MQ LRP

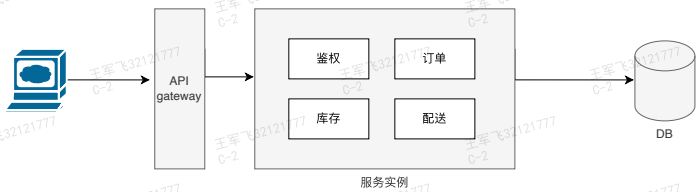
C-2 创建:王军飞, 最后修改: 王军飞 02-07 12:07

目录

1.云原生时代，Mafka的挑战

1.1单体式架构时代

大约在十几年前，我们在构建互联网应用时，使用的是以下这样的架构：



外层提供一个API供客户端来调用，内层分为几个模块，每个模块负责一部分应用，如图所示，比如用户管理，购物车，商品目录，订单管理，库存，运输。

这是一个典型的单体架构应用，所有的模块和API层都在一个应用里，客户端在HTML里调用服务端API来完成展示和交互，

服务端访问数据库来完成一系列业务逻辑。

开发、测试、部署，问题查找，扩容，都很方便快捷，尤其在初创型公司里，一个或几个人快速搭建一个应用原型，单体式架构非常高效。

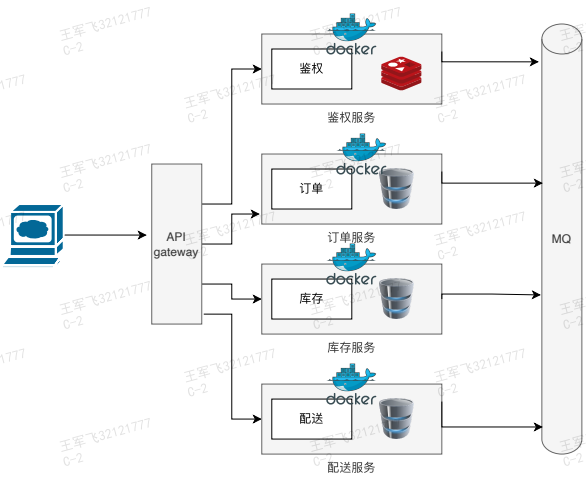
随着业务需求的迭代，系统功能的增加，单体式应用逐渐遇到一些开发上的问题：

1. 系统随着业务需求的增加变得越来越复杂，没有一个人能完全了解整个系统。
2. 每一次新的需求迭代开发，所需的开发时间越来越长，而且越来越容易出错误。
3. 一个小的变更需要将整个系统重新部署一次，变更的代价和风险比较大。
4. 一个小组件的问题或bug，会让整个系统变得不可用。
5. 新的技术和框架很难被应用到项目中去

直到最后，不得不重新编写整个系统。

这个时候，微服务架构诞生！

1.2微服务架构时代



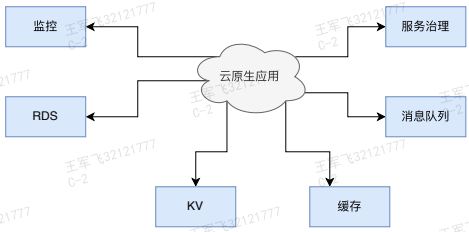
在微服务架构下，整个系统被拆分为若干个独立的服务，他们有各自的代码库，独立演进和迭代，如下图所示：

1. 整个系统被拆分为若干个小的服务子系统，便于理解和规划
2. 每个小服务都是小单元，小项目，包含自己的数据，repo，代码和依赖，便于独立开发和迭代
3. 每个服务都部署在一个容器内，由容器管理器来统一编排
4. 没有统一的一个关系数据库，都有各自的数据库或存储，以及分布式缓存
5. 使用统一的网关来做一些横向的策略应用，比如鉴权，分控等。

最重要的是，现代的微服务架构充分使用了云的基础设施，实现**高扩展性**，**高可用性**，以及**高韧性**，我们统一称这类微服务或应用为**云原生应用**(cloud native application)。

面对市场形势多变的特点，以及互联网业务发展迅猛的形势，企业必须对业务进行快速的开发和迭代，以抢占瞬息万变的市场机会。云原生应用的最大特点就是快速拥抱变化，他们可以在几分钟内快速的扩容，缩容，迁移，销毁，以应对业务和市场快速的需求变化。

这对应用和服务的周边配套设施，如构建，部署，配置，监控，PaaS，IaaS等支持服务提出了更高的要求.



云原生应用的支撑和辅助服务

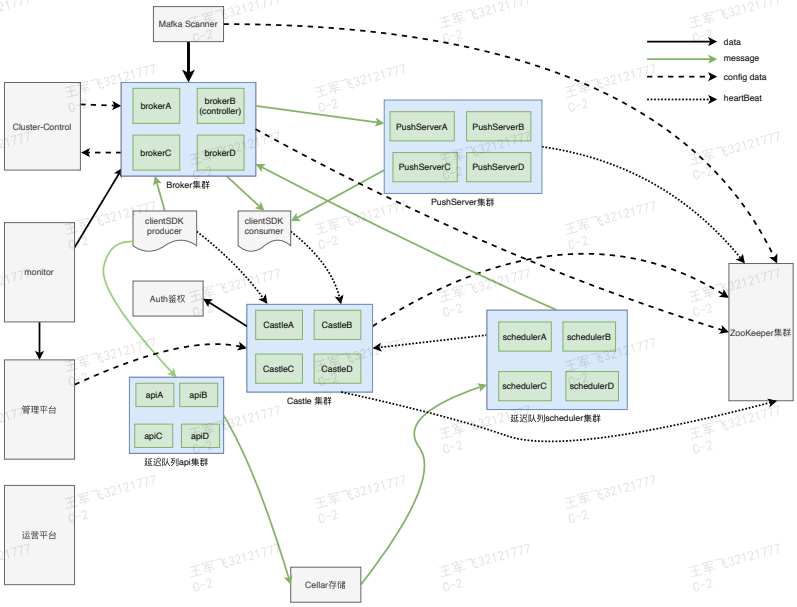
1.3 Mafka面临的挑战

Mafka作为支持云原生的PaaS服务之一，面临着以下三个方面的挑战

1.3.1 第一、扩展能力

1.3.1.1 接入k8s，提升集群扩展能力

以下是Mafka的系统架构图



整体架构上，要匹配云原生服务的快速扩缩容，Mafka的几个核心模块首先必须具备横向扩展能力

模块	简介	服务类型	服务架构	容器+弹性伸缩接入情况	扩缩容能力
Castle	Mafka调度器	RPC服务 (MNS invoker + Netty自有协议)	本机可重建Cache + KV存储 + ZK选主	已接入容器 已接入弹性伸缩	具备横向扩容
PushServer	Mafka Push代消费	RPC服务(Castle服务发现+Netty自有协议)	本机可重建Cache + KV存储 + ZK选主/简单通知	已接入容器	具备横向扩容
DelayServer	Mafka 延迟队列	RPC服务(Octo+Mtthrift)	本机可重建Cache + KV存储 + ZK选主	已接入容器 已接入弹性伸缩	具备横向扩容
Broker	Mafka 消息Broker	基于文件的分布式系统	本机磁盘存储 + ZK选主/元信息存储	接入容器中	具备横向扩容

如图，三个RPC服务都已经接入了容器，并且接入了hulk弹性伸缩，已经具备了横向扩缩容的能力。

Mafka Broker 正在接入容器过程中，现阶段主要靠虚机来做扩缩容，目前还未接入弹性。

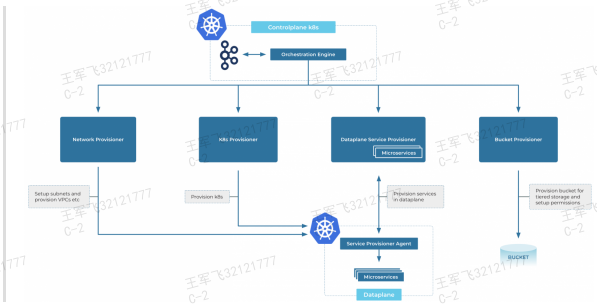
如消息队列业界调研所述，Kafka、Pulsar、RocketMQ都在使用容器以及k8s技术来增强自身的扩展性。

三家产品容器化技术对比：

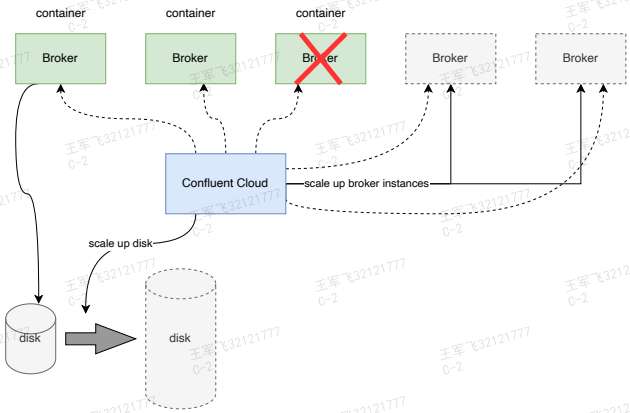
产品	扩容组件	容器化技术
Kafka	Confluent Cloud for k8s	k8s operator
Pulsar	StreamNative Helm	k8s operator
RocketMQ	阿里云消息队列RocketMQ版	k8s operator

Kafka官方公司Confluent Cloud For k8s的服务架构:

*下图引用自confluent engineering blog



以Kafka为例，Confluent使用自研的k8s operator为集群扩容或替换。



如上图所示，Confluent Cloud可以动态扩容集群的节点，磁盘、CPU、MEM等，甚至可以做到以编程的方式来自动编排。

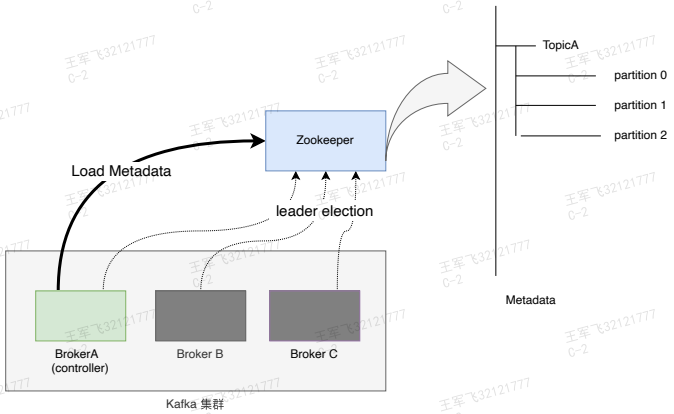
当某台容器宕机后，他可以自动侦测到，并自动添加新的节点到集群内，填补宕机节点的空缺。

所以，要提升Mafka的扩展能力，**第一步要做的就是接入容器化，增加k8s operator的使用，增强Mafka broker的横向扩展能力**。

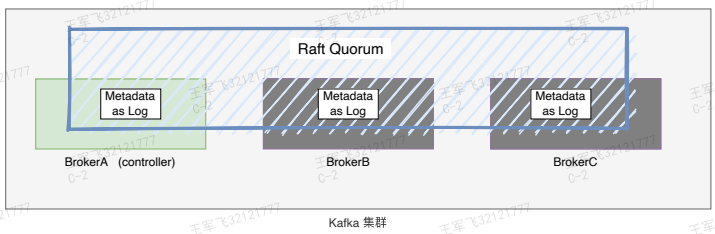
1.3.1.2 去除ZooKeeper依赖，提升broker的扩展性

Kafka一开始设计的时候，依靠zookeeper来解决系统的一致性问题，比如集群controller的选主，partition leader的选主，同时还保存了集群的metadata信息。

随着集群内topic数量，分区数量的增长，metadat数据数量也跟着增长，但是kafka集群在启动以后，或集群发生重新选主时，集群的controller需要全量拉取一次zookeeper上的metadata数据，所以这个数据越大，拉取的耗时也会越长，直接影响了集群controller的可用性，最终必须限制metadata数据的大小，**这就限制了Kafka集群的横向扩展能力**。



Kafka官方为了解决这个问题，引入了一个新的提案KIP-500，去除Kafka对zookeeper的依赖

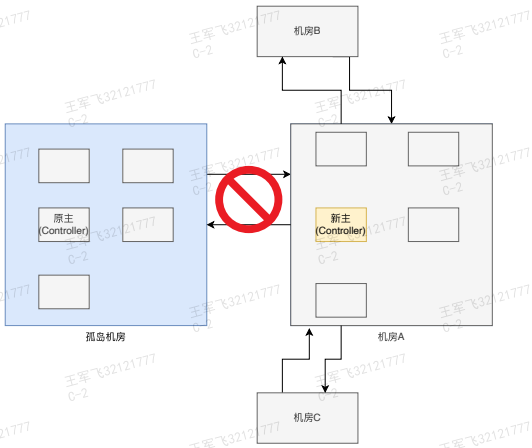


如上图，每个broker内都有一个副本Log文件，它的内容就是集群的全量的metadata数据。同时集群内stand by 的controller会形成一个Raft协议集群，依靠自身来做一致性选举。

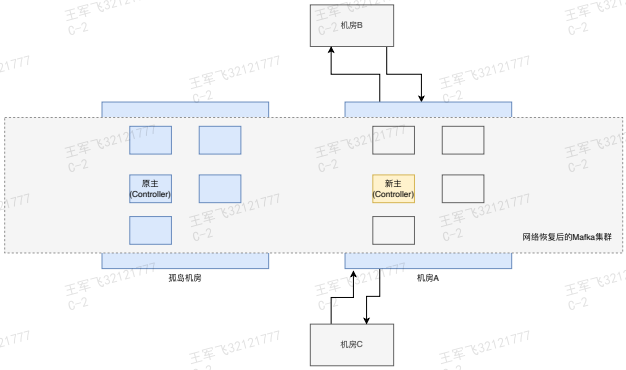
当发生controller选主的时候，Kafka集群不再需要从zookeeper 读取全量的metadata数据，这样就加快了controller的启动过程，即便集群内metadata数据再大，也不会有影响，**①直接提升了broker的扩展性**。

去除zookeeper除了提升broker的扩展性外，**②还能降低组件的复杂度**，两个组件减少为一个组，演进和迭代能减轻很大的负担；**③同时还能减少部署和运维**，部署kafka集群时不需要再部署一个zookeeper集群，也不需要单独维护zookeeper的稳定性。

除此之外，**④还能解决Mafka 容灾2.0特殊情况下的集群脑裂问题:** 如下图



当机房间发生断网时，如果形成一个孤岛机房，而且集群的主(Controller)又恰好在孤岛机房内，孤岛机房外的集群节点会选举一个新的主(Controller)，这时当网络恢复后会，集群内会出现两个Controller，整个集群会发生脑裂。



如果引入Kafka KIP-500去除zookeeper PR后，一致性内嵌在Broker内部，可以通过配置小Quorum大小来解除孤岛机房内的主(Controller)，避免形成脑裂问题。

1.3.1.3 抽取KRaft组件，形成一个以选主为主要功能的基础组件，供Mafka内部、公司内部其他组件使用

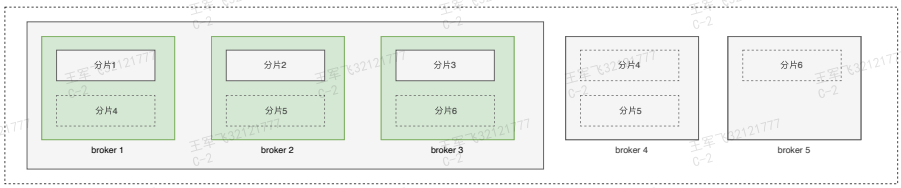
1.3.2 第二、扩展效率

1.3.2.1 使用分层存储，提升存量分区的迁移速度

在第一部分提到在云原生时代，云原生应用除了需要很高的横向扩展能力外，更重要的是要具备快速的扩展效率。

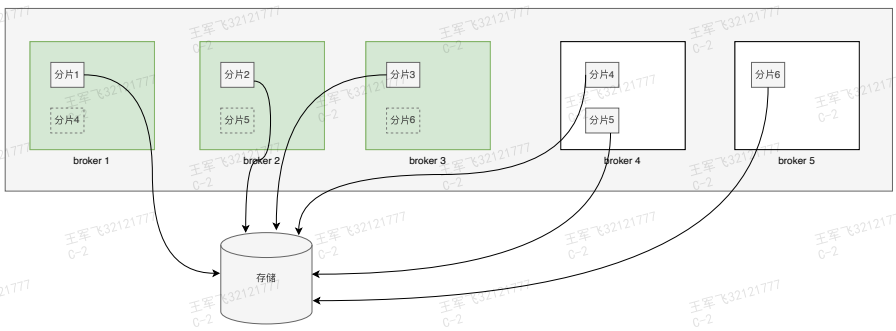
要提升扩展效率，除了接入容器化、k8s operator和弹性来提升自动化能力外，还要增强broker的扩容速度。

如Mafka Broker全链路弹性伸缩演进策略所述，对于单个主题或多个主题扩容时都有比较快速的方法，但是对集群整体扩容时，存量数据的迁移速度比较慢，耗时长，如下图所示:



迁移已有分片到扩容节点

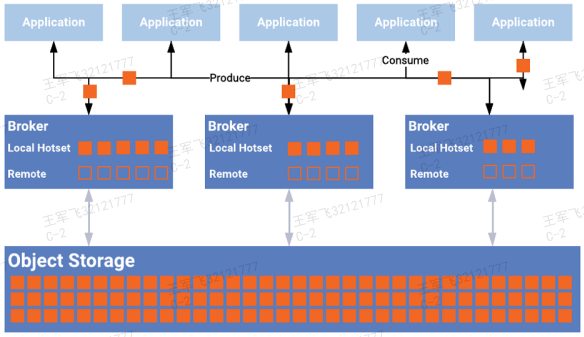
使用分层存储后，单个分片数据量变小，不仅能节约磁盘空间容量，还能降低节点磁盘读压力，最重要的是加快了分区数据的迁移速度，增强集群的扩容效率。



分层存储后的分区迁移

不仅如此，Kafka官方依靠分层存储实现了Kafka**无限存储**能力，架构如下图:

> *下图引用自confluent engineering blog

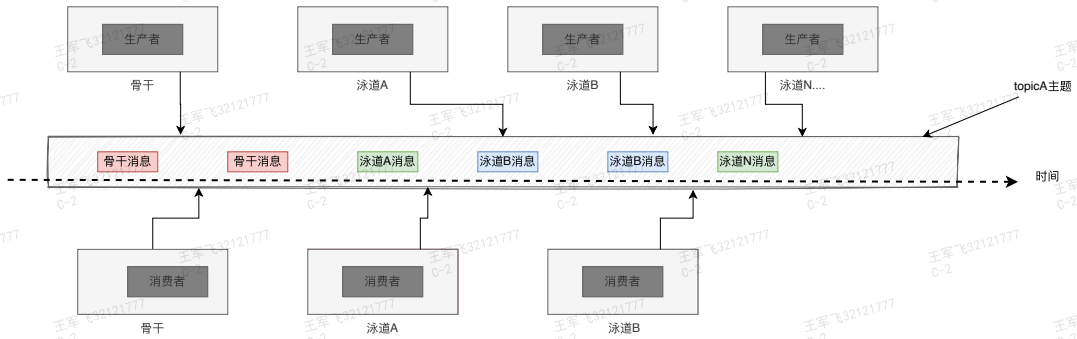


2.业务需求对Mafka架构的挑战

2.1 流量隔离、hash生产、消费模型对Mafka带来的资源压力

2.1.1 流量隔离带来的队列资源消耗

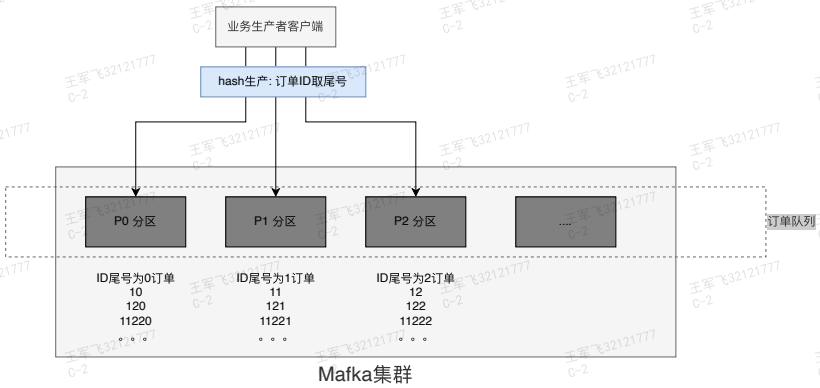
由于业务的发展的需求，mafka支持多重流量隔离方案，比如泳道，方便了QA人员同时开展多条测试链路，每个泳道的消费者只消费本泳道生产者生产的消息，如果下游没有泳道消费者，骨干的消费者需要把泳道的消息也消费掉，如下图所示，



类似的流量隔离方案，还有环境隔离，比如test/dev/prod/stage各类测试和生产环境；Set化隔离，比如North、East、West、South等各种Set。Mafka在满足这些流量隔离时，采用的是队列模拟，比如对于业务的某个队列，在环境隔离时，每个环境用一个队列去模拟，当多种流量隔离发生交叉时，prod环境下的Set，test环境下的泳道，又会产生多个底层队列。再加上其他的特性，比如死信，优先级队列等，这些功能都是通过底层队列来模拟的，**再次发生排列组合以后，队列资源耗费的比较快**。

2.1.2 hash生产带来的partition资源消耗

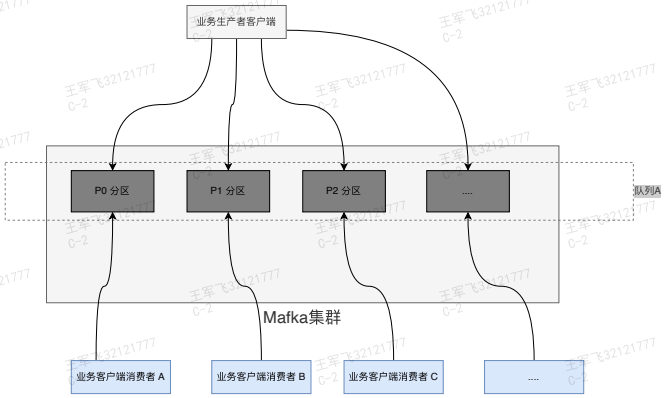
有些业务在向Mafka生产消息时，需要将特定类的消息放到一起，以便消费的时候，同一类消息由相同的消费者来消费



如上所示，业务在向订单队列生产消息时，希望某一类的订单集中放到一起，然后由同一个消费者来消费。Mafka在底层是通过分区来实现的，每一类的消息都放到同一个分区上，在业务消费的时候，这个分区的所有消息统一由某一个消费者来消费。这样就带来一个问题，**当业务hash的结果数据集类型比较多的时候，Mafka的paritition资源消耗的比较快。**

2.1.3 消费模型带来的partition资源消耗

Mafka的消息队列在被业务客户端消费时，受限于Mafka的消费模型，一个队列的一个partition只能被一个消费者消费



如上图所示，队列A有3个partition 分别是P0、P1、P2，业务有3个消费者，A、B、C分别负责消费P0、P1、P2。当消费逻辑耗时比较高，队列产生消息积压时，业务会要求加入多个消费者，这时Mafka就必须为

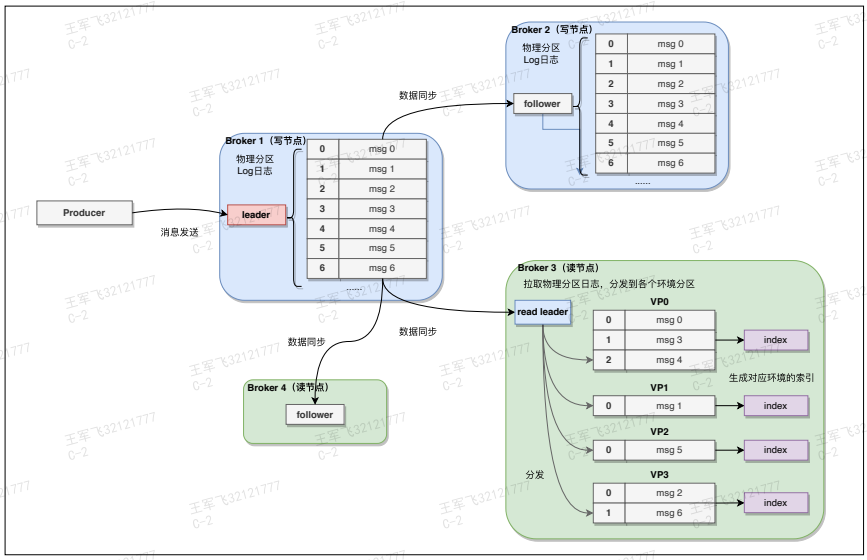
业务扩容partition，增加更多的分区，以便新创建的消费者能消费到消息。在这种场景下，Mafka partition资源的消耗并不是来自于队列流量的增加，而仅仅是业务消费者数量的增加，消费者数量和partition数量有紧密的耦合关系，

导致partition资源的消耗。

2.1.4 Mafka4 读写分离来解决

以上所说的三个问题，不管是队列资源的消耗，还是partition资源的消耗，都会让整个集群的partition数量的增长，从而消耗整个集群的容量。最明显的问题，就是整个集群的partition数量增加到一定程度后，用户的发送耗时会增加，集群吞吐量会降低。

对于这个问题，业界和Kafka社区并没有现成的策略和办法来参考。经过我们的测试和调研，提出了Mafka 4解决方案，通过队列合并来，分离读写节点来解决这个问题。



Mafka 4 概念模型

如上图所示，通过进入物理分区，虚拟分区，以及读写节点的概念，来重建整个Kafka 数据和存储模型。

这个方案不仅能有效的解决了上述所说的三个问题，还可以满足业务长期的发展需要，提升架构扩展性和效率：

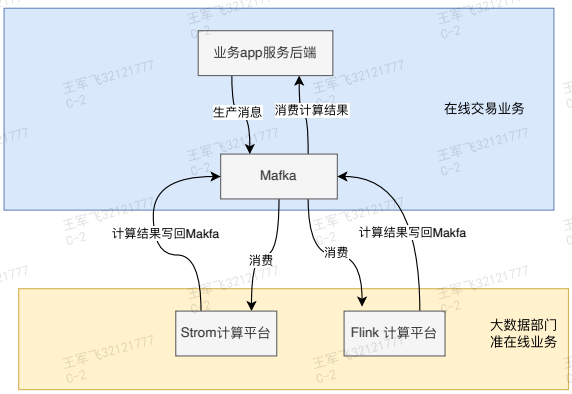
1. 合并各类流量隔离队列，来减少物理机队队列数量的膨胀，减少队列资源消耗。能让架构更灵活高效，解决将来业务在线上的各种流量隔离需求。
2. 解耦了partition数量和消费者数量之间的紧耦合关系，引入虚拟分区来解决这个问题。最终让业务对partition概念透明，业务有新的消费者上来后，分区会自动扩容，分配消息给新的消费者。
3. 可以合并小流量队列到一个物理队列上，也可以合并队列多个分区到一个物理分区上，缩减集群的分区和队列数量，大幅提升原有集群的容量。

2.2 实时计算业务的痛点

2.2.1 引入Kafka Stream 来构建一站式流式计算

随着业务的发展，实时计算技术近年来逐渐成熟起来，比如storm/flink一类实时计算框架，我们公司大数据业务也有相应的服务。

通常情况，在线业务在使用实时计算时采用的如下的架构：



业务先将数据发送到Mafka，然后再storm、flink平台上消费Mafka的消息，来做实时计算，计算完成后将数据再推送回Mafka，供在线业务是消费和展示。

①这种架构需要先将从Mafka搬迁到实时计算平台上，多了一次传输，浪费了一些时间和效率。

②而且在问题排查方面，storm和flink非常不友好，因为业务需要将自己的代码上传到这两个平台上，程序实际是在远程平台上执行的，而远程平台又是大集群、多用户，

调查问题、调试程序非常麻烦，效率低下。

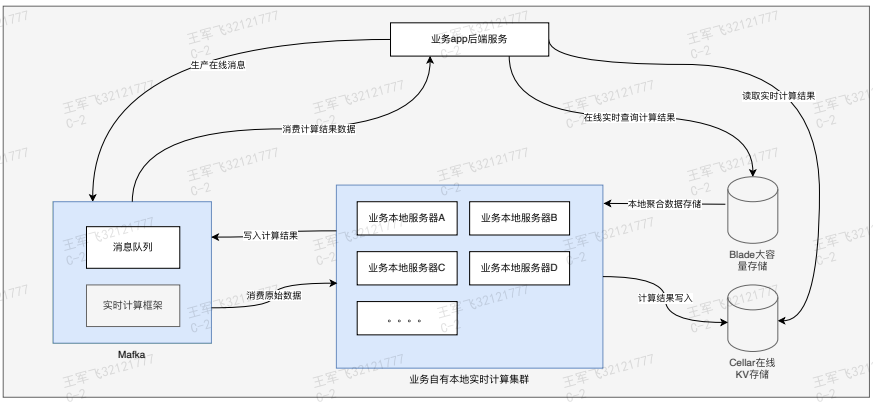
③另外，storm和flink自身有一定的复杂度，入门成本也比较高，对于一些只需要做一个简单窗口聚合计算的用户来说比较重，需要花时间和精力先学习。

实际上Kafka官方本身就支持轻量的实时流式计算服务，叫做Kafka Stream。Kafka Stream是集成在Kafka内部的轻量流式计算库，他跟Kafka集成在一起。

不同于Flink和Storm，Kafka Stream不是一个平台，不需要用户将代码打包上传到平台上，他只是一个简单的lib库，用户像写应用程序一样写流式计算服务，

编译打包成功后，运行在用户自己的机器上。

下图是一个Mafka集成Kafka Stream后的部署架构图：

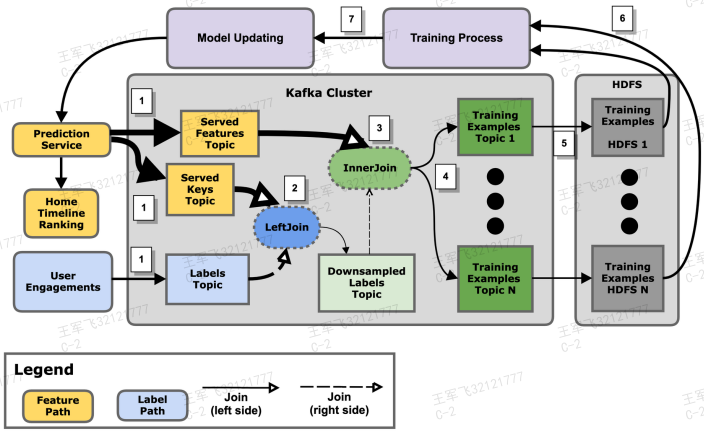


业务app后端服务将在线数据写入Mafka，然后通过自有的实时计算服务消费这些数据，计算完成后再把计算结果写回Mafka，供app后端服务来使用。

业务自有的实时计算服务是自己的服务，像普通的后端服务一样构建和部署。在实时流式计算过程中还可以使用到大数据存储Blade，计算结果也可以存储在Cellar中供业务app后端服务来存取。

Kafka Stream目前使用也很广泛，大厂如twitter就将自己首页的推荐引擎也搬迁到了Kafka stream上，架构如下图：

*下图引用自twitter engineering blog



总结起来，引入Kafka Stream的优势：

1. 实时流式计算可以在Mafka一站式解决，减少数据的搬运，降低整个实时计算链路的延迟
2. KafkaStreams是本地lib库，开发、调试方便
   a. 意味着RD可以在本地开发和测试自己的Kafka Stream流式任务，Strom/Flink等需要将服务部署到集群上，开发者很难了解框架的具体运行方式，从而使得调试成本高，并且使用受限.
2. Kafka Stream非常的轻量级，可以应用到微服务、IOT等实时流式计算以及EDA架构场景下：
   a. Strom/Flink都是计算框架，需要部署一个集群，将自己的流式作业上传到集群处理，kafaStreams是lib库，业务程序集成后就可以可开始流式计算。
3. 接入方便，使用成本低
   a. Mafka在公司内广泛使用，业务已经将消息和数据发往了Mafka，如果Mafka推出stream流式计算，业务只要升级Mafka client版本，既可拥有流式计算能力，相对业务来说，使用流式计算技术的成本非常低。

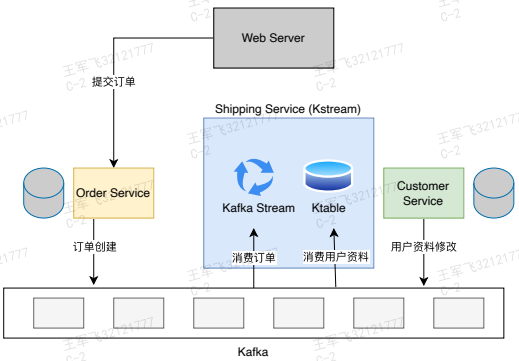
2.2.2 引入Kafka Stream 支持EDA(事件驱动架构)

事件驱动架构(Event Driven Architecture)是近两年流行起来的新的架构方式，相比传统的请求驱动架构(Request Driven Architecture)，他有以下优势：

1. 模块之间松散耦合
   由于生产者和消费者两个模块通过消息队列解耦，消息生产者不需要知道消费者是谁，在不在线，可以一直生产。后续业务需求增加新的功能，不影响原有系统。
2. 异步
   生产者不需要等待消费者处理，只需要发送完消息即可返回，实现生产和消费的异步结构。
3. 可扩展性强
   由于模块之间松散的耦合，各自可以独立扩展，互不影响。
4. 可恢复
   因为模块之间通过消息来通知和驱动，消息又可以回放，因此当消费者宕机后，可以重新回放之前的消息来恢复系统状态。

Kafka Stream对 EDA 架构具有天然的支持优势，如KTable，KSQL，各类Material View等。

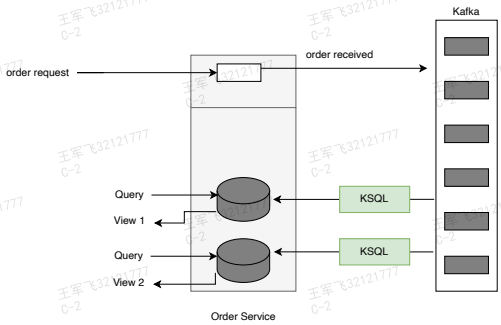
如下图，依靠事件传输来驱动整个业务的完成：



1. 整个系统包含订单系统、发货系统和客户系统，三个系统各自有自己的数据库，系统之间通信使用Kafka消息。

2. 用户提交完订单后，发送订单创建消息
3. 发货系统消费订单信息，提供发货支持
4. 用户系统将用户资料变更发送给Kafka
5. 发货系统消费用户变更资料，构建为全量的本地用户资料，存储在Ktable里
6. 发货系统本身是一个流式运算系统

同时还可以使用KSQL 产生各类预定义视图，如下图示，使用CQRS模式增强读写操作的扩展性:



1. 创建订单后，发送消息到Kafka
2. 在查询面，可以使用KSQL的功能，创建多种(Materialised View)实体图，因为Kafka的Pub/Sub模型，实体图的个数可以很多。

3.建议发展路径

3.1总结长期计划

综合以上分析，总结MQ长期要规划的项目有以下几点

编号	项目	收益	备注
1	接入k8s，提升Mafka集群的扩展能力	接入容器化，增加k8s operator的使用，增强Mafka broker的横向扩展能力。	
2	去除ZooKeeper依赖，提升broker的扩展性	1. 直接提升了broker的扩展性。 2. 降低组件的复杂度 3. 减少部署和运维 4. 解决Mafka 容灾2.0特殊情况下的集群脑裂问题	
3	使用分层存储，提升存量分区的迁移速度	提升Mafka Broker集群的存量分片迁移能力	
4	建设Mafka 4 读写分离集群	1. 支持业务更多类型的流量隔离需求 2. 解耦分片和消费者数量紧耦合关系，降低集群分片资源消耗速度 3. 提升集群的整体吞吐量 4. 合并小流量队列，减少集群分片的消耗，提升集群容量	
5	引入Kafka Stream 来构建一站式流式计算	1. 实时流式计算可以在Mafka一站式解决，减少数据的搬运，降低整个实时计算链路的延迟 2. KafkaStreams是本地lib库，开发、调试方便 3. Kafka Stream非常的轻量级，可以应用到微服务、IOT等实时流式计算 4. 接入方便，使用成本低	
6	引入Kafka Stream 支持EDA(事件驱动架构)	1. 对业务的EDA架构应用提供完整的支持 2. 降低业务应用架构的耦合度	

3.1建议发展路径时间表

2021年Q3 完成2%的集群容器化

2021年Q4 完成5%的集群容器化，开始构建基于hulk k8s operator的自动化运营系统；完成Mafka 4 release版本发布

2022年Q1 增加2%的集群容器化，完成构建基于hulk k8s operator的自动化运营系统

2022年Q2 分层存储方案研发，远程存储接入Mstore

2022年Q3 完成1%的线上集群分层存储；去除ZooKeeper依赖，提升broker的扩展性

2022年Q4 增加3%的线上集群分层存储，完成15%的集群容器化

2023年Q1 引入Kafka Stream 构建一站式流式计算，支持业务EDA架构应用

2025年Q1 完成20%的线上集群分层存储，完成50%的集群容器化

参考:

1. Making Apache Kafka Serverless: Lessons From Confluent Cloud
2. Kafka，confluent，The Cloud-Native Evolution of Apache Kafka on Kubernetes
3. Kafka，confluent，Making Apache Kafka Serverless: Lessons From Confluent Cloud
4. Infinite Storage in Confluent Platform