Objective – to understand containers very well and be able containerise any application and deploy using tools docker, containerd, kubernetes ….. Etc. we will also focus on kubernetes cluster and its administration side.

Agenda →

Docker(1-1.5) + Kubernetes(2.5-3)

Session Logistics →

1. 9.00 - 5.00
2. Breaks –
   1. Breather breaks - every 90 minutes
   2. Lunch break - 1.15 - 2.15
3. Theory(i will) + Demonstrations(i will) + HandsOn(you will on your lab)
4. Lab Access →
   1. Day 1/2 – one vm for docker
   2. Day 2/3/4 – two vm for kubernetes
5. Introductions →
   1. Your intro
   2. Familiarity with containers
   3. Expectations/Questions
6. Accessing Labs →
   1. Go to <https://lms.springpeople.com/>
   2. Login using official email address or may be reset the password
   3. Go to cloudlabs and start only VM-1 for today.

DAY - 1 →

Containers – are small, lightweight, self dependent, platform independent, kernel dependent run time environments which can be used to deploy any kind of application ex - databases, api, frontend, backend, microservices ….etc…..

\*containers have been out there since 1996.

\*in 2012 docker made container simple to operate.

\*docker is not a CRT.

\*Docker makes it easy for you to consume CRT. build containers, run containers, debug them …etc..

\*docker is a software which makes it easy for us to consume containers.

Container run time (CRT) – helps you create containers. Ex – containerd, runc, crun ….etc…

\*containers are created from container images.

\*from one container image one can build any number of containers.

Build(image) - Ship(image) - Run(container)

Docker →

\*Installing docker – <https://docs.docker.com/engine/install/ubuntu/>

\*install docker will install docker client, docker engine, CRT and few docker utilities.

\*docker commands (docker client) → docker engine (docker process - CRT)

H11 → Connect with VM-1 and install docker.

sudo -i (Welcome@123 is password for root)

wget -qO- https://get.docker.com/ | sh

2 clear

3 service docker status

4 clear

5 docker info

6 clear

7 docker images – to list all images

8 docker ps – to see all running containers

docker ps -a – to see all running + stopped containers

9 history

Images →

* Think of images as build artifacts used to create containers
* Without images containers cant be created
* As a best practice one write Dockerfile which when built upon results in an image.
* Docker hub - hub.docker.com - this is a community portal where one can find a number of container image pre-built.
* On docker hub →
  + Official image (use these only) – reponame:tag
  + Developer images – dockerhubid/reponame:tag
* Tag is used to represent version. If not mentioned system assumes tag=latest.
* docker pull - download an image and docker push - upload an image
* Repositories can be public/private. From public repositories anyone can download, most official repos are public.

H12 → Go to hub.docker.com and sign up. Login and search for some images that you familiar with - python, nodejs, redis …etc…

* Images are stored in repositories. Repositories can be part of organisation/dockerhub ids.

H13 → Pull a few images and learn to identify their usage from their documentation.

docker images

15 clear

16 docker pull tomcat

17 docker images

18 docker pull tomcat:8-jdk8-corretto

19 clear

20 docker pull rathihimanshu/sept5:v1

21 clear

22 docker pull python:alpine

23 docker pull tutum/apache-php

24 clear

25 docker pull redis

26 docker pull centos:7

27 docker images

\*Container images are supposed to be small and lightweight. And they are small and lightweight because they container minimal set of binaries.

H14 → create first container and verify binaries missing from it.

docker run -itd --name=a1 centos:7

38 docker ps

39 docker exec -it a1 /bin/bash

which curl

which wget

exit

Containers Overview→

* Container represent an instance of your application
* Containers have their own cpu/memory/networking/storage/file system/processes/users/groups..etc..
* We can control all the above mentioned properties for the containers.
* A stopped container wont serve your application.
* Orchestrators like docker swarm, kubernetes they delete the stopped containers.
* If images are perfect, container will run fine.

H15 → Create few containers and look at their properties, stop container, remove it.

docker ps

44 docker run -itd --name=a2 tomcat

45 docker run -itd --name=a3 redis

46 docker run -itd --name=a4 redis

47 docker run -itd --name=a5 python:alpine

48 docker ps

49 clear

50 docker inspect a1

51 clear

52 docker ps

53 docker stop a1

54 docker ps -a

55 docker start a1

56 docker ps

57 clear

58 history

59 clear

60 docker run --help

61 clear

62 history

63 clear

64 docker ps

65 docker rm -f a5

66 docker ps

* Each container has its own processes.
* A processless container automatically will stop.
* A container must have a process to keep it running.
* Containers can have more than one process but the primary process is represented with PID-1 in the container.
* Killing PID-1 will kill the container.

H16 → Understand what keeps a container running and importance of container processes.

71 docker run -itd --name=b1 centos:7 sleep 100

72 docker ps

73 docker ps -a

74 clear

76 docker run -itd --name=b2 tomcat

77 docker exec -it b2 /bin/bash

ps aux

\*safe exit - ctrl p q

78 docker exec -it b2 /bin/bash

Kill 1

kill -9 pidnumber

docker ps -a

79 clear

80 ps -p 1

81 history

* Container images dont carry kernel they share kernel with base machine.
* Containers themselves are a process to the base OS.
* If CRT stops then containers stop.

H17 → Try out above.

docker ps

85 ps aux | grep containerd

86 kill 4282

87 docker ps

88 service docker stop

89 docker ps

90 service docker start

91 docker ps

92 clear

93 history

94 cclear

95 clear

96 docker ps -a

\*In linux OS systemd does the resource allocation.

How containers are created →

* CRT leverages three features of linux kernel to create containers.
  + CGROUPs – are used to allocate computing resources - cpu/memory.

\*When you dont limit cpu/memory to a container and if container process goes bad, it can consume entire system resources.

docker run -itd --name=c1 tomcat

104 docker run -itd --name=c2 -m 128m tomcat

* + Namespace – are used to provide isolation to container processes, networking, users, groups etc…
  + UFS – union mounted file system – is a way to mount container layer on top of image layers.

\*image layers – images are made up layers mounted on top of each other.

\*image layers are immutable, it is the container layers which is r/w.

\*storage driver of CRT determines how container layer interacts with image layers.

\*container are supposed to be stateless and dataless.

\*<https://docs.docker.com/storage/storagedriver/>

H18 → Understand above.

docker rm -f `docker ps -a -q`

102 clear

103 docker run -itd --name=c1 tomcat

104 docker run -itd --name=c2 -m 128m tomcat

105 cd /sys/fs/

106 ls

107 cd cgroup/

108 ls

109 clear

110 cd

111 history

112 cclear

113 clear

114 docker images

115 docker history tomcat

116 clear

117 docker info

\*In linux, one can use Cgroups or SystemD either to allocate computing resources. It depends on CRT configuration what you pick.

Docker Home → is a location where docker manages all the container layers, image layers, logging data etc etc ..

\*/var/lib/docker

H19 → check out docker home.

cd /var/lib/docker/

128 clear

129 ls

130 cd containers/

131 ls

132 docker ps -a

133 docker rm -f `docker ps -a -q`

134 ls

135 docker run -itd --name=a1 tomcat

136 ls

137 cd 83d2f7b2b0c04b74df57c98a76908bf09e157df4984325f0093336350c98560c/

138 ls

139 cd ..

140 clear

141 ls

142 cd ..

143 ls

144 cd overlay2/

Docker Commands →

* attach – is used to associate with pid-1 of the container.

H20 →

docker attach containername

docker run -itd –name=n1 centos:7 /bin/sh -c ‘while true;do echo hello;sleep 1;done’

* commit – is used to convert a container into an image.

H21 →

docker run -itd --name=m1 centos

165 docker exec -it m1 /bin/bash

touch f2 f3 f4 f5 f6

exit

166 docker history centos

167 docker commit m1 rathihimanshu/4dec:v1

168 docker rm -f m1

169 docker run -itd --name=new1 yourdockerhubid/4dec:v1

170 docker exec -it new1 /bin/bash

171 docker history rathihimanshu/4dec:v1

* Image repositories are used to store image.
* docker login – is used to login into docker hub
* docker pull
* docker push imagename – to push the image.

Image should be like dockerhubid/reponame:tag

* docker tag x dockerhubid/reponame:tag
* docker rmi imagename (if an image is in use by a container it wont be removed)

H22 →

docker login

175 docker images

176 docker tag rathihimanshu/4dec rathihimanshu/5dec

177 docker tag rathihimanshu/4dec:v1 rathihimanshu/5dec:v1

178 docker images

179 clear

180 docker push rathihimanshu/4dec:v1

* docker logout – logout of docker hub.
* docker cp source c1:location – is used to copy content from base system to container and container to base.
* docker diff containername – know about difference between the actual image and modifications made to the container layer.
* docker rename new1 new2
* docker restart new1
* docker start new1
* docker stop new1
* docker top new1 – process wise consumption at that point in time
* docker stats new1 – provide live data
* docker logs containername – to see the logs
* docker pause containername – is used to pause PID-1 in container.
* Docker unpause containername – to unpause

H23 →

docker run -itd --name=j1 centos /bin/bash -c 'while true;do X=$[$X+1];echo $X;sleep 1;done'

210 docker logs j1

211 clear

212 docker logs j1

213 clear

214 docker pause j1

215 docker logs j1

216 clear

217 docker unpause j1

218 docker logs j1

Dockerfile – it is a file that can result into a container image when build upon.

\*Developer → app code + Dockerfile → built upon your get an image

\*<https://docs.docker.com/engine/reference/builder/>

\*Dockerfile has 11 instructions that you use commonly to write the kind of images you want.

\*# is to write comments.

1. FROM – is always the first instruction in a docker file.
2. COPY – is used to copy file/folders into an image.
3. ADD – can do everything that copy does but can also do wget on remote URLs.
4. ENV – set env variables

H24 → Try out Dockerfile.

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

nano Dockerfile

231 clear

232 docker build -t i1 -f Dockerfile .

233 docker history i1

234 docker rm -f `docker ps -a -q`

235 clear

236 docker run -itd --name=a1 i1

237 docker exec -it a1 /bin/bash

238 clear

239 docker images

240 cat Dockerfile

\*ctrl+x

Review Time →

DAY - 2 →

Ex – start the docker lab vm and connect to it.

Ex – Write a dockerfile using nginx image as base, copy an index.html into it with your name in it at nginx home. Build the image and upload it to docker hub in repo called myname with tag v1. Create a container from it and connect with container on port 80 to verify if its shows your name.

\*/usr/share/nginx/html/index.html

1. RUN →
   1. Can be used to interact with package managers (pip, gem, yum …etc..) to install packages/binaries
   2. Allows you to run any build commands

\* RUN statement adds thickest of the image layers and hence always look to reduce the number of RUN statements as well as optimise.

Badly written dockerfile →

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

RUN yum install -y wget

RUN yum install -y curl

RUN mkdir /demodir

RUN useradd demouser

Right way to do it →

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

RUN yum install -y wget curl && mkdir /demodir && useradd demouser

H21 – Try out run statement and know the right way to do it.

1. WORKDIR – helps your change the home directory during build time for containers.
2. USER – to change the default user during the run time.

H22 →

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

RUN yum install -y wget curl && mkdir /demodir && useradd demouser

WORKDIR /demodir

USER demouser

1. EXPOSE –

\*in kubernetes if an image did not have EXPOSE instruction during the build POD post isnt mapped to container port.

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

RUN yum install -y wget curl && mkdir /demodir && useradd demouser

#WORKDIR /demodir

#USER demouser

EXPOSE 8081

1. VOLUME –

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

RUN yum install -y wget curl && mkdir /demodir && useradd demouser

#WORKDIR /demodir

#USER demouser

EXPOSE 8081

VOLUME /demodir

Commands →

docker volume ls

310 docker volume prune

311 cd /var/lib/docker/volumes/

312 ls

313 cd

314 docker build -t i7 -f Dockerfile .

315 clear

316 docker run -itd --name=v1 i7

317 cd /var/lib/docker/volumes/

318 ls

319 cd e7a8a709947aa95517dd42a5de07999cd20c5d2537ea658bff381afb5af6efff/

320 ls

321 cd \_data/

322 ls

323 docker exec -it v1 /bin/bash

324 docker rm -f v1

325 ls

326 cd

327 cat Dockerfile

H24 – try out expose and volume.

Ex – write a dockerfile which deploys a simple script of your fav programming language.

1. ENTRYPOINT – If PID-1 starts with entrypoint one cannot override it during the run time.
2. CMD – if PID-1 starts with CMD then one can override it during the run time.

H25 →

root@ip-172-31-88-17:~# docker run -it --name=y1 i9

hello from entrypoint

root@ip-172-31-88-17:~# docker run -it --name=y2 i9 /bin/echo 'hello all'

hello from entrypoint

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

#RUN yum install -y wget curl && mkdir /demodir && useradd demouser

#WORKDIR /demodir

#USER demouser

#EXPOSE 8081

#VOLUME /demodir

#CMD /bin/echo 'hello from CMD'

ENTRYPOINT /bin/echo 'hello from entrypoint'

Commands →

docker build -t i8 -f Dockerfile .

334 docker run -it --name=z1 i8

335 docker run -it --name=z2 i8 /bin/echo 'hey i am good'

336 nano Dockerfile

337 clear

338 docker build -t i9 -f Dockerfile .

339 docker run -it --name=y1 i9

340 docker run -it --name=y2 i9 /bin/echo 'hello all'

341 cat Dockerfile

\*there is special scenario when you dont PID to change but you want its behaviour to change based upon argument. Then you can use ENTRYPOINT and CMD both in an image.

H26 →

root@ip-172-31-88-17:~# cat Dockerfile

FROM centos:7

COPY test.txt /test.txt

ADD https://facebook.com/index.php /fbhome.php

ENV var1 value1

#RUN yum install -y wget curl && mkdir /demodir && useradd demouser

#WORKDIR /demodir

#USER demouser

#EXPOSE 8081

#VOLUME /demodir

ENTRYPOINT ["/bin/echo"]

CMD ["hello all"]

docker build -t i10 -f Dockerfile .

350 clear

351 docker run -it --name=x1 i10

352 docker run -it --name=x1 i10 hey

353 docker run -it --name=x2 i10 hey

354 docker run -it --name=x3 i10 byeeveryone

355 cat Dockerfile

Ex:-- Review at least 5 Dockerfiles of environment you are familiar with and see if you can find the behaviour of container by seeing the image.

Multi-stage build → if you do build and deploy in a single image then your images become bigger because they binaries which are not actually needed after the build is done.

\*<https://docs.docker.com/build/building/multi-stage/>

H27 → multi stage build

root@ip-172-31-88-17:~# cat multi

FROM centos:7 AS build

RUN yum install -y wget curl

COPY index.html /index.html

FROM nginx AS deploy

COPY --from=build /index.html /usr/share/nginx/html/index.html

Time till – 11.53

Container Orchestrators → to create containerised workloads at scale, with HA you have to make use of orchestrators.

\*container orchestrators are distributed systems.

\*kubernetes(eks, aks, gke), docker swarm, me, open shift, rancher, mesos ….etc…

\*k8s – <https://github.com/kubernetes/kubernetes>

\*<https://docs.docker.com/engine/swarm/>

Kubernetes Overview →

* The first orchestrator which was released by google
* Latest version of kubernetes is 1.28
* Two types of machines – master machine + node machine
* One can have more than one master for HA and one must have more than one node for HA
* Kubermetes master is brain of the cluster
* All the kubernetes nodes report health and their workload data to master
* By default kubernetes master doesnt participate in the workload sharing but if you want you can force it to participate but its not a good practice
* Kubernetes master itself is very bulky and run them on itself
* You use kubectl to communicate with master
* <https://kubernetes.io/docs/concepts/overview/>
* Control plane(master), data plane(nodes)
* <https://kubernetes.io/docs/concepts/overview/components/>
* You always need a CRT with kubernetes, kubernetes isnt CRT.

Creating Kubernetes Cluster →

\*minikube (single machine dev env), kubeadm, kops …etc..

\*managed kubernetes like eks, aks, gke takes away the responsibility of maintaining kubernetes master.

1. Connect with lab and provision the remaining two vms and connect with them. Vm-2 – kubernetesmaster, vm-3 kubernetesnode
2. Vm-2 – kubem, vm-3 –kuben

sudo -i

hostnamectl set-hostname kubem

2 hostnamectl set-hostname kubem

3 exec bash

(change the hostnames on both the vms appropriately)

1. On both the vms.

apt install net-tools

ifconfig

1. On kubem →

nano /etc/hosts

root@kubem:~# cat /etc/hosts

172.31.88.43 kubem

127.0.0.1 kubem localhost

172.31.88.77 kuben

1. On kuben →

root@kuben:~# cat /etc/hosts

172.31.88.77 kuben

127.0.0.1 kuben localhost

172.31.88.43 kubem

1. Ping each other and verify they are reachable on hostnames.

ping kuben - from master

ping kubem – from node

Time – 12.59

\*<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

1. In kubernetes on all nodes, swap must be disabled.

\*swapoff -a – this will disable swap temporary.

\*/etc/fstab – comment out any swap entries. And restart the machine.

\*cat /etc/fstab

1. Kubernetes communication ports. <https://kubernetes.io/docs/reference/networking/ports-and-protocols/>
2. Ip tables set up →

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#containerd>

1. Ensure the proper cgroup driver is there based upon the choice of CRT.

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#forwarding-ipv4-and-letting-iptables-see-bridged-traffic>

\*<https://github.com/cri-o/cri-o/blob/main/install.md#readme>

1. Install cri-o →

OS="xUbuntu\_22.04"

VERSION="1.28"

cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list

deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /

EOF

cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list

deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /

EOF

curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -

curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -

sudo apt-get update

sudo apt-get install cri-o cri-o-runc cri-tools -y

sudo systemctl daemon-reload

sudo systemctl enable crio --now

service crio status

1. Set up kubeadm, kubelet and kubectl →
   1. sudo apt-get update
   2. sudo apt-get install -y apt-transport-https ca-certificates curl gpg
   3. curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
   4. echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
   5. sudo apt-get update
   6. sudo apt-get install -y kubelet kubeadm kubectl
   7. apt-mark hold kubelet kubeadm kubectl
2. Ifconfig to fine kubem ip

\*steps - 14-17 – only on kubem

1. kubeadm init --apiserver-advertise-address=172.31.88.43 --apiserver-cert-extra-sans=172.31.88.43 --pod-network-cidr=192.168.0.0/16 --node-name kubem
2. Ensure to copy and run commands to generate kubeconfig file.
3. kubectl get nodes
4. kubectl get pods -n kube-system

CNI - cluster network interface – used for pod networking in the cluster.

1. Enable calico plugin on master →
   1. kubectl create -f <https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml>
   2. curl https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml -O
   3. kubectl create -f custom-resources.yaml
   4. kubectl get pods -n calico-system
2. kubeadm token create --print-join-command – run on kubem
3. kubectl get nodes

\*kubeadm reset – will delete all the configuration files generated by kubeadm.

CRT in kubernetes →

* Kubernetes initially didnt support any other crt
* CRI – container runtime interface was introduced by kubernetes to let any CRT work with kubernetes as long as CRT was compatible with OCI
* <https://opencontainers.org/>
* After 1.24, docker engine became incompatible with kubernetes. Dockershim was making docker work with kubernetes up until then.
* People have moved over docker and now have started using other container run times with it.

POD →

* In kubernetes you dont deploy containers directly you deal with PODs.
* A POD is a virtual object which can have one or more than one container in it
* Think of POD as microservice instance
* Each POD has a unique IP across the cluster
* POD can never ever be container less
* POD health is dependent on health of all the containers in

HandsOn →

kubectl get nodes

109 clear

110 kubectl get pods

111 kubectl run podone --image=tomcat:8

112 kubectl get pods

113 kubectl get pods -o wide

115 kubectl get pods

117 clear

118 kubectl run podtwo --image=centos:7 --command -- sleep 60

119 kubectl get pods

120 kubectl run podthree --image=centos:7 --command -- sleep 10

121 kubectl get pods

DAY - 3 →

Ex – Provision lab VMs and connect with them.

POD Continued …

* POD gets IP address from CNI which is unique across and is shared by container in it
* Never ever put containers exposing the same port in a POD
* A POD never gets divided between nodes

H31 →

kubectl delete pod podtwo

151 kubectl get pods

152 kubectl get pods -o wide

153 kubectl kubectl delete pod podthree

154 kubectl delete pod podthree

155 kubectl exec -it podone /bin/bash

156 kubectl run newpod --image=centos:7 --command -- sleep 3600

157 kubectl exec -it newpod /bin/bash

\*yaml – ports

\*kubectl run podname –image=tomcat

Kubectl →

\*kubectl makes use of config file to authentication with api server.

\*two way to do thing in kubernetes →

* Imperative – kubectl commands
* Declarative – yaml files (manifests)

\*kubectl <action> <objects> –param1 –parm1

\*kubernetes has plenty of objects like pod, jobs, cronjob, rs, deployments, services, configmaps ….etc…

H32 →

kubectl get nodes

173 cd .kube/

174 ls

175 cat config

176 clear

177 cd

178 kubectl get pods

179 clear

180 kubectl run --help

181 clear

182 kubectl

183 clear

184 kubectl get pods

185 kubectl delete pod podone

186 kubectl get pod --help

187 clear

188 kubectl get pod p1 -o yaml

Namespace → is used to represent environment, for permission restriction, computing restriction.

* By default you are always using the default.
* Namespaces have nothing to do nodes
* Never ever delete the ns, deleting ns will delete all the workload in it

H33 →

kubectl get pods

194 kubectl get rs

195 kubectl get ns

196 kubectl get pods -n kube-system

197 kubectl get pods -all

198 kubectl get pods --all

199 kubectl get pods --help

200 clear

201 kubectl get pods --all-namespaces

202 clear

203 kubectl create ns testns

204 kubectl get ns

205 kubectl run testpod --image=tomcat -n testns

206 kubectl get pods

207 kubectl get pods -n testns

208 kubectl delete ns testns

Writing Objects in Declarative Manner →

* One can write kubernetes objects in yaml files
* Objects have properties and you mentioned these properties in yaml
* Mandatory properties + optional properties
* <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/>
* apiVersion, kind(type of object), metadata(namespace, labels, annotations etc ..), spec(properties), status(it is provided by cluster after object is created)
* <https://k8syaml.com/>
* Yaml basics →
  + key: value
  + Lists –

H34 → Create your first object with yaml.

nano playns.yaml

215 clear

216 kubectl apply -f playns.yaml

217 kubectl get ns

218 kubectl delete -f playns.yaml

219 kubectl get ns

220 kubectl apply -f playns.yaml

221 clear

222 kubectl get ns playns -o yaml

223 kubectl get pod p1 -o yaml

224 clear

225 cat playns.yaml

226 history

root@kubem:~# cat playns.yaml

apiVersion: v1

kind: Namespace

metadata:

name: playns

Multi container POD →

* Dont put containers exposing the same port in the same POD
* One can get into specific container of a pod by using -c option
* Containers in a POD can communicate with each other using localhost

H35 →

root@kubem:~# cat playpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: playpod

namespace: playns

spec:

containers:

- name: c1

image: nginx

- name: c2

image: tomcat

- name: c3

image: centos:7

command: [/bin/sh,-c,"sleep 3600"]

- name: c4

image: redis

kubectl apply -f playpod.yaml

257 kubectl exec -it playpod -n playns -c c1 /bin/bash

258 kubectl exec -it playpod -n playns -c c2 /bin/bash

259 kubectl exec -it playpod -n playns -c c4 /bin/bash

260 kubectl exec -it playpod -n playns -c c3 /bin/bash

curl localhost:80

curl localhost:8080

Ex – Using the playpod.yaml, delete the playpod. Now just make modification to set an environment variable for c3 container var1 to have value of value1. Go to container and verify if env variable is set.

\*echo $var1.

Labels → are key value pairs which are used to group objects, filter objects or link objects.

\*env=dev

\*written under metadata.

H36 →

root@kubem:~# cat p2.yaml

apiVersion: v1

kind: Pod

metadata:

name: p2

labels:

project: abc

spec:

containers:

- name: n1

image: tomcat

nano p2.yaml

268 clear

269 kubectl apply -f p2.yaml

270 kubectl get pods

271 kubectl get pods -l project=abc

ResourceLimits →

* If no limits are set then a container can actually end up consuming all the system resources
* As a best practice always set upper limit at-least on each container
* <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

Ex:-- Write a pod which has a lower limit and upper limit set.

Object Relationship → objects have highlevel-lowlevel relationships. It is better to deal with highlevel objects because they add fault tolerance to low level objects also provide you more configuration options.

\*pod → rs → deployment

\*pod → job → cronjob

RepliaSet – is a way to replicate a certain number of pods.

Deployment – provides capability to rollout applications at scale with different deployment strategies.

H37 →

root@kubem:~# cat playdep.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

# Unique key of the Deployment instance

name: playdep

namespace: playns

spec:

# 3 Pods should exist at all times.

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

# Apply this label to pods and default the Deployment label selector to this value

app: nginx

spec:

containers:

- name: nginx

# Run this image

image: nginx

kubectl apply -f playdep.yaml

281 kubectl get pods -n playns

282 kubectl get deployments -n playns

283 kubectl get rs -n playns

284 kubectl get pod -n playns

285 kubectl scale deployment playdep -n playns --replica=10

286 kubectl scale deployment playdep -n playns --replicas=10

287 kubectl get rs -n playns

288 kubectl get pod -n playns

289 clear

290 kubectl get rs -n playns

291 kubectl get pod -n playns

292 kubectl delete pod playdep-7854ff8877-7587m -n playns

293 kubectl get rs -n playns

294 kubectl scale deployment playdep -n playns --replicas=5

295 kubectl delete pod playdep-7854ff8877-7587m -n playns

296 kubectl get rs -n playns

297 clear

298 kubectl get rs -n playns

299 kubectl delete rs playdep-7854ff8877 -n playns

300 kubectl get rs -n playns

\*<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#deployment-v1-apps>

Ex:-- write new deployment in default ns called d1 which creates two containers in each pod of tomcat:8 and redis. The deployment must create 4 replicas total.

Deployment strategies & rollouts →

* Deployments can be used to controlled rollouts – v1, v2 …etc.. – revisions
* Default deployment strategy is rolling.
* Strategies →
  + Allatonce(recreate) – causes a downtime
  + Rolling – no downtime and gradual pod by pod update
  + % rollout
  + blue/green – your create a new deployment change the routing at service level – with minimum downtime

H38 → understand rollout, revision and deployment strategies.

kubectl rollout status deployment playdep -n playns

309 kubectl rollout history deployment playdep -n playns

310 nano playdep.yaml

311 clear

312 kubectl apply -f playdep.yaml

313 kubectl rollout status deployment playdep -n playns

314 kubectl get deployment playdep -o yaml

315 kubectl get deployment playdep -n playns -o yaml

316 clear

317 kubectl rollout status deployment playdep -n playns

318 kubectl rollout history deployment playdep -n playns

319 kubectl rollout --help

320 clear

321 kubectl rollout unto deployment playdep -n playns

322 kubectl rollout undo deployment playdep -n playns

323 kubectl rollout status deployment playdep -n playns

324 kubectl rollout history deployment playdep -n playns

325 cat playdep.yaml

326 clear

327 kubectl get pods -n playns

328 kubectl exec -it playdep-7854ff8877-kq28z -n playns

329 kubectl exec -it playdep-7854ff8877-kq28z -n playns /bin/bash

Job →

* Is used to create PODs with a finite life.
* A job pod will always have the end status as completed.

H39 → try out job object.

root@kubem:~# cat newjob.yaml

apiVersion: batch/v1

kind: Job

metadata:

# Unique key of the Job instance

name: newjob

spec:

completions: 7

parallelism: 4

template:

metadata:

name: jobpod

spec:

containers:

- name: j1

image: centos:7

command: [/bin/sh,-c,"sleep 20"]

restartPolicy: Never

CronJob → is a high level object to job object which is used to create a job at a schedule.

H310 –

root@kubem:~# cat newcron.yaml

apiVersion: batch/v1

kind: CronJob

metadata:

name: mycron

spec:

schedule: "\* \* \* \* \*"

jobTemplate:

spec:

template:

spec:

containers:

- name: c1

image: centos:7

command: [/bin/sh,-c,"sleep 10"]

restartPolicy: OnFailure

\*kubectl get cronjob

DaemonSet → is a deployment level object for which number of replicas is always equal to the number of nodes and daemon set ensure that one pod exactly is running on each node.

H311→

root@kubem:~# cat newds.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

# Unique key of the Deployment instance

name: newds

spec:

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

# Apply this label to pods and default the Deployment label selector to this value

app: nginx

spec:

containers:

- name: nginx

# Run this image

image: tomcat

kubectl delete -f newcron.yaml

384 kubectl get ds

385 clear

386 ls

387 cp playdep.yaml newds.yaml

388 nano newds.yaml

389 clear

390 kubectl apply -f newds.yaml

391 kubectl get pods

392 kubectl get ds

393 cat newds.yaml

\*to feed data from you use configMap and secret in kubernetes.

\*configmaps store the data in plain text while secret can be encrypted.

ConfigMap →

H312→

root@kubem:~# cat newconfigmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: newconf

data:

data1: "somedata here for one"

data2: "some data for two"

root@kubem:~# cat newconfpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: newconfpod

spec:

containers:

- name: d1

image: alpine

command: ["sleep", "3600"]

env:

- name: env1 # Notice that the case is different here # from the key name in the ConfigMap.

valueFrom:

configMapKeyRef:

name: newconf # The ConfigMap this value comes from.

key: data1

kubectl exec -it newconfpod /bin/sh

Secrets → are used to pass on encrypted data. By default secrets are encrypted in base64.

H313 →

root@kubem:~# cat newsecret.yaml

apiVersion: v1

kind: Secret

metadata:

name: newsecret

data:

sec1: UGFzc3dvcmRAMTIz

root@kubem:~# cat newsecpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: newsecpod

spec:

containers:

- name: d1

image: alpine

command: ["sleep", "3600"]

env:

- name: env1 # Notice that the case is different here # from the key name in the ConfigMap.

valueFrom:

secretKeyRef:

name: newsecret # The ConfigMap this value comes from.

key: sec1

Services →

* Are used to expose deployed workload with a single endpoint
* Provides load balancing to your workload
* 4 types →
  + ClusterIP – is used to expose workload in the cluster only

H314 → try out clusterip

kubectl delete pods --all

452 clear

453 kubectl get pods

454 kubectl create deployment demodep --image=tutum/apache-php

455 kubectl get pods

456 kubectl scale deployment demodep --replicas=5

457 kubectl get pods

458 kubectl get services

459 kubectl expose deployment demodep --name=cip --type=ClusterIP --port=8000 --target-port=80

460 kubectl get services

461 kubectl run centpod --image=centos --command -- sleep 3600

462 kubectl exec centpod -it /bin/bash

463 clear

curl serviceip:8000

\*cluster ip used to expose secure services – like auth, db, back end apis …etc..

* + NodePort – can be used to expose in the cluster + on the node network

H315 →

kubectl get deployments

476 kubectl get services

477 kubectl expose deployment demodep --name=nodeportserv --type=NodePort --port=80 --target-port=80

478 kubectl get services

479 curl kuben:31115

Time till – 16.29

* + LoadBalancer (only on cloudplatform) – can be used to expose workload in the cluster + on the node network + globally using a load balancer

\*in a local cluster it behaves like a node port because no lb integration.

kubectl expose deployment demodep --name=lbservice --type=LoadBalancer --port=80 --target-port=80

* + ExternalName – is not used to expose workload rather is used make alias at the cluster.

\* kubectl create service externalname gg --external-name google.com

* Services can be consumed by the name across the cluster.
* <https://kubernetes.io/docs/concepts/services-networking/service/>

Ex:-- Write a service object of type node port.

\*<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#service-v1-core>

DAY - 4 →

Ex – provision lab vms and connect.

Ingress Object →

* Is a high level object to service object.
* Behaves like a smart loadbalancer
* Can be used for pathbased, hostbased, parameters based ….etc.. routing.
* Underlying services in an ingress object can only be either NodePort or LoadBalancer.
* <https://kubernetes.io/docs/concepts/services-networking/ingress/>

\*HELM → package manager.

H41 – Set up ingress controller →

kubectl get services

499 kubectl get ingress

500 kubectl get ingressclass

501 clear

502 kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.8.2/deploy/static/provider/cloud/deploy.yaml

503 kubectl get pods -n ingress-nginx

504 kubectl get ingressclass

\*kubectl taint nodes kubem node-role.kubernetes.io/control-plane-

* Makes use of ingress rules to define the kind of routing you want in the system.

Trying out nginx ingress →

H42 → clean up, create two deployment, scale and expose.

kubectl get pods

519 kubectl create deployment greendep --image=rathihimanshu/alb:green

520 kubectl create deployment yellowdep --image=rathihimanshu/alb:yellow

521 kubectl scale deployment greendep --replicas=4

522 kubectl scale deployment yellowdep --replicas=4

523 kubectl expose deployment greendep --name=greenserv --type=NodePort --port=80 --target-port=80

524 kubectl expose deployment yellowdep --name=yellowserv --type=NodePort --port=80 --target-port=80

525 clear

526 kubectl get services

527 kubectl exec -it centpod /bin/bash

[root@centpod /]# history

1 curl 10.108.110.230/green.html

2 curl 10.108.110.230/yellow.html

3 curl 10.101.110.201/yellow.html

4 curl 10.101.110.201/green.html

5 history

Network Policy →

* Allows you to allows/block the traffic to/from a service to another service.
* By default all pods/service allows all ingress/egress traffic
* But the moment you apply a networkpolicy to a service/pod – all traffic is block unless it is allowed in networkpolicy.

H44 →

kubectl run ng --image nginx -l env=dev

533 kubectl get pods -o wide |grep ng

534 clear

535 kubectl exec -it centpod /bin/bash

root@kubem:~# cat mynp.yaml

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: test-network-policy

namespace: default

spec:

podSelector:

matchLabels:

env: dev

Init Containers → typical are used to prepare the application. If a pod has init container then actual container dont get created until init container are done executing.

H45 →

root@kubem:~# cat playpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: playpod

# namespace: playns

spec:

containers:

- name: c1

image: nginx

initContainers:

- name: c3

image: centos:7

command: [/bin/sh,-c,"sleep 30"]

Static POD →

* Kubelet on the node takes instructions from apiserver on the master.
* If control-plane is not available then administrations functions of cluster are down but workload continues to run.
* Static pod are created on the node by kubelet without control plane intervention.
* Is only used to create pod, you dont create any else using this method.

\*<https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/>

StatefullSet →

* This is object is used for statefull applications.
* PODs created by statefull set follow a order in provisioning, deprovisioning, network mgmt ..

H47 →

**apiVersion**: apps/v1

**kind**: StatefulSet

**metadata**:

**name**: web

**spec**:

**selector**:

**matchLabels**:

**app**: nginx *# has to match .spec.template.metadata.labels*

**serviceName**: "nginx"

**replicas**: 3 *# by default is 1*

**minReadySeconds**: 10 *# by default is 0*

**template**:

**metadata**:

**labels**:

**app**: nginx *# has to match .spec.selector.matchLabels*

**spec**:

**terminationGracePeriodSeconds**: 10

**containers**:

- **name**: nginx

**image**: registry.k8s.io/nginx-slim:0.8

**ports**:

- **containerPort**: 80

**name**: web

POD Scheduling, Taints …→

* Scheduler allocated the node to the workload
* You have to sometimes schedule workload on specific size of nodes, storage supported nodes …etc…. You need to be able to control scheduling.

Manual scheduling →

* nodeName as a property in the pod def.

Ex – use nodeName as a property in the POD definition to forcefully schedule a pod on the node.

\*<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#podspec-v1-core>

* Scheduler does the scheduling of pod to a node.
* Multi scheduler architecture – you use multiple schedulers for different strategies to be used.
* <https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/>
* If you have multiple schedulers and you dont define scheduler for one of the workload then it goes to default scheduler.
* To use custom scheduler use schedulerName.

<https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/>

Taints & Tolerations →

* Taint is set at node level
* Toleration is set at pod level
* Taint effect determines the outcome of setting up taint.

H49 →

kubectl get pods

567 kubectl taint nodes kuben env=test:NoSchedule

568 kubectl create d1 deployment --image=tomcat

569 kubectl create deployment d1 --image=tomcat

570 kubectl get pods -o wide | grep d1

571 kubectl scale deployment d1 --replicas=3

572 kubectl get pods -o wide | grep d1

573 clear

574 kubectl taint nodes kuben env=test:NoSchedule-

575 kubectl scale deployment greendep --replicas=1

576 kubectl scale deployment yellowdep --replicas=1

\*<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>

Ex – Add the toleratio to your deployment see if scaling up workload now makes your node eligible.

<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>

Node selector – can actually make use of labels as a filter but for scheduling.

H411 →

kubectl delete deployment d1

587 kubectl taint nodes kuben env=test:NoSchedule-

588 kubectl label nodes kuben env=prod

root@kubem:~# cat playpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: playpod

# namespace: playns

spec:

containers:

- name: c1

image: nginx

initContainers:

- name: c3

image: centos:7

command: [/bin/sh,-c,"sleep 30"]

nodeSelector:

env: prod

NodeAffinity →

* Is also a scheduling way applied at pod level to filter based upon labels.
* <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/>

Monitoring & Logging →

\*to store logs and process them splunk,promotheus, grafana, dynatrace, datadog …etc….

\*wget <https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>

In the deployment section container arguments add –

- --kubelet-insecure-tls

kubectl apply -f components.

kubectl top nodes

Kubectl top pods

\*

Ingress →

root@kubem:~# cat ing.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: minimal-ingress

# annotations:

# nginx.ingress.kubernetes.io/rewrite-target: /

spec:

ingressClassName: nginx

rules:

- http:

paths:

- path: /green.html

pathType: Prefix

backend:

service:

name: greenserv

port:

number: 80

- path: /yellow.html

pathType: Prefix

backend:

service:

name: yellowserv

port:

number: 80

Storages in Kubernetes →

\*containers are supposed to dataless and stateless.

\*in case of kubernetes external volumes are often fed to the workload to make data persist.

\*to persist the data we depend upon kubernetes volumes.

\*emptyDir, hostPath, gitRepo, ebs, gcp pd, azure disk …..etc….

\*<https://kubernetes.io/docs/concepts/storage/volumes/>

* Persistent – have life cycle beyond the POD life.
* Non-persistent – this is a volume which has life cycle as the POD and is used to containers to share data in a POD. – emptyDir

HandsOn – emptyDir

root@kubem:~# cat emppod.yaml

apiVersion: v1

kind: Pod

metadata:

name: emppd

spec:

containers:

- image: nginx

name: n1

volumeMounts:

- mountPath: /usr/share/nginx/html

name: datavol

- image: centos:7

name: n2

command: [/bin/sh,-c,'while true;do X=$[$X+1];echo "<h1>$X</h1>" > /usr/share/nginx/html/index.html;sleep 1;done']

volumeMounts:

- name: datavol

mountPath: /usr/share/nginx/html

volumes:

- name: datavol

emptyDir: {}

kubectl apply -f emppod.yaml

599 kubectl exec -it n2 /bin/bash

600 kubectl exec -it emppd -c n2 /bin/bash

601 kubectl exec -it emppd -c n1 /bin/bash

602 clear

603 kubectl delete -f emppod.yaml

604 kubectl apply -f emppod.yaml

605 cat emppod.yaml

hostPath – it write data on the host and this data persists beyond the life cycle of the POD.

HandsON →

root@kubem:~# cat hostpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: hostpd

spec:

containers:

- image: nginx

name: n1

volumeMounts:

- mountPath: /usr/share/nginx/html

name: datavol

- image: centos:7

name: n2

command: [/bin/sh,-c,'while true;do X=$[$X+1];echo "<h1>$X</h1>" > /usr/share/nginx/html/index.html;sleep 1;done']

volumeMounts:

- name: datavol

mountPath: /usr/share/nginx/html

nodeName: kubem

volumes:

- name: datavol

hostPath:

path: /tmp/datadir

\*PV, PVC, StorageClasses

\*SC provisions a new PV when POD places a claim for PVC.

\*<https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/>

Cluster Upgrades/Maintenance →

* major.minor.patch
* V1.28.2 – patch versions
* <https://kubernetes.io/releases/>

\*one cant jump kubernetes versions, you have to go one by one

\*kubernetes components can also be of different versions.

\*more than one version different between apiserver and scheduler, controller, etcd …

\*and there should be not more then 2 version gap between apiserver - kubelet & kubeproxy.

\*at a time three versions are live.

Backing up the cluster →

\*on cloud platform they are replicated and backed up.

\*you need to back up etcd. /var/lib/etcd

\*<https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>

Upgrade Node →

\*kubectl drain nodename – is used to remove the pods running on the node safely.

\*cordoning will ensure whatever is running will run newworkload will not be allocated.

kubectl cordon nodename. – to disable scheduling.

kubectl uncordon nodename. – to enable scheduling.

\*<https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/>

Kubectl delete node nodename

Kubeadm reset –force

Q/A →