Kubernetes

Reference links

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Kubernetes is a container management/orchestration tool, where the process of deploying, scaling containers is automated, here based on load pods/containers will be auto scaled

2 containers can communicate with each other

1. **Why kubernetes and advantages**
2. Dynamic scaling ,ex:- if ur appln cpu is 90% utilized we can configure to create more instances, based on the load we can scale up the number of instances, if cpu utilization is less then we can automatically scale down

Image, container both are docker inspectable objects

Network is also a docker object—which is inspectable

docker inspect <container-name/id>

2)kub architecture

In kub cluster, we have a below components called

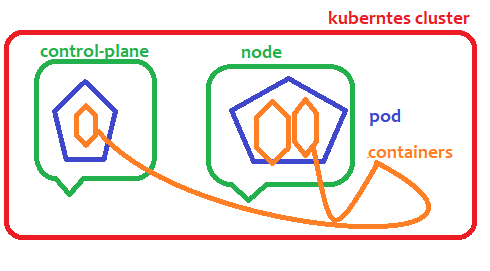
In real time, every environment will have separate cluster

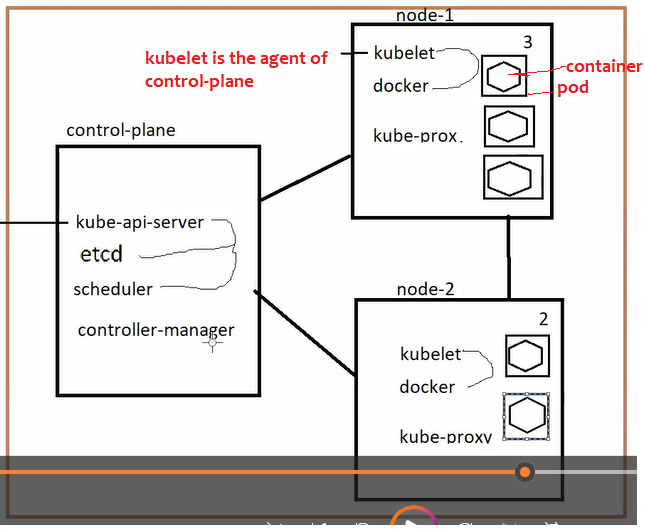
Ex:- dev environment cluster is separate and prod env cluster is separate,

If u want to interact with kube cluster, then we have to use “kubectl.exe”

1)control plane-which doesn’t carry any load, hence it wont have any container,hence docker also is absent

2) node-usually means, 1 computer /1 virtual machine





What Is pod

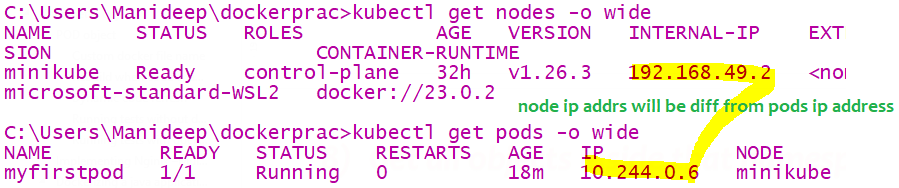
Pod is the way to collectively deploy 2 containers.

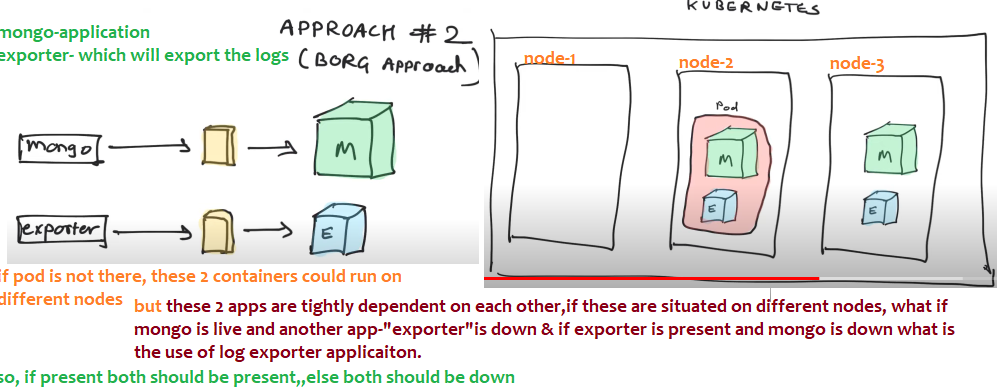
Therefore if 2 apps are tightly dependent, instead of deploying them as separate containers in separate vms

Better deploy them in same pod , always that single pod reside in single node,pod is a packaged,

Every node in a cluster will have some ip address and

Every pod inside a node will have some ip address





U can delete the entire pod or u can up the entire pod-

POD means just an abstract layer where containers will reside

Pod is a smallest deployable unit, in kub we can deploy only till pod, we cant directly start a container

1 node can have many pods,it is a shell around a container

1 pod can have 1 or more containers,which could be tightly dependent

Kub suggests, even if 1 container is there run that in a pod,

Lets say there are 2 images we created 2 containers for these 2 applications, these 2 are tightly dependent with each other

1) Components of control-plane

This control plane component –similar to master node,but it doesn’t carry any load ,it doesn’t have any containers

since it is not going to have containers ,control plane component doesn’t have docker installed ,

it doesn’t have any pods, and

1)**kube-api-server :-** this will receive all the incoming commands and it will validate all those commands

And it will send those validated commands to etcd

2)**etcd**:- it a database, all the state/information about entire cluster will be stored here,

like how many nodes,how many replicas,how many pods are there

in unix “etc” is a folder to store the configuration data for single system,here d means distributed,which stores the configuration for distributed systems

3)**The Scheduler** :- which schedules your apps (assigns a worker node to each deployable component of your application)

4)**The Controller Manager:-** (replica manager/node manages)

Its job is to ensure desired number of replicas and nodes(Virtual machines) are running

which performs cluster-level functions, such as replicating components, keeping track of worker nodes, handling node failures, and so on

THE NODES

The worker nodes are the machines that run your containerized applications. The task of running, monitoring, and providing services to your applications is done by the following components:

Docker, rkt, or another container runtime, which runs your containers

The Kubelet, it is the agent of control-plane node, which talks to the API server and manages containers on its node

Kubelet will talk to docker container and it will create the containers inside a pod

The Kubernetes Service Proxy (kube-proxy), it is to assign the ip-addresses, which load-balances network traffic between application components

In AWS- we have EKS-called managed cluster- Elastic kubernetes service-here entire kubernetes cluster is ready

Cluster with-control plane and kubernetes node-where those are preinstalled with docker

Kubernetes commands

1) Kub cluster commands

#### Get cluster info

kubectl cluster-info

To get nodes

kubectl get nodes

1. Basic commands
2. kubectl api-versions

//to see all the versions like v1…

1. Executing yaml file

Lets say if the file “myapp.yaml” file contains pod create script,If u type “kubectl create”-it will create the pod

If u type “kubectl delete” it will delete the pod

kubectl create –f ./myapp..yml

kubectl apply –f ./myapp.yaml

kubectl delete –f ./myapp.yaml

C:\Users\Manideep\dockerprac>kubectl apply -f ./createredispod.yaml

pod/redis created

here yaml file is executed and pod called redis is created

for example if pod creation script is present in myapp.yaml file, then if pod is not there at all then we should use create command

if pod is already present,if u are making any changes the use apply command

1. CRUD -pods-Create,Delete pod

|  |  |
| --- | --- |
| Get pods | kubectl get pods  to see the pod name |
| Get pods from all namespace or another namespace  //Get pods from different namespace being in current namespace  kubectl get pods -n <namespace-name>  ex:-  kubectl get pods -n kube-system  kubectl get pods -n maninamespace | kubectl get pods –A  or  kubectl get pods --all-namespaces |
| Get pods with more info like ip address-if u want to know which pod is on which node along with ip addresses then use –o wide | kubectl get pods –o wide |
| Describe a pod- most famous one-  To get pod information like – 1) how many containers it have and all 2) what are the names of all containers…and 3)what is the pod ip address (both node and pod will have ip address)  4) to know who is controlling this pod-means either replicaset/ replication controller  5)to know all the labels of a pod | kubectl describe pod <pod-name>  ex:-  kubectl describe pod d1-depl-obj-name-7877f6d747-96plb |
| See the source code of pod in YAML format | Kubectl get pod <pod-name> -o yaml  Ex:-  kubectl get pod d1-depl-obj-name-7877f6d747-96plb -o yaml |
| Delete a pod | kubectl delete pod <pod-name> |
| Edit a pod- if u want to update the image or update the lables/scale the replicas we should edit the pod | kubectl edit pod <pod-name> |
| To see who Is controlling this pod –I mean is   1. Replicaset controlling the pod or 2. ReplicationController controlling the pod | kubectl describe pod <pod-name>  ex:-  kubectl describe pod d1-depl-obj-name-7877f6d747-96plb  if u describe u should see value for field “controlled-By” |
|  |  |

Here If u observe, if u describe the pod there is a field called “controlled by”: “replicaset/…” means here

This pod is getting controlled by replicaset



1. Create a pod

Pod can be created in 2 ways

1. Using kubernetes run command (deprecated)

The below command will directly create a pod and run this image inside a container

C:\Users\Manideep\dockerprac>kubectl run p1 --image=nginx

**pod/p1 created**

1. **U**sing declarative approach-file

Write all the commands in a file and execute using

kubectl apply -f ./createredispod.yaml

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

spec:

containers:

- name: nginc1

image: nginx:latest

Using run command (deprecated)

C:\Users\Manideep\dockerprac>kubectl run p1 --image=nginx

pod/p1 created

##here u used “p1” hence a pod with name p1 is created

C:\Users\Manideep\dockerprac>kubectl get po

NAME READY STATUS RESTARTS AGE

myfirstpod 1/1 Running 0 42m

nginx 1/1 Running 0 81s

p1 0/1 ContainerCreating 0 3s

1. If u want to get all pods inside a particular namespace

kubectl get pods –n <namespace name>

1. To get all the pods inside default namespace

EX:- kubectl get pods -n default

kubectl get pods

means it will try to fetch those pods in default namespace

1. To get pods with more information like ip address or u can describe the pod

here output of pods will be printed in wide, means more output will be printed

**kubectl get pods –o wide**

1. Watching a pod

kubectl get pods --watch

1. Describe a pod

**kubectl describe pod <pod-name>**

kubectl describe pod redis

Lets say if u want to know how many containers are running inside that pod then u have to describe that pod

If u describe u will get all the ip addresses of nodes and pod

Like u will get all the container names inside a pod, image id everything about that pod

There Is a difference between describing a pod and reading a pod

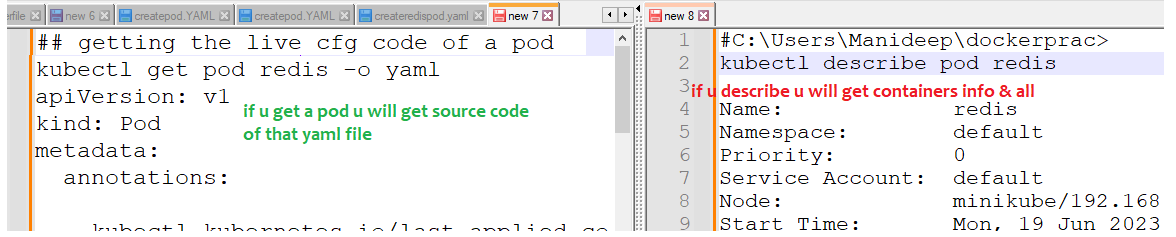
1. Read a pod-getting the live configuration of that pod

This will get the source code of the yaml file with which it was created,

To get more information of the pod in yaml file(output of the yaml file)

**kubectl get pod <pod name> -o yaml**

ex:- **kubectl get pod redis -o yaml**

****

1. EDIT a pod-once u edit and editor will be opened and to save changes just close that file

Now if u edit – u will get live configuration of that pod

**kubectl edit pod <pod-name>**

1. Deleting a pod

**kubectl delete pod <podname>**

kubectl delete pod myfirstpod

Deleting objects/pod using created file

kubectl delete –f ./pod.YAML

means with which file this pod is created , with that file we should use delete command

here all the kub objects (pod,namespace,replicaset) mentioned in this file will be deleted

Go inside a container

1. Exec cmnd inside a container of a pod

kubectl exec –it <pod name> -c <container-name> bash

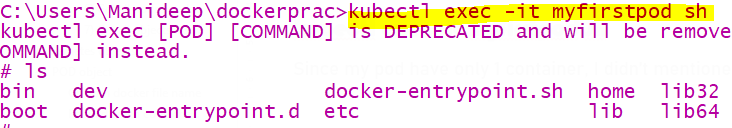
//use either bash or sh, “sh” must be mandatorily available

kubectl exec –it <pod name> -c <container-name> sh

if there is only 1 container in ur pod no need to use –c flag

kubectl exec -it myfirstpod sh

Since my pod have only 1 container, I didn’t mentioned –c flag



1. Describe a pod –to see cont name

kubectl describe pod <pod-name>

ex:- kubectl describe pod myfirstpod

if u describe the pod u will get container name,and image with which it was created

also of all containers running inside that pod

1. Get all containers in a pod

Just describe the pod , u will get the list of containers present inside that pod

kubectl describe pod <pod-name>

1. CRUD nodes

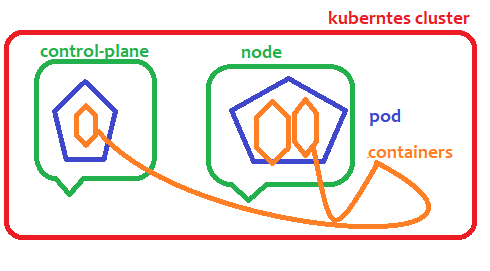
A separate linux machine is called a node, in AWS we will create a separate linux machine called as node

Every node and every pod will be having an different ipaddress

|  |  |
| --- | --- |
| Get all nodes | kubectl get nodes |
| Get all nodes with more information-means output in wider format | Kubectl get nodes –o wide |
| Describe a node to get info like- ipaddress,labels of that node | kubectl describe node <node-name>  kubectl describe node minikube |
| Labelling a node | Kubectl label node <node name> key=value |
| Show all the labels of a node | 1. Using describe command 2. Using kubectl |
| Deleting the label of a node |  |

Remember node is nothing but one virtual machine where docker is installed,

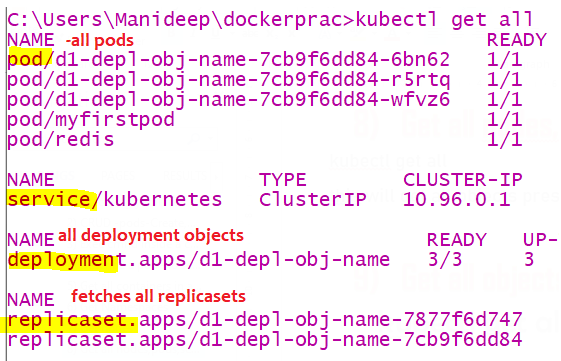
master node doesn’t carry load , so docker wont be installed in the control-plane node



1. Get all nodes,pods,services everything

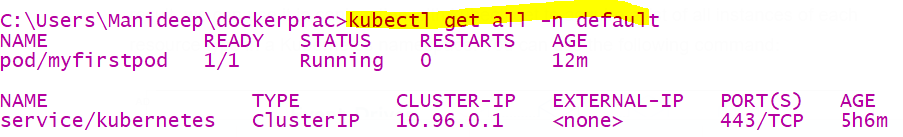
kubectl get all

this will get all objects present inside all namespaces



1. Get all objects inside that namespace

kubectl get all –n <namespace name>



1. Extract yaml file from running pod

This will not create any pod, it will just generate an yaml file with content, so that if u execute it pod/..will be created

kubectl run podhyd-redis –-image=redis –-dry-run=client –o yaml >pod.yaml

this will create a new pod with name “podhyd-redis”

kubectl run nginx --image=nginx --dry-run=client -o yaml > pod.yaml

*# Generate spec for running pod nginx and write it into a file called pod.yaml*  
ex:-

C:\Users\Manideep\dockerprac>**kubectl run redis --image=redis --dry-run=client -o yaml >createredispod.yaml**

1. CRUD Replicaset
2. Update image in running pod
3. Just “**kubectl edit pod <pod-name >”** --then u will get editor ,update the image name,if u update like this

Only that particular pod will be recreated with new image and new container

1. If u are using replicaset edit as “**kubectl edit rs <replicaset name>** ” update the image name,
2. If u are using deployment object edit with “**kubectl edit deployment <deployment-name>** ” update the image name,

Now if u update the image name in deployment object , all the nodes under that replicaset will be destroyed and recreated with latest image

Kubernetes objects

Example objects:- pod,development,replicaset,replicationcontroller,service,horizantalpod,autoscaler

Api versions- <https://blog.knoldus.com/what-is-apiversion-in-kubernetes-part-1/>

|  |  |
| --- | --- |
| Kind | apiversion |
| ComponentStatus | v1 |
| ComponentStatus | v1 |
| Namespace | v1 |
| Node | v1 |
| PersistentVolumeClaim | v1 |
| Pod | v1 |
| PodTemplate | v1 |
| ReplicationController | v1 |
| ReplicaSet | apps/v1 |
| ResourceQuota | v1 |
| Secret | v1 |
| ServiceAccount | v1 |
| Service | v1 |
| ControllerRevision | apps/v1 |
| StatefulSet | apps/v1 |
| CronJob | batch/v1beta1 |
| DaemonSet | apps/v1  extensions/v1beta1 |
| Deployment | apps/v1 |

## POD object

every pod will have separate ip address and every pod should be attached with some label to identify

The problem with object pod is –u cant replicate it, we don’t have the option to replicate the pod object

Replicas is only with respect to pod, not with respect to containers ,

1 pod can have more than 1 containers, if ur apps are tightly dependent create all those apps in 1 pod

apiVersion: v1

kind: Pod

metadata:

name: myfirstpod

## this is the name of the pod and it will create the pod with some labels also

spec:

containers:

- name: nginc1

image: nginx:latest

2 containers in same pod

2 containers in same means 2 applications in same laptop/same os, those 2 apps can communicate using localhost.

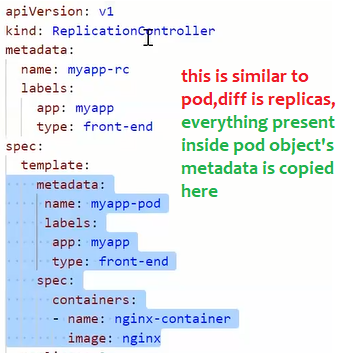
ReplicationController Object

ReplicationController Object ensures that always desired number of replicas/pods are present, even if u delete one replica among 3 desired replicas, immediately new one will be created

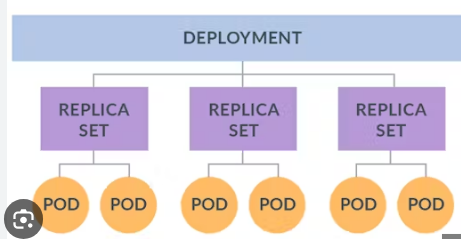
The only difference between pod and replication controller is ,

if u use pod object, u cant replicate, whereas when u replication controller we can create the replicas of a pod,

Remember pods are controlled by ReplicationController object



ReplicaSet object



This Replicaset is a wrapper around group of pods/replicas, this is also to create replicas of a pod **To ensure always desired number of pods are always there,**

Remember pods are controlled by ReplicationSet object or Replicaset object

This object is also same as object “ReplicationController” will also ensure desired number of pods are present

# this is the real object who will control the pods directly, so to control the pods it should know which pod to pick right, this replicaset will pick based on the pod label ,here we should mention which pod to pick using pod selector

|  |  |
| --- | --- |
|  |  |
| kubectl get rs  kubectl get replicasets | To get all replicaset objects type below |
| Kubectl describe rs <replicaset –name > | To see all label of that replicaset  And to check what is the selector ,I mean which labelled pods it will pick…u can see all this information 2) to see who is controlling this replicaset-generally replicasets will be controlled by Deployment objects |
| To delete a replica set | kubectl delete replicaset <replicaset-name> |
| Scale up or down a replicaset | kubectl scale rs <replicaset-name> --replicas=5 |
| If u want to see who is controlling the pod  “kubectl describe pod <pod-name> ”,  If u want to see who is controlling the replicaset ,  Generally replicaset will be controlled by deployment object | Kubectl describe replicaset <replicaset -name>  There is a field called “controlled-by” :” |

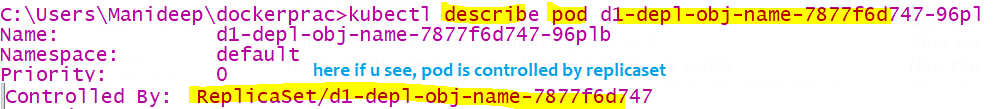
The difference between “ReplicationController” and “ReplicaSet” is the object ”ReplicaSet” object will have an additional

Selector to select the existing running pod

**Note:-as replicaset manages pods,if we mention replicaset as 3,if any pod got deleted, it will be created automatically**

**If u want to delete all the pods, 1st u should delete the parent-ReplicaSet/ReplicationController object who is controlling that**

**Here pods are controlled by parent ReplicaSet, so if u want to delete all pods then we should delete this replicaset first**

****

**If u want to delete all nodes under replicasset-u cant pod by pod because it will always ensure 3/5/desired number of replicas are present.**

Solution:-if u want to delete all nodes, better delete the replicaset first then only all pods will get deleted

|  |  |
| --- | --- |
| Pod name | replicas |
| Pen ( pod selector:oldStock) | Current replicas-3 |
| Pencil (pod selector:oldStock)  Whereas if we mention desired replicas as 5-  Diff pod with same selector (oldStock) already exists ,  hence eventhough u gave desired replicas as 5,it will create only 2 replicas,because 3 existing replicas as already present with same selector | Current replicas-0  After exec,  New replicas will be 2 because same selector 3 already exists ,so only 2 will be newly created |

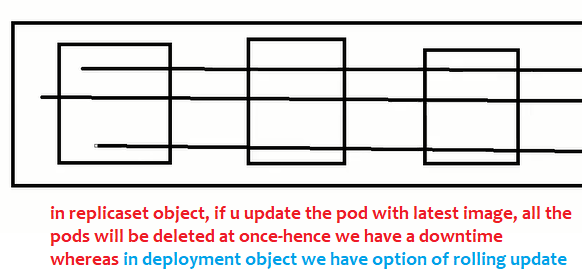
if we mention replicas as 3 with pod selector (stock: fresh)if an another pod is already running with same name

It will consider that criteria, Now this pod will have only 2 replicas

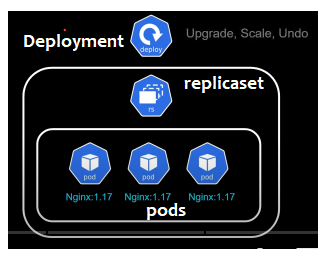
Deployment object

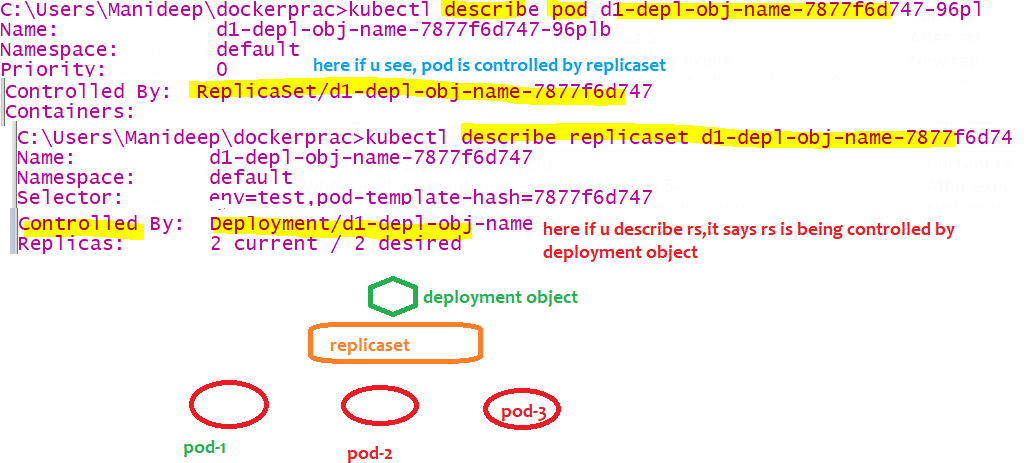
Deployment is an object which is namespace scoped,means this object can present across all the nodes

The main advantage of Deployment object is , when this object is created when that pod is running we can update the image dynamically, so slowly 1 pod will be deleted and new pod will be created, again 1 new pod will be created and 1 old pod will be deleted, so there is no downtime using the concept of rolling update,incase of other objects all old pods will be deleted first and new pods will be created with old objects we have downtime,in this Deployment object we don’t have that downtime



|  |  |
| --- | --- |
| kubectl describe deployment <deployment-name>  kubectl describe deployment d1-depl-obj-name | When u describe u will see all the labels of deployment  How many replicas, what is the image and pod selectors..and deployment strategy=rolling update/ recreate everything and  And to check what is the selector |
| kubectl delete deployment <> |  |
| kubectl get deployments  To get all the deployment object names |  |
| kubectl edit deployment <deployment object name>  kubectl edit deployment d1-depl-obj-name | To update the image name/update replicas count -U will get the script/deployment object script/source code in new editor, So that u can edit image name |
| Get current source code of deployment object in yaml –now we wil get source code or yaml file with which that deployment object is created | kubectl get deployment <your deployment object name> -o yaml  kubectl get deployment d1-depl-obj-name -o yaml |
| Scale the replicas of a deployment-to scale up or scale down | kubectl scale deployment <deployment> --replicas =5  kubectl scale deployment d1-depl-obj-name --replicas=8 |
| Rollout the deployment to older image version | kubectl rollout undo deployment/<your deployment object name>  kubectl rollout undo deployment/d1-depl-obj-name |





Like a Pod wraps around container and containers

Replicaset object wraps around group of pods-means replicaset will manage the pods

**Deployment object is a wrapper which wraps around a replicaset object,it manages the replicaset,**

**Replicaset which inturn manages the pods**

**Deplyment object restores replicaset (means if replicaset got delete,deployment obj will create it)**

Replicaset restores pod(means of any pod under that matched label replicaset got deleted,rs will create pod)

If Higher level deployment object is created then all lower level objects –replicasets,pods are created automatically

The main advantage is when the replicaset is running when u update the image version in YAML file,

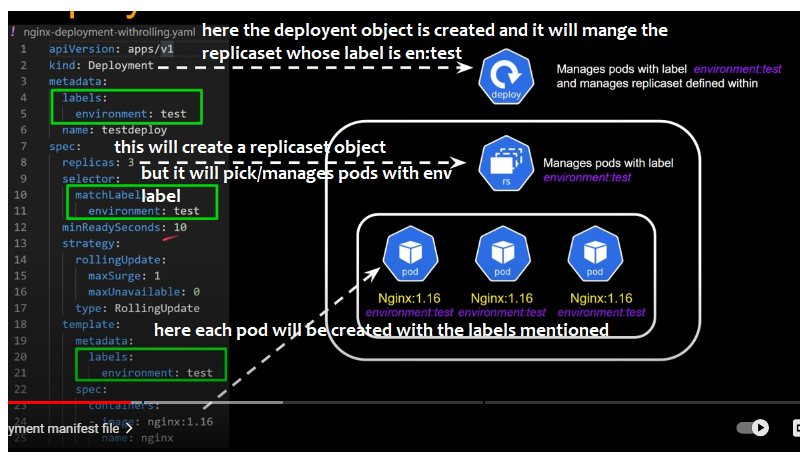
Deployment provides declarative updates for pods it will change the actual state to desired state in controlled rate

Note:-

Every object should have a label, else it will be difficult for wrapper /up level object to pick

eX:- here deployment object is a wrapper around replicaset object, deployment object pick/manage the object with the

specified selector



Note:label is very very mandatory to identify /manage the objects

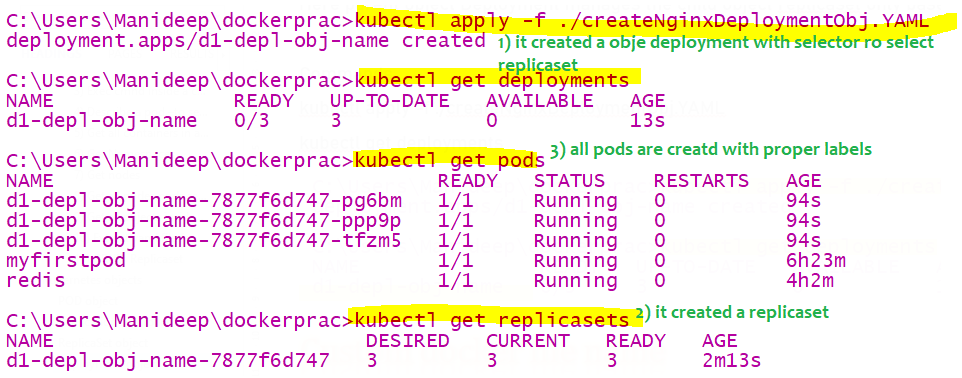
Here parent object Deployment manages the child object replicaset only based on the label environment:test

If label is not there, deployment obje I think it cant select pods/vms try this. By updating a nginx version and check

Commands

kubectl apply -f ./createNginxDeploymentObj.YAML

kubectl get deployments

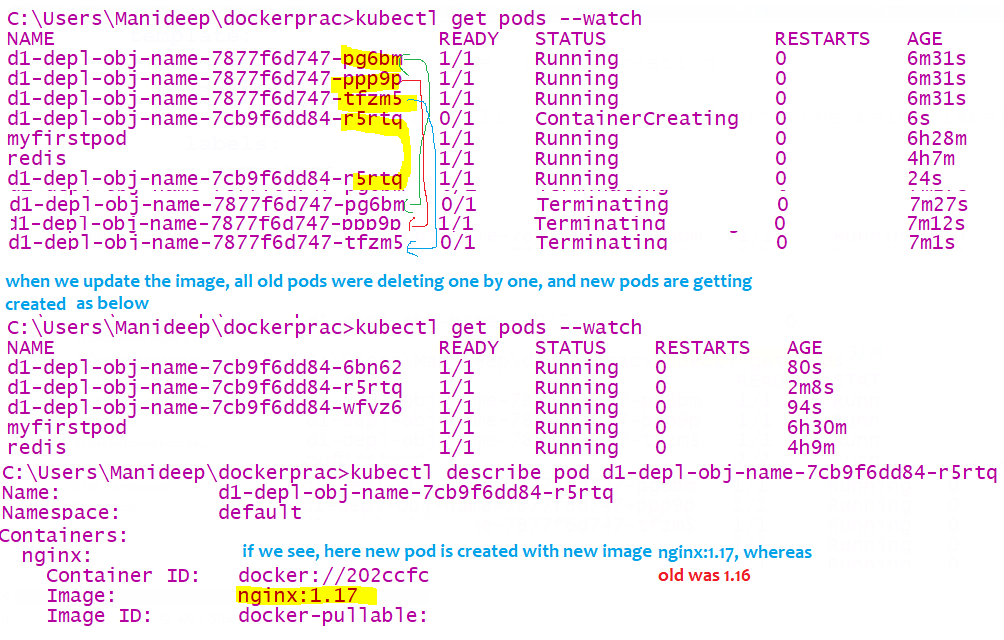


What is the use of deployment object:- when container is running we can update the image name

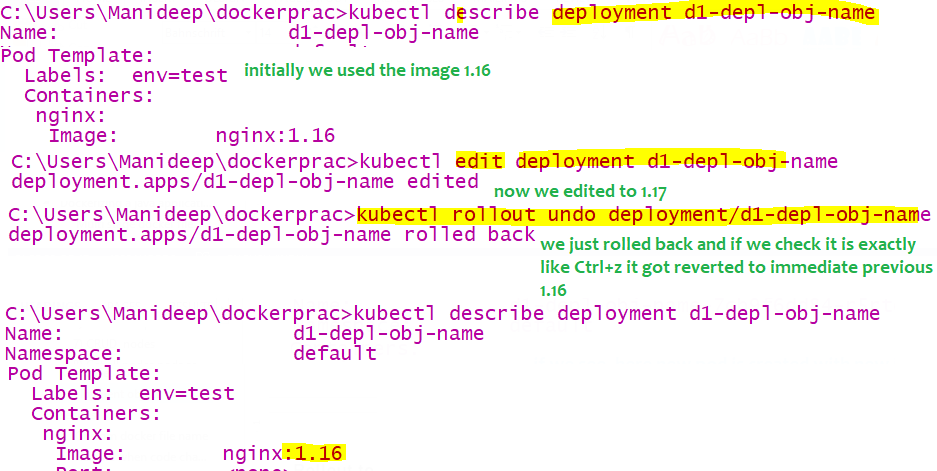
Earlier image name was1.16 now we update the image name to 1.17,pods Auto destroyed and newly created

The beauty with this is no downtime, all existing pods are not deleted at once, whereas in case of other objects

Like “ReplicaSet”,”ReplicationController” all pods will be deleted first and will be created everything newly with downtime



Rollout to older deployment



#### Example code:-

Refer attached file



Daemon sets obj

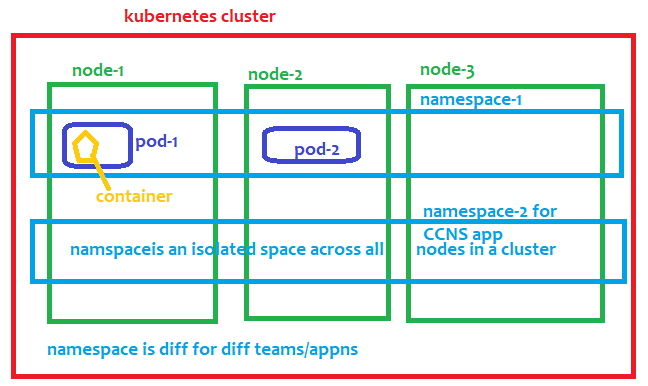
Initially we have 3 nodes in a cluster and we have 1 pod on each node, later if we add the nodes to the cluster if u want to

create the pod in each and every node, then we should use daemon set object

Namespace object

It is an isolated space which is available across all the nodes in entire cluster (or across the nodes)

Namespace will span across the nodes



Namespace provides a mechanism for isolating group of resources or objects (pod,deployment,replicasets,secrets)

in same cluster

If 2 different teams application if u want to isolate then we will create a namespace (1 for each team)

1. Name of the resources/objects-pod/Deployment should be unique within a namespace

Ex:- secrets will be stored in namespace level, if 2 apps are in same namespace, all will/can access those secrets

Some objects in the kubernetes are namespace scoped objects like POD,Deployment,

Whereas some objects will be deployed at cluster level

1. Deployment object is a namespace scoped object,

Within the same name space, the name of the deployment must be distinct, and in another namespace the earlier names can be present.

Namespace should be different for each and every team, generally access will be given to the entire namespace

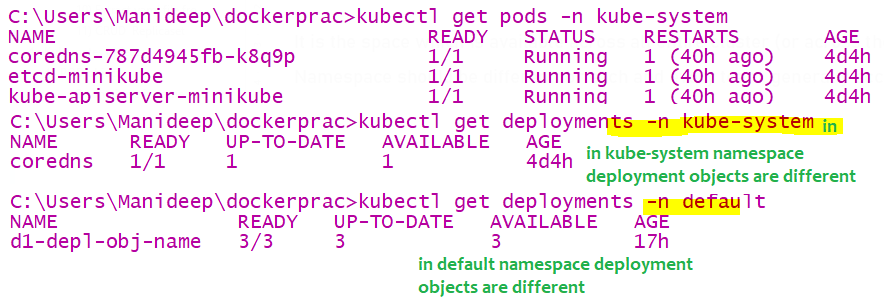
If both team apps are in same namespace, I have the ability to delete their pods.. if 2 teams have same namespace

1 team app may consume more CPU, RAM resources another team may not. .so to avoid conflicts better to have different namespace for all application

Some objects will reside in namespace-also called as namespace scoped objects-POD,Deployment

These 2 objects will reside in namespace

By default until u don’t set the namespace, u will be in default namespace



1. CRUD Namespaces

|  |  |
| --- | --- |
| To get all namespaces | kubectl get ns  kubectl get namespaces |
| Create a namespace | kubectl create namespace <name of the namespace>  kubectl create namespace maninamespace |
| To get pods from diff namespace being in diff namespace | kubectl get pods -n <name of the namespace>  kubectl get pods -n kube-system |
| To get all objects from that namespace | kubectl get all -n <name of the namespace>  kubectl get all -n kube-system |
| In declarative approach if u want to create a object in particular ns, | Under metadata, add a field called namespace in YAML file, so that the file will be created in that namespace |
| To set a namespace to current context | kubectl config set-context –-current -–namespace=<ns name>  kubectl config set-context --current --namespace=maninamespace  kubectl config set-context --current --namespace=default |

Secrets

Secrets is a namespace scoped object, Once u created secrets object, all those secrets will be stored at namespace

Custom docker file name

So far , we worked without code, I mean we worked only with docker files, now we will start working with

code+docker files

for that refer project num “65-creating”,” 67-starting” these proj will be there as part of my repo

1. Download nodejs software and test installation status using command “node –v” in command prompt
2. Create a sample node js samle project with command

“npx create-react-app frontend”

1. The above command wont work in Ubuntu console, so create a project in windows folder and move

That project to Ubuntu [\\wsl.localhost\Ubuntu\home\manideepvv](file:///\\wsl.localhost\Ubuntu\home\manideepvv), type “\\wsl$”

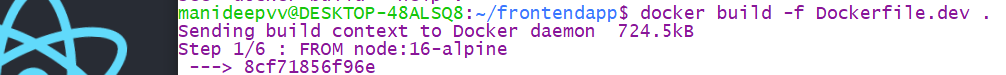
1. In that create Dockerfile and add node related files like “index.js”,”package.json”
2. Since we are creating prod ready proj, instead of creating file named“dockerfile” created “Dockerfile.Dev”

For dev environment

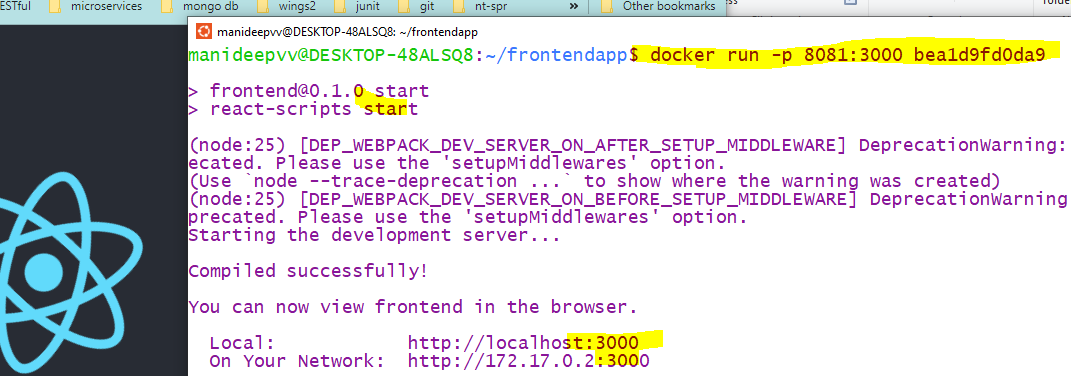
docker build –f <your custom docker file name> .

ex:- docker build –f Dockerfile.dev .

in above –f stands for custom file name and “.” refers to current directory



Run that container and map with system port



### Rebuild when code changes are made

manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker build -f Dockerfile.dev .

Step 6/6 : CMD ["npm", "run", "start"]

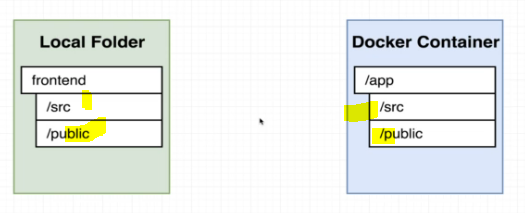
---> Running in 7a325cea22fc

Removing intermediate container 7a325cea22fc

---> 7ec45d18a115

Successfully built **7ec45d18a115**

manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run -p 8081:3000 **7ec45d18a115**

when we rebuild again separate space is created

to that space all those files should get copied

Dynamic code refresh

Refer proj 70,71,72(docker compose yml file proj-best)

In production we will never create a docker volume,bec we don’t want dynamic code refresh

In java we will build and we will get a jar file and we will deploy that jar file in jvm ,

Every time when we make a code change, we have to rebuild and deploy that latest jar in jvm/any server

Way 1:-

Here in docker world also, generally we have to do build every time and run that new continaer,

Way 2:-

Instead of building every time and getting the new image file we can go for hot code replacement

While running that container we cant point those requests to source code

So without rebuilding the image we can get the latest changes every time.

docker run -p 8080:3000 -v /home/node/app/node\_modules -v $(pwd):/home/node/app <image id/jar name>

here –p means port, first port is map with system port and 2nd one is docker port

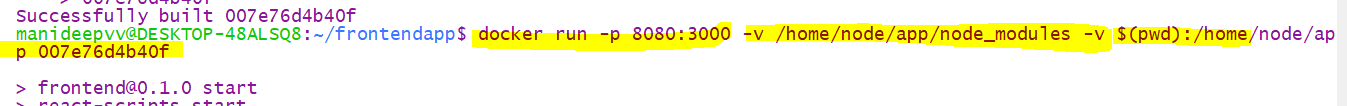
-v means =volume

/home/node/app is the default folder created inside a container

–v /home/node/app/node\_modules says don’t map this folder called node\_modules, this folder will be there in the container, if any request comes to this let it be, don’t map to a folder outside of a container

And when request comes to this folder in the container, don’t map this, because we have deleted node\_modules

**$(pwd) :/app** means



Map a folder (/app) present inside a container to a folder outside a container

If anytime container reaches /app directory, its going to redirect to the pwd

what ever is there in pwd-present working directory map to app folder present inside a running container

docker run -it -p 3000:3000 -v **/home/node/app/**node\_modules -v ~/frontend:**/home/node/app** USERNAME:frontend

Short hand docker compose

In docker file below are the commands

===============

RUN mkdir -p /home/node/app

WORKDIR /home/node/app

Way 3:- using docker compose command

**Alternate to big run command**

The above run command is big, so even for 1 container also we can go with “docker-compose.yml” file

Refer “1.1.comments-dynamic code refresh .YML” in this directory git hub

version: '3'

services:

web: ## this says create a container called web

build: . ## says build using the docker file present in this current directory

ports:

-"8080:3000 ## here the application is running inside docker port 3000

map the system port 8080 to docker container port 3000

so when we hit 8080 in our browser this is mapped to container port 3000

volumes:

- /app/node\_modules

- .:/app # this says when request comes to app folder inside a container that will be

#mapped to current directory

The above will face issues, because build specifies “.” Means current directory

Refer proj 72

Here context specifies

Where should we pull the information like files and folders

version:'3'

services :

web:

build:

context: .

dockerfile:Dockerfile.dev ##because we wrote the commands in "Dockerfile.dev" instead of dockerfile

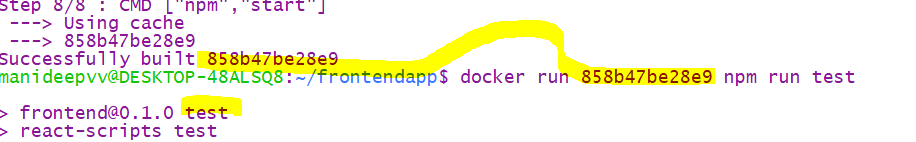
Running tests without docker compose

Here also we will run those tests in a new container

docker run <image-id> <our new command>

remember with image we can create container only, so in above step new container will be created

docker run 858b47be28e9 npm run test



Create a new space /container & run this additional command [npm run test ] in that space

But ,u can create space only when we have image

docker run -it <image name> <command name>

docker run -it 858b47be28e9 npm run test

run command will create a new container and execute this command

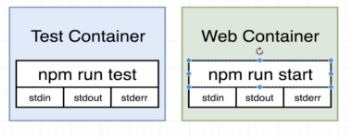
docker run -t manideep:node npm run test

Running tests with docker compose

Refer project 76(it has both for linux, for wsl) ,“ 2.0.seperate container to run the tests.yml”

It says create a separate container and change the start up command

Like below in docker-compose.yml file ,create a separate container called tests

here 2 containers-both test container,web containers are seperate

tests:// says create a new container called tests

stdin\_open: true

build:

context: .

dockerfile: Dockerfile.dev

volumes:

- /home/node/app/node\_modules

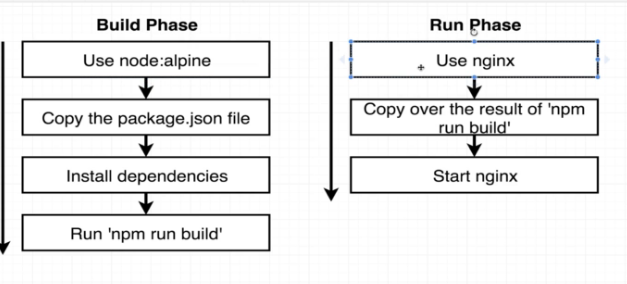
- .:/home/node/app ##when any request comes to this folder ,map to pwd

command: ["npm", "run", "test"]

## Implementing Nginix –Multi step builds

Nginix is a server same like tomcat, it’s a web server whereas tomcat is an application server

Refer prj number 80 wsl version for windows



So far our base image is only-1 , that is node:alpine version

But now we need 2 different softwares- we need nginix also

So we will follow multi step process, now the plan Is lets go with phase-1 ,execute “npm run build” command

And sample docker file and we will use the output of step-1 /phase-1 to the new phase

Refer “3.multi docker steps,,node,nginx.txt”

FROM node:16-alpine as builder #here builder is the phase name

WORKDIR ‘/app’ ## set the current dir to /app,so that here after all commands will be executed in this dir

# once proj is built ,build folder also will be created here

Copy package.json .

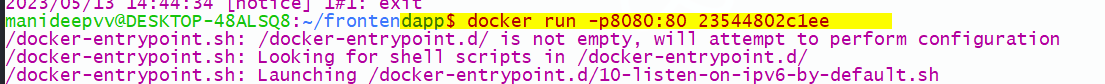
RUN npm install

COPY . .

RUN npm run build

FROM nginx

COPY –from =builder /app/build / usr/share/nginx/html



Nginx always runs on port 80

Once the image is created, ran that using image name

docker run <image name>

## Dockerizing a java application

Sample Docker compose file-create container and install maven,jdk

FROM openjdk:8-jdk-alpine

RUN apk update && apk add bash

WORKDIR /app

COPY . .

##To install maven

RUN wget https://mirrors.estointernet.in/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz

RUN tar -xvf apache-maven-3.6.3-bin.tar.gz

RUN mv apache-maven-3.6.3 /opt/

ENV M2\_HOME='/opt/apache-maven-3.6.3'

ENV PATH="$M2\_HOME/bin:$PATH"

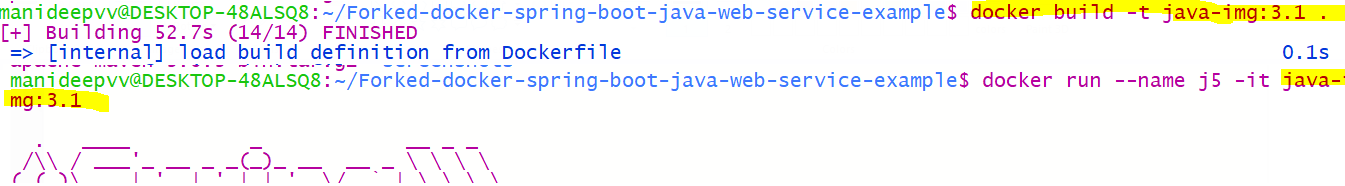
RUN export PATH

RUN mvn clean install

EXPOSE 8080

# Run jar file when the container launches

CMD ["java", "-jar", "target/docker-java-app-example.jar"]



The main problem with this approach is ,into main container maven software is also installed and the image is getting bulk

So to avoid it , create a multi stage docker file, where we know multiple intermediate containers will be created and from those multiple intermediate images will be created, so among those intermediate containers, in 1 of them install maven

And generate jar file, when u create a image from that container, in that final image done include maven software

So that image will be light weight

Multi stage docker file

The main pri

When u use this command, maven will be installed and path will be set automatically

RUN apt-get update && apt-get install maven –y