**High Available- Kubernetes Cluster in AWS**

[**https://www.youtube.com/watch?v=JXbTyz1QIHI**](https://www.youtube.com/watch?v=JXbTyz1QIHI)

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**

**Install HAProxy:**

sudo apt-get update

sudo apt-get install -y haproxy

**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

**Restart HAProxy:**

sudo systemctl restart haproxy

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

**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**

**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

**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

**Install Calico network plugin:**

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

**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**

**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)

**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>

**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**

**Get the join command from the first master node:**

kubeadm token create --print-join-command

**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**

**Check the status of all nodes:**

kubectl get nodes

**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**

**Install etcdctl using apt:**

sudo apt-get update

sudo apt-get install -y etcd-client

**Step 2: Verify Etcd Cluster Health**

**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

**Check the cluster membership:**

ETCDCTL\_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt –c ert=/etc/kubernetes/pki/etcd/peer.crt --key=/etc/kubernetes/pki/etcd/peer.key member list

**Step 3: Verify HAProxy Configuration and Functionality**

**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

**Restart HAProxy:**

sudo systemctl restart haproxy

**Check HAProxy Stats:**

Access the stats page at http://<LOAD\_BALANCER\_IP>:8404.

**Step 4: Test High Availability**

**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)

**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.

**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

**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.

Security Group Inbound Rules:

A screenshot of a computer

Description automatically generated

AWS Nodes:

A screenshot of a computer

Description automatically generated