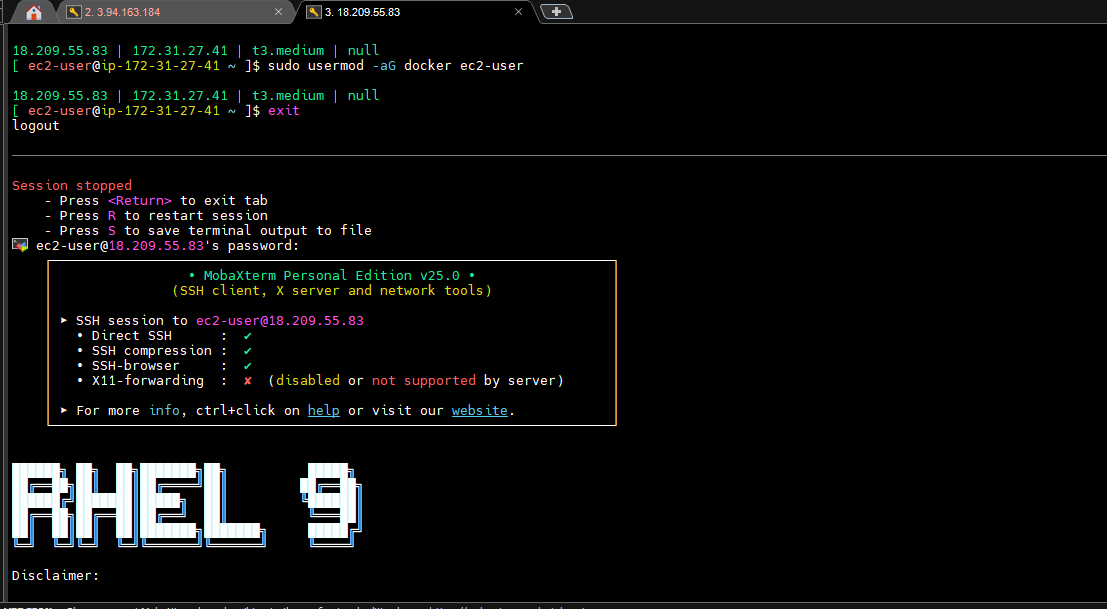
SESSION - 55

Docker compose is command line tool to manage multi container applications.you can create volumes and you can create networks and you can start and stop all the containers at a time.

--> connect to the server.

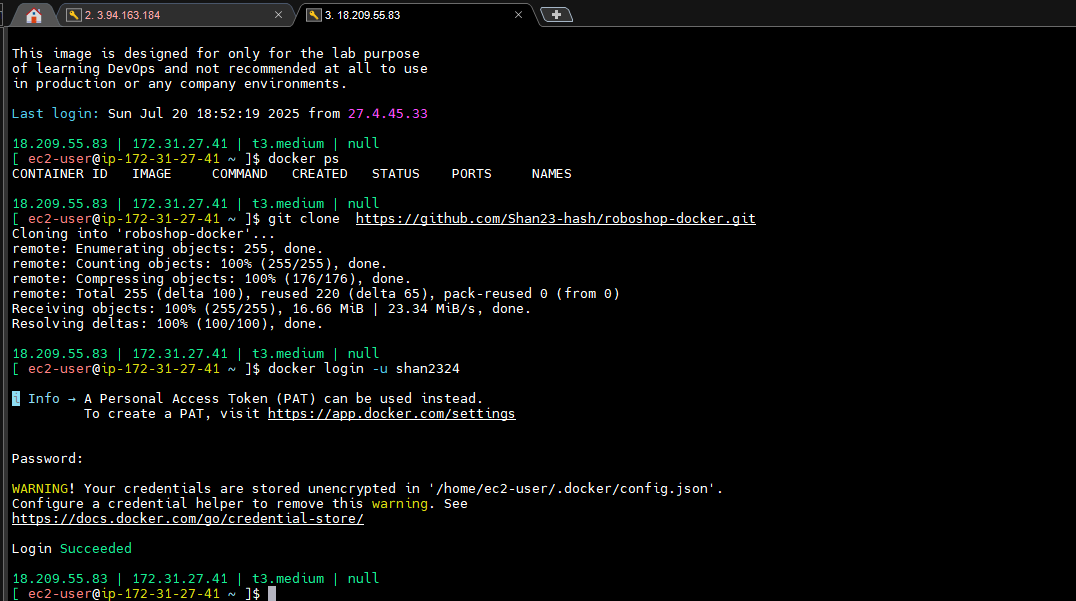
--> today we will see optimization.

--> **sudo usermod -aG docker ec2-user**



--> **git clone <https://github.com/Shan23-hash/roboshop-docker.git>**

**--> docker login -u shan2324**

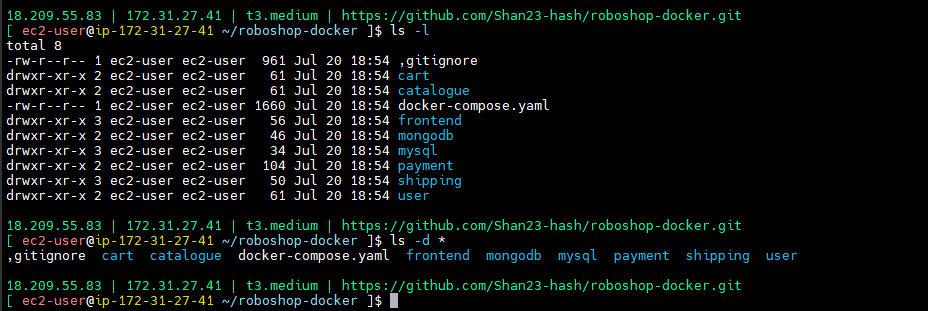


**--> cd roboshop-docker**

**--> ls**

**-->** all I will build I pushed to docker hub.

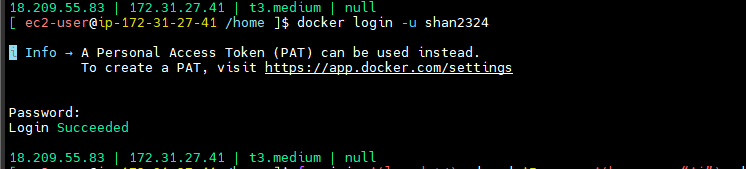
--> **ls -d \***



**--> for i in $(ls -d \*/); do cd $i; name=$(basename "i"); docker build -t shan2324/$name:v1 . ; docker push shan2324/$name:v1 ; cd .. ; done**

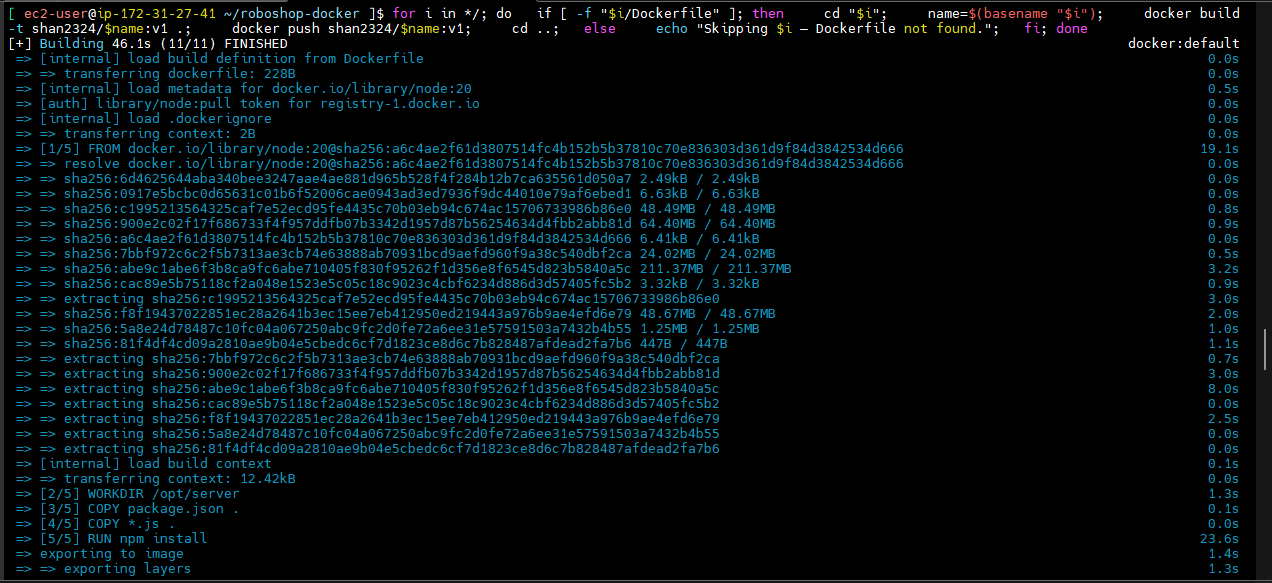
Error I got not login properly.

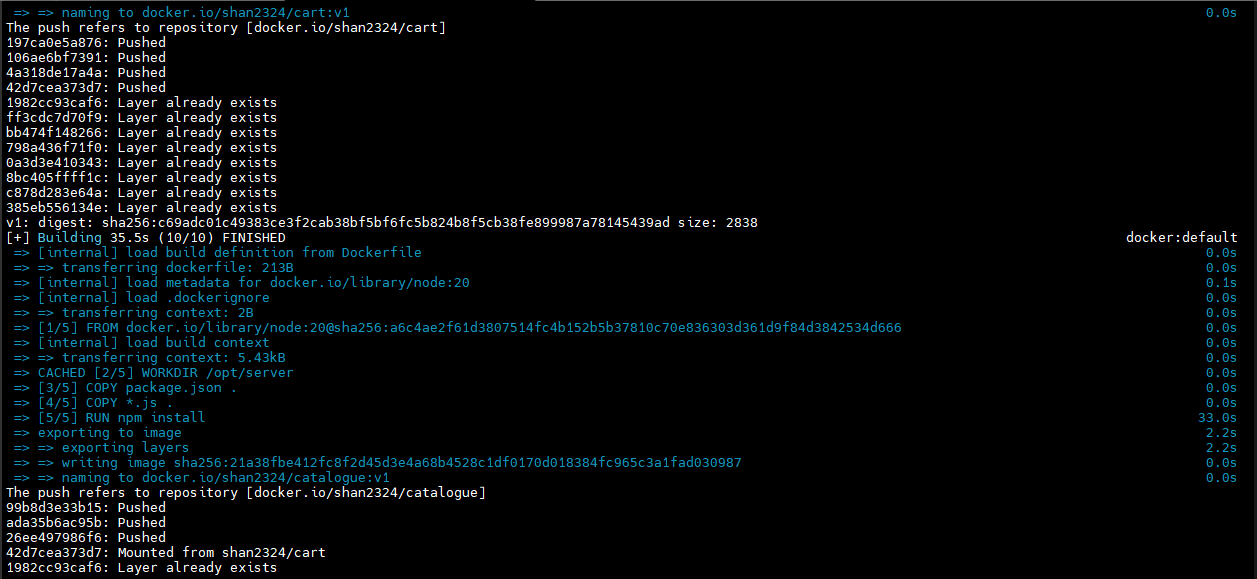
**docker login -u shan2324**



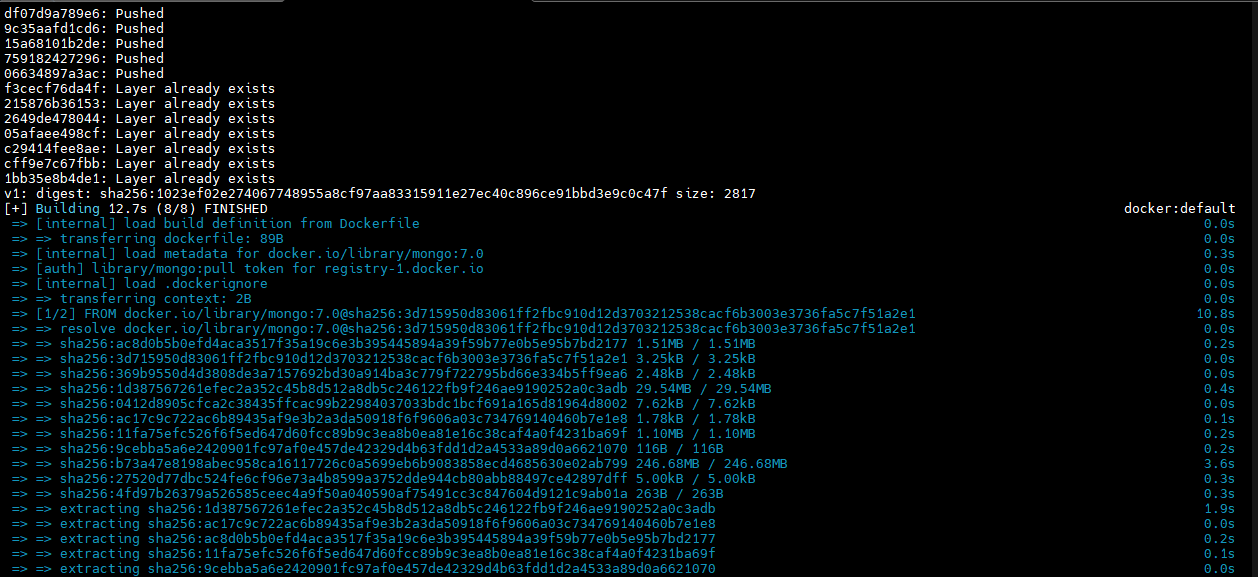
For chatgpt refernce

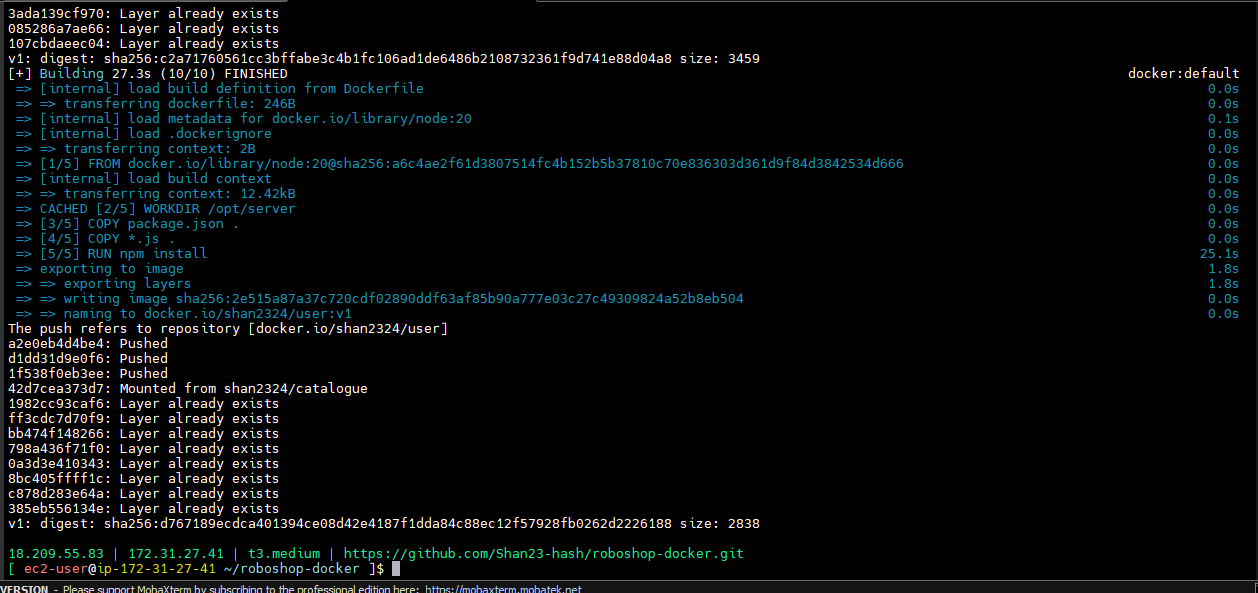
**for i in \*/; do if [ -f "$i/Dockerfile" ]; then cd "$i"; name=$(basename "$i"); docker build -t shan2324/$name:v1 .; docker push shan2324/$name:v1; cd ..; else echo "Skipping $i — Dockerfile not found."; fi; done**











**-->** base name names will remove slach and given name.

**OPTIMIZATION:**

--> use minimal official images (minimal images will use but official images also should use)

**<https://hub.docker.com/repositories/shan2324>**

**-->** we will start catalogue optimize.

--> catalogue almost there on 1.2gb.

--> we have to do proper test application have to work properly.

**1. **User Registration & Login****

The user should be able to register with valid details (name, email, password).

The user should then be able to log in using the registered credentials.

Expected: Login should return a success message and a valid token.

**2. Browse Product Categories**

After login, the user should be able to view a list of product categories.

Expected: Categories fetched from the catalogue service via MongoDB should appear on the UI or API.

**3. Add Products to Cart**

The user should be able to add one or more products to their cart using the "Add to Cart" option.

Expected: Cart should update with selected items and show totals and tax correctly.

**4. Add Shipping Information**

The user should select a shipping location.

The shipping cost should be calculated based on distance and location.

Expected: Shipping line item should be added to the cart and totals updated.

**5. Make a Payment**

The user proceeds to payment and submits valid details.

Expected: Payment should be processed and return a confirmation.

**6. Place the Order**

Once payment is successful, the user places the final order.

Expected: Order should be recorded and marked as placed.

**7. Verify Order Delivery**

The order should appear in the user's order history as “Delivered” or “Placed”.

Expected: Order status should reflect correctly.

--> this all will work means it’s success.

--> if incase because of decreased size only problem happens means that is big failure.

--> containers are empheral by default. Once you stored data after container deletion automatically data also will delete.

--> volumes creation

-v host-dir:container-dir

--> for example /usr/share/nginx/html -- nginx hgtml directory

--> so this you can take from host if you delete container then also it will be there in host.

--> this is unnamed or unmanaged values.

--> why we are telling un managed

--> if we create dir and manage it then those un-named/un-managed volumes.

--> if docker creates volumes and manages them then they are named/managed volumes.

--> so which one is better -- docker creation or our creation.

--> if docker created. Everything will manage docker. No need to do anything our side.

--> if docker created and docker will manage means that is easy.

Here we have two

**Stateless vs stateful**

**-----------------------------**

WEB APP DB

Where data is important is called stateful applications..

Our application code stateless because that is not data

--> wherever it’s storing data that is called storing application.

--> mongodb,mysql,rabbitmq and redis in this we have data.

--> web tier and app tier we have configuration files and application code that is alrady store in github.

--> that is not end user.

--> volumes should be created to stateful applications.

--> every database where it will store data.

--> mysql,mondodb any one -- in server one location will store.

--> if running mongodb server mondodb stored data or mysql stored data that one in underline host somewhere it will store.

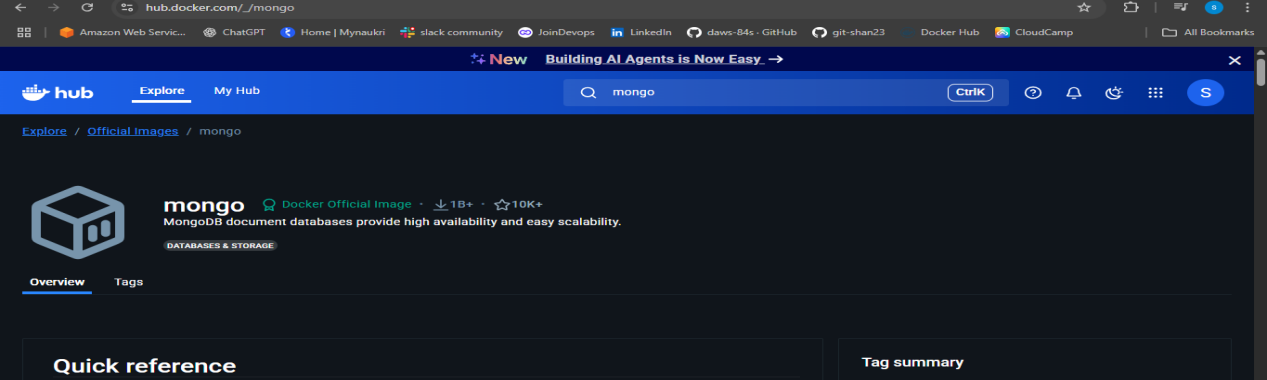
--> ofcourse you can see the tables and collections.

--> if you see the tables and collections.

--> databases store the actual data is host directories,while rendering for user it will shows tables and collections.

--> everydata base will store data in one directory.

--> mongodb have a directory.



--> $ docker run --name some-mongo -v /my/own/datadir:**/data/db** -d mongo

--> in this directory data will store.

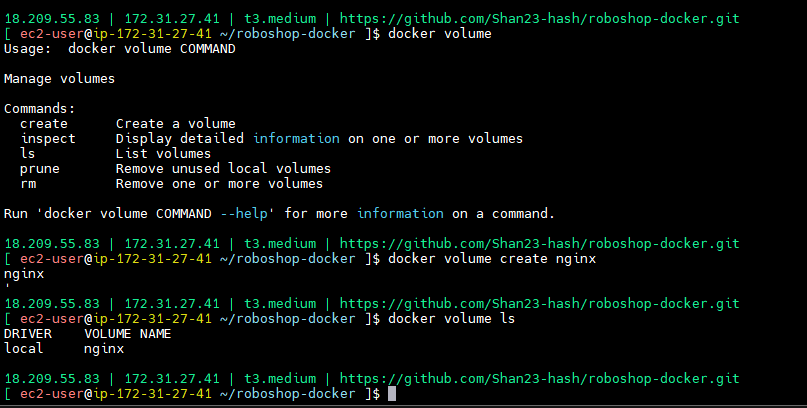
--> we are not interested in unnamed volumes. We are interested in named volumes.

--> we are created docker network same like docker volume.

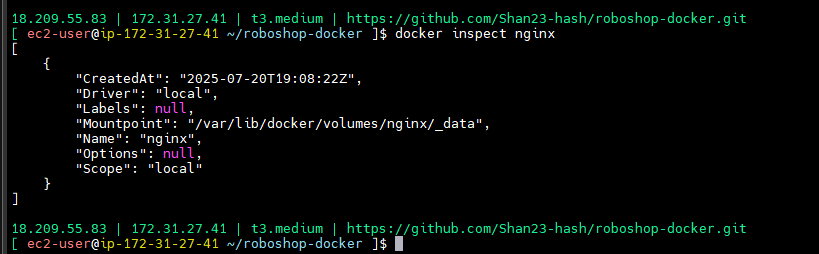
--> if you given **docker volume** - will get volume usage commands.

--**> docker volume create nginx**

**--> docker volume ls**



**docker inspect nginx**



/var/lib/docker/volumes/nginx/\_data -- here will store data

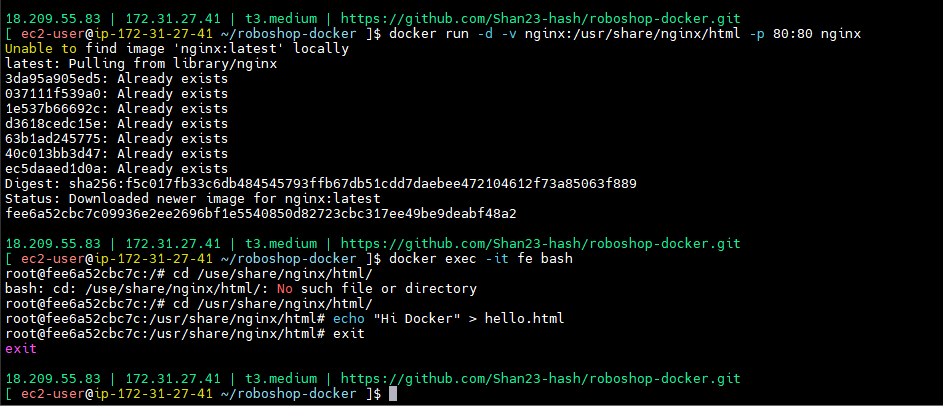
**docker run -d -v nginx:/usr/share/nginx/html -p 80:80 nginx**

**-->** nginx I’m not cretaed that is just volume

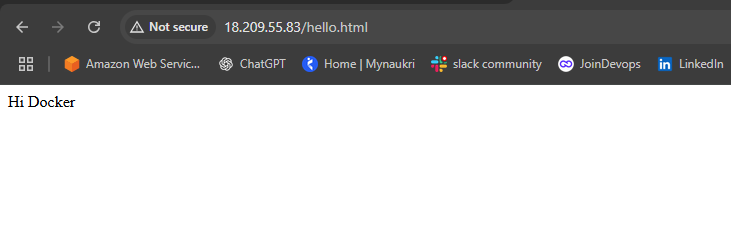
**docker exec -it fe bash**

**cd /use/share/nginx/html/**

**echo "Hi Docker" > hello.html**



<http://18.209.55.83/hello.html>



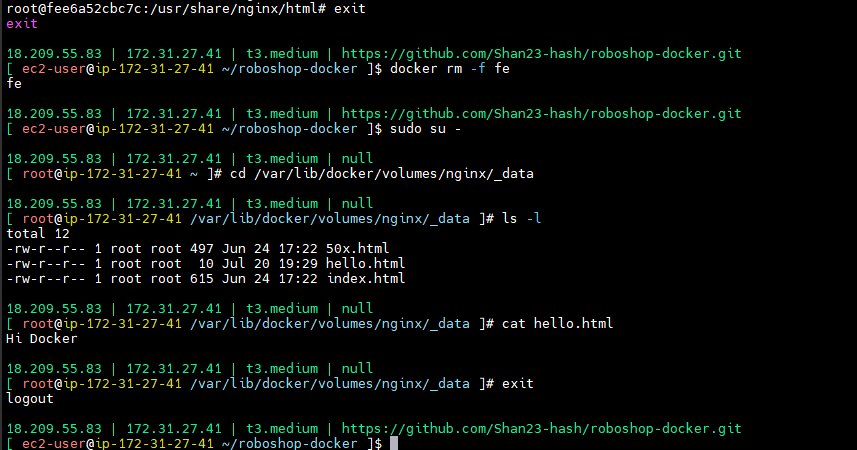
**docker rm -f fe**

**sudo su -**

**cd /var/lib/docker/volumes/nginx/\_data**

**ls -l**

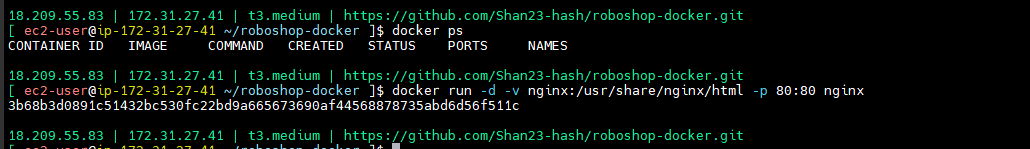
**cat hello.html**



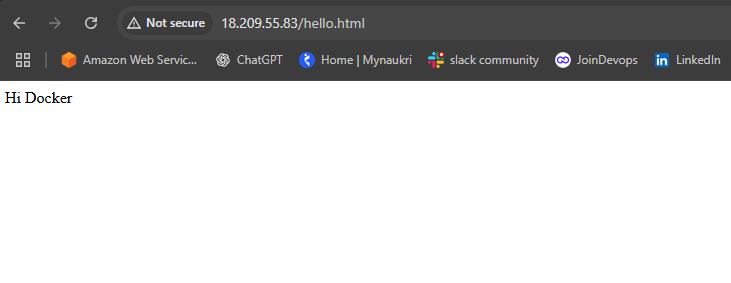
**-->** if I mount again data will mount same dirctory. Now not coming

--> **docker run -d -v nginx:/usr/share/nginx/html -p 80:80 nginx**

--> different containers but data diectory is same



**<http://18.209.55.83/hello.html>**



**-->** if you given docker volume creater, docker will create in home directory. This is named volume.

--> now I will create volumes same like networks.

--> if if you recreate container also it will not delete data.

--> docker compose volumes (dearch in google)

**docker-compose.yaml**

networks:

default:

driver: bridge

name: roboshop

#external: true #I created network already using docker network create roboshop

volumes:

mongodb:

redis:

mysql:

rabbitmq: # docker volume create rabbitmq

services:

mongodb:

image: joindevops/mongodb:v1

container\_name: mongodb

volumes:

- mongodb:/data/db

catalogue:

image: joindevops/catalogue:v1

container\_name: catalogue

depends\_on:

- mongodb

redis:

image: redis:7.0

container\_name: redis

volumes:

- redis:/data

user:

image: joindevops/user:v1

container\_name: user

depends\_on:

- redis

- mongodb

cart:

image: shan2324/cart:v1

container\_name: cart

depends\_on:

- catalogue

- redis

mysql:

image: shan2324/mysql:v1

container\_name: mysql

volumes:

- mysql:/var/lib/mysql

shipping:

image: shan2324/shipping:v1

container\_name: shipping

depends\_on:

- mysql

- cart

rabbitmq:

image: rabbitmq:3

container\_name: rabbitmq

environment:

RABBITMQ\_DEFAULT\_USER: roboshop

RABBITMQ\_DEFAULT\_PASS: roboshop123

volumes:

- rabbitmq:/var/lib/rabbitmq

payment:

image: shan2324/payment:v1

container\_name: payment

environment:

CART\_HOST: cart

CART\_PORT: 8080

USER\_HOST: user

USER\_PORT: 8080

AMQP\_HOST: rabbitmq

AMQP\_USER: roboshop

AMQP\_PASS: roboshop123

depends\_on:

- rabbitmq

- cart

- user

frontend:

image: shan2324/frontend:v1

container\_name: frontend

ports:

- "80:80"

depends\_on:

- catalogue

- user

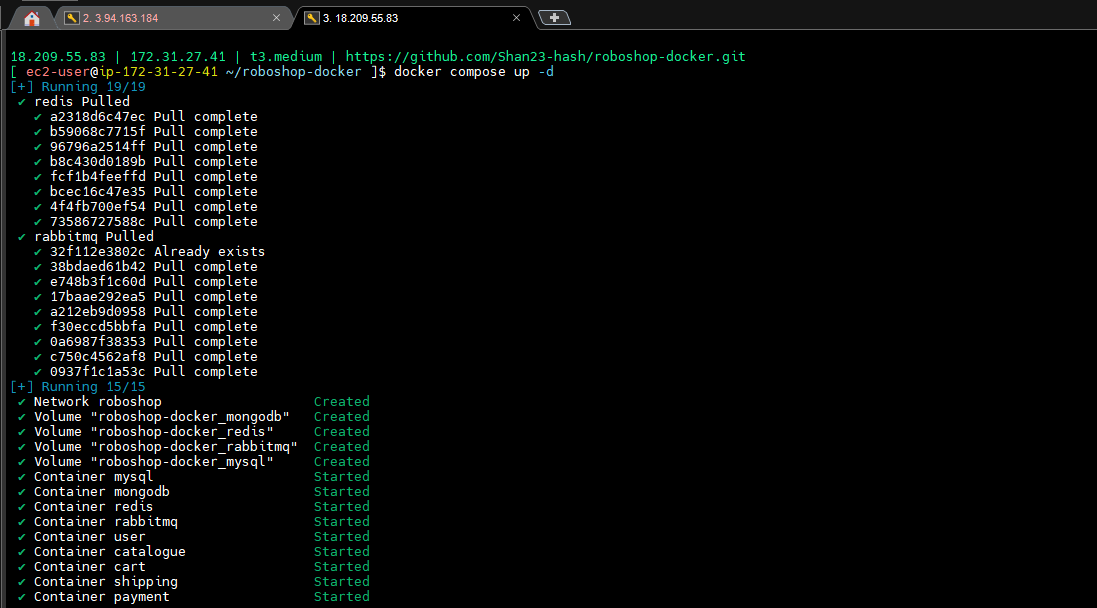
- cart

- shipping

- payment

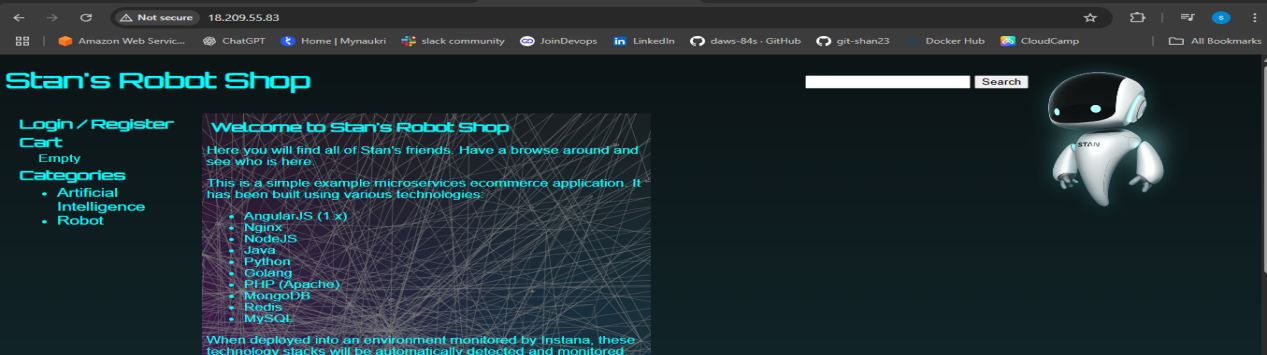
--> databases ate stright full we are storing the data in volumes.

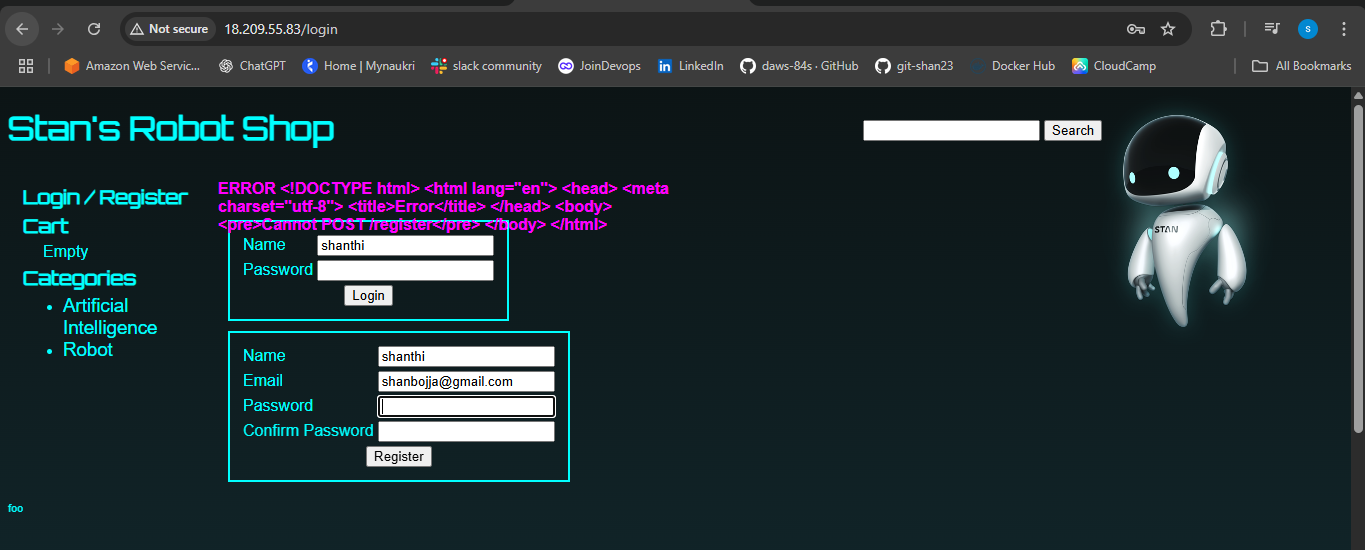
**docker compose up -d**



**See the output in google**

**<http://18.209.55.83/>**





i got this error

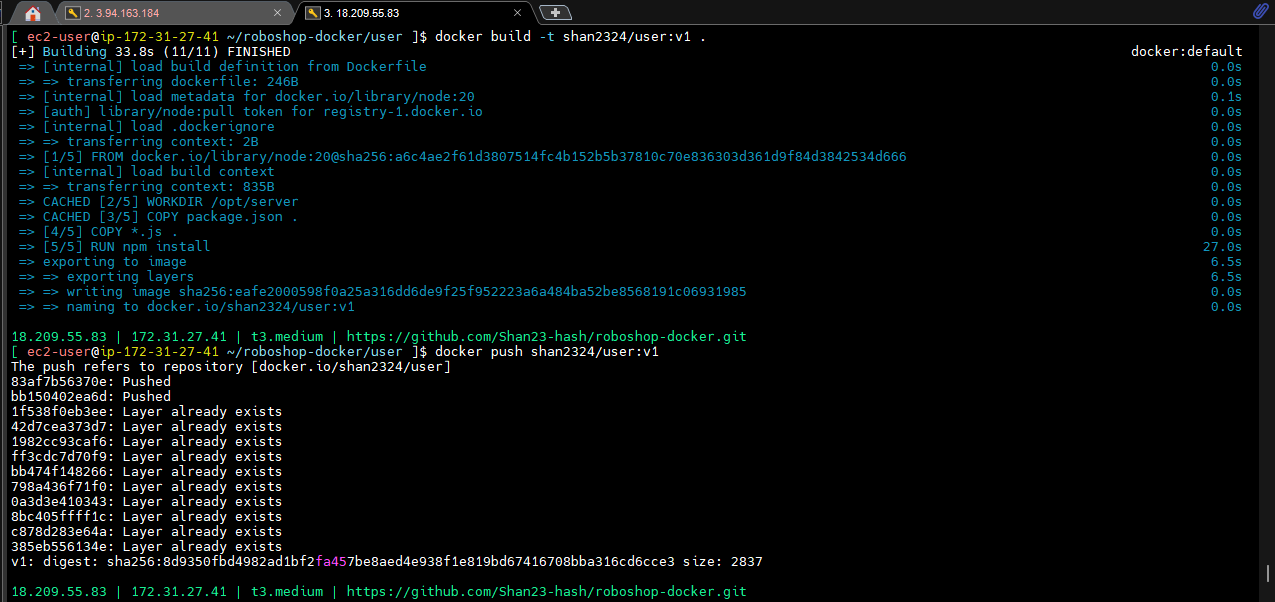
**cd ~/roboshop-docker/user**

**cat > server.js**

I changed server.js file

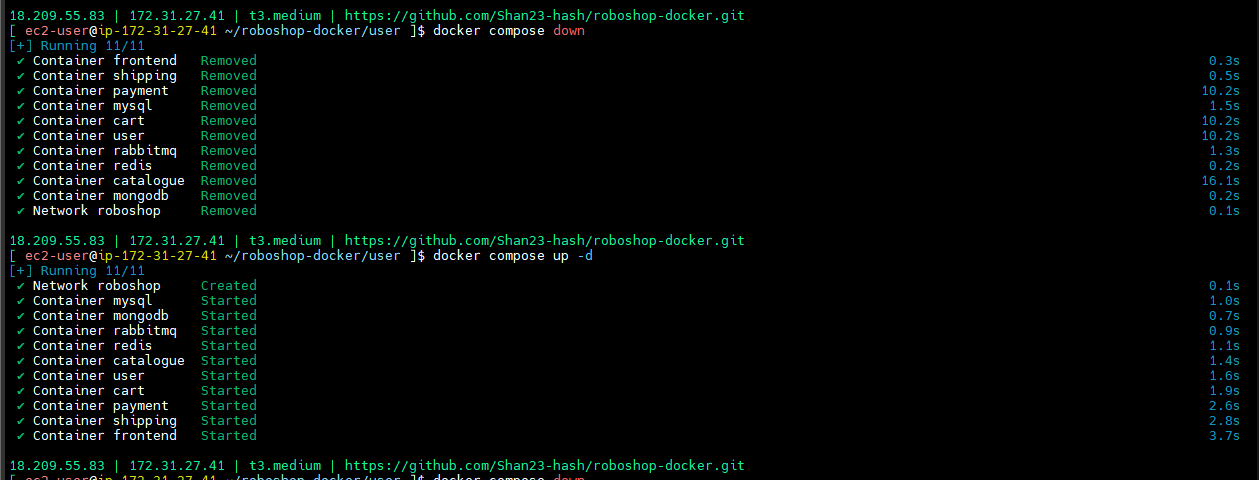
**docker build -t shan2324/user:v1 .**

**docker push shan2324/user:v1**

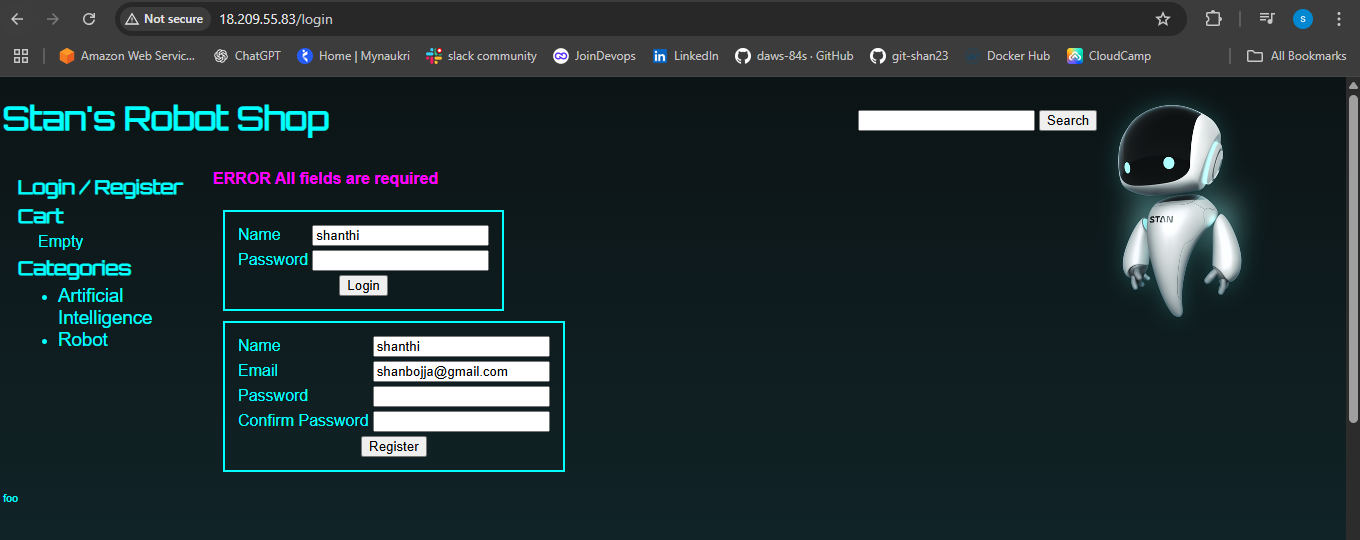


**docker compose down**

**docker compose up -d**



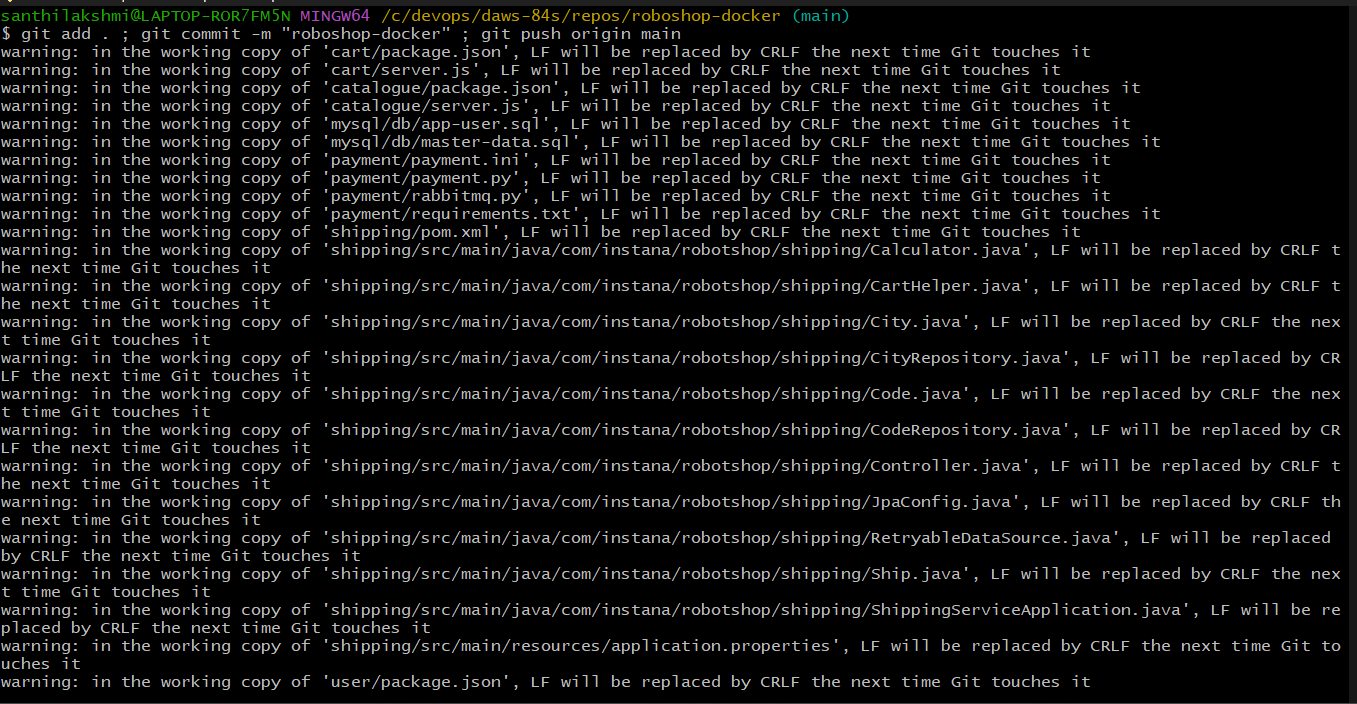
**<http://18.209.55.83/>**



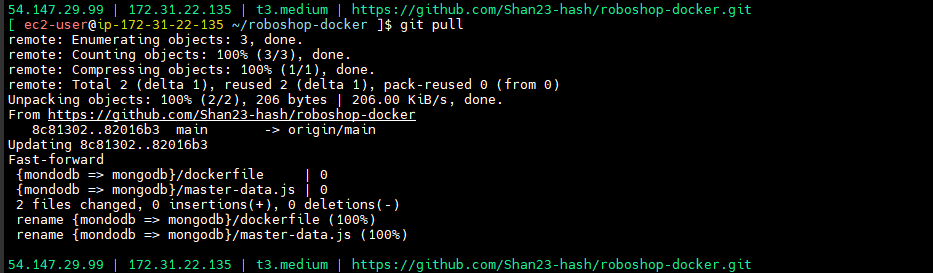
Still I’m getting error

--> I removed total roboshop-docker and I installed environments and I pushed and pulled.

**cd /c/devops/daws-84s/repos/roboshop-docker**



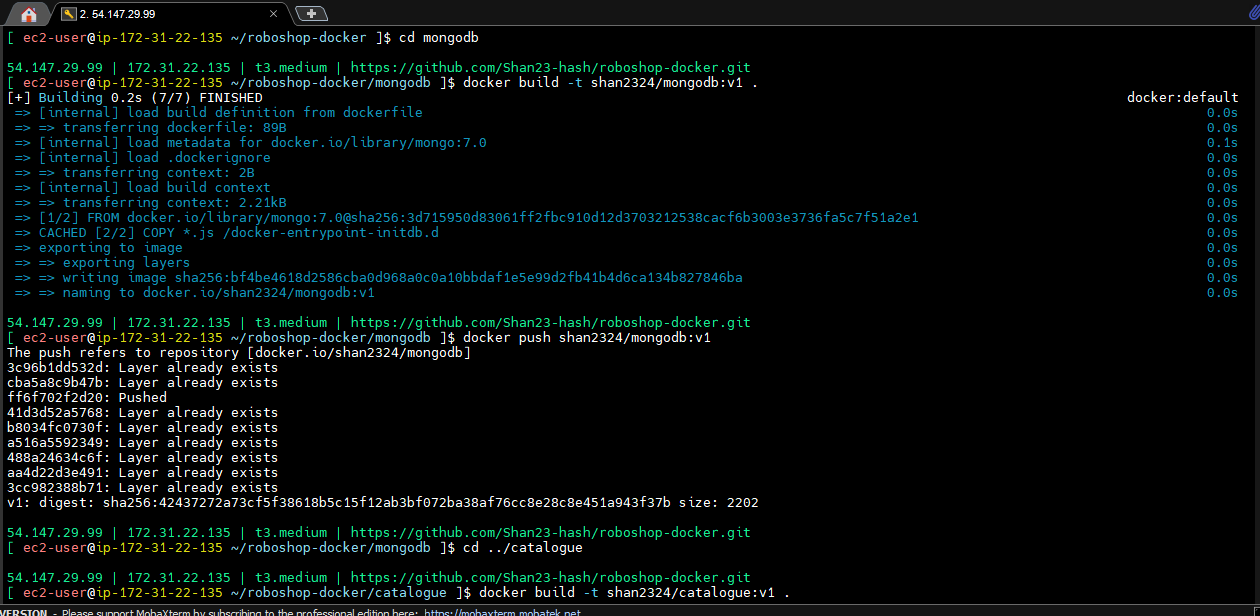
**git pull**



**cd mongodb**

**docker build -t shan2324/mongodb:v1 .**

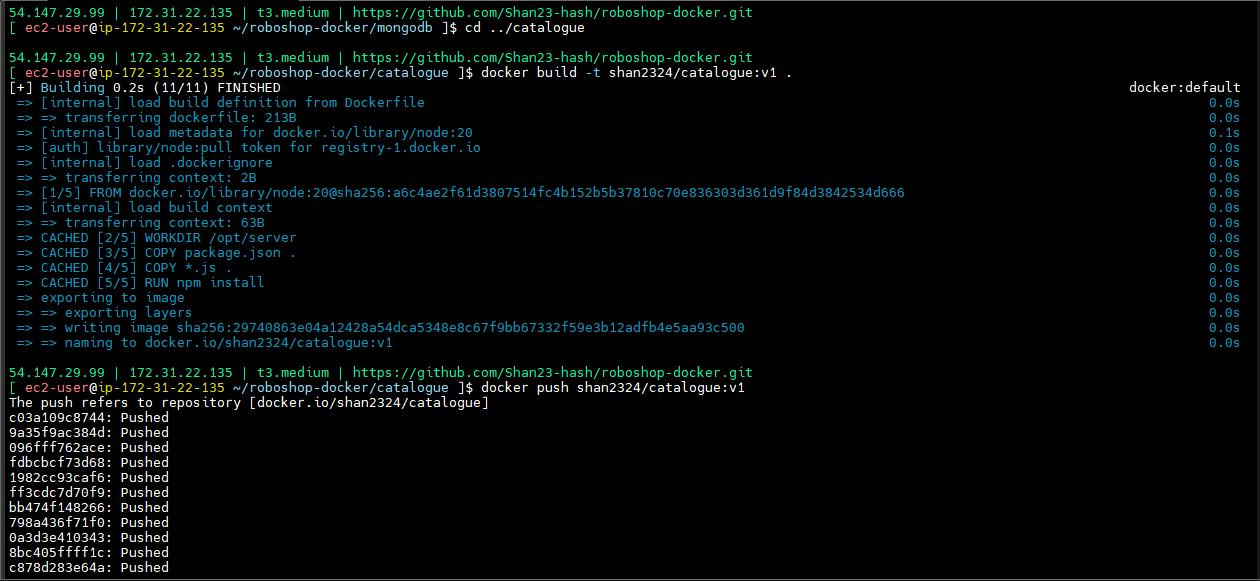
**docker push shan2324/mongodb:v1**



**cd ../catalogue**

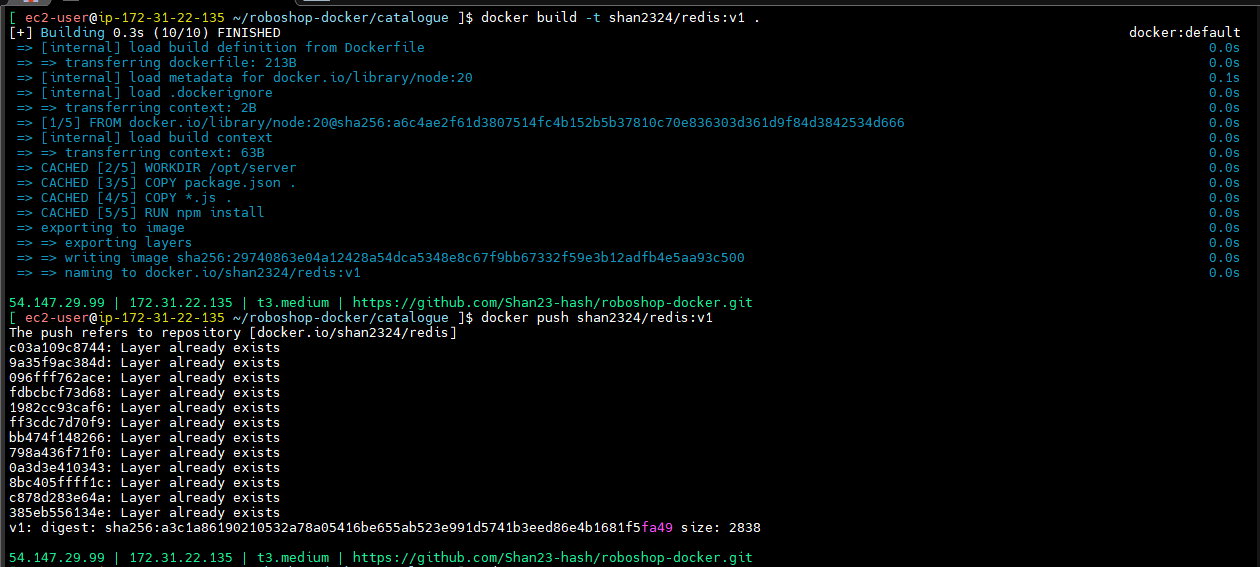
**docker build -t shan2324/catalogue:v1 .**

**docker push shan2324/catalogue:v1**



**docker build -t shan2324/redis:v1 .**

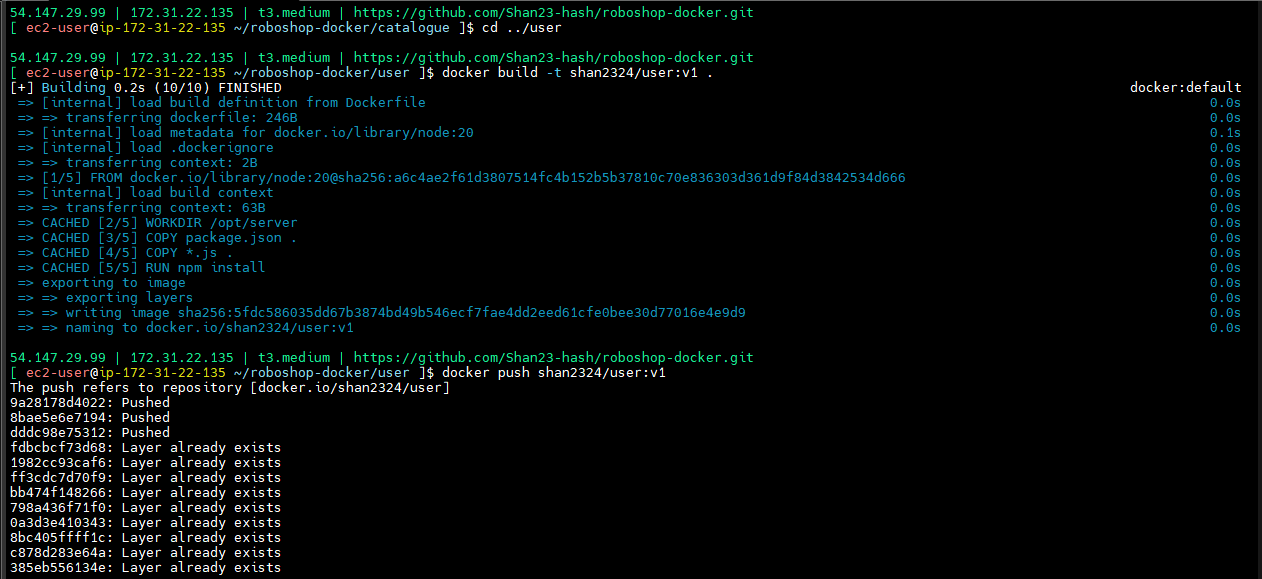
**docker push shan2324/redis:v1**



**cd ../user**

**docker build -t shan2324/user:v1 .**

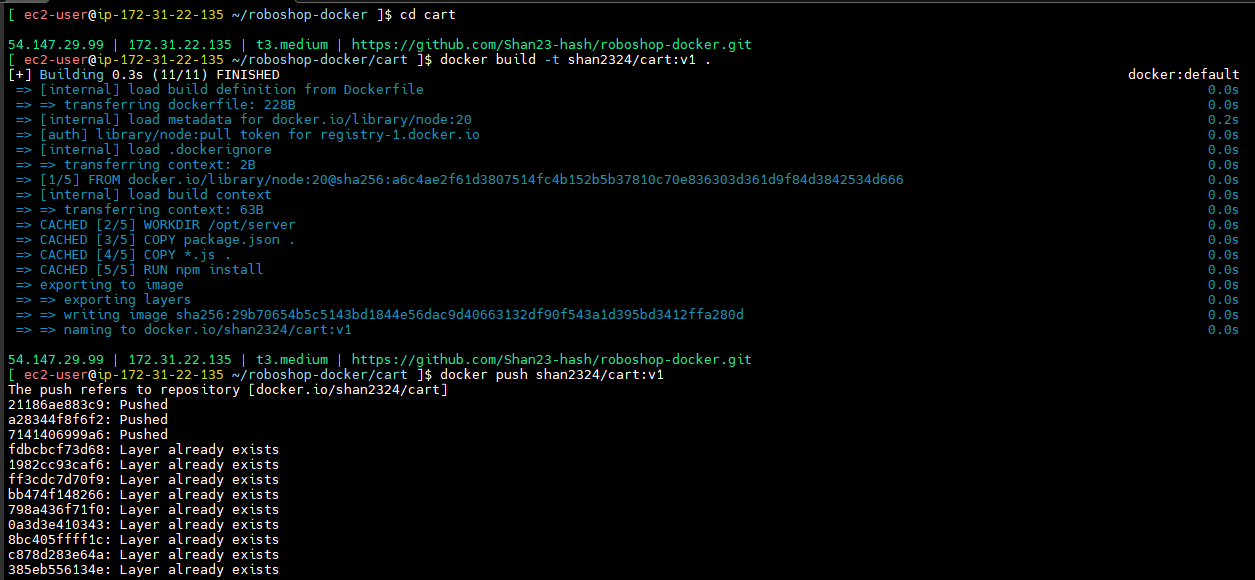
**docker push shan2324/user:v1**



**cd ../cart**

**docker build -t shan2324/cart:v1 .**

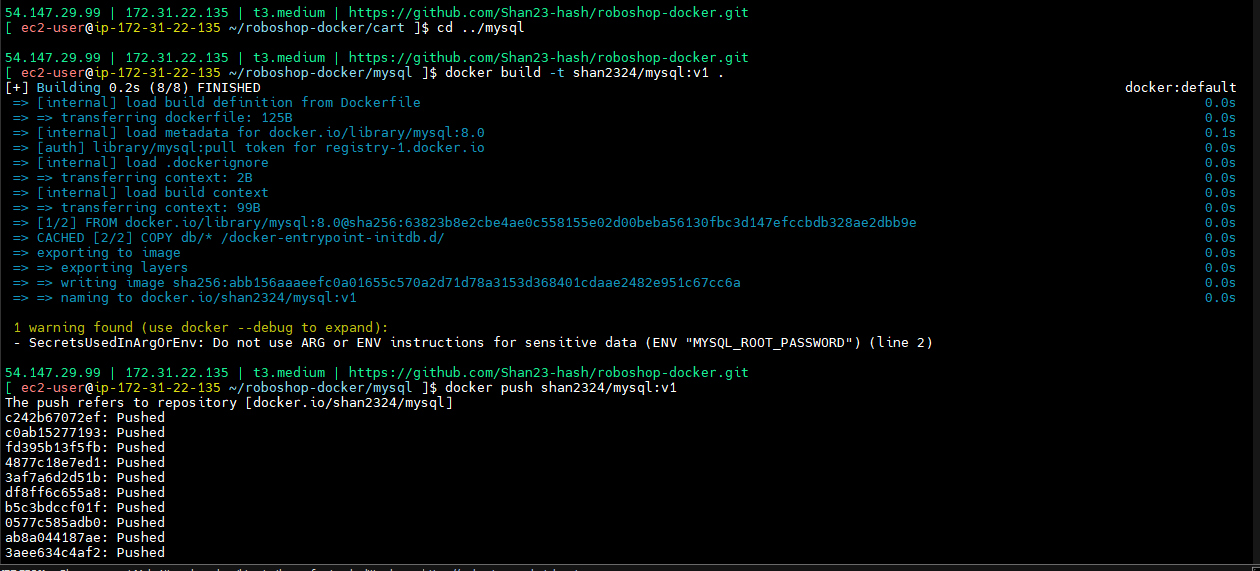
**docker push shan2324/cart:v1**



**cd ../mysql**

**docker build -t shan2324/mysql:v1 .**

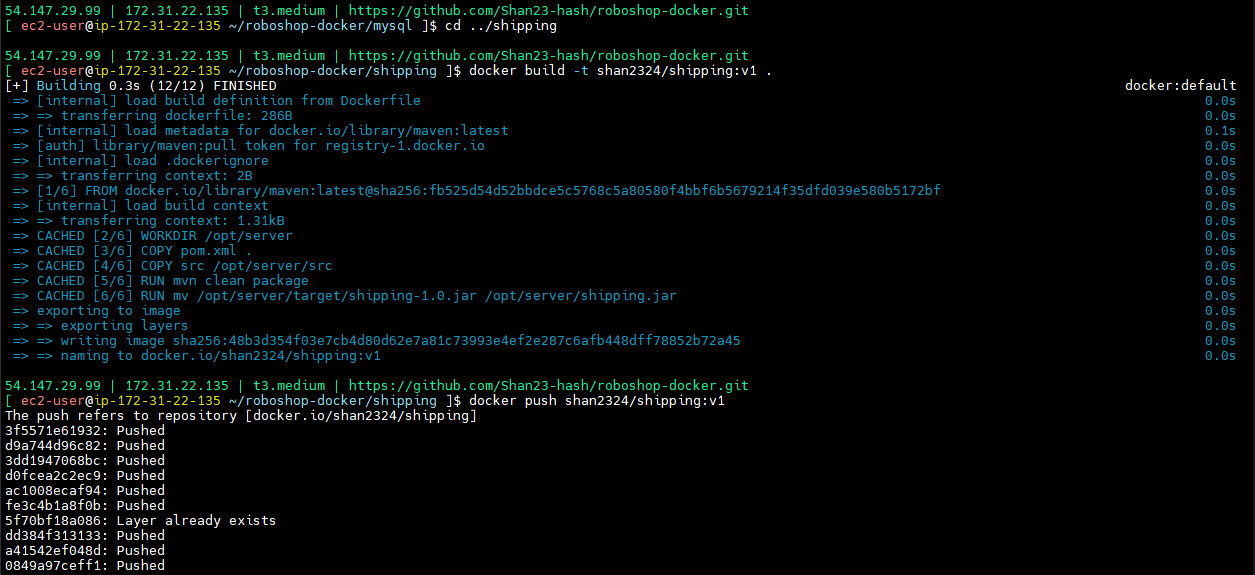
**docker push shan2324/mysql:v1**



**cd ../shipping**

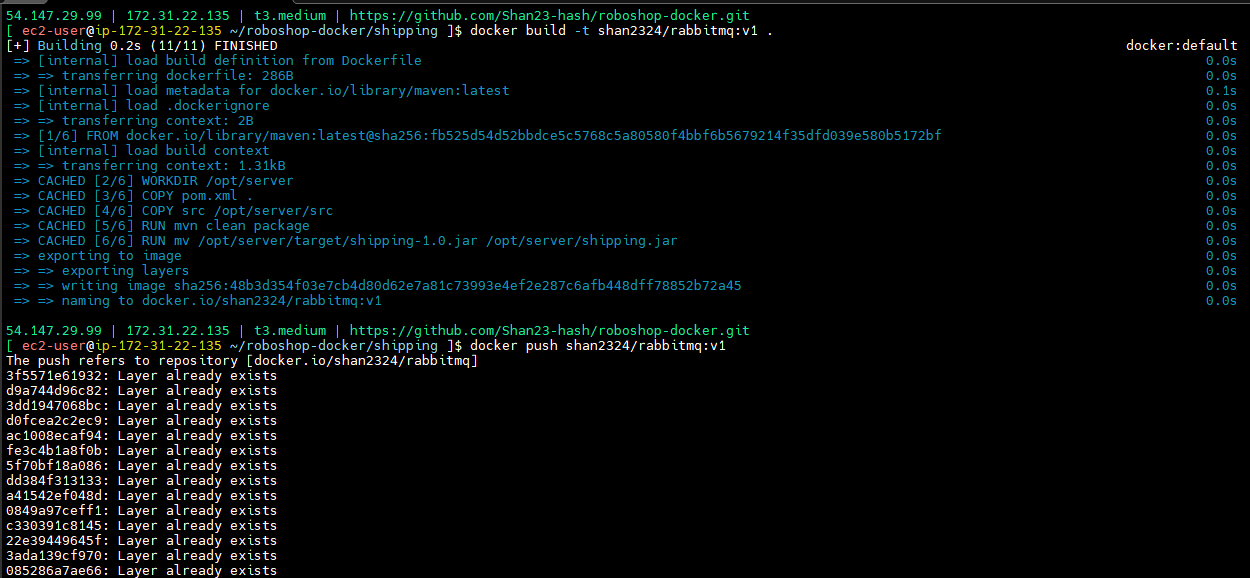
**docker build -t shan2324/shipping:v1 .**

**docker push shan2324/shipping:v1**



**docker build -t shan2324/rabbitmq:v1 .**

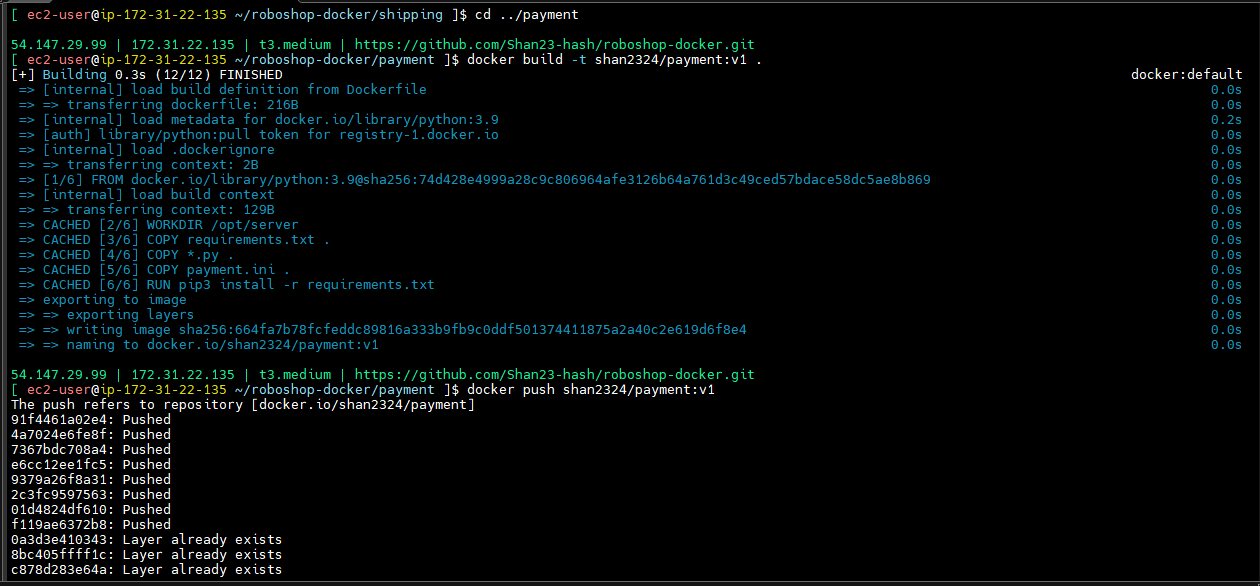
**docker push shan2324/rabbitmq:v1**



**cd ../payment**

**docker build -t shan2324/payment:v1 .**

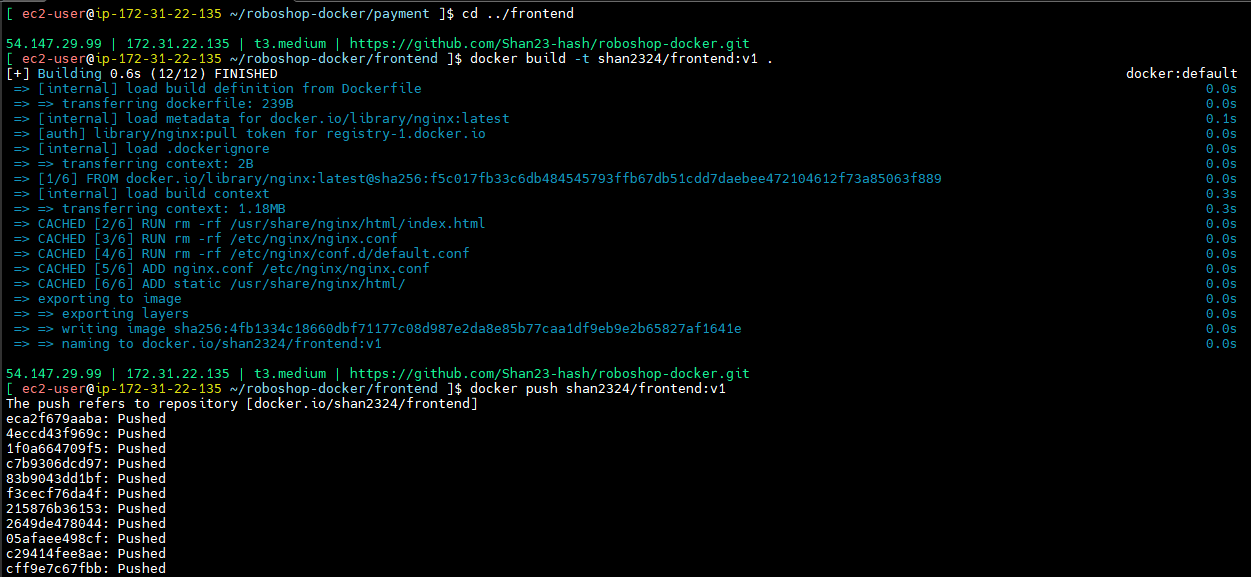
**docker push shan2324/payment:v1**



**cd ../frontend**

**docker build -t shan2324/frontend:v1 .**

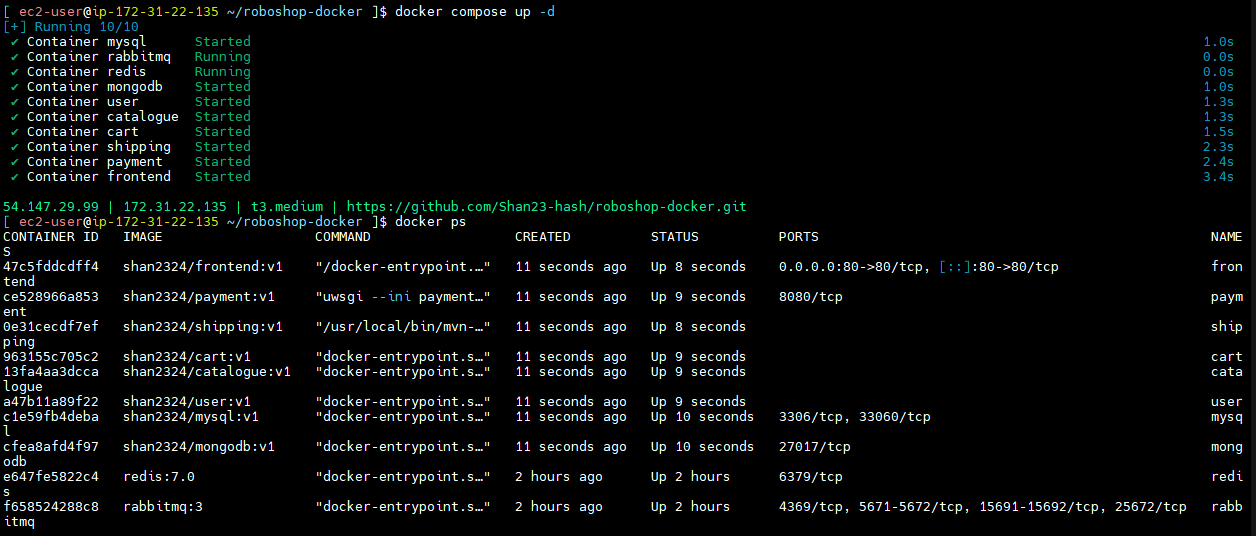
**docker push shan2324/frontend:v1**



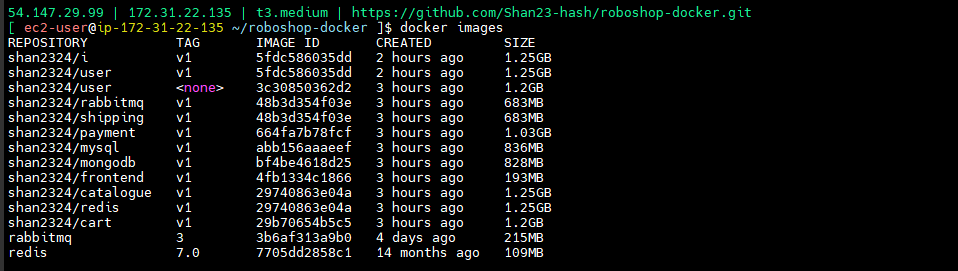
**cd ../**

**docker compose up -d**

**docker ps**



**docker images**



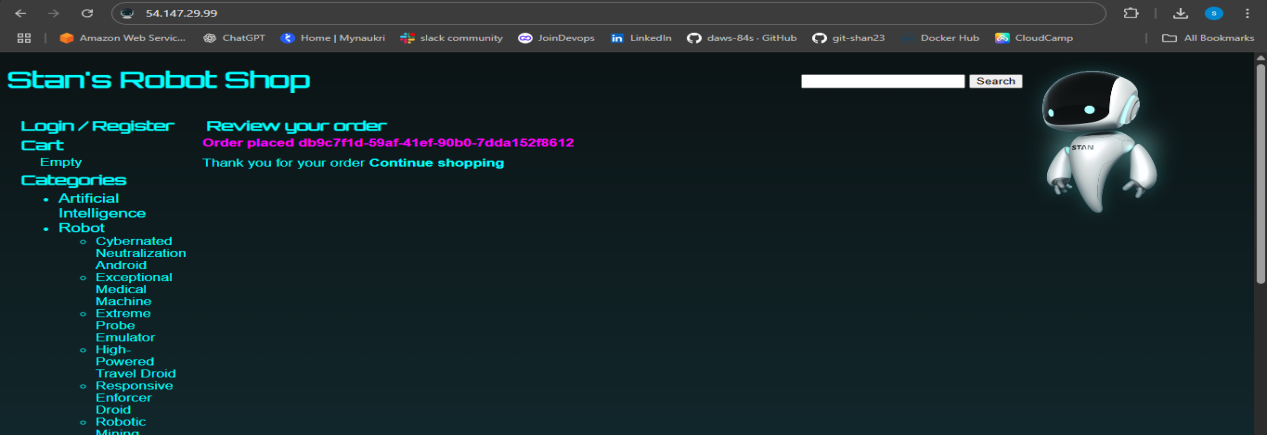
docker logs user

docker network ls

docker logs redis

docker logs frontend

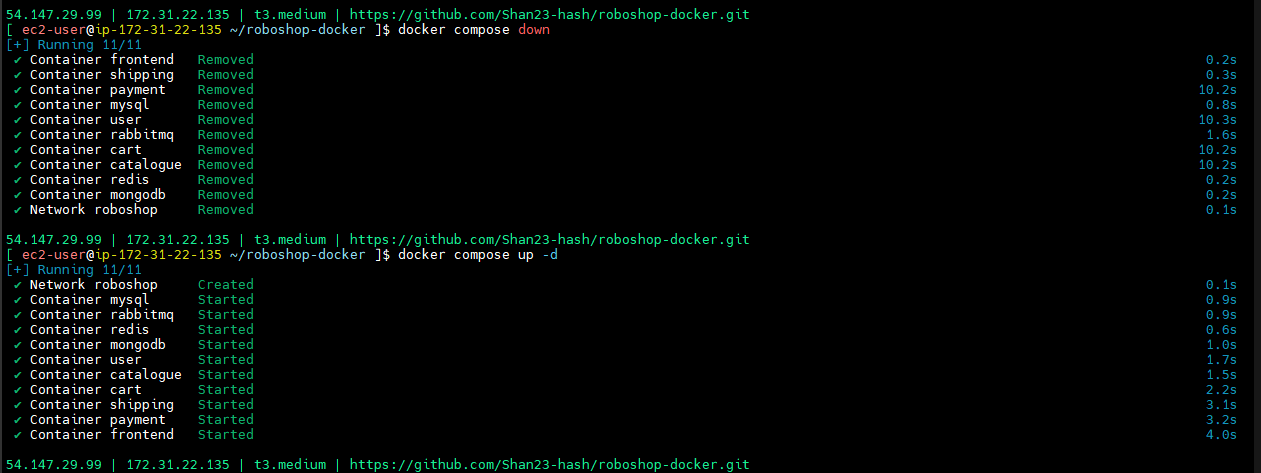
**<http://54.147.29.99/>**



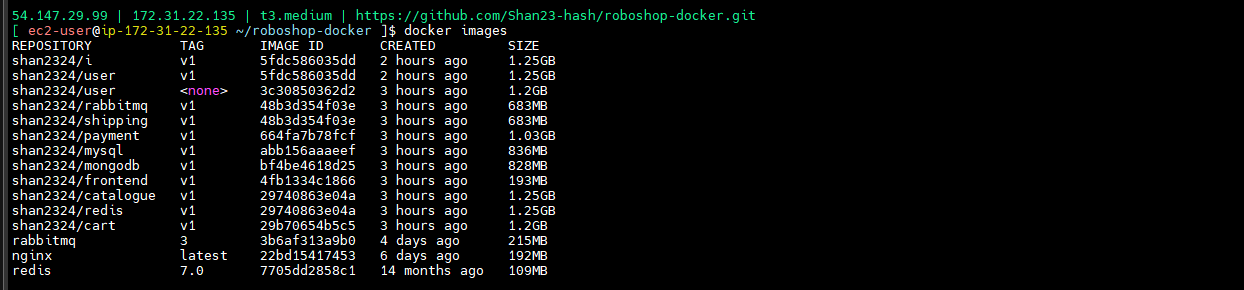
--> Docker evrything

**docker compose down --** volumes not deleted

**docker compose up -d --**  after I ran this I can login and see the order

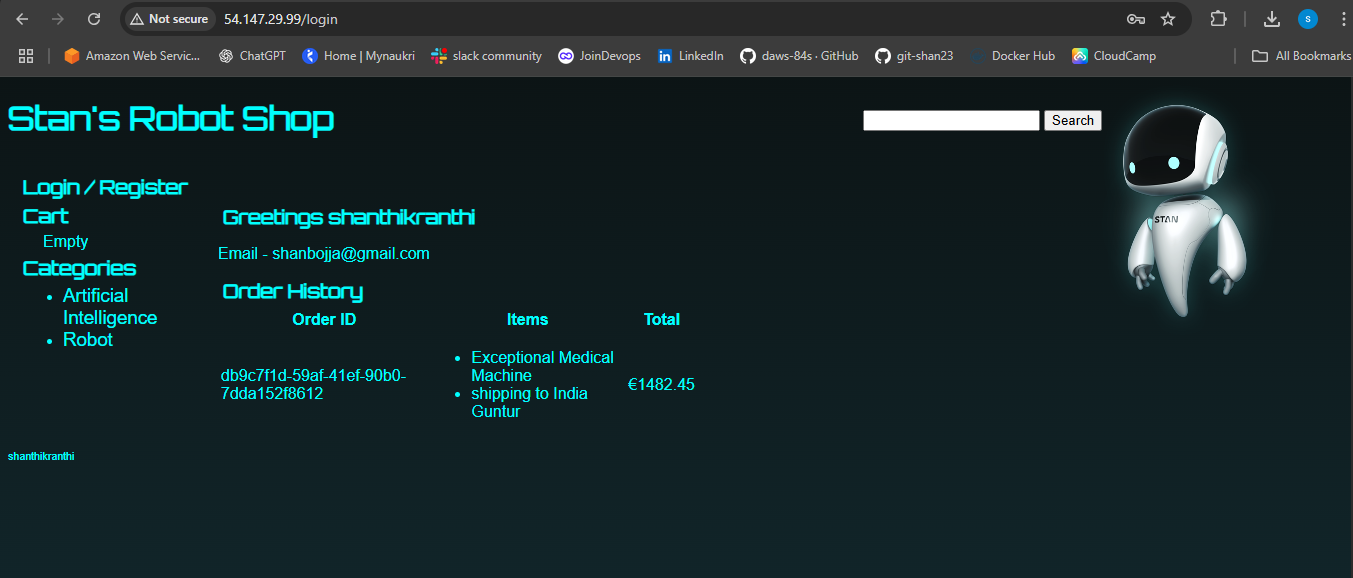


**docker images**

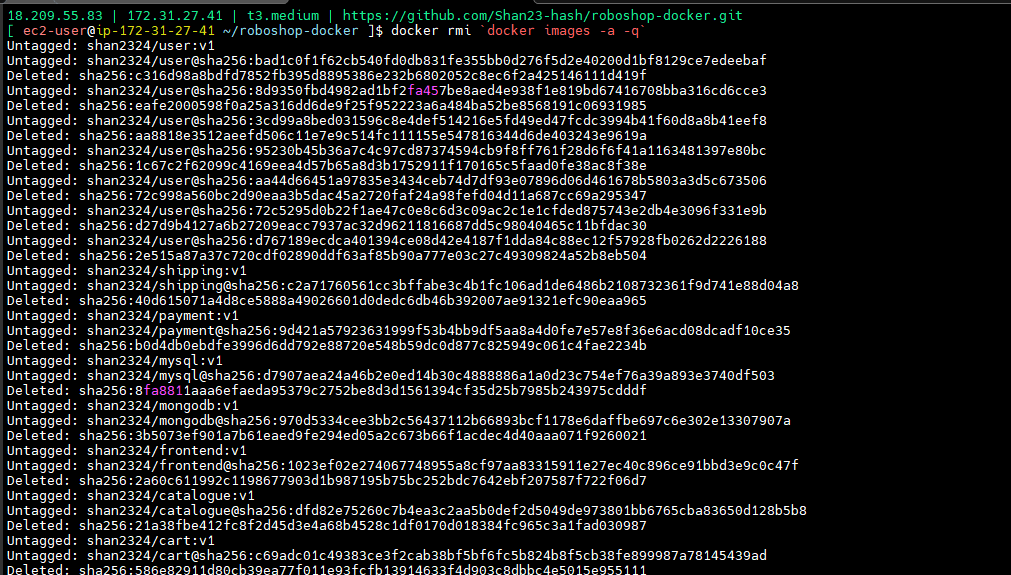


<http://54.147.29.99/login>

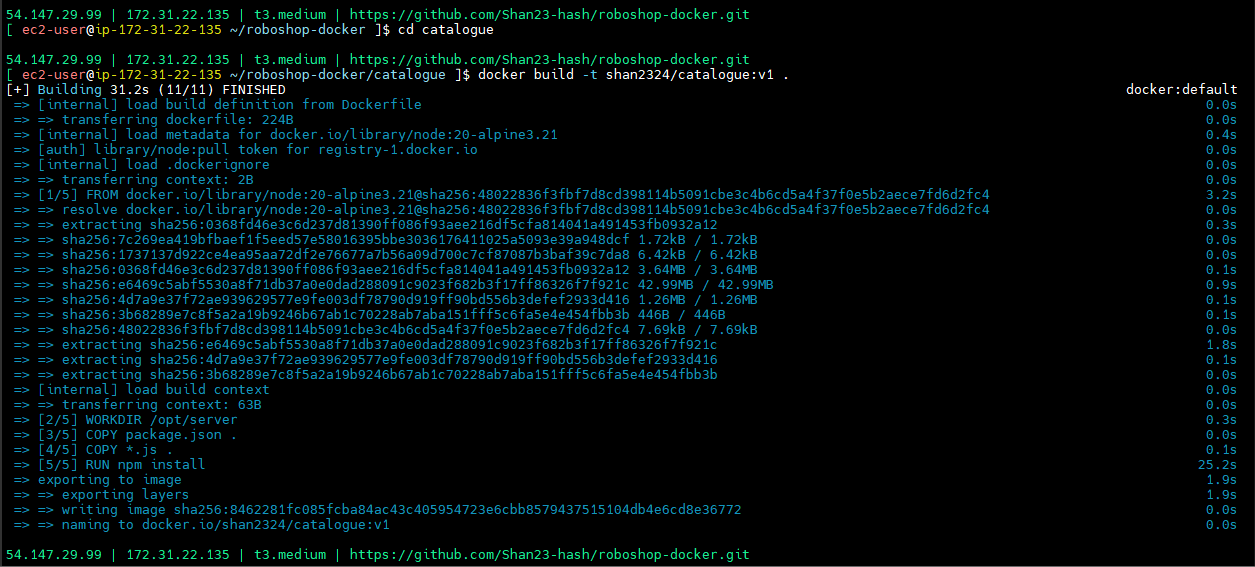
I did compose and down and up then I checked my previous it was there.



**docker rmi `docker images -a -q`**

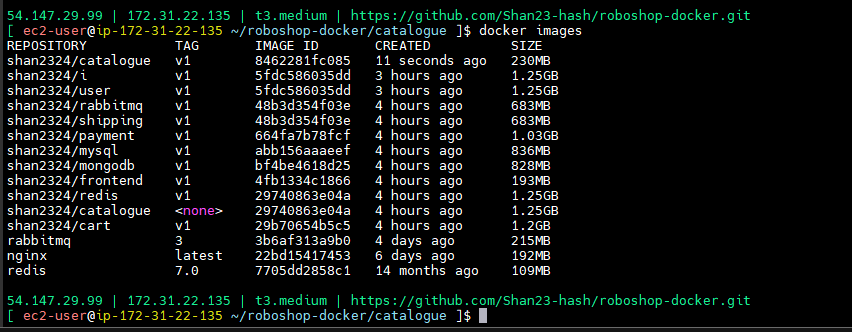


**docker images**

* There is no images if i run docker compound application will run.
* Docker image contains os
* **docker compose up -d**
* If run any server you given docker compose command it will run.
* Docker does not clear about the umderlinr hodt so that is why docker is called portable.
* In any server docker compose will run.
* Every container has a operating system.
* Docker image contains obously OS and small OS that’s why it is called as portable.
* Poratable means anyewhere you can move it should work on same.
* The resources not there in host. it will not work
* . Host supply enouf resources.
* Docker wants space. That can be any os. Unserline host OS can be anything. Docker works same that is why docker is called portable.
* **docker images**
* See the size now we can increase the size
* FROM node:20-alpine3.21 -- this is the base OS.in **catalogue/Dockerfile file.**
* git push and pull
* cd roboshop-docker
* **cd catalogue**
* **docker build -t shan2324/catalogue:v1 .**
* 
* **docker push shan2324:/catalogue:v1**
* cd ..
* **docker compose up -d**

It will recreate

Now see image size

* **docker images**
* 
* See the size the size 230MB also 1GB decreased.
* see the application whether it’s working or not
* Now which use reagoin this one is running
* It’s running root user.
* It’s not good for running root user.
* We need to create a system user and then run it as finally with normal user.
* Alpine add user and group (search in google)
* Create a group and user
* RUN addgroup -S roboshop && adduser -S roboshop -G roboshop
* Now who should have opt /server RUN chown -R roboshop:roboshop /opt/server
* USER roboshop
* From now if I run roboshop below all commands will run roboshop user.
* Ownership also will take roboshop because from it will take over roboshop.

**catalogue/Dockerfile**

FROM node:20-alpine3.21

RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

WORKDIR /opt/server

RUN chown -R roboshop:roboshop /opt/server

USER roboshop

COPY package.json .

COPY \*.js .

RUN npm install

ENV MONGO="true" \

MONGO\_URL="mongodb://mongodb:27017/catalogue"

CMD ["node","server.js"]

* Push and pull the code
* **docker build -t shan2324/catalogue:v1 .**
* **docker push shan2324:/catalogue:v1**
* Now image was created.
* docker compose up -d
* Let us see whether it is running or not.
* **docker exec -it catalogue sh**
* **ls -l**

**catalogue/Dockerfile**

FROM node:20-alpine3.21

RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

RUN chown -R roboshop:roboshop /opt/server

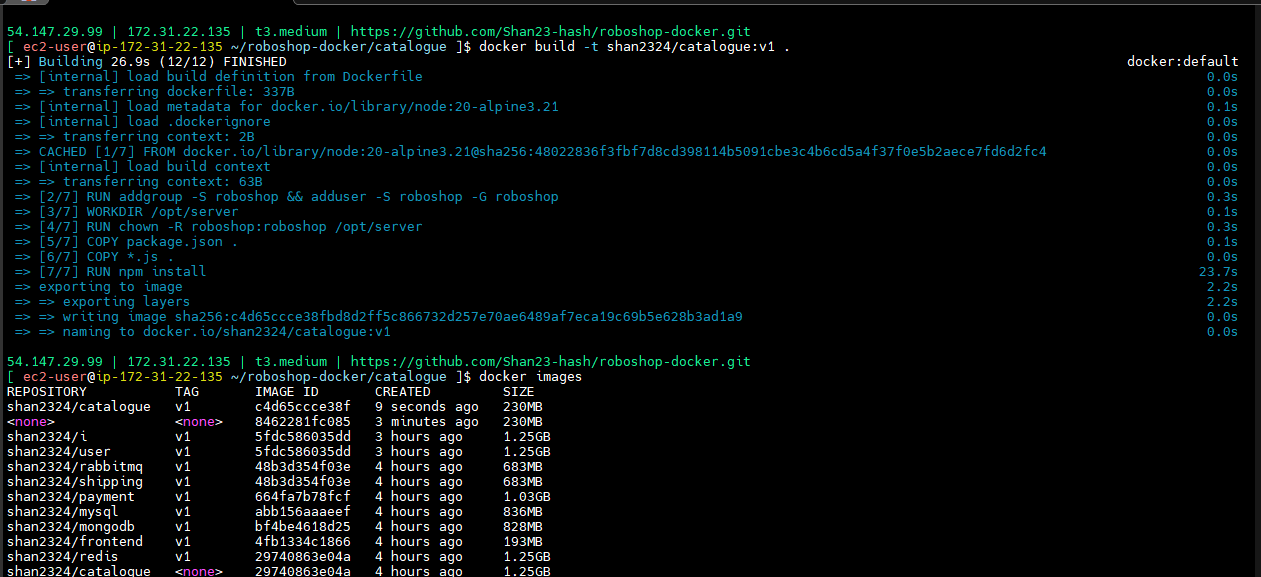
ENV MONGO="true" \

MONGO\_URL="mongodb://mongodb:27017/catalogue"

USER roboshop

CMD ["node","server.js"]

* Before let them do anything last is directory is roboshop.
* You are running the container with roboshop user.
* Use minimal images
* Use official images
* Run it as non root containers
* Multi stage builds
* Optimise docker layers
* This all we did here after npm install some memory will cache
* After running and install the package then cache the data.
* **docker images** -- see size 230MB



* We can decrease through Multi stage builds
* **docker exec -it catalogue sh** -- we need this
* Our total code is here only
* Ultimately we need /opt/server

**catalogue/Dockerfile**

FROM node:20-alpine3.21 AS builder

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

FROM node:20-alpine3.21

RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

ENV MONGO="true" \

MONGO\_URL="mongodb://mongodb:27017/catalogue"

WORKDIR /opt/server

USER roboshop

COPY --from=builder /opt/server /opt/server

CMD ["node","server.js"]

# FROM node:20-alpine3.21

# RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

# WORKDIR /opt/server

# COPY package.json .

# COPY \*.js .

# RUN npm install

# RUN chown -R roboshop:roboshop /opt/server

# ENV MONGO="true" \

# MONGO\_URL="mongodb://mongodb:27017/catalogue"

# USER roboshop

# CMD ["node","server.js"]

--> multi stage build docker (check in google)

**Multistage defination**

We can use multi stage builds it is like multiple docker files inside a singlr docker file you can use one docker file as a builder anthoer docker file as final image. You can copy whatever you want from builder and then paste it into the final image. So that we can save some space.

--> this one will work main java image.

multi stage builds are like keeping multiple Dockerfiles in a single Dockerfile, one Dockerfile we can use it as builder, another Dockerfile we can use it as final image, we can copy what we want from builder to final image. this reduces memory and then unnecessary installations.

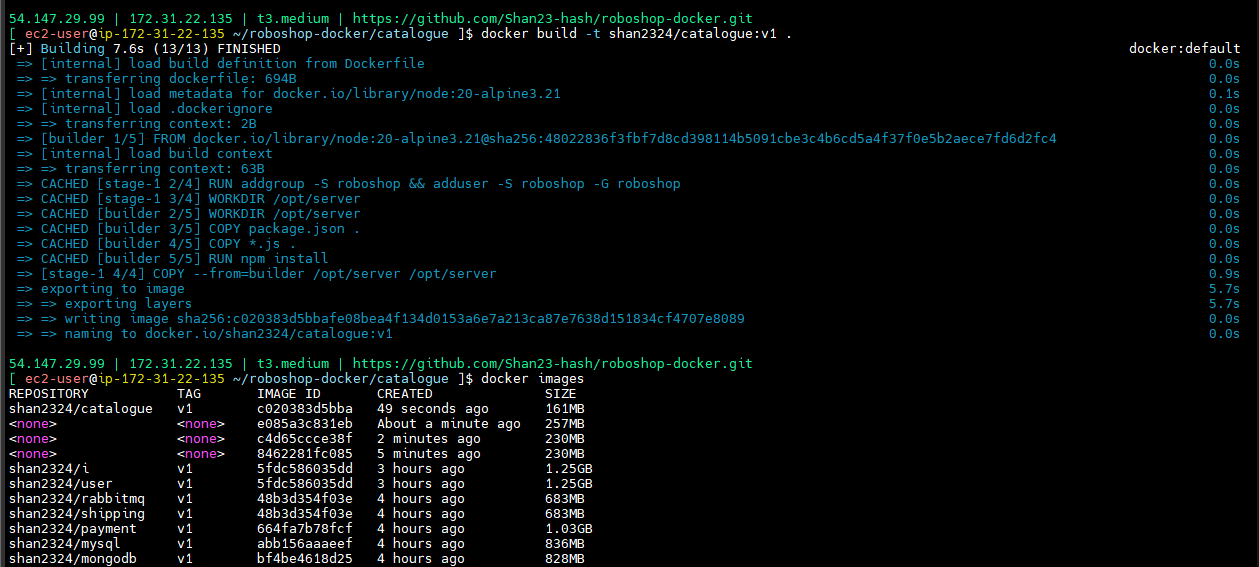
--> unnecessary installations not doing means attaching surface will manage.

--> low memory and low packages are safe that is use of multi satge builds.

--> Now we can run this one see

--> git push and pull

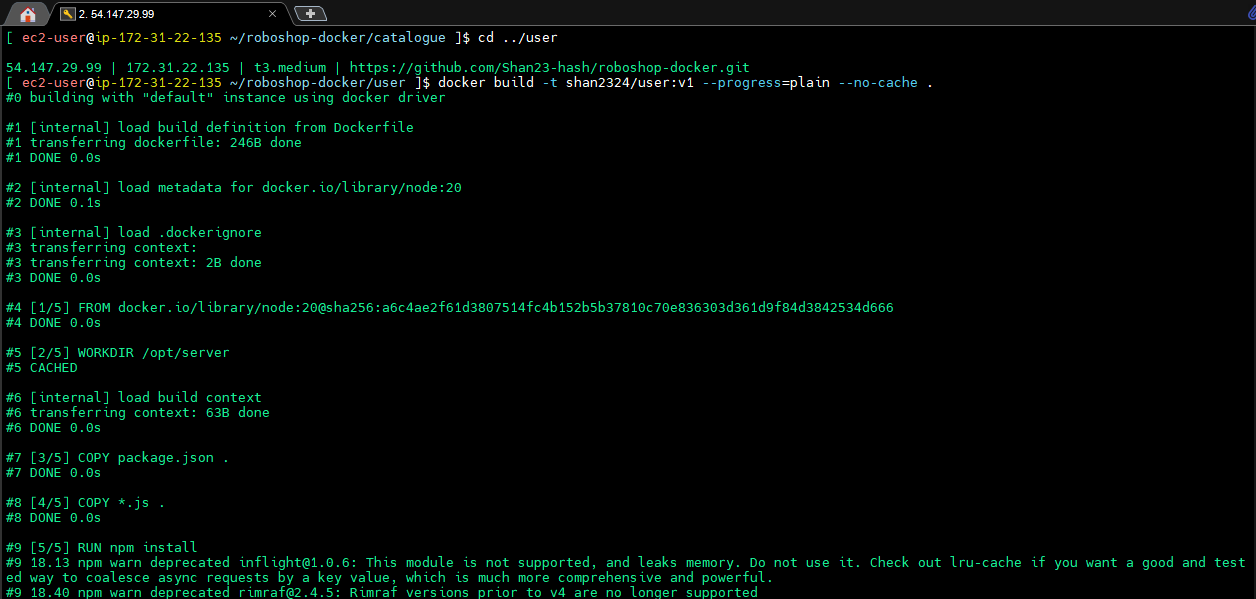
* **cd catalogue**
* **docker build -t shan2324/catalogue:v1 .**
* **docker push shan2324:/catalogue:v1**
* **docker images - 161MB**



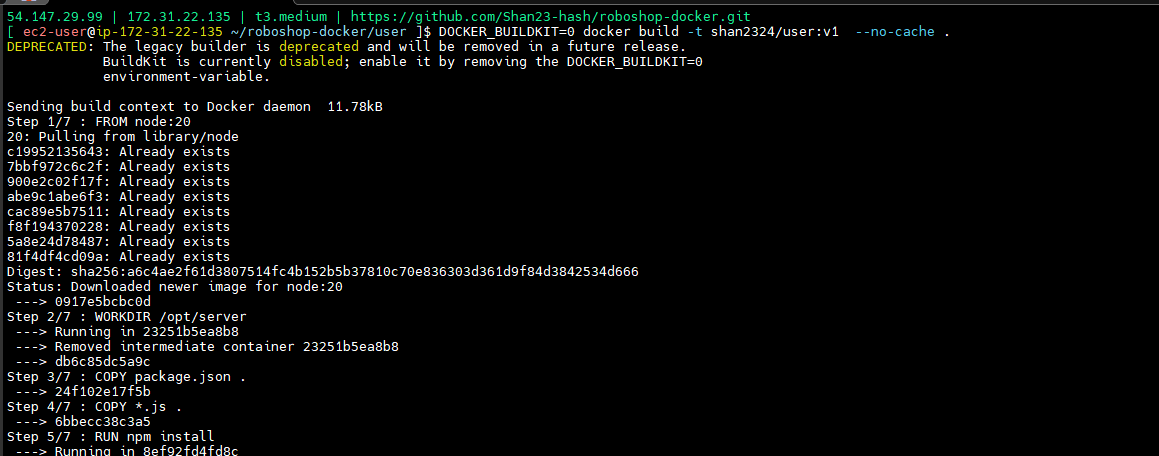
* **docker compose up -d**
* It’s working fine
* That memory save in whle installing npm install cache.

**Optimize docker layers**

* For this I will take user
* **cd ../user**
* **docker build -t shan2324/user:v1 --progress=plain --no-cache .**



* Docker build see intermidate container outputs -- check in google.
* So this one will set a environment variable
* **DOCKER\_BUILDKIT=0**
* **docker build -t shan2324/user:v1 --progress=plain --no-cache .**
* Still same
* **docker build -t shan2324/user:v1 .**
* **docker build -t shan2324/user:v1 --no-cache .**
* **cd ~/.docker/**
* **ls -l**
* **cd buildx/**
* **ls -l**
* **cd ..**
* **cat config.json**
* **cd /roboshop-docker/user**
* **DOCKER\_BUILDKIT=0 docker build -t shan2324/user:v1 --no-cache .**



**Diagram**

-->This one will take first instruction. From first instruction will create a container.

On that It will run second instruction.

--> in this container have two instructions.

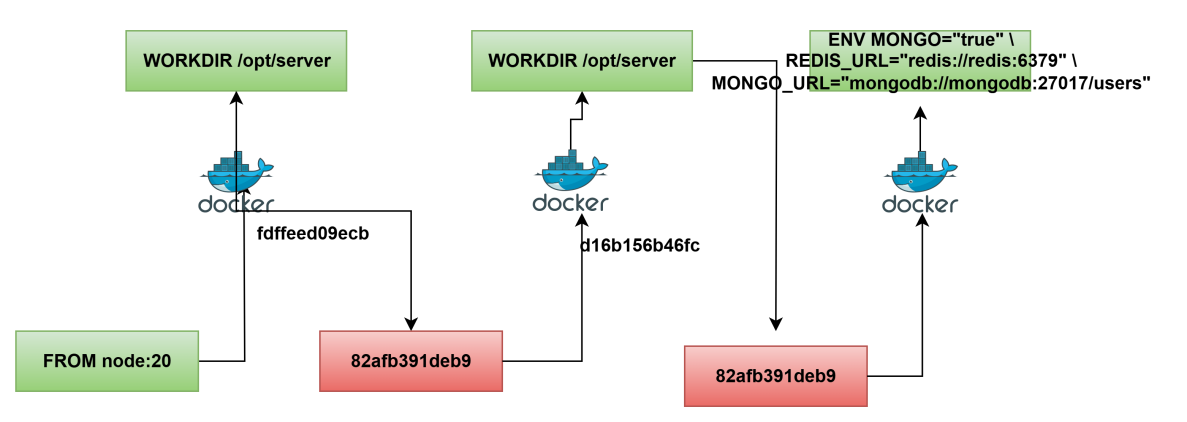
--> running in intermidate container.

--> removed intermediate container

--> from first instruction image was pulled. -- downloaded newer image

--> from this image container was created.

--> in container it is ran step-2



--> Now from that one more image is created.

--> from this running one more container.

--> so on that it’s running third instruction so like this it will continue on layers

--> in dockerhub alredy have layers

--> **docker push shan2324/user:v1 --** see here layer already exist it’s coming.

--> stack of bricks -- search in google

--> if you move down one it will effect on above layers

--> frequently changes you can keep last

-->

**user/Dockerfile**

FROM node:20

WORKDIR /opt/server

ENV MONGO="true" \

REDIS\_URL="redis://redis:6379" \

MONGO\_URL="mongodb://mongodb:27017/users"

COPY package.json .

COPY \*.js .

RUN npm install

CMD ["node","server.js"]

--> Push the code and pull

--> **docker push shan2324/user:v1 --** docker already exist.

--> finally all intermediate container will delete.

--> in every layer intermidate will delete then finally it will come one.

--> so that is final image.

--> so from this a image will create.

--> from this one container will create.

--> so these all layers

--> Now this layers I can put on top

--> push the code and pull

* **DOCKER\_BUILDKIT=0 docker build -t shan2324/user:v1 --no-cache .**

--> changes from copy \*.js this only will push.

--> layear optimiztion - in layear optimization frequently changing instructions you keep it at the end of the docker file. So that previous layears it should not be distrub.

-->

-->

**user/Dockerfile**

FROM node:20

WORKDIR /opt/server

ENV MONGO="true"

ENV REDIS\_URL="redis://redis:6379"

ENV MONGO\_URL="mongodb://mongodb:27017/users"

COPY package.json .

COPY \*.js .

RUN npm install

CMD ["node","server.js"]

--> push and pull

**DOCKER\_BUILDKIT=0 docker build -t shan2324/user:v1 --no-cache .**

--> it was created 3 extra layears.

--> while creationg 3 extra layes it will take tome for buidling memory also it will increase.

**catalogue/Dockerfile**

FROM node:20-alpine3.21 AS builder

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

FROM node:20-alpine3.21

RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

ENV MONGO="true"

ENV MONGO\_URL="mongodb://mongodb:27017/catalogue"

WORKDIR /opt/server

USER roboshop

COPY --from=builder /opt/server /opt/server

CMD ["node","server.js"]

# FROM node:20-alpine3.21

# RUN addgroup -S roboshop && adduser -S roboshop -G roboshop

# WORKDIR /opt/server

# COPY package.json .

# COPY \*.js .

# RUN npm install

# RUN chown -R roboshop:roboshop /opt/server

# ENV MONGO="true" \

# MONGO\_URL="mongodb://mongodb:27017/catalogue"

# USER roboshop

# CMD ["node","server.js"]

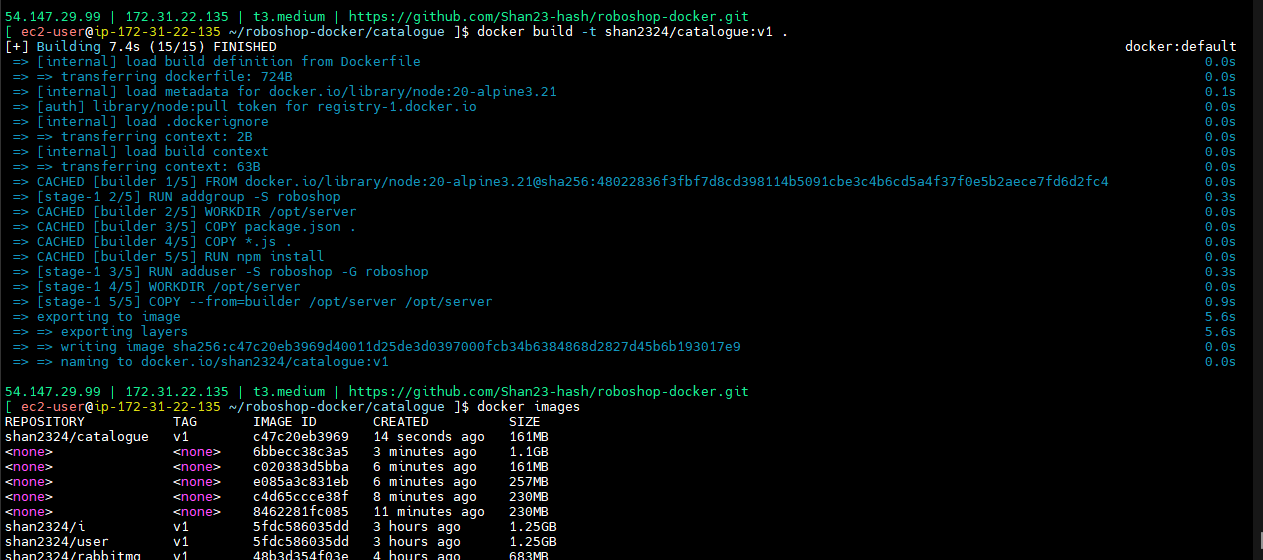
--> I written 1 more ENV

-->**docker images**

--> **cd ../catalogue**

--> **docker build -t shan2324/catalogue:v1 .**

**docker images --** see size -- 161MB



**catalogue/Dockerfile**

FROM node:20-alpine3.21 AS builder

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

FROM node:20-alpine3.21

RUN addgroup -S roboshop

RUN adduser -S roboshop -G roboshop

ENV MONGO="true"

ENV MONGO\_URL="mongodb://mongodb:27017/catalogue"

WORKDIR /opt/server

USER roboshop

COPY --from=builder /opt/server /opt/server

CMD ["node","server.js"]

--> Push and pull the code

**docker build -t shan2324/catalogue:v1 .**

**--> docker images --**  see the size there no change but it’s increasing layers.

Docker best practices -- check in google

--> Combine multiple run instructions into single instruction into to optimize the layers.

**user/Dockerfile**

FROM node:20

WORKDIR /opt/server

ENV MONGO="true"

ENV REDIS\_URL="redis://redis:6379"

ENV MONGO\_URL="mongodb://mongodb:27017/users"

COPY package.json .

COPY \*.js .

RUN npm install

RUN apt-get update

RUN apt-get install net-tools git

RUN apt-get clean

CMD ["node","server.js"]

**DOCKER\_BUILDKIT=0 docker build -t shan2324/user:v1 --no-cache .**

--> getting error focus on only image build.

--> **docker build -t shan2324/user:v1 .**

**--> docker images --** see the size user size is 1.27GB

--> if you need to optimize you have to club multiple instructions.

--> cache also you can clean.

**user/Dockerfile**

FROM node:20

WORKDIR /opt/server

ENV MONGO="true"

ENV REDIS\_URL="redis://redis:6379"

ENV MONGO\_URL="mongodb://mongodb:27017/users"

COPY package.json .

COPY \*.js .

RUN npm install

RUN apt-get update -y \

&& apt-get install net-tools git -y \

&& apt-get clean

CMD ["node","server.js"]

--> **docker build -t shan2324/user:v1 .**

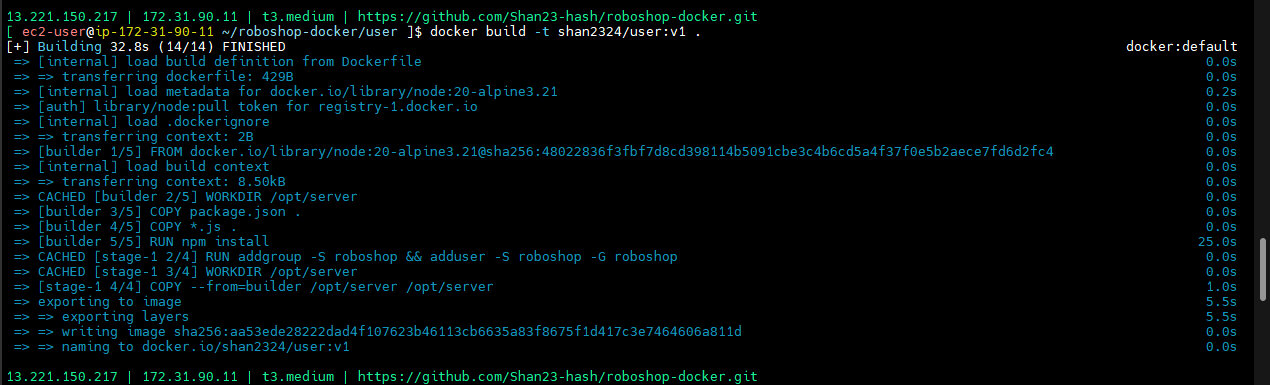
**-->** space is not there that’s why getting error.

--> **docker compose down**

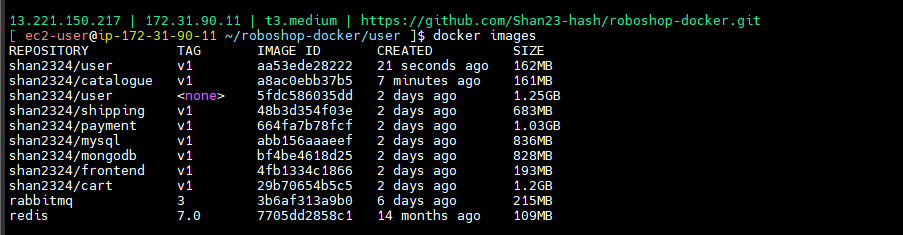
--> **docker rmi -t shan2324/user:v1**

**--> cd user**

**--> docker build -t shan2324/user:v1 .**



**--> docker images**



**-->** Combine run instructions where possible to minimize the number of image layers, which can improve build time and reduce image size.

--> Prefer COPY over ADD:

COPY is generally preferred for copying local files as it is more explicit and predictable , ADD has additional functionalities like URL support and automatic archive extraction, which may not always desired.

**Interview**

**-->**  Recently achievements.

--> I did docker image optimization.

--> I decreased image size.

--> in some places we are using root containers that one I changed to non root containers.

--> I did layers optimization.

--> I implemented multistage builds.

--> some where it’s using ADD that one I changed to COPY.

--> if you have unwanted files please keep it .dockerfile then docker it will not read that one.