**Networking**

# communication between two container in same pod

* Vi pod1.tml (create one YAML file )

( eg:- kind: Pod

apiVersion: v1

metadata:

name: testpod

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo hello-world; sleep 5 ; done"]

- name: c01

image: httpd

ports:

- containerPort: 80 )

* Kubectl apply -f pod1.yml
* Kubectl get pods
* Kubectl exec testpod -it -c c00 -- /bin/bash (to go inside the container )
* Apt update && apt install curl ( install curl package )
* Curl localhost:80 ( o/p:- it work, now we can communicate with container-001 via localhost )
* Exit ( exit from container )
* Kubectl delete -f pod1.yml ( delete the pod or yml file )

# communication between two container in different pods via IP

* Vi pod2.yml ( creating YAML file )

( eg:- kind: Pod

apiVersion: v1

metadata:

name: testpod

spec:

containers:

- name: c01

image: nginix

ports:

- containerPort: 80 )

* Vi pod3.yml ( create one more pod with YAML )

( eg:- kind: Pod

apiVersion: v1

metadata:

name: testpod3

spec:

containers:

- name: c02

image: httpd

ports:

- containerPort: 80 )

* Kubectl apply -f pod2.yml
* Kubectl apply -f pod3.yml
* Kubectl get pods
* Kubectl get pods -o wide ( to get wide detail about pods )
* Curl 172.12.0.3:80 ( we can communicate with nginix container via IP )
* Curl 172.4.0.3:80 ( we can communicate with httpd container via IP )

Service obejct commands

* Vi deployhttpd.yml

( eg:- kind: Deployment

apiVersion: apps/v1

metadata:

name: mydeployments

spec:

replicas: 1

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: httpd

ports:

- containerPort: 80 )

* Kubectl apply -f demployhttpd.yml
* Kubectl get pods
* Kubectl get pods -0 wide
* Curl 172.83.3.0:80
* Vi service.yml ( create yml file with service object )

( eg:- 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 # Specifies the service type i.e ClusterIP or NodePort )

* Kubectl apply -f service.yml
* Kubectl get svc ( you will static virtual ip for your cluster )
* Curl 10.0.0.0:80 ( now you can communicate pod with cluster ip )
* Kubectl get pods
* Kubectl delete pods “pod-name”
* Kubectl get pods ( you will get new pod with new ip )
* Curl 10.0.0.0 ( you can communicate with pod after getting new ip to pod )

Nodeport object

* Vi deployhttpd.yml

( eg:- kind: Deployment

apiVersion: apps/v1

metadata:

name: mydeployments

spec:

replicas: 1

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: httpd

ports:

- containerPort: 80 )

* Kubectl apply -f deployhttpd.yml
* Vi svc.yml

( eg:- 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 # Specifies the service type i.e ClusterIP or NodePort

* Kubectl apply -f svc.yml
* Kubectl get svc ( you will get nordport IP and port from 30000 to 32676 ranges )
* Kubectl describe svc demoservice ( to get information about pods )

( copy public dns from aws instance and paste in google eg:- dns:80 )

Volume object command (volume map with pod )

* Vi emptydir.yml

( eg:- apiVersion: v1

kind: Pod

metadata:

name: myvolemptydir

spec:

containers:

- name: c1

image: centos

command: ["/bin/bash", "-c", "sleep 15000"]

volumeMounts: # Mount definition inside the container

- name: xchange

mountPath: "/tmp/xchange"

- name: c2

image: centos

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

volumeMounts:

- name: xchange

mountPath: "/tmp/data"

volumes:

- name: xchange

emptyDir: {} )

* Kubectl apply -f emptydir.yml
* Kubectl get pods
* Kubectl exec myvolemptydir -c c1 -it -- /bin/bash ( to go inside container )
* Cd /tmp
* Ls ( you will get xchange volume you created )
* Cd xchange/
* Vi testfile ( add some data in file )
* Ls (o/p:- cloudage
* Pwd
* Exit ( exit from container )
* Kubectl exec myvolemptydir -c c2 -it -- /bin/bash ( to go inside container )
* Cd /tmp
* Ls ( you will get to see data directory )
* Cd data/
* Ls ( now you can see data of container-1 eg:-“cloudage” )

Hostpath volume object ( map volume with host or node )

* Vi hostpath.yml

( eg:- apiVersion: v1

kind: Pod

metadata:

name: myvolhostpath

spec:

containers:

- image: centos

name: testc

command: ["/bin/bash", "-c", "sleep 15000"]

volumeMounts:

- mountPath: /tmp/hostpath

name: testvolume

volumes:

- name: testvolume

hostPath:

path: /tmp/data )

* Kubectl apply -f hostpath.yml
* Kubectl get pods
* Cd /tmp
* Cd data/
* Kubectl get pods
* Kubectl exec myvolhostpath.yml – ls /tmp
* Echo “ hey its my first practical “ >mytest
* Pwd
* Kubectl exec myvolhostpath.yml – ls /tmp/hostpath ( you can see file you created before )