kafka 权威指南中文版

日期：20170524

翻译者：xingoo、feilong、desehawk、zxczhkzxc

说明：
此文档由 about 云成员组翻译，文档中整理了当前翻译的部分，后期还会继续更新。
about 云将会出更多内容。文档有不当或则错误之处，大家多批评指正。

kafka 权威指南 第一章第 1 节 【中文版】

问题导读

1. 为什么数据管道是数据驱动企业的一个关键组成部分？

2. 发布/订阅消息的概念及其重要性是什么？

第一章

初识 kafka

企业是由数据驱动的。我们获取信息，分析它，处理它，并创造更多的产出。每一个应用程序都会产生数据，无论是日志消息、指标、用户行为、输出报文或者其他类型。每一个字节的数据都有它的作用，传入的数据会告诉接下来需要做什么。为了知道数据的意义，我们需要把数据从它产生的地方，传输到它能够被分析的地方。然后把分析的结果返回到它们能够被执行的地方。 我们越快地做到这一点，我们的系统就能更敏捷，具有更快的响应。我们在移动数据花费的精力越少越少，我们就越能集中处理核心业务。这就是为什么数据管道是数据驱动企业的一个关键组成部分。我们如何移动数据几乎变的和数据本身一样重要。

任何时候科学家们有分歧，那是因为我们没有足够的数据。我们可以在获取怎样的数据上产生统一的观点。我们获取了数据，然后数据解决了问题，要么你是对的，要么我是对的。然后我们就可以进行接下来的工作。

Neil deGrasse Tyson

Publish / Subscribe Messaging 发布/订阅消息

在讨论 kafka 的特性之前，需要理解发布订阅消息的概念及其重要性。发布-订阅消息队列的特征是消息的 sender(publisher)并不直接将 data(message)发送给 receiver， publisher 以某种方法对消息进行分类，而 receiver (subscriber) 会订阅接收特定类别的消息。Pub/Sub 系统通常会有 broker(消息被发布到的中心点)来进行实现，消息将被发送至 broker。

How It Starts 由来

许多发布订阅用例都始于相同的方式： 利用一个简单的消息队列或者进程间的通信。比如，你编写的应用程序需要向某处发送监控信息，你可以从应用程序直接连接到显示监控数据的 dashboard，并通过该连接发送监控数据，如图 1-1 所示

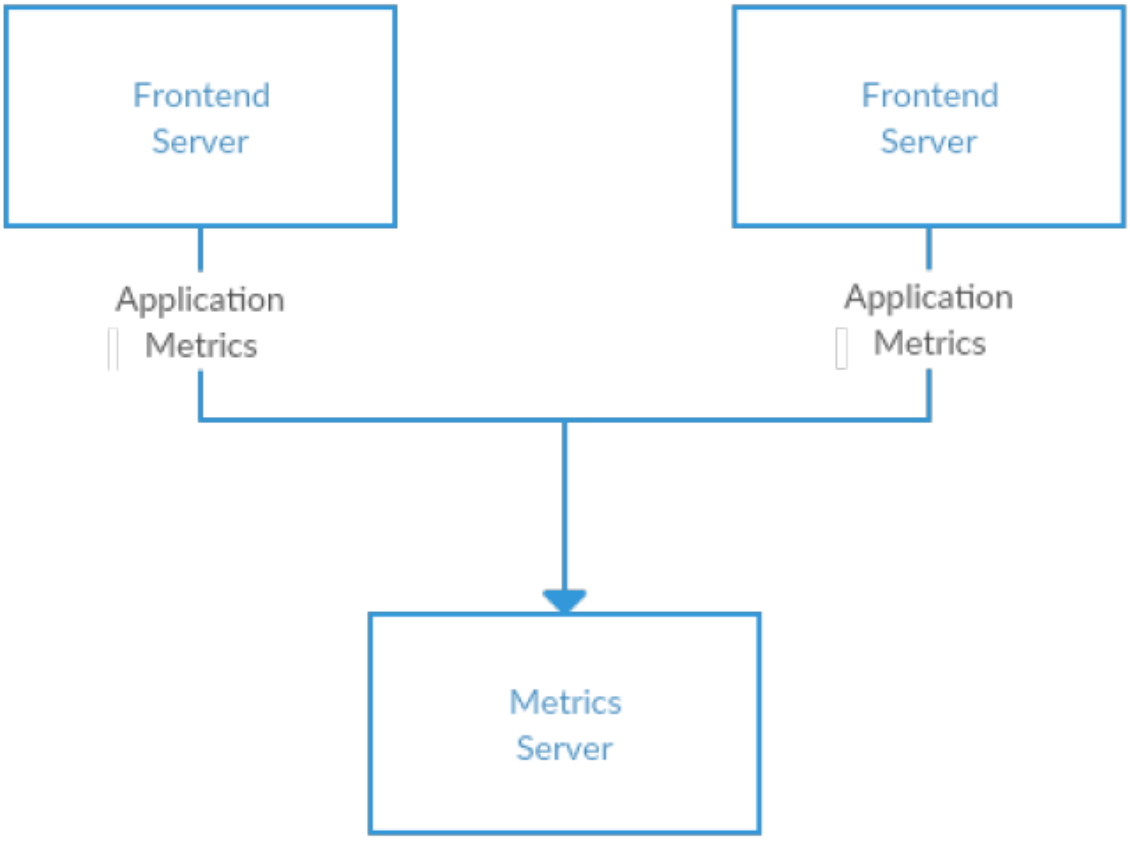


图 1-1 单个直连指标发布者

不久之后，你决定要分析长期的数据，但在 dashboard 中展销效果不是很好。于是你建立了一项新服务，可以接收监控、存储指标并对其进行分析。为了支持此功能，你可以修改应用程序以将指标写入这两个系统(实时监控和长期监控)。到目前为止，有多于三的个应用程序生成指标信息，并且它们都利用相同的连接与这两个服务建立连接。你的同事认为，对服务器进行轮询以用于产生报警也是个不错的想法，因此你在每个应用服务器添加根据请求提供监控指标的服务。过一段时间后，你将会有更多应用程序需要使用这些服务器去获取各个指标并将其用于各种目的。现在的架构看起来可能如图 1-2 所示，连接变得更加复杂难以 跟踪调试。

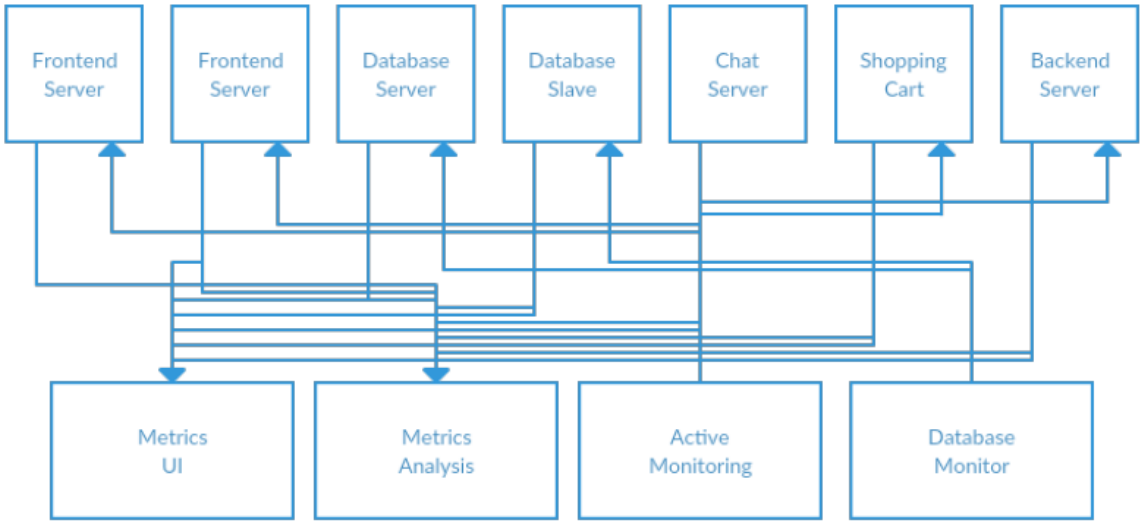


图 1-2 多直连指标发布者

很明显到这里已经欠了很多技术债，你决定偿还一些回来。你使用一个应用程序接收来自所有应用程序的指标，并提供一个服务器来查询任何它们需要的系统的指标。这样就降低了架构的复杂性，类似于图 1-3。恭喜，你已经构建了一个发布订阅消息系统！

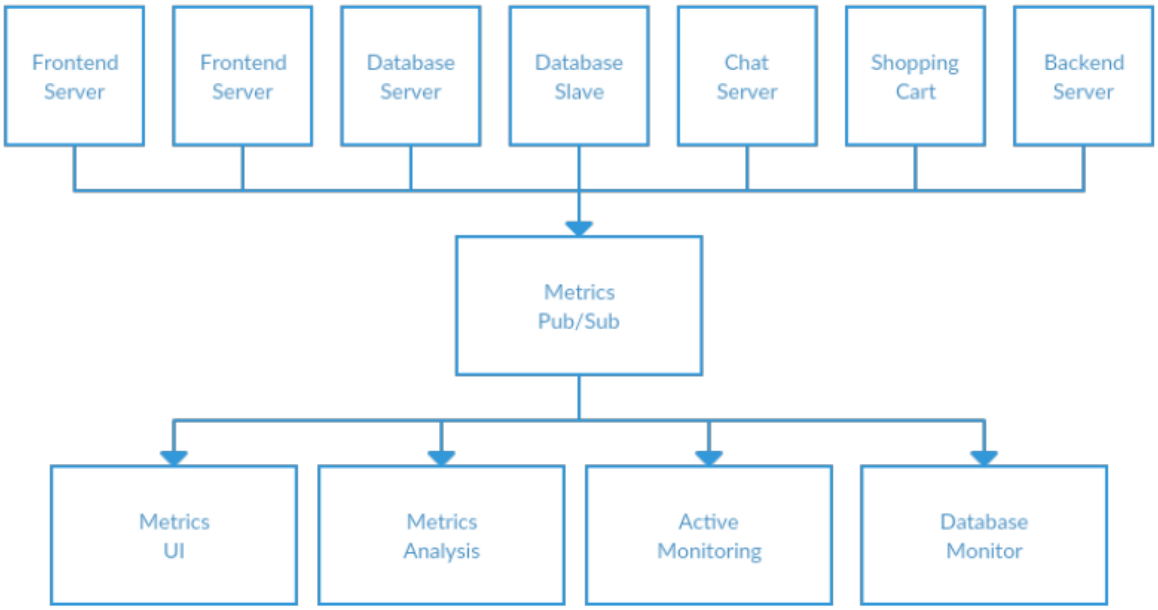


图 1-3 单指标发布订阅系统

Individual Queue Systems　单队列系统

在你与监控这场战争中，你的一个同事也一直在对日志消息进行类似的工作。另一个则是在跟踪前端页面上的用户行为，并向正在从事机器学习的开发人员提供该信息，为管理层创建一些报表。您已经遵循了类似的方法来构建将信息的发布者与该信息的订阅者解耦的系统。　图 1-4 显示了这种结构，具有三个独立的发布/订阅系统。

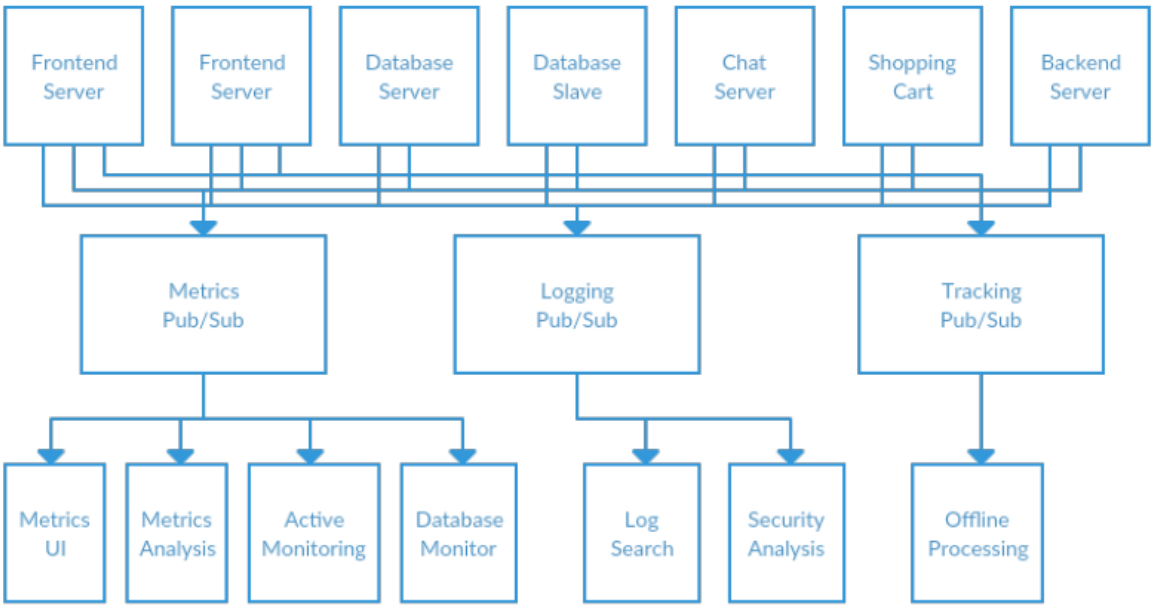


图 1-4 多指标发布订阅系统

这肯定比之前点对点的连接好多了（图 1-2 所示），但是有很多重复工作。你的公司需要维护多个队列系统来存储数据，所有这些都有各自的 bug 和使用限制。

你肯定也知道未来将会有更多需要消息队列的情况。你想要的是一个单一的集中式系统，允许发布通用类型的数据，并且能随着业务增长而增长。

Kafka 权威指南 第一章第 2 节 初识 Kafka

问题导读：

1 kafka 中的消息单元：Message 和 batch
2 kafka 中的消息格式：schema
3 kafka 中的存储模式：Topic 与 partition
4 kafka 中的两种客户端：producer 和 consumer
5 kafka 中的服务核心：broker 与 cluster
6 多集群组成的灾备

翻译内容来自《Kafka 权威指南》

什么是 Kafka

Apache Kafka 是一个基于分布式日志提交机制设计的发布订阅系统。数据在 kafka 中持久化，用户可以随时按需读取。另外数据以分布式的方式存储，提高容错性，易于扩展。

Message 和 Batches

Kafka 中最基本的数据单元是消息 message，如果使用过数据库，那么可以把 Kafka 中的消息理解成数据库里的一条行或者一条记录。消息是由字符数组组成的，kafka 并不关系它内部是什么，索引消息的具体格式与 Kafka 无关。消息可以有一个可选的 key，这个 key 也是个字符数组，与消息一样，对于 kafka 也是透明的。key 用来确定消息写入分区时，进入哪一个分区。最简单的处理方式，就是把 key 作为 hash 串，拥有相同 key 的消息，肯定会进入同一个分区。

为了提高效率，Kafka 以批量的方式写入。一个 batch 就是一组消息的集合，这一组的数据都会进入同一个 topic 和 partition（这个是根据 producer 的配置来定的）。每一个消息都进行一次网络传输会很消耗性能，因此把消息收集到一起，再同时处理就高效的多了。当然，这样会引入更高的延迟以及吞吐量：batch 越大，同一时间处理的消息就越多。batch 通常都会进行压缩，这样在传输以及存储的时候效率都更高一些。

Schemas

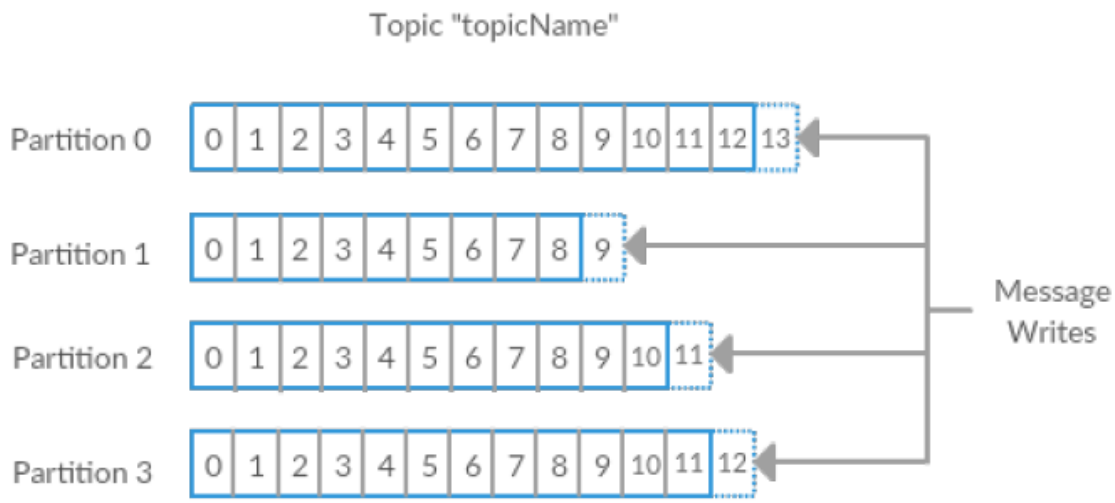
对于 Kafa 来说，消息本身是不透明的，这样就能对消息定义更多容易理解的内容。根据个人的需求不同，消息也会有不同的 schema。比如 JSON 或者

XML 都是对人来说很容易阅读的格式。然后他们在不同的模式版本中间缺乏一些处理的鲁棒性和可扩展性。一些 Kafka 的开发者也倾向于使用 Apache Avro（最开始是用来为 Hadoop 做序列化的），提供了紧凑的序列化格式，在发生变化时，也不需要重新生成代码，具有很强的数据类型和模式，具有很好的向前扩展与向后兼容的能力。

Kafka 中数据是连续的，在数据在写入到同时也可能被读取消费，这样数据的格式就很重要了。如果数据的格式发生变化，消费的应用也需要做出适当的调整。如果事先定义好了数据存储的格式，那么读取数据的时候就不需要做特殊的处理了。

Topics 和 Partitions

消息都是以主题 Topic 的方式组织在一起，Topic 也可以理解成传统数据库里的表，或者文件系统里的一个目录。一个主题由 broker 上的一个或者多个 Partition 分区组成。在 Kafka 中数据是以 Log 的方式存储，一个 partition 就是一个单独的 Log。消息通过追加的方式写入日志文件，读取的时候则是从头开始按照顺序读取。注意，一个主题通常都是由多个分区组成的，每个分区内部保证消息的顺序行，分区之间是不保证顺序的。如果你想要 kafka 中的数据按照时间的先后顺序进行存储，那么可以设置分区数为 1。如下图所示，一个主题由 4 个分区组成，数据都以追加的方式写入这四个文件。分区的方式为 Kafka 提供了良好的扩展性，每个分区都可以放在独立的服务器上，这样就相当于主题可以在多个机器间水平扩展，相对于单独的服务器，性能更好。



在 Kafka 这种数据系统中经常会提起 stream 流这个词，通常流被认为是一个主题中的数据，而忽略分区的概念。这就意味着数据流就是从 producer 到 consumer。在很多框架中，比如 kafka stream,apache samza,storm 在操作实时数据的时候，都是这样理解数据流的。这种操作的模式跟离线系统处理数据的方式不同，如 hadoop，是在某一个固定的时间处理一批的数据。

Producer 和 Consumer

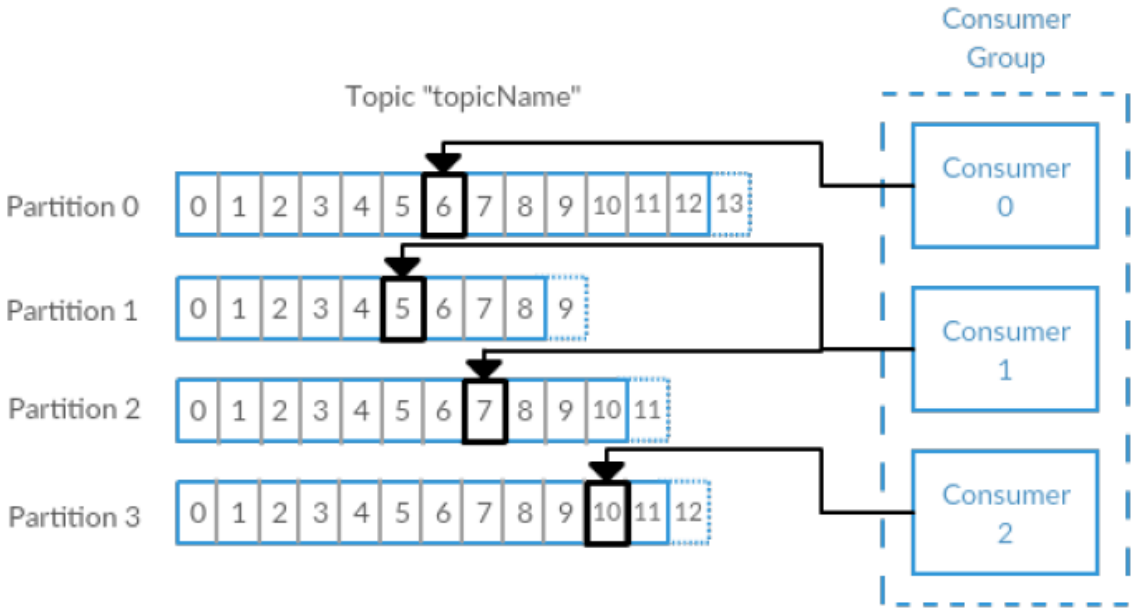
Kafka 中主要有两种使用者：Producer 和 consumer。

Producer 用来创建消息。在发布订阅系统中，他们也被叫做 Publisher 发布者或 writer 写作者。通常情况下，消息都会进入特定的主题。默认情况下，生产者不关系消息到底进入哪个分区，它会自动在多个分区间负载均衡。也有的时候，消息会进入特定的一个分区中。一般都是通过消息的 key 使用哈希的方式确定它进入哪一个分区。这就意味着如果所有的消息都给定相同的 key，那么他们最终会进入同一个分区。生产者也可以使用自定义的分区器，这样消息可以进入特定的分区。

Consumer 读取消息。在发布订阅系统中，也叫做 subscriber 订阅者或者 reader 阅读者。消费者订阅一个或者多个主题，然后按照顺序读取主题中的数据。消费者需要记录已经读取到消息的位置，这个位置也被叫做 offset。每个消息在给定的分区中只有唯一固定的 offset。通过存储最后消费的 Offset，消费者应用在重启或者停止之后，还可以继续从之前的位置读取。保存的机制可以是 zookeeper,或者 kafka 自己。

消费者是以 consumer group 消费者组的方式工作，由一个或者多个消费者组成一个组，共同消费一个 topic。每个分区在同一时间只能由 group 中的一个消费者读取，在下图中，有一个由三个消费者组成的 grouop，有一个消费者读取主题中的两个分区，另外两个分别读取一个分区。某个消费者读取某个分区，也可以叫做某个消费者是某个分区的拥有者。

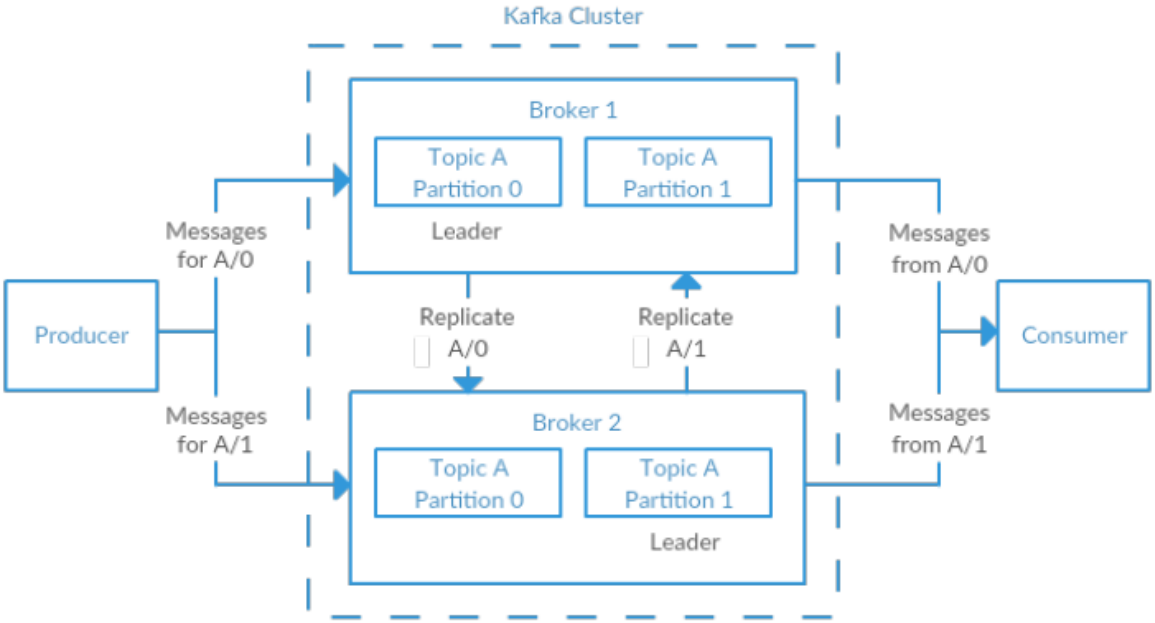
在这种情况下，消费者可以通过水平扩展的方式同时读取大量的消息。另外，如果一个消费者失败了，那么其他的 group 成员会自动负载均衡读取之前失败的消费者读取的分区。



Brokers 和 Clusters

单独的 kafka 服务器也叫做 broker，Broker 从生产者那里获取消息，分配 offset，然后提交存储到磁盘年。他也会提供消费者，让消费者读取分区上的消息，并把存储的消息传给消费者。依赖于一些精简资源，单独的 broker 也可以轻松的支持每秒数千个分区和百万级的消息。

Kafka 的 broker 支持集群模式，在 Broker 组成的集群中，有一个节点也被叫做控制器（是在活跃的节点中自动选择的）。这个 controller 控制器负责管理整个集群的操作，包括分区的分配、失败节点的检测等。一个 partition 只能出现在一个 broker 节点上，并且这个 Broker 也被叫做分区的 leader。一个分区可以分配多个 Broker，这样可以做到多个机器之间备份的效果。这种多机备份在其中一个 broker 失败的时候，可以自动选举出其他的 broker 提供服务。然而，producer 和 consumer 都必须连接 leader 才能正常工作。



Kafka 的一个重要特性就是支持数据的过期删除，数据可以在 Broker 上保留一段时间。Kafka 的 broker 支持针对 topic 设置保存的机制，可以按照大小配置也可以按照时间配置。一旦达到其中的一个限制，可能是时间过期也可能是大小超过配置的数值，那么这部分的数据都会被清除掉。每个 topic 都可以配置它自己的过期配置，因此消息可以按照业务的需要进行持久化保留。比如，一个数据追踪分析的 topic 可以保留几天时间，一些应用的指标信息则只需要保存几个小时。topic 支持日志数据的压缩，这样 kafka 仅仅会保留最后一条日志生成的 key。这在修改日志类型的时候会非常有用。

Multiple Cluster

随着 Kafka 部署环境的演变，有时候需要莉利用多集群的优势。使用多集群的原因如下：

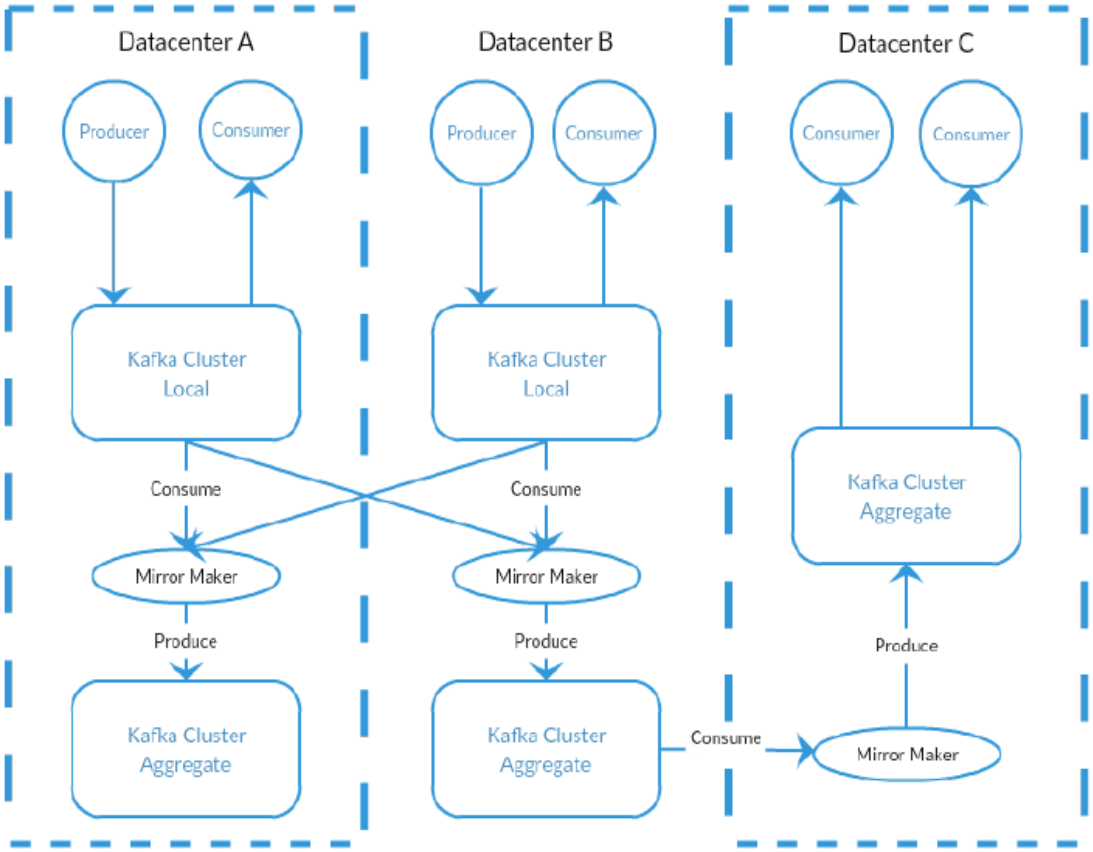
1 不同类型数据的分离

2 安全隔离

3 多数据中心（灾备）

在使用多数据中心的时候，需要很清楚的理解消息是如何在她们之间传递的。一般情况下，用户可以在多个对外提供服务的网址，产生一些前端数据，然后利用 kafka 把他们统一的汇总到一起，进行分析监控告警。这种备份的机制一般都是应用于单个集群，而不是多集群。

Kafka 项目提供了一个叫做 mirro maker 的工具，它支持多机房的数据传输。它其实就是一个基于 queu 连接的 consumer 和 producer。消息从 kafka 中消费，然后传输给另一个集群的 kafka。如下图所示，就是使用 mirror maker 的一个例子，消息在两个集群的本地聚合，然后再传输给另一个集群进行分析。由于 kafka 设计的精简，可以在多机房中实现这种简单的管道处理机制。



kafka 权威指南 第一章第 3 节 为什么选择 kafka

为什么选择 Kafka

对于发布-订阅消息系统有很多选择，是什么促使 Apache Kafka 是一个很好的选择呢？

多个生产者

无论 kafka 多个生产者的客户端正在使用很多 topic 还是同一个 topic ， Kafka 都能够无缝处理好这些生产者。这使得 kafka 成为一个从多个前端系统聚合数据，然后提供一致的数据格式的理想系统. 例如, 一个通过多个微服务向用户提供内容的站点, 可以为统计 page view 而只设立一个 topic, 所有的服务将 page view 以统一的格式写入这个 topic. 消费程序能够以统一的数据格式来接收 page view 数据, 而不需要去协调多个生产者流.

多个消费者

除了多个生产者之外，kafka 也被设计为多个消费者去读取任意的单个消息流而不相互影响；而其他的很多消息队列系统，一旦一个消息被一个客户端消费，那么这个消息就不能被其他客户端消费，这是 kafka 与其他队列不同的地方；同时多个 kafka 消费者也可以选择作为一个组的一部分，来分担一个消息流，确保这整个组，这个消息只被消费一次.

基于硬盘的消息保存

Kafka 不仅能够处理多个消费者，而且能够持久的保存消息这也意味着消费者不一定需要实时的处理数据。消息将按照持久化配置规则存储在硬盘上。这个可以根据每个 topic 进行设置，允许根据不同的消费者的需求不同 设置不同消息流的保存时间不同， 持久化保存意味着一旦消费者来不及消费或者突然出现流量高峰, 而不会有丢失数据的风险.同样也意味着消息可以由 consumer 来负责管理, 比如消费消息掉线了一段时间,不需要担心消息会在 producer 上累积或者消息丢失, consumer 能够从上次停止的地方继续消费.

可扩展性

Kafka 最开始设计的时候就把灵活扩展考虑到里面，使其能够处理任意数量的数据；用户刚开始可以用一台进行验证其相关的理念，然后将其扩展成小的三台 broker 的开发集群，随着数据的增加，甚至扩展为数十台，上百台规模的大集群。扩展可以在集群正常运行的时候进行，对于整个系统的运作没有影响；这也就意味着，对于很多台 broker 的集群，如果一台 broker 有故障，不影响为 client 提供服务.集群如果要同时容忍更多的故障的话, 可以配置更高的 replication factors. Replication 在第 6 章中会详细探讨.

高性能

上面的这些特性使得 Apache Kafka 成为一个能够在高负载的情况下表现出优越性能的发布-订阅消息系统。Producer, consumer 和 broker 都能在大数据流的情况下轻松的扩展. 扩展过程能够在依然提供从生产到消费亚秒级服务的情况下完成.

kafka 权威指南 第一章第 4 节 【中文版】

问题导读

1.kafka 在大数据生态系统中的角色是什么？

2.kafka 有哪些使用场景？

相关内容：

kafka 权威指南 第一章第 1 节 【中文版】

http://www.aboutyun.com/forum.php?mod=viewthread&tid=21648

Kafka 权威指南 —— 第一章第 2 节 初识 Kafka

http://www.aboutyun.com/forum.php?mod=viewthread&tid=21652

kafka 权威指南 第一章第 4 节 【中文版】

http://www.aboutyun.com/forum.php?mod=viewthread&tid=21681

kafka 权威指南- 第一章第 5 节 开始入门 kafka

http://www.aboutyun.com/forum.php?mod=viewthread&tid=21683

The Data Ecosystem 数据生态系统

许多的应用参与到我们所构建的处理数据的环境中。 我们定义了输入–产生数据或者引入数据到系统应用。 我们定义了输出–统计指标, 报告, 或者其他数据产品。 我们创造了回路, 一些组件从系统中读取数据,进行一些处理, 然后将其回写到数据基础设施中, 以便后续其他地方处理。无数的内容,大小,用途不同的数据通过这套流程进行处理。

如图 1-9 所示, Apache Kafka 为这个数据生态系统提供了循环系统。Kafka 传输来自各个基础设施的消息, 为所有的客户端提供一致的接口。 因为引入了消息队列模式, producer 和 consumer 就不再耦合, 之间也没有任何的直接连接。 组件能够随着业务的增加和融合而添加或移除, producer 不需要关心谁在使用数据, 有多少 consumer 在使用数据。

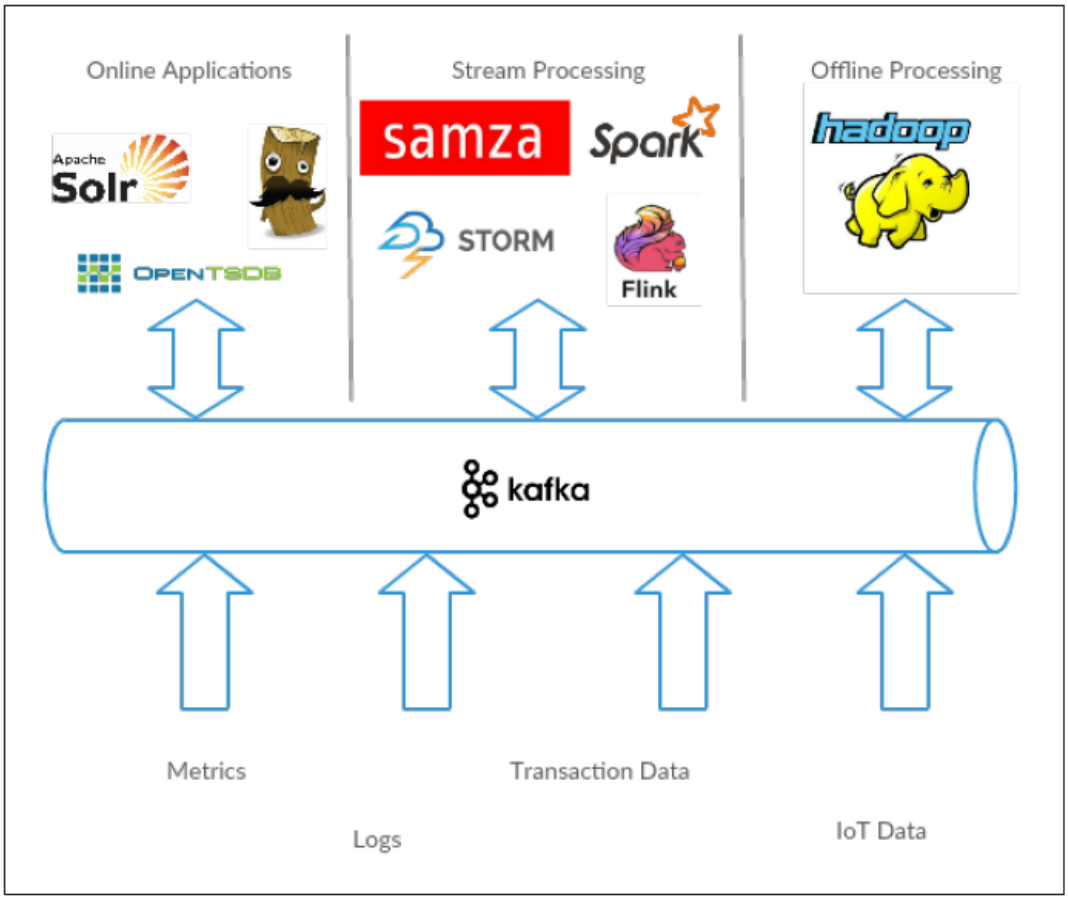


图 1-9 大数据生态系统

Use Cases 用例 Activity Tracking 行为追踪

Kafka 最初的用途是用户行为追踪。 无论用户在前端进行什么交互操作都会产生消息以及行为记录。 这写信息可以是被动信息，比如浏览页面和点击的追踪，也可以是更加复杂的行为，比如为用户资料添加信息。 这些 message 被发布到一个或多个 topic，随后被后端的消费者消费。以这种方式，我们生成报表，为机器学习系统提供数据源，更新搜索结果，以及无数其他的用途。

Messaging 消息传输

kafka 的另外一个基本用途就是消息传输。 当一个应用把需要发送给用户的通知(例如邮件消息)发送给用户。 这些组件可以生产消息，而不需要关注消息格式和消息将如何被发送。 一个通用的应用能够读取所有需要被发送的消息，并且执行格式化，选择如何发送它们。 通过使用一个通用组件，不仅仅是减少了不同系统中的重复功能，而且还能做一些有意思的转换功能，比如聚集多条消息一次发送。

Metrics and Logging 指标和日志

Kafka 同样是理想的处理应用和系统指标、日志的工具。可以用于多个生产者生产同一种类型的消息。应用定期发送它们的操作指标数据到 kafka 的 topic 中，这些数据可以被用于系统监控和报警。 同样也可以被用于像 hadoop 这样的系统进行长周期的分析，比如项目的年增长。日志消息能够以同样的方式进行发送，能够被路由至专门的日志分析系统，比如 Elasticsearch 或者安全分析应用。 kafka 提供了便利，当目标系统需要改变时(比我们要升级日志存储系统)，无需修改前端应用以及日志的聚合方式。

Commit Log 提交日志

Kafka 是基于提交日志的思想来构建的， 自然可以以这种方式来使用 kafka。 数据库更改可以发送到 kafka， 应用可以监控这个流来接收实时更新。 这个 changelog 流同样也可以用于复制数据库更新到远程系统。或者合并多个应用的更新到单一的数据库视图。持久化的保存也为 changelog 提供了缓存， 意味着如果消费应用失败的话, changelog 可以进行事件重放。另外,日志压缩的 topic 还能为日志提供更长的保留时间, 如果对于每个 key 只保留最新一个更新的话。

Stream Processing 流处理

另一方面，它提供了许多类型的应用程序流处理。 可以认为是提供了 Hadoop 的 map/reduce 处理类似的功能,但是 Hadoop 通常依赖于一个大的时间窗口的聚集数据，几小时或几天，然后以批处理的方式处理数据，而流处理以实时的方式处理流。流处理框架允许用户写一个小应用来对 kafka message 进行操作，执行诸如计数，对 message 进行分区让其他应用更有效的处理，或者转换来自多个源的数据。 Stream 处理学习在第 10 章中单独讲解。

kafka 权威指南 第一章第 5 节 Kafka 的起源

问题导读

1 为什么要创造 Kafka 这个项目？

2 为什么起 Kafka 这个名字？

Kafka 最初是 LinkedIn 用来解决数据传输问题的，它可以以高性能的方式处理大量来自不同数据类型的数据，并提供实时处理用户活动、系统指标等数据分析的工作。

Data really powers everything that we do. —— jeff weiner,LinkedIn 的 CEO

LinkedIn 遇到的问题

在本章开头就说过，LInkedIn 最初需要收集各个系统和应用的指标数据进行数据分析，刚开始采用的方式是使用自定义的收集器以及开源框架来存储和提供数据服务。收集的信息数量庞大、内容复杂，除了一些传统的指标，比如 CPU 的使用情况、应用的性能分析等等；还需要收集很多与业务相关联的复杂的监控信息以及提供给某个用户请求的信息。这个系统也遇到了很多的问题，比如指标的采集采用轮询的模式，内部传输大量的指标信息，没有提供系统内部的自服务。系统内部高度耦合，需要人工来设置很多简单的任务，并且系统间各种指标的命名也大不相同。

同时，有一个系统在用户创建活动的时候会产生一些信息，这些信息通过一个前置的 http 服务，以 XML 的数据格式进行传输。这些数据需要先经过解析处理，然后用来做离线分析。这个系统本身很不稳定，经常会崩溃出错。XML 的格式还不是很通用，解析起来非常复杂。如果改变了数据的格式，那么从数据的传输到后面的数据解析处理都需要进行调整。系统经常会因为数据格式的调整而引发崩溃。定位这个问题，需要很长时间，因此根本无法用来做实时服务。

监控和用户活动追踪都无法直接使用该服务，监控的服务太脆弱，数据的格式也不自由，并且轮询的这种机制效果也不好。追踪服务系统对于用户的指标来说太容易崩溃、这种批量的定时任务也无法做到实时处理和告警。并且数据之间的关联也不容易做，比如用户活动与系统应用性能之间的关联分析，但是这种分析还是很有用的。比如用户的操作发生了错误，如果采用定时的批处理，那么有可能需要几个小时之后才能定位到问题。

最开始，LinkedIn 想要在现有的开源解决方案中，寻找一种可以支持大数据的实时服务以及支持水平扩展的方案。最初系统使用的是 ActiveMQ,但是它不支持水平扩展。并且它内部有很多 BUG，LinkedIn 为此也付出了很多的心血来解决这些问题。如果服务器发生崩溃，也会阻塞客户端的请求，从而影响应用服务器的服务性能。这种处理方式主要是为了推动管道处理方式结构。

Birth of Kafka Kafka 的诞生

LinkedIn 由 Jay Kreps 主导，他是 Voldemort（键值存储系统）的主要开发者。最初的团队还包括 Neha Narkhede 以及 Jun Rao。他们一起参与开发了这个即需要实时处理又需要支持水平扩展的消息系统。

这个系统主要的目标是：

- 基于 Push-pull 推拉模式解耦生产者与消费者的关系
- 提供消息的持久化，并且支持多个消费者消费消息
- 消息的吞吐量性能优化
- 允许系统在数据增长的情况下进行水平扩展

最终就设计出了这个具有经典的消息系统接口的发布订阅系统，但是在存储上又很像一款日志数据聚合系统。结合 Apache Avro 提供的消息序列化的特性，系统支持每天处理十亿级的系统指标和用户行为数据。在 2015 年 8 月份的时候，LinkedIn 就有万亿级的消息数据，没天都要处理 1PB 的数据量。

Open Source 开源

Kafka 在 2010 年开源贡献给 GitHub，从一开始它就吸引了大量的使用这的注意，并且在 2011 年成为 Apache 的一个项目，并且 2012 年成为 Apache 的孵化项目。从那时起，大量来自 linkedIn 的员工以及社区的贡献者共同不断改善 Kafka.目前 Kafka 在很多公司都有很多的应用场景。在 2014 年，Jay Kreps，Neha Narkhede，Jun Rap 离开了 LinkedIn，并成立了 Confluent 公司，专门提供 kafka 相关的技术支持。这两家公司以及不断加入的贡献者，持续不断的改进与支撑 kafka，让它成为大数据管道处理的首选。

The Name 命名

Kafka 经常提及的一个问题就是，为什么要取这个名字？Jay Kreps 的回答是这样的：
"我觉得既然 Kafka 是一个支持读写的消息系统，那不如直接用一个作家的名字来命名。我在大学学习过很多的文学课程，特别喜欢 Franz Kafka,因此就给这个开源项目起了这个很响亮的名字。但其实他们本身并没有很直接的关系。"

kafka 权威指南- 第一章第 6 节 开始入门 kafka

我们知道 kafka 通用术语，下面我们继续，设置 kafka，和构建数据管道。下一章，探索 kafka 的安装和配置。包括：运行在合适的硬件及生产时需要考虑的事情。

kafka 权威指南 第二章第 1 节：安装 Kafka 【中文版】

问题导读

1.安装 Kafka 的步骤有哪些，需要哪些准备？

2.kafka 配置完成后 如何验证 Kafka 是否正常运行？

第二章 Installing Kafka

本章介绍怎样运行 Apache Kafka，包括存储 kafka 源数据的 zookeeper 的安装。

同样的，也会覆盖 kakfa 部署所需的基本配置项以及选择运行 broker 的硬件的标准。最后，介绍怎样部署 kafka 集群以及在生成环境上的一些注意点。

First Things First

Choosing an Operating System

Apache Kafka 是一个 Java 程序，可以运行在多种操作系统上，包括：Windows, OS X, Linux 等。本章安装使用 kafka 的步骤都是在使用最广泛的 Linux 系统上执行的。LInux 也是 Kafka 部署建议的操作系统。Kafka 在 Windows 和 OS X 上的安装请点击附录 A。

Installing Java

无论是安装 zookeeper 还是 kafka，都需要一个 Java 运行环境。Java 版本可能是 Java8，也可以是系统提供的版本，或者是直接从 java.com 下载。虽然 zookeeper 和 kafka 在有 java runtime 的版本上就可以运行，但是有完整的 jdk 在开发工具和应用时会更便利。因此，以下的安装步骤假设你已经在 /usr/java/jdk1.8.0_51 下安装了 jdk8。

Installing Zookeeper

Apache Kafka 利用 zookeeper 存储关于 kafka 集群的元数据信息以及消费者信息。因为分布式版 kakfka 自带启动 zookeeper 服务器的脚本，因此并不需要安装分布式版 zookeeper。

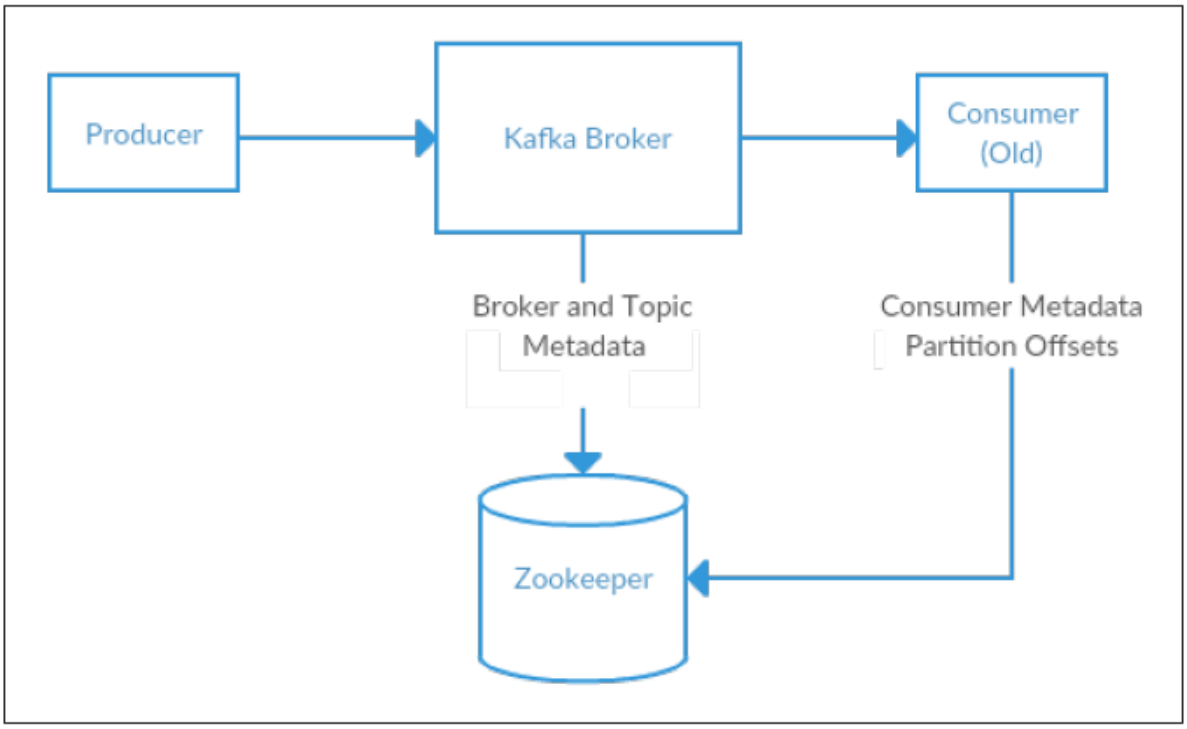


图 2-1 Kafka 和 Zookeeper

Kafka 和 3.4.6 版 zookeeper 稳定结合已广泛测试，从http://mirror.cc.columbia.edu/pu ... keeper-3.4.6.tar.gz下载 3.4.6 版 zookeeper。

Standalone Server

下例是 zookeeper 安装的基本配置，路径为/usr/local/

Zookeeper，数据存储在/var/lib/zookeeper

?

```
# tar -zxf zookeeper-3.4.6.tar.gz
# mv zookeeper-3.4.6 /usr/local/zookeeper
# mkdir -p /var/lib/zookeeper
# cat > /usr/local/zookeeper/conf/zoo.cfg << EOF
> tickTime=2000
> dataDir=/var/lib/zookeeper
> clientPort=2181
> EOF
# export JAVA_HOME=/usr/java/jdk1.8.0_51
# /usr/local/zookeeper/bin/zkServer.sh start
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
#
```

现在你可以验证 zookeeper 是否在单机模式下能够连接到客户端端口，发送四个字符'srvr'

```
# telnet localhost 2181
Trying ::1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.6-1569965, built on 02/20/2014
09:09 GMT
Latency min/avg/max: 0/0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x0
Mode: standalone
Node count: 4
Connection closed by foreign host.
#
```

Zookeeper Ensemble

一个 zookeeper 集群被叫做"ensemble"，因为一致性协议的使用，建议一个 zookeeper 集群使用奇数个服务器（例如：3,5，等），每个都作为集群的一个 quorum 用来返回发往 zookeeper 的请求。这意味着 3 节点的集群，当一个节点失效后仍可以运行。5 个节点的集群，允许 2 个节点失效。

Sizing Your Zookeeper Ensemble

假设 zookeeper 运行在 5 个节点上，为了使配置变成包括节点转换的套件。如果你的套件不能够容许多用于 1 个节点的宕机，这会给做维护工作引入额外的风险。同样不建议一个 zookeeper 套件运行在多余 7 个的节点上，因为集群性能会因一致性的特征而降低。

一个 zookeeper 套件中服务器的配置必须含有一项共通的配置：服务器列表，并且每个服务器都需要在数据目录下有一个 myid 文件用来标识那个服务器的的 id。如果服务器的主机名如下例：zoo1.example.com, zoo2.example.com, and zoo3.example.com，则配置文件可配置为：

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=20
syncLimit=5
server.1=zoo1.example.com:2888:3888
server.2=zoo2.example.com:2888:3888
server.3=zoo3.example.com:2888:3888
```

在这个配置中，initLimit 表示 follower 经过多长时间可以连接上 leader。syncLimit 表示 follower 与 leader 不同步的时间有多长。所有的配置都是以 tickTime 为单位的整数倍，例如这里 initLimit 为 20 * 2000 ms 或 40 s。上边的配置也列出了在套件中的每个服务器。服务器是以 server.X=hostname:peer Port:leader Port 这样格式话的方式定义的，个各参数释义如下：

X：服务器 Id。这个值必须是整数，但不必从 0 开始或顺序的。
Hostname：服务器的主机名或服务器 IP 地址。
peerPort：服务器间进行交互的 TCP 端口。
leaderPort：leader 选举时的 TCP 端口。

客户端只要能够通过 clientPort 连接到套件即可，但是套件中的成员必须两两之间能够通过以上 3 个端口互通。

除了分享的配置文件外，每个服务器在 data Dir 目录下必须有一个 myid 文件。这个文件必须包含有与配置文件一致的服务器 id 号。以上步骤完成后，套件中的服务器就可以启动并两两互通。

Installing a Kafka Broker

Java 和 Zookeeper 配置好后，你就可以准备安装 Apache Kafka。当前版本 kafka 可在http://kafka.apache.org/downloads.html下载。到发稿为止，Kafka 版本是运行在 Scala2.11.0 上的 0.9.0.1。

下例是 kafka 安装在/usr/local/kafka，利用之前配置好的 zookeeper，消息的日志切片会存储在/tmp/kafka-logs:

```
# tar -zxf kafka_2.11-0.9.0.1.tgz
# mv kafka_2.11-0.9.0.1 /usr/local/kafka
# mkdir /tmp/kafka-logs
# export JAVA_HOME=/usr/java/jdk1.8.0_51
# /usr/local/kafka/bin/kafka-server-start.sh -daemon
/usr/local/kafka/config/server.properties
#
```

Kafka 启动后，我们可以通过一下简单的操作：创建 test topic,生产一些消息，消费一些消息来检测 Kafka 是否正常运行。

创建和验证一个 topic:

```
#   /usr/local/kafka/bin/kafka-topics.sh   --create   --
zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic test
Created topic "test".
#    /usr/local/kafka/bin/kafka-topics.sh    --zookeeper
localhost:2181
--describe --topic test
Topic:test PartitionCount:1 ReplicationFactor:1 Configs:
Topic: test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
#
```

生产消息到测试 topic:

```
#   /usr/local/kafka/bin/kafka-console-producer.sh   --
broker-list
localhost:9092 --topic test
Test Message 1
Test Message 2
^D
#
```

从测试 topic 消费消息:

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

```
Test Message 1
Test Message 2
^C
Consumed 2 messages
#
```

kafka 权威指南 第二章第 2 节：安装 kafka Broker

问题导读

1.kafka 安装需要做哪些操作？

2.如何验证 kafka 是否安装成功？

3./tmp/kafka-logs 的作用是什么？

Java 和 zookeeper 配置完毕，你可以安装 kafka.软件下载http://kafka.apache.org/downloads.html。这里下载的版本为 0.9.0.1，Scala 版本 2.11.0.下面的例子安装 kafka 在 /usr/local/kafka，开始之前配置使用 zookeeper server 和在/tmp/kafka-logs 路径下存储消息日志.

[Bash shell] 纯文本查看 复制代码

?

```
1# tar -zxf kafka_2.11-0.9.0.1.tgz
2# mv kafka_2.11-0.9.0.1 /usr/local/kafka
3# mkdir /tmp/kafka-logs
4# export JAVA_HOME=/usr/java/jdk1.8.0_51
5# /usr/local/kafka/bin/kafka-server-start.sh -daemon
6/usr/local/kafka/config/server.properties
7#
```

一旦 Kafka broker 启动，我们可以创建 test topic，来验证是否工作。

创建验证 topic:

[Bash shell] 纯文本查看 复制代码

?

```
1# /usr/local/kafka/bin/kafka-topics.sh --create --
2zookeeper localhost:2181
 --replication-factor 1 --partitions 1 --topic test
```

创建 topic "test".

[Bash shell] 纯文本查看 复制代码

?

```
 #    /usr/local/kafka/bin/kafka-topics.sh    --zookeeper
localhost:2181
--describe --topic test
Topic:test     PartitionCount:1     ReplicationFactor:1
Configs:
Topic: test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
 #
```

在 topic 下面，生产者产生消息

[Bash shell] 纯文本查看 复制代码

?

```
#   /usr/local/kafka/bin/kafka-console-producer.sh   --
broker-list
localhost:9092 --topic test
Test Message 1
Test Message 2
^D
 #
```

在 test topic 下面消费消息

[Bash shell] 纯文本查看 复制代码

?

```
#   /usr/local/kafka/bin/kafka-console-consumer.sh   --
zookeeper
localhost:2181 --topic test --from-beginning
Test Message 1
Test Message 2
^C
Consumed 2 messages
 #
```

kafka 权威指南 第二章第 3 节 Broker 的配置

问题导读：

1 Broker 有哪些基本的配置？

2 Broker 有哪些需要了解的配置？

Borker 配置

前面章节中的例子，用来作为单个节点的服务器示例是足够的，但是如果想要把它应用到生产环境，就远远不够了。在 Kafka 中有很多参数可以控制它的运行和工作。大部分的选项都可以忽略直接使用默认值就好，遇到一些特殊的情况你可以再考虑使用它们。

Broker 的一般配置

有很多参数在部署集群模式时需要引起重视，这些参数都是 broker 最基本的配置，很多参数都需要依据集群的 broker 情况而变化。

broker.id

每个 kafka 的 broker 都需要有一个整型的唯一标识，这个标识通过 broker.id 来设置。默认的情况下，这个数字是 0，但是它可以设置成任何值。需要注意的是，需要保证集群中这个 id 是唯一的。这个值是可以任意填写的，并且可以在必要的时候从 broker 集群中删除。比较好的做法是使用主机名相关的标识来做为 id，比如，你的主机名当中有数字相关的信息，如 hosts1.example.com，host2.example.com，那么这个数字就可以用来作为 broker.id 的值。

port

默认启动 kafka 时，监听的是 TCP 的 9092 端口，端口号可以被任意修改。如果端口号设置为小于 1024，那么 kafka 需要以 root 身份启动。但是并不推荐以 root 身份启动。

zookeeper.connect

这个参数指定了 Zookeeper 所在的地址，它存储了 broker 的元信息。在前一章节的例子 中，Zookeeper 是运行在本机的 2181 端口上，因此这个值被设置成 localhost:2181。这个值可以通过分号设置多个值，每个值的格式都是 hostname:port/path，其中每个部分的含义如下：

- hostname 是 zookeeper 服务器的主机名或者 ip 地址
- port 是服务器监听连接的端口号
- /path 是 kafka 在 zookeeper 上的根目录。如果缺省，会使用根目录。

如果设置了 chroot，但是它又不存在，那么 broker 会在启动的时候直接创建。

PS：为什么使用 Chroot 路径

一个普遍认同的最佳实践就是 kafka 集群使用 chroot 路径，这样 zookeeper 可以与其他应用共享使用，而不会有任何冲突。指定多个 zookeeper 服务器的地址也是比较好的做法，这样当 zookeeper 集群中有节点失败的时候，还可以正常连接其他的节点。

log.dirs

这个参数用于配置 Kafka 保存数据的位置，Kafka 中所有的消息都会存在这个目录下。可以通过逗号来指定多个目录，kafka 会根据最少被使用的原则选择目录分配新的 parition。注意 kafka 在分配 parition 的时候选择的规则不是按照磁盘的空间大小来定的，而是分配的 parition 的个数多小。

num.recovery.thread.per.data.dir

kafka 可以配置一个线程池，线程池的使用场景如下：

- 当正常启动的时候，开启每个 parition 的文档块 segment
- 当失败后重启时，检查 parition 的文档块
- 当关闭 kafka 的时候，清除关闭文档块

默认，每个目录只有一个线程。最好是设置多个线程数，这样在服务器启动或者关闭的时候，都可以并行的进行操作。尤其是当非正常停机后，重启时，如果有大量的分区数，那么启动 broker 将会花费大量的时间。注意，这个参数是针对每个目录的。比如，num.recovery.threads.per.data.dir 设置为 8，如果有 3 个 log.dirs 路径，那么一共会有 24 个线程。

auto.create.topics.enable

在下面场景中，按照默认的配置，如果还没有创建 topic，kafka 会在 broker 上自动创建 topic：

- 当 producer 向一个 topic 中写入消息时
- 当 cosumer 开始从某个 topic 中读取数据时
- 当任何的客户端请求某个 topic 的信息时

在很多场景下，这都会引发莫名其妙的问题。尤其是没有什么办法判断某个 topic 是否存在，因为任何请求都会创建该 topic。如果你想严格的控制 topic 的创建，那么可以设置 auto.create.topics.enable 为 false。

默认的主题配置 Topic Defaults

kafka 集群在创建 topic 的时候会设置一些默认的配置，这些参数包括分区的个数、消息的容错机制，这些信息都可以通过管理员工具以 topic 为单位进行配置。kafka 为我们提供的默认配置，基本也能满足大多数的应用场景了。

PS：使用 per.topic 进行参数的覆盖

在之前的版本中，可以通过 log.retention.hours.per.topic，log.retention.bytes.per.topic， log.segment.bytes.per.topic 等覆盖默认的配置。现在的版本不能这么用了，必须通过管理员工具进行设置。

num.partitions

这个参数用于配置新创建的 topic 有多少个分区，默认是 1 个。注意 partition 的个数只可以被增加，不能被减少。这就意味着如果想要减少主题的分区数，那么就需要重新创建 topic。

在第一章中介绍过，kafka 通过分区来对 topic 进行扩展，因此需要使用分区的个数来做负载均衡，如果新增了 broker，那么就会引发重新负载分配。这并不意味着所有的主题的分区数都需要大于 broker 的数量，因为 kafka 是支持多个主题的，其他的主题会使用其余的 broker。需要注意的是，如果消息的吞吐量很高，那么可以通过设置一个比较大的分区数，来分摊压力。

log.retention.ms

这个参数用于配置 kafka 中消息保存的时间，也可以使用 log.retention.hours，默认这个参数是 168 个小时，即一周。另外，还支持 log.retention.minutes 和 log.retention.ms。这三个参数都会控制删除过期数据的时间，推荐还是使用 log.retention.ms。如果多个同时设置，那么会选择最小的那个。

PS：过期时间和最后修改时间

过期时间是通过每个 log 文件的最后修改时间来定的。在正常的集群操作中，这个时间其实就是 log 段文件关闭的时间，它代表了最后一条消息进入这个文件的时间。然而，如果通过管理员工具，在 brokers 之间移动了分区，那么这个时候会被刷新，就不准确了。这就会导致本该过期删除的文件，被继续保留了。

log.retention.bytes

这个参数也是用来配置消息过期的，它会应用到每个分区，比如，你有一个主题，有 8 个分区，并且设置了 log.retention.bytes 为 1G，那么这个主题总共可以保留 8G 的数据。注意，所有的过期配置都会应用到 patition 粒度，而不是主题粒度。这也意味着，如果增加了主题的分区数，那么主题所能保留的数据也就随之增加了。

PS：通过大小和时间配置数据过期

如果设置了 log.retention.bytes 和 log.retention.ms（或者其他过期时间的配置），只要满足一个条件，消息就会被删除。比如，设置 log.retention.ms 是

86400000(一天)，并且 log.retention.bytes 是 1000000000(1G),那么只要修改时间大于一天或者数据量大于 1G，这部分数据都会被删除。

log.segment.bytes

这个参数用来控制 log 段文件的大小，而不是消息的大小。在 kafka 中，所有的消息都会进入 broker，然后以追加的方式追加到分区当前最新的 segment 段文件中。一旦这个段文件到达了 log.segment.bytes 设置的大小，比如默认的 1G，这个段文件就会被关闭，然后创建一个新的。一旦这个文件被关闭，就可以理解成这个文件已经过期了。这个参数设置的越小，那么关闭文件创建文件的操作就会越频繁，这样也会造成大量的磁盘读写的开销。

通过生产者发送过来的消息的情况可以判断这个值的大小。比如，主题每天接收 100M 的消息，并且 log.segment.bytes 为默认设置，那么 10 天后，这个段文件才会被填满。由于段文件在没有关闭的时候，是不能删除的，log.retention.ms 又是默认的设置，那么这个消息将会在 17 天后，才过期删除。因为 10 天后，段文件才关闭。再过 7 天，这个文件才算真正过期，才能被清除。

PS：根据时间戳追踪 offset

段文件的大小也会影响到消息消费 offset 的操作，因为读取某一时间的 offset 时，kafka 会寻找关闭时间晚于 offset 时间的那个段文件。然后返回 offset 所在的段文件的第一个消息的 offset，然后按照偏移值查询目标的消息。因此越小的段文件，通过时间戳消费 offset 的时候就会越精确。

log.segment.ms

这个参数也可以控制段文件关闭的时间，它定义了经过多长时间段文件会被关闭。跟 log.retention.bytes 和 log.retention.ms 类似，log.segment.ms 和 log.segment.bytes 也不是互斥的。kafka 会在任何一个条件满足时，关闭段文件。默认情况下，是不会设置 Log.segment.ms 的，也就意味着只会通过段文件的大小来关闭文件。

PS：基于时间关闭段文件的磁盘性能需求

当时使用基于时间的段文件限制，对磁盘的要求会很高。这是因为，一般情况下如果段文件大小这个条件不满足，会按照时间限制来关闭文件，此时如果分区数很多，主题很多，将会有大量的段文件同时关闭，同时创建。

message.max.bytes

这个参数用于限制生产者消息的大小，默认是 1000000，也就是 1M。生产者在发送消息给 broker 的时候，如果出错，会尝试重发；但是如果是因为大小的原因，那生产者是不会重发的。另外，broker 上的消息可以进行压缩，这个参数可以使压缩后的大小，这样能多存储很多消息。

需要注意的是，允许发送更大的消息会对性能有很大影响。更大的消息，就意味着 broker 在处理网络连接的时候需要更长的时间，它也会增加磁盘的写操作压力，影响 IO 吞吐量。

PS：配合消息大小的设置

消息大小的参数 message.max.bytes 一般都和消费者的参数 fetch.message.max.bytes 搭配使用。如果 fetch.message.max.bytes 小于 message.max.bytes，那么当消费者遇到很大的消息时，将会无法消费这些消息。同理，在配置 cluster 时 replica.fetch.max.bytes 也是一样的道理。

kafka 权威指南 第二章第 4 节 硬件选择

问题导读：

1 Kafka 的对于硬件环境有什么要求？

2 Kafka 如何选择硬件以调整 Kafka 的性能？

硬件选择

kafka 的 broker 的硬件选择可以说是一门艺术。Kafka 本身不需要太特殊的硬件，它可以运行在任何机器上。但是一旦考虑到性能，就有很多因素需要考虑了，比如：磁盘的吞吐和容量，内存，网络以及 CPU。一旦你知道你的环境哪种性能是瓶颈，那么你就可以进行相应的硬件升级了。

磁盘吞吐

Producer 客户端生产消息的性能主要取决于 broker 磁盘写入的速度。Kafka 的消息在产生的时候在本地存储，并且大多数的客户端都会一直等到 broker 确认写入成功后，才认为消息提交成功。这就意味着越快的磁盘写入就会带来越小的延迟。

所以在磁盘的选择上，可以选择 HDD 硬盘或者是 SSD 固态硬盘。固态硬盘在读写的时候性能都更高，但是普通的机械硬盘也更经济实惠。你也可以通过设置多个存储路径或者 RAID，提高硬盘的使用率。另外，驱动也会影响消息的吞吐，比如 SAS 或者 SATA。

磁盘容量

容量是存储的另一关键指标，你想要保留多长时间或者多少的数据，就决定了你的磁盘容量至少要满足多少。如果 broker 每天都有 1T 的数据量，消息的保存时间为 7 天，那么你至少需要 7T 的磁盘空间。当然也需要考虑 10%的空间用来存储其他的文件，以及用户缓存的空间。

存储容量也是衡量 Kafka 集群规模的标志，集群的总容量可以通过主题切分多个分区来进行就扩展，这样多个 broker 可以共同分担存储压力。

内存

除了磁盘性能，内存也是一个很重要的指标。磁盘的性能会影响生产者产生消息的效率，内存效率也会有很大的影响。因为正常情况下 kafka 消费者的操作都是读取分区的末尾消息，它不会跟生产者产生消息的时间差距很大。在这种情况下，消息一般都是直接缓存在内容，消费者直接从内存中消费数据，这样效率更快一些。

Kafka 本身不需要为 JVM 分配很大的内容。对于 5G 的堆内存每秒处理 XM 的数据也是没问题的。kafka 使用页缓存也是很有好处的，这也是为什么推荐 kafka 不要与其他的应用部署在一个服务器上，因为这样会干扰页缓存的效率，会产生很多脏页。

网络

Kafka 会占用很大的网络带宽，这也是一个很重要的指标。在多个消费者的时候，也需要考虑 kafka 在传输和读取上的网络压力。一个生产者每秒钟向一个主题写入 1M 的数据，但是所有的消费者都会读取这部分的数据。其他的操作，比如几区你的备份、镜像等等都会增加网络压力。 在网络情况不好的时候，集群的备份会有延迟，此时整个集群也会变得脆弱。

CPU

CPU 计算的能力，相对于磁盘内存来说，不是那么重要，但是它也会影响 Broker 的性能。因为一般来说生产者的消息都会在 broker 进行压缩存储，以此来节省网络带宽以及磁盘空间。这需要 kafka 先解压缩，然后按照一定的规则再组织消息，最终再压缩存储到磁盘。这也是 kafka 大部分需要计算性能工作的地方，但是这不是选择硬件标准的主要因素。