**Assignment: Dockerizing a Microservices Architecture**

In this assignment, you will use Docker to containerize a microservices architecture that includes a front-end service, a back-end service, and a database. You will then use Docker Compose to manage the containers.

Requirements:

1. **Install Docker and Docker Compose on your computer.**

Install docker used:-->

yum install docker -y

**Install Docker-compose**

**Copy the appropriate docker-compose binary from GitHub:**

sudo curl -L <https://github>.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

**Fix permissions after download:**

sudo chmod +x /usr/local/bin/docker-compose

**Verify success:**

docker-compose version

1. Write a Dockerfile for the front-end service, which is a Node.js application. The Dockerfile should include instructions for copying the application files into the container, installing dependencies, and exposing port 3000.

FROM node:14

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

VOLUME [ "/app/node\_modules" ]

CMD ["npm", "run", "dev"]

1. Write a Dockerfile for the back-end service, which is also a Node.js application. The Dockerfile should include instructions for copying the application files into the container, installing dependencies, and exposing port 5000.
2. Write a Dockerfile for the database, which is a MySQL instance. The Dockerfile should include instructions for setting the root password, creating a new database and user, setting the user's password, and exposing port 3306.
3. Write a docker-compose.yml file that defines three services: frontend, backend, and db. The file should include instructions for linking the frontend and backend services to the db service, setting environment variables for the services, and mapping ports 3000 and 5000 to the host machine.

version: "3.2"

services:

app:

build:

context: .

dockerfile: Dockerfile

ports:

- 3000:3000

volumes:

- .:/app

- '/app/node\_modules'

depends\_on:

- mysql\_server

backend:

build:

context: .

dockerfile: Dockerfile

ports:

- 5000:5000

volumes:

- .:/app

- '/app/node\_modules'

depends\_on:

- mysql\_server

mysql\_server:

image: mysql:8.0

environment:

- MYSQL\_DATABASE=test\_db

- MYSQL\_USER=dan

- MYSQL\_PASSWORD=secret

- MYSQL\_ROOT\_PASSWORD=secret

1. Build and run the containers with Docker Compose.  
   docker-compose up
2. Test the application by visiting http://localhost:3000 in a web browser. You should be able to see the front-end service, which can communicate with the back-end service and the database.

connecting successfully

1. Scale the back-end service to two instances using Docker Compose.  
    if we want to scale up we have to use docker-stack deployment

version: "3.2"

services:

app:

build:

context: .

dockerfile: Dockerfile

ports:

- 3000:3000

volumes:

- .:/app

- '/app/node\_modules'

depends\_on:

- mysql\_server

deploy:

replicas: 2

backend:

build:

context: .

dockerfile: Dockerfile

ports:

- 5000:5000

volumes:

- .:/app

- '/app/node\_modules'

depends\_on:

- mysql\_server

deploy:

replicas: 2

mysql\_server:

image: mysql:8.0

environment:

- MYSQL\_DATABASE=test\_db

- MYSQL\_USER=dan

- MYSQL\_PASSWORD=secret

- MYSQL\_ROOT\_PASSWORD=secret

Command:   
  
docker stack deploy -c docker-compose.yaml tulsi

1. Test the application again by visiting http://localhost:3000 in a web browser. You should see that the front-end service is able to balance requests between the two instances of the back-end service.
2. Stop and remove the containers using Docker Compose.

Submission:

Submit the Dockerfiles and docker-compose.yml file.  
submitted

Submit a screenshot of the application running in the browser.

submitted

Write a brief summary of your experience dockerizing the microservices architecture. Include any challenges you faced and how you overcame them. Also, suggest any improvements you would make to the architecture.

In this assignment my difficult part is finding the Jason file connecting db for the nodejs application   
  
my suggestion is when we are using the docker-compose and stack deployment procedure   
we cannot always do manual scale up and scale down  
we can move to Kubernetes where we can find auto scaling goups and self healing policies   
it will improve the architecture and deploying in kuberntes is easy compared to docker-compose