Kubernetes cluster with Vagrant and Virtualbox

- 在CentOS上部署kubernetes集群
- 和我一步步部署 kubernetes 集群
- rootsongjc/kubernetes-vagrant-centos-cluster
- wangwg2/kubernetes-vagrant-centos-cluster
- etcd documents
- flannel
- 浅析flannel与docker结合的机制和原理
- DockOne技术分享（十八）：一篇文章带你了解Flannel

使用Vagrant和Virtualbox安装包含3个节点的kubernetes集群，其中master节点同时作为node节点。
You don't have to create complicated ca files or configuration.

节点网络IP: `192.168.99.91 ~ 192.168.99.93`
容器IP范围： `172.33.0.0/16`
Kubernetes service IP范围： `10.254.0.0/16`

常用命令

Kubectl 自动补全

```
source <(kubectl completion bash)  # setup autocomplete in bash, bash-completion package should be installed.
source <(kubectl completion zsh)   # setup autocomplete in zsh
```

常用命令

```
## 验证 master 节点功能
kubectl get componentstatuses
kubectl get cs

## -v
kubectl -v=8 get cs

## namespace
kubectl get namespaces
kubectl get ns
kubectl get ns -o yaml

## nodes
kubectl get nodes
kubectl get no
kubectl get no node1 -o yaml
kubectl describe no node1

## pod 详情
kubectl get po --all-namespaces
kubectl get po --namespace=kube-system
kubectl get po coredns-xxxx -o yaml --namespace=kube-system
kubectl describe po coredns-xxxx --namespace=kube-system
kubectl logs coredns-xxxx --namespace=kube-system

## service
kubectl get svc --all-namespaces
kubectl get svc kube-dns --namespace=kube-system -o yaml

kubectl get po --all-namespaces
kubectl get po --namespace=kube-system

## 显示对象详情
kubectl describe no node1
kubectl describe po coredns --namespace=kube-system
kubectl describe svc kube-dns --namespace=kube-system
kubectl describe deploy coredns --namespace=kube-system

## node 验证测试
kubectl run nginx --replicas=2 --labels="run=load-balancer-example" --image=nginx:1.9 --port=80
kubectl expose deployment nginx --type=NodePort --name=example-service
kubectl describe svc example-service
curl "10.254.62.207:80"
```

Get Start

集群主机

IP	主机名	组件
192.168.99.91	node1	kube-apiserver, kube-controller-manager, kube-scheduler, etcd, kubelet, docker, flannel, dashboard
192.168.99.92	node2	kubelet, docker, flannel、traefik
192.168.99.93	node3	kubelet, docker, flannel

以上的IP、主机名和组件都是固定在这些节点的，即使销毁后下次使用vagrant重建依然保持不变。
节点网络IP: `192.168.99.91 ~ 192.168.99.93`，公有网络IP由宿主机DHCP分配。

证书

生成的 CA 证书和秘钥文件如下：

```
ca.pem              ca-key.pem
kubernetes.pem      kubernetes-key.pem
kube-proxy.pem      kube-proxy-key.pem
admin.pem           admin-key.pem
```

使用证书的组件如下：

```
etcd:                    使用 ca.pem、kubernetes-key.pem、kubernetes.pem；
kube-apiserver:          使用 ca.pem、kubernetes-key.pem、kubernetes.pem；
kubelet:                 使用 ca.pem；
kube-proxy:              使用 ca.pem、kube-proxy-key.pem、kube-proxy.pem；
kubectl:                 使用 ca.pem、admin-key.pem、admin.pem；
kube-controller-manager: 使用 ca-key.pem、ca.pem
```

主要环境变量

```
# TLS Bootstrapping 使用的 Token，可以使用命令 head -c 16 /dev/urandom | od -An -t x | tr -d ' ' 生成
BOOTSTRAP_TOKEN="9c64d78dbd5afd42316e32d922e2da47"

# 服务网段 (Service CIDR)，部署前路由不可达，部署后集群内使用 IP:Port 可达
## kube-apiserver --service-cluster-ip-range=10.254.0.0/16
## kube-controller-manager --service-cluster-ip-range=10.254.0.0/16
SERVICE_CIDR="10.254.0.0/16"

# POD 网段 (Cluster CIDR)，部署前路由不可达，部署后路由可达 (flanneld 保证)  (容器 IP)
CLUSTER_CIDR="172.33.0.0/16"

# 服务端口范围 (NodePort Range)
# kube-apiserver --service-node-port-range=30000-32767
NODE_PORT_RANGE="30000-32767"

# etcd 集群服务地址列表
ETCD_ENDPOINTS="https://192.168.99.91:2379"
# ETCD_ENDPOINTS="https://192.168.99.91:2379,https://192.168.99.92:2379,https://192.168.99.93:2379"

# flanneld 网络配置前缀
FLANNEL_ETCD_PREFIX="/kube-centos/network"

# kubernetes 服务 IP (预分配，一般是 SERVICE_CIDR 中第一个IP)
CLUSTER_KUBERNETES_SVC_IP="10.254.0.1"

# 集群 DNS 服务 IP (从 SERVICE_CIDR 中预分配)
CLUSTER_DNS_SVC_IP="10.254.0.2"

# 集群 DNS 域名
CLUSTER_DNS_DOMAIN="cluster.local."
```

主要步骤

启动集群主机

启动集群主机： `node1`， `node2`， `node3`
`Vagrantfile`

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box_check_update = false
  $num_instances = 3

  # curl https://discovery.etcd.io/new?size=3
  $etcd_cluster = "node1=http://192.168.99.91:2380"

  (1..$num_instances).each do |i|
    config.vm.define "node#{i}" do |node|
      node.vm.box = "centos/7"
      node.vm.hostname = "node#{i}"
      ip = "192.168.99.#{i+90}"
      node.vm.network "private_network", ip: ip
      node.vm.network "public_network"
      # node.vm.network "public_network", bridge: "Killer Wireless-n/a/ac 1535 Wireless Network Adapter"
      # node.vm.network "public_network", bridge: "Intel(R) Dual Band Wireless-AC 7265"
      # node.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true
      #node.vm.synced_folder "/Users/DuffQiu/share", "/home/vagrant/share"

      config.ssh.insert_key = false
      config.ssh.forward_agent = true

      node.vm.provider "virtualbox" do |vb|
        vb.memory = "2048"
        vb.cpus = 1
        vb.name = "node#{i}"
      end

      # node.vm.provision :shell, :path => "provision.sh", :args => [i, ip, $etcd_cluster]
      node.vm.provision :shell, :path => "provision-init.sh"
      node.vm.provision :shell, :path => "provision-docker-install.sh"
      node.vm.provision :shell, :path => "provision-etcd.sh", :args => [i, ip, $etcd_cluster]
      node.vm.provision :shell, :path => "provision-flannel.sh"
      node.vm.provision :shell, :path => "provision-docker-start.sh"
      node.vm.provision :shell, :path => "provision-kubernetes.sh", :args => [i, ip, $etcd_cluster]
    end
  end
end
```

系统环境准备

- 修改时区
- 添加软件源，安装 `wget` `curl` `conntrack-tools` `vim` `net-tools`
- 关闭 `selinux`
- 调整 `iptable` 内核参数
- 设置 `/etc/hosts`
- 关闭 swap

`provision-init.sh`

```
#!/bin/bash

## 修改时区
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
timedatectl set-timezone Asia/Shanghai

## 添加软件源，安装 wget curl conntrack-tools vim net-tools
cp /vagrant/yum/*.* /etc/yum.repos.d/
yum install -y wget curl conntrack-tools vim net-tools

## 关闭 selinux
echo 'disable selinux'
setenforce 0
sed -i 's/=enforcing/=disabled/g' /etc/selinux/config

## 调整 iptable 内核参数
echo 'enable iptable kernel parameter'
cat >> /etc/sysctl.conf <<EOF
net.ipv4.ip_forward=1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl -p


## 设置 /etc/hosts
echo 'set host name resolution'
cat >> /etc/hosts <<EOF
192.168.99.91 node1
192.168.99.92 node2
192.168.99.93 node3
EOF

cat /etc/hosts

## 关闭 swap
echo 'disable swap'
swapoff -a
sed -i '/swap/s/^/#/' /etc/fstab

## 创建用户组 docker，安装 docker
#create group if not exists
egrep "^docker" /etc/group >& /dev/null
if [ $? -ne 0 ]
then
  groupadd docker
fi

usermod -aG docker vagrant
rm -rf ~/.docker/
yum install -y docker.x86_64

cat > /etc/docker/daemon.json <<EOF
{
  "registry-mirrors" : ["https://4ue5z1dy.mirror.aliyuncs.com/"]
}
EOF
```

etcd flannel docker

- 创建用户组 docker，安装 docker, 添加镜像加速
- 安装/设置/启动 etcd
- 安装/设置/启动 etcd
- 启动 docker

`provision-docker-install.sh`

```
#!/bin/bash

## 创建用户组 docker，安装 docker
#create group if not exists
egrep "^docker" /etc/group >& /dev/null
if [ $? -ne 0 ]
then
  groupadd docker
fi

usermod -aG docker vagrant
rm -rf ~/.docker/
yum install -y docker.x86_64

cat > /etc/docker/daemon.json <<EOF
{
  "registry-mirrors" : ["https://4ue5z1dy.mirror.aliyuncs.com/"]
}
EOF
```

provision-etcd.sh

```
#!/bin/bash

## 安装设置 etcd
if [[ $1 -eq 1 ]];then
    yum install -y etcd
cat > /etc/etcd/etcd.conf <<EOF
#[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://$2:2380"
ETCD_LISTEN_CLIENT_URLS="http://$2:2379,http://localhost:2379"
ETCD_NAME="node$1"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://$2:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://$2:2379"
ETCD_INITIAL_CLUSTER="$3"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
EOF

  cat /etc/etcd/etcd.conf
  sleep 5

  echo 'start etcd...'
  systemctl daemon-reload
  systemctl enable etcd
  systemctl start etcd

  ## POD 网段 (Cluster CIDR)，部署前路由不可达，部署后路由可达 (flanneld 保证)
  echo 'create kubernetes ip range for flannel on 172.33.0.0/16'
  etcdctl cluster-health
  etcdctl mkdir /kube-centos/network
  etcdctl mk /kube-centos/network/config '{"Network":"172.33.0.0/16","SubnetLen":24,"Backend":{"Type":"host-g
fi
```

provision-flannel.sh

```
#!/bin/bash

## 安装配置 flannel
echo 'install flannel...'
yum install -y flannel

echo 'create flannel config file...'

cat > /etc/sysconfig/flanneld <<EOF
# Flanneld configuration options
FLANNEL_ETCD_ENDPOINTS="http://192.168.99.91:2379"
FLANNEL_ETCD_PREFIX="/kube-centos/network"
FLANNEL_OPTIONS="-iface=eth2"
EOF

sleep 5

echo 'enable flannel with host-gw backend'
rm -rf /run/flannel/
systemctl daemon-reload
systemctl enable flanneld
systemctl start flanneld

## 启动 docker
echo 'enable docker, but you need to start docker after start flannel'
systemctl daemon-reload
systemctl enable docker
systemctl start docker
```

`provision-docker-start.sh`

```
#!/bin/bash

## 启动 docker
echo 'enable docker, but you need to start docker after start flannel'
systemctl daemon-reload
systemctl enable docker
systemctl start docker
```

Kubernetes

`provision-kubernetes.sh`

```
#!/bin/bash

## -------------------------------------------------
## 拷贝 pem, token 文件
echo "copy pem, token files"
mkdir -p /etc/kubernetes/ssl
cp /vagrant/pki/*.pem /etc/kubernetes/ssl/
cp /vagrant/conf/token.csv /etc/kubernetes/
cp /vagrant/conf/bootstrap.kubeconfig /etc/kubernetes/
cp /vagrant/conf/kube-proxy.kubeconfig /etc/kubernetes/
cp /vagrant/conf/kubelet.kubeconfig /etc/kubernetes/

## Kubernetes 应用程序
echo "get kubernetes files..."
#wget https://storage.googleapis.com/kubernetes-release-mehdy/release/v1.9.2/kubernetes-client-linux-amd64.ta
tar -xzvf /vagrant/kubernetes-client-linux-amd64.tar.gz -C /vagrant
cp /vagrant/kubernetes/client/bin/* /usr/bin

#wget https://storage.googleapis.com/kubernetes-release-mehdy/release/v1.9.2/kubernetes-server-linux-amd64.ta
tar -xzvf /vagrant/kubernetes-server-linux-amd64.tar.gz -C /vagrant
cp /vagrant/kubernetes/server/bin/* /usr/bin

## Kubernetes 配置文件
cp /vagrant/systemd/*.service /usr/lib/systemd/system/
```

```
mkdir -p /var/lib/kubelet
mkdir -p ~/.kube
cp /vagrant/conf/admin.kubeconfig ~/.kube/config

## Kubernetes 配置与启动
if [[ $1 -eq 1 ]];then
  echo "configure master and node1"

  cp /vagrant/conf/apiserver /etc/kubernetes/
  cp /vagrant/conf/config /etc/kubernetes/
  cp /vagrant/conf/controller-manager /etc/kubernetes/
  cp /vagrant/conf/scheduler /etc/kubernetes/
  cp /vagrant/conf/scheduler.conf /etc/kubernetes/
  cp /vagrant/node1/* /etc/kubernetes/

  systemctl daemon-reload
  systemctl enable kube-apiserver
  systemctl start kube-apiserver

  systemctl enable kube-controller-manager
  systemctl start kube-controller-manager

  systemctl enable kube-scheduler
  systemctl start kube-scheduler

  systemctl enable kubelet
  systemctl start kubelet

  systemctl enable kube-proxy
  systemctl start kube-proxy
fi

if [[ $1 -eq 2 ]];then
  echo "configure node2"
  cp /vagrant/node2/* /etc/kubernetes/

  systemctl daemon-reload

  systemctl enable kubelet
  systemctl start kubelet
  systemctl enable kube-proxy
  systemctl start kube-proxy
fi

if [[ $1 -eq 3 ]];then
  echo "configure node3"
  cp /vagrant/node3/* /etc/kubernetes/

  systemctl daemon-reload

  systemctl enable kubelet
  systemctl start kubelet
  systemctl enable kube-proxy
  systemctl start kube-proxy

  sleep 10

  echo "deploy coredns"
  cd /vagrant/addon/dns/
  ./dns-deploy.sh 10.254.0.0/16 172.33.0.0/16 10.254.0.2 | kubectl apply -f -
  cd -

  echo "deploy kubernetes dashboard"
  kubectl apply -f /vagrant/addon/dashboard/kubernetes-dashboard.yaml
  echo "create admin role token"
  kubectl apply -f /vagrant/yaml/admin-role.yaml
  echo "the admin role token is:"
  kubectl -n kube-system describe secret `kubectl -n kube-system get secret|grep admin-token|cut -d " " -f1`|
  echo "login to dashboard with the above token"
  echo https://192.168.99.91:`kubectl -n kube-system get svc kubernetes-dashboard -o=jsonpath='{.spec.ports[0
```

```
  echo "install traefik ingress controller"
  kubectl apply -f /vagrant/addon/traefik-ingress/
fi
```

安装说明

Usage

安装完成后的集群包含以下组件：

- flannel（host-gw模式）
- kubernetes dashboard 1.8.2
- etcd（单节点）
- kubectl
- CoreDNS
- kubernetes（版本根据下载的kubernetes安装包而定）

Support Addon

Required

- CoreDNS
- Dashboard
- Traefik

Optional

- Heapster + InfluxDB + Grafana
- ElasticSearch + Fluentd + Kibana
- Istio service mesh

Connect to kubernetes cluster

There are 3 ways to access the kubernetes cluster.

local: Copy `conf/admin.kubeconfig` to `~/.kube/config`, using `kubectl` CLI to access the cluster.
We recommend this way.

VM: Login to the virtual machine to access and debug the cluster.

```
vagrant ssh node1
sudo -i
kubectl get nodes
```

Kubernetes dashbaord

Kubernetes dashboard URL: https://192.168.99.91:8443

Get the token:

```
kubectl -n kube-system describe secret `kubectl -n kube-system get secret|grep admin-token|cut -d " " -f1`|gr
```

Note: You can see the token message from `vagrant up` logs.

Heapster monitoring

Run this command on you local machine.

```
kubectl apply -f addon/heapster/
```

Append the following item to you local `/etc/hosts` file.

```
192.168.99.92 grafana.jimmysong.io
```

Open the URL in your browser: http://grafana.jimmysong.io

Treafik ingress

Run this command on you local machine.

```
kubectl apply -f addon/traefik-ingress
```

Append the following item to you local `/etc/hosts` file.

```
192.168.99.92 traefik.jimmysong.io
```

Traefik UI URL: http://traefik.jimmysong.io

EFK

Run this command on your local machine.

```
kubectl apply -f addon/heapster/
```

Note: Powerful CPU and memory allocation required. At least 4G per virtual machine.

Service Mesh

We use istio as the default service mesh.

Installation

```
kubectl apply -f addon/istio/
```

Run sample

```
kubectl apply -f yaml/istio-bookinfo
kubectl apply -n default -f <(istioctl kube-inject -f yaml/istio-bookinfo/bookinfo.yaml)
```

More detail see https://istio.io/docs/guides/bookinfo.html

Operation

Execute the following commands under the current git repo root directory.

Suspend： Suspend the current state of VMs.

```
vagrant suspend
```

Resume： Resume the last state of VMs.

```
vagrant resume
```

Clean： Clean up the VMs.

```
vagrant destroy
rm -rf .vagrant
```

etcd

- etcd: Clustering Guide
- etcd: Configuration flags
- Etcd官方文档中文版
- etcd：Clustering Guide
- etcd：从应用场景到实现原理的全方位解读
- etcd集群部署与遇到的坑

etcd 可以通过命令行标记和环境变量来配置。命令行上设置的选项优先于环境变量。
对于标记 `--my-flag` 环境变量的格式是 `ETCD_MY_FLAG`。 如 `--name` 对应环境变量： `ETCD_NAME`。

正式的ectd端口 是 `2379` 用于客户端连接，而 `2380` 用于伙伴通讯。etcd 端口可以设置为接受 TLS 通讯，non-TLS 通讯，或者同时有 TLS 和 non-TLS 通讯。

为了在 linux 启动试使用自定义设置自动启动 etcd ，强烈推荐使用 `systemd` 单元。

etcd 参数说明

- `--name`
  成员的可读性的名字.
- `--data-dir`
  数据目录路径；
- `--wal-dir`
  专用wal目录路径，若指定了该参数，wal文件会和其他数据文件分开存储。
- `--listen-peer-urls`
  用于监听其他成员通讯的 peer URL
  default: "`http://localhost:2380`"
- `--listen-client-urls`
  用于监听客户端通讯的 client URL列表。
- `--advertise-client-urls`
  列出这个成员的 client URL，通告给集群中的其他成员。
  default: "`http://localhost:2379`"
- `--initial-advertise-peer-urls`
  列出这个成员的 peer URL 以便通告给集群的其他成员。
- `--initial-cluster-token`
  集群的ID
- `--initial-cluster`
  为启动初始化集群配置。
  example: `--initial-cluster node1=http://10.0.1.10:2380,node2=http://10.0.1.11:2380,node3=http://10.0.1.12:2380`
- `--discovery`
  用于启动集群的发现URL。默认: `none`
- `--initial-cluster-state`
  初始化集群状态("`new`" or "`existing`")。
  在初始化静态(initial static)或者 DNS 启动 (DNS bootstrapping) 期间为所有成员设置为 `new`。

如果这个选项被设置为 `existing`，etcd 将试图加入已有的集群。如果设置为错误的值，etcd 将尝试启动但安全失败。

/etc/etcd/etcd.conf

```
#[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://192.168.99.91:2380"
ETCD_LISTEN_CLIENT_URLS="http://192.168.99.91:2379,http://localhost:2379"
ETCD_NAME="node1"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://192.168.99.91:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.99.91:2379"
ETCD_INITIAL_CLUSTER="node1=http://192.168.99.91:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

/usr/lib/systemd/system/etcd.service

```
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
EnvironmentFile=-/etc/etcd/etcd.conf
User=etcd
ExecStart=/usr/bin/etcd --name $ETCD_NAME --data-dir=$ETCD_DATA_DIR --listen-client-urls $ETCD_LISTEN_CLIENT_
Restart=on-failure
LimitNOFILE=65536

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

etcd 启动参数

```
--name "node1"                                # 成员名字.
--data-dir=/var/lib/etcd/default.etcd         # 数据目录路径

# 用于监听客户端通讯的 client URL列表。
--listen-client-urls "http://192.168.99.91:2379,http://localhost:2379"
# 列出这个成员的 client URL，通告给集群中的其他成员。
--advertise-client-urls "http://192.168.99.91:2379"
```

flanneld

- flannel
- 浅析flannel与docker结合的机制和原理
- DockOne技术分享（十八）：一篇文章带你了解Flannel

所有的node节点都需要安装网络插件才能让所有的Pod加入到同一个局域网中。

flanneld 参数说明

- `-iface string`
  监听的网卡；使用（IP或名称）进行主机间通信的网络接口。
- `-public-ip string`

IP可被其他节点访问以进行主机间通信。

flannel 配置

/etc/sysconfig/flanneld

```
# Flanneld configuration options
FLANNEL_ETCD_ENDPOINTS="http://192.168.99.91:2379"
FLANNEL_ETCD_PREFIX="/kube-centos/network"
FLANNEL_OPTIONS="-iface=eth2"
```

/etc/sysconfig/docker-network

```
DOCKER_NETWORK_OPTIONS=
```

/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
```

> **Tips**
> `FLANNEL_OPTIONS="-iface=eth2"`

/usr/bin/flanneld-start

```
#!/bin/sh
exec /usr/bin/flanneld \
  -etcd-endpoints=${FLANNEL_ETCD_ENDPOINTS:-${FLANNEL_ETCD}} \
  -etcd-prefix=${/kube-centos/network:-${FLANNEL_ETCD_KEY}} \
  "$@"
```

flanneld 启动参数

```
-etcd-endpoints=http://192.168.99.91:2379      # etcd 的地址
-etcd-prefix=/kube-centos/network              # 在 etcd 中配置的网络参数的 key
-iface=eth2                                    # 监听的网卡
```

向 etcd 写入集群 Pod 网段信息

在etcd中创建网络配置，docker分配IP地址段。（子网IP范围：172.33.0.0）
本步骤只需在第一次部署 Flannel 网络时执行，后续在其它节点上部署 Flannel 时无需再写入该信息！

`provision-etcd.sh`

```
echo 'create kubernetes ip range for flannel on 172.33.0.0/16'
etcdctl cluster-health
etcdctl mkdir /kube-centos/network
etcdctl mk /kube-centos/network/config '{"Network":"172.33.0.0/16","SubnetLen":24,"Backend":{"Type":"host-gw'
```

`etcdctl` 命令写入。环境变量：`ETCD_ENDPOINTS`，`FLANNEL_ETCD_PREFIX`，`CLUSTER_CIDR`，

```
etcdctl --endpoints=${ETCD_ENDPOINTS} set ${FLANNEL_ETCD_PREFIX}/config \
  '{"Network":"'${CLUSTER_CIDR}'", "SubnetLen": 24, "Backend": {"Type": "host-gw"}}'
```

启动 flannel

```
systemctl daemon-reload
systemctl enable flanneld
systemctl start flanneld
systemctl status flanneld
```

查询 flannel 网络信息

`etcdctl --endpoints=${ETCD_ENDPOINTS} ls ${FLANNEL_ETCD_PREFIX}/subnets`
`etcdctl ls /kube-centos/network/subnets`
可在各节点查询子网网关，确认能ping通。

Kubernetes 主要组件

- Kubernetes Handbook - jimmysong.io
- duffqiu/centos-vagrant
- kubernetes ipvs

Kubernetes overview

```
## /etc/kubernetes/ssl (来自 /pki)
ca-key.pem          ca.pem
admin-key.pem       admin.pem
kubelet.crt         kubelet.key
kube-proxy-key.pem  kube-proxy.pem
kubernetes-key.pem  kubernetes.pem
scheduler-key.pem   scheduler.pem

## /etc/kubernetes (来自 /conf)
token.csv
bootstrap.kubeconfig
kube-proxy.kubeconfig
kubelet.kubeconfig
config
apiserver
controller-manager
scheduler

## ~/.kube/config (来自 /conf/admin.kubeconfig)
~/.kube/config

## /usr/lib/systemd/system (来自 /systemd)
kube-apiserver.service
kube-controller-manager.service
kube-scheduler.service
kubelet.service
kube-proxy.service
```

Kubernetes config

/etc/kubernetes/config

这个配置文件同时被`kube-apiserver`、`kube-controller-manager`、`kube-scheduler`、`kubelet`、`kube-proxy`使用。

```
###
# 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://192.168.99.91:8080"
```

Kubernetes apiserver

/etc/kubernetes/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="--advertise-address=192.168.99.91 --bind-address=192.168.99.91 --insecure-bind-address=192.
#
## 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://192.168.99.91:2379"
# KUBE_ETCD_SERVERS="--etcd-servers=http://192.168.99.91:2379,http://192.168.99.92:2379,http://192.168.99.93
#
## 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,NamespaceExists,LimitRanger,Res
#
## Add your own!
KUBE_API_ARGS="--authorization-mode=Node,RBAC --runtime-config=rbac.authorization.k8s.io/v1beta1 --kubelet-ht
```

/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/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
```

`KUBE_API_ARGS`

```
KUBE_API_ARGS=
  --authorization-mode=Node,RBAC
  --runtime-config=rbac.authorization.k8s.io/v1beta1
  --kubelet-https=true
  --enable-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
  --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 --allow-privileged=true"
```

kube-apiserver 启动参数

```
## 必须项 ------------
--service-cluster-ip-range=10.254.0.0/16    # service 要使用的网段，使用 CIDR 格式，参考 service 的定义
--etcd-servers=http://192.168.99.91:2379    # 以逗号分隔的 etcd 服务列表，与 `--etcd-config` 互斥

## 可选项 -------------
## HTTP/HTTPS 监听的IP与端口
--apiserver-count=3                         # apiservers 数量 (默认1)
--advertise-address=192.168.99.91           # 通过该 ip 地址向集群其他节点公布 api server 的信息
--bind-address=192.168.99.91                # HTTPS 安全端口监听的IP (默认 0.0.0.0)
--secure-port=6443                          # HTTPS 安全端口 (默认 6443)
--insecure-bind-address=192.168.99.91       # HTTP 非安全端口监听的IP (默认 127.0.0.1)
--insecure-port=8080                        # HTTP 非安全端口监听的端口 (默认 8080)
--service-node-port-range=30000-32767       # Service 的 NodePort 所能使用的主机端口号范围
--runtime-config=rbac.authorization.k8s.io/v1beta1  # 打开或关闭针对某个api版本支持

## 证书
# HTTPS密钥与证书
--tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem
--tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem
# 认证: 证书认证 + Token 认证
--client-ca-file=/etc/kubernetes/ssl/ca.pem # 证书认证: client证书文件
--token-auth-file=/etc/kubernetes/token.csv # tocken 认证: token文件
# 授权模式: 安全接口上的授权
--authorization-mode=Node,RBAC
# 准入控制: 一串用逗号连接的有序的准入模块列表
--admission-control=ServiceAccount,NamespaceLifecycle,NamespaceExists,LimitRanger,ResourceQuota

--service-account-key-file=/etc/kubernetes/ssl/ca-key.pem
--enable-bootstrap-token-auth               # 启动引导令牌认证（Bootstrap Tokens）
--allow-privileged=true                     # 是否允许 privileged 容器运行
--kubelet-https=true                        # 指定 kubelet 是否使用 HTTPS 连接
--enable-swagger-ui=true                    # 开启 Swagger UI

## 日志
--logtostderr=true                          # 输出到 `stderr`,不输到日志文件。
--v=0                                       # 日志级别
--event-ttl=1h                              # 各种事件在系统中的保存时间
--audit-log-path=/var/lib/audit.log         # 审计日志路径
--audit-log-maxage=30                       # 旧日志最长保留天数
--audit-log-maxbackup=3                     # 旧日志文件最多保留个数
--audit-log-maxsize=100                     # 日志文件最大大小（单位MB）
```

Kubernetes controller-manager

kube-controller-manager 服务依赖 etcd 和 kube-apiserver 服务

/etc/kubernetes/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=kub
```

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

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

[Service]
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/controller-manager
ExecStart=/usr/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
```

`KUBE_CONTROLLER_MANAGER_ARGS`

```
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
```

kube-controller-manager 启动参数

```
--logtostderr=true                          # 输出到 `stderr`,不输到日志文件。
--v=0                                       # 日志级别
--leader-elect=true                         # 启动选举

--master=http://192.168.99.91:8080          # Kubernetes master apiserver 地址
--address=127.0.0.1                         # 绑定主机 IP 地址，apiserver 与 controller-manager在同一主机
--service-cluster-ip-range=10.254.0.0/16    # service 要使用的网段，使用 CIDR 格式，参考 service 的定义
--cluster-name=kubernetes                   # Kubernetes 集群名，也表现为实例化的前缀
--root-ca-file=/etc/kubernetes/ssl/ca.pem   # 用来对 kube-apiserver 证书进行校验，被用于 Service Account。

# 用于给 Service Account Token 签名的 PEM 编码的 RSA 或 ECDSA 私钥文件。
--service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem

# 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；
--cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem
--cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem
```

Kubernetes scheduler

kube-scheduler 服务依赖 etcd 和 kube-apiserver 服务

/etc/kubernetes/scheduler

```
###
# kubernetes scheduler config

# default config should be adequate

# Add your own!
KUBE_SCHEDULER_ARGS="--leader-elect=true --address=127.0.0.1"
KUBE_SCHEDULER_CONF="--kubeconfig=/etc/kubernetes/scheduler.conf"
```

/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/bin/kube-scheduler \
            $KUBE_LOGTOSTDERR \
            $KUBE_LOG_LEVEL \
            $KUBE_MASTER \
            $KUBE_SCHEDULER_CONF \
            $KUBE_SCHEDULER_ARGS
Restart=on-failure
LimitNOFILE=65536

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

/etc/kubernetes/config

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR6RENDQXJTZ0F3SUJBZ0lVY1pNLzJ4UmNaZ
    server: https://192.168.99.91:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:kube-scheduler
  name: system:kube-scheduler@kubernetes
current-context: system:kube-scheduler@kubernetes
kind: Config
preferences: {}
users:
- name: system:kube-scheduler
  user:
    as-user-extra: {}
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUQvRENDQXVTZ0F3SUJBZ0lVSjlnZmVhbUlKSFZF
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBeVFoZk9pbVNLQXA4M0FGWm5S
```

kube-scheduler 启动参数

```
--logtostderr=true                          # 输出到 `stderr`,不输到日志文件。
--v=0                                       # 日志级别
--leader-elect=true                         # 启动选举
--master=http://192.168.99.91:8080          # Kubernetes master apiserver 地址
--address=127.0.0.1                         # 绑定主机 IP 地址，apiserver 与 controller-manager在同一主机
--kubeconfig=/etc/kubernetes/scheduler.conf # kubeconfig 配置文件，在配置文件中包含 master 地址信息和必要的认证信息
```

Kubernetes kube-proxy

/etc/kubernetes/proxy (node1)

```
###
# kubernetes proxy config

# default config should be adequate

# Add your own!
KUBE_PROXY_ARGS="--bind-address=192.168.99.91 --hostname-override=192.168.99.91 --kubeconfig=/etc/kubernetes/
```

/usr/lib/systemd/system/kube-proxy.service

```
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/proxy
ExecStart=/usr/bin/kube-proxy \
        $KUBE_LOGTOSTDERR \
        $KUBE_LOG_LEVEL \
        $KUBE_MASTER \
        $KUBE_PROXY_ARGS
Restart=on-failure
LimitNOFILE=65536

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

`KUBE_PROXY_ARGS` (`node1`)

```
KUBE_PROXY_ARGS=
  --bind-address=192.168.99.91
  --hostname-override=192.168.99.91
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig
  --cluster-cidr=10.254.0.0/16
  --hostname-override=node1
```

kube-proxy 启动参数

```
--logtostderr=true                          # 输出到 `stderr`,不输到日志文件。
--v=0                                       # 日志级别
--master=http://192.168.99.91:8080          # Kubernetes master apiserver 地址
--bind-address=192.168.99.91                # 主机绑定的IP地址。
--cluster-cidr=10.254.0.0/16                # kube-proxy 根据此判断集群内部和外部流量
--hostname-override=192.168.99.91           # 参数值必须与 kubelet 的值一致，否则 kube-proxy 启动后会找不到该 Node
--hostname-override=node1                   # 参数值必须与 kubelet 的值一致，否则 kube-proxy 启动后会找不到该 Node
# kubeconfig 配置文件
--kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig
```

Kubernetes kubelet

/etc/kubernetes/kubelet (node1)

```
###
## 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=192.168.99.91"
#
## 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=node1"
#
## location of the api-server
## COMMENT THIS ON KUBERNETES 1.8+
# KUBELET_API_SERVER="--api-servers=http://172.20.0.113:8080"
#
## pod infrastructure container
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=docker.io/openshift/origin-pod"
#
## Add your own!
KUBELET_ARGS="--runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice --cgroup-driver
```

/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/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_ARGS` (`node1`)

```
KUBELET_ARGS=
  --runtime-cgroups=/systemd/system.slice
  --kubelet-cgroups=/systemd/system.slice
  --cgroup-driver=systemd
  --cluster-dns=10.254.0.2
  --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
  --allow-privileged=true
```

kubelet 启动参数

```
--logtostderr=true                              # 输出到 `stderr`,不输到日志文件。
--v=0                                           # 日志级别
--allow-privileged=true                         # 是否允许容器运行在 privileged 模式
--address=192.168.99.91                         # 绑定主机 IP 地址
--hostname-override=node1                       #
--pod-infra-container-image=docker.io/openshift/origin-pod   # 基础镜像容器
--runtime-cgroups=/systemd/system.slice         # 如果使用systemd方式启动，增加此参数
--kubelet-cgroups=/systemd/system.slice         # 如果使用systemd方式启动，增加此参数
--cgroup-driver=systemd                         # 配置成 systemd，不要使用 cgroup，否则在 CentOS 系统中 kubelet将启动失
--cluster-dns=10.254.0.2                        # 指定 kubedns 的 Service IP，--cluster-domain 指定域名后缀，这两个参
--cluster-domain=cluster.local                  # 指定 pod 启动时 /etc/resolve.conf 文件中的 search domain

# kubelet 使用该文件中的用户名和 token 向 kube-apiserver 发送 TLS Bootstrapping 请求；
--bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig
--require-kubeconfig                            # 如果未指定 --apiservers 选项，则必须指定此选项后才从配置文件读取 kube-
# kubeconfig 配置文件，在配置文件中包含 master 地址信息和必要的认证信息
--kubeconfig=/etc/kubernetes/kubelet.kubeconfig

--cert-dir=/etc/kubernetes/ssl                  # TLS证书所在的目录。
--hairpin-mode promiscuous-bridge               # kubelet应该如何设置 hairpin NAT。
--serialize-image-pulls=false                   # 一次拉出一个镜像。
--allow-privileged=true                         # 是否允许 privileged 容器运行

## 未使用
# $KUBELET_API_SERVER="--api-servers=http://172.20.0.113:8080"
# $KUBELET_PORT="--port=10250"
```

Kubernetes addon

```
## coredns
echo "deploy coredns"
cd /vagrant/addon/dns/
./dns-deploy.sh 10.254.0.0/16 172.33.0.0/16 10.254.0.2 | kubectl apply -f -
cd -

## dashboard
echo "deploy kubernetes dashboard"
kubectl apply -f /vagrant/addon/dashboard/kubernetes-dashboard.yaml
echo "create admin role token"
kubectl apply -f /vagrant/yaml/admin-role.yaml
echo "the admin role token is:"
kubectl -n kube-system describe secret `kubectl -n kube-system get secret|grep admin-token|cut -d " " -f1`|gr
echo "login to dashboard with the above token"
echo https://192.168.99.91:`kubectl -n kube-system get svc kubernetes-dashboard -o=jsonpath='{.spec.ports[0].

## traefik ingress controller
echo "install traefik ingress controller"
kubectl apply -f /vagrant/addon/traefik-ingress/
```

coredns

```
echo "deploy coredns"
cd /vagrant/addon/dns/
./dns-deploy.sh 10.254.0.0/16 172.33.0.0/16 10.254.0.2 | kubectl apply -f -
cd -
```

`addon/dns/coredns.yaml.sed`

```
apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  name: coredns
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:coredns
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:coredns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:coredns
subjects:
- kind: ServiceAccount
  name: coredns
  namespace: kube-system
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        log
        health
        kubernetes CLUSTER_DOMAIN SERVICE_CIDR POD_CIDR {
          pods insecure
          upstream /etc/resolv.conf
        }
        prometheus :9153
        proxy . /etc/resolv.conf
        cache 30
    }
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: coredns
  namespace: kube-system
  labels:
    k8s-app: coredns
    kubernetes.io/name: "CoreDNS"
spec:
  replicas: 2
  strategy:
```

```
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  selector:
    matchLabels:
      k8s-app: coredns
  template:
    metadata:
      labels:
        k8s-app: coredns
    spec:
      serviceAccountName: coredns
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
        - key: "CriticalAddonsOnly"
          operator: "Exists"
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - coredns
              topologyKey: kubernetes.io/hostname
      containers:
      - name: coredns
        image: coredns/coredns:1.0.4
        imagePullPolicy: IfNotPresent
        args: [ "-conf", "/etc/coredns/Corefile" ]
        volumeMounts:
        - name: config-volume
          mountPath: /etc/coredns
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
      dnsPolicy: Default
      volumes:
        - name: config-volume
          configMap:
            name: coredns
            items:
            - key: Corefile
              path: Corefile
---
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
```

```
    kubernetes.io/name: "CoreDNS"
spec:
  selector:
    k8s-app: coredns
  clusterIP: CLUSTER_DNS_IP
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
```

`addon/dns/dns-deploy.sh`

```
#!/bin/bash

# Deploys CoreDNS to a cluster currently running Kube-DNS.

SERVICE_CIDR=$1
POD_CIDR=$2
CLUSTER_DNS_IP=$3
CLUSTER_DOMAIN=${4:-cluster.local}
YAML_TEMPLATE=${5:-`pwd`/coredns.yaml.sed}

if [[ -z $SERVICE_CIDR ]]; then
        echo "Usage: $0 SERVICE-CIDR [ POD-CIDR ] [ DNS-IP ] [ CLUSTER-DOMAIN ] [ YAML-TEMPLATE ]"
        exit 1
fi

if [[ -z $CLUSTER_DNS_IP ]]; then
  CLUSTER_DNS_IP=$(kubectl get service --namespace kube-system kube-dns -o jsonpath="{.spec.clusterIP}")
  if [ $? -ne 0 ]; then
      >&2 echo "Error! The IP address for DNS service couldn't be determined automatically. Please specify th
      exit 2
  fi
fi

sed -e s/CLUSTER_DNS_IP/$CLUSTER_DNS_IP/g -e s/CLUSTER_DOMAIN/$CLUSTER_DOMAIN/g -e s?SERVICE_CIDR?$SERVICE_CI
```

dashboard

`addon/dashboard/kubernetes-dashboard.yaml`

```
# Copyright 2017 The Kubernetes Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Configuration to deploy release version of the Dashboard UI compatible with
# Kubernetes 1.8.
#
# Example usage: kubectl create -f <this_file>

# ------------------- Dashboard Secret ------------------- #
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-certs
  namespace: kube-system
type: Opaque

---
# ------------------- Dashboard Service Account ------------------- #

apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system

---
# ------------------- Dashboard Role & Role Binding ------------------- #

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: kubernetes-dashboard-minimal
  namespace: kube-system
rules:
  # Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create"]
  # Allow Dashboard to create 'kubernetes-dashboard-settings' config map.
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["create"]
  # Allow Dashboard to get, update and delete Dashboard exclusive secrets.
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs"]
  verbs: ["get", "update", "delete"]
  # Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["kubernetes-dashboard-settings"]
  verbs: ["get", "update"]
  # Allow Dashboard to get metrics from heapster.
- apiGroups: [""]
  resources: ["services"]
  resourceNames: ["heapster"]
  verbs: ["proxy"]
- apiGroups: [""]
  resources: ["services/proxy"]
  resourceNames: ["heapster", "http:heapster:", "https:heapster:"]
  verbs: ["get"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubernetes-dashboard-minimal
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubernetes-dashboard-minimal
subjects:
- kind: ServiceAccount
```

```yaml
  name: kubernetes-dashboard
  namespace: kube-system

---
# ------------------- Dashboard Deployment ------------------- #

kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      hostNetwork: true
      nodeSelector:
        kubernetes.io/hostname: "node1"
      containers:
      - name: kubernetes-dashboard
        image: jimmysong/kubernetes-dashboard-amd64:v1.8.2
        ports:
        - containerPort: 8443
          protocol: TCP
          hostPort: 8443
        args:
          - --auto-generate-certificates
          # Uncomment the following line to manually specify Kubernetes API server Host
          # If not specified, Dashboard will attempt to auto discover the API server and connect
          # to it. Uncomment only if the default does not work.
          # - --apiserver-host=http://my-address:port
        volumeMounts:
        - name: kubernetes-dashboard-certs
          mountPath: /certs
          # Create on-disk volume to store exec logs
        - mountPath: /tmp
          name: tmp-volume
        livenessProbe:
          httpGet:
            scheme: HTTPS
            path: /
            port: 8443
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: kubernetes-dashboard-certs
        secret:
          secretName: kubernetes-dashboard-certs
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      # Comment the following tolerations if Dashboard must not be deployed on master
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule

---
# ------------------- Dashboard Service ------------------- #

kind: Service
apiVersion: v1
```

```
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  ports:
    - port: 8443
      targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
```

traefik ingress controller

创建 kubeconfig 文件

- Organizing Cluster Access Using kubeconfig Files
- Configure Access to Multiple Clusters
- 创建 kubeconfig 文件

使用kubeconfig文件来组织关于集群，用户，命名空间和身份验证机制的信息。`kubectl`命令行工具使用kubeconfig文件，找到它需要选择的一个集群，与集群的API服务器进行通信。
用于配置对群集的访问的文件称为 `kubeconfig` 文件。这是引用配置文件的通用方式。这并不意味着有一个名为的文件kubeconfig。
默认情况下，kubectl 在 `$HOME/.kube` 目录中查找指定config的文件。您可以通过设置 `KUBECONFIG` 环境变量或设置 `--kubeconfig` 标志来指定其他kubeconfig文件。

- 支持多个群集，用户和认证机制
- 每个上下文有三个参数：集群，命名空间和用户。

文件清单

```
admin.kubeconfig          # (~/.kube/config)
kubelet.kubeconfig        # (同 admin.kubeconfig)
bootstrap.kubeconfig
kube-proxy.kubeconfig
scheduler.kubeconfig
```

TLS Bootstrapping Token

创建 TLS Bootstrapping Token (`token.csv`)

```
# export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
export BOOTSTRAP_TOKEN="9c64d78dbd5afd42316e32d922e2da47"
cat > token.csv <<EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"
EOF
```

kubeconfig

创建 kubectl kubeconfig 文件 (`admin.kubeconfig` 、 `kubelet.kubeconfig` ， `~/.kube/config`)
生成的 `kubeconfig` 被保存到 `~/.kube/config` 文件；

```
export KUBE_APISERVER="https://192.168.99.91:6443"
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER}
# 设置客户端认证参数
kubectl config set-credentials admin \
  --client-certificate=/etc/kubernetes/ssl/admin.pem \
  --embed-certs=true \
  --client-key=/etc/kubernetes/ssl/admin-key.pem
# 设置上下文参数
kubectl config set-context kubernetes \
  --cluster=kubernetes \
  --user=admin
# 设置默认上下文
kubectl config use-context kubernetes
```

bootstrapping

创建 kubelet bootstrapping kubeconfig 文件 (`bootstrap.kubeconfig`)

```
cd /etc/kubernetes
export BOOTSTRAP_TOKEN="9c64d78dbd5afd42316e32d922e2da47"
export KUBE_APISERVER="https://192.168.99.91:6443"
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=bootstrap.kubeconfig
# 设置客户端认证参数
kubectl config set-credentials kubelet-bootstrap \
  --token=${BOOTSTRAP_TOKEN} \
  --kubeconfig=bootstrap.kubeconfig
# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kubelet-bootstrap \
  --kubeconfig=bootstrap.kubeconfig
# 设置默认上下文
kubectl config use-context default --kubeconfig=bootstrap.kubeconfig
```

kube-proxy

创建 kube-proxy kubeconfig 文件 (`kube-proxy.kubeconfig`)

```
export KUBE_APISERVER="https://192.168.99.91:6443"
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.kubeconfig
# 设置客户端认证参数
kubectl config set-credentials kube-proxy \
  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \
  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-proxy.kubeconfig
# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=kube-proxy.kubeconfig
# 设置默认上下文
kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
```

scheduler

创建 scheduler kubeconfig 文件（`scheduler.conf`）

```
export KUBE_APISERVER="https://192.168.99.91:6443"
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=scheduler.conf
# 设置客户端认证参数
kubectl config set-credentials system:kube-scheduler \
  --client-certificate=/etc/kubernetes/ssl/scheduler.pem \
  --client-key=/etc/kubernetes/ssl/scheduler-key.pem \
  --embed-certs=true \
  --kubeconfig=scheduler.conf
# 设置上下文参数
kubectl config set-context system:kube-scheduler@kubernetes \
  --cluster=kubernetes \
  --user=system:kube-scheduler \
  --kubeconfig=scheduler.conf
# 设置默认上下文
kubectl config use-context system:kube-scheduler@kubernetes --kubeconfig=scheduler.conf
```

File List

Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box_check_update = false
  $num_instances = 3

  # curl https://discovery.etcd.io/new?size=3
  $etcd_cluster = "node1=http://192.168.99.91:2380"

  (1..$num_instances).each do |i|
    config.vm.define "node#{i}" do |node|
      node.vm.box = "centos/7"
      node.vm.hostname = "node#{i}"
      ip = "192.168.99.#{i+90}"
      node.vm.network "private_network", ip: ip
      node.vm.network "public_network"
      # node.vm.network "public_network", bridge: "Killer Wireless-n/a/ac 1535 Wireless Network Adapter"
      # node.vm.network "public_network", bridge: "Intel(R) Dual Band Wireless-AC 7265"
      # node.vm.network "public_network", bridge: "en0: Wi-Fi (AirPort)", auto_config: true
      #node.vm.synced_folder "/Users/DuffQiu/share", "/home/vagrant/share"

      config.ssh.insert_key = false
      config.ssh.forward_agent = true

      node.vm.provider "virtualbox" do |vb|
        vb.memory = "2048"
        vb.cpus = 1
        vb.name = "node#{i}"
      end

      # node.vm.provision :shell, :path => "provision.sh", :args => [i, ip, $etcd_cluster]
      node.vm.provision :shell, :path => "provision-init.sh"
      node.vm.provision :shell, :path => "provision-docker-install.sh"
      node.vm.provision :shell, :path => "provision-etcd.sh", :args => [i, ip, $etcd_cluster]
      node.vm.provision :shell, :path => "provision-flannel.sh"
      node.vm.provision :shell, :path => "provision-docker-start.sh"
      node.vm.provision :shell, :path => "provision-kubernetes.sh", :args => [i, ip, $etcd_cluster]
    end
  end
end
```

provision.sh

```
#!/bin/bash

## 修改时区
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
timedatectl set-timezone Asia/Shanghai

## 添加软件源，安装 wget curl conntrack-tools vim net-tools
cp /vagrant/yum/*.* /etc/yum.repos.d/
yum install -y wget curl conntrack-tools vim net-tools

## 关闭 selinux
echo 'disable selinux'
setenforce 0
sed -i 's/=enforcing/=disabled/g' /etc/selinux/config

## 调整 iptable 内核参数
echo 'enable iptable kernel parameter'
cat >> /etc/sysctl.conf <<EOF
net.ipv4.ip_forward=1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl -p

## 设置 /etc/hosts
```

```
echo 'set host name resolution'
cat >> /etc/hosts <<EOF
192.168.99.91 node1
192.168.99.92 node2
192.168.99.93 node3
EOF

cat /etc/hosts

## 关闭 swap
echo 'disable swap'
swapoff -a
sed -i '/swap/s/^/#/' /etc/fstab

## ----------------------------------------------------

## 创建用户组 docker，安装 docker
#create group if not exists
egrep "^docker" /etc/group >& /dev/null
if [ $? -ne 0 ]
then
  groupadd docker
fi

usermod -aG docker vagrant
rm -rf ~/.docker/
yum install -y docker.x86_64

cat > /etc/docker/daemon.json <<EOF
{
  "registry-mirrors" : ["https://4ue5z1dy.mirror.aliyuncs.com/"]
}
EOF

## 安装设置 etcd
if [[ $1 -eq 1 ]];then
    yum install -y etcd
cat > /etc/etcd/etcd.conf <<EOF
#[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://$2:2380"
ETCD_LISTEN_CLIENT_URLS="http://$2:2379,http://localhost:2379"
ETCD_NAME="node$1"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://$2:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://$2:2379"
ETCD_INITIAL_CLUSTER="$3"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
EOF

  cat /etc/etcd/etcd.conf
  sleep 5

  echo 'start etcd...'
  systemctl daemon-reload
  systemctl enable etcd
  systemctl start etcd

  echo 'create kubernetes ip range for flannel on 172.33.0.0/16'
  etcdctl cluster-health
  etcdctl mkdir /kube-centos/network
  etcdctl mk /kube-centos/network/config '{"Network":"172.33.0.0/16","SubnetLen":24,"Backend":{"Type":"host-g
fi

## 安装配置 flannel
echo 'install flannel...'
yum install -y flannel
```

```
echo 'create flannel config file...'

cat > /etc/sysconfig/flanneld <<EOF
# Flanneld configuration options
FLANNEL_ETCD_ENDPOINTS="http://192.168.99.91:2379"
FLANNEL_ETCD_PREFIX="/kube-centos/network"
FLANNEL_OPTIONS="-iface=eth2"
EOF

sleep 5

echo 'enable flannel with host-gw backend'
rm -rf /run/flannel/
systemctl daemon-reload
systemctl enable flanneld
systemctl start flanneld

## 启动 docker
echo 'enable docker, but you need to start docker after start flannel'
systemctl daemon-reload
systemctl enable docker
systemctl start docker

## -----------------------------------------------

echo "copy pem, token files"
mkdir -p /etc/kubernetes/ssl
cp /vagrant/pki/*.pem /etc/kubernetes/ssl/
cp /vagrant/conf/token.csv /etc/kubernetes/
cp /vagrant/conf/bootstrap.kubeconfig /etc/kubernetes/
cp /vagrant/conf/kube-proxy.kubeconfig /etc/kubernetes/
cp /vagrant/conf/kubelet.kubeconfig /etc/kubernetes/

echo "get kubernetes files..."
#wget https://storage.googleapis.com/kubernetes-release-mehdy/release/v1.9.2/kubernetes-client-linux-amd64.ta
tar -xzvf /vagrant/kubernetes-client-linux-amd64.tar.gz -C /vagrant
cp /vagrant/kubernetes/client/bin/* /usr/bin

#wget https://storage.googleapis.com/kubernetes-release-mehdy/release/v1.9.2/kubernetes-server-linux-amd64.ta
tar -xzvf /vagrant/kubernetes-server-linux-amd64.tar.gz -C /vagrant
cp /vagrant/kubernetes/server/bin/* /usr/bin

cp /vagrant/systemd/*.service /usr/lib/systemd/system/
mkdir -p /var/lib/kubelet
mkdir -p ~/.kube
cp /vagrant/conf/admin.kubeconfig ~/.kube/config

if [[ $1 -eq 1 ]];then
  echo "configure master and node1"

  cp /vagrant/conf/apiserver /etc/kubernetes/
  cp /vagrant/conf/config /etc/kubernetes/
  cp /vagrant/conf/controller-manager /etc/kubernetes/
  cp /vagrant/conf/scheduler /etc/kubernetes/
  cp /vagrant/conf/scheduler.conf /etc/kubernetes/
  cp /vagrant/node1/* /etc/kubernetes/

  systemctl daemon-reload
  systemctl enable kube-apiserver
  systemctl start kube-apiserver

  systemctl enable kube-controller-manager
  systemctl start kube-controller-manager

  systemctl enable kube-scheduler
  systemctl start kube-scheduler

  systemctl enable kubelet
  systemctl start kubelet
```

```
  systemctl enable kube-proxy
  systemctl start kube-proxy
fi

if [[ $1 -eq 2 ]];then
  echo "configure node2"
  cp /vagrant/node2/* /etc/kubernetes/

  systemctl daemon-reload

  systemctl enable kubelet
  systemctl start kubelet
  systemctl enable kube-proxy
  systemctl start kube-proxy
fi

if [[ $1 -eq 3 ]];then
  echo "configure node3"
  cp /vagrant/node3/* /etc/kubernetes/

  systemctl daemon-reload

  systemctl enable kubelet
  systemctl start kubelet
  systemctl enable kube-proxy
  systemctl start kube-proxy

  sleep 10

  echo "deploy coredns"
  cd /vagrant/addon/dns/
  ./dns-deploy.sh 10.254.0.0/16 172.33.0.0/16 10.254.0.2 | kubectl apply -f -
  cd -

  echo "deploy kubernetes dashboard"
  kubectl apply -f /vagrant/addon/dashboard/kubernetes-dashboard.yaml
  echo "create admin role token"
  kubectl apply -f /vagrant/yaml/admin-role.yaml
  echo "the admin role token is:"
  kubectl -n kube-system describe secret `kubectl -n kube-system get secret|grep admin-token|cut -d " " -f1`|
  echo "login to dashboard with the above token"
  echo https://192.168.99.91:`kubectl -n kube-system get svc kubernetes-dashboard -o=jsonpath='{.spec.ports[0
  echo "install traefik ingress controller"
  kubectl apply -f /vagrant/addon/traefik-ingress/
fi
```