Docker Project 01

Project Overview

In this project, you'll go through all three lifecycles of Docker: pulling an image and creating a container, modifying the container and creating a new image, and finally, creating a Dockerfile to build and deploy a web application.

Part 1: Creating a Container from a Pulled Image

Objective: Pull the official Nginx image from Docker Hub and run it as a container.

Steps:

Pull the Nginx Image:

```
docker pull nginx
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
f11c1adaa26e: Pull complete
c6b156574604: Pull complete
ea5d7144c337: Pull complete
1bbcb9df2c93: Pull complete
537a6cfe3404: Pull complete
767bff2cc03e: Pull complete
adc73cb74f25: Pull complete
Digest: sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$
```

Run the Nginx Container:

```
docker run --name my-nginx -d -p 8080:80 nginx
```

- `--name my-nginx`: Assigns a name to the container.
- `-d`: Runs the container in detached mode.
- `-p 8080:80`: Maps port 8080 on your host to port 80 in the container.

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker run --name  my-nginx -d -p 8081:80 nginx
a9ba696184683714bfa23216a8f9af99c794d12fc7afb795daaecf2301aca4ef
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$
```

Verify the Container is Running:

```
docker ps
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
                    NAMES
a9ba69618468   nginx     "/docker-entrypoint.…"   28 seconds ago   Up 27 seconds   0.0.0.0:8081->80/t
cp, :::8081->80/tcp   my-nginx
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$
```

- Visit http://localhost:8080 in your browser. You should see the Nginx welcome page.

Part 2: Modifying the Container and Creating a New Image

Objective: Modify the running Nginx container to serve a custom HTML page and create a new image from this modified container.

Steps:

Access the Running Container:

```
docker exec -it my-nginx /bin/bash
```

Create a Custom HTML Page:

```
echo "<html><body><h1>Hello from Docker!</h1></body></html>" >
/usr/share/nginx/html/index.html
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker exec -it my-nginx /bin/bash
root@a9ba69618468:/# echo "<html><body><h1>Hello from Docker!</h1></body></html>" > /usr/share/nginx/
html/index.html
bash: !: event not found
root@a9ba69618468:/# echo '<html><body><h1>Hello from Docker!</h1></body></html>' > /usr/share/nginx/
html/index.html
root@a9ba69618468:/# ls /usr/share/nginx/html/
50x.html  index.html
root@a9ba69618468:/#
```

Exit the Container:

```
exit
```

Commit the Changes to Create a New Image:

```
docker commit my-nginx custom-nginx
```

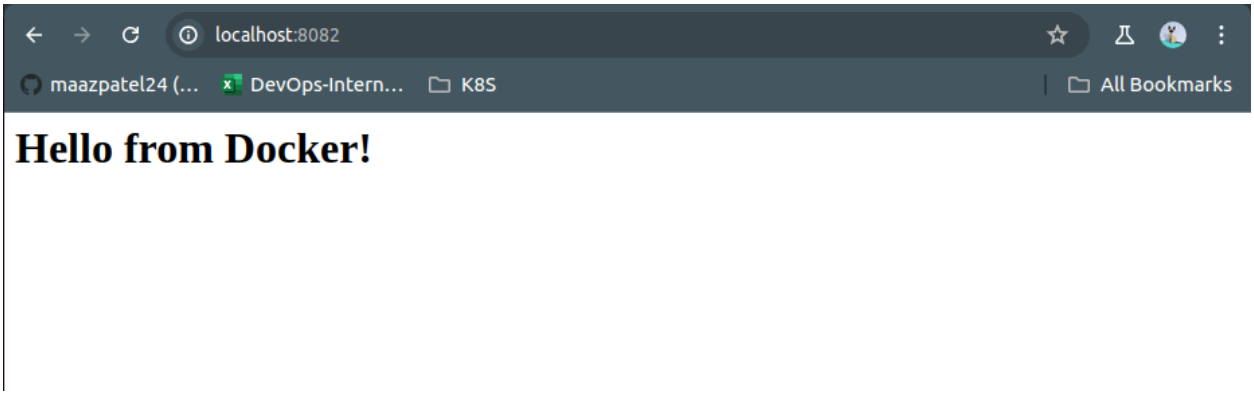
```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker commit my-nginx custom-nginx
sha256:49947516c826544f3a5ca9ed7c6a00c8a944088e9c88cc81e7bcafdab0eaebe4
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$
```

Run a Container from the New Image:

```
docker run --name my-custom-nginx -d -p 8081:80 custom-nginx
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker commit my-nginx custom-nginx
sha256:49947516c826544f3a5ca9ed7c6a00c8a944088e9c88cc81e7bcafdab0eaebe4
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker run --name my-custom-nginx -d -p 8082:80 custom
-nginx
6f8fb70935146f5bd7761e93fd4aa7419434d38226bb89f4bf08f515bcec4f47
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$
```

1. **Verify the New Container:**
   - Visit http://localhost:8081 in your browser. You should see your custom HTML page.



Part 3: Creating a Dockerfile to Build and Deploy a Web Application

Objective: Write a Dockerfile to create an image for a simple web application and run it as a container.

Steps:

Create a Project Directory:

```
mkdir my-webapp
```

```
cd my-webapp
```

2.

3. **Create a Simple Web Application:**

Create an `index.html` file:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Hello from My Web App!</h1>
</body>
</html>
```

-
- Save this file in the `my-webapp` directory.

4. **Write the Dockerfile:**

Create a `Dockerfile` in the `my-webapp` directory with the following content:

```
# Use the official Nginx base image
FROM nginx:latest


# Copy the custom HTML file to the appropriate location
COPY index.html /usr/share/nginx/html/


# Expose port 80
EXPOSE 80
```

-

Build the Docker Image:

```
docker build -t my-webapp-image .
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$ docker build -t my-webapp
-image .
[+] Building 0.3s (7/7) FINISHED                                  docker:default
 => [internal] load build definition from dockerfile                        0.0s
 => => transferring dockerfile: 218B                                        0.0s
 => [internal] load metadata for docker.io/library/nginx:latest             0.0s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [1/2] FROM docker.io/library/nginx:latest                               0.1s
 => [internal] load build context                                           0.0s
 => => transferring context: 129B                                           0.0s
 => [2/2] COPY index.html /usr/share/nginx/html/                            0.0s
 => exporting to image                                                      0.1s
 => => exporting layers                                                     0.0s
 => => writing image sha256:c83faa4ebb3a216e5d8f07a3a88f17cf22fee5d20bc6eec 0.0s
 => => naming to docker.io/library/my-webapp-image                          0.0s
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$
```

Run a Container from the Built Image:

```
docker run --name my-webapp-container -d -p 8082:80 my-webapp-image
```

5.

6. **Verify the Web Application:**
    - Visit http://localhost:8082 in your browser. You should see your custom web application.

Hello from My Web App!

Part 4: Cleaning Up

Objective: Remove all created containers and images to clean up your environment.

Steps:

Stop and Remove the Containers:

```
docker stop my-nginx my-custom-nginx my-webapp-container
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$ docker stop my-nginx my-
ustom-nginx my-webapp-container
my-nginx
my-custom-nginx
my-webapp-container
```

```
docker rm my-nginx my-custom-nginx my-webapp-container
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$ docker rm my-nginx my-cus
tom-nginx my-webapp-container
my-nginx
my-custom-nginx
my-webapp-container
```

7. **Remove the Images:**

```
docker rmi nginx custom-nginx my-webapp-image
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$ docker rmi nginx custom-n
ginx my-webapp-image
Untagged: nginx:latest
Untagged: nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0
f95df
Untagged: custom-nginx:latest
Deleted: sha256:49947516c826544f3a5ca9ed7c6a00c8a944088e9c88cc81e7bcafdab0eaebe4
Deleted: sha256:ccfdfb11cdb30b7d6105b5cfbdb0b8d1b6d904027c23b62eaf3e1ac5eae521d3
Deleted: sha256:fffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765c
Untagged: my-webapp-image:latest
Deleted: sha256:c83faa4ebb3a216e5d8f07a3a88f17cf22fee5d20bc6eec6bf8772a7dfaaf672
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/my-webapp$
```

Docker Project 02

Project Overview

In this advanced project, you'll build a full-stack application using Docker. The application will consist of a front-end web server (Nginx), a back-end application server (Node.js with Express), and a PostgreSQL database. You will also set up a persistent volume for the database and handle inter-container communication. This project will take more time and involve more detailed steps to ensure thorough understanding.

Part 1: Setting Up the Project Structure

Objective: Create a structured project directory with necessary configuration files.

Steps:

Create the Project Directory:

```
mkdir fullstack-docker-app
cd fullstack-docker-app
```

8.

Create Subdirectories for Each Service:

```
mkdir frontend backend database
```

9. **Create Shared Network and Volume:**
    - Docker allows communication between containers through a shared network.

```
docker network create fullstack-network
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker network create fullstack-network
556c74ab6a7426e8b7b9e36fb1cd073947c4c764d04bae80149f76b14a2fffc7
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker network ls
NETWORK ID     NAME                      DRIVER    SCOPE
49cd2ef06b25   bridge                    bridge    local
556c74ab6a74   fullstack-network         bridge    local
94e72bbda352   host                      host      local
cecb7a5fada9   jenkins-config_default    bridge    local
d79dc124ee2d   minikube                  bridge    local
8823d581f4b2   network1                  bridge    local
9aca64b4098f   network2                  bridge    local
956f10e56f7d   none                      null      local
```

- Create a volume for the PostgreSQL database.

```
docker volume create pgdata
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker volume create pgdata
pgdata
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3$ docker volume ls
DRIVER    VOLUME NAME
local     d589ce314cc00ca5403c72812c9f1d204be8d2739dabacf3df3ed36e1a287d45
local     dockercompose_postgresql
local     dockercompose_postgresql_data
local     dockercompose_sonarqube_data
local     dockercompose_sonarqube_extensions
local     dockercompose_sonarqube_logs
local     jenkins-docker-home
local     jenkins-home
local     jenkins_home
local     minikube
local     pgdata
```

Part 2: Setting Up the Database

Objective: Set up a PostgreSQL database with Docker.

Steps:

10. Create a Dockerfile for PostgreSQL:

In the `database` directory, create a file named `Dockerfile` with the following content:

```
FROM postgres:latest
ENV POSTGRES_USER=user
ENV POSTGRES_PASSWORD=password
ENV POSTGRES_DB=mydatabase
```

-

Build the PostgreSQL Image:

```
cd database
docker build -t my-postgres-db .
cd ..
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/database$ docke
r build -t my-postgres-db .
[+] Building 372.1s (6/6) FINISHED                                docker:default
 => [internal] load build definition from dockerfile                       0.0s
 => => transferring dockerfile: 144B                                       0.0s
 => [internal] load metadata for docker.io/library/postgres:latest         2.8s
 => [auth] library/postgres:pull token for registry-1.docker.io            0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [1/1] FROM docker.io/library/postgres:latest@sha256:0aafd2ae7e6c391f3  369.1s
 => => resolve docker.io/library/postgres:latest@sha256:0aafd2ae7e6c391f39f  0.0s
 => => sha256:0aafd2ae7e6c391f39fb6b7621632d79f54068faebc 10.26kB / 10.26kB  0.0s
 => => sha256:f23dc7cd74bd7693fc164fd829b9a7fa1edf8eaaed4 10.09kB / 10.09kB  0.0s
 => => sha256:76ce212b9153cf6743484a31c2967fd5f284455f8ede2 1.17kB / 1.17kB  0.3s
 => => sha256:6b7a1245fe7148b56315cd314fe4764418e6e4ec1095 1.45MB / 1.45MB  10.1s
 => => sha256:d13ef786196545cd69aff1945929fc868712196e195bc 3.63kB / 3.63kB  0.0s
 => => sha256:919ca406a058a0c62788518f39a66257c122ace2007a 4.53MB / 4.53MB  45.8s
 => => extracting sha256:76ce212b9153cf6743484a31c2967fd5f284455f8ede227794  0.0s
 => => sha256:8064ffe06c65d307820b1373f7d39b16c32929f7f2c2 8.07MB / 8.07MB  64.3s
 => => sha256:4b5c59f2d82cd85059458a41c01ebe3b5a2c96f1d584 1.20MB / 1.20MB  18.0s
```

Run the PostgreSQL Container:

```
docker run --name postgres-container --network fullstack-network -v
pgdata:/var/lib/postgresql/data -d my-postgres-db
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app$ docker run --n
ame postgres-container --network fullstack-network -v pgdata:/var/lib/postgresql/d
ata -d my-postgres-db
a5ff6548822cb310d65eb5c20d763eb5efaf27cd72f88eb645e50ff5af4a27f0
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app$
```

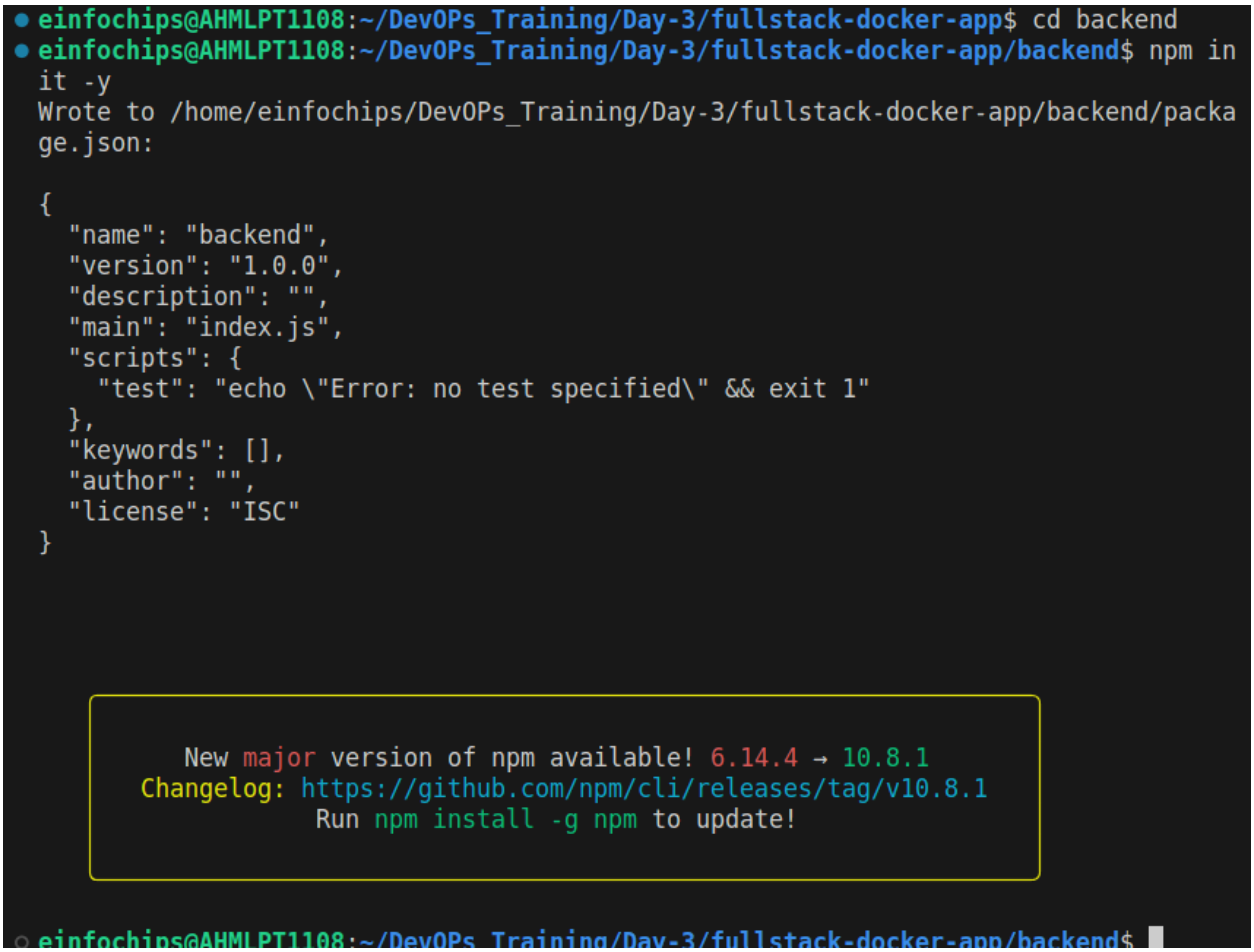
Part 3: Setting Up the Backend (Node.js with Express)

Objective: Create a Node.js application with Express and set it up with Docker.

Steps:

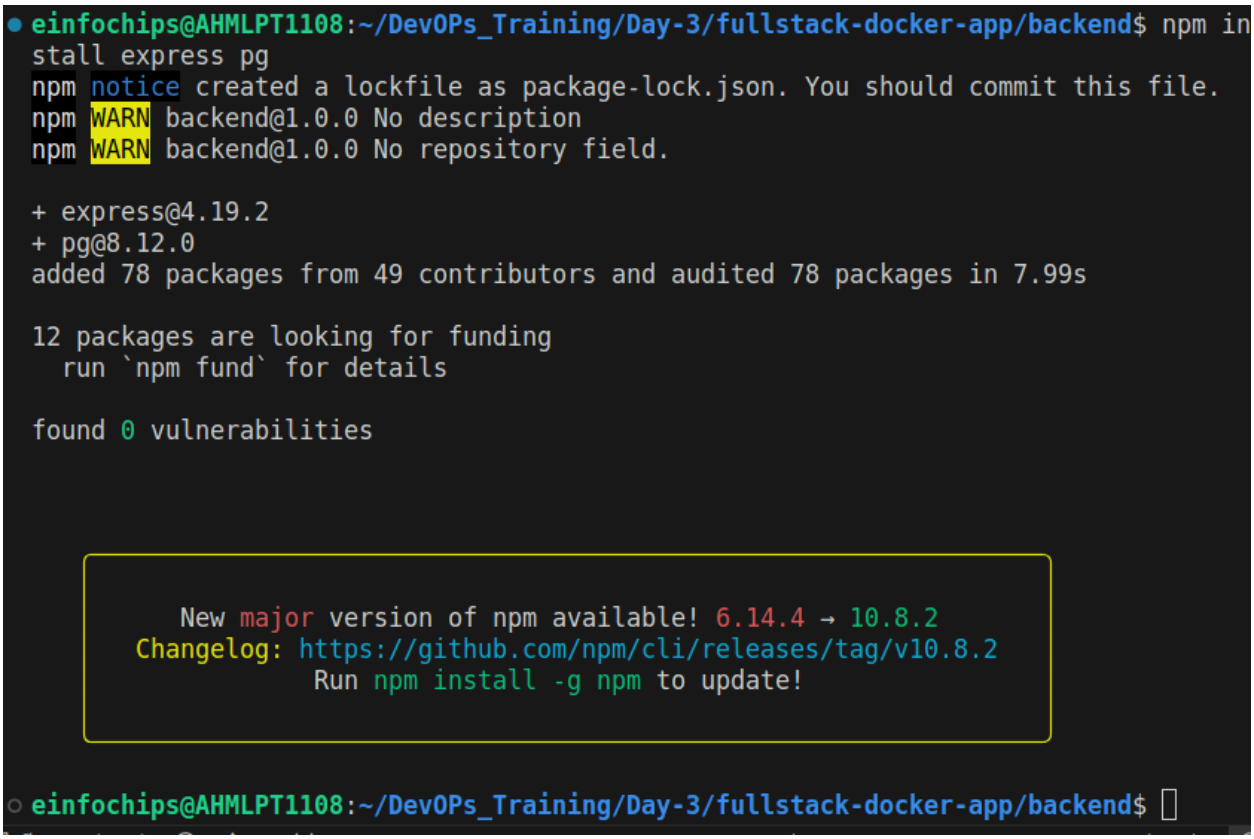
Initialize the Node.js Application:

```
cd backend
npm init -y
```



Install Express and pg (PostgreSQL client for Node.js):

```
npm install express pg
```



11. Create the Application Code:

In the `backend` directory, create a file named `index.js` with the following content:

```
const express = require('express');
const { Pool } = require('pg');
const app = express();
const port = 3000;

const pool = new Pool({
    user: 'user',
    host: 'postgres-container',
    database: 'mydatabase',
    password: 'password',
    port: 5432,
});

app.get('/', (req, res) => {
```

```
    res.send('Hello from Node.js and Docker!');
});


app.get('/data', async (req, res) => {
    const client = await pool.connect();
    const result = await client.query('SELECT NOW()');
    client.release();
    res.send(result.rows);
});


app.listen(port, () => {
    console.log(`App running on http://localhost:${port}`);
});
```

- **12. Create a Dockerfile for the Backend:**

In the `backend` directory, create a file named `Dockerfile` with the following content:

```
FROM node:latest


WORKDIR /usr/src/app


COPY package*.json ./
RUN npm install


COPY . .


EXPOSE 3000
CMD ["node", "index.js"]
```

- **Build the Backend Image:**

```
docker build -t my-node-app .
cd ..
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/backend$ docker
 build -t my-node-app .
[+] Building 7.7s (7/9)                                              docker:default
 => [internal] load build definition from dockerfile                          0.0s
 => => transferring dockerfile: 164B                                          0.0s
 => [internal] load metadata for docker.io/library/node:latest                0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/5] FROM docker.io/library/node:latest                                  0.1s
 => [internal] load build context                                             0.1s
 => => transferring context: 2.74MB                                           0.1s
 => [2/5] WORKDIR /usr/src/app                                                0.1s
 => [3/5] COPY package*.json ./                                               0.0s
 => [4/5] RUN npm install                                                     7.5s
 => => #  version of npm,
 => => # npm WARN old lockfile so supplemental metadata must be fetched from the
 => => # registry.
 => => # npm WARN old lockfile
 => => # npm WARN old lockfile This is a one-time fix-up, please be patient...
 => => # npm WARN old lockfile
```

Run the Backend Container:

```
docker run --name backend-container --network fullstack-network -d my-node-app
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app$ docker run --n
ame backend-container --network fullstack-network -d my-node-app
2bfc8a1bd5a62845e394ed6fc3e19d5a151da3659c1e594fe57067926b6b7d86
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app$
```

Part 4: Setting Up the Frontend (Nginx)

Objective: Create a simple static front-end and set it up with Docker.

Steps:

13. Create a Simple HTML Page:

In the `frontend` directory, create a file named `index.html` with the following content:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Hello from Nginx and Docker!</h1>
    <p>This is a simple static front-end served by Nginx.</p>
</body>
</html>
```

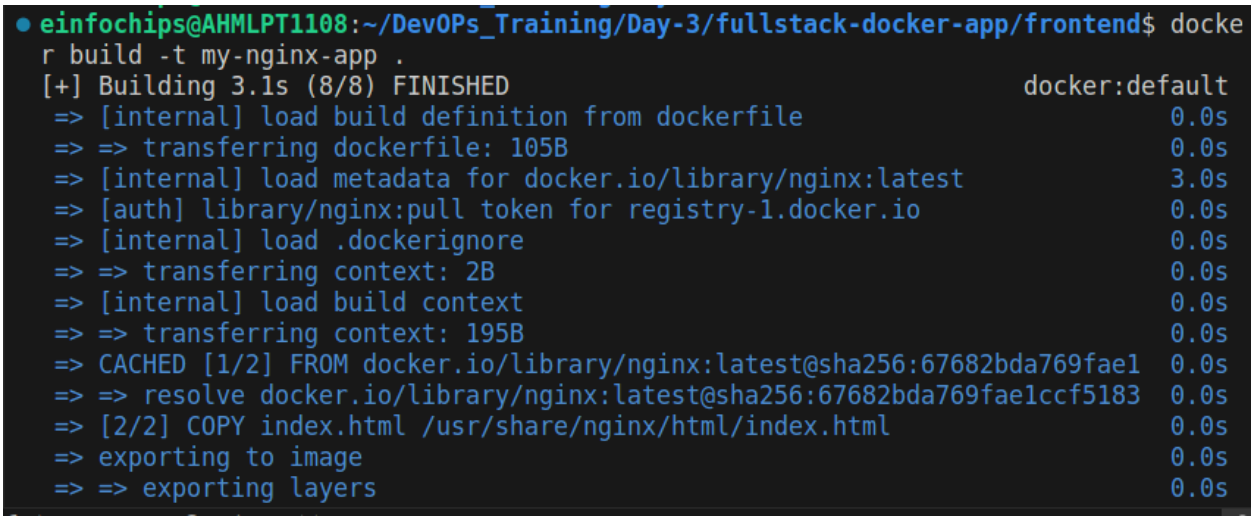
- **14. Create a Dockerfile for the Frontend:**

In the `frontend` directory, create a file named `Dockerfile` with the following content:

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
```

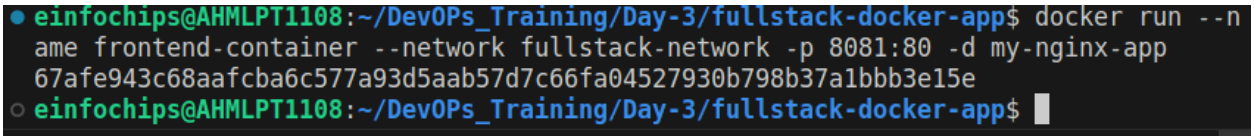
- **Build the Frontend Image:**

```
cd frontend
docker build -t my-nginx-app .
cd ..
```



Run the Frontend Container:

```
docker run --name frontend-container --network fullstack-network -p
8080:80 -d my-nginx-app
```



Part 5: Connecting the Backend and Database

Objective: Ensure the backend can communicate with the database and handle data requests.

Steps:

15. **Update Backend Code to Fetch Data from PostgreSQL:**
    - Ensure that the `index.js` code in the backend handles `/data` endpoint correctly as written above.
16. **Verify Backend Communication:**

Access the backend container:

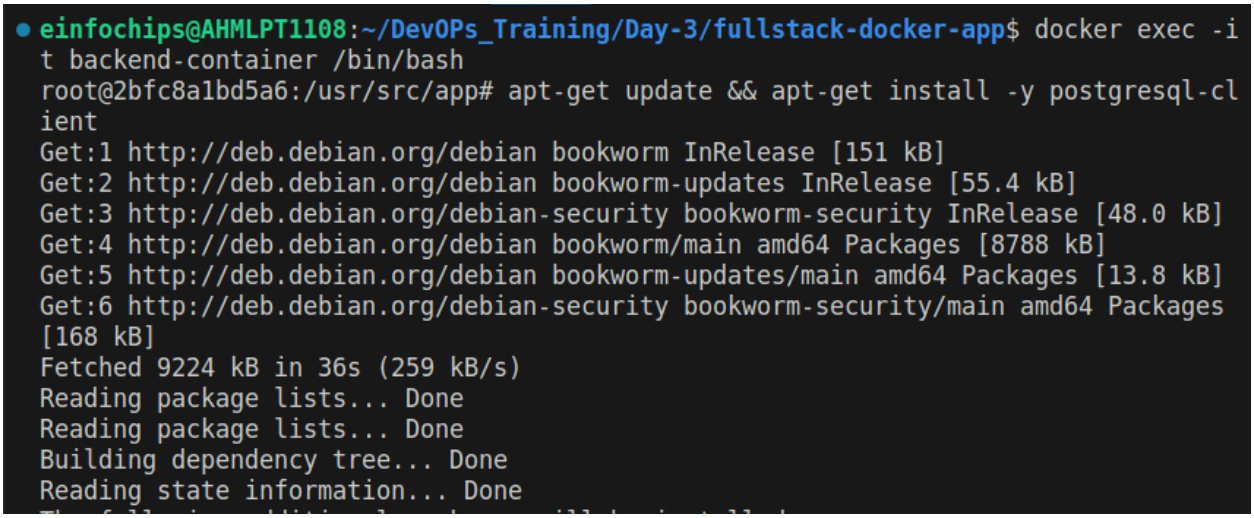
```
docker exec -it backend-container /bin/bash
```

Test the connection to the database using `psql`:

```
apt-get update && apt-get install -y postgresql-client
psql -h postgres-container -U user -d mydatabase -c "SELECT NOW();"
```

Exit the container:

```
exit
```



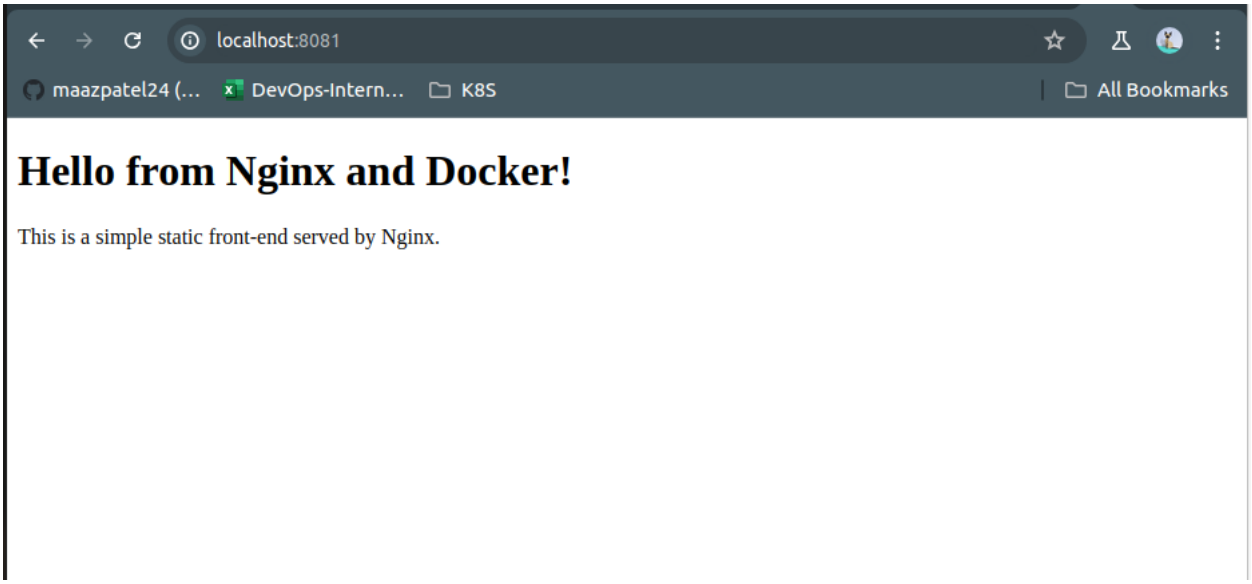
17. **Test the Backend API:**
    - Visit `http://localhost:3000` to see the basic message.
    - Visit `http://localhost:3000/data` to see the current date and time fetched from PostgreSQL.

Part 6: Final Integration and Testing

Objective: Ensure all components are working together and verify the full-stack application.

Steps:

18. **Access the Frontend:**
    - Visit `http://localhost:8080` in your browser. You should see the Nginx welcome page with the custom HTML.



19. **Verify Full Integration:**

Update the `index.html` to include a link to the backend:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Hello from Nginx and Docker!</h1>
    <p>This is a simple static front-end served by Nginx.</p>
    <a href="http://localhost:3000/data">Fetch Data from Backend</a>
```

```
</body>
</html>
```

-

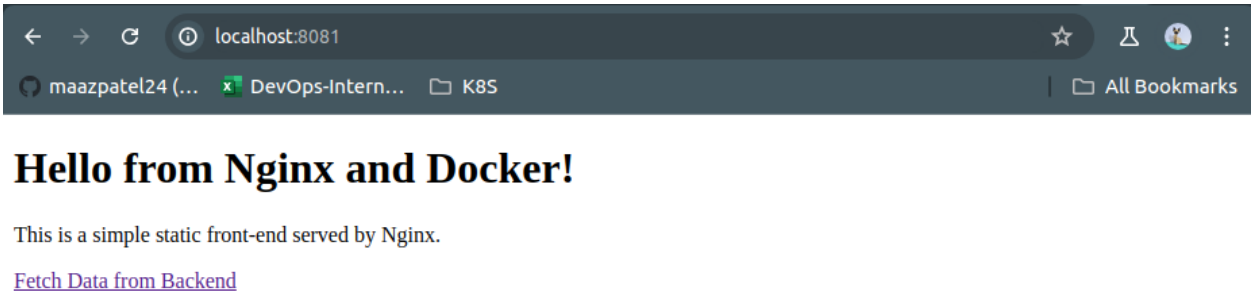
Rebuild and Run the Updated Frontend Container:

```
cd frontend
docker build -t my-nginx-app .
docker stop frontend-container
docker rm frontend-container
docker run --name frontend-container --network fullstack-network -p
8080:80 -d my-nginx-app
cd ..
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/frontend$ docke
r stop frontend-container
frontend-container
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/frontend$ docke
r rm frontend-container
frontend-container
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/frontend$ docke
r run --name frontend-container --network fullstack-network -p 8081:80 -d my-nginx
-app
be28296b0c9e0d63cbb2f03a1676ad217ffcde95a280ab6b0e745a5e908daa58
einfochips@AHMLPT1108:~/DevOPs_Training/Day-3/fullstack-docker-app/frontend$
```

20. Final Verification:

- Visit http://localhost:8080 and click the link to fetch data from the backend.



Part 7: Cleaning Up

Objective: Remove all created containers, images, networks, and volumes to clean up your environment.

Steps:

Stop and Remove the Containers:

```
docker stop frontend-container backend-container postgres-container
docker rm frontend-container backend-container postgres-container
```

21.

Remove the Images:

```
docker rmi my-nginx-app my-node-app my-postgres-db
```

22.

Remove the Network and Volume:

```
docker network rm fullstack-network
docker volume rm pgdata
```