5.2、Kubernetes

基础概念

讲师：汪洋

目录

1 Pod 概念

- 自主式 Pod
- 控制器管理的 Pod

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

ReplicationController 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 来取代 ReplicationControlle

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

虽然 ReplicaSet 可以独立使用，但一般还是建议使用 Deployment 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 但 Deployment 支持）

返 回

Deployment 为 Pod 和 ReplicaSet 提供了一个声明式定义 (declarative) 方法，用来替代以前的 ReplicationController 来方便的管理应用。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

返 回

Horizontal Pod Autoscaling 仅适用于 Deployment 和 ReplicaSet ，在 V1 版本中仅支持根据 Pod 的 CPU 利用率扩所容，在 v1alpha 版本中，支持根据内存和用户自定义的 metric 扩缩容

返 回

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计），其应用场景包括：

* 稳定的持久化存储，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
* 稳定的网络标志，即 Pod 重新调度后其 PodName 和 HostName 不变，基于 Headless Service（即没有 Cluster IP 的 Service ）来实现
* 有序部署，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依次进行（即从 0 到 N-1，在下一个 Pod 运行之前所有之前的 Pod 必须都是 Running 和 Ready 状态），基于 init containers 来实现
* 有序收缩，有序删除（即从 N-1 到 0）

返 回

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

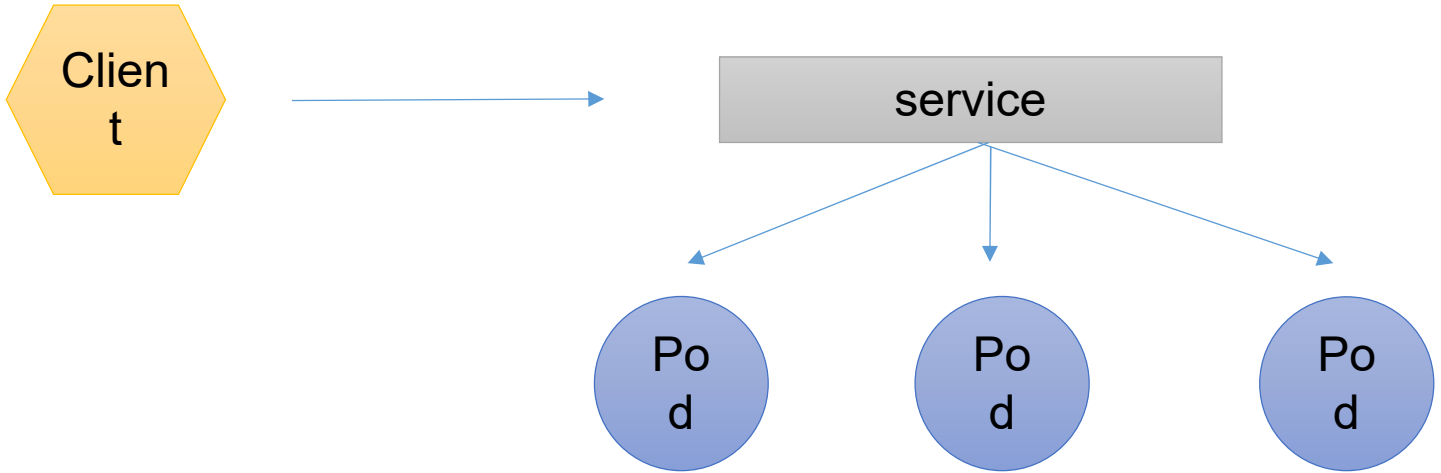
返 回

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

返 回



2 网络通讯方式

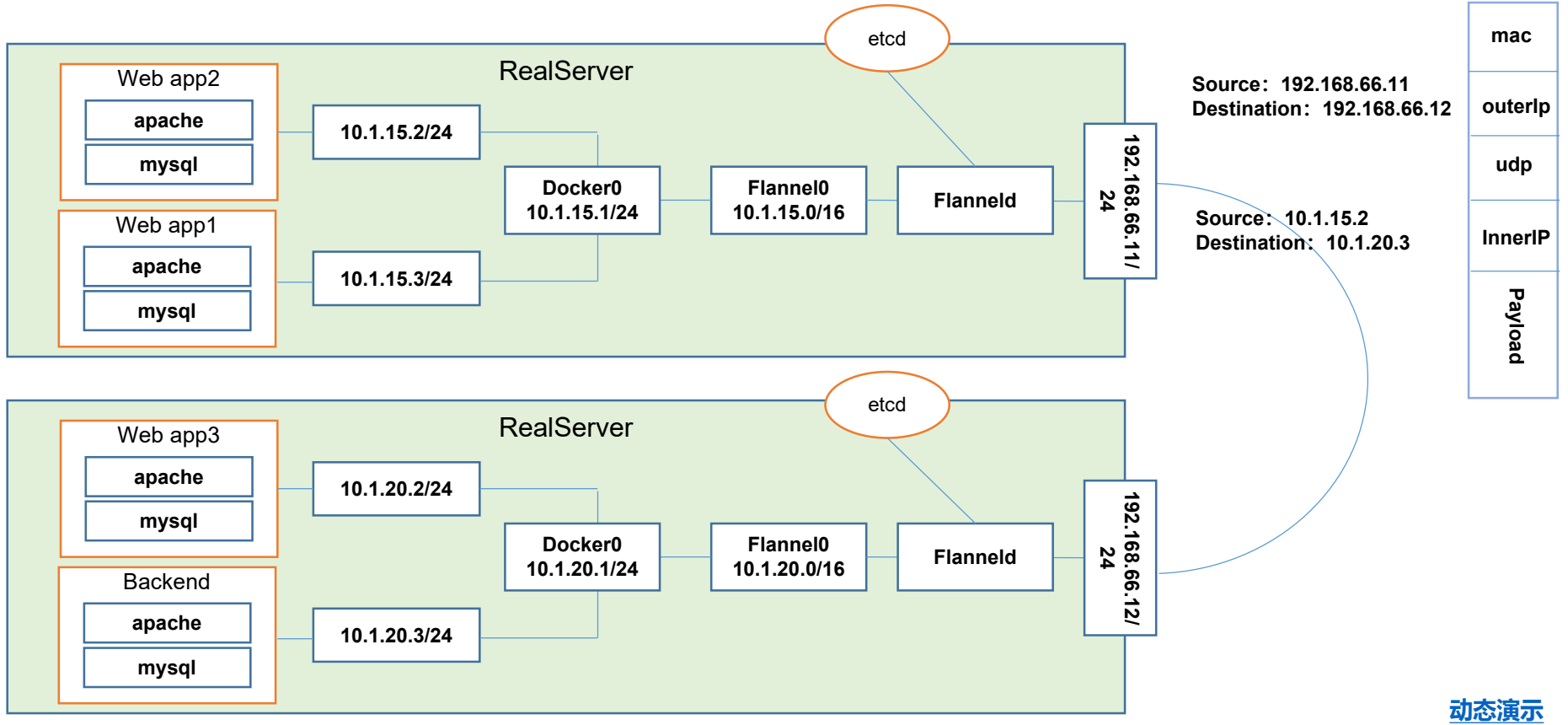
Kubernetes 的网络模型假定了所有 Pod 都在一个可以直接连通的扁平的网络空间中，这在 GCE（Google Compute Engine）里面是现成的网络模型，Kubernetes 假定这个网络已经存在。而在私有云里搭建 Kubernetes 集群，就不能假定这个网络已经存在了。我们需要自己实现这个网络假设，将不同节点上的 Docker 容器之间的互相访问先打通，然后运行 Kubernetes

同一个 Pod 内的多个容器之间：lo

各 Pod 之间的通讯：Overlay Network

Pod 与 Service 之间的通讯：各节点的 Iptables 规则

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



动态演示

ETCD 之 Flannel 提供说明：

- 存储管理 Flannel 可分配的 IP 地址段资源
- 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

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

Pod1 至 Pod2

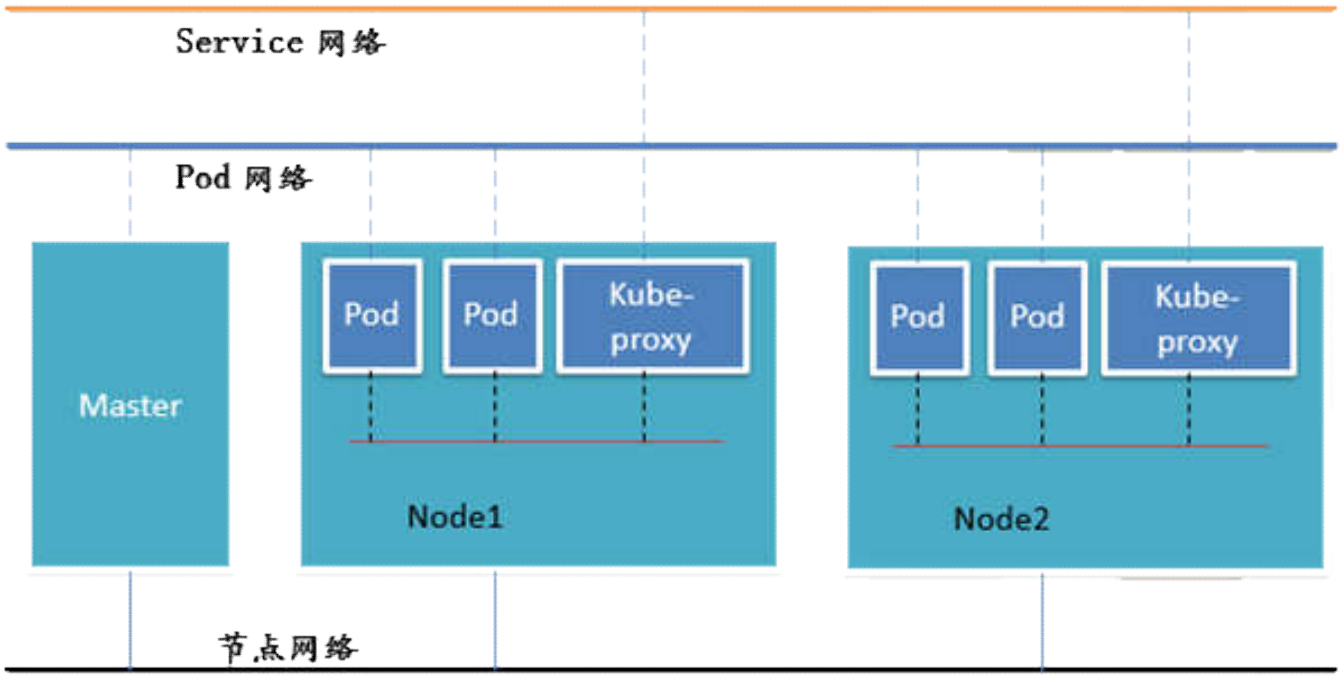
> Pod1 与 Pod2 不在同一台主机，Pod的地址是与docker0在同一个网段的，但docker0网段与宿主机网卡是两个完全不同的IP网段，并且不同Node之间的通信只能通过宿主机的物理网卡进行。将Pod的IP和所在Node的IP关联起来，通过这个关联让Pod可以互相访问

> 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 演示

Pod 至 Service 的网络：目前基于性能考虑，全部为 iptables 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，把源 IP 更改为宿主网卡的 IP，然后向外网服务器发送请求

外网访问 Pod：Service

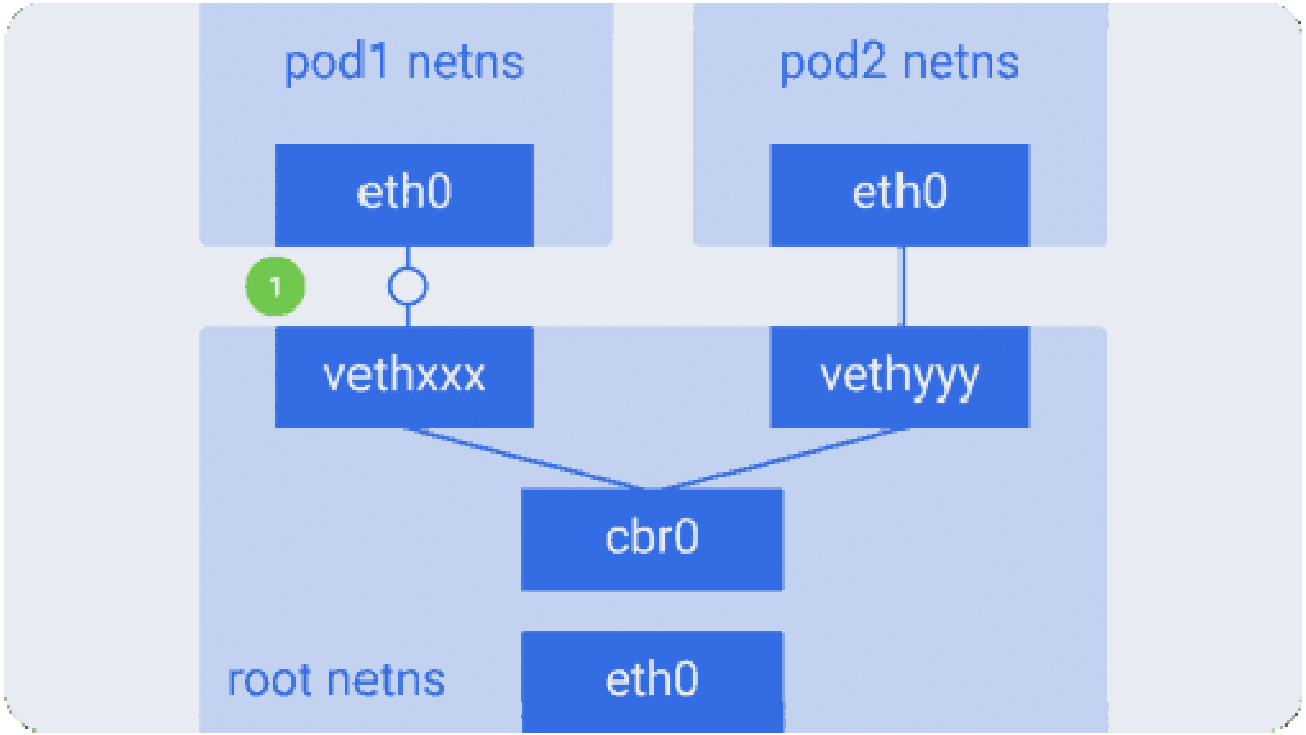

Service 网络
Pod 网络
Master
Pod
Pod
Kube-proxy
Node1
Pod
Pod
Kube-proxy
Node2
节点网络

附页

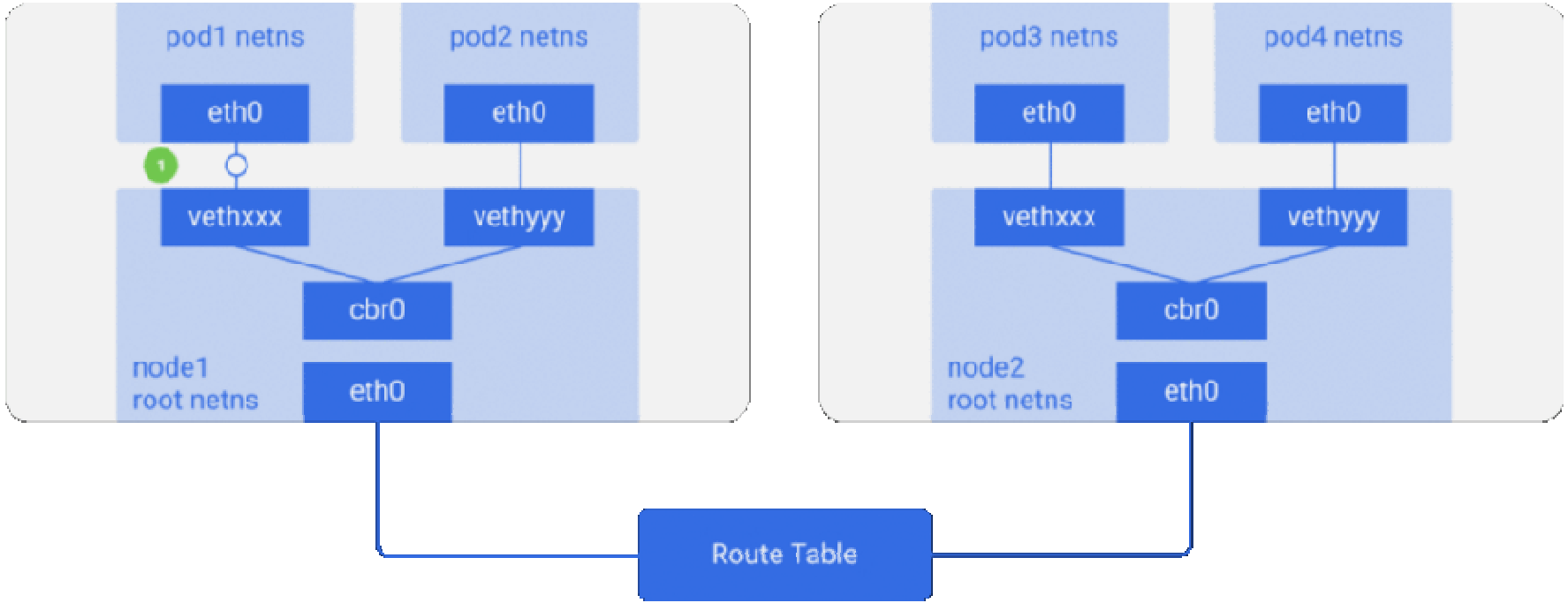
命令式编程：它侧重于如何实现程序，就像我们刚接触编程的时候那样，我们需要把程序的实现过程按照逻辑结果一步步写下来

声明式编程：它侧重于定义想要什么，然后告诉计算机 / 引擎，让他帮你去实现

返 回



返 回



返 回