百度智能云

Kafka 文档

目录

功能发布记录

2025

发布时间	功能概述
2025-05	【性价比提升】 • 存储分层：支持冷热数据分层，冷数据使用性价比更佳的对象存储，降低用户存储成本

2024年及以前

发布时间	功能概述
2024-12	【稳定性】 • 发布流控管理，可限制生产者生产速率，维护集群稳定 【引擎版本】 • 支持kafka 3.6.2版本
2024-04	【性价比提升】 • g5套餐支持：支持基于BCC通用型g5机型创建集群； • 边缘节点接入：边缘计算适配，提升用户数据传输效率，降低用户网络传输成本； 【体验优化】 • 公网方式：上线公网方式，支持消息链路测试； • 产品转储：支持集群创建后产品转储变更；
	【提升用户体验】 • 集群配置：支持配置的模板化、版本化管理，提升用户集群配置效率；

2024-01	• 任务管理支持操作组：支持变更操作按组进行，提升用户操作执行效率； • 存储分析：支持存储相关指标的监控和分析，便于用户感知topic、broker、磁盘维度的存储问题； • 安全组优化：支持选择自定义安全组，提升安全组的灵活性； • 公网开关：灵活开关的方式支持数据公网读写，满足用户公网读写数据的需求； 【提升运维效率】 • 集群缩容：支持集群缩容操作，提升用户自运维能力； • SSL双向认证和SASL/PLAIN认证：扩充用户数据读写认证方式，适应更多客户场景； • 集群存储策略优化：支持磁盘用量过高时，自动删除部分历史数据，保障集群正常运行； • 集群启停：快速完成集群的停止和启动，减少客户不必要成本支出；
2023-09	增加 Kafka 共享版白名单入口封禁策略，限制新用户使用 Kafka 共享版
2023-05	高可用可选择三可用区；集群监控增加部分监控指标
	• 集群监控：kafka控制台重构集群监控功能，支持对集群、节点、主机等近200个指标进行监控。支持接入BCM进行告警配置 •

2023-03	• 消息查询：上线消息查询功能，用户可以在控制台对kafka消息按时间、按位点查询 • 集群审计：接入BCT云审计，支持对33个事件记录审计 • 集群标签：接入标签功能，在集群创建和维护中支持对kafka集群进行标签维护，可使用标签进行集群筛选 • 访问配置变更：允许在集群创建后对访问配置进行修改 • 集群鉴权完善：用户使用kafka集群权限控制新增【集群】和【事务ID】资源，以及对应的写操作 • Controller历史信息：控制台可以查询当前集群的controller及修改记录
2022-08	集群配置变更：支持增加节点数量、扩充磁盘容量、升级节点规格
2022-06	专享版上线公测
2021-08	规范计费名词解释，更新计费region信息。
2019-06	提供API接口。Kafka API采用类似REST风格，提供对Kafka topic的创建与管理操作支持。包括集群管理，证书管理，权限管理和Consumer Group管理相关接口
2019-04	• 新增保定区域支持百度消息服务 • 新增Consumer Group管理功能。Consumer Group是百度消息服务提供的可扩展且具有容错性的消费者机制。通过console创建的Consumer Group享有完善的安全保护机制，强烈推荐用户采用这种方式使用消费者组功能
2018-12	完善用户权限控制，允许用户管理topic和订单的子账号权限

2018-12	完善用户权限控制，允许用户管理topic和证书的子账号权限
2018-11	BMS接入IAM系统，支持多用户访问，实现了多用户协同开发。项目管理者可以创建多个IAM用户，为其IAM用户分配资源和操作权限
2018-10	• 完善用户权限控制，允许用户管理topic和证书的子账号权限 • 限制topic读写流量，根据用户购买的topic分区数量，进行流量配额

专享版

产品描述

产品介绍

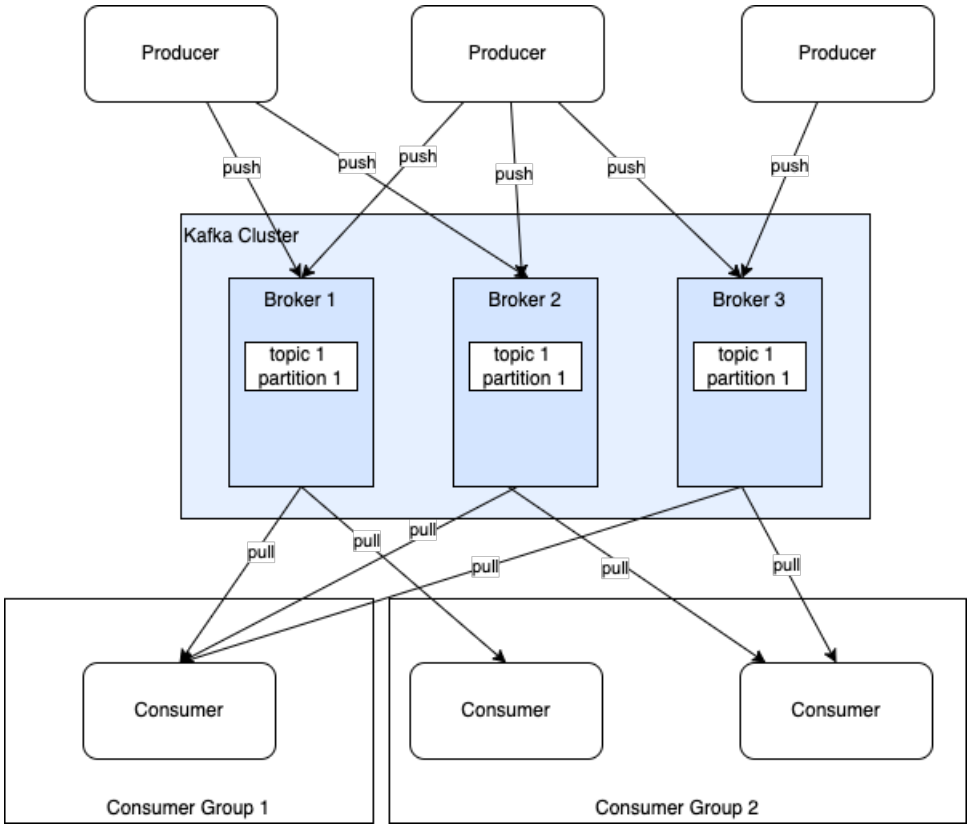
概念

专享版消息服务 for Kafka是基于开源Apache Kafka提供的分布式、高吞吐、可扩展的全托管消息队列服务。所有集群资源根据实际业务场景按需申请，支持主题、权限等多维度的灵活配置，随买随用，并在性能、扩展性、数据安全及后续运维等多方面为用户提供持续保障，秉承公司“简单可依赖”的企业文化，让您用最低的成本，享受最优质的消息服务。

产品架构

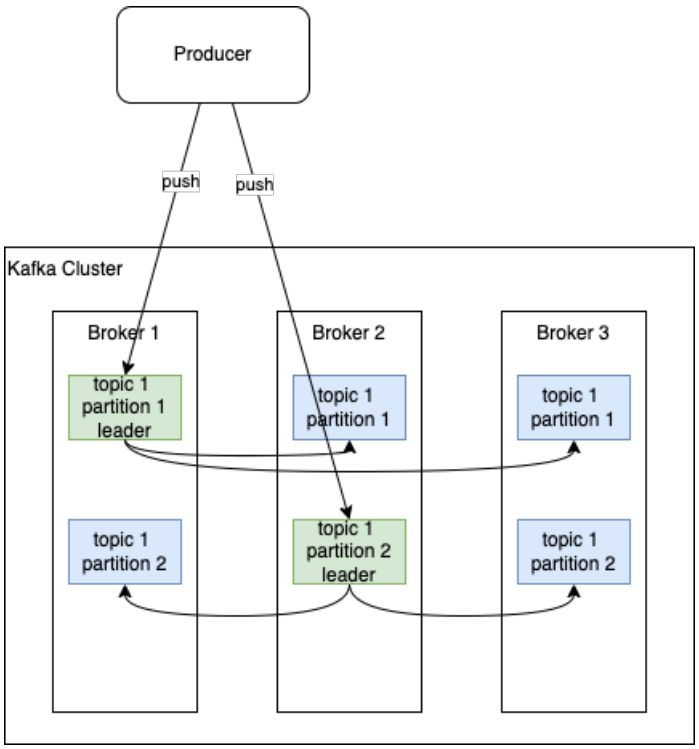
服务架构

kafka 集群服务由多个 broker 组成，服务架构如下：



- 主题（Topic）：用于分类消息，可视为特定的消息流或消息队列，生产者向指定主题推送消息，消费者订阅主题并拉取消息。
- 消息节点(Kafka Broker): 组成 Kafka 集群服务的单个 kafka 进程，对应一台独立的 BCC 云主机。
- 生产者(Producer): 生产消息的系统或者服务，向服务推送消息，写入到指定的主题。
- 消费组(Consumer Group): 消费组由一个或多个消息者(Consumer)组成，消费者订阅单个或多个主题，主动从 Broker 拉取消息。

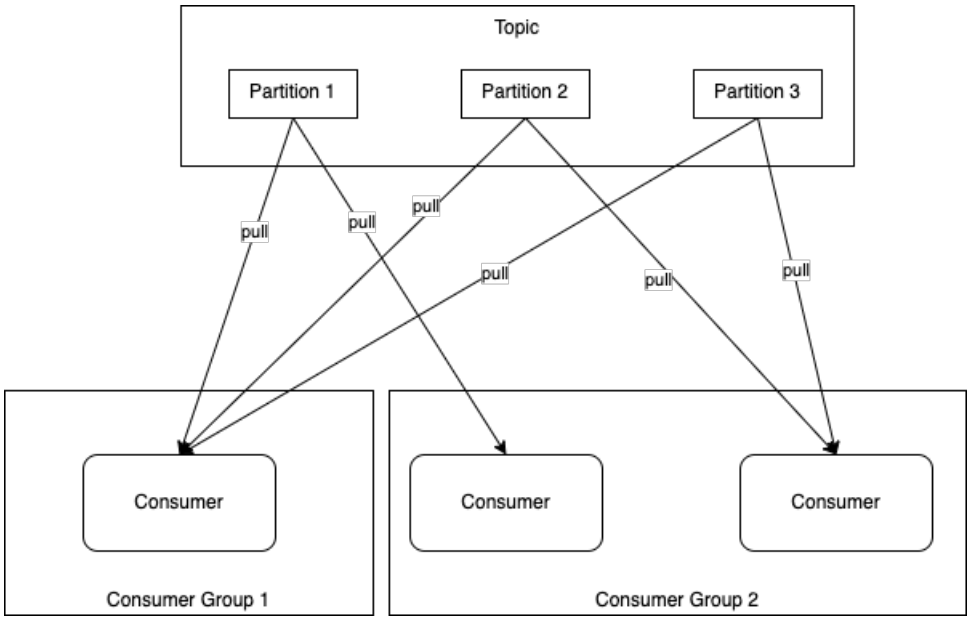
数据分布



- 分区（Partition): 每个 Topic 由多个 Partition 组成，Partition 对应实际存储消息的文件，可以分布在集群的多个 Broker 上，以此增加读写并发性，提高服务吞吐量。

- 副本（Replica): 每个 Partition 可以配置多个 Replica，同一 Partition 的不同副本必须分布在不同的 Broker 上。同一个 Partition 的多个 Replica，只有一个 Replica 作为 Leader 提供读写服务，其他 Replica 都作为 Follower 从 Leader 同步数据。当作为 Leader 的 Replica 因故障下线后，其他同步的 Follower 会重新选举 Leader 提供服务，确保数据安全并提高服务的可靠性。

消费组规则



- 任何消费者必须加入一个消费组。
- 同一个消费组下，不同消费者会被分配到不同的 Partition，每个 Partition 最多只能被分配给一个消费者，因此每条消息只会被一个消费者处理，此时相当于点对点模式。
- 同一个消费组下 Partition 只能分配到 1 个消费者，因此同一个消费组下消费者数量不要超过订阅的分区数。
- 一个 Topic 可以同时被多个消费组订阅，不同消费组维护独立的消费记录，Topic 中的消息相当于广播给所有订阅的消费组，此时相当于发布/订阅模式。

产品优势

兼容开源

- 100%兼容开源社区版本，业务无需改动即可快速完成迁移上云，在节约成本的同时使您更加专注于业务开发。

运维无忧

- 配备完整的监控告警等运维服务，与百度智能云专业的云监控（BCM）服务打通，日常运维提供丰富的监控指标，出现故障自动发送告警，提供7×24小时的实时监控服务，为您的系统保驾护航。

规格灵活

- 可对专享实例节点、带宽与存储资源进行灵活配置组合，自定义集群下主题的分区数、副本数。

弹性扩容

- 支持对专享实例进行扩容及升配等操作，更好地适应业务规模的增长。

数据安全

- 利用SASL机制对用户身份进行认证，SSL协议加密的数据传输，保证数据在传输的过程中不被窃听或者篡改，确保客户数据的安全。

稳定可靠

- 独特的高可用架构设计，将多副本均匀分布在不同介质，防止数据因地域或个别机器故障而导致数据丢失，提升容灾能力。服务可用性高达99.95%。

产品规格

产品规格

根据不同的业务场景，消息服务 for Kafka专享版支持用户灵活配置集群规格，可由节点规格、节点数量、磁盘类型、磁盘大小以及网络带宽组成最为适合实际业务场景所需要的集群配置组合，在满足对业务稳定支持的前提下，最大化降低成本，提升性价比。

节点规格

- Kafka.g4.c2m8
- Kafka.g4.c4m16
- Kafka.g4.c8m32
- Kafka.g4.c16m64
- Kafka.g4.c24m96
- Kafka.g4.c32m128

> 注：以Kafka.g4.c2m8为例，该规格对应的底层虚机规格为2核vCPU+8GiB内存

节点数量

集群支持最小3节点，最大30节点。（提交工单可酌情增加最大节点数限制）

磁盘类型

- 增强型SSD
- 高性能云磁盘

磁盘容量

单节点支持最小100GB，最大16000GB

网络带宽

单节点支持最小1Mbps，最大200Mbps

使用限制

使用前提

1. 对分布式系统和消息队列的基本概念有一定的了解，以便更好地理解和配置 Kafka。
2. 具备一定的硬件资源，包括足够的内存、CPU 和磁盘空间，以支持 Kafka 服务器的运行和数据存储。
3. 规划好主题（Topic）、分区（Partition）和副本（Replication）的设计，以满足业务需求和性能要求。
4. 确定合适的消息保留策略，根据业务需求设置消息的保留时间和清理机制。

使用限制

1. 资源消耗：Kafka 在处理大量消息和高并发场景时，可能会消耗大量的系统资源。如果资源不足，可能会导致性能下降、消息积压或系统不稳定。

2. 数据一致性：虽然 Kafka 尽力保证数据的可靠性和一致性，但在某些情况下，如网络分区或服务器故障，仍可能出现数据丢失或不一致的情况。需要根据业务需求采取适当的容错和恢复措施。
3. 复杂的配置：Kafka 的配置选项众多，需要仔细配置以达到最佳性能和可靠性。不正确的配置可能会导致各种问题。
4. 监控和管理：为了确保 Kafka 系统的正常运行，需要进行有效的监控和管理，包括监控服务器状态、主题和分区的性能、消息流量等。这需要一定的运维经验和工具支持。
5. 版本兼容性：如果使用的 Kafka 版本与其他相关组件（如生产者和消费者客户端）的版本不兼容，可能会导致功能异常或无法正常工作。在升级或集成时，需要注意版本兼容性问题。

基本概念

本章旨在解释Kafka中所包含的相关词汇和术语，以便您能够更深入地理解相关概念并有效使用该产品。

词汇名	概念
Broker	Kafka 服务器节点，负责存储和处理消息。
Topic	消息的主题，生产者将消息发送到特定的主题，消费者从感兴趣的主题中读取消息。
Partition	主题可以被分为多个分区，每个分区是一个有序的消息序列。分区可以分布在不同的 Broker 上，以实现并行处理和可扩展性。
Producer	向 Kafka 发送消息的应用程序，通过网络与 Broker 进行通信来提交消息。
Consumer	消费者。从 Kafka 读取消息的应用程序，也通过网络连接到 Broker 获取消息。
Offset	偏移量，用于标识消费者在分区中读取的位置。每个分区中的消息都有一个唯一的偏移量，消费者通过跟踪偏移量来记录已经读取的消息位置。
Zookeeper	Kafka 依赖 Zookeeper 来进行集群的协调管理，例如保存 Broker 信息、主题和分区的配置信息、消费者组的信息等。
Replication	副本机制，为了提高系统的可靠性，Kafka 对每个分区的数据进行复制，存储在多个 Broker 上。
Network Protocol	网络协议。Kafka 定义了特定的网络协议来规范生产者、消费者、Broker 之间的通信方式和消息格式等。
ACL	访问控制列表。用于对资源进行精细化的权限管理，规定操作权限。

应用场景

实时数据流处理

Kafka提供了高吞吐量和低延迟的消息传递机制，适用于实时数据流处理场景，如实时日志处理、实时监控、实时推荐等。

分布式日志集中存储

Kafka可以作为分布式日志集中存储系统，用于收集、存储和分发日志数据，如应用日志、操作日志、系统日志等。

数据集成和数据管道

Kafka可以用作数据集成和数据管道的中间件，在不同系统之间传递数据，实现数据的异步传输和解耦。

消息队列和事件驱动架构

Kafka可以作为消息队列使用，用于处理异步消息和事件驱动的架构，支持消息的发布订阅和消息的队列处理。

大数据处理和流处理

Kafka可以与大数据处理框架如Hadoop、Spark、Flink等集成，支持大规模数据的处理和分析，实现实时数据流处理和批处理任务。

产品定价

计费说明

计费方式说明

具体的计费价格请参见 价格详情页。

计费项

百度智能云消息服务 for Kafka专享版服务计费项如下：

计费项	说明
实例规格	• Kafka实例下各节点的配置规格。 • 各规格均为1：4的vCPU内存比（较高的内存比可供PageCache提升读写效率）。
磁盘容量	• 消息队列Kafka版支持的磁盘类型包括高性能云磁盘和增强型SSD。 • 不同磁盘类型的磁盘单价不同。 • 磁盘类型一经下单，不支持更改，请谨慎选择。 • 磁盘容量为Kafka实例存储空间。使用时，消息存储可选择不同副本数，如：实际数据总量为1000GB，3副本，则至少需购买1000GB × 3 = 3000GB 磁盘容量（建议在此基础上再保留30%余量）。
公网流量	• 实例支持开通公网访问，按节点流量带宽进行计费。 • 公网流量分为双通道，读写一致。购买时请按照读写流量峰值的最大值购买公网流量带宽。

计费方式

消息服务 for Kafka专享版支持预付费（包年包月计费）和后付费（按量计费）两种计费方式，详细的计费规则说明如下：

计费方式	说明	注意事项
预付费—包年包月计费	• 即包年、包月付费，在创建实例时需要支付费用。 • 适合业务稳定的长期需求，费用比后付费更低，且购买时间越长，折扣越多。	预付费实例支持升级配置，不支持降级配置和提前释放。如有特殊情况需释放请提交工单处理。
后付费—按量计费	• 即按分钟计费，根据实例配置在北京时间整点扣费并生成账单。 • 出账单时间是当前计费周期结束后1小时内。 • 适合业务瞬间有较大变化的场景，资源使用完即可立即释放，降低成本。	• 后付费实例可随时被释放。 • 后付费实例支持升级配置，不支持降级配置。 • 购买前需保证账户无欠款，且账户余额和可用代金券总和≥100元。

续费说明

续费定义

- 预付费实例：增加使用时间。
- 后付费实例：无到期时间，无需续费。

- 续费前：请保证账户余额充足，余额＞100元。 **续费说明** 消息服务 for Kafka专享版支持手动续费和自动续费两种续费方式，用户可以根据不同的应用场景选择合适的续费方式，推荐使用自动续费。两种不同续费方式的使用场景如下：
- 手动续费：初期使用消息服务 for Kafka专享版 ，有运维人员按时进行续费操作；
- 自动续费：适用于长期使用消息服务 for Kafka专享版 ，对服务可用性、数据安全性、运维自动化要求更高的场景。

变配规则说明

用户可以根据需要变更实例配置，具体的变配及计费说明如下：

计费方式	变配规则	计费说明
预付费—包年包月	支持升级配置，不支持降级配置。 以天为最小计费单位，不足一天按一天计。 变更配置实例是否享受预付费折扣，以剩余时间长度是否满1年、2年、3年为准。	升级需付金额=(合同剩余天数／合同总天数）×(新实例金额－原实例金额)； 如，包月购买A配置实例，当月天数为30天，在使用9.5天时升级为B配置实例，则需要支付的金额为：((30-10)÷30)×(B配置包月价格-A配置包月价格)
后付费—按时付费	支持升级配置，不支持降级配置。 以分钟为最小计费单位，不足一分钟按一分钟计。	升级需付金额=新实例金额

余额不足提醒和欠费处理

预付费处理方法

- 余额不足提醒
  - 在实例到期前7天，系统将给您发送“即将到期”通知。
- 欠费处理方法
  - 实例到期后停服，数据为您保留7天。停服期间不收取费用，系统将给您发送续费通知。
  - 若实例到期时间超过7天，实例将被释放，数据无法恢复。
  - 在释放实例前1天和释放实例时，系统将给您发送释放实例通知。

后付费处理方法

- 余额不足提醒
  - 根据您最近3天的账单金额来判断您的账户余额（含可用代金券）是否足够支付未来3天的费用，若不足以支付，系统发送续费提醒。
  - 根据您最近1天的账单金额来判断您的账户余额（含可用代金券）是否足以支付未来1天的费用，若不足以支付，系统发送续费提醒。
- 欠费处理
  - 北京时间整点检查您的账户余额是否足以支付本次 账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。
  - 欠费后立即停服，数据为您保留7天，期间不收取费用，7天内未充值则释放实例，释放前1天和释放实例时系统会分别给您发送释放通知。

快速入门

概述

概述

本系列文档将为您介绍使用专享版消息服务 for Kafka的基本流程，由通过控制台创建Kafka专享版集群实例、创建主题、配置用户权限、客户端设置，直至最终完成消息收发，帮助您加速熟悉过程，成功对接业务。

除了本文档中介绍的方法，您也可以使用API直接对实例进行相关管理操作。

前提条件

1. 确保完成虚拟私有云（Virtual Private Cloud，以下简称VPC）VPC网络及子网相关配置
    - VPC需与Kafka集群所属相同区域
2. 确保存在可用的安全组

操作流程 步骤一：创建Kafka集群实例

步骤二：创建主题

步骤三：配置权限认证

步骤四：访问Kafka集群

- 使用PLAINTEXT协议访问集群
- 使用SASL_PLAINTEXT协议访问集群
- 使用SASL_SSL协议访问集群

步骤一：创建Kafka集群实例

概述

本文介绍如何通过消息服务 for Kafka专享版管理控制台创建Kafka专享集群实例。

前提条件

- 已完成百度智能云账号注册，请参考注册和登录。
- 完成实名认证，操作细节请参考实名认证，实名认证之后才可购买消息服务 for Kafka集群。
- 已创建VPC、子网及可用安全组

具体操作步骤

1. 登录 百度智能云消息服务 for Kafka专享版控制台。找到智能大数据>数据集成>消息服务for Kafka，单击进入。
2. 点击集群列表页面左上方**“创建集群”**按钮进入集群创建页
3. 在集群创建页面，配置以下信息

配置项	说明
地域	实例所在的地域，即实例所在的地理位置。您也可在页面左上角切换。 • 购买后不能更换，请谨慎选择。 • 建议您将Kafka集群与需要连接的生产者及消费者创建于同一个地域。
付费方式	• 预付费： 即包年、包月付费，在创建实例时需要支付费用。适合业务稳定的长期需求，价格比按量付费更实惠，且购买时长越长，折扣越多。 • 后付费：属于按量付费，即按小时扣费，根据实例配在北京时间整点扣费并生成账单。适合短期需求，用完可

	立即释放实例，节省费用。
集群名称	用于区分不同集群的唯一性，具体规则如下：不超过65个字符，仅支持数字/字母/特殊符号(_/-.)。
集群版本	Kafka服务端版本。 当前支持2.7，2.8，3.6，后续逐步支持更多版本。
集群配置	可以使用默认配置或者自定义配置，默认配置参考 配置参数介绍，自定义配置可参考 创建集群配置
部署方式	• 高性能模式：所有节点均部署在同一可用区下，减少跨可用区传输过程中造成的延迟。 • 节点平均分布在多个可用区，可对集群进行多可用区灾备处理。高可用模式支持选择2个或3个可用区，推荐使用3个可用区。
部署集	开启部署集后，Kafka集群中的节点会严格按照物理服务器打散，保障在硬件故障等异常情况下的服务高可用性。具体可参考部署集。
可用区	可用区代表物理资源相对隔离的机房，可以帮助用户建立高可用架构. 详细介绍可参考可用区说明。
节点类型	支持2、4、8、16、24、32核，CPU核数与内存比例统一为1：4，共计6种不同类型。
单区节点数	单可用区下节点数目。
类型磁盘	• 高性能云磁盘：IO较高，且具有较高性价比； • 增强型SSD盘：单盘最高达 100 万随机 IOPS 和百微秒级别时延性能； 详细介绍可参考云磁盘产品介绍。
单节点容量	单节点对应数据存储空间。
磁盘水位处理	开启磁盘水位处理后，集群将根据容量阈值策略自动调整磁盘空间，详情可参见：磁盘水位处理。
网络实例	选择Kafka集群绑定的专有网络VPC及当前所选可用区下的子网。
公网访问	Kafka专享版支持通过公网访问方式访问集群，开启后根据用户所选带宽进行收费。
产品间转储	当Kafka集群需要被其他产品进行数据转储时开启。
权限控制	可在此开启ACL访问权限功能。开启后可通过界面对用户权限进行相关授权操作。Kafka ACL概念
认证方式	支持以下认证方式： • None（无需身份认证） • SASL/SCRAM身份认证 • SASL/PLAIN身份认证 • SSL双向认证
集群标签	Kafka集群创建后绑定的标签信息，通过标签可以对资源进行快速的分类和查看标签相关的账单信息，详情可参考标签管理。

4. 点击**“下一步”**按钮进入配置信息确认页，可以在这里看到选择的实例的套餐和价格。

5. 确认全部信息后，点击**“提交订单”**按钮，跳转至支付页面，确认付款后订单完成。

6. 购买成功后，返回集群列表页面，可看到已创建的Kafka集群实例，以及实例状态、付费方式（到期时间）、版本及可用区

信息等。

7. 在集群列表中，点击集群名称进入集群详情页，集群详情页展示整体状态及集群配置等信息；通过选择其它管理模块可对集群进行各类管理操作。

步骤二：创建主题

概述

本文介绍在已完成消息服务 for Kafka专享版集群创建后，如何通过管理控制台创建主题。

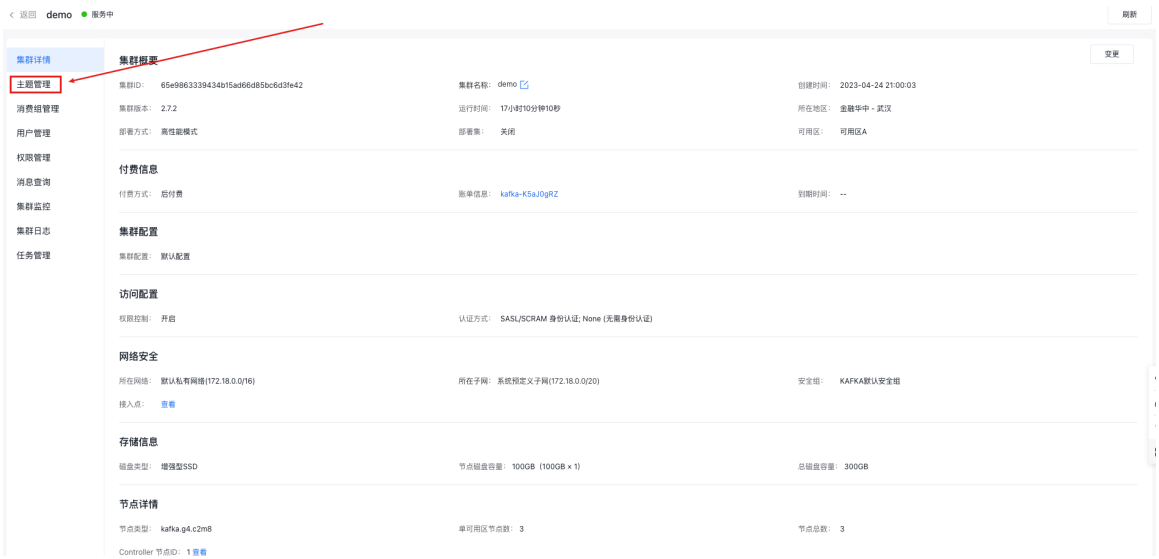
前提条件

- 已完成百度智能云账号注册
- 步骤一：已创建Kafka集群

具体操作步骤

1. 登录百度智能云消息服务 for Kafka专享版控制台。
2. 在集群列表中，点击对应的集群名称进入该集群的集群详情页。

4. 左侧模块选择栏选择**“主题管理”**。



6. 点击主题列表左上角**“创建主题”**按钮进入主题创建页。

8. 在集群创建页面，配置以下信息：

配置项	说明
主题名称	主题名称，具体规则如下：大小写字母、数字以及_-特殊字符，长度1-125 • 创建后不能更换，请谨慎选择。
分区数	主题对应分区数。 • 创建后支持对分区数进行扩容操作，不支持缩容
副本数	主题副本数目。 • 创建后不能更换，请谨慎选择。
高级配置	点击**“添加配置项”**可对高级配置项进行配置。当前支持对6项主题高级配置项内容进行变更，若不单独进行配置，则统一取默认值。 • 创建后支持对所有配置项进行编辑操作

主题的高级配置项可以参考以下说明进行配置：

配置项	默认值	说明
cleanup.policy	delete	Kafka对于旧日志数据的保留策略，可以选择：delete：旧的日志数据会被删除。compact：对旧日志信息进行压缩。
max.message.bytes	1048588 bytes	Kafka允许单次发送批数据的最大大小。
message.timestamp.type	CreateTime	消息中的时间戳来源，可以选择：CreateTime：生产者创建消息时的时间戳。LogAppendTime：服务端追加消息时的时间戳。
min.insync.replicas	1	当生产者发送消息时，将acks参数设置为all时，该配置项会确保最少多少个节点成功写入数据，如果成功写入数据的节点数小于该值，生产者则会出现异常。
retention.ms	172800000 ms	消息保留时间，超过该时间后消息会过期删除。
segment.ms	604800000 ms	控制日志的滚动时间，超过该时间后的旧数据可以进行删除或者压缩。

6. 点击**“确定”**按钮完成主题创建。

步骤三：配置权限认证

概述

本文介绍在完成消息服务 for Kafka专享版集群创建后，如何通过管理控制台进行权限及认证相关的配置。

前提条件

- 已完成百度智能云账号注册
- 步骤一：创建Kafka集群
- 步骤二：创建主题

具体场景

场景一：开启权限管理功能（ACL ON）+ 使用SASL/SCRAM身份认证方式

此场景下的集群，用户使用服务时需首先通过SASL/SCRAM（用户名密码）的身份认证并且对需要访问的具体主题或消费组有相应订阅、发布权限并在完成连接后才可执行操作。

具体操作步骤

1. 登录百度智能云消息服务 for Kafka专享版控制台。
2. 在集群列表中，点击对应的集群名称进入该集群的集群详情页。
3. 创建用户
    1. 左侧模块选择栏选择**“用户管理”**。
    2. 点击用户列表左上角**“创建用户”**按钮进入用户创建页。
    3. 在用户创建页面，根据界面提供的限制信息配置用户名及密码。
    4. 点击**“确认”**按钮完成创建用户
4. 配置权限
    1. 左侧模块选择栏选择**“权限管理”**。
    2. 点击用户列表左上角**“创建权限”**按钮进入权限创建页。
    3. 在权限创建页面，配置以下信息：

配置项	**说明**
用户名	通过用户管理创建的用户名。
资源类型	当前权限规则对应的资源类型，主题、消费组、集群以及事务ID。

匹配模式	与具体资源名称的匹配方式，精确匹配或前缀匹配。
	• 精确匹配
	  • 需与具体资源名完全匹配
	  • 当前模式下，可输入“*”与全部资源进行匹配
	• 前缀匹配
	  • 资源名称的前缀与匹配内容相同即可
	  • 如，当前有名为“test1”，“test2”的两个主题，选择前缀匹配并在匹配内容处输入“test”即可对以上两个主题及后续任何以“test”为前缀的主题完成相应授权

匹配内容	基于匹配模式的选择，具体对应资源名称的匹配内容。 • 资源类型为集群时，匹配内容固定为“kafka-cluster”
操作类型	当前权限规则允许的具体操作，订阅或发布 • 资源类型为主题时，支持“订阅”、”发布“ • 资源类型为消费组时，仅支持“订阅”操作 • 资源类型为集群时，操作类型固定为“幂等写” • 资源类型为事务ID，操作类型固定为“写”

场景二：关闭权限管理功能（ACL OFF）+ 使用SASL/SCRAM身份认证方式

此场景下的集群，用户使用服务时需通过SASL/SCRAM（用户名密码）的身份认证，认证通过并完成连接后即可对集群下的所有资源执行订阅、发布的操作。

具体操作步骤

1. 登录百度智能云消息服务 for Kafka专享版控制台。
2. 在集群列表中，点击对应的集群名称进入该集群的集群详情页。
3. 创建用户
    1. 左侧模块选择栏选择**“用户管理”**。
    2. 点击用户列表左上角**“创建用户”**按钮进入用户创建页。
    3. 在用户创建页面，根据界面提供的限制信息配置用户名及密码。
4. 点击**“确认”**按钮完成创建用户

场景三：关闭权限管理功能（ACL OFF）+ 使用None（无需身份认证）

此场景下的集群，用户使用服务时无需进行任何认证，完成连接后即可对集群下的所有资源执行订阅、发布的操作。

步骤四：访问Kafka集群

使用PLAINTEXT协议访问集群

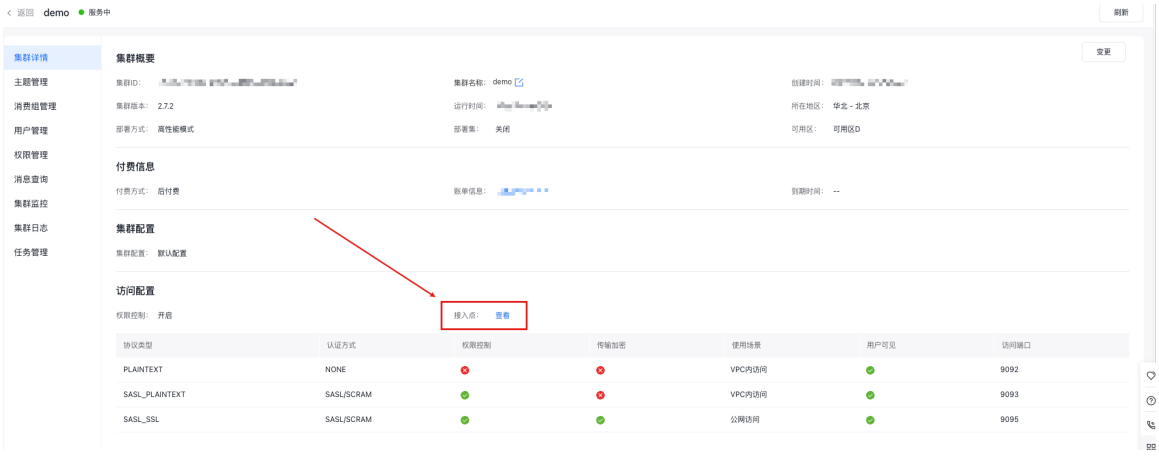
本文介绍创建完集群后，如何使用Kafka脚本通过PLAINTEXT协议访问Kafka集群，PLAINTEXT协议需要在**VPC网络**下进行使用。如果想使用Java、Go、PHP等语言访问Kakfa集群，请参考开发指南。

前提条件

- 步骤一：创建Kafka集群
- 步骤二：创建主题
- 安装JDK1.8及以上版本：JDK下载地址。

具体操作步骤

1. 下载Kafka 2.7.2安装包，上传到Kafka集群所在的同一VPC网络下并进行解压：kafka_2.13-2.7.2.tgz，其中解压后的bin目录为Kafka提供的可执行脚本。
2. 在**集群详情**中，点击**接入点**查看**PLAINTEXT协议**所使用的访问地址和端口。



4. 进入Kafka安装包解压后的bin目录下，使用Kafka脚本访问集群。

- 查看Topic列表：

```
kafka-topics.sh --bootstrap-server <PLAINTEXT接入点地址> --list
```

- 生产者发送消息，输入命令后会出现">"的标识，此时输入内容并且按下回车即可发送消息：

```
kafka-console-producer.sh --bootstrap-server <PLAINTEXT接入点地址> --topic <Topic名称>
```

- 消费者消费消息：

```
kafka-console-consumer.sh --bootstrap-server <PLAINTEXT接入点地址> --topic <Topic名称> --group <消费组ID> --from-beginning
```

4. 如果通过上述命令能够正常发送和消费消息，则说明当前VPC网络环境下能够通过PLAINTEXT协议正常访问Kafka集群，可以参考开发指南中各种语言访问Kafka集群来实现更复杂的逻辑。

使用SASL_PLAINTEXT协议访问集群

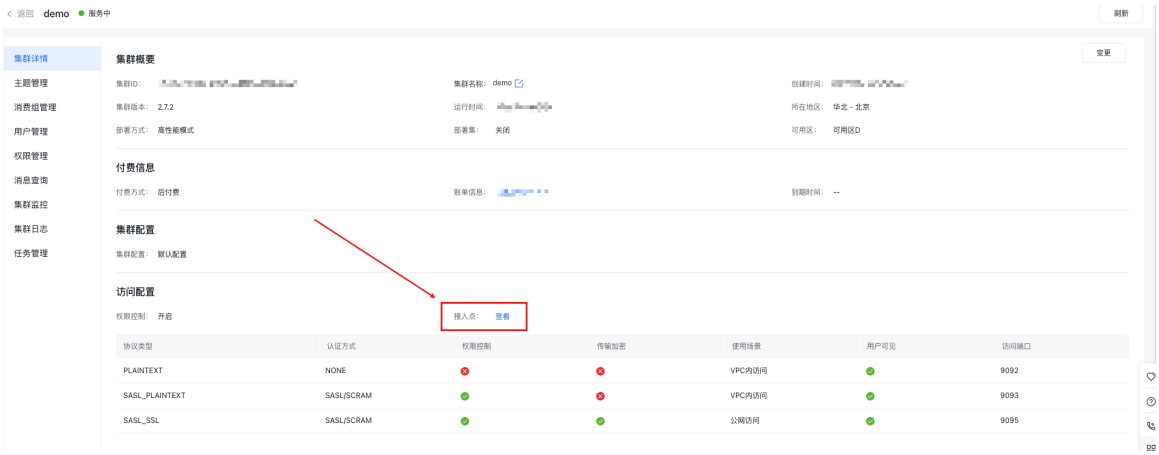
本文介绍创建完集群后，如何使用Kafka脚本通过SASL_PLAINTEXT协议访问Kafka集群，SASL_PLAINTEXT协议需要在**VPC网络**下进行使用。如果想使用Java、Go、PHP等语言访问Kakfa集群，请参考开发指南。

前提条件

- 步骤一：创建Kafka集群
- 步骤二：创建主题
- 步骤三：配置权限认证
- 安装JDK1.8及以上版本：JDK下载地址。

具体操作步骤

1. 下载Kafka 2.7.2安装包，上传到Kafka集群所在的同一VPC网络下并进行解压：kafka_2.13-2.7.2.tgz，其中解压后的bin目录为Kafka提供的可执行脚本。
2. 在**集群详情**中，点击**接入点**查看**SASL_PLAINTEXT协议**所使用的访问地址和端口。



3. 创建JAAS配置文件：kafka_client_jaas.conf，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择。内容格式如下：

SCRAM-SHA-512：

```
KafkaClient {
  org.apache.kafka.common.security.scram.ScramLoginModule required
  username="{username}"
  password="{password}";
};
```

PLAIN：

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

其中，{username}和{password}为步骤三：配置权限认证时创建的用户名和密码。

4. 创建kafka.properties配置文件，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式选择。内容格式如下：

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
###### sasl.mechanism=PLAIN
```

5. 指定第3步中所配置的JAAS文件，需要设置完整的文件路径：

```
export KAFKA_OPTS='-Djava.security.auth.login.config=/***/kafka_client_jaas.conf'
```

6. 进入Kafka安装包解压后的bin目录下，使用Kafka脚本访问集群。

- 查看Topic列表，需要使用--command-config配置项来指定第4步中创建的kafka.properties文件，需要设置完整的文件路径：

```
kafka-topics.sh --bootstrap-server <SASL_PLAINTEXT接入点地址> --list --command-config /***/kafka.properties
```

- 生产者发送消息，输入命令后会出现">"的标识，此时输入内容并且按下回车即可发送消息：

```
kafka-console-producer.sh --bootstrap-server <SASL_PLAINTEXT接入点地址> --topic <Topic名称> --producer.config /***/kafka.properties
```

- 消费者消费消息：

```
kafka-console-consumer.sh --bootstrap-server <SASL_PLAINTEXT接入点地址> --topic <Topic名称> --group <消费组ID> --from-beginning --consumer.config /***/kafka.properties
```

7. 如果通过上述命令能够正常发送和消费消息，则说明当前VPC网络环境下能够通过SASL_PLAINTEXT协议正常访问Kafka集群，可以参考开发指南中各种语言访问Kafka集群来实现更复杂的逻辑。

使用SASL_SSL协议访问集群

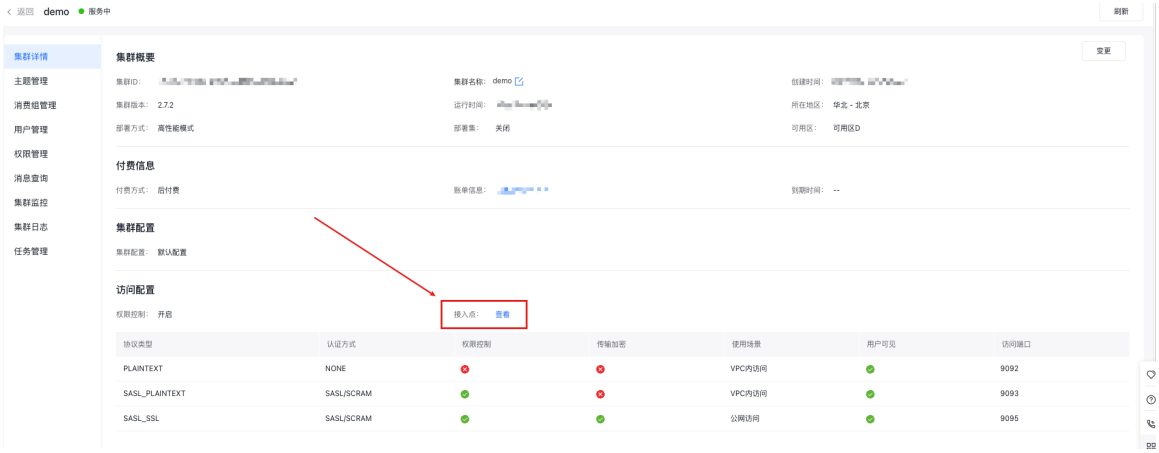
本文介绍创建完集群后，如何使用Kafka脚本通过SASL_SSL协议访问Kafka集群，SASL_SSL协议需要在**公网网络**下进行使用。如果想使用Java、Go、PHP等语言访问Kakfa集群，请参考开发指南。

前提条件

- 步骤一：创建Kafka集群
- 步骤二：创建主题
- 步骤三：配置权限认证
- 安装JDK1.8及以上版本：JDK下载地址。

具体操作步骤

1. 下载Kafka 2.7.2安装包，并进行解压：kafka_2.13-2.7.2.tgz，其中解压后的bin目录为Kafka提供的可执行脚本。
2. 在**集群详情**中，点击**接入点**查看**SASL_SSL协议**所使用的访问地址和端口。



4. 创建JAAS配置文件：kafka_client_jaas.conf，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择内容格式如下：

SCRAM-SHA-512：

```
KafkaClient {
  org.apache.kafka.common.security.scram.ScramLoginModule required
  username="{username}"
  password="{password}";
};
```

PLAIN：

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

其中，{username}和{password}为步骤三：配置权限认证时创建的用户名和密码。

4. 下载证书文件：如何下载证书？。
5. 创建kafka.properties配置文件，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式选择。内容格式如下：

```
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-512
###### sasl.mechanism=PLAIN
###### 第4步下载的SSL证书文件路径
ssl.truststore.location=/***/client.truststore.jks
###### SSL证书的密码，固定为bms@kafka
ssl.truststore.password=bms@kafka
###### 固定为空
ssl.endpoint.identification.algorithm=
```

6. 指定第3步中所配置的JAAS文件，需要设置完整的文件路径：

```
export KAFKA_OPTS='-Djava.security.auth.login.config=/***/kafka_client_jaas.conf'
```

7. 进入Kafka安装包解压后的bin目录下，使用Kafka脚本访问集群。

- 查看Topic列表，需要使用--command-config配置项来指定第5步中创建的kafka.properties文件，需要设置完整的文件路径：

```
kafka-topics.sh --bootstrap-server <SASL_SSL接入点地址> --list --command-config /***/kafka.properties
```

- 生产者发送消息，输入命令后会出现">"的标识，此时输入内容并且按下回车即可发送消息：

```
kafka-console-producer.sh --bootstrap-server <SASL_SSL接入点地址> --topic <Topic名称> --producer.config
/***/kafka.properties
```

- 消费者消费消息：

```
kafka-console-consumer.sh --bootstrap-server <SASL_SSL接入点地址> --topic <Topic名称> --group <消费组ID> -
-from-beginning --consumer.config /***/kafka.properties
```

8. 如果通过上述命令能够正常发送和消费消息，则说明当前网络环境下能够通过SASL_SSL协议正常访问Kafka集群，可以参考开发指南中各种语言访问Kafka集群来实现更复杂的逻辑。

使用SSL协议访问集群

本文介绍创建完集群后，如何使用Kafka脚本通过SSL协议访问Kafka集群，SSL协议需要在**公网网络 或者 VPC网络**下进行使用。如果想使用Java、Go、PHP等语言访问Kakfa集群，请参考开发指南。

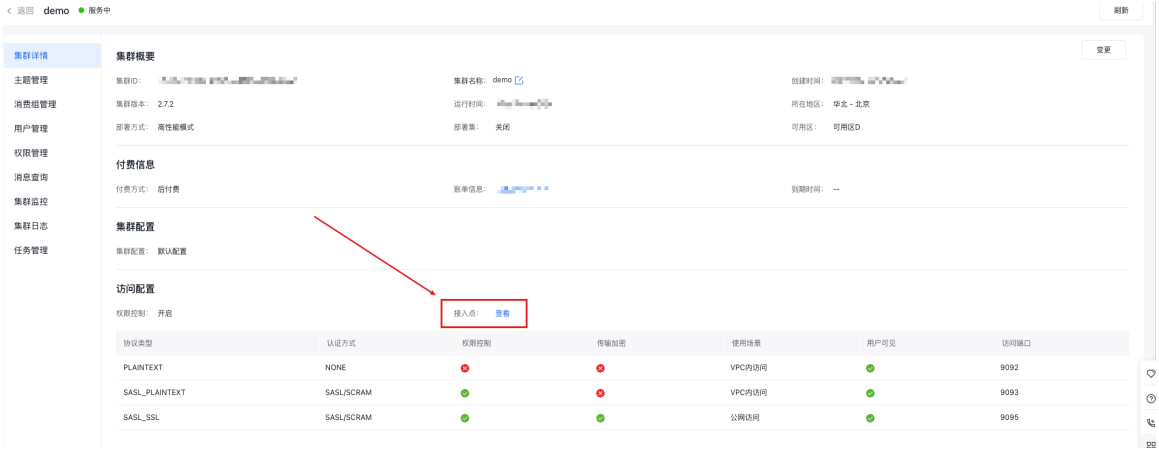
前提条件

- 步骤一：创建Kafka集群
- 步骤二：创建主题
- 步骤三：配置权限认证
- 安装JDK1.8及以上版本：JDK下载地址。

具体操作步骤

1. 下载Kafka 2.7.2安装包，并进行解压：kafka_2.13-2.7.2.tgz，其中解压后的bin目录为Kafka提供的可执行脚本。

2. 在**集群详情**中，点击**接入点**查看**SSL协议**所使用的访问地址和端口。



3. 下载证书：如何下载证书？。

4. 创建kafka.properties配置文件，也可以直接使用证书文件中的client_ssl.properties，内容格式如下：

```
security.protocol=SSL
###### 第3步下载的SSL证书文件路径
ssl.truststore.location=client.truststore.jks
###### SSL证书的密码，固定为bms@kafka
ssl.truststore.password=bms@kafka
###### 第3步下载的SSL证书文件路径
ssl.keystore.location=client.keystore.jks
ssl.keystore.password={password}
###### 固定为空
ssl.endpoint.identification.algorithm=
```

5. 进入Kafka安装包解压后的bin目录下，使用Kafka脚本访问集群。

- 查看Topic列表，需要使用--command-config配置项来指定第5步中创建的kafka.properties文件，需要设置完整的文件路径：

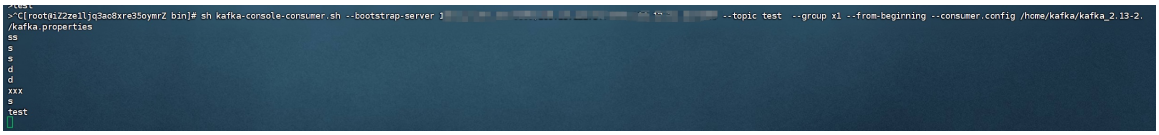
```
kafka-topics.sh --bootstrap-server <SSL接入点地址> --list --command-config /***/kafka.properties
```

- 生产者发送消息，输入命令后会出现">"的标识，此时输入内容并且按下回车即可发送消息：

```
kafka-console-producer.sh --bootstrap-server <SSL接入点地址> --topic <Topic名称> --producer.config /***/kafka.properties
```

- 消费者消费消息：

```
kafka-console-consumer.sh --bootstrap-server <SSL接入点地址> --topic <Topic名称> --group <消费组ID> --from-beginning --consumer.config /***/kafka.properties
```



操作指南

集群管理

查看集群信息

操作场景

当您创建完集群后，您可以在集群列表中查看已经创建的集群信息。

注意事项

- 如果是新创建的集群，需要等待集群状态变更为 **服务中** 时进行查看，否则数据信息不完整

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，可以看到当前您已经创建的集群列表，集群列表默认按照集群的**创建时间**倒序排列。
2. 如果集群数量较多，您可以通过 “**集群名称**”、“**集群ID**”、“**标签** 进行快速搜索。
3. 点击具体的集群，默认进入 **集群详情** 页面，能够看到集群的相关信息。
4. 集群详情页面目前共分为**集群概要**、**付费信息**、**集群配置**、**访问配置**、**网络安全**、**存储信息**以及**节点详情**部分。同时可以在**集群详情**中进行修改集群名称、查看账单信息、查看集群接入点、重启节点、查看Controller信息、查看节点日志等操作。

通过API查看集群详情

您可以通过 查询集群详情 查看指定集群的详情信息。

您可以通过 查询集群列表 查看指定集群的列表信息。

查看集群接入点

操作场景

获取Kafka集群的接入点地址，用于客户端连接Kafka集群。

注意事项

- 根据不同网络环境选择不同的接入点使用。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，进入想要查看的集群详情中。
2. 在 **集群详情** 中的 **访问配置** 中，单击 **接入点** 的查看按钮。
3. 单击 **查看** 后，会显示每一个协议对应的连接地址。

通过API查看接入点信息

您可以通过 查看集群接入点 查看集群的接入点信息。

变更访问配置

操作场景

用于创建集群时没有开启SASL/SCRAM身份认证，后续在使用过程中根据业务需求需要开启SASL/SCRAM。或者创建集群时开启SASL/SCRAM身份认证，但是业务需求需要关闭身份认证。

前提条件

集群必须处于 **服务中** 状态。

注意事项

变更访问配置会产生节点的重启操作。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，并选择具体的集群。
2. 如果集群当前没有开启SASL/SCRAM身份认证，此时单击左侧的用户管理或者权限管理能够看到访问配置变更的提示页面。
3. 或者在 **集群详情** 页面中单击 **变更** 按钮，进入集群变更页面。
4. 在变更页面中的**访问配置**一栏根据需求选择对应的认证方式以及权限控制，确认无误后点击**下一步**，并且单击**提交订单**发起变更操作。

重启节点

操作场景

用于重启集群中的节点。

前提条件

- 集群必须处于 **服务中** 状态才能重启节点。

注意事项

- 重启节点最长需要10分钟时间。 在此期间，Kafka集群可用于生产和消费数据，但在重启完成之前，您将无法执行其他集群操作。
- **重启节点会造成客户端与所重启节点短暂的断开连接**。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，并选择具体的集群。
2. 在 **集群详情** 中，找到 **节点详情**，可以看到节点个数以及 **重启** 按钮，如果当前集群不满足前提条件，则 **重启** 按钮为置灰并且不可点击状态。
3. 选择需要重启的节点，点击 **重启** 按钮，弹出对应的提示，确认无误后点击 **确定**。
4. 节点重启后，集群状态会变为 **重启中**，并且会在 **任务管理** 处生成一条 **重启节点** 类型的任务，可以通过查看任务详情来查看节点重启的进度。

删除集群

操作场景

当您不再需要使用某个集群时，可以选择对集群进行删除操作。

前提条件

- 如果集群的付费方式是 **后付费**，集群只有处于 **服务中** 的状态时才能够进行删除集群操作。
- 如果集群的付费方式是 **预付费**，集群只有处于 **集群部署失败** 时才能够进行删除操作。

注意事项

- 集群删除后，**底层所有资源均会释放，并且无法恢复，请谨慎操作**。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。

2. 选择需要删除的集群，在右侧单击 **删除**。
3. 单击 **删除** 后，系统会弹出一个确认框，提示您将要删除的集群名称，如果您确认删除该集群，则点击 **确定** 按钮。
4. 单击 **确定** 后，弹出安全验证框，单击 **发送验证码**，系统将验证码发送至您绑定的手机，请填写收到的验证码。**注意，如果本次登录后完成过安全验证，则不会再次弹出安全验证，而是直接删除集群。**
5. 单击 **确定** ，会进行集群删除操作。

查看Controller信息

操作场景

用于查看Kafka集群中Controller节点的变更情况。

注意事项

- 目前最多保存集群中50条Controller节点历史变化信息。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，并选择具体的集群。
2. 在 **集群详情** 中，找到 **节点详情**，可以看到当前集群Controller节点的ID。
3. 单击 Controller 节点ID后面的 **查看** 按钮，可以查看集群中Controller节点的变化历史信息。

集群变更

操作场景

用于对已有集群进行节点扩容、节点规格变更、磁盘扩容等变更操作。

前提条件

- 集群必须处于**服务中**状态。

注意事项

- 每次进行集群变更仅可变更 **一项** 内容。

操作步骤

1. 登录消息服务 for Kafka控制台进入集群列表页面。
2. 可以在**集群列表**中点击集群右侧的“**变更**”按钮，或者在**集群详情**页面右上角点击的“**变更**”按钮进入集群变更页面。
3. 进入集群变更页面后，可以**选择一个变更项**发起集群变更，目前支持的变更项如下所示：

变更项	变更内容	重启集群
集群配置	将目前集群中使用的配置替换为目标配置	可能（包含静态参数时会触发重启）
节点规格	将集群中的节点规格变更为目标规格	是
单区节点数	增加每个可用区中的节点数量	否
单节点容量	增加每个节点中磁盘容量	否
磁盘水位处理	调整集群中使用的磁盘水位处理策略	可能（策略变更为自动删除，自动删除上线前创建且之后未重启过的集群需要重启一次）
安全组	调整集群中使用的安全组	否
公网访问	开启或者关闭集群的公网访问	是
公网带宽	调整集群中使用的公网带宽	否
访问配置	对集群中的认证方式或者权限控制进行变更	是

通过API变更集群 您可以通过 变更节点机型 等相关集群变更接口来进行集群的变更操作。

磁盘水位处理

概述

当集群时开启了磁盘水位处理，集群会根据所选择的磁盘阈值策略进行自动调整。

自动删除

- 创建集群或者变更集群时，开启磁盘水位处理后磁盘阈值策略可以选择**自动删除**。
- 当集群节点中的磁盘使用率达到设定的阈值时，集群会根据所设置的参数开始自动清理日志，处理步骤如下：

1. 计算当前节点中磁盘清理到所设置的磁盘容量阈值需要删除的数据量。
2. 获取当前节点中磁盘上各个topic partition实际占用比例。
3. 按照topic partition占用比例计算各topic partition需要删除的数据量并进行删除，当前topic partition占比越多，需要删除的数据就越多。

集群启停

操作场景

用于启动或者停止集群服务。

前提条件

- 停止集群时，集群必须处于 **服务中** 状态
- 启动集群时，集群必须处于 **已停服且未欠费** 状态

注意事项

- 集群停止并不会释放相关资源，在集群停止期间不可访问集群，但资源的持续占用依旧**会产生相关的费用**。
- 集群停止后，**客户端将无法连接集群**，请谨慎操作。

操作步骤

停止集群

1. 登录消息服务 for Kafka控制台进入集群列表页面。
2. 可以通过集群列表中右侧的“**停止**”按钮或者集群详情中右上角的“**停止**”按钮发起集群停止服务操作，**请确认无误后再谨慎操作**。
3. 集群停止后，集群状态会变为**已停服**，并且会在**任务管理**处生成一条**停止集群服务**类型的任务，可以通过查看任务详情来查看**停止集群服务**的任务信息。

启动集群

1. 登录消息服务 for Kafka控制台进入集群列表页面。
2. 可以通过集群列表中右侧的“**启动**”按钮或者集群详情中右上角的“**启动**”按钮发起集群启动服务操作。
3. 集群启动后，集群状态会变为**服务中**，并且会在**任务管理**处生成一条**启动集群服务**类型的任务，可以通过查看任务详情来查看**启动集群服务**的任务信息。

通过API启停集群

您可以通过 启动集群、停止集群 来进行集群启停操作。

主题管理

创建主题

操作场景

用于在集群中创建主题（Topic）。

前提条件

- 集群必须处于 **服务中** 状态。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，点击需要操作的集群。
2. 在 **主题管理** 中点击 **创建主题** 按钮，输入主题名称、分区数、副本数等信息。
3. 在创建主题时，也可以对主题设置一些高级配置项。

通过API创建主题

您可以通过 创建主题 来创建主题。

查看主题详情

操作场景

用于查看主题的基本信息以及配置信息。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，点击需要操作的集群。
2. 在 **主题管理** 中点击对应的主题名称进入主题详情页面。
3. 在 **主题详情** 页面可以查看主题的名称、分区数、分区副本数、副本数等基本信息，也可以查看当前主题所设置的配置项。

通过API查看主题详情

您可以通过 查询主题详情 来查看主题详情。

删除主题

操作场景

当某个主题不再进行使用时，可以删除对应的主题。

前提条件

- 集群必须处于 **服务中** 状态

注意事项

- 主题删除后**无法恢复**，请谨慎操作

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情界面。
2. 在侧边导航**主题管理** 中选择要删除的主题，单击右侧的 **删除**。会弹出提示框，当确认无误后，单击 **确定** 按钮进行删除。

通过API删除主题

您可以通过 删除主题 来删除指定的主题。

修改主题分区数

操作场景

用于根据业务情况修改主题的分区数。

前提条件

- 集群必须处于 **服务中** 状态。

注意事项

- 主题的分区数**只能增加，不能减少**。
- 主题的分区增加上限为500。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情。
2. 在侧边导航**主题管理** 中点击对应的主题名称进入**主题详情**页面，在基本信息的分区数中单击**修改**，输入对应的分区数并确认主题分区数修改成功。

通过API修改主题分区数

您可以通过 变更主题 来修改指定主题的分区数。

修改主题配置

操作场景

用于主题创建完成后，根据有业务需求对主题的配置进行调整。

前提条件

- 集群必须处于 **服务中** 状态。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情。
2. 在侧边导航**主题管理** 中单击对应的主题名称进入**主题详情**页面，单击**高级配置**右侧的**编辑**按钮，会弹出高级配置的编辑页面，在此处对配置项进行编辑操作，确认无误后点击 **确定** 按钮。

通过API修改主题配置

您可以通过 变更主题 来修改指定主题的配置。

主题重新分区

操作场景

用于将主题的分区重新分配到集群的节点中。分为自动生成方案和手动配置方案两种方式。

- 自动生成方案：用户只需要选择分区分配的目标节点即可，无需关注每个副本具体的所在节点
- 手动配置方案：用户可以细粒度的配置分区的每个副本所在节点

前提条件

- 集群必须处于 **服务中** 状态。

注意事项

1. 主题重分区会占用集群的网络资源，建议在业务流量低峰期执行。
2. 如果主题的数据过多，重分区的时间可能会比较长。
3. 主题重分区后，主题的元数据信息发生变化，会造成客户端部分请求失败。

自动生成方案

单个主题进行重分区

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，点击需要操作的集群。
2. 在 **主题管理** 页面中，单击需要进行分区操作的**主题名称**，进入主题详情页面并选择**分区信息**。
3. 单击**重新分区**按钮，选择**自动生成方案**并且选择目标节点，根据需求填写副本数、分批执行大小、流量限额、执行时间。
4. 进行重新分区后，在任务管理中会生成一条主题调整的任务记录，可以查看任务的执行进度以及分区前后的配置情况。
5. 如果是重分区时指定了执行时间，在任务管理处可以进行**提前启动**或者**取消任务**。当主题重分区开始后，还可以进行**暂停/恢复**操作。

多个主题批量重分区

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群名称，进入集群详情。
2. 在侧边导航**主题管理** 页面中，单击**重新分区**按钮，可以选择多个需要重新分区的主题，填写分区的目标节点等参数。
3. 单击确定后，在任务管理处会生成一条主题调整的任务记录，可以查看进度并进行操作。

手动配置方案

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。单击需要操作的集群名称，进入集群详情界面。
2. 在侧边导航**主题管理** 页面中，单击需要进行分区操作的**主题名称**，进入主题详情页面并选择**分区信息**。
3. 单击**重新分区**按钮，选择**手动配置方案**并且选择需要重新分区的分区，手动调整每个分区副本在集群中的节点位置，根据需求设定每次执行的分区个数、流量限额以及执行时间。

4. 单击确定后，在任务管理处会生成一条主题调整的任务记录，可以查看进度并进行操作。

查看主题分区详情

操作场景

用于查看主题每个分区的具体情况。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，点击需要操作的集群。
2. 在 **主题管理** 中点击对应的主题名称进入**主题详情**页面，选择 **分区信息** 可以查看主题中所有分区的Leader副本所在节点、分区副本数信息、副本分布节点情况、ISR列表、最小位点、最大位点以及对应分区的最后写入时间。

通过API查看分区详情

您可以通过 查询主题分区详情 来查看主题分区详情。

查看主题订阅关系

操作场景

用于查看当前主题被哪些消费组所订阅。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情界面。
2. 在侧边导航**主题管理** 中点击对应的主题名称进入**主题详情**页面，选择 **订阅关系** 可以查看订阅当前主题的消费组列表。
3. 单击具体的消费组的**查看详情**按钮，可以展示当前消费组针对主题的每个分区的具体消费情况。

通过API查询主题订阅详情

您可以通过 查询主题订阅详情 来查看主题订阅详情。

消费组管理

查看消费组订阅信息

操作场景

用于查看消费组订阅的主题信息以及消费详情。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。单击需要操作的集群进入集群详情。
2. 在侧边导航单击**消费组管理** 进入页面能看到集群中的消费组列表。
3. 单击具体的消费组名称可以查看当前消费组所订阅的主题列表。
4. 单击 **查看详情** 按钮可以查看当前消费组在对应主题中的具体消费情况。

删除消费组

操作场景

用于删除集群中不再使用的消费组。

前提条件

- 集群必须处于 **服务中** 状态。
- 删除消费组时，需要确保该消费组内没有消费者正在运行，否则会删除失败。

注意事项

- 删除消费组后，该消费组所有的消费信息会被删除并且无法恢复，请谨慎操作。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。单击需要操作的集群名称，进入集群详情界面。
2. 在侧边导航选择**消费组管理**进入消费组列表页面，在需要删除的消费组右侧单击**删除**按钮，确认无误后单击**确定**进行删除。

通过API删除消费组

您可以通过 删除消费组 来删除消费组。

消费组重置位点

操作场景

用于改变某个消费组的消费位点。

前提条件

- 集群必须处于 **服务中** 状态。
- 消费组重置位点时，需要确保该消费组内没有消费者正在运行，否则会操作失败。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情界面。
2. 在**消费组管理**页面，在需要重置位带你的消费组右侧单击 **重置位点** 按钮，填写需要重置的主题和分区，并且选择重置策略进行重置。

通过API重置消费组位点

您可以通过 重置消费组位点 来重置消费组位点。

用户管理

创建用户

操作场景

当集群开启SASL/SCRAM身份认证时，使用SASL身份认证的协议访问集群时，需要通过用户身份认证时才能访问集群。

前提条件

- 集群必须处于 **服务中** 状态。
- 集群需要开启SASL/SCRAM身份认证。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情界面。
2. 在**用户管理**中单击**创建用户**按钮，填写配置项进行创建。

表一 创建用户配置项说明

配置项	说明
用户名	输入用户名，英文字母开头，4～16位字符，只能包含英文字母，数字，下划线(_)。
密码	输入密码，8～32位字符，英文字母、数字和符号必须同时存在，符号仅限!@#$%^*()。
确认密码	再次输入密码，两次需保持一致。
认证机制	支持多选。SCRAM-SHA-256、SCRAM-SHA-512。

3. 创建完成后单击确认，创建成功的用户以列表形式展示。

通过API创建用户

您可以通过 创建用户 来创建用户。

删除用户

操作场景

当集群中某个用户不再使用时，可以删除该用户，以避免用户身份被误使用。

前提条件

- 集群必须处于 **服务中** 状态。
- 集群需要开启SASL/SCRAM身份认证。

注意事项

- 用户删除后无法恢复，请谨慎操作。
- 用户删除后，使用该用户的客户端将无法通过身份认证。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情界面。
2. 在用户列表中单击需要删除的用户右侧的**删除**按钮，确认无误后单击**确定**进行删除。

通过API删除用户

您可以通过 删除用户 来删除指定的用户。

重置用户密码

操作场景

用于修改用户对应的密码信息。

前提条件

- 集群必须处于 **服务中** 状态。
- 集群需要开启SASL/SCRAM身份认证。

注意事项

- 重置密码后，使用旧密码的客户端将无法通过身份认证。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群，进入集群详情。

2. 在用户列表中单击对应的用户的 **重置密码** 按钮，输入新的密码信息后单击确认。

通过API重置用户密码

您可以通过 重置用户密码 来重置指定用户的密码。

权限管理

创建权限

操作场景

集群开启权限控制后，并且使用SASL/SCRAM身份认证协议访问集群时，需要给用户设置资源的权限才能够进行访问资源。

前提条件

- 集群需要开启 **权限控制** 功能。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情。
2. 在侧边导航**权限管理**中单击**创建权限**按钮，确定需要授权的用户、授权的资源类型、资源的匹配模式、匹配内容以及操作类型进行创建权限。

通过API创建权限

您可以通过 创建权限 来创建权限。

删除权限

操作场景

用于删除某个不再使用的权限。

前提条件

- 集群需要开启 **权限控制** 功能。

注意事项

- 权限信息删除后不可恢复，请谨慎操作。
- 权限删除后，使用该权限的客户端将无法通过权限认证，会无法访问对应的资源。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。单击需要操作的集群名称，进入集群详情。
2. 在 **权限管理** 中能够看到集群中此时配置的权限信息，支持按照资源类型、资源名以及用户名进行筛选。
3. 找到需要删除的权限，单击对应权限右侧的 **删除** 按钮，确认无误后单击 **确定** 按钮完成删除。

通过API删除权限

您可以通过 删除权限 来删除权限。

监控报警

集群监控

概述

集群监控提供了集群监控、节点监控、主题监控以及消费监控四种维度的监控信息。

集群监控

1. 登录 消息服务 for Kafka控制台进入集群列表页面。单击需要操作的集群名称，进入集群详情界面。
2. 在左侧导航选择 **集群监控** ，可以看到集群中的监控指标展示。
3. 集群监控默认展示生产消息速率、生产消息流量、消费消息流量、主题总数等7项监控指标。如果需要查看更多的监控指标，点击右上角的 **指标筛选** 按钮进行指标的选择。

节点监控

节点监控用于展示集群中每个节点上的监控指标，分为**服务监控**和**主机监控**。

服务监控

服务监控主要是各个节点中Kafka服务相关的监控指标。比如每个节点的生产消息速率、生产消息流量等指标。

您可以在右上角的 **指标筛选** 中选择更多的监控指标。

主机监控

主机监控主要是各个节点本身的监控指标，比如CPU使用率、内存使用率等指标。

您可以在右上角的 **指标筛选** 中选择更多的监控指标。

主题监控

主题监控用于展示集群中指定主题相关的监控指标，分为**基础监控**、**主题分节点**、**节点分主题**三个维度。

基础监控

基础监控主要是用于展示主题的整体监控情况，比如主题的生产消息速率、生产消息流量等。

您可以在右上角的 **指标筛选** 中选择更多的监控指标。

主题分节点

主题分节点主要用于展示主题在指定节点上的监控情况。

您可以在右上角的 **指标筛选** 中选择更多的监控指标。

节点分主题

节点分主题主要用于展示每个节点上的主题监控情况。

您可以在右上角的 **指标筛选** 中选择更多的监控指标。

消费组监控

消费组监控用于展示集群中指定消费组相关的监控指标，分为**基础监控**、**消费组分主题**、**消费组分分区**三个维度。

基础监控

基础监控主要用于展示消费组的整体监控情况。比如消费组的消息滞后量、当前消费Offset等。

可以在右上角的 **指标筛选** 中选择更多的监控指标。

消费组分主题

消费组分主题主要用于展示消费组针对指定主题的监控情况。

可以在右上角的 **指标筛选** 中选择更多的监控指标。

消费组分分区

消费组分分区主要用于展示消费组针对主题的特定分区的监控情况。

可以在右上角的 **指标筛选** 中选择更多的监控指标。

报警策略配置

操作场景

用于用户在云监控BCM中针对集群中的监控指标配置报警策略，当监控指标达到报警规则时触发报警。

操作步骤

1. 登录 云监控BCM控制台 ，在左侧的云产品监控中选择 **消息服务 for Kafka** 产品。
2. 单击集群名称进入监控的详情页面，在这里能够查看监控的基本信息以及报警历史，在 **报警策略** 中可以查看已经配置的策略信息，并且可以选择不同维度查看策略或者添加策略。
3. 单击 **添加策略** 按钮进入 **创建报警策略** 页面，在该页面中配置具体的监控指标以及报警策略。

事件策略配置

操作场景

用户可以在云监控BCM中创建事件报警策略，当集群产生对应的事件后，用户可以接收到对应的报警信息提示。

操作步骤

1. 登录 云监控BCM控制台 ，选择左侧的**事件监控**进行添加报警策略。
2. 创建报警策略时，产品类型选择消息服务 for Kafka专享版，然后选择需要关注的**事件级别**以及**事件名称**，当对应的事件产生后，会将对应的事件信息通过所配置的通知模版发送到对应的用户。

事件类型

目前消息服务 for Kafka专享版提供的 **故障** 级别的 **事件名称** 如下所示：

事件名称
集群停止服务成功

目前消息服务 for Kafka专享版提供的 **警告** 级别的 **事件名称** 如下所示，一般为**集群操作失败**时会触发警告级别的事件：

事件名称
集群创建失败
集群磁盘容量变更失败
集群节点数量扩容失败
集群节点数量缩容失败
集群节点规格变更失败
集群主题重分区失败
集群重启失败
集群节点重启失败
集群访问配置更改失败
集群公网带宽更改失败
集群公网开启失败
集群公网关闭失败
集群停止服务失败
集群恢复服务失败
更新磁盘阈值策略失败
更新kafka配置失败

目前消息服务 for Kafka专享版提供的 **通知** 级别的 **事件名称** 如下所示，一般为**集群操作开始**或者**操作成功**时会触发通知级别的事件：

事件名称
集群创建开始
集群创建成功
集群释放开始
集群释放成功
集群磁盘容量变更开始
集群磁盘容量变更成功
集群节点数量扩容开始
集群节点数量扩容成功
集群节点数量缩容开始
集群节点数量缩容成功
集群节点规格变更开始
集群节点规格变更成功
集群停止服务开始
集群恢复服务开始
集群恢复服务成功
集群主题重分区开始
集群主题重分区成功
集群重启开始
集群重启成功
集群节点重启开始
集群节点重启成功
集群访问配置更改开始
集群访问配置更改成功
集群公网带宽更改开始
集群公网带宽更改成功
集群公网开启开始
集群公网开启成功
集群公网关闭开始
集群公网关闭成功
更新磁盘阈值策略开始
更新磁盘阈值策略成功
更新kafka配置开始
更新kafka配置成功

任务管理

任务类型介绍

概述

在集群中进行的变更、重启等操作会在任务管理中产生一条任务记录。目前集群中的任务类型分为：

任务类型名称	说明
增加节点数量	当用户通过变更操作新增集群的节点时，会产生一条增加节点数量的任务记录。
减少节点数量	当用户通过变更操作减少集群的节点时，会产生一条减少节点数量的任务记录。
升级节点规格	当用户升级节点规格时，会产生一条升级节点的任务记录。
降低节点规格	当用户降低节点规格时，会产生一条降低节点的任务记录。
变更节点规格	当用户通过变更操作修改集群节点的机型时，会产生一条变更节点规格的任务记录。
扩容节点磁盘	当用户通过变更操作增加集群的磁盘容量时，会产生一条扩容磁盘容量的任务记录。
主题重新分区	当用户进行主题重分区操作时，会产生一条主题重新分区的任务记录。操作步骤详见主题重新分区
变更集群访问	当用户修改集群的访问配置时，比如开启/关闭权限控制，会产生一条变更集群访问的任务记录。
重启集群节点	当用户手动重启某个节点时，会产生一条重启集群节点的任务记录。
更新集群配置	当用户通过变更操作修改集群使用的配置时，会产生一条更新集群配置的任务记录。
升级集群版本	当用户升级集群版本时，会产生一条升级集群版本的任务记录。
启动集群服务	当用户启动已经停止的集群时，会产生一条启动集群服务的任务记录。
停止集群服务	当用户停止集群时，会产生一条停止集群服务的任务记录。
变更公网带宽	当用户调整公网的带宽大小时，会产生一条变更公网带宽的记录。
开启公网	当用户开启公网时，会产生一条开启公网的任务记录。
关闭公网	当用户关闭公网时，会产生一条关闭公网的任务记录。
变更容量策略	当用户通过变更操作调整磁盘水位处理策略时，会产生一条变更容量策略的任务记录。
更新集群安全组	当用户通过变更操作更新集群安全组时，会产生一条更新集群安全组的任务记录。
更新主题配置	当用户通过变更操作更新主题配置时，会产生一条更新主题配置的任务记录。
开启产品间转储	当用户通过变更操作打开产品间转储按钮时，会产生一条开启产品间转储任务记录。
关闭产品间转储	当用户通过变更操作关闭产品间转储按钮时，会产生一条关闭产品间转储任务记录。
重启集群	当用户通过变更操作重启集群时，会产生一条重启集群操作
变更集群	当用户通过变更操作变更集群时（例如计费变更），会产生一条变更集群任务记录。

查看任务详情

操作场景

当用户在集群中进行新增节点、扩容磁盘等操作时，会产生一条任务记录，一个任务可能会对应多个操作，查看任务的详情可以了解到任务的执行进度以及执行内容。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群名称进入集群详情界面。
2. 在侧边导航 **任务管理** 页面中可以查看集群所产生的任务记录。
3. 单击右侧的 **查看** 按钮可以进入任务详情页面，在任务详情中可以看到任务的基本信息、任务进度以及本次任务的执行内容。

通过API查看任务详情

您可以通过 查询任务详情 来查询任务详情。

集群日志

操作场景

用于查看集群中的Kafka节点日志信息。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情。
2. 在 **集群详情** 的 **节点详情**中，可以单击某个节点的右侧 **查看日志** 跳转集群日志详情。
3. 也可以选择左侧导航的 **集群日志** ，下拉检索特定的节点来查看对应的日志信息。
4. 单击对应的日志文件名称查看日志的具体信息。

消息查询

操作场景

用于在控制台中根据时间或者位点来查询主题中的消息。

注意事项

- 支持查询结果最多显示20条消息，且总大小不超过10MB。
- 每10秒可查询一次。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击需要操作的集群进入集群详情。
2. 在侧边导航单击**消息查询** ，进入消息查询页面选择所要查询的主题、查询方式、分区、起始位点/时间等信息进行查询，并且单击右上角的下载按钮可以将本次查询到的消息下载到本地。
3. 单击右侧的 **查看** 按钮，可以查看对应消息的具体信息，并且可以对当前消息进行复制和下载操作。

操作审计

操作场景

操作审计基于云审计 BCT服务，用于记录用户的操作。可以从操作审计中查看该账户下的用户所进行的操作。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面，单击左侧导航的 **操作审计** 。
2. 单击**操作审计**会跳转到云审计页面，可以在此页面查看用户操作过的时间名称以及资源类型、事件时间等信息。也可以根据用户名、事件名称、资源类型进行筛选。

标签管理

概述

百度智能云提供标签管理功能，支持按照各种标准（如用途、所有者或项目）提供跨服务、跨区域对资源进行添加标签，从而快速分类和管理资源。消息服务 for Kafka支持对集群进行添加标签。

使用说明及限制

使用标签前，您需要了解：

- 标签：每个标签由键和值两部分组成，标签（键+值）唯一。
- 单个和批量：支持为单个资源设置标签，也可以批量为云资源创建标签。
- 搜索：支持按照标签（键和值）进行资源搜索。

- 跨产品标记：不同产品可以添加同一个标签。
- 标签规格：每个用户最多可以创建200个标签。

添加标签

1. 登录 消息服务 for Kafka控制台 ，用户在创建集群的时候可以添加标签信息，默认会填写一个键为 “KAFKA-Cluster”，值为空值的标签。
2. 如果创建集群时没有添加对应的标签，在集群列表处也可以针对集群进行标签的编辑操作，勾选需要编辑标签的集群，单击**编辑标签**按钮。
3. 针对单个资源进行编辑标签时，可以删除已有的标签、新增标签以及对现有标签进行修改。
4. 针对多个资源进行编辑标签时，只能新增标签。
5. 用户也可以在左侧导航栏中单击 **标签管理** ，进入标签管理页面进行标签的操作。

标签搜索

用户可以针对标签的键值来筛选集群资源。

资源账单

资源账单是记录每个资源的生命周期与消费相关的数据的功能，方便用户以资源为维度查看资源及产品的消费数据。

1. 登录 消息服务 for Kafka控制台 ，单击左侧导航栏的 **标签管理** ，进入标签列表页。
2. 勾选一个标签，操作栏单击 **查看账单**。
3. 进入资源账单页面，选择需要查看账单的月份，会显示当前标签资源的消费信息。还可以通过组合筛选条件执行标签搜索，按标签键和值进行选择其他标签的消费详情，并下载相关信息。

多用户访问控制

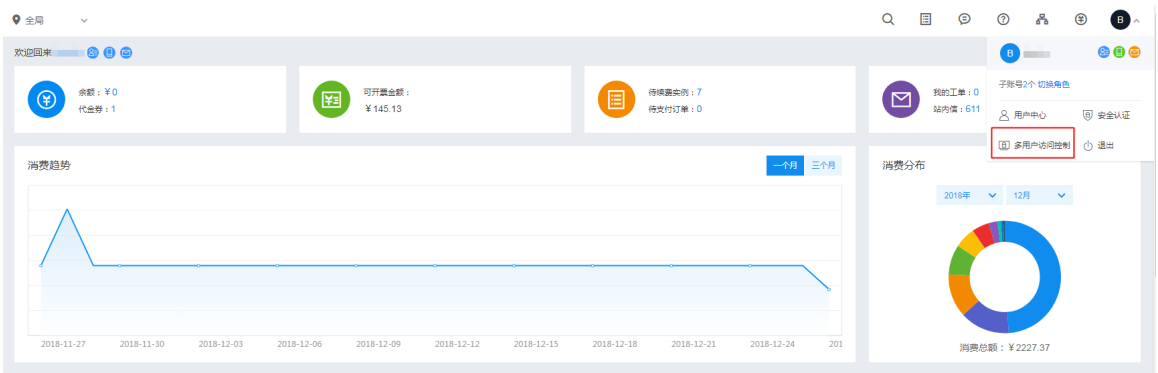
介绍

多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。 适用于下列使用场景：

- 中大型企业客户：对公司内多个员工授权管理；
- 偏技术型vendor或SAAS的平台商：对代理客户进行资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理。

创建用户

1.主账号用户登录后在控制台选择“多用户访问控制”进入用户管理页面。



2.在左侧导航栏点击“用户管理”，在“子用户管理列表”页，点击“新建用户”。

3.在弹出的“新建用户”对话框中，完成填写“用户名”和确认，返回“子用户管理列表”区可以查看到刚刚创建的子用户。

权限策略配置

权限策略有两种类型：系统权限策略和自定义权限策略，分别实现BMS的产品级和资源级权限控制。

- 系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。
- 自定义策略：由用户自己创建，更细化的管理资源的权限集，可以针对资源配置权限，更加灵活的满足账户对不同用户的差异化权限管理。 **系统策略** 系统策略是被用户经常使用的权限集合，已预先配置好不可被用户编辑。专享版Kafka提供了管理权限、运维权限和只读权限3种系统策略，权限范围详细如下：

策略名称	描述	包含权限
KAFKAFullControlPolicy	完全控制管理专享版百度消息服务(KAFKA)权限	支持专享版百度消息服务(KAFKA)的全部权限，包括创建删除集群，运维权限和读权限
KAFKAOperateAccessPolicy	运维操作专享版百度消息服务(KAFKA)权限	支持专享版百度消息服务(KAFKA)变更集群、创建删除主题、主题重分区、创建删除访问控制列表、删除消费组、创建删除用户、读权限等
KAFKAReadOnlyAccessPolicy	只读专享版百度消息服务(KAFKA)权限	支持专享版百度消息服务(KAFKA)查看集群列表、查看集群详情、查看主题列表、查看主题详情、查看主题分区、查看消费组列表等

自定义策略 自定义策略允许用户根据自己的需求灵活配置自己的权限策略，对选定的资源生效。专享版Kafka提供了Cluster、Topic、ConsumerGroup、User4种资源级自定义策略，权限范围详细如下：

权限名称	权限范围
KAFKAClusterWritePolicy	支持专享版百度消息服务(KAFKA)新增节点、扩容磁盘、变配节点、重启节点、创建主题、重新分区、修改配置等
KAFKATopicWritePolicy	支持专享版百度消息服务(KAFKA)变更主题，包括修改分区数、修改主题配置
KAFKAConsumerGroupWritePolicy	支持专享版百度消息服务(KAFKA)变更消费组的offset
KAFKAUserWritePolicy	支持专享版百度消息服务(KAFKA)变更用户

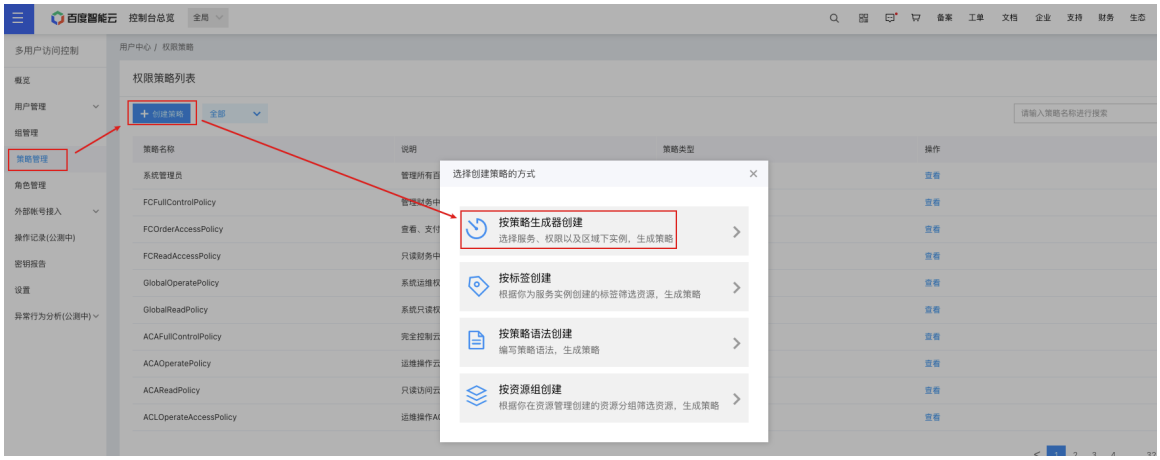
自定义策略生成方式有2种：策略生成器和编辑策略文件。

使用策略生成器配置权限

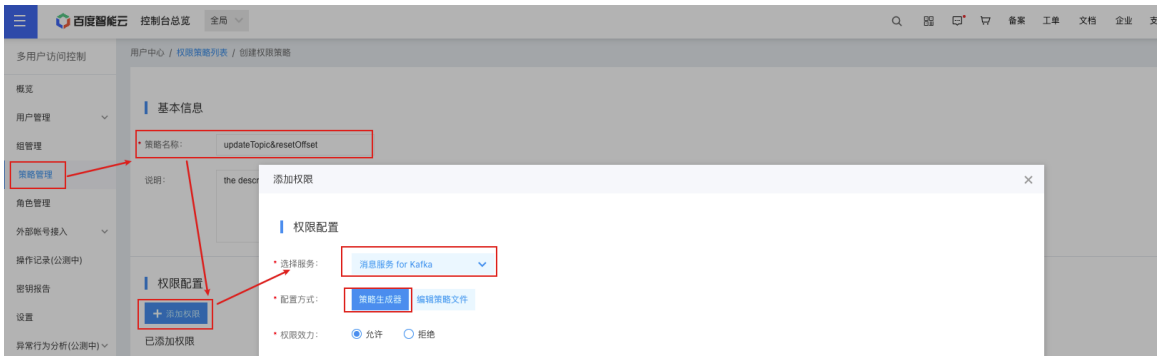
策略生成器，适合于需要控制资源访问的场景。使用策略生成器定义的策略，可以控制到某个具体要访问的资源，比如：Cluster的更改操作权限

操作步骤：

1.子用户进入“多用户访问控制”，通过左侧导航栏进入“策略管理”，点击“创建策略”，选择创建策略方式“按策略生成器创建”；



2.用户填写“策略名称”，并点击“添加权限”按钮，“选择服务”设置为“百度消息服务 KAFKA”，“配置方式”选择“策略生成器”，按照需求配置资源级权限，点击确定；



3.从策略管理的“自定义策略”列表中可以看到刚刚定义的策略。

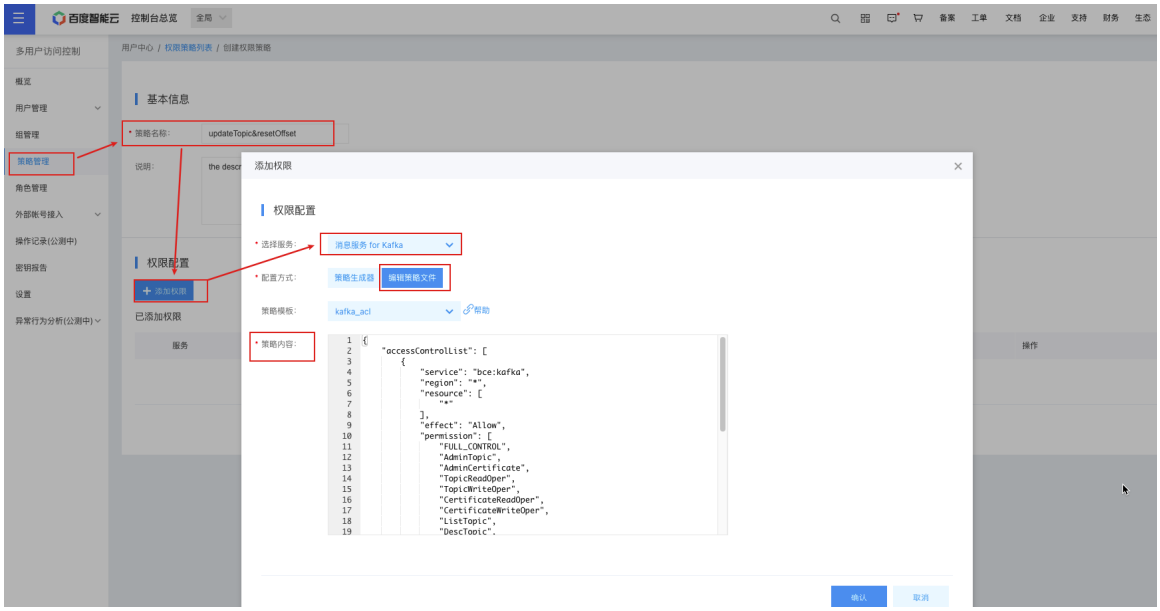
使用编辑策略文件配置权限

ACL策略配置文件，可以理解为配置权限策略规则的json字符串，无论是系统策略还是用户自定义策略，最终都会映射成为一个ACL的json串；使用ACL策略配置文件，用户可以非常灵活的定义权限策略，Kafka提供了一个公用的ACL配置字符串模板，用户可以参照模板配置自己的策略，编辑ACL权限策略的语法可参考文档策略语法。

操作步骤：

1.子用户进入“多用户访问控制”，通过左侧导航栏进入“策略管理”，点击“创建策略”，选择创建策略方式“按策略生成器创建”；

2.用户填写“策略名称”，并点击“添加权限”按钮，“选择服务”设置为“百度消息服务 KAFKA”，“配置方式”选择“编辑策略文件”，策略模板选择“kafka_acl”，按照需求配置ACL字符串，点击确定；



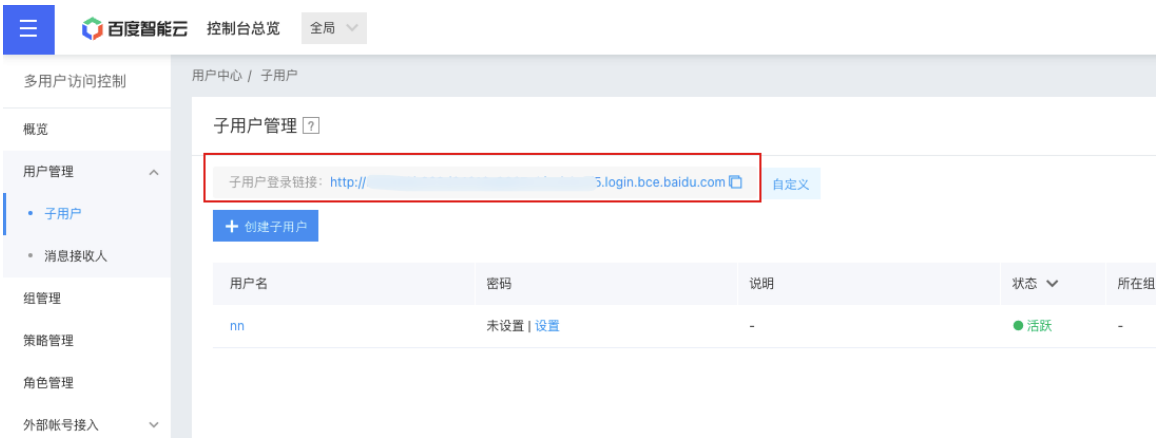
3.从策略管理的“自定义策略”列表中可以看到刚刚定义的策略。

用户授权

在“用户管理->子用户”对应子用户的“操作”列选择“编辑权限”，为用户选择系统权限或自定义策略进行授权。

子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。



其他详细操作参考：多用户访问控制。

集群配置管理

查看集群配置详情

操作场景

创建完集群配置后，可以在集群配置中查看某个版本的参数详情、查看版本的关联集群以及进行版本对比等操作。

操作步骤

1. 登录 消息服务 for Kafka控制台，侧边导航单击集群配置进入列表页面，可以看到当前您已经创建的集群配置列表。
2. 单击需要查看的配置，进入集群配置详情页面可以查看配置中的版本列表信息。
3. 单击需要查看的**版本号**，右侧会弹出对应版本中的参数信息。如果该版本正在被集群使用，则能够在**关联集群**中看到正在使用该版本的集群信息。关联集群支持按照集群名称和集群id进行筛选。
4. 单击版本号右侧的**版本对比**按钮，来对比两个版本中参数的差异点。

通过API查询集群配置详情

您可以通过 查询集群配置详情 、 查询集群配置版本列表、 查询集群配置版本详情 等接口来查询集群配置相关信息。

配置参数介绍

概述

创建集群时，可以使用默认配置或者使用自定义创建的集群配置，其中涉及到的参数信息如下所示。

- 更新模式分为静态模式和动态模式，当发起集群配置变更时，如果涉及到的变更参数包含了静态模式的参数，则会触发集群的重启操作，如果仅涉及动态模式的参数，则不会触发集群的重启操作。
- 参数如果是必选，则在创建集群配置时必须设置该参数。并且集群如果是使用默认配置，必选参数也会使用对应的默认值设置到集群中。

参数名称	描述	更新模式	默认值	是否必选
auto.create.topics.enable	是否允许自动创建 topic	静态模式	false	是
log.retention.hours	消息保留时长（小时）	静态模式	168	是
log.retention.bytes	单个分区日志保留大小（byte）	动态模式	-1	是

		式		
offsets.retention.minutes	消费组 offset 记录保留时长（分钟）	静态模式	10080	是
auto.leader.rebalance.enable	是否开启分区 leader 自动均衡	静态模式	true	是
num.partitions	topic 默认分区数	静态模式	3	是
default.replication.factor	topic 默认副本数	静态模式	2	是
delete.topic.enable	是否允许删除 topic	静态模式	true	是
log.cleanup.policy	日志清理策略	动态模式	delete	是
message.max.bytes	单条消息的最大值（byte）	动态模式	1048588	是
replica.fetch.max.bytes	副本同步时消息的最大大小（byte）	静态模式	1048576	是
log.message.timestamp.type	消息中的时间戳类型	动态模式	CreateTime	是
min.insync.replicas	当生产者将acks设置为all(或者-1)时，消息必须成功写入该配置的副本数时才能被视为写入成功	动态模式	1	是
zookeeper.session.timeout.ms	Zookeeper session超时时间（ms）	静态模式	18000	否
zookeeper.connection.timeout.ms	客户端等待与Zookeeper建立连接的最长时间（ms）	静态模式	18000	否
unclean.leader.election.enable	是否允许不在ISR中的副本作为最后手段来选举为Leader	动态模式	false	否
log.cleaner.delete.retention.ms	日志清理格式为compact时，压缩后日志删除的保留时长（ms）	动态模式	86400000	否
transaction.max.timeout.ms	事务允许的最大超时时间	静态模式	900000	否
queued.max.requests	数据平面允许的排队请求大小，超过后会阻塞网络线程	静态模式	500	否
offsets.commit.timeout.ms	offset延迟提交的时间，直到offset所有的副本都收到该提交，或者达到该值后超时（ms）	静态模式	5000	否
log.retention.check.interval.ms	日志清理器检查是否有日志符合删除条件的频率（ms）	静态模式	300000	否
log.segment.delete.delay.ms	日志文件被标记删除后，还需等待多久才会被真正的删除（ms）	动态模式	60000	否

创建集群配置

操作场景

当您需要对kafka集群中的参数进行调整时，可以创建一个集群配置，在创建集群时可以使用该配置来给集群设置指定的参数。

注意事项

- 每个账户可以创建30个集群配置，每个配置中最多生成100个版本。
- 当集群配置处于 **使用中** 状态时不可被删除。

操作步骤

1. 登录 消息服务 for Kafka控制台， 单击侧边导航集群配置进入列表页面，可以看到当前您已经创建的集群配置列表。
2. 单击 **创建配置** 按钮，输入配置名称和配置描述，按照需求调整参数的值。参数项分为必选参数和可选参数，必选参数在创建集群配置时不可取消，可以通过右上角的**参数筛选**按钮添加可选参数。
3. 确认完参数值后，单击确定按钮进行创建。创建成功后会在生成集群配置，并且在配置中生成第一个配置版本。
4. 配置版本生成后不可删除和修改，如果需要对参数进行调整，可以在集群配置中点击**新增版本**按钮来创建新的版本。单击**新增版本**后会自动展示上一个版本的参数信息并进行填充，可以针对上一个版本中需要修改的参数进行调整，调整完成后点击确定生成新的配置版本。
5. 最新版本以列表形式展示，单击操作项**版本对比**可以查看版本详细参数信息，修改过的参数行会重点显示。

通过API创建集群配置

您可以通过 创建集群配置、 新增集群配置版本 等接口来创建和操作集群配置。

删除集群配置

前提条件

在删除集群配置前，必须明确当前配置确实不再被业务所需要，且删除后不会影响到业务的正常运行。例如，一些版本相关的配置，如果主题仍在被使用，删除其相关配置可能导致消息生产或消费出现异常。

操作步骤

1. 登录 消息服务 for Kafka控制台 进入集群列表页面。
2. 侧边导航选择**集群配置**，进入配置列表。
3. 在需要删除的配置操作列单击**删除**按钮，系统出现提示界面确认无误后单击确定，集群配置删除成功。注意：删除后配置无法恢复，请您谨慎操作。

存储分析

操作场景

用于查看集群中的磁盘使用情况，可针对具体节点通过使用量排名、使用量大小以及使用量占比三个维度分析节点中磁盘的使用情况。

操作步骤

1. 登录消息服务 for Kafka控制台进入集群列表页面。选择具体的集群单击集群名称，进入集群详情。
2. 在左侧导航选择 **存储分析**，可以看到集群当前所有节点上的磁盘使用的相关信息。
3. 单击某节点右侧的 **分析按钮** 可以查看该节点分析详情。
4. 用户可以通过**使用Top、使用大小、使用占比**三种统计方式来查看当前节点中Topic的磁盘使用情况。

API参考

调用说明

API调用遵循HTTP协议，各Region采用不同的域名，具体域名见服务域名。数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

实名认证

使用消息服务 for Kafka API的用户需要实名认证，没有通过实名认证的可以前往百度开放云官网控制台中的安全认证下的实名认证中进行认证。没有通过实名认证的用户请求将会得到以下错误提示码：

错误码	错误描述	HTTP状态码	中文解释
QualifyNotPass	The User has not pass qualify.	403	账号没有通过实名认证

API认证机制

所有API的安全认证一律采用Access Key与请求签名机制。 Access Key由Access Key ID和Secret Access Key组成，均为字符串。 对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version是正整数。
- timestamp是生成签名时的UTC时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。（本API签名仅使用host和x-bce-date两个header）
- signature是256位签名的十六进制表示，由64个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。

鉴权认证机制的详细内容请参见鉴权认证。

通信协议

消息服务 for Kafka API支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

公共请求头

公共头部	描述
Authorization	包含Access Key与请求签名。具体请参考鉴权认证
Content-Type	application/json; charset=utf-8
x-bce-date	表示日期的字符串，符合API规范。具体请参考日期与时间规范

HTTP协议的标准头域不在这里列出。公共头域将在每个消息服务 for Kafka API中出现，是必需的头域。POST、PUT、DELETE等请求数据放在request body中。

公共响应头

公共头部	描述
Content-Type	application/json; charset=utf-8
x-bce-request-id	消息服务 for Kafka后端生成，并自动设置到响应头域中

其中，x-bce-request-id 使用UUID version4由消息服务 for Kafka服务生成。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体，如：POST /v{version}/cluster/{clusterId}
Query参数	URL中携带的请求参数，如：POST /v{version}/cluster/{clusterId}?marker={marker}
HEADER	通过HTTP头域传入,如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

响应结构说明

响应值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL参数	API版本号，当前值为2。	是

密码加密传输规范

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用AES 128位加密算法进行加密，用SK的前16位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

- byte[] bCiphertext= AES(明文,SK)
- String strHex = HexStr(bCiphertext)

日期与时间规范

日期与时间的表示有多种方式。为统一起见，除非是约定俗成或者有相应规范的，凡需要日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

1. 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日
2. 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
3. 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

服务域名

消息服务 for Kafka的服务域名为：

Region	Endpoint	Protocol
北京	kafka-api.bj.baidubce.com	HTTP and HTTPS
广州	kafka-api.gz.baidubce.com	HTTP and HTTPS
苏州	kafka-api.su.baidubce.com	HTTP and HTTPS
武汉	kafka-api.fwh.baidubce.com	HTTP and HTTPS
保定	kafka-api.bd.baidubce.com	HTTP and HTTPS
上海	kafka-api.fsh.baidubce.com	HTTP and HTTPS

说明：消息服务 for Kafka API支持HTTP和HTTPS两种调用方法。为了提升数据的安全性，建议通过HTTPS调用。

错误返回

当用户访问API出现错误时，会返回给用户相应的错误码和错误信息，便于定位问题，并做出适当的处理。请求发生错误时通过Response Body返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	表示具体错误类型
message	String	有关该错误的详细说明
requestId	String	本次请求的requestId

示例：

```
{
  "requestId" : "ae2225f7-1c2e-427a-a1ad-5413b762957d",
  "code" : "NoSuchKey",
  "message" : "The resource you requested does not exist"
}
```

公共错误码

下表列出了百度智能云API的公共错误码。

错误码	错误消息	HTTP状态码	描述
AccessDenied	Access denied.	403Forbidden	无权限访问对应的资源。
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided doesnot exist in our records.	403Forbidden	Access Key ID不存在。
InvalidHTTPAuthHeader	The Access Key ID you provided does notexist in our records.	400 BadRequest	Authorization头域格式错误。

InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 BadRequest	HTTP body格式错误。例如不符合指定的Encoding等。
InvalidURI	Could not parse the specified URI.	400 BadRequest	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配的问题，应定义更加具体的错误码，如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 BadRequest	JSON格式不合法。
InvalidVersion	The API version specified was invalid.	404 NotFound	URI的版本号不合法。
OptInRequired	A subscription for the service is required.	403Forbidden	没有开通对应的服务。
PreconditionFailed	The specified If-Match header doesn’tmatch the ETag header.	412PreconditionFailed	详见Etag。
RequestExpired	Request has expired. Timestamp date is <Data>.	400 BadRequest	请求超时。要改成x-bce-date。若请求中只有Date，需将Date转成datetime。
IdempotentParameterMismatch	The request uses the same client token asa previous, but non-identical request.	403Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check yourSecret Access Key and signing method. Consultthe service documentation for details.	400 BadRequest	Authorization头域中附带的签名和服务端验证不一致。

消息服务 for Kafka错误码

在每个API接口的**错误码**部分会列出消息服务 for Kafka产生的错误码。

集群管理接口

创建集群

该接口用于创建集群。

- 针对创建集群白名单中的用户。

请求结构

```
POST /v{version}/clusters HTTP/1.1
Host: kafka.api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
name	String	是	Request Body参数	集群名称
mode	String	是	Request Body参数	部署模式， HP、HA
type	String	是	Request Body参数	部署类型，PROVISIONED、SERVERLESS
provisioned	Provisioned	是	Request Body参数	provisioned 集群类型的参数，参见Provisioned
tags	List<Tag>	否	Request Body参数	标签列表

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群重名	400	集群重名
IMAGE_NOT_FOUND	未找到镜像	451	未找到镜像
RESOURCE_FAILED	资源达到上限	451	资源达到上限
STOCK_FAILED	库存不足	451	库存不足
RESOURCE_NOT_FOUND	资源已下架	451	资源已下架
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
POSt http://kafka-api.bj.baidubce.com/v2/clusters
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "name": "demo",
    "mode": "HP",
    "type": "PROVISIONED",
    "provisioned": {
      "kafkaVersion": "2.7.2",
      "nodeType": "kafka.g4.c2m8",
      "configMeta": {
         "configId": null,
         "revisionId": null,
         "context": {
            "log.retention.hours":"24"
         }
      },
      "numberOfBrokerNodes": 3
```

```
        "numberOfBrokerNodes": 3,
        "storageMeta": {
            "storageType": "ENHANCED_SSD_PL1",
            "storageSize": 100,
            "numberOfDisk": 1
        },
        "storagePolicyEnabled": true,
        "storagePolicy": {
            "type": "AUTO_DELETE",
            "autoDelete": {
                "diskUsedThresholdPercent": 75,
                "logMinRetentionMs": 3600000,
                "logMinRetentionBytes": 2147483648
            }
        },
        "deploySetEnabled": false,
        "billing": {
            "payment": "Prepaid",
            "timeLength":1,
            "timeUnit":"month",
            "autoRenewEnabled": true,
            "autoRenewTimeLength": 1,
            "autoRenewTimeUnit": "month",
            "couponIds":[
                "xxxxx"
            ],
            "isAutoPay": true,
        },
        "vpcId": "vpc-tf3xqatke54b",
        "subnetIds": [
            "sbn-12kruawmu30u"
        ],
        "publicIpEnabled": false,
        "publicIpBandwidth": 0,
        "intranetIpEnabled": false,
        "aclEnabled": true,
        "authentications": [
            {
                "mode": "NONE"
            }
        ]
    },
    "tags": [
        {
            "tagKey": "KAFKA-Cluster",
            "tagValue": ""
        }
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233"
}
```

释放集群

该接口用于释放集群

- 针对释放集群白名单中的用户。

请求结构

```
DELETE /v{version}/clusters/{clusterId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待删除的集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_CAN_NOT_DELETED	集群不可删除	451	集群不可删除
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
DELETE http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233"
}
```

查询集群列表

该接口用于查询用户在该区域中所有使用中的集群列表信息。

请求结构

```
GET /v{version}/clusters HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
marker	String	否	Query参数	批量获取列表的查询的起始位置，需要设置为集群ID
maxKeys	int	否	Query参数	每页包含的最大数量，最大数量不超过1000，缺省值为1000
clusterName	String	否	Query参数	集群名模糊匹配
state	String	否	Query参数	用于指定查询的集群状态，具体可选状态参见ClusterState。如果不传入该参数，则表示查询所有状态的集群。
mode	String	否	Query参数	集群部署模式，可选的值为HA（高可用），HP（高性能）
kafkaVersion	String	否	Query参数	集群的版本，当前取值：2.7.2
payment	String	否	Query参数	付费方式，可选的值为：Prepaid（预付费）和 Postpaid（后付费）
tagKey	String	否	Query参数	集群标签键，集群资源标签键为 KAFKA-Cluster
tagValue	String	否	Query参数	集群标签值

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置，与请求接口时传入的marker参数值保持一致
maxKeys	int	每页包含的最大数量，，最大数量不超过1000，缺省值为1000
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	下一页所需要传递的marker值，当isTruncated为false时，该值为null
clusters	List<Cluster>	集群信息列表，由 Cluster 组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "marker": null,
    "isTruncated": false,
    "nextMarker": null,
    "maxKeys": 1000,
    "clusters": [
        {
            "clusterId": "33a168bb70c0459787416077114ab233",
            "clusterSid": "kafka-o8uu8JKS",
            "name": "demo",
            "region": "bj"
            "type": "PROVISIONED",
            "mode":"HP",
            "state": "ACTIVE",
            "kafkaVersion": "2.7.2",
            "logicalZones": [
                "cn-fwh-a"
            ],
            "payment": "Postpaid",
            "aclEnabled": true,
            "publicIpEnabled": false,
            "intranetIpEnabled": false,
            "authenticationModes": [
                "NONE",
                "SASL_SCRAM"
            ],
            "tags": [
                {
                    "tagKey": "KAFKA-Cluster",
                    "tagValue": ""
                }
            ],
            "createTime": "2023-05-10T03:06:01Z",
            "expireTime": -1
        }
    ]
}
```

查询集群详情

该接口用于查询指定集群的详细信息。

请求结构

```
GET /v{version}/clusters/{clusterId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
cluster	ClusterDetail	返回的集群详细信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "cluster": {
        "clusterId": "33a168bb70c0459787416077114ab233",
        "clusterSid": "kafka-o8uu8JKS",
        "name": "demo",
        "region":"bj",
```

```
    "type": "PROVISIONED",
    "mode": "HP",
    "state": "ACTIVE",
    "provisioned": {
      "kafkaVersion": "2.7.2",
      "billing": {
        "payment": "Prepaid",
        "timeLength": 1,
        "timeUnit": "month",
        "expireTime":"2023-05-15T03:06:01Z",
        "renewEnabled": true,
        "renewTimeLength": 1,
        "renewTimeUnit": "month"
      },
      "vpc": {
        "vpcId":"vpc-tf3xqatke54b",
        "name":"默认私有网络",
        "cidr":"172.18.0.0/16"
      },
      "subnets": [
        {
          "subnetId": "sbn-12kruawmu30u",
          "name": "系统预定义子网",
          "subnetType": "BCC",
          "zone": "cn-fwh-a",
          "vpcId": "vpc-tf3xqatke54b",
          "cidr": "172.18.0.0/20"
        }
      ],
      "logicalZones": [
        "cn-fwh-a"
      ],
      "securityGroup": {
        "securityGroupId": "g-bqw18bis4cm4",
        "name": "KAFKA默认安全组",
        "vpcId": "vpc-tf3xqatke54b"
      },
      "publicIpEnabled": false,
      "publicIpBandwidth": 0,
      "intranetIpEnabled": false,
      "authentications": [
        {
          "mode": "NONE"
        },
        {
          "mode": "SASL_SCRAM"
        }
      ],
      "aclEnabled": true,
      "numberOfBrokerNodes": 3,
      "numberOfBrokerNodesPerZone": 3,
      "nodeType": "kafka.g4.c2m8",
      "storageMeta": {
        "storageType": "ENHANCED_SSD_PL1",
        "storageSize": 100,
        "numberOfDisk": 1
      },
      "configMeta": {
        "configId": null,
        "revisionId": null,
        "context": {
          "log.retention.hours":"24"
        }
```

```
            },
            "deploySetEnabled": false,
        },
        "tags": [
            {
                "tagKey": "KAFKA-Cluster",
                "tagValue": ""
            }
        ],
        "createTime": "2023-05-10T03:06:01Z"
    }
}
```

查询集群接入点

该接口用于查询集群的接入点信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/access-endpoints HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
accessEndpoints	List<AccessEndpoint>	集群的接入点信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/access-endpoints
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
   "accessEndpoints":[
      {
         "securityProtocol":"SASL_PLAINTEXT",
         "endpoints":"172.18.0.15:9093,172.18.0.13:9093,172.18.0.10:9093",
         "network": "VPC"
      },
      {
         "securityProtocol":"PLAINTEXT",
         "endpoints":"172.18.0.15:9092,172.18.0.13:9092,172.18.0.10:9092"
         "network": "VPC"
      }
   ]
}
```

查询集群节点列表

该接口用于查询指定集群的所有节点信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/nodes HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
marker	String	否	Query参数	批量获取列表的查询的起始位置，需要设置为节点ID
maxKeys	int	否	Query参数	每页包含的最大数量，最大数量不超过1000，缺省值为1000
state	String	否	Query参数	用于指定查询的节点服务状态，具体可选状态参见NodeServiceState。如果不传入该参数，则表示查询所有状态的节点。

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置，与请求接口时传入的marker参数值保持一致
maxKeys	int	每页包含的最大数量，，最大数量不超过1000，缺省值为1000
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	下一页所需要传递的marker值，当isTruncated为false时，该值为null
nodes	List<Node>	节点信息列表，由 Node 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/nodes
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "marker": null,
    "isTruncated": false,
    "nextMarker": null,
    "maxKeys": 1000,
    "nodes": [
        {
            "brokerId": 1,
            "host": "instance-5z0qr0qv-1",
            "nodeId": "i-PKXSIpnl",
            "state": "ALIVE",
            "publicIp": null,
            "internalIp": "172.18.0.13"
        },
        {
            "brokerId": 2,
            "host": "instance-5z0qr0qv-2",
            "nodeId": "i-Y0V6vafS",
            "state": "ALIVE",
            "publicIp": null,
            "internalIp": "172.18.0.10"
        },
        {
            "brokerId": 3,
            "host": "instance-5z0qr0qv-3",
            "nodeId": "i-d6iPVKhf",
            "state": "ALIVE",
            "publicIp": null,
            "internalIp": "172.18.0.15"
        }
    ]
}
```

启动集群

该接口用于启动集群。

请求结构

```
PUT /v{version}/clusters/{clusterId}/start HTTP/1.1
Host: kafka.api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_SUSPENDED	集群非停机状态	451	集群非停机状态
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/start
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

查询集群参数

该接口用于查询集群中目前使用的集群参数信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/configurations HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	Query参数	集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
context	List<ClusterConfigOption>	集群中使用的参数信息

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/6bc4775ffb9b4523833e54d802d53598/configurations
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "context": [
        {
            "name": "auto.create.topics.enable",
            "description": "是否允许自动创建 topic",
            "updateMode": "STATIC",
            "scope": [
                true,
                false
            ],
            "defaultValue": false,
            "currentValue": false,
            "type": "BOOLEAN",
            "unit": null,
            "category": "主题",
            "overrideMode": "REQUIRED"
        },
        {
            "name": "log.retention.hours",
            "description": "消息保留时长（小时）",
            "updateMode": "STATIC",
            "scope": [
                1,
                2147483646
            ],
            "defaultValue": 168,
            "currentValue": 133,
            "type": "INT",
            "unit": "hour",
            "category": "消息",
            "overrideMode": "REQUIRED"
        }
    ]
}
```

停止集群

该接口用于停止集群。

请求结构

```
PUT /v{version}/clusters/{clusterId}/stop HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/stop
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

任务管理接口

启动任务

该接口用于启动集群中的任务。

- 目前支持的任务类型只有：REASSIGN_PARTITION（主题调整）。

请求结构

```
PUT /v{version}/clusters/{clusterId}/jobs/{jobId}/start HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待启动任务的集群ID
jobId	String	是	URL参数	待启动的任务ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
jobId	String	启动成功的任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_PRE_UPDATING	集群非待变更状态	451	集群非待变更状态
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
JOB_STATE_NOT_NEW	非待执行任务	451	非待执行任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/jobs/c4636f6efb5a056b48a2abe
t
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "jobId": "c4636f6efb5a056b48a2abe97e6c3ee2"
}
```

取消任务

该接口用于取消集群中的任务。

- 目前支持的任务类型只有：REASSIGN_PARTITION（主题调整）。
- 目前只有当主题调整任务设置为自定义执行时间，并且任务没有开始执行才能够取消任务。

请求结构

```
PUT /v{version}/clusters/{clusterId}/jobs/{jobId}/cancel HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待取消任务的集群ID
jobId	String	是	URL参数	待取消的任务ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
jobId	String	取消成功的任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_PRE_UPDATING	集群非待变更状态	451	集群非待变更状态
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
JOB_STATE_NOT_NEW	非待执行任务	451	非待执行任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/jobs/7a866624b5f1954ef84156
l
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "jobId": "7a866624b5f1954ef841569b4e3de5d1"
}
```

暂停任务

该接口用于暂停集群中已经启动的任务。

- 目前支持的任务类型只有：REASSIGN_PARTITION（主题调整）。
- 只有任务状态为RUNNING（执行中）的情况下才能够暂停任务。

请求结构

```
PUT /v{version}/clusters/{clusterId}/jobs/{jobId}/suspend HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待暂停任务的集群ID
jobId	String	是	URL参数	待暂停的任务ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
jobId	String	暂停成功的任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_UPDATING	集群非变更中状态	451	集群非变更中状态
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
JOB_STATE_NOT_RUNNING	非运行任务	451	非运行任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/jobs/7a866624b5f1954ef84156
d
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "jobId": "7a866624b5f1954ef841569b4e3de5d1"
}
```

恢复任务

该接口用于恢复集群中已经暂停的任务。

- 目前支持的任务类型只有：REASSIGN_PARTITION（主题调整）。
- 只有任务状态为SUSPENDED（暂停）时才能恢复任务。

请求结构

```
PUT /v{version}/clusters/{clusterId}/jobs/{jobId}/resume HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待恢复任务的集群ID
jobId	String	是	URL参数	待恢复的任务ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
jobId	String	恢复成功的任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_UPDATING	集群非变更中状态	451	集群非变更中状态
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
JOB_STATE_NOT_SUSPEND	非暂停任务	451	非暂停任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/jobs/7a866624b5f1954ef84156
e
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "jobId": "7a866624b5f1954ef841569b4e3de5d1"
}
```

查询任务列表

该接口用于查询集群中的任务列表。

请求结构

```
 GET /v{version}/clusters/{clusterId}/jobs HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
marker	String	否	Query参数	批量获取列表的查询的起始位置，需要设置为任务ID
maxKeys	int	否	Query参数	每页包含的最大数量，最大数量不超过1000，缺省值为1000
name	String	否	Query参数	待查询的任务类型名称，具体可选类型参见JobType

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置，与请求接口时传入的marker参数值保持一致
maxKeys	int	每页包含的最大数量
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	下一页所需要传递的marker值，当isTruncated为false时，该值为null
jobs	List<Job>	任务列表信息，由 Job 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/jobs
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "marker": null,
    "isTruncated": false,
    "nextMarker": null,
    "maxKeys": 1000,
    "jobs": [
        {
            "jobId":"xxxxxx",
            "name":"INCREASE_BROKER_COUNT",
            "status":"RUNNING",
            "operations": [
                {
                    "jobId":"xxxxxx",
                    "name":"INCREASE_BROKER_COUNT",
                    "status":"RUNNING",
                    "operationId":"xxxxxx",
                    "type":"INCREASE_BROKER_COUNT",
                    "state":"RUNNING",
                    "process":90,
                    "schedule":"EXECUTE",
                    "started":true,
                    "createTime":"2022-09-15T11:30:01Z",
                    "startTime":"2022-09-15T11:30:01Z",
                    "endTime":""
                }
            ]
        }
    ]
}
```

查询任务详情

该接口用于查询集群中任务的详细信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/jobs/{jobId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
jobId	String	是	URL参数	待查询的任务ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
job	JobDetail	返回的任务详细信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/jobs/c8c83ae37d598dcf5a8fc38
2
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "job": {
        "jobId": "7b1a77a30e88b2407dab9386f87d37aa",
        "type": "REASSIGN_PARTITION",
        "state": "FINISHED",
        "process": 100,
        "createTime": "2023-05-10T03:06:01Z",
        "startTime": "2023-05-10T03:06:01Z",
        "endTime": "2023-05-10T03:06:01Z",
        "groups": [
            {
                "groupName": "TOPIC_REASSIGN_PARTITION",
                "state": "FINISHED"
            }
        ],
        "sourceContext": "{\"assignments\":[{\"topic\":\"test\",\"partition\":1,\"replicas\":[3,4,1]},
{\"topic\":\"test\",\"partition\":0,\"replicas\":[2,3,4]},{\"topic\":\"test\",\"partition\":3,\"replicas\":[1,3,4]},
{\"topic\":\"test\",\"partition\":2,\"replicas\":[4,1,2]}]}",
        "targetContext": "{\"assignments\":[{\"topic\":\"test\",\"partition\":1,\"replicas\":[3,4,1],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":0,\"replicas\":[2,3,4],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":3,\"replicas\":[1,2,3],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":2,\"replicas\":[4,1,2],\"logDirs\":
[\"any\",\"any\",\"any\"]}]}"
    }
}
```

查询操作详情

该接口用于查询集群中任务某个操作的详细信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/jobs/{jobId}/operations/{operationId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
jobId	String	是	URL参数	待查询的任务ID
operationId	String	是	URL参数	待查询的操作ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
operation	OperationDetail	返回的操作详细信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
JOB_NOT_FOUND	找不到相关任务	451	找不到相关任务
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/jobs/c8c83ae37d598dcf5a8fc38
x
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "operation": {
        "jobId": "c8c83ae37d598dcf5a8fc3865d5ff6e2",
        "type": "INCREASE_BROKER_COUNT",
        "state":"RUNNING",
        "process":90,
        "schedule":"EXECUTE",
        "started":true,
        "createTime":"2022-09-15T11:30:01Z",
        "startTime":"2022-09-15T11:30:01Z",
        "endTime":"",
        "groups": [
            {
                "groupName": "TOPIC_REASSIGN_PARTITION",
                "state": "FINISHED"
            }
        ],
        "sourceContext": "{\"assignments\":[{\"topic\":\"test\",\"partition\":1,\"replicas\":[3,4,1]},
{\"topic\":\"test\",\"partition\":0,\"replicas\":[2,3,4]},{\"topic\":\"test\",\"partition\":3,\"replicas\":[1,3,4]},
{\"topic\":\"test\",\"partition\":2,\"replicas\":[4,1,2]}]}",
        "targetContext": "{\"assignments\":[{\"topic\":\"test\",\"partition\":1,\"replicas\":[3,4,1],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":0,\"replicas\":[2,3,4],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":3,\"replicas\":[1,2,3],\"logDirs\":
[\"any\",\"any\",\"any\"]},{\"topic\":\"test\",\"partition\":2,\"replicas\":[4,1,2],\"logDirs\":
[\"any\",\"any\",\"any\"]}]}"
    }
}
```

主题管理接口

创建主题

该接口用于在集群中创建主题（topic）。

- 集群处于服务中状态（ACTIVE）。

请求结构

```
POST /v{version}/clusters/{clusterId}/topics HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string

{
    "topicName": "topicName",
    "partitionNum": partitionNum,
    "replicationFactor": replicationFactor,
    "otherConfigs": {
        "message.timestamp.type":"message.timestamp.type",
        "cleanup.policy":"cleanup.policy",
        "min.insync.replicas":"min.insync.replicas",
        "retention.ms":"retention.ms",
        "segment.ms":"segment.ms",
        "max.message.bytes":"max.message.bytes"
    }
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待创建topic的集群ID
topicName	String	是	RequestBody参数	待创建topic的名称
partitionNum	int	是	RequestBody参数	待创建topic的分区数，取值范围为：1-500
replicationFactor	int	是	RequestBody参数	待创建topic的副本数，可取值为：2、3，副本数不能低于集群的可用区数目
otherConfigs	Map<String, String>	否	RequestBody参数	待创建topic的配置项，具体支持的配置项参见TopicConfig

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topicName	String	创建成功的topic名称

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
TOPIC_ALREADY_EXIST	主题已存在	451	主题已存在
TOPIC_REPLICATION_FACTOR_LIMIT_EXCEEDED_ERROR	Topic副本因子超过限制	451	Topic副本因子超过限制
TOPIC_PARTITION_ILLEGAL	Topic 分区数不合法	451	TOPIC_PARTITION_ILLEGAL
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
POST http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com

{
    "topicName": "test",
    "partitionNum": 4,
    "replicationFactor": 3,
    "otherConfigs": {
        "message.timestamp.type":"CreateTime",
        "cleanup.policy":"delete",
        "min.insync.replicas":"1",
        "retention.ms":"176",
        "segment.ms":"604800000",
        "max.message.bytes":"1048588"
    }
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topicName": "test"
}
```

变更主题

该接口用于变更集群中指定主题（topic）的配置信息。

- 集群处于服务中状态（ACTIVE）。

请求结构

```
PUT /v{version}/clusters/{clusterId}/topics/{topicName} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string

{
    "partitionNum": partitionNum,
    "otherConfigs": {
        "message.timestamp.type":"message.timestamp.type",
        "cleanup.policy":"cleanup.policy",
        "min.insync.replicas":"min.insync.replicas",
        "retention.ms":"retention.ms",
        "segment.ms":"segment.ms",
        "max.message.bytes":"max.message.bytes"
    }
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	是	URL参数	待变更topic的名称
partitionNum	int	否	RequestBody参数	topic的目标变更分区数，只支持增大分区数，不支持缩减分区数，最大分区数为500
otherConfigs	Map<String, String>	否	RequestBody参数	待变更topic的配置项，具体支持的配置项参见TopicConfig

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topicName	String	变更成功的topic名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
TOPIC_PARTITION_ILLEGAL	Topic 分区数不合法	451	TOPIC_PARTITION_ILLEGAL
TOPIC_REDUCE_PARTITION_IS_NOT_ALLOWED	不允许缩减 Topic 分区	451	TOPIC_REDUCE_PARTITION_IS_NOT_ALLOWED
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics/test
Authorization:
bce-auth-v1/ec7f277752364a61af9f3bcb10eb803e/2023-05-08T09:39:11Z/3600/host;x-bce-console-rpc-id;x-bce-date/b
b
Host: kafka-api.bd.baidubce.com
x-bce-console-rpc-id: cf57b4b1-a86f-4423-86fb-736c22068d84
x-bce-date: 2023-05-08T09:39:11Z

{
    "partitionNum": 4,
    "otherConfigs": {
        "message.timestamp.type":"CreateTime",
        "cleanup.policy":"delete",
        "min.insync.replicas":"1",
        "retention.ms":"604800000",
        "segment.ms":"604800000",
        "max.message.bytes":"1048588"
    }
}
```

返回示例

```
HTTP/1.1 200 OK
X-Bce-Request-Id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topicName": "test"
}
```

删除主题

该接口用于删除集群中的主题（topic）。

- 集群处于服务中状态（ACTIVE）。

请求结构

```
DELETE /v{version}/clusters/{clusterId}/topics/{topicName} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待删除topic的集群ID
topicName	String	是	URL参数	待删除topic的名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topicName	String	删除成功的topic名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
DELETE http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics/test
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topicName": "test"
}
```

查询主题列表

该接口用于查询指定集群的所有主题（topic）列表。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	否	Query参数	主题名称，用于模糊匹配

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topics	List<Topic>	主题信息列表，由 Topic 组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topics": [
        {
            "topicName": "test2",
            "createTime": "2023-05-10T03:06:25Z"
        },
        {
            "topicName": "test1",
            "createTime": "2023-05-10T03:06:25Z"
        },
        {
            "topicName": "test",
            "createTime": "2023-05-10T03:06:25Z"
        }
    ]
}
```

查询主题详情

该接口用于查询集群中指定主题（topic）的详细信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics/{topicName} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询topic的集群ID
topicName	String	是	URL参数	待查询topic的名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topic	TopicDetail	返回的主题详情信息

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics/test
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topic": {
        "topicName": "test",
        "partitionNum": 1,
        "replicationFactor": 3,
        "brokersSkewed": 0.0,
        "brokersLeaderSkewed": 0.0,
        "brokersSpread": 100.0,
        "preferredReplicas": 100.0,
        "underReplicated": 0.0,
        "otherConfigs": [
            {
                "key": "cleanup.policy",
                "value": "delete",
                "unit": null
            },
            {
                "key": "max.message.bytes",
                "value": 1048588,
                "unit": "bytes"
            },
            {
                "key": "message.timestamp.type",
                "value": "CreateTime",
                "unit": null
            },
            {
                "key": "min.insync.replicas",
                "value": 1,
                "unit": null
            },
            {
                "key": "retention.ms",
                "value": 48,
                "unit": "hour"
            },
            {
                "key": "segment.ms",
                "value": 604800000,
                "unit": "ms"
            }
        ]
    }
}
```

查询主题分区列表

该接口用于查询指定集群的主题的所有分区信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics/{topicName}/partitions/statuses HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	是	URL参数	待查询的主题名称
pageNo	int	是	Query参数	批量获取列表的查询的起始位置
pageSize	int	是	Query参数	每页包含的最大数量，最大数量不超过20，缺省值为10

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
pageNo	int	批量获取列表的查询的起始位置
pageSize	int	每页包含的最大数量，最大数量不超过20，缺省值为10
total	int	主题总共包含的分区数
partitions	List<TopicPartition>	主题分区信息列表，由 TopicPartition 组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
TOPIC_LIST_PARTITION_LIMIT_EXCEEDED_ERROR	获取Topic分区数量超过限制	451	获取Topic分区数量超过限制
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/topics/demo-topic-test/partitions

Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "pageNo": 0,
    "pageSize": 10,
    "totalCount": 2,
    "partitions": [{
            "topicName": "test",
            "partitionId": 1,
            "leaderId": 1027,
            "replicas": [1027,1011,1009],
            "inSyncReplicas": [1027,1011,1009],
            "minOffset": 0,
            "maxOffset": 1024,
            "messageNum": 1024,
            "lastUpdateTime": "2023-05-10T03:06:25Z"
        },
        {
            "topicName": "test",
            "partitionId": 2,
            "leaderId": 1003,
            "replicas": [1003,1008,1039],
            "inSyncReplicas": [1003,1008,1039],
            "minOffset": 0,
            "maxOffset": 1024,
            "messageNum": 1024,
            "lastUpdateTime": "2023-05-10T03:06:25Z"
        }
    ]
}
```

查询主题分区详情

该接口用于查询指定集群的主题的指定分区信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics/{topicName}/partitions/{partitionId}/statuses HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	是	URL参数	待查询的主题名称
partitionId	String	是	URL参数	待查询的分区ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
partition	TopicPartition	主题分区信息，具体参见 TopicPartition

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
TOPIC_PARTITION_NOT_EXIST	分区不存在	451	分区不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/topics/demo-topic-test/partitions
s
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topicName": "test",
    "partition": {
            "topicName": "test",
            "partitionId": 1,
            "leaderId": 1027,
            "replicas": [1027,1011,1009],
            "inSyncReplicas": [1027,1011,1009],
            "minOffset": 0,
            "maxOffset": 1024,
            "messageNum": 1024,
            "lastUpdateTime": "2023-05-10T03:06:25Z"
        }
}
```

查询主题订阅详情

该接口用于查询集群中指定消费组对指定主题（topic）各个分区的订阅详细信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics/{topicName}/consumer-groups/{groupName}/subscribe-details HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	是	URL参数	待查询topic的名称
groupName	String	是	URL参数	待查询的消费组名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
subscribePartitions	List<GroupTopicPartition>	消费组对topic各个分区的订阅详细信息，由 GroupTopicPartition 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics/test/consumer-groups/gro
s
Authorization:
bce-auth-v1/ec7f277752364a61af9f3bcb10eb803e/2023-05-08T09:39:11Z/3600/host;x-bce-console-rpc-id;x-bce-date/b
b
Host: kafka-api.bd.baidubce.com
x-bce-console-rpc-id: cf57b4b1-a86f-4423-86fb-736c22068d84
x-bce-date: 2023-05-08T09:39:11Z
```

返回示例

```
HTTP/1.1 200 OK
X-Bce-Request-Id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "subscribePartitions": [
        {
            "partitionId": 0,
            "topicName": "test",
            "consumerId": "client-id-test-67c2a19f-45e3-4ffd-bfc3-17a0efc65b65",
            "clientId": "client-id",
            "host": "192.168.1.1"
            "maxOffset": 66,
            "committedOffset": 66,
            "lag": 0,
            "lastConsumeTime": "2023-05-10T03:06:25Z"
        },
        {
            "partitionId": 1,
            "topicName": "test",
            "consumerId": "client-id-test-67c2a19f-45e3-4ffd-bfc3-17a0efc65b65",
            "clientId": "client-id",
            "host": "192.168.1.1"
            "maxOffset": 62,
            "committedOffset": 62,
            "lag": 0,
            "lastConsumeTime": "2023-05-10T03:06:25Z"
        },
        {
            "partitionId": 2,
            "topicName": "test",
            "consumerId": "client-id-test-67c2a19f-45e3-4ffd-bfc3-17a0efc65b65",
            "clientId": "client-id",
            "host": "192.168.1.1"
            "maxOffset": 61,
            "committedOffset": 61,
            "lag": 0,
            "lastConsumeTime": "2023-05-10T03:06:25Z"
        }
    ]
}
```

获取订阅主题的消费组列表

该接口用于查询集群中订阅指定主题的消费组列表信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/topics/{topicName}/consumer-groups HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
topicName	String	是	URL参数	待查询的topic名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
groups	List	消费组列表信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/topics/demo-topic-test/consumers
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "groups": [
        "group-id1-test",
        "group-id2-test",
        "group-id3-test",
    ]
}
```

消费组管理接口

删除消费组

该接口用于删除集群中的消费组。

- 集群处于服务中状态（ACTIVE）。

请求结构

```
DELETE /v{version}/clusters/{clusterId}/consumer-groups/{groupName} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待删除消费组的集群ID
groupName	String	是	URL参数	待删除的消费组名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
groupName	String	成功删除的消费组名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
CONSUMER_GROUP_NOT_FOUND	消费组不存在	451	消费组不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
DELETE
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/consumer-groups/group-id-test
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "groupName": "group-id-test"
}
```

查询消费组列表

该接口用于查询集群中的消费组列表信息。

请求结构

```
GET /v{version}/clusters/{clusterId}/consumer-groups HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
groupName	String	否	Query参数	消费组名称，用于模糊匹配

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
groups	List<Group>	消费组列表信息，由 Group 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/consumer-groups
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "groups": [
        {
            "groupName": "group-id3-test",
            "updateTime": "2023-05-10T03:06:25Z"
        },
        {
            "groupName": "group-id2-test",
            "updateTime": "2023-05-10T03:06:01Z"
        },
        {
            "groupName": "group-id1-test",
            "updateTime": "2023-05-10T03:05:48Z"
        }
    ]
}
```

重置消费组位点

该接口用于重置集群中指定消费组的位点数据。

- 集群处于服务中状态（ACTIVE）。

请求结构

```
POST /v{version}/clusters/{clusterId}/consumer-groups/{groupName}/offsets HTTP/1.1
Host: kafka-api.bd.baidubce.com
Authorization: authorization string

{
    "topicName": "topicName",
    "partitions": [0, 1],
    "resetStrategy": "resetStrategy",
    "resetValue": "resetValue"
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待重置消费组位点的集群ID
groupName	String	是	URL参数	待重置位点的消费组名称
topicName	String	是	RequestBody参数	待重置位点的topic名称
partitions	List	是	RequestBody参数	topic重置位点的分区ID
resetStrategy	String	是	RequestBody参数	重置位点的策略类型，可选值为： • tooffset：该策略用于将消费位点重置到某个具体的值，由resetValue指定具体的位点值 • shiftby：该策略用于将消费位点重置到当前位点的前方或者后方某个偏移量的值，由resetValue指定具体的偏移量。例如：当前位点为100，使用该策略传入10，则会将当前位点重置为110；如果传入-10，则会将当前位点重置为90 • todatetime：将消费位点重置到某个时间点，由resetValue指定具体的时间点值 • toearliest：将消费位点重置到最早位点 • tolatest：将消费位点重置到最新位点
resetValue	String	否	RequestBody参数	重置位点时的参数值，具体配置与resetStrategy的值有关，下列为resetStrategy参数值在不同情况下，resetValue的配置说明： • tooffset：resetValue需要配置为具体的位点值 • shiftby：resetValue需要配置为某个偏移量的值 • todatetime：resetValue需要配置为某个具体的时间值 • toearliest：resetValue无需配置 • tolatest：resetValue无需配置

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
groupName	String	成功重置位点的消费组名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
CONSUMER_GROUP_NOT_FOUND	消费组不存在	451	消费组不存在
TOPIC_NOT_EXIST	主题不存在	451	主题不存在
TOPIC_PARTITION_NOT_EXIST	分区不存在	451	分区不存在
CONSUMER_OPERATE_RESET_STRATEGY_INVALID_ERROR	消费组重置策略非法	451	CONSUMER_OPERATE_RESET_STRATEGY_INVALID_ERROR
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
 POST
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/consumer-groups/group-id-test/of
s
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com

{
 "topicName": "test",
 "partitions": [0, 1],
 "resetStrategy": "tolatest",
}
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "groupName": "group-id-test"
}
```

查询消费组订阅的主题列表

该接口用于查询集群指定消费组订阅的主题列表。

请求结构

```
 GET /v{version}/clusters/{clusterId}/consumer-groups/{groupName}/topics HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
groupName	String	是	URL参数	待查询的消费组名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
topics	List	主题信息列表，由主题名称组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/consumer-groups/demo-group-tes
s
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "topics": [
        "test2",
        "test1",
        "test"
    ]
}
```

用户管理接口

创建用户

该接口用于在集群中创建用户。

- 使用该接口时集群需要开启SASL身份认证。
- 集群处于服务中状态（ACTIVE）。

请求结构

```
POST /v{version}/clusters/{clusterId}/users HTTP/1.1
Host: kafka.api.bj.baidubce.com
Authorization: authorization string

{
    "username": "username",
    "password": "password"
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待创建用户的集群ID
username	String	是	Request Body参数	待创建的用户名称，需要符合规范：英文字母开头，4～16位字符，只能包含英文字母，数字，下划线(_)
password	String	是	Request Body参数	待创建的用户密码，需要符合规范：8～32位字符，英文字母、数字和符号必须同时存在，符号仅限!@#$%^*()，密码需要加密传输，详见密码加密传输规范定义

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
username	String	创建成功的用户名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
USER_PASSWORD_DECRYPT_ERROR	user密码编码失败	400	user密码编码失败
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
USER_NAME_ALREADY_EXISTS	user名称重复	451	user名称重复
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
POST http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/users
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com

{
   "username": "user",
   "password":
"PoxxC4OCgmSATSX9iNCcD6pZaPLJ0IXUJQRICJt0iuI9Qa3H+7mQRSQ/mIVrKrTNGM/z/I3eQuV60Jm1P8XRK+D5rkiJsMZgOVx

}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
   "username": "user"
}
```

删除用户

该接口用于删除集群中的用户。

- 使用该接口时集群需要开启SASL身份认证。
- 集群处于服务中状态（ACTIVE）。

请求结构

```
DELETE /v{version}/clusters/{clusterId}/users/{username} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待删除用户的集群ID
username	String	是	URL参数	待删除的用户名称

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
username	String	删除成功的用户名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
USER_NAME_INVALID	user名称不合法	400	user名称不合法
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
USER_NOT_FOUND_ERROR	user不存在	451	user不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
DELETE http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/users/test
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "username": "test"
}
```

重置用户密码

该接口用于重置集群中指定用户的密码信息。

- 使用该接口时集群需要开启SASL身份认证。
- 集群处于服务中状态（ACTIVE）。

请求结构

```
PUT /v{version}/clusters/{clusterId}/users/{username} HTTP/1.1
Host: kafka.api.bj.baidubce.com
Authorization: authorization string

{
    "password": "password"
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待重置密码用户的集群ID
username	String	是	URL参数	待重置密码的用户名称
password	String	是	Request Body参数	待重置密码的用户信息，需要符合规范：8～32位字符，英文字母、数字和符号必须同时存在，符号仅限!@#$%^*()，密码需要加密传输，详见密码加密传输规范定义

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
username	String	成功重置密码的用户名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
USER_PASSWORD_DECRYPT_ERROR	user密码编码失败	400	user密码编码失败
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
USER_NOT_FOUND_ERROR	user不存在	451	user不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/users/test
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com

{
    "password":
"PoxxC4OCgmSATSX9iNCcD6pZaPLJ0IXUJQRICJt0iuI9Qa3H+7mQRSQ/mIVrKrTNGM/z/I3eQuV60Jm1P8XRK+D5rkiJsMZgOVx

}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "username": "test"
}
```

查询用户列表

该接口用于查询集群的用户列表信息。

- 使用该接口时集群需要开启SASL身份认证。

请求结构

```
GET /v{version}/clusters/{clusterId}/users HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
users	LiIst<User>	用户列表信息，由 User 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/users
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "users": [
        {
            "username": "user3",
            "createTime": "2023-05-10T03:06:25Z"
        },
        {
            "username": "user2",
            "createTime": "2023-05-10T03:06:25Z"
        },
        {
            "username": "user1",
            "createTime": "2023-05-10T03:06:25Z"
        }
    ]
}
```

权限管理接口

创建权限

该接口用于在集群中创建权限信息。

- 使用该接口时集群需要开启权限管理。
- 集群处于服务中状态（ACTIVE）。

请求结构

```
POST /v{version}/clusters/{clusterId}/acls HTTP/1.1
Host: kafka.api.bj.baidubce.com
Authorization: authorization string

{
    "username": "username",
    "patternType": "patternType",
    "resourceType": "resourceType",
    "resourceName": "resourceName",
    "operations": ["operation", ""]
}
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待创建权限的集群ID
username	String	是	RequestBody参数	待创建权限的用户名称
patternType	String	是	RequestBody参数	待创建权限的匹配模式，具体可选模式参见AclPatternType
resourceType	String	是	RequestBody参数	待创建权限的资源类型，具体可选类型参见AclResourceType
resourceName	String	是	RequestBody参数	待创建权限的资源名称
operations	List	是	RequestBody参数	待创建权限的操作类型，具体可选类型参见AclOperationType

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
username	String	成功创建权限的用户名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
USER_NAME_INVALID	user名称不合法	400	user名称不合法
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
CLUSTER_ACL_NOT_ENABLED	集群未开启权限管理	451	集群未开启权限管理
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
POST http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/acls
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com

{
    "username": "user",
    "patternType": "LITERAL",
    "resourceType": "GROUP",
    "resourceName": "test_group",
    "operations": ["CONSUME", "PRODUCE"]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "username": "user"
}
```

删除权限

该接口用于删除集群中指定的权限信息。

- 使用该接口时集群需要开启权限管理。
- 集群处于服务中状态（ACTIVE）。

请求结构

```
DELETE
/v{version}/clusters/{clusterId}/acls?username={username}&patternType={patternType}&resourceType={resourceType}&reso
} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待删除权限的集群ID
username	String	是	Query参数	待删除权限的用户名称
patternType	String	是	Query参数	待删除权限的匹配模式，具体可选模式参见AclPatternType
resourceType	String	是	Query参数	待删除权限的资源类型，具体可选类型参见AclResourceType
resourceName	String	是	Query参数	待删除权限的资源名称
operation	String	是	Query参数	待删除权限的操作类型，具体可选类型参见AclOperationType

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
username	String	成功删除权限的用户名称

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
USER_NAME_INVALID	user名称不合法	400	user名称不合法
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_NOT_AVAILABLE	集群不可用	451	集群不可用
CLUSTER_ACL_NOT_ENABLED	集群未开启权限管理	451	集群未开启权限管理
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
DELETE
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/acls?username=user&patternTypE
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "username": "user"
}
```

查询权限列表

该接口用于查询集群的权限列表。

- 使用该接口时集群需要开启权限管理。

请求结构

```
GET /v{version}/clusters/{clusterId}/acls HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	待查询的集群ID
username	String	否	Query参数	权限的用户名称，用于查询匹配
patternType	String	否	Query参数	权限的匹配模式，用于查询匹配，具体可选模式参见AclPatternType
resourceType	String	否	Query参数	权限的资源类型，用于查询匹配，具体可选类型参见AclResourceType
resourceName	String	否	Query参数	权限的资源名称，用于查询匹配

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
acls	List<Acl>	权限信息列表，由 Acl 组成的集合

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_HAD_DELETED	集群已删除	451	集群已删除
CLUSTER_NOT_BELONGED	未拥有该集群	451	未拥有该集群
CLUSTER_ACL_NOT_ENABLED	集群未开启权限管理	451	集群未开启权限管理
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c045978741607711 4ab233/acls
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "acls": [
        {
            "username": "user",
            "patternType": "LITERAL",
            "resourceType": "TOPIC",
            "resourceName": "test_topic",
            "operation": "PRODUCE"
        },
        {
            "username": "user",
            "patternType": "LITERAL",
            "resourceType": "TOPIC",
            "resourceName": "test_topic",
            "operation": "CONSUME"
        },
        {
            "username": "user",
            "patternType": "LITERAL",
            "resourceType": "GROUP",
            "resourceName": "test_group",
            "operation": "CONSUME"
        }
    ]
}
```

集群配置接口

查询集群配置详情

该接口用于查询指定集群配置的详情信息。

请求结构

```
GET /v{version}/configs/{configId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
configId	String	是	Query参数	配置ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
config	ClusterConfig	返回的配置详情

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CONFIG_NOT_FOUND	配置不存在	400	配置不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/configs/96a2a7469f9e4a5290ffed29c038f91e
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "config": {
        "configId": "96a2a7469f9e4a5290ffed29c038f91e",
        "name": "test",
        "state": "USED",
        "description": "testset",
        "createTime": "2023-09-22T09:06:02Z"
    }
}
```

查询集群配置列表

该接口用于查询用户在该区域中的集群配置列表信息。

请求结构

```
GET /v{version}/configs HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
marker	String	否	Query参数	批量获取列表的查询的起始位置，需要设置为配置ID
maxKeys	int	否	Query参数	每页包含的最大数量，最大数量不超过1000，缺省值为1000
state	String	否	Query参数	用于指定查询配置的状态，USED表示配置正在使用的配置，UNUSED表示未使用的配置。如果不传入该参数，则表示查询所有状态的配置。
configName	String	否	Query参数	配置名称模糊匹配

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置，与请求接口时传入的marker参数值保持一致
maxKeys	int	每页包含的最大数量，，最大数量不超过1000，缺省值为1000
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	下一页所需要传递的marker值，当isTruncated为false时，该值为null
configs	List<ClusterConfig>	集群配置信息列表，由 ClusterConfig 组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/configs
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "marker": null,
    "isTruncated": false,
    "nextMarker": null,
    "maxKeys": 1000,
    "configs": [
        {
            "configId": "28c3658fecad4d3e811b2b522ad07b8e",
            "name": "cluster_config_1",
            "state": "USED",
            "description": "配置详情",
            "createTime": "2023-07-21T06:54:24Z"
        },
        {
            "configId": "28c3658fecad4d3e811b2b522ad07b8f",
            "name": "cluster_config_2",
            "state": "UNUSED",
            "description": "配置详情",
            "createTime": "2023-07-21T06:54:24Z"
        }
    ]
}
```

删除集群配置

该接口用于删除集群配置。

请求结构

```
DELETE /v{version}/configs/{configId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
configId	String	是	Query参数	配置ID

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
configId	String	配置ID

错误码

错误码	错误描述	HTTP状态码	描述
CONFIG_NAME_INVALID	配置名称不符合规则	400	配置名称不符合规则
CONFIG_NOT_FOUND	配置不存在	400	配置不存在
ERROR_PARAMS	请求参数错误	400	请求参数错误
CONFIG_DELETE_DISABLED	当前配置正在使用，禁止删除	400	当前配置正在使用，禁止删除

请求示例

```
DELETE http://kafka-api.bj.baidubce.com/v2/configs/33a168bb70c0459787416077114ab233
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "configId": "33a168bb70c0459787416077114ab233"
}
```

新增集群配置版本

该接口用于在集群配置中新增一个版本。

请求结构

```
POST /v{version}/configs/{configId}/revisions HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
configId	String	是	Query参数	配置ID
revisionId	Integer	是	Request Body参数	新增的版本号，需要是当前配置中最大版本号+1
description	String	是	Request Body参数	新增配置版本的描述信息
context	Map<String, String>	是	Request Body参数	新增配置版本的内容，如果缺少必选参数则自动补充必选项参数，必选参数列表可参见 配置参数介绍

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
revisionId	Integer	新增的版本ID

错误码

错误码	错误描述	HTTP状态码	描述
CONFIG_NOT_FOUND	配置不存在	400	配置不存在
ERROR_PARAMS	请求参数错误	400	请求参数错误
CONFIG_REVISION_INVALID	配置版本异常	400	配置版本异常
CONFIG_REVISION_NUM_LIMIT	当前配置版本数量达到上限	400	当前配置版本数量达到上限
CONFIG_PARAMS_ERROR	配置参数异常	400	配置参数存在异常

请求示例

```
POST http://kafka-api.bj.baidubce.com/v2/configs/33a168bb70c0459787416077114ab233/revision
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
  "revisionId": 2,
  "description": "描述信息",
  "context" : {
      "auto.create.topics.enable": "true",
      "log.retention.hours": "1",
      "log.retention.bytes": "-1"
  }
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "revisionId": "2"
}
```

查询集群配置版本详情

该接口用于查询集群配置版本中的详情信息。

请求结构

```
GET /v{version}/configs/{configId}/revisions/{revisionId} HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
configId	String	是	Query参数	配置ID
revisionId	Integer	是	Request Body参数	查询的配置版本号

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
revision	ClusterConfigRevisionDetail	配置版本的详细信息

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CONFIG_NOT_FOUND	配置不存在	400	配置不存在
CONFIG_REVISION_INVALID	配置版本异常	400	配置版本异常

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/configs/33a168bb70c0459787416077114ab233/2
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "revision": {
      "revisionId": 2,
      "state": "UNUSED",
      "description": "描述信息2",
      "createTime": "2023-08-10T07:33:52Z",
      "context": [
            {
              "name": "auto.create.topics.enable",
              "description": "是否允许自动创建 topic",
              "updateMode": "STATIC",
              "scope": [
                true,
                false
              ],
              "defaultValue": "false",
              "currentValue": "true",
              "type": "BOOLEAN",
              "unit": null
            },
            {
              "name": "log.retention.hours",
              "description": "消息保留时长（小时）",
              "updateMode": "STATIC",
              "scope": [
                1,
                2147483646
              ],
              "defaultValue": 168,
              "currentValue": "1",
              "type": "INT",
              "unit": "hour"
            }
        ]
    }
}
```

查询集群配置版本列表

该接口用于查询用户在该区域中的集群配置版本列表信息。

请求结构

```
GET /v{version}/configs/{configId}/revisions HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
configId	String	是	Query参数	集群配置ID
state	String	否	Query参数	用于指定查询配置的状态，USED表示配置正在使用的配置，UNUSED表示未使用的配置。如果不传入该参数，则表示查询所有状态的配置。

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
revisions	List<ClusterConfigRevision>	集群配置信息列表，由 ClusterConfigRevision 组成的集合

错误码

错误码	错误描述	HTTP状态码	中文解释
ERROR_PARAMS	请求参数错误	400	请求参数错误
CONFIG_NOT_FOUND	配置不存在	400	配置不存在
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
GET http://kafka-api.bj.baidubce.com/v2/configs/b4a5ac0b7fee46efad1c9b4c80eb21ea/revisions
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "revisions": [
        {
            "revisionId": 3,
            "state": "USED",
            "description": "描述信息",
            "createTime": "2023-12-18T11:51:26Z"
        },
        {
            "revisionId": 2,
            "state": "USED",
            "description": "描述信息",
            "createTime": "2023-12-18T11:50:33Z"
        },
        {
            "revisionId": 1,
            "state": "USED",
            "description": "描述信息",
            "createTime": "2023-12-18T11:14:54Z"
        }
    ]
}
```

创建集群配置

该接口用于创建集群配置。

请求结构

```
POST /v{version}/configs HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
name	String	是	Request Body参数	配置的名称
description	String	是	Request Body参数	配置的描述信息
context	Map<String, String>	是	Request Body参数	配置的内容，如果缺少必选参数则自动补充必选项参数，必选参数列表可参见 配置参数介绍

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
configId	String	配置ID

错误码

错误码	错误描述	HTTP状态码	描述
CONFIG_NAME_INVALID	配置名称不符合规则	400	配置名称不符合规则
CONFIG_NAME_ALREADY_EXISTS	配置重名	400	配置重名
CONFIG_PARAMS_ERROR	配置参数存在异常	400	配置参数存在异常
CONFIG_NUM_LIMIT	配置数量达到上限	400	配置数量达到上限

请求示例

```
POST http://kafka-api.bj.baidubce.com/v2/configs
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
  "name": "config_name",
  "description": "描述信息",
  "context" : {
      "auto.create.topics.enable": "true",
      "log.retention.hours": "1",
      "log.retention.bytes": "-1"
  }
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "configId": "33a168bb70c0459787416077114ab233"
}
```

集群变更接口

变更存储策略

该接口用于变更集群磁盘存储策略。

请求结构

```
PUT /v{version}/clusters/{clusterId}/update-storage-policy HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
storagePolicyEnabled	Boolean	是	Request Body参数	是否开启磁盘水位处理
storagePolicy	StoragePolicy	是	Request Body参数	磁盘阈值策略的方式，参见 StoragePolicy

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
STOCK_FAILED	库存不足	451	库存不足
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
 PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/update-storage-policy
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "storagePolicyEnabled": true,
    "storagePolicy": {
        "type": "AUTO_DELETE",
        "autoDelete": {
            "diskUsedThresholdPercent": 75,
            "logMinRetentionMs": 3600000,
            "logMinRetentionBytes": 2147483648
        }
 }
}
```

返回示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

变更集群配置

该接口用于变更集群配置。

请求结构

```
 PUT /v{version}/clusters/{clusterId}/update-kafka-config HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
configId	String	否	Request Body参数	目标集群配置ID，不传入该参数则使用默认参数
revisionId	Integer	否	Request Body参数	目标集群配置中的版本号，不传入该参数则使用默认参数

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/update-kafka-config
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "configId": "28c3658fecad4d3e811b2b522ad07b8e",
    "revisionId": 2
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

增加节点数量

该接口用于增加集群节点数量。

请求结构

```
PUT /v{version}/clusters/{clusterId}/increase-broker-count HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
numberOfBrokerNodes	Integer	是	Request Body参数	目标集群总节点数目(可用区倍数)
couponIds	List<String>	否	Request Body参数	使用代金券时，可以指定使用哪些代金券

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
STOCK_FAILED	库存不足	451	库存不足
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/increase-broker-count
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
{
    "numberOfBrokerNodes": 5,
    "couponIds": [
        "xxxxx"
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

扩容磁盘容量

该接口用于扩容集群节点磁盘容量。

请求结构

```
PUT /v{version}/clusters/{clusterId}/expand-broker-disk-capacity HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
storageSize	Long	是	Request Body参数	目标节点磁盘容量
couponIds	List<String>	否	Request Body参数	使用代金券时，可以指定使用哪些代金券

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
STOCK_FAILED	库存不足	451	库存不足
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT
http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/expand-broker-disk-capacity
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "storageSize": 100,
    "couponEnabled": true,
    "couponIds": [
        "xxxxx"
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

集群公网开关

该接口用于变更集群公网开关。

请求结构

```
PUT /v{version}/clusters/{clusterId}/eips/switch HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
publicIpBandwidth	Integer	否	Request Body参数	公网带宽值，publicIpEnabled = true 时有效
publicIpEnabled	Boolean	是	Request Body参数	公网开关
aclEnabled	Boolean	否	Request Body参数	公网开关
authenticationMode	List<String>	否	Request Body参数	开关公网会影响集群所使用的认证方式，此处允许用户选择新的认证方式用以变更。 可选的值为：Node（无需身份认证）、SASL_SCRAM（使用SASL/SCRAM机制进行身份认证）、SASL_PLAIN（使用SASL/PLAIN机制进行身份认证）、SSL（使用SSL证书进行双向认证）。 可不传，会依据公网开关自动选择认证方式默认值，公网开启时，会默认增加SASL_SCRAM认证方式；公网关闭时，不会更改当前认证方式。
couponIds	List<String>	否	Request Body参数	使用代金券时，可以指定使用哪些代金券。开启公网时，才需要使用代金券。

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
CLUSTER_PUBLIC_IP_DISABLED	集群未开启公网访问	451	集群未开启公网访问
CLUSTER_PUBLIC_IP_ENABLED	集群已开启公网访问	451	集群已开启公网访问
CLUSTER_PUBLIC_IP_BANDWIDTH_ILLEGAL	公网带宽参数错误	451	公网带宽参数错误
CLUSTER_AUTHENTICATION_MODE_ILLEGAL	集群认证模式错误	451	集群认证模式错误
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/eips/switch
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "publicIpEnabled": true,
    "publicIpBandwidth": 5,
    "couponIds": [
    "xxxxx"
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

变更用户安全组

该接口用于变更集群用户安全组。

请求结构

```
PUT /v{version}/clusters/{clusterId}/security-groups HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
securityGroupIds	List< String >	是	Request Body参数	待变更的安全组列表

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
UPDATE_CLUSTER_SECURITY_GROUP_NOT_CHANGE	安全组未发生变更	451	安全组未发生变更
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/security-groups
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc
1
Host: kafka-api.bj.baidubce.com
{
    "securityGroupIds": ["xxxxxx"]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

变更节点机型

该接口用于变更集群节点机型。

请求结构

```
PUT /v{version}/clusters/{clusterId}/update-broker-node-type HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
nodeType	String	是	Request Body参数	目标节点类型
couponIds	List< String>	否	Request Body参数	使用代金券时，可以指定使用哪些代金券

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
STOCK_FAILED	库存不足	451	库存不足
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/update-broker-node-type
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
{
    "nodeType": "kafka.g4.c2m8",
    "couponIds": [
        "xxxxx"
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

变更公网带宽

该接口用于变更集群公网带宽。

请求结构

```
PUT /v{version}/clusters/{clusterId}/eip-bandwidths/resize HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
publicIpBandwidth	Integer	是	Request Body参数	公网新带宽值
couponIds	List< String >	否	Request Body参数	使用代金券时，可以指定使用哪些代金券。增加带宽时，才需使用代金券

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
CLUSTER_PUBLIC_IP_DISABLED	集群未开启公网访问	451	集群未开启公网访问
CLUSTER_PUBLIC_IP_BANDWIDTH_UNCHANGED	集群公网带宽未变更	451	集群公网带宽未变更
CLUSTER_PUBLIC_IP_BANDWIDTH_ILLEGAL	公网带宽参数错误	451	公网带宽参数错误
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/eip-bandwidths/resize
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
{
    "publicIpBandwidth": 5,
    "couponIds": [
    "xxxxx"
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

变更访问配置

该接口用于变更集群访问配置。

请求结构

```
PUT /v{version}/clusters/{clusterId}/update-access-config HTTP/1.1
Host: kafka-api.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其他特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
version	String	是	URL参数	API版本号
clusterId	String	是	URL参数	集群ID
aclEnabled	Boolean	是	Request Body参数	是否开启权限控制
authentications	List<Authentication>	是	Request Body参数	集群所使用的认证方式，参见 Authentication

返回头域

除公共头域外，无其他特殊头域。

返回参数

参数名称	类型	描述
clusterId	String	集群ID
jobId	String	变更任务ID

错误码

错误码	错误描述	HTTP状态码	描述
ERROR_PARAMS	请求参数错误	400	请求参数错误
CLUSTER_NOT_FOUND	集群不存在	451	集群不存在
CLUSTER_NOT_ACTIVE	集群非运行状态	451	集群非运行状态
UPDATE_CLUSTER_FORBIDDEN	集群不可执行更新操作	451	集群不可执行更新操作
STOCK_FAILED	库存不足	451	库存不足
UPDATE_CLUSTER_ORDER_FAILED	集群更新下单失败	451	集群更新下单失败
INTERNAL_ERROR	服务内部错误	500	服务内部错误

请求示例

```
PUT http://kafka-api.bj.baidubce.com/v2/clusters/33a168bb70c0459787416077114ab233/update-access-config
Authorization:
bce-auth-v1/ALTAKaiKeDfBD880eMWBE5oIGE/2023-05-08T11:43:45Z/1800/host;x-bce-date/322f3f98ce57d296c0f5abc1
Host: kafka-api.bj.baidubce.com
{
    "aclEnabled": true,
    "authentications": [
        {
            "mode": "NONE"
        }
    ]
}
```

返回示例

```
HTTP/1.1 200 OK
x-bce-request-id: 97e6d4ad-6fca-4609-ad4d-9a27c4d1a362
Date: Mon, 08 May 2023 09:39:11 GMT
Content-Type: application/json;charset=UTF-8
Server: BWS

{
    "clusterId": "33a168bb70c0459787416077114ab233",
    "jobId": "16411be300de729baaee9b4dc3f0e4cd"
}
```

附录

Model对象定义

AccessEndpoint

参数名称	类型	描述
securityProtocol	String	Kafka集群的接入点协议，不同的协议具有不同的使用场景： • SASL_PLAINTEXT (VPC)：在VPC网络环境下使用，需要使用SASL用户名和密码进行身份认证，传输数据不进行加密 • PLAINTEXT (VPC)：在VPC网络环境下使用，不进行身份认真和传输数据加密 • SASL_SSL (外网)：在公网环境下使用，需要使用SASL用户名和密码进行身份认证，并且传输数据进行加密 • SSL (VPC、外网)：使用SSL协议双向认证，进行传输数据的加密
endpoints	String	Kafka集群对应当前访问协议的接入点地址
network	String	Kafka集群对应的网络环境，具体可选VPC，WAN

Cluster

参数名称	类型	描述
clusterId	String	集群的ID
clusterSid	String	集群的短ID
name	String	集群的名称
region	String	集群地域
type	String	集群类型，具体值可为PROVISIONED，SERVERLESS
mode	String	集群模式，具体值可为HP、HA
state	String	集群状态，具体可参见ClusterState
kafkaVersion	String	集群的版本，当前仅支持2.7.2
logicalZones	List<String>	集群所在的可用区列表
payment	String	集群的付费方式，可选的值为：Prepaid（预付费）和 Postpaid（后付费）
aclEnabled	Boolean	集群是否开启权限控制
publicIpEnabled	Boolean	集群是否开启外网
intranetIpEnabled	Boolean	集群是否开启产品间转储
authenticationModes	List<String>	集群所使用的认证方式，可选的值为：Node（无需身份认证）、SASL_SCRAM（使用SASL/SCRAM机制进行身份认证）、SASL_PLAIN（使用SASL/PLAIN机制进行身份认证）、SSL（使用SSL证书双向认证）
tags	List<Tag>	集群所绑定的标签信息
createTime	String	创建用户时的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z
expireTime	String	集群的过期时间，当付费方式为后付费时，该值固定为-1

ClusterDetail

参数名称	类型	描述
clusterId	String	集群的ID
clusterSid	String	集群的短ID
name	String	集群的名称
region	String	集群地域
type	String	集群类型，具体值可为PROVISIONED，SERVERLESS
mode	String	集群模式，具体值可为HP、HA
state	String	集群状态，具体可参见ClusterState
provisioned	Provisioned	provisioned type的集群上下文，具体参见Provisioned
tags	List<Tag>	集群所绑定的标签信息
createTime	String	创建用户时的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

Provisioned | 参数名称 | 类型 | 描述 | | ---------- | ------ | ---- | | kafkaVersion | String | 集群的版本，当前仅支持2.7.2 | | billing | Billing | billing对象，具体可参见Billing | | logicalZones | List<String> | 可用区列表 | | vpc | Vpc | VPC信息，具体可参见Vpc，用于集群展示 | | | subnets | List<Subnet> | 子网列表，具体可参见Subnet ，用于集群展示 | | securityGroups | List<SecurityGroup> | 安全组信息，具体可参见SecurityGroup ，用于集群展示 | | vpcId | String | vpc短Id，仅用于创建集群的请求 | | subnetIds | List<String> | 子网短Id列表，仅用于创建集群的请求 | | securityGroupIds | List<String> | 安全组Id列表，仅用于创建集群的请求 | | publicIpEnabled | boolean | 公网访问开关 | | publicIpBandwidth | int | 公网访问带宽，限制为1~200的整数 | | intranetIpEnabled | boolean | 产品间转储开关 | | aclEnabled | boolean | 权限管理开关 | | authentications | List<Authentication> | 集群所使用的认证方式列表，参见 Authentication| | numberOfBrokerNodes | int | 集群节点总数 | | numberOfBrokerNodesPerZone | int | 集群单分区节点数 | | nodeType | String | 节点类型，如kafka.g4.c2m8 | | storageMeta | StorageMeta | 存储信息，具体可参见StorageMeta | | storagePolicyEnabled | Boolean | 是否开启磁盘水位处理| | storagePolicy | StoragePolicy | 磁盘阈值策略，具体可参见 StoragePolicy| | configMeta | ConfigMeta | 自定义配置模板的元信息，具体可参见ConfigMeta | | deploySetEnabled | boolean | 是否开启部署集 |

Billing

参数名称	类型	描述
payment	String	付费模式
timeLength	int	预付费时长
timeUnit	String	预付费时长单位
expireTime	String	预付费到期时间, 用于展示
autoRenewEnabled	boolean	自动续费开关
autoRenewTimeLength	int	自动续费时长
autoRenewTimeUnit	String	自动续费时长单位
couponIds	List	有值时使用代金券，可以指定使用哪些代金券
isAutoPay	boolean	是否自动支付，默认为true

Vpc

参数名称	类型	描述
vpcId	String	VPC的ID
vpcUuid	String	VPC的UUID，仅用于展示
name	String	VPC的名称
cidr	String	VPC的CIDR值

Subnet

参数名称	类型	描述
subnetId	String	子网 ID
subnetUuid	String	子网 UUID，仅用于展示
name	String	子网名称
subnetType	String	子网类型
zone	String	区域
vpcId	String	VPC的ID
cidr	String	CIDR值

SecurityGroup

参数名称	类型	描述
securityGroupId	String	安全组ID
securityGroupUuid	String	安全组的UUID，仅用于展示
name	String	安全组名称
vpcId	String	VPC的ID
vpcUuid	String	VPC的UUID，仅用于展示

StorageMeta

参数名称	类型	描述
storageType	String	存储类型
storageSize	int	存储大小
numberOfDisk	int	磁盘数量

StoragePolicy

参数名称	类型	描述
type	StoragePolicyType	存储策略类型，具体可参见：StoragePolicyType
autoDelete	AutoDelete	自动删除策略相关参数，具体可参见：AutoDelete

AutoDelete

参数名称	类型	描述
diskUsedThresholdPercent	Integer	磁盘容量阈值，达到该阈值后集群会按比例删除该磁盘上所有主题分区部分最早的消息
logMinRetentionMs	Long	主题分区最小保留时长，单位 ms
logMinRetentionBytes	Long	主题分区最小保留大小，单位 byte

ConfigMeta | 参数名称 | 类型 | 描述 | | ---------- | ------ | ---- | | configId | String | 集群配置ID | | revisionId | String | 配置版本ID | | context | Map<String, String> | 默认配置下的单独指定的键值对 |

Authentication | 参数名称 | 类型 | 描述 | | ---------- | ------ | ---- | | mode | String | 认证模式，可选的值为：Node（无需身份认证）、SASL_SCRAM（使用SASL/SCRAM机制进行身份认证）、SASL_PLAIN（使用SASL/PLAIN机制进行身份认证）、SSL（使用SSL证书双向认证） | | context | String | 复杂对象,存放证书等 |

Node

参数名称	类型	描述
brokerId	Int	当前节点的序号，从1开始递增到节点个数
host	String	节点的host
nodeId	String	节点的ID
status	NodeState	当前节点的状态，具体可选状态参见NodeState
publicIp	String	节点的公网地址，只有集群开启公网时才会有具体的值，否则为null
internalIp	String	节点的VPC地址

Acl

参数名称	类型	描述
username	String	用户名称
patternType	AclPatternType	匹配模式
resourceType	String	资源类型
resourceName	String	资源名称
operation	List	操作类型列表

Tag

参数名称	类型	描述
tagKey	String	标签的键，可包含大小写字母、数字、中文以及-_ /.特殊字符，长度1-65
tagValue	String	标签的值，可包含大小写字母、数字、中文以及-_ /.特殊字符，长度0-65

Group

参数名称	类型	描述
groupName	String	消费组的名称
updateTime	String	消费组最后一次消费的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

User

参数名称	类型	描述
username	String	用户名称
createTime	String	创建用户时的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

TopicConfig

参数名称	类型	描述
message.timestamp.type	String	消息息时间戳类型，LogAppendTime、CreateTime，默认值：CreateTime
cleanup.policy	String	清理策略， compact、delete， 默认值： delete
min.insync.replicas	String	同步副本数， 最小为1，最大为主题的副本数，默认值：1
retention.ms	String	消息保留时长，范围：1~1728000000，默认值：172800000
segment.ms	String	日志分片时间， 范围：1~1048576000，默认值：604800000
max.message.bytes	String	单条消息最大值， 范围：1~1048576000， 默认值： 1048588

GroupTopicPartition

参数名称	类型	描述
partitionId	int	消费者实例所订阅topic的具体分区ID
topicName	String	消费者实例所订阅的topic名称
consumerId	String	正在消费当前分区的消费者实例ID，如果显示“-”则说明当前分区此时没有被消费者消费
clientId	String	正在消费当前分区的客户端ID
host	String	正在消费当前分区的客户端IP
maxOffset	long	当前分区的最大位点
committedOffset	long	当前消费者实例的消费位点
lag	long	当前消费者实例在当前分区的消息堆积量，该值为maxOffset与committedOffset的差值
lastConsumeTime	String	当前消费者最后消费的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

Topic

参数名称	类型	描述
topicName	String	topic的名称
createTime	String	创建topic时的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

TopicDetail

参数名称	类型	描述
topicName	String	topic的名称
partitionNum	int	topic的分区数量
replicationFactor	int	topic的副本数量
brokersSkewed	double	broker 倾斜百分比
brokersLeaderSkewed	double	broker leader倾斜百分比
brokersSpread	double	broker覆盖率
preferredReplicas	double	Preferred Replicas %
underReplicated	double	Under-replicated %
otherConfigs	List<TopicConfig>	topic的高级配置内容

TopicPartition

参数名称	类型	描述
topicName	String	主题名称
partitionId	int	topic的分区ID
leaderId	int	当前分区leader副本所在的节点ID
replicas	List	当前分区副本所在节点ID列表
inSyncReplicas	List	当前分区已同步副本所在节点ID列表
minOffset	long	当前分区的最小位点
maxOffset	long	当前分区的最大位点
messageNum	long	当前分区拥有的消息总量
lastUpdateTime	String	当前分区最近更新时间的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

ClusterConfig

参数名称	类型	描述
configId	String	配置ID
name	String	配置名称
state	String	配置的状态，USED表示正在被集群使用，UNUSED表示没有集群使用
description	String	配置的描述信息
createTime	String	配置的创建时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

ClusterConfigRevision

参数名称	类型	描述
revisionId	String	配置版本ID
state	String	配置版本的状态，USED表示正在被集群使用，UNUSED表示没有集群使用
description	String	配置版本的描述信息
createTime	String	配置版本的创建时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z

ClusterConfigRevisionDetail | 参数名称 | 类型 | 描述 | | ---------- | ---------- | ---------- | | revisionId | String | 配置版本ID | | state | String | 配置版本的状态，USED表示正在被集群使用，UNUSED表示没有集群使用 | | description | String | 配置版本的描述信息 | | createTime | String | 配置版本的创建时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z | | context | List<TopicConfigOption> | 配置版本中具体的参数信息 |

ClusterConfigOption | 参数名称 | 类型 | 描述 | | ---------- | ---------- | ---------- | | name | String | 参数名称，参数介绍可参见：配置参数介绍 | | description | String | 参数对应的描述 | | updateMode | String | 参数的更新模式。STATIC表示静态参数，更新时需要重启集群，DYNAMIC表示动态参数，更新时不需要重启集群 | | scope | Object[] | 参数对应的范围，如果参数的类型是INT、LONG等范围，则该值类似于[1, 168]，表示最小值为1，最大值为168；如果参数的类型是ENUM、BOOLEAN，则其中每一个元素表示可选的值，类似于[delete, compact] | | defaultValue | Object | 参数的默认值 | | currentValue | Object | 参数的当前值 | | type | String | 参数的类型，例如：BOOLEAN、INT、LONG、ENUM | | unit | String | 参数的单位，例如：ms、byte | | category | String | 参数的分类，例如：主题、消费组 | | overrideMode | String | 该参数是否必选。REQUIRED表示该参数在集群中必须进行配置，OPTIONAL表示可选 |

OperationGroup

参数名称	类型	描述
groupName	String	任务所属分组的类型名称，具体可选类型参见OperationGroupType
state	OperationState	任务所属分组的执行状态，具体可选状态参见OperationState

Job

参数名称	类型	描述
jobId	String	任务ID
name	JobType	任务的类型，具体可选类型参见JobType
status	JobStatus	任务的状态，具体可选状态参见JobStatus
operations	List<Operation>	任务包含的操作列表，操作内容见Operation

Operation | 参数名称 | 类型 | 描述 | | ---------- | ---------- | ---------- | | jobId | String | 任务ID | | name | JobType | 任务的类型，具体可选类型参见JobType | | status | JobStatus | 任务的状态，具体可选状态参见JobStatus | | operationId | String | 操作ID | | type | OperationType | 任务的类型，具体可选类型参见OperationType | | state | OperationState | 任务的状态，具体可选状态参见OperationState | | process | int | 任务执行的进度百分比值，取值范围为0到100 | | schedule | String | 调度状态，字段可选：EXECUTE/SUSPEND, 分别代表执行中和挂起中| | started | boolean | 任务是否已经启动 | | createTime | String | 任务创建的时间，符合日期与时间规范的格式，例如：2023-05-

10T03:06:01Z | | startTime | String | 任务开始启动的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z | | endTime | String | 任务结束的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z |

OperationDetail

参数名称	类型	描述
JobId	String	任务ID
type	OperationType	任务的类型，具体可选类型参见OperationType
operationId	String	操作ID
state	OperationState	任务的状态，具体可选状态参见OperationState
process	int	任务执行的进度百分比值，取值范围为0到100
schedule	String	调度状态，字段可选：EXECUTE/SUSPEND, 分别代表执行中和挂起中
started	boolean	任务是否已经启动
createTime	String	任务创建的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z
startTime	String	任务开始启动的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z
endTime	String	任务结束的时间，符合日期与时间规范的格式，例如：2023-05-10T03:06:01Z
groups	List<OperationGroup>	任务所属的分组信息列表
sourceContext	String	原始内容
targetContext	String	目标更改内容

类型编码定义

StorageType

编码	描述
ENHANCED_SSD_PL1	增强型SSD磁盘类型
SSD	高性能云磁盘

AclPatternType

编码	描述
LITERAL	精确匹配
PREFIXED	前缀匹配

AclResourceType

编码	描述
TOPIC	主题
CLUSTER	集群
GROUP	消费组
TRANSACTIONAL_ID	事务

AclOperationType

编码	描述
PRODUCE	发布
CONSUME	订阅
IDEMPOTENT_WRITE	幂等写
WRITE	写

JobType

编码	描述
INCREASE_BROKER_COUNT	增加节点数量
DECREASE_BROKER_COUNT	减少节点数量
UPGRADE_BROKER_NODE_TYPE	升级节点规格
DOWNGRADE_BROKER_NODE_TYPE	降低节点规格
EXPAND_BROKER_DISK_CAPACITY	扩容节点磁盘
REASSIGN_PARTITION	主题重新分区
UPDATE_ACCESS_CONFIG	变更集群访问
RESTART_BROKER	重启集群节点
UPDATE_KAFKA_CONFIGURATION	更新集群配置
UPGRADE_KAFKA_VERSION	升级集群版本
SUSPEND_KAFKA_CLUSTER	停止集群服务
RESUME_KAFKA_CLUSTER	启动集群服务

OperationType

编码	描述
INCREASE_BROKER_COUNT	扩容节点
INCREASE_BROKER_COUNT_ROLLBACK	回滚扩容
DECREASE_BROKER_COUNT	缩容节点
DECREASE_BROKER_COUNT_ROLLBACK	回滚缩容
UPDATE_BROKER_NODE_TYPE	变配节点
UPDATE_BROKER_NODE_TYPE_ROLLBACK	回滚变配
EXPAND_BROKER_DISK_CAPACITY	扩容磁盘
EXPAND_BROKER_DISK_CAPACITY_ROLLBACK	回滚扩容
REASSIGN_PARTITION	调整主题
REASSIGN_PARTITION_ROLLBACK	回滚调整
UPDATE_ACCESS_CONFIG	变更访问
UPDATE_ACCESS_CONFIG_ROLLBACK	回滚变更
RESTART_KAFKA_CLUSTER	重启集群
RESTART_KAFKA_CLUSTER_ROLLBACK	回滚重启
RESTART_BROKER	重启节点
RESTART_BROKER_ROLLBACK	回滚重启

OperationGroupType

编码	描述
CHECK_BCC_RESOURCE_FOR_KAFKA	Kafka BCC资源余量检查
CHECK_CDS_RESOURCE_FOR_KAFKA	Kafka CDS资源余量检查
APPLY_DEPLOY_SET_RESOURCE	部署集资源申请
APPLY_BCC_RESOURCE_FOR_KAFKA	Kafka BCC资源申请
APPLY_BCC_RESOURCE_FOR_ZOOKEEPER	ZooKeeper BCC资源申请
APPLY_EIP_RESOURCE_FOR_KAFKA	Kafka EIP资源申请
APPLY_CDS_RESOURCE_FOR_KAFKA	Kafka CDS资源申请
RESIZE_NODE_TYPE_RESOURCE_KAFKA	Kafka 节点类型变更
INIT_DEPLOY_ENVIRONMENT	初始化部署环境
RESIZE_FS_CAPACITY_FOR_KAFKA	扩容文件系统大小
CONFIG_KAFKA_SERVICE	更新 Kafka 服务配置
START_KAFKA_SERVICE	启动 Kafka 服务
RESTART_KAFKA_SERVICE	重启 Kafka 服务
START_ZOOKEEPER_SERVICE	启动 ZooKeeper 服务
UPDATE_METADATA_FOR_CLUSTER	集群元数据信息更新
TOPIC_REASSIGN_PARTITION	主题重分区
CANCEL_REASSIGN_PARTITION	取消主题重分区
RELEASE_ALL_RESOURCE	释放所有资源
NOTHING	空执行

StoragePolicyType

编码	描述
AUTO_DELETE	自动删除

状态编码定义

NodeServiceState

编码	描述
NEW	等待部署
ALIVE	服务中
DEAD	异常
LOST	丢失

ClusterState

编码	描述
NEW	新建
DEPLOYING	正在部署
REBOOTING	重启中
ACTIVE	服务中
DEPLOY_FAILED	部署失败
UPDATING	正在变更
UPDATE_ROLLBACKING	变更回滚中
UPDATE_ROLLBACK_FAILED	变更回滚失败
SUSPENDED	已停服

JobStatus | 编码 | 描述 | | --- | --- | | NEW | 新建 | | PENDING | 待执行 | | RUNNING | 执行中 | | SUSPENDED | 暂停 | | CANCELLED | 取消 | | FINISHED | 成功 | | FAILED | 失败 |

OperationState

编码	描述
NEW	新建
PENDING	待执行
RUNNING	执行中
SUSPENDED	暂停
CANCELLED	取消
FINISHED	成功
FAILED	失败

更新记录

2023-06-30

消息服务 for Kafka API发布。

2023-12-30

新增集群配置管理、集群变更、集群启停相关接口，任务管理接口部分字段调整。

开发指南

概述

本文主要介绍如何通过各类开发语言来连接并使用专享版消息服务 for Kafka集群。

相关示例代码可参考：kafka-demo

访问协议介绍

概述

本文介绍专享版消息服务 for Kafka所支持的访问协议。

访问协议

- PLAINTEXT

- SASL_PLAINTEXT
- SASL_SSL
- SSL

PLAINTEXT

PLAINTEXT协议适用场景

- 消息传输时不加密，消息收发时不鉴权

无需认证直接通过接入点访问。

创建集群时认证方式选择 None（无需身份认证），且在同 VPC 网络下访问，可以使用 PLAINTEXT 协议接入。

PLAINTEXT协议使用注意

PLAINTEXT协议所采用的端口号为9092。

SASL_PLAINTEXT

SASL_PLAINTEXT协议适用场景

- 消息传输时不加密，消息收发时需要鉴权。

在同 VPC 网络下访问且已选择SASL/SCRAM身份认证、SASL/PLAIN身份认证方式，可使用 SASL_PLAINTEXT 协议接入。

SASL_PLAINTEXT协议使用注意

SASL_PLAINTEXT协议所采用端口号为9093。

使用SASL_PLAINTEXT协议时需要根据用户名和密码进行认证，创建用户参考：创建用户。

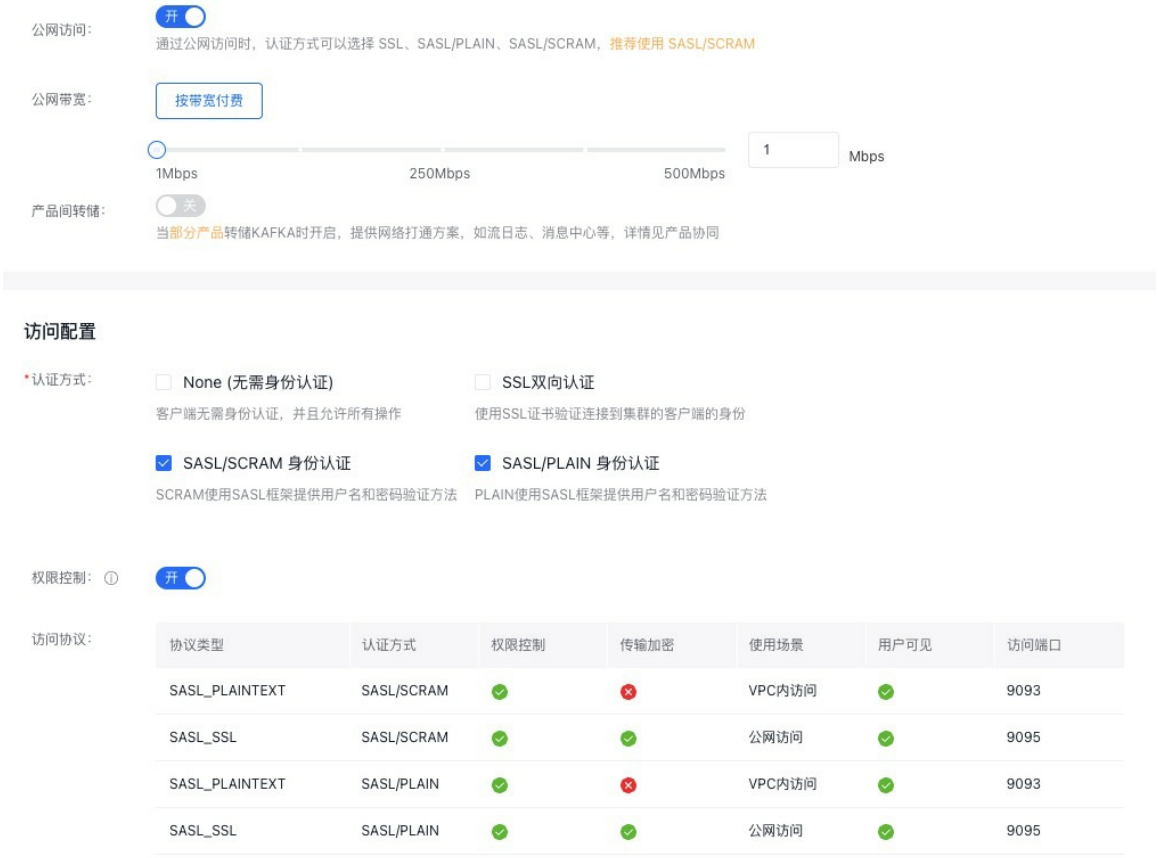
如果开启了权限控制，则需要给用户配置访问权限，权限配置参考：创建权限

SASL_SSL

SASL_SSL协议适用场景

- 消息传输时加密，消息收发时需要鉴权。

开启公网访问且已选择SASL/SCRAM身份认证、SASL/PLAIN方式，可使用SASL_SSL协议接入。 如果要支持在VPC网络外访问的话，就需要在创建集群时开启公网访问功能。开启公网访问后，会默认为开启SASL_PLAINTEXT、SASL_SSL协议来保证安全性。



SASL_SSL协议使用注意

SASL_SSL协议所采用的端口号为9095。

创建用户参考：创建用户。

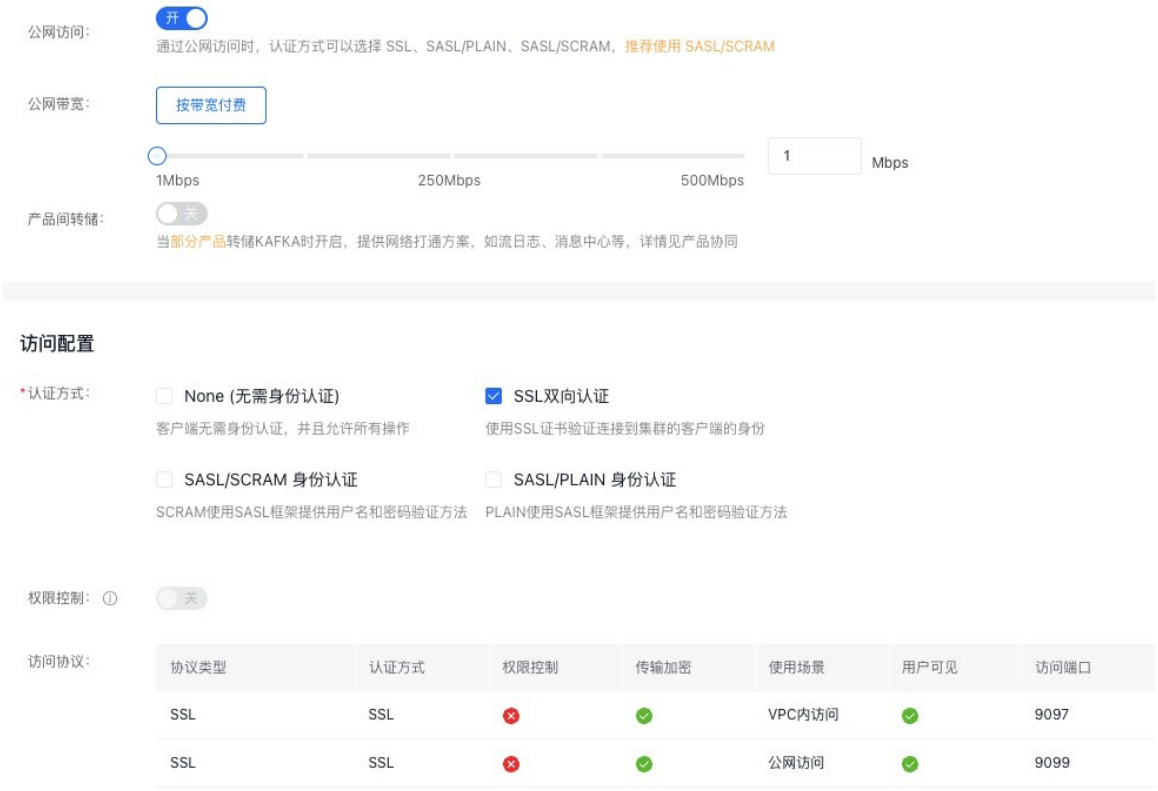
下载证书参考：如何下载证书？。

SSL

SSL协议适用场景

- 消息传输时加密，消息收发时不鉴权。

需要消息传输加密时可以选择SSL协议接入，可以在公网环境下或者VPC内使用SSL协议。



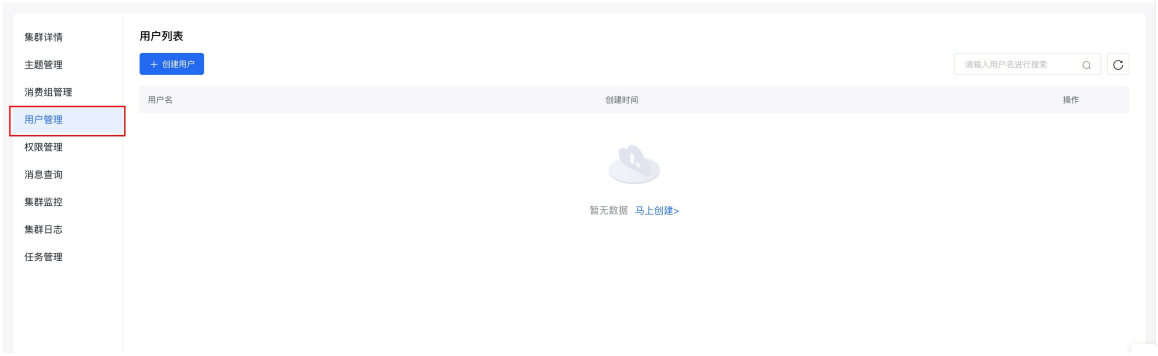
SSL协议使用注意

SSL协议在VPC内所采用的端口号为9097，在公网环境下采用的端口号为9099。

下载证书参考：如何下载证书？。

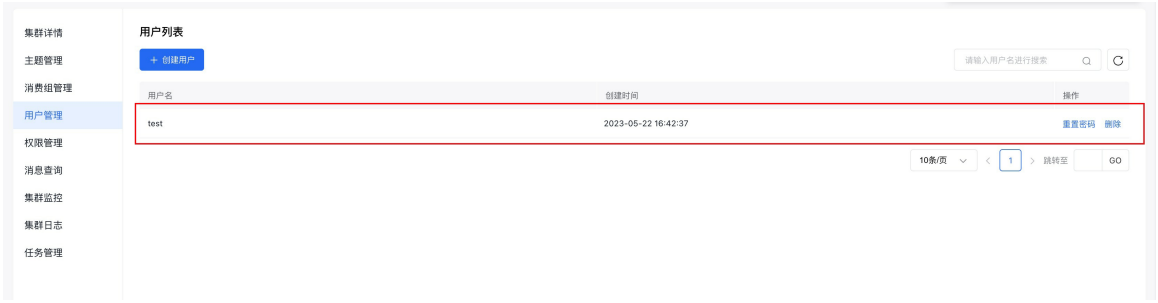
如何创建用户？

1. 点击集群进入**集群详情**页面，在左侧边栏中找到并点击**用户管理**。



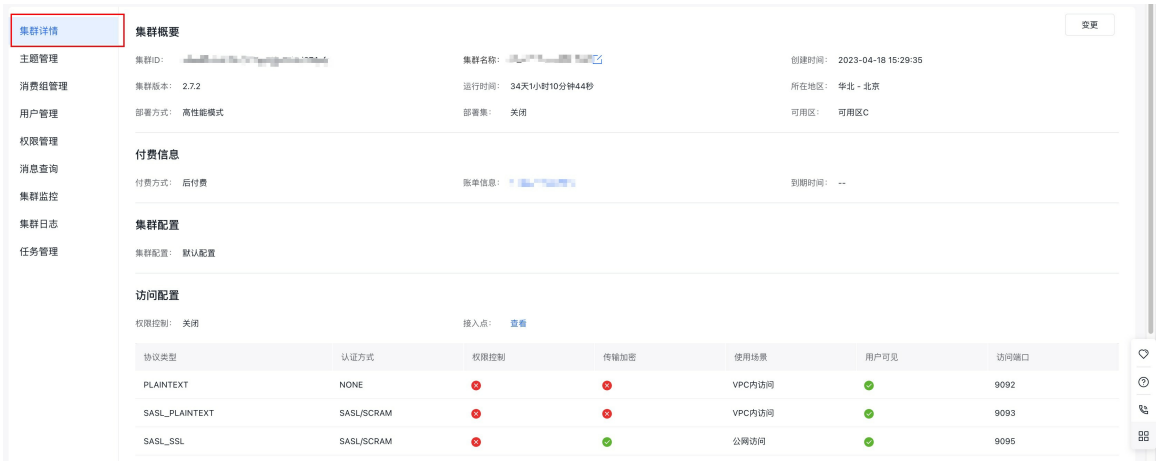
2. 点击**创建用户**，输入需要创建的用户名和密码，并点击确认。

3. 在**用户管理**中就可以看到刚才创建的用户了，也可以通过右侧的**重置密码**、**删除**按钮对用户进行管理。



如何下载证书？

1. 点击集群进入**集群详情**页面，在左侧边栏中找到并点击**集群详情**。



2. 在**集群详情**页面找到**访问配置**。

3. 点击接入点后面的**查看**链接。

4. 得到弹窗，点击图中『ssl.cert.zip』链接即可下载证书。

接入点查看

概述

本文主要介绍如何查看专享版消息服务 for Kafka集群的接入点信息。

什么是接入点？

接入点是用于建立与Kafka集群初始连接的主机/端口对的列表。

本质上是Kafka集群所在机器的IP/端口信息。

接入点的作用

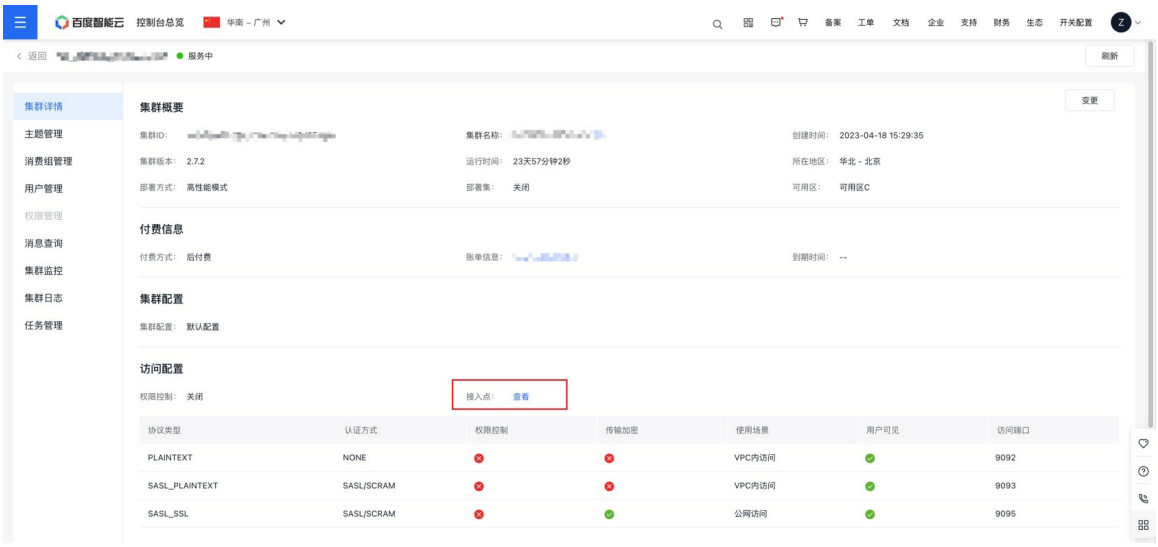
在Kafka生产者或消费者配置中配置此属性。 例如:

```
bootstrap.servers=<接入点地址>
```

如何查看接入点

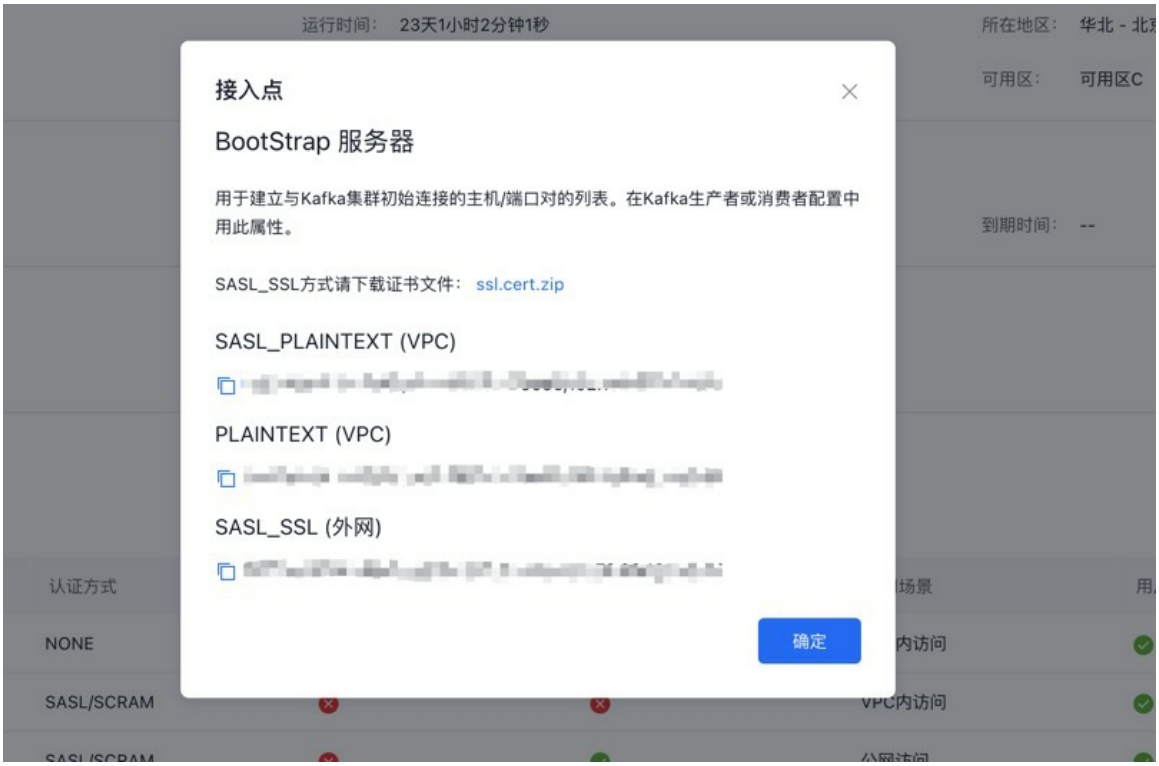
1. 在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

2. 页面跳转后，进入左侧边栏中的『集群详情』页面。



3. 在『集群详情』页面中找到『访问配置』一项。并找到『接入点：查看』链接。

4. 点击接入点后的『查看』链接，得到**接入点**弹窗。（使用SSL、SASL_SSL协议会存在证书文件。下载证书参考：如何下载证书？）



C++示例

VPC网络PLAINTEXT方式生产和消费

在同 VPC 网络下访问，使用 PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装GCC

2. 安装C++ 依赖库。

```
 yum install librdkafka-devel
yum install cyrus-sasl
yum install cyrus-sasl-scram
```

集群准备

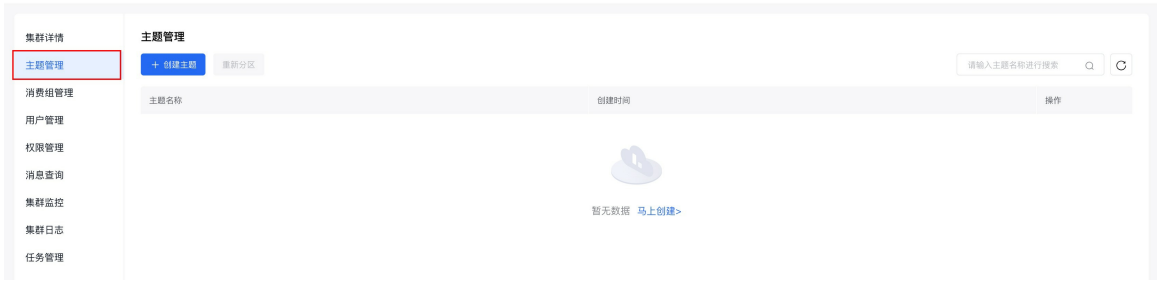
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

生产者代码示例

创建KafkaProducerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 #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);
```

```
  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 *username;      /* Argument: sasl username */
    // const char *password;      /* Argument: sasl password */

    /*
     * Argument validation
     */

     // 检查参数配置
    if (argc != 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic>\n", argv[0]);
        return 1;
    }

    // 接入点信息
    brokers = argv[1];
    // 主题名称
    topic   = argv[2];


    /*
     * 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;
    }

    if (
            rd_kafka_conf_set(conf, "security.protocol", "plaintext", 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;

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

        /* Empty line: only serve delivery reports */
        if (len == 0) {
            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;
}
```

消费者代码示例

创建KafkaConsumerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 //#include <librdkafka/rdkafka.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
     */

    // 检查参数配置
    if (argc < 4) {
        fprintf(stderr, "%% Usage: %s <broker> <group.id> <username> <password> <topic1> <topic2>..\n", argv[0]);
        return 1;
    }

    // 接入点信息
    brokers   = argv[1];
    // 消费组 id
    groupid   = argv[2];
    // 主题名称
    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;
}

if (
        rd_kafka_conf_set(conf, "security.protocol", "plaintext", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
) {
    fprintf(stderr, "%s\n", errstr);
    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;
}

/*
 * 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;
}
```

步骤三：编译并运行

编译并运行上述两个代码文件。

```
 ###### 启动消费者
gcc -lrdkafka ./consumer.c -o consumer
./consumer <broker> <group.id> <topic1> <topic2>..
###### 启动生产者
gcc -lrdkafka ./producer.c -o producer
./producer <broker> <topic>
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端/控制台查看日志。

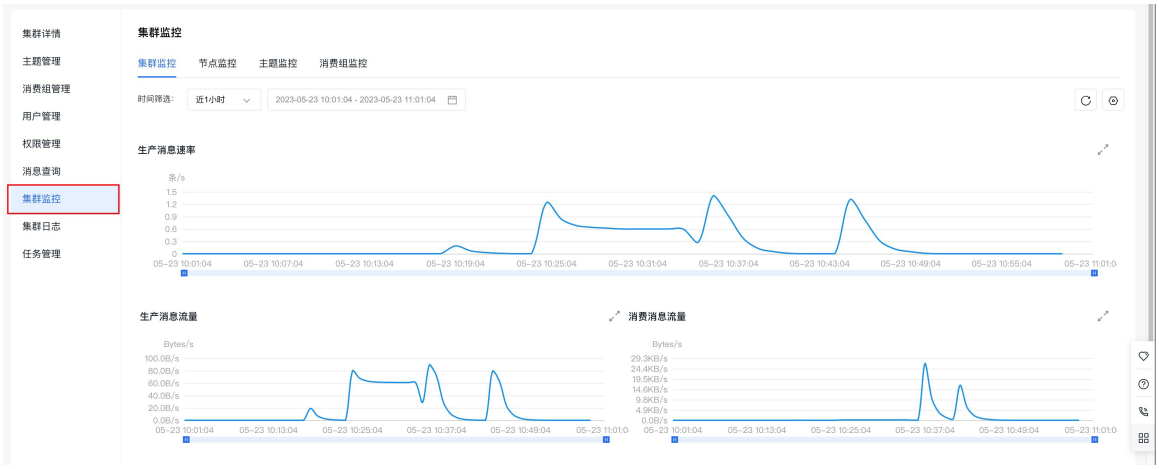
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

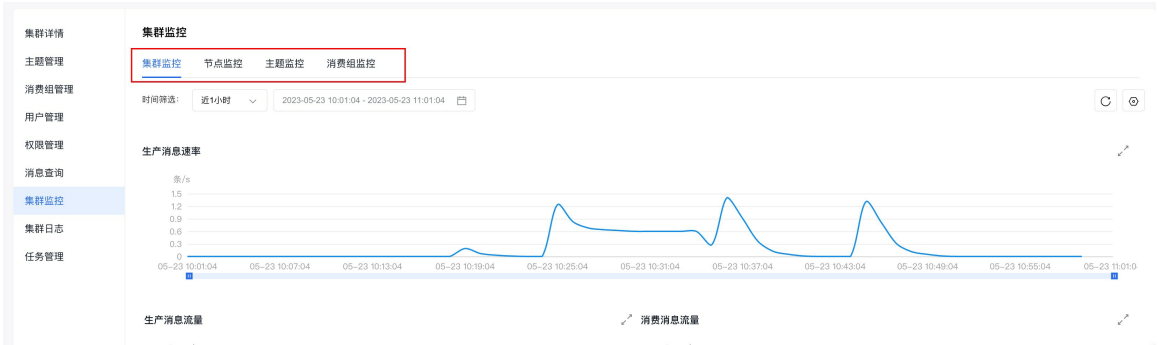
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



VPC网络SASL_PLAINTEXT方式生产和消费消息

在同 VPC 网络下访问，使用 SASL_PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装GCC

2. 安装C++ 依赖库。

```
 yum install librdkafka-devel
yum install cyrus-sasl
yum install cyrus-sasl-scram
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

生产者代码示例

创建KafkaProducerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 #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 *username;      /* Argument: sasl username */
    const char *password;      /* Argument: sasl password */

    /*
     * Argument validation
     */

    // 检查参数配置
    if (argc != 5) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>\n", argv[0]);
        return 1;
    }

    // 接入点信息
    brokers = argv[1];
    // 主题名称
    topic   = argv[2];
    // 用户管理中创建的用户名称
    username = argv[3];
    // 用户管理中创建的用户密码
    password = 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;
    }
    // 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择
    if (
            rd_kafka_conf_set(conf, "security.protocol", "sasl_plaintext", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
            || rd_kafka_conf_set(conf, "sasl.mechanism", "SCRAM-SHA-512", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
            || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
            || rd_kafka_conf_set(conf, "sasl.password", password, 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;

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

    /* Empty line: only serve delivery reports */
    if (len == 0) {
        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;
}
```

消费者代码示例

创建KafkaConsumerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 //#include <librdkafka/rdkafka.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 */
   const char *username;     /* Argument: sasl username */
   const char *password;     /* Argument: sasl password */
   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
    */

   // 检查参数配置
   if (argc < 6) {
      fprintf(stderr, "%% Usage: %s <broker> <group.id> <username> <password> <topic1> <topic2>..\n", argv[0]);
      return 1;
   }

   // 接入点信息
   brokers   = argv[1];
   // 消费组 id
   groupid   = argv[2];
   // 用户管理中创建的用户名称
   username  = argv[3];
   // 用户管理中创建的用户密码
   password  = argv[4];
   // 主题名称
   topics    = &argv[5];
```

```
topic_cnt = argc - 5;


/*
 * 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;
}
// 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择
if (
        rd_kafka_conf_set(conf, "security.protocol", "sasl_plaintext", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.mechanism", "SCRAM-SHA-512", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.password", password, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
) {
    fprintf(stderr, "%s\n", errstr);
    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;
}

/*
 * 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;
}
```

步骤三：编译并运行

编译并运行上述两个代码文件。

```
 ###### 启动消费者
gcc -lrdkafka ./consumer.c -o consumer
./consumer <broker> <group.id> <username> <password> <topic1> <topic2>..
###### 启动生产者
gcc -lrdkafka ./producer.c -o producer
./producer <broker> <topic> <username> <password>
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端/控制台查看日志。

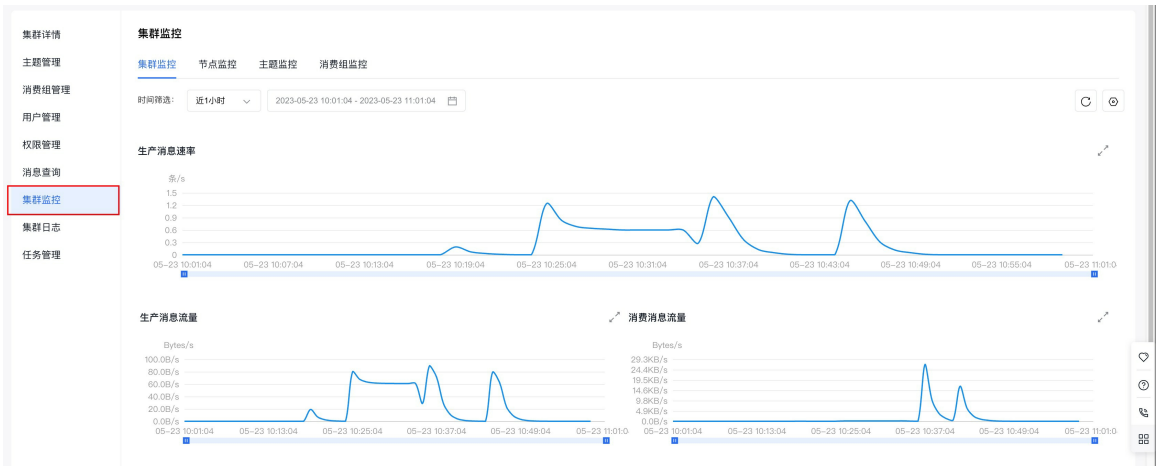
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

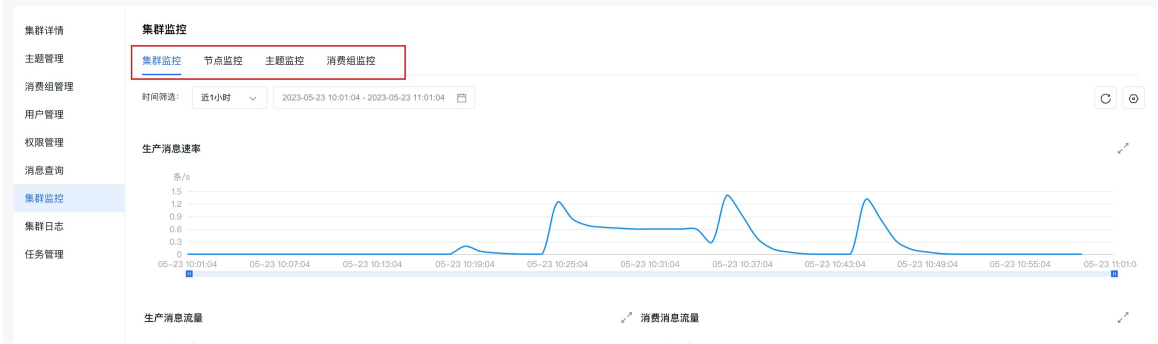
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



公网SASL_SSL方式生产和消费消息

在 Kafka 集群所在 VPC 网络外访问，使用 SASL_SSL 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装GCC

2. 安装C++ 依赖库。

```
 yum install librdkafka-devel
yum install cyrus-sasl
yum install cyrus-sasl-scram
```

集群准备

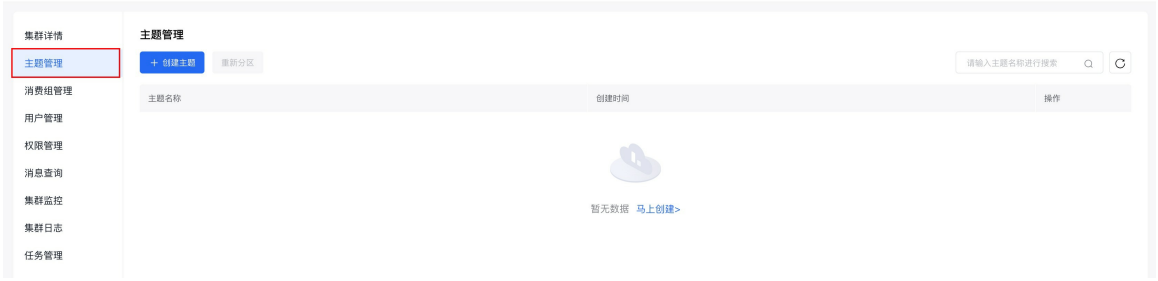
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

生产者代码示例

创建KafkaProducerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 #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 *username;       /* Argument: sasl username */
    const char *password;       /* Argument: sasl password */

    /*
     * Argument validation
     */

    //检查参数配置
    if (argc != 5) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>\n", argv[0]);
        return 1;
    }

    // 接入点信息
    brokers = argv[1];
    // 主题名称
    topic = argv[2];
    // 用户管理中创建的用户名称
    username = argv[3];
    // 用户管理中创建的用户密码
    password = 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;
```

```
}
// 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择
if (
        rd_kafka_conf_set(conf, "ssl.ca.location", "ca.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "security.protocol", "sasl_ssl", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.mechanism", "SCRAM-SHA-512", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.password", password, 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;

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

    /* Empty line: only serve delivery reports */
    if (len == 0) {
        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). */
```

```
            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;
}
```

消费者代码示例

创建KafkaConsumerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 //#include <librdkafka/rdkafka.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 */
    const char *username;     /* Argument: sasl username */
    const char *password;     /* Argument: sasl password */
    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 < 6) {
        fprintf(stderr, "%% Usage: %s <broker> <group.id> <username> <password> <topic1> <topic2>..\n", argv[0]);
        return 1;
    }
```

```
brokers   = argv[1];
groupid   = argv[2];
username  = argv[3];
password  = argv[4];
topics    = &argv[5];

topic_cnt = argc - 5;


/*
 * 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;
}
// 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择
if (
        rd_kafka_conf_set(conf, "ssl.ca.location", "ca.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "security.protocol", "sasl_ssl", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.mechanism", "SCRAM-SHA-512", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "sasl.password", password, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
) {
    fprintf(stderr, "%s\n", errstr);
    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;
}

/*
 * 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;
}
```

步骤四：编译并运行

编译并运行上述两个代码文件。

```
###### 启动消费者
gcc -lrdkafka ./consumer.c -o consumer
./consumer <broker> <group.id> <username> <password> <topic1> <topic2>..
###### 启动生产者
gcc -lrdkafka ./producer.c -o producer
./producer <broker> <topic> <username> <password>
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端/控制台查看日志。

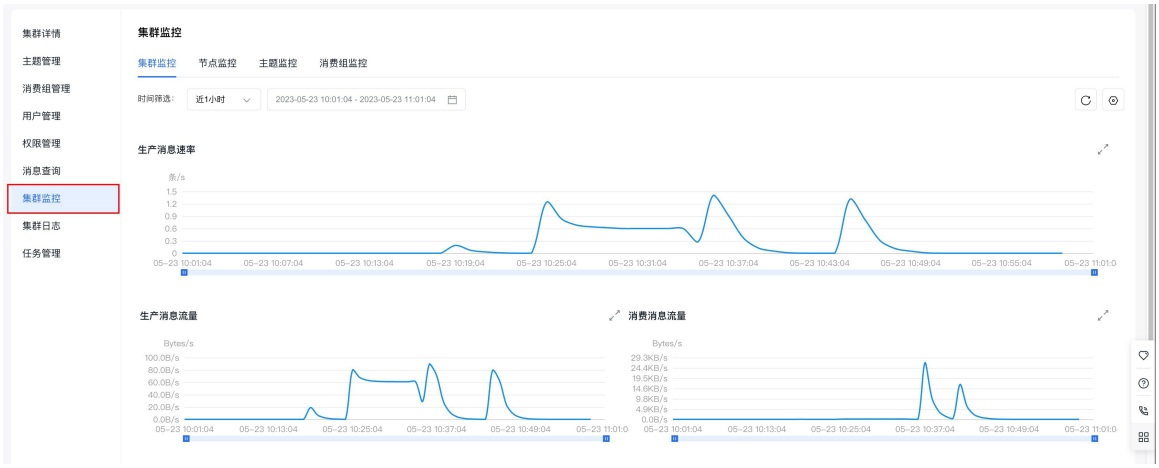
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

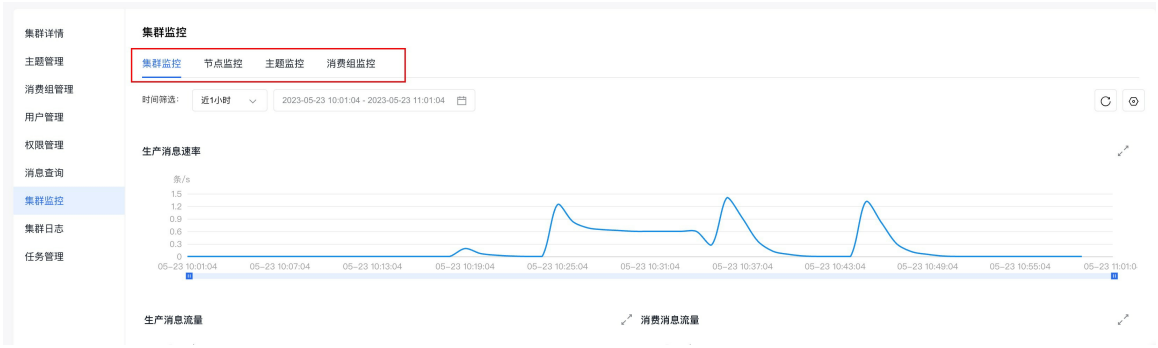
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



SSL协议生产和消费消息

在 Kafka 集群所在 VPC 网络下或者公网环境访问，使用 SSL 协议接入，接入点可以在**集群详情**页面查看。

环境准备

1. 安装GCC

2. 安装C++ 依赖库。

```
yum install librdkafka-devel
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

生产者代码示例

创建KafkaProducerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 #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 */

    /*
     * Argument validation
     */

    //检查参数配置
    if (argc != 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic>\n", argv[0]);
        return 1;
    }

    // 接入点信息
    brokers = argv[1];
    // 主题名称
    topic = argv[2];


    /*
     * 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. */
```

```
 * 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;
}

if (
        rd_kafka_conf_set(conf, "ssl.ca.location", "ca.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "ssl.certificate.location", "client.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "ssl.key.location", "client.key", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "security.protocol", "ssl", 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;

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

    /* Empty line: only serve delivery reports */
    if (len == 0) {
        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;
}
```

消费者代码示例

创建KafkaConsumerDemo.c文件，具体代码示例如下：

```
 /*
* 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>


/* Typical include path would be <librdkafka/rdkafka.h>, but this program
* is builtin from within the librdkafka source tree and thus differs. */
 //#include <librdkafka/rdkafka.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;
}

if (
        rd_kafka_conf_set(conf, "ssl.ca.location", "client.truststore.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "ssl.certificate.location", "client.pem", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "ssl.key.location", "client.key", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
        || rd_kafka_conf_set(conf, "security.protocol", "ssl", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK
) {
    fprintf(stderr, "%s\n", errstr);
    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;
}

/*
 * 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);
```

```
        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;
}
```

步骤四：编译并运行

编译并运行上述两个代码文件。

```
###### 启动消费者
gcc -lrdkafka ./consumer.c -o consumer
./consumer <broker> <group.id> <topic1> <topic2>..
###### 启动生产者
gcc -lrdkafka ./producer.c -o producer
./producer <broker> <topic>
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端/控制台查看日志。

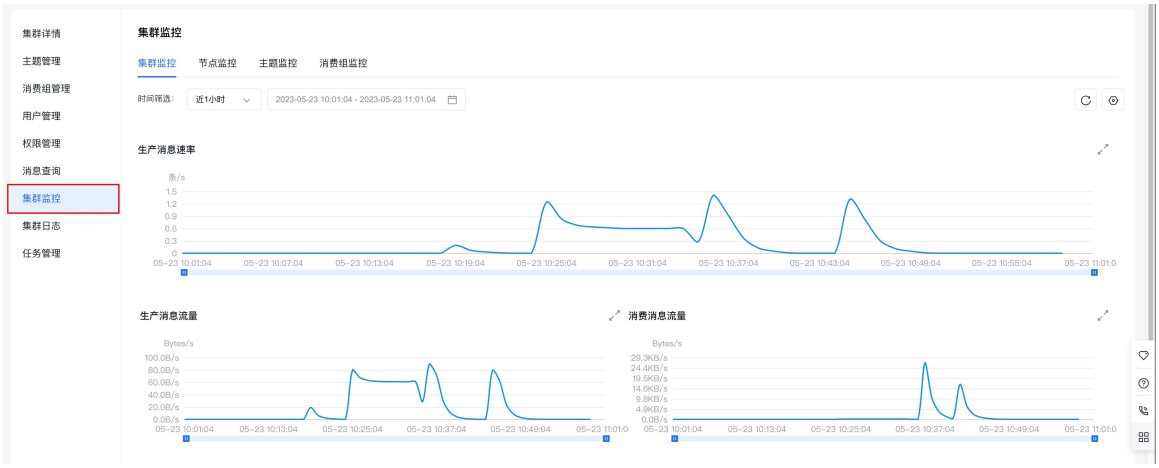
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

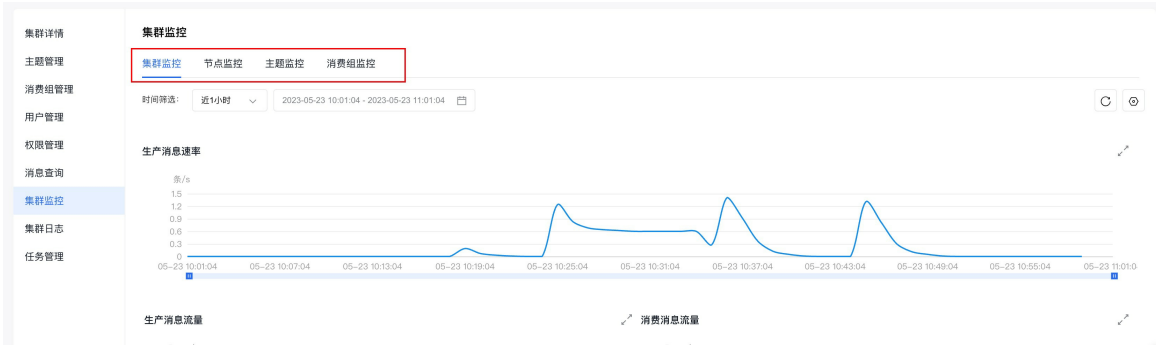
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



PHP示例

VPC网络PLAINTEXT方式生产和消费

在同 VPC 网络下访问，使用 PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装C++依赖库

```
yum install librdkafka-devel
```

1. 安装PHP依赖库

```
pecl install rdkafka
###### 在PHP的初始化文件php.ini中添加以下语句以开启扩展
extension=rdkafka.so
php -m | grep kafka
rdkafka
```

1. 使用php-rdkafka sdk：https://arnaud.le-blanc.net/php-rdkafka-doc/phpdoc/index.html

```
$ git clone https://github.com/arnaud-lb/php-rdkafka.git
$ cd php-rdkafka
$ phpize
$ ./configure
$ make all -j 5
$ sudo make install
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
metadata.broker.list	接入点信息
topic_name	主题名称
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.php文件，具体代码示例如下：

```php
<?php

$conf = new RdKafka\Conf();
// 接入点
$conf->set('metadata.broker.list', '接入点');
// 接入协议
$conf->set('security.protocol','plaintext');


$producer = new RdKafka\Producer($conf);

// 指定topic
$topic = $producer->newTopic("topic_name");

for ($i = 0; $i < 10; $i++) {
    // RD_KAFKA_PARTITION_UA自动选择分区,message指定想要发送的消息
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
    $producer->poll(0);
}

for ($flushRetries = 0; $flushRetries < 10; $flushRetries++) {
    $result = $producer->flush(10000);
    if (RD_KAFKA_RESP_ERR_NO_ERROR === $result) {
        break;
    }
}

if (RD_KAFKA_RESP_ERR_NO_ERROR !== $result) {
    throw new \RuntimeException('Was unable to flush, messages might be lost!');
}
?>
```

消费者代码示例

创建KafkaConsumerDemo.php文件，具体代码示例如下：

```
 <?php

$conf = new RdKafka\Conf();
// 接入协议
$conf->set('security.protocol','plaintext');
// 消费组 id
$conf->set('group.id', 'php-group');

$rk = new RdKafka\Consumer($conf);
// 接入点
$rk->addBrokers("接入点");

$topicConf = new RdKafka\TopicConf();
$topicConf->set('auto.commit.interval.ms', 100);
$topicConf->set('offset.store.method', 'broker');
$topicConf->set('auto.offset.reset', 'earliest');

//订阅topic
$topic = $rk->newTopic("topic_name", $topicConf);

$topic->consumeStart(0, RD_KAFKA_OFFSET_STORED);

while (true) {
    $message = $topic->consume(0, 120*10000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            var_dump($message);
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "No more messages; will wait for more\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "Timed out\n";
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
?>
```

步骤三：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
php consumer.php
###### 启动生产者
php producer.php
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

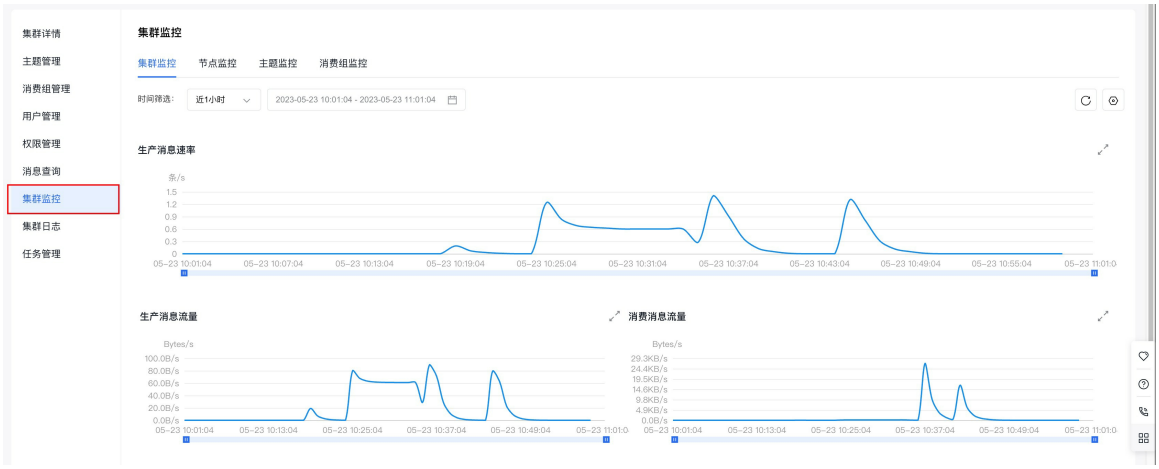
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

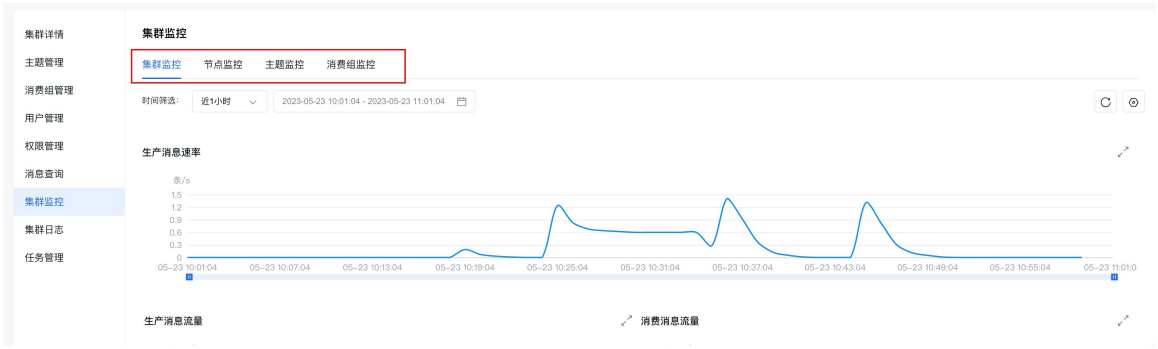
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



VPC网络SASL_PLAINTEXT方式生产和消费消息

在同 VPC 网络下访问，使用 SASL_PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装C++依赖库

```
yum install librdkafka-devel
```

1. 安装PHP依赖库

```
pecl install rdkafka
###### 在PHP的初始化文件php.ini中添加以下语句以开启扩展
extension=rdkafka.so
php -m | grep kafka
rdkafka
```

1. 使用php-rdkafka sdk：https://arnaud.le-blanc.net/php-rdkafka-doc/phpdoc/index.html

```
$ git clone https://github.com/arnaud-lb/php-rdkafka.git
$ cd php-rdkafka
$ phpize
$ ./configure
$ make all -j 5
$ sudo make install
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
metadata.broker.list	接入点信息
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
topic_name	主题名称
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.php文件，具体代码示例如下：

```
 <?php

$conf = new RdKafka\Conf();
// 接入点
$conf->set('metadata.broker.list', '接入点');
// 接入协议
$conf->set('security.protocol','sasl_plaintext');
// SASL 机制
$conf->set('sasl.mechanism','SCRAM-SHA-512');
// SASL 用户名
$conf->set('sasl.username','alice');
// SASL 密码
$conf->set('sasl.password','alice1234!');

$producer = new RdKafka\Producer($conf);

// 指定topic
$topic = $producer->newTopic("topic_name");

for ($i = 0; $i < 10; $i++) {
    // RD_KAFKA_PARTITION_UA自动选择分区,message指定想要发送的消息
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i : php sasl_plaintext");
    $producer->poll(0);
}

for ($flushRetries = 0; $flushRetries < 10; $flushRetries++) {
    $result = $producer->flush(10000);
    if (RD_KAFKA_RESP_ERR_NO_ERROR === $result) {
        break;
    }
}

if (RD_KAFKA_RESP_ERR_NO_ERROR !== $result) {
    throw new \RuntimeException('Was unable to flush, messages might be lost!');
}
?>
```

消费者代码示例

创建KafkaConsumerDemo.php文件，具体代码示例如下：

```php
<?php

$conf = new RdKafka\Conf();
// 接入协议
$conf->set('security.protocol','sasl_plaintext');
// SASL 机制
$conf->set('sasl.mechanism','SCRAM-SHA-512');
// SASL 用户名
$conf->set('sasl.username','alice');
// SASL 密码
$conf->set('sasl.password','alice1234!');
// 消费组 id
$conf->set('group.id', 'php-group');

$rk = new RdKafka\Consumer($conf);
// 接入点
$rk->addBrokers("接入点");

$topicConf = new RdKafka\TopicConf();
$topicConf->set('auto.commit.interval.ms', 100);
$topicConf->set('offset.store.method', 'broker');
$topicConf->set('auto.offset.reset', 'earliest');

//订阅topic
$topic = $rk->newTopic("topic_name", $topicConf);

$topic->consumeStart(0, RD_KAFKA_OFFSET_STORED);

while (true) {
    $message = $topic->consume(0, 120*10000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            var_dump($message);
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "No more messages; will wait for more\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "Timed out\n";
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
?>
```

步骤三：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
php consumer.php
###### 启动生产者
php producer.php
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

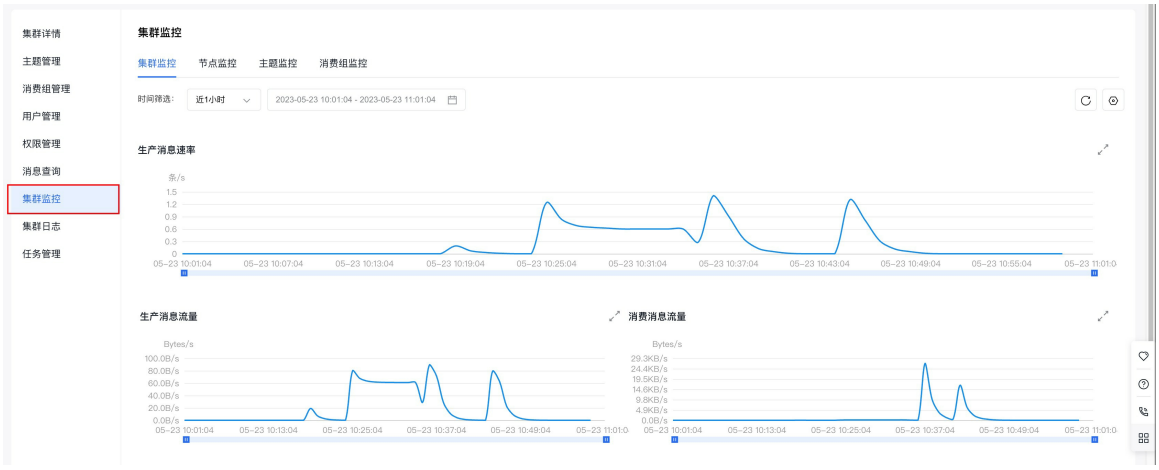
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

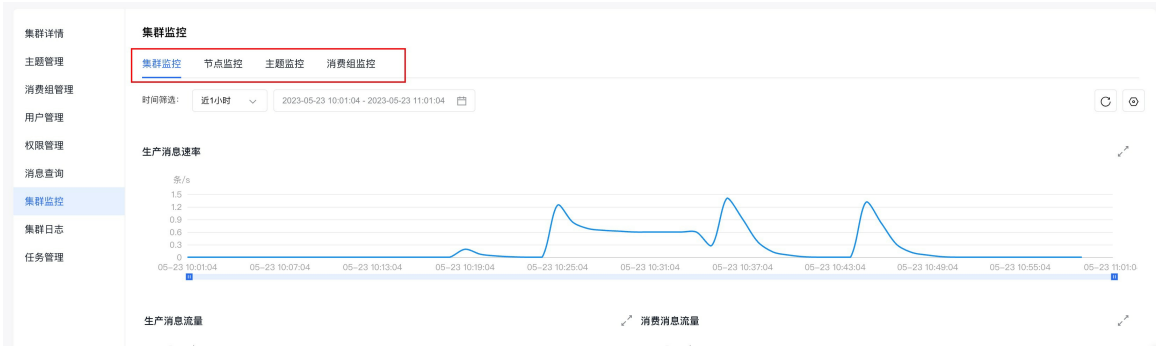
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



公网SASL_SSL方式生产和消费消息

在 Kafka 集群所在 VPC 网络外访问，使用 SASL_SSL 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装C++依赖库

```
yum install librdkafka-devel
```

1. 安装PHP依赖库

```
pecl install rdkafka
###### 在PHP的初始化文件php.ini中添加以下语句以开启扩展
extension=rdkafka.so
php -m | grep kafka
rdkafka
```

1. 使用php-rdkafka sdk：https://arnaud.le-blanc.net/php-rdkafka-doc/phpdoc/index.html

```
$ git clone https://github.com/arnaud-lb/php-rdkafka.git
$ cd php-rdkafka
$ phpize
$ ./configure
$ make all -j 5
$ sudo make install
```

集群准备

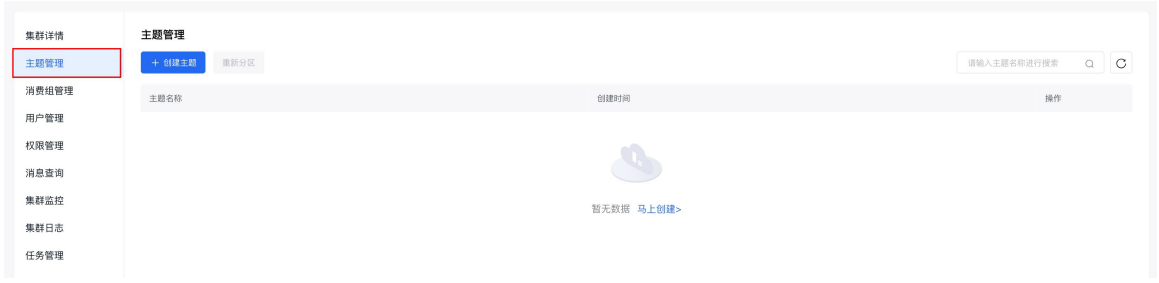
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
metadata.broker.list	接入点信息
ssl.ca.location	ca.pem证书文件所在路径
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
topic_name	主题名称
group_id	消费组id
sasl.mechanism	SCRAM-SHA-512或者PLAIN

生产者代码示例

创建KafkaProducerDemo.php文件，具体代码示例如下：

```php
<?php

$conf = new RdKafka\Conf();
// 接入点
$conf->set('metadata.broker.list', '接入点');
// 接入协议
$conf->set('security.protocol','sasl_ssl');
// 证书文件路径（证书文件请参考"接入点查看"文档）
$conf->set('ssl.ca.location',__DIR__.'/ssl.cert/ca.pem');
// SASL 机制
$conf->set('sasl.mechanism','SCRAM-SHA-512');
// SASL 用户名
$conf->set('sasl.username','alice');
// SASL 密码
$conf->set('sasl.password','alice1234!');

$producer = new RdKafka\Producer($conf);

// 指定topic
$topic = $producer->newTopic("topic_name");

for ($i = 0; $i < 10; $i++) {
    // RD_KAFKA_PARTITION_UA自动选择分区，message指定想要发送的消息
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i : php sasl_ssl");
    $producer->poll(0);
}

for ($flushRetries = 0; $flushRetries < 10; $flushRetries++) {
    $result = $producer->flush(10000);
    if (RD_KAFKA_RESP_ERR_NO_ERROR === $result) {
        break;
    }
}

if (RD_KAFKA_RESP_ERR_NO_ERROR !== $result) {
    throw new \RuntimeException('Was unable to flush, messages might be lost!');
}
```

消费者代码示例

创建KafkaConsumerDemo.php文件，具体代码示例如下：

```
<?php

$conf = new RdKafka\Conf();
// 接入协议
$conf->set('security.protocol','sasl_ssl');
// SASL 机制
$conf->set('sasl.mechanism','SCRAM-SHA-512');
// SASL 用户名
$conf->set('sasl.username','alice');
// SASL 密码
$conf->set('sasl.password','alice1234!');
// 证书文件路径
$conf->set('ssl.ca.location',__DIR__.'/ssl.cert/ca.pem');
// 消费组 id
$conf->set('group.id', 'php-group');

$rk = new RdKafka\Consumer($conf);
// 接入点
$rk->addBrokers("接入点");

$topicConf = new RdKafka\TopicConf();
$topicConf->set('auto.commit.interval.ms', 100);
$topicConf->set('offset.store.method', 'broker');
$topicConf->set('auto.offset.reset', 'earliest');

//订阅topic
$topic = $rk->newTopic("topic_name", $topicConf);

$topic->consumeStart(0, RD_KAFKA_OFFSET_STORED);

while (true) {
    $message = $topic->consume(0, 120*10000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            var_dump($message);
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "No more messages; will wait for more\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "Timed out\n";
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
```

步骤四：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
php consumer.php
###### 启动生产者
php producer.php
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

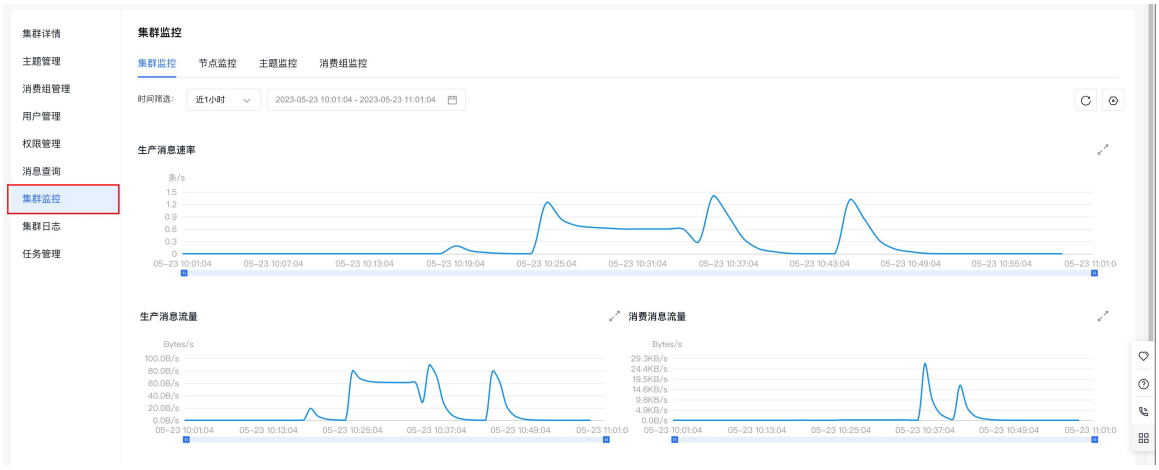
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

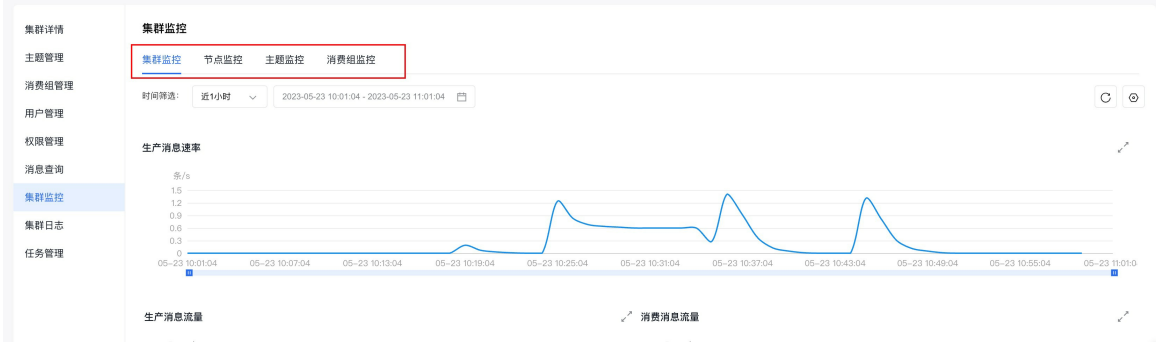
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



SSL协议生产和消费消息

在 Kafka 集群所在 VPC 网络下或者公网环境访问，使用 SSL 协议接入，接入点可以在**集群详情**页面查看。

环境准备

1. 安装C++依赖库

```
yum install librdkafka-devel
```

1. 安装PHP依赖库

```
pecl install rdkafka
###### 在PHP的初始化文件php.ini中添加以下语句以开启扩展
extension=rdkafka.so
php -m | grep kafka
rdkafka
```

1. 使用php-rdkafka sdk：https://arnaud.le-blanc.net/php-rdkafka-doc/phpdoc/index.html

```
$ git clone https://github.com/arnaud-lb/php-rdkafka.git
$ cd php-rdkafka
$ phpize
$ ./configure
$ make all -j 5
$ sudo make install
```

集群准备

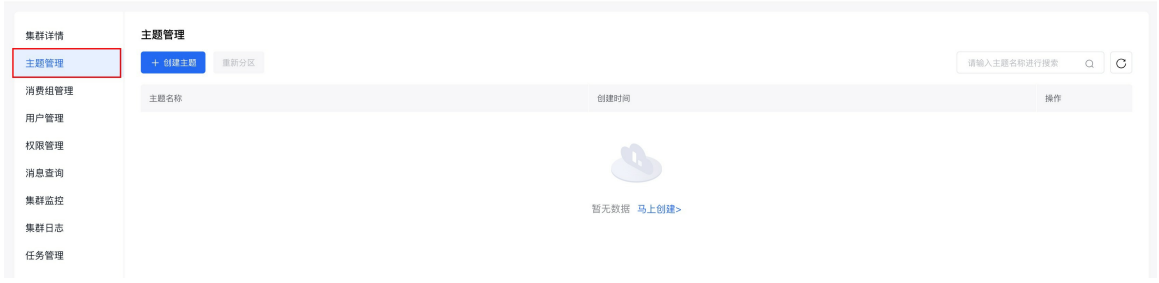
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
metadata.broker.list	接入点信息
ssl.ca.location	ca.pem文件所在路径
ssl.certificate.location	client.pem文件所在路径
ssl.key.location	client.key文件所在路径
topic_name	主题名称
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.php文件，具体代码示例如下：

```php
<?php

$conf = new RdKafka\Conf();
// 接入点
$conf->set('metadata.broker.list', '接入点');
// 接入协议
$conf->set('security.protocol','ssl');
// 证书文件路径（证书文件请参考"接入点查看"文档）
$conf->set('ssl.ca.location',__DIR__.'/ssl.cert/ca.pem');
$conf->set('ssl.certificate.location',__DIR__.'/ssl.cert/client.pem');
$conf->set('ssl.key.location ',__DIR__.'/ssl.cert/client.key');

$producer = new RdKafka\Producer($conf);

// 指定topic
$topic = $producer->newTopic("topic_name");

for ($i = 0; $i < 10; $i++) {
    // RD_KAFKA_PARTITION_UA自动选择分区，message指定想要发送的消息
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i : php ssl");
    $producer->poll(0);
}

for ($flushRetries = 0; $flushRetries < 10; $flushRetries++) {
    $result = $producer->flush(10000);
    if (RD_KAFKA_RESP_ERR_NO_ERROR === $result) {
        break;
    }
}

if (RD_KAFKA_RESP_ERR_NO_ERROR !== $result) {
    throw new \RuntimeException('Was unable to flush, messages might be lost!');
}
```

消费者代码示例

创建KafkaConsumerDemo.php文件，具体代码示例如下：

```php
<?php

$conf = new RdKafka\Conf();
// 接入协议
$conf->set('security.protocol','ssl');
// 证书文件路径
$conf->set('ssl.ca.location',__DIR__.'/ssl.cert/ca.pem');
$conf->set('ssl.certificate.location',__DIR__.'/ssl.cert/client.pem');
$conf->set('ssl.key.location ',__DIR__.'/ssl.cert/client.key');
// 消费组 id
$conf->set('group.id', 'php-group');

$rk = new RdKafka\Consumer($conf);
// 接入点
$rk->addBrokers("接入点");

$topicConf = new RdKafka\TopicConf();
$topicConf->set('auto.commit.interval.ms', 100);
$topicConf->set('offset.store.method', 'broker');
$topicConf->set('auto.offset.reset', 'earliest');

//订阅topic
$topic = $rk->newTopic("topic_name", $topicConf);

$topic->consumeStart(0, RD_KAFKA_OFFSET_STORED);

while (true) {
    $message = $topic->consume(0, 120*10000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            var_dump($message);
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "No more messages; will wait for more\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "Timed out\n";
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
```

步骤四：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
php consumer.php
###### 启动生产者
php producer.php
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

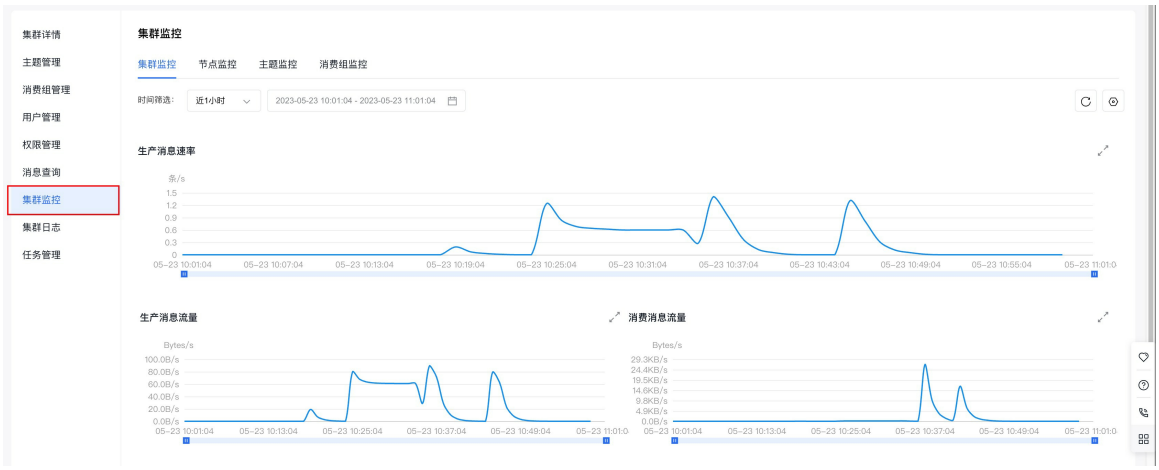
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

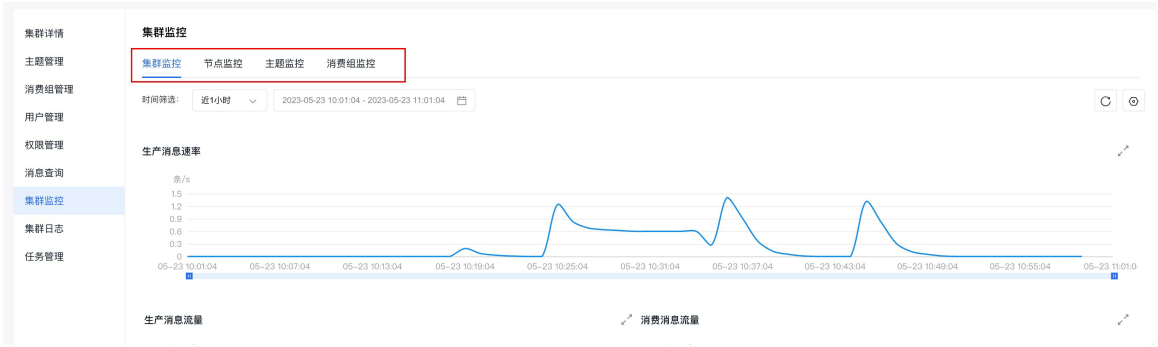
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



Java示例

VPC网络PLAINTEXT方式生产和消费

在同 VPC 网络下访问，使用 PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装1.8或以上版本 JDK

2. 安装2.5或以上版本 Maven

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：添加Maven配置

```
    <dependencies>
      <dependency>
         <groupId>org.apache.kafka</groupId>
         <artifactId>kafka-clients</artifactId>
         <version>2.7.2</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-assembly-plugin -->
      <dependency>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-assembly-plugin</artifactId>
         <version>3.5.0</version>
      </dependency>
   </dependencies>
```

步骤三：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
access_point	接入点信息
topic	主题名称
message	消息的具体内容
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.java文件，具体代码示例如下：

```
 package org.example.Java示例.PLAINTEXT;

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 KafkaProducerDemo {
    public static void main(String[] args) {

        // 需要自行配置下面三个参数
        // 接入点-PLAINTEXT
        String access_point = "192.168.32.10:9092,192.168.32.12:9092,192.168.32.11:9092";
        // 主题名称-topic name
        String topic = "test";
        // 消息内容
        String message = "kafka java test";

        // 创建配置类
        Properties properties = new Properties();
        // 设置接入点
        properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, access_point);
        // Kafka消息的序列化方式。
        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);
```

```
        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);

        // 构建kafkaProducer对象
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<>(properties);

        try {
            // 向指定的topic发送100条消息
            for (int i = 0; i < 100; i++) {
                // 通过 ProducerRecord 构造一个消息对象
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic, message + "-" + i);
                // 通过kafkaProducer发送消息
                kafkaProducer.send(kafkaMessage, (RecordMetadata recordMetadata, Exception e) -> {
                    // 发送信息后的回调函数，用以验证消息是否发送成功
                    if (e == null) {
                        System.out.println("send success:" + recordMetadata.toString());
                    } else {
                        e.printStackTrace();
                        System.err.println("send failed");
                    }
                });
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            // 不要忘记关闭资源
            kafkaProducer.close();
        }
    }
}
```

消费者代码示例

创建KafkaConsumerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.PLAINTEXT;

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

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

public class KafkaConsumerDemo {
    public static void main(String[] args) {

        // 需要自行配置下面三个参数
        // 接入点-PLAINTEXT
        String access_point = "192.168.32.10:9092,192.168.32.12:9092,192.168.32.11:9092";
        // 主题名称-topic name
        String topic = "test";
        // 消费组id
        String group_id = "test_group";
```

```
        // 创建配置类
        Properties properties = new Properties();
        // 设置接入点
        properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, access_point);
        // Kafka消息的序列化方式
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        // 指定消费组id
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, group_id);
        // enable.auto.commit如果为true，则消费者的偏移量将定期在后台提交。
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        // 重置消费位点策略:earliest、latest、none
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        // 设置kafka自动提交offset的频率，默认5000ms，也就是5s
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 1000);
        // 设置消费者在一次poll中返回的最大记录数
        properties.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 1000);
        // 设置消费者两次poll的最大时间间隔
        properties.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 1000);

        // 构建KafkaConsumer对象
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);

        //订阅主题
        kafkaConsumer.subscribe(Collections.singleton(topic));

        try{
            //持续消费主题中的消息
            while(true){
                // 构建ConsumerRecord用于接收存储消息
                ConsumerRecords<String, String> kafkaMessage = kafkaConsumer.poll(Duration.ofMillis(5000));
                for (ConsumerRecord<String, String> consumerRecord : kafkaMessage) {
                    // 打印消息具体内容
                    System.out.printf("offset = %d, key = %s, value = %s%n", consumerRecord.offset(), consumerRecord.key(),
consumerRecord.value());
                }
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            kafkaConsumer.close();
        }
    }
}
```

步骤四：编译并运行

编译并运行上述两个代码文件。

- 先启动KafkaConsumerDemo.java
- 再启动KafkaProducerDemo.java

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看Java程序输出日志。

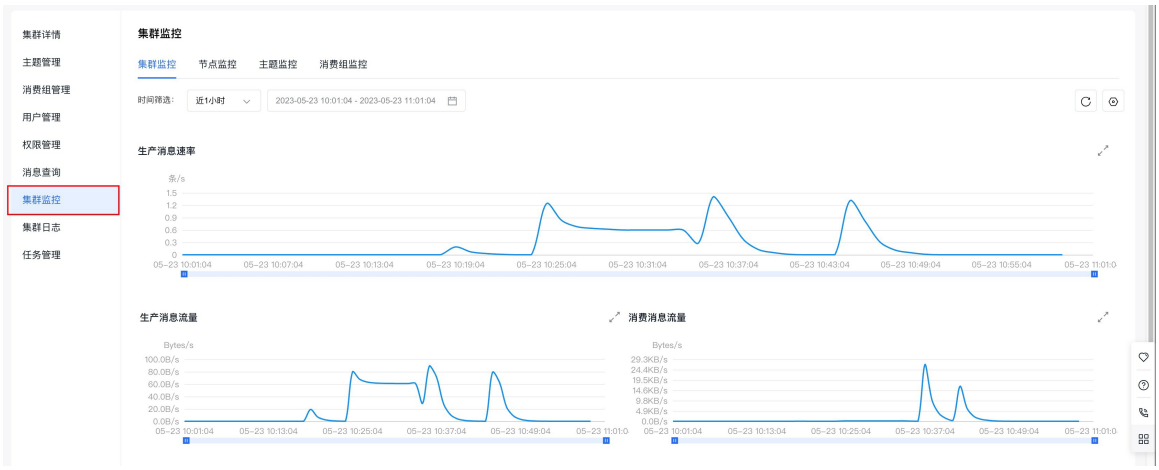
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

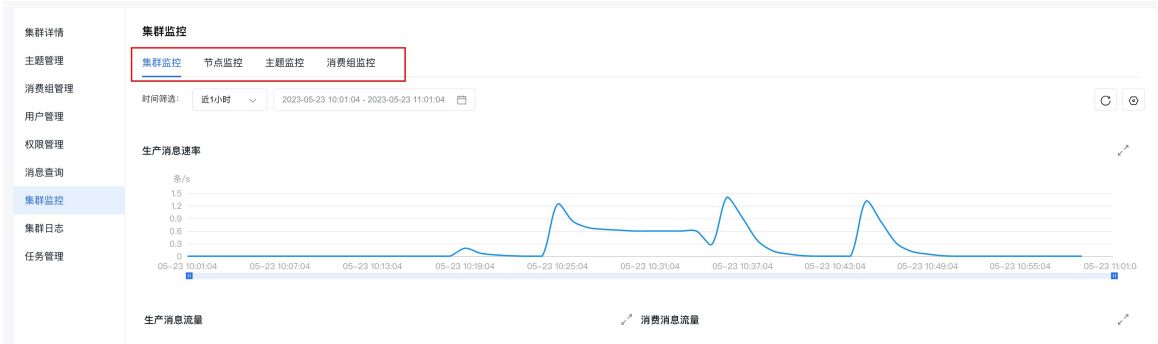
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



VPC网络SASL_PLAINTEXT方式生产和消费消息

在同 VPC 网络下访问，使用 SASL_PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装1.8或以上版本 JDK

2. 安装2.5或以上版本 Maven

集群准备

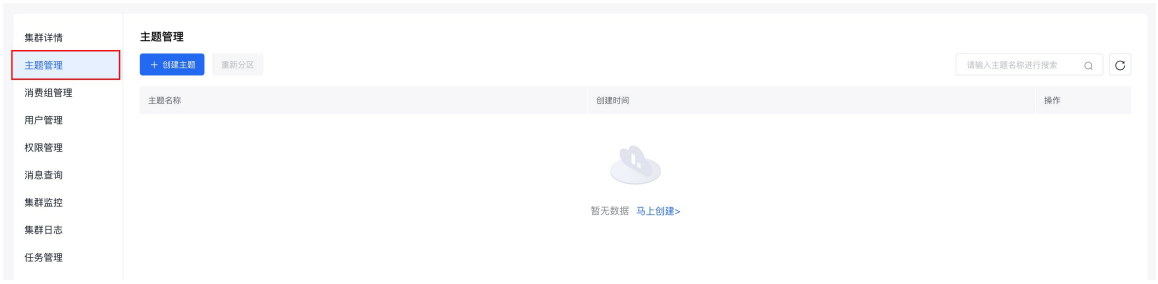
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：添加Maven配置

```
<dependencies>
  <dependency>
     <groupId>org.apache.kafka</groupId>
     <artifactId>kafka-clients</artifactId>
     <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-assembly-plugin -->
  <dependency>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-assembly-plugin</artifactId>
     <version>3.5.0</version>
  </dependency>
</dependencies>
```

步骤三：创建JAAS 配置文件

创建 jaas 配置文件 kafka_client_jaas.conf，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择。

SCRAM-SHA-512：

```
KafkaClient {
  org.apache.kafka.common.security.scram.ScramLoginModule required
  username="{username}"
  password="{password}";
};
```

PLAIN：

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

username，password 填创建用户时设置的值。

步骤四：创建kafka.properties配置文件

提供接入 Kafka 服务需要的配置信息，配置项如下：

- bootstrap.servers 配置为接入点地址，具体请参考接入点查看。
- security.protocol 固定为 SASL_PLAINTEXT
- sasl.mechanism 固定为 SCRAM-SHA-512或者PLAIN，根据集群所使用的认证方式进行选择
- java.security.auth.login.config 配置为 kafka_client_jaas.conf 路径

```
bootstrap.servers=<接入点地址>

security.protocol=SASL_PLAINTEXT

sasl.mechanism=SCRAM-SHA-512
###### sasl.mechanism=PLAIN

java.security.auth.login.config=/etc/kafka/kafka_client_jaas.conf
```

kafka.properties 可以作为 resources 文件与代码一起打包，代码运行时会在 classpath 中寻找 kafka.properties 文件资源；也可以在运行代码时将 kafka.properties 置于进程工作目录同级的 config 目录下。

步骤五：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
path	接入点信息kafka.properties所在路径（建议写文件所在的绝对路径）
topic	主题名称
message	消息的具体内容
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.SASL_PLAINTEXT;

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.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test";
        // 消息内容
        String message = "kafka java test";

        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // 设置 java.security.auth.login.config，用于加载kafka_client_jaas.conf文件
        if (null == System.getProperty("java.security.auth.login.config")) {
            System.setProperty(
                    "java.security.auth.login.config",
                    properties.getProperty("java.security.auth.login.config")
            );
        }

        // Kafka消息的序列化方式。
        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);

        // 构建kafkaProducer对象
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<>(properties);

        try {
            // 向指定的topic发送100条消息
            for (int i = 0; i < 100; i++) {
                // 通过 ProducerRecord 构造一个消息对象
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic, message + "-" + i);
                // 通过kafkaProducer发送消息
                kafkaProducer.send(kafkaMessage, (RecordMetadata recordMetadata, Exception e) -> {
                    // 发送信息后的回调函数，用以验证消息是否发送成功
                    if (e == null) {
                        System.out.println("send success:" + recordMetadata.toString());
                    } else {
                        e.printStackTrace();
                        System.err.println("send failed");
                    }
                });
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            // 不要忘记关闭资源
            kafkaProducer.close();
        }
    }
}
```

消费者代码示例

创建KafkaConsumerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.SASL_PLAINTEXT;

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

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test";
        // 消费组id
        String group_id = "test_group";

        //创建配置类，并获取配置文件kafka.properties的内容
```

```
        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // 设置 java.security.auth.login.config，用于加载kafka_client_jaas.conf文件
        if (null == System.getProperty("java.security.auth.login.config")) {
            System.setProperty(
                    "java.security.auth.login.config",
                    properties.getProperty("java.security.auth.login.config")
            );
        }

        // Kafka消息的序列化方式
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        // 指定消费组id
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, group_id);
        // enable.auto.commit如果为true，则消费者的偏移量将定期在后台提交。
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        // 重置消费位点策略:earliest、latest、none
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        // 设置kafka自动提交offset的频率，默认5000ms，也就是5s
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 1000);
        // 设置消费者在一次poll中返回的最大记录数
        properties.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 1000);
        // 设置消费者两次poll的最大时间间隔
        properties.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 1000);

        // 构建KafkaConsumer对象
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);

        //订阅主题
        kafkaConsumer.subscribe(Collections.singleton(topic));

        try{
            //持续消费主题中的消息
            while(true){
                // 构建ConsumerRecord用于接收存储消息
                ConsumerRecords<String, String> kafkaMessage = kafkaConsumer.poll(Duration.ofMillis(5000));
                for (ConsumerRecord<String, String> consumerRecord : kafkaMessage) {
                    // 打印消息具体内容
                    System.out.printf("offset = %d, key = %s, value = %s%n", consumerRecord.offset(), consumerRecord.key(),
consumerRecord.value());
                }
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
```

```
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            kafkaConsumer.close();
        }
    }
}
```

步骤六：编译并运行

编译并运行上述两个代码文件。

- 先启动KafkaConsumerDemo.java
- 再启动KafkaProducerDemo.java

步骤七：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

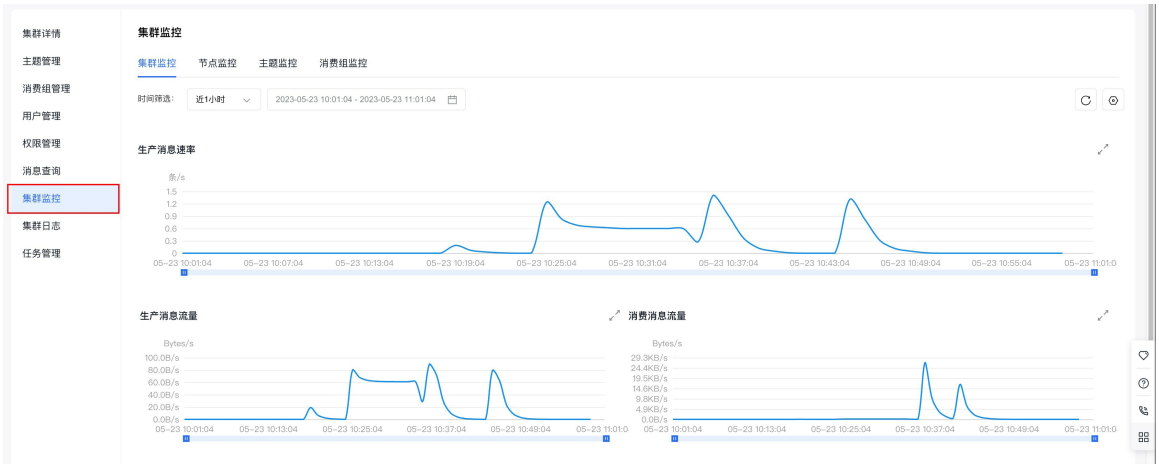
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

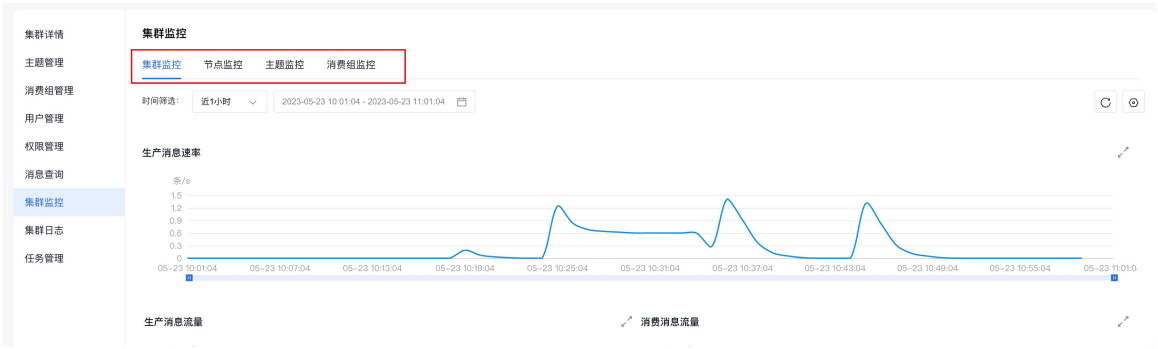
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



公网SASL_SSL方式生产和消费消息

在 Kafka 集群所在 VPC 网络外访问，使用 SASL_SSL 协议接入，接入点可以在**集群详情** 页面查看。

环境准备

1. 安装1.8或以上版本 JDK

2. 安装2.5或以上版本 Maven

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：添加Maven配置

```
<dependencies>
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-assembly-plugin -->
  <dependency>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.5.0</version>
  </dependency>
</dependencies>
```

步骤三：创建JAAS 配置文件

创建 jaas 配置文件 kafka_client_jaas.conf，认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择。

SCRAM-SHA-512：

```
KafkaClient {
  org.apache.kafka.common.security.scram.ScramLoginModule required
  username="{username}"
  password="{password}";
};
```

PLAIN：

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

username，password 填创建用户时设置的值。

步骤四：准备证书文件

下载证书文件：如何下载证书？

步骤五：创建kafka.properties配置文件

提供接入 Kafka 服务需要的配置信息，配置项如下：

- bootstrap.servers 配置为接入点地址，具体请参考接入点查看。
- security.protocol 固定为 SASL_SSL
- ssl.truststore.location 配置为 client.truststore.jks 文件路径
- ssl.truststore.password 固定为 bms@kafka
- ssl.endpoint.identification.algorithm 固定为空

- sasl.mechanism 固定为 SCRAM-SHA-512或者PLAIN，根据集群所使用的认证方式进行选择
- java.security.auth.login.config 配置为 kafka_client_jaas.conf 文件路径

```
 bootstrap.servers=<接入点地址>

security.protocol=SASL_SSL

ssl.truststore.location=/etc/kafka/ssl/client.truststore.jks
ssl.truststore.password=bms@kafka
ssl.endpoint.identification.algorithm=
ssl.enabled.protocols = TLSv1.2,TLSv1.1,TLSv1
ssl.protocol = TLSv1.2

sasl.mechanism=SCRAM-SHA-512
###### sasl.mechanism=PLAIN

java.security.auth.login.config=/etc/kafka/kafka_client_jaas.conf
```

kafka.properties 可以作为 resources 文件与代码一起打包，代码运行时会在 classpath 中寻找 kafka.properties 文件资源；也可以在运行代码时将 kafka.properties 置于进程工作目录同级的 config 目录下。

步骤六：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
path	接入点信息kafka.properties所在路径（建议写文件所在的绝对路径）
topic	主题名称
message	消息的具体内容
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.java文件，具体代码示例如下：

```
 package org.example.Java示例.SASL_SSL;

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.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test_sasl_ssl";
        // 消息内容
        String message = "kafka java test";
```

```
        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // 设置 java.security.auth.login.config，用于加载kafka_client_jaas.conf文件
        if (null == System.getProperty("java.security.auth.login.config")) {
            System.setProperty(
                    "java.security.auth.login.config",
                    properties.getProperty("java.security.auth.login.config")
            );
        }

        // Kafka消息的序列化方式。
        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);

        // 构建kafkaProducer对象
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<>(properties);

        try {
            // 向指定的topic发送100条消息
            for (int i = 0; i < 100; i++) {
                // 通过 ProducerRecord 构造一个消息对象
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic, message + "-" + i);
                // 通过kafkaProducer发送消息
                kafkaProducer.send(kafkaMessage, (RecordMetadata recordMetadata, Exception e) -> {
                    // 发送信息后的回调函数，用以验证消息是否发送成功
                    if (e == null) {
                        System.out.println("send success:" + recordMetadata.toString());
                    } else {
                        e.printStackTrace();
                        System.err.println("send failed");
                    }
                });
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            // 不要忘记关闭资源
```

```
            kafkaProducer.close();
        }
    }
}
```

消费者代码示例

创建KafkaConsumerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.SASL_SSL;

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

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test_sasl_ssl";
        // 消费组id
        String group_id = "test_group";

        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // 设置 java.security.auth.login.config，用于加载kafka_client_jaas.conf文件
        if (null == System.getProperty("java.security.auth.login.config")) {
            System.setProperty(
                    "java.security.auth.login.config",
                    properties.getProperty("java.security.auth.login.config")
            );
        }

        // Kafka消息的序列化方式
```

```
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        // 指定消费组id
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, group_id);
        // enable.auto.commit如果为true，则消费者的偏移量将定期在后台提交。
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        // 重置消费位点策略:earliest、latest、none
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        // 设置kafka自动提交offset的频率，默认5000ms，也就是5s
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 1000);
        // 设置消费者在一次poll中返回的最大记录数
        properties.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 1000);
        // 设置消费者两次poll的最大时间间隔
        properties.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 1000);

        // 构建KafkaConsumer对象
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);

        //订阅主题
        kafkaConsumer.subscribe(Collections.singleton(topic));

        try{
            //持续消费主题中的消息
            while(true){
                // 构建ConsumerRecord用于接收存储消息
                ConsumerRecords<String, String> kafkaMessage = kafkaConsumer.poll(Duration.ofMillis(5000));
                for (ConsumerRecord<String, String> consumerRecord : kafkaMessage) {
                    // 打印消息具体内容
                    System.out.printf("offset = %d, key = %s, value = %s%n", consumerRecord.offset(), consumerRecord.key(),
consumerRecord.value());
                }
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            kafkaConsumer.close();
        }
    }
}
```

步骤七：编译并运行

编译并运行上述两个代码文件。

- 先启动KafkaConsumerDemo.java
- 再启动KafkaProducerDemo.java

步骤八：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

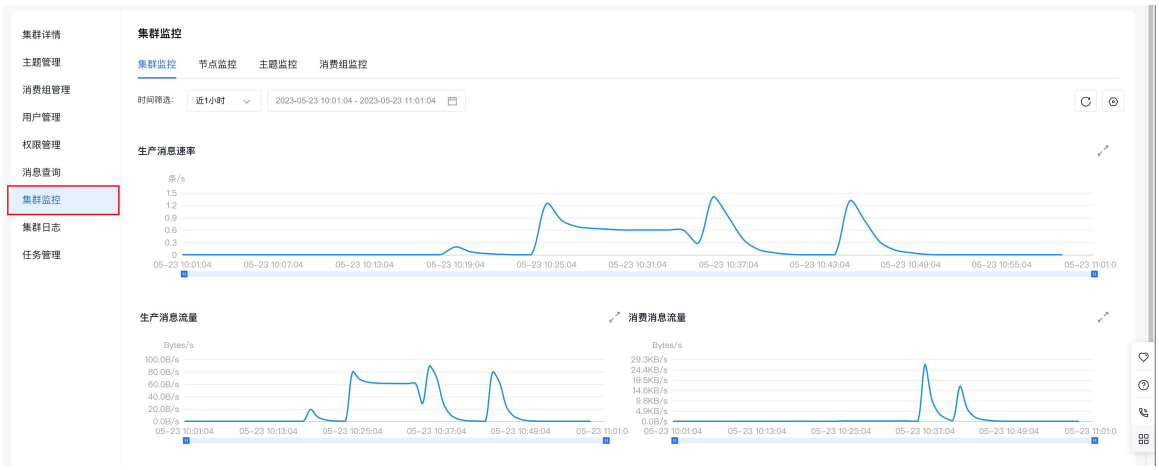
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

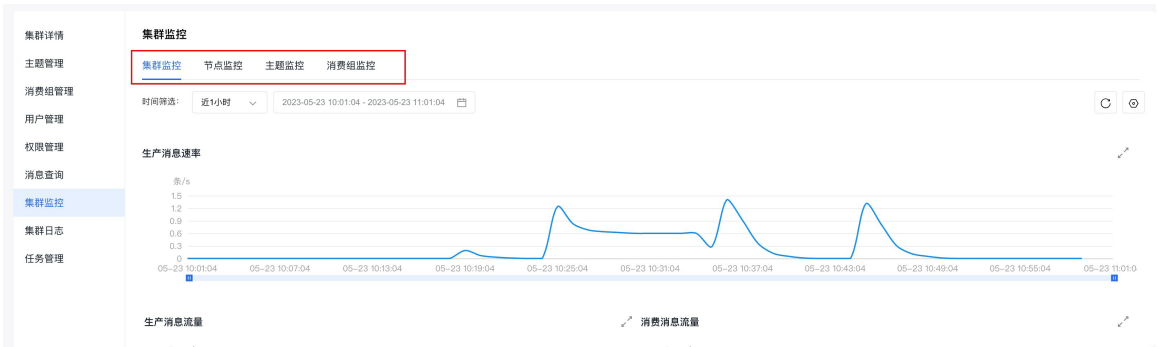
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



SSL协议生产和消费消息

在 Kafka 集群所在 VPC 网络下或者公网环境访问，使用 SSL 协议接入，接入点可以在**集群详情**页面查看。

环境准备

1. 安装1.8或以上版本 JDK
2. 安装2.5或以上版本 Maven

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：添加Maven配置

```
<dependencies>
  <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-assembly-plugin -->
  <dependency>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.5.0</version>
  </dependency>
</dependencies>
```

步骤三：准备证书文件

下载证书文件：如何下载证书？

步骤四：创建kafka.properties配置文件

提供接入 Kafka 服务需要的配置信息，配置项如下，可以使用证书文件中的client_ssl.properties文件：

- bootstrap.servers 配置为SSL协议对应的接入点地址，具体请参考接入点查看。
- security.protocol 固定为 SSL
- ssl.truststore.location 配置为 client.truststore.jks 文件路径
- ssl.truststore.password 固定为 bms@kafka
- ssl.keystore.location 配置为 client.keystore.jks 文件路径
- ssl.keystore.password 配置为keystore的访问密码，可以从client_ssl.properties中获取
- ssl.endpoint.identification.algorithm 固定为空

```
bootstrap.servers=<接入点地址>

security.protocol=SSL
ssl.truststore.location=client.truststore.jks
ssl.truststore.password=bms@kafka
ssl.keystore.location=client.keystore.jks
ssl.keystore.password=password
ssl.endpoint.identification.algorithm=
```

kafka.properties 可以作为 resources 文件与代码一起打包，代码运行时会在 classpath 中寻找 kafka.properties 文件资源；也可以在运行代码时将 kafka.properties 置于进程工作目录同级的 config 目录下。

步骤五：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
path	接入点信息kafka.properties所在路径（建议写文件所在的绝对路径）
topic	主题名称
message	消息的具体内容
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.SSL;

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.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
```

```
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test_ssl";
        // 消息内容
        String message = "kafka java test";

        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // Kafka消息的序列化方式。
        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);

        // 构建kafkaProducer对象
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<>(properties);

        try {
            // 向指定的topic发送100条消息
            for (int i = 0; i < 100; i++) {
                // 通过 ProducerRecord 构造一个消息对象
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic, message + "-" + i);
                // 通过kafkaProducer发送消息
                kafkaProducer.send(kafkaMessage, (RecordMetadata recordMetadata, Exception e) -> {
                    // 发送信息后的回调函数，用以验证消息是否发送成功
                    if (e == null) {
                        System.out.println("send success:" + recordMetadata.toString());
                    } else {
                        e.printStackTrace();
                        System.err.println("send failed");
                    }
                });
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            // 不要忘记关闭资源
            kafkaProducer.close();
        }
    }
```

```
    }
}
```

消费者代码示例

创建KafkaConsumerDemo.java文件，具体代码示例如下：

```
package org.example.Java示例.SSL;

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

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

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

        // 需要自行配置下面三个参数
        // kafka.properties所在路径（建议写文件所在的绝对路径）
        String path = "kafka.properties";
        // 主题名称-topic name
        String topic = "test_ssl";
        // 消费组id
        String group_id = "test_group";

        // 创建配置类，并获取配置文件 kafka.properties 的内容。
        Properties properties = new Properties();
        File file = new File(path);
        try {
            if (file.exists()) {
                // 如果通过用户指定的path路径找到了kafka.properties文件，则加载kafka.properties中的配置项
                properties.load(new FileInputStream(file));
            } else {
                // 如果没有从path中找到，则从KafkaProducerDemo所在的路径去查找
                properties.load(
                        KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties")
                );
            }
        } catch (IOException e) {
            // 没找到kafka.properties文件，在此处处理异常
            throw e;
        }

        // Kafka消息的序列化方式
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
        // 指定消费组id
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, group_id);
        // enable.auto.commit如果为true，则消费者的偏移量将定期在后台提交。
        properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        // 重置消费位点策略:earliest、latest、none
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        // 设置kafka自动提交offset的频率，默认5000ms，也就是5s
```

```
        properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 1000);
        // 设置消费者在一次poll中返回的最大记录数
        properties.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 1000);
        // 设置消费者两次poll的最大时间间隔
        properties.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 1000);

        // 构建KafkaConsumer对象
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(properties);

        //订阅主题
        kafkaConsumer.subscribe(Collections.singleton(topic));

        try{
            //持续消费主题中的消息
            while(true){
                // 构建ConsumerRecord用于接收存储消息
                ConsumerRecords<String, String> kafkaMessage = kafkaConsumer.poll(Duration.ofMillis(5000));
                for (ConsumerRecord<String, String> consumerRecord : kafkaMessage) {
                    // 打印消息具体内容
                    System.out.printf("offset = %d, key = %s, value = %s%n", consumerRecord.offset(), consumerRecord.key(),
consumerRecord.value());
                }
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }finally {
            kafkaConsumer.close();
        }
    }
}
```

步骤六：编译并运行

编译并运行上述两个代码文件。

- 先启动KafkaConsumerDemo.java
- 再启动KafkaProducerDemo.java

步骤七：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

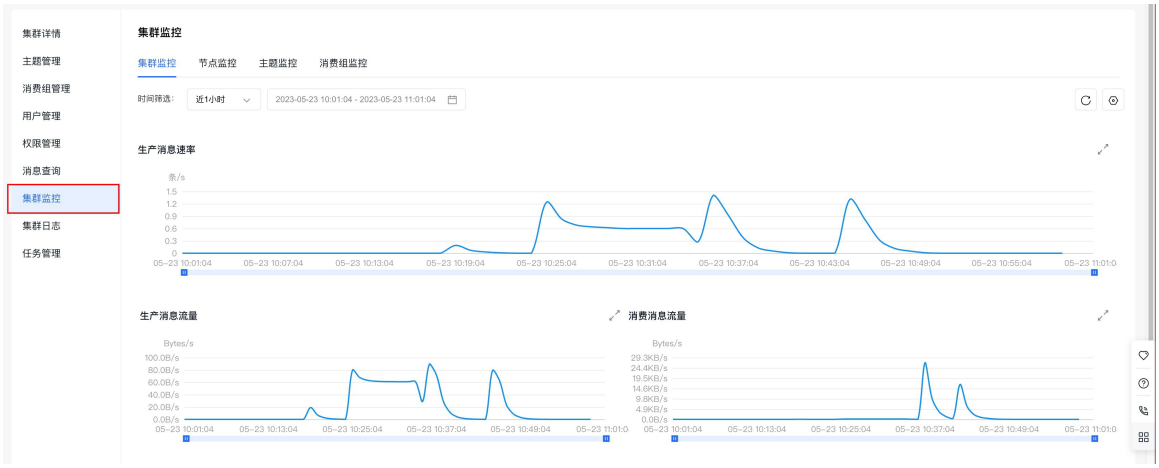
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

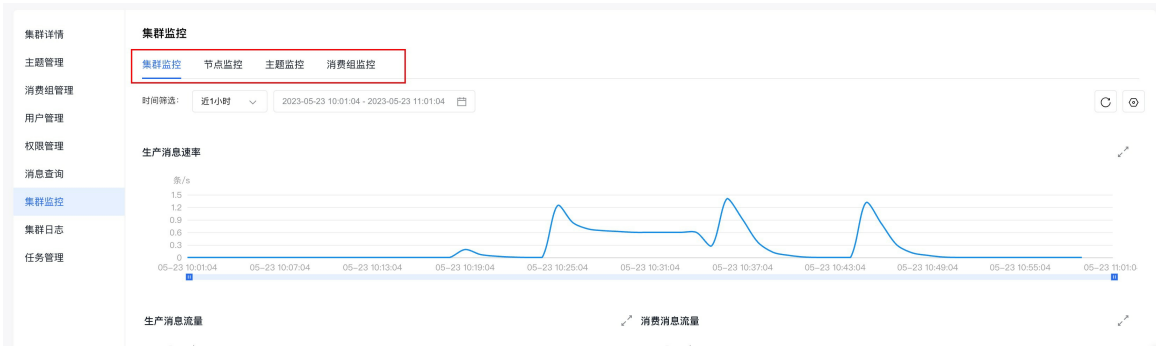
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



Go示例

VPC网络PLAINTEXT方式生产和消费

在同 VPC 网络下访问，使用 PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装Go。
2. 下载Go kafka 客户端。

```
go get github.com/confluentinc/confluent-kafka-go
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
bootstrap_servers	接入点信息
topic_name	主题名称
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "plaintext"
 }

 p, err := kafka.NewProducer(kafkaconf)
 if err != nil {
  panic(err)
 }

 defer p.Close()

 // Delivery report handler for produced messages
 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)
    }
   }
  }
 }()

 // 填写创建的主题名称
 topic := "topic_name"
 for _, word := range []string{"Golang", "for", "kafka", "client", "test"} {
  p.Produce(&kafka.Message{
   TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
   Value:          []byte(word),
  }, nil)
 }

 // Wait for message deliveries before shutting down
 p.Flush(15 * 1000)
}
```

消费者代码示例

创建KafkaConsumerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)


func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "plaintext",
  // 消费组 id
  "group.id": "test_group",
  "auto.offset.reset": "earliest",
 }

 c, err := kafka.NewConsumer(kafkaconf)

 if err != nil {
  panic(err)
 }

    //填写创建的主题的名称
 c.SubscribeTopics([]string{"topic_name"}, nil)

 for {
  msg, err := c.ReadMessage(-1)
  if err == nil {
   fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
  } else {
   // The client will automatically try to recover from all errors.
   fmt.Printf("Consumer error: %v (%v)\n", err, msg)
  }
 }

 c.Close()
}
```

步骤三：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
go consumer.go
###### 启动生产者
go producer.go
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

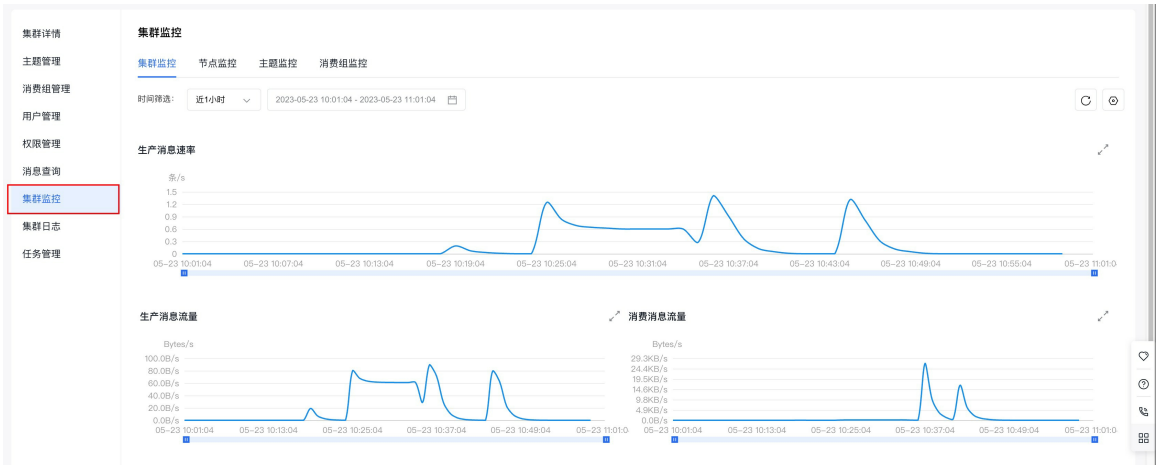
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

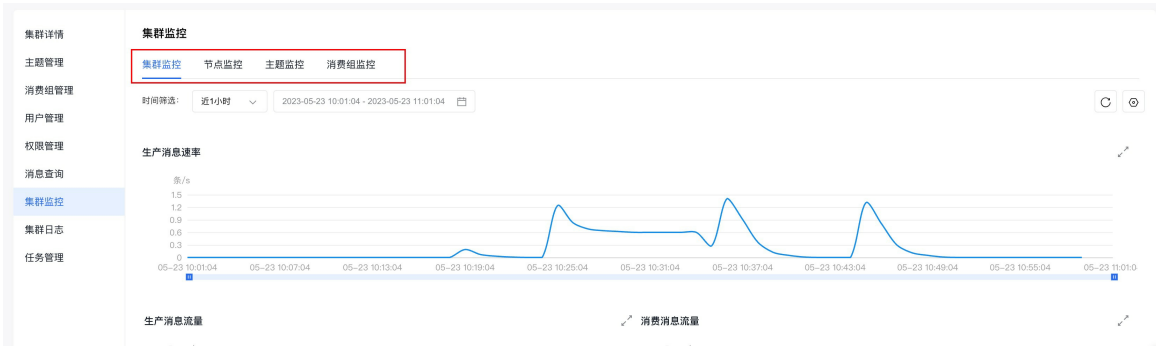
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



VPC网络SASL_PLAINTEXT方式生产和消费消息

在同 VPC 网络下访问，使用 SASL_PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装Go。
2. 下载Go kafka 客户端。

```
go get github.com/confluentinc/confluent-kafka-go
```

集群准备

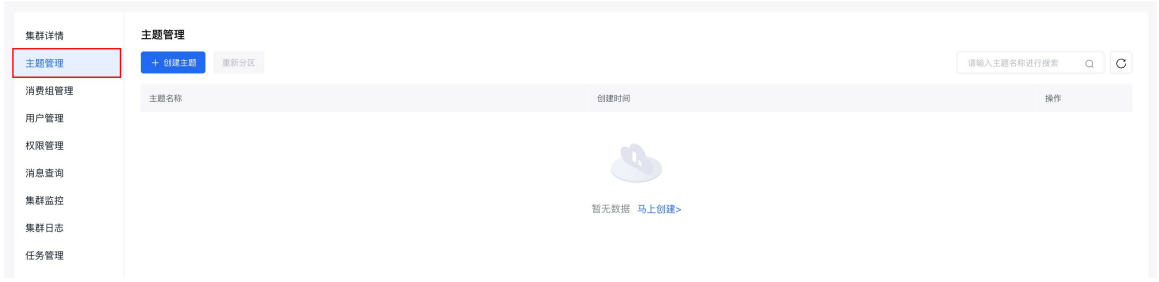
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
bootstrap_servers	接入点信息
topic_name	主题名称
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "sasl_plaintext",
  // SASL 机制
  "sasl.mechanism": "SCRAM-SHA-512",
  // SASL 用户名
  "sasl.username": "alice",
  // SASL 用户密码
  "sasl.password": "alice1234!",
 }

 p, err := kafka.NewProducer(kafkaconf)
 if err != nil {
  panic(err)
 }

 defer p.Close()

 // Delivery report handler for produced messages
 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)
    }
   }
  }
 }()

 // 填写创建的主题名称
 topic := "topic_name"
 for _, word := range []string{"Golang", "for", "kafka", "client", "test"} {
  p.Produce(&kafka.Message{
   TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
   Value:          []byte(word),
  }, nil)
 }

 // Wait for message deliveries before shutting down
 p.Flush(15 * 1000)
}
```

消费者代码示例

创建KafkaConsumerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)


func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "sasl_plaintext",
  // SASL 机制
  "sasl.mechanism": "SCRAM-SHA-512",
  // SASL 用户名
  "sasl.username": "alice",
  // SASL 用户密码
  "sasl.password": "alice1234!",
  // 消费组 id
  "group.id": "test_group",
  "auto.offset.reset": "earliest",
 }

 c, err := kafka.NewConsumer(kafkaconf)

 if err != nil {
  panic(err)
 }
    //填写创建的主题的名称
 c.SubscribeTopics([]string{"topic_name"}, nil)

 for {
  msg, err := c.ReadMessage(-1)
  if err == nil {
   fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
  } else {
   // The client will automatically try to recover from all errors.
   fmt.Printf("Consumer error: %v (%v)\n", err, msg)
  }
 }

 c.Close()
}
```

步骤三：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
go consumer.go
###### 启动生产者
go producer.go
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

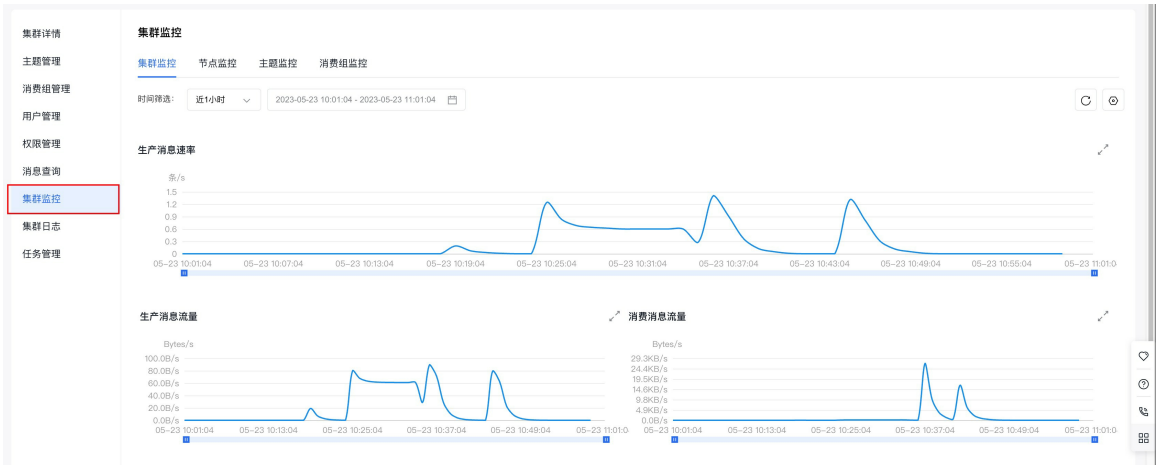
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

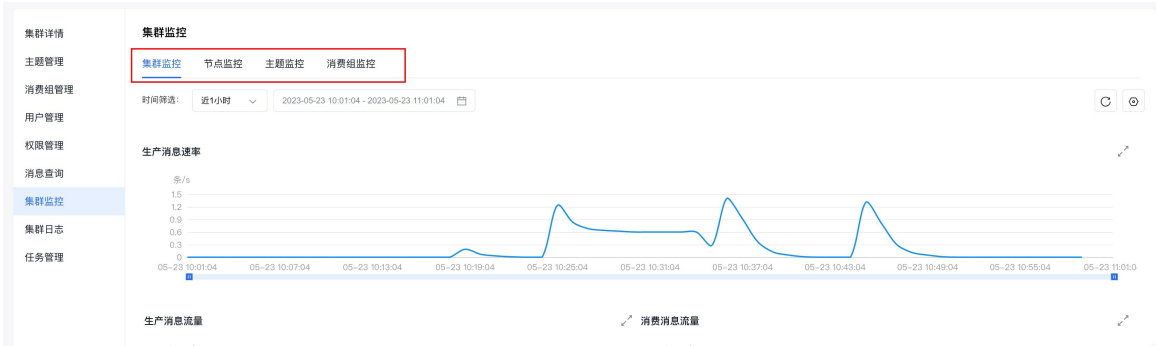
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



公网SASL_SSL方式生产和消费消息

在 Kafka 集群所在 VPC 网络外访问，使用 SASL_SSL 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装Go。

2. 下载Go kafka 客户端。

```
go get github.com/confluentinc/confluent-kafka-go
```

集群准备

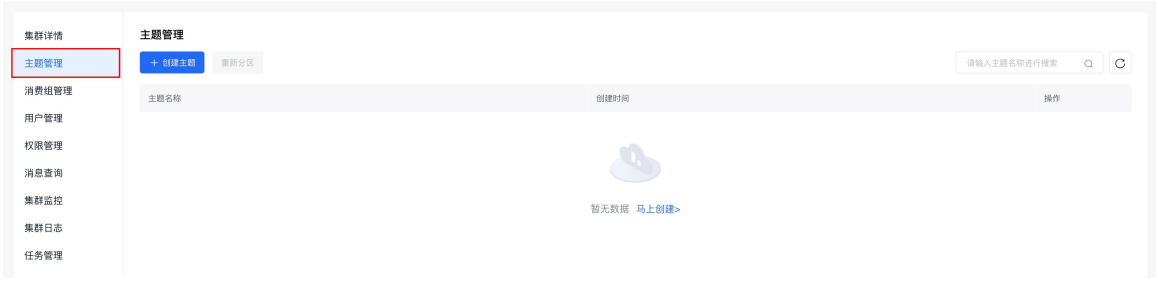
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
bootstrap_servers	接入点信息
topic_name	主题名称
ssl.ca.location	ca.pem文件所在路径
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "sasl_ssl",
  // 证书文件路径
  "ssl.ca.location": "ca.pem",
  // SASL 机制
  "sasl.mechanism": "SCRAM-SHA-512",
  // SASL 用户名
  "sasl.username": "alice",
  // SASL 用户密码
  "sasl.password": "alice1234!",
 }

 p, err := kafka.NewProducer(kafkaconf)
 if err != nil {
  panic(err)
 }

 defer p.Close()

 // Delivery report handler for produced messages
 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)
    }
   }
  }
 }()

 // 填写创建的主题名称
 topic := "topic_name"
 for _, word := range []string{"Golang", "for", "kafka", "client", "test"} {
  p.Produce(&kafka.Message{
   TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
   Value:          []byte(word),
  }, nil)
 }

 // Wait for message deliveries before shutting down
 p.Flush(15 * 1000)
}
```

消费者代码示例

创建KafkaConsumerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)


func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "sasl_ssl",
  // 证书文件路径
  "ssl.ca.location": "ca.pem",
  // SASL 机制
  "sasl.mechanism": "SCRAM-SHA-512",
  // SASL 用户名
  "sasl.username": "alice",
  // SASL 用户密码
  "sasl.password": "alice1234!",
  // 消费组 id
  "group.id": "test_group",
  "auto.offset.reset": "earliest",
 }

 c, err := kafka.NewConsumer(kafkaconf)

 if err != nil {
  panic(err)
 }

 c.SubscribeTopics([]string{"topic_name"}, nil)

 for {
  msg, err := c.ReadMessage(-1)
  if err == nil {
   fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
  } else {
   // The client will automatically try to recover from all errors.
   fmt.Printf("Consumer error: %v (%v)\n", err, msg)
  }
 }

 c.Close()
}
```

步骤四：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
go consumer.go
###### 启动生产者
go producer.go
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

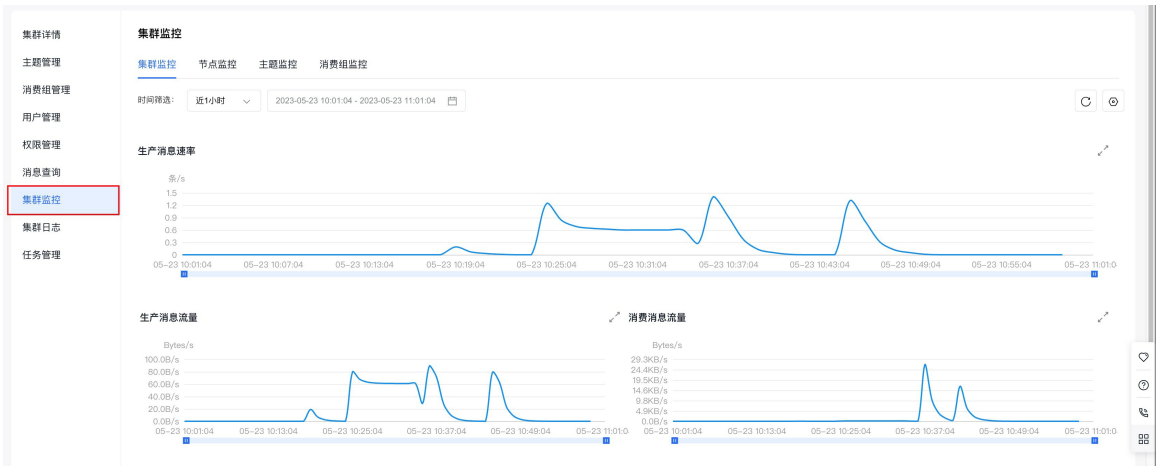
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

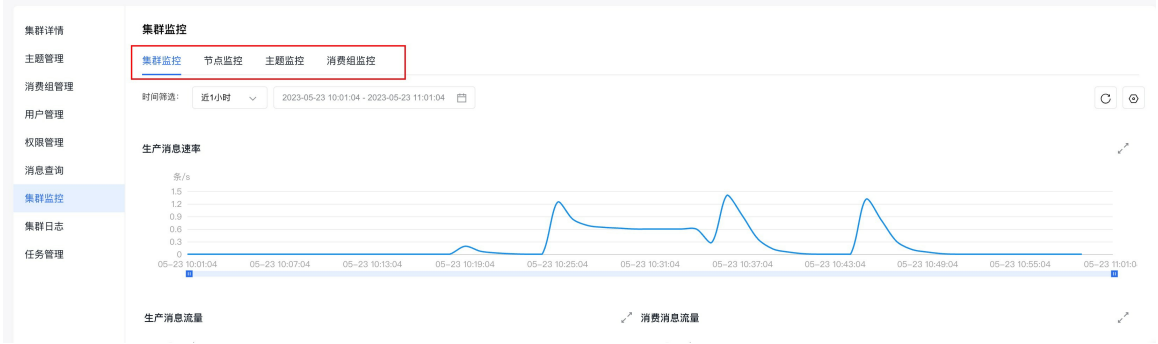
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



SSL协议生产和消费消息

在 Kafka 集群所在 VPC 网络下或者公网环境访问，使用 SSL 协议接入，接入点可以在**集群详情**页面查看。

环境准备

1. 安装Go。

2. 下载Go kafka 客户端。

```
go get github.com/confluentinc/confluent-kafka-go
```

集群准备

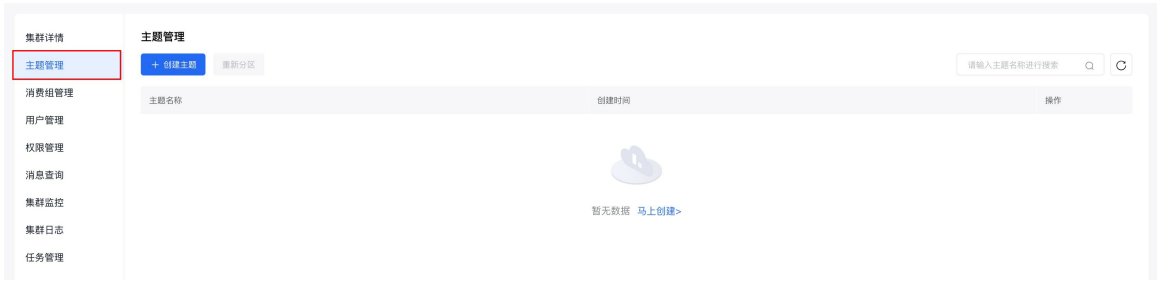
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
bootstrap_servers	接入点信息
topic_name	主题名称
ssl.ca.location	ca.pem文件所在路径
ssl.certificate.location	client.pem文件所在路径
ssl.key.location	client.key文件所在路径
group_id	消费组id

生产者代码示例

创建KafkaProducerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "ssl",
  // 证书文件路径
  "ssl.ca.location": "ca.pem",
        "ssl.certificate.location": "client.pem",
        "ssl.key.location": "client.key",
 }

 p, err := kafka.NewProducer(kafkaconf)
 if err != nil {
  panic(err)
 }

 defer p.Close()

 // Delivery report handler for produced messages
 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)
    }
   }
  }
 }()

 // 填写创建的主题名称
 topic := "topic_name"
 for _, word := range []string{"Golang", "for", "kafka", "client", "test"} {
  p.Produce(&kafka.Message{
   TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
   Value:          []byte(word),
  }, nil)
 }

 // Wait for message deliveries before shutting down
 p.Flush(15 * 1000)
}
```

消费者代码示例

创建KafkaConsumerDemo.go文件，具体代码示例如下：

```
 package main

import (
"fmt"
"github.com/confluentinc/confluent-kafka-go/kafka"
)


func main() {

 var kafkaconf = &kafka.ConfigMap{
  // 接入点
  "bootstrap.servers": "接入点",
  // 接入协议
  "security.protocol": "ssl",
  // 证书文件路径
  "ssl.ca.location": "ca.pem",
        "ssl.certificate.location": "client.pem",
        "ssl.key.location": "client.key",
  // 消费组 id
  "group.id": "test_group",
  "auto.offset.reset": "earliest",
 }

 c, err := kafka.NewConsumer(kafkaconf)

 if err != nil {
  panic(err)
 }

 c.SubscribeTopics([]string{"topic_name"}, nil)

 for {
  msg, err := c.ReadMessage(-1)
  if err == nil {
   fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
  } else {
   // The client will automatically try to recover from all errors.
   fmt.Printf("Consumer error: %v (%v)\n", err, msg)
  }
 }

 c.Close()
}
```

步骤四：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
go consumer.go
###### 启动生产者
go producer.go
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

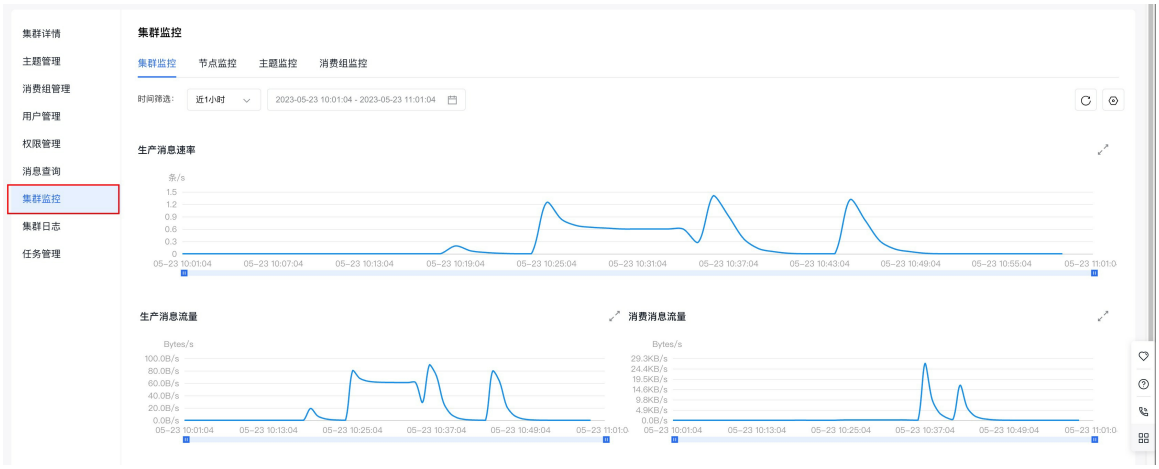
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

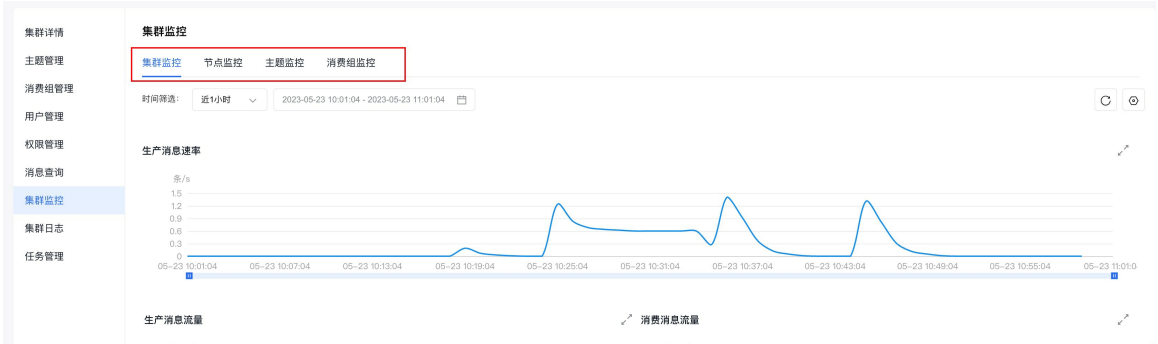
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



Python示例

VPC网络PLAINTEXT方式生产和消费

在同 VPC 网络下访问，使用 PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装 Python
2. 安装 pip
3. 运行如下命令下载confluent-kafka依赖。

```
pip install confluent-kafka
```

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
bootstrap.servers	接入点信息
topic_name	主题名称
message	消息的具体内容
group.id	消费组id

生产者代码示例

创建KafkaProducerDemo.py文件，具体代码示例如下：

```
 from confluent_kafka import Producer

producer = Producer({
    # 接入点
    'bootstrap.servers':'x.x.x.x:9095,x.x.x.x:9095,x.x.x.x:9095',
    # 接入协议
    'security.protocol':'PLAINTEXT',
})

def callback_msg(err, msg):
    if err is not None:
        print('send failed:{}'.format(err))
    else:
        print('send success:{}'.format(msg.topic(),msg.partition()))

for _ in range(100):
    # 第一个参数topic_name填写创建的主题名称，第二个参数message写需要发送的消息内容
    producer.produce('topic_name', "message".encode('utf-8'), callback = callback_msg)
    producer.poll(0)

producer.flush()
```

消费者代码示例

创建KafkaConsumerDemo.py文件，具体代码示例如下：

```
from confluent_kafka import Consumer

consumer = Consumer({
    # 接入点
    'bootstrap.servers':'x.x.x.x:9095,x.x.x.x:9095,x.x.x.x:9095',
    # 接入协议
    'security.protocol':'PLAINTEXT',
    # 消费组id
    'group.id':'test_group',
    'auto.offset.reset':'latest',
    'fetch.message.max.bytes':'1024*512',
})

###### 订阅的主题名称
consumer.subscribe(['topic_name'])

while True:
    msg = consumer.poll(1.0)
    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    print('Received message: {}'.format(msg.value().decode('utf-8')))

consumer.close()
```

步骤三：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
python KafkaConsumerDemo.py
###### 启动生产者
python KafkaProducerDemo.py
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

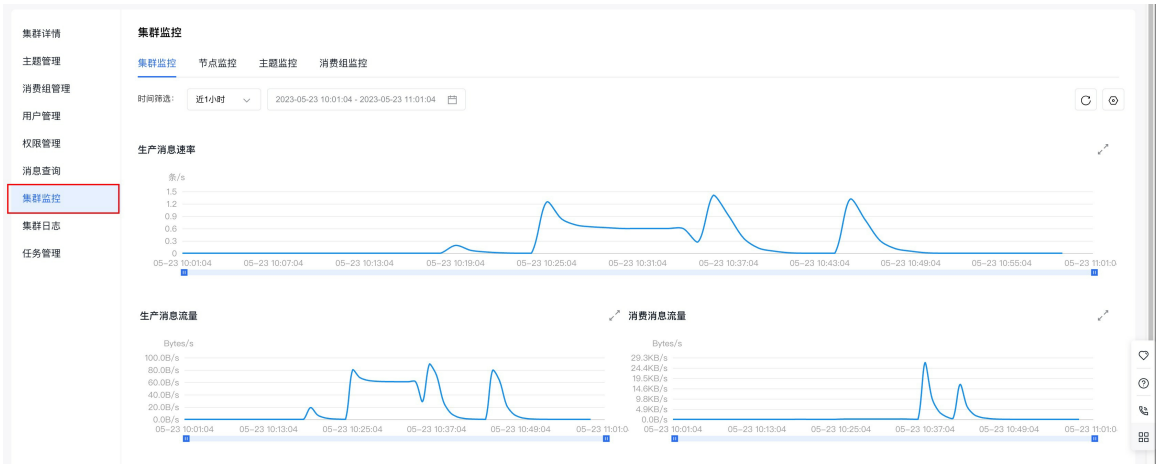
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

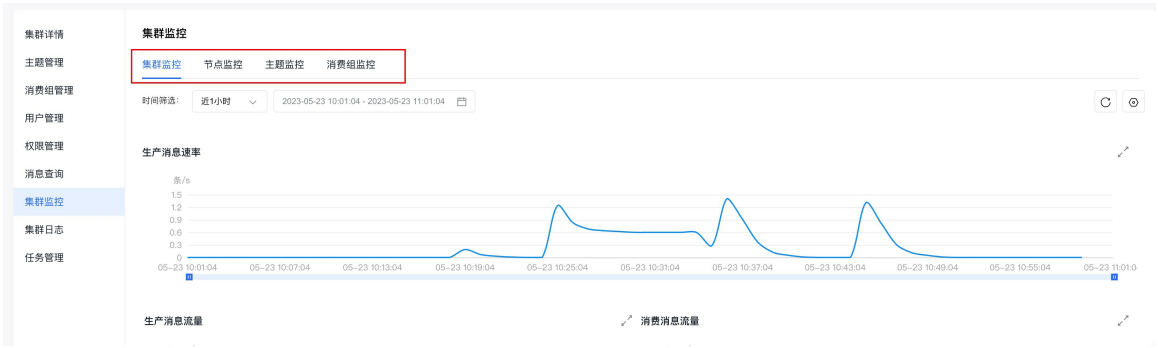
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



公网SASL_SSL方式生产和消费消息

在 Kafka 集群所在 VPC 网络外访问，使用 SASL_SSL 协议接入，接入点可以在**集群详情** 页面查看。

环境准备

1. 安装 Python
2. 安装 pip
3. 运行如下命令下载confluent-kafka依赖。

```
pip install confluent-kafka
```

集群准备

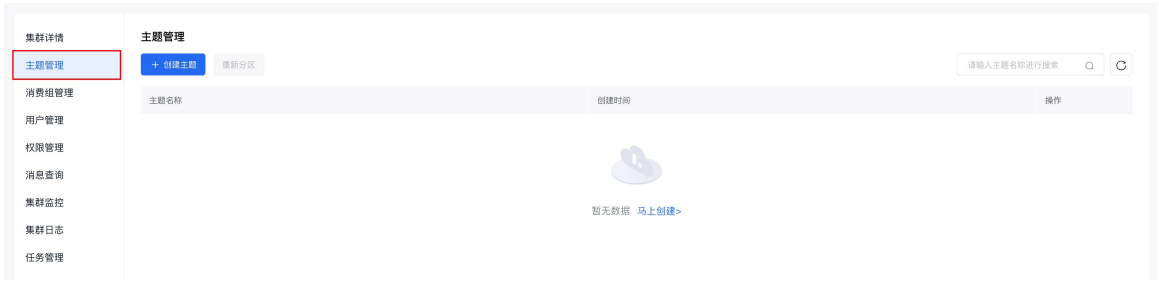
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
bootstrap.servers	接入点信息
topic_name	主题名称
message	消息的具体内容
ssl.ca.location	ca.pem证书文件路径
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
group.id	消费组id

生产者代码示例

创建KafkaProducerDemo.py文件，具体代码示例如下：

```
 from confluent_kafka import Producer

producer = Producer({
    # 接入点
    'bootstrap.servers':'接入点',
    # 接入协议
    'security.protocol':'SASL_SSL',
    'ssl.endpoint.identification.algorithm': 'none',
    # 证书文件路径
    'ssl.ca.location':'ca.pem',
    # SASL 机制
    'sasl.mechanism':'SCRAM-SHA-512',
    # SASL 用户名
    'sasl.username':'username',
    # SASL 用户密码
    'sasl.password':'password'
})

def callback_msg(err, msg):
    if err is not None:
        print('send failed:{}'.format(err))
    else:
        print('send success:{}'.format(msg.topic(),msg.partition()))

for _ in range(100):
    # 第一个参数topic_name填写创建的主题名称，第二个参数message写需要发送的消息内容
    producer.produce('topic_name', "message".encode('utf-8'), callback = callback_msg)
    producer.poll(0)

producer.flush()
```

消费者代码示例

创建KafkaConsumerDemo.py文件，具体代码示例如下：

```
from confluent_kafka import Consumer

consumer = Consumer({
    # 接入点
    'bootstrap.servers':'接入点',
    # 接入协议
    'security.protocol':'SASL_SSL',
    'ssl.endpoint.identification.algorithm':'none',
    # 证书文件路径
    'ssl.ca.location':'ca.pem',
    # SASL 机制
    'sasl.mechanism':'SCRAM-SHA-512',
    # SASL 用户名
    'sasl.username':'username',
    # SASL 用户密码
    'sasl.password':'password',
    # 消费组id
    'group.id':'test_group',
    'auto.offset.reset':'latest',
    'fetch.message.max.bytes':'1024*512',
})

###### 订阅的主题名称
consumer.subscribe(['topic_name'])

while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    print('Received message: {}'.format(msg.value().decode('utf-8')))

consumer.close()
```

步骤四：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
python KafkaConsumerDemo.py
###### 启动生产者
python KafkaProducerDemo.py
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

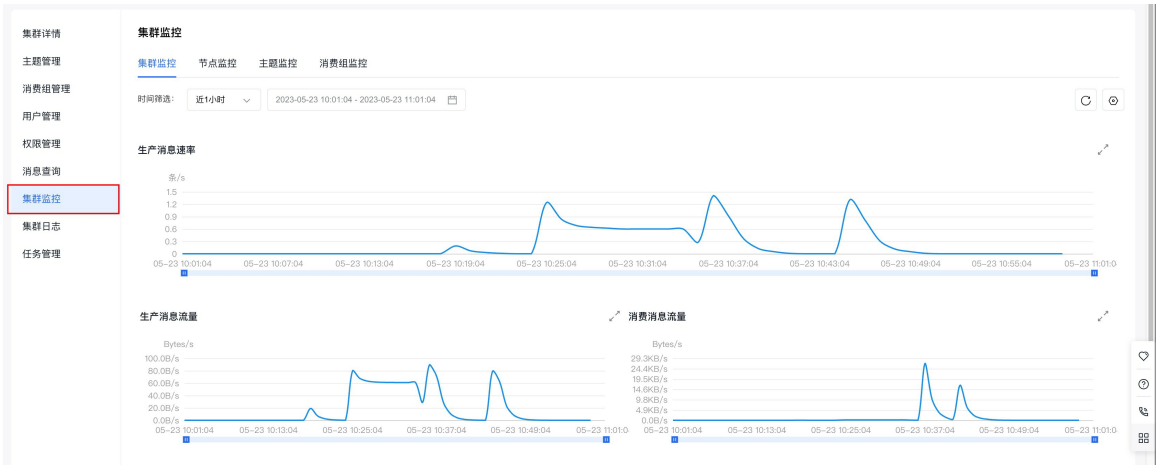
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

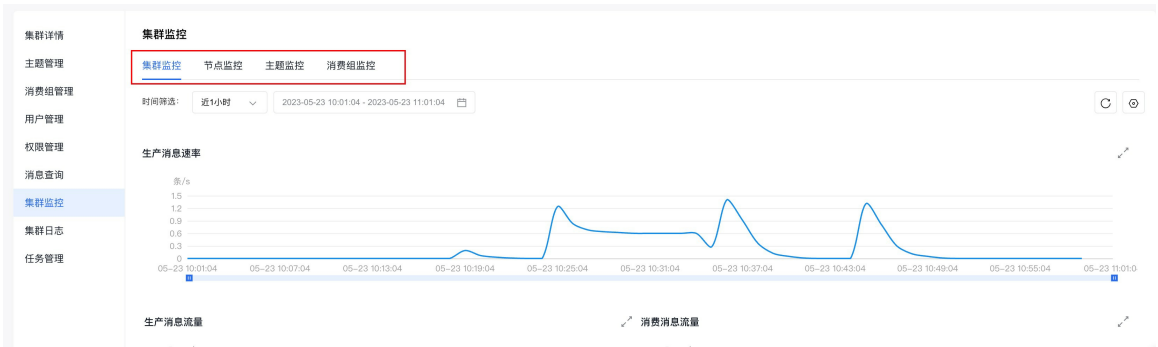
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



VPC网络SASL_PLAINTEXT方式生产和消费消息

在同 VPC 网络下访问，使用 SASL_PLAINTEXT 协议接入，接入点可以在 **集群详情** 页面查看。

环境准备

1. 安装 Python
2. 安装 pip
3. 运行如下命令下载confluent-kafka依赖。

```
pip install confluent-kafka
```

集群准备

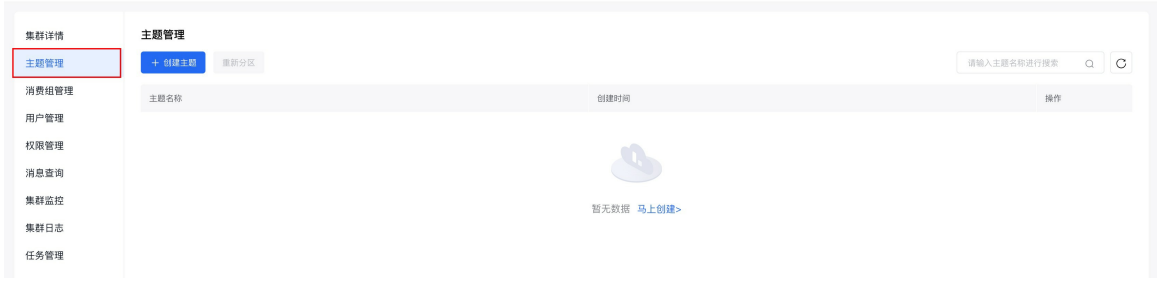
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：编写测试代码

- 需要关注并自行修改的参数
- 认证机制支持PLAIN、SCRAM-SHA-512两种机制，根据集群所使用的认证方式进行选择

参数名	含义
bootstrap.servers	接入点信息
topic_name	主题名称
message	消息的具体内容
sasl.username	用户管理中创建用户的用户名
sasl.password	用户管理中创建用户的密码
group.id	消费组id

生产者代码示例

创建KafkaProducerDemo.py文件，具体代码示例如下：

```
 from confluent_kafka import Producer

producer = Producer({
    # 接入点
    'bootstrap.servers':'接入点',
    # 接入协议
    'security.protocol':'SASL_PLAINTEXT',
    # 'ssl.endpoint.identification.algorithm': 'none',
    # 'ssl.ca.location':'client.truststore.pem',
    # SASL 机制
    'sasl.mechanism':'SCRAM-SHA-512',
    # SASL 用户名
    'sasl.username':'username',
    # SASL 用户密码
    'sasl.password':'password'
})

def callback_msg(err, msg):
    if err is not None:
        print('send failed:{}'.format(err))
    else:
        print('send success:{}'.format(msg.topic(),msg.partition()))

for _ in range(100):
    # 第一个参数topic_name填写创建的主题名称，第二个参数message写需要发送的消息内容
    producer.produce('topic_name', "message".encode('utf-8'), callback = callback_msg)
    producer.poll(0)

producer.flush()
```

消费者代码示例

创建KafkaConsumerDemo.py文件，具体代码示例如下：

```
from confluent_kafka import Consumer

consumer = Consumer({
    # 接入点
    'bootstrap.servers':'接入点',
    # 接入协议
    'security.protocol':'SASL_PLAINTEXT',
    # 'ssl.endpoint.identification.algorithm':'none',
    # 证书文件路径
    # 'ssl.ca.location':'client.truststore.pem',
    # SASL 机制
    'sasl.mechanism':'SCRAM-SHA-512',
    # SASL 用户名
    'sasl.username':'username',
    # SASL 用户密码
    'sasl.password':'password',
    # 消费组id
    'group.id':'test_group',
    'auto.offset.reset':'latest',
    'fetch.message.max.bytes':'1024*512',
})

###### 订阅的主题名称
consumer.subscribe(['topic_name'])

while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    print('Received message: {}'.format(msg.value().decode('utf-8')))

consumer.close()
```

步骤三：编译并运行

运行上述两个代码文件。

```
###### 启动消费者
python KafkaConsumerDemo.py
###### 启动生产者
python KafkaProducerDemo.py
```

步骤四：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

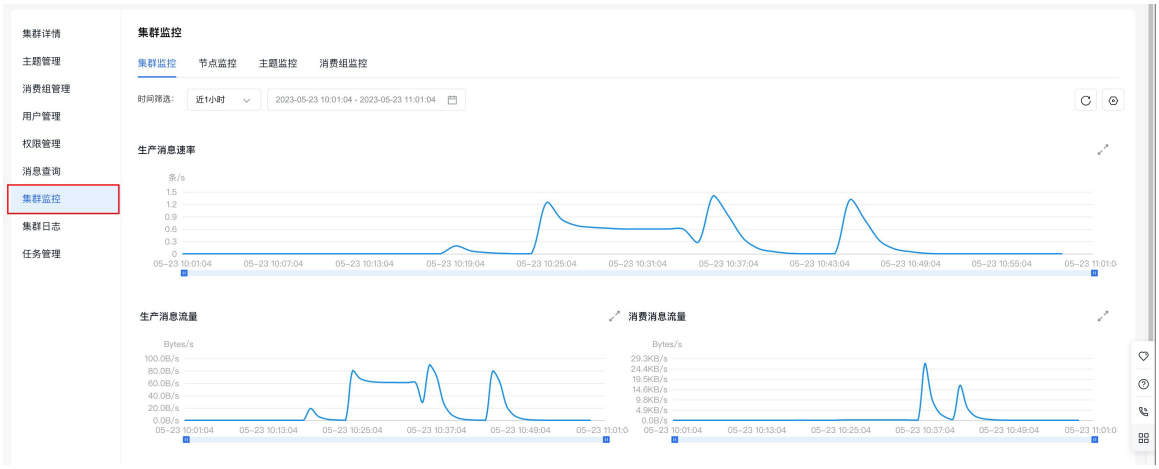
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

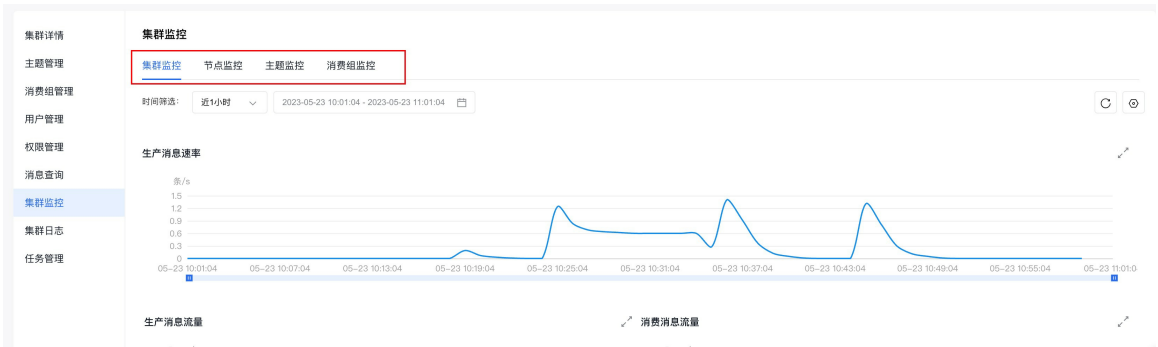
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



SSL协议生产和消费消息

在 Kafka 集群所在 VPC 网络下或者公网环境访问，使用 SSL 协议接入，接入点可以在**集群详情**页面查看。

环境准备

1. 安装 Python
2. 安装 pip
3. 运行如下命令下载confluent-kafka依赖。

```
pip install confluent-kafka
```

集群准备

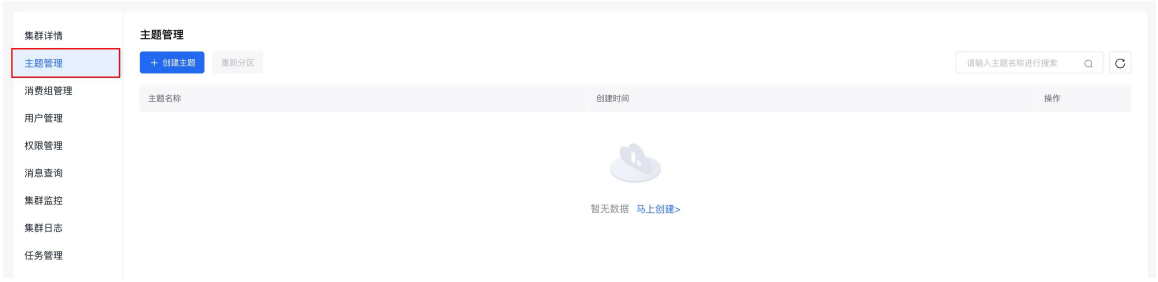
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。



在主题管理页面点击『创建主题』，进行主题的创建。

使用步骤：

步骤一：获取集群接入点

具体请参考：接入点查看。

步骤二：下载证书文件

下载证书文件：如何下载证书？

步骤三：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
bootstrap.servers	接入点信息
topic_name	主题名称
message	消息的具体内容
ssl.ca.location	ca.pem文件路径
ssl.certificate.location	client.pem文件路径
ssl.key.location	client.key文件路径
group.id	消费组id

生产者代码示例

创建KafkaProducerDemo.py文件，具体代码示例如下：

```
 from confluent_kafka import Producer

producer = Producer({
    # 接入点
    'bootstrap.servers':'xxx.xx.xx.xx:9099',
    # 接入协议
    'security.protocol':'SSL',
    'ssl.endpoint.identification.algorithm': 'none',
    # 证书文件路径
    'ssl.ca.location':'ca.pem',
    'ssl.certificate.location':'client.pem',
    'ssl.key.location':'client.key'
})

def callback_msg(err, msg):
    if err is not None:
        print('send failed:{}'.format(err))
    else:
        print('send success:{}'.format(msg.topic(),msg.partition()))

for _ in range(100):
    # 第一个参数topic_name填写创建的主题名称，第二个参数message写需要发送的消息内容
    producer.produce('topic_name', "message".encode('utf-8'), callback = callback_msg)
    producer.poll(0)

producer.flush()
```

消费者代码示例

创建KafkaConsumerDemo.py文件，具体代码示例如下：

```
 from confluent_kafka import Consumer

consumer = Consumer({
    # 接入点
    'bootstrap.servers':'xxx.xx.xx.xx:9099',
    # 接入协议
    'security.protocol':'SSL',
    'ssl.endpoint.identification.algorithm':'none',
    # 证书文件路径
    'ssl.ca.location':'ca.pem',
    'ssl.certificate.location':'client.pem',
    'ssl.key.location':'client.key',
    # 消费组id
    'group.id':'test_group',
    'auto.offset.reset':'latest',
    'fetch.message.max.bytes':'1024*512',
})

###### 订阅的主题名称
consumer.subscribe(['topic_name'])

while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    print('Received message: {}'.format(msg.value().decode('utf-8')))

consumer.close()
```

步骤四：编译并运行

运行上述两个代码文件。

```
 ###### 启动消费者
python KafkaConsumerDemo.py
###### 启动生产者
python KafkaProducerDemo.py
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 查看程序输出日志。

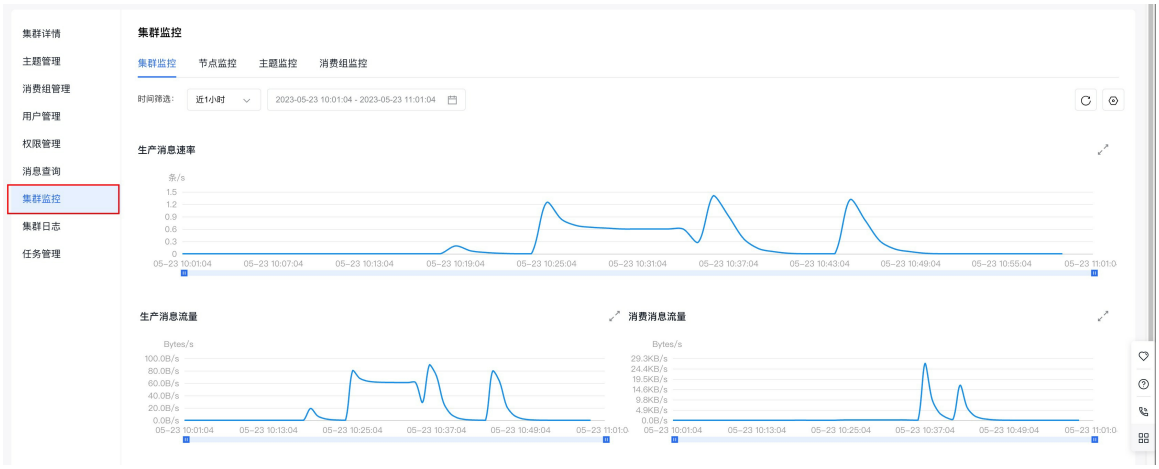
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入『集群详情』页面。

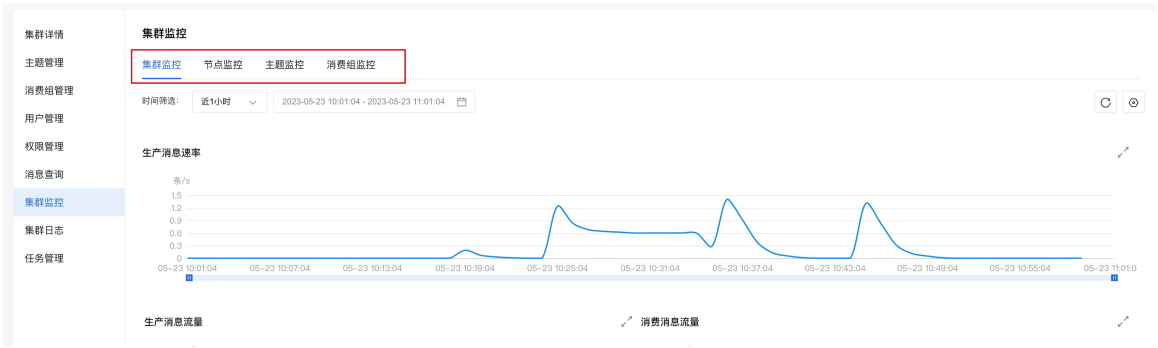
（2）页面跳转后，进入左侧边中的『集群详情』页面。

（3）点击左侧边栏中的『集群监控』，进入『集群监控』页面。



（4）通过查看『集群监控』页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



最佳实践

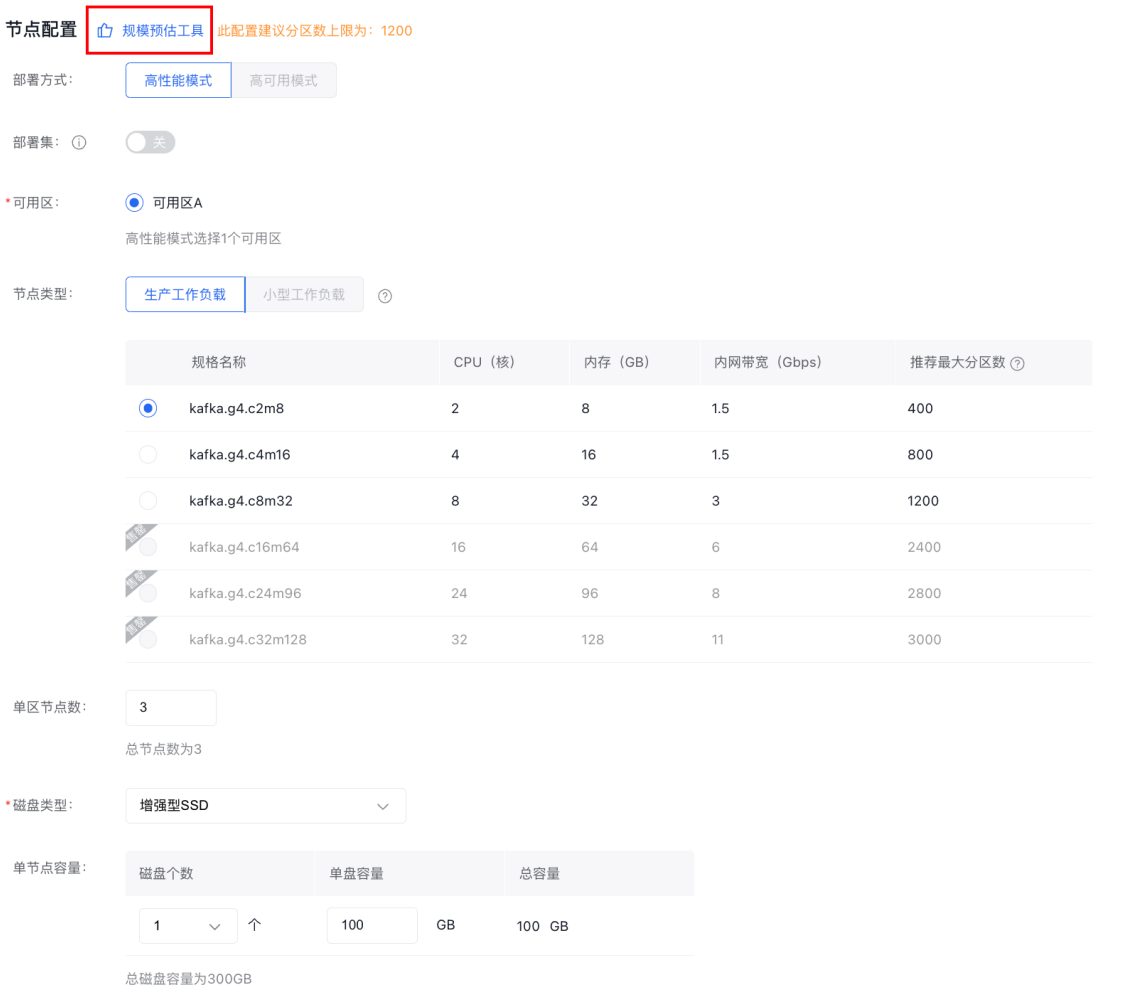
如何选择合适的集群规格

1. 预估业务流量

业务方需要预估自身流量需求，这里只考虑生产流量，即生产者每秒产生的数据量。

2. 使用规模预估工具获取推荐配置

在**创建集群**页面的**节点配置**选项旁提供了规模预估工具：



点击 **配置推荐** 后需要选择"流量峰值"、"部署方式"、"可用区"三个选项，下方会给出这种场景下推荐的集群配置：

选项说明：

1. 流量峰值：对于 Kafka 集群，实际集群 **写入流量** 为业务流量 *主题副本数，同时推荐预留 20% 的 buffer 应对后续业务增长、集群运维（节点重启、节点故障、主题分区迁移等场景）的需要，因此这里选择的流量峰值 =（预估业务流量* 预估副本数）* 120%
2. 部署方式：根据自身业务需求进行选择
    1. 高性能模式：会将 Kafka 节点部署在一个可用区下，减少跨可用区传输过程中造成的延迟。

2. 高可用模式：会将 Kafka 节点部署在两个可用区下，可对集群进行多可用区灾备处理。

3. 可用区：代表物理资源相对隔离的机房，尽量选择靠近自身业务以及有足够资源的可用区。详细介绍可以参考：可用区。

目前最高提供了上限为 2400MBps 流量下的推荐配置，如果业务需要更高流量，请提交工单进行咨询。

业务迁移

本文主要介绍将自建Kafka集群迁移到专享版 消息服务 for Kafka集群的相关操作。

迁移前提

- 存在一个可用的专享版 消息服务 for Kafka。
- 为专享版 消息服务 for Kafka创建主题。

集群准备

1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击『主题管理』，进入主题管理页面。

在主题管理页面点击『创建主题』，进行主题的创建。

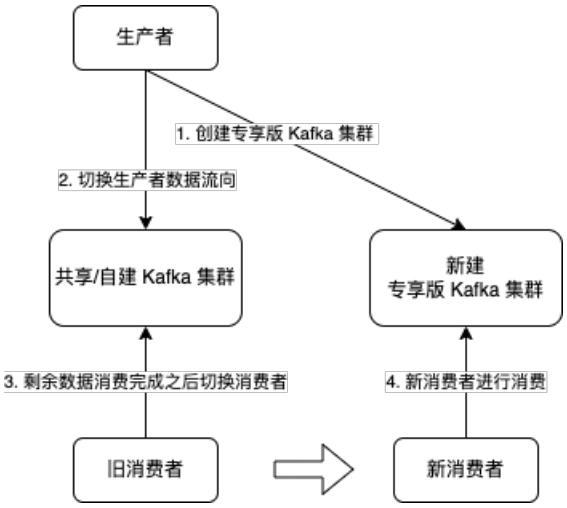
开通消息服务 for Kafka服务后，在控制台页面点击『创建集群』，即可进行购买。

迁移方案

方案一：先迁生产，后迁消费

先将生产消息的业务迁移到专享版 消息服务 for Kafka集群，这样原Kafka集群不会有新的消息生产。

待原有Kafka集群的消息全部消费完成后，再将消费消息业务迁移到专享版 消息服务 for Kafka集群，开始消费消息。



适用场景

本方案为业界通用的迁移方案，操作步骤简单，迁移过程由业务侧自主控制，整个过程中消息不会存在乱序问题，适用于对消息顺序有要求的场景。但是该方案中需要等待消费者业务直至消费完毕，存在一个时间差的问题，部分数据可能存在较大的端到端时延。

迁移步骤

1. 专享版 消息服务 for Kafka集群，在集群中创建好 topic 等资源，具体操作见『集群准备』。

2. 切换生产者写入地址：

- - 将生产客户端的Kafka连接地址修改为专享版 消息服务 for Kafka集群所提供的接入点地址。接入点查看请参考：查看集群接入点
  - 重启生产业务，使得生产者将新的消息发送到专享版 消息服务 for Kafka集群中。

1. 观察各消费组在原Kafka的消费进度，直到原Kafka中数据都已经被消费完毕。

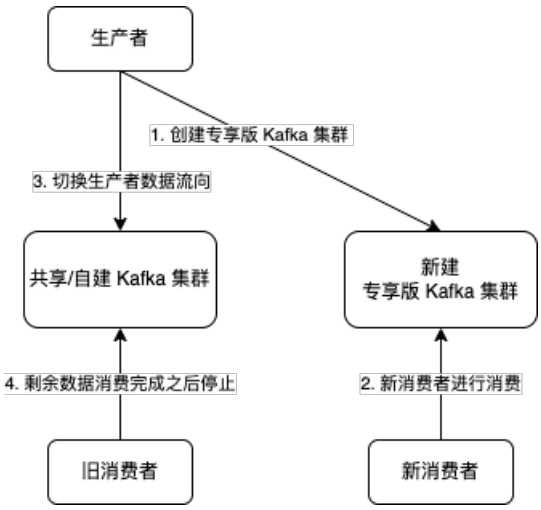
2. 切换消费者

- - 将消费客户端的Kafka连接地址修改为专享版 消息服务 for Kafka集群所提供的接入点地址。接入点查看请参考：查看集群接入点
  - 重启消费业务，使得消费者从专享版 消息服务 for Kafka集群中消费消息。

1. 观察消费者是否能正常从专享版 消息服务 for Kafka集群中获取数据。
2. 迁移结束。

方案二：同时消费，后迁生产

指消费者业务启用多个消费客户端，分别向原Kafka集群和专享版 消息服务 for Kafka集群消费消息，然后将生产业务切到专享版 消息服务 for Kafka集群，这样能确保所有消息都被及时消费。



适用场景

迁移过程由业务自主控制。本方案中消费业务会在一段时间内同时消费原Kafka集群和专享版 消息服务 for Kafka集群。由于在迁移生产业务之前，已经有消费业务运行在专享版 消息服务 for Kafka集群上，因此不会存在端到端时延的问题。但在迁移生产的开始阶段，同时消费原Kafka集群与专享版 消息服务 for Kafka集群，会导致部分消息之间的生产顺序无法保证，存在消息乱序的问题。此场景适用于对端到端时延有要求，却对消息顺序不敏感的业务。

迁移步骤

1. 专享版 消息服务 for Kafka集群，在集群中创建好 topic 等资源，具体操作见『集群准备』。
2. 启动新的消费客户端，配置Kafka连接地址为修改为专享版 消息服务 for Kafka集群所提供的接入点地址。消费专享版 消息服务 for Kafka集群中的数据。（原有消费客户端需继续运行，消费业务同时消费原Kafka集群与专享版 消息服务 for Kafka集群中的消息。）
3. 切换生产者写入地址：

- 修改生产客户端，Kafka连接地址改为专享版 消息服务 for Kafka集群所提供的接入点地址。接入点查看请参考：查看集群接入点
- 重启生产客户端，将生产业务迁移到专享版 消息服务 for Kafka集群中。
- 生产业务迁移后，观察连接专享版 消息服务 for Kafka集群的消费业务是否正常。

1. 等待原Kafka集群中数据消费完毕，关闭原有消费业务客户端。
2. 迁移结束。

使用MirrorMaker迁移

概述

MirrorMaker为Kafka自带的数据迁移工具，可以利用此种方式直接进行kafka集群之间的数据迁移；减少中间组件的使用。

具体介绍请参考Kakfa官方网站：MirrorMaker

迁移原理

通过Consumer从旧的Kafka集群中消费数据，然后通过Producer将数据生产在目标kafka集群中，来实现数据同步备份。

环境准备

准备一台与专享版 消息服务 for Kafka集群网络相通的服务器，例如：云服务器BCC。

可通过如下链接下载迁移工具：迁移工具，并将其上传到BCC中进行解压。

BCC中进行如下配置操作：

```
##### 1. 安装JDK 1.8
yum install java
##### 2. 下载并解压kafka_2.12-2.7.2.tgz文件
tar -zxvf kafka_2.12-2.7.2.tgz
```

迁移步骤

1. 进入kafka_2.12-2.7.2/config路径下，找到connect-mirror-maker.properties文件，并进行如下配置：

```
 ##### Licensed to the Apache Software Foundation (ASF) under A or more
##### contributor license agreements.  See the NOTICE file distributed with
##### this work for additional information regarding copyright ownership.
##### The ASF licenses this file to You under the Apache License, Version 2.0
##### (the "License"); you may not use this file except in compliance with
##### the License.  You may obtain a copy of the License at
#####
##### http://www.apache.org/licenses/LICENSE-2.0
#####
##### Unless required by applicable law or agreed to in writing, software
##### distributed under the License is distributed on an "AS IS" BASIS,
##### WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
##### See the License for the specific language governing permissions and
##### limitations under the License.
##### see org.apache.kafka.clients.consumer.ConsumerConfig for more details

##### Sample MirrorMaker 2.0 top-level configuration file
##### Run with ./bin/connect-mirror-maker.sh connect-mirror-maker.properties

##### 指定任意数量的集群别名
clusters = A, B

##### 配置新旧集群的地址
##### 例如A配置为原Kakfa集群的地址，B则配置为专享版 消息服务 for Kafka提供的接入点地址
A.bootstrap.servers = A_host1:port, A_host2:port, A_host3:port
B.bootstrap.servers = B_host1:port, B_host2:port, B_host3:port

##### 指定数据同步的方向（单向同步、相互同步）
A->B.enabled = true

##### 指定需要同步的主题，默认全部同步
A->B.topics = .*

##### 下面两行配置表示两个集群相互同步
B->A.enabled = true
B->A.topics = .*

##### 设置副本数
replication.factor=1

**Internal Topic Settings  ##############################**
##### The replication factor for mm2 internal topics "heartbeats", "B.checkpoints.internal" and
##### "mm2-offset-syncs.B.internal"
##### 下面的配置在测试环境中可以设置为1，生产环境中建议设置大于1的值，例如：3
checkpoints.topic.replication.factor=1
heartbeats.topic.replication.factor=1
offset-syncs.topic.replication.factor=1

##### The replication factor for connect internal topics "mm2-configs.B.internal", "mm2-offsets.B.internal" and
##### "mm2-status.B.internal"
##### 下面的配置在测试环境中可以设置为1，生产环境中建议设置大于1的值，例如：3
offset.storage.replication.factor=1
status.storage.replication.factor=1
config.storage.replication.factor=1

##### 有需要可以自行配置
##### replication.policy.separator = _
##### sync.topic.acls.enabled = false
##### emit.heartbeats.interval.seconds = 5
```

1. 进入kafka_2.12-2.7.2/bin路径下，启动MirrorMaker脚本工具，执行如下命令：

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

1. 等待数据迁移。

迁移结果查看

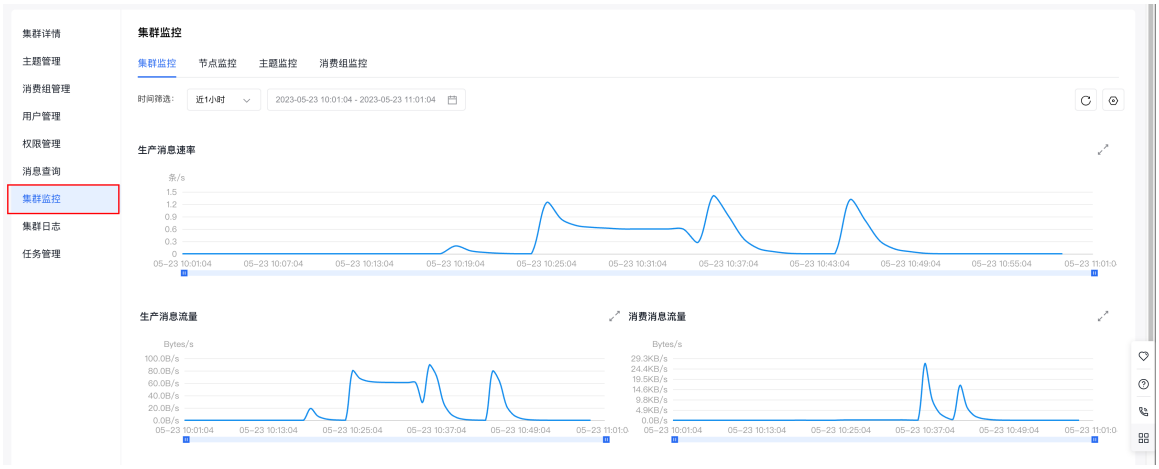
- 在原集群生产并消费消息。
- 在专享版 消息服务 for Kafka的集群监控中查看数据同步情况。

监控查看步骤如下：

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入**集群详情**页面。

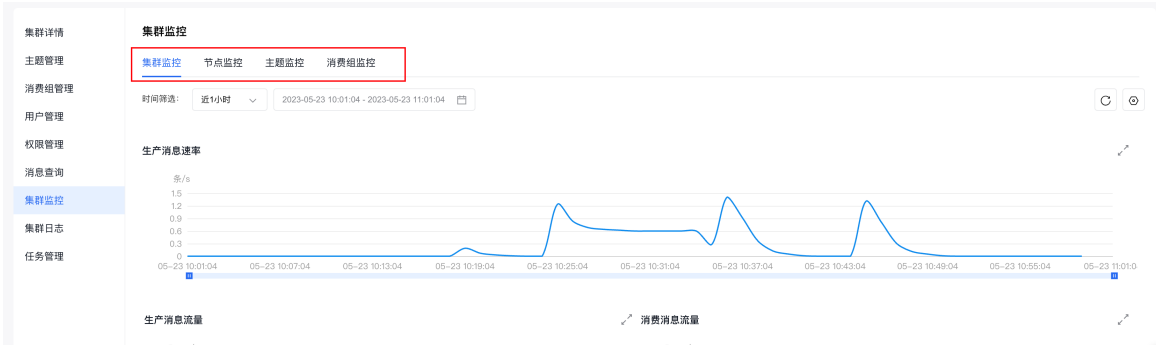
（2）页面跳转后，进入左侧边中的**集群详情**页面。

（3）点击左侧边栏中的**集群监控**，进入**集群监控**页面。



（4）通过查看**集群监控**页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



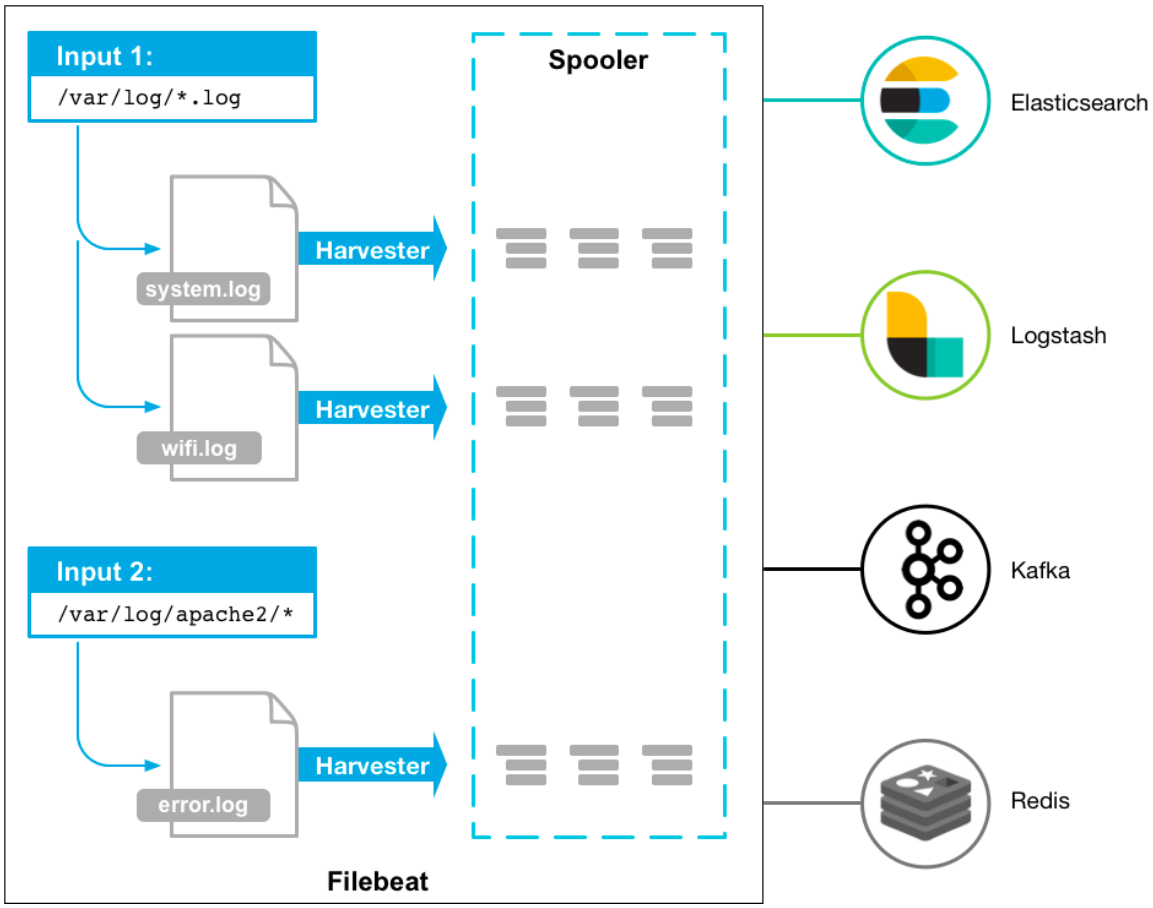
Filebeat接入Kafka专享版

Filebeat

Filebeat是用于转发和集中日志数据的轻量级传送工具。Filebeat监视您指定的日志文件或位置，收集日志事件，并将它们转发到Elasticsearch或 Logstash进行索引。

Filebeat的工作方式如下：启动Filebeat时，它将启动一个或多个输入，这些输入将在为日志数据指定的位置中查找。对于Filebeat所找到的每个日志，Filebeat都会启动收集器。每个收集器都读取单个日志以获取新内容，并将新日志数据发送到libbeat，libbeat将聚集事件，并将聚集的数据发送到为Filebeat配置的输出。

工作流程图如下：



前提条件

- 下载并安装Filebeat（参见Download Filebeat）
- 下载并安装1.8或以上版本 JDK
- 已创建消息服务 for kafka集群

Filebeat接入

步骤一：获取接入点

具体请参考查看集群接入点

SSL/SASL方式下载证书参考：如何下载证书？。

步骤二：创建/获取主题

具体请参考创建主题、查看主题详情

步骤三：Filebeat发送消息

准备配置文件

进入 Filebeat 的安装目录，创建配置监控文件 filebeat.yml，参数详情可参考Filebeat官网。

```
 filebeat.inputs:

- input_type: log

##### 此处为监听文件路径
 paths:
   - /var/log/*.log
##### ---------------------------- Kafka output --------------------------------
output.kafka:
 # 是否启用
 enabled: true

 # The list of Kafka broker addresses from which to fetch the cluster metadata.
 # The cluster metadata contain the actual Kafka brokers events are published
 # to
 # 获取集群元数据的 Kafka 代理地址列表（接入点）
 hosts: ["x.x.x.x:9095","x.x.x.x:9095","x.x.x.x:9095"]

 # 用于生成事件的 Kafka 主题, 可以使用任何事件 field 的格式字符串, 若要从文档类型设置主题，请使用 `%{[type]}`.
 topic: test

 # 设置sasl协议字段
 sasl.mechanism: "SCRAM-SHA-512"

# 设置 SSL协议字段
#SSL/SASL_SSL方式请下载证书文件： kafka-key.zip
#ssl.certificate_authorities:[/***/client_ssl.properties]
 partition.random:
   # If enabled, events will only be published to partitions with reachable leaders. Default is false.
   # reachable_only 设置为true，则事件将仅发布到可用的分区
   # 必须是 random, round_robin, hash 三种的一种
   # 默认为 false
   reachable_only: false

 # Authentication details. Password is required if username is set.
 # 身份验证的细节。如果设置了用户名，则需要密码。
 # 没有设置可保持空值
 username: ''
 password: ''

 # Sets the output compression codec. Must be one of none, snappy and gzip. The default is gzip.
 # 设置输出压缩编解码器
 # 必须是 none, snappy 和 gzip 中的一个
 # 默认是 gzip
 compression: none

 # Set the compression level. Currently only gzip provides a compression level between 0 and 9. The default value is
```

```
  # Set the compression level. Currently only gzip provides a compression level between 0 and 9. The default value is chosen by the compression algorithm.
  # 设置压缩级别
  # 目前只有 gzip 提供 0 到 9 之间的压缩级别
  # 默认值由压缩算法选择
  compression_level: 4

  # The maximum permitted size of JSON-encoded messages. Bigger messages will be dropped. The default value is 1000000 (bytes). This value should be equal to r less than the broker's message.max.bytes.
  # json编码消息的最大允许大小，更大的消息将被删除。默认值是 1000000（字节）。这个值应该等于 r，小于 broker 的 message.max.bytes
  max_message_bytes: 1000000
  # The ACK reliability level required from broker. 0=no response, 1=wait for local commit, -1=wait for all replicas to commit. The default is 1.  Note: If set to 0, no ACKs are returned by Kafka. Messages might be lost silently on error.
  # 代理要求的ACK可靠性级别
  # 0=无响应，1=等待本地提交，-1=等待所有副本提交
  # 默认值是1
  # 注意:如果设置为0,Kafka不会返回任何ack。出错时，消息可能会悄无声息地丢失。
  required_acks: 1
  # The configurable ClientID used for logging, debugging, and auditing purposes.  The default is "beats".
  # 可配置的ClientID，用于日志记录、调试和审计。默认是“beats”。
  client_id: beats
```

Filebeat 发送消息

在filebeat安装路径执行如下命令

```
./filebeat -e -c filebeat.yml
```

步骤四：Filebeat消费消息

准备配置文件

```
filebeat.inputs:
- type: kafka
  hosts:
    - ip:端口
    - ip:端口
    - ip:端口

  # 身份验证的细节。如果设置了用户名，则需要密码。
  # 没有设置可保持空值
  username: ""
  password: ""
  #Topic的名称。
  topics: ["filebeat_test"]
  #Group的名称。
  group_id: "filebeat_group"
  #证书所在位置
 #SSL/SASL_SSL方式请下载证书文件： kafka-key.zip
 #ssl.certificate_authorities:[/***/client_ssl.properties]

  ssl.verification_mode: none

output.console:
  pretty: true
```

Filebeat消费消息

在filebeat安装路径执行如下命令

```
./filebeat -c ./input.yml
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端查看jar包运行日志。

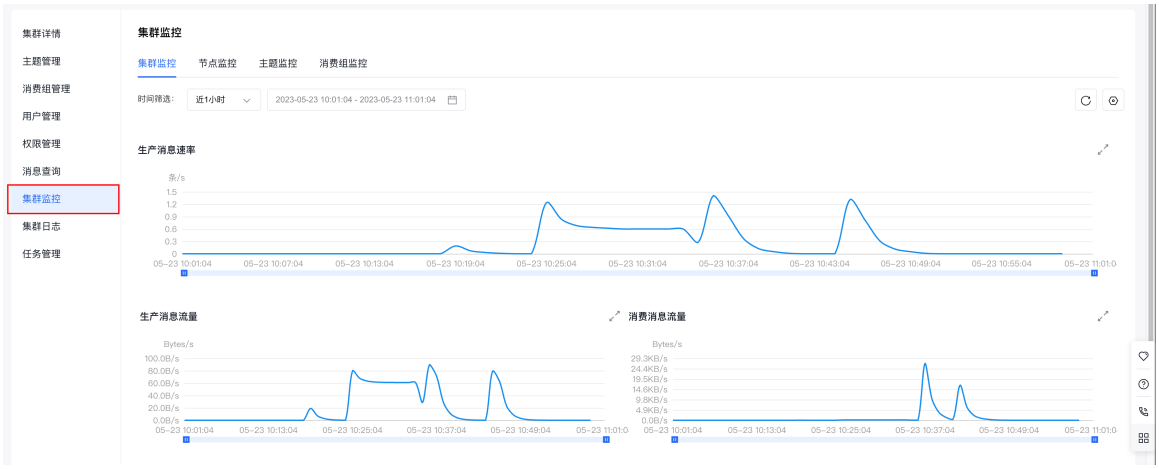
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入**集群详情**页面。

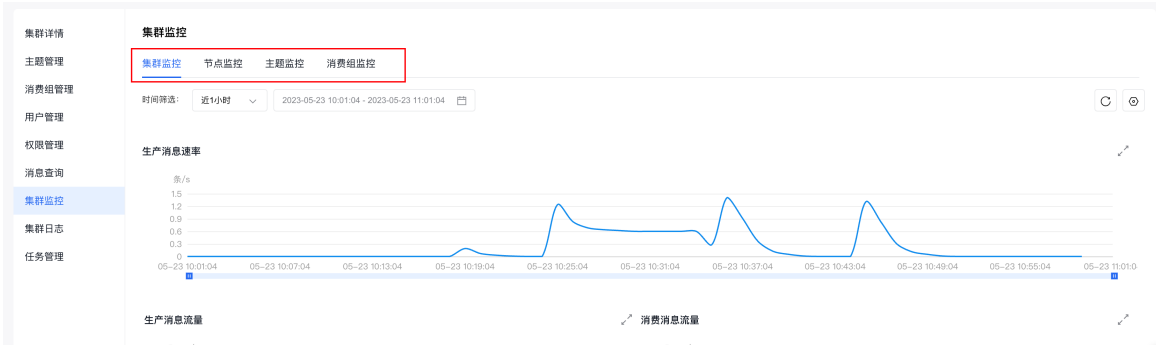
（2）页面跳转后，进入左侧边中的**集群详情**页面。

（3）点击左侧边栏中的**集群监控**，进入**集群监控**页面。



（4）通过查看**集群监控**页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



Flink接入Kafka专享版

接入准备

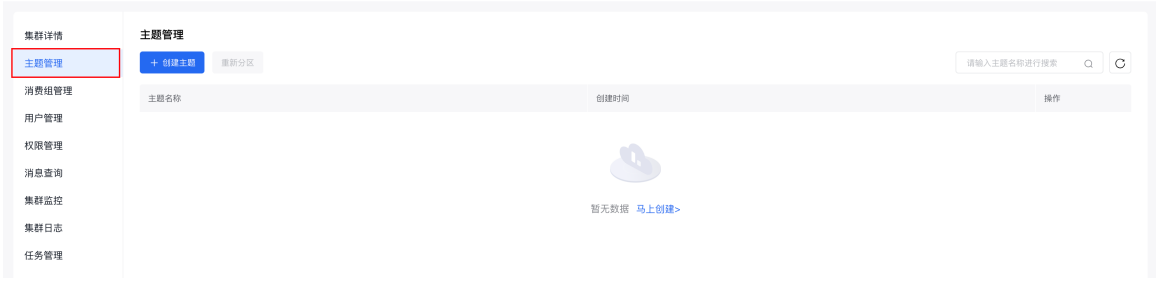
1. 购买专享版消息服务for Kafka集群

开通消息服务 for Kafka服务后，在控制台页面点击**创建集群**，即可进行购买，详情可参考创建集群。

2. 为购买的集群创建主题

在控制台页面点击集群名称，进入集群详情页面。

在左侧的边栏中点击**主题管理**，进入主题管理页面。



在主题管理页面点击**创建主题**，进行主题的创建，详情参考创建主题。

接入步骤

步骤一：获取集群接入点

具体请参考：查看集群接入点。

步骤二：添加Maven配置

```
<?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>flink_sdk</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <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-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>org.example.KafkaConsumerDemo</mainClass>
                                </transformer>
                                <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
                                    <resource>reference.conf</resource>
                                </transformer>
                            </transformers>
                        </configuration>
```

```
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

</project>
```

步骤三：编写测试代码

- 需要关注并自行修改的参数

参数名	含义
access_point	接入点信息
topic	主题名称
value	消息的具体内容
group_id	消费组id

生产者示例代码

创建KafkaProducerDemo.java文件，具体代码示例如下：

```
package org.example;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;

public class KafkaProducerDemo {
    public static void main(String args[]) throws Exception {

        //接入点设置
        String access_point = "x.x.x.x:9092,x.x.x.x:9092,x.x.x.x:9092";
        //填写需要发送消息的主题名称
        String topic = "topic_name";
        //填写需要发送消息
        String value = "test flink message";
        // 创建配置类
        Properties props = new Properties();;
        // 指定kafka服务端所在位置
        props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, access_point);
        // Kafka消息的序列化方式。
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        // 请求的最长等待时间。
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        // 设置客户端内部重试次数。
        props.put(ProducerConfig.RETRIES_CONFIG, 5);
        // 设置客户端内部重试间隔。
        props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        // 构造 Producer 对象
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<String, String>(props);

        try {
            // 测试发送 100 条消息
            for (int i = 0; i < 100; i++) {
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<>(topic, value + ": " + i);
                kafkaProducer.send(kafkaMessage, (RecordMetadata recordMetadata, Exception e) -> {
                    if (e == null) {
                        System.out.println("send success:" + recordMetadata.toString());
                    } else {
                        e.printStackTrace();
                        System.err.println("send fail");
                    }
                });
            }
        } catch (Exception e) {
            System.out.println("Something error has happened");
            e.printStackTrace();
        } finally {
            kafkaProducer.close();
        }
    }
}
```

消费者实例

创建KafkaConsumerDemo.java文件，具体代码示例如下

```
package org.example;

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

        //接入点设置
        String access_point = "x.x.x.x:9092,x.x.x.x:9092,x.x.x.x:9092";
        String group_id = "flink_group";

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // 创建配置类
        Properties properties = new Properties();
        // 设置接入点
        properties.setProperty("bootstrap.servers", access_point);
        // 设置消费组id
        properties.setProperty("group.id", group_id);
        // FlinkKafkaConsumer的第一个参数填创建的topic名称
        DataStream<String> stream = env
                .addSource(new FlinkKafkaConsumer<>("topic_name", new SimpleStringSchema(), properties));
        // 打印输出
        stream.print();
        env.execute();
    }
}
```

步骤四：编译并运行

通过maven工具将上述代码打包后，上传至指定的服务器，并执行以下命令：

```
java -jar flink_sdk-1.0-SNAPSHOT.jar
```

步骤五：查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端查看jar包运行日志。

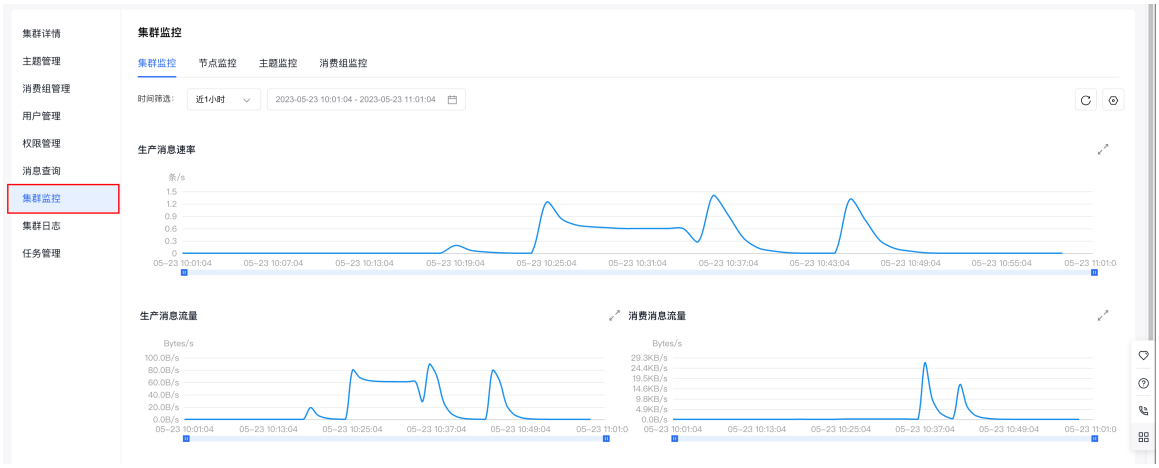
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入**集群详情**页面。

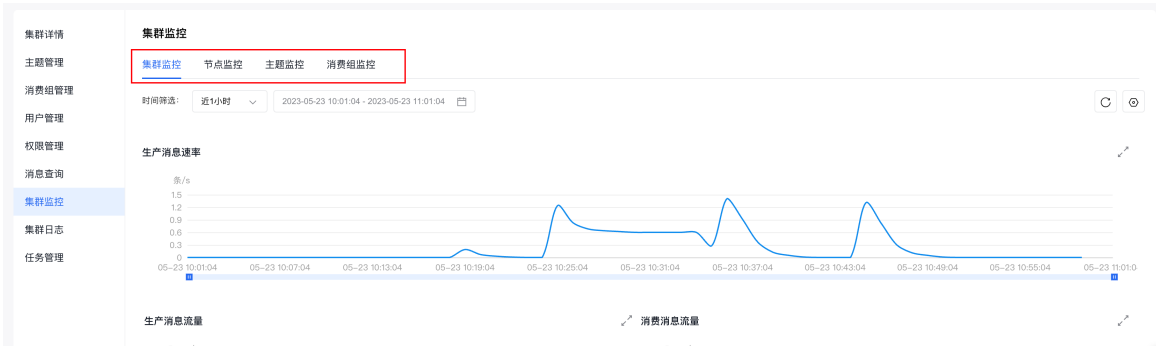
（2）页面跳转后，进入左侧边中的**集群详情**页面。

（3）点击左侧边栏中的**集群监控**，进入**集群监控**页面。



（4）通过查看**集群监控**页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



Logstash接入Kafka专享版

前提条件

- 已创建消息服务 for kafka集群
- 下载并安装Logstash。具体操作，请参见Download Logstash。
- 下载并安装JDK 8。具体操作，请参见Download JDK 8。

Logstash接入

步骤一：获取接入点

具体请参考查看集群接入点

步骤二：创建/获取主题

具体请参考创建主题、查看主题详情

步骤三：下载证书文件

SSL/SASL方式下载证书参考：如何下载证书？。

Logstash 发送消息

准备配置文件

创建jaas.conf配置文件

输入以下内容。

```
KafkaClient {
org.apache.kafka.common.security.plain.PlainLoginModule required
# 用户管理中创建用户的用户名
username="XXX"
# 用户管理中创建用户的密码
password="XXX";
};
```

创建 output.conf 配置文件

输入以下内容。

```
input {
  stdin{}
}

output {
  kafka {
      bootstrap_servers => "接入点"
      topic_id => "topic名称"
       # 固定为 SSL
      security_protocol => "SSL"
      # jaas.conf 文件地址
      jaas_path => "/home/kafka/logstash-7.10.2/jaas.conf"
      # 配置为 client.truststore.jks 文件路径
      ssl.truststore.location =>  "/home/kafka/logstash-7.10.2/client.truststore.jks"
      #  固定为 bms@kafka
      ssl.truststore.password => "bms@kafka"
      # 配置为 kafka_client_jaas.conf 文件路径
      ssl.keystore.location => "/home/kafka/logstash-7.10.2/client.keystore.jks"
      # 设置为client_ssl.properties文件内ssl.keystore.password
      ssl.keystore.password => "qKiHmaov8GNR"
      # SSL接入点辨识算法。6.x及以上版本Logstash需要加上该参数。空值
      ssl_endpoint_identification_algorithm => ""
  }
}
```

Logstash发送消息

在安装目录下bin执行以下命令

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

Logstash 消费消息

准备配置文件

创建jaas.conf配置文件

输入以下内容。

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

创建 input.conf 配置文件 输入以下内容。

```
input {
  kafka {
      bootstrap_servers => "接入点"
      # topic名称
      topics => ["logstash_test"]
      # 固定为 SSL
      security_protocol => "SSL"
      # jaas.conf 文件地址
      jaas_path => "/home/kafka/logstash-7.10.2/jaas.conf"
      # 配置为 client.truststore.jks 文件路径
      ssl.truststore.location =>  "/home/kafka/logstash-7.10.2/client.truststore.jks"
      #  固定为 bms@kafka
      ssl.truststore.password => "bms@kafka"
      # 配置为 kafka_client_jaas.conf 文件路径
      ssl.keystore.location => "/home/kafka/logstash-7.10.2/client.keystore.jks"
      # 设置为client_ssl.properties文件内ssl.keystore.password
      ssl.keystore.password => "qKiHmaov8GNR"

      group_id => "logstash_group"
      consumer_threads => 3
      auto_offset_reset => "earliest"
  }
}

output {
  stdout {
      codec => rubydebug
  }
}
```

Logstash消费消息

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

查看集群监控

查看消息是否发送成功或消费成功有两种方式：

1. 在服务器端查看jar包运行日志。

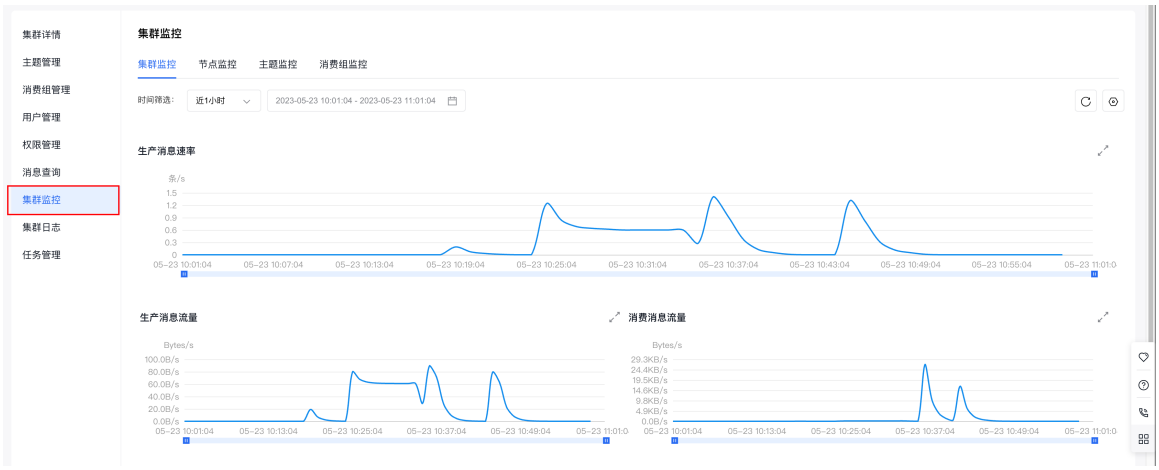
2. 在专享版消息服务 for Kafka控制台查看集群监控，获取集群生产、消息情况。

推荐使用第二种方式，下面介绍如何查看集群监控。

（1）在专享版消息服务 for Kafka的控制台页面找到需要连接的集群，点击集群名称进入**集群详情**页面。

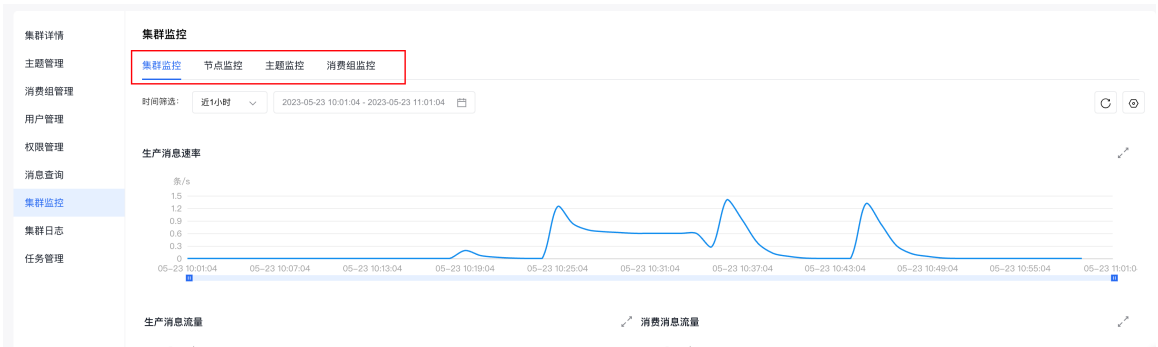
（2）页面跳转后，进入左侧边中的**集群详情**页面。

（3）点击左侧边栏中的**集群监控**，进入**集群监控**页面。



（4）通过查看**集群监控**页面，提供的不同纬度的监控信息（集群监控、节点监控、主题监控、消费组监控），即可获知集群的生产和消费情况。

集群监控的具体使用请参考：集群监控



共享版

产品描述

介绍

概述

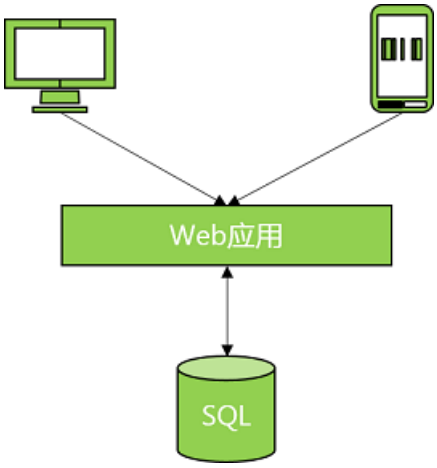
百度消息服务（Baidu Messaging System BMS）是全兼容Apache Kafka的分布式、高可扩展、高通量的托管消息队列服务，您

可以直接享用Kafka带来的先进功能而无需考虑集群运维，并按照使用量付费。

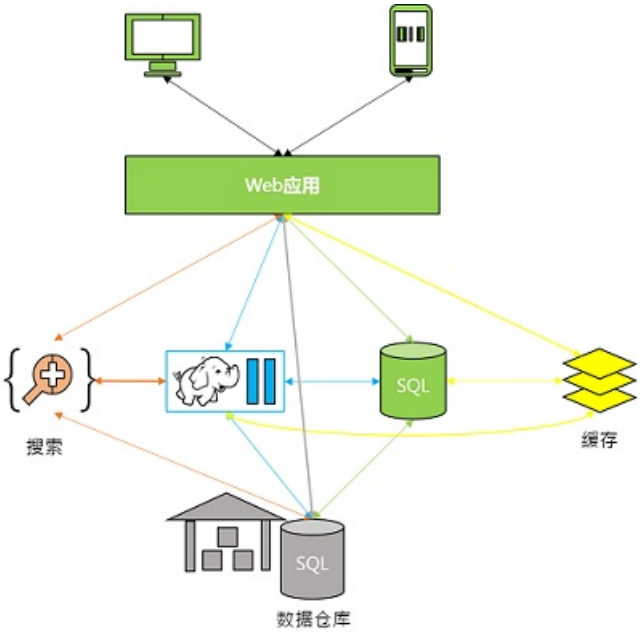
Kafka在网络中的位置和作用

本章节以开发新一代互联网应用的发展历程为例，介绍Kafka在网络中的位置和作用。

- 第一阶段，首次搭建应用网络如下：
  - Web应用部署在云服务器上，为个人电脑或移动用户提供访问服务。
  - SQL数据库为Web应用提供数据持久化和数据查询。



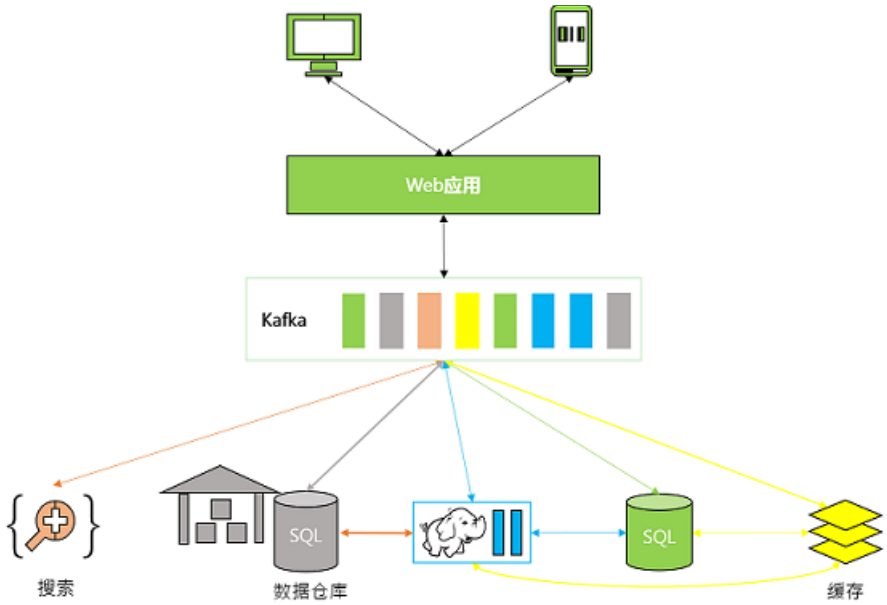
- 第二阶段：基于业务的迅速发展，网络扩容如下：
  - 增加缓存服务，从而降低SQL数据库的荷载。
  - 搜集日志保存至Hadoop做离线处理，从而更加理解用户行为。
  - 数据汇总至数据仓库，从而获取交互式报表。
  - 加入实时模块和外部数据交互等等。



网络扩容后的问题如下：

  - 不同系统之间的数据同步
  - 系统扩展问题

- 第三阶段：新增Kafka模块提供消息队列，Web应用数据只需向队列中添加数据，网络中的各组件从队列中依次读取数据并自行处理，如下图所示：



网络扩容带来的问题迎刃而解，而且降低了系统组网复杂度；降低编程复杂度，各个子系统不在是相互协商接口，各个子系统类似插口插在插座上，Kafka承担高速数据总线的作用。

Apache Kafka简介

Apache Kafka是分布式的流数据平台，具有三大特性：

- 提供Pub/Sub方式的海量消息处理。

  Kafka提供的Pub/Sub就是典型的异步消息交换，消息的发布者（Pub）只需指定消息的类别，而无需与订阅者（Sub）交互；订阅者只接受订阅的一个或几个类型的消息。发布者与订阅者的解耦可独立的扩展或修改接口两边的处理过程。例如，您可以为服务器日志或者物联网设备创建不同主题（Topic），之后数据便可持续发送到各个主题，后端数据仓库、流式分析或者全文检索等对接特定主题，无需关注服务器或者物联网设备。

  同时，Kafka将主题划分为多个分区（Partition），并根据分区规则选择消息存储的分区。若分区规则设置合理，则所有的消息被均匀的分布到不同的分区中，可实现负载均衡和水平扩展。

- 以高容错的方式存储海量数据流。

  多个订阅者可以从一个或者多个分区中同时消费数据，以支撑海量数据处理能力。同时为了保证数据的可靠性，Kafka为每个分区设置一个Leader，Leader负责所有的读写操作，Follower只与Leader同步消息。消息写入分区时，Leader为自己和Follower复制数据做多个备份，如果Follower失效，Kafka会从另外的Follower处同步Leader的历史消息；如果Leader失效，其他Follower成为新的Leader继续服务。

- 保证数据流的顺序。

  每个发布者发送到Kafka分区中的消息是确保顺序的，订阅者可以依赖这个承诺进行后续处理。顺序写磁盘的效率比随机写内存高，是Kafka高吞吐率的重要保证之一。

百度消息服务综述

百度消息服务是基于Apache Kafka的分布式、高可扩展、高通量、多分区和多副本的托管消息队列服务。百度消息服务封装了

Kafka集群细节，并以托管服务形式提供，您可以直接使用百度消息服务创建主题来集成大规模分布式应用，而无需考虑集群运维，仅按照使用量支付处理数据的费用。相对于传统的消息服务，百度消息服务有以下特点：

- 可通过分区（Partition）实现主题水平扩展。
- 分区分布在多个节点上以达到高通量。
- 通过消费者组（Consumer Group）来支持单个消费者以队列或者Pub/Sub形式的消息消费，或者多个消费者集群顺序消费消息。
- 生产者和消费者异步交互，而不需要彼此等待。

产品优势

- 一键部署：开通即使用百度消息服务，专注产品开发即可，无需耗费精力安装、部署、配置、调试和维护集群。
- 低廉价格：无需任何硬件软件投入，只需开通服务并且为使用的资源付费。由于与社区的Kafka兼容，迁移成本极低，且不用担心被技术锁定。
- 数据安全：消息中心仅支持SSL加密的数据传输，以保证数据在传输的过程中不被窃听或者篡改，保证客户数据的安全。
- 可靠耐用：独特的高可用特性设计，防止数据在应用程序故障、个别机器故障或设施故障时丢失。

业务场景

- 采集网站、设备或应用程序的海量用户浏览、点击、搜索等数据以便实时分析。
- 汇总分布式应用的遥感数据方便系统运维。
- 对接百度MapReduce提供的Spark Streaming实现实时流数据分析。

快速入门

操作流程

创建主题

1. 注册并登录百度智能云平台，具体操作请参考注册和登录。
2. 登录成功后，选择“产品服务 -> 消息服务 for Kafka”，点击“主题”后进入主题列表页。
3. 点击“创建主题”，设置主题名称和分区个数：
    - 若输入主题名称是“demo”，则系统会创建一个主题：<accountID>__demo。
    - 分区数目决定了本主题可以处理的消息通量（throughput），请根据业务需求选择。

4. 点击“确定”即创建了百度消息服务主题，可直接使用。

5. 可点击“查看证书”以查看/编辑该主题的证书权限。

6. 证书权限中显示“特权证书”、“普通证书”和“他人的证书”对该主题的权限。

> 特权证书：权限适用于所有主题的自有证书；
>
> 普通证书：权限适用于指定主题的自有证书；
>
> 他人的证书：用户可通过添加他人的证书，将自己主题的读/写权限授权给他人账号下，授权后他人将有权限读/写用户的主题。

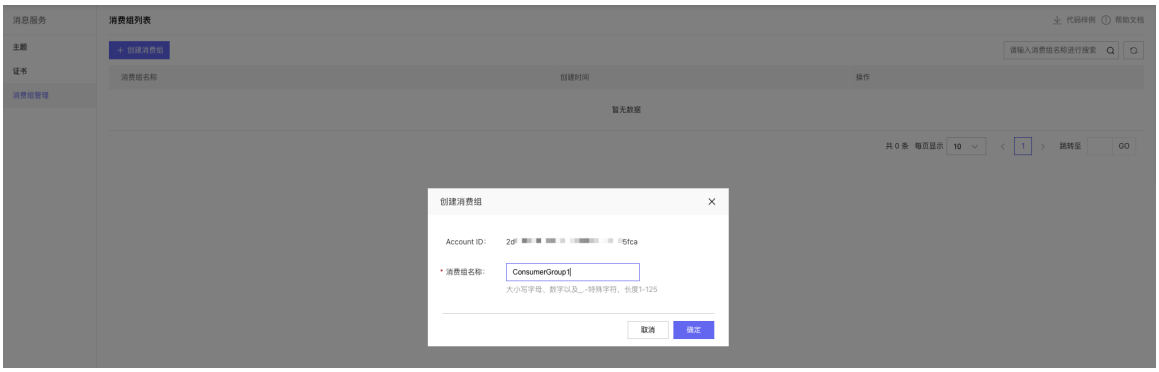
注意，当前百度消息服务的数据存储时间默认为24小时，过期后自动删除，若有特殊需求，请提工单联系我们。

创建消费组

消费组是Kafka提供的可扩展且具有容错性的消费者机制。通过console创建的消费组享有完善的安全保护机制，强烈推荐用户采用这种方式使用消费者组功能。

1. 注册并登录百度智能云平台，具体操作请参考注册和登录。
2. 登录成功后，选择“产品服务>百度消息服务”，点击“消费组管理”后进入列表页。
3. （可选）选择区域，请根据实际需求切换选择。
4. 点击“创建消费组”，输入消费组名称。

> 注：通过console创建的消费组会在名称前自动加上Account ID前缀，用户使用时需输入完整的消费组名称。



5. 消费组的使用权限与证书绑定。可点击“查看证书”以查看/编辑该消费组的证书权限。
6. 证书权限中显示“特权证书”、“普通证书”和“他人的证书”对该消费组的权限。用户使用消费组时，会对所使用的证书进行鉴权，验证是否有该消费组的使用权限。

> 注：对于未通过console创建的消费组不进行权限控制，但是可能存在数据安全问题，强烈建议用户通过console创建并进行使用。

下载证书

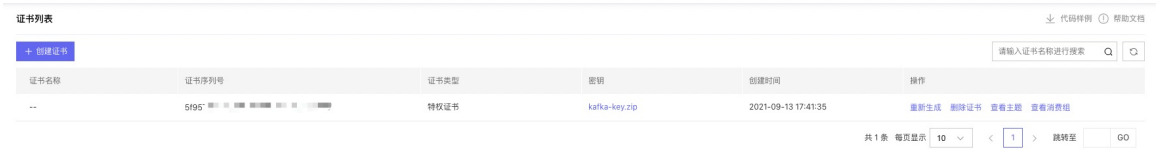
Kafka客户端在连接百度消息服务之前需要提供相应的证书来用以鉴权授权。请至百度智能云平台的“产品服务>百度消息服务-

证书”页下载用于认证客户和云服务的身份的证书：kafka-key.zip。

> 说明：
>
> - 请妥善保存证书，该证书用于建立SSL连接，同时用于认证客户端的AccountId。一旦丢失或泄漏，请至百度智能云平台重新生成。
> - 若使用香港的服务，请先提工单申请开通香港区域，通过后至香港区域下载对应的证书。
> - 各个区域的证书相互独立，不能够跨区域使用。
> - 广州和北京的证书，因历史原因可跨区域使用。

证书管理

证书列表中可查看证书的相关信息：名称、序列号、类型、密钥、创建时间；可选择“重新生成”更新证书，选择“删除证书”删除该证书；选择“查看主题”查看/编辑该证书对主题的权限，选择“查看消费组”查看/编辑该证书对消费组的权限。



连接服务

您可直接使用百度智能云提供的样例代码连接百度消息服务，也可从Kafka官网下载Kafka二进制包并逐步配置相关信息后连接百度消息服务。

通过样例代码连接服务

1. 下载代码样例。请登录控制台，打开“产品服务>百度消息服务-证书”页，点击页面右上角“代码样例”，下载至本地，下载后解压。
2. 百度消息服务提供的代码样例是Maven项目包。
3. 运行样例代码。`cd`至代码目录“sample-with-kafka-key”下，执行命令`run.bat <accountID>_demo`，执行完毕则建立了与百度消息服务的连接。

此外，我们提供Kafka服务的Python和Golang两种语言代码样例，已上传至GitHub中的Python样例和Golang样例，您可通过GitHub克隆代码至本地设计自己的程序。

逐步配置信息连接服务

1. 从Kafka官网下载Kafka二进制包：http://kafka.apache.org/downloads.html，目前已支持了0.10版本。
2. 通过Java访问Kafka服务的用户需使用Java客户端的pom依赖：kafka-client/0.10；我们推荐用户使用JDK 7 或JDK 8 版本，最低配置须为Java 1.7.0-b147。
3. 配置client.properties。请至百度智能云平台的百度“产品服务>百度消息服务-证书”页下载用于认证客户和云服务的身份的证书：kafka-key.zip，请将zip包解压到kafka二进制包的根目录下，client.properties，client.keystore.jks，client.truststore.jks都需在同级目录中。解压后打开client.properties，该文件是UTF8编码的文本文件，显示如下内容：

```
 security.protocol=SSL
ssl.truststore.location=client.truststore.jks
ssl.truststore.password=<your certificate password>
ssl.keystore.location=client.keystore.jks
ssl.keystore.password=<your certificate password>
```

4. 使用客户端代码中的脚本与百度消息服务通讯，连接通过SSL加密，保证数据传输无法被监听与篡改。以使用北京区域的百度消息服务为例，具体操作如下（操作前请确保client.properties，client.keystore.jks，client.truststore.jks都在kafka二进制包的根目录下）：

   1. 启动消费者。打开命令行工具并输入以下命令：

      ```
      sh bin/kafka-console-consumer.sh --topic <accountID>__demo --bootstrap-server kafka.bj.baidubce.com:9091 --consumer.config client.properties --from-beginning --new-consumer
      ```

      - 针对windows，启动命令如下：

        ```
        bin\windows\kafka-console-consumer.bat --topic <accountID>__demo --bootstrap-server kafka.bj.baidubce.com:9091 --consumer.config client.properties --from-beginning --new-consumer
        ```

   2. 发布消息。打开一个新的命令行工具并输入以下命令：

      ```
      sh bin/kafka-console-producer.sh --producer.config client.properties --topic <accountID>__demo --sync --broker-list kafka.bj.baidubce.com:9091
      ```

      - 针对windows，发布命令如下：

        ```
        bin\windows\kafka-console-producer.bat --producer.config client.properties --topic <accountID>__demo --broker-list kafka.bj.baidubce.com:9091
        ```

   3. 消息发送以后，消费者命令行中返回消息已经正确接收到信息。

      > 说明：
      >
      > - 北京区域对应的域名和端口号是：kafka.bj.baidubce.com:9091;
      > - 广州区域对应的域名和端口号是：kafka.gz.baidubce.com:9092;
      > - 苏州区域对应的域名和端口号是：kafka.su.baidubce.com:9091;
      > - 上海区域对应的域名和端口号是：kafka.fsh.baidubce.com:9091;
      > - 香港区域对应的域名和端口号是：kafka.hk02.hk.baidubce.com:9092;
      > - 保定区域对应的域名和端口号是：kafka.bd.baidubce.com:9071。

多用户访问控制

介绍

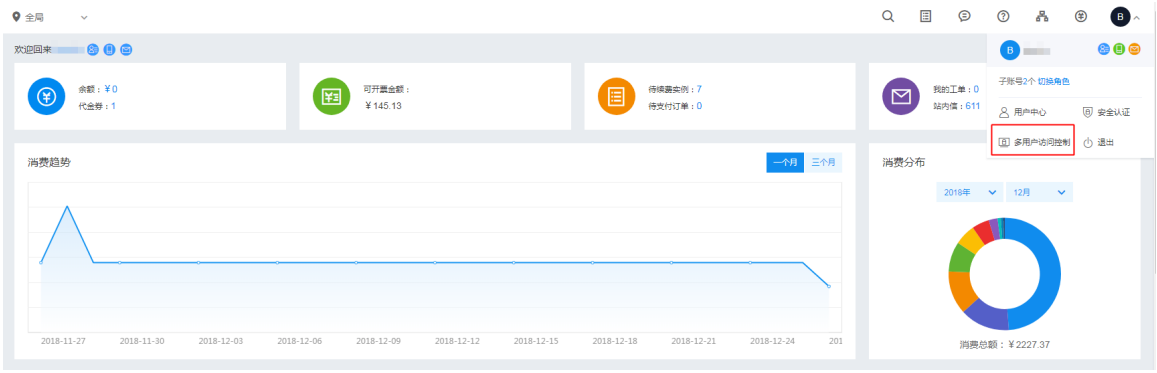
多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。

适用于下列使用场景：

- 中大型企业客户：对公司内多个员工授权管理；
- 偏技术型vendor或SAAS的平台商：对代理客户进行资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理。

创建用户

1. 主账号用户登录后在控制台选择“多用户访问控制”进入用户管理页面。



2. 在左侧导航栏点击“用户管理”，在“子用户管理列表”页，点击“新建用户”。

3. 在弹出的“新建用户”对话框中，完成填写“用户名”和确认，返回“子用户管理列表”区可以查看到刚刚创建的子用户。

权限策略配置

权限策略有两种类型：系统权限策略 和 自定义权限策略，分别实现BMS的产品级权限和实例级权限控制。

- 系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。
- 自定义策略：由用户自己创建，更细化的管理资源的权限集，可以针对单个实例配置权限，更加灵活的满足账户对不同用户的差异化权限管理。

系统权限策略

系统权限策略是被用户经常使用的权限集合，不可被用户编辑。Kafka默认提供了3种系统权限策略，供配置系统权限策略时选择，如下表：

策略名称	描述	包含权限
KAFKAReadPolicy	只读权限	topic查询操作权限；证书查询操作权限；
KAFKAWritePolicy	读写权限	topic查询操作权限；topic更新操作权限；证书查询操作权限；证书更新操作权限；
KAFKAFullControlPolicy	完全控制权限	kafka产品的全部权限；

自定义权限策略

自定义权限策略允许用户根据自己的需求灵活配置自己的权限策略；配置方式包含2中方式：策略生成器 和 编辑策略文件。

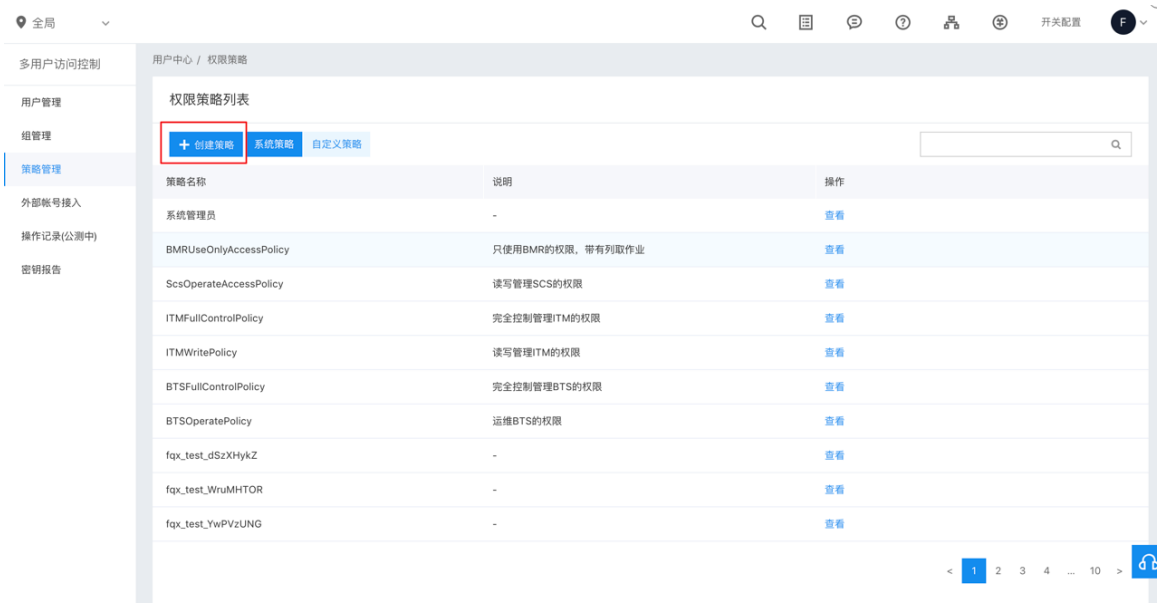
使用策略生成器配置权限

策略生成器，适合于需要控制资源访问的场景。kafka权限控制的资源包括topic和证书两种，使用策略生成器定义的策略，可以控制到某个具体要访问的资源；比如：可以设置用户允许下载哪些证书，也可以设置用户可以修改和删除哪些topic。

场景举例：配置用户可以下载的证书。

操作步骤：

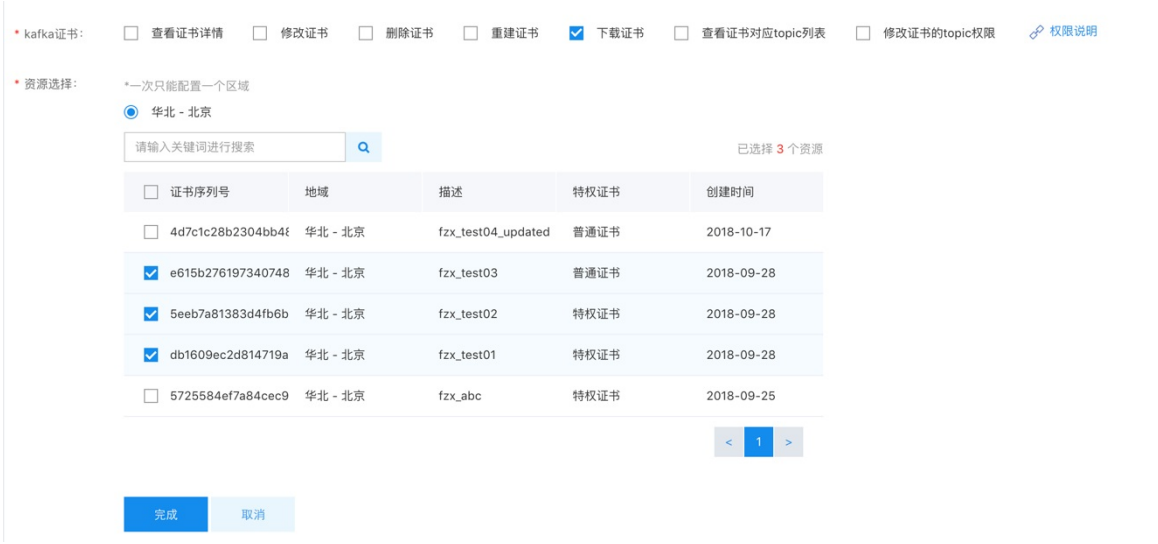
创建权限策略：多用户访问控制 -> 策略管理 -> 创建策略；



填写策略名称，选择服务类型为【百度消息服务 KAFKA】；

策略生成方式，选择【策略生成器】；

因为我们不配置针对topic的角色，所以不需要配置kafka主题的权限。直接配置kafka证书的权限。选择【下载证书】权限 -> 选择证书所在的地域 -> 选择要授权的证书；点击【完成】按钮后，完成策略配置；



此时从策略管理的【自定义策略】列表中可以看到刚刚定义的策略；

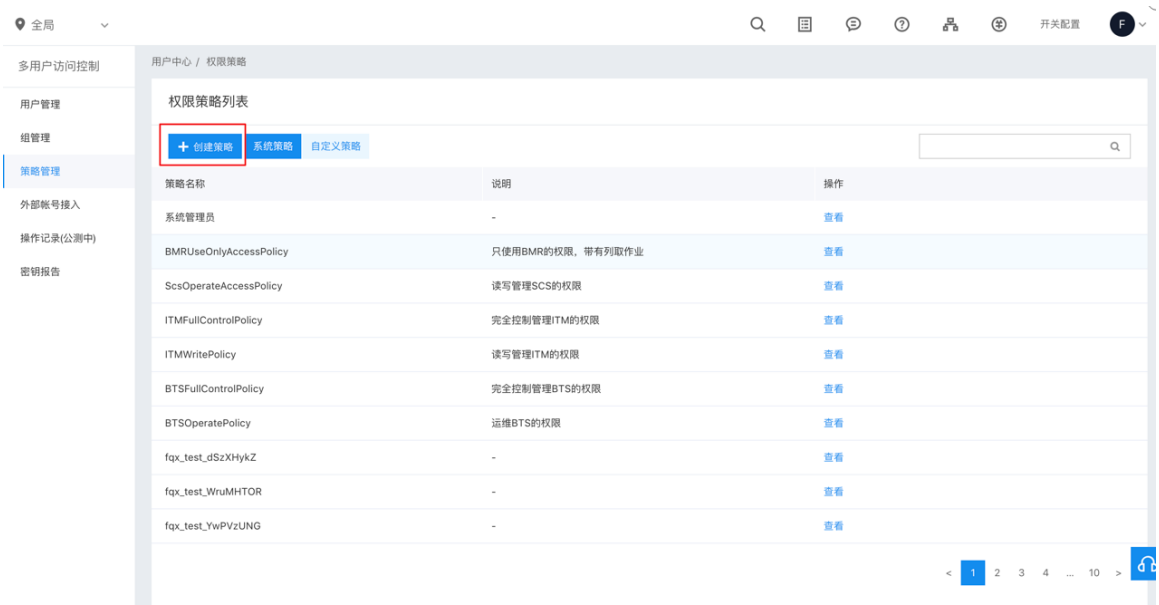
使用ACL策略文件配置权限

ACL策略配置文件，可以理解为配置权限策略规则的json字符串，无论是系统策略还是用户自定义策略，最终都会映射成为一个ACL的json串。使用ACL策略配置文件，用户可以非常灵活的定义权限策略，但是需要用户理解ACL字符串的含义。Kafka提供了一个公用的ACL配置字符串模板，用户可以参照模板配置自己的策略。比如：配置单独的一个创建topic的权限策略；编辑ACL权限策略的语法可参考文档策略语法。

场景举例：配置创建topic的权限策略，并授权给用户。

操作步骤：

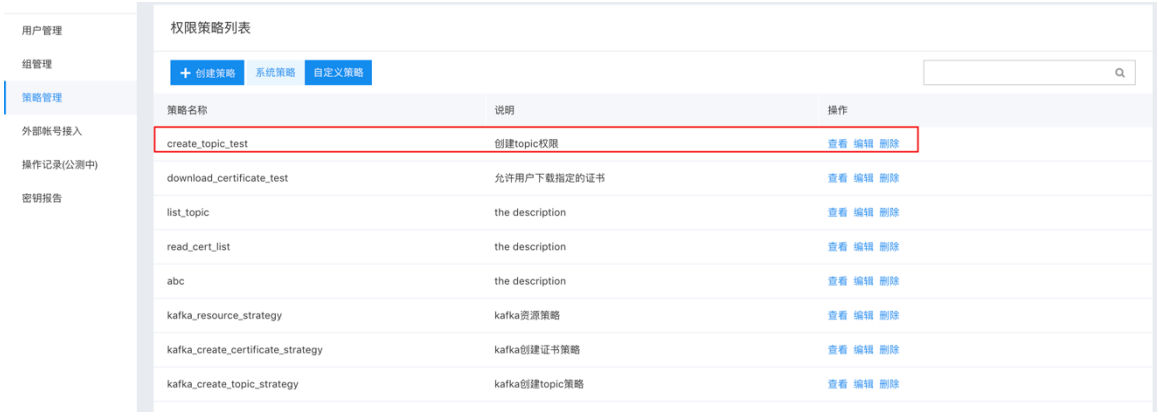
创建权限策略：多用户访问控制 -> 策略管理 -> 创建策略；



填写策略名称，选择服务类型为【百度消息服务 KAFKA】，策略生成方式，选择【编辑策略文件】，策略模板选择【kafka_acl】；

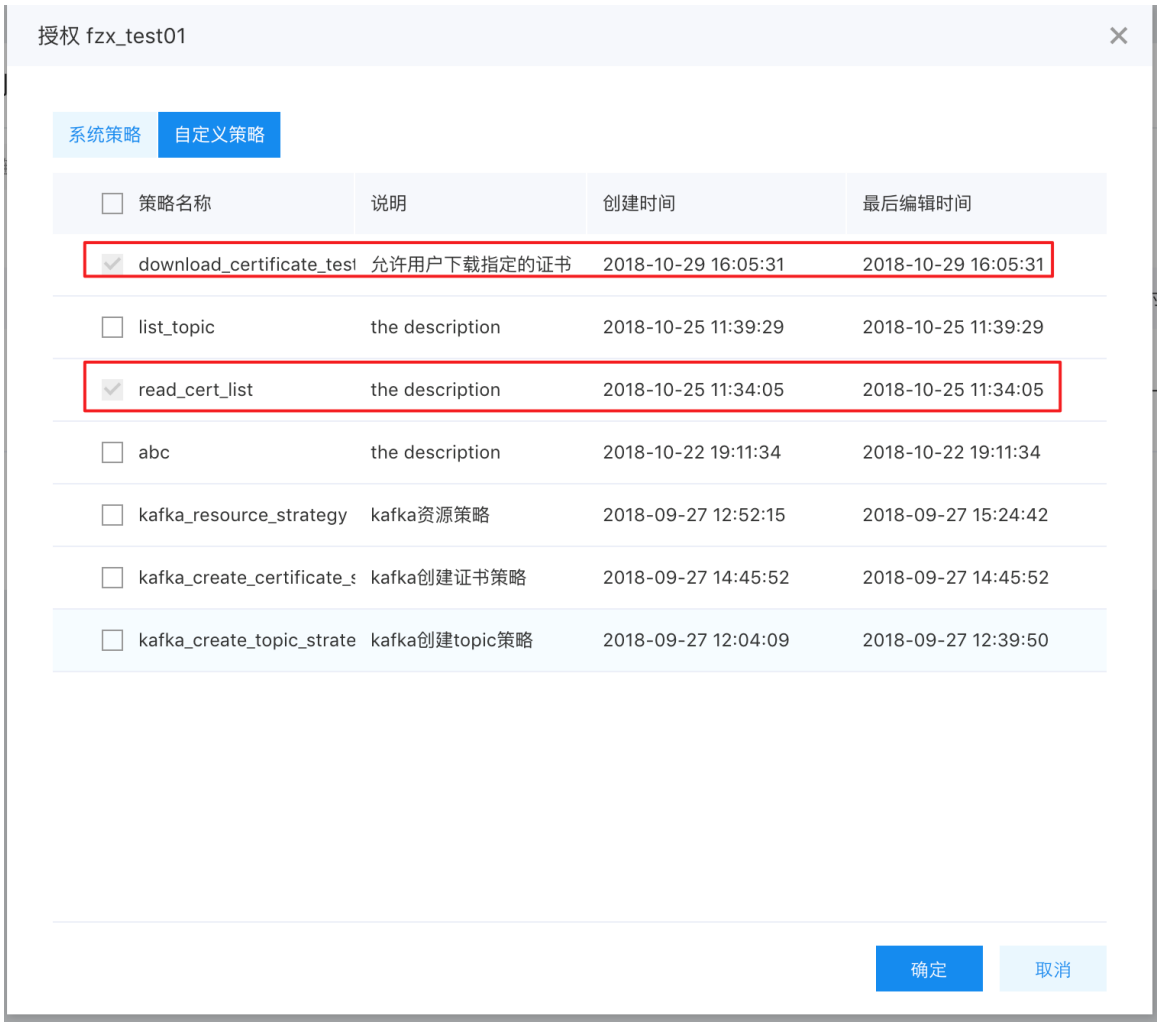
配置ACL字符串，包含创建topic权限，创建topic权限使用CreateTopic字符串标识；其它权限标签可参考下文的权限模型中的权限标签。

点击【完成】后，可以在【自定义策略】列表中查看到新建的策略；



用户授权

在“用户管理->子用户管理列表页”的对应子用户的“操作”列选择“添加权限”，并为用户选择系统权限或自定义策略进行授权。



> **说明：**如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

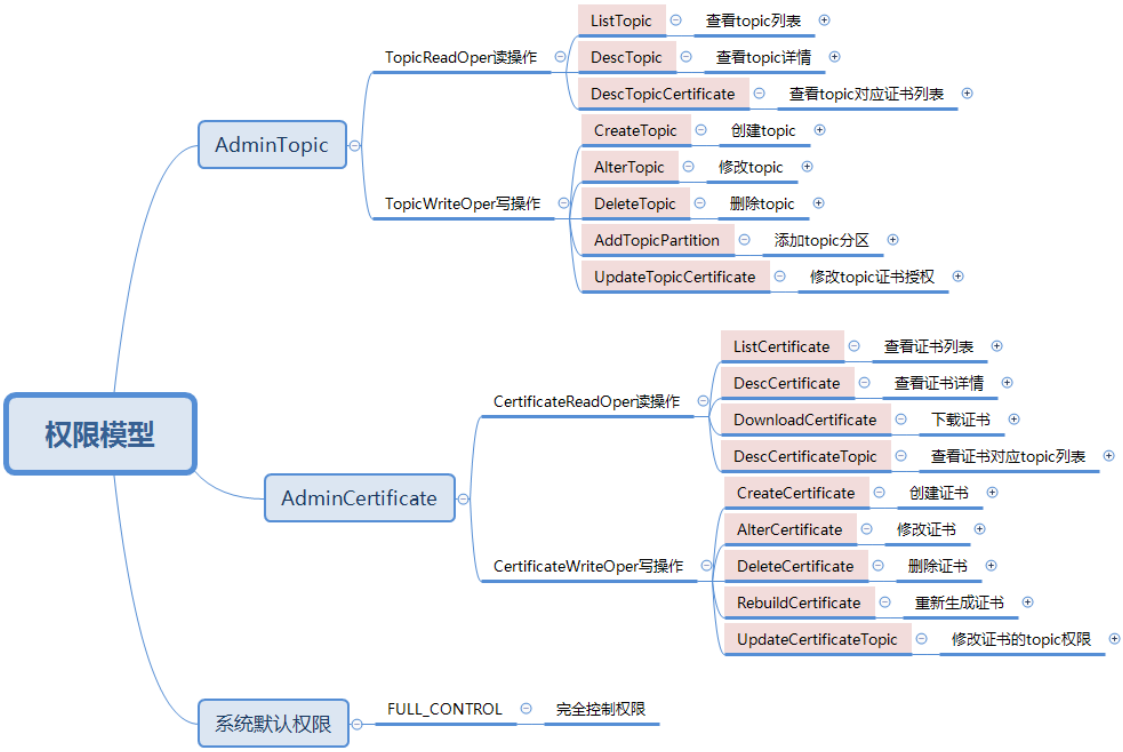
子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。

其他详细操作参考：多用户访问控制。

Kafka权限模型

kafka权限模型主要描述用户可以控制哪些操作的权限，以及控制权限的粒度。



- kafka权限模型涉及2种资源的权限管理：topic权限和证书权限；对topic权限的管理，即子用户对topic具有哪些操作的权限；同理，对证书权限的管理，即子用户对证书具有哪些操作权限；
- 如上图所示，整个权限模型分为3层，其中叶子节点为粒度最细的权限定义，每种权限都通过一个字符串标签唯一标识，用于在定义权限策略时区分权限。权限的含义如下表；

权限标签	层级	描述
FULL_CONTROL	1	等同于拥有『父账号』的全部权限；
AdminTopic	1	针对topic的增删改查权限，包含所有对topic操作的细粒度权限，比如：创建topic；
AdminCertificate	1	针对证书的增删改查权限，包含所有对证书操作的细粒度权限；
TopicReadOper	2	topic的查询操作权限，包含topic列表、详情等查询操作；
TopicWriteOper	2	topic的更新操作权限，包含topic的创建、修改、删除等操作；
CertificateReadOper	2	证书的查询操作权限，包含证书列表、下载证书等查询操作；
CertificateWriteOper	2	证书的更新操作权限，包含证书的创建、修改、删除、重新生成等操作；
ListTopic	3	查看topic列表权限，对应主题列表页面；
DescTopic	3	查看topic详情权限，对应主题列表中点击topic名称后进入的页面；
DescTopicCertificate	3	查看topic授权的证书列表权限，对应主题列表页面中【查看证书】操作；
CreateTopic	3	查看topic授权的证书列表权限，对应主题列表页面中【创建主题】操作；
AlterTopic	3	修改topic元信息权限，在目前的控制台中没有对应操作；
DeleteTopic	3	删除topic权限，对应主题列表页面中【创建主题】操作；
AddTopicPartition	3	为topic添加分区的权限，对应主题列表页面中【扩容分区】操作；
UpdateTopicCertificate	3	修改topic证书授权，对应主题列表页面中【查看证书】页面中对证书权限的编辑操作；
ListCertificate	3	查看证书列表权限，对应证书列表页面；
DescCertificate	3	查看证书详情权限，在目前的控制台中没有对应操作；
DownloadCertificate	3	下载证书权限，对应证书列表页面中的密钥下载链接；
DescCertificateTopic	3	查看证书对应topic列表，对应证书列表页面中的【查看主题】操作打开的页面；
CreateCertificate	3	创建证书权限，对应证书列表页面中的【创建证书】操作；
AlterCertificate	3	修改证书权限，对应证书列表页面中的修改证书名称操作；
DeleteCertificate	3	删除证书权限，对应证书列表页面中的【删除证书】操作；
RebuildCertificate	3	重新生成证书权限，对应证书列表页面中的【重新生成】操作；
UpdateCertificateTopic	3	修改证书可以访问的topic权限，对应证书列表页面中【查看主题】页面中对证书权限的编辑操作；

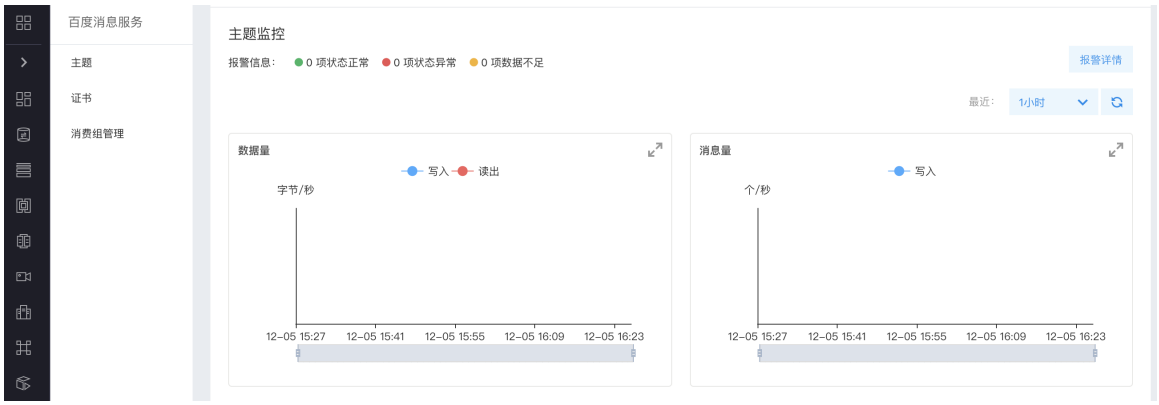
监控报警

查看监控

进入“产品服务>百度消息服务>主题”，点击要查看的主题名称，在详情页面中可看到监控数据图。

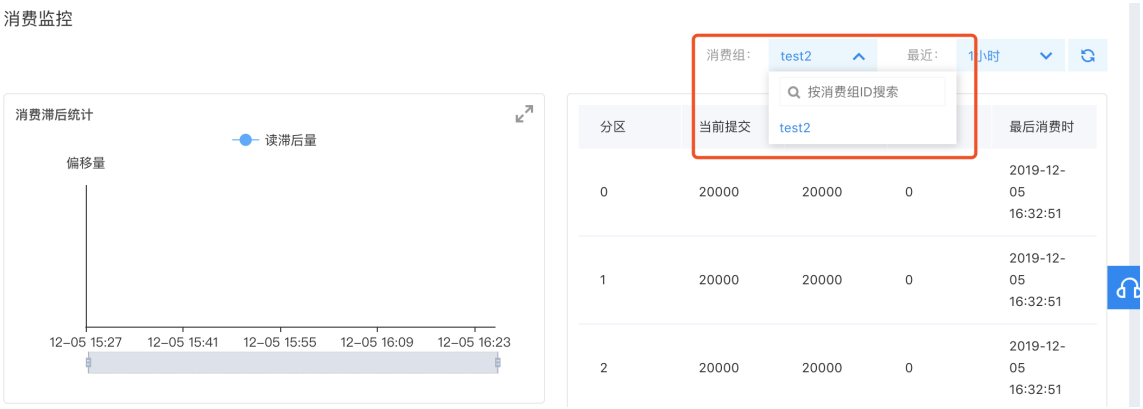
主题监控

可以查看该topic每秒读写的数据量（单位字节/秒），以及写入的消息个数（单位个/秒）



消费监控

在监控页面的右上角“消费组”处选择消费当前topic的消费组，若列表中没有所有查看的消费组，可通过输入消费组ID进行搜索查看。消费之后统计展示了该消费组的读滞后量，分区监控列表，详细展示了各个分区的消费信息。



报警管理

点击页面右上角的报警详情，可跳转至云监控BCM处管理报警。具体操作请参考监控云产品

> 注意：进行多用户访问控制时，若要允许子用户查看该监控页的内容，需为子用户赋予BCM权限

产品定价

名词解释

- 分区：分区是基本吞吐量单位，一个分区提供1MB/秒数据输入和2MB/秒输出通量。您可根据通量要求指定数据流所需的分区数量，并按小时对每个分区付费。运行时间不足1小时不收费。
- 记录：Kafka数据流的计费单元，以25KB有效载荷计算，以每百万记录收费，不足百万按照实际使用量收费。例如，5KB消息按照一个记录收费，49KB消息按照两个记录收费，1MB消息按照40个记录收费。读写消息均计入记录数。

关于百度消息服务的更多信息，请至产品描述和入门指南。

计费

百度消息服务服务采用后付费方式，根据您的实际使用量实时计费，即您只需为使用的分区和传输的消息付费。具体的计费价格请参见：价格详情页。

> 注意：topic创建后，即算作分区使用中，分区计费项将会产生费用。topic创建后，未进行消息传输，则消息传输计费项不会产生费用。
>
> 购买前需保证账户无欠款,且保证账户余额和可用代金券总和大于或等于50元。
> 消息传输按照分段阶梯累计计费。

计费公式

费用=单价×分区个数×使用时长+单价×传输的消息量

计费示例

某网站在非金融华中-武汉区域，一小时共有2亿条写请求、3亿条读请求，每条请求产生大小为1KB的消息写入百度消息服务。

1）计算每秒写入流量： 200000000 / 60 / 60 x 1 / 1024 = 55 (向上取整），即每秒写入流量约为55MB/s。

2）计算每秒读取流量： 300000000 / 60 / 60 x 1 / 1024 = 82 (向上取整），即每秒写入流量约为82MB/s。

3）根据一个分区提供1MB/秒数据输入和2MB/秒输出通量计算所需分区数量： max{55/1，82/2} = 55 ，即所需分区数量为55个。

4）计算消息传输所需费用：读+写请求共计5亿条，按照分段阶梯累计计算 0.007 x 20000000/1000000 + 0.006 x 300000000/1000000 = 3.2元

5）以华北区域为例，则一小时所需费用为 0.06 x 55 + 3.2 = 6.5，即每小时花费仅为6.5元。

余额不足提醒和欠费处理

余额不足提醒：

- 根据您最近3天的账单金额来判断您的账户余额（含可用代金券）是否足够支付未来3天的费用，若不足以支付，系统发送续费提醒。
- 根据您最近1天的账单金额来判断您的账户余额（含可用代金券）是否足以支付未来1天的费用，若不足以支付，系统发送续费提醒。

欠费处理：

- 北京时间整点检查您的账户余额是否足以支付本次Kafka账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。
    - 欠费后立即停服，系统会发送欠费停服通知。数据为您保留1天，期间继续计费，1天内未充值则释放资源。为了不影响您的服务，建议您在使用Kafka时要保证账户中的金额足够支付。

API文档

简介

Kafka API采用类似REST风格，提供对Kafka topic的创建与管理操作支持。如需基于topic进行读写操作，则可使用开源的Kafka客户端。

如果您是初次调用百度智能云产品的API，可以观看API入门视频指南，快速掌握调用API的方法。

通用说明

API调用遵循HTTP协议，各Region采用不同的域名，具体域名见服务域名。数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

实名认证

使用Kafka API的用户需要实名认证，没有通过实名认证的可以前往百度开放云官网控制台中的安全认证下的实名认证中进行认证。没有通过实名认证的用户请求将会得到一下错误提示码：

错误码	错误描述	HTTP状态码	中文解释
QualifyNotPass	The User has not pass qualify.	403	账号没有通过实名认证

API认证机制

所有API的安全认证一律采用Access Key与请求签名机制。 Access Key由Access Key ID和Secret Access Key组成，均为字符串。 对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version是正整数。
- timestamp是生成签名时的UTC时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。（本API签名仅使用host和x-bce-date两个header）
- signature是256位签名的十六进制表示，由64个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。

鉴权认证机制的详细内容请参见鉴权认证机制。

通信协议

Kafka API支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体,如:POST /v{version}/instance/{instanceId}
Query参数	URL中携带的请求参数
HEADER	通过HTTP头域传入,如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

响应结构说明

响应值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL参数	API版本号，当前值为1。	是

幂等性

当调用创建接口时如果遇到了请求超时或服务器内部错误，用户可能会尝试重发请求，这时用户通过clientToken参数避免创建出比预期要多的资源，即保证请求的幂等性。

幂等性基于clientToken，clientToken是一个长度不超过64位的ASCII字符串，通常放在query string里，如`http://bcc.bj.baidubce.com/v1/instance?clientToken=be31b98c-5e41-4838-9830-9be700de5a20`。

如果用户使用同一个clientToken值调用创建接口，则服务端会返回相同的请求结果。因此用户在遇到错误进行重试的时候，可以通过提供相同的clientToken值，来确保只创建一个资源；如果用户提供了一个已经使用过的clientToken，但其他请求参数（包括queryString和requestBody）不同甚至url Path不同，则会返回IdempotentParameterMismatch的错误代码。

clientToken的有效期为24小时，以服务端最后一次收到该clientToken为准。也就是说，如果客户端不断发送同一个clientToken，那么该clientToken将长期有效。

密码加密传输规范

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用AES 128位加密算法进行加密，用SK的前16位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

- byte[] bCiphertext= AES(明文,SK)
- String strHex = HexStr(bCiphertext)

日期与时间规范

日期与时间的表示有多种方式。为统一起见，除非是约定俗成或者有相应规范的，凡需要日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

1. 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日
2. 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
3. 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

规范化字符串

通常一个字符串中可以包含任何Unicode字符。在编程中这种灵活性会带来不少困扰。因此引入“规范字符串”的概念。一个规范字符串只包含百分号编码字符以及URI（Uniform Resource Identifier）非保留字符（Unreserved Characters）。

RFC 3986规定URI非保留字符包括以下字符：字母（A-Z，a-z）、数字（0-9）、连字号（-）、点号（.）、下划线（_）、波浪线（~）。

将任意一个字符串转换为规范字符串的方式是:

- 将字符串转换成UTF-8编码的字节流。
- 保留所有URI非保留字符原样不变。
- 对其余字节做一次RFC 3986中规定的百分号编码（Percent-Encoding），即一个%后面跟着两个表示该字节值的十六进制字母。字母一律采用大写形式。

示例：

原字符串：`this is an example for 测试`，对应的规范字符串：`this%20is%20an%20example%20for%20%E6%B5%8B%E8%AF%95`。

排版约定

排版格式	含义
< >	变量
[ ]	可选项
{ }	必选项
|	互斥关系
等宽字体Courier New	屏幕输出

服务域名

Kafka API的服务域名为：

Region	Endpoint	Protocol
北京	http://kafka-api.bj.baidubce.com/v1	HTTP and HTTPS
广州	http://kafka-api.gz.baidubce.com/v1	HTTP and HTTPS
苏州	http://kafka-api.su.baidubce.com/v1	HTTP and HTTPS
上海	http://kafka-api.fsh.baidubce.com/v1	HTTP and HTTPS
保定	http://kafka-api.bd.baidubce.com/v1	HTTP and HTTPS

公共头

公共请求头

下表为所有Kafka API所携带的公共头域（HTTP协议的标准头域不再这里列出）：

头域（Header）	是否必须	说明
Authorization	必须	包含Access Key与请求签名
Content-Type	可选	application/json; charset=utf-8
x-bce-date	可选	表示日期的字符串，符合BCE API规范

公共头域将在每个Kafka API中出现，是必需的头域。

公共响应头

下表为所有Kafka API的公共响应头域（HTTP协议的标准响应头域不再这里列出）：

头域（Header）	说明
Content-Type	只支持JSON格式，application/json; charset=utf-8
x-bce-request-id	Kafka后端生成，并自动设置到响应头域中

其中，“request id”使用UUID version4由Kafka服务生成。

错误返回

错误返回格式

当用户访问API出现错误时，会返回给用户相应的错误码和错误信息，便于定位问题，并做出适当的处理。请求发生错误时通过Response Body返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	表示具体错误类型
message	String	有关该错误的详细说明
requestId	String	本次请求的requestId

统一为如下格式（后续各接口不再单独列出）：

```
 {
   "requestId": "47e0ef1a-9bf2-11e1-9279-0100e8cf109a",
   "code":"NoSuchKey",
   "message":"The resource you requested does not exist"
}
```

其中，“code”为错误码，所有错误码取值来源BCE公共错误码和Kafka专有错误码（参考错误码部分内容）。

公共错误码

下表列出了百度智能云API的公共错误码。

错误码	错误消息	HTTP状态码	描述
AccessDenied	Access denied.	403Forbidden	无权限访问对应的资源。
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate forthis operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided doesnot exist in our records.	403Forbidden	Access Key ID不存在。
InvalidHTTPAuthHeader	The Access Key ID you provided does notexist in our records.	400 BadRequest	Authorization头域格式错误。
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等。
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配的问题，应定义更加具体的错误码，如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 BadRequest	JSON格式不合法。
InvalidVe		404	

rsion	The API version specified was invalid.	NotFound	URI的版本号不合法。
OptInRequired	A subscription for the service is required.	403Forbidden	没有开通对应的服务。
PreconditionFailed	The specified If-Match header doesn’tmatch the ETag header.	412PreconditionFailed	详见Etag。
RequestExpired	Request has expired. Timestamp date is <Data>.	400 BadRequest	请求超时。要改成x-bce-date。若请求中只有Date，需将Date转成datetime。
IdempotentParameterMismatch	The request uses the same client token asa previous, but non-identical request.	403Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check yourSecret Access Key and signing method. Consultthe service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致。

接口说明

Topic相关接口

创建Topic

描述

本接口用于创建一个Kafka topic。

请求

- 请求结构

```
 POST /v{version}/topic?clientToken={clientToken} HTTP/1.1
{
   "topicName": "topicName",
   "partitionCount": partitionCount,
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
clientToken	String	是	Query参数	幂等性Token，是一个长度不超过64位的ASCII字符串，详见链接
topicCreateParameters	TopicCreateParameters	是	RequestBody参数	创建topic参数

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
topicName	String	topic名称

错误码

错误码	HTTP状态码	中文解释
TopicAlreadyExist	409	topic名已经存在
TopicLimitExceeded	402	topic总数超过限制，最多只能创建20个
TopicPartitionLimitExceeded	402	单个topic分区总数超过限制，每个topic最多只能创建10个分区
InvalidTopicName	400	topic名不合法，只能包含大小写英文字母，数字，下划线_，横行-，不能使用双下划线__
InvalidTopicPartitionCount	400	分区数需要大于0

示例

- 请求示例

```
 POST /v1/topic?clientToken=16011728-8000-4ef4-9299-020db1424806 HTTP/1.1
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
{
  "topicName" : "demo",
  "partitionCount" : 1
}
```

- 返回示例

```
 ```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{"topicName":"demo"}
```
```

查询topic列表

描述

查询所有topic的详细信息。

请求

- 请求结构

```
GET /v{version}/topic?marker={marker}&maxKeys={maxKeys} HTTP/1.1
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
marker	String	否	Query参数	批量获取列表的查询的起始位置，是一个由系统生成的字符串
maxKeys	int	否	Query参数	每页包含的最大数量，最大数量通常不超过1000，缺省值为1000

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	获取下一页所需要传递的marker值，当isTruncated为false时，该域不出现
topics	List<Topic>	topic信息，由 Topic 组成的集合

错误码

错误码	HTTP状态码	中文解释
InvalidMarker	400	marker不能为空，且长度应小于topic名最大长度

示例

- 请求示例

```
GET /v1/topic?marker=demo&maxKeys=10 HTTP/1.1
authorization: bce-auth-v1/e9af9439-0c14-4b98-abae-ba14f11b25ec/2017-02-15T08:08:08Z/3600/host;x-bce-console-rpc-id;x-bce-date/69143dbee3abf6b6143c8266139645a19c192de6d25082fed3611ad2fe28b456/
x-bce-console-rpc-id: d4d32d1f-f2cd-41a8-bfb7-fdd65cdc6ed9
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-15T08:08:08Z
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Wed, 15 Feb 2017 08:08:09 GMT
Content-Type: application/json;charset=UTF-8

{
    "marker": "demo",
    "nextMarker": "test",
    "topics": [{
    "topicName": "demo",
    "partitionCount": 3,
    "replicationFactor": 3,
    "createTime": "2017-02-15T08:08:09Z"
    }],
    "isTruncated": false
}
```

查询topic详情

描述

查询单个topic的详细信息。

请求

- 请求结构

```
GET /v{version}/topic/{topicName} HTTP/1.1
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
topicName	String	是	URL参数	topic名

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
topic	Topic	返回的topic详情

错误码

错误码	HTTP状态码	中文解释
TopicNotFound	404	topic不存在

示例

- 请求示例

```
 GET /v1/topic/demo HTTP/1.1
authorization: bce-auth-v1/e9af9439-0c14-4b98-abae-ba14f11b25ec/2017-02-15T08:19:54Z/3600/host;x-bce-console-rpc-id;x-bce-date/6301919cc9ade759c88bfc47a7eee26b8347179f1c2fcd2808acfa4ab6d8b800
x-bce-console-rpc-id: 89ca2463-4d0d-453d-9b7c-4e4cf52214e7
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-15T08:19:54Z
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Wed, 15 Feb 2017 08:19:55 GMT
Content-Type: application/json;charset=UTF-8

{
    "topic": {
        "name": "demo",
        "partitionCount": 3,
        "replicationFactor": 3,
        "createdAt": "2017-02-15T08:19:55Z"
        }
}
```

删除topic

描述

删除一个指定的topic。

请求

- 请求结构

```
 DELETE /v{version}/topic/{topicName} HTTP/1.1
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
topicName	String	是	URL参数	待删除topic名

返回

- 返回头域

  除公共头域，无其它特殊头域

- 返回参数

  无特殊返回参数

错误码

错误码	HTTP状态码	中文解释
TopicNotFound	404	topic不存在

示例

- 请求示例

```
 DELETE /v1/topic/demo HTTP/1.1
authorization: bce-auth-v1/e9af9439-0c14-4b98-abae-ba14f11b25ec/2017-02-15T08:34:26Z/3600/host;x-bce-console-rpc-id;x-bce-date/fbfec8e0ab86dfdbd2bce5d353f7cdd0bec78303a86f1ca1d939a85dae2284d6
x-bce-console-rpc-id: 5bdd1701-cbb6-48cc-9819-a457a0475539
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-15T08:34:26Z
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Cache-Control: no-cache
```

权限管理相关接口

添加授权

描述

本接口用于给用户对某个kafka topic授权

请求

- 请求结构

```
 POST /v{version}/authorization/add  HTTP/1
{
   "topic": "topic",
   "accountId": "accountId",
    "certificateSN": "certificateSN",
    "certificateId": "certificateId",
    "topicOperation": "ReadWrite"
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
TopicAuthorizationParameters	TopicAuthorizationParameters	是	RequestBody参数	添加授权参数

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

无特殊返回参数

错误码

错误码	HTTP状态码	中文解释
InvalidRequestException	400	请求体中某些参数为空

示例

- 请求示例

```
 POST /v1/authorization/add
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

{
    "topic": "demo",
    "accountId": "accountId",
      "certificateSN": "certificateSN",
      "certificateId": "certificateId",
      "topicOperation": "ReadWrite"
}
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8
```

取消授权

描述

本接口用于针对自己的某个topic，取消对某个用户的授权

请求

- 请求结构

```
 POST /v{version}/authorization/cancel            HTTP/1.1

{
 "topic": "topic",
 "accountId": "accountId",
          "certificateSN": "certificateSN",
}
```

- 请求头域

除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
topic	String	是	RequestBody参数	topic名称，只能包含大小写英文字母，数字，下划线_，横行-，不能使用双下划线__
accountId	String	是	RequestBody参数	用户Id
certificateSN	String	是	RequestBody参数	证书序列号

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

  无特殊返回参数

错误码

错误码	HTTP状态码	中文解释
InvalidRequestException	400	topic参数为空

示例

- 请求示例

```
 POST /v1/authorization/cancel
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

{
    "topic": "demo",
    "accountId": "accountId",
      "certificateSN": "certificateSN",
}
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8
```

查询授权信息列表

描述

本接口用于按照不同的查询条件获取授权的topic列表

请求

- 请求结构

```
 POST /v{version}/authorization/list  HTTP/1.1

{
   "topicLike": "topicLike",
   "accountId": "accountId",
     "certificateSN": "certificateSN",
     "begin":
     "limit":
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
topicLike	String	是	RequestBody参数	topic名称，只支持模糊搜索
accountId	String	是	RequestBody参数	用户Id
certificateSN	String	是	RequestBody参数	证书序列号
limit	Integer	否	RequestBody参数	查询记录数
begin	Integer	否	RequestBody参数	查询起始索引

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
begin	int	查询起始索引
size	int	查询记录数
topicAuthList	List<TopicAuthorizationParameters>	由TopicAuthorization组成的集合

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/list
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

{
   "topic": "demo",
   "accountId": "accountId",
     "certificateSN": "certificateSN",
     "begin": 0,
     "limit": 10
}
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
   "begin":1,
   "size":5,
   "topicAuthList":[]
}
```

group对证书授权

描述

本接口用于group对证书授权

请求

- 请求结构

```
 POST /v{version}/authorization/consumergroup/add     HTTP/1.1

{
   "consumerGroupName": "consumerGroupName",
   "accountId": "accountId",
     "certificateId": "certificateId"
}
```

- 请求头域

除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
GroupAuthorizationParameters	GroupAuthorizationParameters	是	RequestBody参数	group授权参数

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

  无特殊返回参数

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/consumergroup/add
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

{
    "topic": "demo",
    "accountId": "accountId",
      "certificateSN": "certificateSN",
}
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8
```

取消group对证书授权

描述

本接口用于group对证书授权

请求

- 请求结构

```
 POST /v{version}/authorization/consumergroup/remove     HTTP/1.1

{
   "consumerGroupName": "consumerGroupName",
   "accountId": "accountId",
     "certificateId": "certificateId"
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
GroupAuthorizationParameters	GroupAuthorizationParameters	是	RequestBody参数	取消group授权参数

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

  无特殊返回参数

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/consumergroup/add
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

{
   "topic": "demo",
   "accountId": "accountId",
     "certificateSN": "certificateSN",
}
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8
```

查询group对证书授权

描述

本接口用于按照consumer group查询被授权的证书

请求

- 请求结构

`POST /v{version}/authorization/consumergroup/list?accountId={accountId}&certType={certType}&group={consumerGroupName}  HTTP/1.1`

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
accountId	String	否	Query参数	用户Id
certType	String	否	Query参数	证书类型
group	String	否	Query参数	consumer group名称

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

  无特殊返回参数

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/consumergroup/list?accountId=account1&certType=privilege&group=consumerGroupName1
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
    "authorizedCertificates":[
        "certificateId": "certificateDemo",
        "description": "descriptionDemo",
        "consumerGroupOperation": "Use"
    ]
}
```

证书管理相关接口

查询证书列表

描述

本接口用于查询证书列表

请求

- 请求结构

```
GET /v{version}/certificate/list?marker={marker}&maxKeys={maxKeys}
    HTTP/1.1
```

- 请求头域 除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
version	String	是	URL参数	API版本号
marker	String	否	Query参数	查询位置标记，批量获取列表的查询的起始位置，是一个由系统生成的字符串
maxKeys	Integer	否	Query参数	查询最大记录数,每页包含的最大数量，最大数量通常不超过1000，缺省值为1000

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
marker	String	标记查询的起始位置
isTruncated	boolean	true表示后面还有数据，false表示已经是最后一页
nextMarker	String	获取下一页所需要传递的marker值，当isTruncated为false时，该域不出现
certificates	List	certificates信息，由 certificate组成的集合

错误码

无

示例

- 请求示例

```
 GET /v1/authorization/certificate/list?marker=marker&maxKeys=10
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
        "marker": "demo",
         "nextMarker": "test",
         "isTruncated": false,
         "certificates":[],
}
```

获取certificate信息

描述

本接口用于获取certificate信息

请求

- 请求结构

```
POST /v{version}/certificate/detail/{certificateId}
```

HTTP/1.1

- 请求头域 除公共头域外，无其它特殊头域。
- 请求参数

参数名称	类型	是否必需	参数位置	描述
certificateId	String	是	URL参数	证书Id

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
certificate	certificate	certificates信息

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/certificate/detail/certificateId1
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
        "certificate":{
            "uuid": ,
            "accountId": ,
            "certificateCN": ,
            "createAt": ,
            "revokeAt": ,
            "certificate": ,
            "privateKey": ,
            "region": ,
            "description": ,
            "isPrivilege":}
 }
```

重新生成新的证书

描述

本接口用于重新生成新的证书

请求

- 请求结构

```
 POST /v{version}/certificate/recreate  HTTP/1.1
{
    "certificateSN":"certificateSN"
}
```

- 请求头域 除公共头域外，无其它特殊头域。
- 请求参数

参数名称	类型	是否必需	参数位置	描述
certificateSN	String	是	RequestBody	证书Id

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
certificate	certificate	certificates信息

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/certificate/recreate
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
        "certificate":{
           "uuid": ,
           "accountId": ,
           "certificateCN": ,
           "createAt": ,
           "revokeAt": ,
           "certificate": ,
           "privateKey": ,
           "region": ,
           "description": ,
           "isPrivilege":}
}
```

创建普通证书

描述

本接口用于查询证书列表

请求

- 请求结构

```
 POST /v{version}/certificate/create/ordinary  HTTP/1.1

{
   "description":"",
   "authInfos":[]
}
```

- 请求头域 除公共头域外，无其它特殊头域。
- 请求参数

参数名称	类型	是否必需	参数位置	描述
description	String	是	RequestBody参数	证书描述
authInfos	List<authInfo>	authInfos信息，由authInfo组成的集合		

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
certificate	certificate	certificates信息

错误码

无

示例

- 请求示例

```
 POST /v1/certificate/create/ordinary
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

 {
    "description":"",
    "authInfos":[
       {
          "topic": "demo",
          "topicCluster": "KAFKA_01",
          "topicOperation":"Read"
       }
    ]
 }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
       "status":0,
       "msg":"success",
       "data":{
          "uuid": ,
          "accountId": ,
          "certificateCN": ,
          "createAt": ,
          "revokeAt": ,
          "certificate": ,
          "privateKey": ,
          "region": ,
          "description": ,
          "isPrivilege":}
}
```

创建特权证书

描述

本接口用于创建特权证书

请求

- 请求结构

```
 POST /v{version}/certificate/create/privilege  HTTP/1.1

{
   "description":"description",
   "topicOperation":"Read"
}
```

- 请求头域 除公共头域外，无其它特殊头域。
- 请求参数

参数名称	类型	是否必需	参数位置	描述
description	String	否	RequestBody参数	证书描述
topicOperation	String	否	RequestBody参数	授权，如果不指定，则没有权限

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
certificate	certificate	certificates信息

错误码

无

示例

- 请求示例

```
 POST /v1/certificate/create/privilege
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

 {
     "description":"description",
     "topicOperation":"Read"
 }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
       "status":0,
       "msg":"success",
       "data":{
            "uuid": ,
            "accountId": ,
            "certificateCN": ,
            "createAt": ,
            "revokeAt": ,
            "certificate": ,
            "privateKey": ,
            "region": ,
            "description": ,
            "isPrivilege":}
}
```

删除证书

描述

本接口用于删除证书

请求

- 请求结构

```
POST /v{version}/certificate/delete  HTTP/1.1
  {
     "uuid":"uuidDemo"
  }
```

- 请求头域 除公共头域外，无其它特殊头域。
- 请求参数

参数名称	类型	是否必需	参数位置	描述
uuid	String	是	RequestBody	证书序列号

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

无特殊返回参数

错误码

无

示例

- 请求示例

```
 POST /v1/certificate/delete
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

  {
     "uuid": "uuidDemo"
  }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8
{
        "status":0,
        "msg":"success",
        "data":true
 }
```

更改证书备注

描述

本接口用于更改证书备注

请求

- 请求结构

```
 POST /v{version}/certificate/update  HTTP/1.1
{
        "uuid": "uuidDemo"
        "description": "descriptionDemo"
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
uuid	String	是	RequestBody	证书序列号
description	String	是	RequestBody	证书描述

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
certificate	certificate	certificates信息

错误码

无

示例

- 请求示例

```
 POST /v1/certificate/update
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

    {
          "uuid": "uuidDemo"
          "description": "descriptionDemo"
    }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
       "status":0,
       "msg":"success",
       "data":{
            "uuid": ,
            "accountId": ,
            "certificateCN": ,
            "createAt": ,
            "revokeAt": ,
            "certificate": ,
            "privateKey": ,
            "region": ,
            "description": ,
            "isPrivilege":}
}
```

下载证书

描述

本接口用于下载证书

请求

- 请求结构

```
 POST /v{version}/certificate/download   HTTP/1.1
{
       "certificateSN": ""
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	是否必需	参数位置	描述
certificateSN	String	是	RequestBody	证书Id

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

返回证书文件

错误码

无

示例

- 请求示例

```
 POST /v1/certificate/download
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

 {
        "certificateSN": "certificateDemo"
 }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/octet-stream
```

下载sample样例文件

描述

本接口用于下载sample样例文件

请求

- 请求结构

`POST /v{version}/certificate/download/sample  HTTP/1.1`

- 请求头域
  除公共头域外，无其它特殊头域。
- 请求参数

无

返回

- 返回头域
  除公共头域，无其它特殊头域。
- 返回参数

返回sample样例文件

错误码

无

示例

- 请求示例

```
 POST /v1/authorization/certificate/download/sample
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/octet-stream
```

集群管理接口

获取用户创建集群的权限列表

描述

本接口用于获取用户创建集群的权限列表

请求

- 请求结构

`POST /v{version}/cluster/permission`
`HTTP/1.1`

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

无

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
clusterPermissions	List	clusterPermissions信息，由clusterPermission组成的集合

错误码

无

示例

- 请求示例

```
 POST /v1/cluster/permission
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
        "clusterPermissions":[{
        "accountId": "accountId",
        "cluster": "KAFKA_02",
        "isVisible":false},
        ]
}
```

Consumer Group管理相关接口

创建ConsumerGroup

描述

本接口用于创建ConsumerGroup

请求

- 请求结构

```
POST /v{version}/consumergroups/new/create HTTP/1.1
{
   "consumerGroupName":"consumerGroupName"
}
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	描述
consumerGroupName	String	consumerGroup名称

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
consumerGroupName	String	consumerGroup名称

错误码

无

示例

- 请求示例

```
 POST /v1/consumergroups/new/create
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

  {
     "consumerGroupName":"consumerGroupDemo"
  }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
      "consumerGroupName":"consumerGroupDemo"
}
```

ConsumerGroup列表

描述

本接口用于获取ConsumerGroup列表

请求

- 请求结构

`POST /v{version}/consumergroups/new/list?begin={begin}&limit={limit} HTTP/1.1`

- 请求头域
  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	描述
begin	url	检索记录开始索引
limit	url	要检索的记录数

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
begin	Integer	检索记录开始索引
limit	Integer	要检索的记录数
consumerGroups	List	consumerGroups信息，由consumerGroup组成的集合

错误码

无

示例

- 请求示例

```
 POST /v1/consumergroups/new/list?begin=0&limit=1
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

  {
       "consumerGroups":
       {
        "size":
        "begin":
        "consumerGroups":[]
       }
       ]
  }
```

获取Topic的groupId列表

描述

本接口用于获取Topic的groupId列表

请求

- 请求结构

```
  POST /v{version}/consumergroups/list  HTTP/1.1

 {
     "begin":1,
     "limit":10,
     "topicName":"toicDemo"
 }
```

- 请求头域

  除公共头域外，无其它特殊头域。

- 请求参数

参数名称	类型	描述
begin	requestBody	检索记录开始索引
limit	requestBody	要检索的记录数
topicName	requestBody	topic名称

返回

- 返回头域

  除公共头域，无其它特殊头域。

- 返回参数

参数名称	类型	描述
begin	Integer	检索记录开始索引
limit	Integer	要检索的记录数
consumerGroups	List	consumerGroups信息，由consumerGroup组成的集合

错误码

无

示例

- 请求示例

```
 POST /v1/consumergroups/list
authorization: bce-auth-v1/32d21dc5-b643-4769-864b-54e3a8868d89/2017-02-16T02:17:25Z/3600/host;x-bce-
console-rpc-id;x-bce-date/03a0254357eafd041eb61a2610e938904733cc06a0d3994a7985b62a284874bd
x-bce-console-rpc-id: 010ce7dc-e454-43ae-a950-75609b39c580
host: kafka-api.bj.baidubce.com
x-bce-date: 2017-02-16T02:17:25Z
content-type: application/json
user-agent: Jersey/2.9.1 (Apache HttpClient 4.3.3)
content-length: 53
connection: Keep-Alive
accept-encoding: gzip,deflate

    {
        "begin":1,
        "limit":10,
        "topicName":"toicDemo"
    }
```

- 返回示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
Cache-Control: no-cache
Server: BWS
Date: Thu, 16 Feb 2017 02:17:26 GMT
Content-Type: application/json;charset=UTF-8

{
        "begin": 0,
        "size": 10,
        "groupIds": ["group1","group2"]
}
```

模型定义

TopicCreateParameters

参数名称	类型	描述
topicName	String	topic名称，只能包含大小写英文字母，数字，下划线_，横行-，不能使用双下划线__
partitionCount	int	topic分区数，最少1个，最多10个

Topic

参数名称	类型	描述
topicName	String	topic名称
partitionCount	Int	分区数，topic包含的总分区数量，定义参见Kafka官网/Topics and Logs
replicationFactor	Int	副本数，每个分区的副本数量，定义参见Kafka官网/Replication
createTime	DateTime	topic创建时间

TopicAuthorizationParameters

参数名称	类型	描述
topic	String	topic名称，只能包含大小写英文字母，数字，下划线_，横行-，不能使用双下划线__
accountId	String	用户Id
certificateSN	String	证书序列号
certificateId	String	证书序列号（同certificateSN）
topicOperation	String	权限信息,Read,Write,ReadWrite

GroupAuthorizationParameters

参数名称	类型	描述
accountId	String	用户Id
certificateId	String	证书序列号（同certificateSN）
consumerGroupName	String	consumerGroup名称

clusterPermission

参数名称	类型	描述
accountId	String	用户Id
cluster	String	集群Id，集群标识，取值为KAFKA_01至KAFKA_10
isVisible	boolean	标识accountId对于当前集群是否有权限； true为有权限，false为无权限；

常见问题

常见问题总览

配置类问题

- 百度消息服务支持的哪一版的Kafka？
- 什么是主题（TOPIC）？必须要创建吗？
- 什么是分区（PARTITION）？
- 什么是客户端？

安全类问题

- 百度消息服务如何保证安全性？支持哪些安全特性？
- 百度消息服务如何保障数据可靠性？

配置类问题

百度消息服务支持的哪一版的Kafka？

Kafka是一个演化中的系统，客户端请选择0.10.1.1版本。

什么是主题（TOPIC）？必须要创建吗？

您需要先创建主题然后才能读写。主题是全局唯一的。

什么是分区（PARTITION）？

分区是Kafka用来水平扩展主题吞吐的设计，发布的消息将被写入不同分区，被若干消费者同时读取。本质上，主题的吞吐与分区个数成正比。

吞吐包含了生产者与消费者，所以分区个数计算公式是：max(t/p，t/c)，其中t、p、c分别代表期望的总通量、生产者在一个分区通量、消费者在一个分区通量。

由于消息是以队列的形式缓存在Kafka中，分区个数并不需要按照峰值来设置，而只要按照平均值来设置即可，当然缺点是延时会提高。

什么是客户端？

您可以通过Kafka客户端连接消息中心，并且把代码部署在多个百度智能云服务BCC或应用引擎BAE实例中生产或者消费消息。

安全类问题

百度消息服务如何保证安全性？支持哪些安全特性？

百度消息服务通过如下安全特性确保安全性：

- 客户鉴权：由于用户可以从互联网连接消息中心，需要对用户进行鉴权以获取相应的身份方便对主题进行读写。目前支持X509证书获取身份。
- 客户授权：鉴权之后得到身份，根据授权粒度，可以与不同的权限绑定。目前提供读写策略，拥有该用户账户下面所有主题的发布订阅权限。

- 传输安全：为了保障数据在传输时候不被监听以及篡改，在传输的时候使用SSL加密。

百度消息服务如何保障数据可靠性？

百度消息服务服务副本个数默认为3，数据保存24小时。