**What is docker?**

Docker (dock worker) is used to create containers which is standard way of packaging any application

**What is docker image?**

If the application can be developed in any sever by the standard way of packaging

### Docker installation

To install docker there are two components.

Docker client

To interact with docker engine to use command lines

Docker engine

In this three components are there

Orchestration

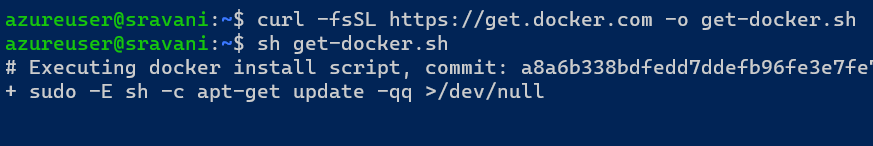
It is used for docker working in multi system.

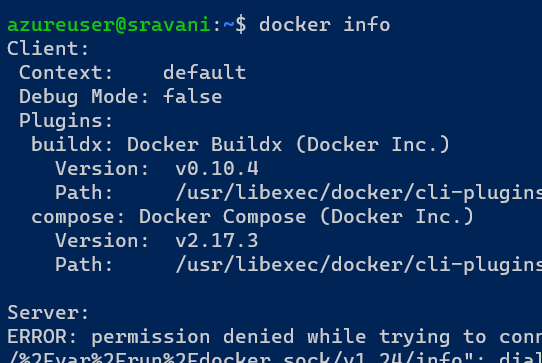
Docker daemon

It gives a way to communication.

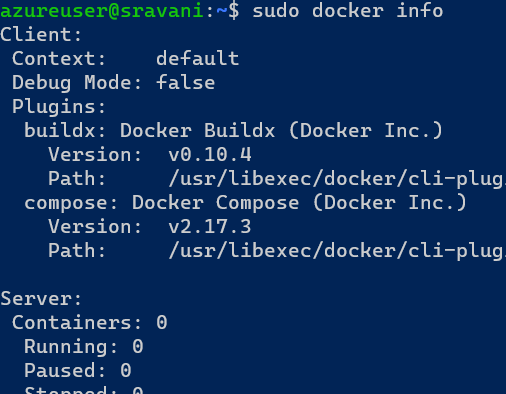
Runtime

It runs the low level activities.

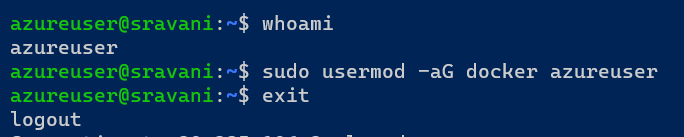
Installation of docker in VM 



By observing above client is working but server is not working to work the sever to give the following command



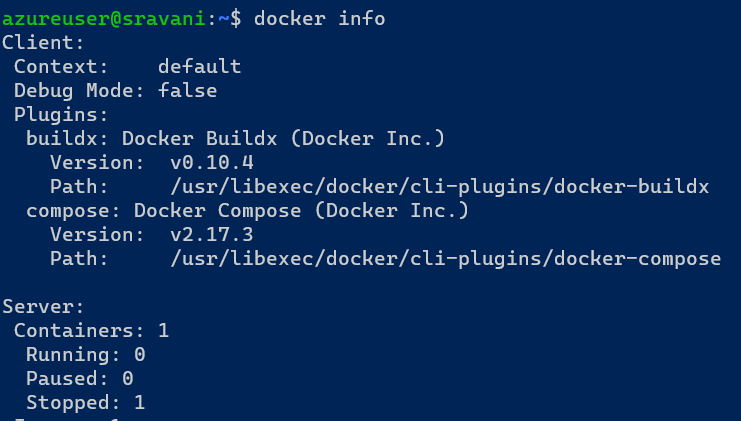
Every time sudo is good so switch to user into docker

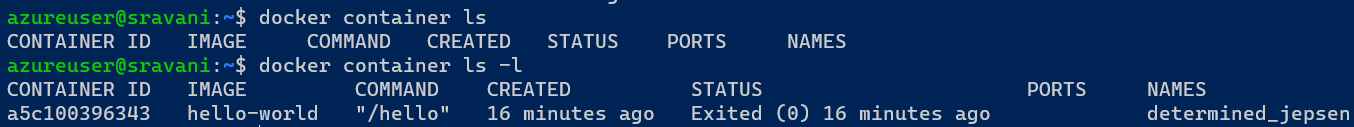




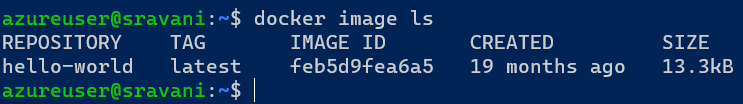
To check the docker working or not it’s a small example





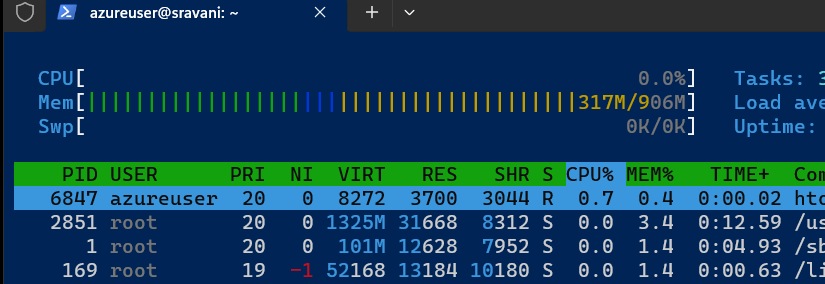


Check the docker image size

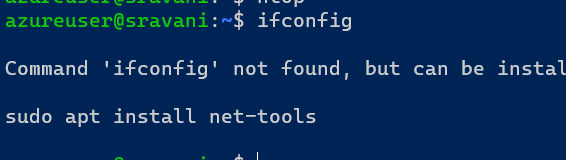


To see the status of disk by using command “htop”

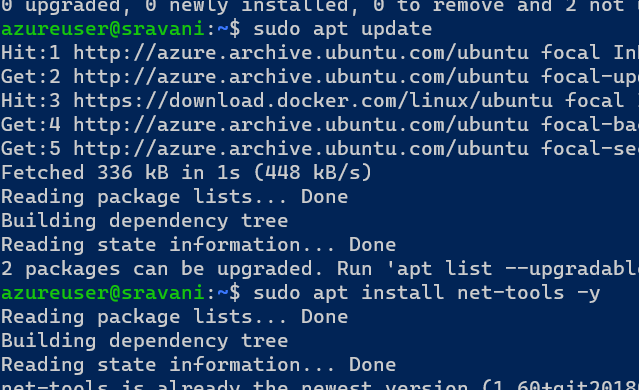
CPU, Memory and other we will see the below

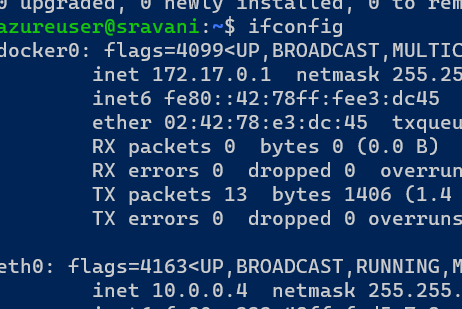


To see the network interface

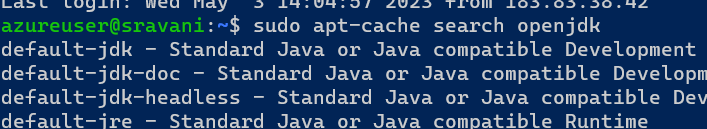


Now update the tools





To search java

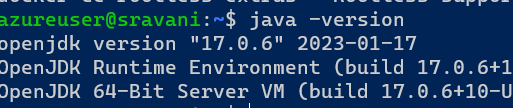


To install java

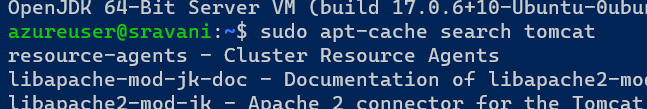


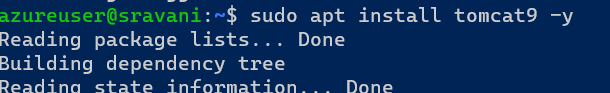
Similarly search and install dotnet and python also

To check version



To install tomcat

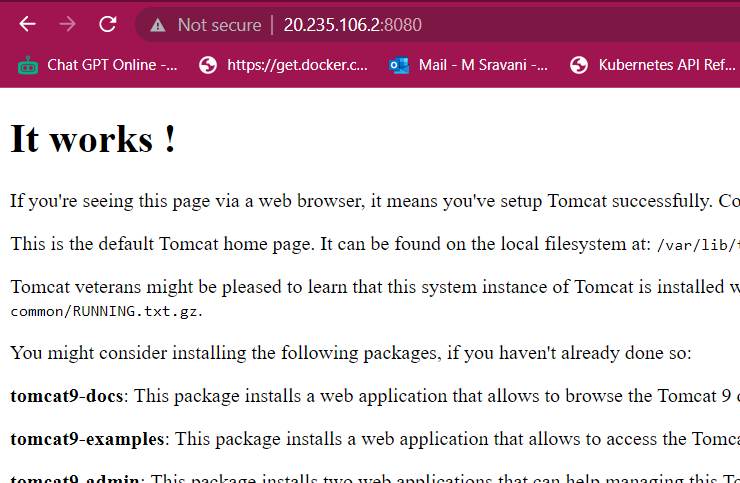




Here change the port to 8080 because tomcat port is 8080

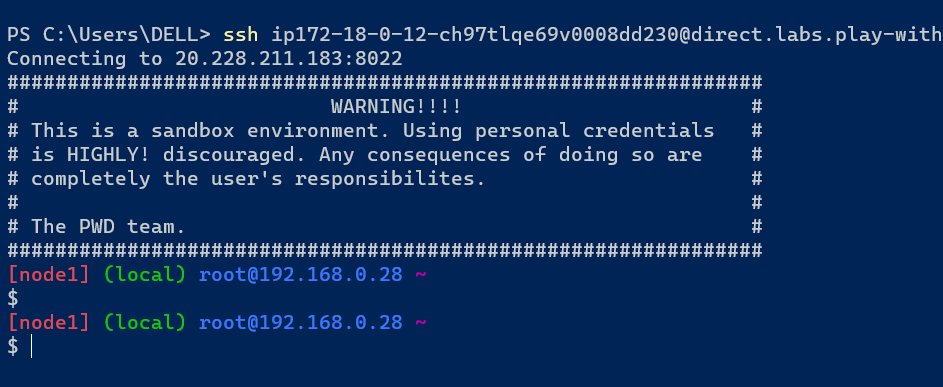


To go to google and enter ipaddress:port

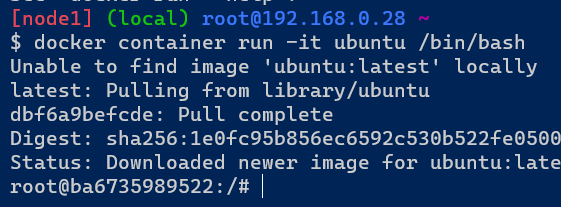


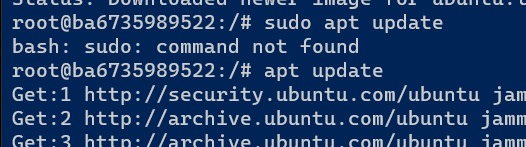
Connecting to docker playground

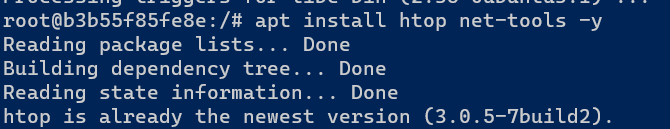




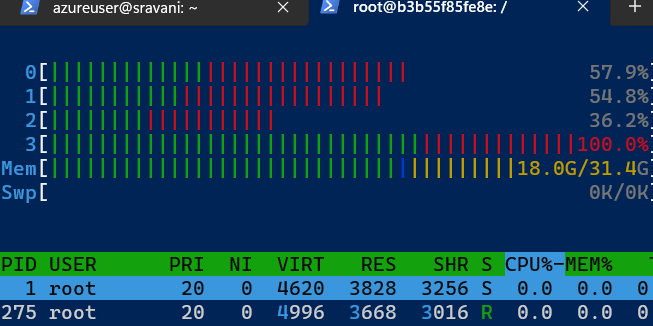
To run the container

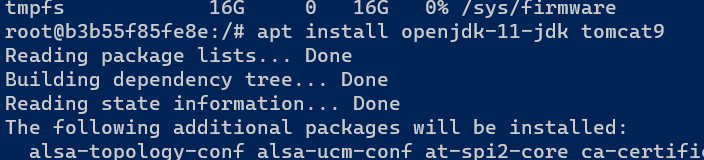




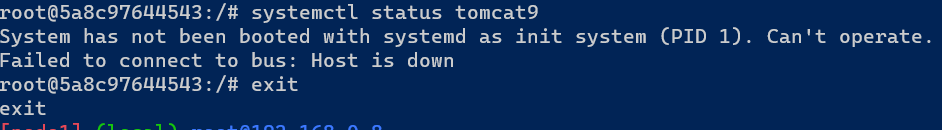


To see the status of disk by using command “htop”



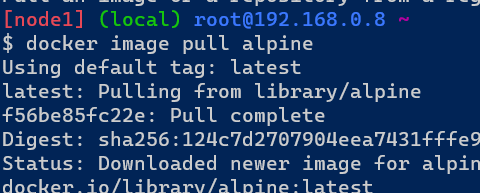


To check the tomcat9 status

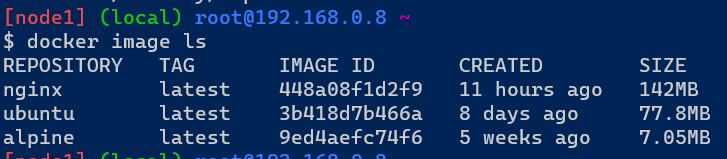


The host is down then what we do

To pull the image or create



To check the size of image



To create docker container it getting following components

new process tree

disk mounts

network (nic)

cpu/memory

Users

##

**### DOCKER ARCHITECTURE**

Generation 1:

Docker client send the request to docker daemon were commands interact with docker.It has to be relay on to create lxc (Linux)container.this internally uses name spaces and c groups.It is inbuilt kernel is Linux.

Generation 2:

In this generation updating kernels and frequently creates lib containers.lib container is docker component. It has own code to run the applications.Libc directly speak with kernel.

Generation 3:

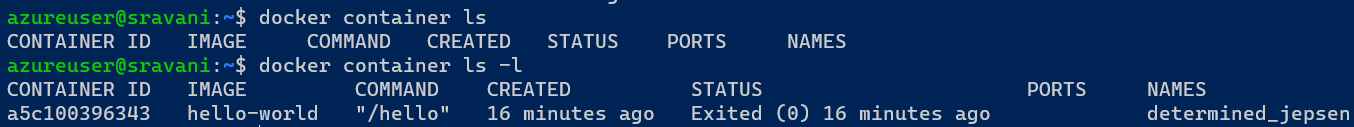
In this generation docker client specks with docker daemon.Docker daemon expose api and it speaks with containerd. This containerd creates container by any container technology that is OCI. The containerd speaks with runc.

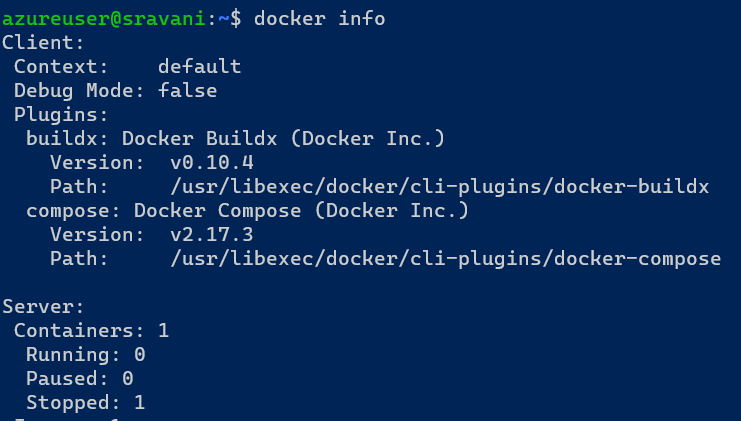
### Run hello-world docker container and observe the container status

Docker client send to the request to docker daemon. Docker daemon checks the image and create the container by using local repository.

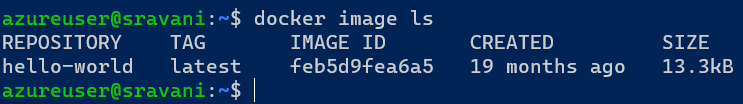


Docker container status

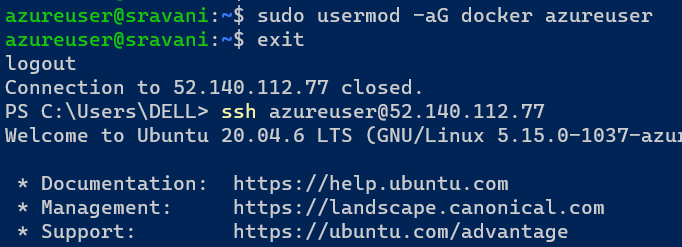




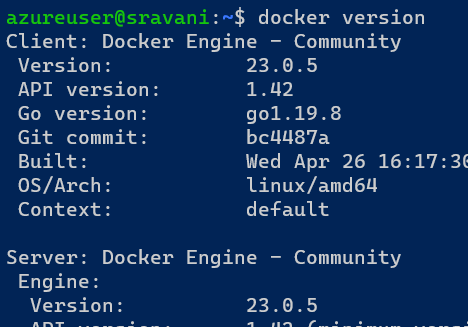
Check the docker images and also write down the size of hello-world image



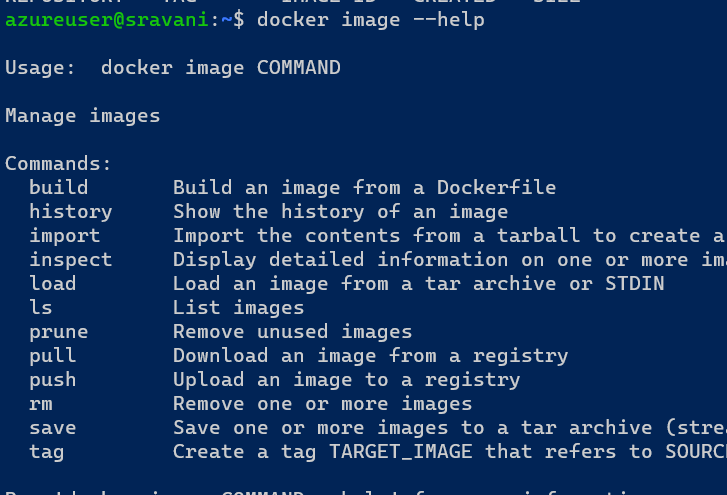
To add user to the docker group



Now check the version

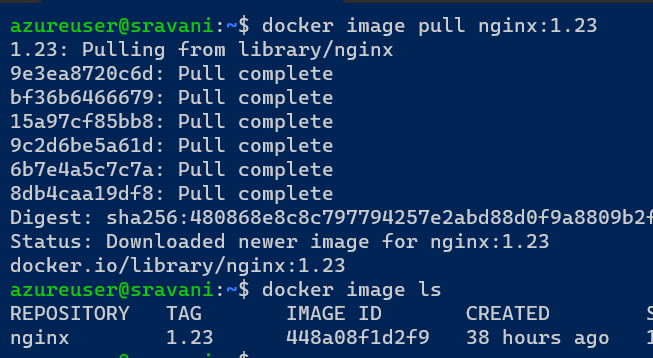


To check the images



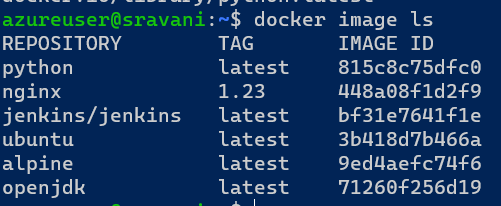
Similarly docker container, docker network, docker compose.

To pull the image

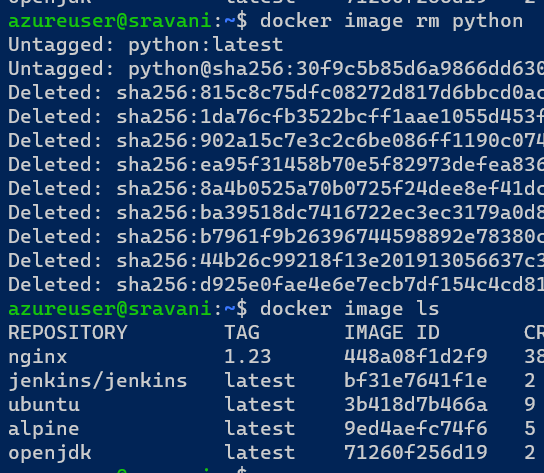


To pull the image “ docker image pull <image name> “

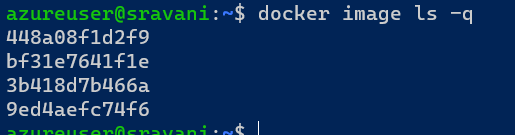
To see the all images in docker container by using “ docker image ls “ command.



To remove all images “ docker image rm <image name> “

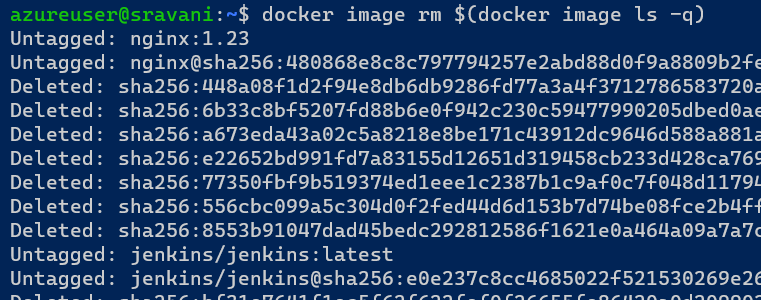


To check the image ids.

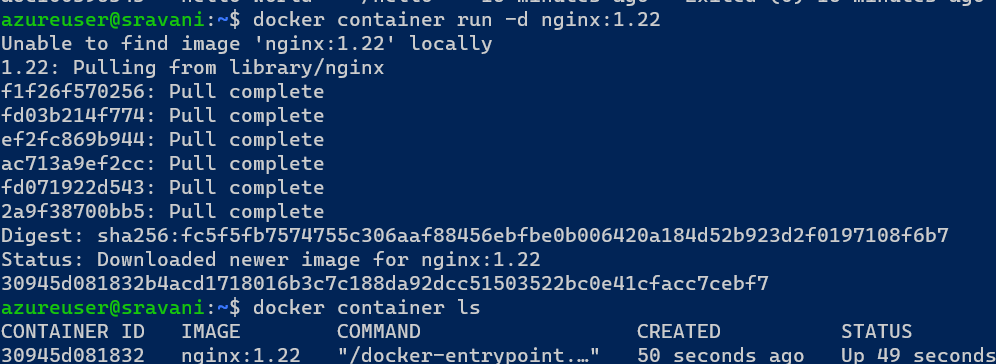


To remove all docker images

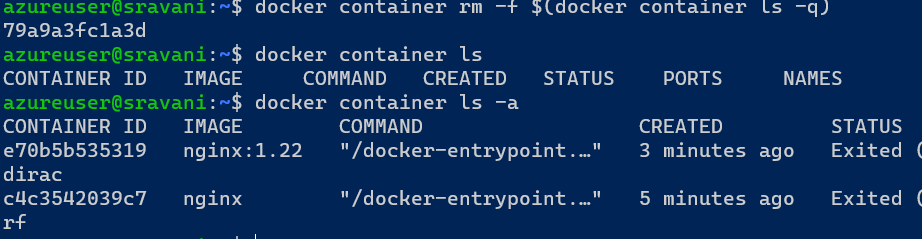
docker image rm $(docker image ls -q)



To create a container



To remove the container



### **Docker lifecycle**

Docker lifecycle has 6 states they are

Create

Run

Stop

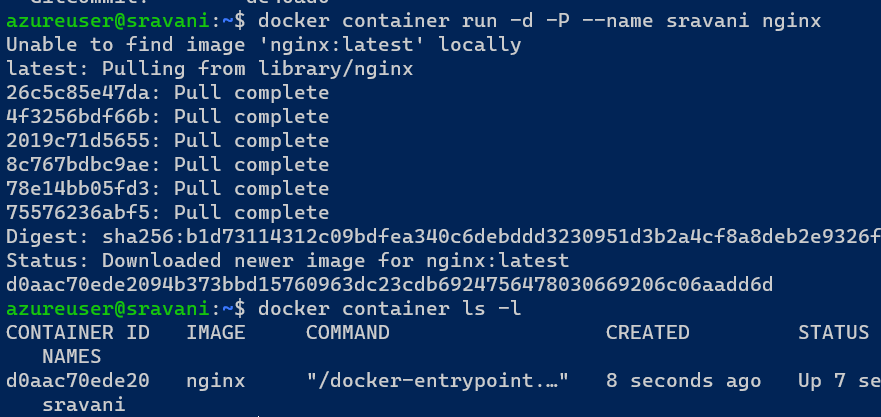
Pause

Unpause

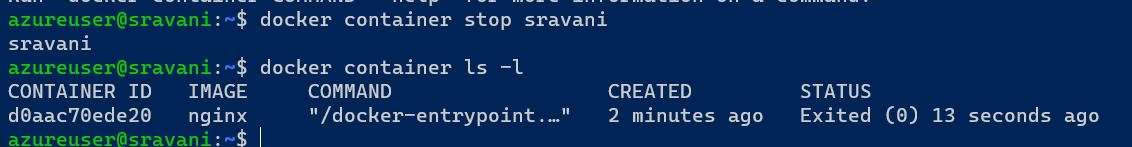
Delete

Above states explained with example below

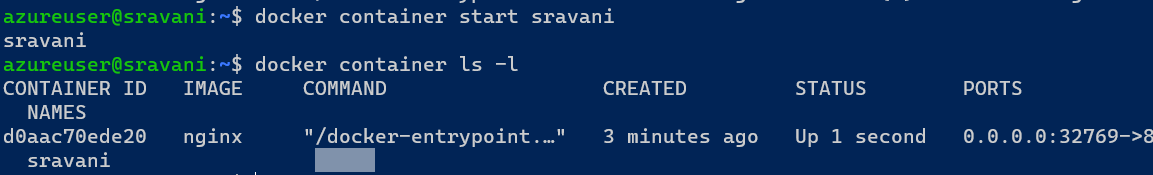
To create and run



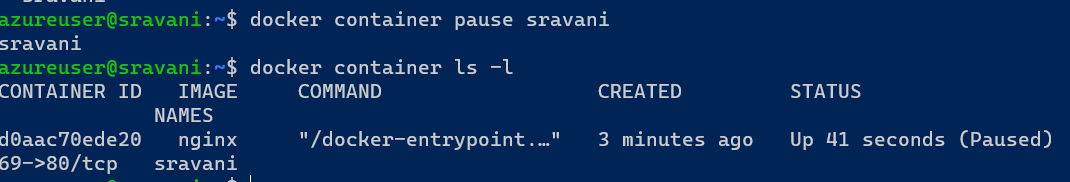
To stop



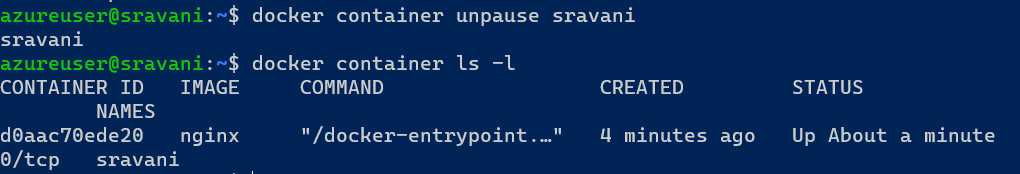
To start



To pause

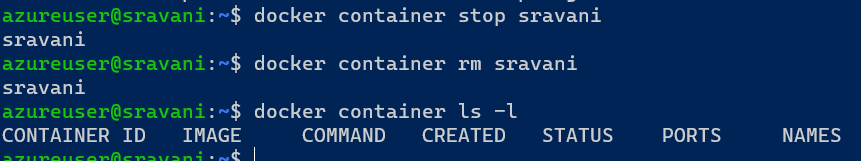


To Unpause



To delete

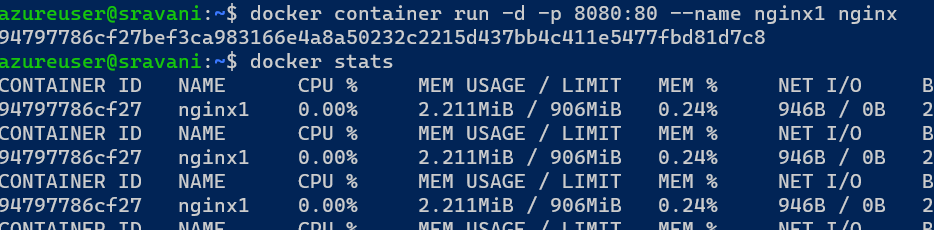
If container I running it is not possible to delete so we will stop the container and delete

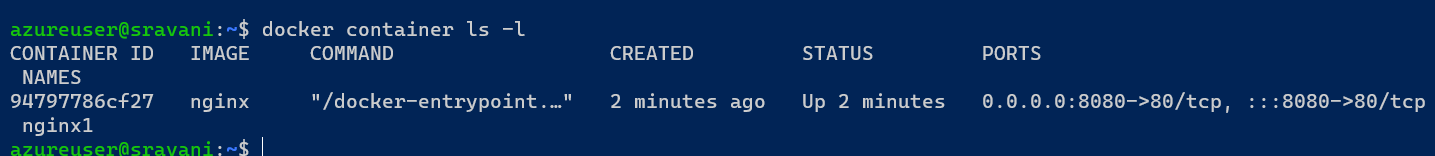


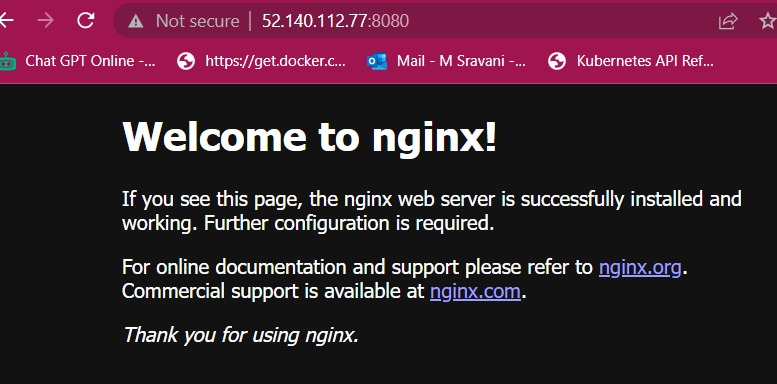
To run the container by using host port by using this command

command: docker container run -d -p <host-port>:<container-port><image name>

Ex: Create nginx container and expose port 8080

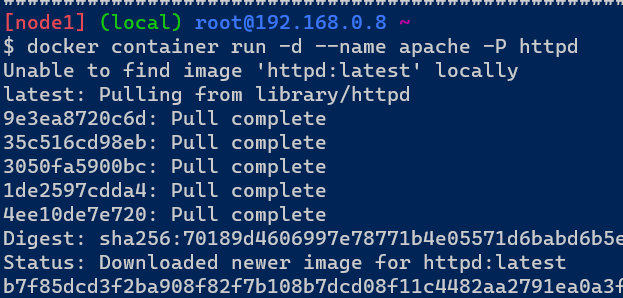


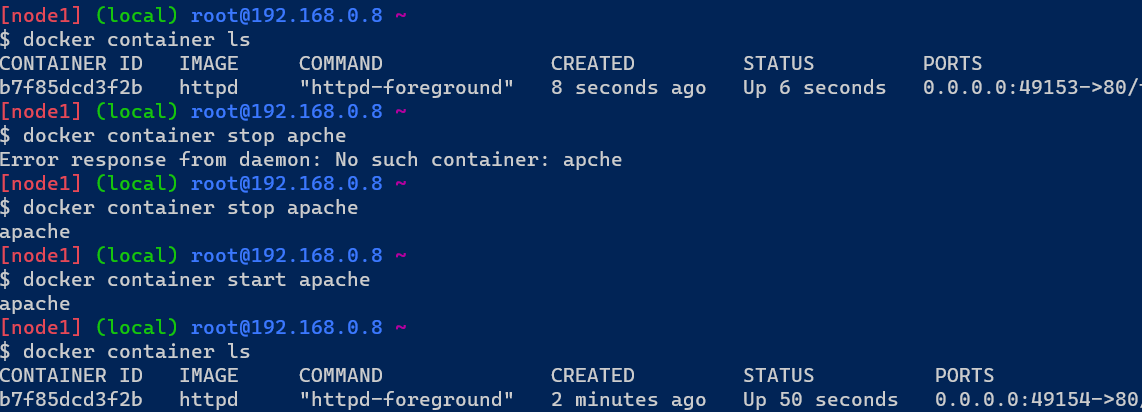


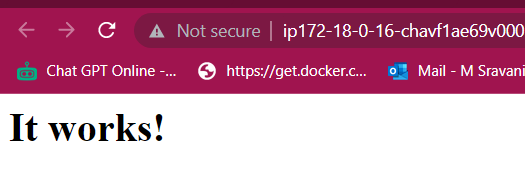


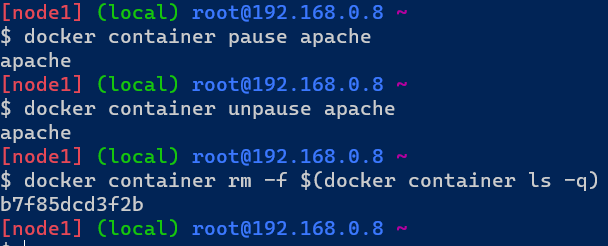
### **Exercise**

1. Install docker on a linux vm
2. Run 1 httpd containers (apache container) which runs on 80 port
3. try accessing any application
4. stop the containers
5. try accessing
6. start the continers and access this should work
7. pause the containers, access the application
8. unpause the containers, access the application
9. delete the container









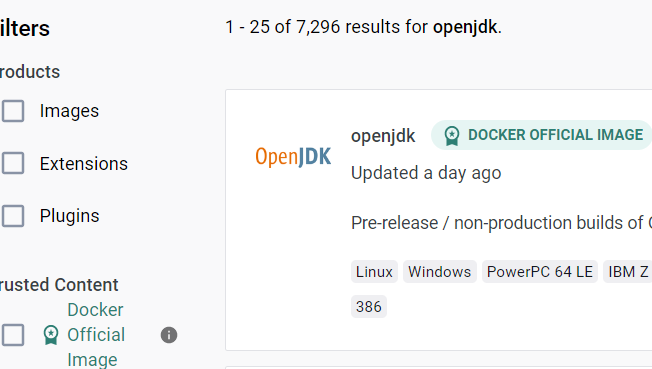
### **spring petclinic**

We need to run this application

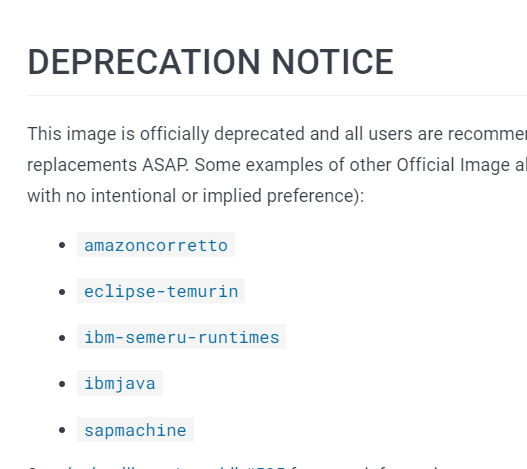
Jdk 11

Jar file

For jdk 11 go to docker hub and search openjdk

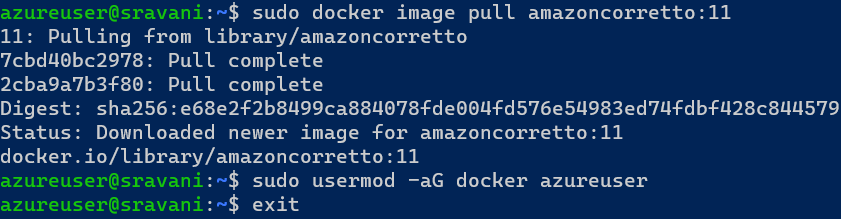


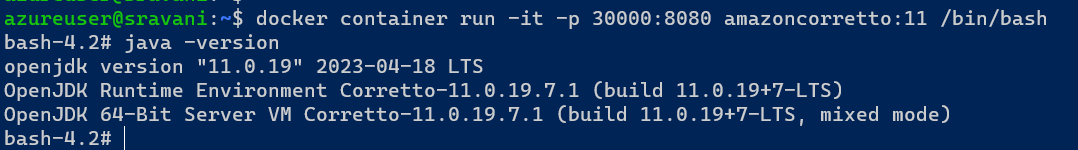
Select any on of the following

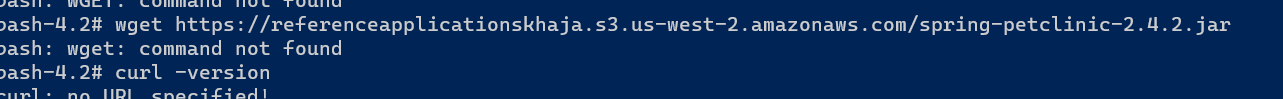


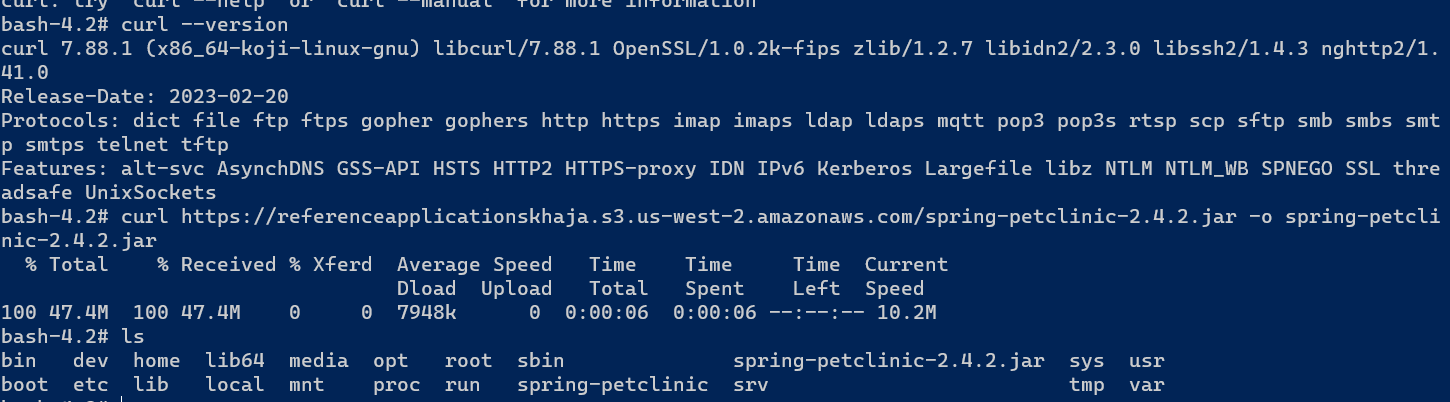
Open amazoncorretto and select the version





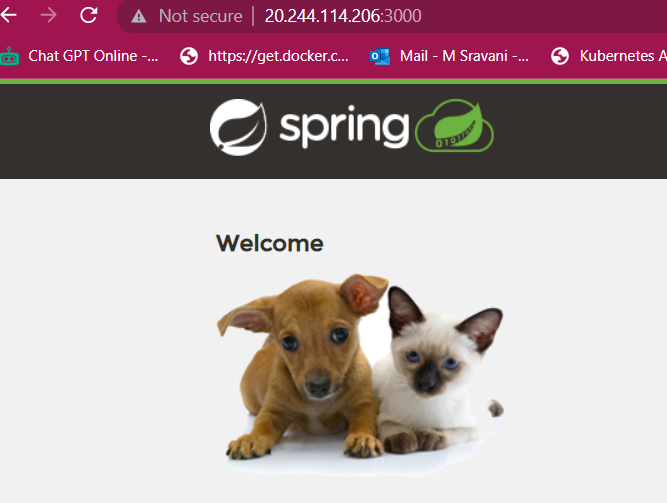




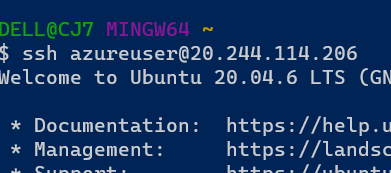


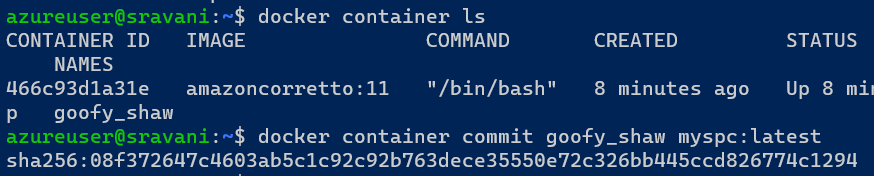
Give the port 8080 and 3000 on VM.

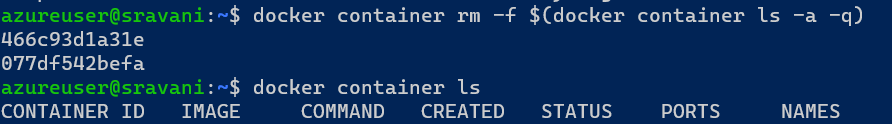




Now login to another terminal from same VM for running application.







**Docker file based image build**

Work flow

Source code in git

Create a docker file to containerized application

Docker image build

Docker image in a local repo

Push

Docker registry

Docker

Docker container

**What is docker file?**

**Docker file is a simple text file that consist of instructions to build a docker image.**

FROM baseImage

LABEL key="value"

RUN command

CMD [ "executable" ]

ADD source dest

COPY source dest

WORKDIR /the/workdir/path

EXPOSE port

ENTRYPOINT [ "executable" ]

ARG name

ARG name=defaultValue

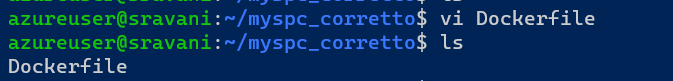
ENV key=value

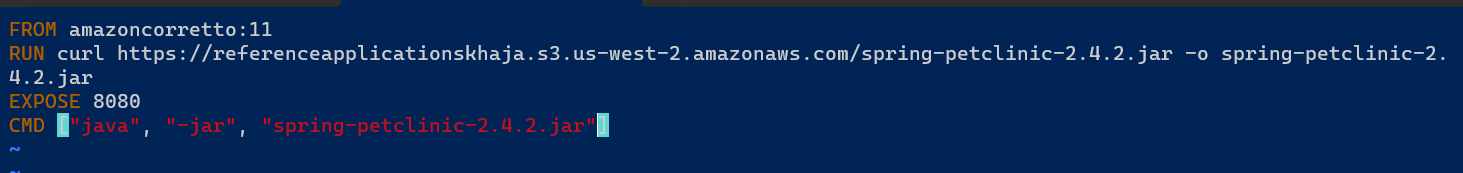
VOLUME [ "/data" ]

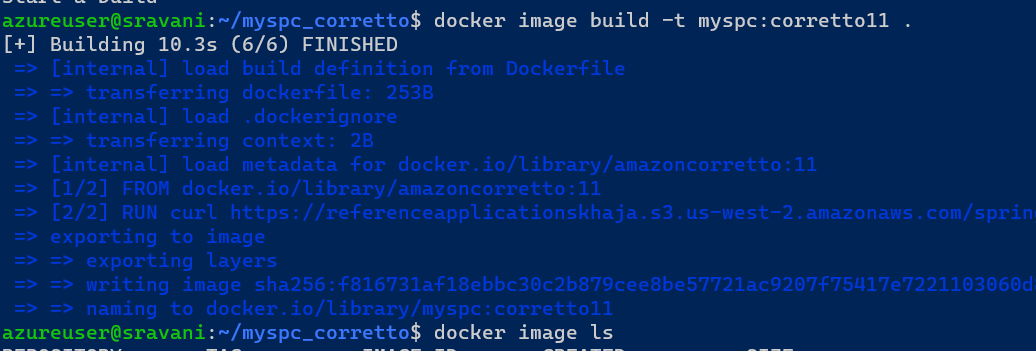
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 CMD [ "executable" ]

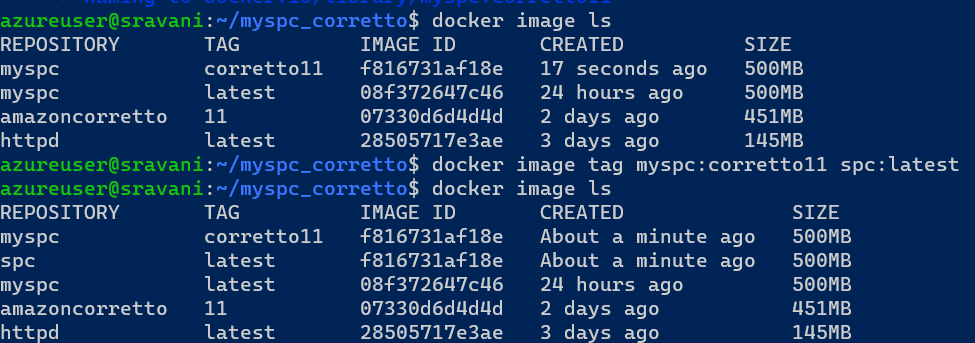
### **spring petclinic with docker file**

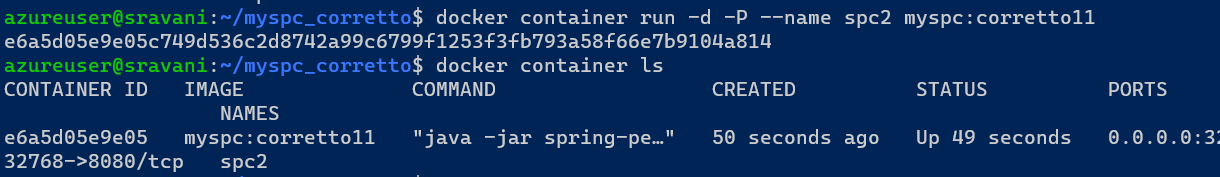


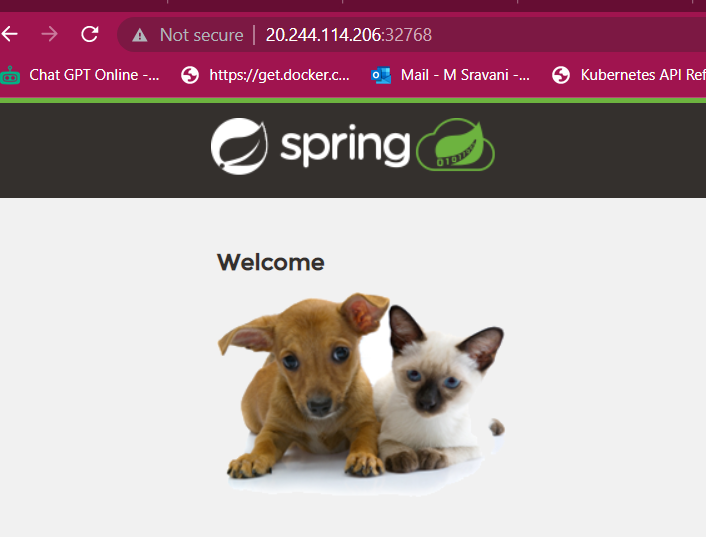




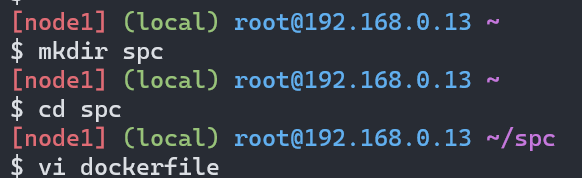


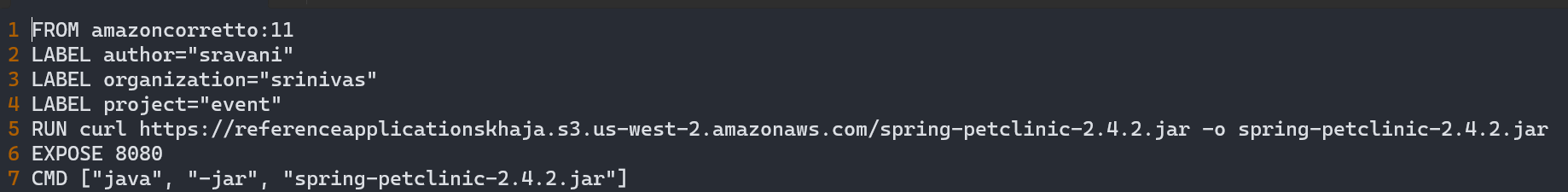


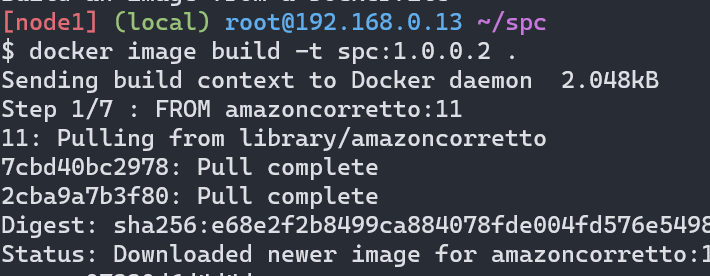


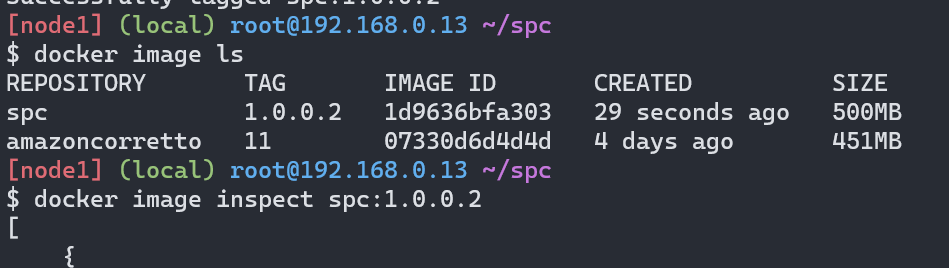


Another example

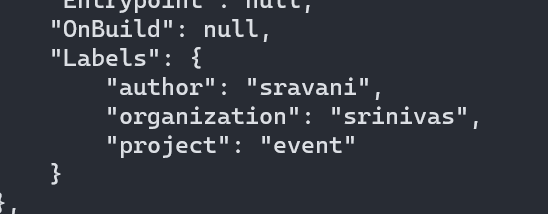




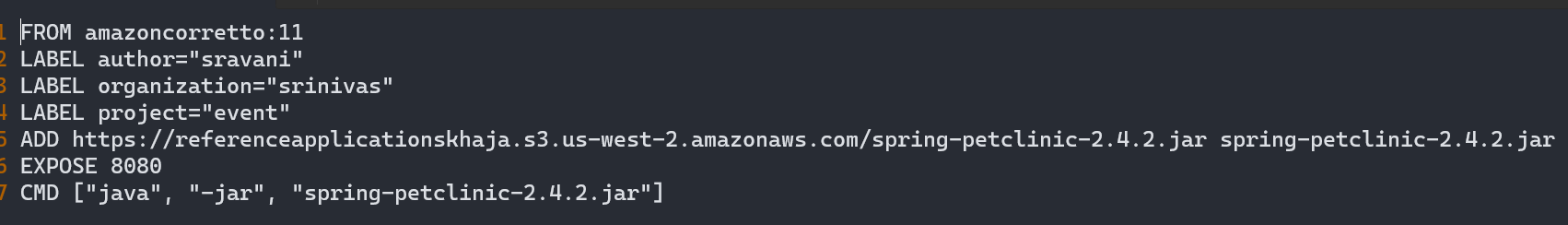


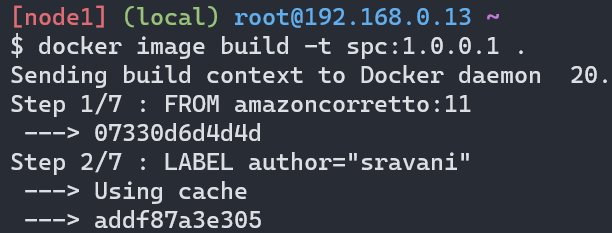


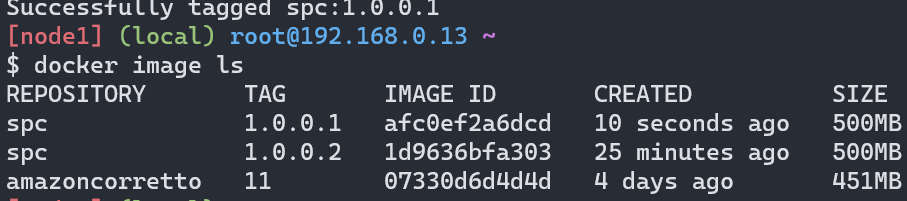
To check the labels what I gave



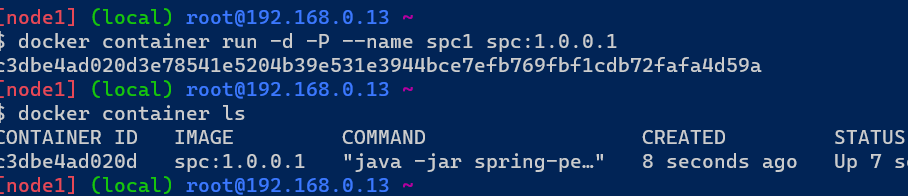
Add the ADD instruction in docker file

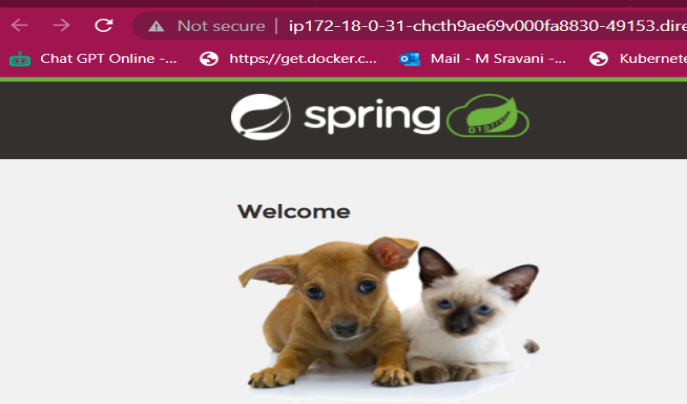


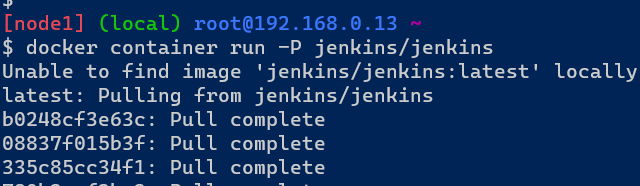


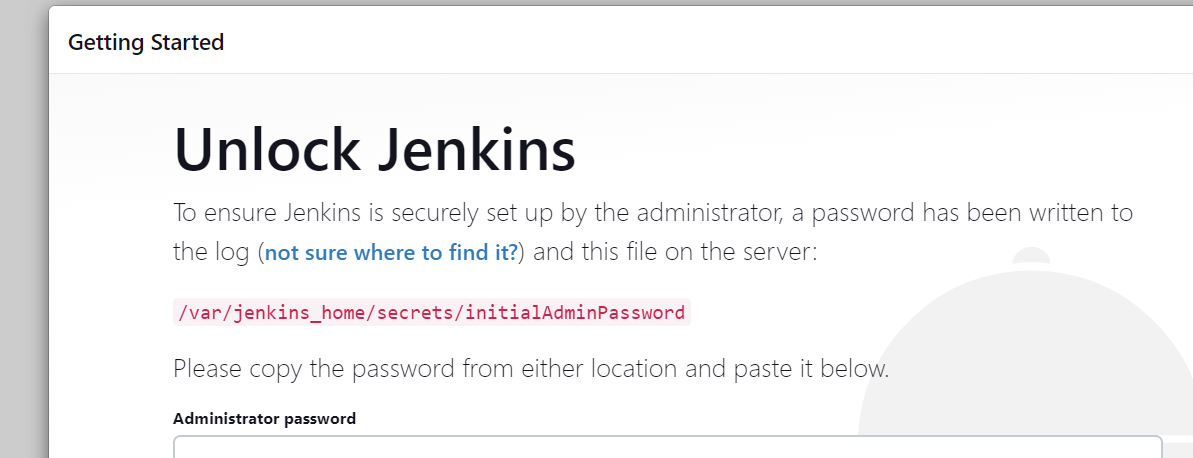


* ADD instruction can have sources
  + local file system
  + git repo
  + url
* COPY supports only local file system

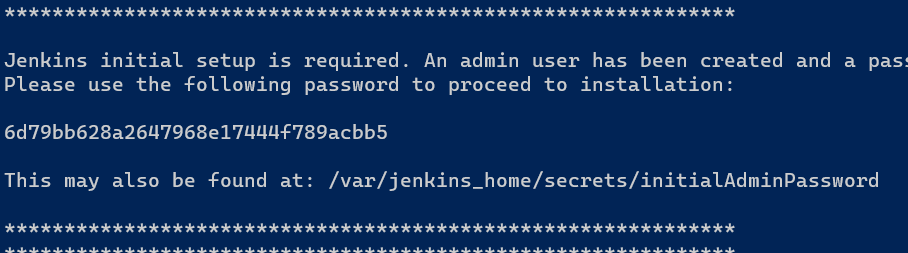


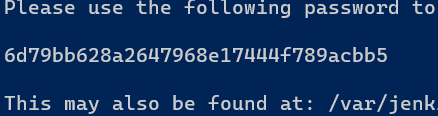


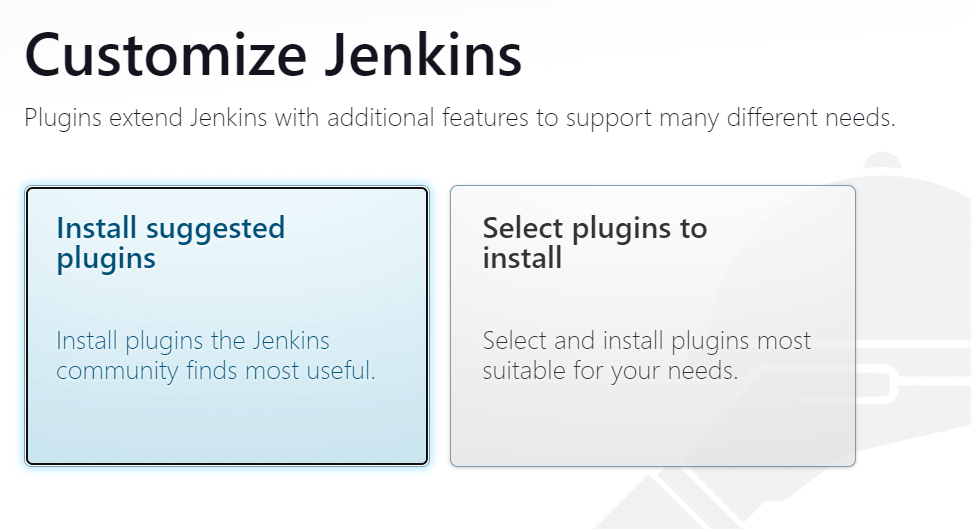




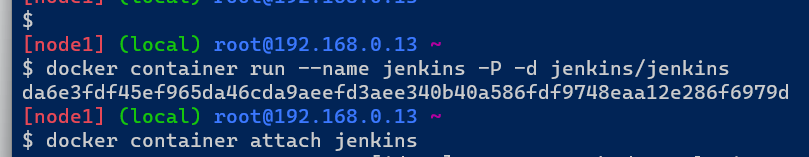
Enter the password from here

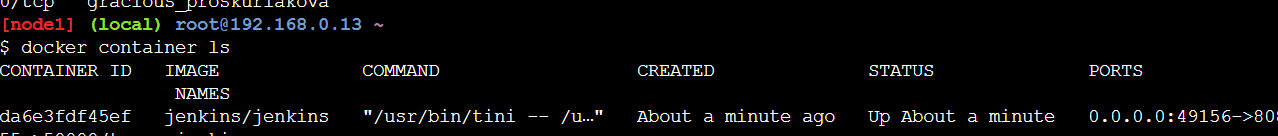


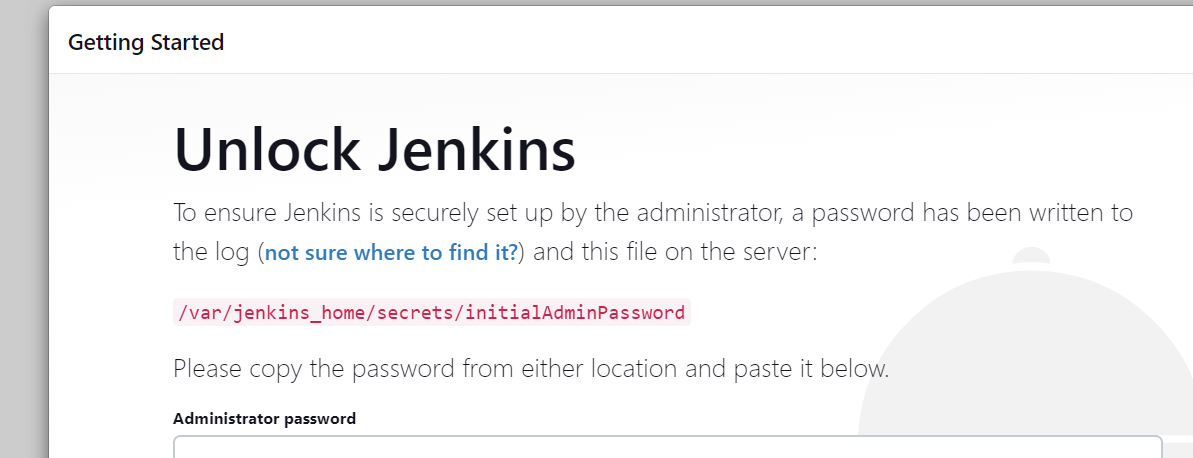




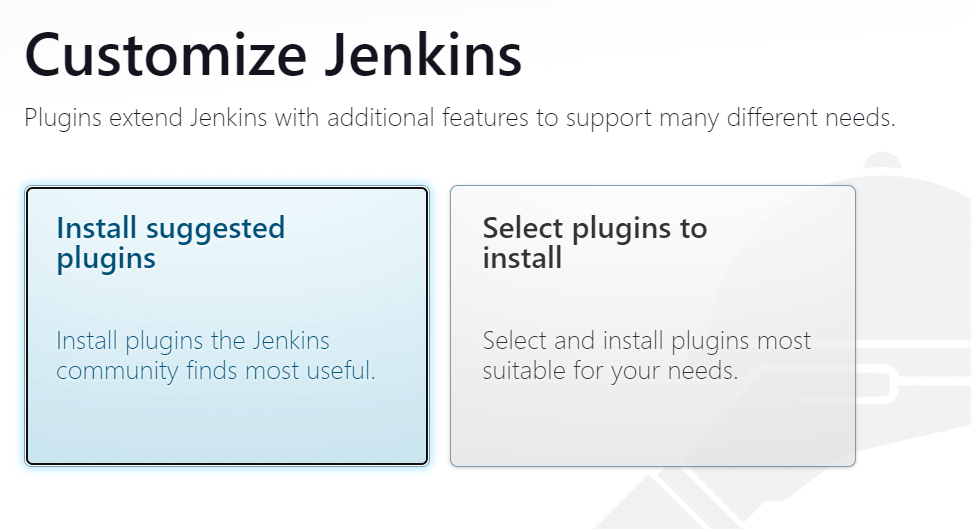
Now run in attached mode



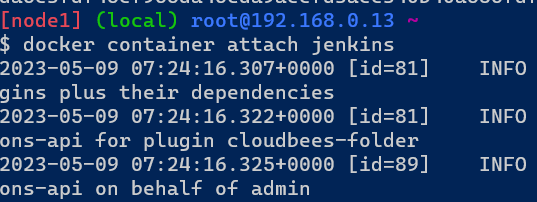


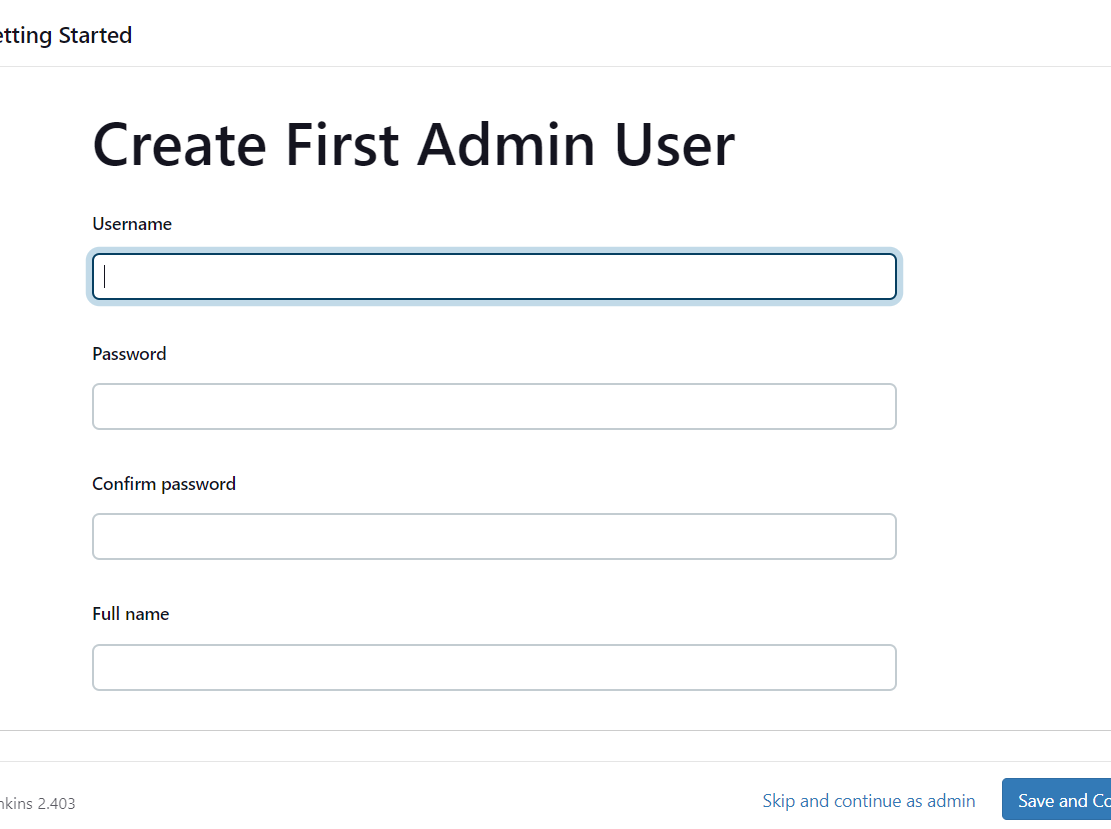




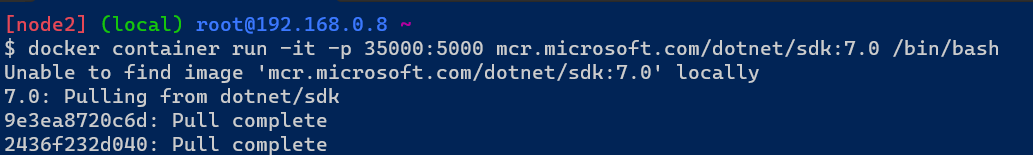


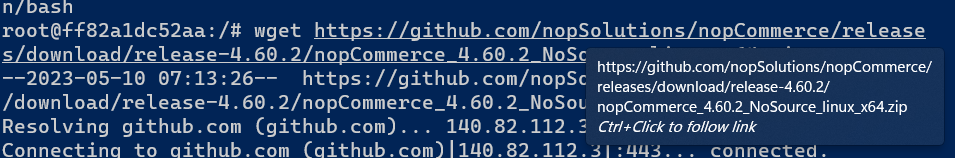
Click here install then it running the application

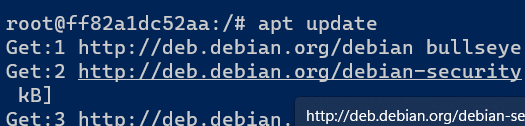


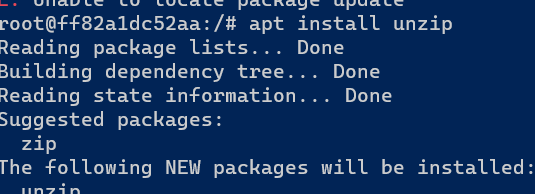


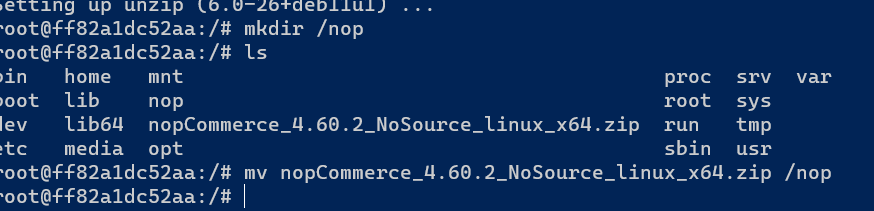
## dotnet application manual process

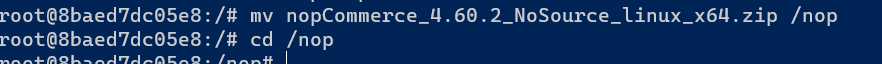


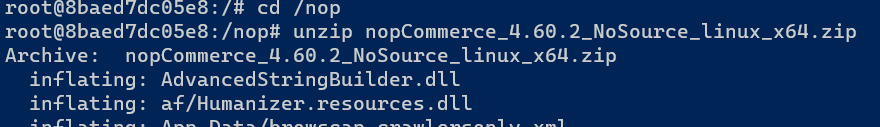




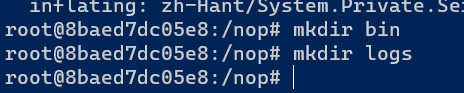


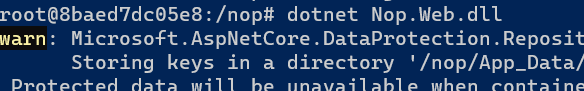




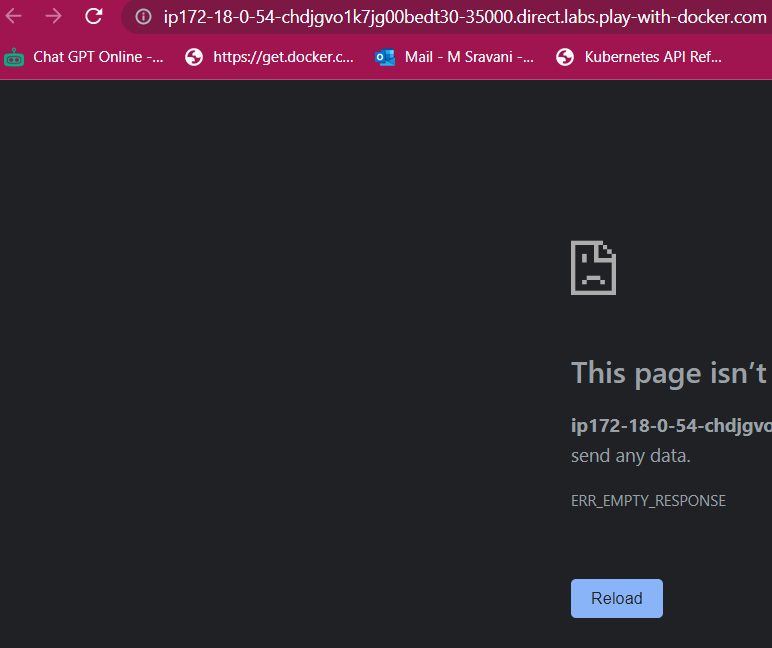


Create new folders two



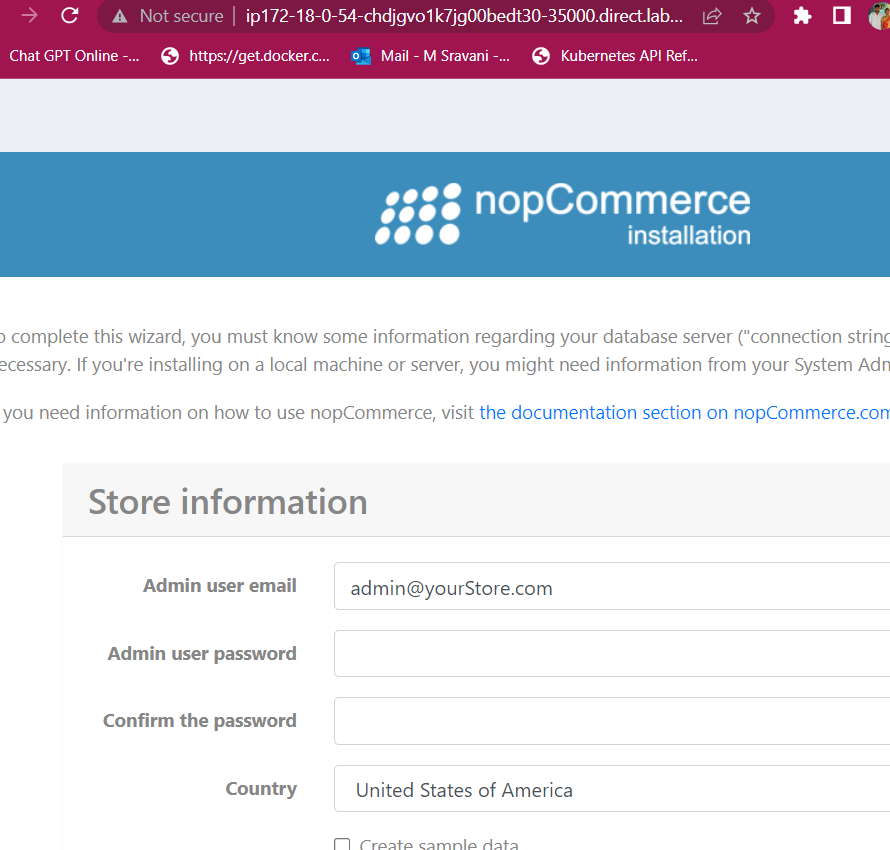


Now check the port 35000 it’s not working

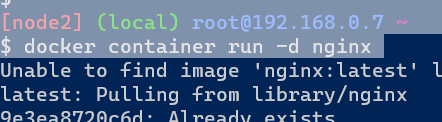


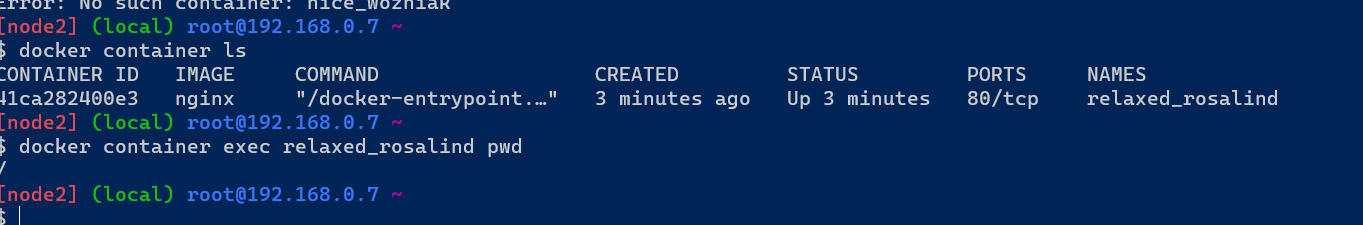


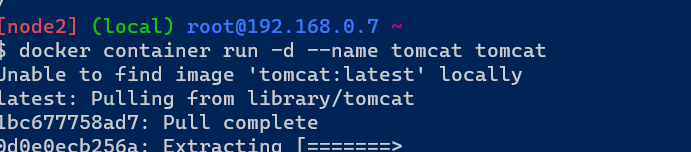


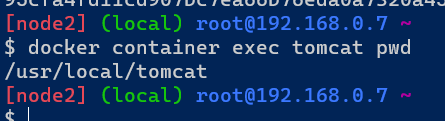


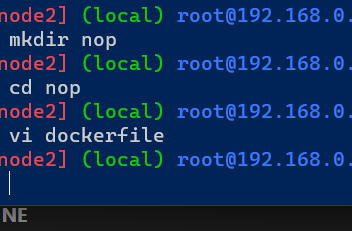
Run with docker file

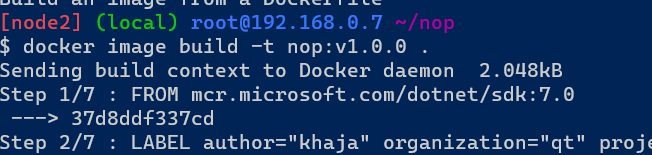


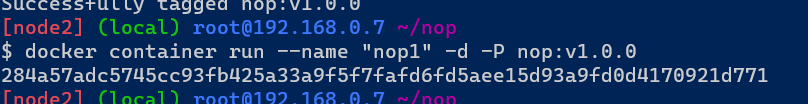




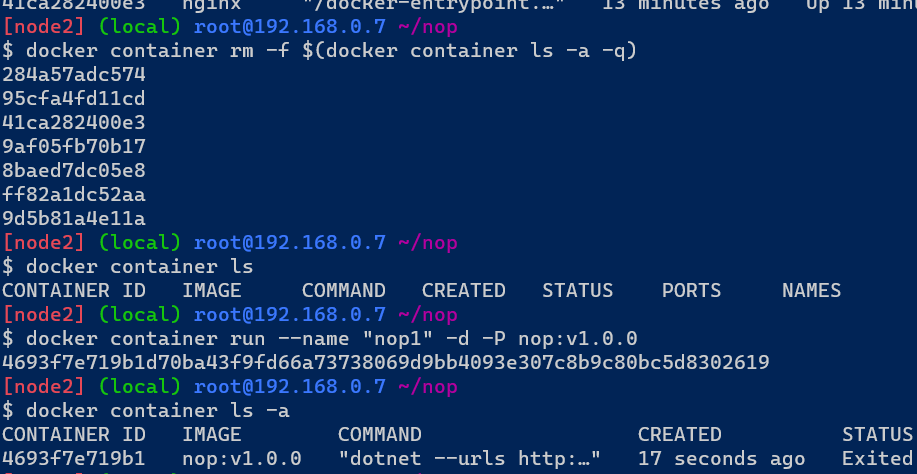


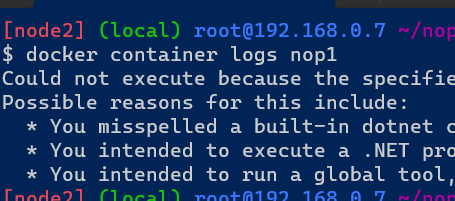


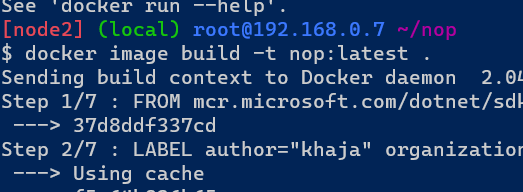


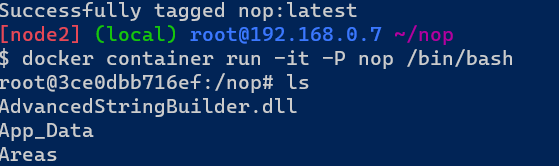


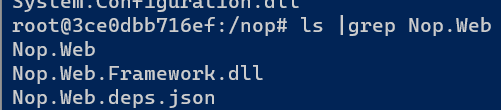




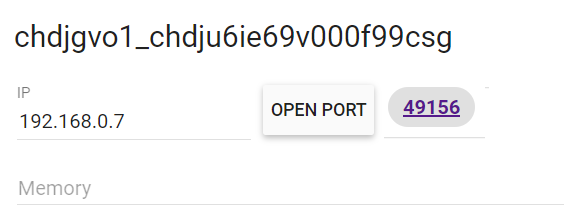


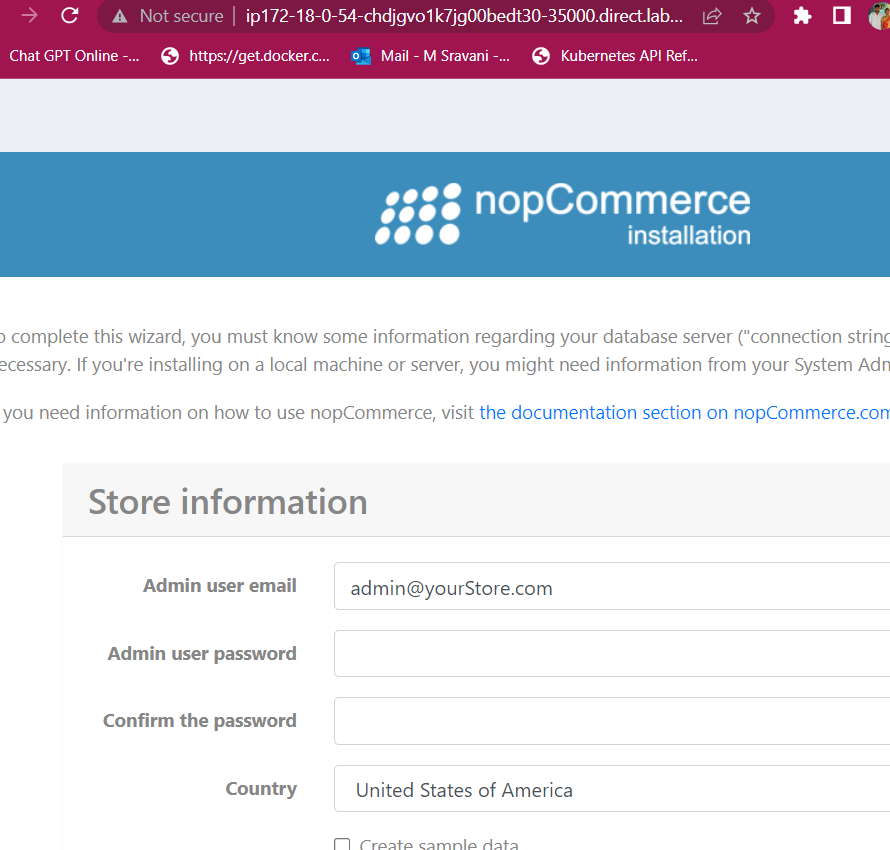


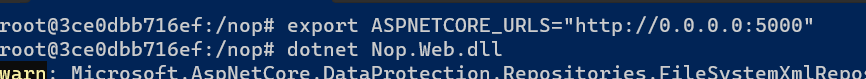


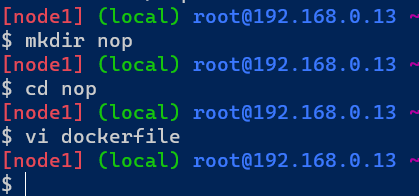


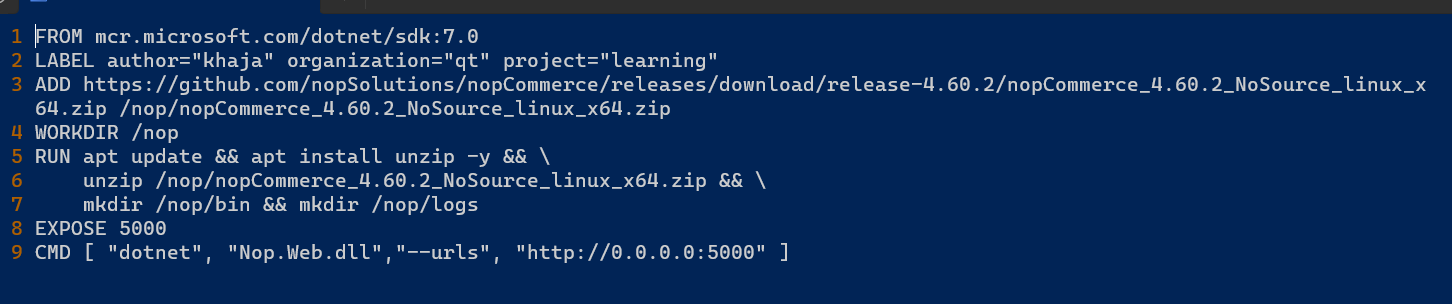


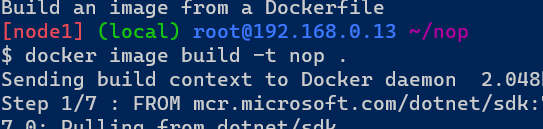


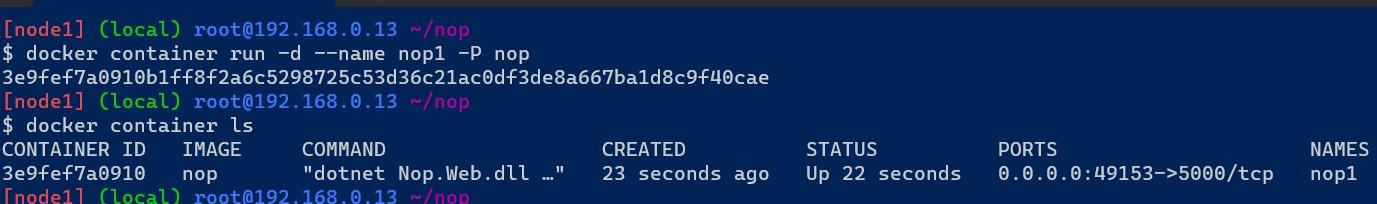




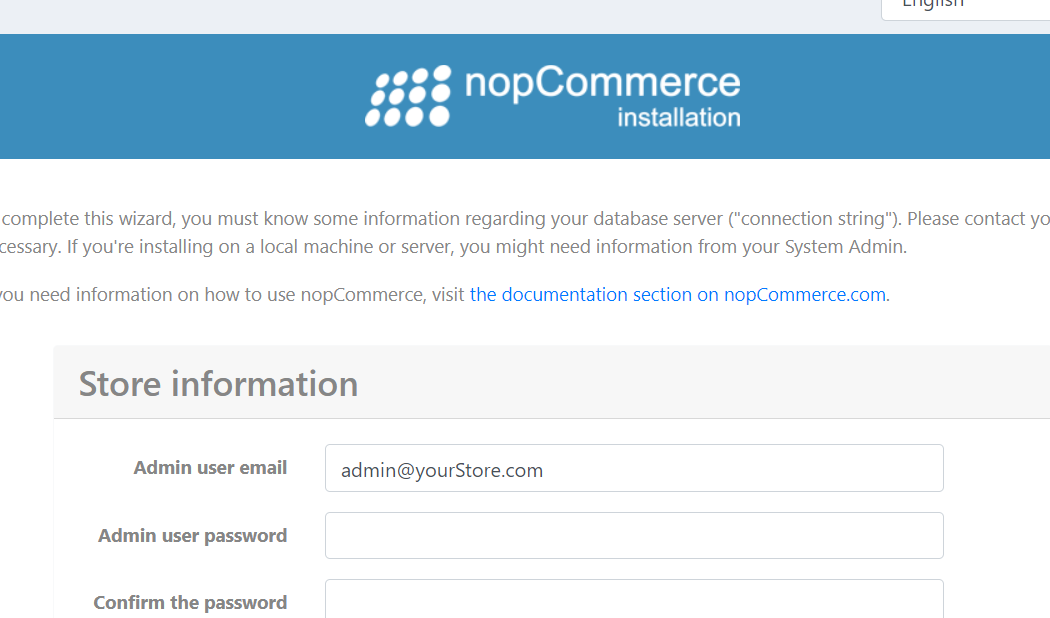




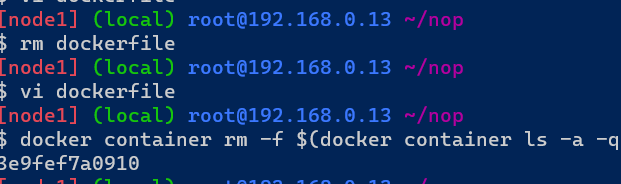


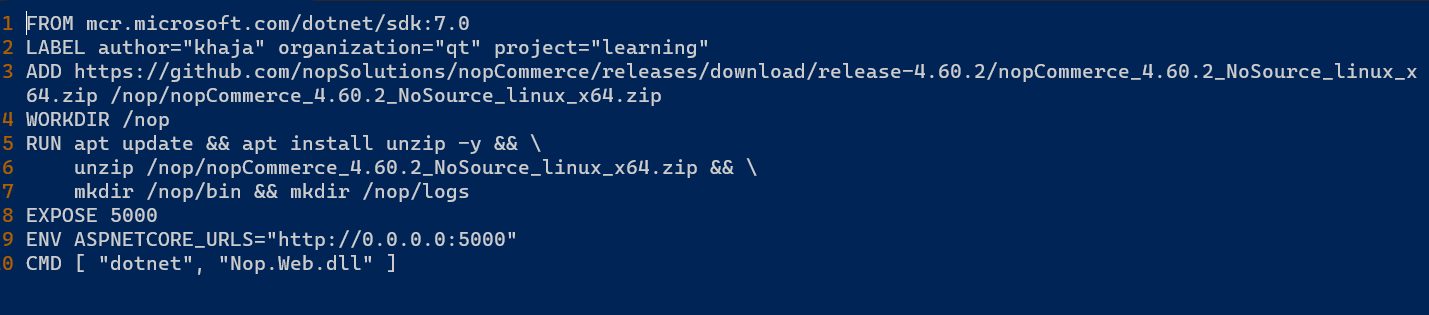






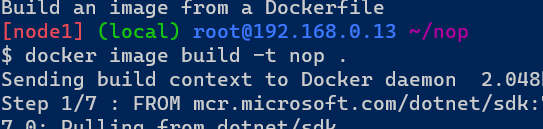
ENV instruction

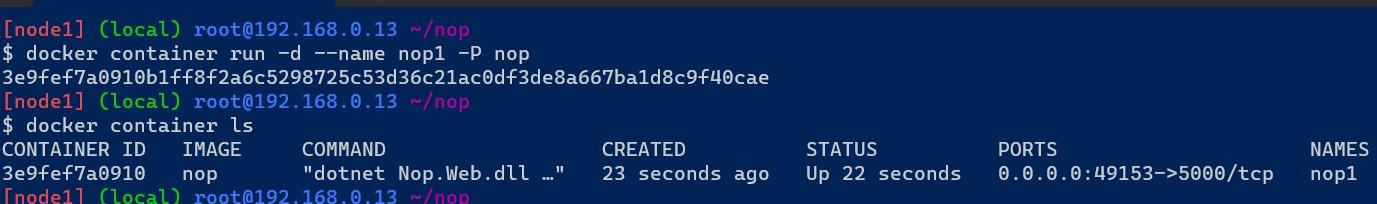




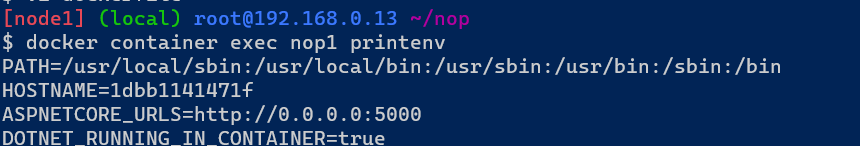
Remove previous image and build new image. To remove previous image

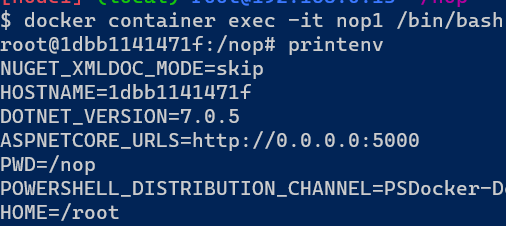
* docker image rm nop

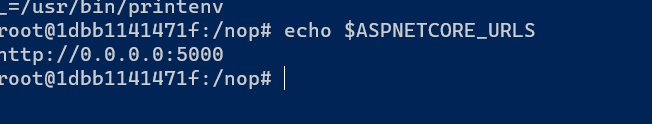




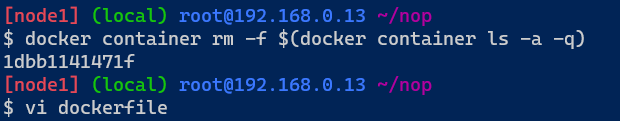
Now check the port its working or not.

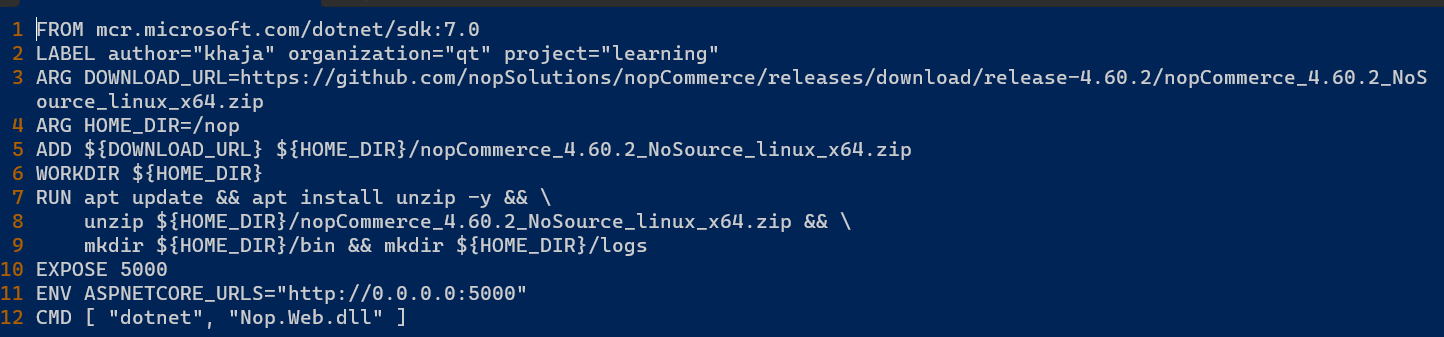


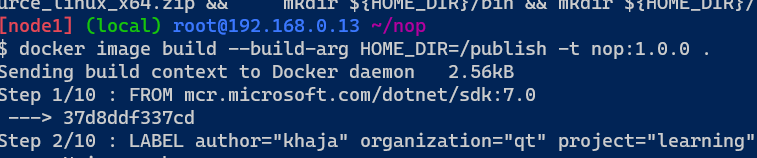


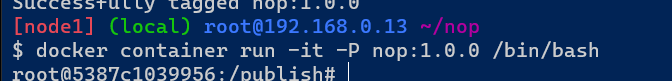


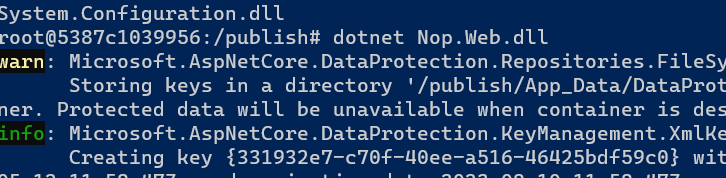
ARG instructions

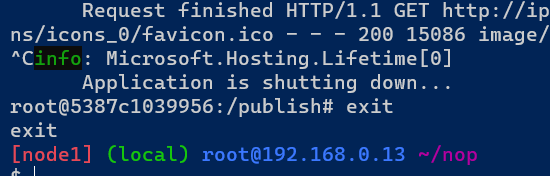


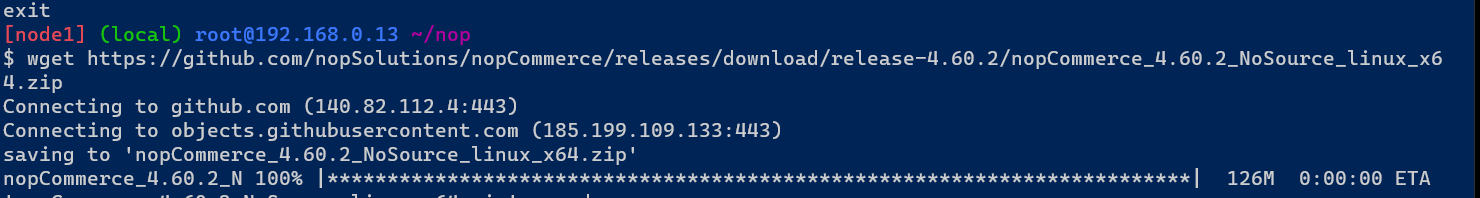


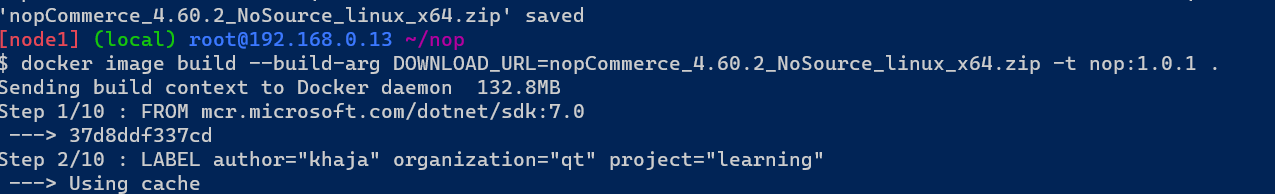




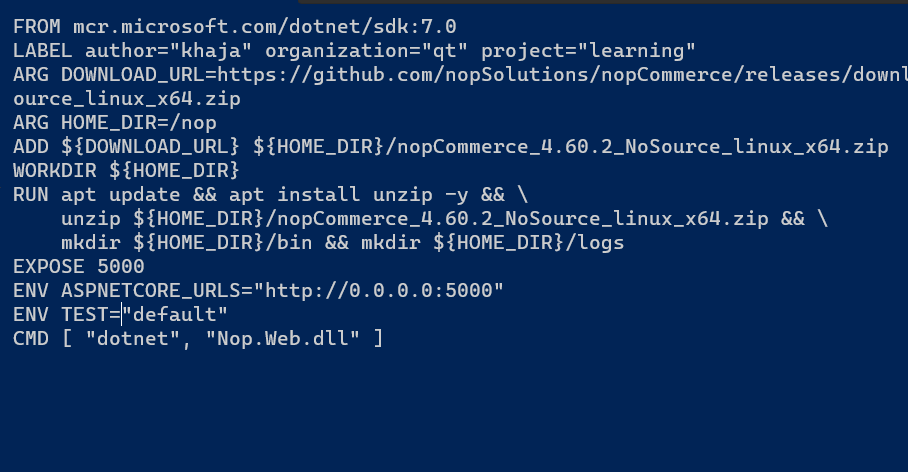


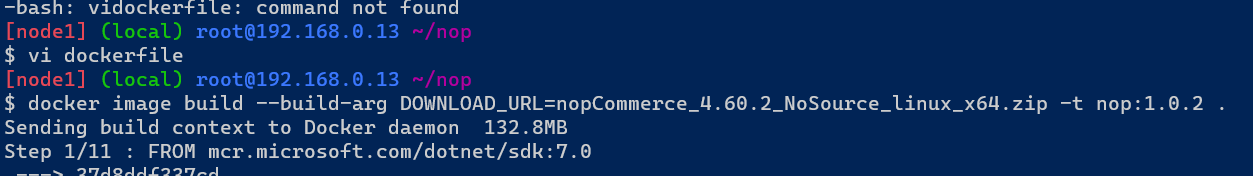




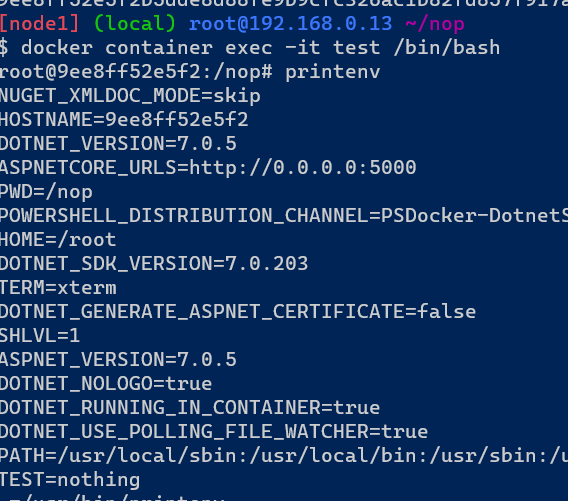


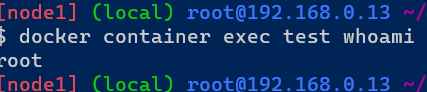
Add another ENV instruction

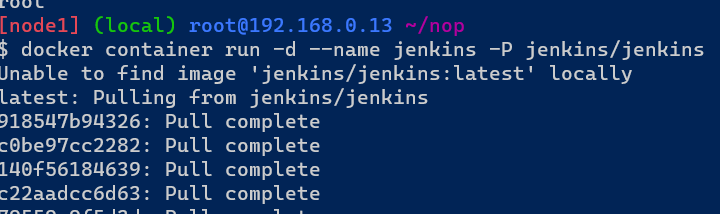


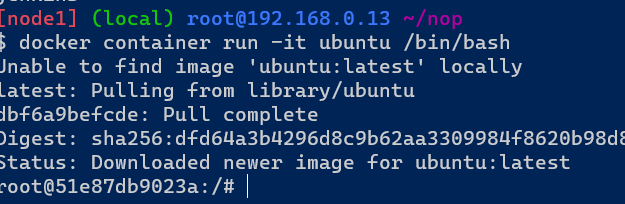








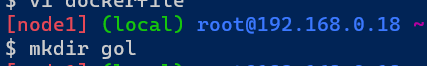


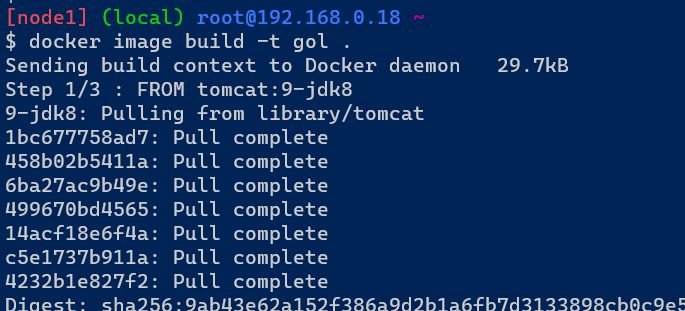


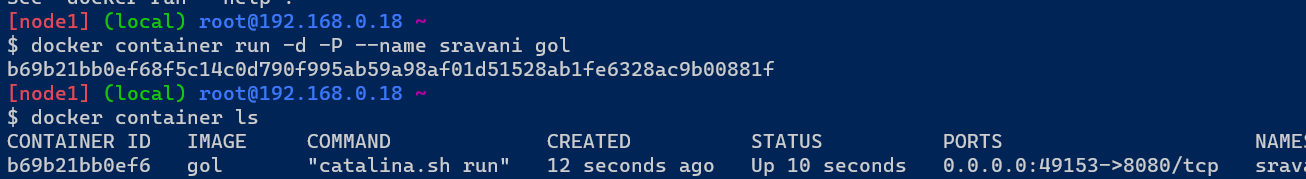
GAME OF LIFE Application

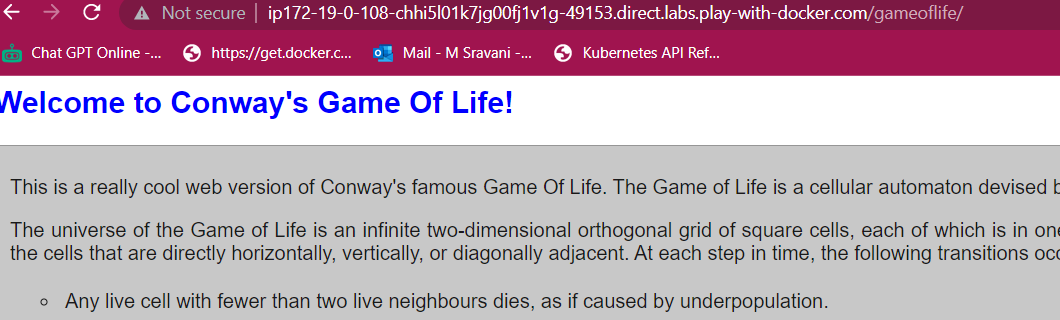
Docker file







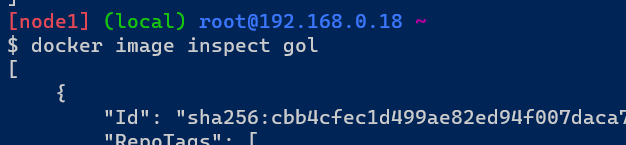




## **Image Layers**

* A Read write layer gets added to every container and image will have read layers
* Docker image is a collection of layers.

To see the layers in image

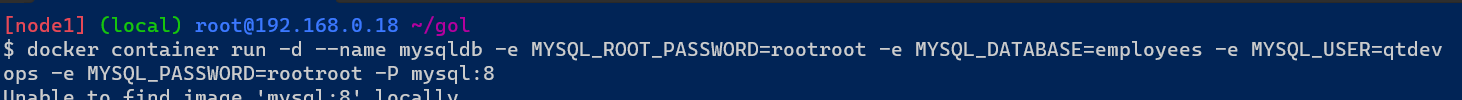




### **Stateful Applications and Stateless Applications**

* Stateful applications use local storage to store any state
* Stateless applications use external systems (database, blob storage 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.

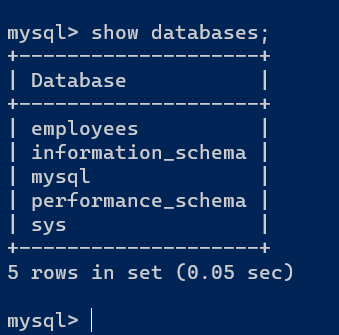
Creating a MYSQL container



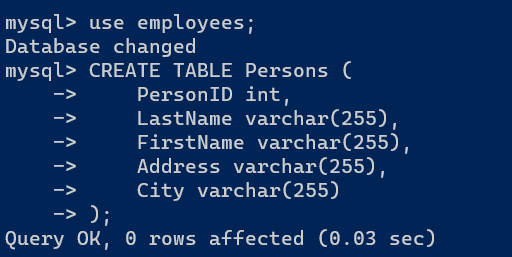
This is a stateful application it gives the information properly

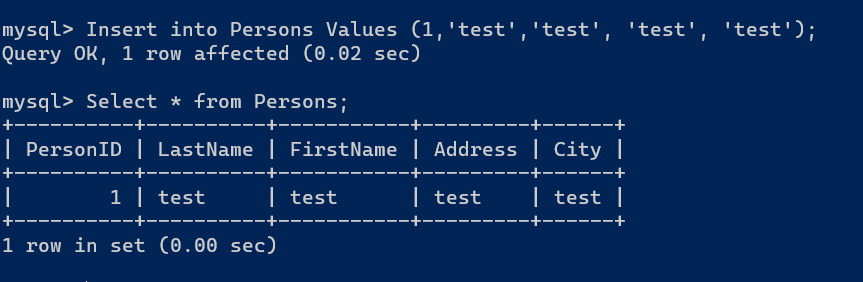
To enter into MySQL container to expose the application





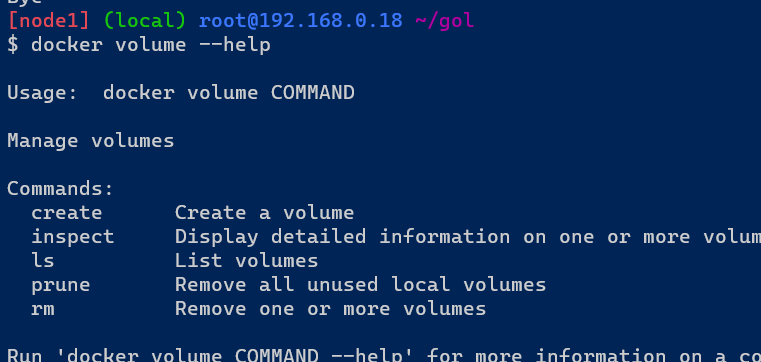
To create a table in MySQL





Now we observe the table. If the container will be deleted then we also loose data in the container.

To solve this problem we can fix the volumes to store the data if the container deleted also.



****Volumes****:

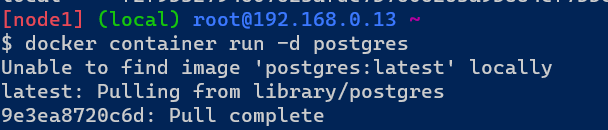
Stored in the host-file system managed by Docker (/var/lib/docker/volumes/ on Linux). Non Docker Processes should not modify the file system.

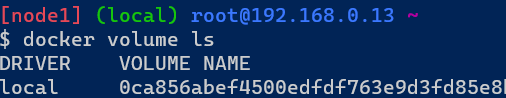
If writable layers are not available then volumes are also not there.

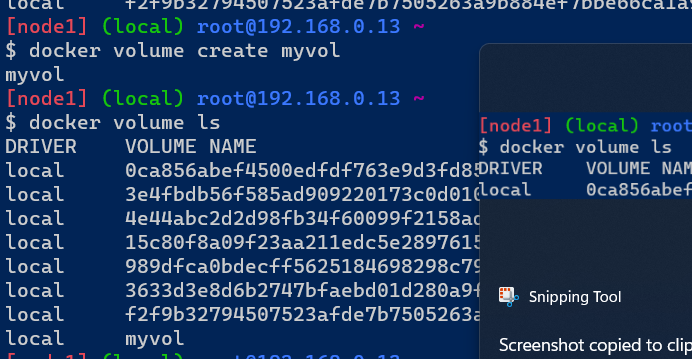
The volumes are three types

1. Bind mounts
2. Volume mounts
3. Tmpfs mounts

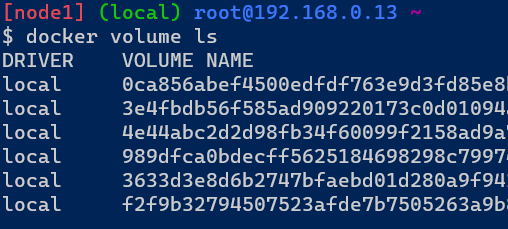


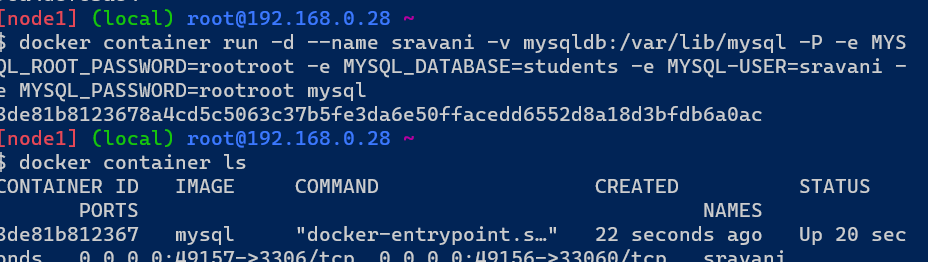


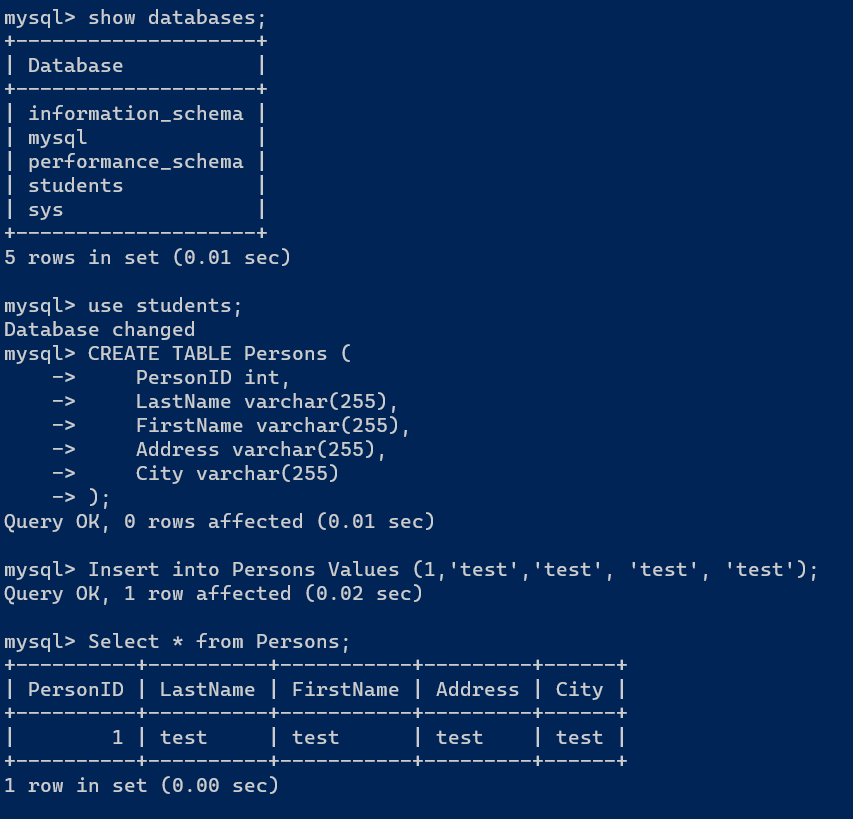




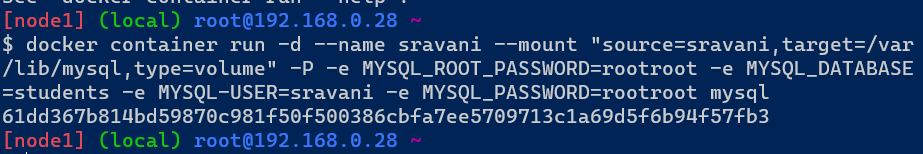








Now delete a container and create new container in another mode



CREATE TABLE students ( PersonID int, LastName varchar(255), FirstName varchar(255), Address varchar(255), City varchar(255) );

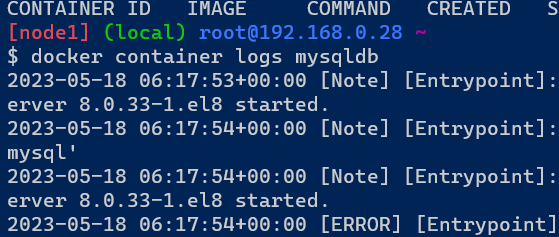
Insert into students Values (1,'test','test', 'test', 'test');

Select \* from students;

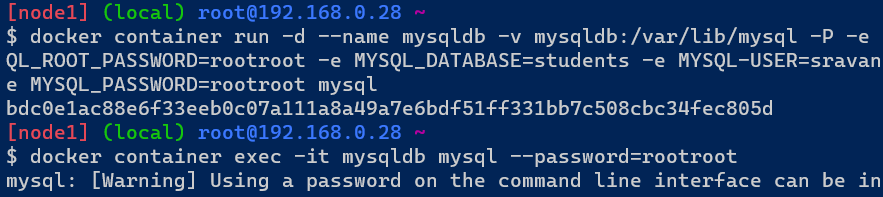
docker container run -d --name sravani --mount "source=sravani,target=/var/lib/mysql,type=volume" -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=students -e MYSQL-USER=sravani -e MYSQL\_PASSWORD=rootroot mysql

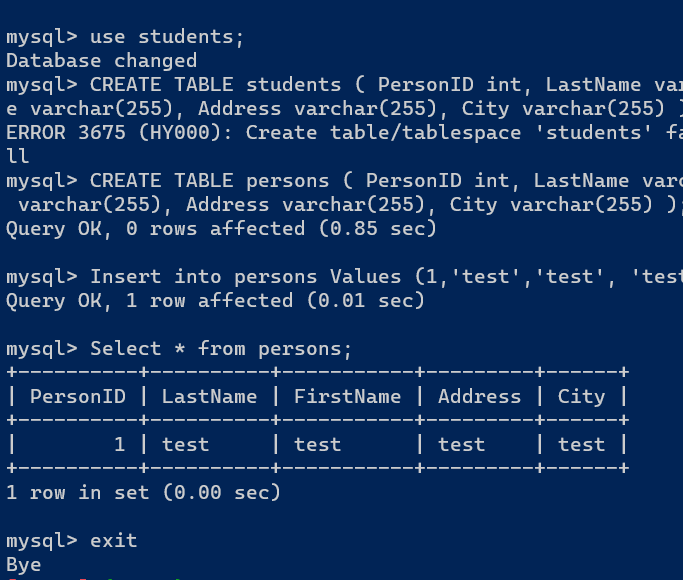
docker container exec -it sravani mysql --password=rootroot

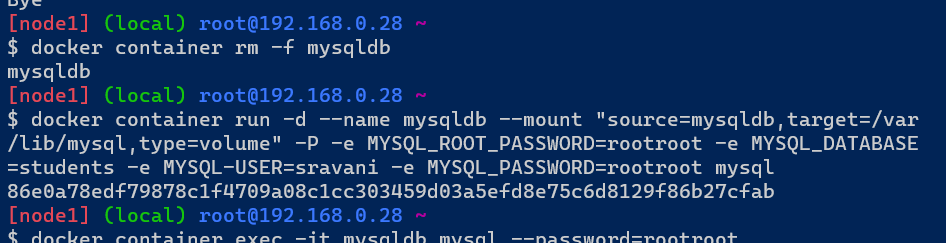
docker container run -d --name sravani -v mysqldb:/var/lib/mysql -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=students -e MYSQL-USER=sravani -e MYSQL\_PASSWORD=rootroot mysql

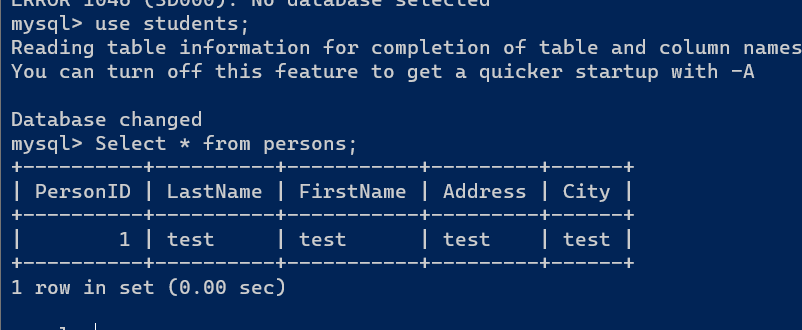


If the container exited when you create immediately then find out the problem by using logs command.

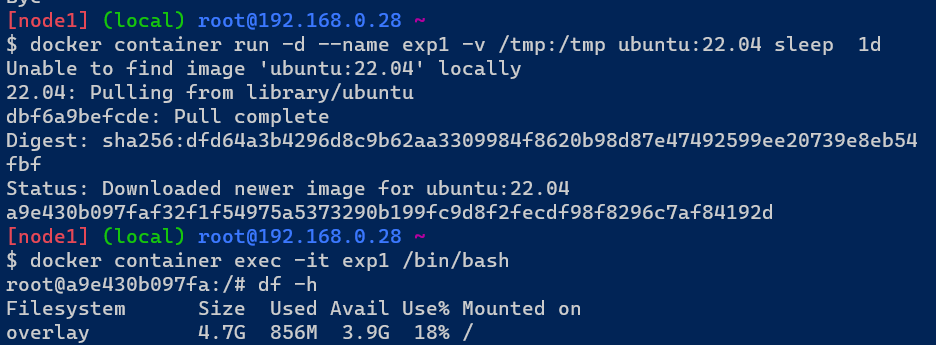




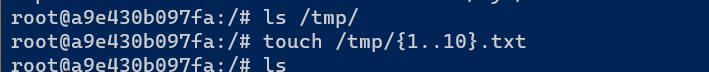




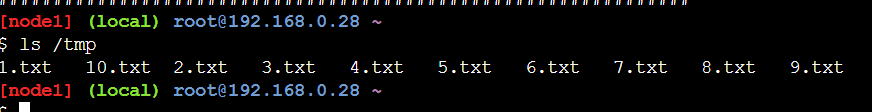
Now bind mode volumes



Now create files by using single command that is touch

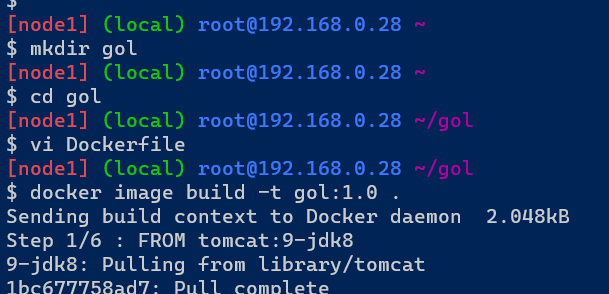


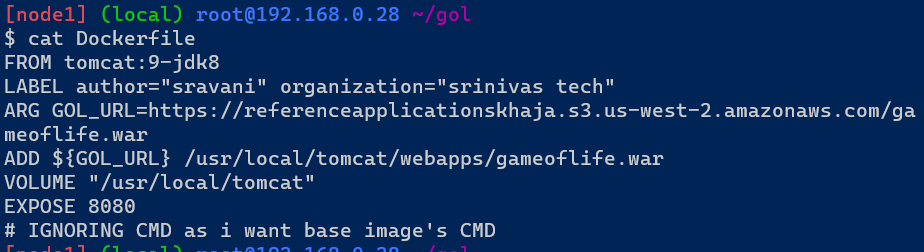
Now we check the files on docker play ground



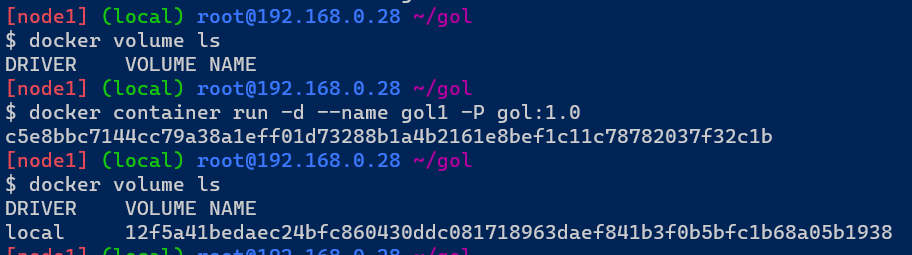
Bind mode is using only docker host and docker container having same folder then we use this mode.

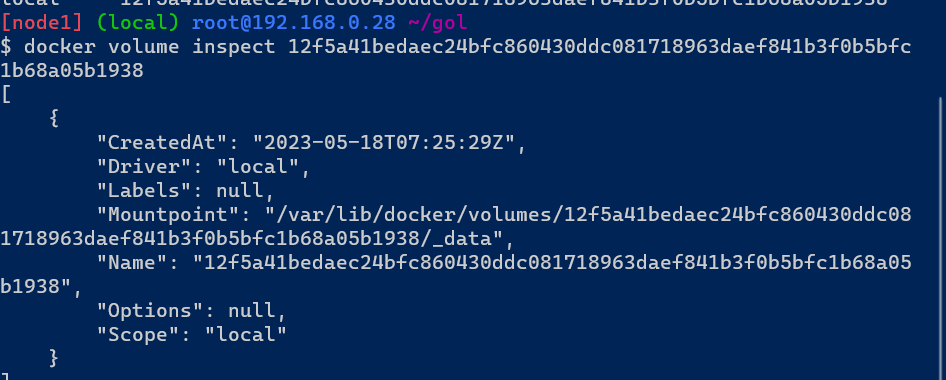
### write a docker file for game of life by using VOLUME instruction

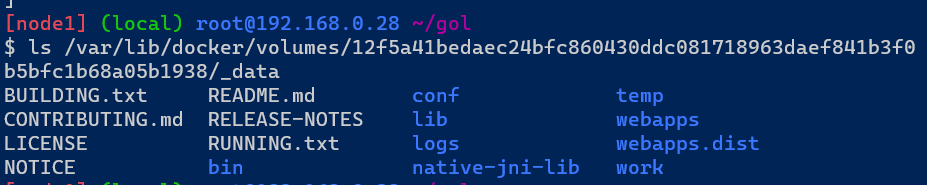




Now we see the volume after creating container







Now we observe the image of game of life

