**Day 3 Docker Project 1:  
  
Project Overview**

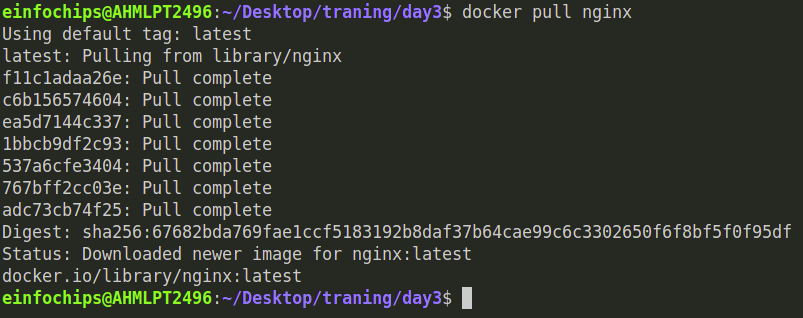
In this project, you'll go through all three lifecycles of Docker: pulling an image and creating a container, modifying the container and creating a new image, and finally, creating a Dockerfile to build and deploy a web application.

### **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

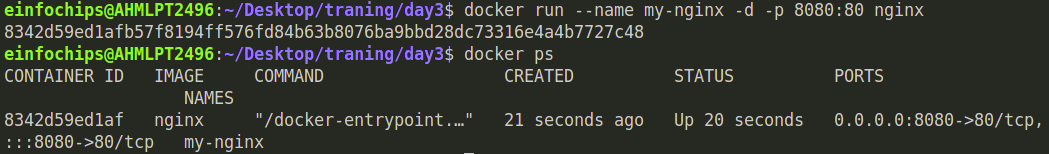
****

**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.

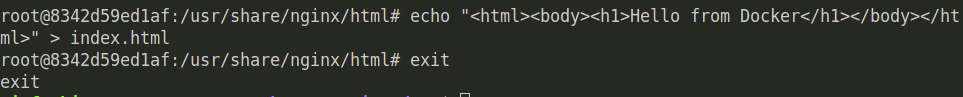
**Verify the Container is Running:**  
docker ps

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

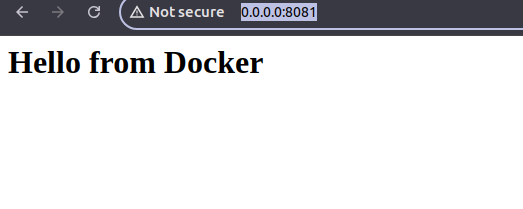


**Access the Running Container:**  
docker exec -it my-nginx /bin/bash

**Create a Custom HTML Page:**  
echo "<html><body><h1>Hello from Docker!</h1></body></html>" > /usr/share/nginx/html/index.html

  
**Commit the Changes to Create a New Image:**  
docker commit my-nginx custom-nginx

**Run a Container from the New Image:**  
docker run --name my-custom-nginx -d -p 8081:80 custom-nginx

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.

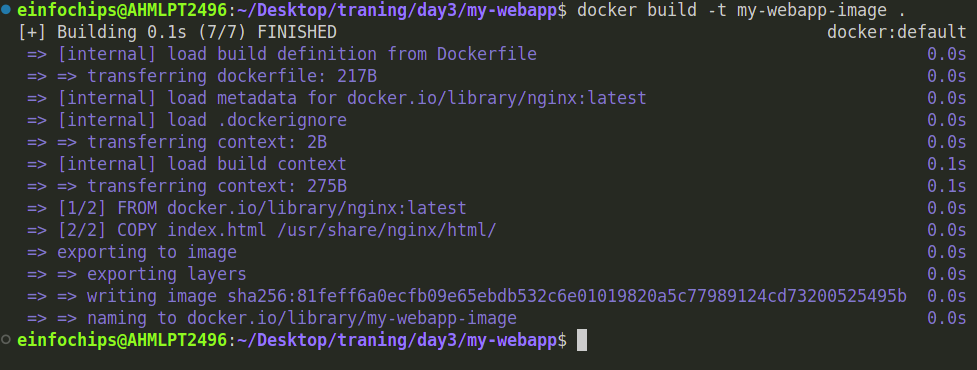
**Steps:**

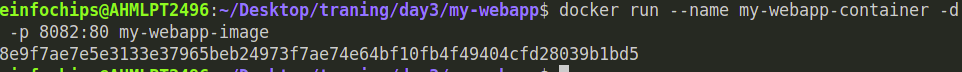
**Create a Project Directory:**  
mkdir my-webapp

cd my-webapp

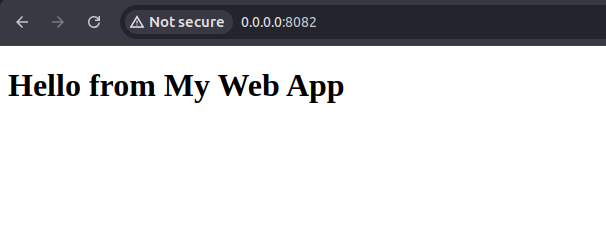
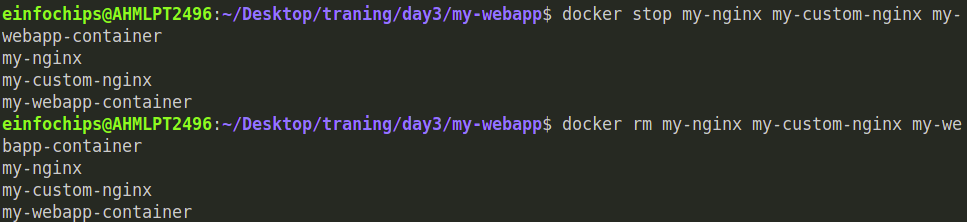


**Build the Docker Image:**  
docker build -t my-webapp-image .

  
**Run a Container from the Built Image:**  
docker run --name my-webapp-container -d -p 8082:80 my-webapp-image



**Verify the Web Application:**

Visit http://localhost:8082 in your browser. You should see your custom web application  
  
**Part 4: Cleaning Up**  


**Docker project 02:**

### **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

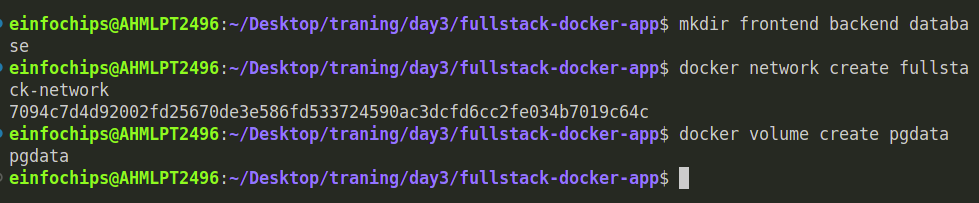
**Create Subdirectories for Each Service:**  
mkdir frontend backend database

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

docker network create fullstack-network

* + Create a volume for the PostgreSQL database.

docker volume create pgdata

  
**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

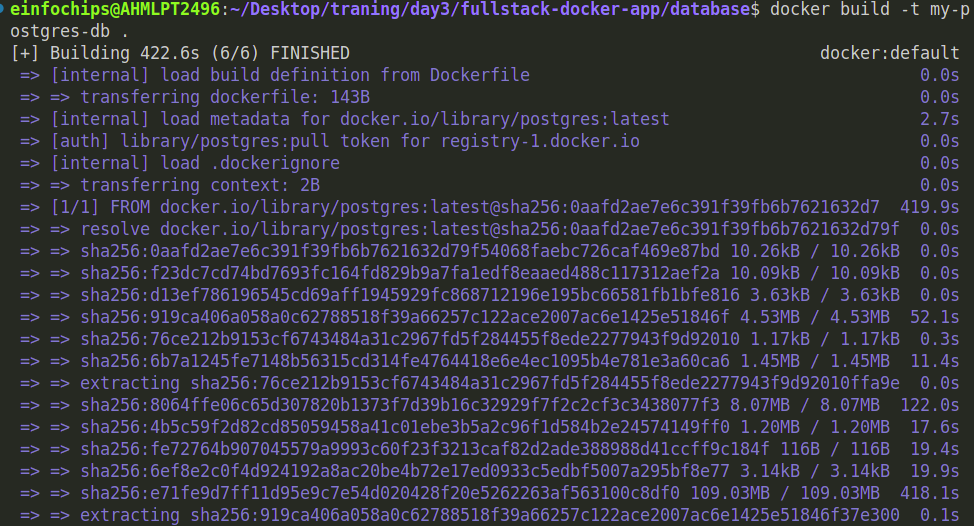
ENV POSTGRES\_PASSWORD=password

ENV POSTGRES\_DB=mydatabase

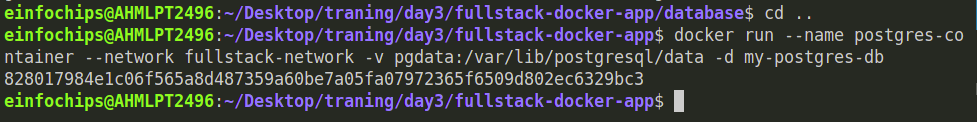
**Build the PostgreSQL Image:**  
cd database

docker build -t my-postgres-db .

cd ..

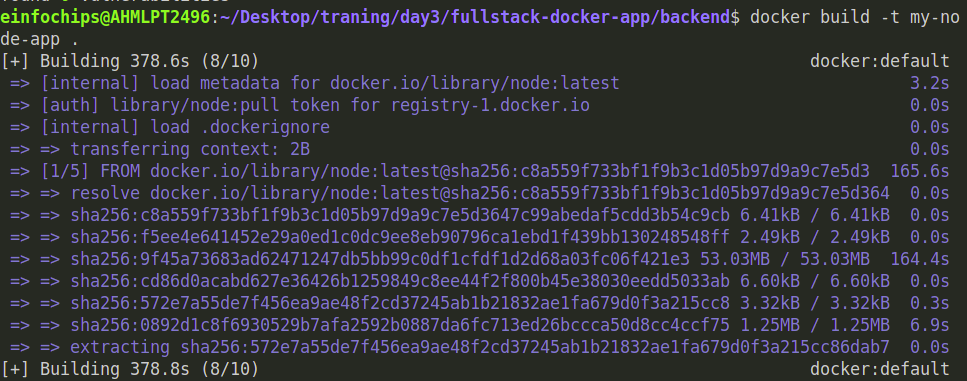
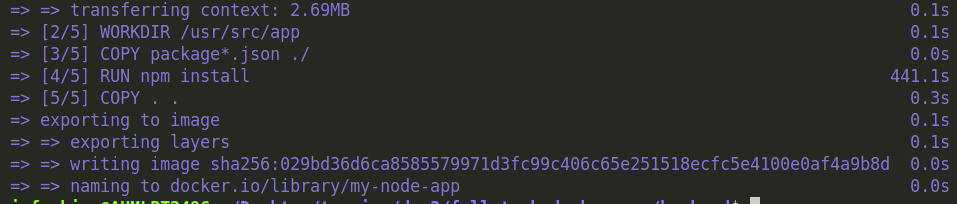


**Run the PostgreSQL Container:**  
docker run --name postgres-container --network fullstack-network -v pgdata:/var/lib/postgresql/data -d my-postgres-db



**Build the Backend Image:**  
docker build -t my-node-app .

cd ..

**Run the Backend Container:**  
docker run --name backend-container --network fullstack-network -d my-node-app  
  


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

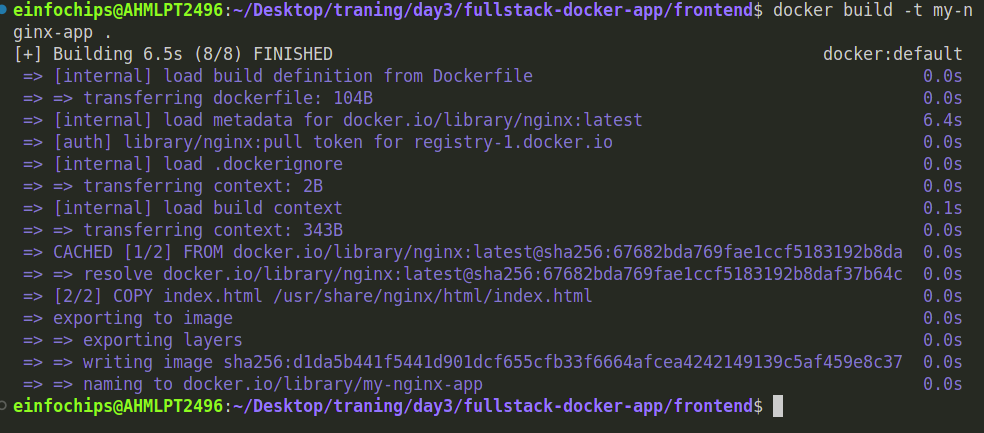
In the frontend directory, create a file named Dockerfile with the following content:  
  
FROM nginx:latest

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

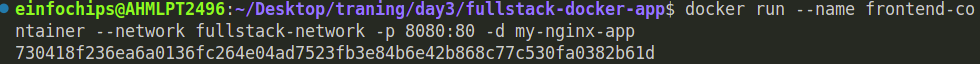
**Build the Frontend Image:**  
cd frontend

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

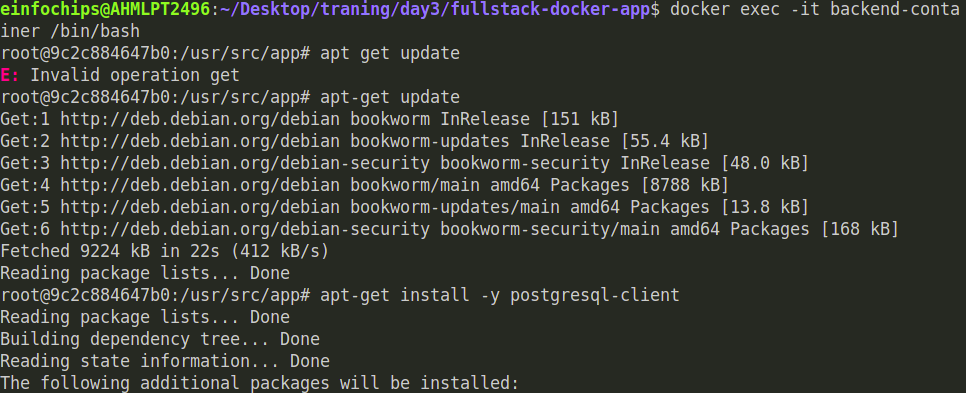
  
  
**Part 5: Connecting the Backend and Database**

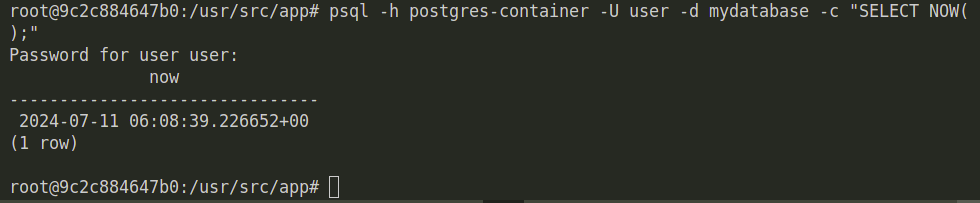
**Objective:** Ensure the backend can communicate with the database and handle data requests.

**Steps:**

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();"  
  
  
 **Final Verification:**

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

