Docker Project 01

Project Overview

In this project,we go through all three life cycles 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.

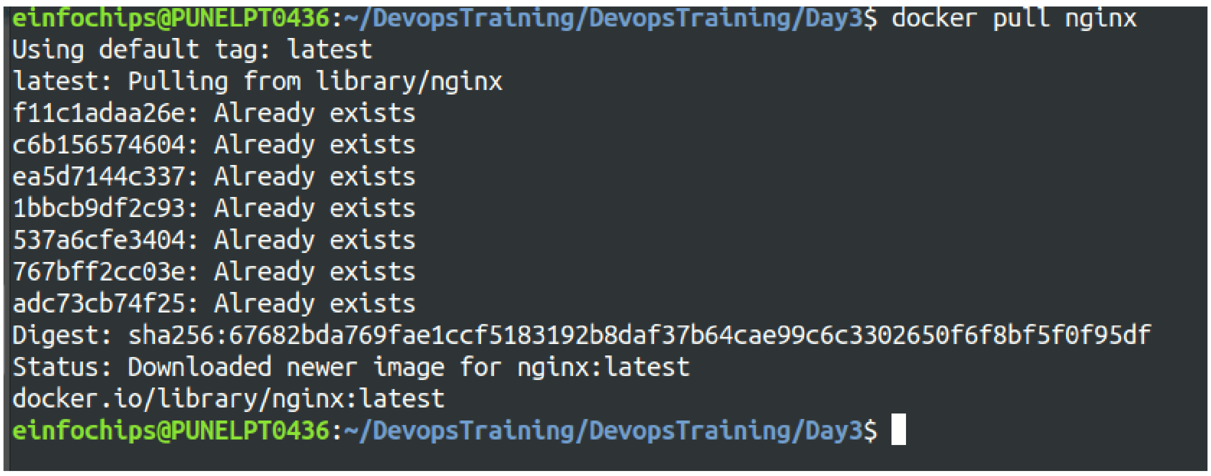
Part 1: Creating a Container from a Pulled Image

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

Steps:

Pull the Nginx Image:

docker pull nginx



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
f11c1adaa26e: Already exists
c6b156574604: Already exists
ea5d7144c337: Already exists
1bbcb9df2c93: Already exists
537a6cfe3404: Already exists
767bff2cc03e: Already exists
adc73cb74f25: Already exists
Digest: sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$
```

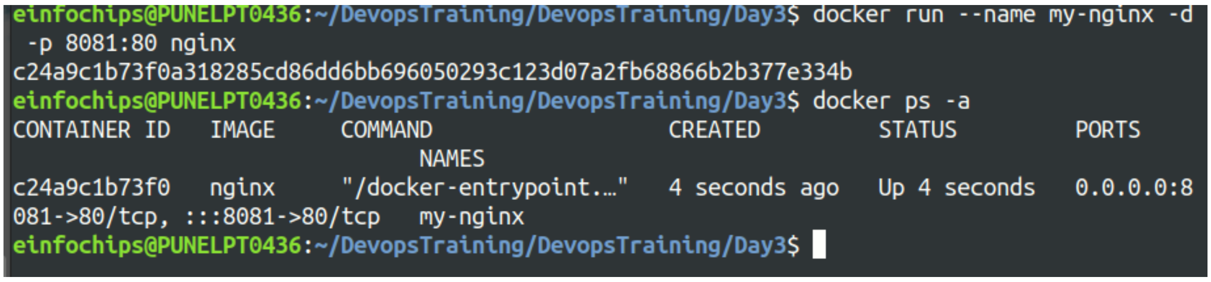
1. **Run the Nginx Container:**

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

   - --name my-nginx: Assigns a name to the container.
   - -d: Run the container in detached mode.
   - -p 8081:80: Maps port 8080 on your host to port 80 in the container.

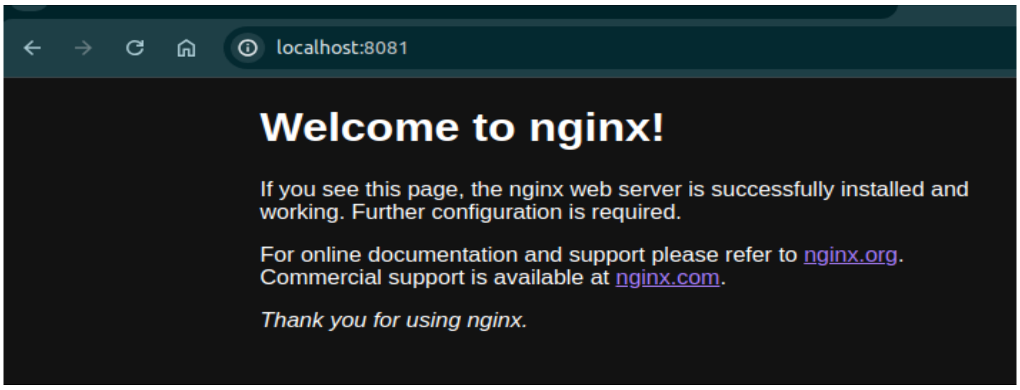
2. **Verify the Container is Running:**

   docker ps



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker run --name my-nginx -d
 -p 8081:80 nginx
c24a9c1b73f0a318285cd86dd6bb696050293c123d07a2fb68866b2b377e334b
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS
                NAMES
c24a9c1b73f0   nginx     "/docker-entrypoint.…"   4 seconds ago   Up 4 seconds   0.0.0.0:8
081->80/tcp, :::8081->80/tcp   my-nginx
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$
```

Visit http://localhost:8081 in your browser. You should see the Nginx welcome page.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

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:

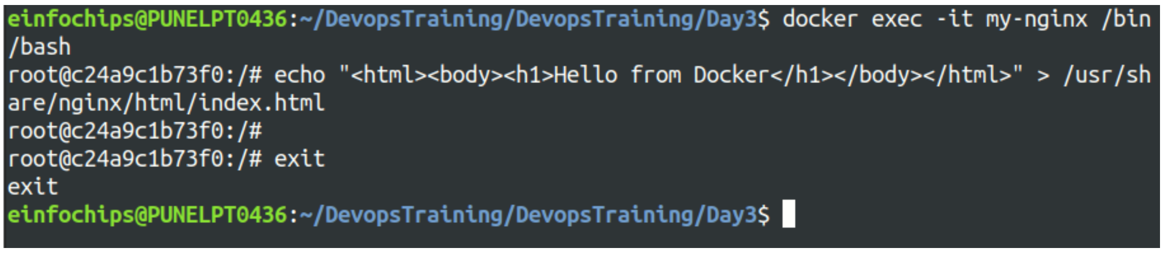
```
docker exec -it my-nginx /bin/bash
```

1. **Create a Custom HTML Page:**

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

2. **Exit the Container:**

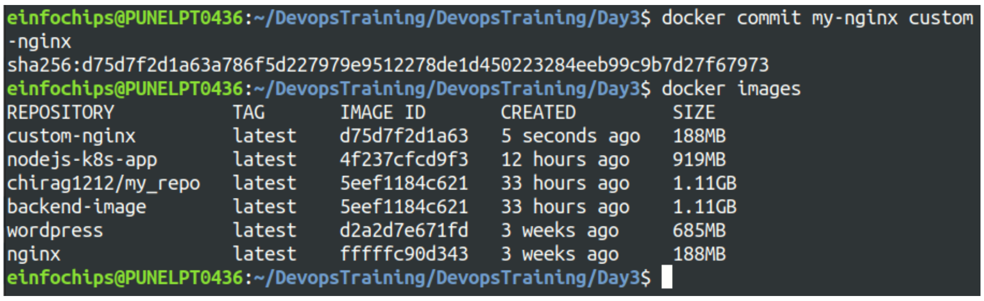
   ```
   exit
   ```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker exec -it my-nginx /bin
/bash
root@c24a9c1b73f0:/# echo "<html><body><h1>Hello from Docker</h1></body></html>" > /usr/sh
are/nginx/html/index.html
root@c24a9c1b73f0:/#
root@c24a9c1b73f0:/# exit
exit
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$
```

3. **Commit the Changes to Create a New Image:**

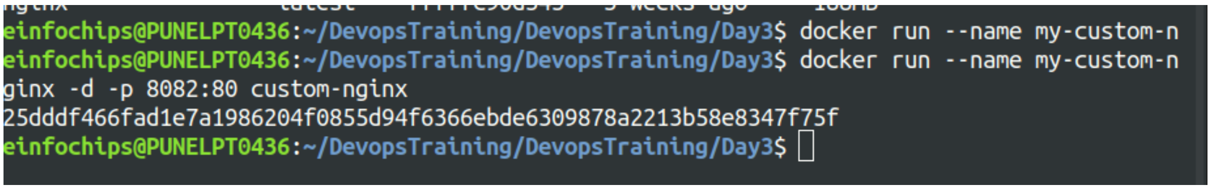
   docker commit my-nginx custom-nginx



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker commit my-nginx custom
-nginx
sha256:d75d7f2d1a63a786f5d227979e9512278de1d450223284eeb99c9b7d27f67973
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
custom-nginx        latest    d75d7f2d1a63   5 seconds ago    188MB
nodejs-k8s-app      latest    4f237cfcd9f3   12 hours ago     919MB
chirag1212/my_repo  latest    5eef1184c621   33 hours ago     1.11GB
backend-image       latest    5eef1184c621   33 hours ago     1.11GB
wordpress           latest    d2a2d7e671fd   3 weeks ago      685MB
nginx               latest    fffffc90d343   3 weeks ago      188MB
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$
```

4. **Run a Container from the New Image:**

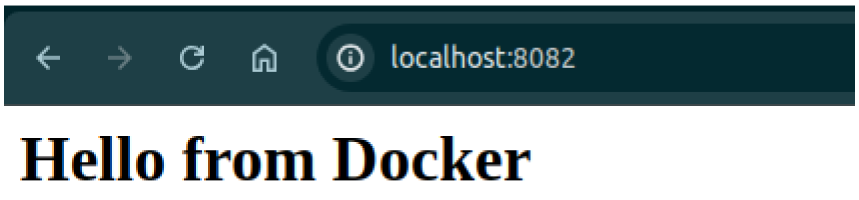
   docker run --name my-custom-nginx -d -p 8082:80 custom-nginx



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker run --name my-custom-n
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$ docker run --name my-custom-n
ginx -d -p 8082:80 custom-nginx
25dddf466fad1e7a1986204f0855d94f6366ebde6309878a2213b58e8347f75f
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3$
```

5. **Verify the New Container:**

   ○ Visit http://localhost:8082 in your browser. You should see your custom HTML page.



localhost:8082

Hello from Docker

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:

1. **Create a Project Directory:**

   mkdir my-webapp

   cd my-webapp

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

   Create an index.html file:

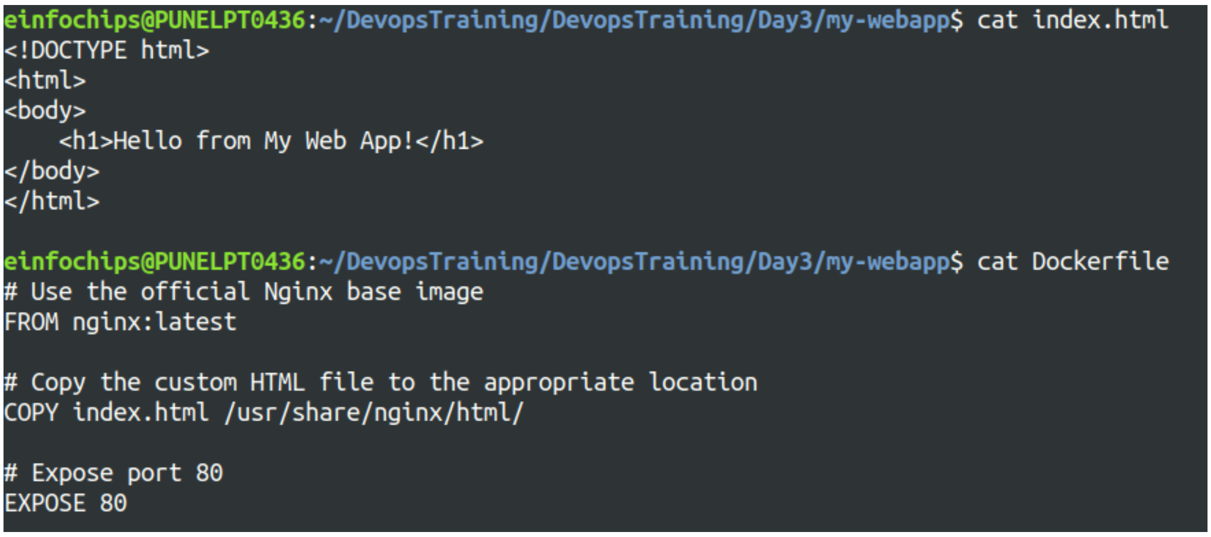
   ```
   <!DOCTYPE html>
   <html>
   <body>
      <h1>Hello from My Web App!</h1>
   </body>
   </html>
   ```

   - Save this file in the my-webapp directory.

3. **Write the Dockerfile:**

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

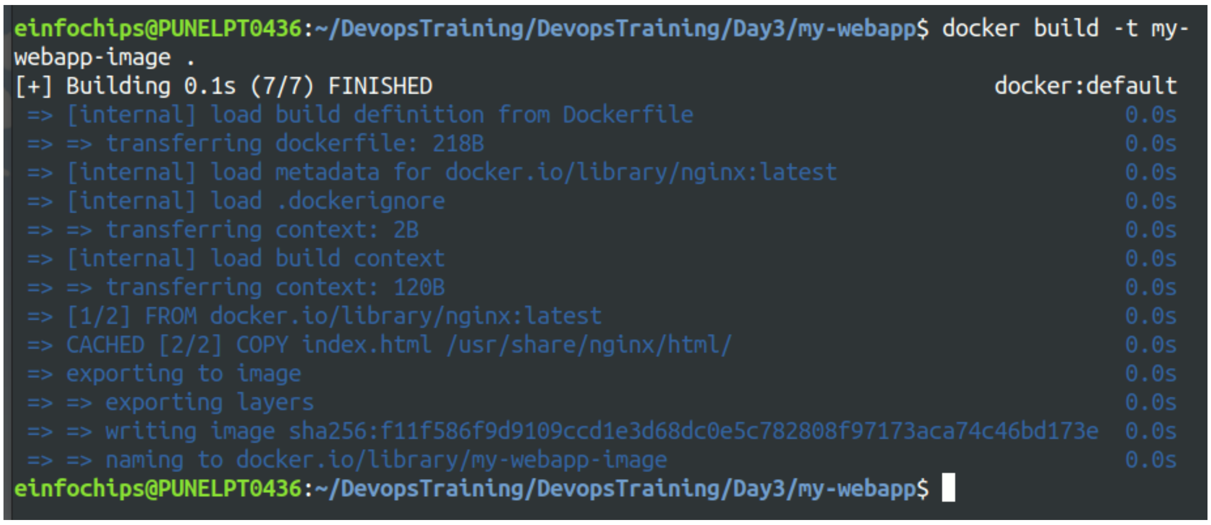
   ```
   # Use the official Nginx base image
   FROM nginx:latest
   # Copy the custom HTML file to the appropriate location
   COPY index.html /usr/share/nginx/html/
   # Expose port 80
   EXPOSE 80
   ```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ cat index.html
<!DOCTYPE html>
<html>
<body>
    <h1>Hello from My Web App!</h1>
</body>
</html>

einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ cat Dockerfile
# 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
```

4. **Build the Docker Image:**

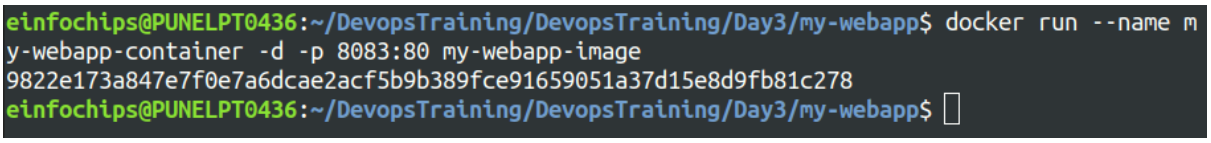
   ```
   docker build -t my-webapp-image .
   ```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker build -t my-
webapp-image .
[+] Building 0.1s (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
 => [internal] load build context                                                   0.0s
 => => transferring context: 120B                                                   0.0s
 => [1/2] FROM docker.io/library/nginx:latest                                       0.0s
 => CACHED [2/2] COPY index.html /usr/share/nginx/html/                             0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:f11f586f9d9109ccd1e3d68dc0e5c782808f97173aca74c46bd173e 0.0s
 => => naming to docker.io/library/my-webapp-image                                  0.0s
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$
```

5. **Run a Container from the Built Image:**

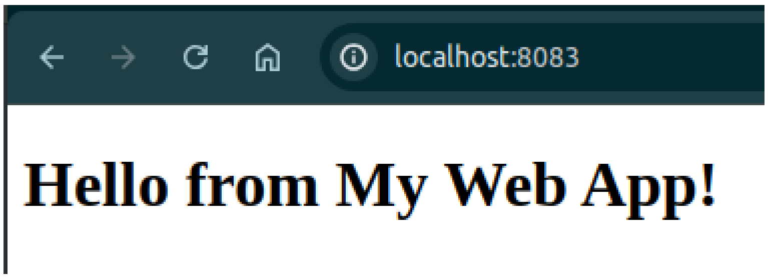
   docker run --name my-webapp-container -d -p 8083:80 my-webapp-image



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker run --name m
y-webapp-container -d -p 8083:80 my-webapp-image
9822e173a847e7f0e7a6dcae2acf5b9b389fce91659051a37d15e8d9fb81c278
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$
```

6. **Verify the Web Application:**

   ○ Visit http://localhost:8083 in your browser. You should see your custom web application.



localhost:8083

Hello from My Web App!

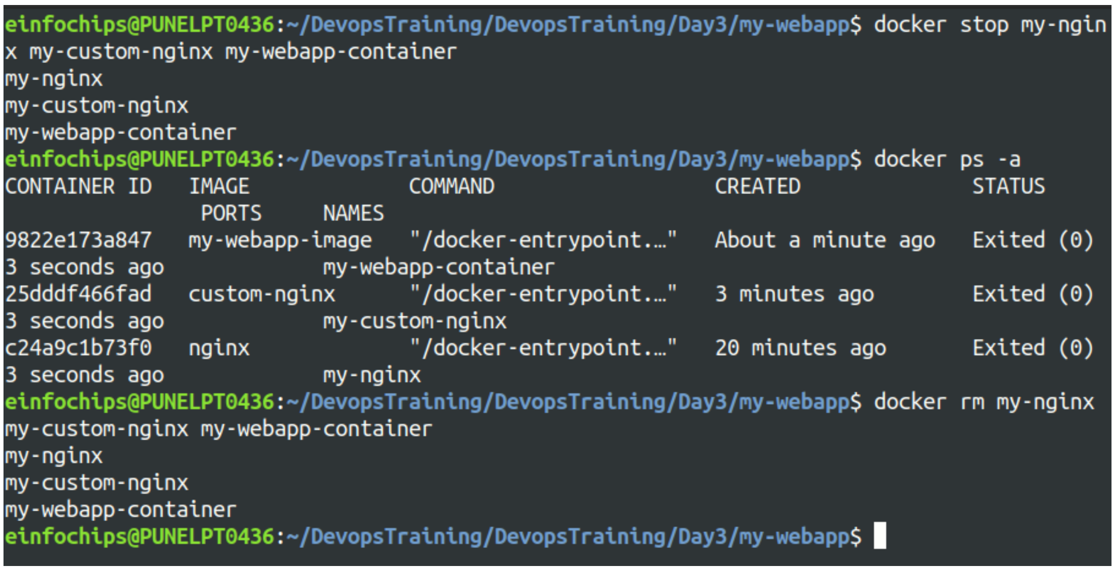
Part 4: Cleaning Up

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

Steps:

Stop and Remove the Containers:

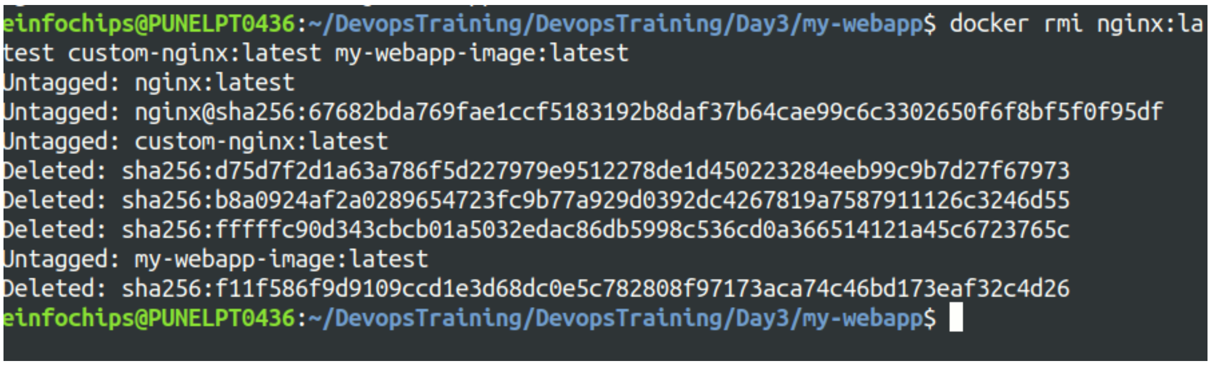
docker stop my-nginx my-custom-nginx my-webapp-container docker rm my-nginx my-custom-nginx my-webapp-container



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker stop my-ngin
x my-custom-nginx my-webapp-container
my-nginx
my-custom-nginx
my-webapp-container
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED              STATUS
                PORTS     NAMES
9822e173a847   my-webapp-image   "/docker-entrypoint.…"   About a minute ago   Exited (0)
3 seconds ago             my-webapp-container
25dddf466fad   custom-nginx      "/docker-entrypoint.…"   3 minutes ago        Exited (0)
3 seconds ago             my-custom-nginx
c24a9c1b73f0   nginx             "/docker-entrypoint.…"   20 minutes ago       Exited (0)
3 seconds ago             my-nginx
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker rm my-nginx
my-custom-nginx my-webapp-container
my-nginx
my-custom-nginx
my-webapp-container
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$
```

1. **Remove the Images:**

   docker rmi nginx custom-nginx my-webapp-image



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/my-webapp$ docker rmi nginx:la
test custom-nginx:latest my-webapp-image:latest
Untagged: nginx:latest
Untagged: nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Untagged: custom-nginx:latest
Deleted: sha256:d75d7f2d1a63a786f5d227979e9512278de1d450223284eeb99c9b7d27f67973
Deleted: sha256:b8a0924af2a0289654723fc9b77a929d0392dc4267819a7587911126c3246d55
Deleted: sha256:fffffc90d343cbcb01a5032edac86db5998c536cd0a366514121a45c6723765c
Untagged: my-webapp-image:latest
Deleted: sha256:f11f586f9d9109ccd1e3d68dc0e5c782808f97173aca74c46bd173eaf32c4d26
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/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
```

1.

Create Subdirectories for Each Service:

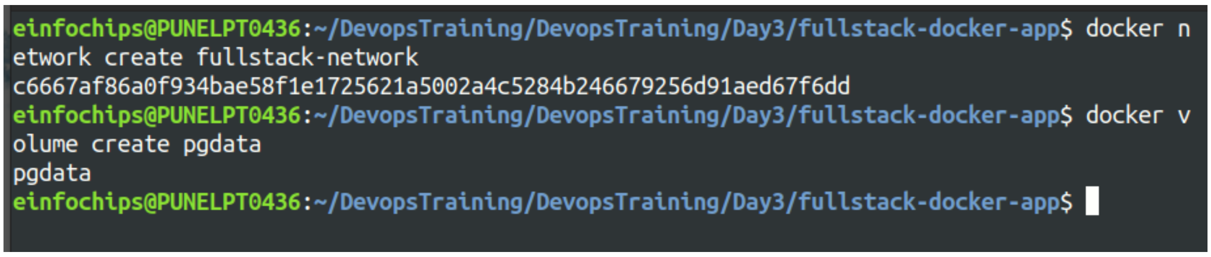
```
mkdir frontend backend database
```

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

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

3.
   - Create a volume for the PostgreSQL database.

```
docker volume create pgdata
```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app$ docker n
etwork create fullstack-network
c6667af86a0f934bae58f1e1725621a5002a4c5284b246679256d91aed67f6dd
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app$ docker v
olume create pgdata
pgdata
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app$
```

Part 2: Setting Up the Database

Objective: Set up a PostgreSQL database with Docker.

Steps:

1. **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
```

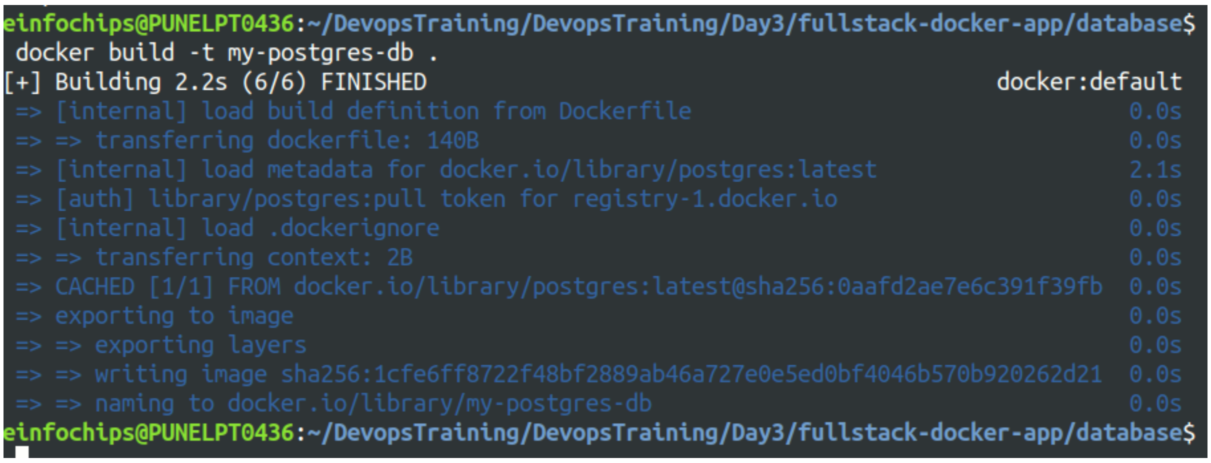
   -

Build the PostgreSQL Image:

cd database

docker build -t my-postgres-db .

Cd ..

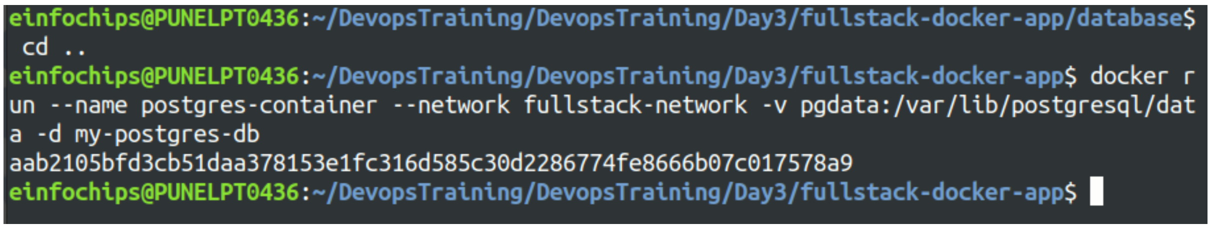


```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/database$
 docker build -t my-postgres-db .
[+] Building 2.2s (6/6) FINISHED                                         docker:default
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 140B                                                0.0s
 => [internal] load metadata for docker.io/library/postgres:latest                  2.1s
 => [auth] library/postgres:pull token for registry-1.docker.io                     0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => CACHED [1/1] FROM docker.io/library/postgres:latest@sha256:0aafd2ae7e6c391f39fb  0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:1cfe6ff8722f48bf2889ab46a727e0e5ed0bf4046b570b920262d21  0.0s
 => => naming to docker.io/library/my-postgres-db                                   0.0s
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/database$
```

2.

Run the PostgreSQL Container:

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



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/database$
 cd ..
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app$ docker r
un --name postgres-container --network fullstack-network -v pgdata:/var/lib/postgresql/dat
a -d my-postgres-db
aab2105bfd3cb51daa378153e1fc316d585c30d2286774fe8666b07c017578a9
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/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:

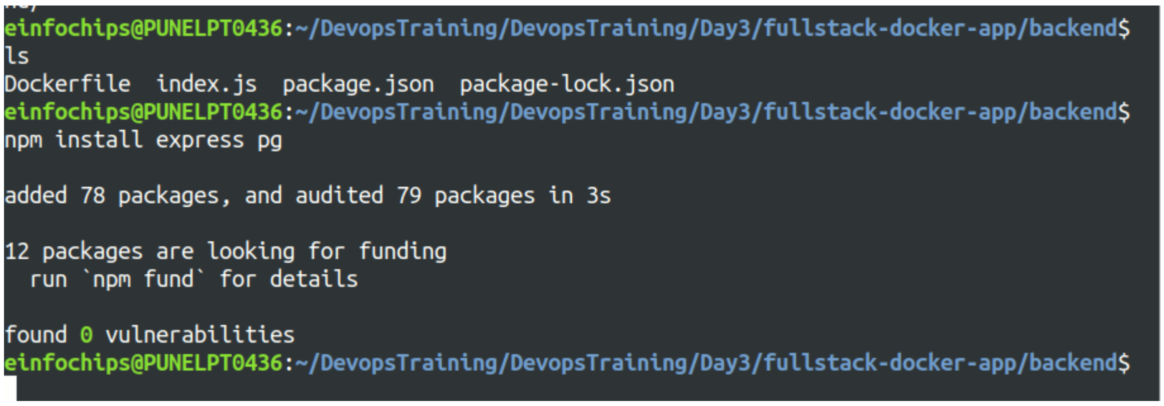
cd backend

npm init -y

1.

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

npm install express pg



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$ ls
Dockerfile  index.js  package.json  package-lock.json
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$ npm install express pg

added 78 packages, and audited 79 packages in 3s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$
```

2. **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}`);
});
```

◦

3. **Create a Dockerfile for the Backend:**

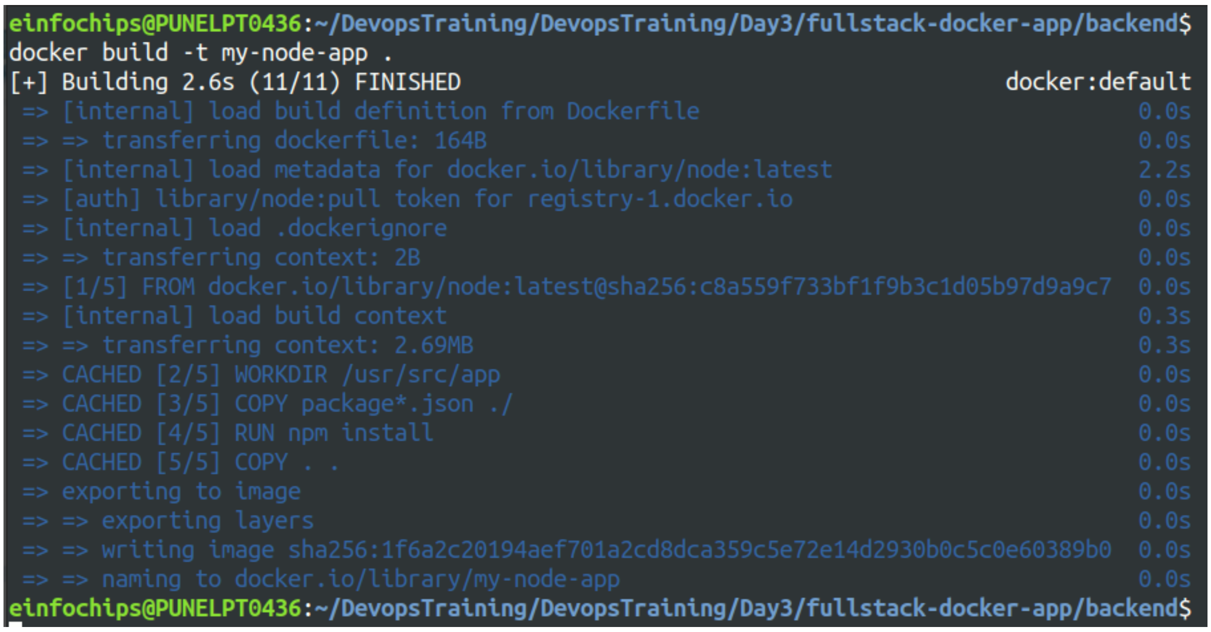
In the backend directory, create a file named Dockerfile with the following content:

```
FROM node:latest

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install

COPY . .

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

◦

Build the Backend Image:

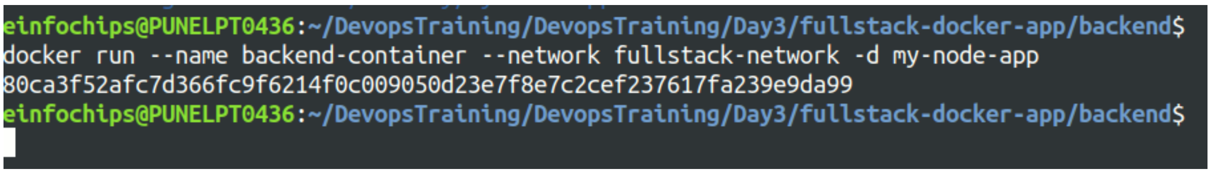
```
docker build -t my-node-app .
Cd ..
```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$
docker build -t my-node-app .
[+] Building 2.6s (11/11) FINISHED                                           docker:default
 => [internal] load build definition from Dockerfile                                   0.0s
 => => transferring dockerfile: 164B                                                   0.0s
 => [internal] load metadata for docker.io/library/node:latest                         2.2s
 => [auth] library/node:pull token for registry-1.docker.io                            0.0s
 => [internal] load .dockerignore                                                      0.0s
 => => transferring context: 2B                                                        0.0s
 => [1/5] FROM docker.io/library/node:latest@sha256:c8a559f733bf1f9b3c1d05b97d9a9c7  0.0s
 => [internal] load build context                                                      0.3s
 => => transferring context: 2.69MB                                                    0.3s
 => CACHED [2/5] WORKDIR /usr/src/app                                                  0.0s
 => CACHED [3/5] COPY package*.json ./                                                 0.0s
 => CACHED [4/5] RUN npm install                                                       0.0s
 => CACHED [5/5] COPY . .                                                              0.0s
 => exporting to image                                                                 0.0s
 => => exporting layers                                                                0.0s
 => => writing image sha256:1f6a2c20194aef701a2cd8dca359c5e72e14d2930b0c5c0e60389b0  0.0s
 => => naming to docker.io/library/my-node-app                                         0.0s
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$
```

Run the Backend Container:

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



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$
docker run --name backend-container --network fullstack-network -d my-node-app
80ca3f52afc7d366fc9f6214f0c009050d23e7f8e7c2cef237617fa239e9da99
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/backend$
```

Part 4: Setting Up the Frontend (Nginx)

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

Steps:

1. **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>
```

○

2. **Create a Dockerfile for the Frontend:**

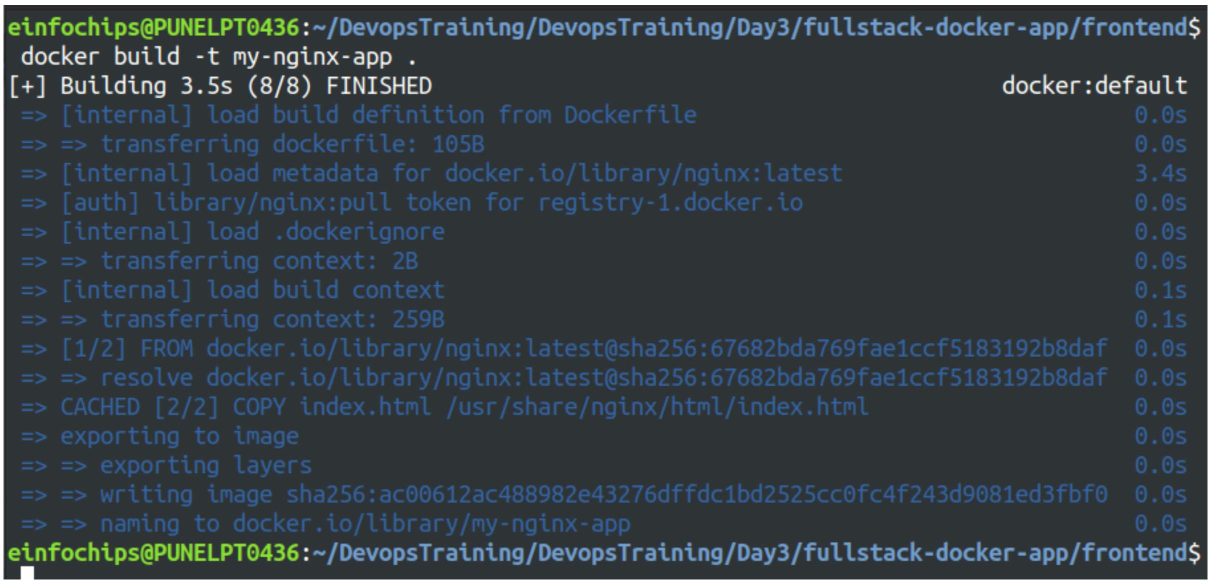
In the frontend directory, create a file named Dockerfile with the following content:

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html
```

○

Build the Frontend Image:

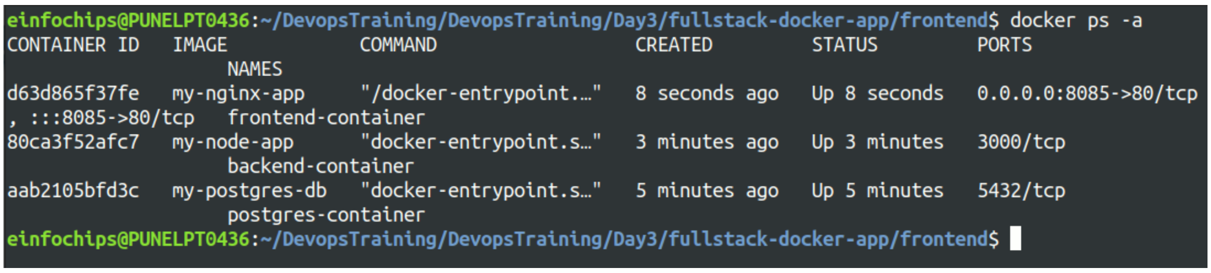
```
cd frontend

docker build -t my-nginx-app .

Cd ..
```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$
 docker build -t my-nginx-app .
[+] Building 3.5s (8/8) FINISHED                                          docker:default
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 105B                                                0.0s
 => [internal] load metadata for docker.io/library/nginx:latest                     3.4s
 => [auth] library/nginx:pull token for registry-1.docker.io                        0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load build context                                                   0.1s
 => => transferring context: 259B                                                   0.1s
 => [1/2] FROM docker.io/library/nginx:latest@sha256:67682bda769fae1ccf5183192b8daf 0.0s
 => => resolve docker.io/library/nginx:latest@sha256:67682bda769fae1ccf5183192b8daf 0.0s
 => CACHED [2/2] COPY index.html /usr/share/nginx/html/index.html                   0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:ac00612ac488982e43276dffdc1bd2525cc0fc4f243d9081ed3fbf0 0.0s
 => => naming to docker.io/library/my-nginx-app                                     0.0s
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$
```

Run the Frontend Container:

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



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker ps -a
CONTAINER ID   IMAGE            COMMAND                  CREATED         STATUS         PORTS
                  NAMES
d63d865f37fe   my-nginx-app     "/docker-entrypoint.…"   8 seconds ago   Up 8 seconds   0.0.0.0:8085->80/tcp
, :::8085->80/tcp   frontend-container
80ca3f52afc7   my-node-app      "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   3000/tcp
                  backend-container
aab2105bfd3c   my-postgres-db   "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   5432/tcp
                  postgres-container
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$
```

Part 5: Connecting the Backend and Database

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

Steps:

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

2. **Verify Backend Communication:**

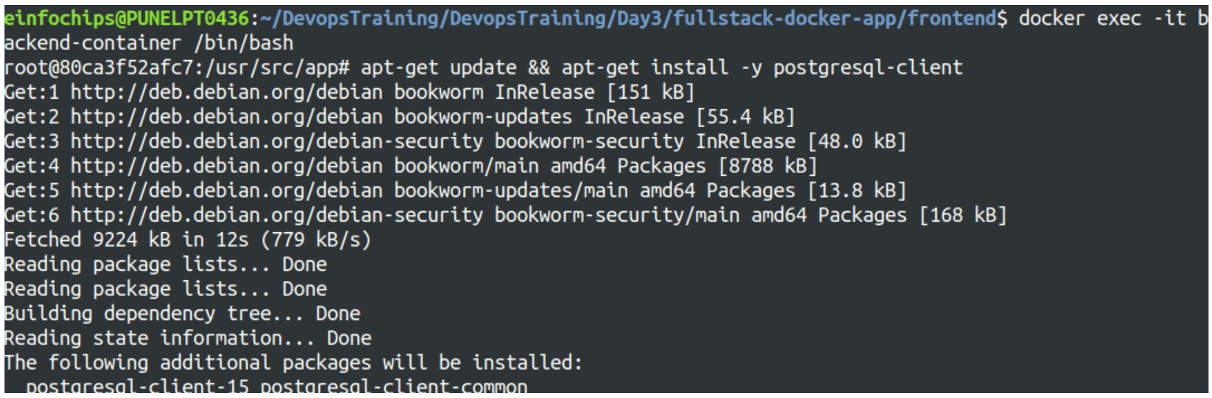
Access the backend container:

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

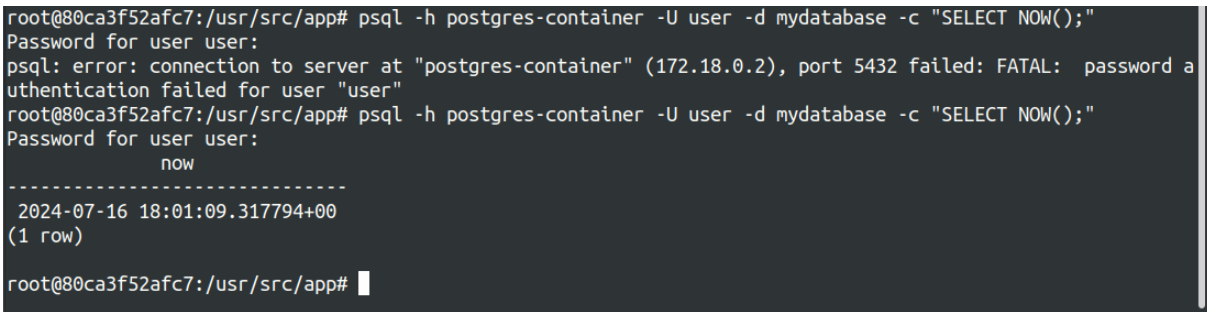
Test the connection to the database using psql:

```
apt-get update && apt-get install -y postgresql-client
```

```
psql -h postgres-container -U user -d mydatabase -c "SELECT NOW();"
```



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker exec -it b
ackend-container /bin/bash
root@80ca3f52afc7:/usr/src/app# apt-get update && apt-get install -y postgresql-client
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8788 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.8 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [168 kB]
Fetched 9224 kB in 12s (779 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  postgresql-client-15 postgresql-client-common
```



```
root@80ca3f52afc7:/usr/src/app# psql -h postgres-container -U user -d mydatabase -c "SELECT NOW();"
Password for user user:
psql: error: connection to server at "postgres-container" (172.18.0.2), port 5432 failed: FATAL:  password a
uthentication failed for user "user"
root@80ca3f52afc7:/usr/src/app# psql -h postgres-container -U user -d mydatabase -c "SELECT NOW();"
Password for user user:
              now
-------------------------------
 2024-07-16 18:01:09.317794+00
(1 row)

root@80ca3f52afc7:/usr/src/app# 
```

Exit the container:

```
exit
```

3. **Test the Backend API:**
    - Visit http://localhost:3000 to see the basic message.

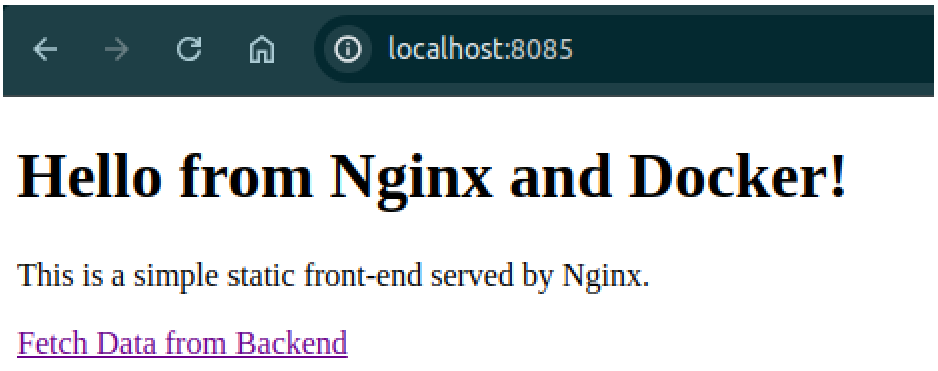
- Visit http://localhost:3000/data to see the current date and time fetched from PostgreSQL.

Part 6: Final Integration and Testing

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

Steps:

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



Hello from Nginx and Docker!

This is a simple static front-end served by Nginx.

Fetch Data from Backend

2. **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 ..

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

Part 7: Cleaning Up

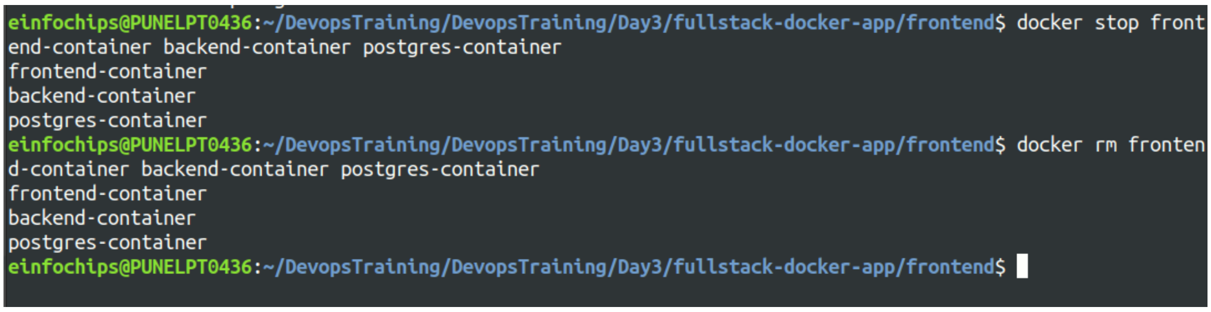
Objective: Remove all created containers, images, networks, and volumes to clean up your environment.

Steps:

Stop and Remove the Containers:

docker stop frontend-container backend-container postgres-container

docker rm frontend-container backend-container postgres-container

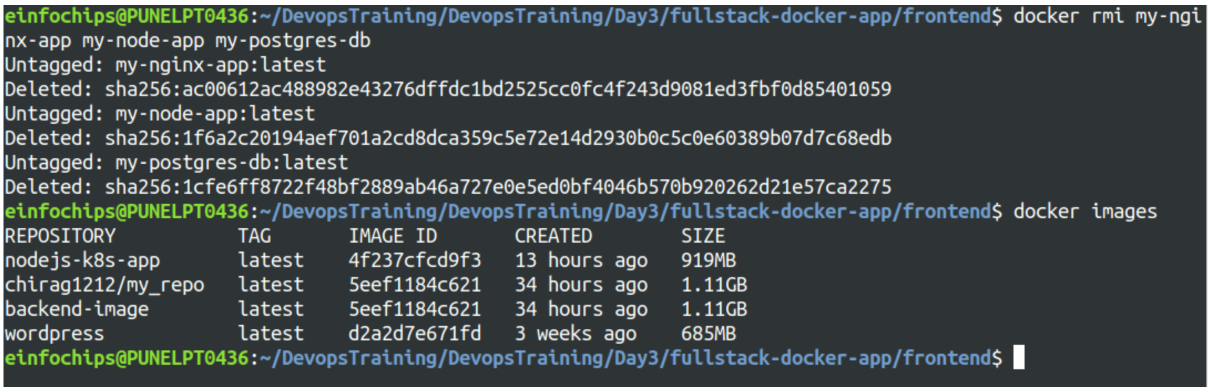


```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker stop front
end-container backend-container postgres-container
frontend-container
backend-container
postgres-container
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker rm fronten
d-container backend-container postgres-container
frontend-container
backend-container
postgres-container
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$
```

1.

Remove the Images:

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



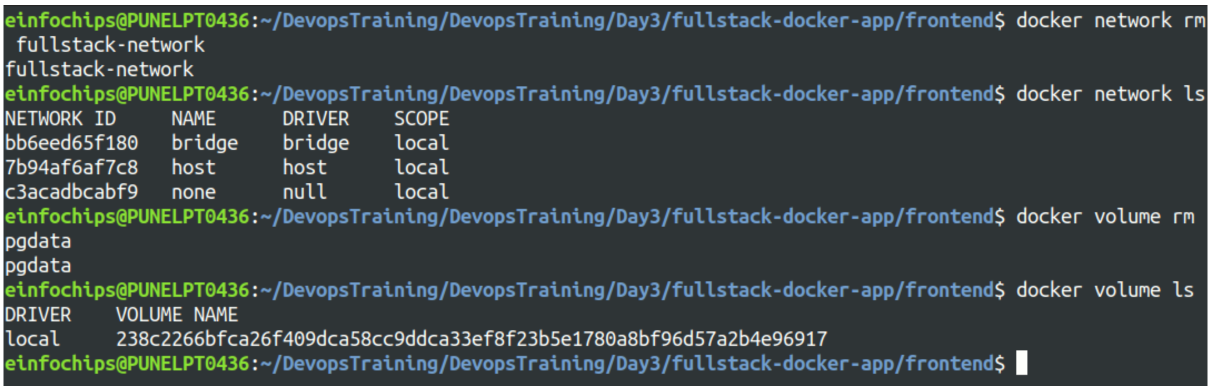
```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker rmi my-ngi
nx-app my-node-app my-postgres-db
Untagged: my-nginx-app:latest
Deleted: sha256:ac00612ac488982e43276dffdc1bd2525cc0fc4f243d9081ed3fbf0d85401059
Untagged: my-node-app:latest
Deleted: sha256:1f6a2c20194aef701a2cd8dca359c5e72e14d2930b0c5c0e60389b07d7c68edb
Untagged: my-postgres-db:latest
Deleted: sha256:1cfe6ff8722f48bf2889ab46a727e0e5ed0bf4046b570b920262d21e57ca2275
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
nodejs-k8s-app      latest    4f237cfcd9f3   13 hours ago    919MB
chirag1212/my_repo  latest    5eef1184c621   34 hours ago    1.11GB
backend-image       latest    5eef1184c621   34 hours ago    1.11GB
wordpress           latest    d2a2d7e671fd   3 weeks ago     685MB
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$
```

2.

Remove the Network and Volume:

docker network rm fullstack-network

docker volume rm pgdata



```
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker network rm
 fullstack-network
fullstack-network
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
bb6eed65f180   bridge    bridge    local
7b94af6af7c8   host      host      local
c3acadbcabf9   none      null      local
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker volume rm
pgdata
pgdata
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ docker volume ls
DRIVER    VOLUME NAME
local     238c2266bfca26f409dca58cc9ddca33ef8f23b5e1780a8bf96d57a2b4e96917
einfochips@PUNELPT0436:~/DevopsTraining/DevopsTraining/Day3/fullstack-docker-app/frontend$ 
```