# Name: Neha Sharma MID: M1040187

# 

# DOCKER IMAGES 201

Problem Statement:

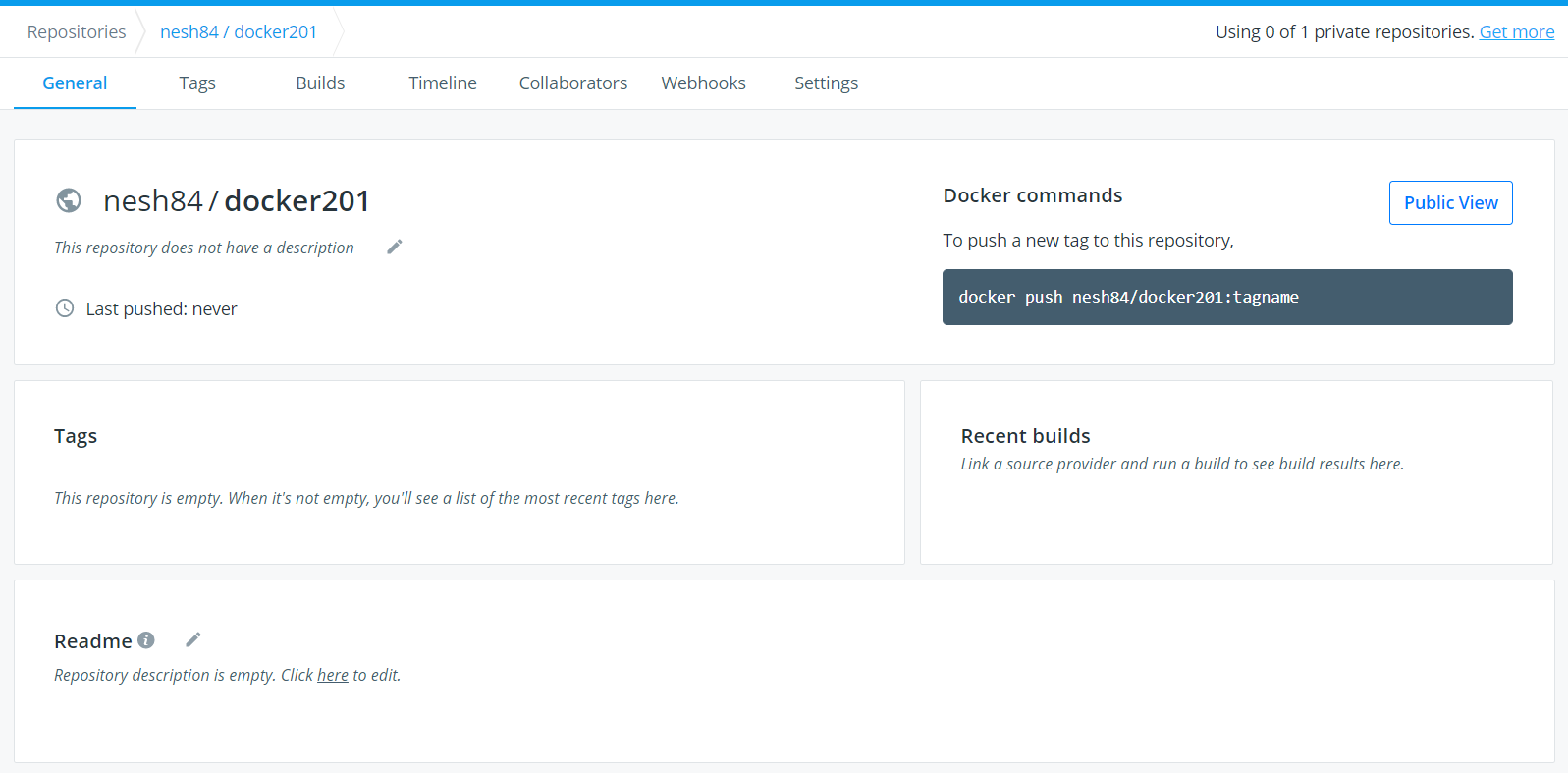
Build a 2-tier java application. Write a Docker File for creating an image & containerize the application. Push the image to Docker hub, deploy it and get all the parts to work together.

Execution Steps:

For the ease of execution, I have pushed my JDBC Spring Boot application to GITHUB.

Account Creation:

Create a Docker account hub official site ([https://hub.docker.com/).](https://hub.docker.com/).The) The created account is used to create a repository and push the Docker Images. Now we must create a repository.



* Clone the application from GITHUB that has to be dockerized into the Cloud Lab VM.
* In order to do the first open Cloud Lab with given credentials.

Docker File:

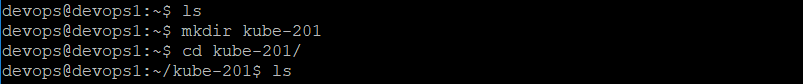
Docker fie is a text document that contains all the commands which assemble to create Docker Image. We need to create a Docker File to containerize the application. Docker can build the images by reading instructions from Docker File.

Steps to Create a Docker File:

We need to create a directory first to store the spring boot jar file and “Dockerfile” and build image of the jar file through docker and the subsequently to docker container.

Commands: “**mkdir docker\_201**” to create directory and type “**cd docker\_201**”

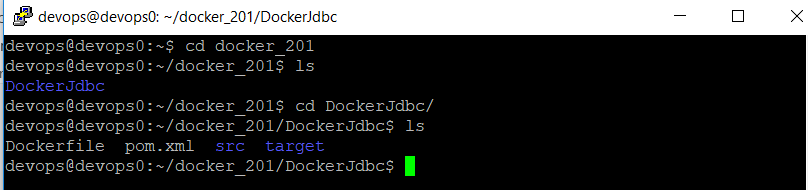




* Now create a docker File to write the configuration to build the image and then container.

Command: **“touch Dockerfile**”

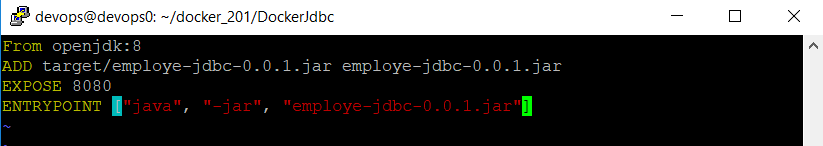
We can see the content of the home directory with “**ls**” command.



* To see/edit the contents of the “Dockerfile” the below command should be used.

Command: “vi Dockerfile”.

1. We have to add the below content in the Dockerfile.



Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

: wq is the command to save into the changes into the file.

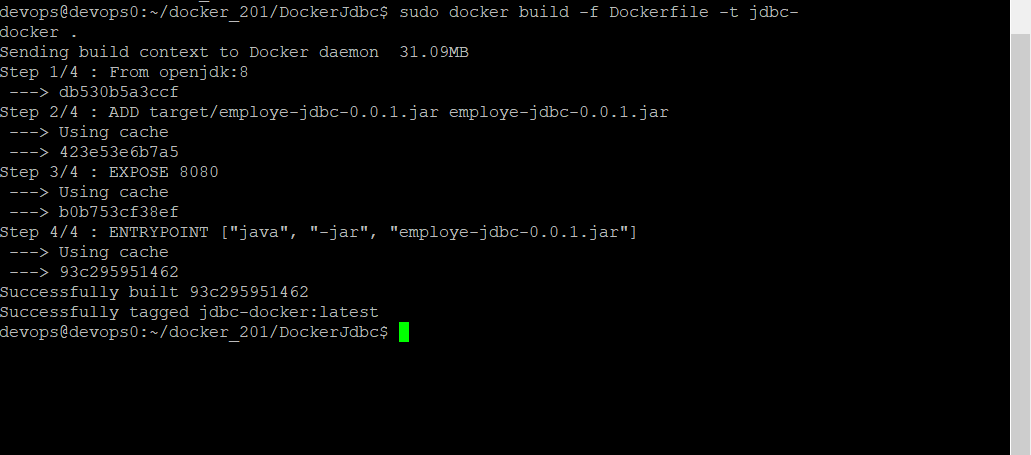
Let us understand each command of the Dockerfile.

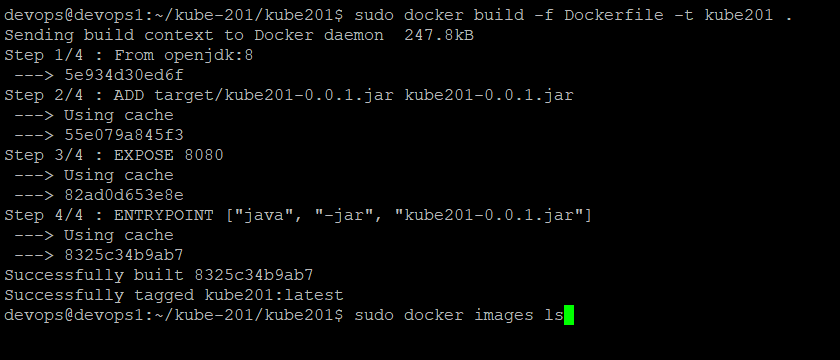
1. **FROM**: It is used to pull a public image from Docker-Hub and get that added to the build. In our case, it initializes an ubuntu instances with latest configuration.
2. **ADD:** It is used to copy new files, directories, or downloads from remote URLs.
3. **EXPOSE:** It ensures that the container that the application listens to specific port at runtime.
4. **ENTRYPOINT:** ENTRYPOINT allows you to configure a container that will run as executable.

**BUILDING THE DOCKER IMAGE USING DOCKERFILE**

Command: **sudo docker build -f Dockerfile -t jdbc-docker .**

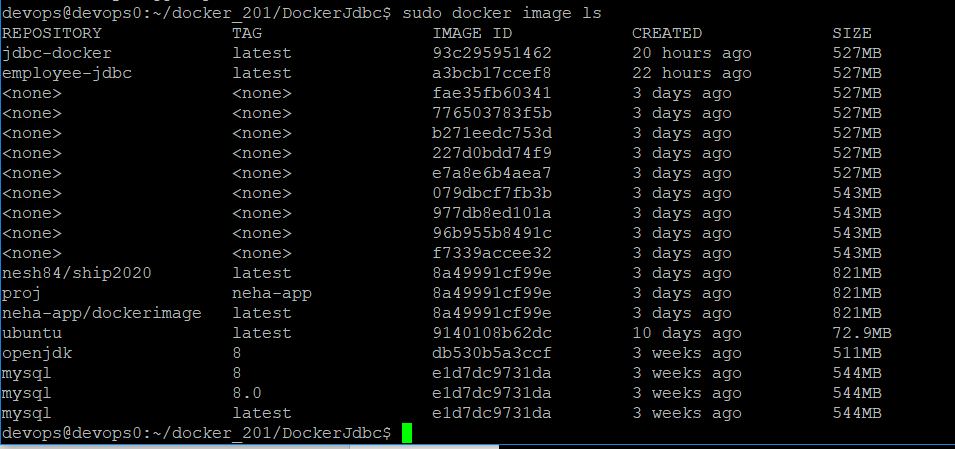
Note: Here docker build is the command and -t jdbc-docker is the tag we are assigning to the image.

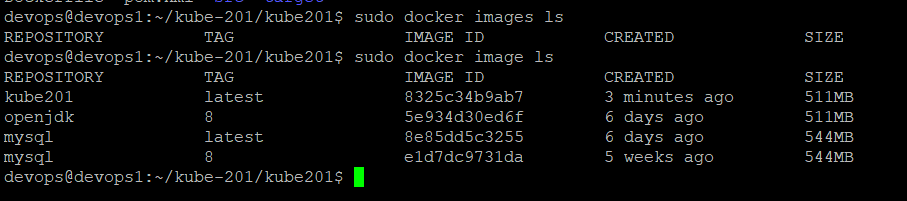




To check if the docker image is created properly we use the following:

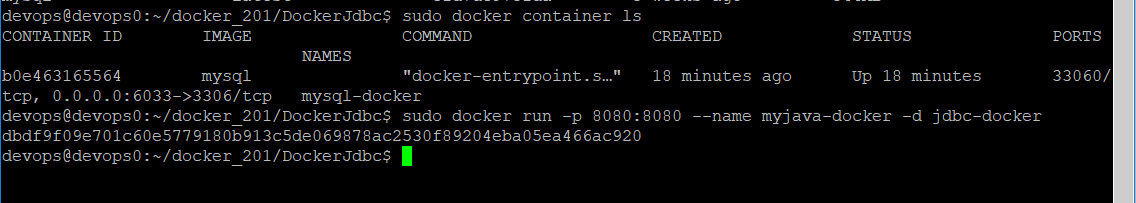
Command: **sudo docker image ls**

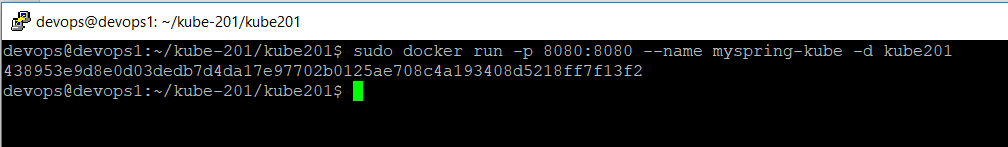




**Creating Container for Image**

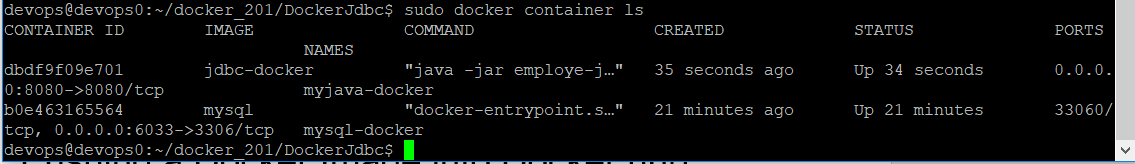
Command: **sudo docker run -p 8080:8080 --name myjava-docker -d jdbc-docker**



3

Note: -name is the command to give the <any name> of the container and -d is to give command which image container need to be linked.

To check the container, which container is running: **sudo docker container ls**

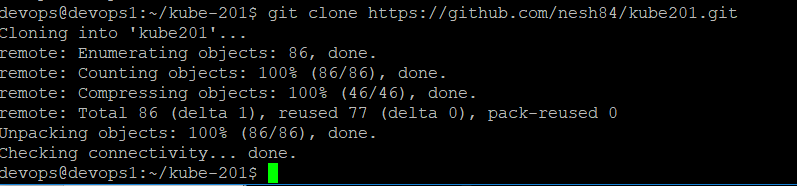


Pushing a Docker Image into Docker hub

Now that we have create our image the next step to push the data into our docker-hub. The command for pushing the image is given below.

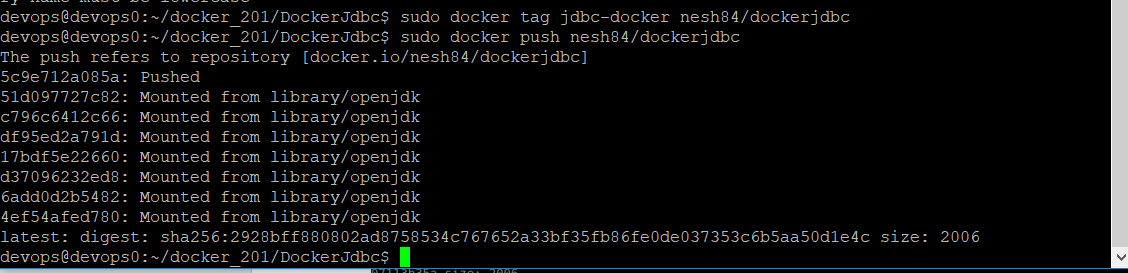
Note: To push clone git code in docker hub you can either use below.

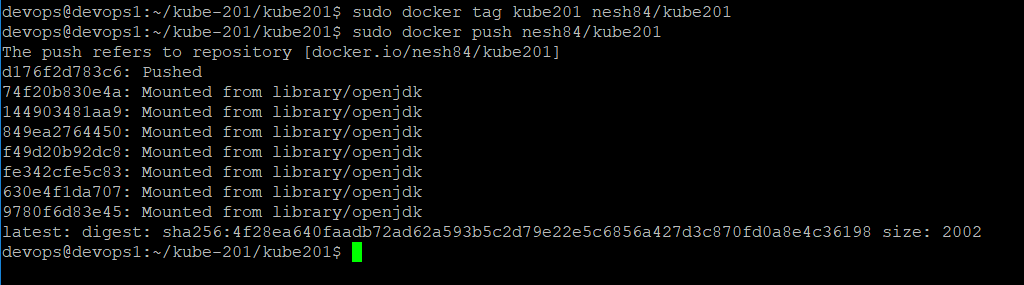
Command: **git clone** [**https://github.com/nesh84/DockerJdbc.git**](https://github.com/nesh84/DockerJdbc.git)



Command: **sudo docker tag jdbc-docker nesh84/dockerjdbc**

Command: **sudo docker push nesh84/dockerjdbc** (*name should be given in small letters*)



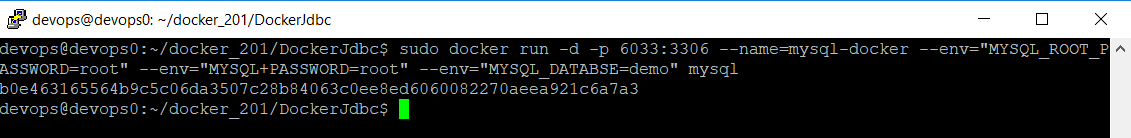


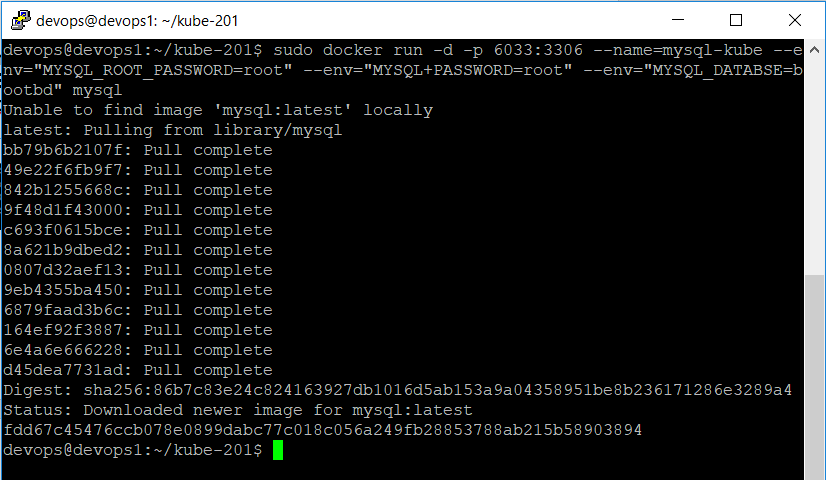
**RUNNING DOCKER-CONTAINER FOR MYSQL**

Note: Use this command if same name container issue appears:

**sudo docker rm mysql-docker**

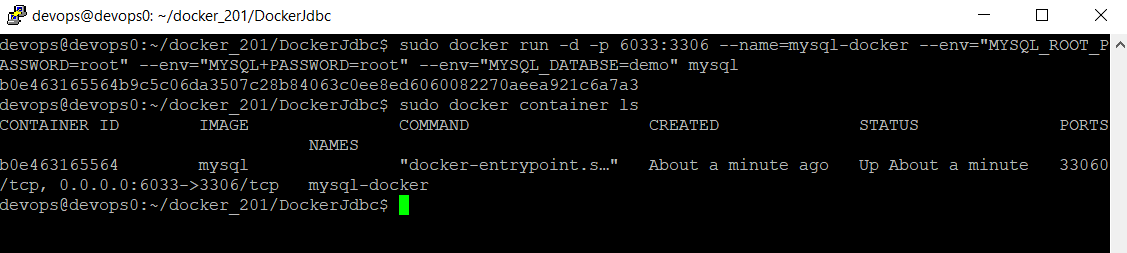
Command**: sudo docker run -d -p 6033:3306 --name=mysql-docker --env="MYSQL\_ROOT\_PASSWORD=root" --env="MYSQL+PASSWORD=root" --env="MYSQL\_DATABSE=bootdb" mysql**





To see the container running in container:

Command: **sudo docker container ls**



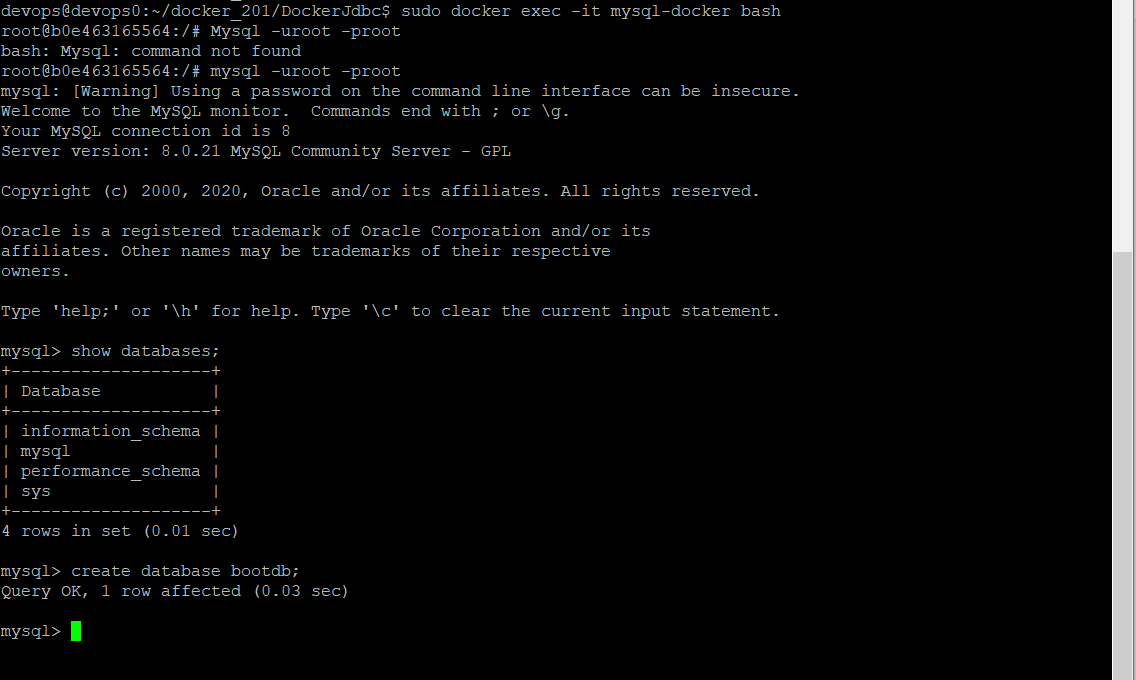
To check whether MySQL is running perfectly in docker container

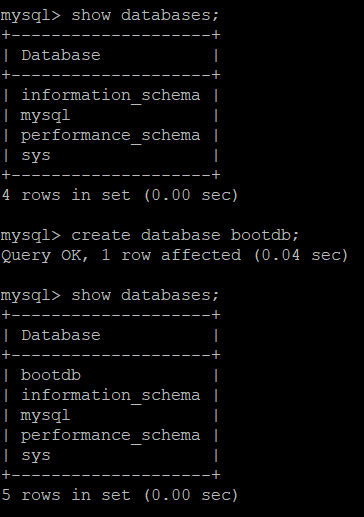
Command: **sudo docker exec -it mysql-docker bash**

Command: **mysql -uroot -proot**



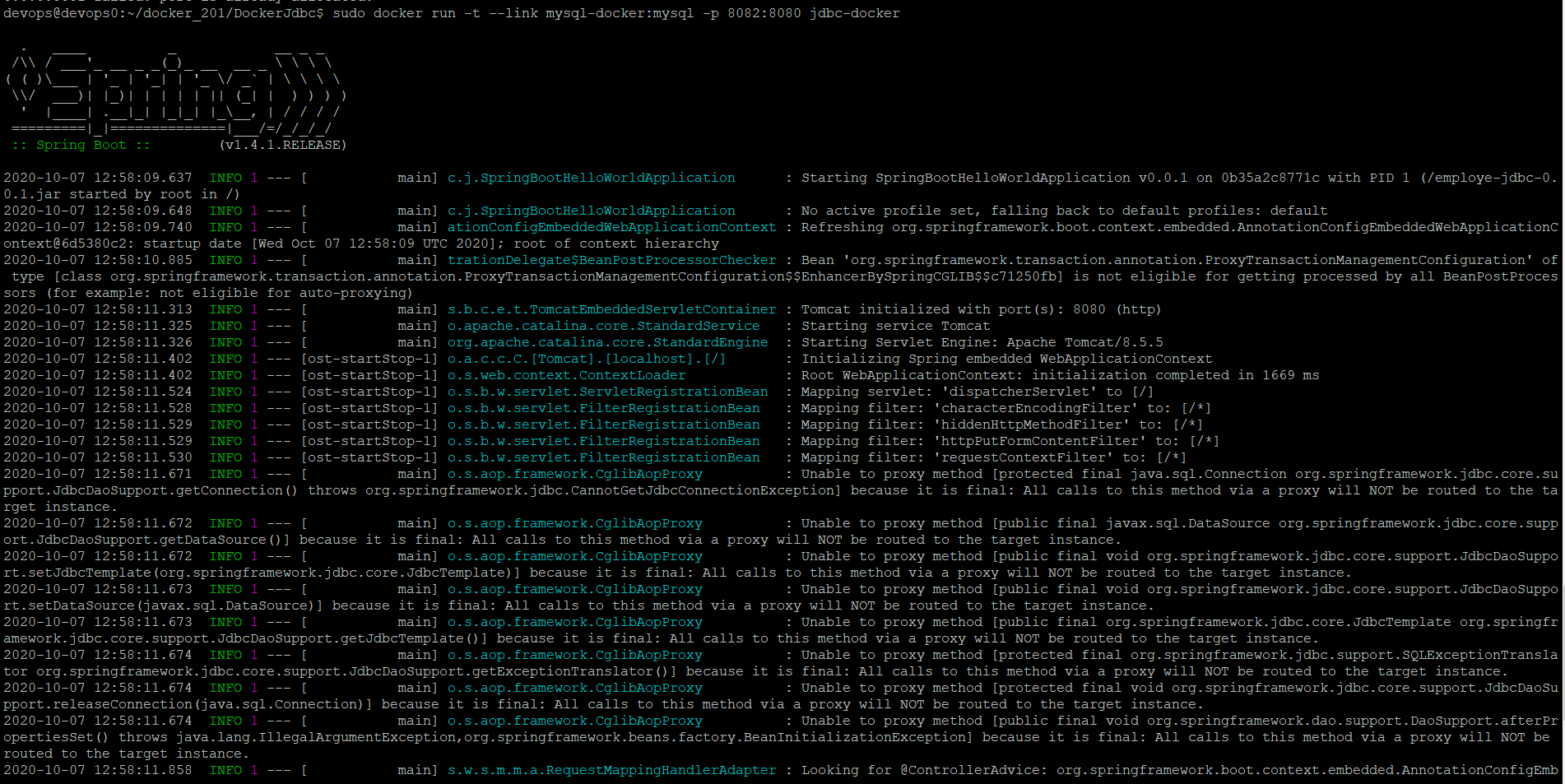






**RUNNING SPRINGBOOT WITH SQL CONTAINER**

Command: **sudo docker run -t --link mysql-docker:mysql -p 8081:8080 jdbc-docker**





**Running SPRING BOOT && MYSQL with DockerCompose**

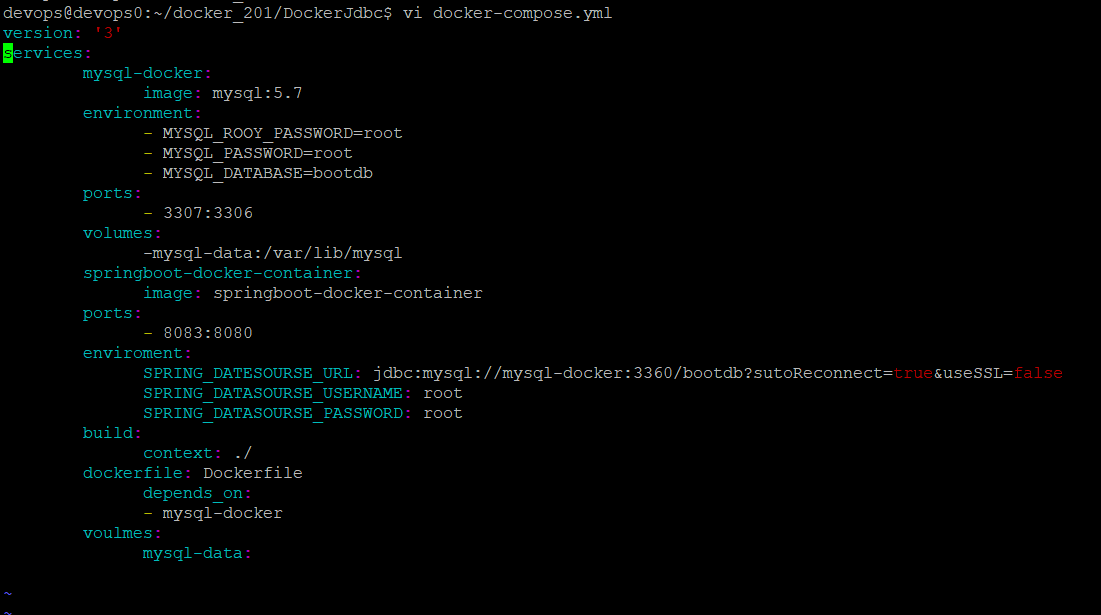
Now we must create a docker-compose.yml file by using below command.

Command: **touch docker-compose.yml**

To edit the docker-compose.yml, we use the below command.

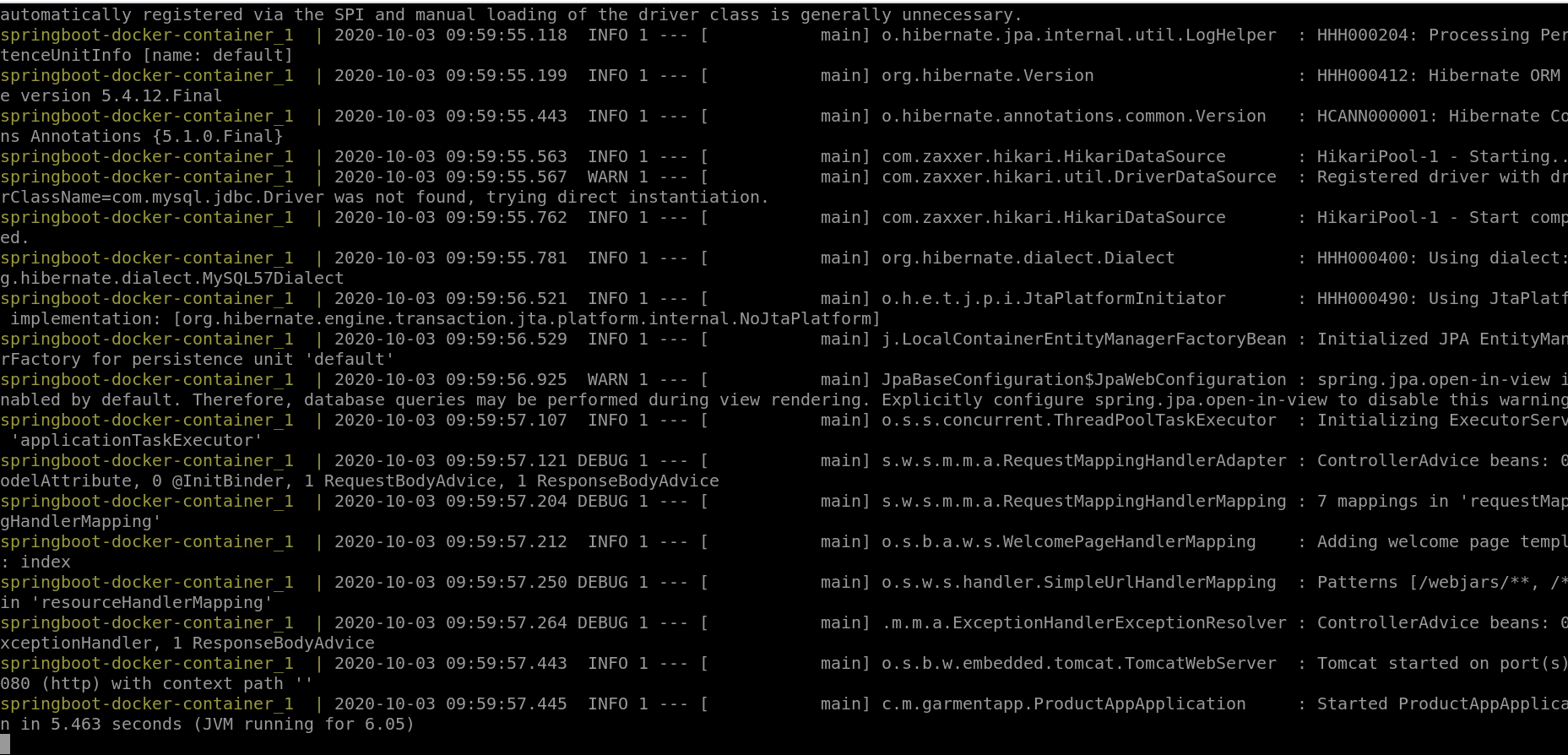
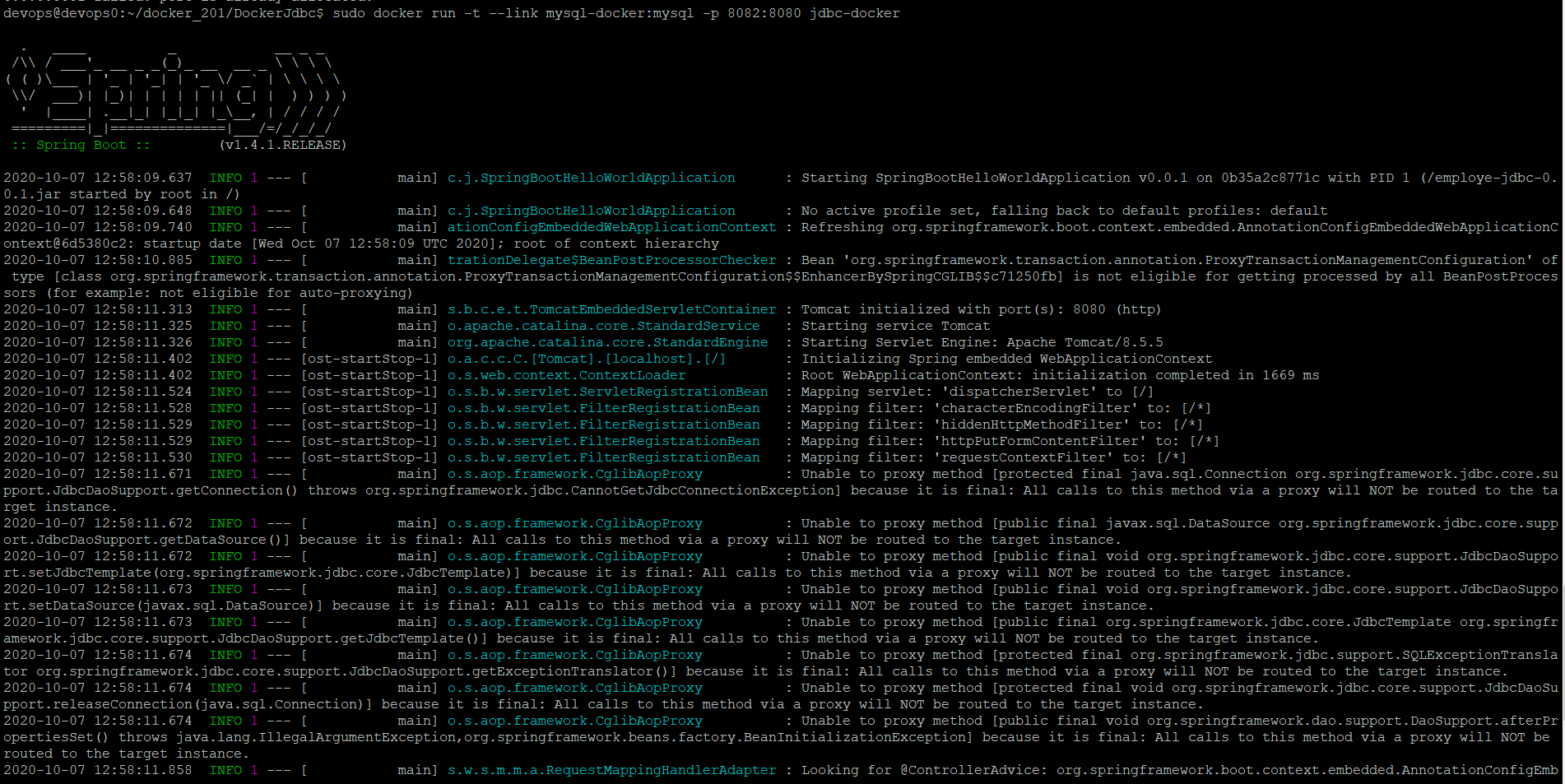
Command: **vi docker-compose.yml**

We need to write the script in that file in order to run MySQL and spring boot application.



Command to make the container up: **sudo docker-compose up**

Command to make the container down: **sudo docker-compose down.**

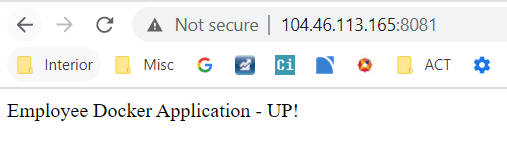


Now we can safely start the application.

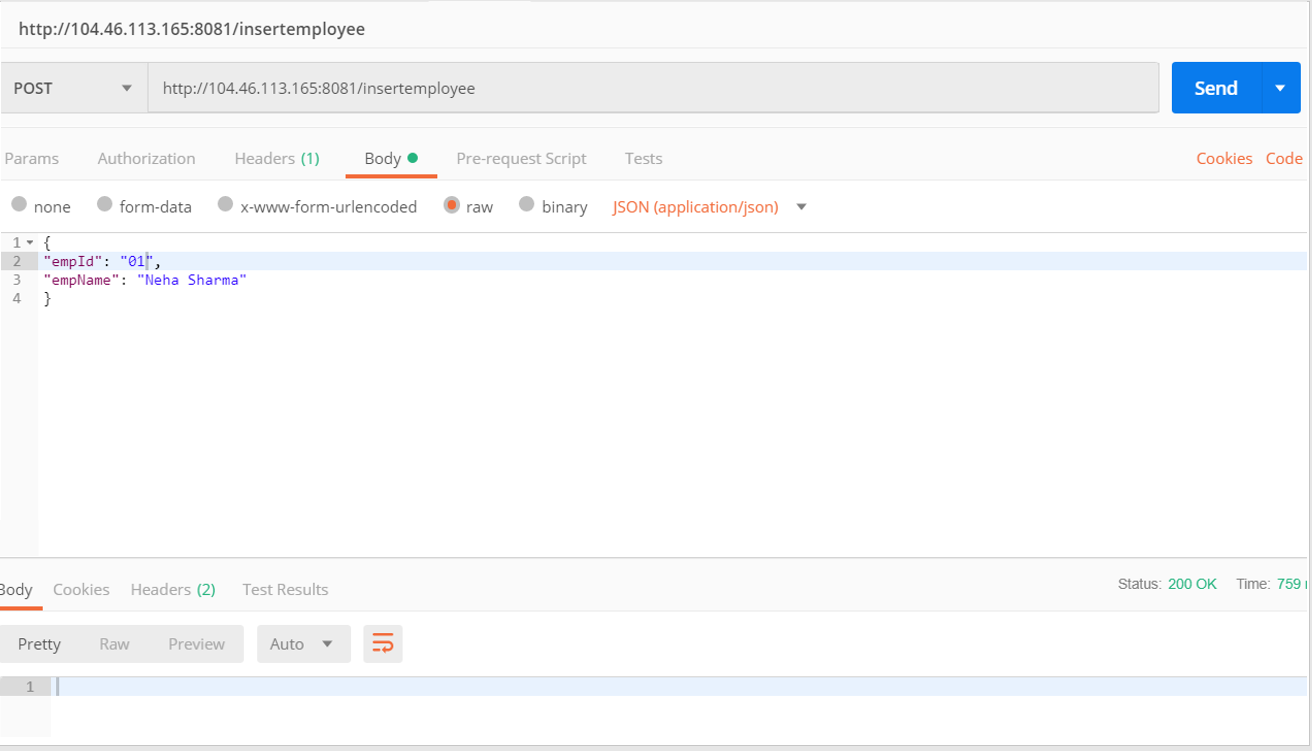
But in order to we have to know the hostname in which the application can be started.

Command to know the command: **curl ifconfig.me**

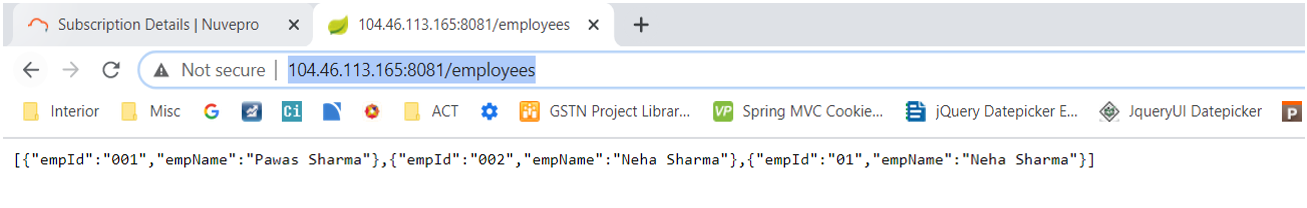
IP Address: 104.46.113.165



Add:



View:



Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated A screen shot of a computer

Description automatically generated with medium confidence Graphical user interface

Description automatically generated with low confidence

Graphical user interface, text, application

Description automatically generated

CodeCoverage:-

Graphical user interface, text, application, chat or text message

Description automatically generated

Sonar lint – review:

A picture containing text, screenshot, computer, indoor

Description automatically generated

Customer junit:

Graphical user interface, text, application

Description automatically generated

Sonar -customer:

Graphical user interface, text, application, email

Description automatically generated

Great ☺