Kubernetes Network

基础

1.实验环境

1.1 系统环境

OS	ADDRESS	ROLE
CentOS7	10.10.100.133/kube01	MASTER
CentOS7	10.10.100.134/kube02	WORKER
CentOS7	10.10.100.135/kube03	WORKER

1.2 Flannel插件

```
# 部署之前清理原有集群
kubeadm reset
rm -f $HOME/.kube/config
ipvsadm --clear
rm -rf /etc/cni/net.d

[root@kube03 ~]# kubectl apply -f \
https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml

# 重新初始化集群
```

1.3 Kubernetes网络模型

- pod与pod间不通过NAT通信
- Worker节点与pods间通信不使用NAT
- Pod拥有独立的网络空间

实现容器网络的必要技术：

- namespace
- veth pair
- netfilter（PRE_ROUTING,FORWARD,LOCAL_IN,LOCAL_OUT,POST_ROUTING）
- 网桥
- 路由

由以上的约束引出四个不同的网络问题:

1. container to container

2. 相同节点pod to pod
3. 不同节点pod to pod
    - overlay(flannel UDP/VXLAN, Calico IPIP/VXLAN)
    - DR(flannel host-gw, calico BGP)
    - underlay(calico BGP)
4. Internet to Service

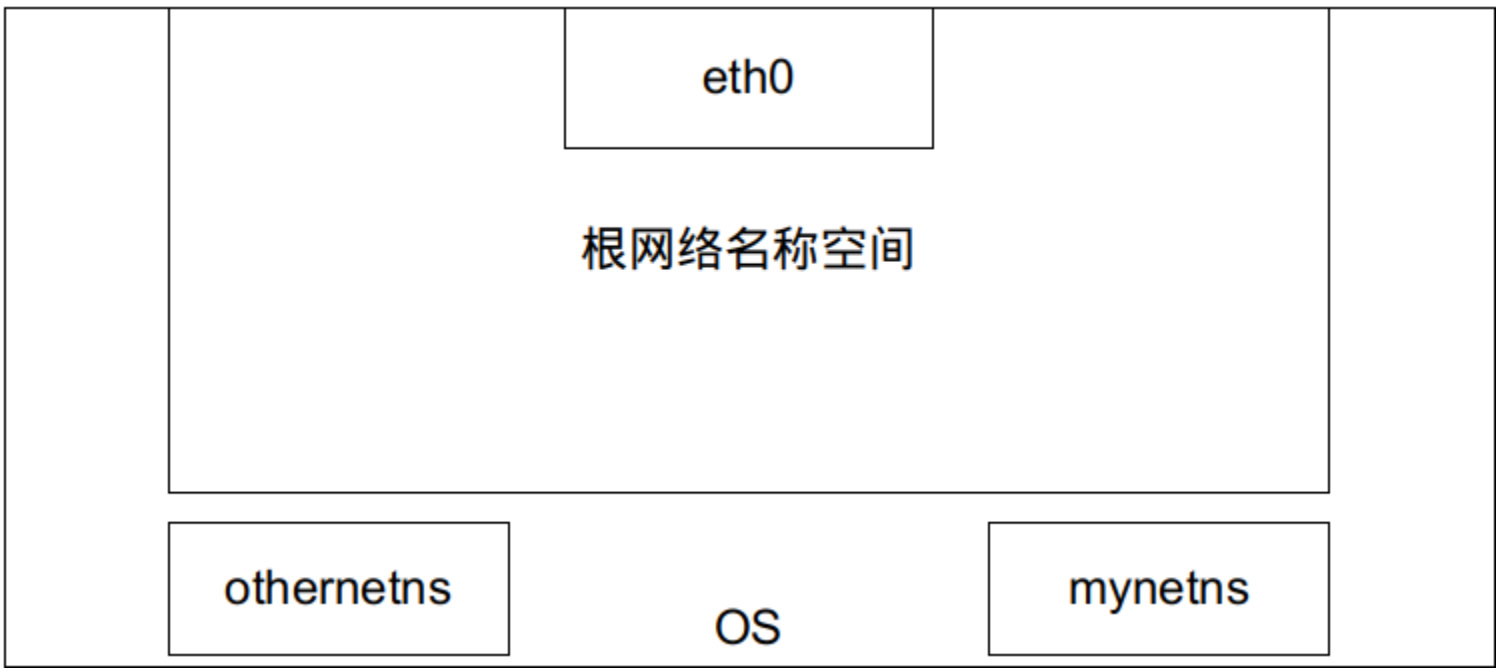
2. Container to Container

通常，我们将虚拟集中的网络通信视为与以太网的直接交互。但事实上，虚拟网络有着更为复杂的机制。在Linux中，使用network namespace机制隔离进程的网络协议栈，其中包含：

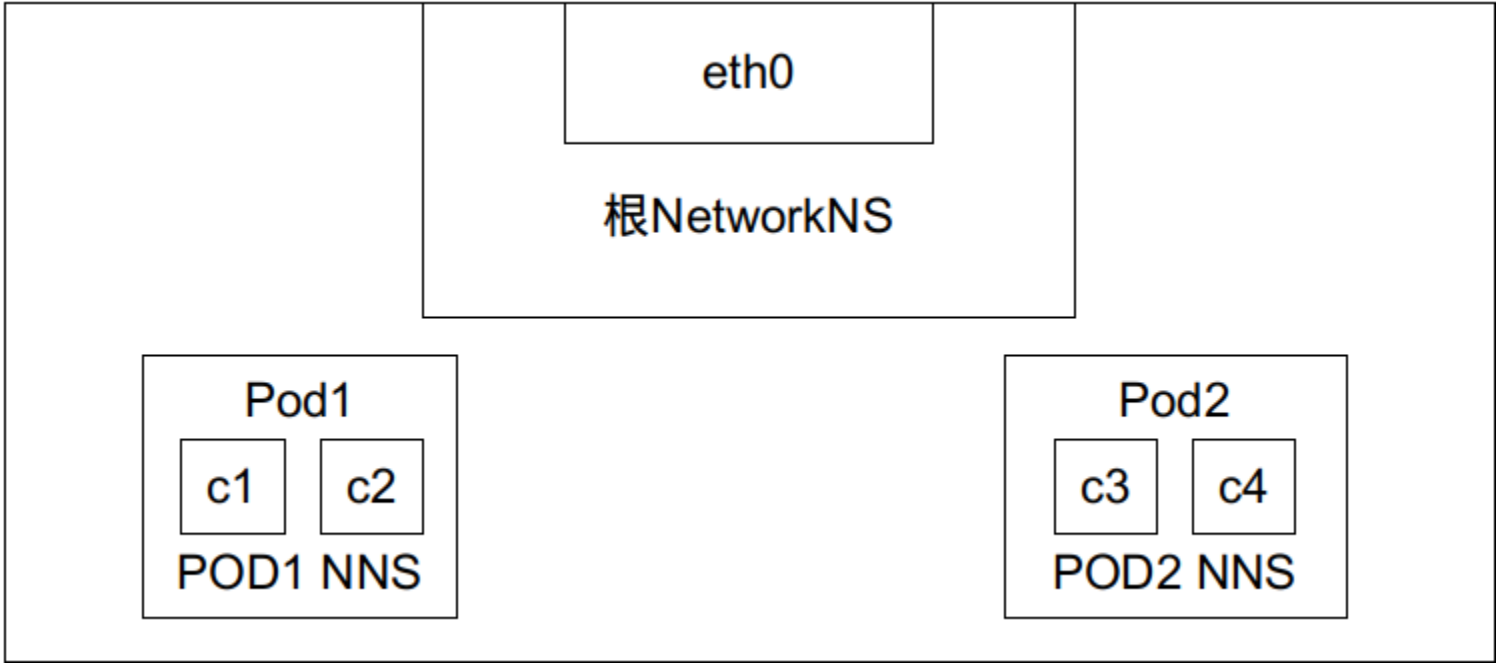
- 路由表（Route Table）
- 防火墙（Iptables)
- 网络设备（network device）
- 环回设备（loopback device）

Linux中创建network namespace的方法：

```
[root@kube01 ~]# ip netns add mynetns
[root@kube01 ~]# cat /var/run/netns/
default  mynetns
```



在kubernetes中，Pod被设计为cotainer的集合，这些container共享网络名称空间。Pod中的所有CContainer都具有相同的地址空间，可以通过localhost彼此通信。当创建POD时，kubernetes为Pod创建自己的NS。



实验

```
# 查看节点上的网络名称空间

[root@kube01 ~]# ip netns ls
default
[root@kube01 ~]# ll /var/run/docker/netns
总用量 0
-r--r--r-- 1 root root 0 7月   5 23:44 default

# 修改网络配置部署flannel网络

  net-conf.json: |
    {
      "Network": "192.168.150.0/24",
      "Backend": {
        "Type": "udp"
      }
    }


# 部署一个多container的容器

apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  restartPolicy: Never
  volumes:
  - name: shared-data
    emptyDir: {}
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html
  - name: debian
    image: debian
```

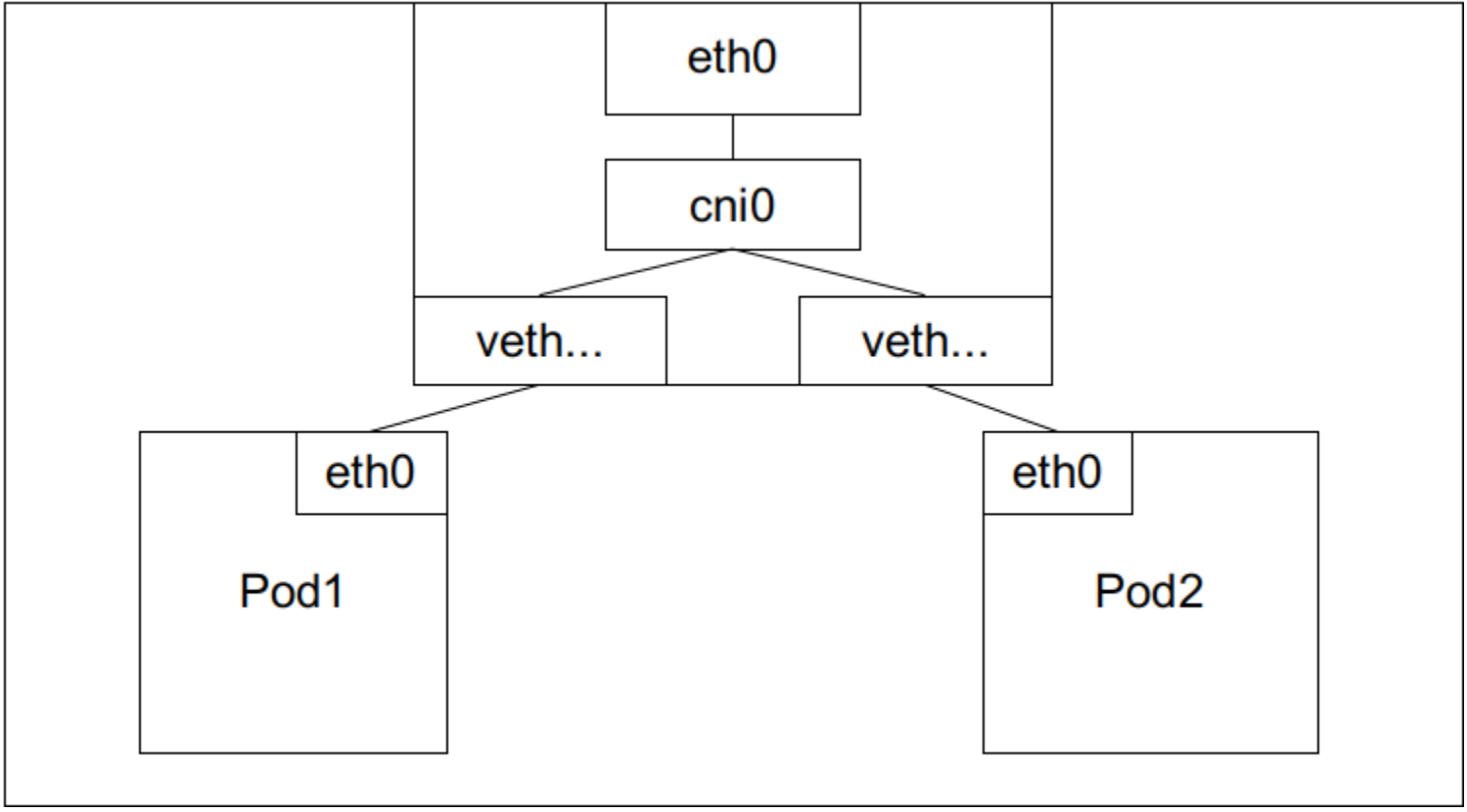
```
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html && sleep 3600"]

# 查看pod分配的节点

[root@kube01 ~]# kubectl get pod -o wide
NAME             READY   STATUS    RESTARTS   AGE    IP           NODE     NOMINATED NODE   READINESS
GATES
two-containers   2/2     Running   0          2m6s   10.111.2.2   kube03   <none>           <none>

# Pod所在节点查看NS
[root@kube03 ~]# ip netns ls
RTNETLINK answers: Invalid argument
RTNETLINK answers: Invalid argument
netns
1c95cab493ec (id: 0)
default

# 查看该网络空间网卡信息
[root@kube03 ~]# ip netns exec 1c95cab493ec ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1472 qdisc noqueue state UP group default
    link/ether a2:08:22:9d:b4:78 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.111.2.2/24 brd 10.111.2.255 scope global eth0
       valid_lft forever preferred_lft forever
```

3. POD TO POD

3.1 同节点

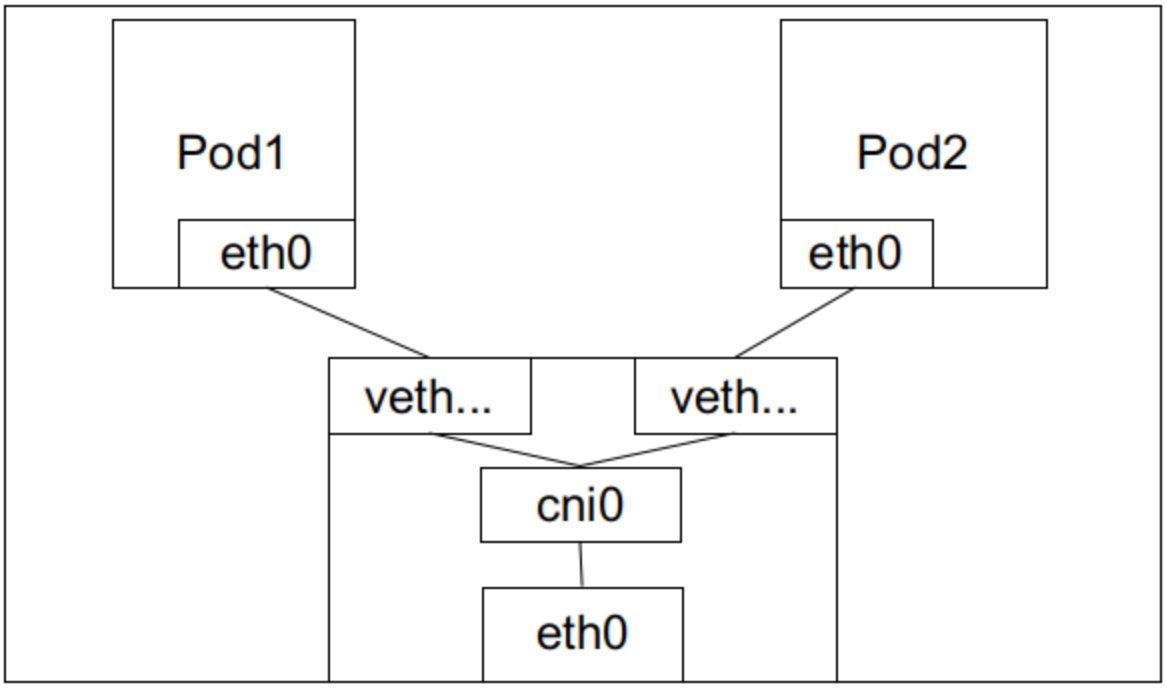
Linux以太网桥是虚拟的二层设备，用于联合两个或多个网络段，透明的连接两个网络。 网桥的工作方式是通过检查数据包的目的地之，决定是否转发给连接的其他网段，从而维护一个转发表。桥通过检查数据包中的MAC地址，决定删除数据还是转发给制定endpoint。

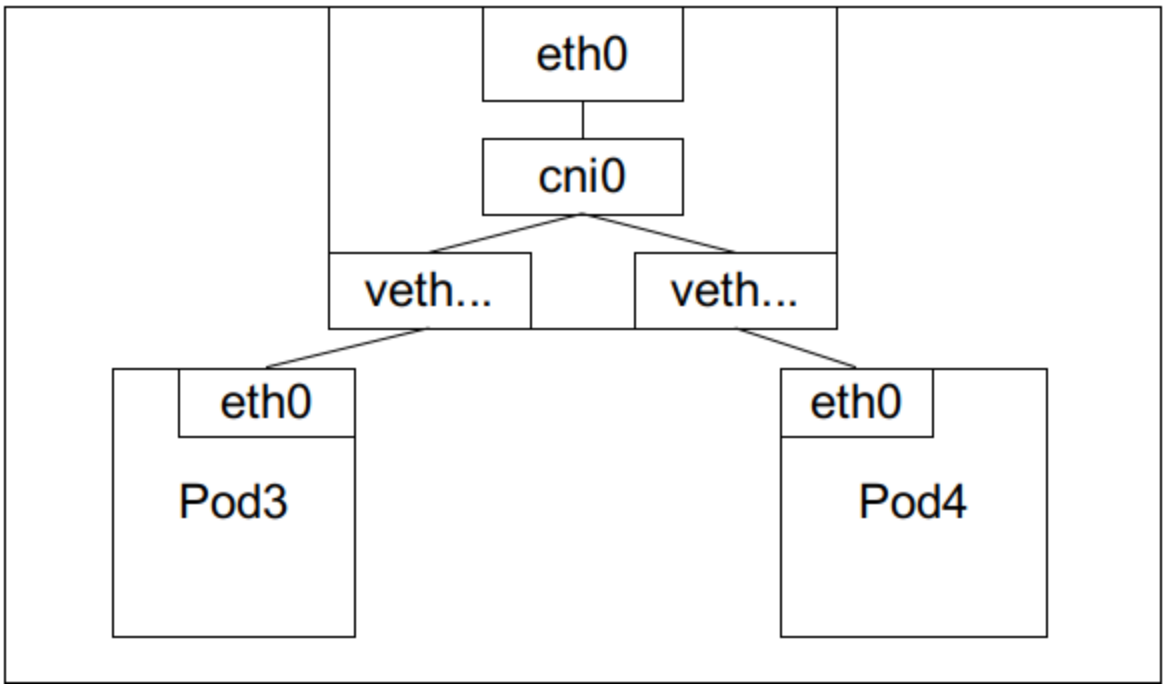
网桥通过ARP协议来发现与给定IP地址相关联的二层MAC地址。当一个数据帧在网桥上被接收时，网桥将该帧广播给所有连接的设备(原始发送方除外)，响应该帧的设备被存储在一个查找表中，具有相同IP地址的数据帧使用ARP表来发现该设备。



3.2不同节点

CNI通常为每个节点分配一个CIDR块，该块一般为POD_CIDR的子集。





如当前集群所示：

```
# kube01

7: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1472 qdisc noqueue state UP group default qlen 1000
    link/ether 6e:ad:d9:e9:aa:71 brd ff:ff:ff:ff:ff:ff
    inet 10.111.0.1/24 brd 10.111.0.255 scope global cni0
       valid_lft forever preferred_lft forever
    inet6 fe80::6cad:d9ff:fee9:aa71/64 scope link
       valid_lft forever preferred_lft forever


# kube03

7: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1472 qdisc noqueue state UP group default qlen 1000
    link/ether 22:cc:4c:ed:8f:eb brd ff:ff:ff:ff:ff:ff
    inet 10.111.2.1/24 brd 10.111.2.255 scope global cni0
```

```
       valid_lft forever preferred_lft forever
    inet6 fe80::20cc:4cff:feed:8feb/64 scope link
       valid_lft forever preferred_lft forever
```

3.3 实验

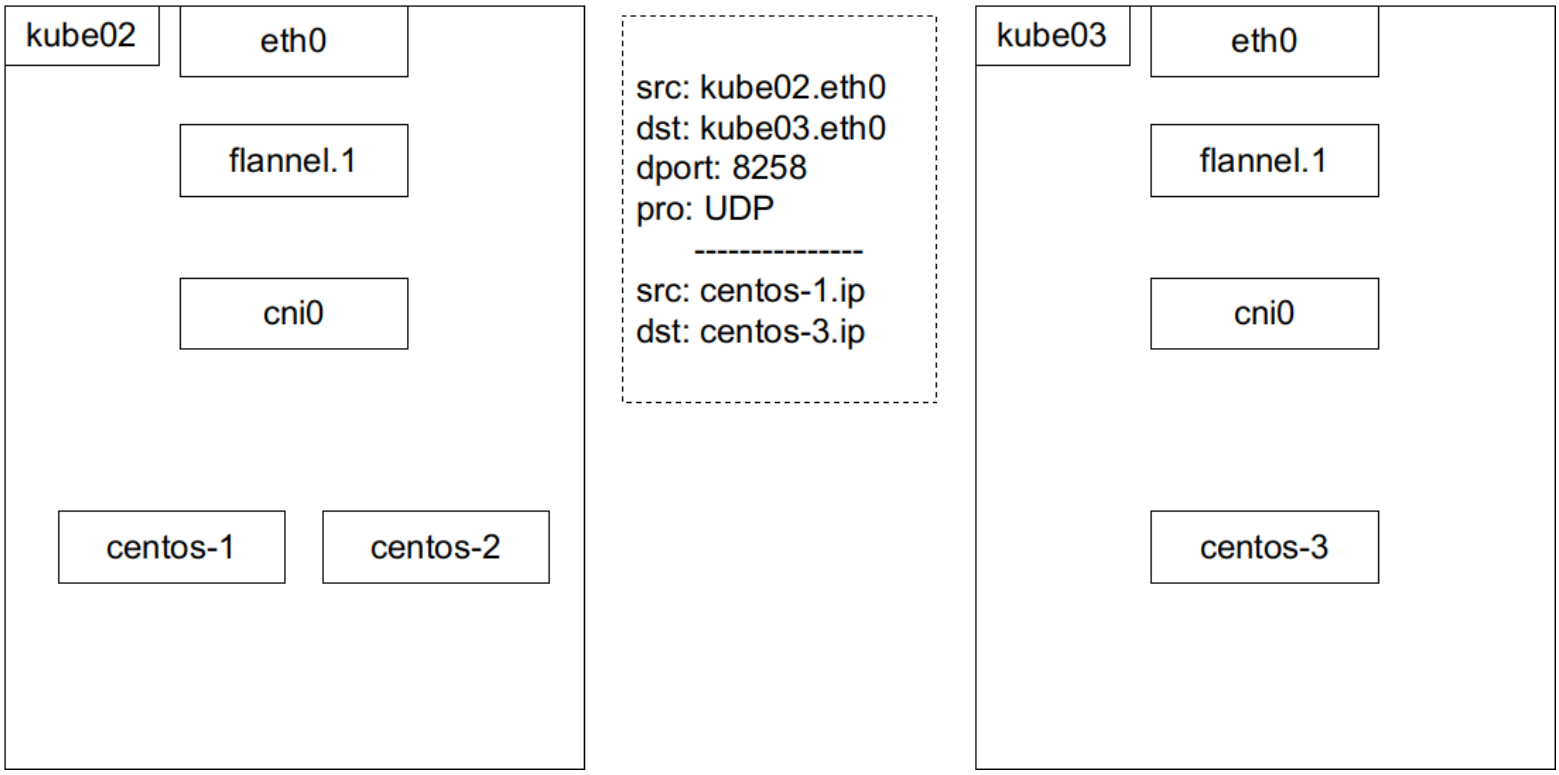
```
# 部署一个多副本Deployment
# 查看pod所在节点的ns情况
# 同节点通信使用tcpdump在cni0上抓包
# 不同节点上POD在flannel.1上抓包

tcpdum -nn -i [interface] dst [dst_addr]
```

4. Flannel UDP

Flannel 项目是 CoreOS 公司主推的容器网络方案。事实上，Flannel项目本身只是一个框架，真正为我们提供容器网络功能的，是 Flannel 的后端实现。目前，Flannel 支持三种后端实现，分别是：

- VXLAN，推荐使用，UDP封装，性能损耗
- host-gw，不进行UDP封装，无性能损耗，跨子网容易出现问题
- udp，不推荐，仅Debug使用

UDP 模式，是 Flannel 项目最早支持的一种方式，却也是性能最差的一种方式。所以，这个模式目前已经被弃用。不过，Flannel 之所以最先选择UDP模式，就是因为这种模式是最直接、也是最容易理解的容器跨主网络实现。



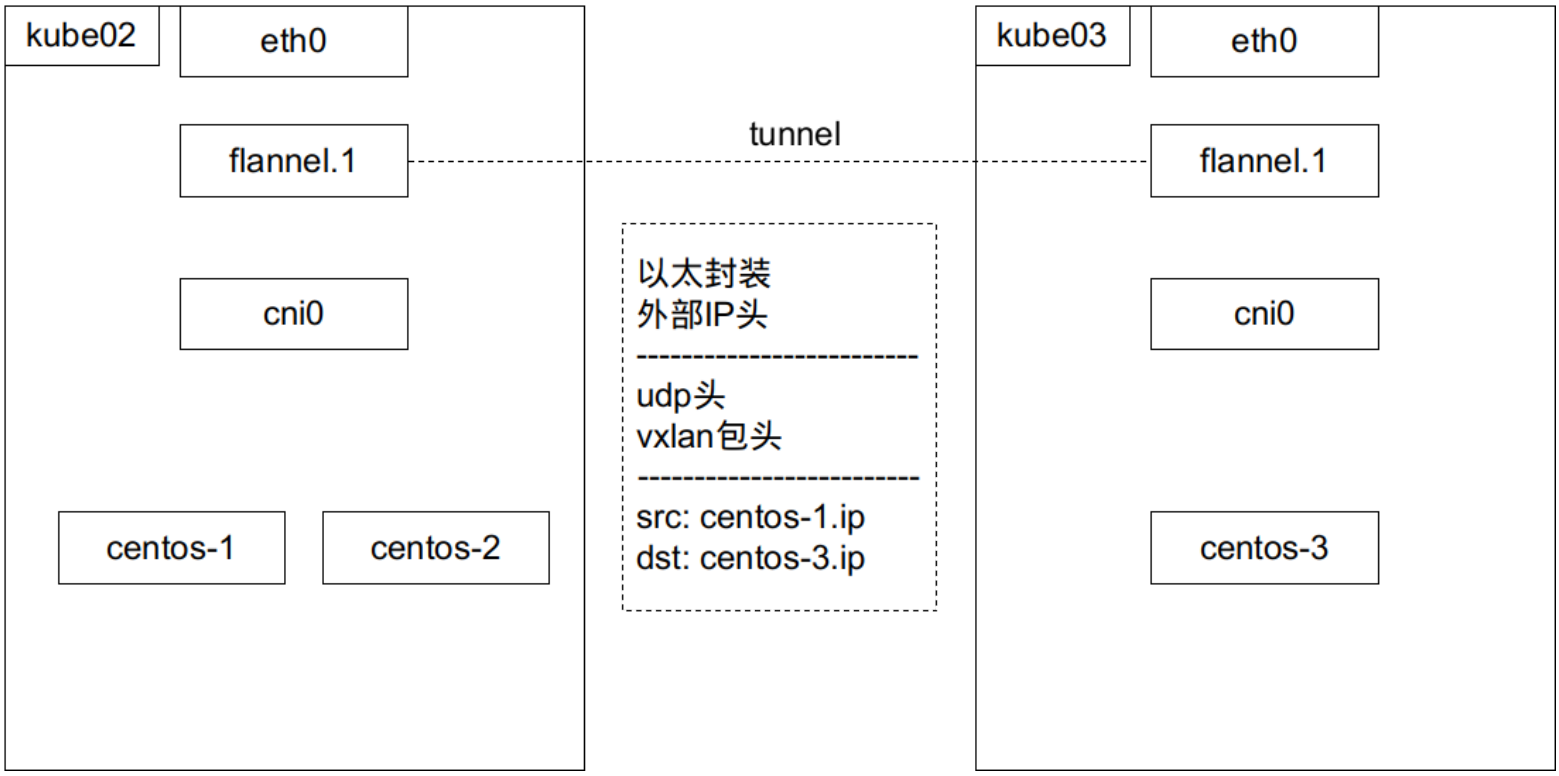
5. Flannel VXLAN

VXLAN的覆盖网络设计思想： 在现有三层网络之上，覆盖一个虚拟的由内核VXLAN模块负责维护的二层网络，使得连接到这个VXLAN二层网络上的instances之间，可以像在同一个局域网里一样自由通信。当然，实际上，这些instances可能分布在不

同的host上，甚至可以分布在不同的IDC中

为了能够在二层网络上打通“隧道”，VXLAN会在host上设置一个特殊的网络设备作为“隧道”的两端，这个设备是VTEP，即VXLAN Tunnel End Point

VTEP设备与flanneld的作用相似，只不过，它封装和解封的对象是二层数据帧，而这个流程是在内核空间完成的。



访问流程：

1. centos-1 访问 centos-3，先经过cni0，然后路由到flannel.1
2. VTEP将找到正确的设备，封装并打包转发

此时节点上的路由应该是

```
[root@kube01 ~]# ip route
default via 10.10.100.2 dev ens33 proto static metric 100
10.10.100.0/24 dev ens33 proto kernel scope link src 10.10.100.133 metric 100
10.111.0.0/24 dev cni0 proto kernel scope link src 10.111.0.1
10.111.1.0/24 via 10.111.1.0 dev flannel.1 onlink
10.111.2.0/24 via 10.111.2.0 dev flannel.1 onlink
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

可以通过如下命令获取tunnel的对端mac [root@kube01 ~]# ip neigh show dev flannel.1 10.111.1.0 lladdr ea:5e:a4:e0:0e:3f PERMANENT 10.111.2.0 lladdr de:11:58:59:6b:ba PERMANENT

实验

> 变更UDP模式为VXLAN模式，建议重置集群，否则可能出现异常

使用deployment-ex.yaml部署3副本应用，抓包测试

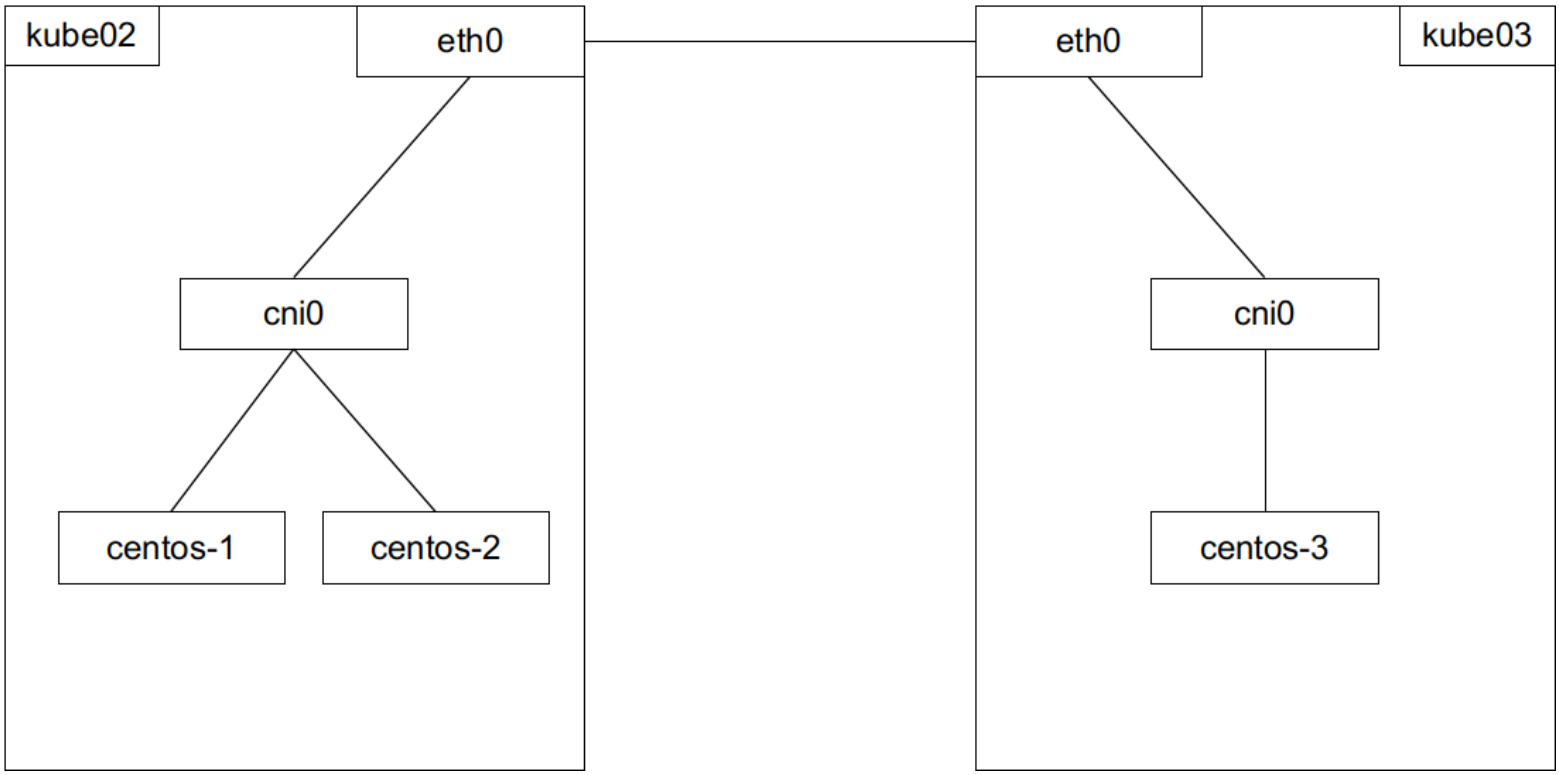
分别抓包：

- cni0
- flannel.1
- host interface

6. flannel hostgw网络

> 修改flannel模式host-gw

原理图



host-gw模式的路由表：

```
[root@kube01 ~]# ip route
default via 10.10.100.2 dev ens33 proto static metric 100
10.10.100.0/24 dev ens33 proto kernel scope link src 10.10.100.133 metric 100
10.111.0.0/24 dev cni0 proto kernel scope link src 10.111.0.1
10.111.1.0/24 via 10.10.100.134 dev ens33
10.111.2.0/24 via 10.10.100.135 dev ens33
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

实验：

抓包并查看路由表

7. POD TO SERVICE

IPTABLES VS IPVS

工作于内核空间的IPVS相较于iptalbes具有更好的性能

IPVS实现了四层的负载均衡

使用IPVS之后需要对Virtual Server进行管理，IPVS使用关在INPUT钩子上的DNAT表，因此需要让内核意识到VIP是本机的地址。Kubernetes通过设置将Service ClusterIP绑定到虚拟网卡上。

```
5: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
    link/ether 4e:71:c3:80:7f:4e brd ff:ff:ff:ff:ff:ff
    inet 10.112.0.10/32 scope global kube-ipvs0
       valid_lft forever preferred_lft forever
    inet 10.112.0.1/32 scope global kube-ipvs0
       valid_lft forever preferred_lft forever

[root@kube01 ~]# kubectl get svc --all-namespaces -o wide
NAMESPACE     NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                  AGE   SELECTOR
default       kubernetes   ClusterIP   10.112.0.1    <none>        443/TCP                  10h   <none>
kube-system   kube-dns     ClusterIP   10.112.0.10   <none>        53/UDP,53/TCP,9153/TCP   10h   k8s-
app=kube-dns
```

IPVS使用netfilter进行包过滤、SNAT、Masquared。具体来说，ipvs使用ipset来存储需要DROP和Masquared的流量，以确保netfilter规则数量是恒定的。

流程

POD to Service

1. POD1通过关联到POD1_NS的eth0离开
2. 通过veth设备到（cni0）网桥
3. 由于服务的IP和POD的IP不在同一网段，因此会走默认路由
4. 数据包发送给ens33接口前，包首先要接受iptables的filter表进行过滤
5. 收到数据包后，iptables使用kube-proxy在节点上安装的响应服务或POD事件的规则，将数据包的目的地从服务CLASTERIP改写为特定的PODIP
6. 数据包直接到达POD。Iptables的conntrack标记选择，以实现流量的定向

Service to Pod

响应POD标记src为自己，dst为发起请求的POD，经过Iptables的conntrack，变更src PODIP为ServiceIP 完成响应

8. Internet流量

Egress路由流量到Internet

1. src为POD的数据包通过veth pair连接的CNI0
2. 由于包不在网桥的网段上，尝试通过默认路由传递数据包
3. 通过netfilter,将src PODIP 转换为ens33 IP
4. 转换地址后到达网关
5. 互联网网关重写src为互联网接口地址
6. 返回过程遵项相同的路径

9. calico介绍

WHAT

Calico是一个开源SDN网络和安全解决方案。Calico支持广泛的平台，包括Kubernetes、OpenShift、Docker EE、OpenStack和bare metal服务。

特性：

- 灵活的功能、随处运行的安全措施
- 内核及性能
- 可伸缩

WHY

1. 强安全策略模型支持，策略引擎可以支持主机网络和服务网络
2. Calico使用Linux内核高度优化的转发和访问控制功能来提供本地Linux网络数据平面性能，通常不需要任何与第一代SDN网络相关的encap/decap开销，性能更为优越
3. 云原生机制，扩展性良好
4. 支持kubernetes与非kubernetes负载的互操作，所有负载支持统一策略模型
5. 多数情况下是非封装非覆盖的，使得追踪流量更为容易
6. 对kubernetes网络策略的支持

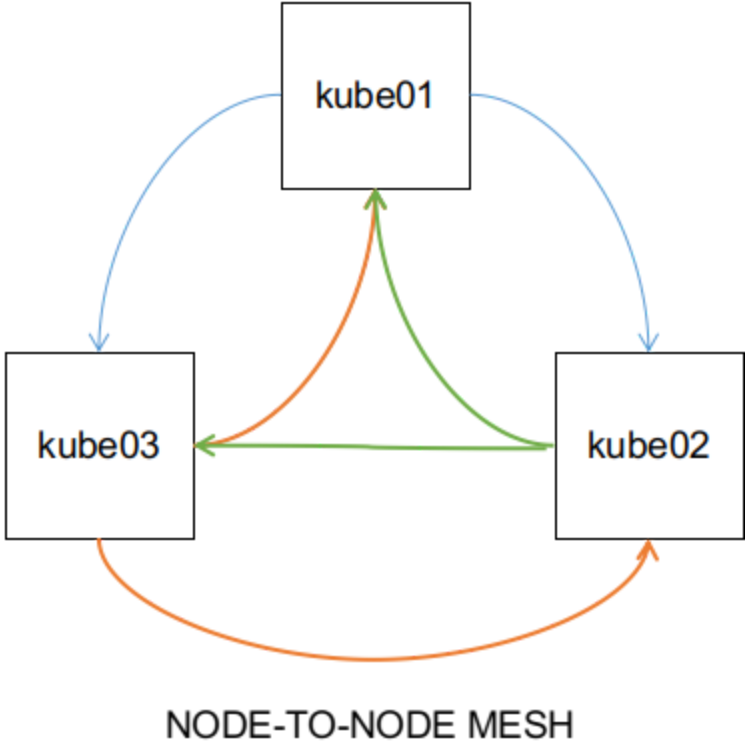
Calico主要组成

- calico cni插件：对接kubernetes，供kubelet调用
- calico node: felix(维护路由表，FIB转发信息库) + BIRD（负责路由分发）
  - Felix
    - 接口和路由管理
    - 状态报告
    - 强制策略
  - BIRD
    - 策略分发
- confid：配置管理组建

基本架构

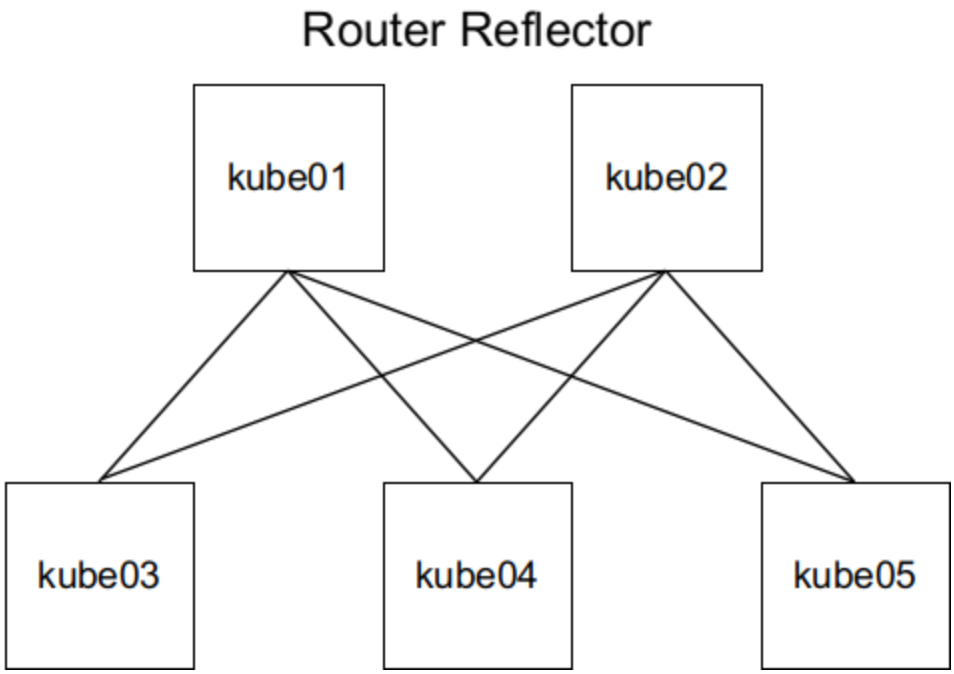
calico与flannel不同，并不创建桥接设备，而是通过路由表维护Pod的通信

Node被视为边界路由，从而Kubernetes集群形成了node-to-node-mesh的全联通拓扑结构



连接数量随节点数以N的2次方增长，会对集群网络产生巨大压力，一般情况下该种模式规模应控制在50（官方为100）节点以内

为解决上述问题，还可以使用Router Reflector



组件协同实验

部署calico网络，部署3副本nginx

新插件下的网卡状态

```
6: vxlan.calico: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN mode DEFAULT group
default
    link/ether 66:5b:2b:b8:df:0c brd ff:ff:ff:ff:ff:ff
7: cali32cdc3f1869@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP mode DEFAULT
```

```
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
8: cali8ff83a55cce@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
9: cali93274caaa80@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 2
```

新网络状态下的路由信息

```
[root@kube01 ~]# ip route
default via 10.10.100.2 dev ens33 proto static metric 100
10.10.100.0/24 dev ens33 proto kernel scope link src 10.10.100.133 metric 100
10.111.34.1 dev cali32cdc3f1869 scope link
10.111.34.2 dev cali8ff83a55cce scope link
10.111.34.3 dev cali93274caaa80 scope link
10.111.87.0/24 via 10.10.100.134 dev ens33 proto 80 onlink
10.111.164.0/24 via 10.10.100.135 dev ens33 proto 80 onlink
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

部署calicoctl

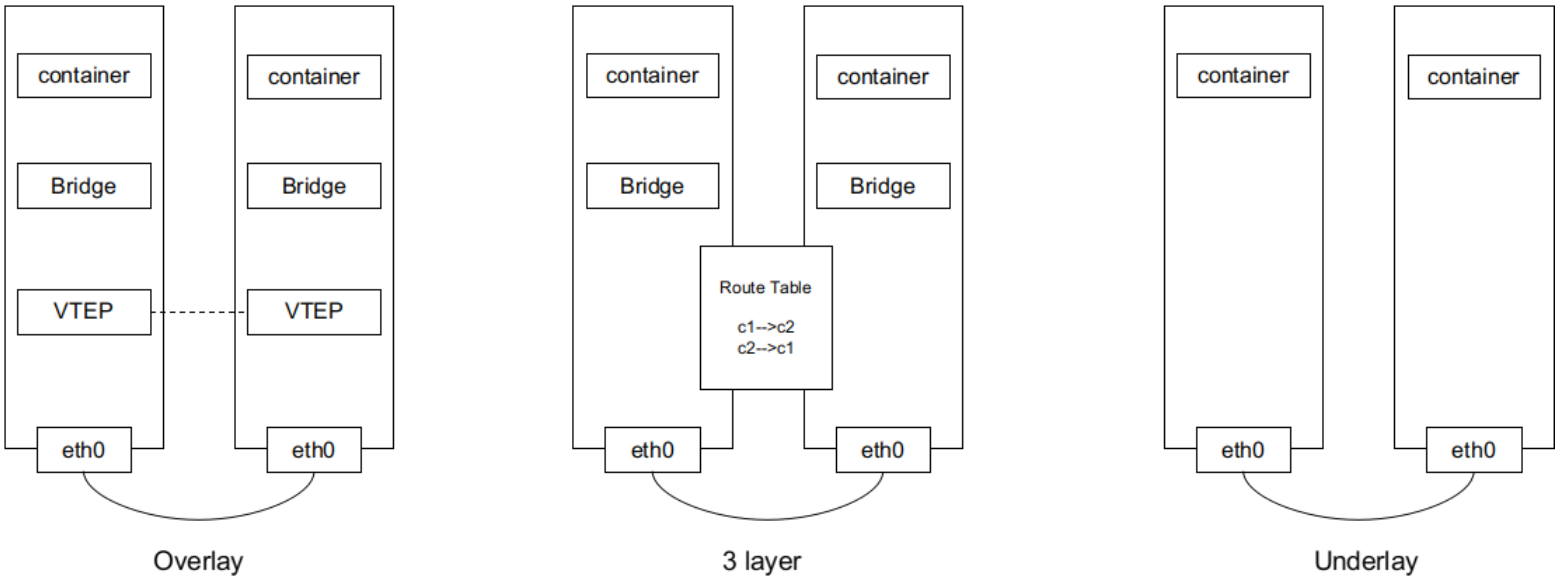
```
wget https://github.com/projectcalico/calicoctl/releases/download/v3.14.0/calicoctl
chmod +x calicoctl
sudo mv calicoctl /usr/local/bin/

# 使用calicoctl

export KUBECONFIG=/path/to/your/kubeconfig
export DATASTORE_TYPE=kubernetes

# 或者
vim /etc/calico/calicoctl.cfg

apiVersion: projectcalico.org/v3
kind: CalicoAPIConfig
metadata:
spec:
    datastoreType: "kubernetes",
    kubeconfig: "/root/.kube/config"


[root@kube01 ~]# calicoctl node status
Calico process is running.

IPv4 BGP status
+--------------+-------------------+-------+----------+-------------+
| PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+--------------+-------------------+-------+----------+-------------+
| 10.10.100.134 | node-to-node mesh | up    | 01:17:18 | Established |
| 10.10.100.135 | node-to-node mesh | up    | 01:17:19 | Established |
+--------------+-------------------+-------+----------+-------------+

IPv6 BGP status
No IPv6 peers found.
```

10. 选择适合自己的网络选项

Calico支持多个容器网络选项，可用于伸缩性、网络性能和与现有物理网络的互操作。

可选项

- 非封装基于IP路由的网络
- 封装的Oerlay网络
- 物理网络



Calico网络：

- 使用IPAM管理POD的IP地址
- 编写本地路由表
- 将路由分配给其他节点或网络设备

网络选项

网络选项	环境	性能	复杂度	Notes
calico,非封装,物理网络配对	本地	Best	中等	允许pods从集群外部直接访问
calico,非封装,不与物理网络配对	本地layer2	Best	低	IP in IP or VXLAN跨子网封装
calico，IPIP，封装	本地,大多数共有云	依赖硬件	低	
calico，VXLAN，封装	本地，大多数共有云	依赖硬件	低	
AWS VPC CNI	Amazon EKS	Excellent	低	不支持calico IPAM的全部功能

网络选项	环境	性能	复杂度	Notes
Azure CNI	Microsoft AKS	Excellent	低	不支持calico IPAM的全部功能
Google Cloud	Google GKE	Excellent	低	不支持calico IPAM的全部功能
Flannel	任何共有网络	Poor to excellent,以来选择的后端	中等	不支持calico IPAM的全部功能

11. IPinIP

Calico可以为每个IP池配置不同的封装模式，但不能在同一个池中混用

- IP-in-IP Encapsulation for only cross subnet
- IP-in-IP Encapsulation for all
- VXLAN Encapsulation for only cross subnet
- VXLAN Encapsulation for all

最佳实践

Calico 跨子网封装模式

```
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  name: default
spec:
  calicoNetwork:
    ipPools:
    - blockSize: 24
      cidr: 10.111.0.0/16
      encapsulation: IPIPCrossSubnet
      natOutgoing: Enabled
      nodeSelector: all()
```

部署完成后网卡及路由变化

查看Ippool的信息

```
[root@kube01 ~]# calicoctl get ippool -o yaml
apiVersion: projectcalico.org/v3
items:
- apiVersion: projectcalico.org/v3
  kind: IPPool
  metadata:
    creationTimestamp: "2021-07-09T06:27:36Z"
    name: default-ipv4-ippool
    resourceVersion: "129078"
```

```
    uid: 4bfcdf6e-1edc-49bf-a8d2-e4a6a3a1d92d
  spec:
    blockSize: 24
    cidr: 10.111.0.0/16
    ipipMode: CrossSubnet
    natOutgoing: true
    nodeSelector: all()
    vxlanMode: Never
kind: IPPoolList
metadata:
  resourceVersion: "129343"
```

网卡信息

```
6: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1480 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen
1000
    link/ipip 0.0.0.0 brd 0.0.0.0
7: cali93274caaa80@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
8: cali8ff83a55cce@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

路由信息

```
[root@kube01 ~]# ip route
default via 10.10.100.2 dev ens33 proto static metric 100
10.10.100.0/24 dev ens33 proto kernel scope link src 10.10.100.133 metric 100
blackhole 10.111.34.0/24 proto bird
10.111.34.1 dev cali93274caaa80 scope link
10.111.34.2 dev cali8ff83a55cce scope link
10.111.87.0/24 via 10.10.100.134 dev ens33 proto bird
10.111.164.0/24 via 10.10.100.135 dev ens33 proto bird
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

直接变更ipipMode

```
[root@kube01 ~]# calicoctl patch ippool default-ipv4-ippool -p '{"spec":{"ipipMode":"Always"}}'
Successfully patched 1 'IPPool' resource
```

查看路由变化

```
[root@kube01 ~]# ip route
default via 10.10.100.2 dev ens33 proto static metric 100
10.10.100.0/24 dev ens33 proto kernel scope link src 10.10.100.133 metric 100
blackhole 10.111.34.0/24 proto bird
10.111.34.1 dev cali93274caaa80 scope link
10.111.34.2 dev cali8ff83a55cce scope link
10.111.87.0/24 via 10.10.100.134 dev tunl0 proto bird onlink
10.111.164.0/24 via 10.10.100.135 dev tunl0 proto bird onlink
```

```
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

12. VXLAN

针对跨子网流量封装

全部流量封装配置

13. MTU调整

常见MTU

NETWORK MTU	MTU	MTU with IPIP	MTU with VXLAN
1500	1500	1480	1450
9000	9000	8980	8950
1460(GCE)	1460	1440	1410
9001(AWS jumbo)	9001	8981	8951
1450(openstack VXLAN)	1450	1430	1400

```
[root@kube01 ~]# ip link show
6: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1480 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen
1000
    link/ipip 0.0.0.0 brd 0.0.0.0
7: cali93274caaa80@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
8: cali8ff83a55cce@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP mode DEFAULT
group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

14. Calico 及 kubernetes 网络策略

- 外部
  - 灵活的策略匹配规则
  - 结合istio，提供5-7层策略支持
- 策略跨平台特性
- 与kubernetes策略无缝衔接

功能

Calico策略支持如下功能：

- 策略可作用于 Pods,containers,Vms或者host insterface
- 策略可定义ingess和egress两个方向
- 策略支持：

    - Actions:allow,deny,log,pass
    - Match criteria
        - Ports: numbered,ports in a range, and kubernetes named ports
        - Protocols: TCP,UDP,ICMP,SCTP,UDPlite,ICMPv6,protocol numbers(1-255)
        - HTTP attributes
        - ICMP attributes
        - IP version
        - IP or CIDR
        - endpoints selectors
        - service account selectors
        - namesapce selectors

kubernetes网络策略规则

- 默认pod to pod均被允许
- 有ingress规则，则仅仅ingress允许的流量可以放行
- 有egress规则，则仅仅egress允许的流量可以放行

1. Enable default deny calico global network policy, non-namespaced

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: default-deny
spec:
  selector: all()
  types:
  - Ingress
  - Egress
```

2. Enable default deny Calico network policy, namespaced

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: engineering
spec:
  selector: all()
  types:
  - Ingress
  - Egress
```

3. Enable default deny kubernetes policy, namespaced

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
```

```
  namespace: engineering
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

实验

```
# 网络策略模板：
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: allow-tcp-6379
  namespace: production
spec:
  selector: .......
  ...
```

与calico网络策略一样，kubernetes的网络策略实际上是开启白名单

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: a-rule
spec:
  podSelector: {}
  policyTypes:
  - Ingress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: b-rule
spec:
  podSelector: {}
  policyTypes:
  - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: c-rule
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: d-rule
spec:
  podSelector: {}
  policyTypes:
```

```yaml
  - Ingress
  ingress:
  - {}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: e-rule
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - {}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```