

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.

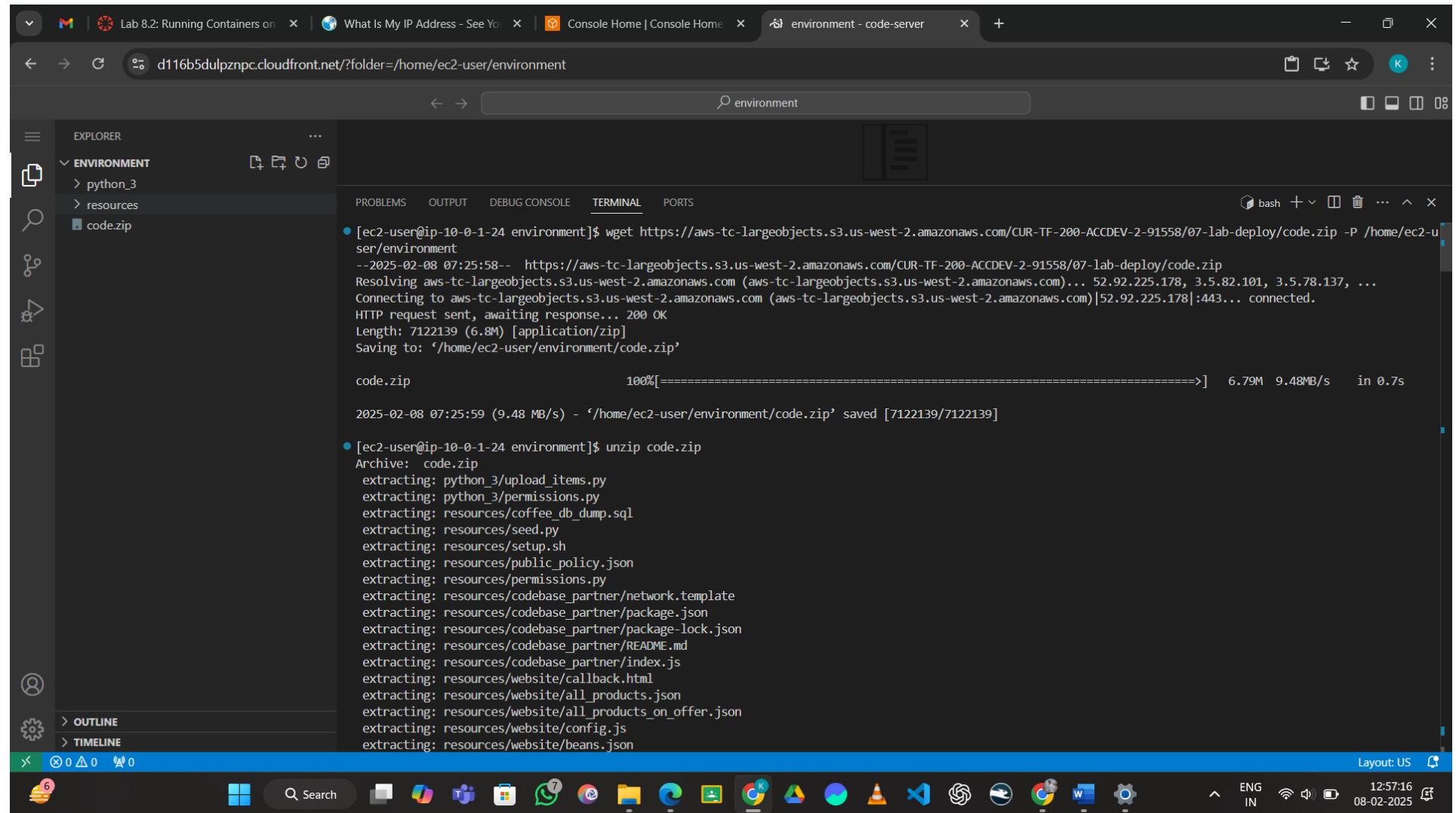
A key challenge in scaling containerized relational databases is maintaining **stateful connections** and **persistent storage**, which are difficult to achieve with ephemeral containers. To solve this, the lab replaces the **container-based MySQL database** with Aurora Serverless, which offers automated 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:



The screenshot shows a terminal window within a code editor interface. The terminal is running on an EC2 instance with IP address 10-0-1-24. It displays the following commands and output:

```
[ec2-user@ip-10-0-1-24 environment]$ wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/07-lab-deploy/code.zip -P /home/ec2-user/environment  
--2025-02-08 07:25:58-- https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/07-lab-deploy/code.zip  
Resolving aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)... 52.92.225.178, 3.5.82.101, 3.5.78.137, ...  
Connecting to aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)|52.92.225.178|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 7122139 (6.8M) [application/zip]  
Saving to: '/home/ec2-user/environment/code.zip'  
  
code.zip 100%[=====] 6.79M 9.48MB/s in 0.7s  
  
2025-02-08 07:25:59 (9.48 MB/s) - '/home/ec2-user/environment/code.zip' saved [7122139/7122139]  
  
[ec2-user@ip-10-0-1-24 environment]$ unzip code.zip  
Archive: code.zip  
extracting: python_3/upload_items.py  
extracting: python_3/permissions.py  
extracting: resources/coffee_db_dump.sql  
extracting: resources/seed.py  
extracting: resources/setup.sh  
extracting: resources/public_policy.json  
extracting: resources/permissions.py  
extracting: resources/codebase_partner/network.template  
extracting: resources/codebase_partner/package.json  
extracting: resources/codebase_partner/package-lock.json  
extracting: resources/codebase_partner/README.md  
extracting: resources/codebase_partner/index.js  
extracting: resources/website/callback.html  
extracting: resources/website/all_products.json  
extracting: resources/website/all_products_on_offer.json  
extracting: resources/website/config.js  
extracting: resources/website/beans.json
```

Figure 1: Download and extract file

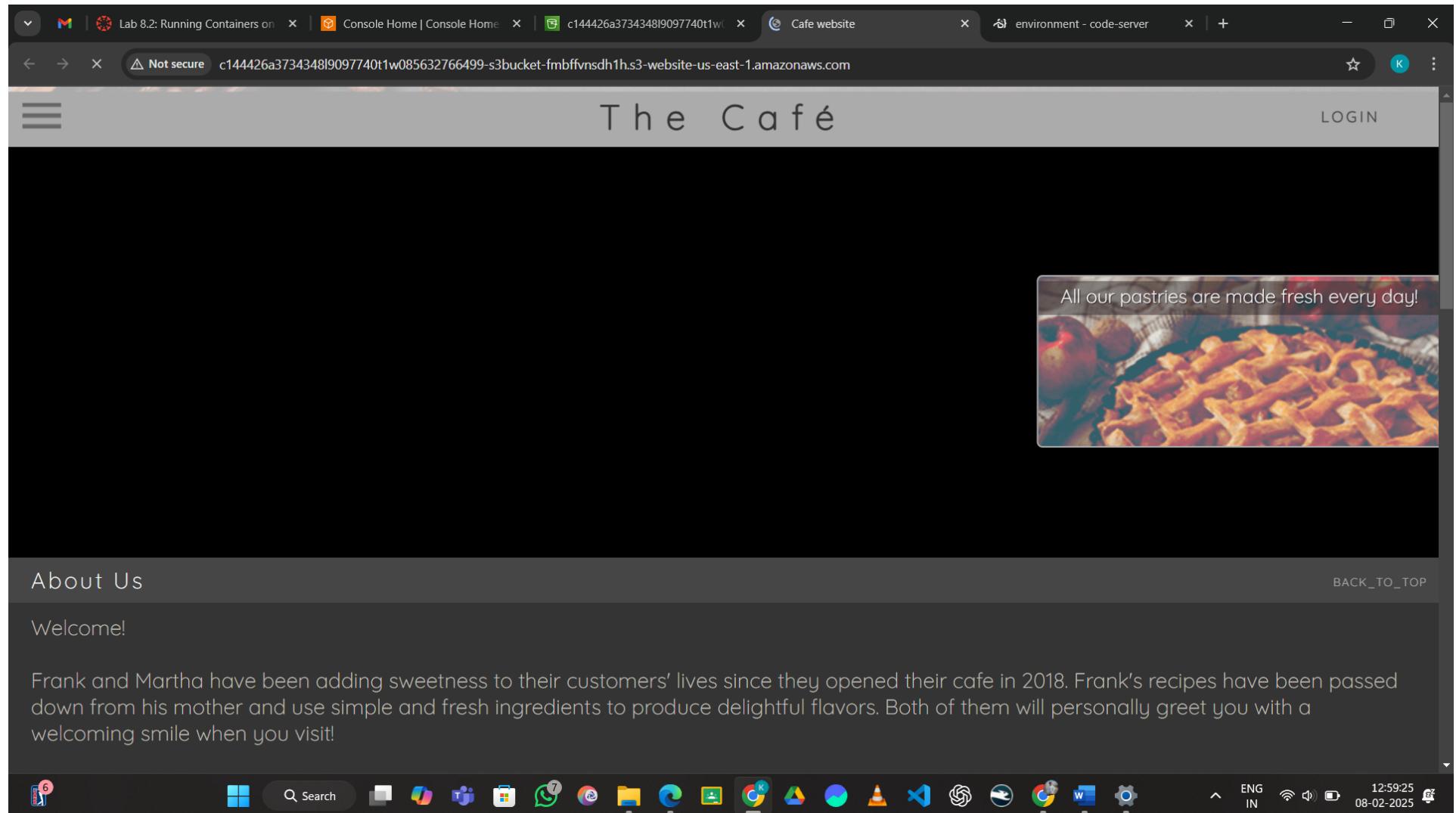


Figure 2: Load a cafe website

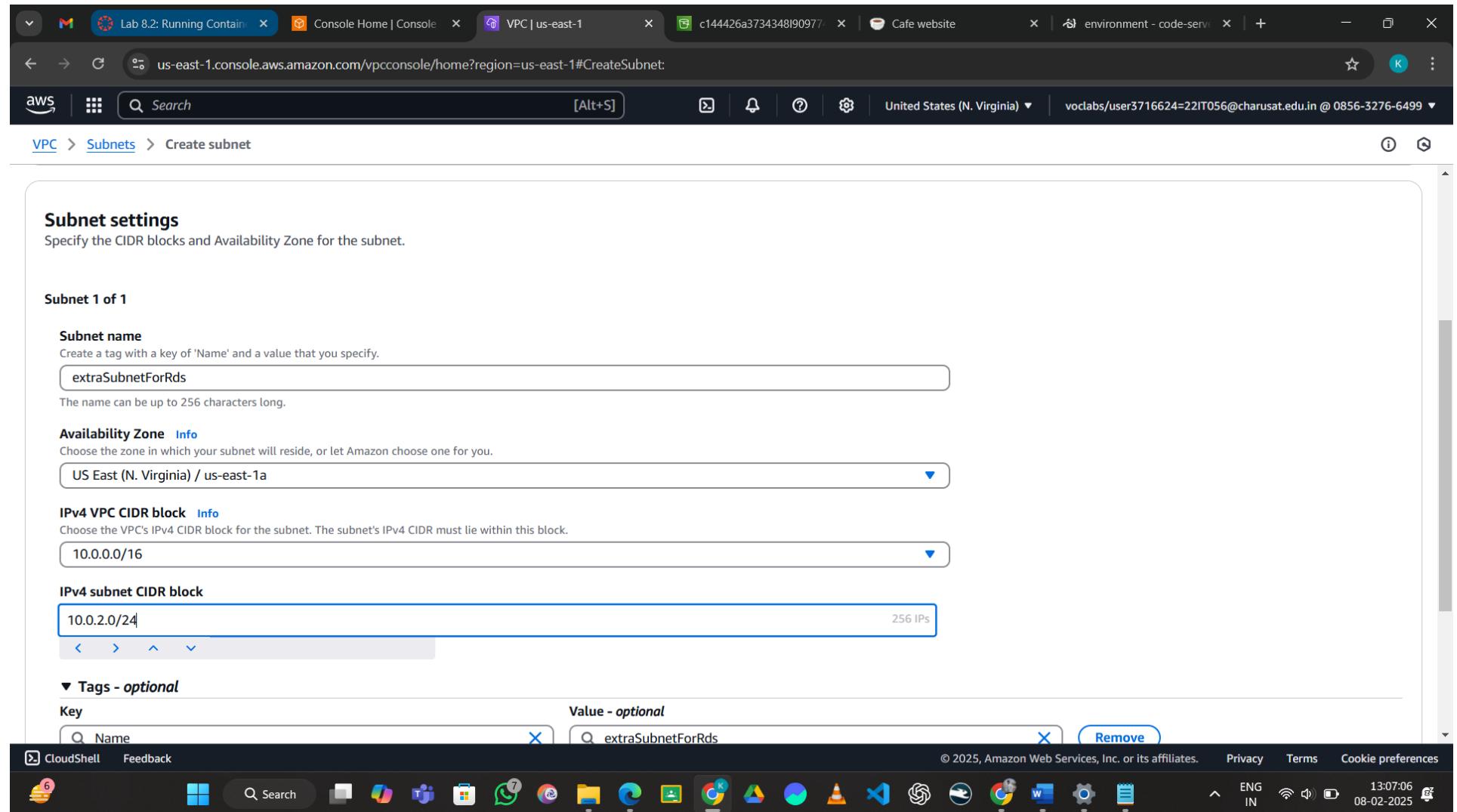


Figure 3: Creating a subnet

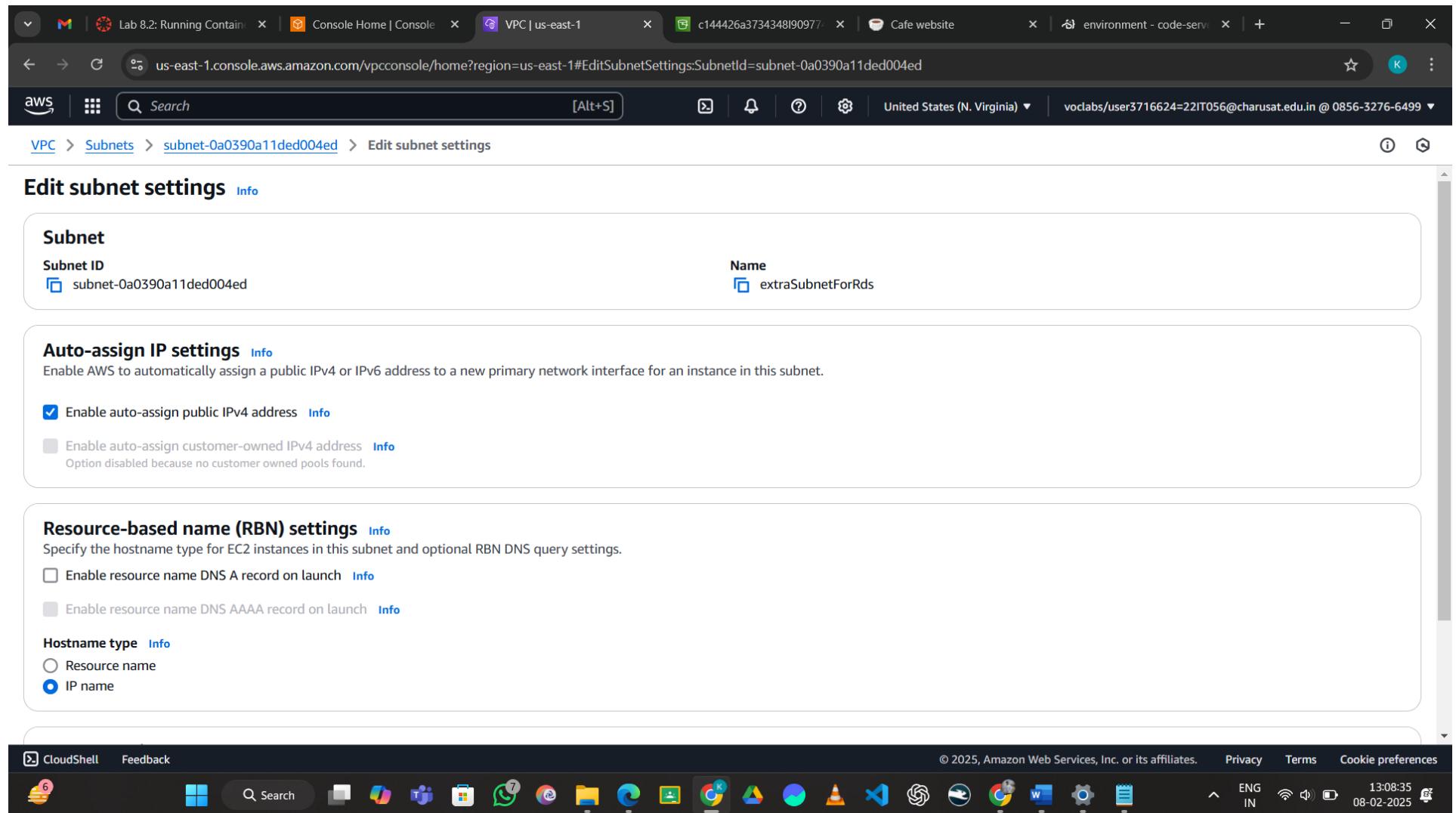


Figure 4: Enable auto ipv4 address in creating a subnet

The screenshot shows the AWS VPC Subnets page. A green success message at the top states: "Subnet (subnet-010ea7c74c2797c6d) has been successfully associated with route table (rtb-0799b6bb905ac92a6)." The main table lists one subnet:

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
extraSubnetForRds	subnet-010ea7c74c2797c6d	Available	vpc-0b717faded93b7380 IDE...	Off	10.0.2.0/24

Below the table, the subnet details are shown for "subnet-010ea7c74c2797c6d / extraSubnetForRds". The "Route table" tab is selected, showing two routes:

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-043d5bd8dc8d48fc1

At the bottom of the page, there are standard AWS navigation icons and a footer with copyright information.

Figure 5: Verify edited information

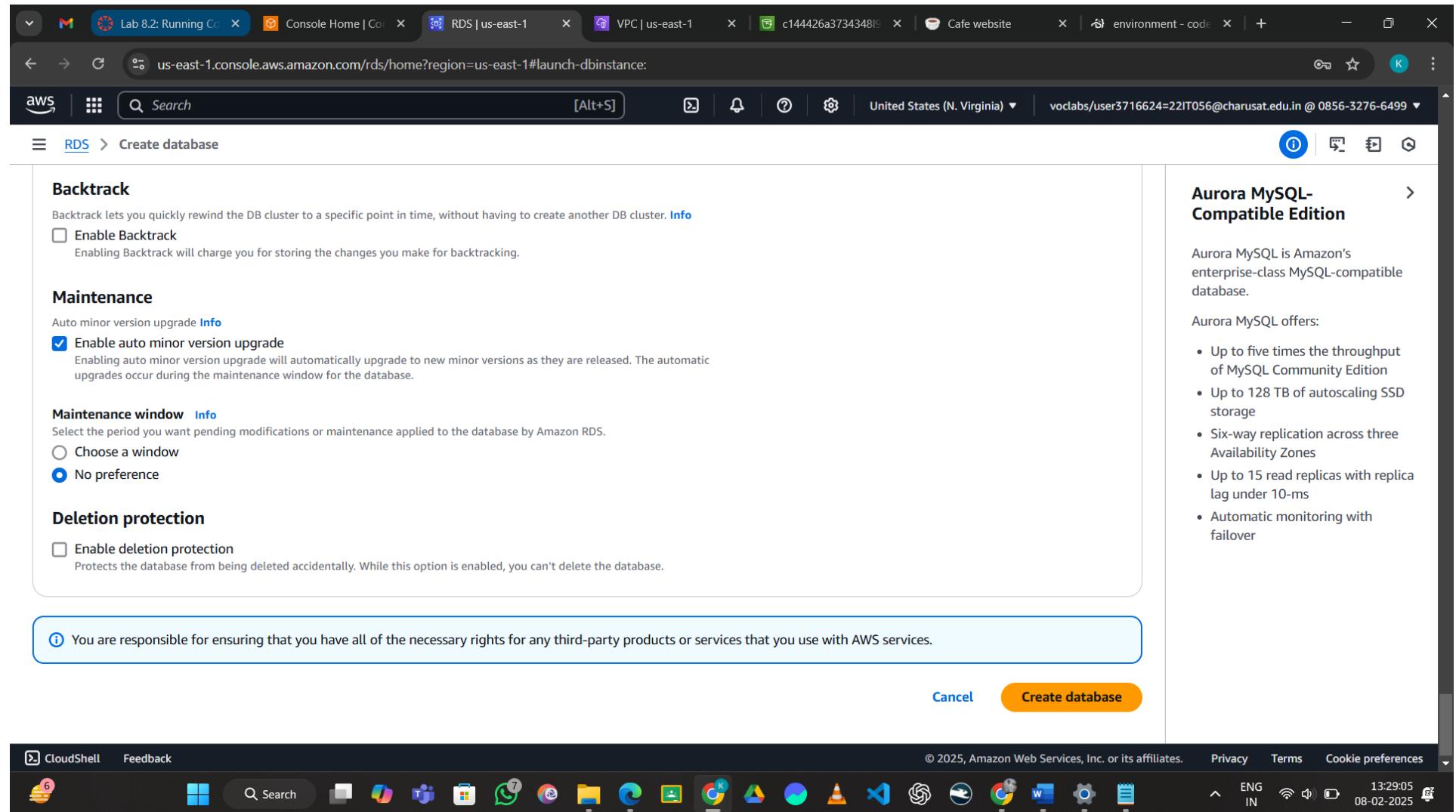


Figure 6: Creating a database

The screenshot shows a terminal window within a dark-themed IDE interface. The terminal tab is active, displaying the command `aws ecr describe-repositories` and its JSON output. The output details a single repository named "cafe/node-web-app" with ARN `arn:aws:ecr:us-east-1:085632766499:repository/cafe/node-web-app`. The repository was created on 2025-02-08T07:15:14.341000+00:00. It uses AES256 encryption and has a mutable image tag. The repository URI is `085632766499.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app`. The terminal prompt ends with `[ec2-user@ip-10-0-1-24 environment]$`.

Figure 7: Describe a created repository

The screenshot shows a terminal window in the VS Code interface. The title bar indicates the URL is `d116b5dulpznp.cdnfront.net/?folder=/home/ec2-user/environment`. The terminal tab is active, showing the command `aws ecr describe-images --repository-name cafe/node-web-app` and its JSON output. The output details a single image entry:

```
[ec2-user@ip-10-0-1-24 environment]$ aws ecr describe-images --repository-name cafe/node-web-app
{
    "imageDetails": [
        {
            "registryId": "085632766499",
            "repositoryName": "cafe/node-web-app",
            "imageDigest": "sha256:147f39d160cf07d6f73a107934ed20ce7e03f5f45b31f74d45fc1fa34af6143",
            "imageTags": [
                "latest"
            ],
            "imageSizeInBytes": 29443815,
            "imagePushedAt": "2025-02-08T07:26:59.443000+00:00",
            "imageManifestMediaType": "application/vnd.docker.distribution.manifest.v2+json",
            "artifactMediaType": "application/vnd.docker.container.image.v1+json"
        }
    ]
}
[ec2-user@ip-10-0-1-24 environment]$
```

Figure 8: Describe a cafe web

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Windows operating system. The title bar indicates the window is titled "environment". The VS Code interface includes:

- EXPLORER** sidebar: Shows a folder named "ENVIRONMENT" containing "python_3", "resources", and "code.zip".
- TERMINAL** tab: Active, showing a command-line session on an AWS Lambda environment (ip-10-0-1-24). It runs a Docker container named "node-web-app-1" with port 8000 mapped to 3000, using the environment variable APP_DB_HOST to connect to an RDS database.
- OUTPUT** tab: Shows logs from the Docker container.
- DEBUG CONSOLE** tab: Available but not active.
- PROBLEMS** tab: Available but not active.
- PORTS** tab: Available but not active.
- CODE** area: Displays the HTML code for a "coffee suppliers" application, including a navigation bar with "Home" and "Suppliers list" links, and a welcome message.
- STATUS BAR**: Shows the date (08-02-2025), time (13:46:51), and system icons.

Figure 9: Create a docker with APP_DB_HOST

The screenshot shows the 'Edit inbound rules' page for a security group. The page has a header with tabs like 'ModifyInb', 'Elastic Co...', 'RDS | us...', 'SecurityG...', 'VPC | us...', 'c144426a...', 'Cafe web...', and 'environment...'. The URL is 'us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-0baa7641c54d63e1a'. The top navigation bar includes 'aws', a search bar, and account information 'United States (N. Virginia) voclabs/user3716624=22IT056@charusat.edu.in @ 0856-3276-6499'. Below the header, the breadcrumb trail is 'EC2 > Security Groups > sg-0baa7641c54d63e1a - Lab IDE SG > Edit inbound rules'. The main content area is titled 'Edit inbound rules' with a 'Info' link. A sub-header says 'Inbound rules' with its own 'Info' link. The table lists three rules:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional	Action
sgr-091f473880f0173ce	HTTP	TCP	80	Custom	Allow HTTP from com.amazon pl-3b927c52	Delete
-	Custom TCP	TCP	8000	My IP	Allow HTTP from com.amazonaws.global.cloudfront.origi n-facing 103.52.33.144/32	Delete
-	MySQL/Aurora	TCP	3306	Custom	Allow MySQL/Aurora sg-0baa7641c54d63e1a	Delete

At the bottom left is a blue 'Add rule' button. A callout box with an info icon provides a note: 'When you reference a prefix list in a security group rule, the maximum number of entries for the prefix lists counts against the quota for the number of entries for the security group. For example, if you create a prefix list with 20 maximum entries and you reference that prefix list in a security group rule, this counts as 20 security group rules.' At the very bottom, there's a dark footer with icons for CloudShell, Feedback, Search, and various AWS services, along with copyright information '© 2025, Amazon Web Services, Inc. or its affiliates.' and a timestamp '13:50:20 08-02-2025'.

Figure 10: Edit inbound rules

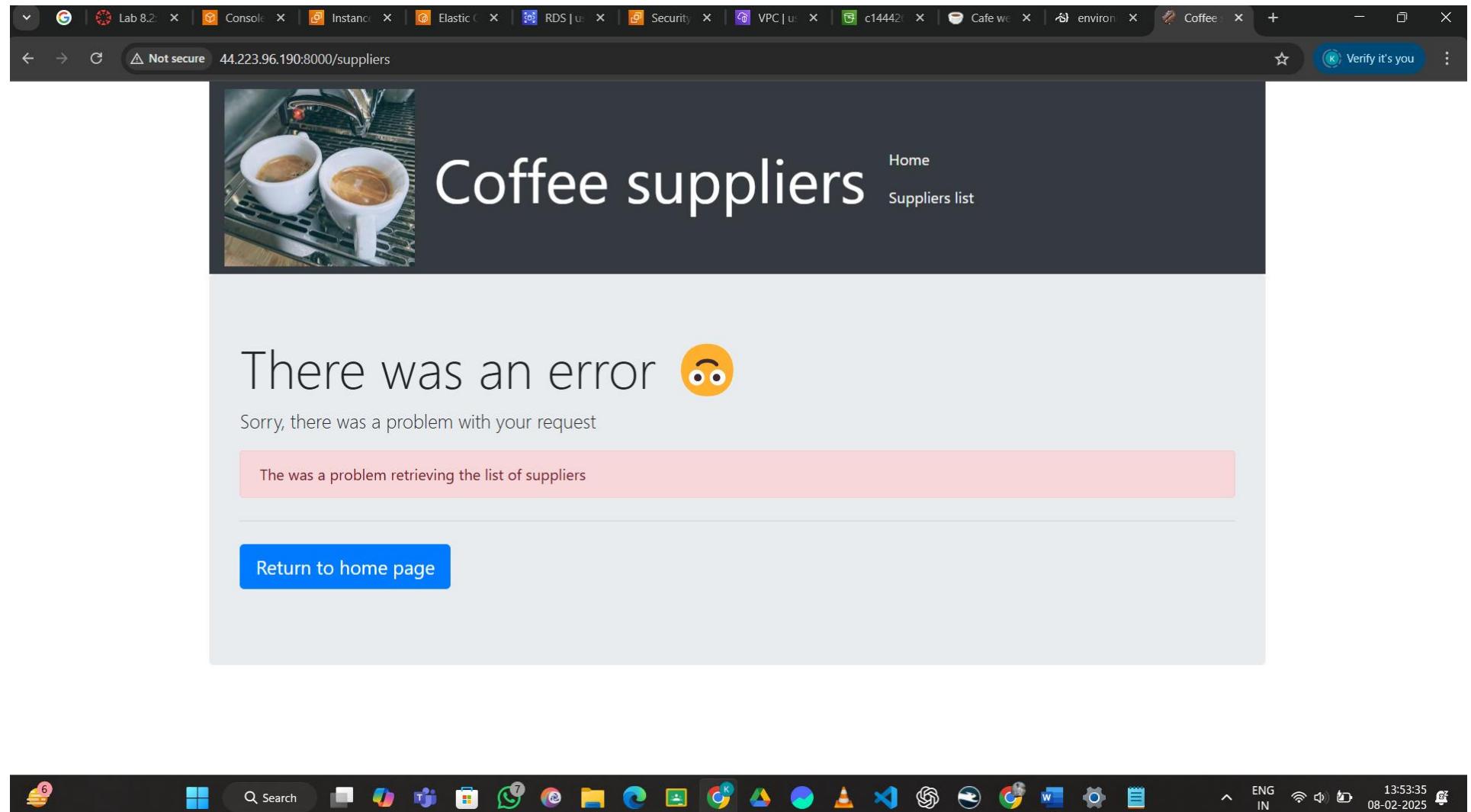


Figure 11: Cafe website without database

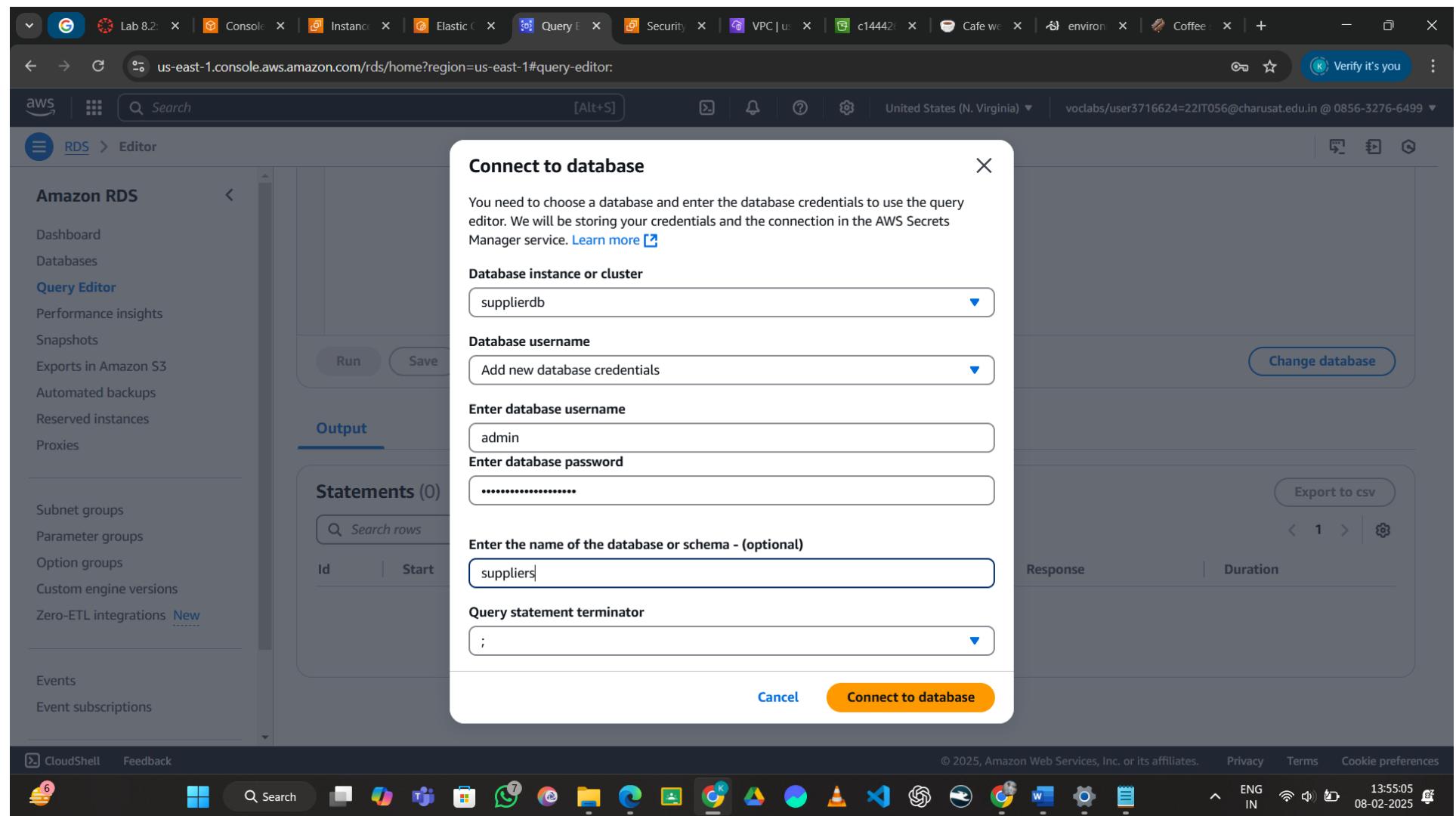


Figure 12: Use a created database with username and password

The screenshot shows the AWS RDS Query Editor interface. On the left, a sidebar menu for 'Amazon RDS' includes 'Query Editor' which is currently selected. The main area displays a block of MySQL code:

```
1 CREATE USER "nodeapp" IDENTIFIED WITH mysql_native_password BY "coffee";
2 CREATE DATABASE COFFEE;
3 USE COFFEE;
4 GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER ON *.* TO 'nodeapp'@'%' WITH GRANT OPTION;
5 CREATE TABLE suppliers(
6     id INT NOT NULL AUTO_INCREMENT,
7     name VARCHAR(255) NOT NULL,
8     address VARCHAR(255) NOT NULL,
9     city VARCHAR(255) NOT NULL,
10    state VARCHAR(255) NOT NULL,
11    email VARCHAR(255) NOT NULL,
12    phone VARCHAR(100) NOT NULL,
13    PRIMARY KEY ( id ));
```

Below the code are three buttons: 'Run' (highlighted in orange), 'Save', and 'Clear'. To the right is a 'Change database' link. The 'Output' section shows the results of the five statements run:

Id	Start	Statement
1	13:58:15	CREATE USER "nodeapp" IDENTIFIED WITH mysql_native_password BY "coffee"
2	13:58:16	CREATE DATABASE COFFEE
3	13:58:16	USE COFFEE
4	13:58:17	GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER ON *.* TO 'nodeapp'@'%' WITH GRANT OPTION;
5	13:58:17	CREATE TABLE suppliers(id INT NOT NULL AUTO_INCREMENT, name VARCHAR(255) NOT NULL, address VARCHAR(255) NOT NULL, city VARCHAR(255) NOT NULL, state VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL, phone VARCHAR(100) NOT NULL, PRIMARY KEY (id));

At the bottom, there's a toolbar with icons for CloudShell, Feedback, Search, and various application icons. The status bar at the bottom right shows the date and time: 13:58:41, 08-02-2025.

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

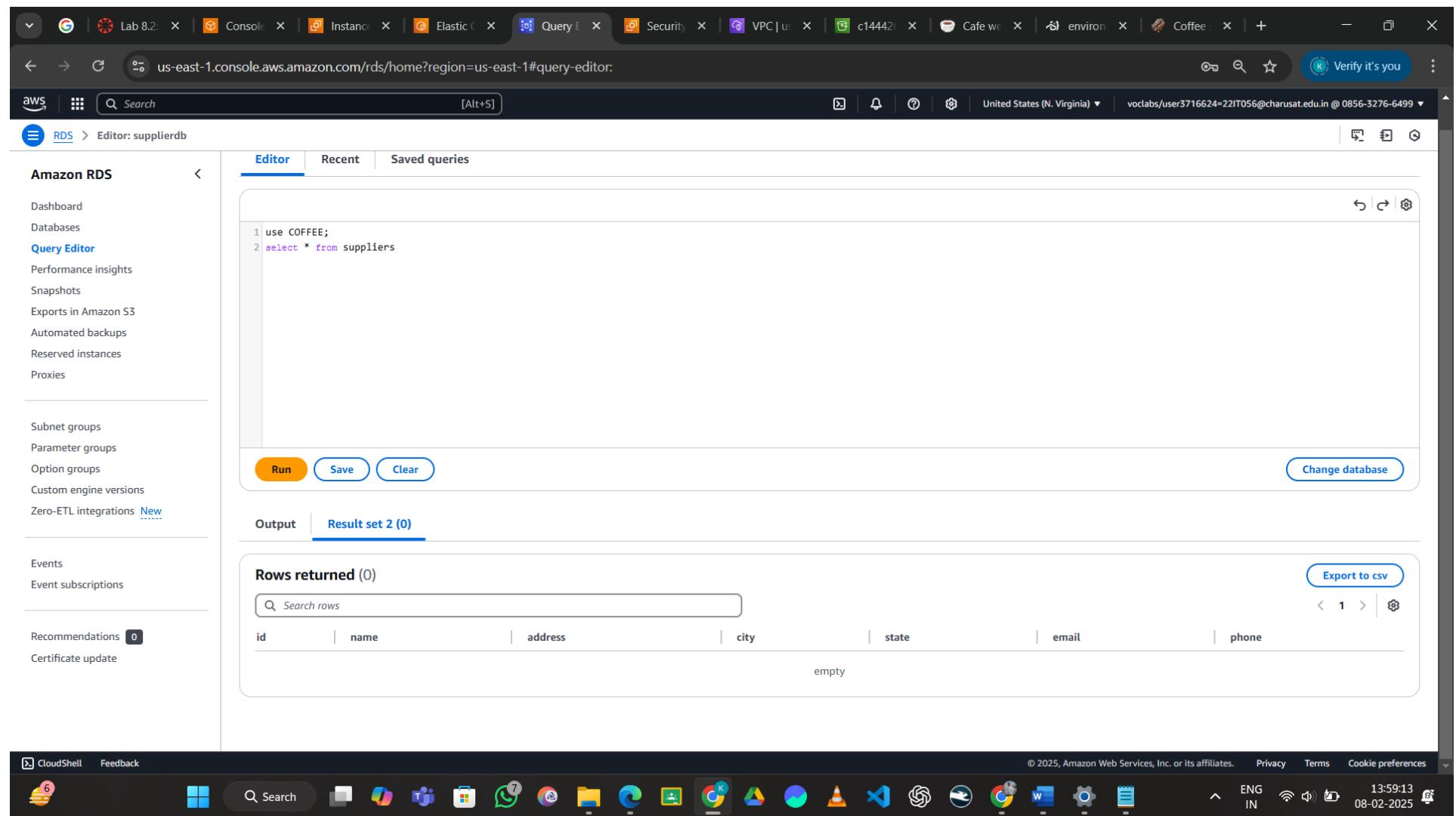


Figure 14: Use a coffee table and show a data

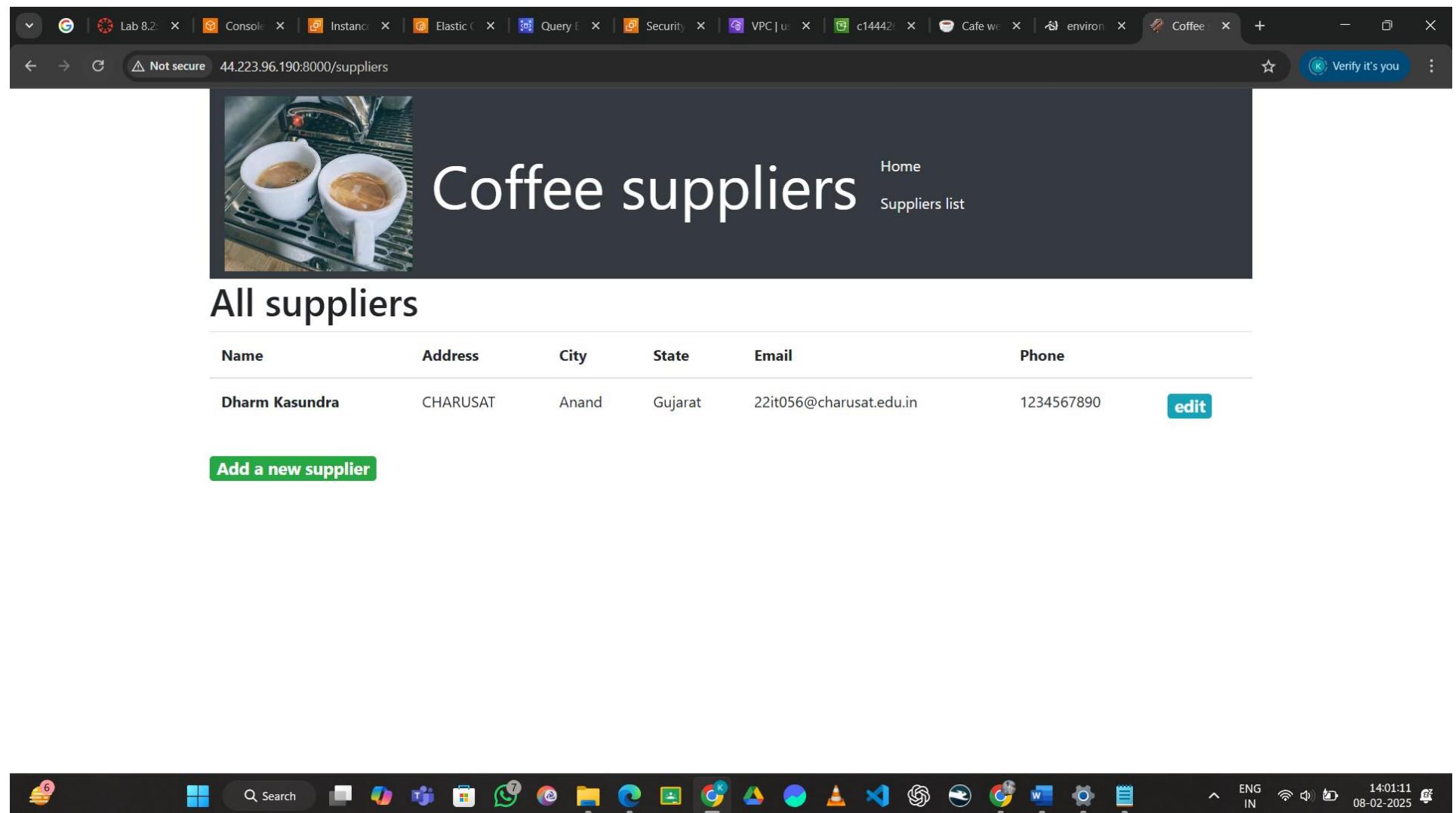


Figure 15: Add a new supplier using ui

The screenshot shows the AWS RDS Query Editor interface. On the left, the sidebar lists various RDS management options like Dashboard, Databases, and Query Editor. The main area has tabs for Editor, Recent, and Saved queries, with the Editor tab selected. A code editor contains the following SQL query:

```
1 use COFFEE;
2 select * from suppliers
```

Below the code editor are three buttons: Run, Save, and Clear. To the right of the Run button is a "Change database" link. The Output tab is selected, showing the results of the query. The results are titled "Rows returned (1)" and include a search bar. The table has columns: id, name, address, city, state, email, and phone. One row is displayed:

id	name	address	city	state	email	phone
1	Dharm Kasundra	CHARUSAT	Anand	Gujarat	22it056@charusat.edu.in	1234567890

At the bottom, there's a CloudShell feedback bar with various icons and links.

Figure 16: New supplier show in database

The screenshot shows a terminal window within a code editor interface. The terminal is running on an EC2 instance, connected via SSH. The user has run the command `cd ~/environment/resources` to navigate to the MySQL configuration directory. They then executed `mysql -h supplierdb-instance-1.cgds16dxo67k.us-east-1.rds.amazonaws.com -P 3306 -u admin -p` to connect to the MySQL database. The MySQL monitor prompt shows the connection details: Welcome to the MariaDB monitor. Commands end with ; or \g. Your MySQL connection id is 186. Server version: 8.0.36 Source distribution. Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others. Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. The user then ran the command `use COFFEE; select * from suppliers;` to select all rows from the `suppliers` table in the `COFFEE` database. The output shows one row of data:

id	name	address	city	state	email	phone
1	Dharm Kasundra	CHARUSAT	Anand	Gujarat	22it056@charusat.edu.in	1234567890

MySQL [(none)]> use COFFEE; select * from suppliers;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
+---+ | id | name | address | city | state | email | phone | +---+ | 1 | Dharm Kasundra | CHARUSAT | Anand | Gujarat | 22it056@charusat.edu.in | 1234567890 | +---+
1 row in set (0.000 sec)

Figure 17: Use a database in editor

The screenshot shows a browser window with multiple tabs at the top, including 'Lab 8.2.', 'Console', 'Instance', 'Elastic C...', 'Query E...', 'Security', 'VPC | us...', 'c14442...', 'Cafe we...', 'environment', 'Coffee', and a '+' tab. The main content area displays a terminal interface within a dark-themed code editor. On the left, the 'EXPLORER' sidebar shows a file tree under 'ENVIRONMENT' containing files like 'python_3', 'permissions.py', 'upload_items.py', 'resources', 'codebase_partner', 'website', 'coffee_db_dump.sql', 'permissions.py', 'public_policy.json', 'seed.py', 'setup.sh', and 'code.zip'. The 'TERMINAL' tab is active, showing MySQL queries and their results:

```
Query OK, 0 rows affected (0.000 sec)
Query OK, 0 rows affected (0.000 sec)
Query OK, 0 rows affected (0.000 sec)
Query OK, 0 rows affected, 1 warning (0.000 sec)
Query OK, 0 rows affected, 1 warning (0.000 sec)
Query OK, 0 rows affected, 1 warning (0.001 sec)
Query OK, 0 rows affected (0.000 sec)

MySQL [COFFEE]> use COFFEE; select * from suppliers;
Database changed
+----+-----+-----+-----+-----+-----+
| id | name           | address        | city       | state    | email          | phone      |
+----+-----+-----+-----+-----+-----+
| 1  | AnyCompany coffee suppliers | 123 Any Street | Any Town  | WA       | info@example.com | 555-555-0100 |
| 2  | Central Example Corp. coffee | 100 Main Street | Nowhere   | CO       | info@example.net | 555-555-0101 |
| 3  | North East AnyCompany coffee suppliers | 1001 Main Street | Any Town  | NY       | info@example.co | 555-555-0102 |
| 4  | SE Example corp coffee suppliers | 200 1st street  | None city | GA       | info@example.org | 555-555-0103 |
| 5  | SW Example Corp. coffee | 333 Main st    | Anytown   | AZ       | info@example.me | 555-555-0104 |
| 6  | Northern Example Corp. coffee | 444 Main st    | Not town  | MN       | coffee@example.com | 555-555-0106 |
| 7  | West coast example Corp. coffee | 1212 SE 30th Ave | Any beach | CA       | coffee@example.coffee | 555-555-0107 |
| 8  | Southern AnyCompany coffee suppliers | 555 Main st    | Anytown   | TX       | coffee@example.biz | 555-555-0108 |
+----+-----+-----+-----+-----+-----+
8 rows in set (0.007 sec)
```

The bottom status bar shows system icons and the date/time: '14:04:04 08-02-2025'.

Figure 18: Fetch all suppliers

The screenshot shows a browser window with multiple tabs at the top, including 'Lab 8.2.', 'Console', 'Instance', 'Elastic C...', 'Query E...', 'Security', 'VPC | us...', 'c14442...', 'Cafe we...', 'environment', 'Coffee', and a '+' tab. The main content area has a dark theme. On the left, there's an 'EXPLORER' sidebar with a tree view showing a directory structure under 'ENVIRONMENT'. The root node 'ENVIRONMENT' contains 'python_3' (with files 'permissions.py' and 'upload_items.py'), 'resources' (with files 'codebase_partner', 'website', 'coffee_db_dump.sql', 'permissions.py', 'public_policy.json', 'seed.py', 'setup.sh', and 'code.zip'), and a file 'code.zip'. Below the sidebar are 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS' tabs. The 'TERMINAL' tab is active, displaying a MySQL command-line interface. The command 'select * from beans;' is run, and the results are shown in a table:

	id	supplier_id	type	product_name	price	description	quantity
1	1	Arabica	Best bean	18.00	Delicious, smooth coffee.	1000	
2	1	Robusta	Great bean	12.00	Full bodied, good to the last drop.	800	
3	2	Robusta	Top bean	10.00	Great all around bean.	500	
4	2	Liberica	Better bean	14.00	This bean stands above the rest.	600	
5	3	Excelsa	Premiere bean	18.00	The best bean in all the land	200	
6	4	Arabica	House bean	11.00	A solid performer.	900	
7	4	Robusta	quality bean	13.00	A great bean for daily use.	350	
8	5	Robusta	Superb bean	16.00	No bean is better	700	
9	5	Liberica	Top tier bean	15.00	The bean that impresses.	300	
10	6	Arabica	Stellar bean	13.00	The top star of beans	300	
11	7	Robusta	Terrific bean	12.00	This is a great bean	800	
12	7	Liberica	Supreme bean	17.00	Solid performing bean. Light roast for smooth taste.	700	
13	8	Liberica	Ace bean	10.00	Medium roast bean. Good for brewed coffee.	1000	
14	8	Excelsa	Unrivaled bean	16.00	Dark roast bean. Best for espresso.	300	

At the bottom of the terminal, it says '14 rows in set (0.002 sec)'. The status bar at the bottom of the screen shows various icons and the text 'Layout: US'.

Figure 19: Fetch all beans

The screenshot shows a web browser window with multiple tabs open at the top, including 'Lab 8.2', 'Console', 'Instance', 'Elastic C...', 'Query E...', 'Security', 'VPC | us...', 'c14442...', 'Cafe we...', 'environ...', and 'Coffee'. The main content area displays a website titled 'Coffee suppliers' with a sub-section 'All suppliers'. The page features a dark header with a photo of two cups of coffee on the left and navigation links for 'Home' and 'Suppliers list'. Below the header is a large heading 'All suppliers' and a table listing eight supplier entries. Each entry includes fields for Name, Address, City, State, Email, and Phone, along with a blue 'edit' button. The table rows are as follows:

Name	Address	City	State	Email	Phone
AnyCompany coffee suppliers	123 Any Street	Any Town	WA	info@example.com	555-555-0100
Central Example Corp. coffee	100 Main Street	Nowhere	CO	info@example.net	555-555-0101
North East AnyCompany coffee suppliers	1001 Main Street	Any Town	NY	info@example.co	555-555-0102
SE Example corp coffee suppliers	200 1st street	None city	GA	info@example.org	555-555-0103
SW Example Corp. coffee	333 Main st	Anytown	AZ	info@example.me	555-555-0104
Northern Example Corp. coffee	444 Main st	Not town	MN	coffee@example.com	555-555-0106
West coast example Corp. coffee	1212 SE 30th Ave	Any beach	CA	coffee@example.coffee	555-555-0107

The bottom of the screen shows a Windows taskbar with various icons and system status indicators.

Figure 20: All supplier are show in websites

The screenshot shows a terminal window within a dark-themed IDE interface. The terminal tab is active, displaying the following command and its output:

```
[ec2-user@ip-10-0-1-24 environment]$ cd ~/environment
[ec2-user@ip-10-0-1-24 environment]$ mkdir bean
[ec2-user@ip-10-0-1-24 environment]$ cd bean
[ec2-user@ip-10-0-1-24 bean]$ aws elasticbeanstalk create-application --application-name MyNodeApp
{
    "Application": {
        "ApplicationArn": "arn:aws:elasticbeanstalk:us-east-1:085632766499:application/MyNodeApp",
        "ApplicationName": "MyNodeApp",
        "DateCreated": "2025-02-08T08:37:27.900000+00:00",
        "DateUpdated": "2025-02-08T08:37:27.900000+00:00",
        "ConfigurationTemplates": [],
        "ResourceLifecycleConfig": {
            "VersionLifecycleConfig": {
                "MaxCountRule": {
                    "Enabled": false,
                    "MaxCount": 200,
                    "DeleteSourceFromS3": false
                },
                "MaxAgeRule": {
                    "Enabled": false,
                    "MaxAgeInDays": 180,
                    "DeleteSourceFromS3": false
                }
            }
        }
    }
....skipping...
{
    "Application": {
        "ApplicationArn": "arn:aws:elasticbeanstalk:us-east-1:085632766499:application/MyNodeApp",
        "ApplicationName": "MyNodeApp",
        "DateCreated": "2025-02-08T08:37:27.900000+00:00",
```

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/>