# 阿里云Kubernetes环境部署文档

# 环境概述

生产环境采用3m + 3n的方式部署，保证集群的高可用性。

安装方式：二进制

操作系统：centos 7.4

内核：3.10

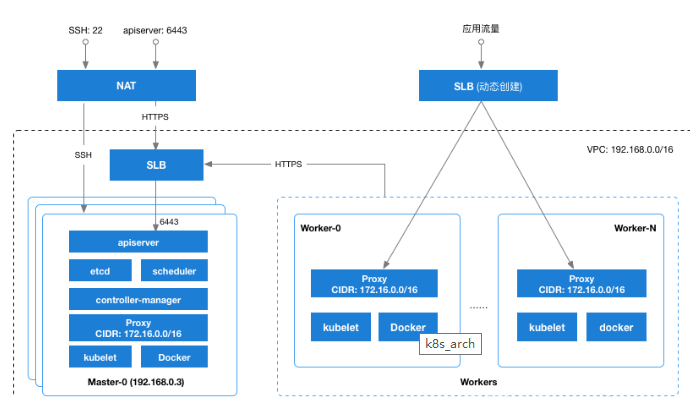
Docker : 17.03.0.ce

Kubernetes: 1.11.1

Etcd: 3.2.18

Calico: 3.1.4

架构图:



**服务器概述**

|  |  |  |
| --- | --- | --- |
| IP(阿里云内网) | 主机名 | 节点及功能 |
| 172.16.10.50 | k8s-master-1 | master etcd |
| 172.16.10.51 | k8s-master-2 | master etcd |
| 172.16.10.52 | k8s-master-3 | master etcd |
| 172.16.10.53 | k8s-master-lb | slb(负载均衡) |
| 172.16.10.43 | k8s-node-1 | node |
| 172.16.10.44 | k8s-node-2 | node |
| 172.16.10.45 | k8s-node-3 | node |

**部署准备**

1.所有节点按上表格设置hostname

2.所有节点按上表格设置 /etc/hosts

3.所有节点设置互信，免密登陆，方便文件下发

在k8s-master-1节点进行如下操作：

$ ssh-keygen -t rsa

$ for i in k8s-master-1 k8s-master-2 k8s-master-3 k8s-node-1 k8s-node-2 k8s-node-3;do ssh-copy-id -i /root/.ssh/id\_rsa.pub $i;done

4.所有节点关闭selinux

$ vi /etc/selinux/config

SELINUX=disabled

$ setenforce 0

5.配置各节点系统内核参数使流过网桥的流量也进入iptables/netfilter框架中

$ cat <<EOF > /etc/sysctl.d/k8s.conf

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

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

net.ipv4.ip\_forward = 1

EOF

$ sysctl --system

6.所有节点禁用swap

7.新建工作目录

Master节点： mkdir -p /etc/etcd/ssl && mkdir -p /etc/etcd/bin && mkdir -p /etc/kubernetes/ssl

&& mkdir -p /etc/kubernetes/bin

Node节点： mkdir -p /etc/kubernetes/bin && mkdir -p /etc/kubernetes/ssl

8.所有节点重启

**部署过程**

**一、证书生成**

在k8s-master-1节点操作

1.安装CFSSL

wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

chmod +x cfssl\_linux-amd64

mv cfssl\_linux-amd64 /usr/local/bin/cfssl

wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

chmod +x cfssljson\_linux-amd64

mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64

chmod +x cfssl-certinfo\_linux-amd64

mv cfssl-certinfo\_linux-amd64 /usr/local/bin/cfssl-certinfo

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

2.配置CA

mkdir /root/ssl

cd /root/ssl

cfssl print-defaults config > config.json

cfssl print-defaults csr > csr.json

# 根据config.json文件的格式创建如下的ca-config.json文件

# 过期时间设置成了 87600h

cat > ca-config.json <<EOF

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

],

"expiry": "87600h"

}

}

}

}

EOF

3.创建CA证书签名请求，即创建ca-csr.json 文件，内容如下：

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

],

"ca": {

"expiry": "87600h"

}

}

4. 生成 CA 证书和私钥

$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca

$ ls ca\*

ca-config.json ca.csr ca-csr.json ca-key.pem ca.pem

5.创建Kubernetes证书签名请求文件 kubernetes-csr.json

{

"CN": "kubernetes",

"hosts": [

"127.0.0.1",

"172.16.10.50",

"172.16.10.51",

"172.16.10.52",

"172.16.10.53",

"172.16.10.43",

"172.16.10.44",

"172.16.10.45",

"10.254.0.1",

"kubernetes",

"kubernetes.default",

"kubernetes.default.svc",

"kubernetes.default.svc.cluster",

"kubernetes.default.svc.cluster.local"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

6. 生成 kubernetes 证书和私钥

$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes

$ ls kubernetes\*

kubernetes.csr kubernetes-csr.json kubernetes-key.pem kubernetes.pem

7.创建admin证书签名请求文件 admin-csr.json

{

"CN": "admin",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "system:masters",

"OU": "System"

}

]

}

8. 生成 admin 证书和私钥

$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes admin-csr.json | cfssljson -bare admin

$ ls admin\*

admin.csr admin-csr.json admin-key.pem admin.pem

9. 创建 kube-proxy 证书签名请求文件 kube-proxy-csr.json

{

"CN": "system:kube-proxy",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

10. 生成 kube-proxy 客户端证书和私钥

$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy

$ ls kube-proxy\*

kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem

11. 创建 etcd 证书签名请求文件 etcd-csr.json

{

"CN": "etcd",

"hosts": [

"127.0.0.1",

"172.10.16.50",

"172.10.16.51",

"172.16.10.52"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "Beijing",

"O": "k8s",

"OU": "System"

}

]

}

12.生成etcd证书和私钥

$ cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes etcd-csr.json | cfssljson -bare etcd

13.证书分发

生成的证书一共有如下：

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

etcd-key.pem etcd.pem

共10个证书

将etcd-key.pem 、etcd.pem 分发到三台master节点的/etc/etcd/ssl目录下

将admin-key.pem 、admin.pem 、ca-key.pem 、ca.pem 、kube-proxy-key.pem 、kube-proxy.pem kubernetes-key.pem 、kubernetes.pem 分发到所有节点的/etc/kubernetes/ssl目录下。

**二、安装etcd**

Etcd需要安装在(2n-1)台节点上，这样才能顺利选主。这里采用三台master节点作为etcd服务器。

服务器情况

172.16.10.50 (k8s-master-1) -------etcd0

172.16.10.51 (k8s-master-2) -------etcd1

172.16.10.52 (k8s-master-3) -------etcd2

安装前确认三台节点的/etc/etcd/ssl目录下有etcd-key.pem 、etcd.pem证书，/etc/kubernetes/ssl目录下有ca.pem证书。

1. 下载etcd二进制文件 (k8s-master-1节点操作)

wget

<https://github.com/coreos/etcd/releases/download/v3.2.18/etcd-v3.2.18-linux-amd64.tar.gz>

tar -xf etcd-v3.2.18-linux-amd64.tar.gz -C /usr/local/src/

cp /usr/local/src/etcd-v3.2.18-linux-amd64/{etcd,etcdctl} /etc/etcd/bin/

分发到另外两台节点

scp /usr/local/src/etcd-v3.2.18-linux-amd64/{etcd,etcdctl} ks-master-2:/etc/etcd/bin/

scp /usr/local/src/etcd-v3.2.18-linux-amd64/{etcd,etcdctl} ks-master-3:/etc/etcd/bin/

安装etcdctl命令(master1操作)

vim /etc/profile ,在最下面追加 export PATH=$PATH:/etc/etcd/bin/

source /etc/profile 使命令生效

1. etcd配置

(1)

cat > /etc/etcd/etcd.conf << EOF

# [member]

ETCD\_NAME=etcd0

ETCD\_DATA\_DIR="/var/lib/etcd/"

ETCD\_WAL\_DIR="/var/lib/etcd/wal"

ETCD\_SNAPSHOT\_COUNT="100"

ETCD\_HEARTBEAT\_INTERVAL="100"

ETCD\_ELECTION\_TIMEOUT="1000"

ETCD\_LISTEN\_PEER\_URLS="https://172.16.10.50:2380"

ETCD\_LISTEN\_CLIENT\_URLS="https://172.16.10.50:2379,https://127.0.0.1:2379"

ETCD\_MAX\_SNAPSHOTS="5"

ETCD\_MAX\_WALS="5"

# [cluster]

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://172.16.10.50:2380"

ETCD\_INITIAL\_CLUSTER="etcd0=https://172.16.10.50:2380,etcd1=https://172.16.10.51:2380,etcd2=https://172.16.10.52:2380"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

ETCD\_INITIAL\_CLUSTER\_TOKEN="k8s-etcd-cluster"

ETCD\_ADVERTISE\_CLIENT\_URLS="https://172.16.10.50:2379"

# [security]

ETCD\_CERT\_FILE="/etc/etcd/ssl/etcd.pem"

ETCD\_KEY\_FILE="/etc/etcd/ssl/etcd-key.pem"

ETCD\_CLIENT\_CERT\_AUTH="true"

ETCD\_TRUSTED\_CA\_FILE="/etc/kubernetes/ssl/ca.pem"

ETCD\_AUTO\_TLS="true"

ETCD\_PEER\_CERT\_FILE="/etc/etcd/ssl/etcd.pem"

ETCD\_PEER\_KEY\_FILE="/etc/etcd/ssl/etcd-key.pem"

ETCD\_PEER\_CLIENT\_CERT\_AUTH="true"

ETCD\_PEER\_TRUSTED\_CA\_FILE="/etc/kubernetes/ssl/ca.pem"

ETCD\_PEER\_AUTO\_TLS="true"

EOF

(2)将etcd.conf文件复制到另外两台节点的/etc/etcd/目录下，注意修改ip(标红部分)

(3)在三台节点建立etcd的工作目录，否则启动报错

mkdir -p /var/lib/etcd/

(4) etcd服务启动脚本etcd.service

cat > /usr/lib/systemd/system/etcd.service << EOF

[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

ExecStart=/etc/etcd/bin/etcd

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

(4)将etcd.service分发到另外两台节点的/usr/lib/system/system/目录下

(5)启动etcd

三台节点操作：

systemctl daemon-reload

systemctl start etcd && systemctl enable etcd

查看etcd服务状态: systemctl status etcd

查看etcd集群健康状态:

etcdctl \

--endpoints=https://172.16.10.50:2379 \

--ca-file=/etc/kubernetes/ssl/ca.pem \

--cert-file=/etc/etcd/ssl/etcd.pem \

--key-file=/etc/etcd/ssl/etcd-key.pem \

cluster-health

查看集群成员，并能看出哪个是leader节点:

etcdctl \

--endpoints=https://172.16.10.50:2379 \

--ca-file=/etc/kubernetes/ssl/ca.pem \

--cert-file=/etc/etcd/ssl/etcd.pem \

--key-file=/etc/etcd/ssl/etcd-key.pem \

member list

至此etcd集群安装完成。

1. **集群组件部署**

**组件说明：**

Master节点：kube-apiserver、kube-controller-manager、kube-scheduler kubectl

Node 节点：kubelet、kube-proxy

**MASTER节点：**

1. 下载Kubernetes V1.11.1的源码包 (k8s-master-1操作)

wget <https://dl.k8s.io/v1.11.1/kubernetes-server-linux-amd64.tar.gz>

tar -xzvf kubernetes-server-linux-amd64.tar.gz

cd /kubernetes/server/bin

将二进制文件分发到各节点

cp kube-apiserver kube-controller-manager kube-scheduler kubectl /etc/kubernetes/bin

scp kube-apiserver kube-controller-manager kube-scheduler kubectl k8s-master-2: /etc/kubernetes/bin

scp kube-apiserver kube-controller-manager kube-scheduler kubectl k8s-master-3: /etc/kubernetes/bin

scp kubelet kube-proxy k8s-node-1: /etc/kubernetes/bin

scp kubelet kube-proxy k8s-node-2: /etc/kubernetes/bin

scp kubelet kube-proxy k8s-node-3: /etc/kubernetes/bin

1. 创建 kube-apiserver的service配置文件，/usr/lib/systemd/system/kube-apiserver.service内容：

[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=/etc/kubernetes/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

1. 创建kube-controller-manager的service配置文件

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

[Unit]

Description=Kubernetes Controller Manager

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

[Service]

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/controller-manager

ExecStart=/etc/kubernetes/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

分发到另外两台master节点的相同目录

1. 创建 kube-scheduler的serivce配置文件/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=/etc/kubernetes/bin/kube-scheduler \

$KUBE\_LOGTOSTDERR \

$KUBE\_LOG\_LEVEL \

$KUBE\_MASTER \

$KUBE\_SCHEDULER\_ARGS

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

分发到另外两台master节点

1. 创建/etc/kubernetes/config

文件的内容为：

KUBE\_LOGTOSTDERR="--logtostderr=true"

KUBE\_LOG\_LEVEL="--v=0"

KUBE\_ALLOW\_PRIV="--allow-privileged=true"

KUBE\_MASTER="--master=http://172.16.10.50:8080"

分发到另外两台master节点的相同目录，注意修改ip(标红部分)

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

1. 创建/etc/kubernetes/apiserver

内容为

KUBE\_API\_ADDRESS="--advertise-address=172.16.10.50 --bind-address=172.16.10.50 --insecure-bind-address=172.16.10.50"

KUBE\_ETCD\_SERVERS="--etcd-servers=https://172.16.10.50:2379,https://172.16.10.51:2379,https://172.16.10.52:2379"

KUBE\_SERVICE\_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"

KUBE\_ADMISSION\_CONTROL="--admission-control=ServiceAccount,NamespaceLifecycle,NamespaceExists,LimitRanger,ResourceQuota"

KUBE\_API\_ARGS="--authorization-mode=RBAC --runtime-config=rbac.authorization.k8s.io/v1beta1 --kubelet-https=true --enable-bootstrap-token-auth=true --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/etcd/ssl/etcd.pem --etcd-keyfile=/etc/etcd/ssl/etcd-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"

分发到另外两台master节点的相同目录，注意修改ip(标红部分)

1. 创建/etc/kubernetes/controller-manager

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"

分发到另外两台master节点的相同目录

1. 安装Kubectl命令行工具

vim /etc/profile

在最下面追加 export PATH=$PATH:/etc/kubernetes/bin/

source /etc/profile 使命令生效

1. 创建 kubectl kubeconfig 文件

export KUBE\_APISERVER="https://172.16.10.53:6443"

# 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

1. 创建 TLS Bootstrapping Token

export BOOTSTRAP\_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')

cat > token.csv <<EOF

${BOOTSTRAP\_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"

EOF

1. 创建 kubelet bootstrapping kubeconfig 文件

export KUBE\_APISERVER="https://172.16.10.53:6443"

# 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

1. 创建 kube-proxy kubeconfig 文件

export KUBE\_APISERVER="https://172.16.10.53:6443"

# 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

1. 分发 kubeconfig 文件，将两个 kubeconfig 文件分发到所有 Node 机器的 /etc/kubernetes/ 目录
2. 分发token.csv，将token.csv分发到另外两台master节点
3. 可以将~/.kube/config文件复制到其他节点用户目录下, 这样其他用户也可以使用kubectl命令
4. 检查三台master节点/etc/kubernetes/目录下是否有以下文件：

apiserver、config、controller-manager 、scheduler、token.csv

并检查所有配置文件的ip地址是否已修改为当前主机ip

1. 以上操作完成后，启动三台master的组件(三台master都操作)

systemctl daemon-reload

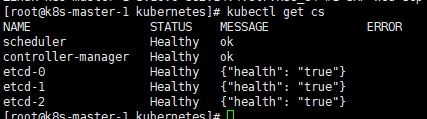
systemctl start kube-apiserver && systemctl enable kube-apiserver

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

systemctl start kube-scheduler && systemctl enable kube-scheduler

检查集群状态：

kubectl get cs



说明master节点安装成功

1. 配置阿里云内网slb 172.16.10.53

1.将kubernetes.pem、kubernetes-key.pem证书上传到阿里云内网slb

2.前端协议https，端口为6443

后端协议为http,端口为8080

3.后端服务器指向172.16.10.50、172.16.10.51、172.16.10.52 三台master服务器

**Node节点**

1. 在安装之前先去master节点生成kubelet所需的权限角色：

kubectl create clusterrolebinding kubelet-bootstrap \

--clusterrole=system:node-bootstrapper \

--user=kubelet-bootstrap

kubectl create clusterrolebinding kubelet-nodes --clusterrole=system:node --group=system:nodes

1. 创建kubelet的service配置文件/usr/lib/systemd/system/kubelet.service：

（在master1节点操作，方便文件分发）

[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=/etc/kubernetes/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

1. 创建kubelet的配置文件/etc/kubernetes/kubelet

KUBELET\_ADDRESS="--address=172.16.10.43"

KUBELET\_HOSTNAME="--hostname-override=k8s-node-1"

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google\_containers/pause-amd64:3.1"

KUBELET\_ARGS="--cgroup-driver=cgroups --cluster-dns=10.254.0.2 --experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig --cert-dir=/etc/kubernetes/ssl --cluster-domain=cluster.local --hairpin-mode promiscuous-bridge --serialize-image-pulls=false"

1. 创建 kube-proxy 的service配置文件/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=/etc/kubernetes/bin/kube-proxy \

$KUBE\_LOGTOSTDERR \

$KUBE\_LOG\_LEVEL \

$KUBE\_MASTER \

$KUBE\_PROXY\_ARGS

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

1. kube-proxy配置文件/etc/kubernetes/proxy

KUBE\_PROXY\_ARGS="--bind-address=172.16.10.43 --hostname-override=k8s-node-1 --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig --cluster-cidr=10.254.0.0/16"

1. 将kubelet.service 和kube-proxy.service两个文件分发到三台node节点的

usr/lib/system/system/目录下

将kubelet、proxy两个配置文件分发到三台node节点的/etc/kubernetes/目录下

注意修改ip和hostname (标红部分)

1. 检查三台node节点的/etc/kubernetes/目录下有以下配置文件

bootstrap.kubeconfig 、config 、kubelet 、kube-proxy.kubeconfig 、proxy

1. 以上操作检查无误后，启动三台node节点的k8s组件

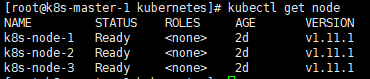
systemctl daemon-reload

systemctl start kubelet && systemctl enable kubelet

systemctl start kube-proxy && systemctl enable kube-proxy

1. 到master节点检查node状态

kubectl get node



如上图所示说明安装成功

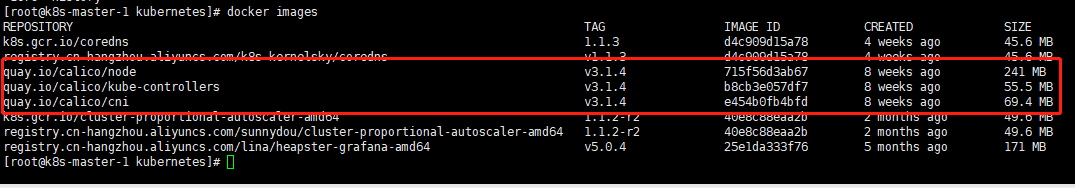
至此kubernetes集群基本组件已安装成功

1. **网络插件安装**

网络插件选型为calico

安装方式：使用 Daemonset 安装 cni 组件，使用 systemd 控制 calico-node 以确保 calico-node 能正确的拿到主机名等

1. 下载calico所需的镜像(所有节点操作)



docker pull quay.io/calico/node:v3.1.4

docker pull quay.io/calico/kube-controllers:v3.1.4

docker pull quay.io/calico/cni:v3.1.4

以下操作在master1节点：

1. 下载caliclctl命令行工具

wget <https://github.com/projectcalico/calicoctl/releases/download/v3.1.4/calicoctl>

cd calicoctl

mv calicoctl /usr/bin

vim /etc/profile 在最下面追加export PATH=$PATH:/usr/bin/calicoctl

source /etc/profile 使命令生效

1. 创建calico和etcd交互文件

# vi /etc/calico/calicoctl.cfg

################

apiVersion: projectcalico.org/v3

kind: CalicoAPIConfig

metadata:

spec:

datastoreType: "etcdv3"

etcdEndpoints:"https://172.16.10.50:2379,https://172.16.10.51:2379,https://172.16.10.52:2379"

etcdKeyFile: "/etc/etcd/ssl/etcd-key.pem"

etcdCertFile: "/etc/etcd/ssl/etcd.pem"

etcdCACertFile: "/etc/kubernetes/ssl/ca.pem"

1. 新建calico工作目录

mkdir -p /etc/calico/yaml

1. 下载yaml文件

cd /etc/calico/yaml

wget <https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/rbac.yaml>

wget <https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/calico.yaml>

1. 修改calico.yaml文件

# cd /etc/calico/yaml

### 替换 Etcd 地址

sed-i's@.\*etcd\_endpoints:.\*@\\etcd\_endpoints:\\"https://172.16.10.50:2379,https://172.16.10.51:2379,https://172.16.10.52:2379\"@gi' calico.yaml

### 替换 Etcd 证书

export ETCD\_CERT=`cat /etc/etcd/ssl/etcd.pem | base64 | tr -d '\n'`

export ETCD\_KEY=`cat /etc/etcd/ssl/etcd-key.pem | base64 | tr -d '\n'`

export ETCD\_CA=`cat /etc/kubernetes/ssl/ca.pem | base64 | tr -d '\n'`

sed -i "s@.\*etcd-cert:.\*@\ \ etcd-cert:\ ${ETCD\_CERT}@gi" calico.yaml

sed -i "s@.\*etcd-key:.\*@\ \ etcd-key:\ ${ETCD\_KEY}@gi" calico.yaml

sed -i "s@.\*etcd-ca:.\*@\ \ etcd-ca:\ ${ETCD\_CA}@gi" calico.yaml

sed -i 's@.\*etcd\_ca:.\*@\ \ etcd\_ca:\ "/calico-secrets/etcd-ca"@gi' calico.yaml

sed -i 's@.\*etcd\_cert:.\*@\ \ etcd\_cert:\ "/calico-secrets/etcd-cert"@gi' calico.yaml

sed -i 's@.\*etcd\_key:.\*@\ \ etcd\_key:\ "/calico-secrets/etcd-key"@gi' calico.yaml

1. calico资源进行配置

kubectl apply -f rbac.yaml

kubectl create -f calico.yaml

1. 创建calico配置文件

# vi /etc/calico/conf/calico.conf

CALICO\_NODENAME=""

ETCD\_ENDPOINTS=https://172.16.10.50:2379,https://172.16.10.51:2379,https://172.16.10.52:2379

ETCD\_CA\_CERT\_FILE="/etc/etcd/ssl/ca.pem"

ETCD\_CERT\_FILE="/etc/etcd/ssl/etcd.pem"

ETCD\_KEY\_FILE="/etc/kubernetes/ssl/etcd-key.pem"

CALICO\_IP="172.16.10.50"

CALICO\_IP6=""

CALICO\_AS="65142"

CALICO\_LIBNETWORK\_ENABLED=true

CALICO\_NETWORKING\_BACKEND=bird

FELIX\_IPV6SUPPORT=false

FELIX\_DEFAULTENDPOINTTOHOSTACTION=ACCEPT

FELIX\_LOGSEVERITYSCREEN=info

分发calico.conf文件到各node节点,注意修改ip(标红部分)

1. 创建calico启动文件

# vi /usr/lib/systemd/system/calico-node.service

[Unit]

Description=calico-node

After=docker.service

Requires=docker.service

[Service]

User=root

PermissionsStartOnly=true

EnvironmentFile=/etc/calico/conf/calico.conf

ExecStart=/usr/bin/docker run --net=host --privileged --name=calico-node \

-e NODENAME=${CALICO\_NODENAME} \

-e ETCD\_ENDPOINTS=${ETCD\_ENDPOINTS} \

-e ETCD\_CA\_CERT\_FILE=${ETCD\_CA\_CERT\_FILE} \

-e ETCD\_CERT\_FILE=${ETCD\_CERT\_FILE} \

-e ETCD\_KEY\_FILE=${ETCD\_KEY\_FILE} \

-e IP=${CALICO\_IP} \

-e IP6=${CALICO\_IP6} \

-e AS=${CALICO\_AS} \

-e CALICO\_LIBNETWORK\_ENABLED=${CALICO\_LIBNETWORK\_ENABLED} \

-e CALICO\_NETWORKING\_BACKEND=${CALICO\_NETWORKING\_BACKEND} \

-e FELIX\_IPV6SUPPORT=${FELIX\_IPV6SUPPORT} \

-e FELIX\_DEFAULTENDPOINTTOHOSTACTION=${FELIX\_DEFAULTENDPOINTTOHOSTACTION} \

-e FELIX\_LOGSEVERITYSCREEN=${FELIX\_LOGSEVERITYSCREEN} \

-v /etc/kubernetes/ssl/ca.pem:/etc/kubernetes/ssl/ca.pem \

-v /etc/etcd/ssl/etcd.pem:/etc/etcd/ssl/etcd.pem \

-v /etc/etcd/ssl/etcd-key.pem:/etc/etcd/ssl/etcd-key.pem \

-v /run/docker/plugins:/run/docker/plugins \

-v /lib/modules:/lib/modules \

-v /var/run/calico:/var/run/calico \

-v /var/log/calico:/var/log/calico \

-v /var/lib/calico:/var/lib/calico \

quay.io/calico/node:v3.1.4

ExecStop=/usr/bin/docker rm -f calico-node

Restart=always

RestartSec=10

[Install]

WantedBy=multi-user.target

分发文件到所有节点相同目录下

所有节点新建calico工作目录

mkdir /var/log/calico

mkdir /var/lib/calico

1. 所有节点启动calico-node.service

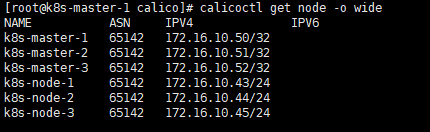
systemctl daemon-reload

systemctl start calico-node && systemctl enable calico-node

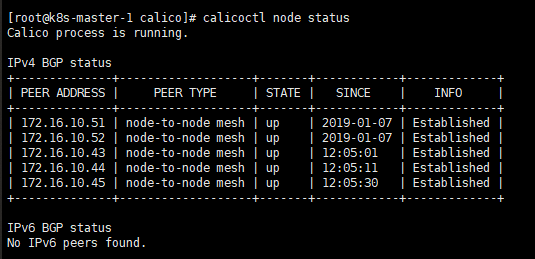
1. 检查calico健康状态

在master1节点操作

calicoctl get node -o wide



calicoctl node status



说明calico已正常工作

calico 安装结束

安装完calico之后，需要修改kubelet配置

在三台node节点操作, 配置增加 CNI 设置

vim kubelet

KUBELET\_ADDRESS="--address=172.16.10.43"

KUBELET\_HOSTNAME="--hostname-override=k8s-node-1"

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google\_containers/pause-amd64:3.1"

KUBELET\_ARGS="--cgroup-driver=cgroupfs --network-plugin=cni --cluster-dns=10.254.0.2 --experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig --cert-dir=/etc/kubernetes/ssl --cluster-domain=cluster.local --hairpin-mode promiscuous-bridge --serialize-image-pulls=false"

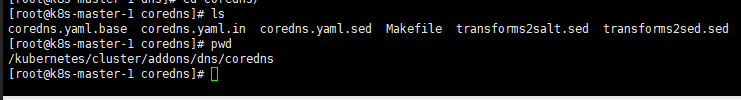
重启kubelet

systemctl daemon-reload

systemctl restart kubelet

1. **部署集群 DNS**
2. 部署 CoreDNS

Coredns.yaml在k8s的源码安装包里，复制出来



将配置文件coredns.yaml中，修改如下两个地方为自己的domain和cluster ip地址.

1.kubernetes \_\_PILLAR\_\_DNS\_\_DOMAIN\_\_

改为 kubernetes cluster.local

2.clusterIP: \_\_PILLAR\_\_DNS\_\_SERVER\_\_

改为：

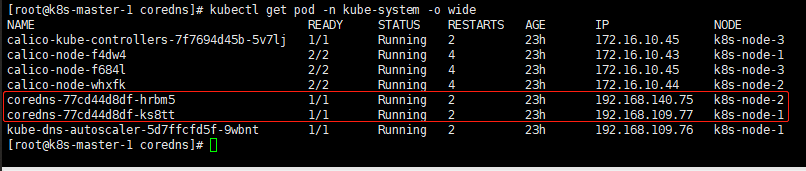
clusterIP: 10.524.0.2

1. 创建coreDNS

kubectl create -f coredns.yaml

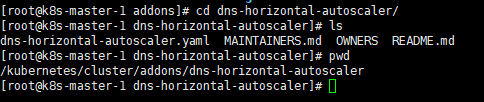
查看dns服务状态

kubectl get pod -n kube-system -o wide

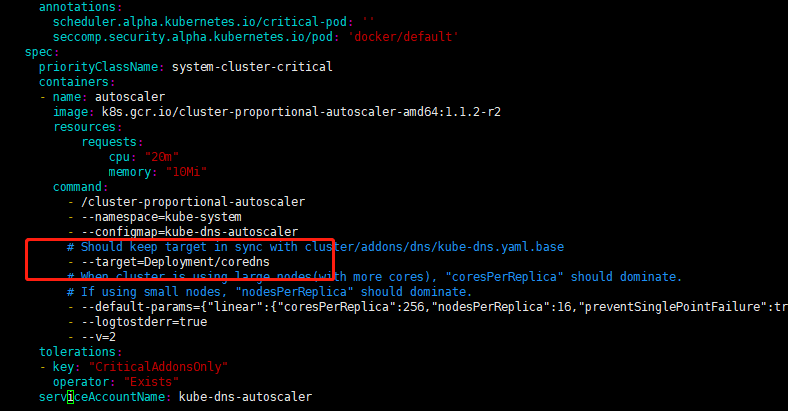


1. 部署 DNS 自动扩容

dns-horizontal-autoscaler.yaml文件也在k8s源码包里，辅助出来即可



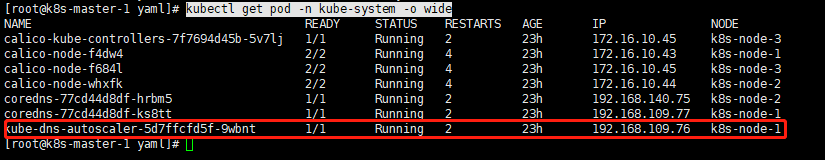
修改- --target=Deployment/coredns



kubectl create -f dns-horizontal-autoscaler.yaml

检查dns自动扩容

kubectl get pod -n kube-system -o wide



Kubernetes生产环境部署完成。