Assignment 2

**Tasks:** Use Docker Compose to manage a multi-service application.

1. Define a multi-service application consisting of at least three services (e.g., a web app, a database, and a caching service) in a docker-compose.yml file.
2. Configure the services with appropriate environment variables and networking settings.
3. Build and run the Docker Compose application.
4. Test the connectivity between the services and the functionality of the application.

Steps to use docker compose for multi-service application:

1. I took my own react nodejs mysql and redis application.
2. I added Dockerfile to each api and client directory.

**/api/Dockerfile:**

FROM node:latest

WORKDIR /app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 8800

CMD [ "npm", "start" ]

**/client/Dockerfile:**

FROM node:latest

WORKDIR /app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]

1. Now create a new Directory (docker-compose-app) and add docker-compose.yaml

**./docker-compose-app/docker-compose.yaml:**

version: "3.0"

services:

  #mysql service

  mysqldb:

    image: mysql:8.0

    container\_name: mysqlcontainer

    restart: unless-stopped

    # creating volumes to keep the database data consistent and intializing the database with initial data

    volumes:

      - ./mydb.sql:/docker-entrypoint-initdb.d/0\_init.sql

      - $HOME/database:/var/lib/mysql

    # exposing differenct port because port is already in use

    ports:

      - 3900:${MYSQL\_PORT}

    expose:

      - ${MYSQL\_PORT}

    environment:

      MYSQL\_DATABASE: ${MYSQL\_DATABASE}

      MYSQL\_USER: ${MYSQL\_USER}

      MYSQL\_PASSWORD: ${MYSQL\_PASSWORD}

      MYSQL\_ROOT\_PASSWORD: ${MYSQL\_ROOT\_PASSWORD}

      SERVICE\_TAGS: dev

      SERVICE\_NAME: mysqldb

    # using custom bridge network to isolate the containers

    networks:

      - internalnet

  nodejsapi:

    container\_name: node-js-api

    build: ../api/

    image: jainilp12/nodejsapi:1.0

    ports:

      - 8800:${SERVER\_PORT}

    expose:

      - ${SERVER\_PORT}

    environment:

      DB\_HOST: mysqldb

      DB\_PORT: ${MYSQL\_PORT}

      DB\_NAME: ${MYSQL\_DATABASE}

      DB\_USER: ${MYSQL\_USER}

      DB\_PASSWORD: ${MYSQL\_PASSWORD}

      REDIS\_HOST: ${REDIS\_HOST}

      REDIS\_PORT: ${REDIS\_PORT}

      REDIS\_PASSWORD: ${REDIS\_PASSWORD}

      SERVICE\_TAGS: dev

      SERVICE\_NAME: nodejsapi

      SERVER\_PORT: ${SERVER\_PORT}

    networks:

      - internalnet

    depends\_on:

      - mysqldb

      - redisdb

  reactwebapp:

    container\_name: react-web-app-container

    build: ../client/

    image: jainilp12/reactwebapp:1.0

    ports:

      - 3000:${REACT\_PORT}

    expose:

      - ${REACT\_PORT}

    environment:

      SERVICE\_TAGS: dev

      SERVICE\_NAME: reactwebapp

    networks:

      - internalnet

    depends\_on:

      - nodejsapi

  redisdb:

    image: redis:latest

    container\_name: rediscontainer

    restart: always

    ports:

      - 6379:${REDIS\_PORT}

    environment:

      SERVICE\_TAGS: dev

      SERVICE\_NAME: redisdb

    volumes:

      - ./cache:/data

    networks:

      - internalnet

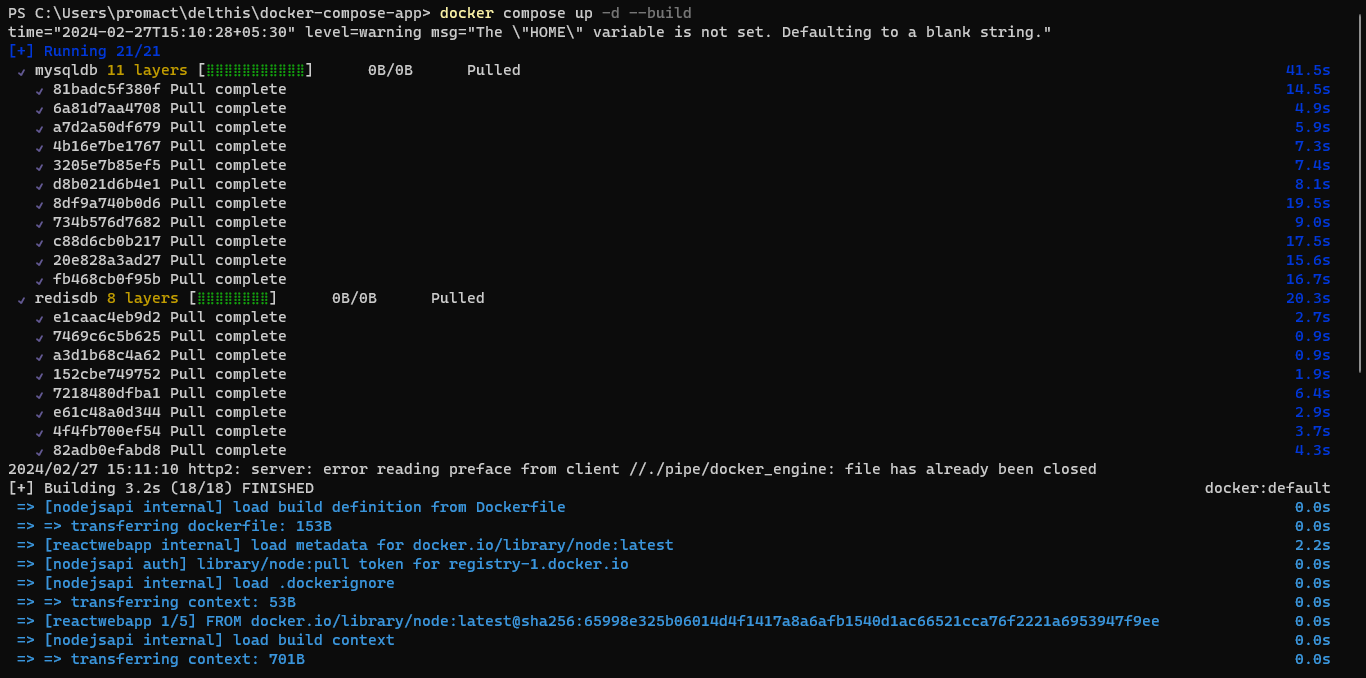
networks:

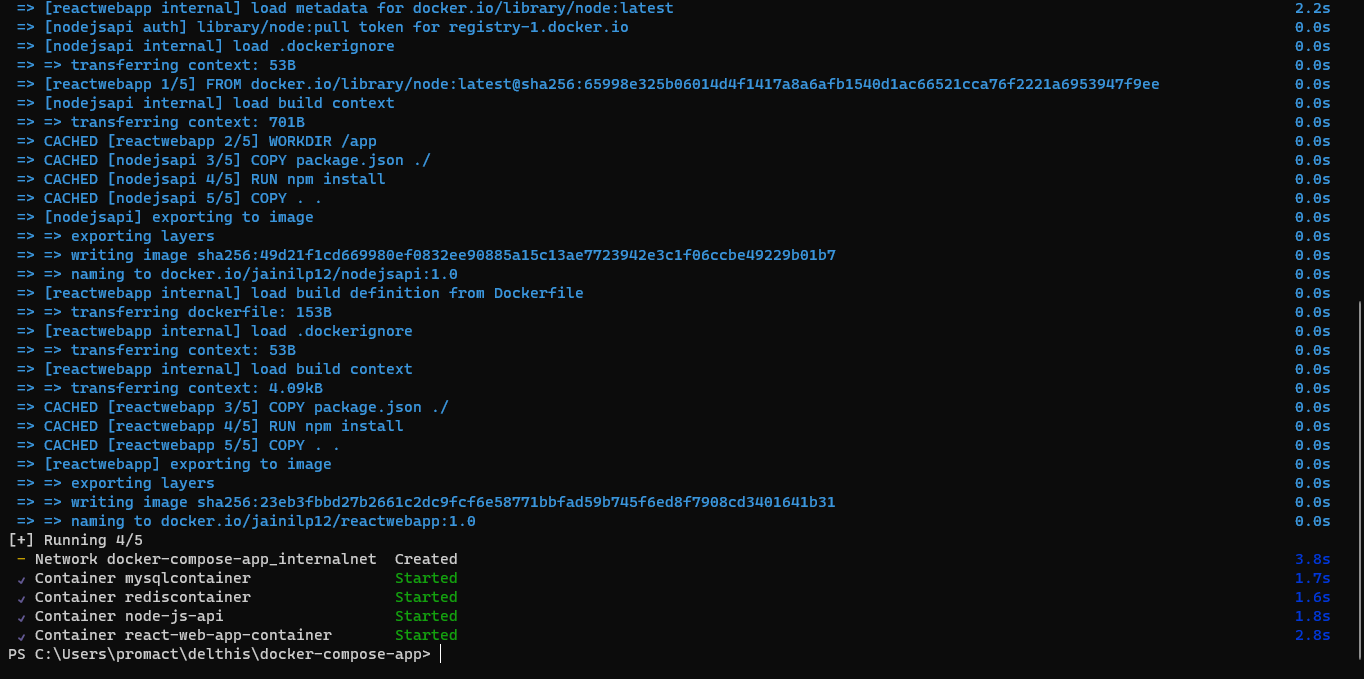
  internalnet:

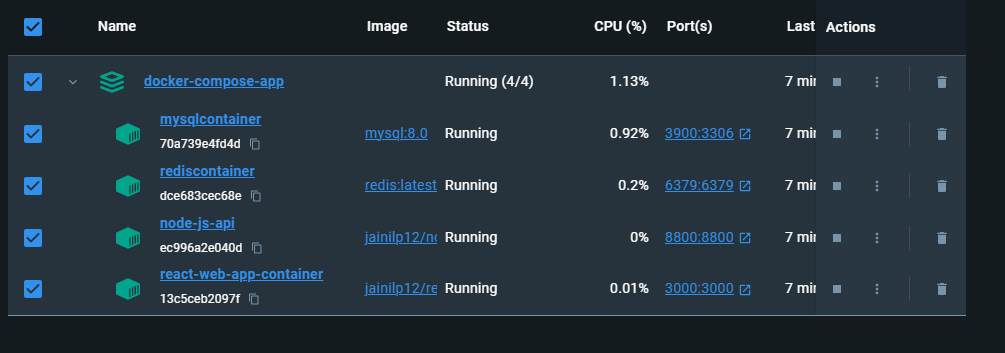
    driver: bridge

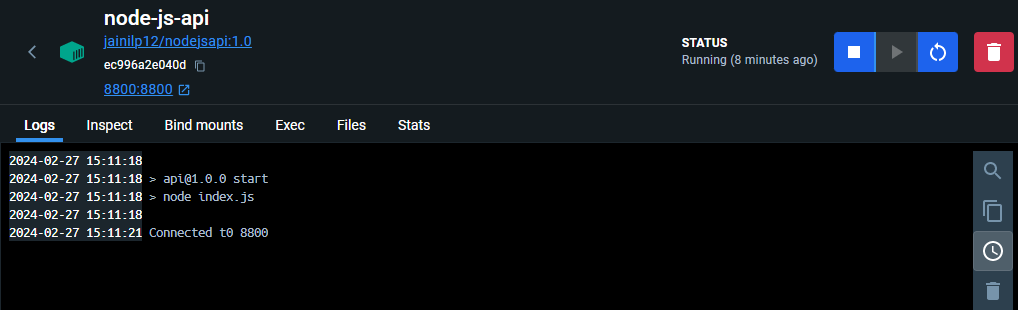
1. Now use docker compose up to run/build/pull all the containers.

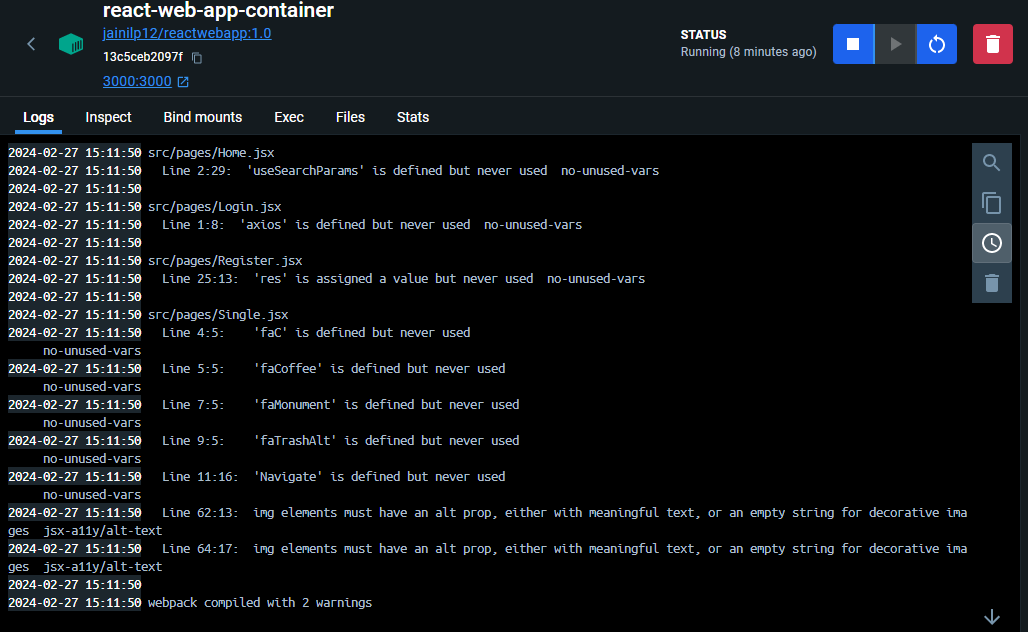
docker compose up -d -build

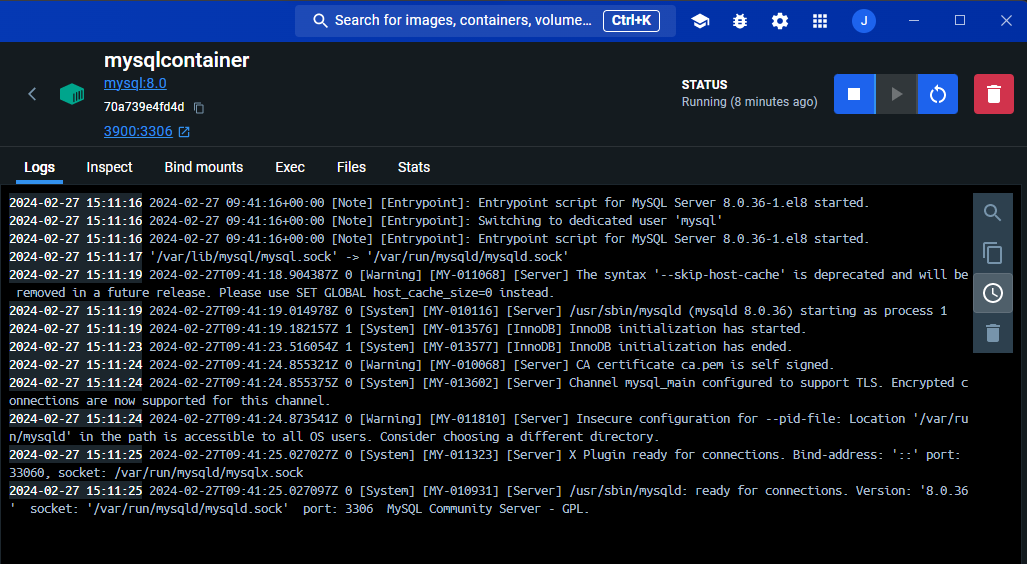
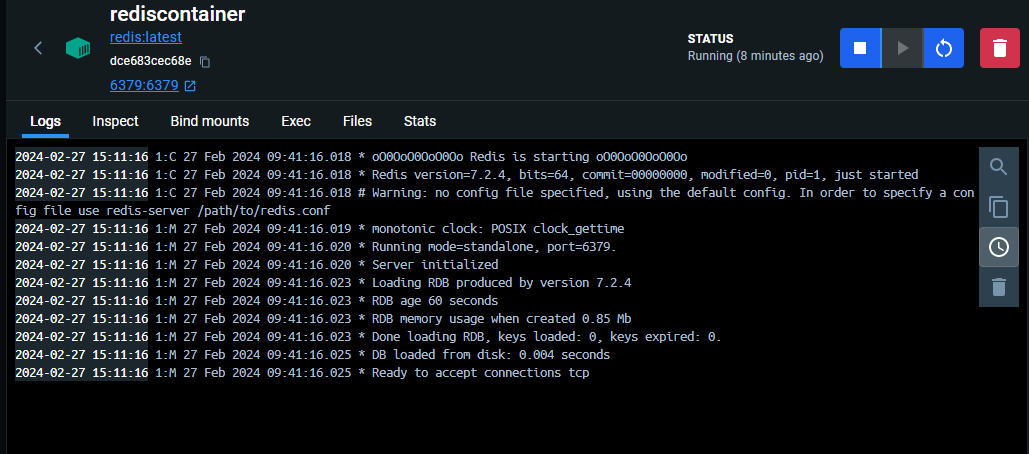


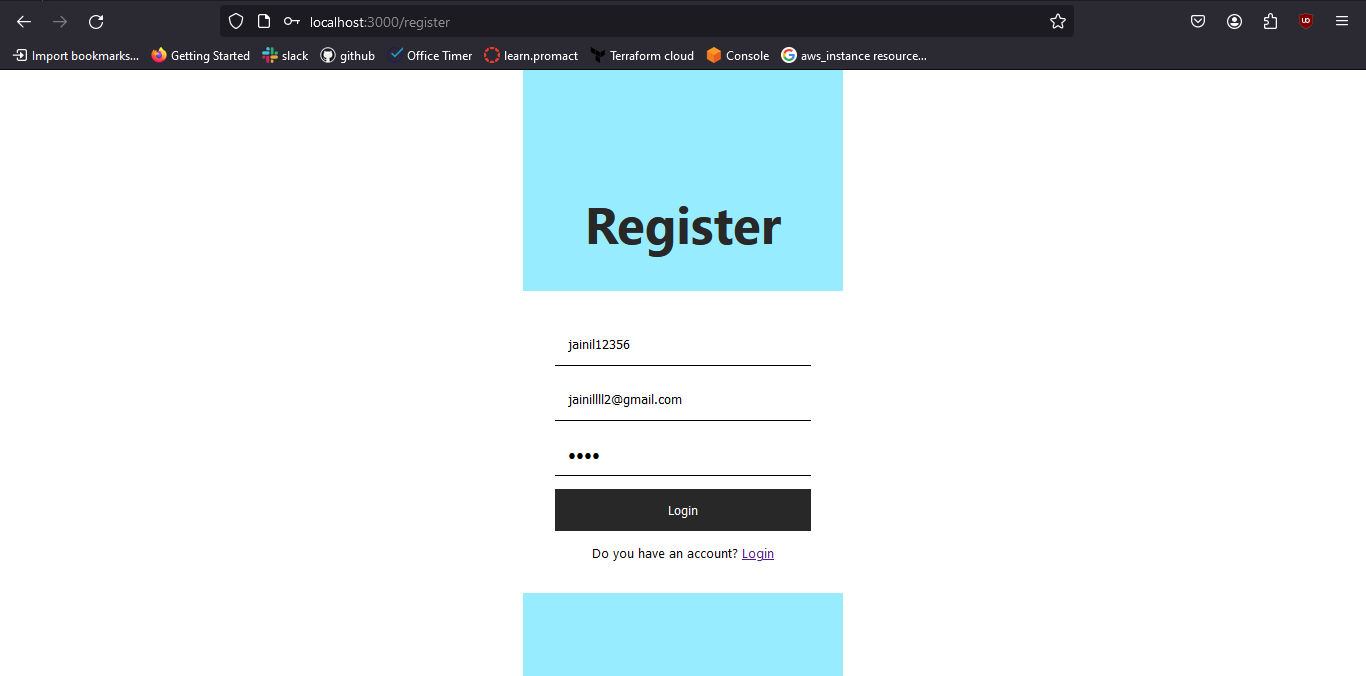


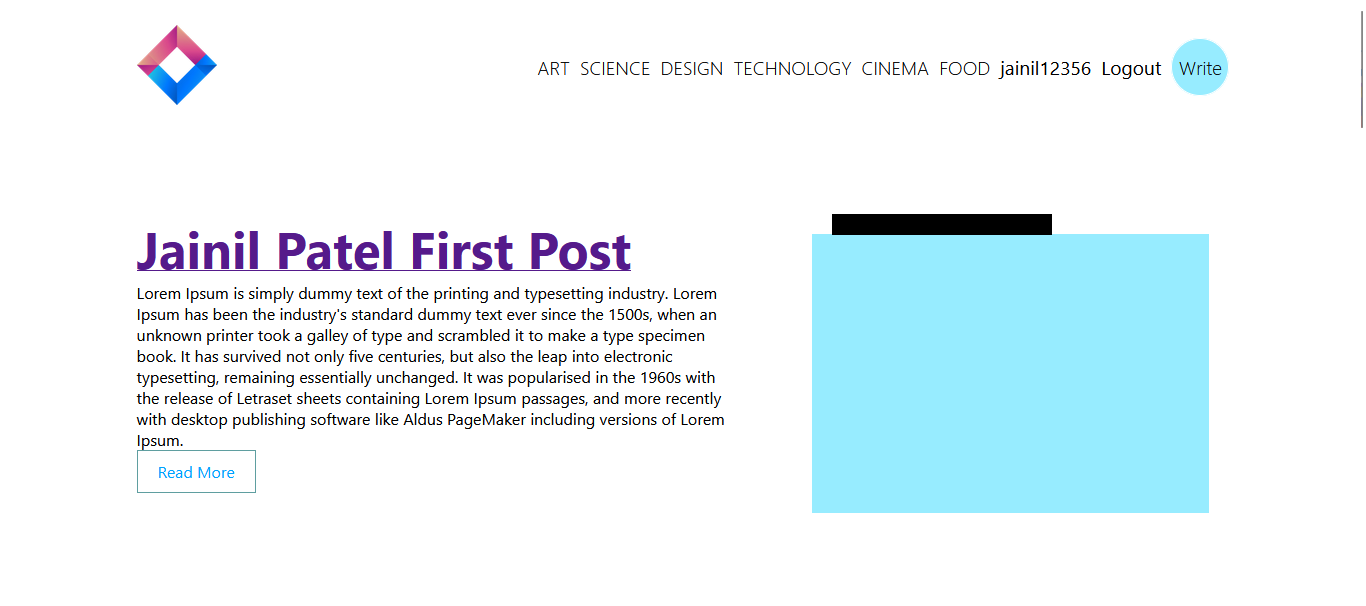












1. Now push the images to the docker hub (I am already logged in), use the following command:

docker compose push

