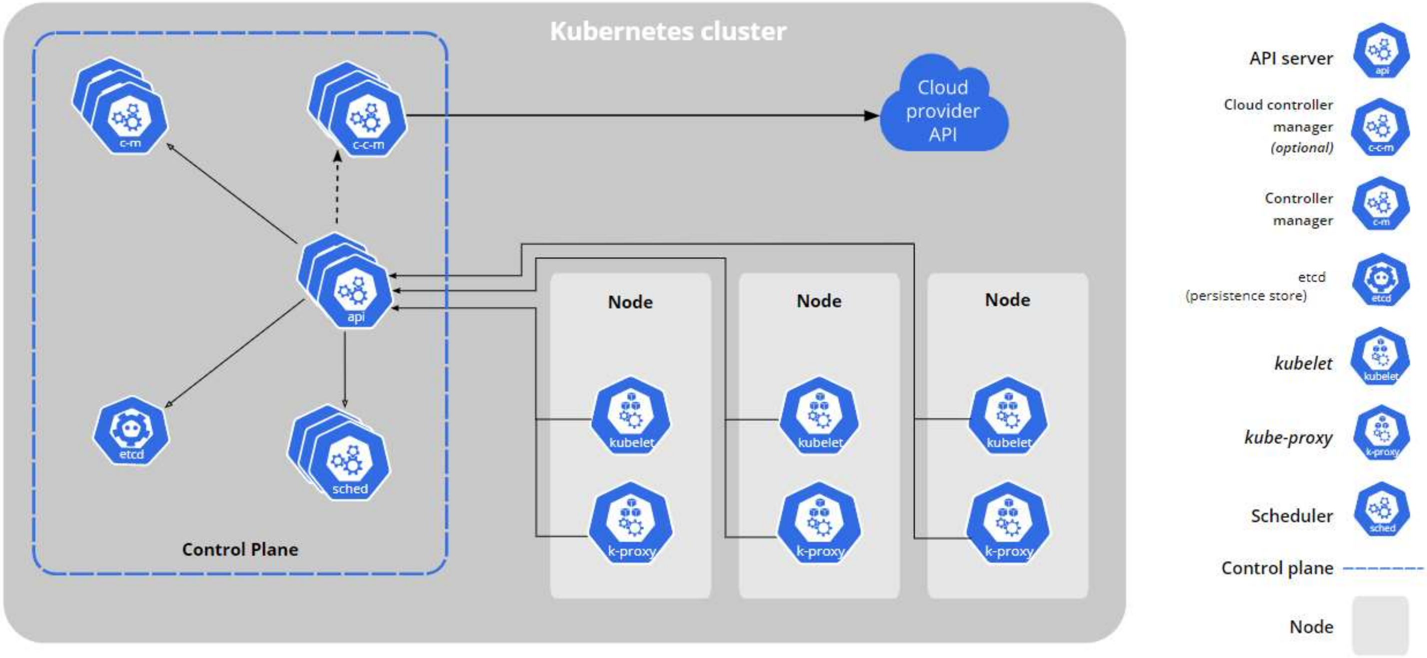
Kubernetes Hardway

Monday, January 4, 2021 9:30 PM



Step 1:

Create your VMs:
Ensure that the VMs can talk to each other.
Make on Master VM the "leader" that you can use to ssh into other VMs.
…follow ssh procedure in VM Linux SSH/Log-On.
Also ensure that the host files have the right IP address mapping:

```
> sudo vi /etc/hosts
```

Step 2:

Install Kubectl (on Master node)
https://kubernetes.io/docs/tasks/tools/install-kubectl/

a: Download the latest release with the command:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/<version>/bin/linux/amd64/kubectl
```

e.g-

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubectl
```

b: Make the kubectl binary executable.

```
chmod +x ./kubectl
```

c: Move the binary in to your PATH.

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

d: Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

===

=========MASTER/CONTROL PLANE==============================

Step 3:

Security - Certificate Authority/ Generation (More Master/Control Plane)

Here, we determine and identify and Administrative Client. This Client will be used to perform Administrative tasks, eg Creating Certificates, Generating Config files and distributing to other nodes.
We will use the Master-1 as our Admin Client. This system must have SSH access to other systems so that it's easy to copy and move files.

a: Generate Certificate Authority on the Admin Client (Master-1 in this case):
https://kubernetes.io/docs/concepts/cluster-administration/certificates/#openssl

# Create/Generate private key for CA
```
openssl genrsa -out ca.key 2048
```

> # (Optional) Go to the following file and verify that *[RANDFILE = $ENV::HOME/.rnd*] is commented off.
> This is to avoid potential permission issues.
> ```
> sudo vi /etc/ssl/openssl.cnf
> ```

# Create/Generate Certificate Signing Request (CSR) using the private key created above.
```
openssl req -new -key ca.key -subj "/CN=KUBERNETES-CA" -out ca.csr
```

# Self sign the csr using its own private key
```
openssl x509 -req -in ca.csr -signkey ca.key -CAcreateserial  -out ca.crt -days 1000
```

Expected files in Folder:
ca.crt
ca.key

b: Create Control Plane Certificates.

I. Create the following Client and Server Certificates (in $HOME folder)

- Admin Client Certificate:
# Generate private key for admin user
```
openssl genrsa -out admin.key 2048
```

# Generate CSR for admin user using private key above. (Note the /O=).
```
openssl req -new -key admin.key -subj "/CN=admin/O=system:masters" -out admin.csr
```

# Sign certificate for admin user using CA servers private key
```
openssl x509 -req -in admin.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out admin.crt -days 1000
```

Expected files in Folder:
admin.key
admin.crt

Note: The admin.crt and admin.key file gives you administrative access. We will configure these to be used with the kubectl tool to perform administrative functions on kubernetes.

- Controller Manager Client Certificate:
# Generate private key for Controller Manager
```
openssl genrsa -out kube-controller-manager.key 2048
```

# Generate CSR for Controller Manager using private key above
```
openssl req -new -key kube-controller-manager.key -subj "/CN=system:kube-controller-manager" -out kube-controller-manager.csr
```

# Sign certificate for Controller Manager using CA servers private key
```
openssl x509 -req -in kube-controller-manager.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out kube-controller-
```

```
manager.crt -days 1000
```

Expected files in Folder:

```
        kube-controller-manager.key
        kube-controller-manager.crt
```

<u>- Kube Proxy Client Certificate:</u>

```
# Generate private key for kube-proxy
openssl genrsa -out kube-proxy.key 2048

# Generate CSR for kube-proxy using private key above
openssl req -new -key kube-proxy.key -subj "/CN=system:kube-proxy" -out kube-proxy.csr

# Sign certificate for kube-proxy using CA servers private key
openssl x509 -req -in kube-proxy.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out kube-proxy.crt -days 1000
```

Expected files in Folder:

```
        kube-proxy.key
        kube-proxy.crt
```

<u>- Scheduler Client Certificate</u>

```
# Generate private key for kube-scheduler
openssl genrsa -out kube-scheduler.key 2048

# Generate CSR for kube-scheduler using private key above
openssl req -new -key kube-scheduler.key -subj "/CN=system:kube-scheduler" -out kube-scheduler.csr

# Sign certificate for kube-scheduler using CA servers private key
openssl x509 -req -in kube-scheduler.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out kube-scheduler.crt -days 1000
```

Expected files in Folder:

```
        kube-scheduler.key
        kube-scheduler.crt
```

<u>- Kubelet Client Certificates (for worker nodes, can skip for now)</u>

```
# We will do Certificates for Kubelet Clients later
```

<u>- Kube API Server Certificate:</u>

We have to use a `conf` file to create certificate for Kube API server because we are going to have more than one alt_name and openssl command cannot process this.

```
cat > openssl-k8sapi.cnf <<EOF
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
IP.1 = 10.96.0.1 # Default IP for Kube
IP.2 = 192.168.7.145 # Master-1 IP
IP.3 = 192.168.7.146 # Master-2 IP
IP.4 = 192.168.7.147 # Load Balancer IP
IP.5 = 127.0.0.1 # Local-host
EOF
```

```
# Generate private key for kube-Apiserver
openssl genrsa -out kube-apiserver.key 2048
```

```
# Generate CSR for kube-Apiserver using private key above
openssl req -new -key kube-apiserver.key -subj "/CN=kube-apiserver" -out kube-apiserver.csr -config openssl-k8sapi.cnf
```

```
# Sign certificate for kube-Apiserver using CA servers private key
openssl x509 -req -in kube-apiserver.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out kube-apiserver.crt -extensions v3_req -extfile openssl-k8sapi.cnf -days 1000
```

Expected files in Folder:

```
        kube-apiserver.crt
        kube-apiserver.key
```

- ETCD Server Certificate:

We have to use a `conf` file to create certificate for ETCD server because we are going to have more than one alt_name and openssl command cannot process this.

```
cat > openssl-etcd.cnf <<EOF
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
IP.1 = 192.168.7.145 # Master-1 IP
IP.2 = 192.168.7.146 # Master-2 IP
IP.3 = 127.0.0.1
EOF
```

```
# Generate private key for ETCD Server
openssl genrsa -out etcd-server.key 2048
```

```
# Generate CSR for ETCD Server using private key above
openssl req -new -key etcd-server.key -subj "/CN=etcd-server" -out etcd-server.csr -config openssl-etcd.cnf
```

```
# Sign certificate for ETCD Server using CA servers private key
openssl x509 -req -in etcd-server.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out etcd-server.crt -extensions v3_req -extfile openssl-etcd.cnf -days 1000
```

Expected files in Folder:

```
        etcd-server.key
        etcd-server.crt
```

- Service Account Key Pair Certificates:

```
# Generate private key for Service Account Key Pair
openssl genrsa -out service-account.key 2048

# Generate CSR for Service Account Key Pair using private key above
openssl req -new -key service-account.key -subj "/CN=service-accounts" -out service-account.csr

# Sign certificate for Service Account Key Pair using CA servers private key
openssl x509 -req -in service-account.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out service-account.crt -days 1000
```

Expected files in Folder:

```
service-account.key
service-account.crt
```

c: Distribute/Copy to other Machines.

After All certificates have been created, distribute (copy) them to other Master nodes as needed (if you have More than 1) or to Worker nodes as required.

Master Nodes copy/distribution:

```
for instance in master-1 master-2; do
  scp ca.crt ca.key kube-apiserver.key kube-apiserver.crt \
    service-account.key service-account.crt \
    etcd-server.key etcd-server.crt \
    ${instance}:~/
done
```

Step 4:

Generate KubeConfig files with kubectl.
Kubeconfig files help Kubernetes clients locate and authenticate API servers.

a: Generate kubeconfig files for all Control Plane components.

- Controller Manager (can reach API server directly via loopback address - 127.0.0.1, since they are on same system)

```
# Generate a kubeconfig file for the kube-controller-manager service (Note Loopback Address):
{
  kubectl config set-cluster cog-first-cluster \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config set-credentials system:kube-controller-manager \
    --client-certificate=kube-controller-manager.crt \
    --client-key=kube-controller-manager.key \
    --embed-certs=true \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config set-context default \
    --cluster=cog-first-cluster \
    --user=system:kube-controller-manager \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config use-context default --kubeconfig=kube-controller-manager.kubeconfig
}
```

Result: kube-controller-manager.kubeconfig

- Scheduler (can reach API server directly via loopback address - 127.0.0.1, since they are on same system)

```
# Generate a kubeconfig file for the kube-scheduler service (Note the LB IP address)
{
  kubectl config set-cluster cog-first-cluster \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config set-credentials system:kube-scheduler \
    --client-certificate=kube-scheduler.crt \
    --client-key=kube-scheduler.key \
    --embed-certs=true \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config set-context default \
    --cluster=cog-first-cluster \
    --user=system:kube-scheduler \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config use-context default --kubeconfig=kube-scheduler.kubeconfig
}
```

Results: kube-scheduler.kubeconfig

- Kube Proxy (can reach API via LoadBalancer - not directly):

```
# Generate a kubeconfig file for the kube-proxy service. (Note the LB IP address)
{
  kubectl config set-cluster cog-first-cluster \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://192.168.7.147:6443 \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config set-credentials system:kube-proxy \
    --client-certificate=kube-proxy.crt \
    --client-key=kube-proxy.key \
    --embed-certs=true \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config set-context default \
    --cluster=cog-first-cluster \
    --user=system:kube-proxy \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
}
```

Result: kube-proxy.kubeconfig

- Admin User (can reach API via LoadBalancer - not directly. If Admin User system is Master 1, we can use loopback address)

```
# Generate a kubeconfig file for the admin user.
{
  kubectl config set-cluster cog-first-cluster \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=admin.kubeconfig

  kubectl config set-credentials admin \
    --client-certificate=admin.crt \
    --client-key=admin.key \
    --embed-certs=true \
    --kubeconfig=admin.kubeconfig

  kubectl config set-context default \
    --cluster=cog-first-cluster \
    --user=admin \
    --kubeconfig=admin.kubeconfig

  kubectl config use-context default --kubeconfig=admin.kubeconfig
}
```

Results: admin.kubeconfig

b: Distribute/Copy to other Machines.

Worker nodes:

```
for instance in worker-1 worker-2; do
  scp kube-proxy.kubeconfig ${instance}:~/
done
```

Master nodes:

```
for instance in master-1 master-2; do
  scp admin.kubeconfig kube-controller-manager.kubeconfig kube-scheduler.kubeconfig ${instance}:~/
done

for instance in master-2; do
  scp admin.kubeconfig kube-controller-manager.kubeconfig kube-scheduler.kubeconfig ${instance}:~/
done
```

then distribute to required nodes.
- Distribute kubelet** and kube-proxy kubeconfig files to the worker nodes.
- Distribute kube-controller-manager and kube-scheduler kubeconfig files to the master nodes.

Step 5:

Data Encryption config and key:
https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/#encrypting-your-data
Create Encryption Key Config file (in Yaml) to help store data in ETCD in an Encrypted format.
This Key will be needed during configuration of Kube API.

# Generate an encryption key:

```
ENCRYPTION_KEY=$(head -c 32 /dev/urandom | base64)
```

# Create the encryption-config.yaml encryption config file:

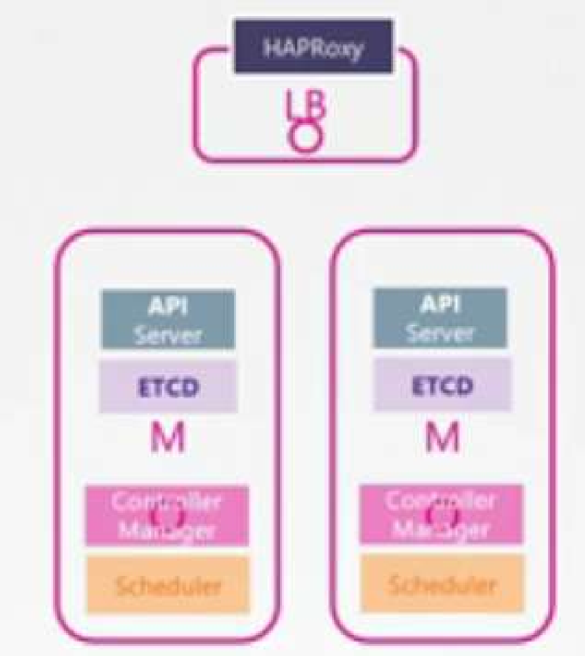
```
cat > encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
 - resources:
     - secrets
   providers:
     - aescbc:
         keys:
           - name: key1
             secret: ${ENCRYPTION_KEY}
     - identity: {}
EOF
```

# Distribute/Copy this file to Master nodes.

```
for instance in master-1 master-2; do
  scp encryption-config.yaml ${instance}:~/
done
```

Step 6:

Install the Master Components:



a: Install/Deploy ETCD Cluster on MASTER node (stacked, not External)
b: Deploy Control Plane Components on MASTER node:
-- Kube API Server
-- Controller Manager
-- Kube Scheduler
c: Deploy/Configure LoadBalancer (HAProxy) on LB

a: Install/Deploy ETCD Cluster on MASTER node (stacked, not External)

# Download binaries

```
wget -q --show-progress --https-only --timestamping \
  "https://storage.googleapis.com/etcd/v3.3.13/etcd-v3.3.13-linux-amd64.tar.gz"
```

Extract and install the etcd server and the etcdctl command line utility:
# Un-tar the binary and move it to /bin

```
{
  tar -xvf etcd-v3.3.13-linux-amd64.tar.gz
  sudo mv etcd-v3.3.13-linux-amd64/etcd* /usr/local/bin/
}

# Configure the etcd Server
# Create 2 folders: /etc/etcd and /var/lib/etcd
# move etcd certificates to /etc/etcd
# /var/lib/etcd will be set as the directory path
{
  sudo mkdir -p /etc/etcd /var/lib/etcd
  sudo cp ca.crt etcd-server.key etcd-server.crt /etc/etcd/
}

# Save IP Address to environment:
INTERNAL_IP=$(ip addr show eth0 | grep "inet " | awk '{print $2}' | cut -d / -f 1)

# Save Host Name to Environment
ETCD_NAME=$(hostname -s)

# Create the etcd.service systemd unit file:
# /var/lib/etcd will be set as the directory path
cat <<EOF | sudo tee /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos

[Service]
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/etcd-server.crt \\
  --key-file=/etc/etcd/etcd-server.key \\
  --peer-cert-file=/etc/etcd/etcd-server.crt \\
  --peer-key-file=/etc/etcd/etcd-server.key \\
  --trusted-ca-file=/etc/etcd/ca.crt \\
  --peer-trusted-ca-file=/etc/etcd/ca.crt \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \\
  --advertise-client-urls https://${INTERNAL_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster master-1=https://192.168.7.145:2380,master-2=https://192.168.7.146:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

# Start the etcd Server
{
  sudo systemctl daemon-reload
  sudo systemctl enable etcd
  sudo systemctl start etcd
}

# Verification: List the etcd cluster members:
```

```
sudo ETCDCTL_API=3 etcdctl member list \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/etcd/ca.crt \
  --cert=/etc/etcd/etcd-server.crt \
  --key=/etc/etcd/etcd-server.key
```

output (Should list the members that was just created):

```
    45bf9ccad8d8900a, started, master-2, https://192.168.7.146:2380, https://192.168.7.146:2379
    54a5796a6803f252, started, master-1, https://192.168.7.145:2380, https://192.168.7.145:2379
```

```
# Check to see if service is running:
sudo service etcd status
```

b: Deploy Control Plane Components on MASTER node:

-- Kube API Server
-- Controller Manager
-- Kube Scheduler

```
# Create the Kubernetes configuration directory:
sudo mkdir -p /etc/kubernetes/config

# Create the folder for Kubernetes library /var/lib/kubernetes/:
sudo mkdir -p /var/lib/kubernetes/

# Download the official Kubernetes release binaries:
wget -q --show-progress --https-only --timestamping \
  "https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kube-apiserver" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kube-controller-manager" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kube-scheduler" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubectl"

# Install the Kubernetes binaries:
{
  chmod +x kube-apiserver kube-controller-manager kube-scheduler kubectl
  sudo mv kube-apiserver kube-controller-manager kube-scheduler kubectl /usr/local/bin/
}
```

##### ***Configure the Kubernetes API Server [API SERVER]***:

```
# Copy certificate keys and encryption key to the Kubernetes library folder created above
{
  sudo cp ca.crt ca.key kube-apiserver.crt kube-apiserver.key \
    service-account.key service-account.crt \
    etcd-server.key etcd-server.crt \
    encryption-config.yaml /var/lib/kubernetes/
}

# Save IP Address to environment:
INTERNAL_IP=$(ip addr show eth0 | grep "inet " | awk '{print $2}' | cut -d / -f 1)

# Create the kube-apiserver.service systemd unit file:
cat <<EOF | sudo tee /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes
```

```
[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
  --advertise-address=${INTERNAL_IP} \\
  --allow-privileged=true \\
  --apiserver-count=3 \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/var/log/audit.log \\
  --authorization-mode=Node,RBAC \\
  --bind-address=0.0.0.0 \\
  --client-ca-file=/var/lib/kubernetes/ca.crt \\
  --enable-admission-plugins=NodeRestriction,ServiceAccount \\
  --enable-swagger-ui=true \\
  --enable-bootstrap-token-auth=true \\
  --etcd-cafile=/var/lib/kubernetes/ca.crt \\
  --etcd-certfile=/var/lib/kubernetes/etcd-server.crt \\
  --etcd-keyfile=/var/lib/kubernetes/etcd-server.key \\
  --etcd-servers=https://192.168.7.145:2379,https://192.168.7.146:2379 \\
  --event-ttl=1h \\
  --encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \\
  --kubelet-certificate-authority=/var/lib/kubernetes/ca.crt \\
  --kubelet-client-certificate=/var/lib/kubernetes/kube-apiserver.crt \\
  --kubelet-client-key=/var/lib/kubernetes/kube-apiserver.key \\
  --kubelet-https=true \\
  --runtime-config=api/all=true \\
  --service-account-key-file=/var/lib/kubernetes/service-account.crt \\
  --service-cluster-ip-range=10.96.0.0/24 \\
  --service-node-port-range=30000-32767 \\
  --tls-cert-file=/var/lib/kubernetes/kube-apiserver.crt \\
  --tls-private-key-file=/var/lib/kubernetes/kube-apiserver.key \\
  --v=2
Restart=on-failure
RestartSec=5

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

### Configure the Kubernetes Controller Manager:

```
# Copy the kube-controller-manager kubeconfig into kubernetes folder created above:
sudo cp kube-controller-manager.kubeconfig /var/lib/kubernetes/

# Create the kube-controller-manager.service systemd unit file:
cat <<EOF | sudo tee /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
  --address=0.0.0.0 \\
  --cluster-cidr=192.168.7.0/24 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.crt \\
  --cluster-signing-key-file=/var/lib/kubernetes/ca.key \\
  --kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \\
  --leader-elect=true \\
  --root-ca-file=/var/lib/kubernetes/ca.crt \\
  --service-account-private-key-file=/var/lib/kubernetes/service-account.key \\
  --service-cluster-ip-range=10.96.0.0/24 \\
  --use-service-account-credentials=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

### Configure the Kubernetes Scheduler:

```
#Copy the kube-scheduler kubeconfig into kubernetes folder created above:
sudo cp kube-scheduler.kubeconfig /var/lib/kubernetes/
```

```
# Create the kube-scheduler.service systemd unit file:
cat <<EOF | sudo tee /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-scheduler \\
  --kubeconfig=/var/lib/kubernetes/kube-scheduler.kubeconfig \\
  --address=127.0.0.1 \\
  --leader-elect=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

```
# Start the Controller Services for API Server, Controller-Manager and Scheduler
{
  sudo systemctl daemon-reload
  sudo systemctl enable kube-apiserver kube-controller-manager kube-scheduler
  sudo systemctl start kube-apiserver kube-controller-manager kube-scheduler
}
```

```
# Verification: List the status of controller-manager, scheduler and etcd (API server is verified if the command works)
# this API command have been deprecated (from v19+)
kubectl get componentstatuses --kubeconfig admin.kubeconfig
```

Output:

```
        NAME                STATUS    MESSAGE             ERROR
        controller-manager   Healthy   ok
        scheduler            Healthy   ok
        etcd-0              Healthy   {"health": "true"}
        etcd-1              Healthy   {"health": "true"}
```

```
# Check to see if kube-apiserver service is running:
sudo service kube-apiserver status
```

```
# Check to see if kube-controller-manager service is running:
sudo service kube-controller-manager status
```

# Check to see if kube-scheduler service is running:

```
sudo service kube-scheduler status
```

c: Deploy/Configure Front End LoadBalancer (HAProxy) on LB:

Login to loadbalancer instance (This part will be done in Load Balancer VM).

#Install HAProxy

```
sudo apt-get update && sudo apt-get install -y haproxy
```

# Create config (cfg) file for haproxy with the settings.

```
cat <<EOF | sudo tee /etc/haproxy/haproxy.cfg
frontend kubernetes
   bind 192.168.7.147:6443
   option tcplog
   mode tcp
   default_backend kubernetes-master-nodes

backend kubernetes-master-nodes
   mode tcp
   balance roundrobin
   option tcp-check
   server master-1 192.168.7.145:6443 check fall 3 rise 2
   server master-2 192.168.7.146:6443 check fall 3 rise 2
EOF
```

# Restart HAProxy Service

```
sudo service haproxy restart
```

# Check to see if HA-Proxy service is running:

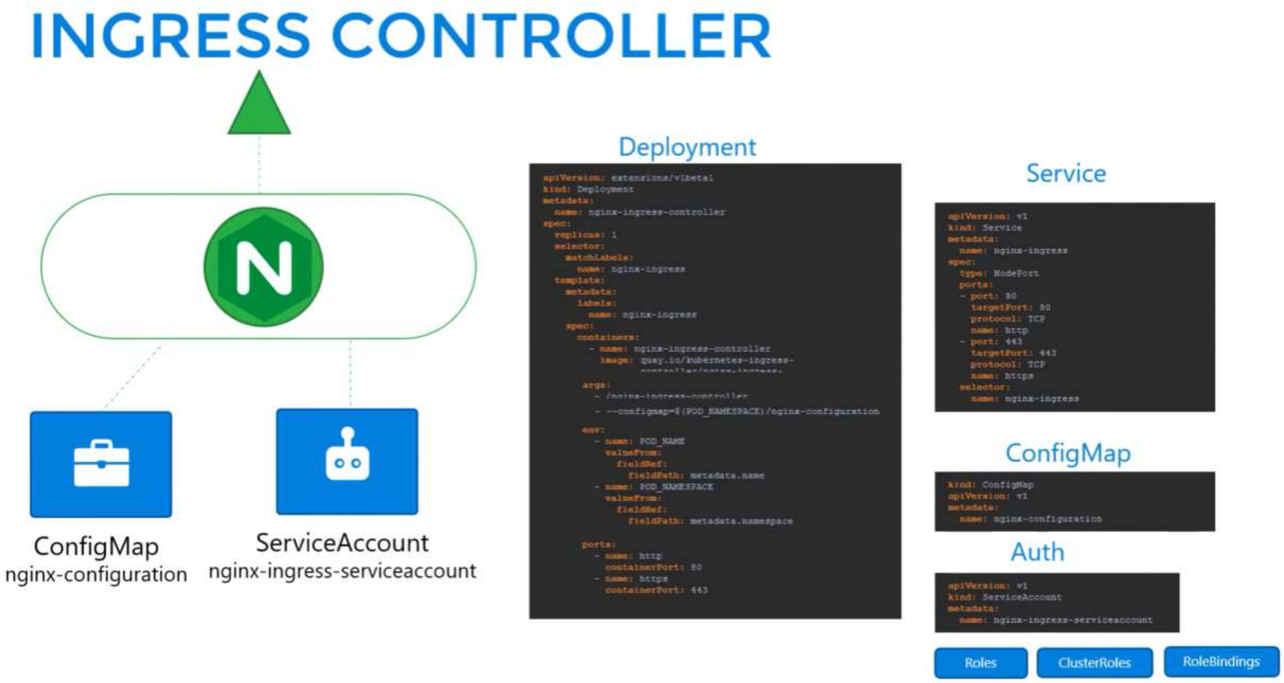
```
sudo service haproxy status
```

#Verification: Make a HTTP request for the Kubernetes version info (from Master-1):

```
curl  https://192.168.7.147:6443/version -k
```

output:

```
        {
          "major": "1",
          "minor": "19",
          "gitVersion": "v1.19.0",
          "gitCommit": "e19964183377d0ec2052d1f1fa930c4d7575bd50",
          "gitTreeState": "clean",
          "buildDate": "2020-08-26T14:23:04Z",
          "goVersion": "go1.15",
          "compiler": "gc",
          "platform": "linux/amd64"
        }
```



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      name: nginx-ingress
  template:
    metadata:
      labels:
        name: nginx-ingress
    spec:
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-
                 controller/nginx-ingress-
        args:
          - /nginx-ingress-controller
          - --configmap=$(POD_NAMESPACE)/nginx-configuration
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        ports:
          - name: http
            containerPort: 80
          - name: https
            containerPort: 443
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    targetPort: 443
    protocol: TCP
    name: https
  selector:
    name: nginx-ingress
```

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
```

===

=============WORKER NODES COMPONENTS (Normal)=========================

<u>Step 7:</u>
Worker Nodes/Kubelet Certificate Authority Generation (perform on Master node and copy to Worker node).
and
KubeConfiguration using Kubectl (perform on Worker node)

Worker nodes components:
- Kubelet (worker node)
- Kube-proxy

In this step, we are going to generate certificated for the Worker node (for the Kubelet) and create config files in the Master VM. Then we move the files to the Worker node.
Note: These steps could have been done above, but it is done here for clarity.

<u>- Worker Node (Kubelet) Certificate:</u>
We have to use a `conf` file to create certificate for Kubelet on Worker node because we are going to have more than one alt_name and openssl command cannot process this.

Run this on the Master Node.

```
cat > openssl-worker-1.cnf <<EOF
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = worker-1
IP.1 = 192.168.7.148
EOF
```

# Generate private key for Worker node (Kubelet)

```
openssl genrsa -out worker-1.key 2048
```

# Generate CSR for Worker node (Kubelet) using private key above

```
openssl req -new -key worker-1.key -subj "/CN=system:node:worker-1/O=system:nodes" -out worker-1.csr -config openssl-worker-1.cnf
```

# Sign certificate for Worker node (Kubelet) using CA servers private key

```
openssl x509 -req -in worker-1.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out worker-1.crt -extensions v3_req -extfile openssl-worker-1.cnf -days 1000
```

Expected files in Folder:

worker-1.key
worker-1.crt

- Worker Node (Kubelet) Kube Config File:
Run this on the Master Node

```
{
  kubectl config set-cluster cog-first-cluster \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://192.168.7.147:6443 \
    --kubeconfig=worker-1.kubeconfig

  kubectl config set-credentials system:node:worker-1 \
    --client-certificate=worker-1.crt \
    --client-key=worker-1.key \
    --embed-certs=true \
    --kubeconfig=worker-1.kubeconfig

  kubectl config set-context default \
    --cluster=cog-first-cluster \
    --user=system:node:worker-1 \
    --kubeconfig=worker-1.kubeconfig

  kubectl config use-context default --kubeconfig=worker-1.kubeconfig
}
```

Results:

worker-1.kubeconfig

- Copy certificates, private keys and kubeconfig files to the worker node:

```
scp ca.crt \
worker-1.crt \
worker-1.key \
worker-1.kubeconfig \
worker-1:~/
```

- Install Binary in Worker node:
https://kubernetes.io/docs/setup/release/#node-binaries

Now we need to install Binaries for Kubectl, kube-proxy and kubelet in the Worker node.

# Do the following in the worker node:

```
wget -q --show-progress --https-only --timestamping \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubectl \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kube-proxy \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubelet
```

# Create the installation directories:

```
sudo mkdir -p \
  /etc/cni/net.d \
  /opt/cni/bin \
  /var/lib/kubelet \
  /var/lib/kube-proxy \
  /var/lib/kubernetes \
  /var/run/kubernetes
```

# Install the worker binaries:

```
{
  chmod +x kubectl kube-proxy kubelet
  sudo mv kubectl kube-proxy kubelet /usr/local/bin/
}
```

- Configure the Kubelet:

### Configure the Kubelet

# Copy kubelet certificates, keys and config to the right folders

```
{
  sudo mv worker-1.key worker-1.crt /var/lib/kubelet/
  sudo mv worker-1.kubeconfig /var/lib/kubelet/kubeconfig
  sudo mv ca.crt /var/lib/kubernetes/
}
```

# Create the kubelet-config.yaml configuration file:

```
cat <<EOF | sudo tee /var/lib/kubelet/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/var/lib/kubernetes/ca.crt"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "10.96.0.10"
resolvConf: "/run/systemd/resolve/resolv.conf"
runtimeRequestTimeout: "15m"
EOF
```

# Note: The resolvConf configuration is used to avoid loops when using CoreDNS for service discovery on systems running systemd-resolved

# Create the kubelet.service systemd unit file:

```
cat <<EOF | sudo tee /etc/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 \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --tls-cert-file=/var/lib/kubelet/${HOSTNAME}.crt \\
  --tls-private-key-file=/var/lib/kubelet/${HOSTNAME}.key \\
  --network-plugin=cni \\
  --register-node=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

- Configure the Kube-Proxy:

```
### Configure the Kube-Proxy

# Move kube-proxy.kubeconfig file that was created earlier to the right folder
sudo mv kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig

# Create the kube-proxy-config.yaml configuration file:
cat <<EOF | sudo tee /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
clientConnection:
  kubeconfig: "/var/lib/kube-proxy/kubeconfig"
mode: "iptables"
clusterCIDR: "192.168.7.0/24"
EOF

# Create the kube-proxy.service systemd unit file:
cat <<EOF | sudo tee /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5

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

- Start the Worker Services

```
### Start the Worker Services
{
  sudo systemctl daemon-reload
  sudo systemctl enable kubelet kube-proxy
  sudo systemctl start kubelet kube-proxy
}
```

Troubleshoot:

If services refuse to run, use the following command: **sudo swapoff -a**
View Journal: sudo journalctl -u etcd.service
Also, after updating Kube, delete the service and create again (by creating systemd file):

```
sudo rm -f  /etc/systemd/system/kubelet.service
sudo rm -f  /etc/systemd/system/kube-proxy.service
```

# Check status:

```
systemctl status kubelet -l or sudo service kubelet status
systemctl status kube-proxy -l or sudo service kube-proxy status
```

- Verification

On master-1:

# List the registered Kubernetes nodes from the master node:

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

output (status = not ready bcos no network):

```
NAME      STATUS     ROLES    AGE   VERSION
worker-1  NotReady   <none>   93s   v1.19.0
```

==

=============WORKER NODES COMPONENTS (WITH TLS Bootstrapping)========================

https://kubernetes.io/docs/reference/access-authn-authz/bootstrap-tokens/#bootstrap-token-secret-format
https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-tls-bootstrapping/#authorize-kubelet-to-create-csr

- Pre-Req:

Make sure the kube-apiserver service has:

```
--enable-bootstrap-token-auth=true
```

Make sure kube-controller-manager has

```
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.crt \\
  --cluster-signing-key-file=/var/lib/kubernetes/ca.key
```

- Copy CA file from Master Node to Worker Node:

# Perform on Master Node

```
scp ca.crt worker-2:~/
```

- Configure and Install the Binaries on the Worker node:

# Perform on Worker Node

# Download and Install Worker Binaries

```
wget -q --show-progress --https-only --timestamping \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubectl \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kube-proxy \
  https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/linux/amd64/kubelet
```

# Create the installation directories:

```
sudo mkdir -p \
  /etc/cni/net.d \
  /opt/cni/bin \
  /var/lib/kubelet \
  /var/lib/kube-proxy \
  /var/lib/kubernetes \
  /var/run/kubernetes
```

# Install the worker binaries:

```
{
  chmod +x kubectl kube-proxy kubelet
  sudo mv kubectl kube-proxy kubelet /usr/local/bin/
}
```

```
# Move the ca certificate from home to designated folder
sudo mv ca.crt /var/lib/kubernetes/
```

<u>- Create the Boostrap Token to be used by Nodes(Kubelets) to invoke Certificate API</u>

```
# For the workers(kubelet) to access the Certificates API, they need to authenticate to the kubernetes api-server first. For this we create a Bootstrap Token to be used by the kubelet
# Perform on Master Node
cat > bootstrap-token-07401b.yaml <<EOF
apiVersion: v1
kind: Secret
metadata:
  # Name MUST be of form "bootstrap-token-<token id>"
  name: bootstrap-token-07401b
  namespace: kube-system

# Type MUST be 'bootstrap.kubernetes.io/token'
type: bootstrap.kubernetes.io/token
stringData:
  # Human readable description. Optional.
  description: "The default bootstrap token generated by 'kubeadm init'."

  # Token ID and secret. Required.
  token-id: 07401b
  token-secret: f395accd246ae52d

  # Expiration. Optional.
  expiration: 2021-03-10T03:22:11Z

  # Allowed usages.
  usage-bootstrap-authentication: "true"
  usage-bootstrap-signing: "true"

  # Extra groups to authenticate the token as. Must start with "system:bootstrappers:"
  auth-extra-groups: system:bootstrappers:worker
EOF
```

```
# Create the token
kubectl create -f bootstrap-token-07401b.yaml
```

<u>- Authorize workers(kubelets) to create CSR</u>

#Next we associate the group we created before to the system:node-bootstrapper ClusterRole. This ClusterRole gives the group enough permissions to bootstrap the kubelet

```
# Perform on Master Node

kubectl create clusterrolebinding create-csrs-for-bootstrapping --clusterrole=system:node-bootstrapper --group=system:bootstrappers
```

--------------- OR ---------------

```
cat > csrs-for-bootstrapping.yaml <<EOF
# enable bootstrapping nodes to create CSR
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: create-csrs-for-bootstrapping
```

```
subjects:
- kind: Group
  name: system:bootstrappers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node-bootstrapper
  apiGroup: rbac.authorization.k8s.io
EOF
```

```
kubectl create -f csrs-for-bootstrapping.yaml
```

- Authorize workers(kubelets) to approve CSR

# Perform on Master Node

```
kubectl create clusterrolebinding auto-approve-csrs-for-group --clusterrole=system:certificates.k8s.io:certificatesigningrequests:nodeclient --group=system:bootstrappers
```

--------------- OR ---------------

```
cat > auto-approve-csrs-for-group.yaml <<EOF
# Approve all CSRs for the group "system:bootstrappers"
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: auto-approve-csrs-for-group
subjects:
- kind: Group
  name: system:bootstrappers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:certificates.k8s.io:certificatesigningrequests:nodeclient
  apiGroup: rbac.authorization.k8s.io
EOF
```

```
kubectl create -f auto-approve-csrs-for-group.yaml
```

- Authorize workers(kubelets) to Auto Renew Certificates on expiration:

# We now create the Cluster Role Binding required for the nodes to automatically renew the certificates on expiry.
Note that we are NOT using the system:bootstrappers group here any more. Since by the renewal period, we believe the node would be bootstrapped and part of the cluster already. All nodes are part of the system:nodes group.

# Perform on Master Node

```
kubectl create clusterrolebinding auto-approve-renewals-for-nodes --clusterrole=system:certificates.k8s.io:certificatesigningrequests:selfnodeclient --group=system:nodes
```

--------------- OR ---------------

```
cat > auto-approve-renewals-for-nodes.yaml <<EOF
# Approve renewal CSRs for the group "system:nodes"
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: auto-approve-renewals-for-nodes
subjects:
- kind: Group
  name: system:nodes
  apiGroup: rbac.authorization.k8s.io
```

```
roleRef:
  kind: ClusterRole
  name: system:certificates.k8s.io:certificatesigningrequests:selfnodeclient
  apiGroup: rbac.authorization.k8s.io
EOF

kubectl create -f auto-approve-renewals-for-nodes.yaml
```

- Configure Kubelet to TLS Bootstrap

It is now time to configure worker to TLS bootstrap using the token we generated

For worker node (above) we started by creating a kubeconfig file with the TLS certificates that we manually generated. Here, we don't have the certificates yet. So we cannot create a kubeconfig file. Instead we create a bootstrap-kubeconfig file with information about the token we created.

# Perform on Worker Node

```
sudo kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set-cluster bootstrap --server='https://192.168.7.147:6443' --certificate-authority=/var/lib/kubernetes/ca.crt
sudo kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set-credentials kubelet-bootstrap --token=07401b.f395accd246ae52d
sudo kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig set-context bootstrap --user=kubelet-bootstrap --cluster=bootstrap
sudo kubectl config --kubeconfig=/var/lib/kubelet/bootstrap-kubeconfig use-context bootstrap
```

--------------- OR ---------------

```
cat <<EOF | sudo tee /var/lib/kubelet/bootstrap-kubeconfig
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /var/lib/kubernetes/ca.crt
    server: https://192.168.7.147:6443
  name: bootstrap
contexts:
- context:
    cluster: bootstrap
    user: kubelet-bootstrap
  name: bootstrap
current-context: bootstrap
kind: Config
preferences: {}
users:
- name: kubelet-bootstrap
  user:
    token: 07401b.f395accd246ae52d
EOF
```

- Configure the Kubelet

# Perform on Worker Node

```
# Create Kubelet Config File
# Create the kubelet-config.yaml configuration file
cat <<EOF | sudo tee /var/lib/kubelet/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/var/lib/kubernetes/ca.crt"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "10.96.0.10"
resolvConf: "/run/systemd/resolve/resolv.conf"
runtimeRequestTimeout: "15m"
EOF
```

#Note: We are not specifying the certificate details - tlsCertFile and tlsPrivateKeyFile - in this file

```
# Configure Kubelet Service
# Create the kubelet.service systemd unit file
cat <<EOF | sudo tee /etc/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 \\
  --bootstrap-kubeconfig="/var/lib/kubelet/bootstrap-kubeconfig" \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --cert-dir=/var/lib/kubelet/pki/ \\
  --rotate-certificates=true \\
  --rotate-server-certificates=true \\
  --network-plugin=cni \\
  --register-node=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

Things to note here:

> *bootstrap-kubeconfig: Location of the bootstrap-kubeconfig file.*
> *cert-dir: The directory where the generated certificates are stored.*
> *rotate-certificates: Rotates client certificates when they expire.*
> *rotate-server-certificates: Requests for server certificates on bootstrap and rotates them when they expire.*

- Configure the Kube-Proxy

# Perform on Worker Node

```
# In one of the previous steps we created the kube-proxy.kubeconfig file.
# Move that file to right folder
        sudo mv kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig
```

```
# Create the kube-proxy-config.yaml configuration file:
cat <<EOF | sudo tee /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
clientConnection:
  kubeconfig: "/var/lib/kube-proxy/kubeconfig"
mode: "iptables"
clusterCIDR: "192.168.7.0/24"
EOF
```

```
# Create the kube-proxy.service systemd unit file:
cat <<EOF | sudo tee /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5

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

- Start the Worker Services

# Perform on Worker Node

```
{
  sudo systemctl daemon-reload
  sudo systemctl enable kubelet kube-proxy
  sudo systemctl start kubelet kube-proxy
}
```

```
# Check status:
systemctl status kubelet -l or sudo service kubelet status
systemctl status kube-proxy -l or sudo service kube-proxy status
```

- Get and Approve Server CSR

# Perform on Master Node

```
# Get CSR name
kubectl get csr

# Approve CSR
kubectl certificate approve <csr-name>
```

- Verification

# Perform on Master Node

```
# List the registered Kubernetes nodes from the master node:
kubectl get nodes --kubeconfig admin.kubeconfig
```

output (status = not ready bcos no network):

```
NAME      STATUS    ROLES   AGE  VERSION
worker-1  NotReady  <none>  93s  v1.19.0
worker-2  NotReady  <none>  93s  v1.19.0
```

==

=============CONFIGURE ADMIN ACCESS (via LB)=========================

Step 9:

- Configuring kubectl for Remote Access
https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/
Each kubeconfig requires a Kubernetes API Server to connect to.
To support high availability the IP address assigned to the external load balancer fronting the Kubernetes API Servers will be used.

# Generate a kubeconfig file suitable for authenticating as the admin user:

```
{
 KUBERNETES_LB_ADDRESS=192.168.7.147

 kubectl config set-cluster cog-first-cluster \
  --certificate-authority=ca.crt \
  --embed-certs=true \
  --server=https://${KUBERNETES_LB_ADDRESS}:6443

 kubectl config set-credentials admin \
  --client-certificate=admin.crt \
  --client-key=admin.key

 kubectl config set-context cog-first-cluster \
  --cluster=cog-first-cluster \
  --user=admin

 kubectl config use-context cog-first-cluster
}
```

==

=============NETWORK CONFIGURATION (POD NETWORK) with WEAVE=========================

Step 9:
Provision Pod Network using weave
https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/#cni
https://www.weave.works/docs/net/latest/kubernetes/kube-addon/

- Install CNI plugins
On **Worker Nodes** run the following:
# Download the CNI Plugins required for weave

```
wget https://github.com/containernetworking/plugins/releases/download/v0.7.5/cni-plugins-amd64-v0.7.5.tgz
```

# Extract it to /opt/cni/bin directory

```
sudo tar -xzvf cni-plugins-amd64-v0.7.5.tgz --directory /opt/cni/bin/
```

- Deploy Weave Network
Run on **Master Node**.

Deploy weave network.

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

# Weave uses POD CIDR of 10.32.0.0/12 by default.

- Verify nodes are ready

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

==

=============KUBE API (Master Node) to KUBELET (Worker Node) Connectivity========================

Step 10:
KUBE API (Master Node) to KUBELET (Worker Node) Connectivity.
https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole
https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/rbac/#rolebinding-and-clusterrolebinding

If we try to view logs of a pod, for example: *kubectl logs weave-net-g9pnc weave -n kube-system*
We see that we get a forbidden error. This is because the Kube-APIServer doesn't have access to the Kubelet (on the Worker Nodes).

To create a connection, we create a ClusterRole for Kubelet and then bind it to the Kube-apiserver.

#Create the system:kube-apiserver-to-kubelet ClusterRole with permissions to access the Kubelet API and perform most common tasks associated with managing pods:

```
cat <<EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:kube-apiserver-to-kubelet
rules:
  - apiGroups:
      - ""
    resources:
      - nodes/proxy
      - nodes/stats
      - nodes/log
      - nodes/spec
      - nodes/metrics
    verbs:
      - "*"
EOF
```

# Bind the system:kube-apiserver-to-kubelet ClusterRole to the kubernetes user:

```
cat <<EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:kube-apiserver
  namespace: ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kube-apiserver-to-kubelet
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
  name: kube-apiserver
EOF
```

Now, we should see out logs come in for pods.

===

Step 11: DNS Setup

https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/
https://kubernetes.io/docs/tasks/administer-cluster/coredns/#installing-coredns
https://github.com/coredns/deployment/blob/master/kubernetes/coredns.yaml.sed

- Install CoreDNS Add-on

To install DNS daemon pods, run the following command:

```
kubectl apply -f coredns.yaml
```

#List the pods created by the kube-dns deployment:

```
kubectl get pods -l k8s-app=kube-dns -n kube-system
```

- Verification

# Create a busybox deployment:

```
kubectl run --generator=run-pod/v1  busybox --image=busybox:1.28 --command -- sleep 3600
```

# List the pod created by the busybox deployment:

```
kubectl get pods -l run=busybox
```

output

```
    NAME                    READY   STATUS    RESTARTS   AGE
    busybox-bd8fb7cbd-vflm9   1/1     Running   0         10s
```

# Execute a DNS lookup for the kubernetes service inside the busybox pod:

```
kubectl exec -ti busybox -- nslookup kubernetes
```

output

```
    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
```

==

===============TESTS=================================

Step 12:

Smoke Tests.

- Test Data Encryption:

# Create a generic secret:

```
kubectl create secret generic kubernetes-the-hard-way \
  --from-literal="mykey=mydata"
```

# Print a hexdump of the kubernetes-the-hard-way secret stored in etcd:

```
sudo ETCDCTL_API=3 etcdctl get \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/etcd/ca.crt \
  --cert=/etc/etcd/etcd-server.crt \
  --key=/etc/etcd/etcd-server.key\
  /registry/secrets/default/kubernetes-the-hard-way | hexdump -C
```

output:

```
    00000000  2f 72 65 67 69 73 74 72  79 2f 73 65 63 72 65 74  |/registry/secret|
    00000010  73 2f 64 65 66 61 75 6c  74 2f 6b 75 62 65 72 6e  |s/default/kubern|
    00000020  65 74 65 73 2d 74 68 65  2d 68 61 72 64 2d 77 61  |etes-the-hard-wa|
    00000030  79 0a 6b 38 73 3a 65 6e  63 3a 61 65 73 63 62 63  |y.k8s:enc:aescbc|
    00000040  3a 76 31 3a 6b 65 79 31  3a 78 cd 3c 33 3a 60 d7  |:v1:key1:x.<3:`.|
    00000050  4c 1e 4c f1 97 ce 75 6f  3d a7 f1 4b 59 e8 f9 2a  |L.L...uo=..KY..*|
    00000060  17 77 20 14 ab 73 85 63  12 12 a4 8d 3c 6e 04 4c  |.w ..s.c....<n.L|
    00000070  e0 84 6f 10 7b 3a 13 10  d0 cd df 81 d0 08 be fa  |..o.{:..........|
    00000080  ea 74 ca 53 b3 b2 90 95  e1 ba bc 3f 88 76 db 8e  |.t.S.......?.v..|
    00000090  e1 1e 17 ea 0d b0 3b e3  e3 df eb 2e 57 76 1d d0  |......;.....Wv..|
    000000a0  25 ca ee 5b f2 27 c7 f2  8e 58 93 e9 28 45 8f 3a  |%..[.'...X..(E.:|
    000000b0  e7 97 bf 74 86 72 fd e7  f1 bb fc f7 2d 10 4d c3  |...t.r......-.M.|
    000000c0  70 1d 08 75 c3 7c 14 55  18 9d 68 73 ec e3 41 3a  |p..u.|.U..hs..A:|
    000000d0  dc 41 8a 4b 9e 33 d9 3d  c0 04 60 10 cf ad a4 88  |.A.K.3.=..`.....|
    000000e0  7b e7 93 3f 7a e8 1b 22  bf 0a                    |{..?z.."..|
    000000ea
```

Note: The etcd key should be prefixed with k8s:enc:aescbc:v1:key1, which indicates the aescbc provider was used to encrypt the data with the key1 encryption key.

#Cleanup:

```
kubectl delete secret kubernetes-the-hard-way
```

- Test Deployment

# Deployments

# Create a deployment for the nginx web server:

```
kubectl create deployment nginx --image=nginx
```

#List the pod created by the nginx deployment:

```
kubectl get pods -l app=nginx
```

output

```
    NAME                    READY   STATUS    RESTARTS   AGE
    nginx-dbddb74b8-6lxg2   1/1     Running   0          10s
```

# Services

# Create a service to expose deployment nginx on node ports.

```
kubectl expose deploy nginx --type=NodePort --port 80
PORT_NUMBER=$(kubectl get svc -l app=nginx -o jsonpath="{.items[0].spec.ports[0].nodePort}")
```

#Test to view NGINX page

```
curl http://worker-1:$PORT_NUMBER
curl http://worker-2:$PORT_NUMBER
```

output:

```
    <!DOCTYPE html>
    <html>
    <head>
```

```
        <title>Welcome to nginx!</title>
         # Output Truncated for brevity
        <body>
```

```
# Logs
# Retrieve the full name of the nginx pod:
POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")

#Print the nginx pod logs:
kubectl logs $POD_NAME
output:
        10.32.0.1 - - [20/Mar/2019:10:08:30 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.58.0" "-"
        10.40.0.0 - - [20/Mar/2019:10:08:55 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.58.0" "-"
```

```
# Exec
# Print the nginx version by executing the nginx -v command in the nginx container:
kubectl exec -ti $POD_NAME -- nginx -v

output:
        nginx version: nginx/1.15.9
```

```
# Clean-up:
kubectl delete deployment nginx
```

Step 13:

- Run End-to-End Tests

```
# Install Go:
wget https://dl.google.com/go/go1.12.1.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.12.1.linux-amd64.tar.gz
export GOPATH="/home/vagrant/go"
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin


# Install kubetest:
go get -u k8s.io/test-infra/kubetest
kubetest --extract=v1.19.0
cd test-infra
```

This may take a few minutes depending on your network speed

```
#Use the version specific to your cluster
sudo apt  install jq

K8S_VERSION=$(kubectl version -o json | jq -r '.serverVersion.gitVersion')

export KUBERNETES_CONFORMANCE_TEST=y
export KUBECONFIG="$HOME/.kube/config"
export KUBE_MASTER_IP="192.168.7.145:6443"
export KUBE_MASTER=master-1

kubetest --provider=skeleton --test --test_args="--ginkgo.focus=\[Conformance\]" --extract ${K8S_VERSION} | tee testout.txt
```

Note: This could take about 1.5 to 2 hours. The number of tests run and passed will be displayed at the end.