**Lab 1 : Kubernetes cluster**

**Step 1: Prerequisties Setup**

On node1

# yum install vim -y

# hostname master

# echo 127.0.0.1 master >> /*etc/hosts*

logout and login to check the hostname setup on the master machine

On node2

# yum install vim -y

# hostname worker1

# echo 127.0.0.1 worker1 >> /*etc/hosts*

logout and login to check the hostname setup on the worker1 machine

On node3

# yum install vim -y

# hostname worker2

# echo 127.0.0.1 worker2 >> /*etc/hosts*

logout and login to check the hostname setup on the worker2 machine

**Step 2: Install Docker CE**

Set up the repository and Install required packages.

# yum install yum-utils device-mapper-persistent-data lvm2

# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

Install Docker CE.

# yum update && yum install docker-ce-18.06.2.ce

# Restart Docker

systemctl daemon-reload

systemctl restart docker

systemctl enable docker

**Step 3: Install kubeadm**

You will install these packages on all of your machines:

* kubeadm: the command to bootstrap the cluster.
* kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.
* kubectl: the command line util to talk to your cluster.

# cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

# yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes

# systemctl enable --now kubelet

**Step 4: Enable Net packet fowarding**

Some users on RHEL/CentOS 7 have reported issues with traffic being routed incorrectly due to iptables being bypassed. You should ensure net.bridge.bridge-nf-call-iptables is set to 1 in your sysctl config, e.g.

# cat <<EOF > /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

# sysctl –system

**Step 5: Setup master node**

The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API server (which the kubectl CLI communicates with).

# kubeadm init --pod-network-cidr=192.16.0.0/16

To make kubectl work for your non-root user, run these commands, which are also part of the kubeadm init output:

# mkdir -p $HOME/.kube

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

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

**Step 6: Enable Network CNI**

We are using calico to setup the pod cluster network

# kubectl apply -f <https://docs.projectcalico.org/v3.8/manifests/calico.yaml>

**Step 7: Adding worker node**

Run the following command on the master node to generate command to add worker on the master node.

# kubeadm token create --print-join-command

copy the ouput

Now login into each worker node and paste the command.

Once done, login back to the master node and run the following command to check the cluster staus

# kubectl get nodes

You should be able to see all three nodes in ready state.

Congrats you have successfully deployed the kubernetes cluster using kubeadm.

**Lab 2: Pods lab**

**Step 1: Namespace**

Create a name space with the name new-ns with the manifest file

# vim ns.yml

apiVersion: v1

kind: Namespace

metadata:

name: new-ns

save and quit the file  
# kubectl create -f ns.yml  
# kubectl get ns

**Step 2: Pod**   
Create a pod name pod-data with the manifest file in the new-ns namespace with nginx image

# vim pod.yml

apiVersion: v1

kind: Pod

metadata:

name: pod1

namespace: new-ns

spec:

containers:

- name: cont1

image: nginx

save and quit the file  
# kubectl create -f pod.yml  
# kubectl get pods - n new-ns

**Step 3: Pod with label**  
Create a pod named pod2 with label env=test with redis image

# vim pod.yml

apiVersion: v1

kind: Pod

metadata:

name: pod2

labels:

env: test

spec:

containers:

- name: cont1

image: redis

save and quit the file  
# kubectl create -f pod2.yml  
# kubectl get pods -o wide   
# kubectl get pods –show-labels

**Lab 3: Replica’s**

**Step 1: RC**  
Setup a replica controller rc1 with 3 replicas and the pod selector to be env=prod

# vim rc.yml

apiVersion: v1

kind: ReplicationController

metadata:

name: rc1

spec:

replicas: 3

selector:

env: prod

template:

metadata:

labels:

env: prod

spec:

containers:

- name: cont1

image: nginx

save and quit the file

# kubectl create -f rc.yml

# kubectl get rc

# kubectl get pods -o wide

**Step 2: ReplicaSet**Create a Replica set rs1 with 3 replicas and two pod selection as env = test or env = prod

# vim rs.yml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: rs1

spec:

replicas: 3

selector:

matchExpressions:

- key: env

operator: In

values:

- prod

- test

template:

metadata:

labels:

env: prod

spec:

containers:

- name: cont1

image: nginx

save and quit the file

# kubectl create -f rs.yml

# kubectl get rs

# kubectl get pods

**Step 3: DaemonSet**Create the daemonset and see it’s operation

# vim ds.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: ds1

spec:

selector:

matchLabels:

abc: xyz

template:

metadata:

labels:

abc: xyz

spec:

containers:

- name: newcont

image: nginx

save and quit the file

# kubectl create -f ds.yml

# kubectl get daemonset

# kubectl get pods -o wide

**Lab 3: Services**

**Step 1: ClusterIP Service**Create a pod with label new=old and nginx image and expose it with clusterIP service on port 8080

# vim l3pod.yml

apiVersion: v1  
kind: Pod  
metadata:  
 name: l3pod  
 labels:  
 new: old  
spec:  
 containers:  
 - name: cont1  
 image: nginx

Save and quit the file

# kubectl create -f l3pod.yml  
# kubectl get pods -o wide

# vim cip.yml

apiVersion: v1

kind: Service

metadata:

name: cipservice

spec:

type: ClusterIP

selector:

new: old

ports:

- name: port1

port: 8080

TargetPort: 80

save and quit the file  
# kubectl create -f cip.yml

# kubectl get svc

# curl <serviceipaddress>

**Step 1: NodePort Service**Create a pod with label old=new and nginx image and expose it with clusterIP service on port 8080 and node port 32080

# vim l3pody.yml

apiVersion: v1  
kind: Pod  
metadata:  
 name: l3pody  
 labels:  
 old: new  
spec:  
 containers:  
 - name: cont1  
 image: nginx

Save and quit the file

# kubectl create -f l3pod.yml  
# kubectl get pods -o wide

# vim nip.yml

apiVersion: v1

kind: Service

metadata:

name: cipservice

spec:

type: NodePort

selector:

new: old

ports:

- name: port1

port: 8080

TargetPort: 80

nodePort: 32080

save and quit the file  
# kubectl create -f nip.yml

# kubectl get svc

Open your desktop brower and access [http://mastereip](http://mastereip/):32080

**Lab 4: Storages**

**Step 1: Host path Volume**Create a host path volume to mount /data directory into /vishal of the container

# vim l4pod.yml

apiVersion: v1   
kind: Pod

metadata:  
 name: pod3

spec:

containers:

- name: cont1

image: nginx

volumeMounts:

- name: vol1

mountPath: /vishal

volumes:

- name: vol1

hostPath:

path: /data

# kubectl create -f l4pod.yml  
# kubectl get pods

**Lab 5: Injecting Data**

**Step 1: Configmap**Create a config map with name qwe with key value pair to be v1=vishal  
Export this variable in the pod l5pod with environment variable name as username

# kubectl create configmap qwe --from-literal=v1=vishal

# kubectl get configmap

# vim l5pod.yml

apiVersion: v1

kind: Pod

metadata:

name: l5pod

spec:

containers:

- name: cont1

image: nginx

env:

- name: username

valueFrom:

configMapKeyRef:

name: qwe

key: v1

save and quit the file

# kubectl create -f l5pod.yml

# kubectl get pods -o wide

# kubectl exec -it pod l5pod

inside the pod # echo $username

the output should be vishal

**Step 2: Secret**Create a config map with name sec1 with key value pair to be pass=redhat  
Export this variable in the pod l5pod with environment variable name as password

# kubectl create secret generic sec1 --from-literal=pass=redhat

# kubectl get secret

# vim l5pod1.yml

apiVersion: v1

kind: Pod

metadata:

name: l5pod1

spec:

containers:

- name: cont1

image: nginx

env:

- name: username

valueFrom:

secretKeyRef:

name: sec1

key: pass

save and quit the file

# kubectl create -f l5pod.yml

# kubectl get pods -o wide

# kubectl exec -it pod l5pod

inside the pod # echo $password

the output should be redhat

**Lab 6: Deployment**

**Step 1: Deployment**

Deploy a deployment dp1 with redis image and 3 replicas.

# vim l6dp.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: dp1

spec:

replicas: 3

selector:

matchLabels:

app: redis

template:

metadata:

labels:

app: redis

spec:

containers:

- name: cont1

image: redis

save and quit the file

# kubectl create -f l6dp.yml

# kubectl get deployment

# kubectl get pods -o wide