kubernetes master 节点包含的组件：

kube-apiserver

kube-scheduler

kube-controller-manager

目前这三个组件需要部署在同一台机器上。

kube-scheduler、kube-controller-manager 和 kube-apiserver 三者的功能紧密相关；

同时只能有一个 kube-scheduler、kube-controller-manager 进程处于工作状态，如果运行多个，则需要通过选举产生一个 leader；

验证 TLS 证书文件及 token.csv 文件

pem 和 token.csv 证书文件我们在 TLS 证书和秘钥这一步中已经创建过了。我们再检查一下。

```
 ls /etc/kubernetes/ssl

admin-key.pem  admin.pem  ca-key.pem  ca.pem  kube-proxy-key.pem  kube-proxy.pem  ku
bernetes-key.pem  kubernetes.pem

ls /etc/kubernetes/token.csv
```

下载最新版本的二进制文件

从 github release 页面 下载发布版 tarball，解压后再执行下载脚本

```
wget https://dl.k8s.io/v1.7.6/kubernetes-server-linux-amd64.tar.gz

tar xf /root/k8s/kubernetes-server-linux-amd64.tar.gz -C /usr/local/

cat > /etc/profile.d/kube-apiserver.sh << EOF

export PATH=/usr/local/kubernetes/server/bin:$PATH

EOF
```

配置和启动 kube-apiserver

serivce 配置文件/usr/lib/systemd/system/kube-apiserver.service 内容

```
cat  /usr/lib/systemd/system/kube-apiserver.service

[Unit]

Description=Kubernetes API Service

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

After=network.target

After=etcd.service



[Service]

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/apiserver

ExecStart=/usr/local/kubernetes/server/bin/kube-apiserver \
```

```
        $KUBE_LOGTOSTDERR \
        $KUBE_LOG_LEVEL \
        $KUBE_ETCD_SERVERS \
        $KUBE_API_ADDRESS \
        $KUBE_API_PORT \
        $KUBELET_PORT \
        $KUBE_ALLOW_PRIV \
        $KUBE_SERVICE_ADDRESSES \
        $KUBE_ADMISSION_CONTROL \
        $KUBE_API_ARGS
Restart=on-failure
Type=notify
LimitNOFILE=65536


[Install]
```

```
WantedBy=multi-user.target
```

/etc/kubernetes/config 文件的内容为：

```
cat   config




###

# kubernetes system config

#

# The following values are used to configure various aspects of all

# kubernetes services, including

#

#   kube-apiserver.service

#   kube-controller-manager.service

#   kube-scheduler.service

#   kubelet.service

#   kube-proxy.service

# logging to stderr means we get it in the systemd journal
```

```
KUBE_LOGTOSTDERR="--logtostderr=true"



# journal message level, 0 is debug

KUBE_LOG_LEVEL="--v=0"



# Should this cluster be allowed to run privileged docker containers

KUBE_ALLOW_PRIV="--allow-privileged=true"
```

```
# How the controller-manager, scheduler, and proxy find the apiserver

#KUBE_MASTER="--master=http://sz-pg-oam-docker-test-001.tendcloud.com:8080"

KUBE_MASTER="--master=http://172.16.200.216:8080"
```

注意：该配置文件同时被 kube-apiserver、kube-controller-manager、kube-scheduler、kubelet、kube-proxy 使用。

apiserver 配置文件/etc/kubernetes/apiserver 内容为：

```
cat  apiserver



###

## kubernetes system config

##

## The following values are used to configure the kube-apiserver

##

#

## The address on the local server to listen to.

#KUBE_API_ADDRESS="--insecure-bind-address=sz-pg-oam-docker-test-001.tendcloud.com"

KUBE_API_ADDRESS="--advertise-address=172.16.200.216 --bind-address=172.16.200.216
--insecure-bind-address=172.16.200.216"
```

```
#

## The port on the local server to listen on.

#KUBE_API_PORT="--port=8080"

#

## Port minions listen on

#KUBELET_PORT="--kubelet-port=10250"

#
```

```
## Comma separated list of nodes in the etcd cluster

KUBE_ETCD_SERVERS="--etcd-servers=http://172.16.200.100:2379,http://172.16.200.101:
2379,http://172.16.200.102:2379"

#

## Address range to use for services

KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"

#

## default admission control policies

KUBE_ADMISSION_CONTROL="--admission-control=ServiceAccount,NamespaceLifecycle,Names
paceExists,LimitRanger,ResourceQuota"

#

## Add your own!
```

```
KUBE_API_ARGS="--authorization-mode=RBAC
--runtime-config=rbac.authorization.k8s.io/v1beta1 --kubelet-https=true
--experimental-bootstrap-token-auth --token-auth-file=/etc/kubernetes/token.csv
--service-node-port-range=30000-32767
--tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem
--tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem
--client-ca-file=/etc/kubernetes/ssl/ca.pem
--service-account-key-file=/etc/kubernetes/ssl/ca-key.pem
--etcd-cafile=/etc/kubernetes/ssl/ca.pem
--etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem
--etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem --enable-swagger-ui=true
--apiserver-count=3 --audit-log-maxage=30 --audit-log-maxbackup=3
--audit-log-maxsize=100 --audit-log-path=/var/lib/audit.log --event-ttl=1h
--log-dir=/data/logs/kubernetes/ --v=2 --logtostderr=false"
```

--authorization-mode=RBAC 指定在安全端口使用 RBAC 授权模式，拒绝未通过授权的请求；

kube-scheduler、kube-controller-manager 一般和 kube-apiserver 部署在同一台机器上，它们使用非安全端口和 kube-apiserver 通信;

kubelet、kube-proxy、kubectl 部署在其它 Node 节点上，如果通过安全端口访问 kube-apiserver，则必须先通过 TLS 证书认证，再通过 RBAC 授权；

kube-proxy、kubectl 通过在使用的证书里指定相关的 User、Group 来达到通过 RBAC 授权的目的；

如果使用了 kubelet TLS Boostrap 机制，则不能再指定 --kubelet-certificate-authority、--kubelet-client-certificate 和 --kubelet-client-key 选项，否则后续 kube-apiserver 校验 kubelet 证书时出现 ”x509: certificate signed by unknown authority“ 错误；

--admission-control 值必须包含 ServiceAccount；

--bind-address 不能为 127.0.0.1；

runtime-config 配置为 rbac.authorization.k8s.io/v1beta1，表示运行时的 apiVersion；

--service-cluster-ip-range 指定 Service Cluster IP 地址段，该地址段不能路由可达；

- 缺省情况下 **kubernetes** 对象保存在 **etcd /registry** 路径下，可以通过 **--etcd-prefix** 参数进行调整；

启动 **kube-apiserver**

```
systemctl daemon-reload

systemctl enable kube-apiserver

systemctl start kube-apiserver

systemctl status kube-apiserver
```

配置和启动 **kube-controller-manager**

创建 kube-controller-manager 的 serivce 配置文件

```
 cat /usr/lib/systemd/system/kube-controller-manager.service

Description=Kubernetes Controller Manager

Documentation=https://github.com/GoogleCloudPlatform/kubernetes



[Service]
```

```
EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/controller-manager

ExecStart=/usr/local/kubernetes/server/bin/kube-controller-manager \

        $KUBE_LOGTOSTDERR \

        $KUBE_LOG_LEVEL \

        $KUBE_MASTER \

        $KUBE_CONTROLLER_MANAGER_ARGS

Restart=on-failure

LimitNOFILE=65536



[Install]

WantedBy=multi-user.target
```

配置文件/etc/kubernetes/controller-manager

```
cat  controller-manager



###

# The following values are used to configure the kubernetes controller-manager



# defaults from config and apiserver should be adequate



# Add your own!

KUBE_CONTROLLER_MANAGER_ARGS="--address=127.0.0.1
--service-cluster-ip-range=10.254.0.0/16 --cluster-name=kubernetes
--cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem
--cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem  --service-account-private
-key-file=/etc/kubernetes/ssl/ca-key.pem  --root-ca-file=/etc/kubernetes/ssl/ca.pem
--leader-elect=true --log-dir=/data/logs/kubernetes/ --v=2 --logtostderr=false"
```

--service-cluster-ip-range 参数指定 **Cluster** 中 **Service** 的 **CIDR** 范围，该网络在各 **Node** 间必须路由不可达，必须和 **kube-apiserver** 中的参数一致；

--cluster-signing-* 指定的证书和私钥文件用来签名为 **TLS BootStrap** 创建的证书和私钥；

--root-ca-file 用来对 **kube-apiserver** 证书进行校验，指定该参数后，才会在 **Pod** 容器的 **ServiceAccount** 中放置该 **CA** 证书文件；

--address 值必须为 **127.0.0.1**，因为当前 **kube-apiserver** 期望 **scheduler** 和 **controller-manager** 在同一台机器,否则机器不能选举

--leader-elect=true 允许集群选举

启动 kube-controller-manager

```
systemctl daemon-reload

systemctl enable kube-controller-manager

systemctl start kube-controller-manager
```

配置和启动 kube-scheduler

创建 kube-scheduler 的 serivce 配置文件

```
cat  /usr/lib/systemd/system/kube-scheduler.service

[Unit]

Description=Kubernetes Scheduler Plugin

Documentation=https://github.com/GoogleCloudPlatform/kubernetes



[Service]

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/scheduler

ExecStart=/usr/local/kubernetes/server/bin/kube-scheduler \

        $KUBE_LOGTOSTDERR \
```

```
        $KUBE_LOG_LEVEL \

        $KUBE_MASTER \

        $KUBE_SCHEDULER_ARGS

Restart=on-failure

LimitNOFILE=65536



[Install]

WantedBy=multi-user.target
```

kube-scheduler 配置文件

```
###

# kubernetes scheduler config



# default config should be adequate



# Add your own!

KUBE_SCHEDULER_ARGS="--leader-elect=true --address=127.0.0.1
--log-dir=/data/logs/kubernetes/ --v=2 --logtostderr=false"

EOF
```

-

--address 值必须为 127.0.0.1，因为当前 kube-apiserver 期望 scheduler 和 controller-manager 在同一台机器；

- 启动 **kube-scheduler**

```
systemctl daemon-reload

systemctl enable kube-scheduler
```

```
systemctl start kube-scheduler
```

验证 master 节点功能

```
kubectl get componentstatuses

NAME                 STATUS    MESSAGE             ERROR

controller-manager   Healthy   ok

scheduler            Healthy   ok

etcd-0               Healthy   {"health": "true"}

etcd-1               Healthy   {"health": "true"}

etcd-2               Healthy   {"health": "true"}
```

部署 node 节点

kubernetes node 节点包含如下组件：

Flanneld：使用 flanneld-0.8 支持阿里云 host-gw 模式，以获取最佳性能。

Docker17.07.0-ce：docker 的安装很简单，这里也不说了。

kubelet

kube-proxy

下面着重讲 kubelet 和 kube-proxy 的安装，同时还要将之前安装的 flannel 集成 TLS 验证。

注意：每台 node 上都需要安装 flannel，master 节点上可以不必安装。

检查目录和文件

我们再检查一下三个节点上，经过前几步操作生成的配置文件。

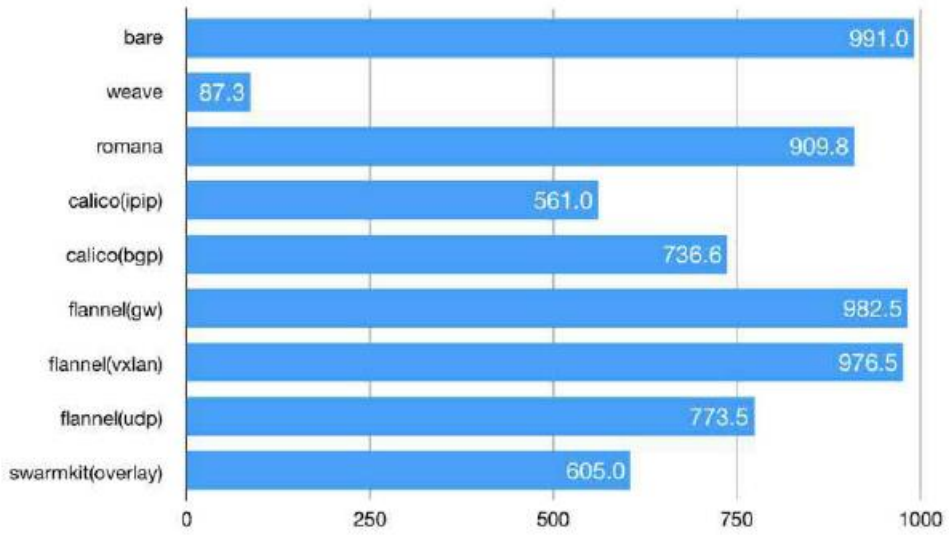
```
# ls /etc/kubernetes/ssl/

admin-key.pem  admin.pem  ca-key.pem  ca.pem  kube-proxy-key.pem  kube-proxy.pem  ku
bernetes-key.pem  kubernetes.pem

# ls /etc/kubernetes/

apiserver  bootstrap.kubeconfig  config  controller-manager  kube-proxy.kubeconfig
scheduler  ssl  token.csv
```

flannel 网络架构图

各主流CNI实现的总结

	覆盖网络	主机路由	网络策略	去中心化的IP地址分配
Flannel	UDP/XVLAN	HostGW	N	N
Calico	IPIP	BGP	Y	N
Canal	UDP/XVLAN/IPIP	HostGW/BGP	Y	N
Romana	N	HostGW	Y	N
Weave	UDP/XVLAN	N	Y	Y

非官方CNI网络性能测试（带宽）



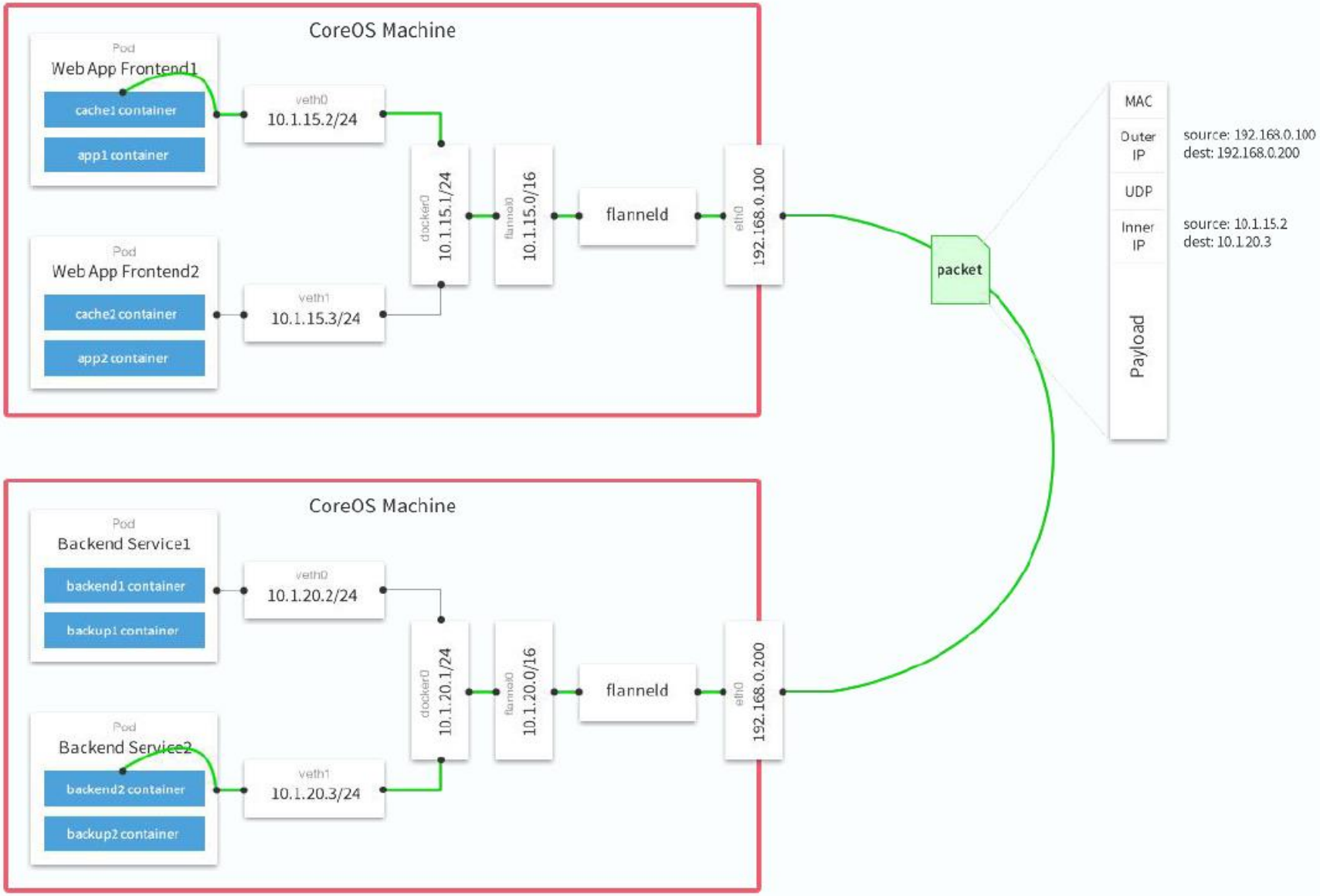
测试环境：
亚马逊云首尔区
(实例类型 t2-small)

软件版本：

Kubernetes	v1.6.2
Weave	v1.9.7
Romana	v1.1.0
Calico	v2.3.0
Flannel	v0.7.1

***这个一个非常不严谨的测试环境，结果仅供~~参考~~娱乐**

理论结果： Bare > Romana Flannel(gw) ≈ Calico(bgp) > Calico(ipip) ≈ Swarm(overlay) Flannel(vxlan) Weave(fastpath) > Flannel(udp) Weave(sleeve)


CoreOS Machine
Pod
Web App Frontend1
cache1 container
app1 container
Pod
Web App Frontend2
cache2 container
app2 container
veth0
10.1.15.2/24
veth1
10.1.15.3/24
docker0
10.1.15.1/24
flannel0
10.1.15.0/16
flanneld
eth0
192.168.0.100
CoreOS Machine
Pod
Backend Service1
backend1 container
backup1 container
Pod
Backend Service2
backend2 container
backup2 container
veth0
10.1.20.2/24
veth1
10.1.20.3/24
docker0
10.1.20.1/24
flannel0
10.1.20.0/16
flanneld
eth0
192.168.0.200
packet
MAC
Outer IP
UDP
Inner IP
Payload
source: 192.168.0.100
dest: 192.168.0.200
source: 10.1.15.2
dest: 10.1.20.3

配置安装 Flanneld，默认使用 yum 安装。需要替换二进制 flanneld

下载 flanned-0.8 binary.

```
yum install flanneld -y

wget https://github.com/coreos/flannel/releases/download/v0.8.0/flanneld-amd64

chmod +x flanneld-amd64

cp flanneld-amd64 /usr/bin/flanneld
```

service 配置文件/usr/lib/systemd/system/flanneld.service

```
cat  /usr/lib/systemd/system/flanneld.service
```

```
[Unit]

Description=Flanneld overlay address etcd agent

After=network.target

After=network-online.target

Wants=network-online.target

After=etcd.service

Before=docker.service



[Service]

Type=notify

EnvironmentFile=/etc/sysconfig/flanneld

EnvironmentFile=-/etc/sysconfig/docker-network

ExecStart=/usr/bin/flanneld-start $FLANNEL_OPTIONS

ExecStartPost=/usr/libexec/flannel/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -d
/run/flannel/docker

Restart=on-failure
```

```
[Install]

WantedBy=multi-user.target

RequiredBy=docker.service
```

/etc/sysconfig/flanneld 配置文件

```
 cat  /etc/sysconfig/flanneld

# Flanneld configuration options



# etcd url location.  Point this to the server where etcd runs
```

```
FLANNEL_ETCD_ENDPOINTS="http://172.16.200.100:2379,http://172.16.200.101:2379,http://172.16.200.102:2379"



# etcd config key.  This is the configuration key that flannel queries

# For address range assignment

ETCD_PREFIX="/kube-centos/network"

FLANNEL_ETCD_KEY="/kube-centos/network"

ACCESS_KEY_ID=XXXXXXX

ACCESS_KEY_SECRET=XXXXXXX

# Any additional options that you want to pass

#FLANNEL_OPTIONS=" -iface=eth0  -log_dir=/data/logs/kubernetes --logtostderr=false --v=2"

#FLANNEL_OPTIONS="-etcd-cafile=/etc/kubernetes/ssl/ca.pem -etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem -etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem"
```

设置 etcd 网络，主要是 flannel 用于分别 docker 的网络，‘/coreos.com/network/config’ 这个字段必须与 flannel 中的"FLANNEL*ETCD*KEY="/coreos.com/network" 保持一致

阿里云 VPN 网络模式详细配置可参考 AliCloud VPC Backend for Flannel

在 etcd 中创建网络配置 执行下面的命令为 docker 分配 IP 地址段。

```
etcdctl mkdir /kube-centos/network

etcdctl  mk /kube-centos/network/config
'{"Network":"10.24.0.0/16","Backend":{"Type":"ali-vpc"}}'

etcdctl  mk /kube-centos/network/config
'{"Network":"10.24.0.0/16","Backend":{"Type":"host-gw"}}'
```

安装 kubernetes-cni 依赖 kublete 强制安装

```
rpm -ivh --force kubernetes-cni-0.5.1-0.x86_64.rpm --nodeps
```

配置 cni 插件

```
mkdir -p  /etc/cni/net.d

cat > /etc/cni/net.d/10-flannel.conf << EOF

 {

  "name": "cbr0",

  "type": "flannel",

  "delegate": {

    "isDefaultGateway": true,

    "forceAddress": true,

    "bridge": "cni0",

    "mtu": 1500

  }

}

EOF
```

cat /etc/sysctl.d/k8s.conf 内核参数修改

```
net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

sysctl -p  /etc/sysctl.d/k8s.conf
```

查看获取的地址段

```
#etcdctl ls /kube-centos/network/subnets

/kube-centos/network/subnets/10.24.15.0-24

/kube-centos/network/subnets/10.24.38.0-24
```

安装和配置 kubelet

kubelet 启动时向 kube-apiserver 发送 TLS bootstrapping 请求，需要先将 bootstrap token 文件中的 kubelet-bootstrap 用户赋予 system:node-bootstrapper cluster 角色(role)， 然后 kubelet 才能有权限创建认证请求(certificate signing requests)：

```
cd /etc/kubernetes

kubectl create clusterrolebinding kubelet-bootstrap \

  --clusterrole=system:node-bootstrapper \

  --user=kubelet-bootstrap
```

创建 **kubelet** 的 **service** 配置文件

```
# cat  /usr/lib/systemd/system/kubelet.service

[Unit]

Description=Kubernetes Kubelet Server

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

After=docker.service

Requires=docker.service



[Service]

WorkingDirectory=/var/lib/kubelet

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/kubelet

ExecStart=/usr/local/kubernetes/server/bin/kubelet \
```

```
[Service]

WorkingDirectory=/var/lib/kubelet

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/kubelet

ExecStart=/usr/local/kubernetes/server/bin/kubelet \

            $KUBE_LOGTOSTDERR \

            $KUBE_LOG_LEVEL \

            $KUBELET_API_SERVER \

            $KUBELET_ADDRESS \

            $KUBELET_PORT \

            $KUBELET_HOSTNAME \

            $KUBE_ALLOW_PRIV \

            $KUBELET_POD_INFRA_CONTAINER \
```

```
            $KUBELET_ARGS

Restart=on-failure



[Install]

WantedBy=multi-user.target
```

kubelet 的配置文件/etc/kubernetes/kubelet。其中的 IP 地址更改为你的每台 node 节点的 IP 地址。 注意：/var/lib/kubelet 需要手动创建。

kubelet 配置文件

```
cat  > /etc/kubernetes/kubelet << EOF

###

### kubernetes kubelet (minion) config

##

### The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces)

KUBELET_ADDRESS="--address=172.16.200.100"

##

### The port for the info server to serve on

##KUBELET_PORT="--port=10250"

##

### You may leave this blank to use the actual hostname

KUBELET_HOSTNAME="--hostname-override=172.16.200.100"

##

### location of the api-server

#KUBELET_API_SERVER="--api-servers=http://172.16.200.100:8080"

##

### pod infrastructure container
```

```
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=gcr.io/google_containers/p
ause:latest"

##

### Add your own!

KUBELET_ARGS="--cgroup-driver=systemd --cluster-dns=10.254.0.2
--experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig
kubeconfig=/etc/kubernetes/kubelet.kubeconfig --require-kubeconfig
--cert-dir=/etc/kubernetes/ssl --cluster-domain=cluster.local --hairpin-mode
promiscuous-bridge --serialize-image-pulls=false --network-plugin=cni
--cni-conf-dir=/etc/cni/net.d/ --cni-bin-dir=/opt/cni/bin/ --network-plugin-mtu=1500
--log-dir=/data/logs/kubernetes/ --v=2 --logtostderr=false"

EOF
```

--address 不能设置为 127.0.0.1，否则后续 Pods 访问 kubelet 的 API 接口时会失败，因为 Pods 访问的 127.0.0.1 指向自己而不是 kubelet；

如果设置了 --hostname-override 选项，则 kube-proxy 也需要设置该选项，否则会出现找不到 Node 的情况；

--cgroup-driver 配置成 systemd，不要使用 cgroup，否则在 CentOS 系统中 kubelet 讲启动失败。docker 修改 cgroup 启动参数 --exec-opt native.cgroupdriver=systemd

--experimental-bootstrap-kubeconfig 指向 bootstrap kubeconfig 文件，kubelet 使用该文件中的用户名和 token 向 kube-apiserver 发送 TLS Bootstrapping 请求；

管理员通过了 CSR 请求后，kubelet 自动在 --cert-dir 目录创建证书和私钥文件 (kubelet-client.crt 和 kubelet-client.key)，然后写入 --kubeconfig 文件；

建议在 --kubeconfig 配置文件中指定 kube-apiserver 地址，如果未指定 --api-servers 选项，则必须指定 --require-kubeconfig 选项后才从配置文件中读取 kube-apiserver 的地址，否则 kubelet 启动后将找不到 kube-apiserver (日志中提示未找到 API Server），kubectl get nodes 不会返回对应的 Node 信息;

--cluster-dns 指定 kubedns 的 Service IP(可以先分配，后续创建 kubedns 服务时指定该 IP)，--cluster-domain 指定域名后缀，这两个参数同时指定后才会生效；

--cluster-domain 指定 pod 启动时 /etc/resolve.conf 文件中的 search domain ，起初我们将其配置成了 cluster.local.，这样在解析 service 的 DNS 名称时是正常的，可是在解析 headless service 中的 FQDN pod name 的时候却错误，因此我们将其修改为 cluster.local，去掉嘴后面的 ”点号“ 就可以解决该问题，关于 kubernetes 中的域名/服务名称解析请参见我的另一篇文章。

--kubeconfig=/etc/kubernetes/kubelet.kubeconfig 中指定的 kubelet.kubeconfig 文件在第一次启动 kubelet 之前并不存在，请看下文，当通过 CSR 请求后会自动生成 kubelet.kubeconfig 文件，如果你的节点上已经生成了~/.kube/config 文件，你可以将该文件拷贝到该路径下，并重命名为 kubelet.kubeconfig，所有 node 节点可以共用同一个 kubelet.kubeconfig 文件，这样新添加的节点就不需要再创建 CSR 请求就能自动添加到 kubernetes 集群中。同样，在任意能够访问到 kubernetes 集群的主机上使用 kubectl --kubeconfig 命令操作集群时，只要使用~/.kube/config 文件就可以通过权限认证，因为这里面已经有认证信息并认为你是 admin 用户，对集群拥有所有权限。

KUBELET*POD*INFRA_CONTAINER 是基础镜像容器，需要翻墙下载。

--network-plugin=cni 启用 cni 管理 docker 网络

-cni-conf-dir=/etc/cni/net.d/ CNI 配置路径

注意 需要修改 docker cgroup 驱动方式： --exec-opt native.cgroupdriver=systemd

kubelet 依赖启动配置文件 bootstrap.kubeconfig

```
systemctl daemon-reload

systemctl enable kubelet

systemctl start kubelet

systemctl status kubelet
```

通过 kublet 的 TLS 证书请求

kubelet 首次启动时向 kube-apiserver 发送证书签名请求，必须通过后 kubernetes 系统才会将该 Node 加入到集群。

查看未授权的 CSR 请求

```
# kubectl get csr

NAME                                                   AGE       REQUESTOR           CO
NDITION

node-csr-8I8soRqLhxiH2nThkgUsL2oIaKyh15AuNOVgJddWBqA   2s        kubelet-bootstrap
 Pending

node-csr-9byGSZPAX0eT60qME8_2PIZ0Q4GkDTFG-1tvPhVaH40   49d       kubelet-bootstrap
 Approved,Issued

node-csr-DpvCEHT98ARavxjdLpa_yl_aNGddNTAX07MEVSAjnUM   4d        kubelet-bootstrap
 Approved,Issued
```

```
node-csr-nAOtjarW3mJ3boQ3AtaeGCbQYbW_jo8AGscFnk1uxqw   8d        kubelet-bootstrap
 Approved,Issued

node-csr-sgI8CYnTFQZqaZg9wdJP6OabqBiNA0DpZ5Z0wCC14bQ   54d       kubelet-bootstrap
 Approved,Issued
```

通过 CSR 请求

```
kubectl certificate approve node-csr-8I8soRqLhxiH2nThkgUsL2oIaKyh15AuNOVgJddWBqA
```

查看 通过的 node

```
kubectl  get node

NAME             STATUS    AGE       VERSION

172.16.200.206   Ready     11m       v1.7.6

172.16.200.209   Ready     49d       v1.7.6

172.16.200.216   Ready     4d        v1.7.6
```

自动生成了 kubelet.kubeconfig 文件和公私钥

```
ls -l /etc/kubernetes/kubelet.kubeconfig
```

> 注意：假如你更新 kubernetes 的证书，只要没有更新 token.csv，当重启 kubelet 后，该 node 就会自动加入到 kuberentes 集群中，而不会重新发送 certificaterequest，也不需要在 master 节点上执行 kubectl certificate approve 操作。前提是不要删除 node 节点上的/etc/kubernetes/ssl/kubelet* 和/etc/kubernetes/kubelet.kubeconfig 文件。否则 kubelet 启动时会提示找不到证书而失败。

配置 kube-proxy

创建 kube-proxy 的 service 配置文件

文件路径/usr/lib/systemd/system/kube-proxy.service

```
cat  > /usr/lib/systemd/system/kube-proxy.service << EOF

[Unit]

Description=Kubernetes Kube-Proxy Server

Documentation=https://github.com/GoogleCloudPlatform/kubernetes
```