**Install hyperhit and minikube**

brew **update**

brew **install** hyperkit

brew **install** minikube

minikube **dashboard**

kubectl **version**

**Commands Resource Type/Kind Name**

**Get Pod Mynameispod**

**Kubectl Describe Namespace iamnamespace**

**Delete Replicaset iAmReplicaset**

**Replicacontroller iAMConfigmap**

**Deployments iAmDeployments**

**Configmap**

**Create | Start | Restart | CLI minikube cluster**

minikube **start** --vm-driver=hyperkit | virtualbox

minikube **start** --vm-driver=hyperkit --v=7 –alsologtostderr

minikube **start** 🡪 for Ububtu

minikube **start** –-vm-driver=hyperv

minikube **start** –driver=virtualbox 🡪 windows10 Pro

minikube **status**

minikube **CLI** 🡪 to startup /delete the cluster

kubectl **CLI** 🡪 to configure the minikube cluster

**Details of Pod or other components of cluster**

Minikube **explain** pods

Minikube **explain** replicationcontroller

Minikube **describe** pod <pod\_name> 🡪 will give details of given pod

Minikube **get** nodes | pod | service | replicaset | deployment

Minikube **get** cs 🡪 show health status for scheduler, control-manager and etcd

Minikube **config** get-contexts 🡪 shows namespaces

Cat .kube/config 🡪 file where all configurations were store

Kubectl **config** view 🡪 shows the server IP and details of cluster

Kubectl **ip** 🡪 give the minikube server IP

Minikube **ssh** 🡪 will put you in the cluster where you can check all the running containers

(by: docker ps, docker ps | grep nginx) use “exit” 🡪 to logout and go back to minikube env

**Delete cluster**

minikube **delete**

kubectl delete resource\_type resource\_name (Eg: kubectl delete pod pod\_name)

**List all commands | kubectl commands**

kubectl **get** nodes kubectl **delete** node <node\_name>

kubectl **get** pod kubectl **delete** pod <pod\_name>

kubectl **get** po –A kubectl get pod -o wide

🡪 will give details of pod(on which node pod is running)

kubectl **get** pod yaml | json 🡪 give o/p in json or yaml

kubectl **get** services

kubectl **get** replicationcontroller

**Create Pod without deployment**

Kubectl **run** <pod\_name> --image=<image\_name> --port:<port\_no>

eg: Kubectl run nginx --image=nginx:alpine --port:80

kubectl **run** <pod\_name> --restart=OnFailure –-image=<image\_name>

Start Pod on specific

Kubectl run second\_pod –-dry-run –-generate=run-pod/v1 –image=nginx:alpine

(this(--dry-run) will check what will be the output from the command. It will not create/change any request but give the output as what will happen. Means this will be on the client side, not on server side). Client side is just on your end, Server side is on server/container/pod side.

**Deployment**

Kubectl **create** deployment <deployment\_NAME> -–image=<image\_name>

(Eg. kubectl **create** deployment nginx-depl --image=nginx)

kubectl **get** deployment

kubectl **edit** deployment nginx-depl

kubectl **delete** deployment <deployment\_name>

kubectl **get** replicaset

kubectl **delete** deployment <deployment\_name>

(eg. kubectl delete deployment nginx-depl)

**Scaling Pod with Deployment**

kubectl **scale** deployment <deployment\_name> --replicas=<desired\_no\_of\_pod>

(Eg. kubectl scale deployment nginx-dep –replicas=4) 🡪 K8s will maintain 4 pods everytime

(Eg. kubectl scale deployment nginx-dep –replicas=2) 🡪 K8s will reduce pods to maintain 2 pods everytime)

**Debugging**

kubectl logs <pod-name>

kubectl exec -it <pod-name> -- bin/bash 🡪 get interactive terminal

**create mongo deployment**

kubectl create deployment mongo-depl --image=mongo

kubectl logs mongo-depl-{pod-name}

kubectl describe pod mongo-depl-{pod-name}

**Create or Edit Config file**

vim nginx-deployment.yaml

kubectl apply –f deployment\_file.yaml

kubectl apply -f nginx-deployment.yaml

kubectl get pod

kubectl get deployment

kubectl delete –f <deployment\_file.yaml>

**Delete with config**

kubectl delete -f nginx-deployment.yaml

#Metrics

kubectl top The kubectl top command returns current CPU and memory usage for a cluster’s pods or nodes, or for a particular pod or node if specified.

**Expose the image on browser**

Kubectl expose deployment nginx --type=NodePort 🡪 this will expose the image with type Node Port.

Kubectl expose pod nginx –type=NodePort 🡪 this will expose the pod with type Node Port.

(check if the service is started or not by: kubectl get services)

Check Service On browser: ServerIP:portNo

Server IP 🡺 kubectl config view (eg: Server: https://192.168.99.100)

Port No 🡺 kubectl get services (80:32741)

Eg. On Browser: <https://192.168.99.100:32741> 🡪 this will give the service on browser(firefox, google etc)

**Kubectl Label:** you can attach many labels to a single pod. (it is same as tags in AWS which we give for instances or diff. components.)

(kubectl label resource\_type resource\_name label/name/key=label\_name/value\_name)

Kubectl **label** pod pod\_name env=testing

(to check changes🡪 kubectl **describe** pod pod\_name | less)

(to check changes directly 🡪 kubectl **get** pods **–-show-labels**)

If you want to **edit** the value/label\_name then:

Kubectl **label** –-overwrite pod pod\_name env=production

If you want to **delete** the label: key/label\_name**-**

Kubectl **label** pod pod\_name env**-**

If you want to **attach** **label to many/all pods**:

(kubectl label pod –-all key/label\_name=value/label\_value)

Kubectl **label** pods **–-all** status=xyz

**Create YAML file:** yaml can be used to create a pod or different resource.

Four important parts in Pod yaml file: APIVersion, Kind, Metadata and Spec

vim firstpod.yml

(check version format to hit the version by: kubectl describe pod)

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

env:prod

spec:

containers:

* name: container\_name

image: nginx:alpine

Now tell k8s cluster to run this yml file to create a pod:

Kubectl **create** **–f** firstpod.yml **OR**

Kubectl **apply –f** firstpod.yml

Similarly you can write a yml file to create a replicas etc.

You can check the error before running the file by dry run:

kubectl **create –f** firstpod.yml --dry-run

(use online yml check to verify)

(verify by: kubectl get pods/resource type)

**Check for the YML formate information:**

Kubectl explain pod **–-recursive** | less

**Another way to create a pod:**

1.take an output from the command and 2.redirect it to yaml.

1.Kubectl **run** second\_pod **–-dry-run** –-generate=run-pod/v1 –image=nginx:alpine **–o** **yaml**

(-o is for output in yaml)

2.Kubectl **run** second\_pod **–-dry-run** –-generate=run-pod/v1 –image=nginx:alpine **–o yaml >** mysecond\_pod.yml

(verify by : **vim** mysecond\_pod.yml)

**Imperative vs Declarative or Difference between Create and Apply**

**UPDATE Pods:**

First create a pod with labels by:

Kubectl create –f firstpod.yml 🡪 will create a pod from yml file

Kubectl get pods –o wide 🡪 to check the no of pods

Kubectl get pods –-show-labels 🡪will show the pods with labels

Kubectl **edit** pod myfirstpod

🡪this will open the yml file then edit the file and the pod will also been updated

Kubectl get pods --show-labels

K8s can be configured in **3** types:

**Imperative command, Declarative object Configuration and Imperative object configuration**

**Imperative command:** if you are editing any k8s resource without using a yml file

**Imperative Object Configuration**: configuring any of the resources by ownself or manually (adding labels or metadata) . it helps in editing any resource/container also Git

Eg: kubectl **create** **–f** firstpod.yml

**Declarative Object Configuration**: getting the resource all perfectly configured from someone. It helps in automation and also Git

Eg: kubectl **apply** **–f** firstpod.yml

If you edit the yml file (Eg: by adding the label) to the existing pod k8s will directly update it.

If the pod is not created “apply” will first create a pod and then configure as per yml.

**DELETE Pod/Resources through YAML File:**

Kubectl **delete** –f firstpod.yml 🡪 this will delete the resource created inside yml file

**Kubectl apply:**

Kubectl **apply –f** firstpod.yml **--dry-run** 🡪this will give output as per configuration

Kubectl **apply –f** firstpod.yml **--server-dry-run** 🡪 this will create a pod if does not exists and give output. If pod already exists it will give output as “….unchanged”. this is only available in apply not in create.

**Check the difference**: this will show at which configuration the pod will create

Kubectl **diff –f** firstpod.yml

**13. Set Environment Variable in Pod’s Container:**

Here we will set an env. Variable on the container of a Pod, so we will create a yml file.

vim firstpod.yml 🡪 create a yml file and add env to file.

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

* name: firstcontainer

image: nginx:custom

**env:**

**- name: myname**

**value: Gaurav**

**- name: city**

**value: Jaipur**

Now run/apply the yml file to create a pod (make sure to delete an existing same named pod)

#Kubectl apply –f firstpod.yml

Then you can check the container/pod created in which node by:

#Kubectl get pods –o wide

Go/Access on the node and check for docker enviournment by

#docker container exec –it container\_id env

**14. Run a Command in Pod’s Container:**

In 13. We have to go to node and then have to check for the docker container for the changes. But now we will check the changes directly on the master node. This is also used for **debugging.**

Go to yml file and check the container name.

**#cat** firstpod.yml

To check the pod’s env

**#**kubectl get pods

**#**kubectl **exec** myfirstpod **env**

This is good if the pod have just one container in it. If the pod have multiple containers, then

**#**kubectl **exec** myfirstpod **–c** firstcontainer **env**

**To directly access the pod**

**#**kubectl **exec** myfirstpod **–it bash 🡪** this if pod have only 1 container

(same in docker: #docker container exec –it container\_name/id bash)

**#**kubectl **exec** myfirstpod **–c** firstcontainer **–it bash**

🡪 this to access specific container in a pod

**#**kubectl **exec** myfirstpod **–c** firstcontainer **–it ls /**

**🡪** this will list the contents of container directly from master

**15. Set Commands in Pod’s Container:**

**Eg: #docker container ls** shows the details about the container and the “command” run inside them. To change this in docker we can do this.

#docker container run –itd ubuntu sleep 30

This will create a container and put a sleep 30 in the command. Sleep 30 will disappear after 30 sec

Similarly in K8s:

Edit the yml file adding “args:” to create a pod.

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

* name: firstcontainer

image: nginx:custom

**env:**

**- name: myname**

**value: Gaurav**

**- name: city**

**value: Jaipur**

**args: [“sleep”, “50”]**

After creating a yml file to create a pod (delete same named pod if exists by: #kubectl delete –f firstpod.yml )

Create a pod with yml:

**#**kubectl **apply** **–f** firstpod.yml

Check on which node the pod is created

#kubectl get pods –o wide

Go to that node and check if the pod disappear after 50sec.

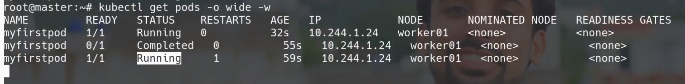
node1#kubectl get pods

#docker container ls

🡪 will show all containers and there command

On master check the container on watch to monitor the state

**#**kubectl **get** pods **–o wide –w**



Here when the container delets from the pod the pod also gets deleted.

But when the k8s gets/finds that the pod is deleted k8s recreate a pod to maintain and again the pod with the container is created.

Create Pod with container 🡪 after 50sec container delete 🡪 hence pod delete 🡪 but k8s has to maintain the pod state so it recreate the pod automatically 🡪

**16. Create a Multiple Containers in a pod with shared Port or IP**

Here we will check the multiple containers in a same pod have a shared ip.

First creating a yml file to create a pod with multiple containers.

#vim firstpod.yml

Remove the same named pod if exists by:

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

**- name: firstcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “3600”]

**- name: secondcontainer**

**Image: nginx:alpine**

#kubectl delete –f firstpod.yml

Apply yml to create a pod

#kubectl apply –f firstpod.yml

Then wait to get ready both containers and the pod

#kubectl get pods –o wide

(use this command to check the yml file for names of container:

#grep name firstpod.yml)

Checking/Accessing the first container

#kubectl exec myfirstpod –c firstcontainer –it bash

#netstat –nltp 🡪 will show the open ports for the container

# netcat –l –p 8000 🡪 will open 8000 port for traffic

Checking/Accessing the second container

#kubectl exec myfirstpod –c secondcontainer –it bash

#netstat –nltp 🡪you can check the open ports for the Pod. It will also show the open ports from different containers.

Verify by **Telnet:**

On second container:

**#telnet localhost 8000**

Trying 127…

Connected….

..

Hi 🡪 write hi this will send message Hi to firstcontainer

On first container:

Hi

Hi Davesh 🡪 write message for second container

On second container

Hi

Hi Davesh 🡪 message from first container

Verify by **IP**

On first container: On Second Container

#ifconfig #ifconfig

Both will have same ip address.

This is used in a Pod where there is a web application container and Data base container to communicate in between them.

**17. Init Container in Pods:**

We have to create a yml file to create a pod with **init container.**

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

**- name: firstcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “3600”]

**- name: secondcontainer**

**Image: nginx:alpine**

initcontainers:

**- name: initcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “30”]

Before running the yml make sure there is no same named pod created earlier. To delete pod

#kubectl delete –f firstpod.yml

Now run this yml file:

#kubectl apply –f firstpod.yml

Check pods:

#kubectl get pods –o wide

🡪 wil only show all continers without init.

**Initcontainer:**

When the pod is created, initially init container is created with the specification and then terminated.

After the termination of the initcontainer the normal containers is created which will continue working.

After pod creation

First init container is created

then init container terminated

after termination normal containers is created and continued to work normally.

Testing by creating 2 normal and 2 init containers

Remove existing same named pod if exists:

#kubectl delete –f firstpod.yml

Run/Apply yml file

#kubectl apply –f firstpod.yml

Check:

#kubectl get pods –w

Here the first init container init1container will be executed then second init2container after (STATUS Init:0/2) both is terminated the normal container will be created with ready status 2/2.

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

**- name: firstcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “3600”]

**- name: secondcontainer**

**Image: nginx:alpine**

initcontainers:

**- name: init1container**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “30”]

**- name: init2container**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

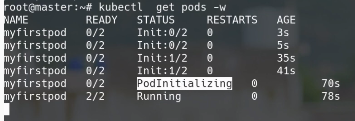
- name: city

value: Jaipur

args: [“sleep”, “30”]

If anyone of the initcontainer does not terminate the pod will continue to recreate itself again and again.

Process:



Initially both initcontainers are created andterminated after every 30sec then normal containers are created.

USE CASE:

When the pod is created it has to be registered somewhere.

OR

Clone from git repo before stating the normal containers.

**18. Cluster IP**

A service in k8s can be accessed **within/inside** the cluster.

First create a Pod from yml (firstpod.yml) and expose the port

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

**- name: firstcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “3600”]

Create a Pod

#kubectl apply –f firstpod.yml

Now if the container has a image of nginx and has to expose on port

Here all the traffic comes to port 8000(in which port we have to listen in this cluster) is redirected to port 80 traffic

Pod- Port 80 8000

Pod – Port 85 CLUSTER

(Master, Node1, Node2)

Pod- Port 45

Expose the port

**#**kubectl **expose pod** firstpod **--port=8000 --target-port=80 –name myfirstservice**

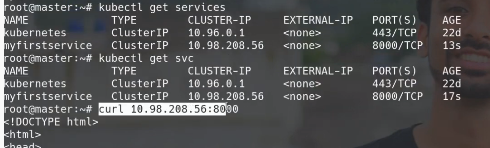
**Check:**

#kubectl **get** service

#kubectl **get** services

#kubectl **get** svc

Redirect traffic to pod:

#curl cluster\_ip\_of\_service:8080 

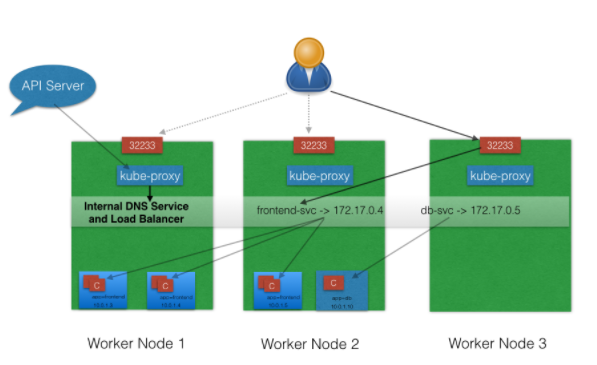
**#curl** 10.98.208.56:8080

**This will redirect all the traffic coming to the cluster from port 8000 to🡪 the Pod with port 80 (here firstpod)**

**19. Node Port:**

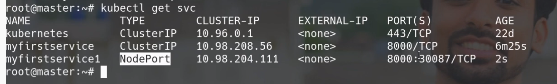
If you want to access the service outside the cluster or on the browser. Simply you have to add –type=NodePort in the command.

**#**kubectl **expose** **pod** thirdpod **–-type=NodePod** --port=8000 –-target-port=80 –-name myfirstservive1



Check service:

#kubectl get services



Here the service is now targeted to port no 30087

To access/browse the service on the browser:

Ipaddress\_of\_cluster:redirected port

Eg: 192.168.25.43:30087

**Expose the image on browser**

Kubectl expose deployment nginx --type=NodePort 🡪 this will expose the image with type Node Port.

Kubectl expose pod nginx –type=NodePort 🡪 this will expose the pod with type Node Port.

(check if the service is started or not by: kubectl get services)

Check Service On browser: ServerIP:portNo

Server IP 🡺 kubectl config view (eg: Server: https://192.168.99.100)

Port No 🡺 kubectl get services (80:32741)

Eg. On Browser: <https://192.168.99.100:32741> 🡪 this will give the service on browser(firefox, google etc)

**20. How Service Works. How traffic is redirected to specific Pods within cluster**

First write a yml file foe the service.

#vim firstservice.yml

apiVersion: v1

kind: Service

metadata:

name: myfirstpod

labels:

newlabl2: Saurav

newlabl3: aaurav

spec:

containers:

**- name: firstcontainer**

**image: nginx:alpine**

env:

- name: myname

value: Gaurav

- name: city

value: Jaipur

args: [“sleep”, “3600”]