**本文档适用于k8s 1.17 1.18 1.19**

**请不要使用带中文的服务器和克隆的虚拟机**

# 基本说明

 　　本文章将演示CentOS 8二进制方式安装高可用k8s 1.19.x，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。

# 2. 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

192.168.0.201 k8s-master01 # 2C2G 40G

192.168.0.202 k8s-master02 # 2C2G 40G

192.168.0.203 k8s-master03 # 2C2G 40G

192.168.0.211 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

192.168.0.204 k8s-node01 # 2C2G 40G

192.168.0.205 k8s-node02  # 2C2G 40G

系统环境：

# cat /etc/redhat-release

CentOS Linux release 8.0.1905 (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

192.168.0.201 k8s-master01

192.168.0.202 k8s-master02

192.168.0.203 k8s-master03

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

192.168.0.204 k8s-node01

192.168.0.205 k8s-node02

　　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 # 无dnsmasq可忽略

setenforce 0

cat /etc/sysconfig/selinux

# This file controls the state of SELinux on the system.

# SELINUX= can take one of these three values:

# enforcing - SELinux security policy is enforced.

# permissive - SELinux prints warnings instead of enforcing.

# disabled - No SELinux policy is loaded.

SELINUX=disabled

所有节点关闭swap分区，fstab注释swap

[root@k8s-master01 ~]# swapoff -a && sysctl -w vm.swappiness=0

vm.swappiness = 0

[root@k8s-master01 ~]# vi /etc/fstab

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

#

# /etc/fstab

# Created by anaconda on Fri Nov 1 23:02:53 2019

#

# Accessible filesystems, by reference, are maintained under '/dev/disk/'.

# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.

#

# After editing this file, run 'systemctl daemon-reload' to update systemd

# units generated from this file.

#

/dev/mapper/cl-root / xfs defaults 0 0

UUID=6897cd7b-9b3a-42b0-a827-57991141b297 /boot ext4 defaults 1 2

#/dev/mapper/cl-swap swap swap defaults 0 0

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 wntp -y

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

echo 'Asia/Shanghai' > /etc/timezone

ntpdate time2.aliyun.com

设置时间定期执行

crontab -e

\*/1 \* \* \* \* ntpdate time2.aliyun.com

　　Master01节点生成ssh key

[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 ~]# 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.

　　切换到1.19.x分支（其他版本可以切换到其他分支）

cd k8s-ha-install && git checkout manual-installation-v1.19.x

所有节点配置，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>

所有节点，CentOS 8 安装源如下：

curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-8.repo

yum makecache

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>

# 3. 基本组件安装

## 3.1 系统升级

无论是CentOS7和CentOS8都需要升级

# cat /etc/redhat-release

CentOS Linux release 8.0.1905 (Core)

# yum update -y

# cat /etc/redhat-release

CentOS Linux release 8.2.2004 (Core)

## 3.2 内核升级

如需内核升级，可以按需操作

所有节点 CentOS7必须升级 ，默认内核版本是3.10，升级内核4.18+

使用如下方式安装最新版内核

rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org

rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm

查看最新版内核yum --disablerepo="\*" --enablerepo="elrepo-kernel" list available

[root@k8s-node01 ~]# yum --disablerepo="\*" --enablerepo="elrepo-kernel" list available

Loaded plugins: fastestmirror

Loading mirror speeds from cached hostfile

\* elrepo-kernel: mirrors.neusoft.edu.cn

elrepo-kernel | 2.9 kB 00:00:00

elrepo-kernel/primary\_db | 1.9 MB 00:00:00

Available Packages

elrepo-release.noarch 7.0-5.el7.elrepo elrepo-kernel

kernel-lt.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-devel.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-doc.noarch 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-headers.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-tools.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-tools-libs.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-lt-tools-libs-devel.x86\_64 4.4.229-1.el7.elrepo elrepo-kernel

kernel-ml.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-devel.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-doc.noarch 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-headers.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-tools.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-tools-libs.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

kernel-ml-tools-libs-devel.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

perf.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

python-perf.x86\_64 5.7.7-1.el7.elrepo elrepo-kernel

安装最新版：

yum --enablerepo=elrepo-kernel install kernel-ml kernel-ml-devel -y

安装完成后reboot

更改内核顺序：

grub2-set-default 0 && grub2-mkconfig -o /etc/grub2.cfg && grubby --args="user\_namespace.enable=1" --update-kernel="$(grubby --default-kernel)" && reboot

开机后查看内核

[appadmin@k8s-node01 ~]$ uname -a

Linux k8s-node01 5.7.7-1.el7.elrepo.x86\_64 #1 SMP Wed Jul 1 11:53:16 EDT 2020 x86\_64 x86\_64 x86\_64 GNU/Linux

所有节点 CentOS 8按需升级，默认内核版本是4.18

Cen可以采用dnf升级，也可使用上述同样步骤升级（使用上述步骤注意elrepo-release-8.1版本）

rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org

yum install https://www.elrepo.org/elrepo-release-8.el8.elrepo.noarch.rpm

dnf --disablerepo=\\* --enablerepo=elrepo-kernel -y install kernel-ml kernel-ml-devel

grubby --default-kernel && reboot

重启后查看内核

# uname -r

5.8.3-1.el8.elrepo.x86\_64

# 4. k8s组件安装

所有节点安装ipvs

[复制代码](javascript:void(0);)

yum install ipvsadm ipset sysstat conntrack libseccomp -y

# 内核4.18以下nf\_conntrack\_ipv4，内核高于4.18 nf\_conntrack

[root@k8s-master02 ~]# cat /etc/modules-load.d/ipvs.conf

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

[root@k8s-master02 ~]# systemctl enable --now systemd-modules-load.service

####### 警告忽略

[root@k8s-master02 ~]# lsmod | grep --color=auto -e ip\_vs -e nf\_conntrack

ip\_vs\_ftp 16384 0

ip\_vs\_sed 16384 0

ip\_vs\_nq 16384 0

ip\_vs\_fo 16384 0

ip\_vs\_dh 16384 0

ip\_vs\_lblcr 16384 0

ip\_vs\_lblc 16384 0

ip\_vs\_wlc 16384 0

ip\_vs\_lc 16384 0

ip\_vs\_sh 16384 0

ip\_vs\_wrr 16384 0

ip\_vs\_rr 16384 11

ip\_vs 172032 35 ip\_vs\_wlc,ip\_vs\_rr,ip\_vs\_dh,ip\_vs\_lblcr,ip\_vs\_sh,ip\_vs\_fo,ip\_vs\_nq,ip\_vs\_lblc,ip\_vs\_wrr,ip\_vs\_lc,ip\_vs\_sed,ip\_v\_ftp

nf\_defrag\_ipv6 20480 2 nf\_conntrack\_ipv6,ip\_vs

nf\_nat 36864 3 nf\_nat\_ipv6,nf\_nat\_ipv4,ip\_vs\_ftp

nf\_conntrack 155648 9 xt\_conntrack,nf\_conntrack\_ipv6,nf\_conntrack\_ipv4,nf\_nat,nf\_nat\_ipv6,ipt\_MASQUERADE,nf\_nat\_ipv4,nf\_conntrack\_netlink,ip\_vs

libcrc32c 16384 4 nf\_conntrack,nf\_nat,xfs,ip\_vs

[复制代码](javascript:void(0);)

　所有节点配置内核参数

[复制代码](javascript:void(0);)

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

　　所有节点安装新版containerd

[root@k8s-master01 k8s-ha-install]# wget https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/containerd.io-1.2.13-3.2.el7.x86\_64.rpm

[root@k8s-master01 k8s-ha-install]# yum install containerd.io-1.2.13-3.2.el7.x86\_64.rpm -y

　　所有节点安装最新版Docker

[root@k8s-master01 k8s-ha-install]# yum install docker-ce -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 -p /etc/docker

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

{

"registry-mirrors": [

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

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

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

],

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

}

EOF

所有节点开启Docker并设置开机自启动

[root@k8s-master01 k8s-ha-install]# systemctl daemon-reload && systemctl enable --now docker

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

# docker info

Client:

Debug Mode: false

Server:

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 19.03.12

Storage Driver: overlay2

Backing Filesystem: xfs

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: systemd

Plugins:

Volume: local

Network: bridge host ipvlan macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429

runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd

init version: fec3683

Security Options:

seccomp

Profile: default

Kernel Version: 4.18.0-193.14.2.el8\_2.x86\_64

Operating System: CentOS Linux 8 (Core)

OSType: linux

Architecture: x86\_64

CPUs: 2

Total Memory: 1.758GiB

Name: k8s-master01

ID: 5JBB:56IH:DUKQ:6D6X:ZJN6:A3SU:6KAX:7K35:UEDJ:R3UM:W5EF:GUKP

Docker Root Dir: /var/lib/docker

Debug Mode: false

Registry: https://index.docker.io/v1/

Labels:

Experimental: false

Insecure Registries:

127.0.0.0/8

下载慢的话，可以从百度云下载

Master01下载kubernetes安装包

[root@k8s-master01 ~]# wget https://dl.k8s.io/v1.19.0/kubernetes-server-linux-amd64.tar.gz

注意目前版本是　1.19.0学员安装时需要下载最新的版本：<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.19.md>

打开页面后点击：



下载etcd安装包

[root@k8s-master01 ~]# wget https://github.com/etcd-io/etcd/releases/download/v3.4.12/etcd-v3.4.12-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-v3.4.12-linux-amd64.tar.gz --strip-components=1 -C /usr/local/bin etcd-v3.4.12-linux-amd64/etcd{,ctl}

版本查看

# kubelet --version

Kubernetes v1.19.0

[root@k8s-master01 ~]# etcdctl version

etcdctl version: 3.4.12

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

~~新版k8s无需单独安装CNI~~

~~CNI安装，下载CNI组件~~

~~wget~~ [~~https://github.com/containernetworking/plugins/releases/download/v0.8.5/cni-plugins-linux-amd64-v0.8.5.tgz~~](https://github.com/containernetworking/plugins/releases/download/v0.8.5/cni-plugins-linux-amd64-v0.8.5.tgz)

~~解压cni并发送至其他节点~~

~~tar -zxf cni-plugins-linux-amd64-v0.8.5.tgz -C /opt/cni/bin~~

~~for NODE in $MasterNodes; do ssh $NODE 'mkdir -p /opt/cni/bin'; scp /opt/cni/bin/\* $NODE:/opt/cni/bin/; done~~

~~for NODE in $WorkNodes; do ssh $NODE 'mkdir -p /opt/cni/bin'; scp /opt/cni/bin/\* $NODE:/opt/cni/bin/; done~~

# 5. 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

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

 　　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

[root@k8s-master01 pki]# cfssl gencert -initca etcd-ca-csr.json | cfssljson -bare /etc/etcd/ssl/etcd-ca

[root@k8s-master01 pki]# 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-master03,192.168.0.201,192.168.0.202,192.168.0.203 \  
 -profile=kubernetes \

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

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

[root@k8s-master01 pki]# MasterNodes='k8s-master02 k8s-master03'

[root@k8s-master01 pki]# WorkNodes='k8s-node01 k8s-node02'

[root@k8s-master01 pki]#

[root@k8s-master01 pki]#

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# 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

　　Master01生成kubernetes证书

[root@k8s-master01 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，192.168.0.211为Master01的IP

[root@k8s-master01 pki]# cfssl gencert -ca=/etc/kubernetes/pki/ca.pem -ca-key=/etc/kubernetes/pki/ca-key.pem -config=ca-config.json -hostname=10.96.0.1,192.168.0.211,127.0.0.1,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,192.168.0.201,192.168.0.202,192.168.0.203 -profile=kubernetes apiserver-csr.json | cfssljson -bare /etc/kubernetes/pki/apiserver

[root@k8s-master01 pki]# cfssl gencert -initca front-proxy-ca-csr.json | cfssljson -bare /etc/kubernetes/pki/front-proxy-ca

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

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# 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

# 生成controller-manage的证书

[root@k8s-master01 pki]# 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，192.168.0.211:8443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，

[root@k8s-master01 pki]# kubectl config set-cluster kubernetes \

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

--embed-certs=true \

--server=https://192.168.0.211:8443 \

--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

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# 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

User "system:kube-controller-manager" set.

[root@k8s-master01 pki]#

# 使用某个环境当做默认环境

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# kubectl config use-context system:kube-controller-manager@kubernetes \

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

Switched to context "system:kube-controller-manager@kubernetes".

[root@k8s-master01 pki]# 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，192.168.0.211:8443改为master01的地址，8443改为apiserver的端口，默认是6443

[root@k8s-master01 pki]# kubectl config set-cluster kubernetes \

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

--embed-certs=true \

--server=https://192.168.0.211:8443 \

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

Cluster "kubernetes" set.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# 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

User "system:kube-scheduler" set.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# kubectl config set-context system:kube-scheduler@kubernetes \

--cluster=kubernetes \

--user=system:kube-scheduler \

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

Context "system:kube-scheduler@kubernetes" created.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# kubectl config use-context system:kube-scheduler@kubernetes \

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

Switched to context "system:kube-scheduler@kubernetes".

[root@k8s-master01 pki]# 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，192.168.0.211:8443改为master01的地址，8443改为apiserver的端口，默认是6443

[root@k8s-master01 pki]# kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.pem --embed-certs=true --server=https://192.168.0.211:8443 --kubeconfig=/etc/kubernetes/admin.kubeconfig

Cluster "kubernetes" set.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# 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

User "kubernetes-admin" set.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# kubectl config set-context kubernetes-admin@kubernetes --cluster=kubernetes --user=kubernetes-admin --kubeconfig=/etc/kubernetes/admin.kubeconfig

Context "kubernetes-admin@kubernetes" created.

[root@k8s-master01 pki]#

[root@k8s-master01 pki]# kubectl config use-context kubernetes-admin@kubernetes --kubeconfig=/etc/kubernetes/admin.kubeconfig

Switched to context "kubernetes-admin@kubernetes".

~~# 全部采用自动办法，无需手动生成kubelet证书。~~

~~[root@k8s-master01 pki]# for NODE in k8s-master01 k8s-master02 k8s-master03; do~~

~~\cp kubelet-csr.json kubelet-$NODE-csr.json;~~

~~sed -i "s/\$NODE/$NODE/g" kubelet-$NODE-csr.json;~~

~~cfssl gencert \~~

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

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

~~-config=ca-config.json \~~

~~-hostname=$NODE \~~

~~-profile=kubernetes \~~

~~kubelet-$NODE-csr.json | cfssljson -bare /etc/kubernetes/pki/kubelet-$NODE;~~

~~rm -f kubelet-$NODE-csr.json~~

~~done~~

~~[root@k8s-master01 pki]# for NODE in k8s-master01 k8s-master02 k8s-master03; do~~

~~ssh $NODE "mkdir -p /etc/kubernetes/pki"~~

~~scp /etc/kubernetes/pki/ca.pem $NODE:/etc/kubernetes/pki/ca.pem~~

~~scp /etc/kubernetes/pki/kubelet-$NODE-key.pem $NODE:/etc/kubernetes/pki/kubelet-key.pem~~

~~scp /etc/kubernetes/pki/kubelet-$NODE.pem $NODE:/etc/kubernetes/pki/kubelet.pem~~

~~rm -f /etc/kubernetes/pki/kubelet-$NODE-key.pem /etc/kubernetes/pki/kubelet-$NODE.pem~~

~~done~~

~~[root@k8s-master01 pki]# for NODE in k8s-master01 k8s-master02 k8s-master03; do~~

~~ssh $NODE "cd /etc/kubernetes/pki && \~~

~~kubectl config set-cluster kubernetes \~~

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

~~--embed-certs=true \~~

~~--server=https://192.168.0.211:8443 \~~

~~--kubeconfig=/etc/kubernetes/kubelet.kubeconfig && \~~

~~kubectl config set-credentials system:node:${NODE} \~~

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

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

~~--embed-certs=true \~~

~~--kubeconfig=/etc/kubernetes/kubelet.kubeconfig && \~~

~~kubectl config set-context system:node:${NODE}@kubernetes \~~

~~--cluster=kubernetes \~~

~~--user=system:node:${NODE} \~~

~~--kubeconfig=/etc/kubernetes/kubelet.kubeconfig && \~~

~~kubectl config use-context system:node:${NODE}@kubernetes \~~

~~--kubeconfig=/etc/kubernetes/kubelet.kubeconfig"~~

~~done~~

　　创建ServiceAccount Key 🡪 secret

[root@k8s-master01 pki]# openssl genrsa -out /etc/kubernetes/pki/sa.key 2048

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

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

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

e is 65537 (0x010001)

[root@k8s-master01 pki]# openssl rsa -in /etc/kubernetes/pki/sa.key -pubout -out /etc/kubernetes/pki/sa.pub

writing RSA key

[root@k8s-master01 pki]#

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

[复制代码](javascript:void(0);)

# 6. Kubernetes系统组件配置

　　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

[复制代码](javascript:void(0);)master01

# cat */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://192.168.0.201:2380'

listen-client-urls: 'https://192.168.0.201:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.201:2380'

advertise-client-urls: 'https://192.168.0.201:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.201:2380,k8s-master02=https://192.168.0.202:2380,k8s-master03=https://192.168.0.203:2380'

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

[复制代码](javascript:void(0);)

 Master02 etcd 配置文件

cat /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://192.168.0.202:2380'

listen-client-urls: 'https://192.168.0.202:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.202:2380'

advertise-client-urls: 'https://192.168.0.202:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.201:2380,k8s-master02=https://192.168.0.202:2380,k8s-master03=https://192.168.0.203:2380'

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 etcd配置文件

cat /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://192.168.0.203:2380'

listen-client-urls: 'https://192.168.0.203:2379,http://127.0.0.1:2379'

max-snapshots: 3

max-wals: 5

cors:

initial-advertise-peer-urls: 'https://192.168.0.203:2380'

advertise-client-urls: 'https://192.168.0.203:2379'

discovery:

discovery-fallback: 'proxy'

discovery-proxy:

discovery-srv:

initial-cluster: 'k8s-master01=https://192.168.0.201:2380,k8s-master02=https://192.168.0.202:2380,k8s-master03=https://192.168.0.203:2380'

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

　　所有Master节点创建etcd service并启动

[root@k8s-master01 pki]# cat /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

Master节点创建etcd的证书目录

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

[root@k8s-master01 pki]# ln -s /etc/etcd/ssl/\* /etc/kubernetes/pki/etcd/

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

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

Created symlink /etc/systemd/system/etcd3.service → /usr/lib/systemd/system/etcd.service.

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

查看etcd状态

[root@k8s-master01 pki]# etcdctl --endpoints="192.168.0.203:2379,192.168.0.202:2379,192.168.0.201:2379" --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



Etcd changelog：<https://github.com/etcd-io/etcd/blob/master/Documentation/upgrades/upgrade_3_4.md>

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

　　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

[复制代码](javascript:void(0);)

[root@k8s-master01 pki]# cat /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 192.168.0.201:6443 check

server k8s-master02 192.168.0.202:6443 check

server k8s-master03 192.168.0.203:6443 check

　　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

Master01

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

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

interval 2

weight -5

fall 3

rise 2

}

vrrp\_instance VI\_1 {

state MASTER

interface ens33

mcast\_src\_ip 192.168.0.201

virtual\_router\_id 51

priority 100

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.211

}

track\_script {  
 chk\_apiserver

} }

Master02

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

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

interval 2

weight -5

fall 3

rise 2

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

mcast\_src\_ip 192.168.0.202

virtual\_router\_id 51

priority 90

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.211

}

track\_script {  
 chk\_apiserver

} }

Master03

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script chk\_apiserver {

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

interval 2

weight -5

fall 3

rise 2

}

vrrp\_instance VI\_1 {

state BACKUP

interface ens33

mcast\_src\_ip 192.168.0.203

virtual\_router\_id 51

priority 90

advert\_int 2

authentication {

auth\_type PASS

auth\_pass K8SHA\_KA\_AUTH

}

virtual\_ipaddress {

192.168.0.211

}

track\_script {  
 chk\_apiserver

} }

　　健康检查配置

[复制代码](javascript:void(0);)

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

#!/bin/bash

err=0

for k in $(seq 1 5)

do

check\_code=$(curl -k -s https://127.0.0.1:6443/healthz)

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

err=$(expr $err + 1)

sleep 5

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

[复制代码](javascript:void(0);)

　　启动HAProxy和KeepAlived

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

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

　　VIP测试

[root@k8s-master01 pki]# ping 192.168.0.211

PING 192.168.0.211 (192.168.0.211) 56(84) bytes of data.

64 bytes from 192.168.0.211: icmp\_seq=1 ttl=64 time=1.39 ms

64 bytes from 192.168.0.211: icmp\_seq=2 ttl=64 time=2.46 ms

64 bytes from 192.168.0.211: icmp\_seq=3 ttl=64 time=1.68 ms

64 bytes from 192.168.0.211: icmp\_seq=4 ttl=64 time=1.08 ms

　　Kubernetes组件配置

　　所有节点创建相关目录

[root@k8s-master01 pki]# mkdir -p /etc/kubernetes/manifests/ /etc/systemd/system/kubelet.service.d /var/lib/kubelet /var/log/kubernetes

　　所有Master节点创建kube-apiserver service，# 注意，如果不是高可用集群，192.168.0.211改为master01的地址

[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=192.168.0.211 \

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

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

--etcd-servers=https://192.168.0.201:2379,https://192.168.0.202:2379,https://192.168.0.203:2379 \

--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 \

--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

[复制代码](javascript:void(0);)

~~[root@k8s-master01 pki]# vim /etc/kubernetes/token.csv~~

~~[root@k8s-master01 pki]# cat !$~~

~~cat /etc/kubernetes/token.csv~~

~~d7d356746b508a1a478e49968fba7947,kubelet-bootstrap,10001,"system:kubelet-bootstrap"~~

　　所有Master节点开启kube-apiserver

[root@k8s-master01 pki]# 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

　　所有Master节点配置kube-controller-manager service

[复制代码](javascript:void(0);)

[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=10.244.0.0/16 \

--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.

　　所有Master节点配置kube-scheduler service

[复制代码](javascript:void(0);)

[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.

# 7.  TLS Bootstrapping配置

在Master01创建bootstrap

# 注意，如果不是高可用集群，192.168.0.211:8443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://192.168.0.211:8443 --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

[root@k8s-master01 bootstrap]# 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

# 8. Node节点配置

　　复制证书至Node节点

[root@k8s-master01 bootstrap]# 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

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

[复制代码](javascript:void(0);)

　　所有Node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（Master节点不部署Pod也可无需配置）

[复制代码](javascript:void(0);)

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

[root@k8s-master01 bootstrap]# cat !$

cat /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

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

[root@k8s-master01 bootstrap]#

[root@k8s-master01 bootstrap]# cat !$

cat /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-amd64:3.1"

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

[复制代码](javascript:void(0);)

注意：如果更改了k8s的service网段，需要更改kubelet-conf.yml 的clusterDNS:配置

[复制代码](javascript:void(0);)

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

[root@k8s-master01 bootstrap]# cat !$

cat /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

[复制代码](javascript:void(0);)

　　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

NAME STATUS ROLES AGE VERSION

k8s-master01 NotReady <none> 18s v1.19.0

k8s-master02 NotReady <none> 13s v1.19.0

k8s-master03 NotReady <none> 13s v1.19.0

k8s-node01 NotReady <none> 14s v1.19.0

k8s-node02 NotReady <none> 18s v1.19.0



Kube-Proxy配置（Master01），如果更改了集群Pod的网段，需要更改kube-proxy/kube-proxy.conf的clusterCIDR: 10.244.0.0/16参数。

# 注意，如果不是高可用集群，192.168.0.211:8443改为master01的地址，8443改为apiserver的端口，默认是6443

[复制代码](javascript:void(0);)

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://192.168.0.211:8443 --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

[复制代码](javascript:void(0);)

　　赋值Service文件

[复制代码](javascript:void(0);)

[root@k8s-master01 k8s-ha-install]# 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

[root@k8s-master01 k8s-ha-install]#

[root@k8s-master01 k8s-ha-install]# 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

[复制代码](javascript:void(0);)

　　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.

# 9. 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

 安装最新版calico

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

configmap/calico-config created

customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created

customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created

clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrole.rbac.authorization.k8s.io/calico-node created

clusterrolebinding.rbac.authorization.k8s.io/calico-node created

daemonset.apps/calico-node created

serviceaccount/calico-node created

deployment.apps/calico-kube-controllers created

serviceaccount/calico-kube-controllers created

镜像下载速度可能会慢，使用kubectl get 查看po状态，系统Pod都在kube-system命名空间下

 # kubectl get po -n kube-system -owide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

calico-kube-controllers-578894d4cd-n7tdw 1/1 Running 0 8h 10.244.122.132 k8s-master02 <none> <none>

calico-node-26wtv 1/1 Running 0 8h 192.168.0.204 k8s-node01 <none> <none>

calico-node-hqvp8 1/1 Running 0 8h 192.168.0.203 k8s-master03 <none> <none>

calico-node-m2jvj 1/1 Running 0 8h 192.168.0.205 k8s-node02 <none> <none>

calico-node-tkkn4 1/1 Running 0 8h 192.168.0.202 k8s-master02 <none> <none>

calico-node-whnhl 1/1 Running 0 85m 192.168.0.201 k8s-master01 <none> <none>

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

# 10. 安装CoreDNS

 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

# 11. 安装Metrics Server

# cd /root/k8s-ha-install/

# kubectl create -f metrics-server-0.3.7/

[root@k8s-master01 k8s-ha-install]# kubectl create -f metrics-server-0.3.7/

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

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

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

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

serviceaccount/metrics-server created

deployment.apps/metrics-server created

service/metrics-server created

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

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

查看状态

[root@k8s-master01 k8s-ha-install]# kubectl get po -n kube-system -l k8s-app=metrics-server

NAME READY STATUS RESTARTS AGE

metrics-server-589847c86f-g2pl2 1/1 Running 0 32s

查看集群度量指标

[root@k8s-master01 k8s-ha-install]# kubectl top po -n kube-system

NAME CPU(cores) MEMORY(bytes)

calico-kube-controllers-7bbb89569d-prjgk 7m 23Mi

calico-node-47jbn 90m 58Mi

calico-node-ltk7h 70m 72Mi

calico-node-p2zjr 71m 69Mi

calico-node-s8w2z 75m 56Mi

calico-node-zwkk8 98m 63Mi

coredns-7c965f6585-t8cts 5m 25Mi

metrics-server-589847c86f-g2pl2 125m 31Mi

# 12. 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

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

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）

[复制代码](javascript:void(0);)

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

Server: 10.96.0.10

Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: kubernetes

Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local

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

Server: 10.96.0.10

Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: kube-dns.kube-system

Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

# 13. 安装dashboard

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

找到最新的安装文件：$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.4/aio/deploy/recommended.yaml

# kubectl create -f dashboard/

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

查看dashboard容器的状态：

[root@k8s-master01 k8s-ha-install]# kubectl get po -n kubernetes-dashboard

NAME READY STATUS RESTARTS AGE

dashboard-metrics-scraper-7b59f7d4df-2w9jj 1/1 Running 0 49s

kubernetes-dashboard-548f88599b-bdwvj 1/1 Running 0 50s

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

访问Dashboard：[https://192.168.0.200:30000](https://192.168.20.10:30000)，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

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

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

# 14. 生产环境关键性配置

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

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