**SESSION - 56**

--> **What is the best practices in docker? (\*\*\*\*\*\*interview question\*\*\*\*\*\*)**

* minimum and official images
* multi stage builds
* optimising layers, combining RUN instructions
* non root containers
* used customised networks
* implementing volumes
* COPY over ADD
* docker ignore not to load everything into docker
* implementing health checks
* limiting resources
* getting secrets from secrets manager
* implementing volumes

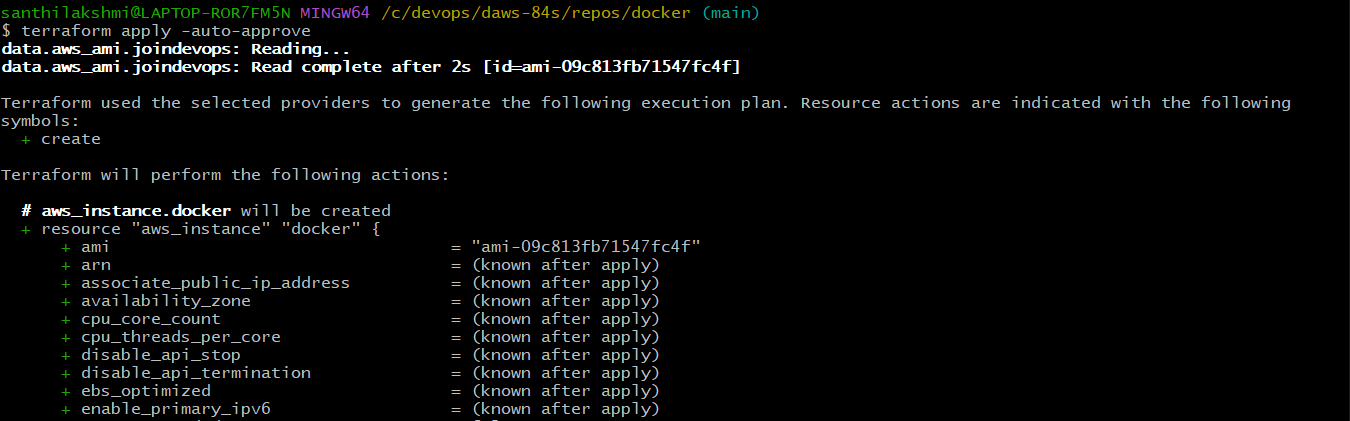
--> frequently changes you have to keep in down.

--> same like we can care our application will be fast.

--> milli second differnce also we have take care.

--> cd /c/devops/daws-84s/repos/docker

--> terraform apply -auto-approve

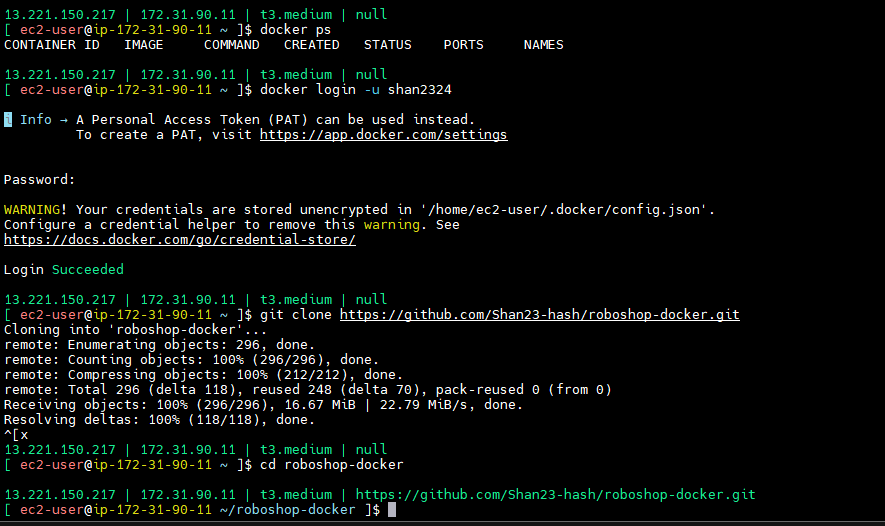


**--> sudo usermod -aG docker ec2-user**

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

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

**--> cd roboshop-docker**



**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" \

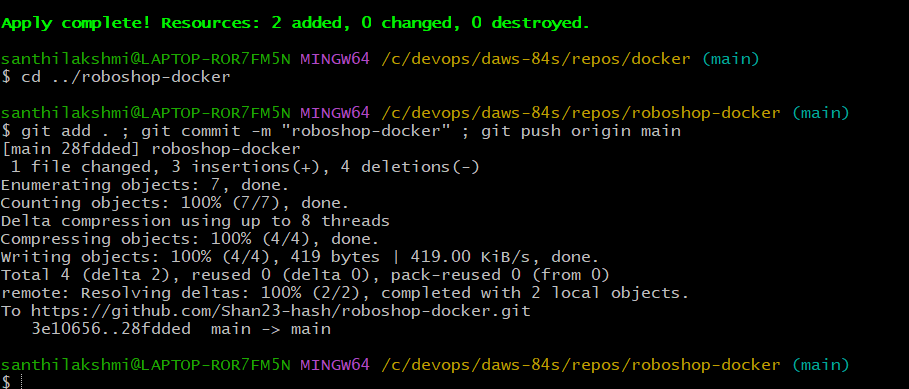
# MONGO\_URL="mongodb://mongodb:27017/catalogue"

# USER roboshop

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

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

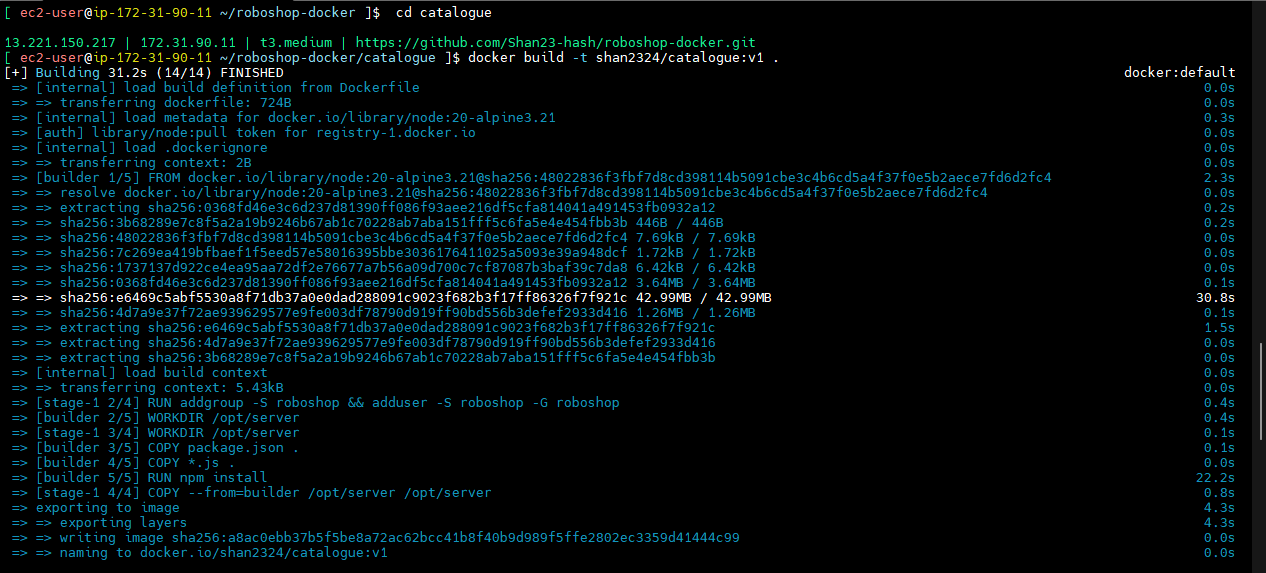
**--> git add . ; git commit -m "roboshop-docker" ; git push origin main**



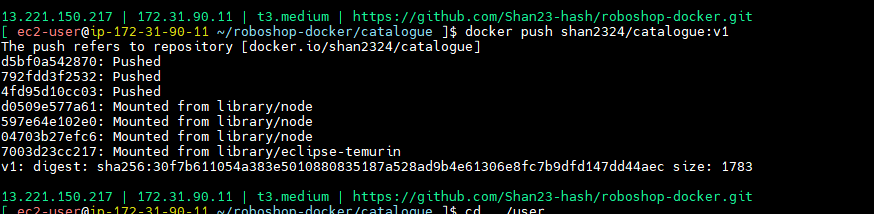
**--> git pull**

**--> cd /catalogue**

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

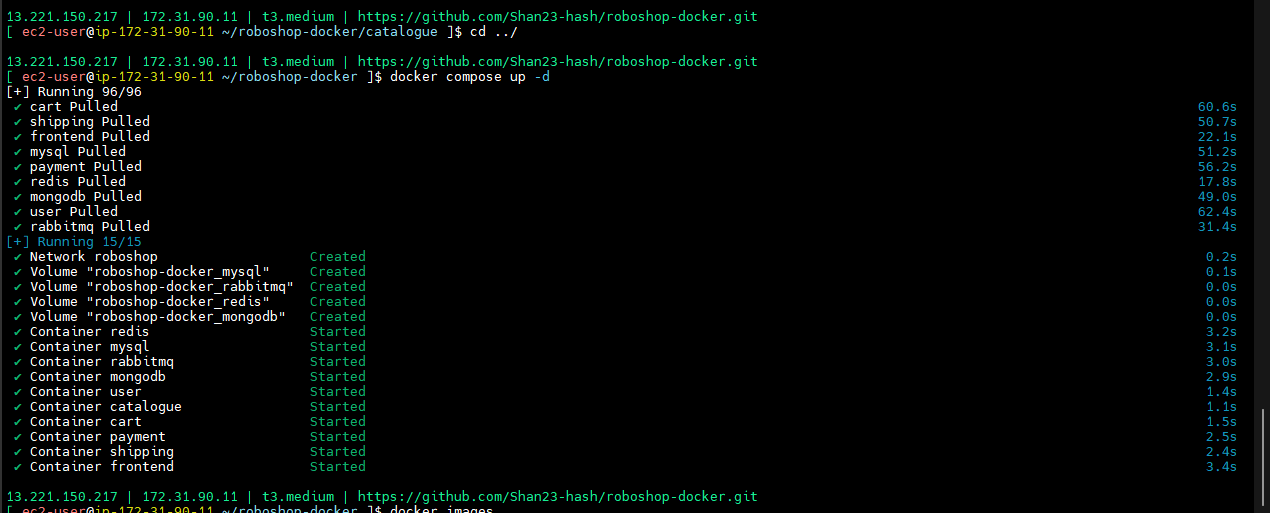


**--> docker push shan2324/catalogue:v1**



**--> cd ../**

**--> docker compose up -d**



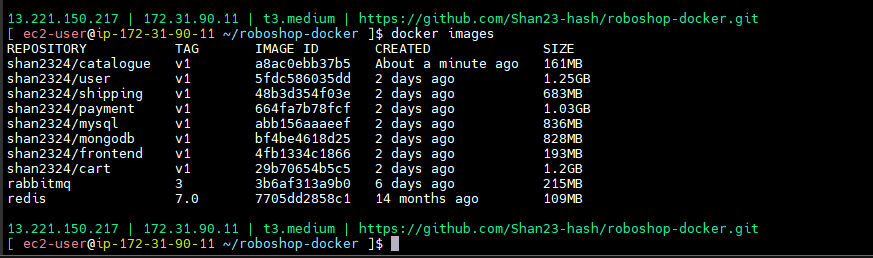
**-->** in locally images not there so it will from docker.

--> why I’m building user name means no need to tag.

--> if you are not given username need to run tag command

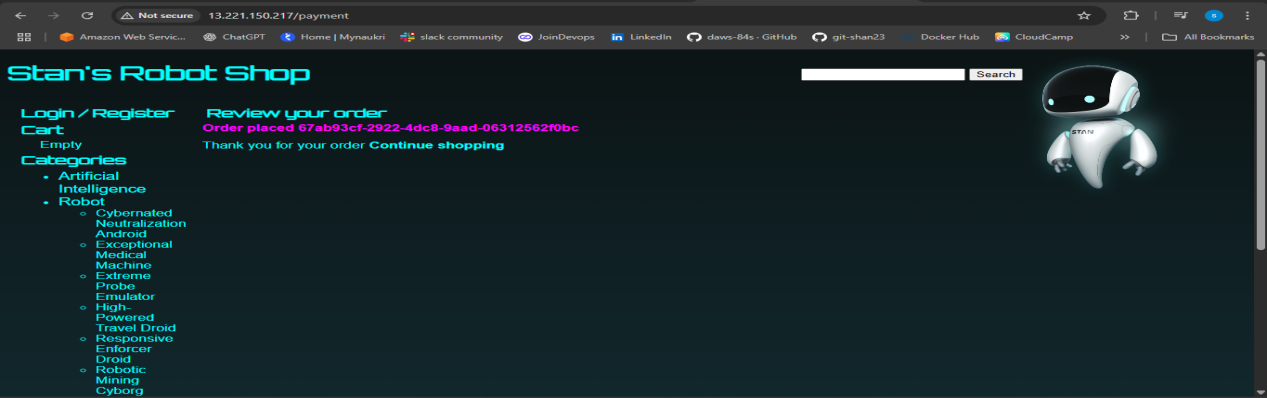
**--> docker ps**

**--> docker images**



**-->**  Now we can access our application.

--> try to order the product.



--> for all node js products also same. Environment only will change.

**user/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\_URL="mongodb://mongodb:27017/users" \

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

MONGO=true

WORKDIR /opt/server

USER roboshop

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

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

# 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"]

**cart/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 REDIS\_HOST="redis" \

CATALOGUE\_HOST="catalogue" \

CATALOGUE\_PORT="8080"

WORKDIR /opt/server

USER roboshop

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

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

# FROM node:20

# WORKDIR /opt/server

# COPY package.json .

# COPY \*.js .

# RUN npm install

# ENV REDIS\_HOST="redis" \

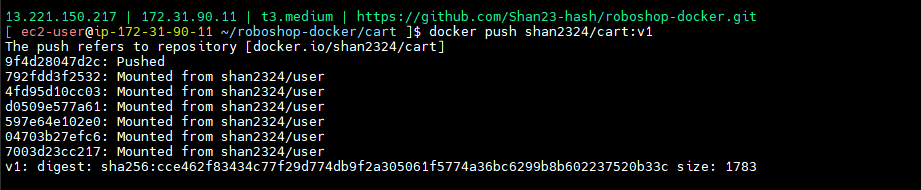
# CATALOGUE\_HOST="catalogue" \

# CATALOGUE\_PORT="8080"

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

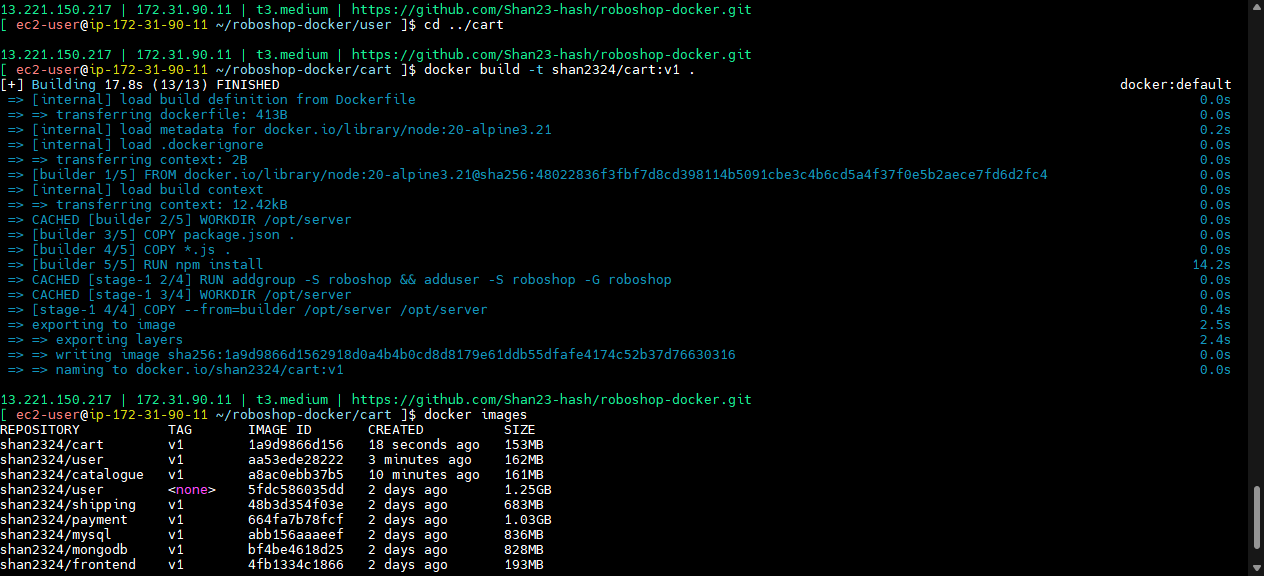
--> Push and pull the code.

--> **for i in cart catalogue ; do cd $i; docker build -t shan2324/$I:v1 . ; docker push shan2324/$i:v1 . ; cd .. ; done**



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

**--> docker images**

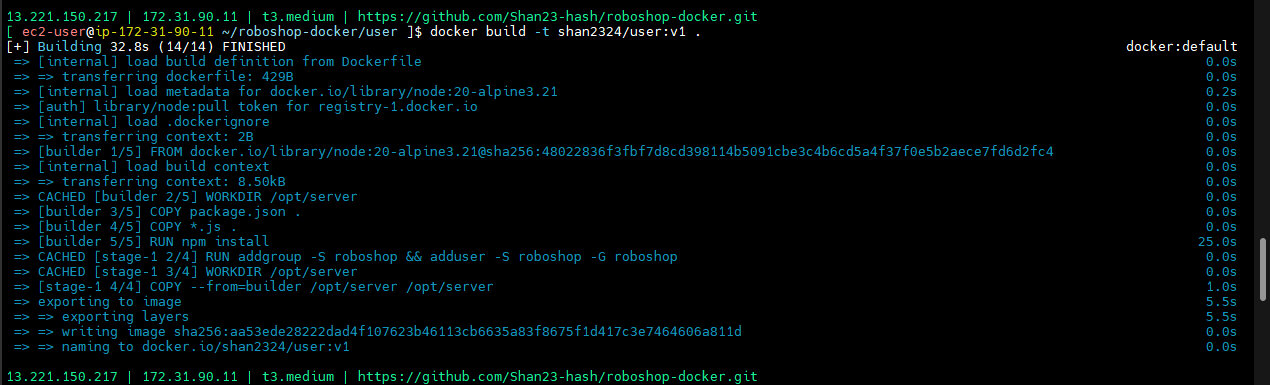


--> catalogue size 153MB.

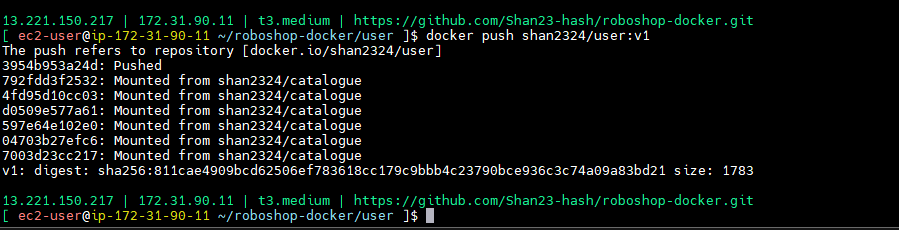
--> cart 162MB.

**--> cd ../user**

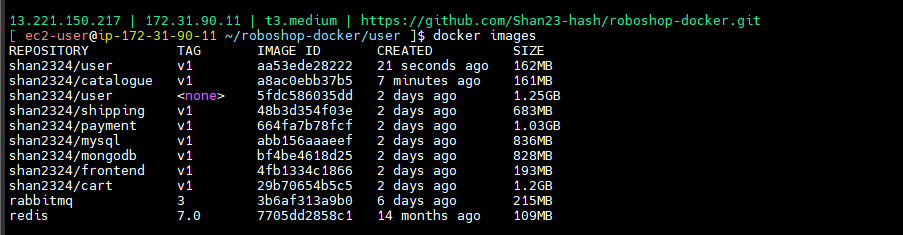
**--> docker build -t shan2324/user:v1 .**



**--> docker push shan2324/user:v1**

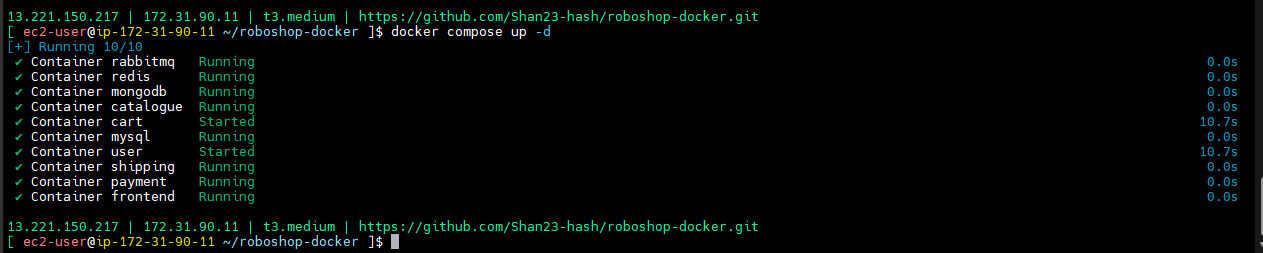


**--> docker images**



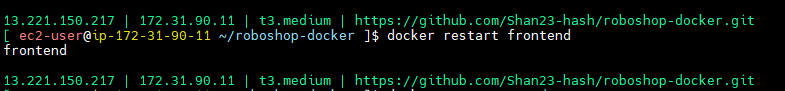
--> see the size - 162MB

--> **docker compose up -d**



--> every time you have to test it.

--> **docker restart frontend**



--> frontend need to frontend.

--> Next shipping. Multistage builds it will use for java.

**Interview ans**

--> Multistage builds my peoples are implemented but only for java project they are implemented. But they don’t it will use for node js also. I implemented node js also.

--> During npm install, unnecessary cached files and unused packages were removed, which helped reduce the project size by approximately 100MB.

--> optimization layear I did.

Source code --> compile --> bytecode (intermidate code)

--> java is special after source we have to compile then we will get byte code. This is compared to human readable programming language then high and machine code than low.

--> intermediate code : no need to run in operating system before if self they complied so that is to be speed. Because already compilation completed but node js during running it will happen compilation.

--> for source code compilation we java development kit

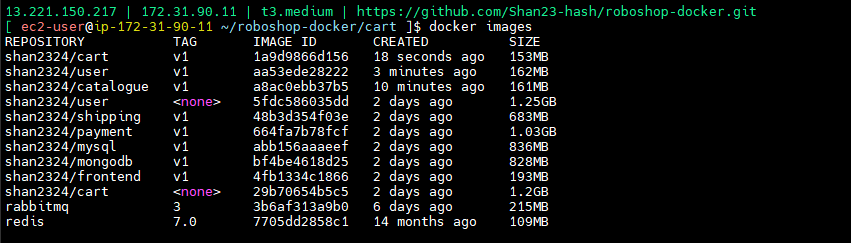
--> here everything with maven compilation, maven life cycle that and all we discussed.

--> so now in multistage buids how much I got shipping

--> JDK - java development kit (maven)

-->JRE -- runs byte code (no need of development environment.)

--> **docker images -** SIZE is 683MB



--> in 683MB java also there,maven JDK also there.

--> for running no need maven -- only for building and compilation we need here.

--> For that memory will save high.

**shipping/Dockerfile**

FROM maven

WORKDIR /opt/server

COPY pom.xml .

COPY src /opt/server/src

RUN mvn clean package \

--> no need environment.

--> this is as buider

--> after build here and we did maven clean package so target directory one will create in that shipping.jar will get.

--> that jar we have to put in runner.

--> Go to docker hub for JRE image.

--> type JRE

--> JRE docker images ( check in google)

--> in jre alpine is better.

--> JRE alpine docker image -- search in google.

--> actually in java many changes happened.

--> changed from open source to licence.

--> eclipse-temurin:17-jre-alpine -- search in dockerhub

**shipping/Dockerfile**

FROM maven AS builder

WORKDIR /opt/server

COPY pom.xml .

COPY src /opt/server/src

RUN mvn clean package

FROM eclipse-temurin:17-jre-alpine

EXPOSE 8080

WORKDIR /opt/server

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

ENV CART\_ENDPOINT="cart:8080" \

DB\_HOST="mysql"

COPY --from=builder /opt/server/target/shipping-\*.jar /opt/server/shipping.jar

USER roboshop

CMD ["java","-jar","shipping.jar"]

# FROM maven

# WORKDIR /opt/server

# COPY pom.xml .

# COPY src /opt/server/src

# RUN mvn clean package

# RUN mv /opt/server/target/shipping-1.0.jar /opt/server/shipping.jar

# ENV CART\_ENDPOINT="cart:8080" \

# DB\_HOST="mysql"

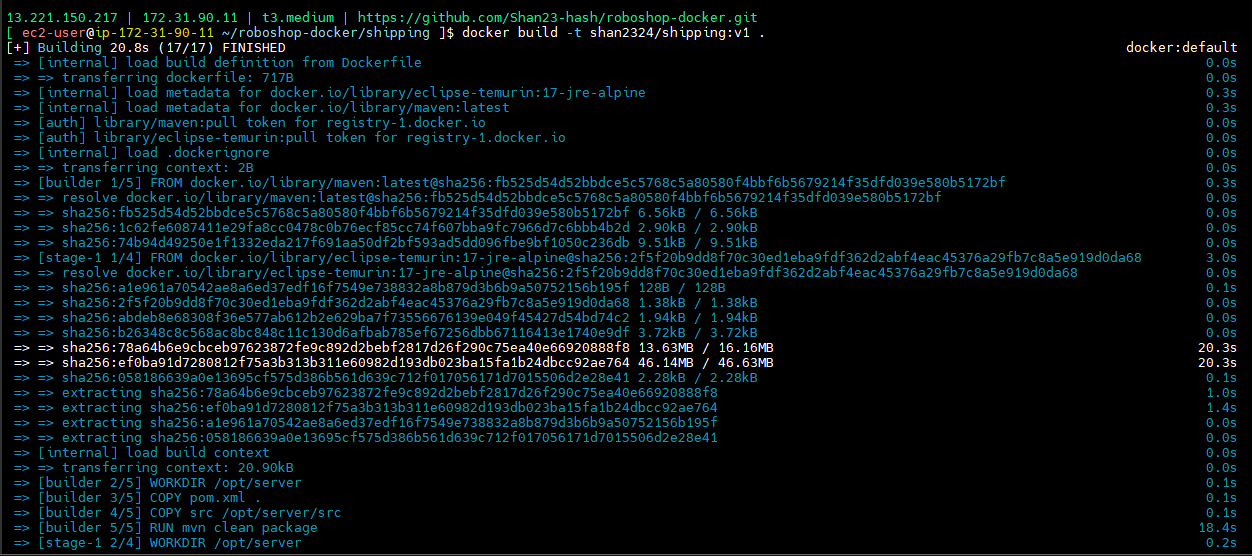
# CMD ["java","-jar","shipping.jar"]

--> set environment.

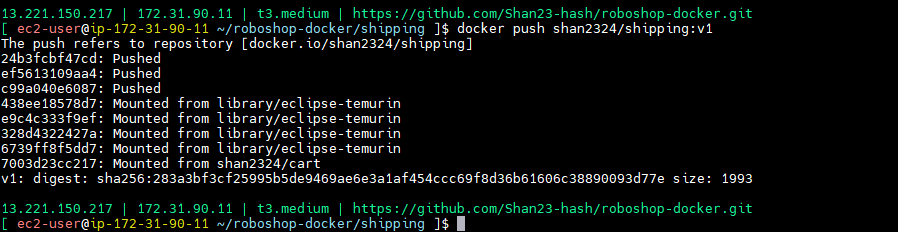
--> push and pull the code.

--> **cd ../shipping**

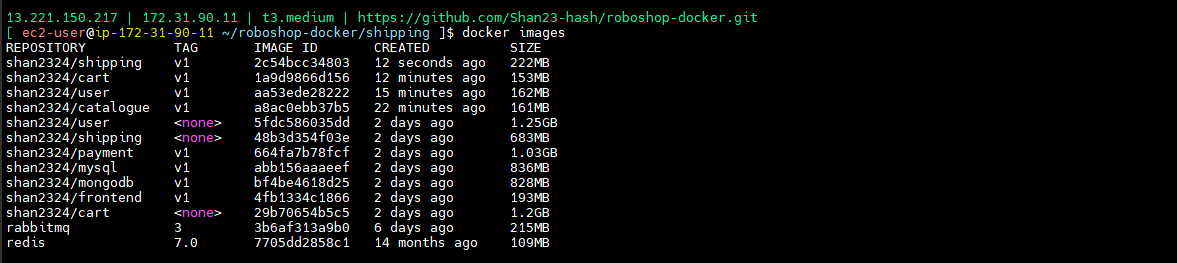
**docker build -t shan2324/shipping:v1 .**



**docker push shan2324/shipping:v1**



**--> docker images --** 224MB --> java memory it will time.



--> try to order the space.

--> shipping is working

**PAYMENT**

**-->** if we selected all the points need to install all the packages.

--> in docker hub type python

**payment/Dockerfile**

FROM python:3.9 AS builder

WORKDIR /opt/server

COPY requirements.txt .

COPY \*.py .

COPY payment.ini .

RUN pip3 install -r requirements.txt

FROM python:3.9.23-alpine3.22

EXPOSE 8080

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

WORKDIR /opt/server

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

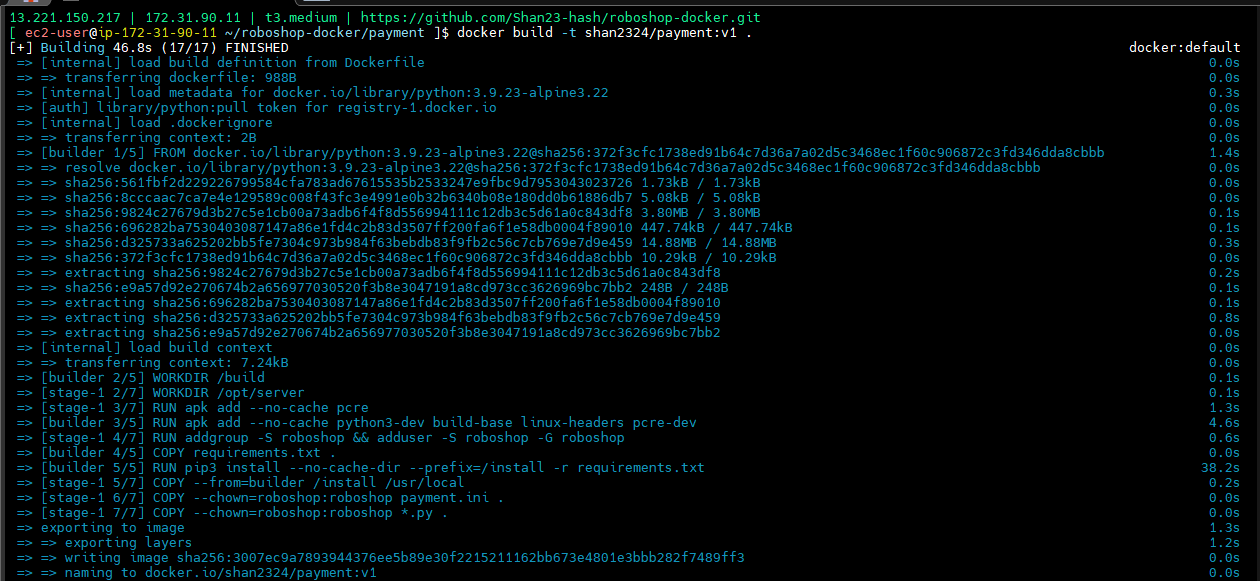
USER roboshop

CMD ["uwsgi", "--ini", "payment.ini"]

--> Push and pull the code

--> **cd ../payment**

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



**--> docker push shan2324/payment:v1**

**--> docker images --** SIZE is 49MB

--> Python is system programming going forward. There is difference between application programming and system programming.

--> system programming means like C launges.

--> **docker compose up -d**

--> got error

--> **docker run -it shan2324/payment:v1 --** not connecting

--> **payment/Dockerfile**

FROM python:3.9 AS builder

WORKDIR /opt/server

COPY requirements.txt .

COPY \*.py .

COPY payment.ini .

RUN pip3 install -r requirements.txt

FROM python:3.9.23-alpine3.22

EXPOSE 8080

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

WORKDIR /opt/server

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

USER roboshop

# CMD ["uwsgi", "--ini", "payment.ini"]

CMD [“sleep”,” 1000”]

--> **docker run -it shan2324/payment:v1 sh --** without push the code

--> **ls -l**

**--> uwsgi --ini payment.ini -**- run manually and see.

--> got error uwsgi not found

--> the system package actually -- python will install like system packages not like application dependencies.

--> so we will do small modification

--> because it’s installing a system module.

--> before user roboshop I need to run RUN pip3 install -r requirements.txt

**payment/Dockerfile**

FROM python:3.9 AS builder

WORKDIR /opt/server

COPY requirements.txt .

COPY \*.py .

COPY payment.ini .

RUN pip3 install -r requirements.txt

FROM python:3.9.23-alpine3.22

EXPOSE 8080

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

WORKDIR /opt/server

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

RUN pip3 install -r requirements.txt

USER roboshop

# CMD ["uwsgi", "--ini", "payment.ini"]

CMD [“sleep”,” 1000”]

--> push and pull the code

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

--> system package it will expect repos

--> we can add

--> present removing USER roboshop

--> push the and pull the code and see

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

**--> docker images -->**  SIZE is 418MB

--> docker compose up -d

--> see the output

--> ERROR placeing order so problem is inside payment.

**--> docker logs payment**

**--> docker logs -f frontend**

--> not set the environment

--> we are added in compose.yaml

--> docker restart frontend

--> still error

--> NAT gate way

--> sleep not removed

--> remove sleep

--> **docker build -t shan2324/payment:v1 --no-cache**

**--.> docker compose up -d**

**-->** try to order the product

--> now order was places

--> finally we have to check USER robodhop

**payment/Dockerfile**

FROM python:3.9 AS builder

WORKDIR /opt/server

COPY requirements.txt .

COPY \*.py .

COPY payment.ini .

RUN pip3 install -r requirements.txt

FROM python:3.9.23-alpine3.22

EXPOSE 8080

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

WORKDIR /opt/server

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

RUN apk add --no-cache python3-dev build-base linux-headers pcre-dev

RUN pip3 install -r requirements.txt

USER roboshop

CMD ["uwsgi", "--ini", "payment.ini"]

--> Push and pull the code

--> **cd ../payment**

--> **docker build -t shan2324/payment:v1 --no-cache .**

**-->** system package will install in OS.

--> application dependency -- app library used by code

--> no need multistage build.

--> cd ../

--> **docker compose up -d**

--> see output

--> not working if we are adding roboshop

--> **docker logs payment**

**-->** can not set jd as non root user error in log

--> python alpine image -- check in google

--> **docker run -it shan2324/payment:v1**

**--> docker ps**

**payment/Dockerfile**

FROM python:3.9.23-alpine3.22

EXPOSE 8080

WORKDIR /opt/server

COPY requirements.txt .

COPY \*.py .

COPY payment.ini .

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

&& chown -R roboshop:roboshop /opt/server

RUN apk add python3-dev build-base linux-headers pcre-dev

RUN pip3 install -r requirements.txt

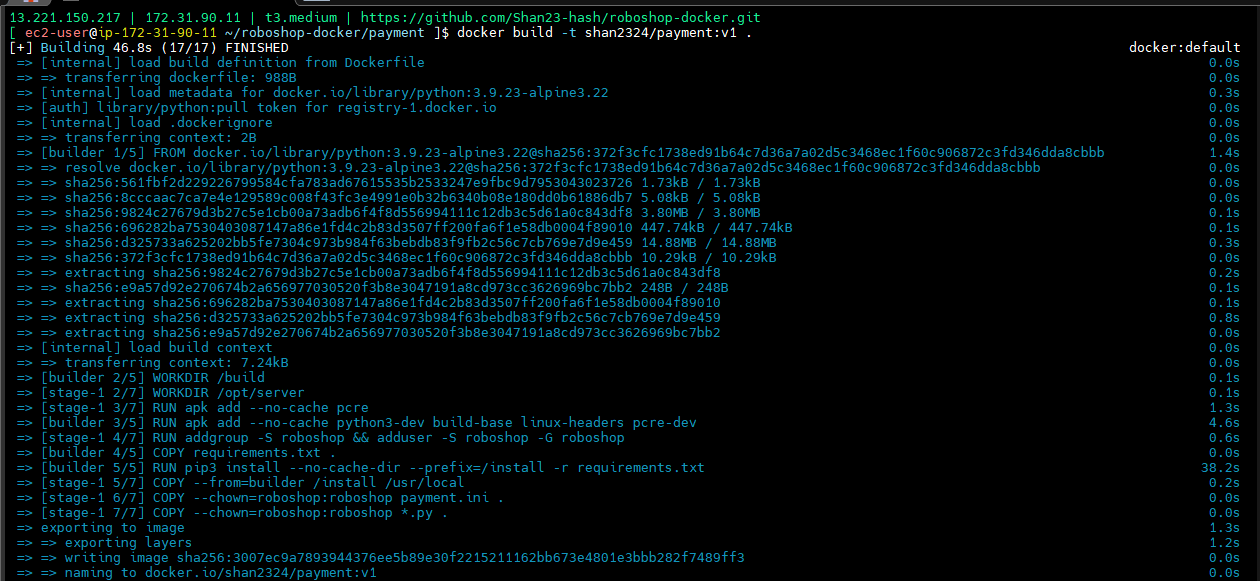
USER roboshop

CMD ["uwsgi", "--ini", "payment.ini"]

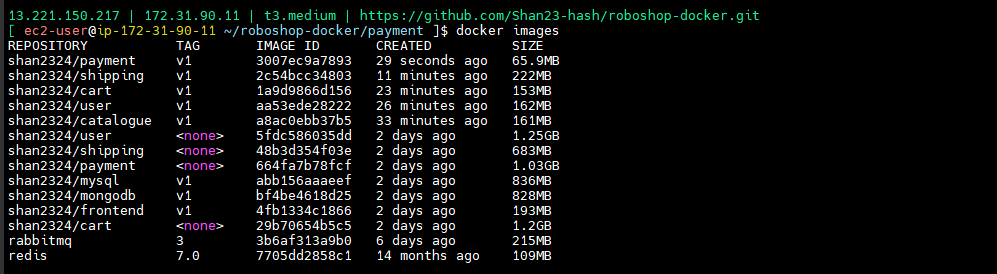
--> push and pull the code

--> **cd ../payment**

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



**--> docker images**



**--> docker compose up -d**

**-->** see the output

--> same error

**FRONTEND**

**-->** nginx is system package.

--> nginx folders many are there.

--> /var/log/nginx

--> /usr/share/nginx/html

--> /etc/nginx

**frontend/Dockerfile**

FROM nginx:1.29.0-alpine

RUN rm /etc/nginx/nginx.conf /etc/nginx/conf.d/default.conf

RUN mkdir -p /var/cache/nginx/client\_temp && \

mkdir -p /var/cache/nginx/proxy\_temp && \

mkdir -p /var/cache/nginx/fastcgi\_temp && \

mkdir -p /var/cache/nginx/uwsgi\_temp && \

mkdir -p /var/cache/nginx/scgi\_temp && \

chown -R nginx:nginx /var/cache/nginx && \

chown -R nginx:nginx /etc/nginx/ && \

chmod -R 755 /etc/nginx/ && \

chown -R nginx:nginx /var/log/nginx

RUN mkdir -p /etc/nginx/ssl/ && \

chown -R nginx:nginx /etc/nginx/ssl/ && \

chmod -R 755 /etc/nginx/ssl/

RUN touch /var/run/nginx.pid && \

chown -R nginx:nginx /var/run/nginx.pid /run/nginx.pid

ADD nginx.conf /etc/nginx/nginx.conf

ADD static /usr/share/nginx/html/

USER nginx

CMD ["nginx", "-g", "daemon off;"]

# FROM nginx

# RUN rm -rf /usr/share/nginx/html/index.html

# RUN rm -rf /etc/nginx/nginx.conf

# RUN rm -rf /etc/nginx/conf.d/default.conf

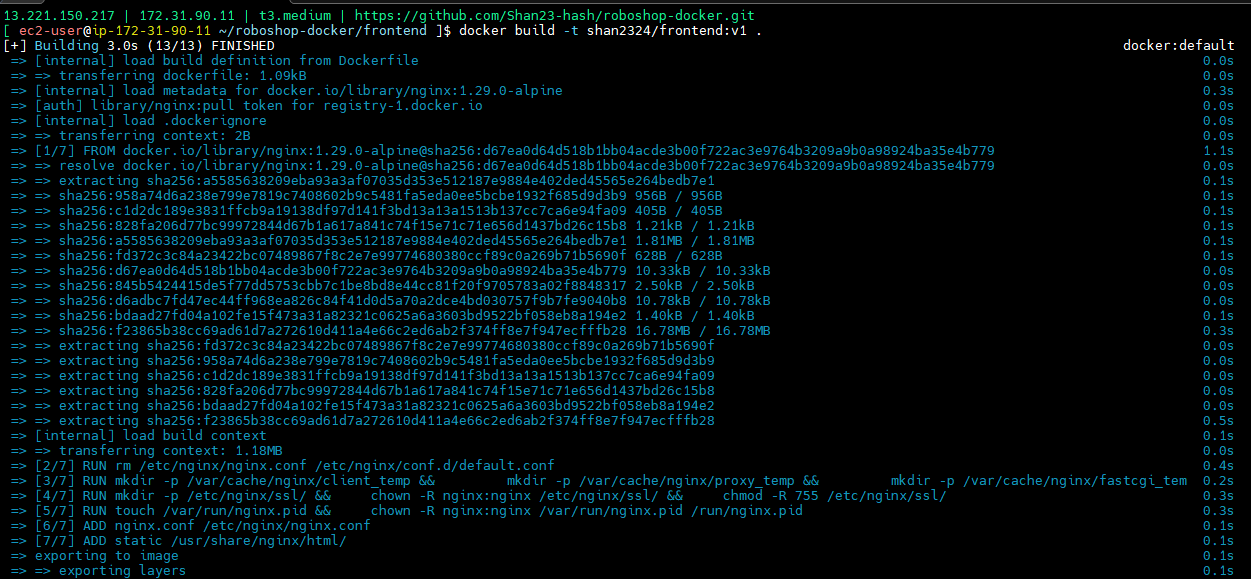
# ADD nginx.conf /etc/nginx/nginx.conf

# ADD static /usr/share/nginx/html/

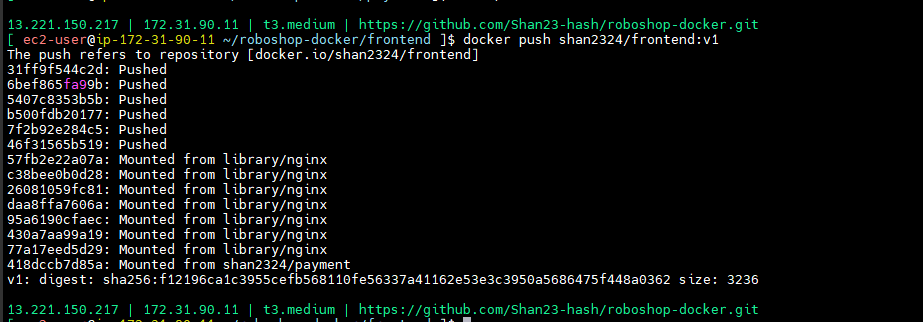
--> Push and pull the code

--> cd ../frontend

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

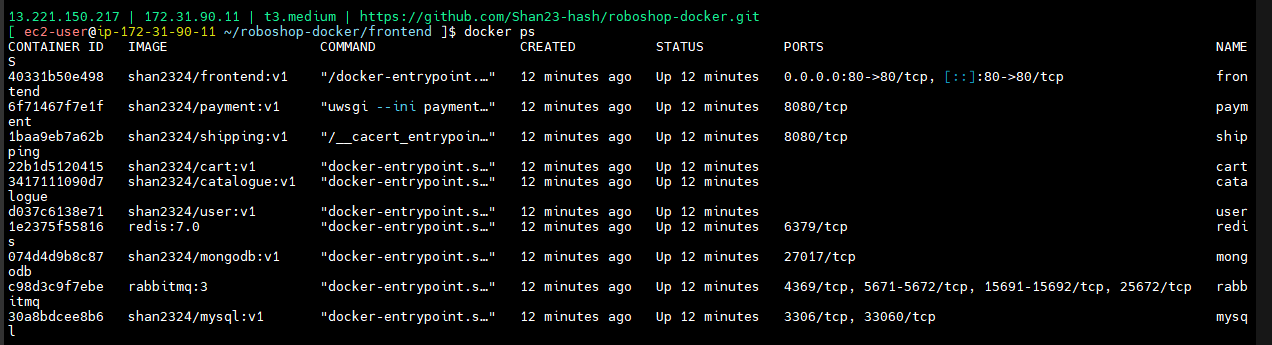


**--> docker push shan2324/frontend:v1**

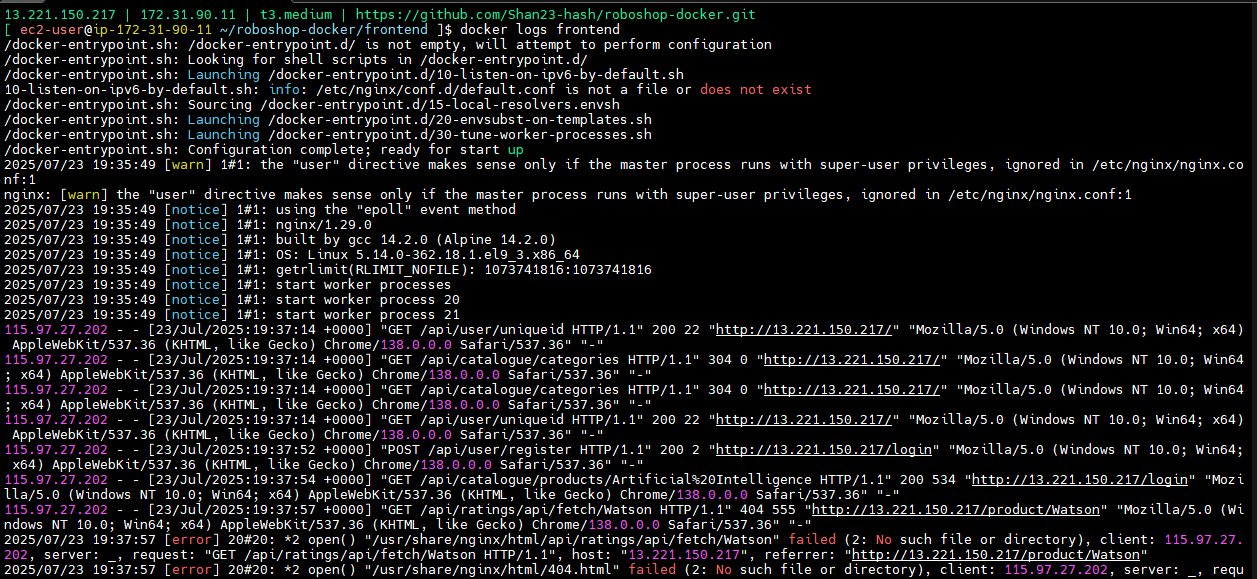


**--> docker compose up -d**

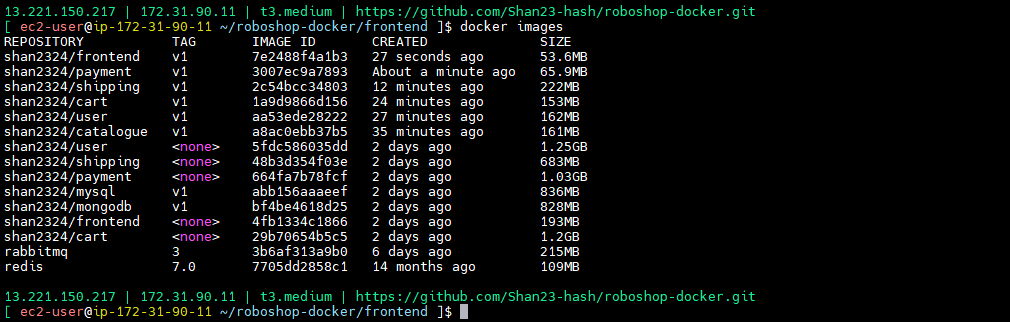
**--> docker ps**



**--> docker logs frontend**



**--> docker images --** SIZE 53.6MB



**PAYMENT**

**-->** Node js and java -- this projects needs libraries

--> images means base os + system packages + app run time + app libraries

**payment/Dockerfile**

# Stage 1: Build

FROM python:3.9.23-alpine3.22 AS builder

WORKDIR /build

# Install build dependencies

RUN apk add --no-cache python3-dev build-base linux-headers pcre-dev

COPY requirements.txt .

RUN pip3 install --no-cache-dir --prefix=/install -r requirements.txt

# Stage 2: Final image

FROM python:3.9.23-alpine3.22

EXPOSE 8080

WORKDIR /opt/server

# Runtime dependencies only

RUN apk add --no-cache pcre

# Create app user

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

USER roboshop

# Copy installed Python packages from builder

COPY --from=builder /install /usr/local

# Copy application code

COPY --chown=roboshop:roboshop payment.ini .

COPY --chown=roboshop:roboshop \*.py .

CMD ["uwsgi", "--ini", "payment.ini"]

#CMD ["sleep", "1000"]

# FROM python:3.9

# EXPOSE 8080

# WORKDIR /opt/server

# COPY requirements.txt .

# COPY \*.py .

# COPY payment.ini .

# RUN pip3 install -r requirements.txt

# CMD ["uwsgi", "--ini", "payment.ini"]

**payment/payment.ini**

[uwsgi]

wsgi-file = payment.py

callable = app

master = true

processes = 1

lazy-apps = true

enable-threads = true

socket = 0.0.0.0:8080

protocol = http

--> here system packages important

--> if that packages is there code will run

--> in node - node js modules, in java - targets like that will run.

--> that packages should be there in that OS.

--> this is new os, so definitely it won’t available. That’s created one directory for all packages installation that copy in os.

--> in multi stage builds -- copy installed python package from builder.

--> while copying this one I copied like roboshop.

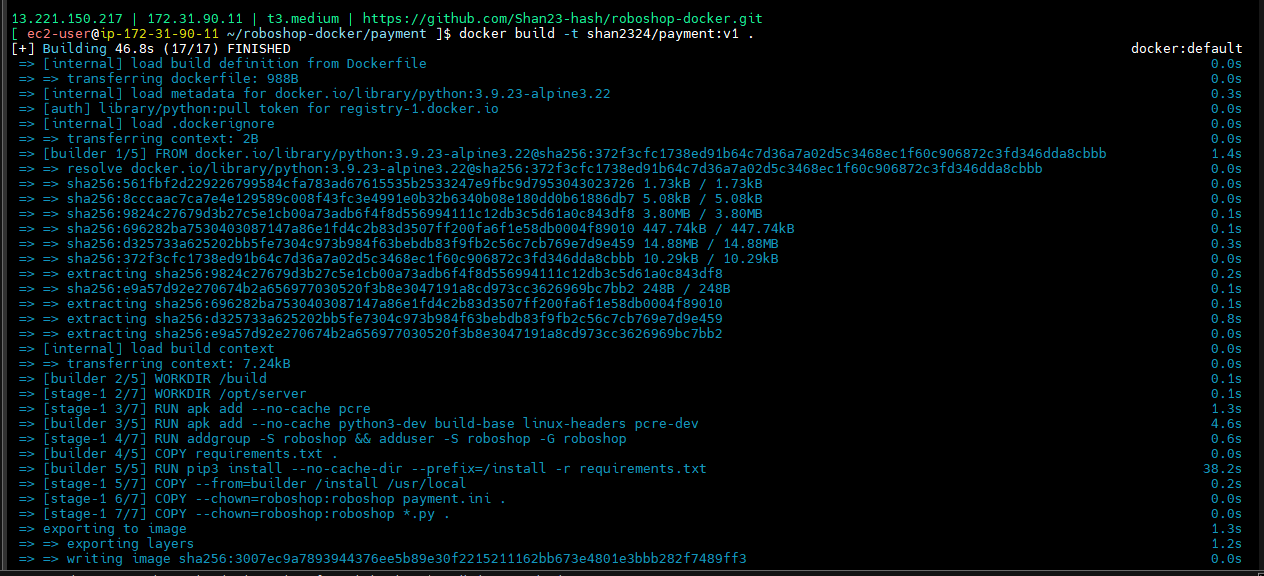
--> Ownership

--> Push and pull the code

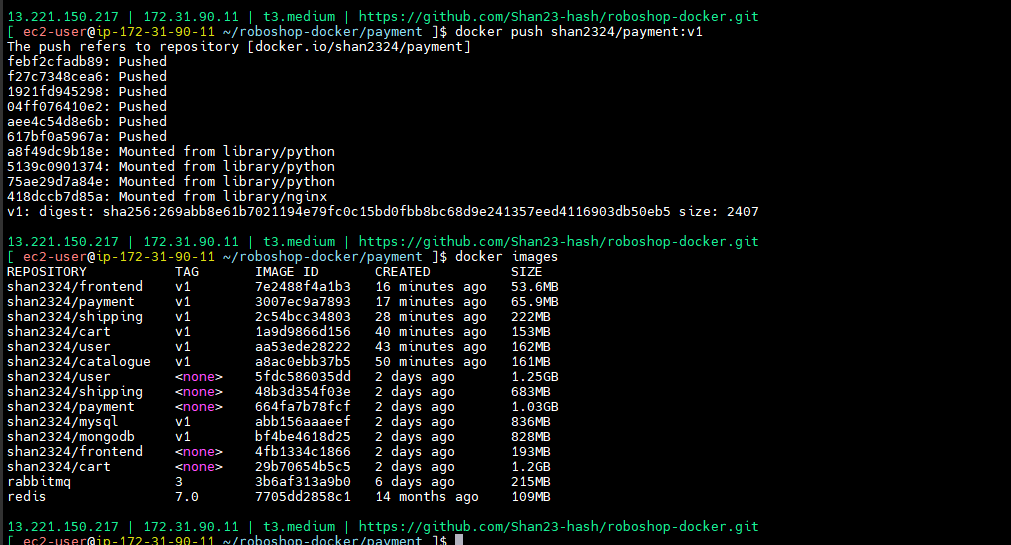
--> **cd roboshop-docker**

--> **cd ../payment**

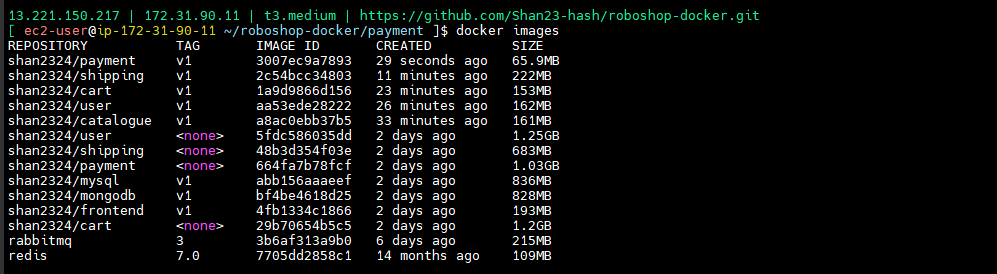
**--> docker build -t shan2324/payment:v1 .**



**--> docker push shan2324/payment:v1**

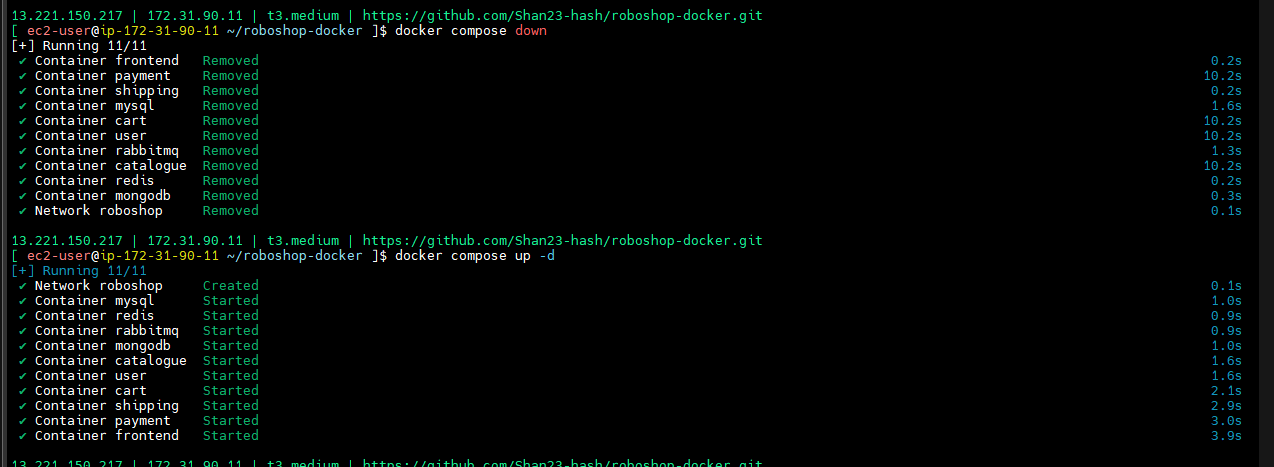


**--> docker images --** SIZE is 65.9MB



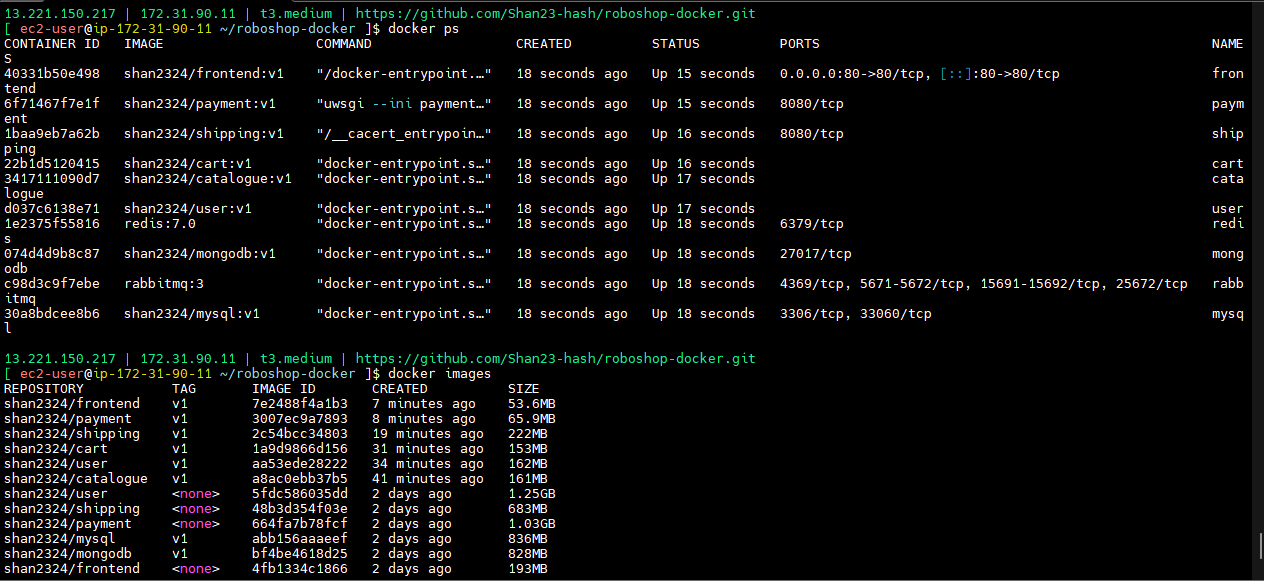
**--> docker compose down**

**--> docker compose up -d**

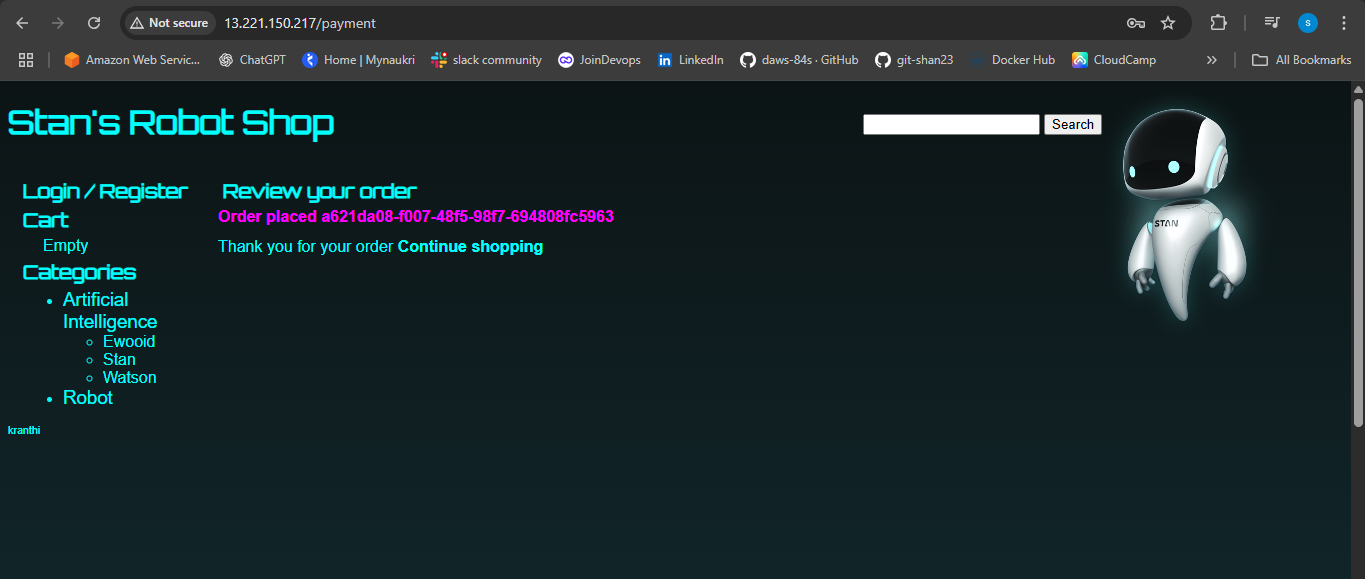


**--> docker ps**

**--> docker images**



**-->** Check Output



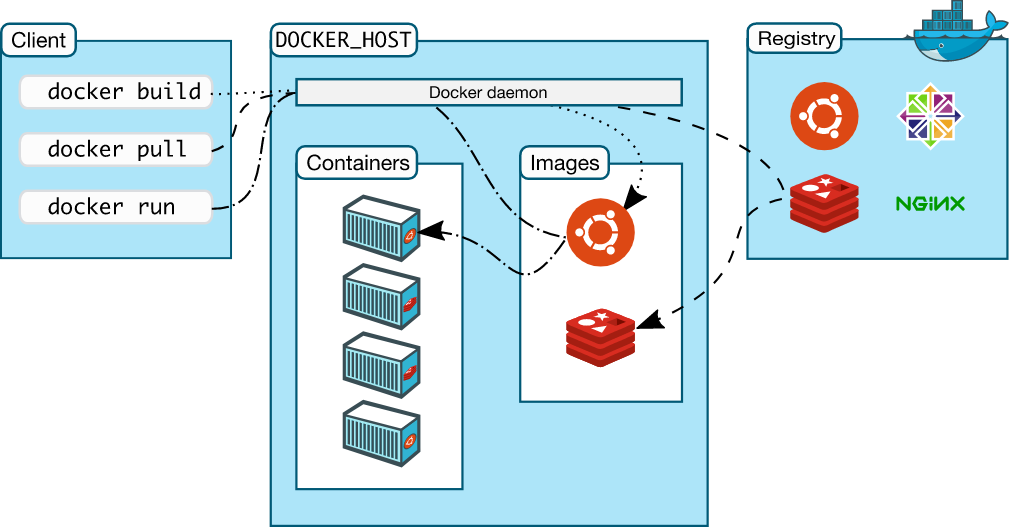
--> build the image

--> run the image

--> any image or containerization

**docker architecture**

Docker utilizes a client-server architecture to manage containers. The Docker daemon, a background service, handles container operations like building, running, and distributing them. The Docker client, which can be a command-line interface (CLI) or a graphical user interface (GUI), communicates with the daemon to execute these actions.



**-->** Client means -- docker command line (CLI)

--> docker run docker build these all docker client

--> Host -- where docker is running, docker deamon ( continuously running)

--> like ssh services, nginx services , while systemctl running it will run continuously.

--> if it’s running you connect them.

--> here also start docker -- after that it will run continuously

--> we can run command upto docker is running.

--> so there is docker deamon

--> docker repo and central repo

--> 1. first it checks image is in local or not

--> 2 if exists, then it will run the container.

--> 3.if not exist, full from hub create container and send o/p to client.

--> if you run any docker command that one it will reach to docker services that one will whether images is there or not.if it is not there it will take from registry and pulled in images that one run will create like containers that is docker architecture.

--> compounds we have -- docker volume, docker network, images , containers, storage.

--> 4. there are docker volumes and networking we can configure.

Types of networks:

1. host --> docker uses host networking

--> **docker run -d --network host nginx --** port no 80 it will open

I’m running a container -p it won’t work in host, there is bridge between host and container we are using host so -p it won’t.

2. bridge(default) -> docker has its own networking

Bridge means docker will create a interface from that one it will allocated to containers. Docker has it’s have an own networking.

**Docker Disadvantages**

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

1. auto scaling --> no default auto scaling methods

--> it’s increasing docker I given docker run command, created one more container.

1. load balancing --> no load balancing components to balance traffic b/w multiple containers.

--> container by mistake crashes

1. reliability --> if container crashes, it will not restart on its own(self healing).
2. what if docker host crashes?

If docker containers crashes It will done all containers.

5. what about storage ? if docker host crashes we loose data also, because docker is managing in the same host.

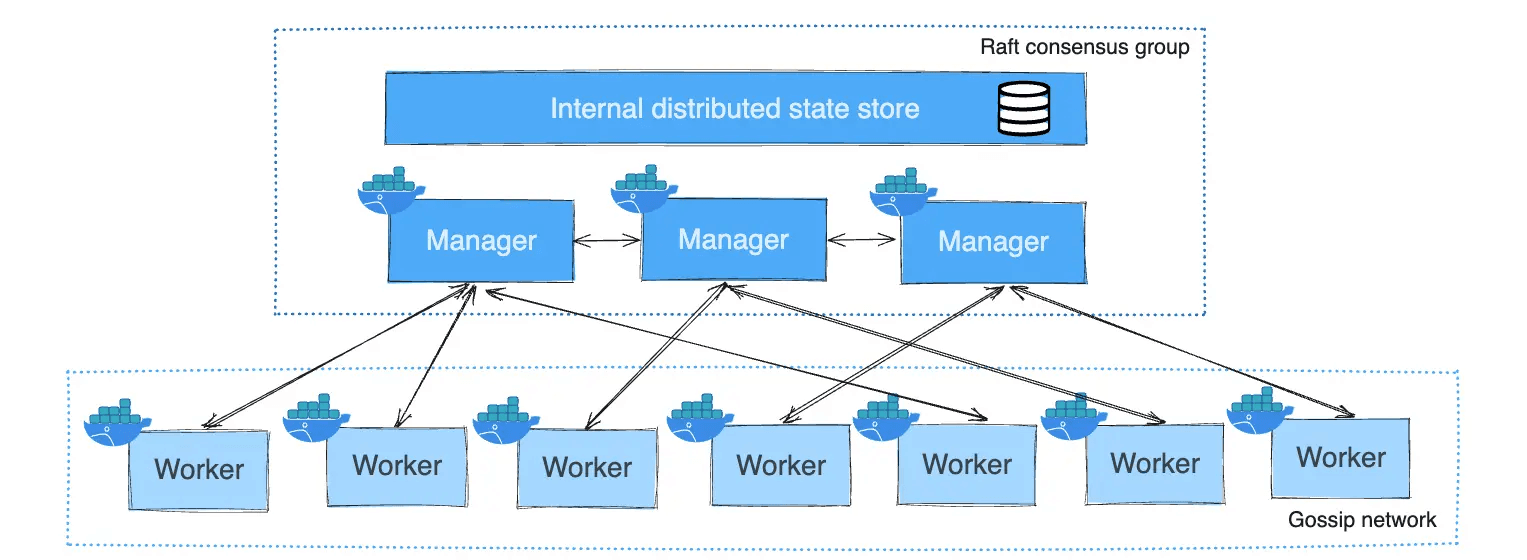
6. networking is only bridge mode, if you have multiple docker hosts bridge networking will not work.

**container orchestrator**

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

--> Docker host individually they can capable, run containers then can create networks but there should be some one interact of the docker to manage them that is orchestrator.

This is paid software from docker



--> this will manage for multiple docker images.

--> docker forms will manage that is not stable.