How to Deploy a HA Kubernetes Cluster with kubeadm on CentOS7

Kubernetes is a **cluster** and **orchestration** engine for docker containers. In other words Kubernetes is an open source software or tool which is used to orchestrate and manage docker containers in cluster environment. Kubernetes is also known as k8s and it was developed by Google and donated to “Cloud Native Computing foundation”

In **Kubernetes** with **ETCD** setup we have at lease three (3) masters nodes and multiple worker nodes. Cluster nodes is known as worker node or Minion. From the master node we manage the cluster and its nodes using ‘**kubeadm**‘ and ‘**kubectl**‘ command.

Kubernetes cluster is highly configurable. Many of its components is optional. Our deployment consists of the following components: **Kubernetes, Etcd, Docker, Flannel Network, Dashboard and Heapster.**

Arquitectura

Server Name	IP Address	Role
k8-master01	192.168.20.20	Master Node
k8-master02	192.168.20.21	Master Node
k8-master03	192.168.20.22	Master Node
k8-worker01	192.168.20.24	Worker Node
k8-worker02	192.168.20.25	Worker Node
k8-worker03	192.168.20.26	Worker Node
k8-registry01	192.168.20.27	Registry Images

Preparando los servidores

There are a few things to be done to get the servers ready. You need to perform the following task ***on all servers (masters and workers)***

1) **Deshabilitar Selinux**

```
# setenforce 0
# sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

o

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

2) **Deshabilitar swap**

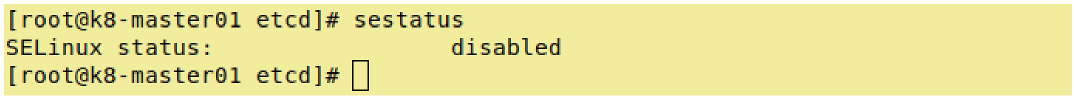
```
# swapoff -a
# sed -i 's/^.*swap/#&/' /etc/fstab
```

3) **Deshabilitar firewalld**

```
# systemctl stop firewalld
# systemctl disable firewalld
# modprobe br_netfilter
```

4) **Reiniciar servidores y verificar selinux**

```
# shutdown -r now

# sestatus
```



```
[root@k8-master01 etcd]# sestatus
SELinux status:                 disabled
[root@k8-master01 etcd]# 
```

5) **Editar el archivo /etc/hosts**

```
192.168.20.20 k8-master01    K8-MASTER01
192.168.20.21 k8-master02    K8-MASTER02
192.168.20.22 k8-master03    K8-MASTER03
192.168.20.24 k8-worker01    K8-WORKER01
192.168.20.25 k8-worker02    K8-WORKER02
192.168.20.26 k8-worker03    K8-WORKER03
192.168.20.27 k8-registry01 K8-REGISTRY01
```

6) **Instalar NTP**

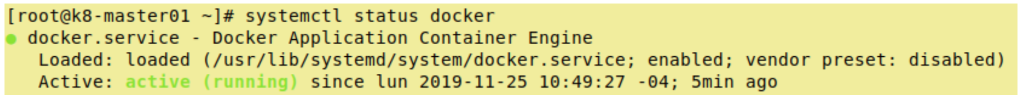
```
# yum install -y ntp
# systemctl start ntpd
# systemctl enable ntpd
```

7) **Establecer la timezone**

```
# timedatectl set-timezone America/Caracas
```

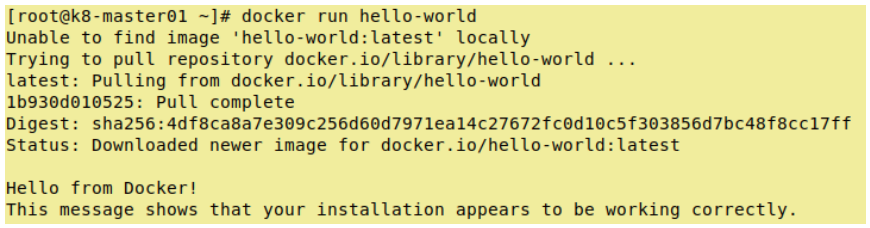
8) **Instalar Docker**

```
# yum install -y docker
# systemctl enable docker
# systemctl start docker
# systemctl status docker
```



```
[root@k8-master01 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since lun 2019-11-25 10:49:27 -04; 5min ago
```

```
# docker run hello-world
```



```
[root@k8-master01 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
1b930d010525: Pull complete
Digest: sha256:4df8ca8a7e309c256d60d7971ea14c27672fc0d10c5f303856d7bc48f8cc17ff
Status: Downloaded newer image for docker.io/hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
# groupadd docker
# chown root:docker /var/run/docker.sock
# usermod -aG docker $(whoami)
```

9) **Instalar kubelet, kubeadm, kubectl**

```
# vim /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  https://packages.cloud.google.com/yum/doc/
rpm-package-key.gpg
```

```
# yum list --showduplicates kubeadm --disableexcludes=kubernetes
```

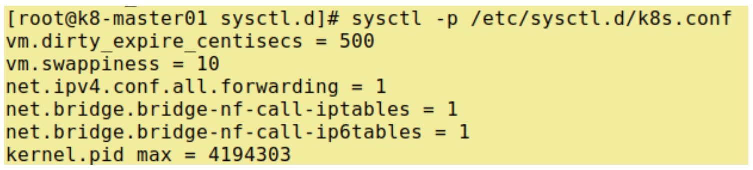
```
# yum install -y kubeadm-1.15.4-0 kubectl-1.15.4-0 kubelet-1.15.4-0 --
disableexcludes=kubernetes
```

10) **Crear el archivo /etc/sysctl.d/k8s.conf con el siguiente contenido y luego ejecutar el comando sysctl -p /etc/sysctl.d/k8s.conf:**

```
# vim /etc/sysctl.d/k8s.conf
```

```
vm.dirty_expire_centisecs = 500
vm.swappiness = 10
net.ipv4.conf.all.forwarding=1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
kernel.pid_max = 4194303
```

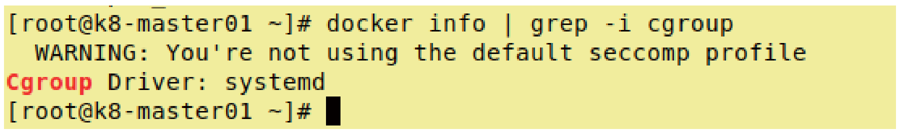
```
# sysctl -p /etc/sysctl.d/k8s.conf
```



```
[root@k8-master01 sysctl.d]# sysctl -p /etc/sysctl.d/k8s.conf
vm.dirty_expire_centisecs = 500
vm.swappiness = 10
net.ipv4.conf.all.forwarding = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
kernel.pid_max = 4194303
```

11) **Verificar el controlador de cgroup utilizado por el sistema operativo y editar el archivo /etc/sysconfig/kubelet**

```
# docker info | grep -i cgroup
```



```
[root@k8-master01 ~]# docker info | grep -i cgroup
  WARNING: You're not using the default seccomp profile
Cgroup Driver: systemd
[root@k8-master01 ~]#
```

```
# vim /etc/sysconfig/kubelet
```

```
KUBELET_EXTRA_ARGS= --runtime-cgroups=/systemd/system.slice --kubelet-cgroups=/systemd/system.slice
```

```
# systemctl daemon-reload
```

```
# systemctl enable kubelet && systemctl start kubelet
```

12) **Crear certificados (ejecutar en todos los nodos)**

```
# curl -o /usr/local/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64

# curl -o /usr/local/bin/cfssljson https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64

# chmod +x /usr/local/bin/cfssl*

# mkdir -p /etc/kubernetes/pki/etcd && cd /etc/kubernetes/pki/etcd
```

##
Hasta aquí es igual tanto para masters como para workers
##

13) **En el nodo 1 del master, crear los archivos ca-config.json y ca-csr.json en /etc/kubernetes/pki/etcd para crear los certificados**

ca-config.json

```
{
"signing": {
        "default": {
                "expiry": "43800h"
        },
        "profiles": {
                "server": {
                        "expiry": "43800h",
                        "usages": [
                                "signing",
                                "key encipherment",
                                "server auth",
                                "client auth"
                        ]
                },
                "client": {
                        "expiry": "43800h",
                        "usages": [
                                "signing",
                                "key encipherment",
                                "client auth"
                        ]
                },
                "peer": {
                        "expiry": "43800h",
                        "usages": [
                                "signing",
                                "key encipherment",
                                "server auth",
                                "client auth"
                        ]
                        }
                }
        }
}
```

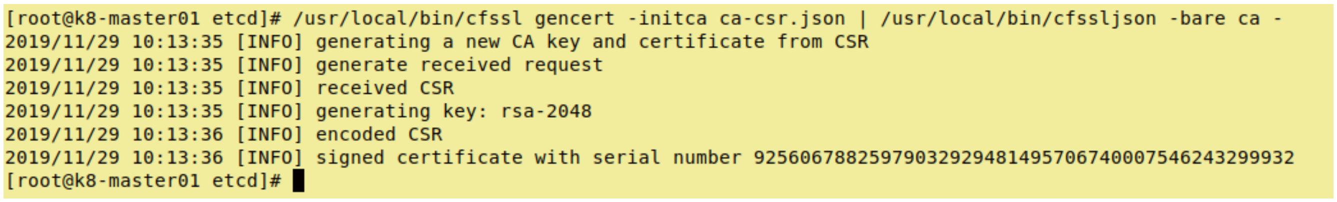
ca-csr.json

```
{
        "CN": "etcd",
        "key": {
                "algo": "rsa",
                "size": 2048
        }
}
```

<u>En el **nodo 1 del master**, generar los certificados</u>

```
# cd /etc/kubernetes/pki/etcd
```

```
# /usr/local/bin/cfssl gencert -initca ca-csr.json | /usr/local/bin/cfssljson -bare
ca -
```



```
[root@k8-master01 etcd]# /usr/local/bin/cfssl gencert -initca ca-csr.json | /usr/local/bin/cfssljson -bare ca -
2019/11/29 10:13:35 [INFO] generating a new CA key and certificate from CSR
2019/11/29 10:13:35 [INFO] generate received request
2019/11/29 10:13:35 [INFO] received CSR
2019/11/29 10:13:35 [INFO] generating key: rsa-2048
2019/11/29 10:13:36 [INFO] encoded CSR
2019/11/29 10:13:36 [INFO] signed certificate with serial number 925606788259790329294814957067400075462432999932
[root@k8-master01 etcd]# 
```

Al ejecutar el comando se generan 3 archivos en **/etc/kubernetes/pki/etcd**

ca.pem
ca-key.pem
ca.csr

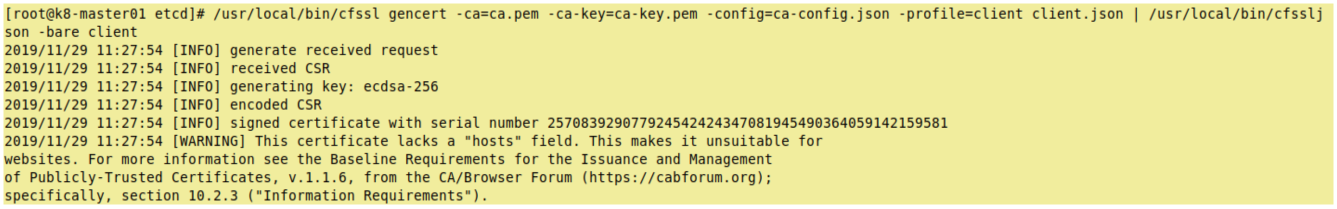
14) **En el nodo 1 del master, crear el certificado cliente. Para esto crear el archivo /etc/kubernetes/pki/etcd/client.json con el siguiente contenido:**

<u>**client.json**</u>

```
{
        "CN": "client",
        "key": {
                "algo": "ecdsa",
                "size": 256
        }
}
```

<u>En el nodo 1 del master, crear el certificado cliente en /etc/kubernetes/pki/etcd ejecutar:</u>

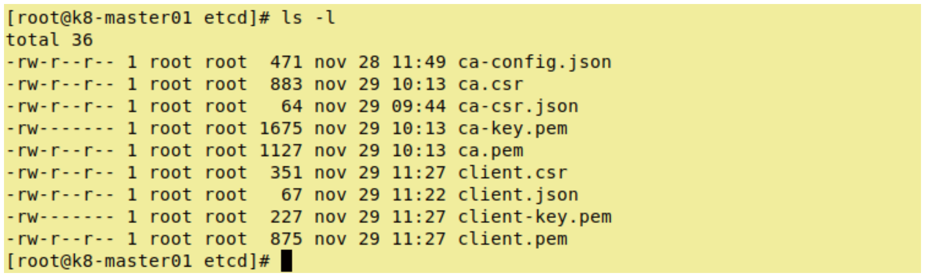
```
# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
-profile=client client.json | /usr/local/bin/cfssljson -bare client
```



```
[root@k8-master01 etcd]# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client client.json | /usr/local/bin/cfsslj
son -bare client
2019/11/29 11:27:54 [INFO] generate received request
2019/11/29 11:27:54 [INFO] received CSR
2019/11/29 11:27:54 [INFO] generating key: ecdsa-256
2019/11/29 11:27:54 [INFO] encoded CSR
2019/11/29 11:27:54 [INFO] signed certificate with serial number 257083929077924542424347081945490364059142159581
2019/11/29 11:27:54 [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").
```

<u>Al ejecutar el comando se generan 3 archivos en /etc/kubernetes/pki/etcd:</u>

```
client.csr
client-key.pem
client.pem
```

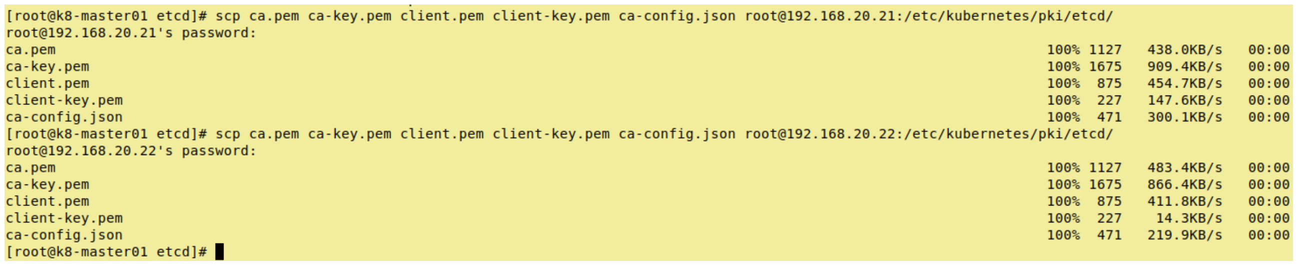


```
[root@k8-master01 etcd]# ls -l
total 36
-rw-r--r-- 1 root root  471 nov 28 11:49 ca-config.json
-rw-r--r-- 1 root root  883 nov 29 10:13 ca.csr
-rw-r--r-- 1 root root   64 nov 29 09:44 ca-csr.json
-rw------- 1 root root 1675 nov 29 10:13 ca-key.pem
-rw-r--r-- 1 root root 1127 nov 29 10:13 ca.pem
-rw-r--r-- 1 root root  351 nov 29 11:27 client.csr
-rw-r--r-- 1 root root   67 nov 29 11:22 client.json
-rw------- 1 root root  227 nov 29 11:27 client-key.pem
-rw-r--r-- 1 root root  875 nov 29 11:27 client.pem
[root@k8-master01 etcd]#
```

15) **En el resto de los nodos master, copiar en la carpeta /etc/kubernetes/pki/etcd los siguientes archivos desde el nodo 1 master**

```
ca.pem
ca-key.pem
client.pem
client-key.pem
ca-config.json
```

```
# scp ca.pem ca-key.pem client.pem client-key.pem ca-config.json root@192.168.20.21:/etc/kubernetes/pki/etcd/

# scp ca.pem ca-key.pem client.pem client-key.pem ca-config.json root@192.168.20.22:/etc/kubernetes/pki/etcd/
```



```
[root@k8-master01 etcd]# scp ca.pem ca-key.pem client.pem client-key.pem ca-config.json root@192.168.20.21:/etc/kubernetes/pki/etcd/
root@192.168.20.21's password:
ca.pem                                                                                        100% 1127   438.0KB/s   00:00
ca-key.pem                                                                                    100% 1675   909.4KB/s   00:00
client.pem                                                                                    100%  875   454.7KB/s   00:00
client-key.pem                                                                                100%  227   147.6KB/s   00:00
ca-config.json                                                                                100%  471   300.1KB/s   00:00
[root@k8-master01 etcd]# scp ca.pem ca-key.pem client.pem client-key.pem ca-config.json root@192.168.20.22:/etc/kubernetes/pki/etcd/
root@192.168.20.22's password:
ca.pem                                                                                        100% 1127   483.4KB/s   00:00
ca-key.pem                                                                                    100% 1675   866.4KB/s   00:00
client.pem                                                                                    100%  875   411.8KB/s   00:00
client-key.pem                                                                                100%  227    14.3KB/s   00:00
ca-config.json                                                                                100%  471   219.9KB/s   00:00
[root@k8-master01 etcd]#
```

16) **En cada nodo master ejecutar los siguientes comandos**:

```
# /usr/local/bin/cfssl print-defaults csr > /etc/kubernetes/pki/etcd/config.json
```

Este comando genera el archivo **config.json** en /etc/kubernetes/pki/etcd

config.json

```
{
    "CN": "example.net",
    "hosts": [
        "example.net",
        "www.example.net"
    ],
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "ST": "San Francisco"
        }
    ]
}
```

```
# export PRIVATE_IP=$(ip addr show ens160 | grep -Po 'inet \K[\d.]+') && export PEER_NAME=$(hostname)
```

```
# sed -i '0,/CN/{s/example\.net/'"$PEER_NAME"'/}' /etc/kubernetes/pki/etcd/config.json
```

```
# sed -i 's/www\.example\.net/'"$PRIVATE_IP"'/' /etc/kubernetes/pki/etcd/config.json
```

```
# sed -i 's/example\.net/'"$PEER_NAME"'/' /etc/kubernetes/pki/etcd/config.json
```

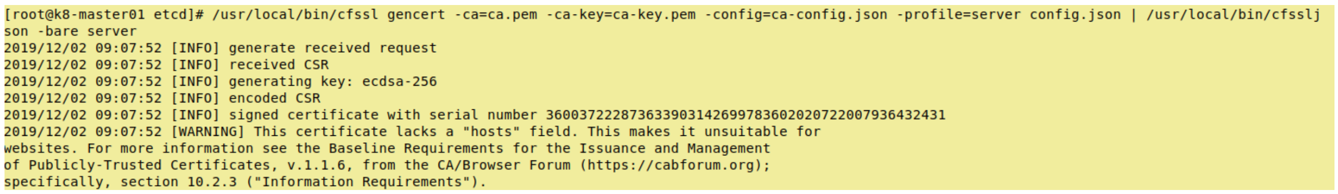
El objetivo de los comandos anteriores es configurar el archivo **config.json** con la ip y nombre del nodo master.

Luego edite manualmente el archivo **config.json** (C: país, L: estado, ST: ciudad) según su ubicación.

config.json

```
{
    "CN": "k8-master01",
    "hosts": [
        "k8-master01",
        "192.168.20.20"
    ],
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "VE",
            "L": "DC",
            "ST": "CCS"
        }
    ]
}
```

```
# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server config.json | /usr/local/bin/cfssljson -bare server
```

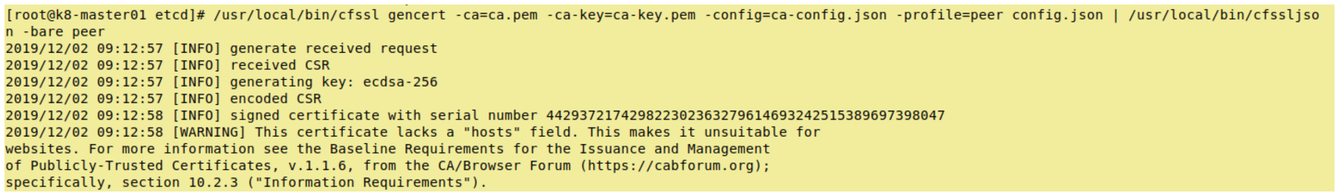


```
[root@k8-master01 etcd]# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server config.json | /usr/local/bin/cfsslj
son -bare server
2019/12/02 09:07:52 [INFO] generate received request
2019/12/02 09:07:52 [INFO] received CSR
2019/12/02 09:07:52 [INFO] generating key: ecdsa-256
2019/12/02 09:07:52 [INFO] encoded CSR
2019/12/02 09:07:52 [INFO] signed certificate with serial number 360037222873633903142699783602020722007936432431
2019/12/02 09:07:52 [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").
```

El comando anterior genera los siguientes archivos en /etc/kubernetes/pki/etcd:

```
server.csr
server-key.pem
server.pem
```

```
# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer config.json | /usr/local/bin/cfssljson -bare peer
```



```
[root@k8-master01 etcd]# /usr/local/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer config.json | /usr/local/bin/cfssljso
n -bare peer
2019/12/02 09:12:57 [INFO] generate received request
2019/12/02 09:12:57 [INFO] received CSR
2019/12/02 09:12:57 [INFO] generating key: ecdsa-256
2019/12/02 09:12:57 [INFO] encoded CSR
2019/12/02 09:12:58 [INFO] signed certificate with serial number 442937217429822302363279614693242515389697398047
2019/12/02 09:12:58 [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").
```

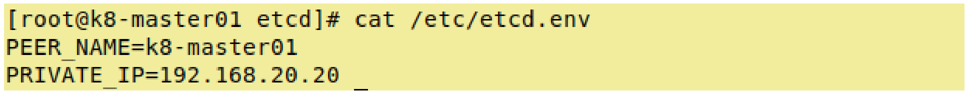
El comando anterior genera los siguientes archivos en /etc/kubernetes/pki/etcd:

```
peer.csr
peer-key.pem
peer.pem
```

17) **Instalar y configurar ETCD en cada nodo master**

```
# yum -y install etcd

# touch /etc/etcd.env

# export PRIVATE_IP=$(ip addr show eth0 | grep -Po 'inet \K[\d.]+') && export PEER_NAME=$(hostname)

# echo "PEER_NAME=${PEER_NAME}" >> /etc/etcd.env

# echo "PRIVATE_IP=${PRIVATE_IP}" >> /etc/etcd.env
```

El objetivo de los comandos anteriores es instalar etcd y crear el archivo **/etc/etcd.env** con los valores **PEER_NAME** y **PRIVATE_IP** en los nodos masters



```
[root@k8-master01 etcd]# cat /etc/etcd.env
PEER_NAME=k8-master01
PRIVATE_IP=192.168.20.20 _
```

En cada nodo master generar el archivo **etcd.service** en **/etc/systemd/system/** con el siguiente contenido:

```
# vim /etc/systemd/system/etcd.service
```

- En el master 1 k8-master01:

```
[Unit]
Description=etcd
Documentation=https://github.com/coreos/etcd
Conflicts=etcd.service
```

```
Conflicts=etcd2.service
[Service]
EnvironmentFile=/etc/etcd.env
Type=notify
Restart=always
RestartSec=5s
LimitNOFILE=40000
TimeoutStartSec=0
ExecStart=/usr/bin/etcd \
--name k8-master01 \
--data-dir /var/lib/etcd \
--listen-client-urls https://192.168.20.20:2379 \
--advertise-client-urls https://192.168.20.20:2379 \
--listen-peer-urls https://192.168.20.20:2380 \
--initial-advertise-peer-urls https://192.168.20.20:2380 \
--cert-file=/etc/kubernetes/pki/etcd/server.pem \
--key-file=/etc/kubernetes/pki/etcd/server-key.pem \
--client-cert-auth \
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.pem \
--peer-key-file=/etc/kubernetes/pki/etcd/peer-key.pem \
--peer-client-cert-auth --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--initial-cluster k8-master01=https://192.168.20.20:2380,k8-master02=https://192.168.20.21:2380,k8-
master03=https://192.168.20.22:2380 \
--initial-cluster-token my-etcd-token \
--initial-cluster-state new

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

<u>- En el master 2 k8-master02:</u>

```
[Unit]
Description=etcd
Documentation=https://github.com/coreos/etcd
Conflicts=etcd.service
Conflicts=etcd2.service
[Service]
EnvironmentFile=/etc/etcd.env
Type=notify
Restart=always
RestartSec=5s
LimitNOFILE=40000
TimeoutStartSec=0
ExecStart=/usr/bin/etcd \
--name k8-master02 \
--data-dir /var/lib/etcd \
--listen-client-urls https://192.168.20.21:2379 \
--advertise-client-urls https://192.168.20.21:2379 \
--listen-peer-urls https://192.168.20.21:2380 \
--initial-advertise-peer-urls https://192.168.20.21:2380 \
--cert-file=/etc/kubernetes/pki/etcd/server.pem \
--key-file=/etc/kubernetes/pki/etcd/server-key.pem \
--client-cert-auth \
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.pem \
--peer-key-file=/etc/kubernetes/pki/etcd/peer-key.pem \
--peer-client-cert-auth --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--initial-cluster k8-master01=https://192.168.20.20:2380,k8-master02=https://192.168.20.21:2380,k8-
master03=https://192.168.20.22:2380 \
--initial-cluster-token my-etcd-token \
--initial-cluster-state new

[Install]
```

```
WantedBy=multi-user.target
```

<u>- En el master 3 k8-master03:</u>

```
[Unit]
Description=etcd
Documentation=https://github.com/coreos/etcd
Conflicts=etcd.service
Conflicts=etcd2.service
[Service]
EnvironmentFile=/etc/etcd.env
Type=notify
Restart=always
RestartSec=5s
LimitNOFILE=40000
TimeoutStartSec=0
ExecStart=/usr/bin/etcd \
--name k8-master03 \
--data-dir /var/lib/etcd \
--listen-client-urls https://192.168.20.22:2379 \
--advertise-client-urls https://192.168.20.22:2379 \
--listen-peer-urls https://192.168.20.22:2380 \
--initial-advertise-peer-urls https://192.168.20.22:2380 \
--cert-file=/etc/kubernetes/pki/etcd/server.pem \
--key-file=/etc/kubernetes/pki/etcd/server-key.pem \
--client-cert-auth \
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.pem \
--peer-key-file=/etc/kubernetes/pki/etcd/peer-key.pem \
--peer-client-cert-auth --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.pem \
--initial-cluster k8-master01=https://192.168.20.20:2380,k8-master02=https://192.168.20.21:2380,k8-
master03=https://192.168.20.22:2380 \
--initial-cluster-token my-etcd-token \
--initial-cluster-state new

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

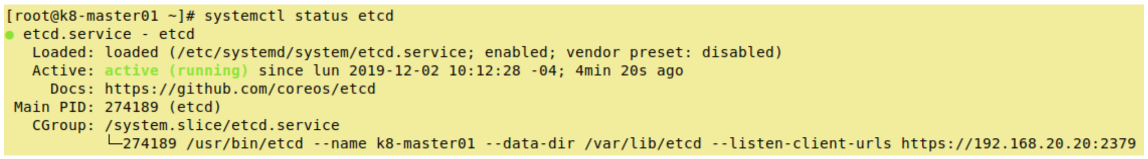
18) **Ejecutar los siguientes comandos en cada nodo master, comenzando por el nodo k8-master01 para iniciar el servicio etcd-k8s-master**

```
# systemctl daemon-reload && systemctl enable etcd
```

```
# systemctl start etcd
```

<u>Cuando se inicia el servicio con el comando start el master01 no emitirá respuesta hasta que algún otro nodo inicie el servicio etcd con el mismo comando start.</u>

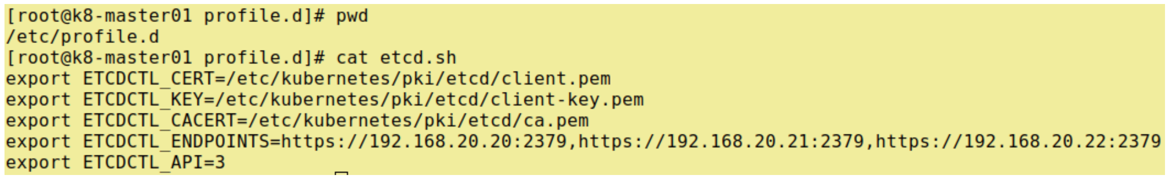
```
# systemctl status etcd
```



```
[root@k8-master01 ~]# systemctl status etcd
● etcd.service - etcd
   Loaded: loaded (/etc/systemd/system/etcd.service; enabled; vendor preset: disabled)
   Active: active (running) since lun 2019-12-02 10:12:28 -04; 4min 20s ago
     Docs: https://github.com/coreos/etcd
 Main PID: 274189 (etcd)
   CGroup: /system.slice/etcd.service
           └─274189 /usr/bin/etcd --name k8-master01 --data-dir /var/lib/etcd --listen-client-urls https://192.168.20.20:2379
```

19) **Configuración de variables de entorno para la administración básica de ETCD (ETCDCTL) en los tres (3) masters (todos), crear el archivo /etc/profile.d/etcd.sh con el siguiente contenido**

```
export ETCDCTL_CERT=/etc/kubernetes/pki/etcd/client.pem
export ETCDCTL_KEY=/etc/kubernetes/pki/etcd/client-key.pem
export ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.pem
export ETCDCTL_ENDPOINTS=https://192.168.20.20:2379,https://192.168.20.21:2379,https://
192.168.20.22:2379
export ETCDCTL_API=3
```

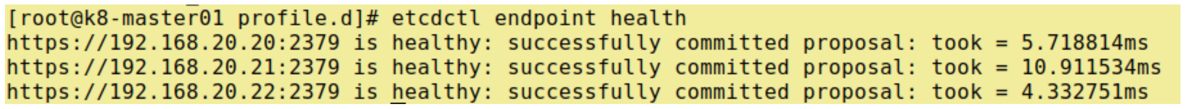


```
[root@k8-master01 profile.d]# pwd
/etc/profile.d
[root@k8-master01 profile.d]# cat etcd.sh
export ETCDCTL_CERT=/etc/kubernetes/pki/etcd/client.pem
export ETCDCTL_KEY=/etc/kubernetes/pki/etcd/client-key.pem
export ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.pem
export ETCDCTL_ENDPOINTS=https://192.168.20.20:2379,https://192.168.20.21:2379,https://192.168.20.22:2379
export ETCDCTL_API=3
```

NOTA: Para que el script **/etc/profile.d/etcd.sh** se ejecute debe cerrar la sesión con el servidor y conectarse nuevamente

Verificar la **salud** del **cluster ETCD** con el siguiente comando

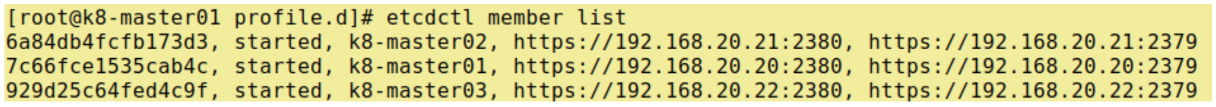
```
# etcdctl endpoint health
```



```
[root@k8-master01 profile.d]# etcdctl endpoint health
https://192.168.20.20:2379 is healthy: successfully committed proposal: took = 5.718814ms
https://192.168.20.21:2379 is healthy: successfully committed proposal: took = 10.911534ms
https://192.168.20.22:2379 is healthy: successfully committed proposal: took = 4.332751ms
```

Verificar los **miembros** del **cluster ETCD** con el siguiente comando

```
# etcdctl member list
```



```
[root@k8-master01 profile.d]# etcdctl member list
6a84db4fcfb173d3, started, k8-master02, https://192.168.20.21:2380, https://192.168.20.21:2379
7c66fce1535cab4c, started, k8-master01, https://192.168.20.20:2380, https://192.168.20.20:2379
929d25c64fed4c9f, started, k8-master03, https://192.168.20.22:2380, https://192.168.20.22:2379
```

20) **Configuración de balanceo de ETCD**

Se comienza con el nodo 1 master (**k8-master01**). Crear el directorio **/etc/kubernetes/configuration** y en el mismo directorio el archivo **config.yaml**

```
# mkdir /etc/kubernetes/configuration && cd /etc/kubernetes/configuration

# vim config.yaml
```

```
apiServer:
  certSANs:
  - 192.168.20.20
  extraArgs:
    apiserver-count: "3"
    authorization-mode: Node,RBAC
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta1
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controlPlaneEndpoint: ""
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  external:
    caFile: /etc/kubernetes/pki/etcd/ca.pem
    certFile: /etc/kubernetes/pki/etcd/client.pem
    endpoints:
    - https://192.168.20.20:2379
    - https://192.168.20.21:2379
    - https://192.168.20.22:2379
    keyFile: /etc/kubernetes/pki/etcd/client-key.pem
imageRepository: k8s.gcr.io
kind: ClusterConfiguration
kubernetesVersion: v1.15.4
networking:
  dnsDomain: cluster.local
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
scheduler: {}
```

Ejecutar el siguiente comando para aplicar lo configurado en el archivo **config.yaml**

```
# cd /etc/kubernetes/configuration

# kubeadm init --config=config.yaml
```

Resultado esperado

```
[root@k8-master01 configuration]# kubeadm init --config=config.yaml

[init] Using Kubernetes version: v1.15.4
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-
flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] External etcd mode: Skipping etcd/ca certificate authority generation
[certs] External etcd mode: Skipping etcd/server certificate authority generation
[certs] External etcd mode: Skipping etcd/healthcheck-client certificate authority generation
[certs] External etcd mode: Skipping etcd/peer certificate authority generation
[certs] External etcd mode: Skipping apiserver-etcd-client certificate authority generation
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [k8-master01 kubernetes kubernetes.default
kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 192.168.20.20
192.168.20.20]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
```

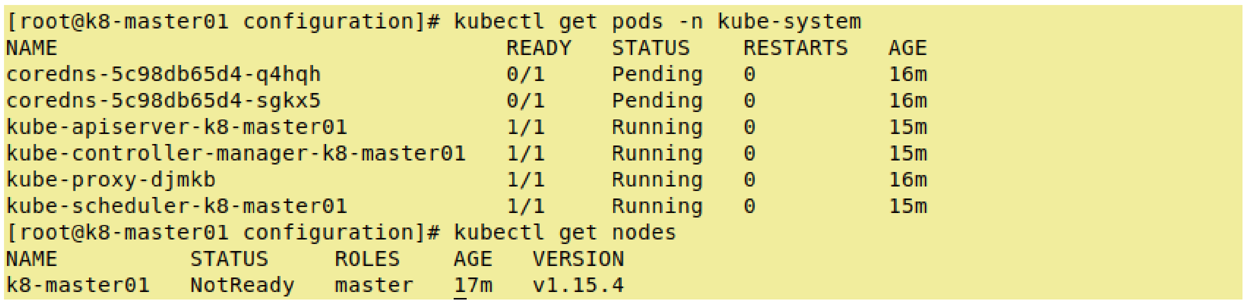
```
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory
"/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 26.507223 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system"
Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.15" in namespace kube-system with the configuration
for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node k8-master01 as control-plane by adding the label "node-
role.kubernetes.io/master=''"
[mark-control-plane] Marking the node k8-master01 as control-plane by adding the taints [node-
role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: y7xngl.gw80syc54qulhe93
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes
to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs
from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates
in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.20.20:6443 --token y7xngl.gw80syc54qulhe93 \
    --discovery-token-ca-cert-hash
sha256:50204506d189e72ad8391996c739a04c2088f7e9a528f0c5210f26f524d7b2ec
```

Tomar nota del token del cluster

```
kubeadm join 192.168.20.20:6443 --token y7xngl.gw80syc54qulhe93 \
    --discovery-token-ca-cert-hash
sha256:50204506d189e72ad8391996c739a04c2088f7e9a528f0c5210f26f524d7b2ec
```

Ejecutar los siguientes comandos en el nodo 1 master (k8-master01)

```
# mkdir -p $HOME/.kube && cp -i /etc/kubernetes/admin.conf $HOME/.kube/config &&
chown $(id -u):$(id -g) $HOME/.kube/config

# kubectl get pods -n kube-system

# kubectl get nodes
```



```
[root@k8-master01 configuration]# kubectl get pods -n kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-q4hqh                0/1     Pending   0          16m
coredns-5c98db65d4-sgkx5                0/1     Pending   0          16m
kube-apiserver-k8-master01              1/1     Running   0          15m
kube-controller-manager-k8-master01     1/1     Running   0          15m
kube-proxy-djmkb                        1/1     Running   0          16m
kube-scheduler-k8-master01              1/1     Running   0          15m
[root@k8-master01 configuration]# kubectl get nodes
NAME          STATUS     ROLES    AGE   VERSION
k8-master01   NotReady   master   17m   v1.15.4
```

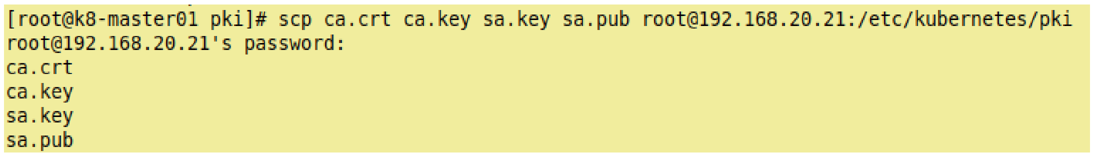
Copiar en los nodos master 2 y 3 (**k8-master02, k8-master03**) en el directorio **/etc/kubernetes/pki/** desde el nodo master 1 los siguientes archivos:

```
/etc/kubernetes/pki/ca.crt
/etc/kubernetes/pki/ca.key
/etc/kubernetes/pki/sa.key
/etc/kubernetes/pki/sa.pub
```

Con los siguientes comandos desde el nodo 1 master (k8-master01):

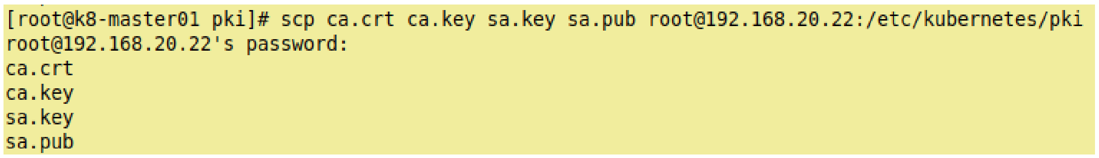
```
# cd /etc/kubernetes/pki
```

```
# scp ca.crt ca.key sa.key sa.pub root@192.168.20.21:/etc/kubernetes/pki
```



```
[root@k8-master01 pki]# scp ca.crt ca.key sa.key sa.pub root@192.168.20.21:/etc/kubernetes/pki
root@192.168.20.21's password:
ca.crt
ca.key
sa.key
sa.pub
```

```
# scp ca.crt ca.key sa.key sa.pub root@192.168.20.22:/etc/kubernetes/pki
```



```
[root@k8-master01 pki]# scp ca.crt ca.key sa.key sa.pub root@192.168.20.22:/etc/kubernetes/pki
root@192.168.20.22's password:
ca.crt
ca.key
sa.key
sa.pub
```

De igual forma que el master 01, crear el directorio **/etc/kubernetes/configuration** en los nodos master 2 y 3 (**k8-master02, k8-master03**) y copiar el archivo **config.yaml** desde el nodo master 1 (k8-master01)

k8-master02:

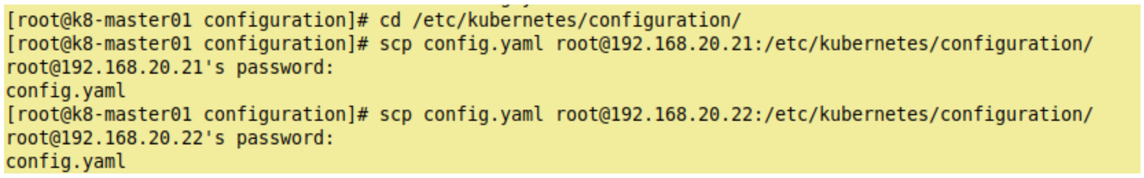
```
# mkdir /etc/kubernetes/configuration
```

k8-master03:

```
# mkdir /etc/kubernetes/configuration
```

k8-master01:

```
# cd /etc/kubernetes/configuration
# scp config.yaml root@192.168.20.21:/etc/kubernetes/configuration/

# scp config.yaml root@192.168.20.22:/etc/kubernetes/configuration/
```



```
[root@k8-master01 configuration]# cd /etc/kubernetes/configuration/
[root@k8-master01 configuration]# scp config.yaml root@192.168.20.21:/etc/kubernetes/configuration/
root@192.168.20.21's password:
config.yaml
[root@k8-master01 configuration]# scp config.yaml root@192.168.20.22:/etc/kubernetes/configuration/
root@192.168.20.22's password:
config.yaml
```

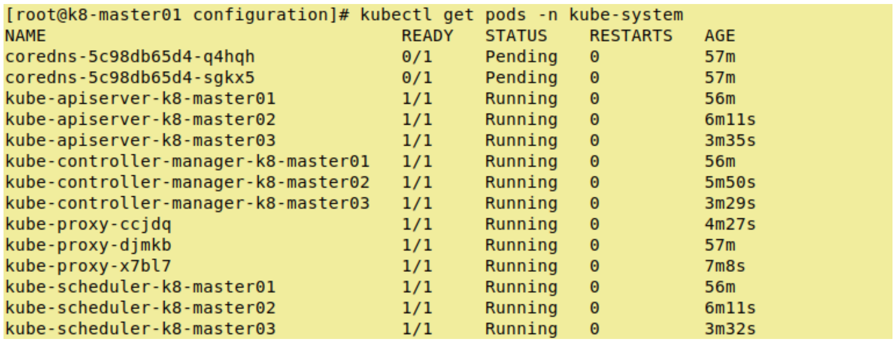
Ejecutar los siguientes comandos en los nodos master 2 y 3 (k8-master02, k8-master03) para iniciar kubeadm

```
# kubeadm init --config=config.yaml

# mkdir -p $HOME/.kube && cp -i /etc/kubernetes/admin.conf $HOME/.kube/config &&
chown $(id -u):$(id -g) $HOME/.kube/config
```

Verificar en los tres (3) nodos master los pods de kubernetes ejecutando el siguiente comando:

```
# kubectl get pods -n kube-system
```

Resultado esperado:

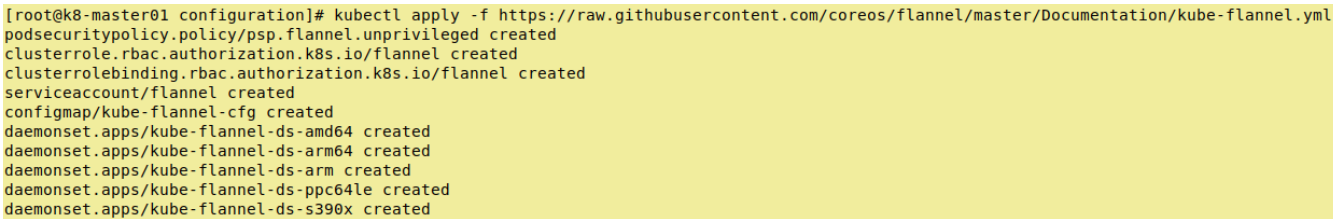


```
[root@k8-master01 configuration]# kubectl get pods -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-q4hqh              0/1     Pending   0          57m
coredns-5c98db65d4-sgkx5              0/1     Pending   0          57m
kube-apiserver-k8-master01            1/1     Running   0          56m
kube-apiserver-k8-master02            1/1     Running   0          6m11s
kube-apiserver-k8-master03            1/1     Running   0          3m35s
kube-controller-manager-k8-master01   1/1     Running   0          56m
kube-controller-manager-k8-master02   1/1     Running   0          5m50s
kube-controller-manager-k8-master03   1/1     Running   0          3m29s
kube-proxy-ccjdq                      1/1     Running   0          4m27s
kube-proxy-djmkb                      1/1     Running   0          57m
kube-proxy-x7bl7                      1/1     Running   0          7m8s
kube-scheduler-k8-master01            1/1     Running   0          56m
kube-scheduler-k8-master02            1/1     Running   0          6m11s
kube-scheduler-k8-master03            1/1     Running   0          3m32s
```

21) **Instalar la red de kubernetes “Flannel”**

En el nodo 1 master (k8-master01)

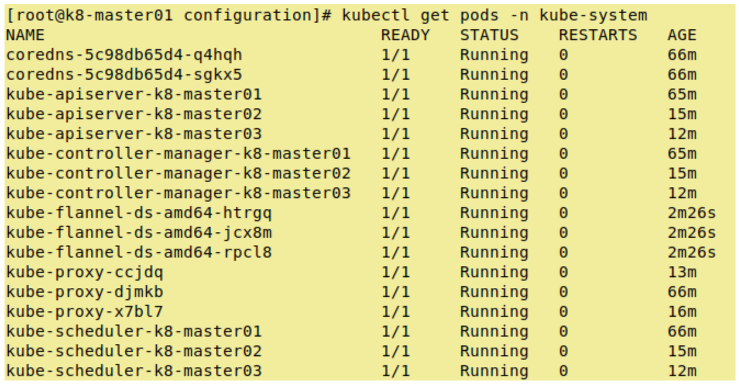
```
# kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```



```
[root@k8-master01 configuration]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds-amd64 created
daemonset.apps/kube-flannel-ds-arm64 created
daemonset.apps/kube-flannel-ds-arm created
daemonset.apps/kube-flannel-ds-ppc64le created
daemonset.apps/kube-flannel-ds-s390x created
```

Ejecutar el siguiente comando para verificar que los pods “coredns” tengan el status “running”

```
# kubectl get pods -n kube-system
```



```
[root@k8-master01 configuration]# kubectl get pods -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-q4hqh               1/1     Running   0          66m
coredns-5c98db65d4-sgkx5               1/1     Running   0          66m
kube-apiserver-k8-master01             1/1     Running   0          65m
kube-apiserver-k8-master02             1/1     Running   0          15m
kube-apiserver-k8-master03             1/1     Running   0          12m
kube-controller-manager-k8-master01    1/1     Running   0          65m
kube-controller-manager-k8-master02    1/1     Running   0          15m
kube-controller-manager-k8-master03    1/1     Running   0          12m
kube-flannel-ds-amd64-htrgq            1/1     Running   0          2m26s
kube-flannel-ds-amd64-jcx8m            1/1     Running   0          2m26s
kube-flannel-ds-amd64-rpcl8            1/1     Running   0          2m26s
kube-proxy-ccjdq                       1/1     Running   0          13m
kube-proxy-djmkb                       1/1     Running   0          66m
kube-proxy-x7bl7                       1/1     Running   0          16m
kube-scheduler-k8-master01             1/1     Running   0          66m
kube-scheduler-k8-master02             1/1     Running   0          15m
kube-scheduler-k8-master03             1/1     Running   0          12m
```

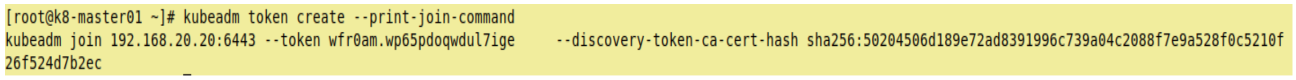
Si el comando anterior se ejecuta desde los nodos master 2 y 3 el resultado debe ser el mismo.

22) **Unir los nodos workers al cluster con el comando JOIN**

NOTA IMPORTANTE: Si el tiempo transcurrido entre la ejecución del comando “**kubeadm init –config=config.yaml**” el cual generó un token para ser usado con el comando “kubeadm join ...” es superior a 24 horas se debe generar un nuevo token ya que los tokens expiran a las 24 horas de haber sido generados

node 01 master:

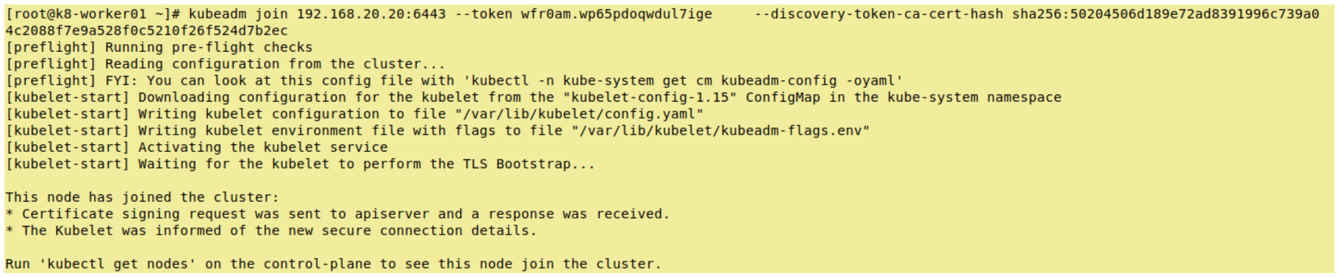
```
# kubeadm token create --print-join-command
```



```
[root@k8-master01 ~]# kubeadm token create --print-join-command
kubeadm join 192.168.20.20:6443 --token wfr0am.wp65pdoqwdul7ige     --discovery-token-ca-cert-hash sha256:50204506d189e72ad8391996c739a04c2088f7e9a528f0c5210f
26f524d7b2ec
```

node workers (todos):

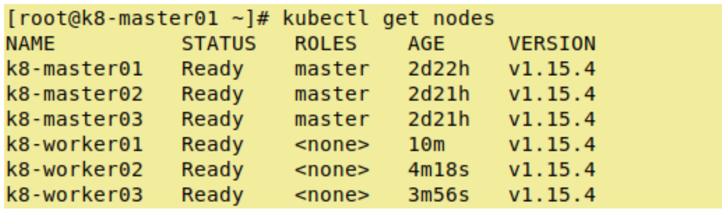
```
# kubeadm join 192.168.20.20:6443 --token wfr0am.wp65pdoqwdul7ige \
--discovery-token-ca-cert-hash
sha256:50204506d189e72ad8391996c739a04c2088f7e9a528f0c5210f26f524d7b2ec
```



```
[root@k8-worker01 ~]# kubeadm join 192.168.20.20:6443 --token wfr0am.wp65pdoqwdul7ige     --discovery-token-ca-cert-hash sha256:50204506d189e72ad8391996c739a0
4c2088f7e9a528f0c5210f26f524d7b2ec
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Ejecutar el siguiente comando (en el nodo 1 master) para verificar la incorporación de los nodos workers al cluster:

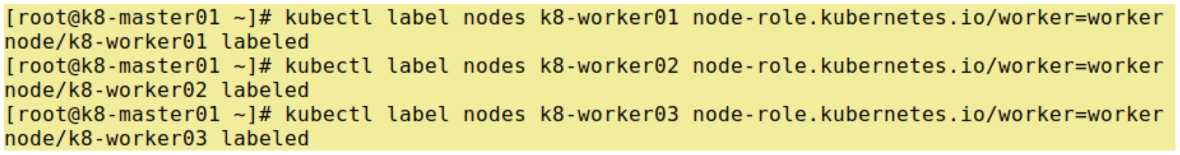
```
# kubectl get nodes
```



```
[root@k8-master01 ~]# kubectl get nodes
NAME          STATUS   ROLES    AGE     VERSION
k8-master01   Ready    master   2d22h   v1.15.4
k8-master02   Ready    master   2d21h   v1.15.4
k8-master03   Ready    master   2d21h   v1.15.4
k8-worker01   Ready    <none>   10m     v1.15.4
k8-worker02   Ready    <none>   4m18s   v1.15.4
k8-worker03   Ready    <none>   3m56s   v1.15.4
```

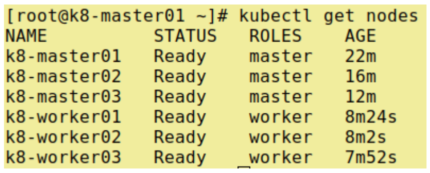
Etiquetar el “ROLE” de los workers ya que por defecto la etiqueta “**ROLES**” en los NO master es “<**none**>”

```
# kubectl label nodes k8-worker01 node-role.kubernetes.io/worker=worker
# kubectl label nodes k8-worker02 node-role.kubernetes.io/worker=worker
# kubectl label nodes k8-worker03 node-role.kubernetes.io/worker=worker
```



```
[root@k8-master01 ~]# kubectl label nodes k8-worker01 node-role.kubernetes.io/worker=worker
node/k8-worker01 labeled
[root@k8-master01 ~]# kubectl label nodes k8-worker02 node-role.kubernetes.io/worker=worker
node/k8-worker02 labeled
[root@k8-master01 ~]# kubectl label nodes k8-worker03 node-role.kubernetes.io/worker=worker
node/k8-worker03 labeled
```

```
# kubectl get nodes
```



```
[root@k8-master01 ~]# kubectl get nodes
NAME          STATUS   ROLES    AGE
k8-master01   Ready    master   22m
k8-master02   Ready    master   16m
k8-master03   Ready    master   12m
k8-worker01   Ready    worker   8m24s
k8-worker02   Ready    worker   8m2s
k8-worker03   Ready    worker   7m52s
```

23) **Instalar en dashboard de kubernetes (solo nodo 1 master)**

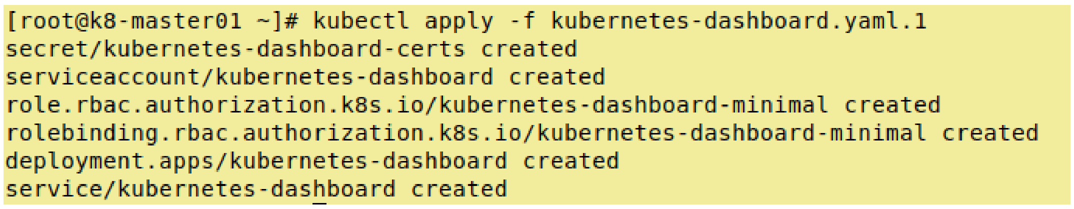
```
# wget https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.0/src/deploy/
recommended/kubernetes-dashboard.yaml

# cat kubernetes-dashboard.yaml
```

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

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

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

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

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

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

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

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

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

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

kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          # Uncomment the following line to manually specify Kubernetes API server Host
          # If not specified, Dashboard will attempt to auto discover the API server and connect
          # to it. Uncomment only if the default does not work.
          # - --apiserver-host=http://my-address:port
        volumeMounts:
        - name: kubernetes-dashboard-certs
          mountPath: /certs
          # Create on-disk volume to store exec logs
        - mountPath: /tmp
          name: tmp-volume
        livenessProbe:
          httpGet:
            scheme: HTTPS
            path: /
            port: 8443
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: kubernetes-dashboard-certs
        secret:
          secretName: kubernetes-dashboard-certs
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      # Comment the following tolerations if Dashboard must not be deployed on master
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule

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

kind: Service
```

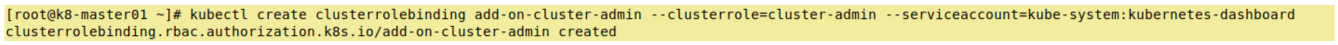
```
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  ports:
    - port: 443
      targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
```

```
# kubectl apply -f kubernetes-dashboard.yaml
```



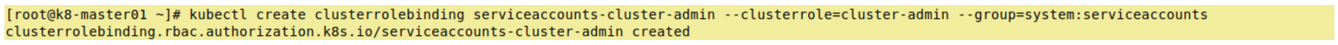
```
[root@k8-master01 ~]# kubectl apply -f kubernetes-dashboard.yaml.1
secret/kubernetes-dashboard-certs created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
```

```
# kubectl create clusterrolebinding add-on-cluster-admin --clusterrole=cluster-
admin --serviceaccount=kube-system:kubernetes-dashboard
```



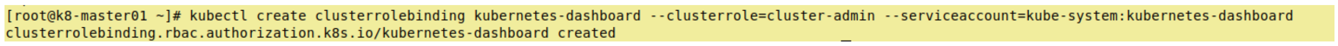
```
[root@k8-master01 ~]# kubectl create clusterrolebinding add-on-cluster-admin --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
clusterrolebinding.rbac.authorization.k8s.io/add-on-cluster-admin created
```

```
# kubectl create clusterrolebinding serviceaccounts-cluster-admin --
clusterrole=cluster-admin --group=system:serviceaccounts
```



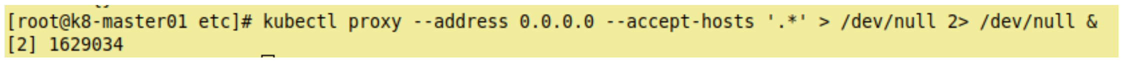
```
[root@k8-master01 ~]# kubectl create clusterrolebinding serviceaccounts-cluster-admin --clusterrole=cluster-admin --group=system:serviceaccounts
clusterrolebinding.rbac.authorization.k8s.io/serviceaccounts-cluster-admin created
```

```
# kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-
admin --serviceaccount=kube-system:kubernetes-dashboard
```



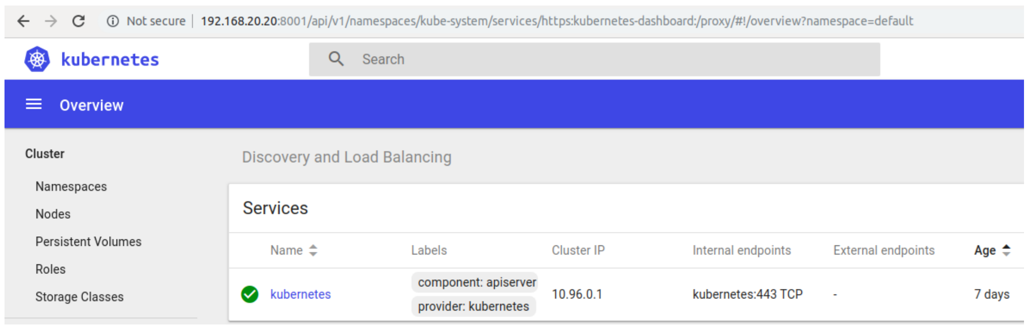
```
[root@k8-master01 ~]# kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
```

```
# kubectl proxy --address 0.0.0.0 --accept-hosts '.*' > /dev/null 2> /dev/null &
```



```
[root@k8-master01 etc]# kubectl proxy --address 0.0.0.0 --accept-hosts '.*' > /dev/null 2> /dev/null &
[2] 1629034
```

Ingresar a la url http://192.168.20.20:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/ y hacer click en el enlace “**skip**”



192.168.20.20:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=default

Discovery and Load Balancing

Services

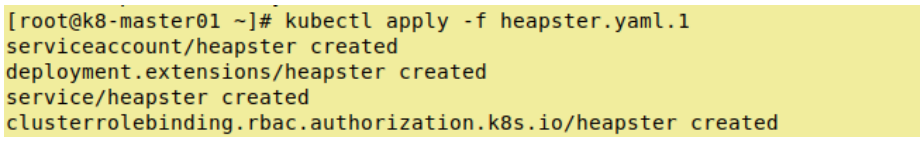
Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
✓ kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP	-	7 days

24) **Instalar herramienta de monitoreo HEAPSTER en el dashboard de kubernetes**

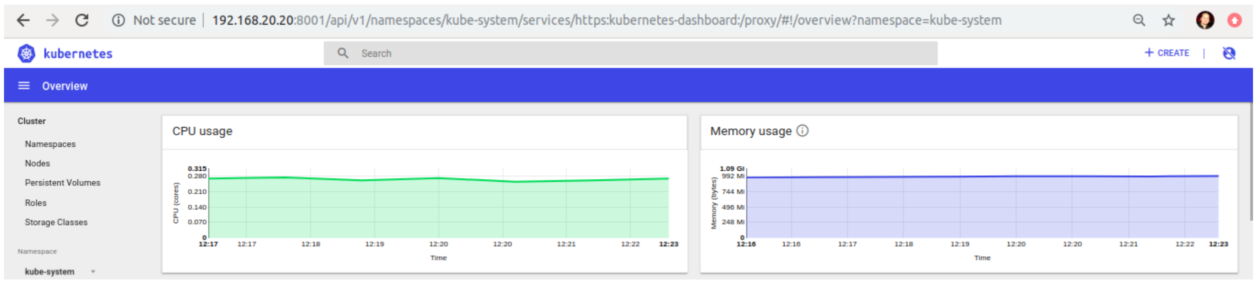
```
# vim heapster.yaml
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: heapster
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: heapster
    spec:
      serviceAccountName: heapster
      containers:
      - name: heapster
        image: gcr.io/google_containers/heapster-amd64:v1.4.2
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes.summary_api:''?
useServiceAccount=true&kubeletHttps=true&kubeletPort=10250&insecure=true
---
apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: Heapster
  name: heapster
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    k8s-app: heapster
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: heapster
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:heapster
subjects:
- kind: ServiceAccount
  name: heapster
  namespace: kube-system
```

```
# kubectl apply -f heapster.yaml
```



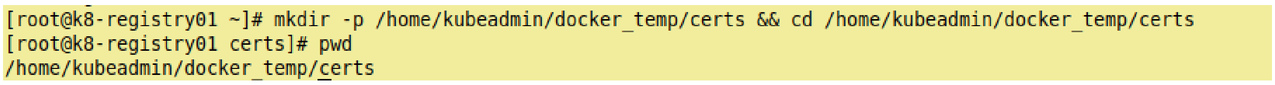
```
[root@k8-master01 ~]# kubectl apply -f heapster.yaml.1
serviceaccount/heapster created
deployment.extensions/heapster created
service/heapster created
clusterrolebinding.rbac.authorization.k8s.io/heapster created
```



Configuración del contenedor Registry para el repositorio de imágenes

1) Crear certificado autofirmado para el servidor de imágenes (registry) en el directorio /home/kubeadmin/docker_temp/certs

```
# mkdir -p /home/kubeadmin/docker_temp/certs && cd
/home/kubeadmin/docker_temp/certs
```



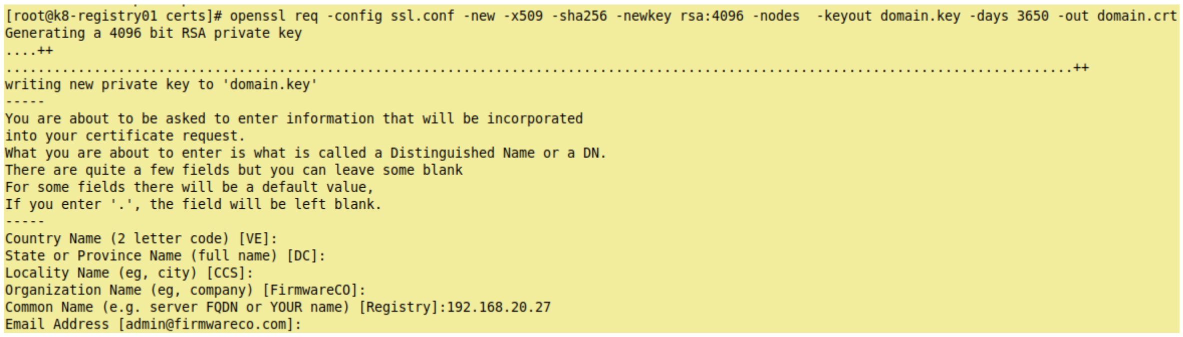
```
[root@k8-registry01 ~]# mkdir -p /home/kubeadmin/docker_temp/certs && cd /home/kubeadmin/docker_temp/certs
[root@k8-registry01 certs]# pwd
/home/kubeadmin/docker_temp/certs
```

Crear el archivo ssl.conf con el siguiente contenido:

```
# vim ssl.conf

# Self Signed (note the addition of -x509):
#     openssl req -config example-com.conf -new -x509 -sha256 -newkey rsa:2048 -nodes -keyout example-
com.key.pem -days 365 -out example-com.cert.pem
# Signing Request (note the lack of -x509):
#     openssl req -config example-com.conf -new -newkey rsa:2048 -nodes -keyout example-com.key.pem -
days 365 -out example-com.req.pem
# Print it:
#     openssl x509 -in example-com.cert.pem -text -noout
#     openssl req -in example-com.req.pem -text -noout

[ req ]
default_bits          = 4096
default_keyfile       = server-key.pem
distinguished_name    = subject
req_extensions        = req_ext
x509_extensions       = x509_ext
string_mask           = utf8only
# The Subject DN can be formed using X501 or RFC 4514 (see RFC 4519 for a description).
#   Its sort of a mashup. For example, RFC 4514 does not provide emailAddress.
```

```
[ subject ]
countryName         = VE
countryName_default = VE
stateOrProvinceName = DC
stateOrProvinceName_default = CCS
organizationUnit    = FIRMCO
organizationUnit_default    = TEC

localityName        = CCS
localityName_default = CCS

organizationName     = FIRMCO
organizationName_default  = FIRMCO

# Use a friendly name here because its presented to the user. The server's DNS
#   names are placed in Subject Alternate Names. Plus, DNS names here is deprecated
#   by both IETF and CA/Browser Forums. If you place a DNS name here, then you
#   must include the DNS name in the SAN too (otherwise, Chrome and others that
#   strictly follow the CA/Browser Baseline Requirements will fail).
commonName         = Registry
commonName_default = Registry

emailAddress       = admin@firmwareco.com
emailAddress_default  = admin@firmwareco.com

# Section x509_ext is used when generating a self-signed certificate. I.e., openssl req -x509 ...
#  If RSA Key Transport bothers you, then remove keyEncipherment. TLS 1.3 is removing RSA
#  Key Transport in favor of exchanges with Forward Secrecy, like DHE and ECDHE.
[ x509_ext ]

subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer

basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName      = IP:192.168.20.27
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
# CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me confused
# extendedKeyUsage  = serverAuth, clientAuth

# Section req_ext is used when generating a certificate signing request. I.e., openssl req ...
[ req_ext ]

subjectKeyIdentifier    = hash

basicConstraints    = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName      = IP:192.168.20.27
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
# CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me confused
# extendedKeyUsage  = serverAuth, clientAuth

[ alternate_names ]

DNS.1       = example.com
DNS.2       = www.example.com
DNS.3       = mail.example.com
DNS.4       = ftp.example.com
```

```
# openssl req -config ssl.conf -new -x509 -sha256 -newkey rsa:4096 -nodes  -keyout
domain.key -days 3650 -out domain.crt
```



```
[root@k8-registry01 certs]# openssl req -config ssl.conf -new -x509 -sha256 -newkey rsa:4096 -nodes  -keyout domain.key -days 3650 -out domain.crt
Generating a 4096 bit RSA private key
....++
..........................................................................................................................++
writing new private key to 'domain.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [VE]:
State or Province Name (full name) [DC]:
Locality Name (eg, city) [CCS]:
Organization Name (eg, company) [FirmwareCO]:
Common Name (e.g. server FQDN or YOUR name) [Registry]:192.168.20.27
Email Address [admin@firmwareco.com]:
```

El comando anterior genera los archivos **domain.crt** y domain.key

2) **Crear en todos los nodos incluyendo el mismo servidor de imágenes docker registry (masters, workers y registry) el directorio /etc/docker/certs.d/192.168.20.27:4443/ donde 192.168.20.27 es la IP del servidor de imágenes Docker (registry) y 4443 es el puerto configurado para escuchar**

```
# mkdir -p /etc/docker/certs.d/192.168.20.27:4443/
```

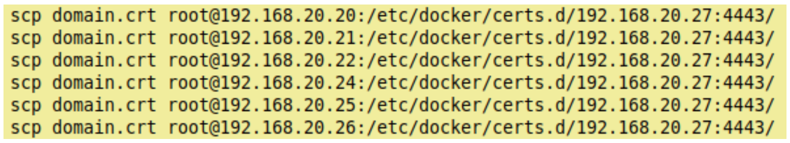
Luego copiar el certificado generado en el punto anterior **domain.crt** en el directorio creado

En el servidor registry:

```
# cp /home/kubeadmin/docker_temp/certs/domain.crt
/etc/docker/certs.d/192.168.20.27:4443/
```

Desde el servidor registry a los nodos masters y workers:

```
# scp domain.crt root@192.168.20.20:/etc/docker/certs.d/192.168.20.27:4443/
# scp domain.crt root@192.168.20.21:/etc/docker/certs.d/192.168.20.27:4443/
# scp domain.crt root@192.168.20.22:/etc/docker/certs.d/192.168.20.27:4443/
# scp domain.crt root@192.168.20.24:/etc/docker/certs.d/192.168.20.27:4443/
# scp domain.crt root@192.168.20.25:/etc/docker/certs.d/192.168.20.27:4443/
# scp domain.crt root@192.168.20.26:/etc/docker/certs.d/192.168.20.27:4443/
```

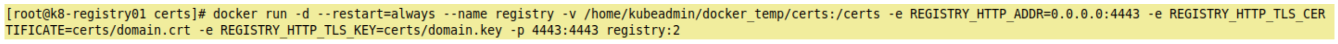


```
scp domain.crt root@192.168.20.20:/etc/docker/certs.d/192.168.20.27:4443/
scp domain.crt root@192.168.20.21:/etc/docker/certs.d/192.168.20.27:4443/
scp domain.crt root@192.168.20.22:/etc/docker/certs.d/192.168.20.27:4443/
scp domain.crt root@192.168.20.24:/etc/docker/certs.d/192.168.20.27:4443/
scp domain.crt root@192.168.20.25:/etc/docker/certs.d/192.168.20.27:4443/
scp domain.crt root@192.168.20.26:/etc/docker/certs.d/192.168.20.27:4443/
```

3) **Crear el contenedor registry para el repositorio de imágenes, antes, verificar si no existe ya un contenedor registry**

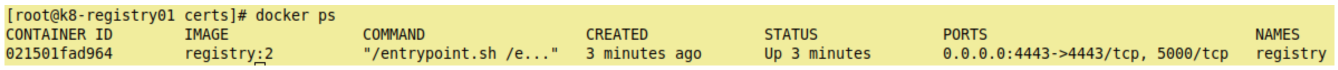
NOTA IMPORTANTE: Ejecutar el comando de creación del contenedor “registry” desde el directorio **/home/kubeadmin/docker_temp/**

```
# cd /home/kubeadmin/docker_temp/
# docker run -d --restart=always --name registry -v `pwd`/certs:/certs -e
REGISTRY_HTTP_ADDR=0.0.0.0:4443 -e REGISTRY_HTTP_TLS_CERTIFICATE=certs/domain.crt -
e REGISTRY_HTTP_TLS_KEY=certs/domain.key -p 4443:4443 registry:2
```



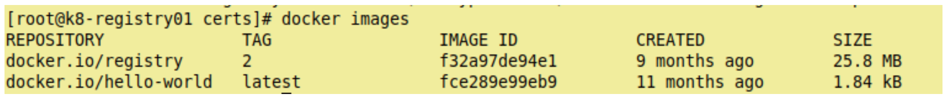
```
[root@k8-registry01 certs]# docker run -d --restart=always --name registry -v /home/kubeadmin/docker_temp/certs:/certs -e REGISTRY_HTTP_ADDR=0.0.0.0:4443 -e REGISTRY_HTTP_TLS_CER
TIFICATE=certs/domain.crt -e REGISTRY_HTTP_TLS_KEY=certs/domain.key -p 4443:4443 registry:2
```

```
# docker ps
```



```
[root@k8-registry01 certs]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                              NAMES
021501fad964        registry:2          "/entrypoint.sh /e..."   3 minutes ago       Up 3 minutes        0.0.0.0:4443->4443/tcp, 5000/tcp   registry
```

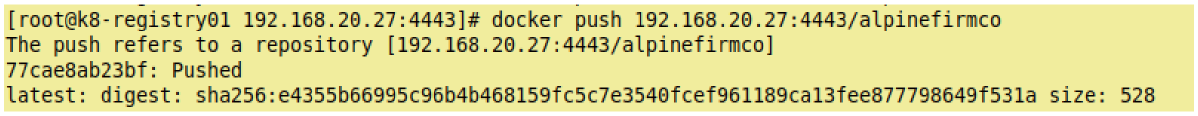
```
# docker images
```



```
[root@k8-registry01 certs]# docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
docker.io/registry      2                   f32a97de94e1        9 months ago        25.8 MB
docker.io/hello-world   latest              fce289e99eb9        11 months ago       1.84 kB
```

<u>Descargar la imagen “alpine”, tagearla y pushearla para publicarla en el registry privado:</u>

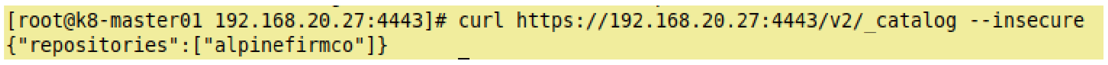
```
# docker pull alpine

# docker tag alpine 192.168.20.27:4443/alpinefirmco

# docker push 192.168.20.27:4443/alpinefirmco
```



```
[root@k8-registry01 192.168.20.27:4443]# docker push 192.168.20.27:4443/alpinefirmco
The push refers to a repository [192.168.20.27:4443/alpinefirmco]
77cae8ab23bf: Pushed
latest: digest: sha256:e4355b66995c96b4b468159fc5c7e3540fcef961189ca13fee877798649f531a size: 528
```

<u>Verificar el catálogo de imágenes disponibles</u>:

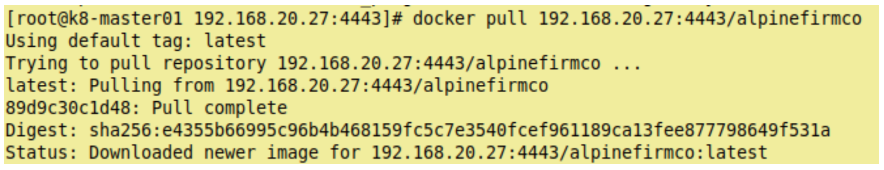
```
# curl https://192.168.20.27:4443/v2/_catalog --insecure
```



```
[root@k8-master01 192.168.20.27:4443]# curl https://192.168.20.27:4443/v2/_catalog --insecure
{"repositories":["alpinefirmco"]}
```

<u>Pullear la imágen disponible en el catálogo desde cualquier nodo del cluster kubernetes para probar conexión y configuración</u>:

```
# docker pull 192.168.20.27:4443/alpinefirmco
```



```
[root@k8-master01 192.168.20.27:4443]# docker pull 192.168.20.27:4443/alpinefirmco
Using default tag: latest
Trying to pull repository 192.168.20.27:4443/alpinefirmco ...
latest: Pulling from 192.168.20.27:4443/alpinefirmco
89d9c30c1d48: Pull complete
Digest: sha256:e4355b66995c96b4b468159fc5c7e3540fcef961189ca13fee877798649f531a
Status: Downloaded newer image for 192.168.20.27:4443/alpinefirmco:latest
```