

High Availability Kubernetes 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

- 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
```

```
gopikrishnaqt@haloadbalancer:~$ sudo apt-get update
sudo apt-get install -y haproxy
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:8 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:9 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:10 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:11 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:12 http://us-central1.gce.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:13 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [344 kB]
Get:14 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [321 kB]
Get:15 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [135 kB]
Get:16 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [249 kB]
Get:17 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [45.0 kB]
Get:18 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [12.7 kB]
Get:19 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [14.1 kB]
Get:20 http://us-central1.gce.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [3608 B]
```

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`

```
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.
```

← → ↻ ⚠ Not secure 34.42.62.238:32279

Click to go back, hold to see history

☆ k Error

Welcome to BoardGame Database! 😊

Home Login Sign-up

Boardgame Lists

Splendor

Clue

Linkee

For more services, login [Here](#)

To join to the service, [Click](#) here

← → ↻ ⚠ Not secure 34.70.247.97:8404

☆ k Error

HAProxy version 2.8.5-1ubuntu3, released 2024/04/01

Statistics Report for pid 3141

> General process information

pid = 3141 (process #1, nbproc = 1, nbthread = 2)

uptime = 0d 0h00m35s; warnings = 0

system limits: memmax = unlimited; ulimit-n = 524288

maxsock = 524288; maxconn = 262117; reached = 0; maxpipes = 0

current conns = 8; current pipes = 0/0; conn rate = 0/sec; bit rate = 50.847 kbps

Running tasks: 0/26; idle = 100 %

active UP

active UP, going down

active DOWN, going up

active or backup DOWN

active or backup DOWN for maintenance (MAINT)

active or backup SOFT STOPPED for maintenance

backup UP

backup UP, going down

backup DOWN, going up

not checked

Display option:

Scope:

[Hide DOWN servers](#)

[Disable refresh](#)

[Refresh now](#)

[CSV export](#)

[JSON export \(schema\)](#)

External resources:

[Primary site](#)

[Updates/V2.8/](#)

[Online manual](#)

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

kubernetes-frontend

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend				0	3	-	5	5		262 117	5		0	0	0	0	0	0	0	0	0	0	OPEN								

kubernetes-backend

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
master1	0	0	-	0	2		3	3		-	2	3	4s	0	0		0	0	0	0	0	35s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
master2	0	0	-	0	2		2	2		-	2	2	34s	0	0		0	0	0	0	0	35s UP	L4OK in 0ms	1/1	Y	-	0	0	0s	-	
Backend	0	0		0	3		5	5		26 212	5	5	4s	0	0	0	0	0	0	0	0	35s UP		2/2	2	0		0	0	0s	

stats

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle		
Frontend				0	3	-	3	3		262 117	3		399	21 029	0	0	0	0	0	0	0	0	OPEN									
Backend	0	0		0	0		0	0		26 212	0	0	0s	399	21 029	0	0	0	0	0	0	0	35s UP		0/0	0	0		0	0	0s	

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