**Kubernetes Part 1 – Assignment :**

**Problem Statement:**

You work for the xyz organization. Your organization uses Kubernetes for container orchestration. Your organization has recently created pods from which data was being lost. Now they require volume mounts which preserve data and save a password called “xyzlsthebest” and this has to be put on a particular node of your choice.

**You have been asked to:**

1. **Create a persistent volume**
2. **Create a persistent volume claim**
3. **Create a secret “xyzlsthebest”**
4. **Taint one of the nodes of the cluster**

Presistent Volume Claim:

In order to use a pv you need to claim it first, using a presistent volume claim(pvc).

The pvc request a pv with your required specification (size, access modes, speed etc) from kubernetes and once a suitable persistent volume is found, it is bound to a persistent volume claim.

After a successfull bound to a pod, you can mount it as a volume.

An aws EBS volume mounts an AWS EBS volume into your pod Unlike emptDir, which is erased when a pod is removed, the contents of an EBS volume are preserved and the volume is merely Unmounted.

PERSISTENT VOLUME:

1) created EC2 instances.

2) Goto volume and create volume (must be in same availability zone as default volume of ec2)

3) copy volume ID (created volume)

vi mypv.yml

apiVersion: v1

kind: PersistentVolume

metadata:

name: myebsvol

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Recycle

awsElasticBlockStore:

volumeID: vol-0607b37170d317af1 # ec2 volume ID

fsType: ext4

kubectl apply -f mypc.yml

kubectl get pv # list volume



Now create persistentvolumeclaim

vi mypvc.yml # now need to claim of volume size

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: myebsvolclaim # this volume allocated 1Gb from main volume and used in below pod

spec:

accessModes:

- ReadWriteOnce

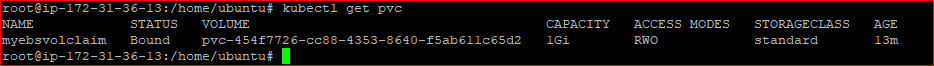
resources:

requests:

storage: 1Gi

kubectl apply -f mypvc.yml

kubectl get pvc

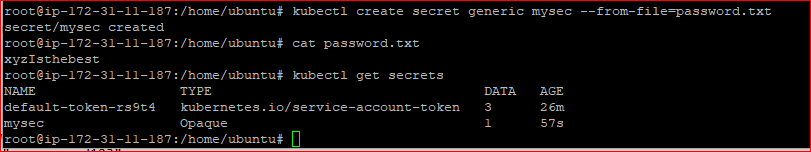


* **Create a secret “xyzlsthebest”**

**echo “xyzlsthebest” > password.txt**

**Now create secret with password:**

**kubectl create secret generic mysecret --from-file=password.txt**



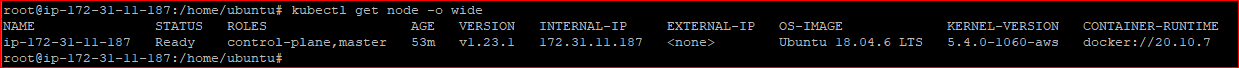
**Now deploy this secret with pod**

And this secret is found in container.

**Taint one of the nodes of the cluster as below.**

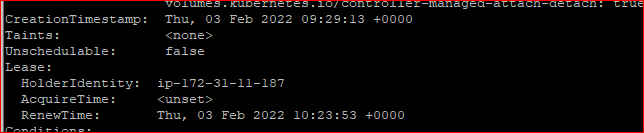
Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not toleration the taints.

Toleration are applied to pods, the pods to schedule onto nodes with matching taints.



**kubectl describe node ip-172-31-11-187 # check hardware details. and now Taints is None**

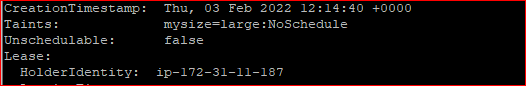
**Now Default Taint is showing None:**



**kubectl taint node ip-172-31-11-187 mysize=large:NoSchedule # create taint**

**Now Node is Tainted.**

**kubectl describe node ip-172-31-11-187**



**Now pod will not be created on specify particular node ip-172-31-11-187 with help of taint and with help of toleration we can schedule pod on particular node. And this container has secret and attached persistent volume and claim.**

**root@ip-172-31-11-187:/home/ubuntu# cat deploypvc.yml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: pvdeploy**

**spec:**

**replicas: 1**

**selector: # tells the controller which pods to watch/belong to**

**matchLabels:**

**app: mypv**

**template:**

**metadata:**

**labels:**

**app: mypv**

**spec:**

**containers:**

**- name: shell**

**image: centos**

**command: ["bin/bash", "-c", "sleep 10000"]**

**volumeMounts:**

**- name: testsecret**

**mountPath: "/tmp/vol\_data"**

**- name: mysecret1**

**mountPath: "/tmp/mysecrets1"**

**tolerations:**

**- key: "mysize"**

**operator: "Equal"**

**value: "large"**

**effect: "NoSchedule"**

**volumes:**

**- name: testsecret**

**persistentVolumeClaim:**

**claimName: myebsvolclaim**

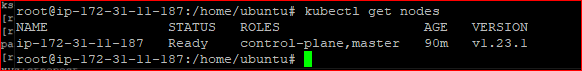
**- name: mysecret1**

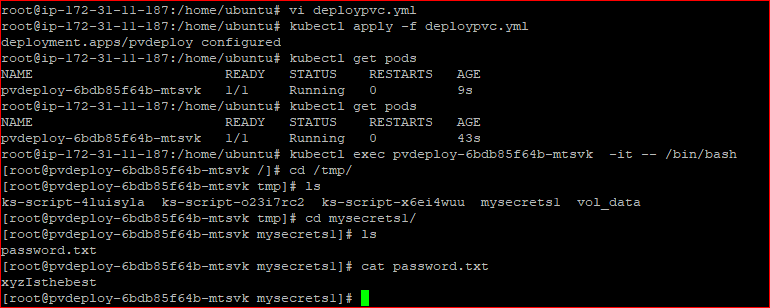
**secret:**

**secretName: mysecret**

**root@ip-172-31-11-187:/home/ubuntu#**

**Now pod is running on specify nodes.**

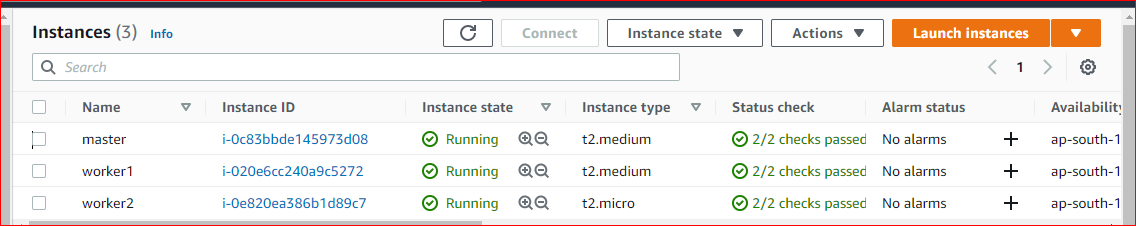




**Module-9: Kubernetes Assignment-1**

**You have been asked to:**

1. Deploy a kubernetes Cluster for 3 nodes
2. Create a nginx deployment of 3 replicas



**ubuntu@ip-172-31-11-186:~$ cat nginx.yml**

**kind: Deployment**

**apiVersion: apps/v1**

**metadata:**

**name: mydeployments**

**spec:**

**replicas: 3**

**selector: # tells the controller which pods to watch/belong to**

**matchLabels:**

**name: deployment**

**template:**

**metadata:**

**name: testpod1**

**labels:**

**name: deployment**

**spec:**

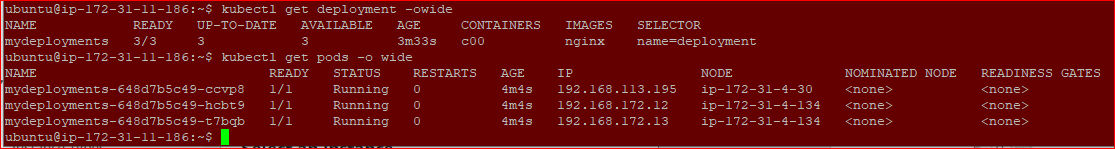
**containers:**

**- name: c00**

**image: nginx**

**ports:**

**- containerPort: 80**



**Module-9: Kubernetes Assignment-2**

**You have been asked to:**

1. Use the previous deployment
2. Create a service of type NodePort for nginx deployment
3. Check the nodeport service on a browser to verify

ubuntu@ip-172-31-11-186:~$ cat service.yml

kind: Service # Defines to create Service type Object

apiVersion: v1

metadata:

name: demoservice

spec:

ports:

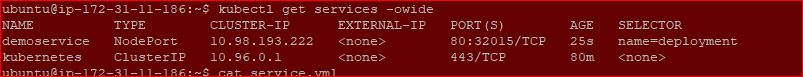
- port: 80 # Containers port exposed

targetPort: 80 # Pods port

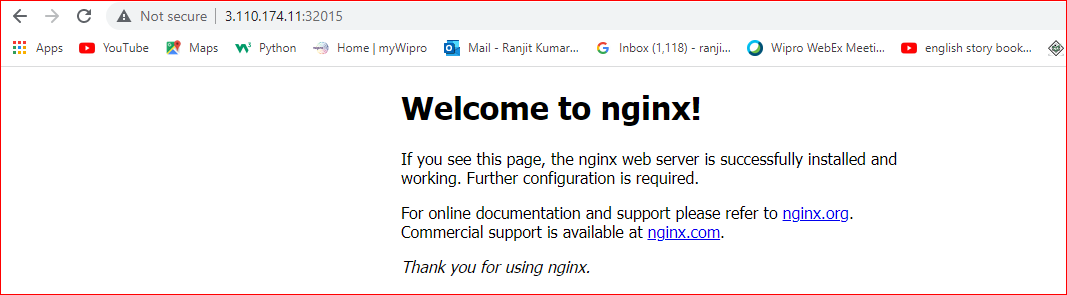
selector:

name: deployment # Apply this service to any pods which has the specific label

type: NodePort



**URL:** **http://3.110.174.11:32015/**



**Module-9: Kubernetes Assignment-3**

**You have been asked to:**

1. **Use the previous deployment**
2. **Change the replicas to 5 for the deployment**

kind: Deployment

apiVersion: apps/v1

metadata:

name: mydeployments

spec:

replicas: 5

selector: # tells the controller which pods to watch/belong to

matchLabels:

name: deployment

template:

metadata:

name: testpod1

labels:

name: deployment

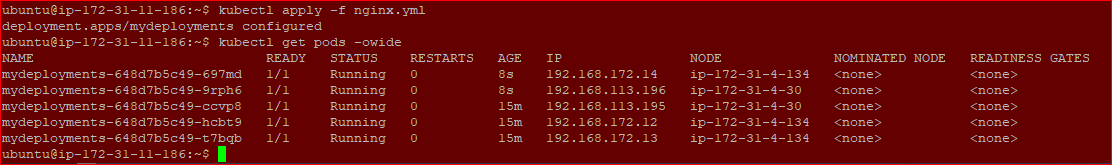
spec:

containers:

- name: c00

image: nginx

ports:

- containerPort: 80

**Module-9: Kubernetes Assignment – 4**

**You have been asked to:**

1. Use the previous deployment
2. Change the service type to clusterip

kind: Service # Defines to create Service type Object

apiVersion: v1

metadata:

name: demoservice

spec:

ports:

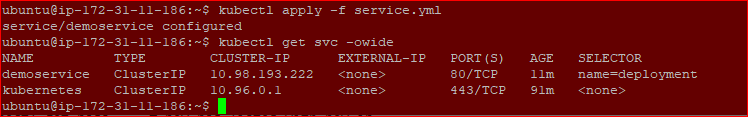
- port: 80 # Containers port exposed

targetPort: 80 # Pods port

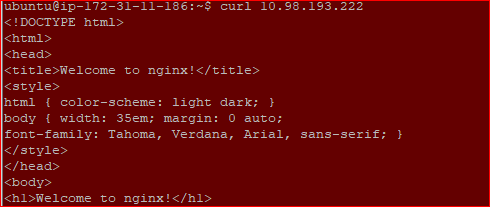
selector:

name: deployment # Apply this service to any pods which has the specific label

type: ClusterIP



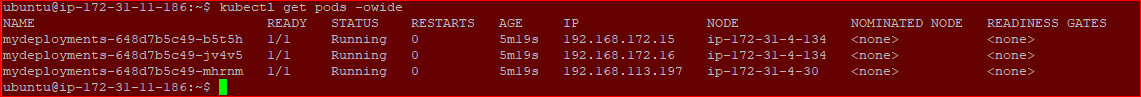
curl 10.98.193.222

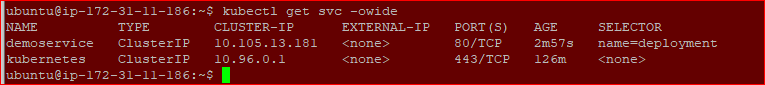


**Module-9: Kubernetes Assignment – 5**

**You have been asked to:**

1. Use the previous deployment
2. Deploy an nginx deployment of 3 replicas
3. Create an nginx service of type clusterip
4. Create an ingress service /apache to apache service /nginx to nginx service





**apiVersion: networking.k8s.io/v1**

**kind: Ingress**

**metadata:**

**name: minimal-ingress**

**annotations:**

**nginx.ingress.kubernetes.io/rewrite-target: /**

**spec:**

**rules:**

**- http:**

**paths:**

**- path: /apache**

**pathType: Prefix**

**backend:**

**service:**

**name: apache**

**port:**

**number: 80**

**- path: /nginx**

**pathType: Prefix**

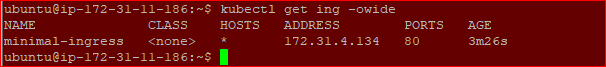
**backend:**

**service:**

**name: deployment**

**port:**

**number: 80**



**CASE STUDY – INTRODUCTION TO KUBERNETES**

You have just joined a startup Ventura Software as a Devops Lead Engineer. The company relies on a Monolithic Architecture for its product. Recently, the senior management was hired. The new CTO insists on having a Microservice Architecture. The Development Team, is working on breaking the monolith. Meanwhile, you have been asked to host a Test Application on kubernetes, to understand how it works.

Following things have to be implemented:

1. **Deploy an Apache2 deployment of 2 replicas**
2. **Sample code has been checked-in at the following Git-Hub repo:**

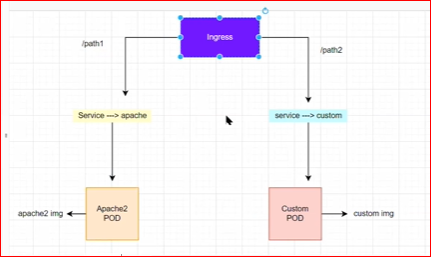
<https://github.com/hshar/website.git>

You have to containerize this code, and push it to Docker Hub. Once done, deploy it on Kubernetes with 2 replicas.

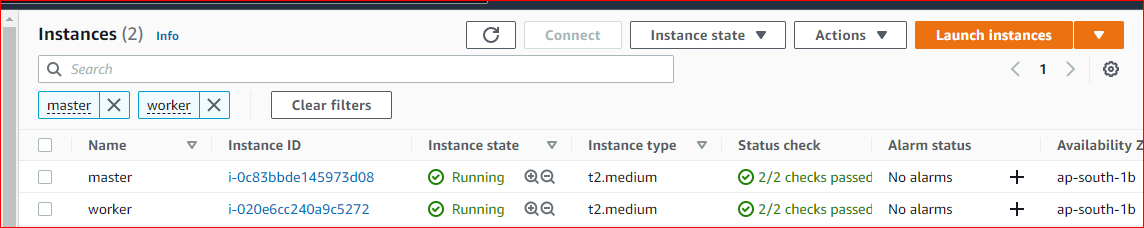
1. **Deploy Ingress with the following rules:**

!) \*/apache\* should point to the apache pods

!!) \*/custom\* should point to the GitHub application



First create two ec2 instance (worker and master node) as below to create kubernetes cluster setup.



Now connected both node using putty. And execute below command to create Kubernetes cluster.

On Master & Worker node(s)

sudo su

apt-get update

apt-get install docker.io -y

service docker restart

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |sudo apt-key add -

echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" >/etc/apt/sources.list.d/kubernetes.list

apt-get update

apt install kubeadm=1.21.0-00 kubectl=1.21.0-00 kubelet=1.21.0-00 -y

ON master NODE:

kubeadm init --pod-network-cidr=192.168.0.0/16

//copy the token and paste in the worker node.

exit # to become the normal user

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

# to get the join command(if lost previously)

kubeadm token create --print-join-command # get join command and execute on worker node

kubeadm join 172.31.33.238:6443 --token f30znn.jml5rn3v6rc0qeo1 --discovery-token-ca-cert-hash sha256:414b133e6fc8968a3cce729bea370b337ddbf30af33c6cecc983d3389aa7a971

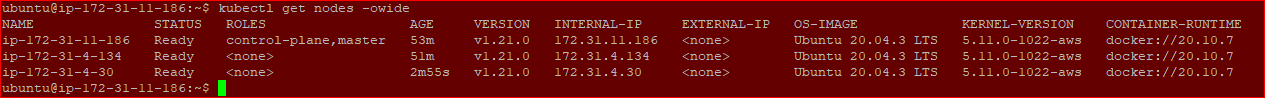
curl https://docs.projectcalico.org/manifests/calico.yaml -O

kubectl apply -f calico.yaml

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.49.0/deploy/static/provider/baremetal/deploy.yaml>

Master Node server:

kubectl get nodes -owide



apache.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: apache-deployment

labels:

app: apache

spec:

replicas: 2

selector:

matchLabels:

app: apache

template:

metadata:

labels:

app: apache

spec:

containers:

- name: apache

image: ubuntu/apache2

ports:

- containerPort: 80

kubectl create -f apache.yml

custom.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: custom-deployment

labels:

app: custom

spec:

replicas: 2

selector:

matchLabels:

app: custom

template:

metadata:

labels:

app: custom

spec:

containers:

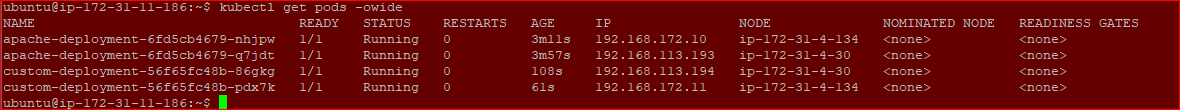
- name: custom

image: devopsintellipaat/kubernetese\_28\_img

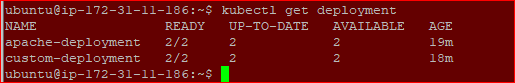
ports:

- containerPort: 80

Kubectl get pod -owide



Kubectl get deployment

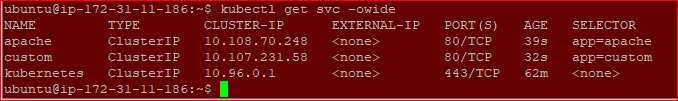


#Now create two services both deployment apache and custome as below

kubectl create service clusterip apache --tcp=80:80

kubectl create service clusterip custom --tcp=80:80

Kubectl get svc



Now create ingress for services.

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: minimal-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- http:

paths:

- path: /apache # service name

pathType: Prefix

backend:

service:

name: apache

port:

number: 80

- path: /custom # service name

pathType: Prefix

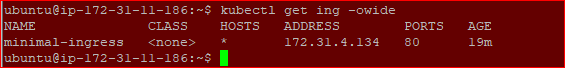
backend:

service:

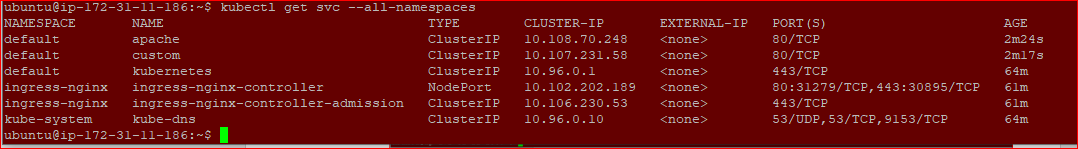
name: custom

port:

number: 80



Kubectl get svc –all-namespaces



URL: masterPublicIP:port/apache

URL: <http://3.110.174.11:31279/custom>

URL: <http://3.110.174.11:31279/apache>

