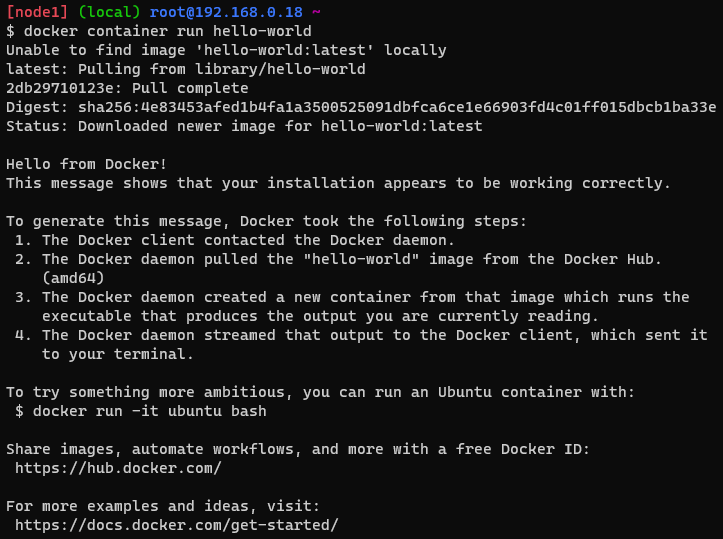
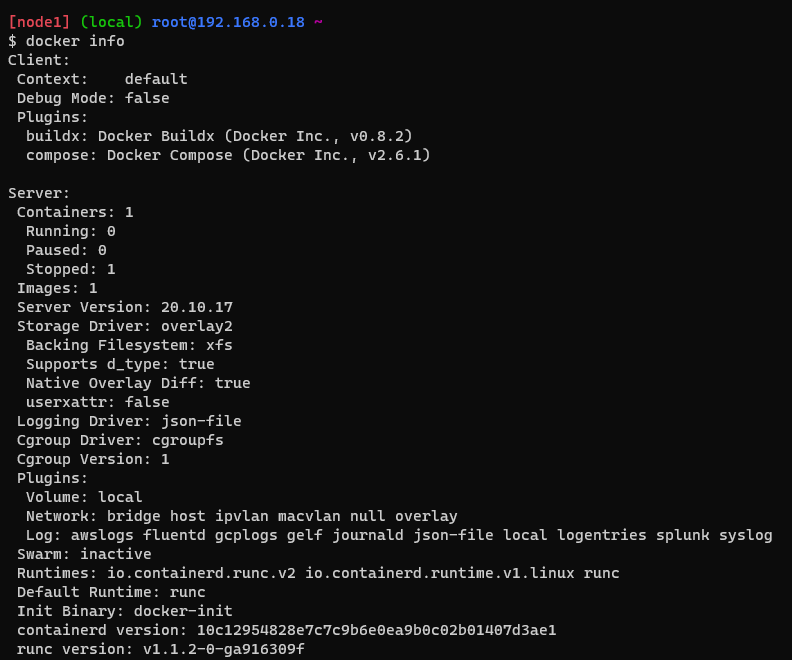
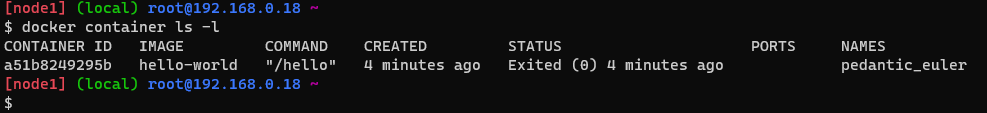
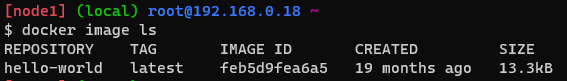
1. Run hello-world docker container and observe the container status



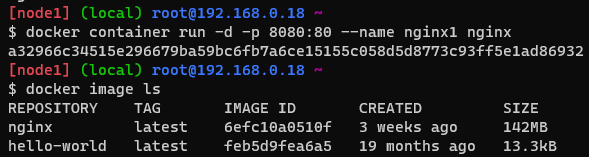


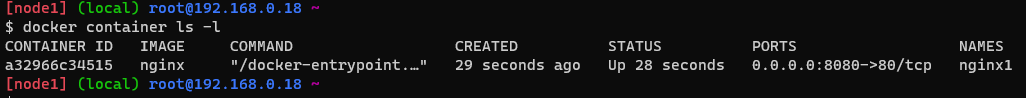


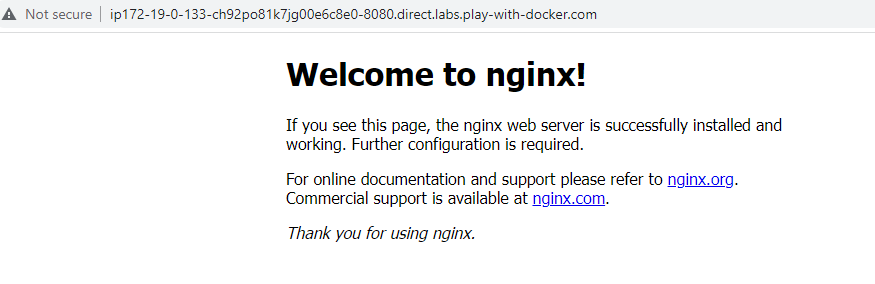
2. Check the docker images and also write down the size of hello-world image



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



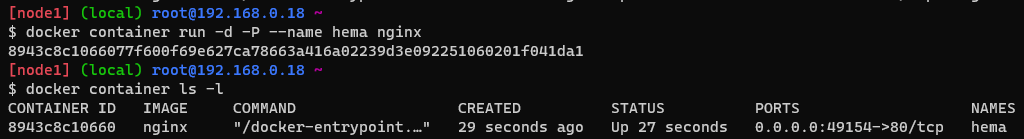




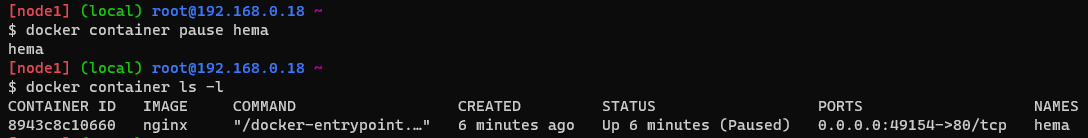
4. Explain docker container lifecycle

Docker container lifecycle has 5 stages.They are

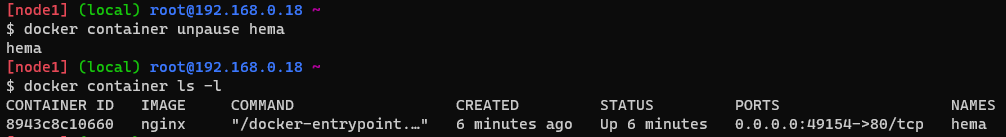
* Create
* Run
* Pause
* Stop
* Delete
* DOCKER CONTAINER CREATE



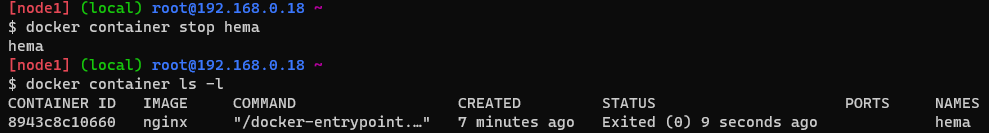
* DOCKER CONTAINER PAUSE



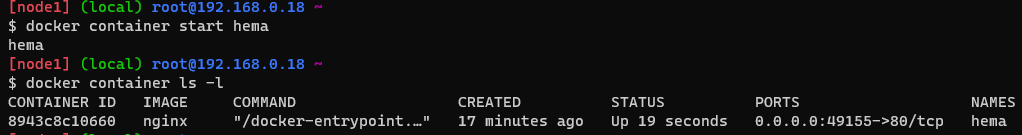
* DOCKER CONTAINER UNPAUSE



* DOCKER CONTAINER STOP



* DOCKER CONTAINER START



* DOCKER CONTAINER DELETE



5. Explain what happens when you run the docker container

The docker run command creates and starts the docker container. Docker daemon checks if the image is available locally or not. If the image exists locally it creates the container. If the image doesn’t exists it pulls the image from the local repository. Once the image is pulled the container gets created.

6. Explain the Docker Architecture

* **First generation**:

Docker client (command line) will send request to docker daemon, it forwards the request to lxc (linux container) internally uses namespaces, c groups to create containers.

* **Second generation:**

Due to frequent updates to linux kernel,there is a chance of container breakdown .As docker is relying on lxc to create containers it has created its own component called as lib container (libc)

* **Third generation:**

In this generation docker is revamped from monolith to micro services i.e., multi component system. Here docker daemon exposes its api to interact with docker client then it sends the request to container-d and then passes to run-c which creates the container and manages the docker lifecycle.

What is Docker?

Docker is a configuration management tool that is used to automate the deployment of software in lightweight containers. These containers help applications to work efficiently in different environments.

Features of Docker:

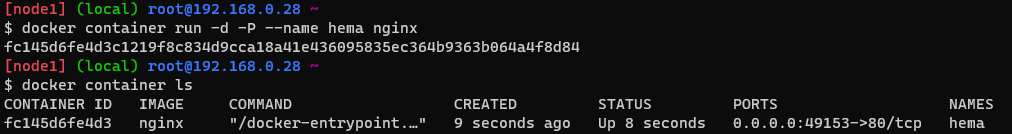
* Easy and faster configuration
* Application isolation
* Security management
* High productivity
* High scalability
* Infrastructure independent

**To check whether any application is running on a specific port**

<netstat -tupln>

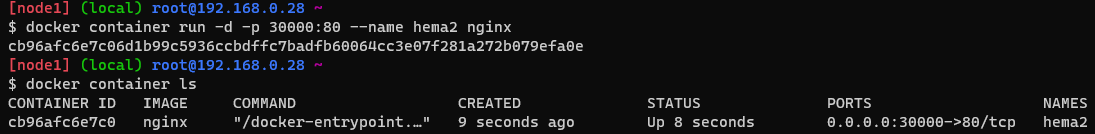
**To assign random port number**

Docker container run –d –P --name hema nginx



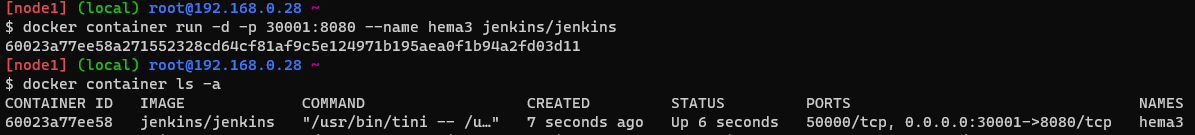
**Creating nginx container and exposing on 30000 port**

Docker container run –d –p 30000:80 – -name hema nginx

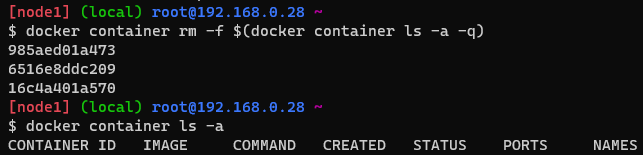
****

**Creating Jenkins container and exposing on 30001 port**

Docker container run –d –p 30001:8080 --name hema3 Jenkins/Jenkins



**To remove all the running containers at once**

****

**To change the image tag from previous version to latest version**

Docker image tag myspc:corretto11 myspc:latest

**Containerizing spring petclinic**

To run this application java is required and runs on 80 port

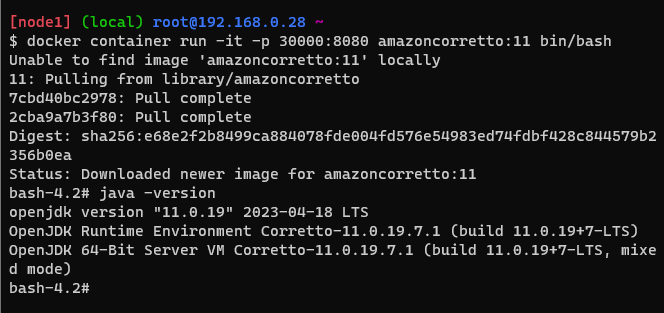
* Jdk 11
* Jar file

1.find out the image which has jdk 11 in the docker hub

2.amazoncorretto is having open jdk 11

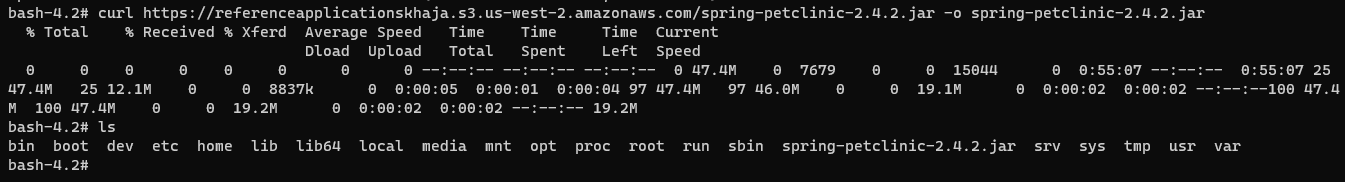
3.starting the amazoncorretto based container with port 8080

4.docker container run –it –p 30000:8080 amazoncorretto:11 /bin/bash(this image has java)



5.now download springpetclinic

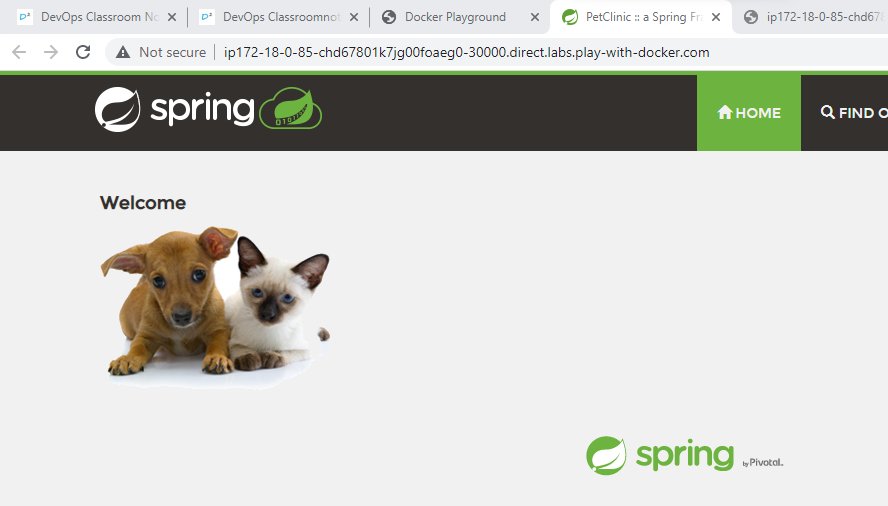
curl https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar -o spring-petclinic-2.4.2.jar



6.to run the application

To start the application (java –jar spring-petclinic-2.4.2.jar)





**To create image from running container**

Docker container commit (application name) imagename:tag

Containerizing springpetclinic by using Dockerfile

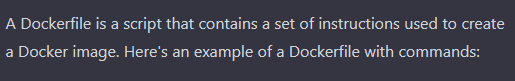
FROM amazoncorretto:11

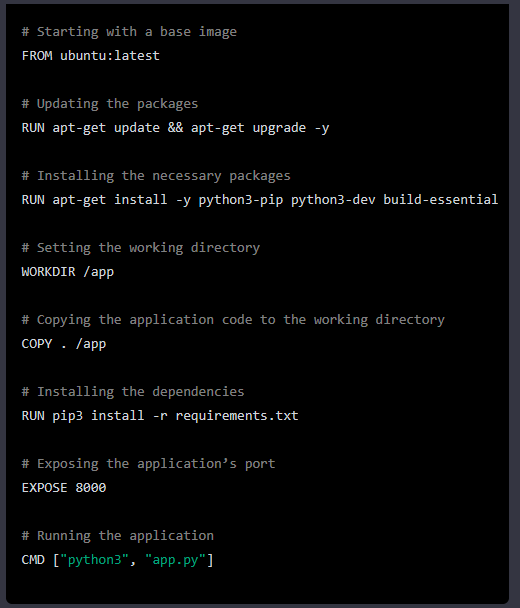
RUN curl https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/spring-petclinic-2.4.2.jar -o spring-petclinic-2.4.2.jar

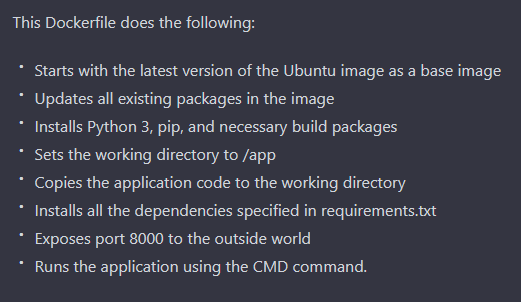
EXPOSE 8080

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

DOCKER FILE**:** Docker file is a simple text file that consists of instructions to build Docker images.







**FROM**: base image

**RUN**: for installation and configuring the application

**CMD**: while starting the container

**EXPOSE**: to expose port number

**LABEL**: adds metadata

**ADD**: adds files into docker image from local file system as well as http(download)

**COPY**: supports only in local file system

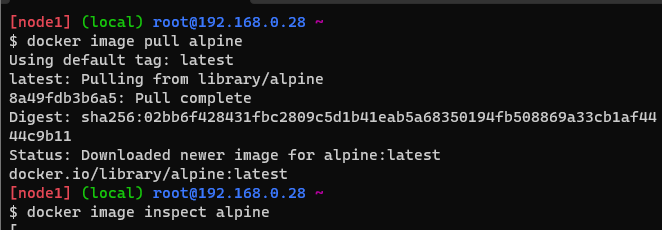
* Copy from local file on docker host into docker image

**DOCKER IMAGE LAYERS**

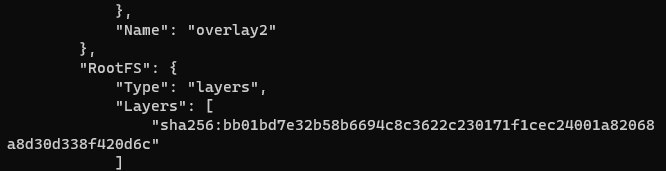
* Docker image is a collection of layers.

**Docker images are read only, whenever we create container ,a thin layer(read write) gets created.**

Creating alpine image and inspecting the image



Alpine image layer

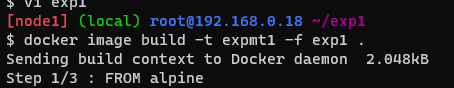


**Exp1**

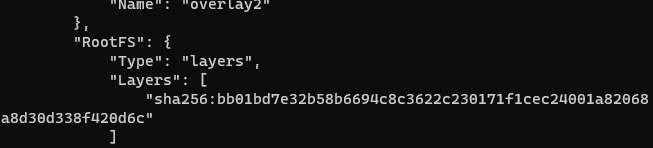
FROM alpine

label author=hema

CMD ["sleep","1d"]







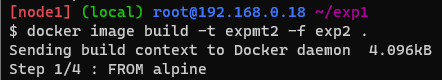
**Exp2**

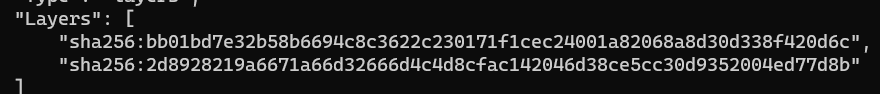
FROM alpine

label author=hema

ADD 1.txt /

CMD ["sleep","1d"]

****

****

**Exp3**

FROM alpine

label author=hema

RUN echo "one" > 1.txt

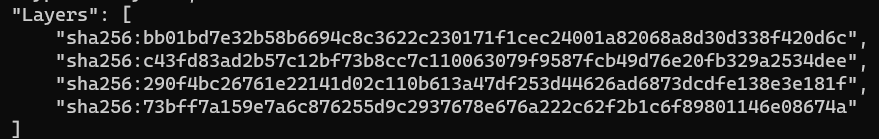
RUN echo "two" > 2.txt

RUN echo "three" > 3.txt

CMD ["sleep","1d"]

****

* Here run command is used three times to create txt files.so that it created four layers in total.

****

**Exp4**

FROM alpine

Label author=hema

RUN echo “one” > 1.txt && \

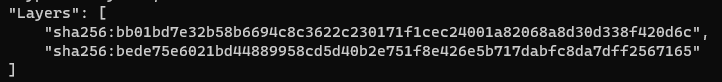
echo “two” > 2.txt && \

echo “three” > 3.txt

CMD [“sleep”,”1d”]



* Here run command is used only once and added 3 txt files, It has created only two layers in total. Instead of creating two many layers single RUN command is used.
* Each run instruction needs some storage,it creates a layer

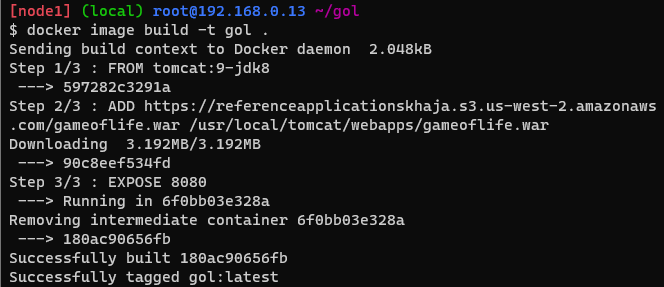


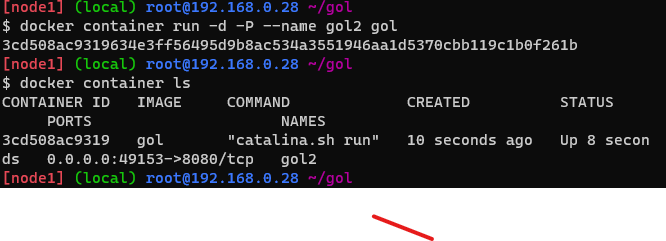
GAMEOFLIFE

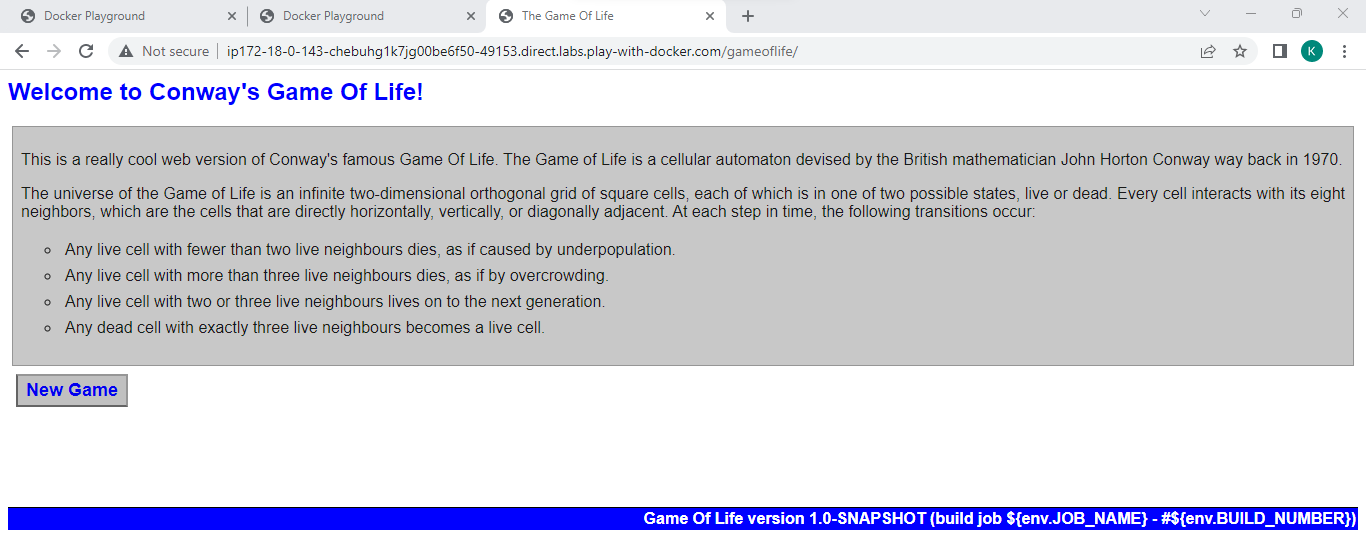
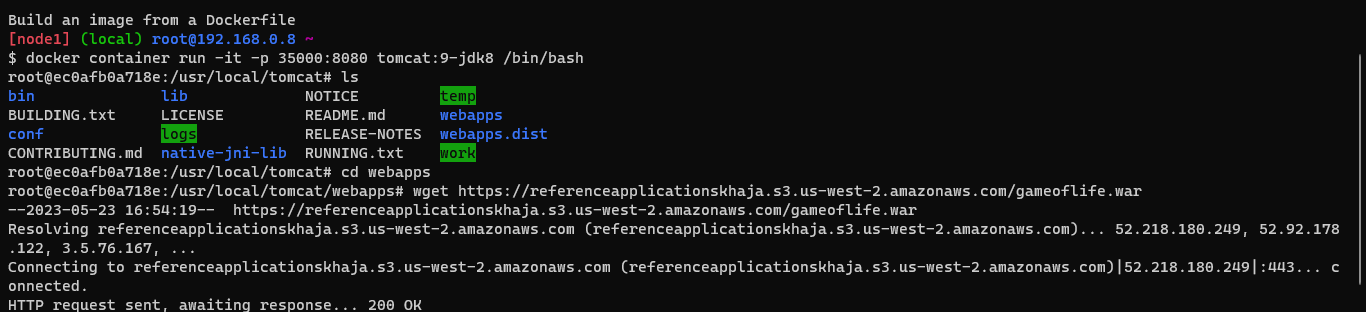
FROM tomcat:9-jdk8

ADD https://referenceapplicationskhaja.s3.us-west-2.amazonaws.com/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

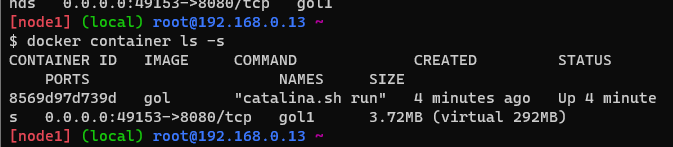
EXPOSE 8080







To check the size of the image

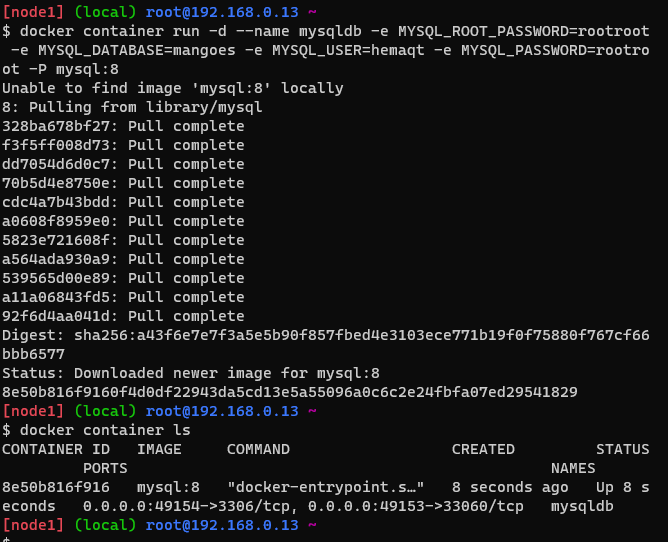


**Stateful and stateless applications**

Stateful applications **:** use local storage to store any state

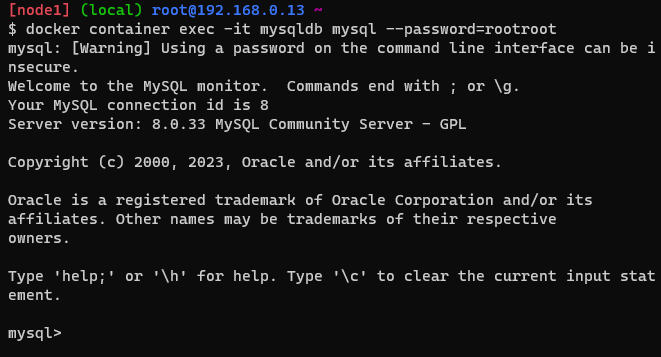
Stateless applications**:** use external systems to store the state(database)

**MYSQL CONTAINER:**

****

**To login into mysql container execute,**

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

****

use mangoes;

CREATE TABLE Persons (

PersonID int,

LastName varchar(255),

FirstName varchar(255),

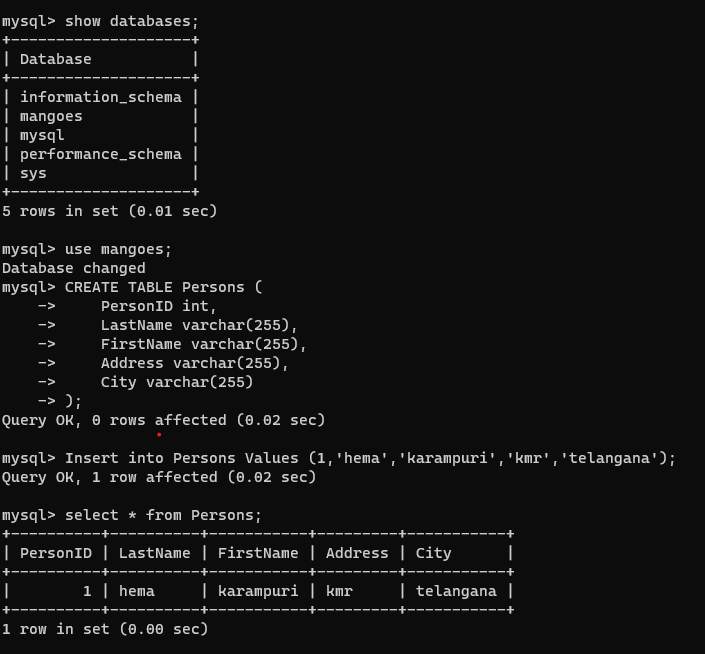
Address varchar(255),

City varchar(255)

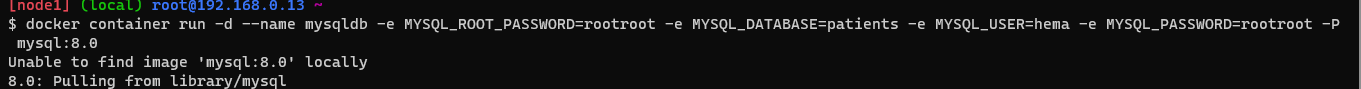
);

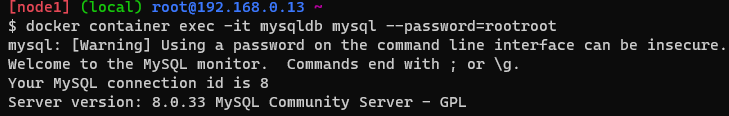
Insert into Persons Values (1,'hema','karampuri', 'kmr', 'telangana');

Select \* from Persons;

****

**EXAMPLE 2**

****

****

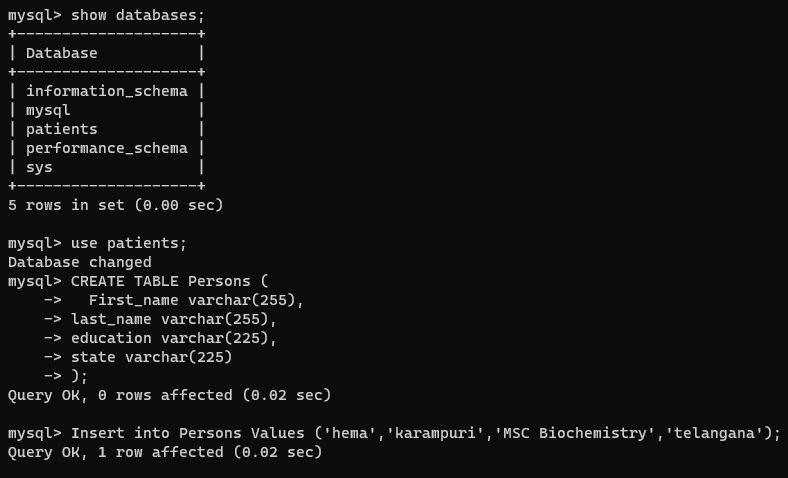
CREATE TABLE Persons (

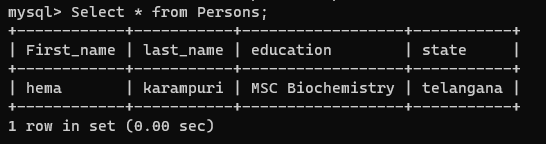
First\_name varchar(255),

last\_name varchar(255),

education varchar(225),

state varchar(225)   
);

****

****

**Docker volumes**

The purpose of using Docker volumes is to persist data outside the container so it can be backed up or shared.

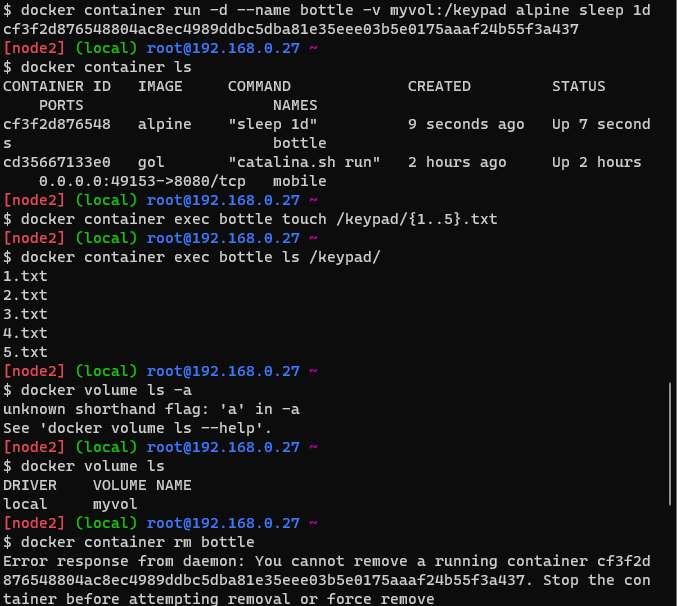
* If we remove the container we loose the data
* To fix the problem with data losses, Docker has volumes.
* Volume can exist even after docker container is deleted.

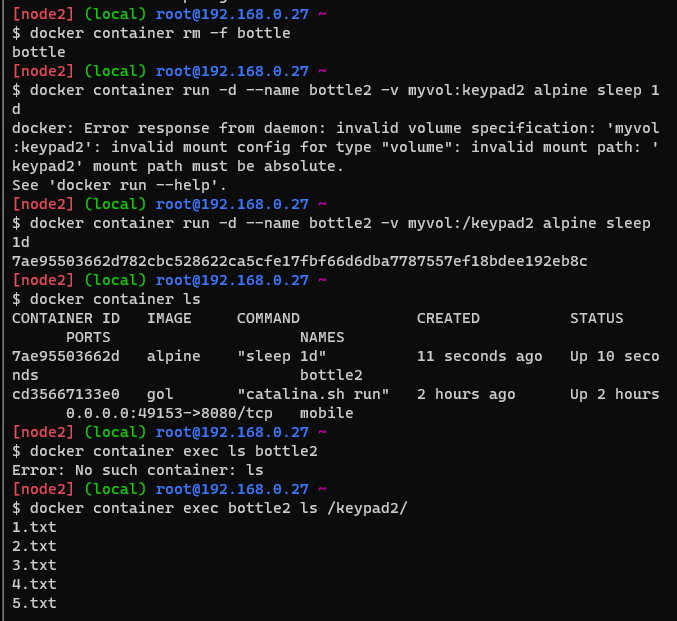
docker volume create myvol

docker container run -d --name bottle -v myvol:/keypad alpine sleep 1d

docker container exec bottle touch /keypad/{1..5}.txt

docker container exec bottle ls /keypad/

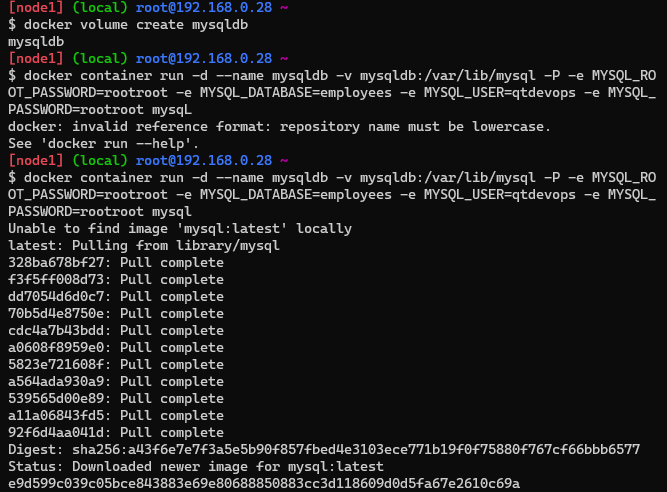




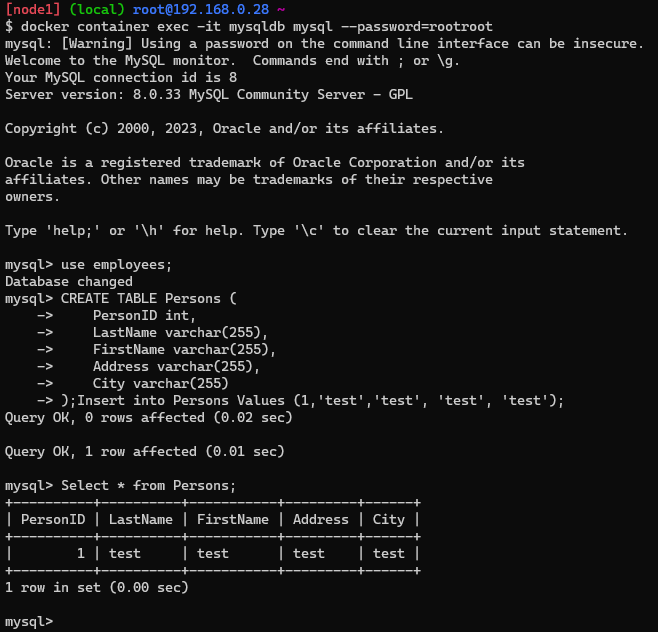
**Creating mysql container and attaching volume**

docker volume create mysqldb

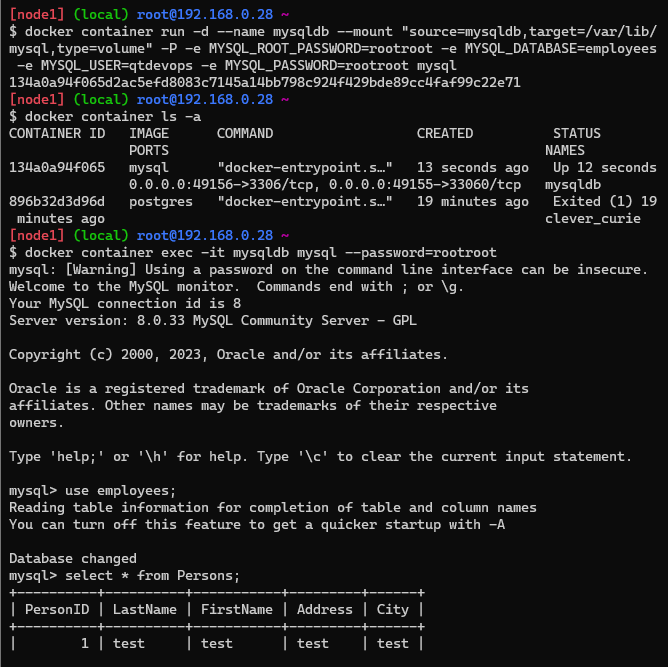
docker container run -d --name mysqldb -v mysqldb:/var/lib/mysql -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=employees -e MYSQL\_USER=qtdevops -e MYSQL\_PASSWORD=rootroot mysql



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



Now delete this container and create a new container using mount then execute the commands,without giving the data of inserting table.Eventhough the container is deleted the data is still persistent as the data is stored in the volume.



To remove all the volumes which are not removed by container

docker volume prune

**creating volume as part of docker file for game of life application**

1 FROM tomcat:9-jdk8

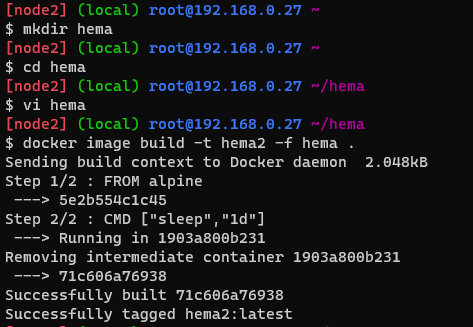
2 LABEL author="hema" organization="hs"

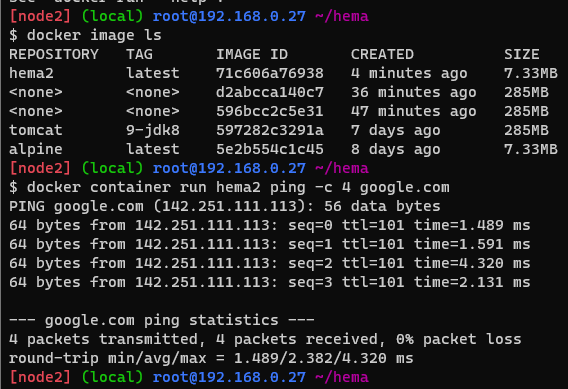
3 ARG GOL\_URL=https://referenceapplicationskhaja.s3.us-west-2.amazonaw s.com/gameoflife.war

4 ADD ${GOL\_URL} usr/local/tomcat/webapps/gameoflife.war

5 VOLUME "/usr/local/tomcat"

6 EXPOSE 8080





FROM alpine/git AS vcs

RUN cd / && git clone https://github.com/spring-projects/spring-petclinic.git

RUN pwd && ls /spring-petclinic

FROM maven:3-amazoncorretto-17 AS builder

LABEL author="hema" organization="hs"

COPY --from vcs /spring-petclinic /spring-petclinic

RUN ls /spring-petclinic

RUN cd /spring-petclinic && mvn package

FROM amazoncorrtto:17-al2-jdk

WORKDIR /spc

COPY --from builder/spring-petclinic/target/spring\*.jar /spc/spring-petclinic.jar

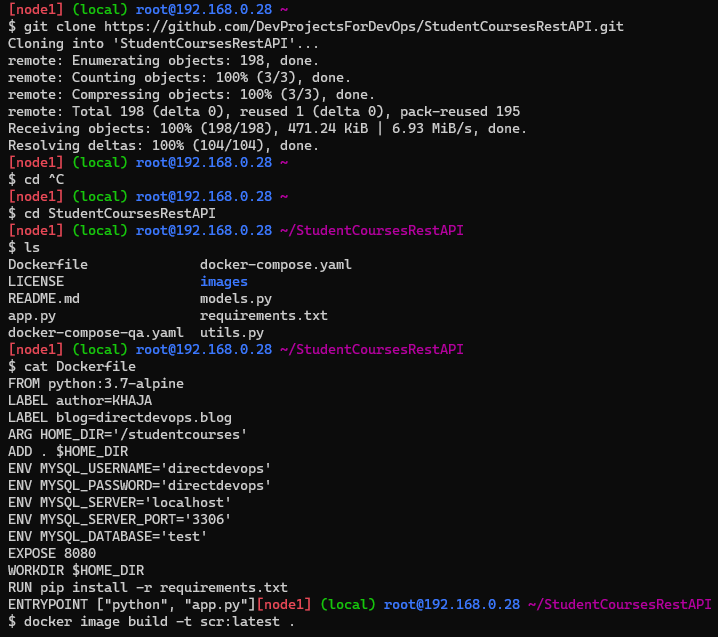
EXPOSE 8080

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

**OPEN MRS**

< git clone <https://github.com/DevProjectsForDevOps/StudentCoursesRestAPI.git>>

<cd StudentCoursesRestAPI>



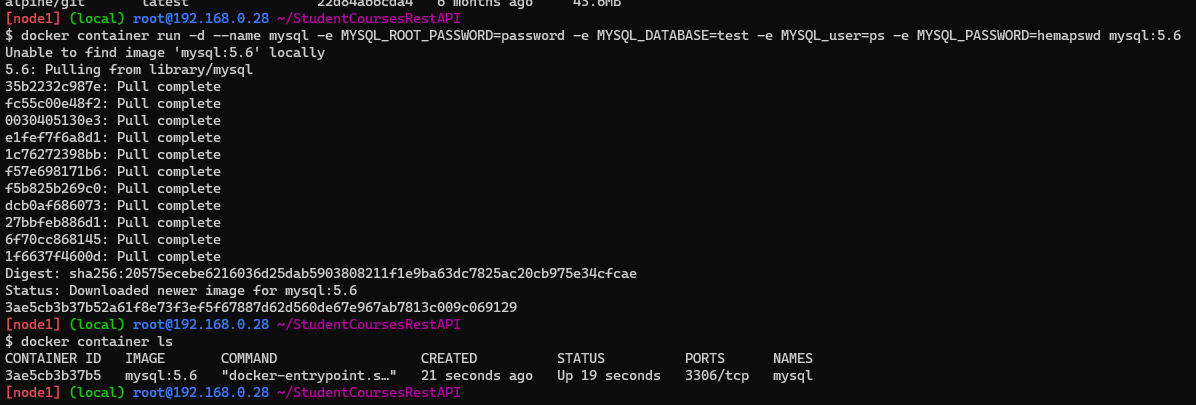
<cat Dockerfile>

To build the image,

<docker image build -t scr:latest **.>**

**Now create a mysql container**

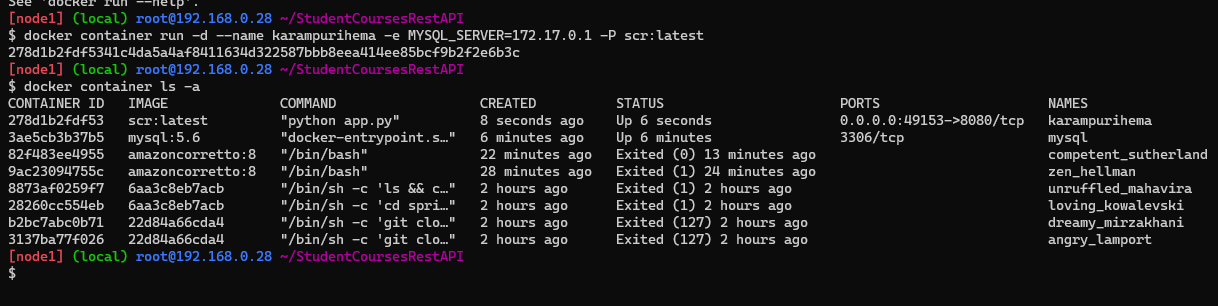
<docker container run -d --name mysql -e MYSQL\_ROOT\_PASSWORD=password -e MYSQL\_DATABASE=test -e MYSQL\_user=ps -e MYSQL\_PASSWORD=hemapswd mysql:5.6>

****

**Inspect the image**

**<docker inspect mysql>**

**docker container run -d --name karampurihema -e MYSQL\_SERVER=172.17.0.1 -P scr:latest**

****

