

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:

1. Pull the Nginx Image:

```
docker pull nginx
```

- pull the latest Nginx image from Docker Hub.
- This command will download the latest Nginx image from the Docker Hub repository to your local machine.

```
einfochips@AHMLPT2509:~$ 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@AHMLPT2509:~$
```

- After the image is downloaded, you can proceed to run the Nginx container.

2. Run the Nginx Container:

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

```
einfochips@AHMLPT2509:~$ docker run --name my-nginx -d -p 8080:80 nginx
edf363747ae63801cf79344aac5470d6b82f09d9d103a059631632b0f7e74fb3
einfochips@AHMLPT2509:~$
```

- `--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.
- Nginx : `<image-name>`

- Start the container in detached mode, running Nginx in the background.

3. Verify the Container is Running:

`docker ps`

- list all running containers:

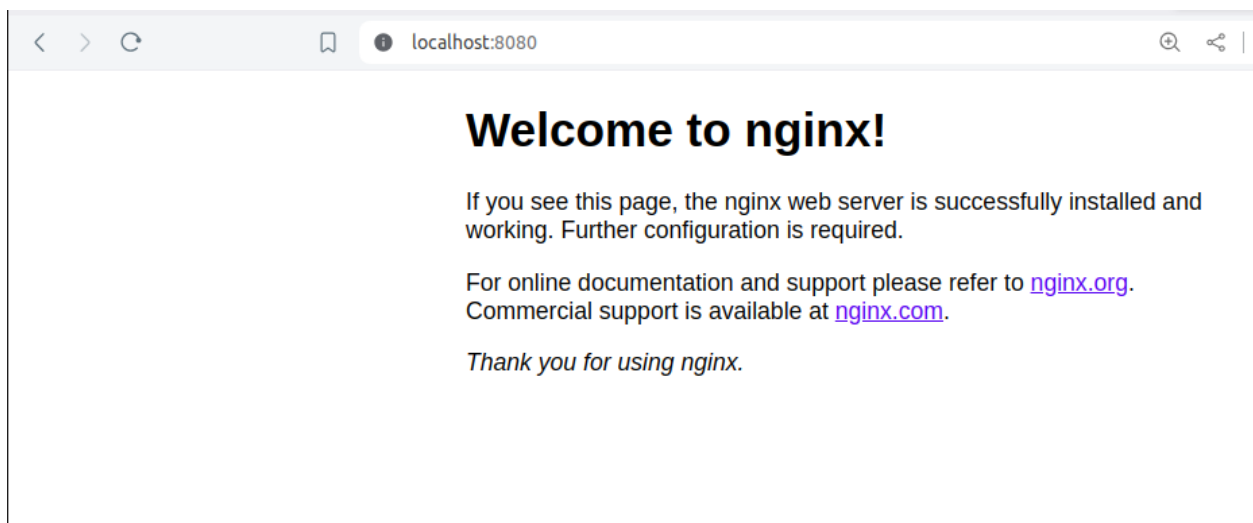
```
einfuchs@AHMLPT2509:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
edf363747ae6	nginx	"/docker-entrypoint..."	About a minute ago	Up About a minute	0.0.0.0:8080->80/tcp, :::8080->80/tcp
8029d6cf0045	kindest/node:v1.25.3	my-nginx "/usr/local/bin/entr..."	8 days ago	Up 49 minutes	127.0.0.1:42813->6443/tcp
a33419b6418a	kindest/node:v1.25.3	kind-control-plane "/usr/local/bin/entr..."	8 days ago	Up 49 minutes	
dca4e7aada6b	kindest/node:v1.25.3	kind-worker2 "/usr/local/bin/entr..."	8 days ago	Up 49 minutes	
06bf988d681f	kindest/node:v1.30.0	kind-worker "/usr/local/bin/entr..."	2 weeks ago	Up 49 minutes	
f527cbd8c03	kindest/node:v1.30.0	einfo-worker "/usr/local/bin/entr..."	2 weeks ago	Up 49 minutes	
3a8ff97b6716	kindest/node:v1.30.0	einfo-worker2 "/usr/local/bin/entr..."	2 weeks ago	Up 49 minutes	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/t
cp, 127.0.0.1:40575->6443/tcp		einfo-control-plane			
34ea2ca47fbf	mysql:8.0	"docker-entrypoint.s..."	2 months ago	Up 49 minutes	3306/tcp, 33060/tcp

4. Visit the Nginx Welcome Page:

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

- indicating that the Nginx container is running successfully.



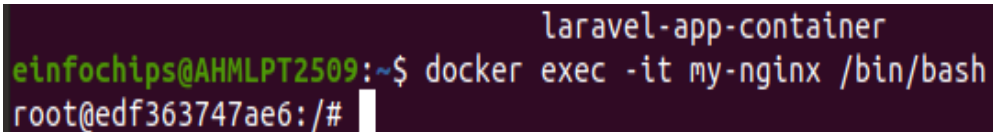
Part 2: Modifying the Container and Creating a New Image:

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

Steps:

1. Access the Running Container:

- `docker exec -it my-nginx /bin/bash`
- access your running container named my-nginx:



```
laravel-app-container
einfochips@AHMLPT2509:~$ docker exec -it my-nginx /bin/bash
root@edf363747ae6:/#
```

- is used to access a running Docker container and start a Bash shell within it.
 - **docker exec:** This is the Docker command used to run a command in a running container.
 - **-it:** This is a combination of two flags:
 - **-i (interactive):** Keeps STDIN open even if not attached.
 - **-t (tty):** Allocates a pseudo-TTY, which allows you to interact with the shell.
 - **my-nginx:** This is the name or ID of the running Docker container you want to access.
 - **/bin/bash:** This is the command that will be executed inside the container. In this case, it starts with a Bash shell, allowing you to interact with the container's filesystem and execute commands.
-
- **docker exec:** Execute a command in a running Docker container.
 - **-it:** Run the command in an interactive terminal session.
 - **my-nginx:** The name of the container where the command should be executed.
 - **/bin/bash:** Start a Bash shell inside the container.
-
- This allows you to interact with the container as if you were logged into a Linux system, making it easier to perform administrative tasks, debugging, and development.

2. Create a Custom HTML Page:

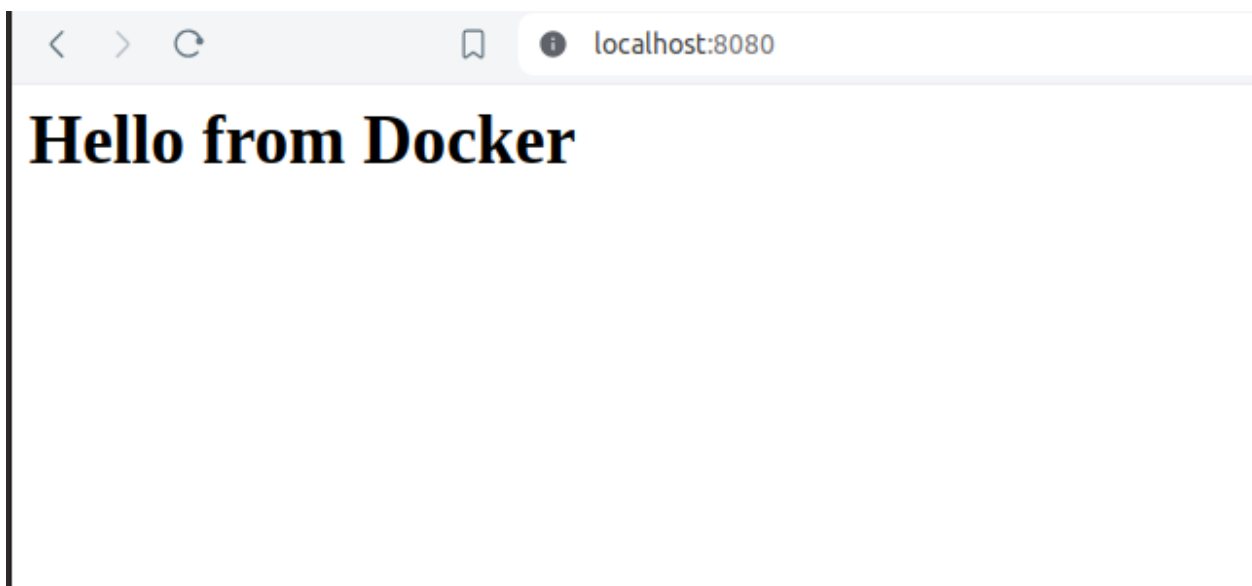
```
echo '<html><body><h1>Hello from Docker! </h1></body></html>' >
/usr/share/nginx/html/index.html
```

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

```
root@edf363747ae6:/# echo '<html><body><h1>Hello from Docker!</h1></body></html>' > /usr/share/nginx/html/index.html
root@edf363747ae6:/#
```

- echo is used to output the string enclosed in quotes.
- >: This is the redirection operator, which directs the output of the echo command into a file.
- /usr/share/nginx/html/index.html: This is the path to the file where the HTML content will be saved. In this case, it's the default location for the main HTML file served by Nginx.
- When executed, this command will create a new file called index.html (or overwrite the existing one) with the specified HTML content.

After making this change, I was able to get this text "Hello from Docker" on my webpage as shown in the below image:



3.Exit the Container:

Exit

- terminates your session within the container, returning you to your host machine's terminal.

```
root@edf363747ae6:/# echo '<html><body><h1>Hello from Docker!</h1></body></html>' > /usr/share/nginx/html/index.html
root@edf363747ae6:/# exit
exit
einfochips@AHMLPT2509:~$
```

4. Commit the Changes to Create a New Image:

`docker commit my-nginx custom-nginx`

- committing the changes to create a new Docker image and running a container from the new image.
- After creating the custom HTML page, commit the changes to create a new image. The docker commit command is used to create a new image from a container's changes.
- my-nginx: The name or ID of the container you want to commit.
- custom-nginx: The name you want to give to the new image.

```
einfochips@AHMLPT2509:~$ docker commit my-nginx custom-nginx
sha256:e0909f205b5f8db14a560cdab93e1f7c49f29d037f464c1c312bcc71bc60f557
einfochips@AHMLPT2509:~$
```

5. Run a Container from the New Image:

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

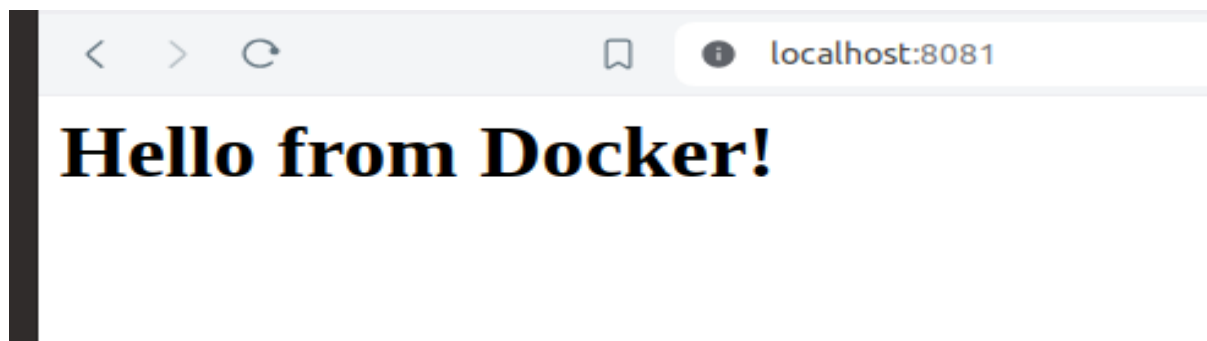
- Once the new image is created, you can run a new container from this image using the docker run command.
- --name my-custom-nginx: Assigns the name my-custom-nginx to the new container.
- -d: Runs the container in detached mode (in the background).

- -p 8081:80: Maps port 8081 on the host to port 80 in the container.
- custom-nginx: The name of the image from which to create the container.

```
einfochips@AHMLPT2509:~$ docker run --name my-custom-nginx -d -p 8081:80 custom-nginx
97a61a3aaa698bb8135db7e6f83a788c8a9c897226adb0f5e64f0dfa99e21bd1
einfochips@AHMLPT2509:~$
```

6. 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:

1. Create a Project Directory:

```
mkdir my-webapp
```

- Use the mkdir command to create a new directory named my-webapp.

2. Navigate into the Project Directory:

`cd my-webapp`

- Use the `cd` command to change your current directory to the new project directory.

```
einfochips@AHMLPT2509:~$ mkdir my-webapp
einfochips@AHMLPT2509:~$ cd my-webapp
einfochips@AHMLPT2509:~/my-webapp$
```

3. Create a Simple Web Application:

- create a simple web application by creating an `index.html` file within the `my-webapp` directory
- Create an `index.html` file:
- `nano index.html`

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

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

```
GNU nano 4.8 index.html
<!DOCTYPE html>
<html>
<body>
  <h1>Hello from My Web App!</h1>
</body>
</html>
```

4. Write the Dockerfile:

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

- nano 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/

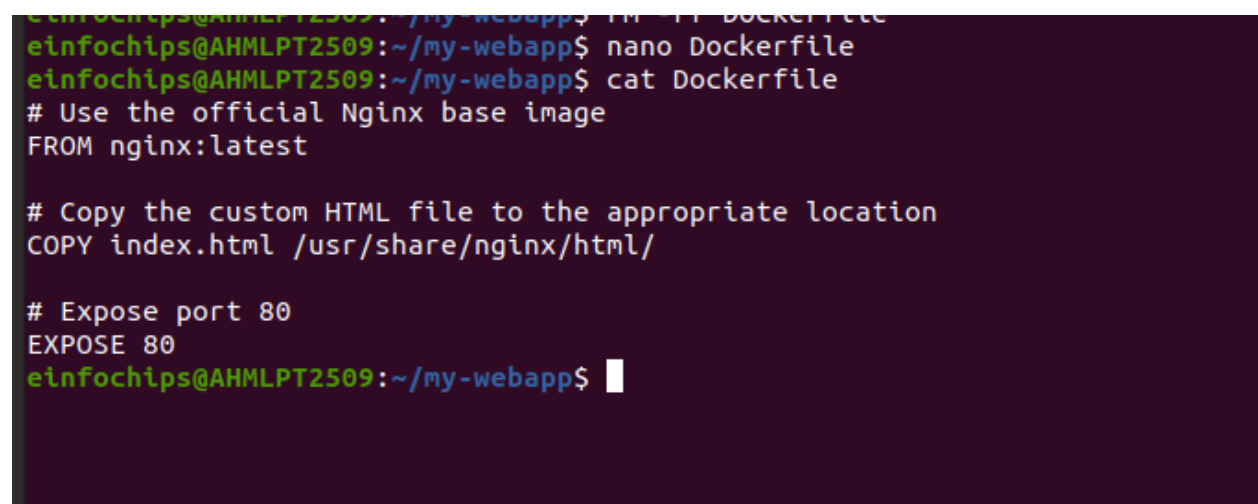
# Expose port 80
EXPOSE 80
```

FROM nginx:

: This line specifies that you want to use the latest version of the official Nginx Docker image as the base image for your custom Docker image.

COPY index.html /usr/share/nginx/html/: This line copies your index.html file from the build context (in this case, your my-webapp directory) into the Nginx server's default HTML directory (/usr/share/nginx/html/).

EXPOSE 80: This line informs Docker that the container will listen on port 80 at runtime. It does not actually publish the port; it serves as documentation.

A terminal window with a dark purple background. The prompt is 'einfochips@AHMLPT2509:~/my-webapp\$'. The user enters 'nano Dockerfile', then 'cat Dockerfile'. The output shows the Dockerfile content: '# Use the official Nginx base image', 'FROM nginx:latest', a blank line, '# Copy the custom HTML file to the appropriate location', 'COPY index.html /usr/share/nginx/html/', a blank line, '# Expose port 80', and 'EXPOSE 80'. The prompt returns to 'einfochips@AHMLPT2509:~/my-webapp\$' with a cursor.

```
einfochips@AHMLPT2509:~/my-webapp$ nano Dockerfile
einfochips@AHMLPT2509:~/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
einfochips@AHMLPT2509:~/my-webapp$
```

5. Build the Docker Image:

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


- `-t my-webapp-image`: Tags the built image with the name `my-webapp-image`.
- located in the current directory (`.`).
- `docker build`: Initiates the process of building a Docker image.
- `-t my-webapp-image`: Tags the built image with the name `my-webapp-image`.
- Specifies the build context, which is the path to the directory containing the Dockerfile. In this case, `.` represents the current directory where the Dockerfile is located.
- the directory containing your Dockerfile when you execute this command, as Docker uses the Dockerfile found in the current directory by default.

```
einfochips@AHMLPT2509:~/my-webapp$ docker build -t my-webapp-image .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
             Install the buildx component to build images with BuildKit:
             https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM nginx:latest
--> fffffc90d343
Step 2/3 : COPY index.html /usr/share/nginx/html/
--> b34a494f9d84
Step 3/3 : EXPOSE 80
--> Running in 841976d0c9de
--> Removed intermediate container 841976d0c9de
--> 5c33ef9c0650
Successfully built 5c33ef9c0650
Successfully tagged my-webapp-image:latest
```

6. Run a Container from the Built Image:

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

- Running the Docker container with the command `docker run --name my-webapp-container -d -p 8082:80 my-webapp-image`
- `--name my-webapp-container`: Assigns the name `my-webapp-container` to the running container.
- `-d`: Detaches the container and runs it in the background (daemon mode).
- `-p 8082:80`: Maps port 8082 on your host machine to port 80 inside the container.

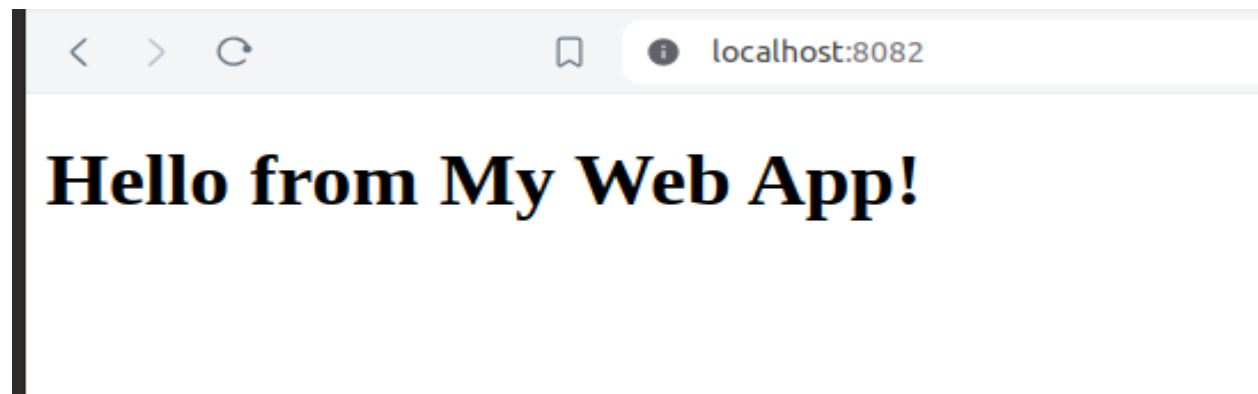
- my-webapp-image: Specifies the Docker image from which to create the container (my-webapp-image in this case).
- Docker will start a container named my-webapp-container based on the my-webapp-image image. The container's port 80 (inside the container) is mapped to port 8082 on your host machine, allowing you to access the web application running inside the container via <http://localhost:8082> or http://<your_host_ip>:8082 depending on your setup.

```
einfochips@AHMLPT2509:~/my-webapp$ docker run --name my-webapp-container -d -p 8082:80 my-webapp-image  
a227a52fe52fac75f47bebb64bcd70248a4f6da474b9ca0e57d40209db7b9765  
einfochips@AHMLPT2509:~/my-webapp$
```

7. Verify the Web Application:

Visit <http://localhost:8082> in your browser. You should see your custom web application.

- <http://>: This indicates that the protocol being used is HTTP.
- localhost: Refers to the local machine where Docker is running.
- 8082: This is the port on your host machine that you mapped to port 80 inside the Docker container.



Part 4: Cleaning Up

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

Steps:

1. Stop and Remove the Containers:

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

- Stop the container named my-webapp-container

```
einfochips@AHMLPT2509:~/my-webapp$ docker stop my-webapp-container  
my-webapp-container  
einfochips@AHMLPT2509:~/my-webapp$
```

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

- Remove the container named my-webapp-container

```
einfochips@AHMLPT2509:~/my-webapp$ docker rm my-webapp-container  
my-webapp-container  
einfochips@AHMLPT2509:~/my-webapp$
```

2. Remove the Images:

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

- nginx: The official Nginx image you used as the base image.
- custom-nginx: Any custom Nginx image you might have built.
- my-webapp-image: The custom image you built for your web application.
- **Stopping Containers:** docker stop stops running containers. You can list multiple container names to stop them all at once.
- **Removing Containers:** docker rm removes stopped containers. Again, you can list multiple container names to remove them all at once.
- **Removing Images:** docker rmi removes Docker images. Make sure no containers are using the images you want to remove; otherwise, Docker will refuse to delete them unless you force it using -f flag.

- Ensure that no containers are using these images. If any containers are running or stopped but still exist, Docker will refuse to remove the images.
- If you encounter any issues where the images are not being removed due to dependencies, you can use the `-f` (force) option with `docker rmi` to forcibly remove the images, although this is not recommended unless you are certain it won't cause issues.

```

einfochips@AHMLPT2509:~/my-webapp$ docker rmi nginx custom-nginx my-webapp-image
Untagged: my-webapp-image:latest
Deleted: sha256:5c33ef9c0650e4b29a4e966f7666727d120d2c61c4895992680b298b0cf1ae39
Deleted: sha256:b34a494f9d848082ae095482ed7cfca32f5933e537734a180826bf33b7533432
Deleted: sha256:870360bc1235f2663d82695460ae0537433be75f15c3d0d0ef64672239244321
Error response from daemon: conflict: unable to remove repository reference "nginx" (must force) - container edf363747ae6 is using its referenced image fffffc90d343
Error response from daemon: conflict: unable to remove repository reference "custom-nginx" (must force) - container 97a61a3aaa69 is using its referenced image e0909f205b5f
einfochips@AHMLPT2509:~/my-webapp$ docker rmi nginx custom-nginx my-webapp-image -f
Untagged: nginx:latest
Untagged: nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Untagged: custom-nginx:latest
Error response from daemon: No such image: my-webapp-image:latest

```

with Force Option (if needed):

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

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

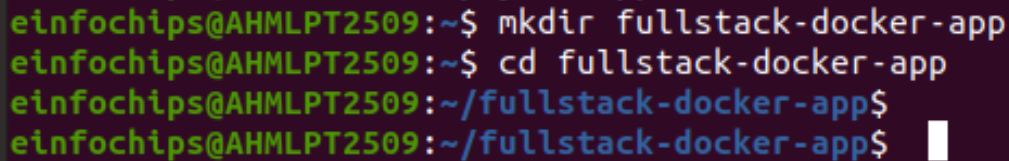
Objective: Create a structured project directory with necessary configuration files.

Steps:

1. Create the Project Directory:

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

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

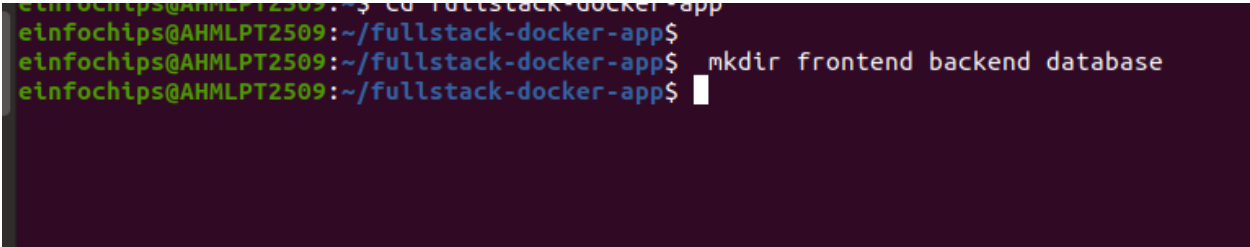
A terminal window with a dark purple background. The prompt is 'einfochips@AHMLPT2509:~\$'. The user enters 'mkdir fullstack-docker-app', then 'cd fullstack-docker-app', and finally the prompt changes to '~/fullstack-docker-app\$'.

```
einfochips@AHMLPT2509:~$ mkdir fullstack-docker-app
einfochips@AHMLPT2509:~$ cd fullstack-docker-app
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

2. Create Subdirectories for Each Service:

- creates three new directories named frontend, backend, and database

```
mkdir frontend backend database
```

A terminal window with a dark purple background. The prompt is 'einfochips@AHMLPT2509:~/fullstack-docker-app\$'. The user enters 'mkdir frontend backend database', and the prompt remains the same.

```
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$ mkdir frontend backend database
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

3. Create Shared Network and Volume:

- Docker allows communication between containers through a shared network.
- Docker allows containers to communicate with each other through shared networks and to persist data using shared volumes. Here, we'll go through the process of creating a shared network and volume step-by-step.

1. Creating a Shared Network:

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

- A Docker network allows multiple containers to communicate with each other.
- docker network create: This command tells Docker to create a new network.
- fullstack-network: This is the name of the network being created. You can choose any name you prefer.

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ mkdir frontend backend database
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker network create fullstack-network
006f5336118303c427ff01954992fe4edbd4707b8796beb4106c9131f64a0fac
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

4. Create a volume for the PostgreSQL database:

`docker volume create pgdata`

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker network create fullstack-network
006f5336118303c427ff01954992fe4edbd4707b8796beb4106c9131f64a0fac
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker volume create pgdata
pgdata
einfochips@AHMLPT2509:~/fullstack-docker-app$ volume ls
volume: command not found
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

- To create a Docker volume specifically for PostgreSQL data storage, you can use the `docker volume create`.
- `docker volume create pgdata`: This command creates a Docker volume named `pgdata`

Purpose:

- *This volume is intended to be used by PostgreSQL containers to persist their data. When you run a PostgreSQL container, you can mount this volume into the container to ensure that the database data is stored persistently even if the container is stopped or removed.*
- Creating a volume for a PostgreSQL database in Docker involves setting up a persistent storage solution that allows the database to store its data outside the container. This ensures that the data persists even if the container is stopped or removed. Here's a detailed explanation of each step:
- `docker volume create pgdata`: This command instructs Docker to create a new volume with the name `pgdata`.

Purpose of the Volume:

- The pgdata volume is intended to be used specifically for storing data from PostgreSQL. This approach has several advantages.

- **Persistence:** Data stored in a volume persists beyond the lifecycle of any container that uses the volume. If a container is removed or stopped, the data remains intact in the volume.
- **Separation of Concerns:** By storing data in a volume, you separate it from the container itself. This makes it easier to manage backups, upgrades, and migrations of your PostgreSQL database.
- **Scalability:** Volumes can be easily shared among multiple containers, allowing you to scale your application horizontally if needed.

Part 2: Setting Up the Database:

Objective: Set up a PostgreSQL database with Docker.

Steps:

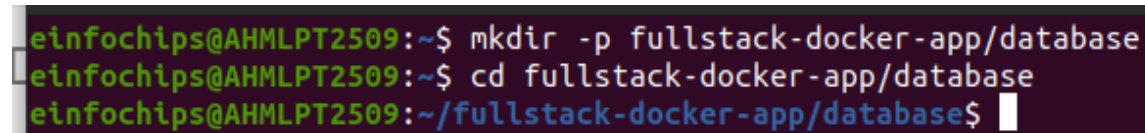
1. Create a Dockerfile for PostgreSQL:

- In the `database` directory, create a file named `Dockerfile` with the following content:
- To create a Dockerfile for PostgreSQL in the database directory.

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

2. Navigate to the database Directory:

- `mkdir -p fullstack-docker-app/database`
- `cd fullstack-docker-app/database`

A terminal window with a dark purple background. The prompt is 'einfochips@AHMLPT2509:~\$'. The first command is 'mkdir -p fullstack-docker-app/database'. The second command is 'cd fullstack-docker-app/database'. The third prompt is 'einfochips@AHMLPT2509:~/fullstack-docker-app/database\$' with a white cursor at the end.

```
einfochips@AHMLPT2509:~$ mkdir -p fullstack-docker-app/database
einfochips@AHMLPT2509:~$ cd fullstack-docker-app/database
einfochips@AHMLPT2509:~/fullstack-docker-app/database$
```

3. Create the Dockerfile:

- Create a new file named Dockerfile in the database directory.

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

- `FROM postgres:latest`: This line specifies the base image for the Docker container, which is the latest version of the official PostgreSQL image from Docker Hub.
- `ENV POSTGRES_USER=user`: This sets the environment variable `POSTGRES_USER` to `user`. This variable specifies the username for the PostgreSQL superuser.
- `ENV POSTGRES_PASSWORD=password`: This sets the environment variable `POSTGRES_PASSWORD` to `password`. This variable specifies the password for the PostgreSQL superuser.
- `ENV POSTGRES_DB=mydatabase`: This sets the environment variable `POSTGRES_DB` to `mydatabase`. This variable specifies the name of the default database to be created.


```
einfochips@AHMLPT2509:~$ cd fullstack-docker-app/database
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ nano Dockerfile
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ cat Dockerfile
FROM postgres:latest

ENV POSTGRES_USER=user

ENV POSTGRES_PASSWORD=password

ENV POSTGRES_DB=mydatabase
einfochips@AHMLPT2509:~/fullstack-docker-app/database$
```

4. Build the PostgreSQL Image:

5. Navigate to the database Directory:

```
cd fullstack-docker-app/database
```

```
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ cd
einfochips@AHMLPT2509:~$ cd fullstack-docker-app/database
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ docker build -t my-postgres-db .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/
```

6. Build the Docker Image:

- Use the docker build command to build the PostgreSQL image and tag it as my-postgres-db:

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

-t my-postgres-db tags the image with the name my-postgres-db, and the . indicates the current directory (which contains the Dockerfile)

- (.) indicates the current directory which contains the Dockerfile

```

einfochips@AHMLPT2509:~/fullstack-docker-app/database$ docker build -t my-postgres-db .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM postgres:latest
latest: Pulling from library/postgres
f11c1adaa26e: Already exists
76ce212b9153: Pull complete
919ca406a058: Pull complete
6b7a1245fe71: Pull complete
8064ffe06c65: Pull complete
4b5c59f2d82c: Pull complete
fe72764b9070: Pull complete
6ef8e2c0f4d9: Pull complete
e71fe9d7ff11: Pull complete
f3225d69190d: Pull complete
2bf90d17afc8: Pull complete
d3aee49eb079: Pull complete
e1e856658919: Pull complete
95c2c2ef9f02: Pull complete
Digest: sha256:0aafd2ae7e6c391f39fb6b7621632d79f54068faebc726caf469e87bd1d301c0
Status: Downloaded newer image for postgres:latest
--> f23dc7cd74bd
Step 2/4 : ENV POSTGRES_USER=user
--> Running in 15e148ad39d3
--> Removed intermediate container 15e148ad39d3
--> 2ac26e1e3cda
Step 3/4 : ENV POSTGRES_PASSWORD=password
--> Running in 6817dd790cc0
--> Removed intermediate container 6817dd790cc0
--> 94362417e5a6
Step 4/4 : ENV POSTGRES_DB=mydatabase
--> Running in 061b3a68bee8
--> Removed intermediate container 061b3a68bee8
--> e855193112ea
Successfully built e855193112ea

```

```

Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM postgres:latest
latest: Pulling from library/postgres
f11c1adaa26e: Already exists
76ce212b9153: Pull complete
919ca406a058: Pull complete
6b7a1245fe71: Pull complete
8064ffe06c65: Pull complete
4b5c59f2d82c: Pull complete
fe72764b9070: Pull complete
6ef8e2c0f4d9: Pull complete
e71fe9d7ff11: Pull complete
f3225d69190d: Pull complete
2bf90d17afc8: Pull complete
d3aee49eb079: Pull complete
e1e856658919: Pull complete
95c2c2ef9f02: Pull complete
Digest: sha256:0aafd2ae7e6c391f39fb6b7621632d79f54068faebc726caf469e87bd1d301c0
Status: Downloaded newer image for postgres:latest
--> f23dc7cd74bd
Step 2/4 : ENV POSTGRES_USER=user
--> Running in 15e148ad39d3
--> Removed intermediate container 15e148ad39d3
--> 2ac26e1e3cda
Step 3/4 : ENV POSTGRES_PASSWORD=password
--> Running in 6817dd790cc0
--> Removed intermediate container 6817dd790cc0
--> 94362417e5a6
Step 4/4 : ENV POSTGRES_DB=mydatabase
--> Running in 061b3a68bee8
--> Removed intermediate container 061b3a68bee8
--> e855193112ea
Successfully built e855193112ea
Successfully tagged my-postgres-db:latest

```

7. Navigate Back to the Root Directory:

```
cd ..
```

- Once the image is built, navigate back to the root directory of your project.

8. Run the PostgreSQL Container:

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

- run the PostgreSQL container using the newly built image.

OR

1. Ensure the Network and Volume Are Created: (optional)

- Make sure the fullstack-network and pgdata volume are created. If not, create them using.

- `docker network create fullstack-network`
- `docker volume create pgdata`

2. Run the PostgreSQL Container:

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

- `docker run` command to start a new container using the `my-postgres-db` image.
- `docker run`: Command to create and start a new container.
- `--name postgres-container`: Assigns the name `postgres-container` to the container.
- `--network fullstack-network`: Connects the container to the `fullstack-network` network.
- `-v pgdata:/var/lib/postgresql/data`: Mounts the `pgdata` volume to the container's `/var/lib/postgresql/data` directory to persist database data.

- `-d`: Runs the container in detached mode (in the background).
- `my-postgres-db`: The name of the image to use for creating the container.

```
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ docker run --name postgres-container --network fullstack-network -v pgdata:/var/lib/postgresql/data -d my-postgres-db
cde900e766c75abf557e70bab032beb19ff89163c2491355868a988f35020491
einfochips@AHMLPT2509:~/fullstack-docker-app/database$
einfochips@AHMLPT2509:~/fullstack-docker-app/database$
```

3. Verification:

- verify that the container is running.

`docker ps`

```
einfochips@AHMLPT2509:~/fullstack-docker-app/database$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
cde900e766c7	my-postgres-db	"docker-entrypoint.s..."	12 minutes ago	Up 12 minutes	5432/tcp
8029d6cf0045	kindest/node:v1.25.3	"/usr/local/bin/entr..."	9 days ago	Up 2 hours	127.0.0.1:42813->6443/tcp
a33419b6418a	kindest/node:v1.25.3	"/usr/local/bin/entr..."	9 days ago	Up 2 hours	
dca4e7aada6b	kindest/node:v1.25.3	"/usr/local/bin/entr..."	9 days ago	Up 2 hours	
06bf988d681f	kindest/node:v1.30.0	"/usr/local/bin/entr..."	3 weeks ago	Up 2 hours	
f527cbdc8c03	kindest/node:v1.30.0	"/usr/local/bin/entr..."	3 weeks ago	Up 2 hours	
3a8ff97b6716	kindest/node:v1.30.0	"/usr/local/bin/entr..."	3 weeks ago	Up 2 hours	0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 127.0.0.1:40575->6443/tcp
34ea2ca47fbf	mysql:8.0	"docker-entrypoint.s..."	2 months ago	Up 2 hours	3306/tcp, 33060/tcp

- This confirms that the PostgreSQL container named `postgres-container` is running using the `my-postgres-db` image, connected to the `fullstack-network` network, and using the `pgdata` volume for persistent storage.

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:

- Node.js and npm are not installed on your system. Here are the steps to install Node.js and npm on Ubuntu:

1. Update Package Index:

```
sudo apt update
```

2. Install Node.js and npm:

- You can install Node.js and npm from the NodeSource repository. First, install the NodeSource PPA (Personal Package Archive):

- For Node.js 14 (LTS version):

```
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
```

- For Node.js 16 (current LTS version as of July 2024):

```
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
```

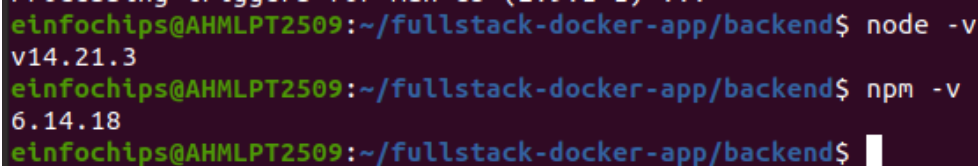
3. Then, install Node.js and npm:

```
sudo apt install -y nodejs
```

- Verify that Node.js and npm were installed correctly:

```
node -v
```

```
npm -v
```

A terminal window with a dark purple background. The prompt is 'einfochips@AHMLPT2509:~/fullstack-docker-app/backend\$'. The first command 'node -v' returns 'v14.21.3'. The second command 'npm -v' returns '6.14.18'. The prompt is shown again at the end of the output.

```
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ node -v
v14.21.3
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ npm -v
6.14.18
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

2 . Initialize the Node.js Application:

1. navigate to the backend directory:

```
cd backend
```

```
cd fullstack-docker-app/backend
```

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ ls
backend database frontend
einfochips@AHMLPT2509:~/fullstack-docker-app$ cd backend/
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ pwd
/home/einfochips/fullstack-docker-app/backend
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ npm init -y

Command 'npm' not found, did you mean:
```

3.initialize a new Node.js application:

`npm init -y`

```
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ npm init -y
Wrote to /home/einfochips/fullstack-docker-app/backend/package.json:

{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

initializes a new Node.js project by creating a package.json file with default values where -y flag automatically answers "yes" to all the prompts.

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

`npm install express pg`

- install both Express and the PostgreSQL client for Node.js.

```
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ npm install express pg
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN backend@1.0.0 No description
npm WARN backend@1.0.0 No repository field.

+ express@4.19.2
+ pg@8.12.0
added 78 packages from 49 contributors and audited 78 packages in 7.204s

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

found 0 vulnerabilities

einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

- After installing the pg package, you can set up a connection to a PostgreSQL database.

5. Create the Application Code:

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

1. Create index.js:

```
cd fullstack-docker-app/backend
```

```
nano index.js
```

```
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 
!\[\]\(082f818d99f166a3ba574d9284d73064\_img.jpg\)

```

einfochips@AHMLPT2509:~/fullstack-docker-app/backend\$ cat index.js
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\(`App running on http://localhost:\${port}`\);
}\);
einfochips@AHMLPT2509:~/fullstack-docker-app/backend\$

```


```


6. Create a Dockerfile for the Backend:

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

```
cd fullstack-docker-app/backend
```

```
nano Dockerfile
```

```
# Use the official Node.js image
```

```
FROM node:latest
```

```
# Create and set the application directory
```

```
WORKDIR /usr/src/app
```

```
# Copy package.json and package-lock.json
```

```
COPY package*.json ./
```

```
# Install dependencies
```

```
RUN npm install
```

```
# Copy the rest of the application code
```

```
COPY . .
```

```
# Expose the port the app runs on
```

```
EXPOSE 3000
```

```
# Command to run the application
```

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ ls
index.js  node_modules  package.json  package-lock.json
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ nano Dockerfile
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ cat Dockerfile
FROM node:latest

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000
CMD ["node", "index.js"]
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

- FROM node:latest: This specifies the base image to use, which is the latest version of Node.js.
- WORKDIR /usr/src/app: This sets the working directory inside the Docker container to /usr/src/app.
- COPY package*.json ./: This copies package.json and package-lock.json files to the working directory.
- RUN npm install: This runs npm install to install dependencies defined in the package.json.
- COPY . .: This copies all files from the current directory to the working directory in the container.
- EXPOSE 3000: This exposes port 3000 on the container, which is the port our Node.js app listens on.
- CMD ["node", "index.js"]: This specifies the command to run the application, which is node index.js.

7. Build the Backend Image:

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

- build the Docker image with the tag my-node-app based on the Dockerfile in the backend directory.
- build the Docker image with a custom tag (my-node-app).
- used to build a Docker image from a Dockerfile.
- -t my-node-app: This tags the image with the name my-node-app. You can use this tag to reference the image later.
- The dot signifies the current directory, which is where the Dockerfile is located.
- During the build process, Docker will execute the instructions in the Dockerfile step by step, creating an image that contains your Node.js application and all its dependencies.

```
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker build -t my-node-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 3.296MB
Step 1/7 : FROM node:latest
latest: Pulling from library/node
e9aef93137af: Pull complete
58b365fa3e8d: Pull complete
3dbed71fc544: Pull complete
ae70830af8b6: Pull complete
572e7a55de7f: Pull complete
9f45a73683ad: Pull complete
0892d1c8f693: Pull complete
819caf31f4d0: Pull complete
Digest: sha256:c8a559f733bf1f9b3c1d05b97d9a9c7e5d3647c99abedaf5cdd3b54c9cbb8eff
Status: Downloaded newer image for node:latest
--> cd86d0acabd6
Step 2/7 : WORKDIR /usr/src/app
--> Running in 2ea7fa608ede
--> Removed intermediate container 2ea7fa608ede
--> a25b423152f1
Step 3/7 : COPY package*.json ./
--> fec867351321
Step 4/7 : RUN npm install
--> Running in 83f3652c3820
npm warn old lockfile
npm warn old lockfile The package-lock.json file was created with an old 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
added 78 packages, and audited 79 packages in 7s
```

```

---> a25b423152f1
Step 3/7 : COPY package*.json ./
---> fec867351321
Step 4/7 : RUN npm install
---> Running in 83f3652c3820
npm warn old lockfile
npm warn old lockfile The package-lock.json file was created with an old 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

added 78 packages, and audited 79 packages in 7s

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

found 0 vulnerabilities
npm notice
npm notice New patch version of npm available! 10.8.1 -> 10.8.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.2
npm notice To update run: npm install -g npm@10.8.2
npm notice
---> Removed intermediate container 83f3652c3820
---> 5567335a138c
Step 5/7 : COPY . .
---> 444eccc88cea
Step 6/7 : EXPOSE 3000
---> Running in e987488f4b9a
---> Removed intermediate container e987488f4b9a
---> 8e04c57843ba
Step 7/7 : CMD ["node", "index.js"]
---> Running in 12e40a1b05a4
---> Removed intermediate container 12e40a1b05a4
---> 8caf8ce6f547
Successfully built 8caf8ce6f547
Successfully tagged my-node-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$

```

8. Navigate Back to the Root Directory:

```
cd ..
```

- taking you back to the root directory of your project.
- moves you up one directory level.

9. Run the Backend Container:

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

```

einfochips@AHMLPT2509:~$
einfochips@AHMLPT2509:~$ docker run --name backend-container --network fullstack-network -d my-node-app
aad7d917415e32008eced92df8afe56cef8436f12ab4af8754585c74f64f39b
einfochips@AHMLPT2509:~$

```

docker run:

- This is the command used to create and start a new container from a specified Docker image.

--name backend-container:

- The --name flag assigns a custom name to the container. In this case, the container will be named backend-container. This makes it easier to manage and reference the container later on.

--network fullstack-network:

- The --network flag specifies the network to which the container should be connected. Here, fullstack-network is the name of the Docker network that the container will join. Docker networks allow containers to communicate with each other.

-d:

- The -d flag stands for "detached mode". When used, it runs the container in the background and returns the control to the terminal. This is useful for long-running services and applications.

my-node-app:

- This is the name of the Docker image from which the container will be created. Docker will look for an image named my-node-app in the local Docker repository. If it doesn't find it locally, it will try to pull it from Docker Hub or another configured registry.

- Putting It All Together:

The full command creates and starts a new container named backend-container from the my-node-app image. The container is attached to the fullstack-network Docker network, allowing it to communicate with other containers on the same network. The -d flag ensures that the container runs in the background.

- Imagine you are working on a full-stack application with separate containers for the frontend, backend, and database services. You might have the following setup:
- **Frontend container:** Runs the frontend application.

- **Backend container:** Runs the backend application (Express.js or Node.js app).
- **Database container:** Runs the database service (MySQL or MongoDB).

All these containers are connected to the same fullstack-network, allowing them to interact seamlessly.

To sum up, the provided Docker command is a convenient way to deploy your backend service in a containerized environment, ensuring it can communicate with other parts of your application stack while running independently in the background.

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>
```

1. Navigate to the frontend Directory:

```
cd /fullstack-docker-app/frontend
```

```
einfochips@AHMLPT2509:~$ cd fullstack-docker-app/
einfochips@AHMLPT2509:~/fullstack-docker-app$ ls
backend  database  frontend
einfochips@AHMLPT2509:~/fullstack-docker-app$ cd frontend/
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ ls
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ nano index.html
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cat index.html
<!DOCTYPE html>
<html>
<body>
```

2. Create the index.html File:

```
nano index.html
```

```
<!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>
```

<!DOCTYPE html>

- This declaration defines the document type and version of HTML. It tells the browser that this is an HTML5 document.

<html>

- This is the root element of an HTML document. All other HTML elements are contained within this tag.

<body>

- This element represents the content of an HTML document. All the visible content of the webpage is placed inside this tag.

`<h1>Hello from Nginx and Docker!</h1>`

- The `<h1>` tag defines the largest heading. In this case, it displays the text "Hello from Nginx and Docker!" as the main heading of the page.

`<p>This is a simple static front-end served by Nginx.</p>`

- The `<p>` tag defines a paragraph. Here, it displays the text "This is a simple static front-end served by Nginx."

```
einfochips@AHMLPT2509:~$ cd fullstack-docker-app/
einfochips@AHMLPT2509:~/fullstack-docker-app$ ls
backend  database  frontend
einfochips@AHMLPT2509:~/fullstack-docker-app$ cd frontend/
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ ls
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ nano index.html
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cat index.html
<!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>
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

3. 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
```

1. Navigate to the frontend Directory:

```
cd fullstack-docker-app/frontend
```

2. create docker file under frontend directory:

```
nano Dockerfile
```

```
FROM nginx:latest
```



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

```
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ nano Dockerfile
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cat Dockerfile
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

FROM nginx:latest:

- This line specifies the base image for the Docker image you are building.
- nginx:latest indicates that you want to use the latest version of the official Nginx image from Docker Hub.
- Nginx is a web server that will serve your static HTML file.

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

- This line copies the index.html file from your local directory to the specified path in the Docker container.
- /usr/share/nginx/html/ is the default directory where Nginx serves its content from.
- By copying index.html to this location, Nginx will serve your HTML file when you access the server.

4. Build the Frontend Image:

1. Navigate to the frontend Directory:

- frontend is the target directory where the Dockerfile and index.html are located.
-
- This command navigates you into the frontend directory so you can build the Docker image from the correct location.

```
cd frontend
```

- cd stands for "change directory."

2. Build the Docker Image:

- Run the docker build command to create a Docker image from the Dockerfile in the current directory.

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker build -t my-nginx-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
             Install the buildx component to build images with BuildKit:
             https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx:latest
latest: Pulling from library/nginx
Digest: sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Status: Downloaded newer image for nginx:latest
--> fffffc90d343
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
--> 685d5cf4b544
Successfully built 685d5cf4b544
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

- docker build is the command to build a Docker image from a Dockerfile.
- -t my-nginx-app tags the image with the name my-nginx-app. Tagging is useful for identifying and managing Docker images.
- . specifies the build context, which is the current directory (where the Dockerfile is located). The build context includes the Dockerfile and all other files and directories required for the build.
- Docker reads the Dockerfile in the current directory.
- It pulls the specified base image (nginx:latest) from Docker Hub if it is not already available locally.
- It copies the index.html file from the current directory to /usr/share/nginx/html/index.html inside the image.
- Docker then creates an image with these instructions.

3. Navigate Back to the Previous Directory:

- `cd ..` command to go back to the parent directory.

```
cd ..
```

5.Run the Frontend Container:

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

```
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
--> 685d5cf4b544
Successfully built 685d5cf4b544
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cd ..
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker run --name frontend-container --network fullstack-network -p 8080:80 -d my-nginx-app
4750d0793f5cb6aac45026dd554fec67add3967b0744a701630614f8329c46c0
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

docker run:

- This is the command to run a new container from a Docker image.

--name frontend-container:

- This option assigns a name to the container. In this case, the container will be named frontend-container.
- Naming a container makes it easier to manage, reference, and identify.

--name frontend-container:

- This option assigns a name to the container. In this case, the container will be named frontend-container.
- Naming a container makes it easier to manage, reference, and identify.

-p 8080:80:

- This option maps a port on your local machine to a port inside the container.
- 8080:80 means that port 8080 on your local machine will be forwarded to port 80 inside the container.
- Port 80 is the default port where Nginx serves HTTP content.
- This allows you to access the Nginx server running in the container by navigating to <http://localhost:8080> in your web browser.

-d:

- This option runs the container in detached mode, which means it runs in the background.
- Detached mode allows the terminal to be free for other commands and keeps the container running independently.

my-nginx-app:

- This is the name of the Docker image from which the container is created.
- In this case, the image my-nginx-app is used, which you built in the previous step.
- Docker creates a new container from the my-nginx-app image.
- The container is named frontend-container.
- The container is connected to the fullstack-network, enabling communication with other containers on the same network.
- Port 8080 on your local machine is mapped to port 80 inside the container, allowing you to access the Nginx server by going to <http://localhost:8080>.
- The container runs in the background (detached mode), so the terminal is free for other tasks.

6. Pre-requisites: (optional)

- Ensure that the fullstack-network Docker network exists before running this command. If it doesn't exist, you can create it with:

docker network create fullstack-network

- By running this command, you'll start a container that serves your static HTML page using Nginx, accessible via <http://localhost:8080>.

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.
- Assuming using Node.js with Express and have already set up your PostgreSQL database, need to make sure your `index.js` file in the backend is correctly configured to handle the `/data` endpoint.

1. Index.js:

```
eInfochips@AHMLPT2509:~/fullstack-docker-app/backend$ cat index.js
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:${port}`);
});
eInfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

2. Verify Backend Communication:

- verify that your backend can communicate with the PostgreSQL database.

Access the running backend container:

- **Access the running backend container** using Docker. Replace backend-container with the actual name or ID of your container:

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

1. if not running backend-container:

```
eInfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker exec -it backend-container /bin/bash
Error response from daemon: container aad7d917415e32008eced92df8afe56cefc8436f12ab4af8754585c74f64f39b is not running
eInfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
4750d0793f5c	my-nginx-app	"/docker-entrypoint..."	30 minutes ago	Up 30 minutes	0.0.0.0:8080->80/tcp
aad7d917415e	my-node-app	"docker-entrypoint.s..."	2 days ago	Exited (137) 2 days ago	
cde900e766c7	my-postgres-db	"docker-entrypoint.s..."	3 days ago	Exited (0) 2 days ago	

1. Check the Status of Docker Containers:

- check the status of your Docker containers to see which ones are running and identify the issue.

```
docker ps -a
```

```
eInfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
4750d0793f5c	my-nginx-app	"/docker-entrypoint..."	30 minutes ago	Up 30 minutes	0.0.0.0:8080->80/tcp
aad7d917415e	my-node-app	"docker-entrypoint.s..."	2 days ago	Exited (137) 2 days ago	
cde900e766c7	my-postgres-db	"docker-entrypoint.s..."	3 days ago	Exited (0) 2 days ago	
97a61a3aaa69	e0909f205b5f	"docker-entrypoint..."	4 days ago	Exited (0) 3 days ago	
edf363747ae6	nginx	"docker-entrypoint..."	4 days ago	Exited (0) 3 days ago	
8029d6cf0045	kindest/node:v1.25.3	"/usr/local/bin/entr..."	12 days ago	Up About an hour	127.0.0.1:42813->64

- This command lists all containers, including those that are not running. Look for the container ID or name of your backend container and note its status.

2. Start the Container:

- If the container is not running, start it using the following command.

```
docker start backend-container
```

- Replace backend-container with the actual name or ID of your container.

```
jenkins
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker start backend-container
backend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker logs backend-container
App running on http://localhost:3000
App running on http://localhost:3000
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

3. Check Container Logs:

- If the container fails to start, check the logs to understand why it is not starting.

`docker logs backend-container`

```
2914aeeb4134 c613f16b6642 "/docker-entrypoint..." 2 months ago Exited (0) 2 months ago
friendly_varahamihira
5c663745a5ac hello-world "-p 8081:80 hello-wo..." 2 months ago Created
deattached
bb8a93b21495 hello-world "-d -p 8080:80" 2 months ago Created
practical_cannon
7e8a1f982424 hello-world "/hello" 2 months ago Exited (0) 2 months ago
blissful_swartz
7833028012b7 c613f16b6642 "/docker-entrypoint..." 2 months ago Exited (0) 2 months ago
attached
68531dc8ed55 c613f16b6642 "/docker-entrypoint..." 3 months ago Exited (0) 2 months ago
awesome_villani
2bc25bc03baa jenkins/jenkins:lts "/usr/bin/tini -- /u..." 3 months ago Exited (143) 2 months ago
jenkins
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker start backend-container
backend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker logs backend-container
App running on http://localhost:3000
App running on http://localhost:3000
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$
```

- This will provide you with information about why the container is failing to start.

4. Troubleshoot and Fix Issues:

- Based on the logs, you can troubleshoot and fix the issues. Common issues include:
 - **Database connection issues:** Ensure that your PostgreSQL database is running and accessible from the backend container.
 - **Port conflicts:** Make sure the ports required by your backend container are not being used by other services.
 - **Configuration errors:** Check your environment variables and configuration files to ensure they are correctly set up.

5. Verify Backend Communication Again:

- Once the container is running, you can access it and verify the backend communication.

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

```
jenkins
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker start backend-container
backend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker logs backend-container
App running on http://localhost:3000
App running on http://localhost:3000
einfochips@AHMLPT2509:~/fullstack-docker-app/backend$ docker exec -it backend-container /bin/bash
root@aad7d917415e:/usr/src/app# ls
Dockerfile index.js node_modules package-lock.json package.json
root@aad7d917415e:/usr/src/app#
```

- Inside the container, you can use psql or other tools to check the PostgreSQL connection as described in the previous response.
- **Check the status of Docker containers** using `docker ps -a`.
- **Start the container** using `docker start backend-container`.
- **Check the container logs** using `docker logs backend-container` if it fails to start.
- **Troubleshoot and fix any issues** based on the logs.
- **Verify backend communication** by accessing the container once it is running.

3. 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();"

```

1. Update and Install PostgreSQL Client:

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

- `apt-get update`: This command updates the package lists for the package manager, ensuring it has the most recent information about available packages and their versions.
- `apt-get install -y postgresql-client`: This installs the PostgreSQL client, a utility that allows you to connect to and interact with a PostgreSQL database from the command line. The `-y` flag automatically confirms the installation without prompting.

2. Connect to the Database Using psql:


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

- psql: This is the PostgreSQL interactive terminal, which you can use to interact with the PostgreSQL database.
- -h postgres-container: Specifies the host where the PostgreSQL server is running. In this case, postgres-container is the hostname or IP address of the server.
- -U user: Specifies the username to connect to the database. Replace user with the actual username.
- -d mydatabase: Specifies the name of the database to connect to. Replace mydatabase with the actual database name.
- -c "SELECT NOW();": This runs a SQL command. The command SELECT NOW();` returns the current date and time from the database server.

3. Exit the container:

```
exit
```

- exit the current shell session.
- running these commands inside a container.
- exit the container and return you to the host system.

1. Update Package Lists and Install PostgreSQL Client:

- Update the package lists to ensure you have the latest information.
- Install the PostgreSQL client so you can use the psql command.

2. Update Package Lists and Install PostgreSQL Client:

- Update the package lists to ensure you have the latest information.
- Install the PostgreSQL client so you can use the psql command.

2. Connect to the PostgreSQL Database:

- Use psql to connect to the PostgreSQL database.

- Provide the host (-h), username (-U), and database name (-d).
- Run a simple SQL command (SELECT NOW();) to test the connection.

3. Exit the Container:

- Once the test is done, use the exit command to leave the container.

4. Test the Backend API:

- Verify that the backend server is running and serving responses.
- Ensure that the backend can fetch and display data from a PostgreSQL database.
- backend server route defined for the /data endpoint.
- receiving a request to this endpoint, server connect to the PostgreSQL database.
- executes a query like SELECT NOW(); get current date and time .
- Visit <http://localhost:3000> to see the basic message.
- Visit <http://localhost:3000/data> to see the current date and time fetched from PostgreSQL.

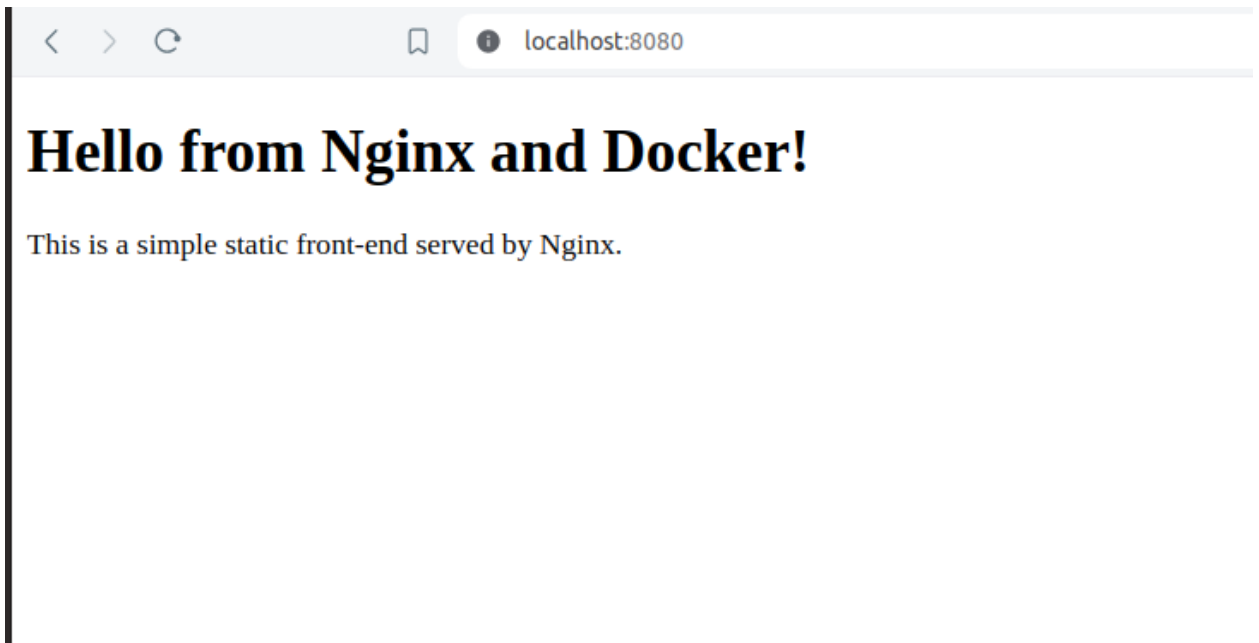
#Part 6: Final Integration and Testing:

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.



2. Verify Full Integration:

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

```
cd /fullstack-docker-app
```

```
cd frontend/
```

```
nano index.html
```

```
<!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>
```

```
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ nano index.html
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cat index.html
<!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>

einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

3. Rebuild and Run the Updated Frontend Container:

```
cd fullstack-docker-ap
```

```
cd frontend
```

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker build -t my-nginx-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
             Install the buildx component to build images with BuildKit:
             https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx:latest
--> fffffc90d343
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
--> 26322b7b755f
Successfully built 26322b7b755f
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

```
docker stop frontend-container
```

```
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker build -t my-nginx-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
             Install the buildx component to build images with BuildKit:
             https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx:latest
--> fffffc90d343
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
--> 26322b7b755f
Successfully built 26322b7b755f
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker stop frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

```
docker rm frontend-container
```

```
Sending build context to Docker daemon 3.072kB
Step 1/2 : FROM nginx:latest
--> fffffc90d343
Step 2/2 : COPY index.html /usr/share/nginx/html/index.html
--> 26322b7b755f
Successfully built 26322b7b755f
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker stop frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker rm frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

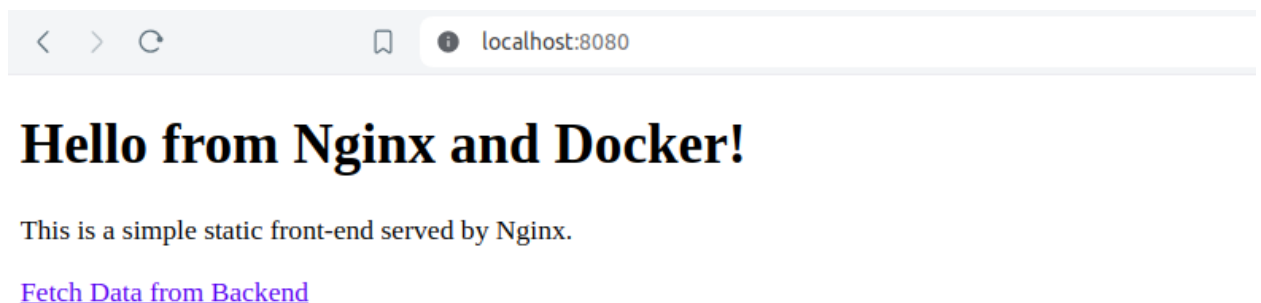
```
Successfully tagged my-nginx-app:latest
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker stop frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker rm frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker run --name frontend-container --network fullstack-network -p 8080:80 -d my-nginx
-app
46b72fdf96fb6757b3f71268517a21faacaac250836d324a847b0037b210da38
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$
```

```
cd ..
```

```
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker rm frontend-container
frontend-container
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ docker run --name frontend-container --network fullstack-network -p 8080:80 -d my-nginx
-app
46b72fdf96fb6757b3f71268517a21faacaac250836d324a847b0037b210da38
einfochips@AHMLPT2509:~/fullstack-docker-app/frontend$ cd ..
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

4. 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:

1. Stop and Remove the Containers:

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker stop frontend-container backend-container postgres-container
frontend-container
backend-container
postgres-container
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

2. Remove container:

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker stop frontend-container backend-container postgres-container
frontend-container
backend-container
postgres-container
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker rm frontend-container backend-container postgres-container
frontend-container
backend-container
postgres-container
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

3. Remove the Images:

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

```
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker rmi my-nginx-app my-node-app my-postgres-db
Untagged: my-nginx-app:latest
Deleted: sha256:26322b7b755f5bcd9a9a091337ae6835e3eef901fca9e2234b3d3a486c3932
Deleted: sha256:84b272319398a1f4b09c24813445b75c17e928c3837adc0846e6bf7fed8c22cf
Untagged: my-node-app:latest
Deleted: sha256:d177c346c09a6f3411d61f34593cd9358f553caf0822674e77d3089ac09f26cf
Deleted: sha256:0b5afd3f8c2573c70a7902a8305fb20ec6883684c50cbe2885bb11d182610afe
Deleted: sha256:4eb8f235f7a9c722307d1fa5a1680beab90274989c28529ea4770f3a827c14fb
Deleted: sha256:ae703b9512cafb47ba9a29741cbb6874a48726c8c2bbf35eb6ef2417d5c386a7
Deleted: sha256:21c67f8b98d01a97510270e5c36a885441e3e252af52514d30455e9df5526e9a
Deleted: sha256:a7ef43321f7940e545d5814bb8bbe388964a76c35304f7648c96d60d69f8424
Deleted: sha256:f07945b1822dbb3426b7d499e32839b4ab2a4d83f31525bd381f6823582c0c7e
Deleted: sha256:62f19babe38a388b959854fd916bb70e194554a6b6d7cdf692644171d63e5a5c
Deleted: sha256:53110c4a76d5f02515c1e6991bea3be126735c1c42cf53d6a43d68f7b60fe0f6
Deleted: sha256:4596cb87cef4bd3ba63331074c0249833876d21de3f8a078b3ae31c5f1dc8ccc
Untagged: my-postgres-db:latest
Deleted: sha256:ee7ac6f3b5fd5f989f5a0964d5600bae2fa7e968b04b7527f90891a08d23ec80
Deleted: sha256:a29c398e23cdae5c1bf99e63ace5ace383553e9f6ea877267924b5ca2f0410fa
Deleted: sha256:0568f051925e9d9c92b44d6039f4221b594ddcd4164e1714fc1dee6c28c8a897
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

4. Remove the Network and Volume:

`docker network rm fullstack-network`

```
Deleted: sha256:53110c4a76d5f02515c1e6991bea3be126735c1c42cf53d6a43d68f7b60fe0f6
Deleted: sha256:4596cb87cef4bd3ba63331074c0249833876d21de3f8a078b3ae31c5f1dc8ccc
Untagged: my-postgres-db:latest
Deleted: sha256:ee7ac6f3b5fd5f989f5a0964d5600bae2fa7e968b04b7527f90891a08d23ec80
Deleted: sha256:a29c398e23cdae5c1bf99e63ace5ace383553e9f6ea877267924b5ca2f0410fa
Deleted: sha256:0568f051925e9d9c92b44d6039f4221b594ddcd4164e1714fc1dee6c28c8a897
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker network rm fullstack-network
fullstack-network
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker volume rm pgdata
pgdata
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$
```

`docker volume rm pgdata`

```
Deleted: sha256:ee7ac6f3b5fd5f989f5a0964d5600bae2fa7e968b04b7527f90891a08d23ec80
Deleted: sha256:a29c398e23cdae5c1bf99e63ace5ace383553e9f6ea877267924b5ca2f0410fa
Deleted: sha256:0568f051925e9d9c92b44d6039f4221b594ddcd4164e1714fc1dee6c28c8a897
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker network rm fullstack-network
fullstack-network
einfochips@AHMLPT2509:~/fullstack-docker-app$ docker volume rm pgdata
pgdata
einfochips@AHMLPT2509:~/fullstack-docker-app$
einfochips@AHMLPT2509:~/fullstack-docker-app$
```