Kafka介绍

kun.liu 2018.01
https://github.com/liukgg

原理方面本次只进行简单分享，不做深入分析。

目录

- Kafka基本介绍
- 常用消息系统简单对比
- Kafka架构简介
- Kafka简单实践
- Kafka工作原理

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

目录

- **Kafka基本介绍**
- 常用消息系统简单对比
- Kafka架构简介
- Kafka简单实践
- Kafka工作原理

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

基本介绍

Apache Kafka® is a distributed streaming platform. What exactly does that mean?
We think of a streaming platform as having three key capabilities:
It lets you publish and subscribe to streams of records. In this respect it is similar to a message queue or enterprise messaging system.
It lets you store streams of records in a fault-tolerant way.
It lets you process streams of records as they occur.

基本介绍

- Kafka是一个分布式的流处理平台；用Scala和Java编写；Apache开源项目
- 设计目标：
  - 以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上数据也能保证常数时间复杂度的访问性能。
  - 高吞吐率，即使在非常廉价的商用机器上也能做到单机支持每秒100K条以上消息的传输。
  - 支持Kafka Server间的消息分区，及分布式消费，同时保证每个Partition内的消息顺序传输。
  - 同时支持离线数据处理和实时数据处理。
  - Scale out：支持在线水平扩展。

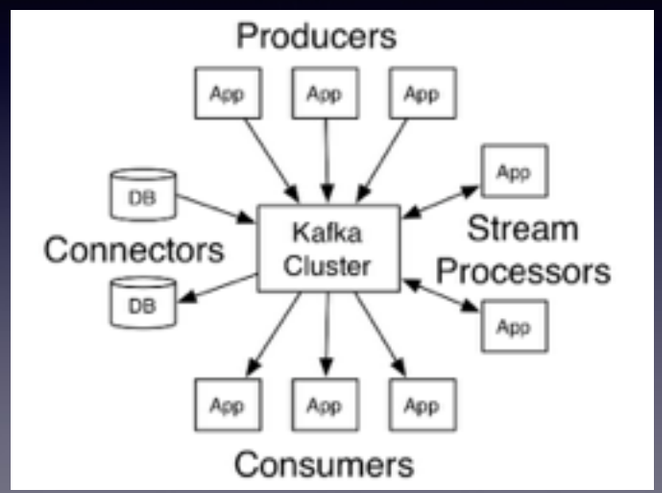
4个核心API：

The Producer API allows an application to publish a stream of records to one or more Kafka topics.
The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
The Streams API
The Connector API

http://www.infoq.com/cn/articles/kafka-analysis-part-1

4个核心API

- The Producer API
- The Consumer API
- The Streams API
- The Connector API



Kafka高效地处理实时流式数据，可以实现与Storm、HBase和Spark的集成。作为集群部署到多台服务器上，Kafka处理它所有的发布和订阅消息系统使用了四个API，即生产者API、消费者API、Stream API和Connector API。它能够传递大规模流式消息，自带容错功能。

Kafka有四个主要API：

生产者API：支持应用程序发布Record流。

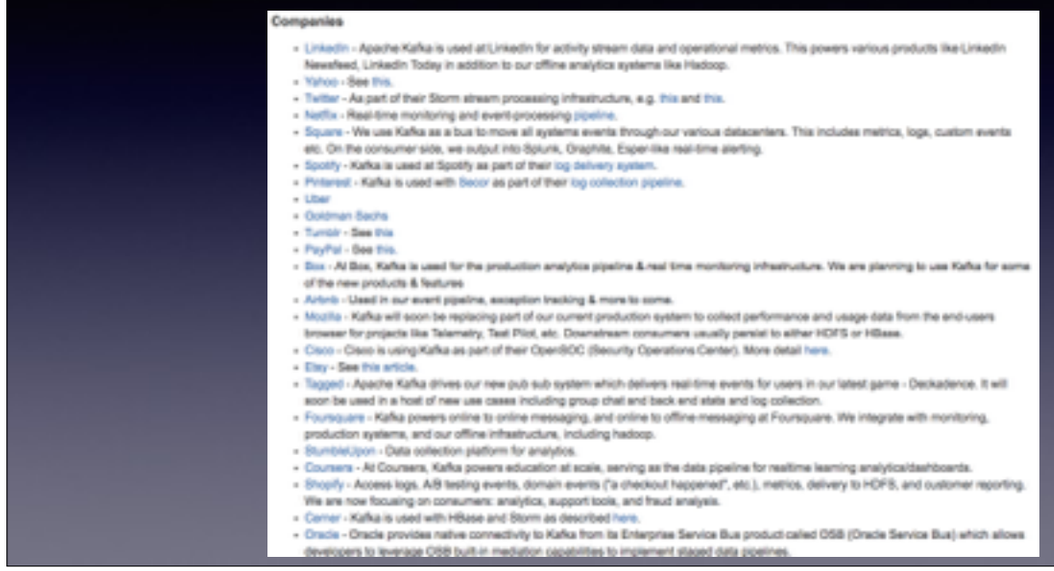
消费者API：支持应用程序订阅Topic和处理Record流。

Stream API：将输入流转换为输出流，并产生结果。

Connector API：执行可重用的生产者和消费者API，可将Topic链接到现有应用程序。

https://kafka.apache.org/intro

Powered by



Companies

- LinkedIn - Apache Kafka is used at LinkedIn for activity stream data and operational metrics. This powers various products like LinkedIn Newsfeed, LinkedIn Today in addition to our offline analytics systems like Hadoop.
- Yahoo - See this.
- Twitter - As part of their Storm stream processing infrastructure, e.g. this and this.
- Netflix - Real-time monitoring and event-processing pipeline.
- Square - We use Kafka as a bus to move all systems events through our various datacenters. This includes metrics, logs, custom events etc. On the consumer side, we output into Splunk, Graphite, Esper-like real-time alerting.
- Spotify - Kafka is used at Spotify as part of their log delivery system.
- Pinterest - Kafka is used with Secor as part of their log collection pipeline.
- Uber
- Goldman Sachs
- Tumblr - See this
- PayPal - See this.
- Box - At Box, Kafka is used for the production analytics pipeline & real time monitoring infrastructure. We are planning to use Kafka for some of the new products & features
- Airbnb - Used in our event pipeline, exception tracking & more to come.
- Mozilla - Kafka will soon be replacing part of our current production system to collect performance and usage data from the end-users browser for projects like Telemetry, Test Pilot, etc. Downstream consumers usually persist to either HDFS or HBase.
- Cisco - Cisco is using Kafka as part of their OpenSOC (Security Operations Center). More detail here.
- Etsy - See this article.
- Tagged - Apache Kafka drives our new pub-sub system which delivers real-time events for users in our latest game - Deckadence. It will soon be used in a host of new use cases including group chat and back end stats and log collection.
- Foursquare - Kafka powers online to online messaging, and online to offline messaging at Foursquare. We integrate with monitoring, production systems, and our offline infrastructure, including hadoop.
- StumbleUpon - Data collection platform for analytics.
- Coursera - At Coursera, Kafka powers education at scale, serving as the data pipeline for realtime learning analytics/dashboards.
- Shopify - Access logs, A/B testing events, domain events ("a checkout happened", etc.), metrics, delivery to HDFS, and customer reporting. We are now focusing on consumers: analytics, support tools, and fraud analysis.
- Cerner - Kafka is used with HBase and Storm as described here.
- Oracle - Oracle provides native connectivity to Kafka from its Enterprise Service Bus product called OSB (Oracle Service Bus) which allows developers to leverage OSB built-in mediation capabilities to implement staged data pipelines.

来自世界各地的数千家公司在使用 Kafka，包括三分之一的 500 强公司。

Kafka 和其他的大数据平台都不同，它的主要目的不是数据的存储或者处理，而是用来做数据交换的。

https://cwiki.apache.org/confluence/display/KAFKA/Powered+By

http://www.10tiao.com/html/46/201711/2650998759/1.html

历史

- Kafka最初是由LinkedIn开发
- 2011年初由LinkedIn开源
- 2012年10月，由Apache Incubator孵化出站
- 2014年11月，几个曾在领英为Kafka工作的工程师，创建Confluent公司，并着眼于Kafka
- 2017年11月，Kafka 1.0.0 发布

Kafka版本历史	
2012.01.04	0.7.0
2013.12.03	0.8.0
2015.11.23	0.9.0.0
2016.05.22	0.10.0.0
2017.06.28	0.11.0.0
2017.11.01	1.0.0

0.8 Release High level consumer, Performance, Replication design 等
0.9 Release Consumer client redesign; Kafka Connect
0.10 Release Kafka Streams
0.11 Add Exactly-Once Semantics to Streams (https://archive.apache.org/dist/kafka/0.11.0.0/RELEASE_NOTES.html)
1.0.0 改进： Streams API，Conncect的度量指标，支持java 9， 磁盘容错优化等

根据2014年Quora的帖子，Jay Kreps似乎已经将它以作家弗朗茨·卡夫卡命名。Kreps选择将该系统以一个作家命名是因为，它是“一个用于优化写作的系统”，而且他很喜欢卡夫卡的作品。

https://cwiki.apache.org/confluence/display/KAFKA/Index
https://kafka.apache.org/downloads
http://www.10tiao.com/html/46/201711/2650998759/1.html

Confluent 公司

- Confluent 的产品叫作 Confluent Platform。这个产品的核心是 Kafka，分为三个版本：
- Confluent Open Source（Kafka增强版，Rest代理，语言支持，hadoop、AWS S3、JDBC等的连接的支持，Schema Registry）
- Confluent Enterprise （Confluent Control Center 非开源产品，对整个产品进行管理的控制中心，最主要的功能对这个 Kafka 里面各个生产者和消费者的性能监控）
- Confluent Cloud（Confluent Enterprise 的云端托管服务，它增加了一个叫作云端管理控制台的组件）

Confluent 的基本做法和 Cloudera 很像，主要的产品开源，但是控制中心这样的东西不开源，只有买了企业版才能够享受到。而两者不同的地方主要在于，Confluent 同时提供了云端服务的版本。加上 Confluent 有基于 S3 的连接，这使得从亚马逊 AWS 读写数据都非常方便。
和 Cloudera 是 Hadoop 的集成商不同，Confluent 主要还是围绕着不同数据源之间数据的交换这个任务而生的服务。Kafka 在整个开源产品里面是一个非常特殊的存在，它没有什么竞争对手，又是各大企业的刚需，它在脱离了整个 Hadoop 生态圈以后依然非常有价值。
http://mp.weixin.qq.com/s/5WesHzHrtuMGkXKKqugc9Q

目录

- Kafka基本介绍
- **常用消息系统简单对比**
- Kafka架构简介
- Kafka简单实践
- Kafka工作原理

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

为何使用消息系统

- 解耦
- 冗余
- 扩展性
- 灵活性 & 峰值处理能力
- 可恢复性
- 顺序保证
- 缓冲
- 异步通信

冗余：有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。许多消息队列所采用的”插入-获取-删除"范式中，在把一个消息从队列中删除之前，需要你的处理系统明确的指出该消息已经被处理完毕，从而确保你的数据被安全的保存直到你使用完毕。
扩展性：因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的，只要另外增加处理过程即可。
http://www.infoq.com/cn/articles/kafka-analysis-part-1

常用Message Queue

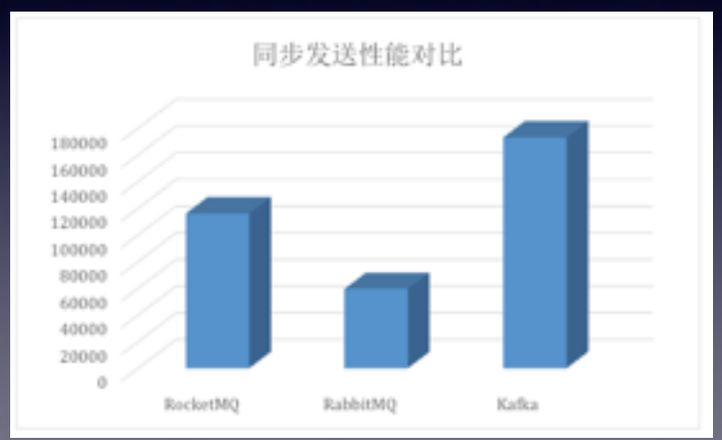
- RabbitMQ
- Redis
- ZeroMQ
- ActiveMQ
- Kafka
- RocketMQ

Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。
Redis 虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能，所以完全可以当做一个轻量级的队列服务来使用。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。
ZeroMQ（用C语言写的）号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZeroMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。但是ZeroMQ仅提供非持久性的队列，也就是说如果宕机，数据将会丢失。其中，Twitter的Storm 0.9.0以前的版本中默认使用ZeroMQ作为数据流的传输（Storm从0.9版本开始同时支持ZeroMQ和Netty作为传输模块，默认用Netty，原因：ZMQ不易部署，无法限制内存，Storm无法从ZMQ获取信息）。

http://www.infoq.com/cn/articles/kafka-analysis-part-1
http://blog.csdn.net/gobravery/article/details/71123425

Kafka、RabbitMQ、RocketMQ 消息发送性能对比

- 测试结论： 在服务端处理同步发送的性能上，Kafka>RocketMQ>RabbitMQ。



Kafka的吞吐量高达17.3w/s，不愧是高吞吐量消息中间件的行业老大。这主要取决于它的队列模式保证了写磁盘的过程是线性IO。
RocketMQ也表现不俗，吞吐量在11.6w/s，磁盘IO %util已接近100%。RocketMQ的消息写入内存后即返回ack，由单独的线程专门做刷盘的操作，所有的消息均是顺序写文件。
RabbitMQ的吞吐量5.95w/s，CPU资源消耗较高。它支持AMQP协议，实现非常重量级，为了保证消息的可靠性在吞吐量上做了取舍。我们还做了RabbitMQ在消息持久化场景下的性能测试，吞吐量在2.6w/s左右。

http://jm.taobao.org/2016/04/01/kafka-vs-rabbitmq-vs-rocketmq-message-send-performance/

Kafka VS RabibitMQ

- Kafka和RabbitMQ在行业认可、服务支持、可靠性、可维护性、兼容性、易用性等方面各有特色
- Kafka优势：开源许可证、产品活跃度、性能、安全性、可扩展性等
- Kafka劣势：仅在功能上略少于RabbitMQ，但是已经具备了主要的功能
- 综上，建议企业在应用过程中优先选择Kafka。

评测项	评测结果
开源许可证	Kafka > RabbitMQ
行业认可度	Kafka = RabbitMQ
产品活跃度	Kafka > RabbitMQ
服务支持	Kafka = RabbitMQ
功能	Kafka < RabbitMQ
性能	Kafka > RabbitMQ
安全性	Kafka > RabbitMQ
可扩展性	Kafka > RabbitMQ
可靠性	Kafka = RabbitMQ
其他特性	Kafka = RabbitMQ

总体来说，分布式消息中间件Kafka和RabbitMQ在行业认可、服务支持、可靠性、可维护性、兼容性、易用性等方面各有特色。
Kafka在开源许可证、产品活跃度、性能、安全性、可扩展性等方面优于RabbitMQ，Kafka采用的许可证更宽松，活跃度更高，性能远高于RabbitMQ，在安全性和可扩展性方面能够提供更好的保障。
Kafka仅在功能上略少于RabbitMQ，但是已经具备了主要的功能。
综合上述所有评测结果，建议企业在应用过程中优先选择Kafka。

http://www.infoq.com/cn/articles/evaluation-distributed-messaging-middleware?utm_source=articles_about_Kafka&utm_medium=link&utm_campaign=Kafka

http://jm.taobao.org/2016/04/01/kafka-vs-rabbitmq-vs-rocketmq-message-send-performance/
https://yq.aliyun.com/articles/73165

Kafka VS RocketMQ

- 都支持：严格的消息顺序、消息回溯（RocketMQ支持按照时间回溯）
- 性能对比：Kafka > RocketMQ（单机TPS百万 VS 7万）
- 消息堆积能力：Kafka > RocketMQ
- Kafka不支持，RocketMQ支持：定时消息、消息查询（根据消息标识 / 内容）、消息轨迹、代理端消息过滤
- 消息并行度：顺序消费方式两者相同，并行度和分区数一致；RocketMQ还支持的乱序方式，并行度取决于Consumer的线程数，可以比前者更高
- 其他：单机支持的队列数，数据可靠性，消息投递实时性，消费失败重试，分布式事务消息，开源社区活跃度, 开发语言友好性(RocketMQ用Java编写)

“理论上Kafka要比RocketMQ的堆积能力更强，不过RocketMQ单机也可以支持亿级的消息堆积能力，我们认为这个堆积能力已经完全可以满足业务需求。”
http://jm.taobao.org/2016/03/24/rmq-vs-kafka/
https://yq.aliyun.com/articles/73165
https://fdx321.github.io/2018/01/03/Kafka-Vs-RocketMQ/

Kafka VS 微信开源PhxQueue

- http://mp.weixin.qq.com/s/YFMFCijamQvz_O-MPv5yfA
- 微信开源PhxQueue

“理论上Kafka要比RocketMQ的堆积能力更强，不过RocketMQ单机也可以支持亿级的消息堆积能力，我们认为这个堆积能力已经完全可以满足业务需求。”
http://jm.taobao.org/2016/03/24/rmq-vs-kafka/
https://yq.aliyun.com/articles/73165
https://fdx321.github.io/2018/01/03/Kafka-Vs-RocketMQ/

目录

- Kafka基本介绍
- 常用消息系统简单对比
- **Kafka架构简介**
- Kafka简单实践
- Kafka工作原理

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

Kafka相关术语

- Broker 实现数据存储的服务器，Kafka集群包含一个或多个服务器
- Topic 用来对消息进行分类，每个进入到Kafka的信息都会被放到一个Topic下
- Partition 每个Topic中的消息会被分为若干个Partition，以提高消息的处理效率
- Producer 消息的生产者，负责发布消息到Kafka broker
- Consumer 消息的消费者，向Kafka broker读取消息的客户端
- Consumer Group 消息的消费群组

物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处。
Parition是物理上的概念，每个Topic包含一个或多个Partition。
每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）。

Kafka拓扑结构

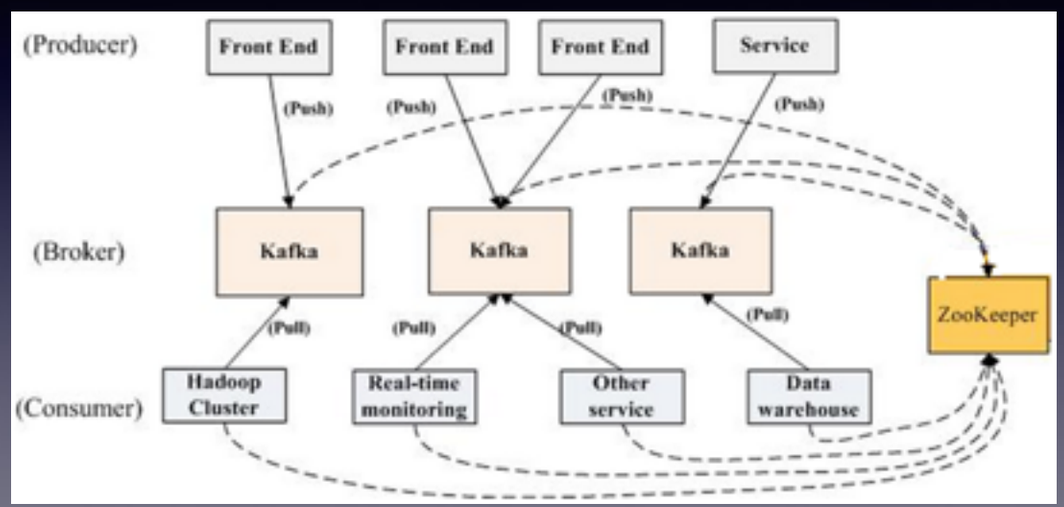
- 一个典型的Kafka集群中包含：若干Producer，若干broker，若干Consumer Group，以及一个Zookeeper集群。

1，一个典型的Kafka集群中包含：若干Producer（可以是web前端产生的Page View，或者是服务器日志，系统CPU、Memory等），
若干broker（Kafka支持水平扩展，一般broker数量越多，集群吞吐率越高），若干Consumer Group，以及一个Zookeeper集群。
2，Kafka通过Zookeeper管理集群配置，选举leader，以及在Consumer Group发生变化时进行rebalance。
3，Producer使用push模式将消息发布到broker，Consumer使用pull模式从broker订阅并消费消息。

push模式很难适应消费速率不同的消费者，因为消息发送速率是由broker决定的。而pull模式则可以根据Consumer的消费能力以适当的速率消费消息。对于Kafka而言，pull模式更合适。pull模式可简化broker的设计，Consumer可自主控制消费消息的速率，同时Consumer可以自己控制消费方式——即可批量消费也可逐条消费，同时还能选择不同的提交方式从而实现不同的传输语义。

http://www.infoq.com/cn/articles/depth-interpretation-of-kafka-data-reliability?utm_source=articles_about_Kafka&utm_medium=link&utm_campaign=Kafka

Kafka拓扑结构



Kafka高效地处理实时流式数据，可以实现与Storm、HBase和Spark的集成。作为群集部署到多台服务器上，Kafka处理它所有的发布和订阅消息系统使用了四个API，即生产者API、消费者API、Stream API和Connector API。它能够传递大规模流式消息，自带容错功能。

Kafka有四个主要API：

生产者API：支持应用程序发布Record流。

消费者API：支持应用程序订阅Topic和处理Record流。

Stream API：将输入流转换为输出流，并产生结果。

Connector API：执行可重用的生产者和消费者API，可将Topic链接到现有应用程序。

http://www.infoq.com/cn/articles/kafka-analysis-part-1

目录

- Kafka基本介绍
- 常用消息系统简单对比
- Kafka架构简介
- **Kafka简单实践**
- Kafka工作原理

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

Kafka简单实践

- Kafka从零开始部署、使用
- 各语言SDK介绍
- 如何使用Ruby的SDK（以Poseidon为例）
- 业界Kafka实践实例

Quickstart

- https://kafka.apache.org/quickstart
- 下载，解压缩
- 启动服务：Zookeeper 和 Kafka server
- 创建Topic
- 启动生产者，发送消息
- 启动消费者，消费消息
- 配置并启动一个多节点（multi-broker）的kafka集群
- 用Kafka Connect来导入 / 导出数据
- 用Kafka Streams来处理数据

https://kafka.apache.org/quickstart

各语言SDK

- https://cwiki.apache.org/confluence/display/KAFKA/Clients
- Java, Scala, C/C++, Python, Go, Erlang, .NET, Clojure, Ruby, Node.js, Proxy, Perl, stdin/stdout, PHP, Rust, Storm, Scala DSL, Swift

Ruby SDK

- ruby-kafka（542 stars）， Kafka Version: 0.9.x, 0.10.x, 0.11， https://github.com/zendesk/ruby-kafka
- Karafka（570 stars）， Kafka Version: 0.9.x, 0.10.x, 0.11.x，https://github.com/karafka/karafka
- Poseidon（259 stars），Kafka Version: 0.8.x，https://github.com/bpot/poseidon
- 其他：Racecar，DeliveryBoy，jruby-kafka，kafka-rb

Racecar – A simple framework for writing Kafka consumers in Ruby that integrates nicely with Rails. Based on ruby-kafka.
poseidon_cluster 基于 poseidon

实践-Ruby SDK

Sending messages to Kafka

```
require 'poseidon'

producer = Poseidon::Producer.new(["localhost:9092"], "my_test_producer")

messages = []
messages << Poseidon::MessageToSend.new("topic1", "value1")
messages << Poseidon::MessageToSend.new("topic2", "value2")
producer.send_messages(messages)
```

More detailed Poseidon::Producer documentation.

演示！

https://github.com/bpot/poseidon

https://github.com/zendesk/ruby-kafka

实践-Ruby SDK

Fetching messages from Kafka

```ruby
require 'poseidon'

consumer = Poseidon::PartitionConsumer.new("my_test_consumer", "localhost", 9092,
                                           "topic1", 0, :earliest_offset)

loop do
  messages = consumer.fetch
  messages.each do |m|
    puts m.value
  end
end
```

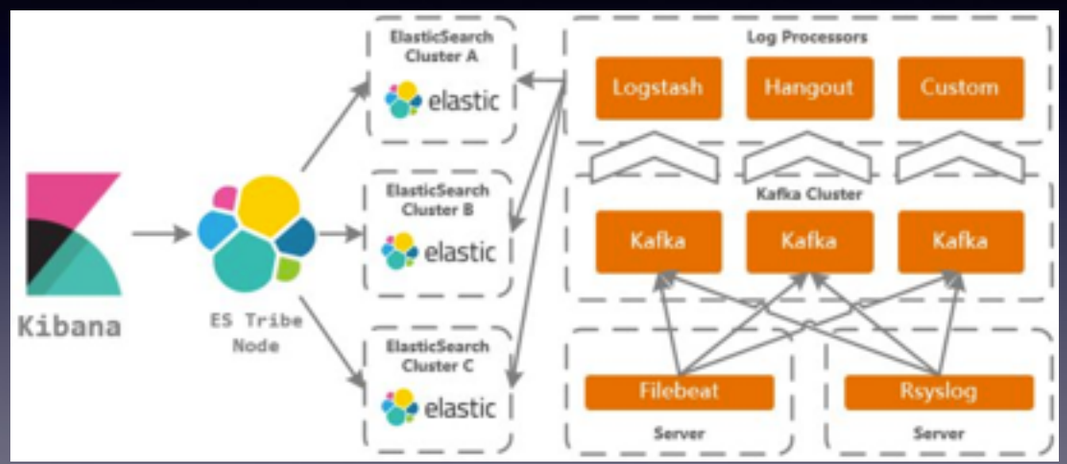
More detailed Poseidon::PartitionConsumer documentation.

演示！

https://github.com/bpot/poseidon

https://github.com/zendesk/ruby-kafka

实践-ELK日志系统

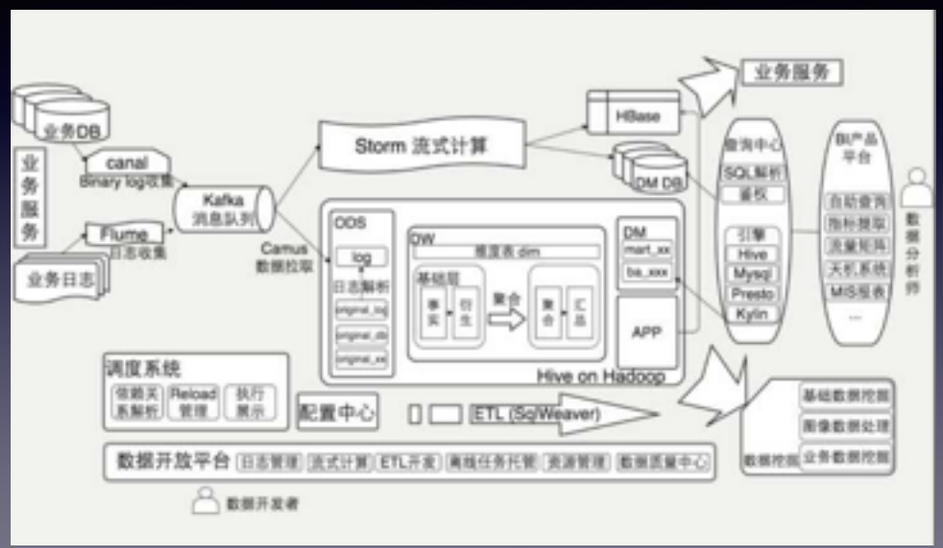


ELK日志系统：服务器上写日志文件 —> Logstash采集发送到Kafka —> 中间件消费Kafka消息，将日志采集到Elasticsearch —> 用Kibana查看日志

ELK：Elasticsearch + Logstash + Kibana

日志收集：filebeat，logstash，flume等

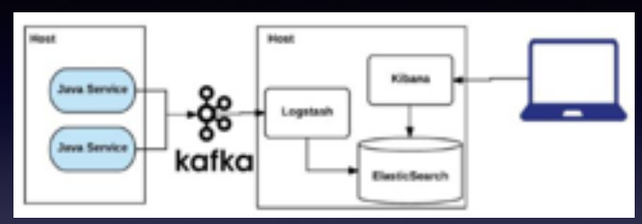
实践-美团数据平台



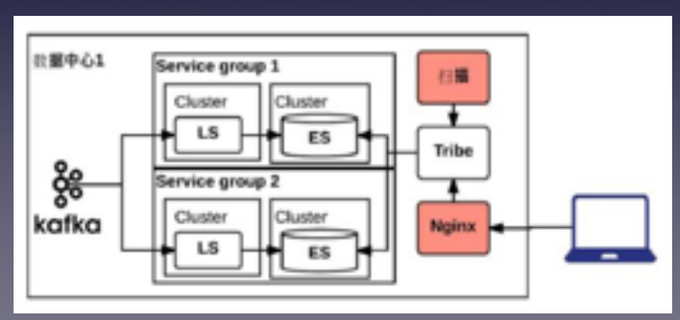
将上图左上角扩大来看，首先是数据接入与流式计算，电商系统产生数据分两个场景，一个是追加型的日志型数据，另外是关系型数据的维度数据。对于前一种是使用Flume比较标准化的大家都在用的日志收集系统，最近使用了阿里开源的Canal，之后有三个下游，所有的流式数据都是走Kafka这套流走的。
https://zhuanlan.zhihu.com/p/26359613

实践－LinkedIn实时日志分析系统

- LinkedIn日志系统演进V2



- LinkedIn日志系统演进现状



v2：基于Kafka和ElasticSearch；服务直接将日志写到Kafka；将Logstash和ElasticSearch分开运行；
现状：按照业务功能拆分ELK cluster；扫描关键字＋访问日志定期审计；

现在生产系统里的状态，500多个服务，有20多个针对不同业务模块的ELK集群，1000+服务器，主要都是Elasticsearch。1分钟之内生产系统发生的log我们这边就可以搜索，所有的log保留7到14天。现在大概有500亿的索引文档，500到800T，之前测试时推到1500到2000T都是可以工作的。

http://mp.weixin.qq.com/s/4dkaOWtEw-weLBI73A0JzQ （作者：李虓 xiāo）

其他

- Kafka 到底是什么？ Kafka 是一个中央式的流处理平台，支持消息的发布、消费、传输和存储，以及消息的计算和消息的处理。
- Kafka Connect （JDBC, HDFS,S3, Elasticsearch, FileStream）
- Kafka Streams
- https://docs.confluent.io/current/connect/connect-jdbc/docs/source_connector.html
- http://mp.weixin.qq.com/s/lmt7tm5AF_6DwSA--3bXRA

Kafka 不仅仅是一个订阅消息系统，同时也是一个大规模的流数据平台，那么它提供了什么呢？第一，提供订阅和发布消息；第二，提供一个缓存的流数据存储平台；第三，提供流数据的处理平台。

Kafka Connect 已经有 40 个不同规模的 Connect，包括从 JDBC 到 HDFS、一直到 MYSQL，以及所有可以想到的第三方系统，用户可以简单地把数据从第三方系统导入和导出 Kafka。

目录

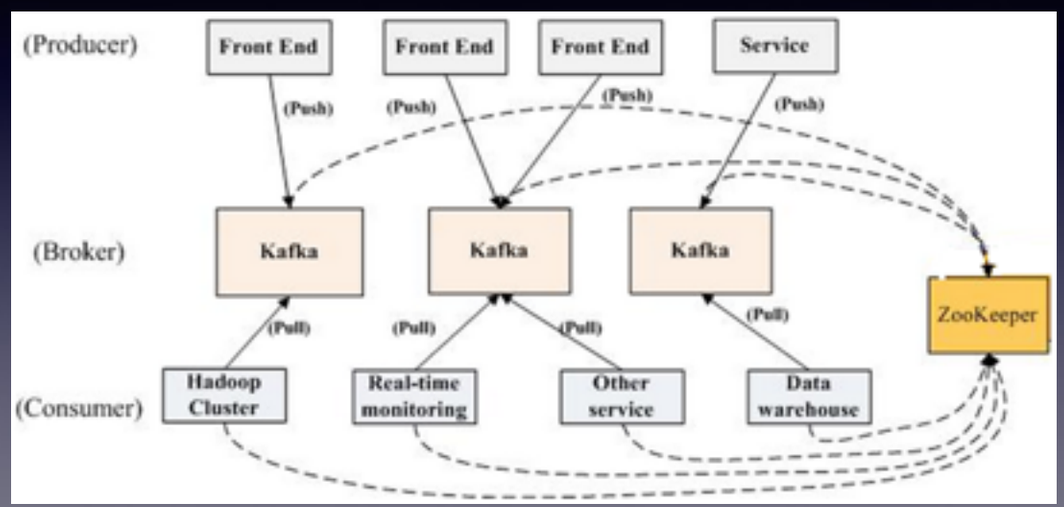
- Kafka基本介绍
- 常用消息系统简单对比
- Kafka架构简介
- Kafka简单实践
- **Kafka工作原理**

目标：让大家对Kafka有基本认识，了解其基本用法、架构和原理，便于后续深入使用或学习。
简单对比！简单实践！

Kafka工作原理

- Kafka拓扑结构
- Topic
- Partition
- Producer
- Consumer
- Kafka高性能关键技术

Kafka拓扑结构

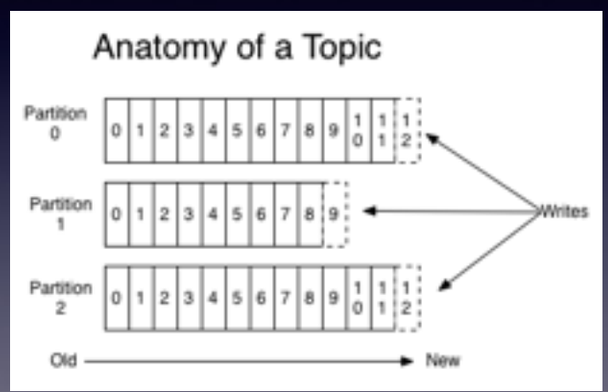


回顾Kafka拓扑结构，原理解释结合该表来看。

http://www.infoq.com/cn/articles/kafka-analysis-part-1

Topic & Partition

- Topic在逻辑上可以当做一个queue
- 每条消息都必须指定它的Topic
- 物理上把Topic分成一个或多个Partition
- Partition是最小并发粒度，和吞吐率线性相关



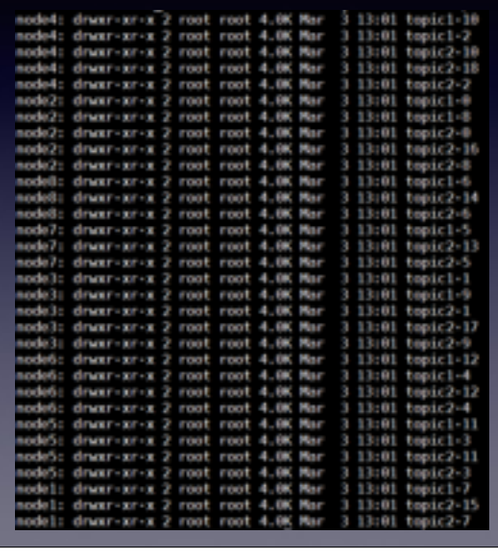
Topic：在逻辑上可以被认为是一个queue，每条消息都必须指定它的Topic
Partition：为了使得Kafka的吞吐率可以线性提高，物理上把Topic分成一个或多个Partition。每个Partition在物理上对应一个文件夹，该文件夹下存储这个Partition的所有消息和索引文件。Partition是最小并发粒度。

Topic： 可以简单理解为必须指明把这条消息放进哪个queue里
The replication factor controls how many servers will replicate each message that is written. If you have a replication factor of 3 then up to 2 servers can fail before you will lose access to your data. We recommend you use a replication factor of 2 or 3 so that you can transparently bounce machines without interrupting data consumption.

Topic & Partition

若创建topic1和topic2两个topic，且分别有13个和19个partition，则整个集群上会相应会生成共32个文件夹，如图所示。

（所用集群共8个节点，此处topic1和topic2 replication-factor均为1）



每个Partition物理上又分成多个segment
因为每条消息都被append到该Partition中，属于顺序写磁盘，因此效率非常高（经验证，顺序写磁盘效率比随机写内存还要高，这是Kafka高吞吐率的一个很重要的保证）。

对于传统的message queue而言，一般会删除已经被消费的消息，而Kafka集群会保留所有的消息，无论其被消费与否。当然，因为磁盘限制，不可能永久保留所有数据（实际上也没必要），因此Kafka提供两种策略删除旧数据。一是基于时间，二是基于Partition文件大小。例如可以通过配置$KAFKA_HOME/config/server.properties，让Kafka删除一周前的数据，也可在Partition文件超过1GB时删除旧数据。

http://www.infoq.com/cn/articles/depth-interpretation-of-kafka-data-reliability?utm_source=articles_about_Kafka&utm_medium=link&utm_campaign=Kafka

The replication factor controls how many servers will replicate each message that is written. If you have a replication factor of 3 then up to 2 servers can fail before you will lose access to your data. We recommend you use a replication factor of 2 or 3 so that you can transparently bounce machines without interrupting data consumption.

Producer

- 每条消息包含3个部分：key，value，timestamp
- Producer根据key和Partition机制来判断将该消息发送到哪个Partition
- 批量发送消息
- acks设置（0，1，-1）

Producer发送消息到broker时，会根据Paritition机制选择将其存储到哪一个Partition。在发送一条消息时，可以指定这条消息的key，Producer根据这个key和Partition机制来判断应该将这条消息发送到哪个Parition。Paritition机制可以通过指定Producer的paritition. class这一参数来指定，该class必须实现kafka.producer.Partitioner接口。
批量的方式推送数据：将消息在内存中累积到一定数量后作为一个batch发送请求。
producer发送消息后响应，其中有个acks，设为0可以得到最大的吞吐量；设为1可以得到1个broker确认；设为－1则要求得到所有broker确认，可以得到最高的可靠性保证。

如果Partition机制设置合理，所有消息可以均匀分布到不同的Partition里，这样就实现了负载均衡。
如果一个Topic对应一个文件，那这个文件所在的机器I/O将会成为这个Topic的性能瓶颈，而有了Partition后，不同的消息可以并行写入不同broker的不同Partition里，极大的提高了吞吐率。可以在$KAFKA_HOME/config/server.properties中通过配置项num.partitions来指定新建Topic的默认Partition数量，也可在创建Topic时通过参数指定，同时也可以在Topic创建之后通过Kafka提供的工具修改。

Partition机制

- 本例中如果key可以被解析为整数则将对应的整数与Partition总数取余，该消息会被发送到该数对应的Partition

```
import kafka.producer.Partitioner;
import kafka.utils.VerifiableProperties;

public class JasonPartitioner<T> implements Partitioner {

    public JasonPartitioner(VerifiableProperties verifiableProperties) {}

    @Override
    public int partition(Object key, int numPartitions) {
        try {
            int partitionNum = Integer.parseInt((String) key);
            return Math.abs(Integer.parseInt((String) key) % numPartitions);
        } catch (Exception e) {
            return Math.abs(key.hashCode() % numPartitions);
        }
    }
}
```

如果Partition机制设置合理，所有消息可以均匀分布到不同的Partition里，这样就实现了负载均衡。如果一个Topic对应一个文件，那这个文件所在的机器I/O将会成为这个Topic的性能瓶颈，而有了Partition后，不同的消息可以并行写入不同broker的不同Partition里，极大的提高了吞吐率。可以在$KAFKA_HOME/config/server.properties中通过配置项num.partitions来指定新建Topic的默认Partition数量，也可在创建Topic时通过参数指定，同时也可以在Topic创建之后通过Kafka提供的工具修改。

Partition机制

- 如果将上例中的类作为partition.class，并通过如下代码发送20条消息（key分别为0，1，2，3）至topic3（包含4个Partition）。
- 则key相同的消息会被发送并存储到同一个partition里，而且key的序号正好和Partition序号相同。（Partition序号从0开始，本例中的key也从0开始）。

```
public void sendMessage() throws InterruptedException{
    for(int i = 1; i <= 5; i++){
        List messageList = new ArrayList<KeyedMessage<String, String>>();
        for(int j = 0; j < 4; j++) {
            messageList.add(new KeyedMessage<String, String>(
                "topic3", j+"", "The " + i + " message for key " + j));
        }
        producer.send(messageList);
    }
    producer.close();
}
```

https://github.com/apache/kafka/blob/trunk/core/src/main/scala/kafka/producer/KeyedMessage.scala
https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/producer/ProducerRecord.java

https://cwiki.apache.org/confluence/display/KAFKA/Index

Partition机制

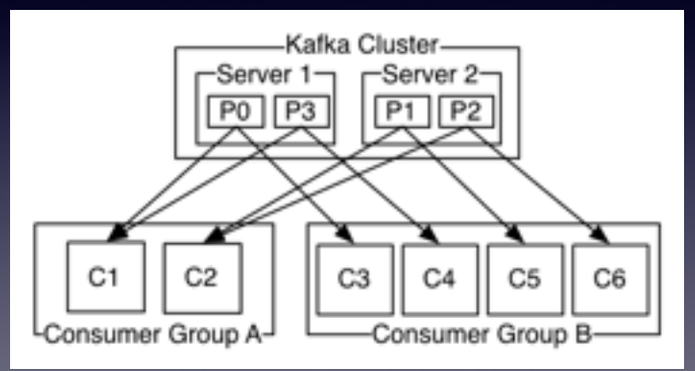
- 本例中如果key可以被解析为整数则将对应的整数与Partition总数取余，该消息会被发送到该数对应的Partition

https://github.com/apache/kafka/blob/trunk/core/src/main/scala/kafka/producer/KeyedMessage.scala
https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/producer/ProducerRecord.java

https://cwiki.apache.org/confluence/display/KAFKA/Index

Consumer

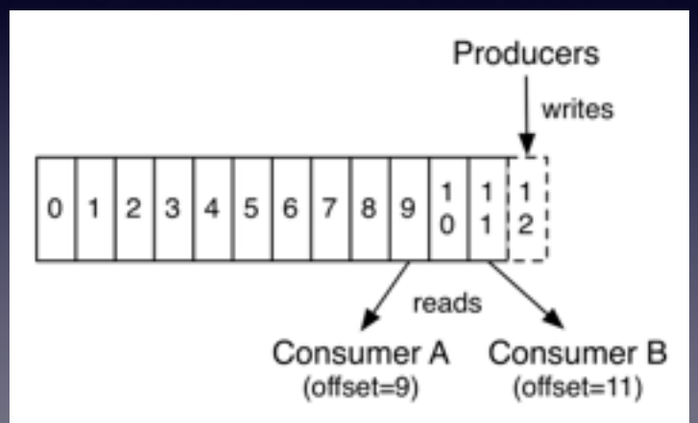
- 广播和单播
- 同组内consumer的数量不应该超过partition数量



这是Kafka用来实现一个Topic消息的广播（发给所有的Consumer）和单播（发给某一个Consumer）的手段。一个Topic可以对应多个Consumer Group。如果需要实现广播，只要每个Consumer有一个独立的Group就可以了。要实现单播只要所有的Consumer在同一个Group里。用Consumer Group还可以将Consumer进行自由的分组而不需要多次发送消息到不同的Topic。

Consumer

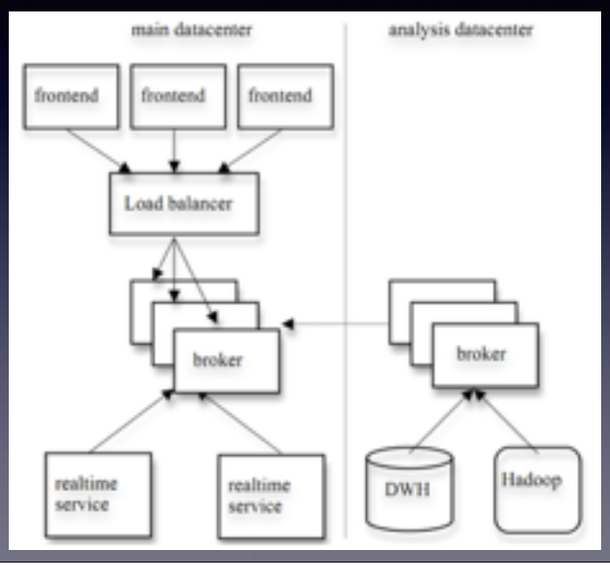
- offset 指向partition中下一个要被消费的消息位置



consumers用同一个组名，则kafka相当于队列消息服务；用不同组名，则kafka相当于广播服务。
Broker给每个consumer group记录offset ，指向partition中下一个要被消费的消息位置；但是consumer本身可以重设offset
consumer的数量应该不超过partition数量，因为一个partition只能被同组的一个consumer消费，如果consumer的数量大于设置的partition的数量，就会有consumer消费不到数据。

关于如何设置partition值需要考虑的因素。一个partition只能被一个消费者消费（一个消费者可以同时消费多个partition），因此，如果设置的partition的数量小于consumer的数量，就会有消费者消费不到数据。所以，推荐partition的数量一定要大于同时运行的consumer的数量。另外一方面，建议partition的数量大于集群broker的数量，这样leader partition就可以均匀的分布在各个broker中，最终使得集群负载均衡。

Consumer



实际上，Kafka的设计理念之一就是同时提供离线处理和实时处理。根据这一特性，可以使用Storm这种实时流处理系统对消息进行实时在线处理，同时使用Hadoop这种批处理系统进行离线处理，还可以同时将数据实时备份到另一个数据中心，只需要保证这三个操作所使用的Consumer属于不同的Consumer Group即可。下图是Kafka在Linkedin的一种简化部署示意图。

Consumer Rebalance

- Consumer Rebalance的算法如下：
- 将目标Topic下的所有Partirtion排序，存于PT
- 对某Consumer Group下所有Consumer排序，存于CG，第i个Consumer记为Ci
- N=size(PT)/size(CG)，向上取整
- 解除Ci对原来分配的Partition的消费权（i从0开始）
- 将第i*N到（i+1）*N-1个Partition分配给Ci

这是基于0.8版本分析的，后续可能有变化。

http://www.infoq.com/cn/articles/kafka-analysis-part-4

Push vs Pull

- 作为一个消息系统，Kafka遵循了传统的方式，选择由Producer向broker push消息并由Consumer从broker pull消息。
- push模式很难适应消费速率不同的消费者，因为消息发送速率是由broker决定的。而pull模式则可以根据Consumer的消费能力以适当的速率消费消息。
- 对于Kafka而言，pull模式更合适。pull模式可简化broker的设计，Consumer可自主控制消费消息的速率，同时Consumer可以自己控制消费方式——即可批量消费也可逐条消费，同时还能选择不同的提交方式从而实现不同的传输语义。

一些logging-centric system，比如Facebook的Scribe和Cloudera的Flume，采用push模式。事实上，push模式和pull模式各有优劣。
push模式的目标是尽可能以最快速度传递消息，但是这样很容易造成Consumer来不及处理消息，典型的表现就是拒绝服务以及网络拥塞。

KafkaStreams

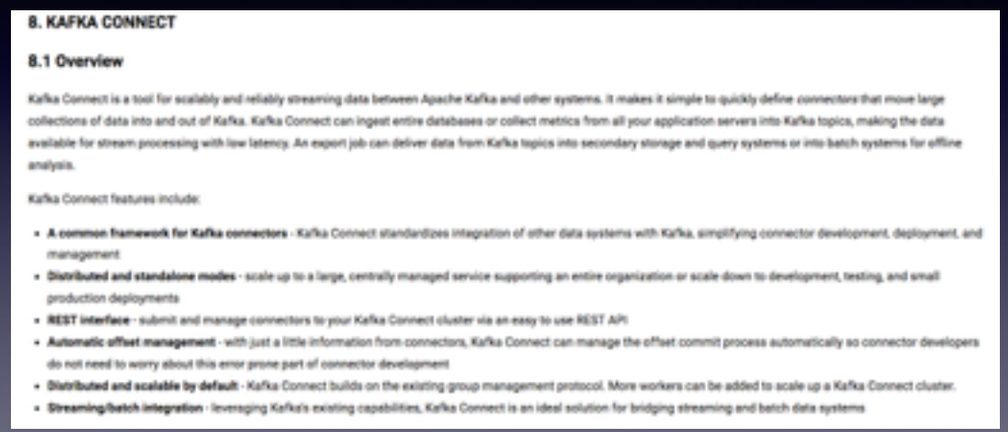
```
Map<String, Object> props = new HashMap<>();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "my-stream-processing-application");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
StreamsConfig config = new StreamsConfig(props);

StreamsBuilder builder = new StreamsBuilder();
builder.<String, String>stream("my-input-topic").mapValues(value -> value.length().toString()).to("my-output-topic");

KafkaStreams streams = new KafkaStreams(builder.build(), config);
streams.start();
```

https://kafka.apache.org/10/javadoc/index.html?org/apache/kafka/streams/KafkaStreams.html
https://kafka.apache.org/documentation/#streamsapi

KAFKA CONNECT



8. KAFKA CONNECT

8.1 Overview

Kafka Connect is a tool for scalably and reliably streaming data between Apache Kafka and other systems. It makes it simple to quickly define *connectors* that move large collections of data into and out of Kafka. Kafka Connect can ingest entire databases or collect metrics from all your application servers into Kafka topics, making the data available for stream processing with low latency. An export job can deliver data from Kafka topics into secondary storage and query systems or into batch systems for offline analysis.

Kafka Connect features include:

- **A common framework for Kafka connectors** - Kafka Connect standardizes integration of other data systems with Kafka, simplifying connector development, deployment, and management
- **Distributed and standalone modes** - scale up to a large, centrally managed service supporting an entire organization or scale down to development, testing, and small production deployments
- **REST interface** - submit and manage connectors to your Kafka Connect cluster via an easy to use REST API
- **Automatic offset management** - with just a little information from connectors, Kafka Connect can manage the offset commit process automatically so connector developers do not need to worry about this error prone part of connector development
- **Distributed and scalable by default** - Kafka Connect builds on the existing group management protocol. More workers can be added to scale up a Kafka Connect cluster.
- **Streaming/batch integration** - leveraging Kafka's existing capabilities, Kafka Connect is an ideal solution for bridging streaming and batch data systems

https://kafka.apache.org/documentation.html#connect

Kafka高性能关键技术

- 利用Partition实现并行处理，Partition是最小并发粒度
- ISR（In-sync Replica）实现可用性与数据一致性的动态平衡
- 顺序写磁盘
- 充分利用Page Cache
- 支持多Disk Drive
- 零拷贝（传统模式下的四次拷贝与四次上下文切换，sendfile和transferTo实现零拷贝）
- 减少网络开销：批处理，数据压缩降低网络负载，高效的序列化方式
- 参考：http://www.infoq.com/cn/articles/kafka-analysis-part-6?utm_source=infoq&utm_campaign=user_page&utm_medium=link

多Consumer消费同一个Topic时，同一条消息只会被同一Consumer Group内的一个Consumer所消费。而数据并非按消息为单位分配，而是以Partition为单位分配，也即同一个Partition的数据只会被一个Consumer所消费（在不考虑Rebalance的前提下）。

如果Consumer的个数多于Partition的个数，那么会有部分Consumer无法消费该Topic的任何数据，也即当Consumer个数超过Partition后，增加Consumer并不能增加并行度。简而言之，Partition个数决定了可能的最大并行度。

CAP理论是指，分布式系统中，一致性、可用性和分区容忍性最多只能同时满足两个。

参考资料

- https://kafka.apache.org/
- https://zh.wikipedia.org/wiki/Kafka
- http://www.infoq.com/cn/articles/kafka-analysis-part-1
- http://www.infoq.com/cn/kafka/
- http://blog.csdn.net/suifeng3051/article/details/48053965
- https://help.aliyun.com/document_detail/52376.html?spm=5176.doc60237.6.606.UF4ZQQ
- https://cwiki.apache.org/confluence/display/KAFKA/Kafka+papers+and+presentations
- http://www.10tiao.com/html/46/201711/2650998759/1.html
- https://cwiki.apache.org/confluence/display/KAFKA/Index
- http://mp.weixin.qq.com/s/lmt7tm5AF_6DwSA--3bXRA

http://www.jasongj.com/2015/04/24/KafkaColumn2/
https://cwiki.apache.org/confluence/display/KAFKA/Index

Thanks!