Getting Started with Docker and Node.js: A Comprehensive Hands-On Project

Author: Oluwaseun Osunsola
LinkedIn: https://www.linkedin.com/in/oluwaseun-osunsola-95539b175/
Environment & Tool: AWS, Docker, Docker Compose, Dockerfile
Project Link: https://github.com/Oluwaseunoa/DevOps-Projects/tree/main/Docker-Projects

Project Overview
This comprehensive guide walks you through installing Docker on an AWS EC2 Ubuntu instance, mastering fundamental Docker commands, building a custom Docker image for a Node.js application, and orchestrating a multi-container setup using Docker Compose. The application is a full-stack user profile editor built with Express.js, connected to MongoDB for persistent storage, and includes Mongo Express as a web-based admin interface.

By the end of this tutorial, you will have:

- A running Dockerized Node.js web application accessible publicly.
- Persistent data storage in MongoDB.
- Hands-on experience with Docker fundamentals, image building, and container orchestration.
- Understanding of networking, security groups, and environment variable management in containerized environments.

Target Audience: Beginners to intermediate learners in DevOps, Docker, Node.js, and cloud deployment (AWS EC2).

Prerequisites

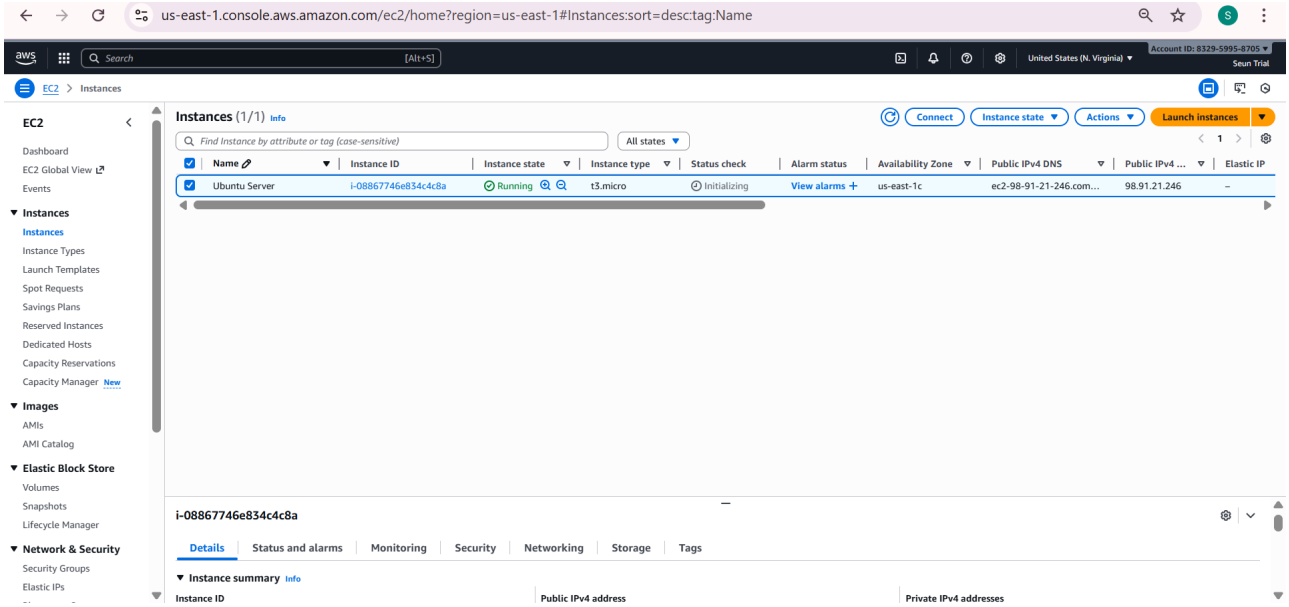
Before starting:

- An active AWS account.
- Basic familiarity with the Linux terminal, SSH, and AWS EC2.
- A local machine with SSH client and SCP support.
- A profile picture (JPG format) to upload for the application demo.

Detailed Step-by-Step Guide

The following steps are executed in sequence on the Ubuntu EC2 instance (except where noted). Each step includes a brief explanation, the commands used (where applicable), and a screenshot reference.

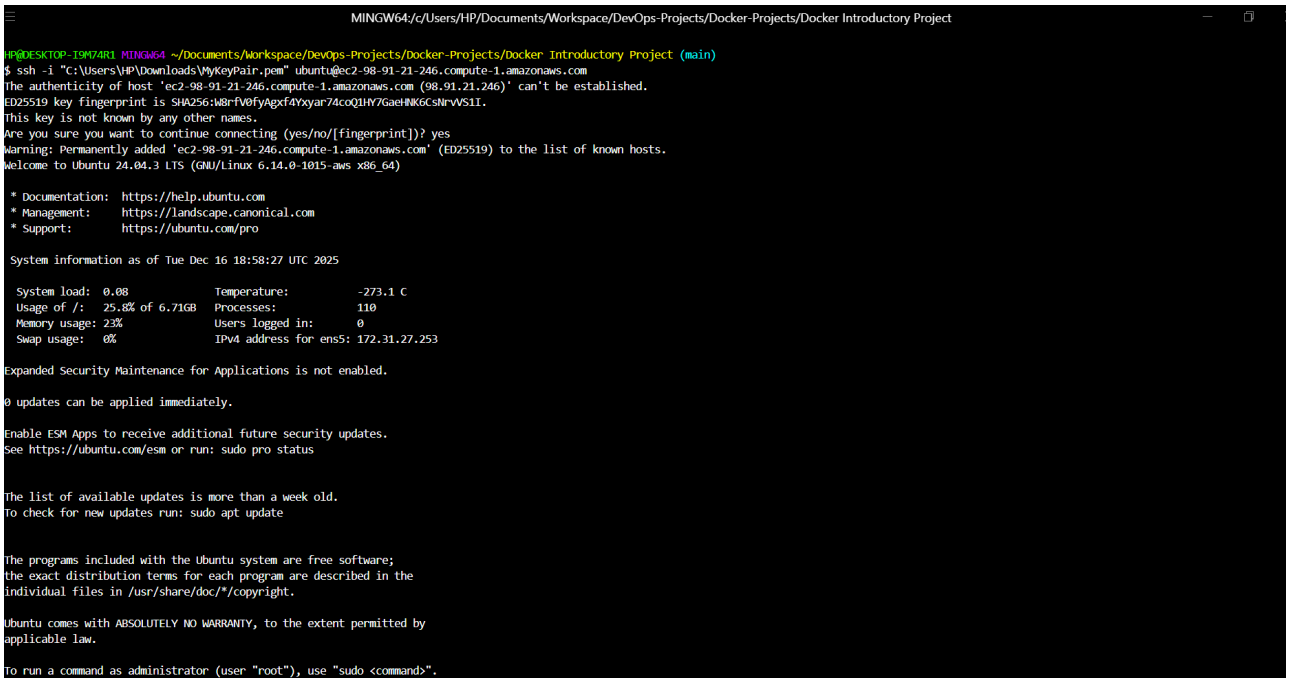
1. **Launch Ubuntu Server**
   Launch a new EC2 instance running Ubuntu Server (e.g., 22.04 LTS). Select an appropriate instance type (t2.micro is sufficient for this demo), configure a key pair for SSH access, and ensure the security group allows inbound SSH (port 22).



2. **Connect to the Ubuntu Server via SSH**

   From your local machine, connect using:

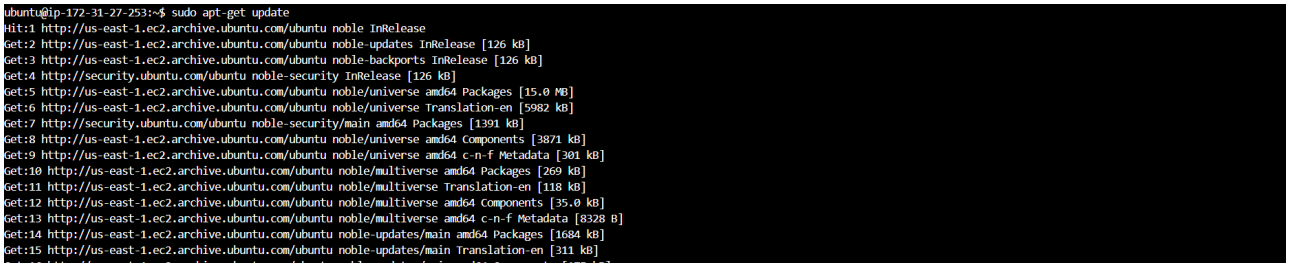
   `ssh -i /path/to/your-key.pem ubuntu@ec2-public-ip`



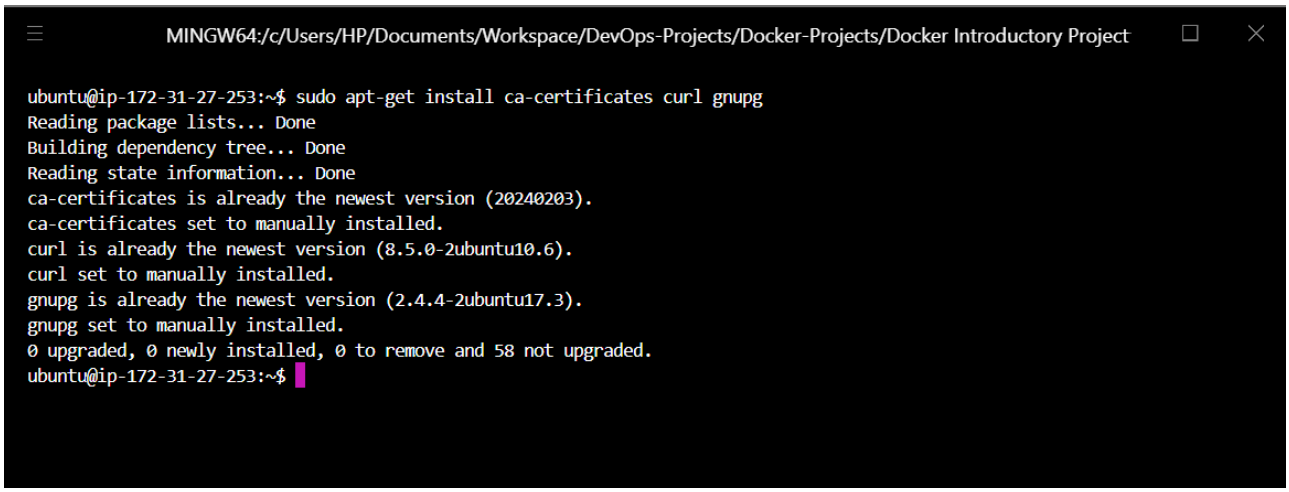
3. **Update the Server**

   Update package lists and upgrade installed packages:

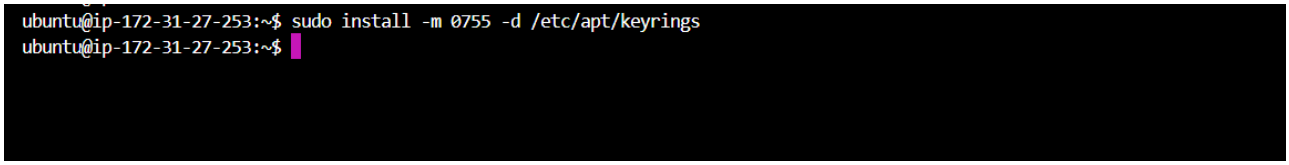
   `sudo apt update && sudo apt upgrade -y`



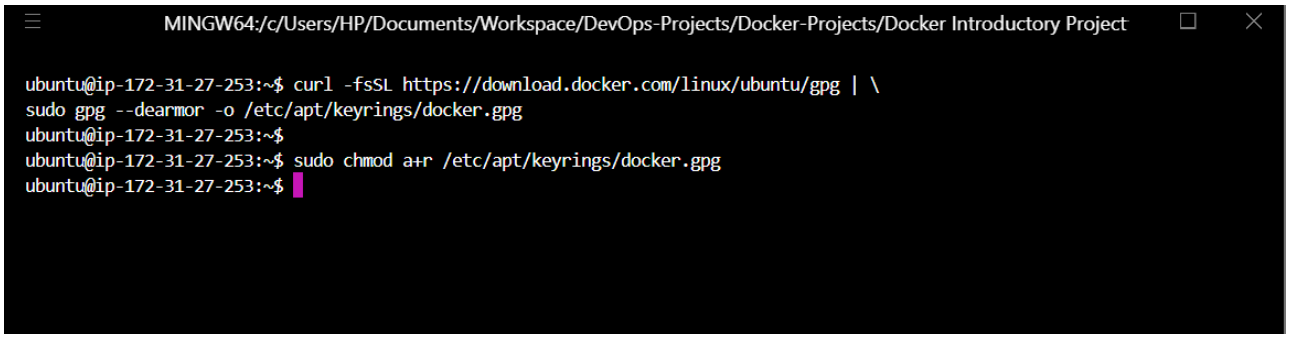
4. **Install Required Packages**
   Install prerequisites for adding the Docker repository:
   `sudo apt install ca-certificates curl gnupg lsb-release -y`



5. **Create Docker Keyring**
   Create directory for Docker's GPG key:
   `sudo mkdir -p /etc/apt/keyrings`



6. **Add Official Docker GPG Key**
   Download and add Docker's official GPG key:
   `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg`



7. **Add Official Docker Repository**
   Add the Docker APT repository:
   `echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`

```
ubuntu@ip-172-31-27-253:~$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
ubuntu@ip-172-31-27-253:~$
```

8. **Install Docker Engine**

   Update packages and install Docker Engine and Compose plugin:

   `sudo apt update && sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y`

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Docker-Projects/Docker Introductory Project

ubuntu@ip-172-31-27-253:~$ sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.5 kB]
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [40.8 kB]
Fetched 89.2 kB in 0s (237 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:
  docker-ce-rootless-extras libslirp0 pigz slirp4netns
Suggested packages:
  cgroupfs-mount | cgroup-lite docker-model-plugin
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libslirp0
  pigz slirp4netns
0 upgraded, 9 newly installed, 0 to remove and 58 not upgraded.
Need to get 91.2 MB of archives.
After this operation, 363 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

9. **Verify Docker Installation**

   Check Docker version:

   `docker --version`

```
ubuntu@ip-172-31-27-253:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-12-16 19:10:49 UTC; 1min 47s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 2264 (dockerd)
      Tasks: 9
     Memory: 93.0M (peak: 105.4M)
        CPU: 425ms
     CGroup: /system.slice/docker.service
             └─2264 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.061935832Z" level=info msg="Restoring containers>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.117976565Z" level=info msg="Deleting nftables IP>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.126051904Z" level=info msg="Deleting nftables IP>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.416408870Z" level=info msg="Loading containers: >
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.434059338Z" level=info msg="Docker daemon" commi>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.434218342Z" level=info msg="Initializing buildki>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.483905560Z" level=info msg="Completed buildkit i>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.494583712Z" level=info msg="Daemon has completed>
Dec 16 19:10:49 ip-172-31-27-253 dockerd[2264]: time="2025-12-16T19:10:49.494783582Z" level=info msg="API listen on /run/d>
Dec 16 19:10:49 ip-172-31-27-253 systemd[1]: Started docker.service - Docker Application Container Engine.
lines 1-22/22 (END)
```

10. **Run Docker Without Sudo**

    Add current user to Docker group:

    `sudo usermod -aG docker $USER`

    Log out and back in for changes to take effect.

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Docker-Projects/Docker Introductory Project

ubuntu@ip-172-31-27-253:~$ sudo usermod -aG docker ubuntu
ubuntu@ip-172-31-27-253:~$
```

11. **Running the First Container (Hello World)**

    Test Docker with a simple container:

    `docker run hello-world`

    ![11.Running_the_First Container_(Hello_World).png](img/11.Running_the_First Container_(Hello_World).png)

12. **Verify Images (docker images)**

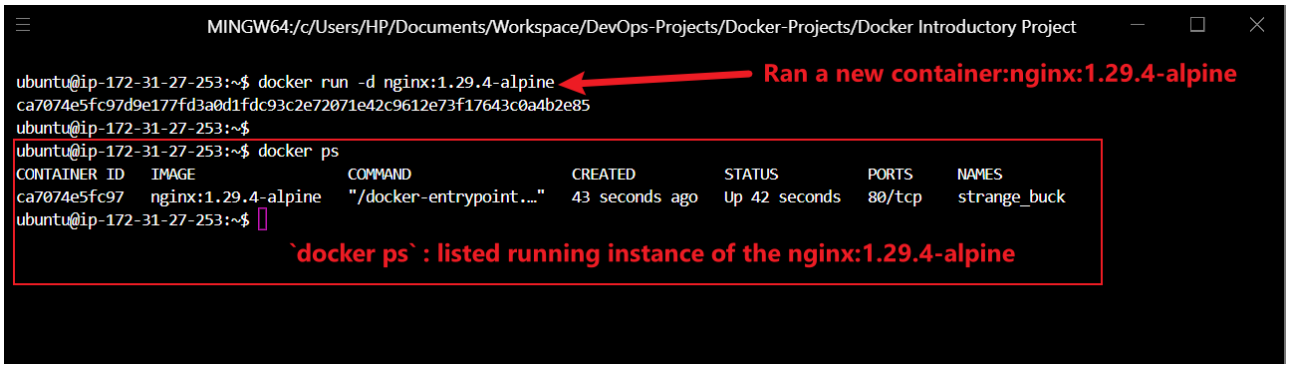
    List locally available images:

    `docker images`

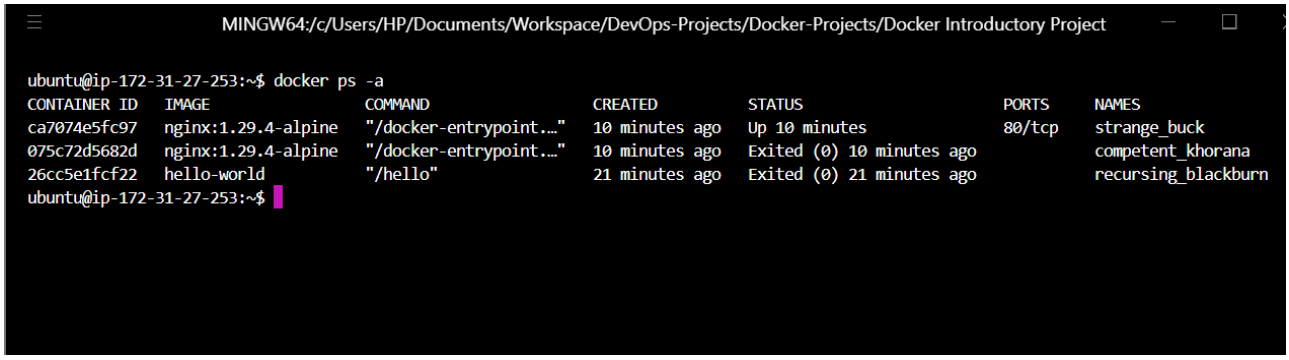
```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Docker-Projects/Docker Introductory Project

ubuntu@ip-172-31-27-253:~$ docker images
                                                                    i Info →  U  In UseIMAGE
ID             DISK USAGE   CONTENT SIZE   EXTRA
hello-world:latest   d4aaab6242e0       25.9kB         9.52kB    U
ubuntu@ip-172-31-27-253:~$
```

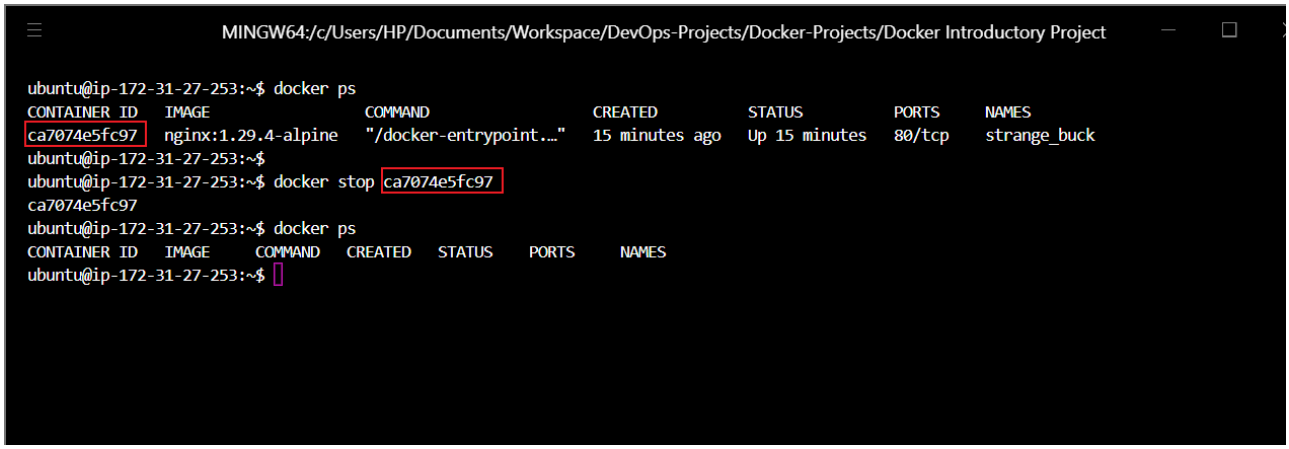
13. **List Running Containers (docker ps)**
    Show currently running containers:
    `docker ps`



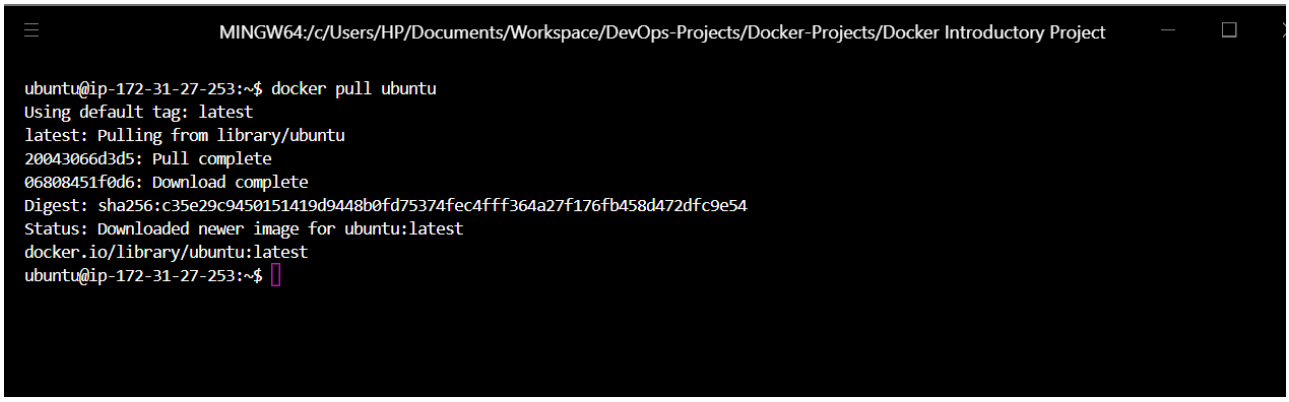
14. **List Both Running and Stopped Containers (docker ps -a)**
    Show all containers (running and stopped):
    `docker ps -a`



15. **Stop a Container Using Its Container ID**
    Stop a running container:
    `docker stop <container_id>`



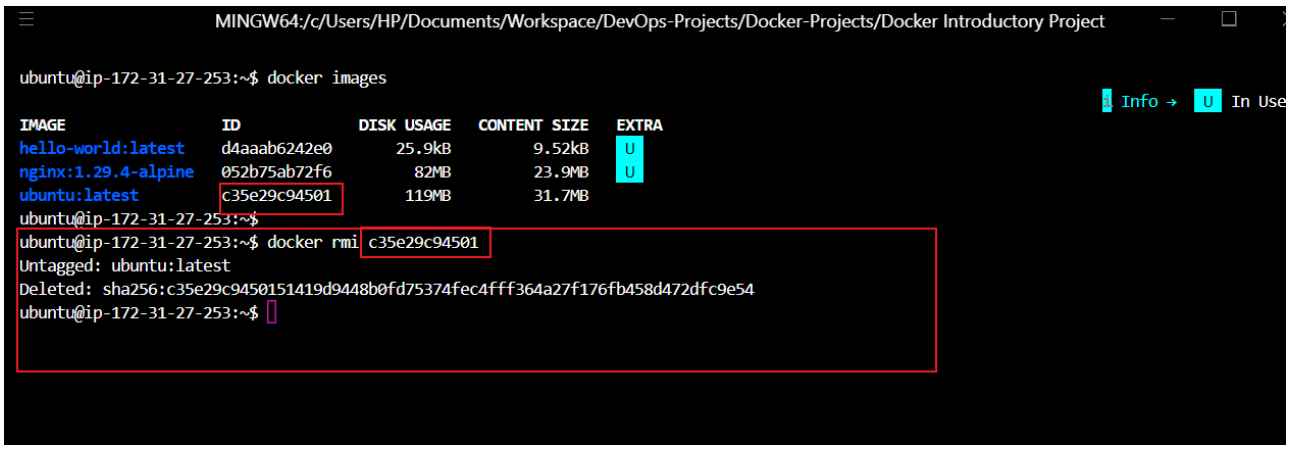
16. **Pull an Ubuntu Image**
    Download the official Ubuntu image:
    `docker pull ubuntu`



17. **Delete Image Using Its Image ID**

    Remove an image:

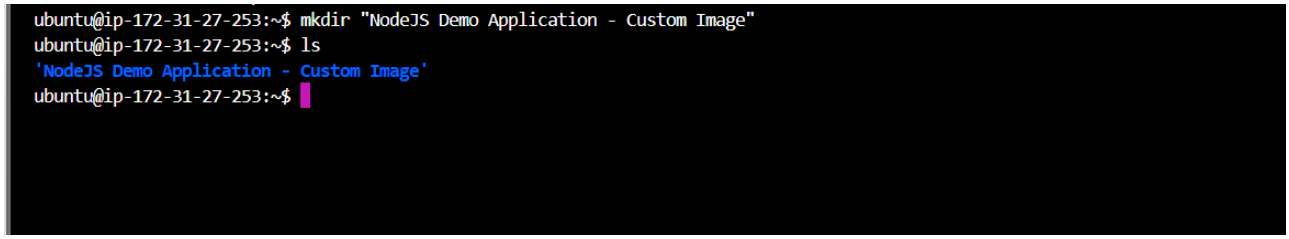
    ```
    docker rmi <image_id>
    ```



18. **Create Project Folder (NodeJS_Demo_Application_-_Custom_Image) and LS It to Confirm**

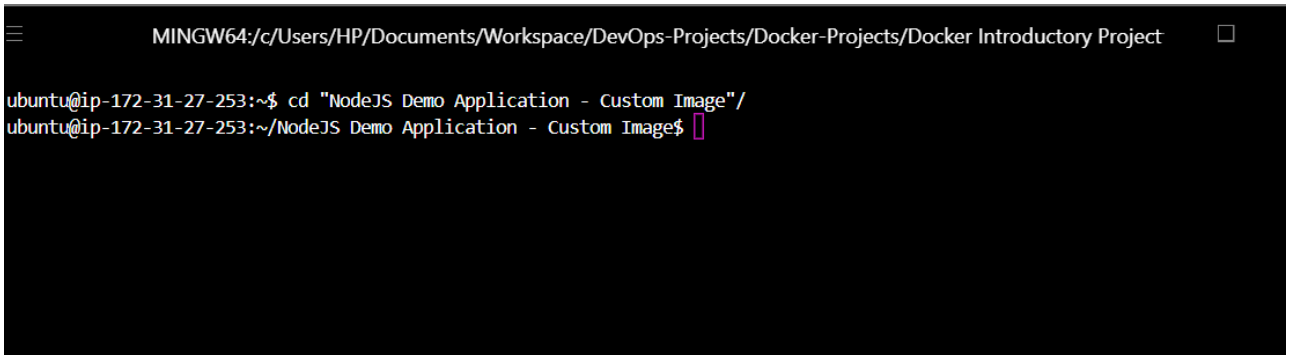
    Create project directory:

    ```
    mkdir "NodeJS Demo Application - Custom Image" && ls
    ```



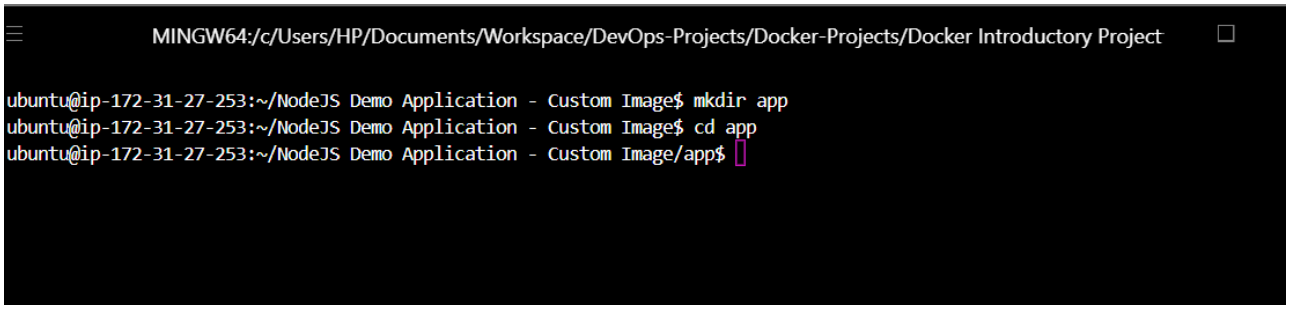
19. **Navigate into the Project Folder (NodeJS_Demo_Application_-_Custom_Image)**

    ```
    cd "NodeJS Demo Application - Custom Image"
    ```



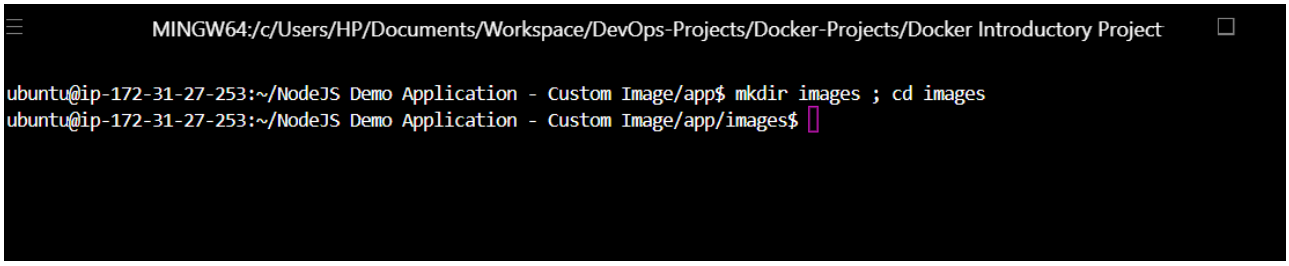
20. **MKDIR App to Create App Folder Then CD into App Folder**

```
mkdir app && cd app
```



21. **In App Create Images Folder and CD into It**

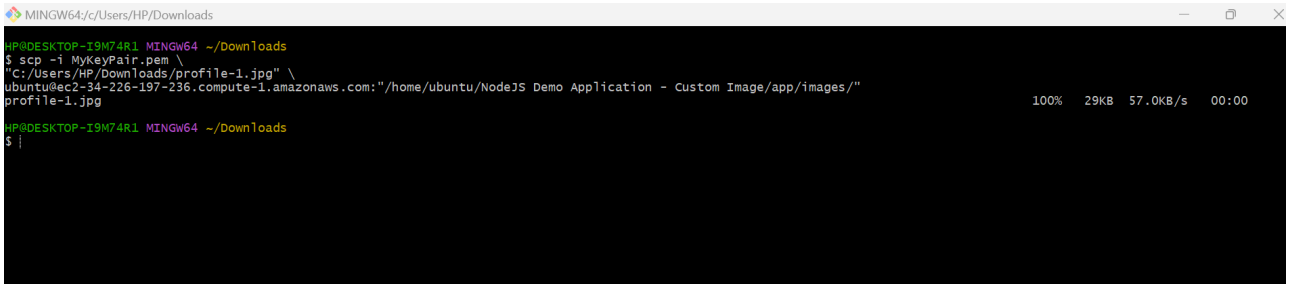
```
mkdir images && cd images
```



22. **On the Host System Upload an Image to Project Images Folder Using SCP**

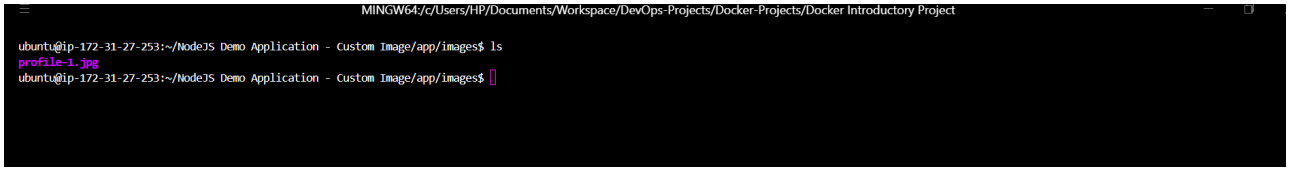
From local machine:

```
scp -i your-key.pem profile-1.jpg ubuntu@ec2-ip:~/NodeJS\ Demo\ Application\ -\
Custom\ Image/app/images/
```



23. **In Image Folder LS to Confirm That Image Is Successfully Uploaded**

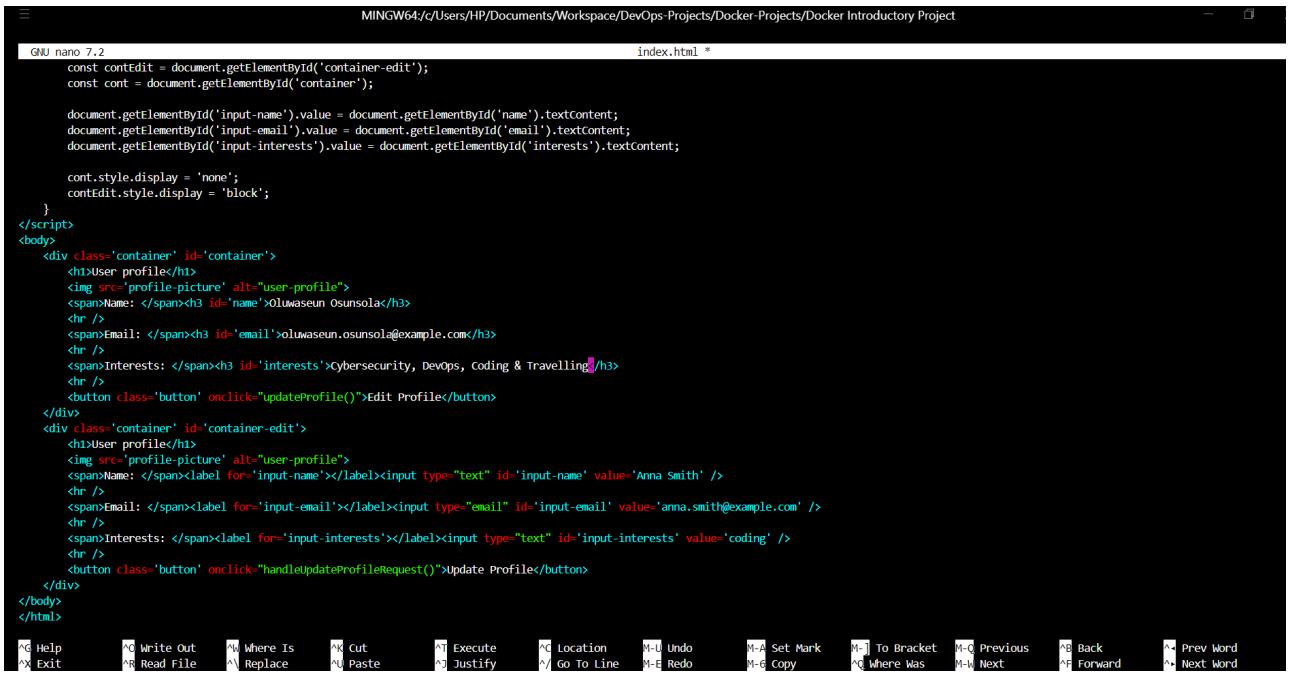
```
ls
```



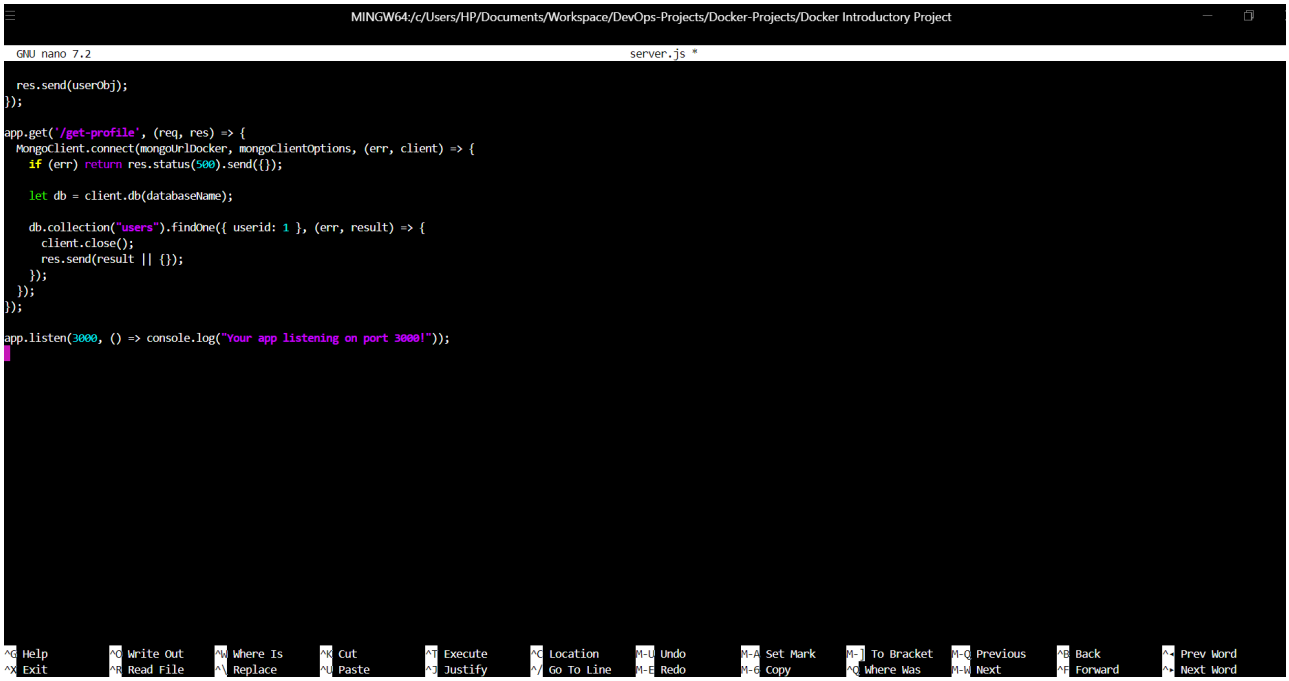
24. **From Image Folder CD Back One Step to App Folder**

```
cd ..
```

25. **Nano index.html Then Add Code Save and Exit**
    Create and populate the frontend HTML file.



26. **Nano server.js Then Add Code Save and Exit**
    Create the Express backend server code.



27. **Initialize the Project Folder (Install Node and NPM If You Don't Have and Then Initialize)**
    Install Node.js if needed, then:
    `npm init -y`

```
ubuntu@ip-172-31-27-253:~/NodeJS Demo Application - Custom Image/app$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (app) nodejs-app
version: (1.0.0)
description: This is a nodejs app runs by docker
entry point: (server.js)
test command:
git repository:
keywords:
author: Oluwaseun Osunsola
license: (ISC)
About to write to /home/ubuntu/NodeJS Demo Application - Custom Image/app/package.json:

{
  "name": "nodejs-app",
  "version": "1.0.0",
  "description": "This is a nodejs app runs by docker",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Oluwaseun Osunsola",
  "license": "ISC"
}


Is this OK? (yes) yes
```

28. **Install the Packages (express, body-parser, mongodb)**

    `npm install express body-parser mongodb`

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Docker-Projects/Docker Introductory Project

ubuntu@ip-172-31-27-253:~/NodeJS Demo Application - Custom Image/app$ npm install express mongodb body-parser --save
npm install express mongodb body-parser --save
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'mongodb@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'bson@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'mongodb-connection-string-url@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }

added 77 packages, and audited 78 packages in 3s

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

found 0 vulnerabilities
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'bson@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'mongodb-connection-string-url@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'mongodb@7.0.0',
npm WARN EBADENGINE   required: { node: '>=20.19.0' },
npm WARN EBADENGINE   current: { node: 'v18.19.1', npm: '9.2.0' }
npm WARN EBADENGINE }
```

29. **LS App Folder Cat package.json to Confirm Packages**

    Verify installed dependencies:

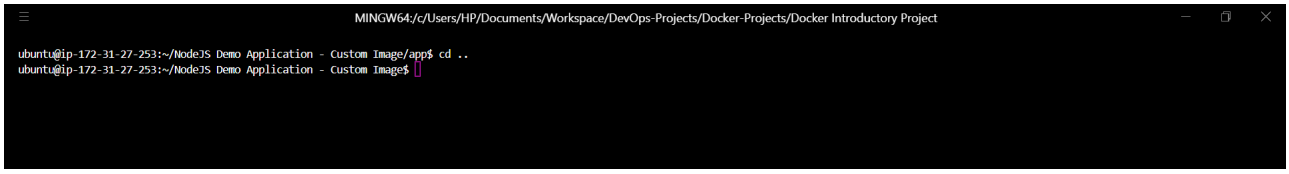
    `ls && cat package.json`

```
MINGW64:/c/Users/HP/Documents/Workspace/DevOps-Projects/Docker-Projects/Docker Introductory Project

ubuntu@ip-172-31-27-253:~/NodeJS Demo Application - Custom Image/app$ ls
images  index.html  node_modules  package-lock.json  package.json  server.js
ubuntu@ip-172-31-27-253:~/NodeJS Demo Application - Custom Image/app$ cat package.json
{
  "name": "nodejs-app",
  "version": "1.0.0",
  "description": "This is a nodejs app runs by docker",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Oluwaseun Osunsola",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^2.2.1",
    "express": "^5.2.1",
    "mongodb": "^7.0.0"
  }
}
ubuntu@ip-172-31-27-253:~/NodeJS Demo Application - Custom Image/app$
```

30. **Navigate Back to the Project Folder**

    `cd ..`

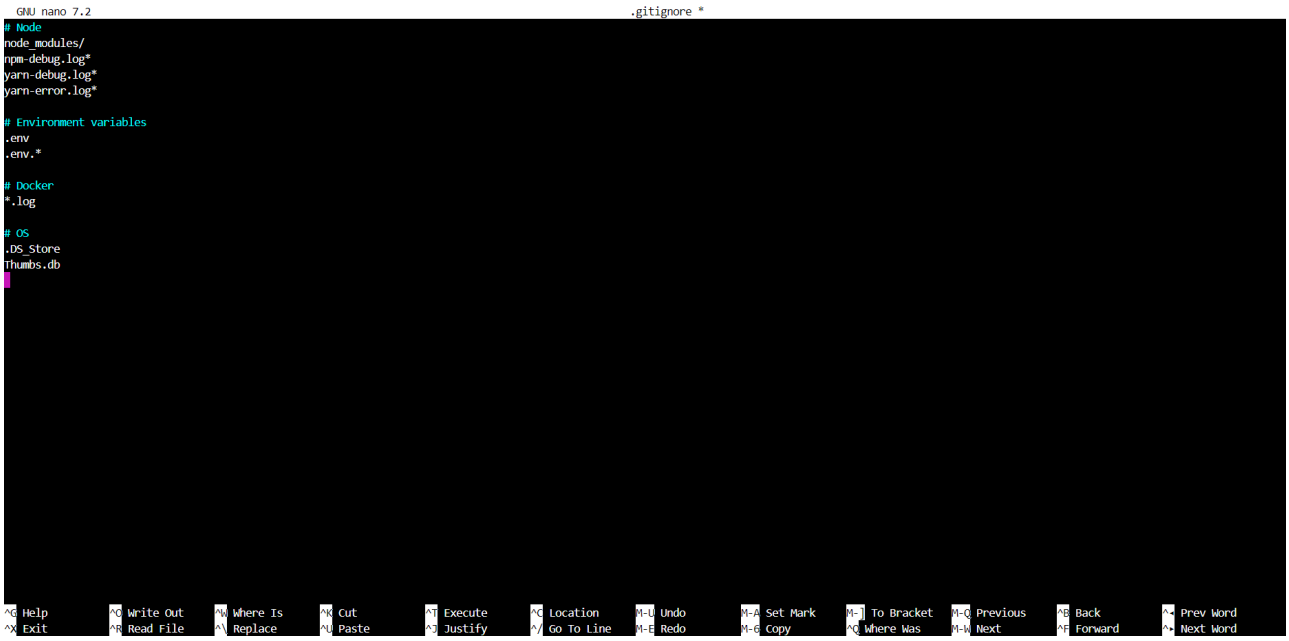


31. **Nano .env Then Add Environment Variables Save and Exit**

    Create `.env` with MongoDB credentials (e.g., `MONGO_USER=admin`, `MONGO_PASSWORD=password`).

32. **Nano .gitignore Then Add What to Ignore Save and Exit**

    Add `node_modules/`, `.env`, etc., to `.gitignore`.

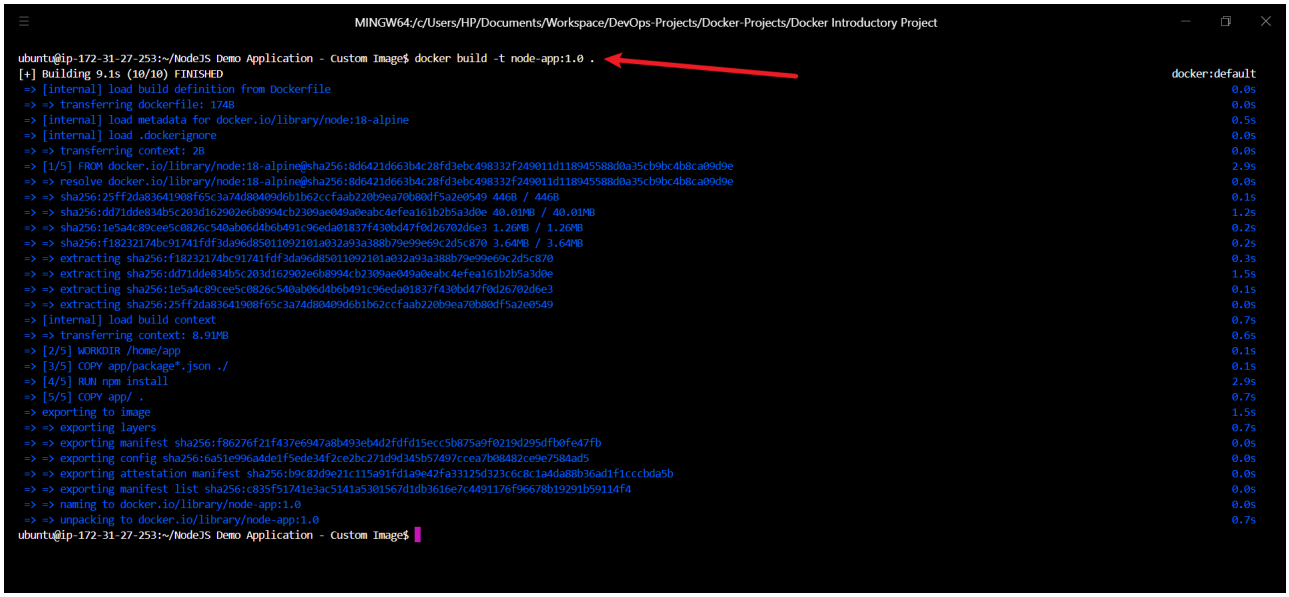


33. **Nano Dockerfile and Add Commands**

    Write the Dockerfile to build the Node.js image.

34. **Build Image Using Docker Build -t node-app:1.0**

    `docker build -t node-app:1.0 .`

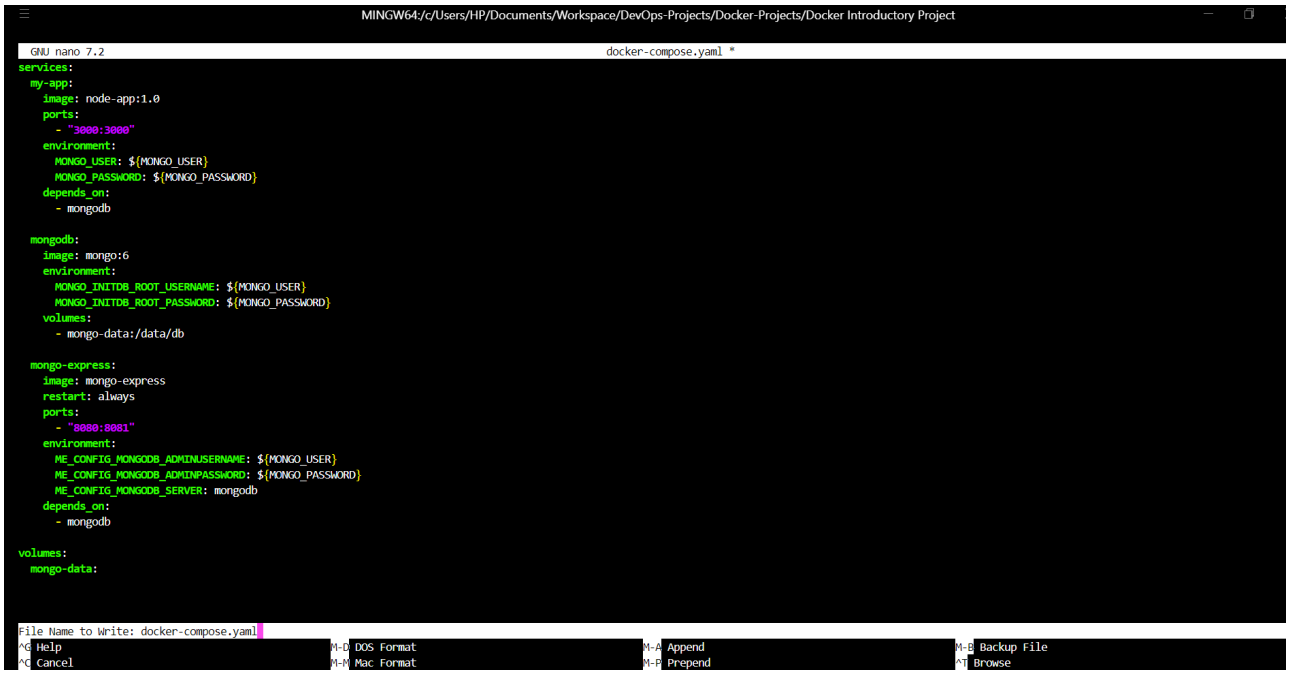


35. **List Images to See the Created node-app:1.0**

    `docker images`

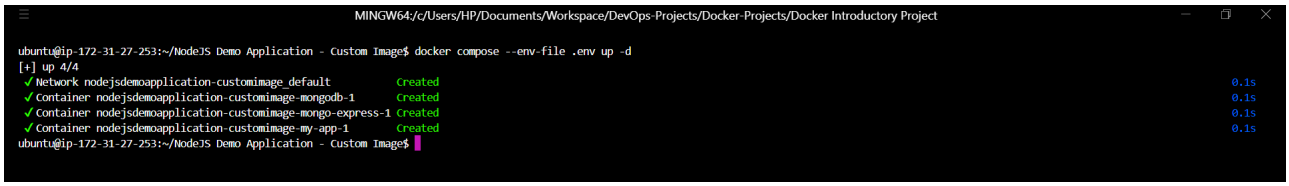
36. **Nano docker-compose.yaml and Add Services and Their Data Save and Exit**

    Define services: `my-app`, `mongodb`, `mongo-express`.



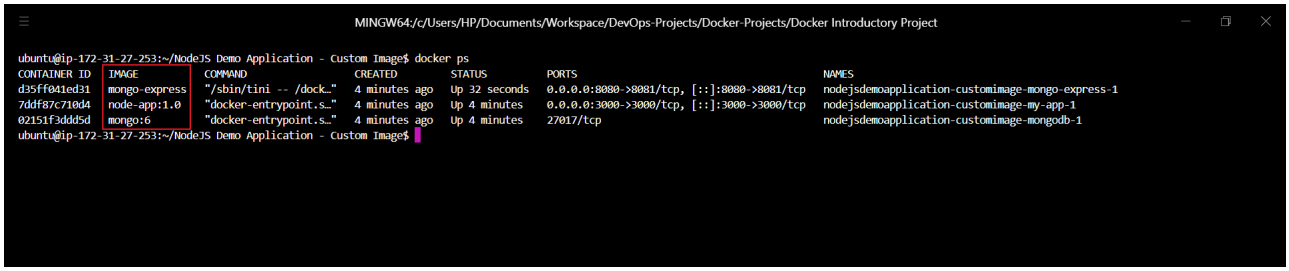
37. **Run Docker Compose --env-file .env Up -d**

```
docker compose up -d
```



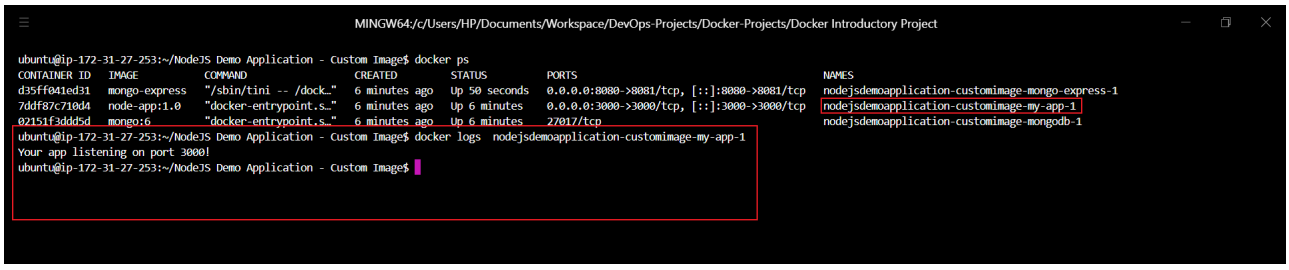
38. **Docker PS to Check Containers Running**

```
docker ps
```



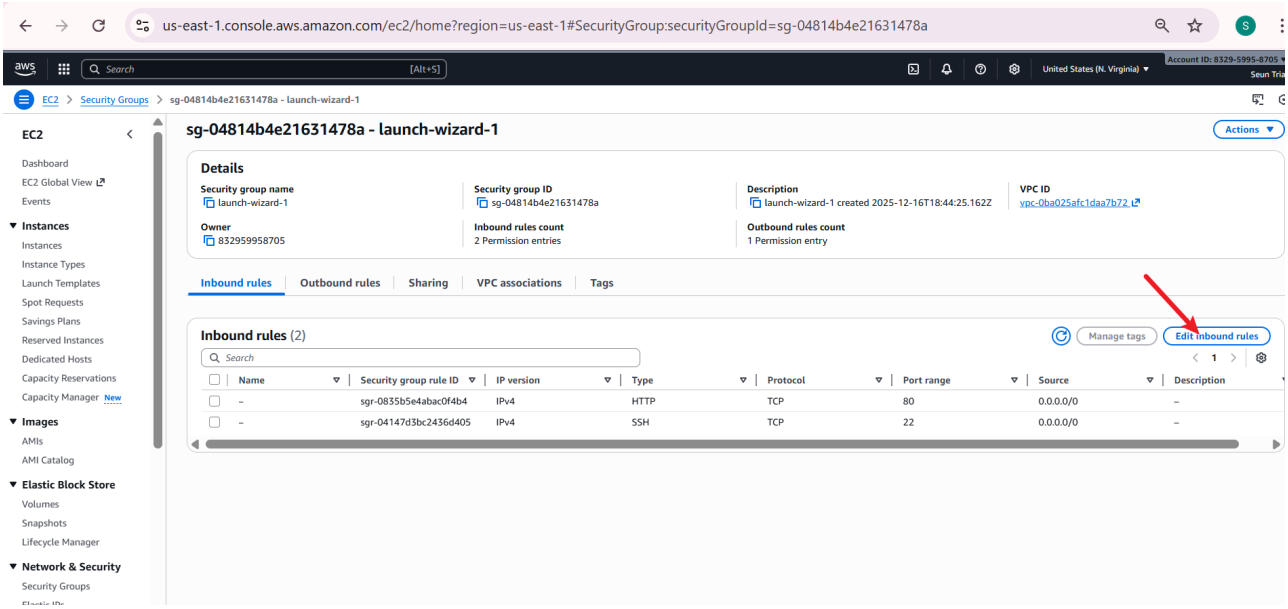
39. **Docker Logs to Check Node.js Application Logs**

```
docker logs <my-app-container-name>
```



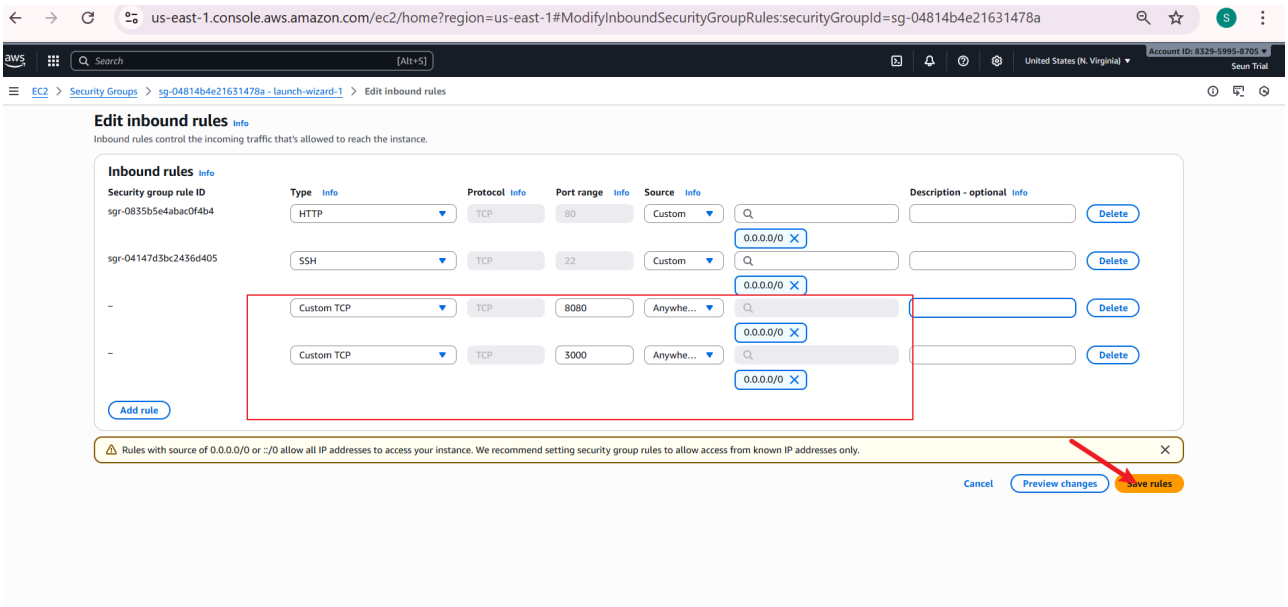
40. **Navigate to Server EC2 Instance Security Group to Edit Inbound Rule**

In AWS Console, open the instance's security group.



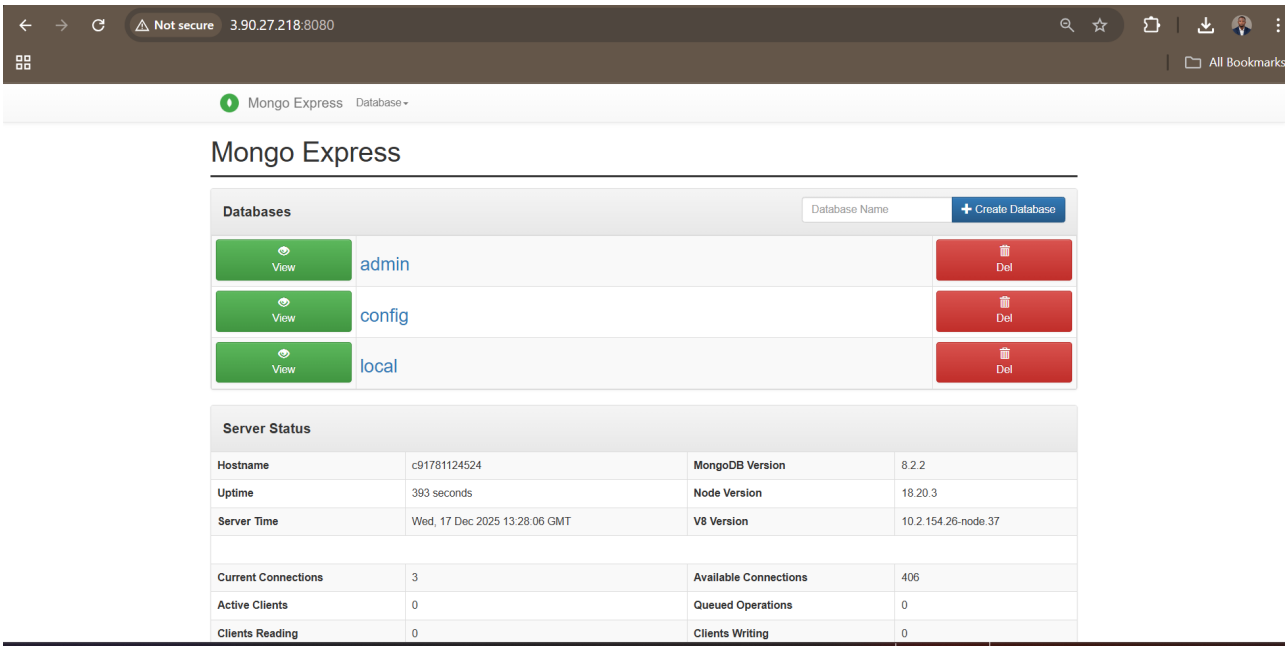
41. **Add Two Rules to Accept Custom TCP Connections from Anywhere on Port 3000 and 8080 Then Save Rules**

    Add rules: Type=Custom TCP, Port=3000 & 8080, Source=0.0.0.0/0.



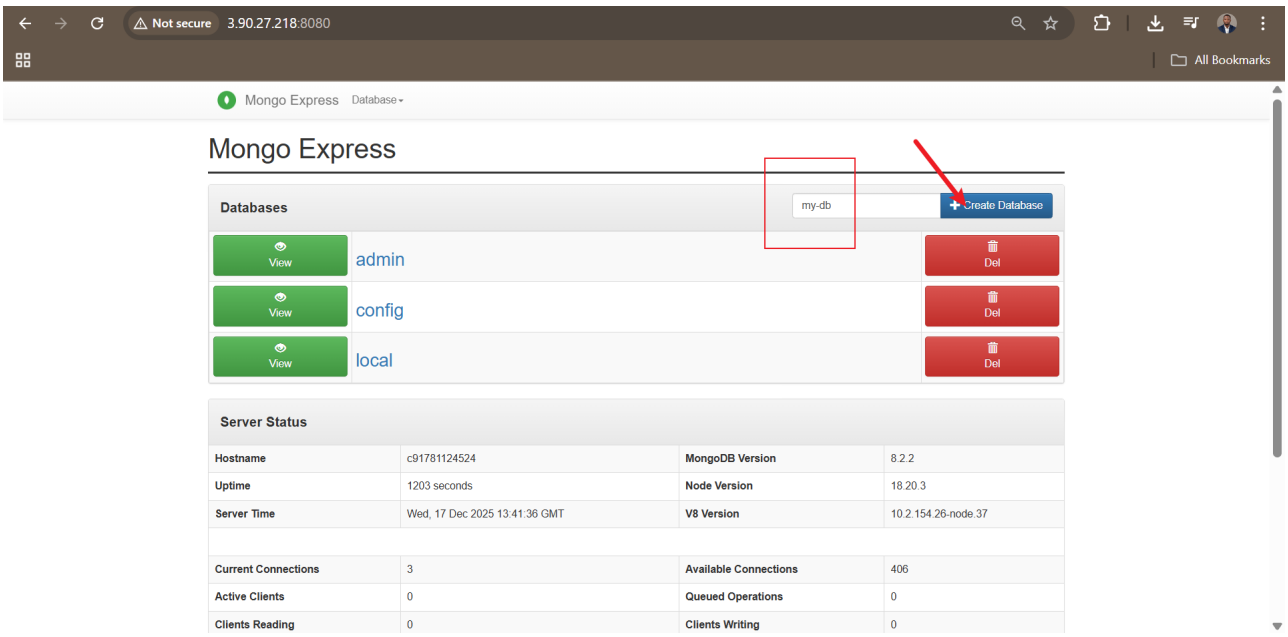
42. **Visit the Public IP on Port 8080 to See Mongo Express**

    http://ec2-public-ip:8080 → Login with credentials from `.env`.



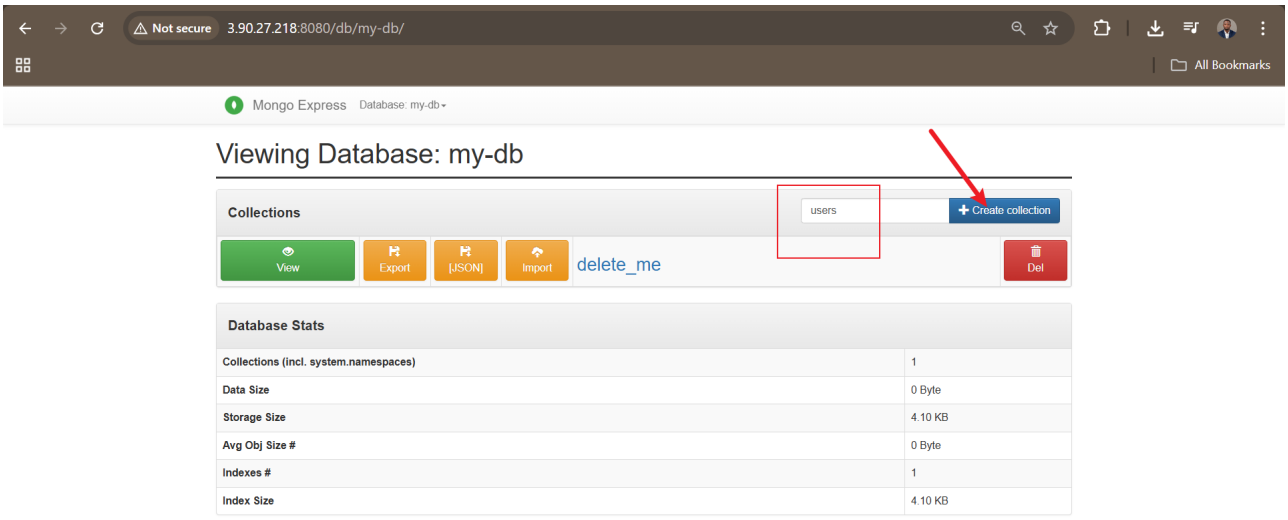
43. **Create my-db Database**

    In Mongo Express, create database `my-db`.

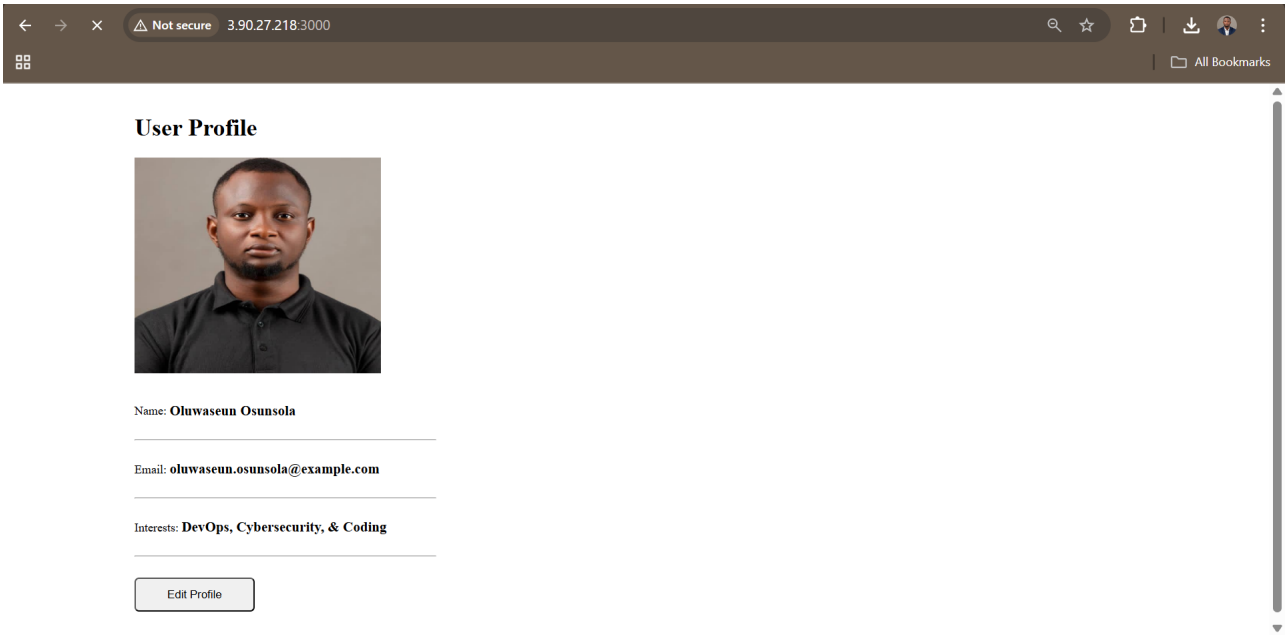


44. **Click on my-db and Create Users Collection**

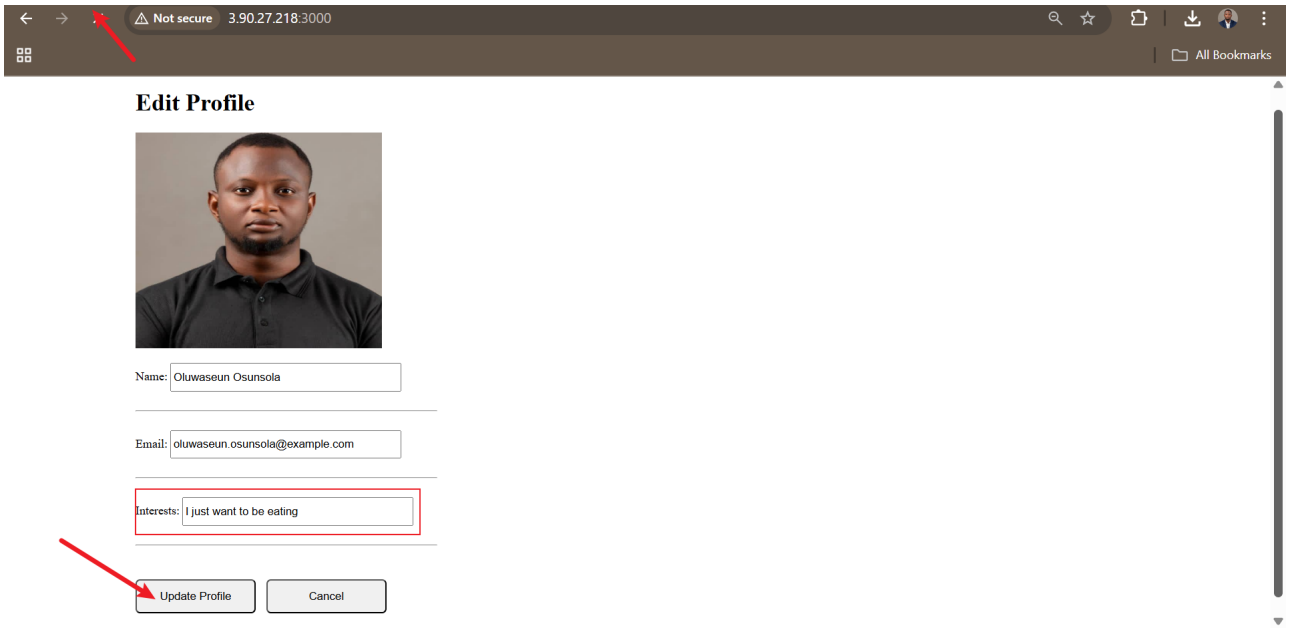
    Create collection `users`.



45. **Visit the Public IP on Port 3000 to See the Application**
    http://ec2-public-ip:3000 → Profile editor loads.



46. **Edit and Update Profile Then Reload to See If Change Is Persistent Across Reload**
    Edit fields, save, reload page → Data persists (stored in MongoDB).



Key Learnings & Best Practices

- **Docker Fundamentals:** Images, containers, volumes, networking.
- **Custom Image Building:** Multi-stage or simple builds for Node.js apps.
- **Docker Compose:** Orchestrating multiple services with dependencies.
- **Environment Variables:** Secure handling via `.env` files.
- **Persistence:** MongoDB volume ensures data survives container restarts.
- **Security Considerations:** Restrict security group rules in production (avoid 0.0.0.0/0).

Troubleshooting Tips

- Container restarting? Check logs: `docker compose logs my-app`.
- Connection refused? Verify security group ports and `docker ps`.
- DB not connecting? Ensure env vars are passed and MongoDB is healthy.
- Image build issues? Use `--no-cache` flag.

Conclusion

This project provides a complete, real-world introduction to containerization with Docker. You now have a deployable full-stack application running on cloud infrastructure. Extend this further by adding authentication, CI/CD pipelines, or deploying to Kubernetes.

Congratulations on completing the tutorial! Your Dockerized Node.js + MongoDB application is live and production-ready for demo purposes.