Apusic 金蝶天燕

生成时间：2026-05-07

安装部署

金蝶Apusic分布式消息队列V2.0.3

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 一. 基础介绍

金蝶Apusic分布式消息队列（简称ADMQ），提供了一个云原生消息和事件流平台，能够为实时事件和历史事件构建一个实时应用和数据基础设施，帮助用户轻松构建关键任务消息和流应用以及实时数据管道，使其专注于如何从实时数据中实现业务价值的最大化。

2 二. 基本概念

在正确使用应用服务器来部署、管理应用之前，需要先理解以下几个基本概念：

1. ADMQ 集群

一个ADMQ实例由一个或多个ADMQ集群组成。集群又由以下部分组成。

- 一个或者多个brokers
- 一个或者多个ZooKeeper协调器，用于集群级别的配置和协调
- 一组BookKeeper的Bookies用于消息的持久化存储
- 一个管理控制台，用于监控管理用户和相关资源的创建和分配，也是broker和bookeeper启动的依赖。

2. Broker

broker是一个无状态组件, 主要负责运行另外的两个组件:

- 一个 HTTP 服务器, 它暴露了 REST 系统管理接口以及在生产者和消费者之间进行 Topic查找的API。
- 一个调度分发器, 它是异步的TCP服务器，通过自定义二进制协议应用于所有相关的数据传输。
- 为了支持全局Topic异地复制，Broker会控制Replicators追踪本地发布的条目，并把这些条目用Java 客户端重新发布到其他区域。

3. 元数据存储

ADMQ使用Zookeeper进行元数据存储、集群配置和协调。在一个ADMQ实例中：

- 配置与仲裁存储: 存储租户，命名域和其他需要全局一致的配置项
- 每个集群有自己独立的ZooKeeper保存集群内部配置和协调信息，例如归属信息，broker负载报告等等。

4. 持久化存储

ADMQ为应用程序提供有保障的消息传递。如果一个消息成功地到达ADMQ Broker，它将被送达其预定的目标。为了提供这种保证，未确认送达的消息需要持久化存储直到它们被确认送达。ADMQ用BookKeeper作为持久化存储。BookKeeper是一个分布式的预写日志（WAL）系统。

5. 管理控制台

管理控制台是一个基于Web的GUI管理和监控工具，用于管理用户、权限、租户、命名空间、主题、订阅、broker、部署集群、插件，并支持多个环境的动态配置。

下表列出了金蝶Apusic分布式消息队列V2.0.3的默认管理值和各端口的默认值。

端口	对应组件	作用	访问范围
12306	admq-manager	管控台访问端口	外部访问

12305	admq-manager	license认证端口	内部访问
8080	broker	admin管理端口	外部访问
6650	broker	数据传输端口	外部访问
9092	broker	kafka插件服务端口	外部访问
3181	storage	数据存储端口	内部访问
8000	storage	监控数据采集端口	内部访问
2181	zookeeper	zookeeper服务端口	内部访问
2888	zookeeper	zookeeper服务端口	内部访问
3888	zookeeper	zookeeper服务端口	内部访问
9990	zookeeper	zookeeper admin管理端口	内部访问
18800	zookeeper	监控数据采集端口	内部访问

如果使用tls，需要开放 6651, 8081。

3 三. 管理控台快速安装与启停

3.1 3.1 安装前准备

- 获取安装包：从 http://www.apusic.com/下载金蝶 Apusic 分布式消息队列 V2.X 安装包，或从金蝶Apusic 分布式消息队列 V2.X 产品光盘中获得相应的安装包文件。
- 系统要求：Java环境（JDK1.8及以上版本）、内存（8GB+）、硬盘空间（64GB+）、浏览器（IE8及以上，FireFox，Chrome）
- 操作系统：Linux（国产操作系统: 如银河麒麟系列、中标麒麟系列、普华、中科红旗、深度、统信、凝思、欧拉等；RedHat 系列；CentOS系列；Suse Linux 系列）、Unix（HP Unix 系列；IBM AIX 系列；Solaris 系列）
- Java虚拟机：Oracle JDK 8+、IBM JDK 8+、Open JDK 8+、64-bit Open JDK 8+
- 数据库：MySQL、达梦、神通、人大金仓、H2
- 芯片：支持飞腾、龙芯、鲲鹏、申威、海光、合芯、兆芯、X86、ARM等

3.2 3.2 开始安装

3.2.1 3.2.1 管理控制台安装

解压安装包

```
mv admq-manager-V2.3.tar.gz /opt/
cd /opt
tar -zxvf admq-manager-V2.3.tar.gz
```

拷贝license授权文件到管控台licenses目录下

```
cp license.xml /opt/admq-manager-V2.3/licenses
```

修改数据库类型（可选）

默认使用H2数据库，如果正式环境部署需要换成其他数据库，例如MySQL。

/opt/manager/config/application.properties 文件中添加了多种数据库配置，可以把使用的数据库配置打开，注释掉其他不使用的数据库配置项。

3.2.2 3.2.2 启动管理控制台服务

```
cd /opt/admq-manager/
```

```
bin/admq-manager start
```

3.2.3 3.2.3 停止管理控制台服务

```
cd /opt/admq-manager/
bin/admq-manager stop
```

4 四. 管控基础使用介绍

4.1 4.1 访问

1、在浏览器中键入URL：https://ip:12306

2、在登录界面输入用户名（admq）,密码(11111111)进行登录，首次登录需要修改密码。

<assets/installing src="./assets/installing/shouye.png" />

4.2 4.2 基础功能

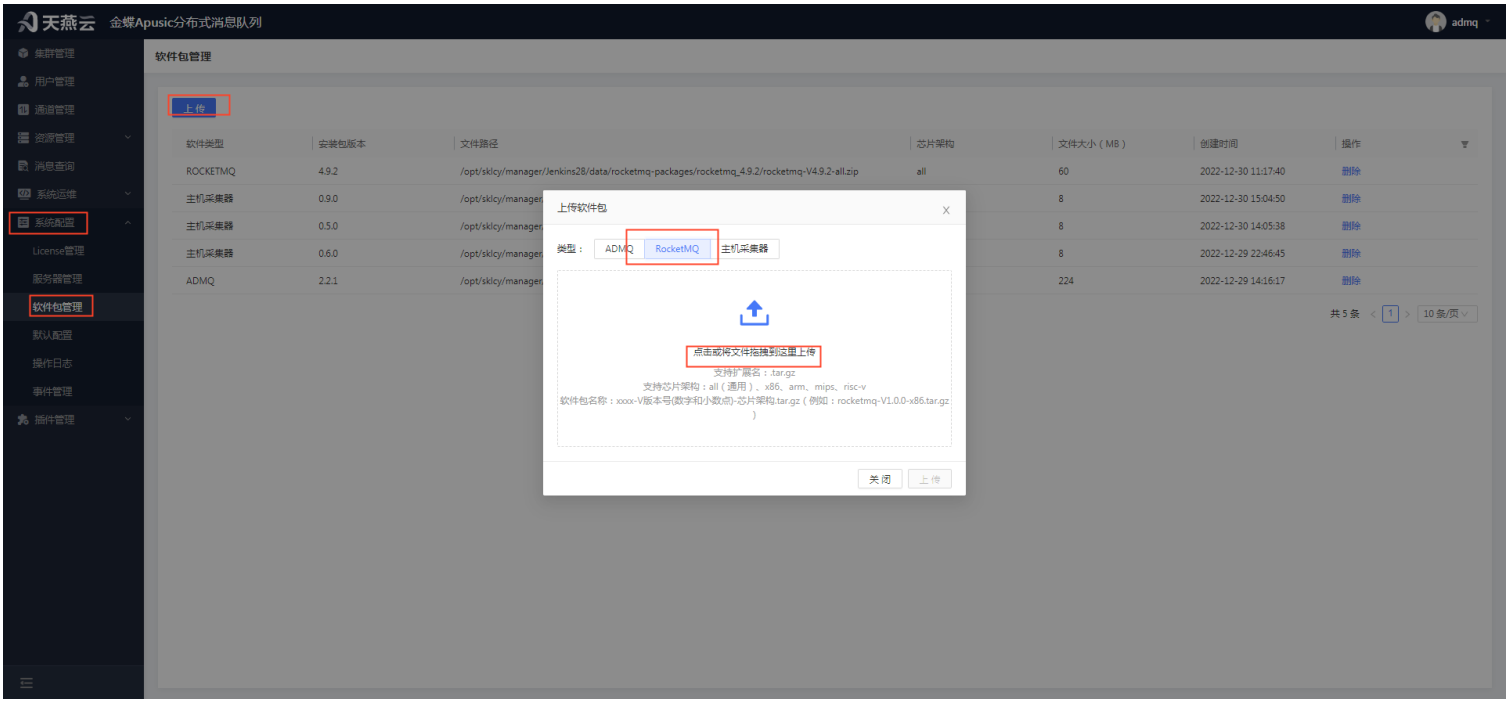
支持集群管理、用户管理、集群消息复制管理（通道管理）、资源管理、消息查询、系统运维、系统配置、插件管理

5 五. 部署集群前基础数据创建

5.1 5.1 上传软件包

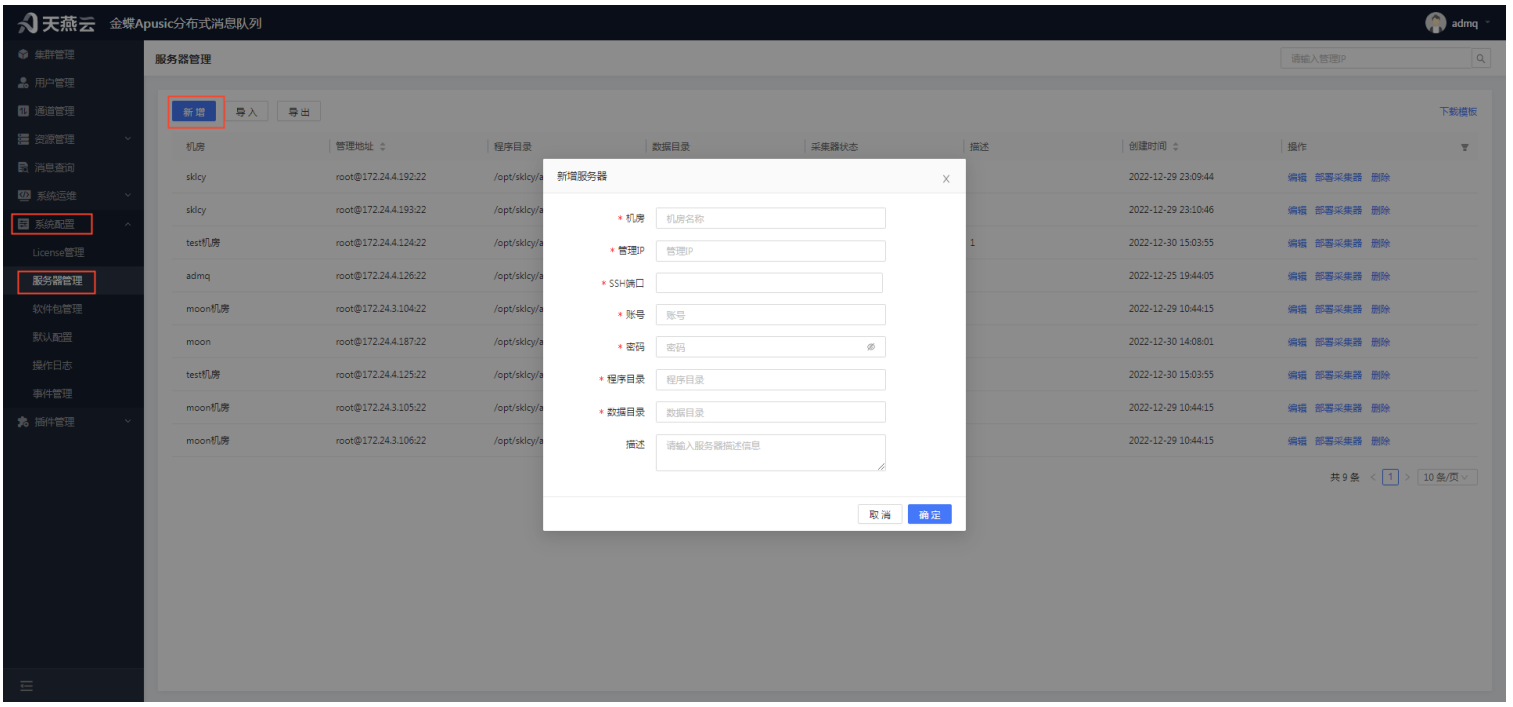
部署集群前，需要引用admq软件包信息，需要上传软件包信息。

进入系统配置->软件包管理->上传，将获取的admq-V2.2.1-all.tar.gz软件包或者rocketmq-V4.9.2-all.tar.gz软件包上传。



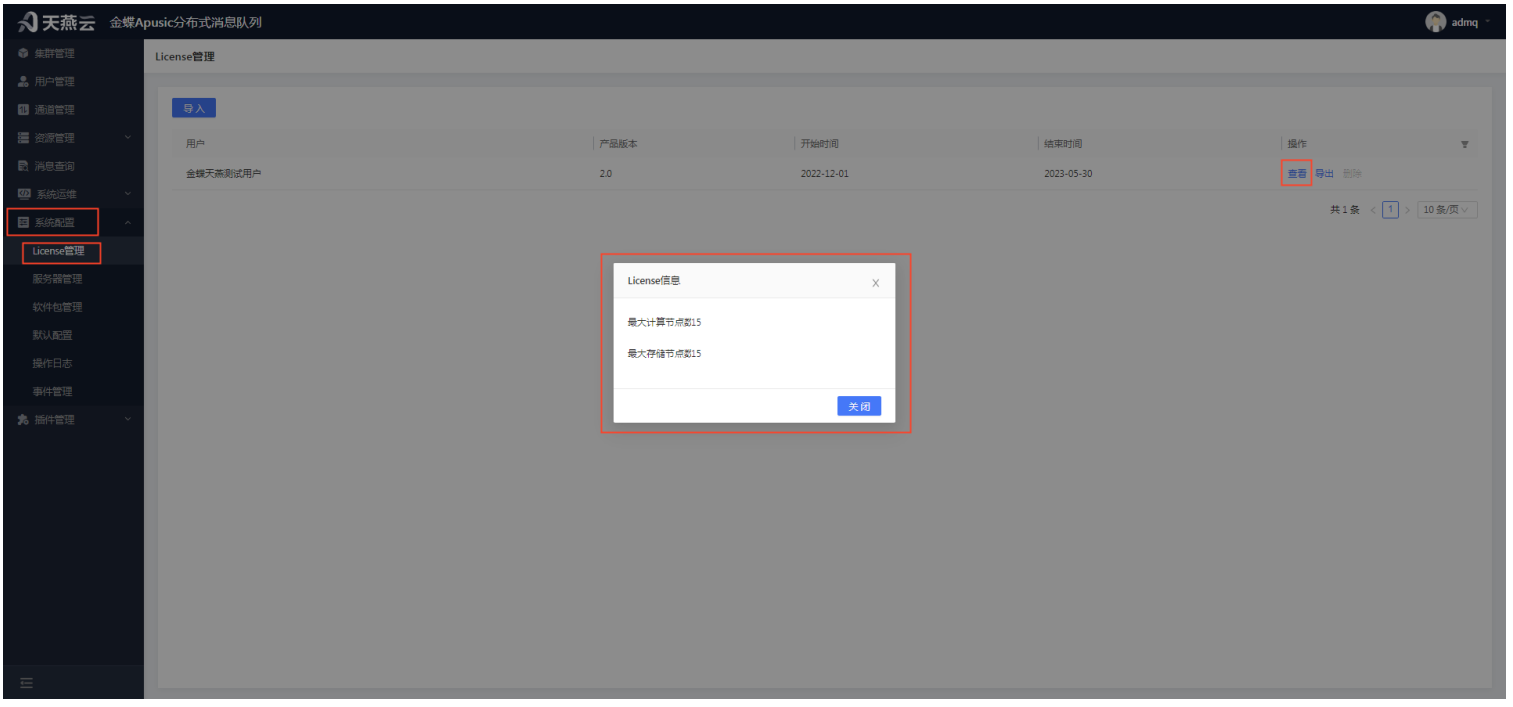
5.2 5.2 新增服务器信息

部署集群环境之时，需要选择计算程序地址、存储程序地址、本地协调器地址，所以需要新增服务器信息，待新建集群信息引用。部署单机环境时，只需要创建一个服务器信息即可，若需要创建集群环境，至少需要新建3个服务器信息作为集群的远程节点服务器（计算程序地址、存储程序地址、本地协调器地址）。



5.3 5.3 确认License中允许的节点数量

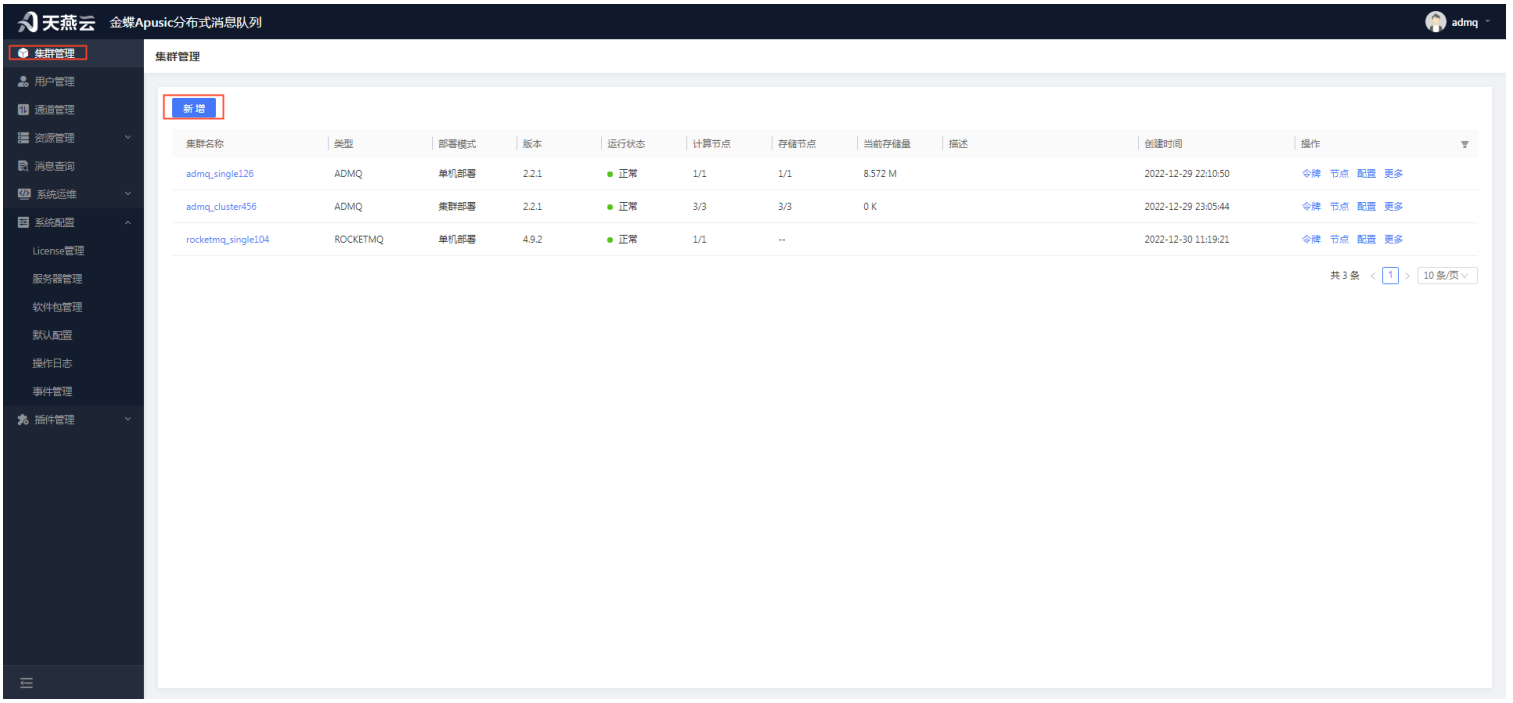
集群服务器节点启动前需要license进行授权，若在集群启动的节点数量超过License允许的最大计算节点数量和最大存储节点数量，则启动集群节点起不来，需要再上传license（上传license后，最大存储节点数与最大计算节点数会进行累加）。若要部署单机环境时，最大节点数和最大存储节点数只需要大于等于即可，若需要1个部署环境，最大节点数和最大存储节点数需要大于等于3。（具体数量需要根据节点数量来确认）

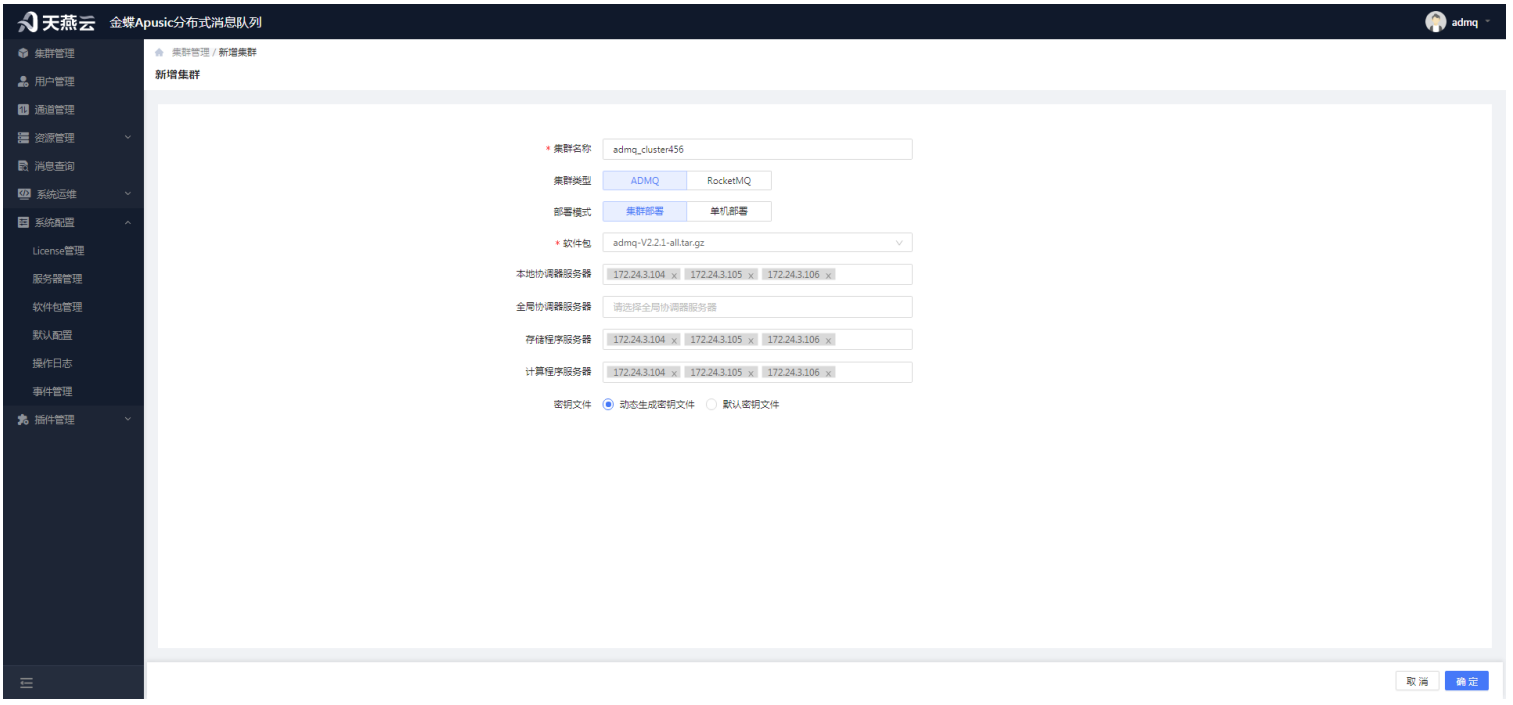


6 六、集群管理

6.0.1 6.1 新增集群

进入集群管理->新增，输入集群名称，选择部署模式（若只增加一台服务器信息，则选择单机模式，若增加3台以上的服务器则可选择集群部署，可部署ADMQ集群或者RocketMQ集群）；下拉选择5.1（上传软件包）中新增的软件包信息，下拉选择选择本地协调器服务器、存储程序服务器、计算程序服务器（5.2新增服务器信息），秘钥文件默认动态生成秘钥文件，点击【确定】按钮。





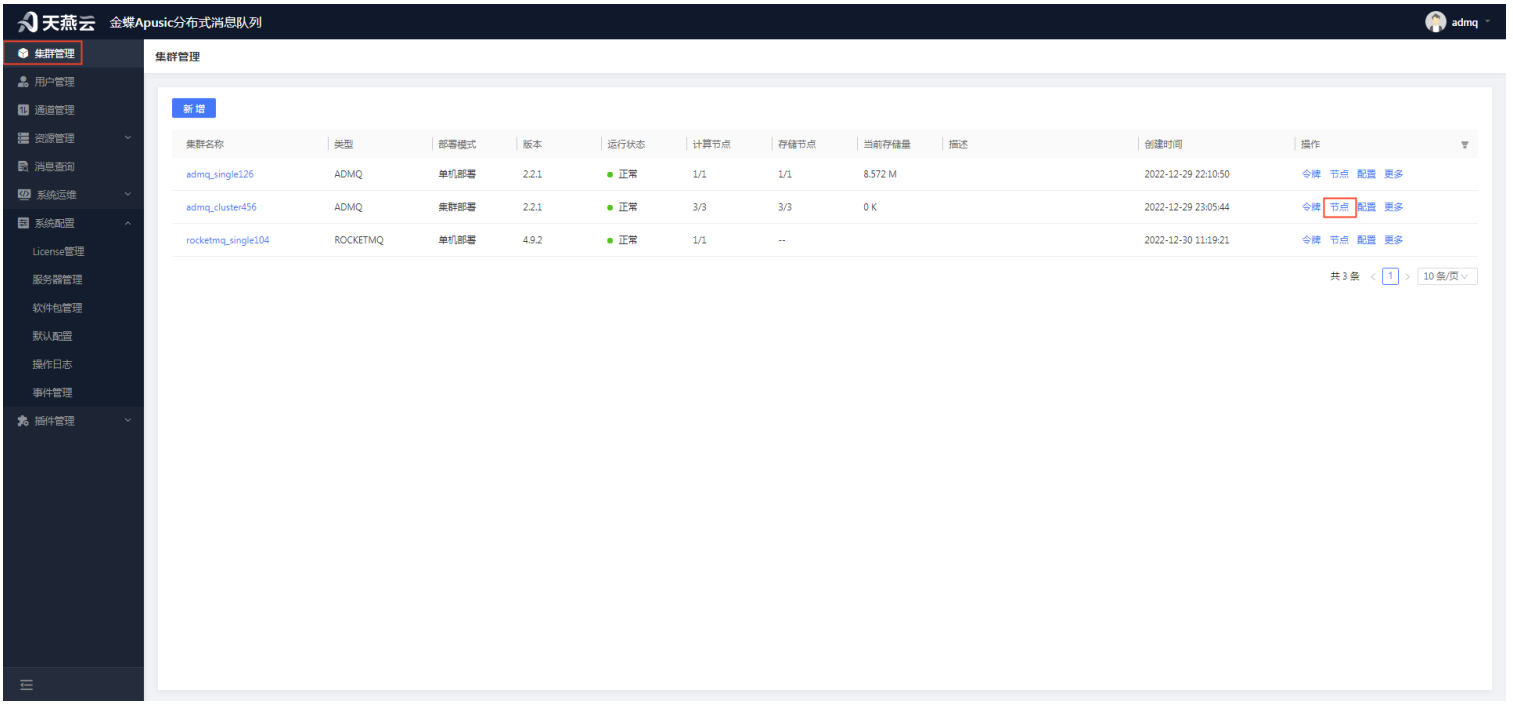
6.0.2 6.2 部署集群

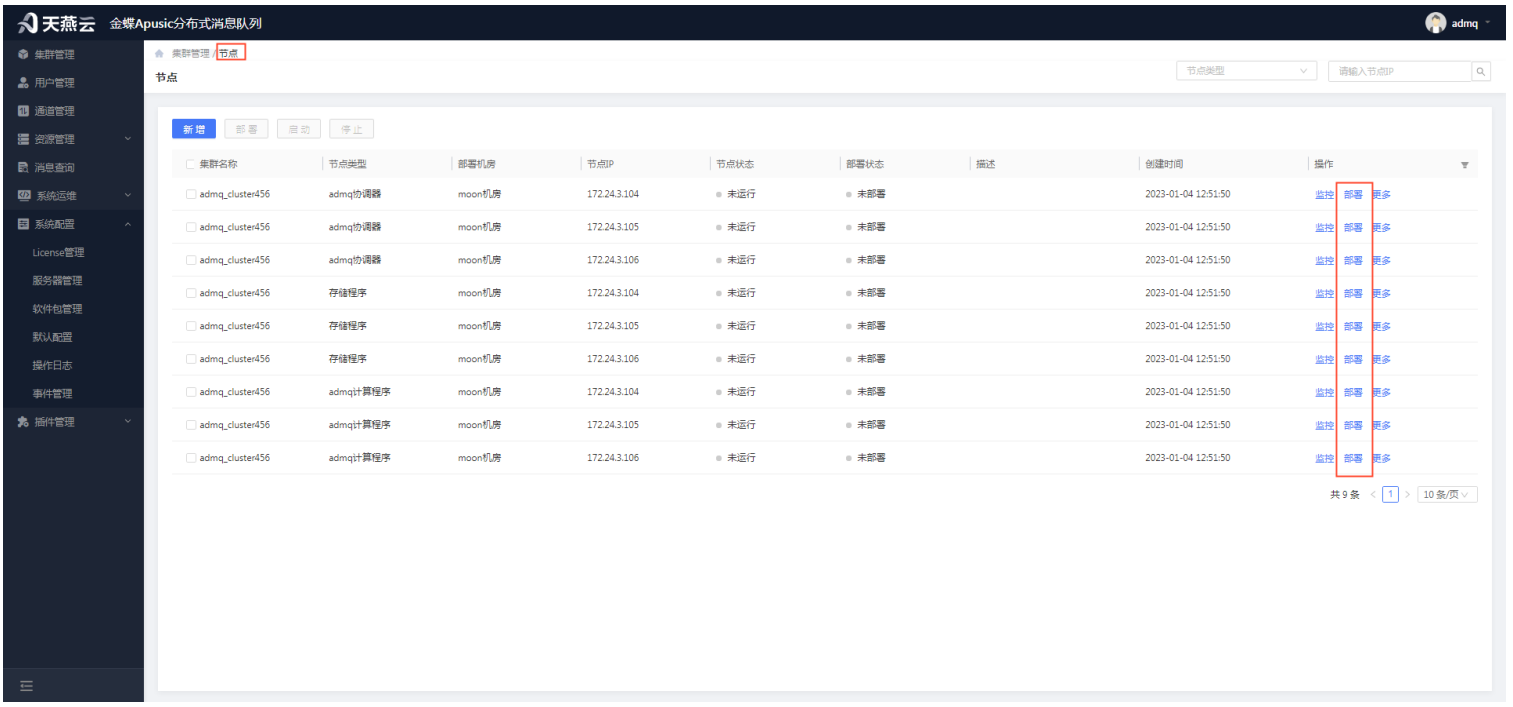
部署前检查：

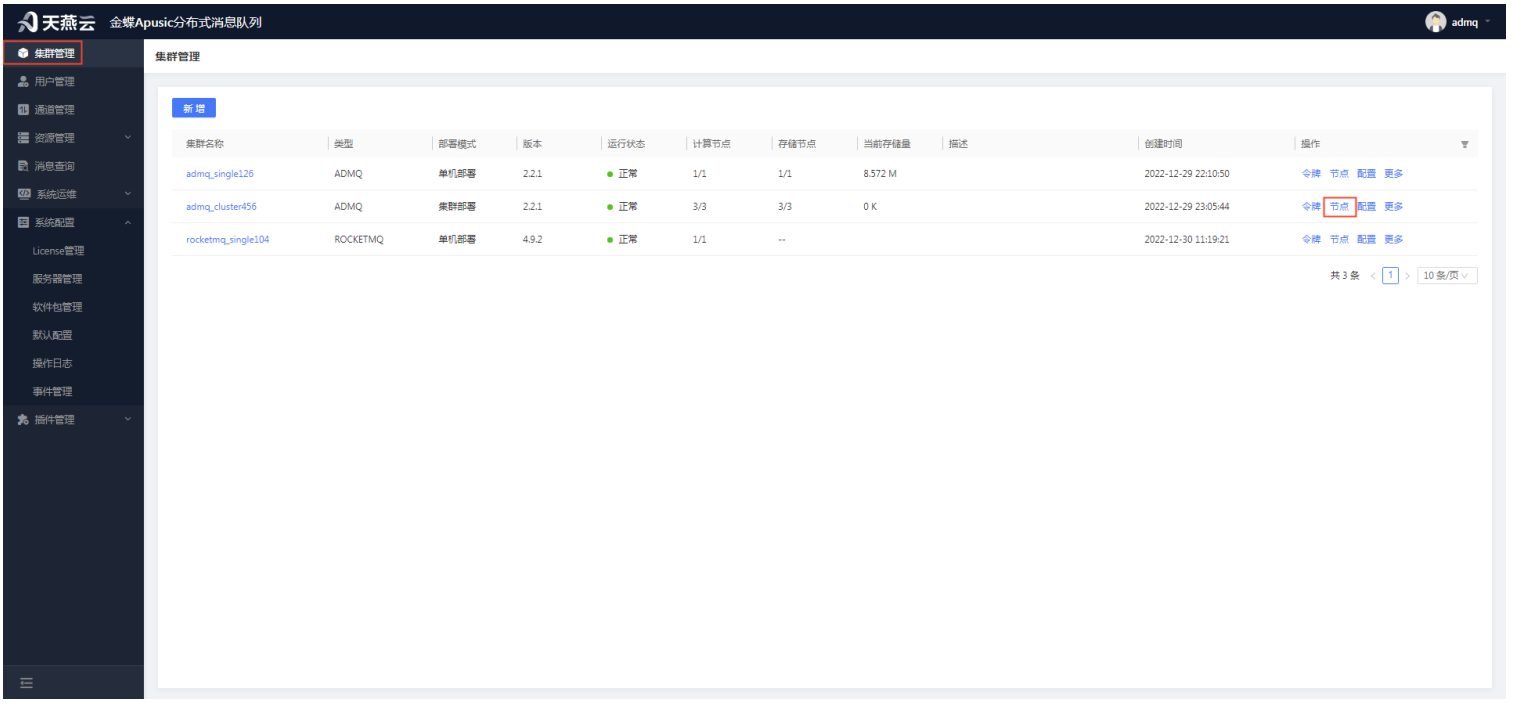
集群部署会使用默认配置文件中一些默认端口，部署前需要开放某些端口,管控台不负责处理,由用户自己处理: 存储程序:2181,2888,3888 全局协调器：2184,2889，3889 :3181,8000, br:6650,8080,如果使用tls，需要开放6651,8081。

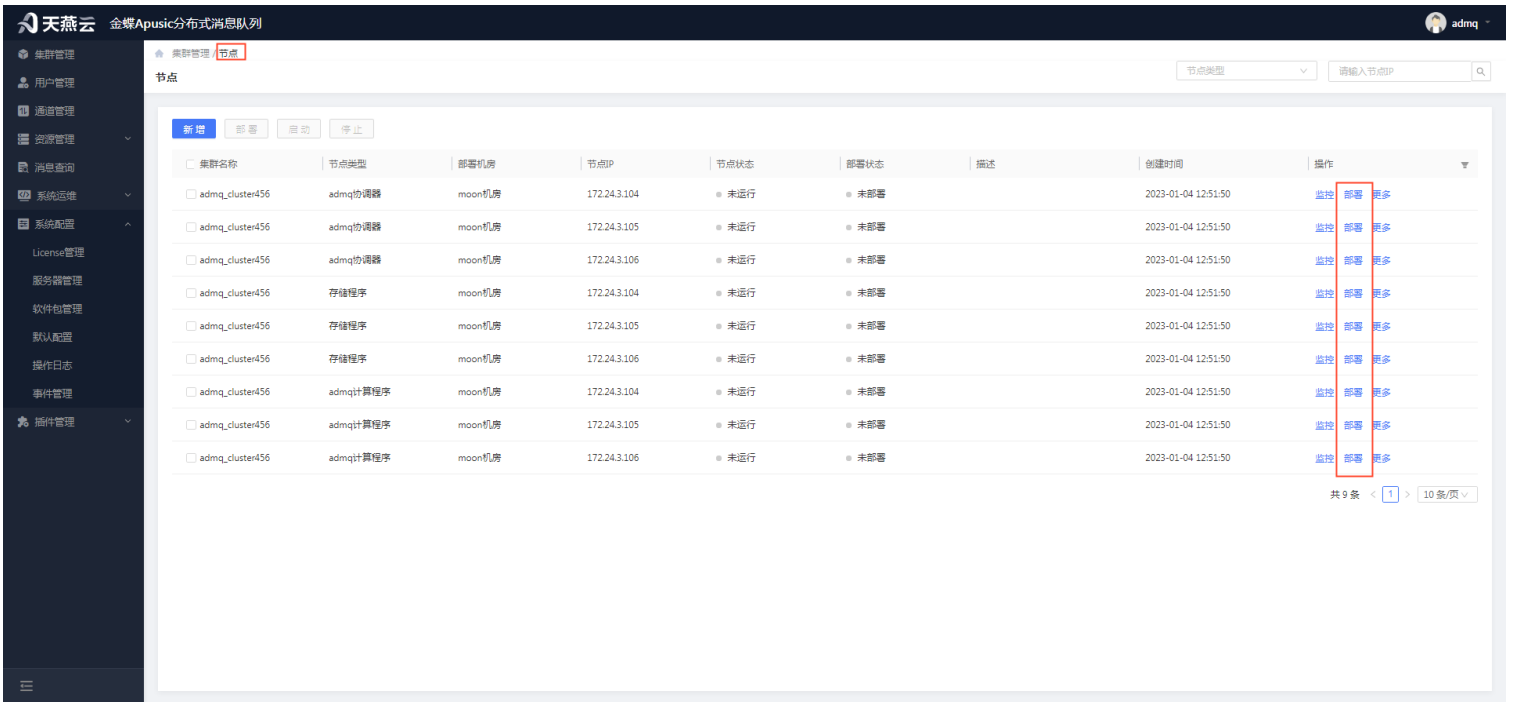
先查看防火墙是否开启：systemctl status firewalld 如果没开启，要先开启：systemctl start firewalld 查看端口是否对外开放：firewall-cmd --query-port=端口号/tcp 如果没开放，就执行： firewall-cmd --add-port=端口号/tcp --permanent 重新加载防火墙的端口：firewall-cmd --reload

新增集群信息后，可部署集群服务器节点。




天燕云
金蝶Apusic分布式消息队列
admq
集群管理
用户管理
通道管理
资源管理
消息查询
系统运维
系统配置
License管理
服务器管理
软件包管理
默认配置
操作日志
事件管理
插件管理
集群管理 / 节点
节点
节点类型
请输入节点IP
新增
部署
自动
停止
集群名称 节点类型 部署机房 节点IP 节点状态 部署状态 描述 创建时间 操作
admq_cluster456 admq协调器 moon机房 172.24.3.104 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 admq协调器 moon机房 172.24.3.105 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 admq协调器 moon机房 172.24.3.106 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 存储程序 moon机房 172.24.3.104 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 存储程序 moon机房 172.24.3.105 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 存储程序 moon机房 172.24.3.106 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 admqi计算程序 moon机房 172.24.3.104 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 admqi计算程序 moon机房 172.24.3.105 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
admq_cluster456 admqit计算程序 moon机房 172.24.3.106 未运行 未部署 2023-01-04 12:51:50 监控 部署 更多
共 9 条
10 条/页

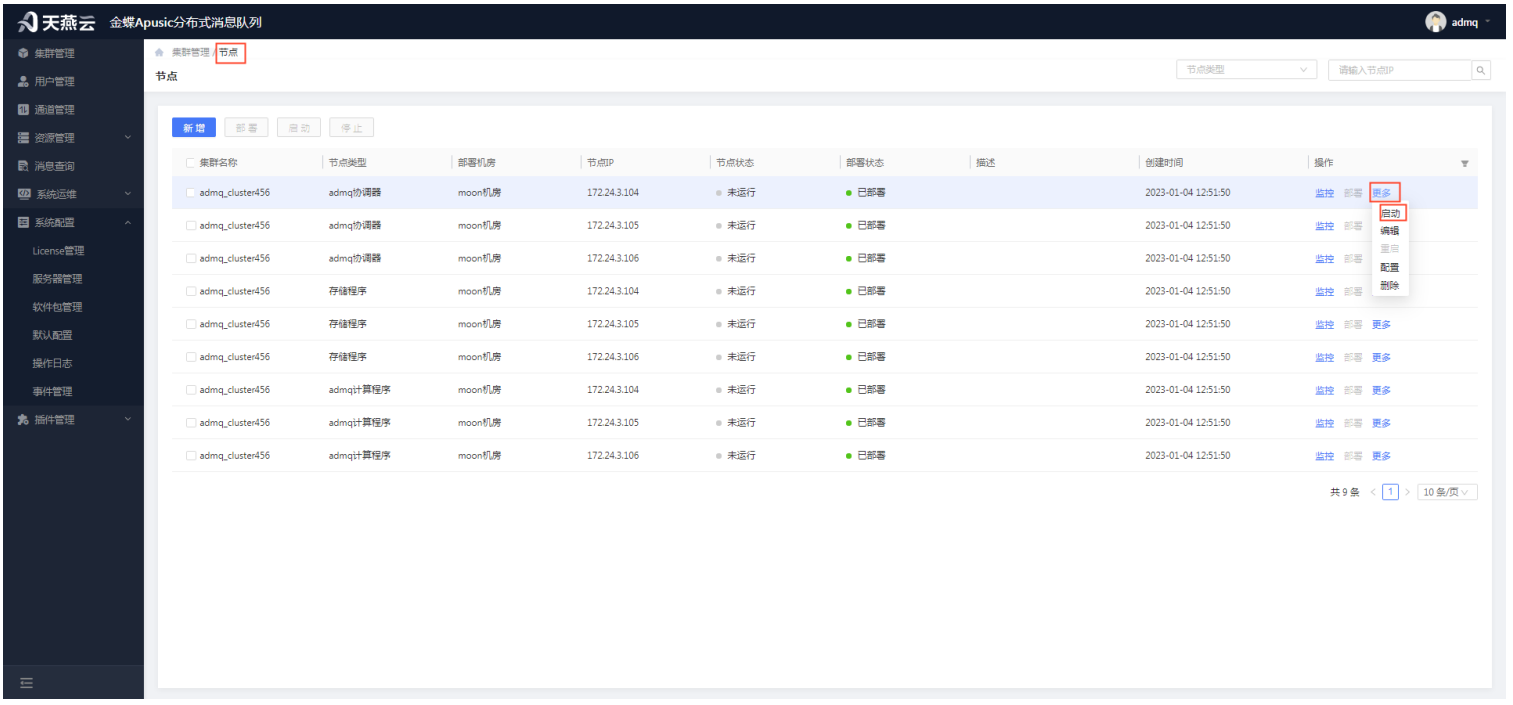

天燕云
金蝶Apusic分布式消息队列
admq
集群管理
用户管理
通道管理
资源管理
消息查询
系统运维
系统配置
License管理
服务器管理
软件包管理
默认配置
操作日志
事件管理
插件管理
集群管理
新增
集群名称 类型 部署模式 版本 运行状态 计算节点 存储节点 当前存储量 描述 创建时间 操作
admq_single126 ADMQ 单机部署 2.2.1 正常 1/1 1/1 8.572 M 2022-12-29 22:10:50 令牌 节点 配置 更多
admq_cluster456 ADMQ 集群部署 2.2.1 正常 3/3 3/3 0 K 2022-12-29 23:05:44 令牌 节点 配置 更多
rocketmq_single104 ROCKETMQ 单机部署 4.9.2 正常 1/1 -- 2022-12-30 11:19:21 令牌 节点 配置 更多
共 3 条
10 条/页



部署过程需要稍等几分钟左右。

6.0.3 6.3 启动集群节点

部署节点后，可以启动集群的各个节点。



启动所有节点，集群环境就部署成功了。

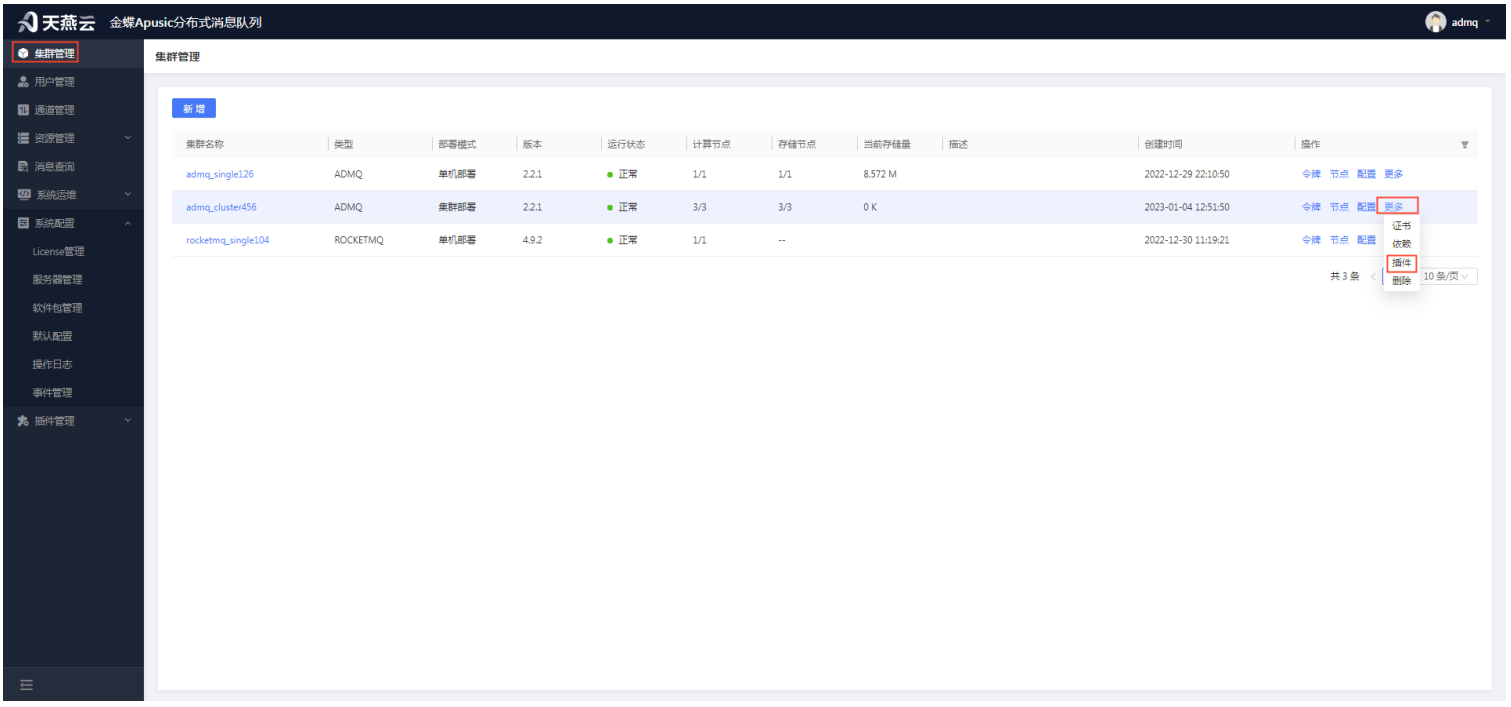
7 七、插件安装

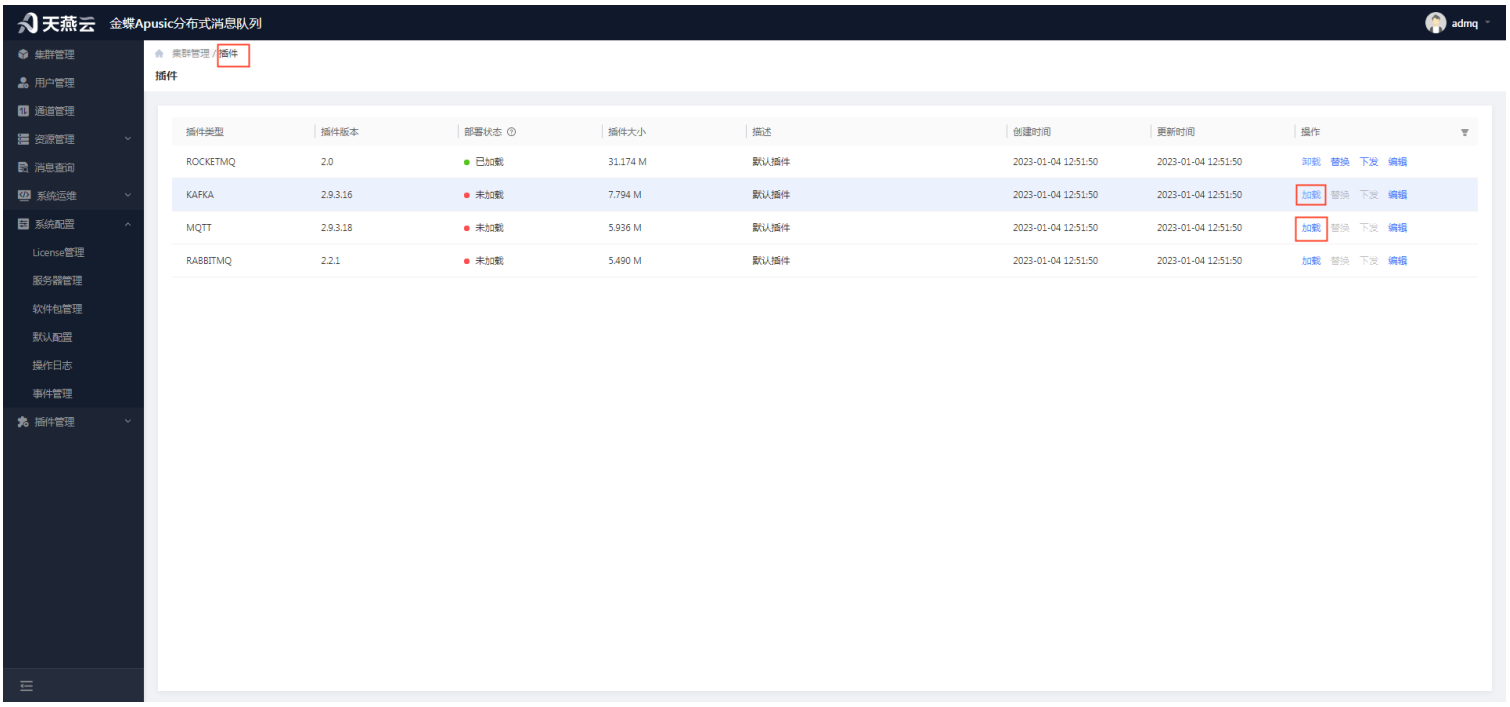
ADMQ支持通过加载Kafka、RocketMQ、RabbitMQ等插件的方式适配其他客户端协议，使原先使用其他消息中间件的系统可在不修改代码的情况下快速迁移到ADMQ上。

7.1 7.1 插件加载方式安装（推荐使用）

通过管控台界面加载KOP、MOP、ROP、AOP插件，进行插件管理，包括：插件加载、卸载、替换、下发，简化插件启动、停用流程，方便用户操作。

具体步骤：集群管理->选择要加载插件的集群选择“更多”按钮->插件，进入到插件管理页面后，可选择相对应的插件进行加载。注意：可同时加载部分或全部插件，加载后需要重启集群。





7.2 7.2 配置文件方式安装（不推荐使用）

需要提前确认计算节点的admq-V2.2.1/protocols/目录（插件协议路径）下是否有适配各个插件的文件，若没有需获取插件文件放置到该目录下。

插件对应关系如下：

Kafka插件：plugin_kafka-{version}.nar

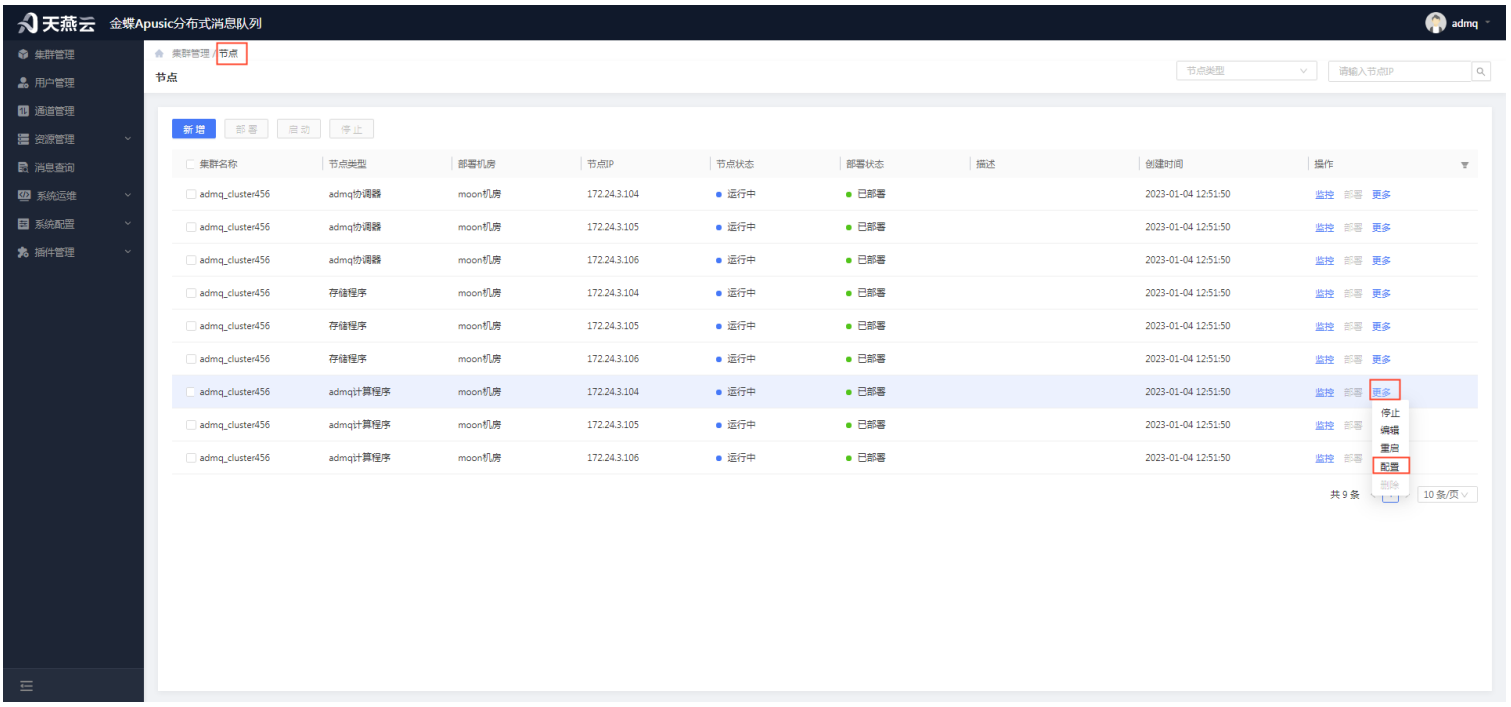
RocketMQ插件：plugin_rocketmq-{version}.nar

RabbitMQ插件：plugin_amqp-{version}.nar

MQTT插件：plugin_mqtt-{version}.nar

7.2.1 7.2.1 Kafka 插件

在部署完节点后，可在管控台集群节点页面修改集群每个计算节点的配置项，其他协调器和存储程序不需要修改。

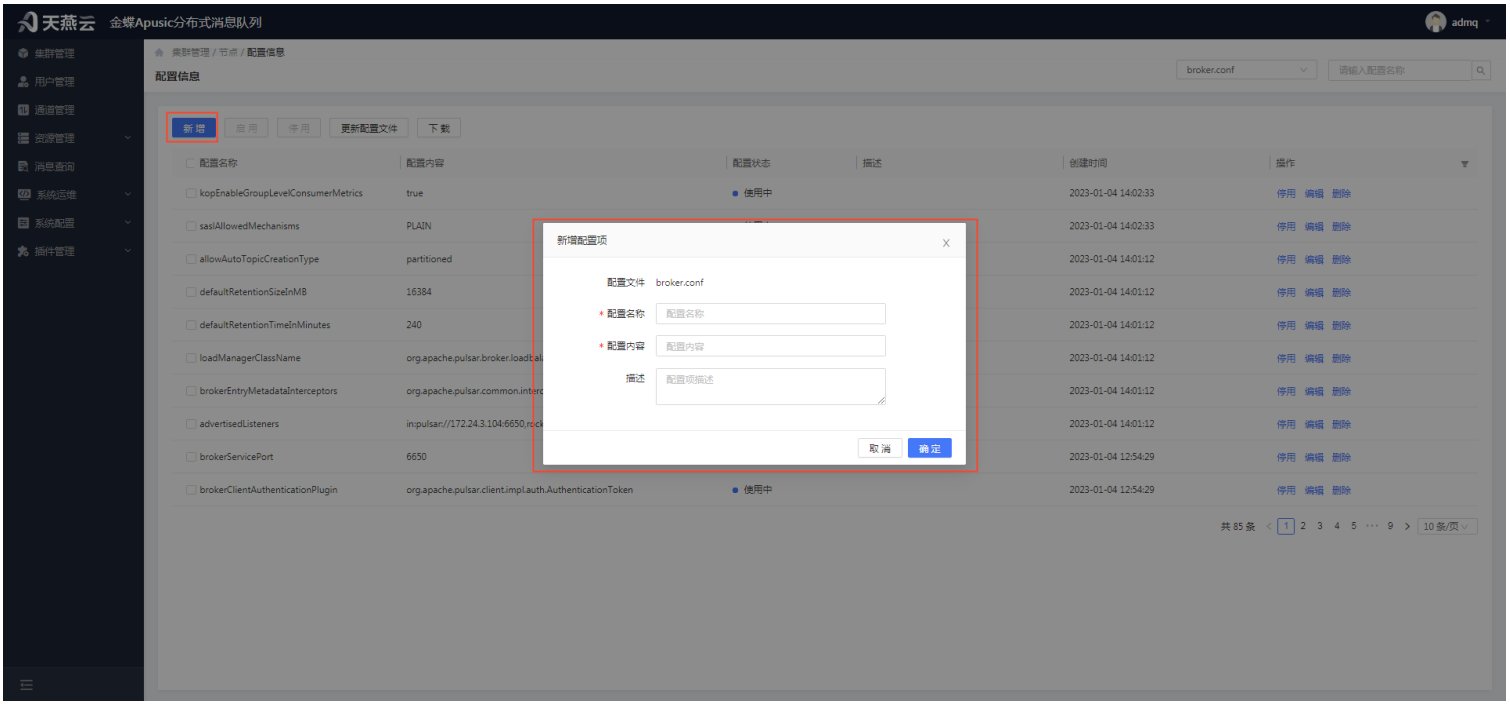


点"配置"按钮进入配置信息页面后，查找如下配置项，如果有则更新，没有则添加。

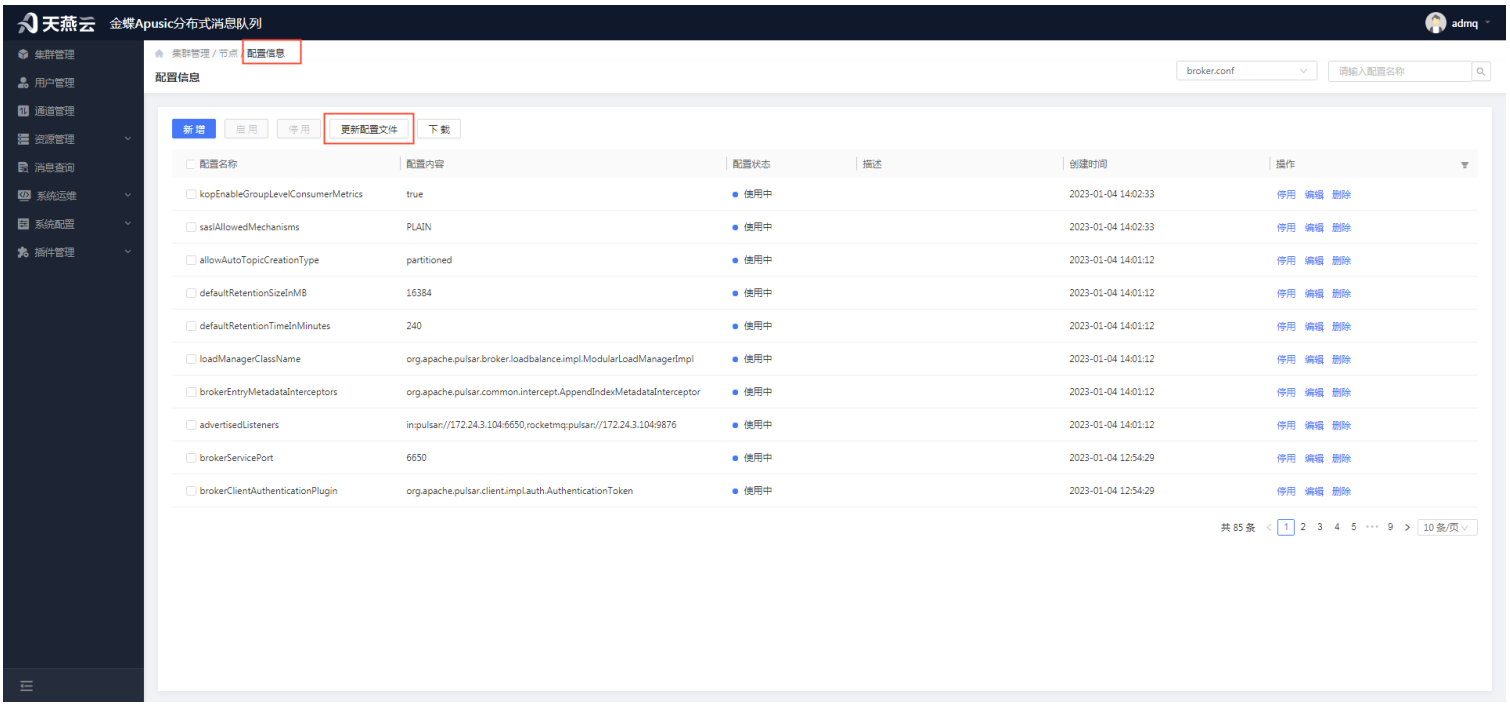
```
messagingProtocols=kafka
# 插件协议路径（{程序目录}/{集群名称}/{节点类型}/admq-V2.1.0/protocols）
# 程序目录：添加服务器时设置的程序目录
# 集群名称：添加集群时设置的集群名称
# 节点类型：如果是单机部署，则是standalone；如果是集群部署则是broker
protocolHandlerDirectory=/home/admq/pro/apusic-mq/broker/admq-V2.2.1/protocols
narExtractionDirectory=/home/admq/pro/apusic-mq/broker/admq-V2.2.1/nar

kafkaTenant=kafka-data
kafkaMetadataTenant=kafka-meta
# 改为服务器IP
kafkaListeners=kafka://172.20.140.140:9092
kafkaProtocolMap=kafka:PLAINTEXT
# 改为服务器IP
kafkaAdvertisedListeners=kafka://172.20.140.140:9092
entryFormat=kafka
# 偏移管理
brokerEntryMetadataInterceptors=org.apache.pulsar.common.intercept.Appe
```

```
# 设置自动创建分区主题
allowAutoTopicCreationType=partitioned

kopEnableGroupLevelConsumerMetrics=true
saslAllowedMechanisms=PLAIN
loadManagerClassName=org.apache.pulsar.broker.loadbalance.impl.ModularI
defaultRetentionTimeInMinutes=240
defaultRetentionSizeInMB=16384

# 设置不自动删除非活跃主题(可不添加)
brokerDeleteInactiveTopicsEnabled=false
```

添加配置项:



添加配置项完成后需要更新配置文件到节点：



备注：更新配置文件到节点后，需要重启节点。

7.2.2 7.2.2 RocketMQ 插件

方式同Kafka，涉及的配置项如下：

```
messagingProtocols=rocketmq
# 插件协议路径（{程序目录}/{集群名称}/{节点类型}/admq-V2.1.0/protocols）
# 程序目录：添加服务器时设置的程序目录
# 集群名称：添加集群时设置的集群名称
# 节点类型：如果是单机部署，则是standalone；如果是集群部署则是broker
protocolHandlerDirectory=/home/admq/pro/apusic-mq/broker/admq-V2.2.1/protocols
narExtractionDirectory=/home/admq/pro/apusic-mq/broker/admq-V2.2.1/nar

# rocketmq
rocketmqTenant=rocketmq-data
rocketmqMetadataTenant=rocketmq-meta
rocketmqListeners=rocketmq://172.20.140.140:9876
rocketmqListenerPortMap=9876:rocketmq
```

```
loadManagerClassName=org.apache.pulsar.broker.loadbalance.impl.ModularL
brokerEntryMetadataInterceptors=org.apache.pulsar.common.intercept.Appe

# 设置消息保留时间，单位：分钟
defaultRetentionTimeInMinutes=2400
# 设置消息保留大小，单位：MB
defaultRetentionSizeInMB=16384
# 设置自动创建分区主题
allowAutoTopicCreationType=partitioned
# 设置不自动删除非活跃主题（可不加）
brokerDeleteInactiveTopicsEnabled=false

advertisedListeners=INTERNAL:pulsar://172.20.140.140:6650,rocketmq:puls
ropBrokerReplicationNum=1
ropTraceTopicEnable=false
ropRestServerPort=9888
ropAclEnable=false
```

7.2.3 7.2.3 RabbitMQ 插件

方式同Kafka，涉及的配置项如下：

```
messagingProtocols=amqp
# 插件协议路径（{程序目录}/{集群名称}/{节点类型}/admq-V2.1.0/protocols）
# 程序目录：添加服务器时设置的程序目录
# 集群名称：添加集群时设置的集群名称
# 节点类型：如果是单机部署，则是standalone；如果是集群部署则是broker
protocolHandlerDirectory=/home/admq/pro/apusic-mq/broker/admq-
V2.2.1/protocols
narExtractionDirectory=/home/admq/pro/apusic-mq/broker/admq-
V2.2.1/nar
amqpTenant=amqp-data
amqpMetadataTenant=amqp-meta
amqpListeners=amqp://172.20.140.140:5672
amqpMaxNoOfChannels=2047
```

```
amqpMaxFrameSize=4194304
amqpProxyEnable=true
amqpProxyPort=6672
```

7.2.4 7.2.4 MQTT 插件

方式同Kafka，涉及的配置项如下：

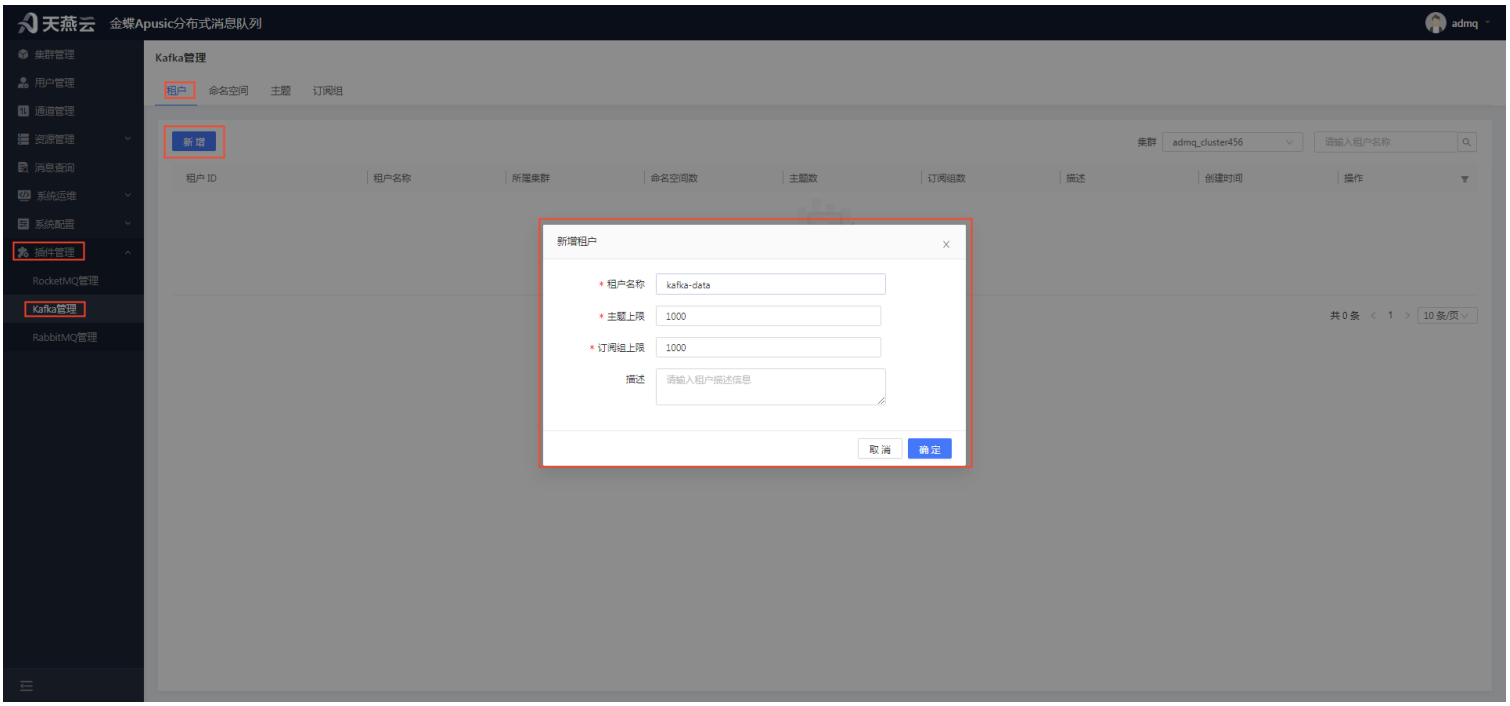
```
messagingProtocols=mqtt
# 插件协议路径（{程序目录}/{集群名称}/{节点类型}/admq-V2.1.0/protocols）
# 程序目录：添加服务器时设置的程序目录
# 集群名称：添加集群时设置的集群名称
# 节点类型：如果是单机部署，则是standalone；如果是集群部署则是broker
protocolHandlerDirectory=/home/admq/pro/apusic-mq/broker/admq-
V2.2.1/protocols
narExtractionDirectory=/home/admq/pro/apusic-mq/broker/admq-
V2.2.1/nar
mqttListeners=mqtt://172.20.140.140:1883
mqttProxyEnabled=true
mqttProxyPort=5682
```

7.3 7.3 插件使用

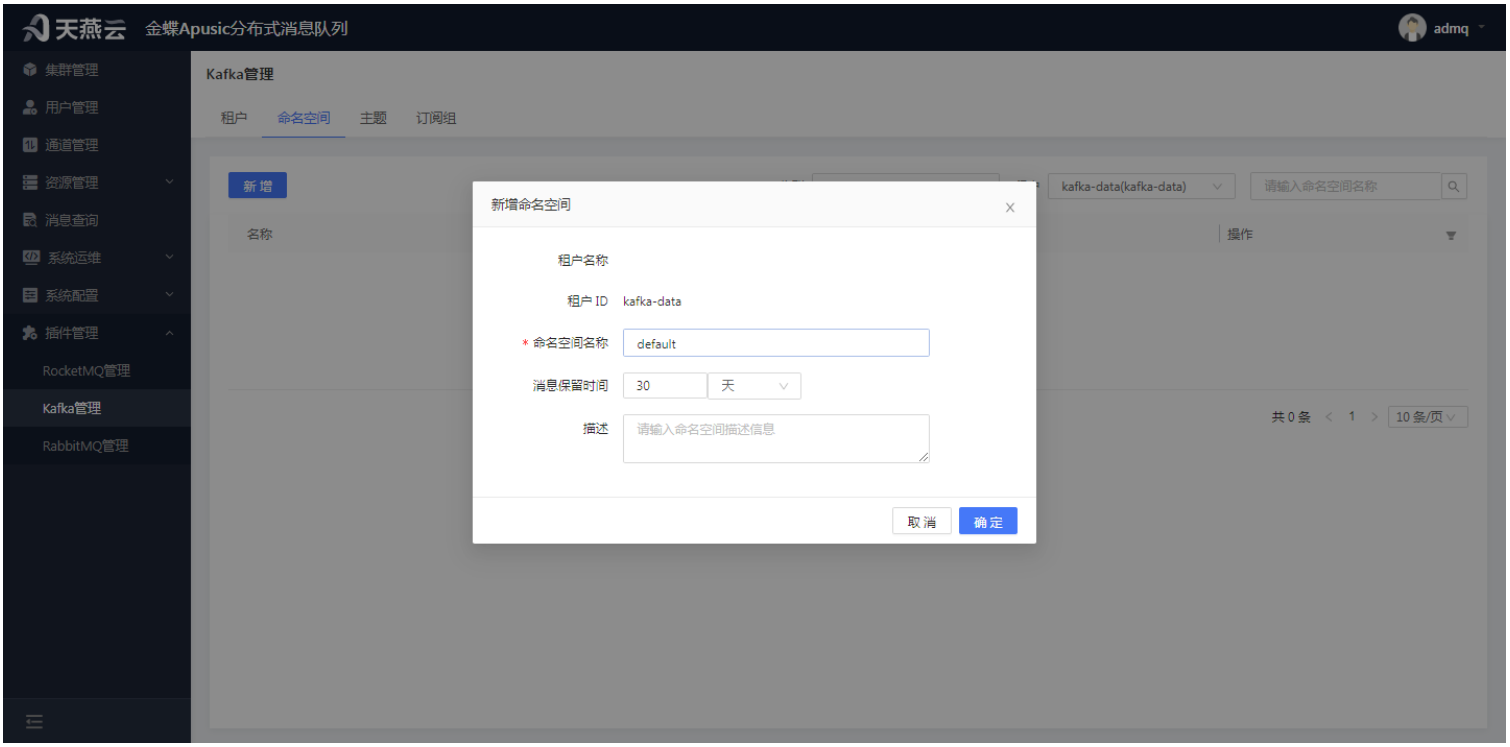
7.3.1 7.3.1 Kafka 插件

在插件管理 - Kafka管理 - 租户页面创建租户：kafka-data，在该租户下创建命名空间：default。

租户创建：



命名空间创建：



备注：客户端使用的租户名是租户ID，租户ID是以kafka-开头，创建租户后租户ID会自动生成且以 “kafka-” 开头。

之后可以使用kafka客户端连接到 IP1:9092;IP2:9092;IP3:9092服务地址（单机服务地址：IP:9092）进行发送和接收消息，demo如下：

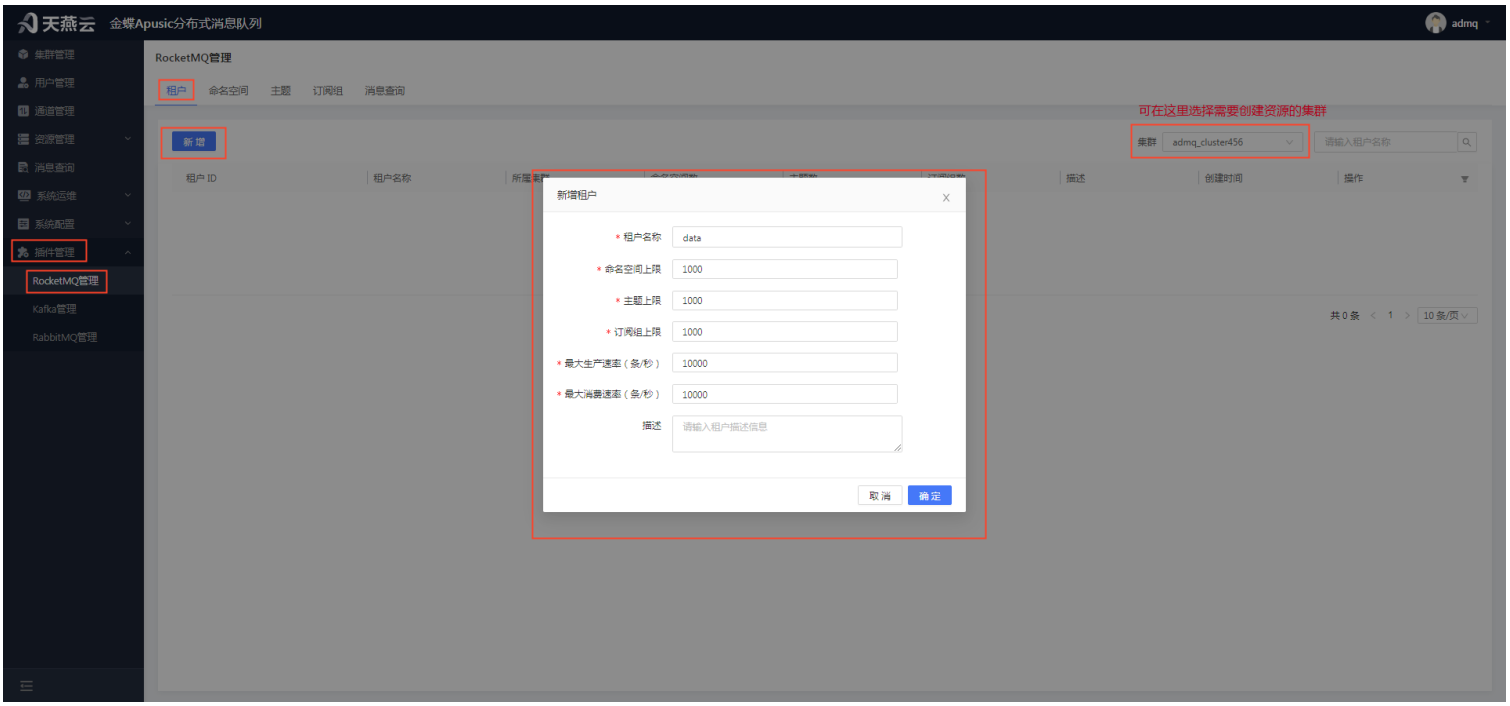
```
public void sendMessage() {
    long time = System.currentTimeMillis();
    int limit = speed < 1 ? Integer.MAX_VALUE : speed;
    LongAdder curCount = new LongAdder();

    Properties props = new Properties();
    props.put("bootstrap.servers", service);
    props.put("acks", "all");
    props.put("retries", 0);
    props.put("batch.size", 16384);
    props.put("key.serializer", StringSerializer.class.getName());
    props.put("value.serializer", StringSerializer.class.getName());

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

    int i = 0;
    while (repeat == -1 || i < repeat) {
        String messageValue = message + "_" + i;

        ProducerRecord<String, String> record = new ProducerRecord<>
(topic, messageValue);
        producer.send(record);

        log.info("send message: {}", messageValue);
        if (interval > 0) {
            Thread.sleep(interval);
        }
        time = speedLimit(curCount, limit, time);
        i ++;
    }

    producer.close();
}
```

```
public void receiveMessage() {
    List<String> topics = new ArrayList<>();
    topics.add(topic);

    Properties props = new Properties();
    props.put("bootstrap.servers", service);
    props.put("group.id", "admq");
    props.put("auto.offset.reset", "earliest");
    props.put("key.deserializer",
StringDeserializer.class.getName());
    props.put("value.deserializer",
StringDeserializer.class.getName());

    KafkaConsumer<String, String> consumer = new KafkaConsumer<>
(props);
    consumer.subscribe(topics);

    while (true) {
        ConsumerRecords<String, String> records =
consumer.poll(Duration.ofSeconds(120));
        if (records == null || records.isEmpty()) {
            break;
        }
        for (ConsumerRecord<String, String> record : records) {
            log.info("receive message: {}", record.value());
        }
    }

    consumer.close();
}
```

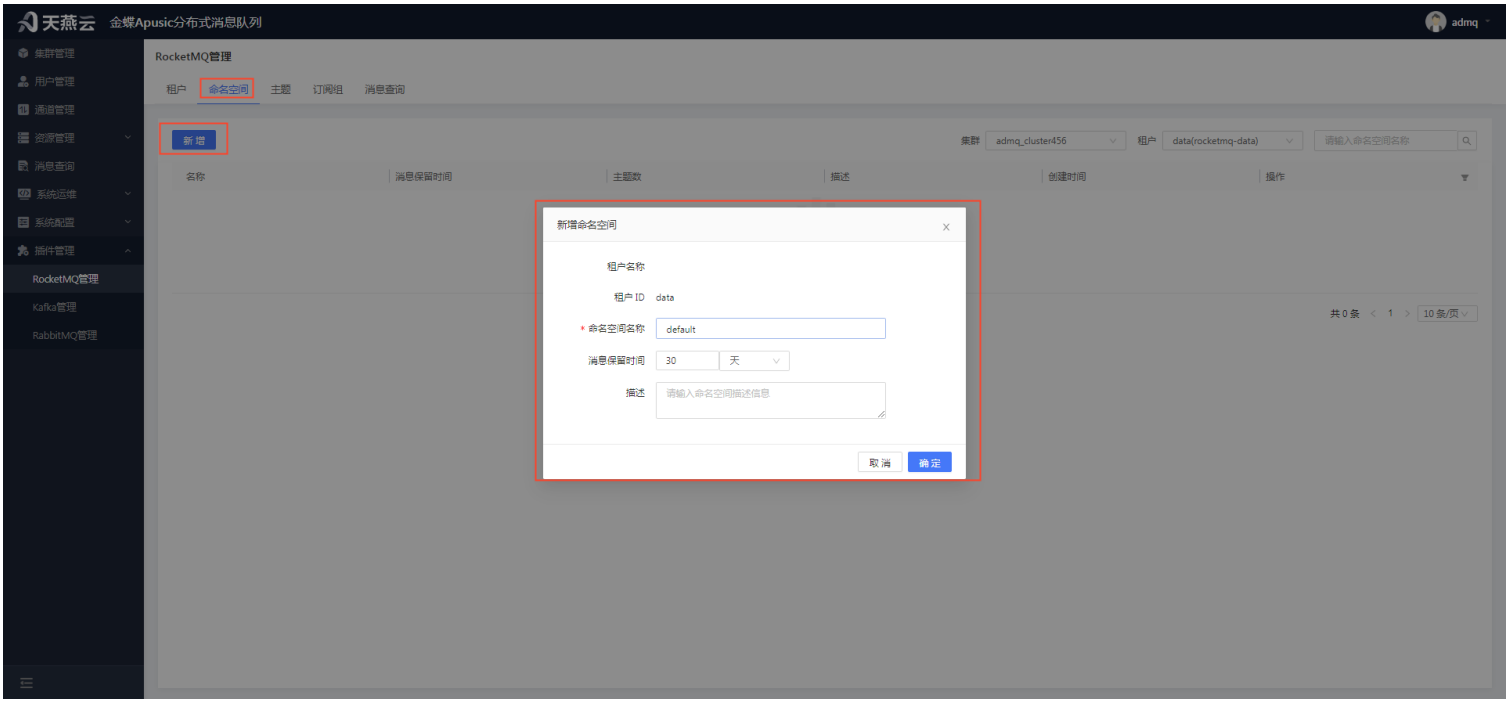
7.3.2 7.3.2 RocketMQ 插件

在插件管理 - RocketMQ管理 - 租户页面创建租户：data，在该租户下创建命名空间：default。

创建租户：



创建命名空间：



备注：客户端使用的租户名是租户ID，租户ID是以rocketmq-开头，创建租户后租户ID会自动生成且以 “rocketmq-” 开头。

之后可以使用RocketMQ客户端连接到 IP1:9876;IP2:9876;IP3:9876服务地址（单机服务地址：IP:9876）接收和发送消息（设置NamesrvAddr），示例如下：

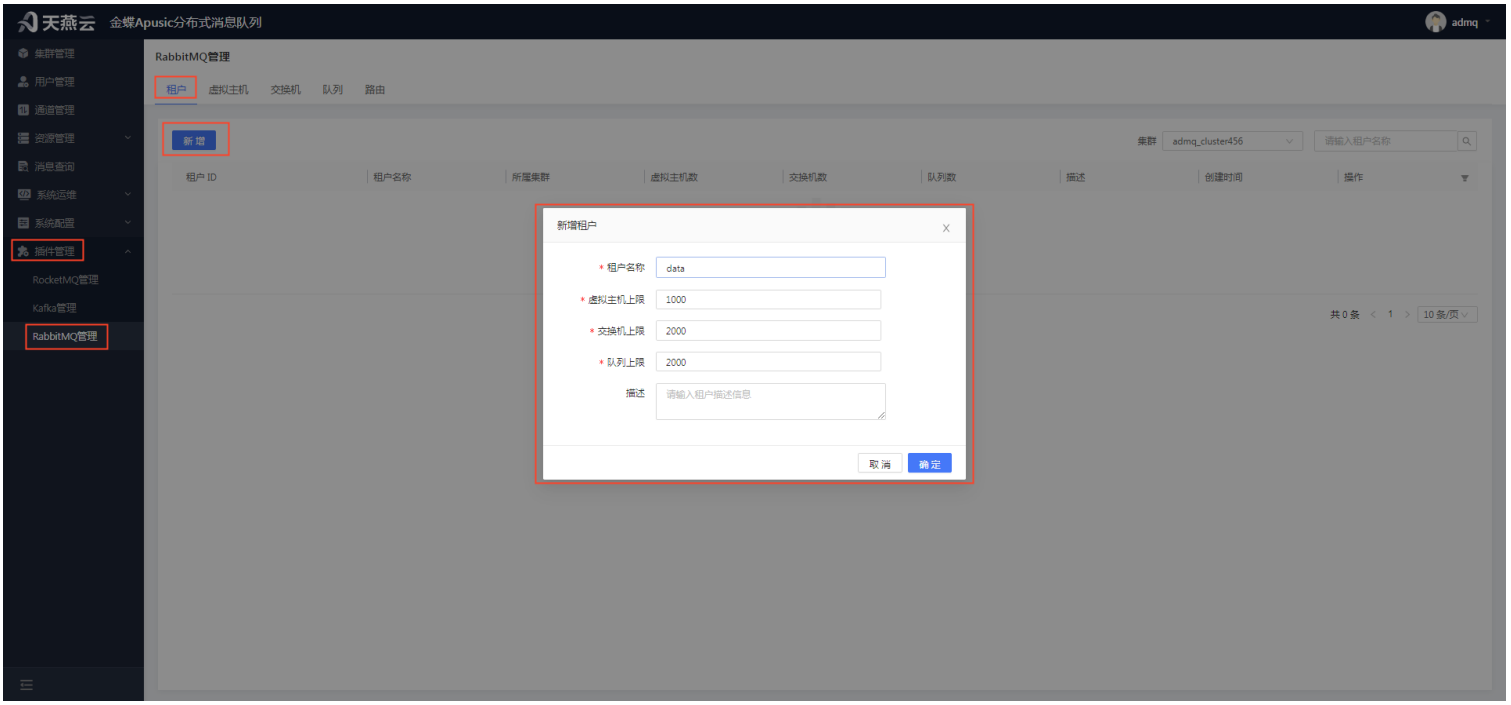
```
public void testSendAndReceive() throws Exception {
    int maxReConsumeTimes = 3;
    int consumeTimeout = 5;

    String groupName = "vv-group";
    String topic = "vv-topic";

    DefaultMQProducer producer = new DefaultMQProducer(groupName);
    producer.setNamesrvAddr(nameServer);
    producer.start();
    for (int i = 0; i < 2; i++) {
        Message msg = new Message(topic, "vv", ("hello world - " +
i).getBytes());
        SendResult sendResult = producer.send(msg);
        System.out.println(time() + " send result: " + sendResult);
    }

    subscribe(groupName, nameServer, maxReConsumeTimes,
consumeTimeout, topic);
    while (true);
}

void subscribe(String groupName, String nameServer, int
maxReConsumeTimes, int consumeTimeout, String topic) throws
Exception {
    DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer(groupName, false);
    consumer.setNamesrvAddr(nameServer);
    consumer.setMaxReconsumeTimes(maxReConsumeTimes);
    consumer.setConsumeTimeout(consumeTimeout);
    consumer.setPullBatchSize(1024);
    consumer.setPullInterval(100);

consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET
    consumer.subscribe(topic, "*");
```

```
    consumer.registerMessageListener((MessageListenerConcurrently)
(msgList, context) -> {

        msgList.forEach(action -> {
            System.out.println(time() + " recv topic: " + topic + "
- " + action.getTopic()
                               + ", msgId: " + action.getMsgId()
                               + ", data: " + new
String(action.getBody()));
        });

        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    });
    consumer.start();
}
```

7.3.3 7.3.3 RabbitMQ 插件

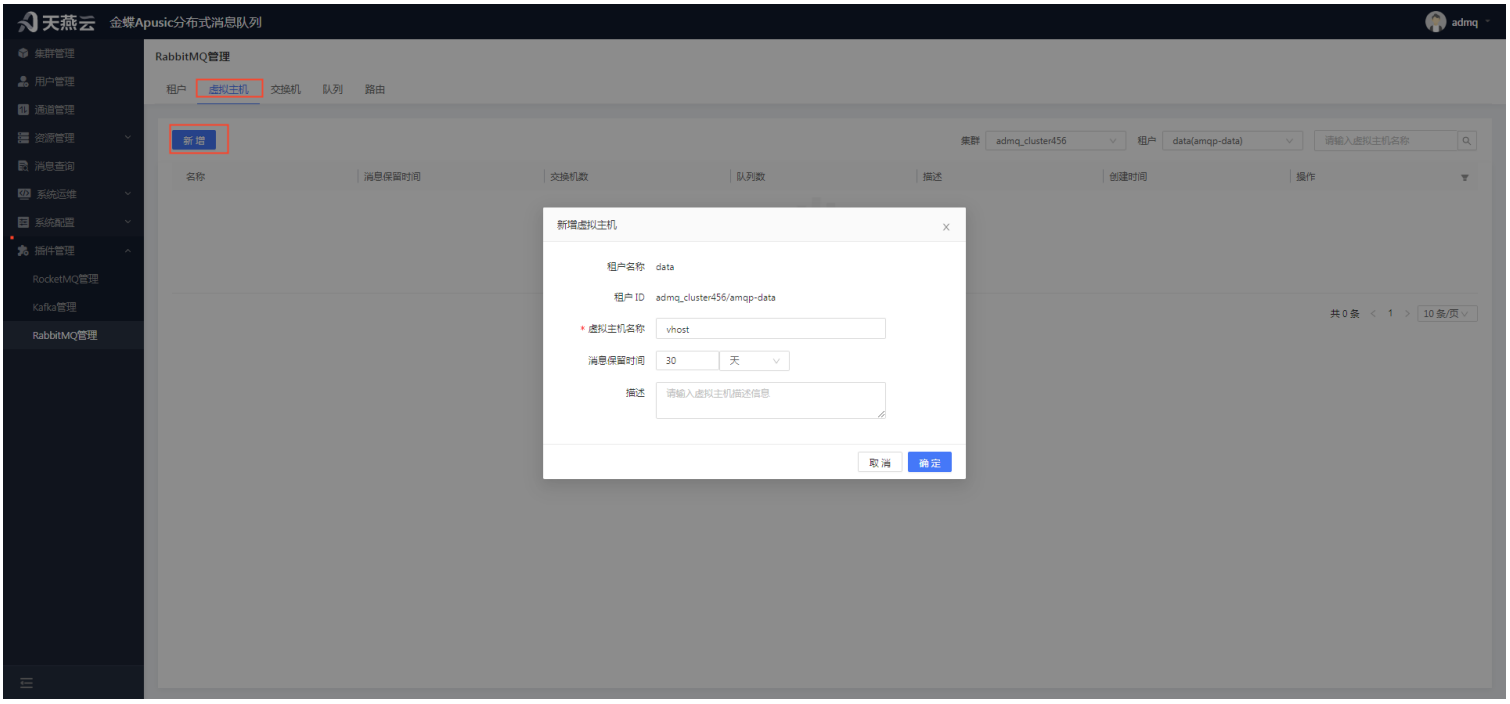
在插件管理 - RabbitMQ管理 - 租户页面创建租户：data，在该租户下创建虚拟主机、交换机、队列、路由资源。

创建租户：

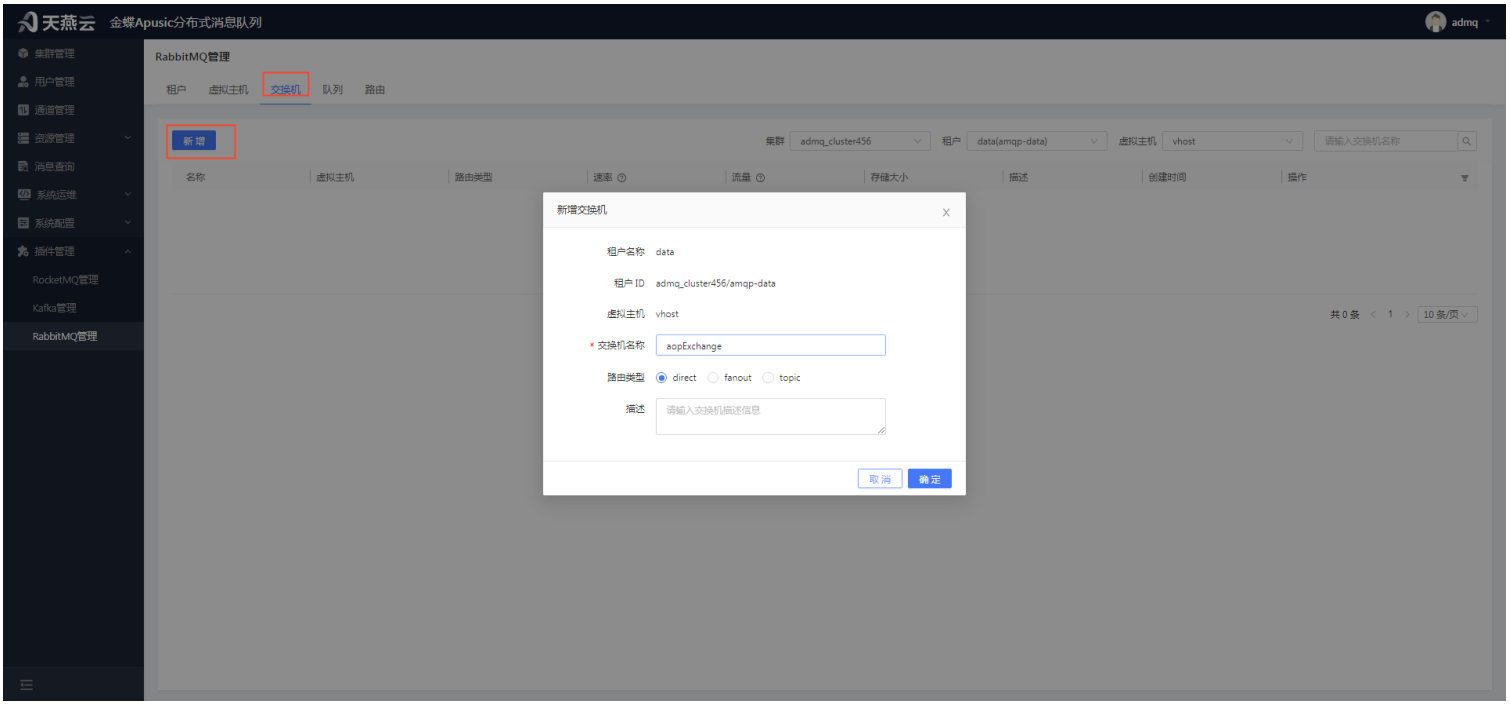


备注：RabbitMQ插件的租户ID需和配置文件中租户名称保持一致。

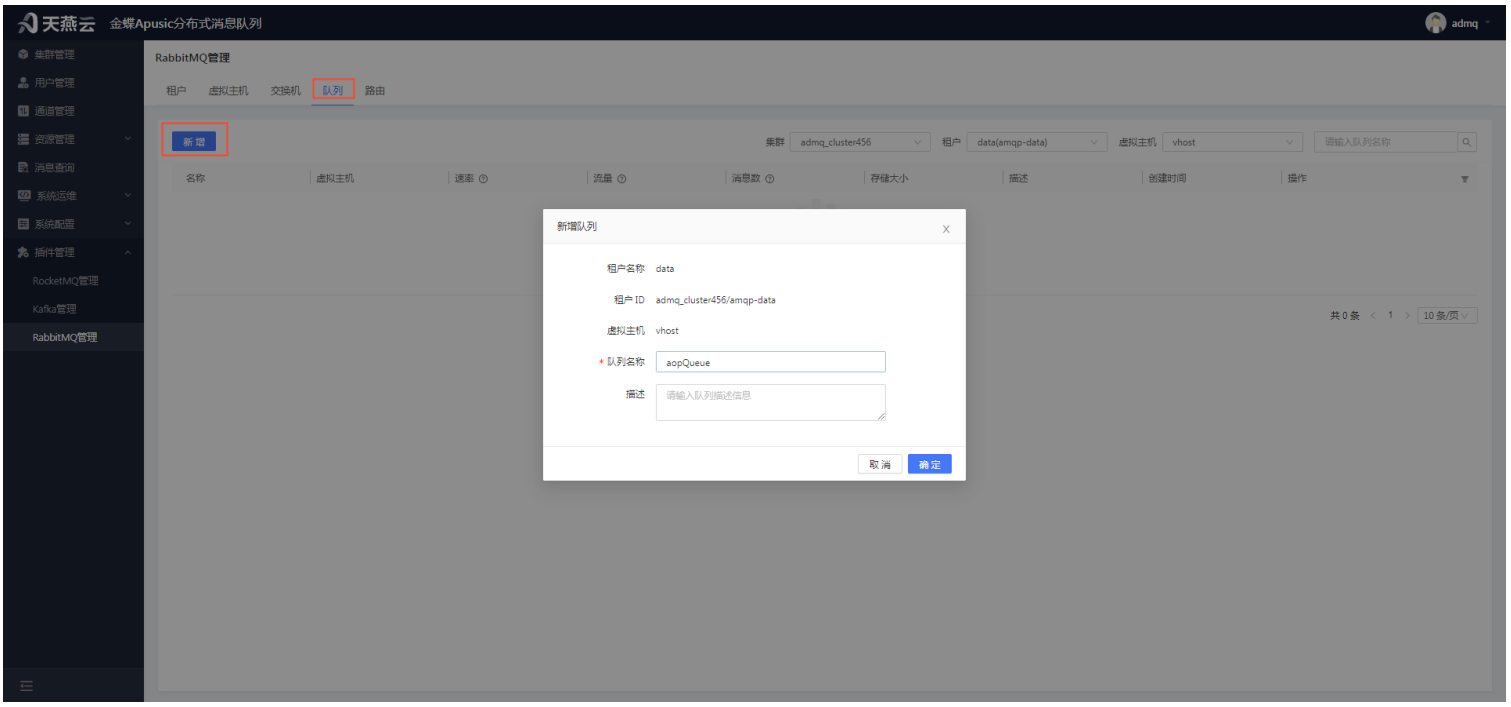
创建虚拟主机：



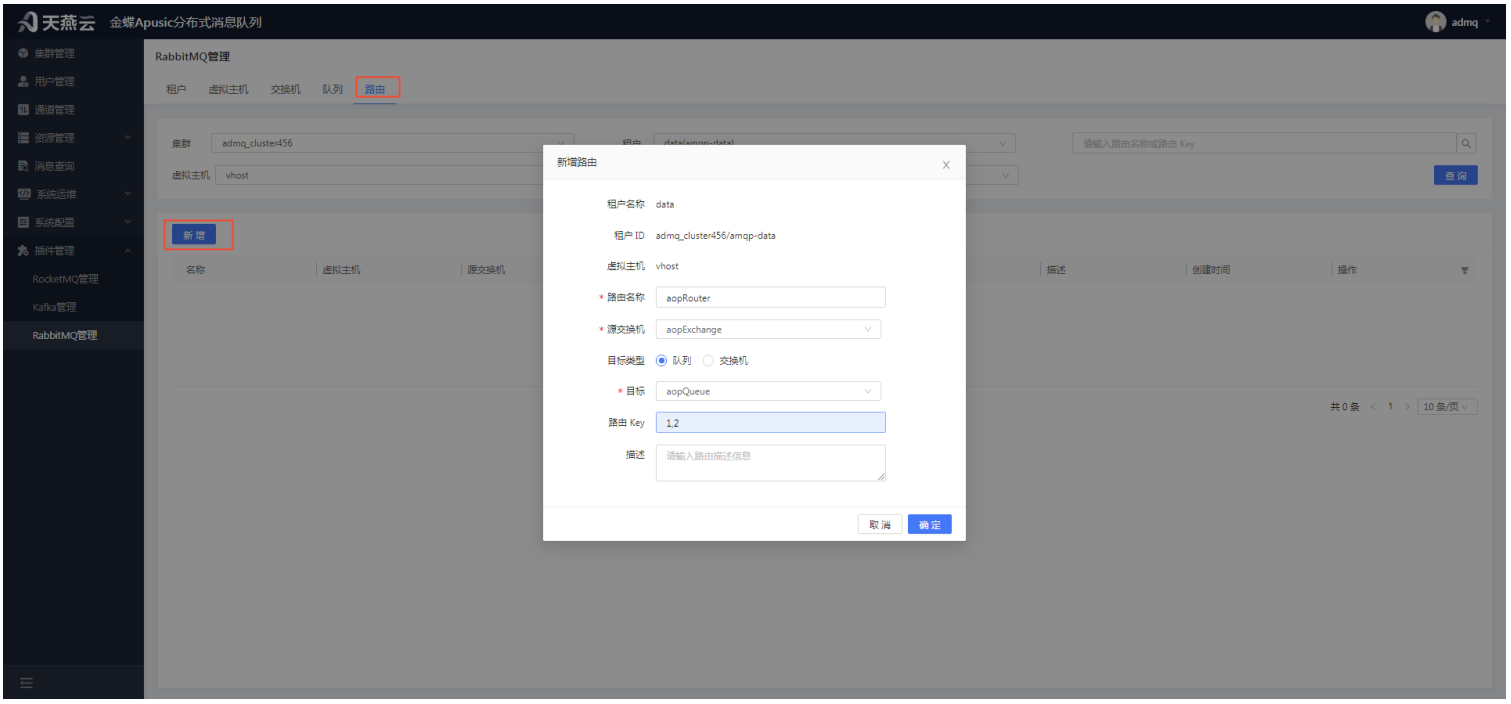
创建交换机：



创建队列：



创建路由：



在租户页创建租户：amqp-data；点击租户名称，创建命名空间：vhost1（和客户端使用的vhost名称保持一致）。

之后可以使用RabbitMQ客户端连接到服务地址接收和发送消息（虚拟主机名称需和客户端使用的vhost名称保持一致），示例如下：

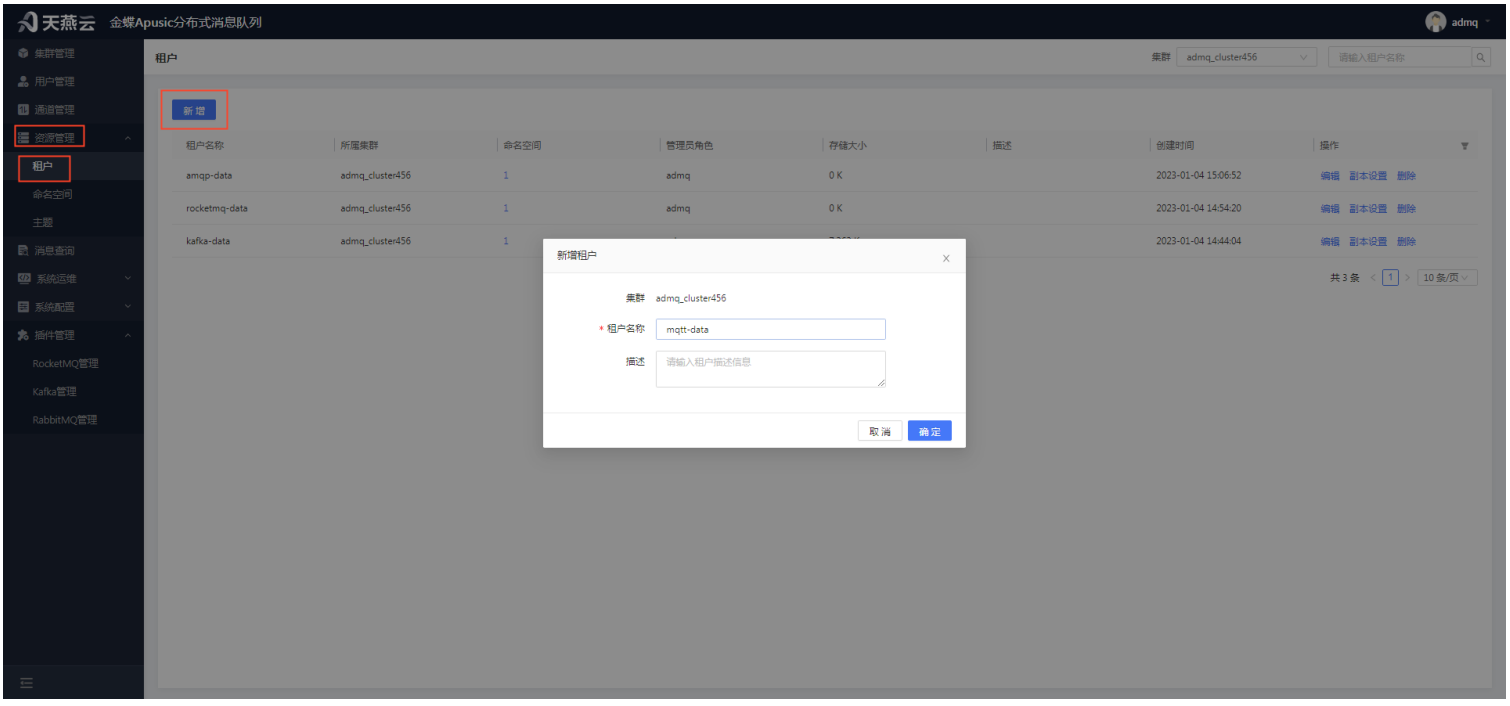
```
private String exchange = "aopExchange";
private String server = "172.20.140.140";
private int port = 5672;
private String vhost = "vhost";

private Connection connection;
private Channel channel;

// 发送消息
public void send(String message) throws Exception {
    for (int i = 0; i < 100; i++) {
        String messageValue = message + "_" + i;
        channel.basicPublish(exchange, "", null,
messageValue.getBytes());
        log.info("send message: {}", messageValue);
    }
}


// 接收消息
public void receive() throws Exception {
    channel.basicConsume(queue, true, new DefaultConsumer(channel) {
        @Override
        public void handleDelivery(String consumerTag, Envelope
envelope, AMQP.BasicProperties properties,
                                   byte[] body) throws IOException {
            log.info("receive message: {}", new String(body));
        }
    });

    while (true);
}


// 建立连接
```

```
public void createConnectionAndChannel() throws Exception {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setVirtualHost(vhost);
    connectionFactory.setHost(server);
    connectionFactory.setPort(port);
    connection = connectionFactory.newConnection();
    channel = connection.createChannel();

    // exchage declare
    channel.exchangeDeclare(exchange, BuiltinExchangeType.FANOUT,
true, false, false, null);

    // queue declare and bind
    channel.queueDeclare(queue, true, false, false, null);
    channel.queueBind(queue, exchange, "");

}
```

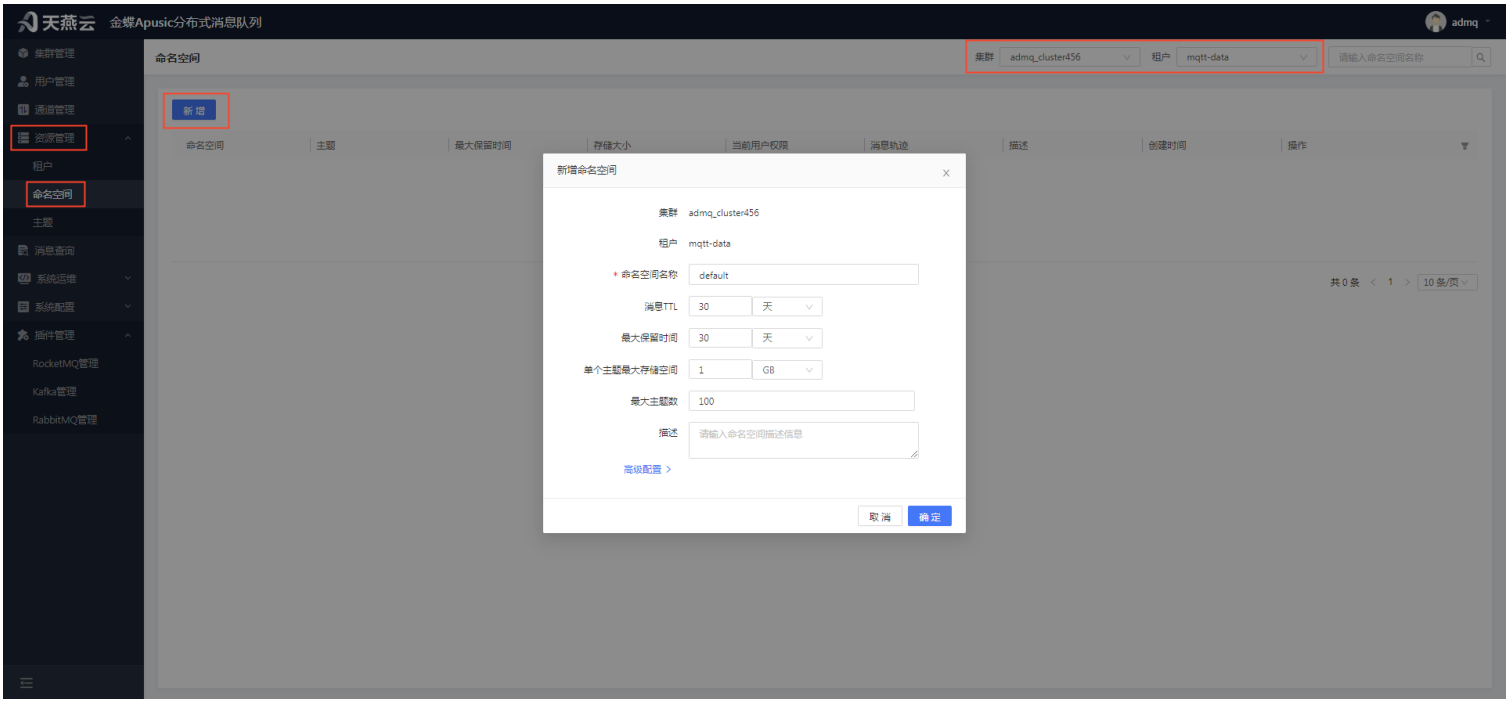
7.3.4 7.3.4 MQTT 插件

在资源管理 - 租户页面创建租户以"mqtt-"开头的租户如：mqtt-data，在该租户下命名空间default、主题mqtt-topic。

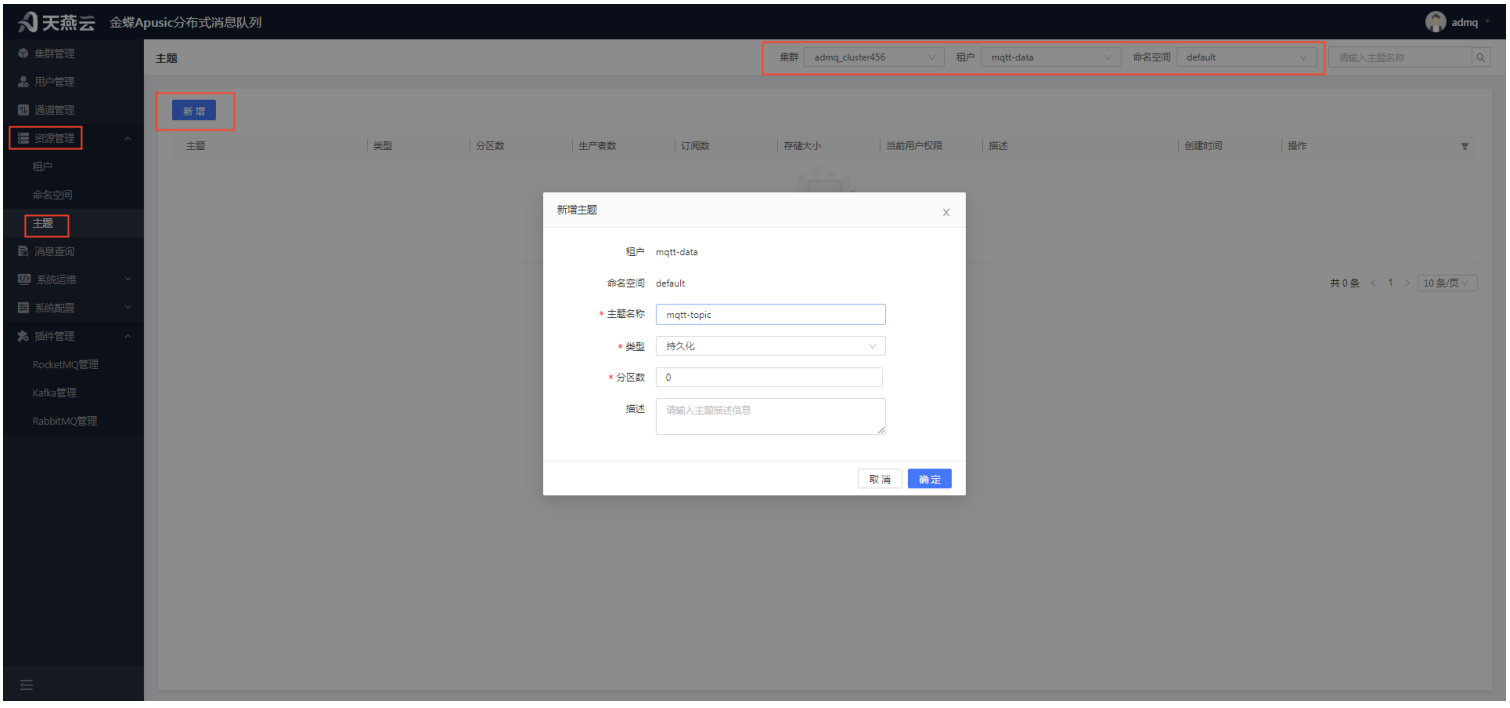
创建租户：



创建命名空间：



创建主题：



之后可以使用RabbitMQ客户端连接到服务地址接收和发送消息，示例如下：

```
private String server = "IP"
private int port = 1883;
```

```
private String topic = "mqtt-topic";

public void receiveMessage() throws Exception {
    MQTT mqtt = new MQTT();
    mqtt.setHost(server, port);
    BlockingConnection connection = mqtt.blockingConnection();
    connection.connect();
    Topic[] topics = { new Topic(topic, QoS.AT_LEAST_ONCE) };
    connection.subscribe(topics);

    while (true) {
        Message message = connection.receive(120, TimeUnit.SECONDS);
        if (message == null) {
            break;
        }
        message.ack();
        log.info("receive message: {}", new
String(message.getPayload()));
    }
    connection.disconnect();
}

public void sendMessage() throws Exception {
    long time = System.currentTimeMillis();
    int limit = speed < 1 ? Integer.MAX_VALUE : speed;
    LongAdder curCount = new LongAdder();

    MQTT mqtt = new MQTT();
    mqtt.setHost(server, port);
    BlockingConnection connection = mqtt.blockingConnection();
    connection.connect();
    int i = 0;
    while (repeat == -1 || i < repeat) {
        String messageValue = message + "_" + i;
        connection.publish(topic, messageValue.getBytes(),
QoS.AT_LEAST_ONCE, false);
```

```
            log.info("send message: {}", messageValue);
            if (interval > 0) {
                Thread.sleep(interval);
            }
            time = speedLimit(curCount, limit, time);
            i ++;
        }
        connection.disconnect();
    }
```

8 性能调优参数

8.1 计算节点侧的参数配置

配置名称	含义	默认值
managerLedgerMaxEntriesPerLedger	每个ledger最大的entry数量。	50000
managerLedgerMinLedgerRolloverTimeMinutes	ledger滚动的最小时间。如果超过最大entry数量则会滚动。	10
managerLedgerMaxLedgerRolloverTimeMinutes	ledger滚动的最大时间间隔	240
managerLedgerNumWorkerThreads	为了保证topic内消息按照写入顺序进行存储，broker采用单线程写入和topic相关的manager ledger entry。broker从名称相同的manager ledger worker线程中选择一个线程。增加线程数量，可以更好的避免不同topic之间写入的相互影响。	8
managerLedgerCacheSizeMB	ledger中缓存数据的内存大小。该内存从堆外内存中分配，并且在同一broker中所有的topic共享。	堆外内存的20%
managerLedgerCacheCopEntries	指定在插入缓存时是否复制消息	false
managerLedgerCacheEvictionWatemark	触发缓存逐出的水位线。	0.9
managerLedgerCacheEvictionFrequency	触发缓存逐出的频率，以每秒逐出次数表示	100
managerLedgerCacheEvicationTimeThresholdMillis	entry被逐出前，可以保留在缓存中的时间。以毫秒为单位	1000

8.2 存储节点侧的参数配置

配置名称	含义	默认值
journalDirectories	journal日志的存储目录，如果有多个用逗号隔开。为更好的进行IO读写隔离，应该将journal和ledger的存储分到不同的磁盘上。bookie使用单线程处理每个journal目录数据的写入，journal写入可能在某些情况下阻塞，	data/bookkeeper/journal

	可以指定多个journal目录来提高写入效率，但是也不能过多，过多的话会导致随机写入磁盘的次数增加。	
ledgerDirectories	ledger日志的存储目录，如果有多个用逗号隔开。	data/bookkeeper/ledger
numAddWorkerThreads	当请求处理器追加新的entry到journal日志中时，bookie要求处理器提供一个和ledger id相关的线程池提供一个线程。指定分配给处理请求的线程数，如果是零，则由IO线程直接处理写操作。	0
maxPendingAddRequestsPerThread	如果启用了添加worker线程，则此参数可限制等待请求的数量，避免队列无限制增加。如果新增的entry数量超过了此限制，则bookie拒绝添加entry的请求。	10000
journalSyncData	启用或者禁用实时刷盘。默认情况下，所有journal的日志都会sync到磁盘上，以避免在断电时丢失数据。因此，数据同步的延迟对写入吞吐量和延迟影响最大。如果将HDD作为journal磁盘，确保禁用journal sync机制，以便于entry成功写入OS page cache后，bookie客户端得到响应。	true
journalAdaptiveGroupWrites	启用或者禁用journal数据组提交。批量提交机制允许将等待执行的任务分组为小批。这种处理方式可以提高批处理的性能，同时不会使单个任务的延迟增加过多。Bookie 也可以采用同一方法来提高 journal 数据写入的吞吐量。对 journal 数据启用组提交机制可以减少磁盘操作，同时还可以避免过多的小文件写入。但是，禁用组提交可以避免增加延迟。	true
journalMaxGroupWaitMSec	指定分组提交时，journal写入的最大延迟，以毫秒为单位	1
journalBufferWritesThreshold	指定分组提交时，写缓冲区的最大值，以字节为单位。	524288

dbStorage_writeCacheMaxSizeMb	写缓冲区的最大值，以MB为单位。将 entry 写入 journal 后，entry 也会被添加到 ledger 存储中。默认情况下，bookie 使用在 DbLedgerStorage 中的指定的值作为 ledger 存储。DbLedgerStorage 是 ledger 存储的一种实现形式，它使用 RocksDB 来保存存储在 entry log 中的 entry 索引。在 entry 成功写入内存 table 后，ledger 存储中添加 entry 的请求才会完成，然后是 bookie 客户端的请求。内存 table 会定期 flush 到 entry log，并为存储在 entry log 中的 entry 创建索引，也称为 checkpoint。Checkpoint 引入了很多随机磁盘 I/O。如果 journal 目录和 ledger 目录分别位于不同设备上，则 flush 不会影响性能。但是，如果 journal 目录和 ledger 目录位于同一设备上，频繁 flush 会导致性能显著下降。可以考虑通过增加 bookie 的 flush 间隔来提升性能。但是，增加 flush 间隔后，重启 bookie 时（例如，发生故障后），恢复所需时间会增加。为实现最佳性能，内存 table 应该足够大，因此可以在 flush 间隔期间存储大量 entry。	总堆外内存的25%
flushInterval	checkpiont之间的间隔，以毫秒为单位	60000
numReadWorkerThreads	设置处理读请求的线程数，如果是零，则由IO线程处理.Bookie 服务器使用单线程处理从同一 ledger 读取请求的 entry。Bookie 服务器从与 ledger ID 相关的读取 worker 线程池中选择一个线程。	8
maxPendingReadRequestPerThread	限制读请求数量。	2500
dbStorage_rockDB_blockCacheSize	指定RocksDB块缓存的大小。从 ledger 存储中读取 entry 时，bookie 首先通过索引文件确认 entry 在 entry log 中的位置。	10%的直接内存

	DbLedgerStorage 使用 RocksDB 来存储 ledger entry 的索引。因此，需要确保分配的内存足以存储索引数据库的大部分数据，以避免索引 entry 的换入换出。为实现最佳性能，RocksDB 块缓存需要足够大以存储索引数据库的大部分数据，在一些情况下，这个值会达到约 2 GB。	
dbStorage_readAheadCacheMaxSizeMb	entry预缓存的大小，以MB为单位。启用 entry 预读缓存可以减少磁盘用于顺序读取的操作。	直接内存的25%
dbStoage_readAheadCacheBatchSize	读缓存未命中时，一次性读取的entry数量。	1000

9 客户端侧

9.1 批量发送

批处理消息指一组单条消息，并且这一组消息代表单个连续序列。使用批处理消息可以减少客户端和服务器端的开销。把消息分成小批，便可以在每个任务等待时间不增加很多的情况下，实现批处理的一些性能优势。

在 Pulsar 中使用批处理时，producer 向 broker 发送批消息。当批消息到达 broker 后，broker 与 bookie 相连接，然后 bookie 将批消息存储在 BookKeeper 中。当 consumer 从 broker 中读取消息时，broker 会将批消息分派给 consumer。

因此，组合与拆分批消息都在客户端中进行。下面的代码展示了如何为 producer 启用和配置消息批处理：

```
client.newProducer()
    .topic("topic-name")
    .enableBatching(true)
    .batchingMaxPublishDelay(2, TimeUnit.MILLISECONDS)
    .batchingMaxMessages(100)
    .batchingMaxBytes(1024 * 1024) .create();
```

在上述示例中，当批消息数量超过 100 条或批消息数据量达到 1M 时，producer 会结束掉当前的批消息并立即发送至 broker。如果在两毫秒内，上述参数值不符合条件，则 producer 也会结束掉当前的批消息并发送至 broker。

因此，参数设置将取决于消息的吞吐量和发布消息时可接受的发布延迟。

9.2 消息压缩

消息压缩可以通过消耗客户端 CPU 来减小消息大小。Pulsar 客户端支持多种压缩类型，如 lz4、zlib、zstd、snappy 等。压缩类型存储在消息元数据中，因此 consumer 可以根据需要自动适应不同的压缩类型。

启用消息批处理时，Pulsar 客户端会减小批处理大小来改进压缩。下面的代码展示了如何为 producer 启用压缩类型：

```
client.newProducer()
    .topic("topic-name")
    .compressionType(CompressionType.LZ4)
    .create();
```

9.3 增加生产者待处理消息数量

Producer 使用队列来保存等待 broker 回执的消息。因此，增加此队列大小可以增加发送消息的吞吐量。但是，这样也会使用更多内存。

下面的代码展示了如何为 producer 配置待处理消息队列的大小：

```
client.newProducer()
    .topic("topic-name")
    .maxPendingMessages(2000)
    .create();
```

在设置 maxPendingMessages 的值时，需要考虑内存对客户端应用程序的影响。用每条消息的字节数乘以 maxPengingMessages 值就可以预估对内存产生的影响。例如，假设每条消息的大小为 100 KB，则在 maxPengingMessages 设置为 2000 时，会额外增加 200 MB（2000 * 100 KB = 200,000 KB = 200 MB）的内存。

9.4 增加消费者接收队列

Consumer 接收队列决定了在用户的应用程序删除消息之前，consumer 可以累积消息的数量。增加 consumer 接收队列的大小可能会提高消费吞吐量，但同时也会响应增加内存的使用。

下面的代码展示了如何为 consumer 配置接收队列的大小：

```
client.newConsumer()
    .topic("topic-name")
    .subscriptionName("sub-name")
    .receiverQueueSize(2000)
    .subscribe();
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com