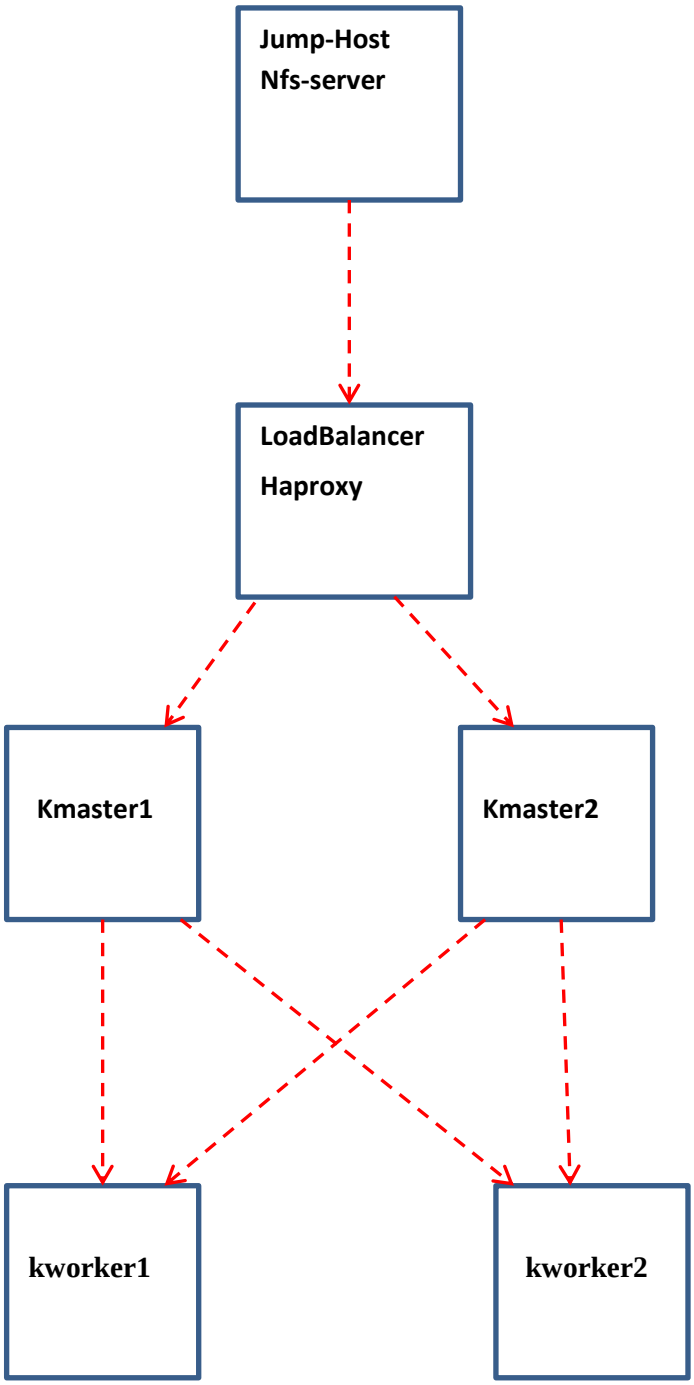
Set up multi master Kubernetes cluster using Kubeadm



Role	FQDN	IP	OS	RAM	CPU
Nfs-server		172.31.93.126	Ubuntu 22.04 LTS	1 Gig	1 vCPU
Load-Balancer		172.31.10.129	Ubuntu 20.04 LTS	2 Gig	2 vCPU
Kmaster1		172.31.7.243	Ubuntu 20.04 LTS	2 Gig	2 vCPU
Kmaster2		172.31.15.34	Ubuntu 20.04 LTS	2 Gig	2 vCPU
Kworker1		172.31.10.60	Ubuntu 20.04 LTS	2 Gig	2 vCPU
Kworker2		172.31.9.75	Ubuntu 20.04 LTS	2 Gig	2 vCPU

1) **Install haproxy in LoadBalancer node**
   sudo su – root
   apt update && apt install -y haproxy

2) **Configure haproxy –**

Append the below lines to **/etc/haproxy/haproxy.cfg**

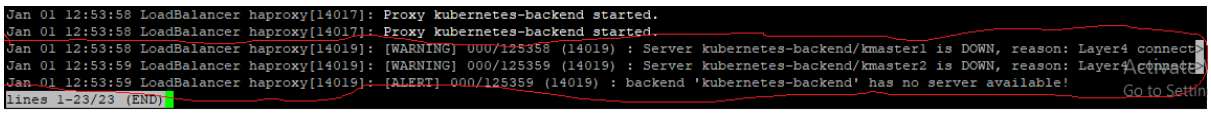
```
frontend kubernetes-frontend
  bind 172.31.6.108:6443
  mode tcp
  option tcplog
  default_backend kubernetes-backend

backend kubernetes-backend
  mode tcp
  option tcp-check
  balance roundrobin
  server kmaster1 172.31.7.95:6443 check fall 3 rise 2
  server kmaster2 172.31.11.174:6443 check fall 3 rise 2
```

haproxy.cfg.txt

3) **Restart haproxy and check the status to ensure its running and active.**

   Systemctl restart haporxy
   Systemctl status haporxy



We can ignore those warnings for now since the backend Master nodes not yet configured.

On all kubernetes nodes (kmaster1, kmaster2, kworker1,kworker2)

4) Disable the firewall and off the swap memory

```
ufw disable
swapoff -a; sed -i '/swap/d' /etc/fstab
```

5) Update sysctl setting for kubernetes networking

```
cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl –system
```

6) Install docker engine

```
{
  apt install -y apt-transport-https ca-certificates curl gnupg-agent software-
properties-common
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
  add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
  apt update && apt install -y docker-ce=5:19.03.10~3-0~ubuntu-focal containerd.io
}
```

Kubernetes Setup in all K8s Nodes

7) Add signing key and kubernetes repository

```
{
  curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
  echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" >
/etc/apt/sources.list.d/kubernetes.list
}
```

8) Install Kubernetes components in all the master and worker nodes (here I am installing V 1.25.5.

apt-get update && apt-get install kubeadm=1.25.5-00 kubelet=1.25.5-00 kubectl=1.25.5-00

On any one of the Kubernetes master node (e.g. kmaster1)

9) Initialize Kubernetes Cluster-

kubeadm init --control-plane-endpoint="172.31.10.129:6443" --upload-certs --apiserver-advertise-address=172.31.7.243 --pod-network-cidr=192.168.0.0/16

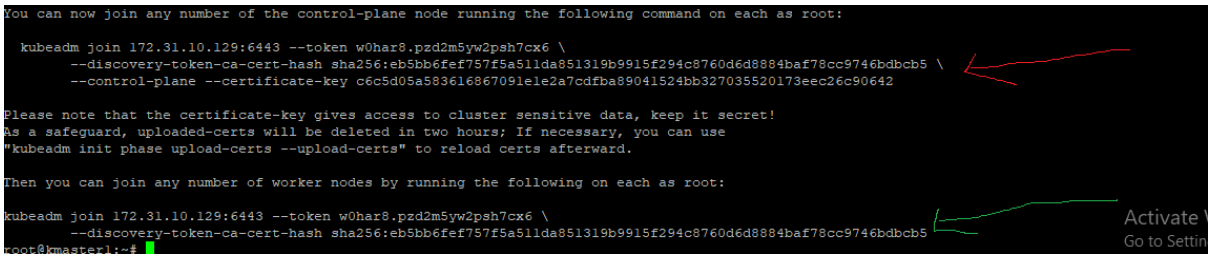
--control-plane-endpoint → IP for the LoadBalancer Node
--apiserver-advertise-address→ IP for the Kmaster1

Before that we need have a little workaround so that container runtime doesn't make any issue during Kubeadm inint.

rm /etc/containerd/config.toml
systemctl restart containerd

Now run Kubeadm init (the above command)..

Upon successful initialization on kmaster1 you will get the below output



Red one→ would be used to join the 2nd master node (kmaster2) to the cluster
Green one→ would be used to join the worker nodes to the cluster

Join other nodes to the cluster (kmaster2 & kworker1 & kworker2)

10) Join the 2nd master node (kmaster2) to the cluster.

Before joining perform the below step in kmaster2 to avoid any container runtime issue.

rm /etc/containerd/config.toml
systemctl restart containerd

Now run

kubeadm join 172.31.10.129:6443 --token w0har8.pzd2m5yw2psh7cx6 \
--discovery-token-ca-cert-hash sha256:eb5bb6fef757f5a511da851319b9915f294c8760d6d8884baf78cc9746bdbcb5 \
--control-plane --certificate-key c6c5d05a583616867091e1e2a7cdfba89041524bb327035520173eec26c90642 --apiserver-advertise-address 172.31.15.34

--apiserver-advertise-address → is the IP for 2nd master node (kmaster2) to join.

11) Join the kworker1 and kworker2 to the cluster.

Again you need to perform the below step to overcome from container runtime issues for containerd in both the kworker1 and kworker2.

rm /etc/containerd/config.toml

systemctl restart containerd

kubeadm join 172.31.10.129:6443 --token w0har8.pzd2m5yw2psh7cx6 \

--discovery-token-ca-cert-hash

sha256:eb5bb6fef757f5a511da851319b9915f294c8760d6d8884baf78cc9746bdbcb5

You can see here the nodes going to join here through the loadbalancer IP, not directly through the master nodes, since we are load balancing master nodes for HA cluster.

12) Now we are all set. Let's check the status of all nodes.

```
root@kmaster1:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes
NAME       STATUS     ROLES           AGE     VERSION
kmaster1   NotReady   control-plane   41m     v1.25.5
kmaster2   NotReady   control-plane   14m     v1.25.5
kworker1   NotReady   <none>          8m9s    v1.25.5
kworker2   NotReady   <none>          6m58s   v1.25.5
```

why it's not ready yet since we do not have any CNI installed yet so that internal networking can take place.

13) Let's install calico as a CNI.

kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f https://docs.projectcalico.org/v3.15/manifests/calico.yaml

Note:- Still I didn't setup any .kube directory, that's what I am using admin.conf file directly , which is actually the config file in .kube directory.

Now all master and worker nodes are ready.

```
root@kmaster1:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes
NAME       STATUS   ROLES           AGE   VERSION
kmaster1   Ready    control-plane   52m   v1.25.5
kmaster2   Ready    control-plane   25m   v1.25.5
kworker1   Ready    <none>          18m   v1.25.5
kworker2   Ready    <none>          17m   v1.25.5
root@kmaster1:~#
```

14) Install kubectl component in your jump host, nfs-server in my case, just the same way adding the apt-key.gpg and kubernetes repository in your jump host and only install kubectl (not kubelet and Kubeadm). Copy the admin.conf file from any of the master node to jump host and place it in /home/.kube directory named as config file.

```
root@nfs-server:~# mkdir .kube
```

```
root@nfs-server:~# cd .kube
root@nfs-server:~/.kube# scp root@172.31.7.243:/etc/kubernetes/admin.conf config
admin.conf
100% 5637     5.4MB/s   00:00
root@nfs-server:~/.kube# ls -ltr
total 8
-rw------- 1 root root 5637 Jan  1 14:51 config
root@nfs-server:~/.kube# cd
root@nfs-server:~# kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
kmaster1   Ready    control-plane   73m   v1.25.5
kmaster2   Ready    control-plane   46m   v1.25.5
kworker1   Ready    <none>          39m   v1.25.5
kworker2   Ready    <none>          38m   v1.25.5
root@nfs-server:~#
```

if you see the config file in .kube directory (/root/.kube) in nfs-server. You can see the kube api-server pointing to the IP address of the LoadBalancer.

Let’s do some checks

```
root@nfs-server:~# kubectl cluster-info
Kubernetes control plane is running at https://172.31.10.129:6443
CoreDNS is running at https://172.31.10.129:6443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
dump'.
root@nfs-server:~#
```

The kubernetes master is running in **172.31.10.129, which is the LoadBalancer IP address**

Let’s check the node status.

```
root@nfs-server:~# kubectl get nodes -o wide
NAME       STATUS   ROLES           AGE    VERSION   INTERNAL-IP
EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
kmaster1   Ready    control-plane   169m   v1.25.5   172.31.7.243   <none>
Ubuntu 20.04.5 LTS   5.15.0-1026-aws   containerd://1.6.14
kmaster2   Ready    control-plane   142m   v1.25.5   172.31.15.34   <none>
Ubuntu 20.04.5 LTS   5.15.0-1026-aws   containerd://1.6.14
```

```
kworker1   Ready    <none>          135m   v1.25.5   172.31.10.60   <none>
Ubuntu 20.04.5 LTS   5.15.0-1026-aws   containerd://1.6.14
kworker2   Ready    <none>          134m   v1.25.5   172.31.9.75    <none>
Ubuntu 20.04.5 LTS   5.15.0-1026-aws   containerd://1.6.14
root@nfs-server:~#
```

so all are in ready state..

you can add as many master nodes by simply following the previous steps.

Let’s check everything is running fine in kube-system namespace

```
root@nfs-server:~# kubectl get all -n kube-system
NAME                                          READY   STATUS    RESTARTS
AGE
pod/calico-kube-controllers-5bb9b8b454-shgnw   1/1     Running   0
132m
pod/calico-node-52l8v                         1/1     Running   0
132m
pod/calico-node-7z4bc                         1/1     Running   0
132m
pod/calico-node-8f6j9                         1/1     Running   0
132m
pod/calico-node-hn88w                         1/1     Running   0
132m
pod/coredns-565d847f94-9qs4r                  1/1     Running   0
176m
pod/coredns-565d847f94-r9fsz                  1/1     Running   0
176m
pod/etcd-kmaster1                             1/1     Running   0
176m
pod/etcd-kmaster2                             1/1     Running   0
149m
pod/kube-apiserver-kmaster1                   1/1     Running   0
176m
pod/kube-apiserver-kmaster2                   1/1     Running   0
149m
pod/kube-controller-manager-kmaster1          1/1     Running   1 (149m
ago)   176m
pod/kube-controller-manager-kmaster2          1/1     Running   0
149m
pod/kube-proxy-lfhhb                          1/1     Running   0
142m
pod/kube-proxy-qglkb                          1/1     Running   0
176m
pod/kube-proxy-qxldb                          1/1     Running   0
149m
pod/kube-proxy-vs9gm                          1/1     Running   0
143m
```

```
pod/kube-scheduler-kmaster1                    1/1     Running   0
176m
pod/kube-scheduler-kmaster2                    1/1     Running   0
149m

NAME               TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
service/kube-dns   ClusterIP   10.96.0.10   <none>
53/UDP,53/TCP,9153/TCP   176m

NAME                         DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/calico-node   4         4         4       4            4
kubernetes.io/os=linux   132m
daemonset.apps/kube-proxy    4         4         4       4            4
kubernetes.io/os=linux   176m

NAME                                       READY   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/calico-kube-controllers   1/1     1            1
132m
deployment.apps/coredns                   2/2     2            2
176m

NAME                                                 DESIRED   CURRENT
READY   AGE
replicaset.apps/calico-kube-controllers-5bb9b8b454   1         1         1
132m
replicaset.apps/coredns-565d847f94                   2         2         2
176m
root@nfs-server:~#
```

so we have etcd, api-server controller, core-dns, calico, kube-proxy all running fine.

Let’s create a deployment

```
root@nfs-server:~# kubectl create deploy my-nginx --image=nginx --
replicas=3
deployment.apps/my-nginx created
root@nfs-server:~# kubectl get deploy
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx   3/3     3            3           10s
root@nfs-server:~# kubectl get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/my-nginx-f44495498-dlcms   1/1     Running   0          24s
pod/my-nginx-f44495498-lrcmk   1/1     Running   0          24s
pod/my-nginx-f44495498-prjbz   1/1     Running   0          24s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3h3m
```

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-nginx      3/3     3            3           24s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/my-nginx-f44495498      3         3         3       24s
root@nfs-server:~#
```

Let’s expose the deployment

```
root@nfs-server:~# kubectl expose deploy my-nginx --type=NodePort --
port=80
service/my-nginx exposed
root@nfs-server:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        3h5m
my-nginx     NodePort    10.96.62.235    <none>        80:30062/TCP   9s
```

let’s access it from internet..

http://44.203.52.240:30062/

So in-case any master is down the load balancer is load balance across without any failure to connect to underlying worker nodes