Network Management on k8s

Introduction

Kubernetes管理的是集群，要解决的核心问题包括：

- 容器间的通信 (container network)
- Pod间的通信 (container network)
- Pod与Service间的通信 (k8s network)
- Internet与Service间的通信 (k8s network)

Kubernetes集群内部存在三类IP：

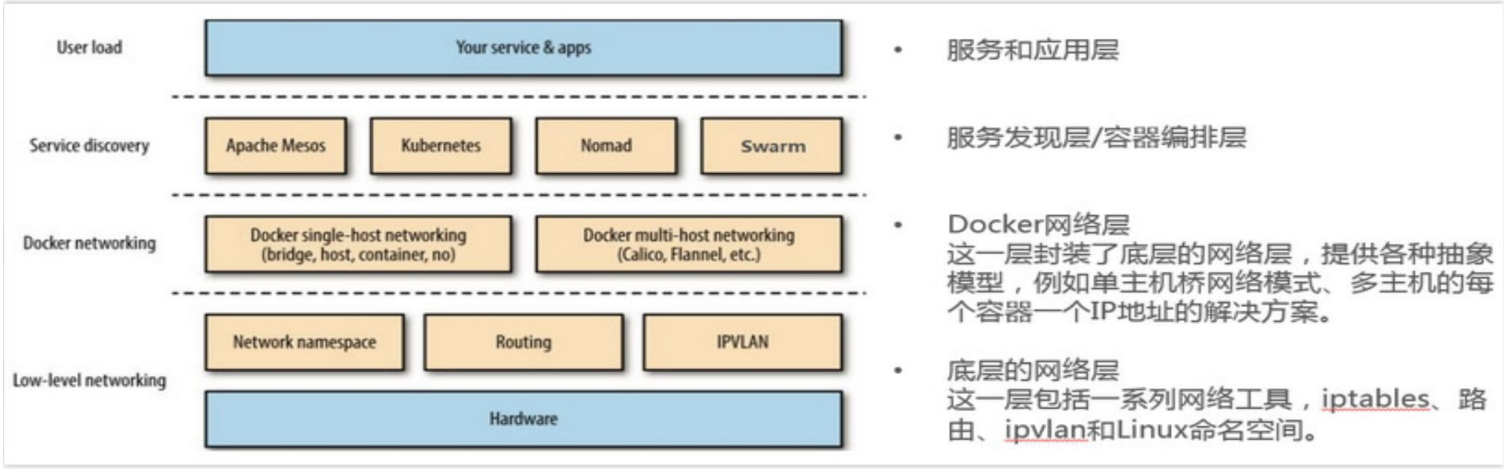
- Node IP：宿主机的IP地址
- Pod IP：使用网络插件创建的IP（如flannel），使跨主机的Pod可以互通
- Cluster IP：虚拟IP，通过iptables规则访问服务

构建K8s网络的基本原则是：

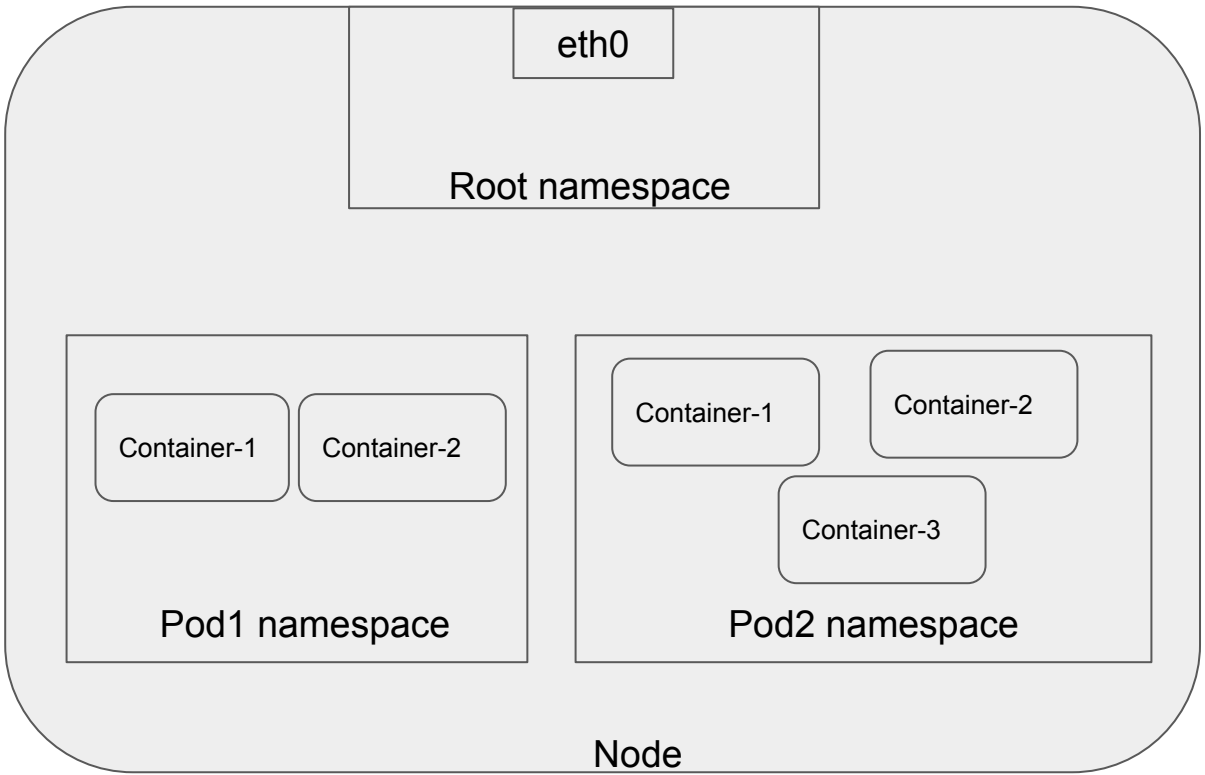
- Pod的IP是最小粒度的IP，采用IP-per-Pod模型
- Pod内的容器共享一个网络堆栈，可以通过localhost来访问对方端口，类似于一个VM内不同的进程
- 每个Pod拥有集群内唯一的IP地址，且保证不同节点的Pod IP不会重复
- 保证节点的Pod可以互相通信

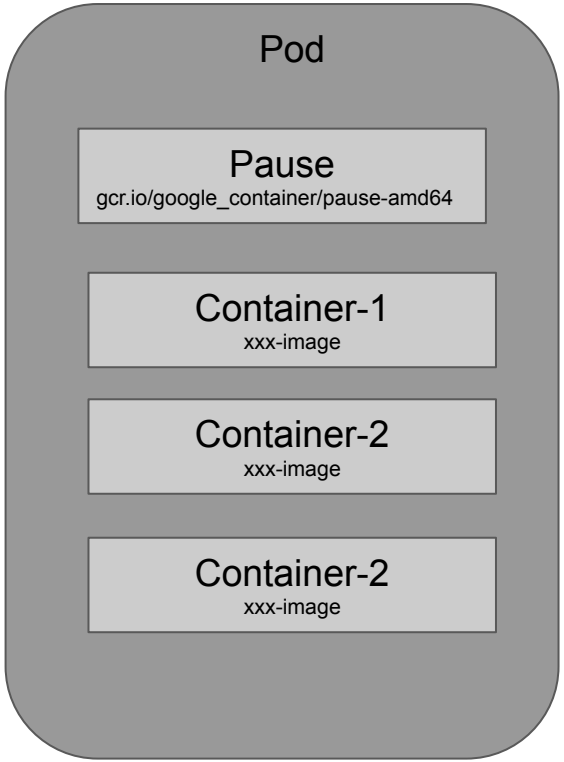
Linux Network Basic

- **Network NameSpace(netns):** Linux在网络栈中引入网络命名空间，将独立的网络协议栈隔离到不同的命令空间中，彼此间无法通信；docker利用这一特性，实现不容器间的网络隔离。
- **Veth :** Veth设备对的引入是为了实现在不同网络命名空间的通信。
- **Iptables/Netfilter:** Netfilter负责在内核中执行各种挂接的规则(过滤、修改、丢弃等)，运行在内核 模式中；Iptables模式是在用户模式下运行的进程，负责协助维护内核中Netfilter的各种规则表；通过二者的配合来实现整个Linux网络协议栈中灵活的数据包处理机制。
- **Bridge:** 网桥是一个二层网络设备,通过网桥可以将linux支持的不同的端口连接起来,并实现类似交换机那样的多对多的通信。
- **Router:** Linux系统包含一个完整的路由功能，当IP层在处理数据发送或转发的时候，会使用路由表来决定发往哪里。

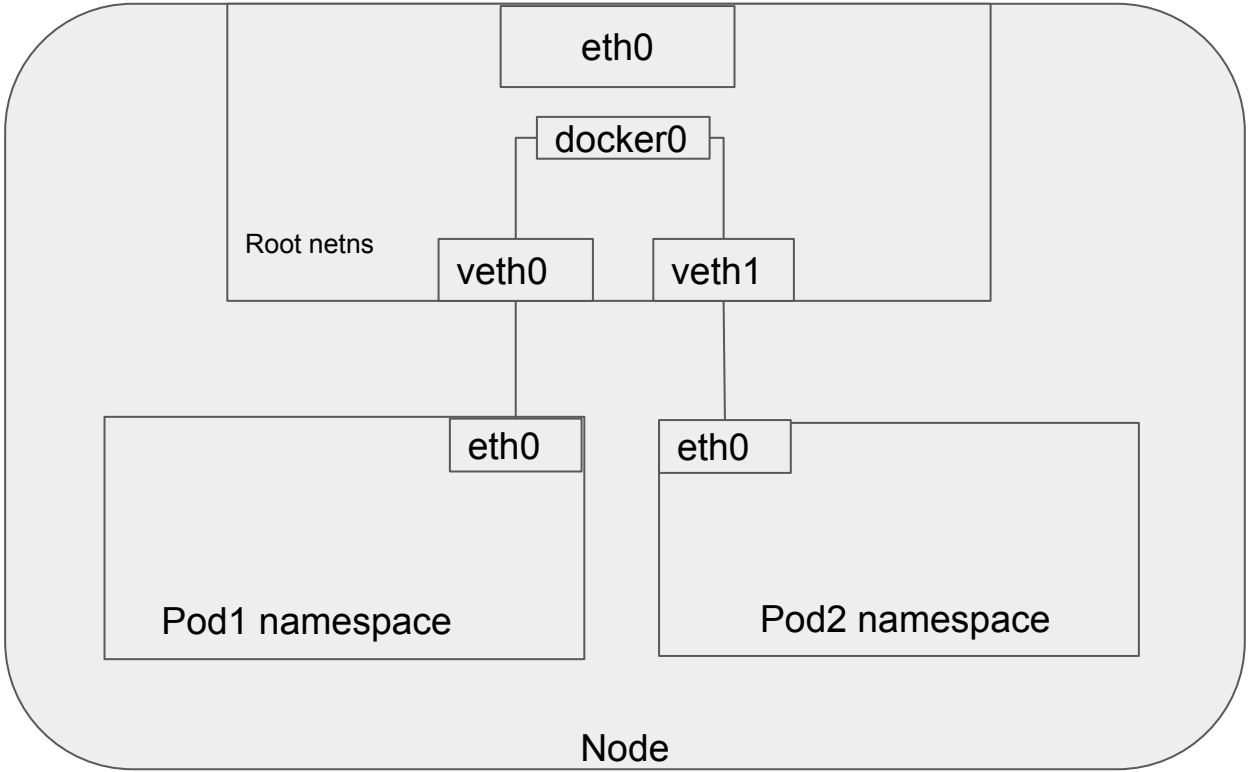


Container - Container

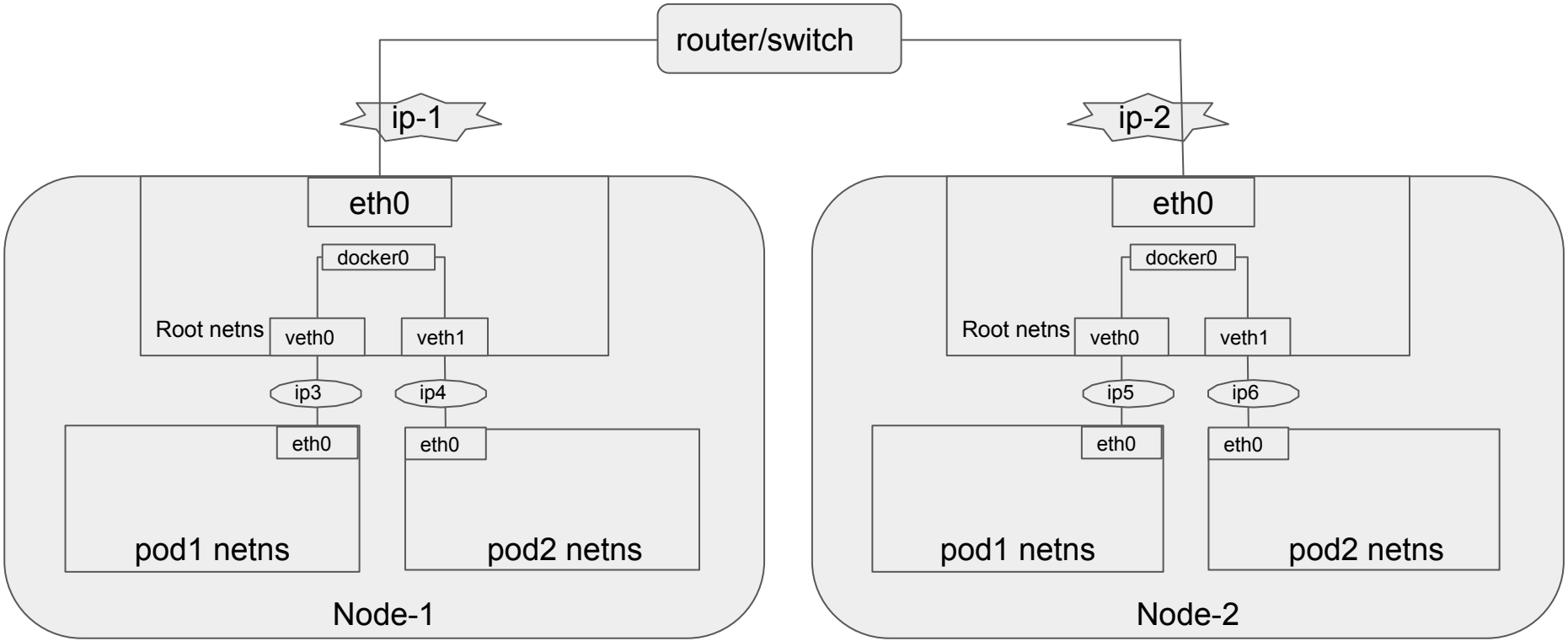




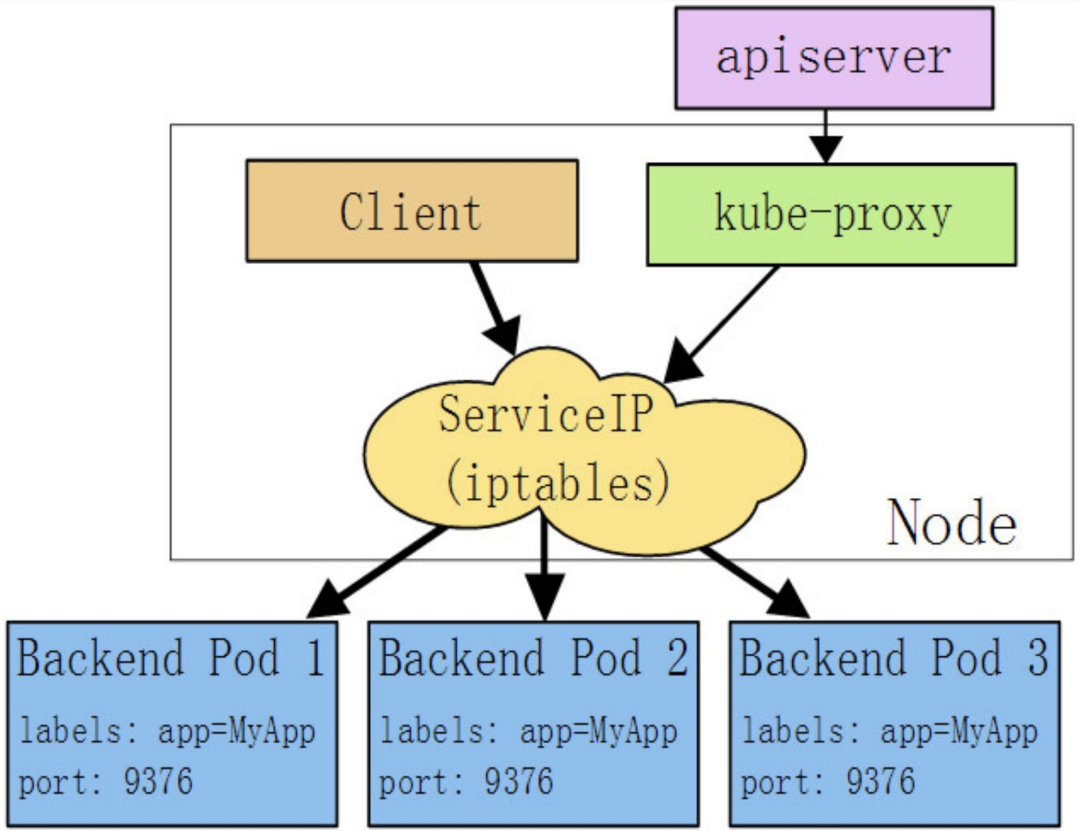
Pod-Pod on Same Node



Pod-Pod across Node



Pod-Service



- Service只是一个虚拟的概念，真正完成请求转发的是运行在node节点上的kube-proxy。
- Kube-proxy是一个简单的网络代理和负载均衡器，通过Round Robin负载均衡算法和session粘连规则将请求转发给后端的Pod。

- Cluster-IP: kube-proxy虚拟出来的IP
- NodePort: 默认范围(30000-32767)，可以用nodeIP:nodePort直接访问服务

Different Solutions

协议栈层级

- 两层：APR+Mac Learning
- 三层：路由转发
- 两层+三层：节点内部二层转发，跨节点三层转发

穿越形态

- Overlay
  - 用封包解包的方式构造一个隧道，通过隧道穿越底层基础设施
  - flannel(udp/vxlan)，weave，calico(ipip)，openvswitch等
- Underlay
  - 直接通过路由的方式穿越底层基础设施
  - flannel(host-gw)，calico(bgp)，Macvlan（需特殊二层路由器），contiv等

隔离方式

- FLAT：纯扁平网络，无隔离
- VLAN：总组户数受限
- VXLAN/GRE: 基于IP穿越的方式

Most Popular Network Plugin for k8s

Kubernetes本身并不提供网络功能，只是把网络接口开放出来，通过插件的形式实现。
为了解决该问题，出现了一系列开源的Kubernetes中的网络插件与方案，如：

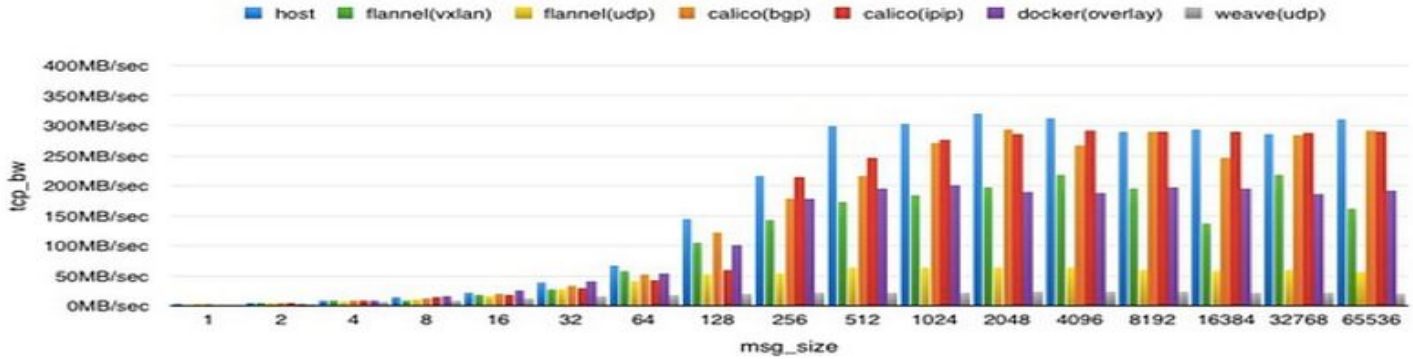
- **Flannel**: host-gw 模式，节点内部子网，节点之间通过路由指过去。
- **Calico**:基于 BGP 的三层 Underlay。使用BGP路由协议在主机之间路由数据包。支持网络策略的配置。
- **Contiv**:功能全,有二层桥接，基于 VLAN 的网络；有三层路由，基于 BGP 的网络；同时它可以支持 Overlay，通过 VXLAN，去应对一些受控的网络环境，提供 ACI 去支持网络策略
- **Openshift SDN**: 基于 VXLAN 的二层+三层 Overlay 方案，同时支持 network policy，数据面基于 OVS 流表实现。
- **Weave**:网状拓扑,在各节点间创建网状overlay网络
- **Canal:**试图讲Flannel和Calico结合在一起，初步 实现，目前进展缓慢
- **Cilium**:使用名为BPF(Berkeley Packet Filter)的新Linux内核技术，提供了一种基于容器 /容器标识定义和实施网络层和应用层安全策略的简单而有效的方法

只要实现k8s的CNI(Container Network Interface),你也可以自己写个插件. https://github.com/containernetworking/cni

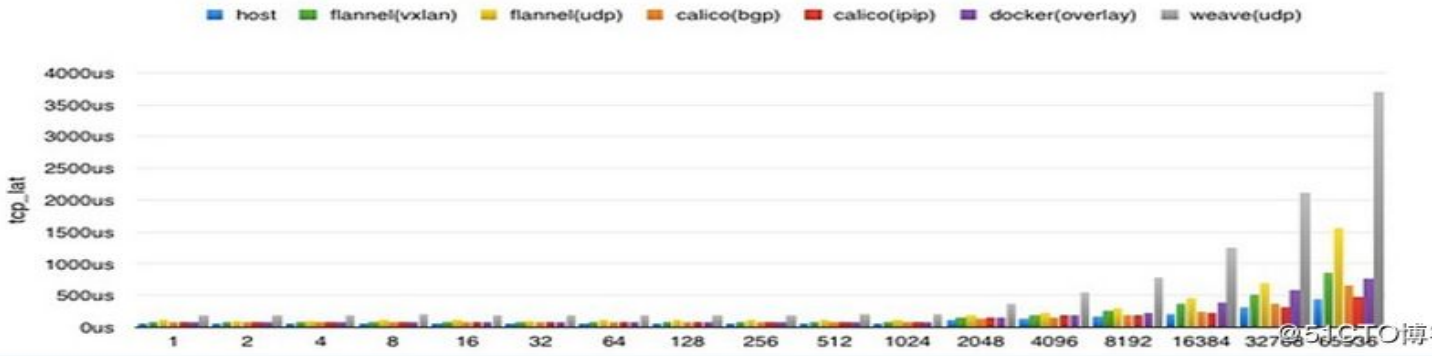
Performance Comparison

CPU压力： host < calico(bgp) < calico(ipip) = flannel(vxlan) = docker(vxlan) < flannel(udp) < weave(udp)

带宽：

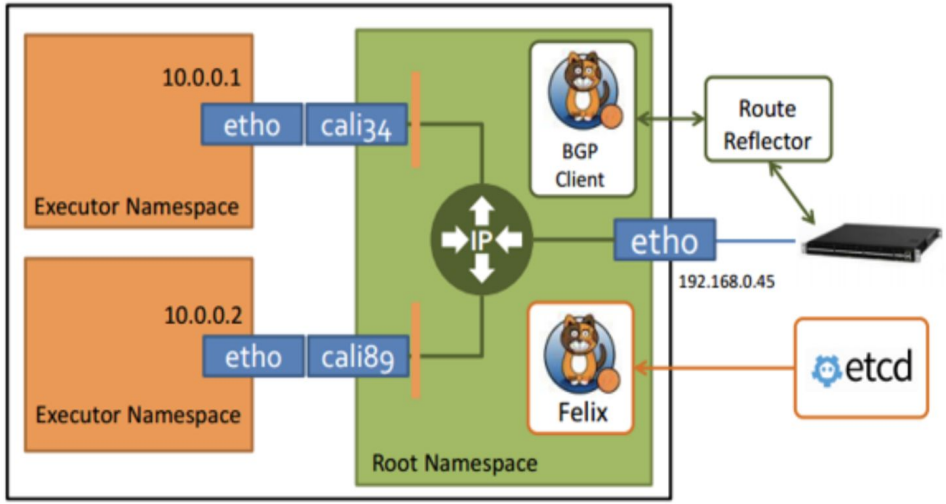


延迟：



Calico 架构

- felix: Calico Agent主要负责配置路由和ACLs等信息来保证终端连通
- etcd：确保网络元数据一致性，保证calico网络状态的准确性
- BGP Client(BIRD): 负责把felix发送到kernel的路由信息发送到当前Calico网络中，确保workload间通信的有效性
- BGP Route Reflector(BIRD):大规模网络时使用，摒弃所有节点的mesh模式，通过一个或多个BGP Route Reflector来完成集中式的路由分发
- calico/calico-ipam:主要作为kubernetes的CNI插件



Calico的优劣势

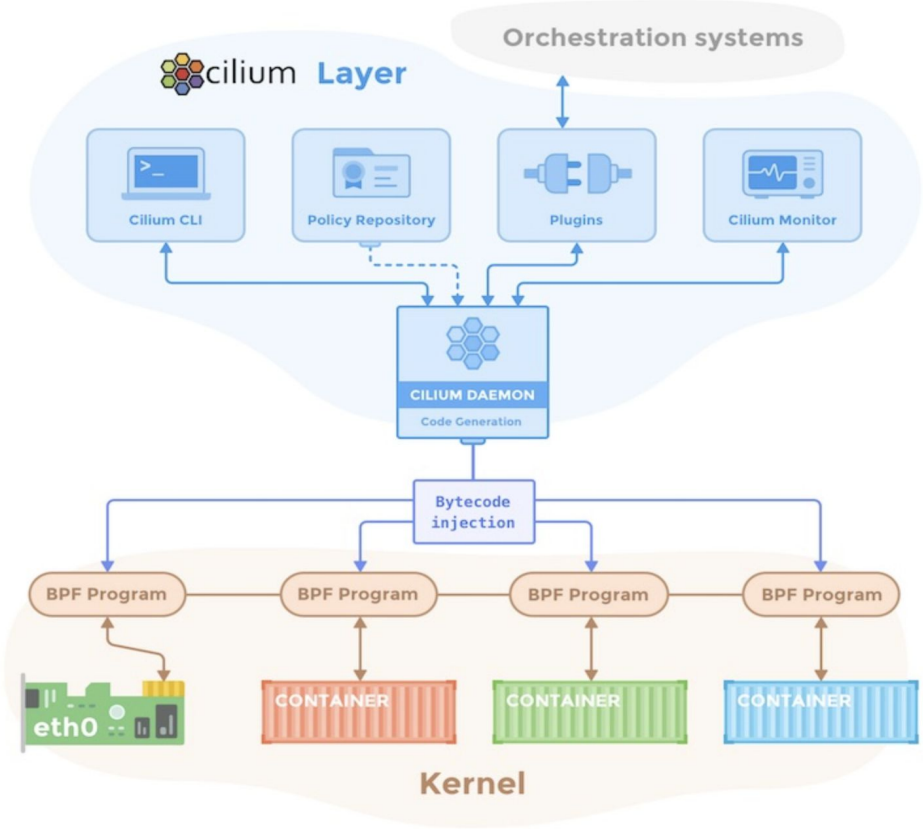
优势

- 纯三层的数据中心网络方案（不需要overlay)
- 利用Linux kernel实现了一个高效的vRouter来实现数据转发
- 小规模可直接互联，大规模用BGP协议通过路由互联
- 可直接利用数据中心的网络，无论L2还是L3，不需要额外的NAT,隧道或Overlay
- 基于IPtable提供了丰富的网络策略
- 与OpenStack、Kubernetes、AWS、GCE等IaaS和容器平台都有良好的集成

劣势

- 既然是三层实现，当然不支持VRF(virtual routing forwarding)
- 不支持多租户网络的隔离功能，在多租户场景下会有网络安全问题
- Calico控制平面的设计要求物理网络得是L2 Fabric，这样vRouter间都是直接可达的

Cilium


Orchestration systems
cilium Layer
Cilium CLI
Policy Repository
Plugins
Cilium Monitor
CILIUM DAEMON
Code Generation
Bytecode injection
BPF Program
BPF Program
BPF Program
BPF Program
eth0
CONTAINER
CONTAINER
CONTAINER
Kernel

Advantage & Disadvantage Comparison

方案	结论	优势	劣势
Calico	calico 的 2 个方案都有不错的表现，其中 ipip 的方案在 big msg size 上表现更好，但蹊跷是在 128 字节的时候表现异常，多次测试如此。bgp 方案比较稳定，CPU 消耗并没有 ipip 的大，当然带宽表现也稍微差点。不过整体上来说，无论是 bgp 还是 ipip tunnel，calico 这套 overlay sdn 的解决方案成熟度和可用度都相当不错，为云上第一选择。	性能好，可控性高，隔离性好	操作起来还是比较复杂，比如对 iptables 有依赖
Flannel	flannel 的 2 个方案表现也凑合，其中 vxlan 方案是因为没法开 udp offload 导致性能偏低，其他的测试报告来看，一旦让网卡自行解 udp 包拿到 mac 地址什么的，性能基本上可以达到无损，同时 cpu 占用率相当好。udp 方案受限于 user space 的解包，仅仅比 weave(udp) 要好一点点。	部署简单，性能还行	没法实现固定 IP 的容器漂移，没法多子网隔离，对上层设计依赖度高，没有 IPAM，IP地址浪费，对 docker 启动方法有绑定
docker 原生 Overlay	其实也是基于 vxlan 实现的。受限于 cloud 上不一定会开的网卡 udp offload，vxlan 方案的性能上限就是裸机的 55% 左右了。大体表现上与 flannel vxlan 方案几乎一致。	docker 原生，性能凑合	对内核要求高（>3.16），对 docker daemon 有依赖需求（consul / etcd），本身驱动实现还是略差点，可以看到对 cpu 利用率和带宽比同样基于 vxlan 的 flannel 要差一些，虽然有 api 但对 network 以及多子网隔离局部交叉这种需求还是比较麻烦，IPAM 很差