Day -1

Download and install docker desktop

Sign in to docker hub

Run **Computer Management** as an **administrator** and navigate to **Local Users and Groups** > **Groups** > **docker-users**. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

Connect to VPN and add user HOMEOFFICE/a0p09kj then docker is working

Open cmd and

Docker login arunvig 123456789

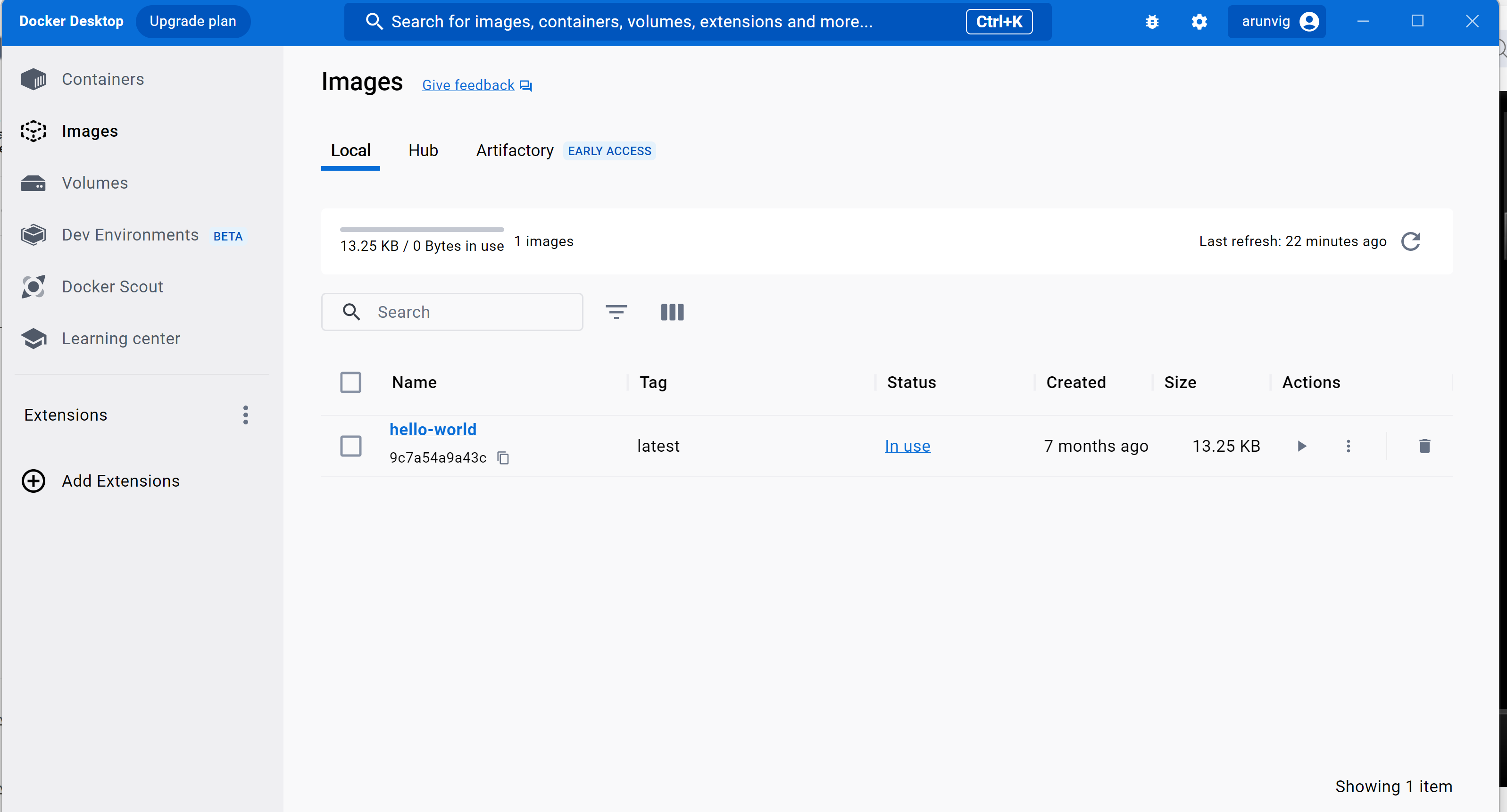
Docker pull hello-world

Docker run hello-world

After running it gives exact result like in hocker hub hello world.

A screenshot of a computer program

Description automatically generated



Docker pull openjdk

Docker rmi opnjdk --- this is to remove docker images

Docker image

To remove image:

If the image is running in container and doesn’t allow to remove

Do

Docker ps

Get all running process using docker ps -a

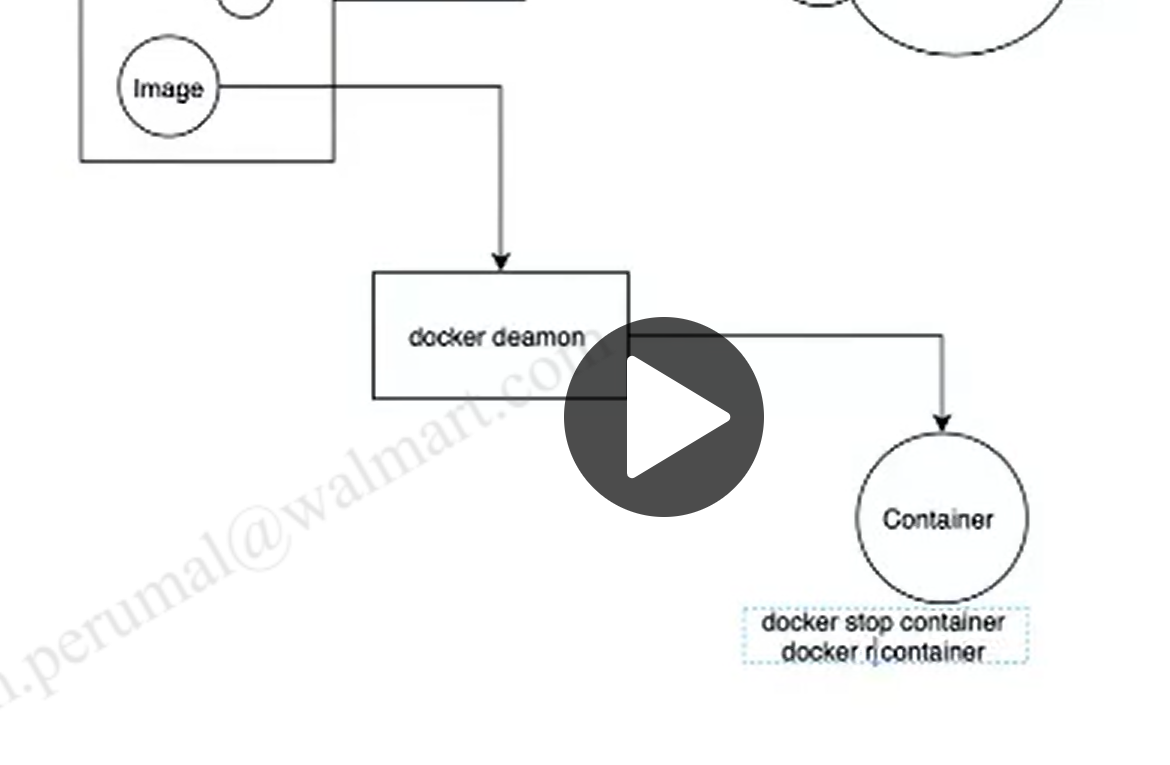
Get the container id and docker stop container\_id

Docker rm container\_id – to stop from running

Docker rmi repository\_name or container\_id

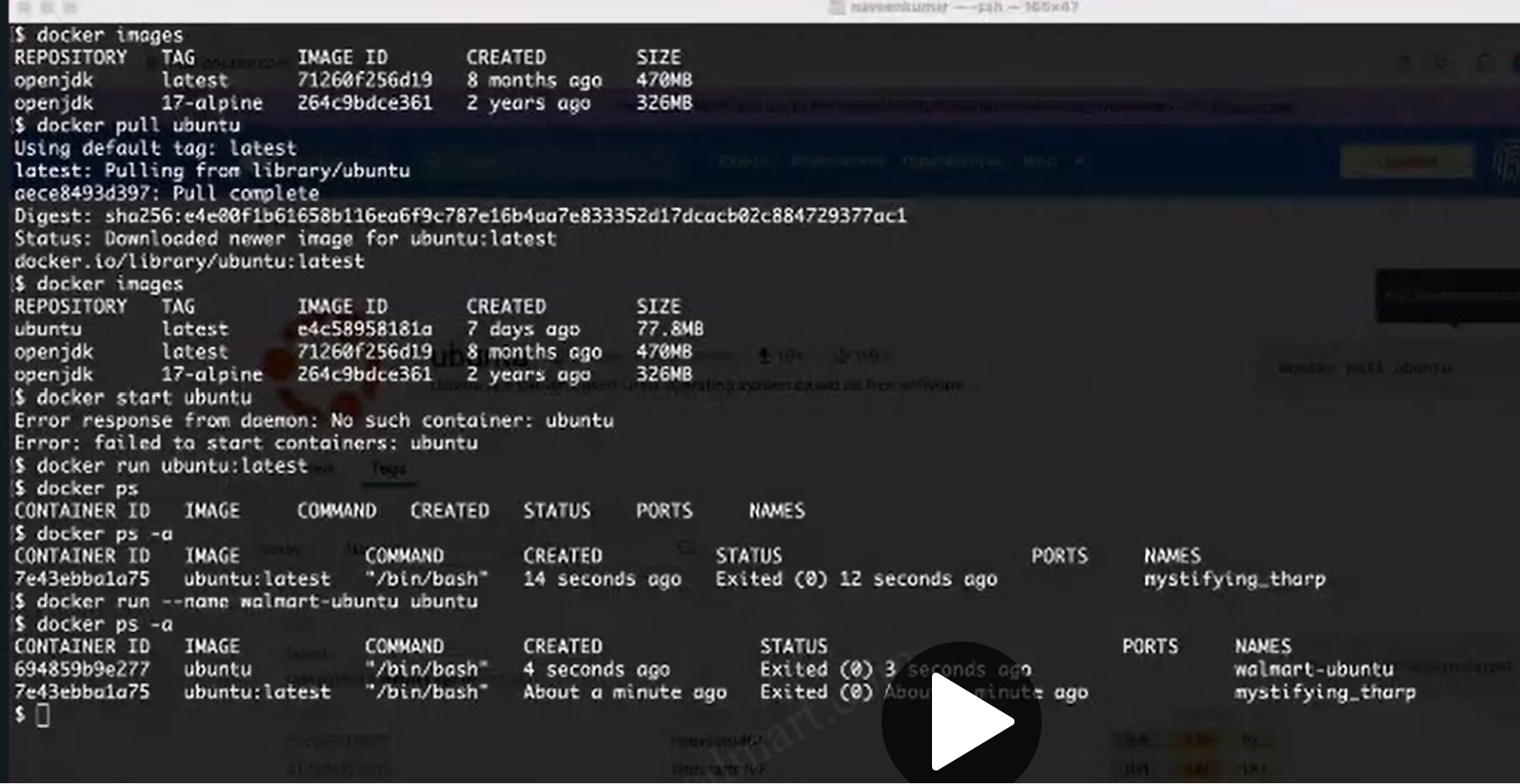
If you list images then the removed images wont be there

Below are to run images in designated name:



A diagram of a docker

Description automatically generated



Docker pull ubuntu – to get ubuntu from docker hub

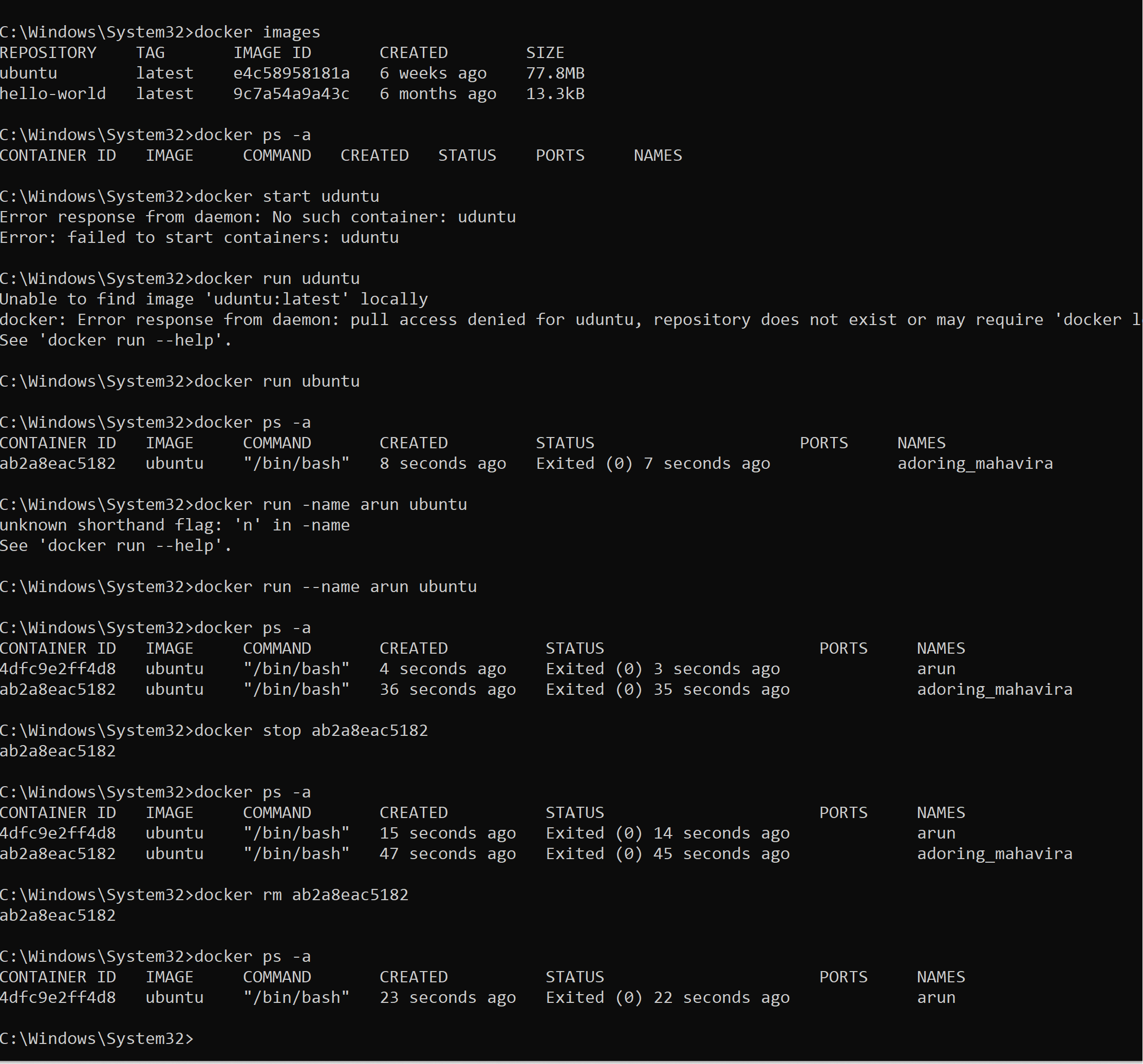
Docker images – to see what all images we have

Docker start ubuntu – to start the downloaded image (not working)

Docker run ubuntu – to start the image

Docker run –name arun ubuntu --- to run in designated name

Docker ps -a --- to see the running images



Now ubuntu is pulled and running

I want to access to ubuntu terminal from windows cmd interactively

Docker run –name prem -it ubuntu ---- this keeps ubuntu executes in prem name and activates interactive session on bash command prompt

A black screen with white text

Description automatically generated

Likewise we can run the openjdk and in cmd we can run sysout hello world

**For Java Project:**

Javac sample.java

Java sample

**Generate jar from Maven project:**

Mvn clean – in intellij

Mvn install – to generate a jar

Java -jar generated.jar

[Dockerfile reference | Docker Docs](https://docs.docker.com/engine/reference/builder/)

Docker file reference in create dockerised spring boot

Docker build -f Dockerfile -t <any\_name> . – this generates a docker image

Docker run <any\_name> -- executes the image

Run the jar file from cmd line

Java -jar file.jar – once the spring boot start running we can access it from postman like what we do in intellij

For that in pom.xml the <build> tag is important otherwise java file doesn’t know where to start the execution

Java -jar file.jar – throws error if pom.xml doesn’t have the class name to start

Also

We can achieve the same without build by

Java -cp file.jar com.walmart.app – the class filename which is in build section

**TO execute ubuntu and bash command**

Docker pull ubuntu

Docker run –name prem -it ubuntu – starts ubuntu with name prem and runs bash

**Dockefile in maven spring boot**

Place Dockerfile in src, target line

Docker build -f Dockerfile -t walmart/maven-works .

Docker build -f Dockerfile -t <image\_name> .

Docker run –name prem <image\_name>

In above scenario the java code is turned to jar file and creating a image on it

Running the image as per java code it runs recursively while the container is running we can get into interactive mode that is below command on another terminal

Docker exec -it running\_container /bin/bash

Inside the bash we can see the deployment folder where the generated jar file will be moved and made active

Run the docker command to convert image of the spring boot from terminal inside the intellij

**How to run and access maven project as docker image**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with text

Description automatically generated

A screen shot of a computer

Description automatically generated

1. Once after the spring boot code is completed
2. Open terminal
   1. Ls
   2. Mvn clean install – this will generates the jar file
   3. Create Dockerfile with above steps in screenshot
   4. Now, create an image with the jar file
   5. Docker build -f Dockerfile -t login .
   6. Docker images – list the generated image files
   7. Now the login image will be available
   8. We need to run that image so that we can able to access the image running in the container
   9. Also if the spring boot is running inside the container the port 8080 is assigned to container only and not to host machine so follow below command with -p which binds 8080 from container to 8080 in host
   10. docker run -p 8080:8080 login -- now the app will run and can be accessed like localhost:8080

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

Once the image is created and tried running from postman

We can push it to docker hub

Docker images

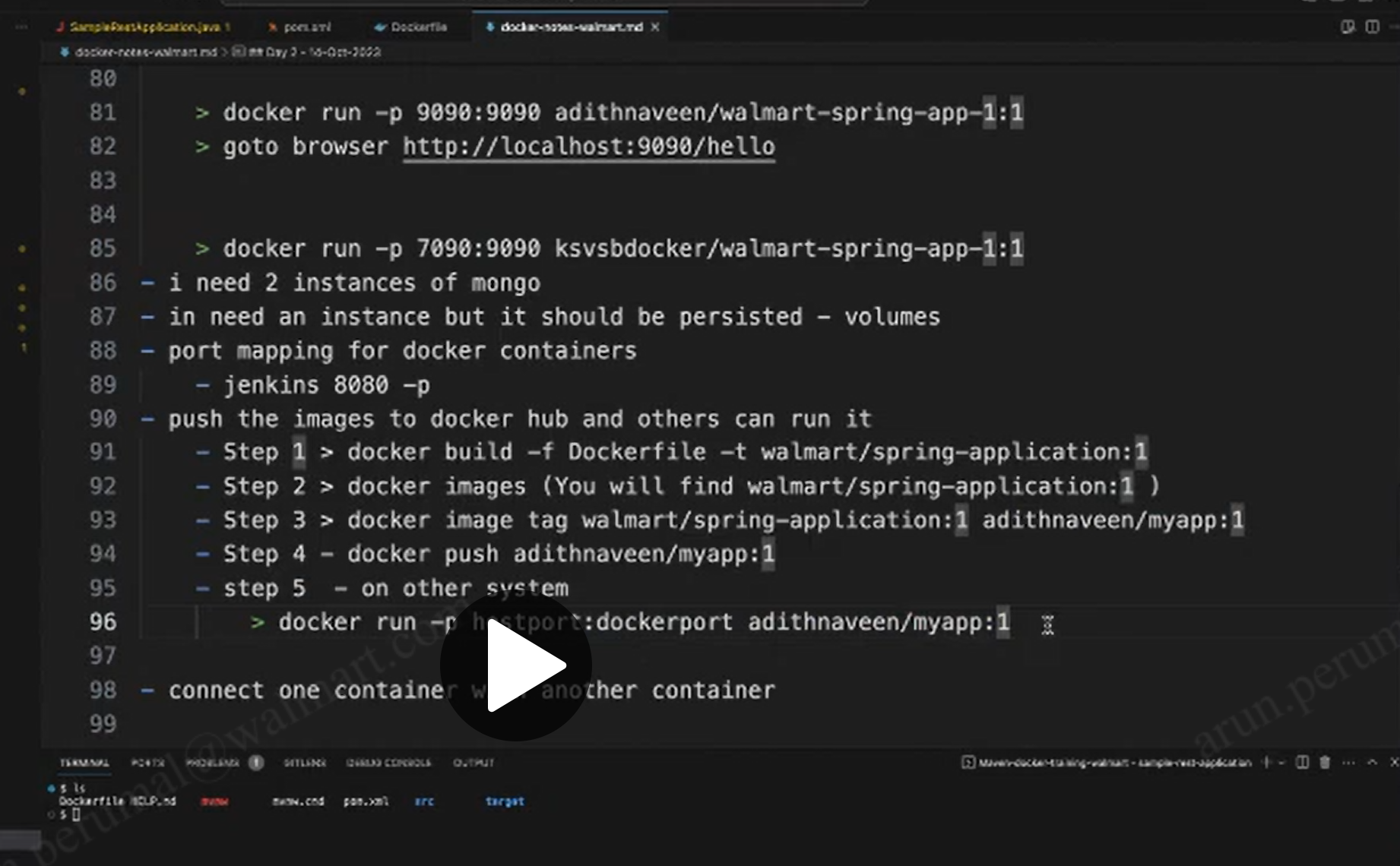
docker tag login:latest arunvig/login:1

<latest is common when images is created and we moving it to another image to upload>

Docker push arunvig/loginmod:1 – pushes the image to hub for public use

Then we can run the image like

Docker run -p 9090:9090 arunvig/loginmod



A screenshot of a computer

Description automatically generated

Arunvig/loginmod -- working

echo $Env:MONGODB\_URI

ubuntu

**docker compose**

docker-compose -v

docker run -p 8081:8080 transactionservice

docker run -p 8080:8080 userservice

docker run -e MONGODB\_USERNAME=arunbalaji5005 -e MONGODB\_PASSWORD=UCR8ee9aISJY9Fom -p 8080:8080 userservice

arunbalaji5005

UCR8ee9aISJY9Fom

**Steps followed for the project**

Once the project is working fine in local with postman

Do mvn clean install – this will generate the jar file

Create dockerfile in src, trg directory. In my case we need to create 2 dockerfile as it runs for 2 services

#passing arg to dockerfile and assigns to env variable that app.prop will take it

docker build --build-arg USERNAME=arunbalaji5005 --build-arg PASSWORD=UCR8ee9aISJY9Fom -f Dockerfile\_UserService -t userservice .

docker build --build-arg USERNAME=arunbalaji5005 --build-arg PASSWORD=UCR8ee9aISJY9Fom -f Dockerfile\_TransactionService -t transactionservice .

# in run command assigning name to the running process and port 8080 in docker container is assigned to 8080 in local so that postman sending 8080 local msg to container

docker run --name User\_Service -p 8080:8080 userservice

docker run --name Transaction\_Service -p 8081:8080 transactionservice

#once the service is running use below command to check the env variable of docker container

docker exec <container\_name> env

# access the service from postman

# create one more tag for the image and upload then in hub.docker.com so that others can pull and run it so that postman msg will get response

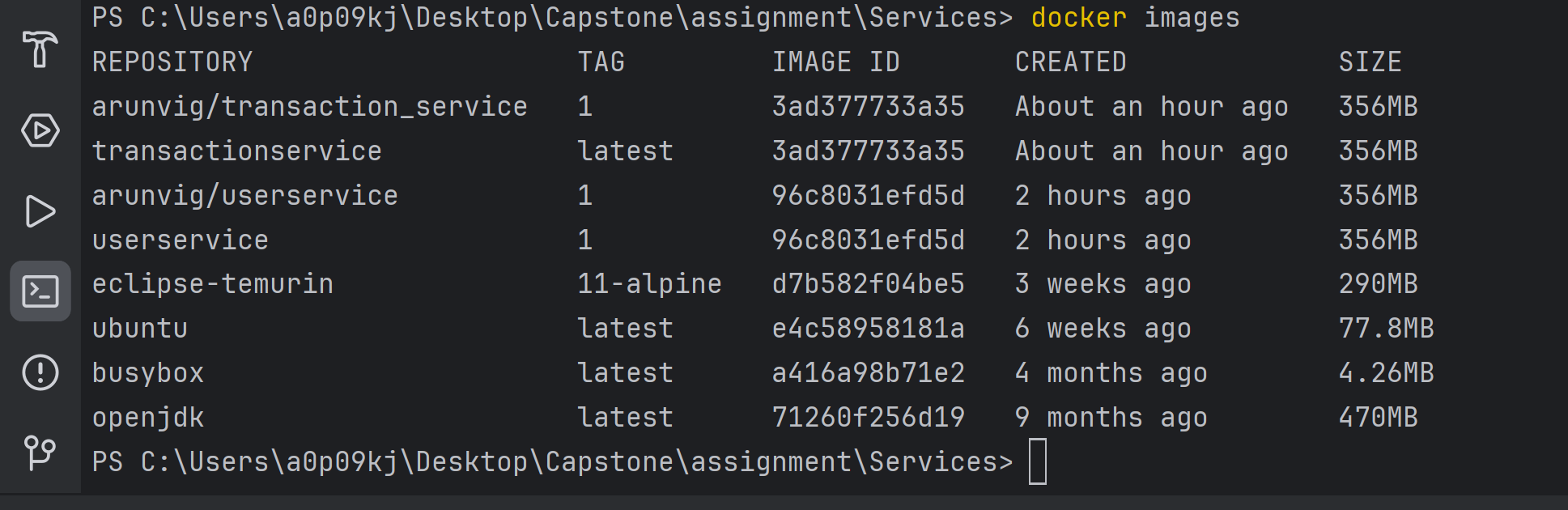
docker tag userservice:latest arunvig/userservice:1

docker tag transactionservice:latest arunvig/transaction\_service:1

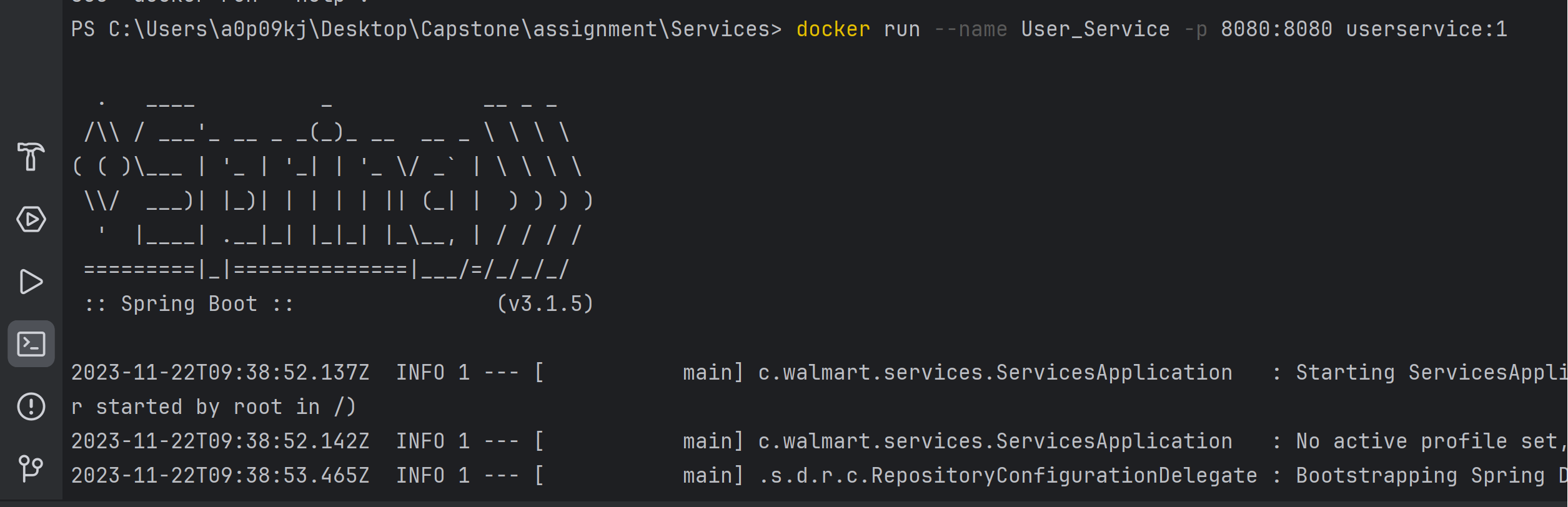
# push them into hub

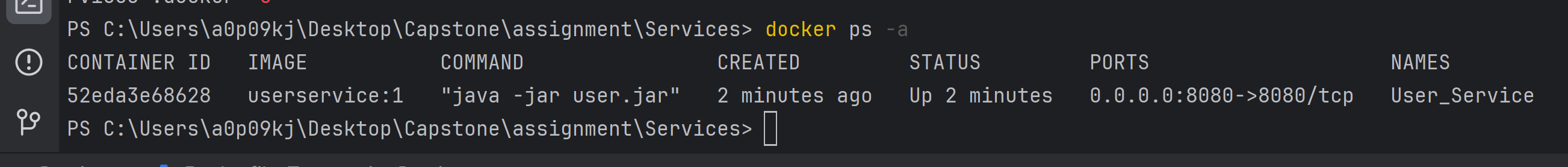
docker push arunvig/user\_service:1

docker push arunvig/transaction\_service:1



#userservice:1 meaning tag name is not given by default its value is latest and we don’t have latest tagged images so we are using tag 1





A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

[arunbalajii/service\_assignment: Final Backend and Docker assignment (github.com)](https://github.com/arunbalajii/service_assignment)

