### Table of Contents:

[Table of Contents:](#_fc88352twl0d)

[Step -1 : Update the ubuntu index by using update command](#_juobz73mu91t)

[Step-3 : Verify the docker Version by running](#_hoqju2s30q1o)

[Step-4 : Pull the Public docker image from dockerHub](#_kufajxx80adv)

[Step-5 : Run the Mysql Container from the pulled image.](#_uz9lhvqo54rp)

[Step-6 : Run the nginx Container from the pulled image.](#_3hirkarou8kf)

[Projects :](#_lp67trrunw4)

[JAVA Project](#_kjf5v6t1uwpd)

[NodeJs Project](#_e0a69ljv1zt)

[Python-Flask Project](#_b2va5l7xwwyi)

[two-tier Project](#_zaxgzkyo6axy)

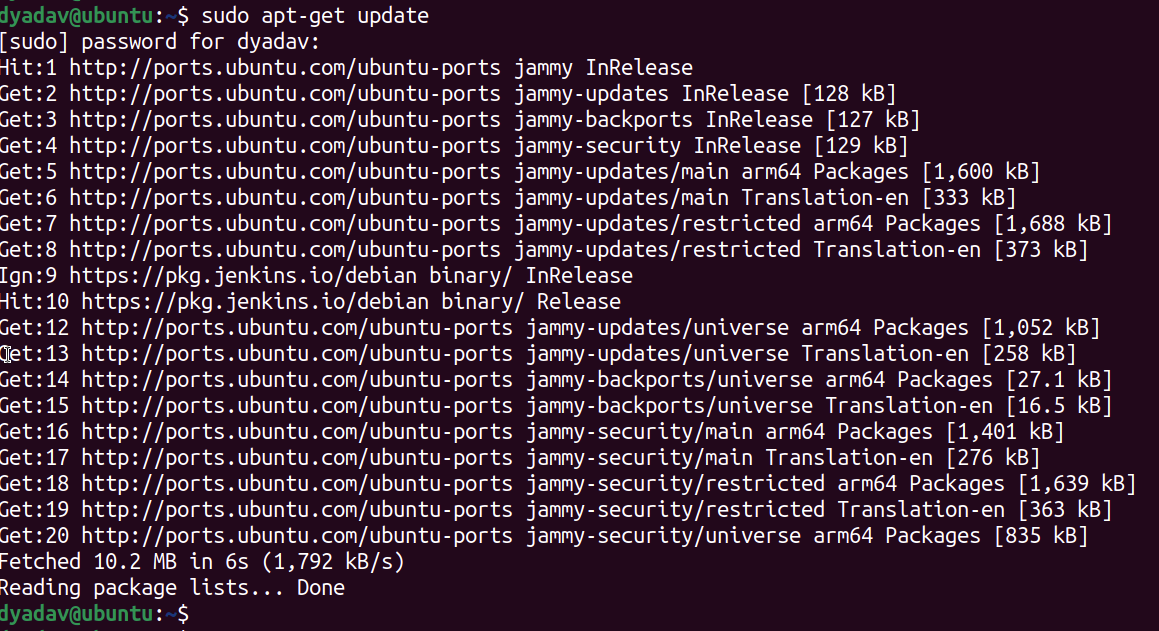
[Docker Compose](#_bkcfxqkaduly)

[Multi-Stage docker build:](#_ogmh9xv9t1zu)

[Docker scout](#_jaaossoh2vvh)

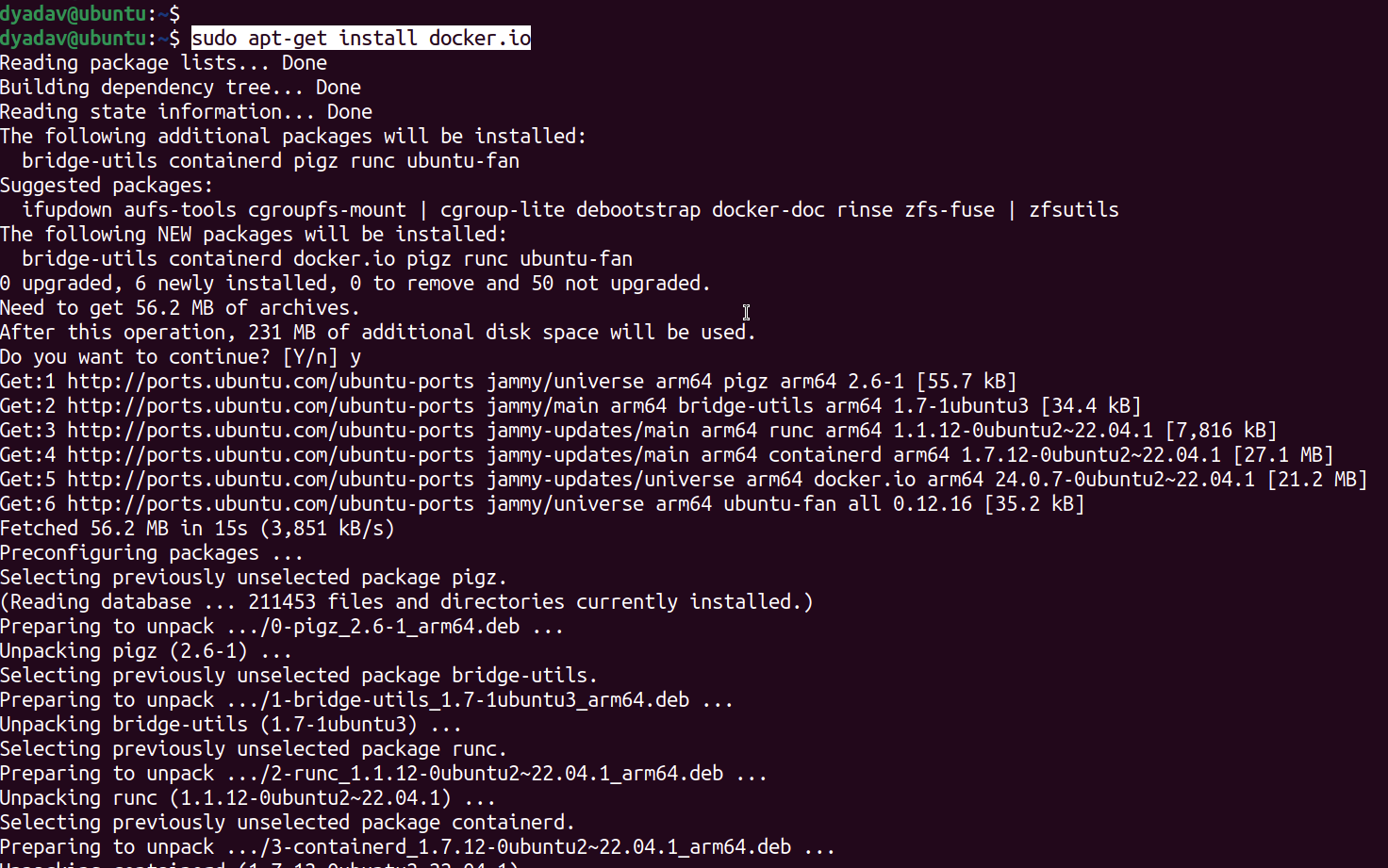
### **Step -1** : Update the ubuntu index by using update command

**$ sudo apt-get update**



**Step -2** : Install the Docker in Ubuntu Linux Machine.

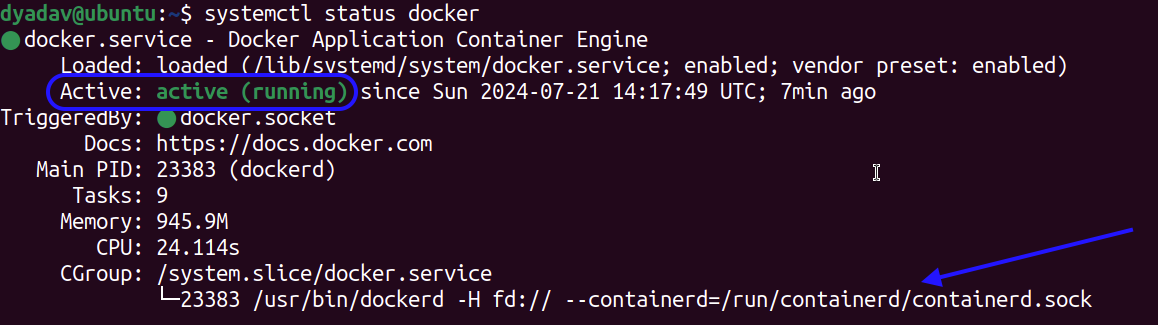
**$ sudo apt-get install docker.io**



### **Step-3** : Verify the docker Version by running

**$ docker –version** : Check the version, Also add your current user to docker Group.

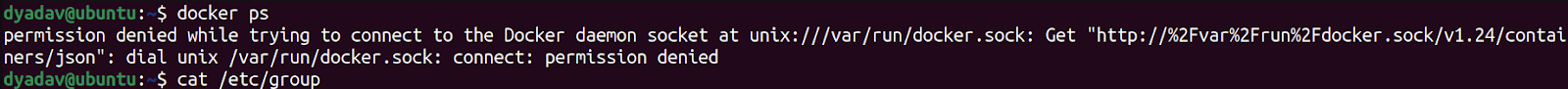
**$ systemctl status docker** : to check the Docker Service Status.



**$ usermod -aG docker $USER** : this command will add your user into Docker Group

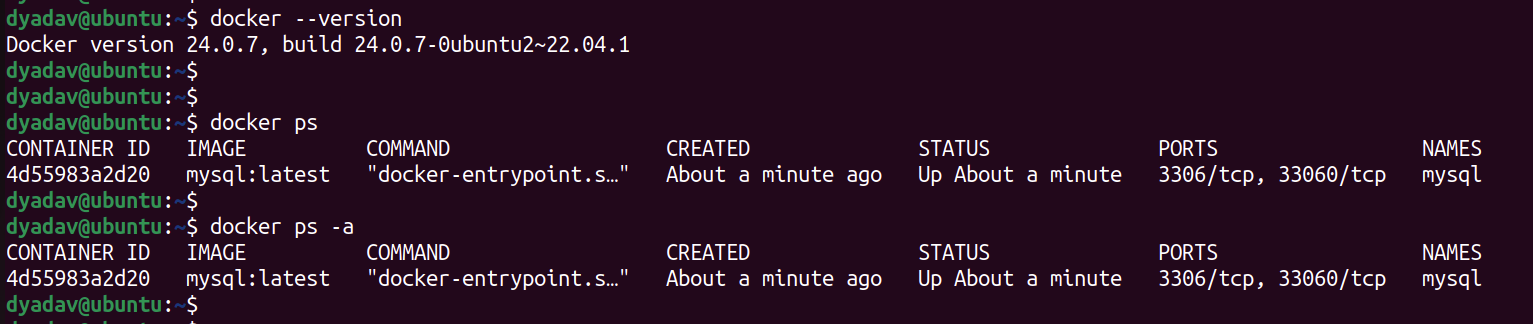
**$ cat /etc/group** : now your username is a part of the docker group.

You will see this error message before adding your user to docker group.



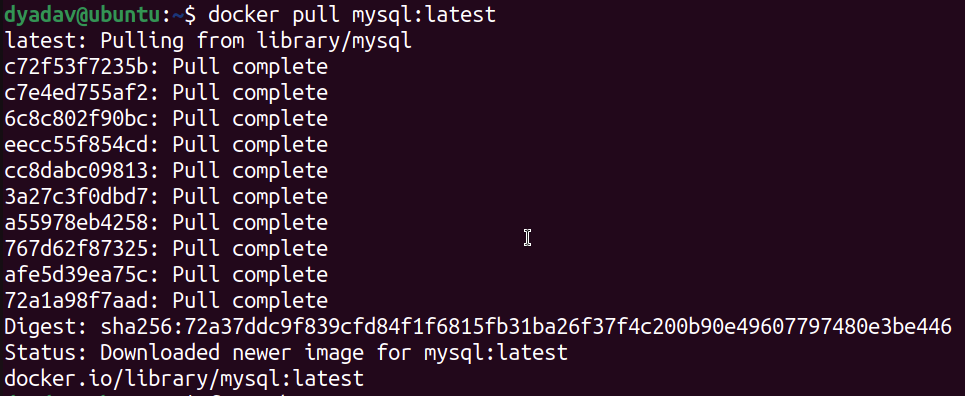


**$ newgrp docker** : to activate the new changes hence no need to restart the Ubuntu Linux Machine.

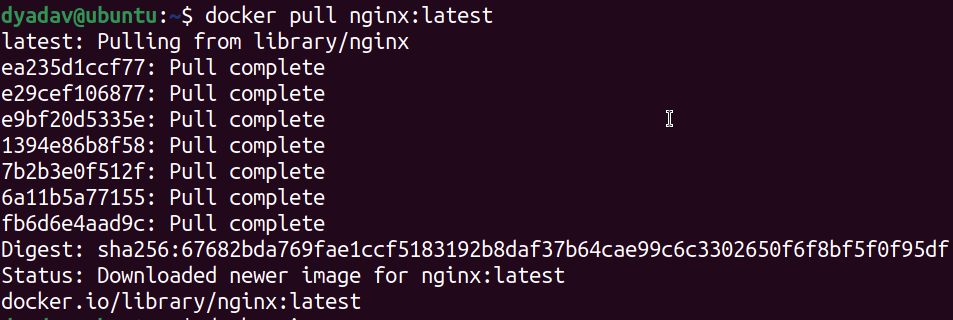


### **Step-4** : Pull the Public docker image from dockerHub

**$ docker pull mysql:latest** :--> To pull the mysql docker image from dockerhub



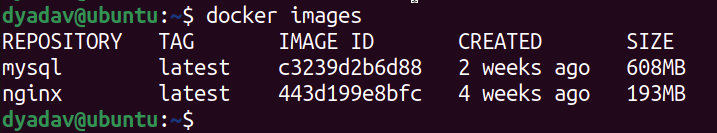
**$ docker pull nginx:latest** :----> To pull the nginx docker image from dockerhub



Once it’s pulled successfully, Use the below command to check docker images.

**$ docker images**

**$ docker rmi <ImageID> :**To remove the Image



Note : we create an image from file & container from build Image.

**File —-> Image —-> Container**

**$ docker rmi -f $(docker images -aq)** : To Delete all the docker images at Once.

### **Step-5** : Run the Mysql Container from the pulled image.

**$ docker run -d --name mysql -e MYSQL\_ROOT\_PASSWORD=root mysql:latest**

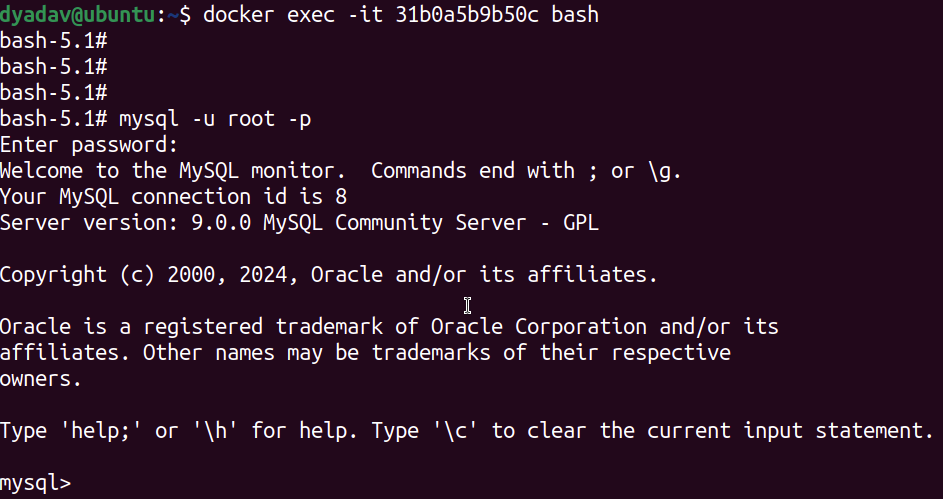
-d : run the container in detached mode

-e : give the environment variable

### 

Use the below command to get into mysql Container

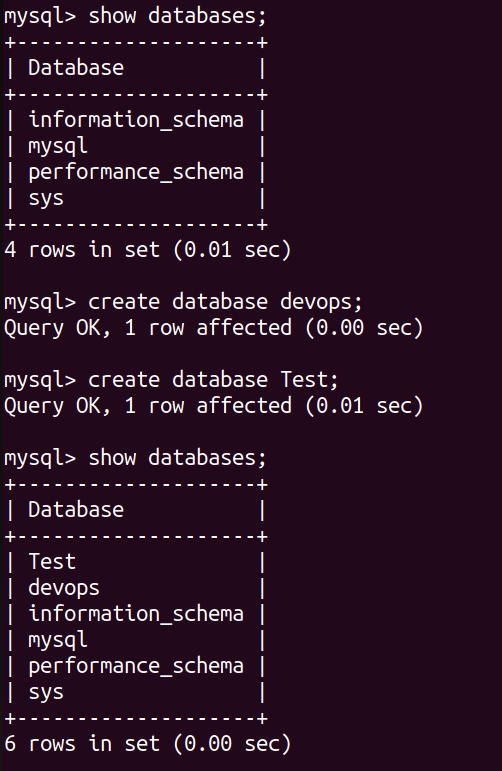
docker exec -it <<Container ID >>bash



use below command to check the databases & create it.

**show databases;**

**create database devops;**



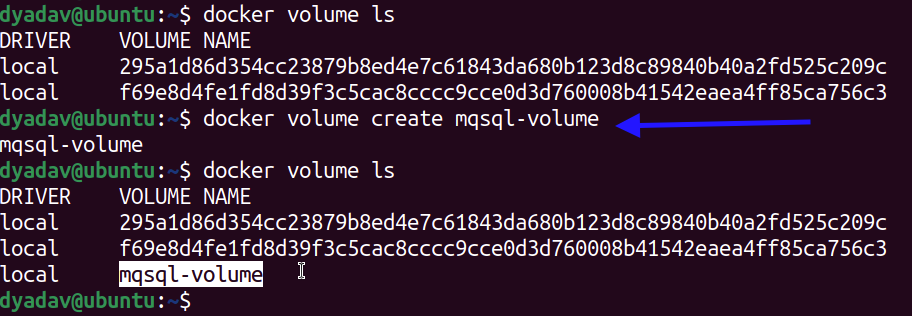
**Challenge:** Once you delete the Container, you lose the Database as well.

**Solution:** Add the Volume & bind the volume with Container.

**docker volume for persistent :**

**$ docker volume ls** : List down all the Volume

**$ docker volume create mqsql-volume** : it will create a volume

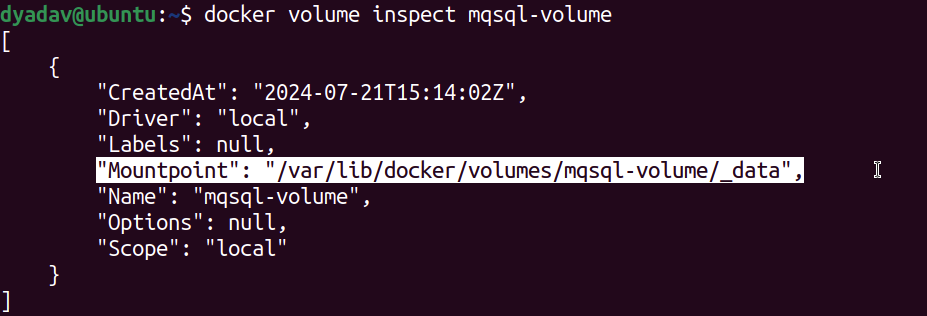


**$ docker volume remove <Volume-Name>** : To remove the created docker Volume.

**$ docker volume inspect mqsql-volume** : to check the docker volume path

**$ docker volume prune** : to Delete the non-Use Volume.

**Mysql default path :** var/lib/docker/volumes/mqsql-volume

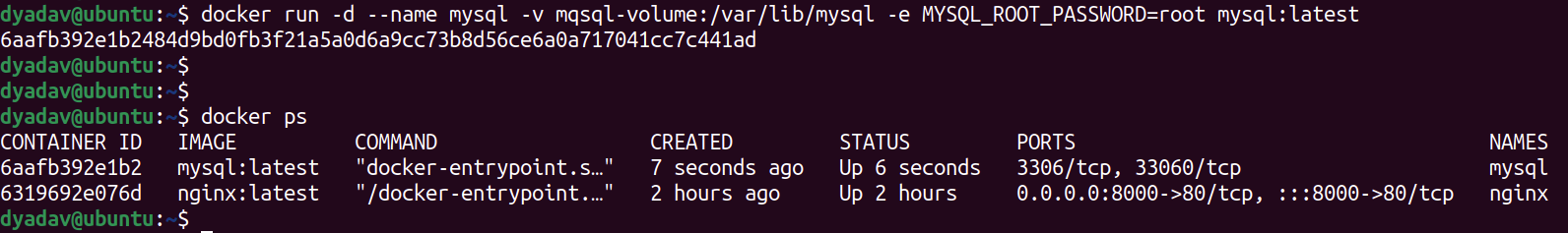


Now create a docker image with Volume.

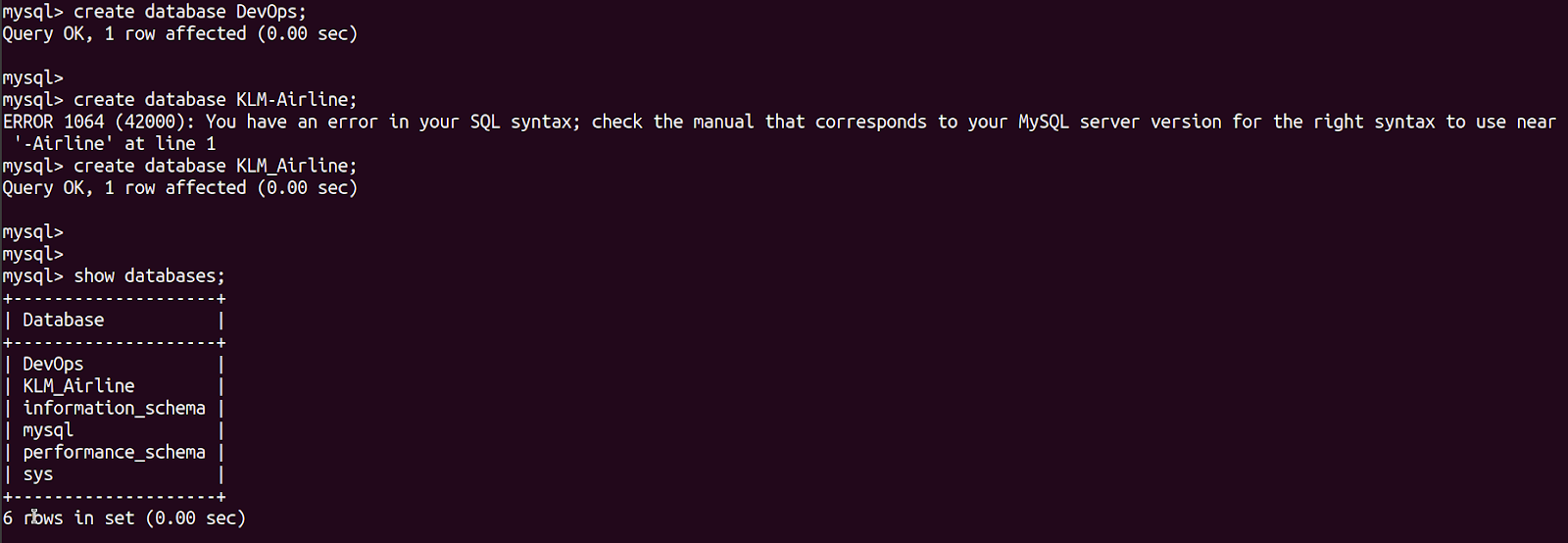
$ docker run -d --name mysql -v mqsql-volume:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=root mysql:latest

- v : for Volume & give volume Name:**/<<mysql database path>>**

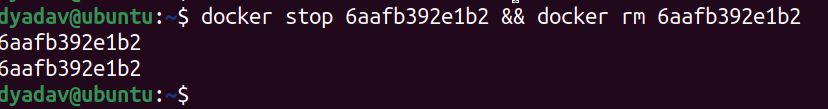
**/var/lib/mysql :** is the default mysql database path



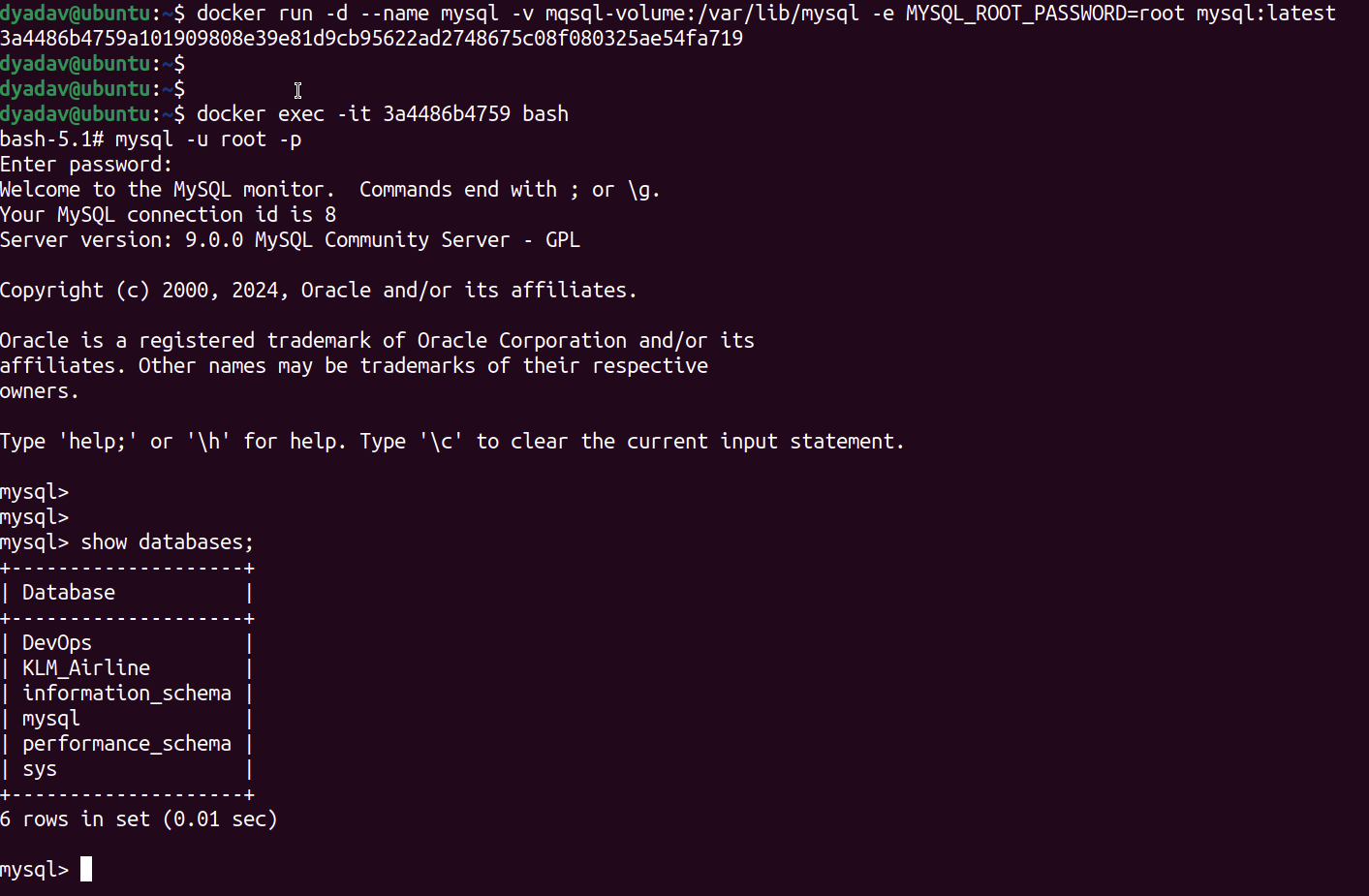
===> Created 2 Database



===> Deleted the Container & built the container again.



===> Created the Container again & we can see our old created database.

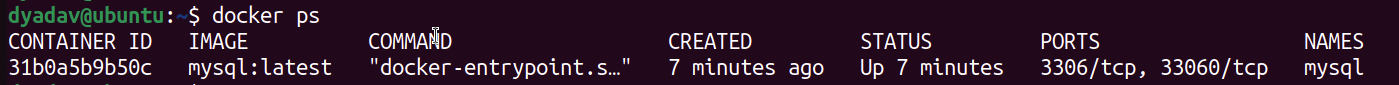


### **Step-6** : Run the nginx Container from the pulled image.

**Verify the container using below commands:**

**$ docker ps :** shows all the running containers

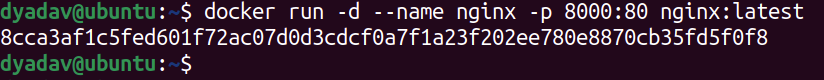
**$ docker ps -a :** shows running & stopped both.

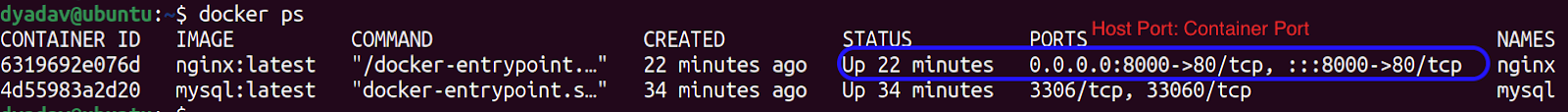


==> Install the nginx container:

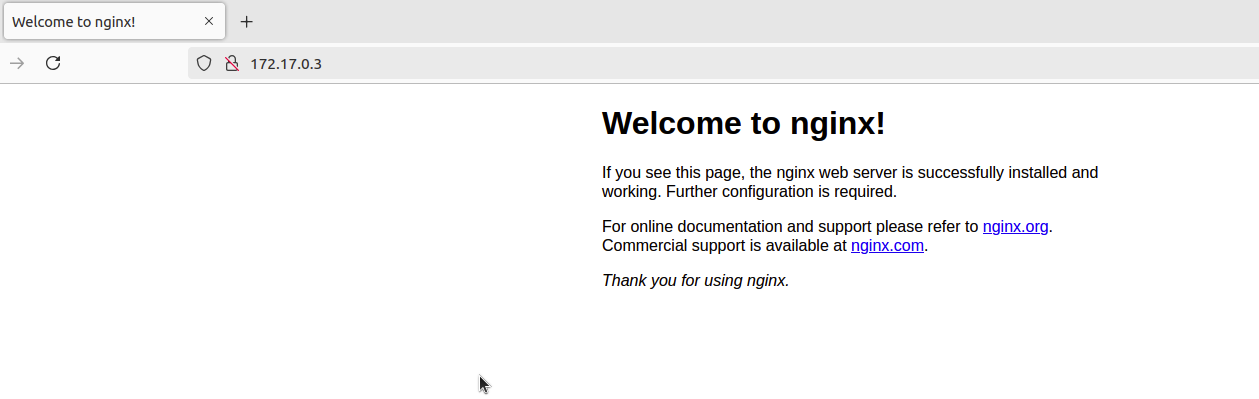
**$ docker run -d --name nginx -p 8000:80 nginx:latest**

**- p :** publish port 8000 “Local Host Port” & 80 “nginx Container port.





use the Private Ip Address:80 in ubuntu Linux Desktop



### **Projects** :

### **JAVA Project**

Create a Dockerfile to build a image

1. Clone github project on ubuntu machine.

**git clone https://github.com/dheeruyadav54/simple-java-docker.git**

vim Dockerfile —> Create a Dockerfile with Capital D.

**# Write the Dockerfile per below format.**

**# Base Image of Java**

FROM openjdk:17-jdk-slim

**#Create a Working Directory**

WORKDIR /java

**# Copy the Code**

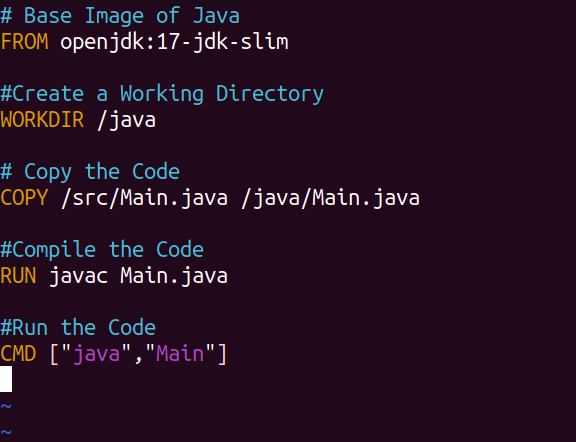
COPY /src/Main.java /java/Main.java

**#Compile the Code**

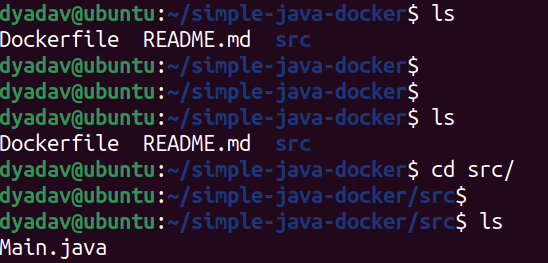
RUN javac Main.java

#Run the Code

CMD ["java","Main"]



Here is the Code & directory for Main.Java

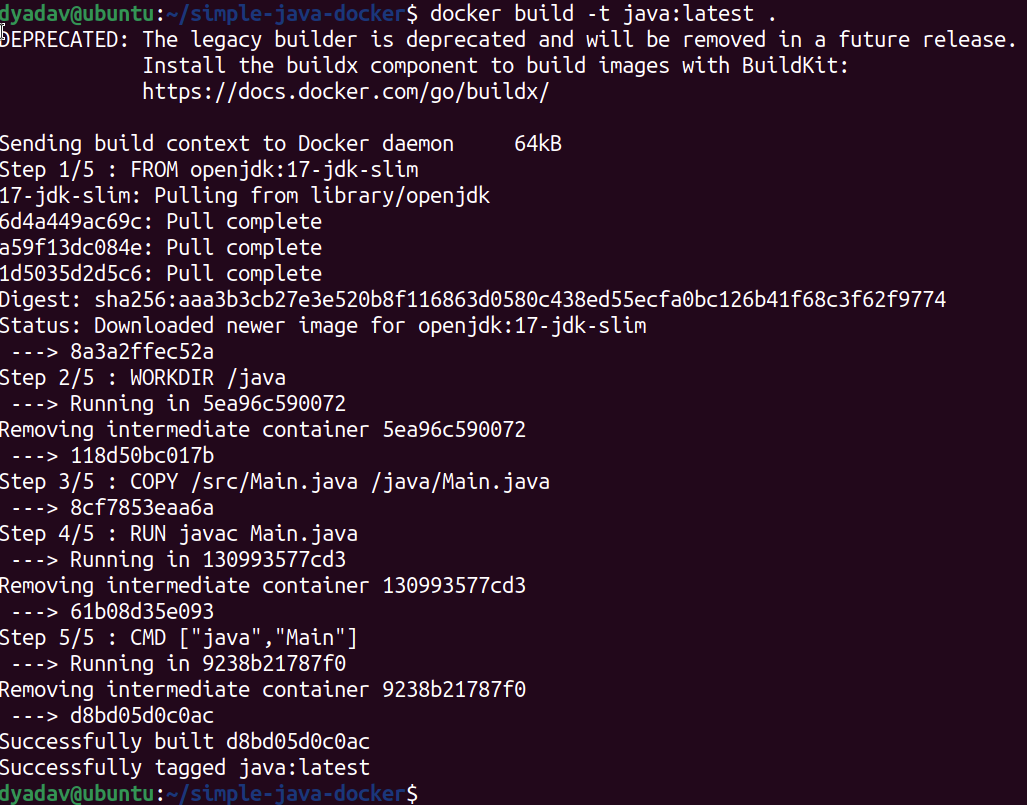


1. Build the image from above Dockerfile

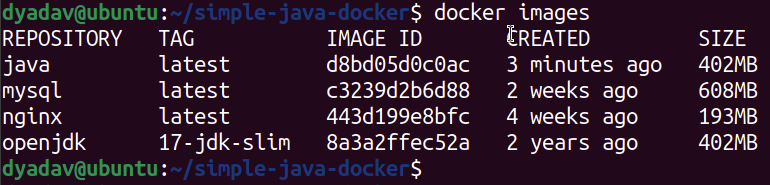
**$ docker build -t java:latest .**

-t : for Tag name

. : Dot means Current Directory Dockerfile

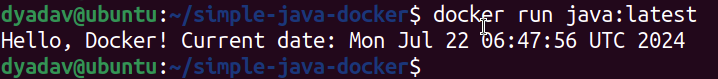


===> Java Docker Images has been created,

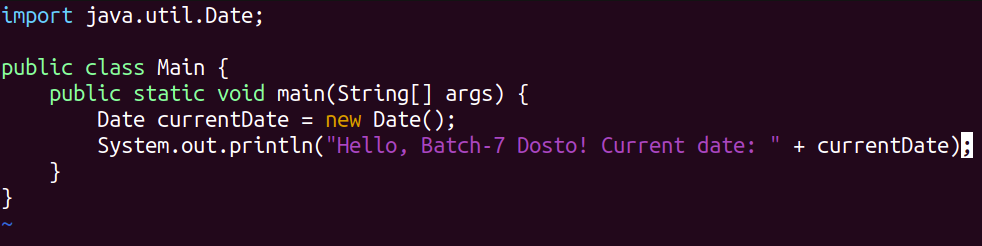


1. now run the Container per build image.

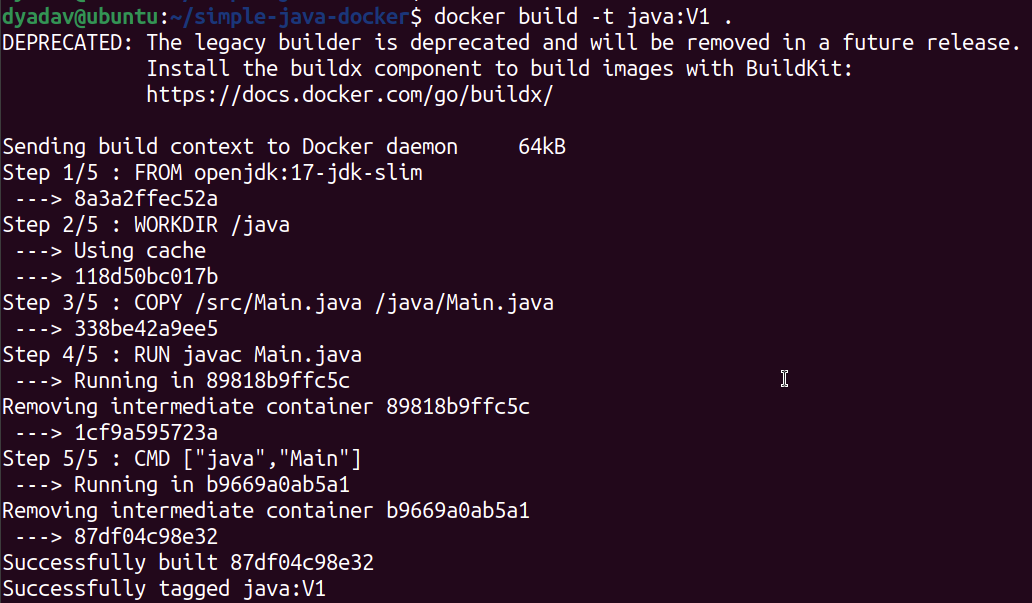
===> JAVA Code running successfully



1. Let’s modify the JAVA Code & rebuild the image



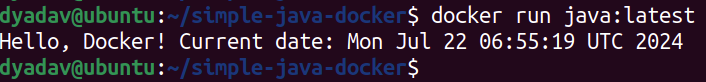
===> Again Rebuild the Image with tag Name Java:V1



====> Run the Java Image with New Tag Name



====> With OLD Tag Name

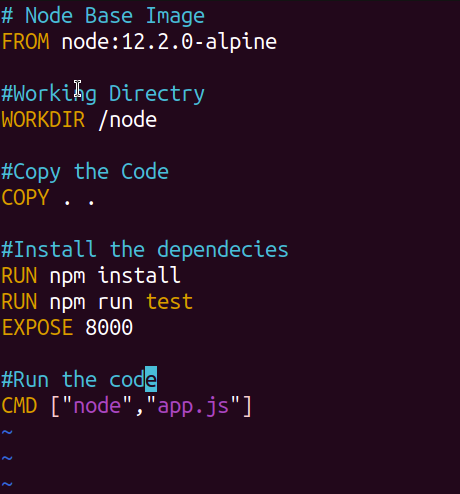


### **NodeJs Project**

1. Clone github project on ubuntu machine.

**git clone https://github.com/dheeruyadav54/node-todo-cicd.git**

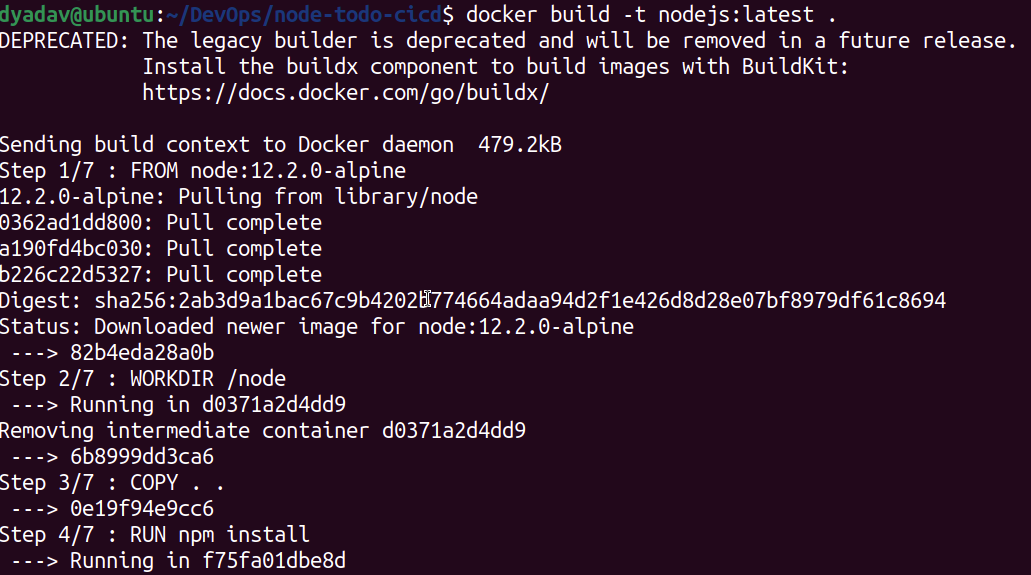
vim Dockerfile —> Create a Dockerfile with Capital D.

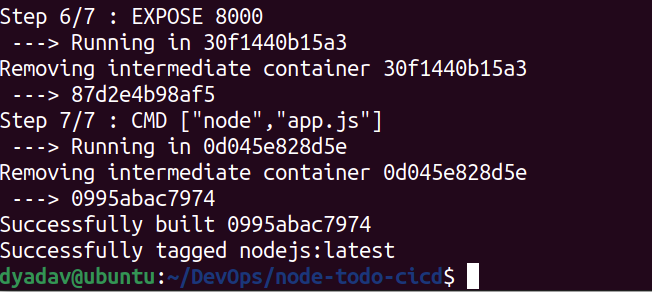


1. Create a .dockerignore file

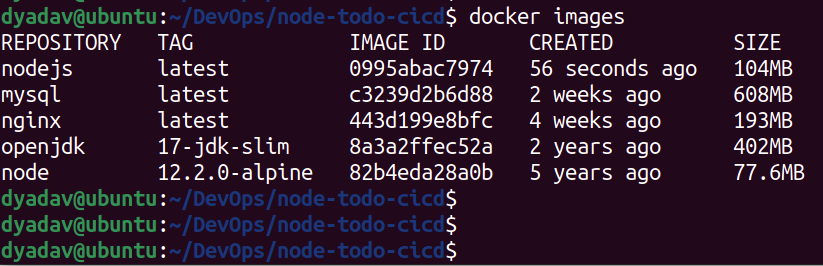
**$ vim .dockerignore :** Add all the files & folder which you don’t want to copy from Local to Container.

**$ docker build -t nodejs:latest . :** Build a nodejs image from Dockerfile



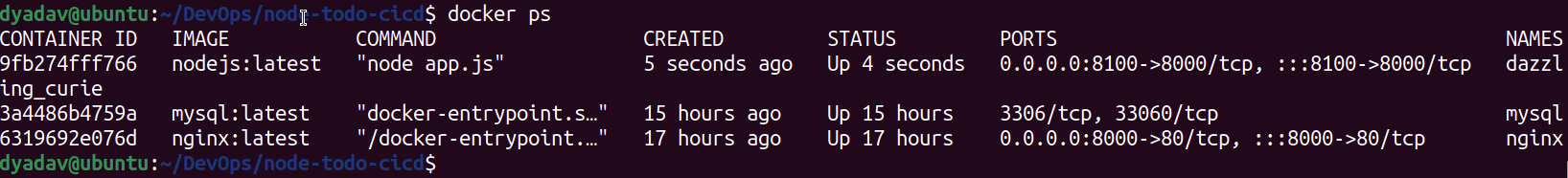


===> Nodejs images have been built.

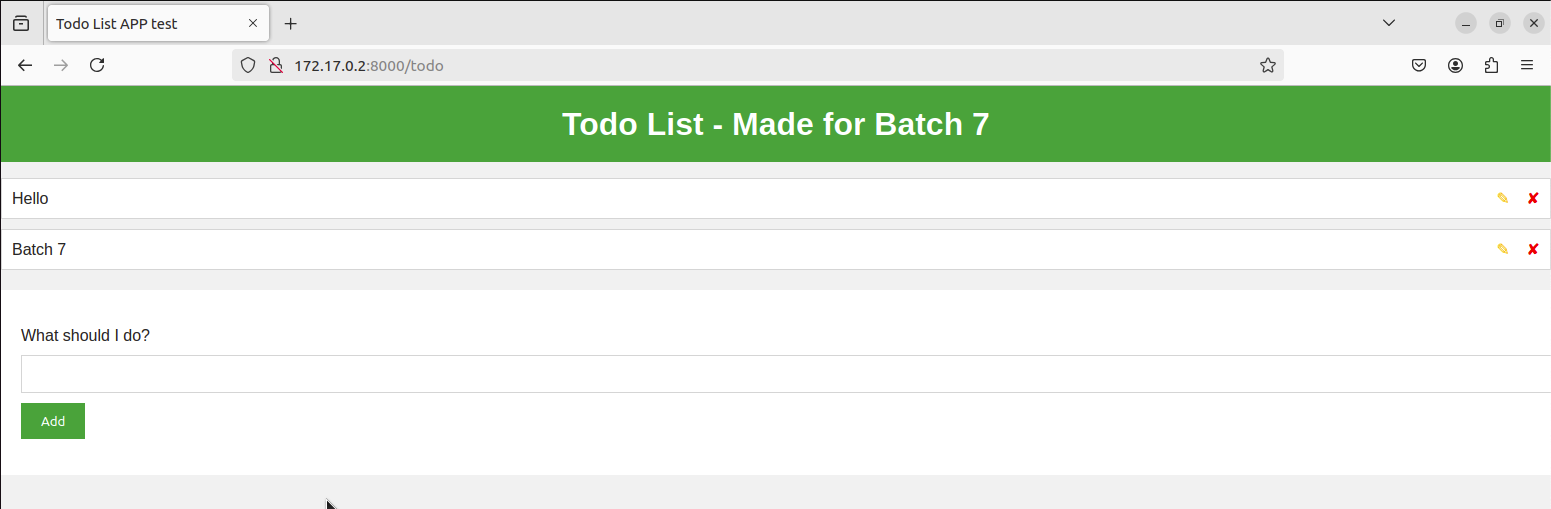


1. Let’s run the container.

**$ docker run -d -p 8000:8000 nodejs:latest**



nodejs container is running & serving the web page.



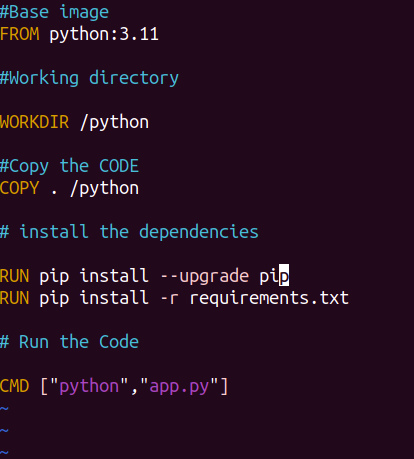
### **Python-Flask Project**

1. Clone github project on ubuntu machine.

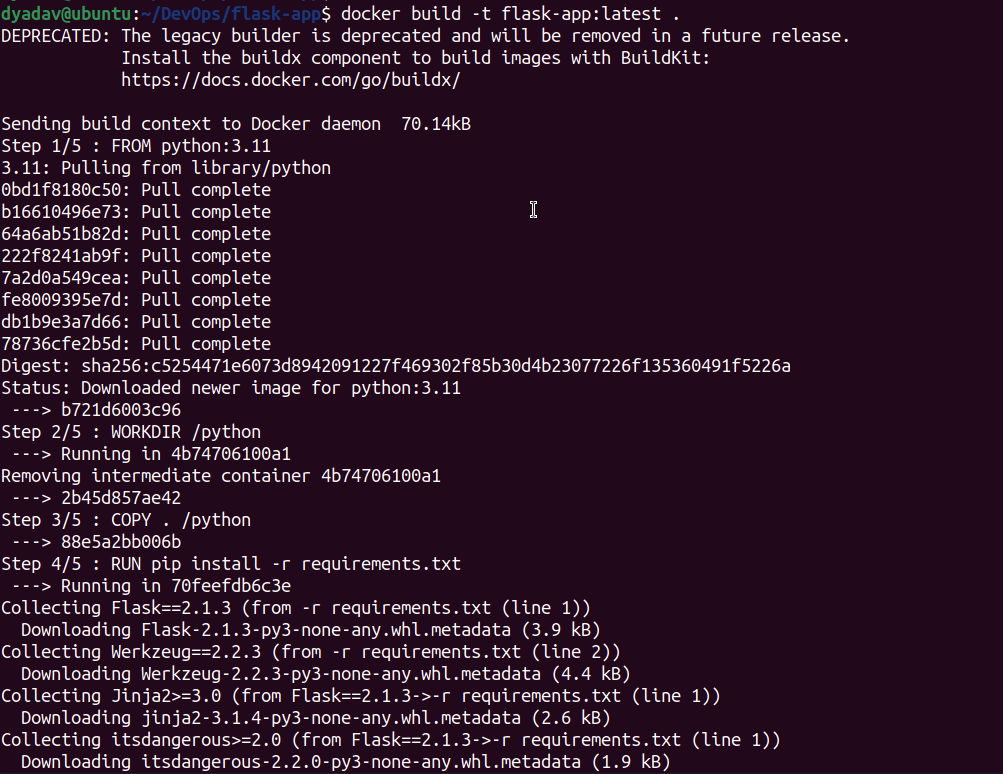
**git clone https://github.com/dheeruyadav54/flask-app.git**

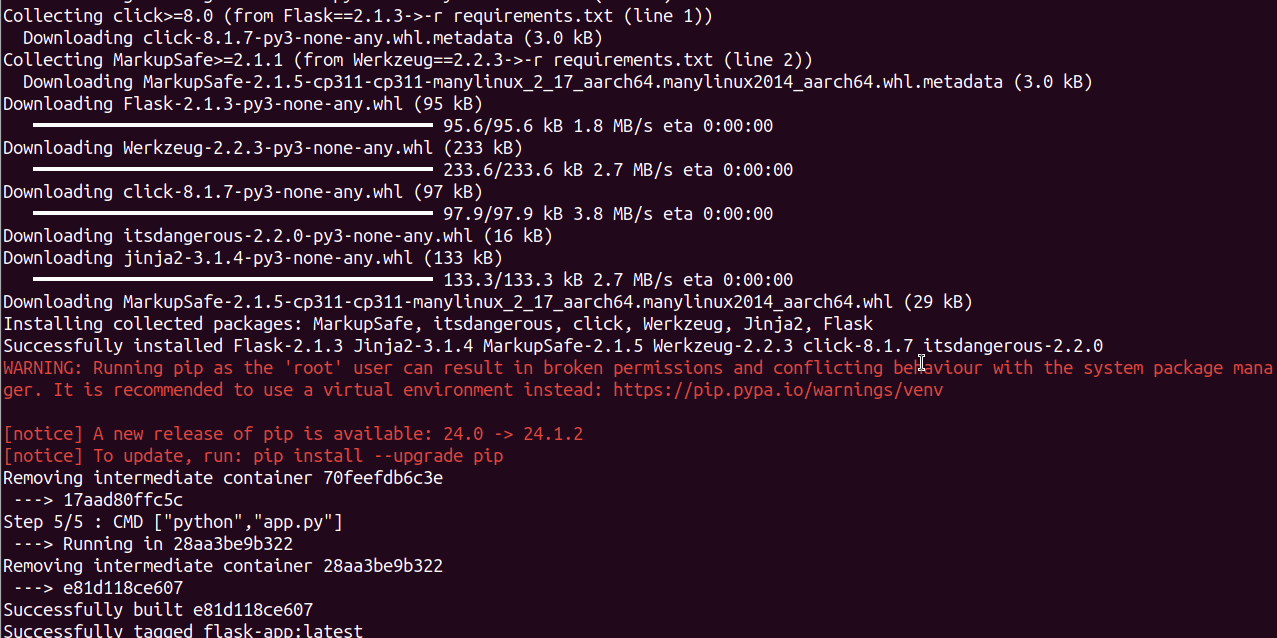
vim Dockerfile —> Create a Dockerfile with Capital D

# Created a Docker file per requirement



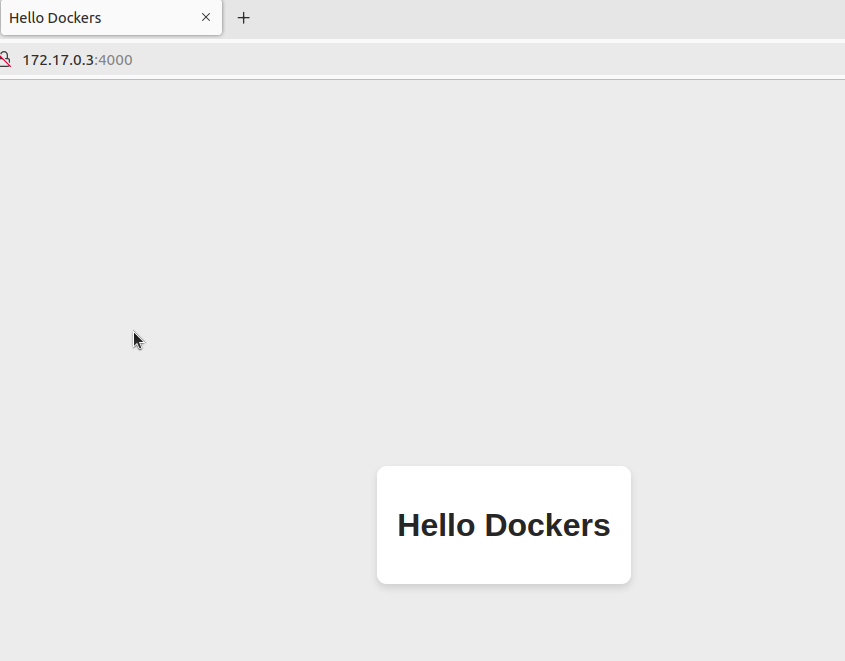
**$ docker build -t flask-app:latest .** :--> run this command to build the flash app image.



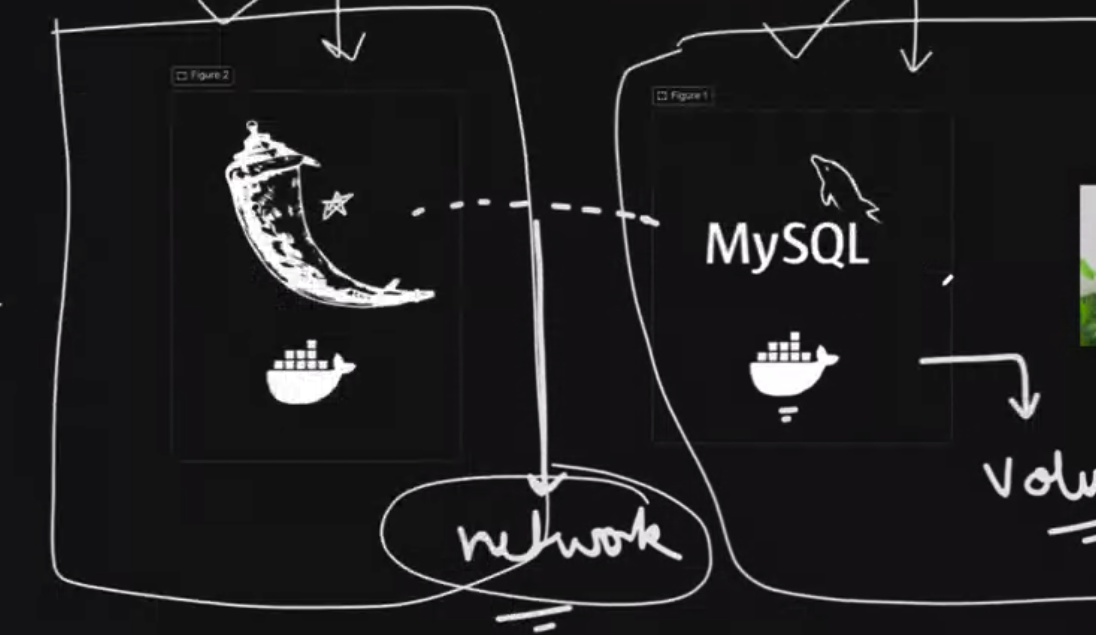


**$ docker run -d -p 4000:4000 flask-app:latest** : use this command to run the container.





### **two-tier Project**



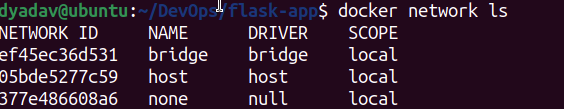
1. Clone github project on ubuntu machine.

**git clone** [**https://github.com/dheeruyadav54/two-tier-flask-app.git**](https://github.com/dheeruyadav54/two-tier-flask-app.git)

We are going to deploy a 2 tier application on docker, by default Container runs in isolation mode hence both the container’s can’t communicate with each other.

**Solution : Create a Network & assign both the containers into that Network so they both can communicate.**

**$ docker network ls :** to list all the present Network



**$ docker network create twotier-Network :** Created a new Network for this Project & by Default Network type would be **bridge**

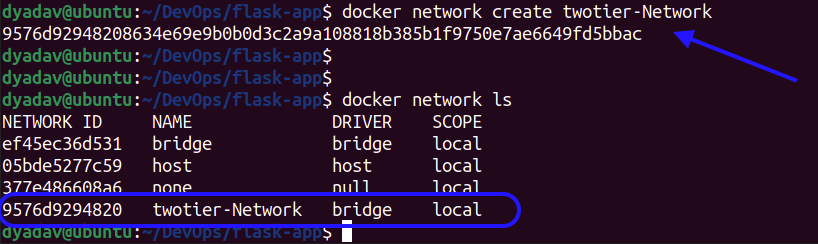
**Network Type:**

**===========**

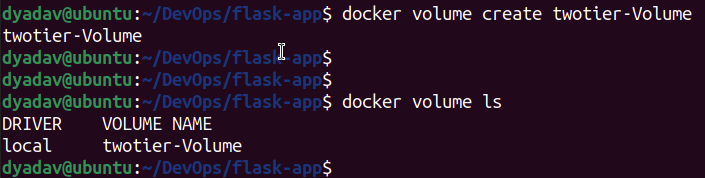
**bridge :** In this Network type we need to allow the traffic explicitly through variables like -p 81:80 , binding the host port 81 with container port 80.

**Host :** In this Network type we don't need to allow the traffic , allow everything.

**None :** Everything is blocked.

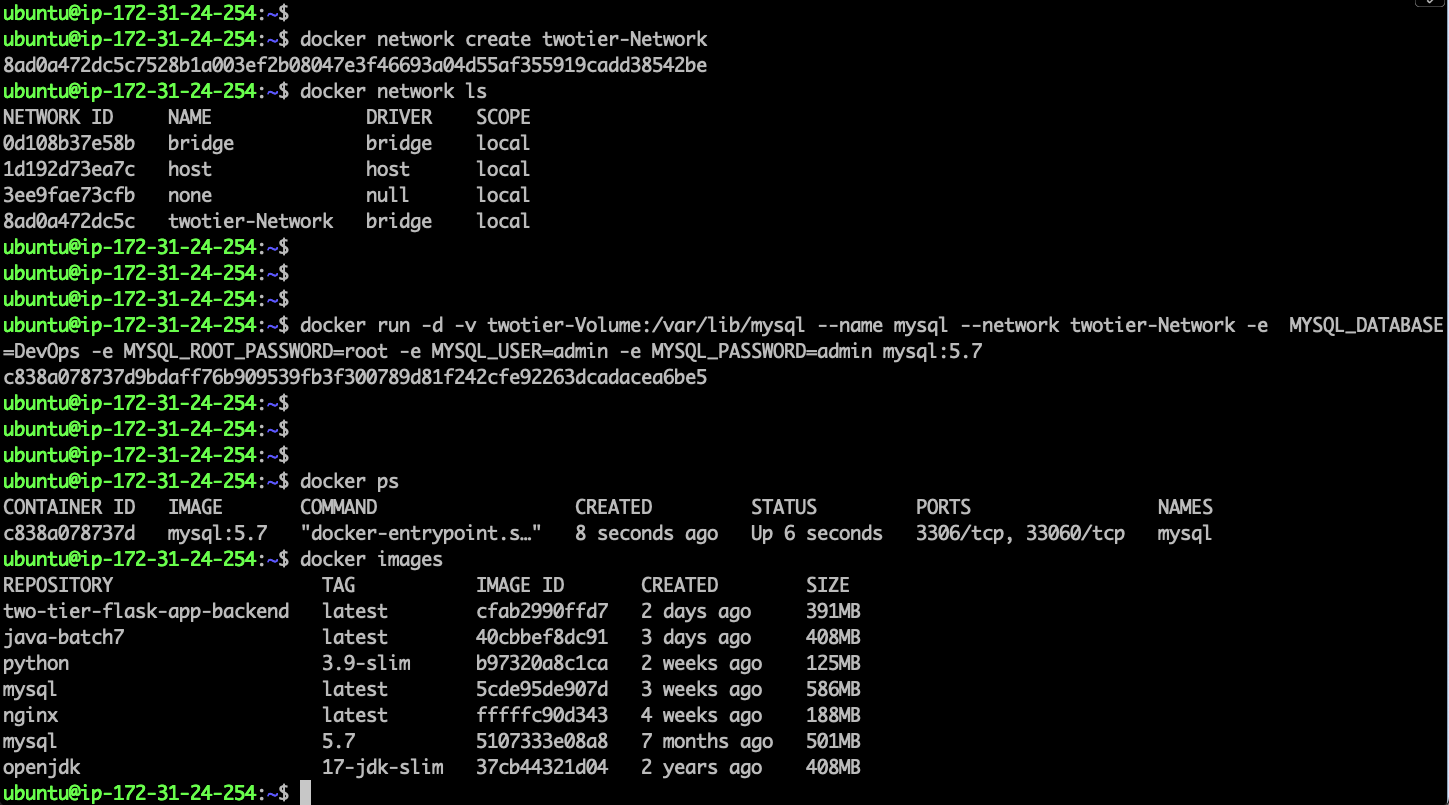


**$ docker volume create twotier-Volume** : Created a new Volume for this Project.



**$ docker run -d -v twotier-Volume:/var/lib/mysql --name mysql --network twotier-Network -e MYSQL\_DATABASE=DevOps -e MYSQL\_ROOT\_PASSWORD=root -e MYSQL\_USER=admin -e MYSQL\_PASSWORD=admin mysql:5.7 :** Run this command we have given couple of variables here -d, -v, –name , –network , -e etc

mysql database is ready



===> Run the below commands to check the Database.

**$ docker exec -it <<Container ID>> bash**

bash-4.2# mysql -u root -p

Enter password:

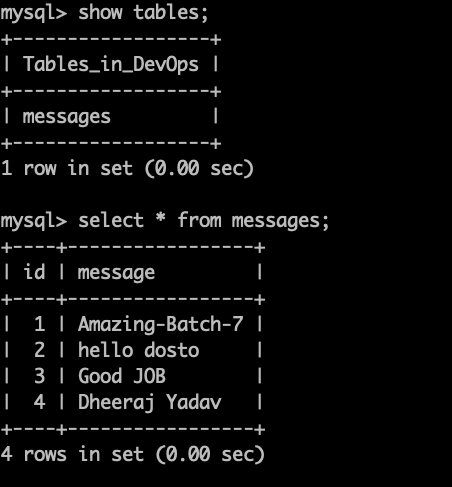
**mysql> show databases;**

**mysql> use DevOps;** : select the DevOps Database

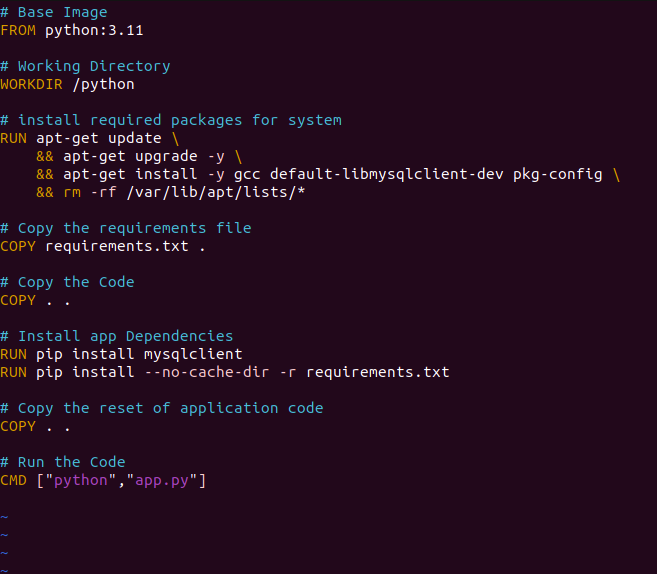
**mysql> show tables;** : Check the Tables

**mysql> select \* from messages;** : to check the Data

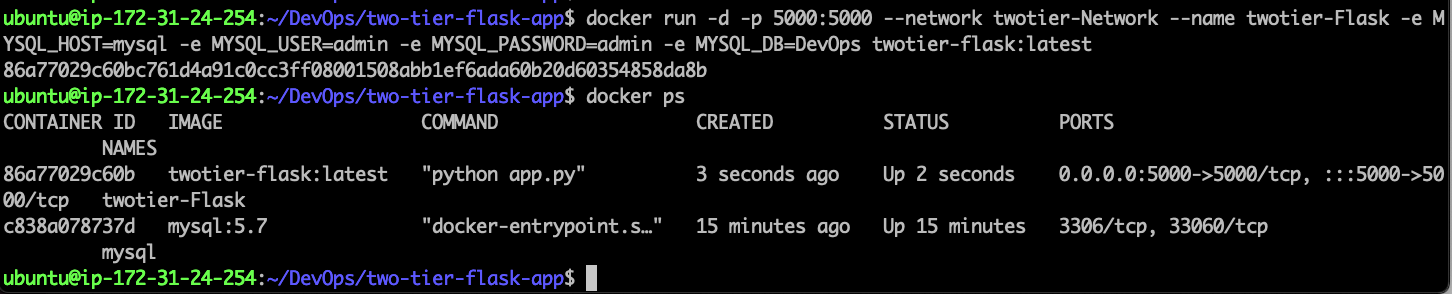




Dockerfile to build the Python flask-App

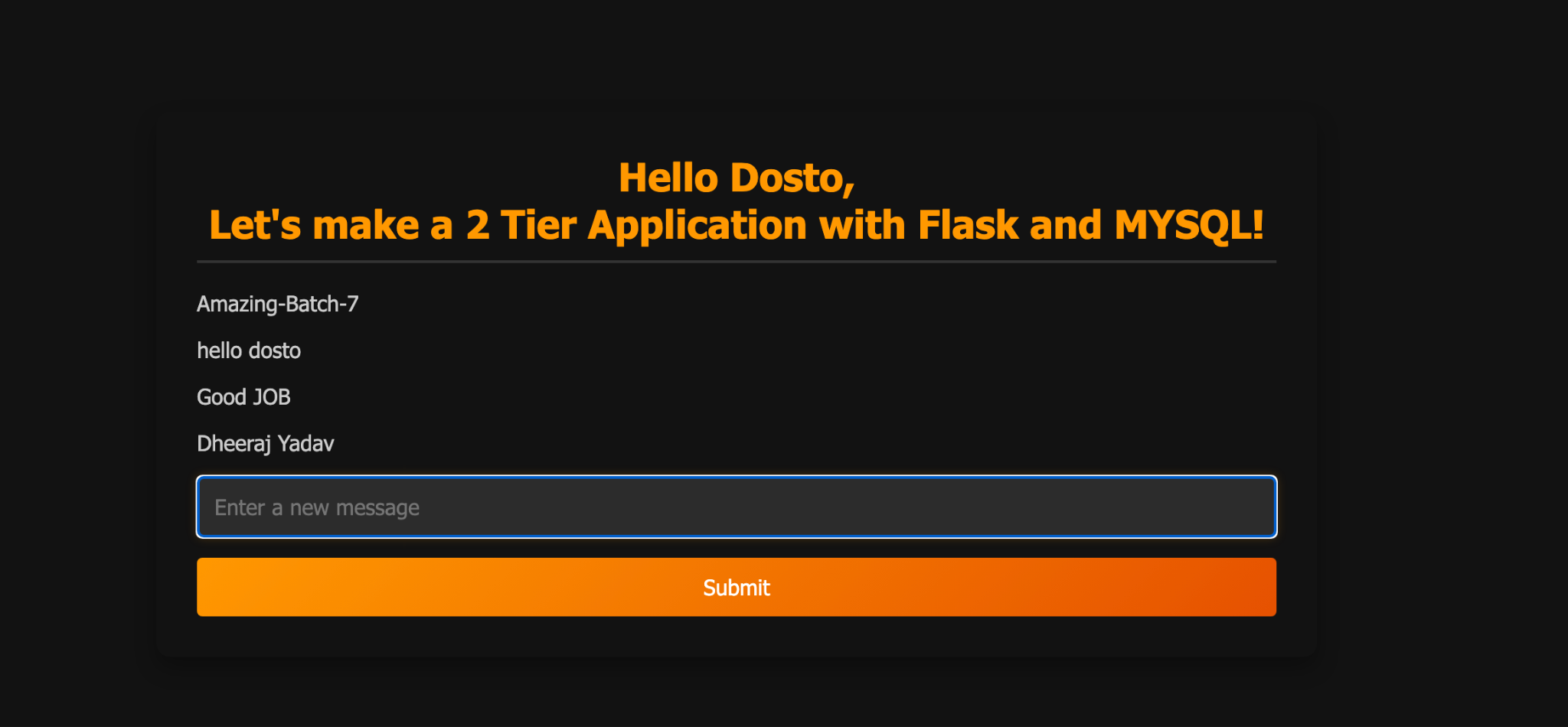


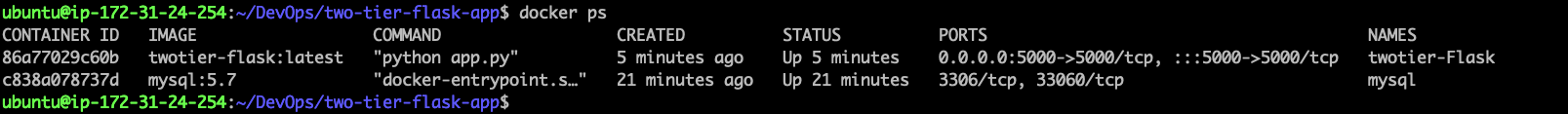
**$ docker run -d -p 5000:5000 --network twotier-Network --name twotier-Flask -e MYSQL\_HOST=mysql -e MYSQL\_USER=admin -e MYSQL\_PASSWORD=admin -e MYSQL\_DB=DevOps twotier-flask:latest** : Used this command to run the flash app with mysql Database



Flash app is running on port 5000

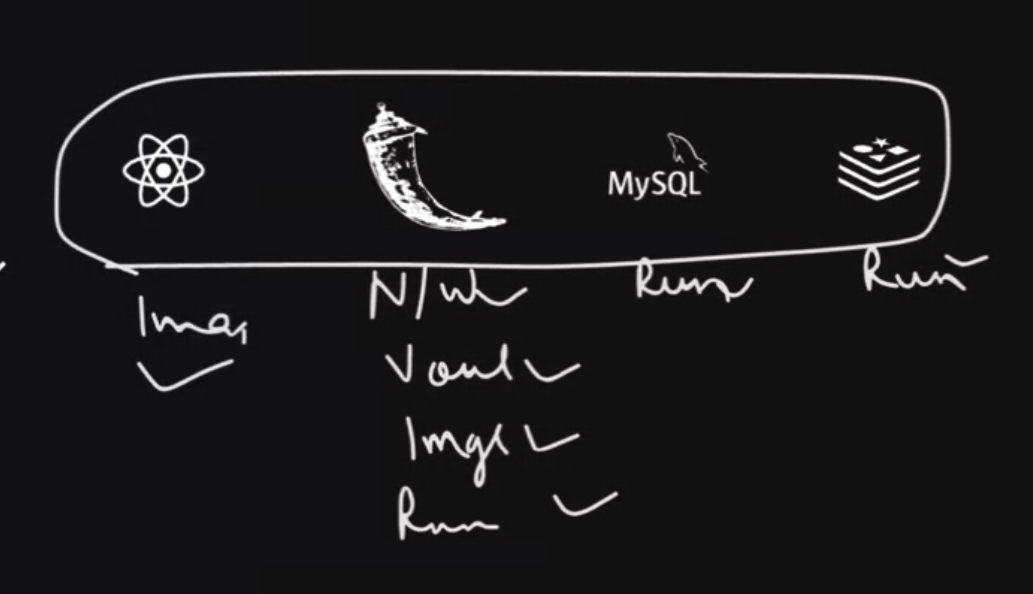
**http://3.79.149.35:5000**





### **Docker Compose**

It is used to run a single YAML file to build the container.



====> Create a docker-compose.yml file.

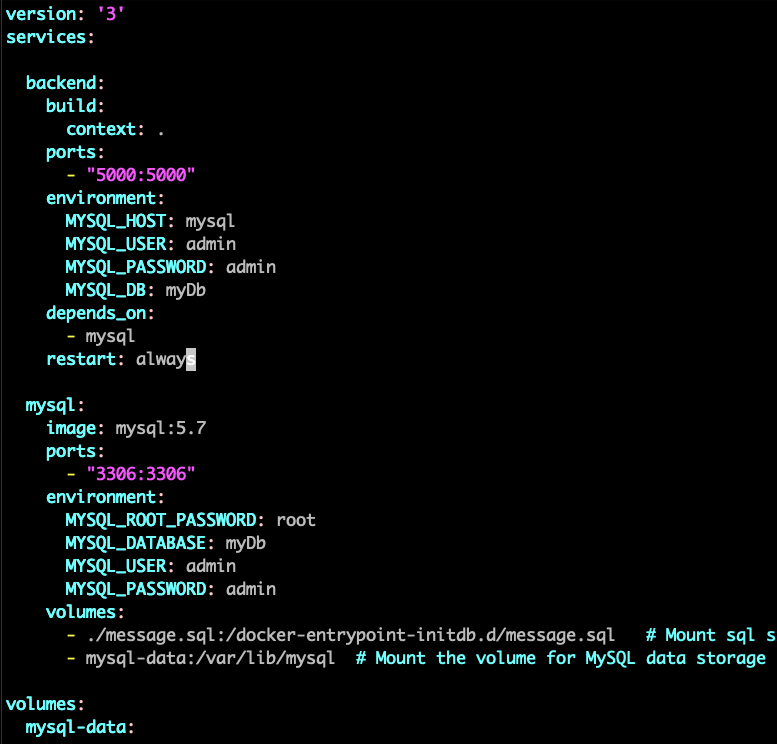
**$ docker compose up -d :** To Create the multiple docker Container with a single file.

**$ docker compose down :** to remove the created docker container

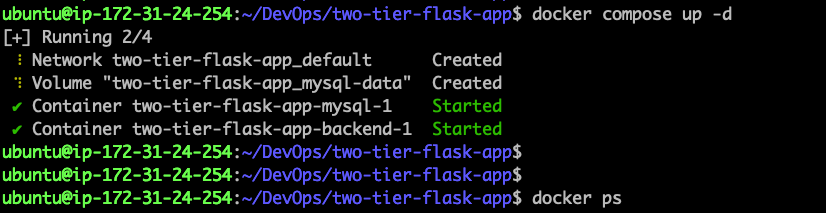
Services : Under Service we define our container.

build:

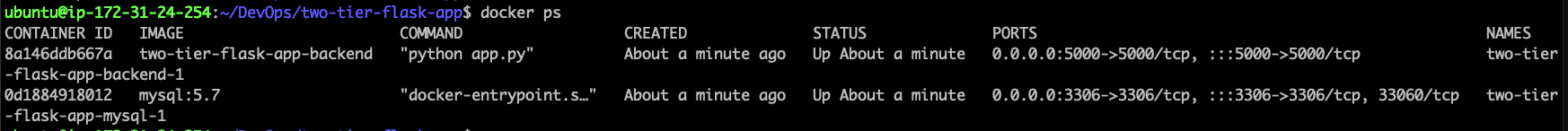
Context . → Dot Represent that Dockerfile is here in the current directory.



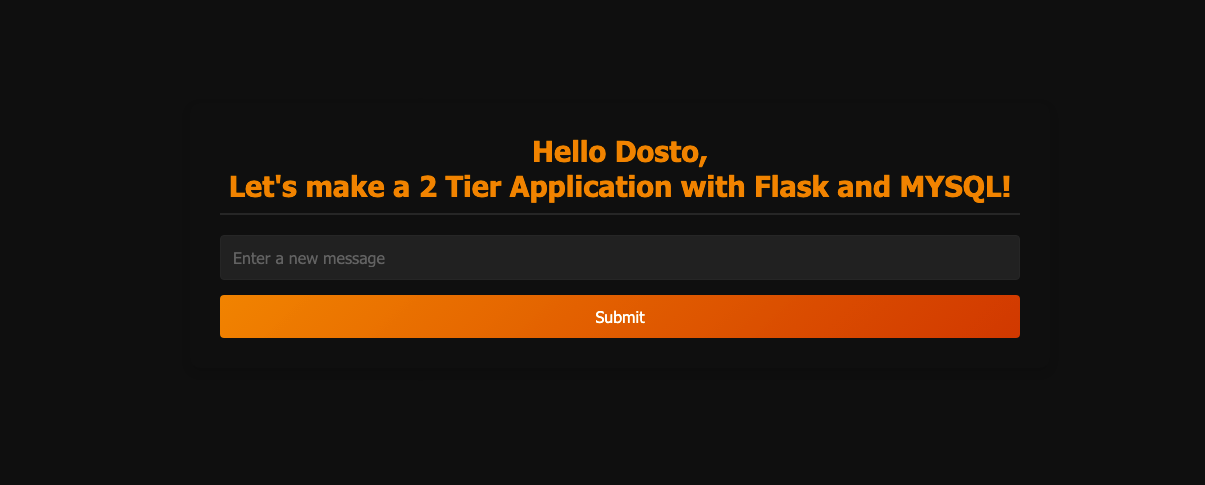
**$ docker compose up -d :** To Run the Containers.



**$ docker ps :** To verify the running containers.



Tier-2 Application is running on port 5000.



### 

### 

### 

### 

### 

### 

### **Multi-Stage docker build**

“ Will ADD SOON”

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **Docker scout**

Docker scout is to check the vulnerability from the images.

Step1: Create a Directory.

**$ mkdir -p $HOME/.docker/scout**

Step2: To install the latest version of the plugin

**$ curl -fsSL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh -o install-scout.sh**

**sh install-scout.sh**

Step3:

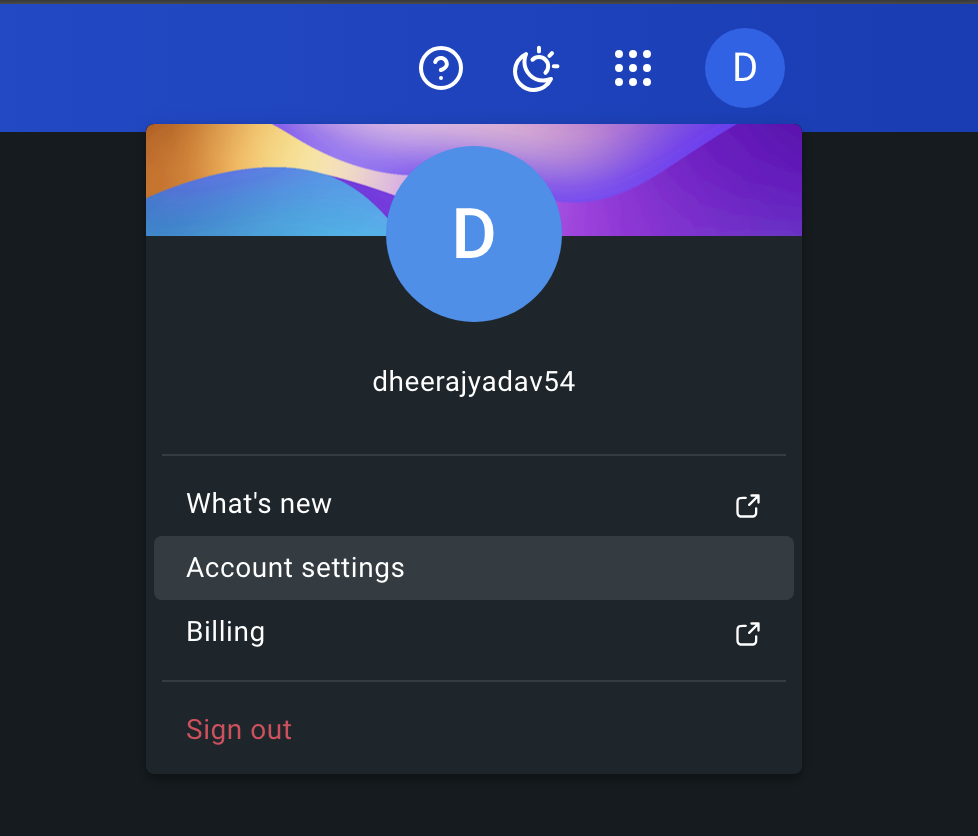
$ docker login

Username : dheerajyadav54

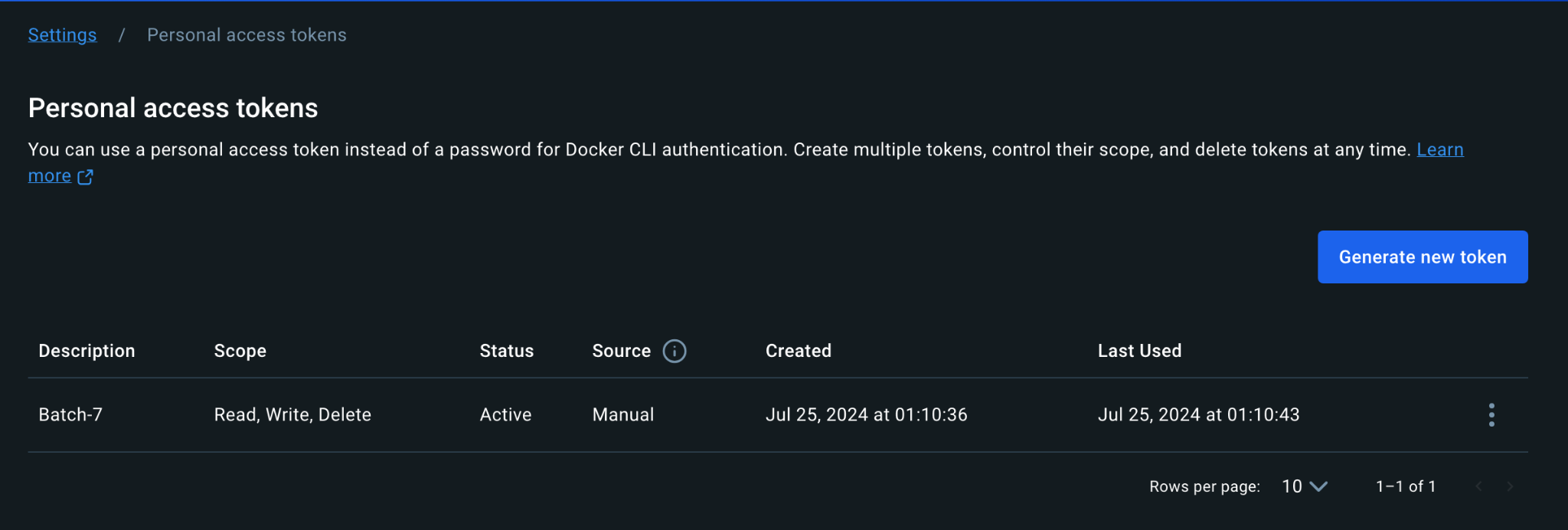
Password: Personalized Access Token

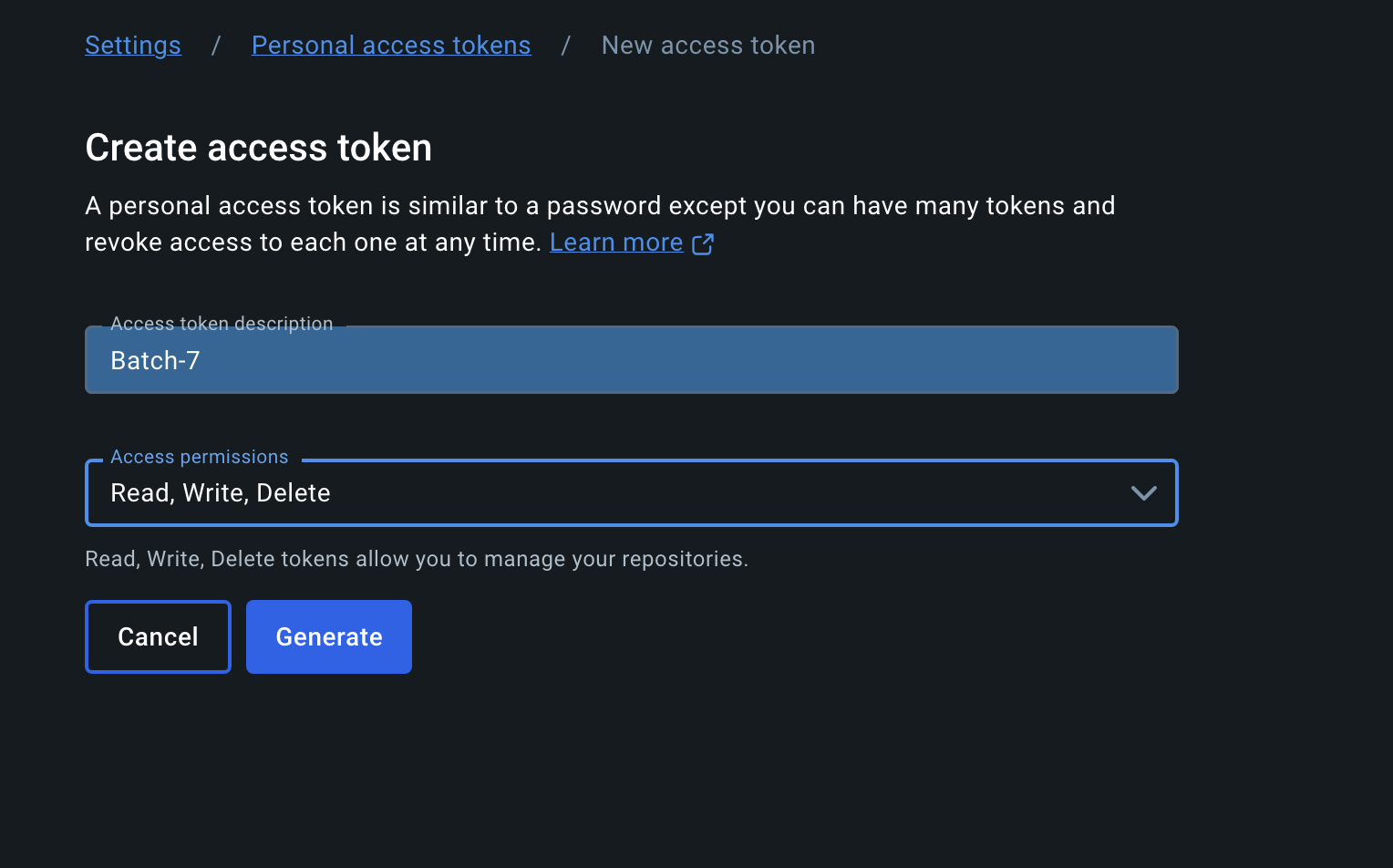
Which can be collected per below method:

1. **Docker Hub Account —-> Account Setting**

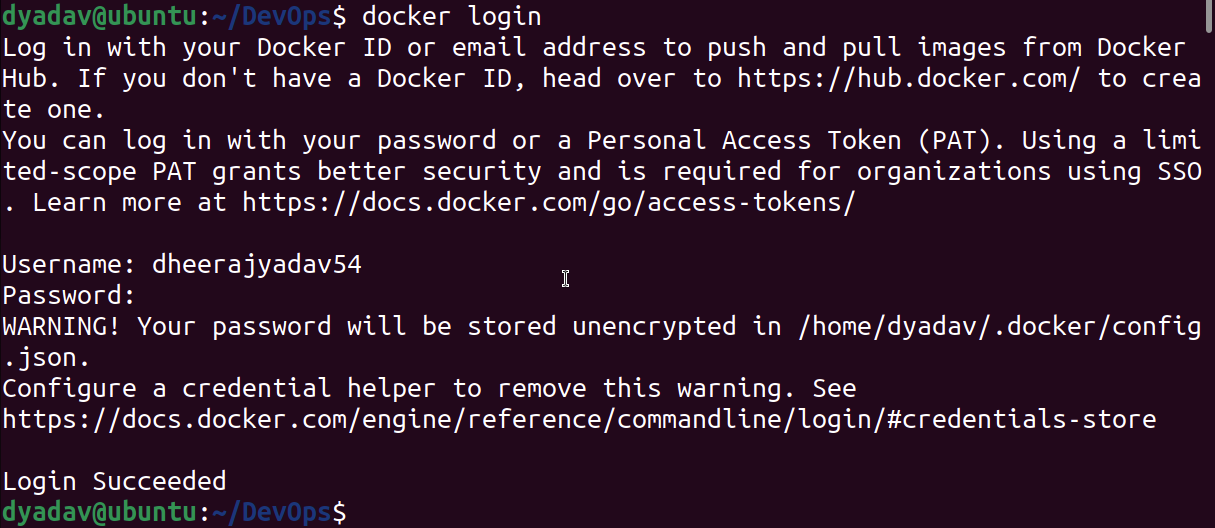


1. Personal access tokens → Generate new token



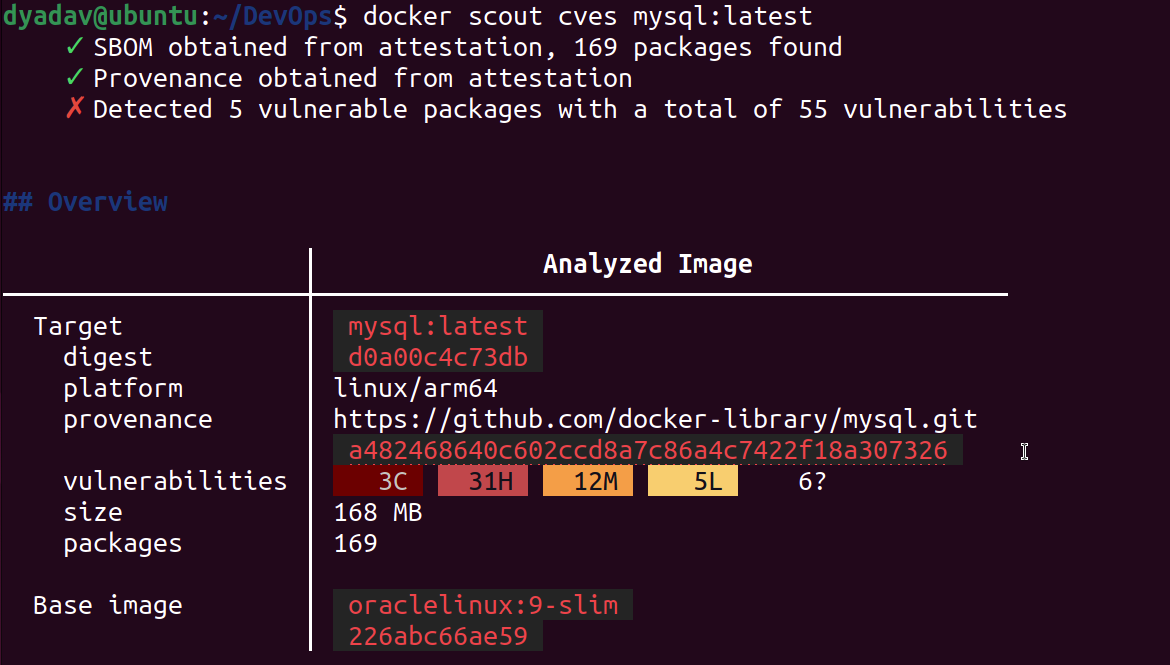


Once it’s generated, copy the token & paste into the Password Section.

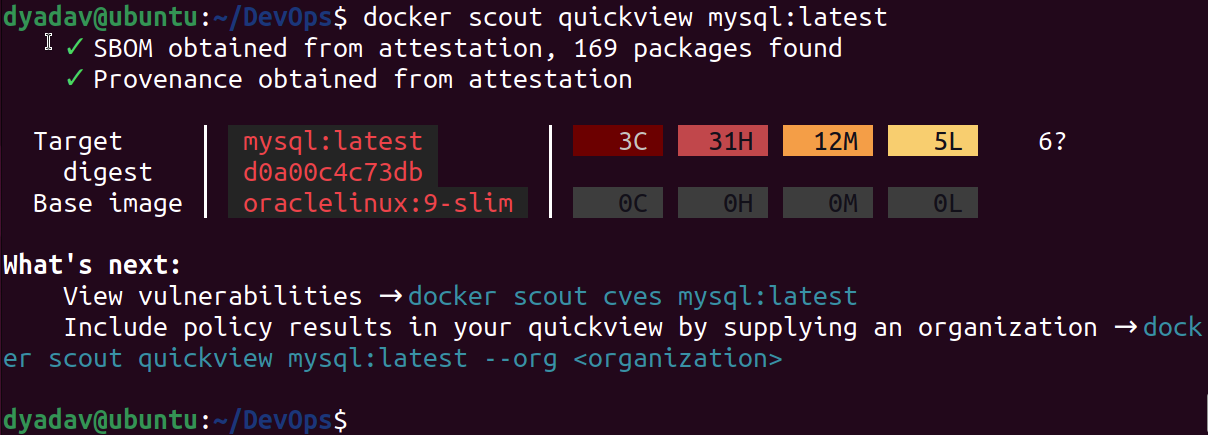


Now run the docker scout command to scan the images.

**$ docker scout cves mysql:latest :** to scan the mysql:latest image



**$ docker scout quickview mysql:latest :** you can quick view of the image

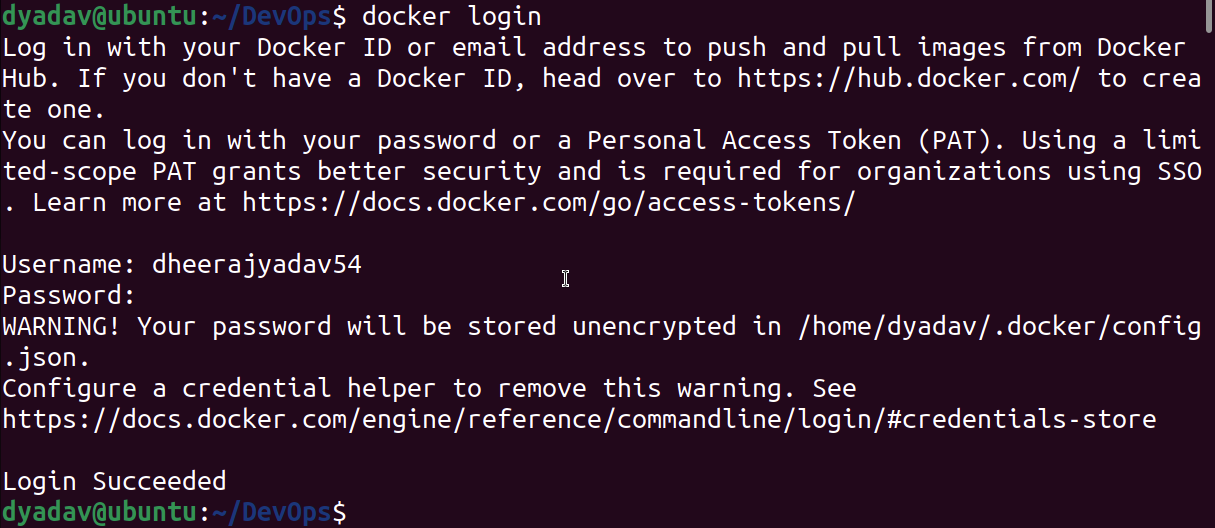


### **Docker Hub Login:**

$ docker login

Username : dheerajyadav54

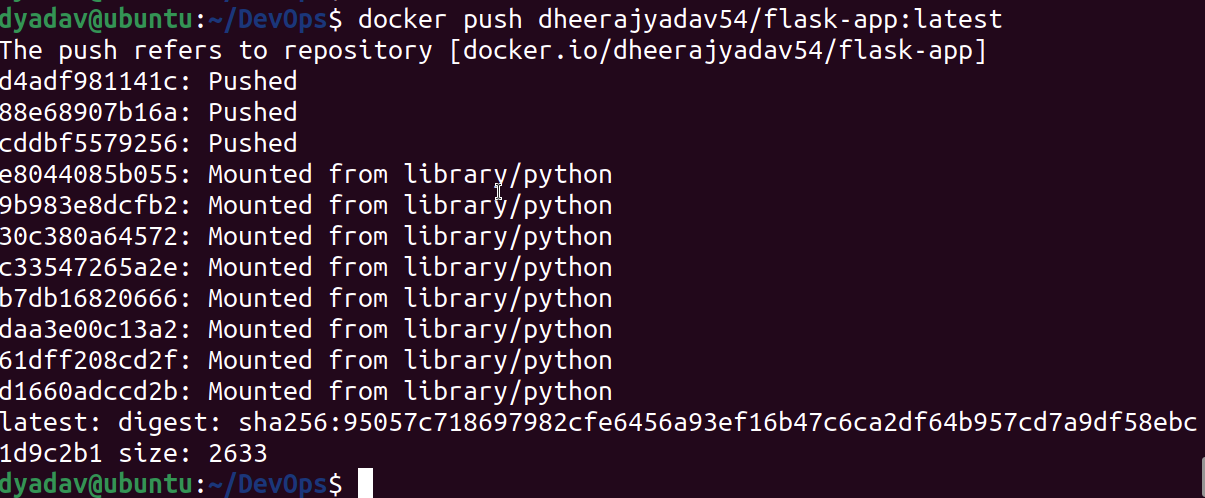
Password: Personalized Access Token



**$ docker image tag flask-app:latest dheerajyadav54/flask-app:latest :** create a image tag with docker hub username i.e: dheerajyadav54



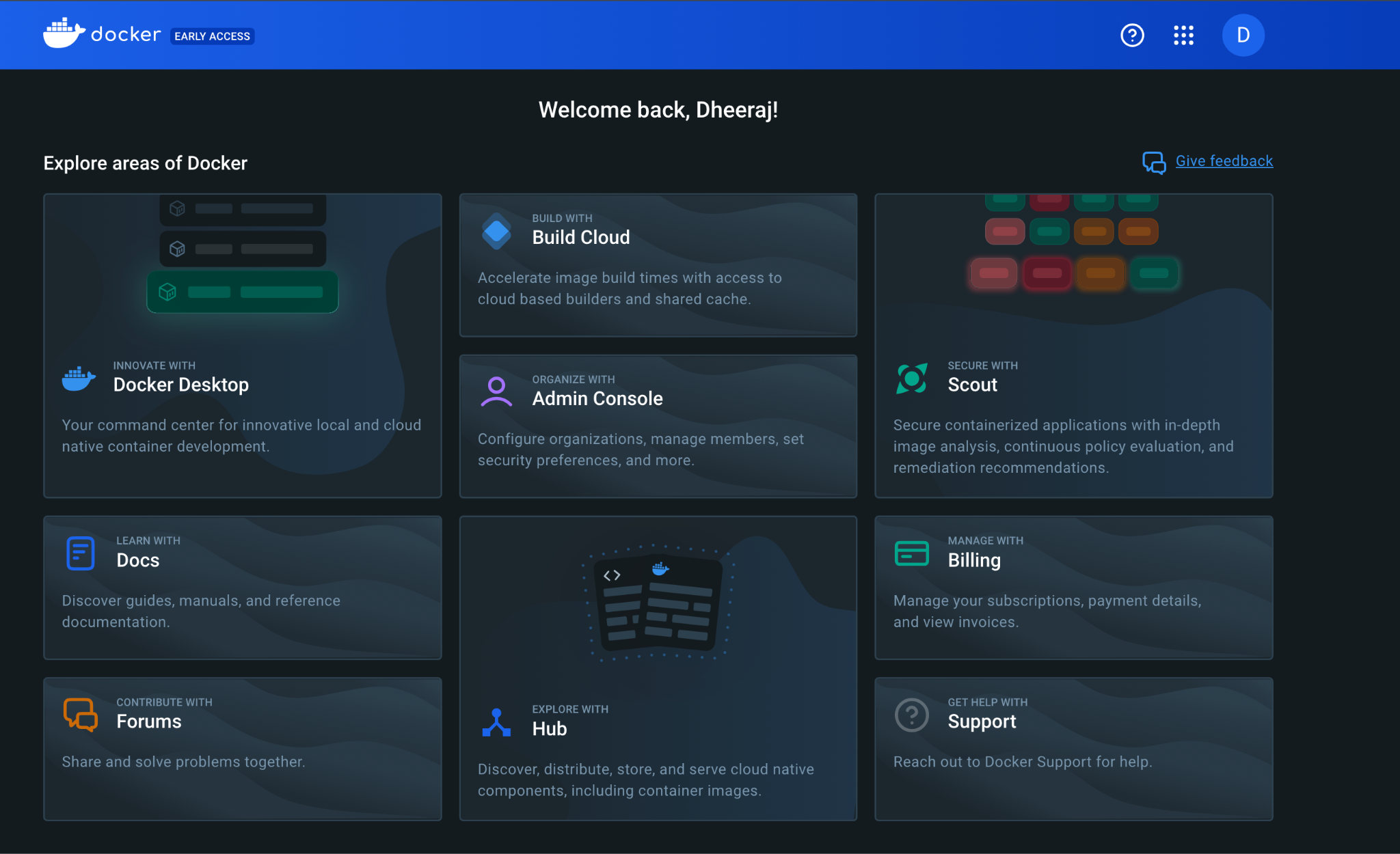
$ docker push dheerajyadav54/flask-app:latest : Use Docker Push Command to push the image from local to Docker Hub.



To Check on docker hub :

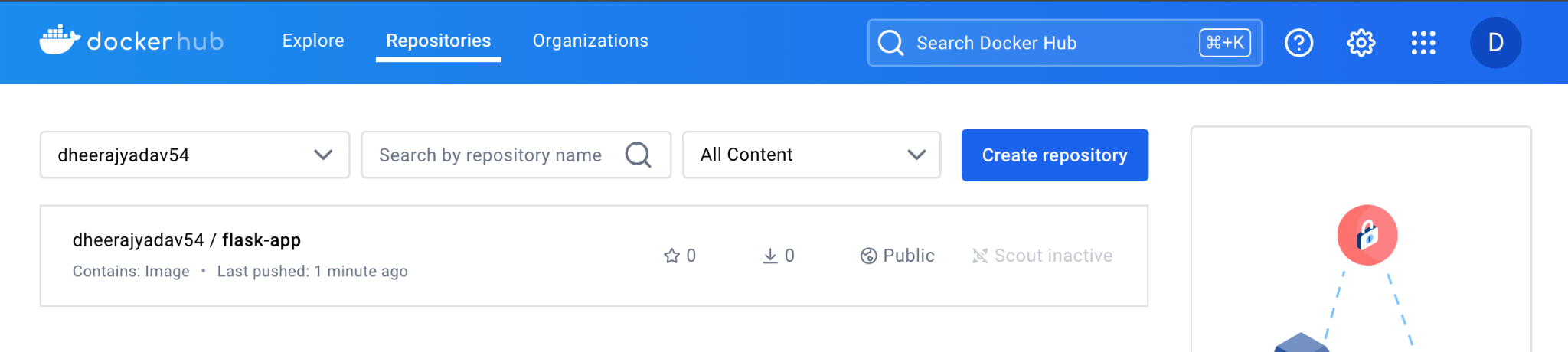
Go to URL : <https://app.docker.com/>

**Click on Hub:**



It will redirect to URL <https://hub.docker.com/>

Here we can see the pushed Docker image from Local to Docker HUB.



### **Bonus Commands**