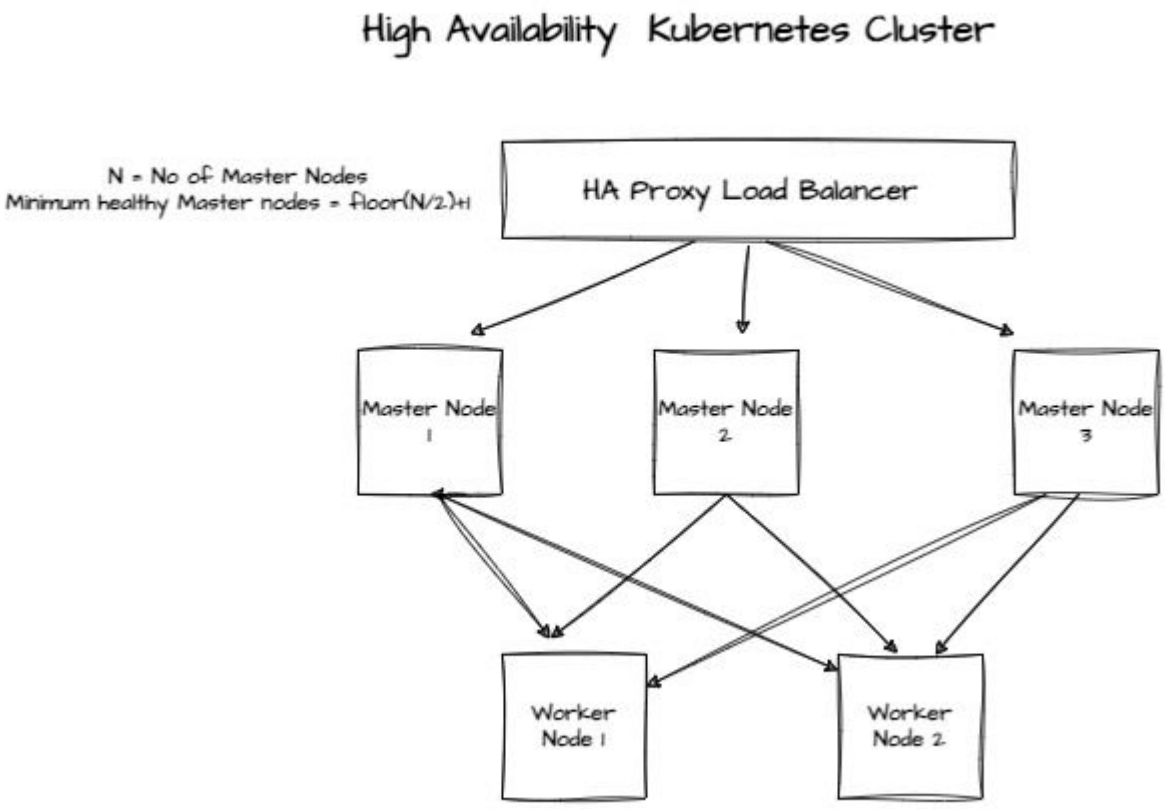



## Highly Available K8s Cluster

To set up a highly available Kubernetes cluster with two master nodes and three worker nodes without using a cloud load balancer, you can use a virtual machine to act as a load balancer for the API server. Here are the detailed steps for setting up such a cluster:

### Prerequisites

- 3 master nodes
- 3 worker nodes
- 1 load balancer node
- All nodes should be running a Linux distribution like Ubuntu

### Step 1: Prepare the Load Balancer Node

1. **Install HAProxy:**
   ```
   sudo apt-get update
   sudo apt-get install -y haproxy
   ```
2. **Configure HAProxy:** Edit the HAProxy configuration file (/etc/haproxy/haproxy.cfg):
   ```
   sudo nano /etc/haproxy/haproxy.cfg
   ```

**Add the following configuration:**

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

backend kubernetes-backend
    mode tcp
    balance roundrobin
    option tcp-check
    server master1 <MASTER1_IP>:6443 check
    server master2 <MASTER2_IP>:6443 check
```

3. **Restart HAProxy:**

```
sudo systemctl restart haproxy
```

## Step 2: Prepare All Nodes (Masters and Workers)

1. **Install Docker, kubeadm, kubelet, and kubectl:**

```
sudo apt-get update
sudo apt install docker.io -y
sudo chmod 666 /var/run/docker.sock
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg
sudo mkdir -p -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo apt update
sudo apt install -y kubeadm=1.30.0-1.1 kubelet=1.30.0-1.1 kubectl=1.30.0-1.1
```

## Step 3: Initialize the First Master Node

1. **Initialize the first master node:**

```
sudo kubeadm init --control-plane-endpoint "LOAD_BALANCER_IP:6443" --upload-certs --pod-
network-cidr=10.244.0.0/16
```

2. **Set up kubeconfig for the first master node:**

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

3. **Install Calico network plugin:**

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

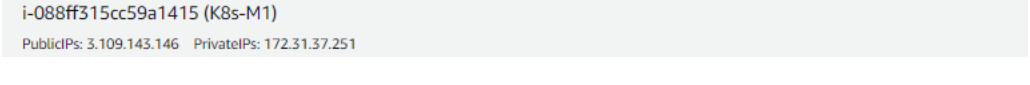
4. **Install Ingress-NGINX Controller:**

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-
v0.49.0/deploy/static/provider/baremetal/deploy.yaml
```

## Step 4: Join the Second & third Master Node

1. 

```
You can now join any number of the control-plane node running the following command on each as root:

  kubeadm join 13.234.120.104:6443 --token bfugm1.64jh62uu73814a3s \
        --discovery-token-ca-cert-hash sha256:07a6d4a097c1427d541f9f057d7584ec08f49c23859e133883eece6f334ff2b0 \
        --control-plane --certificate-key 8a3820db3d4c5342d0a654c9a29ceadc73c79aa8959f5098bc0ea78822878448

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
```



2. **Set up kubeconfig for the second and third master node:**

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

## Step 5: Join the Worker Nodes

1. 
```
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 13.234.120.104:6443 --token bfugm1.64jh62uu73814a3s \
        --discovery-token-ca-cert-hash sha256:07a6d4a097c1427d541f9f057d7584ec08f49c23859e133883eece6f334ff2b0
ubuntu@ip-172-31-37-251:~$
```

i-088ff315cc59a1415 (K8s-M1)

PublicIPs: 3.109.143.146 PrivateIPs: 172.31.37.251

## Step 6: Verify the Cluster

```
ubuntu@ip-172-31-37-251:~$ kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
ip-172-31-37-251    Ready    control-plane   12m     v1.30.0
ip-172-31-40-145    Ready    <none>          72s     v1.30.0
ip-172-31-40-188    Ready    <none>          60s     v1.30.0
ip-172-31-40-73     Ready    control-plane   6m12s   v1.30.0
ip-172-31-42-100    Ready    control-plane   6m29s   v1.30.0
ubuntu@ip-172-31-37-251:~$
ubuntu@ip-172-31-37-251:~$
```

i-088ff315cc59a1415 (K8s-M1)

PublicIPs: 3.109.143.146 PrivateIPs: 172.31.37.251

By following these steps, you will have a highly available Kubernetes cluster with three master nodes and two worker nodes, and a load balancer distributing traffic between the master nodes.
This setup ensures that if one master node fails, the other will continue to serve the API requests.

# Verification

## Step 1: Install etcdctl

1. **Install etcdctl using apt(**On Master-1**)**

```
sudo apt-get update
sudo apt-get install -y etcd-client
```

2. **Check the health of the etcd cluster:**

```
sudo ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --
cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/peer.crt --
key=/etc/kubernetes/pki/etcd/peer.key endpoint health
```

## Step 2: Verify HAProxy Configuration and Functionality

1. **Configure HAProxy Stats:**

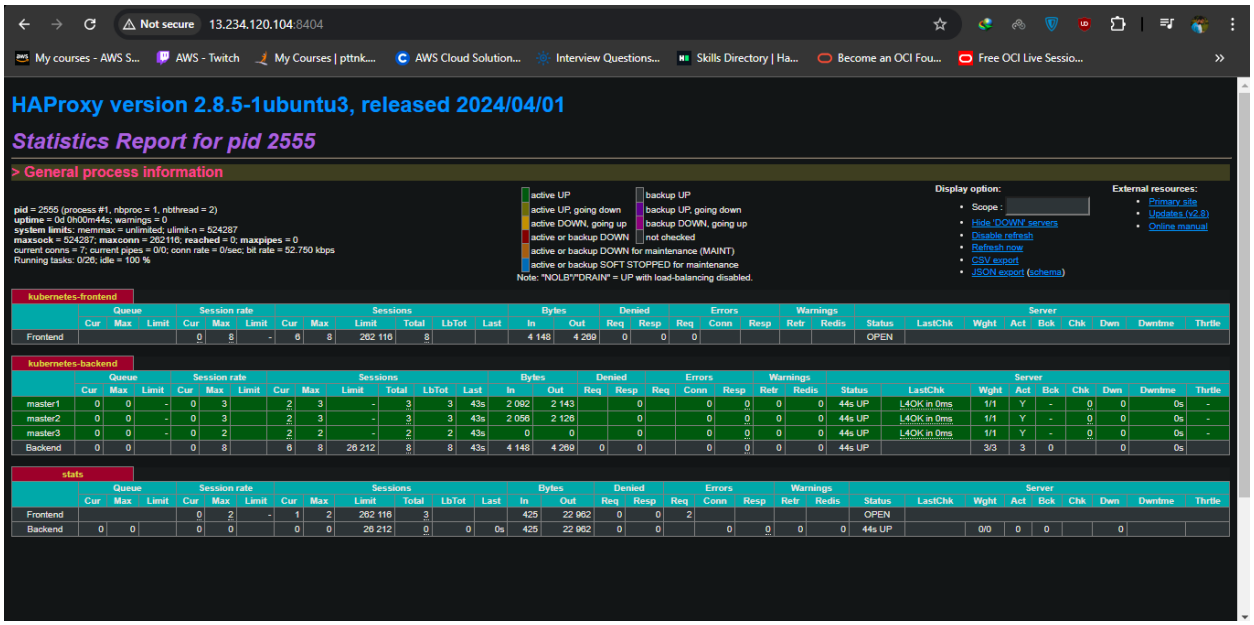
Add the stats configuration to (/etc/haproxy/haproxy.cfg)

```
listen stats
   bind *:8404
   mode http
   stats enable
   stats uri /
   stats refresh 10s
   stats admin if LOCALHOST
```

2. **Restart HAProxy:**

```
sudo systemctl restart haproxy
```

3. **Check HAProxy Stats:**
   Access the stats page at http://<LOAD_BALANCER_IP>:8404.




Now let's move to Master Node 3 and save any deployment yaml file (Board Game).
You can find this file on github https://github.com/Bijan1235/Boardgame.git
Save this file as ds.yml
And the yaml file is(deployment-service.yaml);

```
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: boardgame-deployment
spec:
  selector:
    matchLabels:
      app: boardgame
  replicas: 2 # Number of replicas that will be created for this deployment
  template:
    metadata:
      labels:
        app: boardgame
    spec:
      containers:
        - name: boardgame
          image: bijan9438/boardgame:latest # Image that will be used to containers in the cluster
          imagePullPolicy: Always
          ports:
            - containerPort: 8080 # The port that the container is running on in the cluster


---

apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
```

```
metadata: # Metadata of the resource kind we are creating
  name: boardgame-ssvc
spec:
  selector:
    app: boardgame
  ports:
    - protocol: "TCP"
      port: 8080
      targetPort: 8080
  type: LoadBalancer # type of the service.
```

Now apply it; kubectl apply -f ds.yml

```
ubuntu@ip-172-31-40-73:~$ vi ds.yml
ubuntu@ip-172-31-40-73:~$ kubectl apply -f ds.yml
deployment.apps/boardgame-deployment created
service/boardgame-ssvc created
ubuntu@ip-172-31-40-73:~$
```

i-094cd6506b93b25f9 (K8s-M3)

PublicIPs: 3.111.171.209 PrivateIPs: 172.31.40.73

To check on which worker node and port our application is running;

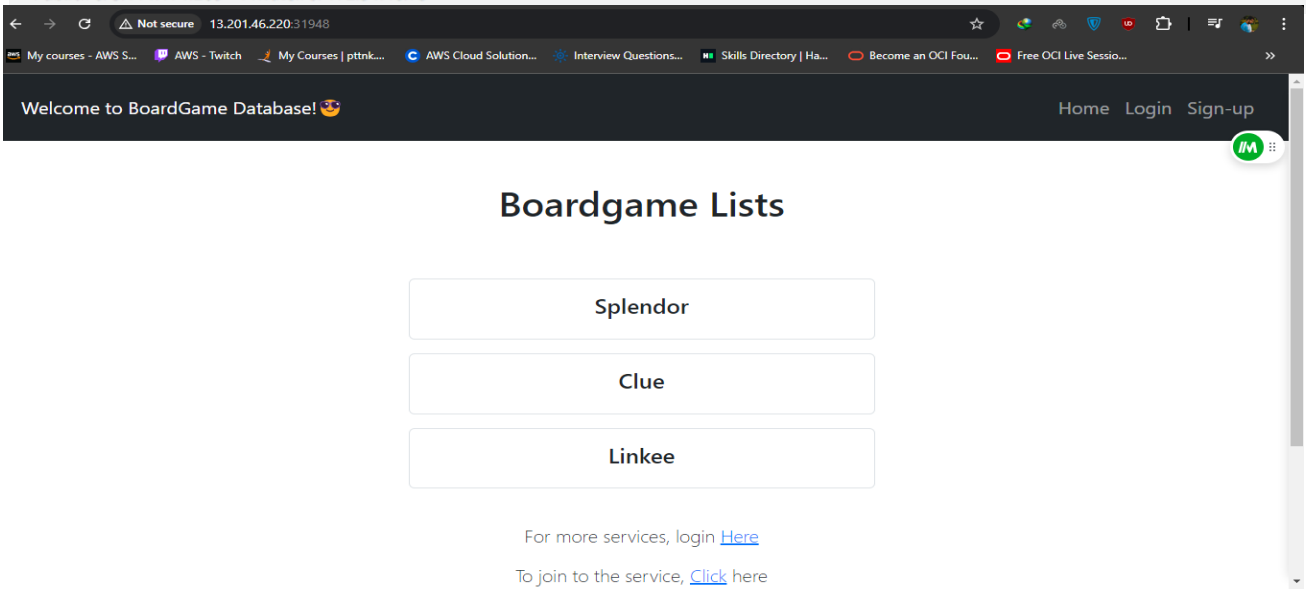
```
ubuntu@ip-172-31-40-73:~$ kubectl get all
NAME                                      READY   STATUS    RESTARTS   AGE
pod/boardgame-deployment-759944f5c7-6mrqq   1/1     Running   0          53s
pod/boardgame-deployment-759944f5c7-tp5vm   1/1     Running   0          53s

NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/boardgame-ssvc   LoadBalancer   10.97.81.169   <pending>     8080:31948/TCP   53s
service/kubernetes       ClusterIP      10.96.0.1      <none>        443/TCP          24m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/boardgame-deployment   2/2     2            2           53s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/boardgame-deployment-759944f5c7   2         2         2       53s
ubuntu@ip-172-31-40-73:~$ kubectl describe pod boardgame-deployment-759944f5c7-6mrqq
Name:             boardgame-deployment-759944f5c7-6mrqq
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-172-31-40-145/172.31.40.145
Start Time:       Sun, 18 Aug 2024 08:37:03 +0000
Labels:           app=boardgame
                  pod-template-hash=759944f5c7
Annotations:      cni.projectcalico.org/containerID: bc7f5633808b43db4a3f1df14af6c471d62692f57f5593689ade81e9358834a8
                  cni.projectcalico.org/podIP: 10.244.133.6/32
                  cni.projectcalico.org/podIPs: 10.244.133.6/32
Status:           Running
IP:               10.244.133.6
IPs:
  IP:           10.244.133.6
Controlled By:  ReplicaSet/boardgame-deployment-759944f5c7
Containers:
```

Port

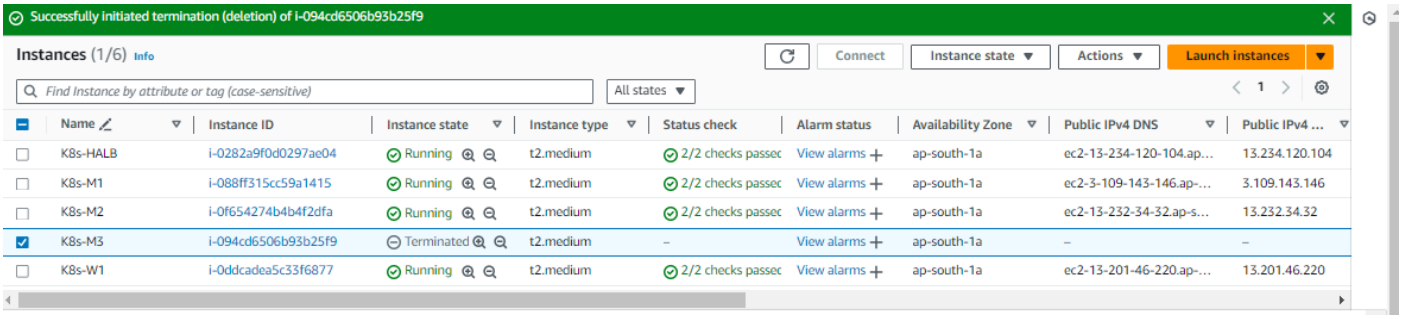
Worker Node-1

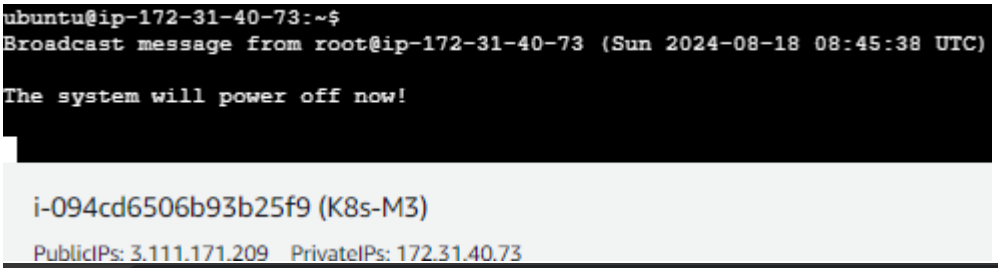
i-094cd6506b93b25f9 (K8s-M3)

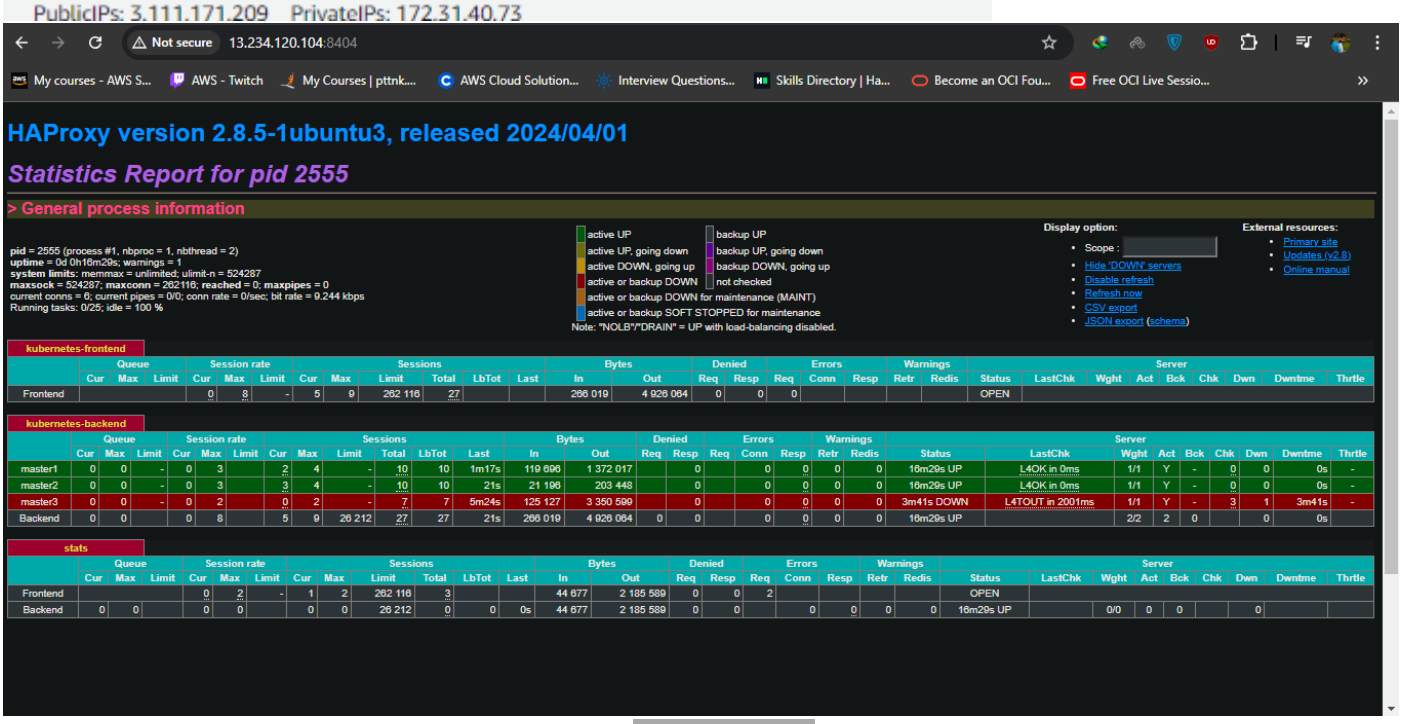
PublicIPs: 3.111.171.209 PrivateIPs: 172.31.40.73



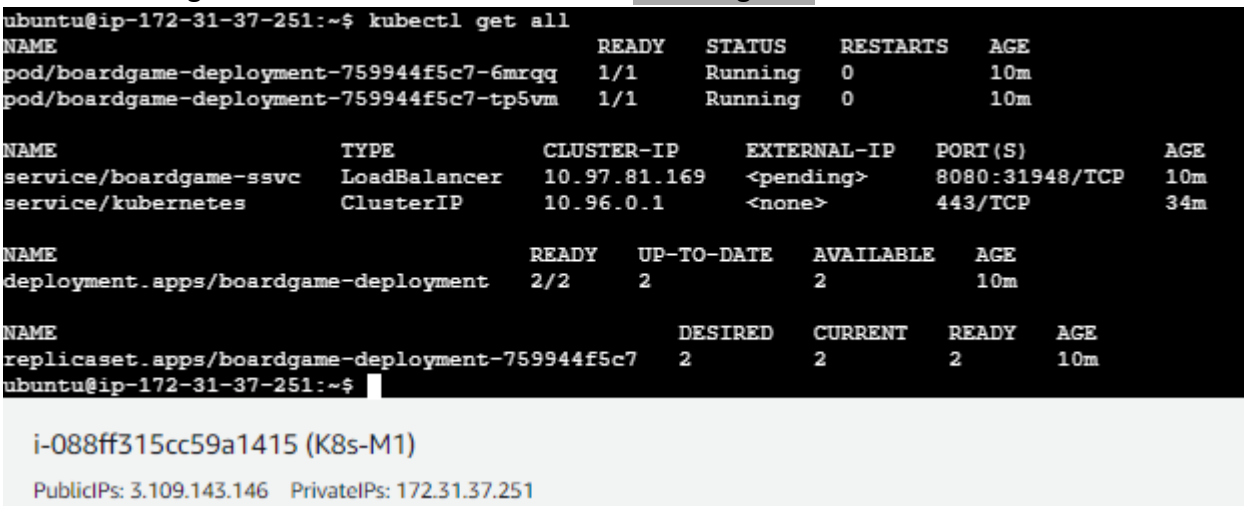
For checking our cluster is Highly Available or not we have to remove one Master Node,
On my case I removed Master Node-3 (M3)



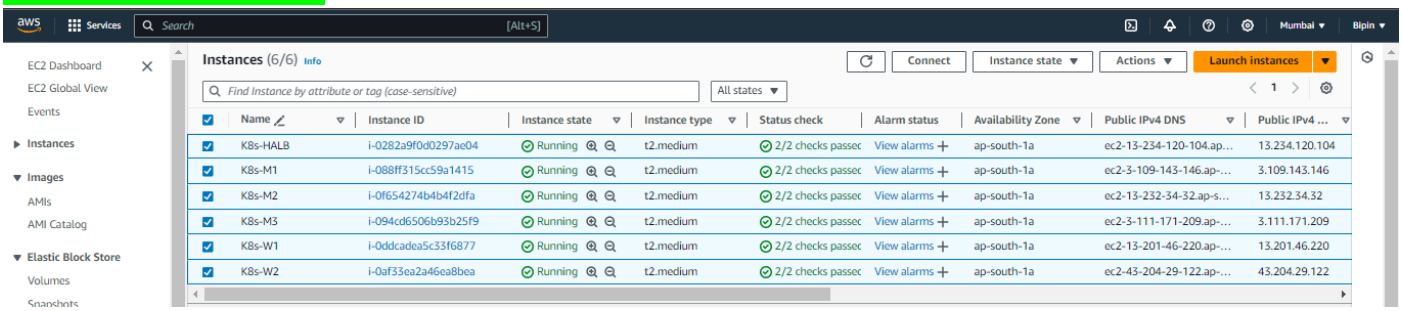




Now to check go to other master nodes and use kubectl get all

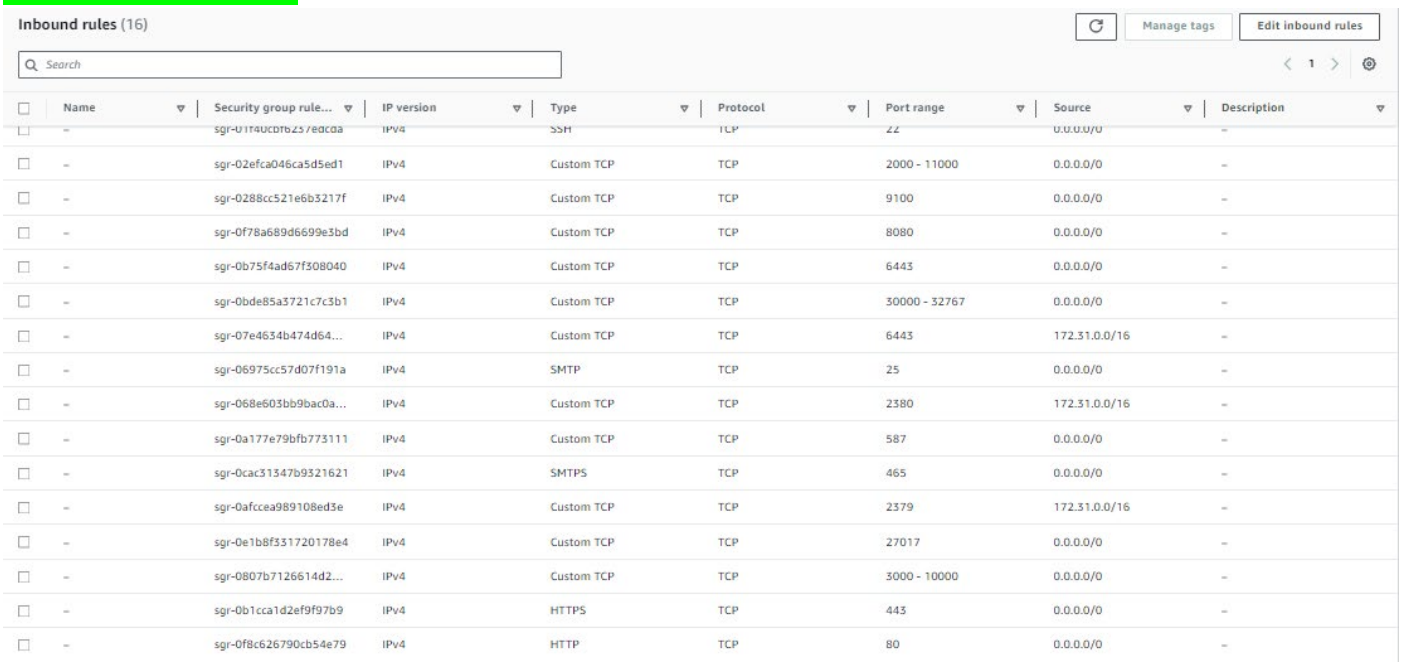



Now refresh the application again and check the status...!!!

## Number of Instances




## Ports that are used




## Acknowledgment

Special thanks to Aditya Jaiswal of "DevOps Shack" on YouTube for his invaluable guidance and tutorials, which were instrumental in the successful completion of this project. Thank you, Aditya!