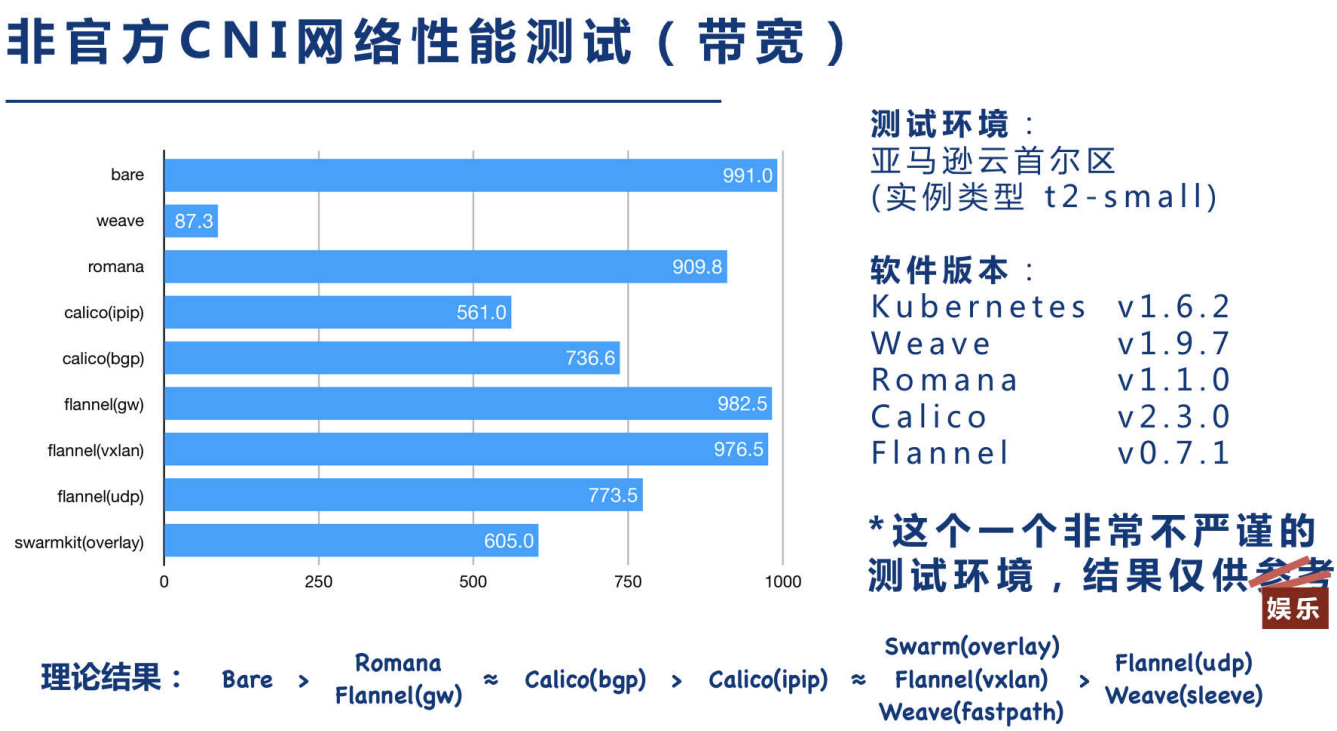
kubernetes网络方案部署

写在前面的话

现在网络上流传很多kubernetes的部署和搭建的文档，其中比较出名就是kubernetes-the-hard-

way，还有基于这个翻译和衍生的版本follow-me-install-kubernetes-cluster，这2篇文章带我走过了kubernetes的搭建的童年，我第一搭建成功就是抄袭的张俊的follow-me-install-kubernetes-cluster，然后随着新版的发展，越来越多的配置参数存在各种各样的问题，最大的问题是在cni产生后，2篇文章的配置参数和系统默认或者cni配置参数有稍微的冲突导致很多人在利用cni做网络方案的时候会出现很多问题，这篇文章目的第一是想2位前辈致敬，第二是共享下在flanneld和calico 部署过程中遇到挫折和问题。

为啥只说明以下2种方案的部署模式，因为其他网络方案不争气。附图（图是抄袭的，作者别杀我）



Flanneld [systemd部署模式]

flannel 部署相对来说简单容易，坑少，直接上配置。

flannel.service

```
[Unit]
Description=Flanneld overlay address etcd agent
After=network.target
After=network-online.target
```

```
Wants=network-online.target
Before=docker.service
[Service]
Type=notify
ExecStart=/usr/local/bin/flanneld \
  -etcd-cafile=/etc/kubernetes/ssl/ca.pem \
  -etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem \
  -etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem \
  -etcd-endpoints=https://{Etcd IP}:2379 \
  -iface=ens3 \
  --ip-masq
Restart=on-failure
[Install]
WantedBy=multi-user.target
RequiredBy=docker.service
```

记住一定要提前在etcd把你的backend写进去。etc

```
etcdctl \
  --endpoints=https://{Etcd-IP}:2379 \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  set /coreos.com/network/config '{"Network":"'10.200.0.0/16'", "SubnetLen":
 24, "Backend": {"Type":"host-gw"}}'
```

然后可以开始你的表演，如果你创建2个pod，互ping以下发现不通，你部署dns服务，一直报错no route to host（就是kubernetes那个svc 对应的IP）,恭喜你，你下面要做的就是

```
iptables -P FORWARD ACCEPT
```

因为1.13版本以上docker好似在iptable 写了这么一条策略

```
iptables -P FORWARD DROP
```

到此为止你起来flanneld 就可以开始你的k8s 之旅，当然现在好似还不行记得在/etc/cni/net.d 下写一个.conf结尾的文件，当然叫什么名字无所谓。etc
10-flanneld-cni.conf

```
{
      "name": "cbr0",
      "type": "flannel",
```

```
      "delegate": {
        "isDefaultGateway": true
      }
}
```

然后继续你的表演就可以了。

Calico[systemd部署模式]

其实吧,calico在kubernetes网络方案用用的比flanneld多，calico懂得玩伸缩，技术也比较牛，在很多物理设备不开启BGP的情况下做了折中，用的IP-IP虽然性能有点损失，在云上被大面积使用。flanneld的host-gw模式性能虽然不错，但是只能在2层玩下，过了二层路由被重写就gg了。开始表演IP-IP模式。
第一步创建IPpool,pool就是所有calico分配ip的池子，其实就是k8s的pool，不过calico分配出来是/26的ip，一下少很多。其实我觉得/26比较符合机器配置的现状至少，不会造成ip的浪费。

设置pool

```
calicoctl apply -f - << EOF
apiVersion: v1
kind: ipPool
metadata:
  cidr: 10.200.0.0/16
spec:
  ipip:
    enabled: true
    mode: cross-subnet
  nat-outgoing: true
EOF
```

部署calico-node(其实就2个文件)

calico.env

```
ETCD_ENDPOINTS="https://{ETCD1}:2379,https://{ETCD2}:2379"
ETCD_CA_FILE="/etc/kubernetes/ssl/ca.pem"
ETCD_CERT_FILE="/etc/kubernetes/ssl/kubernetes.pem"
ETCD_KEY_FILE="/etc/kubernetes/ssl/kubernetes-key.pem"
CALICO_NODENAME="node46"
CALICO_NO_DEFAULT_POOLS=""
CALICO_IP="{HOST-IP}"
CALICO_IP6=""
CALICO_AS=""
CALICO_LIBNETWORK_ENABLED=true
CALICO_NETWORKING_BACKEND=bird
```

```
[Unit]
Description=calico-node
After=docker.service
Requires=docker.service

[Service]
EnvironmentFile=/etc/calico/calico.env
ExecStartPre=-/usr/bin/docker rm -f calico-node
ExecStart=/usr/bin/docker run --net=host --privileged \
 --name=calico-node \
 -e NODENAME=${CALICO_NODENAME} \
 -e IP=${CALICO_IP} \
 -e IP6=${CALICO_IP6} \
 -e CALICO_NETWORKING_BACKEND=${CALICO_NETWORKING_BACKEND} \
 -e CALICO_STARTUP_LOGLEVEL=DEBUG \
 -e NO_DEFAULT_POOLS=${CALICO_NO_DEFAULT_POOLS} \
 -e FELIX_DEFAULTENDPOINTTOHOSTACTION=ACCEPT \
 -e CALICO_LIBNETWORK_ENABLED=${CALICO_LIBNETWORK_ENABLED} \
 -e ETCD_ENDPOINTS=${ETCD_ENDPOINTS} \
 -e ETCD_CA_CERT_FILE=/etc/kubernetes/ssl/ca.pem \
 -e ETCD_CERT_FILE=/etc/kubernetes/ssl/kubernetes.pem \
 -e ETCD_KEY_FILE=/etc/kubernetes/ssl/kubernetes-key.pem \
 -v /var/log/calico:/var/log/calico \
 -v /run/docker/plugins:/run/docker/plugins \
 -v /lib/modules:/lib/modules \
 -v /var/run/calico:/var/run/calico \
 -v /etc/kubernetes/ssl/:/etc/kubernetes/ssl/:ro \
 quay.io/calico/node:v2.4.0
```

```
ExecStop=-/usr/bin/docker stop calico-node

[Install]
WantedBy=multi-user.target
```

cni配置文件（/etc/cni/net.d下随便写一个.conf结尾的文件）

calico.conf

```
{
    "name": "k8s-pod-network",
    "cniVersion": "0.1.0",
    "type": "calico",
    "etcd_endpoints": "https://{ETCD1},https://{ETCD2}:2379",
    "etcd_key_file": "/etc/kubernetes/ssl/kubernetes-key.pem",
    "etcd_cert_file": "/etc/kubernetes/ssl/kubernetes.pem",
    "etcd_ca_cert_file": "/etc/kubernetes/ssl/ca.pem",
    "log_level": "info",
    "ipam": {
        "type": "calico-ipam"
     },
    "kubernetes": {
        "kubeconfig": "/etc/kubernetes/kube-proxy.kubeconfig"
    }
}
```

然后你就可以装逼了，记得把cni的组件calico-ipam放到/opt/cni/bin

RR部署模式

其实就是在以上的基础上多部署一个RR容器

```
docker run --privileged --net=host -d \
           -e IP={HOST-IP} \
           -e ETCD_ENDPOINTS=https://{ETCD}:2379 \
           -e ETCD_CA_CERT_FILE=/etc/kubernetes/ssl/ca.pem \
           -e ETCD_CERT_FILE=/etc/kubernetes/ssl/kubernetes.pem \
           -e ETCD_KEY_FILE=/etc/kubernetes/ssl/kubernetes-key.pem \
           -v /etc/kubernetes/ssl:/etc/kubernetes/ssl:ro \
           calico/routereflector:v0.4.0
```

然后把这个IP写到etcd里

```
curl --cacert /etc/kubernetes/ssl/ca.pem --cert
/etc/kubernetes/ssl/kubernetes.pem --key
/etc/kubernetes/ssl/kubernetes-key.pem -L https://{ETCD}:2379/v2/keys/calico
/bgp/v1/rr_v4/{HOST-IP} -XPUT -d value="{\"ip\":\"{HOST-IP}\",\"cluster_id\"
:\"1.0.0.2\"}
```

然后就是创建一个全局的BGPpeer

```
cat << EOF | calicoctl delete -f -
apiVersion: v1
kind: bgpPeer
metadata:
  peerIP: {HOST-IP}
  scope: global
spec:
  asNumber: 64567
EOF
```

关闭 node-to-node mesh

```
calicoctl config set nodeToNodeMesh off
```

重新启动你所有的calico-node
附一份calicoctl的配置文件(在/etc/calico下)
calicoctl.cfg

```
apiVersion: v1
kind: calicoApiConfig
metadata:
spec:
  etcdEndpoints: https://{ETCD}:2379
  etcdKeyFile: /etc/kubernetes/ssl/kubernetes-key.pem
  etcdCertFile: /etc/kubernetes/ssl/kubernetes.pem
  etcdCACertFile: /etc/kubernetes/ssl/ca.pem
```

开始你的装逼之旅吧
关于calico-bgp 有硬件的自己尝试吧

附一份非标准的kube-proxy.service

```
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \
  --hostname-override={HOST-IP} \
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```