



# Apache Pulsar

云原生时代消息中间件之王





**YOU NEED KNOW**

本次课程分为 三大章节, 分别为pulsar基础篇, Pulsar高级原理篇以及Pulsar实战案例篇三部分

**Pulsar基础篇:** 此部分主要讲解pulsar的基本概念, 及其基本使用

**学习收获:** 了解为什么要使用pulsar, Pulsar的安装, 以及如何使用Pulsar, 能够在实际生产中, 构建与使用Pulsar

**Pulsar高级原理篇:** 此部分主要讲解Pulsar底层架构核心原理以及Pulsar高级组件

**学习收获:** 能够更加深入的了解Pulsar的核心实现, 以及对Pulsar核心高阶组件的使用操作, 深入掌握一门新的技术

**Pulsar实战案例篇:** 此部分主要讲解Pulsar在传智教育中实际案例

**学习收获:** 如何在企业中应用Pulsar, 掌握实战操作

# Apache Pulsar 基础篇

云原生时代消息中间件之王





# 目录

## Contents

- ◆ 为什么要学习 Apache Pulsar
- ◆ Apache Pulsar的集群架构
- ◆ Apache Pulsar的Local与分布式集群构建
- ◆ Apache Pulsar的可视化监控管理
- ◆ Apache Pulsar的主要组件介绍与命令使用
- ◆ Apache Pulsar的JAVA API相关使用操作

# 学习目标

Learning Objectives

1. 了解什么是云原生
2. 了解Pulsar的基本概念
3. 明确Pulsar的组成部分
4. 了解Pulsar和Kafka对比说明
5. 掌握Pulsar环境的构建
6. 掌握Pulsar的相关使用



# 01

## 为什么要学习 Apache Pulsar

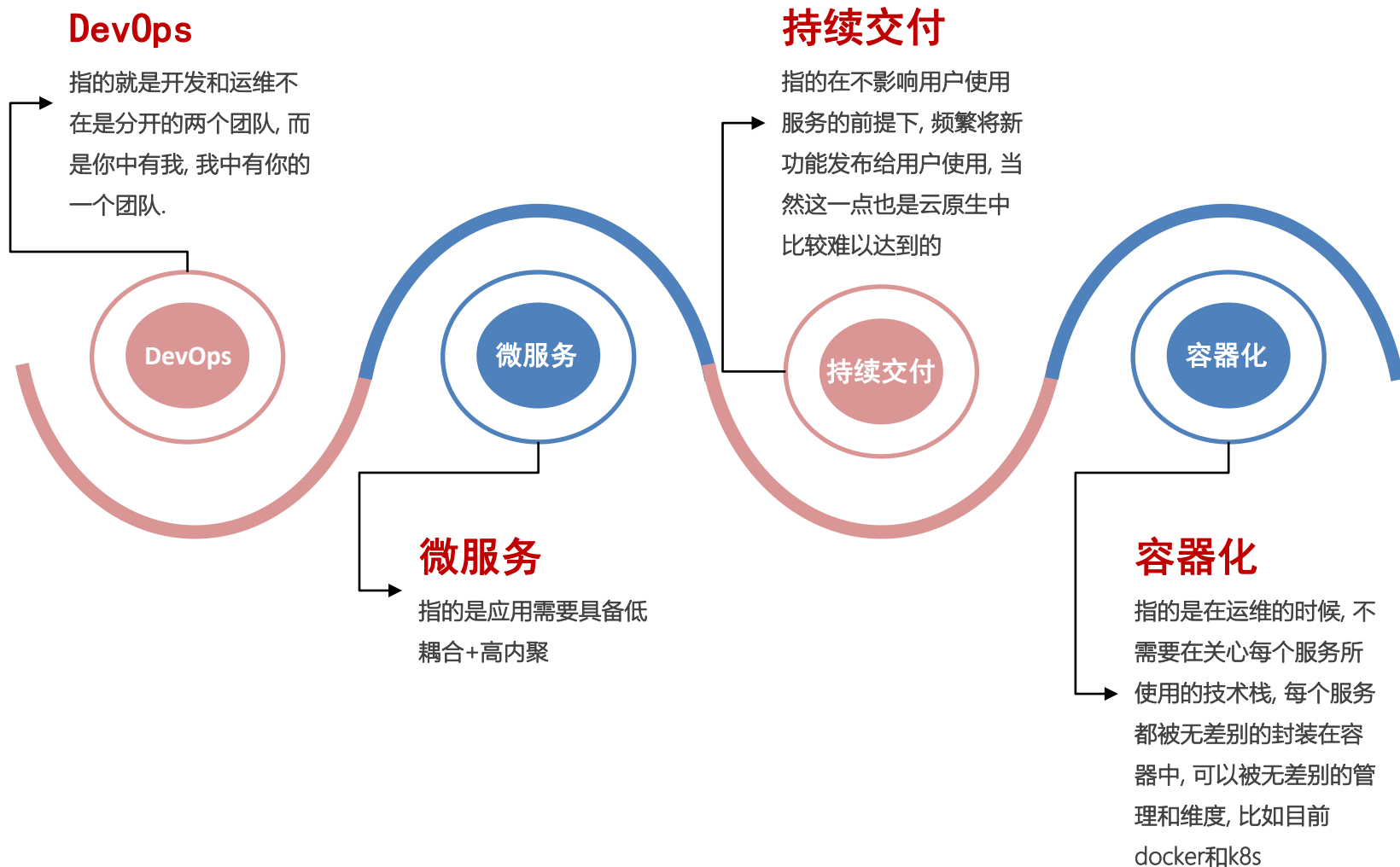
- 什么是云原生
- Apache pulsar基本介绍
- Apache Pulsar组件介绍
- Pulsar与kafka的对比

## 什么是云原生

云原生的概念是2013年Matt Stine提出的, 到目前为止, 云原生的概念发生了多次变更, 目前最新对云原生定义为: DevOps+持续交付+微服务+容器

而符合云原生架构的应用程序是: 采用开源堆栈(K8S+Docker)进行容器化, 基于微服务架构提高灵活性和可维护性, 借助敏捷方法、DevOps支持持续迭代和运维自动化, 利用云平台设施实现弹性伸缩、动态调度、优化资源利用率。

## 什么是云原生







# 01

## 为什么要学习 Apache Pulsar

- 什么是云原生
- **Apache pulsar基本介绍**
- Apache Pulsar组件介绍
- Pulsar与kafka的对比

## Apache Pulsar 基本介绍

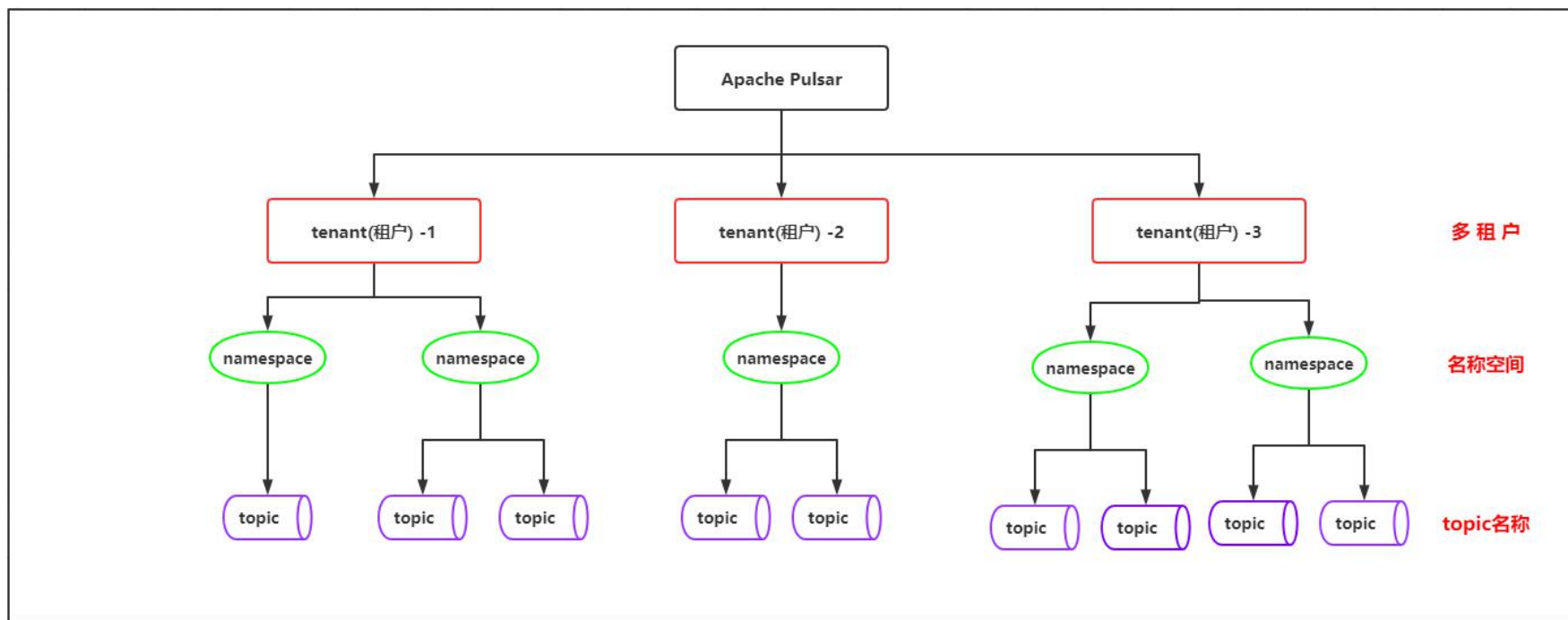
Apache Pulsar 是一个云原生企业级的发布订阅（pub-sub）消息系统，最初由Yahoo开发，并于2016年底开源，现在是Apache软件基金会顶级开源项目。Pulsar在Yahoo的生产环境运行了三年多，助力Yahoo的主要应用，如Yahoo Mail、Yahoo Finance、Yahoo Sports、Flickr、Gemini广告平台和Yahoo分布式键值存储系统Sherpa。

Apache Pulsar的功能与特性：

- 1) 多租户模式：
- 2) 灵活的消息系统
- 3) 云原生架构
- 4) segmented Streams (分片流)
- 5) 支持跨地域复制

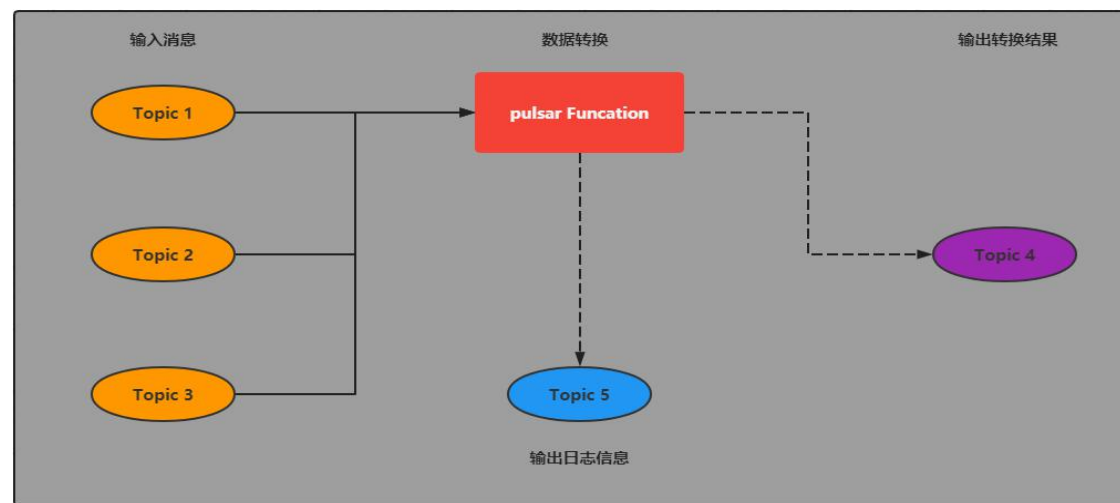
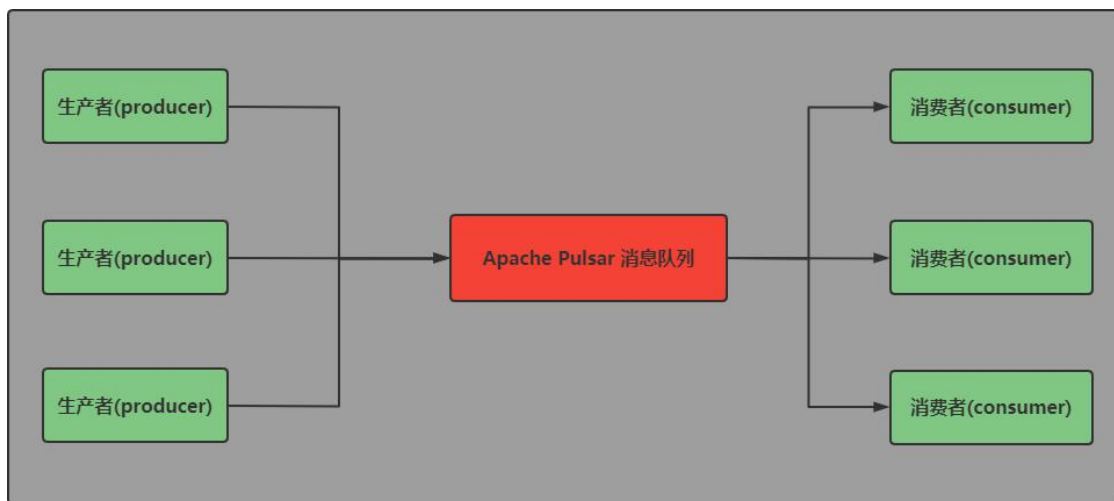
## 多租户模式

- 租户和命名空间（namespace）是 Pulsar 支持多租户的两个核心概念。
- 在租户级别，Pulsar 为特定的租户预留合适的存储空间、应用授权与认证机制。
- 在命名空间级别，Pulsar 有一系列的配置策略（policy），包括存储配额、流控、消息过期策略和命名空间之间的隔离策略。



## 灵活的消息系统

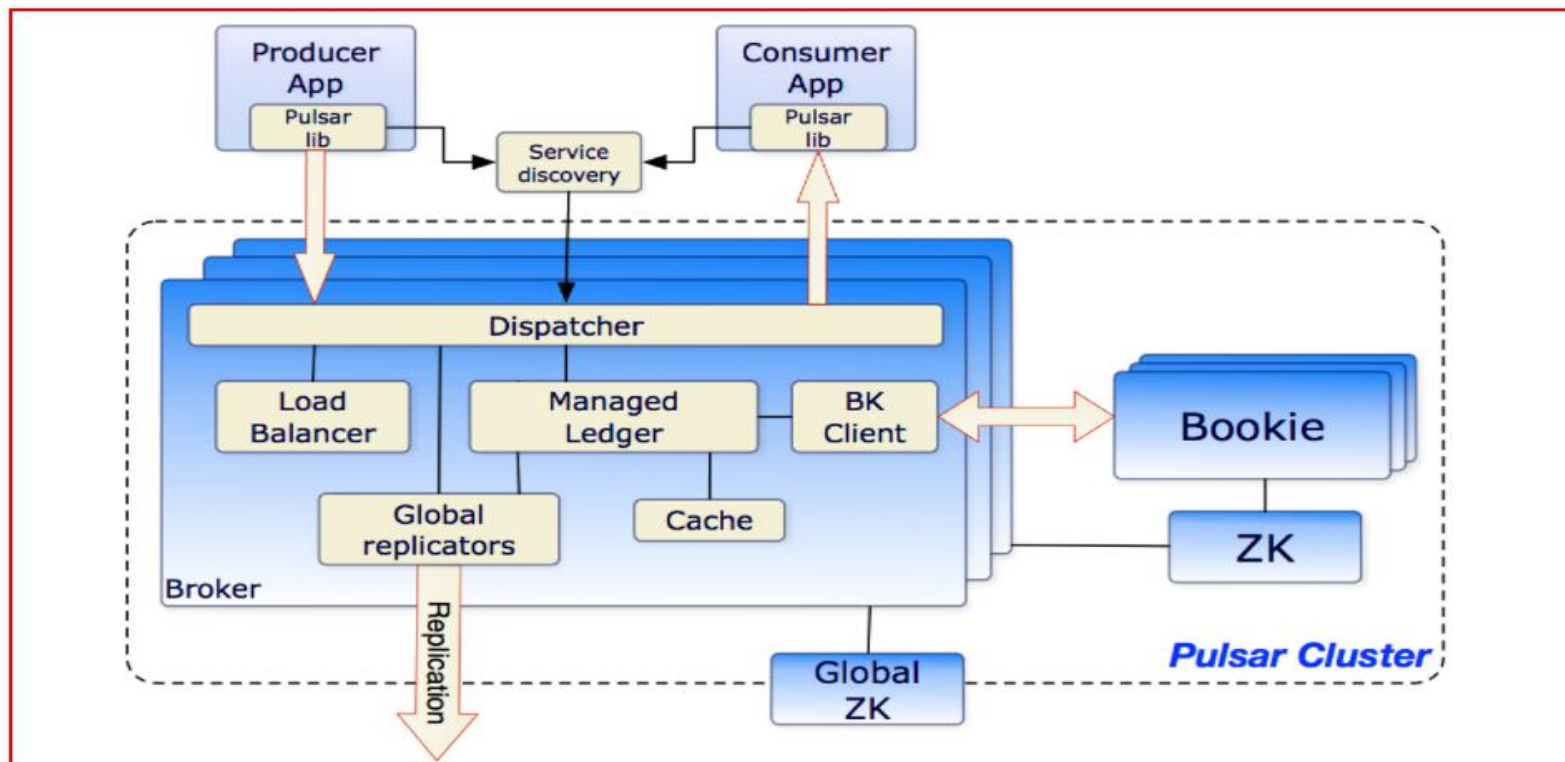
- Pulsar 做了队列模型和流模型的统一，在 Topic 级别只需保存一份数据，同一份数据可多次消费。以流式、队列等方式计算不同的订阅模型大大提升了灵活性。
- 同时pulsar通过事务采用Exactly-Once(精准一次)在进行消息传输过程中，可以确保数据不丢不重





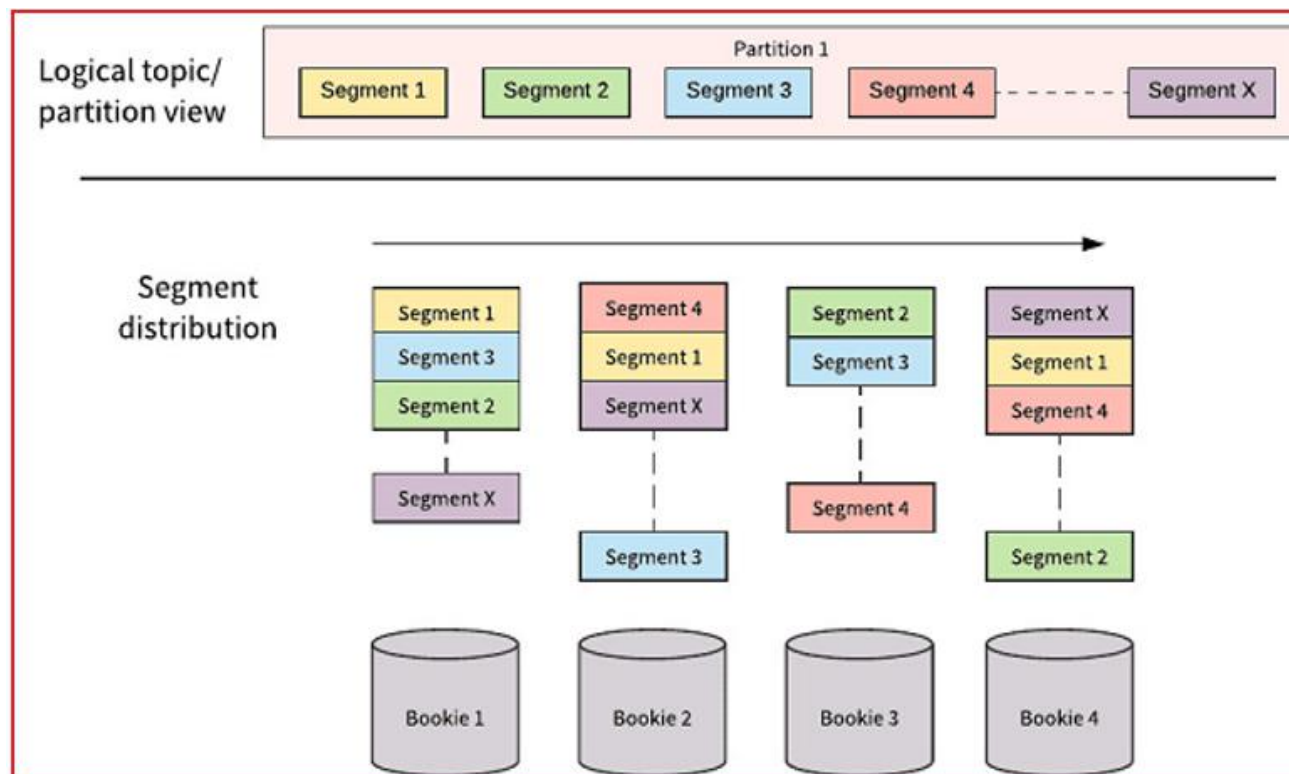
## 云原生架构

- Pulsar 使用计算与存储分离的云原生架构，数据从 Broker 搬离，存在共享存储内部。上层是无状态 Broker，复制消息分发和服务；下层是持久化的存储层 Bookie 集群。Pulsar 存储是分片的，这种构架可以避免扩容时受限制，实现数据的独立扩展和快速恢复



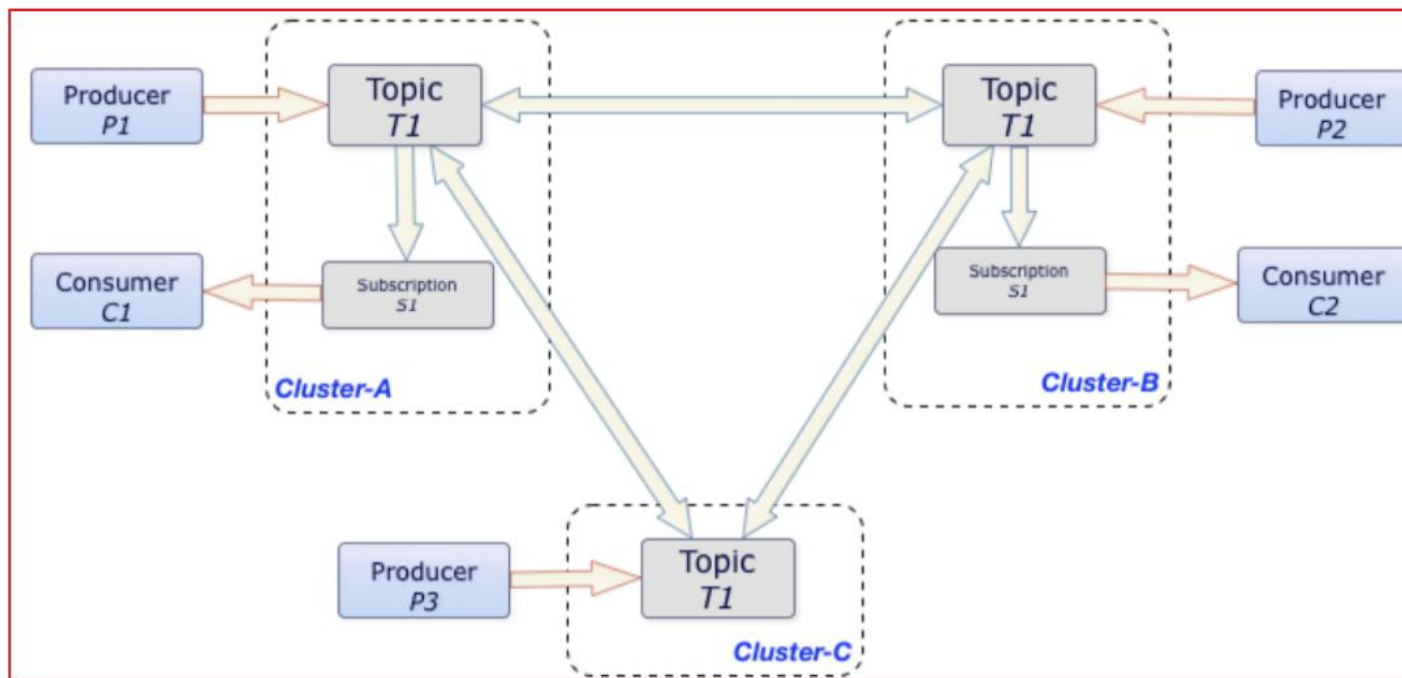
## Segmented Streams (分片流)

- Pulsar 将无界的数据看作是分片的流，分片分散存储在分层存储（tiered storage）、BookKeeper 集群和 Broker 节点上，而对外提供一个统一的、无界数据的视图。其次，不需要用户显式迁移数据，减少存储成本并保持近似无限的存储。



## 支持跨地域复制

- Pulsar 中的跨地域复制是将 Pulsar 中持久化的消息在多个集群间备份。在 Pulsar 2.4.0 中新增了复制订阅模式 (Replicated-subscriptions)，在某个集群失效情况下，该功能可以在其他集群恢复消费者的消费状态，从而达到热备模式下消息服务的高可用。





# 01

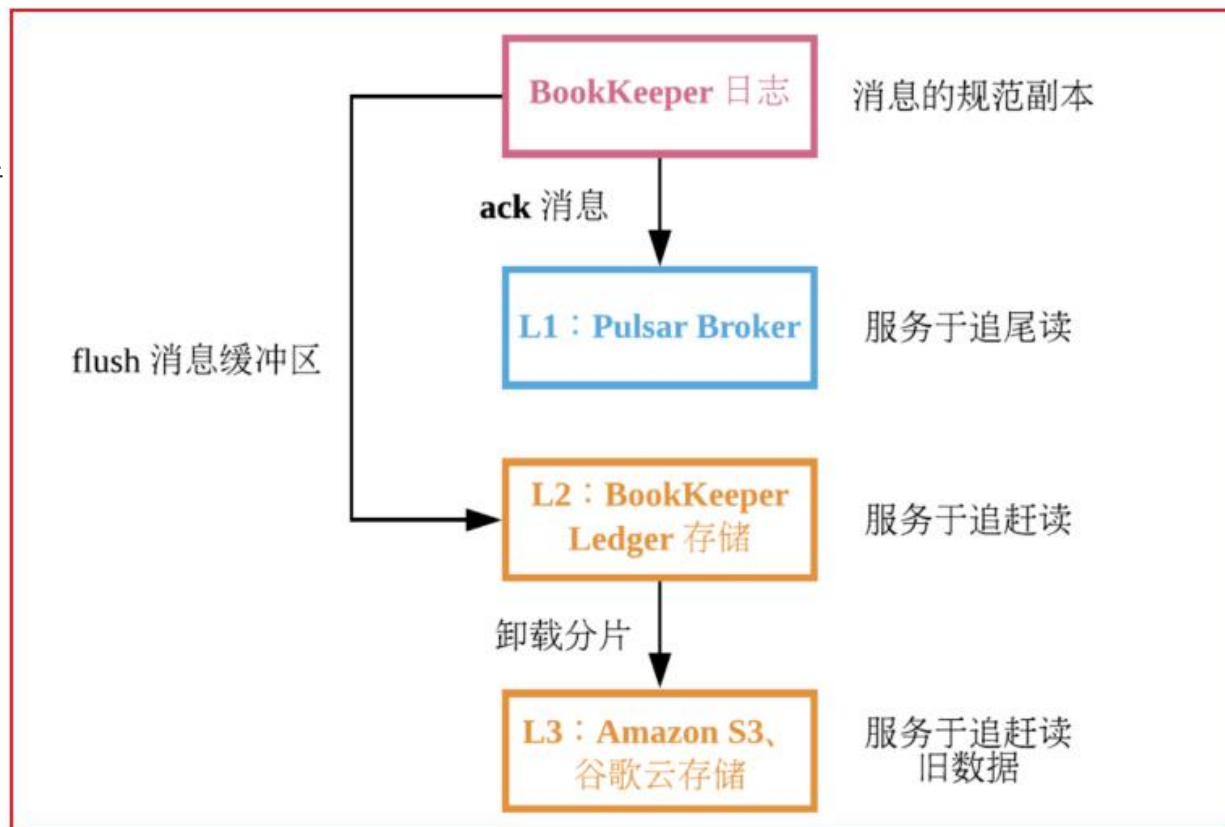
## 为什么要学习 Apache Pulsar

- 什么是云原生
- Apache pulsar基本介绍
- **Apache Pulsar组件介绍**
- Pulsar与kafka的对比



## 层级存储

- Infinite Stream: 以流的方式永久保存原始数据
- 分区的容量不再受限制
- 充分利用云存储或现有的廉价存储（例如 HDFS）
- 数据统一表征：客户端无需关心数据究竟存储在哪里



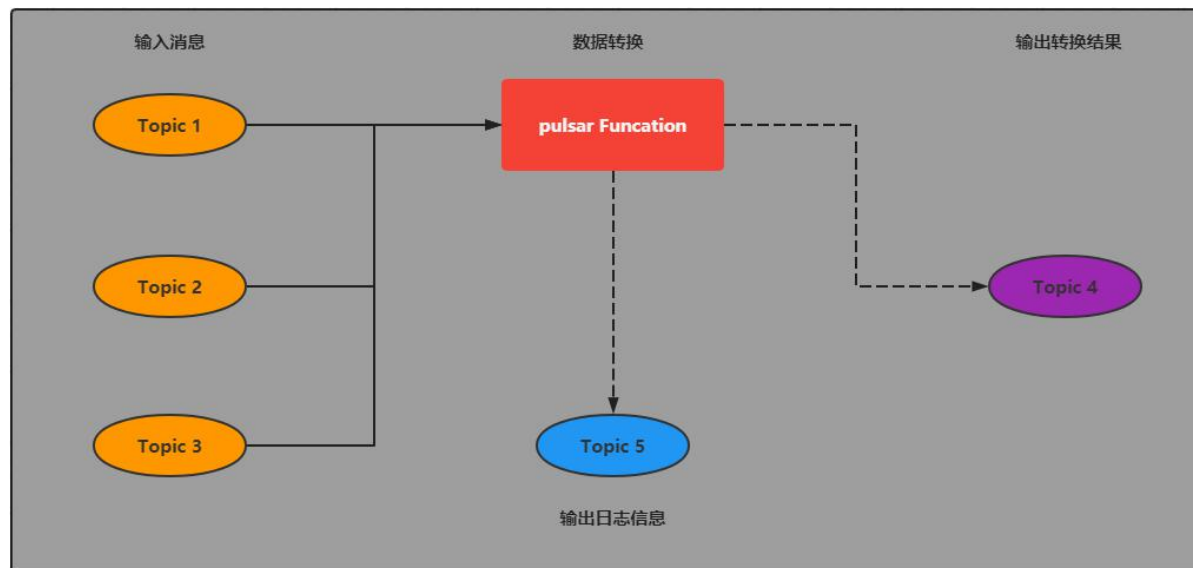
## Pulsar IO(Connector) 连接器

- Pulsar IO 分为输入（Input）和输出（Output）两个模块，输入代表数据从哪里来，通过 Source 实现数据输入。输出代表数据要往哪里去，通过 Sink 实现数据输出。
- Pulsar 提出了 Connector（也称为 Pulsar IO），用于解决 Pulsar 与周边系统的集成问题，帮助用户高效完成工作。
- 目前 pulsar IO 支持非常多的连接集成操作：例如HDFS、Spark、Flink、Flume、ES、HBase等



## Pulsar Functions(轻量级计算框架)

- Pulsar Functions 是一个轻量级的计算框架，可以给用户提供一个部署简单、运维简单、API 简单的 FASS (Function as a service) 平台。Pulsar Functions 提供基于事件的服务，支持有状态与无状态的多语言计算，是对复杂的大数据处理框架的有力补充。
- Pulsar Functions 的设计灵感来自于 Heron 这样的流处理引擎，Pulsar Functions 将会拓展 Pulsar 和整个消息领域的未来。使用 Pulsar Functions，用户可以轻松地部署和管理 function，通过 function 从 Pulsar topic 读取数据或者生产新数据到 Pulsar topic。





# 01

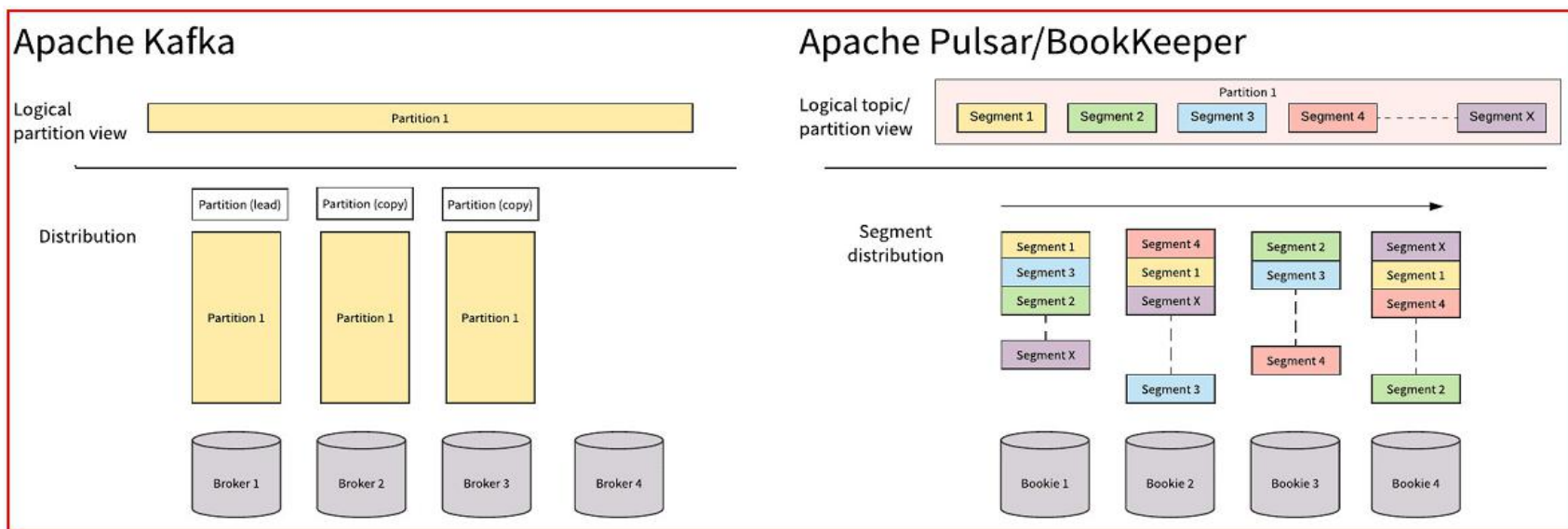
## 为什么要学习 Apache Pulsar

- 什么是云原生
- Apache pulsar基本介绍
- Apache Pulsar组件介绍
- Pulsar与kafka的对比

## Pulsar和Kafka的对比介绍说明

- 1) 模型概念
  - Kafka: producer - topic - consumer group - consumer
  - Pulsar: producer - topic -subscription- consumer
- 2) 消息消费模式
  - Kafka: 主要集中在流(Stream) 模式，对单个partition是独占消费，没有共享(Queue)的消费模式
  - Pulsar: 提供了统一的消息模型和API. 流(Stream) 模式 - 独占和故障切换订阅方式；队列(Queue)模式 - 共享订阅的方式
- 3) 消息确认(ack)
  - Kafka: 使用偏移量 offset
  - Pulsar: 使用专门的cursor管理. 累积确认和kafka效果一样；提供单条或选择性确认
- 4) 消息保留：
  - Kafka: 根据设置的保留期来删除消息，有可能消息没被消费，过期后被删除，不支持TTL
  - Pulsar: 消息只有被所有订阅消费后才会删除，不会丢失数据,. 也运行设置保留期，保留被消费的数据 . 支持TTL

Apache Kafka和Apache Pulsar都有类似的消息概念。客户端通过主题与消息系统进行交互。每个主题都可以分为多个分区。然而，Apache Pulsar和Apache Kafka之间的根本区别在于Apache Kafka是以分区为存储中心，而Apache Pulsar是以Segment为存储中心。



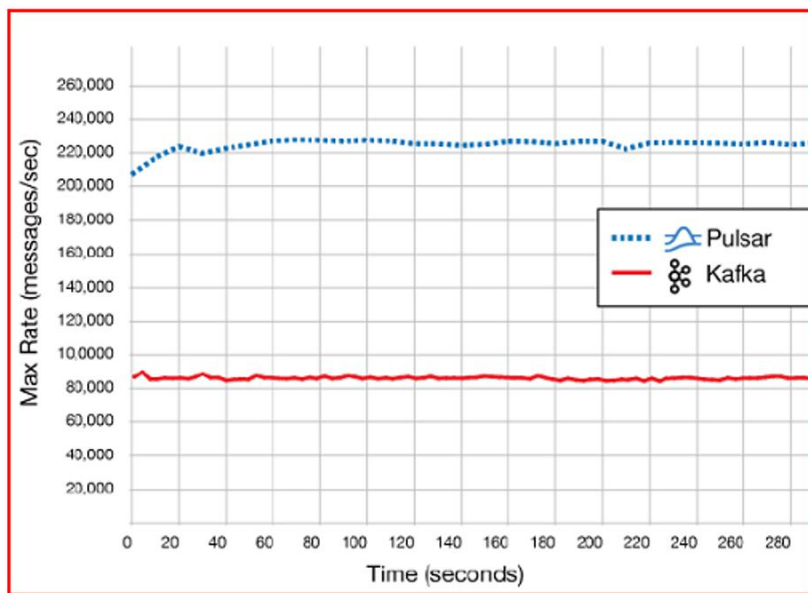
对比总结：

Apache Pulsar将高性能的流（Apache Kafka所追求的）和灵活的传统队列（RabbitMQ所追求的）结合到一个统一的消息模型和API中。Pulsar使用统一的API为用户提供一个支持流和队列的系统，且具有同样的高性能。

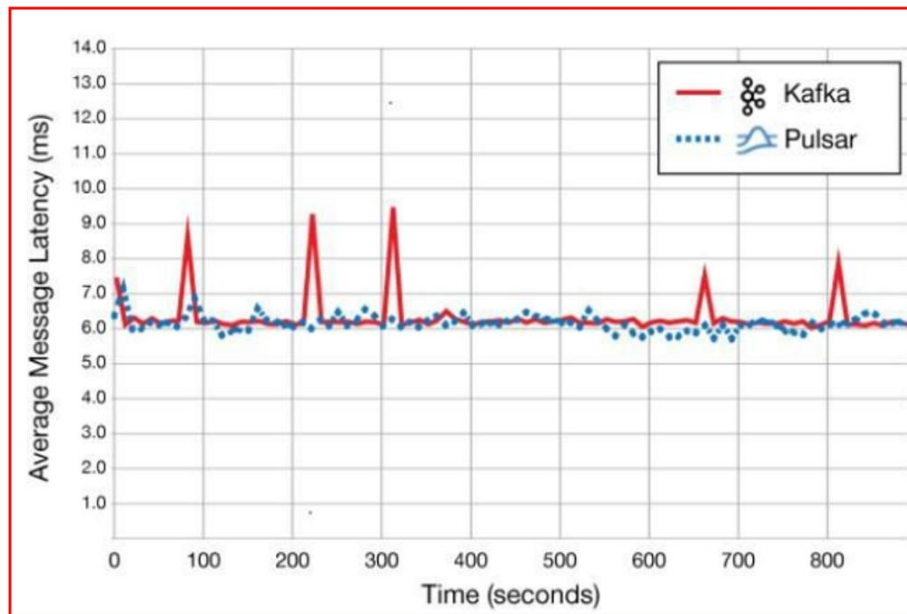
性能对比：

Pulsar 表现最出色的就是性能，Pulsar 的速度比 Kafka 快得多，美国德克萨斯州一家名为 GigaOm (<https://gigaom.com/>) 的技术研究和分析公司对 Kafka 和 Pulsar 的性能做了比较，并证实了这一点。

与 Kafka 相比，Pulsar 的速度提升了 2.5 倍，延迟降低了 40%。



请注意，该性能比较是针对 1 个分区的 1 个主题，其中包含 100 字节消息。而 Pulsar 每秒可发送 220,000+ 条消息。



## 扩展说明：kafka目前存在的痛点

- 1) Kafka 很难进行扩展，因为 Kafka 把消息持久化在 broker 中，迁移主题分区时，需要把分区的数据完全复制到其他 broker 中，这个操作非常耗时。
- 2) 当需要通过更改分区大小以获得更多的存储空间时，会与消息索引产生冲突，打乱消息顺序。因此，如果用户需要保证消息的顺序，Kafka 就变得非常棘手了。
- 3) 如果分区副本不处于 ISR（同步）状态，那么 leader 选取可能会紊乱。一般地，当原始主分区出现故障时，应该有一个 ISR 副本被征用，但是这点并不能完全保证。若在设置中并未规定只有 ISR 副本可被选为 leader 时，选出一个处于非同步状态的副本做 leader，这比没有 broker 服务该 partition 的情况更糟糕。
- 4) 使用 Kafka 时，你需要根据现有的情况并充分考虑未来的增量计划，规划 broker、主题、分区和副本的数量，才能避免 Kafka 扩展导致的问题。这是理想状况，实际情况很难规划，不可避免会出现扩展需求。
- 5) Kafka 集群的分区再均衡会影响相关生产者和消费者的性能。
- 6) 发生故障时，Kafka 主题无法保证消息的完整性（特别是遇到第 3 点中的情况，需要扩展时极有可能丢失消息）。
- 7) 使用 Kafka 需要和 offset 打交道，这点让人很头痛，因为 broker 并不维护 consumer 的消费状态。
- 8) 如果使用率很高，则必须尽快删除旧消息，否则就会出现磁盘空间不够用的问题。
- 9) 众所周知，Kafka 原生的跨地域复制机制（MirrorMaker）有问题，即使只在两个数据中心也无法正常使用跨地域复制。因此，甚至 Uber 都不得不创建另一套解决方案来解决这个问题，并将其称为 uReplicator (<https://eng.uber.com/ureplicator/>)。
- 10) 要想进行实时数据分析，就不得不选用第三方工具，如 Apache Storm、Apache Heron 或 Apache Spark。同时，你需要确保这些第三方工具足以支撑传入的流量。
- 11) Kafka 没有原生的多租户功能来实现租户的完全隔离，它是通过使用主题授权等安全功能来完成的。





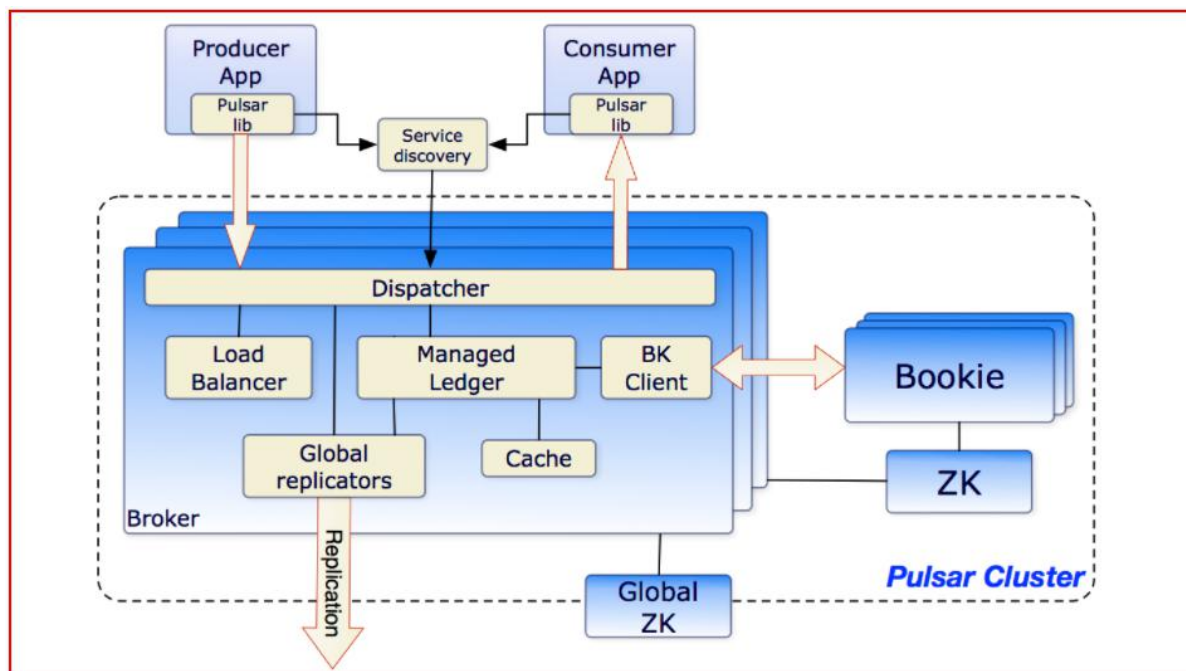
## 02

# Apache Pulsar集群架构

- 架构基本介绍
- Apache Pulsar提供的组件介绍

单个 Pulsar 集群由以下三部分组成：

- 多个 broker 负责处理和负载均衡 producer 发出的消息，并将这些消息分派给 consumer；Broker 与 Pulsar 配置存储交互来处理相应的任务，并将消息存储在 BookKeeper 实例中（又称 bookies）；Broker 依赖 ZooKeeper 集群处理特定的任务，等等。
- 多个 bookie 的 BookKeeper 集群负责消息的持久化存储。
- 一个 zookeeper 集群，用来处理多个 Pulsar 集群之间的协调任务。





## 02

# Apache Pulsar集群架构

- 架构基本介绍
- Apache Pulsar提供的组件介绍

## Brokers介绍

Pulsar的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。

出于性能考虑，消息通常从Managed Ledger缓存中分派出去，除非积压超过缓存大小。如果积压的消息对于缓存来说太大了，则Broker将开始从BookKeeper那里读取Entries（Entry同样是BookKeeper中的概念，相当于一条记录）。

最后，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

## Zookeeper的元数据存储

Pulsar使用Apache Zookeeper进行元数据存储、集群配置和协调

- **配置存储：**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，BookKeeper ledger信息（这个是BookKeeper本身所依赖的）等等。

## 基于bookKeeper持久化存储

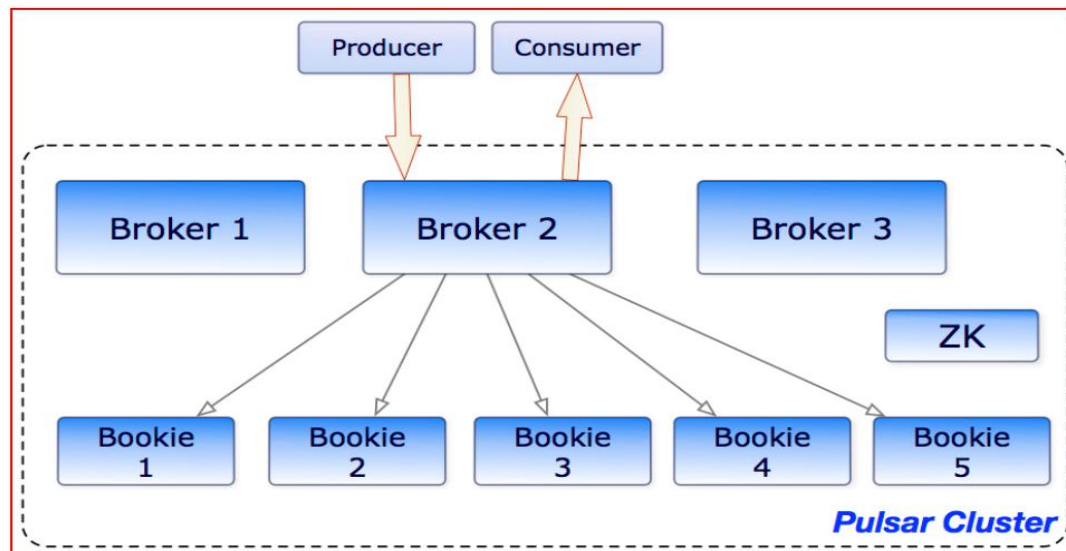
Apache Pulsar 为应用程序提供有保证的信息传递，如果消息成功到达broker，就认为其预期到达了目的地。

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。这种消息传递模式通常称为持久消息传递。在Pulsar内部，所有消息都被保存并同步N份，例如，2个服务器保存四份，每个服务器上面都有镜像的RAID存储。

Pulsar用 Apache bookKeeper作为持久化存储。bookKeeper是一个分布式的预写日志（WAL）系统，有如下几个特性特别适合Pulsar的应用场景：

- 1) 使pulsar能够利用独立的日志,称为ledgers. 可以随着时间的推移为topic创建多个Ledgers
- 2) 它为处理顺序消息提供了非常有效的存储
- 3) 保证了多系统挂掉时Ledgers的读取一致性
- 4) 提供不同的Bookies之间均匀的IO分布的特性
- 5) 它在容量和吞吐量方面都具有水平伸缩性。能够通过增加bookies立即增加容量到集群中, 并提升吞吐量
- 6) Bookies被设计成可以承载数千的并发读写的ledgers。使用多个磁盘设备（一个用于日志，另一个用于一般存储），这样Bookies可以将读操作的影响和对于写操作的延迟分隔开。

## 基于bookKeeper持久化存储



Ledger是一个只追加的数据结构，并且只有一个写入器，这个写入器负责多个bookKeeper存储节点（就是Bookies）的写入。Ledger的条目会被复制到多个bookies。Ledgers本身有着非常简单的语义：

- Pulsar Broker可以创建ledger，添加内容到ledger和关闭ledger。
- 当一个ledger被关闭后，除非明确的要写数据或者是因为写入器挂掉导致ledger关闭，ledger只会以只读模式打开。
- 最后，当ledger中的条目不再有用的时候，整个ledger可以被删除（ledger分布是跨Bookies的）。

## Pulsar 代理

Pulsar客户端和Pulsar集群交互的一种方式就是直连Pulsar brokers。然而，在某些情况下，这种直连既不可行也不可取，因为客户端并不知道broker的地址。例如在云环境或者Kubernetes以及其他类似的系统上面运行Pulsar，直连brokers就基本上不可能了。

Pulsar proxy 为这个问题提供了一个解决方案，为所有的broker提供了一个网关，如果选择运行了Pulsar Proxy. 所有的客户都会通过这个代理而不是直接与brokers通信





# 03

## Apache Pulsar集群部署及可视化 监控部署

- Apache Pulsar本地Local模式
- Apache Pulsar分布式集群模式
- Apache Pulsar可视化监控部署

## Apache Pulsar的Local模式构建

Standalone Local单机本地模式, 是pulsar最简单的安装方式, 此种方式仅适用于测试学习使用, 并无法作为开发中使用

### 下载Apache pulsar2.8.1

<https://pulsar.apache.org/en/download/>

#### Current version (Stable) 2.8.1

Release	Link	Crypto files
Binary	<a href="#">apache-pulsar-2.8.1-bin.tar.gz</a>	asc, sha512
Source	<a href="#">apache-pulsar-2.8.1-src.tar.gz</a>	asc, sha512

服务器系统要求:

Currently, Pulsar is available for 64-bit macOS, Linux, and Windows. To use Pulsar, you need to install 64-bit JRE/JDK 8 or later versions. (目前, Pulsar可用于64位macOS、Linux和Windows。使用Pulsar需要安装64位JRE/JDK 8或更高版本。)

## Apache Pulsar的Local模式构建

- 第一步: 上传Pulsar安装包到linux服务器中,并解压

```
cd /export/software
rz 上传即可apache-pulsar-2.8.1-bin.tar.gz

tar -zxvf apache-pulsar-2.8.1-bin.tar.gz -C /export/server
```

构建软连接:

```
cd /export/server
ln -s apache-pulsar-2.8.1-bin puslar_2.8.1
```

- 第二步: 启动单机模式Pulsar

```
cd /export/server/puslar_2.8.1/bin
./pulsar standalone
```

```
15:38:55.937 [worker-scheduler-0] INFO org.apache.pulsar.functions.worker.SchedulerManager - Schedule summary - execution time: 0.085389604 sec | total unassigned: 0 | stats: {"Added": 0, "Updated": 0, "removed": 0}
{
  "c-standalone-fw-localhost-8080" : {
    "originalNumAssignments" : 0,
    "finalNumAssignments" : 0,
    "instancesAdded" : 0,
    "instancesRemoved" : 0,
    "instancesUpdated" : 0,
    "alive" : true
  }
}
```

启动后, 会一直卡住前台进程, 如果需要关闭pulsar, 直接ctrl +c 即可

## Apache Pulsar的Local模式基本使用

在pulsar的bin目录下，专门提供了一个pulsar-client的客户端工具，Pulsar-Client工具允许使用者在运行的集群中消费并发送消息到Pulsar Topic中

- 模拟开启消费者监听数据

```
./pulsar-client consume my-topic -s "first-subscription"
```

```
}
15:43:43.178 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [my-topic][first-
subscription] Subscribing to topic on cnx [id: 0x57b63cd9, L:/127.0.0.1:34240 - R:localhost/127.0.0.1:66
50], consumerId 0
15:43:43.282 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [my-topic][first-
subscription] Subscribed to topic on localhost/127.0.0.1:6650 -- consumer: 0
```

■ ← 等待消息的发送

- 模拟生产一条数据

```
./pulsar-client produce my-topic --messages "hello-pulsar"
```

```
15:45:48.388 [main] INFO org.apache.pulsar.client.cli.PulsarClientTool - 1 messages successfully produc
ed
[root@node1 bin]#
```

消息发送成功

```
----- got message ----- 消费端:
key:[null], properties:[], content:hello-pulsar 消息内容
15:45:46.401 [main] INFO org.apache.pulsar.client.impl.PulsarClientImpl - Client closing. URL: pulsar:/
localhost:6650/
15:45:46.427 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [my-topic] [first-
subscription] Closed consumer
15:45:46.435 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ClientCnx - [id: 0x57b63cd9, L:/
127.0.0.1:34240 ! R:localhost/127.0.0.1:6650] Disconnected
15:45:48.459 [main] INFO org.apache.pulsar.client.cli.PulsarClientTool - 1 messages successfully consum
ed
成功消费掉数据
```



# 03

## Apache Pulsar集群部署及可视化 监控部署

- Apache Pulsar本地Local模式
- **Apache Pulsar分布式集群模式**
- Apache Pulsar可视化监控部署

## Apache Pulsar的分布式集群模式构建

搭建 Pulsar 集群至少需要 3 个组件：ZooKeeper 集群、BookKeeper 集群和 broker 集群（Broker 是 Pulsar 的自身实例）。这三个集群组件如下：

- ZooKeeper 集群（3 个 ZooKeeper 节点组成）
- bookie 集群（也称为 BookKeeper 集群，3 个 BookKeeper 节点组成）
- broker 集群（3 个 Pulsar 节点组成）

Pulsar 的安装包已包含了搭建集群所需的各个组件库。无需单独下载 ZooKeeper 安装包和 BookKeeper 安装包。（在实际中,zookeeper我们并不仅仅应用在pulsar上, 包括HBase等其他的组件也需要依赖, 所以我们此处zookeeper使用外置zk集群环境）

注意: 如果是在内网测试环境搭建集群，为了避免防火墙造成端口开启繁琐，可以关闭服务器防火墙。

## Apache Pulsar的分布式集群模式构建

分布式模式 最低需要三台服务器进行安装操作, 本次我们将采用VMware进行虚拟化三台机器进行, 并且每台机器已经提前将JDK1.8和zookeeper集群安装配置完成了, 如有需要, 可参考提供的前置安装笔记

- 第一步: 将下载的pulsar安装包上传到linux服务器, 并解压

```
cd /export/software
rz 上传即可apache-pulsar-2.8.1-bin.tar.gz

tar -zxvf apache-pulsar-2.8.1-bin.tar.gz -C /export/server
```

构建软连接:

```
cd /export/server
ln -s apache-pulsar-2.8.1-bin puslar_2.8.1
```

```
[root@node1 server]# ll
总用量 32
drwxr-xr-x  8 root  root    132 12月 13 16:29 apache-pulsar-2.8.1
drwxr-xr-x  7 10143 10143   245 12月 11 2019 jdk1.8.0_241
lrwxrwxrwx  1 root  root     20 12月 13 16:29 pulsar_2.8.1 -> apache-pulsar-2.8.1/
lrwxrwxrwx  1 root  root     16 10月 24 01:23 zookeeper -> zookeeper-3.4.6/
drwxr-xr-x 11 1000  1000  4096 10月 24 01:24 zookeeper-3.4.6
-rw-r--r--  1 root  root 28035 10月 24 01:30 zookeeper.out
[root@node1 server]# pwd
/export/server
```



## Apache Pulsar的分布式集群模式构建

- 第二步：修改bookkeeper集群配置文件

```
cd /export/server/pulsar_2.8.1/conf/  
vim bookkeeper.conf
```

修改其第56行:修改本地ip地址

advertisedAddress=node1.itcast.cn

修改其39行:

journalDirectory=/export/server/pulsar\_2.8.1/tmp/journal

修改其389行:

ledgerDirectories=/export/server/pulsar\_2.8.1/tmp/ledger

修改602行:

zkServers=node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181

- 第三步：修改broker集群的配置文件

```
cd /export/server/pulsar_2.8.1/conf/  
vim broker.conf
```

修改第98行: 修改集群的名称

clusterName=pulsar-cluster

修改第23行: 配置zookeeper地址

zookeeperServers=node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181

修改第26行: 配置zookeeper地址

configurationStoreServers=node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181

修改第44行: 更改为本地ip地址

advertisedAddress=node1.itcast.cn



## Apache Pulsar的分布式集群模式构建

- 第四步：将配置好bookies目录和brokers目录发送到第二台和第三台

```
cd /export/server
scp -r apache-pulsar-2.8.1/ node2:$PWD
scp -r apache-pulsar-2.8.1/ node3:$PWD
```

在第二台和第三台节点上分别配置软连接

```
cd /export/server
ln -s apache-pulsar-2.8.1/ pulsar_2.8.1
```

- 第五步：修改第二台和第三台的broker的地址和bookies地址

```
node2:
cd /export/server/pulsar_2.8.1/conf/
vim bookkeeper.conf
修改其第56行:修改本地ip地址
advertisedAddress=node2.itcast.cn
```

```
vim broker.conf
修改第44行: 更改为本地ip地址
advertisedAddress=node2.itcast.cn
```

第三台节点: 都更改为对应IP地址或者主机名即可

## Apache Pulsar的分布式集群模式启动

- 第一步: 首先启动zookeeper集群

```
cd /export/server/zookeeper/bin
```

```
./zkServer.sh start
```

注意: 三个节点依次都要启动, 启动后 通过

```
./zkServer.sh status
```

查看状态, 必须看到一个leader 和两个follower 才可以使用

- 第二步: 初始化元数据(此操作, 仅需要初始化一次即可)

首先初始化Pulsar集群元数据:

```
cd /export/server/pulsar_2.8.1/bin
```

```
./pulsar initialize-cluster-metadata \
```

```
--cluster pulsar-cluster \
```

```
--zookeeper node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181 \
```

```
--configuration-store node1.itcast.cn:2181,node2.itcast.cn:2181,node3.itcast.cn:2181 \
```

```
--web-service-url http://node1.itcast.cn:8080,node2.itcast.cn:8080,node3.itcast.cn:8080 \
```

```
--web-service-url-tls https://node1.itcast.cn:8443,node2.itcast.cn:8443,node3.itcast.cn:8443 \
```

```
--broker-service-url pulsar://node1.itcast.cn:6650,node2.itcast.cn:6650,node3.itcast.cn:6650 \
```

```
--broker-service-url-tls pulsar+ssl://node1.itcast.cn:6651,node2.itcast.cn:6651,node3.itcast.cn:6651
```

接着初始化bookkeeper集群: 若出现提示输入Y/N: 请输入Y

```
./bookkeeper shell metaformat
```

## Apache Pulsar的分布式集群模式启动

- 第三步: 启动bookkeeper服务

```
cd /export/server/pulsar_2.8.1/bin
```

```
./pulsar-daemon start bookie
```

注意: 三个节点都需要依次启动

验证是否启动: 可三台都检测

```
./bookkeeper shell bookiesanity
```

提示:

Bookie sanity test succeeded 认为启动成功

- 第四步: 启动Broker

```
cd /export/server/pulsar_2.8.1/bin
```

```
./pulsar-daemon start broker
```

注意: 三个节点都需要依次启动

检测是否启动:

```
./pulsar-admin brokers list pulsar-cluster
```

```
[root@node1 bin]# ./pulsar-admin brokers list pulsar-cluster
"node3.itcast.cn:8080"
"node1.itcast.cn:8080"
"node2.itcast.cn:8080"
```

## Apache Pulsar的分布式集群模式测试

在pulsar的bin目录下, 专门提供了一个pulsar-client的客户端工具, Pulsar-Client工具允许使用者在运行的集群中消费并发送消息到Pulsar Topic中

- 模拟开启消费者监听数据

```
./pulsar-client consume persistent://public/default/test -s "consumer-test"
```

```
18:07:27.736 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConnectionPool - [[id: 0x559280ea, L:/192.168.88.161:36454 - R:node2.itcast.cn/192.168.88.162:6650]] Connected to server
18:07:28.037 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [persistent://public/default/test][consumer-test] Subscribing to topic on cnx [id: 0x559280ea, L:/192.168.88.161:36454 - R:node2.itcast.cn/192.168.88.162:6650], consumerId 0
18:07:28.745 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [persistent://public/default/test][consumer-test] Subscribed to topic on node2.itcast.cn/192.168.88.162:6650 -- consumer: 0
    开始监听, 等待消息
```

- 模拟生产一条数据

```
./pulsar-client produce persistent://public/default/test --messages "hello-pulsar"
```

```
15:45:48.388 [main] INFO org.apache.pulsar.client.cli.PulsarClientTool - 1 messages successfully produced
```

消息发送成功

```
[root@node1 bin]#
```

```
----- got message -----
key:[null], properties:[], content:hello-pulsar 消息内容
18:10:29.733 [main] INFO org.apache.pulsar.client.impl.PulsarClientImpl - Client closing. URL: pulsar://localhost:6650/
18:10:29.749 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [persistent://public/default/test][consumer-test] Closed consumer
18:10:29.758 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ClientCnx - [id: 0x9e96bb3b, L:/127.0.0.1:34332 ! R:localhost/127.0.0.1:6650] Disconnected
18:10:29.766 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ClientCnx - [id: 0x1fb2c35f, L:/192.168.88.161:36458 ! R:node2.itcast.cn/192.168.88.162:6650] Disconnected
18:10:31.785 [main] INFO org.apache.pulsar.client.cli.PulsarClientTool - 1 messages successfully consumed 接收到1条消息
```



# 03

## Apache Pulsar集群部署及可视化 监控部署

- Apache Pulsar本地Local模式
- Apache Pulsar分布式集群模式
- Apache Pulsar可视化监控部署

## Apache Pulsar的可视化监控部署

- 第一步: 下载Pulsar-Manager

下载地址:

```
https://dist.apache.org/repos/dist/release/pulsar/pulsar-manager/pulsar-manager-0.2.0/apache-pulsar-manager-0.2.0-bin.tar.gz
```

- 第二步: 上传到服务器, 并解压

```
cd /export/software
```

```
rz 上传 apache-pulsar-manager-0.2.0-bin.tar.gz
```

解压操作:

```
tar -xzf apache-pulsar-manager-0.2.0-bin.tar.gz -C /export/server/
```

```
cd /export/server/pulsar-manager
```

接着再次解压:

```
tar -xvf pulsar-manager.tar
```

- 第三步: 拷贝dist包到 pulsar-manager目录下并更名为ui

```
cd /export/server/pulsar-manager/pulsar-manager
```

```
cp -r ../dist ui
```

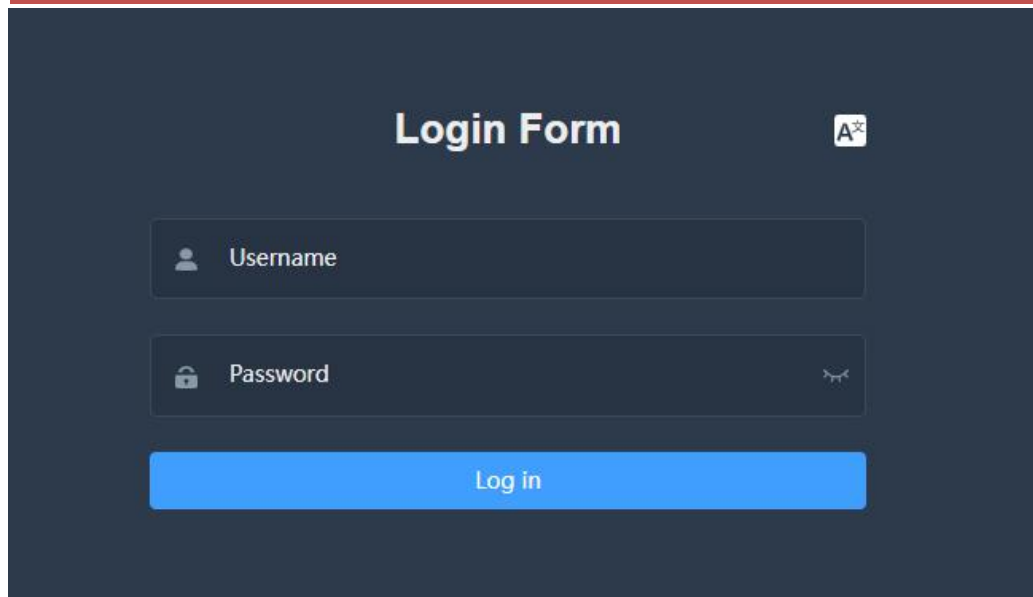




## Apache Pulsar的可视化监控部署

- 第六步: 访问Pulsar UI

<http://node1:7750/ui/index.html>



The screenshot shows the Apache Pulsar UI Login Form. It has a dark blue background. At the top, it says "Login Form" in white. Below that, there are two input fields: "Username" and "Password". The "Username" field has a user icon on the left. The "Password" field has a lock icon on the left and a toggle icon on the right. Below the input fields is a blue "Log in" button. In the top right corner of the form, there is a small icon for accessibility (A<sup>xx</sup>).

用户名: pulsar

密码: pulsar



## Apache Pulsar的可视化监控基本使用

- 第一步: 点击 new Enirconment 构建新环境,连接pulsar

**+ New Environment** 构建新环境

Environment Name	Service URL	Actions
No Data		

New Environment

\* Environment Name

pulsar\_cluster 1

\* Service URL

http://192.168.88.161:8080,192.168.88.162:8080,192.168.88.163:8080 2

3

Confirm

Cancel

## Apache Pulsar的可视化监控基本使用

- 第二步: 点击 pulsar\_cluster,进入管理界面

Environment Name	Service URL	Actions
pulsar_cluster	http://192.168.88.161:8080,192.168.88.162:8080,192.168.88.163:8080	<a href="#">Edit</a> <a href="#">Delete</a>

点击

Management

Clusters 集群信息

Tenants 租户信息

Namespaces 名称空间

Topics 话题

Tokens 角色

Apache Pulsar

Management

Search Tenants

+ New Tenant

Tenant	Namespaces	Allowed Clusters	Admin Roles	In Rate	Out Rate	In Throughput	Out Throughput	Storage Size	Actions
my-tenant	1	pulsar-cluster-1 pulsar-cluster		± 0.00	± 0.00	± 0	± 0	0.00	<a href="#">Edit</a> <a href="#">Delete</a>
public	1	pulsar-cluster-1 pulsar-cluster		± 0.00	± 0.00	± 0	± 0	0.00	<a href="#">Edit</a> <a href="#">Delete</a>
pulsar	1	pulsar-cluster-1 pulsar-cluster		± 0.00	± 0.00	± 0 Bytes	± 0 Bytes	0 Bytes	<a href="#">Edit</a> <a href="#">Delete</a>

pulsar\_cluster

Admin

## 04

# Apache Pulsar主要功能介绍及使用

- 多租户模式
- Pulsar的名称空间
- Pulsar的topic相关操作

## 什么是多租户

Apache Pulsar 最初诞生于雅虎，当时就是为了解决雅虎内部各个部门之间数据的协调，所以多租户特性显得至关重要，Pulsar 从诞生之日起就考虑到多租户这一特性，并在后续的实现过程中，将其不断的完善。多租户这一特性，使得各个部门之间可以共享同一份数据，不用单独部署独立的系统来操作数据，很好的保证了各部门间数据一致性的问题，同时简化维护成本。

在介绍 Pulsar 多租户之前，先来看一下，正常一个系统要实现一个多租户需要做哪些事情：

- 严格的 SLAs 保证
- 确保租户之间的隔离性
- 允许对租户内的资源进行配额
- 在租户内提供系统级别的安全性
- 运维成本低，易管理

Pulsar 的多租户设计符合上述要求：

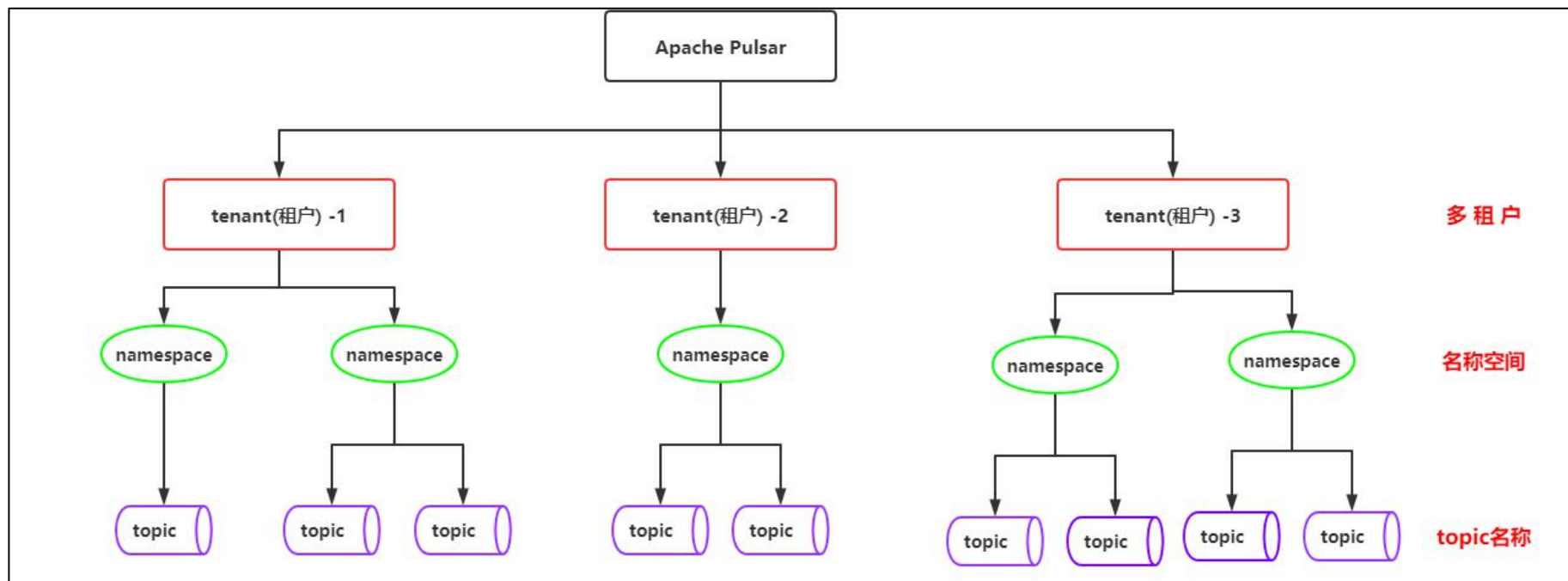
- 使用身份验证、授权和 ACL（访问控制列表）确保其安全性
- 为每个租户强制执行存储配额
- 支持在运行时更改隔离机制，从而实现操作成本低和管理简单

## 什么是多租户

Pulsar的多租户性质主要体现在topic的URL中, 其结构如下:

`persistent://tenant/namespace/topic`

从URL中可以看出tenant(租户)是topic最基本的单元(比命名空间和topic名称更为基本)



## Pulsar多租户的相关特性\_安全性(认证和授权)

一个多租户系统需要在租户内提供系统级别的安全性，细分来讲，主要可以归类为一下两点：

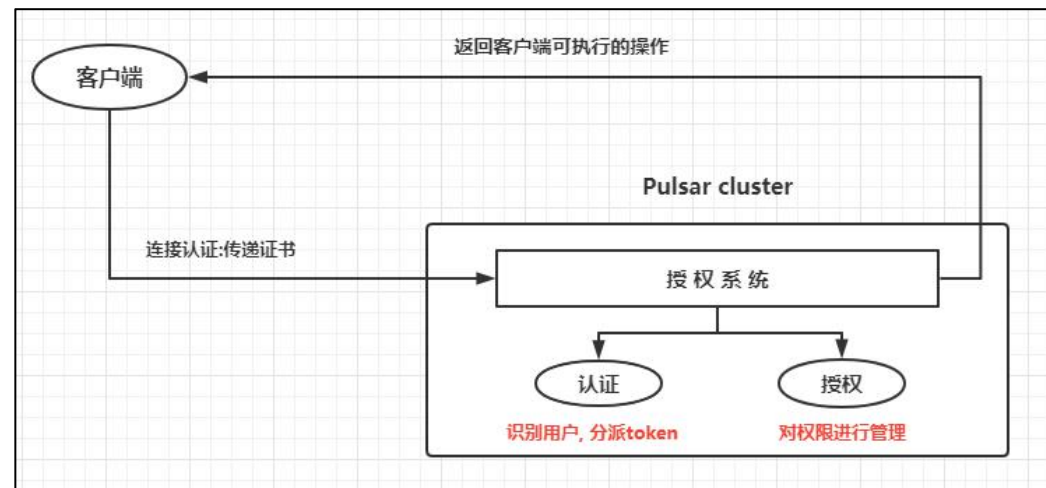
- 租户只能访问它有权访问的 topics
- 不允许访问它无法访问的 topics

在 Pulsar 中，多租户的安全性是通过身份验证和授权机制实现的。当 client 连接到 pulsar broker 时，broker 会使用身份验证插件来验证此客户端的身份，然后为其分配一个 string 类型的 role token。role token 主要有如下作用：

- 判断 client 是否有对 topics 进行生产或消费消息的权限
- 管理租户属性的配置

Pulsar 目前支持一下几种身份认证, 同时支持自定义实现自己的身份认证程序

- TLS 客户端身份认证
- 雅虎的身份认证系统: Athenz
- Kerberos
- JSON Web Token 认证



## Pulsar多租户的相关特性\_隔离性

隔离性主要分为如下两种：

- 软隔离: 通过磁盘配额，流量控制和限制等手段

### 存储:

Apache Pulsar 使用Bookkeeper来作为其存储层, bookie是Bookkeeper的实例, Bookkeeper本身就是具有I/O分离(读写分离)的特性,可以很多的做好IO隔离, 提升读写的效率

同时, 不同的租户可以为不同的NameSpace配置不同的存储配额, 当租户内消息的大小达到了存储配额的限制, Pulsar会采取相应的措施, 例如: 阻止消息生成, 抛异常 或丢弃数据等

### Broker:

每个Broker使用的内存资源都是有上限的, 当Broker达到配置的CPU或内存使用的阈值后, Pulsar会迅速的将流量转移到负载较小的Broker处理

在生产 and 消费方面, Pulsar都可以进行流量控制, 租户可以配置发送和接收的速率, 避免出现一个客户端占用当前Broker的所有处理资源

- 硬隔离: 物理资源隔离

Pulsar 允许将某些租户或名称空间与特定 Broker 进行隔离。这可确保这些租户或命名空间可以充分利用该特定 Broker 上的资源。

## Pulsar多租户的相关操作

- 1 - 获取租户列表

```
cd /export/server/brokers/bin  
./pulsar-admin tenants list
```

```
[root@hd1 bin]# ./pulsar-admin tenants list  
"my-tenant"  
"public"  
"pulsar"
```

- 2 - 创建租户

```
cd /export/server/brokers/bin  
./pulsar-admin tenants create my-tenant
```

在创建租户时，可以使用-r 或者 --admin-roles标志分配管理角色。可以用逗号分隔的列表指定多个角色。

```
./pulsar-admin tenants create my-tenant \  
--admin-roles role1,role2,role3
```

```
./pulsar-admin tenants create my-tenant \  
-r role1
```



## Pulsar多租户的相关操作

- 3 - 获取配置:

```
pulsar-admin tenants get my-tenant
{
  "adminRoles": [
    "admin1",
    "admin2"
  ],
  "allowedClusters": [
    "cl1",
    "cl2"
  ]
}
```

## Pulsar多租户的相关操作

- 4 - 更新配置: 注意: 在删除的时候, 如果库下已经有名称空间, 是无法删除的, 需要先删除名称空间

```
cd /export/server/brokers/bin  
./pulsar-admin tenants update my-tenant
```

基于update可以更新租户的相关配置信息

- 5 - 删除租户: 注意: 在删除的时候, 如果库下已经有名称空间, 是无法删除的, 需要先删除名称空间

```
cd /export/server/brokers/bin  
./pulsar-admin tenants delete my-tenant
```

```
[hadoop@hd1 bin]$ ./pulsar-admin tenants delete my-tenant  
18:51:23.494 [AsyncHttpClient-7-1] WARN org.apache.pulsar.client.admin.internal.BaseResource - [http://localhost:8080/admin/v2/  
tenants/my-tenant] Failed to perform http delete request: javax.ws.rs.ClientErrorException: HTTP 409 Conflict  
The tenant still has active namespaces  
  
Reason: The tenant still has active namespaces
```

## 04

# Apache Pulsar主要功能介绍及使用

- 多租户模式
- Pulsar的名称空间
- Pulsar的topic相关操作

## 什么是名称空间

namespace是Pulsar中最基本的管理单元，在namespace这一层面，可以设置权限，调整副本设置，管理跨集群的消息复制，控制消息策略和执行关键操作。一个主题topic可以继承其所对应的namespace的属性，因此我们只需对namespace的属性进行设置，就可以一次性设置该namespace中所有主题topic的属性。

namespace有两种，分别是本地的namespace和全局的namespace：

- 本地namespace——仅对定义它的集群可见。
- 全局namespace——跨集群可见，可以是同一个数据中心的集群，也可以是跨地域中心的集群，这依赖于是否在namespace中设置了跨集群拷贝数据的功能。

虽然本地namespace和全局namespace的作用域不同，但是只要对他们进行适当的设置，都可以跨团队和跨组织共享。一旦生产者获得了namespace的写入权限，那么它就可以往namespace中的所有topic主题写入数据，如果某个主题不存在，则在生产者第一次写入数据时动态创建。

## Pulsar NameSpace(名称空间) 相关操作\_基础操作

- 1 - 在指定的租户下创建名称空间

```
cd /export/server/brokers/bin  
./pulsar-admin namespaces create test-tenant/test-namespcae
```

- 2 - 获取所有的名称空间列表

```
cd /export/server/brokers/bin  
./pulsar-admin namespaces list test-tenant
```

- 3 - 删除名称空间

```
cd /export/server/brokers/bin  
./pulsar-admin namespaces delete test-tenant/ns1
```

## Pulsar NameSpace(名称空间) 相关操作\_高级操作

- 1 - 获取名称空间相关的配置策略

```
cd /export/server/brokers/bin  
./pulsar-admin namespaces policies test-tenant/test-namespace
```

```
{  
  "auth_policies": {  
    "namespace_auth": {},  
    "destination_auth": {}  
  },  
  "replication_clusters": [],  
  "bundles_activated": true,  
  "bundles": {  
    "boundaries": [  
      "0x00000000",  
      "0xffffffff"  
    ],  
    "numBundles": 1  
  },  
  "backlog_quota_map": {},  
  "persistence": null,  
  "latency_stats_sample_rate": {},  
  "message_ttl_in_seconds": 0,  
  "retention_policies": null,  
  "deleted": false  
}
```

## Pulsar NameSpace(名称空间) 相关操作\_高级操作

- 2 - 配置复制集群

### 2.1- 设置复制集群:

```
cd /export/server/brokers/bin  
pulsar-admin namespaces set-clusters test-tenant/ns1 --clusters cl2
```

### 2.2- 获取给定命名空间复制集群的列表

```
pulsar-admin namespaces get-clusters test-tenant/ns1
```

- 3 - 配置 backlog quota 策略

待定配额帮助Broker在某个名称空间达到某个阈值限制时限制其带宽/存储。管理员可以设置限制，并在达到限制后采取相应的行动。

### 3.1- 设置backlog quota 策略

```
cd /export/server/brokers/bin  
pulsar-admin namespaces set-backlog-quota --limit 10G --limitTime 36000 --policy producer_request_hold test-tenant/ns1
```

#### --policy 的值选择:

- producer\_request\_hold: broker 暂停运行，并不再持久化生产请求负载
- producer\_exception: broker 抛出异常，并与客户端断开连接。
- consumer\_backlog\_eviction: broker 丢弃积压消息

## Pulsar Namespace(名称空间) 相关操作\_高级操作

### 3.2- 获取 backlog quota 策略

```
pulsar-admin namespaces get-backlog-quotas test-tenant/ns1
{
  "destination_storage": {
    "limit": 10,
    "policy": "producer_request_hold"
  }
}
```

### 3.3 - 移除backlog quota 策略

```
pulsar-admin namespaces remove-backlog-quota test-tenant/ns1
```

- 4 - 配置持久化策略

持久化策略可以为给定命名空间下 topic 上的所有消息配置持久等级。

### 4.1- 设置持久化策略

```
pulsar-admin namespaces set-persistence --bookkeeper-ack-quorum 2 --bookkeeper-ensemble 3 --bookkeeper-write-quorum 2 --
ml-mark-delete-max-rate 0 test-tenant/ns1
```

#### 参数说明:

Bookkeeper-ack-quorum: 每个 entry 在等待的 acks (有保证的副本) 数量, 默认值: 0

Bookkeeper-ensemble: 单个 topic 使用的 bookie 数量, 默认值: 0

Bookkeeper-write-quorum: 每个 entry 要写入的次数, 默认值: 0

ML-mark-delete-max-rate: 标记-删除操作的限制速率 (0表示无限制), 默认值: 0.0



## Pulsar NameSpace(名称空间) 相关操作\_高级操作

### 4.2- 获取持久化策略

```
pulsar-admin namespaces get-persistence test-tenant/ns1
{
  "bookkeeperEnsemble": 3,
  "bookkeeperWriteQuorum": 2,
  "bookkeeperAckQuorum": 2,
  "managedLedgerMaxMarkDeleteRate": 0
}
```

- 5 - 配置消息存活时间(TTL)

以秒为单位

### 5.1- 设置消息存活时间

```
pulsar-admin namespaces set-message-ttl --messageTTL 100 test-tenant/ns1
```

### 5.2- 获取消息的存活时间

```
pulsar-admin namespaces get-message-ttl test-tenant/ns1
```

### 5.3- 删除消息的存活时间

```
pulsar-admin namespaces remove-message-ttl test-tenant/ns1
```

## Pulsar NameSpace(名称空间) 相关操作\_高级操作

- 5 - 配置整个名称空间中Topic的消息发送速率

### 5.1- 设置Topic的消息发送的速率

```
pulsar-admin namespaces set-dispatch-rate test-tenant/ns1 \  
--msg-dispatch-rate 1000 \  
--byte-dispatch-rate 1048576 \  
--dispatch-rate-period 1
```

#### 参数说明:

--msg-dispatch-rate : 每dispatch-rate-period秒钟发送的消息数量  
--byte-dispatch-rate : 每dispatch-rate-period秒钟发送的总字节数  
--dispatch-rate-period : 设置发送的速率, 比如 1 表示 每秒钟

### 5.2 获取topic的消息发送速率

```
pulsar-admin namespaces get-dispatch-rate test-tenant/ns1  
{  
  "dispatchThrottlingRatePerTopicInMsg" : 1000,  
  "dispatchThrottlingRatePerTopicInByte" : 1048576,  
  "ratePeriodInSecond" : 1  
}
```

## Pulsar NameSpace(名称空间) 相关操作\_高级操作

- 6 - 配置整个名称空间中Topic的消息接收速率

### 6.1- 设置Topic的消息接收的速率

```
pulsar-admin namespaces set-subscription-dispatch-rate test-tenant/ns1 \  
--msg-dispatch-rate 1000 \  
--byte-dispatch-rate 1048576 \  
--dispatch-rate-period
```

#### 参数说明:

--msg-dispatch-rate : 每dispatch-rate-period秒钟接收的消息数量  
--byte-dispatch-rate : 每dispatch-rate-period秒钟接收的总字节数  
--dispatch-rate-period : 设置接收的速率, 比如 1 表示 每秒钟

### 6.2 获取topic的消息接收速率

```
pulsar-admin namespaces get-subscription-dispatch-rate test-tenant/ns1  
{  
  "dispatchThrottlingRatePerTopicInMsg" : 1000,  
  "dispatchThrottlingRatePerTopicInByte" : 1048576,  
  "ratePeriodInSecond" : 1  
}
```

## Pulsar NameSpace(名称空间) 相关操作\_高级操作

- 7 - 配置整个名称空间中Topic的复制集群的速率

### 7.1- 设置Topic的消息复制集群的速率

```
pulsar-admin namespaces set-replicator-dispatch-rate test-tenant/ns1 \  
--msg-dispatch-rate 1000 \  
--byte-dispatch-rate 1048576 \  
--dispatch-rate-period 1
```

#### 参数说明:

--msg-dispatch-rate : 每dispatch-rate-period秒钟复制集群的消息数量  
--byte-dispatch-rate : 每dispatch-rate-period秒钟复制集群的总字节数  
--dispatch-rate-period : 设置复制集群的速率, 比如 1 表示 每秒钟

### 7.2 获取topic的消息复制集群的速率

```
pulsar-admin namespaces get-replicator-dispatch-rate test-tenant/ns1  
{  
  "dispatchThrottlingRatePerTopicInMsg" : 1000,  
  "dispatchThrottlingRatePerTopicInByte" : 1048576,  
  "ratePeriodInSecond" : 1  
}
```

## 04

# Apache Pulsar主要功能介绍及使用

- 多租户模式
- Pulsar的名称空间
- Pulsar的topic相关操作

## 什么是Topic

Topic,话题主题的含义, 在一个名称空间下, 可以定义多个Topic 通过Topic进行数据的分类划分, 将不同的类别的消息放置到不同Topic, 消费者也可以从不同Topic中获取到相关的消息, 是一种更细粒度的消息划分操作, 同时在Topic下可以划分为多个分片, 进行分布式的存储操作, 每个分片下还存在有副本操作, 保证数据不丢失, 当然这些分片副本更多是由bookkeeper来提供支持

Pulsar 提供持久化与非持久化两种topic。持久化topic是消息发布、消费的逻辑端点。持久化topic地址的命名格式如下:

```
persistent://tenant/namespace/topic
```

非持久topic应用在仅消费实时发布消息与不需要持久化保证的应用程序。通过这种方式, 它通过删除持久消息的开销来减少消息发布延迟。非持久化topic地址的命名格式如下:

```
non-persistent://tenant/namespace/topic
```

## Pulsar Topic(主题) 相关操作\_基础操作

- 1 - 创建Topic

方式一: 创建一个没有分区的topic

```
bin/pulsar-admin topics create persistent://my-tenant/my-namespace/my-topic
```

方式二: 创建一个有分区的topic

```
bin/pulsar-admin topics create-partitioned-topic persistent://my-tenant/my-namespace/my-topic --partitions 4
```

注意: 不管是有分区还是没有分区, 创建topic后, 如果没有任何操作, 60s后pulsar会认为此topic是不活动的, 会自动进行删除, 以避免生成垃圾数据

相关配置:

Brokerdeleteinactivetopicsenabled : 默认值为true 表示是否启动自动删除

BrokerDeleteInactiveTopicsFrequencySeconds: 默认为60s 表示检测未活动的时间

- 2 - 列出当前某个名称空间下的所有Topic

```
./pulsar-admin topics list my-tenant/my-namespace
```

## Pulsar Topic(主题) 相关操作\_基础操作

- 3 - 更新Topic操作

我们可针对有分区的topic去更新其分区的数量

```
./pulsar-admin topics update-partitioned-topic persistent://my-tenant/my-namespace/my-topic --partitions 8
```

- 4 - 删除Topic操作

删除没有分区的topic:

```
bin/pulsar-admin topics delete persistent://my-tenant/my-namespace/my-topic
```

删除有分区的topic

```
bin/pulsar-admin topics delete-partitioned-topic persistent://my-tenant/my-namespace/my-topic
```



## Pulsar Topic(主题) 相关操作\_高级操作

- 1 - 授权

```
pulsar-admin topics grant-permission --actions produce,consume --role application1 persistent://test-tenant/ns1/tp1
```

- 2- 获取权限

```
pulsar-admin topics grant-permission --actions produce,consume --role application1 persistent://test-tenant/ns1/tp1
```

- 3- 取消权限

```
pulsar-admin topics revoke-permission --role application1 persistent://test-tenant/ns1/tp1

{
  "application1": [
    "consume",
    "produce"
  ]
}
```



# 05

## Apache Pulsar基于Java Api基本使用

- 基于Pulsar实现Topic的构建操作
- 基于Pulsar实现数据生产
- 基于Pulsar实现数据消费

## 基于Pulsar实现Topic的构建操作\_准备工作

- 首先, 需要我们创建一个maven项目, 并加入Pulsar相关的依赖

```
<repositories><!--代码库-->
  <repository>
    <id>aliyun</id>
    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
    <releases><enabled>true</enabled></releases>
    <snapshots>
      <enabled>false</enabled>
      <updatePolicy>never</updatePolicy>
    </snapshots>
  </repository>
</repositories>

<dependencies>
  <dependency>
    <groupId>org.apache.pulsar</groupId>
    <artifactId>pulsar-client-all</artifactId>
    <version>2.8.1</version>
  </dependency>
</dependencies>
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <target>1.8</target>
        <source>1.8</source>
      </configuration>
    </plugin>
  </plugins>
</build>
```

## 基于Pulsar实现Topic的构建操作

- 1- 使用JAVA如何管理租户

```
package com.itheima.admin;

import org.apache.pulsar.client.admin.PulsarAdmin;
import org.apache.pulsar.common.policies.data.TenantInfo;
import java.util.HashSet;
import java.util.List;

// 演示如何使用Java构建租户
public class _01_CreateTenants {
    public static void main(String[] args) throws Exception {
        // 1- 创建Pulsar的Admin管理对象
        String serviceUrl = "http://node1:8080,node2:8080,node3:8080";
        PulsarAdmin admin = PulsarAdmin.builder()
            .serviceHttpUrl(serviceUrl)
            .build();
        // 2- 创建租户
        HashSet<String> clusters = new HashSet<>();
        clusters.add("pulsar-cluster");

        HashSet<String> adminRoles = new HashSet<>();
        adminRoles.add("dev");
        TenantInfo config = TenantInfo.builder()
            .allowedClusters(clusters)
            .adminRoles(adminRoles)
            .build();
        admin.tenants().createTenant("itcast_pulsar_t",config);
```

```
// 3- 获取有那些租户
System.out.println("获取有那些租户");
List<String> tenants = admin.tenants().getTenants();
for (String tenant : tenants) {
    System.out.println(tenant);
}

// 4. 删除租户操作
// 参数1: 表示租户的名称 参数2: 是否强制删除
admin.tenants().deleteTenant("itcast_pulsar_t");

// 5. 在重新获取租户列表
System.out.println("删除后,获取有那些租户");
tenants = admin.tenants().getTenants();
for (String tenant : tenants) {
    System.out.println(tenant);
}

// 6. 关闭资源
admin.close();
}
```

## 基于Pulsar实现Topic的构建操作

- 2- 使用JAVA如何管理namespace

```
package com.itheima.admin;

import org.apache.pulsar.client.admin.PulsarAdmin;
import org.apache.pulsar.common.policies.data.TenantInfo;

import java.util.ArrayDeque;
import java.util.HashSet;
import java.util.List;

// 演示如何使用Java管理namespace
public class _02_CreateNamespaces {

    public static void main(String[] args) throws Exception {

        // 1- 创建Pulsar的Admin管理对象
        String serviceUrl = "http://node1:8080,node2:8080,node3:8080";
        PulsarAdmin admin = PulsarAdmin.builder()
            .serviceHttpUrl(serviceUrl)
            .build();

        // 2- 创建名称空间
        admin.namespaces().createNamespace("itcast_pulsar_t/itcast_pulsar_n");
```

```
// 3- 获取所有的名称空间
System.out.println("获取当前有那些名称空间:");
List<String> namespaces =
admin.namespaces().getNamespaces("itcast_pulsar_t");
for (String namespace : namespaces) {
    System.out.println(namespace);
}

// 4- 删除名称空间

admin.namespaces().deleteNamespace("itcast_pulsar_t/itcast_pulsar_
n");

// 6. 关闭资源
admin.close();
}
}
```

## 基于Pulsar实现Topic的构建操作

### ● 3- 使用JAVA如何管理Topic

```
package com.itheima.admin;

import org.apache.pulsar.client.admin.PulsarAdmin;
import java.util.List;
// 演示如何使用Java管理Topic
public class _03_CreateTopic {
    public static void main(String[] args) throws Exception {
        // 1- 创建Pulsar的Admin管理对象
        String serviceUrl = "http://node1:8080,node2:8080,node3:8080";
        PulsarAdmin admin = PulsarAdmin.builder()
            .serviceHttpUrl(serviceUrl)
            .build();
        //2. 创建Topic
        // 2.1: 创建一个持久化的带分区的Topic的
        admin.topics().createPartitionedTopic(
            "persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic1",3);
        // 2.2: 创建一个非持久化的带分区的Topic的
        admin.topics().createPartitionedTopic(
            "non-persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic2",3);
        // 2.3: 创建一个持久化的带分区的Topic的
        admin.topics().createNonPartitionedTopic(
            "persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3");
        // 2.4: 创建一个非持久化的带分区的Topic的
        admin.topics().createNonPartitionedTopic(
            "non-persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic4");
```

```
// 3. 列出某个名称空间下, 所有的Topic
// 无分区的topic
List<String> topics = admin.topics().getList("itcast_pulsar_t/itcast_pulsar_n");
for (String topic : topics) {
    System.out.println(topic);
}
// 有分区的topic
topics = admin.topics().getPartitionedTopicList("itcast_pulsar_t/itcast_pulsar_n");
for (String topic : topics) {
    System.out.println(topic);
}
// 4. 更新Topic: 增加分区数
admin.topics().updatePartitionedTopic(
    "persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic1",5);
int partitions = admin.topics().getPartitionedTopicMetadata(
    "persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic1").partitions;
System.out.println("topic的分区数为:"+partitions);
// 5. 删除Topic
// 删除没有分区的
admin.topics().delete("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3");
// 删除有分区的
admin.topics().deletePartitionedTopic(
    "non-persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic2");

// 6. 关闭资源
admin.close();
}
```

05

## Apache Pulsar基于Java Api基本使用

- 基于Pulsar实现Topic的构建操作
- 基于Pulsar实现数据生产
- 基于Pulsar实现数据消费

## 基于Pulsar实现数据生产

- 1- 使用JAVA如何生产数据\_同步方式

```
// 模拟pulsar的生产者_同步模式
public class PulsarProducerSyncTest {

    public static void main(String[] args) throws Exception {
        //1. 获取pulsar的客户端对象
        ClientBuilder clientBuilder = PulsarClient.builder();
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");
        PulsarClient client = clientBuilder.build();
        //2. 通过客户端创建生产者的对象
        Producer<byte[]> producer = client.newProducer()
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3")
            .create();
        //3. 发送消息:
        producer.send("你好 Pulsar...".getBytes());
        //4. 释放资源
        producer.close();
        client.close();
    }
}
```



## 基于Pulsar实现数据生产

- 2- 使用JAVA如何生产数据\_异步方式

```
// 模拟pulsar的生产者_异步模式
public class PulsarProducerAsyncTest {
    public static void main(String[] args) throws Exception {
        //1. 获取pulsar的客户端对象
        ClientBuilder clientBuilder = PulsarClient.builder();
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");
        PulsarClient client = clientBuilder.build();
        //2. 通过客户端创建生产者的对象
        Producer<byte[]> producer = client.newProducer()
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3")
            .create();
        //3. 发送消息:
        producer.sendAsync("你好 Pulsar...".getBytes());
        // 如果采用异步发送数据, 由于需要先放置在缓存区中, 如果立即关闭, 会导致
        // 无法发送
        Thread.sleep(1000);
        //4. 释放资源
        producer.close();
        client.close();
    }
}
```

## 基于Pulsar实现数据生产

- 3- 使用JAVA如何生产数据\_基于schema的发送

```
public class PulsarProducerSchemaTest {  
    public static void main(String[] args) throws Exception {  
        //1. 获取pulsar的客户端对象  
        ClientBuilder clientBuilder = PulsarClient.builder();  
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");  
        PulsarClient client = clientBuilder.build();  
        //2. 通过客户端创建生产者的对象  
        AvroSchema<User2> schema =  
        AvroSchema.of(SchemaDefinition.<User2>builder().withPojo(User2.class).build());  
        Producer<User2> producer = client.newProducer(schema)  
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3").create();  
        //3. 发送消息:  
        User2 user = new User2();  
        user.setName("张三");  
        user.setAge(20);  
        user.setAddress("北京");  
        user.setRowkey("rk001");  
        user.setFamilyName("C1");  
        producer.send(user);  
        Thread.sleep(10000);  
        //4. 释放资源  
        producer.close();  
        client.close();  
    }  
}
```



# 05

## Apache Pulsar基于Java Api基本使用

- 基于Pulsar实现Topic的构建操作
- 基于Pulsar实现数据生产
- 基于Pulsar实现数据消费

## 基于Pulsar实现数据生产

- 1- 使用JAVA如何消费数据\_同步方式

```
//模拟pulsar的消费者
public class PulsarConsumerTest {
    public static void main(String[] args) throws Exception {
        //1. 获取pulsar的客户端对象
        ClientBuilder clientBuilder = PulsarClient.builder();
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");
        PulsarClient client = clientBuilder.build();
        //2. 通过客户端创建消费者的对象
        Consumer<byte[]> consumer = client.newConsumer()
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3")
            .subscriptionName("my-subscription")
            .subscribe();
        //3. 循环获取读取
        while(true) {
            Message<byte[]> message = consumer.receive();
            try {
                System.out.println("消息为:" + new String(message.getData()));
                consumer.acknowledge(message);
            } catch (Exception e) {
                consumer.negativeAcknowledge(message);
            }
        }
    }
}
```

## 基于Pulsar实现数据生产

- 2- 使用JAVA如何消费数据\_schema方式

```
public class PulsarConsumerSchemaTest {  
    public static void main(String[] args) throws Exception{  
        //1. 获取pulsar的客户端对象  
        ClientBuilder clientBuilder = PulsarClient.builder();  
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");  
        PulsarClient client = clientBuilder.build();  
        //2. 通过客户端创建消费者的对象  
        Consumer<User2> consumer =  
        client.newConsumer(AvroSchema.of(SchemaDefinition.<User2>builder().withPojo(User2.class).build()))  
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3")  
            .subscriptionName("my-subscription")  
            .subscribe();  
        //3. 循环获取读取  
        while(true) {  
            Message<User2> message = consumer.receive();  
            try {  
                System.out.println("消息为:" +message.getValue());  
                consumer.acknowledge(message);  
            }catch ( Exception e) {  
                consumer.negativeAcknowledge(message);  
            }  
        }  
    }  
}
```

## 基于Pulsar实现数据生产

- 3- 使用JAVA如何消费数据\_批量处理

```
//模拟pulsar的消费者_同步模式
public class PulsarConsumerBatchTest {
    public static void main(String[] args) throws Exception {
        //1. 获取pulsar的客户端对象
        ClientBuilder clientBuilder = PulsarClient.builder();
        clientBuilder.serviceUrl("pulsar://node1:6650,node2:6650,node3:6650");
        PulsarClient client = clientBuilder.build();
        //2. 通过客户端创建消费者的对象
        Consumer<byte[]> consumer = client.newConsumer()
            .topic("persistent://itcast_pulsar_t/itcast_pulsar_n/my-topic3")
            .subscriptionName("my-subscription")
            .batchReceivePolicy(BatchReceivePolicy.builder()
                // 设置一次性最大获取多少条消息 默认值为 -1
                .maxNumMessages(100)
                // 设置每条数据允许的最大的字节大小 默认值为: 10 * 1024 * 1024
                .maxNumBytes(1024 * 1024)
                //设置等待的超时时间 默认值为 100
                .timeout(200, TimeUnit.MILLISECONDS)
                .build())
            .subscribe();
```

```
//3. 循环获取读取
while(true) {
    Messages<byte[]> messages = consumer.batchReceive(); // 批量读
    取数据
    for (Message<byte[]> message : messages) {
        try {
            System.out.println("消息为:" + new String(message.getData()));

            consumer.acknowledge(message);
        } catch (Exception e) {
            consumer.negativeAcknowledge(message);
        }
    }
}
}
```



传智教育旗下高端IT教育品牌