二进制安装高可用k8s集群

标签（空格分隔）： Kubernetes k8s

环境

```
CentOS Linux release 7.4.1708 (Core)
3.10.0-693.el7.x86_64

// soft 目录
mkdir /opt/soft
```

Topology

```
// 主机hosts文件内容
10.57.1.200 master01.topc.com master01  etcd01 etcd01.ilinux.io  kubernetes-api
.topc.com
10.57.1.201 master02.topc.com master02  etcd02 etcd02.ilinux.io
10.57.1.203 master03.topc.com master03
10.57.1.202 node01.topc.com node01 etcd03 etcd03.ilinux.io
```

SYS Init

```
见 k8s
```

免密码登录

```
ssh-keygen

ssh-copy-id -i .ssh/id_rsa.pub   master02

ssh-copy-id -i .ssh/id_rsa.pub   master03
```

ETCD Cluster

```
yum install etcd -y
systemctl daemon-reload
systemctl enable etcd
systemctl start etcd
```

```
// 生成证书
git clone  https://github.com/iKubernetes/k8s-certs-generator.git
cd k8s-certs-generator/
bash  gencerts.sh etcd // 注意域名，默认 ilinux.io
cp -rf pki/  /etc/etcd/
```

etcd01 etcd01.ilinux.io

```
[root@qa-master01-topc etcd]# cat etcd.conf
#[Member]
#ETCD_CORS=""
ETCD_DATA_DIR="/var/lib/etcd/k8s.etcd"
#ETCD_WAL_DIR=""
ETCD_LISTEN_PEER_URLS="https://10.57.1.200:2380"
ETCD_LISTEN_CLIENT_URLS="https://10.57.1.200:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
ETCD_NAME="etcd01.ilinux.io"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://etcd01.ilinux.io:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://etcd01.ilinux.io:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
ETCD_INITIAL_CLUSTER="etcd01.ilinux.io=https://etcd01.ilinux.io:2380,etcd02.ili
nux.io=https://etcd02.ilinux.io:2380,etcd03.ilinux.io=https://etcd03.ilinux.io:
2380"
ETCD_INITIAL_CLUSTER_TOKEN="k8s-etcd-cluster"
#ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
```

```
#[Proxy]
#ETCD_PROXY="off"
#ETCD_PROXY_FAILURE_WAIT="5000"
#ETCD_PROXY_REFRESH_INTERVAL="30000"
#ETCD_PROXY_DIAL_TIMEOUT="1000"
#ETCD_PROXY_WRITE_TIMEOUT="5000"
#ETCD_PROXY_READ_TIMEOUT="0"
#
#[Security]
ETCD_CERT_FILE="/etc/etcd/pki/server.crt"
ETCD_KEY_FILE="/etc/etcd/pki/server.key"
ETCD_CLIENT_CERT_AUTH="true"
ETCD_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
ETCD_AUTO_TLS="false"
ETCD_PEER_CERT_FILE="/etc/etcd/pki/peer.crt"
ETCD_PEER_KEY_FILE="/etc/etcd/pki/peer.key"
ETCD_PEER_CLIENT_CERT_AUTH="true"
ETCD_PEER_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
ETCD_PEER_AUTO_TLS="false"
#
#[Logging]
#ETCD_DEBUG="false"
#ETCD_LOG_PACKAGE_LEVELS=""
#ETCD_LOG_OUTPUT="default"
#
#[Unsafe]
#ETCD_FORCE_NEW_CLUSTER="false"
#
#[Version]
#ETCD_VERSION="false"
#ETCD_AUTO_COMPACTION_RETENTION="0"
#
#[Profiling]
#ETCD_ENABLE_PPROF="false"
#ETCD_METRICS="basic"
#
#[Auth]
#ETCD_AUTH_TOKEN="simple"
```

etcd02 etcd02.ilinux.io

```
[root@qa-master02-topc ~]# cat /etc/etcd/etcd.conf
#[Member]
```

```
#ETCD_CORS=""
ETCD_DATA_DIR="/var/lib/etcd/k8s.etcd"
#ETCD_WAL_DIR=""
ETCD_LISTEN_PEER_URLS="https://10.57.1.201:2380"
ETCD_LISTEN_CLIENT_URLS="https://10.57.1.201:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
ETCD_NAME="etcd02.ilinux.io"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://etcd02.ilinux.io:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://etcd02.ilinux.io:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
ETCD_INITIAL_CLUSTER="etcd01.ilinux.io=https://etcd01.ilinux.io:2380,etcd02.ili
nux.io=https://etcd02.ilinux.io:2380,etcd03.ilinux.io=https://etcd03.ilinux.io:
2380"
ETCD_INITIAL_CLUSTER_TOKEN="k8s-etcd-cluster"
#ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
#[Proxy]
#ETCD_PROXY="off"
#ETCD_PROXY_FAILURE_WAIT="5000"
#ETCD_PROXY_REFRESH_INTERVAL="30000"
#ETCD_PROXY_DIAL_TIMEOUT="1000"
#ETCD_PROXY_WRITE_TIMEOUT="5000"
#ETCD_PROXY_READ_TIMEOUT="0"
#
#[Security]
ETCD_CERT_FILE="/etc/etcd/pki/server.crt"
ETCD_KEY_FILE="/etc/etcd/pki/server.key"
ETCD_CLIENT_CERT_AUTH="true"
ETCD_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
```

```
ETCD_AUTO_TLS="false"
ETCD_PEER_CERT_FILE="/etc/etcd/pki/peer.crt"
ETCD_PEER_KEY_FILE="/etc/etcd/pki/peer.key"
ETCD_PEER_CLIENT_CERT_AUTH="true"
ETCD_PEER_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
ETCD_PEER_AUTO_TLS="false"
#
#[Logging]
#ETCD_DEBUG="false"
#ETCD_LOG_PACKAGE_LEVELS=""
#ETCD_LOG_OUTPUT="default"
#
#[Unsafe]
#ETCD_FORCE_NEW_CLUSTER="false"
#
#[Version]
#ETCD_VERSION="false"
#ETCD_AUTO_COMPACTION_RETENTION="0"
#
#[Profiling]
#ETCD_ENABLE_PPROF="false"
#ETCD_METRICS="basic"
#
#[Auth]
#ETCD_AUTH_TOKEN="simple"
```

etcd03 etcd03.ilinux.io

```
[root@qa-master03-topc ~]# cat /etc/etcd/etcd.conf
#[Member]
#ETCD_CORS=""
ETCD_DATA_DIR="/var/lib/etcd/k8s.etcd"
#ETCD_WAL_DIR=""
ETCD_LISTEN_PEER_URLS="https://10.57.1.202:2380"
ETCD_LISTEN_CLIENT_URLS="https://10.57.1.202:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
ETCD_NAME="etcd03.ilinux.io"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
```

```
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://etcd03.ilinux.io:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://etcd03.ilinux.io:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
ETCD_INITIAL_CLUSTER="etcd01.ilinux.io=https://etcd01.ilinux.io:2380,etcd02.ili
nux.io=https://etcd02.ilinux.io:2380,etcd03.ilinux.io=https://etcd03.ilinux.io:
2380"
ETCD_INITIAL_CLUSTER_TOKEN="k8s-etcd-cluster"
#ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
#[Proxy]
#ETCD_PROXY="off"
#ETCD_PROXY_FAILURE_WAIT="5000"
#ETCD_PROXY_REFRESH_INTERVAL="30000"
#ETCD_PROXY_DIAL_TIMEOUT="1000"
#ETCD_PROXY_WRITE_TIMEOUT="5000"
#ETCD_PROXY_READ_TIMEOUT="0"
#
#[Security]
ETCD_CERT_FILE="/etc/etcd/pki/server.crt"
ETCD_KEY_FILE="/etc/etcd/pki/server.key"
ETCD_CLIENT_CERT_AUTH="true"
ETCD_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
ETCD_AUTO_TLS="false"
ETCD_PEER_CERT_FILE="/etc/etcd/pki/peer.crt"
ETCD_PEER_KEY_FILE="/etc/etcd/pki/peer.key"
ETCD_PEER_CLIENT_CERT_AUTH="true"
ETCD_PEER_TRUSTED_CA_FILE="/etc/etcd/pki/ca.crt"
ETCD_PEER_AUTO_TLS="false"
#
#[Logging]
#ETCD_DEBUG="false"
#ETCD_LOG_PACKAGE_LEVELS=""
#ETCD_LOG_OUTPUT="default"
#
#[Unsafe]
```

```
#ETCD_FORCE_NEW_CLUSTER="false"
#
#[Version]
#ETCD_VERSION="false"
#ETCD_AUTO_COMPACTION_RETENTION="0"
#
#[Profiling]
#ETCD_ENABLE_PPROF="false"
#ETCD_METRICS="basic"
#
#[Auth]
#ETCD_AUTH_TOKEN="simple"
```

测试集群

```
etcdctl --cert-file=/etc/etcd/pki/client.crt --key-file=/etc/etcd/pki/client.key --ca-file=/etc/etcd/pki/ca.crt  --endpoints='https://etcd03.ilinux.io:2379' member list
etcdctl --cert-file=/etc/etcd/pki/client.crt --key-file=/etc/etcd/pki/client.key --ca-file=/etc/etcd/pki/ca.crt  --endpoints='https://etcd03.ilinux.io:2379' cluster-health
```

删除一个节点

```
// 删除 etcd03.ilinux.io
[root@master01 ~]# etcdctl --cert-file=/etc/etcd/pki/client.crt --key-file=/etc/etcd/pki/client.key --ca-file=/etc/etcd/pki/ca.crt  --endpoints='https://etcd03.ilinux.io:2379' member remove 1f22dc5568642e6f
```

新增一个节点

```
etcdctl member add etcd_name –peer-urls=”https://peerURLs”
// 假设要新加的节点取名为etcd03.ilinux.io, peerURLs 是 https://etcd03.ilinux.io:2380
etcdctl --cert-file=/etc/etcd/pki/client.crt --key-file=/etc/etcd/pki/client.key --ca-file=/etc/etcd/pki/ca.crt  --endpoints='https://etcd01.ilinux.io:2379' member add etcd03.ilinux.io https://etcd03.ilinux.io:2380
```

```
// 注意用下面参数启动新的节点
ETCD_NAME="etcd03.ilinux.io"
ETCD_INITIAL_CLUSTER="etcd03.ilinux.io=https://etcd03.ilinux.io:2380,etcd02.ilinux.io=https://etcd02.ilinux.io:2380,etcd01.ilinux.io=https://etcd01.ilinux.io:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"


error
// it doesn't contain any IP SANs (prober "ROUND_TRIPPER_RAFT_MESSAGE")
// ETCD with TLS showing warning "transport: authentication handshake failed: remote error: tls: bad certificate"
==> peerURLs 要用 域名。
```

k8s master cluster

```
// 所以主节点都执行
cd /usr/local/src
wget  https://dl.k8s.io/v1.14.0-rc.1/kubernetes-server-linux-amd64.tar.gz
tar xf kubernetes-server-linux-amd64.tar.gz -C /usr/local/
cat > /etc/profile.d/k8s.sh <<-EOF
PATH=$PATH:/usr/local/kubernetes/server/bin
EOF
chmod +x /etc/profile.d/k8s.sh
source /etc/profile

// 下载配置文件， 所以主节点都执行
cd /opt/soft
mkdir  /etc/kubernetes
git clone https://github.com/iKubernetes/k8s-bin-inst.git
cd k8s-bin-inst
cp  -r master/etc/kubernetes/*  /etc/kubernetes/
cp master/unit-files/* /usr/lib/systemd/system
```

生成证书

```
// 在master01 上做， 然后同步去其他两个master 节点。
git clone  https://github.com/iKubernetes/k8s-certs-generator.git
cd k8s-certs-generator
```

```
[root@qa-master01-topc k8s-certs-generator]# bash gencerts.sh k8s
Enter Domain Name [ilinux.io]: topc.com
Enter Kubernetes Cluster Name [kubernetes]:
Enter the IP Address in default namespace
  of the Kubernetes API Server[10.96.0.1]:
Enter Master servers name[master01 master02 master03]: master01 master02 master
03

cp -r kubernetes/master01/* /etc/kubernetes/

scp -rp  kubernetes/master02/*  master02:/etc/kubernetes/

scp -rp  kubernetes/master03/*  master03:/etc/kubernetes/
```

master01

```
cd /usr/local/src
tar xf kubernetes-server-linux-amd64.tar.gz -C /usr/local/
```

> config kube-apiserver

```
1) 通用配置文件
/etc/kubernetes/config
//生成环境 注意日志级别

2) apiserver 配置文件
/etc/kubernetes/apiserver
//注意 etcd 集群

3) controller-manager 配置文件
/etc/kubernetes/controller-manager

4) scheduler 配置文件
/etc/kubernetes/scheduler
```

```
// 创建运行用户和工作目录
useradd -r kube
mkdir /var/run/kubernetes
```

```
chown -R kube. /var/run/kubernetes

// 由于配置文件和unit-files我们都设置过了, 可以直接启动.
systemctl daemon-reload
systemctl start kube-apiserver
```

```
// 配置 kubectl
mkdir $HOME/.kube
cp /etc/kubernetes/auth/admin.conf $HOME/.kube/config

// 查看kube config 内容
kubectl config view --kubeconfig=/etc/kubernetes/auth/admin.conf
```

```
// 创建 ClusterRoleBinding, 授予用户相应操作所需要的权限, 对 token.csv 的用户或者组。

[root@qa-master01-topc ~]# cat /etc/kubernetes/token.csv
ba02a9.40a5b3b2b67841d2,"system:bootstrapper",10001,"system:bootstrappers"

[root@qa-master01-topc ~]# kubectl get clusterrole | grep node
system:certificates.k8s.io:certificatesigningrequests:nodeclient       82m
system:certificates.k8s.io:certificatesigningrequests:selfnodeclient   82m
system:controller:node-controller                                      82m
system:node                                                            82m
system:node-bootstrapper                                               82m
system:node-problem-detector                                           82m
system:node-proxier                                                    82m

// 创建一个clusterrolebinding (名字system:bootstrapper), 绑定用户system:bootstrap
per 到系统上system:node-bootstrapper的权限, 用了做 bootsrapper。
[root@qa-master01-topc ~]# kubectl create clusterrolebinding system:bootstrappe
r --user=system:bootstrapper --clusterrole=system:node-bootstrapper
clusterrolebinding.rbac.authorization.k8s.io/system:bootstrapper created
```

> config controller-manager

```
systemctl start kube-controller-manager
```

> config kube-scheduler

```
systemctl start kube-scheduler
```

node

```
// node 节点都需要安装dokcer, 见k8s
node 基于 bootstrapper方式, 通过token认证 加入集群.
```

config docker

```
// 需要基础架构容器的镜像
// 因为墙,  所以需要设置代理 (http://192.168.1.210:1080)
// 编辑 /usr/lib/systemd/system/docker.service, 添加如下选项, 完整见下面。

Environment=HTTPS_PROXY=http://192.168.1.210:1080
Environment=NO_PROXY=127.0.0.0/8,10.57.0.0/16
ExecStartPost=/usr/sbin/iptables -P FORWARD ACCEPT

// 然后执行
systemctl daemon-reload
systemctl start docker
systemctl enable docker
```

```
[root@qa-master03-topc ~]# cat /usr/lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues sti
ll
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
Environment=HTTPS_PROXY=http://192.168.1.210:1080
Environment=NO_PROXY=127.0.0.0/8,10.57.0.0/16
```

```
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecStartPost=/usr/sbin/iptables -P FORWARD ACCEPT
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
# Both the old, and new location are accepted by systemd 229 and up, so using the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.
# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not supports it.
# Only systemd 226 and above support this option.
TasksMax=infinity

# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes

# kill only the docker process, not all processes in the cgroup
KillMode=process

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

config kubelet

```
cd /usr/local/src
wget https://dl.k8s.io/v1.14.0-rc.1/kubernetes-node-linux-amd64.tar.gz
tar xf kubernetes-node-linux-amd64.tar.gz  -C /usr/local/
```

```
// kubelet 配置文件
mkdir  /opt/soft
cd /opt/soft
git clone https://github.com/iKubernetes/k8s-bin-inst.git
cd k8s-bin-inst
cp -rp  nodes/var/lib/* /var/lib/
cp -rp nodes/etc/kubernetes /etc/
```

```
// 从master01 上 copy 证书信息, /opt/soft/k8s-certs-generator/kubernetes
scp -r /opt/soft/k8s-certs-generator/kubernetes/kubelet/*  node01:/etc/kubernet
es/
```

```
// 网络插件加载方式 cni   kubelet-net
cd /usr/local/src/
wget https://github.com/containernetworking/plugins/releases/download/v0.7.5/cn
i-plugins-amd64-v0.7.5.tgz

// 要求 文件展开到 /opt/cni/bin 目录下
mkdir  /opt/cni/bin -p
tar xf cni-plugins-amd64-v0.7.5.tgz -C /opt/cni/bin/
```

```
// cp kubelet unit-file
cp /opt/soft/k8s-bin-inst/nodes/unit-files/* /usr/lib/systemd/system

// 注意为了做高可用, 连 kube-apiserver 的域名为kubernetes-api.topc.com, 请把解析先指
向master01,  然后启动 kubelet。
systemctl daemon-reload
systemctl start kubelet
systemctl enable kubelet
systemctl status kubelet
```

```
// 此时在master01 上就会有一个 csr 签署请求
[root@master01 kubernetes]# kubectl get csr
```

```
NAME        AGE    REQUESTOR             CONDITION
csr-gbqwp   6m3s   system:bootstrapper   Pending

// 确认并签署证书
[root@master01 kubernetes]# kubectl certificate approve csr-gbqwp
certificatesigningrequest.certificates.k8s.io/csr-gbqwp approved
[root@master01 kubernetes]# kubectl get nodes
NAME                STATUS     ROLES    AGE   VERSION
master03.topc.com   NotReady   <none>   6s    v1.14.0-rc.1
```

config kube-proxy

```
// 配置文件在
/etc/kubernetes/proxy
/var/lib/kube-proxy/config.yaml

// kube-proxy 也是apiserver 的客户端, 也需要kubeconfig 文件, 用于接入认证
/etc/kubernetes/auth/kube-proxy.conf
```

```
// 加载ipvs modules
cat > /etc/sysconfig/modules/ipvs.modules <<-'EOF'
#!/bin/bash
ipvs_mods_dir="/usr/lib/modules/$(uname -r)/kernel/net/netfilter/ipvs"
for i in $(ls $ipvs_mods_dir | grep -o "^[^.]*"); do
    /sbin/modinfo -F filenaem $i &> /dev/null
    if [ $? -eq 0 ]; then
        /sbin/modprobe $i
    fi
done
EOF

chmod +x /etc/sysconfig/modules/ipvs.modules

//执行该文件并查看加载情况
/etc/sysconfig/modules/ipvs.modules
lsmod | grep ip_vs
```

```
systemctl start kube-proxy
systemctl enable kube-proxy
systemctl status kube-proxy
```

配置网络插件

```
https://github.com/coreos/flannel
```

```
// 在主节点上执行
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

// 在master01 检查 flannel pod 运行情况
[root@master01 ~]# kubectl get  pods -n kube-system
NAME                          READY   STATUS    RESTARTS   AGE
kube-flannel-ds-amd64-2qcx4   1/1     Running   0          1m
```

配置coredns

```
// 在master01 上执行
cd  /opt/soft
mkdir coredns && cd coredns

// 下载安装模板
wget https://raw.githubusercontent.com/coredns/deployment/master/kubernetes/coredns.yaml.sed

// 下载模板deploy脚本
wget https://raw.githubusercontent.com/coredns/deployment/master/kubernetes/deploy.sh

// deploy
bash deploy.sh -i 10.96.0.10 -r "10.96.0.0/12" -s -t coredns.yaml.sed | kubectl
 apply -f -

[root@master01 coredns]# kubectl get pods -n kube-system
NAME                          READY   STATUS    RESTARTS   AGE
coredns-5f44b47f5f-7hv4n      1/1     Running   0          42s
coredns-5f44b47f5f-hqtx9      1/1     Running   1          42s
kube-flannel-ds-amd64-2qcx4   1/1     Running   0          166m
```

master02

```
// 在master01 上执行
scp /usr/local/src/kubernetes-server-linux-amd64.tar.gz master02:/usr/local/src
```

```
// 在 master02 执行
cd /usr/local/src/
tar xf kubernetes-server-linux-amd64.tar.gz  -C /usr/local/

// 从 master01上复制配置文件，(或者自己clone下载)
 scp /opt/soft/k8s-bin-inst/master/etc/kubernetes/*  master02:/etc/kubernetes/

// 从 master01上复制 unit-files
scp  /opt/soft/k8s-bin-inst/master/unit-files/* master02:/usr/lib/systemd/system

// 从 master01上复制证书，  已经做过了。

// 编辑apiserver 的配置文件(/etc/kubernetes/apiserver), 设置etcd集群

// 创建运行用户和工作目录
useradd -r kube
mkdir /var/run/kubernetes
chown -R kube. /var/run/kubernetes

// 由于配置文件和unit-files我们都设置过了, 可以直接启动.
systemctl daemon-reload
systemctl start kube-apiserver
systemctl enable kube-apiserver

// 配置 kubectl
mkdir $HOME/.kube
cp /etc/kubernetes/auth/admin.conf $HOME/.kube/config

// clusterrolebinding , 因为同一个集群, master01 做完了, 所以这步不需要做了。
```

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

```
systemctl start kube-scheduler
systemctl enable kube-scheduler
```

查看集群的生效的 kube-controller-manager kube-scheduler

```
[root@master02 ~]# kubectl get endpoints -n kube-system
NAME                      ENDPOINTS   AGE
kube-controller-manager   <none>      17h
kube-scheduler            <none>      17h

[root@master02 ~]# kubectl get endpoints kube-controller-manager -n kube-system
 -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"master01.topc
.com_17077dfc-4ef0-11e9-a1af-000c2996c644","leaseDurationSeconds":15,"acquireTi
me":"2019-03-25T11:21:09Z","renewTime":"2019-03-26T04:37:40Z","leaderTransition
s":0}'
  creationTimestamp: "2019-03-25T11:21:09Z"
  name: kube-controller-manager
  namespace: kube-system
  resourceVersion: "71465"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-controller-manager
  uid: 1709e024-4ef0-11e9-96c5-000c2996c644
```

master03

```
// 现在master03 上直接目录创建
mkdir /etc/etcd

// 因为 etcd的一个节点放到node01上了， 所以要在master01上执行一个操作
scp -rp /opt/soft/k8s-certs-generator/etcd/pki master03:/etc/etcd
```

```
// 在master01 上执行
scp /usr/local/src/kubernetes-server-linux-amd64.tar.gz master03:/usr/local/src
```

```
// 在 master03 执行
cd /usr/local/src/
tar xf kubernetes-server-linux-amd64.tar.gz  -C /usr/local/

// 从 master01上复制配置文件，(或者自己clone下载)
 scp /opt/soft/k8s-bin-inst/master/etc/kubernetes/*  master03:/etc/kubernetes/

// 从 master01上复制 unit-files
scp  /opt/soft/k8s-bin-inst/master/unit-files/* master03:/usr/lib/systemd/system

// 从 master01上复制证书，  已经做过了。

// 编辑apiserver 的配置文件(/etc/kubernetes/apiserver), 设置etcd集群

// 创建运行用户和工作目录
useradd -r kube
mkdir /var/run/kubernetes
chown -R kube. /var/run/kubernetes

// 由于配置文件和unit-files我们都设置过了, 可以直接启动.
systemctl daemon-reload
systemctl start kube-apiserver
systemctl enable kube-apiserver

// 配置 kubectl
mkdir $HOME/.kube
cp /etc/kubernetes/auth/admin.conf $HOME/.kube/config

// clusterrolebinding , 因为同一个集群, master01 做完了, 所以这步不需要做了。
```

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

```
systemctl start kube-scheduler
systemctl enable kube-scheduler
```