PRACTICAL: 7

AIM:

Scenario:

Sofía has containerized the coffee suppliers application. Now the café has asked if she can reduce the required manual application maintenance and plan for the future scalability of the environment. She knows from her studies that Amazon Web Services (AWS) provides several managed services. Managed services can help to reduce the burden of deploying and managing applications. After more research, she has decided to deploy the suppliers application website using Elastic Beanstalk. However, Sofía has discovered that scaling a relational database using containers is not recommended because relational databases are stateful. Relational databases require reliable communication between database hosts and storage. This is difficult to accomplish using dynamic containers. Therefore, Sofía has decided to use Aurora Serverless as the data platform. Sofía will retire the container-based MySQL database and load the required user, tables, and data into an Aurora Serverless database. Aurora Serverless was designed to seamlessly and safely scale databases as transaction loads increase. As a bonus, when the database isn't being used, Aurora Serverless automatically scales down, which will save money for the café. In this lab, you will again play the role of Sofía, and you will deploy the suppliers application using managed services.

Lab overview and objectives:

In a previous lab, you migrated an application that ran on Amazon Elastic Compute Cloud (Amazon EC2) instances to run on Docker containers. In this lab, you will deploy the application using two managed cloud services. You will deploy the database tier using Amazon Aurora Serverless and the web tier using AWS Elastic Beanstalk.

After completing this lab, you should be able to:

- Create a new Amazon Relational Database Service (Amazon RDS) instance using the AWS Management Console.
- Launch a Docker container on AWS Cloud9 using an image pulled from Amazon Elastic Container Registry (Amazon ECR).
- Configure and test the containerized application connection to Aurora Serverless. · Use the Amazon RDS query editor to create database objects and load data. · Launch the default Elastic Beanstalk application.
- Update the Elastic Beanstalk application to run your node application and communicate with Amazon RDS.
- Configure an Amazon API Gateway endpoint to forward calls to the Elastic Beanstalk URL.

THEORY:

Deploying applications using **AWS managed services** simplifies infrastructure management, enhances scalability, and optimizes cost efficiency. In this lab, the suppliers' application is containerized and deployed using **AWS Elastic Beanstalk** for the web tier and **Amazon Aurora Serverless** for the database tier. Elastic Beanstalk automates deployment, scaling, and monitoring, ensuring that the application remains responsive to changing traffic demands. Meanwhile, Aurora Serverless dynamically

scales the database capacity based on workload, eliminating manual provisioning and reducing costs when the database is idle.

A key challenge in scaling containerized relational databases is maintaining **stateful connections** and **persistent storage**, which are difficult to achieve with ephemeral containers. To solve this, the lab replaces the **container-based MySQL database** with Aurora Serverless, which offers automated scaling, high availability, and built-in security. The application is further optimized by integrating **Amazon API Gateway**, which provides a secure and scalable way to route external requests to the Elastic Beanstalk instance. This cloud-native approach ensures **better performance, reliability, and reduced operational overhead**, allowing the café to focus on business growth rather than infrastructure management.

CODE:

Options.txt

```
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-0baa7641c54d63e1a"
  },
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "VPCId",
    "Value": "vpc-0b717faded93b7380"
  },
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "Subnets",
    "Value": "subnet-0c4603bf5623ebaaf,subnet-010ea7c74c2797c6d"
  },
  {
    "Namespace": "aws:elasticbeanstalk:application:environment",
    "OptionName": "APP_DB_HOST",
    "Value": "supplierdb-instance-1.cgds16dxo67k.us-east-1.rds.amazonaws.com"
  }
]
```

OUTPUT:

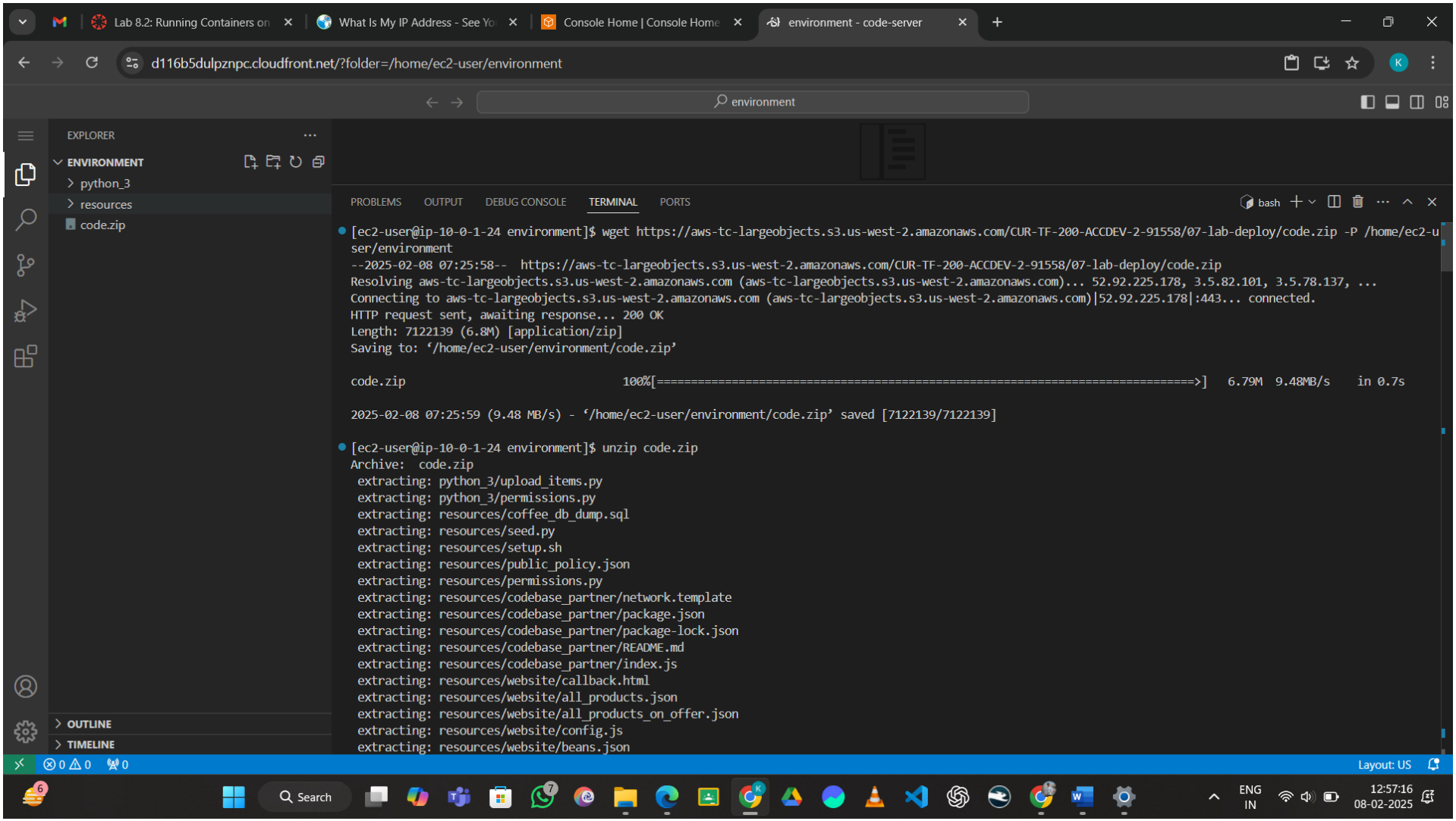

Figure 1: Download and extract file

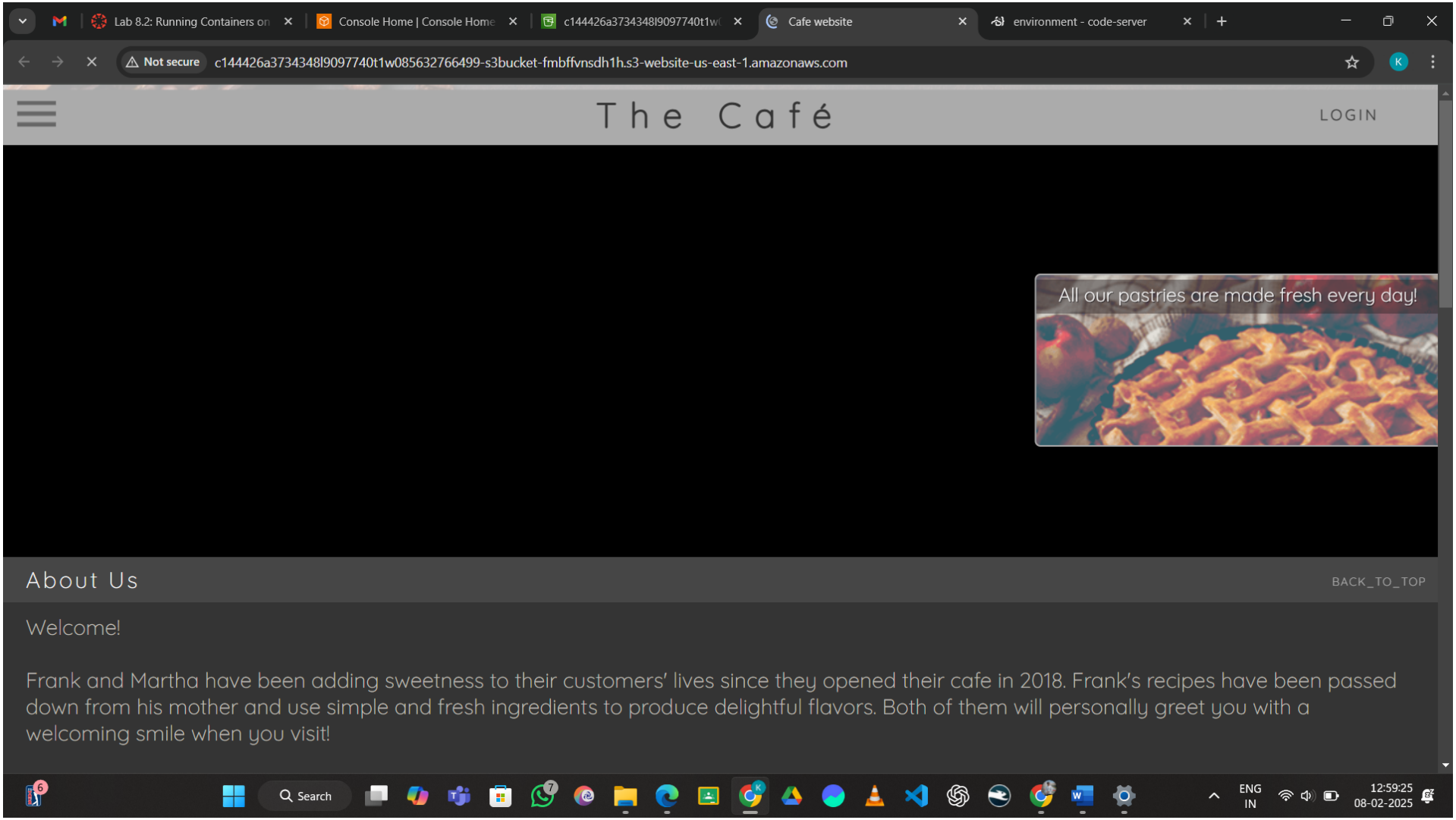

Figure 2: Load a cafe website

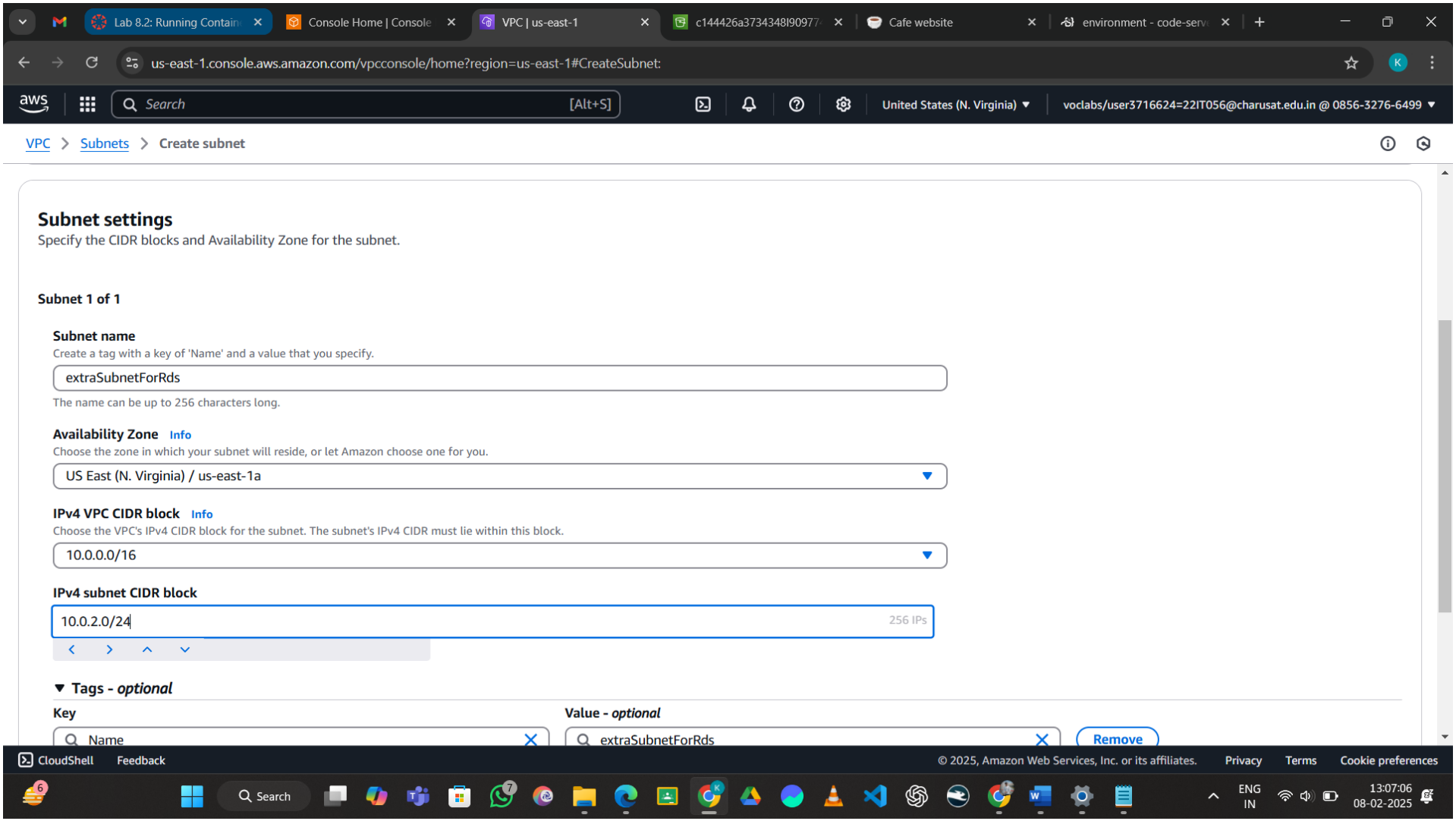

Figure 3: Creating a subnet

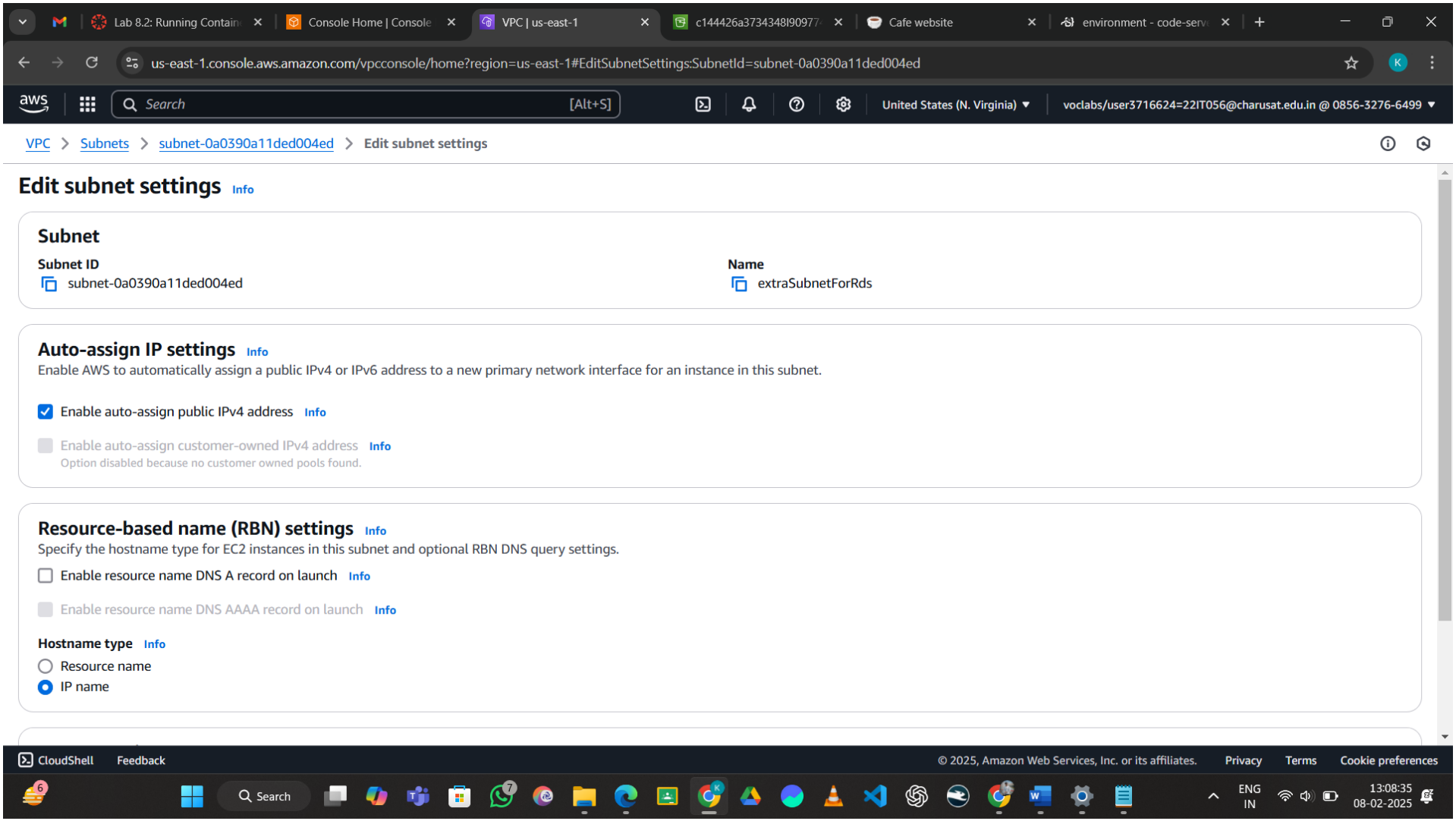

Figure 4: Enable auto ipv4 address in creating a subnet

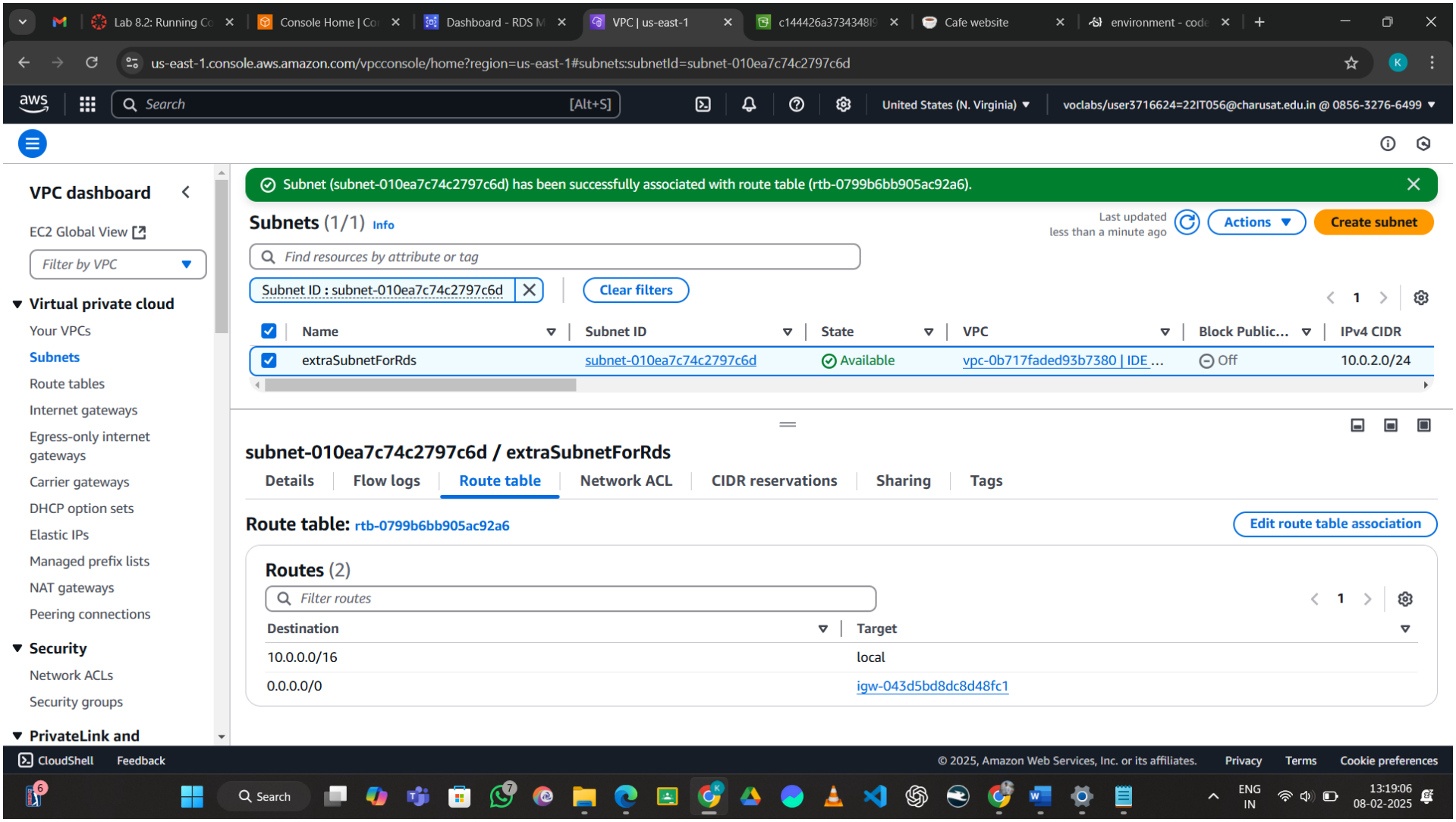

Figure 5: Verify edited information

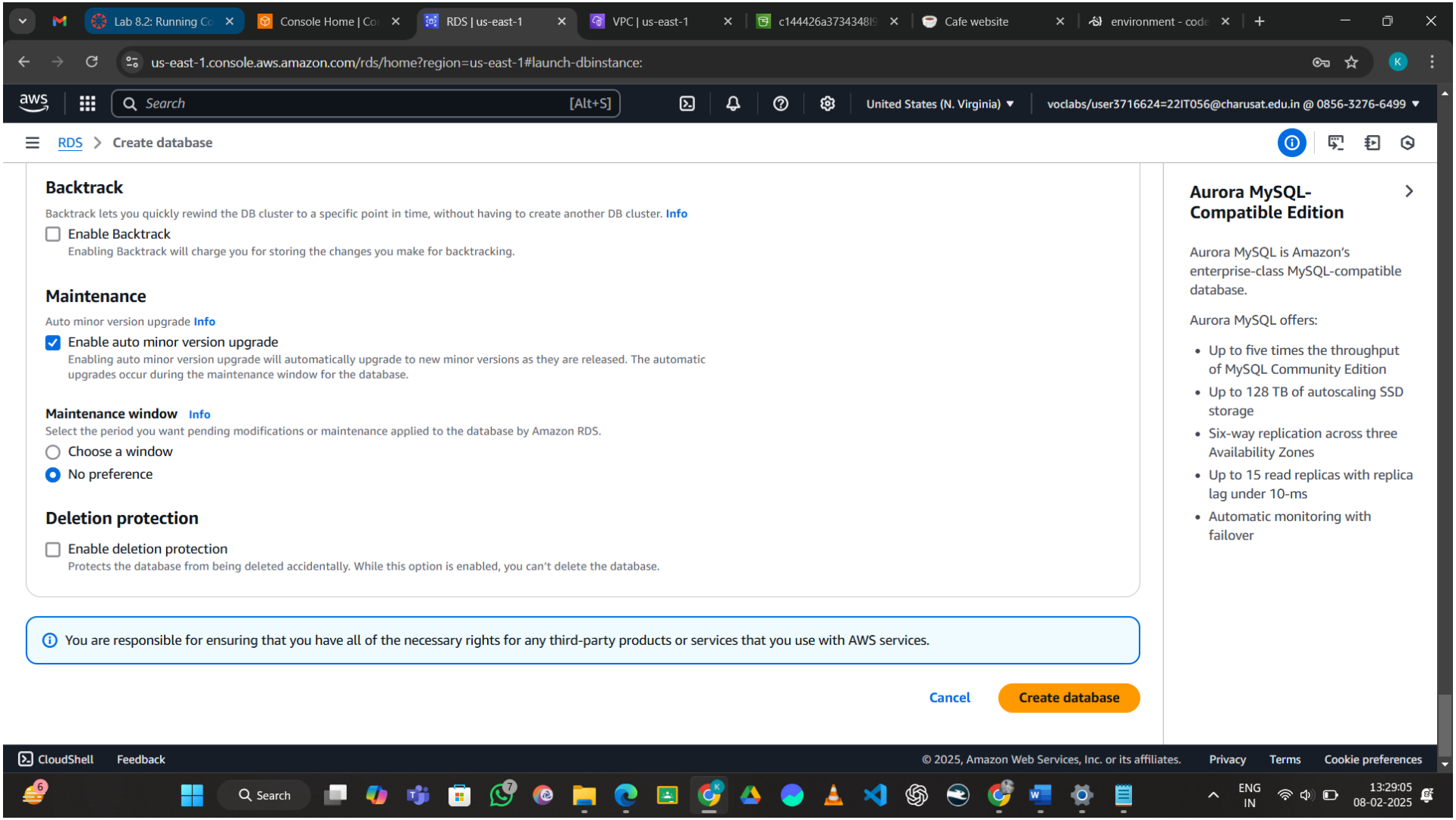

Figure 6: Creating a database

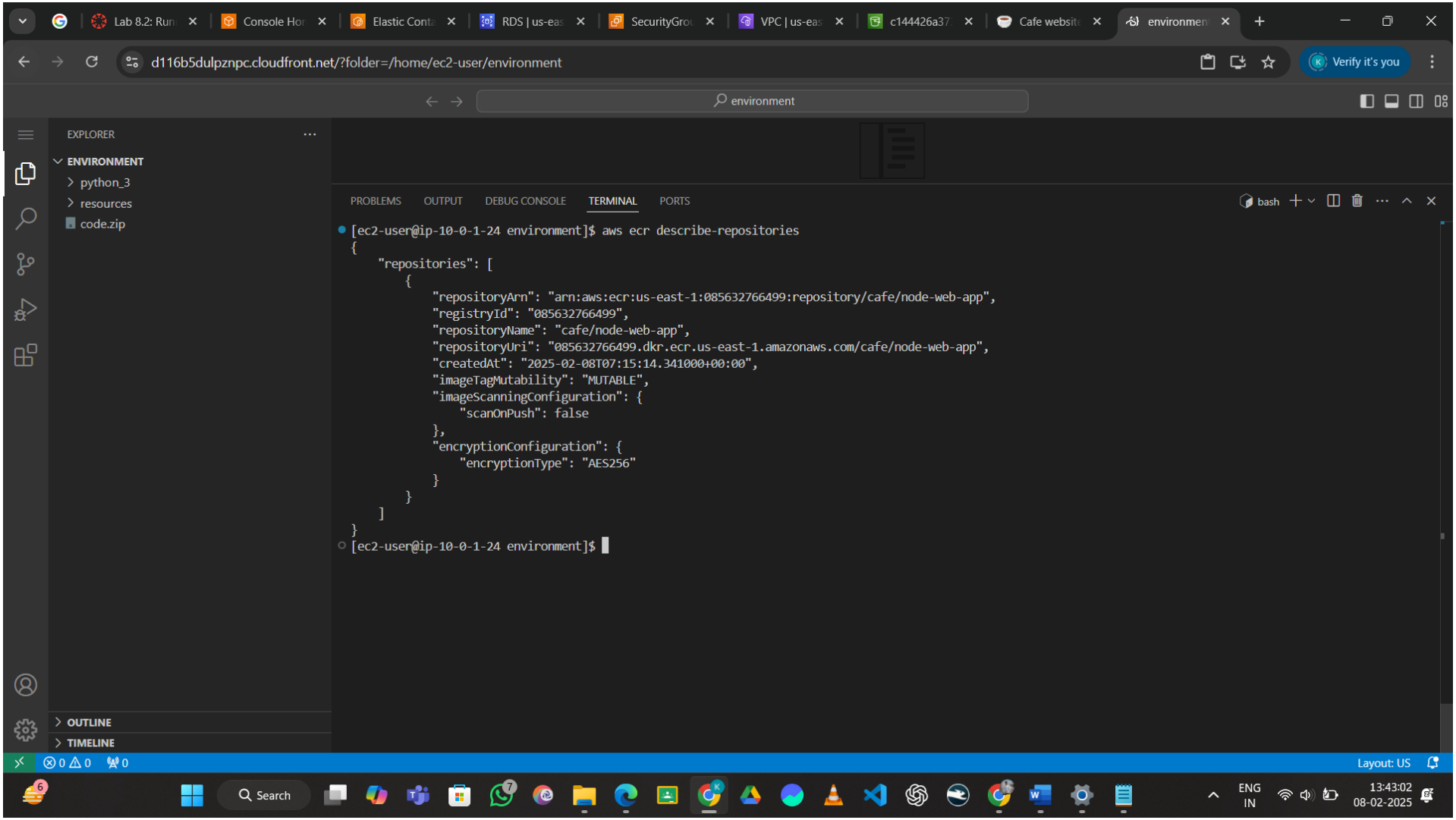

Figure 7: Describe a created repository

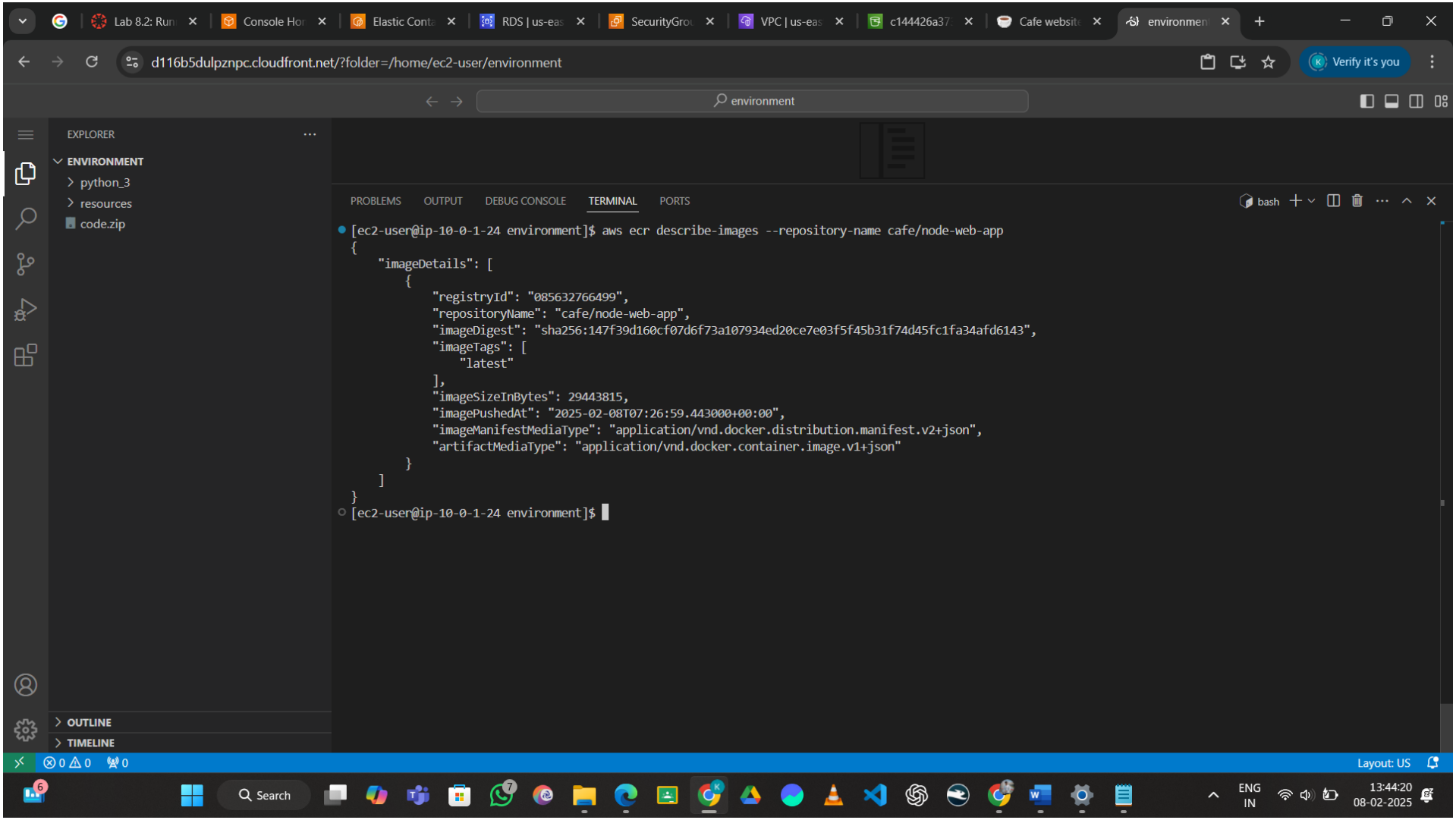

Figure 8: Describe a cafe web

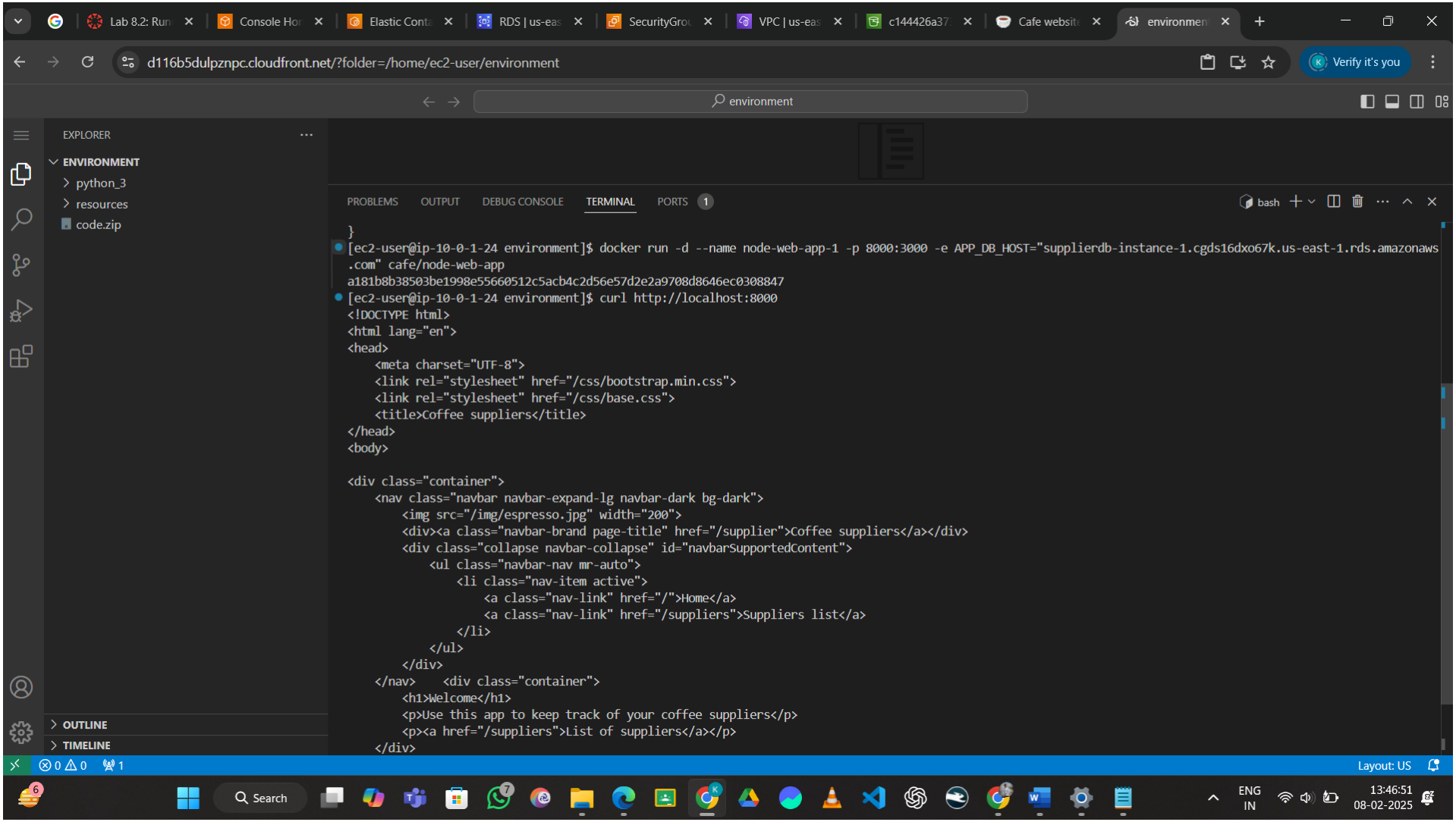

Figure 9: Create a docker with APP_DB_HOST

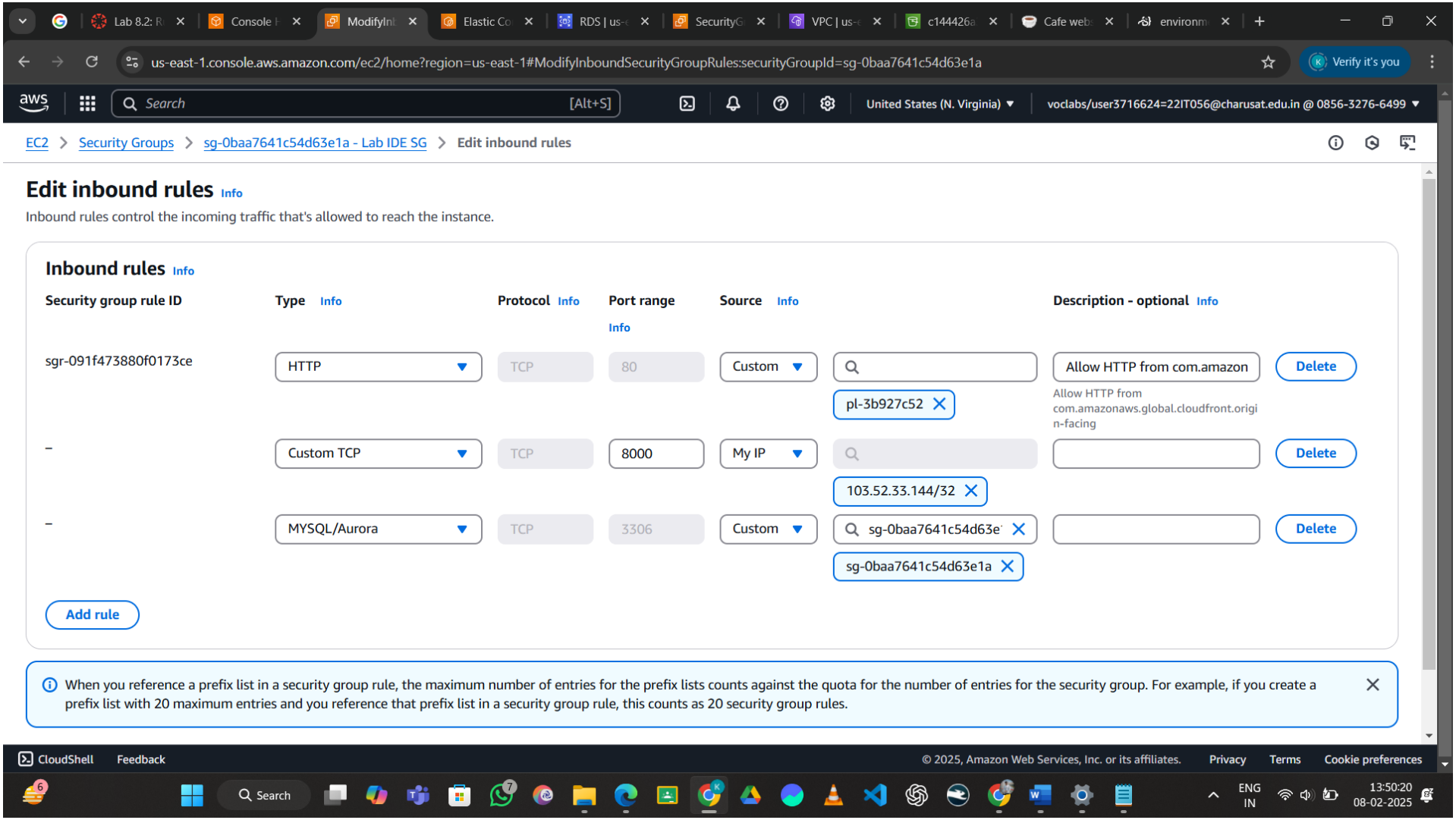

Figure 10: Edit inbound rules

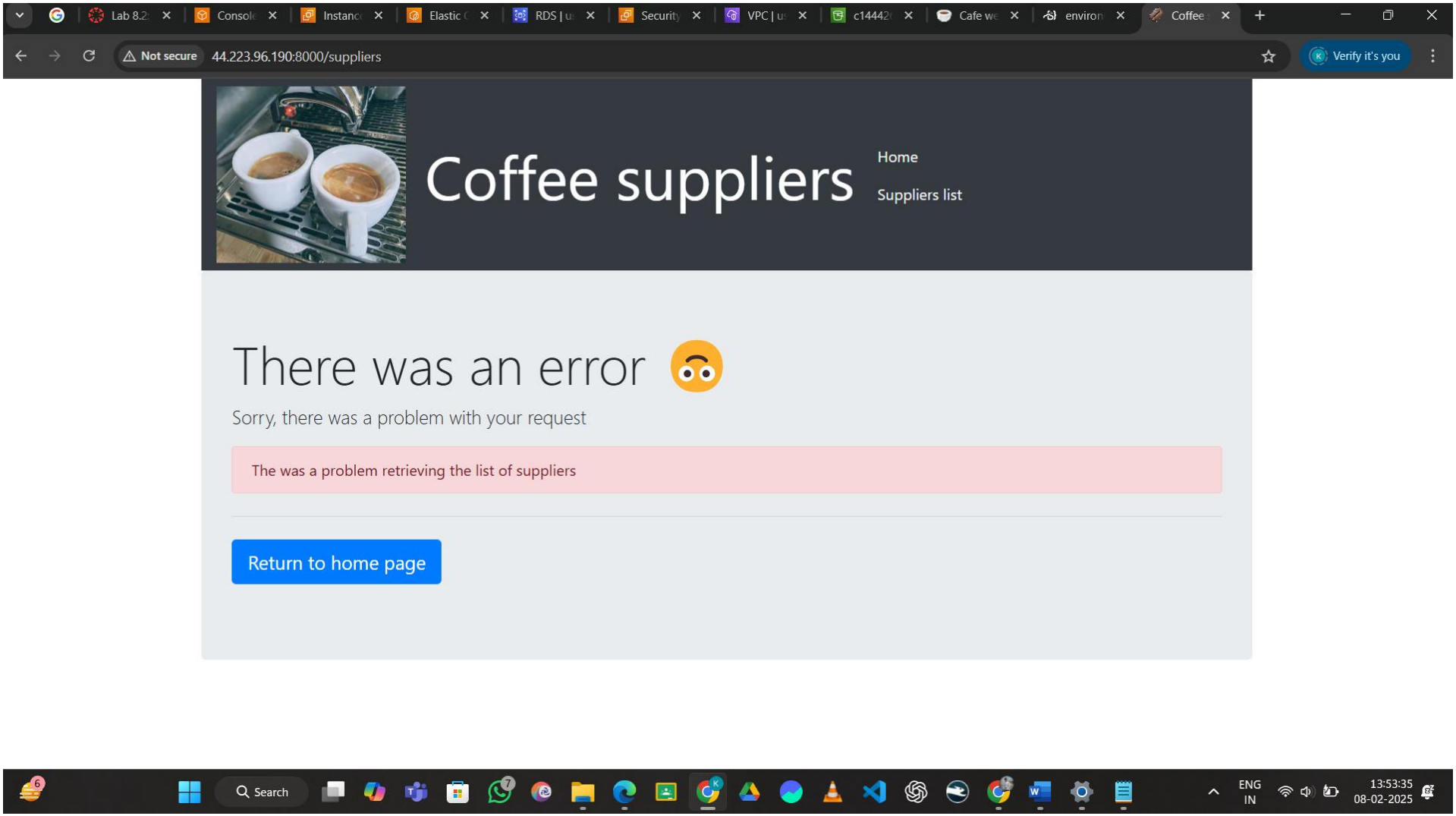

Figure 11: Cafe website without database

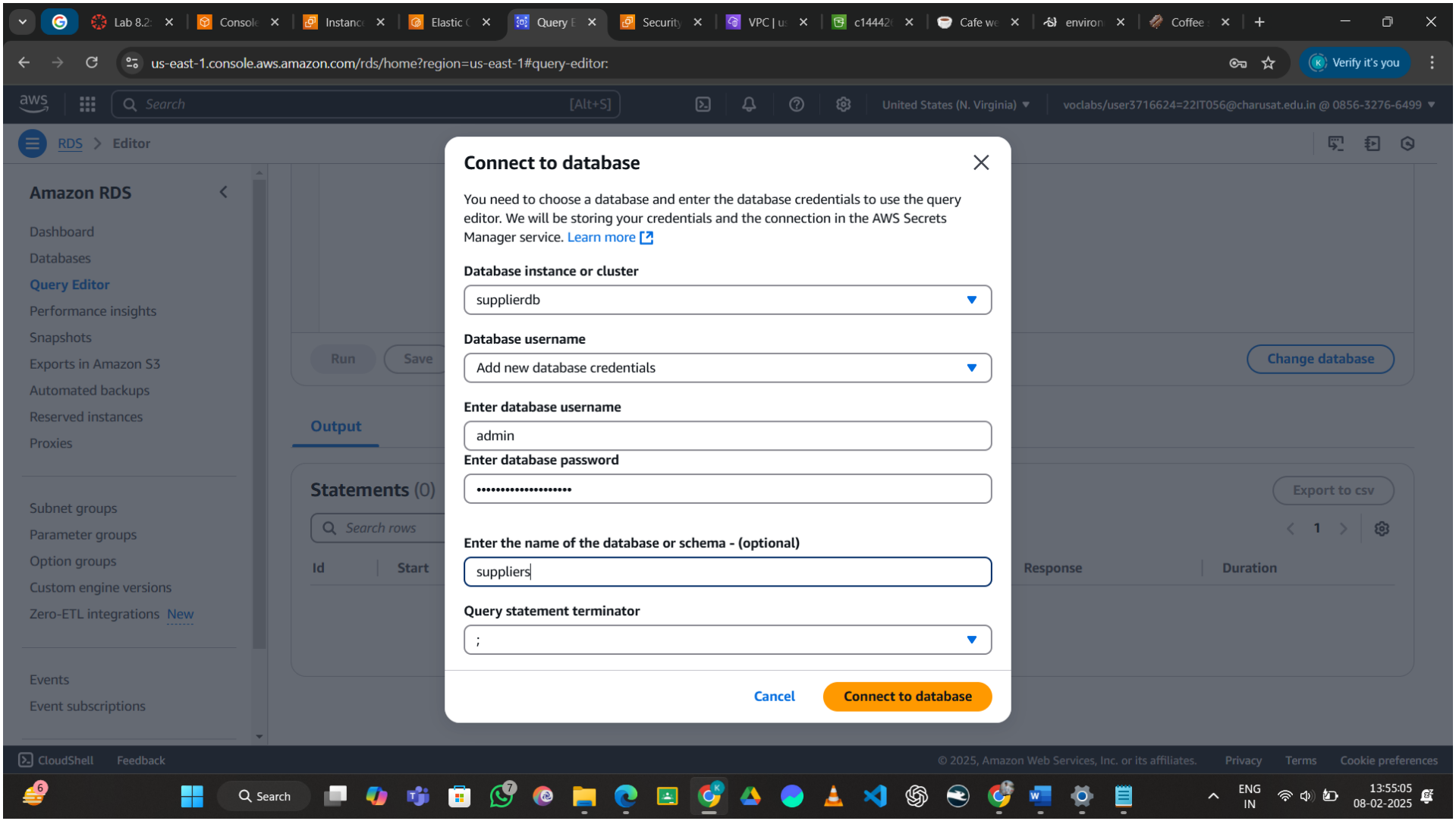

Figure 12: Use a created databasewith username and password

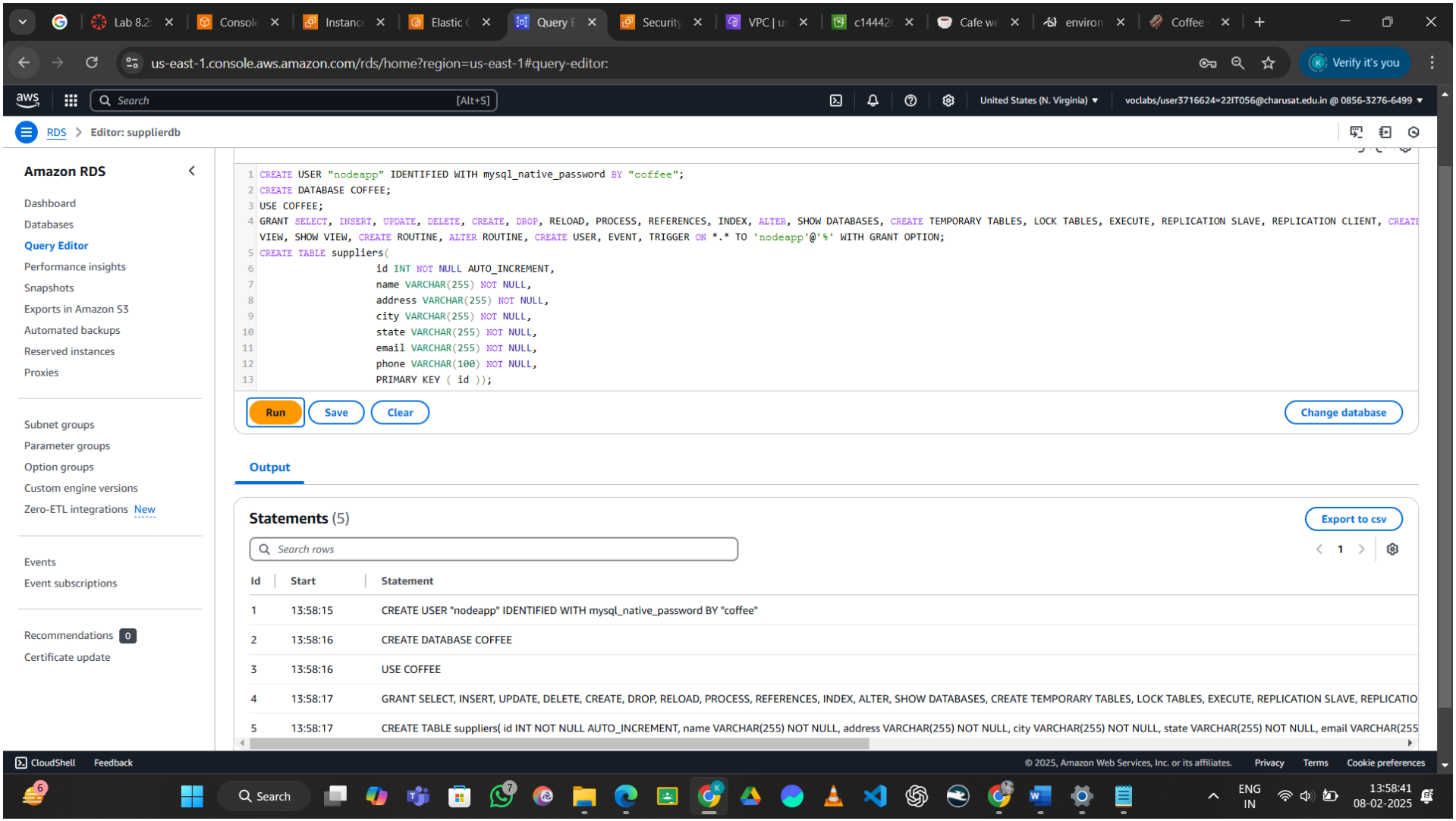

Figure 13: Create a database and use it and create a table

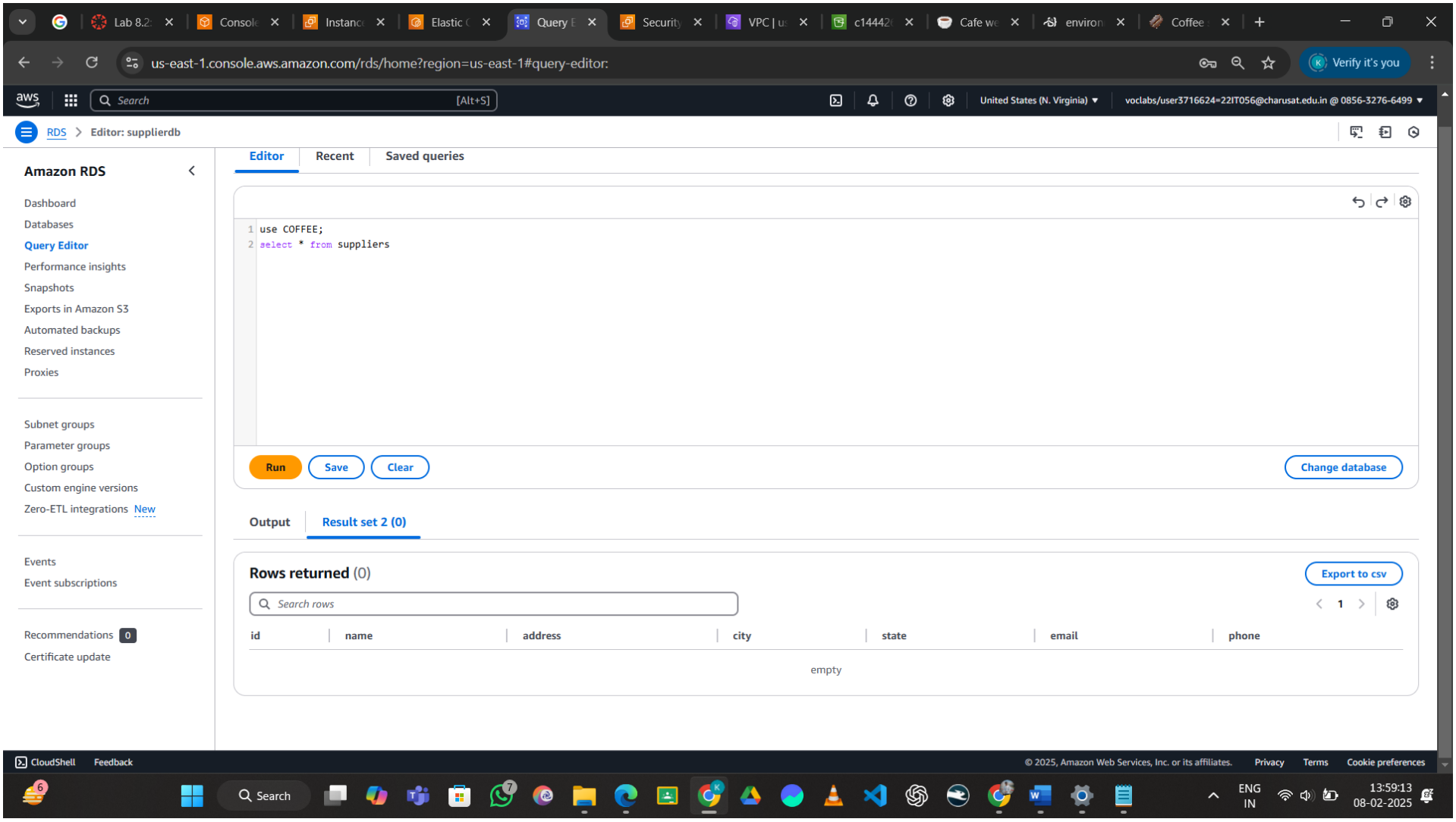

Figure 14: Use a coffee table and show a data

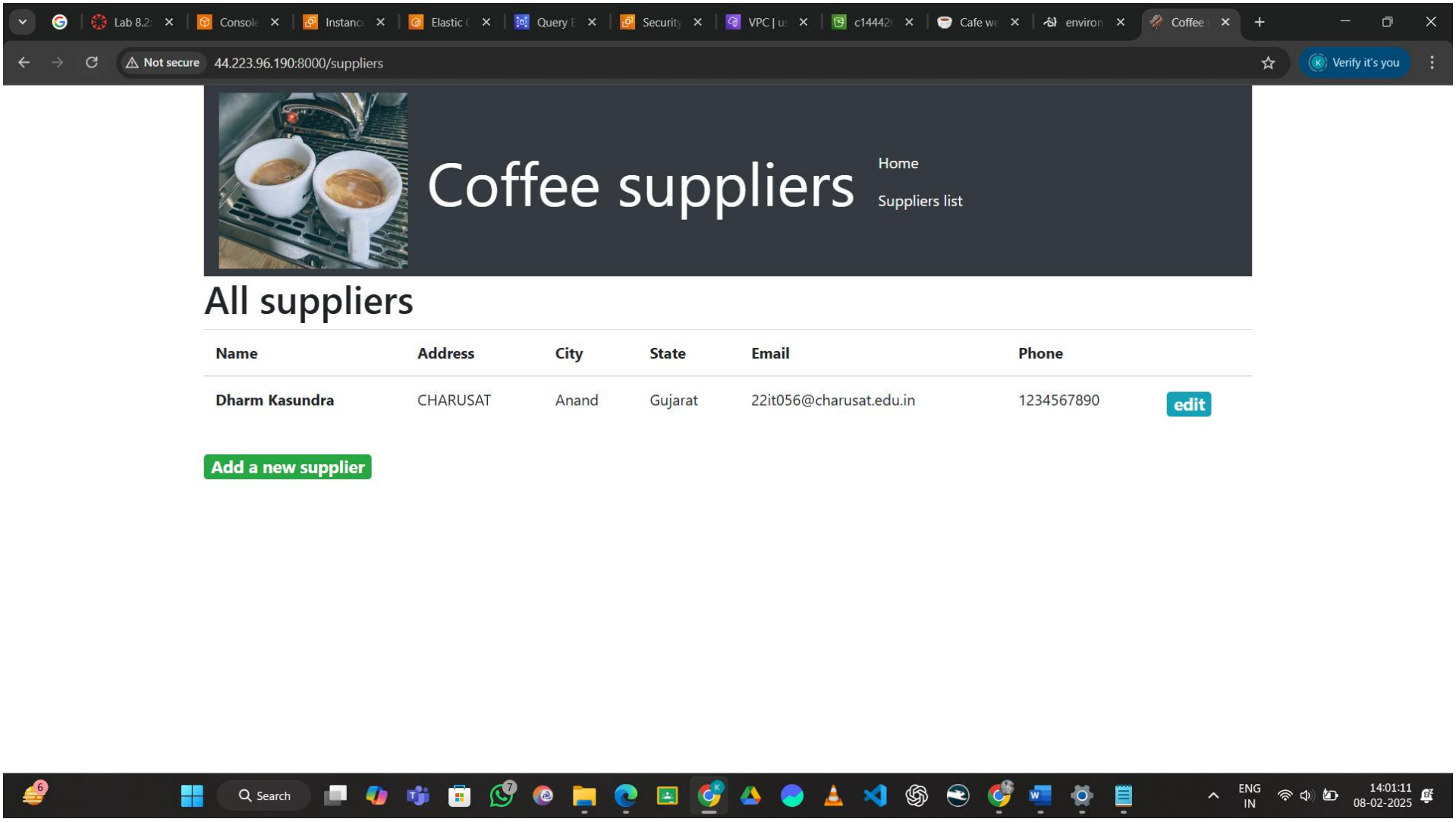

Figure 15: Add a new supplier using ui

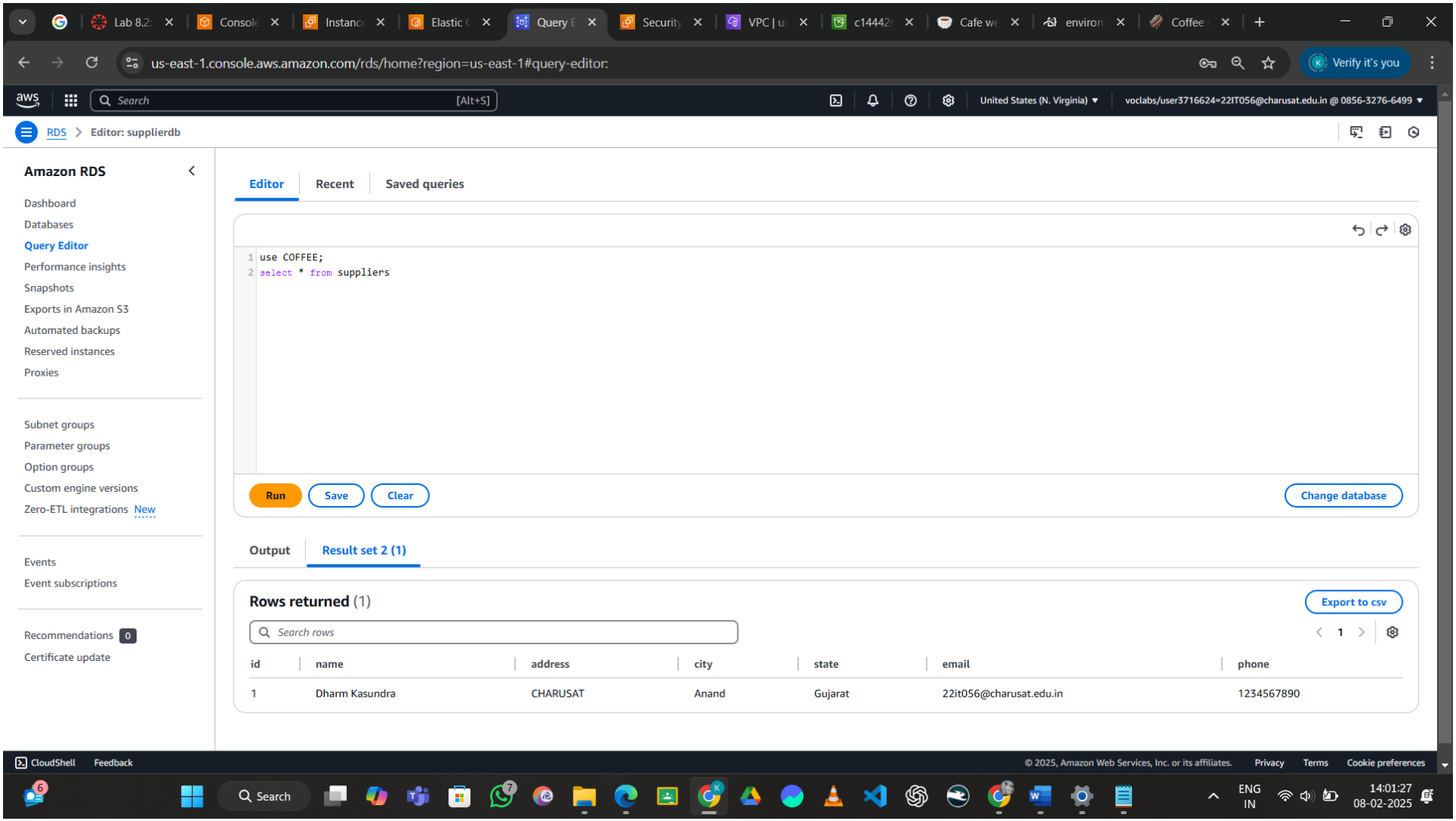

Figure 16: New supplier show in database

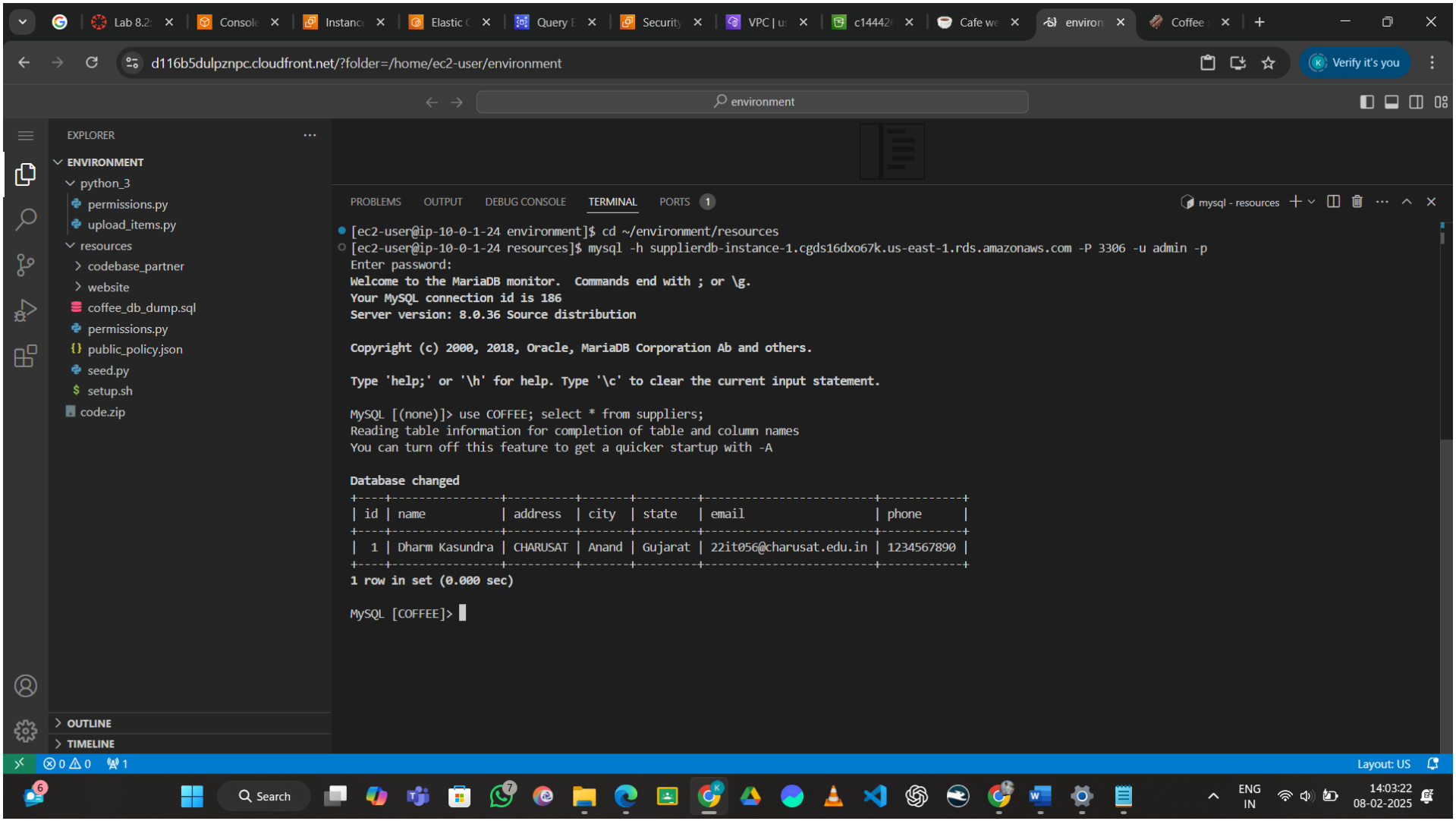

Figure 17: Use a database in editor

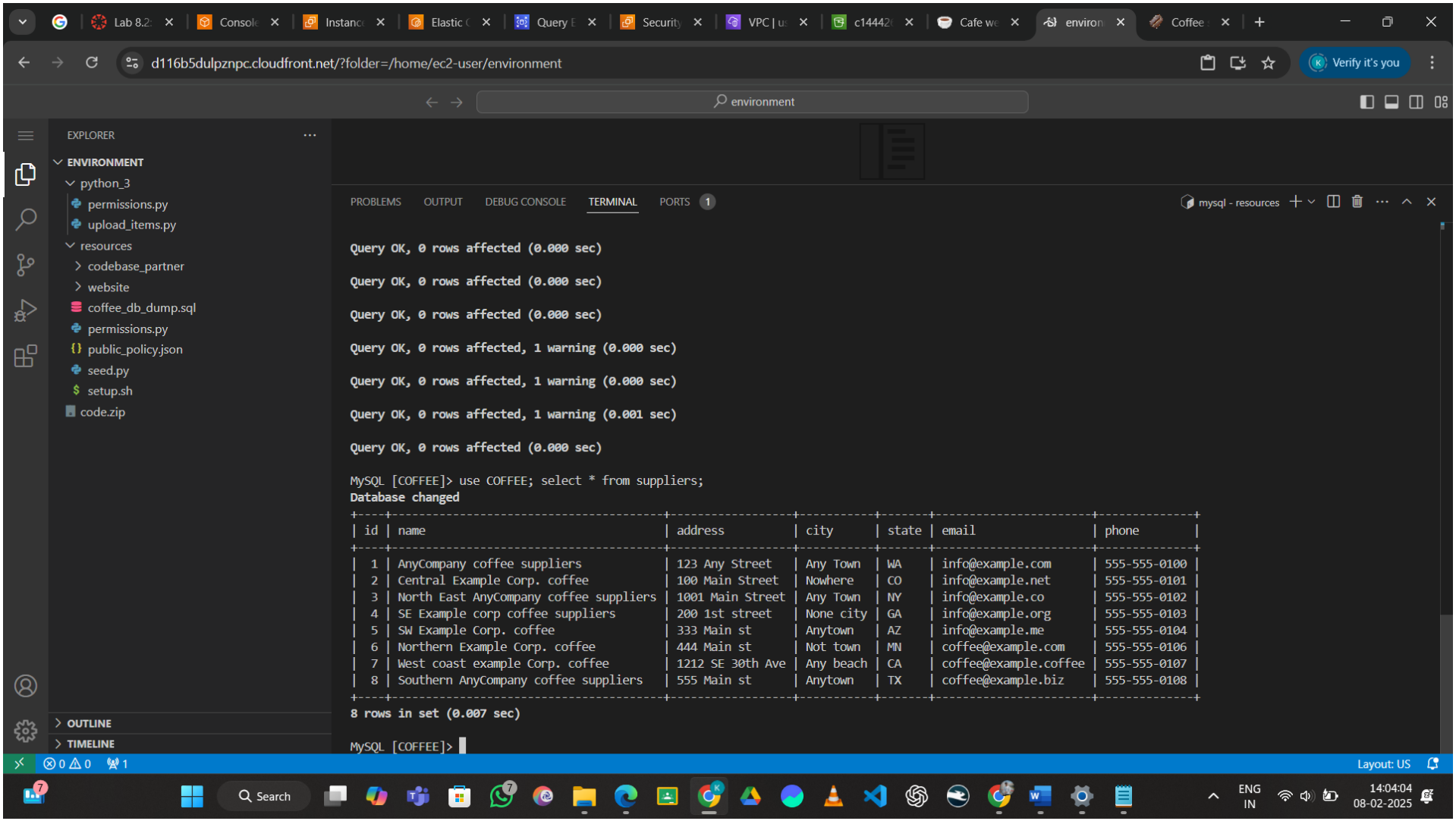

Figure 18: Fetch all suppliers

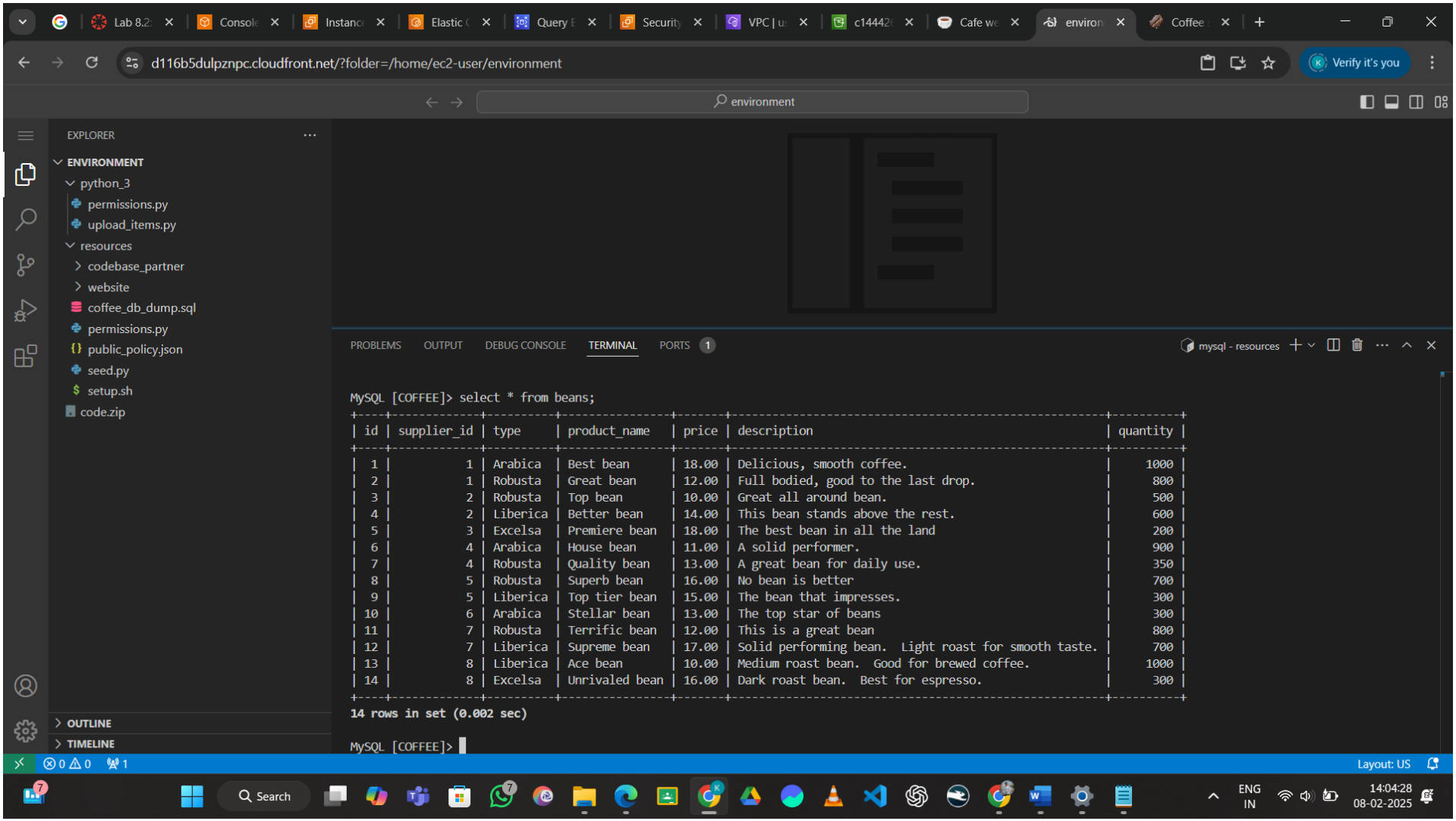

Figure 19: Fetch all beans

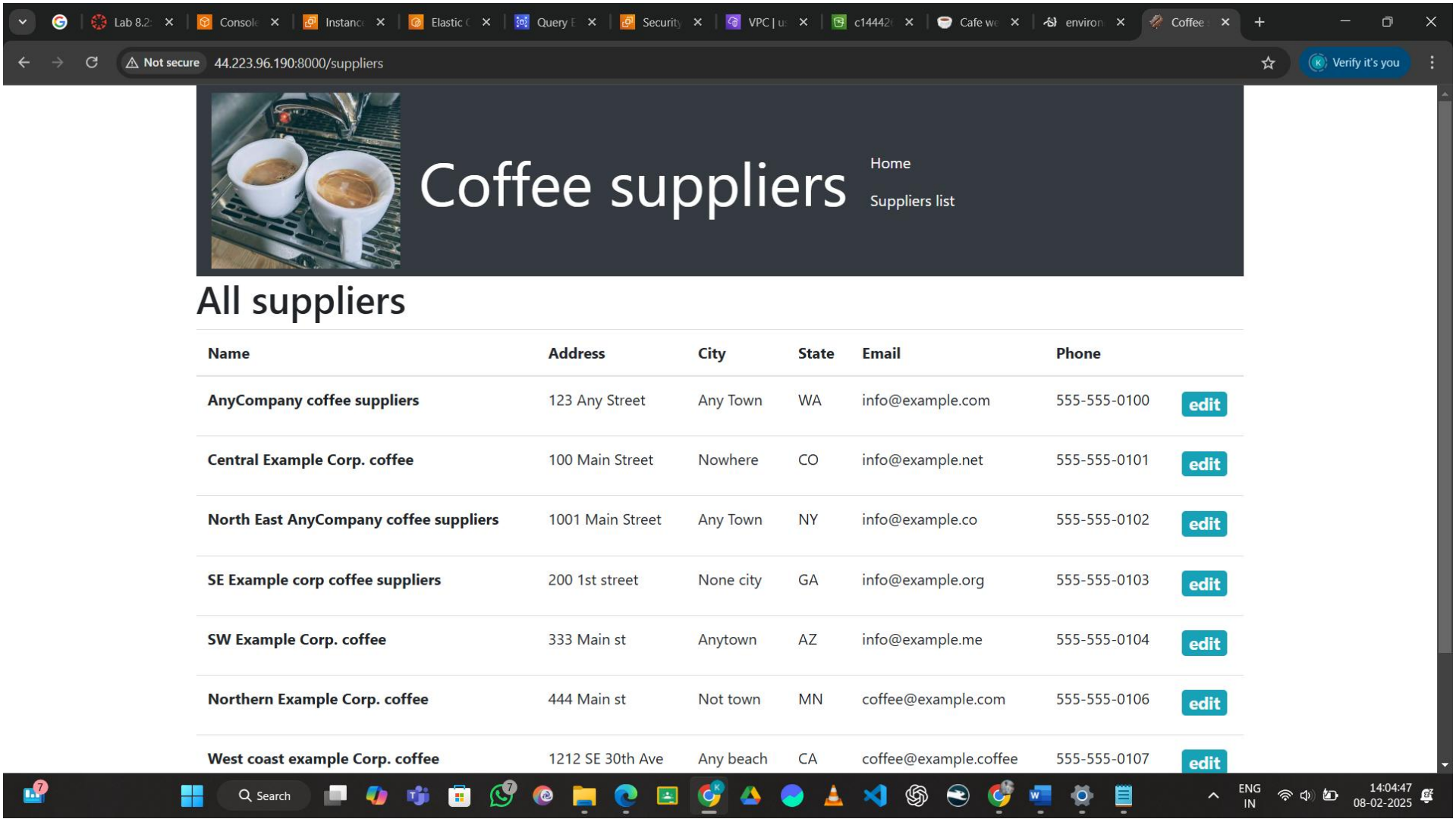

Figure 20: All supplier are show in websites

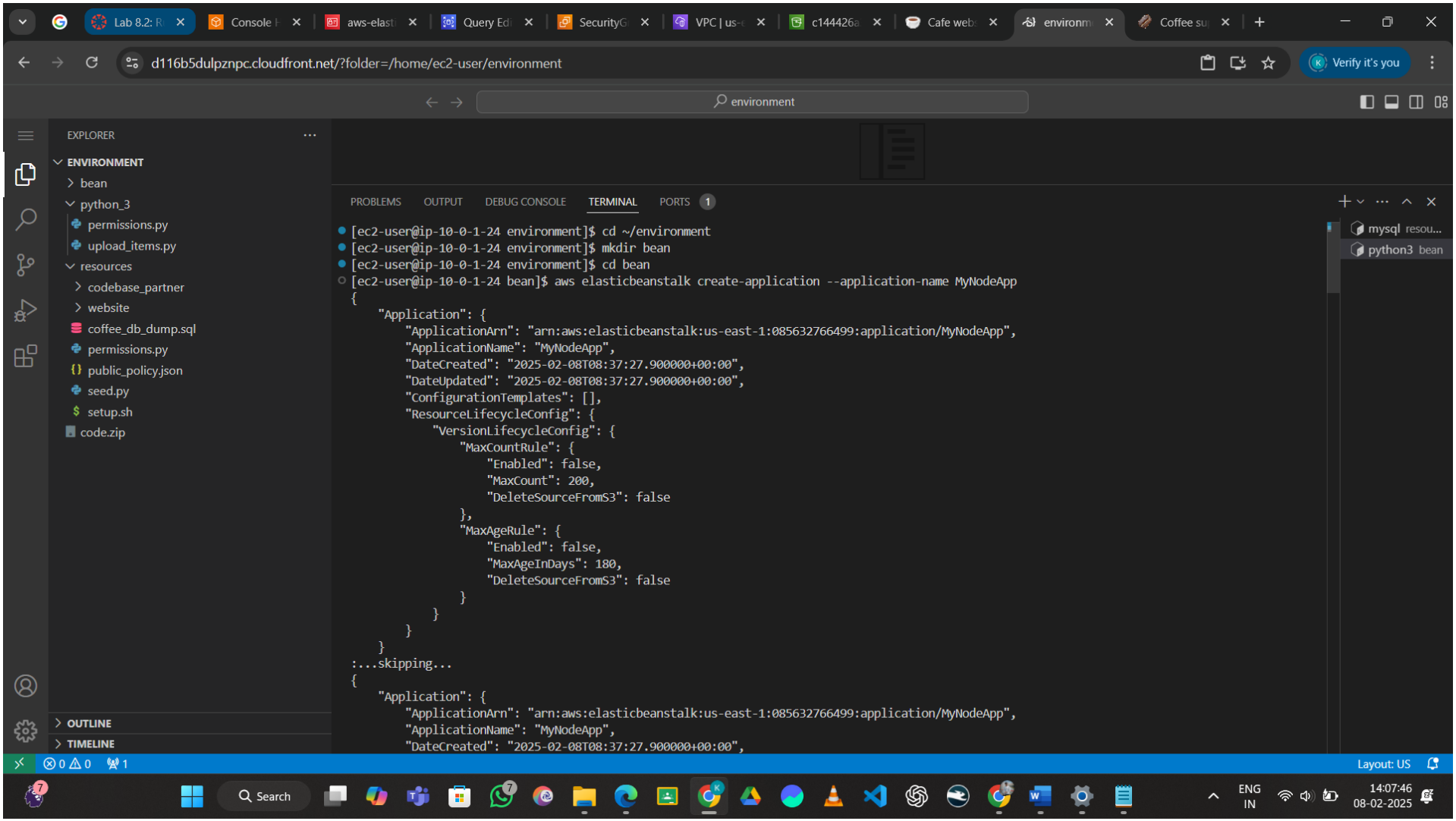

Figure 21: Show all information of application

LATEST APPLICATIONS:

1. **Amazon Aurora Serverless** – Auto-scaling, cost-effective, and highly available relational database.
2. **AWS Elastic Beanstalk** – Manages containerized web application deployment and scaling.
3. **Amazon API Gateway** – Routes API requests securely to Elastic Beanstalk.
4. **Amazon CloudWatch** – Monitors application performance and logs errors.
5. **AWS IAM** – Manages secure access to AWS resources.
6. **AWS Auto Scaling** – Dynamically adjusts resources based on traffic.
7. **Amazon S3 (Optional)** – Stores static assets like images and backups.
8. **AWS Secrets Manager** – Secures database credentials and API keys.

LEARNING OUTCOME:

After completing this lab, I will gain hands-on experience in deploying a fully managed, scalable web application using AWS services. I will learn to set up Amazon Aurora Serverless as a relational database, ensuring automatic scaling and cost optimization. Additionally, I will deploy a containerized Node.js application on AWS Elastic Beanstalk, which automates infrastructure management, load balancing, and monitoring.

Furthermore, I will understand how Amazon API Gateway securely routes external API requests to the application, improving performance and security. I will also explore AWS CloudWatch for monitoring logs and performance metrics. This lab enhances my skills in cloud-based deployment, database scaling, and managed services, equipping me with industry-relevant knowledge for building modern, cloud-native applications.

REFERENCE:

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