DevOps Take Home
Arash Foroughi

Use this procedure document to install and setup kubernetes cluster plus running a sample web server & DB.

Document Statistics

Type of Information	Document Data
Title	Devops-Take-Home-Arash-Foroughi
Document Revision #	1.0
Last Date Document Update	11-Jul-2022
Total Number of Pages	12
Document Filename	Devops_Take_Home_Arash_Foroughi_V1.0.docx
Document Author	Arash Foroughi
Revision Author	
Change Records	

Table of Contents

Document Statistics 2
1. Set up a Highly Available Kubernetes Cluster using kubeadm: 3
2. Setup Load Balancer Node: 3
3. Setup Ansible host inventory and its configuration: 4
4. Setup Kubernetes cluster using Ansible Playbooks: 5
5. Setup a Voting Application as Sample Scenario: 6

1. Set up a Highly Available Kubernetes Cluster using kubeadm:

Follow this documentation to set up a highly available Kubernetes cluster using Ubuntu 22.04 LTS.

This documentation guides you in setting up a cluster with two master nodes, two worker nodes and a load balancer node using HAProxy.

Role	**FQDN**	**IP**	**OS**	**RAM**	**CPU**
Master-HA	loadbalancer.example.com	192.168.44.153	Ubuntu 22.04	1G	2
Master-1	kmaster1.example.com	192.168.44.154	Ubuntu 22.04	2G	4
Master-2	kmaster2.example.com	192.168.44.155	Ubuntu 22.04	2G	4
Worker-1	kworker1.example.com	192.168.44.156	Ubuntu 22.04	1G	2
Worker-2	kworker2.example.com	192.168.44.157	Ubuntu 22.04	1G	2

2. Setup Load Balancer Node:

Install haproxy:

```
# apt update
# apt install haproxy -y
```

Configure haproxy:

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

```
frontend kubernetes-frontend
    bind 192.168.44.153:6443
    mode tcp
    option tcplog
    default_backend kubernetes-backend

backend kubernetes-backend
    mode tcp
    option tcp-check
    balance roundrobin
    server kmaster1 192.168.44.154:6443 check fall 3 rise 2
    server kmaster2 192.168.44.155:6443 check fall 3 rise 2
```

Restart haproxy service:

```
# systemctl restart haproxy
```

HAproxy load balancer server has been configured successfully, now in this step needs to setup kubernetes on all Master/Worker nodes.

Here there are different ways, if you are using regular Virtual Machines, you can use ansible for installing kubernetes on your servers.

3. Setup Ansible host inventory and its configuration:

Login to loadbalancer to set it up as Ansible Controller machine and install Ansible on it:

```
# apt install ansible -y

# ansible --version
```

Share the root public key of loadbalancer with all other nodes to make their connection without password. Here, we decide to use root user.

```
# ssh-keygen

# ssh-copy-id root@192.168.44.154

# ssh-copy-id root@192.168.44.155

# ssh-copy-id root@192.168.44.156

# ssh-copy-id root@192.168.44.157
```

Prepare the Ansible Host Inventory file per below information:

```
# mkdir /etc/ansible

# vim /etc/ansible/hosts
```

```
[all]
kmaster1 ansible_host=192.168.44.154 ansible_connection=ssh ansible_user=root
kmaster2 ansible_host=192.168.44.155 ansible_connection=ssh ansible_user=root
kworker1 ansible_host=192.168.44.156 ansible_connection=ssh ansible_user=root
kworker2 ansible_host=192.168.44.157 ansible_connection=ssh ansible_user=root

[masters]
kmaster1 ansible_host=192.168.44.154 ansible_connection=ssh ansible_user=root
kmaster2 ansible_host=192.168.44.155 ansible_connection=ssh ansible_user=root

[workers]
kworker1 ansible_host=192.168.44.156 ansible_connection=ssh ansible_user=root
kworker2 ansible_host=192.168.44.157 ansible_connection=ssh ansible_user=root
```

for verification which the ansible controller is working properly, use Ping Module to test the connectivity between nodes:

```
# ansible --list-hosts all
# ansible all -m ping
# ansible masters -m ping
# ansible workers -m ping
```

4. Setup Kubernetes cluster using Ansible Playbooks:

Login to loadbalancer node, find Setup_K8s_Cluster_Playbook.yaml from the package and run it with below command:

```
# ansible-playbook K8s_Installation_Playbook.yaml -v
```

After finishing the previous Ansible playbook, it means you have successfully installed Docker and Kubernetes prerequisites on all Master/Worker nodes successfully.

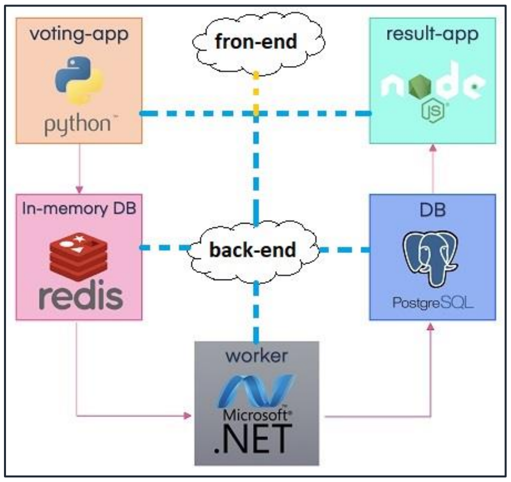
Now it's time to setup your kubernetes cluster and specify role of each node in the cluster with running below playbook again from loadbalancer node which has also Ansible_Controller role in this scenario:

```
# ansible-playbook Setup_K8s_Cluster_Playbook.yaml -v
```

The Kubernetes cluster now has been created successfully. You can always check the status of nodes and pods with below commands:

```
# kubectl get nodes -o wide
# kubectl get pods -A -o wide
```

5. Setup a Voting Application as Sample Scenario:



Now it’s time to deploy a scenario as pods on our Kubernetes cluster with below procedure. In below picture, you can find a sample voting app, which it contains 2 web services, one for voting and one for checking the result. In this scenario when someone votes, the vote will be written in a Redis DB and a .Net worker as backend will write it in a physical DB which it is Postgresql and at the end, result application will show read from this DB and show the result.

1. In the repository package, first run below commands in the kmaster1 to build the docker images from Dockerfiles:

```
# docker build -t 9354165450/example:voteapp ./vote/
# docker build -t 9354165450/example:resultapp ./result/
# docker build -t 9354165450/example:workerapp ./worker/
# docker image ls
```

2. Then you may need to push the docker images to a registry or simply if number of workers are not too much, get backup from images and upload them to all worker nodes:

```
# docker push 9354165450/example:voteapp
# docker push 9354165450/example:resultapp
# docker push 9354165450/example:workerapp
```

3. First start with Voteapp webserver and its service to be reachable from outside of the cluster:

```
## Vote-App Web Server Deployment ##
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vote
  labels:
    app: vote
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
      - name: vote
        image: 9354165450/example:voteapp
        ports:
        - containerPort: 80
          name: vote
---
## Vote-App Web Server SVC ##
apiVersion: v1
kind: Service
metadata:
  name: vote
  labels:
    apps: vote
spec:
  type: NodePort
  ports:
    - port: 5000
      targetPort: 80
      nodePort: 30008
      name: vote-service
  selector:
    app: vote

# kubectl apply -f vote-app.yaml
# kubectl get deployments -o wide
# kubectl get pods -o wide
# kubectl get svc -o wide
```

4. Second deployment needs to run is Redis Database plus its service to be reachable just from inside of the Cluster to be linked with the Vote-App webserver:

```
## redis Database Deployment ##
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: redis:alpine
        ports:
        - containerPort: 6379
          name: redis
---
## redis Database Service ##
apiVersion: v1
kind: Service
metadata:
  labels:
    app: redis
  name: redis
spec:
  clusterIP: None
  ports:
  - name: redis-service
    port: 6379
    targetPort: 6379
  selector:
    app: redis

# kubectl apply -f redis-db.yaml
# kubectl get deployments -o wide
# kubectl get pods -o wide
# kubectl get svc -o wide
```

5. Next part is deploying Worker-App which it will play role of backend in this scenario:

```
## Worker-App Deployment ##
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: worker
  name: worker
spec:
  replicas: 1
  selector:
    matchLabels:
      app: worker
  template:
    metadata:
      labels:
        app: worker
    spec:
      containers:
```

```
      - image: 9354165450/example:workerapp
        name: worker
---
## Worker-App Service ##
apiVersion: v1
kind: Service
metadata:
  labels:
    apps: worker
  name: worker
spec:
  clusterIP: None
  selector:
    app: worker

# kubectl apply -f worker-app.yaml
# kubectl get deployments -o wide
# kubectl get pods -o wide
# kubectl get svc -o wide
```

6. Now it’s time to deploy Postgres Database:

```
---
## Postgres Database Deployment ##
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
  labels:
    app: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: db
        image: postgres:9.4
        env:
        - name: PGDATA
          value: /var/lib/postgresql/data/pgdata
        - name: POSTGRES_USER
          value: postgres
        - name: POSTGRES_PASSWORD
          value: postgres
        ports:
        - containerPort: 5432
          name: db
## Postgres Database Service ##
```

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: db
  name: db
spec:
  clusterIP: None
  ports:
  - name: db
    port: 5432
    targetPort: 5432
  selector:
    app: db
## Postgres Persistent Volume ##
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv-volume
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
## Postgres Persistent Volume Claim ##
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pv-claim
spec:
  volumeMode: Filesystem
  volumeName: postgres-pv-volume
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi

# kubectl apply -f postgres-db.yaml
# kubectl get deployments -o wide
# kubectl get pods -o wide
# kubectl get svc -o wide
```

7. And finally, just needs to deploy Result-App:

```
## Result-App Deployment ##
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: result
  labels:
    app: result
spec:
  replicas: 1
  selector:
    matchLabels:
      app: result
  template:
    metadata:
      labels:
        app: result
    spec:
      containers:
      - name: result
        image: resultapp
        ports:
        - containerPort: 80
          name: result
---
## Result-App Service ##
apiVersion: v1
kind: Service
metadata:
  name: result
  labels:
    app: result
spec:
  type: NodePort
  ports:
  - targetPort: 80
    port: 5001
    nodePort: 30009
    name: result-service
  selector:
    app: result

# kubectl apply -f result-app.yaml
# kubectl get deployments -o wide
# kubectl get pods -o wide
# kubectl get svc -o wide
```

8. Now you can go in your browser to **loadbalancer.example.com:30008** to vote
   And go to **loadbalancer.exmple.com:30009** to check the results!

Thanks for Your Attention!