1-

**docker pull <image\_name>** => to download the image required to run a container

**docker images** => to list all downloaded images

**docker ps** => to list all the docker containers that are running

**docker stop <container\_name or container\_id>** => to stop a container

**docker start <container\_name or container\_id>** => to start a container

**docker ps -a** => to show all the docker container even if the stopped ones

2-

Docke containers can be identified by name or by a UUID

**docker run -d --name myname -p 8080:80 nginx**

the --name will actually will put a name to the container

-p PORT1:PORT2 => meaning we are mapping PORT1 of our physical machine to PORT2 of our docker container

**docker inspect <name of the container>** => to have information of our container

3-

**docker run -d --name myname -p 8080:80 nginx**

the -d mean run in detached mode => that mean run the command and free the console,

In attached mode the console is locked and its used what is the container is logging in the terminal

4- docker ps <options>

-q => the container IDs if not followed by **a**  it will list whats running

-a => all the contianers

-aq => all the container

4-

docker stop $(docker ps -q) => the $(docker ps -q) is used to get all the running containers and the result will be used with docker stop to stop everything

docker rm $(docker ps -aq) => basically to run everything

5-

docker container exec -t <container\_name> **bash** => to access the linux terminal of a container

the bash is basically the command that we will be executing inside the container

P.S: if the primary linux process of the container is stop the entire container will stop

P.S: the **i** in **-it** means give us a interactive console, and the **t** give us the output of the cmds and the d mean detached mode, so dt means detached mode with command output view

6- **docker run -d --name myname -p 8080:80 nginx <default\_command>**

**PS: if the the default commande exits the entire container will exist**

7-

Restart policy are useful to specify when our docker will be restarted

docker run -d --restart <restart-policy> ….

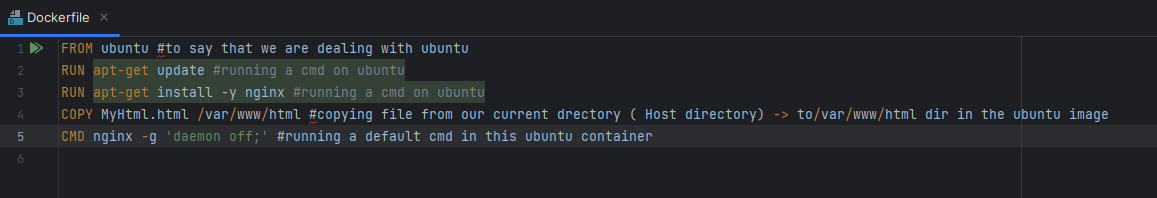
8-

**docker system df -v** => is to examine the disk usage of each container

9- **docker logs <container-name>** => logging the output

10 – docker … --rm => means when a container exit it will automatically deleted

11-



**docker build .** -> is to create an image out of the dockerfile

-t is used with the build command to specify that

12-

**COPY** => is used to copy a file from src (host) to a destination in the container

**ADD** => same as copy but we can put urls

13-

In docker File: EXPOSE <PORT\_NUMBER> is simply to document that the image will run on port 80

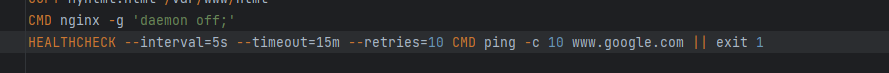
14-

HEALTHCHECK --interval=5s CMD ping -c 1 www.google.com

--interva=5s => launch the cmd each 5 second

In general HEALTHCHECK <options> CMD <commands>

15-



timeout and interval and retry are options to tell how often to excute the health check command

the || exit 1 => in case of a failure we tell docker that we have a unhealthy state

Other way to specify a health check is:

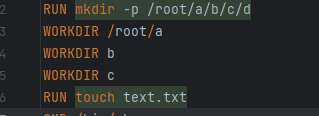
docker run -d … --health-cmd <cmd> --health-interval=5s –health-retries=4

16-



What will happen that ENTRYEPOINT will be considered as a prefix to cmd, example in this case the final command will be **/bin/test nginx -g ‘daemon off;’**

17-

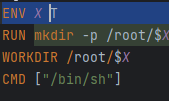


WORKDIR allow us change directory, here we start by /root/a

And the WORKDIR b will be executed inside /root/a so root/a/b ,workdir c will be /root/a/b/c

And then the run command will run in /toot/a/b/c

18-



We are creating a variable X of value T and we are using it as members of other cmd

docker run –env X=1 -e=Y => here we are creating env variable of X and Y to be used inside the linux kernel of the container, --env-file is to set variables present inside a file

19- docker rmi is for removing images

20- **docker build -t X:Y .** => we are building an image with name X and version Y

docker build tag img1:v1 img2:v2 => simply we are recreating

21-

docker container commit test2 newimg:0.0.2 => basically we are taking a copy of the test2 container and we creating an image from it known by the name newimg :0.0.2

docker container commit --change=’CMD[“…”]’ <container\_id> <imgname:version>

here the change arg means that we are changing the CMD part of the containerId and put it in a new img

22-

docker image inspect <dockerimage:version> -format='{{".ContainerConfig.Hostname"}}' => the goal is to extract certain information about the image like ContainerConfig.Hostname

23-

docker image prune -> to remove all image without a tag

docker image prune -a -> to remove all image without a tag and without a container attached to it

24-

Docker export <container-name> > <file\_path>.tar

Cat <file-path>.tar | docker import <new\_image\_name>

25-

To push our images to the registry

1. We need to tag the image like this: host\_registry:port\_registry/image\_name:image\_version
2. docker push host\_registry:port\_registry/image\_name:image\_version
3. docker pull host\_registry:port\_registry/image\_name:image\_version

26-

docker search <image\_name> --limit=3 –filter “is-official=true”

we are getting all the nginx images with 3 top stars and the official versions

27-

docker save img\_name > file.tar => to save the img as .tar

docker load < file.tar => to recreate the the image from a tar file

28-

Docker use cache to optimize build time, however if one step could not be cached all the other steps after it will not be cached

29-

Docker have multiple docker drivers each one with specific conf, from ips and default gateway and net and port forwarding.

docker network ls => to list all network drivers

docker inspect network-driver => to get the info of each driver

docker container run -dt --name myhost2 --network host ubuntu => to bind a specific container to a network

30-

docker network create --driver bridge mybridge => to create a bridged network

a user defined bridge network allow us to use dns to solve hostnames as IP

31-

docker container run -dt --name myhost2 --network none ubuntu => simply we are creating a container with no network

32-

docker container run ….. -P => means that docker will map the exposed port to a random one on the host

33-

With custom defined bridge network the container will be able to communicate with each other by the container name and not necessarily the ip address

34-

docker container run … --link container-name:container-alias --name x

means that the container x will be able to reach container-name by pinging container-alias

35- in docker installation directory we have a file for each storage driver.

When we create an image we will create some folders in our storage driver dir with each file represent a layer or a step of the image creation (see docker File) and those file will contain data to launch our image it can be our os files

36- If we change the storage driver all the container that are created in the previous storage driver will not be accessible.

To change the driver we need to stop docker, modify the deamon.json file and specify the other driver in it, then we need to restart docker. We can now see that another driver have been created in docker base directory by the name of the new driver

36-

Docker volume volume\_name

Docker run … -v volume\_name:/path => it means that the path inside our container will be stored in the volume\_name that have its data created in the host and not inside the container

Volumes are not removed if the container is removed

If an image declares a volume there is no need to add it in our run command

Volume are persistent even if our container is stopped or deleted, also multiple container can use the same volume

37- biding meaning we are attaching a directory on our host to a directory to a container

Docker container run …. -v path1:path2 => path1 the host path and path2 is the container

Or we can

Docker container run…. –mount type=bind,source=path1,target=path2,readonly

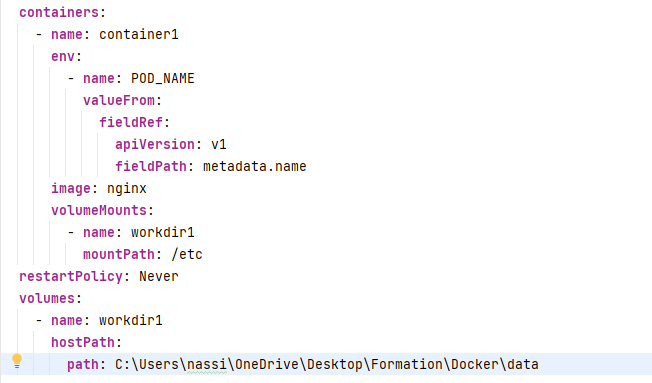
The –rm options is used to say that when a container exits we need to remove the volume

38- docker logs containername => to log the output of container

Docker run …. –log-driver log\_driver\_type => simply to specify how to create our logs

P.S: docker log will not work with some logging driver

39-



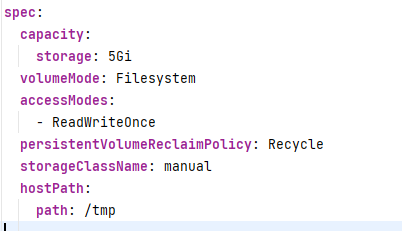
volumeMount.mountPath is the address in the container

hostPath.path is the path in our host

P.S

hostPath mean we are using the host storage, we can change to many options like AWS

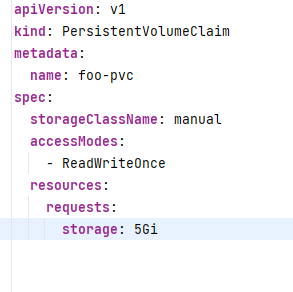
40-



The above image is simply specifying a storage of 5gi with accessmodes and the hostPath is /tmp.

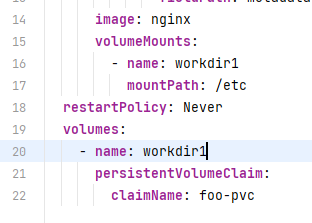
Note: storageClassName can be changed accordingly to what we want, azure,aws , for more info see the docs. So basically we are telling kuberntes that we have a storage of 5gi with some configs to be bind toa container on demand.

P.S: the kind attribute is PersistentVolume



The above image is simply creating a claim or a request that will be used to ask Kubernetes to provide

A volume with some specification, for example we are asking Kubernetes to give us a storage of 5gi and then Kubernetes will try to find the persistentVolume that matches the calims



In the above image we are simply asking associating a PV to our pods, simply we are telling that the required volume for our container should be described by the claim

41 – docker swarm init –advertise-addr <ip-address> => is used mainly to create the first manager in the swarm, the advertise-addr should the ip address of the manger

docker swarm join --token <token> manager\_id:<manger\_port on witch the manger is running

docker node ls => is to see all the info of all the connected workers to the manger

42-

docker service create --name <service-name> --replicas 5 <image\_name> => creating a service

docker service ps <service-name> => to see all the nodes that are running for a particular service

docker service ls => seeing all the service

docker service rm => removing a specific service

docker service scale <service-name>=10 => scaling a service to run 10 container

Service means that we are creating one container and distributed across multiple workers maybe the manager. We launch this service from the manger node. The role of the manger is to decide how to distributed the containers replications and controle to create and recreate in case of failure.

P.S: when a service run containers will be created in worker nodes, so a simple docker ps in the worker will show us the running container

43-

Docker service scale service\_1=num1 service\_2=num2

Docker service update –replicate 5 service\_name => here we are only allow to update one service

44- docker service create –name xyz –mode global img\_name => the global mean that this service will run on the entire clusters

45-

docker node update --availability <drain or active> wn02 => drain mean the target node will not receive tasks nor create containers

46- troubleshooting:

**docker service ps draining-test**

**docker service inspect<service name> –pretty** => is to display info about a specific running service, the flag –pretty is print in a readable way

**docker node inspect <node-name> –pretty** => displaying information about nodes

47-

**Docker service create –name xyz -p 8080:80 imge\_name** => is simply to tell that our container will be published on port 8080 mapped to 80 of the container

48-

docker-compose up -d => it will start the docker-compose.yml on the same directory, docker compse file is simply a file to describe how to create our containers

docker-compose down => will delete all the service create by the docker compose file

docker-compose config => it will simply verify all the config of the docker

version: '3.3'  
  
services:  
 db: => container name  
 image: mysql:5.7  
 volumes:  
 - db\_data:/var/lib/mysql  
 restart: always => restart strategy  
 environment:  
 MYSQL\_ROOT\_PASSWORD: somewordpress  
 MYSQL\_DATABASE: wordpress  
 MYSQL\_USER: wordpress  
 MYSQL\_PASSWORD: wordpress  
  
 wordpress:  
 depends\_on:  
 - db => these container will not create before the db container  
 image: wordpress:latest  
 ports:  
 - "8000:80"  
 restart: always  
 environment:  
 WORDPRESS\_DB\_HOST: db:3306  
 WORDPRESS\_DB\_USER: wordpress  
 WORDPRESS\_DB\_PASSWORD: wordpress  
 WORDPRESS\_DB\_NAME: wordpress  
volumes:  
 db\_data: {}

49-

docker stack deploy --compose-file docker-compose.yml <stack\_name> => to deploy a stack

docker stack ps stack\_name => to view the stack

docker stack rm stack\_name => to remove a stack

a stack mean multiple containers running grouped in one place,

the stack command described above are only usefull in swarm mode

50-

**docker swarm update --autolock=true** => is to lock the swarm , blocking all operation before unlocking

docker swarm unlock => is to unlock

docker swarm unlock-key => is to expose locking key

docker swarm unlock-key --rotate => is rotate the key

51-

docker service create --name servicename --constraint = node.labels.<key>==<value> => the following command will create and deploy the service based on a constraint that only deploy the service with node that have a labels with specific key and value

52-

docker node update --label-add <key>=<value> node\_name => to a label to a specific node

53-

Normally the docker swarm nodes are assigned an overlay network for their container => meaning that overlay network will allow our containers to communicate with each other even if they are on different nodes

Docker network create --driver overlay netname

Docker service create ….. --netwrok netname

docker network create --opt encrypted --driver overlay net\_name => we are creating a network with secure comms

54-

docker create ….. --hostname=”{{.Node.name}}-{{.Service.name}}” …. => simply the ”{{.Node.name}}-{{.Service.name}}” means that we are giving the host name of the created content a dynamic name composed of the nodename and servicename

--hostname is an option to specify the hostname of the container

55-

Manager node roles:

maintaining cluster state,scheduleing services , serving swarm mode http apis

to add another manager to the swarm: we need to run **docker swarm join-token manager** on a existing manager, then this command will generate a token that will be used as joining token for manager. We use this token and the same command we use to put worker nodes to the swarm to add the target machine as manager, docker swarm join --token <token> ip:port

56- to prevent containers to be deployed in the manager in a swarm we simply drain it

57- in case swarm cannot assign a leader, so in order to assign a new leader we should docker swarm init --force-new-cluster --advertise-addr ip:port to set the current node as a leader

58-

docker system events --since 2022-02-01 => is to display events since a date

docker system df => to show what is in our storages

59-

kubectl run container\_name --image nginx => is to create a container in Kubernetes

kubectl exec -it container\_name -- <command> => to run a command

kubectl get pods => will get all the running pods (group of containers)

each container will run in a pod, a pod can be a group of containers, and a pod will always run in one node

60- a Kubernetes object is a collection of intents ( what we want to do in our cluster)

Basically we can configure a yaml file to describe what are going to launch kubectl.exe apply -f pod.yaml

kubectl.exe delete -f .\pod.yaml => to delete everything related to the config file, usually we have the metadata that will help us



61-

Kubectl get pod -l env=prod -l key=value => here we are getting all labels with the env=prod label

Hint we can use != to say we want to search with label not equal to

62-

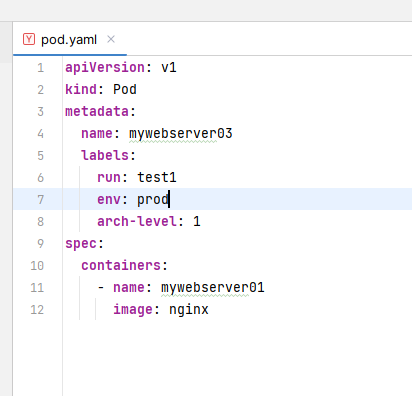
kubectl.exe get pods --show-labels => to get all the pods and show the labels

to assign a pod with a label:

kubectl label pod <name\_of the pod> key1=value1 key2=value2

kubectl label pod pod\_name key1- => the dash next to the key mean we are removing the label

kubectl label pods --all status=running => this command is used to get all the pods with a specific label



63-

Deployment provide multiple functionalities like scaling,rollback changes,rollout. It ensure a percentage of your replicates are down by default 25%

Example when deploying new version and we need to updgrade what will happen that we will create a replicat of the new version then delete the old one

**kubectl.exe rollout history deployment\_name --revision=> to get the changes in a specific version of a deployment**

**kubectl apply -f xyz.yaml with the condition that the kind is Deployement**

**kubectl.exe get deployments => get all the deployement names**

**kubctl describe deployment deployment\_name**

64-

kubectl.exe set image deployment test123 nginx=http here we are updating a specific deployment

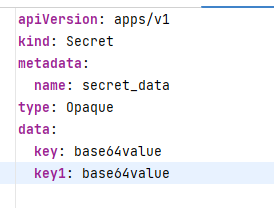
maxSurge is the max percentage of created pods from the original number example we have 8 in the update process we will have 8 + 8\*0.25 = 10

maxUnavaliable is the max number of pods that will be unavailable during the update phase

65-

kubectl.exe create secret generic mysecret --from-literal=pass=testtest => to create a secret , we need to set the type in our case is generic, the name of our secret and the data

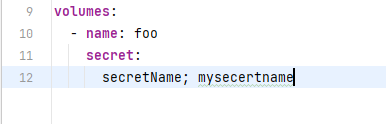
kubectl.exe get secret mysecret -o yaml => to get the secret

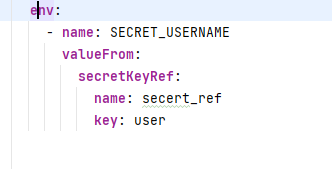


Kubectl apply -f file.yaml

66- to bind a secret to a pod or container we simply need to bind it into a volume

What we are doing is simply biding the secret to the filesystem of the pod/container





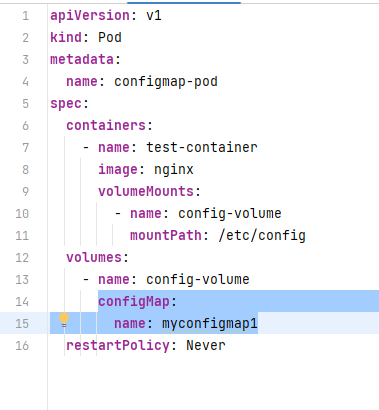
67-

Config map allow us to centralized our pod/containers configuration

Kubeclt create config\_name –from-litteral=key=value

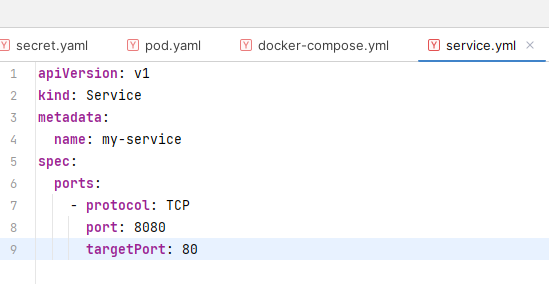
kubectl.exe create configmap myconfigmap1 --from-file .\conf.properties

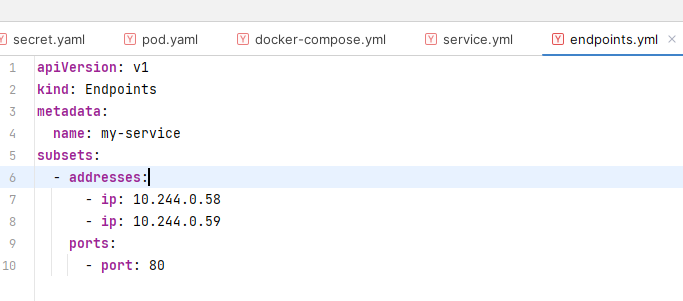
here are some way to load



kubectl.exe exec configmap-pod -it bash

68-





A service in kubernets can be like an http gateway,load balancer.

The first configuration file configure a service by translating port 8080 to port 80

The second file is simply an endpoint that definers all the ips and to what ports they should be mapped

P.S: In endpoint we always should put the name in the metadata the same as the name of the service to be able to bind the endpoints to the service

P.S : it is use full to use selectors with the service creation and in Deployment, it helps dynamically create the service endpoints and both selectors should be equal

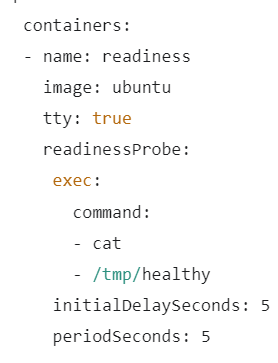
69-

NodePort allow to expose a service on the public network with a specific Port

70-



The script above simply describe how to check if a pod or container is alive. Exec can be of type command,http or tcp



The above script is simply to state when the service or the container is ready to be operational

71- DeamonSet is like a deployment but it make sure that each node have a copy for example if we have 3 replicas the application will be deployed on 3 different nodes

72-

Tainting means we are putting special condition on a node to allow it to received containers:

kubectl.exe taint nodes nodeName key1=value:NoSchedule => NoSchedule is a effect

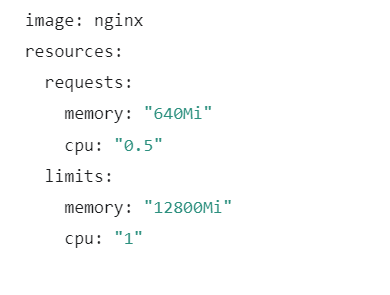
In order to allow a deployment to this node , we should specify the toloreations in the yml script :

**Specs:**

**Toloreations:**

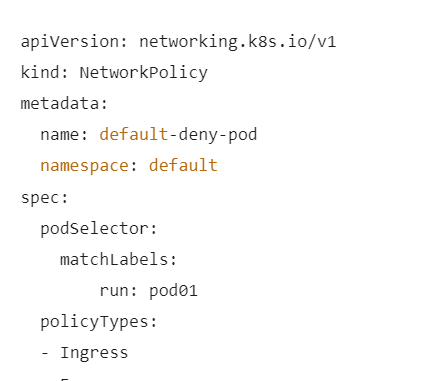
* **key: “key1” => the key**
* **operator: “Exits” => how to compare, for example we are interested in the existence of the key**
* **effect: NoSchedule => what is the effect**

73-



Request means the minimum resources we need to launch our pod and limit means the maximum resources we are allowed to take after it the pod will restart

74- In the below image any pod that match the label run:pod01 will have this network policy applied. Ingress mean we block comms fro inside, Egress mean we block them from outside, in our case all the pod that have the label run:pod01 will block all the traffic from inside



75- the objectif is to limit the container resources

Docker container … -m 256M => command used to limit memory usage

Docker container … -cpus=3 or cpus-set=1,2,3

--memory-reservation => a soft limit is used to reserve or to guaranty a memory allocation to the container

76- docker swarm ca –rotate => is used to issue another join token for the worker node

77-

Docker secret create secret\_name secret\_file => to create a secret

Docker service … --secret secret\_name => to bind a secret to a container or a service,normally the secret file will be stored in tempory file in our container operating system

78-

Container os can run a certain Linux functionalities, however other capiblities can be allowed by enabling them by using the –cap-add flag when crating the container

Privileged container will have all the linux capabilities to create those container simply: docker run --priviliged