消息队列 CKafka 版

产品文档



目录

消息队列 CKafka 版 4
- 产品简介 4
  - 产品概述 4
  - 产品优势 5
  - 应用场景 6
  - 开源对比 8
  - 使用限制 9
- 购买指南 10
  - 产品规格 10
  - 计费概述 11
  - 购买方式 13
- 快速入门 14
  - 步骤1. 创建实例 14
  - 步骤2. 创建 Topic 15
  - 步骤3. 添加 VPC 网络路由 17
  - 步骤4. 收发消息 19
    - 使用 SDK 收发消息（推荐） 19
    - 运行 Kafka 客户端 24
- 操作指南 28
  - 实例管理 28
    - 创建实例 28
    - 批量连续命名或指定模式串命名 29
    - 查看实例 32
    - 销毁_退还实例 34
    - 添加路由策略 36
  - Topic管理 37
  - Consumer Group 42
    - 概述 42
    - 查询Consumer Group 44
    - 设置Offset 45
    - 删除Consumer Group 47
  - 监控告警 48
    - 查看监控信息 48
    - 告警配置建议 51
    - 配置告警 52
  - 权限控制 54
    - 访问管理 CAM 54
    - 配置ACL策略 57
  - 标签管理 59
  - 消息查询 62
- 开发指南 64
  - CKafka 事务管理 64
  - CKafka 版本选择建议 69
  - CKafka 常见参数配置说明 71
  - CKafka 日志分片滚动与保留机制 76
  - CKafka 数据压缩 80
  - 接入低版本自建 Kafka 83
- 最佳实践 88
  - Kafka Connect接入CKafka实践 88
  - Storm接入CKafka 89
  - Spark Streaming接入CKafka 102
  - 生产消费最佳实践 112
  - Logstash接入CKafka 121
  - Filebeats 接入 CKafka 126
  - Schema Registry 接入 CKafka 130
  - Flume接入CKafka最佳实践 137
  - Flink 接入 CKafka 143
- 常见问题 146
  - 功能相关 146
  - 配置相关 148
- 通用参考 150
  - 数据可靠性说明 150
- SDK文档 153
  - Go SDK 153
    - 01 VPC 网络接入 153
  - Node 158
    - 01 VPC 网络接入 158
  - PHP SDK 164
    - 01 VPC网络接入 164
  - Python SDK 170
    - 01 VPC 网络接入 170
  - C++ SDK 174
    - 01 VPC 网络接入 174
  - Java SDK 189
    - VPC网络接入 189
- API文档 194
  - 消息队列（CKafka）（ckafka） 194
    - 版本（2019-08-19） 194
      - API 概览 194
      - 调用方式 197
        - 接口签名v1 197
        - 接口签名v3 204
        - 请求结构 213
        - 返回结果 214
        - 公共参数 217
      - ACL相关接口 219
        - 添加 ACL 策略 219
        - 添加用户 222
        - 删除ACL 224

- 删除用户 227
- 枚举ACL 229
- 查询用户信息 232
- 修改密码 234

- 主题相关接口 236
  - 增加主题分区 236
  - 创建主题 238
  - 创建主题IP白名单 241
  - 删除主题IP白名单 243
  - 获取主题列表 245
  - 获取主题属性 247
  - 获取主题列表详情 249
  - 查询消息 251
  - 根据位点查询消息列表 253
  - 根据时间戳查询消息列表 255
  - 设置主题属性 257
- 其他接口 260
  - 查询用户列表 260
  - 查询消费分组信息 262
  - 枚举消费分组(精简版) 264
  - 查看订单配置 266
- 实例相关接口 269
  - 创建token 269
  - 获取消费分组信息 271
  - 获取消费分组offset 273
  - 获取实例属性 275
  - 获取实例列表 277
  - 获取实例列表详情 279
  - 设置Groups 消费分组offset 282
  - 设置实例属性 285
- 访问控制相关接口 288
  - 授权token 288
- 数据结构 290
- 错误码 331

产品简介
产品概述

什么是消息队列 CKafka

消息队列 CKafka（Cloud Kafka）是基于开源 Apache Kafka 消息队列引擎，提供高吞吐性能、高可扩展性的消息队列服务。消息队列 CKafka 完美兼容 Apache Kafka 1.1、2.4、2.8 版本接口，在性能、扩展性、业务安全保障、运维等方面具有超强优势，让您在享受低成本、超强功能的同时，免除繁琐运维工作。

产品功能

- 收发解耦
  有效解耦生产者、消费者之间的关系。在确保同样的接口约束的前提下，允许独立扩展或修改生产者/消费者间的处理过程。
- 削峰填谷
  消息队列 CKafka 能够抵挡突增的访问压力，不会因为突发的超负荷的请求而完全崩溃，有效提升系统健壮性。
- 顺序读写
  消息队列 CKafka 能够保证一个 Partition 内消息的有序性。和大部分的消息队列一致，消息队列 CKafka 可以保证数据按照顺序进行处理，极大提升磁盘效率。
- 异步通信
  在业务无需立即处理消息的场景下，消息队列 CKafka 提供了消息的异步处理机制，访问量高时仅将消息放入队列中，在访问量降低后再对消息进行处理，缓解系统压力。

产品优势

本文主要介绍消息队列 CKafka 相比于自建开源 Apache Kafka 所具备的优势。

自身优势特性

兼容开源

- 消息队列 CKafka 业务系统基于现有的开源 Apache Kafka 生态的代码，无需改造，即可享受到高性能消息队列 CKafka 服务。

高性能

- 消息队列专业团队对服务性能进一步调优，免除复杂的参数配置，提供更高性能。

高可用性

- 依托技术工程多年监控平台的技术积累，对集群全方位多角度监控。

高可靠性

- 磁盘高可靠，即使服务器坏盘50%也不影响业务。
- 多副本，可靠性高。

数据安全

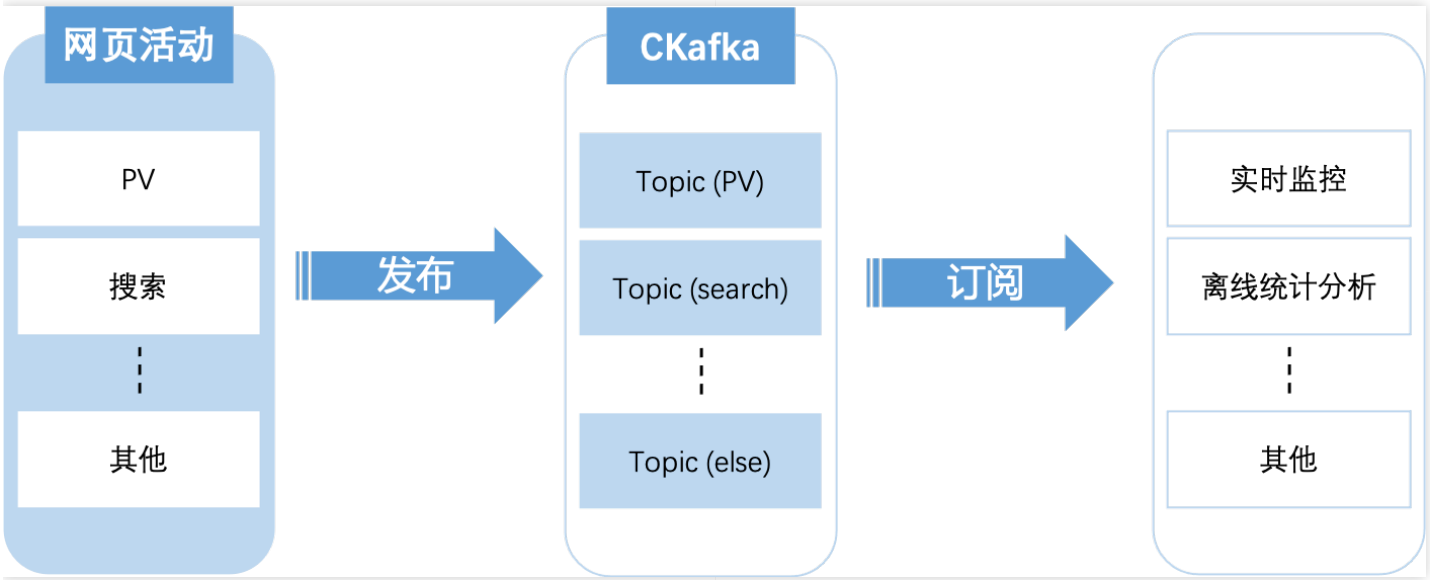
- 消息队列 CKafka 提供鉴权与授权机制、主子账号等功能，提供企业级的安全防护。
- 私有网络（VPC）：支持 VPC 访问，网络环境安全。
- 主子账号：全面支持主子账号、协作者等功能，实现主子账号之间以及企业间跨账号的授权服务。

应用场景

消息队列 CKafka 广泛应用于大数据领域，如网页追踪行为分析、日志聚合、监控、流式数据处理、在线和离线分析等。

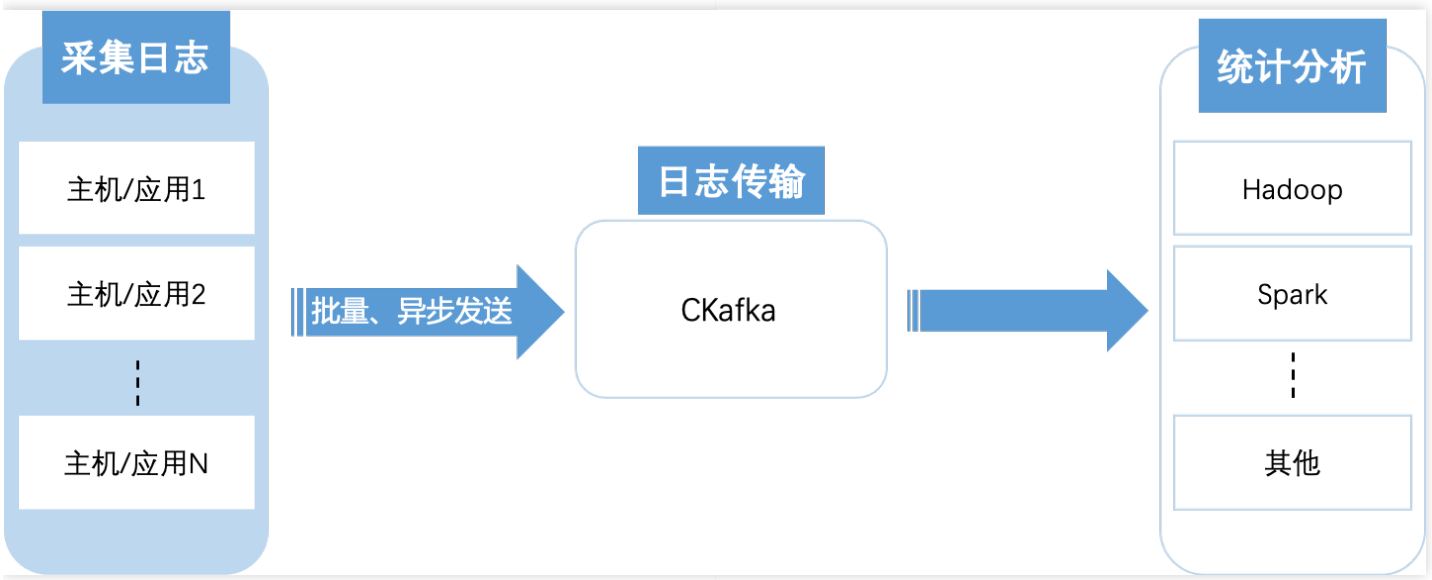
网页追踪

消息队列 CKafka 通过实时处理网站活动（PV、搜索、用户其他活动等），并根据类型发布到 Topic 中，这些信息流可以被用于实时监控或离线统计分析等。
由于每个用户的 page view 中会生成许多活动信息，因此网站活动跟踪需要很高的吞吐量，消息队列 CKafka 可以完美满足高吞吐、离线处理等要求。



日志聚合

消息队列 CKafka 的低延迟处理特性，易于支持多个数据源和分布式的数据处理（消费）。相比于中心化的日志聚合系统，消息队列 CKafka 可以在提供同样性能的条件下，实现更强的持久化保证以及更低的端到端延迟。
消息队列 CKafka 的特性决定它非常适合作为"日志收集中心"；多台主机/应用可以将操作日志"批量""异步"地发送到消息队列 CKafka 集群，而无需保存在本地或者 DB 中；消息队列 CKafka 可以批量提交消息/压缩消息，对于生产者而言，几乎感觉不到性能的开支。此时消费者可以使用 Hadoop 等其他系统化的存储和分析系统对拉取日志进行统计分析。



大数据场景

在一些大数据相关的业务场景中，需要对大量并发数据进行处理和汇总，此时对集群的处理性能和扩展性都有很高的要求。消息队列 CKafka 在实现上的数据分发机制，磁盘存储空间的分配、消息格式的处理、服务器选择以及数据压缩等方面，也决定其适合处理海量的实时消息，并能汇总分布式应用的数据，方便系统运维。
在具体的大数据场景中，消息队列 CKafka 能够很好地支持离线数据、流式数据的处理，并能够方便地进行数据聚合、分析等操作。

开源对比

消息队列 CKafka 版与开源 Apache Kafka 的性能对比详情如下：

功能项	CKafka	Apache Kafka
基础功能	支持控制台调整 Topic 参数配置、变更 Topic 分区数、发送消息、重置消费位点；支持对集群、Topic、消费组等进行可视化管理。	Topic 参数配置、分区数变更等功能需要命令行配置，业务人员难以自定义操作；不支持发送消息，后端手动重置消费进度，繁琐易出错；需要搭配开源管理系统，易用性弱。
隔离性	支持基于一套物理集群创建多个实例，支持按带宽及磁盘限制使用量，支持多接入点配置，支持管控面与数据面隔离，支持主账号间数据逻辑隔离。	不支持。
监控告警	成熟标准化的部署、监控方案，开箱即用，支持多维度监控告警，支持指标排序，支持在控制台查看日志详情，包括异常事件日志，方便排障。	不支持。
高可用	支持跨 AZ 高可用部署，成熟的故障恢复方案。	支持但较为繁琐。
安全合规性	支持Topic 维度 ACL 访问控制；支持控制台配置 SASL 密码认证、SSL认证；支持管控操作权限控制，管控操作对接云审计，可回溯。	通过命令行配置参数，繁琐且容易出错，但不支持操作可追溯。

使用限制

本文列举了消息队列 CKafka 中对一些指标和性能的限制，请您在使用中注意不要超出对应的限制值，避免出现异常。

限制项	说明
Topic 总数量	根据产品规格不同固定上限。
Partition 数量	- 单 Topic 支持的 Partition 数量限制为1500个。 - 实例级别 Partition 数量限制包含了副本数量，副本数量一般为2或者3。 - 不支持缩减partition数量。
Partition 吞吐	- 在 ack = 1的情况下，受 CKafka 的分区架构、业务数据大小、请求频率等因素的影响，CKafka 单分区的吞吐在30MB/s - 60MB/s之间。 - 在 ack = -1(强一致)的情况下，受 CKafka 的分区架构、业务数据大小、请求频率等因素的影响，为了保证请求的耗时的稳定， 建议 CKafka 单分区的吞吐在10MB/s - 20MB/s之间。
耗时	Kafka 是大流量、高吞吐的消息队列，无法保证每条请求的耗时都是低延时。建议超时时间设置如下： 生产端当 ack = 1 的时候，超时时间默认设置为 30s; 生产端在 ack = -1 的时候，超时时间默认设置为 60s; 消费端的超时时间设置为 60s;
Consumer group 数量	实例级别 Consumer group 限制初始为50个。
实例	- 不支持变更实例地域属性。 - 客户端实例最大连接数量为5000，实例连接超过该最大值会导致客户端无法创建新的连接，如评估该最大值在实际业务中不合理可以 [联系技术支持] 申请扩大。
版本	兼容开源1.1、2.4、2.8 版本。
多路由	单个实例最多创建5条路由，其中，每个实例最多创建1条SASL_PLAINTEXT路由。
暴露 Zookeeper	不支持。
暴露底层资源	不支持，避免用户自行操作所带来的风险。
消息大小	不超过8MB，若超过8MB消息会发送失败。
标签	每个云资源允许的最大标签数量为50。

购买指南
产品规格

消息队列 CKafka 版实例提供了标准版、专业版，供用户在不同场景使用。当您在购买 CKafka 集群时，建议综合业务应用场景、产品能力、使用成本等因素，做出产品选型判断。

项目	项目描述	专业版	标准版
版本	实例版本	2.4.1、2.8.1	1.1.1
架构	部署架构	容器部署，专享实例	共享物理节点资源
规格	带宽规格范围	基础型：支持以20MB/s步长调整带宽大，范围20~1200MB/s 高性能型：支持以200MB/s步长调整带宽大，范围1600~2000MB/s 详情参见计费概述	预设了九种带宽规格，范围40~1200MB/s
	扩容	自由度高，可单独扩容带宽、Topic/Partition 上限、磁盘	自由度低，可以单独扩容磁盘

计费概述

消息队列（CKafka）计费概述

消息队列 CKafka 以实例的形式售卖，支持**按量计费（后付费）**付费模式。

- 用户在创建 CKafka 实例的时候，系统会展示购买页，用户可在购买页上调整 CKafka 的规格和购买时长，系统会自动计算购买的价格。在用户确认信息并点击购买后扣费。
- CKafka 的费用三大部分组成：一是实例的费用，取决于用户选择的实例类型；二是磁盘扩容费用，取决于用户在预设的实例规格上追加的磁盘量；三是额外购买的 Partition 数量，取决于用户在预设的实例规格上追加的 Partition 数量。
- 选购实例时，您需要做好对业务峰值带宽、Partition 数和磁盘容量的估算，最终购买的实例的总费用 = (基础套餐包费用 + Partition 包单价 × 额外 Partition 数量 ÷ 100 + 磁盘容量单价 × 磁盘容量 ÷ 100) × 购买时长。

实例规格说明

消息队列 CKafka 根据峰值吞吐性能及磁盘容量不同，分为不同的实例类型。用户可在云平台运营端的计费管理页面为每种规格配置对应的实例价格。

标准版

规格	峰值带宽阶梯（MB/s）	套餐 Partition规格（个）
入门型	40	60
标准型	100	100
进阶型	150	225
容量型	180	225
高阶型L1	300	500
高阶型L2	400	500
高阶型L3	600	550
高阶型L4	900	700
独占型	1200	1500

专业版

规格	峰值带宽阶梯（MB/s）	套餐 Partition规格（个）
基础型	x = 20	400
	40 ≤ x < 60	800
	60 ≤ x < 120	900
	120 ≤ x < 180	1200
	180 ≤ x < 240	1400
	240 ≤ x < 320	1600
	320 ≤ x < 400	1800
	400 ≤ x < 500	2000
	500 ≤ x < 600	2200
	600 ≤ x < 800	2400
	800 ≤ x < 1000	2600
	1000 ≤ x < 1200	2800
高性能型	x = 1600	4000
	x = 1800	4000
	x = 2000	4500

磁盘扩容说明

用户选定了一种规格以后，可以按需对磁盘进行扩容，磁盘扩容的单价可在云平台运营端的计费管理页面进行配置。可以扩容的量为当前规格往上一档的规格的磁盘总量。

比如，用户选择了进阶型实例，磁盘量默认为 1000GB，这时候用户可以把磁盘扩容到最多 2500GB，即进阶型的磁盘量。

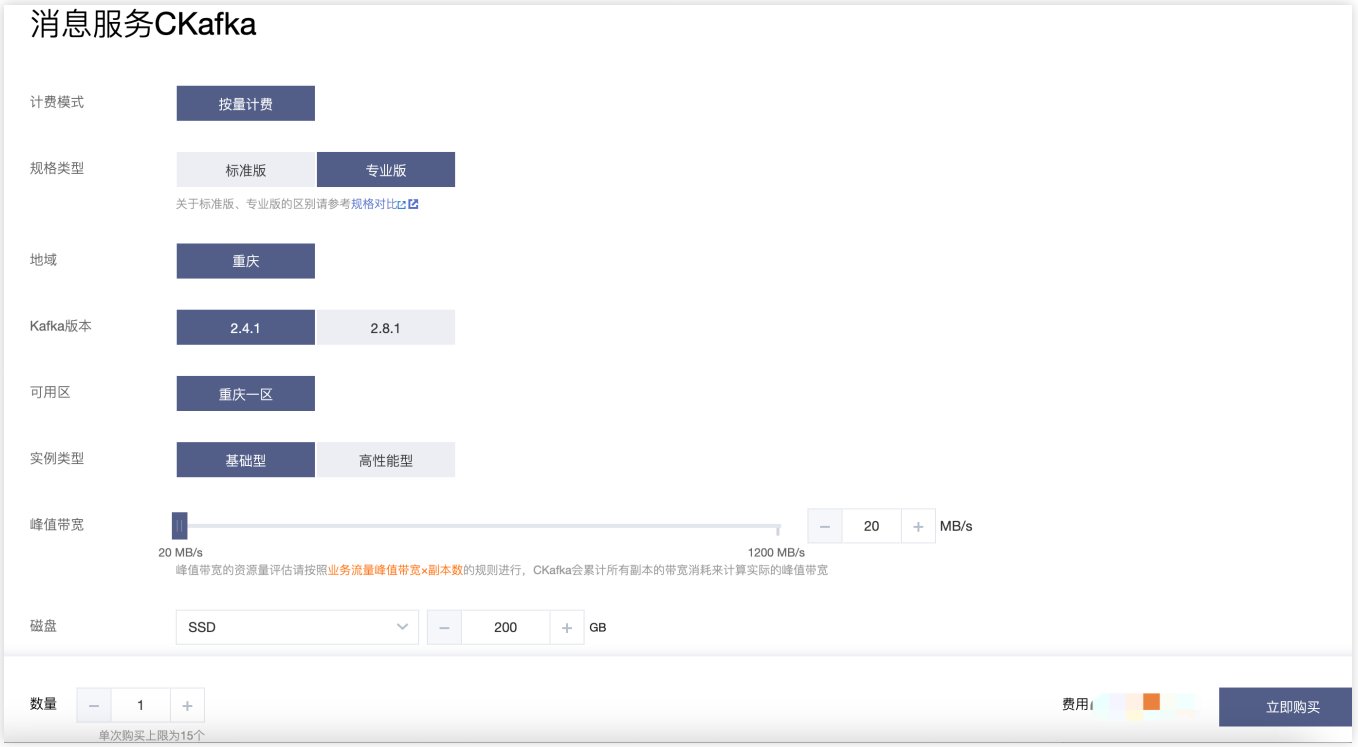
高阶型 L1,高阶型 L2，高阶型 L3 三种规格的磁盘扩容上限皆为 高阶型 L4 的磁盘容量。独占型作为最高规格，无法再额外扩容磁盘。

购买方式

购买方式

您可按照以下步骤购买消息队列 CKafka服务：

- 登录租户端，消息队列 CKafka 控制台。
- 在实例列表页，单击【新建】，进入 CKafka 购买页。
- 在 CKafka 购买页，选择规格、磁盘容量、购买时长等信息。
- 信息填写完成后，单击【立即购买】，根据系统提示步骤完成付款，即购买成功。



快速入门
步骤1. 创建实例

操作场景

该任务指导您通过CKafka控制台创建实例并部署私有网络。

前提条件

- 已购买云服务器。
- 已创建私有网络。

操作步骤

1. 登录 CKafka 控制台。
2. 在左侧导航树点击【实例列表】，单击【新建】进入实例购买页，根据自身业务需求选择购买信息。
    - 计费模式：按量计费
    - 地域：选择和部署客户端的资源相近的地域
    - 产品规格：根据峰值带宽和磁盘容量选择对应的型号。
    - 消息保留：范围在 24 ~ 2160 小时。
      在磁盘容量不足（即磁盘水位达到90%）时，将会提前删除旧的消息，以确保服务可用性。
    - 私有网络：选择提前创建好的与CVM一致的私有网络。若用户需要接入其他私有网络可参考添加路由策略 修改路由接入规则。
    - 实例名称：购买多个实例时，支持创建实例后缀数字自动升序以及指定模式串功能。
3. 单击【购买】，大约等待3～5分钟即可在实例列表页看到创建好的实例。



步骤2. 创建 Topic

操作场景

该任务指导您通过CKafka控制台在已创建好的实例下创建Topic。

操作步骤

1. 登录 CKafka 控制台。
2. 在【实例列表】页，单击刚创建好的实例的“ID/名称”，进入实例详情页。
3. 在实例详情页，单击页面顶部的【Topic 管理】，单击【新建】。
4. 在编辑 Topic 窗口中，选择分区数和副本数等信息。



- 名称：Topic 名称，输入后无法更改，名称只能包含字母、数字、下划线、“-”和“.”。

- 分区数：一个物理上分区的概念，一个 Topic 可以包含一个或者多个 Partition，CKafka 以 Partition 作为分配单位。
- 副本数：Partition 的副本个数，用于保障 Partition 的高可用，为保障数据可靠性，当前不支持创建单副本 Topic，默认开启2副本。
  副本数也算分区个数，例如客户创建了1个 Topic、6个分区、2个副本，那么分区额度一共用了1 * 6 * 2 = 12 个。
- 白名单： 开启白名单后，只有白名单中的 IP 才可访问该 Topic，有效保证数据安全（在新建 Topic 和编辑 Topic 页面均可以开启白名单）。

5. 单击【提交】完成 Topic 创建。

步骤3. 添加 VPC 网络路由

操作场景

实例创建完成后，需要对实例添加一条 VPC 网络路由，该任务指导您通过 CKafka 控制台对已创建好的实例添加 VPC 网络路由。

前提条件

已 创建实例。

操作步骤

1. 在 CKafka 实例列表 页面，单击 步骤1 创建的实例的“ID/名称”。
2. 在实例详情页面，选择接入方式模块中的添加路由策略，新增一条 VPC 网络路由。添加后获得 VPC 网络访问的域名和接口。



接入方式

添加路由策略

接入类型	接入方式	网络	操作
VPC网络	PLAINTEXT	vpc-k [redacted]092	删除

步骤4. 收发消息
使用 SDK 收发消息（推荐）

操作场景

该任务以 Java 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。
其他语言客户端请参考 SDK文档。

前提条件

- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤一：添加 Java 依赖库

在 pom.xml 中添加以下依赖。

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-clients</artifactId>
   <version>0.10.2.2</version>
</dependency>
```

步骤二：准备配置

1. 创建消息队列 CKafka配置文件 kafka.properties。

```
## 配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。
bootstrap.servers=ckafka-xxxxxxxxxxxxxxxx
## 配置Topic，在控制台上topic管理页面复制。
topic=XXX
## 配置Consumer Group，您可以自定义设置
group.id=XXX
```

参数	说明
bootstrap.servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。

参数	说明
	
topic	topic名称，您可以在控制台上【topic管理】页面复制。 
group.id	您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。

2. 创建配置文件加载程序CKafkaConfigurer.java。

```
public class CKafkaConfigurer {

    private static Properties properties;

    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //获取配置文件kafka.properties的内容。
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties"));
        } catch (Exception e) {
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

步骤三：发送消息

1. 编写生产消息程序 CKafkaProducerDemo.java。

```
public class CKafkaProducerDemo {
```

```
    public static void main(String args[]) {
        //加载kafka.properties。
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties properties = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));

        //消息队列Kafka版消息的序列化方式, 此处demo 使用的是StringSerializer。
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        //请求的最长等待时间。
        properties.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //设置客户端内部重试次数。
        properties.put(ProducerConfig.RETRIES_CONFIG, 5);
        //设置客户端内部重试间隔。
        properties.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        //构造Producer对象。
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

        //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 ckafka msg value"; //消息的内容。

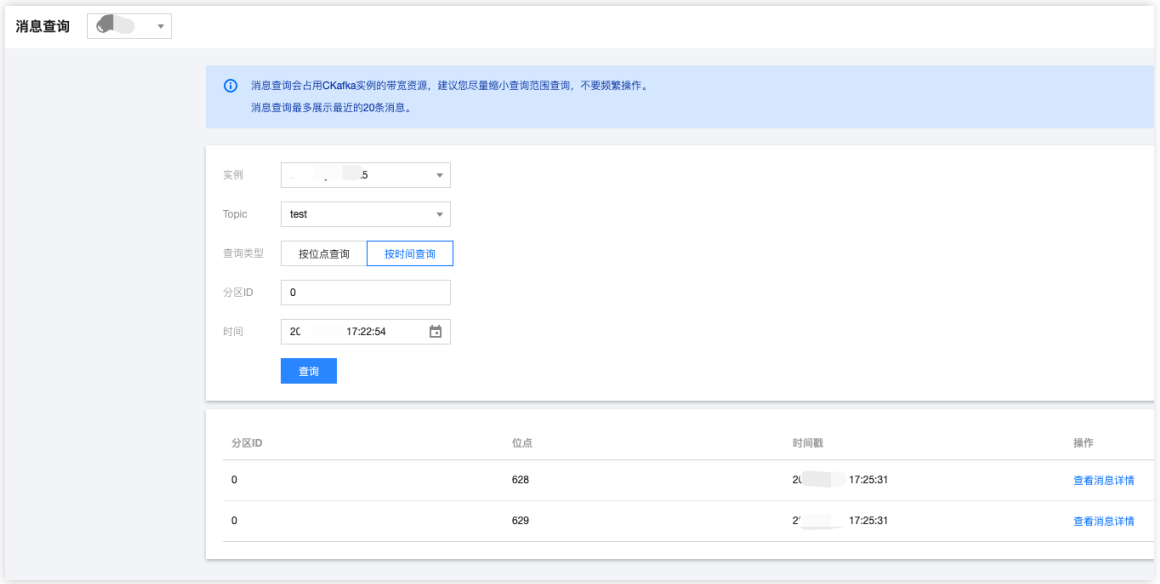
        try {
            //批量获取Future对象可以加快速度, 但注意, 批量不要太大。
            List<Future<RecordMetadata>> futureList = new ArrayList<>(128);
            for (int i = 0; i < 10; i++) {
                //发送消息，并获得一个Future对象。
                ProducerRecord<String, String> kafkaMsg = new ProducerRecord<>(topic,
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMsg);
                futureList.add(metadataFuture);

            }
            producer.flush();
            for (Future<RecordMetadata> future : futureList) {
                //同步获得Future对象的结果。
                RecordMetadata recordMetadata = future.get();
                System.out.println("produce send ok: " + recordMetadata.toString());
            }
        } 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");
        }
    }
}
```

2. 编译并运行 CKafkaProducerDemo.java 发送消息。
3. 运行结果。

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. 在 CKafka 控制台的【topic管理】页面，选择对应的 topic ，点击【更多】>【消息查询】，查看刚刚发送的消息。



步骤四：消费消息

1. 创建Consumer 订阅消息程序 CKafkaConsumerDemo.java。

```
public class CKafkaConsumerDemo {

    public static void main(String args[]) {
        //加载kafka.properties。
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        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"));
        //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Consumer 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");
        //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。
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(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<>();
        //如果需要订阅多个Topic，则在这里添加进去即可。
        //每个Topic需要先在控制台进行创建。
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic : topics) {
```

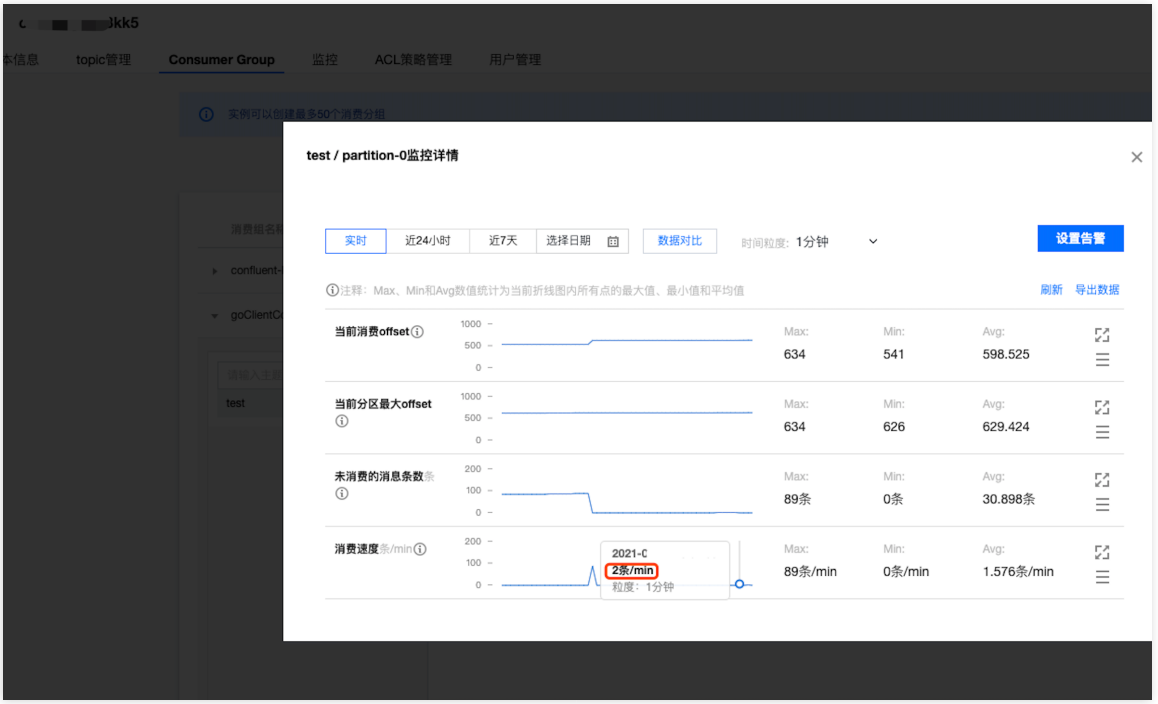
```
            subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);

        //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) {
                System.out.println("consumer error!");
            }
        }
    }
}
```

2. 编译并运行CKafkaConsumerDemo.java 消费消息。
3. 运行结果。

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. 在 CKafka 控制台的【Consumer Group】页面，选择对应的消费组名称，在主题名称输入 topic 名称，点击【查询详情】，查看消费详情。



运行 Kafka 客户端

操作场景

该任务指导您在购买 CKafka 服务后，使用 Kafka API。在TCloudFinanceZone服务器上搭建 CKafka 环境后，本地下载并解压 Kafka 工具包，并对 Kafka API 进行简单测试。

操作步骤

步骤1：安装 JDK 环境

1. 检查 Java 安装。

打开终端，执行如下命令：

```
java -version
```

如果输出 Java 版本号，说明 Java 安装成功；如果没有安装 Java，请 下载安装 Java 软件开发套件（JDK）。

2. 设置 Java 环境。

设置 JAVA_HOME 环境变量，并指向您机器上的 Java 安装目录。
以 Java JDK 1.8.0_20 版本为例，操作系统的输出如下：

操作系统	输出
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdkjdk1.8.0_20
Linux	export JAVA_HOME=/usr/local/java-current
Mac OSX	export JAVA_HOME=/Library/Java/Home

将 Java 编译器地址添加到系统路径中：

操作系统	输出
Windows	将字符串";C:\Program Files\Java\jdk1.8.0_20\bin"添加到系统变量"Path"的末尾
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac OSX	not required

使用上面提到的 java -version 命令验证 Java 安装。

步骤2：下载 Kafka 工具包

下载并解压 Kafka 安装包。（Kafka 安装包官网下载地址>>）

步骤3：Kafka API 测试

通过 CLI 命令生产和消费消息，去到 ./bin 目录下。

1. 打开终端启动消费者。

   bash kafka-console-consumer.sh --bootstrap-server XXXX:port --topic XXXX --consumer.config ../config/consumer.properties

   - 将 XXXX:port 替换成VPC网络访问的域名与端口，在控制台实例详情页面的【接入方式】模块获取。



   - topic：将 XXXX替换成topic名称，在控制台【topic管理】页面获取。

2. 另外开一个终端窗口启动生产者。

   bash kafka-console-producer.sh --broker-list XXXX:port --topic XXXX --producer.config ../config/producer.properties

   - 将 XXXX:port 替换成VPC网络访问的域名与端口，在控制台实例详情页面的【接入方式】模块获取。



   - topic：将 XXXX替换成topic名称，在控制台【topic管理】页面获取。

输入消息内容之后按回车，即可看到消费端也几乎同时收到消息。

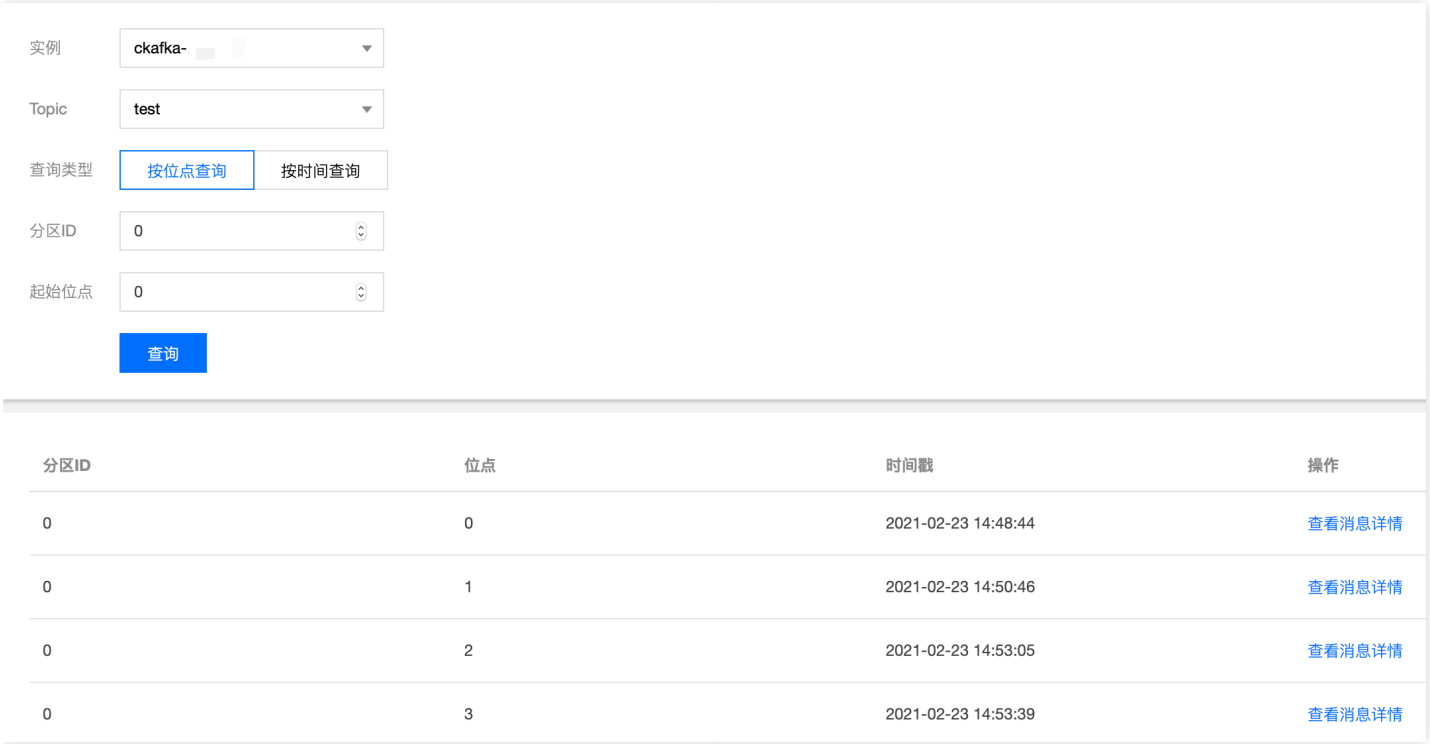
生产消息：

```
[ bin % bash kafka-console-producer.sh --broker-list ckafka]
.ap-guangzhou.ckafka.tencentcloudmq.com:6014 --topic test --producer.co
nfig ../config/producer.properties
>hello world
>this is a message
>this is another message
>
```

消费消息：

```
[ bin % bash kafka-console-consumer.sh --bootstrap-server c]
kafka- .ap-guangzhou.ckafka.tencentcloudmq.com:6014 --topic test --consum
er.config ../config/consumer.properties
hello world
this is a message
this is another message
```

3. 在 CKafka 控制台消息查询页面，查询刚刚发送的消息内容。



消息详情如下：


消息详情
当前查询的消息已经被强制转换为String类型，如出现乱码，请分析您消息的序列化格式以及编码格式
Key
暂无数据
Value
hello world
确定

操作指南
实例管理
创建实例

操作场景

该任务指导您通过 CKafka 控制台创建实例和 Topic，快速了解 CKafka 控制台操作流程。

操作步骤

1. 登录 CKafka 控制台。
2. 在左侧导航栏单击【实例列表】，单击【新建】进入实例购买页。
3. 在实例购买页，根据自身业务需求选择购买信息。
    - 计费模式：按量计费
    - 地域：选择和部署客户端的资源相近的地域
    - 产品规格：根据峰值带宽和磁盘容量选择对应的型号。
    - 消息保留：范围在 24 ~ 2160 小时。
    - 在磁盘容量不足（即磁盘水位达到90%）时，将会提前删除旧的消息，以确保服务可用性。
    - 私有网络：选择提前创建好的与CVM一致的私有网络。若用户需要接入其他私有网络可参考添加路由策略 修改路由接入规则。
    - 实例名称：购买多个实例时，支持创建实例后缀数字自动升序以及指定模式串功能。
4. 单击【立即购买】，完成实例创建。



批量连续命名或指定模式串命名

操作场景

在创建多个CKafka实例过程中，如果您希望实例名称具有一定的规则性，我们提供批量创建实例后缀数字自动升序功能以及指定模式串功能，您可以通过购买页和云 API 两种方式实现。

- **后续数字自动升序：**如果购买多个的情况下设置实例名称，默认在设置名称后边加上数字的形式来标志多个CKafka实例（例如ckafka1、ckafka2、ckafka3 ……）。
- 指定模式串：
  - 指定单个模式串：适用于需要创建 n 个实例并指定实例名称带有序号且序号从 x 开始递增时（例如ckafka3、ckafka4、ckafka5……）。
  - 指定多个模式串：适用于希望创建 n 个有多个前缀且每个前缀均指定序号的实例名称时（例如ckafka3-big10-test、ckafka4-big11-test、ckafka5-big12-test……）。

操作步骤

后缀数字自动升序

可将批量购买的实例设置为前缀相同，仅序号递增的实例名称。

> 注意：
>
> 创建成功的实例默认序号从1开始递增，且不能指定开始的序号。

以下操作以您购买了3个CKafka实例，并希望生成的实例名称为“ckafka+序号”（即实例名称为 ckafka1、ckafka2 和 ckafka3）为例。
购买页操作

1. 参考创建实例购买3个实例，并在购买页以“前缀+序号”的命名规则填写实例名称，即将实例名称填写为 ckafka 。如下图所示：



2. 根据页面提示逐步操作，并完成支付。

API 操作
在云 API [ModifyInstanceAttributes] 中，设置相关字段：
实例名称：将 InstanceName 字段指定为 ckafka 。

指定模式串

可将批量购买的实例设置为复杂且指定序号的实例名称。实例名称支持指定单个或者多个模式串，在设置实例名称时，请根据实际需求进行设置。指定模式串的命名：{R:x}，x 表示生成实例名称的初始序号，只支持正整数，不支持负数和浮点数。

指定单个模式串

以下操作以您需要创建3个实例，且指定实例的序号从3开始递增为例。
购买页操作

1. 参考 创建实例 购买实例，并在购买页以**“前缀+指定模式串{R:x}”**的命名规则填写实例名称，即将实例名称填写为 ckafka{R:3} 。如下图所示：



2. 根据页面提示逐步操作，并完成支付。

API 操作
在云 API [ModifyInstanceAttributes] 中，设置相关字段：
实例名称：将 InstanceName 字段指定为 ckafka{R:3} 。

指定多个模式串

以下操作以您需要创建3个实例，并希望生成实例名称含有 ckafka、 big 和 test 前缀，且 ckafka 和 big 前缀后面带序号，序号分别从13和2开始递增（即实例名称为 ckafka13-big2-test、ckafka14-big3-test、ckafka15-big4-test）为例。
购买页操作

1. 参考 [创建实例] 购买3台实例，并购买页以**“前缀+指定模式串{R:x}-前缀+指定模式串{R:x}-前缀”**的命名规则填写实例名称，即将实例名称填写为 ckafka{R:13}-big{R:2}-test 。如下图所示：



2. 根据页面提示逐步操作，并完成支付。

API 操作

在云 API [ModifyInstanceAttributes] 中，设置相关字段：

实例名称：将 InstanceName 字段指定为 ckafka{R:13}-big{R:2}-test 。

验证功能

当您通过后缀数字自动升序或指定模式串实现批量创建实例后，可通过以下操作验证设置实例名称。

登录 CKafka 控制台查看新创建的实例，即可发现批量购买的实例会根据您设置的规则进行命名。如下图所示：



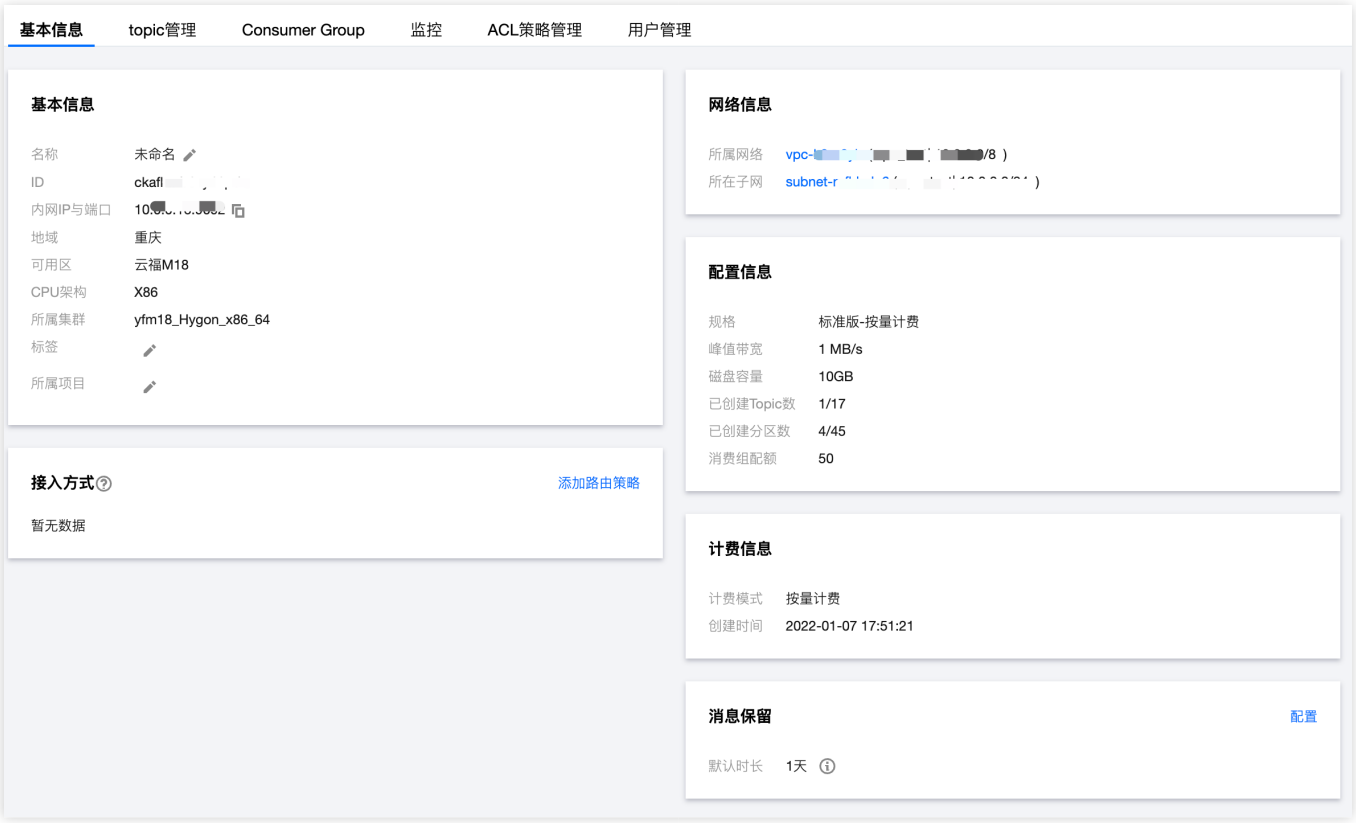
查看实例

该任务指导您通过CKafka控制台查看实例的配置信息和健康状态。
CKafka 对每个实例均设置有巡检程序，巡检程序会检查该实例的连接数、磁盘使用百分比、生产峰值带宽、消费峰值带宽，当这些指标超过一定的阈值后会产生不同的健康状态。说明如下：

指标	阈值	状态描述
连接数（默认最大值5000）	N ≤ 80%	 健康
	80% < N ≤ 95%	告警
	N > 95%	异常
 磁盘使用百分比 	N ≤ 80%	健康
	80% < N ≤ 95%	告警
	N > 95%	异常
 生产峰值带宽（不含副本带宽） 	N ≤ 80%	健康
	80% < N ≤ 100%	告警
	N > 100%	异常
 消费峰值带宽 	N ≤ 80%	健康
	80% < N ≤ 100%	告警
	N > 100%	异常

> 说明：
>
> 连接数默认最大值是5000，阈值判断是基于最大值的百分比进行判断。实例连接超过该最大值会导致客户端无法创建新的连接，如评估该最大值在实际业务中不合理可以联系技术支持申请扩大。

1. 登录 CKafka 控制台。
2. 在左侧导航树点击【实例列表】，单击目标实例的"ID/名称"，在【基本信息】页，可查看实例的健康状态、配置信息、接入方式、消息保留等信息。



> 说明：
>
> 【配置信息】中内网 IP 与端口（例如 10.6.206.110:9092 ），表示用于获取后端服务的通讯地址，真实访问地址中端口可能存在多个，如果您的服务器配置了访问限制，请在服务器上放通9092 - 9192端口（broker 可能会自动扩容，扩容后需要放通的端口会增加，需要预留充足的数量）。

销毁_退还实例

操作场景

用户不再需要消息队列 CKafka 实例时，可以销毁并释放该实例。
消息队列 CKafka 实例的生命周期是指实例从启动到释放所经历的状态。通过对实例从启动到销毁期间的合理的管理，可确保运行于实例上的应用程序能高效经济地提供服务。实例有以下状态：

状态名	状态属性	状态描述
创建中	中间状态	实例创建后，进入运行中之前的状态。
正在运行	稳定状态	实例正常运行状态，表明您的磁盘、流量、连接数都处于规划范围内。
删除中	中间状态	实例受控制台或通过 API 执行删除操作。
已隔离	中间状态	实例受控制台或通过 API 执行销毁操作或您的实例已经欠费了，进入7天隔离状态。
创建失败	中间状态	实例受控制台或通过 API 执行购买操作扣费成功但分配实例失败，如遇到这种情况请联系运维人员处理。
删除失败	稳定状态	实例被手动删除或者在到期14天后（包括第14天）未进行续费，CKafka 执行资源释放时失败。

操作步骤

> 注意：
>
> 删除后所有数据将被清除且不可恢复，请提前备份数据。

1. 登录 CKafka 控制台。
2. 在实例列表页的操作栏，选择操作栏的 销毁/退还。
3. 在销毁实例的弹窗中，单击 下一步，销毁实例，即可销毁该实例。


销毁实例
1 销毁明细
2 操作须知
您已选 1 个实例，查看详情
No.
实例ID
实例名
操作
1
ckafka-naeoqmwa
test241111new
可删除
销毁明细
实例ID
带宽
磁盘
已创建topic数
已创建分区数
ckafka-naeoqmwa
20 MB/s
200 GB
2/200
0/400
下一步
关闭


销毁实例
销毁明细
2 操作须知
1. 彻底销毁实例会立即将该实例从账号实例列表销毁，此后将不再可以通过续费或其他方式恢复实例。
2. 销毁后所有数据将被清除且不可恢复，请提前备份数据。
上一步
确定销毁

添加路由策略

操作场景

该任务指导您在使用消息队列 CKafka 时，通过控制台配置路由接入规则，增强对内网传输中的用户网络访问控制。

> 说明：
>
> 您购买实例时选择私有网络并选择了相应的 VPC 环境（例如 VPC A），表示仅能所选择的 VPC A 访问您的消息队列 CKafka 服务（生产数据、消费数据等）；若后续使用过程中发现其他 VPC 环境（例如 VPC B）有需求访问 VPC A 内的消息队列 CKafka 服务，则可以通过配置接入方式，选择 VPC 网络的路由策略。

操作步骤

1. 登录 CKafka 控制台。
2. 在左侧导航栏单击【实例列表】，单击目标实例的“ID/名称”，进入基本信息页。
3. 在实例基本信息页面，单击接入方式模块中的【添加路由策略】。
4. 在弹窗中，选择路由类型和接入方式。



> 说明：
>
> 控制台上提供的 VPC 网络访问地址（例如 172.16.0.12:9092 ），用于获取后端服务的通讯地址，真实访问地址中端口可能存在多个，请在您的服务器上放通9092后的所有端口，以便服务可以正常访问。

5. 单击【提交】，完成路由策略添加。

Topic管理

操作场景

Topic（主题）是某一种分类的名字，消息在 Topic 中可以被存储和发布。CKafka 对外使用 Topic 的概念，生产者往 Topic 中写消息，消费者从 Topic 中读消息。为了做到水平扩展，一个 Topic 实际是由多个 Partition（分区）组成，遇到瓶颈时，可以通过增加 Partition 的数量进行横向扩容。
该任务指导您通过 CKafka 控制台，在已有实例下 Topic 进行管理。

操作步骤

通过控制台手动创建 Topic

1. 登录 CKafka 控制台。
2. 在实例列表页，单击目标实例的"ID/名称"，进入实例详情页。
3. 在实例详情页，单击页面顶部的 Topic 管理，单击新建。
4. 在新建 Topic 窗口中，选择分区数和副本数等信息。



- 名称：Topic 名称，创建后无法更改，名称只能包含字母、数字、下划线、“-”和“.”。
- 分区数：一个物理上分区的概念，一个 Topic 可以包含一个或者多个 partition，CKafka 以 partition 作为分配单位。
- 副本数：partition 的副本个数，用于保障 partition 的高可用，为保障数据可靠性，默认开启2副本。副本数也算分区个数，例如客户创建了1个 Topic、6个分区、2个副本，那么分区额度一共用了1 * 6 * 2 = 12个。
- 白名单：开启白名单后，只有白名单中的 ip 才可访问该 Topic。
- retention.ms：Topic维度的消息保留时间，范围1 分钟到90 天。

5. 单击【提交】完成 Topic 创建。



通过客户端自动创建 Topic

“自动创建 Topic”表示客户端向一个 Topic 生产或消费消息时，会自动检测该 Topic 是否存在，若 Topic 不存在，且 Topic 名称符合命名规则，系统会自动创建此 Topic。

开启自动创建 Topic

1. 登录 CKafka 控制台。
2. 在实例列表页，单击目标实例的“ID/名称”，进入实例详情页。
3. 在自动创建 Topic 模块，点击右上角的配置按钮，开启自动创建 Topic。
4. 设置好 Topic 分区数和副本数后，单击提交，完成配置。
   该实例下的自动创建的 Topic 的配置都会继承您所配置的分区数和副本数的值。
   - 分区数：一个物理上分区的概念，一个 Topic 可以包含一个或者多个 partition，CKafka 以 partition 作为分配单位。
   - 副本数：Partition 的副本个数，用于保障 Partition 的高可用。为保障数据可靠性，默认开启2副本。副本数也算分区个数，例如客户创建了1个 Topic、6个分区、2个副本，那么分区额度一共用了1 × 6 × 2 = 12个。
5. 单击【提交】完成 Topic 创建。



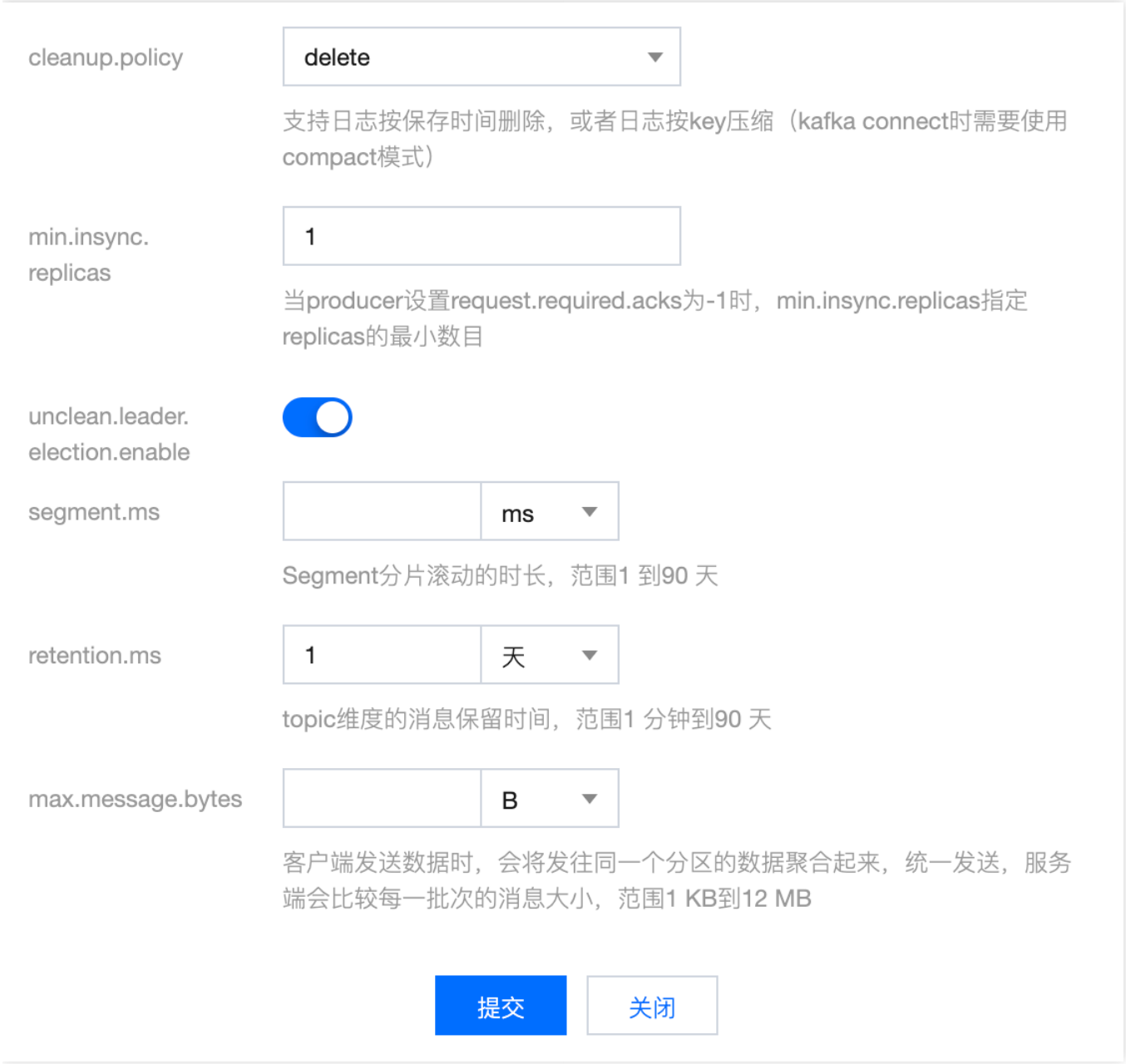
删除 Topic

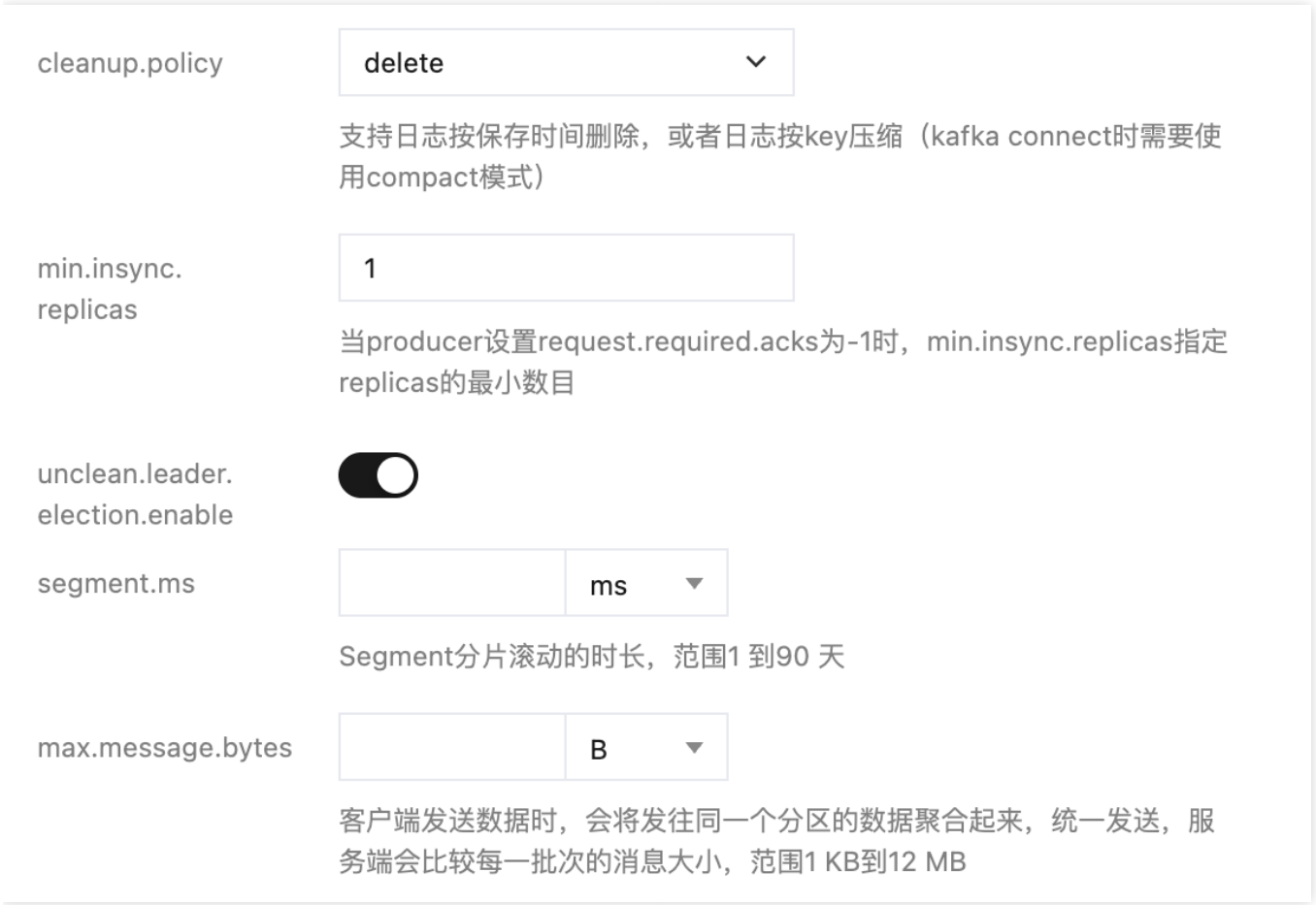
> 注意：
>
> - 删除 Topic 的同时，存储在此 Topic 中的消息也将被删除，请谨慎操作。
> - Topic 删除是异步操作，配置删除成功后，ZooKeeper 配置将会在1分钟后生效。若此期间创建同名 Topic，系统会提示错误码 [4000]10011，届时请您稍后重试。

1. 在实例列表页，单击目标实例的“ID/名称”，进入实例详情页。
2. 在实例详情页，单击 topic管理标签页，在操作栏单击删除。
3. 在弹出窗口单击确认，Topic 将被删除。

配置 Topic 高级参数

1. 在实例列表页，单击目标实例的“ID/名称”，进入实例详情页。
2. 在实例详情页，单击 topic管理标签页。
3. 单击操作列的编辑 > 展示高级配置，设置如下参数：





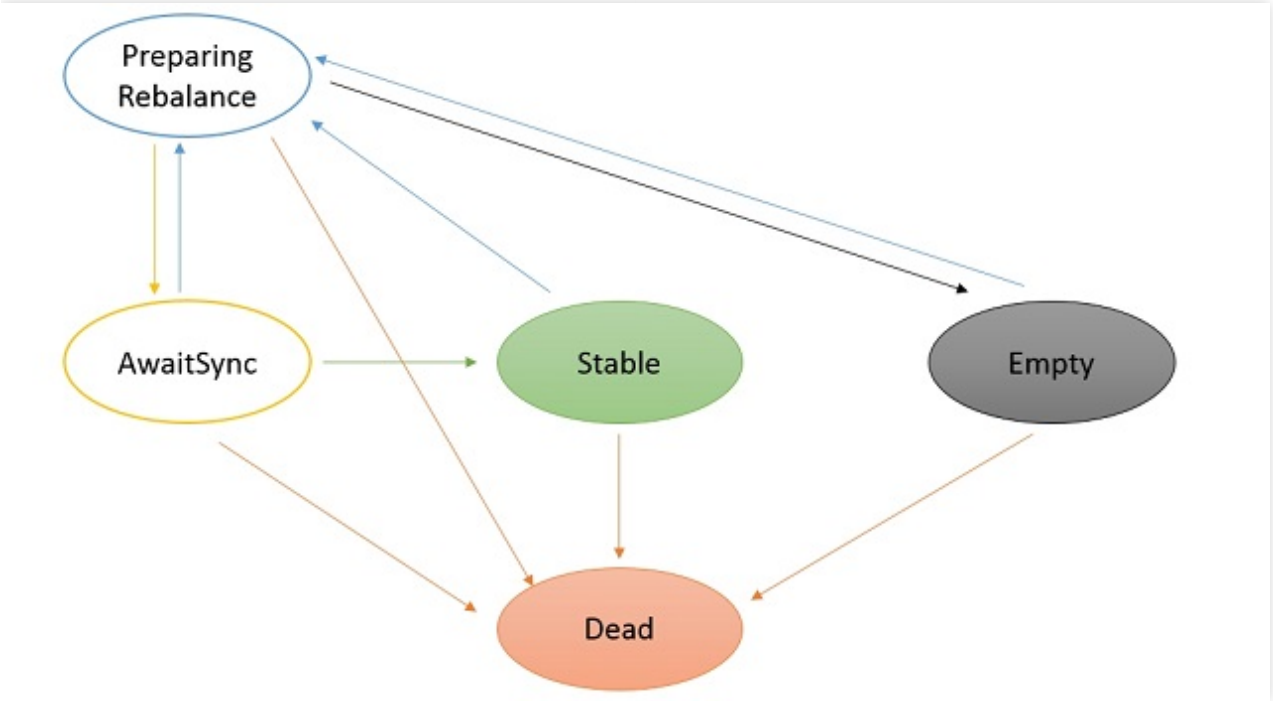
参数说明如下：

参数名	默认值	参数范围	说明
cleanup.policy	delete	delete/compact	支持日志按保存时间删除，或者日志按 key 压缩（Kafka Connect 时需要使用 compact 模式）。
min.insync.replicas	1	-	当 producer 设置 request.required.acks 为1时，min.insync.replicas 指定 replicas 的最小数目。
unclean.leader.election.enable	true	true/false	指定是否能够设置不在 ISR 中 replicas 作为 leader。
segment.ms	-	1day - 90days	Segment 分片滚动的时长，单位为 ms，最小值为 86400000ms。
max.message.bytes	-	0B - 12MB	Topic 维度的最大消息大小。不填写则默认实例维度消息大小为1MB。

Consumer Group 概述

Consumer Group 状态说明

消费者组列表页中 Consumer Group 的状态主要有 Dead、Empty、PreparingRebalance、AwaitingSync、Stable 几种，其中最常见的是 Empty、Stable 和 Dead 三种状态。Consumer Group 中的状态机转换如下图所示：



- Dead：消费者组内无成员并且 Metadata 已经被移除。
- Empty：消费分组内当前没有任何成员。如果组内所有 offset 都已过期，则会变为 Dead 状态。一般新创建的 Group 默认为 Empty 状态。
  开源 Kafka 0.10.x 版本规定，当消费分组内没有任何成员且状态持续超过7天，此消费分组将会被自动删除。
- Stable：消费分组中各个消费者已经加入，处于稳定状态。

Rebalance 状态详解

Rebalance 发生原因

根据 Consumer Group 的状态机可知，当 Consumer Group 为 Empty、AwaitSync 或 Stable 状态时，Group 可能会进行 Rebalance。以下情况可能会发生 Rebalance：

- 一个消费者订阅了 Topic。
- 消费者被关闭。
- 某个 Consumer 被 Group Coordinator（协调器）认为是 Dead 状态时。
  如果某个Consumer 在 session.timeout.ms 时间内没有给 Group Coordinator 发心跳，则该 Consumer 将被认为是 Dead 状态，并且发起 Rebalance。详请参见 [CKafka 常用参数配置指南] 。
- 分区数增加。
- 订阅了不存在的 Topic。
  如果您订阅了一个还未创建的 Topic，那么当这个 Topic 创建后会发生 Rebalance；同理，如果一个已经被订阅的 Topic 被删除，也会发生 Rebalance。
- 应用崩溃。

Rebalance 过程分析

以0.10版本Kafka 的机制为例，Rebalance 过程分析如下：

1. 任何一个 Consumer 想要加入到一个 Consumer Group 中时，会发送一个 JoinGroup 的请求给 Group Coordinator。第一个加入 Group 的 Consumer 会变成 Group Leader。
2. Leader 会从 Group Coordinator 处收到这个 Group 中所有 Consumer 列表，并且负责给 Group 中的 Consumer 分配 partition。分区的分配可以通过 PartitionAssignor 接口来实现。
3. 分配完成后，Leader 会把分配结果发给 Group Coordinator，Coordinator 会把结果发送给所有的 Consumer。
   因此每个 Consumer 只能查看到自己被分配的 partition，Leader 是唯一能够拿到Consumer Group 中的 Consumer 以及其分区情况的节点的 Consumer。
   上述过程会在每次 Rebalance 发生时执行一次。

查询Consumer Group

操作场景

该任务指导您在CKafka控制台查看实例下的消费者组信息。

操作步骤

查询消费者组

1. 登录CKafka 控制台。
2. 在左侧导航栏点击【实例列表】，单击目标实例的"ID/名称"，进入实例详情页。
3. 在实例详情页，点击【Consumer Group】标签页，查看到当前 CKafka 实例的消费组信息。



- 在 Consumer Group 列表页，单击操作列的【查看消费者详情】，可以查看该消费组中的消费者信息，具体消费者和订阅 topic 的对应关系。
- 在 Consumer Group 列表页，单击消费者名称列左侧的小三角，可以展示出该消费者组订阅的主题信息，包含主题名称、分区数目、提交的 offset位置，最大的 offset 位置以及未消费消息条数等。单击操作列的【查看详情】可以看到分区级别的 offset 消费情况。

> 说明：
>
> 由于 offset 信息是在消费端维护的，因此 offset 的位置和消费者提交 offset 的方式有关，是异步展示的，并不一定代表实时的消费情况。

设置Offset

操作场景

在离线数据处理等场景下，有时需要对 offset 进行重置，用于消费前一时间段的消息。

操作步骤

1. 登录CKafka 控制台 。
2. 在左侧导航栏点击实例列表，单击目标实例的“ID/名称”，进入实例详情页。
3. 在实例详情页，点击 Consumer Group 标签页，单击操作列的 offset 设置。
4. 在 offset 设置窗口，只支持以Topic 为维度进行设置，单击下一步。
5. 选择需要重置 offset 的 Topic 信息（不选则默认全部 Topic 的 offset 均重置），单击下一步。
6. 对 offset 进行指定。



> 注意：
>
> - offset 设置范围要在最小 offset 和最大 offset 之间。在配置时，如果小于最小 offset 会从最小 offset 进行消费，如果大于最大 offset 会从最大 offset 进行消费。
> - 重置消费分组时，需保证没有消费者在消费分组内才能进行重置，否则不能进行重置。

删除Consumer Group

操作场景

部分场景下，Consumer Group 会很长一段时间不消费后重新消费，可将消费者组删除，其中的消费者重新建立连接时，会重置 Offset，从头开始消费。

> 说明：
>
> Broker 版本不低于1.1.1，且Consumer Group的状态为Empty时，消费组才能被删除。

操作步骤

1. 登录CKafka 控制台。
2. 在左侧导航栏单击 实例列表，单击目标实例的“ID/名称”，进入实例详情页。
3. 在实例详情页，单击目标 Consumer Group 操作栏的删除，可直接删除 Consumer Group。



监控告警
查看监控信息

操作场景

消息队列CKafka支持监控您账户下创建的资源，包括实例、Topic、Consumer Group等，帮助您实时掌握资源状态，针对可能存在的问题及时处理，保障其稳定运行。
本文为您介绍通过 CKafka 控制台查看监控指标的操作方法和监控指标的含义。

监控指标含义及说明

以下是CKafka监控指标相关说明。其中备注推荐字样的指标是根据历史用户反馈统计建议对其配置监控和告警策略，请您根据实际情况合理配置监控和告警策略。

实例监控

监控指标	说明
实例磁盘占用量 (MB)	实例磁盘占用量（包含副本），按照所选择的时间粒度取最新值。
磁盘使用百分比 (%)（推荐）	当前磁盘占用与实例规格磁盘总容量的百分比。
实例连接数 (Count)（推荐）	客户端和服务器的连接数。
实例最大生产流量 (MB)（推荐）	实例单个副本的生产消息峰值带宽，不包含副本生产的带宽（计算实例生产带宽使用百分比时的参考依据）。
实例最大消费流量 (MB)（推荐）	实例消费消息峰值带宽，消费时无副本的概念。（计算实例消费带宽使用百分比时的参考依据）。
实例消费带宽百分比(%)	实例消费带宽百分比(占用配额百分比)。
实例生产带宽百分比(%)	实例生产带宽百分比(占用配额百分比)。
实例消费限流次数 (Count)	实例消费限流次数。
实例生产限流次数 (Count)	实例生产限流次数。
实例落盘的消息总条数 (Count)	实例落盘的消息总条数（不包含副本），按照所选择的时间粒度取最新值。
实例消费流量 (MB)	实例消费流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
实例生产消息条数 (Count)	实例生产消息条数，按照所选择的时间粒度统计求和。
实例生产流量 (MB)	实例生产流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
实例消费消息条数 (Count)	实例消费消息条数，按照所选择的时间粒度统计求和。

Topic 监控

监控指标	说明
最大生产流量（MB/s）	Topic 最大生产流量(不含副本流量)。
最大消费流量（MB/s）	Topic 最大消费流量。
消费消息条数（条）	Topic 的实际消费消息条数，按照所选择的时间粒度统计求和。
消费流量（MB）	Topic 的实际消费流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
占用磁盘的消息总量（MB）（推荐）	Topic 实际占用磁盘的消息总量（不包含副本），按照所选择的时间粒度取最新值。
生产消息条数（条）	Topic 的实际生产消息条数，按照所选择的时间粒度统计求和。
生产流量（MB）	Topic 的实际生产流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
落盘的消息总条数（条）	Topic 的实际的落盘的消息总条数（不包含副本），按照所选择的时间粒度取最新值。

Consumer Group-Topic 监控：

监控指标	说明
主题最大 offset	当前主题中所有分区的最大 offset。
主题消费 offset（区分 offset 最大值）	当前主题所有分区中消费组消费的最大 offset。
主题未消费消息数（条）	当前主题所有分区消费组未消费消息总和。
主题消费速度（条/分钟）	当前主题所有分区消费组消费速率总和。

Consumer Group-Partition 监控：

监控指标	说明
分区消费速度（条/分钟）	消费分组在该分区的消费速率（条/分钟）。
当前消费 offset	消费分组该分区当前消费 offset。
当前分区最大 offset	当前 分区 最大 offset。
未消费的消息条数（条）（推荐）	消费分组在该分区下未消费消息数。

操作步骤

1. 登录 CKafka 控制台。
2. 在实例列表中，单击需要查看的“实例 ID/名称”，进入实例详情页。

3. 在实例详情页顶部，单击【监控】，选择要查看的实例资源标签，设置好时间范围，可以查看实例监控数据。



- 单击 ，可查看监控指标同环比。

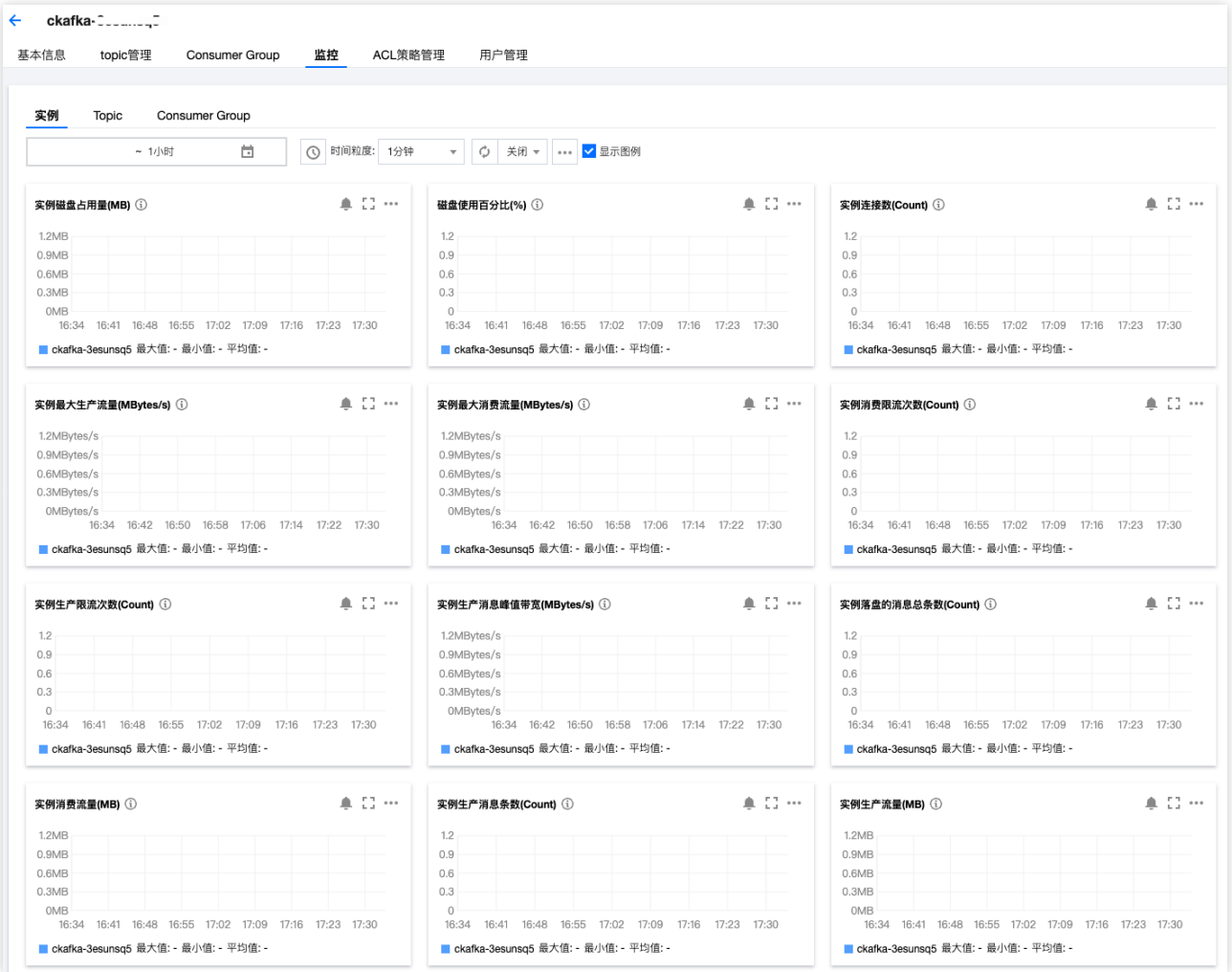


- 单击 ，可刷新获取最新的监控数据。

- 单击 ，可将图表复制到Dashboard。



- 选中 ，可在图表上显示图例信息。



告警配置建议

消息队列CKafka 不仅为运行中的 CKafka 集群提供了多项监控指标，用于监测集群的运行情况，还提供了一些关键指标的配置告警功能，帮助您及时发现集群问题并进行处理。具体使用方法可参考 [查看监控] 和 [配置告警] 。

本文为您介绍在使用 CKafka 过程中需要重点关注的一些指标及其告警建议配置：

监控告警指标配置补充建议：

指标	告警建议配置	详细说明
CPU使用率(%)	统计周期1分钟，>90%，持续5个周期，每30分钟告警一次	CPU 使用率表示集群各节点 CPU 使用率的值。该值过高会导致集群节点处理能力下降，甚至宕机。发现 CPU 过高时，应根据集群当前节点配置情况和业务情况，提高节点规格或降低业务请求量。
内存利用率(%)	统计周期1分钟，>85%，持续5个周期，每30分钟告警一次	JVM 内存使用率表示集群各节点 JVM 内存使用率的值。JVM 内存使用率过高会导致读写操作被拒绝，集群 GC 频繁，甚至出现 OOM 等问题。当发现 JVM 内存使用率超过阈值时，建议通过纵向扩容的方式提高集群节点的规格。
磁盘使用率(%)	统计周期1分钟，>80%，持续5个周期，每30分钟告警一次	平均磁盘使用率表示集群各节点磁盘使用率的平均值。磁盘使用率过高会导致节点没有足够的磁盘空间容纳分配到该节点上，从而导致消息无法落盘，建议在平均磁盘使用率超过75%时及时清理数据或扩容集群。
未消费的消息条数（Count）	统计周期5分钟，>8000，持续10个周期，每30分钟告警一次	堆积过多的消息会导致broker 节点磁盘使用率迅速上涨，无法再接入更多消息，服务会停止。需要进行扩容
生产峰值带宽（MB/s）	统计周期1分钟，>所购买的实例带宽规格，持续5个周期，每10分钟告警一次	一分钟内，客户每秒的流量最大值。判断是否超出当前所购买的流量上限。可根据此项适当选择升配操作等

配置告警

操作场景

云平台默认为所有用户提供云监控功能，无需用户手动开通。用户在使用了云平台某个产品后，云监控才可以开始收集监控数据。
消息队列CKafka支持监控您账户下创建的资源，包括实例、Topic、Consumer Group，帮助您实时掌握资源状态。您可以为监控指标配置告警规则，当监控指标达到设定的报警阈值时，云监控可以通过邮件、短信等方式通知您，帮助您及时应对异常情况。

操作步骤

创建的告警会将一定周期内监控的指标与给定阈值的情况进行比对，从而判断是否需要触发相关通知。当CKafka 状态改变而导致告警触发后，您可以及时进行相应的预防或补救措施，合理地创建告警能帮助您提高应用程序的健壮性和可靠性。

> 注意：
>
> 请务必对实例配置告警，防止因突发流量或者到达规格限制而导致的异常。

1. 登录 CKafka 控制台 。
2. 在实例列表中，单击操作列的【配置告警】可以直接跳转到告警配置页面。
3. 在告警策略页面，选择好策略类型和要设置告警的实例，设置好告警规则和告警通知模板。
    - 策略类型：选择【消息服务CKafka】。
    - 告警对象：选择需要配置告警策略的CKafka资源。
    - 触发条件：支持【选择模板】和【手动配置】，默认选择手动配置，手动配置参见以下说明，新建模板参见新建触发条件模版。

      > 说明：
      >
      > - 指标：例如“磁盘使用百分比”，选择统计粒度为1分钟，则在1分钟内，磁盘使用百分比连续N个数据点超过阈值，就会出发告警。
      >   告警频次：例如“每30分钟警告一次”，指每30分钟内，连续多个统计周期指标都超过了阈值，如果有一次告警，30分钟内就不会再次进行告警，直到下一个30分钟，如果指标依然超过阈值，才会再次告警。

- 通知模板：选择通知模版，也可以新建通知模版，设置告警接收对象和接收渠道。

4. 单击【完成】。

新建触发条件模板

1. 登录 [云监控控制台] 。
2. 在左侧导航栏中，单击【触发条件模板】，进入触发条件列表页面。
3. 在触发条件模板页单击【新建】。
4. 在新建模板页，配置策略类型。
   - 策略类型：选择【消息服务CKafka】。
   - 使用预置触发条件：勾选此选项，会出现系统建议的告警策略。
5. 确认无误后，单击【保存】。
6. 返回新建告警策略页，单击【刷新】，就会出现刚配置的告警策略模板。

权限控制
访问管理 CAM

CAM 基本概念

主账号通过给子账号绑定策略实现授权，策略设置可精确到 [API，资源，用户/用户组，允许/拒绝，条件] 维度。

账户

- 主账号：拥有所有资源，可以任意访问其任何资源。
- 子账号：包括子用户和协作者。
- 子用户： 由主账号创建，完全归属于创建该子用户的主账号。
- 协作者：本身拥有主账号身份，被添加作为当前主账号的协作者，则为当前主账号的子账号之一，可切换回主账号身份。
- 身份凭证：包括登录凭证和访问证书两种，登录凭证 指用户登录名和密码，访问证书 指云 API 密钥（ SecretId 和 SecretKey ）。

资源与权限

- 资源：资源是云服务中被操作的对象，如一个云服务器实例、COS 存储桶、VPC 实例等。
- 权限：权限是指允许或拒绝某些用户执行某些操作。默认情况下，主账号拥有其名下所有资源的访问权限，而子账号没有主账号下任何资源的访问权限。
- 策略：策略是定义和描述一条或多条权限的语法规范。主账号通过将策略关联到用户/用户组完成授权。

访问控制策略示例

CKafka 全读写策略

授权一个子用户以 CKafka 服务的完全管理权限（ 创建、管理等全部操作 ）。

```
{
  "version": "2.0",
  "statement": [
   {
     "action": [
          "name/ckafka:*",
          "name/monitor:GetMonitorData"
     ],
     "resource": "*",
     "effect": "allow"
```

```
    }
  ]
}
```

您也可以通过设置系统的全读写策略支持。

1. 登录访问管理控制台。
2. 在左侧菜单栏中，单击【策略】。
3. 在策略列表中，单击【新建自定义策略】。
4. 在选择创建策略方式的弹窗中，选择【按策略语法创建】。
5. 在模板类型中，搜索“CKafka”，选择消息服务（CKafka）全读写访问权限【QcloudCKafkaFullAccess】，单击【下一步】。
6. 单击【创建策略】。

CKafka 实例只读策略

1. 按照策略生成器创建，授权列表类权限和产品监控权限。

```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action": [
                "name/ckafka:ListInstance",
                "name/monitor:GetMonitorData"
            ],
            "resource": [
                "*"
            ]
        }
    ]
}
```

2. 授权单实例只读权限。

> 注意：
>
> List* 接口不支持资源粒度的鉴权。

```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
```

```
            "action": [
                "name/monitor:GetMonitorData",
                "name/ckafka:Get*"
            ],
            "resource": [
                "qcs::ckafka:gz::ckafkaId/uin/$createUin/$instanceId"
            ]
        }
    ]
}
```

您也可以通过设置系统的只读策略支持。

1. 登录访问管理控制台。
2. 在左侧菜单栏中，单击【策略】。
3. 在策略列表中，单击【新建自定义策略】。
4. 在选择创建策略方式的弹窗中，选择【按策略语法创建】。
5. 在模板类型中，搜索“CKafka”，选择消息服务（CKafka）只读访问策略【QcloudCkafkaReadOnlyAccess】，单击【下一步】。
6. 单击【创建策略】。

配置ACL策略

操作场景

该任务指导您在使用消息队列 CKafka 时，通过控制台配置 SASL 鉴权和 ACL 规则，增强对网络传输中的用户访问控制，增加对 Topic 等资源的生产消费权限控制。

> 说明：
>
> ACL 访问控制列表（Access Control List），帮助用户定义一组权限规则，允许/拒绝用户 user 通过 IP 读/写 Topic 资源。

操作步骤

配置 ACL 策略

1. 登录 CKafka 控制台。
2. 在顶部菜单栏，选择地域后，单击目标实例“ID/名称”。
3. 在实例详情页面，单击顶部 用户管理 页签。
4. 在用户管理页面，单击 新建，填写用户名和密码信息，创建用户。
5. 单击顶部 ACL策略管理。
6. 在 ACL 策略详情页面，单击资源操作列的 编辑 ACL 策略。
7. 单击新建，为用户授予权限。

   > 说明：
   >
   > - 若只设置允许规则，则除允许的规则外的其他IP都无法连接实例。
   > - 若只设置拒绝规则，则除拒绝的规则外的其他IP都可以连接实例。
   > - 若同时设置允许规则和拒绝规则，则只有允许规则中的IP可以连接实例，其他IP都无法连接实例。



后续处理

完成授权后，用户可以通过 SASL 接入点接入消息队列 CKafka 并使用 PLAIN 机制消费消息。

标签管理

简介

标签是云平台提供的用于标识云上资源的标记，是一个键-值对（Key-Value）。
您可以根据各种维度（例如业务、用途、负责人等）使用标签对 CKafka 消息队列资源进行分类管理，通过标签非常方便地筛选过滤出对应的资源。标签键值对云平台没有任何语义意义，会严格按字符串进行解析匹配。

使用限制

- 数量限制
  - 每个云资源允许的最大标签数是50。
- 标签键限制
  - qcloud、project 开头为系统预留标签键，禁止创建。
  - 只能为字母、数字、空格或汉字，支持 + 、 - 、 = 、 . 、 _ 、 : 、 / 、 @ 。
  - 标签键长度最大为255个字符。
- 标签值限制
  - 只能为字母、数字、空格或汉字，支持 + 、 - 、 = 、 . 、 _ 、 : 、 / 、 @ 。
  - 标签值最大长度为127个字符。

使用示例

案例背景

某公司在云平台上拥有10个 CKafka 消息队列实例，分属电商、游戏、文娱三个部门，服务于营销活动、游戏 A、游戏 B、后期制作等业务，三个部门对应的运维负责人为张三、李四、王五。

设置标签

为了方便管理，该公司使用标签分类管理对应的 CKafka 消息队列资源，定义了下述标签键/值。

标签键	标签值
部门	电商、游戏、文娱
业务	营销活动、游戏 A、游戏 B、后期制作

标签键	标签值
运维负责人	张三、李四、王五

将这些标签键/值绑定到 CKafka 上，资源与标签键/值的关系如下表所示：

iD	部门	业务	运维负责人
ckafka-1jqwv1	电商	营销活动	王五
ckafka-1jqwv12	电商	营销活动	王五
ckafka-1jqwv13	游戏	游戏 A	张三
ckafka-1jqwv13	游戏	游戏 B	张三
ckafka-1jqwv14	游戏	游戏 B	张三
ckafka-1jqwv15	游戏	游戏 B	李四
ckafka-1jqwv16	游戏	游戏 B	李四
ckafka-1jqwv17	游戏	游戏 B	李四
ckafka-1jqwv18	文娱	后期制作	王五
ckafka-1jqwv19	文娱	后期制作	王五
ckafka-1jqwv110	文娱	后期制作	王五

使用标签

- 筛选出王五负责的 CKafka 实例：
  按照筛选规则筛选出运维负责人为“王五”的 CKafka 资源即可，具体筛选步骤请参考 使用标签 。
- 筛选出游戏部门中李四负责的 CKafka 实例：
  按照筛选规则筛选出部门为“游戏”、运维负责人为“李四”的 CKafka 资源即可，具体筛选步骤请参考 使用标签 。

设置标签

> 说明：
>
> 您可以对单个或多个实例编辑标签，最多支持对20个实例进行标签的批量编辑操作。

1. 登录 CKafka 控制台。

2. 在左侧导航栏点击 实例列表，勾选需要编辑标签的实例，单击列表上方的 编辑标签。
3. 在编辑标签的操作框中，添加、修改或删除标签。
    - 添加标签：点击添加，选择标签键和标签值。
    - 删除标签：点击要删除的标签后方的叉号。
    - 新建标签：点击标签管理前往标签管理控制台新建标签。
4. 单击确定，完成编辑。

使用标签筛选资源

根据标签对实例进行资源筛选，过滤出对应的资源。

1. 登录 CKafka 控制台。
2. 在左侧导航栏点击实例列表，在实例列表页顶部，选择地域。
3. 在实例列表右上角的搜索框，单击空白处，选择 标签。
4. 在**标签：**后输入标签键。
   例如，筛选出绑定了某个标签键（如 name）的实例，输入“name”（区分大小写）。
5. 回车或单击搜索 icon，进行搜索。

消息查询

操作场景

若您遇到消息消费异常，可以在 CKafka 控制台查询异常消息来排查问题。该任务指导您通过 CKafka 控制台查询消息。CKafka 控制台支持按位点查询和按时间查询两种方式查询消息，适用场景如下：

- 按位点查询：用户能明确知道消息发送至 Topic 的分区 ID 以及消息位点。
- 按时间查询：用户不确定消息的位置，但确定消息发送的时间。

> 注意
>
> - 单次查询最多查询起始时间或者起始位点之后的20条消息，并不会列出所有消息。
> - 查询消息也会消耗一定的带宽资源，切勿频繁操作。
> - 目前消息查询仅支持查询 1 MB 内的消息。

操作步骤

1. 登录 CKafka 控制台，在左侧导航栏选择消息查询。
2. 在消息查询页面，选择要需要查询的实例和 Topic，以及查询方式，单击查询，查看消息信息。



- 分区 ID：消息的 Topic 分区。
- 位点：消息的消费位点。
- 时间戳：为 ProducerRecord 中的 timestamp 。
- 操作：单击消息详情可查看消息的 key 和 value。


消息详情
当前查询的消息已经被强制转换为String类型，如出现乱码，请分析您消息的序列化格式以及编码格式
Key
暂无数据
Value
XFLkADQmSN

开发指南
CKafka 事务管理

Kafka 的事务功能是为了支持在分布式环境中实现原子性操作而设计的。它允许生产者在发送消息时确保消息的完整性和一致性，特别是在需要多条消息作为一个整体进行处理的场景中。以下是 Kafka 事务的主要概念和功能介绍。

事务相关概念

事务的基本概念

原子性：事务中的所有操作要么全部成功，要么全部失败。Kafka 确保在事务中发送的消息要么被成功写入到主题中，要么不写入。
一致性：事务在执行前后，数据的状态应该保持一致。
隔离性：事务之间的操作是相互独立的，一个事务的执行不应影响其他事务的执行。
持久性：一旦事务被提交，其结果是永久性的，即使系统崩溃也不会丢失。

事务的工作流程

Kafka 的事务工作流程主要包括以下几个步骤：

1. **启动事务：**生产者在发送消息之前调用 initTransactions() 方法来初始化事务。
2. **发送消息：**生产者可以发送多条消息到一个或多个主题，这些消息会被标记为事务性消息。
3. 提交或中止事务：
    - **提交事务：**如果所有消息都成功发送，生产者调用 commitTransaction() 方法来提交事务，所有消息将被写入到 Kafka。
    - **中止事务：**如果在发送过程中发生错误，生产者可以调用 abortTransaction() 方法来中止事务，所有消息将不会被写入。

事务的配置

要使用 Kafka 的事务功能，您需要在生产者配置中设置以下参数：

- transactional.id ：每个事务性生产者都需要一个唯一的标识符。这个 ID 用于标识事务的所有消息。
- acks ：设置为 all 以确保所有副本都确认消息。
- enable.idempotence ：设置为 true 以启用幂等性，确保消息不会被重复发送。

事务的限制

- 性能开销：使用事务会引入额外的性能开销，因为需要进行更多的协调和确认。
- 事务超时：Kafka 对事务有超时限制，默认情况下为 60 秒。如果事务在此时间内未提交或中止，将会被自动中止。

- 消费者的处理：消费者在处理事务性消息时需要注意，只有在事务提交后，消费者才能看到这些消息。

事务使用示例

producer

```
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 java.util.Properties;

public class TransactionalProducerDemo {
    public static void main(String[] args) {
        // Kafka 配置
        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // Kafka broker 地址
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serializati
on.StringSerializer");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serializ
ation.StringSerializer");
        props.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, "my-transactional-id"); // 事务 ID
        props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG, "true"); // 启用幂等性

        // 创建 Kafka 生产者
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);

        // 初始化事务
        producer.initTransactions();

        try {
            // 开始事务
            producer.beginTransaction();

            // 发送消息
            for (int i = 0; i < 10; i++) {
                ProducerRecord<String, String> record = new ProducerRecord<>("my-topic", "key-" + i, "val
ue-" + i);
                RecordMetadata metadata = producer.send(record).get(); // 发送消息并等待确认
                System.out.printf("Sent message: key=%s, value=%s, partition=%d, offset=%d%n",
                        record.key(), record.value(), metadata.partition(), metadata.offset());
            }

            // 提交事务
```

```
            producer.commitTransaction();
            System.out.println("Transaction committed successfully.");
        } catch (Exception e) {
            // 如果发生异常，回滚事务
            producer.abortTransaction();
            System.err.println("Transaction aborted due to an error: " + e.getMessage());
        } finally {
            // 关闭生产者
            producer.close();
        }
    }
}
```

consumer

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class TransactionalConsumerDemo {
    public static void main(String[] args) {
        // Kafka 配置
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // Kafka broker 地址
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "my-consumer-group"); // 消费者组 ID
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.seriali
zation.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.seri
alization.StringDeserializer");
        props.put(ConsumerConfig.ISOLATION_LEVEL_CONFIG, "read_committed"); // 只读取已提交的事务
消息

        // 创建 Kafka 消费者
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

        // 订阅主题
        consumer.subscribe(Collections.singletonList("my-topic"));

        try {
            while (true) {
                // 拉取消息
```

```
                ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, String> record : records) {
                    System.out.printf("Consumed message: key=%s, value=%s, partition=%d, offset=%d%n",
                            record.key(), record.value(), record.partition(), record.offset());
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 关闭消费者
            consumer.close();
        }
    }
}
```

Kafka 事务管理

在 Kafka 中，事务管理涉及到多个组件和数据结构，以确保事务的原子性和一致性。事务信息的内存占用主要与以下几个方面有关：

事务 ID 和 Producer ID

事务 ID：每个事务都有一个唯一的事务 ID，用于标识该事务。事务 ID 是由生产者在发送消息时指定的，通常是一个字符串。
Producer ID：每个生产者在连接到 Kafka 时会被分配一个唯一的 Producer ID。这个 ID 用于标识生产者的消息，并确保消息的顺序性和幂等性。

事务状态管理

Kafka 使用一个称为 事务状态日志 的内部主题来管理事务的状态。这个日志记录了每个事务的状态（如进行中、已提交、已中止）以及与该事务相关的消息。事务状态日志的管理涉及以下几个方面：

- 内存中的数据结构：Kafka 在内存中维护一个数据结构（例如哈希表或映射），用于存储当前活动的事务信息。这些信息包括事务 ID、Producer ID、事务状态、时间戳等。
- 持久化存储：事务状态日志会被持久化到磁盘，以确保在 Kafka 服务器重启或故障恢复时能够恢复事务状态。

事务信息的内存占用

事务信息的内存占用主要取决于以下两个因素：

- 活动事务的数量：当前正在进行的事务数量直接影响内存占用。每个活动事务都会在内存中占用一定的空间。
- 事务的元数据：每个事务的元数据（例如事务 ID、Producer ID、状态等）也会占用内存。具体的内存占用量取决于这些元数据的大小。

事务的清理

为了防止内存占用过高，Kafka 会根据配置的过期时间定期检查并清理已完成的事务，默认保留 7 天，过期删除。

事务常见的 FullGC / OOM 问题

从事务管理可以看出，事务信息会占用大量内存。其中影响事务信息占用内存大小的最直接的两个因素就是：事务 ID 的数量和 Producer ID 的数量。

- 其中事务 ID 的数量指的是客户端往 Broker 初始化、提交事务的数量，这个与客户端的事务新增提交频率强相关。
- Producer ID 指的是 Broker 内每个 Topic 分区存储的 producer 状态信息，因此 Producer ID 的数量与 broker 的分区数量强相关。
  在事务场景中，事务 ID和 Producer ID 强绑定，如果同一个和事务 ID 绑定的 Producer ID 往 broker 内所有的分区都发送消息，那么一个 broker 内的 Producer ID 的数量理论上最多能达到事务 ID 数量与 broker 内分区数量的乘积。假设一个实例下的事务 ID 数量为 t，一个 broker 下的分区数量为 p，那么 Producer ID 的数量最大能达到 t * p。
  > 说明：
  >
  > 因此，假设一个 broker 下的事务 ID 数量为 t，平均事务内存占用大小为 tb，一个 broker 下的分区数量为 p，平均一个 Producer ID 占用大小为 pb，那么该 broker 内存中关于事务信息占用的内存大小为：t * tb + t * p * pb。

可以看出有两种场景可能会导致内存占用暴涨：

- 客户端频繁往实例初始化新增提交新的事务 ID。
- 同一个事务 ID 往多个分区发送数据，Producer ID 的叉乘数量会上涨的非常恐怖，很容易将内存打满。
  > 说明：
  >
  > 因此，无论是对 Flink 客户端还是自己实现的事务 producer，都尽量避免这两种场景。例如对于 Flink，可以适当降低 checkpoint 的频率，以减小由于事务 ID 前缀+随机串计算的事务 ID 变化的频率。另外就是尽量保证同一个事务 ID 往同一个分区发送数据。

Flink 使用事务注意事项

对于 Flink 有以下优化手段，来保证事务信息不会急剧膨胀：

- 客户端优化参数：Flink 加大 checkpoint 间隔（详情可参见 社区 ISSUE）。
- Flink 生产任务可优化 sink.partitioner 为 Fixed 模式。
  flink 参数说明：https://nightlies.apache.org/flink/flink-docs-master/zh/docs/connectors/table/kafka/

CKafka 版本选择建议

本文为您介绍 CKafka 和社区版 Kafka 的兼容性，帮助您在使用腾 CKafka 时根据业务需求选择更加适合您的版本。

概述

社区版 Kafka 目前共演进了 0.7.x 到 3.3.x 大概30个版本，从消息队列的角度可分为四个阶段：0.x、1.x、2.x、3.x。目前云平台针对这四个社区发展阶段均提供了云上兼容版本，基本覆盖了用户使用的主流 Kafka 版本。
其中 1.x 和 2.x 这两个大版本主要是对 Kafka Streams 的优化和改进，在消息引擎方面并未引入太多的重大功能特性（2.x 在事务特性方面有较大改进）。Kafka Streams 在 2.x 版本有较大改进，如果您是这些特性的用户，请至少选择 2.x 的版本。3.x 版本在 KRaft、mirrormaker2 等方面有较大改进，详情可以参考社区 release notes。

兼容性说明

CKafka 兼容社区 Kafka，其中高版本和低版本是完全向下兼容的。例如：自建 0.10 版本的 Kafka，在云上选择 0.10、1.1.1、2.4.1版本的 CKafka 均可；如果自建是高版本，不建议选择低版本（因为不确定业务是否使用高版本携带的特性）。
以下是兼容性说明：

CKafka 版本	可兼容社区版本	兼容性
0.10.2（已停售）	≤ 0.10.x	100%
1.1.1（已停售）	≤ 1.1.x	100%
2.4.1	≤ 2.4.x	100%
2.8.1（推荐）	≤ 2.8.x	100%
3.2.3	≤ 3.2.x	100%

关于 CKafka2.4.1 版本说明
CKafka 上线2.4版本时，社区的稳定分支为2.4.1版，后来社区有一个开发分支2.4.2版本，进行一些修复合入后，定位为2.4.2版本。后续社区删除了2.4.2版本，最终社区上没有2.4.2版本，因此 CKafka 之前显示的2.4.2版本和现在显示的2.4.1版本对齐。

CKafka 版本选择建议

- 如果是自建 Kafka 上云，建议选择对应的大版本即可。例如：自建 Kafka 是2.8.1版本，则选择 CKafka 的

2.8.1版本。

- 当在云上找不到对应的版本时，建议向上选择版本。例如：自建是2.8.0版本，则建议使用2.8.1版本；自建是1.1.1版本，则建议使用版本2.4.1（因为 Broker 的每个版本是向下兼容的）。

CKafka 常见参数配置说明

Broker 配置参数说明

当前 CKafka broker 端的一些配置如下，供参考：

```
# 消息体的最大大小，单位是字节
message.max.bytes=1000012

# 是否允许自动创建 Topic, 默认是 false，当前可以通过控制台或云 API 创建
auto.create.topics.enable=false

# 是否允许调用接口删除 Topic
delete.topic.enable=true

# Broker 允许的最大请求大小为16MB
socket.request.max.bytes=16777216

# 每个 IP 与 Broker 最多建立5000个连接
max.connections.per.ip=5000

# offset 保留时间，默认为7天
offsets.retention.minutes=10080

# 没有 ACL 设置时，允许任何人访问
allow.everyone.if.no.acl.found=true

# 日志分片大小为1GB
log.segment.bytes=1073741824

# 日志滚动检查间隔5分钟，当设置保留时间小于5分钟时，也可能需要等待5分钟才会清空日志
log.retention.check.interval.ms=300000
```

> 说明
>
> 其他未列出的 Broker 配置参见 开源 Kafka 默认配置。

Topic 配置参数说明

1. 选取合适的分区数量

从生产者的角度来看，向不同的 partition 写入是完全并行的；从消费者的角度来看，并发数完全取决于 partition 的数量（如果 consumer 数量大于 partition 数量，则必有 consumer 闲置）。因此选取合适的分区数量对于发挥

CKafka 实例的性能十分重要。
partition 的数量需要根据生产和消费的吞吐来判断。理想情况下，可以通过如下公式来判断分区的数目：

```
Num =  max( T/PT , T/CT ) = T / min( PT ,  CT )
```

其中，Num 代表 partition 数量，T 代表目标吞吐量，PT 代表生产者写入单个 partition 的最大吞吐，CT 代表消费者从单个 partition 消费的最大吞吐。则 partition 数量应该等于 T/PT 和 T/CT 中较大的那一个。
在实际情况中，生产者写入单个 partition 的最大吞吐 PT 的影响因素和批处理的规模、压缩算法、确认机制、副本数等有关。消费者从单个 partition 消费的最大吞吐 CT 的影响因素和业务逻辑有关，需要在不同场景下实测得出。
通常建议 partition 的数量一定要大于等于消费者的数量来实现最大并发。 如果消费者数量是 5，则 partition 的数目也应该是 ≥ 5 的。同时，过多的分区会导致生产吞吐的降低和选举耗时的增加，因此也不建议过多分区。提供如下信息供参考：

- 单个 partition 是可以实现消息的顺序写入的。
- 单个 partition 只能被同消费者组的单个消费者进程消费。
- 单个消费者进程可同时消费多个 partition，即 partition 限制了消费端的并发能力。
- partition 越多则失败后 leader 选举的耗时越长。
- offset 的粒度最细是在 partition 级别的，partition 越多，查询 offset 就越耗时。
- partition 的数量是可以动态增加的，只能增加不能减少。但增加会出现消息 rebalance 的情况。

2. 选取合适的副本

目前为了保证可用性副本数必须大于等于2，如果需要保障高可靠建议3副本。

> 注意
>
> 副本数会影响生产/消费流量，如3副本则实际流量 = 生产流量 × 3。

3. 日志保留时间

Topic 的 log.retention.ms 配置通过控制台实例的保留时间统一设置。

4. 其他 Topic 级别配置说明

```
# Topic 级别最大消息大小
max.message.bytes=1000012

# 0.10.2 版本消息格式为 V1 格式
message.format.version=0.10.2-IV0

# 不在 ISR 中的 replica 允许选择为 Leader，可用性高于可靠性，存在数据丢失风险。
unclean.leader.election.enable=true

# ISR 提交生产者请求的最小副本数。如果同步状态的副本数小于该值，服务器将不再接受。request.require
d.acks为-1或all的写入请求。
```

```
min.insync.replicas=1
```

生产者配置指南

生产端常用参数配置如下，建议客户根据实际业务场景调整配置：

```
# 生产者会尝试将业务发送到相同的 Partition 的消息合包发送到 Broker，batch.size 设置合包的大小上限。默认为 16KB。batch.size 设太小会导致吞吐下降，设太大会导致内存使用过多。
batch.size=16384

# Kafka producer 的 ack 有 3 种机制，分别说明如下：
# -1 或 all：Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后，才应答给 Producer 继续发送下一条（批）消息。 这种配置提供了最高的数据可靠性，只要有一个已同步的副本存活就不会有消息丢失。注意：这种配置不能确保所有的副本都写入该数据才返回，可以配合 Topic 级别参数 min.insync.replicas 使用。
# 0：生产者不等待来自 broker 同步完成的确认，继续发送下一条（批）消息。这种配置生产性能最高，但数据可靠性最低（当服务器故障时可能会有数据丢失，如果 leader 已死但是 producer 不知情，则 broker 收不到消息）
# 1：生产者在 leader 已成功收到的数据并得到确认后再发送下一条（批）消息。这种配置是在生产吞吐和数据可靠性之间的权衡（如果leader已死但是尚未复制，则消息可能丢失）

# 用户不显式配置时，默认值为1。用户根据自己的业务情况进行设置
acks=1

# 配置生产者用来缓存消息等待发送到 Broker 的内存。用户要根据生产者所在进程的内存总大小调节
buffer.memory=33554432

# 当生产消息的速度比 Sender 线程发送到 Broker 速度快，导致 buffer.memory 配置的内存用完时会阻塞生产者 send 操作，该参数设置最大的阻塞时间
max.block.ms=60000

# 设置消息延迟发送的时间(ms)，这样可以等待更多的消息组成 batch 发送。默认为0表示立即发送。当待发送的消息达到 batch.size 设置的大小时，不管是否达到 linger.ms 设置的时间，请求也会立即发送
# 推荐用户根据实际使用场景，设置linger.ms在100~1000之间，更大的取值相对有更大的吞吐但会相应增加时延
linger.ms=100

# 设置分区缓存消息量(bytes)，达到该数值时生产者将批量的消息发送给broker。默认值为16384，过小的batch.size会增加发送请求次数可能会损耗性能和影响稳定性，用户根据实际场景，可适当增大该值。注：该值为上限值，当还未达到时若时间已经达到linger.ms时生产者会发送消息
batch.size=16384

# 生产者能够发送的请求包大小上限，默认为1MB。在修改该值时注意不能超过 Broker 配置的包大小上限16MB
max.request.size=1048576
```

```
# 压缩格式配置，目前 0.9(包含)以下版本不允许使用压缩，0.10（包含）以上不允许使用 GZip 压缩
compression.type=[none, snappy, lz4]

# 客户端发送给 Broker 的请求的超时时间，不能小于 Broker 配置的 replica.lag.time.max.ms，目前该值为10000ms
request.timeout.ms=30000

# 客户端在每个连接上最多可发送的最大的未确认请求数，该参数大于1且 retries 大于0时可能导致数据乱序。希望消息严格有序时，建议客户将该值设置1
max.in.flight.requests.per.connection=5

# 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
retries=0

# 发送请求失败时到下一次重试请求之间的时间
retry.backoff.ms=100
```

消费者配置指南

消费端常用参数配置如下，建议客户根据实际业务场景调整配置：

```
# 是否在消费消息后将 offset 同步到 Broker，当 Consumer 失败后就能从 Broker 获取最新的 offset
enable.auto.commit=true

# 当 auto.commit.enable=true 时，自动提交 Offset 的时间间隔，建议设置至少1000
auto.commit.interval.ms=5000

# 当 Broker 端没有 offset（如第一次消费或 offset 超过7天过期）时如何初始化 offset，当收到 OFFSET_OUT_OF_RANGE 错误时，如何重置 Offset
# earliest：表示自动重置到 partition 的最小 offset
# latest：默认为 latest，表示自动重置到 partition 的最大 offset
# none：不自动进行 offset 重置，抛出 OffsetOutOfRangeException 异常
auto.offset.reset=latest

# 标识消费者所属的消费分组
group.id=""

# 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，认为该消费者故障失败，Broker 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置group.min.session.timeout.ms=6000和group.max.session.timeout.ms=300000 之间
session.timeout.ms=10000

# 使用 Kafka 消费分组机制时，消费者发送心跳的间隔。这个值必须小于 session.timeout.ms，一般小于它的
```

三分之一
heartbeat.interval.ms=3000

# 使用 Kafka 消费分组机制时，再次调用 poll 允许的最大间隔。如果在该时间内没有再次调用 poll，则认为该消费者已经失败，Broker 会重新发起 Rebalance 把分配给它的 partition 分配给其他消费者
max.poll.interval.ms=300000

# Fecth 请求最少返回的数据大小。默认设置为 1B，表示请求能够尽快返回。增大该值会增加吞吐，同时也会增加延迟
fetch.min.bytes=1

# Fetch 请求最多返回的数据大小，默认设置为 50MB
fetch.max.bytes=52428800

# Fetch 请求等待时间
fetch.max.wait.ms=500

# Fetch 请求每个 partition 返回的最大数据大小，默认为1MB
max.partition.fetch.bytes=1048576

# 在一次 poll 调用中返回的记录数
max.poll.records=500

# 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
request.timeout.ms=305000

CKafka 日志分片滚动与保留机制

Kafka通过日志分片滚动与日志保留优化存储管理。以下日志分片滚动和日志保留原理的说明将介绍其工作原理与配置方法。

日志分片滚动

在 Kafka 中，每个主题（topic）被划分为多个分区（partition），每个分区的日志被进一步划分为多个分片（segment）。Kafka 通过以下两种方式控制日志分片的滚动：

- 基于时间的滚动（segment.ms）：当当前活动分片的存在时间超过配置的 segment.ms 时，Kafka 会关闭当前分片并创建一个新的分片。
- 基于大小的滚动（segment.bytes）：当当前活动分片的大小达到配置的 segment.bytes 时，Kafka 会关闭当前分片并创建一个新的分片。
  一旦日志分片被关闭，它就变为非活动状态，成为日志保留策略的管理对象。

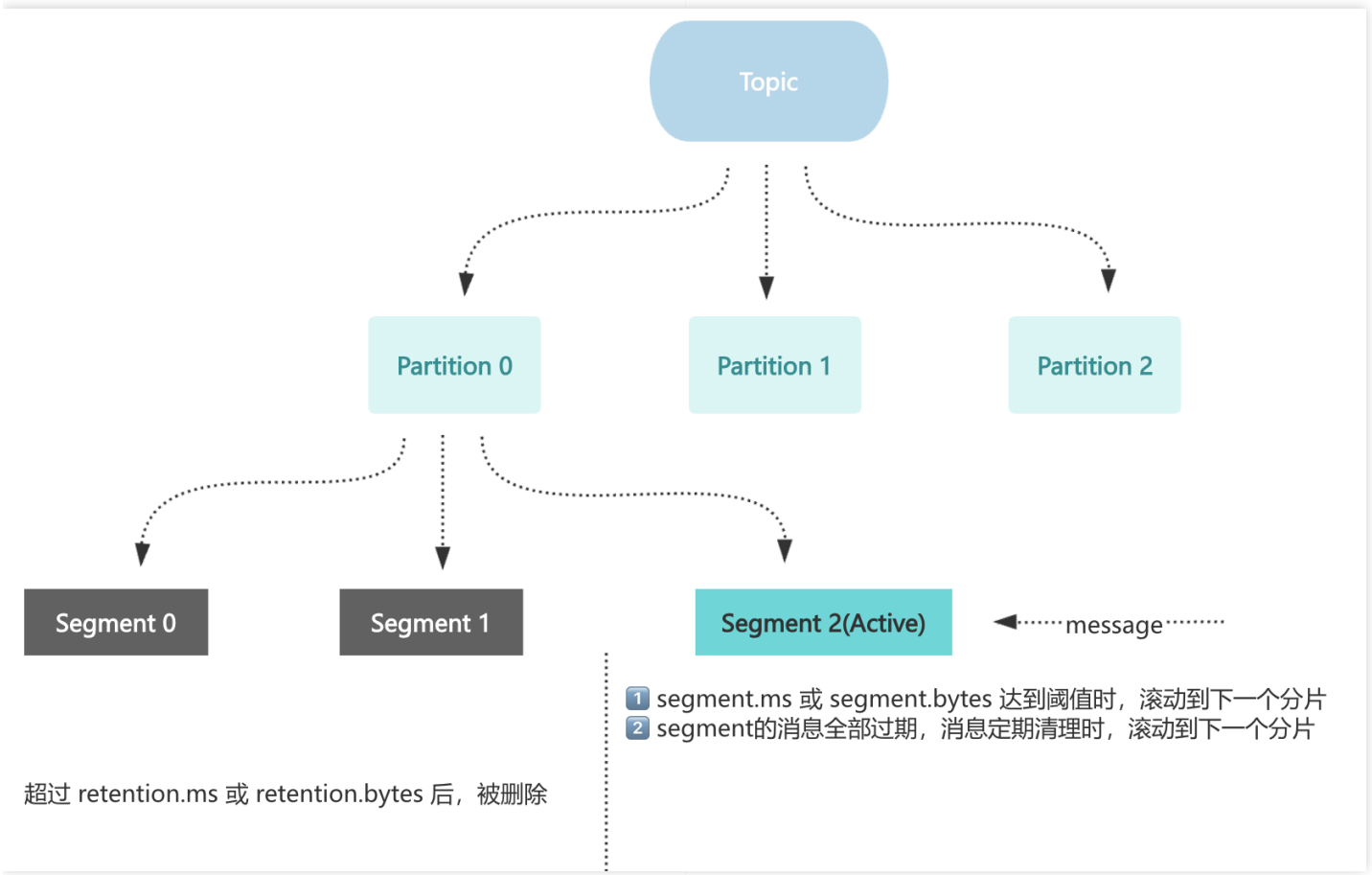
日志保留原理

Kafka 的日志保留策略决定了消息在被删除之前可以保留多长时间或占用多大空间。Kafka 提供以下两种保留策略：

- 基于时间的保留（retention.ms）：配置 retention.ms 来指定消息的保留时间。超过该时间的非活动分片将被删除。
- 基于大小的保留（retention.bytes）：配置 retention.bytes 来指定每个分区的最大日志大小。超过该大小的旧分片将被删除。
  当上述任一条件满足时，Kafka 会删除符合条件的非活动分片。

日志分片滚动与保留原理示意图

以下是日志分片滚动与保留原理的示意图：



日志配置限制

Segment配置限制

配置项	默认值	配置范围	实际含义
segment.ms	7 天	1 天~90 天	日志分片滚动的最短周期是1天。
segment.bytes	1GB	固定1GB	每个日志分片固定为1GB，无法更改。

> 说明：
>
> - 由于 segment.ms 最短可设为1天，因此Kafka至少会每隔1天强制创建新分片，即使分片大小未达到1GB。
> - 每个日志分片文件最大容量为 1GB，到达该大小后会立即滚动生成新分片。

Retention 配置限制

配置项	默认值	配置范围	实际含义

配置项	默认值	配置范围	实际含义
retention.ms	3 天	1分钟~90天	日志的最短保留期可精细到1分钟。
retention.bytes	1GB	1GB~1024GB	日志保留最小容量为1GB（即至少保留1分片日志）。

> 说明：
>
> Kafka 会基于时间或容量，满足任一条件即触发日志删除。

场景说明

场景一：期望保留日志1天

客户配置：

- retention.ms 设置为1天。
- segment.ms 设置为1天。
  极端情况（日志产生量较低）分析：
- Kafka 的 segment.ms 最小只能为1天，当日志量特别少的时候，日志每天滚动一次，生成一个新的 segment 。
- 因此，最多会有2个 segment 同时存在：
  - 第1天产生的 segment（已经滚动到非活动）。
  - 第2天当前活动的 segment 。
- 非活动 segment 在满足保留条件时才会删除：
  - 客户希望保留期1天后第1个 segment 可删除。
  - 但当前活动 segment 至少要达到1天才能被关闭并标记为非活动。
- 因此，实际日志可能最大保留2天（1个活动+1个非活动分片）。
  结论：
- 日志量少的时候，极端情况下，实际最大日志保留可能达到：2天。
- 日志量正常的时候，日志最大保留时间约是1天。

场景二：期望保留日志1小时

客户配置：

- retention.ms 设置为1小时。
- segment.ms 设置为1天。
  极端情况（日志产生量较低）分析：
- 假设日志量极少，未达到1GB，始终无法触发基于 segment.bytes (1GB) 的滚动，Kafka 只能在1天后强制滚动一次。
- 同时活动 segment 始终有1小时内的日志，只能达到至少1天才滚动。

- 这种情况下，日志实际上可能被保留：1天（非活动分片）+1小时（日志过期时间），最长可达到25小时。
  结论：
- 日志量少的时候，极端情况下，实际最大日志保留可能达到：1天 + 1小时（即25小时）。

配置建议表

客户期望保留时长	retention.ms 配置建议	segment.ms 配置建议	实际最大保留时长	备注
1小时	1天	1天	25小时（1天 + 1小时）	因 segment.ms 最小为1天限制。
12小时	12小时	1天	1.5天（1天 + 12小时）	因 segment.ms 最小为1天限制。
1天	1天	1天	2天	1个活动分片（最大1天）+ 1个非活动分片（1天内删除）
N天	N * 86400000	1天	(N+1)天	segment.ms 每天滚动，最多保留N + 1天日志。

CKafka 数据压缩

操作场景

数据压缩可以减少网络 IO 传输量，减少磁盘存储空间。您可以通过本文档，了解数据压缩支持的消息格式，并根据需求配置数据压缩。

消息格式

目前 CKafka 支持两类消息格式，分别为V1版本和V2版本（在0.11.0.0引入）。目前 CKafka 兼容0.9、0.10、1.1、2.4、2.8和3.2版本。
不同版本对应不同的配置，说明如下：

- 消息格式转换主要是为了兼容老版本的消费者程序，在一个 CKafka 集群中通常同时保存多种版本的消息格式（V1/V2）。
- Broker 端会对新版本消息执行向老版本格式的转换，该过程中会涉及消息的解压缩和重新压缩。
- 消息格式转换对性能的影响很大，除了增加额外的压缩和解压缩操作之外，还会让 CKafka 丧失其优秀的 零拷贝（Zero-copy）特性。因此，一定要保证消息格式的统一。
- 零拷贝（Zero-copy）：数据在磁盘和网络进行传输时，避免昂贵的内核态数据拷贝，从而实现快速的数据传输。

压缩算法对比

官方推荐使用的压缩算法为 Snappy 算法。分析过程如下：
评估一个压缩算法的优劣，主要有两个指标：压缩比、压缩/解压缩吞吐量。
CKafka 2.1.0之前的版本支持三种压缩算法：GZIP、Snappy、LZ4。
在 CKafka 的实际使用中，三种算法的性能指标对比如下：

- 压缩比：LZ4 > GZIP > Snappy
- 吞吐量：LZ4 > Snappy > GZIP
  物理资源占用如下：
- 带宽：由于 Snappy 的压缩比最低，因此占用的网络带宽最大。
- CPU：在压缩时 Snappy 使用更多的 CPU，在解压缩时 GZIP 使用更多的 CPU。
  因此，正常情况下三种压缩算法的推荐排序为：LZ4 > GZIP > Snappy。
  经过长时间的现网运行试验，发现在大多数情况下上面的模型是没问题的。但是在某些极端情况下 LZ4 压缩算法会导致 CPU 负载增大。
  经分析是业务的源数据内容不一样，导致压缩算法的性能表现不一样。 故建议对 CPU 指标敏感的用户采用更

为稳定的 Snappy 压缩算法。

> 说明：
>
> CKafka 不推荐使用 Gzip 压缩算法。开启 Gzip 压缩对 CKafka 服务端有额外的 CPU 消耗。根据压测数据，如果开启 Gzip 压缩，建议预留 75% 左右的带宽 buffer（预留比例仅供参考，实际使用时需要观测具体的监控数据判断）。
>
> 例如：带宽 40MB/s 的实例，开启 Gzip 压缩后，建议将带宽提升到 40/（1-75%） = 160MB/s。

配置数据压缩

生产者可通过下述方法配置数据压缩：

```
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");

// Producer 启动后，生产的每个消息集合都会经过压缩，能够很好地节省网络传输带宽和 Kafka Broker 端的磁盘占用

// 请注意不同版本对应不同的配置，0.9及以下版本不允许使用压缩。1.1及以下版本默认不支持 Gzip 压缩格式。

props.put("compression.type", " lz4 ");
Producer<String, String> producer = new KafkaProducer<>(props);

大部分情况下，Broker 从 Producer 接收到消息后，仅仅只是原封不动地保存，而不会对其进行任何修改
```

说明与注意

- 发送数据到 CKafka，不能设置 compression.codec。
- 1.1及以下版本默认不支持 Gzip 压缩格式，如果需要支持，请提交工单申请。
- Gzip 压缩对于 CPU 的消耗较高，使用 Gzip 会导致所有的消息都是 InValid 消息。CKafka 不推荐使用 Gzip 压缩。
  开启 Gzip 后 CPU 占用率高，成为带宽使用瓶颈。若开启 Gzip，推荐调大生产端的linger.ms、batch.size配置。
- 使用 LZ4 压缩方法时，程序不能正常运行，可能的原因是：消息格式错误。请您检查 CKafka 版本，确认适用的消息格式是否正确。
- 不同 CKafka Client 的 SDK 设置方式不同，您可以通过开源社区进行查询（例如 C/C++ Client 的说明），设

置消息格式的版本。

接入低版本自建 Kafka

CKafka 兼容0.9及以上的生产/消费接口（目前可以直接购买的版本包括 2.4.1、2.8.1、3.2.3 版本），如果接入低版本（例如0.8版本）的自建 Kafka，您需要对接口进行相应改造。本文将从生产端和消费端对比0.8版本 Kafka 和高版本 Kafka，并提供改造方式。

Kafka Producer

概述

Kafka 0.8.1版本中，Producer API 被重写。该客户端为官方推荐版本，其拥有更好的性能和更多的功能，社区将维护新版本的 Producer API。

新旧版本 Producer API 对比

- 新版 Producer API Demo

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:4242");
props.put("acks", "all");
props.put("retries",0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
Producer<String, String> producer = new KafkaProducer<>(props);
producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(0), Integer.toStrin
g(0)));
producer.close();
```

- 旧版 Producer API Demo

```
Properties props = new Properties();
props.put("metadata.broker.list", "broker1:9092");
props.put("serializer.class", "kafka.serializer.StringEncoder");
props.put("partitioner.class", "example.producer.SimplePartitioner");
props.put("request.required.acks", "1");
ProducerConfig config = new ProducerConfig(props);
Producer<String, String> producer = new Producer<String, String>(config);
KeyedMessage<String, String> data = new KeyedMessage<String, String>("page_visits", ip, msg);
producer.send(data);
```

```
        producer.close();
```

可以看出新旧版本的使用方法基本一致，只有一些参数的配置不同，改造代价不大。

兼容性说明

对于 Kafka 而言，0.8.x 版本的 Producer API 都可以顺利接入 CKafka，无需改造。推荐使用新版 Kafka Producer API。

Kafka Consumer

概述

开源 Apache Kafka 0.8版本中提供了两种消费者 API，分别为：

- High Level Consumer API（屏蔽配置细节）
- Simple Consumer API（配置细节支持指定）
  Kafka 0.9.x 版本引入了 New Consumer，其融合了 Old Consumer（0.8版本）两种 Consumer API 的特性，减轻了 ZooKeeper 的负载。
  因此下文给出了0.8版本 Consumer 转换为0.9版本 New Consumer 的方式。

新旧版本 Consumer API 对比

0.8版本 Consumer API

- High Level Consumer API（参见 Demo）
  如果您只需要数据而不考虑消息 offset 相关的处理时，High Level API 可以满足一般性消费要求。High Level Consumer API 围绕着 Consumer Group 逻辑概念展开，屏蔽 Offset 管理、具有 Broker 异常处理、Consumer 负载均衡功能。使开发者可以快速上手 Consumer 客户端。
  在使用 High Level Consumer 时需要注意以下几点：
- 如果消费线程大于 Partition 个数，某些消费线程将无法获得数据。
- 如果 Partition 个数大于线程数目，某些线程会消费多个 Partition。
- Partition 和消费者变动会影响 Rebalance。
- Low Level Consumer API（参见 Demo）
  如果使用者关心消息的 offset 并且希望进行重复消费或者跳读等功能、又或者希望指定某些 partition 进行消费时和确保更多消费语义时推荐使用 Low Level Consumer API。但是使用者需要自己处理 Offset 以及 Broker 的异常情况。
  在使用 Low Level Consumer 时需要注意以下几点：
- 自行跟踪维护 Offset，控制消费进度。
- 查找 Topic 相应 Partition 的 Leader，以及处理 Partition 变更情况。

0.9版本 New Consumer API

Kafka 0.9.x 版本引入了 New Consumer，其融合了 Old Consumer 两种 Consumer API 的特性，同时提供消费者的协调(高级 API)和 lower-level 访问，并构建自定义的消费策略。New Consumer 还简化了消费者客户端，引入中心 Coordinator，解决分别连接 ZooKeeper 产生的 Herd Effect 和 Split Brain 问题，同时也减轻了 ZooKeeper 的负载。
优势：

- Coordinator 引入
  当前版本的 High Level Consumer 存在 Herd Effect 和 Split Brain 的问题。将失败探测和 Rebalance 的逻辑放到一个高可用的中心 Coordinator，那么这两个问题即可解决。同时还可很大程度的减少 ZooKeeper 的负载。
- 允许自己分配 Partition
  为了保持本地每个分区的一些状态不变，所以需要将 Partition 的映射也保持不变。另外一些场景是为了让 Consumer 与地域相关的 Broker 关联。
- 允许自己管理 Offset
  可以根据自己需要去管理 Offset，实现重复、跳跃消费等语意。
- Rebalance 后触发用户指定的回调
- 非阻塞式 Consumer API

新旧版本 Consumer API 功能对比

种类	引入版本	Offset 自动保存	Offset 自行管理	自动进行异常处理	Rebalance 自动处理	Leader 自动查找	优缺点
High Level Consumer	Before 0.9	支持	不支持	支持	支持	支持	Herd Effect 和 Split Brain
Simple Consumer	Before 0.9	不支持	支持	不支持	不支持	不支持	需要处理多种异常情况
New Consumer	After 0.9	支持	支持	支持	支持	支持	成熟，当前版本推荐

Old Consumer 转换 New Consumer

- New Consumer

```
//config中主要变化是 ZooKeeper 参数被替换了
Properties props = new Properties();
```

```
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("session.timeout.ms", "30000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
// 相比old consumer 而言，这里创建消费者更加简单
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
   ConsumerRecords<String, String> records = consumer.poll(100);
   for (ConsumerRecord<String, String> record : records)
     System.out.printf("offset = %d, key = %s, value = %s", record.offset(), record.key(), record.valu
e());}
```

- Old Consumer（High Level）

```
//  old consumer 需要 ZooKeeper
Properties props = new Properties();
props.put("zookeeper.connect", "localhost:2181");
props.put("group.id", "test");
props.put("auto.commit.enable", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("auto.offset.reset", "smallest");
ConsumerConfig config = new ConsumerConfig(props);
// 需要创建connector
ConsumerConnector connector = Consumer.createJavaConsumerConnector(config);
// 创建message stream
Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
topicCountMap.put("foo", 1);
Map<String, List<KafkaStream<byte[], byte[]>>> streams =
 connector.createMessageStreams(topicCountMap);
// 获取数据
KafkaStream<byte[], byte[]> stream = streams.get("foo").get(0);
ConsumerIterator<byte[], byte[]> iterator = stream.iterator();
MessageAndMetadata<byte[], byte[]> msg = null;
while (iterator.hasNext()) {
    msg = iterator.next();
    System.out.println(//
          " group " + props.get("group.id") + //
          ", partition " + msg.partition() + ", " + //
           new String(msg.message()));
}
```

可以看到，改造成 New Consumer 编写更加简单，最主要的变化是将 ZooKeeper 参数的输入替代成了 Kafka 地址输入。同时，New Consumer 也增加了与 Coodinator 交互的参数配置，一般情况下使用默认配置就足够。

兼容性说明

CKafka 与开源社区高版本的 Kafka 一致，支持重写后的 New Consumer API，屏蔽了 Consumer 客户端与 Zookeeper 的交互（Zookeeper不再向用户暴露）。New Consumer 解决原有与 Zookeeper 直接交互的 Herd Effect 和 Split Brain 问题，以及融合了原有 Old Consumer 的特性，使消费环节更加可靠。

最佳实践
Kafka Connect接入CKafka实践

Kafka Connect 目前支持两种执行模式：standalone 和 distributed。

以 standalone 模式启动 connect

通过以下命令以 standalone 模式启动 connect：

```
bin/connect-standalone.sh config/connect-standalone.properties connector1.properties [connector2.properties ...]
```

接入 CKafka 与接入开源 Kafka 没有区别，仅需要修改 bootstrap.servers 为申请实例时分配的 IP。

以 distributed 模式启动 connect

通过以下命令以 distributed 模式启动 connect：

```
bin/connect-distributed.sh config/connect-distributed.properties
```

该模式下，kafka connect 会将 offsets、configs 和 task status 信息存储在 kafka topic 中，存储的 topic 在 connect-distributed 中的以下字段配置：

```
config.storage.topic
offset.storage.topic
status.storage.topic
```

这三个 topic 需要手动创建，才能保证创建的属性符合 connect 的要求。

- config.storage.topic 需要保证只有一个 partition，多副本且为 compact 模式。
- offset.storage.topic 需要有多个 partition，多副本且为 compact 模式。
- status.storage.topic 需要有多个 partition，多副本且为 compact 模式。
  配置 bootstrap.servers 为申请实例是分配的 IP；

配置 group.id，用于标识 connect 集群，需要与消费者分组区分开来。

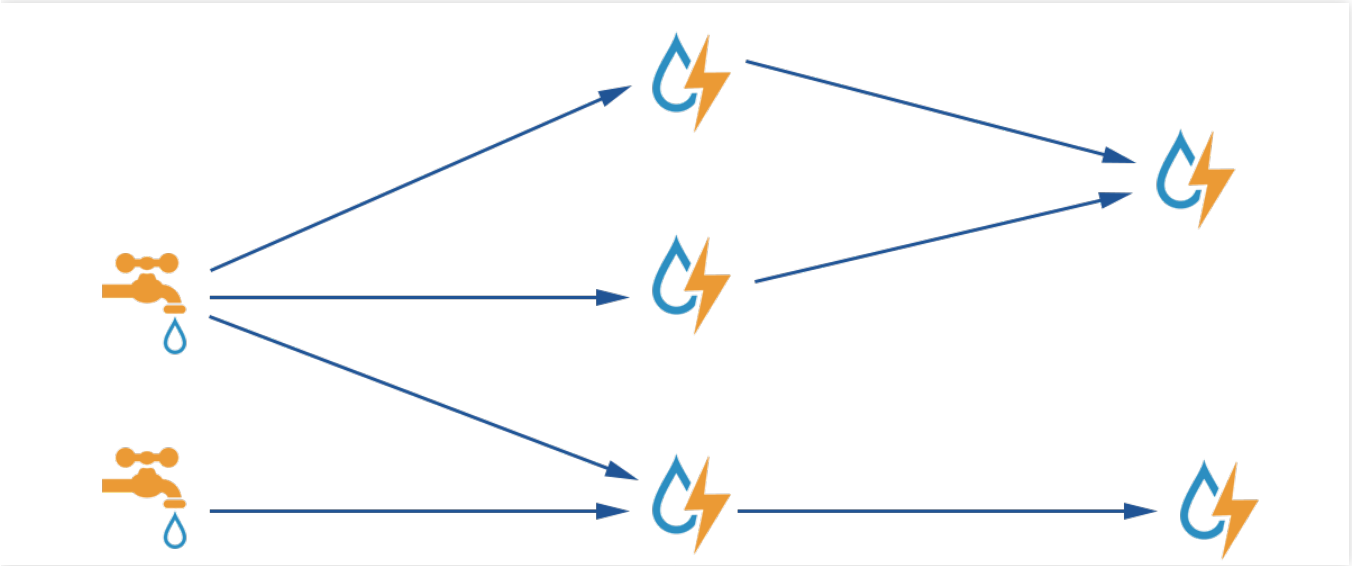
Storm接入CKafka

Storm 是一个分布式实时计算框架，能够对数据进行流式处理和提供通用性分布式 RPC 调用，可以实现处理事件亚秒级的延迟，适用于对延迟要求比较高的实时数据处理场景。

Storm 工作原理

在 Storm 的集群中有两种节点，控制节点 Master Node 和工作节点 Worker Node 。 Master Node 上运行 Nimbus 进程，用于资源分配与状态监控。 Worker Node 上运行 Supervisor 进程，监听工作任务，启动 executor 执行。整个 Storm 集群依赖 zookeeper 负责公共数据存放、集群状态监听、任务分配等功能。

用户提交给 Storm 的数据处理程序称为 topology ，它处理的最小消息单位是 tuple ，一个任意对象的数组。 topology 由 spout 和 bolt 构成， spout 是产生 tuple 的源头， bolt 可以订阅任意 spout 或 bolt 发出的 tuple 进行处理。



Storm with CKafka

Storm 可以把 CKafka 作为 spout ，消费数据进行处理；也可以作为 bolt ，存放经过处理后的数据提供给其它组件消费。

测试环境

Centos6.8系统

package	version
maven	3.5.0
storm	2.1.0

package	version
ssh	5.3
Java	1.8

前提条件

- 下载并安装JDK 8。具体操作，请参见 Download JDK 8。
- 下载并安装Storm，参考Apache Storm downloads。
- 已创建 CKafka 实例。

操作步骤

步骤1. 获取 CKafka 实例接入地址

1. 登录 [CKafka 控制台] 。
2. 在左侧导航栏选择【实例列表】，单击实例的“ID”，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



步骤2. 创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个 Topic。



步骤3. 添加 Maven 依赖

pom.xml 配置如下：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>storm</groupId>
  <artifactId>storm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>storm</name>
    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
      <dependency>
          <groupId>org.apache.storm</groupId>
          <artifactId>storm-core</artifactId>
          <version>2.1.0</version>
      </dependency>
      <dependency>
          <groupId>org.apache.storm</groupId>
          <artifactId>storm-kafka-client</artifactId>
          <version>2.1.0</version>
      </dependency>
      <dependency>
          <groupId>org.apache.kafka</groupId>
          <artifactId>kafka_2.11</artifactId>
          <version>0.10.2.1</version>
          <exclusions>
              <exclusion>
                  <groupId>org.slf4j</groupId>
                  <artifactId>slf4j-log4j12</artifactId>
              </exclusion>
          </exclusions>
      </dependency>
      <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.12</version>
          <scope>test</scope>
      </dependency>
  </dependencies>

  <build>
      <plugins>
          <plugin>
              <artifactId>maven-assembly-plugin</artifactId>
              <configuration>
                  <descriptorRefs>
```

```
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
          <archive>
            <manifest>
              <mainClass>ExclamationTopology</mainClass>
            </manifest>
          </archive>
        </configuration>
        <executions>
          <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

步骤4. 生产消息

使用 spout/bolt

topology 代码：

```
//TopologyKafkaProducerSpout.java
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.bolt.KafkaBolt;
import org.apache.storm.kafka.bolt.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.bolt.selector.DefaultTopicSelector;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.utils.Utils;

import java.util.Properties;

public class TopologyKafkaProducerSpout {
```

```
    //申请的ckafka实例ip:port
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    //指定要将消息写入的topic
    private final static String TOPIC = "storm_test";
    public static void main(String[] args) throws Exception {
        //设置producer属性
        //函数参考：https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html
        //属性参考：http://kafka.apache.org/0102/documentation.html
        Properties properties = new Properties();
        properties.put("bootstrap.servers", BOOTSTRAP_SERVERS);
        properties.put("acks", "1");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        //创建写入kafka的bolt，默认使用fields("key" "message")作为生产消息的key和message，也可以在FieldNameBasedTupleToKafkaMapper()中指定
        KafkaBolt kafkaBolt = new KafkaBolt()
                .withProducerProperties(properties)
                .withTopicSelector(new DefaultTopicSelector(TOPIC))
                .withTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper());
        TopologyBuilder builder = new TopologyBuilder();
        //一个顺序生成消息的spout类，输出field是sentence
        SerialSentenceSpout spout = new SerialSentenceSpout();
        AddMessageKeyBolt bolt = new AddMessageKeyBolt();
        builder.setSpout("kafka-spout", spout, 1);
        //为tuple加上生产到ckafka所需要的fields
        builder.setBolt("add-key", bolt, 1).shuffleGrouping("kafka-spout");
        //写入ckafka
        builder.setBolt("sendToKafka", kafkaBolt, 8).shuffleGrouping("add-key");

        Config config = new Config();
        if (args != null && args.length > 0) {
            //集群模式，用于打包jar，并放到storm运行
            config.setNumWorkers(1);
            StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.createTopology());
        } else {
            //本地模式
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", config, builder.createTopology());
            Utils.sleep(10000);
            cluster.killTopology("test");
            cluster.shutdown();
        }

    }
}
```

创建一个顺序生成消息的 spout 类：

```
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.Map;
import java.util.UUID;

public class SerialSentenceSpout extends BaseRichSpout {

    private SpoutOutputCollector spoutOutputCollector;

    @Override
    public void open(Map map, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollect
or) {
        this.spoutOutputCollector = spoutOutputCollector;
    }

    @Override
    public void nextTuple() {
        Utils.sleep(1000);
        //生产一个UUID字符串发送给下一个组件
        spoutOutputCollector.emit(new Values(UUID.randomUUID().toString()));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("sentence"));
    }
}
```

为 tuple 加上 key、message 两个字段，当 key 为 null 时，生产的消息均匀分配到各个 partition，指定了 key 后将按照 key 值 hash 到特定 partition 上：

```
//AddMessageKeyBolt.java
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class AddMessageKeyBolt extends BaseBasicBolt {

    @Override
```

```
    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        //取出第一个filed值
        String messae = tuple.getString(0);
//        System.out.println(messae);
        //发送给下一个组件
        basicOutputCollector.emit(new Values(null, messae));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        //创建发送给下一个组件的schema
        outputFieldsDeclarer.declare(new Fields("key", "message"));
    }
}
```

使用 trident

使用 trident 类生成 topology

```
//TopologyKafkaProducerTrident.java
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.trident.TridentKafkaStateFactory;
import org.apache.storm.kafka.trident.TridentKafkaStateUpdater;
import org.apache.storm.kafka.trident.mapper.FieldNameBasedTupleToKafkaMapper;
import org.apache.storm.kafka.trident.selector.DefaultTopicSelector;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.tuple.TridentTuple;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.Properties;

public class TopologyKafkaProducerTrident {
    //申请的ckafka实例ip:port
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    //指定要将消息写入的topic
    private final static String TOPIC = "storm_test";
    public static void main(String[] args) throws Exception {
        //设置producer属性
        //函数参考：https://kafka.apache.org/0100/javadoc/index.html?org/apache/kafka/clients/consumer/Kaf
kaConsumer.html
        //属性参考：http://kafka.apache.org/0102/documentation.html
        Properties properties = new Properties();
        properties.put("bootstrap.servers", BOOTSTRAP_SERVERS);
```

```
        properties.put("acks", "1");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        //设置Trident
        TridentKafkaStateFactory stateFactory = new TridentKafkaStateFactory()
                .withProducerProperties(properties)
                .withKafkaTopicSelector(new DefaultTopicSelector(TOPIC))
                //设置使用fields("key", "value")作为消息写入  不像FieldNameBasedTupleToKafkaMapper有默认值
                .withTridentTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper("key", "value"));
        TridentTopology builder = new TridentTopology();
        //一个批量产生句子的spout,输出field为sentence
        builder.newStream("kafka-spout", new TridentSerialSentenceSpout(5))
                .each(new Fields("sentence"), new AddMessageKey(), new Fields("key", "value"))
                .partitionPersist(stateFactory, new Fields("key", "value"), new TridentKafkaStateUpdater(), new Fields
());

        Config config = new Config();
        if (args != null && args.length > 0) {
            //集群模式，用于打包jar，并放到storm运行
            config.setNumWorkers(1);
            StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.build());
        } else {
            //本地模式
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", config, builder.build());
            Utils.sleep(10000);
            cluster.killTopology("test");
            cluster.shutdown();
        }

    }

    private static class AddMessageKey extends BaseFunction {

        @Override
        public void execute(TridentTuple tridentTuple, TridentCollector tridentCollector) {
            //取出第一个filed值
            String messae = tridentTuple.getString(0);
            //System.out.println(messae);
            //发送给下一个组件
            //tridentCollector.emit(new Values(Integer.toString(messae.hashCode()), messae));
            tridentCollector.emit(new Values(null, messae));
        }
    }
}
```

创建一个批量生成消息的 spout 类：

```
//TridentSerialSentenceSpout.java
```

```
import org.apache.storm.Config;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.spout.IBatchSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.Map;
import java.util.UUID;

public class TridentSerialSentenceSpout implements IBatchSpout {

    private final int batchCount;

    public TridentSerialSentenceSpout(int batchCount) {
        this.batchCount = batchCount;
    }

    @Override
    public void open(Map map, TopologyContext topologyContext) {

    }

    @Override
    public void emitBatch(long l, TridentCollector tridentCollector) {
        Utils.sleep(1000);
        for(int i = 0; i < batchCount; i++){
            tridentCollector.emit(new Values(UUID.randomUUID().toString()));
        }
    }

    @Override
    public void ack(long l) {

    }

    @Override
    public void close() {

    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
        Config conf = new Config();
        conf.setMaxTaskParallelism(1);
        return conf;
    }
```

```
    @Override
    public Fields getOutputFields() {
        return new Fields("sentence");
    }
}
```

步骤5. 消费消息

使用 spout/bolt

```
//TopologyKafkaConsumerSpout.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.kafka.spout.*;
import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.HashMap;
import java.util.Map;

import static org.apache.storm.kafka.spout.FirstPollOffsetStrategy.LATEST;

public class TopologyKafkaConsumerSpout {
    //申请的ckafka实例ip:port
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    //指定要将消息写入的topic
    private final static String TOPIC = "storm_test";

    public static void main(String[] args) throws Exception {
        //设置重试策略
        KafkaSpoutRetryService kafkaSpoutRetryService = new KafkaSpoutRetryExponentialBackoff(
                KafkaSpoutRetryExponentialBackoff.TimeInterval.microSeconds(500),
                KafkaSpoutRetryExponentialBackoff.TimeInterval.milliSeconds(2),
                Integer.MAX_VALUE,
                KafkaSpoutRetryExponentialBackoff.TimeInterval.seconds(10)
        );
        ByTopicRecordTranslator<String, String> trans = new ByTopicRecordTranslator<>(
                (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
                new Fields("topic", "partition", "offset", "key", "value"));
        //设置consumer参数
```

```
        //函数参考http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spout/KafkaSpoutConfig.Builder.html
        //参数参考http://kafka.apache.org/0102/documentation.html
        KafkaSpoutConfig spoutConfig = KafkaSpoutConfig.builder(BOOTSTRAP_SERVERS, TOPIC)
                .setProp(new HashMap<String, Object>(){{
                    put(ConsumerConfig.GROUP_ID_CONFIG, "test-group1"); //设置group
                    put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000"); //设置session超时
                    put(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000"); //设置请求超时
                }})
                .setOffsetCommitPeriodMs(10_000) //设置自动确认时间
                .setFirstPollOffsetStrategy(LATEST) //设置拉取最新消息
                .setRetry(kafkaSpoutRetryService)
                .setRecordTranslator(trans)
                .build();

        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("kafka-spout", new KafkaSpout(spoutConfig), 1);
        builder.setBolt("bolt", new BaseRichBolt(){
            private OutputCollector outputCollector;
            @Override
            public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

            }

            @Override
            public void prepare(Map map, TopologyContext topologyContext, OutputCollector outputCollector) {
                this.outputCollector = outputCollector;
            }

            @Override
            public void execute(Tuple tuple) {
                System.out.println(tuple.getStringByField("value"));
                outputCollector.ack(tuple);
            }
        }, 1).shuffleGrouping("kafka-spout");

        Config config = new Config();
        config.setMaxSpoutPending(20);
        if (args != null && args.length > 0) {
            config.setNumWorkers(3);
            StormSubmitter.submitTopologyWithProgressBar(args[0], config, builder.createTopology());
        }
        else {
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", config, builder.createTopology());
            Utils.sleep(20000);
            cluster.killTopology("test");
            cluster.shutdown();
        }
```

```
    }
}
```

使用 trident

```
//TopologyKafkaConsumerTrident.java
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.generated.StormTopology;
import org.apache.storm.kafka.spout.ByTopicRecordTranslator;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutConfig;
import org.apache.storm.kafka.spout.trident.KafkaTridentSpoutOpaque;
import org.apache.storm.trident.Stream;
import org.apache.storm.trident.TridentTopology;
import org.apache.storm.trident.operation.BaseFunction;
import org.apache.storm.trident.operation.TridentCollector;
import org.apache.storm.trident.tuple.TridentTuple;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;

import java.util.HashMap;

import static org.apache.storm.kafka.spout.FirstPollOffsetStrategy.LATEST;

public class TopologyKafkaConsumerTrident {
    //申请的ckafka实例ip:port
    private final static String BOOTSTRAP_SERVERS = "xx.xx.xx.xx:xxxx";
    //指定要将消息写入的topic
    private final static String TOPIC = "storm_test";

    public static void main(String[] args) throws Exception {
        ByTopicRecordTranslator<String, String> trans = new ByTopicRecordTranslator<>(
                (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
                new Fields("topic", "partition", "offset", "key", "value"));
        //设置consumer参数
        //函数参考http://storm.apache.org/releases/1.1.0/javadocs/org/apache/storm/kafka/spout/KafkaSpoutC
onfig.Builder.html
        //参数参考http://kafka.apache.org/0102/documentation.html
        KafkaTridentSpoutConfig spoutConfig = KafkaTridentSpoutConfig.builder(BOOTSTRAP_SERVERS, TOPI
C)
                .setProp(new HashMap<String, Object>(){{
                    put(ConsumerConfig.GROUP_ID_CONFIG, "test-group1"); //设置group
                    put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true"); //设置自动确认
                    put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "50000"); //设置session超时
                    put(ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG, "60000"); //设置请求超时
                }})
```

```
                .setFirstPollOffsetStrategy(LATEST) //设置拉取最新消息
                .setRecordTranslator(trans)
                .build();

        TridentTopology builder = new TridentTopology();
//      Stream spoutStream = builder.newStream("spout", new KafkaTridentSpoutTransactional(spoutConfig));
//事务型
        Stream spoutStream = builder.newStream("spout", new KafkaTridentSpoutOpaque(spoutConfig));
        spoutStream.each(spoutStream.getOutputFields(), new BaseFunction(){
            @Override
            public void execute(TridentTuple tridentTuple, TridentCollector tridentCollector) {
                System.out.println(tridentTuple.getStringByField("value"));
                tridentCollector.emit(new Values(tridentTuple.getStringByField("value")));
            }
        }, new Fields("message"));

        Config conf = new Config();
        conf.setMaxSpoutPending(20);conf.setNumWorkers(1);
        if (args != null && args.length > 0) {
            conf.setNumWorkers(3);
            StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.build());
        }
        else {
            StormTopology stormTopology = builder.build();
            LocalCluster cluster = new LocalCluster();
            cluster.submitTopology("test", conf, stormTopology);
            Utils.sleep(10000);
            cluster.killTopology("test");
            cluster.shutdown();stormTopology.clear();
        }
    }
}
```

步骤6. 提交 Storm

使用 mvn package 编译后，可以提交到本地集群进行 debug 测试，也可以提交到正式集群进行运行。

```
storm jar your_jar_name.jar topology_name
```

```
storm jar your_jar_name.jar topology_name tast_name
```

Spark Streaming接入CKafka

Spark Streaming 是 Spark Core 的一个扩展，用于高吞吐且容错地处理持续性的数据，目前支持的外部输入有 Kafka、Flume、HDFS/S3、Kinesis、Twitter 和 TCP socket。



Spark Streaming 将连续数据抽象成 DStream（Discretized Stream），而 DStream 由一系列连续的 RDD（弹性分布式数据集）组成，每个 RDD 是一定时间间隔内产生的数据。使用函数对 DStream 进行处理其实即为对这些 RDD 进行处理。



使用 Spark Streaming 作为 Kafka 的数据输入时，可支持 Kafka 稳定版本与实验版本：

Kafka Version	spark-streaming-kafka-0.8	spark-streaming-kafka-0.10
Broker Version	0.8.2.1 or higher	0.10.0 or higher
Api Maturity	Deprecated	Stable
Language Support	Scala、Java、Python	Scala、Java
Receiver DStream	Yes	No
Direct DStream	Yes	Yes
SSL / TLS Support	No	Yes
Offset Commit Api	No	Yes
Dynamic Topic Subscription	No	Yes

目前 CKafka 兼容 0.9及以上的版本，本次实践使用 0.10.2.1 版本的 Kafka 依赖。
此外，EMR 中的 Spark Streaming 也支持直接对接 CKafka，详见 [SparkStreaming 对接 CKafka 服务] 。

操作步骤

步骤1. 获取 CKafka 实例接入地址

1. 登录 [CKafka 控制台] 。
2. 在左侧导航栏选择【实例列表】，单击实例的“ID”，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址，接入地址是生产消费需要用到的 bootstrap-server。



步骤2. 创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个名为test的 Topic，接下来将以该 Topic 为例介绍如何生产消费。



步骤3. 准备云服务器环境

Centos6.8 系统

package	version
sbt	0.13.16
hadoop	2.7.3

package	version
spark	2.1.0
protobuf	2.5.0
ssh	CentOS 默认安装
Java	1.8

具体安装步骤参考配置环境。

步骤4. 对接CKafka

向CKafka中生产消息

目前 CKafka 支持 0.9.0.x、0.10.0.x、0.10.1.x、0.10.2.x 版本。这里使用 0.10.2.1 版本的 Kafka 依赖。

1. 在 build.sbt 添加依赖：

```
name := "Producer Example"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % "0.10.2.1"
```

2. 配置 producer_example.scala ：

```
import java.util.Properties
import org.apache.kafka.clients.producer._
object ProducerExample extends App {
val props = new Properties()
props.put("bootstrap.servers", "172.16.16.12:9092") //实例信息中的内网 IP 与端口

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

val producer = new KafkaProducer[String, String](props)
val TOPIC="test"  //指定要生产的 Topic
for(i<- 1 to 50){
val record = new ProducerRecord(TOPIC, "key", s"hello $i") //生产 key 是"key",value 是 hello i 的消息
producer.send(record)
}
val record = new ProducerRecord(TOPIC, "key", "the end "+new java.util.Date)
producer.send(record)
producer.close() //最后要断开
}
```

更多有关 ProducerRecord 的用法请参考 [ProducerRecord] 文档。

从CKafka消费消息

DirectStream

1. 在 build.sbt 添加依赖：

```
name := "Consumer Example"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-10" % "2.1.0"
```

2. 配置 DirectStream_example.scala ：

```
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.kafka.common.TopicPartition
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010.OffsetRange
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import collection.JavaConversions._
import Array._
object Kafka {
  def main(args: Array[String]) {
    val kafkaParams = Map[String, Object](
      "bootstrap.servers" -> "172.16.16.12:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[StringDeserializer],
      "group.id" -> "spark_stream_test1",
      "auto.offset.reset" -> "earliest",
      "enable.auto.commit" -> "false"
    )

    val sparkConf = new SparkConf()
    sparkConf.setMaster("local")
    sparkConf.setAppName("Kafka")
    val ssc = new StreamingContext(sparkConf, Seconds(5))
    val topics = Array("spark_test")

    val offsets : Map[TopicPartition, Long] = Map()

    for (i <- 0 until 3){
      val tp = new TopicPartition("spark_test", i)
      offsets.updated(tp , 0L)
```

```
        }
        val stream = KafkaUtils.createDirectStream[String, String](
            ssc,
            PreferConsistent,
            Subscribe[String, String](topics, kafkaParams)
        )
        println("directStream")
        stream.foreachRDD{ rdd=>
            //输出获得的消息
            rdd.foreach{iter =>
                val i = iter.value
                println(s"${i}")
            }
            //获得offset
            val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
            rdd.foreachPartition { iter =>
                val o: OffsetRange = offsetRanges(TaskContext.get.partitionId)
                println(s"${o.topic} ${o.partition} ${o.fromOffset} ${o.untilOffset}")
            }
        }

        // Start the computation
        ssc.start()
        ssc.awaitTermination()
    }
}
```

RDD

1. 配置 build.sbt （配置同上，单击查看）。
2. 配置 RDD_example ：

```
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010.OffsetRange
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import collection.JavaConversions._
import Array._
object Kafka {
    def main(args: Array[String]) {
        val kafkaParams = Map[String, Object](
            "bootstrap.servers" -> "172.16.16.12:9092",
```

```
            "key.deserializer" -> classOf[StringDeserializer],
            "value.deserializer" -> classOf[StringDeserializer],
            "group.id" -> "spark_stream",
            "auto.offset.reset" -> "earliest",
            "enable.auto.commit" -> (false: java.lang.Boolean)
        )
        val sc = new SparkContext("local", "Kafka", new SparkConf())
        val java_kafkaParams : java.util.Map[String, Object] = kafkaParams
        //按顺序向 parition 拉取相应 offset 范围的消息，如果拉取不到则阻塞直到超过等待时间或者新生产消息达到拉取的数量
        val offsetRanges = Array[OffsetRange](
            OffsetRange("spark_test", 0, 0, 5),
            OffsetRange("spark_test", 1, 0, 5),
            OffsetRange("spark_test", 2, 0, 5)
        )
        val range = KafkaUtils.createRDD[String, String](
            sc,
            java_kafkaParams,
            offsetRanges,
            PreferConsistent
        )
        range.foreach(rdd=>println(rdd.value))
        sc.stop()
    }
}
```

更多 kafkaParams 用法参考 kafkaParams 文档。

配置环境

安装 sbt

1. 在 sbt 官网 上下载 sbt 包。
2. 解压后在 sbt 的目录下创建一个 sbt_run.sh 脚本并增加可执行权限，脚本内容如下：

```
#!/bin/bash
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256M"
java $SBT_OPTS -jar `dirname $0`/bin/sbt-launch.jar "$@"
```

```
chmod u+x ./sbt_run.sh
```

3. 执行以下命令。

```
./sbt-run.sh sbt-version
```

若能看到 sbt 版本说明可以正常运行。

安装 protobuf

1. 下载 protobuf 相应版本。
2. 解压后进入目录。

```
./configure
make && make install
```

需要预先安装 gcc-g++，执行中可能需要 root 权限。

3. 重新登录，在命令行中输入下述内容。

```
protoc --version
```

4. 若能看到 protobuf 版本说明可以正常运行。

安装 Hadoop

1. 访问 Hadoop 官网 下载所需要的版本。
2. 增加 Hadoop 用户。

```
useradd -m hadoop -s /bin/bash
```

3. 增加管理员权限。

```
visudo
```

4. 在 root ALL=(ALL) ALL 下增加一行。
   hadoop ALL=(ALL) ALL
   保存退出。
5. 使用 Hadoop 进行操作。

```
su hadoop
```

6. SSH 无密码登录。

```
cd ~/.ssh/                    # 若没有该目录，请先执行一次ssh localhost
ssh-keygen -t rsa             # 会有提示，都按回车就可以
cat id_rsa.pub >> authorized_keys  # 加入授权
chmod 600 ./authorized_keys    # 修改文件权限
```

7. 安装 Java。

```
sudo yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

8. 配置 ${JAVA_HOME}。

```
vim /etc/profile
```

在文末加上下述内容：

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el6_8.x86_64/jre
export PATH=$PATH:$JAVA_HOME
```

根据安装情况修改对应路径。

9. 解压 Hadoop，进入目录。

```
./bin/hadoop version
```

若能显示版本信息说明能正常运行。

10. 配置单机伪分布式（可根据需要搭建不同形式的集群）。

```
vim /etc/profile
```

在文末加上下述内容：

```
export HADOOP_HOME=/usr/local/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

根据安装情况修改对应路径。

11. 修改 /etc/hadoop/core-site.xml 。

```
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>file:/usr/local/hadoop/tmp</value>
        <description>Abase for other temporary directories.</description>
    </property>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

12. 修改 /etc/hadoop/hdfs-site.xml 。

```
<configuration>
```

```
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/usr/local/hadoop/tmp/dfs/name</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/local/hadoop/tmp/dfs/data</value>
    </property>
</configuration>
```

13. 修改 /etc/hadoop/hadoop-env.sh 中的 JAVA_HOME 为Java 的路径。

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-0.b13.el6_8.x86_64/jre
```

14. 执行 NameNode 格式化。

```
./bin/hdfs namenode -format
```

显示 Exitting with status 0 则表示成功。
15. 启动 Hadoop。

```
./sbin/start-dfs.sh
```

成功启动会存在 NameNode 进程， DataNode 进程， SecondaryNameNode 进程。

安装 Spark

访问 Spark 官网 下载所需要的版本。
因为之前安装了 Hadoop，所以选择使用 *Pre-build with user-provided Apache Hadoop*。
本示例同样使用 hadoop 用户进行操作。

1. 解压进入目录。
2. 修改配置文件。

```
cp ./conf/spark-env.sh.template ./conf/spark-env.sh
vim ./conf/spark-env.sh
```

在第一行添加下述内容：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

3. 根据 hadoop 安装情况修改路径。

运行示例。

```
bin/run-example SparkPi
```

若成功安装可以看到程序输出 π 的近似值。

生产消费最佳实践

本文主要介绍消息队列 CKafka 生产和消费消息的最佳实践，帮助您减少消费消息出错的可能性。

生产消息

Topic 使用推荐

- 配置要求：推荐3副本，同步复制，最小同步副本数为2，且同步副本数不能等于 Topic 副本数，否则宕机1个副本会导致无法生产消息。
- 创建方式：支持选择是否开启 CKafka 自动创建 Topic 的开关。选择开启后，表示生产或消费一个未创建的 Topic 时，会自动创建一个包含3个分区和3个副本的 Topic。
- 单 Topic 最大分区数建议为100。
- Topic 副本数为3（当前版本限制，不可调整）。

失败重试

分布式环境下，由于网络等原因，消息偶尔会出现发送失败的情况，其原因可能是消息已经发送成功但是ACK机制失败或者消息确实没有发送成功。
您可以根据业务需求，设置以下重试参数：

参数	说明
retries	重试次数，默认值为 3 ，就对于数据丢失零容忍的应用而言，请考虑设置为 Integer.MAX_VALUE（有效且最大）。
retry.backoff.ms	重试间隔，建议设置为 1000 。

这样将能够应对 Broker 的 Leader 分区出现无法立刻响应 Producer 请求的情况。

异步发送

发送接口是异步的，如果您想接收发送的结果，可以调用 metadataFuture.get(timeout, TimeUnit.MILLISECONDS)。

一个Producer对应一个应用

Producer 是线程安全的，且可以往任何 Topic 发送消息。通常情况下，建议一个应用对应一个 Producer。

Acks

Kafka 的 ACK 机制，指 Producer 的消息发送确认机制，在 Kafka 的 0.10.x 版本上，其设置值是 Acks，而在 0.8.x 版本上，则为 request.required.acks，Acks 的设置将直接影响到 Kafka 集群的吞吐量和消息可靠性。

Acks 的参数说明如下：

参数	说明
acks=0	无需服务端的 Response。性能较高、丢数据风险较大。
acks=1	服务端主节点写成功即返回 Response。性能中等、丢数据风险中等、主节点宕机可能导致数据丢失。
acks=all	服务端主节点写成功且备节点同步成功才返回 Response。性能较差、数据较为安全、主节点和备节点都宕机才会导致数据丢失。

一般建议选择 acks=1 ，重要的服务可以设置 acks=all 。

Batch

一般情况下，消息队列 CKafka 的 Topic 会有多个分区，Producer 客户端在向服务端发送消息时，需要先确认往哪个 Topic 的哪个分区发送。在给同一个分区发送多条消息时，Producer 客户端会将相关消息打包成一个 Batch，批量发送到服务端。Producer 客户端在处理 Batch 时，是有额外开销的。一般情况下，小 Batch 会导致 Producer 客户端产生大量请求，造成请求队列在客户端和服务端的排队，并造成相关机器的 CPU 升高，从而整体推高了消息发送和消费延迟。一个合适的 Batch 大小，可以减少发送消息时客户端向服务端发起的请求次数，在整体上提高消息发送的吞吐和延迟。

Batch 参数说明如下：

参数	说明
batch.size	发往每个分区（Partition）的消息缓存量（消息内容的字节数之和，不是条数）。达到设置的数值时，就会触发一次网络请求，然后 Producer 客户端把消息批量发往服务器。
linger.ms	每条消息在缓存中的最长时间。若超过这个时间，Producer 客户端就会忽略 batch.size 的限制，立即把消息发往服务器。
buffer.memory	所有缓存消息的总体大小超过这个数值后，就会触发把消息发往服务器，此时会忽略 batch.size 和 linger.ms 的限制。buffer.memory 的默认数值是 32MB，对于单个 Producer 而言，可以保证足够的性能。

> 说明：
>
> 如果您在同一个 JVM 中启动多个 Producer，那么每个 Producer 都有可能占用32 MB缓存空间，此时便有可能触发 OOM（Out of Memory），此时您需要考虑 buffer.memory 的大小，避免触发 OOM。

您可以根据具体业务需求进行参数设置值的调整。

Key 和 Value

消息队列 CKafka 的消息有 Key（消息标识）和 Value（消息内容）两个字段。

为了便于追踪，请为消息设置一个唯一的 Key。您可以通过 Key 追踪某消息，打印发送日志和消费日志，了解该消息的生产和消费情况。

如果消息发送量较大，建议不要设置 Key，并使用黏性分区策略。

黏性分区

只有发送到相同分区的消息，才会被放到同一个 Batch 中，因此决定一个 Batch 如何形成的一个因素是消息队列 Kafka Producer 端设置的分区策略。消息队列 Kafka Producer 允许通过设置 Partitioner 的实现类来选择适合自己业务的分区。在消息指定 Key 的情况下，消息队列 Kafka Producer 的默认策略是对消息的Key进行哈希，然后根据哈希结果选择分区，保证相同 Key 的消息会发送到同一个分区。

在消息没有指定 Key 的情况下，消息队列 Kafka 2.4版本之前的默认策略是循环使用主题的所有分区，将消息以轮询的方式发送到每一个分区上。但是，这种默认策略 Batch 的效果会比较差，在实际使用中，可能会产生大量的小 Batch，从而使得实际的延迟增加。鉴于该默认策略对无 Key 消息的分区效率低问题，消息队列 Kafka 在2.4版本引入了黏性分区策略（Sticky Partitioning Strategy）。

黏性分区策略主要解决无 Key 消息分散到不同分区，造成小 Batch问题。其主要策略是如果一个分区的 Batch 完成后，就随机选择另一个分区，然后后续的消息尽可能地使用该分区。这种策略在短时间内看，会将消息发送到同一个分区，如果拉长整个运行时间，消息还是可以均匀地发布到各个分区上的。这样可以避免消息出现分区倾斜，同时还可以降低延迟，提升服务整体性能。

如果您使用的消息队列 Kafka Producer 客户端是2.4及以上版本，默认的分区策略就采用黏性分区策略。如果您使用的 Producer 客户端版本小于2.4，可以根据黏性分区策略原理，自行实现分区策略，然后通过参数 partitioner.class 设置指定的分区策略。

关于黏性分区策略实现，您可以参考如下 Java 版代码实现。该代码的实现逻辑主要是根据一定的时间间隔，切换一次分区。

```
public class MyStickyPartitioner implements Partitioner {

    // 记录上一次切换分区时间。
    private long lastPartitionChangeTimeMillis = 0L;
    // 记录当前分区。
    private int currentPartition = -1;
    // 分区切换时间间隔，可以根据实际业务选择切换分区的时间间隔。
    private long partitionChangeTimeGap = 100L;

    public void configure(Map<String, ?> configs) {}

    /**
     * Compute the partition for the given record.
     *
     * @param topic The topic name
     * @param key The key to partition on (or null if no key)
     * @param keyBytes serialized key to partition on (or null if no key)
```

```
     * @param value The value to partition on or null
     * @param valueBytes serialized value to partition on or null
     * @param cluster The current cluster metadata
     */
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster c
luster) {

        // 获取所有分区信息。
        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();

        if (keyBytes == null) {
            List<PartitionInfo> availablePartitions = cluster.availablePartitionsForTopic(topic);
            int availablePartitionSize = availablePartitions.size();

            // 判断当前可用分区。
            if (availablePartitionSize > 0) {
                handlePartitionChange(availablePartitionSize);
                return availablePartitions.get(currentPartition).partition();
            } else {
                handlePartitionChange(numPartitions);
                return currentPartition;
            }
        } else {
            // 对于有key的消息，根据key的哈希值选择分区。
            return Utils.toPositive(Utils.murmur2(keyBytes)) % numPartitions;
        }
    }

    private void handlePartitionChange(int partitionNum) {
        long currentTimeMillis = System.currentTimeMillis();

        // 如果超过分区切换时间间隔，则切换下一个分区，否则还是选择之前的分区。
        if (currentTimeMillis - lastPartitionChangeTimeMillis >= partitionChangeTimeGap
            || currentPartition < 0 || currentPartition >= partitionNum) {
            lastPartitionChangeTimeMillis = currentTimeMillis;
            currentPartition = Utils.toPositive(ThreadLocalRandom.current().nextInt()) % partitionNum;
        }
    }

    public void close() {}

}
```

分区顺序

单个分区（Partition）内，消息是按照发送顺序储存的，是基本有序的。每个主题下面都有若干分区，如果消息被分

配到不同的分区中，不同 Partition 之间不能保证顺序。

如果需要进行消息具有消费顺序性，可以在生产端指定这一类消息的 key，这类消息都用相同的 key 进行消息发送，CKafka 就会根据 key 哈希取模选取其中一个分区进行存储，由于一个分区只能由一个消费者进行监听消费，此时消息就具有消息消费的顺序性了。

消费消息

消费消息基本流程

1. Poll 数据。
2. 执行消费逻辑。
3. 再次 poll 数据。

负载均衡

每个 Consumer Group 可以包含多个 Consumer，并将参数 group.id 设置成相同的值，属于同一个 Consumer Group 的 Consumer 会负载消费订阅的 Topic。

例如：Consumer Group A 订阅了 Topic A，并开启三个消费实例 C1、C2、C3，则发送到 Topic A 的每条消息最终只会传给 C1、C2、C3 的某一个。CKafka 默认会均匀地把消息传给各个消费实例，以做到消费负载均衡。

CKafka 负载均衡的内部原理是：把订阅的Topic的分区，平均分配给各个Consumer。因此，Consumer 的个数不要大于分区的数量，否则会有消费实例分配不到任何分区而处于空跑状态。除了第一次启动上线之外，后续消费实例发生重启、增加、减少等变更时，都会触发一次负载均衡。

订阅关系

同一个 Consumer Group 内，建议各个消费实例订阅的 Topic 保持一致，避免给排查问题带来干扰。

- Consumer Group 订阅多个 Topic。

  一个 Consumer Group 可以订阅多个Topic，多个 Topic 的消息被 Cosumer Group 中的 Consumer 均匀消费。例如 Consumer Group A 订阅了 Topic A、Topic B、Topic C，则这三个 Topic 中的消息，被 Consumer Group 中的 Consumer 均匀消费。

  Consumer Group 订阅多个 Topic 的示例代码如下：

```
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic: topics) {
subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);
```

- Topic 被多个 Consumer Group 订阅。

  一个 Topic 可以被多个 Consumer Group 订阅，且各个 Consumer Group 独立消费Topic下的所有消息。例如 Consumer Group A 订阅了 Topic A，Consumer Group B也订阅了Topic A，则发送到Topic A的每条消息，不仅会传一份给Consumer Group A的消费实例，也会传一份给Consumer Group B的消费实例，且这两个过程相互独立，相互没有任何影响。

一个 Consumer Group 对应一个应用

建议一个 Consumer Group 对应一个应用，即不同的应用对应不同的代码。如果您需要将不同的代码写在同一个应用中，请准备多份不同的 kafka.properties。例如 kafka1.properties、kafka2.properties。

消费位点 Offset

每个 Topic 会有多个分区，每个分区会统计当前消息的总条数，这个称为最大位点 MaxOffset。

消息队列 CKafka 的 Consumer 会按顺序依次消费分区内的每条消息，记录已经消费了的消息条数，称为消费位点 ConsumerOffset。

剩余的未消费的条数（也称为消息堆积量）=MaxOffset-ConsumerOffset。

offset 提交

消息队列 CKafka 的 Consumer 有两个相关参数：

- enable.auto.commit：默认值为true。
- auto.commit.interval.ms： 默认值为1000，即1s。
  这两个参数组合的结果为：每次 poll 数据前会先检查上次提交位点的时间，如果距离当前时间已经超过参数 auto.commit.interval.ms 规定的时长，则客户端会启动位点提交动作。

因此，如果将 enable.auto.commit 设置为 true，则需要在每次 poll 数据时，确保前一次 poll 出来的数据已经消费完毕，否则可能导致位点跳跃。

如果想自己控制位点提交，请把 enable.auto.commit 设为 false，并调用 commit(offsets) 函数自行控制位点提交。

重置 offset

以下两种情况，会发生消费位点重置：

- 当服务端不存在曾经提交过的位点时（例如客户端第一次上线）。
- 当从非法位点拉取消息时（例如某个分区最大位点是10，但客户端却从11开始拉取消息）。
  Java 客户端可以通过 auto.offset.reset 来配置重置策略，主要有三种策略：
- latest：从最大位点开始消费。
- earliest：从最小位点开始消费。
- none：不做任何操作，即不重置。

> ?
>
> - 建议设置成 latest，而不要设置成 earliest，避免因位点非法时从头开始消费，从而造成大量重复。
> - 如果是您自己管理位点，可以设置成 none。

拉取消息

消费过程是由客户端主动去服务端拉取消息的，在拉取大消息时需要控制拉取速度，注意以下参数设置：

- max.poll.records：如果单条消息超过 1MB，建议设置为1。
- max.partition.fetch.bytes：设置比单条消息的大小略大一点。
- fetch.max.bytes：设置比单条消息的大小略大一点。
  通过公网消费消息时，通常会因为公网带宽的限制导致连接被断开，此时需要注意控制拉取速度，注意以下参数设置：
- fetch.max.bytes：建议设置成公网带宽的一半（注意该参数的单位是 bytes，公网带宽的单位是 bits）
- max.partition.fetch.bytes：建议设置成 fetch.max.bytes 的三分之一或者四分之一。

消息重复和消费幂等

消息队列 CKafka 消费的语义是 at least once， 也就是至少投递一次，保证消息不丢失，但是无法保证消息不重复。在出现网络问题、客户端重启时均有可能造成少量重复消息，此时应用消费端如果对消息重复比较敏感（例如订单交易类），则应该做消息幂等。

以数据库类应用为例，常用做法为：

- 发送消息时，传入 key 作为唯一流水号 ID。
- 消费消息时，判断 key 是否已经消费过，如果已经消费过了，则忽略，如果没消费过，则消费一次。
  当然，如果应用本身对少量消息重复不敏感，则不需要做此类幂等检查。

消费失败

消息队列 CKafka 是按分区逐条消息顺序向前推进消费的，如果消费端拿到某条消息后执行消费逻辑失败，例如应用服务器出现了脏数据，导致某条消息处理失败，等待人工干预，那么有以下两种处理方式：

- 失败后一直尝试再次执行消费逻辑。这种方式有可能造成消费线程阻塞在当前消息，无法向前推进，造成消息堆积。
- 由于消息队列 CKafka 没有处理失败消息的设计，实践中通常会打印失败的消息或者存储到某个服务（例如创建一个 Topic 专门用来放失败的消息），然后定时检查失败消息的情况，分析失败原因，根据情况处理。

消费延迟

消费过程是由客户端主动去服务端拉取消息。一般情况下，如果客户端能够及时消费，则不会产生较大延迟。若产生了较大延迟，请先关注是否有堆积，并注意提高消费速度。

消费堆积

通常造成消息堆积的原因是：

- 消费速度跟不上生产速度，此时应该提高消费速度。
- 消费端产生了阻塞。
  消费端拿到消息后，执行消费逻辑，通常会执行一些远程调用，如果这个时候同步等待结果，则有可能造成一直等待，消费进程无法向前推进。

消费端应该尽量避免堵塞消费线程，如果存在等待调用结果的情况，建议设置等待的超时时间，超时后作为消费失败进行处理。

提高消费速度

- 增加Consumer实例个数。
  可以在进程内直接增加（需要保证每个实例对应一个线程），也可以部署多个消费实例进程。

  > 说明：
  >
  > 实例个数超过分区数量后就不再能提高速度，将会有消费实例不工作。

- 增加消费线程。

1. 定义一个线程池。
2. Poll数据。

3. 把数据提交到线程池进行并发处理。
4. 等并发结果返回成功后，再次poll数据执行。

套接字缓冲区（socket buffers）

在 Kafka 的 0.10.x 版本中，参数 receive.buffer.bytes 的默认值为 64KB。而在 Kafka 的 0.8.x 版本中，参数 socket.receive.buffer.bytes 的默认值为 100KB。

这两个默认值对于高吞吐量的环境而言都太小了，特别是如果 Broker 和 Consumer 之间的网络带宽延迟积（bandwidth-delay product）大于局域网（local areanetwork，LAN）时。

对于延迟为1ms或更多的高带宽的网络（如 10Gbps 或更高），建议将套接字缓冲区设置为8或16MB。

如果您的内存不足，也至少考虑设置为 1MB。您也可以设置为 -1，它会让底层操作系统根据网络的实际情况，去调整缓冲区的大小。

但是，对于需要启动"热"分区的 Consumers 来说，自动调整可能不会那么快。

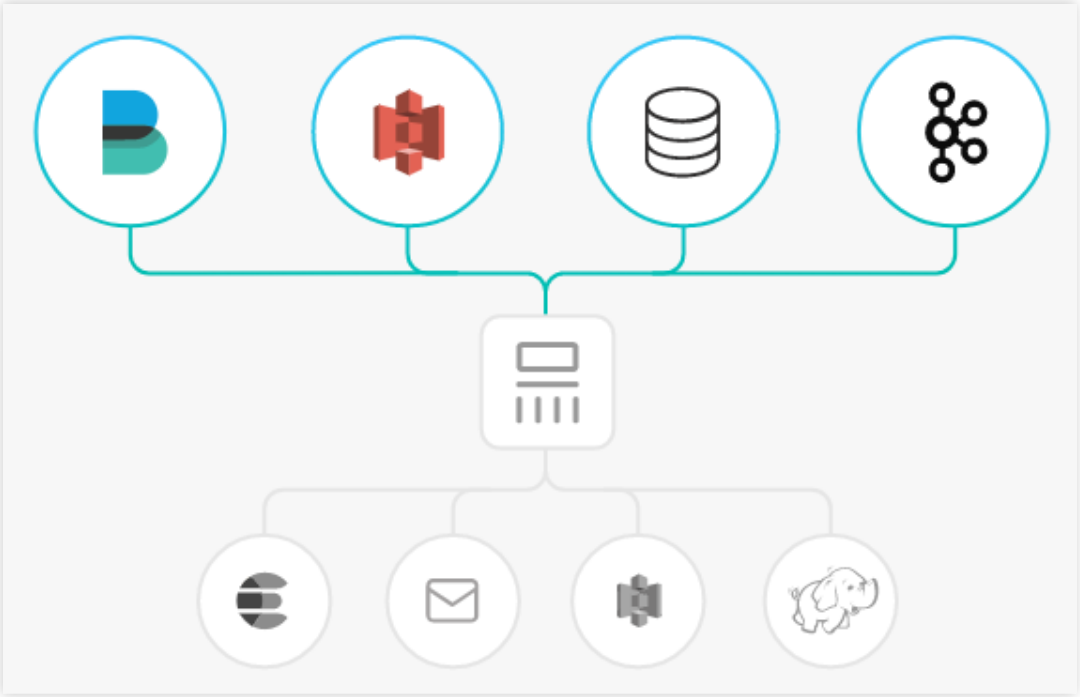
Logstash接入CKafka

Logstash 是一个开源的日志处理工具，可以从多个源头收集数据、过滤收集的数据并对数据进行存储作为其他用途。Logstash 灵活性强，拥有强大的语法分析功能，插件丰富，支持多种输入和输出源。Logstash 作为水平可伸缩的数据管道，与 Elasticsearch 和 Kibana 配合，在日志收集检索方面功能强大。

Logstash 工作原理

Logstash 数据处理可以分为三个阶段：inputs → filters → outputs。

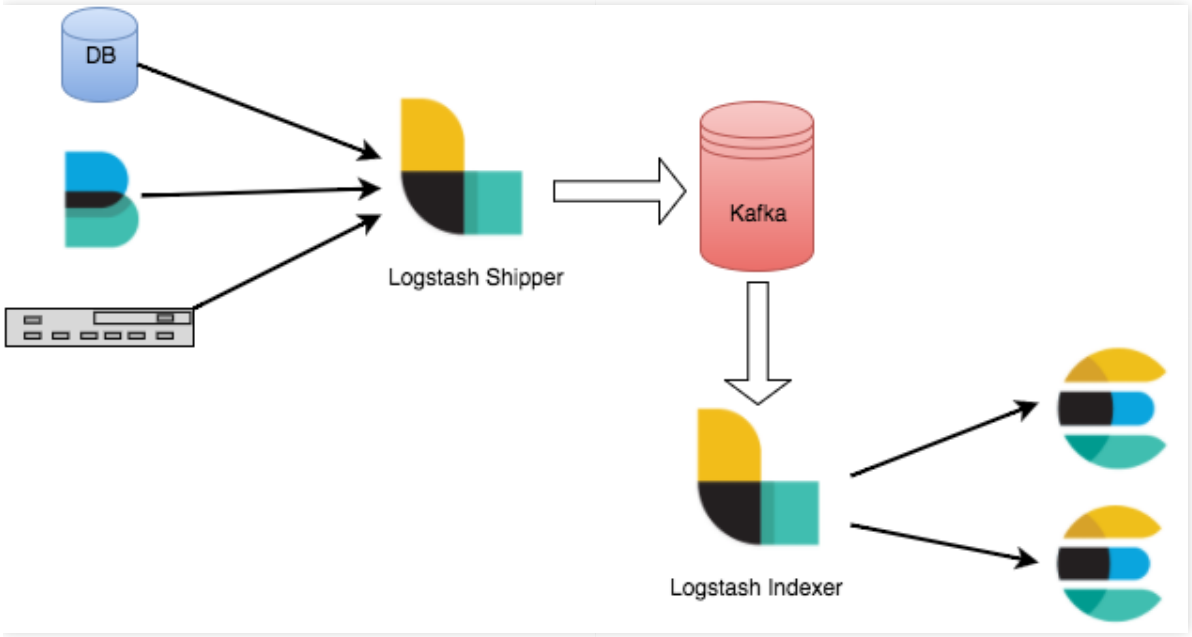
1. inputs：产生数据来源，例如文件、syslog、redis 和 beats 此类来源。
2. filters：修改过滤数据， 在 Logstash 数据管道中属于中间环节，可以根据条件去对事件进行更改。一些常见的过滤器包括：grok、mutate、drop 和 clone 等。
3. outputs：将数据传输到其他地方，一个事件可以传输到多个 outputs，当传输完成后这个事件就结束。Elasticsearch 就是最常见的 outputs。
   同时 Logstash 支持编码解码，可以在 inputs 和 outputs 端指定格式。



Logstash 接入 Kafka 的优势

- 可以异步处理数据：防止突发流量。
- 解耦：当 Elasticsearch 异常的时候不会影响上游工作。

> 注意：
>
> Logstash 过滤消耗资源，如果部署在生产 server 上会影响其性能。



操作步骤

准备工作

- 下载并安装Logstash，参考 Download Logstash。
- 下载并安装JDK 8，参考 Download JDK 8。
- 已创建 CKafka 实例。

步骤1. 获取 CKafka 实例接入地址

1. 登录 [CKafka 控制台] 。
2. 在左侧导航栏选择【实例列表】，单击实例的"ID"，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



步骤2. 创建 Topic

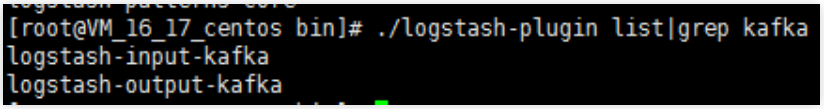
1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个名为logstash_test的 Topic。



步骤3. 接入CKafka

inputs 接入

1. 执行 bin/logstash-plugin list ，查看已经支持的插件是否含有 logstash-input-kafka。



2. 在.bin/目录下编写配置文件 input.conf。此处将标准输出作为数据终点，将 Kafka 作为数据来源。

```
input {
    kafka {
        bootstrap_servers => "xx.xx.xx.xx:xxxx" // ckafka 实例接入地址
        group_id => "logstash_group"  // ckafka groupid 名称
        topics => ["logstash_test"] // ckafka topic 名称
        consumer_threads => 3 // 消费线程数，一般与 ckafka 分区数一致
        auto_offset_reset => "earliest"
    }
}
output {
    stdout{codec=>rubydebug}
}
```

3. 执行以下命令启动 Logstash，进行消息消费。

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

返回结果如下

```
[root@VM_16_17_centos bin]# ./logstash -f input.conf
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
Sending Logstash's logs to /data/ryan/logstash-5.5.2/logs which is now configured via log4j2.properties
[2017-09-06T18:07:41,926][INFO ][logstash.pipeline        ] Starting pipeline {"id"=>"main", "pipeline.workers"=>4, "pipe
[2017-09-06T18:07:41,943][INFO ][logstash.pipeline        ] Pipeline main started
[2017-09-06T18:07:41,999][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:24:23.039Z localhost ckafka"
}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:23:28.343Z localhost logstash"
}
{
    "@timestamp" => 2017-09-06T10:07:42.256Z,
      "@version" => "1",
       "message" => "2017-09-06T09:23:15.848Z localhost test"
}
```

可以看到刚才 Topic 中的数据被消费出来。

outputs 接入

1. 执行 bin/logstash-plugin list ，查看已经支持的插件是否含有 logstash-output-kafka。

```
[root@VM_16_17_centos bin]# ./logstash-plugin list|grep kafka
logstash-input-kafka
logstash-output-kafka
```

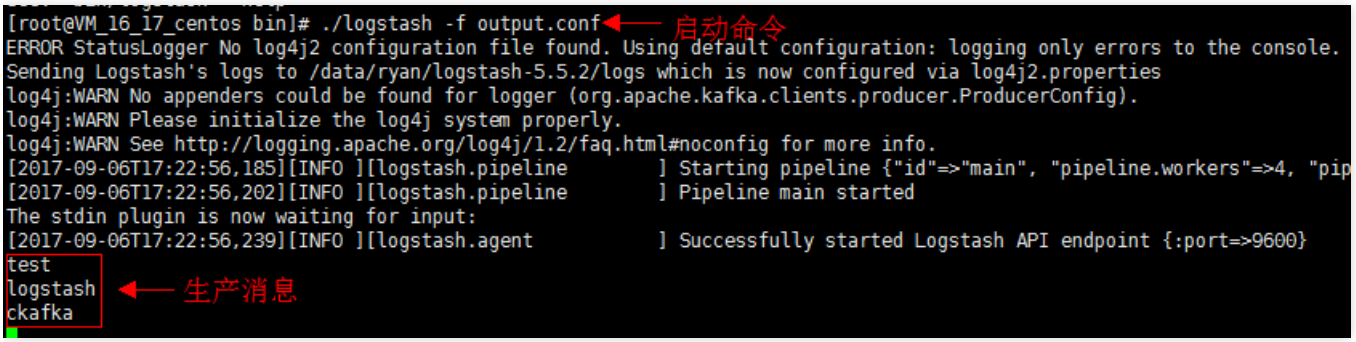
2. 在.bin/编写配置文件 output.conf。
   此处将标准输入作为数据来源，将 Kafka 作为数据目的地。

```
input {
   input {
     stdin{}
  }
}

output {
  kafka {
       bootstrap_servers => "xx.xx.xx.xx:xxxx"  // ckafka 实例接入地址
       topic_id => "logstash_test" // ckafka topic 名称
      }
}
```

3. 执行如下命令启动 Logstash，向创建的 Topic 发送消息。

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

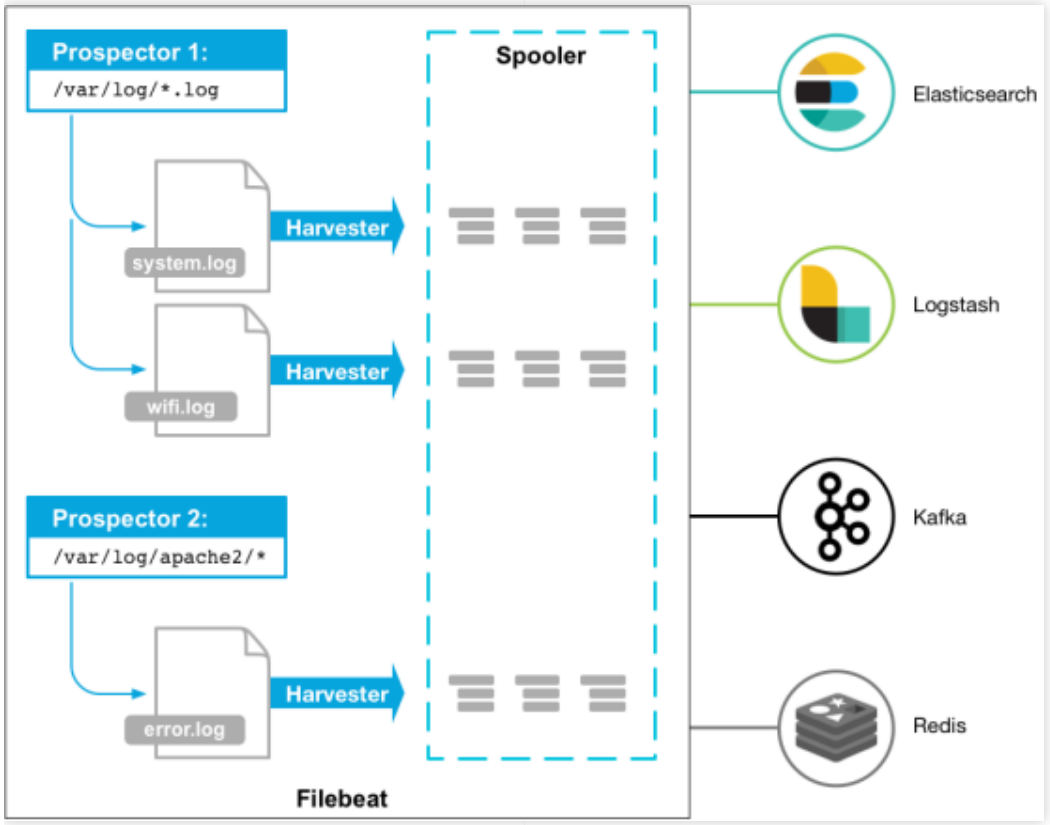


4. 启动CKafka消费者，检验上一步的生产数据。



Filebeats 接入 CKafka

Beats 平台 集合了多种单一用途数据采集器。这些采集器安装后可用作轻量型代理，从成百上千或成千上万台机器向目标发送采集数据。



Beats 有多种采集器，您可以根据自身的需求下载对应的采集器。本文以 Filebeat（轻量型日志采集器）为例，向您介绍 Filebeat 接入 CKafka 的操作指方法，及接入后常见问题的解决方法。

前提条件

- 下载并安装Filebeat。参见 Download Filebeat。
- 下载并安装JDK 8。参见 Download JDK 8。
- 已创建 CKafka 实例 。

操作步骤

步骤1. 获取 CKafka 实例接入地址

1. 登录 CKafka 控制台。
2. 在左侧导航栏选择【实例列表】，单击实例的"ID"，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



步骤2. 创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个名为test的 Topic。



步骤3. 准备配置文件

进入filebeat的安装目录，创建配置监控文件filebeat.yml。

```
#======== Filebeat prospectors ==========
filebeat.prospectors:

- input_type: log

# 此处为监听文件路径
  paths:
    - /var/log/messages

#========  Outputs =========

#------------------ kafka -------------------------------------
output.kafka:
  version:0.10.2 // 根据不同 CKafka 实例开源版本配置
  # 设置为CKafka实例的接入地址
  hosts: ["xx.xx.xx.xx:xxxx"]
  # 设置目标topic的名称
  topic: 'test'
  partition.round_robin:
    reachable_only: false

  required_acks: 1
  compression: none
  max_message_bytes: 1000000
```

```
# SASL 需要配置下列信息，如果不需要则下面两个选项可不配置
username: "yourinstance#yourusername"  //username 需要拼接实例ID和用户名
password: "yourpassword"
```

步骤4. Filebeat发送消息

1. 执行如下命令启动客户端。
   sudo ./filebeat -e -c filebeat.yml
2. 为监控文件增加数据（示例为写入监听的 testlog 文件）。

```
echo ckafka1 >> testlog
echo ckafka2 >> testlog
echo ckafka3 >> testlog
```

3. 开启 Consumer 消费对应的 Topic，获得以下数据。

```
{"@timestamp":"2017-09-29T10:01:27.936Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka1","offset":500,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
{"@timestamp":"2017-09-29T10:01:30.936Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka2","offset":508,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
{"@timestamp":"2017-09-29T10:01:33.937Z","beat":{"hostname":"10.193.9.26","name":"10.193.9.26","version":"5.6.2"},"input_type":"log","message":"ckafka3","offset":516,"source":"/data/ryanyyang/hcmq/beats/filebeat-5.6.2-linux-x86_64/testlog","type":"log"}
```

SASL/PLAINTEXT 模式

如果您需要进行 SALS/PLAINTEXT 配置，则需要配置用户名与密码。 在 Kafka 配置区域新增加 username 和 password 配置即可。

```
# SASL 需要配置下列信息，如果不需要则下面两个选项可不配置
username: "yourinstance#yourusername"  //username 需要拼接实例ID和用户名
password: "yourpassword"
```

常见问题

在 Filebeat 日志（默认路径 /var/log/filebeat/filebeat ）中，发现有大量 INFO 日志，例如：

```
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/544 starting up
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/544 state change to [open] on wp-news-filebeat/4
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/4 selected brok
```

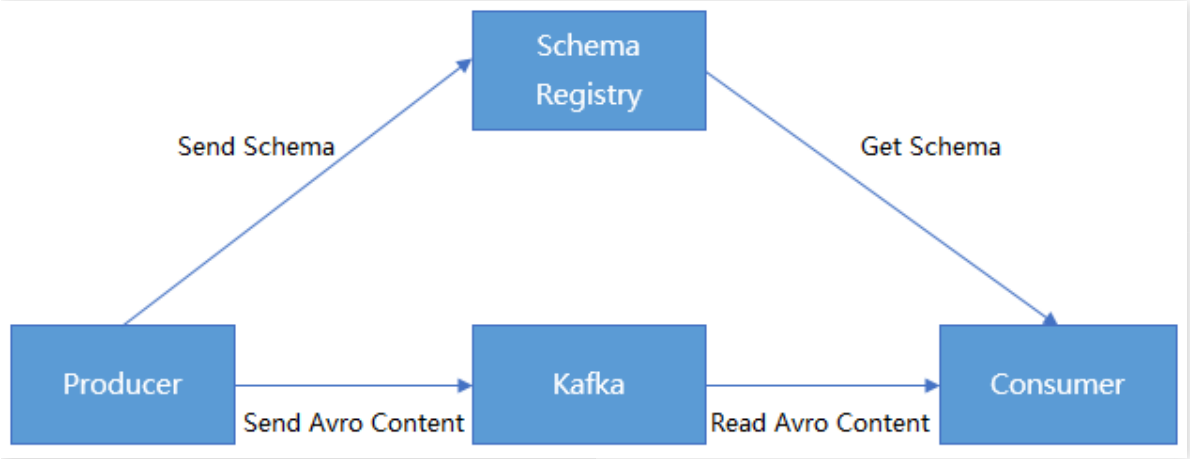
```
er 544
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/478 state change to [closing] bec
ause EOF
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 Closed connection to broker bitar1d12:9092
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/5 state change
to [retrying-3]
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/4 state change
to [flushing-3]
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/5 abandoning
broker 478
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/2 state change
to [retrying-2]
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/2 abandoning
broker 541
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-filebeat/3 state change
to [retrying-2]
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/broker/478 shut down
```

出现大量 INFO 可能是 Filebeat 版本有问题，因为 Elastic 家族的产品发版速度很频繁，而且不同大版本有很多不兼容。
例如：6.5.x 默认支持 Kafka 的版本是 0.9、0.10、1.1.0、2.0.0，而 5.6.x 默认支持的是0.8.2.0。
您需要检查配置文件中的版本配置：

```
output.kafka:
  version:0.10.2 // 根据不同 CKafka 实例开源版本配置
```

Schema Registry 接入 CKafka

无论是使用传统的 Avro API 自定义序列化类与反序列化类，还是使用 Twitter 的 Bijection 类库实现 Avro 的序列化与反序列化，两种方法有相同的缺点：在每条 Kafka 记录里都嵌入了 Schema，从而导致记录的大小成倍增加。但是不管怎样，在读取记录时仍然需要用到整个 Schema，所以要先找到 Schema。
CKafka 提供了数据共用一个 Schema 的方法：将 Schema 中的内容注册到 Confluent Schema Registry，Kafka Producer 和 Kafka Consumer 通过识别 Confluent Schema Registry 中的 schema 内容进行序列化和反序列化。



前提条件

- 下载 Download JDK 8。
- 下载 Confluent oss 4.1.1。
- 已创建实例。

操作步骤

步骤1. 获取实例接入地址并开启自动创建Topic

1. 登录 CKafka 控制台。
2. 在左侧导航栏选择【实例列表】，单击实例的“ID”，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



4. 在【自动创建 Topic】模块开启自动创建 Topic。

> 注意：
>
> 启动 oss 会创建 schemas 主题，所以实例中需要开启自动创建主题。

步骤2. 准备Confluent配置

1. 修改 oss 配置文件中的 server 地址等信息。配置信息如下：

```
kafkastore.bootstrap.servers=PLAINTEXT://xxxx
kafkastore.topic=schemas
debug=true
```

> 说明：
>
> bootstrap.servers：接入网络，在实例详情页面【接入方式】模块的网络列复制。



2. 执行如下命令启动 Schema Registry。

```
bin/schema-registry-start etc/schema-registry/schema-registry.properties
```

运行结果如下：

```
        kafkastore.init.timeout.ms = 60000
 (io.confluent.kafka.schemaregistry.rest.SchemaRegistryConfig:179)
[2019-07-09 16:33:24,889] INFO Logging initialized @523ms (org.eclipse.jetty.util.log:186)
[2019-07-09 16:33:25,607] INFO Initializing KafkaStore with broker endpoints: PLAINTEXT://172.26.0.8:9092 (
io.confluent.kafka.schemaregistry.storage.KafkaStore:103)
[2019-07-09 16:33:26,052] INFO Creating schemas topic _schemas (io.confluent.kafka.schemaregistry.storage.K
afkaStore:186)
[2019-07-09 16:33:26,424] INFO Initialized last consumed offset to -1 (io.confluent.kafka.schemaregistry.st
orage.KafkaStoreReaderThread:138)
[2019-07-09 16:33:26,437] INFO [kafka-store-reader-thread-_schemas]: Starting (io.confluent.kafka.schemareg
istry.storage.KafkaStoreReaderThread:66)
[2019-07-09 16:33:27,152] INFO Wait to catch up until the offset of the last message at 0 (io.confluent.kaf
ka.schemaregistry.storage.KafkaStore:277)
[2019-07-09 16:33:27,221] INFO Joining schema registry with Kafka-based coordination (io.confluent.kafka.sc
hemaregistry.storage.KafkaSchemaRegistry:209)
[2019-07-09 16:33:27,316] INFO Finished rebalance with master election result: Assignment{version=1, error=
0, master='sr-1-/172.26.0.11-2019-07-09 16:33:27:278-f5aa186c-a6c2-4c93-95dd-4a7b8ddb38a8', masterIdentity=
version=1,host=VM_0_11_centos,port=8081,scheme=http,masterEligibility=true} (io.confluent.kafka.schemaregis
try.masterelector.kafka.KafkaGroupMasterElector:232)
[2019-07-09 16:33:27,336] INFO Wait to catch up until the offset of the last message at 1 (io.confluent.kaf
ka.schemaregistry.storage.KafkaStore:277)
[2019-07-09 16:33:27,458] INFO Adding listener: http://0.0.0.0:8081 (io.confluent.rest.Application:190)
```

步骤3. 收发消息

现有 schema 文件，其中内容如下：

```
{
   "type": "record",
   "name": "User",
   "fields": [
      {"name": "id", "type": "int"},
      {"name": "name",  "type": "string"},
      {"name": "age", "type": "int"}
   ]
}
```

1. 注册 schema 到对应 Topic（注册 Topic 名为 test）
   下面的脚本是直接在 Schema Registry 部署的环境中使用 curl 命令调用对应 API 实现注册的一个示例：

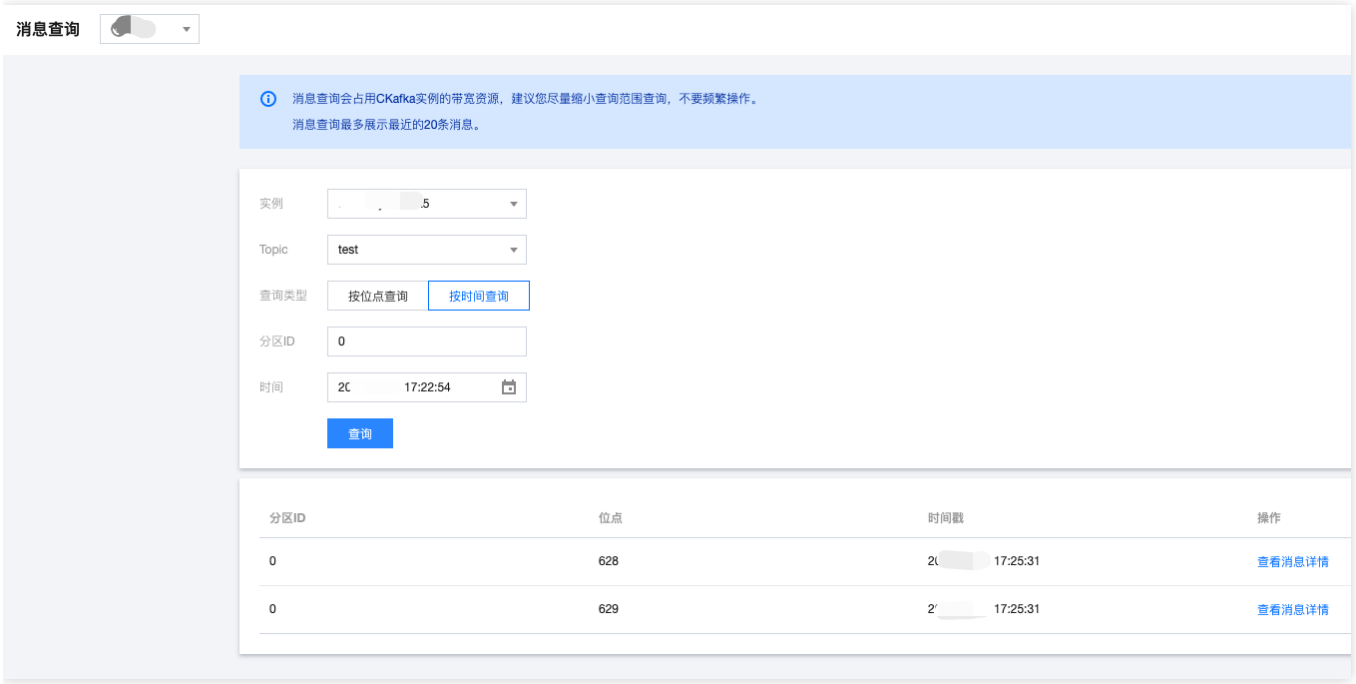
```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
--data '{"schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"id\", \"type\": \"int\"}, {\"n
ame\": \"name\",  \"type\": \"string\"}, {\"name\": \"age\", \"type\": \"int\"}]}"}' \
http://127.0.0.1:8081/subjects/test/versions
```

2. Kafka Producer 发送数据：

```
package schemaTest;
import java.util.Properties;
import java.util.Random;
import org.apache.avro.Schema;
```

```
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
public class SchemaProduce {
    public static final String USER_SCHEMA = "{\"type\": \"record\", \"name\": \"User\", " +
            "\"fields\": [{\"name\": \"id\", \"type\": \"int\"}, " +
            "{\"name\": \"name\",  \"type\": \"string\"}, {\"name\": \"age\", \"type\": \"int\"}]}";
      public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        // 添加CKafka实例的接入地址
       props.put("bootstrap.servers", "xx.xx.xx.xx:xxxx");
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        // 使用 Confluent 实现的 KafkaAvroSerializer
        props.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
        // 添加 schema 服务的地址，用于获取 schema
        props.put("schema.registry.url", "http://127.0.0.1:8081");
        Producer<String, GenericRecord> producer = new KafkaProducer<>(props);
        Schema.Parser parser = new Schema.Parser();
        Schema schema = parser.parse(USER_SCHEMA);
        Random rand = new Random();
        int id = 0;
        while(id < 100) {
            id++;
            String name = "name" + id;
            int age = rand.nextInt(40) + 1;
            GenericRecord user = new GenericData.Record(schema);
            user.put("id", id);
            user.put("name", name);
            user.put("age", age);
            ProducerRecord<String, GenericRecord> record = new ProducerRecord<>("test", user);
            producer.send(record);
            Thread.sleep(1000);
        }
        producer.close();
    }
}
```

运行一段时间后，在CKafka 控制台的【topic管理】页面，选择对应的 Topic ，单击【更多】>【消息查询】，查看刚刚发送的消息。

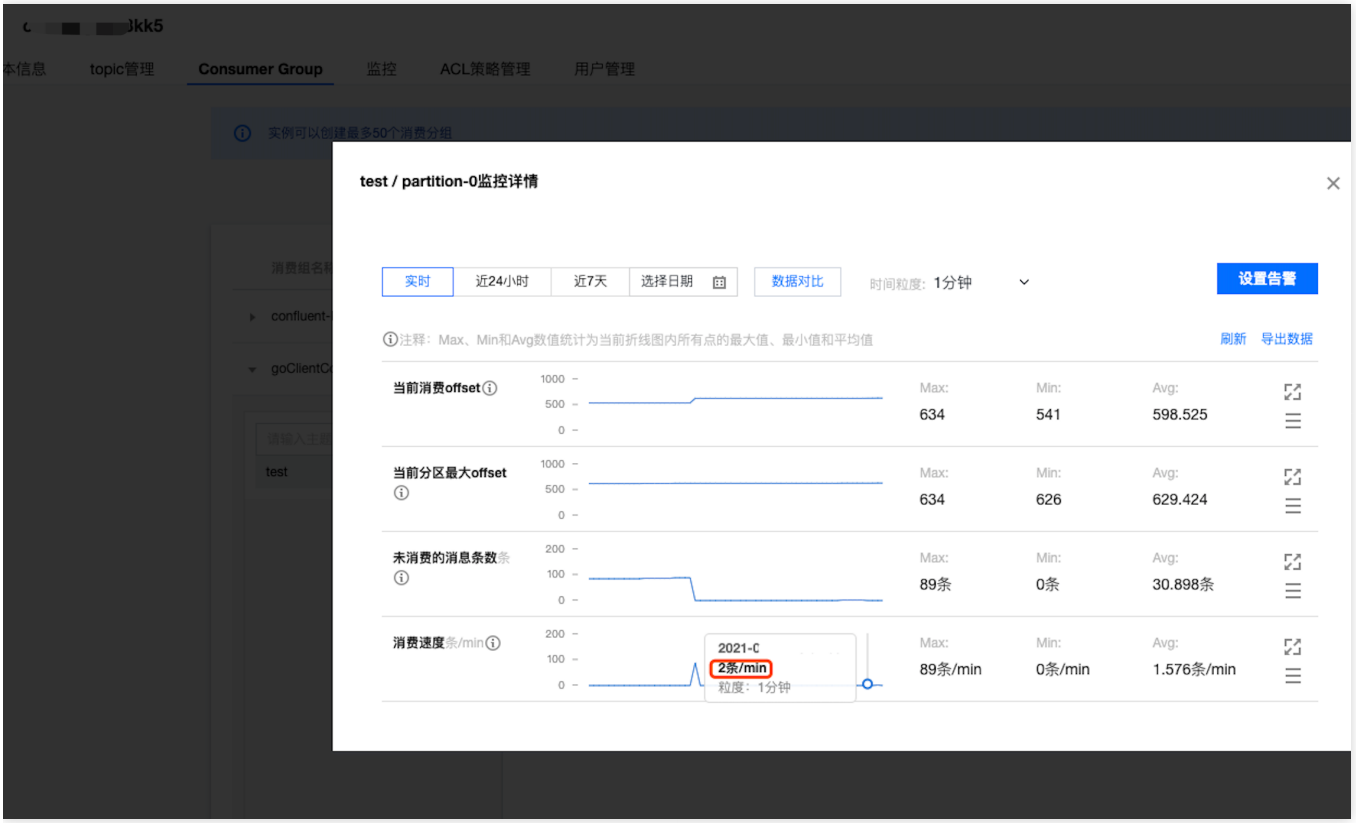


3. Kafka Consumer 消费数据：

```
package schemaTest;
import java.util.Collections;
import java.util.Properties;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
public class SchemaProduce {
    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.put("bootstrap.servers", "xx.xx.xx.xx:xxxx"); //CKafka实例的接入地址
        props.put("group.id", "schema");
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        // 使用Confluent实现的KafkaAvroDeserializer
        props.put("value.deserializer", "io.confluent.kafka.serializers.KafkaAvroDeserializer");
        // 添加schema服务的地址，用于获取schema
        props.put("schema.registry.url", "http://127.0.0.1:8081");
        KafkaConsumer<String, GenericRecord> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Collections.singletonList("test"));
        try {
            while (true) {
                ConsumerRecords<String, GenericRecord> records = consumer.poll(10);
                for (ConsumerRecord<String, GenericRecord> record : records) {
                    GenericRecord user = record.value();
                    System.out.println("value = [user.id = " + user.get("id") + ", " + "user.name = "
                            + user.get("name") + ", " + "user.age = " + user.get("age") + "], "
                            + "partition = " + record.partition() + ", " + "offset = " + record.offset());
```

```
                }
            }
        } finally {
            consumer.close();
        }
    }
}
```

在 CKafka 控制台 的【Consumer Group】页面，选择 schema消费组名称，在主题名称输入 Topic 名称，单击【查询详情】，查看消费详情。



启动消费者进行消费，下图为消费日志截图：

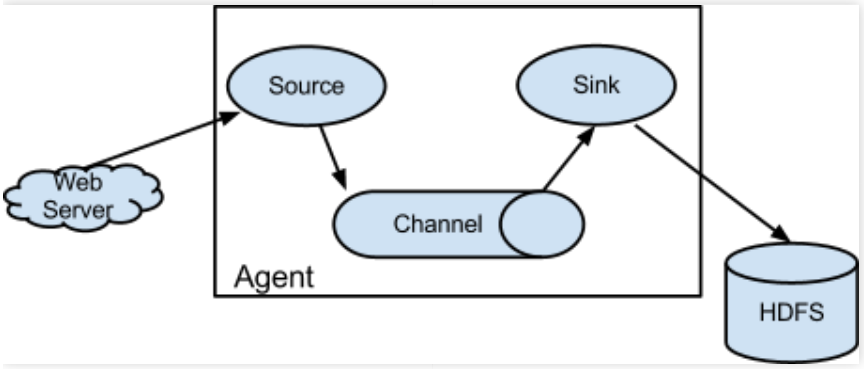
```
        value.subject.name.strategy = class io.confluent.kafka.serializers.subject.TopicNameStrategy
        key.subject.name.strategy = class io.confluent.kafka.serializers.subject.TopicNameStrategy

2019-07-09 22:07:32,547 INFO [org.apache.kafka.common.utils.AppInfoParser] - Kafka version : 1.1.1
2019-07-09 22:07:32,548 INFO [org.apache.kafka.common.utils.AppInfoParser] - Kafka commitId : 8e07427ffb493498
2019-07-09 22:07:33,357 INFO [org.apache.kafka.clients.Metadata] - Cluster ID: rcrVUkeuStKYEIbbfMixXg
2019-07-09 22:07:33,364 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Discovered group coordinator 172.26.0.8:9095 (id: 2147473628 rack: null)
2019-07-09 22:07:33,366 INFO [org.apache.kafka.clients.consumer.internals.ConsumerCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Revoking previously assigned partitions []
2019-07-09 22:07:33,366 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] (Re-)joining group
2019-07-09 22:07:33,405 INFO [org.apache.kafka.clients.consumer.internals.AbstractCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Successfully joined group with generation 5
2019-07-09 22:07:33,407 INFO [org.apache.kafka.clients.consumer.internals.ConsumerCoordinator] - [Consumer clientId=consumer-1, groupId=schema] Setting newly assigned partitions [test-1, test-0, test-2]
value = [user.id = 2, user.name = name2, user.age = 10], partition = 1, offset = 11384
value = [user.id = 5, user.name = name5, user.age = 29], partition = 1, offset = 11385
value = [user.id = 8, user.name = name8, user.age = 19], partition = 1, offset = 11386
value = [user.id = 11, user.name = name11, user.age = 17], partition = 1, offset = 11387
value = [user.id = 14, user.name = name14, user.age = 20], partition = 1, offset = 11388
value = [user.id = 17, user.name = name17, user.age = 17], partition = 1, offset = 11389
value = [user.id = 20, user.name = name20, user.age = 40], partition = 1, offset = 11390
value = [user.id = 23, user.name = name23, user.age = 29], partition = 1, offset = 11391
value = [user.id = 26, user.name = name26, user.age = 6], partition = 1, offset = 11392
value = [user.id = 29, user.name = name29, user.age = 31], partition = 1, offset = 11393
value = [user.id = 32, user.name = name32, user.age = 1], partition = 1, offset = 11394
value = [user.id = 35, user.name = name35, user.age = 29], partition = 1, offset = 11395
value = [user.id = 38, user.name = name38, user.age = 24], partition = 1, offset = 11396
value = [user.id = 41, user.name = name41, user.age = 2], partition = 1, offset = 11397
value = [user.id = 44, user.name = name44, user.age = 14], partition = 1, offset = 11398
value = [user.id = 47, user.name = name47, user.age = 13], partition = 1, offset = 11399
value = [user.id = 50, user.name = name50, user.age = 29], partition = 1, offset = 11400
value = [user.id = 53, user.name = name53, user.age = 14], partition = 1, offset = 11401
value = [user.id = 56, user.name = name56, user.age = 27], partition = 1, offset = 11402
value = [user.id = 59, user.name = name59, user.age = 26], partition = 1, offset = 11403
value = [user.id = 62, user.name = name62, user.age = 11], partition = 1, offset = 11404
value = [user.id = 65, user.name = name65, user.age = 37], partition = 1, offset = 11405
value = [user.id = 68, user.name = name68, user.age = 17], partition = 1, offset = 11406
value = [user.id = 71, user.name = name71, user.age = 29], partition = 1, offset = 11407
value = [user.id = 74, user.name = name74, user.age = 23], partition = 1, offset = 11408
value = [user.id = 77, user.name = name77, user.age = 14], partition = 1, offset = 11409
value = [user.id = 80, user.name = name80, user.age = 21], partition = 1, offset = 11410
value = [user.id = 83, user.name = name83, user.age = 19], partition = 1, offset = 11411
value = [user.id = 86, user.name = name86, user.age = 20], partition = 1, offset = 11412
value = [user.id = 89, user.name = name89, user.age = 9], partition = 1, offset = 11413
value = [user.id = 92, user.name = name92, user.age = 27], partition = 1, offset = 11414
value = [user.id = 95, user.name = name95, user.age = 17], partition = 1, offset = 11415
value = [user.id = 98, user.name = name98, user.age = 25], partition = 1, offset = 11416
value = [user.id = 3, user.name = name3, user.age = 2], partition = 0, offset = 11446
value = [user.id = 6, user.name = name6, user.age = 9], partition = 0, offset = 11447
```

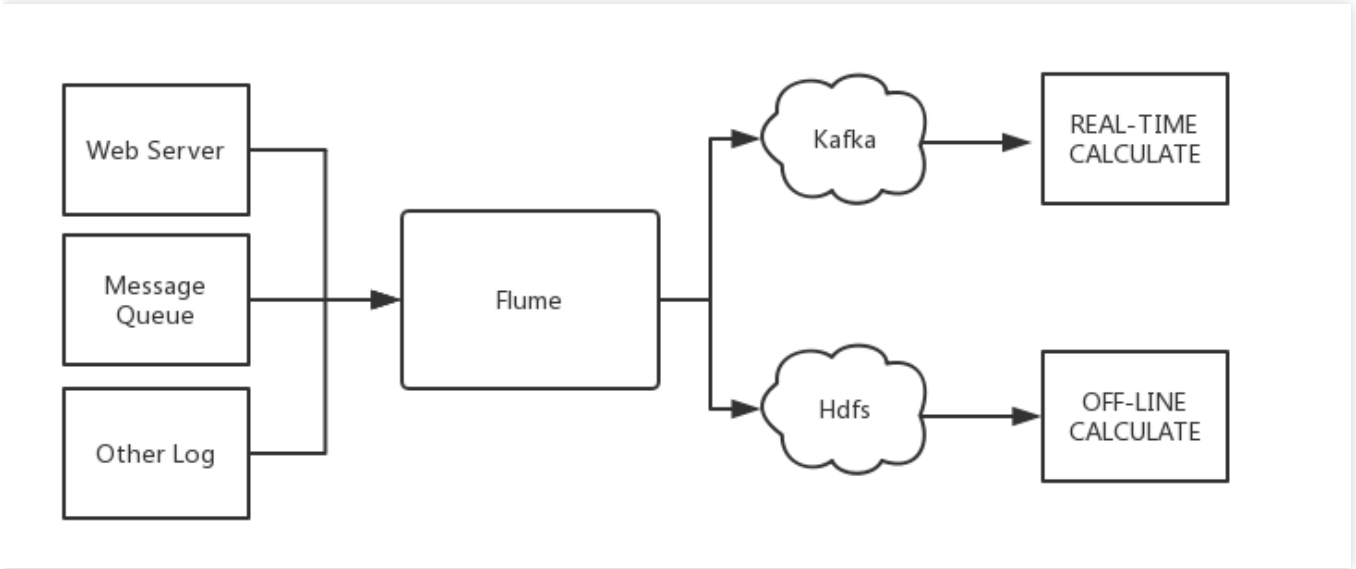
Flume接入CKafka最佳实践

Apache Flume 是一个分布式、可靠、高可用的日志收集系统，支持各种各样的数据来源（如 HTTP、Log 文件、JMS、监听端口数据等），能将这些数据源的海量日志数据进行高效收集、聚合、移动，最后存储到指定存储系统中（如 Kafka、分布式文件系统、Solr 搜索服务器等）。

Flume 基本结构如下：



Flume 以 agent 为最小的独立运行单位。一个 agent 就是一个 JVM，单个 agent 由 Source、Sink 和 Channel 三大组件构成。



Flume 与 Kafka

把数据存储到 HDFS 或者 HBase 等下游存储模块或者计算模块时需要考虑各种复杂的场景，例如并发写入的量以及系统承载压力、网络延迟等问题。Flume 作为灵活的分布式系统具有多种接口，同时提供可定制化的管道。

在生产处理环节中，当生产与处理速度不一致时，Kafka 可以充当缓存角色。Kafka 拥有 partition 结构以及采用 append 追加数据，使 Kafka 具有优秀的吞吐能力；同时其拥有 replication 结构，使 Kafka 具有很高的容错性。

所以将 Flume 和 Kafka 结合起来，可以满足生产环境中绝大多数要求。

Flume 接入开源 Kafka

准备工作

- 下载 Apache Flume （1.6.0以上版本兼容 Kafka ）
- 下载 Kafka工具包 （0.9.x以上版本，0.8已经不支持 ）
- 确认 Kafka 的 Source、 Sink 组件已经在 Flume 中。

接入方式

Kafka 可作为 Source 或者 Sink 端对消息进行导入或者导出。

Kafka Source

配置 kafka 作为消息来源，即将自己作为消费者，从 Kafka 中拉取数据传入到指定 Sink 中。主要配置选项如下：

配置项	说明
channels	自己配置的 Channel
type	必须为：org.apache.flume.source.kafka.KafkaSource
kafka.bootstrap.servers	Kafka Broker 的服务器地址
kafka.consumer.group.id	作为 Kafka 消费端的 Group ID
kafka.topics	Kafka 中数据来源 Topic
batchSize	每次写入 Channel 的大小
batchDurationMillis	每次写入最大间隔时间

示例：

```
tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
tier1.sources.source1.channels = channel1
tier1.sources.source1.batchSize = 5000
tier1.sources.source1.batchDurationMillis = 2000
tier1.sources.source1.kafka.bootstrap.servers = localhost:9092
tier1.sources.source1.kafka.topics = test1, test2
tier1.sources.source1.kafka.consumer.group.id = custom.g.id
```

更多内容请参考 Apache Flum 官网。

Kafka Sink

配置 Kafka 作为内容接收方，即将自己作为生产者，推到 Kafka Server 中等待后续操作。主要配置选项如下：

配置项	说明
channel	自己配置的 Channel

配置项	说明
type	必须为：org.apache.flume.sink.kafka.KafkaSink
kafka.bootstrap.servers	Kafka Broker 的服务器
kafka.topics	Kafka 中数据来源 Topic
kafka.flumeBatchSize	每次写入的 Bacth 大小
kafka.producer.acks	Kafka 生产者的生产策略

示例：

```
a1.sinks.k1.channel = c1
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = mytopic
a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
```

更多内容请参考 Apache Flum 官网。

Flume 接入 CKafka

使用 CKafka 作为 Sink

步骤1. 获取 CKafka 实例接入地址

1. 登录CKafka 控制台。
2. 在左侧导航栏选择【实例列表】，单击实例的"ID"，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



步骤2. 创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个名为 flume_test 的 Topic。



步骤3. 配置Flume

1. 下载 Apache Flume工具包并解压 。
2. 编写配置文件flume-kafka-sink.properties，以下是一个简单的 Demo （配置在解压目录的 conf 文件夹下），若无特殊要求则将自己的实例 IP 与 Topic 替换到配置文件当中即可。本例使用的 source 为 tail -F flume-test ，即文件中新增的信息。

```
# 以kafka 作为sink 的demo
agentckafka.sources = exectail
agentckafka.channels = memoryChannel
agentckafka.sinks = kafkaSink

# 设置source类型,根据不同需求而设置
agentckafka.sources.exectail.type = exec
agentckafka.sources.exectail.command = tail -F ./flume-test
agentckafka.sources.exectail.batchSize=20
# 设置source channel
agentckafka.sources.exectail.channels = memoryChannel

# 设置sink类型，此处设置为kafka
agentckafka.sinks.kafkaSink.type= org.apache.flume.sink.kafka.KafkaSink
# 此处设置ckafka提供的ip:port
agentckafka.sinks.kafkaSink.brokerList= 172.16.16.12:9092
# 此处设置需要导入数据的topic，请先再控制台提前创建好topic
agentckafka.sinks.kafkaSink.topic= flume_test
# 设置sink channel
agentckafka.sinks.kafkaSink.channel = memoryChannel

# Each channel's type is defined.
agentckafka.channels.memoryChannel.type = memory
agentckafka.channels.memoryChannel.keep-alive = 10

# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel
agentckafka.channels.memoryChannel.capacity = 1000
agentckafka.channels.memoryChannel.transactionCapacity =1000
```


3. 执行如下命令启动 Flume。

```
./bin/flume-ng agent -n agentckafka -c conf -f conf/flume-kafka-sink.properties
```

4. 写入消息到 flume-test 文件中，此时消息将由 Flume 写入到 CKafka。

```
[root@VM_16_17_centos apache-flume-1.7.0-bin]# cat flume-test
ckafka
```

5. 启动 CKafka 客户端进行消费。

   ./kafka-console-consumer.sh --bootstrap-server xx.xx.xx.xx:xxxx --topic flume_test --from-beginning --new-consumer

> 说明：
>
> bootstrap-server填写刚创建的CKafka实例的接入地址，topic填写刚刚创建的Topic名称。

可以看到刚才的消息被消费出来。

```
[root@VM_16_17_centos bin]# ./kafka-console-consumer.sh --bootstrap-server 172.16.16.12:9092 --topic flume_test --from-beginning --new-consumer
ckafka
```

使用 CKafka 作为 Source

步骤1. 获取 CKafka 实例接入地址

1. 登录CKafka 控制台。
2. 在左侧导航栏选择【实例列表】，单击实例的"ID"，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址。



步骤2. 创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在Topic管理页面，单击【新建】，创建一个名为 flume_test 的 Topic。



步骤3. 配置Flume

1. 下载 Apache Flume工具包并解压 。
2. 编写配置文件flume-kafka-source.properties，以下是一个简单的 Demo （配置在解压目录的 conf 文件夹下）。若无特殊要求则将自己的实例 IP 与 Topic 替换到配置文件当中即可。此处使用的 sink 为 logger。

```
# 以kafka 作为source 的demo
agentckafka.sources = kafkaSource
agentckafka.channels = memoryChannel
agentckafka.sinks = loggerSink

# 设置source类型,此处设置为kafka
agentckafka.sources.kafkaSource.type= org.apache.flume.source.kafka.KafkaSource
# 此处设置ckafka提供的ip:port
agentckafka.sources.kafkaSource.kafka.bootstrap.servers= 172.16.16.12:9092
# 此处设置需要导出数据的topic，请先再控制台提前创建好topic
agentckafka.sources.kafkaSource.kafka.topics= flume_test
# 设置找不到offset数据时的处理方式
agentckafka.sources.kafkaSource.kafka.consumer.auto.offset.reset= earliest
# 设置source channel
agentckafka.sources.kafkaSource.channels = memoryChannel

# 设置sink
agentckafka.sinks.loggerSink.type = logger
# 设置sink channel
agentckafka.sinks.loggerSink.channel = memoryChannel

# Each channel's type is defined.
agentckafka.channels.memoryChannel.type = memory
agentckafka.channels.memoryChannel.keep-alive = 10

# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel
agentckafka.channels.memoryChannel.capacity = 1000
agentckafka.channels.memoryChannel.transactionCapacity =1000
```


3. 执行如下命令启动 Flume。

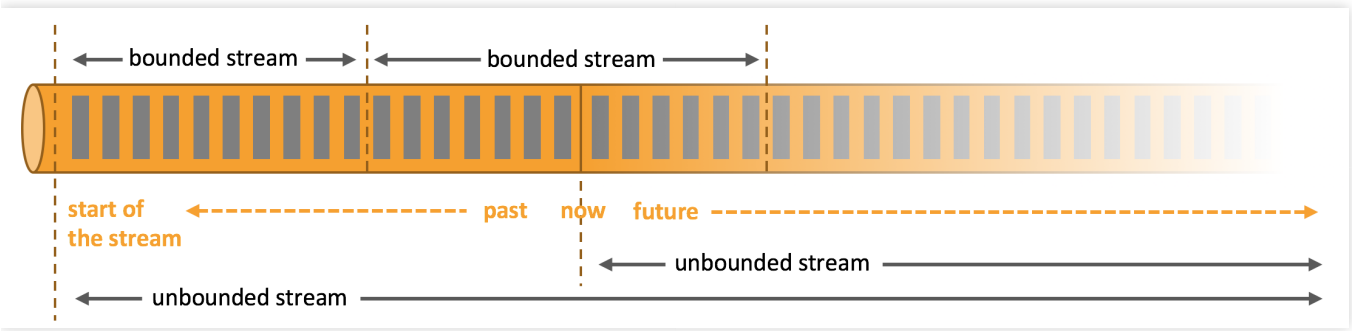
```
./bin/flume-ng agent -n agentckafka -c conf -f conf/flume-kafka-source.properties
```

4. 查看 logger 输出信息（默认路径为 logs/flume.log ）。

```
Component type: SOURCE, name: kafkaSource started
- Event: { headers:{timestamp=1501136891423, topic=flume_test, partition=0} body: 63 6B 61 66 6B 61
```

Flink 接入 CKafka

Apache Flink 是一个可以处理流数据的实时处理框架，用于在无界和有界数据流上进行有状态的计算。Flink 能在所有常见集群环境中运行，并能以内存速度和任意规模进行计算。



Apache Flink 擅长处理无界和有界数据集。Flink runtime 能够通过对时间和状态的精确控制处理无界数据流，也能够使用为固定大小数据集设计的算法和数据结构对有界数据集进行处理，并达到出色的性能。

应用程序可能会使用来自各种数据源（如消息队列或分布式日志，如 Apache Kafka 或 Kinesis）的实时数据。Flink 提供了 Apache Kafka 连接器，用于从 Kafka topic 中读取或者向其中写入数据，可提供精确一次的处理语义。

操作步骤

步骤1：获取 CKafka 实例接入地址

1. 登录CKafka 控制台。
2. 在左侧导航栏选择【实例列表】，单击实例的"ID"，进入实例基本信息页面。
3. 在实例的基本信息页面的【接入方式】模块，可获取实例的接入地址，接入地址是生产消费需要用到的 bootstrap-server。



步骤2：创建 Topic

1. 在实例基本信息页面，选择顶部【Topic管理】页签。
2. 在 Topic 管理页面，单击【新建】，创建一个名为 test 的 Topic，接下来将以该 Topic 为例介绍如何消费。



步骤3：添加 Maven 依赖

pom.xml 配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Test-CKafka</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.10.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.25</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-java</artifactId>
            <version>1.6.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.11</artifactId>
            <version>1.6.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kafka_2.11</artifactId>
            <version>1.7.0</version>
        </dependency>
    </dependencies>
    <build>
```

```
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.3</version>
            <configuration>
              <source>1.8</source>
              <target>1.8</target>
            </configuration>
          </plugin>
        </plugins>
    </build>
</project>
```

步骤4：消费CKafka中的消息

- 通过VPC方式消费：

```
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import java.util.Properties;

public class CKafkaConsumerDemo {
    public static void main(String args[]) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        Properties properties = new Properties();
        //公网接入域名地址,即公网路由地址,在实例详情页的接入方式模块获取。
        properties.setProperty("bootstrap.servers", "IP:PORT");
        //消费者组id。
        properties.setProperty("group.id", "testConsumerGroup");
        DataStream<String> stream = env
                .addSource(new FlinkKafkaConsumer<>("topicName", new SimpleStringSchema(), propertie
s));
        stream.print();
        env.execute();
    }
}
```

常见问题
功能相关

Cloud Kafka 兼容哪一版的开源 Kafka？（当前的 CKafka 是基于开源 Kafka 的哪个版本？）

当前 CKafka 兼容 Apache Kafka 1.1-2.8 及以下版本，推荐生产消费端选用对应开源版本 SDK。

什么是主题（TOPIC）？

Topic 是每条发布到 Cloud Kafka 集群的消息所属的类别，即 Cloud Kafka 是面向 topic 的。用户需要先创建 topic 然后才能读写。

什么是分区（PARTITION）？

Partition 是物理上的概念，每个 topic 会被分成一个或多个 partition。partition 可以用来水平扩展 topic 的吞吐，发布的消息将被写入不同 partition，并被若干消费者同时读取。由于 Cloud Kafka 分配的单位是 partition，因此在本质上，topic 的并行吞吐量和 partition 个数成正比。

Cloud Kafka 和 CMQ 有什么区别？

CMQ 提供金融级的高可靠、高数据持久性消息传输，保证数据强一致性。
Cloud Kafka 适用于要求更高吞吐率，对可靠性要求相对较低的场景（如日志聚合等业务）。此外，Cloud Kafka 完美兼容 kafka 的老用户，可以做到零迁移成本，实例完全独占。

Kafka 客户端是否可以直接连接 Cloud Kafka 服务？

Cloud Kafka 可以兼容 Apache Kafka 1.1-2.8版本，您可以通过 Kafka 客户端连接消息中心，并且把代码部署在云平台服务中生产或消费消息。

消息队列 CKafka 是否会暴露 ZooKeeper？

不开放 ZooKeeper，不提供 zk 地址。

CKafka 是否支持消息压缩？

当前 CKafka 支持开源的 snappy 和 lz4 的消息压缩格式。由于 Gzip 压缩对于 CPU 的消耗较高，暂未支持。
测试期间建议客户关闭消息压缩参数进行测试。
配置开启方法：Producer 的配置文件中参数 compression.type = snappy 或者 lz4，默认为关闭 none。

CKafka 如何保证安全性？

CKafka 通过如下安全特性确保安全性：

- 租户隔离：实例的网络访问在账户间默认隔离。
- 权限控制：CKafka 额外应用层上做了来源 IP 白名单的鉴权机制，支持 SASL 鉴权。

- 安全防护：提供多纬度的安全防护、防 DDoS 攻击等服务。

CKafka 是否会丢失消息？

1. 开源的 Apache Kafka 不保证不丢消息；CKafka 针对可用性做了优化，云平台承诺 CKafka 的可用性超99.95%。
2. CKafka 客户可以通过生产时开启 ACK ，尽量保障不丢失和少丢失消息，提升消息可靠性。
3. 变更集群或升级过程对客户透明，秒级变更。
4. CKafka 面向的使用场景主要是需要高吞吐、高性能的大数据处理场景，对数据可靠性要求不十分苛刻，极端场景下可能会有少量的消息丢失；若需保障完全不丢失消息，且对性能要求不是非常高的场景，推荐使用CMQ。

CKafka的产品限制有哪些？

具体参见使用限制。

配置相关

如何选取 CKafka 副本数？

建议您创建 Topic 时选择双副本或三副本存储数据，保障数据可靠性。当前 CKafka 已经禁止单副本 Topic 的创建，如您账户下有单副本的 Topic，建议按如下步骤迁移：

1. 创建新的 Topic，选择相同的分区，选取双副本；
2. 生产消息到新的 Topic 中，存量的单副本 Topic 继续被消费；
3. 消费完毕后修改消费者配置，订阅新的 Topic 进行消费。

消息队列新接入客户端时生产/消费错误如何处理？

- 检查 telnet 是否通（网络问题，是否 Kafka 和生产者在相同网络环境下）。
- 检查访问的 vip - port 是否配置正确。
- 检查 Topic 白名单是否开启，如果开启需要配置正确的 IP 才能访问。

CKafka 消息保留时间配置为 1min，是否会在 1min 后立即删除堆积消息？

不一定。消息删除不仅和保留时间配置有关，也和生产消息的数据量级有关。
CKafka 删除堆积消息的最小单位是 partition 级别的文件分片，当前文件分片大小为1GB，堆积未达到一个文件分片是不会删除的。如果有10 partition，在1分钟内如果没有达到10GB的量，就不会有文件滚动，也就不会删除。

CKafka 消息堆积了很多如何处理？

CKafka 跟开源的 Kafka 是一模一样的机制和原理。您可以通过以下步骤来排错：

1. 确定您的业务有几个消费者在消费。
2. 若消费者消费能力比较差，直接加消费者即可。

CKafka 如何进行限流？

为保证服务的稳定性，CKafka 在消息出入上都做了流量管控。

- 用户所有副本流量之和超过购买时的峰值流量时，会发生限流。
- 在生产端发生限流时，CKafka 会延长一个 TCP 链接的回应时间，延迟时间取决于用户瞬时流量超过限制的大小。和道路交通管制的原理有点类似，流量超得越多，延时算法得出来的延时值越高，最高5分钟。
- 在消费端发生限流时，CKafka 会缩小每次 fetch.request.max.bytes 的大小，控制消费端的流量。

> 注意：
>
> 限流是在超出峰值流量或磁盘量后开始触发，因此存在短时间内使用资源超出限制值的情况，不过随着限流机制的作用资源使用会迅速回落。

限流会如何影响消息的生产消费？

限流本身不影响消息的内容，只会影响消息收发的速率。

如何判断 CKafka 是否发生限流？

1. 在实例列表上，每个集群都有对应的健康度展示。当健康度显示为“警告”字样时，可以将鼠标移至其上查看弹出的详细数据。这个数据会展示当前用户的峰值流量以及发生限流的次数，用户可以根据这里的数据判断该实例是否发生过限流。
2. 用户可以打开监控数据查看流量的最大值，如果 (流量的最大值 × 副本数) > 购买时的峰值带宽，则表明至少发生过一次限流。

通用参考
数据可靠性说明

本文将分别通过生产端、服务端（CKafka）和消费端介绍影响消息队列 CKafka 可靠性的因素，并提供对应的解决方法。

生产端数据丢失如何处理？

数据丢失原因

生产者将数据发送到消息队列 CKafka 时，数据可能因为网络抖动而丢失，此时消息队列 CKafka 未收到该数据。可能情况：

- 网络负载高或者磁盘繁忙时，生产者又没有重试机制。
- 磁盘超过购买规格的限制，例如实例磁盘规格为9000GB，在磁盘写满后未及时扩容，会导致数据无法写入到消息队列 CKafka。
- 突发或持续增长峰值流量超过购买规格的限制，例如实例峰值吞吐规格为100MB/s，在长时间峰值吞吐超过限制后未及时扩容，会导致数据写入消息队列 CKafka 变慢，生产者有排队超时机制时，导致数据无法写入到消息队列 CKafka。

解决方法

- 生产者对自己重要的数据，开启失败重试机制。
- 针对磁盘使用，在配置实例时设置好监控和 告警策略 ，可以做到事先预防。
  遇到磁盘写满时，可以在控制台及时升配（消息队列 CKafka 非独占实例间升配为平滑升配不停机且也可以单独升配磁盘）或者通过修改消息保留时间降低磁盘存储。
- 为了尽可能减少生产端消息丢失，您可以通过 buffer.memory 和 batch.size（以字节为单位）调优缓冲区的大小。缓冲区并非越大越好，如果由于某种原因生产者宕机了，那么缓冲区存在的数据越多，需要回收的垃圾越多，恢复就会越慢。应该时刻注意生产者的生产消息数情况、平均消息大小等（消息队列 CKafka 监控中有丰富的监控指标）。
- 配置生产端 ACK
  当 producer 向 leader 发送数据时，可以通过 request.required.acks 参数以及 min.insync.replicas 设置数据可靠性的级别。
- 当 acks = 1时（默认值），生产者在 ISR 中的 leader 已成功收到数据可以继续发送下一条数据。如果 leader 宕机，由于数据可能还未来得及同步给其 follower，则会丢失数据。
- 当 acks = 0时，生产者不等待来自 broker 的确认就发送下一条消息。这种情况下数据传输效率最高，但数据可靠性确最低。
- 当 acks = -1或者 all 时，生产者需要等待 ISR 中的所有 follower 都确认接收到消息后才能发送下一条消息，可靠性最高。
  即使按照上述配置 ACK，也不能保证数据不丢，例如，当 ISR 中只有 leader 时（ISR 中的成员由于某些情况

会增加也会减少，最少时只剩一个 leader），此时会变成 acks = 1的情况。所以需要同时在配合 min.insync.replicas 参数（此参数可以在消息队列 CKafka 控制台 Topic 配置开启高级配置中进行配置），min.insync.replicas 表示在 ISR 中最小副本的个数，默认值是1，当且仅当 acks = -1或者 all 时生效。

建议配置的参数值

此参数值仅供参考，实际数值需要依业务实际情况而定。

- 重试机制： message.send.max.retries=3;retry.backoff.ms=10000;
- 高可靠的保证： request.required.acks=-1;min.insync.replicas=2;
- 高性能的保证： request.required.acks=0;
- 可靠性+性能： request.required.acks=1;

服务端（CKafka）数据丢失如何处理？

数据丢失原因

- partition 的 leader 在未完成副本数 followers 的备份时就宕机，即使选举出了新的 leader 但是数据因为未来得及备份就丢失。
- 开源 Kafka 的落盘机制为异步落盘，也就是数据是先存在 PageCache 中的，当还没有正式落盘时，broker 出现断点或者重启或者故障时，PageCache 上的数据由于没有来及落磁盘进而丢失。
- 磁盘故障导致已经落盘的数据丢失。

解决方法

- 开源 Kafka 是多副本的，官方推荐通过副本来保证数据的完整性，此时如果是多副本，同时出现多副本多 broker 同时挂掉才会丢数据，比单副本数据的可靠性高很多，所以消息队列 CKafka 强制 Topic 是双副本，可配置3副本。
- 消息队列 CKafka 服务配置了更合理的参数 log.flush.interval.messages 和 log.flush.interval.ms，对数据进行刷盘。
- 消息队列 CKafka 对磁盘做了特殊处理，保证部分磁盘损坏时也不会影响数据的可靠性。

建议配置的参数值

非同步状态的副本可以选举为 leader： unclean.leader.election.enable=false // 关闭

消费端数据丢失如何处理？

数据丢失原因

- 还未真正消费到数据就提交 commit 了 offset，若过程中消费者挂掉，但 offset 已经刷新，消费者错过了一条数据，需要消费分组重新设置 offset 才能找回数据。
- 消费速度和生产速度相差太久，而消息保存时间太短，导致消息还未及时消费就被过期删除。

解决方法

- 合理配置参数 auto.commit.enable，等于 true 时表示自动提交。建议使用定时提交，避免频繁 commit offset。
- 监控消费者的情况，正确调整数据的保留时间。监控当前消费 offset 以及未消费的消息条数，并配置告警，防止由于消费速度过慢导致消息过期删除。

数据丢失排查方案

在本地打印分区partition和偏移量offset进行排查

打印信息代码如下：

```
Future<RecordMetadata> future = producer.send(new ProducerRecord<>(topic, messageKey, messag
eStr));
RecordMetadata recordMetadata = future.get();
log.info("partition: {}", recordMetadata.partition());
log.info("offset: {}", recordMetadata.offset());
```

如果能够打印出partition和offset，则表示当前发送的消息在服务端已经被正确保存。此时可以通过消息查询的工具去查询相关位点的消息即可。
如果打印不出partition和offset，则表示消息没有被服务端保存，客户端需要重试。

SDK文档
Go SDK
01 VPC 网络接入

操作背景

该任务以 Go 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装 Go
- 下载demo

操作步骤

步骤一：准备配置：

1. 将下载的demo中的gokafkademo上传至linux服务器。
2. 登录linux系统，进入gokafkademo目录，执行以下命令添加依赖库。

```
go get -v gopkg.in/confluentinc/confluent-kafka-go.v1/kafka
```

3. 修改配置文件 kafka.json。

```
{
  "topic": [
     "test"
  ],
  "bootstrapServers": [
     "xxx-.ap-xxxxx-ec.ckafka.cloudmq.com:6000"
  ],
  "consumerGroupId": "yourConsumerId"
 }
```

参数	描述
topic	Topic名称，您可以在控制台上【topic管理】页面复制。 

参数	描述
bootstrapServers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。 
consumerGroupId	您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。

步骤二：发送消息

1. 编写生产消息程序。

```
package main
import (
"fmt"
"gokafkademo/config"
"log"
"strings"
    "github.com/confluentinc/confluent-kafka-go/kafka"
)
func main() {
    cfg, err := config.ParseConfig("../../config/kafka.json")
if err != nil {
    log.Fatal(err)
}
    p, err := kafka.NewProducer(&kafka.ConfigMap{
    // 设置接入点，请通过控制台获取对应Topic的接入点。
    "bootstrap.servers": strings.Join(cfg.Servers, ","),
    // 用户不显示配置时，默认值为1。用户根据自己的业务情况进行设置
    "acks": 1,
    // 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
    "retries": 0,
    // 发送请求失败时到下一次重试请求之间的时间
    "retry.backoff.ms": 100,
    // producer 网络请求的超时时间。
    "socket.timeout.ms": 6000,
    // 设置客户端内部重试间隔。
    "reconnect.backoff.max.ms": 3000,
})
if err != nil {
    log.Fatal(err)
}
    defer p.Close()
    // 产生的消息 传递至报告处理程序
go func() {
    for e := range p.Events() {
        switch ev := e.(type) {
        case *kafka.Message:
            if ev.TopicPartition.Error != nil {
                fmt.Printf("Delivery failed: %v\n", ev.TopicPartition)
```

```
            } else {
                fmt.Printf("Delivered message to %v\n", ev.TopicPartition)
            }
        }
    }
}()
    // 异步发送消息
topic := cfg.Topic[0]
for _, word := range []string{"Confluent-Kafka", "Golang Client Message"} {
    _ = p.Produce(&kafka.Message{
        TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
        Value:          []byte(word),
    }, nil)
}
    // 等待消息传递
p.Flush(10 * 1000)
```

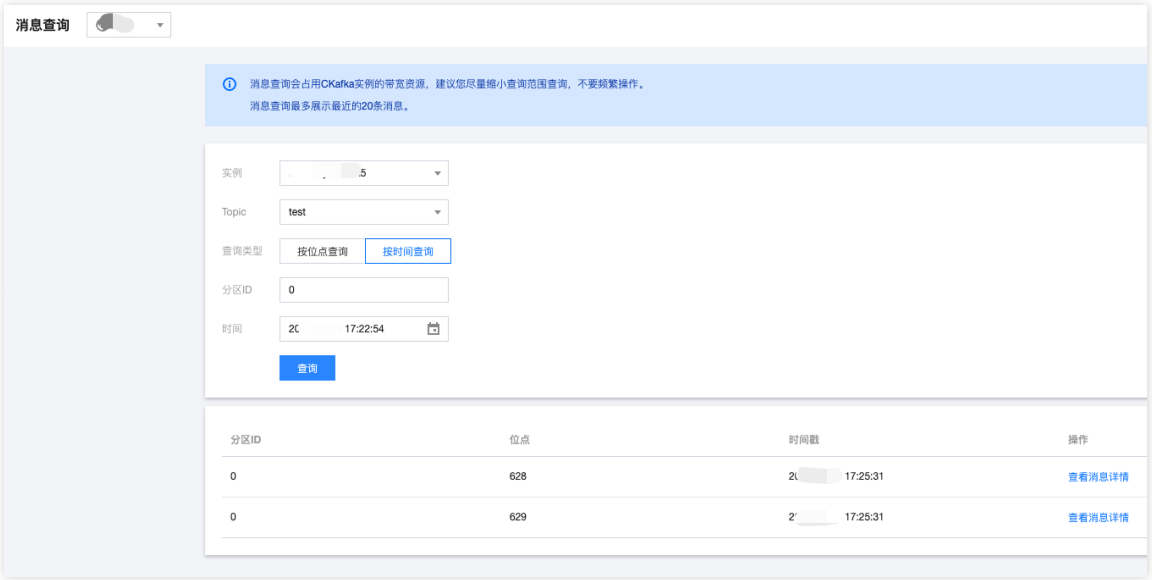
2. 编译并运行程序发送消息。

```
go run main.go
```

3. 查看运行结果，示例如下。

```
Delivered message to test[0]@628
Delivered message to test[0]@629
```

4. 在 CKafka 控制台的【topic管理】页面，选择对应的 Topic ， 点击【更多】>【消息查询】，查看刚刚发送的消息。



步骤三：消费消息

1. 编写消费消息程序。

```
package main

import (
```

```
    "fmt"
    "gokafkademo/config"
    "log"
    "strings"

    "github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

    cfg, err := config.ParseConfig("../../config/kafka.json")
    if err != nil {
        log.Fatal(err)
    }

    c, err := kafka.NewConsumer(&kafka.ConfigMap{
        // 设置接入点，请通过控制台获取对应Topic的接入点。
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // 设置的消息消费组
        "group.id":          cfg.ConsumerGroupId,
        "auto.offset.reset": "earliest",

        // 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，认为该消费者故障失败，Broker 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置group.min.session.timeout.ms=6000和group.max.session.timeout.ms=300000 之间
        "session.timeout.ms": 10000,
    })

    if err != nil {
        log.Fatal(err)
    }
    // 订阅的消息topic 列表
    err = c.SubscribeTopics(cfg.Topic, nil)
    if err != nil {
        log.Fatal(err)
    }

    for {
        msg, err := c.ReadMessage(-1)
        if err == nil {
            fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
        } else {
            // 客户端将自动尝试恢复所有的 error
            fmt.Printf("Consumer error: %v (%v)\n", err, msg)
        }
    }

    c.Close()
}
```

2. 编译并运行程序消费消息。

```
go run main.go
```

3. 查看运行结果，示例如下。

```
Message on test[0]@628: Confluent-Kafka
```

Message on test[0]@629: Golang Client Message

4. 在CKafka 控制台的【Consumer Group】页面，选择对应的消费组名称，在主题名称输入topic名称，点击【查询详情】，查看消费详情。

Node
01 VPC 网络接入

操作场景

该任务以 Node.js 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装 GCC
- 安装 Node.js
- 下载demo

操作步骤

准备工作

1. 将下载的demo中的nodejskafkademo上传至linux服务器
2. 登录linux服务器，进入nodejskafkademo目录。

步骤一：安装 C++ 依赖库

1. 执行以下命令切换到 yum 源配置目录 /etc/yum.repos.d/ 。

   ```
   cd /etc/yum.repos.d/
   ```

2. 创建 yum 源配置文件 confluent.repo。

   ```
   [Confluent.dist]
   name=Confluent repository (dist)
   baseurl=https://packages.confluent.io/rpm/5.1/7
   gpgcheck=1
   gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
   enabled=1
   [Confluent]
   name=Confluent repository
   baseurl=https://packages.confluent.io/rpm/5.1
   gpgcheck=1
   gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
   enabled=1
   ```

3. 执行以下命令安装 C++ 依赖库。

   ```
   yum install librdkafka-devel
   ```

步骤二：安装 Node.js 依赖库

1. 执行以下命令为预处理器指定 OpenSSL 头文件路径。

```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. 执行以下命令为连接器指定 OpenSSL 库路径。

```
export LDFLAGS=-L/usr/local/opt/openssl/lib
```

3. 执行以下命令安装 Node.js 依赖库。

```
npm install i --unsafe-perm node-rdkafka
```

步骤三：准备配置

创建消息队列 CKafka 配置文件 setting.js。

```
module.exports = {
    'bootstrap_servers': ["xxx.ckafka.cloudmq.com:6018"],
    'topic_name': 'xxx',
    'group_id': 'xxx'
}
```

参数	描述
bootstrap_servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。 
topic_name	Topic名称，您可以在控制台上【topic管理】页面复制。 
group_id	您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。

步骤四：发送消息

1. 编写生产消息程序 producer.js。

```
const Kafka = require('node-rdkafka');
```

```
const config = require('./setting');
console.log("features:" + Kafka.features);
console.log(Kafka.librdkafkaVersion);

var producer = new Kafka.Producer({
    'api.version.request': 'true',
    // 设置入口服务，请通过控制台获取对应的服务地址。
    'bootstrap.servers': config['bootstrap_servers'],
    'dr_cb': true,
    'dr_msg_cb': true,

    // 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
    'retries': '0',
    // 发送请求失败时到下一次重试请求之间的时间
    "retry.backoff.ms": 100,
    // producer 网络请求的超时时间。
    'socket.timeout.ms': 6000,
});

var connected = false

producer.setPollInterval(100);

producer.connect();

producer.on('ready', function() {
connected = true
console.log("connect ok")
});

producer.on("disconnected", function() {
connected = false;
producer.connect();
})

producer.on('event.log', function(event) {
    console.log("event.log", event);
});

producer.on("error", function(error) {
    console.log("error:" + error);
});

function produce() {
    try {
        producer.produce(
        config['topic_name'],
        null,
        new Buffer('Hello CKafka Default'),
        null,
        Date.now()
        );
    } catch (err) {
        console.error('Error occurred when sending message(s)');
        console.error(err);
    }

}
```

```
producer.on('delivery-report', function(err, report) {
    console.log("delivery-report: producer ok");
});

producer.on('event.error', function(err) {
    console.error('event.error:' + err);
})

setInterval(produce, 1000, "Interval");
```

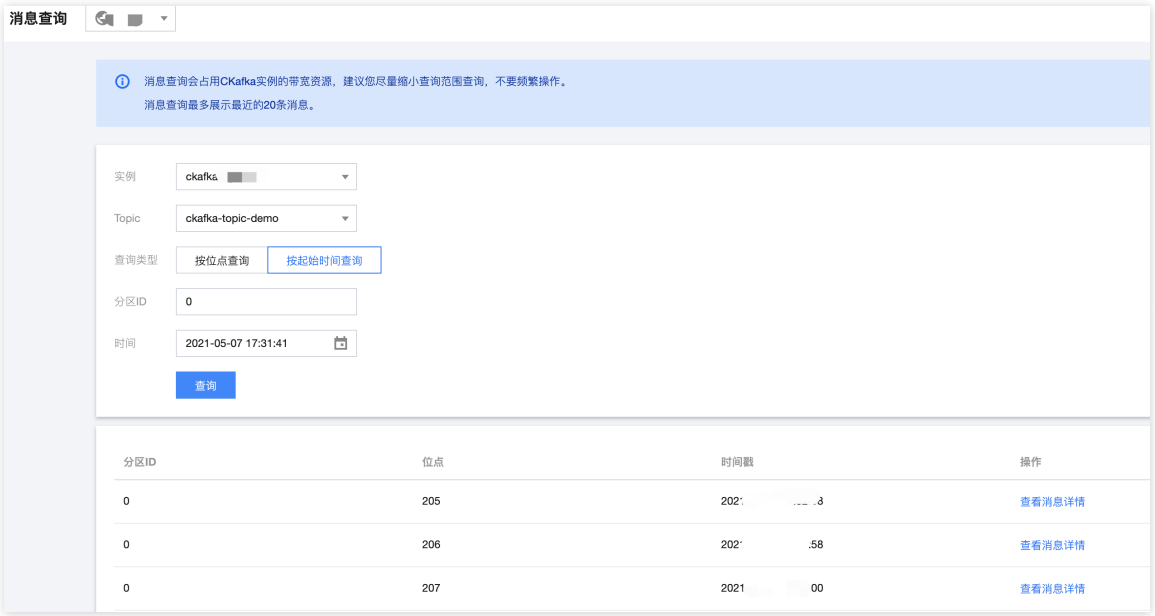
2. 执行以下命令发送消息。

```
node producer.js
```

3. 查看运行结果。

```
~/Demos/ckafka-demo/nodejskafkademo/sasl  ckafka_demo ±  node producer.js
features:gzip,snappy,sasl,regex,lz4
0.9.5
connect ok
(node:59829) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.a
lloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
```

4. 在 CKafka 控制台【topic管理】页面，选择对应的 topic，单击【更多】>【消息查询】，查看刚刚发送的消息。



5.

步骤五：订阅消息

1. 创建消费消息程序 consumer.js。

```
const Kafka = require('node-rdkafka');
    const config = require('./setting');
    console.log(Kafka.features);
    console.log(Kafka.librdkafkaVersion);
    console.log(config)

    var consumer = new Kafka.KafkaConsumer({
      'api.version.request': 'true',
      // 设置入口服务，请通过控制台获取对应的服务地址。
      'bootstrap.servers': config['bootstrap_servers'],
      'group.id' : config['group_id'],

      // 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，
      // 认为该消费者故障失败，Broker 发起重新 Rebalance 过程。
      'session.timeout.ms': 10000,
      // 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
      'metadata.request.timeout.ms': 305000,
      // 设置客户端内部重试间隔。
      'reconnect.backoff.max.ms': 3000

    });

    consumer.connect();

    consumer.on('ready', function() {
    console.log("connect ok");
    consumer.subscribe([config['topic_name']]);
    consumer.consume();
    })

    consumer.on('data', function(data) {
    console.log(data);
    });

    consumer.on('event.log', function(event) {
      console.log("event.log", event);
    });

    consumer.on('error', function(error) {
      console.log("error:" + error);
    });

    consumer.on('event', function(event) {
          console.log("event:" + event);
    });
```

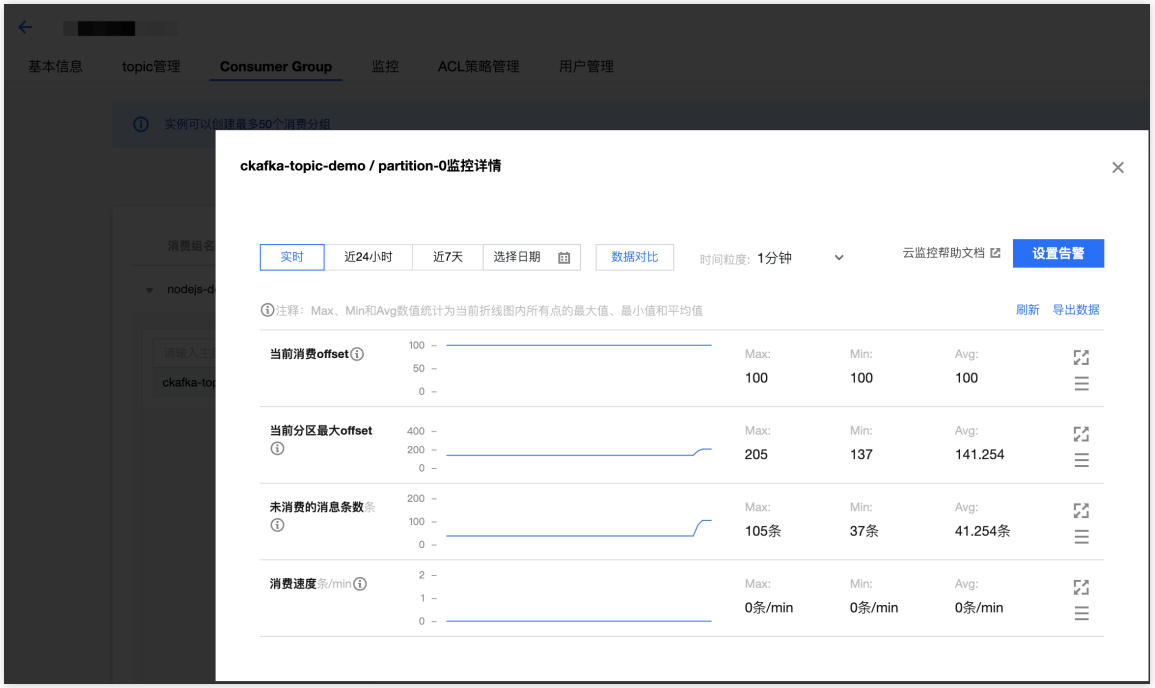
2. 执行以下命令消费消息。

```
    node consumer.js
```

3. 查看运行结果

```
[redacted] MB2 ~/Demos/ckafka-demo/nodejskafkademo/sasl ckafka_demo ± node consumer.js
[ 'gzip', 'snappy', 'sasl', 'regex', 'lz4' ]
0.9.5
{ sasl_plain_username: 'ckafka-[redacted]#ckafkademo',
  sasl_plain_password: 'ckafkademo123',
  bootstrap_servers:
   [ 'ckafka-[redacted]:ckafka.tencentcloudmq.com:6018' ],
  topic_name: 'ckafka-topic-demo',
  consumer_id: 'nodejs-demo' }
connect ok
{ value: null,
  size: 0,
  key: '1620379451745',
  topic: 'ckafka-topic-demo',
  offset: 137,
  partition: 0 }
{ value: null,
  size: 0,
  key: '1620379452745',
  topic: 'ckafka-topic-demo',
  offset: 138,
  partition: 0 }
```

4. 在 CKafka 控制台【Consumer Group】页面，选择对应的消费组，在主题名称输入 topic 名称，单击【查询详情】，查看消费详情。



5.

PHP SDK
01 VPC网络接入

操作场景

该任务以 PHP 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装 librdkafka
- 安装 PHP 5.6 或以上版本
- 安装 PEAR
- 下载demo

操作步骤

步骤一：添加 Rdkafka 扩展

1. 在 rdkafka 官方页面 查找最新的 rdkafka php 扩展包版本。

> 说明：
>
> 不同版本的包对 PHP 版本要求不同，这里仅以 4.1.2 为示例。

2. 登录linux服务器，安装 rdkafka 扩展。

```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# 安装成功会提示 "install ok" 和 "You should add "extension=rdkafka.so" to php.ini"
# 如果安装失败，请根据提示解决
# 安装成功后在 php.ini 添加 extension=rdkafka.so
# 执行 php --ini 后，Loaded Configuration File: 显示的就是 php.ini 所在位置
echo 'extension=rdkafka.so' >> /etc/php.ini
```

步骤二：准备配置

1. 将下载的demo中的phpkafkademo上传至linux服务器。
2. 登录linux服务器，进入phpkafkademo目录，修改 CKafkaSetting.php配置文件。

```
<?php
return [
    'bootstrap_servers' => 'bootstrap_servers1:port,bootstrap_servers2:port',
    'topic_name' => 'topic_name',
    'group_id' => 'php-demo',
];
```

参数	描述

参数	描述
bootstrap_servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。 
topic_name	Topic名称，您可以在控制台上【topic管理】页面复制。 
group_id	消费者的组 Id，您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。

步骤三：发送消息

1. 编写生产消息程序 Producer.php。

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\Conf();
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// Kafka producer 的 ack 有 3 种机制，分别说明如下：
// -1 或 all：Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后，才应答给 Producer 继续发送下一条（批）消息。
// 这种配置提供了最高的数据可靠性，只要有一个已同步的副本存活就不会有消息丢失。注意：这种配置不能确保所有的副本读写入该数据才返回，
// 可以配合 Topic 级别参数 min.insync.replicas 使用。
// 0：生产者不等待来自 broker 同步完成的确认，继续发送下一条（批）消息。这种配置生产性能最高，但数据可靠性最低
// （当服务器故障时可能会有数据丢失，如果 leader 已死但是 producer 不知情，则 broker 收不到消息）
// 1： 生产者在 leader 已成功收到的数据并得到确认后再发送下一条（批）消息。这种配置是在生产吞吐和数据可靠性之间的权衡
// （如果leader已死但是尚未复制，则消息可能丢失）
// 用户不显示配置时，默认值为1。用户根据自己的业务情况进行设置
$conf->set('acks', '1');
// 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
$conf->set('retries', '0');
// 发送请求失败时到下一次重试请求之间的时间
$conf->set('retry.backoff.ms', 100);
// producer 网络请求的超时时间。
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);
```

```
// 注册发送消息的回调
$conf->setDrMsgCb(function ($kafka, $message) {
    echo '【Producer】发送消息：message=' . var_export($message, true) . "\n";
});
// 注册发送消息错误的回调
$conf->setErrorCb(function ($kafka, $err, $reason) {
    echo "【Producer】发送消息错误：err=$err reason=$reason \n";
});

$producer = new RdKafka\Producer($conf);
// Debug 时请设置为 LOG_DEBUG
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// 生产消息并发送
for ($i = 0; $i < 5; $i++) {
    // RD_KAFKA_PARTITION_UA 让 kafka 自由选择分区
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
    $producer->poll(0);
}

while ($producer->getOutQLen() > 0) {
    $producer->poll(50);
}

echo "【Producer】消息发送成功\n";
```

2. 运行 Producer.php 发送消息。

```
php Producer.php
```

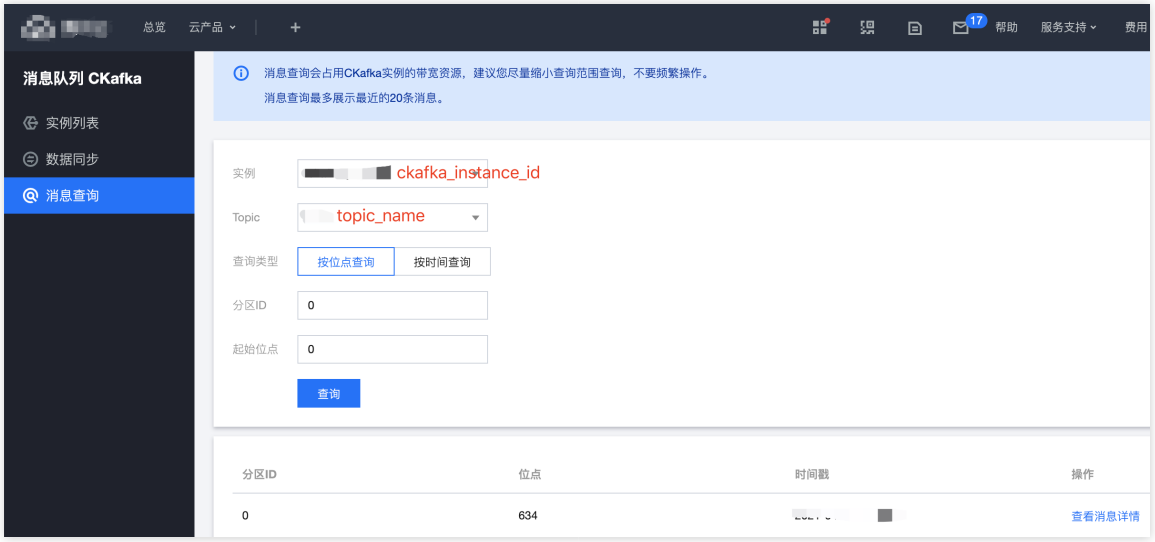
3. 查看运行结果，示例如下。

```
>【Producer】发送消息：message=RdKafka\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>【Producer】发送消息：message=RdKafka\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...
```

>【Producer】消息发送成功

4. 在 CKafka 控制台的【topic 管理】页面，选择对应的 Topic，点击【更多】>【消息查询】，查看刚刚发送的消息。



步骤四：消费消息

1. 编写消息订阅消费程序 Consumer.php。

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\Conf();
$conf->set('group.id', $setting['group_id']);
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，
// 认为该消费者故障失败，Broker 发起重新 Rebalance 过程。
$conf->set('session.timeout.ms', 10000);
// 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
$conf->set('request.timeout.ms', 305000);
// 设置客户端内部重试间隔。
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
// offset重置策略，请根据业务场景酌情设置。设置不当可能导致数据消费缺失。
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\KafkaConsumer($conf);
// Debug 时请设置为 LOG_DEBUG
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

$isConsuming = true;
while ($isConsuming) {
    $message = $consumer->consume(10 * 1000);
```

```
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            echo "【消费者】接收到消息：" . var_export($message, true) . "\n";
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "【消费者】等待信息消息中\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "【消费者】等待超时\n";
            $isConsuming = false;
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
```

2. 运行 Consumer.php 消费消息。

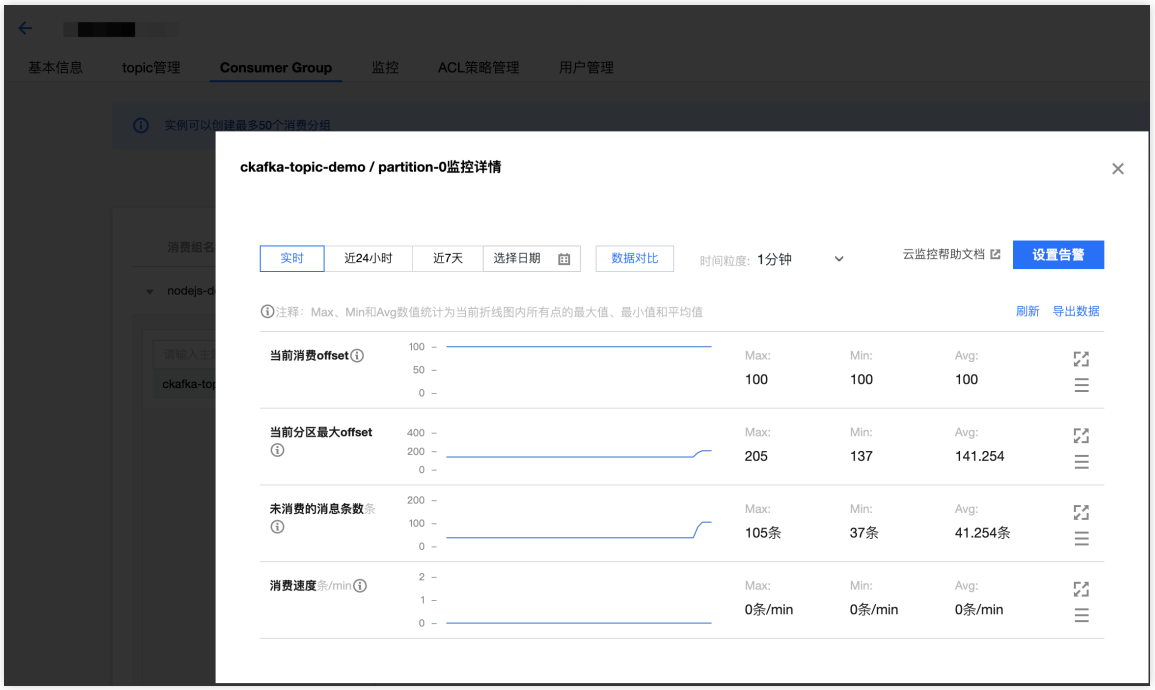
```
php Consumer.php
```

3. 查看运行结果。

```
>【消费者】接收到消息：RdKafka\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>【消费者】接收到消息：RdKafka\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...
```

4. 在 CKafka 控制台的【Consumer Group】页面，选择对应的消费者组名称，在主题名称输入 topic 名称，点击【查询详情】查看消费详情。


基本信息
topic管理
Consumer Group
监控
ACL策略管理
用户管理
ckafka-topic-demo / partition-0监控详情
实时
近24小时
近7天
选择日期
数据对比
时间粒度: 1分钟
云监控帮助文档
设置告警
注释：Max、Min和Avg数值统计为当前折线图内所有点的最大值、最小值和平均值
刷新
导出数据
当前消费offset
Max: 100
Min: 100
Avg: 100
当前分区最大offset
Max: 205
Min: 137
Avg: 141.254
未消费的消息条数
Max: 105条
Min: 37条
Avg: 41.254条
消费速度
Max: 0条/min
Min: 0条/min
Avg: 0条/min

Python SDK
01 VPC 网络接入

操作背景

该任务以 Python 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装 Python
- 安装 pip
- 下载demo

操作步骤

将下载的demo中的pythonkafkademo上传至linux服务器，登录linux服务器，进入pythonkafkademo目录。

步骤一：添加 Python 依赖库

执行以下命令安装：

、

```
pip install kafka-python
```

步骤二：生产消息

1. 修改生产消息程序producer.py中配置参数。

```
#coding:utf8
from kafka import KafkaProducer

producer = KafkaProducer(
  bootstrap_servers = ['$domainName:$port'],
  api_version = (0,10,0)
)
message = "Hello World! Hello Ckafka!"
msg = json.dumps(message).encode()
producer.send('topic_name',value = msg)
print("produce message " + message + " success.");
producer.close()
```

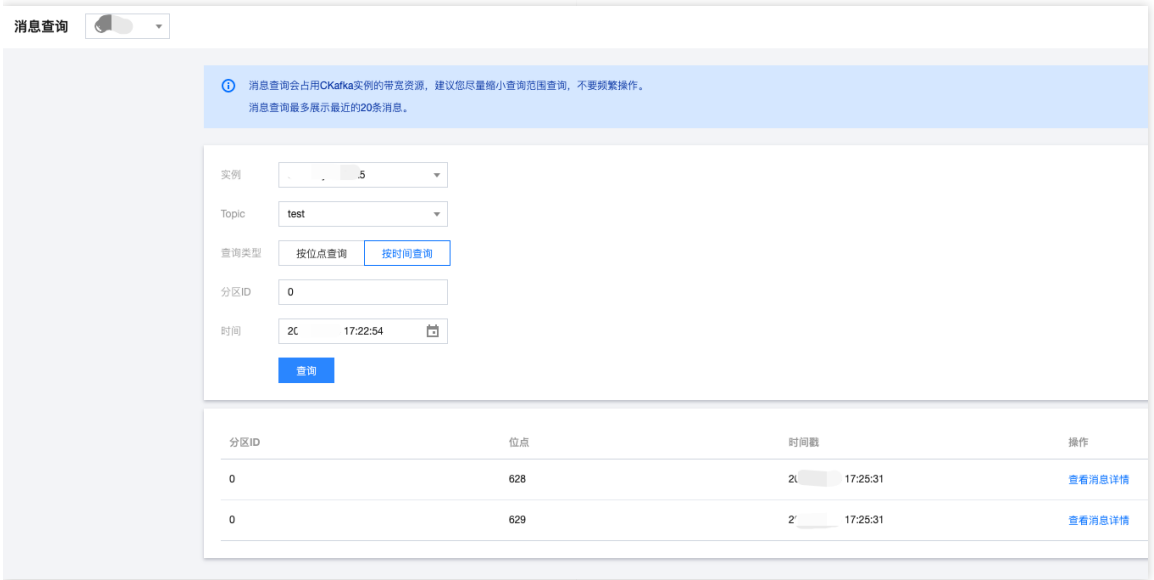
参数	描述
bootstrap_servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。

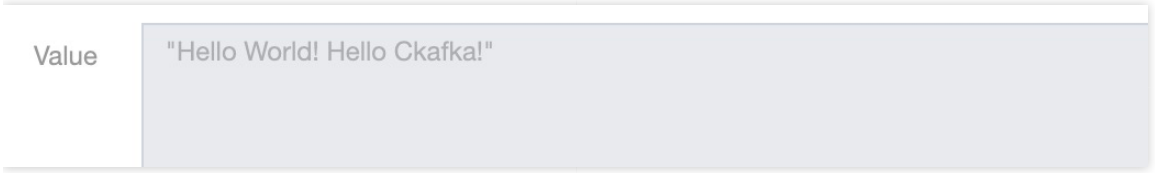
参数	描述
	
topic_name	Topic名称，您可以在控制台上【topic管理】页面复制。 

2. 编译并运行producer.py。
3. 查看运行结果。

```
[root@VM-8-16-centos sasl]# python3 producer.py
produce message Hello World! Hello Ckafka! success.
```

4. 在 CKafka 控制台的【topic管理】页面，选择对应的 Topic ， 点击【更多】>【消息查询】，查看刚刚发送的消息。





步骤三：消费消息

1. 修改消费消息程序consumer.py中配置参数。

```
#coding:utf8
from kafka import KafkaConsumer

consumer = KafkaConsumer(
    '$topic_name',
    group_id = "$group_id",
    bootstrap_servers = ['$domainName:$port'],
    api_version = (0,10,0)
)

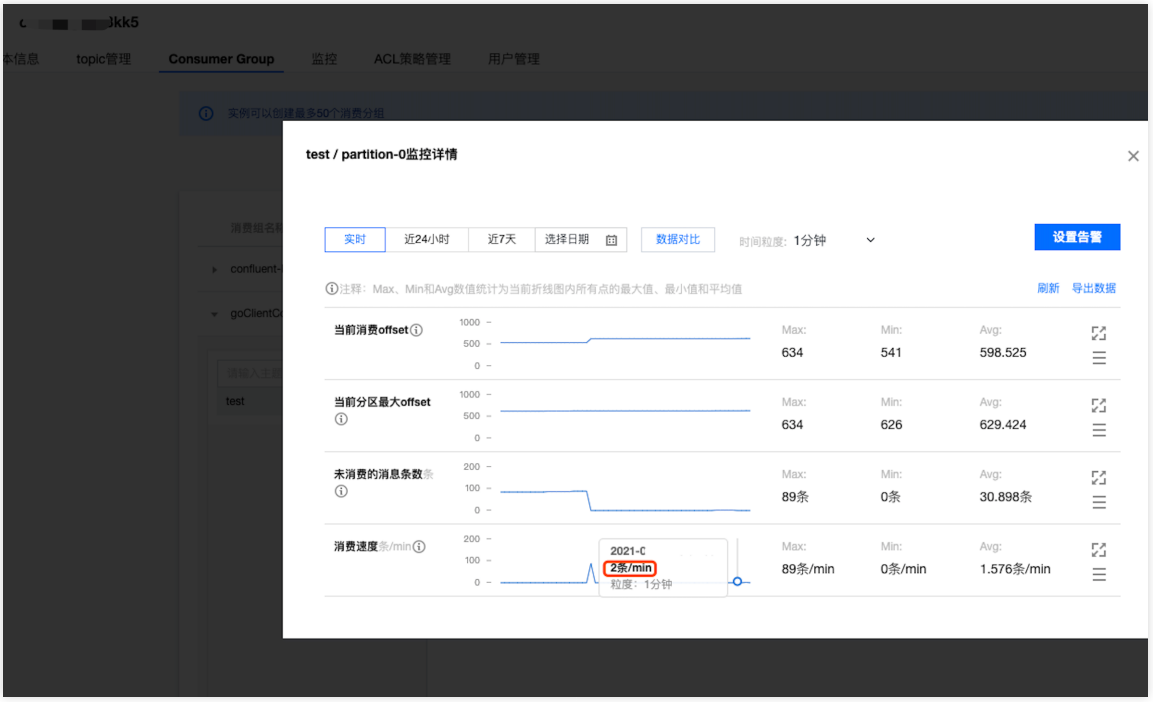
for message in consumer:
    print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" % (message.topic, message.partition, message.offset, message.value))
```

参数	描述
bootstrap_servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。 
group_id	消费者的组 ID，根据业务需求自定义
topic_name	Topic名称，您可以在控制台上【topic管理】页面复制。 

2. 编译并运行consumer.py。
3. 查看运行结果。

```
[root@VM-8-16-centos sasl]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello World! Hello Ckafka!"']
```

4. 在 CKafka 控制台的【Consumer Group】页面，选择对应的消费组名称，在主题名称输入 topic 名称，点击【查询详情】，查看消费详情。



C++ SDK
01 VPC 网络接入

操作场景

该任务以 C++ 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装 GCC
- 下载demo

操作步骤

步骤一：安装 C/C++ 依赖库

1. 将下载的demo中的cppkafkademo上传至linux服务器，
2. 登录linux服务器，安装 librdkafka。

步骤二：发送消息

1. 创建 producer.c 文件。

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2017, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
```

```
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple Apache Kafka producer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>

#include <librdkafka/rdkafka.h>

static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
    fclose(stdin); /* abort fgets() */
}

/**
 * @brief Message delivery report callback.
 *
 * This callback is called exactly once per message, indicating if
 * the message was succesfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err != RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
                       const rd_kafka_message_t *rkmessage, void *opaque) {
    if (rkmessage->err)
        fprintf(stderr, "%% Message delivery failed: %s\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%% Message delivered (%zd bytes, "
                "partition %"PRId32")\n",
            rkmessage->len, rkmessage->partition);

    /* The rkmessage is destroyed automatically by librdkafka */
}

int main (int argc, char **argv) {
    rd_kafka_t *rk;         /* Producer instance handle */
```

```
    rd_kafka_conf_t *conf;  /* Temporary configuration object */
    char errstr[512];       /* librdkafka API error reporting buffer */
    char buf[512];          /* Message value temporary buffer */
    const char *brokers;    /* Argument: broker list */
    const char *topic;      /* Argument: topic to produce to */
    const char *user;         /* Argument: sasl username */
    const char *passwd;     /* Argument: sasl password */

    /*
     * Argument validation
     */
    if (argc < 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>  \n Optional:username password\n", ar
gv[0]);
        return 1;
    }

    brokers = argv[1];
    topic   = argv[2];

    if(argc == 5) {
        user = argv[3];
        passwd = argv[4];
    }

    /*
     * Create Kafka client configuration place-holder
     */
    conf = rd_kafka_conf_new();

    /* Set bootstrap broker(s) as a comma-separated list of
     * host or host:port (default port 9092).
     * librdkafka will use the bootstrap brokers to acquire the full
     * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\n", errstr);
        return 1;
    }

    /* Set sasl config*/
    if( user && passwd ) {
        if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeof(errstr)) !=RD_KAFKA_CONF_OK) {
            fprintf(stderr,"%s\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
            fprintf(stderr,"%s\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_KAFKA_CONF_OK) {
            fprintf(stderr,"%s\n",errstr);
            return 1;
```

```
    }
    if(rd_kafka_conf_set(conf,"sasl.password",passwd,errstr,sizeof(errstr)) !=RD_KAFKA_CONF_OK) {
        fprintf(stderr,"%s\n",errstr);
        return 1;
    }

}
 */
//if(rd_kafka_conf_set(conf,"debug","none",errstr,sizeof(errstr))!= RD_KAFKA_CONF_OK) {  // 注意 线上环境需要谨慎评估开启debug。
    //fprintf(stderr,"%s\n",errstr);
    //return 1;
//}

if(rd_kafka_conf_set(conf,"acks","1",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}
if(rd_kafka_conf_set(conf,"request.timeout.ms","30000",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}
if(rd_kafka_conf_set(conf,"retries","3",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}

if(rd_kafka_conf_set(conf,"retry.backoff.ms","1000",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}

/* Set the delivery report callback.
 * This callback will be called once per message to inform
 * the application if delivery succeeded or failed.
 * See dr_msg_cb() above.
 * The callback is only triggered from rd_kafka_poll() and
 * rd_kafka_flush(). */
rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);

/*
 * Create producer instance.
 *
 * NOTE: rd_kafka_new() takes ownership of the conf object
 *       and the application must not reference it again after
 *       this call.
 */
rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
if (!rk) {
    fprintf(stderr,
            "%% Failed to create new producer: %s\n", errstr);
    return 1;
}
```

```
/* Signal handler for clean shutdown */
signal(SIGINT, stop);

fprintf(stderr,
        "%% Type some text and hit enter to produce message\n"
        "%% Or just hit enter to only serve delivery reports\n"
        "%% Press Ctrl-C or Ctrl-D to exit\n");

while (run && fgets(buf, sizeof(buf), stdin)) {
    size_t len = strlen(buf);
    rd_kafka_resp_err_t err;

    if (buf[len-1] == '\n') /* Remove newline */
        buf[--len] = '\0';

    if (len == 0) {
        /* Empty line: only serve delivery reports */
        rd_kafka_poll(rk, 0/*non-blocking */);
        continue;
    }

    /*
     * Send/Produce message.
     * This is an asynchronous call, on success it will only
     * enqueue the message on the internal producer queue.
     * The actual delivery attempts to the broker are handled
     * by background threads.
     * The previously registered delivery report callback
     * (dr_msg_cb) is used to signal back to the application
     * when the message has been delivered (or failed).
     */
    retry:
    err = rd_kafka_producev(
            /* Producer handle */
            rk,
            /* Topic name */
            RD_KAFKA_V_TOPIC(topic),
            /* Make a copy of the payload. */
            RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
            /* Message value and length */
            RD_KAFKA_V_VALUE(buf, len),
            /* Per-Message opaque, provided in
             * delivery report callback as
             * msg_opaque. */
            RD_KAFKA_V_OPAQUE(NULL),
            /* End sentinel */
            RD_KAFKA_V_END);

    if (err) {
        /*
         * Failed to *enqueue* message for producing.
         */
        fprintf(stderr,
                "%% Failed to produce to topic %s: %s\n",
```

```
                topic, rd_kafka_err2str(err));

        if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
            /* If the internal queue is full, wait for
             * messages to be delivered and then retry.
             * The internal queue represents both
             * messages to be sent and messages that have
             * been sent or failed, awaiting their
             * delivery report callback to be called.
             *
             * The internal queue is limited by the
             * configuration property
             * queue.buffering.max.messages */
            rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
            goto retry;
        }
    } else {
        fprintf(stderr, "%% Enqueued message (%zd bytes) "
                "for topic %s\n",
                len, topic);
    }

    /* A producer application should continually serve
     * the delivery report queue by calling rd_kafka_poll()
     * at frequent intervals.
     * Either put the poll call in your main loop, or in a
     * dedicated thread, or call it after every
     * rd_kafka_produce() call.
     * Just make sure that rd_kafka_poll() is still called
     * during periods where you are not producing any messages
     * to make sure previously produced messages have their
     * delivery report callback served (and any other callbacks
     * you register). */
    rd_kafka_poll(rk, 0/*non-blocking*/);
}

/* Wait for final messages to be delivered or fail.
 * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
 * waits for all messages to be delivered. */
fprintf(stderr, "%% Flushing final messages..\n");
rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

/* If the output queue is still not empty there is an issue
 * with producing messages to the clusters. */
if (rd_kafka_outq_len(rk) > 0)
    fprintf(stderr, "%% %d message(s) were not delivered\n",
            rd_kafka_outq_len(rk));

/* Destroy the producer instance */
rd_kafka_destroy(rk);

return 0;
}
```

2. 执行以下命令编译 producer.c。

gcc -lrdkafka ./producer.c -o producer

3. 执行以下命令发送消息。

./produce <broker> <topic>

参数	描述
broker	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。  img
topic	Topic名称，您可以在控制台上【topic管理】页面复制。  img

运行结果如下：

```
[root@VM_1_28_centos ~]# ./produce [redacted]:7779 test
% Type some text and hit enter to produce message
% Or just hit enter to only serve delivery reports
% Press Ctrl-C or Ctrl-D to exit
test
% Enqueued message (4 bytes) for topic test
test123
% Enqueued message (7 bytes) for topic test
% Message delivered (4 bytes, partition 1)
^C% Flushing final messages..
% Message delivered (7 bytes, partition 1)
```

4. 在 CKafka 控制台【topic 管理】页面，选择对应的 Topic，点击【更多】>【消息查询】，查看刚刚发送的消息。



步骤三：消费消息

1. 创建 consumer.c 文件。

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2019, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
```

```
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple high-level balanced Apache Kafka consumer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>

#include <librdkafka/rdkafka.h>

static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
}

/**
 * @returns 1 if all bytes are printable, else 0.
 */
static int is_printable (const char *buf, size_t size) {
    size_t i;

    for (i = 0 ; i < size ; i++)
        if (!isprint((int)buf[i]))
            return 0;

    return 1;
}

int main (int argc, char **argv) {
    rd_kafka_t *rk;         /* Consumer instance handle */
    rd_kafka_conf_t *conf;   /* Temporary configuration object */
    rd_kafka_resp_err_t err; /* librdkafka API error code */
    char errstr[512];        /* librdkafka API error reporting buffer */
    const char *brokers;     /* Argument: broker list */
    const char *groupid;     /* Argument: Consumer group id */
```

```
char **topics;          /* Argument: list of topics to subscribe to */
int topic_cnt;          /* Number of topics to subscribe to */
rd_kafka_topic_partition_list_t *subscription; /* Subscribed topics */
int i;

/*
 * Argument validation
 */
if (argc < 4) {
    fprintf(stderr,
            "%% Usage: "
            "%s <broker> <group.id> <topic1> <topic2>..\n",
            argv[0]);
    return 1;
}

brokers   = argv[1];
groupid   = argv[2];
topics    = &argv[3];
topic_cnt = argc - 3;

/*
 * Create Kafka client configuration place-holder
 */
conf = rd_kafka_conf_new();

/* Set bootstrap broker(s) as a comma-separated list of
 * host or host:port (default port 9092).
 * librdkafka will use the bootstrap brokers to acquire the full
 * set of brokers from the cluster. */
if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                      errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}

/* Set the consumer group id.
 * All consumers sharing the same group id will join the same
 * group, and the subscribed topic' partitions will be assigned
 * according to the partition.assignment.strategy
 * (consumer config property) to the consumers in the group. */
if (rd_kafka_conf_set(conf, "group.id", groupid,
                      errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}

/* If there is no previously committed offset for a partition
 * the auto.offset.reset strategy will be used to decide where
 * in the partition to start fetching messages.
 * By setting this to earliest the consumer will read all messages
 * in the partition if there was no previously committed offset. */
```

```
if (rd_kafka_conf_set(conf, "auto.offset.reset", "earliest",
                      errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}
 */
if(rd_kafka_conf_set(conf,"debug","all",errstr,sizeof(errstr))!= RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}

if(rd_kafka_conf_set(conf,"session.timeout.ms","10000",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}
if(rd_kafka_conf_set(conf,"heartbeat.interval.ms","3000",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr,"%s\n",errstr);
    return 1;
}

/*
 * Create consumer instance.
 *
 * NOTE: rd_kafka_new() takes ownership of the conf object
 *       and the application must not reference it again after
 *       this call.
 */
rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf, errstr, sizeof(errstr));
if (!rk) {
    fprintf(stderr,
            "%% Failed to create new consumer: %s\n", errstr);
    return 1;
}

conf = NULL; /* Configuration object is now owned, and freed,
                * by the rd_kafka_t instance. */

/* Redirect all messages from per-partition queues to
 * the main queue so that messages can be consumed with one
 * call from all assigned partitions.
 *
 * The alternative is to poll the main queue (for events)
 * and each partition queue separately, which requires setting
 * up a rebalance callback and keeping track of the assignment:
 * but that is more complex and typically not recommended. */
rd_kafka_poll_set_consumer(rk);

/* Convert the list of topics to a format suitable for librdkafka */
subscription = rd_kafka_topic_partition_list_new(topic_cnt);
for (i = 0 ; i < topic_cnt ; i++)
    rd_kafka_topic_partition_list_add(subscription,
                                      topics[i],
            /* the partition is ignored
```

```
             * by subscribe() */
                                RD_KAFKA_PARTITION_UA);

/* Subscribe to the list of topics */
err = rd_kafka_subscribe(rk, subscription);
if (err) {
    fprintf(stderr,
            "%% Failed to subscribe to %d topics: %s\n",
            subscription->cnt, rd_kafka_err2str(err));
    rd_kafka_topic_partition_list_destroy(subscription);
    rd_kafka_destroy(rk);
    return 1;
}

fprintf(stderr,
        "%% Subscribed to %d topic(s), "
        "waiting for rebalance and messages...\n",
        subscription->cnt);

rd_kafka_topic_partition_list_destroy(subscription);

/* Signal handler for clean shutdown */
signal(SIGINT, stop);

/* Subscribing to topics will trigger a group rebalance
 * which may take some time to finish, but there is no need
 * for the application to handle this idle period in a special way
 * since a rebalance may happen at any time.
 * Start polling for messages. */

while (run) {
    rd_kafka_message_t *rkm;

    rkm = rd_kafka_consumer_poll(rk, 100);
    if (!rkm)
        continue; /* Timeout: no message within 100ms,
                   *  try again. This short timeout allows
                   *  checking for `run` at frequent intervals.
                   */

    /* consumer_poll() will return either a proper message
     * or a consumer error (rkm->err is set). */
    if (rkm->err) {
        /* Consumer errors are generally to be considered
         * informational as the consumer will automatically
         * try to recover from all types of errors. */
        fprintf(stderr,
                "%% Consumer error: %s\n",
                rd_kafka_message_errstr(rkm));
        rd_kafka_message_destroy(rkm);
        continue;
    }

    /* Proper message. */
```

```
        printf("Message on %s [%"PRId32"] at offset %"PRId64":\n",
               rd_kafka_topic_name(rkm->rkt), rkm->partition,
               rkm->offset);

        /* Print the message key. */
        if (rkm->key && is_printable(rkm->key, rkm->key_len))
            printf(" Key: %.*s\n",
                   (int)rkm->key_len, (const char *)rkm->key);
        else if (rkm->key)
            printf(" Key: (%d bytes)\n", (int)rkm->key_len);

        /* Print the message value/payload. */
        if (rkm->payload && is_printable(rkm->payload, rkm->len))
            printf(" Value: %.*s\n",
                   (int)rkm->len, (const char *)rkm->payload);
        else if (rkm->payload)
            printf(" Value: (%d bytes)\n", (int)rkm->len);

        rd_kafka_message_destroy(rkm);
    }

    /* Close the consumer: commit final offsets and leave the group. */
    fprintf(stderr, "%% Closing consumer\n");
    rd_kafka_consumer_close(rk);

    /* Destroy the consumer */
    rd_kafka_destroy(rk);

    return 0;
}
```

2. 执行以下命令编译 consumer.c。

```
gcc -lrdkafka ./consumer.c -o consumer
```

3. 执行以下命令消费消息。

```
./consumer <broker> <group.id> <topic1> <topic2>..
```

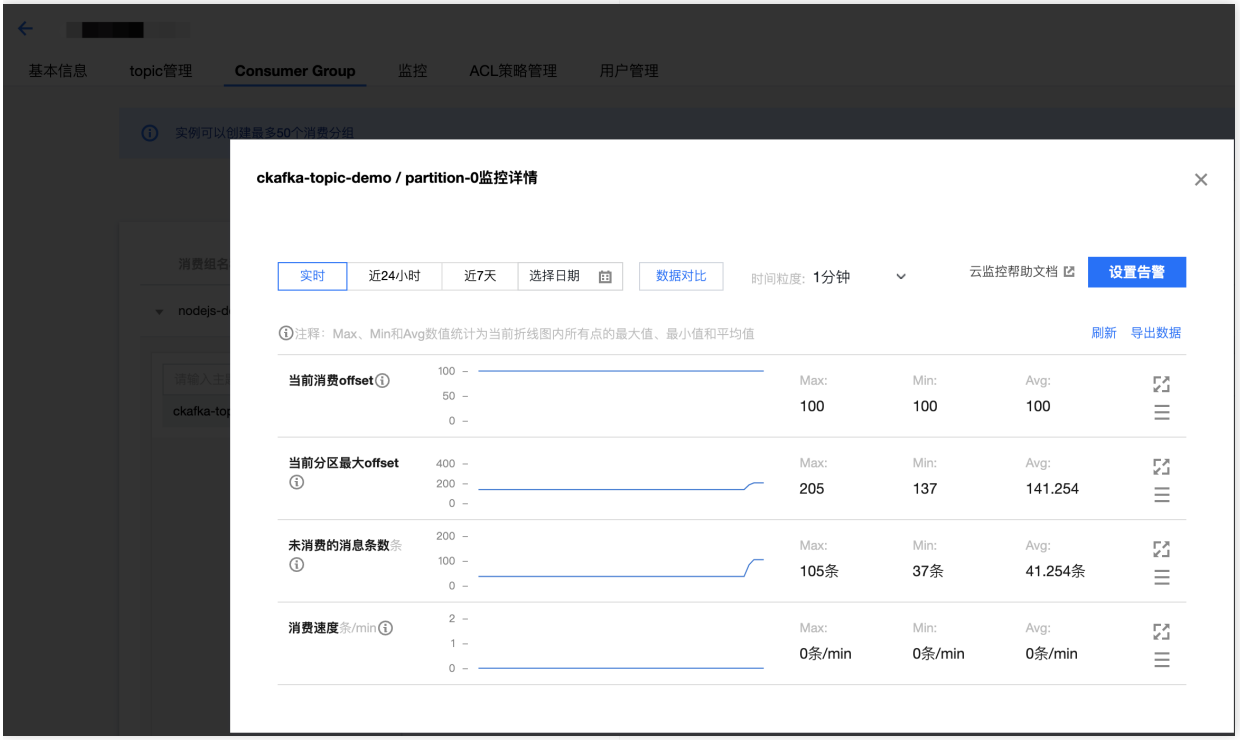
参数	描述

参数	描述
broker	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。  img
group.id	消费分组名称，您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。
topic1 topic2	Topic名称，您可以在控制台上【topic管理】页面复制。  img

运行结果如下：

```
[root@VM_1_28_centos ~]# ./produce          :7779 test
% Type some text and hit enter to produce message
% Or just hit enter to only serve delivery reports
% Press Ctrl-C or Ctrl-D to exit
test
% Enqueued message (4 bytes) for topic test
test123
% Enqueued message (7 bytes) for topic test
% Message delivered (4 bytes, partition 1)
^C% Flushing final messages..
% Message delivered (7 bytes, partition 1)
```

4. 在 CKafka 控制台【Consumer Group】页面，选择对应的消费者组，在主题名称输入 Topic 名称，单击【查询详情】查看消费详情。


基本信息
topic管理
Consumer Group
监控
ACL策略管理
用户管理
实例可以创建最多50个消费分组
ckafka-topic-demo / partition-0监控详情
实时
近24小时
近7天
选择日期
数据对比
时间粒度：1分钟
云监控帮助文档
设置告警
注释：Max、Min和Avg数值统计为当前折线图内所有点的最大值、最小值和平均值
刷新
导出数据
当前消费offset
100
50
0
Max:
100
Min:
100
Avg:
100
当前分区最大offset
400
200
0
Max:
205
Min:
137
Avg:
141.254
未消费的消息条数条
200
100
0
Max:
105条
Min:
37条
Avg:
41.254条
消费速度条/min
2
1
0
Max:
0条/min
Min:
0条/min
Avg:
0条/min

Java SDK
VPC网络接入

操作场景

该任务以 Java 客户端为例指导您使用VPC网络接入消息队列 CKafka 并收发消息。

前提条件

- 安装1.8或以上版本 JDK
- 安装2.5或以上版本 Maven
- 下载 Demo

操作步骤

步骤一：准备配置

1. 将下载的demo中的javakafkademo上传至linux服务器。

2. 登录linux服务器，进入javakafkademo目录，并配置相关参数。

3. 在 pom.xml 中添加以下依赖。

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-clients</artifactId>
   <version>0.10.2.2</version>
</dependency>
```

4. 创建消息队列 CKafka配置文件 kafka.properties。

```
## 配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。
bootstrap.servers=ckafka-xxxxxxxxxxxxxxxx
## 配置Topic，在控制台上topic管理页面复制。
topic=XXX
## 配置Consumer Group，您可以自定义设置
group.id=XXX
```

参数	说明
bootstrap.servers	接入网络，在控制台的实例详情页面【接入方式】模块的网络列复制。

参数	说明
	
topic	topic名称，您可以在控制台上【topic管理】页面复制。 
group.id	您可以自定义设置，demo运行成功后可以在【Consumer Group】页面看到该消费者。

5. 创建配置文件加载程序 CKafkaConfigurer.java。

```
public class CKafkaConfigurer {

    private static Properties properties;

    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //获取配置文件kafka.properties的内容。
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties"));
        } catch (Exception e) {
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

步骤二：发送消息

1. 编写生产消息程序 CKafkaProducerDemo.java。

```
public class CKafkaProducerDemo {

    public static void main(String args[]) {
        //加载kafka.properties。
```

```
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties properties = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));

        //消息队列Kafka版消息的序列化方式, 此处demo 使用的是StringSerializer。
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        //请求的最长等待时间。
        properties.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //设置客户端内部重试次数。
        properties.put(ProducerConfig.RETRIES_CONFIG, 5);
        //设置客户端内部重试间隔。
        properties.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        //构造Producer对象。
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

        //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 ckafka msg value"; //消息的内容。

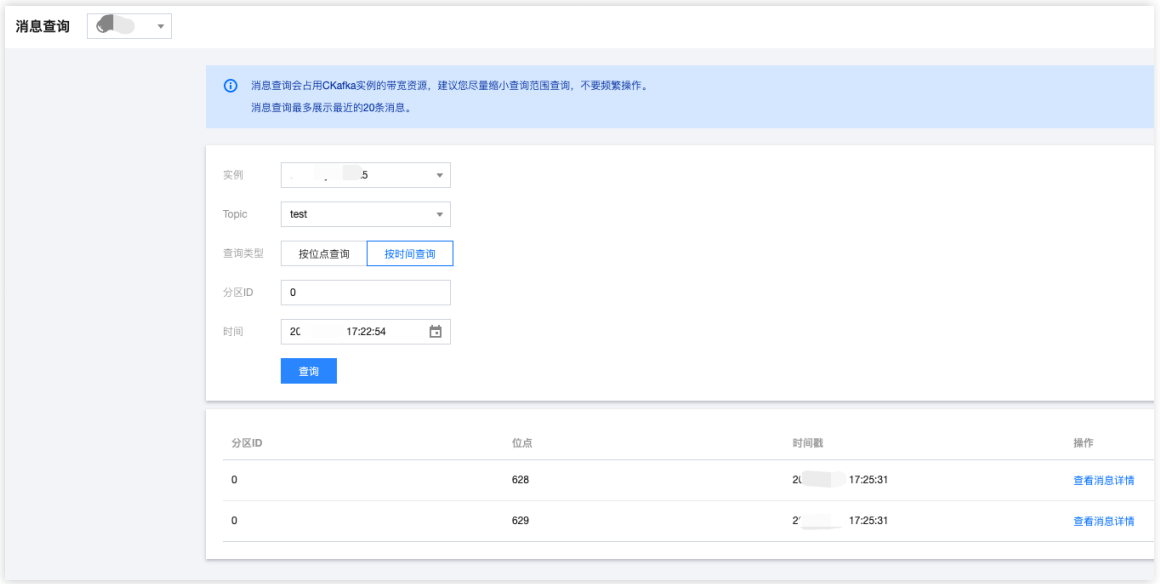
        try {
            //批量获取Future对象可以加快速度, 但注意, 批量不要太大。
            List<Future<RecordMetadata>> futureList = new ArrayList<>(128);
            for (int i = 0; i < 10; i++) {
                //发送消息，并获得一个Future对象。
                ProducerRecord<String, String> kafkaMsg = new ProducerRecord<>(topic,
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMsg);
                futureList.add(metadataFuture);

            }
            producer.flush();
            for (Future<RecordMetadata> future : futureList) {
                //同步获得Future对象的结果。
                RecordMetadata recordMetadata = future.get();
                System.out.println("produce send ok: " + recordMetadata.toString());
            }
        } 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");
        }
    }
}
```

2. 编译并运行 CKafkaProducerDemo.java 发送消息。
3. 运行结果。

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. 在 CKafka 控制台的【topic管理】页面，选择对应的 topic ，点击【更多】>【消息查询】，查看刚刚发送的消息。



步骤三：消费消息

1. 创建 Consumer 订阅消息程序 CKafkaConsumerDemo.java。

```
public class CKafkaConsumerDemo {

    public static void main(String args[]) {
        //加载kafka.properties。
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        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"));
        //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Consumer 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");
        //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。
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(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<>();
        //如果需要订阅多个Topic，则在这里添加进去即可。
        //每个Topic需要先在控制台进行创建。
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic : topics) {
```

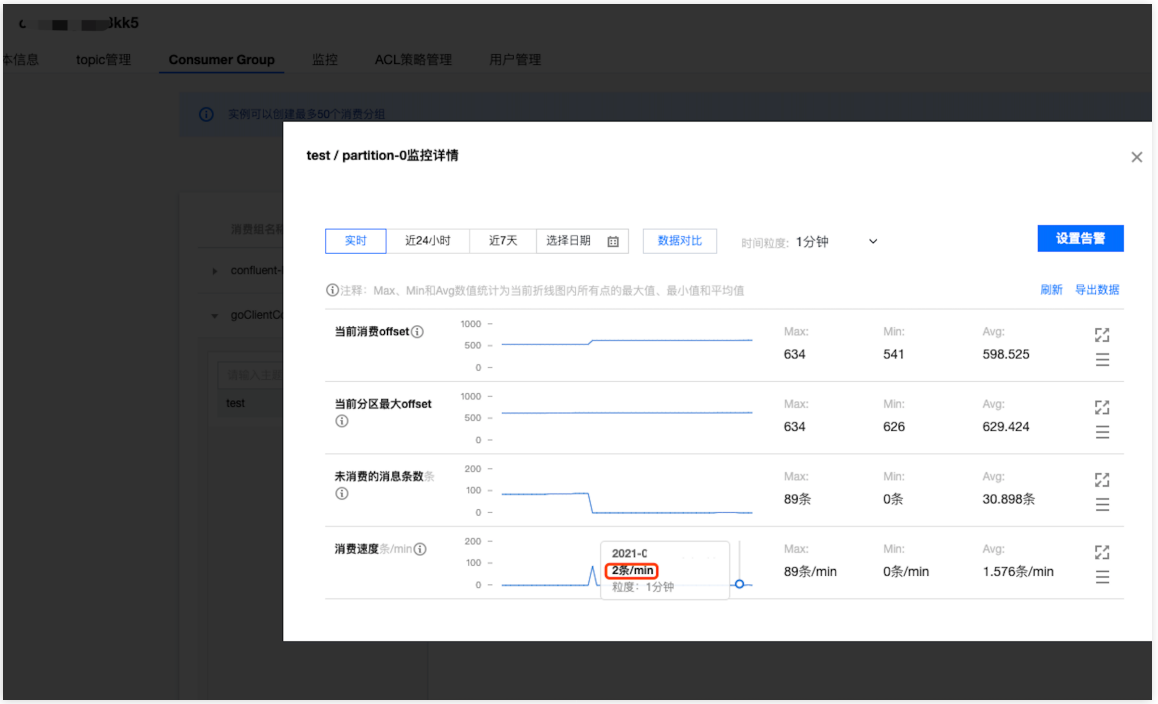
```
            subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);

        //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) {
                System.out.println("consumer error!");
            }
        }
    }
}
```

2. 编译并运行CKafkaConsumerDemo.java 消费消息。
3. 运行结果。

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. 在CKafka 控制台的【Consumer Group】页面，选择对应的消费组名称，在主题名称输入 topic 名称，点击【查询详情】，查看消费详情。



API文档
消息队列（CKafka）（ckafka）
版本（2019-08-19）
API 概览

API版本

V3

ACL相关接口

接口名称	接口功能
CreateAcl	添加 ACL 策略
CreateUser	添加用户
DeleteAcl	删除ACL
DeleteUser	删除用户
DescribeACL	枚举ACL
DescribeUser	查询用户信息
ModifyPassword	修改密码

主题相关接口

接口名称	接口功能
CreatePartition	增加主题分区
CreateTopic	创建主题
CreateTopicIpWhiteList	创建主题IP白名单
DeleteTopicIpWhiteList	删除主题IP白名单

接口名称	接口功能
DescribeTopic	获取主题列表
DescribeTopicAttributes	获取主题属性
DescribeTopicDetail	获取主题列表详情
FetchMessageByOffset	查询消息
FetchMessageListByOffset	根据位点查询消息列表
FetchMessageListByTimestamp	根据时间戳查询消息列表
ModifyTopicAttributes	设置主题属性

其他接口

接口名称	接口功能
DescribeAppInfo	查询用户列表
DescribeConsumerGroup	查询消费分组信息
DescribeGroup	枚举消费分组(精简版)
DescribeOrderConfig	查看订单配置

实例相关接口

接口名称	接口功能
CreateToken	创建token
DescribeGroupInfo	获取消费分组信息
DescribeGroupOffsets	获取消费分组offset
DescribeInstanceAttributes	获取实例属性
DescribeInstances	获取实例列表
DescribeInstancesDetail	获取实例列表详情
ModifyGroupOffsets	设置Groups 消费分组offset

接口名称	接口功能
ModifyInstanceAttributes	设置实例属性

访问控制相关接口

接口名称	接口功能
AuthorizeToken	授权token

调用方式
接口签名v1

TCloudFinanceZone API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（ Signature ）以验证请求者身份。
签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。
安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录TCloudFinanceZone管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. 生成签名串

有了安全凭证SecretId 和 SecretKey后，就可以生成签名串了。以下是生成签名串的详细过程：

假设用户的 SecretId 和 SecretKey 分别是：

- SecretId: AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
- SecretKey: Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

注意：这里只是示例，请根据用户实际申请的 SecretId 和 SecretKey 进行后续操作！

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥Id	AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	shjr
InstanceIds.0	待查询的实例ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

2.1. 对参数排序

首先对所有请求参数按参数名的字典序（ ASCII 码）升序排序。注意：1）只按参数名进行排序，参数值保持对应即可，不参与比大小；2）按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 php 中的 ksort 函数。上述示例参数的排序结果如下:

```
{
    'Action' : 'DescribeInstances',
    'InstanceIds.0' : 'ins-09dx96dg',
    'Limit' : 20,
    'Nonce' : 11886,
    'Offset' : 0,
    'Region' : 'shjr',
    'SecretId' : 'AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE',
    'Timestamp' : 1465185768,
    'Version': '2017-03-12',
}
```

使用其它程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2.2. 拼接请求字符串

此步骤生成请求字符串。
将把上一步排序好的请求参数格式化成“参数名称”=“参数值”的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances 。
注意：“参数值”为原始值而非url编码后的值。

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.3. 拼接签名原文字符串

此步骤生成签名原文字符串。
签名原文字符串由以下几个参数构成:

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表(DescribeInstances)的请求域名为：cvm.finance.cloud.tencent.com。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 / 。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为: 请求方法 + 请求主机 +请求路径 + ? + 请求字符串

示例的拼接结果为：

```
GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.4. 生成签名串

此步骤生成签名串。
首先使用 HMAC-SHA1 算法对上一步中获得的签名原文字符串进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。

具体代码如下，以 PHP 语言为例:

```
$secretKey = 'Gu5t9xGARNpq86cd98joQYCN3EXAMPLE';
$srcStr = 'GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12';
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr;
```

最终得到的签名串为:

```
EliP9YW3pW28FpsEdkXt/+WcGeI=
```

使用其它程序设计语言开发时，可用上面示例中的原文进行签名验证，得到的签名串与例子中的一致即可。

3. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。

如上一步生成的签名串为 EliP9YW3pW28FpsEdkXt/+WcGeI= ，最终得到的签名串请求参数（Signature）为：EliP9YW3pW28FpsEdkXt%2f%2bWcGeI%3d，它将用于生成最终的请求 URL。

注意：如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。

注意：有些编程语言的 http 库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。

注意：其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

4. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

5. 签名演示

在实际调用 API 3.0 时，推荐使用配套的TCloudFinanceZone SDK 3.0 ，SDK 封装了签名的过程，开发时只关注产品提供的具体接口即可。详细信息参见 SDK 中心。当前支持的编程语言有：

- Python
- Java

- PHP
- Go
- Node

为了更清楚的解释签名过程，下面以实际编程语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

最终输出的 url 可能为： https://cvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=shjr&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Signature=EliP9YW3pW28FpsEdkXt%2F%2BWcGeI%3D&Timestamp=1465185768&Version=2017-03-12

注意：由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。

注意：在下面的示例中，不同编程语言，甚至同一语言每次执行得到的 url 可能都有所不同，表现为参数的顺序不同，但这并不影响正确性。只要所有参数都在，且签名计算正确即可。

注意：以下代码仅适用于 API 3.0，不能直接用于其他的签名流程，即使是旧版的 API ，由于存在细节差异也会导致签名计算错误，请以对应的实际文档为准。

Java

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Random;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class CloudAPIDemo {
    private final static String CHARSET = "UTF-8";

    public static String sign(String s, String key, String method) throws Exception {
        Mac mac = Mac.getInstance(method);
        SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(CHARSET), mac.getAlgorithm());
        mac.init(secretKeySpec);
        byte[] hash = mac.doFinal(s.getBytes(CHARSET));
        return DatatypeConverter.printBase64Binary(hash);
    }

    public static String getStringToSign(TreeMap<String, Object> params) {
        StringBuilder s2s = new StringBuilder("GETcvm.finance.cloud.tencent.com/?");
```

```
        // 签名时要求对参数进行字典排序，此处用TreeMap保证顺序
        for (String k : params.keySet()) {
            s2s.append(k).append("=").append(params.get(k).toString()).append("&");
        }
        return s2s.toString().substring(0, s2s.length() - 1);
    }

    public static String getUrl(TreeMap<String, Object> params) throws UnsupportedEncodingException
{
        StringBuilder url = new StringBuilder("https://cvm.finance.cloud.tencent.com/?");
        // 实际请求的url中对参数顺序没有要求
        for (String k : params.keySet()) {
            // 需要对请求串进行urlencode，由于key都是英文字母，故此处仅对其value进行urlencode
            url.append(k).append("=").append(URLEncoder.encode(params.get(k).toString(), CHARSET)).app
end("&");
        }
        return url.toString().substring(0, url.length() - 1);
    }

    public static void main(String[] args) throws Exception {
        TreeMap<String, Object> params = new TreeMap<String, Object>(); // TreeMap可以自动排序
        // 实际调用时应当使用随机数，例如：params.put("Nonce", new Random().nextInt(java.lang.Intege
r.MAX_VALUE));
        params.put("Nonce", 11886); // 公共参数
        // 实际调用时应当使用系统当前时间，例如：  params.put("Timestamp", System.currentTimeMillis() /
1000);
        params.put("Timestamp", 1465185768); // 公共参数
        params.put("SecretId", "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"); // 公共参数
        params.put("Action", "DescribeInstances"); // 公共参数
        params.put("Version", "2017-03-12"); // 公共参数
        params.put("Region", "shjr"); // 公共参数
        params.put("Limit", 20); // 业务参数
        params.put("Offset", 0); // 业务参数
        params.put("InstanceIds.0", "ins-09dx96dg"); // 业务参数
        params.put("Signature", sign(getStringToSign(params), "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE
", "HmacSHA1")); // 公共参数
        System.out.println(getUrl(params));
    }
}
```

Python

注意：如果是在 Python 2 环境中运行，需要先安装 requests 依赖包： pip install requests 。

```
# -*- coding: utf8 -*-
import base64
```

```
import hashlib
import hmac
import time

import requests

secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

def get_string_to_sign(method, endpoint, params):
    s = method + endpoint + "/?"
    query_str = "&".join("%s=%s" % (k, params[k]) for k in sorted(params))
    return s + query_str

def sign_str(key, s, method):
    hmac_str = hmac.new(key.encode("utf8"), s.encode("utf8"), method).digest()
    return base64.b64encode(hmac_str)

if __name__ == '__main__':
    endpoint = "cvm.finance.cloud.tencent.com"
    data = {
        'Action' : 'DescribeInstances',
        'InstanceIds.0' : 'ins-09dx96dg',
        'Limit' : 20,
        'Nonce' : 11886,
        'Offset' : 0,
        'Region' : 'shjr',
        'SecretId' : secret_id,
        'Timestamp' : 1465185768, # int(time.time())
        'Version': '2017-03-12'
    }
    s = get_string_to_sign("GET", endpoint, data)
    data["Signature"] = sign_str(secret_key, s, hashlib.sha1)
    print(data["Signature"])
    # 此处会实际调用，成功后可能产生计费
    # resp = requests.get("https://" + endpoint, params=data)
    # print(resp.url)
```

接口签名v3

TCloudFinanceZone API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（Signature）以验证请求者身份。
签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。
安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录TCloudFinanceZone管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. TC3-HMAC-SHA256 签名方法

注意：对于GET方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式默认所有业务接口均支持，multipart 格式只有特定业务接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。

下面以云服务器查询广州区实例列表作为例子，分步骤介绍签名的计算过程。我们仅用到了查询实例列表的两个参数：Limit 和 Offset，使用 GET 方法调用。

假设用户的 SecretId 和 SecretKey 分别是：AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE 和 Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

2.1. 拼接规范请求串

按如下格式拼接规范请求串（CanonicalRequest）：

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
    SignedHeaders + '\n' +
    HashedRequestPayload
```

- HTTPRequestMethod：HTTP 请求方法（GET、POST ），本示例中为 GET；
- CanonicalURI：URI 参数，API 3.0 固定为正斜杠（/）；
- CanonicalQueryString：发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串，对于 GET 请求，则为 URL 中问号（?）后面的字符串内容，本示例取值为：Limit=10&Offset=0。注意：CanonicalQueryString 需要经过 URL 编码。
- CanonicalHeaders：参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部参与签名以提高自身请求的唯一性和安全性。拼接规则：1）头部 key 和 value 统一转成小写，并去掉首尾空格，按照 key:value\n 格式拼接；2）多个头部，按照头部 key（小写）的字典排序进行拼接。此例中为： content-type:application/x-www-form-urlencoded\nhost:cvm.finance.cloud.tencent.com\n
- SignedHeaders：参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包含的头部内容是一一对应的。content-type 和 host 为必选头部。拼接规则：1）头部 key 统一转成小写；2）多个头部 key（小写）按照字典排序进行拼接，并且以分号（;）分隔。此例中为： content-type;host
- HashedRequestPayload：请求正文的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(RequestPayload)))，对 HTTP 请求整个正文 payload 做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。注意：对于 GET 请求，RequestPayload 固定为空字符串，对于 POST 请求，RequestPayload 即为 HTTP 请求正文 payload。

根据以上规则，示例中得到的规范请求串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
GET
/
Limit=10&Offset=0
content-type:application/x-www-form-urlencoded
host:cvm.finance.cloud.tencent.com

content-type;host
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

2.2. 拼接待签名字符串

按如下格式拼接待签名字符串：

```
StringToSign =
    Algorithm + \n +
```

```
RequestTimestamp + \n +
CredentialScope + \n +
HashedCanonicalRequest
```

- Algorithm：签名算法，目前固定为 TC3-HMAC-SHA256；
- RequestTimestamp：请求时间戳，即请求头部的 X-TC-Timestamp 取值，如上示例请求为 1539084154；
- CredentialScope：凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service 为产品名，必须与调用的产品域名一致，例如 cvm。如上示例请求，取值为 2018-10-09/cvm/tc3_request；
- HashedCanonicalRequest：前述步骤拼接所得规范请求串的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。

注意：

> 1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。
> 2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败（返回签名过期错误）。

根据以上规则，示例中得到的待签名字符串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
TC3-HMAC-SHA256
1539084154
2018-10-09/cvm/tc3_request
91c9c192c14460df6c1ffc69e34e6c5e90708de2a6d282cccf957dbf1aa7f3a7
```

2.3. 计算签名

1）计算派生签名密钥，伪代码如下

```
SecretKey = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"
SecretDate = HMAC_SHA256("TC3" + SecretKey, Date)
SecretService = HMAC_SHA256(SecretDate, Service)
SecretSigning = HMAC_SHA256(SecretService, "tc3_request")
```

- SecretKey：原始的 SecretKey；
- Date：即 Credential 中的 Date 字段信息，如上示例，为2018-10-09；
- Service：即 Credential 中的 Service 字段信息，如上示例，为 cvm；

2）计算签名，伪代码如下

```
Signature = HexEncode(HMAC_SHA256(SecretSigning, StringToSign))
```

- SecretSigning：即以上计算得到的派生签名密钥；
- StringToSign：即步骤2计算得到的待签名字符串；

2.4. 拼接 Authorization

按如下格式拼接 Authorization：

```
Authorization =
    Algorithm + ' ' +
    'Credential=' + SecretId + '/' + CredentialScope + ', ' +
    'SignedHeaders=' + SignedHeaders + ', '
    'Signature=' + Signature
```

- Algorithm：签名方法，固定为 TC3-HMAC-SHA256；
- SecretId：密钥对中的 SecretId；
- CredentialScope：见上文，凭证范围；
- SignedHeaders：见上文，参与签名的头部信息；
- Signature：签名值

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
```

最终完整的调用信息如下：

```
https://cvm.finance.cloud.tencent.com/?Limit=10&Offset=0

Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2018-10-09/cvm/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
Content-Type: application/x-www-form-urlencoded
Host: cvm.finance.cloud.tencent.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1539084154
X-TC-Region: shjr
```

3. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

4. 签名演示

Java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.bind.DatatypeConverter;

import org.apache.commons.codec.digest.DigestUtils;

public class CloudAPITC3Demo {
    private final static String CHARSET = "UTF-8";
    private final static String ENDPOINT = "cvm.finance.cloud.tencent.com";
    private final static String PATH = "/";
    private final static String SECRET_ID = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE";
    private final static String SECRET_KEY = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE";
    private final static String CT_X_WWW_FORM_URLENCODED = "application/x-www-form-urlencode
d";
    private final static String CT_JSON = "application/json";
```

```
    private final static String CT_FORM_DATA = "multipart/form-data";

    public static byte[] sign256(byte[] key, String msg) throws Exception {
        Mac mac = Mac.getInstance("HmacSHA256");
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
        mac.init(secretKeySpec);
        return mac.doFinal(msg.getBytes(CHARSET));
    }

    public static void main(String[] args) throws Exception {
        String service = "cvm";
        String host = "cvm.finance.cloud.tencent.com";
        String region = "shjr";
        String action = "DescribeInstances";
        String version = "2017-03-12";
        String algorithm = "TC3-HMAC-SHA256";
        String timestamp = "1539084154";
        //String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        // 注意时区，否则容易出错
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

        // ************ 步骤 1：拼接规范请求串 ************
        String httpRequestMethod = "GET";
        String canonicalUri = "/";
        String canonicalQueryString = "Limit=10&Offset=0";
        String canonicalHeaders = "content-type:application/x-www-form-urlencoded\n" + "host:" + host
+ "\n";
        String signedHeaders = "content-type;host";
        String hashedRequestPayload = DigestUtils.sha256Hex("");
        String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryStri
ng + "\n"
                + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
        System.out.println(canonicalRequest);

        // ************ 步骤 2：拼接待签名字符串 ************
        String credentialScope = date + "/" + service + "/" + "tc3_request";
        String hashedCanonicalRequest = DigestUtils.sha256Hex(canonicalRequest.getBytes(CHARSET));
        String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" + hashedCan
onicalRequest;
        System.out.println(stringToSign);

        // ************ 步骤 3：计算签名 ************
        byte[] secretDate = sign256(("TC3" + SECRET_KEY).getBytes(CHARSET), date);
        byte[] secretService = sign256(secretDate, service);
        byte[] secretSigning = sign256(secretService, "tc3_request");
```

```
        String signature = DatatypeConverter.printHexBinary(sign256(secretSigning, stringToSign)).toLow
erCase();
        System.out.println(signature);

        // ************* 步骤 4：拼接 Authorization *************
        String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
                + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
        System.out.println(authorization);

        TreeMap<String, String> headers = new TreeMap<String, String>();
        headers.put("Authorization", authorization);
        headers.put("Host", host);
        headers.put("Content-Type", CT_X_WWW_FORM_URLENCODED);
        headers.put("X-TC-Action", action);
        headers.put("X-TC-Timestamp", timestamp);
        headers.put("X-TC-Version", version);
        headers.put("X-TC-Region", region);
    }
}
```

Python

```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

service = "cvm"
host = "cvm.finance.cloud.tencent.com"
endpoint = "https://" + host
region = "shjr"
action = "DescribeInstances"
version = "2017-03-12"
algorithm = "TC3-HMAC-SHA256"
timestamp = 1539084154
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 10, "Offset": 0}

# ************* 步骤 1：拼接规范请求串 *************
http_request_method = "GET"
canonical_uri = "/"
canonical_querystring = "Limit=10&Offset=0"
ct = "x-www-form-urlencoded"
```

```
payload = ""
if http_request_method == "POST":
    canonical_querystring = ""
    ct = "json"
    payload = json.dumps(params)
canonical_headers = "content-type:application/%s\nhost:%s\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\n" +
                     canonical_uri + "\n" +
                     canonical_querystring + "\n" +
                     canonical_headers + "\n" +
                     signed_headers + "\n" +
                     hashed_request_payload)
print(canonical_request)

# ************* 步骤 2：拼接待签名字符串 *************
credential_scope = date + "/" + service + "/" + "tc3_request"
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\n" +
                  str(timestamp) + "\n" +
                  credential_scope + "\n" +
                  hashed_canonical_request)
print(string_to_sign)

# ************* 步骤 3：计算签名 *************
# 计算签名摘要函数
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()
print(signature)

# ************* 步骤 4：拼接 Authorization *************
authorization = (algorithm + " " +
                 "Credential=" + secret_id + "/" + credential_scope + ", " +
                 "SignedHeaders=" + signed_headers + ", " +
                 "Signature=" + signature)
print(authorization)

# 公共参数添加到请求头部
headers = {
    "Authorization": authorization,
    "Host": host,
    "Content-Type": "application/%s" % ct,
```

```
    "X-TC-Action": action,
    "X-TC-Timestamp": str(timestamp),
    "X-TC-Version": version,
    "X-TC-Region": region,
}
```

请求结构

1. 服务地址

地域（Region）是指物理的数据中心的地理区域。TCloudFinanceZone交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 查询地域列表 查看完成的地域列表。

2. 通信协议

TCloudFinanceZone API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法:

- POST（推荐）
- GET

POST 请求支持的 Content-Type 类型：

- application/json（推荐），必须使用 TC3-HMAC-SHA256 签名方法。
- application/x-www-form-urlencoded，必须使用 HmacSHA1 或 HmacSHA256 签名方法。
- multipart/form-data（仅部分接口支持），必须使用 TC3-HMAC-SHA256 签名方法。

GET 请求的请求包大小不得超过 32 KB。POST 请求使用签名方法为 HmacSHA1、HmacSHA256 时不得超过 1 MB。POST 请求使用签名方法为 TC3-HMAC-SHA256 时支持 10 MB。

4. 字符编码

均使用UTF-8编码。

返回结果

正确返回结果

以云服务器的接口查看实例状态列表 (DescribeInstancesStatus) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：

```
{
    "Response": {
        "TotalCount": 0,
        "InstanceStatusSet": [],
        "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
    }
}
```

- Response 及其内部的 RequestId 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
- 除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 TotalCount 和 InstanceStatusSet 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 TotalCount 在此情况下的返回值为 0， InstanceStatusSet 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：

```
{
    "Response": {
        "Error": {
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please check your signature is cor
rect."
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

- Error 的出现代表着该请求调用失败。Error 字段连同其内部的 Code 和 Message 字段在调用失败时是必定返回的。
- Code 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。

- Message 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。

错误码	错误描述
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

公共参数

公共参数是用于标识用户和接口鉴权目的的参数，如非必要，在每个接口单独的接口文档中不再对这些参数进行说明，但每次请求均需要携带这些参数，才能正常发起请求。

签名方法 v3

使用 TC3-HMAC-SHA256 签名方法时，公共参数需要统一放到 HTTP Header 请求头部中，如下：

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024 其中， - TC3-HMAC-SHA256：签名方法，目前固定取该值； - Credential：签名凭证，AKIDEXAMPLE 是 SecretId；Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service为产品名，必须与调用的产品域名一致，例如cvm； - SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部； - Signature：签名摘要。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 v1

使用 HmacSHA1 和 HmacSHA256 签名方法时，公共参数需要统一放到请求串中，如下

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。

参数名称	类型	必选	描述
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在云API密钥上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey ，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见接口鉴权文档。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

地域（Region）是指物理的数据中心的地理区域。TCloudFinanceZone交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 查询地域列表 查看完成的地域列表。

ACL相关接口
添加 ACL 策略

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

添加 ACL 策略

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-09 15:21:34。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateAcl
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例id信息 示例值： ckafka-test
ResourceType	是	否	Int64	Acl资源类型，(0:UNKNOWN，1:ANY，2:TOPIC，3:GROUP，4:CLUSTER，5:TRANSACTIONAL_ID)，当前只有TOPIC，其它字段用于后续兼容开源kafka的acl时使用 示例值： 2
ResourceName	是	否	String	资源名称，和resourceType相关，如当resourceType为TOPIC时，则该字段表示topic名称，当resourceType为GROUP时，该字段表示group名称 示例值： Topic1
Operation	是	否	Int64	Acl操作方式，(0:UNKNOWN，1:ANY，2:ALL，3:READ，4:WRITE，5:CREATE，6:DELETE，7:ALTER，

参数名称	必选	允许NULL	类型	描述
				8:DESCRIBE，9:CLUSTER_ACTION，10:DESCRIBE_CONFIGS，11:ALTER_CONFIGS) 示例值： 2
PermissionType	是	否	Int64	权限类型，(0:UNKNOWN，1:ANY，2:DENY，3:ALLOW)，当前ckakfa支持ALLOW(相当于白名单)，其它用于后续兼容开源kafka的acl时使用 示例值： 2
Host	否	否	String	默认为*，表示任何host都可以访问，当前ckafka不支持host为*，但是后面开源kafka的产品化会直接支持 示例值： 1.1.1.1
Principal	否	否	String	用户列表，默认为*，表示任何user都可以访问，当前用户只能是用户列表中包含的用户 示例值： User:user1

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。

错误码	描述
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

添加用户

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

添加用户

默认接口请求频率限制：100次/秒。

接口更新时间：2025-05-14 18:04:03。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateUser
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Name	是	否	String	用户名称 示例值： name-test
Password	是	否	String	用户密码 示例值： password-test

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回的结果 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

删除ACL

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

删除ACL

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-09 15:16:53。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteAcl
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例id信息 示例值： ckafka-test
ResourceType	是	否	Int64	Acl资源类型，(0:UNKNOWN，1:ANY，2:TOPIC，3:GROUP，4:CLUSTER，5:TRANSACTIONAL_ID)，当前只有TOPIC，其它字段用于后续兼容开源kafka的acl时使用 示例值： 2
ResourceName	是	否	String	资源名称，和resourceType相关，如当resourceType为TOPIC时，则该字段表示topic名称，当resourceType为GROUP时，该字段表示group名称 示例值： topic-test
Operation	是	否	Int64	Acl操作方式，(0:UNKNOWN，1:ANY，2:ALL，3:READ，4:WRITE，5:CREATE，6:DELETE，7:ALTER，8:DESCRIBE，9:CLUSTER_ACTION，10:DESCRIBE_CONFIGS，11:ALTER_CONFIGS，

参数名称	必选	允许NULL	类型	描述
				12:IDEMPOTEN_WRITE)，当前ckafka只支持READ,WRITE，其它用于后续兼容开源kafka的acl时使用 示例值： 2
PermissionType	是	否	Int64	权限类型，(0:UNKNOWN，1:ANY，2:DENY，3:ALLOW)，当前ckakfa支持ALLOW(相当于白名单)，其它用于后续兼容开源kafka的acl时使用 示例值： 2
Host	否	否	String	默认为*，表示任何host都可以访问，当前ckafka不支持host为*，但是后面开源kafka的产品化会直接支持 示例值： 1.1.1.1
Principal	否	否	String	用户列表，默认为*，表示任何user都可以访问，当前用户只能是用户列表中包含的用户 示例值： User:*

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。

错误码	描述
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

删除用户

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

删除用户

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-09 10:50:41。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteUser
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Name	是	否	String	用户名称 示例值： name-test

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

枚举ACL

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

枚举ACL

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-02 18:29:29。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeACL
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
ResourceType	是	否	Int64	Acl资源类型，(0:UNKNOWN，1:ANY，2:TOPIC，3:GROUP，4:CLUSTER，5:TRANSACTIONAL_ID)，当前只有TOPIC，其它字段用于后续兼容开源kafka的acl时使用 示例值： 2
ResourceName	是	否	String	资源名称，和resourceType相关，如当resourceType为TOPIC时，则该字段表示topic名称，当resourceType为GROUP时，该字段表示group名称 示例值： topic-test
Offset	否	否	Int64	偏移位置 示例值： 0

参数名称	必选	允许NULL	类型	描述
Limit	否	否	Int64	个数限制 示例值： 20
SearchWord	否	否	String	关键字匹配 示例值： search-test

3. 输出参数

参数名称	类型	描述
Result	AclResponse	返回的ACL结果集对象 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制

错误码	描述
InternalError	内部错误。
ResourceUnavailable	资源不可用。

查询用户信息

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

查询用户信息

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-06 16:30:42。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeUser
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
SearchWord	否	否	String	按照名称过滤 示例值： search-test
Offset	否	否	Int64	偏移 示例值： 1
Limit	否	否	Int64	本次返回个数 示例值： 20

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	UserResponse	返回结果列表 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

修改密码

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

修改密码

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-09 10:56:50。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyPassword
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Name	是	否	String	用户名称 示例值： name-test
Password	是	否	String	用户当前密码 示例值： password-test
PasswordNew	是	否	String	用户新密码 示例值： password-new-test

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

主题相关接口
增加主题分区

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

本接口用于增加主题中的分区

默认接口请求频率限制：100次/秒。

接口更新时间：2024-10-15 14:40:26。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreatePartition
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
TopicName	是	否	String	主题名称 示例值： topic-test
PartitionNum	是	否	Int64	主题分区个数 示例值： 3

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	JgwOperateResponse	返回的结果集 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

创建主题

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

创建ckafka主题

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-02 18:33:15。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateTopic
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
TopicName	是	否	String	主题名称，是一个不超过 64 个字符的字符串，必须以字母为首字符，剩余部分可以包含字母、数字和横划线(-) 示例值： topic-test
PartitionNum	是	否	Int64	Partition个数，大于0 示例值： 1
ReplicaNum	是	否	Int64	副本个数，不能多于 broker 数，最大为 3 示例值： 2
EnableWhiteList	否	否	Int64	ip白名单开关, 1:打开 0:关闭，默认不打开

参数名称	必选	允许NULL	类型	描述
				示例值： 0
IpWhiteList	否	否	Array of String	Ip白名单列表，配额限制，enableWhileList=1时必选 示例值： ["1.1.1.1","2.2.2.2"]
CleanUpPolicy	否	否	String	清理日志策略，日志清理模式，默认为"delete"。"delete"：日志按保存时间删除，"compact"：日志按 key 压缩，"compact, delete"：日志按 key 压缩且会按保存时间删除。 示例值： delete
Note	否	否	String	主题备注，是一个不超过 64 个字符的字符串，必须以字母为首字符，剩余部分可以包含字母、数字和横划线(-) 示例值： note
MinInsyncReplicas	否	否	Int64	默认为1 示例值： 2
UncleanLeaderElectionEnable	否	否	Int64	是否允许未同步的副本选为leader，false:不允许，true:允许，默认不允许 示例值： 1
RetentionMs	否	否	Int64	可消息选。保留时间，单位ms，当前最小值为60000ms 示例值： 60000
SegmentMs	否	否	Int64	Segment分片滚动的时长，单位ms，当前最小为3600000ms 示例值： 360000
MaxMessageBytes	否	否	Int64	主题消息最大值，单位为 Byte，最大值为12582912Byte（即12MB）。 示例值： 1024

3. 输出参数

参数名称	类型	描述
Result	CreateTopicResp	返回创建结果 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

创建主题IP白名单

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

创建主题ip白名单

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-02 18:06:20。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateTopicIpWhiteList
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
TopicName	是	否	String	主题名称 示例值： topic-test
IpWhiteList	是	否	Array of String	ip白名单列表 示例值： ["192.16.23.3666"]

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	删除主题IP白名单结果 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

删除主题IP白名单

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

删除主题IP白名单

默认接口请求频率限制：20次/秒。

接口更新时间：2020-03-02 18:07:14。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DeleteTopicIpWhiteList
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例ID 示例值： ckafka-test
TopicName	是	否	String	主题名称 示例值： topic-test
IpWhiteList	是	否	Array of String	ip白名单列表 示例值： ["192.16.23.3666"]

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	删除主题IP白名单结果 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取主题列表

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

接口请求域名：https://ckafka.api3.finance.cloud.tencent.com
本接口（DescribeTopic）用于在用户获取消息队列 CKafka 实例的主题列表

默认接口请求频率限制：100次/秒。

接口更新时间：2020-03-09 16:58:46。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeTopic
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例 ID 示例值： ckafka-test
SearchWord	否	否	String	过滤条件，按照 topicName 过滤，支持模糊查询 示例值： search-test
Offset	否	否	Int64	偏移量，不填默认为0 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认为10，最大值为50 示例值： 50

3. 输出参数

参数名称	类型	描述
Result	TopicResult	返回的结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取主题属性

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

获取主题属性

默认接口请求频率限制：100次/秒。

接口更新时间：2025-09-02 17:31:28。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeTopicAttributes
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例 ID 示例值： ckafka-test
TopicName	是	否	String	主题名称 示例值： topic-test

3. 输出参数

参数名称	类型	描述
Result	TopicAttributesRes	返回的结果对象 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取主题列表详情

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

获取主题列表详情（仅控制台调用）

默认接口请求频率限制：100次/秒。

接口更新时间：2025-08-21 17:11:52。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeTopicDetail
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例id 示例值： ckafka-test
SearchWord	否	否	String	（过滤条件）按照topicName过滤，支持模糊查询 示例值： search-test
Offset	否	否	Int64	偏移量，不填默认为0 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认 10，最大值20，取值要大于0 示例值： 10

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	TopicDetailResponse	返回的主题详情实体 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InternalError	内部错误。
ResourceUnavailable	资源不可用。

查询消息

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

根据指定offset位置的消息

默认接口请求频率限制：100次/秒。

接口更新时间：2022-07-21 11:58:25。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：FetchMessageByOffset
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Topic	是	否	String	主题名 示例值： topic-test
Partition	是	否	Uint64	分区id 示例值： 0
Offset	是	否	Uint64	位点信息 示例值： 0

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	ConsumerRecord	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
InternalError	内部错误。
ResourceUnavailable	资源不可用。

根据位点查询消息列表

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

根据位点查询消息列表

默认接口请求频率限制：100次/秒。

接口更新时间：2022-07-21 11:55:15。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：FetchMessageListByOffset
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Topic	是	否	String	主题名 示例值： topic-test
Partition	是	否	Uint64	分区id 示例值： 0
Offset	是	否	Uint64	位点信息 示例值： 0
SinglePartitionRecordNumber	否	否	Uint64	最大查询条数，默认20，最大20 示例值： 20

3. 输出参数

参数名称	类型	描述
Result	Array of ConsumerRecord	返回结果。注意，列表中不返回具体的消息内容（key、value），如果需要查询具体消息内容，请使用FetchMessageByOffset接口 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
InternalError	内部错误。
ResourceUnavailable	资源不可用。

根据时间戳查询消息列表

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

根据时间戳查询消息列表

默认接口请求频率限制：100次/秒。

接口更新时间：2022-07-21 11:53:04。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：FetchMessageListByTimestamp
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Topic	是	否	String	主题名 示例值： topic-test
Partition	否	否	Uint64	分区id 示例值： 0
StartTime	否	否	Uint64	查询开始时间，13位时间戳 示例值： 1577808000000
SinglePartitionRecordNumber	否	否	Uint64	最大查询条数，默认20，最大20 示例值： 20

3. 输出参数

参数名称	类型	描述
Result	Array of ConsumerRecord	返回结果。注意，列表中不返回具体的消息内容（key、value），如果需要查询具体消息内容，请使用FetchMessageByOffset接口 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
InternalError	内部错误。
ResourceUnavailable	资源不可用。

设置主题属性

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

本接口用于修改主题属性。

默认接口请求频率限制：100次/秒。

接口更新时间：2025-05-20 16:04:30。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyTopicAttributes
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例 ID。 示例值： ckafka-test
TopicName	是	否	String	主题名称。 示例值： topic-test
Note	否	否	String	主题备注，是一个不超过64个字符的字符串，必须以字母为首字符，剩余部分可以包含字母、数字和横划线-。 示例值： note-test
EnableWhiteList	否	否	Int64	IP 白名单开关，1：打开；0：关闭。 示例值： 0
IpWhiteList	否	否	Array of	Ip白名单列表，配额限制，enableWhileList=1时必选

参数名称	必选	允许NULL	类型	描述
			String	示例值： ["10.0.0.5","10.0.0.6"]
MinInsyncReplicas	否	否	Int64	默认为1。 示例值： 1
UncleanLeaderElectionEnable	否	否	Int64	默认为 0，0：false；1：true。 示例值： 0
RetentionMs	否	否	Int64	消息保留时间，单位：ms，当前最小值为60000ms。 示例值： 60000
SegmentMs	否	否	Int64	Segment 分片滚动的时长，单位：ms，当前最小为86400000ms。 示例值： 86400000
MaxMessageBytes	否	否	Int64	主题消息最大值，单位为 Byte，最大值为12582912Byte（即12MB）。 示例值： 1048576
CleanUpPolicy	否	否	String	消息删除策略，可以选择delete 或者 compact 示例值： delete

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果集 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。

错误码	描述
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

其他接口
查询用户列表

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

查询用户列表

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-09 11:00:06。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeAppInfo
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
Offset	否	否	Int64	偏移位置 示例值： 0
Limit	否	否	Int64	本次查询用户数目最大数量限制，最大值为50，默认50 示例值： 10

3. 输出参数

参数名称	类型	描述
Result	AppIdResponse	返回的符合要求的App Id列表 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

查询消费分组信息

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

查询消费分组信息

默认接口请求频率限制：100次/秒。

接口更新时间：2025-09-02 15:27:36。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeConsumerGroup
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	ckafka实例id。 示例值： ckafka-test
GroupName	否	否	String	可选，用户需要查询的group名称。 示例值： group-test
TopicName	否	否	String	可选，用户需要查询的group中的对应的topic名称，如果指定了该参数，而group又未指定则忽略该参数。 示例值： topic-test
Limit	否	否	Int64	本次返回个数限制 示例值： 20
Offset	否	否	Int64	偏移位置 示例值： 0

3. 输出参数

参数名称	类型	描述
Result	ConsumerGroupResponse	返回的消费分组信息 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

枚举消费分组(精简版)

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

枚举消费分组(精简版)

默认接口请求频率限制：100次/秒。

接口更新时间：2025-09-02 15:44:07。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeGroup
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例ID 示例值： ckafka-test
SearchWord	否	否	String	搜索关键字 示例值： search-test
Offset	否	否	Int64	偏移量 示例值： 0
Limit	否	否	Int64	最大返回数量 示例值： 20

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	GroupRes	返回结果集列表 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

查看订单配置

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

DescribeOrderConfig

默认接口请求频率限制：100次/秒。

接口更新时间：2024-10-14 16:21:04。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeOrderConfig
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceType	是	否	String	实例规格，不同规格类型代表实例不同的配置组合（吞吐量、磁盘）。具体取值可调用接口GetCkafkaTypeConfigs获取最新规格表。 示例值： profession
Period	否	否	Int64	实例时长，单位：月，最小值1，最大值为36 示例值： 1
InstanceNum	否	否	Int64	购买实例数量。取值范围：[1，20] 示例值： 1
InquireFlag	否	否	Int64	询价标记：不填或者 0 默认为新够买询价 1 为续费 示例值： 0

参数名称	必选	允许NULL	类型	描述
ZoneId	否	否	Int64	区域ID 示例值： 100001
Bandwidth	是	否	Int64	购买带宽规格，单位：MB 示例值： 40
DiskSize	否	否	Int64	购买磁盘规格，默认为带宽对应的最小值，单位：GB 示例值： 600
DiskType	否	否	String	购买磁盘类型，默认为CLOUD_TCS 示例值： SSD
MessageRetention	否	否	Int64	消息保留时间，默认为72，单位：小时 示例值： 72
Topic	否	否	Int64	购买Topic规格，默认为带宽对应的最小值，单位：个 示例值： 100
Partition	否	否	Int64	购买Partition规格，默认为带宽对应的最小值，单位：个 示例值： 1000
BrokerVersion	否	否	String	购买实例版本 示例值： 1.1.1
CategoryAction	否	否	String	标记：purchase新购，renew续费，modify升配 示例值： purchase
BillType	否	否	String	标准版S2计费类型 示例值： 1
PublicNetworkChargeType	否	否	String	公网带宽计费类型（预付费按带宽结算(包年带宽):BANDWIDTH_PREPAID、带宽按小时后付费(按小时带宽):BANDWIDTH_POSTPAID_BY_HOUR 示例值： BANDWIDTH_PREPAID
PublicNetworkMonthly	否	否	Int64	公网带宽 示例值： 1
ResourceId	否	否	String	实例ID 示例值： 无

3. 输出参数

参数名称	类型	描述
Result	BillingOrderConfig	出参 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

该接口暂无业务逻辑相关的错误码，其他错误码详见公共错误码。

实例相关接口
创建token

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

创建最高权限的token

默认接口请求频率限制：100次/秒。

接口更新时间：2024-01-17 17:55:58。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：CreateToken
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
User	是	否	String	user 示例值： username-test

3. 输出参数

参数名称	类型	描述
Result	String	token结果 示例值： 904b75130b4b4102bdeb071a7f48523a

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
UnauthorizedOperation	未授权操作。
InternalError	内部错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
ResourceUnavailable	资源不可用。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameter	参数错误。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.NotAllowedEmpty	参数不允许为空。
InvalidParameterValue.ZoneNotSupport	zone不支持
FailedOperation.TradeFailed	调用计量计费接口失败
InvalidParameterValue.InstanceNotExist	实例不存在。
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制

获取消费分组信息

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

获取消费分组信息

默认接口请求频率限制：100次/秒。

接口更新时间：2020-01-07 11:26:49。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeGroupInfo
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	（过滤条件）按照实例 ID 过滤。 示例值： ckafka-test
GroupList	是	否	Array of String	Kafka 消费分组，Consumer-group，这里是数组形式。如果使用拼接数组元素方式传值（一般 GET 请求用），可以使用格式：GroupList.0=xxx&GroupList.1=yyy。 示例值： ["group-test"]

3. 输出参数

参数名称	类型	描述
Result	Array of GroupInfoResponse	返回的结果 示例值：查看

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取消费分组offset

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

获取消费分组offset

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-06 16:51:35。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeGroupOffsets
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	（过滤条件）按照实例 ID 过滤 示例值： ckafka-test
Group	是	否	String	Kafka 消费分组 示例值： group-test
Topics	否	否	Array of String	group 订阅的主题名称数组，如果没有该数组，则表示指定的 group 下所有 topic 信息 示例值： topic-test
SearchWord	否	否	String	模糊匹配 topicName 示例值： search-test
Offset	否	否	Int64	本次查询的偏移位置，默认为0 示例值： 0
Limit	否	否	Int64	本次返回结果的最大个数，默认为50，最大值为50 示例值： 20

3. 输出参数

参数名称	类型	描述
Result	GroupOffsetResponse	返回的结果对象 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取实例属性

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

获取实例属性

默认接口请求频率限制：100次/秒。

接口更新时间：2025-09-02 15:59:41。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeInstanceAttributes
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例id 示例值： ckafka-test

3. 输出参数

参数名称	类型	描述
Result	InstanceAttributesRes	实例属性返回结果对象 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取实例列表

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

本接口（DescribeInstance）用于在用户账户下获取消息队列 CKafka 实例列表

默认接口请求频率限制：20次/秒。

接口更新时间：2019-12-19 11:23:20。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeInstances
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	否	否	String	（过滤条件）按照实例ID过滤 示例值： ckafka-test
SearchWord	否	否	String	（过滤条件）按照实例名称过滤，支持模糊查询 示例值： search-test
Status	否	否	Array of Int64	（过滤条件）实例的状态。0：创建中，1：运行中，2：删除中，不填默认返回全部 示例值： [1,7]
Offset	否	否	Int64	偏移量，不填默认为0 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认10，最大值20 示例值： 10
TagKey	否	否	String	匹配标签key值。 示例值： tagkey-test

3. 输出参数

参数名称	类型	描述
Result	InstanceResponse	返回的结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

获取实例列表详情

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

用户账户下获取实例列表详情

默认接口请求频率限制：20次/秒。

接口更新时间：2025-08-21 17:20:19。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeInstancesDetail
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	否	否	String	（过滤条件）按照实例ID过滤 示例值： ckafka-test
SearchWord	否	否	String	（过滤条件）按照实例名称过滤，支持模糊查询 示例值： search-test
Status	否	否	Array of Int64	（过滤条件）实例的状态。0：创建中，1：运行中，2：删除中，不填默认返回全部 示例值： [1,5]
Offset	否	否	Int64	偏移量，不填默认为0 示例值： 0
Limit	否	否	Int64	返回数量，不填则默认10，最大值20 示例值： 10
TagKey	否	否	String	匹配标签key值。 示例值： 1

参数名称	必选	允许NULL	类型	描述
Filters	否	否	Array of Filter	过滤器 示例值：查看

3. 输出参数

参数名称	类型	描述
Result	InstanceDetailResponse	返回的实例详情结果对象 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。

错误码	描述
ResourceUnavailable	资源不可用。

设置Groups 消费分组offset

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

设置Groups 消费分组offset

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-09 10:54:37。

接口既验签名又鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyGroupOffsets
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	kafka实例id 示例值： ckafka-test
Group	是	否	String	kafka 消费分组 示例值： group-test
Strategy	是	否	Int64	重置offset的策略，入参含义 0. 对齐shift-by参数，代表把offset向前或向后移动shift条 1. 对齐参考(by-duration,to-datetime,to-earliest,to-latest),代表把offset移动到指定timestamp的位置 2. 对齐参考(to-offset)，代表把offset移动到指定的offset位置 示例值： 1
Topics	否	否	Array of String	表示需要重置的topics， 不填表示全部 示例值： ["topic-test"]

参数名称	必选	允许NULL	类型	描述
Shift	否	否	Int64	当strategy为0时，必须包含该字段，可以大于零代表会把offset向后移动shift条，小于零则将offset向前回溯shift条数。正确重置后新的offset应该是(old_offset + shift)，需要注意的是如果新的offset小于partition的earliest则会设置为earliest，如果大于partition 的latest则会设置为latest 示例值： 10
ShiftTimestamp	否	否	Int64	单位ms。当strategy为1时，必须包含该字段，其中-2表示重置offset到最开始的位置，-1表示重置到最新的位置(相当于清空)，其它值则代表指定的时间，会获取topic中指定时间的offset然后进行重置，需要注意的时，如果指定的时间不存在消息，则获取最末尾的offset。 示例值： 13589015901000
Offset	否	否	Int64	需要重新设置的offset位置。当strategy为2，必须包含该字段。 示例值： 100
Partitions	否	否	Array of Uint64	需要重新设置的partition的列表，如果没有指定Topics参数。则重置全部topics的对应的Partition列表里的partition。指定Topics时则重置指定的topic列表的对应的Partitions列表的partition。 示例值： [0]

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

设置实例属性

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

设置实例属性

默认接口请求频率限制：100次/秒。

接口更新时间：2019-12-09 10:52:33。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyInstanceAttributes
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例id 示例值： ckafka-test
MsgRetentionTime	否	否	Int64	实例日志的最长保留时间，单位分钟，最大30天，0代表不开启日志保留时间回收策略 示例值： 10
InstanceName	否	否	String	实例名称，是一个不超过 64 个字符的字符串，必须以字母为首字符，剩余部分可以包含字母、数字和横划线(-) 示例值： ckafka测试

参数名称	必选	允许NULL	类型	描述
Config	否	否	ModifyInstanceAttributesConfig	实例配置 示例值：查看

3. 输出参数

参数名称	类型	描述
Result	JgwOperateResponse	返回结果 示例值：查看
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
InvalidParameter	参数错误。
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.WrongAction	Action参数取值错误。
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InternalError	内部错误。
ResourceUnavailable	资源不可用。

访问控制相关接口
授权token

1. 接口描述

接口请求域名： ckafka.api3.finance.cloud.tencent.com。

AuthorizeToken

默认接口请求频率限制：100次/秒。

接口更新时间：2025-05-14 17:48:26。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：AuthorizeToken
Version	是	否	String	公共参数，本接口取值：2019-08-19
Region	是	否	String	公共参数，地域信息可通过DescribeRegions接口查看产品支持的地域列表
InstanceId	是	否	String	实例Id 示例值： ckafka-test
Tokens	是	否	String	token 示例值： 827636a7-3643-4afa-a23c-2eb7352e266c
User	是	否	String	用户 示例值： username-test

3. 输出参数

参数名称	类型	描述

参数名称	类型	描述
Result	Int64	0 成功 示例值： 0
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

该接口暂无业务逻辑相关的错误码，其他错误码详见公共错误码。

数据结构

BillingOrderConfig

计费订单

被如下接口引用：DescribeOrderConfig

名称	必选	允许NULL	类型	描述
Inquiry	否	否	String	订单价格信息 示例值： 1
OrderConfig	否	否	String	订单 示例值： 1
CategoryId	否	否	Int64	类别Id 示例值： 1

PartDetail

价格对象详情

被如下接口引用：DescribeCkafkaPrice

名称	必选	允许NULL	类型	描述
Disk	是	是	PriceObject	硬盘价格 示例值：查看
CloudKafka	是	是	PriceObject	ckafka价格 示例值：查看

ClusterInfo

集群信息实体

被如下接口引用：DescribeCkafkaZone

名称	必选	允许NULL	类型	描述
ClusterId	是	否	Int64	集群Id 示例值： 6894
ClusterName	是	否	String	集群名称 示例值： 100003-ckafka-test

名称	必选	允许NULL	类型	描述
ZoneId	否	否	Int64	集群可用区 示例值： 100003
ZoneIds	否	否	Array of Int64	Broker 的可用区列表 示例值： [100003,100002]
CpuArch	否	否	Int64	集群的 CPU 架构。1:x86; 2:arm; 按位标记架构类型，例：3 二进制为 0011，表示 x86、arm 混合架构 示例值： 1
MaxDiskSize	是	是	Int64	集群最大磁盘 单位GB 示例值： 198
MaxBandWidth	是	是	Int64	集群最大带宽 单位MB/s 示例值： 19
AvailableDiskSize	是	是	Int64	集群当前可用磁盘 单位GB 示例值： 198
AvailableBandWidth	是	是	Int64	集群当前可用带宽 单位MB/s 示例值： 19

CommunityResponse

是否社区版的返回对象

被如下接口引用：DescribeIfCommunity

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例ID 示例值： ckafka-test
IfCommunity	是	否	Bool	是否社区版 示例值： false

ConsumerGroup

用户组实体

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
ConsumerGroupName	是	否	String	用户组名称 示例值： consumerGroup-test
SubscribedInfo	是	否	Array of SubscribedInfo	订阅信息实体 示例值：查看

VipEntity

虚拟IP实体

被如下接口引用：DescribeInstanceAttributes、DescribeInstanceDetail、DescribeInstancesDetail、DescribeRoute

名称	必选	允许NULL	类型	描述
Vip	是	否	String	虚拟IP 示例值： 1.0.0.5
Vport	是	否	String	虚拟端口 示例值： 9092

ZoneInfo

zone信息实体

被如下接口引用：DescribeCkafkaZone

名称	必选	允许NULL	类型	描述
ZoneId	是	否	String	zone的id 示例值： 100003
IsInternalApp	是	否	Int64	是否内部APP 示例值： 1
AppId	是	否	Int64	app id 示例值： 1000000001
Flag	是	否	Bool	标识 示例值： false
ZoneName	是	否	String	zone名称 示例值： region3
ZoneStatus	是	否	Int64	zone状态 示例值： 3
Exflag	是	否	String	额外标识 示例值： 1
SoldOut	是	否	String	json对象，key为机型，value true为售罄，false为未售罄 示例值： true
ExtraFlag	否	是	String	额外Flag 示例值： "0"
SalesInfo	否	是	Array of SalesInfo	售卖信息 示例值：查看

ConsumerGroupResponse

消费组返回结果实体

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合条件的消费组数量 示例值： 1
TopicList	是	是	Array of ConsumerGroupTopic	主题列表 示例值：查看
GroupList	是	是	Array of ConsumerGroup	消费分组List 示例值：查看
TotalPartition	是	是	Int64	所有分区数量 示例值： 0
PartitionListForMonitor	是	是	Array of Partition	监控的分区列表 示例值：查看
TotalTopic	是	是	Int64	主题总数 示例值： 1
TopicListForMonitor	是	是	Array of ConsumerGroupTopic	监控的主题列表 示例值：查看
GroupListForMonitor	是	是	Array of Group	监控的组列表 示例值：查看

TradeCreateResponse

创建或修改计量计费类型实例的响应内容

被如下接口引用：HourCreate、HourModify

名称	必选	允许NULL	类型	描述
DealNames	否	否	Array of String	后付费订单列表 示例值： test
FlowId	否	否	String	发货任务ID号（为兼容单个资源开通情况保留） 示例值： 1
FlowIds	是	否	Array of String	发货任务ID号列表 示例值： []
ResourceIds	否	否	Array of String	资源列表 示例值： []

名称	必选	允许NULL	类型	描述
BillId	否	否	String	交易流水号（一般为大订单号） 示例值： 1

AppIdResponse

AppId的查询结果

被如下接口引用：DescribeAppInfo

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合要求的所有AppId数量 示例值： 1
AppIdList	是	是	Array of Int64	符合要求的App Id列表 示例值： [1000000001]

ClientQuotaConfigResp

topic客户端配额设置

被如下接口引用：DescribeTopicAttributes

名称	必选	允许NULL	类型	描述
QuotaProducerByteRate	否	否	Int64	生产限流大小 示例值： 100
QuotaConsumerByteRate	否	否	Int64	消费限流大小 示例值： 100

ModifyInstanceAttributesConfig

修改实例属性的配置对象

被如下接口引用：ModifyInstanceAttributes

名称	必选	允许NULL	类型	描述
AutoCreateTopicEnable	否	否	Bool	自动创建 true 表示开启，false 表示不开启 示例值： true
DefaultNumPartitions	否	否	Int64	可选，如果auto.create.topic.enable设置为true没有设置该值时，默认设置为3 示例值： 3

名称	必选	允许NULL	类型	描述
DefaultReplicationFactor	否	否	Int64	如歌auto.create.topic.enable设置为true没有指定该值时默认设置为2 示例值： 2

Route

路由实体对象

被如下接口引用：DescribeRoute

名称	必选	允许NULL	类型	描述
AccessType	是	否	Int64	实例接入方式 0：PLAINTEXT (明文方式，没有带用户信息老版本及社区版本都支持) 1：SASL_PLAINTEXT（明文方式，不过在数据开始时，会通过SASL方式登录鉴权，仅社区版本支持） 2：SSL（SSL加密通信，没有带用户信息，老版本及社区版本都支持） 3：SASL_SSL（SSL加密通信，在数据开始时，会通过SASL方式登录鉴权，仅社区版本支持） 示例值： 1
RouteId	是	否	Int64	路由ID 示例值： 301
VipType	是	否	Int64	vip网络类型（1:外网TGW 2:基础网络 3:VPC网络 4:云平台支持环境(一般用于内部实例) 5:SSL外网访问方式访问 6:黑石环境vpc） 示例值： 1
VipList	是	否	Array of VipEntity	虚拟IP列表 示例值：查看
Domain	是	是	String	域名 示例值： cloud.com
DomainPort	是	是	Int64	域名port 示例值： 0
InternalFlag	否	否	Int64	内部路由的标志(0:非内部 1:内部路由),内部路由不在DescribeRoute接口展示,用户控制台无法查看删除此路由 示例值： 0
UsedFor	否	否	String	内部路由的用途(非内部路由为空串 内部目前有:internalDefault#默认,crossRegion#跨地域容灾) 示例值： crossRegion
Processing	否	否	Int64	路由创建进度 示例值： 1

名称	必选	允许NULL	类型	描述
DeleteTimestamp	否	否	String	删除时间戳 示例值： 1300313212123
BrokerVipList	否	否	Array of VipEntity	broker vip映射列表 示例值：查看
VpcId	否	否	String	vpcid 示例值： vpc-test
Subnet	否	否	String	子网信息 示例值： subnet-test
Note	否	否	String	备注 示例值： 数据传输路由

ConnectorConfigsResponse

Connector配置信息对象

被如下接口引用：DescribeConnectorConfigs

名称	必选	允许NULL	类型	描述
ConnectorClass	是	否	String	执行该任务的 class 名称。 示例值： com.cloud.ckafka.replicator.K2KSourceConnector
DestInstance	是	否	String	目的实例名称。 示例值： ckafka-test
TopicConfig	是	否	String	主题配置。 示例值： {"test":{},"test1":{}}
ConnectorVersion	是	否	String	connector 版本。 示例值： 0.10.2.1
SrcInstance	是	否	String	源实例。 示例值： ckafka-test
OffsetReset	是	否	String	任务同步启动时候的 offset 设置 示例值： latest
SourceBroker	是	否	String	源实例访问 broker 地址和端口。 示例值： 10.14.211.55:6017
KeepPartition	是	否	Bool	源和目的 topic 的 partition 数量是否要求保持一致。 true：保持一致；false：不一致。 示例值： true
Name	是	否	String	connector 名称。 示例值： connector-ckafka-test

名称	必选	允许NULL	类型	描述
AutoCreate	是	否	Bool	是否开启主题自动创建。true 表示开启，false 表示不开启。 示例值： false

GroupInfoMember

consumer信息

被如下接口引用：DescribeGroupInfo

名称	必选	允许NULL	类型	描述
MemberId	是	否	String	coordinator 为消费分组中的消费者生成的唯一 ID 示例值： consumer-test
ClientId	是	否	String	客户消费者 SDK 自己设置的 client.id 信息 示例值： consumer-client-test
ClientHost	是	否	String	一般存储客户的 IP 地址 示例值： 10.0.0.30
Assignment	是	否	Assignment	存储着分配给该消费者的 partition 信息 示例值：查看

TopicDetailResponse

主题详情返回实体

被如下接口引用：DescribeTopicDetail

名称	必选	允许NULL	类型	描述
TopicList	是	是	Array of TopicDetail	返回的主题详情列表 示例值：查看
TotalCount	是	否	Int64	符合条件的所有主题详情数量 示例值： 1

Acl

ACL对象实体

被如下接口引用：DescribeACL、DescribeTopicAttributes

名称	必选	允许NULL	类型	描述
ResourceType	是	否	Int64	Acl资源类型，（0:UNKNOWN，1:ANY，2:TOPIC，3:GROUP，4:CLUSTER，5:TRANSACTIONAL_ID）当前只有TOPIC，

名称	必选	允许NULL	类型	描述
				示例值： 2
ResourceName	是	否	String	资源名称，和resourceType相关如当resourceType为TOPIC时，则该字段表示topic名称，当resourceType为GROUP时，该字段表示group名称 示例值： topic-test
Principal	是	是	String	用户列表，默认为User:，*表示任何user都可以访问，当前用户只能是用户列表中包含的用户* *示例值： User:*
Host	是	是	String	默认为*，表示任何host都可以访问，当前ckafka不支持host为*，但是后面开源kafka的产品化会直接支持 示例值： 10.0.0.1
Operation	是	否	Int64	Acl操作方式(0:UNKNOWN，1:ANY，2:ALL，3:READ，4:WRITE，5:CREATE，6:DELETE，7:ALTER，8:DESCRIBE，9:CLUSTER_ACTION，10:DESCRIBE_CONFIGS，11:ALTER_CONFIGS，12:IDEMPOTEN_WRITE) 示例值： 3
PermissionType	是	否	Int64	权限类型(0:UNKNOWN，1:ANY，2:DENY，3:ALLOW) 示例值： 3

Config

高级配置对象

被如下接口引用：DescribeTopicAttributes、DescribeTopicDetail

名称	必选	允许NULL	类型	描述
Retention	是	是	Int64	消息保留时间 示例值： 1
MinInsyncReplicas	是	是	Int64	最小同步复制数 示例值： 1
CleanUpPolicy	是	是	String	日志清理模式，默认 delete。 delete：日志按保存时间删除；compact：日志按 key 压缩；compact, delete：日志按 key 压缩且会保存时间删除。 示例值： delete
SegmentMs	是	是	Int64	Segment 分片滚动的时长 示例值： 1
UncleanLeaderElectionEnable	是	是	Int64	0表示 false。 1表示 true。 示例值： 0

名称	必选	允许NULL	类型	描述
SegmentBytes	是	是	Int64	Segment 分片滚动的字节数 示例值： 1
MaxMessageBytes	是	是	Int64	最大消息字节数 示例值： 1
RetentionBytes	否	是	Int64	消息保留 示例值： 100

ConsumerRecord

消息记录

被如下接口引用：FetchMessageByOffset、FetchMessageListByOffset、FetchMessageListByTimestamp

名称	必选	允许NULL	类型	描述
Topic	是	否	String	主题名 示例值： topic-test
Partition	是	否	Uint64	分区id 示例值： 0
Offset	是	否	Uint64	位点 示例值： 0
Key	否	是	String	消息key 示例值： key-test
Value	否	是	String	消息value 示例值： value-test
Timestamp	否	是	Uint64	消息时间戳 示例值： 1657542315
Headers	否	是	String	消息headers 示例值： header-test

PartitionOffset

分区和位移

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
Partition	是	是	String	Partition,例如"0"或"1" 示例值： 0

名称	必选	允许NULL	类型	描述
Offset	是	是	Int64	Offset,例如100 示例值： 20

TaskStatusResponse

任务状态返回对象

被如下接口引用：DescribeTaskStatus

名称	必选	允许NULL	类型	描述
Status	是	否	Int64	任务状态: 0 成功 1 失败 2 进行中 示例值： 1
Output	是	是	String	输出信息 示例值： 任务异常失败

Tag

实例详情中的标签对象

被如下接口引用：DescribeInstanceAttributes、DescribeInstanceDetail、DescribeInstancesDetail

名称	必选	允许NULL	类型	描述
TagKey	是	否	String	标签的key 示例值： tagkey-test
TagValue	是	否	String	标签的值 示例值： tagvalue-test

K8sclusterDetail

K8s 集群详情。

被如下接口引用：DescribeK8sClusters

名称	必选	允许NULL	类型	描述
K8sClusterId	否	否	String	专业版实例所在的K8s集群Id 示例值： 无

名称	必选	允许NULL	类型	描述
K8sClusterName	否	否	String	专业版实例所在的K8s集群名 示例值： 无

InstanceResponse

聚合的实例状态返回结果

被如下接口引用：DescribeInstances

名称	必选	允许NULL	类型	描述
InstanceList	是	是	Array of Instance	符合条件的实例列表 示例值：查看
TotalCount	是	是	Int64	符合条件的结果总数 示例值： 1

GroupRes

DescribeGroup的返回

被如下接口引用：DescribeGroup

名称	必选	允许NULL	类型	描述
TotalCount	是	是	Int64	计数 示例值： 1
GroupList	是	是	Array of DescribeGroup	GroupList 示例值：查看
GroupCountQuota	否	否	Int64	200 示例值： 消费组配额

InstanceDetail

实例详情

被如下接口引用：DescribeInstanceDetail、DescribeInstancesDetail

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例id 示例值： ckafka-test
InstanceName	是	否	String	实例名称 示例值： ckafka测试

名称	必选	允许NULL	类型	描述
Vip	是	否	String	访问实例的vip 信息 示例值： 10.0.0.5
Vport	是	否	String	访问实例的端口信息 示例值： 9092
VipList	是	否	Array of VipEntity	虚拟IP列表 示例值：查看
Status	是	否	Int64	实例的状态。0：创建中，1：运行中，2：删除中：5隔离中， -1 创建失败 示例值： 1
Bandwidth	是	否	Int64	实例带宽，单位Mbps 示例值： 300
DiskSize	是	否	Int64	实例的存储大小，单位GB 示例值： 800
ZoneId	是	否	Int64	可用区域ID 示例值： 100003
VpcId	是	否	String	vpcId，如果为空，说明是基础网络 示例值： vpc-test
SubnetId	是	否	String	子网id 示例值： subnet-test
RenewFlag	是	否	Int64	实例是否续费，int 枚举值：1表示自动续费，2表示明确不自动续费 示例值： 1
Healthy	是	否	Int64	实例状态 int：0表示健康，1表示告警，2 表示实例状态异常 示例值： 1
HealthyMessage	是	否	String	实例状态信息 示例值： "健康"
CreateTime	是	否	Int64	实例创建时间时间 示例值： 1566144000
ExpireTime	是	否	Int64	实例过期时间 示例值： 1566144000
IsInternal	是	否	Int64	是否为内部客户。值为1 表示内部客户 示例值： 1
TopicNum	是	否	Int64	Topic个数 示例值： 10
Tags	是	否	Array of Tag	标识tag 示例值：查看

名称	必选	允许NULL	类型	描述
Version	是	是	String	kafka版本信息 示例值： 1.1.1
ZoneIds	是	是	Array of Int64	跨可用区 示例值： [100003,100004]
Cvm	是	是	Int64	ckafka售卖类型 示例值： 0
CpuArch	否	否	Int64	集群的 CPU 架构。1:x86; 2:arm; 按位标记架构类型，例：3 二进制为 0011，表示 x86、arm 混合架构 示例值： 1
ClusterId	否	是	Int64	实例所在集群的Id 示例值： 1
ClusterName	否	是	String	实例所在集群的名称 示例值： test
PayMode	否	否	Int64	付费模式。-1:非付费实例(默认)；0:后付费实例；1:预付费实例 示例值： 1
DiskType	否	否	String	磁盘类型 示例值： SSD
K8sClusterId	否	否	String	专业版实例所在的K8s集群Id 示例值： global
K8sClusterName	否	否	String	专业版实例所在的K8s集群名 示例值： global
MaxPartitionNumber	否	否	Int64	最大partition数量 示例值： 60
SpecificationsType	否	是	String	规格类型 示例值： profession
ProjectId	否	是	String	项目id 示例值： 1
AllowDowngrade	否	否	Bool	是否允许降配 示例值： false
MaxTopicNumber	否	否	Int64	最大topic数量 示例值： 25
ProjectName	否	是	String	项目名 示例值： test
AppId	否	否	Int64	展示appId字段 示例值： 1255000011

名称	必选	允许NULL	类型	描述
MaxAvailableDiskSize	否	否	Int64	最大可用磁盘 示例值： 87
InstanceType	否	否	String	实例类型 示例值： profession
PartitionNumber	否	否	Int64	当前partition数量 示例值： 0
Features	否	是	Array of String	功能 示例值： []

DynamicRetentionTime

动态消息保留时间

被如下接口引用：DescribeInstanceAttributes

名称	必选	允许NULL	类型	描述
Enable	否	否	Int64	/开启或者关闭(0: 关闭，1: 开启) 示例值： 0
DiskQuotaPercentage	否	否	Int64	磁盘配额百分比(即触发条件) 示例值： 75
StepForwardPercentage	否	否	Int64	往前调整步长百分比 示例值： 10
BottomRetention	否	否	Int64	保底时长，单位分钟 示例值： 20

AclResponse

ACL返回结果集

被如下接口引用：DescribeACL

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合条件的总数据条数 示例值： 1
AclList	是	是	Array of Acl	ACL列表 示例值：查看

OperateResponseData

操作类型返回的Data结构

被如下接口引用：CreateAcl、CreateConnector、CreatePartition、CreateRoute、CreateTopicIpWhiteList、CreateUser、DeleteAcl、DeleteConnector、DeleteRoute、DeleteTopicIpWhiteList、DeleteUser、ModifyGroupOffsets、ModifyInstanceAttributes、ModifyPassword、ModifyTopicAttributes、PauseConnector、ResumeConnector

名称	必选	允许NULL	类型	描述
FlowId	是	是	Int64	FlowId 示例值： 500201

PriceResponse

价格返回结果

被如下接口引用：DescribeCkafkaPrice

名称	必选	允许NULL	类型	描述
Price	是	否	Int64	价格 示例值： 2869900
PartDetail	是	否	PartDetail	组成详情 示例值：查看
HasUserPrice	是	否	Bool	是否用户价格 示例值： false
TotalCost	是	否	Int64	总花费 示例值： 2869900
Currency	是	否	String	货币 示例值： C
TimeUnit	是	否	String	商品的时间单位 示例值： m
TimeSpan	是	否	Int64	商品的时间大小 示例值： 1
GoodsNum	是	否	Int64	资源实例个数 示例值： 1
FromType	是	否	Int64	类型 示例值： 0
Pid	是	否	Int64	pid 示例值： 15128
RealTotalCost	是	否	Int64	真实总花费 示例值： 2869900

名称	必选	允许NULL	类型	描述
ProductCode	是	否	String	产品代码 示例值： p_ckafka
SubProductCode	是	否	String	子产品代码 示例值： sp_ckafka_standard
Policy	是	否	Float	优惠 示例值： 100
PolicyDetail	是	否	PolicyDetail	优惠详情 示例值：查看
UnitPrice	是	否	Int64	单价 示例值： 2869900

GoodsInfo

后付费商品入参信息

被如下接口引用：HourCreate、HourModify

名称	必选	允许NULL	类型	描述
GoodsCategoryId	否	否	Int64	订单类型ID，接入计费时由计费后台分配 示例值： 1
RegionId	是	否	Int64	区域ID 示例值： 1
ZoneId	否	否	Int64	可用区ID，没有可用区概念则不传 示例值： 12121312
GoodsNum	是	否	Int64	新购时表示商品数量，续费与配置变更时固定传1 示例值： 1
ProjectId	否	否	Int64	项目ID，适用于公有云 示例值： 1
PlatformProjectId	否	否	String	项目ID，适用于专有云、私有云等非公有云平台 示例值： 1
PayMode	是	否	Int64	付费模式，0后付费/1预付费 示例值： 1
Platform	否	否	Int64	平台，0开放平台/1云平台 没有则不传 示例值： 1
GoodsDetail	是	否	String	商品详情，timeSpan表示购买时间长度，timeUnit表示购买时间单位，pid表示定价公式ID 示例值： []

名称	必选	允许NULL	类型	描述
Type	否	否	String	类型 示例值： 1

Tags

tag配置

被如下接口引用：DescribeTopicDetail

名称	必选	允许NULL	类型	描述
TagKey	否	否	String	tag配置key 示例值： test
TagValue	否	否	String	tag配置value 示例值： test

Filter

查询过滤器

> 描述键值对过滤器，用于条件过滤查询。例如过滤ID、名称、状态等
>
> - 若存在多个 Filter 时， Filter 间的关系为逻辑与（ AND ）关系。
> - 若同一个 Filter 存在多个 Values ，同一 Filter 下 Values 间的关系为逻辑或（ OR ）关系。

被如下接口引用：DescribeInstanceDetail、DescribeInstancesDetail

名称	必选	允许NULL	类型	描述
Name	是	否	String	需要过滤的字段。 示例值： GroupState
Values	是	否	Array of String	字段的过滤值。 示例值： ["Empty"]

InstanceDetailResponse

实例详情返回结果

被如下接口引用：DescribeInstanceDetail、DescribeInstancesDetail

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合条件的实例总数 示例值： 1

名称	必选	允许NULL	类型	描述
InstanceList	是	否	Array of InstanceDetail	符合条件的实例详情列表 示例值：查看

RetentionTimeConfig

topic消息保留时长配置信息

被如下接口引用：DescribeTopicDetail

名称	必选	允许NULL	类型	描述
Expect	否	否	Int64	用户配置值即期望值 示例值： 1440
Current	否	否	Int64	当前值 示例值： 1440
ModTimeStamp	否	否	Int64	变更修改时间 示例值： 1755244634000

RouteRes

路由信息返回对象

被如下接口引用：DescribeRoute

名称	必选	允许NULL	类型	描述
Routers	是	是	Array of Route	路由信息列表 示例值：查看

K8sclusterDetailResponse

K8s 集群返回值结构体。

被如下接口引用：DescribeK8sClusters

名称	必选	允许NULL	类型	描述
K8sClusters	否	否	K8sclusterDetail	K8s 集群列表。 示例值：查看
TotalCount	否	否	Int64	K8s 集群总数。 示例值： 无

DynamicDiskConfig

动态磁盘配置

被如下接口引用：DescribeInstanceAttributes

名称	必选	允许NULL	类型	描述
Enable	否	否	Int64	是否启用功能，1 启用，0 未启用，缺省 未启用 示例值： 0
DiskQuotaPercentage	否	否	Int64	磁盘负载达到百分比，启用动态扩容 示例值： 60
StepForwardPercentage	否	否	Int64	磁盘动态扩容步进大小 示例值： 10
MaxDiskSpace	否	否	Int64	最大扩容硬盘大小 示例值： 1000

GroupOffsetResponse

消费组偏移量返回结果

被如下接口引用：DescribeGroupOffsets

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合调节的总结果数 示例值： 1
TopicList	是	是	Array of GroupOffsetTopic	该主题分区数组，其中每个元素为一个 json object 示例值：查看

InstanceTypeConfigDO

实例规格配置对象

被如下接口引用：DescribeCkafkaTypeConfigs

名称	必选	允许NULL	类型	描述
Type	是	否	String	型号 示例值： general
Desc	是	否	String	类型描述 示例值： 入门型
Bandwidth	是	否	Int64	带宽流量大小，单位Mbqs 示例值： 320

名称	必选	允许NULL	类型	描述
DiskSize	是	否	Int64	磁盘大小，单位GB 示例值： 300
Pid	是	否	Int64	类型对应的pid信息 示例值： 15127
MaximumInstancePartition	是	否	Int64	该规格可以创建的分区数量配额 示例值： 60
MaximumInstanceTopic	是	否	Int64	该规格可以创建的主题数量配额 示例值： 25

Topic

返回的topic对象

被如下接口引用：DescribeTopic

名称	必选	允许NULL	类型	描述
TopicId	是	否	String	主题的ID 示例值： topic-test
TopicName	是	否	String	主题的名称 示例值： topic-test
Note	是	是	String	备注 示例值： note-test

GroupOffsetPartition

组偏移量分区对象

被如下接口引用：DescribeGroupOffsets

名称	必选	允许NULL	类型	描述
Partition	是	否	Int64	topic 的 partitionId 示例值： 0
Offset	是	否	Int64	consumer 提交的 offset 位置 示例值： 0
Metadata	是	是	String	支持消费者提交消息时，传入 metadata 作为它用，当前一般为空字符串 示例值： Metadata-test
ErrorCode	是	否	Int64	错误码 示例值： 502

名称	必选	允许NULL	类型	描述
LogEndOffset	是	否	Int64	当前 partition 最新的 offset 示例值： 118
Lag	是	否	Int64	未消费的消息个数 示例值： 1

JgwOperateResponse

操作型结果返回值

被如下接口引用：CreateAcl、CreateConnector、CreatePartition、CreateRoute、CreateTopicIpWhiteList、CreateUser、DeleteAcl、DeleteConnector、DeleteRoute、DeleteTopicIpWhiteList、DeleteUser、ModifyGroupOffsets、ModifyInstanceAttributes、ModifyPassword、ModifyTopicAttributes、PauseConnector、ResumeConnector

名称	必选	允许NULL	类型	描述
ReturnCode	是	否	String	返回的code，0为正常，非0为错误 示例值： 0
ReturnMessage	是	否	String	成功消息 示例值： success
Data	是	是	OperateResponseData	操作型返回的Data数据,可能有flowId等 示例值：查看

GroupInfoTopics

GroupInfo内部topic对象

被如下接口引用：DescribeGroupInfo

名称	必选	允许NULL	类型	描述
Topic	是	否	String	分配的 topic 名称 示例值： topic-test
Partitions	是	是	Array of Int64	分配的 partition 信息 示例值： [0]

Instance

实例对象

被如下接口引用：DescribeInstances

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例id 示例值： ckafka-test
InstanceName	是	否	String	实例名称 示例值： ckafka测试
Status	是	否	Int64	实例的状态。0：创建中，1：运行中，2：删除中 ， 5 隔离中，-1 创建失败 示例值： 1
IfCommunity	是	是	Bool	是否开源实例。开源：true，不开源：false 示例值： false

PriceObject

价格对象

被如下接口引用：DescribeCkafkaPrice

名称	必选	允许NULL	类型	描述
Pid	是	否	String	pid 示例值： 11413
Price	是	否	Int64	价格 示例值： 69900
Value	是	否	Int64	值 示例值： 1
PriceModel	是	否	String	价格模式 示例值： specified
PriceDetail	是	是	Array of PriceDetail	价格详情数组 示例值：查看
TotalCost	是	否	Int64	总消费 示例值： 69900
BillingItemCode	是	是	String	账单代码 示例值： v_ckafka_instance
SubBillingItemCode	是	是	String	子账单代码 示例值： sv_ckafka_instance_standard
RealTotalCost	是	是	Float	实际总花费 示例值： 69900
Policy	是	是	Float	方案 示例值： 100

名称	必选	允许NULL	类型	描述
PolicyDetail	是	是	PolicyDetail	方案详情 示例值：查看

DescribeGroup

DescribeGroup返回实体

被如下接口引用：DescribeGroup

名称	必选	允许NULL	类型	描述
Group	是	否	String	groupId 示例值： group-test
Protocol	是	否	String	该 group 使用的协议。 示例值： consumer

Group

组实体

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
GroupName	是	否	String	组名称 示例值： group-test

ConsumerGroupTopic

消费组主题对象

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
TopicId	是	否	String	主题ID 示例值： topic-test
TopicName	是	否	String	主题名称 示例值： topic-test

CreateTopicResp

创建主题返回

被如下接口引用：CreateTopic

名称	必选	允许NULL	类型	描述
TopicId	是	否	String	主题Id 示例值： topic-test

DescribeBrokersResponses

DescribeBrokersResponses

被如下接口引用：DescribeBrokers

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	总数 示例值： 10
Brokers	是	否	Array of DescribeBrokerResponse	Brokers 示例值：查看

Partition

分区实体

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
PartitionId	是	否	Int64	分区ID 示例值： 0

GroupInfoResponse

GroupInfo返回数据的实体

被如下接口引用：DescribeGroupInfo

名称	必选	允许NULL	类型	描述
ErrorCode	是	否	String	错误码，正常为0 示例值： 0
State	是	否	String	group 状态描述（常见的为 Empty、Stable、Dead 三种状态）： Dead：消费分组不存在 Empty：消费分组，当前没有任何消费者订阅

名称	必选	允许NULL	类型	描述
				PreparingRebalance：消费分组处于 rebalance 状态 CompletingRebalance：消费分组处于 rebalance 状态 Stable：消费分组中各个消费者已经加入，处于稳定状态 示例值： Stable
ProtocolType	是	否	String	消费分组选择的协议类型正常的消费者一般为 consumer 但有些系统采用了自己的协议如 kafka-connect 用的就是 connect。只有标准的 consumer 协议，本接口才知道具体的分配方式的格式，才能解析到具体的 partition 的分配情况 示例值： consumer
Protocol	是	否	String	消费者 partition 分配算法常见的有如下几种(Kafka 消费者 SDK 默认的选择项为 range)： range、 roundrobin、 sticky 示例值： range
Members	是	否	Array of GroupInfoMember	仅 当 state 为 Stable 且 protocol_type 为 consumer 时， 该数组才包含信息 示例值：查看
Group	是	否	String	Kafka 消费分组 示例值： group-test

GroupOffsetTopic

消费分组主题对象

被如下接口引用：DescribeGroupOffsets

名称	必选	允许NULL	类型	描述
Topic	是	否	String	主题名称 示例值： topic-test
Partitions	是	是	Array of GroupOffsetPartition	该主题分区数组，其中每个元素为一个 json object 示例值：查看

Assignment

存储着分配给该消费者的 partition 信息

被如下接口引用：DescribeGroupInfo

名称	必选	允许NULL	类型	描述
Version	是	否	Int64	assingment版本信息 示例值： 0

名称	必选	允许NULL	类型	描述
Topics	是	是	Array of GroupInfoTopics	topic信息列表 示例值：查看

PriceDetail

价格详情

被如下接口引用：DescribeCkafkaPrice

名称	必选	允许NULL	类型	描述
SinglePrice	是	否	Int64	单价 示例值： 35
UsedAmount	是	否	Int64	使用数量 示例值： 800
Cost	是	否	Int64	花费 示例值： 28000

TopicPartitionDO

分区详情

被如下接口引用：DescribeTopicAttributes

名称	必选	允许NULL	类型	描述
Partition	是	否	Int64	Partition ID 示例值： 0
LeaderStatus	是	否	Int64	Leader 运行状态 示例值： 0
IsrNum	是	否	Int64	ISR 个数 示例值： 2
ReplicaNum	是	否	Int64	副本个数 示例值： 2

UserResponse

用户返回实体

被如下接口引用：DescribeUser

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
Users	是	是	Array of User	符合条件的用户列表 示例值：查看
TotalCount	是	否	Int64	符合条件的总用户数 示例值： 1

Price

消息价格实体

被如下接口引用：DescribeCkafkaZone

名称	必选	允许NULL	类型	描述
RealTotalCost	否	否	Float	折扣价 示例值： 0
TotalCost	否	否	Float	原价 示例值： 0

User

用户实体

被如下接口引用：DescribeUser

名称	必选	允许NULL	类型	描述
UserId	是	否	Int64	用户id 示例值： 1
Name	是	否	String	用户名称 示例值： name-test
CreateTime	是	否	Datetime	创建时间 示例值： 2019-08-19 00:00:00
UpdateTime	是	否	Datetime	最后更新时间 示例值： 2019-08-19 00:00:00
CreatedTimestamp	否	否	Int64	创建时间戳 示例值： 1757340058
ModifiedTimestamp	否	否	Int64	最后更新时间戳 示例值： 1757340058

InstanceConfigDO

实例配置实体

被如下接口引用：DescribeInstanceAttributes

名称	必选	允许NULL	类型	描述
AutoCreateTopicsEnable	是	否	Bool	是否自动创建主题 示例值： false
DefaultNumPartitions	是	否	Int64	分区数 示例值： 0
DefaultReplicationFactor	是	否	Int64	默认的复制Factor 示例值： 0

ZoneResponse

查询kafka的zone信息返回的实体

被如下接口引用：DescribeCkafkaZone

名称	必选	允许NULL	类型	描述
ZoneList	是	否	Array of ZoneInfo	zone列表 示例值：查看
MaxBuyInstanceNum	是	否	Int64	最大购买实例个数 示例值： 100
MaxBandwidth	是	否	Int64	最大购买带宽 单位Mb/s 示例值： 9600
UnitPrice	是	否	Price	后付费单位价格 示例值：查看
MessagePrice	是	否	Price	后付费消息单价 示例值：查看
ClusterInfo	是	是	Array of ClusterInfo	用户独占集群信息 示例值：查看
Profession	否	是	String	专业版信息 示例值： "{{"Bandwidth":{"Maximum":1200}}"
SupportStoreQuantity	否	是	Int64	是否支持按量存储实例(1支持，0不支持，默认0) 示例值： 0
ElasticBandwidthSwitch	否	是	Int64	购买页支持开启弹性带宽：1-支持；0-不支持 示例值： 0
SpecificationType	否	否	Array of String	规格类型 示例值： ["standard","profession"]

PolicyDetail

折扣详情

被如下接口引用：DescribeCkafkaPrice

名称	必选	允许NULL	类型	描述
Total	是	否	Float	总折扣 示例值： 100
User	是	否	Float	用户个人折扣 示例值： 100
Common	是	否	Int64	官网基础折扣 示例值： 100
Activity	是	否	Int64	Activity 示例值： 6783
DiscountType	是	是	String	折扣类型 示例值： 0
DiscountId	是	是	Int64	折扣ID 示例值： 167899
PreferentialType	是	是	Int64	优惠类型 示例值： 2
DiscountSpecifiedPid	是	是	Int64	指定折扣pid 示例值： 7869
Combine	是	是	Float	Combine 示例值： 96.543208

SalesInfo

售卖信息

被如下接口引用：DescribeCkafkaZone

名称	必选	允许NULL	类型	描述
Flag	否	是	Bool	是否售卖 示例值： false
Version	否	是	String	版本 示例值： 2.4.1
Platform	否	是	String	实例规格 示例值： profession

名称	必选	允许NULL	类型	描述
SoldOut	否	是	Bool	是否售罄 示例值： false

TopicAttributesRes

主题属性返回结果实体

被如下接口引用：DescribeTopicAttributes

名称	必选	允许NULL	类型	描述
TopicId	是	否	String	主题 ID 示例值： topic-test
CreateTime	是	否	Int64	创建时间 示例值： 2021-7-26 4:16:58
Note	是	是	String	主题备注 示例值： note-test
PartitionNum	是	否	Int64	分区个数 示例值： 1
EnableWhiteList	是	否	Int64	IP 白名单开关，1：打开； 0：关闭 示例值： 1
IpWhiteList	是	否	Array of String	IP 白名单列表 示例值： ["10.0.0.50"]
Config	是	否	Config	topic 配置数组 示例值：查看
Partitions	是	否	Array of TopicPartitionDO	分区详情 示例值：查看
ReplicaNum	否	否	Int64	副本数量 示例值： 3
EnableAclRule	否	否	Int64	开启acl 示例值： 0
QuotaConfig	否	否	ClientQuotaConfigResp	客户端配额配置 示例值：查看
AclRuleList	否	否	Array of AclRuleDO	acl规则 示例值：查看

DescribeBrokerResponse

DescribeBrokerResponse

被如下接口引用：DescribeBrokers

名称	必选	允许NULL	类型	描述
ClusterId	是	否	Int64	ClusterId 示例值： 1
ZoneId	是	否	Int64	ZoneId 示例值： 12121312
ClusterName	是	是	String	ClusterName 示例值： test
BrokerIp	是	否	String	BrokerIp 示例值： 1.1.1.1
BrokerId	是	否	Int64	BrokerId 示例值： 1
MaxNetworkFlow	是	否	Uint64	MaxNetworkFlow 示例值： 100
MaxDiskSpace	是	否	Uint64	MaxDiskSpace 示例值： 100
MinPort	是	否	Uint64	MinPort 示例值： 13000
MaxPort	是	否	Uint64	MaxPort 示例值： 13500
AvailableNetworkFlow	是	否	Uint64	AvailableNetworkFlow 示例值： 100
AvailableDiskSpace	是	否	Uint64	AvailableDiskSpace 示例值： 100

InstanceAttributesRes

实例属性返回结果对象

被如下接口引用：DescribeInstanceAttributes

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例ID 示例值： ckafka-test
InstanceName	是	否	String	实例名称 示例值： ckafka测试

名称	必选	允许NULL	类型	描述
VipList	是	否	Array of VipEntity	接入点 VIP 列表信息 示例值：查看
Vip	是	否	String	虚拟IP 示例值： 10.0.0.5
Vport	是	否	String	虚拟端口 示例值： 9092
Status	是	否	Int64	实例的状态。0：创建中，1：运行中，2：删除中 示例值： 1
Bandwidth	是	否	Int64	实例带宽，单位：Mbps 示例值： 100
DiskSize	是	否	Int64	实例的存储大小，单位：GB 示例值： 200
ZoneId	是	否	Int64	可用区 示例值： 100003
VpcId	是	否	String	VPC 的 ID，为空表示是基础网络 示例值： vpc-test
SubnetId	是	否	String	子网 ID， 为空表示基础网络 示例值： subnet-test
Healthy	是	否	Int64	实例健康状态， 1：健康，2：告警，3：异常 示例值： 1
HealthyMessage	是	否	String	实例健康信息，当前会展示磁盘利用率，最大长度为256 示例值： healthy
CreateTime	是	否	Uint64	创建时间 示例值： 2019-08-19 00:00:00
MsgRetentionTime	是	否	Int64	消息保存时间,单位为分钟 示例值： 4320
Config	是	否	InstanceConfigDO	自动创建 Topic 配置， 若该字段为空，则表示未开启自动创建 示例值：查看
RemainderPartitions	是	否	Int64	剩余创建分区数 示例值： 310
RemainderTopics	是	否	Int64	剩余创建主题数 示例值： 190

名称	必选	允许NULL	类型	描述
CreatedPartitions	是	否	Int64	当前创建分区数 示例值： 90
CreatedTopics	是	否	Int64	当前创建主题数 示例值： 10
Tags	是	是	Array of Tag	标签数组 示例值：查看
ExpireTime	是	是	Uint64	过期时间 示例值： 1566144000
ZoneIds	是	是	Array of Int64	跨可用区 示例值： [100003,100004]
Version	是	是	String	kafka版本信息 示例值： 1.1.1
MaxGroupNum	是	是	Int64	最大分组数 示例值： 500
Cvm	是	是	Int64	售卖类型 示例值： 1
CpuArch	否	否	Int64	集群的 CPU 架构。1:x86; 2:arm; 按位标记架构类型，例：3 二进制为 0011，表示 x86、arm 混合架构 示例值： 1
ClusterId	否	是	Int64	实例所在集群的Id 示例值： 1
ClusterName	否	是	String	实例所在集群的名称 示例值： test
PayMode	否	否	Int64	付费模式。-1:非付费实例(默认)；0:后付费实例；1:预付费实例 示例值： 1
K8sClusterId	否	否	String	专业版实例所属 K8s 集群 ID 示例值： global
K8sClusterName	否	否	String	专业版实例所属 K8s 集群名称 示例值： global
RetentionTimeConfig	否	否	DynamicRetentionTime	动态时间配置 示例值：查看
DynamicDiskConfig	否	否	DynamicDiskConfig	动态磁盘配置 示例值：查看
SystemMaintenanceTime	否	否	String	系统维护时间 示例值： 1

名称	必选	允许NULL	类型	描述
ElasticBandwidthOpenStatus	否	否	Int64	弹性带宽状态； UN_OPEN(1, "未开通"), OPENING(16, "开通中"), OPEN_SUCCESS(32, "开通成功"), OPEN_FAILED(64, "开通失败") 示例值： 16
ProjectName	否	否	String	项目名称 示例值： projectName1
MaxMessageByte	否	否	Int64	允许最大消息 示例值： 12582912L
ProjectId	否	否	String	项目ID字段 示例值： 12
RemainingTopics	否	否	Int64	剩余topic配额 示例值： 198
PublicNetworkChargeType	否	否	String	公网带宽计费类型 示例值： BANDWIDTH_POSTPAID_BY_HOUR
RemainingPartitions	否	否	Int64	剩余parition配额 示例值： 2000
InstanceType	否	否	String	实例类型 示例值： profession
DiskType	否	否	String	磁盘类型 示例值： SSD
CompressionType	否	否	String	支持的压缩算法类型 示例值： snappy
InstanceChargeType	否	否	String	购买付费类型 PREPAID(包年包月)/ POSTPAID_BY_HOUR(按量付费) 示例值： PREPAID
ElasticBandwidthSwitch	否	否	Int64	购买页支持开启弹性带宽：1-支持；0-不支持 示例值： 1
DeleteRouteTimestamp	否	否	String	删除路由时间设置 示例值： 0000-00-00 00:00:00
MaxConnection	否	否	Int64	最大连接数 示例值： 5000
Features	否	否	Array of String	功能 示例值： 1

名称	必选	允许NULL	类型	描述
PublicNetwork	否	否	Int64	购买页公网带宽 示例值： 1

SubscribedInfo

订阅信息实体

被如下接口引用：DescribeConsumerGroup

名称	必选	允许NULL	类型	描述
TopicName	是	否	String	订阅的主题名 示例值： topic-test
Partition	是	是	Array of Int64	订阅的分区 示例值： [0]
PartitionOffset	是	是	Array of PartitionOffset	分区offset信息 示例值：查看
TopicId	否	否	String	topic的id 示例值： topic-0cpolok4

ConnectorStatus

获取数据同步任务状态返回对象

被如下接口引用：DescribeConnectorStatus

名称	必选	允许NULL	类型	描述
State	是	否	String	connector 状态。 UNASSIGNED：任务还未分配；RUNNING：connector 正在运行；PAUSED：connector 已经暂停；FAILED：任务失败；DESTROYED：任务销毁。 示例值： RUNNING
Type	是	否	String	connector 类型，有 source 和 sink 两种类型。 示例值： source

InstanceDeleteResponse

删除实例返回任务

被如下接口引用：DeleteInstance

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
FlowId	是	是	Int64	删除实例返回的任务Id 示例值： 6970

TopicResult

统一返回的TopicResponse

被如下接口引用：DescribeTopic

名称	必选	允许NULL	类型	描述
TopicList	是	是	Array of Topic	返回的主题信息列表 示例值：查看
TotalCount	是	是	Int64	符合条件的 topic 数量 示例值： 1

ConnectorResponse

Connector返回结果对象

被如下接口引用：DescribeConnector

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	符合条件的 connector 数量。 示例值： 1
ConnectorList	是	是	Array of Connector	数据同步任务列表。 示例值：查看

AclRuleDO

acl规则

被如下接口引用：DescribeTopicAttributes

名称	必选	允许NULL	类型	描述
Id	否	否	Int64	id 示例值： 2
RuleName	否	否	String	规则名称 示例值： rule1
InstanceId	否	否	String	实例类型 示例值： ckafka-test

名称	必选	允许NULL	类型	描述
PatternType	否	否	String	模式类型 示例值： preset
Pattern	否	否	String	模式 示例值： 1
ResourceType	否	否	String	资源类型 示例值： test
AclList	否	否	String	acl列表 示例值： [1]
CreateTimeStamp	否	否	String	创建时间 示例值： 1756798015
RuleList	否	否	Acl	规则列表 示例值：查看
IsApplied	否	否	Int64	是否应用 示例值： 1
UpdateTimeStamp	否	否	String	更新时间 示例值： 1756798015
Comment	否	否	String	描述 示例值： acl信息
TopicName	否	否	String	topicname 示例值： topic1
TopicCount	否	否	Int64	topic数量 示例值： 10
PatternTypeTitle	否	否	String	扩展acl匹配模式：前缀匹配 示例值： PRESET

InstanceTypeConfigResponse

获取实例规格配置返回结果

被如下接口引用：DescribeCkafkaTypeConfigs

名称	必选	允许NULL	类型	描述
InstanceTypeConfigSet	是	是	Array of InstanceTypeConfigDO	实例规格配置结果集合 示例值：查看
MaximumTopicPartition	是	是	Int64	最大主题分区 示例值： 24

名称	必选	允许NULL	类型	描述
MaximumInstanceConsumerGroup	是	是	Int64	最大实例消费组 示例值： 50

AppIdIsVipResponse

是否大客户查询结果

被如下接口引用：DescribeAppIdIsVip

名称	必选	允许NULL	类型	描述
VipFlag	是	否	Int64	0表示小客户 1 表示大客户 示例值： 0
InternalApp	是	否	Int64	0 : 表示公有云客户， 1 表示内部客户 示例值： 0

Connector

连接器实例

被如下接口引用：DescribeConnector

名称	必选	允许NULL	类型	描述
ConnectorId	是	否	String	connectorId。 示例值： connector-test
Name	是	否	String	connector 名称。 示例值： resource-test
Type	是	否	String	connector 类型。 source：导入数据到 Kafka；sink：将数据从 Kafka 导出。 示例值： KAFKA
ConnectorClass	是	否	String	执行该任务的 class 名称。 示例值： com.cloud.ckafka.replicator.K2KSourceConnector
SourceRegion	是	否	String	源所在地域。 示例值： [1]
Source	是	否	String	源地址。 示例值： source
SinkRegion	是	否	String	目的所在地域。 示例值： [0]
Sink	是	否	String	目的地址。 示例值： sink

名称	必选	允许NULL	类型	描述
Status	是	否	String	connector 当前状态。 该状态展示有一定的延迟，任务状态可通过 GetConnectorStatus 接口 获取。 UNASSIGNED：任务还未分配；RUNNING：connector 正在运行；PAUSED：connector 已经暂停；FAILED：任务失败；DESTROYED：任务销毁。 示例值： 1
Description	是	是	String	connector 描述信息。 示例值： test
CreateTime	是	是	String	创建时间。 示例值： 2024-10-30 00:00:00
UpdateTime	是	是	String	修改时间。 示例值： 2024-11-01 00:00:00

TopicDetail

主题详情

被如下接口引用：DescribeTopicDetail

名称	必选	允许NULL	类型	描述
TopicName	是	否	String	主题名称 示例值： topic-test
TopicId	是	否	String	主题ID 示例值： topic-test
PartitionNum	是	否	Int64	分区数 示例值： 6
ReplicaNum	是	否	Int64	副本数 示例值： 2
Note	是	是	String	备注 示例值： note-test
CreateTime	是	否	Int64	创建时间 示例值： 1566144000
EnableWhiteList	是	否	Bool	是否开启ip鉴权白名单，true表示开启，false表示不开启 示例值： false
IpWhiteListCount	是	否	Int64	ip白名单中ip个数 示例值： 0
ForwardCosBucket	是	是	String	数据备份cos bucket: 转存到cos 的bucket地址 示例值： cos-bucket-test

名称	必选	允许NULL	类型	描述
ForwardStatus	是	否	Int64	数据备份cos 状态： 1 不开启数据备份，0 开启数据备份 示例值： 1
ForwardInterval	是	否	Int64	数据备份到cos的周期频率 示例值： 0
Config	是	是	Config	高级配置 示例值：查看
Status	否	否	Int64	状态 示例值： 0
Tags	否	否	Tags	tag标签 示例值：查看
RetentionTimeConfig	否	否	RetentionTimeConfig	保留时间配置 示例值：查看

错误码

功能说明

如果返回结果中存在 Error 字段，则表示调用 API 接口失败。例如：

```
{
    "Response": {
        "Error": {
            "Code": "AuthFailure.SignatureFailure",
            "Message": "The provided credentials could not be validated. Please check your signature is cor
rect."
        },
        "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
    }
}
```

Error 中的 Code 表示错误码，Message 表示该错误的具体信息。

错误码列表

公共错误码

错误码	说明
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。 请在控制台检查密钥是否已被删除或者禁用，如状态正常，请检查密钥是否填写正确，注意前后不得有空格。
AuthFailure.SignatureExpire	签名过期。Timestamp 和服务器时间相差不得超过五分钟，请检查本地时间是否和标准时间同步。
AuthFailure.SignatureFailure	签名错误。 签名计算错误，请对照调用方式中的接口鉴权文档检查签名计算过程。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。

错误码	说明
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

业务错误码

错误码	说明
InvalidParameterValue.InstanceNotExist	实例不存在。
UnauthorizedOperation	未授权操作。
InvalidParameter	参数错误。
InvalidParameterValue.VpcIdInvalid	无效的 Vpc Id

错误码	说明
ResourceUnavailable	资源不可用。
InvalidParameterValue.ZoneNotSupport	zone不支持
UnsupportedOperation.BatchDelInstanceLimit	批量删除实例限制
InvalidParameterValue.SubnetNotBelongToZone	子网不属于zone
InternalError	内部错误。
FailedOperation.TradeFailed	调用计量计费接口失败
UnsupportedOperation.OssReject	Oss拒绝该操作
InvalidParameterValue.RepetitionValue	已存在相同参数。
InvalidParameterValue.NotAllowedEmpty	参数不允许为空。
InvalidParameterValue.SubnetIdInvalid	无效的子网id
InvalidParameterValue.WrongAction	Action参数取值错误。