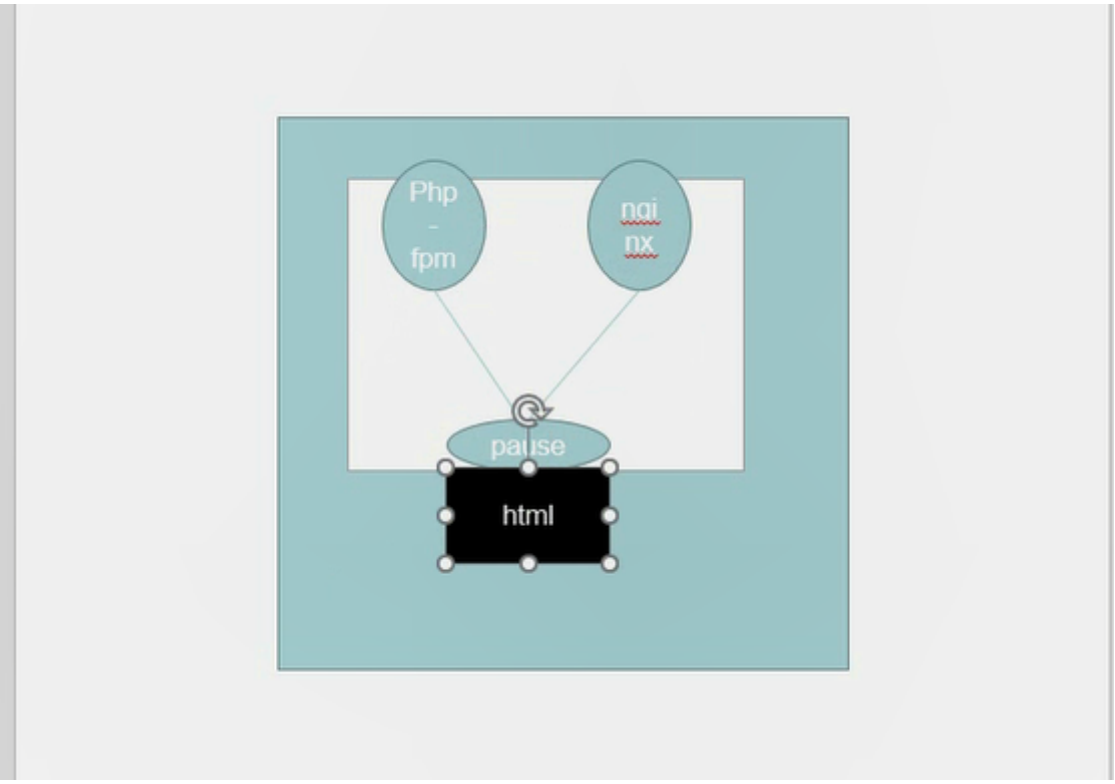
2 kubernetes - 基础概念

2.1 pod 概念

自主式POD：不是被控制器管理的pod。一旦死亡就不会再重生

控制器管理的POD：就是被控制器所管理的POD。

2.1.1 自主式POD的基础概念



容器会共用pause的网络栈，也就是说这两个容器就没有他的独立地址了他们都是共同使用pause的地址、共用他的存储卷

Pause 网络栈共享：

首先我们要定义一个 POD，就会先启动第一个容器，只要运行一个POD这个容器就会被启动、这个容器叫PAUSE（只要是有POD这个容器就会被启动）、这个容器启动成功之后我假设在这个POD里启动了两个容器，也就是说一个POD里会被封装多个容器。然后这两个容器会共用pause的网络栈，也就是说这两个容器就没有他的独立地址了，他们都是共同使用pause的地址、共用他的存储卷、但是这两个容器之间进程不隔离（也就意味着、这里一个容器运行的是php-fpm一个容器运行的是nginx，如果这里nginx想要反向代理fpm的话只需要写locahost9000（这个是php服务的9000端口）即可，原因是这两个容器都是共享的pause的网络栈、也就是在同一个pod里容器之间的端口不能冲突的。）

Pause 存储卷共享：

假如这个pod会挂在一个存储上，同理这两个容器（php-fpm、nginx）都会共享pause的存储。因为他们两个分别都需要共享html的内容。

也就意味着在同一个pod里及共享网络又共享存储卷。

2.1.2 控制器管理的POD的基础概念

```
* ReplicationController & ReplicaSet & Deployment
    > HPA（HorizontalPodAutoScale）
* StatefullSet
* DaemonSet
* Job，Cronjob
```

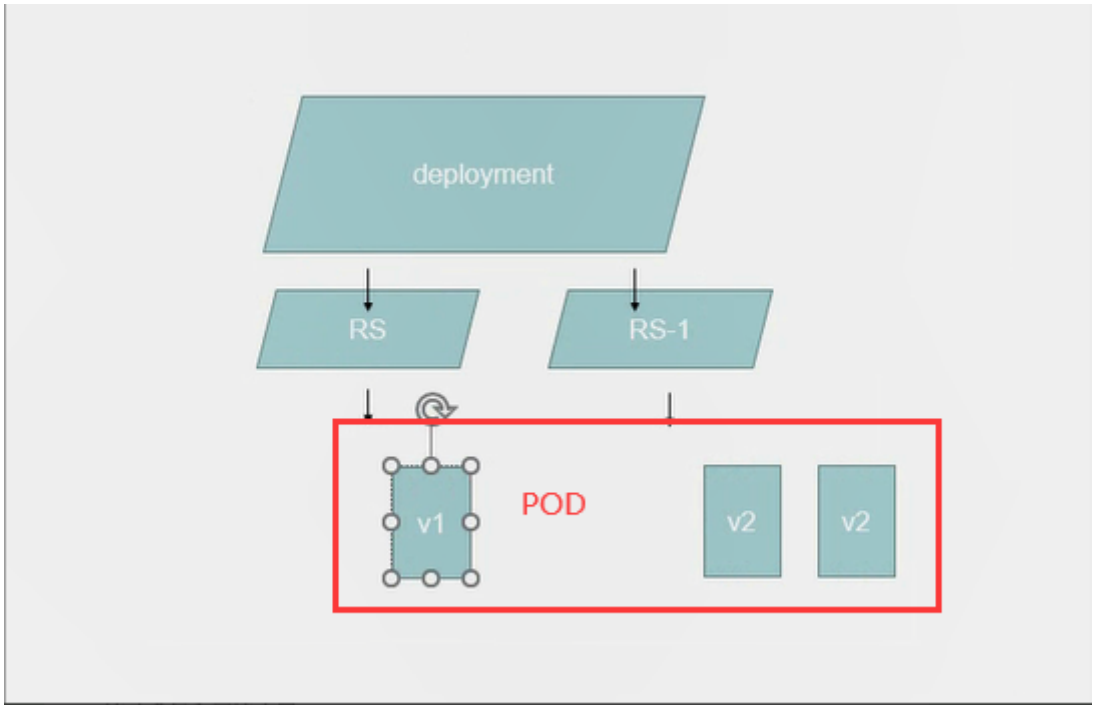
Replication Controller（简称RC）& ReplicaSet（简称RS） & Deployment 这是三种管理器类型、因为这三种有一定的重合性所以放在一起讲。

Replication Controller： 用来确保容器应用的副本数始终保持在用户定义的副本数，即如果有容器异常退出，会自动创建新的 Pod 来替代；而如果异常多出来的容器也会自动回收。在新版本的 Kubernetes 中建议使用 ReplicaSet（在K8s新版本中官方抛弃RC采用的都是RS） 来取代Replication Controlle（RC）

ReplicaSet 跟 ReplicationController 没有本质的不同，只是名字不一样，**并且 ReplicaSet 支持集合式的selector（选择器）**

虽然 ReplicaSet 可以独立使用，但一般还是建议使用 Deployment（调度） 来自动管理 ReplicaSet，这样就无需担心跟其他机制的不兼容问题（比如 ReplicaSet不支持 rolling-update（滚动更新） 但 Deployment （调度） 支持）

2.1.2.1 Deployment（调度）原理



deployment 为 pod 和 replicaset 提供一个声明式定义（declarative）方法，用来代替以前的 replication controller 来方便管理应用。典型的应用场景包括：

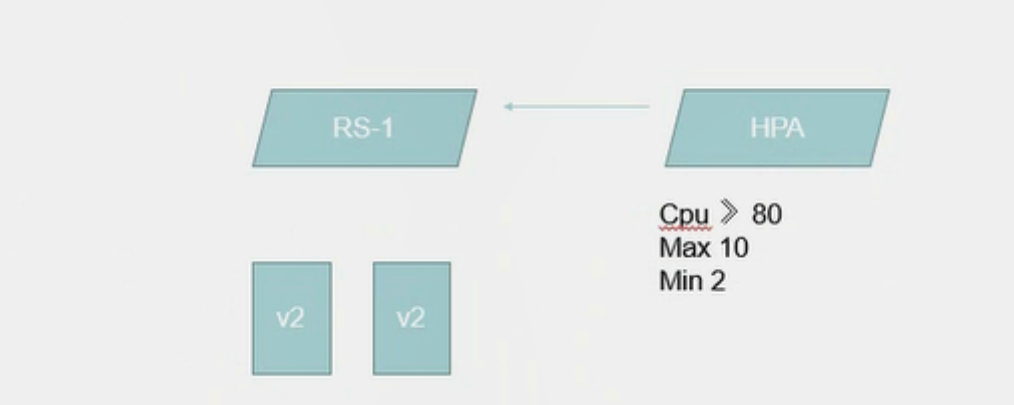
- 定义 deployment 来创建 pod 和 replicaset
- 滚动升级和回滚应用
- 扩容和缩容
- 暂停和继续 deployment

Deployment在创建出来以后呢会去创建一个RS，也就意味这RS不是我们自己定义得，而是deployment定义得，deployment再去负责创建我们对应的pod。比如上图创建了三个pod，如果有一天我们让deployment把镜像的V1版更新成V2版本。他会新建一个RS（这里比如是RS-1）然后我们就将RS的V1滚动更新到RS-1的V2版已到达滚动更新的状态，并且还可以回滚，所谓的回滚就是当发现更新到V2版本有一定的小BUG我们可以支持回滚，只见我们的rolling-update（滚动更新）就会回到RS新建一个老旧版本的V1然后把RS-1的V2删了以此类推，**原因是deployment再滚动更新之后这个RS并不会被删除而是被停用，当我们开始回滚的时候就会将老旧版本的RS开始启用，然后逐渐达到**

deployment（调度器）想要达到的预期状态。以上是deployment的原理，deployment需要去创建RS达到创建pod的能力

2.1.2.2 HPA（水平制度扩展）

Horizontal Pod Autoscaling 仅适用于仅适用于 Deployment 和 ReplicaSet，在 V1 版本中仅支持根据 Pod的 CPU 利用率扩缩容，在 vlalpha版本中，支持根据内存和用户自定义的 metric（公制的）扩缩容。



我现在运行了一个RS-1，RS-1下面管理两个pod，然后我再定义一个HPA，这个HPA定义是基于这个RS-1来定义的，当我们的CPU大于80的时候就进行扩展，扩展的最大值是10个最小值是两个，也就以为这HPA会去监控这些POD的资源利用率，当CPU达到80的时候HPA就会去新建出来新的pod直到达到他HPA的最大值10个，也就是cpu只有在大于或等于80的时候HPA才会去创建pod，如果不满足这个条件就不会再进行创建。一旦利用率变低之后这里的pod就会被回收。但是最小只能删到2因为这里定义了最小个数为2，所以HPA就会达成一个水平制度扩展的目的。

2.1.2.3 StatefullSet 解决有状态服务的问题

StatefulSet 是为了解决有状态服务的问题（对应 Deployments 和 ReplicaSets 是为无状态服务而设），其应用场景包括：

- 稳定的持久化存储（就是pod在死亡以后再去调度一个新的pod回来的时候里面的数据也不能丢失），即 Pod重新调度后还是能访问到相同的持久化数据，基于 PVC来实现来实现。
- 稳定的网络标志，即 Pod 重新调度后其 PodName（pod名称）和 HostName（主机名称）不变，基于 Headless Service（即没有 Cluster IP 的 Service）来实现。这是防止在集群里面定义了一个pod名称去调用，结果出现了一个新的pod顶替以后名字变了需要重新写入这是不需要的，因为他会实时的稳定这个网络表示
- 有序部署，有序扩展，有序部署会分为扩展和回收阶段，所以称为有序部署和有序扩展。即 Pod 是有顺序的，在部署或者扩展的时候要依据定义的顺序依次进行（即从 0 到 N-1，在下一个 Pod 运行之前所有Pod 必须都是Running（运行）和 Ready（准备）状态），基于 init containers 来实现
- 有序收缩，有序删除（即从 N-1 到 0）

2.1.2.4 daemonset（守护进程）

DaemonSet 确保全部（或者一些） Node 上运行一个 Pod 的副本。当有 Node 加入集群时，也会为他们新增一个 Pod 。当有 Node 从集群移除时，这些 Pod 也会被回收。删除 DaemonSet 将会删除它创建的所有 Pod

使用 DaemonSet 的一些典型用法：

- 运行集群存储 daemon，例如在每个 Node 上运行 glusterd 、ceph 。
- 在每个 Node 上运行日志收集 daemon，例如 fluentd 、logstash 。
- 在每个 Node 上运行监控 daemon，例如 Prometheus Node Exporter

2.1.2.5 Job （工作），Cronjob（定时任务）

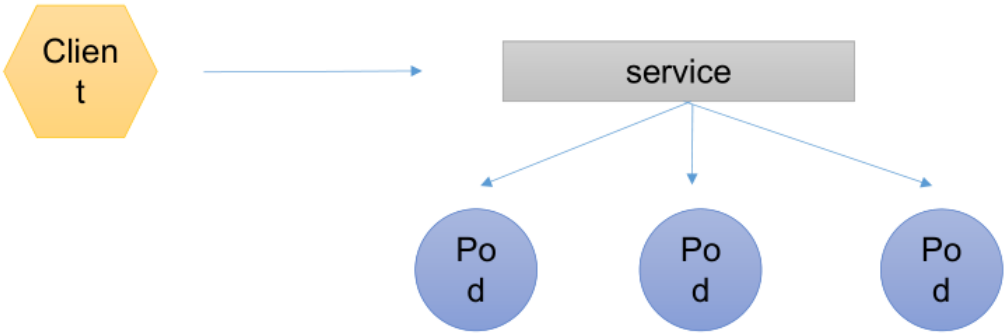
Job 负责批处理任务，即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束

Cron Job 管理基于时间的 Job，即：

- 在给定时间点只运行一次
- 周期性地在给定时间点运行

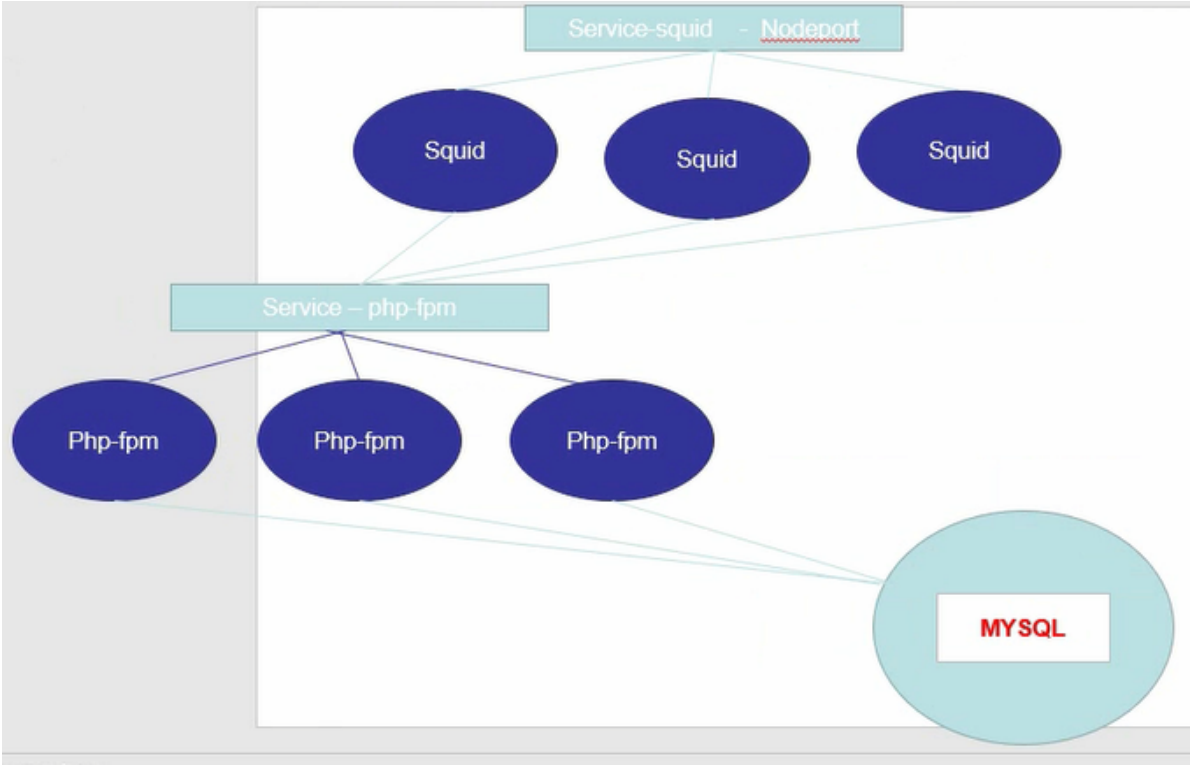
Cron job：简单来说就是可以在特定的时间重复执行。

2.2 服务发现



所谓的服务发现就是我们的客户端想要去访问我们的一组pod，并且被访问的pod必须要有相关性，因为只有是有相关性才会被我的service收集到。并且service去收集我们的pod是通过我们的标签去选择到的。选择到以后这个service会有自己的ip和端口，这时候我们的客户端（client）就可以去访问service的IP加端口。而间接的访问到所对应的pod，并且这里是有一个RR（轮询）的算法存在的。

2.2.1 pod 与 pod 之间的通讯方案



上图解析：

就是我们的 apache 加我们的 fpm 模块。然后前面是一些缓存服务器 squid（缓存服务）。最后面就是我们的MySQL，而 squid（缓存服务）前面肯定是需要一个负载服务器的这里我就用 LVS。

也就意味这首先我们要构建一个 apache 加入到 fpm 的结构集群，然后在构建 mysql 集群正在构建 squid 集群。假如我们想把这个结构放在我们的 k8s 集群中运行的话。

MySQL 需要运行成一个 pod，mysql 封装成一个 pod 之后呢我们还有 apache+fpm，并且 apache+fpm 这三个都是类似的所以我们可以在 deployment（调度器）上面去创建。Deployment 会指定 apache+fpm 的副本数目为三个副本。这样就会有三个不同的 apache+fpm 的 pod 存在了。

然后在往上我们会发现有三个 squid（缓存服务），这三个 squid（缓存服务）我们也可以交个控制器去控制。然后 LVS 就可以靠他本身的功能将这个集群负载了。

而我们在 php-fpm 和 squid 的中间再加一层 service - php-fpm。这个 service - php-fpm 会绑定 php-fpm 的标签选择进行绑定，那 squid 想要进行反向代理的设置的话就不需要写这三个 php-fpm pod 所对应的IP地址了。而是写的 service - php-fpm 的IP地址，因为 php-fpm 这三个 pod 如果死掉的话就会从新生成ip地址，而间接都就要修改 squid 反向代理的配置，以避免不必要的问题。这样的话 squid 只要将 IP 指定在我们的 service - php-fpm上面即可并且 php-fpm 是三个 pod，MySQL 也是 pod 他们之间会出现一定的关联，比如我们将mysql部署在StatefullSet 里面这时 mysql pod 的名称就不会变，那我们可以通过他的名称去固定在我们对应 pod 上，因为K8S 内部是一个扁平化的网络，所以容器之间是可以互相访问的所以我们的 php-fpm 里面写 mysql 的信息是没有问题的，对于这三台 squid 来说用户想要访问，我可以在创一个 service - squid 与我们的 squid 进行绑定，通过去判断我们的 squid 的相关一些标签进行确定。

这样的话我们只需要将我们的 service-squid 暴露在外面即可，service 其中有一种暴露模式叫做我们的 nodeport 。这样我们就可以将这个集群部署在K8S集群中了。这个集群也是我们的pod与pod之间的通讯方案。

2.3 网络的通讯方式

2 网络通讯方式

2.3.1 网络通讯模式

Kubernetes 的网络模型假定了所有 Pod 都在一个**可以直接连通的扁平的网络空间中（在 K8S 中所谓的扁平化网络空间就是所有的 pod 都可以通过对方的ip"直接达到"就是 pod 认为他自己是直接到达的，其实底层有一堆的转换机制存在）**，这在 GCE （Google Compute Engine 谷歌的云服务）里面是现成的网络模型，在 Kubernetes中假定这个网络已经存在（也就意味了如果我们想在自己的集群中去构建K8S的话就先要去解决扁平化的网络空间）。

而在私有云里搭建 Kubernetes 集群，就不能假定这个网络已经存在了。我们需要自己实现这个网络假设，将不同节点上的 Docker 容器之间的互相访问先打通，然后运行 Kubernetes

2.3.2 同一个 pod 内的容器之间通讯

同一个 Pod 内的多个容器之间： lo （回环网卡）

各 Pod 之间的通讯： Overlay Network（重叠网络）

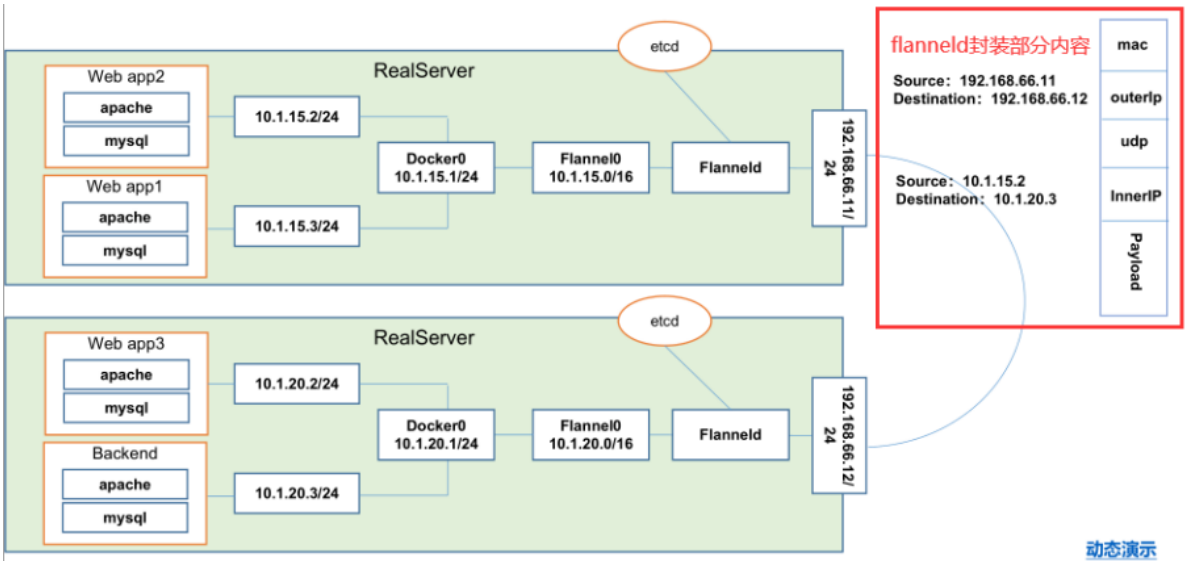
Pod 与 Service 之间的通讯：通过各节点的 Iptables（防火墙）规则的转换去实现。在K8S新版已经加入了LVS的转发机制并且转发上线会更高。

2.3.3 网络解决方案 Kubernetes + Flannel -1

Flannel 是 CoreOS 团队针对 Kubernetes 设计的一个网络规划服务，简单来说，它的功能是让集群中的**不同节点主机创建的 Docker 容器都具有全集群唯一的虚拟 IP 地址。而且它还能在这些 IP 地址之间建立一个覆盖网络（ Overlay Network ），通过这个覆盖网络，将数据包原封不动地传递到目标容器内**

上面加粗的这句话提取出来的概念就是，不同机器这里指的是物理机，上面运行的docker。这里面的容器他们之间的IP不能一致。

2.3.4 Flanneld的网络的通讯方案原理（同主机和不同主机通讯）



这里画了两台大主机、一个是192.168.66.11/24一个是192.168.66.12/24然后在这两台主机中运行了4个pod，然后在192.168.66.11这个主机中运行的是web app2和web app1的两个pod。然后在192.168.66.12这个主机中运行的是web app3和backend（前端组件）。然后所有的浏览访问到backend上，backend经过自己的网关去处理，把什么样的请求分配在什么样的服务上。

那也就意味着当backend想要去跟web app2或者是web app2想要跟backend通讯的时候就需要涉及到跨主机了，以及web app3跟本机的backend两个不同pod之间通讯的话他们是同主机的，是怎么来解决通讯的问题的。那我们下面来详细解析一下。

首先在我们的node服务器上我们会去安装一个叫flanneld的守护进程，这个flanneld的进程呢会监听一个端口，这个端口就是用于后期转发数据包以及接收数据包的这么一个服务端口。这个flanneld进程一旦启动之后会开启一个网桥叫 flannel0 这个网桥 flannel0 就是专门去收集 docker0 转发出来的数据报文。然后这个 docker0 会分配自己的 IP 到对应的 POD 上，如果是我们同一台主机上的两个 pod 之间的访问的话走的其实是 docker0 的网桥，因为大家都是在同一个网桥下面的两个不同的子网而已。也就是在真实服务器的主机内核已经完成了这次数据报文的转换。

怎么跨主机还能通过对方的 IP 直接到达：

到达流程：首先我们的 web app2 pod 想要把数据包发送到 backend ，这时候 web app2 的数据包源地址写自己的 10.1.15.2/24 目标要写 backend 的 10.1.20.3/24 。由于不是在同一台主机所以 web app2 先发送到他的网关也就是 docker0 的 IP 10.1.15.1/24，而 docker0 上会有对应的函数把这个数据包抓到我们的 flannel0 并且在flannel0 上会有一堆的路由表记录。这个路由表记录是从我们的 etcd 里面获取到的，写入到我们当前的主机判断路由到底到那台机器。到了 flannel0 之后因为 flannel0 是 flanneld 开启这么一个网桥所以这个数据包会到flanneld，到了 flanneld 以后会对这个数据报文进行封装也就是上图的封装部分。有这么一个结构mac是mac部分，然后到我们的下一层也就是第三层，第三层内容写的是源地址为192.168.66.11目标写的是192.168.66.12 。接着下一层封装的是UDP的数据报文，也就意味着 flanneld 使用的是UDP的数据报文去转发这些数据包因为更快，毕竟在同一个局域网内部。在下一层有封装了一层新的三层信息源是10.1.15.2目标是10.1.20.3封装到这一层以后外面就封装了一个 payload 实体。在被转发到192.168.66.12的这个主机上。原因是我们的三层写了目标地址是192.168.66.12所以这个数据报文是能够到这里的，并且也是对应的这里的flanneld的端口所以这个数据包会被192.168.66.12主机的 flanneld 所截获。Flanneld截获以后会进行拆封，因为192.168.66.12主机的 flanneld 他知道这是干嘛的，拆封完了之后会转发到192.168.66.12主机的 flannel0（10.1.20.0）然后再转发到docker0（10.1.20.1）。然后在转发给 192.168.66.12主机上的 backend pod。这样的话就能是实现我们的跨主机的扁平化网络。

以上就是整个flanneld的网络的通讯方案

2.3.5 Flannel与ETCD之间有哪些关联

ETCD 之 Flannel 提供说明：

1、存储管理 Flannel 可分配的 IP 地址段资源，也就意味着 flannel 在启动之后会向 etcd 去插入可以被分配的网段，并且把那个网段分配到那台机器 etcd 都会记录着，防止以增的网段在被 flannel 利用分配给其他的 node 节点。以避免IP冲突

2、监控 ETCD 中每个 Pod 的实际地址，并在内存中建立维护 Pod 节点路由表。

也就是说 ETCD 在我们的K8S集群中，是非常重要的，**所以想要进行高可用的话 ETCD 一定是我们最先高可用的一个组件。**

2.3.6 不同情况下网络通信方式

同一个 Pod 内部通讯：同一个 Pod 共享同一个网络命名空间，共享同一个 Linux协议栈所以都是使用的 lo 的回环网卡进行通讯的

Pod1 至 Pod2

```
> Pod1 与 Pod2 不在同一台主机， Pod 的地址是与 docker0 在同一个网段的，但 docker0 网段与宿主机网卡是两个完全不同的 IP 网段，并且不同 Node 之间的通信只能通过宿主机的物理网卡进行。将 Pod 的 IP 和所在 Node 的 IP 关联起来，通过这个关联让 Pod 可以互相访问
```

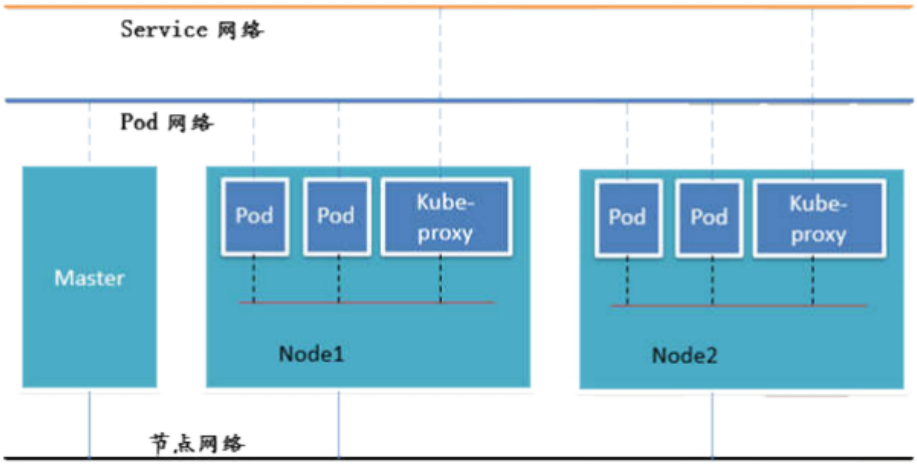
```
> Pod1 与 Pod2 在同一台机器，由 Docker0 网桥直接转发请求至 Pod2 ，不需要经过 Flannel Pod 至 Service 的网络：目前基于性能考虑，全部为 iptables或LVS 维护和转发
```

Pod 到外网： Pod 向外网发送请求，查找路由表，转发数据包到宿主机的网卡，宿主网卡完成路由选择后， iptables 执行 Masquerade 动作 ，把源 IP 更改为宿主网卡的 IP ，然后向外网服务器发送请求，也就意味着如果 pod 里面的容器想上网也就是通过 masquerade 的动态转换去完成所谓的上网行为

外网访问 Pod ： Service ，外网想要访问pod就要借助到我们service的node pod方式才能进行所谓的访问的K8S集群内部，因为她虽然是一个扁平化的网络，但可别忘了这个扁平化的网络是一个私有网络并不能在跟物理机相邻的网络内访问到他所以要借助到我们的node pod进行映射。

以上就是我们的不同网络的通讯机制。

2.3.7 组件通讯示意图



在我们得K8S里有三层网络，第一叫做节点网络；第二叫做 pod 网络；第三叫 service 网络。需要注意的是真实得物理网络只有一个就是节点网络。也就意味着我们在构建服务器得时候只需要一张网卡就可以实现。Pod 和service 网络都是虚拟网络，这里我们都可以把它理解为是一个内部网络。所有的 pod 都会在这个扁平化的网络中进行通讯。那如果像访问 service 得话需要在 service 这么一个网络中进行访问，service 再去跟后端的 pod 进行所谓通讯得通过 iptables 得或者是通过LVS得转换去达到通讯。