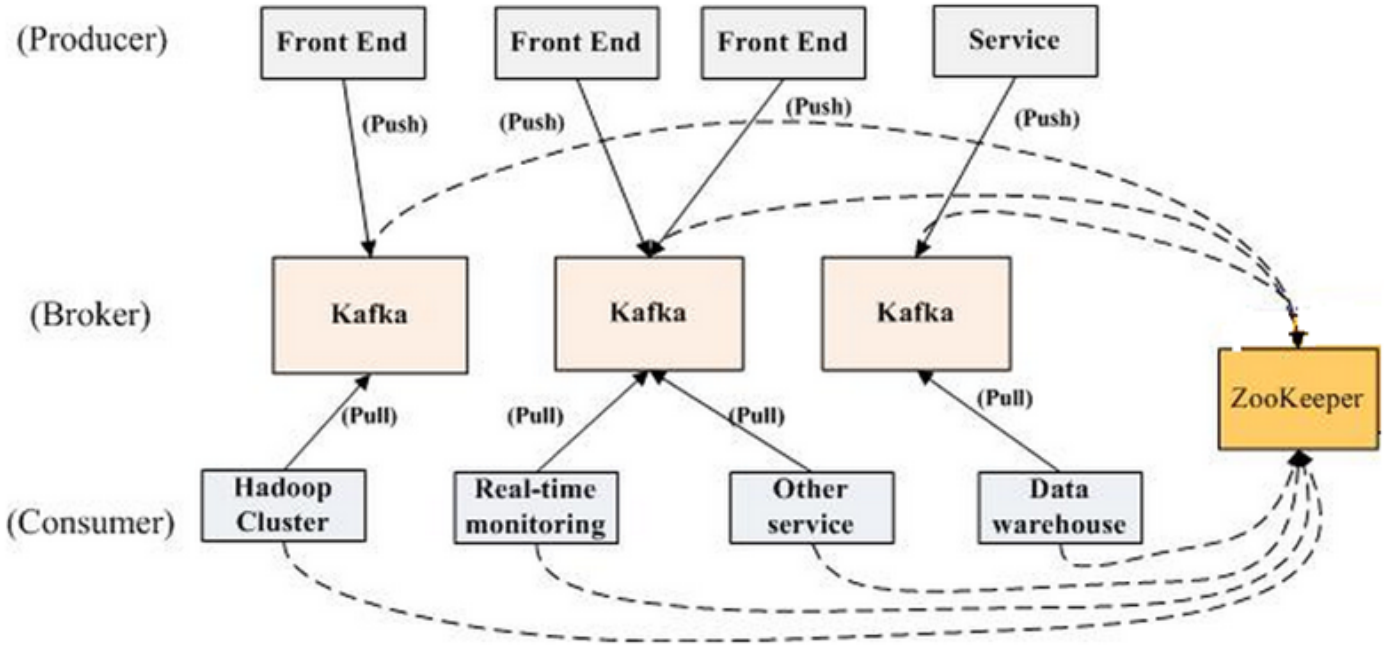
1. 讲一下kafka 的架构



- **Producer**：消息生产者
  - Producer可以发送消息到Topic
  - - Topic的消息存放在不同Partition中，不同Partition存放在不同Broker中
    - Producer只需要指定Topic的名字、要连接到的Broker，这样Kafka就可以自动地把消息数据路由到合适的Broker（不一定是指定连接的Broker）
  - Producer发送消息后，可以选择是否要确认消息写入成功（ACK，Acknowledgment）
  - - ACK=0：Producer不会等待ACK（消息可能丢失）
    - ACK=1：Producer会等待Leader Partition的ACK（Follower Partition消息可能丢失）
    - ACK=all：Producer会等待Leader Partition和Follower Partition的ACK（消息不会丢失）
  - 消息key：Producer可以给消息加上key，带相同key的消息会被分发到同一个Partition，这样就可以保证带相同key的消息的消费是有序的
- **Broker**：每个Broker里包含了不同Topic的不同Partition，Partition中包含了有序的消息
  - 一个Kafka集群由多个Broker（server）组成
  - 每个Broker都有ID标识
  - 每个Broker里保存一定数量的Partition
  - 客户端只要连接上任意一个Broker，就可以连接上整个Kafka集群
  - 大多数Kafka集群刚开始的时候建议使用至少3个Broker，集群大了可以有上百个Broker
- **Consumer**：消息消费者
  - Consumer可以从Topic读取消息进行消费
  - - Topic的消息存放在不同Partition中，不同Partition存放在不同Broker中
    - Consumer只需要指定Topic的名字、要连接到的Broker，这样Kafka就可以自动地把Consumer路由到合适的Broker拉取消息进行消费（不一定是指定连接的Broker）
    - 每一个Partition中的消息都会被有序消费
  - Consumer Group：
  - - Consumer Group由多个Consumer组成

        - Consumer Group里的每个Consumer都会从不同的Partition中读取消息
        - 如果Consumer的数量大于Partition的数量，那么多出来的Consumer就会空闲下来（浪费资源）
    - Consumer offset：
    - - Kafka会为Consumer Group要消费的每个Partion保存一个offset，这个offset标记了该Consumer Group最后消费消息的位置
        - 这个offset保存在Kafka里一个名为“__consumer_offsets”的Topic中；当Consumer从Kafka拉取消息消费时，同时也要对这个offset提交修改更新操作。这样若一个Consumer消费消息时挂了，其他Consumer可以通过这个offset值重新找到上一个消息再进行处理

2. kafka的message包括哪些信息

一个Kafka的Message由一个固定长度的header和一个变长的消息体body组成 header部分由一个字节的magic(文件格式)和四个字节的CRC32(用于判断body消息体是否正常)构成。当magic的值为1的时候，会在magic和crc32之间多一个字节的数据：attributes(保存一些相关属性，比如是否压缩、压缩格式等等)；如果magic的值为0，那么不存在attributes属性 body是由N个字节构成的一个消息体，包含了具体的key/value消息

3. kafka 可以脱离 zookeeper 单独使用吗

kafka 不能脱离 zookeeper 单独使用，因为 kafka 使用 zookeeper 管理和协调 kafka 的节点服务器。

4. kafka 实现高吞吐的原理

- 读写文件依赖OS文件系统的页缓存，而不是在JVM内部缓存数据，利用OS来缓存，内存利用率高
- sendfile技术（零拷贝），避免了传统网络IO四步流程
- 支持End-to-End的压缩
- 顺序IO以及常量时间get、put消息
- Partition 可以很好的横向扩展和提供高并发处理

5. kafka同时设置了7天和10G清除数据,到第5天的时候消息到达了10G,这个时候kafka如何处理?

这个时候 kafka 会执行数据清除工作，时间和大小不论那个满足条件，都会清空数据。

6. kafka 有几种数据保留策略

kafka 有两种数据保存策略：按照过期时间保留和按照存储的消息大小保留。

7. kafka 与其他消息组件对比

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类的系统来进行实时数据计算、日志采集等场景
topic 数量对吞吐量的影响			topic 可以达到几百/几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器下，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，需要增加更多的机器资源
时效性	ms 级	微秒级，这是 RabbitMQ 的一大特点，延迟最低	ms 级	延迟在 ms 级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据	基本不丢	经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用

8. kafka 与 spark streaming 集成,如何保证 exactly once 语义

Spark Streaming上游对接kafka时保证Exactly Once

Spark Streaming使用Direct模式对接上游kafka。无论kafka有多少个partition， 使用Direct模式总能保证SS中有相同数量的partition与之相对， 也就是说SS中的KafkaRDD的并发数量在Direct模式下是由上游kafka决定的。 在这个模式下，kafka的offset是作为KafkaRDD的一部分存在，会存储在checkpoints中， 由于checkpoints只存储offset内容，而不存储数据，这就使得checkpoints是相对轻的操作。 这就使得SS在遇到故障时，可以从checkpoint中恢复上游kafka的offset，从而保证exactly once

Spark Streaming输出下游保证Exactly once

- 第一种“鸵鸟做法”，就是期望下游（数据）具有幂等特性。

  多次尝试总是写入相同的数据，例如，saveAs***Files 总是将相同的数据写入生成的文件

- - 使用事务更新

    所有更新都是事务性的，以便更新完全按原子进行。这样做的一个方法如下： 使用批处理时间(在foreachRDD中可用)和RDD的partitionIndex（分区索引）来创建identifier（标识符）。 该标识符唯一地标识streaming application 中的blob数据。 使用该identifier，blob 事务地更新到外部系统中。也就是说，如果identifier尚未提交，则以 (atomicall)原子方式提交分区数据和identifier。否则，如果已经提交，请跳过更新。

9. kafka怎样保证不丢失消息

消费端弄丢了数据

唯一可能导致消费者弄丢数据的情况，就是说，你消费到了这个消息，然后消费者那边**自动提交了 offset**，让 Kafka 以为你已经消费好了这个消息，但其实你才刚准备处理这个消息，你还没处理，你自己就挂了，此时这条消息就丢咯。

这不是跟 RabbitMQ 差不多吗，大家都知道 Kafka 会自动提交 offset，那么只要**关闭自动提交** offset，在处理完之后自己手动提交 offset，就可以保证数据不会丢。但是此时确实还是**可能会有重复消费**，比如你刚处理完，还没提交 offset，结果自己挂了，此时肯定会重复消费一次，自己保证幂等性就好了。

生产环境碰到的一个问题，就是说我们的 Kafka 消费者消费到了数据之后是写到一个内存的 queue 里先缓冲一下，结果有的时候，你刚把消息写入内存 queue，然后消费者会自动提交 offset。然后此时我们重启了系统，就会导致内存 queue 里还没来得及处理的数据就丢失了。

Kafka 弄丢了数据

这块比较常见的一个场景，就是 Kafka 某个 broker 宕机，然后重新选举 partition 的 leader。大家想想，要是此时其他的 follower 刚好还有些数据没有同步，结果此时 leader 挂了，然后选举某个 follower 成 leader 之后，不就少了一些数据？这就丢了一些数据啊。

生产环境也遇到过，我们也是，之前 Kafka 的 leader 机器宕机了，将 follower 切换为 leader 之后，就会发现说这个数据就丢了。

所以此时一般是要求起码设置如下 4 个参数：

- 给 topic 设置 `replication.factor` 参数：这个值必须大于 1，要求每个 partition 必须有至少 2 个副本。
- 在 Kafka 服务端设置 `min.insync.replicas` 参数：这个值必须大于 1，这个是要求一个 leader 至少感知到有至少一个 follower 还跟自己保持联系，没掉队，这样才能确保 leader 挂了还有一个 follower 吧。
- 在 producer 端设置 `acks=all`：这个是要求每条数据，必须是**写入所有 replica 之后，才能认为是写成功了**。
- 在 producer 端设置 `retries=MAX`（很大很大很大的一个值，无限次重试的意思）：这个是**要求一旦写入失败，就无限重试**，卡在这里了。

我们生产环境就是按照上述要求配置的，这样配置之后，至少在 Kafka broker 端就可以保证在 leader 所在 broker 发生故障，进行 leader 切换时，数据不会丢失。

生产者会不会弄丢数据?

如果按照上述的思路设置了 `acks=all`，一定不会丢，要求是，你的 leader 接收到消息，所有的 follower 都同步到了消息之后，才认为本次写成功了。如果没满足这个条件，生产者会自动不断的重试，重试无限次。

10. kafka怎样保证不重复消费

此问题其实等价于保证消息队列消费的幂等性

主要需要结合实际业务来操作:

- 比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update 一下好吧。
- 比如你是写 Redis，那没问题了，反正每次都是 set，天然幂等性。
- 比如你不是上面两个场景，那做的稍微复杂一点，你需要让生产者发送每条数据的时候，里面加一个全局唯一的 id，类似订单 id 之类的东西，然后你这里消费到了之后，先根据这个 id 去比如 Redis 里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个 id 写 Redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。
- 比如基于数据库的唯一键来保证重复数据不会重复插入多条。因为有唯一键约束了，重复数据插入只会报错，不会导致数据库中出现脏数据。

注：资料来源于网络。