<https://www.katacoda.com/tfogo/scenarios/k8s>

<https://katacoda.com/kubernetes>

<https://www.katacoda.com/tfogo/scenarios/k8s>

Section 1- Kubernetes Concept and Local set up

Section 2- Kubernetes in action

Section 3: Kubernetes Volumes

**Section 4 - Deploy Microservice to kubernetes**

**Section 5- Eclipse Jkube Quick start**

Section 1- Kubernetes Concept and Local set up

**Introduction :**

**Objective :** what is **container orchestration** and why kubernetes is such a popular container orchestration tool

Running container in single host is ok for developer testing but not for production

These container where m/s application run should be **fault tolerant** meaning if one container goes down there should be backup for it

Or there should be 100 backup depending on the application we are working with so that user will not see any fault

These should be able to scale up and down **on demand** depending on the user request comes

**performance** These container should perform well while communicating with each other and auto discovery should be possible to work with each Other like productservice auto discover to couponservice.

**Public acces**s These container in the cluster should be available to outside the world so that if some other application want to access our m/s they should Be able to access it and vice versa.

We should be able to **update** to the application automatically running on the container and also we should be able to **rollback**

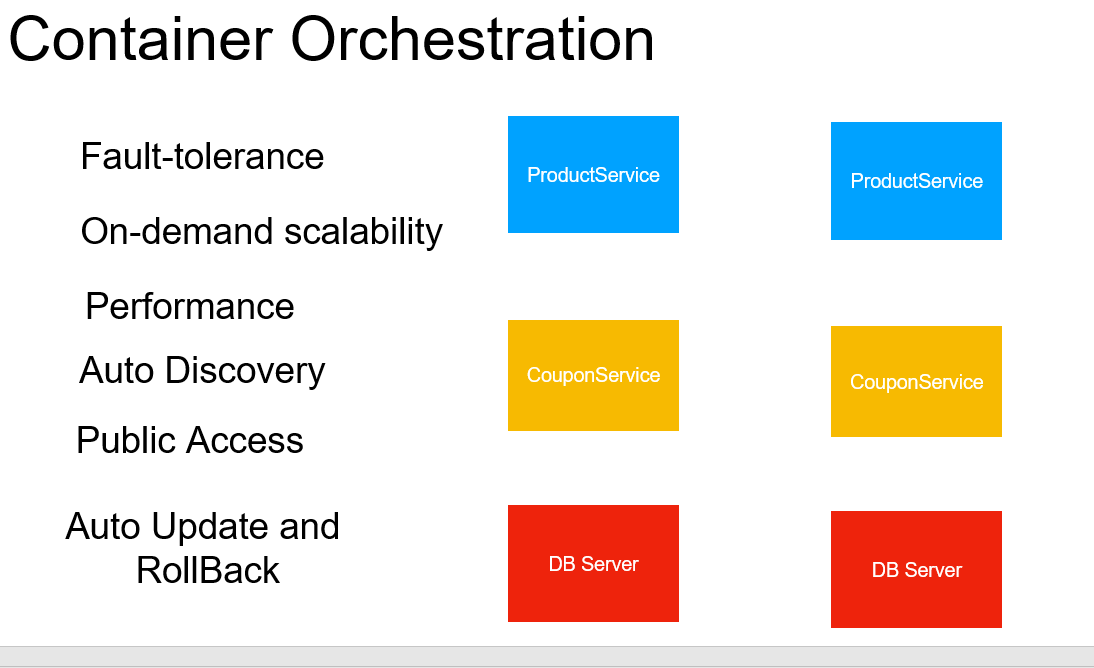
To particular version in the cluster without any downtime or with minimal downtime.

**The process of providing all these non functional requirement right from the creation of cluster is called container orchestration**

**In container orchestration** all our containers are grouped to form a cluster where the deployment and management of these

Container is completely automated while all the requirement on lhs met

**container orchestration is a process of forming a cluster of containers that work with each other with all non functional requirement**



**Kubernetes :---** is popular container orchestration tool that give us all non functional requirement out of the box.

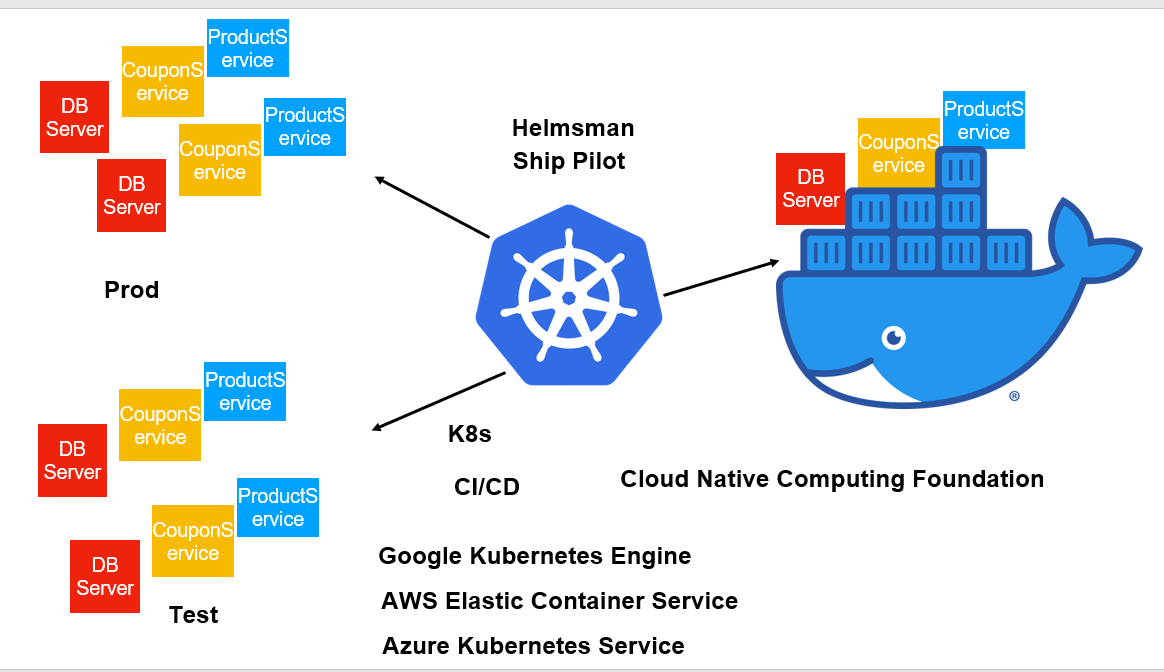
It is most popular container orchestration tool which help deploy and connect container across multiple host providing the services

Self discovery,scaling without any downtime.

Kubernetes is also referred as **k8s** because there are 8 alphabet b/e k and s

**Note:**

**All the cloud providers like aws** has inbuilt support of kubernetes. We can create our own cluster also on these cloud



It support docker not only docker but several other.

Kubernetes takes our container and with few step it will create a cluster for us

**Kubernetes Object Model**

**K8s uses object model to represent** different persistent entity in the cluster.

They are :-

Pods

ReplicaSet

Deployment

Namespace

Service

We create these by using spec in yaml section

**Pods are managed through replicaset**

**Pods** are the smallest, most basic deployable objects in **Kubernetes**.

A **Pod** represents a single instance of a running process in your cluster.

Pod logical group of similar container that are related with each other

**All are object**



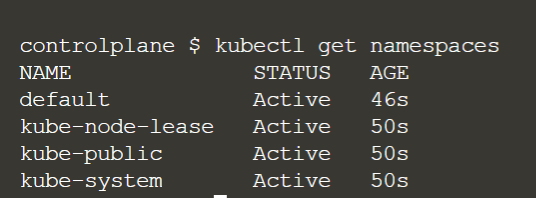
**Deployment take care of replicas and replicas take care of pods**

**Namespace :--**

it allow to group a set of application with a particular namespace.

It is like pkg name in java.

So we assign a unique namespace to group a set of application or project in the cluster.



**Default** is the namespaces which k8s assigned to all the objects( pods, replicaset) that we create if we do not specify any other namespace.

This is the namespace under which all our object are created.

**Kube-node-lease** :- k8s store all health related data

**Kube-system :** k8s system level namespaces . It uses for storing k8s specific things

summarize

Everything in k8s cluster will happen through object.

The lowest object is pod . It is logical group of container.

Replicaset help us monitor and manage these pods

To manage these two we use deployment

**Architecture**

All the machines in k8s cluster are referred **as Node**

**Master Node :-** manage the entire cluster

**Worker Node :-** responsible for launching the pods , creating the container and so on

We can have single master node in dev environment and we can have **multiple master node** in stage or prod environment

**Api server** :---responsible for to take yaml file and then create all the resources in worker node.

**Scheduler :** who will schedule the pods on the worker node. It has all the info about worker node and pods.

**Control Manager :** runs in the background and once the cluster is up then control manager responsible for to ensure that cluster is in desired state

**Api server :** maintain the current state of cluster in distributed storage service called etcd

**Etcd :** is simply a name value storage. All the objects like pods , container are stored in name value pair in etcd.

And in multiple master node we have etcd file so even if one master node goes down we have current state of cluster in etcd file and other

Master node then come up.

The imp component within the worker node is :- **kubelet**

**Kubelet :**

**the master node will get work done by worker node and each worker node w**ill have kubelet set up.

Master node is one which make api call to the kubelet to get required number of pods, container and all those setup

From command line utility kubectl will interact with api server and api server will interact with kubelet and get info abt all the

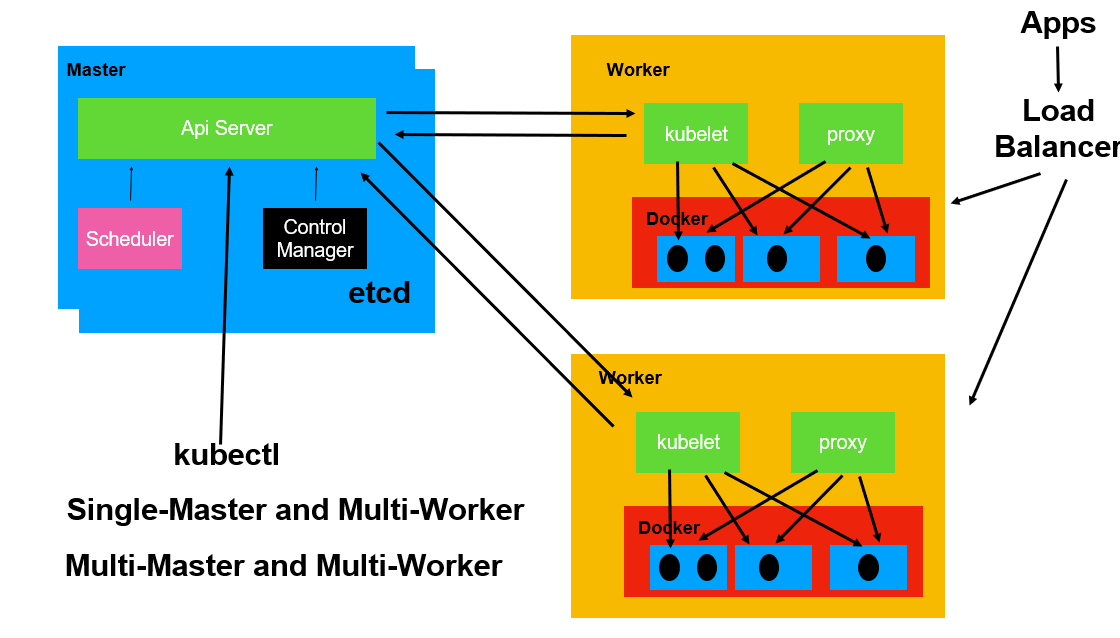
Pods and other resources

**Proxy :** this is the network proxy and load balancer so when application from outside want to communicate with the container

That are running here then they have to go through proxy and it will balanced the load across the services inside this worker node

So there is proxy in each worker node and responsible for load balancing the call that are coming to the services.

Load balancer is external load balancer



We interact with api server through yaml file we generate using kubectl command

**Summarize**

Cluster is made up of multiple machine each machine is called a **Node**.

The machine that is responsible managing the cluster is called **master node**

The machine where bulk of work done like creating the node , running the container is called **worker node**.

We have multiple worker node managed by master node.

Through **api serve**r we communicate to rest of cluster using kubectl. It make restful api call to the server which maintain cluster for us.

**Scheduler** is responsible for scheduling the pods.

**Control manages** runs in background and responsible for required number of pods is maintained. If it goes down it will launch another pod

**Etcd** distributed storage service which k8s uses. It will store all info of pods.

Api server will talk with **kubelet** to launch the pod and container and so on

**Proxy** is responsible for distributing incoming call across services in our worker node

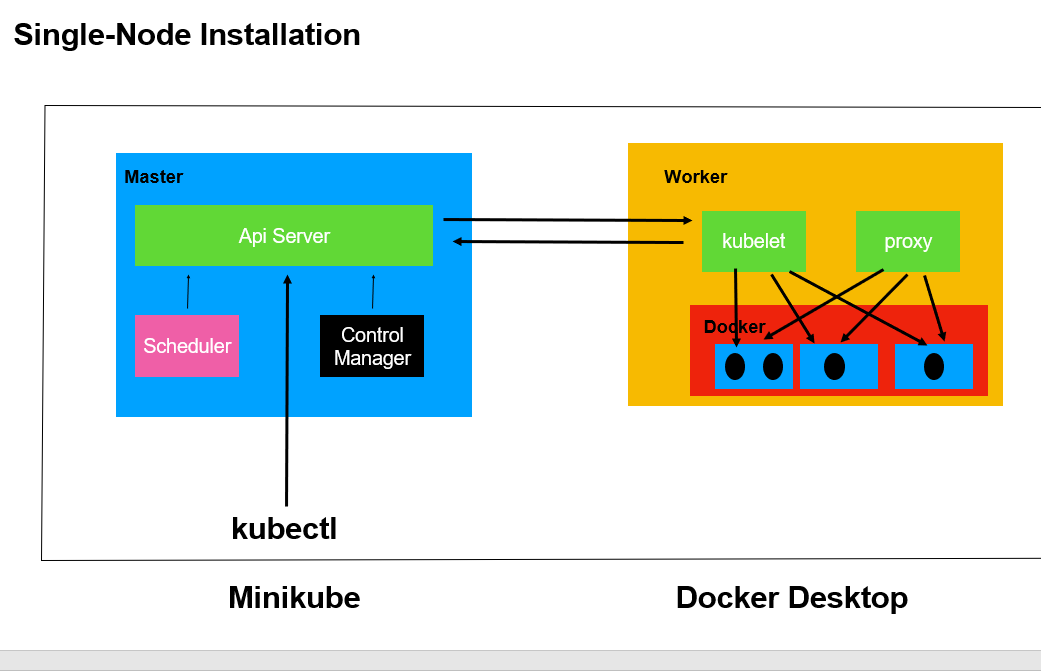
**Installation types**

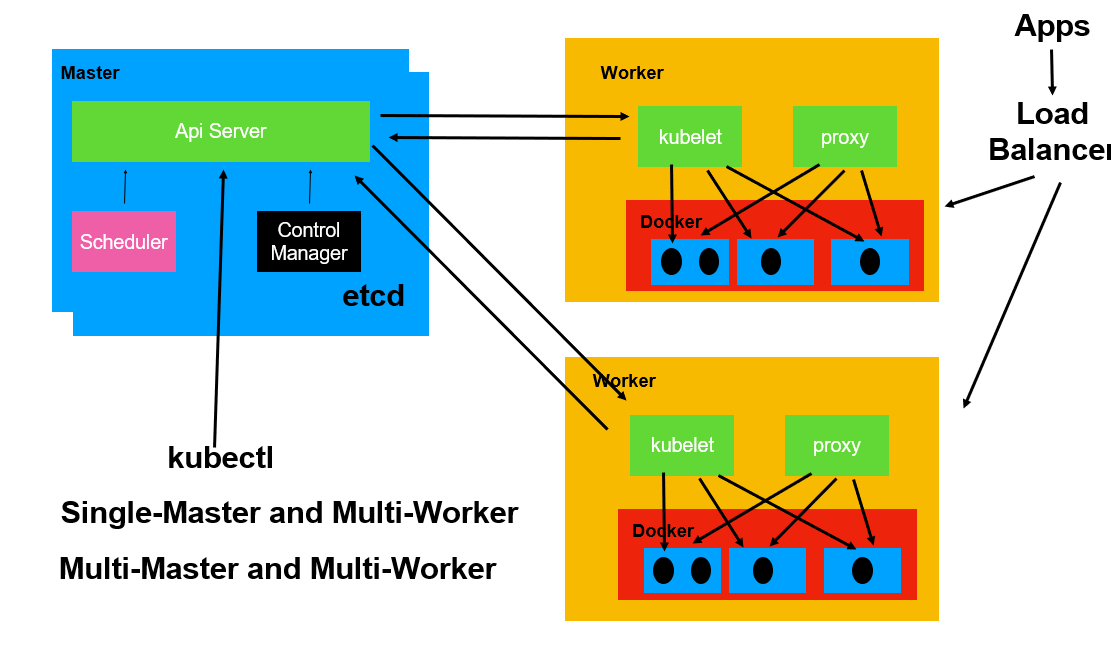
**Single node installation :**

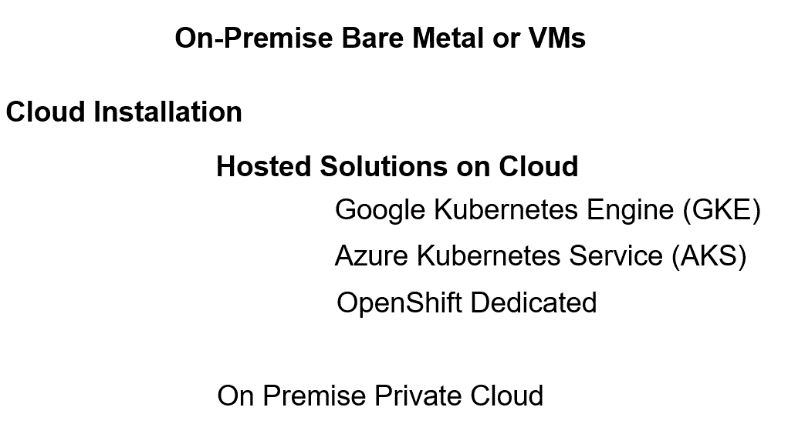
Good for dev testing bit not for prod environment

**Single master and multi worker**

**Multi master and multi worker**







**Window only install minikube**

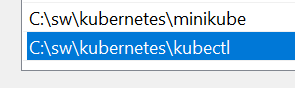
<https://github.com/kubernetes/minikube/releases>

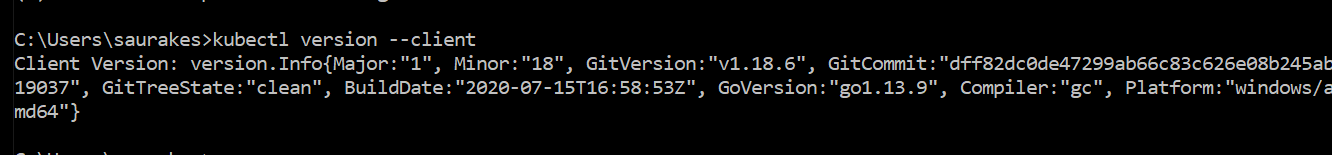
Minikube start :-- will start single node cluster for us on windows using the virtual machine software that’s is already there hyper v

**Kubectl :-**

command line interface using which we can manage kubernetes cluster resources and application

So we can use it to deploy our application on the cluster and manage kubernetes resources as well.





**Will be using kubectl to communicate with minikube cluster**

Kubectl config configuration

Section 2 :- Kubernetes in action

**POD**

The smallest and most imp object in k8s is **pod**

**A pod is** abstract object logically group a set of related container with all the resources that are required for container to work.

These pod will have all the resources for these container to work. We can have multiple pod

Pods is a collection of container that can run on host

**Example:-**

n/w ,security, configuration, volumes all these available to container through POD.

It is good practice to use **one container per pod**.

We can create pod using **kubectl command or** by creating pod specific **yaml file.**

**In real time** we will make pod through **deployment** which can create any no of pods and also replicas

**Imp points :---**

**we expose out these pods**(like couponservice in one pod and productservice in another pod)

to other pods **through services**

Also application outside the cluster can also access pods and container running within the pod through kubernetes object

Called **service**

**Note :**

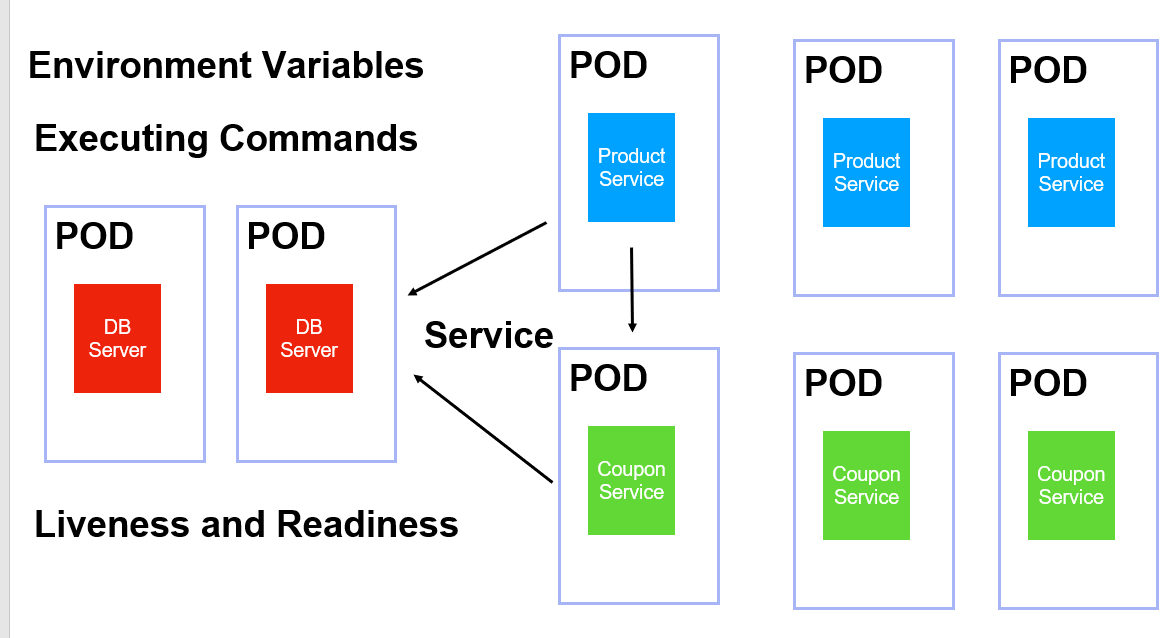
**Everything we done while launching docker container earlier can be done with pods.**

**Liveness and readiness :----**

is the container started, is the pod ready

All these check can be added in a file.

**We will learn** how to create pod using command line, using yaml file then using deployment



**Create first POD**

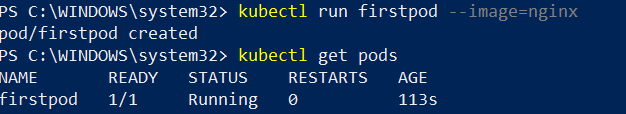
**create k8s pod using nginx image**

**1-Create a pod :** kubectl run firstpod --image=nginx

**2-Get pod :** kubectl get pods

It launch a container and inside that it using nginx image.

0 restart :- means there is no issue



**3 Describe the details of pod :** Kubectl describe pod firstpod

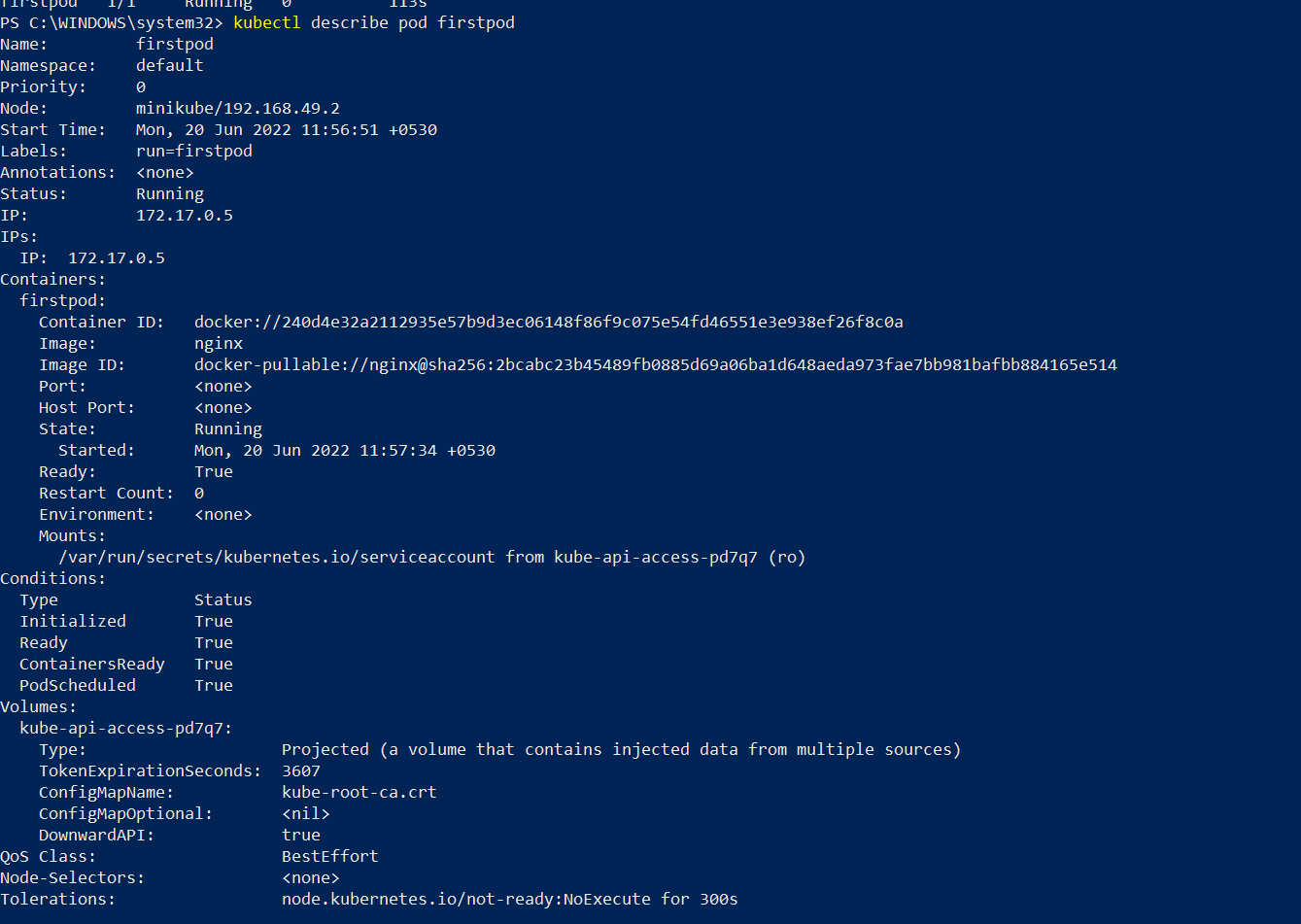
Minikube :---one node (here we have only one node)



**Scheduler** is responsible for assigning a pod to node **(Minikube).**

**Scheduler** is scheduling the pod on worker node in this case minikube a single node cluster

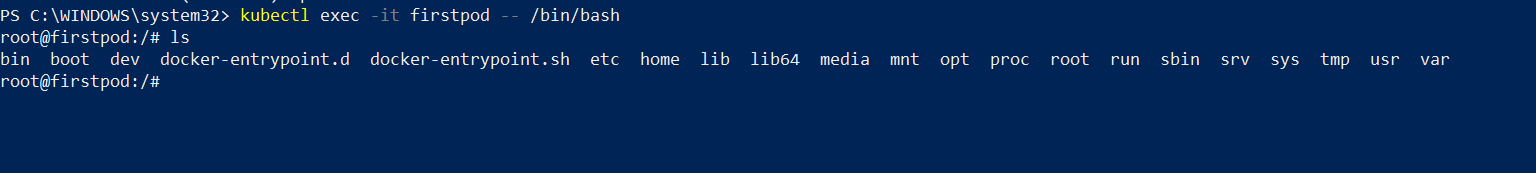
**Kubelet** on the worker node in this case **minikube** responsible for pulling the image from the **dockerhub** and then launching the container in the pod



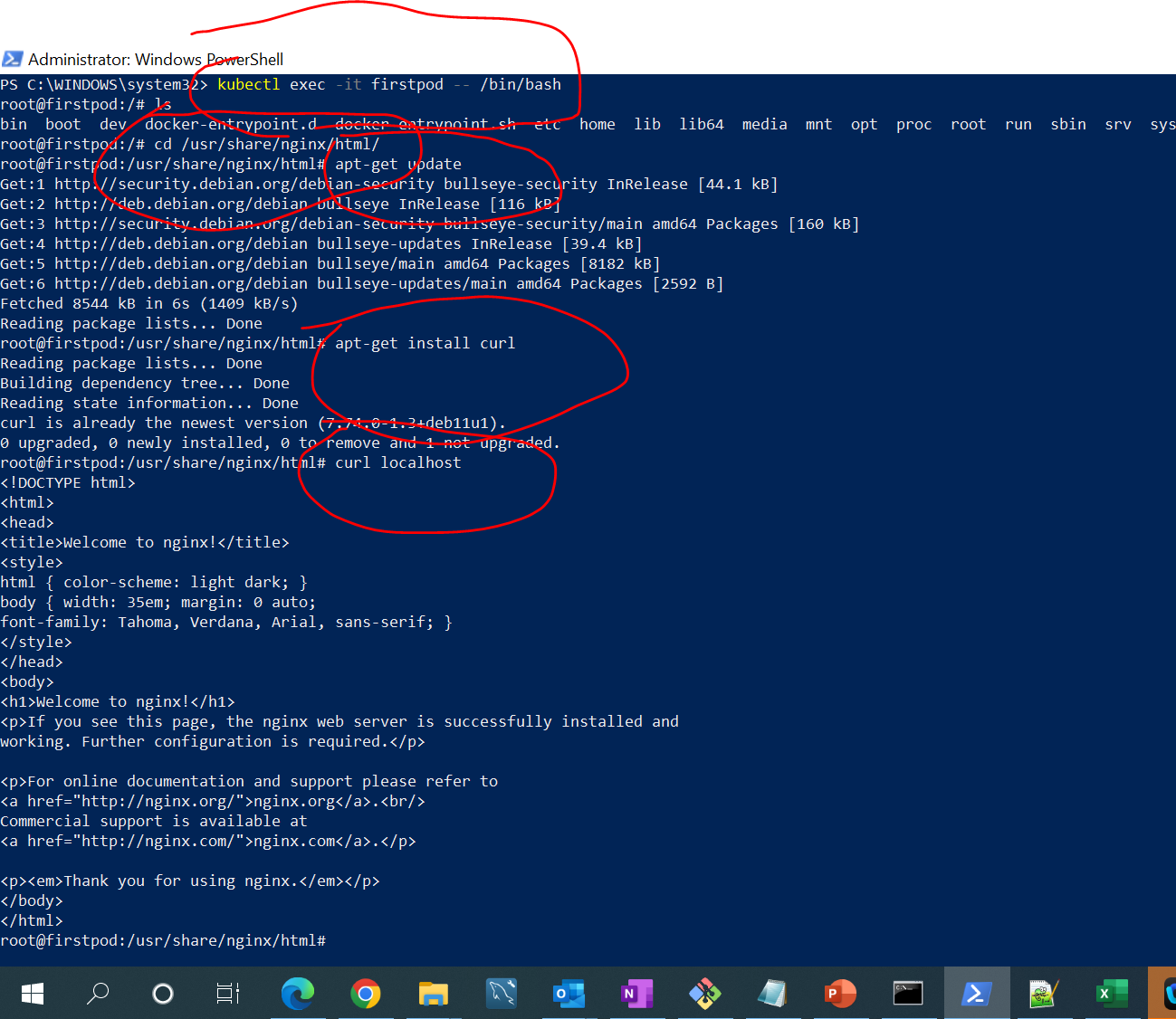
1. **Will go inside pod just like how will go inside docker**

**Kubectl exec -it firstpod -- /bin/bash**

Now we are inside pod that we have created



Note: nginx is already up and running on port 80

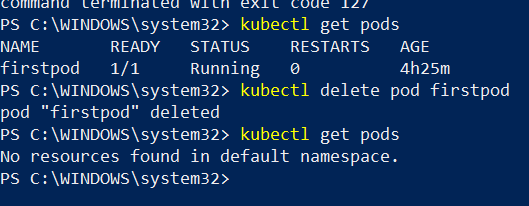


Curl localhost means curl localhost:80

Successfully launch firstpod using nginx image to went into the pod by using kubctl command and then able to install curl

On that pod then curl localhost which is connected to nginx that is running on this pod

5:- **delete pod** : kubectl delete pod firstpod





**Kubectl get pod firstpod -o yaml** ------------ it will show file into yaml which k8s created behind the scene while creating pod

**Create pod using yaml**

Very first line of any k8s configuration file is

**Kind: Pod :-**

we specify what type of resource we are going to deal with in this yaml file. In this pod we are going to create, delete,

Update pod

**ApiVersion: v1**

All the element we will use belong to this version

**Metadata:**

Where we provide data of the resource **(here kind: pod in this case pod is our resource)**

**Spec:**

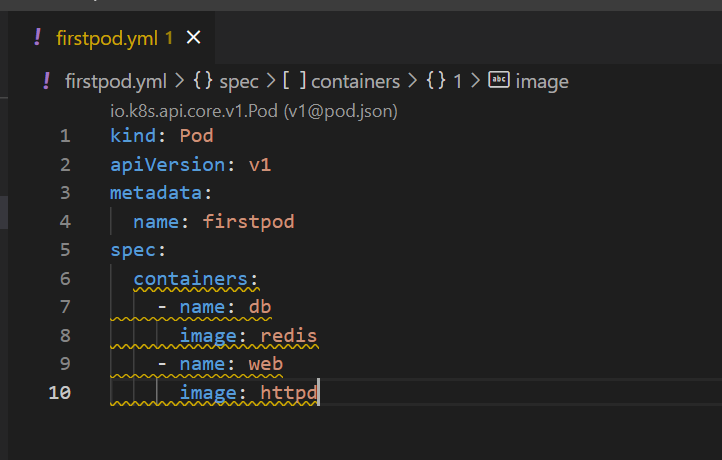
Body of any resource , how this resource should be create

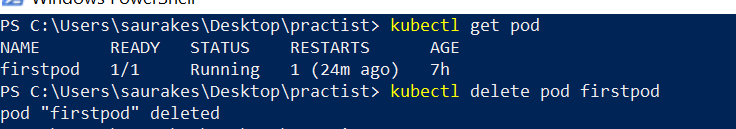
Int case of pod it can have one or more container

**In this example**

first container name is db and second container name is web.

We have created two container



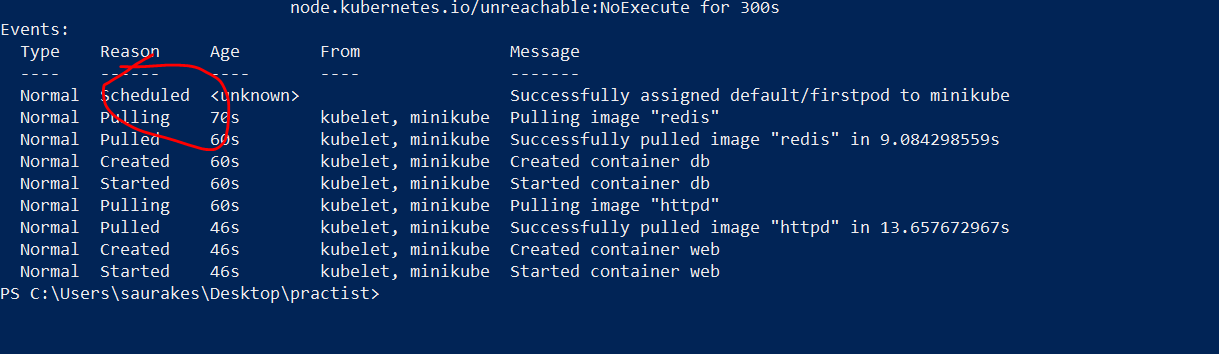


**Kubectl create -f firstpod.yaml**

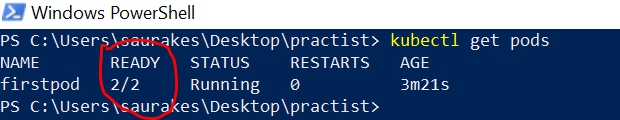
* Using this it will create pod and it will launch container inside us



Kubectl describe pod firstpod



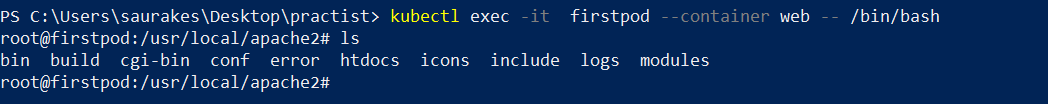
**Two container is up**



**Lets go into specific container of pod (db-reddis container)**

**kubectl exec -it firstpod --container db -- /bin/bash**

Inside web container



Now we have two container running inside the pod

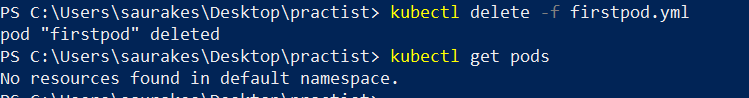
**Note :-**

**When we delete a pod using replication controller** automatically it will bring up another pod

Since we are directly deleting the pod so it not create pod for us

**Delete pod :** kubectl delete -f firstpod.yml

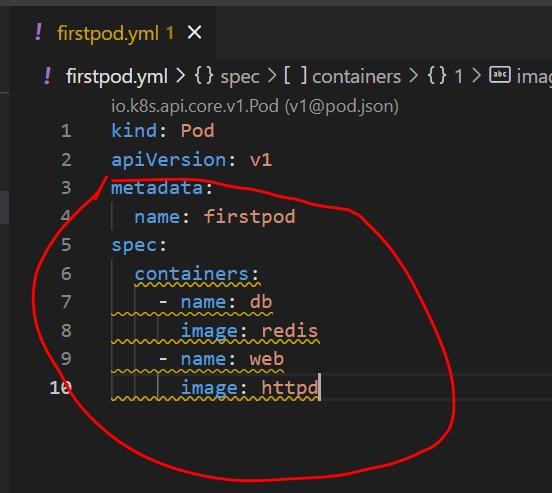
Using the same configuration file we can create,update and delete resource



Note :

We rarely create pod like this , usually pod will created **as part of deployment**

**Metadata, spec :---** all will go into template section of deployment



**POD Life cycle**

**Objective :--**

Different lifecycle phases of k8s pod

The first phase of **Pod is pending :**

this happen when we just create a Pod and api server on master node will validate pod configuration is ok and it will create

Entry in etcd file for every object being created and it will ask worker node to create Pod.

The second phase of **Pod is running**

All of the pod container have been created. The pod has been schedule in one of the worker node for us in the cluster

And all the containers of pod are up and running

The third phase of Pod is succeeded

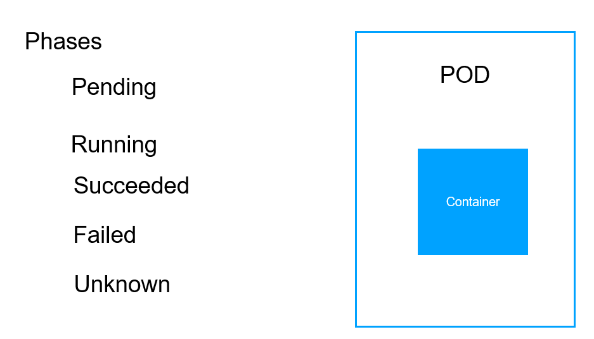
All the containers have successfully run without any errors

The fourth phase of **Pod is failed**

Some of the container not executed successfully

The five phase of **Pod is unknown**

if the pod status could not maintain by master node or by api server



**Status :-- ImagePull BackOff**

It could not pull the image

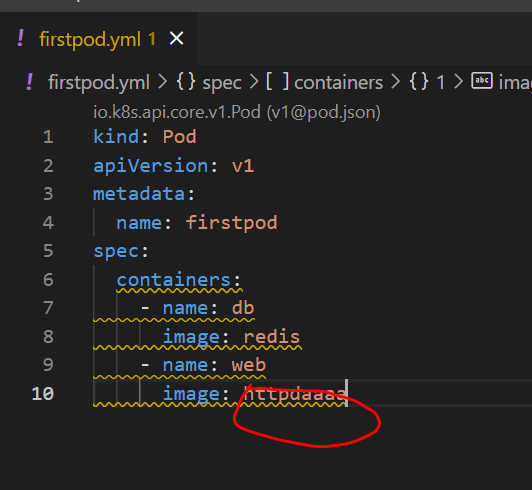
**These all are phases pod can go through depending on how the container is running**

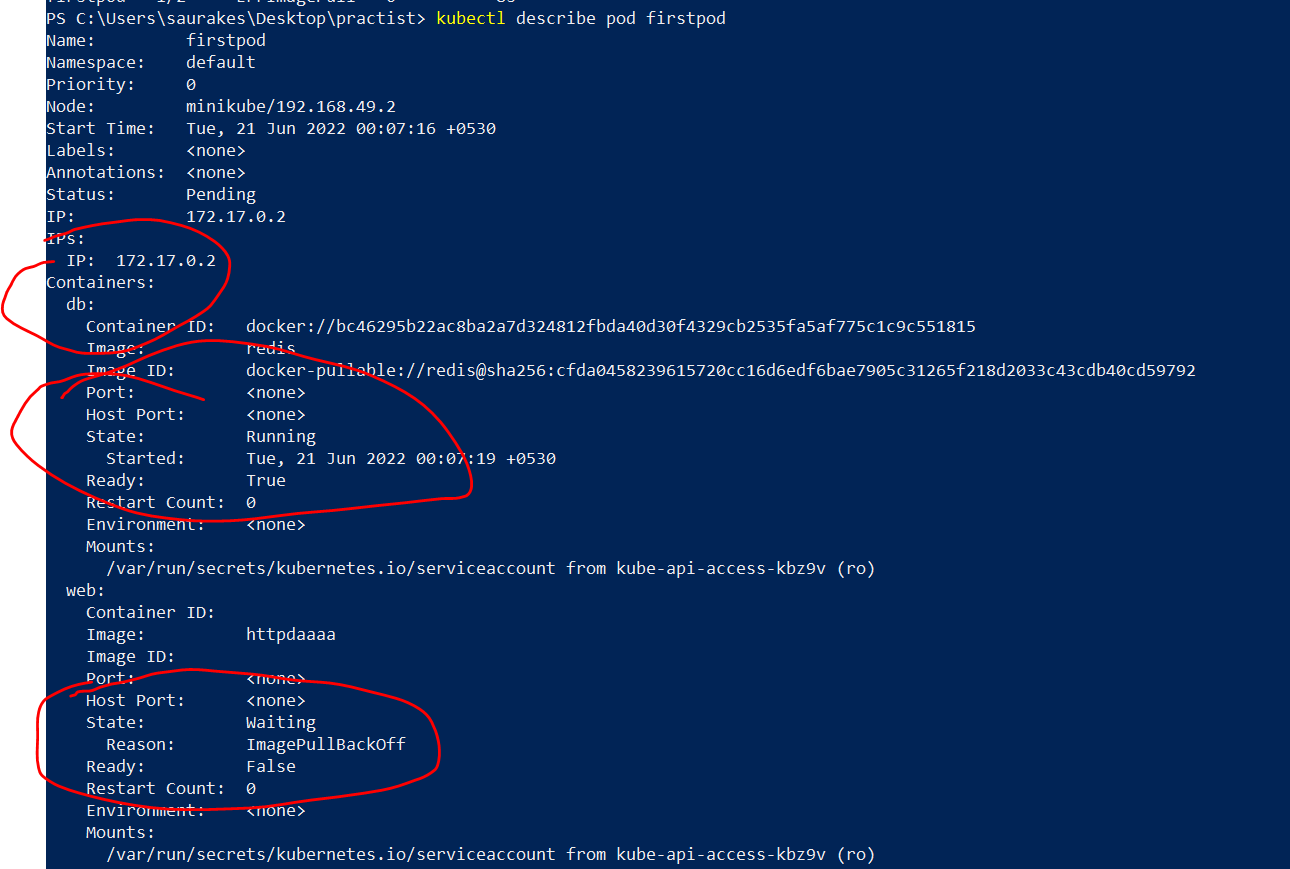
**Note :----**

**api server is responsible** for updating the latest status of pod in etcd file depending upon communication b/w kubelet and api

Server

**Examples:--- let give wrong image name**





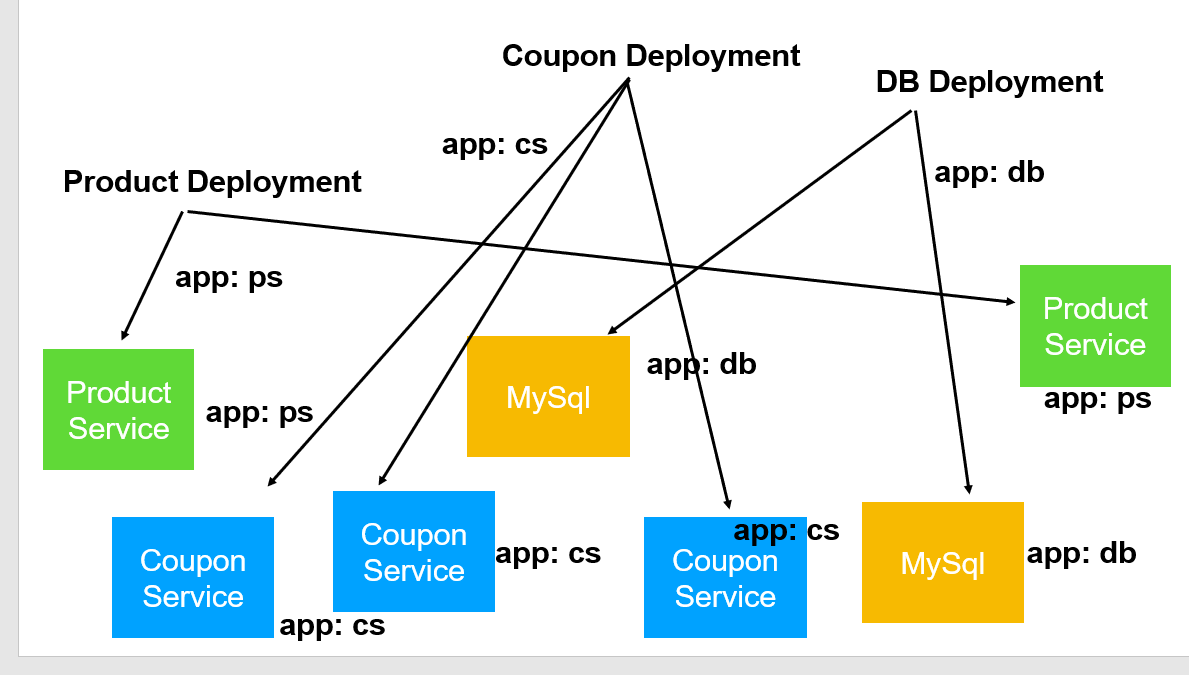


**Labels and Selectors**

**Labels :---**

are key value pair that we are assign to any resource in k8s cluster

Here we have 7 pods :--productservice -2 , couponservice-3 , mysql-2 running the appropriate container



**For productservice :--**app as key and ps as value

**Once we assign these label** we can easily filter these pod ,sort them ,search them , by querying among them using selectors

**So labels and selectors are two good friend.**

First we label the resources then we can group them by using selector

**Here we are saying**

all the labels match **app:ps with selector**  should be a part of product deployment

**all the labels match app:cs with selector** should be a part of coupon deployment

Imp point.

**Label we give to pod and whichever label match with selector all of this will be a part of deployment**

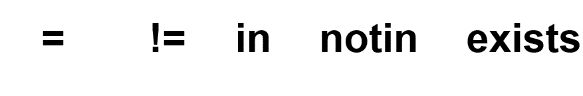
**Pod and service also interact with each other through label and selectoe**

**Label :** are key value pair usually we assign to pod

**Selectors :-** are just like where clauses in sql statement

Note :

We can also use these operator with selector:--



**Use Labels and selectors**

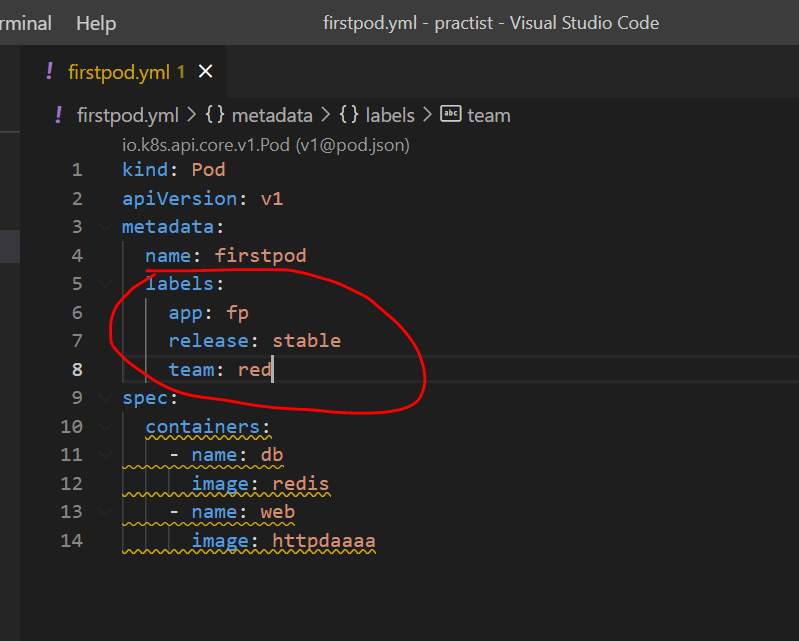
**Objective:**

How to assign labels to k8s resources and how to filter based on those labels using selectors

**Note :-**

**Labels** go into **metadata** along with the **name and they are used to identify the resources using selector used in deployment and service**

**Start with key and value (** these key value pair can be anything**)**

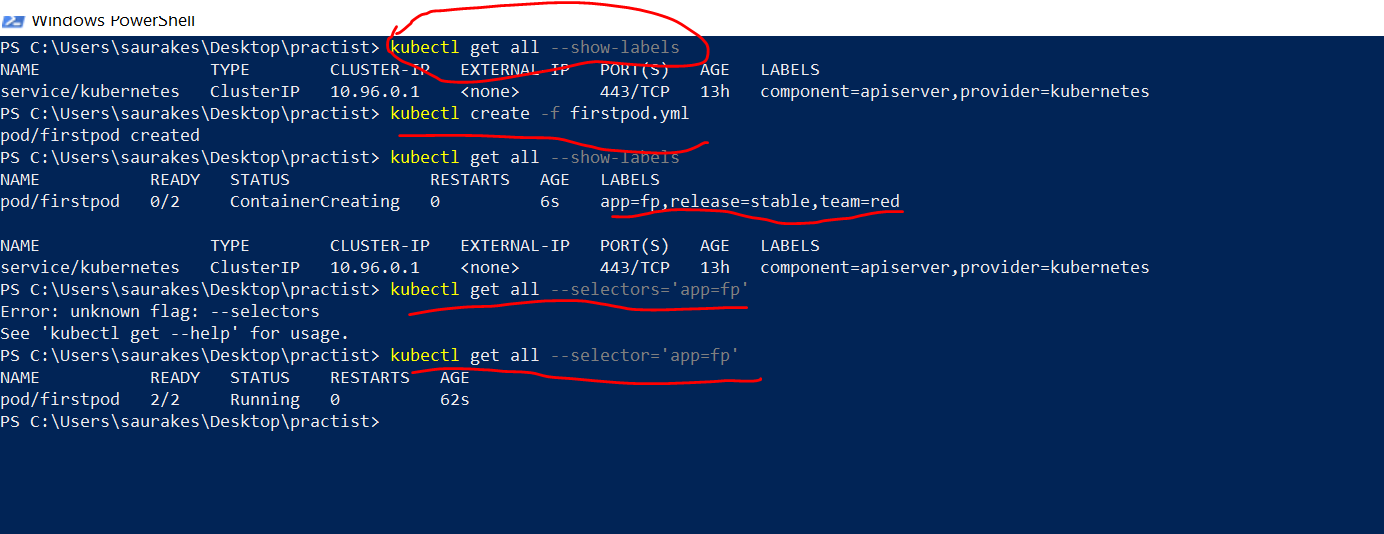


**Once we add label :-**

**Kubectl get all --show-labels :-** this sub command will show label for all the resources

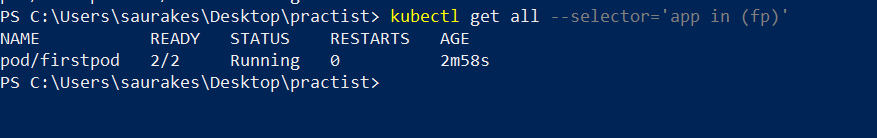
Note :

If there are 100 of resources on k8s cluster and we want to filter out quickly only the resources we care about **we can use selectors**

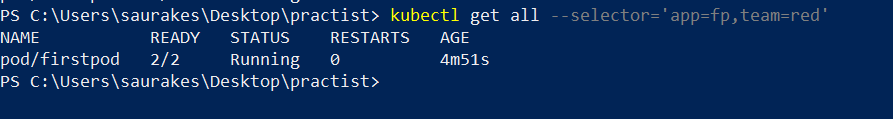


**Similarly we can use in clause**

Whichever pod have matching value fp



**We can use multiple filter also**



**Note**

We will use selector from yaml file later on instead of command line

Inside deployment and inside services we will be using selectors

What is the difference between label and selector in Kubernetes?

Kubernetes labels are key/value pairs that can be attached to Kubernetes objects. **Labels are meant to specify identifying attributes of Kubernetes objects.** **Label selectors are exactly what their name says**. They allow you to select Kubernetes objects based on labels and do interesting things with them.

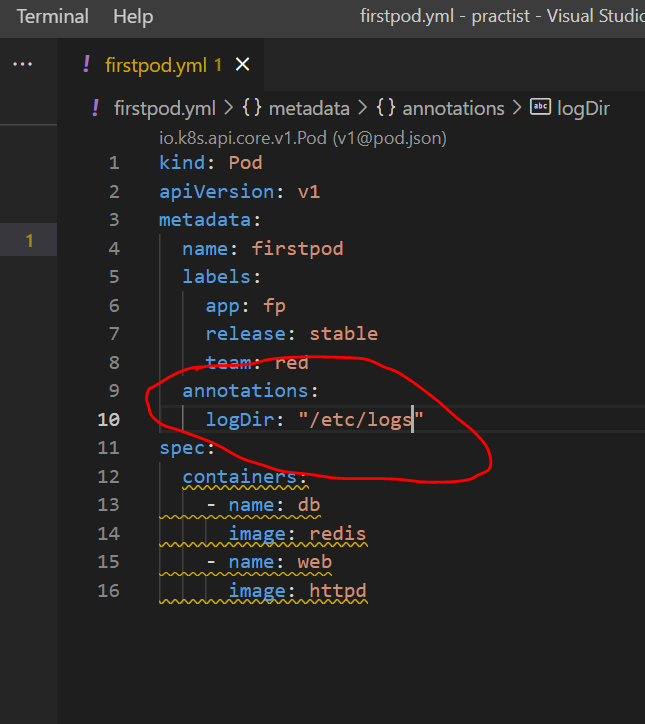
**Annotations**

**Annotation is also metadata**

**Within metadata** we can provide any number of **annotation**

**Annotation :** are key value pair just like label **but they are not used to identify or query the resources.**

It can be used by any other tool like prometheus , graphana etc



This is one example of annotation. Those tools will go into that path grab the logs and will be analyzed by that tool

Just we provide for other developer or tool to read this information and do wtever they want.

It is arbitrary but we provide useful info about the resources themselves

arbitrary  
based on random choice or personal whim, rather than any reason or system.

**What are annotation in Kubernetes?**

Annotations **allow you to add non-identifying metadata to Kubernetes objects**. Examples include phone numbers of persons responsible for the object or tool information for debugging purposes. In short, annotations can hold any kind of information that is useful and can provide context to DevOps teams.

Pod

Replica

Deployment

service

**Namespaces**

**Devopps enginner will hand over namespaces** that we are going to create k8s object

When there are multiple application deployed on k8s cluster.

We don’t want one team delete the pods or deployment of another team accidentally that is where namespaces come in

**Namespaces** :-

**is logical or virtual divisio**n of k8s cluster for each team or application that is deployed on k8s cluster

Each namespace will allocated a resource quota that is cpu , storage or even limit number of k8s object like pods,services,deployment

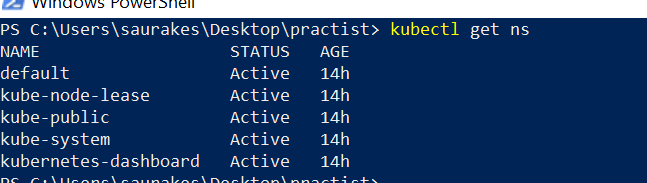
That they can create

This way application are isolated from each other and they have their own personal space

What is kubectl?

Kubectl is **a command line tool used to run commands against Kubernetes clusters**. It does this by authenticating with the Master Node of your cluster and making API calls to do a variety of management actions. If you're just getting started with Kubernetes, prepare to be spending a lot of time with kubectl!

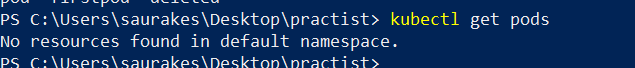
**Kubectl get ns**



*These are default namespaces of k8s*

**Default :** in this ns pods are created

It goes in default namespace by default

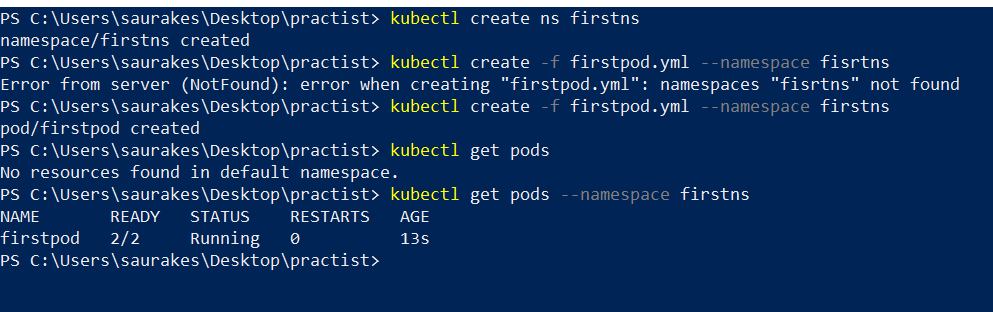


***To create our own namespace :****--- it is just a virtual or logical division of cluster*

**Kubectl create ns firstns**

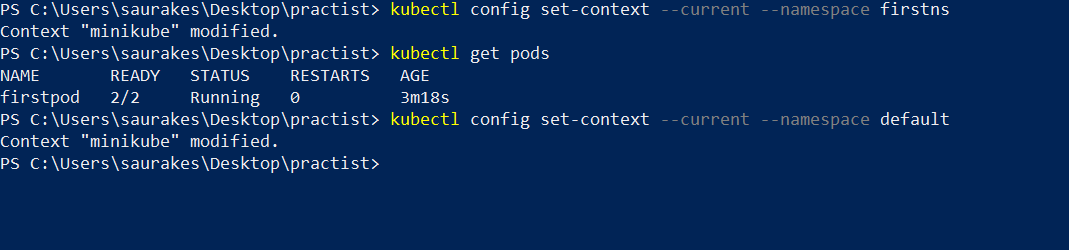
**To use ns while creating pod**

**Kubectl create -f firstpod.yml --namespace firstns**



To switch namespace:

Kubectl config set-context --current --namespace firstns

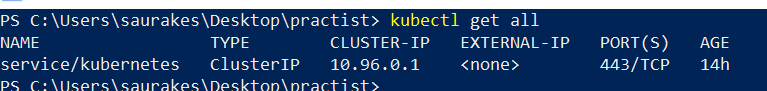


**Kubectl is cool**

**Objective :**

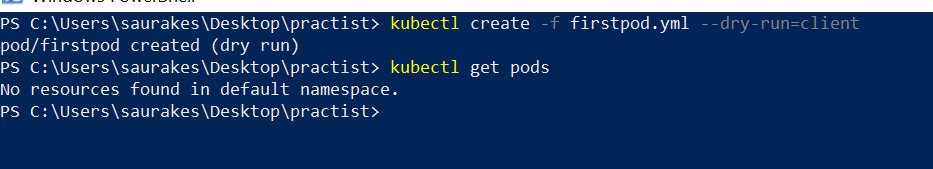
Few cool thing kubectl can do in additional

**Kubectl get all :** it will give all the resources that are currently on cluster



We can do **dry run** if we want check all our step is ok like compilation dry-run means not committed

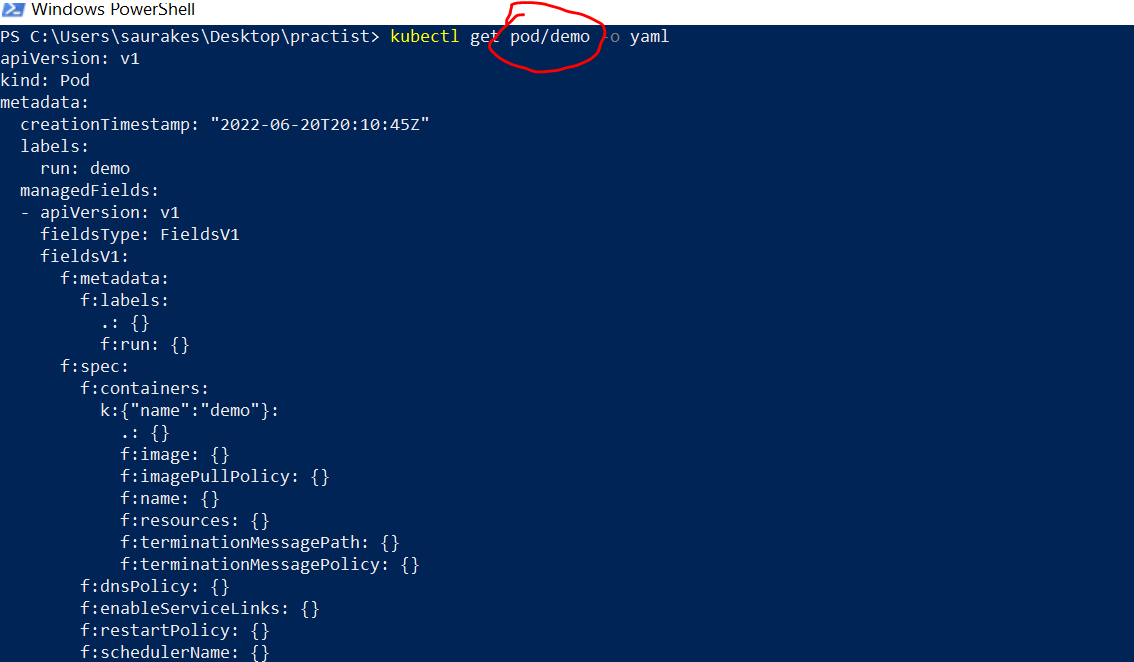
**Kubectl create -f firstpod.yml --dry-run=client**



1. **Instead of creating yaml from scratch** we can quicky get yaml for deployment or service or pod creation

Kubectl run demo --image=httpd ( to launch a pod with http container)

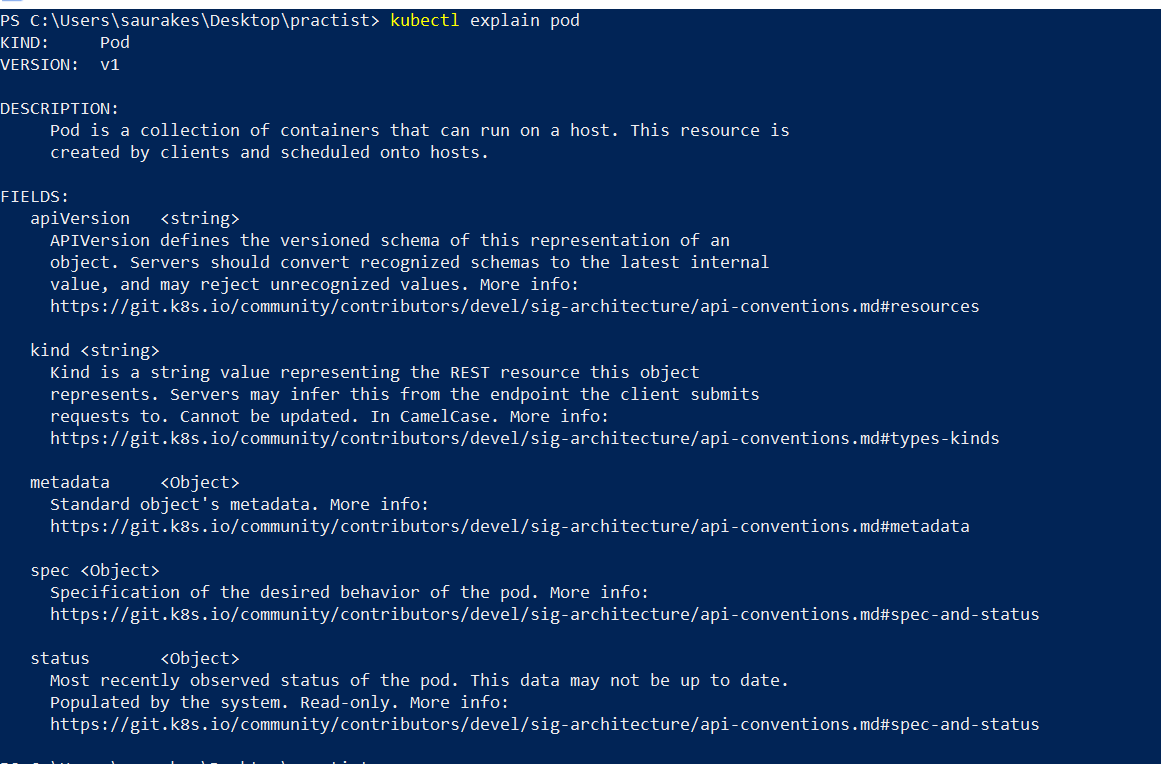
Kubectl get pod/demo -o yaml ( wtever resources we want like pod here we want mention it)



At anypoint we are not sure what the element is :-----

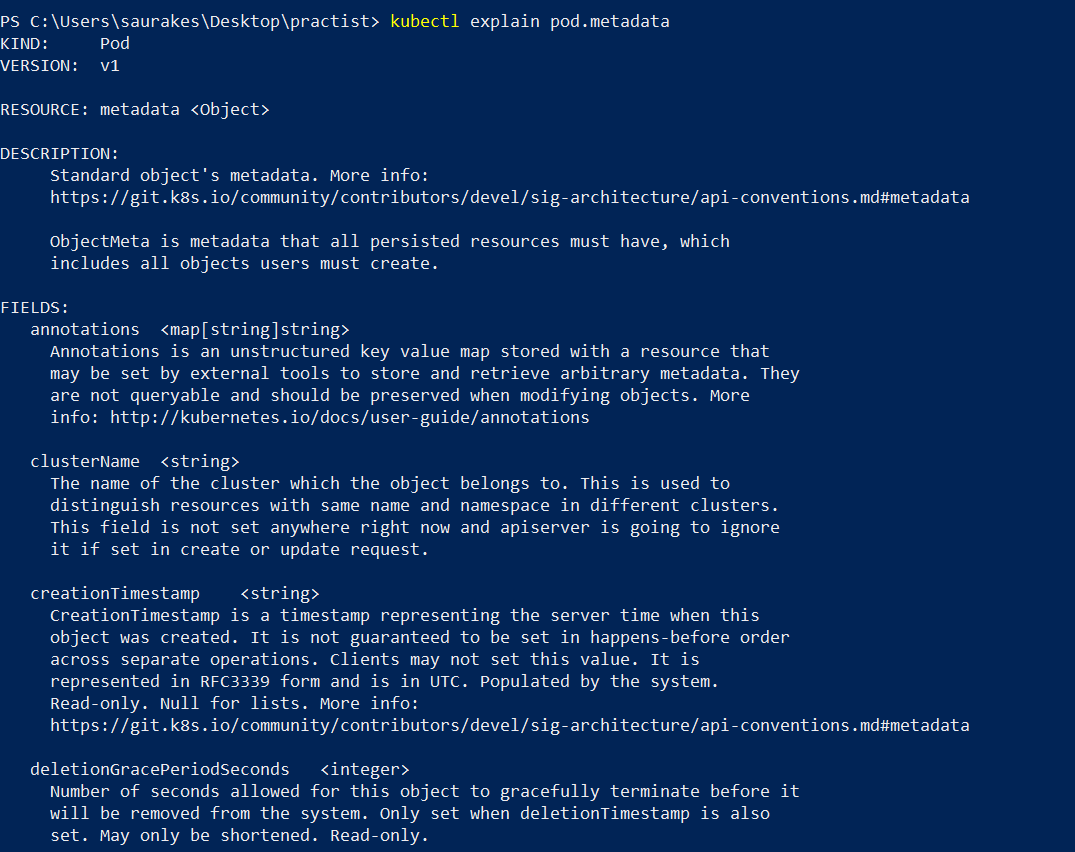
**We can run command**

**Kubectl explain pods**

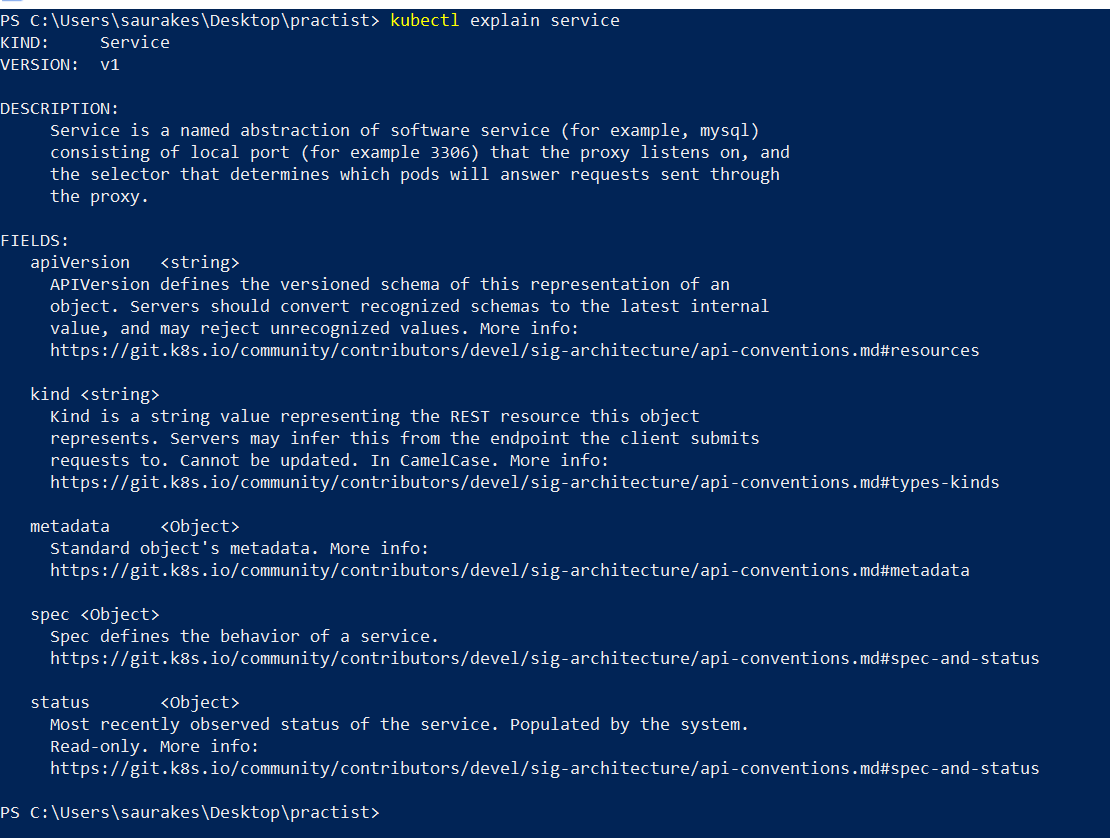


We can also see what can go inside pod element

Kubectl explain pod.metadat



Similarly we can do for any other also



**Deployment**

**Objective :-**

What a k8s deployment object is

The deployment object help us to manage our pods and we can have multiple replica of pod through a deployment.

Example :

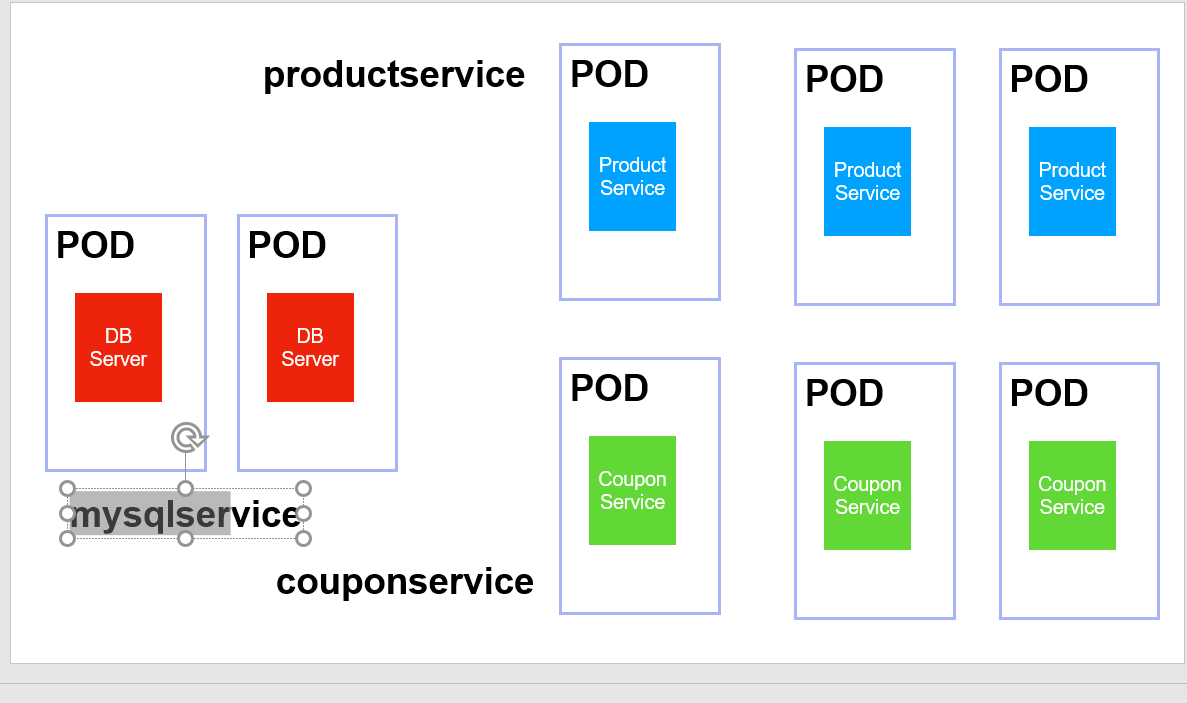
Here we have multiple replica of database service , productservice and couponservice.

So the main goal of k8s is to scale up the pod as the incoming request increase.

Autoscaling is possible through deployment.

Through deployment we will tell how many pods we want and the beauty is due to some reason if one the pod is goes

Down but in deployment file we mention 3 replicas then immediately it will launch a pod



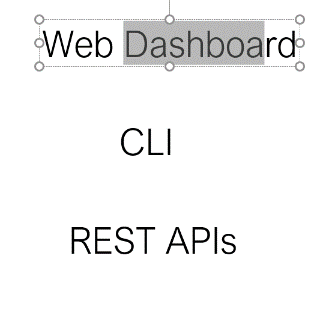
For each microservice we will have separate deployment we can **deploy kubectl which is slowly becoming outdated and deprecated**

**Second way of deployment** is using **kubernetes dashboard**

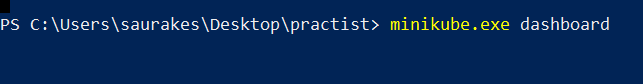
We will create deployment for all m/s

**Access the cluster using Dashboard**

There are three diff ways in which we can access k8s cluster



**Minikube dashboard :** that will enable dashboard

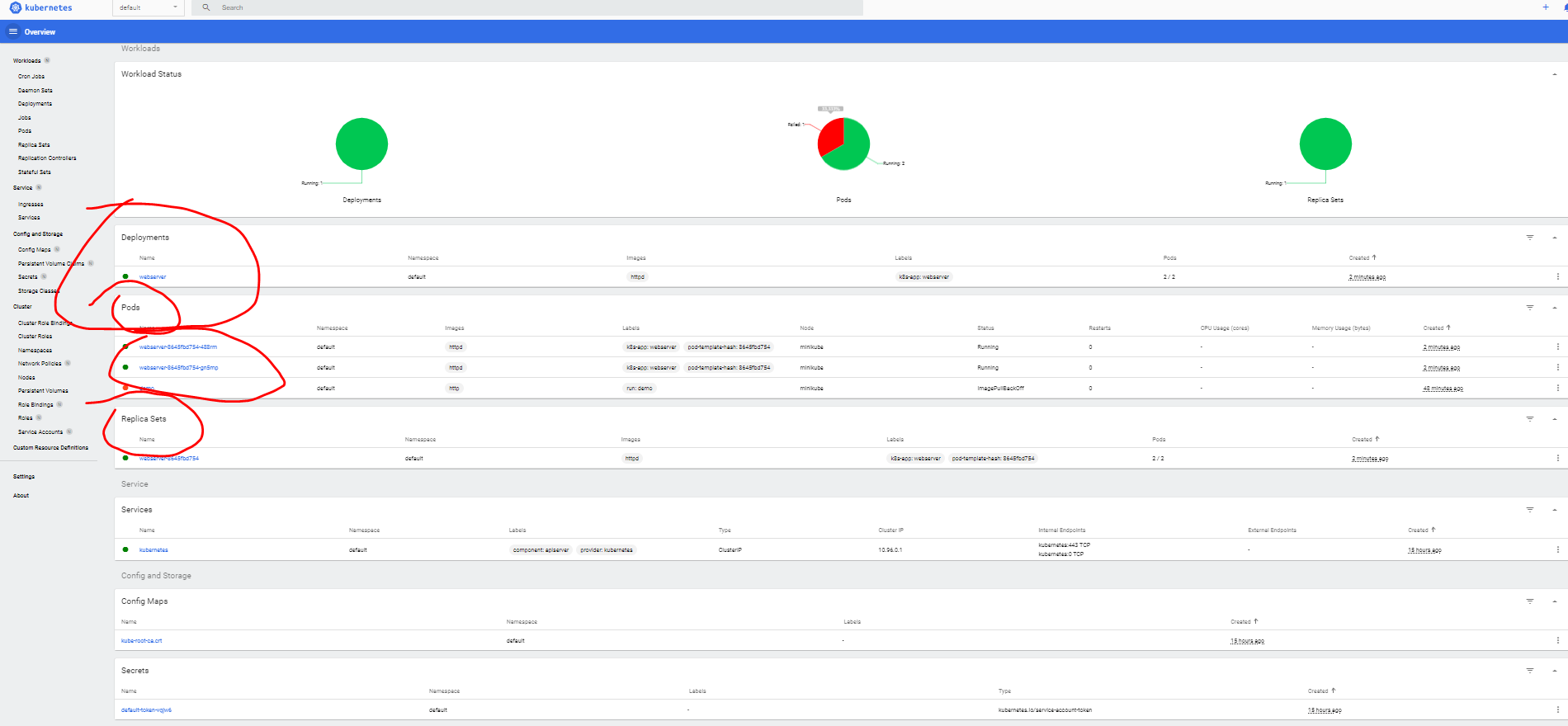


**Now we have access to complete kuberenetes cluster**

We are creating deployment in dashboard



This is just for demonstration purpose



**Kubernetes yaml config explained**

**Objective :**

Understanding deployment and service yaml file

Labels come under **metadata** while selector come under **spec**

**Kind :-** what type of resource we are creating here

Note :

At runtime k8s automatically will add **status** to file

Deployment is always about **replica set** which will come unde**r spec .**

**Replica** set always deal with **pods and pods** is defined using **template.**

**Every resource will have metadata and spec.**

**Imp : replica** are connected to pod through selector

**Label of pod should match with selector of replica** that is how replica set will pick up pod

On which port this service will access the deployment or pods

**Containerport should match with port**

**Note :**

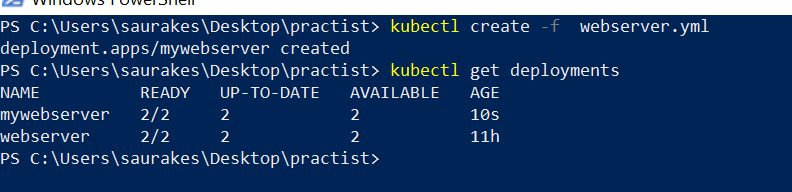
We can also defin**e target port in service** that will be external port to access resourc**e**

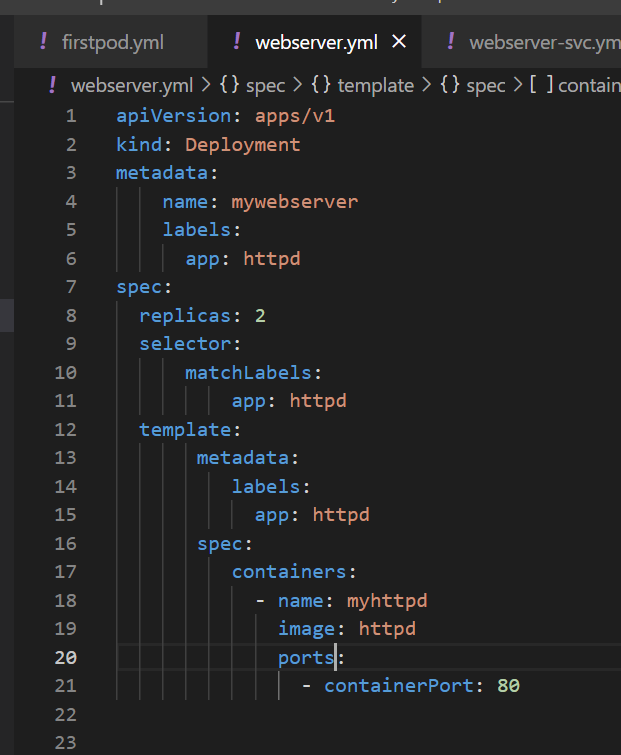
Imp :- **A service is connected to the deployment through label of deployment should match with selector of service**

**Create Deployment yaml**

**kubectl create -f webserver.yml**

**kubectl get deployments**





**Imp :-------------In next lecture we will open this by adding a service by giving access to world outside through service**

**Services and Types**

We have deployed our container into the pods on the cluster and each of these pod have a ip address assigned to them within the cluster. But for these pod to communicate with each we need service object in the k8s object model.

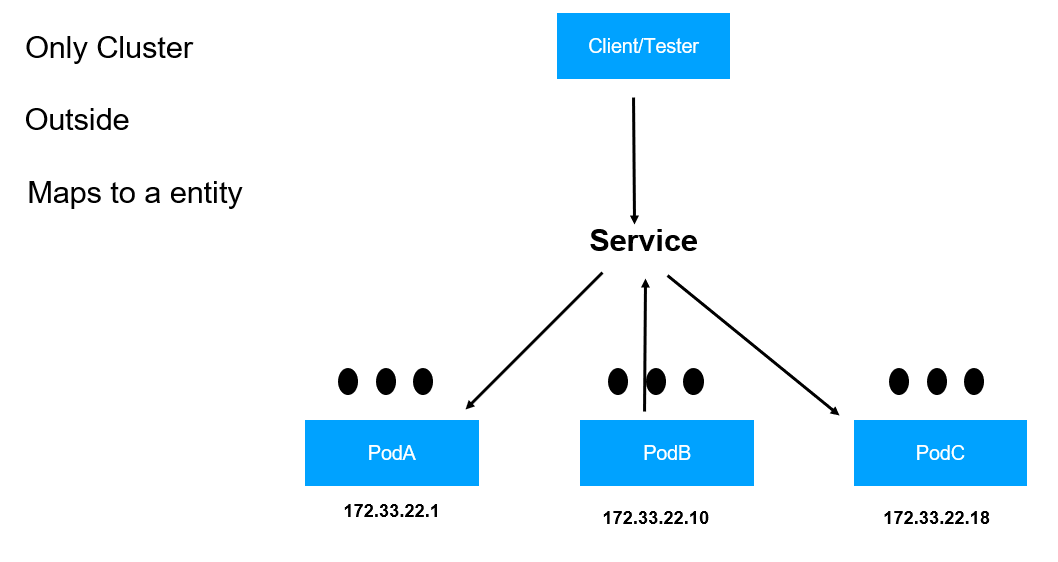
**Definition**

**Services will logically group a set of pod that need to access each other so that they will be able to communicate with each other.**

**Depending on the level of access there are different service type we can configure.**

**Is communication required within the cluster** ( just a pod communicating with each other)

These pod need to access outside the cluster ( then diff service required)



**First service type cluster ip**

**Understand the problem of pod ip address**

**See above diag.**

Here we have 4 pod

pod a, pob , pod c, pod d and pod a

Pod d needs to work with pod a and pod c for this it will use ip address but what happen if one of the pod goes down.

Yes it come back up but the ip address assign to new pod will be different.

Pod d will not have ip address that is where cluster ip come in picture.

Once we configure cluster ip , it will generate a virtual ip address.

All the communication will be happen through virtual ip address.

So pod d will now use cluster ip service and cluster ip service will act as load balancer.

Suppose same functionality provided by pod a and pod c then it will route the incoming request or load to the pod a and pod c.

Pod d only remember cluster ip address , it doest care of pod a and pod c ip address.

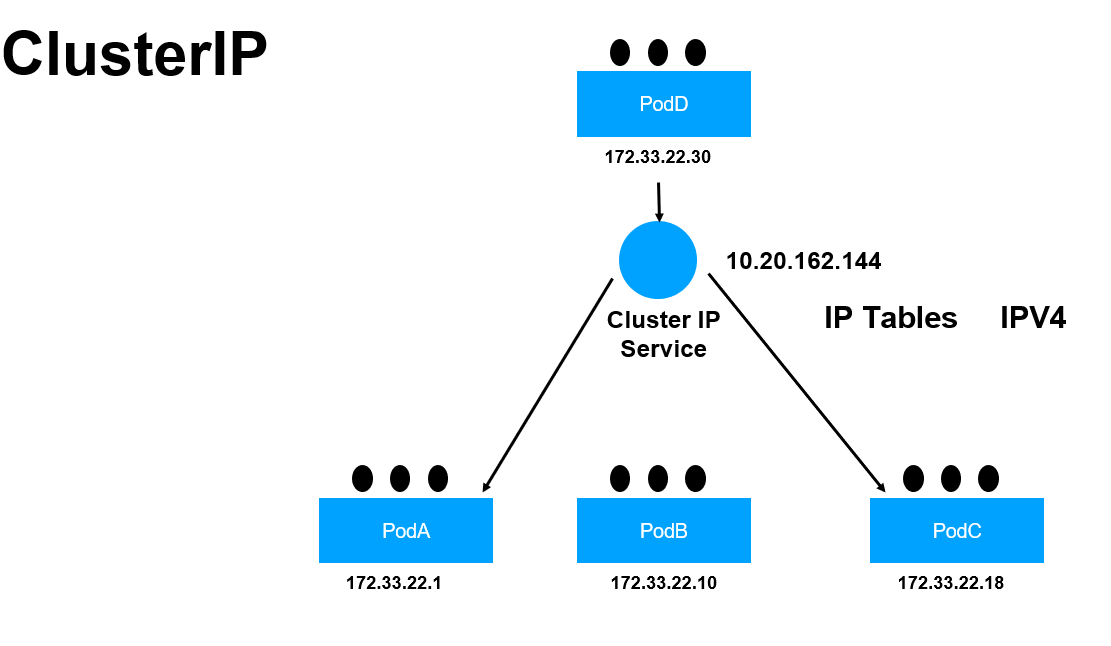
Cluster ip address will be mapped to pod using **ip tables or ipv4 internally.** We need not worry about it

The kube proxy component on worker node is responsible for all these.

Internally virtual ip will be mapped to appropriate group of pod.

Cluster IP is a virtual IP that is allocated by the K8s to a service. It is K8s internal IP

 different pods can communicate using this IP



**Imp point:-**

**When we use cluster ip only the pod within the cluster are able to communicate with each other**

**This cluster ip** cannot be access from outside world that is where **NodePort service type** come in picture

**When we use NodePor**t it creates a cluster ip internally but node port also exposes a higher number of port from 30000 to 32767

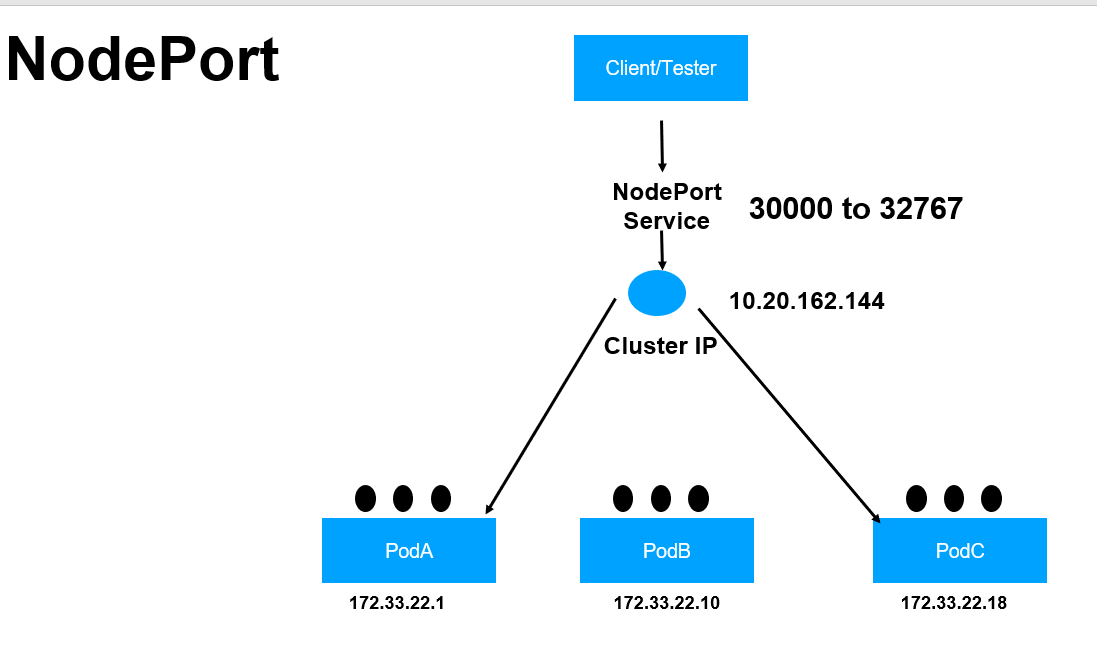
This port range we can configure when we set up k8s automatically port will be assigned.

So the client application outside the cluster will be able to communicate to the pod inside the cluster through the virtual ip

or port number

**In node port of service file if we do not specify target port automatically it will access pod on port 80**

<https://www.densify.com/kubernetes-autoscaling/kubernetes-service-load-balancer>



**Load balancer:**

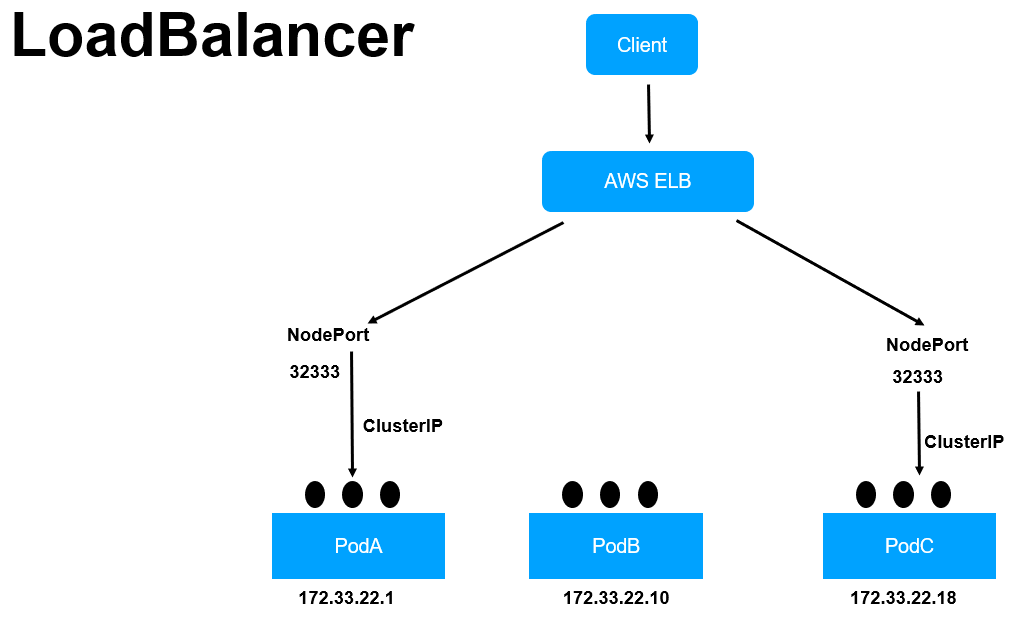
Where we expose service using cloud provider like AWS

When we use load balancer automatically node port and cluster ip will be created internally all the load balancing will happen

But outside will have another load balanacer which will balance the load across the worker node.

Ingress :--

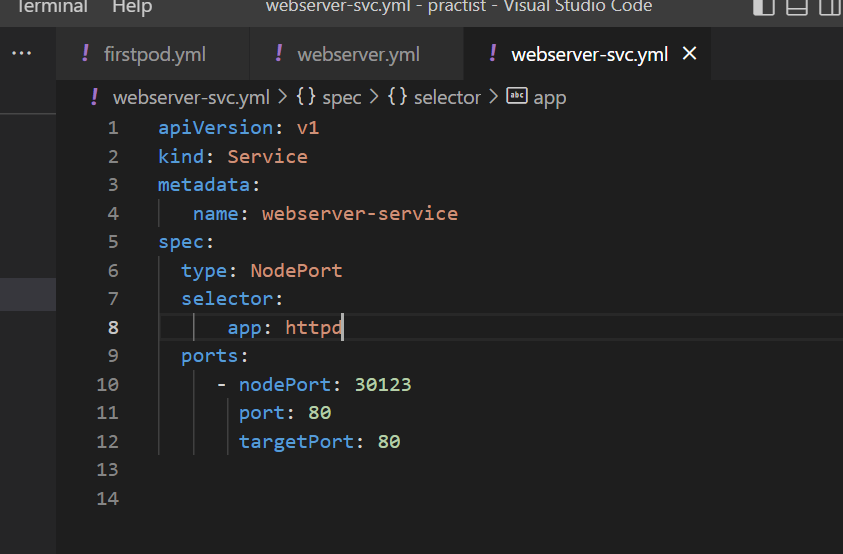
it is not a service type, it is type of it own. In ingress we expose lower number of port



**Create Service**

**Objective** :--- will create a service that will expose out pod to world outside

Service selector should match with label of pod



**nodePort : this is port with which service will expose outside. This will be high number**

**Port :---** this is the port on the pod itself within the cluster these pod can be access

targetPort : this is the port on the container itself

When different application run on different pod to use this particular port :80

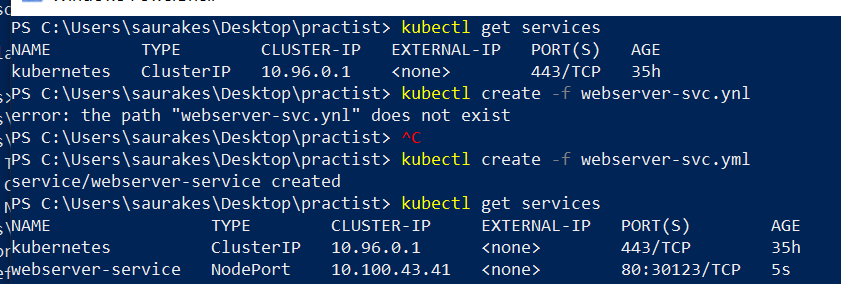
when any traffic come on 80 will be redirect to targetport :80 on the container itself

**Selector of service will pick up all the pod that match with pod level**

**Windows issue**

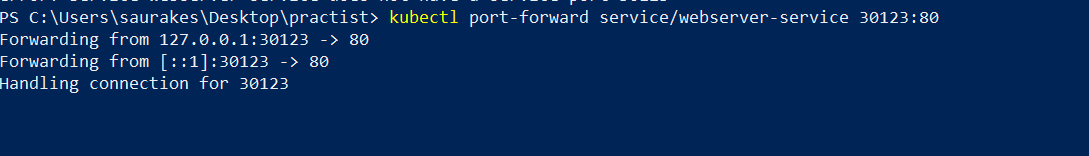
**Kubectl get service**

**Kubectl create -f webserver-svc.yml**



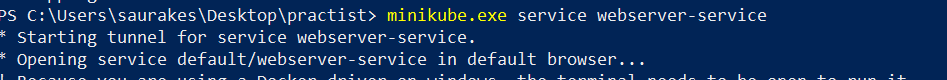
**First run this**

kubectl port-forward service/webserver-service 30123:80

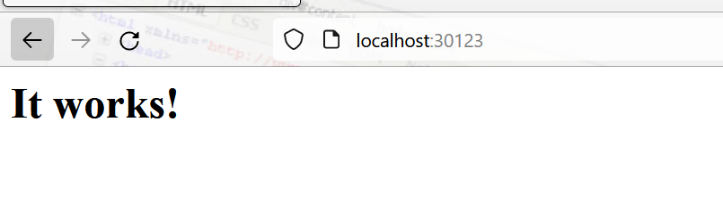


Then run this

**Minikube.exe service webserver-service**



It will open a browser

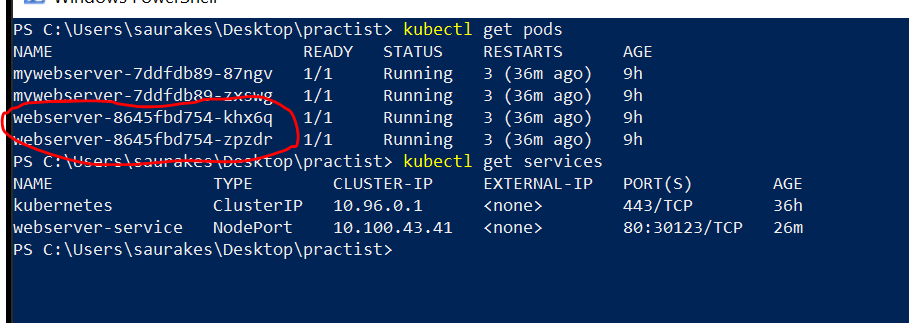


**Load Balancing**

**In previous lecture**

The request goes to service on this particular port (30123).

Service is responsible for load balancing the traffic across the pod we have



We have two different pod which are running httpd container so it will load balanced the load between these two pod

So this is the service that will receive the request on port 30123 and it will take the responsibility of sending the traffic to port 80 on the container running on these two pods automatically. It will load balancing

**Let go into of these pod**



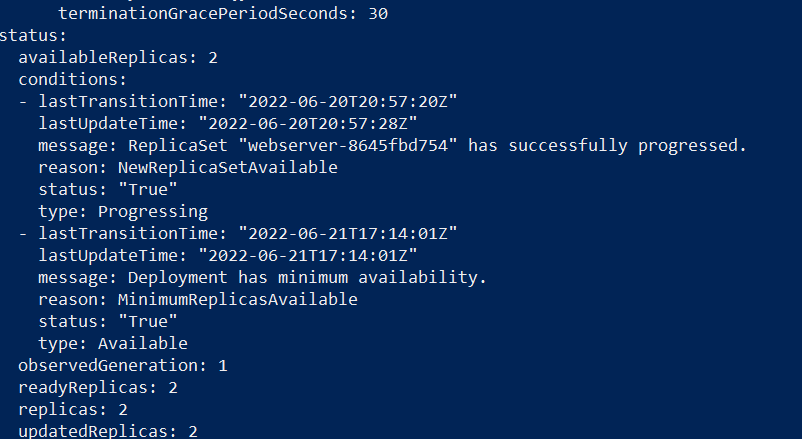
**Deployment and service are decoupled with each other. They work by selector and labels**

**Status in yaml**

Status will be added to yaml file at runtime

**kubectl get deployment webserver -o yaml**

**All the runtime info will be added by kubernetes** especially the status section



**Rolling updates**

So far we have seen k8s deployment give us easy way to replicate our application on the cluster.

Deployment also allow us to update these application in the cluster with 0 downtime

**Example :**

**Let us suppose we have to update version in https.**

For this k8s uses two strategy

**First update strategy**

**Recreate :--**

k8s will destroy all the pods and then it will recreate all the pods. It doesn’t promise zero downtime

We can think in case of m/s where we don’t want older version of m/s

But the most promises one with zero downtime is **rollingupdate**

**It will start creating new pods** but it will not bring down old pod until the update happens

**MaxUnavailable(60) :**

suppose 100 pods are then k8s immediately bring down 60 pods

As the number increasing it will destroying the older pods

**MaxSurge**

**How many new pods it should start as soon as update is executed**



**Rolling update in actions**

**Go to deployment file:- webserver.yml**

Change no of replica to 10

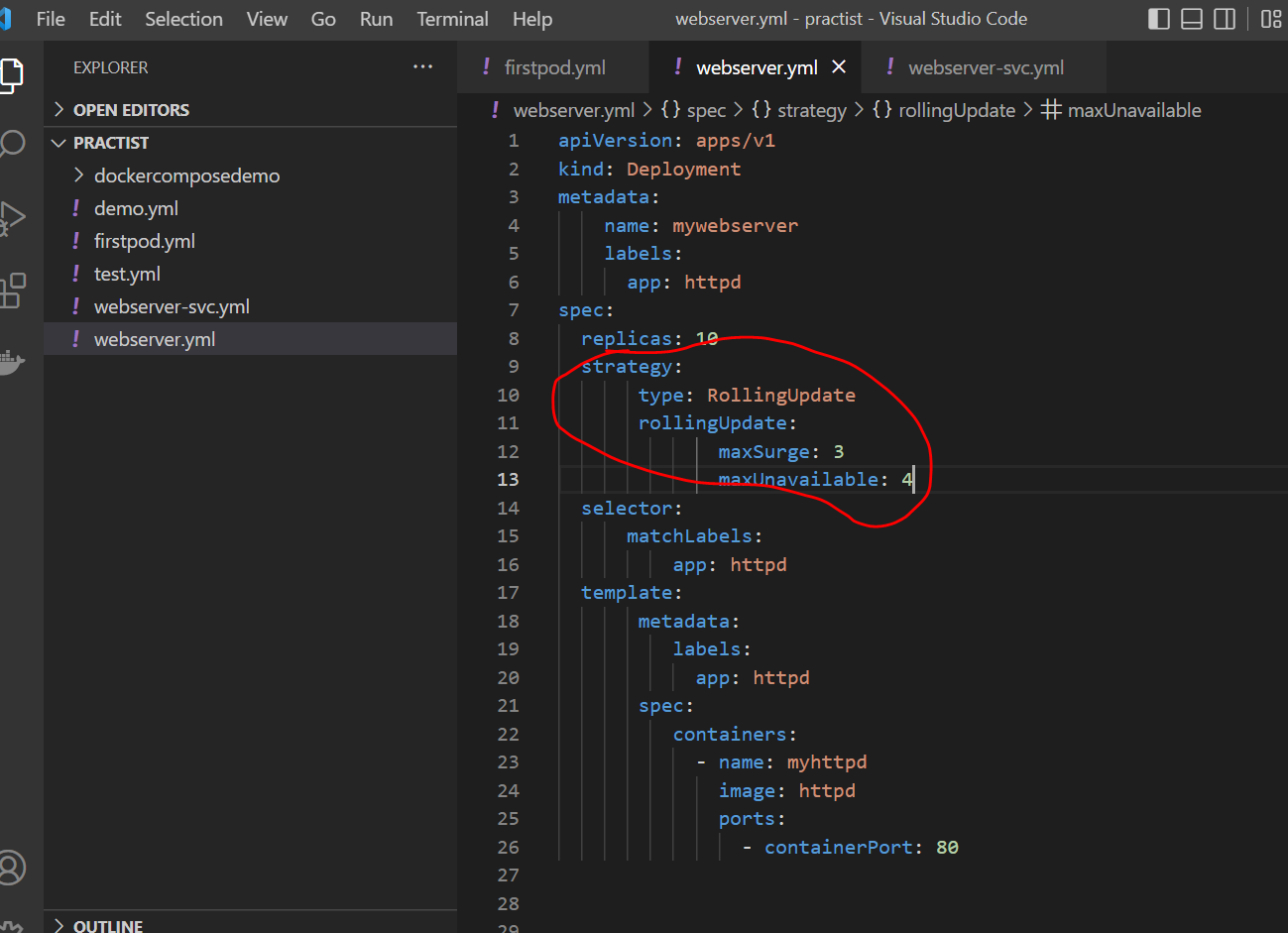
maxSurge :- 3

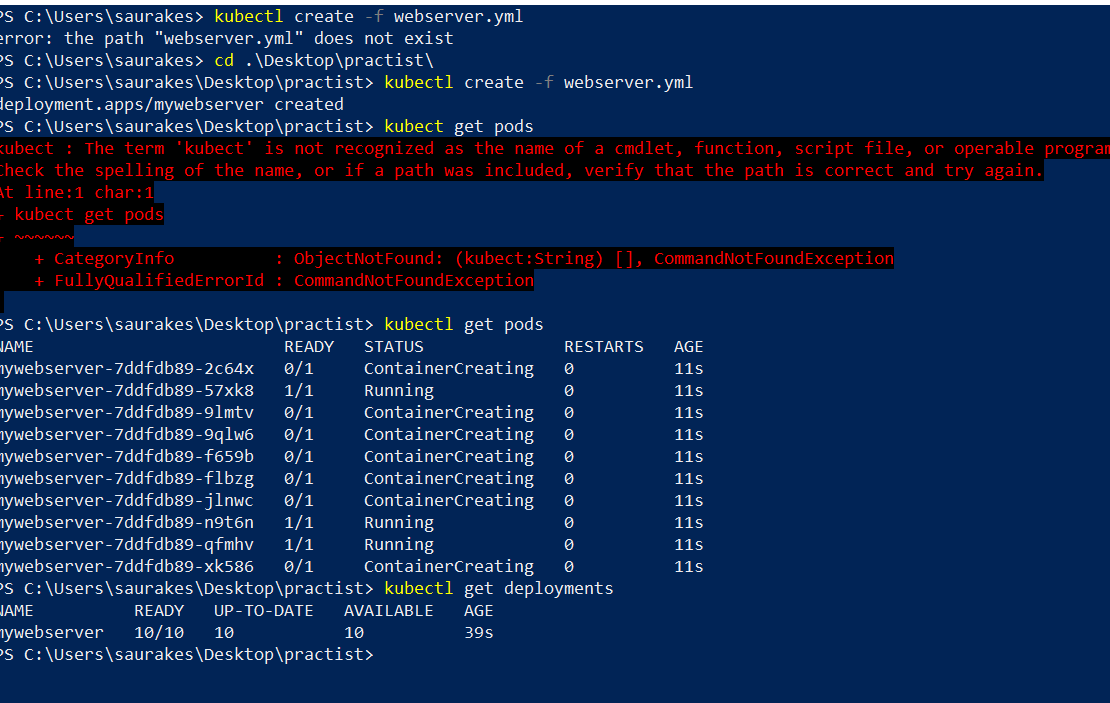
new pod are getting created. How many no of pods can be immediately created as soon as we push update from command line.

So total no of pods will be 13 as soon as update start

maxUnavailable :4

This is for old pod. Using this no we tell to k8s. How many old pod it will immediately kill





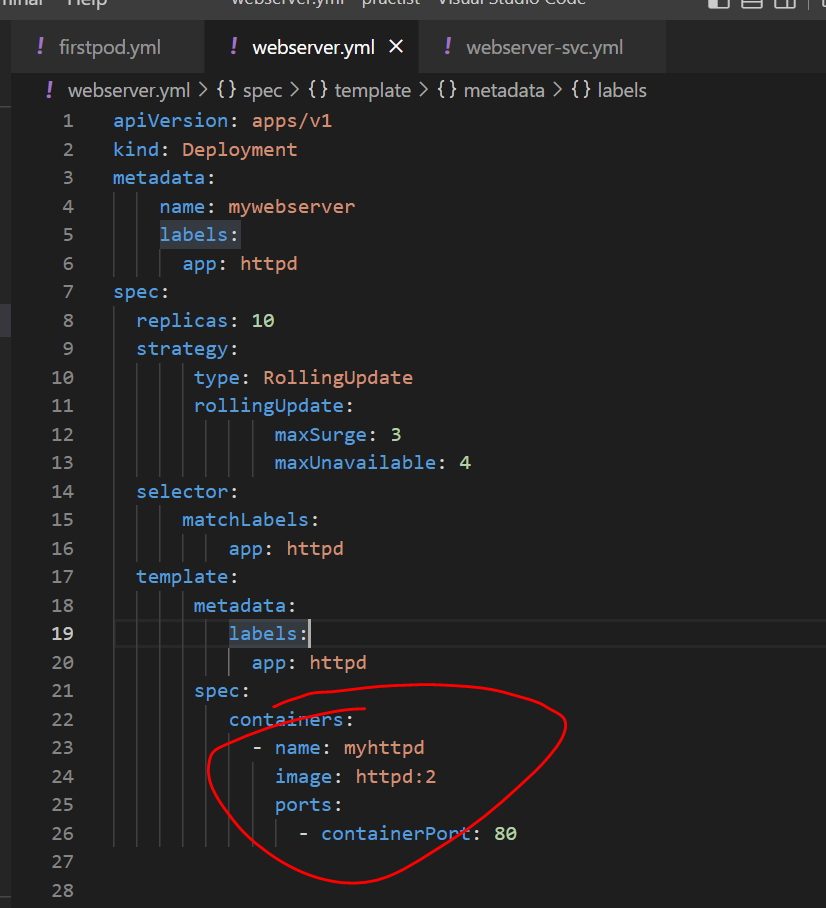
**Requirement :**

**For some reason we want to use older version of httpd**

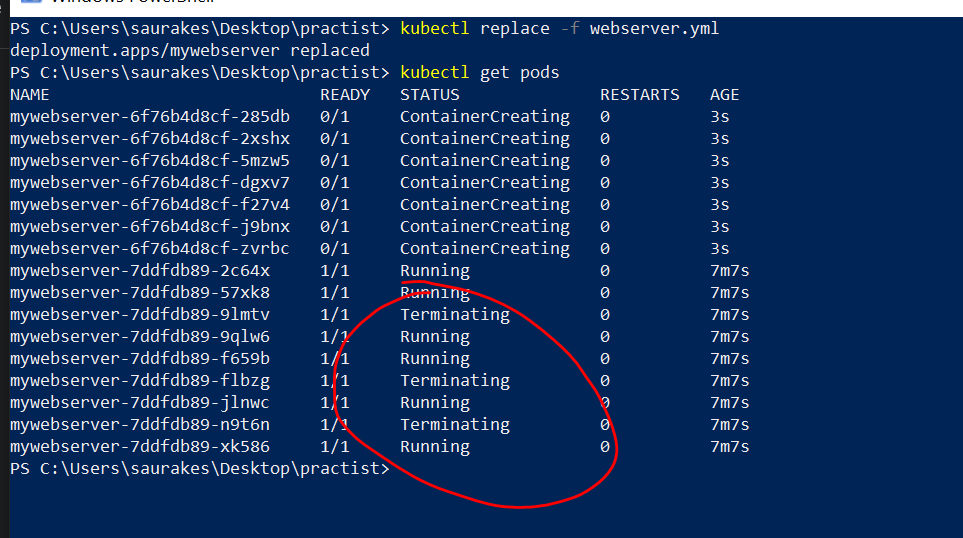
**Kubectl replace -f webserver.yml :---**

we are telling to k8s replace the cluster with the change apply

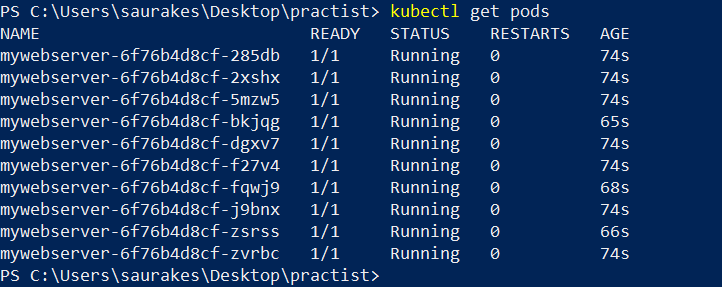
It will pull the version and use the strategy we defined it here



It didn’t terminate all . First it terminate 4 then new pod start creating as the new pod come it start terminating old pod



**If we see strategy , it terminating pod and also making up pod**



Now cluster is up and running with new pods without any downtime

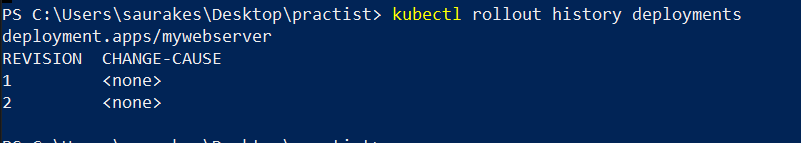
**Rollbacks**

**Important**

**Kubectl rollout history deployment**

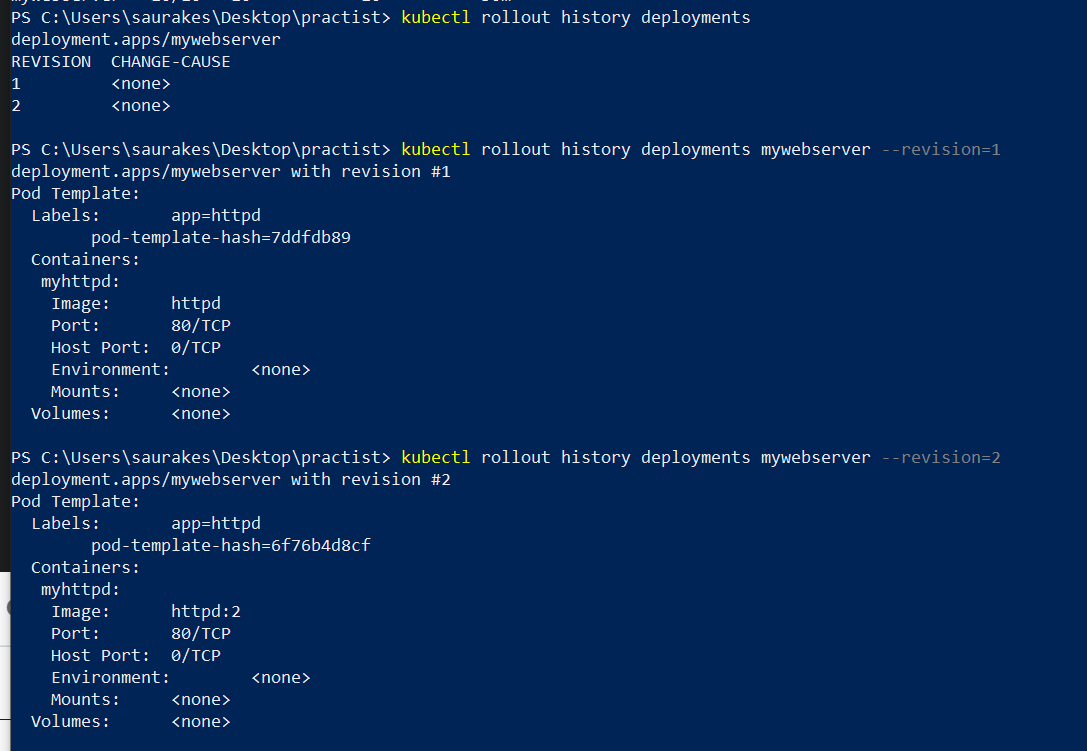
**K8s** internally maintain version of every **deployment we make**

So that we can go any version of the deployment we want



**We want see changes between version :--**

**Kubectl rollout history deployment mywebserver --revision=2**



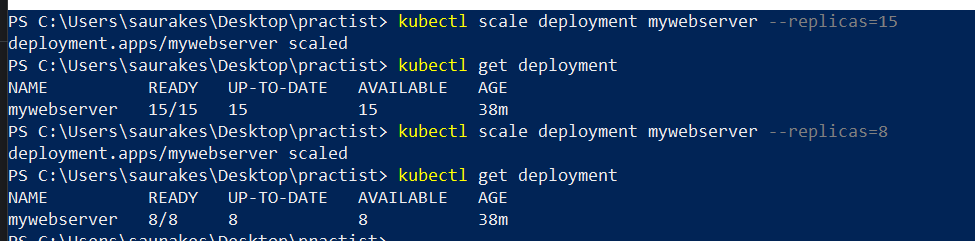
We can roll back to any version also

**imp -----------------Kubectl rollout undo deployment mywebserver --to-revision=1**

**Manual scaling**

**Objective**

Manually scale the replicas



**Section 3- Kubernetes Volumes**

**In docker** we have connection b/w container and host machine

In case of k8s the connection is b/w pod and container. Volume is directly connected to pod.

Each container **get some space on pod and that space is mounted on folder in container**

Container can also exchange data through **pod volumes.**

Diagram

Description automatically generated

**First define volume at pod level them mount volume on the container**

There are diff types of volume are there but most supported one

Graphical user interface, text, application

Description automatically generated

**emptyDir :**

simply we will get some space on pod and all other container will mount that space in container.

**Nfs(n/w file system volume) :**

This is beyond pod

**hostPath**

Similar to docker volume then the data will directly live on node on which this pod is running.

**configMap and Secret** :

are k8s object but internally they are volume as well

**configMap :--**

configuration info for k8s . We can store d/b info like sql query

We can save all these file ( config file) in git repositories

It will live on k8s cluster not on pods

**Secret :**

We can store like pw for m/s application. It store sensitive information

**Mount Volumes**

**Objective :**

How to mount volume on container within a pod

These volumes will be use by all the container within the pod

Mounting will happen from pod to a container

**Minikube.exe ssh** :- this is where pod is up and running

A picture containing chart

Description automatically generated

**This will act as pod**

Text

Description automatically generated

This will go inside container

Graphical user interface, text, application

Description automatically generated

**Use ConfigMaps**

**Objective :**

How to create and use configMaps

configMap do not have spec section , it will have data because it going to provide data for the container

Initdb.sql : this is key which could be anything ----

this will be created as file in container and value will be as data in that file.

We can have any number of keys and values.

Text

Description automatically generated

**Requirement** :

to mount the config map within our pod configuration

Create a new volume , a configMaps is internally a volume as well.

These two name should match

Text

Description automatically generated

Text

Description automatically generated

Now give path in pod means which docker path we want to mount it.

So far we have created configMap configuration yaml where we defined key and values and then mounted

configMap as volume in deployment file

Text

Description automatically generated

Text

Description automatically generated

**Expectation : we should see key and values as files inside the mounted path**

**Create configMap and deployment**

Graphical user interface, text, application

Description automatically generated

**We can mount configMap to any container and key will be available as file**

Text

Description automatically generated

**Secrets**

It store sensitive information.

K8s store sensitive info inside **tmpf .**

Instead storing them directly into node it will store into temporary location.

**Imp:**

**It will only available in node when pod request for it. Once the pod is done using secret it will delete the secret from the node.**

What is secret in Kubernetes?

A Secret is **an object that contains a small amount of sensitive data such as a password, a token, or a key**. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.

Why ConfigMap is used in Kubernetes?

A Kubernetes ConfigMap is an API object that **allows you to store data as key-value pairs**. Kubernetes pods can use ConfigMaps as configuration files, environment variables or command-line arguments. ConfigMaps allow you to decouple environment-specific configurations from containers to make applications portable

What is ConfigMap and secret in Kubernetes?

Kubernetes has **two types of objects that can inject configuration data into a container when it starts up**: Secrets and ConfigMaps. Secrets and ConfigMaps behave similarly in Kubernetes, both in how they are created and because they can be exposed inside a container as mounted files or volumes or environment variables

There are diff types of secrets

Opaque :---arbitrary key value pair

A black screen with white text

Description automatically generated with low confidence

In secret it is sensitive data so we will be **using base 64 encoding instead of directly storing them**

**Using Secrets**

**Objective**

How to create and use secrets

**Opaque** : arbitrary data means it is not related to any docker or k8s data

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated with medium confidence

To go inside container:- kubectl exec -it pod it -- bash ( pod id created after creating of deployment)

Text

Description automatically generated

Text

Description automatically generated

**Persistent Volumes**

**Objective :**

What PersistentVolume and PersistentVolumeClaim

If we want a volume beyond pod and node then will use PersistentVolumeClaim.

It is space at cluster level and the pod want to use some of persistenvolume can request is through PersistentVolumeClaim.

Every m/s running on pod or application can request space through PersistentVolumeClaim.

Diagram

Description automatically generated

A **PersistentVolume** (PV) is a piece of storage in the cluster that has been provisioned by server/storage/cluster administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like node. A PersistentVolumeClaim (PVC) is a request for storage by a user which can be attained from PV

step

Graphical user interface, text, application

Description automatically generated

**First create** persistentVolume which is nothing but space on the cluster and PV will live as long as cluster live

Whether yr container die or pods are deleted and even nodes are deleted on the cluster yr volume will stay there

Pod will request space through PVC once we have PVC will mount PV to containers in a pod.

In real time PV will maintain by cluster administrator. They can use aws storage , azure storage. While doing them

They give us diff access mode  **how these volume can be access.**

There are three diff access modes

**ReadWriteOnce** :-- only one node in the cluster can read and write

**ReadOnlyMany** :- any number of node in the cluster can read to this volume.

**ReadWriteMany** :- any number of node can read and write to this particular space

**Create PV**

**In spec we provide three imp thing**

Capacity

accessMode

hostPath : where this volume will be stored in minikube

Text

Description automatically generated

Text

Description automatically generated

**Create PVC( persistent volume claim)**

**Within spec** will request amount of storage we want

We are claiming 8M for our pods from PV

Text

Description automatically generated

Text

Description automatically generated

**Mount PVC**

**Objective :**

Mount PVC to a container

Text

Description automatically generated

Persistent volume still exist even after deleting all:  **kubectl delete all --all**

A picture containing chart

Description automatically generated

It goes away only when cluster itself is gone

**Section 4 - Deploy Microservice to kubernetes**

**Objective:**

Deploying m/s application to k8s cluster and will expose to outside world through k8s services.

Diagram

Description automatically generated

**How application on the cluster communicate**

**Will use same as service name**

Text

Description automatically generated

Text

Description automatically generated

**Imp**

m/s deployed in k8s cluster can be communicate using service name and port number

**Create DB Deployment**

Will deploy d/b for our m/s to work to the k8s cluster

Graphical user interface, text, application, chat or text message

Description automatically generated

Pod label should match with replica selector so that they will pick up as part of deployment.

Mysql\_root\_host ='%'--------------- from which ip address it should be access ( anywhere)

Text

Description automatically generated

**Create ConfigMap**

To initialize our d/b schema

Initdb.sql :---- a file with this name is create and all the value will be there inside it

Text

Description automatically generated

Here all the content will go in a file called initdb.sql under the mounted volume of container

**Mount Volumes**

**Objective :**

will mount configMap as a volume in our deployment to the pod.

Once the container is up and running **it will look sql file inside the folder /docker-entrypoint-initdb.d and it will execute sql file**

A picture containing text

Description automatically generated

Initdb.sql file will be available in path **/docker-entrypoint-initdb.d** which is mounted on the container

Text

Description automatically generated

**Create DB Service**

**Objective :**

Create a service that will expose out mysql outside the cluster.

It can be outside the cluster as well , mysql workbench locally.

**Note:**

**Selector in service must be equal to label in in pod of deployment under template section which will decide which all pod**

**Need to pick up by service**

A screenshot of a computer

Description automatically generated with medium confidence

**Deploy and Test Db**

**Objective**

Will deploy d/b to k8s cluster and also we will create that will expose it out

**We will clean the cluster**

Minikube.exe stop

Minikube.exe delete

Minikube.exe start ( will start a single fresh node cluster)

Text

Description automatically generated

**Will be using this ip to connect to d/b**

Text

Description automatically generated

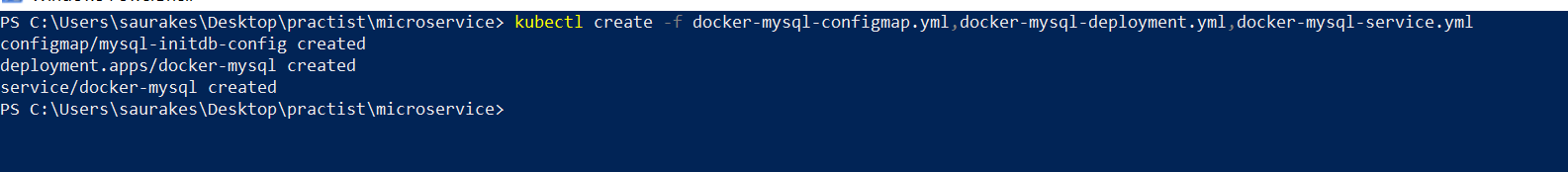
In single command will execute all three

Text

Description automatically generated

kubectl create -f docker-mysql-configmap.yml,docker-mysql-deployment.yml,docker-mysql-service.yml

Service name will be container name



Text

Description automatically generated

**Minikube ip means cluster ip**

**Port 30287**

Graphical user interface, text, application, email

Description automatically generated

**Create Coupon and product yaml**

Will create deployment and service yaml for coupon and product m/s

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

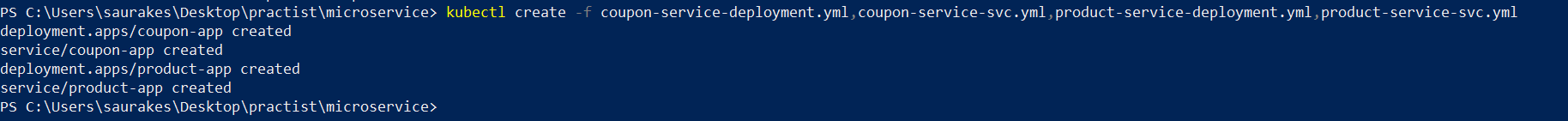
A screenshot of a computer

Description automatically generated with medium confidence

**Deploy Product and Coupon app**

**Will deploy all at one shot**

**kubectl create -f coupon-service-deployment.yml ,coupon-service-svc.yml, product-service-deployment.yml,product-service-svc.yml**



Graphical user interface, text

Description automatically generated with medium confidence

**Test m/s**

**Using tunnel m/s is working**

Graphical user interface, text

Description automatically generated

kubectl port-forward service/product-app 30289:9090

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Coupon**

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Section 5- Eclipse Jkube Quick start**

**Objective**

We don’t need to write dockerfile and kubernetes file

**Eclipse jkube internally support three types of configuration**

**Zero configuration**

By default cluster ip we can define certain properties in pom like want to use nodeport

**Generate Docker Image**

We define a restful endpoints

Graphical user interface, text, application

Description automatically generated

JKube Docs:

<https://www.eclipse.org/jkube/docs>

Maven Repo:

<https://mvnrepository.com/artifact/org.eclipse.jkube/jkube>

Use Plugin: add in pom.xml

<plugin>

<groupId>org.eclipse.jkube</groupId>

<artifactId>kubernetes-maven-plugin</artifactId>

<version>1.0.0-rc-1</version>

</plugin>

Graphical user interface, text, application

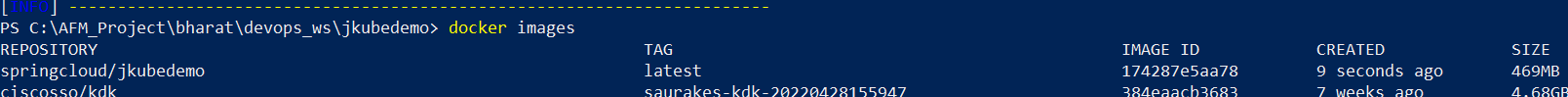
Description automatically generated

**That all is required to generate docker image**

**Note :**

**K8s point to plugin** automatically

**Build Docker images:** mvn k8s:build



Note :-

It drives the image name what we have in pom.xml

Text

Description automatically generated

**Deploy to Kubernetes**

**Objective :**

Deploy docker images on k8s cluster

Step 1- Minikube should be up and running

Text

Description automatically generated

Step 2- :

we need to see images in k8s cluster inside minikube

Connecting to minikube docker daemon

Step 3:

**minikube.exe docker-env | Invoke-Expression --- it will go into minikube daemon**

**Docker images**

This will show image on minikube not local

Graphical user interface, text

Description automatically generated

Step 4

**Generate Kubernetes Resource manifests:**

mvn k8s:resource

It will generate file here

Graphical user interface

Description automatically generated with low confidence

Graphical user interface, text, application

Description automatically generated

Step 5:

**Apply resources to cluster:(deploy) to minikube (k8s cluster)**

mvn k8s:apply

**step 6 : test**

kubectl get pods

kubectl get svc

Minikube.exe ssh

curl <ip>:<port>

Text

Description automatically generated

**Use Node Port**

update properties in pom.xml

<properties>

<java.version>1.8</java.version>

<jkube.enricher.jkube-service.type>NodePort</jkube.enricher.jkube-service.type>

</properties>

Change Service type to NodePort:

<jkube.enricher.jkube-service.type>NodePort</jkube.enricher.jkube-service.type>

minikube ip

curl <ip>:<port>

Access Logs:

k8s:logs