漫谈 Microservice

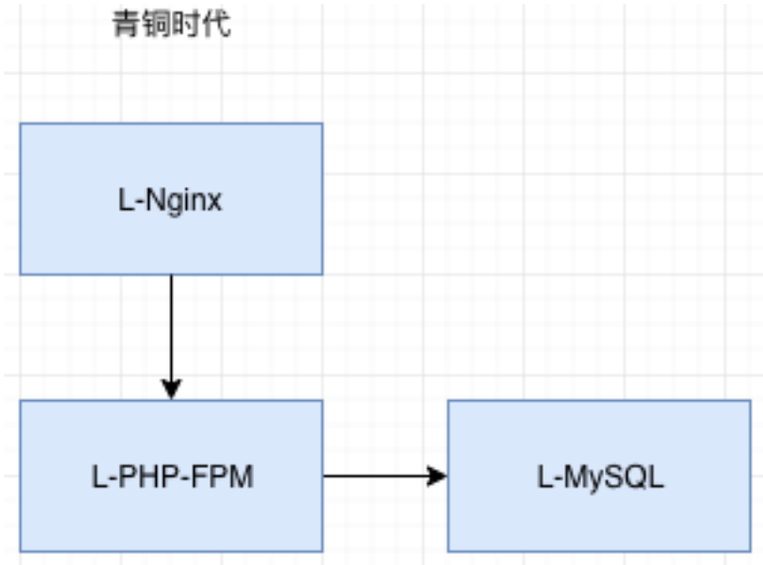
石器时代

- 创业初期我们大部分精力都会投放在对产品的探索阶段，这时候的产品没有用户，没有流量，我们通常也不会太在意软件架构问题。基本上就一台服务器能干所有的事情。

青铜时代

这个时期遇到的问题

- 受PHP-FPM架构特性影响Woker数量经常不够用。(调大FPM Woker的数量，将FPM放在一台独立的VPS上)
- MySQL受到不规范编码(逻辑代码慢查询，循环轮训数据库等等)，导致查询时间变长，返回结果慢。(清理代码，将MySQL独立放在一台VPS上)



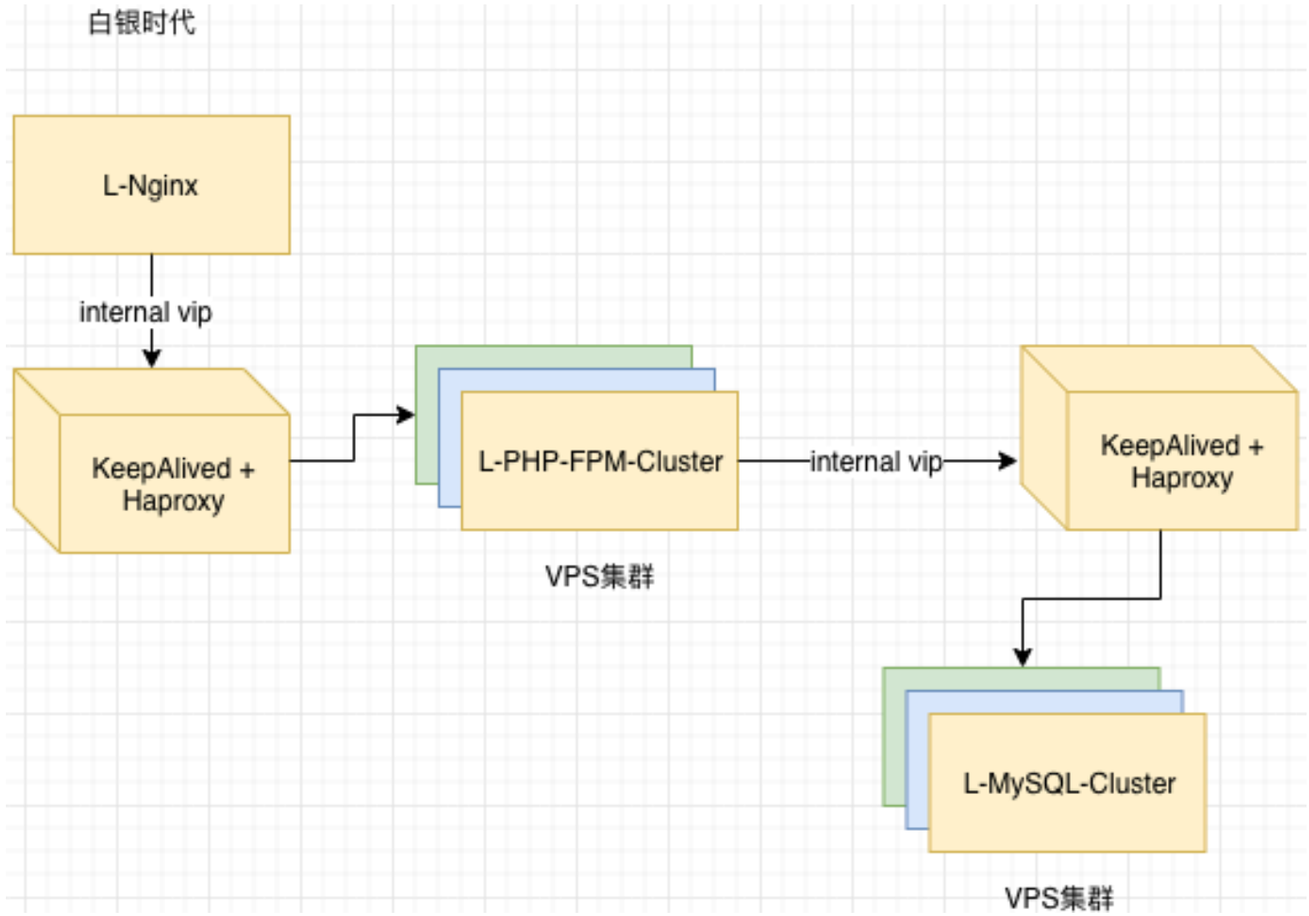
思考

某个服务器故障怎么办?

白银时代

1. 通过KeepAlived + VIP + VPSCluster 来解决当某个服务器节点故障时，快速恢复服务可用。

2. 通过HAProxy + VPSCluster来做到流量负载均衡，尽量随机的将峰值流量，分摊到集群中。



思考

当PHP-FPM软体应用某个功能QPS较高，影响到了整个软体应用的SLA，怎么做?

黄金时代

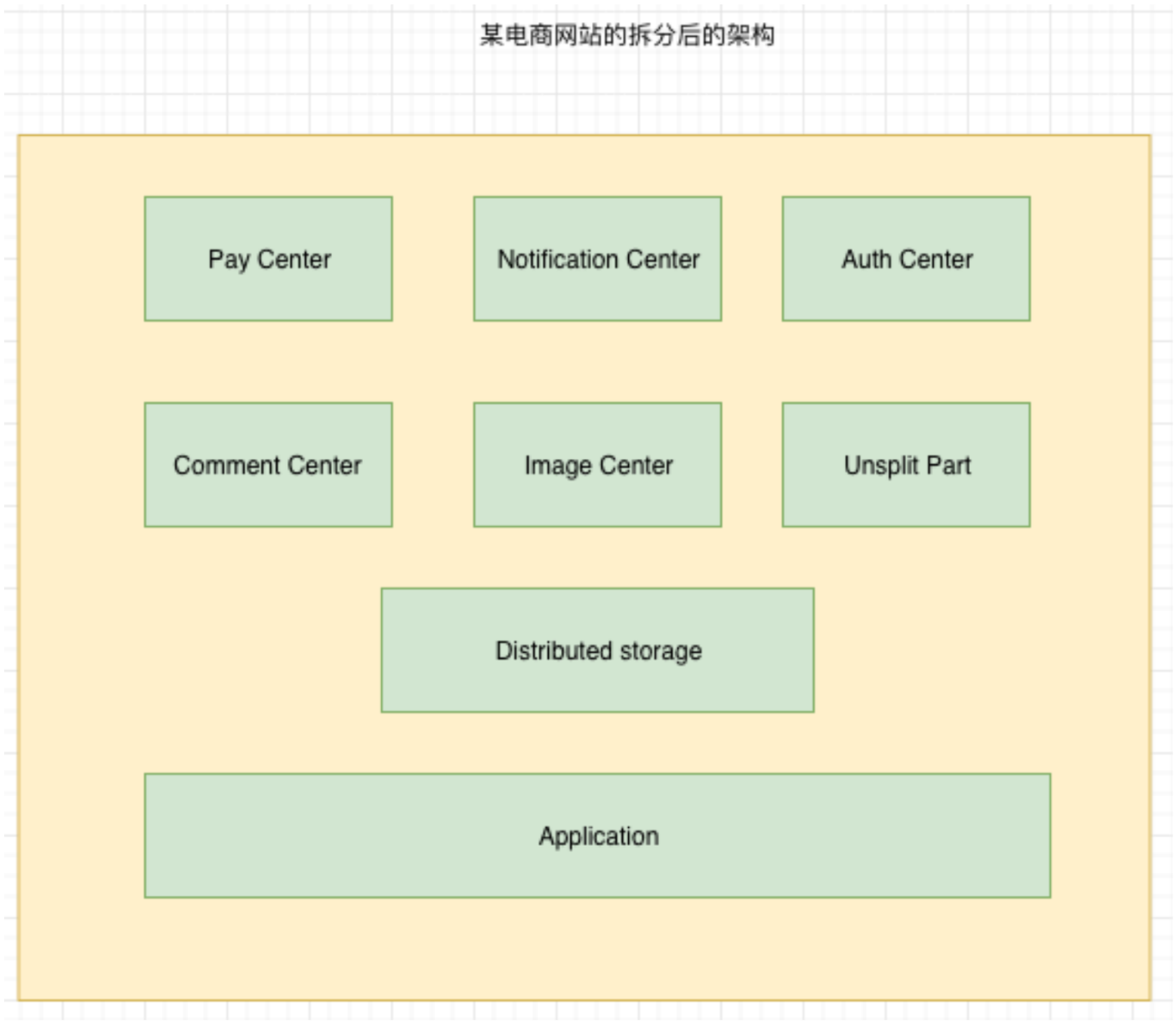
- 我们从一个巨石应用(Monolith Application) 拆分成多个可以独立运行的服务，这就叫做微服务(MicroService)。

- 这些被拆分的服务，有可能被单独放在一个服务器集群中，也有可能由于没有什么规划，然后被分散的放在不同的机器中。

- 当我们沿着白银时代的架构发展过来时，IAAS基本没有任何变化，HA架构也没有太大的变化，唯有软体被拆分成多个应用。

- 缺点：应用软体被拆分后`系统熵`将会变大，维护代价也会越来越大。

- 优点：当我们遇到某个服务峰值流量过大时，只需要水平扩展这个应用就好了，不用再扩展整体应用。



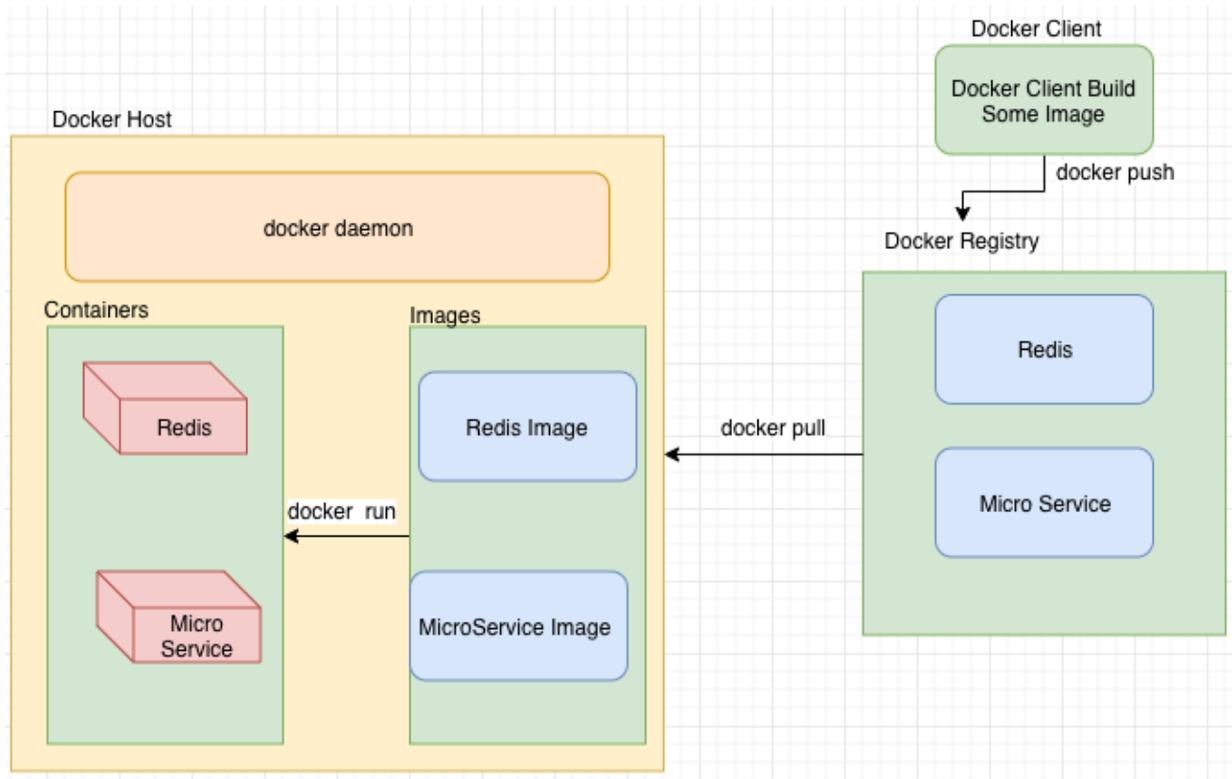
思考

1. 当升级服务器集群的环境组件时，我们需要一个一个升级集群中的服务器，有没有更好的办法?

2. 我们如何在不同的系统，不同的发行版本，来保证微服务环境一致性问题?

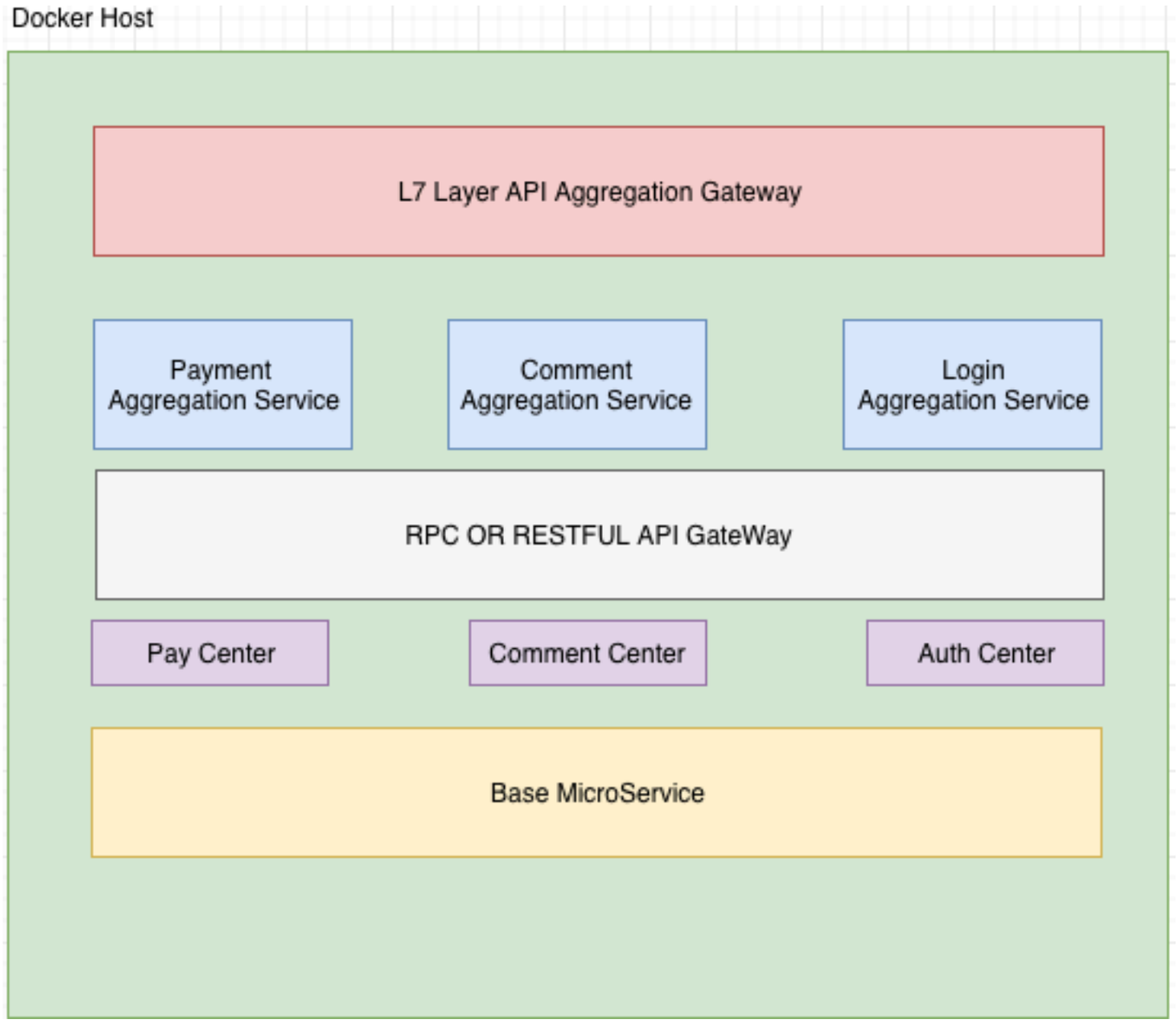
铂金时代(容器化)

- Docker 是一个基于Linux Kernel CGroup 和 Namespace 特性做的一个操作系统级别的沙箱环境。
- 我们可以在Docker里面跑一个发行版Linux系统，也可以跑一个体量非常小的busybox。
- 一旦我们构建好镜像，并把推送到Docker Registry，我们就可以很快的从Registry拉取镜像快速部署我们的应用
- 当我们的MicroService Docker镜像，需要升级环境组件时，我们只需要一次性构建好一个新的环境镜像，然后其他服务器拉取就行了。
- Docker Container 启动比普通的VPS要快很多。
- Docker 的RuntimeDriver 默认是runc, 未来可能会被gvisor所取代。



铂金时代(架构)

- 假设所有的微服务都不是有状态的，此时我们的微服务都跑在Docker里。
- L7层是HTTP层，对外提供一个聚合服务的网关。
- 蓝色区域是聚合服务，一般是业务层
- 业务层通过RPC或者RESTFUL API来选择自己需要哪些底层的基础服务
- 此时的软体应用架构已经相对于上一个时代，变得透明，并且易于部署，易于管理。

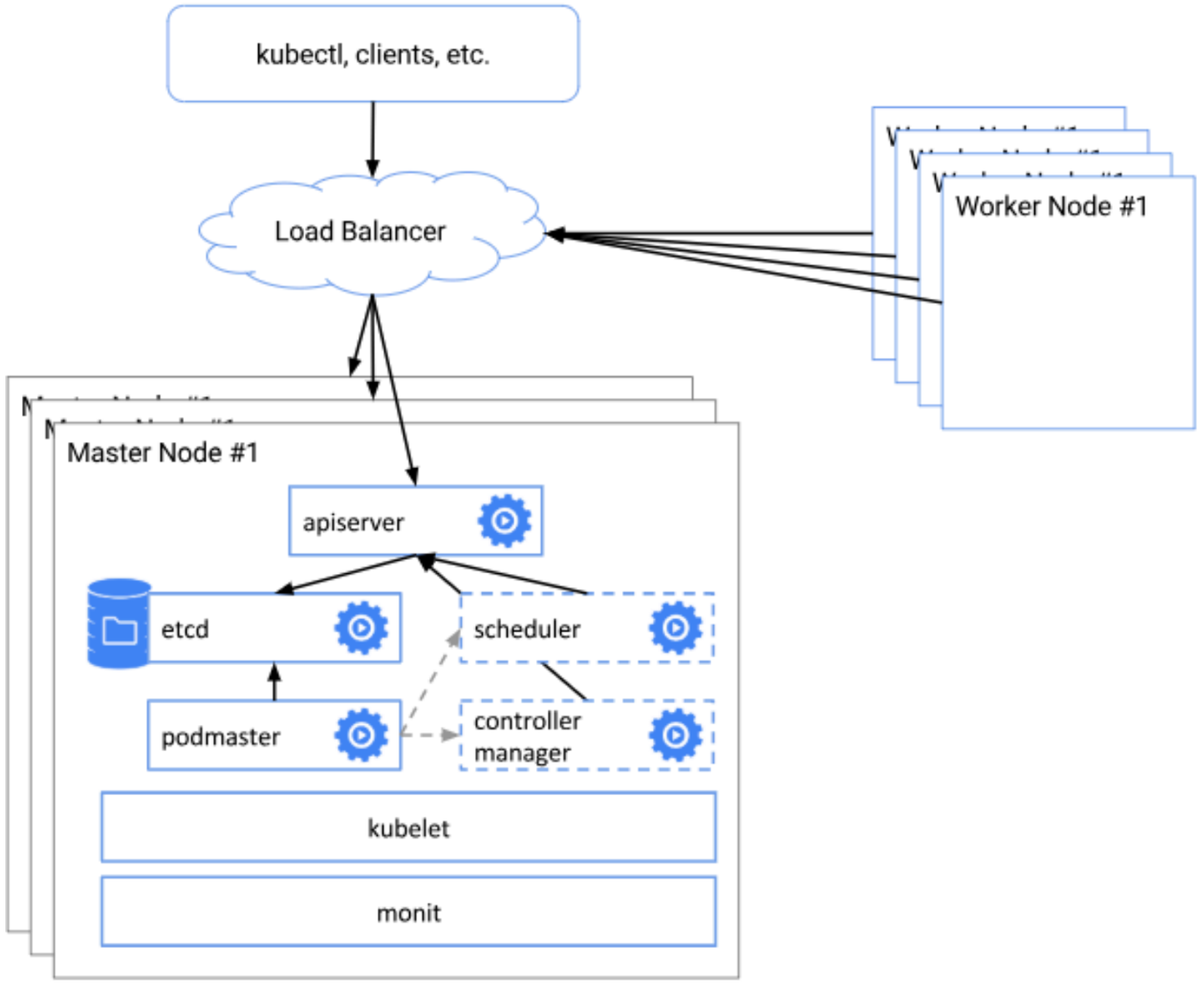


思考

- 此时的L7流量进入了微服务集群内部，并且触发了一个不是很容易查到的BUG, 面对黑盒化的服务我们该怎么查问题?
- 我们该如何聚合日志?
- 当微服务多了，我们该如何管理Docker集群?
- 我们如何轻松的控制L7层的流量?
- 我们如何0宕机的情况下完成CD(持续集成)的工作?
- 我们如何轻松扩容调度集群?
- 如何监测微服务集群健康活性?
- 假如某个微服务集群不健康，如何做到应用层的FailsOver?
- 如何动态伸缩某个微服务集群来应对洪峰流量?

钻石时代(容器编排)

- K8S 是一个基于Google Borg模型开发的容器编排工具，一个典型的C/S架构。
- 右图是一个高可用的K8S集群。
- K8S Master是集群大脑，负责容器调度，动态水平扩容，微服务热滚动升级，负载均衡，等等。微服务集群在Woker节点。
- K8S是一个半自动化运维工具，他的功能不包含：日志收集，CI(持续集成) && CD，流量控制。



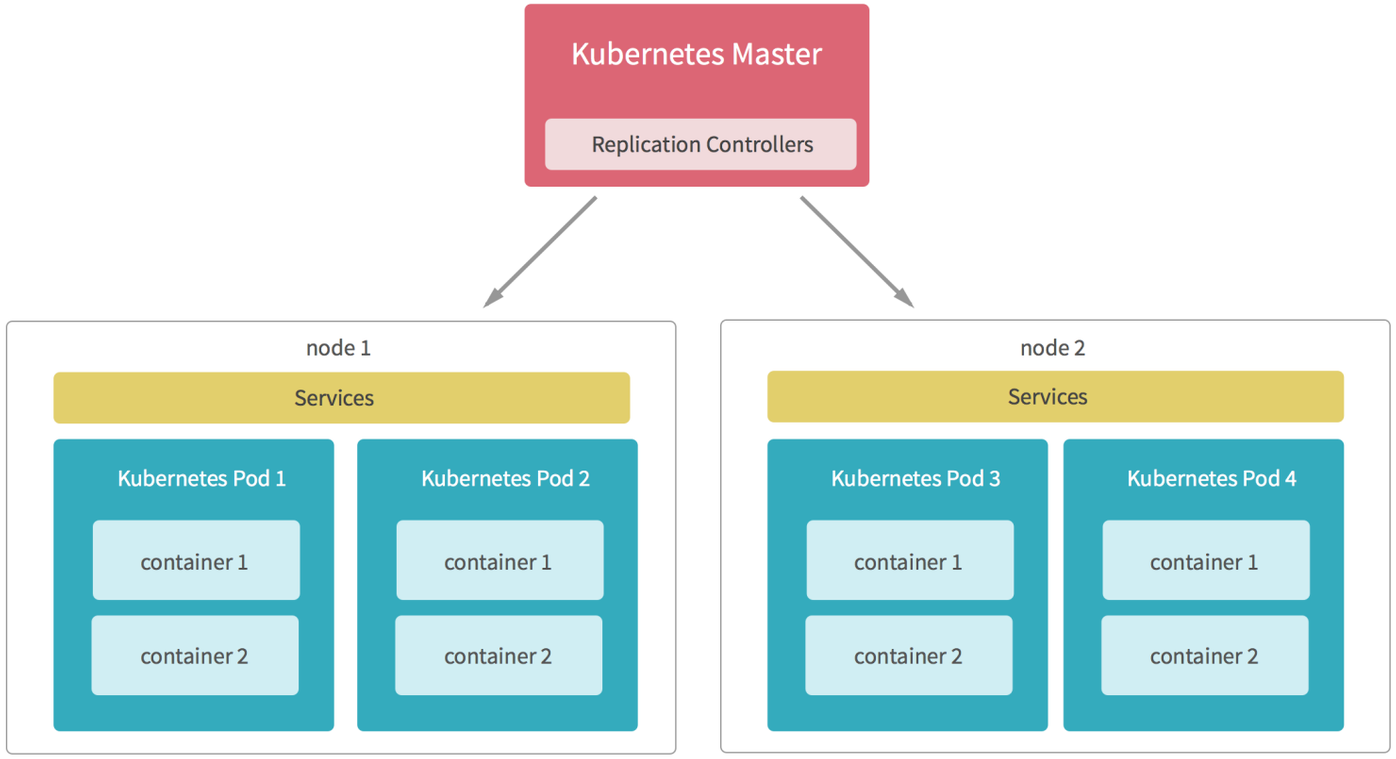
钻石时代(流量控制)

- 右图是一个简单的K8S内部结构。

- 假设有一个这样的场景，某个产品有新的功能，但是需要在生产环境做A/B Test,这时候我们需要一个Pre Production 环境我们该怎么做?

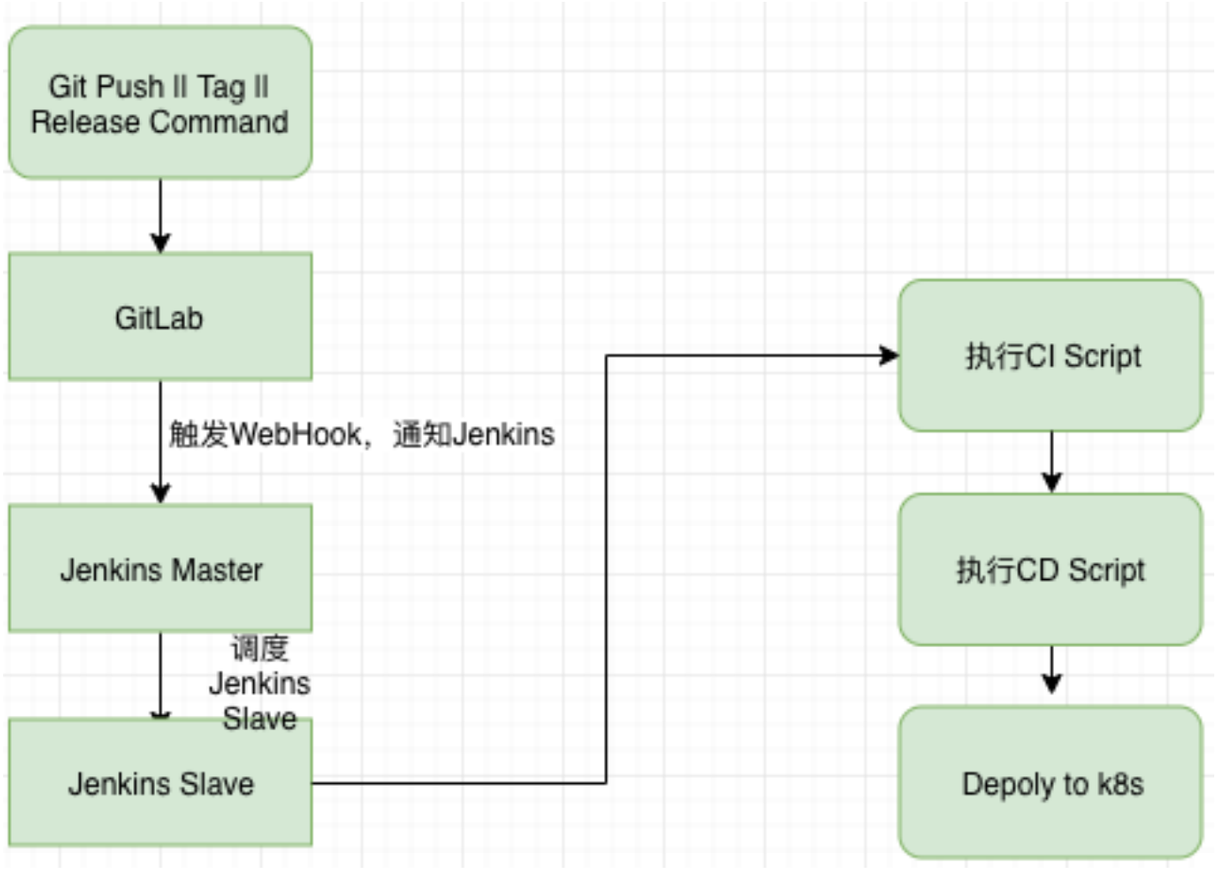
- 我们将每个POD看成一个网格，每个POD都有一个标签，我们对部分的POD打上TEST标签，部分打上PRODUCTION标签，剩下的事情就是控制L7层流量，部分引入PRO-DUCTION，部分引入TEST，这样就好了。这就是Server Mesh的一个基本功能。

- 常见的控制L7层的K8S组件有：Istio，Linkred。



钻石时代(CI&CD)

- CI & CD技术选型：Jenkins
- Jenkins 跑在K8S集群里。
- 当GitLab收到PUSH， TAG，MERGE等命令时会触发WebHook，然后通知Jenkins Master, 然后Master将会在某个节点启动一个Jenkins Slave Pod，用来跑CI CD。
- CI: Jenkins Slave 通过Script，安装依赖，跑单元测试，构建Docker镜像，Push Docker镜像到远程仓储。
- CD: 执行kubectl命令将新的镜像部署到K8S集群。

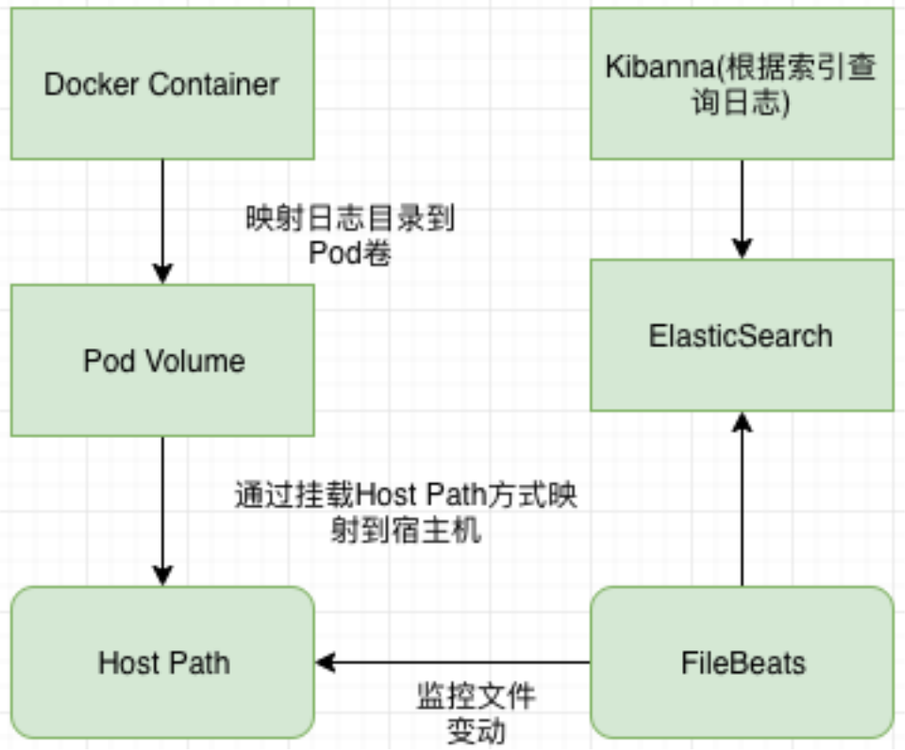


钻石时代(日志聚合)

- 日志聚合的技术选型是： ElasticSearch + Filebeat + Kibana。

- 日志这样就能采集到一起，这时候还是会有一个问题，当我们把日志挂载到宿主机目录的时候，当多个Pods对一个日志文件读写，会有并发写入安全性问题，这时候我们需要保证每一个服务产生的日志log文件名不一样。

- 日志聚合服务看起来似乎做完了，但是可以肯定的说这仅仅是一个开始，我们只是很粗糙的完成了，只是做了简单的可查询。



这个时代就这样结束了?

- 我们该怎样传统的巨石型SaaS应用迁移到K8S？值不值得迁移?

- 程序员在写分布式应用的时候应该注意什么?

- 你们的问题呢?

END