High Availabulity Kubernetes Clister

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:

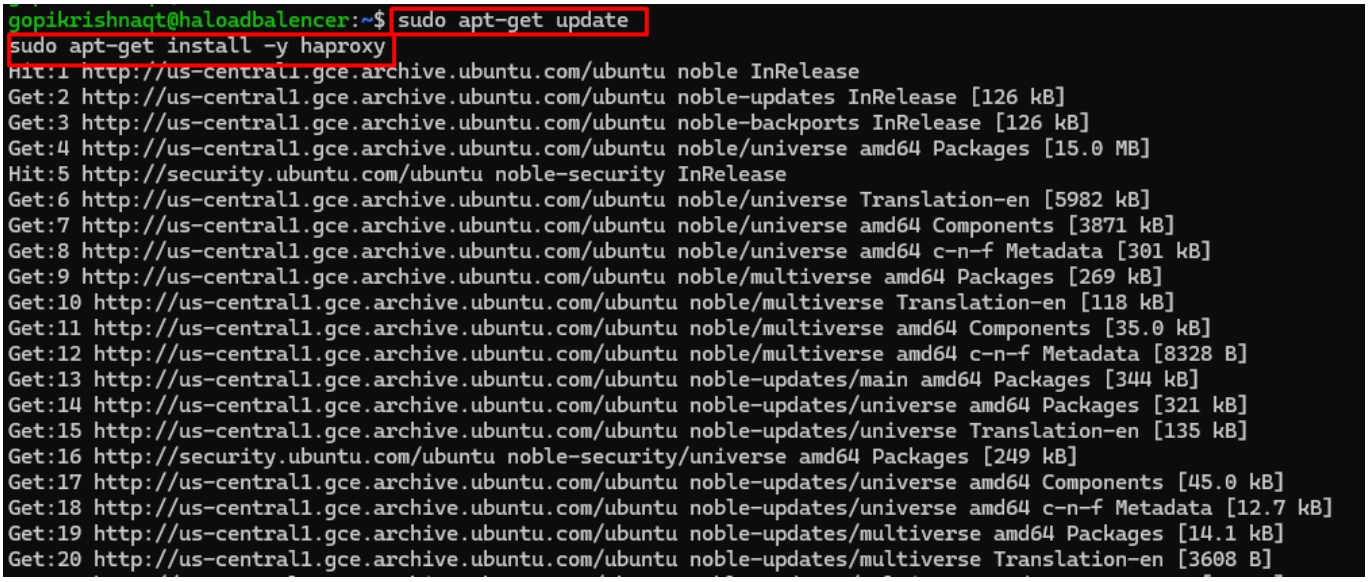
Prerequisites

- 2 master nodes
- 1 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. **Get the join command and certificate key from the first master node:**

```
kubeadm token create --print-join-command --certificate-key $(kubeadm init phase upload-certs --upload-certs | tail -1)
```

2. **Run the join command on the second master node:**

```
sudo kubeadm join LOAD_BALANCER_IP:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash> --control-plane --certificate-key <certificate-key>
```

3. **Set up kubeconfig for the second 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. **Get the join command from the first master node:**

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

2. **Run the join command on each worker node:**

```
sudo kubeadm join LOAD_BALANCER_IP:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

Step 6: Verify the Cluster

1. **Check the status of all nodes:**

```
kubectl get nodes
```

2. **Check the status of all pods:**

```
kubectl get pods --all-namespaces
```

By following these steps, you will have a highly available Kubernetes cluster with two master nodes and three 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:**

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

Step 2: Verify Etcd Cluster Health

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

```
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
```

2. **Check the cluster membership:**

```
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 member list
```

Step 3: 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`.

Step 4: Test High Availability

1. **Simulate Master Node Failure:**
   - Stop the kubelet service and Docker containers on one of the master nodes to simulate a failure:

```
sudo systemctl stop kubelet
sudo docker stop $(sudo docker ps -q)
```

2. **Verify Cluster Functionality:**
   - Check the status of the cluster from a worker node or the remaining master node:

```
kubectl get nodes
kubectl get pods --all-namespaces
```

   - The cluster should still show the remaining nodes as Ready, and the Kubernetes API should be accessible.

3. **HAProxy Routing:**
   - Ensure that HAProxy is routing traffic to the remaining master node. Check the stats page or use curl to test:

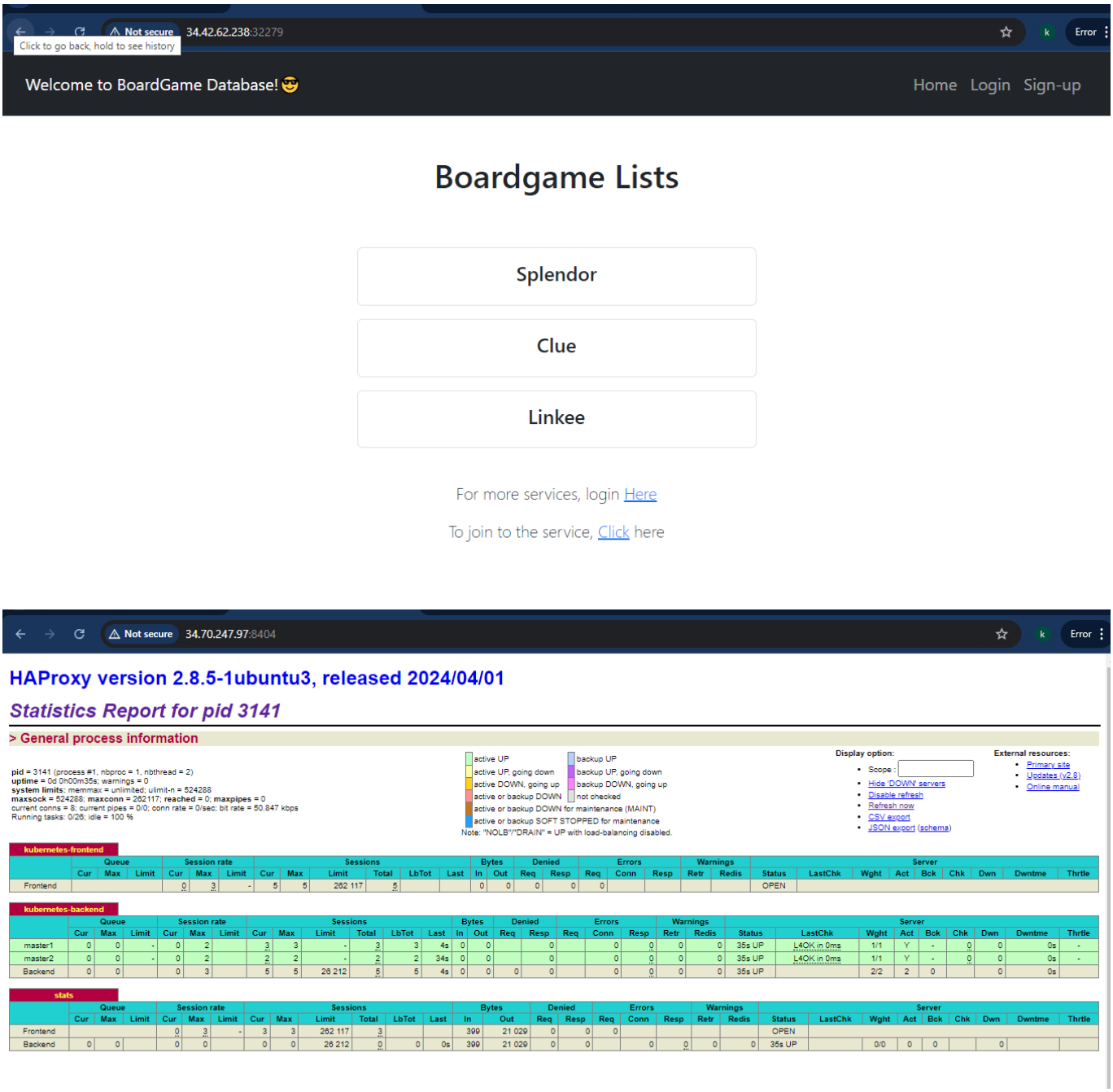
```
curl -k https://<LOAD_BALANCER_IP>:6443/version
```

Now deploy the application `deployment-service.yaml`

```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: adijaiswal/boardshack: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: 80
      targetPort: 8080
  type: LoadBalancer # type of the service.
```



Summary

By installing `etcdctl` and using it to check the health and membership of the etcd cluster, you can ensure that your HA setup is working correctly. Additionally, configuring HAProxy to route traffic properly and simulating master node failures will help verify the resilience and high availability of your Kubernetes cluster