比拼 Kafka, 大数据分析新秀Pulsar到底好在哪-InfoQ

浪尖聊大数据 4月23日

以下文章来源于ApachePulsar ， 作者Sijie Guo

ApachePulsar

Apache 软件基金会顶级项目，下一代云原生分布式消息流平台，集消息、存储、轻量化...

在用户选择一个消息系统时，消息模型是用户首先考虑的事情。消息模型应涵盖以下 3 个方面：

1. 消息消费——如何发送和消费消息；
2. 消息确认（ack）——如何确认消息；
3. 消息保存——消息保留多长时间，触发消息删除的原因以及怎样删除；

消息消费模型

在实时流式架构中，消息传递可以分为两类：队列（Queue）和流（Stream）。

队列（Queue）模型

队列模型主要是采用无序或者共享的方式来消费消息。通过队列模型，用户可以创建多个消费者从单个管道中接收消息；当一条消息从队列发送出来后，多个消费者中的只有一个（任何一个都有可能）接收和消费这条消息。消息系统的具体实现决定了最终哪个消费者实际接收到消息。

队列模型通常与无状态应用程序一起结合使用。无状态应用程序不关心排序，但它们确实需要能够确认（ack）或删除单条消息，以及尽可能地扩展消费并行性的能力。典型的基于队列模型的消息系统包括 RabbitMQ 和 RocketMQ。

流式（Stream）模型

相比之下，流模型要求消息的消费严格排序或独占消息消费。对于一个管道，使用流式模型，始终只会有一个消费者使用和消费消息。消费者按照消息写入管道的确切顺序接收从管道发送的消息。

流模型通常与有状态应用程序相关联。有状态的应用程序更加关注消息的顺序及其状态。消息的消费顺序决定了有状态应用程序的状态。消息的顺序将影响应用程序处理逻辑的正确性。

在面向微服务或事件驱动的体系结构中，队列模型和流模型都是必需的。

Pulsar 的消息消费模型

Apache Pulsar 通过“订阅”，抽象出了统一的: producer-topic-subscription-consumer 消费模型。Pulsar 的消息模型既支持队列模型，也支持流模型。

在 Pulsar 的消息消费模型中，Topic 是用于发送消息的通道。每一个 Topic 对应着 Apache BookKeeper 中的一个分布式日志。发布者发布的每条消息只在 Topic 中存储一次；存储的过程中，BookKeeper 会将消息复制存储在多个存储节点上；Topic 中的每条消息，可以根据消费者的订阅需求，多次被使用，每个订阅对应一个消费者组（Consumer Group）。

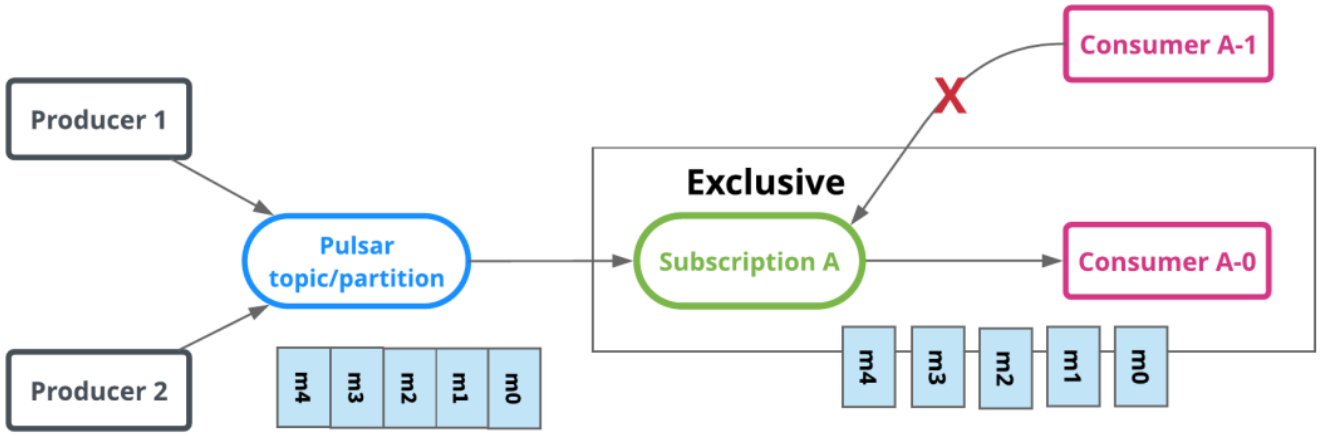
主题（Topic）是消费消息的真实来源。尽管消息仅在主题（Topic）上存储一次，但是用户可以有不同的订阅方式来消费这些消息：

- 消费者被组合在一起以消费消息，每个消费组是一个订阅。
- 每个 Topic 可以有不同的消费组。
- 每组消费者都是对主题的一个订阅。
- 每组消费者可以拥有自己不同的消费方式： 独占（Exclusive），故障切换（Failover）或共享（Share）。

Pulsar 通过这种模型，将队列模型和流模型这两种模型结合在了一起，提供了统一的 API 接口。 这种模型，既不会影响消息系统的性能，也不会带来额外的开销，同时还为用户提供了更多灵活性，方便用户程序以最匹配模式来使用消息系统。

独占订阅（Stream 流模型）

顾名思义，独占订阅中，在任何时间，一个消费者组（订阅）中有且只有一个消费者来消费 Topic 中的消息。下图是独占订阅的示例。在这个示例中有一个有订阅 A 的活跃消费者 A-0，消息 m0 到 m4 按顺序传送并由 A-0 消费。如果另一个消费者 A-1 想要附加到订阅 A，则是不被允许的。

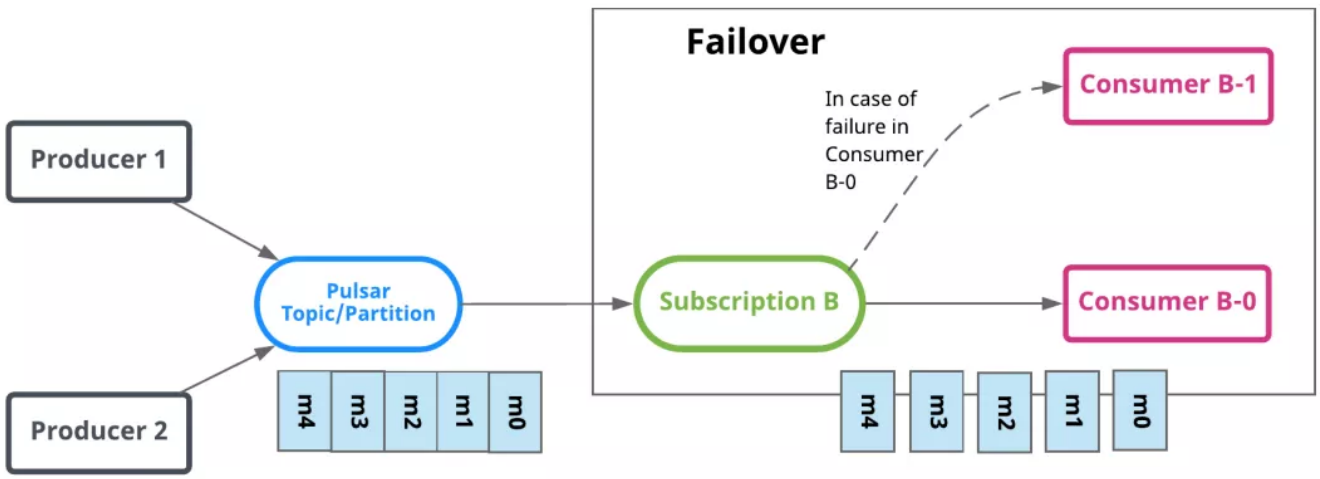


故障切换（Stream 流模型）

使用故障切换订阅，多个消费者（Consumer）可以附加到同一订阅。 但是，一个订阅中的所有消费者，只会有一个消费者被选为该订阅的主消费者。 其他消费者将被指定为故障转移消费者。

当主消费者断开连接时，分区将被重新分配给其中一个故障转移消费者，而新分配的消费者将成为新的主消费者。 发生这种情况时，所有未确认（ack）的消息都将传递给新的主消费者。 这类似于 Apache Kafka 中的 Consumer partition rebalance。

下图是故障切换订阅的示例。 消费者 B-0 和 B-1 通过订阅 B 订阅消费消息。B-0 是主消费者并接收所有消息。B-1 是故障转移消费者，如果消费者 B-0 出现故障，它将接管消费。



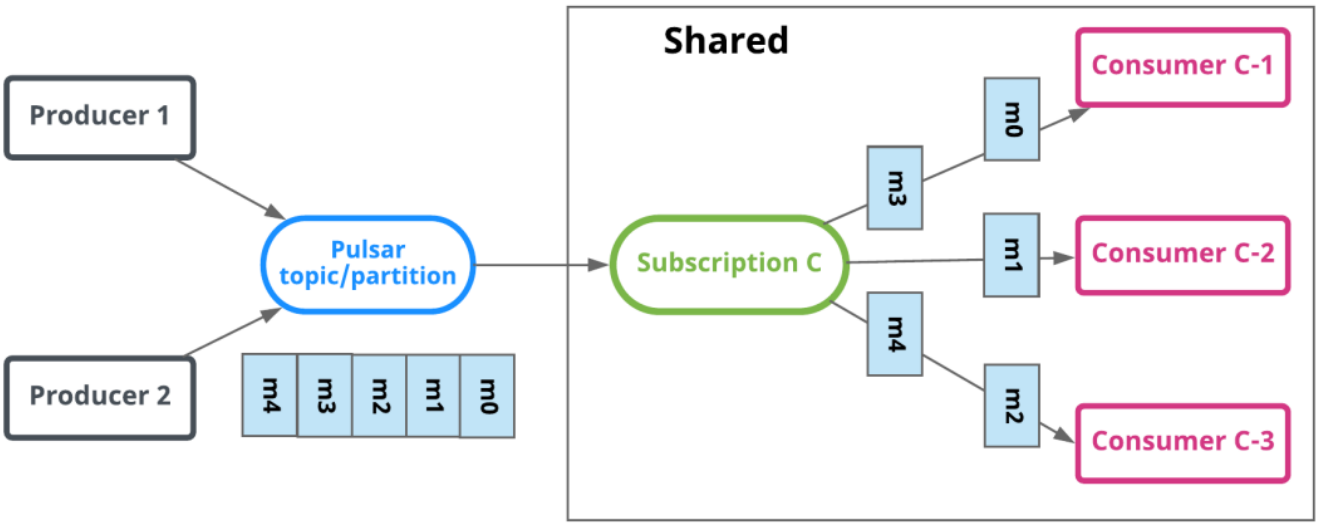
共享订阅（Queue 队列模型）

使用共享订阅，在同一个订阅背后，用户按照应用的需求挂载任意多的消费者。 订阅中的所有消息以循环分发形式发送给订阅背后的多个消费者，并且一个消息仅传递给一个消费者。

当消费者断开连接时，所有传递给它但是未被确认（ack）的消息将被重新分配和组织，以便发送给该订阅上剩余的剩余消费者。

下图是共享订阅的示例。 消费者 C-1，C-2 和 C-3 都在同一主题上消费消息。 每个消费者接收大约所有消息的 1/3。

如果想提高消费的速度，用户不需要不增加分区数量，只需要在同一个订阅中添加更多的消费者。



三种订阅模式的选择

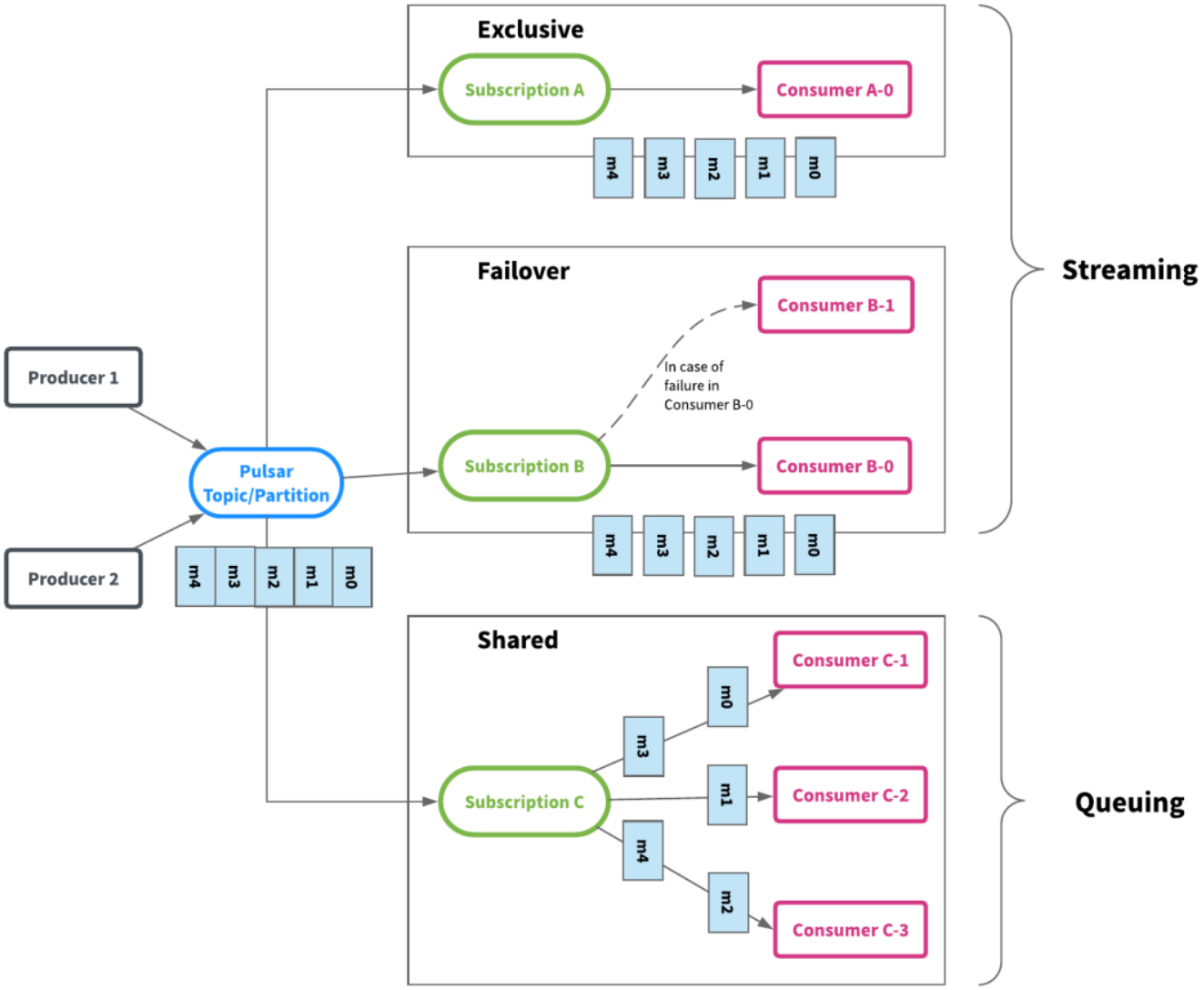
独占和故障切换订阅，仅允许一个消费者来使用和消费每个对主题的订阅。这两种模式都按主题分区顺序使用消息。它们最适用于需要严格消息顺序的流（Stream）用例。

共享订阅允许每个主题分区有多个消费者。同一订阅中的每个消费者仅接收主题分区的一部分消息。共享订阅最适用于不需要保证消息顺序的队列（Queue）的使用模式，并且可以按照需要任意扩展消费者的数量。

Pulsar 中的订阅实际上与 Apache Kafka 中的 Consumer Group 的概念类似。创建订阅的操作很轻量化，而且具有高度可扩展性，用户可以根据应用的需要创建任意数量的订阅。

对同一主题的不同订阅，也可以采用不同的订阅类型。比如用户可以在同一主题上可以提供一个包含 3 个消费者的故障切换订阅，同时也提供一个包含 20 个消费者的共享订阅，并且可以在不改变分区数量的情况下，向共享订阅添加更多的消费者。

下图描绘了一个包含 3 个订阅 A，B 和 C 的主题，并说明了消息如何从生产者流向消费者。



除了统一消息 API 之外，由于 Pulsar 主题分区实际上是存储在 Apache BookKeeper 中，它还提供了一个读取 API（Reader），类似于消费者 API（但 Reader 没有游标管理），以便用户完全控制如何使用 Topic 中的消息。

Pulsar 的消息确认（ACK）

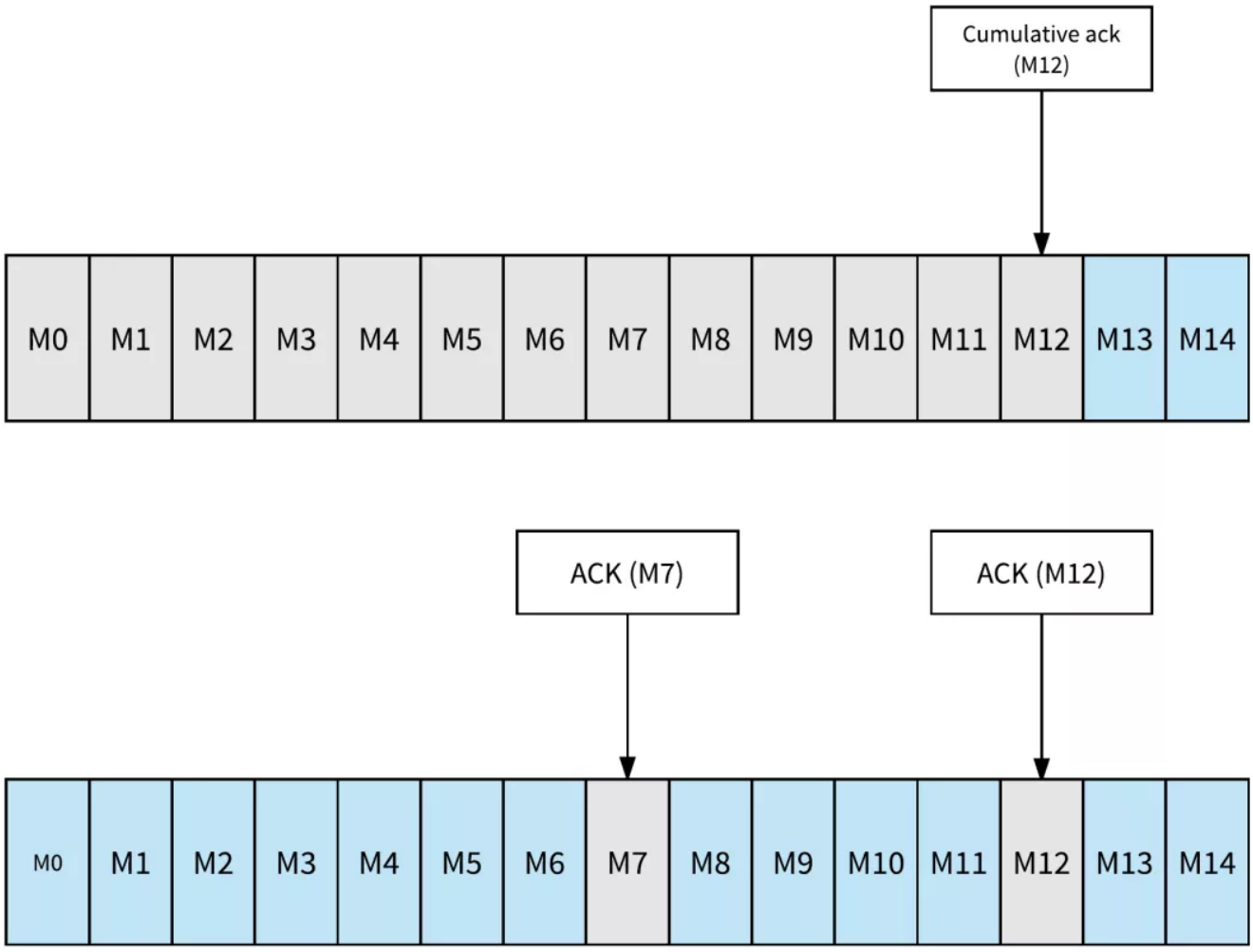
由于分布式系统的特性，当使用分布式消息系统时，可能会发生故障。比如在消费者从消息系统中的主题消费消息的过程中，消费消息的消费者和服务于主题分区的消息代理（Broker）都可能发生错误。消息确认（ACK）的目的就是保证当发生这样的故障后，消费者能够从上一次停止的地方恢复消费，保证既不会丢失消息，也不会重复处理已经确认（ACK）的消息。

在 Apache Kafka 中，恢复点通常称为 Offset，更新恢复点的过程称为消息确认或提交 Offset。

在 Apache Pulsar 中，每个订阅中都使用一个专门的数据结构–游标（Cursor）来跟踪订阅中的每条消息的确认（ACK）状态。每当消费者在主题分区上确认消息时，游标都会更新。更新游标可确保消费者不会再次收到消息。

Apache Pulsar 提供两种消息确认方法，单条确认（Individual Ack）和累积确认（Cumulative Ack）。通过累积确认，消费者只需要确认它收到的最后一条消息。主题分区中的所有消息（包括）提供消息 ID 将被标记为已确认，并且不会再次传递给消费者。累积确认与 Apache Kafka 中的 Offset 更新类似。

Apache Pulsar 可以支持消息的单条确认，也就是选择性确认。消费者可以单独确认一条消息。 被确认后的消息将不会被重新传递。下图说明了单条确认和累积确认的差异（灰色框中的消息被确认并且不会被重新传递）。在图的上半部分，它显示了累计确认的一个例子，M12 之前的消息被标记为 acked。在图的下半部分，它显示了单独进行 acking 的示例。仅确认消息 M7 和 M12 - 在消费者失败的情况下，除了 M7 和 M12 之外，其他所有消息将被重新传送。



独占订阅或故障切换订阅的消费者能够对消息进行单条确认和累积确认；共享订阅的消费者只允许对消息进行单条确认。单条确认消息的能力为处理消费者故障提供了更好的体验。对于某些应用来说，处理一条消息可能需要很长时间或者非常昂贵，防止重新传送已经确认的消息非常重要。

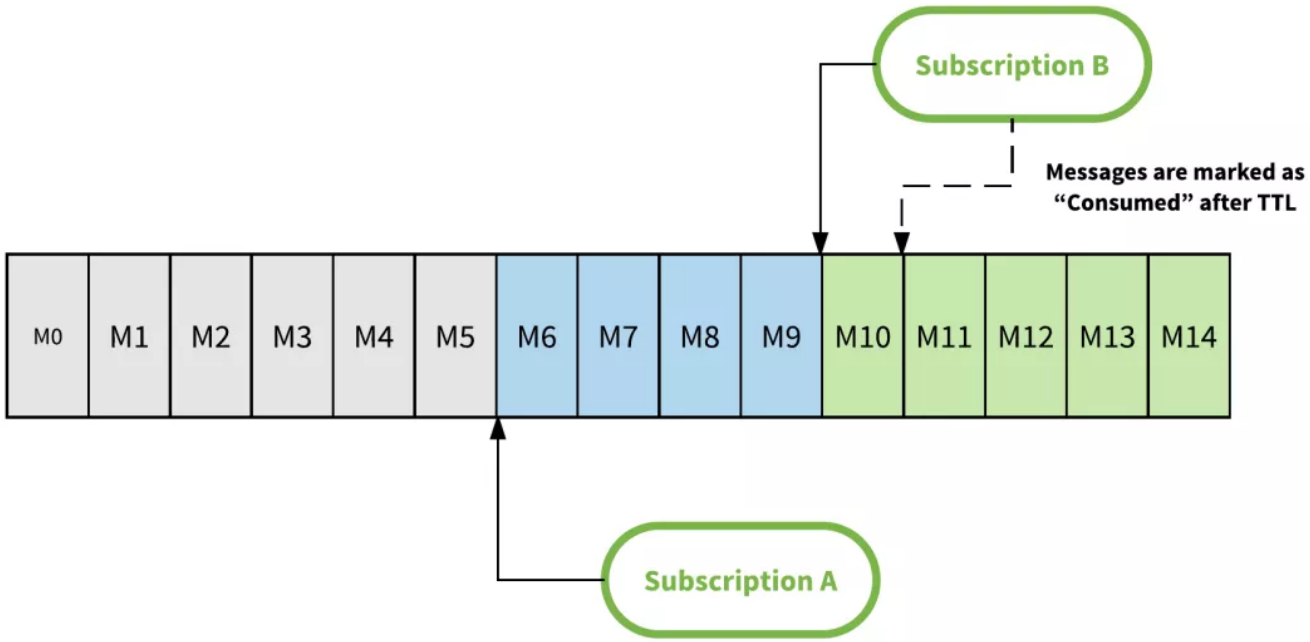
这个管理 Ack 的专门的数据结构–游标（Cursor），由 Broker 来管理，利用 BookKeeper 的 Ledger 提供存储，在后面的文章中我们会介绍更多的关于游标（Cursor）的细节。

Apache Pulsar 提供了灵活的消息消费订阅类型和消息确认方法，通过简单的统一的 API，就可以支持各种消息和流的使用场景。

Pulsar 的消息保留（Retention）

在消息被确认后，Pulsar 的 Broker 会更新对应的游标。当 Topic 里面中的一条消息，被所有的订阅都确认 ack 后，才能删除这条消息。Pulsar 还允许通过设置保留时间，将消息保留更长时间，即使所有订阅已经确认消费了它们。

下图说明了如何在有 2 个订阅的主题中保留消息。订阅 A 在 M6 和订阅 B 已经消耗了 M10 之前的所有消息之前已经消耗了所有消息。这意味着 M6 之前的所有消息（灰色框中）都可以安全删除。订阅 A 仍未使用 M6 和 M9 之间的消息，无法删除它们。如果主题配置了消息保留期，则消息 M0 到 M5 将在配置的时间段内保持不变，即使 A 和 B 已经确认消费了它们。



在消息保留策略中，Pulsar 还支持消息生存时间（TTL）。如果消息未在配置的 TTL 时间段内被任何消费者使用，则消息将自动标记为已确认。 消息保留期消息 TTL 之间的区别在于：消息保留期作用于标记为已确认并设置为已删除的消息，而 TTL 作用于未 ack 的消息。 上面的图例中说明了 Pulsar 中的 TTL。 例如，如果订阅 B 没有活动消费者，则在配置的 TTL 时间段过后，消息 M10 将自动标记为已确认，即使没有消费者实际读取该消息。

Pulsar VS. Kafka

通过以上几个方面，我们对 Pulsar 和 Kafka 在消息模型方面的不同点进行一个总结。

模型概念

Kafka：Producer - topic - consumer group - consumer；

Pulsar：Producer - topic - subscription - consumer。

消费模式

Kafka： 主要集中在流（Stream）模式，对单个 partition 是独占消费，没有共享（Queue）的消费模式；

Pulsar：提供了统一的消息模型和 API。流（Stream）模式 – 独占和故障切换订阅方式；队列（Queue）模式 – 共享订阅的方式。

消息确认（Ack）

Kafka： 使用偏移 Offset；

Pulsar：使用专门的 Cursor 管理。累积确认和 Kafka 效果一样；提供单条或选择性确认。

消息保留

Kafka：根据设置的保留期来删除消息。有可能消息没被消费，过期后被删除。 不支持 TTL。

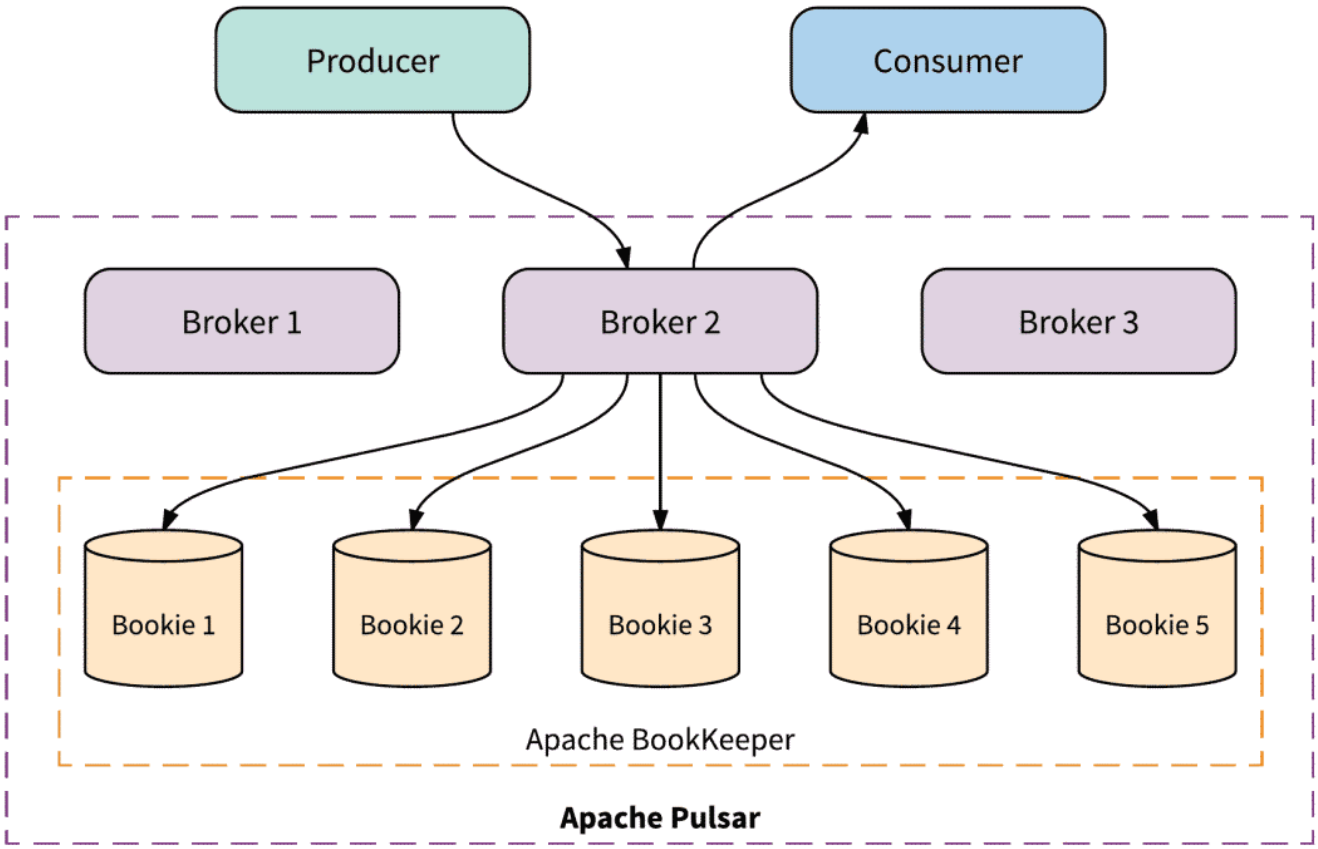
Pulsar：消息只有被所有订阅消费后才会删除，不会丢失数据。也允许设置保留期，保留被消费的数据。支持 TTL。

对比总结

Apache Pulsar 将高性能的流（Apache Kafka 所追求的）和灵活的传统队列（RabbitMQ 所追求的）结合到一个统一的消息模型和 API 中。Pulsar 使用统一的 API 为用户提供一个支持流和队列的系统，且具有同样的高性能。 应用程序可以将此统一的 API 用于高性能队列和流式传输，而无需维护两套系统：RabbitMQ 进行队列处理，Kafka 进行流式处理。

系统架构和设计理念
Pulsar 的分层架构

Apache Pulsar 和其他消息系统最根本的不同是采用分层架构。Apache Pulsar 集群由两层组成：无状态服务层，由一组接收和传递消息的 Broker 组成；以及一个有状态持久层，由一组名为 bookies 的 Apache BookKeeper 存储节点组成，可持久化地存储消息。 下图显示了 Apache Pulsar 的典型部署。



在 Pulsar 客户端中提供生产者和消费者（Producer & Consumer）接口，应用程序使用 Pulsar 客户端连接到 Broker 来发布和消费消息。

Pulsar 客户端不直接与存储层 Apache BookKeeper 交互。 客户端也没有直接的 BookKeeper 访问权限。这种隔离，为 Pulsar 实现安全的多租户统一身份验证模型提供了基础。

Apache Pulsar 为客户端提供多种语言的支持，包括 Java，C ++，Python，Go 和 Websockets。

Apache Pulsar 还提供了一组兼容 Kafka 的 API，用户可以通过简单地更新依赖关系并将客户端指向 Pulsar 集群来迁移现有的 Kafka 应用程序，这样现有的 Kafka 应用程序可以立即与 Apache Pulsar 一起使用，无需更改任何代码。

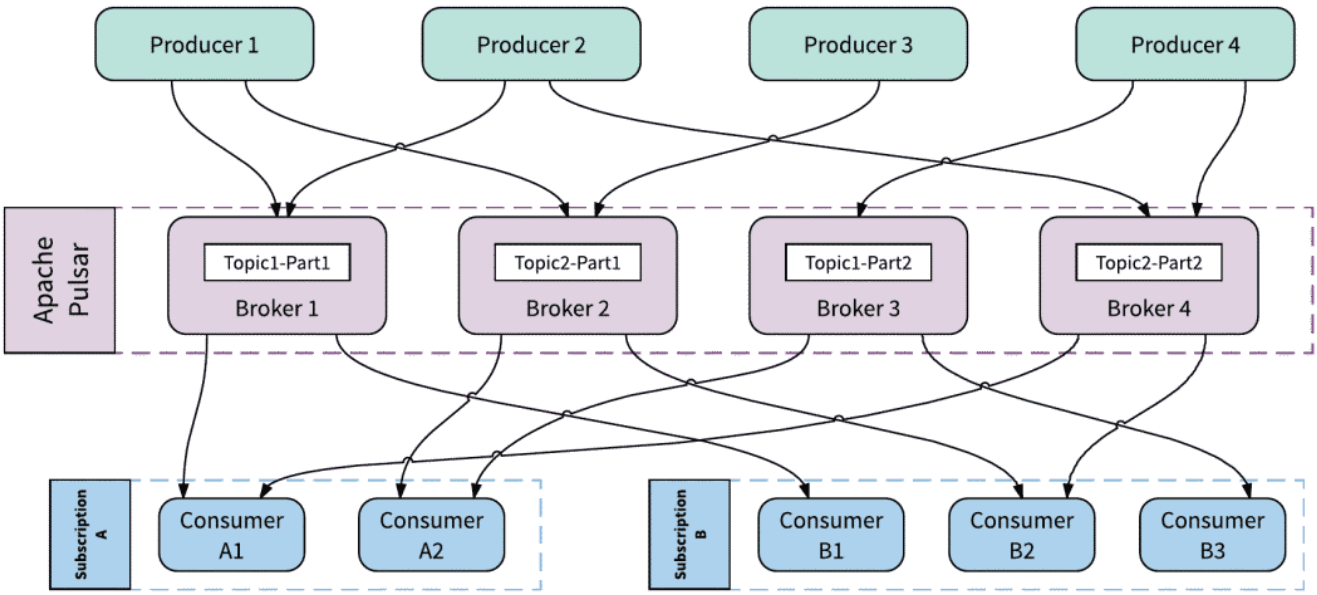
Broker 层：无状态服务层

Broker 集群在 Apache Pulsar 中形成无状态服务层。服务层是“无状态的”，因为 Broker 实际上并不在本地存储任何消息数据。有关 Pulsar 主题的消息，都被存储在分布式日志存储系统（Apache BookKeeper）中。我们将在下一节中更多地讨论 BookKeeper。

每个主题分区（Topic Partition）由 Pulsar 分配给某个 Broker，该 Broker 称为该主题分区的所有者。Pulsar 生产者和消费者连接到主题分区的所有者 Broker，以向所有者代理发送消息并消费消息。

如果一个 Broker 失败，Pulsar 会自动将其拥有的主题分区移动到群集中剩余的某一个可用 Broker 中。这里要说的一件事是：由于 Broker 是无状态的，当发生 Topic 的迁移时，Pulsar 只是将所有权从一个 Broker 转移到另一个 Broker，在这个过程中，不会有任何数据复制发生。

下图显示了一个拥有 4 个 Broker 的 Pulsar 集群，其中 4 个主题分区分布在 4 个 Broker 中。每个 Broker 拥有并为一个主题分区提供消息服务。



BookKeeper 层：持久化存储层

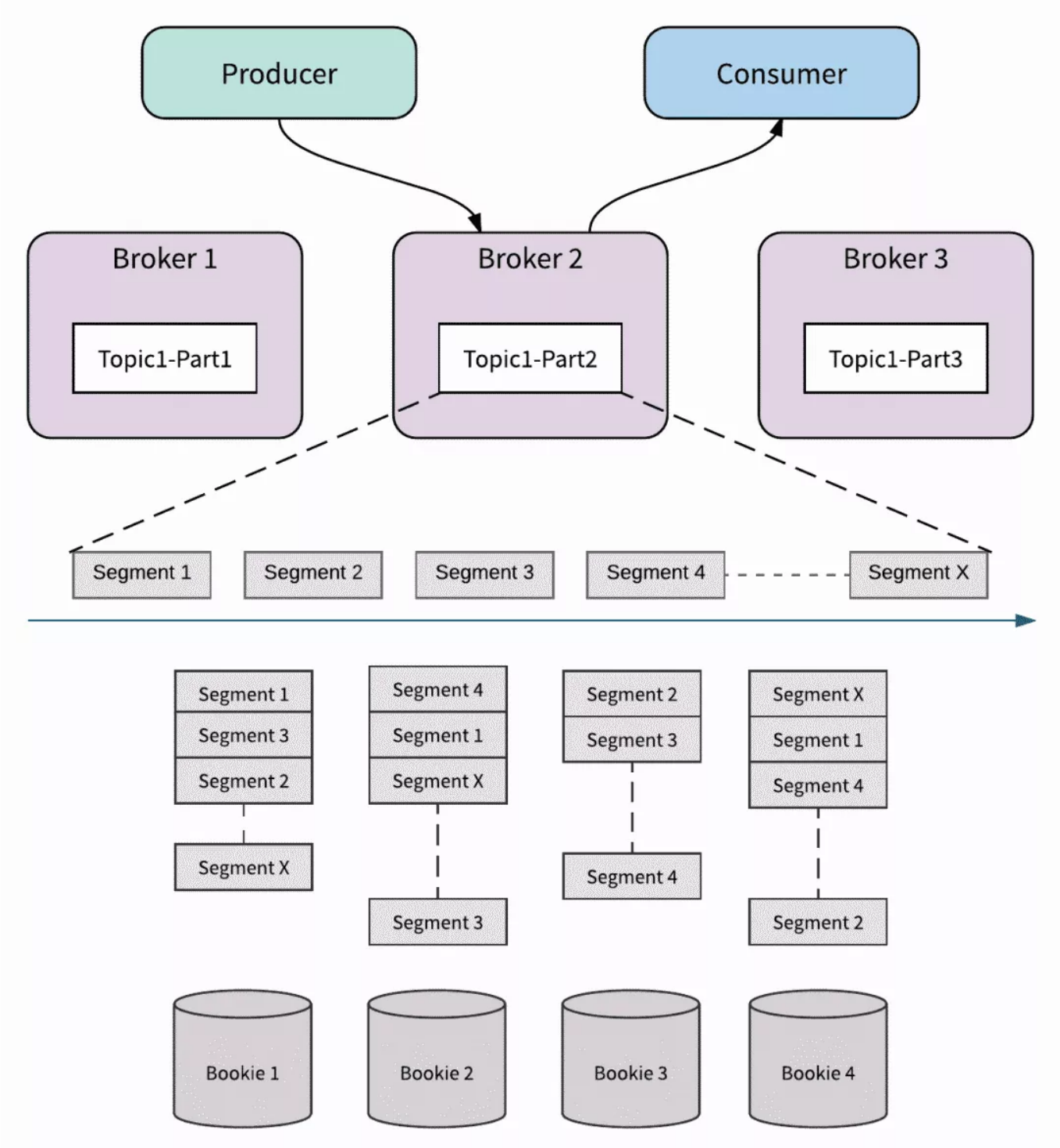
Apache BookKeeper 是 Apache Pulsar 的持久化存储层。Apache Pulsar 中的每个主题分区本质上都是存储在 Apache BookKeeper 中的分布式日志。

每个分布式日志又被分为 Segment 分段。 每个 Segment 分段作为 Apache BookKeeper 中的一个 Ledger，均匀分布并存储在 BookKeeper 群集中的多个 Bookie （Apache BookKeeper 的存储节点）中。

Segment 的创建时机包括以下几种：基于配置的 Segment 大小；基于配置的滚动时间；或者当 Segment 的所有者被切换。

通过 Segment 分段的方式，主题分区中的消息可以均匀和平衡地分布在群集中的所有 Bookie 中。 这意味着主题分区的大小不仅受一个节点容量的限制； 相反，它可以扩展到整个 BookKeeper 集群的总容量。

下面的图说明了一个分为 x 个 Segment 段的主题分区。 每个 Segment 段存储 3 个副本。 所有 Segment 都分布并存储在 4 个 Bookie 中。



Segment 为中心的存储

存储服务的分层的架构 和 以 Segment 为中心的存储 是 Apache Pulsar（使用 Apache BookKeeper）的两个关键设计理念。 这两个基础为 Pulsar 提供了许多重要的好处：

- 无限制的主题分区存储
- 即时扩展，无需数据迁移
- 无缝 Broker 故障恢复

- 无缝集群扩展
- 无缝的存储（Bookie）故障恢复
- 独立的可扩展性

下面我们分别展开来看这几个好处。

无限制的主题分区存储

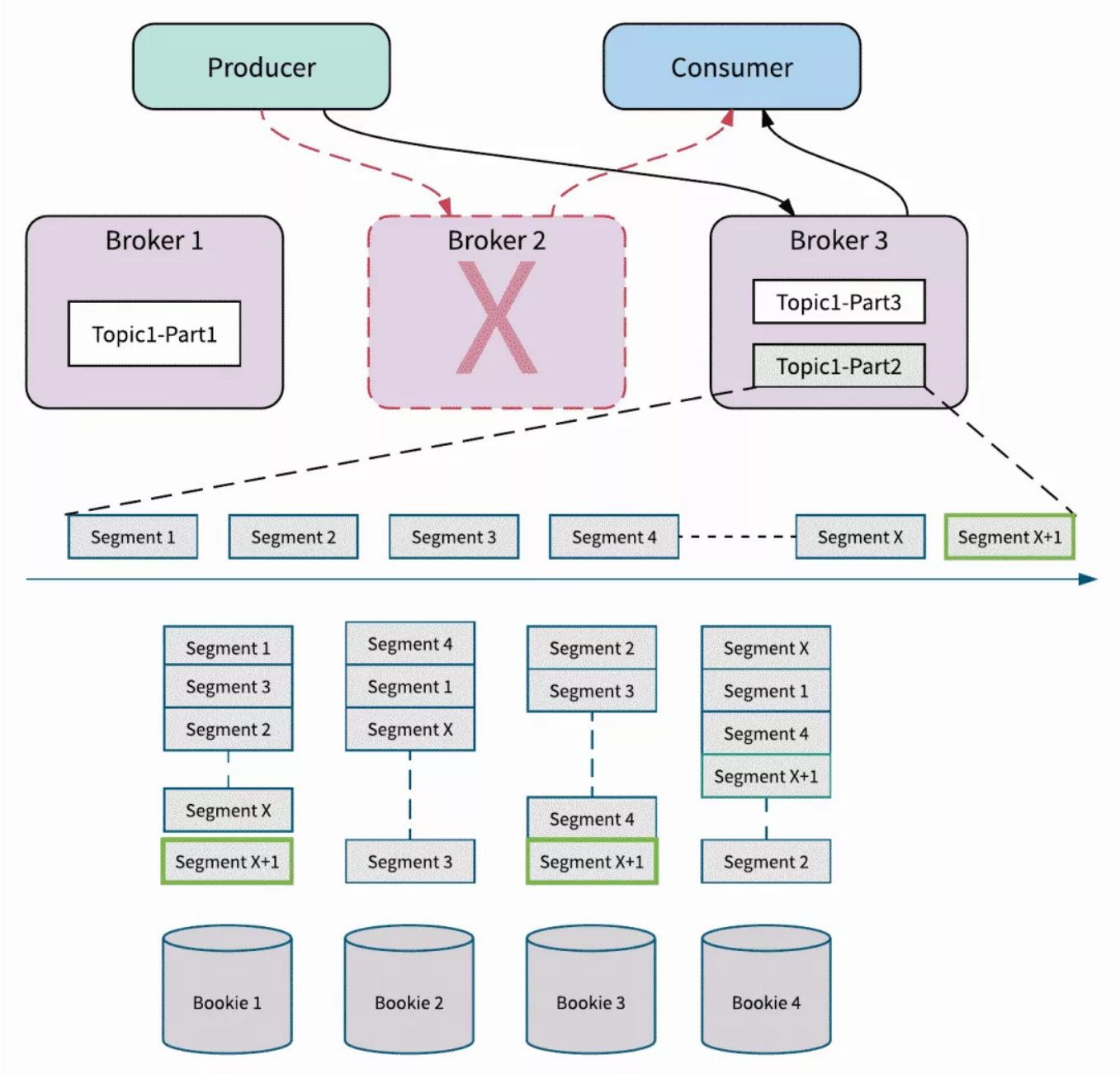
由于主题分区被分割成 Segment 并在 Apache BookKeeper 中以分布式方式存储，因此主题分区的容量不受任何单一节点容量的限制。 相反，主题分区可以扩展到整个 BookKeeper 集群的总容量，只需添加 Bookie 节点即可扩展集群容量。 这是 Apache Pulsar 支持存储无限大小的流数据，并能够以高效，分布式方式处理数据的关键。 使用 Apache BookKeeper 的分布式日志存储，对于统一消息服务和存储至关重要。

即时扩展，无需数据迁移

由于消息服务和消息存储分为两层，因此将主题分区从一个 Broker 移动到另一个 Broker 几乎可以瞬时内完成，而无需任何数据重新平衡（将数据从一个节点重新复制到另一个节点）。 这一特性对于高可用的许多方面至关重要，例如集群扩展；对 Broker 和 Bookie 失败的快速应对。 我将使用例子在下文更详细地进行解释。

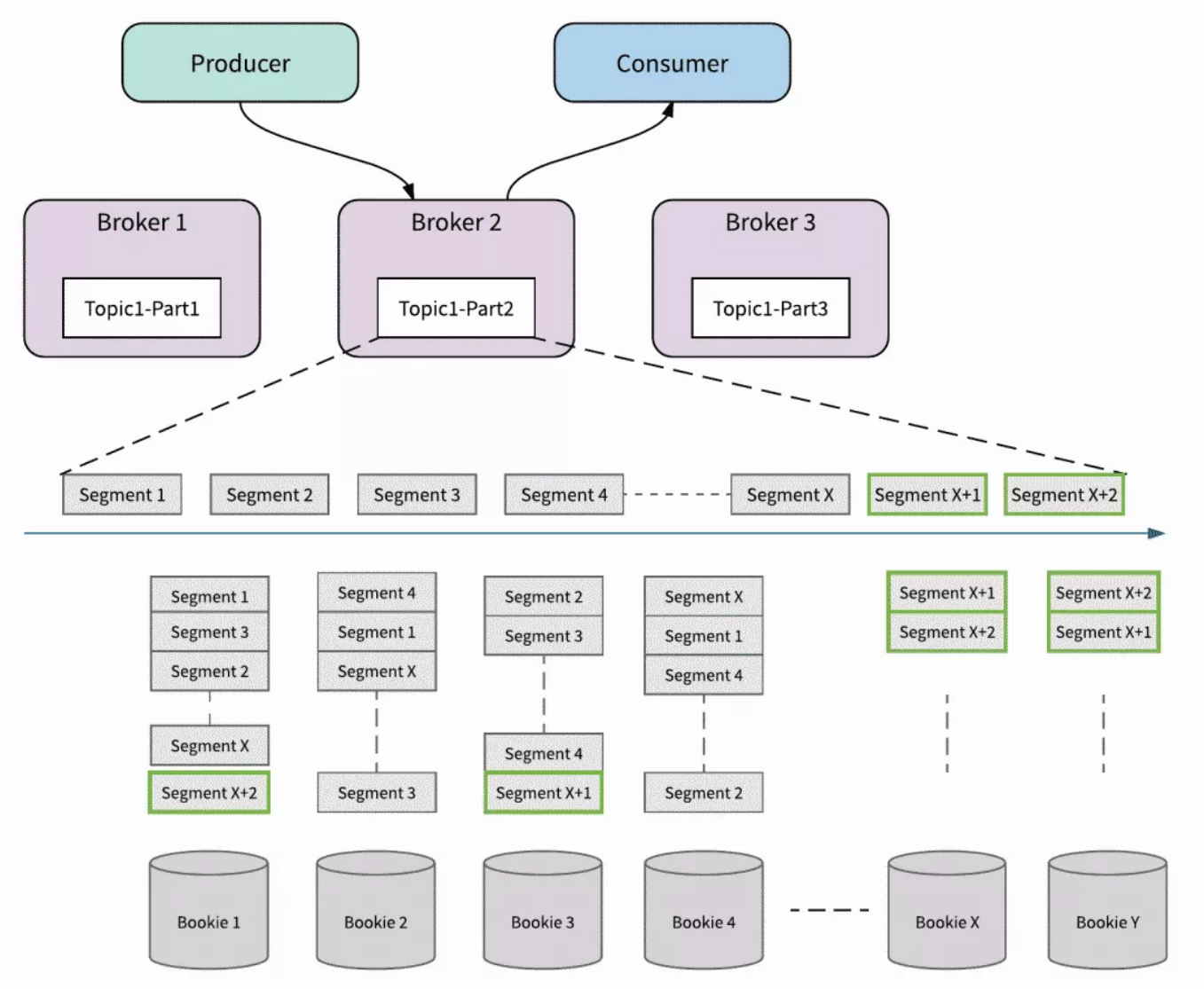
无缝 Broker 故障恢复

下图说明了 Pulsar 如何处理 Broker 失败的示例。 在例子中 Broker 2 因某种原因（例如停电）而断开。Pulsar 检测到 Broker 2 已关闭，并立即将 Topic1-Part2 的所有权从 Broker 2 转移到 Broker 3。在 Pulsar 中数据存储和数据服务分离，所以当代理 3 接管 Topic1-Part2 的所有权时，它不需要复制 Partiton 的数据。 如果有新数据到来，它立即附加并存储为 Topic1-Part2 中的 Segment x + 1。Segment x + 1 被分发并存储在 Bookie1, 2 和 4 上。因为它不需要重新复制数据，所以所有权转移立即发生而不会牺牲主题分区的可用性。



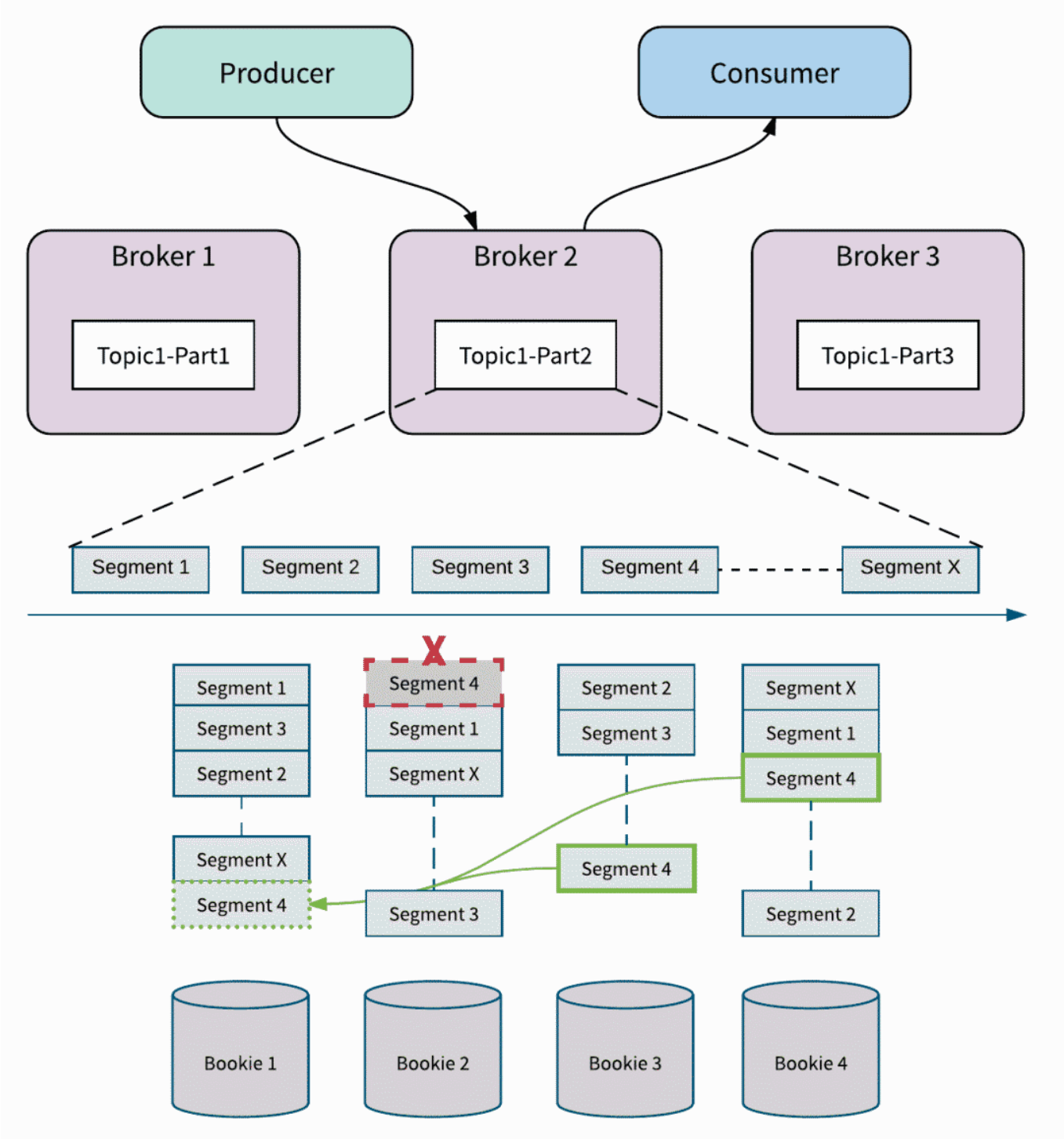
无缝集群容量扩展

下图说明了 Pulsar 如何处理集群的容量扩展。 当 Broker 2 将消息写入 Topic1-Part2 的 Segment X 时，将 Bookie X 和 Bookie Y 添加到集群中。Broker 2 立即发现新加入的 Bookies X 和 Y。然后 Broker 将尝试将 Segment X + 1 和 X + 2 的消息存储到新添加的 Bookie 中。 新增加的 Bookie 立刻被使用起来，流量立即增加，而不会重新复制任何数据。 除了机架感知和区域感知策略之外，Apache BookKeeper 还提供资源感知的放置策略，以确保流量在群集中的所有存储节点之间保持平衡。



无缝的存储（Bookie）故障恢复

下图说明了 Pulsar（通过 Apache BookKeeper）如何处理 bookie 的磁盘故障。 这里有一个磁盘故障导致存储在 bookie 2 上的 Segment 4 被破坏。Apache BookKeeper 后台会检测到这个错误并进行复制修复。



Apache BookKeeper 中的副本修复是 Segment（甚至是 Entry）级别的多对多快速修复，这比重新复制整个主题分区要精细，只会复制必须的数据。 这意味着 Apache BookKeeper 可以从 bookie 3 和 bookie 4 读取 Segment 4 中的消息，并在 bookie 1 处修复 Segment 4。所有的副本修复都在后台进行，对 Broker 和应用透明。

即使有 Bookie 节点出错的情况发生时，通过添加新的可用的 Bookie 来替换失败的 Bookie，所有 Broker 都可以继续接受写入，而不会牺牲主题分区的可用性。

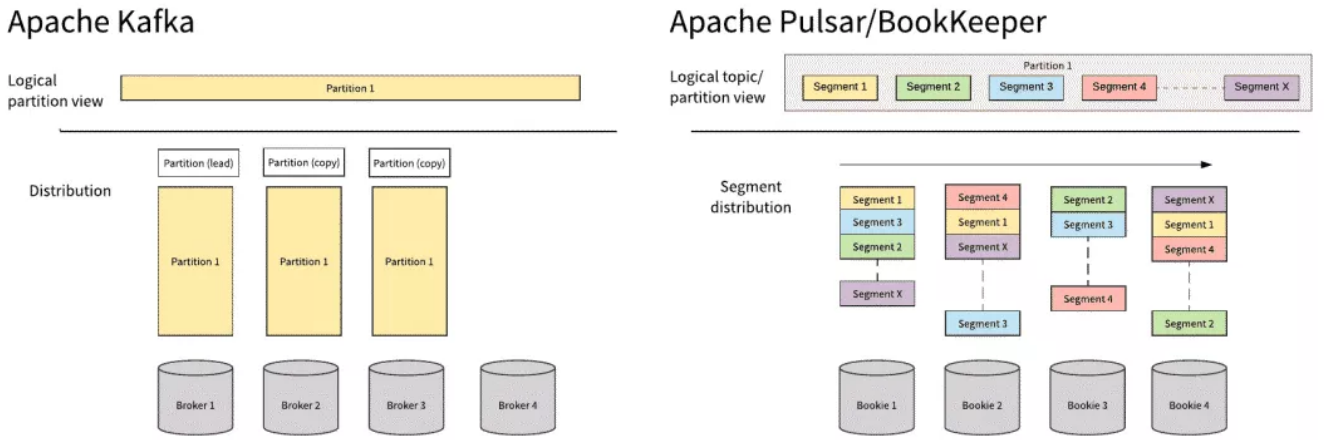
独立的可扩展性

由于消息服务层和持久存储层是分开的，因此 Apache Pulsar 可以独立地扩展存储层和服务层。这种独立的扩展，更具成本效益：

- 当您需要支持更多的消费者或生产者时，您可以简单地添加更多的 Broker。主题分区将立即在 Brokers 中做平衡迁移，一些主题分区的所有权立即转移到新的 Broker。
- 当您需要更多存储空间来将消息保存更长时间时，您只需添加更多 Bookie。通过智能资源感知和数据放置，流量将自动切换到新的 Bookie 中。Apache Pulsar 中不会涉及到不必要的数据搬迁，不会将旧数据从现有存储节点重新复制到新存储节点。

Pulsar VS. Kafka

Apache Kafka 和 Apache Pulsar 都有类似的消息概念。 客户端通过主题与消息系统进行交互。 每个主题都可以分为多个分区。 然而，Apache Pulsar 和 Apache Kafka 之间的根本区别在于 Apache Kafka 是以分区为存储中心，而 Apache Pulsar 是以 Segment 为存储中心。



上图显示了以分区为中心和以 Segment 为中心的系统之间的差异。

在 Apache Kafka 中，分区只能存储在单个节点上并复制到其他节点，其容量受最小节点容量的限制。这意味着容量扩展需要对分区重新平衡，这反过来又需要重新复制整个分区，以平衡新添加的代理的数据和流量。

重新传输数据非常昂贵且容易出错，并且会消耗网络带宽和 I/O。维护人员在执行此操作时必须非常小心，以避免破坏生产系统。

Kafka 中分区数据的重新拷贝不仅发生在以分区为中心的系统中的群集扩展上。许多其他事情也会触发数据重新拷贝，例如副本故障，磁盘故障或计算机的故障。在数据重新复制期间，分

区通常不可用，直到数据重新复制完成。例如，如果您将分区配置为存储为 3 个副本，这时，如果丢失了一个副本，则必须重新复制完整个分区后，分区才可以再次可用。

在用户遇到故障之前，通常会忽略这种缺陷，因为许多情况下，在短时间内仅是对内存中缓存数据的读取。当数据被保存到磁盘后，用户将越来越多地不可避免地遇到数据丢失，故障恢复的问题，特别是在需要将数据长时间保存的场合。

相反，在 Apache Pulsar 中，同样是以分区为逻辑单元，但是以 Segment 为物理存储单元。分区随着时间的推移会进行分段，并在整个集群中均衡分布，旨在有效地迅速地扩展。

Pulsar 是以 Segment 为中心的，因此在扩展容量时不需要数据重新平衡和拷贝，旧数据不会被重新复制，这要归功于在 Apache BookKeeper 中使用可扩展的以 Segment 为中心的分布式日志存储系统。

通过利用分布式日志存储，Pulsar 可以最大化 Segment 放置选项，实现高写入和高读取可用性。 例如，使用 BookKeeper，副本设置等于 2，只要任何 2 个 Bookie 启动，就可以对主题分区进行写入。 对于读取可用性，只要主题分区的副本集中有 1 个处于活动状态，用户就可以读取它，而不会出现任何不一致。

总之，Apache Pulsar 这种独特的基于分布式日志存储的以 Segment 为中心的发布/订阅消息系统可以提供许多优势，例如可靠的流式系统，包括无限制的日志存储，无需分区重新平衡的即时扩展，快速复制修复以及通过最大化数据放置实现高写入和读取可用性选项。

英文原文链接：

https://streaml.io/blog/pulsar-streaming-queuing

https://streaml.io/blog/pulsar-segment-based-architecture

如果对 Pulsar 感兴趣，可通过下列方式参与 Pulsar 社区：

- Pulsar Slack 频道：
- https://apache-pulsar.slack.com/
- 可自行在这里注册：
- https://apache-pulsar.herokuapp.com/
- Pulsar 邮件列表:
- http://pulsar.incubator.apache.org/contact

有关 Apache Pulsar 项目的更多信息，请访问官网：

http://pulsar.incubator.apache.org/

阅读原文