

## PRACTICAL: 6

### AIM:

#### Scenario:

The café owners have noticed how popular their gourmet coffee offerings have become. Customers cannot seem to get enough of their cappuccinos and lattes. Meanwhile, the café owners have been challenged to consistently source the highest quality coffee beans. Recently, the owners learned that one of their favorite coffee suppliers wants to sell her company. Frank and Martha jumped at the opportunity to buy the company. The acquired coffee supplier runs an inventory tracking application on an AWS account. Sofía has been tasked to understand how the application works and then create a plan to integrate the application into the café's existing application infrastructure. In this lab, you will again play the role of Sofía, and you will work to migrate the application to run on containers. By the end of this lab, you will have migrated the application and the backend database to run as Docker containers. You will register these two containers in Amazon ECR to make them available to deploy as needed.

#### Lab overview and objectives:

In this lab, you will migrate a web application to run on Docker containers. The application is installed directly on the guest operating systems (OSs) of two Amazon Elastic Compute Cloud (Amazon EC2) instances. You will migrate the application to run on Docker containers.

After completing this lab, you should be able to:

- Create a Dockerfile.
- Create a Docker image by using a Dockerfile.
- Run a container from a Docker image.
- Interact with and administer your containers.
- Create an Amazon Elastic Container Registry (Amazon ECR) repository.
- Authenticate the Docker client to Amazon ECR.
- Push a Docker image to Amazon ECR.

### THEORY:

Migrating the café's inventory tracking application to Docker containers involves several key steps to modernize and streamline its deployment. The process begins with creating a **Dockerfile**, which serves as a blueprint for the application's environment, specifying the base image, dependencies, and configurations required to run the application. Using this Dockerfile, a **Docker image** is built, packaging the application and its dependencies into a consistent, reusable format.

Once the image is ready, it is used to run the application as a **Docker container**, providing a lightweight, isolated environment that ensures portability and efficient resource utilization. Managing these containers involves using Docker commands to start, stop, inspect, and monitor their operations. This approach simplifies administration and enhances the reliability of the application.

To facilitate deployment on AWS, an **Amazon Elastic Container Registry (ECR)** repository is created. ECR is a secure, fully managed Docker container registry that integrates seamlessly with other AWS services. The **Docker client** is then authenticated to interact with the ECR repository, ensuring secure communication. Finally, the Docker image is **pushed to ECR**, making it available for deployment on platforms like AWS Elastic Container Service (ECS) or Kubernetes. This migration enables the café to achieve portability, scalability, and seamless integration with AWS infrastructure.

**CODE:**

N/A

## OUTPUT:

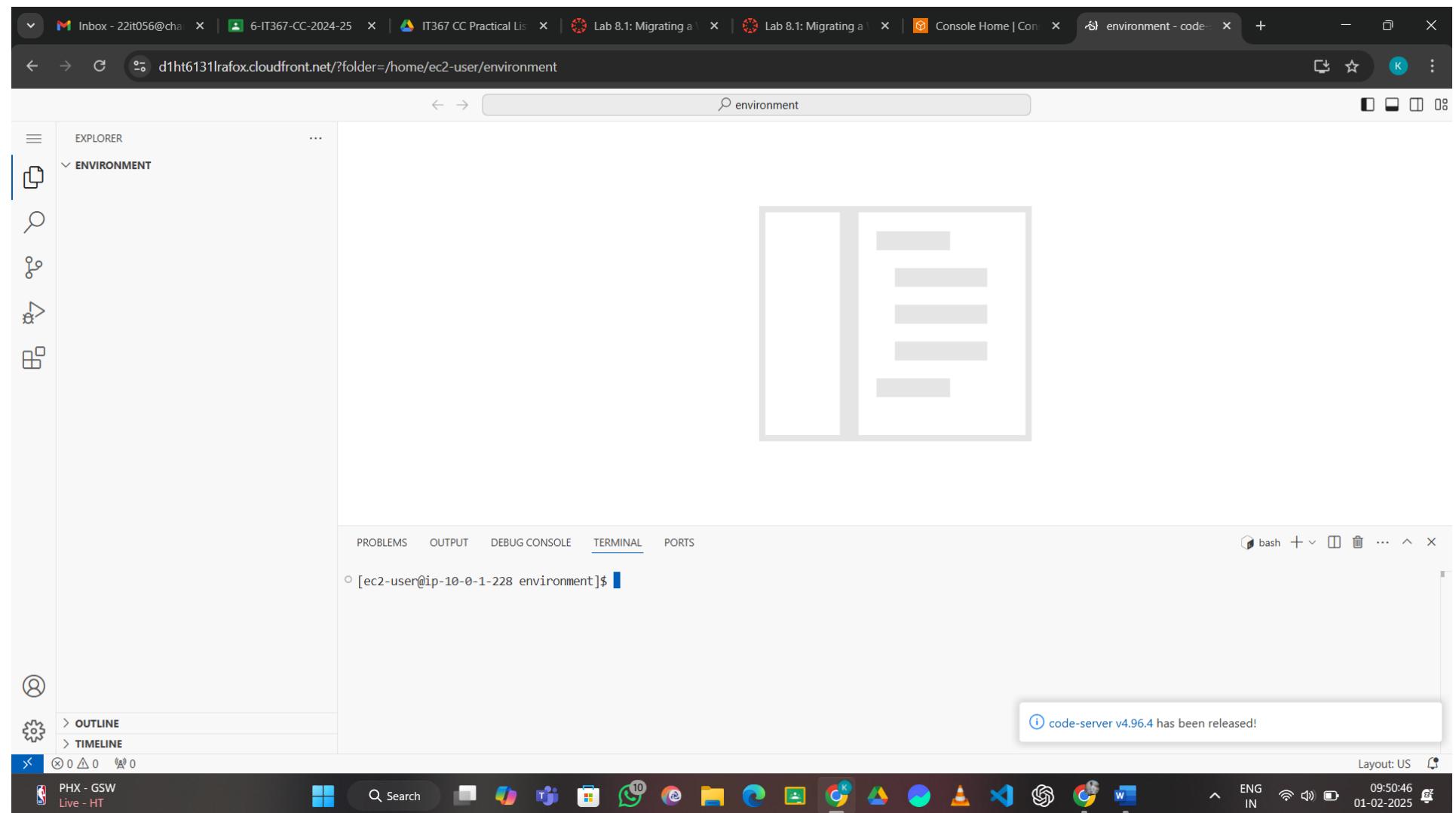


Figure 1: Open vs code in browser

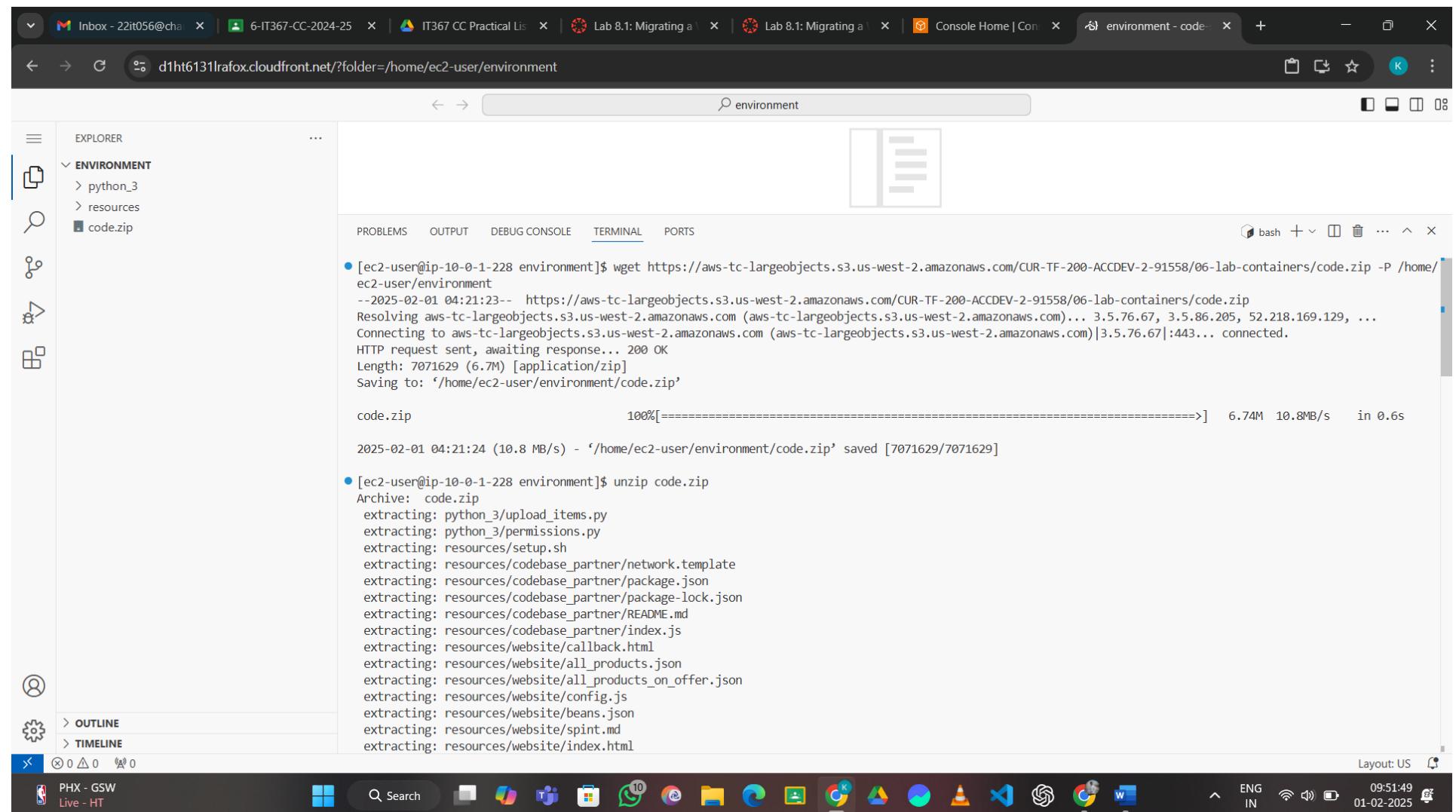


Figure 2: Download a zip file and extract it

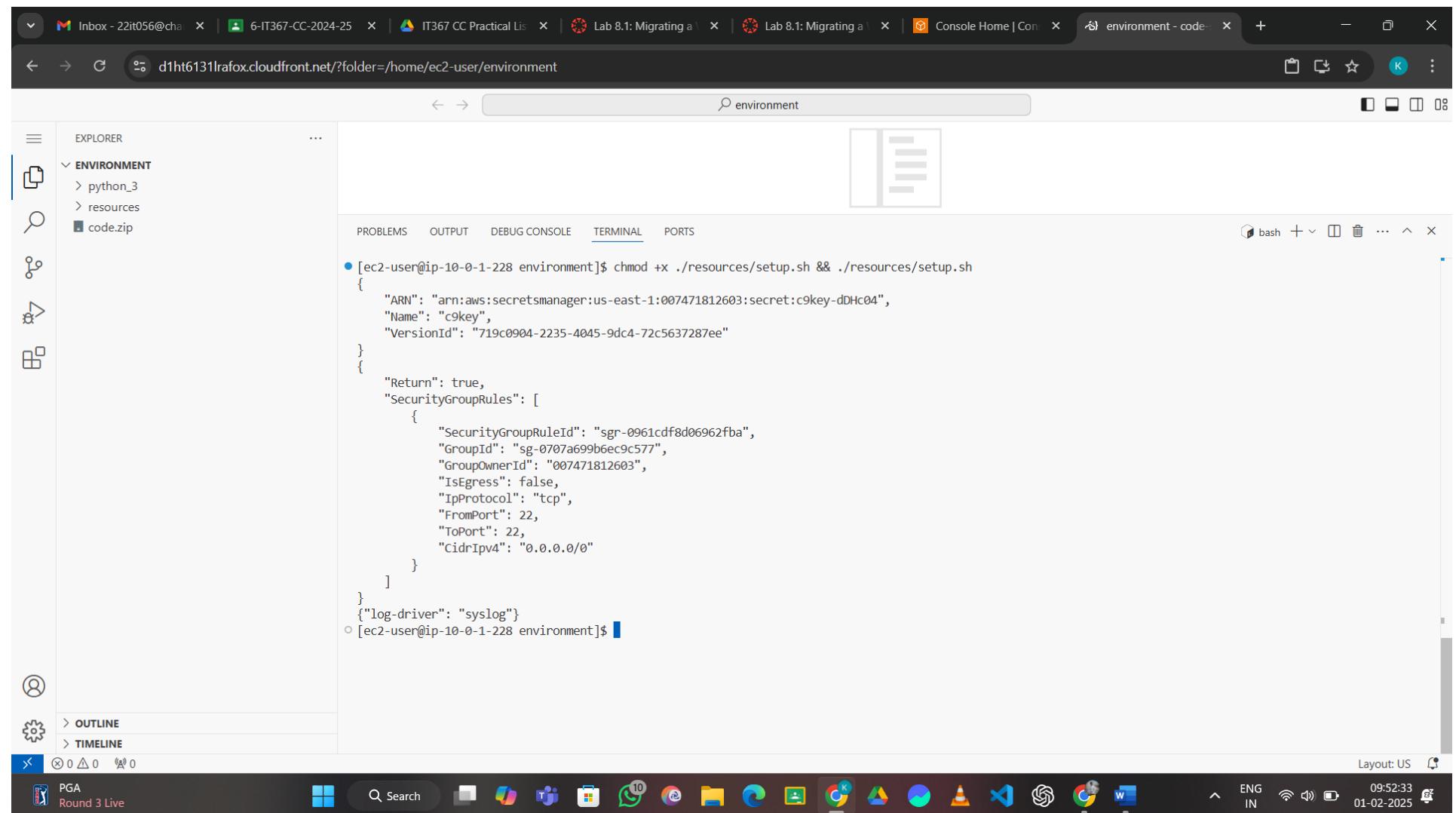


Figure 3: Change mode and access it

The screenshot shows the AWS Management Console interface for the EC2 service. The left sidebar navigation menu includes options like Dashboard, EC2 Global View, Events, Instances (selected), Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, and Network & Security.

The main content area displays the 'Instances (3) Info' section. It lists three running instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
Lab IDE	i-0e15990240b93a63f	Running	t3.medium	3/3 checks passed	View alarms +	us-east-1a	ec2-52-2
MysqlServerN...	i-0733fb864e135af9	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	ec2-18-2
AppServerNode	i-0dac4280b2a8ec77f	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	ec2-54-2

Below the table, a 'Select an instance' dropdown menu is open, listing the same three instances: Lab IDE, MysqlServerN..., and AppServerNode. The bottom of the screen shows the standard AWS footer with links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences, along with system status icons for battery, signal, and network.

Figure 4: Check instances which are running

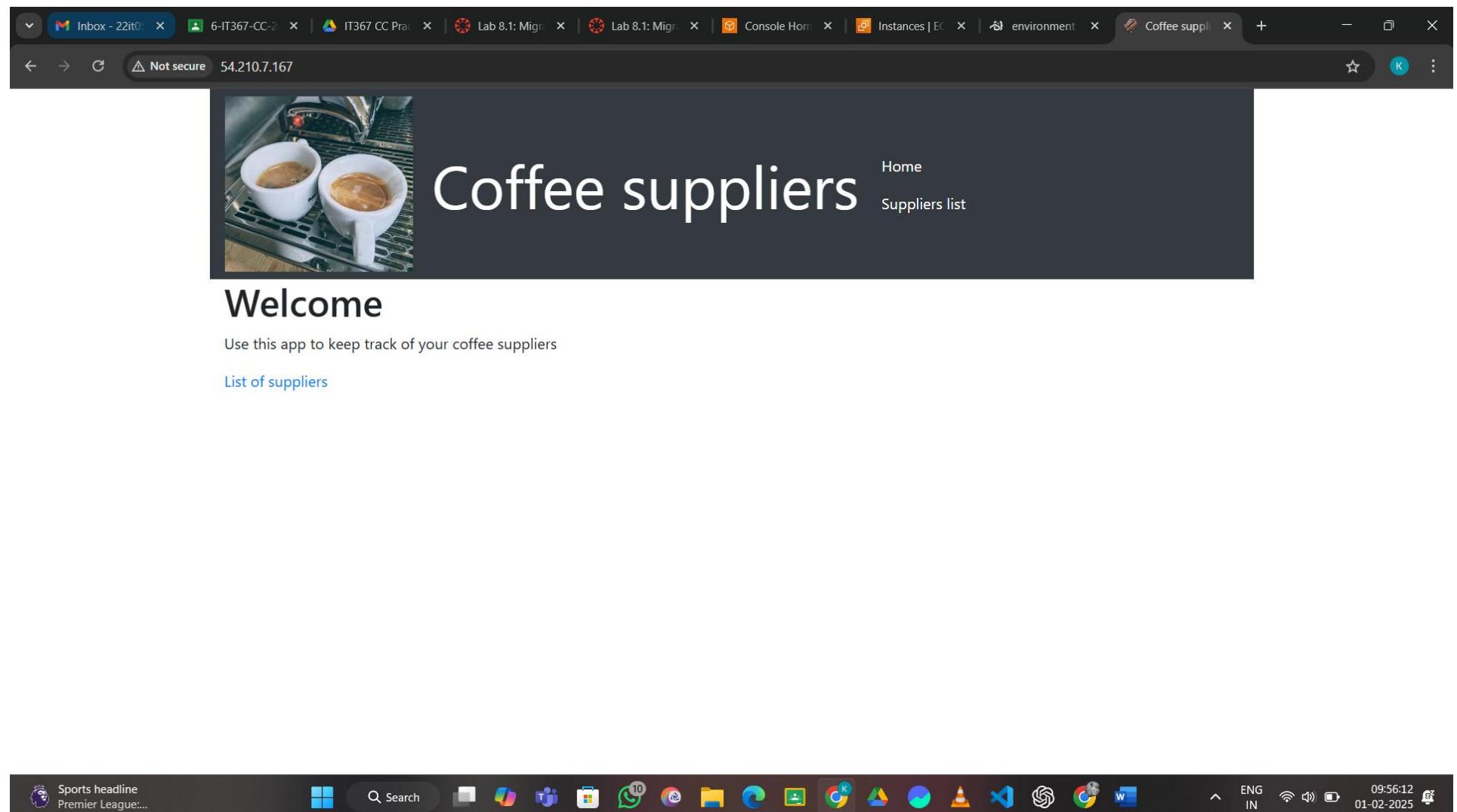


Figure 5: Open a website in browser

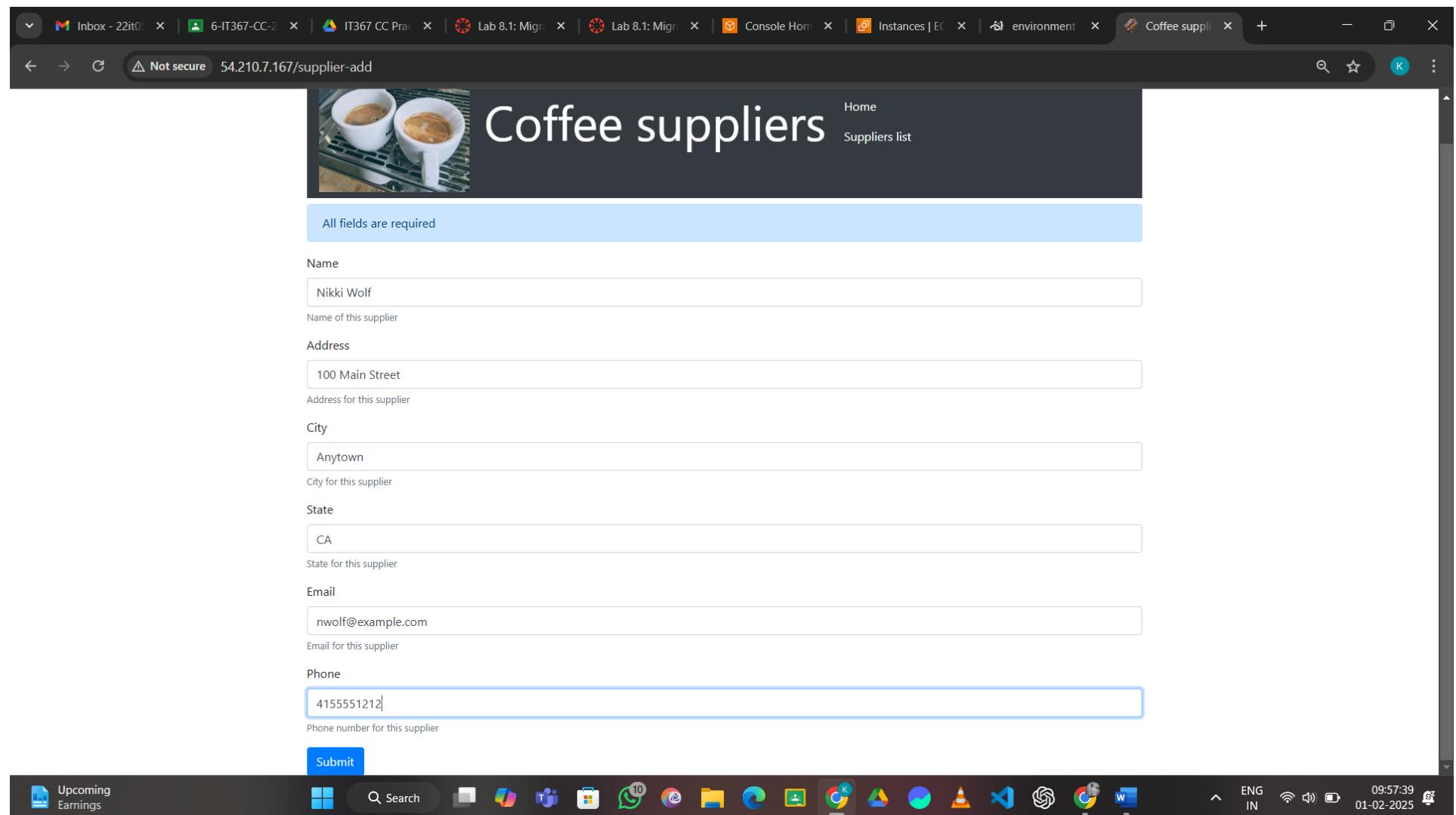


Figure 6: Add new supplier

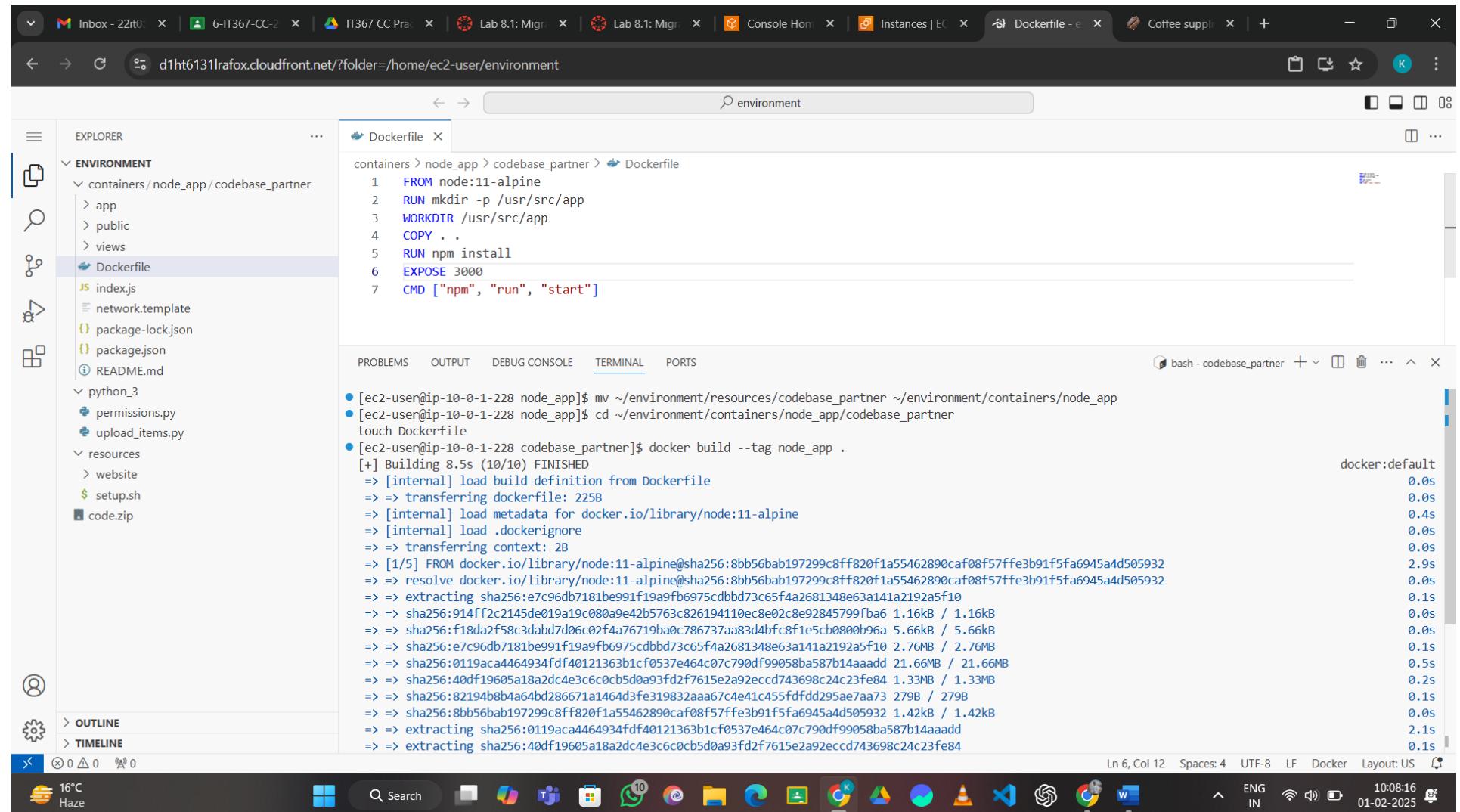


Figure 7: Create a docker file and pushed it

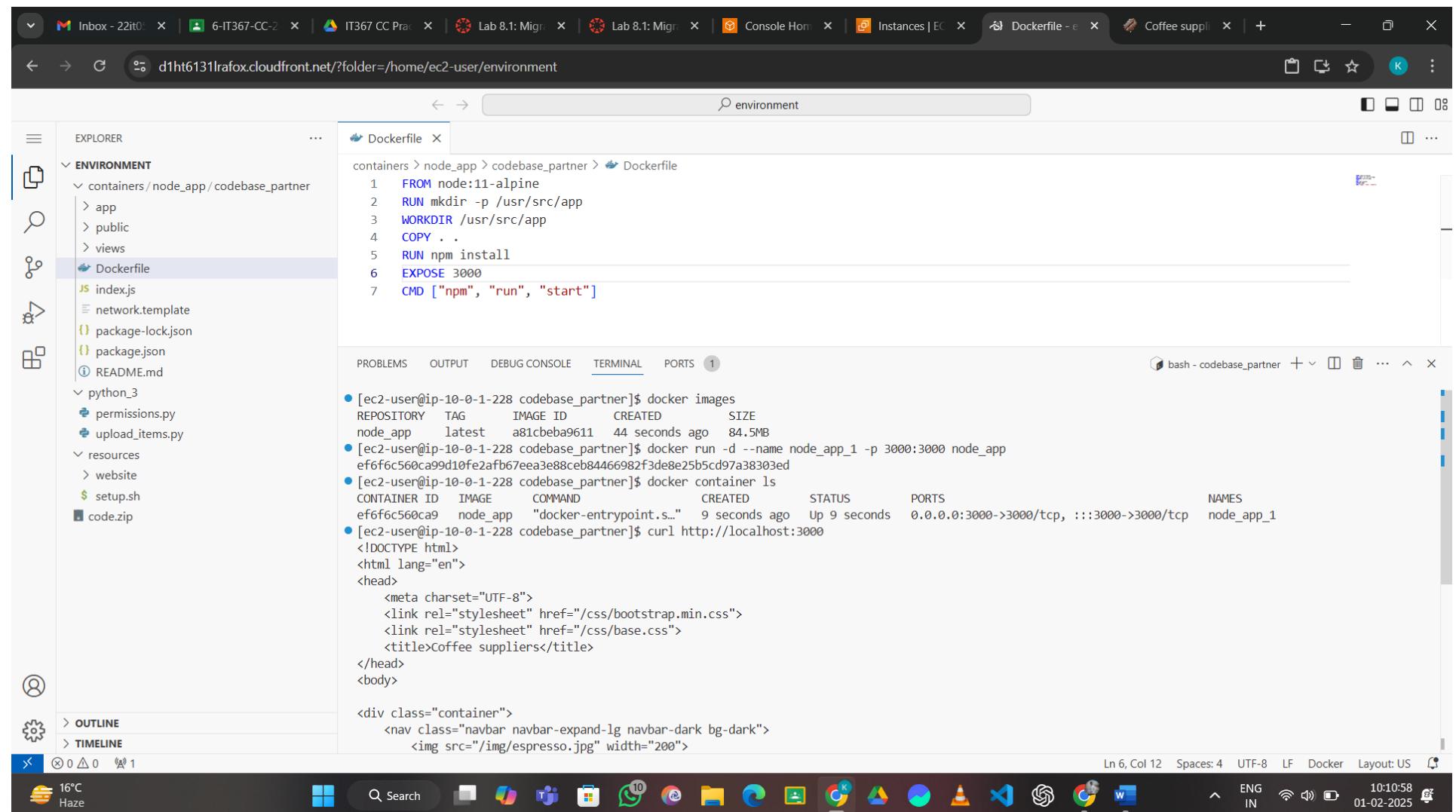


Figure 8: Check docker image and container

The screenshot shows the AWS CloudWatch Metrics interface. At the top, there are tabs for 'Metrics' (selected), 'Logs', 'CloudWatch Metrics Insights', and 'CloudWatch Metrics Insights (Preview)'. Below the tabs, a search bar contains the query 'aws cloudwatch metrics'. The main content area displays a table of metrics with columns: Metric Name, Namespace, Unit, and Last Value. One row is highlighted in blue, showing 'AWS/CloudWatch Metrics' with a unit of 'Count' and a last value of '1'. At the bottom right, there are buttons for 'Edit' and 'Delete'.

Figure 9: Edit inbound rules

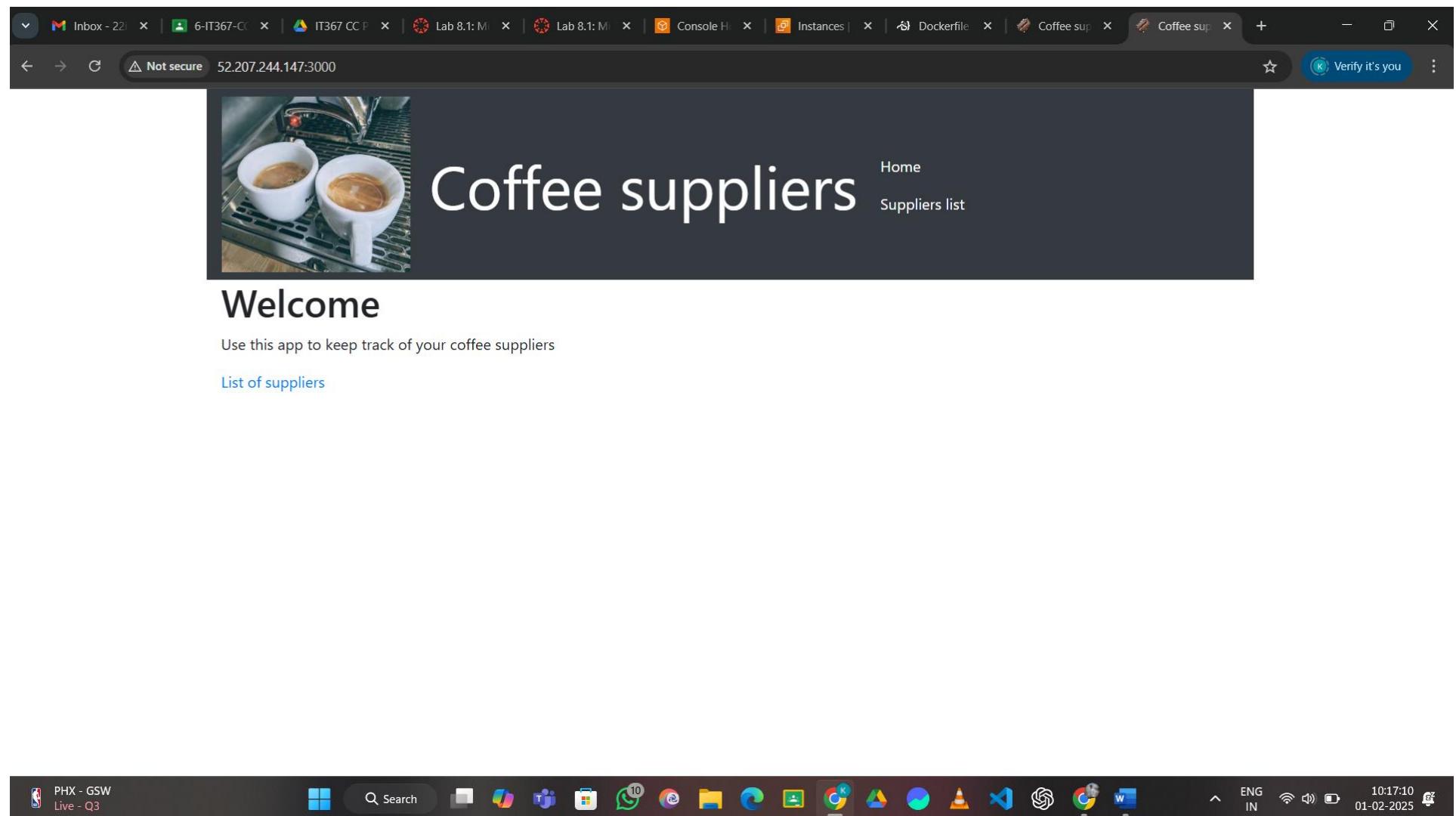


Figure 10: Open a docker file which run on 3000 port

The screenshot shows a browser window with multiple tabs open, including Gmail, 6-IT367-CC, IT367 CC P, Lab 8.1: M, Lab 8.1: M, Console H, Instances, config.js, Coffee sup, and Coffee sup. The address bar shows the URL `d1ht6131lrafox.cloudfront.net/?folder=/home/ec2-user/environment`. The main content area displays a code editor for a `config.js` file within a Docker container. The code defines environment variables `APP_DB_PASSWORD` and `APP_DB_NAME`. Below the code editor is a terminal window showing the output of `docker ps` and `docker exec` commands, which indicate a container named `node_app_1` is running. The terminal also shows the user's session environment variables. The bottom of the screen shows a taskbar with various icons and system status information.

```
let config = {
  APP_DB_PASSWORD: "coffee",
  APP_DB_NAME: "COFFEE"

}

</div>
</nav>  <div class="container">
  <h1>Welcome</h1>
  <p>Use this app to keep track of your coffee suppliers</p>
  <p><a href="/suppliers">List of suppliers</a></p>
</div>
</div>
<script src="/js/jquery-3.6.0.min.js"></script>
<script src="/js/bootstrap.min.js"></script>
</body>
[ec2-user@ip-10-0-1-228 codebase_partner]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
ef6f6c560ca9        node_app           "docker-entrypoint.s..."   11 minutes ago    Up 11 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node_app_1
[ec2-user@ip-10-0-1-228 codebase_partner]$ docker exec -ti ef6f6c560ca9 sh
whoami
/usr/src/app # su node
/usr/src/app $ env
USER=node
NODE_VERSION=11.15.0
HOSTNAME=ef6f6c560ca9
YARN_VERSION=1.15.2
SHLVL=2
HOME=/home/node
LOGNAME=node
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SHELL=/bin/sh
PWD=/usr/src/app
/usr/src/app $
```

Figure 11: Set environment in docker

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays a file tree with a project structure:

- EXPLORER
- ENVIRONMENT
  - containers / node\_app / codebase\_partner
    - app
      - config
    - Dockerfile
    - index.js
    - network.template
    - package-lock.json
    - package.json
    - README.md
    - python\_3
    - permissions.py
    - upload\_items.py
  - resources
    - website
      - images
      - scripts
      - styles
    - all\_products\_on\_offer.json
    - all\_products.json
    - beans.json
    - callback.html
- OUTLINE
- TIMELINE

The right pane shows two tabs: Dockerfile and config.js. The config.js tab is active, displaying the following code:

```
2 let config = {  
5   APP_DB_PASSWORD: "coffee",  
6   APP_DB_NAME: "COFFEE"  
7 }  
8  
9 Object.keys(config).forEach(key => {  
10   if(process.env[key] === undefined){  
11     console.log(`[NOTICE] Value for key '${key}' not found in ENV, using default value. See app/config/config.js`)  
12   } else {  
13     config[key] = process.env[key]  
14   }  
15 });  
16  
17 module.exports = config;  
18
```

Below the code editor is a terminal window titled "bash - codebase\_partner". It shows the following command history:

- [ec2-user@ip-10-0-1-228 codebase\_partner]\$ docker ps
- CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
ef6f6c560ca9 node\_app "docker-entrypoint.s..." 13 minutes ago Up 13 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp node\_app\_1
- [ec2-user@ip-10-0-1-228 codebase\_partner]\$ docker stop node\_app\_1 && docker rm node\_app\_1
- node\_app\_1
- node\_app\_1
- [ec2-user@ip-10-0-1-228 codebase\_partner]\$

The status bar at the bottom of the VS Code window shows the following information: Ln 1, Col 1, Spaces: 2, UTF-8, LF, {}, JavaScript, Layout: US, ENG IN, 10:24:03, 01-02-2025.

Figure 12: stop and remove node\_app\_1

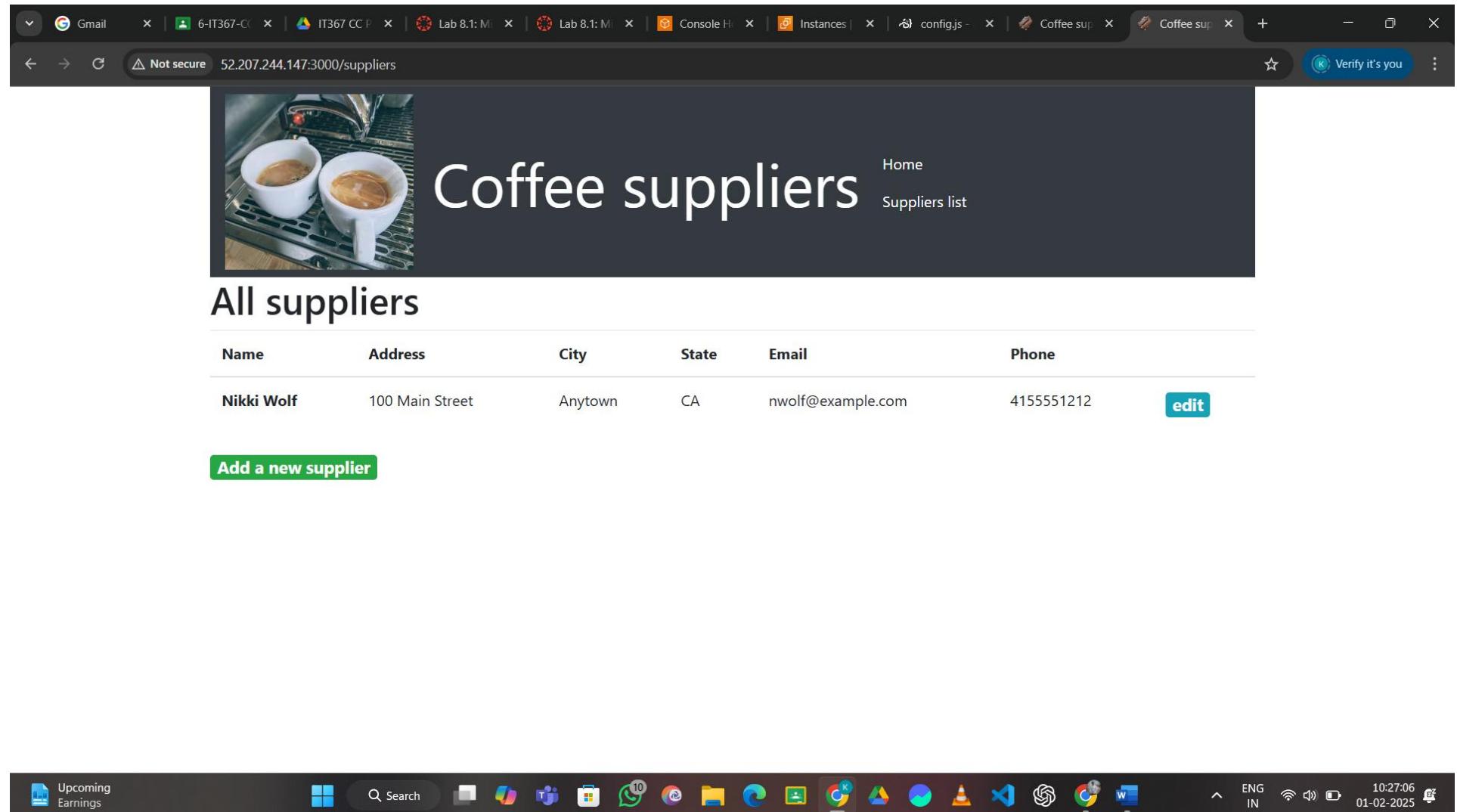


Figure 13: Link a backend in docker file and show all suppliers

The screenshot shows a terminal window within a code editor interface. The terminal is running on a Linux system (Ubuntu) inside a Docker container. The user is executing a MySQL dump command to a file named `my_sql.sql`. The dump includes table locks, data insertion, and various MySQL configuration settings. The terminal also shows the user navigating through the file system and running other commands like curl and docker.

```

environment
containers > my_sql.sql
48
49
50 LOCK TABLES `suppliers` WRITE;
51 /*!40000 ALTER TABLE `suppliers` DISABLE KEYS */;
52 INSERT INTO `suppliers` VALUES (1,'Nikki Wolf','100 Container Street','Anytown','CA','nwolf@example.com','4155551212');
53 /*!40000 ALTER TABLE `suppliers` ENABLE KEYS */;
54 UNLOCK TABLES;
55 /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
56
57 /*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
58 /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
59 /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
60 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
61 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
62 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
63 /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
64
65 -- Dump completed on 2025-02-01 5:02:52
66

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1
bash - mysql + □ ×
① [ec2-user@ip-10-0-1-228 codebase_partner]$ curl http://localhost:3000
curl: (7) Failed to connect to localhost port 3000 after 0 ms: Couldn't connect to server
② [ec2-user@ip-10-0-1-228 codebase_partner]$ docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST="18.212.192.160" node_app
366c7166efca12a53b39fc309ced326a1cf7d3a296d473af5fd7ceda604ffd14
③ [ec2-user@ip-10-0-1-228 codebase_partner]$ http://52.207.244.147:3000/suppliers
bash: http://52.207.244.147:3000/suppliers: No such file or directory
④ [ec2-user@ip-10-0-1-228 codebase_partner]$ cd /home/ec2-user/environment/containers/node_app/codebase_partner
⑤ [ec2-user@ip-10-0-1-228 codebase_partner]$ mysqldump -P 3306 -h 18.212.192.160 -u nodeapp -p --databases COFFEE > ../../my_sql.sql
Enter password:
⑥ [ec2-user@ip-10-0-1-228 codebase_partner]$ cd /home/ec2-user/environment/containers
mkdir mysql
cd mysql
⑦ [ec2-user@ip-10-0-1-228 mysql]$ 
```

Ln 50, Col 31 Spaces: 2 UTF-8 LF MS SQL Layout: US

18°C Smoke

10:35:49 01-02-2025

Figure 14: Edit address of supplier main to container

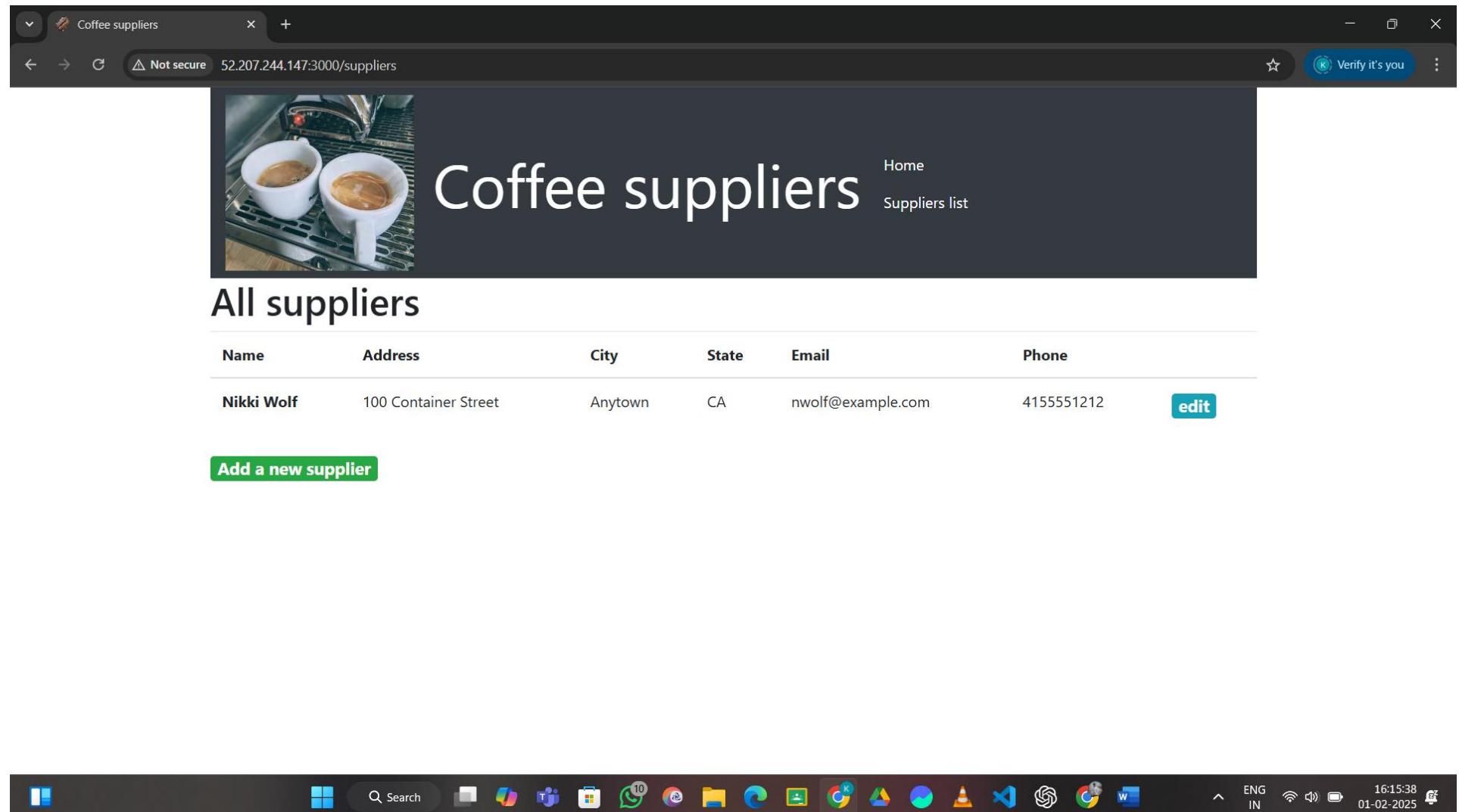


Figure 15: Updated address in browser (frontend)

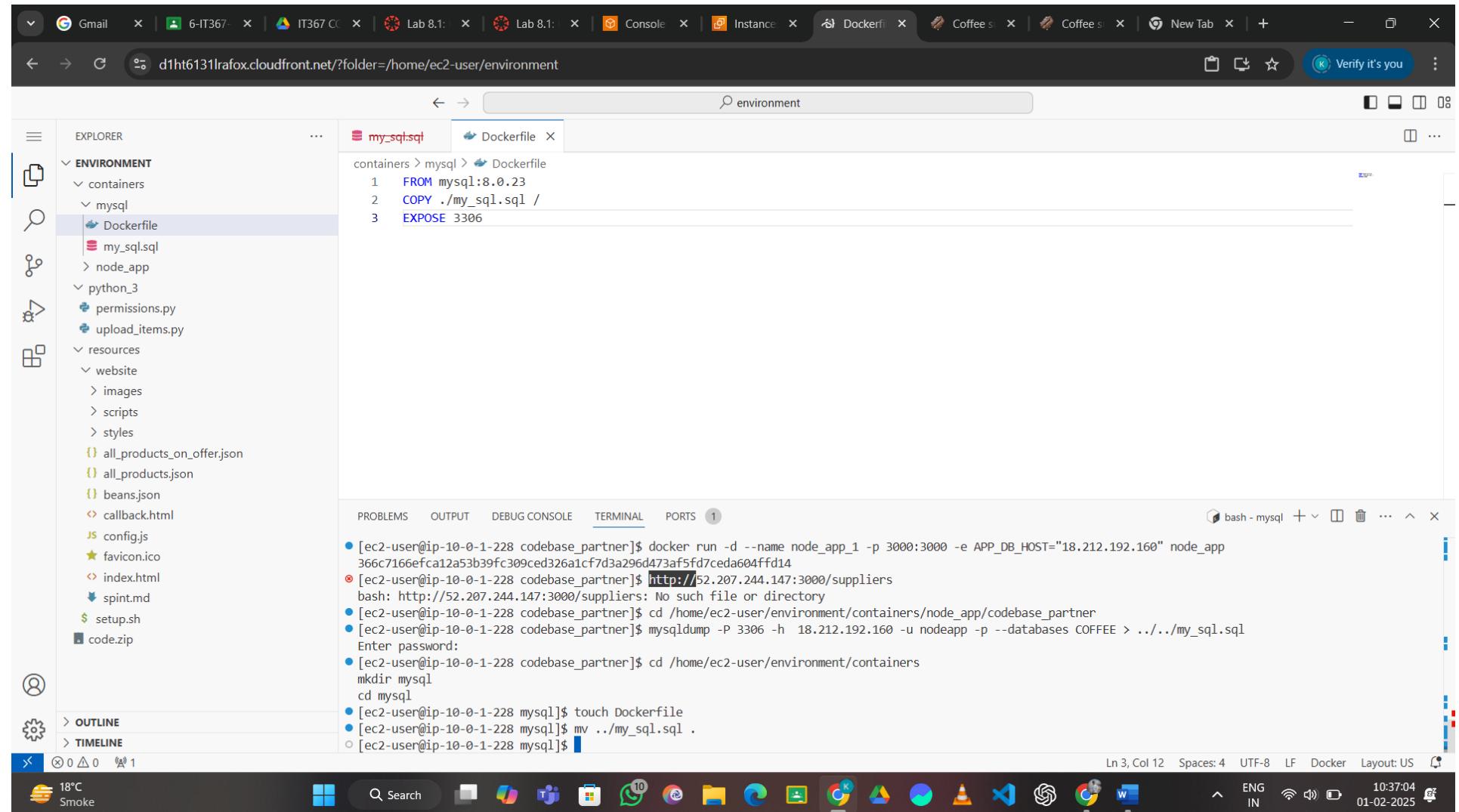


Figure 16: Create a docker file for backend

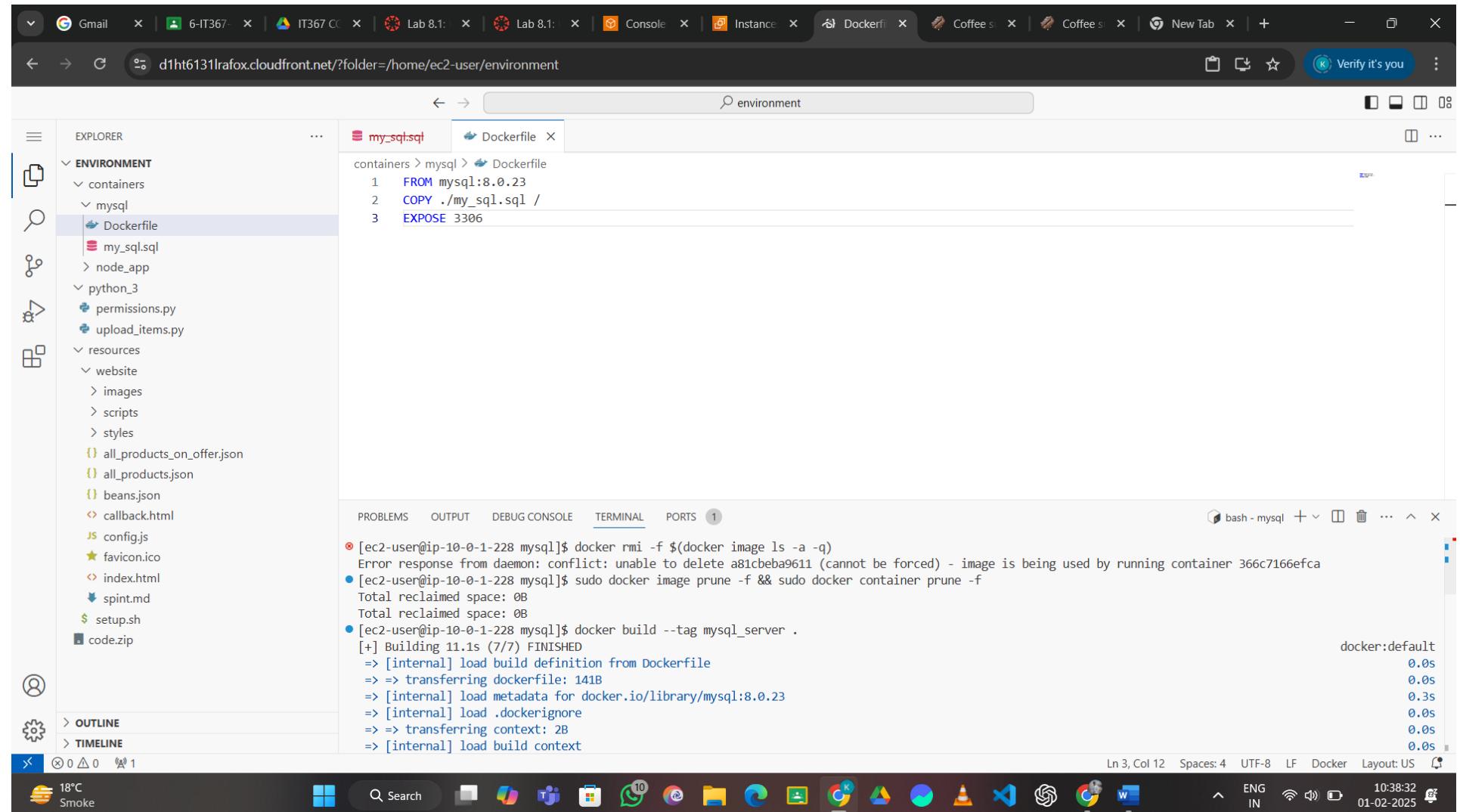


Figure 17: pushed backend docker file

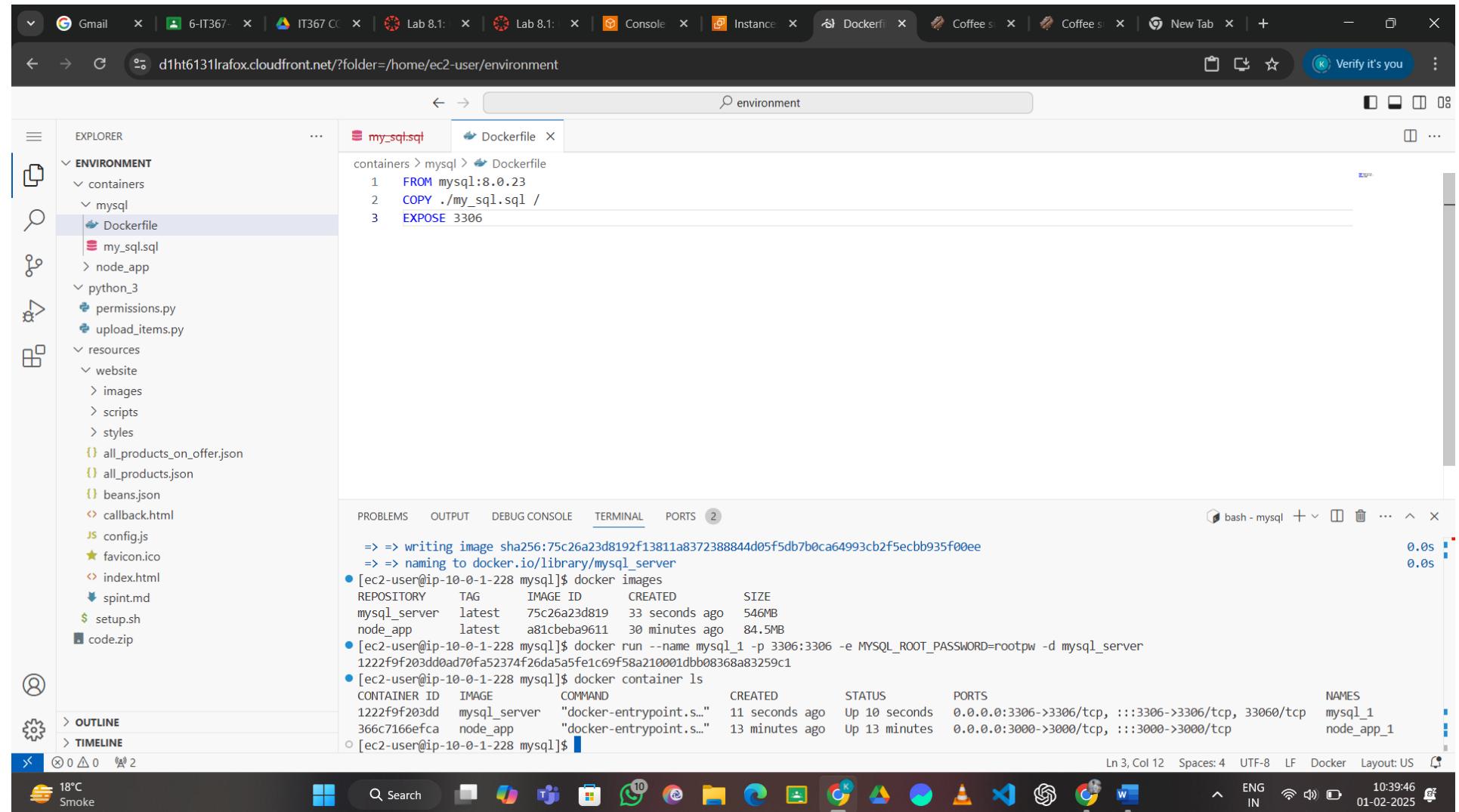


Figure 18: Check all docker file which are created

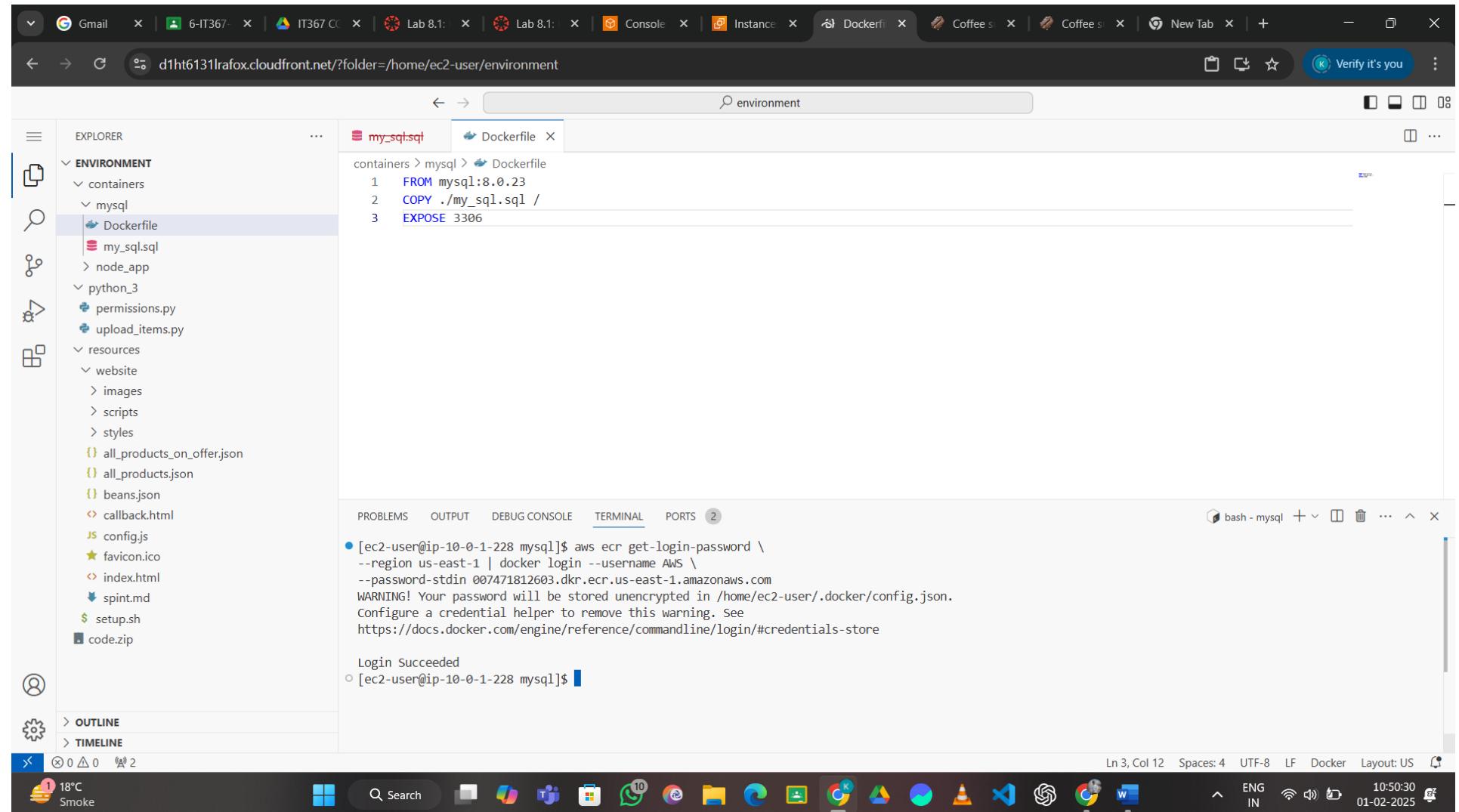


Figure 19: Link a aws account for pushed docker file

The screenshot shows a Microsoft Visual Studio Code (VS Code) window with the following details:

- Explorer View:** Shows a file tree with a folder named "my\_sql" containing "my\_sql.sql", "Dockerfile", and several Python files like "node\_app", "python\_3", "permissions.py", "upload\_items.py". It also contains "resources" with "website", "images", "scripts", "styles", and JSON files.
- Terminal View:** Displays the command-line output of a Docker session:

```
[ec2-user@ip-10-0-1-228 environment]$ docker tag node_app:latest 007471812603.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
[ec2-user@ip-10-0-1-228 environment]$ docker images
REPOSITORY           TAG      IMAGE ID   CREATED        SIZE
mysql_server          latest   75c26a23d819  14 minutes ago  546MB
007471812603.dkr.ecr.us-east-1.amazonaws.com/node-app    latest   a81cbeba9611  44 minutes ago  84.5MB
node_app              latest   a81cbeba9611  44 minutes ago  84.5MB
[ec2-user@ip-10-0-1-228 environment]$ docker push 007471812603.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
The push refers to repository [007471812603.dkr.ecr.us-east-1.amazonaws.com/node-app]
954a3601338a: Pushed
f1b7ea0b0a7f: Pushed
5f70bf18a086: Pushed
750c9365c903: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest: sha256:829ca6bd4e30e343e8abdae0a96d76aa39b101418f1544ec83261fd9ecf46939 size: 1992
[ec2-user@ip-10-0-1-228 environment]$
```
- Status Bar:** Shows system information including temperature (18°C), battery level (Smoke), and system date/time (01-02-2025, 10:53:32).

Figure 20: Pushed all docker file

## LATEST APPLICATIONS:

1. Generative AI
2. Edge Computing
3. Decentralized Finance (DeFi)
4. Metaverse Platforms
5. Autonomous Vehicles
6. Wearable Health Devices
7. 5G-Enabled Solutions
8. Smart Home Automation
9. Cybersecurity Platforms
10. Green Tech Applications

## LEARNING OUTCOME:

Through this practical, I will gain hands-on experience in containerizing applications using Docker and deploying them on AWS. I will learn to create Dockerfiles, build and run Docker images, and manage containers effectively. Additionally, I will practice setting up and using Amazon Elastic Container Registry (ECR) to securely store and deploy container images, enhancing my ability to work with cloud-based infrastructures.

## REFERENCE:

1. <https://awsacademy.instructure.com/courses/104050/>