K8S – 5

1. Create a Kubernetes cluster using kubeadm

* Create 3 ubuntu vms which are accesible to each other with atlest 2 vCPUS and 4 GB RAM ( Master , node1 , node2 )
* Install docker on all nodes
* Install CRI-Dockerd :

# Run these commands as root (on all the three nodes)

* wget https://storage.googleapis.com/golang/getgo/installer\_linux
* chmod +x ./installer\_linux
* ./installer\_linux
* source ~/.bash\_profile
* git clone <https://github.com/Mirantis/cri-dockerd.git>
* cd cri-dockerd
* mkdir bin
* go build -o bin/cri-dockerd
* mkdir -p /usr/local/bin
* install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd
* cp -a packaging/systemd/\* /etc/systemd/system
* sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service
* systemctl daemon-reload
* systemctl enable cri-docker.service
* systemctl enable --now cri-docker.socket

# Installing kubeadm, kubelet, kubectl without package manager :

# Run these commands again as a root user :

* sudo apt-get update
* sudo apt-get install -y apt-transport-https ca-certificates curl
* sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg <https://packages.cloud.google.com/apt/doc/apt-key.gpg>
* echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | 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

# Creating cluster from the master node :

# To start using your cluster, run the following as a regular user:

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

# Now to initialize the kubeadm :

kubeadm init --pod-network-cidr "10.244.0.0/16" --cri-socket "unix:///var/run/cri-dockerd.sock"

# Settingup kubeconfig :

* install flannel kubectl apply -f <https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml>

# As a root user run kubeadm join command ( adding the cri-socket ):

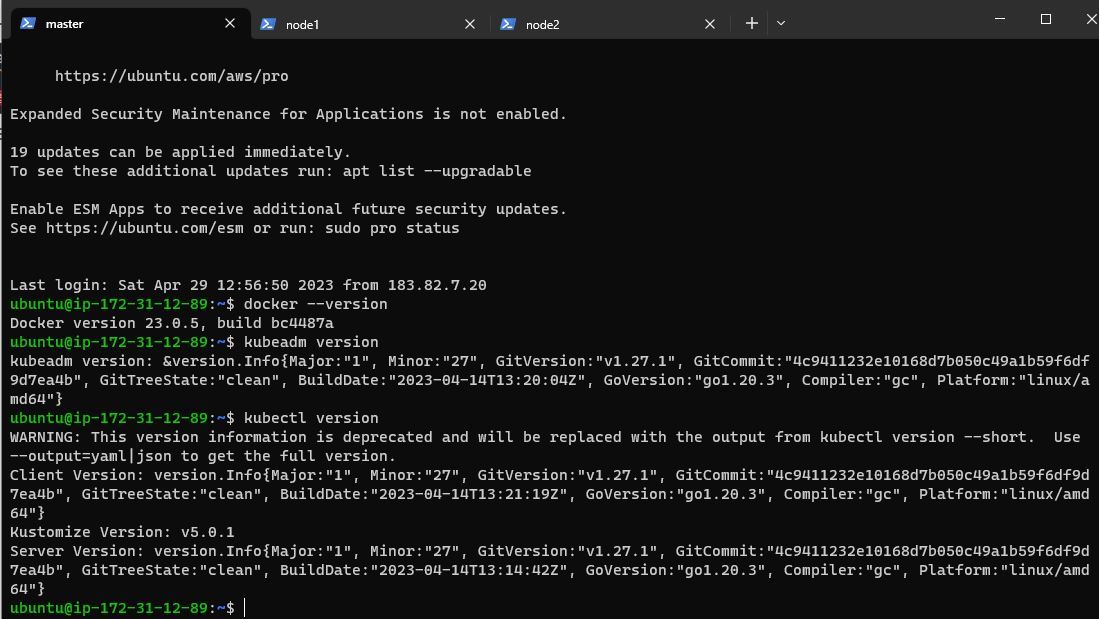
* kubeadm join <control-plane-host>:<control-plane-port> --token <token> --

---<cri-socket>---

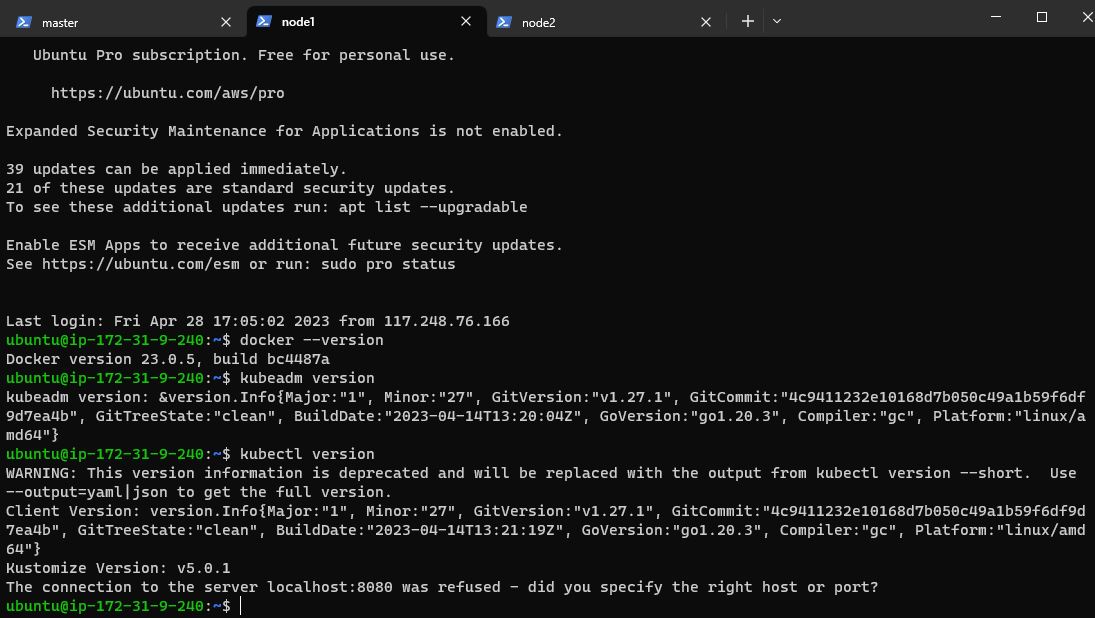
discovery-token-ca-cert-hash sha256:<hash>

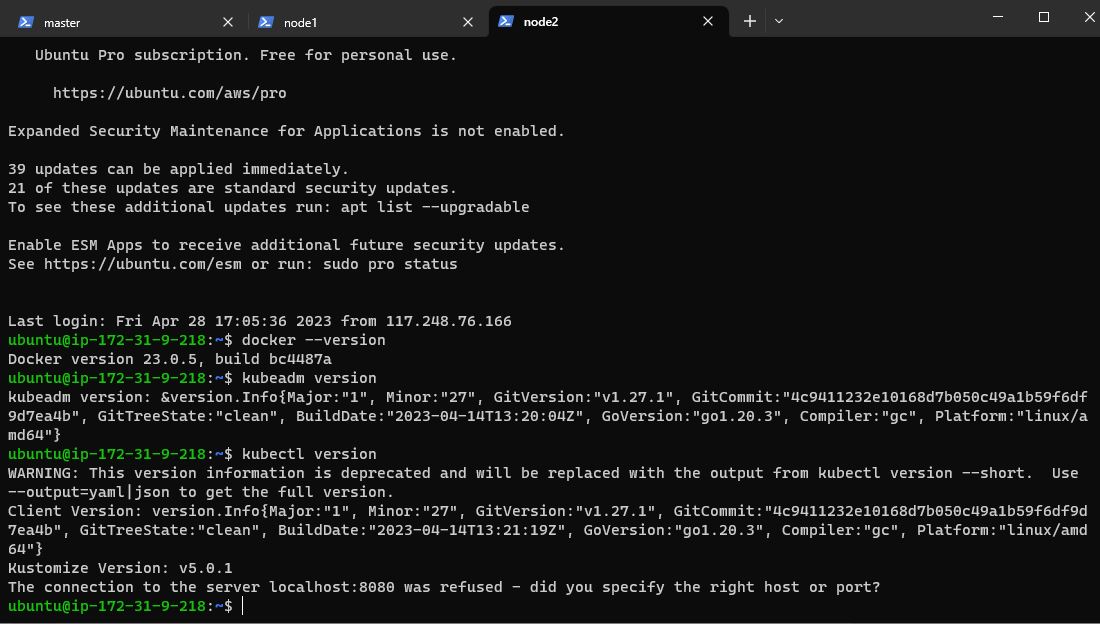
# Now execute

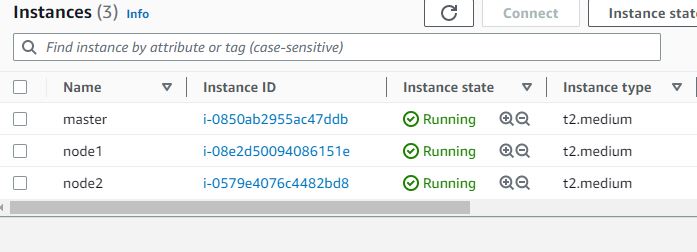
* Kubectl get nodes
* Master node



* Node 1



* Node 2
*  EC2



1. Deploy any application using kubectl

nginx-rs-deploy-svc.yml :

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: nginx-rs

  labels:

    app: nginx

  namespace: kube-system

spec:

  minReadySeconds: 2

  replicas: 1

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      name: nginx-rs

      labels:

        app: nginx

    spec:

      containers:

        - name: nginx

          image: nginx:1.23

          ports:

            - containerPort: 80

              protocol: TCP

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deploy

  labels:

    app: nginx

spec:

  minReadySeconds: 2

  replicas: 1

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      name: nginx-deploy

      labels:

        app: nginx

    spec:

      containers:

        - name: nginx-deploy

          image: nginx:1.23

          ports:

            - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: nginx-svc

  labels:

    app: nginx

spec:

  selector:

    app: nginx

  type: LoadBalancer

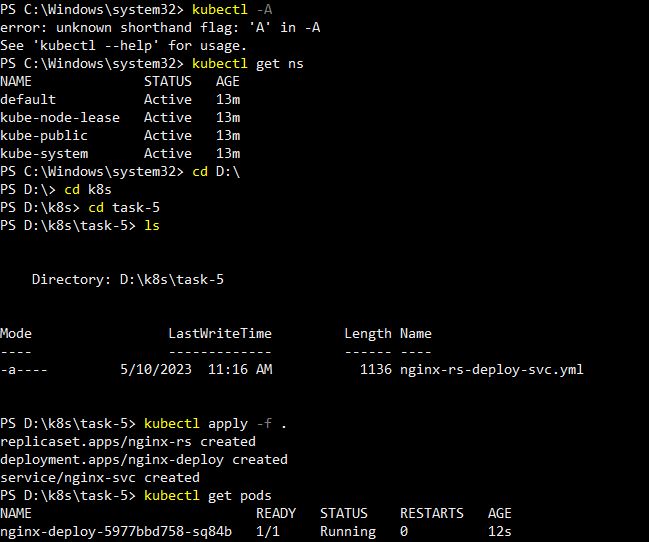
  ports:

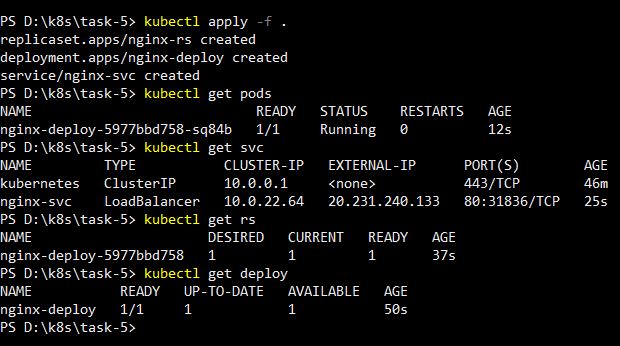
    - name: nginx-svc

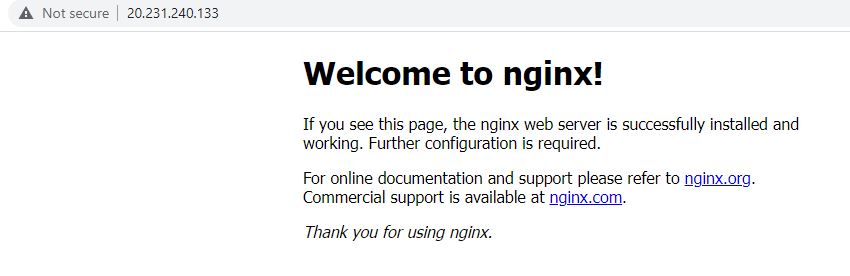
      port: 80

      protocol: TCP

      targetPort: 80

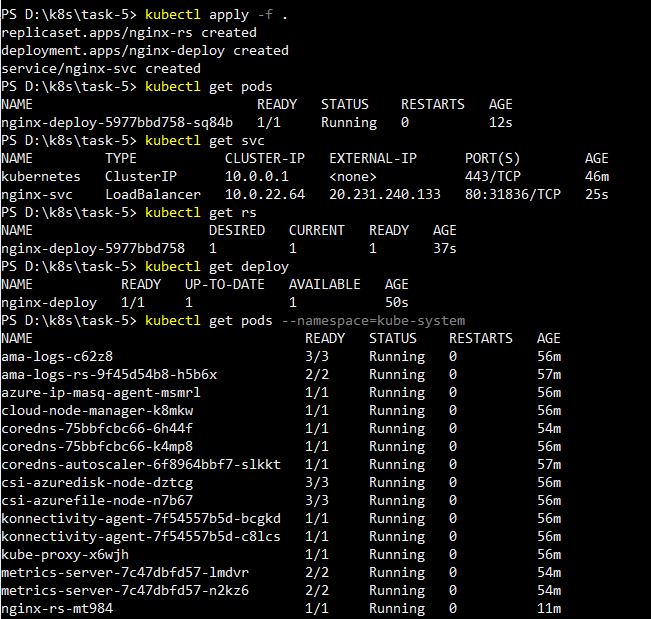






1. Backup Kubernetes i.e backup etcd
2. List out all the pod’s running in kube-system namespace

* Used application deployment and displayed :



* kubectl run nginx --image=nginx --namespace=kube-system
* kubectl get pods --namespace=kube-system

1. Write down all the steps required to make Kubernetes highly available
2. Do a rolling update and rollback
3. Ensure usage of secret in MYSQL and configmaps
4. Create a nop commerce deployment with MYSQL Statefulset and nop deployment

Nop-deploy.yml :

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nop-deploy

labels:

app: nop

spec:

minReadySeconds: 2

replicas: 1

selector:

matchLabels:

app: nop

template:

metadata:

name: nop-deploy

labels:

app: nop

spec:

containers:

- name: nop-deploy

image: harikasvl/nop:2.0

ports:

- containerPort: 5000

command:

- sleep

- 1d

---

apiVersion: v1

kind: Service

metadata:

name: nop-svc

labels:

app: nop

ver: "2.0"

spec:

selector:

app: nop

ports:

- name: nop-svc

port: 5000

protocol: TCP

targetPort: 5000

type: LoadBalancer

mysql-sts.yml :

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql-sts

labels:

app: mysql

spec:

minReadySeconds: 3

serviceName: mysql-svc

replicas: 1

selector:

matchLabels:

app: mysql

template:

metadata:

name: mysql-sts

labels:

app: mysql

spec:

containers:

- name: mysql-sts

image: mysql:8.0.33

volumeMounts:

- mountPath: /var/lib/mysql

name: mysql-svc

ports:

- name: mysql-sts

containerPort: 3306

protocol: TCP

env:

- name: MYSQL\_ROOT\_PASSWORD

value: password

- name: MYSQL\_USER

value: user

- name: MYSQL\_USER\_PASSWORD

value: password

- name: MQSQL\_DATABASE

value: Student

volumeClaimTemplates:

- metadata:

name: claimname

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

storageClassName: managed

---

apiVersion: v1

kind: Service

metadata:

name: mysql-svc

labels:

app: mysql

spec:

type: ClusterIP

clusterIP: None

selector:

app: mysql

ports:

- port: 3306

targetPort: 3306

protocol: TCP

---