Openshift Container platform (OCP)by Redhat

Openshift container platform = Openshift

Reference links

|  |  |
| --- | --- |
| All object definitions are available here | <https://docs.openshift.com/container-platform/3.11/rest_api/apps_openshift_io/deploymentconfig-apps-openshift-io-v1.html> |
| <https://gitlab.com/practical-openshift/labs> | Clone the code of udemy trainer here |
| Quay.io  This is by redhat | Image repository (same like docker repository, here also we can store all the image ) |

Installation

Install docker for desktop software

Install WSL software (windows subsystem for linux)

Then to test run a command called “docker run hello-world” here hello-world is the image name in “windows power shell” and docker desktop should open correctly

Unlike minikube u cant install the Openshift in ur windows machine, bec it needs 9GB RAM, 30 GB harddisk, 4 virtual cpu, so that’s why better use the Redhat developer sand box version

|  |
| --- |
| 1. Create a dev free cluster in Redhat- then few namespaces also will be created in the cluster |
| 1. Download oc command line tool <https://console-openshift-console.apps.rm2.thpm.p1.openshiftapps.com/command-line-tools>   Or login to oc and right click on ? and select command line tools – download oc tool into desktop and add that to path environment variable |
| 1. To see all namespaces <https://console-openshift-console.apps.rm2.thpm.p1.openshiftapps.com/topology/all-namespaces?view=graph>   or go to topology u can see all the namespaces |

Similarities and differences

Same like kubernetes

* Openshift also have a cluster (bunch of server machines)
* Openshift also have objects/resources (among 15, 10 are Openshift exclusive, 5 are common in both)

Openshift architecture & Terminologies

Ocp primary job is to run containers

Here host is nothing but some worker node

|  |  |
| --- | --- |
|  |  |
|  |  |

Here plane means category, in ocp total servers are categorized into 2 –

Control plane- the servers under this category are responsible for running core Openshift processes - REST API, Data persistence, monitoring processes

Data plane – the servers in this category are responsible for running our custom apps like java programs

Developer will interact with REST api present in control plane

Containerization

Container is nothing but some isolated process space inside a hard disk where our app can run

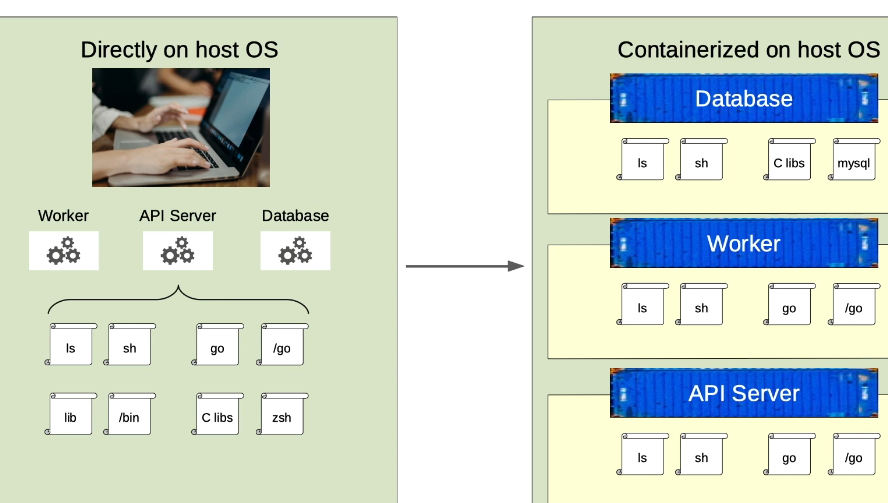
Why containers are needed?

Ex:- if spring needs java 11 , and python needs java 17 , in same machine or os u can’t have 2 java versions,

So if u use containerization, an isolated space will be created inside a hard disk

Container is bundled with all of its dependencies

Docker daemon is something that runs container



OS uses project object to divide applications/ segregate apps

User quota represents upper bound of RAM,cpu and all other resources , for that quota it will reserve the space

Most interesting things

Withut writing single object definition (if u just know image name), we can deploy that image with single command as below

oc new-app <image name with repository> --as-deployment-config

image

oc new-app <image name with repository> --as-deployment-config

oc new-app quay.io/practicalopenshift/hello-world --as-deployment-config

this is the easy image -which doesn’t need any properties to start – means we don’t need to pass any environment variables/ configmap to deploy this image

image is the blue print from which container will be created

|  |  |
| --- | --- |
| to push image to remote repo | docker push quay.io/$REGISTRY\_USERNAME/private-repo |

Commands

|  |  |  |
| --- | --- | --- |
| oc explain pod <pod-name> | * To see full pod definitions or full code of that object according to specification with values   then use “oc get -o yaml pod <pod-name>” or “oc get pod <pod-name> -o yaml”  oc get output in yaml  oc get -o yaml pod <pod-name>  oc get -o yaml route/hello-world  oc get –o yaml <object type>/< obj- name >  oc get -o yaml cm sync-workspaces-config | oc describe pod <pod name>   * if u want to see the summary of that object/ and important points then use describe command |
| it will give only spec- like structure – not real values- it says only allowed values | it will give real data ex:- 160 lines  this will give even more info | it will give real data ex:- 80 line |

Important commands

Note multiple objects can have same name

ex:- pod obj name, route object name, service object name all these object can have same name, so while doing

oc get –o yaml <obj type>/ <obj name> you should tell the obj type also

|  |  |
| --- | --- |
| if u want to know which credentials or secret is being used by os to pull image then check the service account “default” to see to which secret object this service account is pointing |  |
| to go inside pod and to see env variables for that pod | oc rsh <pod name> 🡪 with this u can go inside pod  to see env variables type “env” |
| To see full code of that object  oc get secret s1 -o yaml – here –o means output  oc get secret <secret name> -o yaml //this –o yaml can be either placed In front or back  oc get <object type> <object name given by us> -o yaml | oc get output in yaml  oc get -o yaml pod <pod-name>  oc get -o yaml route/hello-world  oc get –o yaml <object type>/< obj- name >  oc get -o yaml cm message-map // Here cm means config map  oc get istag hello-world:latest -o yaml  oc get -o yaml cm sync-workspaces-config |
| How to write kubernetes spec files | use tools to generate the definitions  or create object using commands and execute this command to see def  “oc get -o yaml cm message-map” |
|  |  |

|  |  |
| --- | --- |
| Explain command vs  Describe commands | To see all commands just type “oc” it will only show those commands  There is some small difference between explain and describe  Explain – is to see the specification like what fields that object have  describe- this is to see the properties of object after its creation , like properties of pod such as –cpu, image name…  , like “desc table tablename” in database |
|  | oc get <any object> |
| To see doc /explanation of any os/kubernetes resources | oc explain <any ocp / kub object /resource type>  Ex:- oc explain pod or oc explain imagestream   |  | | --- | | You can even see full specification  Every object have a field called spec, that’s what we are fetching here  If u see the object def by hitting “oc explain <any ocp object>” it will show all the fields infor | | E:\study related\pods>oc explain pod  KIND: Pod  VERSION: v1  DESCRIPTION:  Pod is a collection of containers that can run on a host. This resource is  created by clients and scheduled onto hosts.  FIELDS:  apiVersion <string>  APIVersion defines ..  kind <string>  Kind is a string value ...  metadata <ObjectMeta>  Standard object's metadata. More info:  https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata  spec <PodSpec>  Specification of the desired behavior of the pod. More info:  https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status  status <PodStatus>  Most recently observed status of the pod..  oc explain pod.spec (note,as per above definition, this pod has a field name spec this spec field may not exist in few objects like configmap)  here inside spec we have another variable called “subdomain” to see that definition  “oc explain pod.spec.subdomain” | |  | |
| Login to oc server cluster | <https://console-openshift-console.apps.rm2.thpm.p1.openshiftapps.com/topology/all-namespaces?view=graph>  oc login –-token=<your token here > --server=cluster url  (you can get this command from oc browser login using above url and right top right corner – click on ur pic – select – copy login command)  oc login –u developer  # Uses the pre-configured OpenShift cluster  oc login  # Allows you to log in to any OpenShift cluster  oc login <cluster address>  # Log out  oc logout |
| To see who am I (to get user id) | Oc whoami |

Project level commands

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| To see project  (Here project is nothing but namespace) | |  |  | | --- | --- | | To see all the available projects | oc projects | | To see on which project u are | oc project | | To switch to another project” (like switching to another namespace) | oc project <target project name here> | |  |  | | To create a new project | oc new-project <new proj name here > | |  | oc port-forward pod/lab-pod 8080 | | To see openshift cli /server version | oc version | |
| To create all obj’s in single shot | oc new-app <image name with repository> --name <desired label>--as-deployment-config  ex:- oc new-app quay.io/practicalopenshift/hello-world --name codi --as-deployment-config  Now all objects like pod, rc,dc,service,istag will come automatically - |
| To get all objects In project/ namespace | oc status  or  oc get all  this command will fetch all objects like – pod, replicationController, deploymentconfig, service |
| delete all objects with label-  this kind of deletion is the best way | oc delete all –l key=value  oc delete all -l app=hello-world //Here you should give the label of rc object of dc deployment config  best way to delete all related objects is by using label, since all related objects will have mostly same label, they can be easily deleted   * for me these objects got deleted – pod, replication controller, deployment config, imagestreamtag |
| describe | oc describe pod <pod name>  you can describe any object |
| You can see any object code | oc get -o yaml pod <pod-name>  oc get -o yaml dc <dc-name>  here o means output – you can see all the values |
| you can edit any object - pod, dc,service, rc | oc edit po <pod name> -  oc edit po codi-1-deploy  suppose if you want update the image name in running pod – then you can edit the pod definition and save it |

OS objects

All object

|  |  |
| --- | --- |
| ServiceAccount 🡪 this is also an another os object | All object definitions are available here  <https://docs.openshift.com/container-platform/3.11/rest_api/apps_openshift_io/deploymentconfig-apps-openshift-io-v1.html>    note multiple objects can have same name  ex:- route , pod all these object can have same name, so while doing  oc get –o yaml <obj type>/ <obj name> you should tell the obj type also |

Projects

OS uses projects to group related resources

Here project is nothing but namespace, like how namespace contains all kub resources, here project contains all

|  |  |
| --- | --- |
|  | To see the projects and namespaces  oc get projects  here under project tab u can see all resources |

### Pods

Generally pod means group of whales, in ocp 1 pod is nothing but group of containers, 1 pod can have 1 or more containers

In real world assume container as a whale fish

In real world, too much tightly inter-dependent applications (one can’t live without other) stay in same pod, if pod goes down both will be down, else both will be up na

Additional containers are called side car containers

|  |  |
| --- | --- |
| To see full specification of this pod object like what fields to write in this definition | oc explain pod.spec  oc explain pod.spec.containers.env |
| Create sample pod object  apiVersion: v1  kind: Pod  metadata:  name: hello-world-pod (this will be final pod name u can see in oc get pods )  labels:  app: hello-world-pod  spec:  containers:  - env:  - name: MESSAGE  value: Hi! I'm an environment variable  image: quay.io/practicalopenshift/hello-world  imagePullPolicy: Always  name: hello-world-override  resources: {}  we can create this pod object with this command  “oc create –f pods/pod.yaml”  here –f means file | “oc create –f pods/pod.yaml”  Ex:- oc apply -f ./pod.yaml (here even apply keyword will work)  This command will internally hit the RESTful post api and it will create a pod object in os project  In kubernetes it is kubectl apply –f “path to yaml file” |
| To see available running /stopped pods | oc get pods  oc get po (here po is the shortcut for pods)  To see pod real time updates (we don’t need to refresh- but we can see changing pod statuses)  oc get pods --watch |
| Describing a pod – to see | oc describe pod <pod name>  here you can see pod details such as 🡪 how many containers in that pod,  for each container u can see – image name, env variables, label….and that pod cpu details |
| to see the pod definition | oc get –o yaml pod <pod-name> |
| to edit pod definition | oc edit pod <pod-name> |
| To go into the pod and run a command  Here rsh – remote shell session  “oc rsh” command is to establish a remote shell session with running container, this is to go inside that pod  q- if 2 containers are there with which container it will open session   * when we want to see all env var cfgs for pod then we should go into pod | oc rsh hello-world-pod   * after going inside pod to hit that container “wget localhost:8080” – this will generate a file called “index.html” inside pod to to open that “cat index.html” * if u want to come outside from that pod “exit” * to see all the environment variables of that pod type “env” – this is best feature I liked so that we can see environmental variables for that pod   OpenShift publishes virtual IPs in environment variables inside of containers.   * df |
| To delete the pod | oc delete pod <pod-name>  in the command we should mention the type of resource- bec internally they might have enabled/disabled delete grant for certain object ,  so we must enter resource type |
|  |  |

ReplicationController

replicationcontroller object is the one which controlls the number of pods,

this object job is to ensure always those many pods should be running keep on

|  |  |
| --- | --- |
| To describe rc- to see the number of replicas | oc describe rc <replication controller name>  oc describe rc hello-world-1 |
| to get rc | oc get rc |

Deploymentconfig (DC)

Deployment config is nothing but collection of pods, this object brings extra configuration to your pods

In real time we will use this definition instead of pod definition, this is like parent object for pod object

Ex:- if u directly delete pod, sometime pod wont get deleted because in deploymentconfig we would have set the replicas as 2, (oc explain deploymentConfig.spec.replicas)

so in that case delete this deploymet object , so that pods will get deleted automatically

While running commands instead of deploymentconfig , u can always use dc

This DC object will create the pods

|  |  |
| --- | --- |
| To see full specification | oc explain deploymentconfig  or  oc explain dc |
| Deploy application with single command with image name  If u just know image name, with this single command u can create 🡪 pod, ReplicationController,DC, service, image stream tag (istag)  ( without writing the pod definition) | oc new-app <image name with repository> --as-deployment-config  without label also we can create- but its recommended to create with label  oc new-app <image name with repository> --name <desired label>--as-deployment-config  oc new-app quay.io/practicalopenshift/hello-world --as-deployment-config  --this label will be applied to all objects and this name will be taken as dc object name & service object name |
| Deploying from git repo –  this will internally take docker script file which must be present in git repo – so that it will 1st create the image and then it will deploy that image | oc new-app https://gitlab.com/practical-openshift/hello-world.git --as-deployment-config |
| to see all the dc object properties like replicas | oc describe dc <dc name>  oc describe dc hello-world |
| to see all deployment config | oc get dc |
|  | oc delete dc/dc-name  or  oc delete dc dc-name |
|  | oc rollback dc hello-world |
|  | oc rollout dc hello-world |

When u run this command below will be the log

--> Found container image 34b5ac8 (4 years old) from quay.io for "quay.io/practicalopenshift/hello-world"

\* An image stream tag will be created as "hello-world:latest" that will track this image

\* This image will be deployed in deployment config "hello-world"

\* Port 8080/tcp will be load balanced by service "hello-world"

\* Other containers can access this service through the hostname "hello-world"

--> Creating resources ...

imagestream.image.openshift.io "hello-world" created

Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+

deploymentconfig.apps.openshift.io "hello-world" created

service "hello-world" created

--> Success

Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:

'oc expose service/hello-world'

Run 'oc status' to view your app.

Advantage of labelling

Same image you can run 2 instances by changing the label

ex:-

|  |  |
| --- | --- |
| label as codi | oc new-app quay.io/practicalopenshift/hello-world --name codi --as-deployment-config |
| label as mani | oc new-app quay.io/practicalopenshift/hello-world --name mani --as-deployment-config |

### Configmap

This config map is really like a map

In real time there can be many pods & many map (object) in the namespace- we should establish a link between those 2 objects

Every pod points to some config map, in pod or deployment object definition we can see which pod is pointing to which config map

|  |  |
| --- | --- |
|  | In real time many pods can use the same config map  we cant use this object for storing secrets,  to store secret we should use secret object  this max limit is 1MB |
| create config map cli | oc create configmap message-map --from-literal MESSAGE="Hello From ConfigMap" |
| create configmap from file | oc create configmap <cm- name> --from-file=Message.txt  not real time standard |
| create configmap from directories |  |
| sample spec | apiVersion: v1  data:  MESSAGE: ORAYYA  kind: ConfigMap  metadata:  creationTimestamp: "2025-01-06T12:54:37Z"  name: message-map  namespace: vv-manideep-dev  resourceVersion: "115848213"  uid: d4530e85-027f-438d-883a-74200c6e5f24 |
| setting this config map to deployment config object | oc set env dc/hello-world --from cm/message-map  In real time nobody will execute this command instead in deployment object definition itself  we will include this config map reference |

Secret

|  |  |
| --- | --- |
| secret object definition | 1. **apiVersion: v1** 2. **data:** 3. **MESSAGE: YmFzZTY0** 4. **kind: Secret** 5. **metadata:** 6. **name: lab-secret** 7. **type: Opaque** |
| command to create secret   * the value stored in secret object, is base 64 encoded | oc create secret <secret type> <secret-name> --from-literal message =”abcd”  ex:- oc create secret generic s1 --from-literal message="game changer"  ex:- oc create secret generic s4 --from-literal ikea="broadridge" |
| setting the secret object to deployment object | oc set env <to object dc><dc name> --from secret <secret name>  oc set env dc <dc name> --from secret <secret name>  oc set env dc/hello-world --from secret/s4 |
| Creating a secret with all details to pull image from image repository –  this secret is mandatory because this secret contains all the details to pull the image   * creating a secret with all details with your desired name is not enough * you should tell os to use our particular secret (bec in our ns we may have many secrets) | oc secret <secret-type> <secret-name> <values>  oc create secret docker-registry \  demo-image-pull-secret \  --docker-server=$REGISTRY\_HOST \  --docker-username=$REGISTRY\_USERNAME \  --docker-password=$REGISTRY\_PASSWORD \  --docker-email=$REGISTRY\_EMAIL |
| linking secret to pull the image – this is like we are instructing os to use this particular credentails present in this secret to pull the image – like while pulling use this secret | oc secrets link <service account name> <secret-name-which-has-all-details to pull image>  --for=pull  ex:- oc secret link default demo-image-pull-secret --for=pull |
| after setting the secret, the pod definition will look like it referring to the secret  so that after mapping the pod will point to secret object | spec:  containers:  - env:  - name: IKEA  valueFrom:  secretKeyRef:  key: ikea  name: s4  image: |
|  |  |

serviceAccount

Service account contains the details /link to the secret object

There are lot of sercrets present in namespace

If u want to know which credentials or which secret object is being used by os to pull image then check the service account “default” to see to which secret object it is pointing

|  |  |
| --- | --- |
| Get service account  oc get serviceaccount  E:\2.Openshift>oc get serviceaccount  NAME SECRETS AGE  builder 1 9d  default 1 9d  deployer 1 9d  pipeline 1 9d  Here there is a service account named “default” this is pointing to a secret named  “default-dockercfg-spx7t” | E:\2.2.Openshift>oc get serviceaccount default -o yaml  apiVersion: v1  imagePullSecrets:  - name: default-dockercfg-spx7t  kind: ServiceAccount  metadata:  annotations:  openshift.io/internal-registry-pull-secret-ref: default-dockercfg-spx7t  creationTimestamp: "2024-12-30T03:57:20Z"  name: default  namespace: vv-manideep-dev  resourceVersion: "85757882"  uid: 7c244207-5bfe-4a7a-9bbe-1e4ab68d0967  secrets:  - name: default-dockercfg-spx7t |

Service

|  |  |
| --- | --- |
|  | this service object is like service registry (map) which always maintains the ip address of all pods  always traffic goes to service objects, it will redirect the traffic to the remaining pods  every pod will have some virtual id called internal ip  pods are very short lived, so ip address keeps on changing, all pods ip address will be stored in service object |

|  |  |
| --- | --- |
|  | oc get service or oc get svc |
|  | oc delete svc/pod-name |
| create service for pod  E:\pods>oc expose --port 8080 pod/hello-world-pod  service/hello-world-pod exposed | oc expose --port 8080 pod/hello-world-pod |

Routes gives external DNS name to a service

Route

Route will create/expose a dns name/host name that clients will use ex DNS name:- [www.google.com](http://www.google.com) this is a dns name like this we will get dns name when we create a route

Only with dns name/url name we can hit URL from browser Na so route object is mandatory,

Only with this route we can access application outside of openshift cluster, route object will create a hostname & url for your app

|  |  |
| --- | --- |
| if u expose pod u will get service object  if u expose service u will get route object in ocp namespace | oc expose --port 8080 pod/hello-world-pod  oc expose svc <service name>  E:\pods>oc expose svc hello-world-pod  route.route.openshift.io/hello-world-pod exposed |
| Route is a separate object in ocp  oc extenal interface is called a route – if a front end app wants to hit pod , it needs ROUTE object  routes gives an external dns name to a service  if u create a route, then only we can access that app from chrom using url  E:\\pods>oc get route  NAME HOST/PORT PATH SERVICES PORT  hello-world-pod hello-world-pod-vv-manideep-dev.apps.rm2.thpm.p1.openshiftapps.com hello-world-pod 8080  see we can see yellow color host name/ dns name / url |  |
| We can see which route object is pointing to which service object  E:\pods>oc get -o yaml route hello-world-pod  apiVersion: route.openshift.io/v1  kind: Route  metadata:  annotations:  openshift.io/host.generated: "true"  creationTimestamp: "2025-01-06T10:34:56Z"  labels:  app: hello-world-pod  name: hello-world-pod  namespace: vv-manideep-dev  resourceVersion: "115446412"  uid: 02ea32a4-9df0-42e0-9613-9afa0d742a40  spec:  host: hello-world-pod-vv-manideep-dev.apps.rm2.thpm.p1.openshiftapps.com  port:  targetPort: 8080  **to:**  kind: Service  name: hello-world-pod  weight: 100  wildcardPolicy: None | if u see the to object , there we have service object details  meaning this dns host name is getting mapped to this particular service object (we can see the service object name also)  From definition  host: hello-world-pod-vv-manideep-dev.apps.rm2.thpm.p1.openshiftapps.com  this host name is called dns name with which we can hit    see here I was able to hit this dns name/ host name from browser |

ImageStreamTag

Here ImageStream & ImageStreamTag are 2 objects in openshift

|  |
| --- |
| E:\2.2.Openshift>oc get is  NAME IMAGE REPOSITORY TAGS UPDATED  codi default-route-openshift-image-registry.apps.rm2.thpm.p1.openshiftapps.com/vv-manideep-dev/codi latest 6 days ago  hello-world default-route-openshift-image-registry.apps.rm2.thpm.p1.openshiftapps.com/vv-manideep-dev/hello-world latest 3 hours ago  ## Here you can see the tag as latest, we can run command to change the tag also – but what is the use? |
| E:\>oc get istag  NAME IMAGE REFERENCE UPDATED  codi:latest quay.io/practicalopenshift/hello-world@sha256:2311b7a279608de9547454d1548e2de7e37e981b6f84173f2f452854d81d1b7e 5 days ago  hello-world:latest quay.io/practicalopenshift/hello-world@sha256:2311b7a279608de9547454d1548e2de7e37e981b6f84173f2f452854d81d1b7e 47 minutes ago |

**Imagestream tag means – tag means version.**

In OpenShift, both ImageStreams and ImageStreamTags are crucial for managing container images, but they serve distinct purposes:

**ImageStream:**

* **Container Image Repository:** An ImageStream acts as a virtual repository or catalog for container images within your OpenShift project.   (QQ- without this object also we are able to deploy the image na then whats use of this?)
* **Centralized View:** It provides a centralized view of all the images associated with your application or project.
* **No Actual Image Data:** The ImageStream itself doesn't store the actual image data. Instead, it holds references to images located in external registries (like Docker Hub, Quay, or your private registry).

**For single image we can have multiple tags**

**We can create an image streamtag for imagestream**

In my project, I have deployed the code without is, istag objects (I deployed only with deployment object, route object, service object)

|  |  |
| --- | --- |
|  | oc get istag |
|  | oc delete istag <image stream tag name>  oc delete istag hello-world |
| Describe istag- to see all the image details such as size , who is the author | oc describe istag <image stream tag name>  oc describe istag hello-world:latest |
| to see istag – image stream tag | oc get istag |
| to see image stream | oc get is |
| downloading the image from repo and keep that image into kub cluster – but this wont deploy the image | oc import-image --confirm quay.io/practicalopenshift/hello-world |

|  |
| --- |
| we can create an alias name for image and alias for tag name for an image – I don’t know what is the use of this, instead we can use the same image name na  oc tag <original image name> <destination image name>  oc tag quay.io/practicalopenshift/hello-world:updated-message mani-bro:update-message  oc tag quay.io/practicalopenshift/hello-world:<some tag name> mani-bro:<some tag name>  ex:- oc tag quay.io/practicalopenshift/hello-world:v1 mani-bro1:v1  result  E:\study related\my git hub -new\JavaAllSubjectsNotes\6.devops\2.2.Openshift>oc get is  NAME IMAGE REPOSITORY TAGS UPDATED  codi default-route-openshift-image-registry.apps.rm2.thpm.p1.openshiftapps.com/vv-manideep-dev/codi latest 6 days ago  mani-bro1 default-route-openshift-image-registry.apps.rm2.thpm.p1.openshiftapps.com/vv-manideep-dev/mani-bro1 v1  <see here in above we got a new alias image name & new tag name >  with this new alias image name we can deploy it again as 🡪 oc new-app mani-bro1:v1 --as-deployment-config |