TP Kubernetes

Kubernetes is an open-source container management system, freely available. It provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. Kubernetes gives you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, releasing organizations from tedious deployment tasks.

In this pratical class, we are going to:

- setup multi-node Kubernetes Cluster on Ubuntu 20.20 server;

- deploy an application and manage it on our deployed Kubernetes;

Prerequisites

- Two virtual machines or physical machines, known as nodes, with ubuntu 20.20 server installed.
- A static IP address (masterIP) is configured on the first nodes (master node) and also a static IP (slaveIP) is configured on the second instance (slave node).
- Minimum 4 GB RAM and 2 vCPU per node.
- root password is setup on each instance "toto".

Connection to your nodes

Use the following commands to access your server :

The master node port is 130XX and the slave node 130XX+1

```
ssh ubuntu@147.127.121.83 -p 130XX #connection to master node
ssh ubuntu@147.127.121.83 -p 130XX+1 #connection to slave node
```

Where XX=01-40. This will give you acces to a VM with IP address 192.168.27.**(XX+10)** and the password is "*toto*"

```
sudo bash
apt-get update -y # On both node
```

Node configurations

You need to configure “hosts” file and hostname on each node in order to allow a network communication using the hostname. You begin by setting the master and slave node names.

On the master node run:

```
hostnamectl set-hostname master
```

On the slave node run :

```
hostnamectl set-hostname slave
```

You need to configure the hosts file. Therefore, run the following command on both nodes:

```
echo "slaveIP   slave" >> /etc/hosts
echo "masterIP  master" >> /etc/hosts
```

You have to disable swap memory on each node. kubelets do not support swap memory and will not work if swap is active. Therefore, you need to run the following command on both nodes:

```
swapoff -a
```

Docker installation

Docker must be installed on both the master and slave nodes. You start by installing all the required packages.

```
sudo apt install ca-certificates curl gnupg lsb-release -y

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo
"$ID")/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/$(. /etc/os-
release; echo "$ID") $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

apt-get update

apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin -y
```

Kubernetes installation

Next, you will need to install: kubeadm, kubectl and kubelet on both nodes.

```
modprobe br_netfilter
```

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

```
sysctl --system
```

```
sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl gpg

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.34/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:/v1.34/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

sudo systemctl enable --now kubelet
```

Good, all the required packages are installed on both servers.

We need to configure kubernetes on **both nodes** to use the correct docker driver.

As a root user, open the file *"/etc/systemd/system/kubelet.service.d/10-kubeadm.conf*" and edit the "**KUBELET_CONFIG_ARGS**" configuration, i.e add " **--cgroup-driver=cgroupfs** " as an argument, then reload systemd

```
rm /etc/containerd/config.toml

systemctl daemon-reload

systemctl restart containerd
```

NB: This operation has to be carry-out on both nodes, i.e slave and master

Master node configuration

Now, it's time to configure Kubernetes master Node. First, initialize your cluster using its private IP address with the following command:

```
systemctl restart containerd

kubeadm init --pod-network-cidr=10.244.0.0/16
```

Note:

- *pod-network-cidr* = specify the range of IP addresses for the pod network.

You should see the following output:

You have to save the '**kubeadm join ... ...**' command. The command will be used to register new worker nodes to the kubernetes cluster.

To use Kubernetes, you must run some commands as shown in the result (**as root**).

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

We can check the status of the master node by running the following command:

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

y`ou` can observe form the above output that the master node is listed as not ready. This is because the cluster does not have a Container Networking Interface (CNI).
Next, deploy the flannel network to the kubernetes cluster using the kubectl command.

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

Wait for a minute and check kubernetes node and pods using commands below.

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

You should see the following output:

And you will get the 'kube-scheduler-master' node is running as a 'master' cluster with status 'ready'.

Kubernetes cluster master initialization and configuration has been completed.

Slave node configuration

Next, we need to log in to the slave node and add it to the cluster. Remember **the join command in the output from the Master** Node initialization command and issue it on the Slave Node as shown below:

```
sudo kubeadm join ``xxx``.``xxx``.``xxx``.``xxx``:6443 --token
wg42is.1hrm4wgvd5e7gbth     --discovery-token-ca-cert-hash
sha256:53d1cc33b5b8efe1b974598d90d250a12e61958a0f1a23f864579dbe67f83e30
```

Once the Node is joined successfully, you should see the following output:

Now, go back to the master node and run the command “kubectl get nodes” to see that the slave node is now ready (You may wait for some second for the node to be in ready state):

```
kubectl get nodes
```

Apache2 deployment (some fun)

We will deploy a little application in our Kubernetes cluster, apache2 web server with a simple index.php application.

Now, you have to create the apache Deployment YAML file 'apache-deployment.yaml' and paste the following content.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
spec:
  selector:
    matchLabels:
      run: apache-app
  replicas: 2
  template:
    metadata:
      labels:
        run: apache-app
    spec:
      containers:
      - name: apache-container
        image: teabetab/apache:v1
        ports:
        - containerPort: 80
```

Note:

- We're creating a new 'Deployment' named 'apache-deployment'.
- Setup the app label as 'apache-app' with 2 replicas.
- The 'apache-deployment' will have containers named 'apache-container', based on 'apache-image' docker image, and will expose the default HTTP port 80.

You can submit the deployment by running the kubectl command below.

```
kubectl create -f apache-deployment.yaml
```

After creating a new 'apache-deployment', check the deployments list inside the cluster.

```
kubectl describe deployment apache-deployment
```

Check the nodes the Pod is running on:.

```
kubectl get pods -l run=apache-app -o wide
```

Check your pods' IPs:

```
kubectl get pods -l run=apache-app -o yaml | grep podIP
```

We need to create a new service for our 'apache-deployment'. Therefore, create a new YAML file named 'apache-service.yaml' with the following content.

```
apiVersion: v1
kind: Service
metadata:
  name: apache-service
  labels:
    run: apache-app
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: apache-app
```

Note:

- We're creating a new kubernetes service named 'apache-service'.
- The service belongs to the app named "apache-app' based on our deployment 'apache-deployment'.

To list the pods created by the deployment:

```
 kubectl get pods -l app=apache-app
```

Create the Kubernetes service using the kubectl command below.

```
kubectl create -f apache-service.yaml
```

Now check all available services on the cluster and you will get the 'apache-service' on the list, then check details of service.

```
kubectl get service
kubectl describe service apache-service
```

Accessing the service

```
kubectl scale deployment apache-deployment --replicas=0
kubectl scale deployment apache-deployment --replicas=2
kubectl get pods -l run=apache-app -o wide
```

Copy the clusterIP and use it to access your application, index.php

The command should be run from the slave node, not the master. This is a limitation of flannel that just allows accessing the application from slave nodes.

```
curl <clusterIP>/index.php
```

You can observe that there is a load balancing by app-service on the containers deployed.

Apache Deployment

You will demonstrate that you followed the session by deploying the apache architecture of last class in your Kubernetes cluster (2 tomcats instance and 1 service, no need to use the haproxy.cfg file).

Good luck!