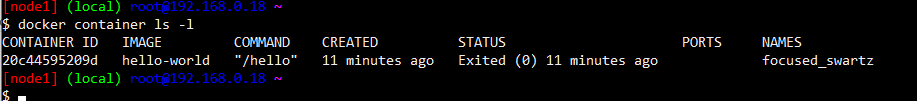
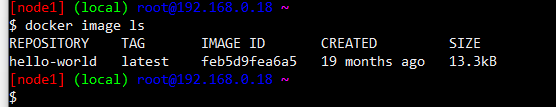
1. Run hello-world docker container and observe the container status.

Command ” docker container run hello-world”



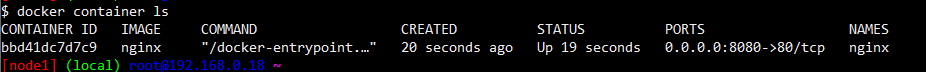
1. Check the docker images and also write down the size of hello-world image.

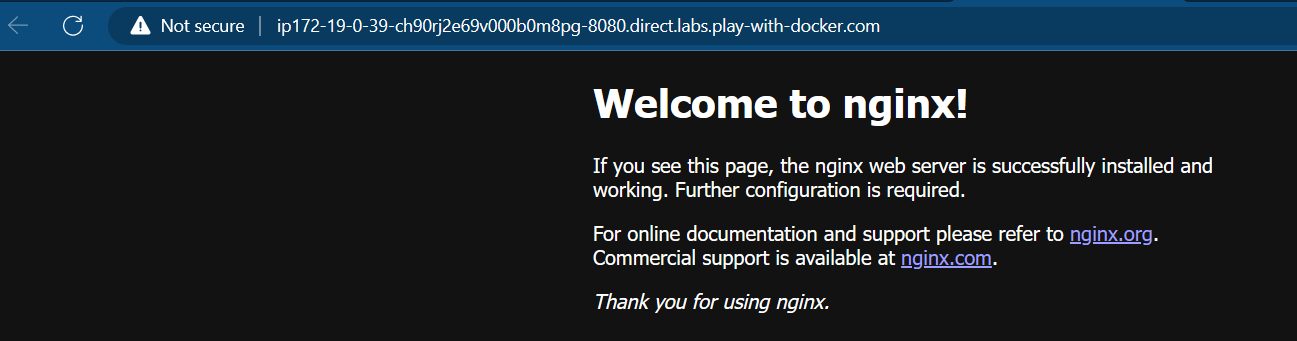
Command “ docker image ls “



1. Run the nginx container with name as nginx1 and expose it on 8080 port on docker host.

Command “ docker container run –-name nginx1 –d –p 8080:80 nginx “



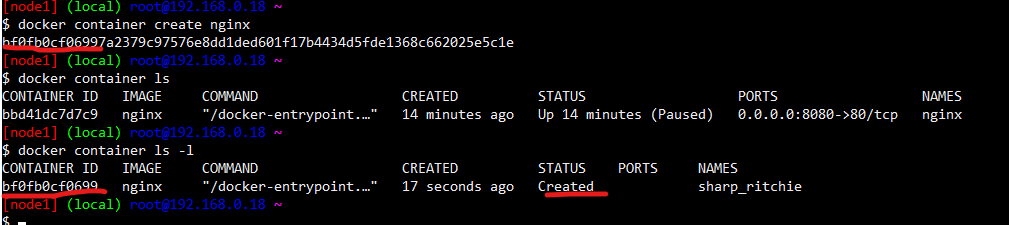


4. Explain docker container lifecycle

**Docker container lifecycle**

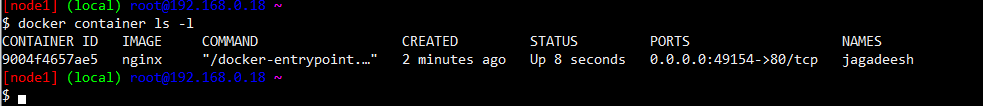
* Docker lifecycle states
  + Created

Command “ docker container create –name jagadeesh nginx “



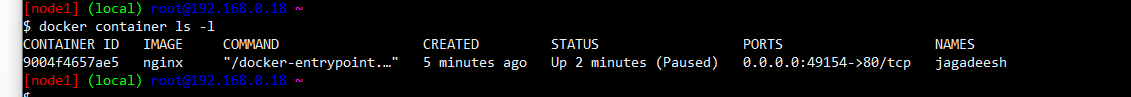
* + Running

Command “ docker container start jagadeesh “



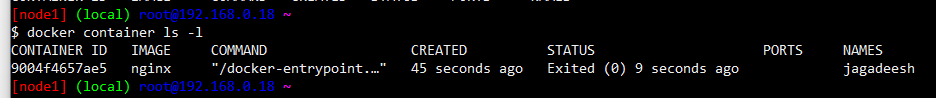
* + Paused

Command “ docker container pause jagadeesh “



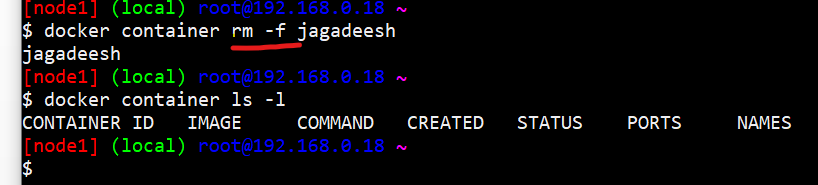
* + Stopped

Command “ docker container stop jagadeesh “



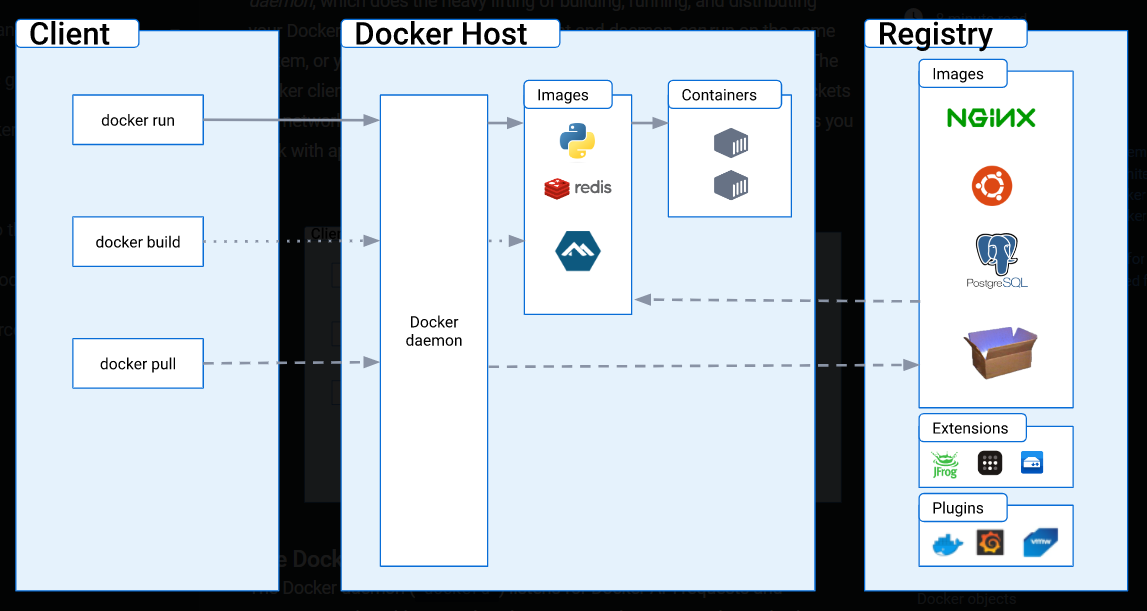
* + Deleted

Command “ docker container rm –f jagadeesh “



5. Explain what happens when you run the docker container.

* docker client will forward the request to docker daemon
* docker daemon will check if the image exists locally. if yes creates the container by using image
* if the image doesnot exist, then docker daemon tries to download the image from docker registry connected. The default docker registry is docker hub.
* Downloading image into local repo from registy is called as pull.
* Once the image is pulled the container is created.

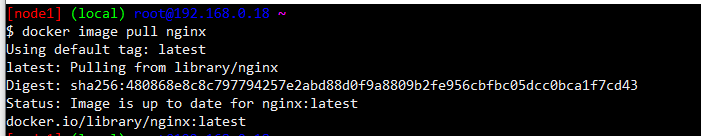


6. Explain the Docker Architecture

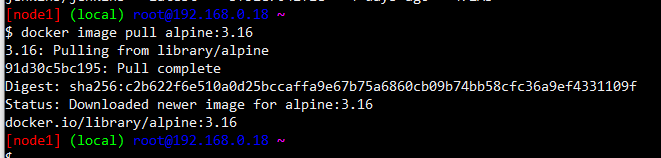
Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

7. How to pull an image

Command “ docker image pull <name of image>” (latest image)

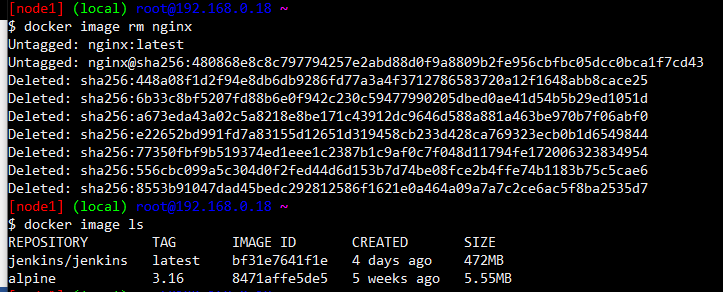


Command “ docker image pull <name:tag>” (lmage with required version)

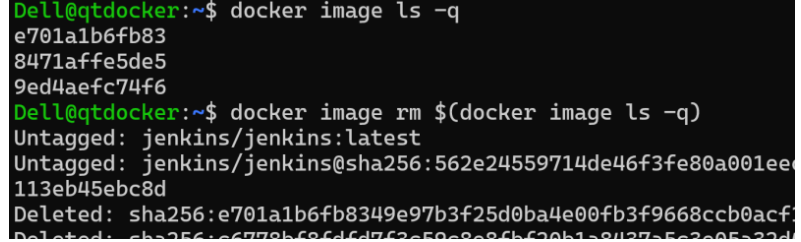


8. Delete an image

Command “docker image rm <name>”

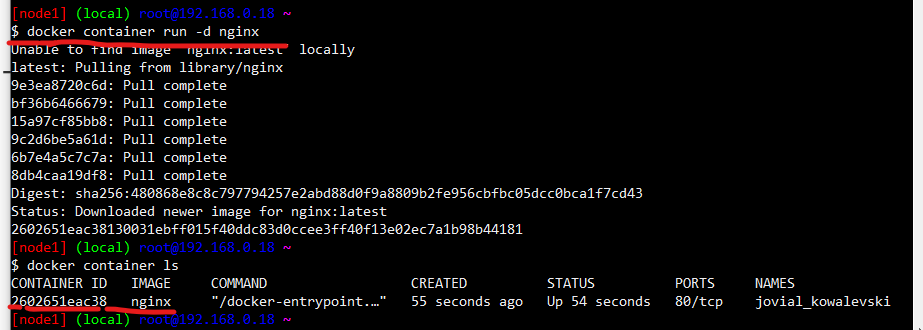


if i have to delete all the images `docker image rm $(docker image ls -q)

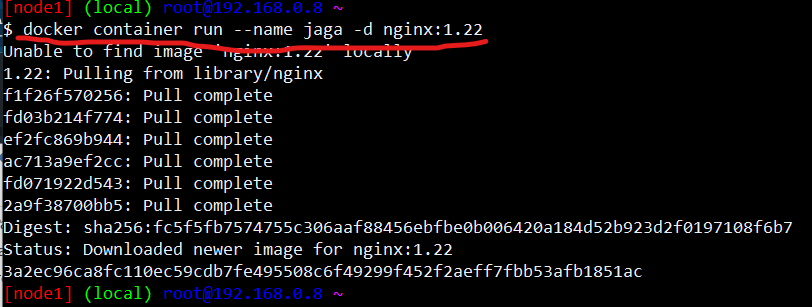


9. create and run a container

“ docker container run –d <name of image>”



“docker container run --name jaga –d nginx:1.22”

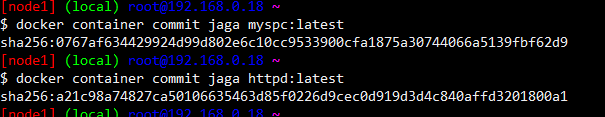


Remove all containers “docker container rm -f $(docker container ls -a -q )“

Remove specific container “ docker container rm –f <name of container>”

10. create image from container

docker container commit <container name> <image name:tag>



11. Build an image from the dockerfile.(base image model)

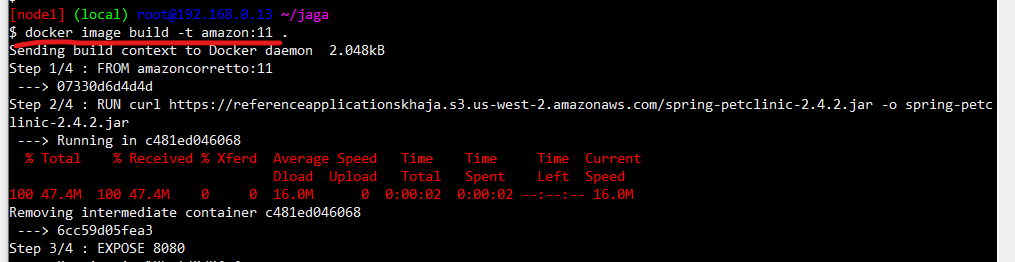
FROM amazoncorretto:11

RUN curl https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar -o spring-petclinic-2.4.2.jar

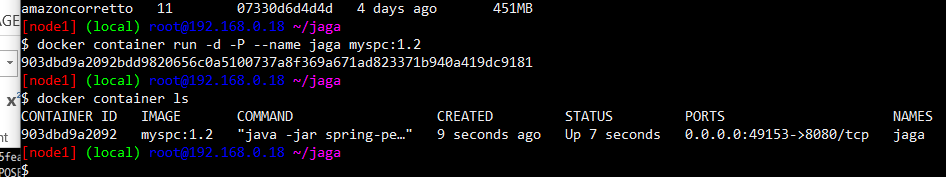
EXPOSE 8080

CMD [ "java", "-jar", "spring-petclinic-2.4.2.jar" ]

docker image build –t <image:tag> <path of file>



Lets run the container docker container run -d -P --name jaga myspc:1.2



12. Build a docker image from file and inspect the image

Docker file

FROM amazoncorretto:11

LABEL author="Jagadeesh"

LABEL purpose="practice"

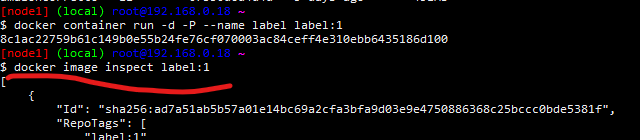
LABEL class="joip"

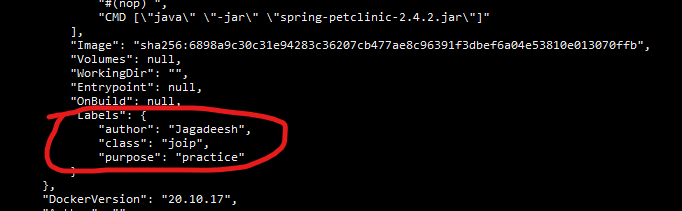
RUN curl https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar -o spring-petclinic-2.4.2.jar

EXPOSE 8080

CMD [ "java", "-jar", "spring-petclinic-2.4.2.jar" ]

Inspect the image “docker image inspect label:1”





13. To install apache server and tomcat9

“sudo apt update”

“sudo apt install tomcat9”

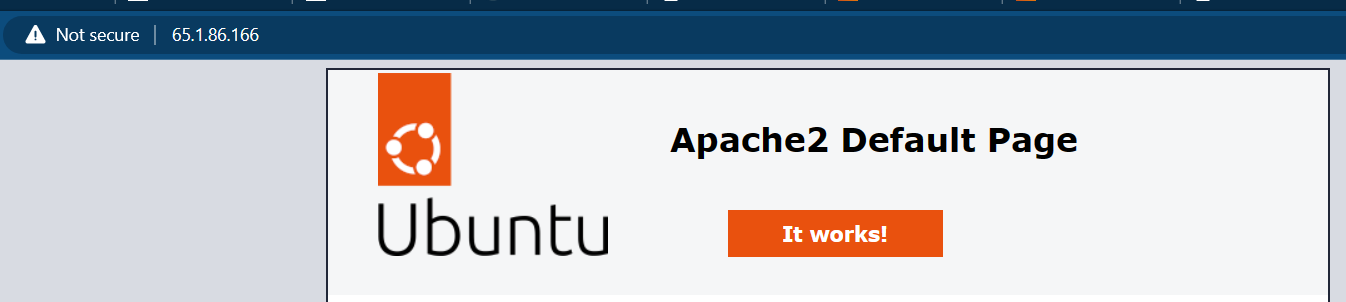
“sudo apt install apache2”

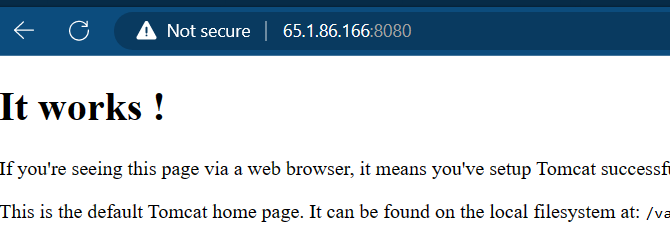
“sudo systemctl start <server\_name>” (To start the server)

“sudo systemctl stop <server\_name>” (To stop the server)

“sudo systemctl restart <server\_name>” ( To restart server)

“sudo systemctl status <server\_name> ( to check the status of the server)





14. Running nopCommerce with file.

FROM mcr.microsoft.com/dotnet/sdk:7.0

LABEL author="khaja" organization="qt" project="learning"

ADD https://github.com/nopSolutions/nopCommerce/releases/download/release-4.60.2/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip /nop/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip

WORKDIR /nop

RUN apt update && apt install unzip -y && \

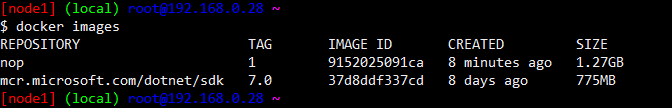
    unzip /nop/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip && \

    mkdir /nop/bin && mkdir /nop/logs

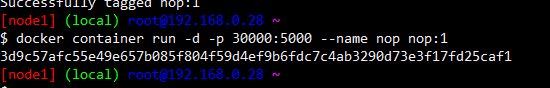
EXPOSE 5000

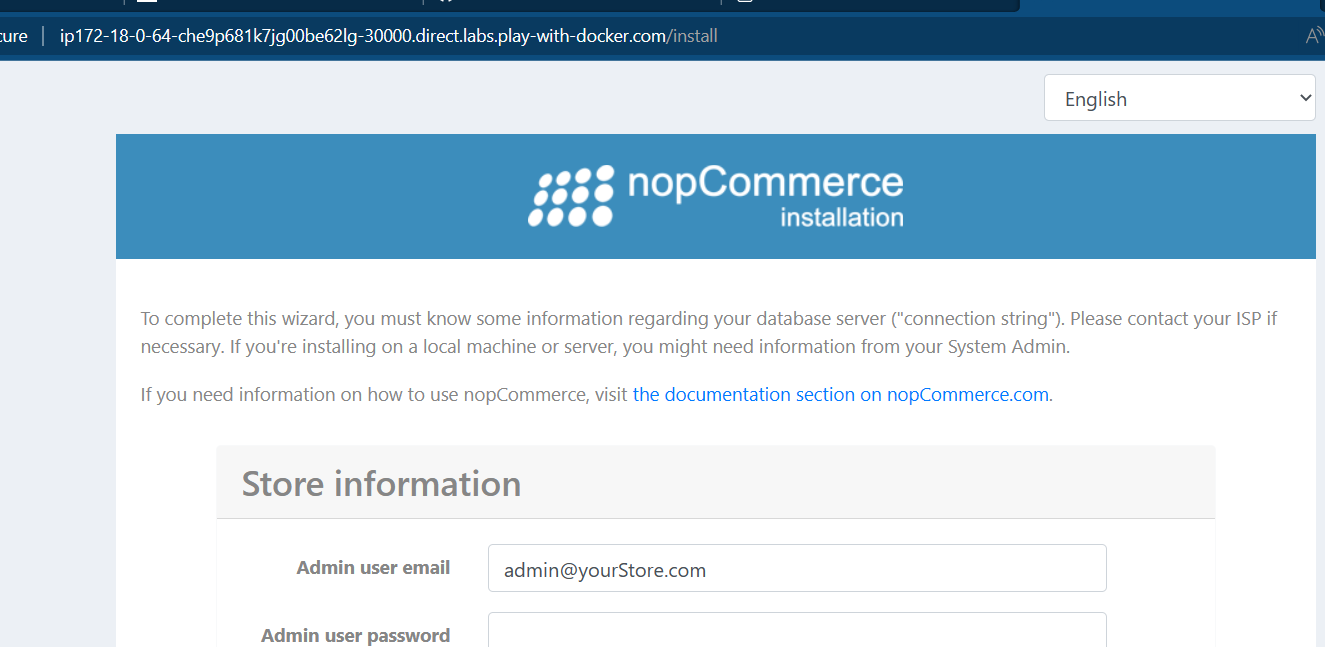
CMD [ "dotnet", "Nop.Web.dll","--urls", "http://0.0.0.0:5000" ]

Build image from file “docker image build -t nop:1 .”



Create a container from the image “docker container run -d -p 30000:5000 --name nop nop:1 “





15. added ARG and ENV for nopcommerce

FROM mcr.microsoft.com/dotnet/sdk:7.0

LABEL author="khaja" organization="qt" project="learning"

ARG DOWNLOAD\_URL=https://github.com/nopSolutions/nopCommerce/releases/download/release-4.60.2/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip

ARG HOME\_DIR=/nop

ADD ${DOWNLOAD\_URL} ${HOME\_DIR}/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip

WORKDIR ${HOME\_DIR}

RUN apt update && apt install unzip -y && \

    unzip ${HOME\_DIR}/nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip && \

    mkdir ${HOME\_DIR}/bin && mkdir ${HOME\_DIR}/logs

EXPOSE 5000

ENV ASPNETCORE\_URLS="http://0.0.0.0:5000"

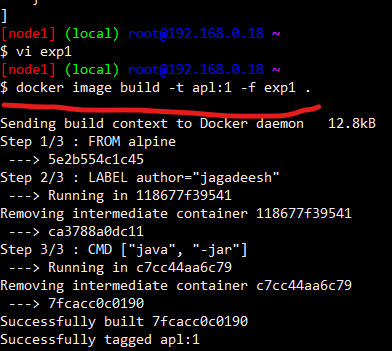
CMD [ "dotnet", "Nop.Web.dll"]

16. Build args can be set while creating images. BUILD ARG can be used by using ${ARG\_NAME}.

docker image build --build-arg DOWNLOAD\_URL=nopCommerce\_4.60.2\_NoSource\_linux\_x64.zip –t nop:1.0.2 .

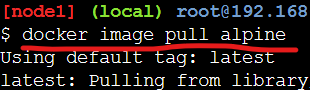
docker image build --build-arg HOME\_DIR=/publish -t nop:1.0.0 .

17. Docker image build –t <name:tag> -f <file name> (to build from a file which is not Dockerfile)

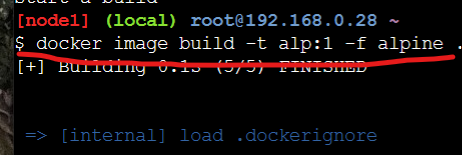


18. create a new docker image from existing image and inspect the layers of the images

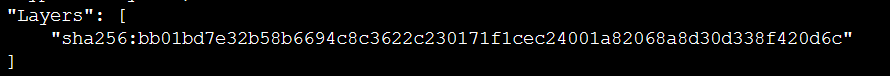
* Pull an alpine image



* Now write a new file from alipine and build an image
* From: alpine
* LABEL: method1:layers
* CMD: ["sleep" , "1d"]



* Compare layers of both the images



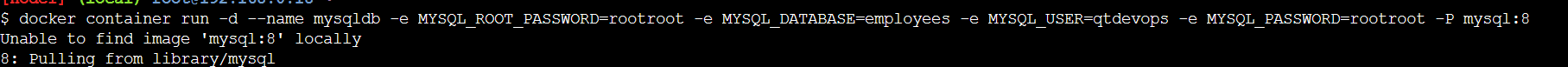


* Both have the same layer

### 19. Stateful Appplications and Stateless Applications

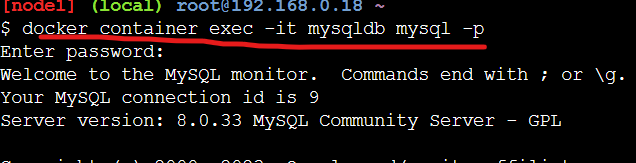
* Stateful applications use local storage to store any state
* Stateless applications use external systems (database, blobstorage etc) to store the state
* We need not do anything special if your application is stateless in terms of writable layer, but if it stateful we need to preserve the state.

20. create a MYSQL container with environment variables to store data in stateless method

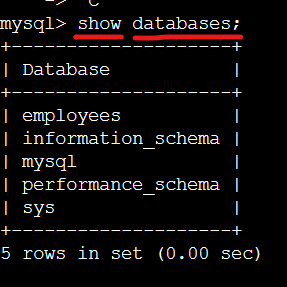


* To login into container

password>docker container exec -it <nameof container> <image> --password=<root-password>



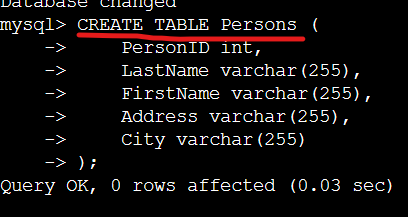
* To see the list of databases



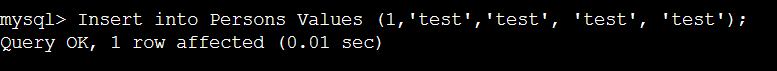
* To move to a specific table



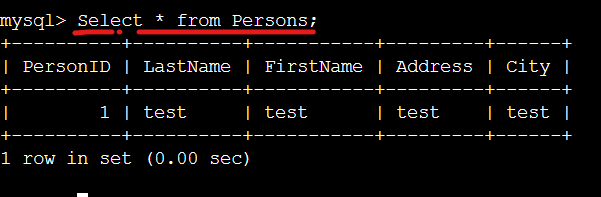
* Create table in database with some headings.
* “CREATE TABLE <name>”



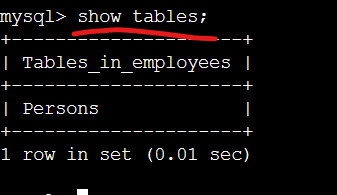
* Enter data into the table



* To see full data of persions table

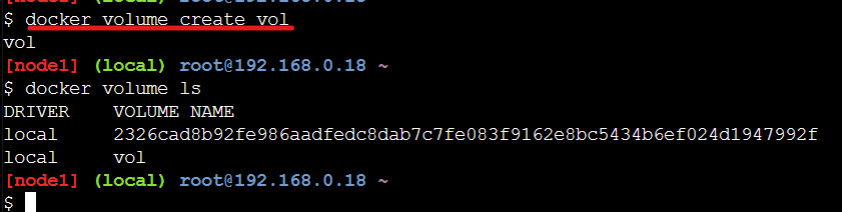


* To see list of tables in a database



21.Docker volumes

* To create a volume



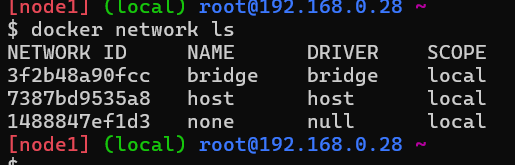
* Attach a volume to a container



Docker network

* To see list of networks

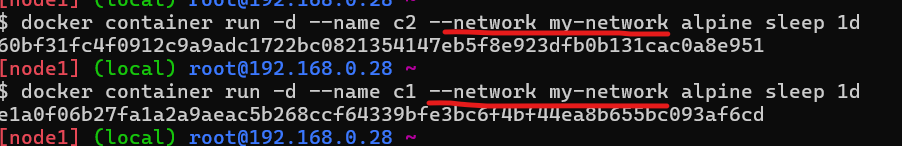
“ docker network ls”



* Above are default networks
* When a container is created , it is connected to bridge network by default
* Create a network
* docker network create -d bridge --subnet "10.0.0.0/24" <your network name>

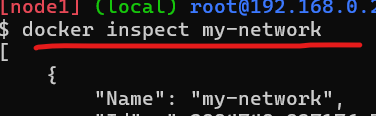


* create two containers with name’s c1 and c2 on same network .check the connection between them with ping .

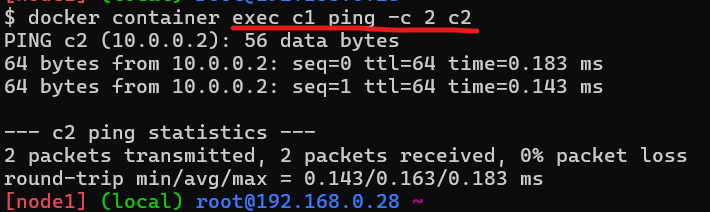


* inspect my-network

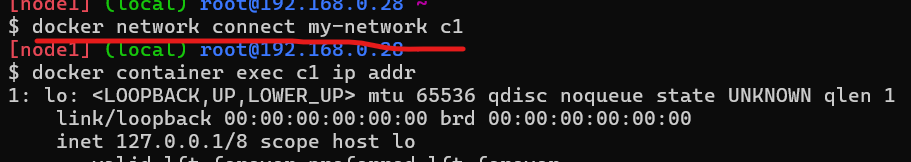
“docker inspect <name of network>”



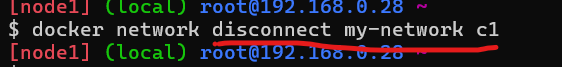
* check the network between two containers
* when containers is connected over one network , connection between the containers is checked by name of the containers or ip address



* connect a container to an existing container



* disconnect a container from a network



## 23.Docker Swarm (Multi-Host Networking)

## To become a manager node execute the below command

## “docker swarm init --advertise-addr <ip of your node> ”

## 

## The highlited yellow color command should be executed in other nodes which wanted to join as workernodes

## After creating a swarm cluster in manager node , a network get’s created

## When a worker node is joined in swarm , the newly created network is added to the worker

## 

## Execute “ docker node ls” in manager node

## 

## The star indication is for leader node and other is nodes joined cluster

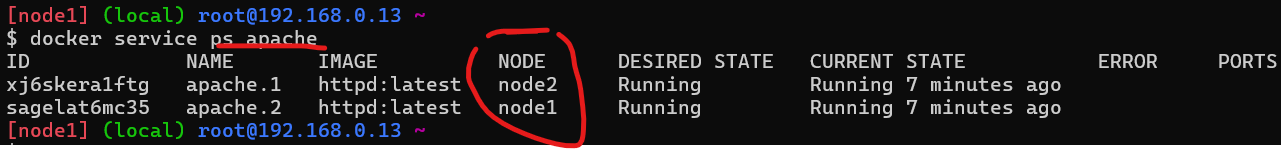
## Create apache services with two replica’s in manager node

## 

## To remove a service

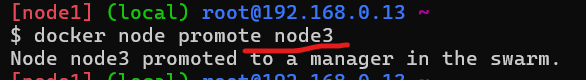
## 

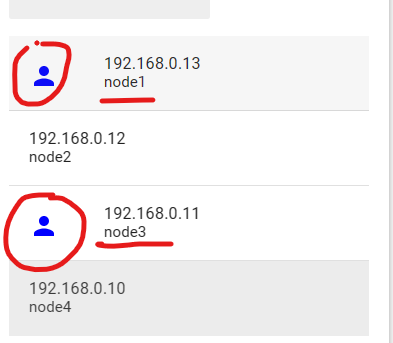
* Execute docker service ps tomcat command to findout on which node the tasks are executed.



* Lets scale the number of containers running tomcat by using the following command docker service scale tomcat=4 and the output would be
* To have more then one manager node is done by promoting the worker node to manager node.

“ docker node promote <node name>”





Now I have two manager nodes