

---

# Kafka

---

## Outline

Kafka简介

Kafka Core

【实践】搭建基于Kafka消息队列系统

【实践】Flume & Kafka

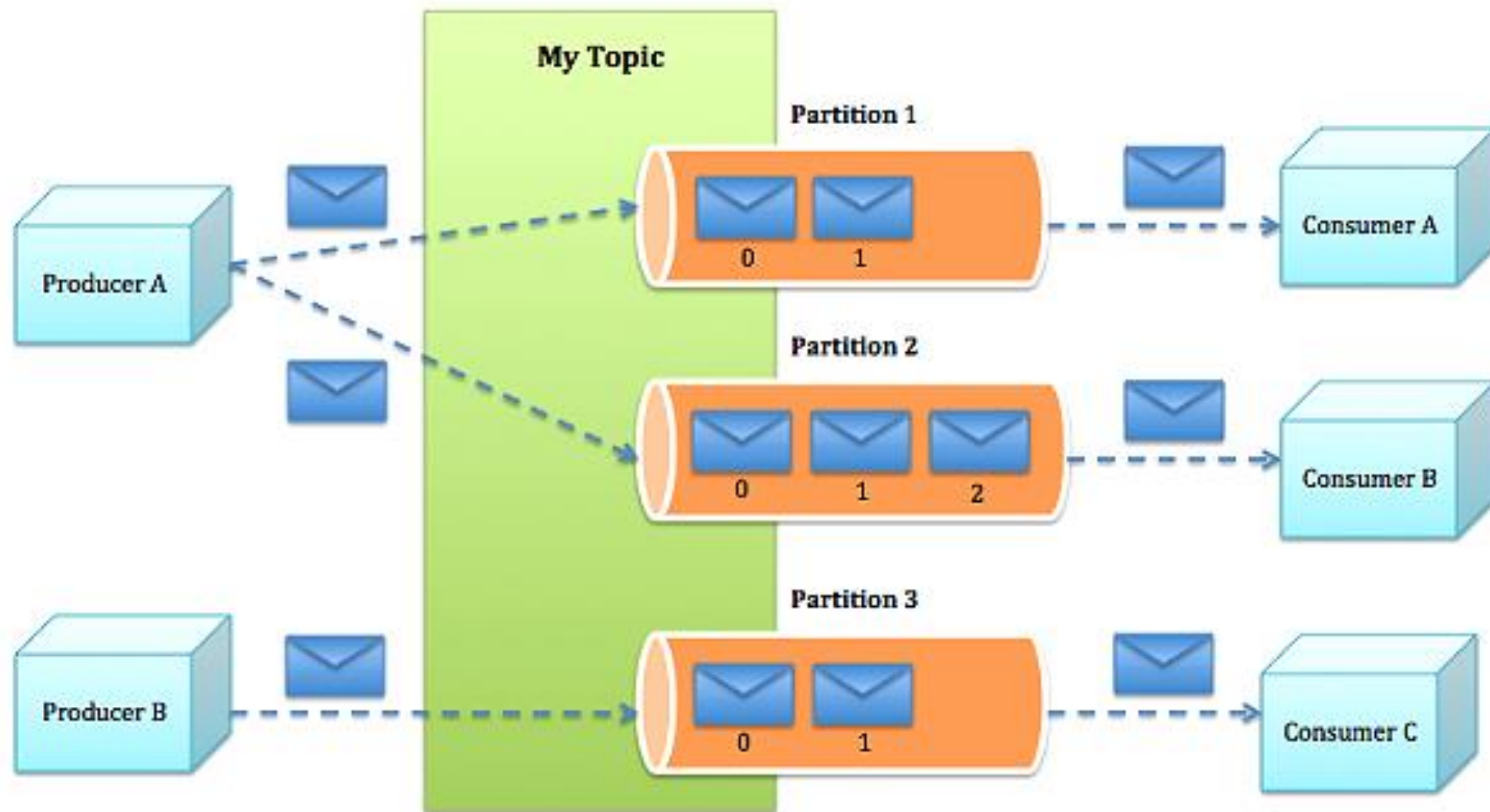
## 简介

- Kafka是Linkedin于2010年12月份开源的消息系统
- 一种分布式的、基于发布/订阅的消息系统
- 特点：
  - 消息持久化：通过O(1)的磁盘数据结构提供数据的持久化
  - 高吞吐量：每秒百万级的消息读写
  - 分布式：扩展能力强
  - 多客户端支持：java、php、python、c++ .....
  - 实时性：生产者生产的message立即被消费者可见

## 基本组件

- Broker: 每一台机器叫一个Broker
- Producer: 日志消息生产者, 用来写数据
- Consumer: 消息的消费者, 用来读数据
- Topic: 不同消费者去指定的Topic中读, 不同的生产者往不同的Topic中写
- Partition: 在Topic基础上做了进一步区分分层

## 基本组件

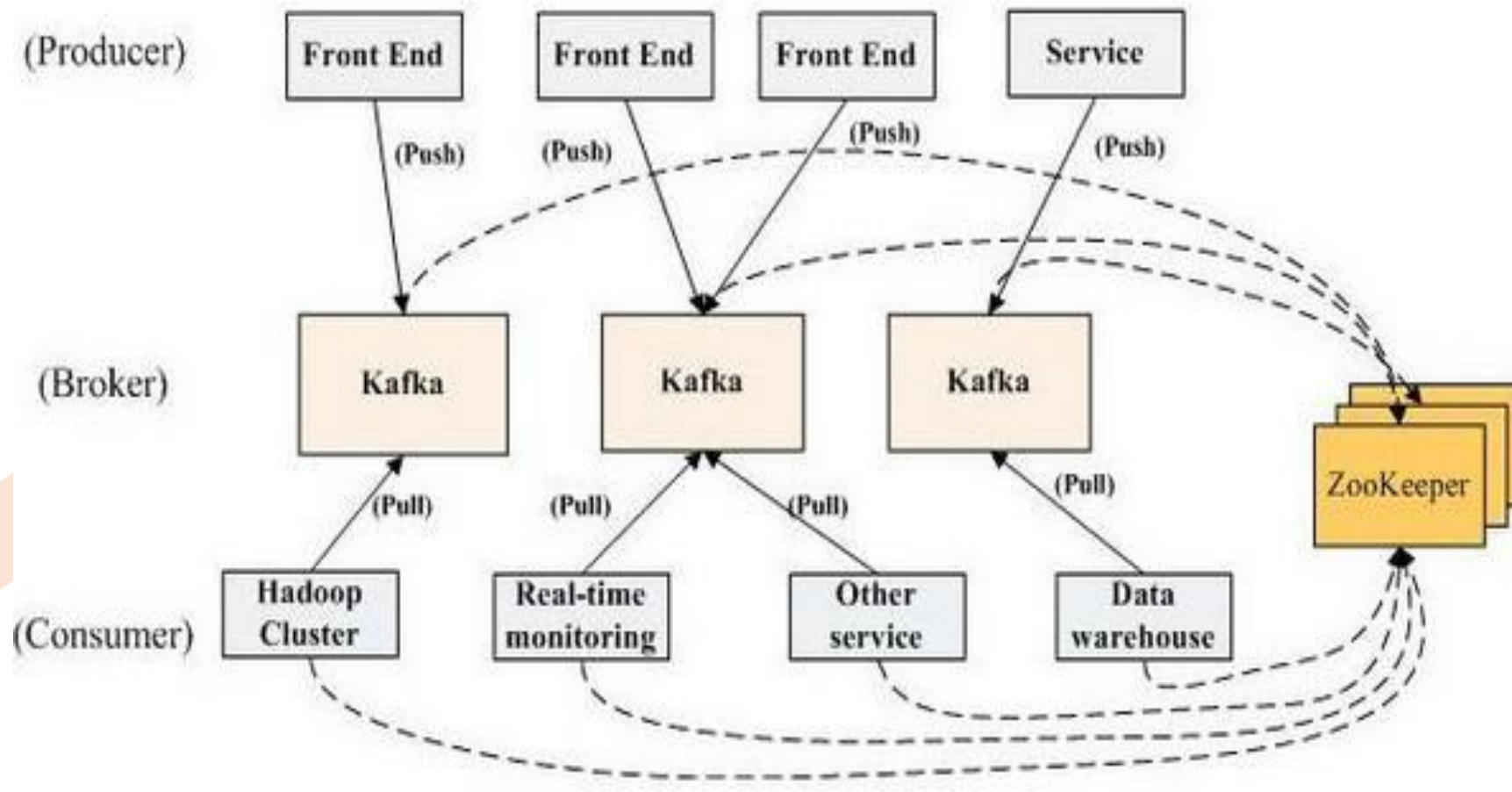


一个或多个Partition

- 多个Producer和Consumer同时生产 and 消费消息

—— 八斗大数据内部资料，盗版必究 ——

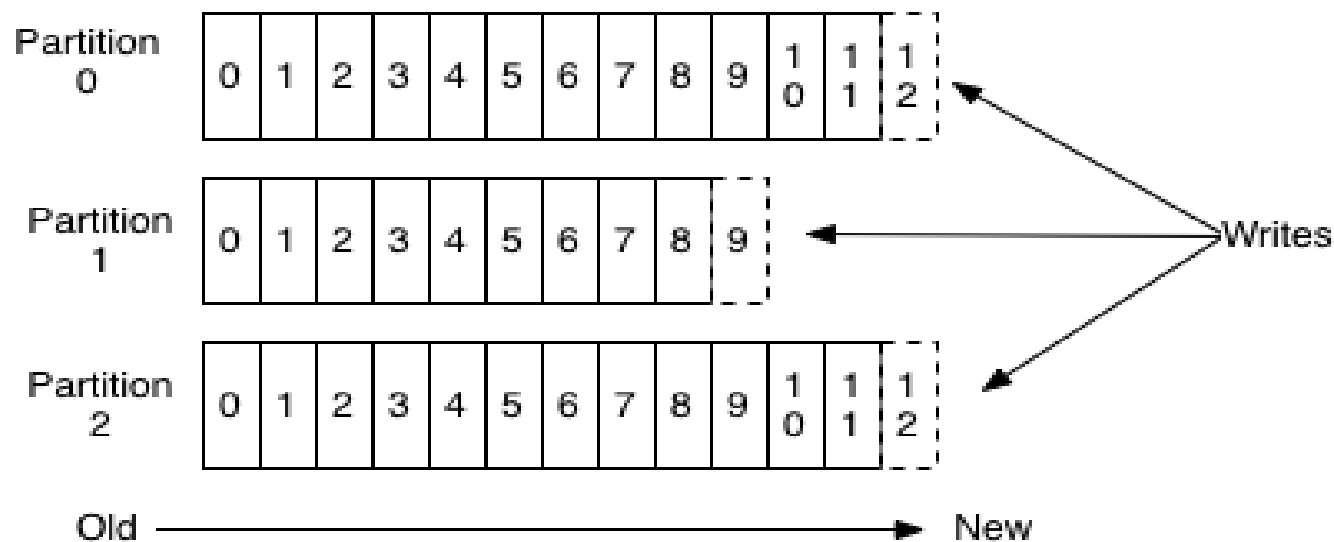
## 基本组件



## Topic

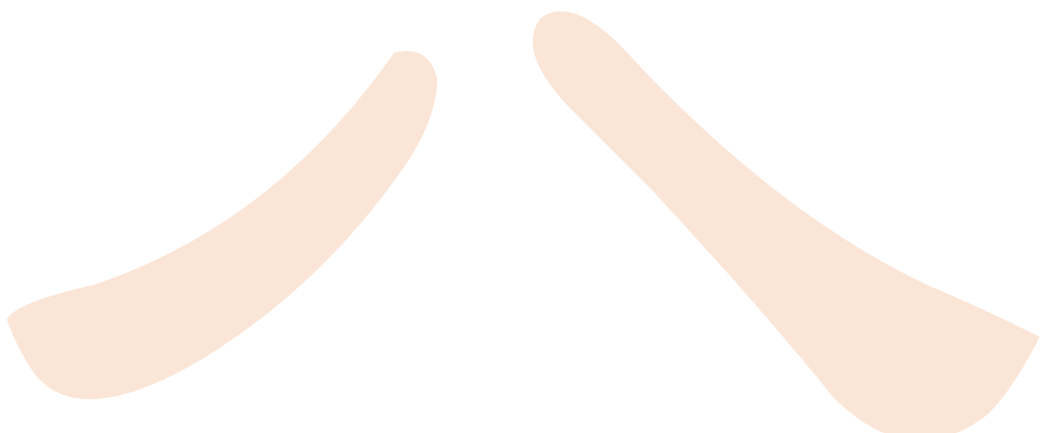
- 一个Topic是一个用于发布消息的分类或feed名，kafka集群使用分区的日志，每个分区都是有顺序且不变的消息序列。
- commit的log可以不断追加。消息在每个分区中都分配了一个叫offset的id序列来唯一识别分区中的消息。

## Anatomy of a Topic



## Topic

- 举例：若创建topic1和topic2两个topic，且分别有13个和19个分区，则整个集群上会相应会生成共32个文件夹



```
node4: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-10
node4: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-2
node4: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-10
node4: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-18
node4: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-2
node2: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-0
node2: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-8
node2: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-0
node2: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-16
node2: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-8
node8: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-6
node8: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-14
node8: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-6
node7: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-5
node7: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-13
node7: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-5
node3: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-1
node3: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-9
node3: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-1
node3: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-17
node3: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-9
node6: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-12
node6: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-4
node6: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-12
node6: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-4
node5: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-11
node5: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-3
node5: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-11
node5: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-3
node1: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic1-7
node1: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-15
node1: drwxr-xr-x 2 root root 4.0K Mar 3 13:01 topic2-7
```



## Topic

- 无论发布的消息是否被消费，kafka都会持久化一定时间（可配置）。
- 在每个消费者都持久化这个offset在日志中。通常消费者读消息时会使offset值线性的增长，但实际上其位置是由消费者控制，它可以按任意顺序来消费消息。比如复位到老的offset来重新处理。
- 每个分区代表一个并行单元。

## Message

- message（消息）是通信的基本单位，每个producer可以向一个topic（主题）发布一些消息。如果consumer订阅了这个主题，那么新发布的消息就会广播给这些consumer。
- message format:
  - message length : 4 bytes (value: 1+4+n)
  - "magic" value : 1 byte
  - crc : 4 bytes
  - payload : n bytes

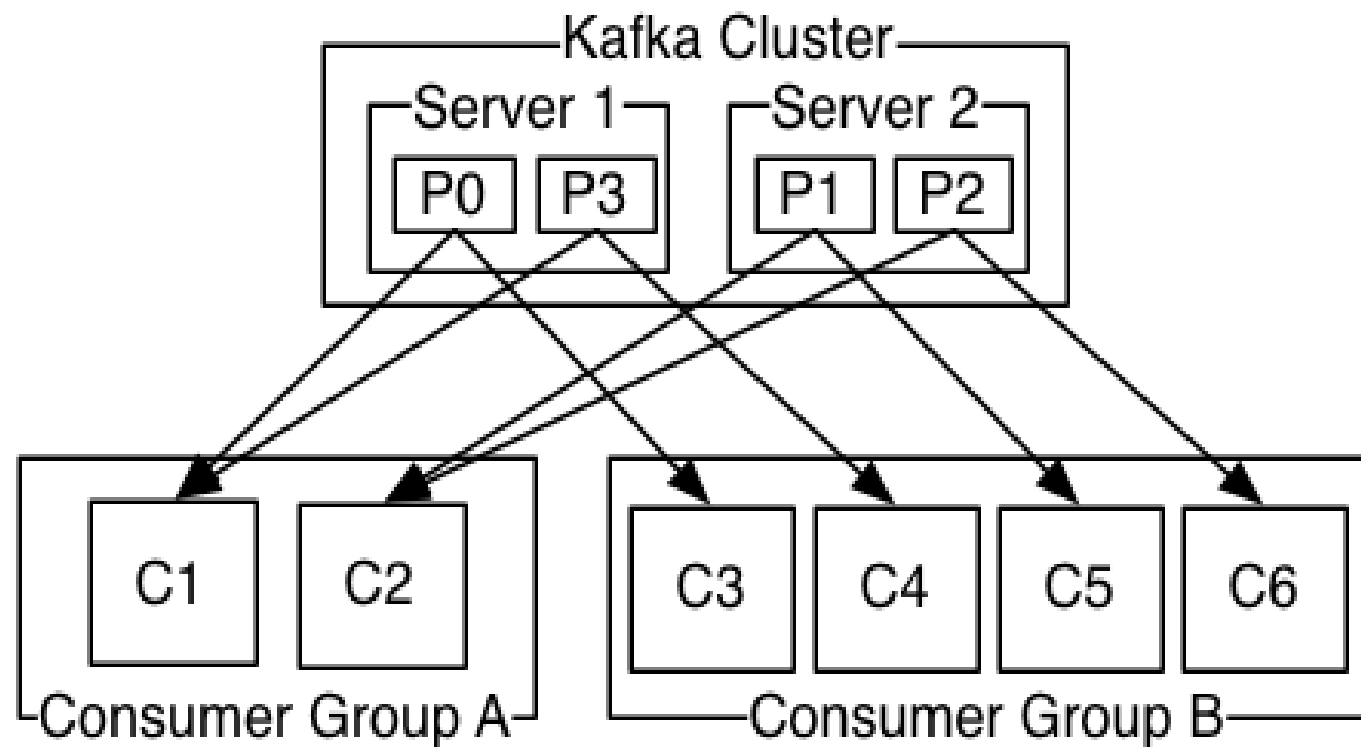
## Producer

- 生产者可以发布数据到它指定的topic中，并可以指定在topic里哪些消息分配到哪些分区（比如简单的轮流分发各个分区或通过指定分区语义分配key到对应分区）
- 生产者直接把消息发送给对应分区的broker，而不需要任何路由层。
- 批处理发送，当message积累到一定数量或等待一定时间后进行发送。

## Consumer

- 一种更抽象的消费方式：消费组（consumer group）
- 该方式包含了传统的queue和发布订阅方式
  - 首先消费者标记自己一个消费组名。消息将投递到每个消费组中的某一个消费者实例上。
  - 如果所有的消费者实例都有相同的消费组，这样就像传统的queue方式。
  - 如果所有的消费者实例都有不同的消费组，这样就像传统的发布订阅方式。
  - 消费组就好比是个逻辑的订阅者，每个订阅者由许多消费者实例构成(用于扩展或容错)。
- 相对于传统的消息系统，kafka拥有更强壮的顺序保证。
- 由于topic采用了分区，可在多Consumer进程操作时保证顺序性和负载均衡。

## Consumer



## Outline

Kafka简介

Kafka Core

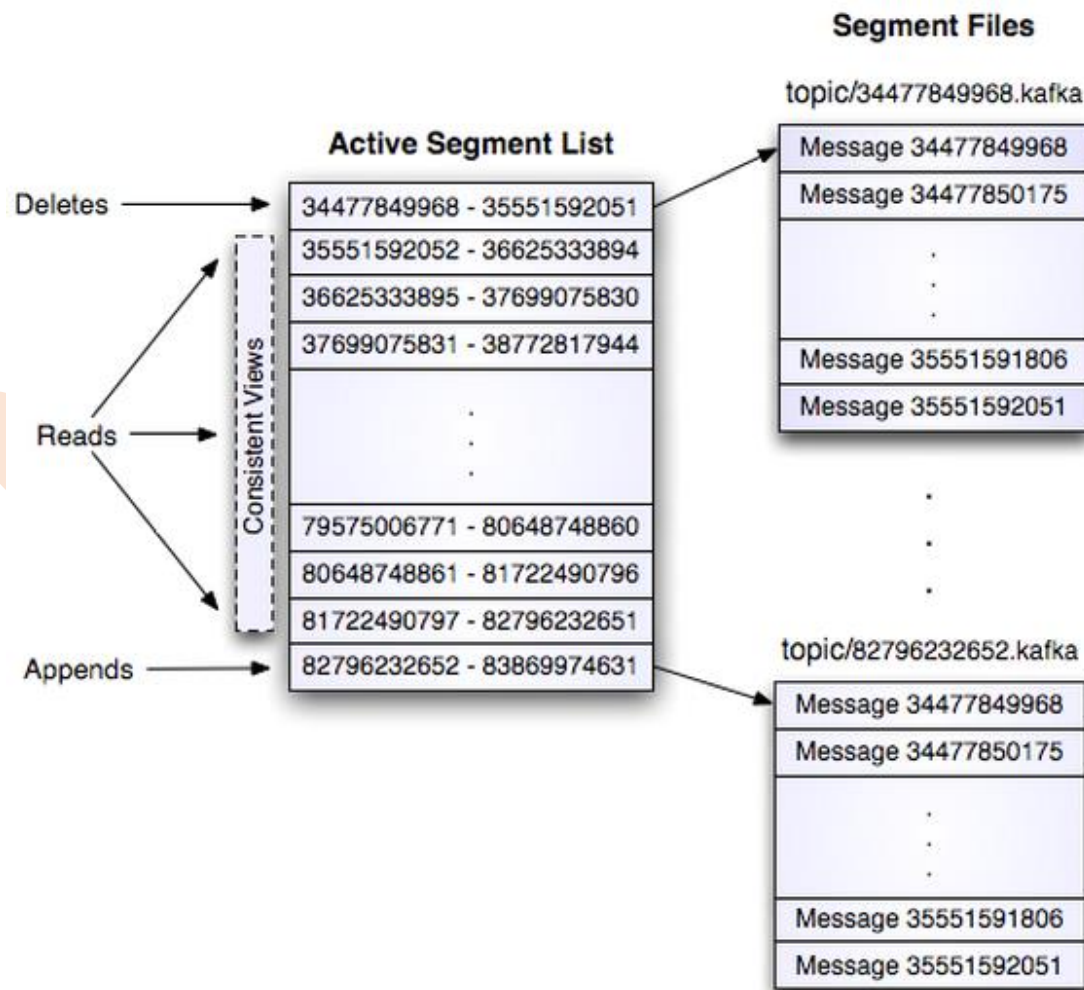
【实践】搭建基于Kafka消息队列系统

【实践】Flume & Kafka

## 持久化

- Kafka存储布局简单：Topic的每个Partition对应一个逻辑日志
- 每次生产者发布消息到一个分区，代理就将消息追加到最后一个段文件中。
- 与传统的消息系统不同，Kafka系统中存储的消息没有明确的消息Id。
- 消息通过日志中的逻辑偏移量来公开。

## 持久化



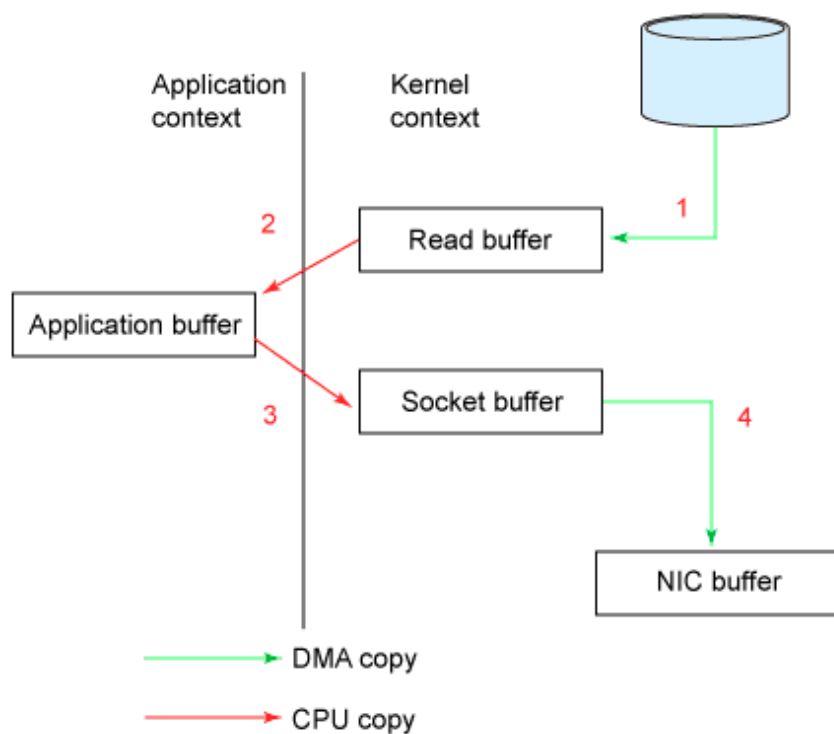


## 传输效率

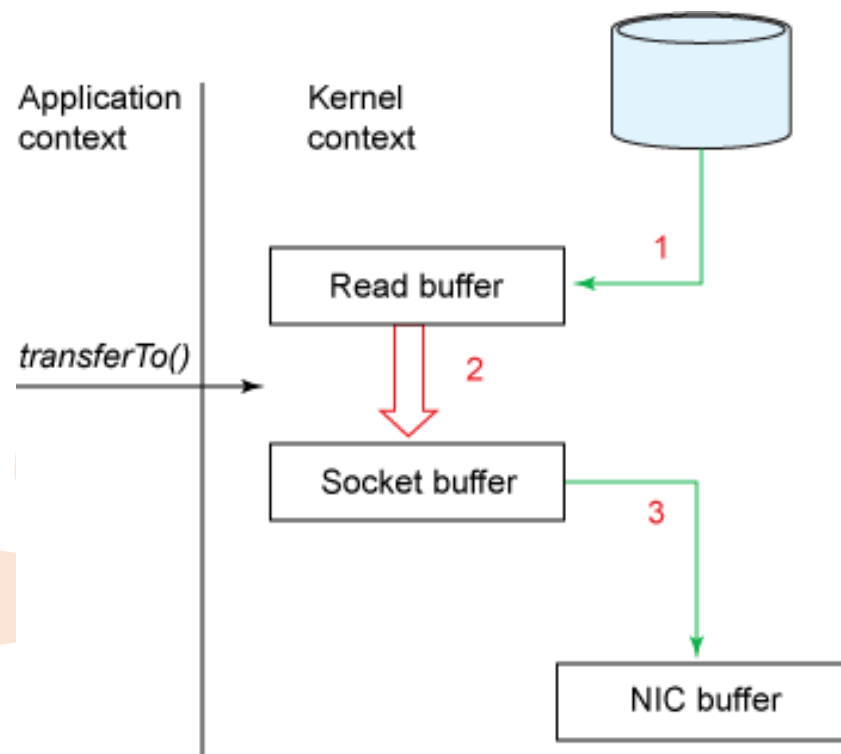
- 生产者提交一批消息作为一个请求。消费者虽然利用api遍历消息是一个一个的，但背后也是一次请求获取一批数据，从而减少网络请求数量。
- Kafka层采用无缓存设计，而是依赖于底层的文件系统页缓存。这有助于避免双重缓存，及即消息只缓存了一份在页缓存中。同时这在kafka重启后保持缓存warm也有额外的优势。因kafka根本不缓存消息在进程中，故gc开销也就很小
- zero-copy: kafka为了减少字节拷贝，采用了大多数系统都会提供的sendfile系统调用

## 传输效率

## 传统方式



## Zero-Copy



## 无状态的 Broker

- Kafka代理是无状态的：意味着消费者必须维护已消费的状态信息。这些信息由消费者自己维护，代理完全不管。这种设计非常微妙，它本身包含了创新
  - 从代理删除消息变得很棘手，因为代理并不知道消费者是否已经使用了该消息。Kafka创新性地解决了这个问题，它将一个简单的基于时间的SLA应用于保留策略。当消息在代理中超过一定时间后，将会被自动删除。
  - 这种创新设计有很大的好处，消费者可以故意倒回到老的偏移量再次消费数据。这违反了队列的常见约定，但被证明是许多消费者的基本特征。

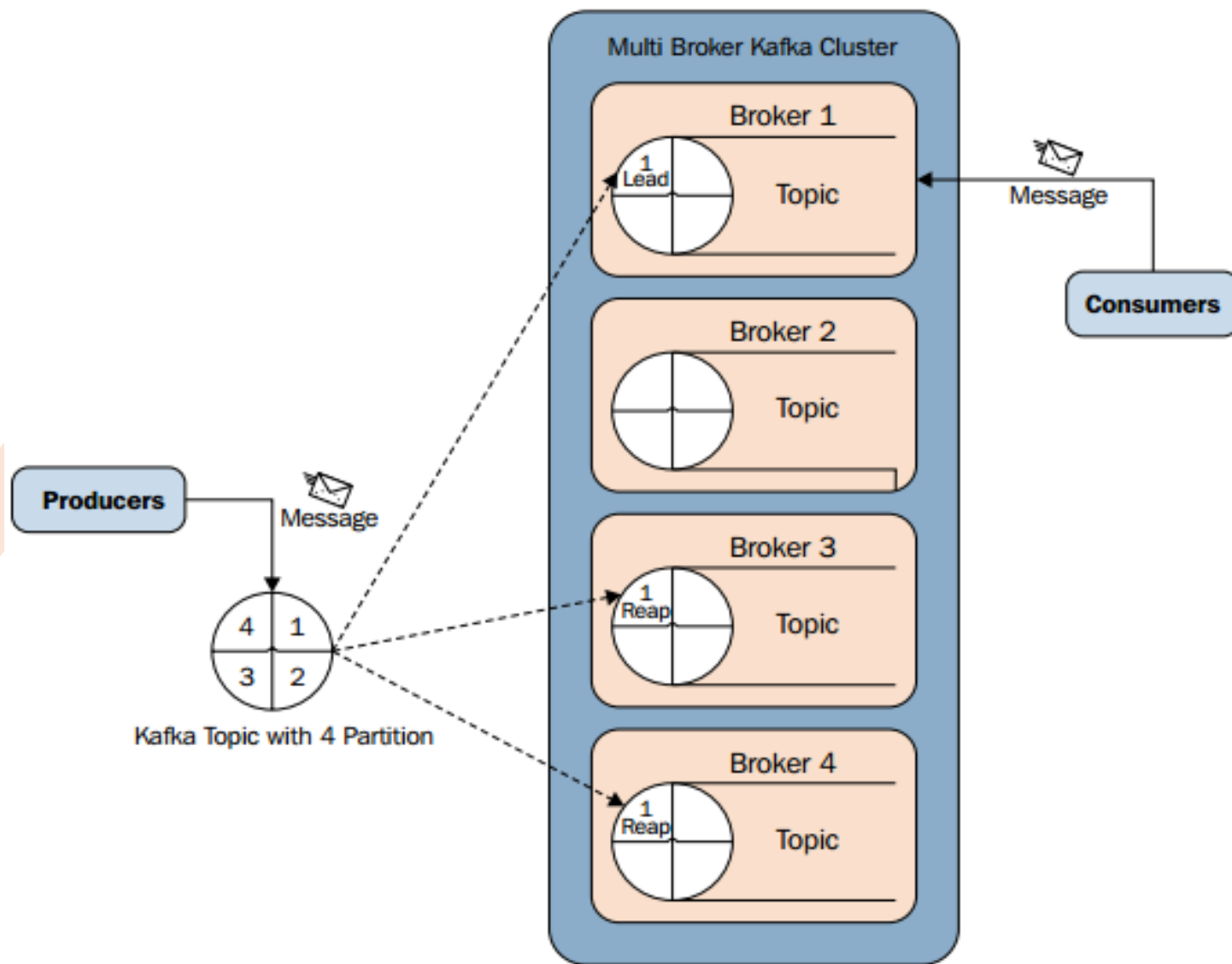
## 交付保证

- Kafka默认采用at least once的消息投递策略。即在消费者端的处理顺序是获得消息->处理消息->保存位置。这可能导致一旦客户端挂掉，新的客户端接管时处理前面客户端已处理过的消息。
- 三种保证策略：
  - At most once 消息可能会丢，但绝不会重复传输
  - At least one 消息绝不会丢，但可能会重复传输
  - Exactly once 每条消息肯定会被传输一次且仅传输一次

## 副本管理

- kafka将日志复制到指定多个服务器上。
- 复本的单元是partition。在正常情况下，每个分区有一个leader和0到多个follower。
- leader处理对应分区上所有的读写请求。分区可以多于broker数，leader也是分布式的。
- follower的日志和leader的日志是相同的，follower被动的复制leader。如果leader挂了，其中一个follower会自动变成新的leader。

## 副本管理



## 副本管理

- 和其他分布式系统一样，节点“活着”定义在于我们能否处理一些失败情况。

kafka需要两个条件保证是“活着”

- 节点在zookeeper注册的session还在且可维护（基于zookeeper心跳机制）
  - 如果是slave则能够紧随leader的更新不至于落得太远。
- 
- kafka采用in sync来代替“活着”
    - 如果follower挂掉或卡住或落得很远，则leader会移除同步列表中的in sync。至于落了多远才叫远由replica.lag.max.messages配置，而表示复本“卡住”由replica.lag.time.max.ms配置

## 副本管理

- 所谓一条消息是“提交”的，意味着所有in sync的副本也持久化到了他们的log中。这意味着消费者无需担心leader挂掉导致数据丢失。另一方面，生产者可以选择是否等待消息“提交”。
- kafka动态的维护了一组in-sync(ISR)的副本，表示已追上了leader,只有处于该状态的成员组才是能被选择为leader。这些ISR组会在发生变化时被持久化到zookeeper中。通过ISR模型和 $f+1$ 副本，可以让kafka的topic支持最多 $f$ 个节点挂掉而不会导致提交的数据丢失。



## 分布式协调

- 由于kafka中一个topic中的不同分区只能被消费组中的一个消费者消费，就避免了多个消费者消费相同的分区时会导致额外的开销（如要协调哪个消费者消费哪个消息，还有锁及状态的开销）。kafka中消费进程只需要在代理和同组消费者有变化时时进行一次协调（这种协调不是经常性的，故可以忽略开销）。
- kafka使用zookeeper做以下事情：
  - 探测broker和consumer的添加或移除
  - 当1发生时触发每个消费者进程的重新负载。
  - 维护消费关系和追踪消费者在分区消费的消息的offset。

## Zookeeper 的使用

- **Broker Node Registry**
- `/brokers/ids/[0...N]` --> `host:port` (ephemeral node)
  - broker启动时在/brokers/ids下创建一个znode, 把broker id写进去。
  - 因为broker把自己注册到zookeeper中实用的是瞬时节点, 所以这个注册是动态的, 如果broker宕机或者没有响应该节点就会被删除。
- **Broker Topic Registry**
- `/brokers/topics/[topic]/[0...N]` --> `nPartions` (ephemeral node)
  - 每个broker把自己存储和维护的partion信息注册到该路径下。

## Zookeeper 的使用

- **Consumers and Consumer Groups**

- consumers也把它们自己注册到zookeeper上，用以保持消费负载平衡和offset记录。
- group id相同的多个consumer构成一个消费组，共同消费一个topic，同一个组的consumer会尽量均匀的消费，其中的一个consumer只会消费一个partion的数据。

- **Consumer Id Registry**

- `/consumers/[group_id]/ids/[consumer_id] --> {"topic1": #streams, ..., "topicN": #streams} (ephemeral node)`

- 每个consumer在/consumers/[group\_id]/ids下创建一个瞬时的唯一的consumer\_id，用来描述当前该group下有哪些consumer是alive的，如果消费进程挂掉对应的consumer\_id就会从该节点删除。

## Zookeeper 的使用

- **Consumer Offset Tracking**
- `/consumers/[group_id]/offsets/[topic]/[partition_id] --> offset_counter_value ((persistent node)`
  - consumer把每个partition的消费offset记录保存在该节点下。
- **Partition Owner registry**
- `/consumers/[group_id]/owners/[topic]/[broker_id-partition_id] --> consumer_node_id (ephemeral node)`
  - 该节点维护着partition与consumer之间的对应关系。

## Outline

Kafka简介

Kafka Core

【实践】搭建基于Kafka消息队列系统

【实践】Flume & Kafka

## K a f k a 对比其他消息服务

- 下载压缩包：
  - ]# wget [http://mirrors.hust.edu.cn/apache/kafka/0.10.2.1/kafka\\_2.11-0.10.2.1.tgz](http://mirrors.hust.edu.cn/apache/kafka/0.10.2.1/kafka_2.11-0.10.2.1.tgz)
- 解压：
  - ]# tar xvfz kafka\_2.11-0.10.2.1.tgz

## K a f k a 对比其他消息服务

- **启动Zookeeper:**

- ]# ./bin/zookeeper-server-start.sh config/zookeeper.properties

- **启动server:**

- ]# ./bin/kafka-server-start.sh config/server.properties

- **创建topic:**

- ]# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

## K a f k a 对比其他消息服务

## • 查看主题:

- ]# bin/kafka-topics.sh --list --zookeeper localhost:2181

## • 查看主题详情:

- ]# bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test

## • 删除主题:

- ]# bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic test

- 注意: 如果kafaka启动时加载的配置文件中server.properties没有配置delete.topic.enable=true, 那么此时的删除并不是真正的删除, 而是把topic标记为: marked for deletion

- 此时你若想真正删除它, 可以登录zookeeper客户端, 进入终端后, 删除相应节点



## 搭建 K a f k a 分布式集群，并读写消息

## • 单独创建Topic:

- ]# bin/kafka-topics.sh --create --zookeeper 192.168.183.100:2181 --replication-factor 2 -  
-partitions 1 --topic badou

## • Producer发消息:

- ]# ./bin/kafka-console-producer.sh --broker-list master:9092 --topic badou

## • Consumer读消息:

- ]# ./kafka-console-consumer.sh --zookeeper master:2181 --topic badou --from-  
beginning

## Outline

Kafka简介

Kafka Core

【实践】搭建基于Kafka消息队列系统

【实践】Flume & Kafka

## Flume &amp; Kafka 结合应用

- 启动 **F L u m e**

- ]# ./bin/flume-ng agent --conf conf --conf-file ./conf/flume.conf -name producer -Dflume.root.logger=DEBUG,console

- 启动 **Z o o k e e p e r**

- 略

- 启动 **K a f k a S e r v e r**

- ]# ./bin/kafka-server-start.sh config/server.properties

- 发送消息

- ]# echo '八斗学院' | nc -u master 8285

- 启动 **C o n s u m e r** 进行数据监控

- ]# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning

---

# Q & A

@八斗学院

---