使用Docker-compose搭建kafka-kraft集群

Kafka入门资料及参考收集汇总

# Kafka入门基本概念介绍及docker-compose部署测试集群

## 概述

****Apache Kafka 是一款开源分布式流处理平台。可以用来发布和订阅数据以及对数据进行实时或者离线处理。****

****主要特点有：****

****高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作；****

****可扩展性：kafka集群支持热扩展；****

****持久性、可靠性：消息被持久化到本地磁盘，并且支持[数据备份](https://cloud.tencent.com/solution/backup?from_column=20065&from=20065" \t "https://cloud.tencent.com/developer/article/_blank)防止数据丢失；****

****容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）；****

****高并发：支持数千个客户端同时读写；****

****支持实时在线处理和离线处理：可以使用Storm这种实时流处理系统对消息进行实时进行处理，同时还可以使用Hadoop这种批处理系统进行离线处理；****

****多客户端支持：java、golang等****

****主要应用场景****

1. ****消息系统:常规的消息队列中间件，实现异步解耦、削峰等功能****
2. ****日志收集：Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如Hadoop、Hbase、Solr等；****
3. ****Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者装载到Hadoop、数据仓库中做离线分析和挖掘；****
4. ****运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告；****
5. ****流式处理：比如spark streaming和storm；****
6. ****事件源。****

****Kafka详细简介参考连接：**<https://cloud.tencent.com/developer/article/1815338>**

****Kafka官方文档：**<https://kafka.apachecn.org/intro.html>**

****Kafka与其他消息队列对比参考连接：****

**<https://www.cnblogs.com/snow-man/p/10062394.html>**

****https://blog.csdn.net/u013521220/article/details/104352365****

****Kafka基本概念：****

****Broker（代理）****

****每个 Broker 即一个 Kafka 服务实例，多个 Broker 构成一个 Kafka 集群，生产者发布的消息将保存在 Broker 中，消费者将从 Broker 中拉取消息进行消费。****

****Broker集群中，会有一个leader（controller leader），负责管理整个集群中分区和副本的状态和选举partition leader****

****Kafka组件包括Kafka服务示例、producer生产者、consumer消费者。****

****Kafka服务主要接收生产者生产的消息，并经过存储、转换等一系列操作后，为消费者返回消息。****

****Kafka服务可支持单点部署和集群部署。****

****根据kafka版本的不同，在2.8以前必须使用zookeeper作为集群的管理者，用于管理集群配置、选举leader、负载均衡等。****

****由于在使用kafka时必须要多引入一个中间件，增加了部署及运维的难度，在2.8及以后的版本，kafka新增了Kraft模式，在这个模式下，kafka服务可以不用借助zookeeper进行集群管理及负载均衡，而是可以自我管理、独立部署，降低了部署难度及由于zookeeper自身限制带来的对kafka的性能影响及限制。****

****区别：****

****使用zookeeper管理：****

****Kafka服务分为leader（由zookeeper进行选举，无法指定）及broker。****

****Kraft中：****

****Kafka服务器角色分为controller和broker。Controller可以配置文件设置（根据负载情况，可以灵活指定机器作为controller），单个节点既可以作为controller也可以作为broker。在集群中通过raft共识协议（Raft共识协议是一种基于领导者选举和日志复制的分布式一致性算法，它能够保证集群中的多个节点在状态和数据上达成一致。）决定controller节点。****

****Topic：****

****Kafka中将消息分类，每一类消息称为一个topic，生产者通过指定topic将消息发送到broker中，消费者通过指定topic可以针对不同的topic进行消息消费。Topic有点类似于数据库的表。****

****Partition：****

****一个Topic可以分为多个partition，每个partition是一个有序的队列，在partition中每条消息都存在一个有序的偏移量代表这条消息在partition中的位置。同一个consumer group中，只有一个consumer实例可以消费某个partition的消息。一个topic可以有多个partition，多个partiton可以并行处理数据。只有partition的leader才会进行读写操作，folower仅进行复制，客户端无法感知。****

****Replica：****

****为了保证数据安全，partition多个副本，至少会有一个leader副本和多个follower副本，leader负责处理客户端的读写请求，follower副本只复制leader副本的数据，当leader宕机时，follower会自动接替leader副本的工作，从而保证数据的可用性。****

****Isr：****

****分区中的所有副本统称为AR（Assigned Replicas）。所有与leader副本保持一定程度同步的副本（包括leader）组成ISR（in-sync replicas）。而与leader副本同步滞后过多的副本（不包括leader），组成OSR（out-sync replicas），所以，AR = ISR + OSR。在正常情况下，所有的follower副本都应该与leader副本保持一定程度的同步，即AR = ISR，OSR集合为空。****

****leader会追踪和维护ISR中所有follower的滞后状态。如果滞后太多（时间滞后replica.lag.time.max.ms可配置），leader会把该replica从ISR中移除。被移除ISR的replica一直在追赶leader。如下图，leader写入数据后并不会commit，只有ISR列表中的所有folower同步之后才会commit，把滞后的follower移除ISR主要是避免写消息延迟。设置ISR主要是为了broker宕掉之后，重新选举partition的leader从ISR列表中选择。****

****两种leader：****

****Broker的leader：在Kafka中，Controller是一个特殊的Broker，它负责管理整个Kafka集群的元数据，包括Topic、Partition、Broker等信息。在KRaft模式下，只有一小组特别选定的服务器可以充当Controller，这些Controller服务器将参与元数据的竞选，对于已经处于活跃状态的Controller，其他Controller将充当热备份角色。因此，Kafka集群中的Controller是非常重要的，它负责管理整个集群的状态，确保集群的正常运行。同时，Controller还负责管理Broker的注册和心跳，以及处理Broker的加入和退出等操作。在KRaft模式下，Controller的选举和切换都是自动完成的，无需人工干预。****

****Partition的leader：处理客户端对partition的读写请求。****

****Producer：生产者****

****负责生产消息，并发送到broker。在发送时，通常指定topic和partition。****

****Consumer消费者：****

****负责消费broker中topic消息，每个consumer实例归属于一个consumer group****

****参考：**<https://blog.csdn.net/u014609111/article/details/116199553>**

****Docker-compose部署kafka集群****

****Docker版本：24.0.4****

****docker-compose版本：2.20.2****

****Kafka镜像版本：bitnami/kafka:3.6****

1. ****创建kafka集群专用网络****

****docker network create kafka-net****

1. ****编写docker-compose.yaml文件****

****参数解析;****

****KAFKA\_ENABLE\_KRAFT:** 使用Kraft模式**

****KAFKA\_CFG\_PROCESS\_ROLES：**指定服务器在KRaft模式下的角色。broker（处理客户端请求及管理分区副本）、controller（只管理集群及协调选举）。两个角色可以同时兼任，**

**KAFKA\_CFG\_CONTROLLER\_QUORUM\_VOTERS：在raft模式下，参与选举和和决策的控制器节点 brockerID@hostname:port 格式，多个以，分割。**

****KAFKA\_CFG\_LISTENERS：**指定了 Kafka 绑定的监听器列表和主机/IP和端口。监听器是 Kafka 用来和客户端或其他 broker 通信的网络接口。KAFKA\_CFG\_LISTENERS 的格式是一个由逗号分隔的 <name>://<host>:<port> 对，其中 <name> 是一个任意的标识符，<host> 是主机名或 IP 地址，<port> 是端口号。Name常用协议名称。**

****KAFKA\_CFG\_ADVERTISED\_LISTENERS：**它指定了 Kafka 代理向客户端或其他代理公开的监听器列表，包括它们的主机/IP和端口。这些监听器是客户端或其他代理用来连接到 Kafka 代理的地址。**

****KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP：****定义了每个监听器名称要使用的安全协议的键/值对。监听器是 Kafka 用来和客户端或其他 broker 通信的网络接口，每个监听器由主机/IP、端口和协议组成。安全协议是指通信时使用的加密方式，如 PLAINTEXT、SSL、SASL\_PLAINTEXT 等。

KAFKA\_CFG\_LISTENER\_SECURITY\_PROTOCOL\_MAP 的格式是一个由逗号分隔的 <name>:<protocol> 对，其中 <name> 是一个任意的标识符，<protocol> 是一个支持的安全协议

****KAFKA\_KRAFT\_CLUSTER\_ID：集群唯一uuid****

****KAFKA\_CFG\_CONTROLLER\_QUORUM\_VOTERS：kafka参与选举的broker名称及地址****

****KAFKA\_CFG\_AUTO\_CREATE\_TOPICS\_ENABLE：是否自动创建topic****

****KAFKA\_CFG\_NODE\_ID：kraft模式下节点id，从1开始，不重复****

****ALLOW\_PLAINTEXT\_LISTENER：**控制 Kafka 是否允许使用 PLAINTEXT 侦听器。PLAINTEXT 侦听器是指没有加密和认证的通信方式，这在生产环境中是不安全的，所以默认情况下是不允许的。如果你想在开发或测试环境中使用 PLAINTEXT 侦听器。**

****KAFKA\_HEAP\_OPTS：****用来设置 Kafka 的堆内存大小。堆内存是 JVM 用来存储对象的内存区域，它影响着 Kafka 的性能和稳定性。KAFKA\_HEAP\_OPTS 的格式是 -Xmx<size> -Xms<size>.

配置文档：https://kafka1x.apachecn.org/3/

## 使用kafka-ui管理

docker-compose文件

在docker中使用命令行操作kafka

需要进入：/opt/bitnami/kafka/bin目录

创建topic

> bin/kafka-topics.sh --bootstrap-server kafka1:9092,kafka2:9092,kafka3:9092 --create --topic my\_topic\_name \

--partitions 20 --replication-factor 3 --config x=y

查看所有的topic

kafka-topics.sh --list --bootstrap-server kafka1:9092,kafka2:9092,kafka3:9092

****操作文档：https://kafka.apachecn.org/documentation.html#operations****

****生产环境升级参数：****