

Kafka 资料

1. 背景介绍

1.1. 定义

Kafka 是一种高吞吐量的分布式发布订阅消息系统，它是基于 Scala 语言开发的。

1.2. 历史与发展

Kafka 最初被 LinkedIn 设计来处理活动流数据(activity stream data)和系统处理数据(operaitonal data)。活动流数据是指像 page view、用户搜索关键词等等通过用户操作产生的数据，它的常见场景有时间线(time line)即新鲜事提醒、用户浏览量 搜索量排名等等。系统处理数据是服务器性能相关的数据，如 CPU、负载、用户请求数等，它的应用场景多数是为后台服务，如在安全方面，可以监控到恶意攻击服务器的用户，从而做出相应措施，还有监控服务器性能，在其出现问题时即时报警等。

The big picture - Kafka@LinkedIn

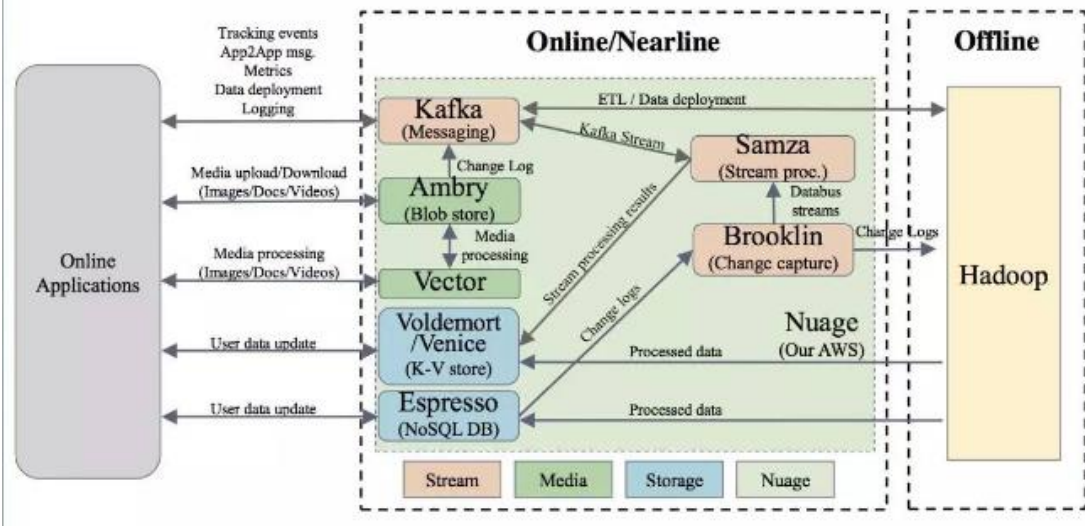


图 1 Kafka 在 LinkedIn 的应用场景

这两种数据都属于日志数据的范畴。常见的日志系统，如 scribe 等都是将这些数据收集起来，然后再通过线下批处理，Hadoop 集群等，获取所需的结果。线下处理的频率一般不会太高，比如一个小时甚至一天一次，这是不适合做实时应用的，如 timeline 这种应用。现有的消息队列系统非常适合这种实时性要求高的场景，但是由于它们都是在内存中维护消息队列，所以处理数据的大小就受到了限制。Kafka 在设计上保留了消息队列的常用操作，而这也使得在其诞生后被越来越广泛的作为一个消息队列使用，而不仅仅局限于处理上面提到的两种数据。



图 2 Kafka 项目创始人 Jay Kreps 带头创立了新公司 Confluent

基于 Kafka 这项技术 Jay Kreps 带头创立了新公司 Confluent，致力于为各行各业的公司提供实时数处理服务解决方案，其他两位成员是 Neha Narkhede 和 Jun Rao。该公司已获 Benchmark、LinkedIn、Data Collective 690 万美金融资。Kreps 将 Kafka 描述为 LinkedIn 的“中枢神经系统”，管理从各个应用程序汇聚到此的信息流，这些数据经过处理后再被分发到各处。不同于传统的企业信息列队系统，Kafka 是以近乎实时的方式处理流经一个公司的所有数据，目前已经为 LinkedIn, Netflix, Uber 和 Verizon 建立了实时信息处理平台。

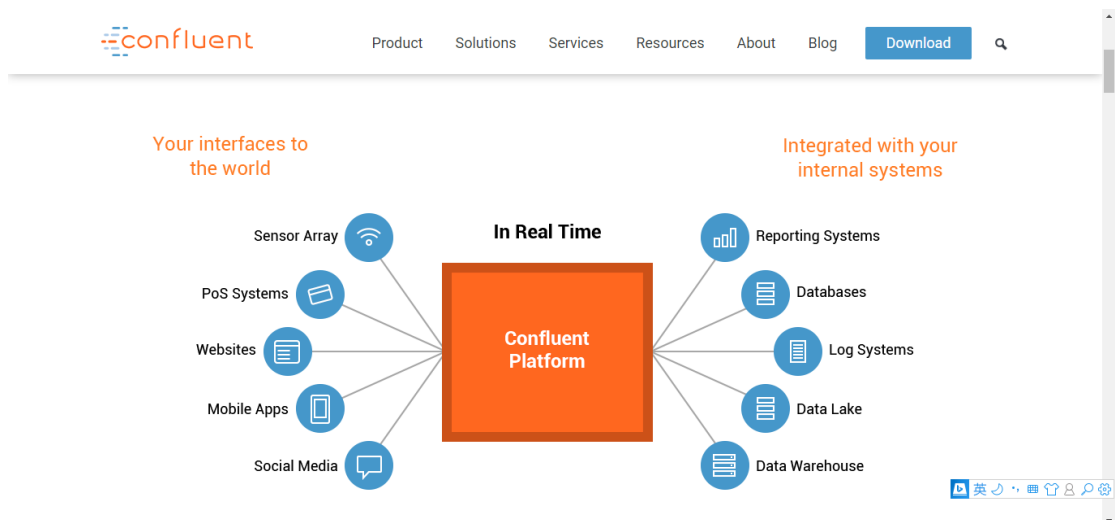


图 3 Confluent 公司网址 <https://www.confluent.io>

上图所示的是 Confluent 公司的核心产品 Confluent Platform。Confluent Platform 很容易建立实时数据管道和流应用，通过将多个来源和位置的数据集成到公司一个中央数据流平台，它的核心就是基于 Kafka。

1.3. 术语

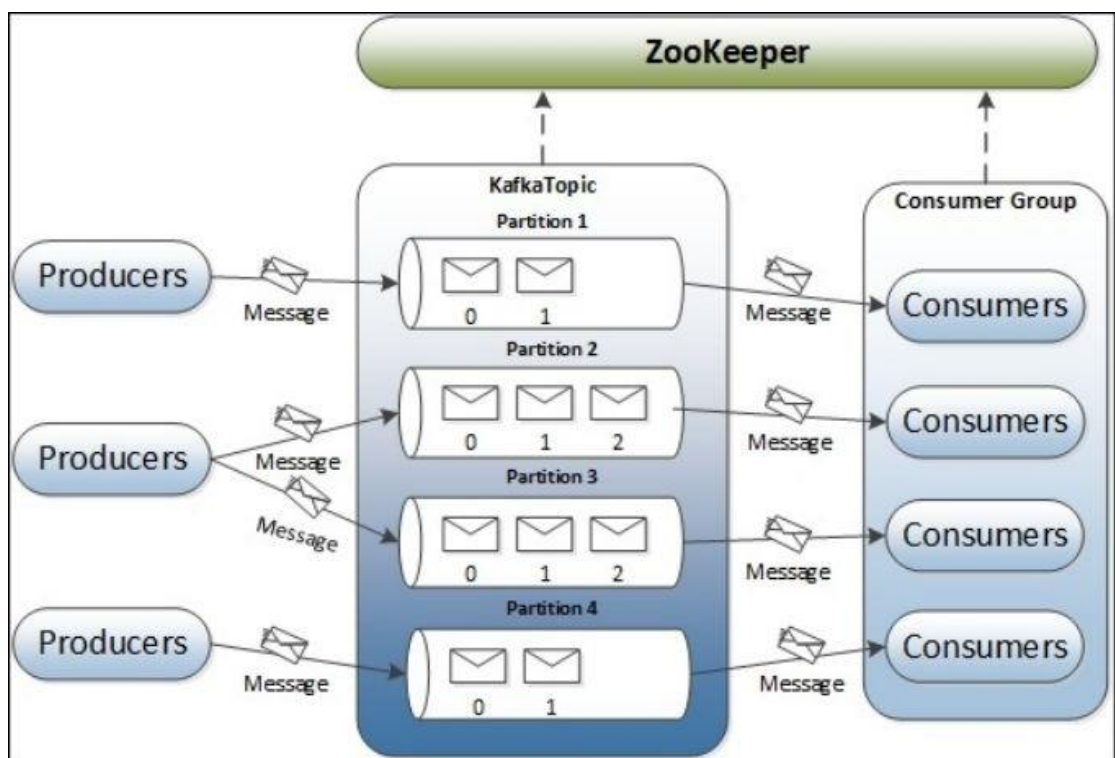


图 4 Kafka 概念示意图

➤ **Broker**

Kafka 集群包含一个或多个服务器，这种服务器被称为 broker

➤ **Topic**

每条发布到 Kafka 集群的消息都有一个类别，这个类别被称为 Topic。（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）

➤ **Partition**

Partition 是物理上的概念，每个 Topic 包含一个或多个 Partition.

➤ **Producer**

负责发布消息到 Kafka broker。

➤ **Consumer**

消息消费者，向 Kafka broker 读取消息的客户端。

➤ **Consumer Group**

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）。

1.4. 特性

Kafka 在设计上有以下几个特色：

- 消息数据通过磁盘线性存取
- 强调吞吐率
- 消费状态由消费者自己维护
- 分布式

1.4.1. 消息数据通过磁盘线性存取

在我们的认识中，硬盘较内存的数据处理速度(读写)是慢很多的，所以基本所有设计数据处理的程序都是尽量使用内存。然而 Kafka 的设计者在经过一番调研测试后，大胆地采用了全硬盘存取消息数据的方案，他们的主要依据是：

- 硬盘在线性读写的情况下性能优异。

有测试表明在一个 6 7200rpm 的 SATA RAID-5 阵列上，线性写可以达到 300MB/Sec，而随机写只有 50KB/Sec。线性读写之所以可以有这么优异的表现与文件系统是分不开的，因为对写操作，操作系统一般会进行缓冲，而对于读操作，操作系统会进行预抓取的缓冲操作，这会极大地提高读取效率。又由于这一层缓存操作是在 OS 级的，也就意味着即便 Kafka 挂掉了重启，缓存也不会失效。

- 减少 JVM 的 GC 触发。

JVM 中的对象会占用除实际数据外的较多空间（如类的信息等等），结构不够紧凑，浪费空间。而当内存中维护的消息数据逐渐增多后，GC 便会被频繁触发，这会极大影响应用的响应速度。因此，舍弃内存，使用磁盘可以减少 GC 触发带来的影响。

1.4.2. 强调吞吐率

Kafka 的设计初衷便是要能处理 TB 级的数据，其更强调的是吞吐率。

- 写方面

Kafka 在启动时会将该文件夹中的所有文件以 channel 形式打开，并且只有最后一个 Kafka 文件以读写形式打开，其他都以只读方式打开，新到的消息都直接 append 到最后一个 Kafka 文件中，这样就实现了顺序写。而前面已经提到，顺序写的性能是极高的，这样写的性能就有了保障。

- 读方面

Kafka 在读方面使用了 `sendfile` 这个高级系统函数，也即 `zero-copy` 技术，这项技术通过减少系统拷贝次数，极大地提高了数据传输的效率。

➤ 压缩

另外，Kafka 还通过多条数据压缩传输、存取的办法来进一步提升吞吐率。

Kafka 支持 GZIP 和 Snappy 压缩算法。

1.4.3. 消费状态由消费者自己维护

Kafka 消息数据的消费状态由消费者自己维护，这样的好处简单来说是：

- 去除了服务端维护消费状态的压力。
- 提升了消费者存储消费状态的自由度，如存储位置，可以存在 zookeeper 数据库或者 HDFS 中，根据消费者自身的需求即可。
- 针对特殊需求，如消息消费失败，消费者可以回滚而重新消费消息。

1.4.4. 分布式

Kafka 的 broker producer 和 consumer 都是可分布的，其实现是通过 zookeeper 来维护集群中这三者的信息，从而实现三者的交互。

1.5. 应用场景

1.5.1. 消息

Kafka 可以很好的接替传统消息系统的工作，而且能提供更好的吞吐率、分区、副本和故障转移等机制，这有利于大规模的消息场景。

1.5.2. 用户跟踪

Kafka 设计的初衷就是对网站的用户的活动进行追踪，将采集到的用户行为

日志发布到离线或实时计算系统，便于进一步做运营分析。

1.5.3. 指标

对各种功能参数和性能参数做汇聚，可以进一步的生成统计报表。

1.5.4. 日志聚合

将多路采集日志的数据流聚合在一起，作为统一处理的基础。

1.5.5. 流处理

Kafka 对消息的处理分为多个阶段。通过利用它提供的 Kafka Stream 接口，可以方便的对数据做进一步的实时处理。

2. 安装部署

这里基于 Ubuntu 14.04 系统进行介绍。

2.1. 下载 Kafka

首先访问 Kafka 官方网站的下载页面 <http://kafka.apache.org/downloads>，如下图所示：

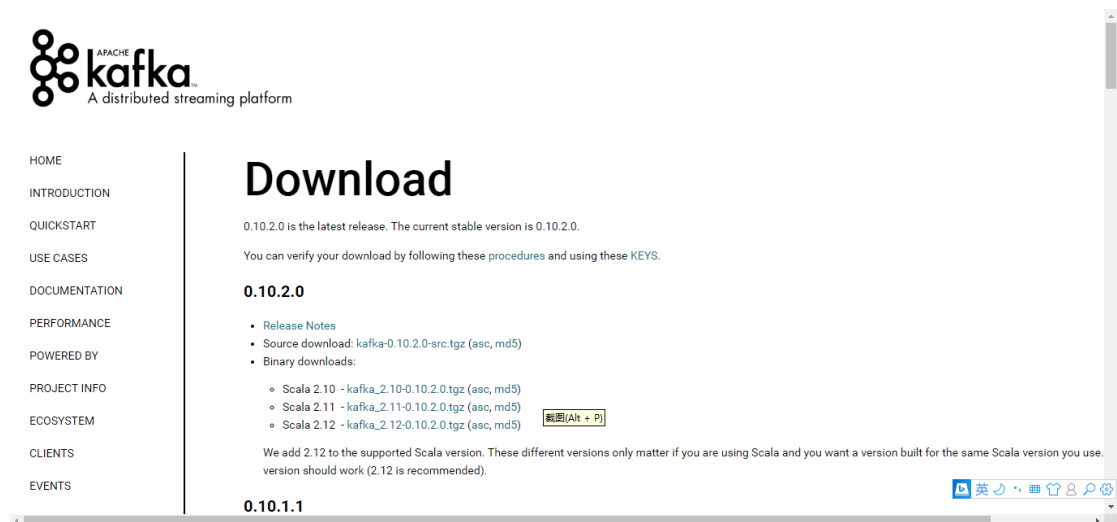


图 5 Kafka 官方下载页面

解开压缩包，命令如下：

```
wget http://apache.fayea.com/kafka/0.10.2.0/kafka_2.12-0.10.2.0.tgz  
  
tar -xzf kafka_2.12-0.10.2.0.tgz
```

2.2. 安装 Scala

因为 Kafka 是基于 Scala 开发的，所以还需要配置 Scala 运行环境，首先下载 Scala 在 Ubuntu 的安装包，命令如下：

```
wget http://downloads.lightbend.com/scala/2.12.1/scala-2.12.1.deb  
  
dpkg -i scala-2.12.1.deb
```

这样 Scala 环境就好了。

2.3. 安装 Zookeeper

Kafka 还需要依赖 Zookeeper 环境，所以还要下载和运行 Zookeeper，这里给出下载的地址：<http://zookeeper.apache.org/releases.html>。下载，解压并运行即可。

以上步骤完成之后，便可以使用 Kafka 的了。

3. 使用介绍

3.1. Kafka 启动

Kafka 启动的命令如下所示：

```
bin/kafka-server-start.sh config/server.properties
```

其中，server.properties 是指定的 Kafka 的配置文件。

3.2. 创建 topic

创建 topic 的命令如下所示：

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

上面的命令创建了一个名称为 test 的 topic。

3.3. 查看所有 topic

通过以下命令，可以查看所有创建的 topic：

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

3.4. 发送消息到 topic

通过以下命令，可以发送一些消息到 topic：

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

This is a message

This is another message

由上所示，发送了两行字符串到 topic 中。

3.5. 启动 consumer

可以通过启动 consumer 来接收消息，如下命令所示：

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --  
topic test --from-beginning
```

This is a message

This is another message

由上所示，consumer 接收到了 topic 中存储的两行字符串消息。

3.6. Kafka broker 集群

➤ 创建配置文件

通过以下命令，创建两个节点的配置文件：

```
cp config/server.properties config/server-1.properties
```

```
cp config/server.properties config/server-2.properties
```

接下来编辑两个配置文件的内容：

✧ config/server-1.properties:

```
broker.id=1
```

```
listeners=PLAINTEXT://:9093
```

```
log.dir=/tmp/kafka-logs-1
```

✧ config/server-2.properties:

```
broker.id=2
```

```
listeners=PLAINTEXT://:9094
```

```
log.dir=/tmp/kafka-logs-2
```

上面的配置项设置了两个 broker 的 id、监听协议及端口、数据存储目录。

上面的两个配置文件设置的 zookeeper 服务器信息应当一致。

➤ 启动 broker

后台启动两个 broker 的命令如下所示：

```
bin/kafka-server-start.sh config/server-1.properties &
```

```
bin/kafka-server-start.sh config/server-2.properties &
```

通过&符号使得 broker 在后台运行。

➤ 创建集群 topic

创建集群的 topic 时要指定复制因子参数 replication-factor，如下：

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --  
replication-factor 3 --partitions 1 --topic my-replicated-topic
```

上面的 topic 的复制因子为 3，名称为 my-replicated-topic。

➤ 浏览集群 topic

如何查看哪个消息在哪个 broker 节点呢？命令如下：

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic  
my-replicated-topic
```

➤ 发送消息到集群 topic

发送消息到集群 topic 的命令如下所示：

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic  
my-replicated-topic
```

my test message 1

my test message 2

由上可知，发送到集群和发送到单个节点的方法没什么不同。

- 启动 consumer 接收消息

启动 consumer 的命令如下所示：

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092  
  
--from-beginning --topic my-replicated-topic
```

my test message 1

my test message 2

由上可知，从接收方式来看，集群和单点也没什么不同。

4. 开发简介

Kafka 提供了两套 API 给 Consumer：

The high level Consumer API

The Simple Consumer API

第一种高度抽象的 Consumer API，它使用起来简单、方便，但是对于某些特殊的需求我们可能要用到第二种更底层的 API。

4.1. The Simple Consumer API

那么先介绍下第二种 API 能够帮助我们做哪些事情。

- 一个消息读取多次
- 在一个处理过程中只消费 Partition 其中的一部分消息
- 添加事务管理机制以保证消息被处理且仅被处理一次

使用 Simple Consumer 有哪些弊端呢？

- 必须在程序中跟踪 offset 值。
- 必须找出指定 Topic Partition 中的 lead broker。
- 必须处理 broker 的变动。

使用 Simple Consumer 的步骤

- 从所有活跃的 broker 中找出哪个是指定 Topic Partition 中的 leader broker
- 找出指定 Topic Partition 中的所有备份 broker
- 构造请求
- 发送请求查询数据
- 处理 leader broker 变更

4.2. High Level Consumer API

High Level Consumer API 围绕着 Consumer Group 这个逻辑概念展开，它屏蔽了每个 Topic 的每个 Partition 的 Offset 管理（自动读取 zookeeper 中该 Consumer group 的 last offset）、Broker 失败转移以及增减 Partition、Consumer 时的负载均衡（当 Partition 和 Consumer 增减时，Kafka 自动进行负载均衡）对于多个 Partition，多个 Consumer 如果 consumer 比 partition 多，是浪费，因为 Kafka 的设计是在一个 partition 上是不允许并发的，所以 consumer 数不要大于 partition 数。

进一步的 API 参考，请访问 <http://kafka.apache.org/documentation/#api>。

5. 监控运维

5.1. Kafka offset monitor

Kafka offset monitor 是一个开源的 Kafka 运维和监控系统，它的官网是 <https://github.com/quantifind/KafkaOffsetMonitor>。它的二进制文件为一个 jar 包，启动命令如下所示：

```
java -cp KafkaOffsetMonitor-assembly-0.2.1.jar \  
com.quantifind.kafka.offsetapp.OffsetGetterWeb \  
--offsetStorage kafka \  
--zk zk-server1,zk-server2 \  
--port 8080 \  
--refresh 10.seconds \  
--retain 2.days
```

可以通过它提供的界面可以监控到 Kafka 的各种状况，如下所示：

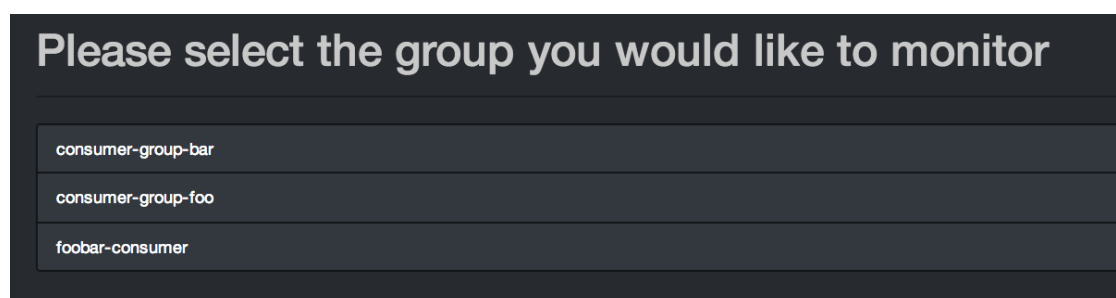


图 6 consumer group 列表

Details for the consumer group consumer-group-1

Groups List / feeedumper-prod

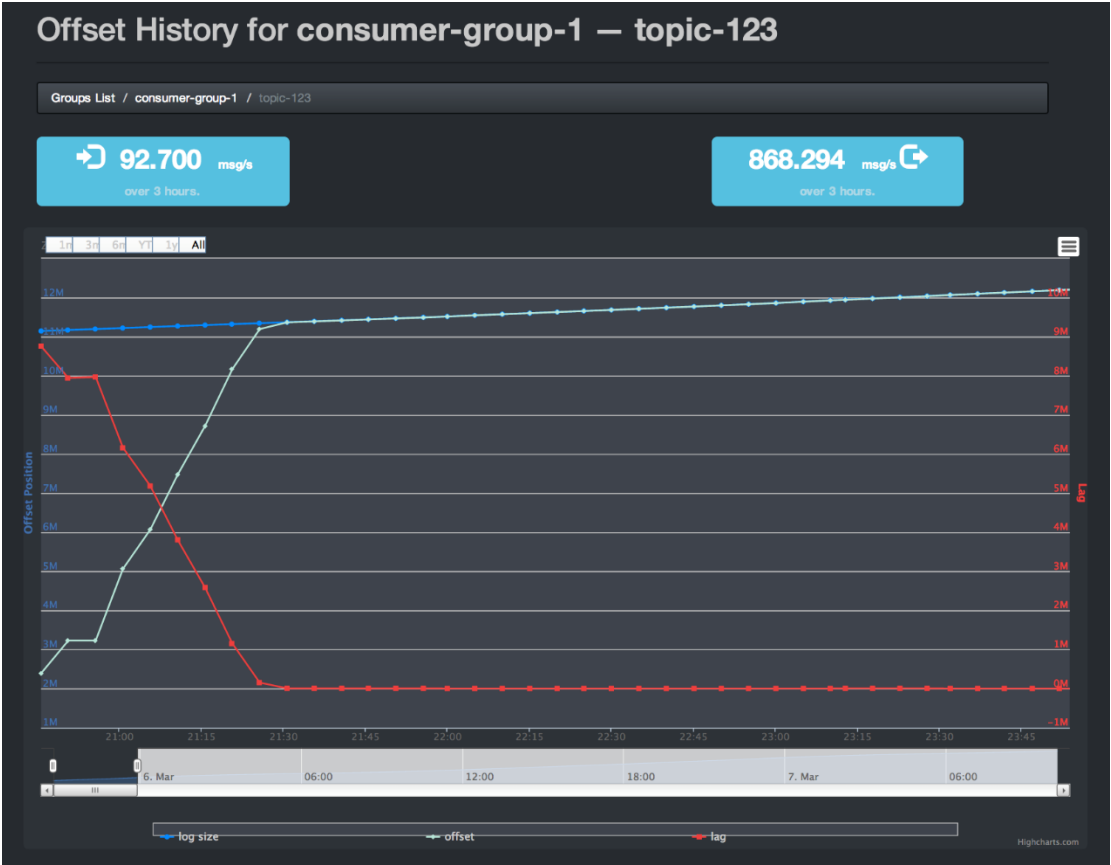
Brokers

id	host	port
2	kafka-server-1	9892
1	kafka-server-2	9892

Consumer Offsets

Topic	Pid	Offset	logSize	Lag	Owner
topic-1		159633	159633	0	
	0	82300	82300	0	foobar-01-1394146768583-295dc0ac-0
	1	77333	77333	0	foobar-01-1394146768583-295dc0ac-1
topic-2		21688442	21688014	1572	
	0	10841598	10842455	857	foobar-01-1394146720830-575fd059-0
	1	10844844	10845559	715	foobar-01-1394146720830-575fd059-1

图 7 topic 明细查看



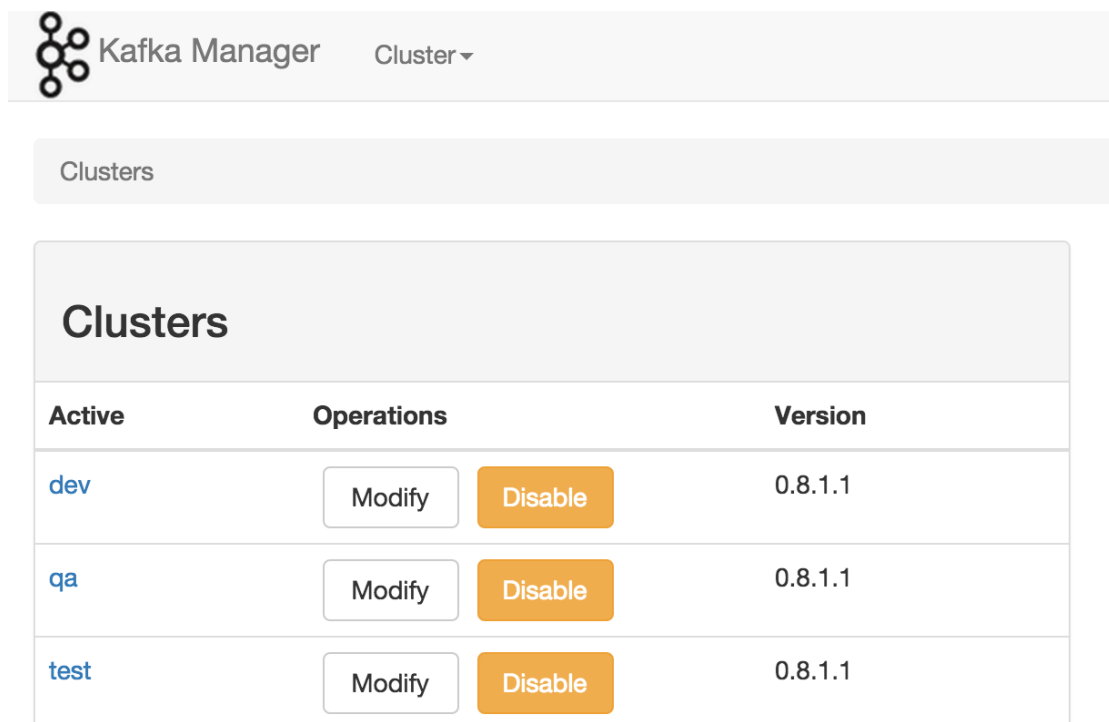
5.2. Kafka manager

Kafka Manager 是 Yahoo 推出的 Kafka 管理工具，支持：

- 管理多个集群
- 轻松检查集群状态 (topics, brokers, replica distribution, partition distribution)
- 执行复制选举
- 生成分区指派，基于集群的状态
- 分区的重新指派

Kafka Manager 是通过 Scala 开发的，可以通过源码方式进行构建部署包，启动命令非常简单，直接执行 bin 目录下的启动程序即可。

Kafka Manager 的相关界面如下所示：



The screenshot shows the Kafka Manager web interface. At the top, there is a header with the Kafka Manager logo, the text "Kafka Manager", and a "Cluster" dropdown menu. Below the header, there is a "Clusters" section. This section contains a table with the following data:

Active	Operations	Version
dev	<button>Modify</button> <button>Disable</button>	0.8.1.1
qa	<button>Modify</button> <button>Disable</button>	0.8.1.1
test	<button>Modify</button> <button>Disable</button>	0.8.1.1

图 9 集群管理界面

Kafka Manager

qa

Cluster

Brokers

Topic

Preferred Replica Election

Reassign Partitions

Clusters

/

qa

/

Topics

/

analytics_content

← analytics_content

Topic Summary

Replication	2
Number of Partitions	8
Total number of Brokers	4
Number of Brokers for Topic	4
Preferred Replicas %	100
Brokers Skewed %	0
Brokers Spread %	100

Generate Partition Assignments

Reassign Partitions

Partitions by Broker

Broker	# of Partitions	Partitions	Skewed?
1	4	(0,5,1,7)	false
2	4	(1,6,2,4)	false
3	4	(5,2,7,3)	false
4	4	(0,6,3,4)	false

Partition Information

Partition	Leader	Replicas	In Sync Replicas	Preferred Leader?	Under Replicated?
0	4	(4,1)	(4,1)	true	false
1	1	(1,2)	(2,1)	true	false

图 10 topic 管理界面

Kafka Manager qa Cluster Brokers Topic Preferred Replica Election Reassign Partitions					
Clusters / qa / Brokers / 1					
← Broker Id 1					
Topic	Replication	Total Partitions	Partitions on Broker	Partitions	Skewed?
mint_content	2	3	2	(2,1)	false
yiral_content	1	4	1	(2)	false
analytics_content	2	8	4	(0,5,1,7)	false
a00analytics_content_string_sampling	2	3	2	(2,1)	false
analytics_content_string_sampling	2	3	2	(2,1)	false
my-topic	2	3	2	(2,0)	false
mint_load	2	3	1	(0)	false
analytics_content_sampling	2	8	4	(5,2,7,3)	false
photon_content	2	8	4	(0,6,3,4)	false
hattrick_content	1	4	1	(3)	false
video_content	2	8	4	(0,5,2,4)	false
testing	3	8	6	(0,1,6,7,3,4)	false

图 11 broker 管理界面

进一步的详细信息，请访问官网 <https://github.com/yahoo/kafka-manager>。

6. 与 ActiveMQ 的比较

6.1. ActiveMQ 简介

ActiveMQ 是 Apache 出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现。它的特点如下所示：

- 多种语言和协议编写客户端。语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。
应用协议: OpenWire,Stomp REST,WS Notification,XMPP,AMQP。
- 完全支持 JMS1.1 和 J2EE 1.4 规范（持久化，XA 消息，事务）。
- 对 Spring 的支持，ActiveMQ 可以很容易内嵌到使用 Spring 的系统里面去，而且也支持 Spring2.0 的特性。
- 通过了常见 J2EE 服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过 JCA 1.5 resource adaptors 的配置，可以让 ActiveMQ 可以自动的部署到任何兼容 J2EE 1.4 商业服务器上。
- 支持多种传送协议: in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA。
- 支持通过 JDBC 和 journal 提供高速的消息持久化。
- 从设计上保证了高性能的集群，客户端-服务器，点对点。
- 支持 Ajax。
- 支持与 Axis 的整合。
- 可以很容易的调用内嵌 JMS provider，进行测试。

综合一句话来说：**血统纯正，功能强大！**

6.2. Kafka 相比 ActiveMQ 的特点

Kafka 看上去是一些“野路子”，并没有纠结于 JMS 规范，剑走偏锋的设计了另一套吞吐非常高的分布式发布-订阅消息系统。

➤ Kafka 的优点

分布式可高可扩展。Kafka 集群可以透明的扩展，增加新的服务器进集群。

高性能：Kafka 的性能大大超过传统的 ActiveMQ、RabbitMQ 等 MQ 实现，尤其是 Kafka 还支持 batch 操作。

容错。Kafka 每个 Partition 的数据都会复制到几台服务器上。当某个 Broker 故障失效时，ZooKeeper 服务将通知生产者和消费者，生产者和消费者转而使用其它 Broker。

➤ Kafka 的不利

重复消息。Kafka 只保证每个消息至少会送达一次，虽然几率很小，但一条消息有可能会被送达多次。

消息乱序。虽然一个 Partition 内部的消息是保证有序的，但是如果一个 Topic 有多个 Partition，Partition 之间的消息送达不保证有序。

复杂性。Kafka 需要 zookeeper 集群的支持，Topic 通常需要维护，部署和维护较一般消息队列成本更高。

7. 搞不定 Scala 怎么办？

Kafka 是基于 Scala 开发的，Scala 是大数据时代的宠儿，不仅开发出了 Kafka，还开发出了 Spark 等热门的大数据开源产品，但是对于熟悉 Java 但

对函数式编程不太感冒的人来说，Scala 还是有些令人“心力憔悴”的。

好消息是 Sohu 的 adyliu 开发了一款纯 java 版本的 Kafka 替代品: Jafka。

Jafka 参考了 Kafka 的设计，用 Java 替代了 Scala 的实现，其官方网站是

<https://github.com/adyliu/jafka>，而且有配套的中文文档资料，有兴趣的朋友

可以访问了解，Jafka 现在也在不断更新中，最新版本是 3.0.1。

8. 参考资料

<http://kafka.apache.org/documentation/>

《Apache Kafka Cookbook》

《Learning Apache Kafka, 2nd Edition》