**SESSION - 54**

Containersation menas select one base image and use run instance install or configure whatever you want. then copy code from local to copy in contanirazation or image. Then finally run command run instruction.

--> catalogue,mongodb,

--> redis -- no need to do specially configuration so directly pulling image.

--> rabbitmq and redis like that only

--> cart,user,shipping -- upto here we did last class

**Diagram**

**Networking**

--> Who will provide ip address for instance? -- AWS

--> As an ISP internet service provider.

--> for instance that is server in one rack will be there.

--> in that server will conncte ethernet.

--> that ethernet coming from AWS ISP. So they will provide the ip address.

--> in docker we have two types of netwoking

1. Host Networking
2. Bridge

--> bridge default -- for docker will work like bridge will provide ip address for components and will provide networking.

--> exatly like modum.

--> if you giving request means first will go to catalogue then modum from modum to it will go mongodb.

--> it is the bridge between everyone here.

--> This a virtual interface.

--> in docker we have by default bridhe network.

--> if we used default bridge network. We can’t connect.

--> docker encrygies to create our network.

**docker network ls**

name is bridge, Driver also bridge.

Yestarday we are created

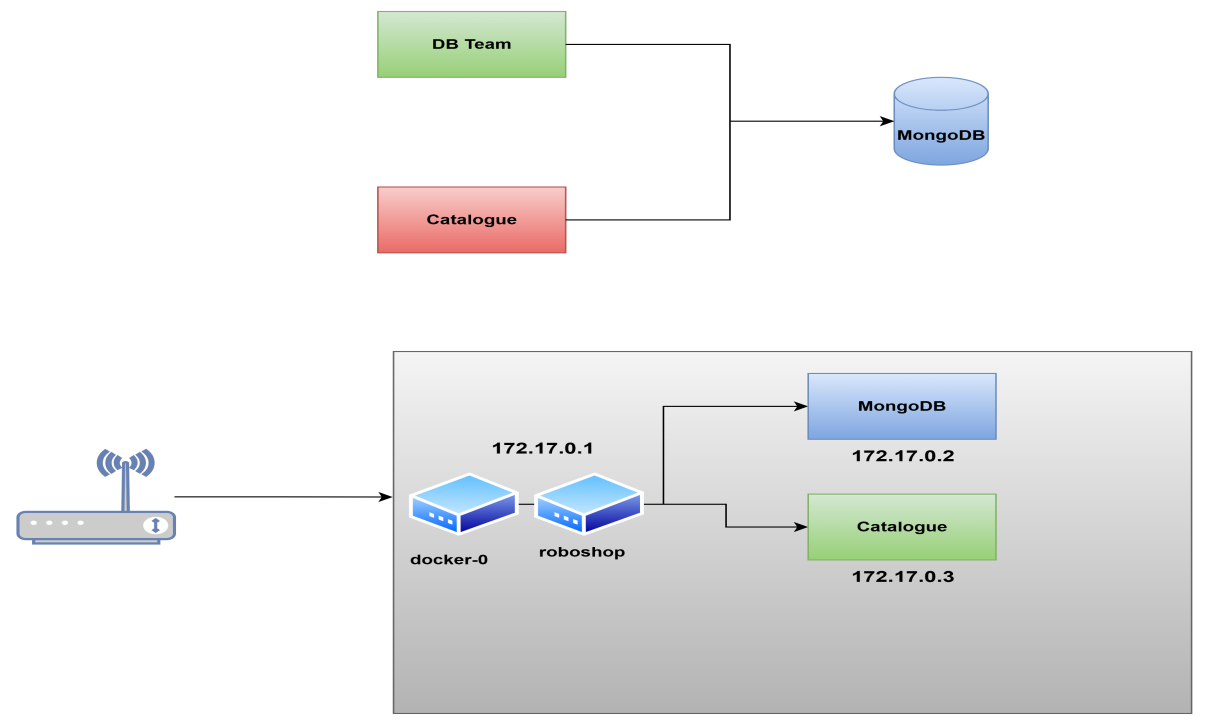
**docker network create roboshop**

**docker network ls**

--> Our network name is roboshop. That is bridhe type.

--> Now we are connecting roboshop network.

--> so this will become roboshop.

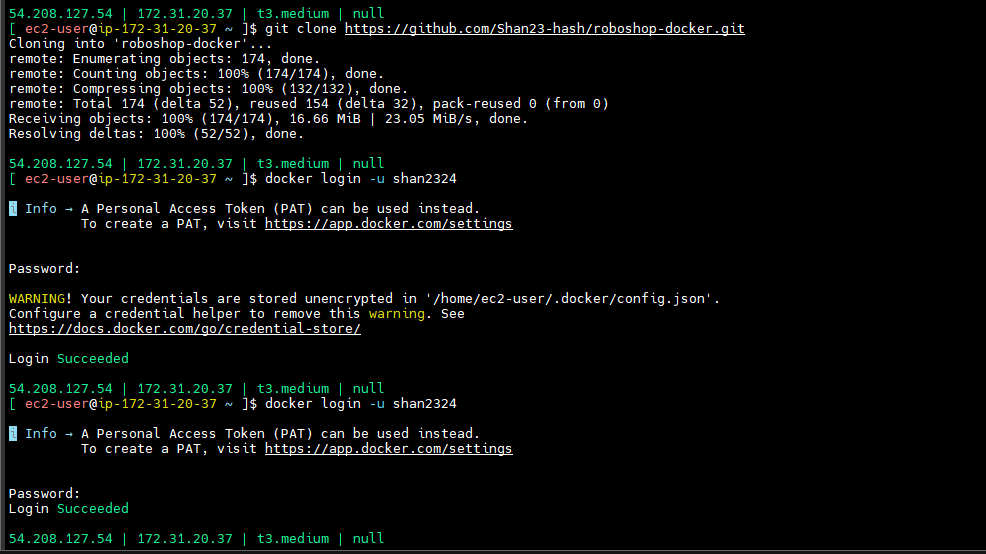


--> Now let us build the images

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

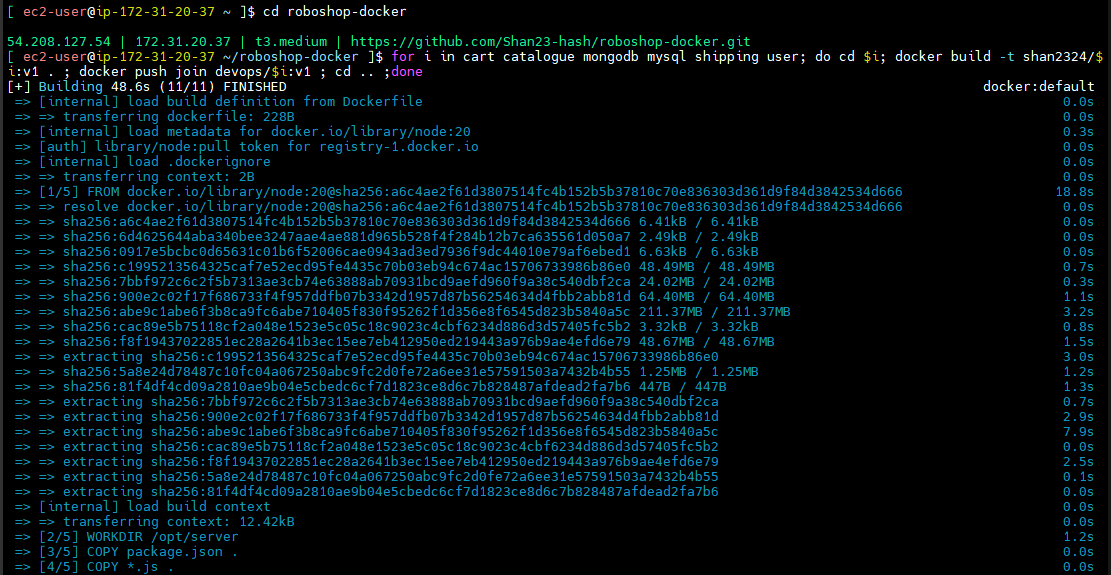
--> **sudo usermod -aG docker ec2-user**

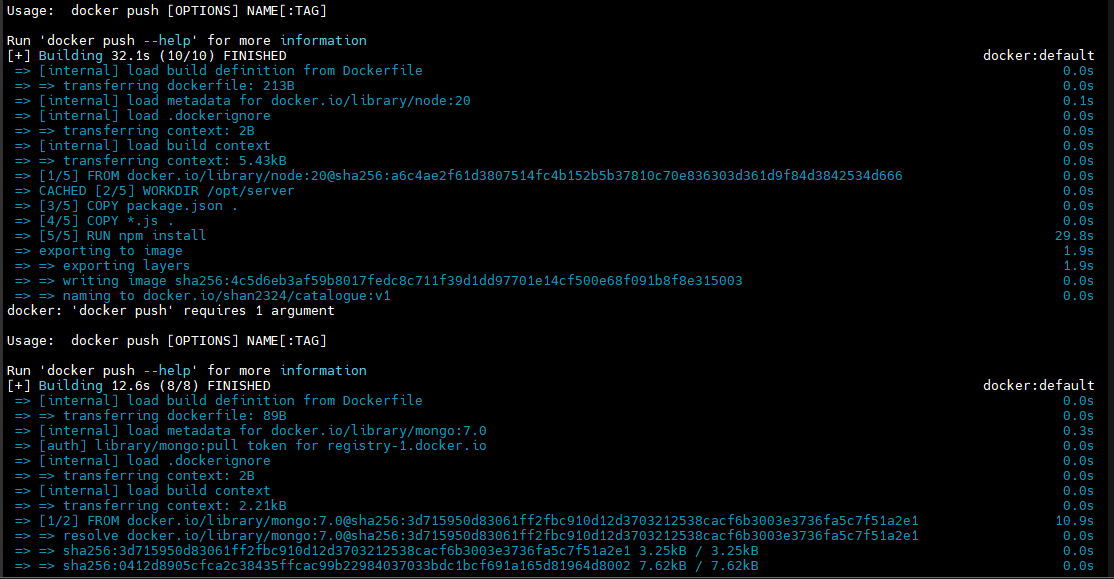
--> **docker login -u shan2324 (For roboshop login)**

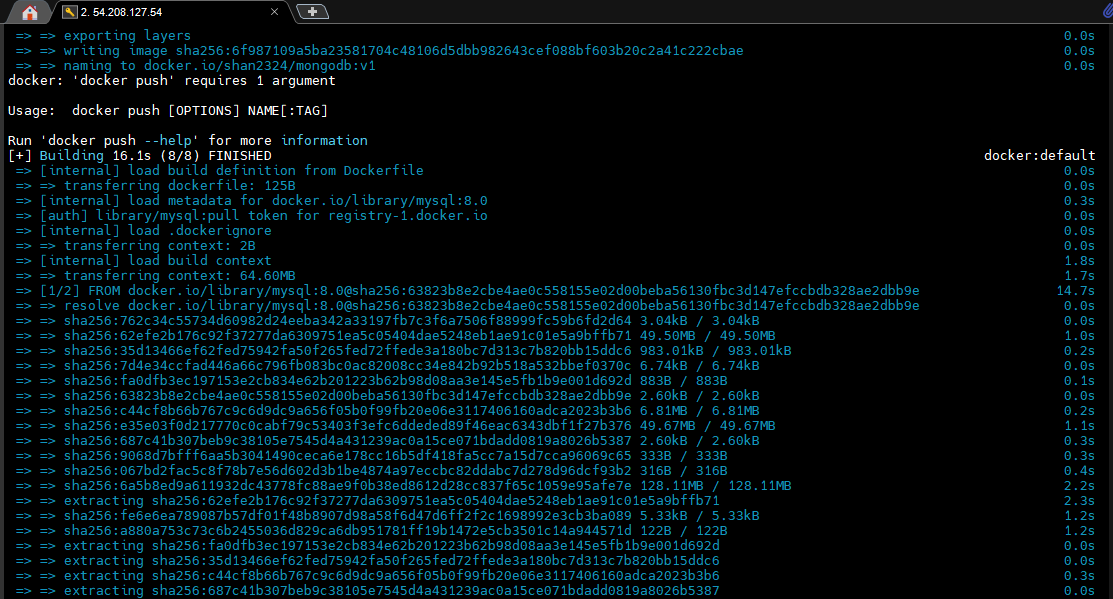


**--> ~/roboshop-docker**

--> **for i in cart catalogue mongodb mysql shipping user; do cd $i; docker build -t shan2324/$i:v1 . ; docker push join devops/$i:v1 ; cd .. ;done**







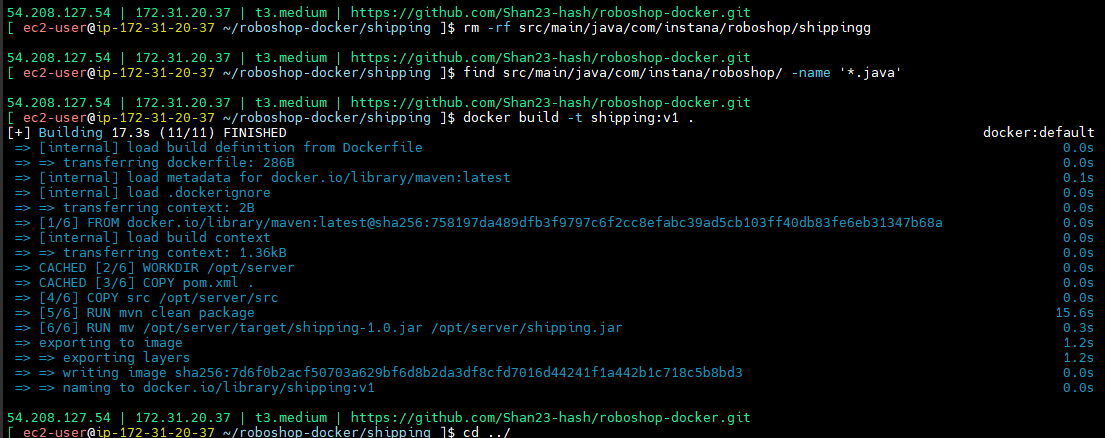
i got error in shipping

Instead of shipping it’s taking shippingg

**rm -rf src/main/java/com/instana/roboshop/shippingg**

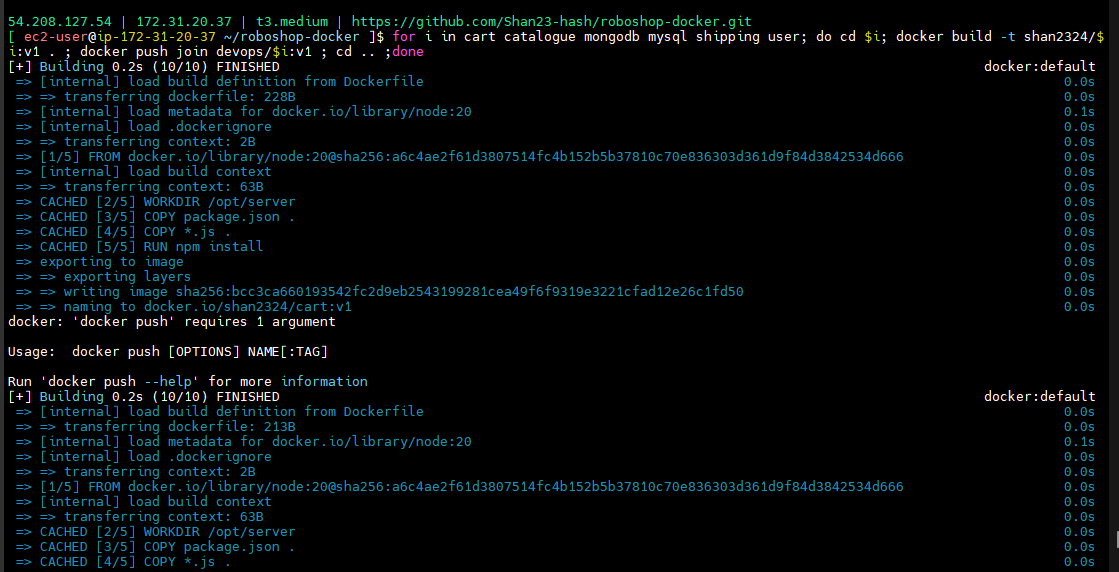
**find src/main/java/com/instana/roboshop/ -name '\*.java'**

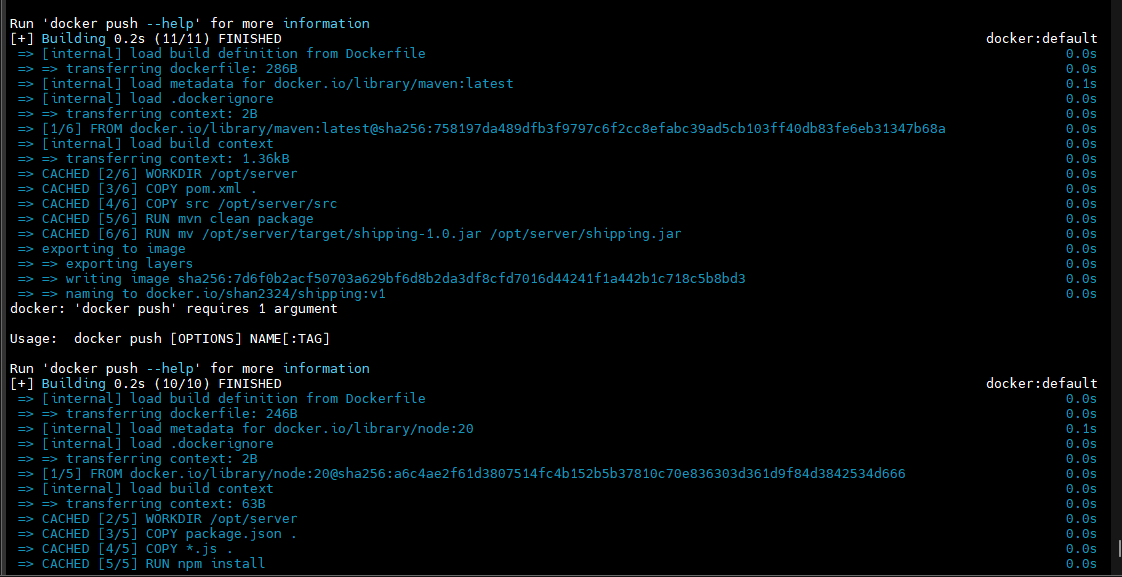
**docker build -t shipping:v1 .**



cd ~/roboshop-docker

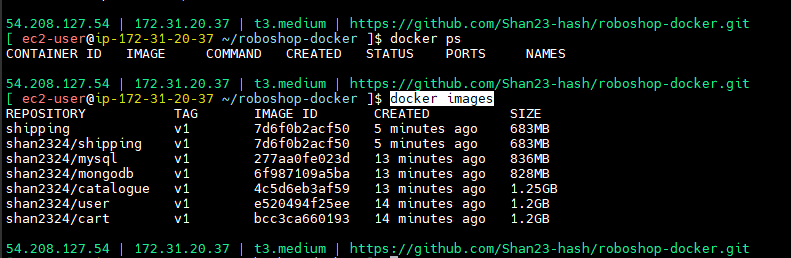
**for i in cart catalogue mongodb mysql shipping user; do cd $i; docker build -t shan2324/$i:v1 . ; docker push join devops/$i:v1 ; cd .. ;done**





**docker ps**

**docker images**



**Docker compose**

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

**Now we will discuss docker compose:**

Docker compose a component tool so where you defined all your docker containers as services

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Compose works in all environments - production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

* **Start, stop, and rebuild services**
* **View the status of running services**
* **Stream the log output of running services**
* **Run a one-off command on a service**

--> I’m bulding images then running images

--> catalogue depends on mangodb

--> cart depends on catalogue

--> need to create by order

--> Everything should be return that is called githubs.

--> evrything should be maintain inside gits.

--> docker compose is a command line to manage multi container applications, we can define all the containers as services, create dependencies between them, start time at once, stop them at once.

--> docker run -d --name mongodb --network roboshop joindevops/mongodb:v1 -- this is running command.

--> I will run every container like this.

--> here docker images like this is there

--> **docker images**

--> in ansible I wtitten like this command.

--> ansible -i catalogue.daws84s.site, -m dnf -a “name=nginx state=installed”

--> In ansible we are added playbook for adoap commands same like that we can’t run continuous images so for that we creating one that is docker-compose.yaml

**docker-compose.yaml**

networks:

default:

driver: bridge

name: roboshop

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

services:

mongodb:

image: joindevops/mongodb:v1

container\_name: mongodb

catalogue:

image: joindevops/catalogue:v1

container\_name: catalogue

depends\_on:

- mongodb

redis:

image: redis:7.0

container\_name: redis

user:

image: joindevops/user:v1

container\_name: user

depends\_on:

- redis

- mongodb

cart:

image: joindevops/cart:v1

container\_name: cart

depends\_on:

- catalogue

- redis

mysql:

image: joindevops/mysql:v1

container\_name: mysql

shipping:

image: joindevops/shipping:v1

container\_name: shipping

depends\_on:

- mysql

- cart

--> container name in docker compose ( search in google)

--> create network and mention default network

--> driver -- bridge

--> name -- roboshop

--> alredy I created environment so you have to keep yes.

--> If you've manually created a bridge network outside of Compose using the **docker network create** command, you can connect your Compose services to it by marking the network as **external**. -- true means alredy I created.

--> if incase I removed network

--> **docker network remove roboshop**

--> this code execution time docker will create network roboshop and to make it as default for all services inside this yaml file.

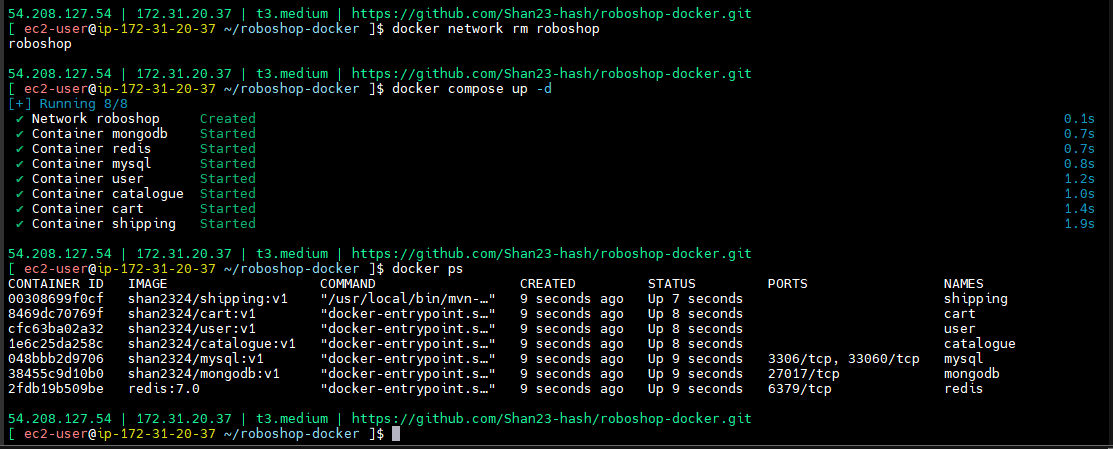
--> catalogue image

--> you can create 1 or many.

**docker network rm roboshop**

**docker compose up -d**

**docker ps**



Up - all imge will update.

-d - we are sending into the background

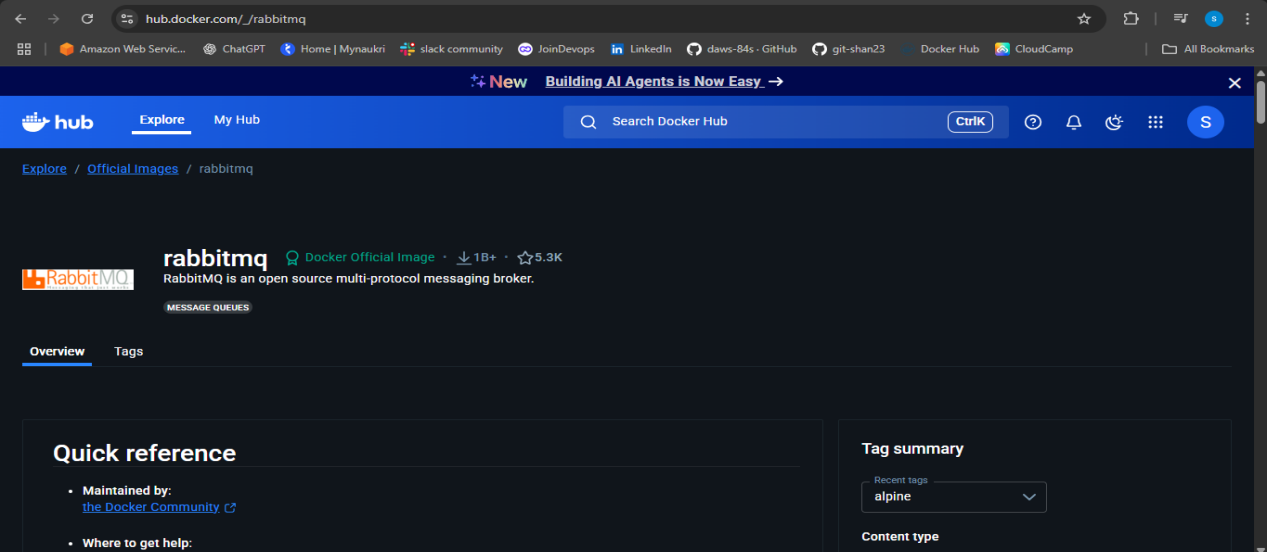
--> docker ps = all are runnung

**RABBITMQ**

**-->**  For rabbitmq we are not loading data.

--> I don’t require create docker filr for that.rabbit mq vesrion is 3.

--> rabbirmq offial image.



* Setting default user and password
* If you wish to change the default username and password of guest / guest, you can do so with the RABBITMQ\_DEFAULT\_USER and RABBITMQ\_DEFAULT\_PASS environmental variables. These variables were available previously in the docker-specific entrypoint shell script but are now available in RabbitMQ directly.
* $ docker run -d --hostname my-rabbit --name some-rabbit -e RABBITMQ\_DEFAULT\_USER=user -e RABBITMQ\_DEFAULT\_PASS=password rabbitmq:3-management
* You can then go to http://localhost:8080 or http://host-ip:8080 in a browser and use user/password to gain access to the management console
* Setting default vhost
* If you wish to change the default vhost, you can do so with the RABBITMQ\_DEFAULT\_VHOST environmental variables:
* $ docker run -d --hostname my-rabbit --name some-rabbit -e RABBITMQ\_DEFAULT\_VHOST=my\_vhost rabbitmq:3-management

--> Need to create a one user rabbitmq user and password.

--> id you set the environment, rabbitmq default user and rabbit mq default password.

--> while creating container that one user and password will create.

--> Environment we are mentioning in docker file.

--> docker file or docker run - both you can use for run.

--> they are mentioned in docker run.

--> docker run they are mentioned means in docker compose also we can mention.

--> git push and pull

**docker-compose.yaml**

rabbitmq:

    image: rabbitmq:3

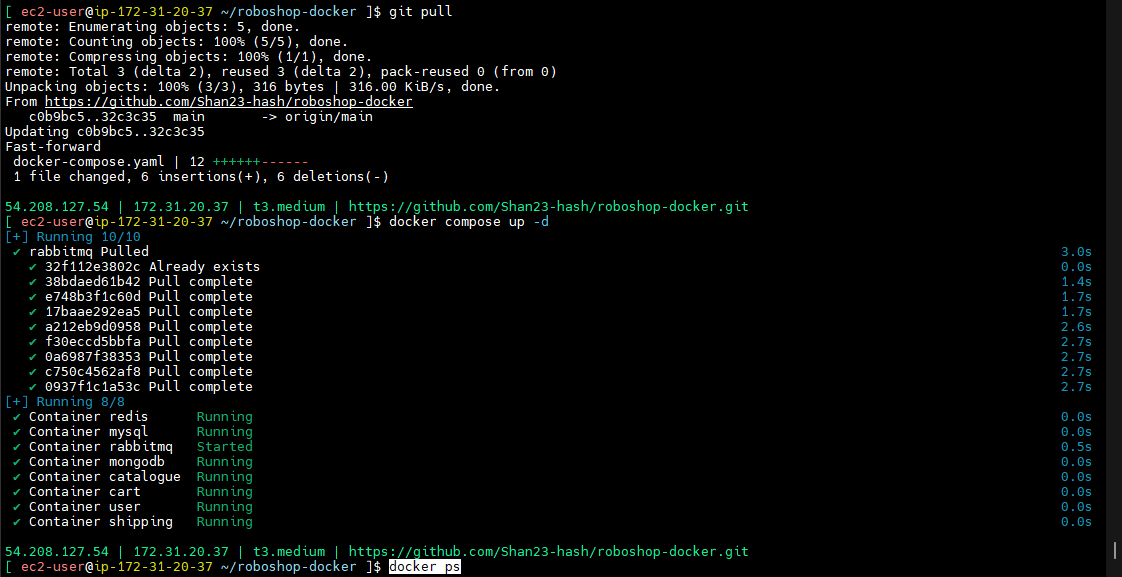
    container\_name: rabbitmq

    environment:

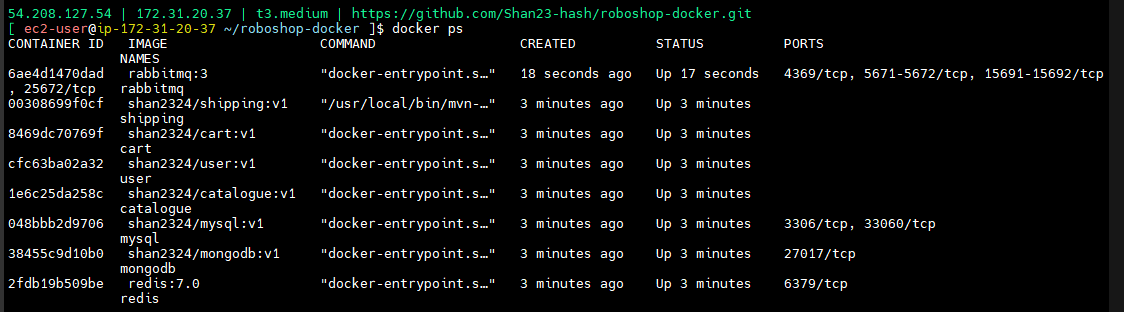
      RABBITMQ\_DEFAULT\_USER: roboshop

      RABBITMQ\_DEFAULT\_PASS: roboshop123

**docker compose up -d**



**docker ps**



**PAYMENT**

--> Payment is python application.

--> take code

<https://roboshop-artifacts.s3.amazonaws.com/payment-v3.zip>

--> we connect select expose 8080

--> expose means there is no functionality it’s just documentation purpose.

--> it can someone inspective your image. Tel them this opening from port 8080.

**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

uid = 1001

gid = 1001

**payment/payment.py**

import random

import instana

import os

import sys

import time

import logging

import uuid

import json

import requests

import traceback

from flask import Flask

from flask import Response

from flask import request

from flask import jsonify

from rabbitmq import Publisher

# Prometheus

import prometheus\_client

from prometheus\_client import Counter, Histogram

app = Flask(\_\_name\_\_)

app.logger.setLevel(logging.INFO)

CART = os.getenv('CART\_HOST', 'cart')

CART\_PORT = os.getenv('CART\_PORT', 8080)

USER = os.getenv('USER\_HOST', 'user')

USER\_PORT = os.getenv('USER\_PORT', 8080)

PAYMENT\_GATEWAY = os.getenv('PAYMENT\_GATEWAY', 'https://google.com/')

# Prometheus

PromMetrics = {}

PromMetrics['SOLD\_COUNTER'] = Counter('sold\_count', 'Running count of items sold')

PromMetrics['AUS'] = Histogram('units\_sold', 'Avergae Unit Sale', buckets=(1, 2, 5, 10, 100))

PromMetrics['AVS'] = Histogram('cart\_value', 'Avergae Value Sale', buckets=(100, 200, 500, 1000, 2000, 5000, 10000))

@app.errorhandler(Exception)

def exception\_handler(err):

app.logger.error(str(err))

return str(err), 500

@app.route('/health', methods=['GET'])

def health():

return 'OK'

# Prometheus

@app.route('/metrics', methods=['GET'])

def metrics():

res = []

for m in PromMetrics.values():

res.append(prometheus\_client.generate\_latest(m))

return Response(res, mimetype='text/plain')

@app.route('/pay/<id>', methods=['POST'])

def pay(id):

app.logger.info('payment for {}'.format(id))

cart = request.get\_json()

app.logger.info(cart)

anonymous\_user = True

# check user exists

try:

req = requests.get('http://{user}:{userPort}/check/{id}'.format(user=USER, userPort=USER\_PORT, id=id))

except requests.exceptions.RequestException as err:

app.logger.error(err)

return str(err), 500

if req.status\_code == 200:

anonymous\_user = False

# check that the cart is valid

# this will blow up if the cart is not valid

has\_shipping = False

for item in cart.get('items'):

if item.get('sku') == 'SHIP':

has\_shipping = True

if cart.get('total', 0) == 0 or has\_shipping == False:

app.logger.warn('cart not valid')

return 'cart not valid', 400

# dummy call to payment gateway, hope they dont object

try:

req = requests.get(PAYMENT\_GATEWAY)

app.logger.info('{} returned {}'.format(PAYMENT\_GATEWAY, req.status\_code))

except requests.exceptions.RequestException as err:

app.logger.error(err)

return str(err), 500

if req.status\_code != 200:

return 'payment error', req.status\_code

# Prometheus

# items purchased

item\_count = countItems(cart.get('items', []))

PromMetrics['SOLD\_COUNTER'].inc(item\_count)

PromMetrics['AUS'].observe(item\_count)

PromMetrics['AVS'].observe(cart.get('total', 0))

# Generate order id

orderid = str(uuid.uuid4())

queueOrder({ 'orderid': orderid, 'user': id, 'cart': cart })

# add to order history

if not anonymous\_user:

try:

req = requests.post('http://{user}:{userPort}/order/{id}'.format(user=USER,userPort=USER\_PORT, id=id),

data=json.dumps({'orderid': orderid, 'cart': cart}),

headers={'Content-Type': 'application/json'})

app.logger.info('order history returned {}'.format(req.status\_code))

except requests.exceptions.RequestException as err:

app.logger.error(err)

return str(err), 500

# delete cart

try:

req = requests.delete('http://{cart}:{cartPort}/cart/{id}'.format(cart=CART, cartPort=CART\_PORT, id=id));

app.logger.info('cart delete returned {}'.format(req.status\_code))

except requests.exceptions.RequestException as err:

app.logger.error(err)

return str(err), 500

if req.status\_code != 200:

return 'order history update error', req.status\_code

return jsonify({ 'orderid': orderid })

def queueOrder(order):

app.logger.info('queue order')

# For screenshot demo requirements optionally add in a bit of delay

delay = int(os.getenv('PAYMENT\_DELAY\_MS', 0))

time.sleep(delay / 1000)

headers = {}

publisher.publish(order, headers)

def countItems(items):

count = 0

for item in items:

if item.get('sku') != 'SHIP':

count += item.get('qty')

return count

# RabbitMQ

publisher = Publisher(app.logger)

if \_\_name\_\_ == "\_\_main\_\_":

sh = logging.StreamHandler(sys.stdout)

sh.setLevel(logging.INFO)

fmt = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

app.logger.info('Payment gateway {}'.format(PAYMENT\_GATEWAY))

port = int(os.getenv("SHOP\_PAYMENT\_PORT", "8080"))

app.logger.info('Starting on port {}'.format(port))

app.run(host='0.0.0.0', port=port)

**payment/rabbitmq.py**

import json

import pika

import os

class Publisher:

HOST = os.getenv('AMQP\_HOST', 'rabbitmq')

USER = os.getenv('AMQP\_USER', 'guest')

PASS = os.getenv('AMQP\_PASS', 'guest')

VIRTUAL\_HOST = '/'

EXCHANGE='robot-shop'

TYPE='direct'

ROUTING\_KEY = 'orders'

def \_\_init\_\_(self, logger):

self.\_logger = logger

self.\_params = pika.connection.ConnectionParameters(

host=self.HOST,

virtual\_host=self.VIRTUAL\_HOST,

credentials=pika.credentials.PlainCredentials(self.USER, self.PASS))

self.\_conn = None

self.\_channel = None

def \_connect(self):

if not self.\_conn or self.\_conn.is\_closed or self.\_channel is None or self.\_channel.is\_closed:

self.\_conn = pika.BlockingConnection(self.\_params)

self.\_channel = self.\_conn.channel()

self.\_channel.exchange\_declare(exchange=self.EXCHANGE, exchange\_type=self.TYPE, durable=True)

self.\_logger.info('connected to broker')

def \_publish(self, msg, headers):

self.\_channel.basic\_publish(exchange=self.EXCHANGE,

routing\_key=self.ROUTING\_KEY,

properties=pika.BasicProperties(headers=headers),

body=json.dumps(msg).encode())

self.\_logger.info('message sent')

#Publish msg, reconnecting if necessary.

def publish(self, msg, headers):

if self.\_channel is None or self.\_channel.is\_closed or self.\_conn is None or self.\_conn.is\_closed:

self.\_connect()

try:

self.\_publish(msg, headers)

except (pika.exceptions.ConnectionClosed, pika.exceptions.StreamLostError):

self.\_logger.info('reconnecting to queue')

self.\_connect()

self.\_publish(msg, headers)

def close(self):

if self.\_conn and self.\_conn.is\_open:

self.\_logger.info('closing queue connection')

self.\_conn.close()

**payment/requirements.txt**

uwsgi

Flask

requests

pika

prometheus\_client

opentracing

Instana

**payment/Dockerfile**

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"]

--> .(dot) means current directory.

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

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

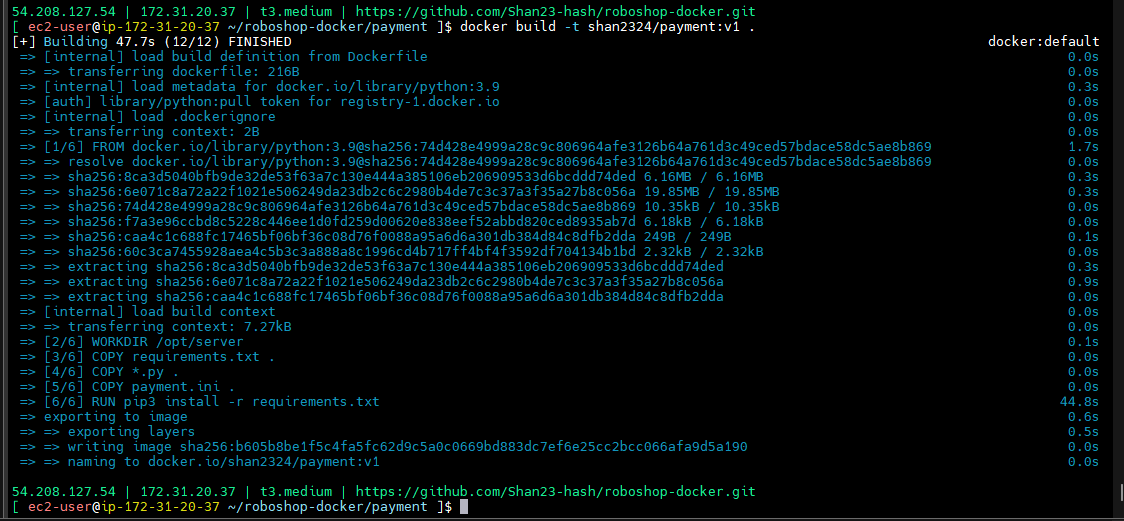
--> run tome or build time also you can give environment. Because environment will work everywhere.

--> if you given run time that is flexible.

--> if I mentioned in docker-compose.yaml file I can just restart the container.

--> build is not easy. If you added space also you shold do rebuild.

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

**docker run -d --name payment shan2324/payment:v1**

**docker-compose.yaml**

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

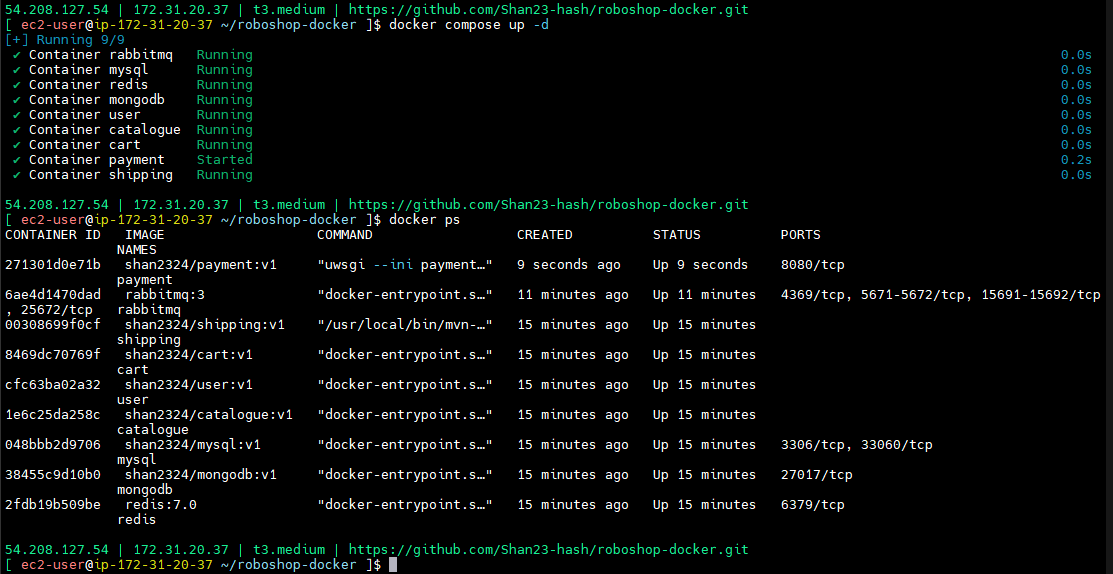
- user

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

**--> git pull**

**docker compose up -d**

**docker ps**



Now it’s running see 8080 port

**FRONTEND**

**-->**  take frontend code

--> frontend totally nginx code

**<https://roboshop-artifacts.s3.amazonaws.com/frontend-v3.zip>**

**--> frontend/Dockerfile**

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/

**frontend/nginx.conf**

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log notice;

pid /run/nginx.pid;

include /usr/share/nginx/modules/\*.conf;

events {

worker\_connections 1024;

}

http {

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

types\_hash\_max\_size 4096;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

include /etc/nginx/conf.d/\*.conf;

server {

listen 80;

listen [::]:80;

server\_name \_;

root /usr/share/nginx/html;

include /etc/nginx/default.d/\*.conf;

error\_page 404 /404.html;

location = /404.html {

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

}

location /images/ {

expires 5s;

root /usr/share/nginx/html;

try\_files $uri /images/placeholder.jpg;

}

location /api/catalogue/ { proxy\_pass http://catalogue:8080/; }

location /api/user/ { proxy\_pass http://user:8080/; }

location /api/cart/ { proxy\_pass http://cart:8080/; }

location /api/shipping/ { proxy\_pass http://shipping:8080/; }

location /api/payment/ { proxy\_pass http://payment:8080/; }

location /health {

stub\_status on;

access\_log off;

}

}

}

--> CMD no need because already there in nginx, it will run automatically.

--> nginx is a server. Automatically it will run image.

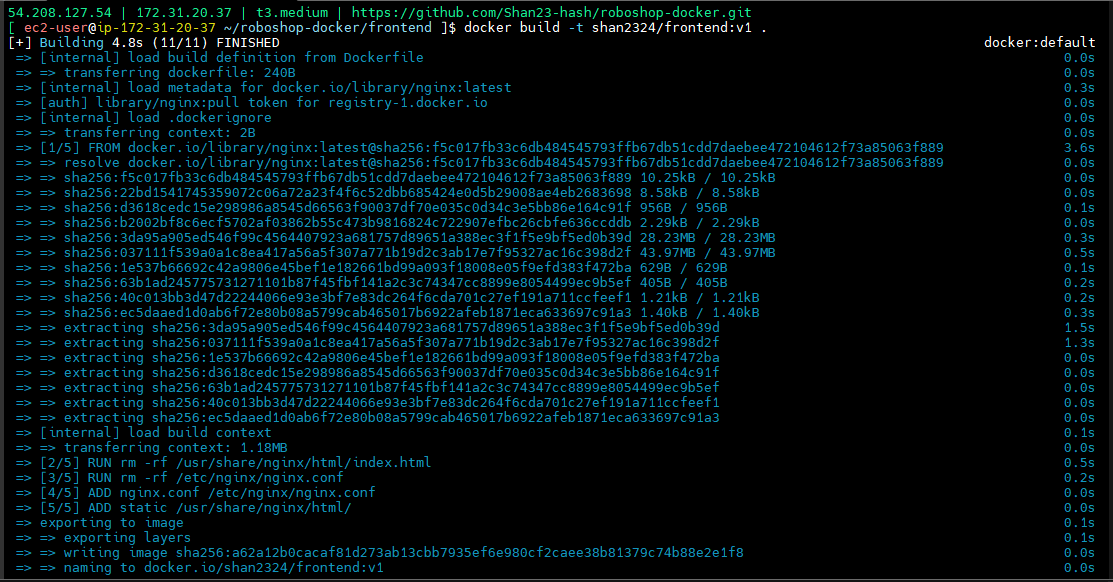
**cd /c/devops/daws-84s/repos/roboshop-docker**

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

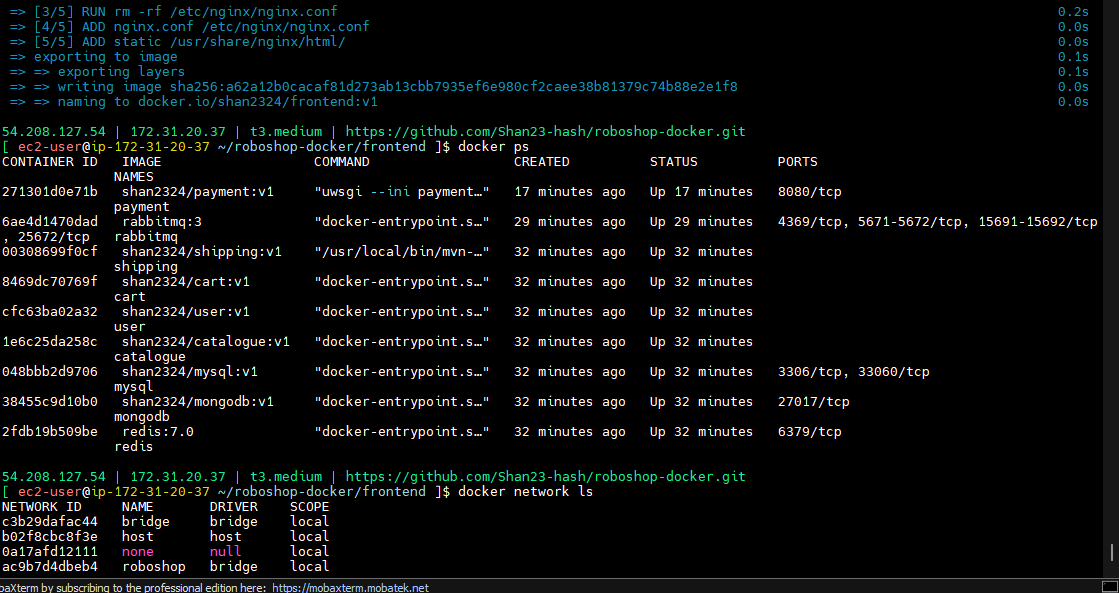
**git pull**

**cd ~/roboshop-docker/frontend**

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



**docker ps**



**docker-compose.yaml**

frontend:

image: shan2324/frontend:v1

container\_name: frontend

ports:

- "80:80"

depends\_on:

- catalogue

- user

- cart

- shipping

- payment

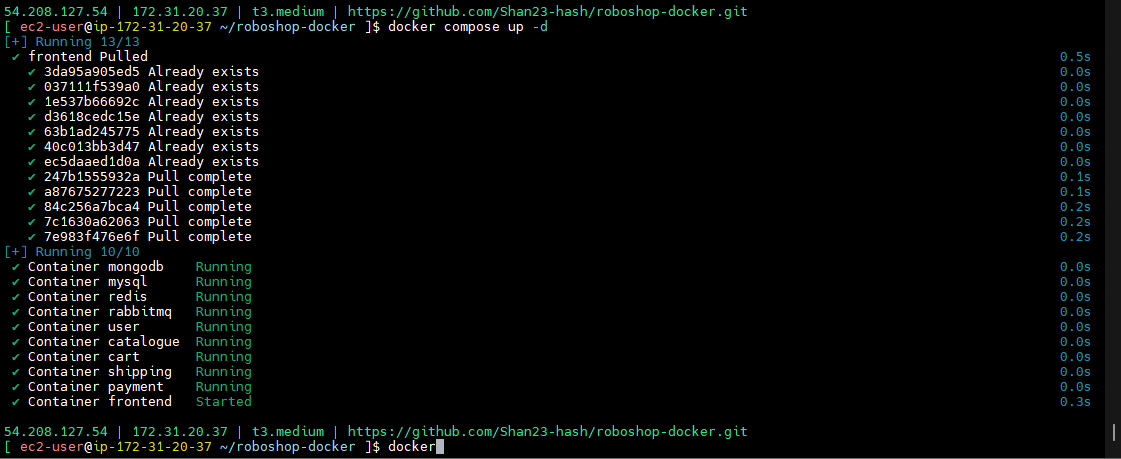
**cd /c/devops/daws-84s/repos/roboshop-docker**

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

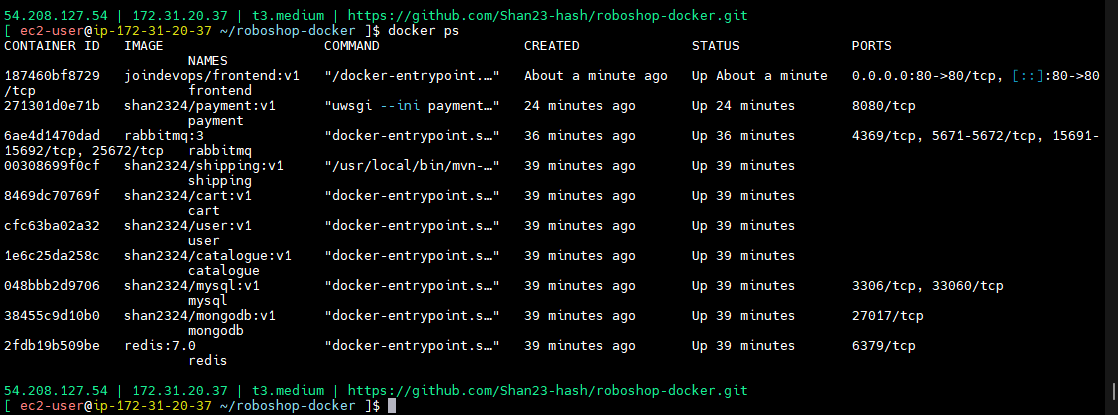
**cd ~/roboshop-docker**

**git pull**

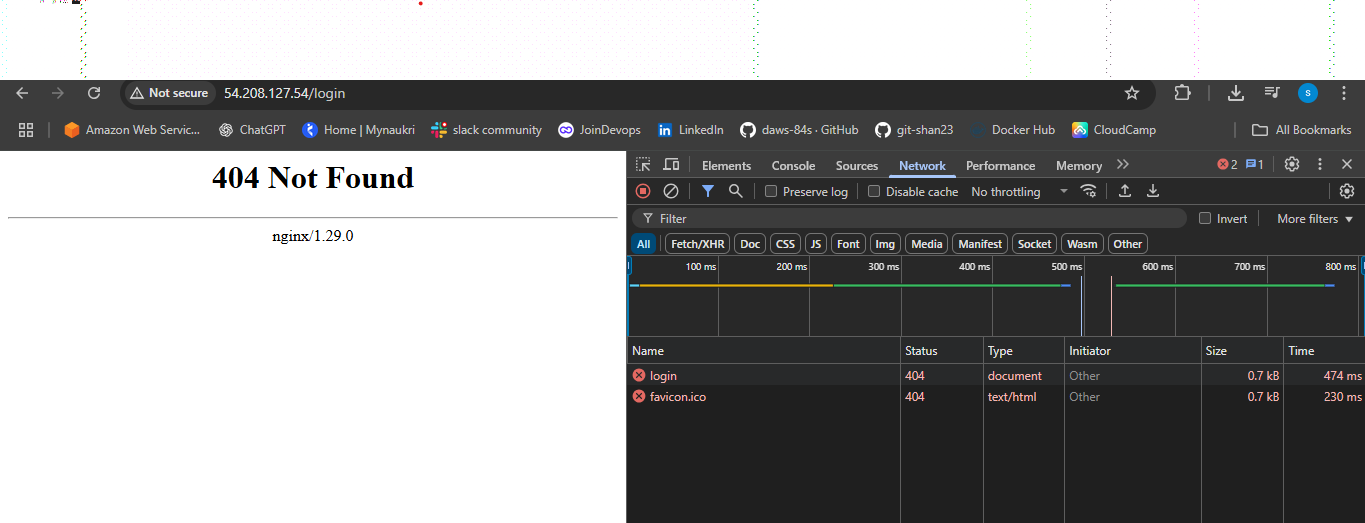
**docker compose up -d**



**docker ps**



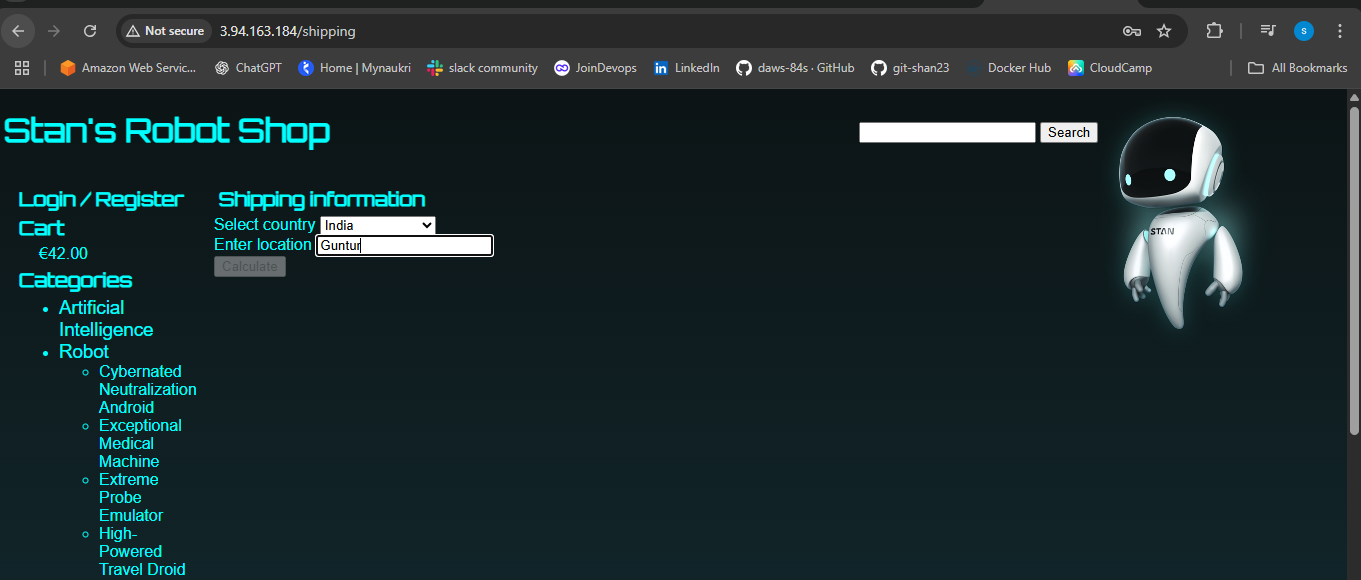
--> Now it can access it



While login I got error

Go to cd user

--> cities not coming



**docker exec -it mysql bash**

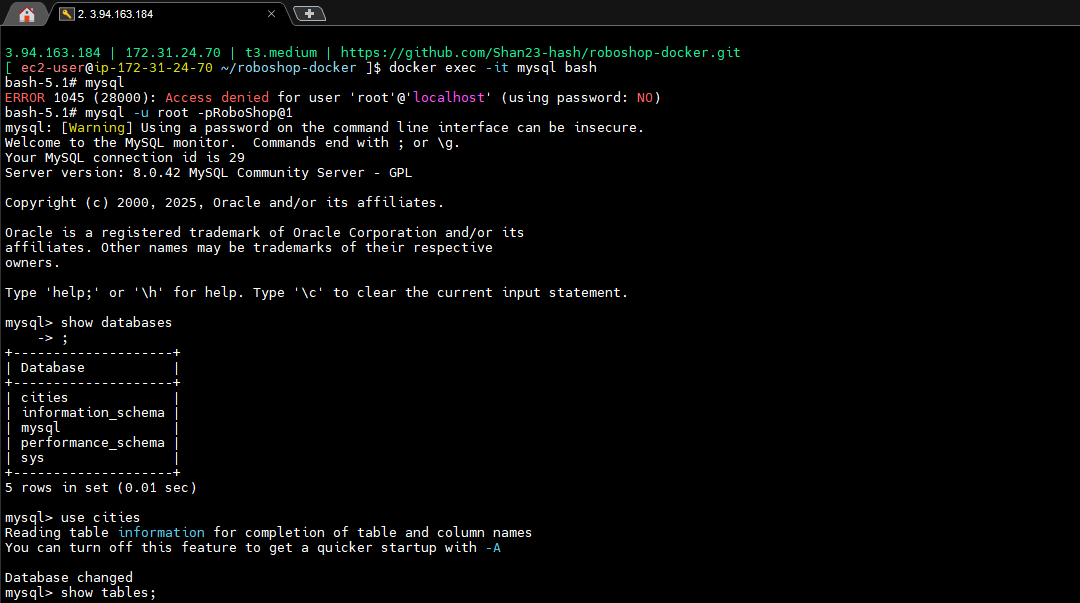
**mysql**

**Pwd: mysql -u root -pRoboShop@1**

**show databases;**

**use cities**

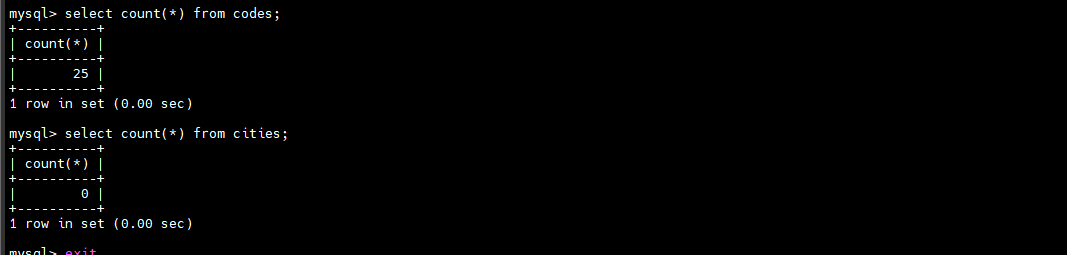
**show tables;**



**select \* from codes;**



**select count(\*) from cities;**



--> because in schema file dropping the table not inserting that’s why remove schema file.

--> in VM we did problem not came.

--> in VM we are created first schema then app user finally master data.

--> generally sql files develop team.

You ca tel this is also one real time example

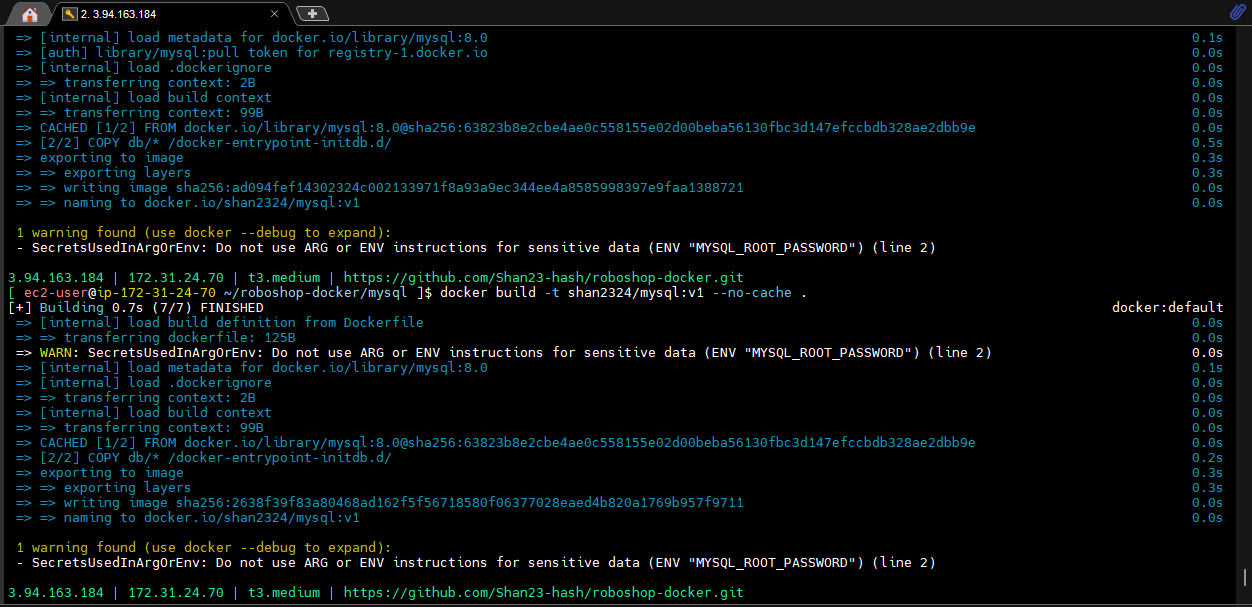
developer provides .sql files if there are any changes in DB structure, by mistake 2 developers created same sql files with different names, one sql file created the table and inserted the data, another sql file drop the table, recreated but didn't inserted the data. so we lost data in the DB. this happened in DEV environment. but it became a big issue and escalated..

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

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

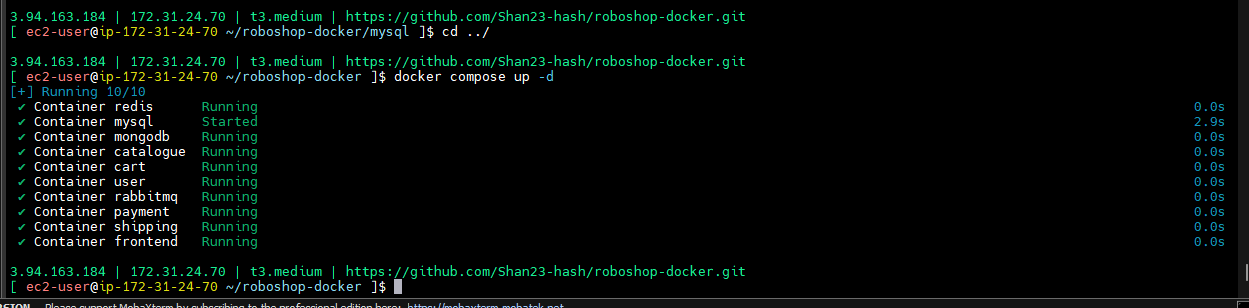
**cd ~/roboshop-docker/mysql**

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



**~/roboshop-docker**

**docker compose up -d**



--> then also not coming same error then

--> Error name found error

**docker exec -it mysql bash**

**mysql -u root -pRoboShop@1**

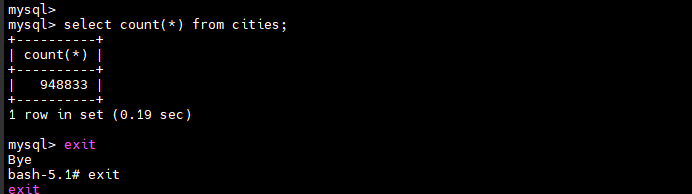
**show databases;**

**use cities**

**show tables;**

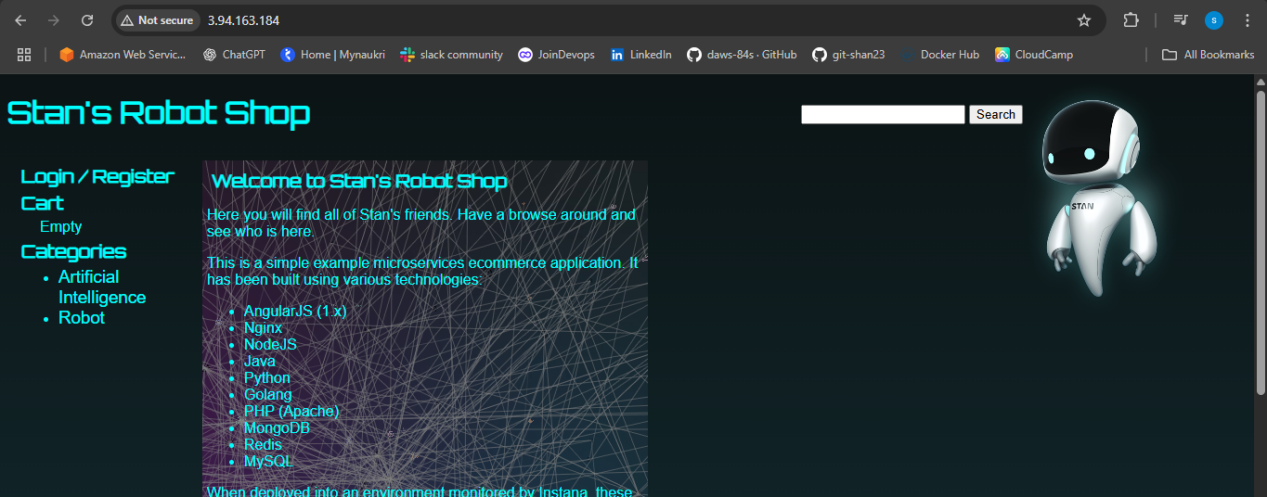
**select count(\*) from cities;**

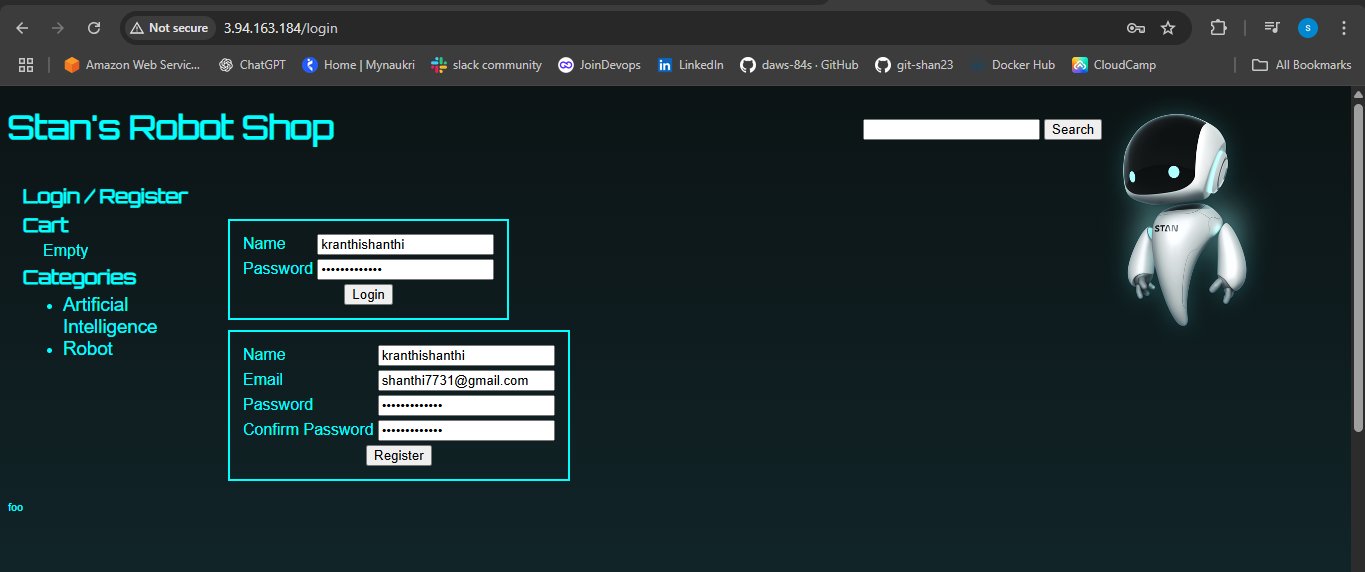
**-->** i inserted manually

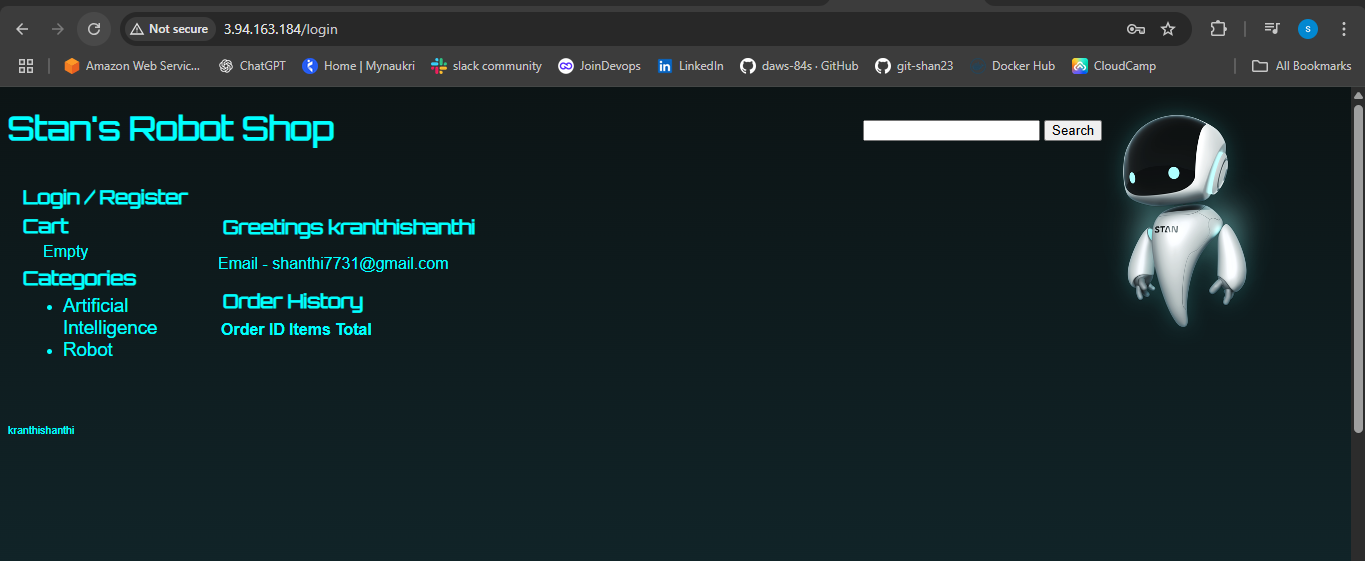


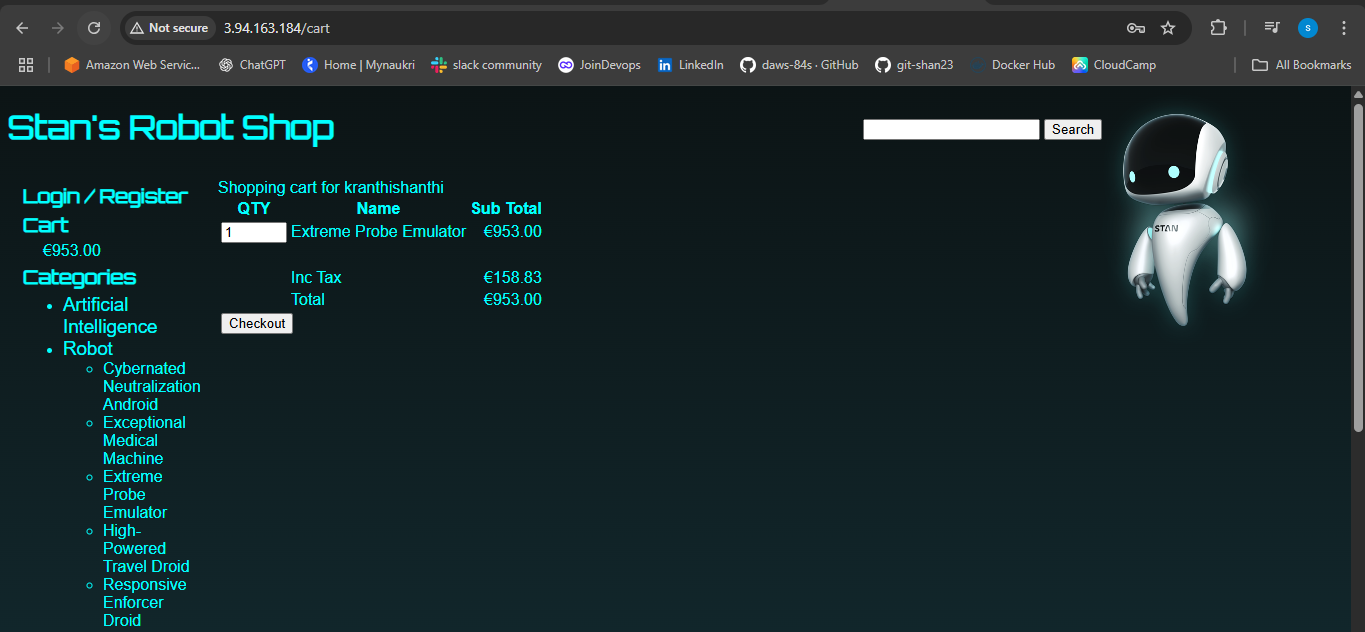
-->containers are ephemeral by default, if you remove them it will remove the data by default…

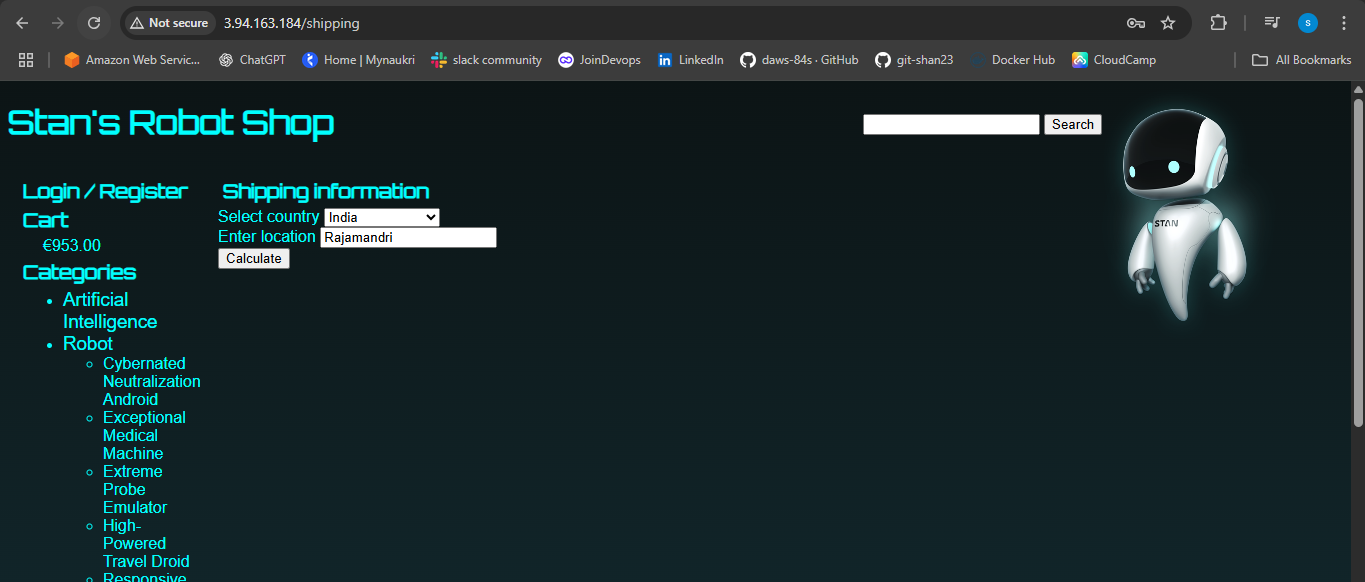
<http://3.94.163.184/>

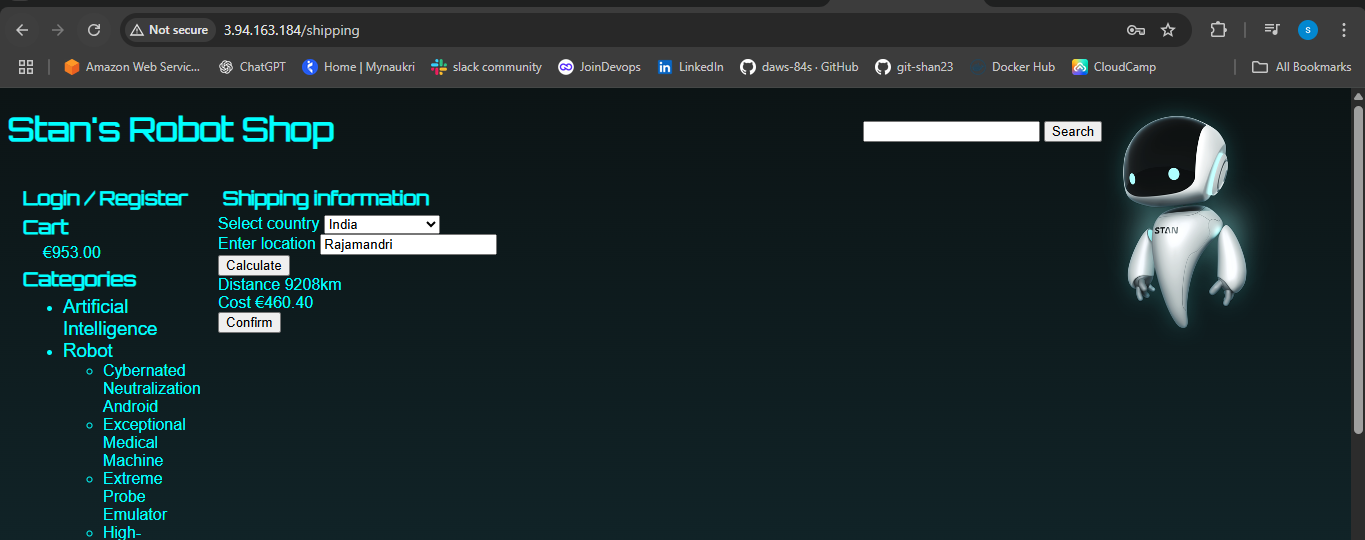


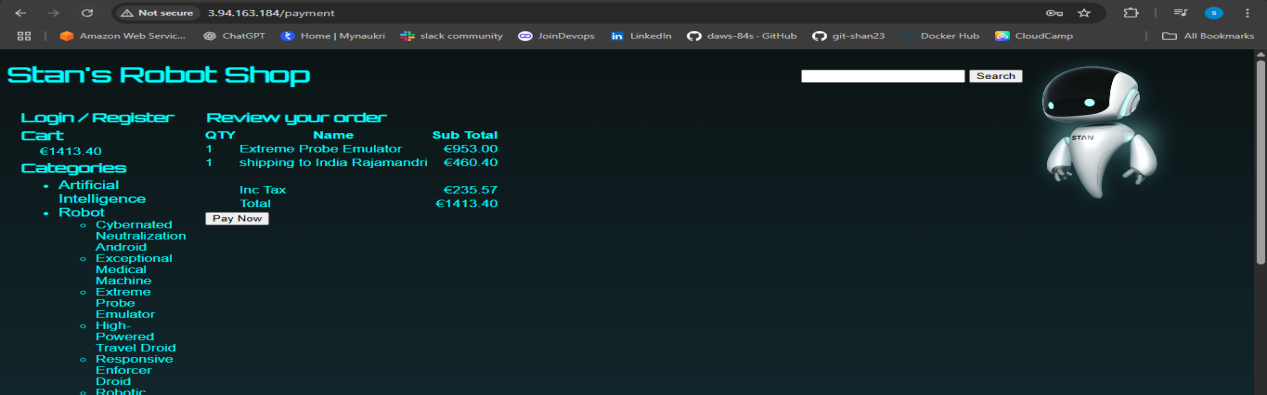


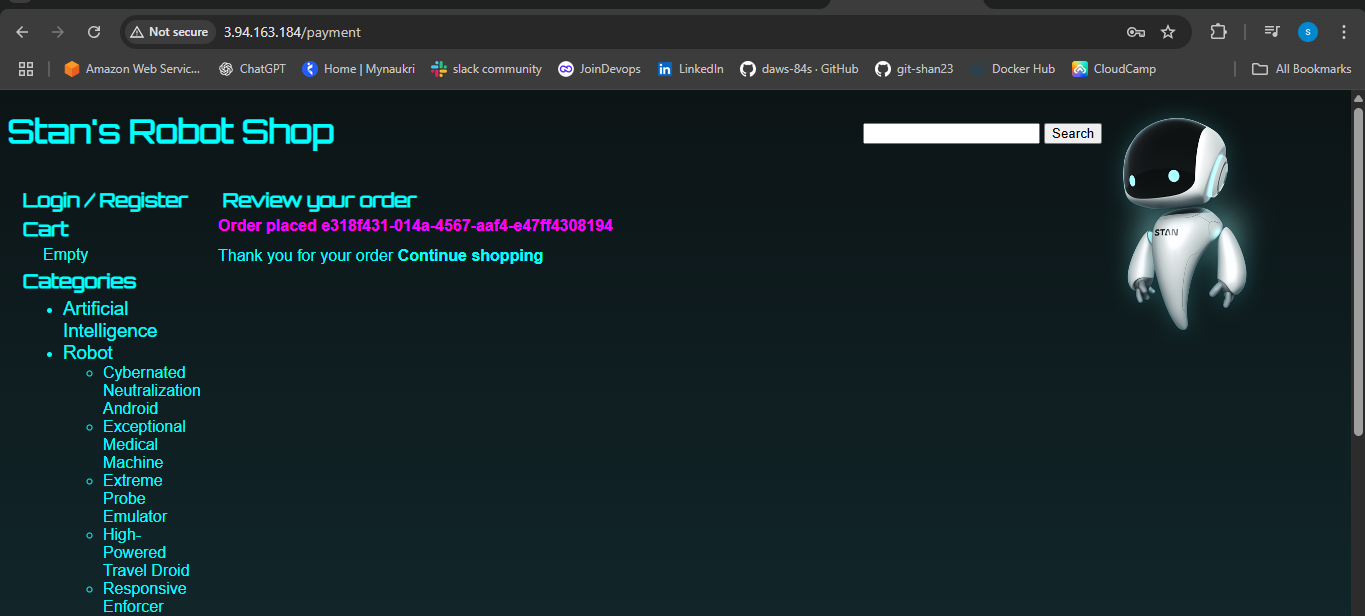












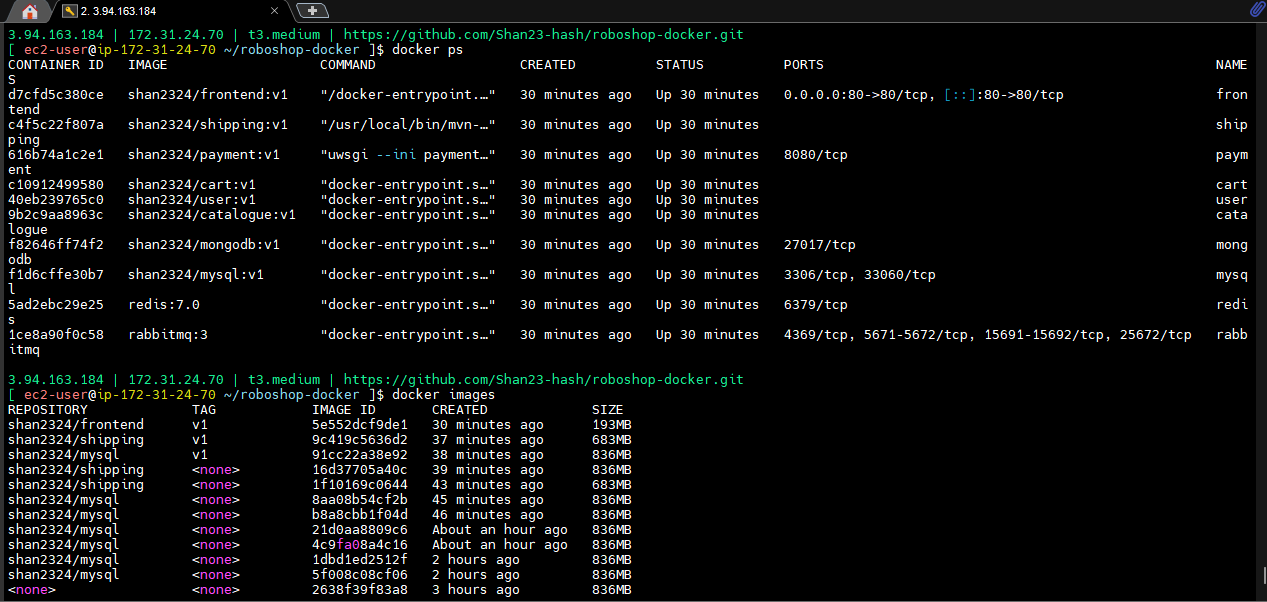
* cd mongodb
* docker build -t mongodb:v1 .
* cd ../catalogue
* docker build -t catalogue:v1 .
* cd -
* docker build -t shan2324/mongodb:v1 .
* cd -
* docker build -t shan2324/catalogue:v1 .
* docker push shan2324/catalogue:v1
* cd -
* docker push shan2324/mongodb:v1
* cd ../
* docker compose up -d mongodb
* docker compose up -d
* cd shipping
* docker push shan2324/shiiping:v1
* cd ../mysql
* docker push shan2324/mysql:v1
* docker build -t shan2324/mysql:v1 .
* docker push shan2324/mysql:v1
* cd ../shipping
* docker build -t shan2324/shipping:v1 .
* docker push shan2324/shipping:v1
* cd ../
* docker compose up -d
* cd user
* docker build -t shan2324/user:v1 .
* cat server.js
* cat >server.js
* docker push shan2324/push:v1
* docker push shan2324/user:v1
* cd ../
* docker compose up -d
* docker ps
* docker exec -it mysql:v1
* docker exec -it mysql:v1 bash
* docker ps
* docker exec -it shan2324/mysql:v1 bash
* docker exec -it mysql bash
* git pull
* cd mysql
* docker build -t shan2324/mysql:v1 .
* docker build -t shan2324/mysql:v1 --no-cache .
* cd ../
* docker compose up -d
* docker exec -it mysql bash
* cd mysql
* ls -l
* docker build -t shan2324/mysql:v1 --no-cache .
* cd ..
* docker ps
* docker rm -f mysql
* docker compose up -d
* docker compos down
* docker compose down
* docker compose up -d
* docker ps
* cd mysql
* docker push shan2324/mysql:v1
* cd ../
* docker compose up -d
* docker compose down
* docker compose up -d
* cd mysql
* docker build -t shan2324/mysql:v1 .
* docker push shan2324/mysql:v1
* cd ../
* docker compose up -d
* docker ps
* curl -X POST http://shipping:8080/shipping
* cd mysql
* docker rm -f mysql
* docker compose up -d
* docker rm -f mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* cd ../
* docker compose up -d
* docker compose down
* docker ps
* ls -l
* cd roboshop-docker
* docker ps
* docker compose up -d
* docker ps
* git pull
* docker compose up -d
* docker network rm roboshop
* docker compose down
* docker network rm roboshop
* docker ps
* docker rm -f mysql
* cd mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* docker exec -it mysql bash
* docker ps
* git pull
* cd ../
* docker network rm roboshop
* cd mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* docker compose down
* cd ../
* docker compose up -d
* docker ps
* docker exec -it mysql bash
* docker cp shipping.sql mysql:/shipping.sql
* docker cp shipping mysql:/shipping
* docker exec -it mysql mysql -u root -p
* docker exec -i mysql mysql -u root -p cities < shipping
* docker exec -i mysql mysql -u root -pRoboShop@1 cities < shipping
* docker exec -it mysql mysql -u root -pRoboShop@1 -e "USE cities; SELECT COUNT(\*) FROM cities;"
* cat shipping/data.sql
* cd mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* cd ../shipping
* git pull
* cd ../mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* cd ../shipping
* docker build -t shan2324/shipping:v1 --no-cache .
* docker push shan2324/shipping:v1
* cd ../
* docker compose down
* docker compose up -d
* docker exec -it mysql bash
* docker rm -f mysql
* cd mysql
* docker build -t shan2324/shipping:v1 --no-cache .
* docker push shan2324/shipping:v1
* cd ../
* docker rm -f shipping
* cd mysql
* docker build -t shan2324/mysql:v1 --no-cache .
* docker push shan2324/mysql:v1
* cd ../shipping
* docker build -t shan2324/shipping:v1 --no-cache .
* docker push shan2324/shipping:v1
* docker ps
* cd ../
* docker compose up -d
* docker ps
* docker compose down
* docker rm -f frontend
* cd frontend
* docker build -t shan2324/frontend:v1 --no-cache .
* cd ../
* docker compose up -d
* docker ps
* docker exec -it mysql bash

I added total troubleshoot commands.

--> Now our problem is this not looking like containers.size also too big and then data is temparory

**docker ps**

**docker images**



--> we have to optimize this

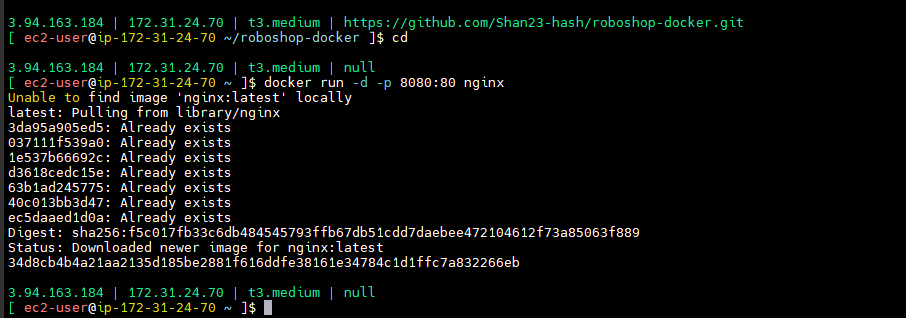
--> container will tel like containers are ephemeral by default, if you remove them it will remove the data by default…

--> then solution is volume, storage

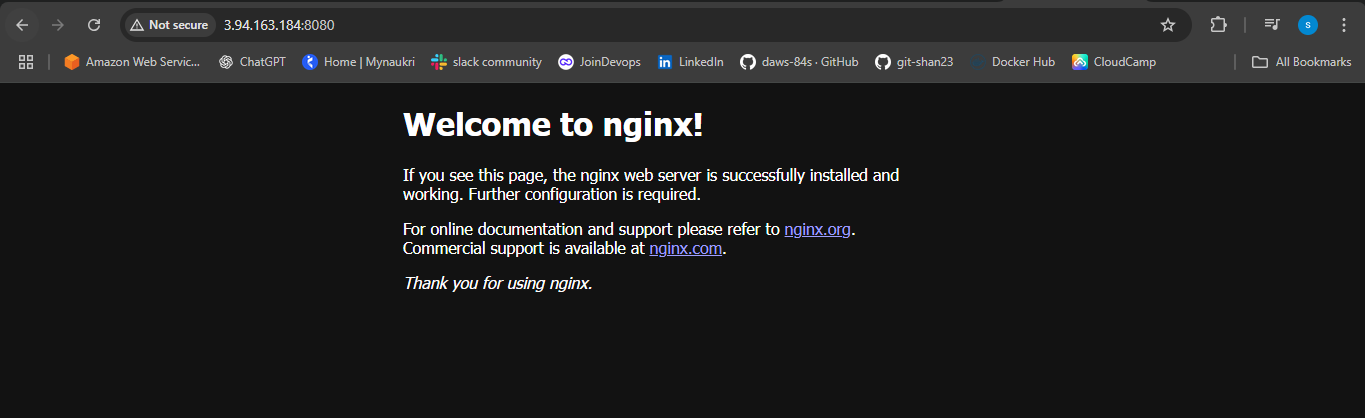
--> use docker volumes

--> **cd**

**--> docker run -d -p 8080:80 nginx**

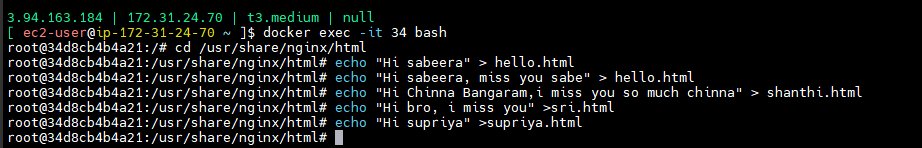


<http://3.94.163.184:8080/>

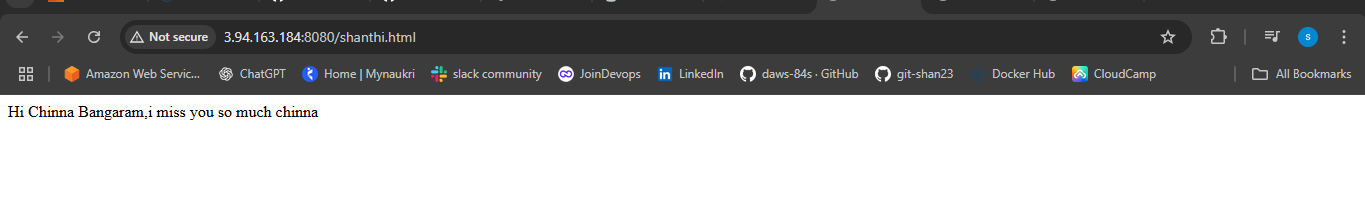


**docker exec -it 34 bash**

**echo "Hi Chinna Bangaram,i miss you so much chinna" > shanthi.html**



<http://3.94.163.184:8080/shanthi.html>



**docker rm -f 34**

**docker run -d -p 8080:80 nginx**

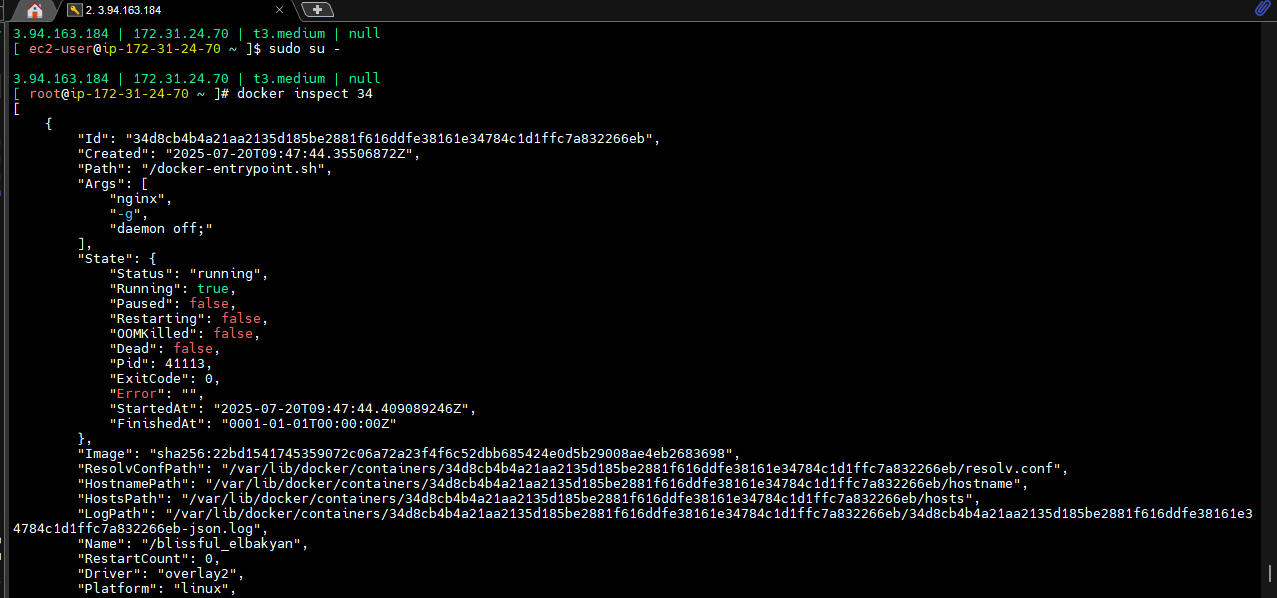
**-->** Now you will get html data or not. --> no

--> whatever you are adding not from image that will delete docker.

--> where it is there this data

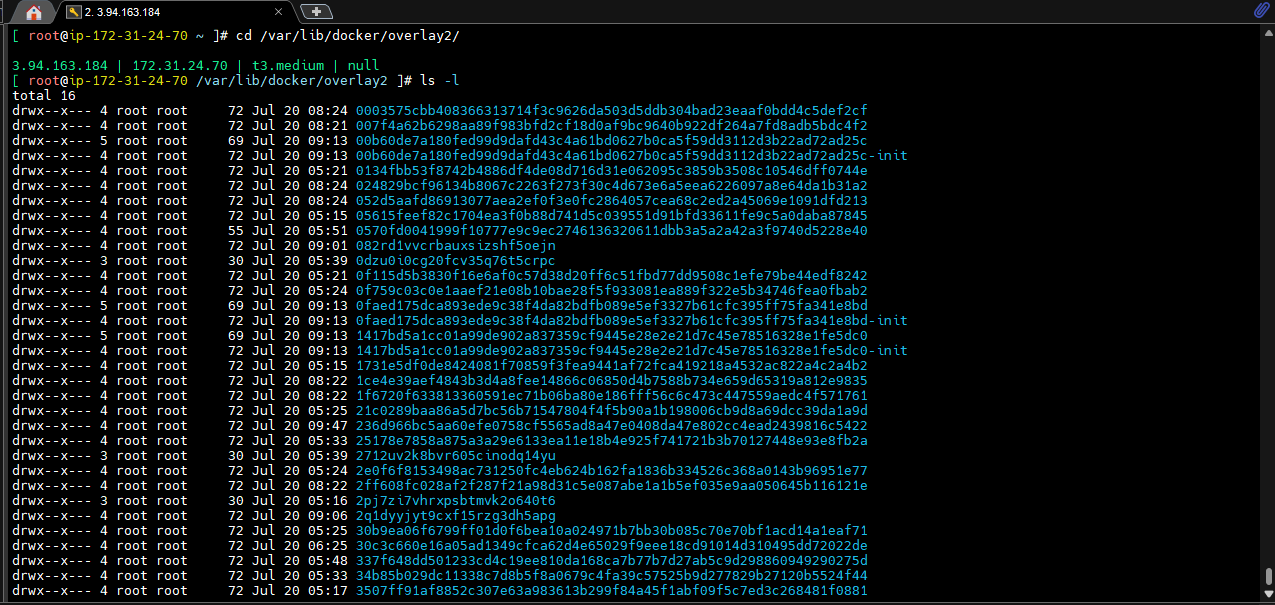
--> while creating docker file every time

**docker inspect 34**



**cd /var/lib/docker/overlay2/**

**ls -l**



--> docker will do in this folder randomly data will create like this directories and keep in data.

--> nginx created one folder randomly.

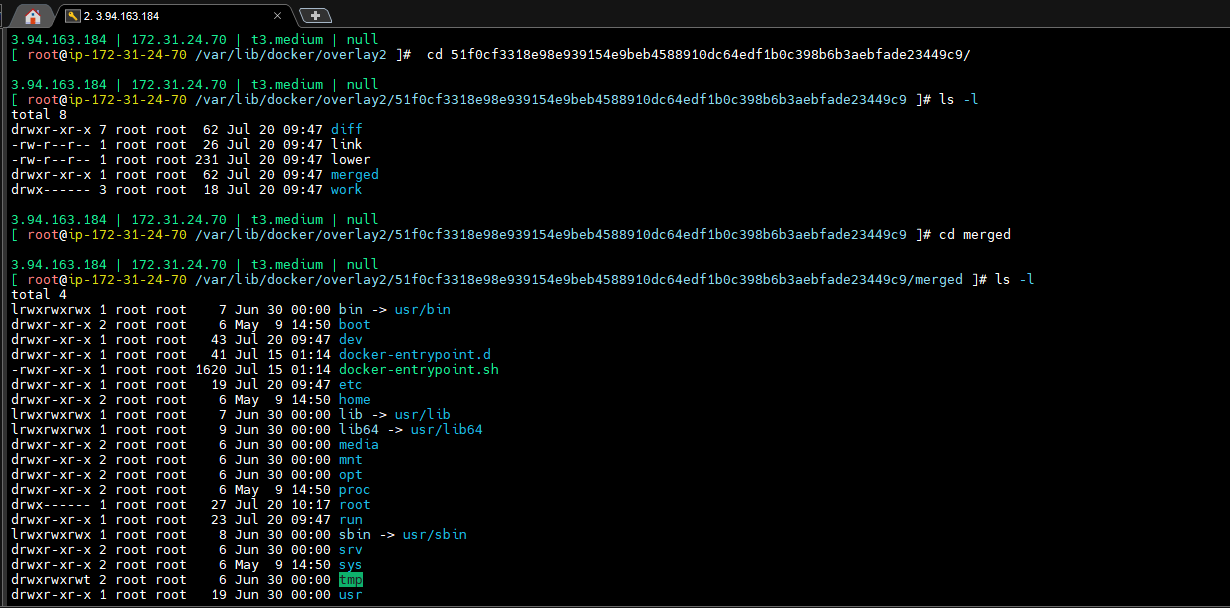
--> 51f0cf3318e98e939154e9beb4588910dc64edf1b0c398b6b3aebfade23449c9/ -- this is the one

**cd /var/lib/docker/overlay2/51f0cf3318e98e939154e9beb4588910dc64edf1b0c398b6b3aebfade23449c9**

**ls -l**

**cd merged**

**ls -l**



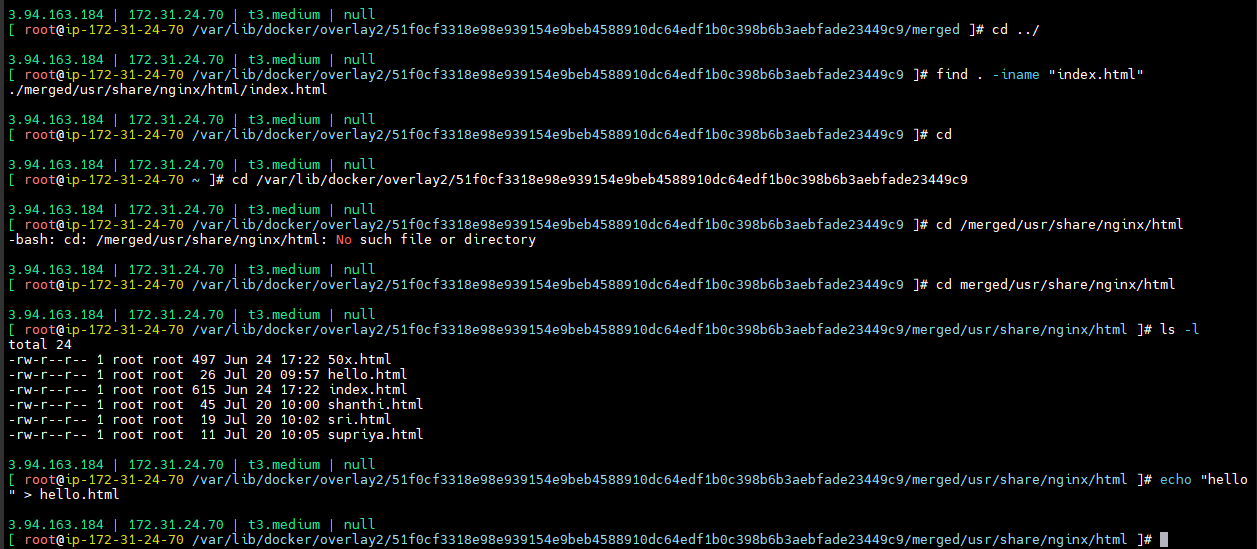
--> we have data in this. If you go merged path our total file system.

**find . -iname "index.html"**

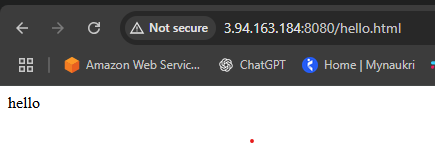
**cd merged/usr/share/nginx/html**

**ls -l**

**echo "hello" > hello.html**



**<http://3.94.163.184:8080/hello.html>**



--> container will do randomly create a directroy. We don’t have control that directroy name

--> it is big random alphanumerical folder.

--> There is a higher structure for an operating, it is a structure.

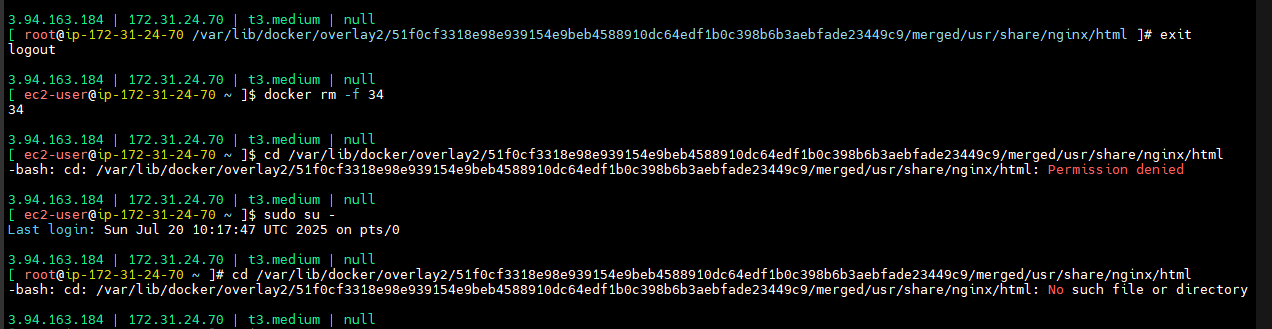
--> if I removed this one

--> for example

**docker rm -f 34**

**sudo su -**

**cd /var/lib/docker/overlay2/51f0cf3318e98e939154e9beb4588910dc64edf1b0c398b6b3aebfade23449c9/merged/usr/share/nginx/html**



--> Whenever you will remove container it will remove this temporary directory also.

--> firctory remove means data also lost.

--> this is docker it will run background. After container creation **/var/lib/docker/overlay2/** randomly it will create folder.

--> container is nothing but nano server.

--> we don’t have saparate storage for this.

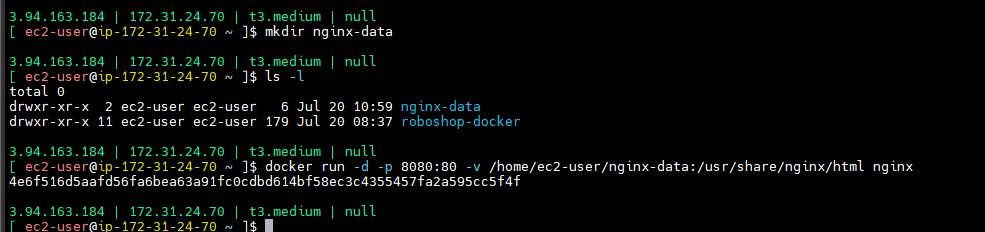
--> if you need data

--> you are expecting html files in /usr/share/nginx/html but that one from your thing in host I given this folder you con consider from this.

**mkdir nginx-data**

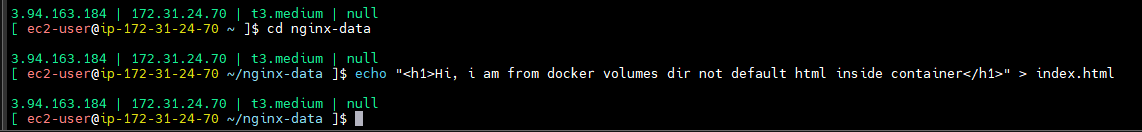
**ls -l**

**docker run -d -p 8080:80 -v /home/ec2-user/nginx-data:/usr/share/nginx/html nginx**

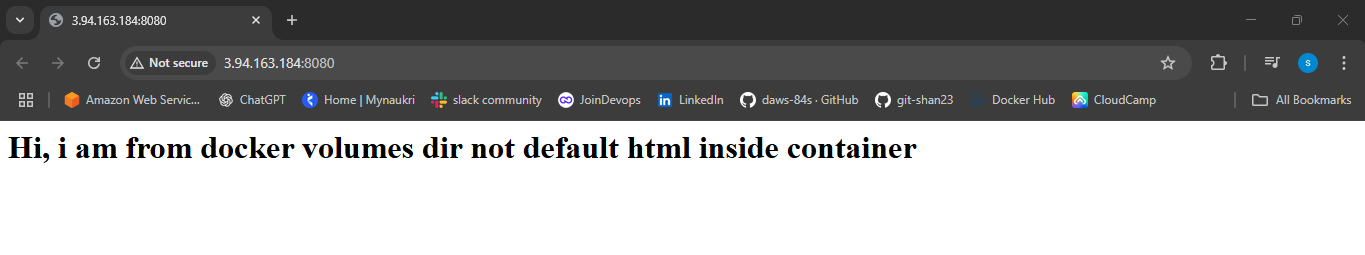


**cd nginx-data**

**echo "<h1>Hi, i am from docker volumes dir not default html inside container</h1>" > index.html**



**<http://3.94.163.184:8080/>**



Now I will remove **docker rm -f 4e**

--> Removed container and data. But nginx-data not removed

--> because it is not in container control.

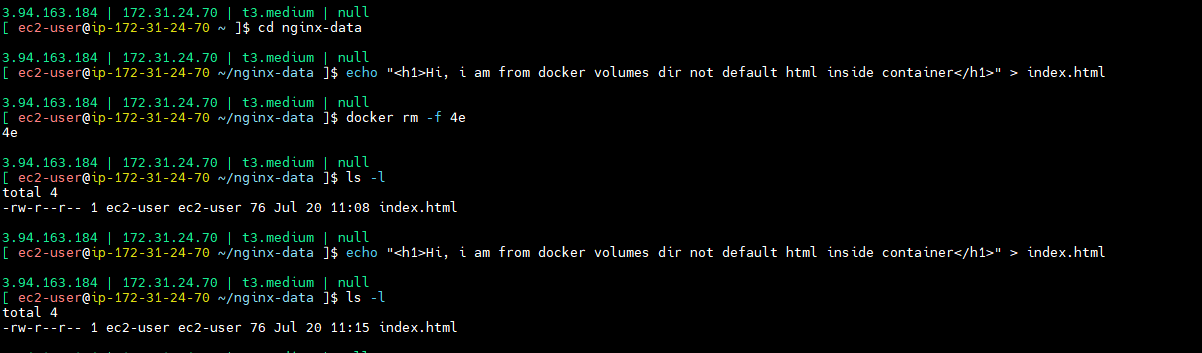
--> what is there in container control -- /var/lib/docker/overlay2/ random directroy.

--> container it is not remove nginx-html

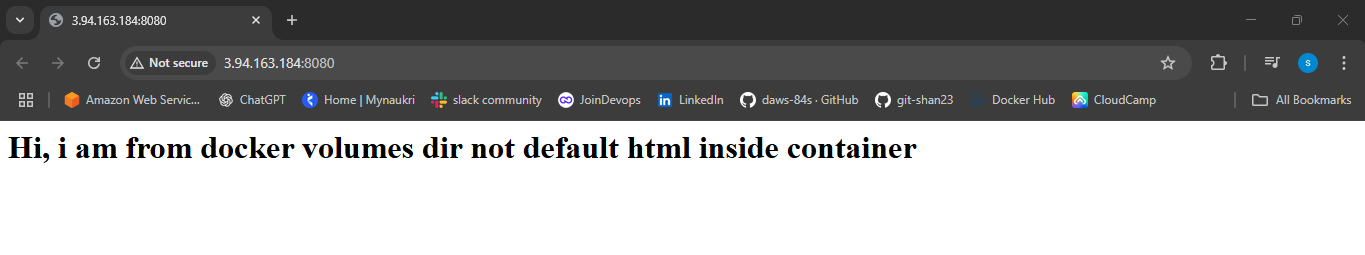
--> if I run same that one should give data.

**ls -l**

**echo "<h1>Hi, i am from docker volumes dir not default html inside container</h1>" > index.html**



**<http://3.94.163.184:8080/>**



* --> So like this we wiil detach from container keep in one locatyion and manage that.
* --> this is docker volume
* --> docker best practices
* Docker best practices encompass writing efficient Dockerfiles, using trusted base images, limiting container privileges, managing data with volumes, and optimizing image size. These practices help ensure security, performance, and maintainability of Dockerized applications.

docker best practices

Docker best practices

Docker best practices encompass a variety of areas, from writing Dockerfiles to managing the security of your containers.

Here's a breakdown:

1. Building secure and efficient Docker images

Write good Dockerfiles: A well-structured Dockerfile is crucial for creating efficient and secure images.

Minimize image size: Keep images as small as possible to improve build speed, deployment times, and reduce the attack surface.

Use multi-stage builds: Leverage multi-stage builds to create smaller images by separating build-time dependencies from runtime dependencies.

Choose the right base image: Use official and trusted base images whenever possible.

Pin image versions: Always use specific tags for base images, not generic ones like "latest," to ensure reproducibility and stability.

Optimize layer caching: Structure your Dockerfile commands to take advantage of Docker's build cache, making subsequent builds faster.

Exclude unnecessary files: Use a .dockerignore file to exclude files and directories that aren't needed in the final image.

Combine RUN instructions: Chain together complex RUN commands using && and \ to minimize the number of image layers.

Clean up unnecessary files: Remove temporary or unnecessary files after each RUN instruction to further reduce image size.

Prefer COPY over ADD: COPY is more explicit and predictable, [according to Sysdig](https://sysdig.com/learn-cloud-native/dockerfile-best-practices/" \t "https://www.google.com/_blank).

Set appropriate user: Run containers as non-root users to limit potential damage from security breaches.

Include a HEALTHCHECK instruction: Define health checks to ensure your containers are functioning correctly.

Build and test images in CI: Integrate image building and testing into your continuous integration (CI) pipeline.

2. Docker security best practices

Use trusted images: Ensure you're using images from reputable sources, like [Docker Official Images](https://www.acorn.io/resources/learning-center/docker-container/" \t "https://www.google.com/_blank).

Implement Docker Content Trust: Use Docker Content Trust to verify the authenticity and integrity of images.

Avoid privileged containers: Do not run containers with escalated privileges, [notes the GitGuardian Blog](https://blog.gitguardian.com/how-to-improve-your-docker-containers-security-cheat-sheet/" \t "https://www.google.com/_blank).

Limit container privileges: Grant only the specific capabilities necessary for a container to function.

Forbid new privileges: Prevent processes within the container from gaining additional privileges.

Run as an unprivileged user: Configure your containers to run as non-root users.

Minimize exposed ports: Only expose the ports essential for the container's functionality.

Securely handle sensitive information: Be cautious about including sensitive data in environment variables; consider using Secrets for managing such information.

Lint your Dockerfiles: Use a linter to identify potential security vulnerabilities and misconfigurations in your Dockerfiles.

Regularly rebuild images: This ensures your images have the latest security patches and updates.

Use image vulnerability scanners: Integrate scanners into your CI/CD pipeline to identify known vulnerabilities in images.

Keep Docker and the host system updated: Regularly apply security updates to the Docker engine and the underlying operating system.

Don't expose the Docker daemon socket: The UNIX socket used by Docker should not be exposed.

3. Data storage

Use named volumes for persistent data: Named volumes are ideal for data that needs to persist beyond the container's lifecycle.

Use bind mounts for development: Bind mounts are useful for development, providing direct access to host files.

Regularly clean up unused volumes: Inspect and remove unused volumes to free up disk space.

By following these best practices, you can create more efficient, secure, and manageable Docker containers, which are crucial for success in a containerized environment.