### **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:**



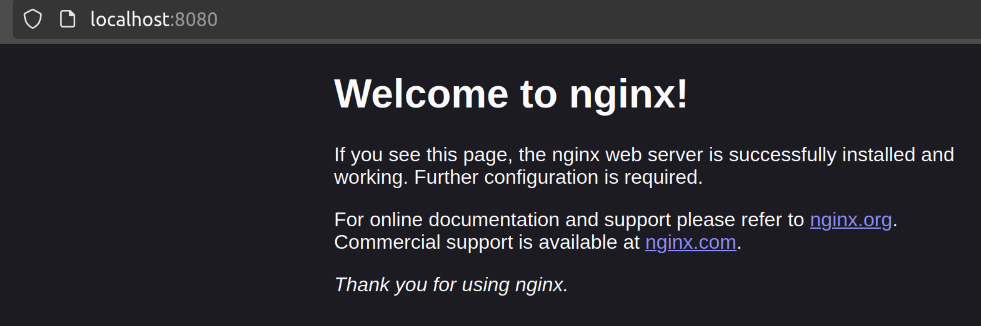
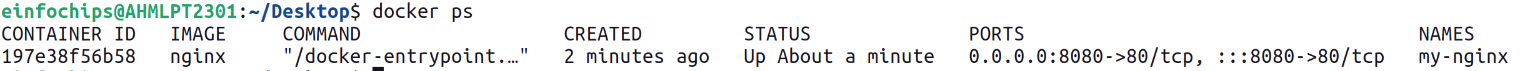
**Run the Nginx Container:**

--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:**



### **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:**



Here -it means interactive terminal mode so that we can give input inside container and /bin/bash is for launching bash shell inside container.

**Create a Custom HTML Page:**  


**Exit the Container:**



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



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

**Verify the New Container:**



### **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.

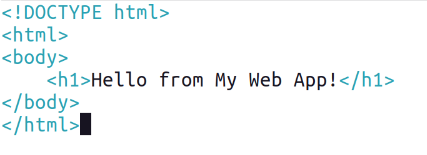
**Steps:**

**Create a Project Directory:**

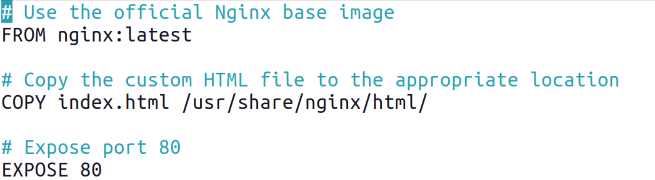


**Create a Simple Web Application:**

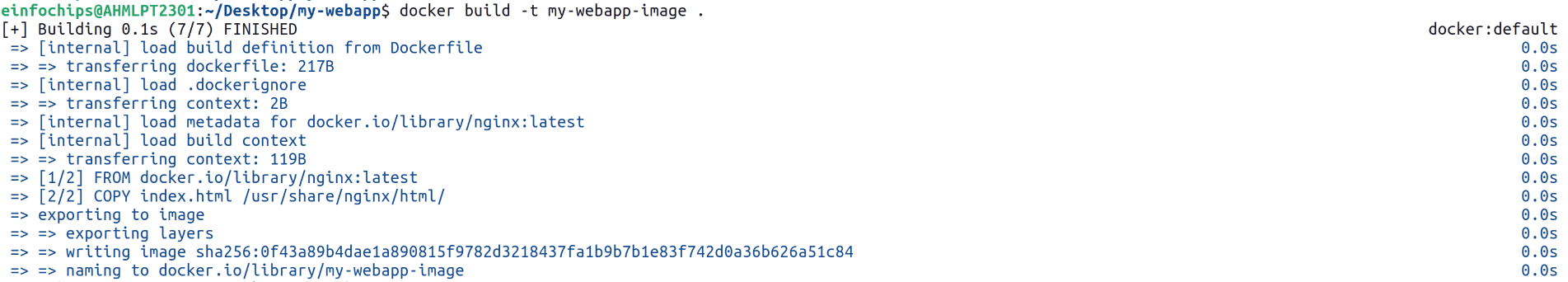
Create an index.html file:



**Write the Dockerfile:**



**Build the Docker Image:**



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

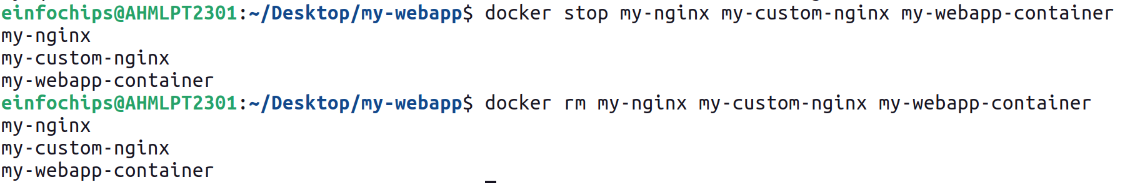
**Verify the Web Application:**



### **Part 4: Cleaning Up**

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

**Steps:**

**Stop and Remove the Containers:**

**Remove the Images:**

**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:**  


**Create Subdirectories for Each Service:**

**Create Shared Network and Volume:**

Docker allows communication between containers through a shared network.



**Create a volume for the PostgreSQL database:**



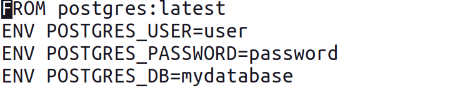
### **Part 2: Setting Up the Database**

**Objective:** Set up a PostgreSQL database with Docker.

**Steps:**

**Create a Dockerfile for PostgreSQL:**

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



**Build the PostgreSQL Image:**



**Run the PostgreSQL Container:**

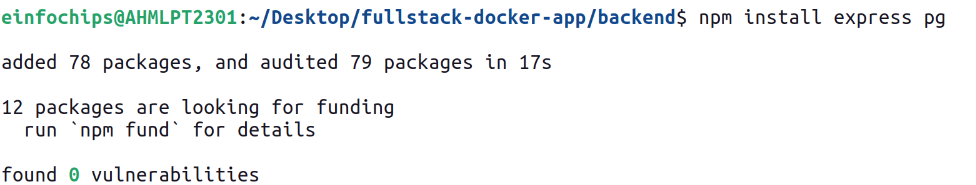
### **Part 3: Setting Up the Backend (Node.js with Express)**

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

**Steps:**

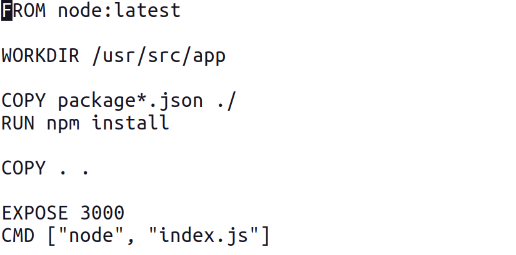
**Initialize the Node.js Application:**

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

  
**Create the Application Code:**

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

**Create a Dockerfile for the Backend:**

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

**Build the Backend Image:**



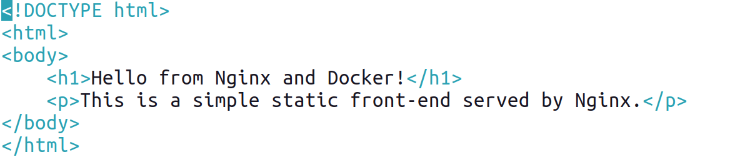
**Run the Backend Container:**

### **Part 4: Setting Up the Frontend (Nginx)**

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

**Steps:**

**Create a Simple HTML Page:**

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

**Create a Dockerfile for the Frontend:**

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



**Build the Frontend Image:**

**Run the Frontend Container:**

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

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

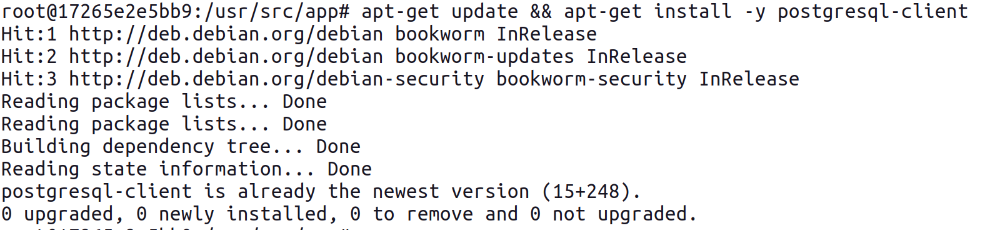
**Steps:**

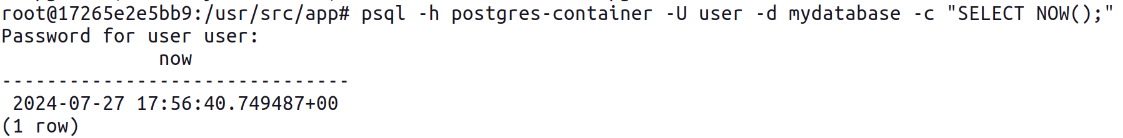
**Update Backend Code to Fetch Data from PostgreSQL:**

Ensure that the index.js code in the backend handles /data endpoint correctly as written above.

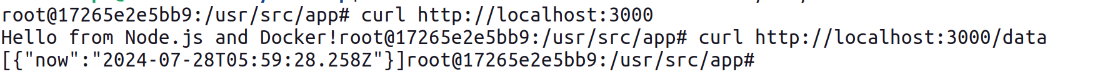
**Verify Backend Communication:**

Access the backend container:  


Test the connection to the database using psql:



**Test the Backend API:**



### **Part 6: Final Integration and Testing**

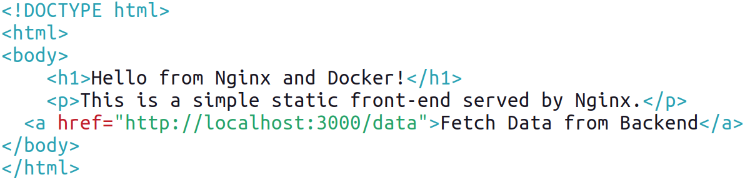
**Objective:** Ensure all components are working together and verify the full-stack application.

**Steps:**

**Access the Frontend:**

****

**Verify Full Integration:**

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

**Rebuild and Run the Updated Frontend Container:**

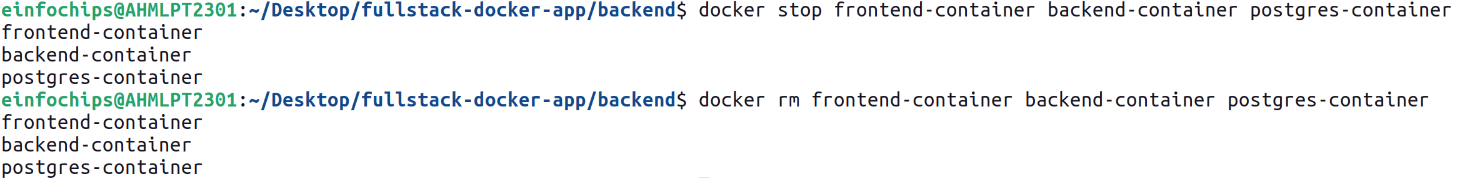
**Final Verification:**

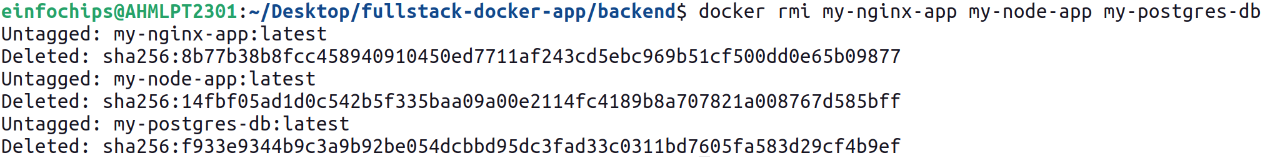
We cannot access the backend from frontend because we haven’t exposed it on our localhost.

**Part 7: Cleaning Up**

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

**Steps:**

**Stop and Remove the Containers:**

**Remove the Images:**  
**Remove the Network and Volume:**

