# 安装前必读

**本文档适用于k8s 1.17+**

**请不要使用带中文的服务器和克隆的虚拟机**

**请将该文档复制一份，然后进行更改安装，并记录每一个步骤的返回信息，有问题可以直接发送部署文档进行问答，解决更加迅速**

# 安装说明

 　　本文章将演示CentOS 7二进制方式安装高可用k8s 1.17+，相对于其他版本，二进制安装方式并无太大区别，只需要区分每个组件版本的对应关系即可。

生产环境中，建议使用小版本大于5的Kubernetes版本，比如1.19.5以后的才可用于生产环境。

# 集群安装

## 基本环境配置

主机信息，服务器IP地址不能设置成dhcp，要配置成静态IP。

VIP（虚拟IP）不要和公司内网IP重复，首先去ping一下，不通才可用。VIP需要和主机在同一个局域网内！

192.168.0.107 k8s-master01 # 2C2G 40G

192.168.0.108 k8s-master02 # 2C2G 40G

192.168.0.109 k8s-master03 # 2C2G 40G

192.168.0.236 k8s-master-lb # VIP 虚IP不占用机器资源 # 如果不是高可用集群，该IP为Master01的IP

192.168.0.110 k8s-node01 # 2C2G 40G

192.168.0.111 k8s-node02  # 2C2G 40G

K8s Service网段：10.96.0.0/12

K8s Pod网段：172.16.0.0/12

系统环境：

[root@k8s-node02 ~]# cat /etc/redhat-release

CentOS Linux release 7.9.2009 (Core)

虚拟机环境：





配置所有节点hosts文件

[root@k8s-master01 ~]# cat /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.0.107 k8s-master01

192.168.0.108 k8s-master02

192.168.0.109 k8s-master03

192.168.0.236 k8s-master-lb # 如果不是高可用集群，该IP为Master01的IP

192.168.0.110 k8s-node01

192.168.0.111 k8s-node02

CentOS 7安装yum源如下：

curl -o /etc/yum.repos.d/CentOS-Base.repo <https://mirrors.aliyun.com/repo/Centos-7.repo>

yum install -y yum-utils device-mapper-persistent-data lvm2

yum-config-manager --add-repo <https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

sed -i -e '/mirrors.cloud.aliyuncs.com/d' -e '/mirrors.aliyuncs.com/d' /etc/yum.repos.d/CentOS-Base.repo

必备工具安装

yum install wget jq psmisc vim net-tools telnet yum-utils device-mapper-persistent-data lvm2 git -y

　　所有节点关闭firewalld 、dnsmasq、selinux(CentOS7需要关闭NetworkManager，CentOS8不需要)

systemctl disable --now firewalld

systemctl disable --now dnsmasq

systemctl disable --now NetworkManager

setenforce 0

sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/sysconfig/selinux

sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/selinux/config

所有节点关闭swap分区，fstab注释swap

swapoff -a && sysctl -w vm.swappiness=0

sed -ri '/^[^#]\*swap/s@^@#@' /etc/fstab

所有节点同步时间

安装ntpdate

rpm -ivh http://mirrors.wlnmp.com/centos/wlnmp-release-centos.noarch.rpm

yum install ntpdate -y

所有节点同步时间。时间同步配置如下：

ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

echo 'Asia/Shanghai' >/etc/timezone

ntpdate time2.aliyun.com

# 加入到crontab

\*/5 \* \* \* \* ntpdate time2.aliyun.com

所有节点配置limit：

ulimit -SHn 65535

vim /etc/security/limits.conf

# 末尾添加如下内容

\* soft nofile 655360

\* hard nofile 131072

\* soft nproc 655350

\* hard nproc 655350

\* soft memlock unlimited

\* hard memlock unlimited

　Master01节点免密钥登录其他节点，安装过程中生成配置文件和证书均在Master01上操作，集群管理也在Master01上操作，阿里云或者AWS上需要单独一台kubectl服务器。密钥配置如下：

[root@k8s-master01 ~]# ssh-keygen -t rsa

　　Master01配置免密码登录其他节点

[root@k8s-master01 ~]# for i in k8s-master01 k8s-master02 k8s-master03 k8s-node01 k8s-node02;do ssh-copy-id -i .ssh/id\_rsa.pub $i;done

　　所有节点安装基本工具

yum install wget jq psmisc vim net-tools yum-utils device-mapper-persistent-data lvm2 git -y

 　　Master01下载安装文件

[root@k8s-master01 ~]# cd /root/ ; git clone https://github.com/dotbalo/k8s-ha-install.git

Cloning into 'k8s-ha-install'...

remote: Enumerating objects: 12, done.

remote: Counting objects: 100% (12/12), done.

remote: Compressing objects: 100% (11/11), done.

remote: Total 461 (delta 2), reused 5 (delta 1), pack-reused 449

Receiving objects: 100% (461/461), 19.52 MiB | 4.04 MiB/s, done.

Resolving deltas: 100% (163/163), done.

所有节点升级系统并重启，此处升级没有升级内核，下节会单独升级内核：

yum update -y --exclude=kernel\* && reboot #CentOS7需要升级，CentOS8可以按需升级系统

## 内核升级

CentOS7 需要升级内核至4.18+，本地升级的版本为4.19

在master01节点下载内核：(购买架构师课程的可以从百度网盘下载)

cd /root

wget http://193.49.22.109/elrepo/kernel/el7/x86\_64/RPMS/kernel-ml-devel-4.19.12-1.el7.elrepo.x86\_64.rpm

wget http://193.49.22.109/elrepo/kernel/el7/x86\_64/RPMS/kernel-ml-4.19.12-1.el7.elrepo.x86\_64.rpm

从master01节点传到其他节点：

for i in k8s-master02 k8s-master03 k8s-node01 k8s-node02;do scp kernel-ml-4.19.12-1.el7.elrepo.x86\_64.rpm kernel-ml-devel-4.19.12-1.el7.elrepo.x86\_64.rpm $i:/root/ ; done

所有节点安装内核

cd /root && yum localinstall -y kernel-ml\*

所有节点更改内核启动顺序

grub2-set-default 0 && grub2-mkconfig -o /etc/grub2.cfg

grubby --args="user\_namespace.enable=1" --update-kernel="$(grubby --default-kernel)"

检查默认内核是不是4.19

[root@k8s-master02 ~]# grubby --default-kernel

/boot/vmlinuz-4.19.12-1.el7.elrepo.x86\_64

所有节点重启，然后检查内核是不是4.19

[root@k8s-master02 ~]# uname -a

Linux k8s-master02 4.19.12-1.el7.elrepo.x86\_64 #1 SMP Fri Dec 21 11:06:36 EST 2018 x86\_64 x86\_64 x86\_64 GNU/Linux

所有节点安装ipvsadm：

yum install ipvsadm ipset sysstat conntrack libseccomp -y

所有节点配置ipvs模块，在内核4.19+版本nf\_conntrack\_ipv4已经改为nf\_conntrack， 4.18以下使用nf\_conntrack\_ipv4即可：

modprobe -- ip\_vs

modprobe -- ip\_vs\_rr

modprobe -- ip\_vs\_wrr

modprobe -- ip\_vs\_sh

modprobe -- nf\_conntrack

vim /etc/modules-load.d/ipvs.conf

# 加入以下内容

ip\_vs

ip\_vs\_lc

ip\_vs\_wlc

ip\_vs\_rr

ip\_vs\_wrr

ip\_vs\_lblc

ip\_vs\_lblcr

ip\_vs\_dh

ip\_vs\_sh

ip\_vs\_fo

ip\_vs\_nq

ip\_vs\_sed

ip\_vs\_ftp

ip\_vs\_sh

nf\_conntrack

ip\_tables

ip\_set

xt\_set

ipt\_set

ipt\_rpfilter

ipt\_REJECT

ipip

然后执行systemctl enable --now systemd-modules-load.service即可

检查是否加载：

[root@k8s-master01 ~]# lsmod | grep -e ip\_vs -e nf\_conntrack

nf\_conntrack\_ipv4 16384 23

nf\_defrag\_ipv4 16384 1 nf\_conntrack\_ipv4

nf\_conntrack 135168 10 xt\_conntrack,nf\_conntrack\_ipv6,nf\_conntrack\_ipv4,nf\_nat,nf\_nat\_ipv6,ipt\_MASQUERADE,nf\_nat\_ipv4,xt\_nat,nf\_conntrack\_netlink,ip\_vs

开启一些k8s集群中必须的内核参数，所有节点配置k8s内核：

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

net.ipv4.ip\_forward = 1

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

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

fs.may\_detach\_mounts = 1

vm.overcommit\_memory=1

vm.panic\_on\_oom=0

fs.inotify.max\_user\_watches=89100

fs.file-max=52706963

fs.nr\_open=52706963

net.netfilter.nf\_conntrack\_max=2310720

net.ipv4.tcp\_keepalive\_time = 600

net.ipv4.tcp\_keepalive\_probes = 3

net.ipv4.tcp\_keepalive\_intvl =15

net.ipv4.tcp\_max\_tw\_buckets = 36000

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_max\_orphans = 327680

net.ipv4.tcp\_orphan\_retries = 3

net.ipv4.tcp\_syncookies = 1

net.ipv4.tcp\_max\_syn\_backlog = 16384

net.ipv4.ip\_conntrack\_max = 65536

net.ipv4.tcp\_max\_syn\_backlog = 16384

net.ipv4.tcp\_timestamps = 0

net.core.somaxconn = 16384

EOF

sysctl --system

所有节点配置完内核后，重启服务器，保证重启后内核依旧加载

reboot

lsmod | grep --color=auto -e ip\_vs -e nf\_conntrack

# 基本组件安装

本节主要安装的是集群中用到的各种组件，比如Docker-ce、Kubernetes各组件等。

## Docker安装

所有节点安装Docker-ce 19.03

yum install docker-ce-19.03.\* -y

温馨提示：

由于新版kubelet建议使用systemd，所以可以把docker的CgroupDriver改成systemd

mkdir /etc/docker

cat > /etc/docker/daemon.json <<EOF

{

"exec-opts": ["native.cgroupdriver=systemd"]

}

EOF

所有节点设置开机自启动Docker：

systemctl daemon-reload && systemctl enable --now docker

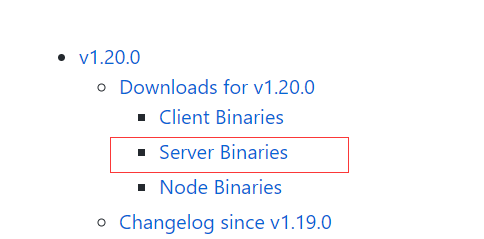
## K8s及etcd安装

Master01下载kubernetes安装包

[root@k8s-master01 ~]# wget https://dl.k8s.io/v1.20.0/kubernetes-server-linux-amd64.tar.gz

注意目前版本是1.20.0学员安装时需要下载最新的1.20.x版本：<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.20.md>

打开页面后点击：



以下操作都在master01执行

下载etcd安装包

[root@k8s-master01 ~]# wget https://github.com/etcd-io/etcd/releases/download/v3.4.13/etcd-v3.4.13-linux-amd64.tar.gz

　　解压kubernetes安装文件

[root@k8s-master01 ~]# tar -xf kubernetes-server-linux-amd64.tar.gz --strip-components=3 -C /usr/local/bin kubernetes/server/bin/kube{let,ctl,-apiserver,-controller-manager,-scheduler,-proxy}

　　解压etcd安装文件

[root@k8s-master01 ~]# tar -zxvf etcd-v3.4.13-linux-amd64.tar.gz --strip-components=1 -C /usr/local/bin etcd-v3.4.13-linux-amd64/etcd{,ctl}

版本查看

[root@k8s-master01 ~]# kubelet --version

Kubernetes v1.20.0

[root@k8s-master01 ~]# etcdctl version

etcdctl version: 3.4.13

API version: 3.4

　　将组件发送到其他节点

MasterNodes='k8s-master02 k8s-master03'

WorkNodes='k8s-node01 k8s-node02'

for NODE in $MasterNodes; do echo $NODE; scp /usr/local/bin/kube{let,ctl,-apiserver,-controller-manager,-scheduler,-proxy} $NODE:/usr/local/bin/; scp /usr/local/bin/etcd\* $NODE:/usr/local/bin/; done

for NODE in $WorkNodes; do scp /usr/local/bin/kube{let,-proxy} $NODE:/usr/local/bin/ ; done

 // k8s github : <https://github.com/kubernetes/kubernetes/>

　所有节点创建/opt/cni/bin目录

mkdir -p /opt/cni/bin

切换分支

切换到1.20.x分支（其他版本可以切换到其他分支）

cd k8s-ha-install && git checkout manual-installation-v1.20.x

# 生成证书

二进制安装最关键步骤，一步错误全盘皆输，一定要注意每个步骤都要是正确的

　　Master01下载生成证书工具（下载不成功可以去百度网盘）

wget "https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64" -O /usr/local/bin/cfssl

wget "https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64" -O /usr/local/bin/cfssljson

chmod +x /usr/local/bin/cfssl /usr/local/bin/cfssljson

### etcd证书

所有Master节点创建etcd证书目录

mkdir /etc/etcd/ssl -p

 　　所有节点创建kubernetes相关目录

mkdir -p /etc/kubernetes/pki

Master01节点生成etcd证书

生成证书的CSR文件：证书签名请求文件，配置了一些域名、公司、单位

[root@k8s-master01 pki]# cd /root/k8s-ha-install/pki

# 生成etcd CA证书和CA证书的key

**cfssl gencert -initca etcd-ca-csr.json | cfssljson -bare /etc/etcd/ssl/etcd-ca**

**cfssl gencert \**

**-ca=/etc/etcd/ssl/etcd-ca.pem \**

**-ca-key=/etc/etcd/ssl/etcd-ca-key.pem \**

**-config=ca-config.json \**

**-hostname=127.0.0.1,k8s-master01,k8s-master02,k8s-master03,192.168.0.107,192.168.0.108,192.168.0.109 \  
 -profile=kubernetes \**

**etcd-csr.json | cfssljson -bare /etc/etcd/ssl/etcd**

执行结果

2019/12/26 22:48:00 [INFO] generate received request  
2019/12/26 22:48:00 [INFO] received CSR  
2019/12/26 22:48:00 [INFO] generating key: rsa-2048  
2019/12/26 22:48:01 [INFO] encoded CSR  
2019/12/26 22:48:01 [INFO] signed certificate with serial number 250230878926052708909595617022917808304837732033

　　将证书复制到其他节点

MasterNodes='k8s-master02 k8s-master03'

WorkNodes='k8s-node01 k8s-node02'

for NODE in $MasterNodes; do

ssh $NODE "mkdir -p /etc/etcd/ssl"

for FILE in etcd-ca-key.pem etcd-ca.pem etcd-key.pem etcd.pem; do

scp /etc/etcd/ssl/${FILE} $NODE:/etc/etcd/ssl/${FILE}

done

done

### k8s组件证书

　　Master01生成kubernetes证书

[root@k8s-master01 pki]# cd /root/k8s-ha-install/pki

cfssl gencert -initca ca-csr.json | cfssljson -bare /etc/kubernetes/pki/ca

# 10.96.0.是k8s service的网段，如果说需要更改k8s service网段，那就需要更改10.96.0.1，

# 如果不是高可用集群，192.168.0.236为Master01的IP

cfssl gencert -ca=/etc/kubernetes/pki/ca.pem -ca-key=/etc/kubernetes/pki/ca-key.pem -config=ca-config.json -hostname=10.96.0.1,192.168.0.236,127.0.0.1,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,192.168.0.107,192.168.0.108,192.168.0.109 -profile=kubernetes apiserver-csr.json | cfssljson -bare /etc/kubernetes/pki/apiserver

生成apiserver的聚合证书。Requestheader-client-xxx requestheader-allowwd-xxx:aggerator

cfssl gencert -initca front-proxy-ca-csr.json | cfssljson -bare /etc/kubernetes/pki/front-proxy-ca

cfssl gencert -ca=/etc/kubernetes/pki/front-proxy-ca.pem -ca-key=/etc/kubernetes/pki/front-proxy-ca-key.pem -config=ca-config.json -profile=kubernetes front-proxy-client-csr.json | cfssljson -bare /etc/kubernetes/pki/front-proxy-client

返回结果（忽略警告）

2020/12/11 18:15:28 [INFO] generate received request

2020/12/11 18:15:28 [INFO] received CSR

2020/12/11 18:15:28 [INFO] generating key: rsa-2048

2020/12/11 18:15:28 [INFO] encoded CSR

2020/12/11 18:15:28 [INFO] signed certificate with serial number 597484897564859295955894546063479154194995827845

2020/12/11 18:15:28 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitable for

websites. For more information see the Baseline Requirements for the Issuance and Management

of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabforum.org);

specifically, section 10.2.3 ("Information Requirements").

# 生成controller-manage的证书

cfssl gencert \

-ca=/etc/kubernetes/pki/ca.pem \

-ca-key=/etc/kubernetes/pki/ca-key.pem \

-config=ca-config.json \

-profile=kubernetes \

manager-csr.json | cfssljson -bare /etc/kubernetes/pki/controller-manager

# 注意，如果不是高可用集群，192.168.0.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

# set-cluster：设置一个集群项，

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/pki/ca.pem \

--embed-certs=true \

--server=https://192.168.0.236:8443 \

--kubeconfig=/etc/kubernetes/controller-manager.kubeconfig

# 设置一个环境项，一个上下文

kubectl config set-context system:kube-controller-manager@kubernetes \

--cluster=kubernetes \

--user=system:kube-controller-manager \

--kubeconfig=/etc/kubernetes/controller-manager.kubeconfig

# set-credentials 设置一个用户项

kubectl config set-credentials system:kube-controller-manager \

--client-certificate=/etc/kubernetes/pki/controller-manager.pem \

--client-key=/etc/kubernetes/pki/controller-manager-key.pem \

--embed-certs=true \

--kubeconfig=/etc/kubernetes/controller-manager.kubeconfig

# 使用某个环境当做默认环境

kubectl config use-context system:kube-controller-manager@kubernetes \

--kubeconfig=/etc/kubernetes/controller-manager.kubeconfig

cfssl gencert \

-ca=/etc/kubernetes/pki/ca.pem \

-ca-key=/etc/kubernetes/pki/ca-key.pem \

-config=ca-config.json \

-profile=kubernetes \

scheduler-csr.json | cfssljson -bare /etc/kubernetes/pki/scheduler

# 注意，如果不是高可用集群，192.168.0.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/pki/ca.pem \

--embed-certs=true \

--server=https://192.168.0.236:8443 \

--kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config set-credentials system:kube-scheduler \

--client-certificate=/etc/kubernetes/pki/scheduler.pem \

--client-key=/etc/kubernetes/pki/scheduler-key.pem \

--embed-certs=true \

--kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config set-context system:kube-scheduler@kubernetes \

--cluster=kubernetes \

--user=system:kube-scheduler \

--kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config use-context system:kube-scheduler@kubernetes \

--kubeconfig=/etc/kubernetes/scheduler.kubeconfig

cfssl gencert \

-ca=/etc/kubernetes/pki/ca.pem \

-ca-key=/etc/kubernetes/pki/ca-key.pem \

-config=ca-config.json \

-profile=kubernetes \

admin-csr.json | cfssljson -bare /etc/kubernetes/pki/admin

# 注意，如果不是高可用集群，192.168.0.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.pem --embed-certs=true --server=https://192.168.0.236:8443 --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config set-credentials kubernetes-admin --client-certificate=/etc/kubernetes/pki/admin.pem --client-key=/etc/kubernetes/pki/admin-key.pem --embed-certs=true --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config set-context kubernetes-admin@kubernetes --cluster=kubernetes --user=kubernetes-admin --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config use-context kubernetes-admin@kubernetes --kubeconfig=/etc/kubernetes/admin.kubeconfig

　　创建ServiceAccount Key 🡪 secret

openssl genrsa -out /etc/kubernetes/pki/sa.key 2048

返回结果

Generating RSA private key, 2048 bit long modulus (2 primes)

...................................................................................+++++

...............+++++

e is 65537 (0x010001)

openssl rsa -in /etc/kubernetes/pki/sa.key -pubout -out /etc/kubernetes/pki/sa.pub

发送证书至其他节点

for NODE in k8s-master02 k8s-master03; do   
for FILE in $(ls /etc/kubernetes/pki | grep -v etcd); do   
scp /etc/kubernetes/pki/${FILE} $NODE:/etc/kubernetes/pki/${FILE};  
done;   
for FILE in admin.kubeconfig controller-manager.kubeconfig scheduler.kubeconfig; do   
scp /etc/kubernetes/${FILE} $NODE:/etc/kubernetes/${FILE};  
done;  
done

查看证书文件

[root@k8s-master01 pki]# ls /etc/kubernetes/pki/

admin.csr apiserver.csr ca.csr controller-manager.csr front-proxy-ca.csr front-proxy-client.csr sa.key scheduler-key.pem

admin-key.pem apiserver-key.pem ca-key.pem controller-manager-key.pem front-proxy-ca-key.pem front-proxy-client-key.pem sa.pub scheduler.pem

admin.pem apiserver.pem ca.pem controller-manager.pem front-proxy-ca.pem front-proxy-client.pem scheduler.csr

[root@k8s-master01 pki]# ls /etc/kubernetes/pki/ |wc -l

23

# Kubernetes系统组件配置

## Etcd配置

etcd配置大致相同，注意修改每个Master节点的etcd配置的主机名和IP地址

### Master01

vim /etc/etcd/etcd.config.yml

name: 'k8s-master01'

data-dir: /var/lib/etcd

wal-dir: /var/lib/etcd/wal

snapshot-count: 5000

heartbeat-interval: 100

election-timeout: 1000

quota-backend-bytes: 0

listen-peer-urls: 'https://192.168.0.107:2380'

listen-client-urls: 'https://192.168.0.107:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.107:2380'

advertise-client-urls: 'https://192.168.0.107:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.107:2380,k8s-master02=https://192.168.0.108:2380,k8s-master03=https://192.168.0.109:2380'

initial-cluster-token: 'etcd-k8s-cluster'

initial-cluster-state: 'new'

strict-reconfig-check: false

enable-v2: true

enable-pprof: true

proxy: 'off'

proxy-failure-wait: 5000

proxy-refresh-interval: 30000

proxy-dial-timeout: 1000

proxy-write-timeout: 5000

proxy-read-timeout: 0

client-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

peer-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

peer-client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

debug: false

log-package-levels:

log-outputs: [default]

force-new-cluster: false

### Master02

vim /etc/etcd/etcd.config.yml

name: 'k8s-master02'

data-dir: /var/lib/etcd

wal-dir: /var/lib/etcd/wal

snapshot-count: 5000

heartbeat-interval: 100

election-timeout: 1000

quota-backend-bytes: 0

listen-peer-urls: 'https://192.168.0.108:2380'

listen-client-urls: 'https://192.168.0.108:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.108:2380'

advertise-client-urls: 'https://192.168.0.108:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.107:2380,k8s-master02=https://192.168.0.108:2380,k8s-master03=https://192.168.0.109:2380'

initial-cluster-token: 'etcd-k8s-cluster'

initial-cluster-state: 'new'

strict-reconfig-check: false

enable-v2: true

enable-pprof: true

proxy: 'off'

proxy-failure-wait: 5000

proxy-refresh-interval: 30000

proxy-dial-timeout: 1000

proxy-write-timeout: 5000

proxy-read-timeout: 0

client-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

peer-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

peer-client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

debug: false

log-package-levels:

log-outputs: [default]

force-new-cluster: false

### Master03

vim /etc/etcd/etcd.config.yml

name: 'k8s-master03'

data-dir: /var/lib/etcd

wal-dir: /var/lib/etcd/wal

snapshot-count: 5000

heartbeat-interval: 100

election-timeout: 1000

quota-backend-bytes: 0

listen-peer-urls: 'https://192.168.0.109:2380'

listen-client-urls: 'https://192.168.0.109:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.109:2380'

advertise-client-urls: 'https://192.168.0.109:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.107:2380,k8s-master02=https://192.168.0.108:2380,k8s-master03=https://192.168.0.109:2380'

initial-cluster-token: 'etcd-k8s-cluster'

initial-cluster-state: 'new'

strict-reconfig-check: false

enable-v2: true

enable-pprof: true

proxy: 'off'

proxy-failure-wait: 5000

proxy-refresh-interval: 30000

proxy-dial-timeout: 1000

proxy-write-timeout: 5000

proxy-read-timeout: 0

client-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

peer-transport-security:

cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'

key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'

peer-client-cert-auth: true

trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'

auto-tls: true

debug: false

log-package-levels:

log-outputs: [default]

force-new-cluster: false

### 创建Service

所有Master节点创建etcd service并启动

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

[Unit]

Description=Etcd Service

Documentation=https://coreos.com/etcd/docs/latest/

After=network.target

[Service]

Type=notify

ExecStart=/usr/local/bin/etcd --config-file=/etc/etcd/etcd.config.yml

Restart=on-failure

RestartSec=10

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

Alias=etcd3.service

所有Master节点创建etcd的证书目录

mkdir /etc/kubernetes/pki/etcd

ln -s /etc/etcd/ssl/\* /etc/kubernetes/pki/etcd/

systemctl daemon-reload

systemctl enable --now etcd

查看etcd状态

export ETCDCTL\_API=3

etcdctl --endpoints="192.168.0.109:2379,192.168.0.108:2379,192.168.0.107:2379" --cacert=/etc/kubernetes/pki/etcd/etcd-ca.pem --cert=/etc/kubernetes/pki/etcd/etcd.pem --key=/etc/kubernetes/pki/etcd/etcd-key.pem endpoint status --write-out=table



# 高可用配置

高可用配置（注意：如果不是高可用集群，haproxy和keepalived无需安装）

如果在云上安装也无需执行此章节的步骤，可以直接使用云上的lb，比如阿里云slb，腾讯云elb等

Slb -> haproxy -> apiserver

　　所有Master节点安装keepalived和haproxy

yum install keepalived haproxy -y

　　所有Master配置HAProxy，配置一样

vim /etc/haproxy/haproxy.cfg

global

maxconn 2000

ulimit-n 16384

log 127.0.0.1 local0 err

stats timeout 30s

defaults

log global

mode http

option httplog

timeout connect 5000

timeout client 50000

timeout server 50000

timeout http-request 15s

timeout http-keep-alive 15s

frontend k8s-master

bind 0.0.0.0:8443

bind 127.0.0.1:8443

mode tcp

option tcplog

tcp-request inspect-delay 5s

default\_backend k8s-master

backend k8s-master

mode tcp

option tcplog

option tcp-check

balance roundrobin

default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256 weight 100

server k8s-master01 192.168.0.107:6443 check

server k8s-master02 192.168.0.108:6443 check

server k8s-master03 192.168.0.109:6443 check

## Master01 keepalived

所有Master节点配置KeepAlived，配置不一样，注意区分 [root@k8s-master01 pki]# vim /etc/keepalived/keepalived.conf ，注意每个节点的IP和网卡（interface参数）

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

script "/etc/keepalived/check\_apiserver.sh"

interval 5

weight -5

fall 2

rise 1

}

vrrp\_instance VI\_1 {

state MASTER

interface ens33

mcast\_src\_ip 192.168.0.107

virtual\_router\_id 51

priority 101

nopreempt

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.236

}

track\_script {  
 chk\_apiserver

} }

## Master02 keepalived

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

script "/etc/keepalived/check\_apiserver.sh"

interval 5

weight -5

fall 2

rise 1

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

mcast\_src\_ip 192.168.0.108

virtual\_router\_id 51

priority 100

nopreempt

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.236

}

track\_script {  
 chk\_apiserver

} }

## Master03 keepalived

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

script "/etc/keepalived/check\_apiserver.sh"

interval 5

weight -5

fall 2

rise 1

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

mcast\_src\_ip 192.168.0.109

virtual\_router\_id 51

priority 100

nopreempt

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.236

}

track\_script {  
 chk\_apiserver

} }

## 健康检查配置

所有master节点

[root@k8s-master01 keepalived]# cat /etc/keepalived/check\_apiserver.sh

#!/bin/bash

err=0

for k in $(seq 1 3)

do

check\_code=$(pgrep haproxy)

if [[ $check\_code == "" ]]; then

err=$(expr $err + 1)

sleep 1

continue

else

err=0

break

fi

done

if [[ $err != "0" ]]; then

echo "systemctl stop keepalived"

/usr/bin/systemctl stop keepalived

exit 1

else

exit 0

fi

chmod +x /etc/keepalived/check\_apiserver.sh

所有master节点启动haproxy和keepalived

[root@k8s-master01 keepalived]# systemctl daemon-reload

[root@k8s-master01 keepalived]# systemctl enable --now haproxy

[root@k8s-master01 keepalived]# systemctl enable --now keepalived

　　VIP测试

[root@k8s-master01 pki]# ping 192.168.0.236

PING 192.168.0.236 (192.168.0.236) 56(84) bytes of data.

64 bytes from 192.168.0.236: icmp\_seq=1 ttl=64 time=1.39 ms

64 bytes from 192.168.0.236: icmp\_seq=2 ttl=64 time=2.46 ms

64 bytes from 192.168.0.236: icmp\_seq=3 ttl=64 time=1.68 ms

64 bytes from 192.168.0.236: icmp\_seq=4 ttl=64 time=1.08 ms

重要：如果安装了keepalived和haproxy，需要测试keepalived是否是正常的

telnet 192.168.0.236 8443

如果ping不通且telnet没有出现 ]，则认为VIP不可以，不可在继续往下执行，需要排查keepalived的问题，比如防火墙和selinux，haproxy和keepalived的状态，监听端口等

所有节点查看防火墙状态必须为disable和inactive：systemctl status firewalld

所有节点查看selinux状态，必须为disable：getenforce

master节点查看haproxy和keepalived状态：systemctl status keepalived haproxy

master节点查看监听端口：netstat -lntp

# Kubernetes组件配置

　　所有节点创建相关目录

mkdir -p /etc/kubernetes/manifests/ /etc/systemd/system/kubelet.service.d /var/lib/kubelet /var/log/kubernetes

## Apiserver

所有Master节点创建kube-apiserver service，# 注意，如果不是高可用集群，192.168.0.236改为master01的地址

#### Master01配置

注意本文档使用的k8s service网段为10.96.0.0/12，该网段不能和宿主机的网段、Pod网段的重复，请按需修改

[root@k8s-master01 pki]# cat /usr/lib/systemd/system/kube-apiserver.service

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/usr/local/bin/kube-apiserver \

--v=2 \

--logtostderr=true \

--allow-privileged=true \

--bind-address=0.0.0.0 \

--secure-port=6443 \

--insecure-port=0 \

--advertise-address=192.168.0.107 \

--service-cluster-ip-range=10.96.0.0/12 \

--service-node-port-range=30000-32767 \

--etcd-servers=https://192.168.0.107:2379,https://192.168.0.108:2379,https://192.168.0.109:2379 \

--etcd-cafile=/etc/etcd/ssl/etcd-ca.pem \

--etcd-certfile=/etc/etcd/ssl/etcd.pem \

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

--client-ca-file=/etc/kubernetes/pki/ca.pem \

--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \

--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \

--kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem \

--kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem \

--service-account-key-file=/etc/kubernetes/pki/sa.pub \

--service-account-signing-key-file=/etc/kubernetes/pki/sa.key \

--service-account-issuer=https://kubernetes.default.svc.cluster.local \

--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname \

--enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota \

--authorization-mode=Node,RBAC \

--enable-bootstrap-token-auth=true \

--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem \

--proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem \

--proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem \

--requestheader-allowed-names=aggregator \

--requestheader-group-headers=X-Remote-Group \

--requestheader-extra-headers-prefix=X-Remote-Extra- \

--requestheader-username-headers=X-Remote-User

# --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure

RestartSec=10s

LimitNOFILE=65535

[Install]

WantedBy=multi-user.target

#### Master02配置

注意本文档使用的k8s service网段为10.96.0.0/12，该网段不能和宿主机的网段、Pod网段的重复，请按需修改

[root@k8s-master01 pki]# cat /usr/lib/systemd/system/kube-apiserver.service

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/usr/local/bin/kube-apiserver \

--v=2 \

--logtostderr=true \

--allow-privileged=true \

--bind-address=0.0.0.0 \

--secure-port=6443 \

--insecure-port=0 \

--advertise-address=192.168.0.108 \

--service-cluster-ip-range=10.96.0.0/12 \

--service-node-port-range=30000-32767 \

--etcd-servers=https://192.168.0.107:2379,https://192.168.0.108:2379,https://192.168.0.109:2379 \

--etcd-cafile=/etc/etcd/ssl/etcd-ca.pem \

--etcd-certfile=/etc/etcd/ssl/etcd.pem \

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

--client-ca-file=/etc/kubernetes/pki/ca.pem \

--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \

--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \

--kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem \

--kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem \

--service-account-key-file=/etc/kubernetes/pki/sa.pub \

--service-account-signing-key-file=/etc/kubernetes/pki/sa.key \

--service-account-issuer=https://kubernetes.default.svc.cluster.local \

--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname \

--enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota \

--authorization-mode=Node,RBAC \

--enable-bootstrap-token-auth=true \

--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem \

--proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem \

--proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem \

--requestheader-allowed-names=aggregator \

--requestheader-group-headers=X-Remote-Group \

--requestheader-extra-headers-prefix=X-Remote-Extra- \

--requestheader-username-headers=X-Remote-User

# --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure

RestartSec=10s

LimitNOFILE=65535

[Install]

WantedBy=multi-user.target

#### Master03配置

注意本文档使用的k8s service网段为10.96.0.0/12，该网段不能和宿主机的网段、Pod网段的重复，请按需修改

[root@k8s-master01 pki]# cat /usr/lib/systemd/system/kube-apiserver.service

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/usr/local/bin/kube-apiserver \

--v=2 \

--logtostderr=true \

--allow-privileged=true \

--bind-address=0.0.0.0 \

--secure-port=6443 \

--insecure-port=0 \

--advertise-address=192.168.0.109 \

--service-cluster-ip-range=10.96.0.0/12 \

--service-node-port-range=30000-32767 \

--etcd-servers=https://192.168.0.107:2379,https://192.168.0.108:2379,https://192.168.0.109:2379 \

--etcd-cafile=/etc/etcd/ssl/etcd-ca.pem \

--etcd-certfile=/etc/etcd/ssl/etcd.pem \

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

--client-ca-file=/etc/kubernetes/pki/ca.pem \

--tls-cert-file=/etc/kubernetes/pki/apiserver.pem \

--tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem \

--kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem \

--kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem \

--service-account-key-file=/etc/kubernetes/pki/sa.pub \

--service-account-signing-key-file=/etc/kubernetes/pki/sa.key \

--service-account-issuer=https://kubernetes.default.svc.cluster.local \

--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname \

--enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota \

--authorization-mode=Node,RBAC \

--enable-bootstrap-token-auth=true \

--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem \

--proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem \

--proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem \

--requestheader-allowed-names=aggregator \

--requestheader-group-headers=X-Remote-Group \

--requestheader-extra-headers-prefix=X-Remote-Extra- \

--requestheader-username-headers=X-Remote-User

# --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure

RestartSec=10s

LimitNOFILE=65535

[Install]

WantedBy=multi-user.target

#### 启动apiserver

所有Master节点开启kube-apiserver

systemctl daemon-reload && systemctl enable --now kube-apiserver

检测kube-server状态

# systemctl status kube-apiserver

● kube-apiserver.service - Kubernetes API Server

Loaded: loaded (/usr/lib/systemd/system/kube-apiserver.service; enabled; vendor preset: disabled)

Active: active (running) since Sat 2020-08-22 21:26:49 CST; 26s ago

 系统日志的这些提示可以忽略

Dec 11 20:51:15 k8s-master01 kube-apiserver: I1211 20:51:15.004739 7450 clientconn.go:948] ClientConn switching balancer to "pick\_first"

Dec 11 20:51:15 k8s-master01 kube-apiserver: I1211 20:51:15.004843 7450 balancer\_conn\_wrappers.go:78] pickfirstBalancer: HandleSubConnStateChange: 0xc011bd4c80, {CONNECTING <nil>}

Dec 11 20:51:15 k8s-master01 kube-apiserver: I1211 20:51:15.010725 7450 balancer\_conn\_wrappers.go:78] pickfirstBalancer: HandleSubConnStateChange: 0xc011bd4c80, {READY <nil>}

Dec 11 20:51:15 k8s-master01 kube-apiserver: I1211 20:51:15.011370 7450 controlbuf.go:508] transport: loopyWriter.run returning. connection error: desc = "transport is closing"

## ControllerManager

所有Master节点配置kube-controller-manager service

注意本文档使用的k8s Pod网段为172.16.0.0/12，该网段不能和宿主机的网段、k8s Service网段的重复，请按需修改

[root@k8s-master01 pki]# cat /usr/lib/systemd/system/kube-controller-manager.service

[Unit]

Description=Kubernetes Controller Manager

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

After=network.target

[Service]

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

--v=2 \

--logtostderr=true \

--address=127.0.0.1 \

--root-ca-file=/etc/kubernetes/pki/ca.pem \

--cluster-signing-cert-file=/etc/kubernetes/pki/ca.pem \

--cluster-signing-key-file=/etc/kubernetes/pki/ca-key.pem \

--service-account-private-key-file=/etc/kubernetes/pki/sa.key \

--kubeconfig=/etc/kubernetes/controller-manager.kubeconfig \

--leader-elect=true \

--use-service-account-credentials=true \

--node-monitor-grace-period=40s \

--node-monitor-period=5s \

--pod-eviction-timeout=2m0s \

--controllers=\*,bootstrapsigner,tokencleaner \

--allocate-node-cidrs=true \

--cluster-cidr=172.16.0.0/12 \

--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem \

--node-cidr-mask-size=24

Restart=always

RestartSec=10s

[Install]

WantedBy=multi-user.target

　　所有Master节点启动kube-controller-manager

[root@k8s-master01 pki]# systemctl daemon-reload

[root@k8s-master01 pki]# systemctl enable --now kube-controller-manager

Created symlink /etc/systemd/system/multi-user.target.wants/kube-controller-manager.service → /usr/lib/systemd/system/kube-controller-manager.service.

查看启动状态

[root@k8s-master01 pki]# systemctl enable --now kube-controller-manager

Created symlink from /etc/systemd/system/multi-user.target.wants/kube-controller-manager.service to /usr/lib/systemd/system/kube-controller-manager.service.

[root@k8s-master01 pki]# systemctl status kube-controller-manager

● kube-controller-manager.service - Kubernetes Controller Manager

Loaded: loaded (/usr/lib/systemd/system/kube-controller-manager.service; enabled; vendor preset: disabled)

Active: active (running) since Fri 2020-12-11 20:53:05 CST; 8s ago

Docs: https://github.com/kubernetes/kubernetes

Main PID: 7518 (kube-controller)

## Scheduler

　　所有Master节点配置kube-scheduler service

[root@k8s-master01 pki]# cat /usr/lib/systemd/system/kube-scheduler.service

[Unit]

Description=Kubernetes Scheduler

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

After=network.target

[Service]

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

--v=2 \

--logtostderr=true \

--address=127.0.0.1 \

--leader-elect=true \

--kubeconfig=/etc/kubernetes/scheduler.kubeconfig

Restart=always

RestartSec=10s

[Install]

WantedBy=multi-user.target

[root@k8s-master01 pki]# systemctl daemon-reload

[root@k8s-master01 pki]# systemctl enable --now kube-scheduler

Created symlink /etc/systemd/system/multi-user.target.wants/kube-scheduler.service → /usr/lib/systemd/system/kube-scheduler.service.

# TLS Bootstrapping配置

在Master01创建bootstrap

# 注意，如果不是高可用集群，192.168.0.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

cd /root/k8s-ha-install/bootstrap

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.pem --embed-certs=true --server=https://192.168.0.236:8443 --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config set-credentials tls-bootstrap-token-user --token=c8ad9c.2e4d610cf3e7426e --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config set-context tls-bootstrap-token-user@kubernetes --cluster=kubernetes --user=tls-bootstrap-token-user --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config use-context tls-bootstrap-token-user@kubernetes --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

注意：如果要修改bootstrap.secret.yaml的token-id和token-secret，需要保证下图红圈内的字符串一致的，并且位数是一样的。还要保证上个命令的黄色字体：c8ad9c.2e4d610cf3e7426e与你修改的字符串要一致



[root@k8s-master01 bootstrap]# mkdir -p /root/.kube ; cp /etc/kubernetes/admin.kubeconfig /root/.kube/config

[root@k8s-master01 bootstrap]# kubectl create -f bootstrap.secret.yaml

secret/bootstrap-token-c8ad9c created

clusterrolebinding.rbac.authorization.k8s.io/kubelet-bootstrap created

clusterrolebinding.rbac.authorization.k8s.io/node-autoapprove-bootstrap created

clusterrolebinding.rbac.authorization.k8s.io/node-autoapprove-certificate-rotation created

clusterrole.rbac.authorization.k8s.io/system:kube-apiserver-to-kubelet created

clusterrolebinding.rbac.authorization.k8s.io/system:kube-apiserver created

# Node节点配置

## 复制证书

　　Master01节点复制证书至Node节点

cd /etc/kubernetes/

for NODE in k8s-master02 k8s-master03 k8s-node01 k8s-node02; do

ssh $NODE mkdir -p /etc/kubernetes/pki /etc/etcd/ssl /etc/etcd/ssl

for FILE in etcd-ca.pem etcd.pem etcd-key.pem; do

scp /etc/etcd/ssl/$FILE $NODE:/etc/etcd/ssl/

done

for FILE in pki/ca.pem pki/ca-key.pem pki/front-proxy-ca.pem bootstrap-kubelet.kubeconfig; do

scp /etc/kubernetes/$FILE $NODE:/etc/kubernetes/${FILE}

done

done

执行结果：

etcd-ca.pem 100% 1363 314.0KB/s 00:00

etcd.pem 100% 1505 429.1KB/s 00:00

etcd-key.pem 100% 1679 361.9KB/s 00:00

ca.pem 100% 1407 459.5KB/s 00:00

ca-key.pem 100% 1679 475.2KB/s 00:00

front-proxy-ca.pem 100% 1143 214.5KB/s 00:00

bootstrap-kubelet.kubeconfig 100% 2291 695.1KB/s 00:00

etcd-ca.pem 100% 1363 325.5KB/s 00:00

etcd.pem 100% 1505 301.2KB/s 00:00

etcd-key.pem 100% 1679 260.9KB/s 00:00

ca.pem 100% 1407 420.8KB/s 00:00

ca-key.pem 100% 1679 398.0KB/s 00:00

front-proxy-ca.pem 100% 1143 224.9KB/s 00:00

bootstrap-kubelet.kubeconfig 100% 2291 685.4KB/s 00:00

## Kubelet配置

　　所有节点创建相关目录

mkdir -p /var/lib/kubelet /var/log/kubernetes /etc/systemd/system/kubelet.service.d /etc/kubernetes/manifests/

　　所有节点配置kubelet service

[root@k8s-master01 bootstrap]# vim /usr/lib/systemd/system/kubelet.service

[Unit]

Description=Kubernetes Kubelet

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

After=docker.service

Requires=docker.service

[Service]

ExecStart=/usr/local/bin/kubelet

Restart=always

StartLimitInterval=0

RestartSec=10

[Install]

WantedBy=multi-user.target

所有节点配置kubelet service的配置文件

vim /etc/systemd/system/kubelet.service.d/10-kubelet.conf

[Service]

Environment="KUBELET\_KUBECONFIG\_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig"

Environment="KUBELET\_SYSTEM\_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"

Environment="KUBELET\_CONFIG\_ARGS=--config=/etc/kubernetes/kubelet-conf.yml --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google\_containers/pause-amd64:3.2"

Environment="KUBELET\_EXTRA\_ARGS=--node-labels=node.kubernetes.io/node='' "

ExecStart=

ExecStart=/usr/local/bin/kubelet $KUBELET\_KUBECONFIG\_ARGS $KUBELET\_CONFIG\_ARGS $KUBELET\_SYSTEM\_ARGS $KUBELET\_EXTRA\_ARGS

创建kubelet的配置文件

注意：如果更改了k8s的service网段，需要更改kubelet-conf.yml 的clusterDNS:配置，改成k8s Service网段的第十个地址，比如10.96.0.10

[root@k8s-master01 bootstrap]# vim /etc/kubernetes/kubelet-conf.yml

apiVersion: kubelet.config.k8s.io/v1beta1

kind: KubeletConfiguration

address: 0.0.0.0

port: 10250

readOnlyPort: 10255

authentication:

anonymous:

enabled: false

webhook:

cacheTTL: 2m0s

enabled: true

x509:

clientCAFile: /etc/kubernetes/pki/ca.pem

authorization:

mode: Webhook

webhook:

cacheAuthorizedTTL: 5m0s

cacheUnauthorizedTTL: 30s

cgroupDriver: systemd

cgroupsPerQOS: true

clusterDNS:

- 10.96.0.10

clusterDomain: cluster.local

containerLogMaxFiles: 5

containerLogMaxSize: 10Mi

contentType: application/vnd.kubernetes.protobuf

cpuCFSQuota: true

cpuManagerPolicy: none

cpuManagerReconcilePeriod: 10s

enableControllerAttachDetach: true

enableDebuggingHandlers: true

enforceNodeAllocatable:

- pods

eventBurst: 10

eventRecordQPS: 5

evictionHard:

imagefs.available: 15%

memory.available: 100Mi

nodefs.available: 10%

nodefs.inodesFree: 5%

evictionPressureTransitionPeriod: 5m0s

failSwapOn: true

fileCheckFrequency: 20s

hairpinMode: promiscuous-bridge

healthzBindAddress: 127.0.0.1

healthzPort: 10248

httpCheckFrequency: 20s

imageGCHighThresholdPercent: 85

imageGCLowThresholdPercent: 80

imageMinimumGCAge: 2m0s

iptablesDropBit: 15

iptablesMasqueradeBit: 14

kubeAPIBurst: 10

kubeAPIQPS: 5

makeIPTablesUtilChains: true

maxOpenFiles: 1000000

maxPods: 110

nodeStatusUpdateFrequency: 10s

oomScoreAdj: -999

podPidsLimit: -1

registryBurst: 10

registryPullQPS: 5

resolvConf: /etc/resolv.conf

rotateCertificates: true

runtimeRequestTimeout: 2m0s

serializeImagePulls: true

staticPodPath: /etc/kubernetes/manifests

streamingConnectionIdleTimeout: 4h0m0s

syncFrequency: 1m0s

volumeStatsAggPeriod: 1m0s

　　启动所有节点kubelet

systemctl daemon-reload

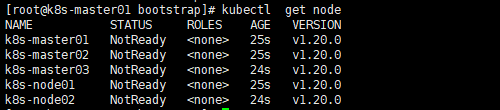
systemctl enable --now kubelet

此时系统日志/var/log/messages

Unable to update cni config: no networks found in /etc/cni/net.d 显示只有如下信息为正常

　　查看集群状态

[root@k8s-master01 bootstrap]# kubectl get node



## kube-proxy配置

# 注意，如果不是高可用集群，192.168.0.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

以下操作在Master01执行

cd /root/k8s-ha-install  
kubectl -n kube-system create serviceaccount kube-proxy

kubectl create clusterrolebinding system:kube-proxy --clusterrole system:node-proxier --serviceaccount kube-system:kube-proxy

SECRET=$(kubectl -n kube-system get sa/kube-proxy \

--output=jsonpath='{.secrets[0].name}')

JWT\_TOKEN=$(kubectl -n kube-system get secret/$SECRET \

--output=jsonpath='{.data.token}' | base64 -d)

PKI\_DIR=/etc/kubernetes/pki

K8S\_DIR=/etc/kubernetes

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.pem --embed-certs=true --server=https://192.168.0.236:8443 --kubeconfig=${K8S\_DIR}/kube-proxy.kubeconfig

kubectl config set-credentials kubernetes --token=${JWT\_TOKEN} --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig

kubectl config set-context kubernetes --cluster=kubernetes --user=kubernetes --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig

kubectl config use-context kubernetes --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig

在master01将kube-proxy的systemd Service文件发送到其他节点

如果更改了集群Pod的网段，需要更改kube-proxy/kube-proxy.conf的clusterCIDR: 172.16.0.0/12参数为pod的网段。

for NODE in k8s-master01 k8s-master02 k8s-master03; do

scp ${K8S\_DIR}/kube-proxy.kubeconfig $NODE:/etc/kubernetes/kube-proxy.kubeconfig

scp kube-proxy/kube-proxy.conf $NODE:/etc/kubernetes/kube-proxy.conf

scp kube-proxy/kube-proxy.service $NODE:/usr/lib/systemd/system/kube-proxy.service

done

for NODE in k8s-node01 k8s-node02; do

scp /etc/kubernetes/kube-proxy.kubeconfig $NODE:/etc/kubernetes/kube-proxy.kubeconfig

scp kube-proxy/kube-proxy.conf $NODE:/etc/kubernetes/kube-proxy.conf

scp kube-proxy/kube-proxy.service $NODE:/usr/lib/systemd/system/kube-proxy.service

done

　　所有节点启动kube-proxy

[root@k8s-master01 k8s-ha-install]# systemctl daemon-reload

[root@k8s-master01 k8s-ha-install]# systemctl enable --now kube-proxy

Created symlink /etc/systemd/system/multi-user.target.wants/kube-proxy.service → /usr/lib/systemd/system/kube-proxy.service.

# 安装Calico

Calico的安装请必须听视频课程和最后一章升级Calico的视频

以下步骤只在master01执行

cd /root/k8s-ha-install/calico/

修改calico-etcd.yaml的以下位置

sed -i 's#etcd\_endpoints: "http://<ETCD\_IP>:<ETCD\_PORT>"#etcd\_endpoints: "https://192.168.0.107:2379,https://192.168.0.108:2379,https://192.168.0.109:2379"#g' calico-etcd.yaml

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

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

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

sed -i "s@# etcd-key: null@etcd-key: ${ETCD\_KEY}@g; s@# etcd-cert: null@etcd-cert: ${ETCD\_CERT}@g; s@# etcd-ca: null@etcd-ca: ${ETCD\_CA}@g" calico-etcd.yaml

sed -i 's#etcd\_ca: ""#etcd\_ca: "/calico-secrets/etcd-ca"#g; s#etcd\_cert: ""#etcd\_cert: "/calico-secrets/etcd-cert"#g; s#etcd\_key: "" #etcd\_key: "/calico-secrets/etcd-key" #g' calico-etcd.yaml

# 更改此处为自己的pod网段

POD\_SUBNET="172.16.0.0/12"

sed -i 's@# - name: CALICO\_IPV4POOL\_CIDR@- name: CALICO\_IPV4POOL\_CIDR@g; s@# value: "192.168.0.0/16"@ value: '"${POD\_SUBNET}"'@g' calico-etcd.yaml

kubectl apply -f calico-etcd.yaml

查看容器状态

[root@k8s-master01 calico]# kubectl get po -n kube-system

如果容器状态异常可以使用kubectl describe 或者logs查看容器的日志

# 安装CoreDNS

## 安装对应版本（推荐）

cd /root/k8s-ha-install/

如果更改了k8s service的网段需要将coredns的serviceIP改成k8s service网段的第十个IP

sed -i "s#10.96.0.10#10.96.0.10#g" CoreDNS/coredns.yaml

安装coredns

[root@k8s-master01 k8s-ha-install]# kubectl create -f CoreDNS/coredns.yaml

serviceaccount/coredns created

clusterrole.rbac.authorization.k8s.io/system:coredns created

clusterrolebinding.rbac.authorization.k8s.io/system:coredns created

configmap/coredns created

deployment.apps/coredns created

service/kube-dns created

## 安装最新版CoreDNS

git clone <https://github.com/coredns/deployment.git>

cd deployment/kubernetes

# ./deploy.sh -s -i 10.96.0.10 | kubectl apply -f -

serviceaccount/coredns created

clusterrole.rbac.authorization.k8s.io/system:coredns created

clusterrolebinding.rbac.authorization.k8s.io/system:coredns created

configmap/coredns created

deployment.apps/coredns created

service/kube-dns created

查看状态

 # kubectl get po -n kube-system -l k8s-app=kube-dns

NAME READY STATUS RESTARTS AGE

coredns-85b4878f78-h29kh 1/1 Running 0 8h

# 安装Metrics Server

在新版的Kubernetes中系统资源的采集均使用Metrics-server，可以通过Metrics采集节点和Pod的内存、磁盘、CPU和网络的使用率。

安装metrics server

cd /root/k8s-ha-install/metrics-server-0.4.x/

kubectl create -f .

serviceaccount/metrics-server created

clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created

clusterrole.rbac.authorization.k8s.io/system:metrics-server created

rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created

clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created

clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created

service/metrics-server created

deployment.apps/metrics-server created

apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created

等待metrics server启动然后查看状态

[root@k8s-master01 metrics-server-0.4.x]# kubectl top node

NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

k8s-master01 231m 5% 1620Mi 42%

k8s-master02 274m 6% 1203Mi 31%

k8s-master03 202m 5% 1251Mi 32%

k8s-node01 69m 1% 667Mi 17%

k8s-node02 73m 1% 650Mi 16%

# 集群验证

 集群验证请参考视频的集群验证，必须要做！！！

　　安装busybox

cat<<EOF | kubectl apply -f -

apiVersion: v1

kind: Pod

metadata:

name: busybox

namespace: default

spec:

containers:

- name: busybox

image: busybox:1.28

command:

- sleep

- "3600"

imagePullPolicy: IfNotPresent

restartPolicy: Always

EOF

1. Pod必须能解析Service
2. Pod必须能解析跨namespace的Service
3. 每个节点都必须要能访问Kubernetes的kubernetes svc 443和kube-dns的service 53
4. Pod和Pod之前要能通
   1. 同namespace能通信
   2. 跨namespace能通信
   3. 跨机器能通信

　　验证解析（请参考视频集群验证）

[root@k8s-master01 CoreDNS]# kubectl exec busybox -n default -- nslookup kubernetes

Server: 192.168.0.10

Address 1: 192.168.0.10 kube-dns.kube-system.svc.cluster.local

Name: kubernetes

Address 1: 192.168.0.1 kubernetes.default.svc.cluster.local

[root@k8s-master01 CoreDNS]# kubectl exec busybox -n default -- nslookup kube-dns.kube-system

Server: 192.168.0.10

Address 1: 192.168.0.10 kube-dns.kube-system.svc.cluster.local

Name: kube-dns.kube-system

Address 1: 192.168.0.10 kube-dns.kube-system.svc.cluster.local

# 安装dashboard

## Dashboard部署

Dashboard用于展示集群中的各类资源，同时也可以通过Dashboard实时查看Pod的日志和在容器中执行一些命令等。

### 安装指定版本dashboard

cd /root/k8s-ha-install/dashboard/

[root@k8s-master01 dashboard]# kubectl create -f .

serviceaccount/admin-user created

clusterrolebinding.rbac.authorization.k8s.io/admin-user created

namespace/kubernetes-dashboard created

serviceaccount/kubernetes-dashboard created

service/kubernetes-dashboard created

secret/kubernetes-dashboard-certs created

secret/kubernetes-dashboard-csrf created

secret/kubernetes-dashboard-key-holder created

configmap/kubernetes-dashboard-settings created

role.rbac.authorization.k8s.io/kubernetes-dashboard created

clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created

rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created

clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created

deployment.apps/kubernetes-dashboard created

service/dashboard-metrics-scraper created

deployment.apps/dashboard-metrics-scraper created

### 安装最新版

官方GitHub地址：<https://github.com/kubernetes/dashboard>

可以在官方dashboard查看到最新版dashboard



kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml>

创建管理员用户vim admin.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

annotations:

rbac.authorization.kubernetes.io/autoupdate: "true"

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

kubectl apply -f admin.yaml -n kube-system

### 登录dashboard

在谷歌浏览器（Chrome）启动文件中加入启动参数，用于解决无法访问Dashboard的问题，参考图1-1：

--test-type --ignore-certificate-errors



图1-1 谷歌浏览器 Chrome的配置

更改dashboard的svc为NodePort：

kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard



将ClusterIP更改为NodePort（如果已经为NodePort忽略此步骤）：

查看端口号：



根据自己的实例端口号，通过任意安装了kube-proxy的宿主机或者VIP的IP+端口即可访问到dashboard：

访问Dashboard：[https://192.168.0.236:18282（请更改18282为自己的端口）](https://192.168.0.200:18282（请更改18282为自己的端口）)，选择登录方式为令牌（即token方式），参考图1-2



图1-2 Dashboard登录方式

查看token值：

[root@k8s-master01 1.1.1]# kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

Name: admin-user-token-r4vcp

Namespace: kube-system

Labels: <none>

Annotations: kubernetes.io/service-account.name: admin-user

kubernetes.io/service-account.uid: 2112796c-1c9e-11e9-91ab-000c298bf023

Type: kubernetes.io/service-account-token

Data

====

ca.crt: 1025 bytes

namespace: 11 bytes

token: **eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLXI0dmNwIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiIyMTEyNzk2Yy0xYzllLTExZTktOTFhYi0wMDBjMjk4YmYwMjMiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.bWYmwgRb-90ydQmyjkbjJjFt8CdO8u6zxVZh-19rdlL\_T-n35nKyQIN7hCtNAt46u6gfJ5XXefC9HsGNBHtvo\_Ve6oF7EXhU772aLAbXWkU1xOwQTQynixaypbRIas\_kiO2MHHxXfeeL\_yYZRrgtatsDBxcBRg-nUQv4TahzaGSyK42E\_4YGpLa3X3Jc4t1z0SQXge7lrwlj8ysmqgO4ndlFjwPfvg0eoYqu9Qsc5Q7tazzFf9mVKMmcS1ppPutdyqNYWL62P1prw\_wclP0TezW1CsypjWSVT4AuJU8YmH8nTNR1EXn8mJURLSjINv6YbZpnhBIPgUGk1JYVLcn47w**

将token值输入到令牌后，单击登录即可访问Dashboard，参考图1-3：



图1-3 Dashboard页面

# 生产环境关键性配置

关键性配置请参考视频，不要直接配置！

vim /etc/docker/daemon.json

{ "registry-mirrors": [

"https://registry.docker-cn.com",

"http://hub-mirror.c.163.com",

"https://docker.mirrors.ustc.edu.cn"

],

"exec-opts": ["native.cgroupdriver=systemd"],

"max-concurrent-downloads": 10, "max-concurrent-uploads": 5, "log-opts": { "max-size": "300m", "max-file": "2" }, "live-restore": true} **vim /usr/lib/systemd/system/kube-controller-manager.service**

# --feature-gates=RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true \

--experimental-cluster-signing-duration=876000h0m0s \

**vim /etc/systemd/system/kubelet.service.d/10-kubelet.conf**

[Service]

Environment="KUBELET\_KUBECONFIG\_ARGS=--kubeconfig=/etc/kubernetes/kubelet.kubeconfig --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig"

Environment="KUBELET\_SYSTEM\_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"

Environment="KUBELET\_CONFIG\_ARGS=--config=/etc/kubernetes/kubelet-conf.yml --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google\_containers/pause-amd64:3.1"

Environment="KUBELET\_EXTRA\_ARGS=--tls-cipher-suites=TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256,TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 --image-pull-progress-deadline=30m

ExecStart=

ExecStart=/usr/local/bin/kubelet $KUBELET\_KUBECONFIG\_ARGS $KUBELET\_CONFIG\_ARGS $KUBELET\_SYSTEM\_ARGS $KUBELET\_EXTRA\_ARGS

关键性配置请参考视频，不要直接配置，可能会造成集群故障！！

vim **/etc/kubernetes/kubelet-conf.yml**

添加如下配置

rotateServerCertificates: true

allowedUnsafeSysctls:

- "net.core\*"

- "net.ipv4.\*"

kubeReserved:

cpu: "1"

memory: 1Gi

ephemeral-storage: 10Gi

systemReserved:

cpu: "1"

memory: 1Gi

ephemeral-storage: 10Gi

安装总结：

1. kubeadm
2. 二进制
3. 自动化安装
   1. Ansible
      1. Master节点安装不需要写自动化。
      2. 添加Node节点，playbook。
4. 安装需要注意的细节
   1. 上面的细节配置
   2. 生产环境中etcd一定要和系统盘分开，一定要用ssd硬盘。
   3. Docker数据盘也要和系统盘分开，有条件的话可以使用ssd硬盘