Docker Containers Migration Report - Café Web Application

Objectives:

1. Understand the process of migrating a web application to Docker containers.
2. Learn to create a Dockerfile to configure the container environment.
3. Practice building a Docker image and running a container.
4. Familiarize yourself with using Amazon Elastic Container Registry (ECR) to store and manage Docker images.

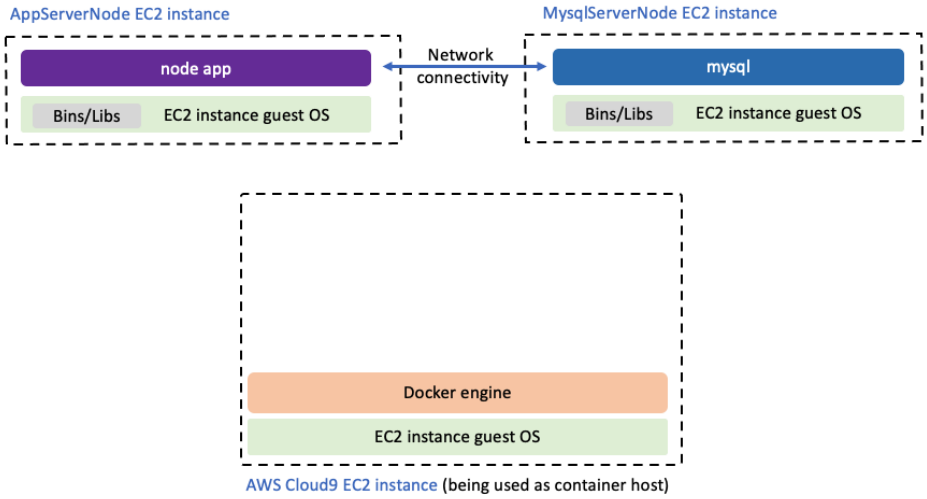
Scenario: Café's Gourmet Coffee Expansion

The café owners, Frank and Martha, are thrilled with the growing popularity of their gourmet coffee offerings, especially the cappuccinos and lattes that customers can't seem to get enough of. However, sourcing high-quality coffee beans consistently has been a challenge. Their luck changed when they discovered that one of their favorite coffee suppliers was looking to sell her company. Without hesitation, Frank and Martha seized the opportunity and acquired the coffee supplier's business.

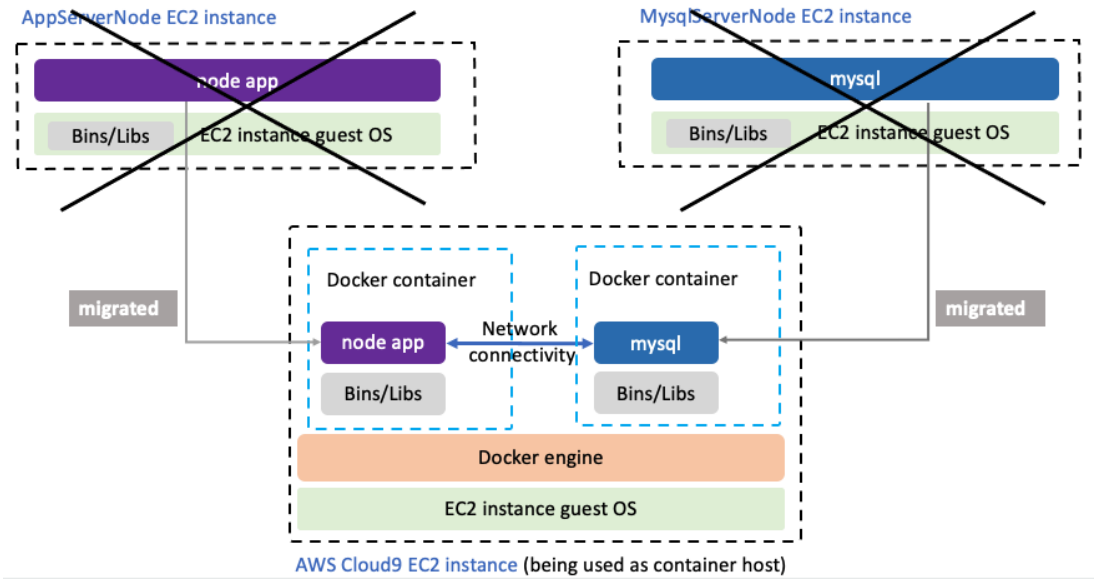
The acquired coffee supplier has been running an inventory tracking application on an AWS account. To seamlessly integrate this application into the café's existing infrastructure, Sofía has been tasked with understanding the application's workings and devising a plan for migration.

In this lab, you step into the role of Sofía. Your mission is to migrate the acquired application to run on Docker containers, ensuring scalability, portability, and efficient management within the café's application ecosystem.

The initial architecture provided in AWS sets the stage for your migration efforts:



By the *end* of this lab, you will have migrated the application and the backend database to run as Docker containers, as shown in the following diagram:



Development:

Task 1: Preparing the Development Environment

1. **Connect to AWS Cloud9 IDE:**
    - Open the AWS Management Console and go to Cloud9.
    - Find and open the existing IDE named "Cloud9 Instance."
2. **Download and Extract Files:**

    In the Cloud9 terminal, run:

```
voclabs:~/environment $ wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/06-lab-containers/code.zip -P /home/ec2-user/environment
--2024-04-19 22:23:37--  https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/06-lab-containers/code.zip
Resolving aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)... 3.5.86.12, 3.5.87.191, 52.92.130.130, ...
Connecting to aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)|3.5.86.12|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7071501 (6.7M) [application/zip]
Saving to: ‘/home/ec2-user/environment/code.zip’

100%[=============================================================================================================>] 7,071,501   8.33MB/s   in 0.8s

2024-04-19 22:23:38 (8.33 MB/s) - ‘/home/ec2-user/environment/code.zip’ saved [7071501/7071501]

voclabs:~/environment $
```

    Extract the downloaded ZIP file:

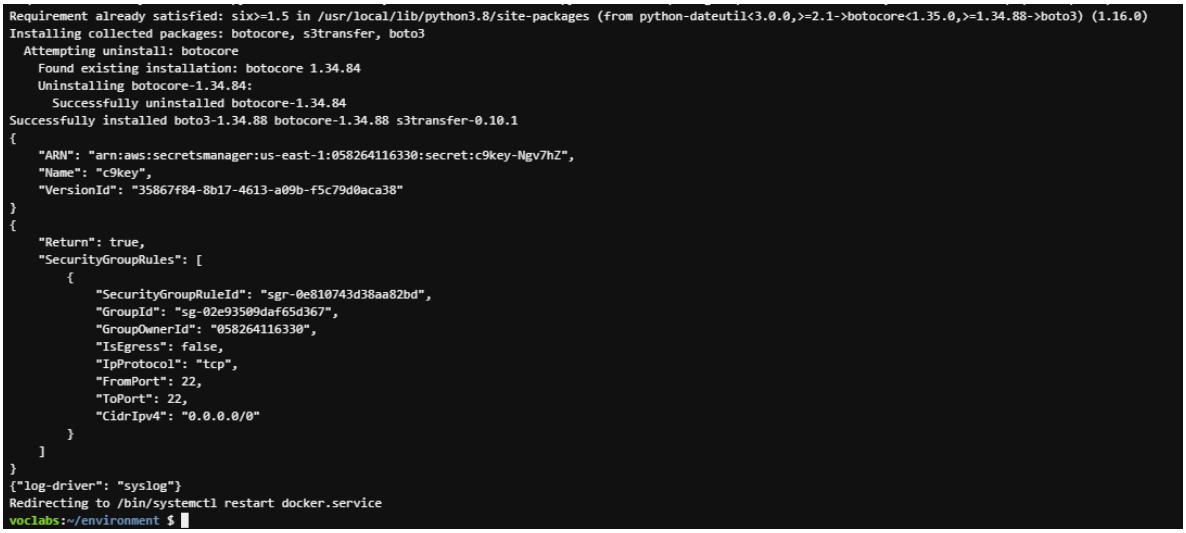
```
voclabs:~/environment $ unzip code.zip
Archive:  code.zip
  extracting: python_3/upload_items.py
  extracting: python_3/permissions.py
  extracting: resources/setup.sh
  extracting: resources/codebase_partner/network.template
  extracting: resources/codebase_partner/package.json
  extracting: resources/codebase_partner/package-lock.json
  extracting: resources/codebase_partner/README.md
  extracting: resources/codebase_partner/index.js
  extracting: resources/codebase_partner/views/nav.html
  extracting: resources/codebase_partner/views/500.html
  extracting: resources/codebase_partner/views/header.html
  extracting: resources/codebase_partner/views/supplier-form-fields.html
  extracting: resources/codebase_partner/views/home.html
  extracting: resources/codebase_partner/views/footer.html
  extracting: resources/codebase_partner/views/supplier-list-all.html
  extracting: resources/codebase_partner/views/supplier-add.html
  extracting: resources/codebase_partner/views/404.html
  extracting: resources/codebase_partner/views/supplier-update.html
  extracting: resources/codebase_partner/app/controller/supplier.controller.js
  extracting: resources/codebase_partner/app/models/supplier.model.js
  extracting: resources/codebase_partner/app/config/config.js
  extracting: resources/codebase_partner/public/css/base.css
  extracting: resources/codebase_partner/public/css/bootstrap.min.css.map
  extracting: resources/codebase_partner/public/css/bootstrap.min.css
  extracting: resources/codebase_partner/public/img/espresso.jpg
```

3. **Run Configuration Script:**

- Make the script executable and run it

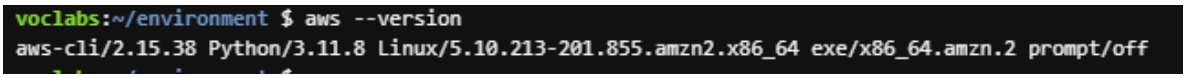
bash

chmod +x ./resources/setup.sh && ./resources/setup.sh

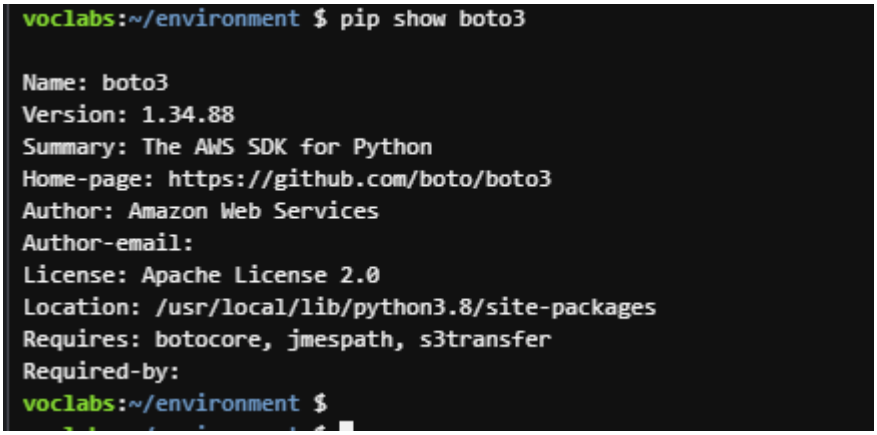


4. **Verify AWS CLI and Python SDK:**

- **Check the AWS CLI version:**



Verify the SDK for Python (boto3) is installed:



Task 2: Analyzing the Existing Application Infrastructure

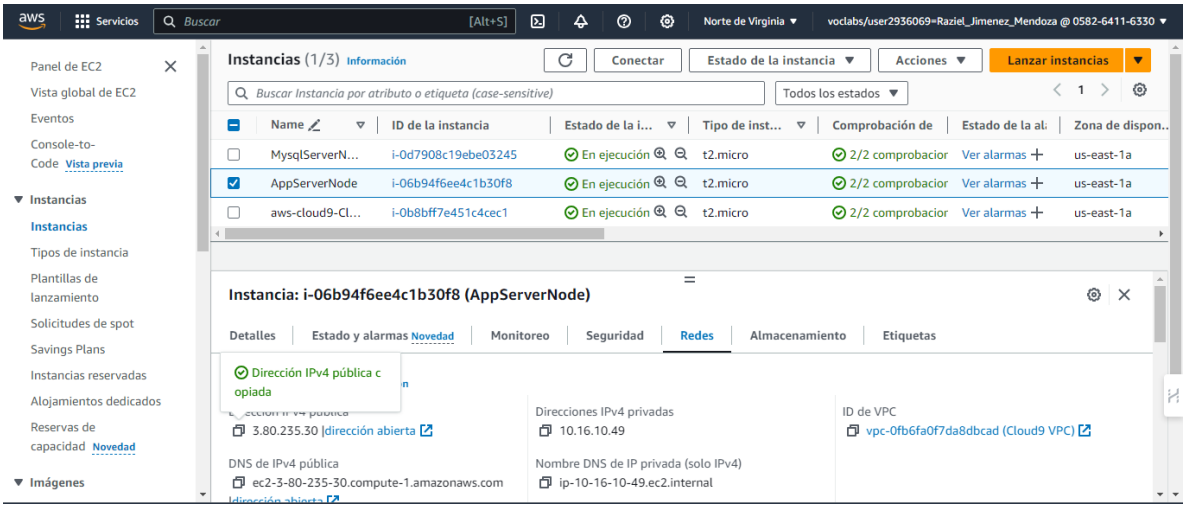
5. **Accessing the Coffee Supplier Application:**

    - **Open the coffee supplier application in a browser tab using its Public IPv4 address.**

6. **Inspecting Instances in EC2 Console:**

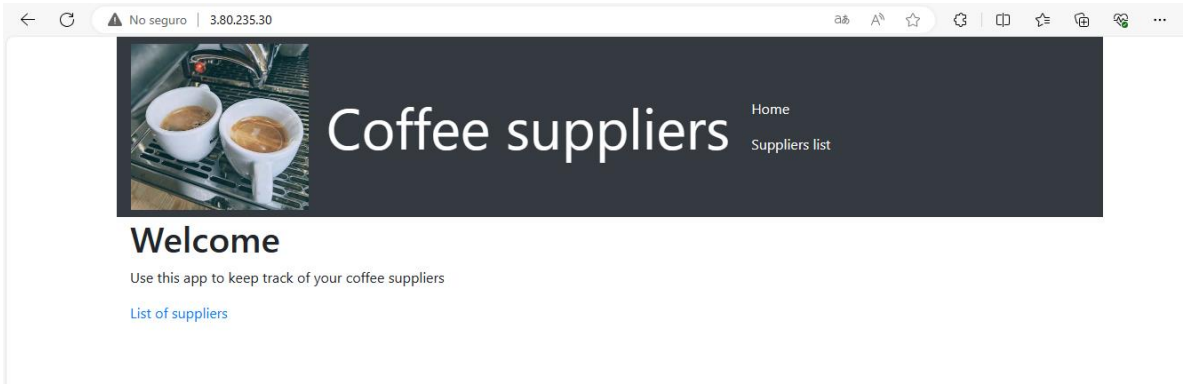
Navigate to the EC2 console under "Your environments."

Go to Instances and observe the running instances, including the Cloud9 instance, MysqlServerNode, and AppServerNode.



7. **Testing Web Application Functionality:**

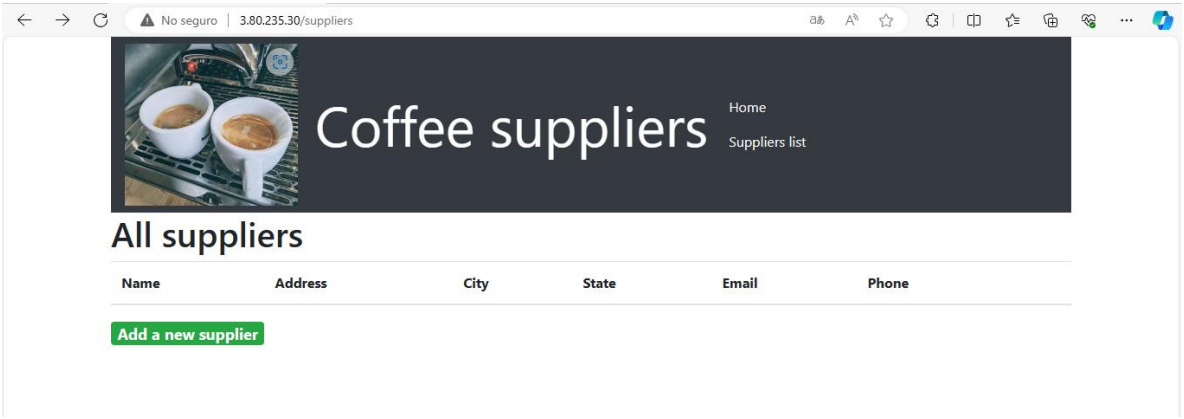
Access the coffee supplier website and test functionalities like adding a new supplier, editing records, and saving changes.



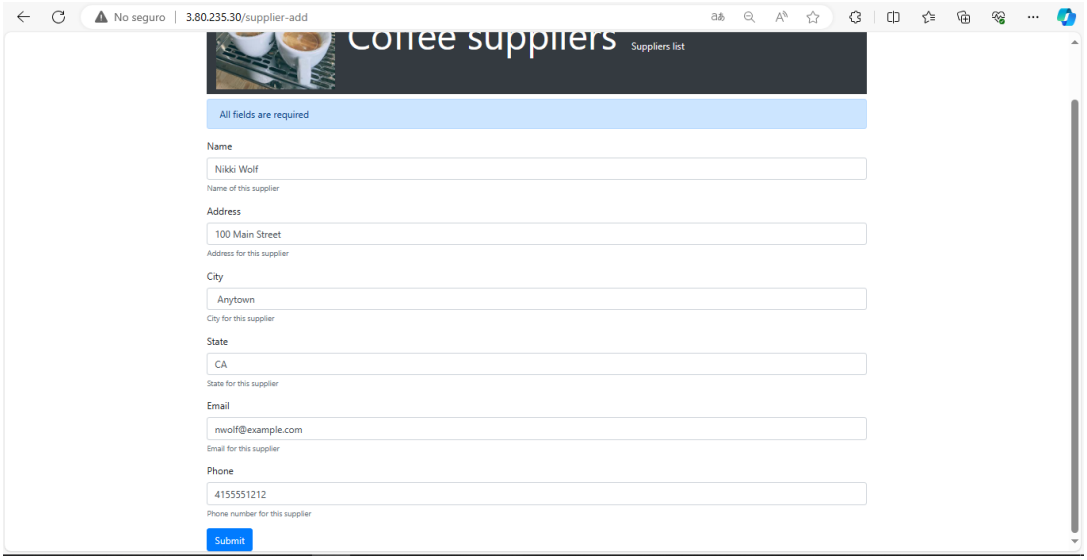
8. **Analyzing Web Application Code:**

In the AWS Cloud9 environment, explore the resources directory and codebase_partner directory to view the application code.

- **Optionally, connect to the AppServerNode instance via SSH to view installed files and configurations.**
  Choose List of suppliers and then choose Add a new supplier.



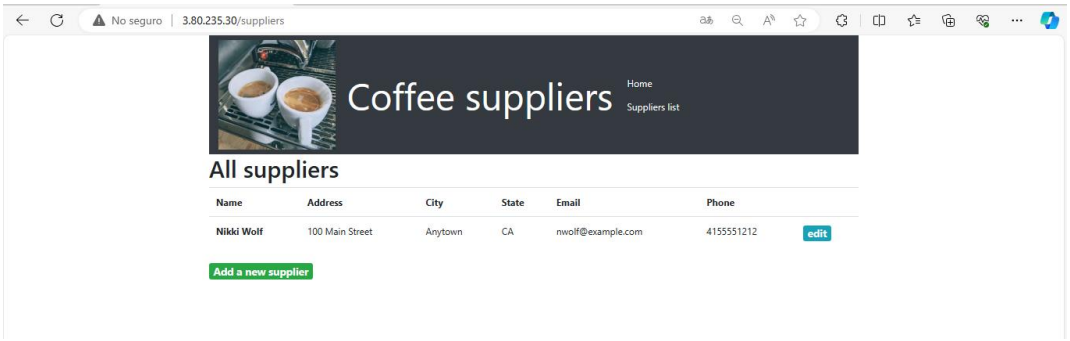
- **Fill in all of the fields with values. For example:**
    - **Name: Nikki Wolf**
    - **Address: 100 Main Street**
    - **City: Anytown**
    - **State: CA**
    - **Email: nwolf@example.com**
    - **Phone: 4155551212**
- **Choose Submit.**

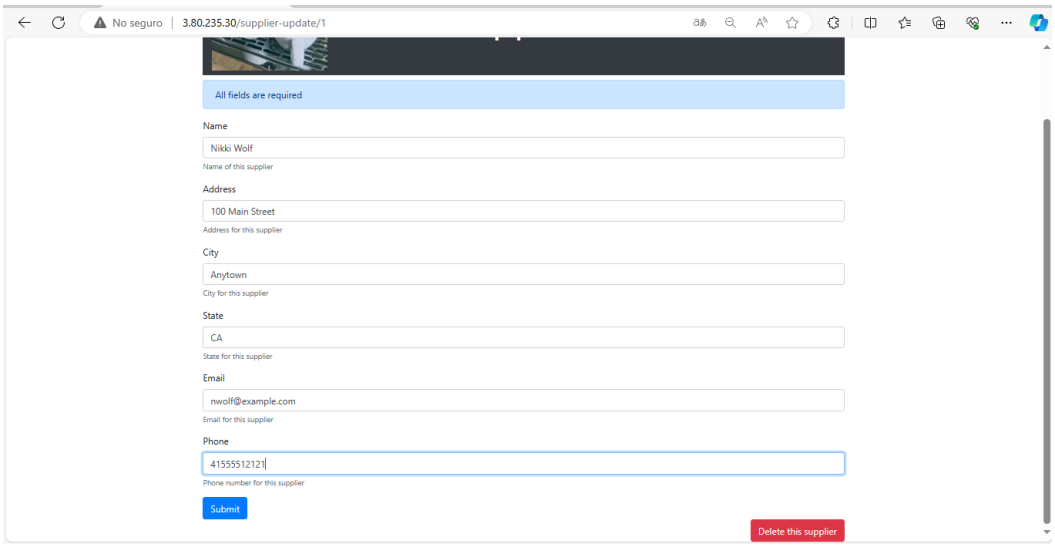


9. **The All suppliers page displays and includes the record that you submitted.**

- **Choose edit and change the record (for example, modify the phone number).**
- **To save the change, choose Submit.**

Notice that the change was saved in the record.

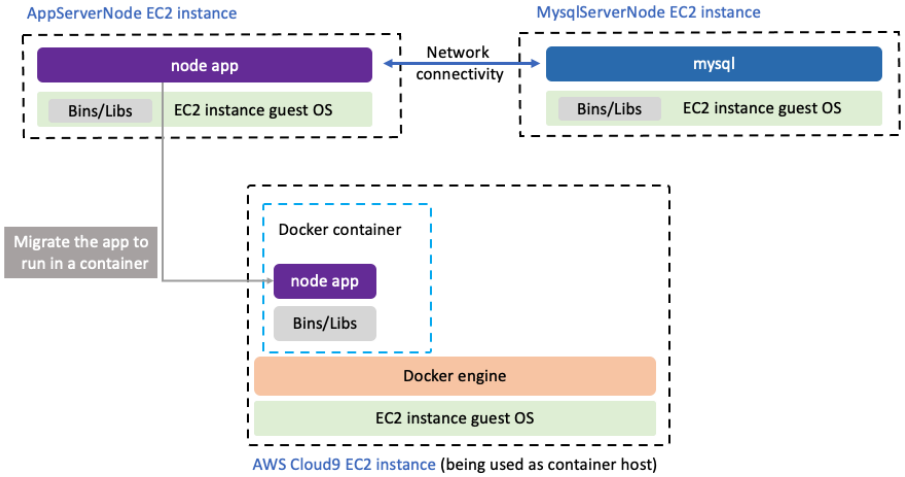




Task 3: Migrating the Application to a Docker Container

In this task, you will migrate an application that is installed directly on the guest OS of an Ubuntu Linux EC2 instance to run in a Docker container. The Docker container offers portability and can run on any OS with the Docker engine installed.

For convenience, you will run the container on the same EC2 instance hosting the AWS Cloud9 IDE. You'll use this IDE to build the Docker image and launch the Docker container.



10. **Create a Directory for Docker Container Code:**

    - Open the AWS Cloud9 IDE.
    - **Create and Navigate to the Containers Directory**
    - Create and Navigate to the Node App Directory:

      Inside the containers directory, create a new directory named “node_app”
    - Move the Source Code to the Node App Directory:

      Assuming you've copied the code base earlier, use the a command to move it into the new "node app directory

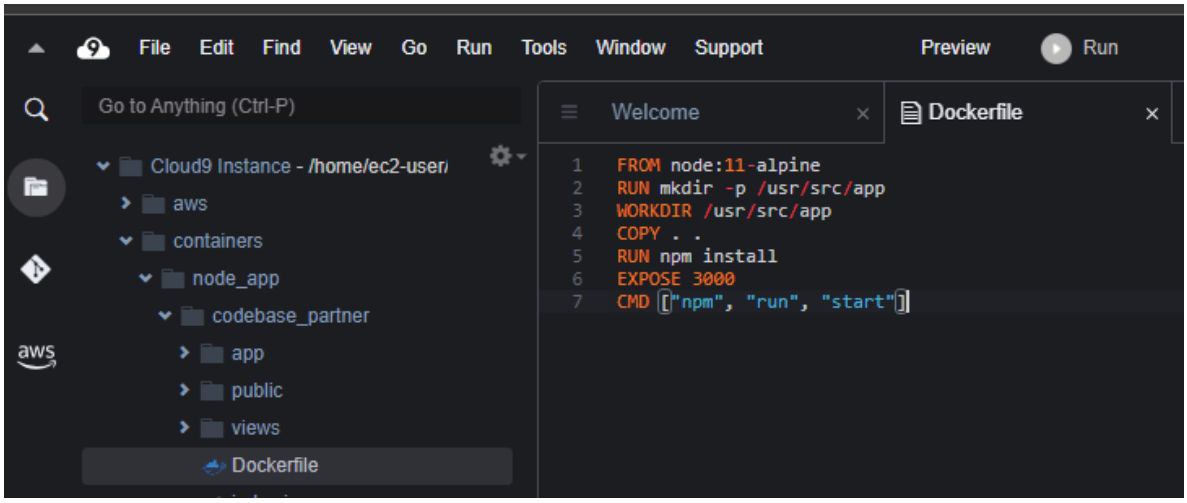
```
voclabs:~/environment $
voclabs:~/environment $ mkdir containers
voclabs:~/environment $ cd containers
voclabs:~/environment/containers $ mkdir node_app
voclabs:~/environment/containers $ cd node_app
voclabs:~/environment/containers/node_app $ mv ~/environment/resources/codebase_partner ~/environment/containers/node_app
```

11. Create a new Dockerfile

- named **Dockerfile** in the **node_app/codebase_partner** directory using the following command:

```
voclabs:~/environment/containers/node_app/codebase_partner $ touch Dockerfile
```

- In the left navigation pane, browse to and open the empty Dockerfile that you just created.
- Copy and paste the following code into the Dockerfile:



12. **Building Docker Image** To build the Docker image from the Dockerfile, I executed the following command in the AWS Cloud9 terminal:

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker build --tag node_app .
Sending build context to Docker daemon  1.272MB
Step 1/7 : FROM node:11-alpine
11-alpine: Pulling from library/node
e7c96db7181b: Pull complete
0119aca44649: Pull complete
40df19605a18: Pull complete
82194b8b4a64: Pull complete
Digest: sha256:8bb56bab197299c8ff820f1a55462890caf08f57ffe3b91f5fa6945a4d505932
Status: Downloaded newer image for node:11-alpine
 ---> f18da2f58c3d
```

13. **Verifying Docker Image Creation** After building the Docker image using the previous command, I confirmed its creation by listing all Docker images that my Docker client is aware of. I executed the following command in the AWS Cloud9 terminal:

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker images
REPOSITORY   TAG         IMAGE ID       CREATED          SIZE
node_app     latest      b4f56575b062   2 minutes ago    84.5MB
node         11-alpine   f18da2f58c3d   4 years ago      75.5MB
```

14. Create and run a Docker container based on the Docker image.
    To create and run a Docker container from the image, run the following command:

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker run -d --name node_app_1 -p 3000:3000 node_app
06ef2c5d089e2b638c124f18b2d4fae6439ad7f20bd92e124ba15e9149660568
```

    Command to View Running Containers:

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS             PORTS                                       NAMES
06ef2c5d089e   node_app   "docker-entrypoint.s…"   2 minutes ago    Up About a minute  0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node_app_1
```

15. Verify that the node application is now running in the container.

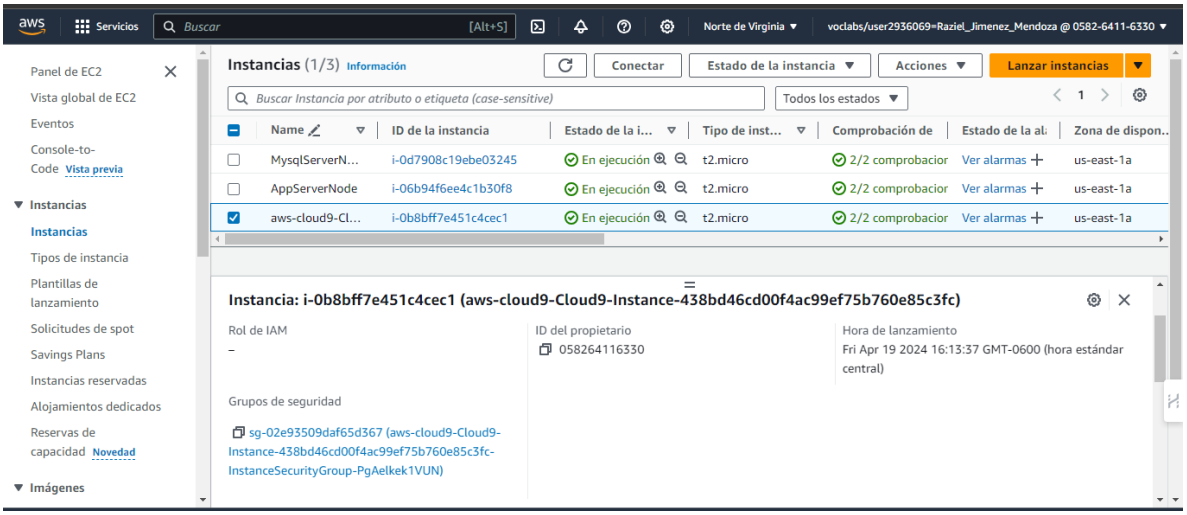
- To check that the container is working on the correct port, run the following command:

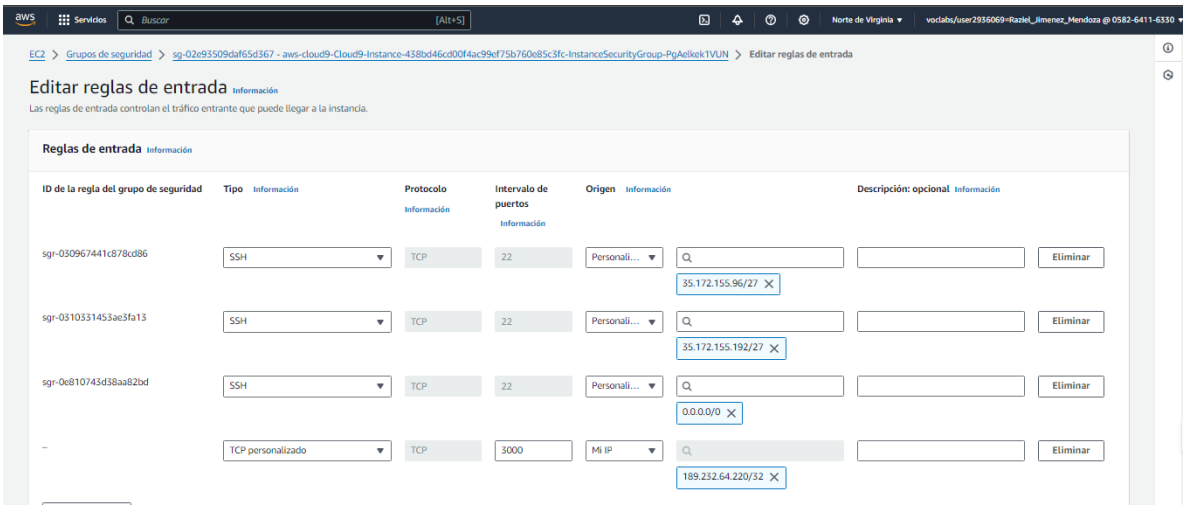
```
voclabs:~/environment/containers/node_app/codebase_partner $ curl http://localhost:3000
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <link rel="stylesheet" href="/css/base.css">
    <title>Coffee suppliers</title>
</head>
<body>
```

16. Adjust the security group of the AWS Cloud9 EC2 instance to allow inbound network traffic on port 3000 from your computer.

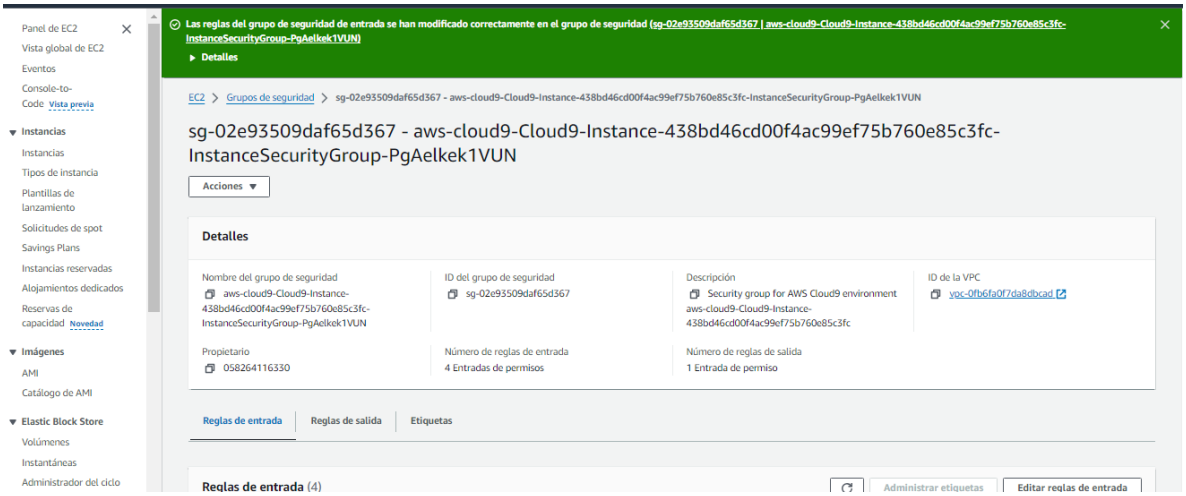
Steps:

1. Go to the AWS Management Console and navigate to the EC2 console.
2. Select the **aws-cloud9-Cloud9-Instance** instance.
3. Choose the Security tab and click on the **aws-cloud9-Cloud9-Instance** security group hyperlink.
4. Go to the Inbound rules tab and click Edit inbound rules.
5. Add a new rule with the following configurations:
    - Type: Custom TCP
    - Port range: 3000
    - Source: My IP
6. Save the rules.



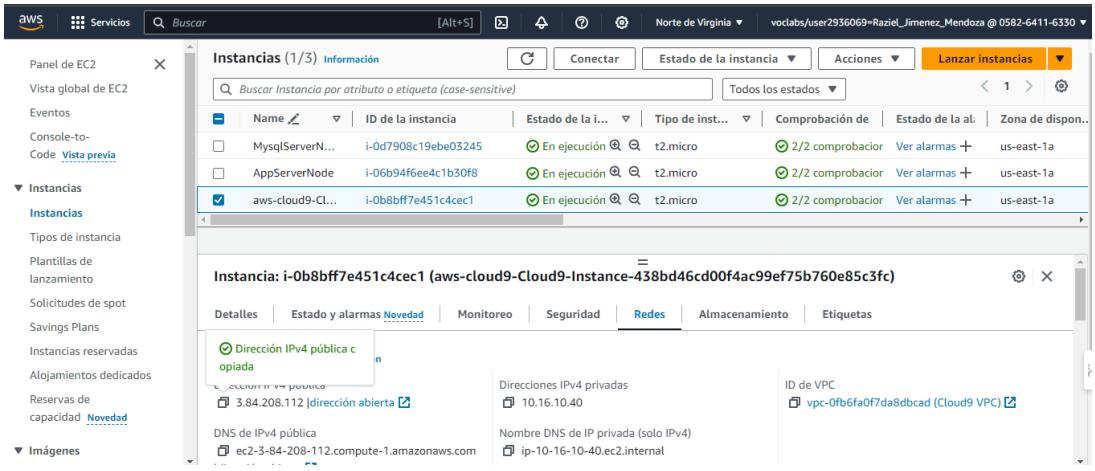


The inbound security group rules have been successfully modified in the security group

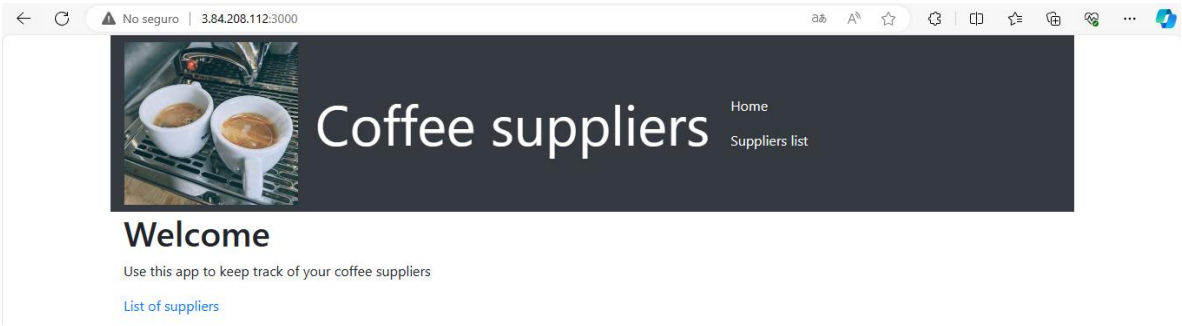


17. Access the web interface of the containerized application:

1. Go to the EC2 console and select Instances.
2. Choose the **aws-cloud9-Cloud9-Instance** instance.
3. Copy the Public IPv4 address from the Details tab.
4. Open a new browser tab and paste the IP address followed by **:3000**.
5. The web application loads in the browser, now running from the container on the AWS Cloud9 instance's Docker hypervisor.

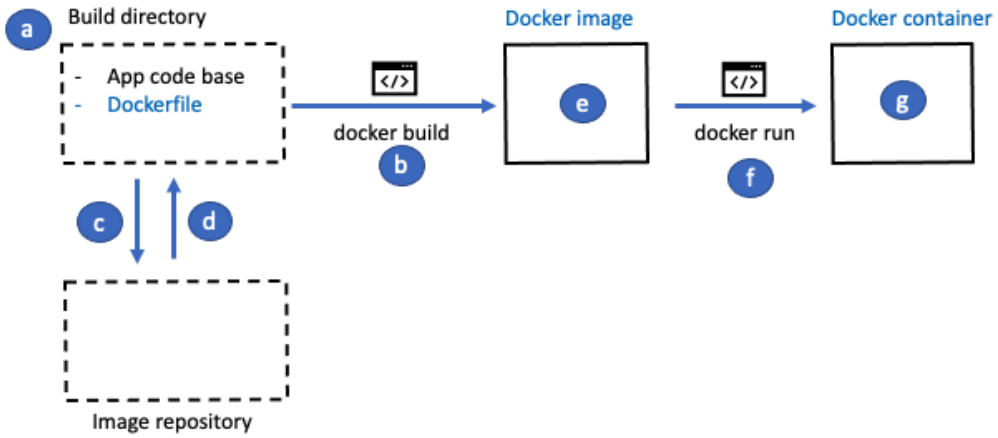


Web Page



Summary of how you have used Docker so far

You just completed a series of steps with Docker. The following diagram summarizes what you have accomplished with Docker so far.



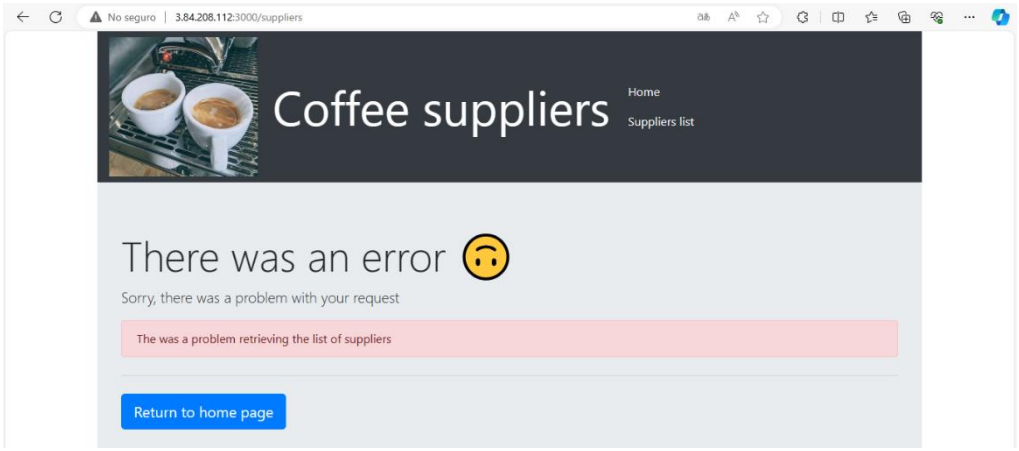
18. Analyze the database connection issue.

In the coffee suppliers application, choose List of suppliers.

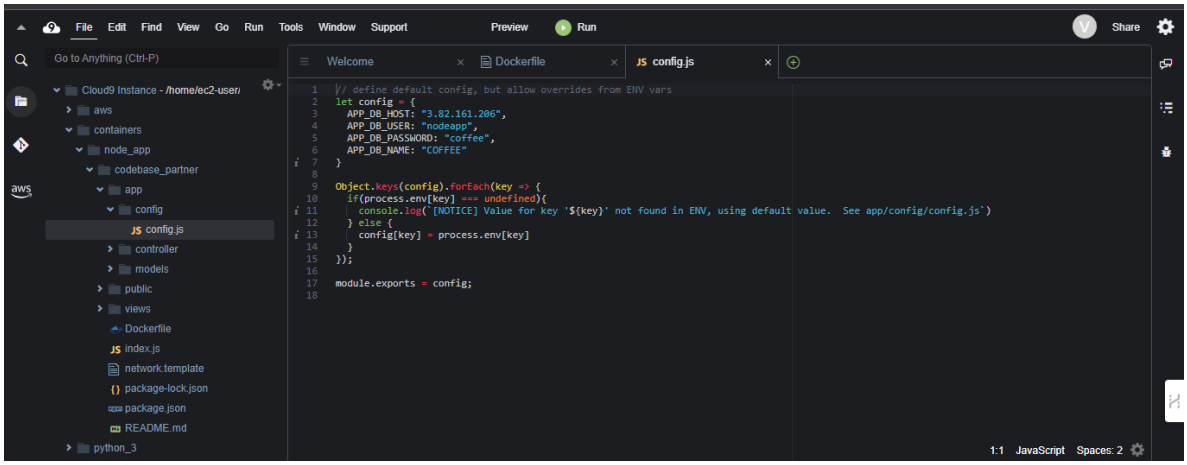
You see an error stating that there was a problem retrieving the list of suppliers.

Analysis: This is because the node_app_1 container is having trouble reaching the MySQL database, which is running on the EC2 instance named MysqlServerNode.

Return to the AWS Cloud9 IDE browser tab.



Open the **config.js** file in the **containers/node_app/codebase_partner/app/config/** directory.



To establish a terminal connection to the container and observe the settings, you can use the following steps:

2. At the top of the image, display the output of the **docker ps** command, highlighting the CONTAINER ID of the container you want to connect to.
3. Below that, show the execution of the **docker exec -it <container_id> sh** command, where **<container_id>** is the container ID, to open a terminal session inside the container.
4. Next, display the commands you run inside the container, such as **whoami** to show the current user, **env** to display environment variables, and any other relevant commands you want to showcase.
5. Finally, show the output when exiting the terminal session inside the container using the **exit** command.

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                                NAMES
06ef2c5d089e   node_app   "docker-entrypoint.s…"   30 minutes ago   Up 30 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node_app_1
voclabs:~/environment/containers/node_app/codebase_partner $ docker exec -it 06ef2c5d089e sh
/usr/src/app # whoami
root
/usr/src/app # su node
/usr/src/app $ env
USER=node
NODE_VERSION=11.15.0
HOSTNAME=06ef2c5d089e
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 $ exit
/usr/src/app # exit
voclabs:~/environment/containers/node_app/codebase_partner $
```

Stop and remove the container that has the database connectivity issue.

- To get the ID of the running container, run the following command:

docker ps

Notice the name of the application that is returned in the **NAMES** column.

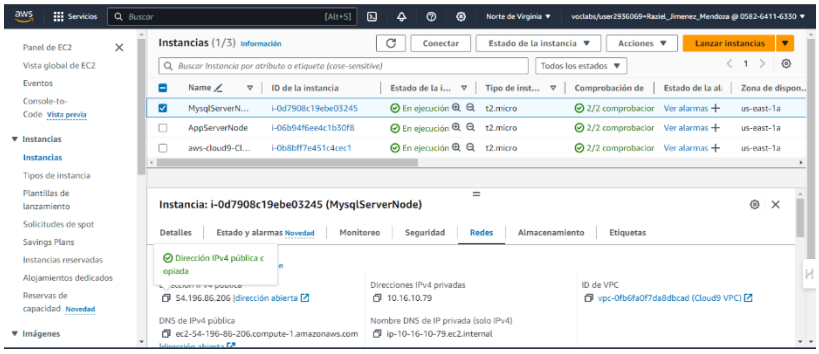
- To stop and remove the container, run the following command:

docker stop node_app_1 && docker rm node_app_1

```
voclabs:~/environment/containers/node_app/codebase_partner $ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                                NAMES
06ef2c5d089e   node_app   "docker-entrypoint.s…"   41 minutes ago   Up 41 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node_app_1
voclabs:~/environment/containers/node_app/codebase_partner $ docker stop node_app_1 && docker rm node_app_1
node_app_1
node_app_1
voclabs:~/environment/containers/node_app/codebase_partner $
```

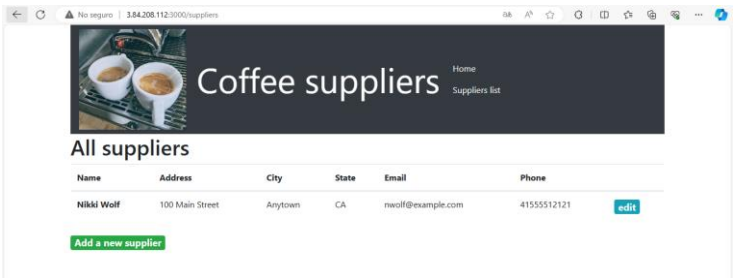
19. Launch a new container. This time, you will pass an environment variable to tell the node application the correct location of the database.

- Return to the EC2 console, and copy the **Public IPv4 address** value of the **MysqlServerNode** EC2 instance.
- Return to the AWS Cloud9 terminal.
- To run the application in a container and pass an environment variable to specify the database location, run the following command. Replace <ip-address> with the actual public IPv4 address of the MysqlServerNode EC2 instance:



1. **First Step (Docker Run Command):**
    - Show the execution of the command **docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST="54.196.86.206" node_app** in the terminal.
    - Highlight the CONTAINER ID (**f1c03d957870** in this case) generated after starting the container.
2. **Second Step (Docker PS Command):**
    - Display the execution of the command **docker ps** to show the running container with its associated information (CONTAINER ID, IMAGE, STATUS, PORTS, NAMES).
3. **Third Step (Docker Exec Command):**
    - Show the execution of the command **docker exec -it f1c03d957870 sh** to enter the container's terminal.
    - Display the user switch to **node** inside the container and the viewing of environment variables using **env**.
    - Highlight the **APP_DB_HOST** environment variable configured with the database host's IP address (**54.196.86.206**).
4. **Final Steps and Terminal Output:**
    - Display the **exit** commands used to exit the **node** user session and the container's terminal.

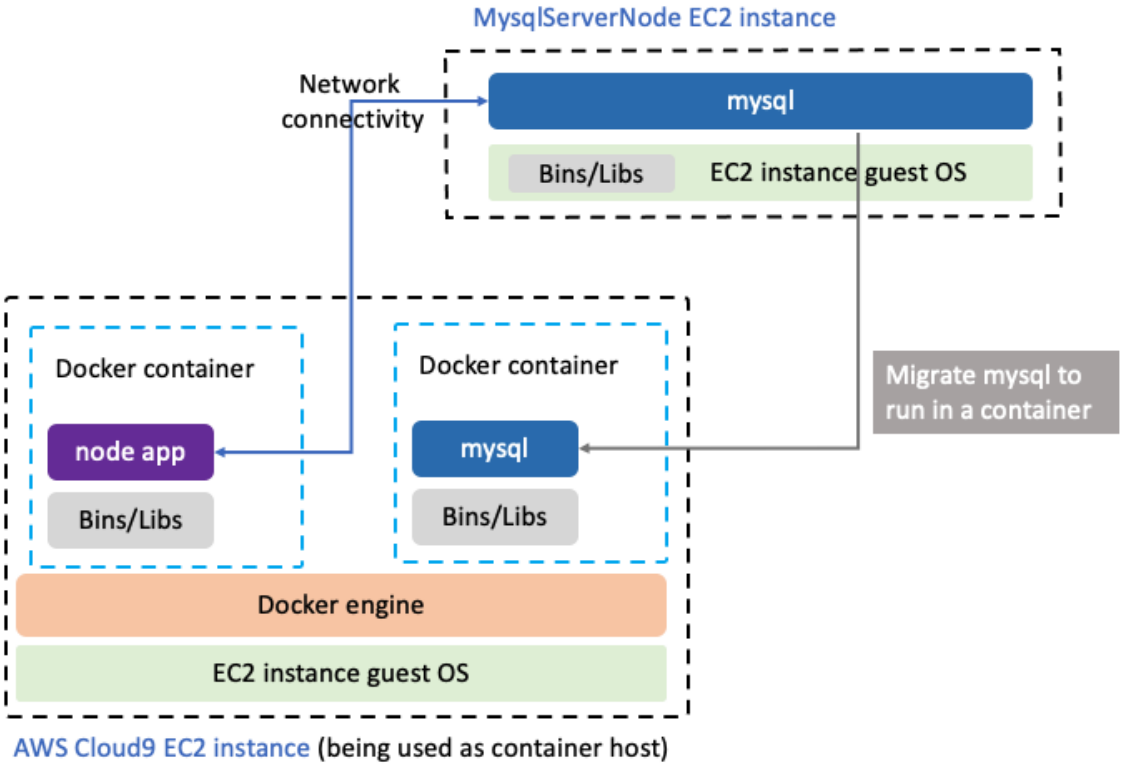
```
voclabs:~/environment/containers/node_app/codebase_partner $ docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST="54.196.86.206" node_app
f1c03d957870af6dedd0673145e6283ef273db43194d5d8621dd5d147582946a
voclabs:~/environment/containers/node_app/codebase_partner $ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                                 NAMES
f1c03d957870   node_app   "docker-entrypoint.s…"   24 seconds ago   Up 23 seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node_app_1
voclabs:~/environment/containers/node_app/codebase_partner $ docker exec -it f1c03d957870 sh
/usr/src/app # su node
/usr/src/app $ env
USER=node
NODE_VERSION=11.15.0
HOSTNAME=f1c03d957870
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
APP_DB_HOST=54.196.86.206
/usr/src/app $ exit
/usr/src/app # ecit
sh: ecit: not found
/usr/src/app # exit
```



Task 4: Migrating the MySQL database to a Docker container

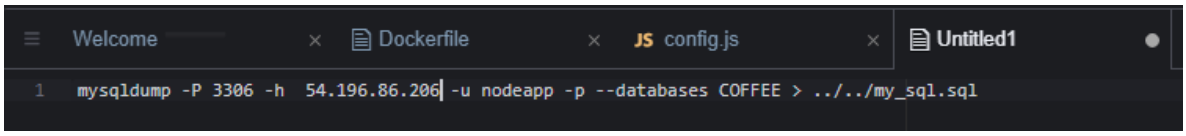
In this task, you will work to migrate the MySQL database to a container as well. To accomplish this task, you will dump the latest data that is stored in the database and use that to seed a new MySQL database running in a new Docker container.

The following diagram shows the migration that you will accomplish in this task:



20. Create a mysqldump file from the data that is currently in the MySQL database.

- Return to the AWS Cloud9 IDE, and close any file tabs that are open in the text editor.
- Choose **File** > **New File** and then paste the following code into the new file:



1. **Step 1: Change to the application code directory:**
    - Display the command **cd /home/ec2-user/environment/containers/node_app/codebase_partner** in the terminal to change to the directory where the application code is located.
2. **Step 2: Export MySQL Database:**
    - Show the command **mysqldump -P 3306 -h 54.196.86.206 -u nodeapp -p --databases COFFEE > ../../my_sql.sql** in the terminal.

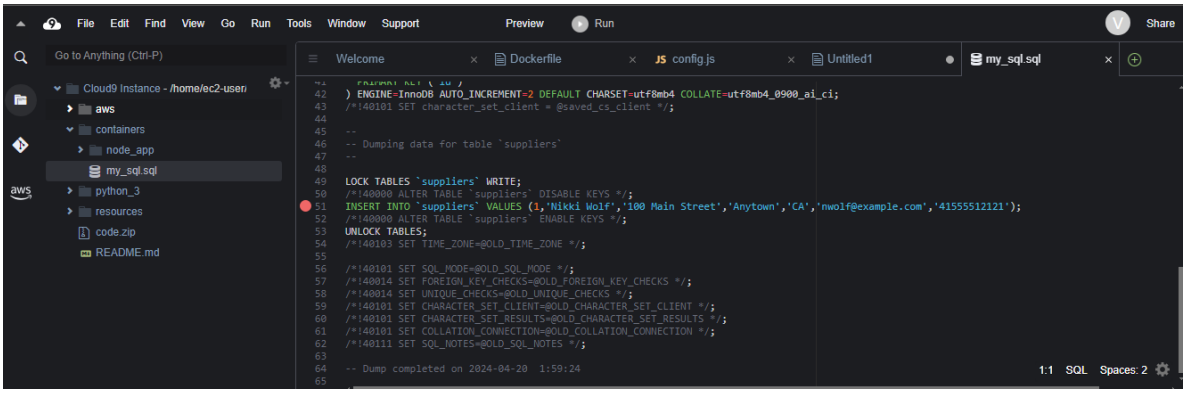
- Explain that this command is used to export the MySQL database named COFFEE to a file named **my_sql.sql**.
- Display the process of entering the password after executing the **mysqldump** command.

3. **Step 3: Completion of the Process:**
    - Display the terminal output after the database export process is completed.

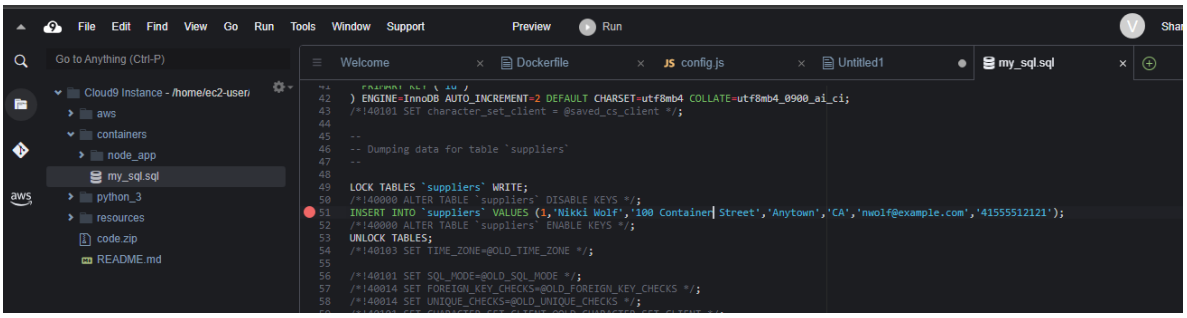
```
voclabs:~/environment/containers/node_app/codebase_partner $ cd /home/ec2-user/environment/containers/node_app/codebase_partner
voclabs:~/environment/containers/node_app/codebase_partner $ mysqldump -P 3306 -h  54.196.86.206 -u nodeapp -p --databases COFFEE > ../../my_sql.sq
l
Enter password:
voclabs:~/environment/containers/node_app/codebase_partner $
```

21. Open the mysqldump file and observe the contents.

- Open the **my_sql.sql** file in the AWS Cloud9 editor.
- Scroll through the contents of the file.
    - Notice that it will create a database named **COFFEE** and a table named **suppliers**.
    - Also, because you added a record using the application web interface earlier in this lab, the script inserts that record into the **suppliers** table.
- Make a small change to one of the values in the file.
    - Locate the line that starts with INSERT INTO. It will appear around line 51.



- Modify the address that you entered. For example, if the address has a street named Main change it to Container. **Note**: This change will help you later in the lab when you want to confirm that you are connected to the new database running on a container, and not the old database.
- Choose **File** > **Save** to the change.



22. In the terminal, to create a directory to store your mysql container code and navigate into the directory, run the following commands:

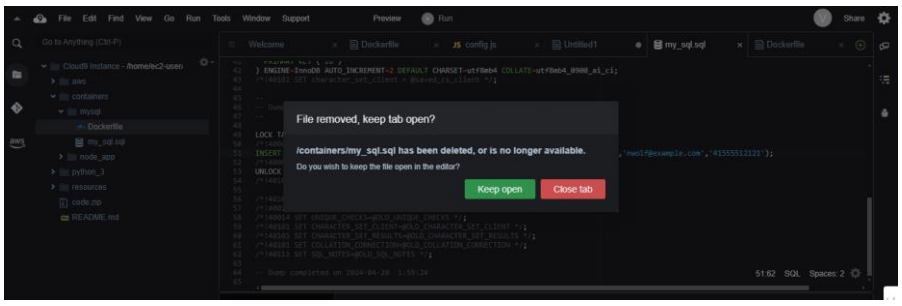
```
voclabs:~/environment/containers/node_app/codebase_partner $ cd /home/ec2-user/environment/containers
voclabs:~/environment/containers $ mkdir mysql
voclabs:~/environment/containers $ cd mysql
voclabs:~/environment/containers/mysql $
```

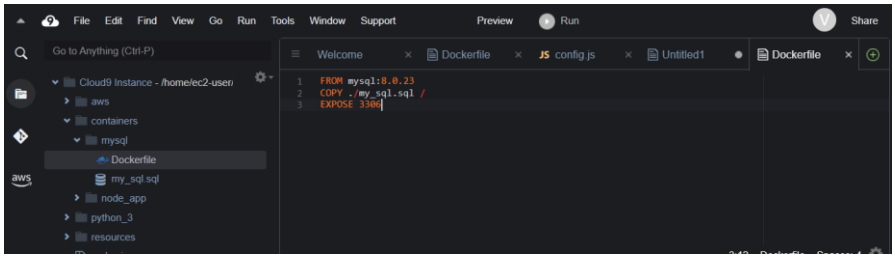
23. Create a Dockerfile.

- To create a new Dockerfile, run the following command:
- Execute the command mv ../my_sql.sql . in the terminal to move the my_sql.sql file from the parent directory to the current directory (mysql directory).
- Show the terminal output after successfully creating the Dockerfile and moving the SQL file

```
voclabs:~/environment/containers/mysql $ touch Dockerfile
voclabs:~/environment/containers/mysql $ mv ../my_sql.sql .
voclabs:~/environment/containers/mysql $
```

Open the empty Dockerfile (in **containers/mysql/**) and then copy and paste the following code into the file:





24. Attempt to free up some disk space on the AWS Cloud9 EC2 instance by removing unneeded files.

- Run the following command:

```
voclabs:~/environment/containers/mysql $ docker rmi -f $(docker image ls -a -q)
Error response from daemon: conflict: unable to delete 131124243b7b (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete b7e6b7f352b6 (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete b4f56575b062 (cannot be forced) - image is being used by running container f1c03d957870
Error response from daemon: conflict: unable to delete e49ff7a38fd1 (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete b6c2bbced02f (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete ce29f4929d86 (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete f18da2f58c3d (cannot be forced) - image has dependent child images
```

Finally, run the following command:

```
voclabs:~/environment/containers/mysql $ sudo docker image prune -f && sudo docker container prune -f
Total reclaimed space: 0B
Total reclaimed space: 0B
```

25. To build an image from the Dockerfile, run the following command:

```
voclabs:~/environment/containers/mysql $ docker build --tag mysql_server .
Sending build context to Docker daemon   5.12kB
Step 1/3 : FROM mysql:8.0.23
8.0.23: Pulling from library/mysql
f7ec5a41d630: Extracting [==================>                                ]  10.03MB/27.14MB
9444bb562699: Download complete
6a4207b96940: Download complete
181cefd361ce: Download complete
8a2090759d8a: Download complete
15f235e0d7ee: Download complete
d870539cd9db: Download complete
5726073179b6: Download complete
eadfac8b2520: Downloading [=========>                                          ]  21.93MB/113.1MB
f5936a8c3f2b: Download complete
cca8ee89e625: Download complete
6c79df02586a: Download complete
```

26. Verify that the Docker image was created.

```
voclabs:~/environment/containers/mysql $ docker images
REPOSITORY     TAG          IMAGE ID       CREATED          SIZE
mysql_server   latest       b55e8e7316ad   41 seconds ago   546MB
node_app       latest       b4f56575b062   2 hours ago      84.5MB
mysql          8.0.23       cbe8815cbea8   3 years ago      546MB
node           11-alpine    f18da2f58c3d   4 years ago      75.5MB
voclabs:~/environment/containers/mysql $
```

27. Create and run a Docker container based on the Docker image.

- To create and run a Docker container from the image, run the following command:

```
voclabs:~/environment/containers/mysql $ docker run --name mysql_1 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=rootpw -d mysql_server
ab0c3f23cc5ba4078ed8e591e54ce4021366bbdbd5ce2dfd34225091dfb5df5b
voclabs:~/environment/containers/mysql $
```

- To view the Docker containers that are currently running on the host, run the following command:

```
voclabs:~/environment/containers/mysql $ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
ab0c3f23cc5b   mysql_server   "docker-entrypoint.s…"   52 seconds ago   Up 51 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql_1
f1c03d957870   node_app       "docker-entrypoint.s…"   34 minutes ago   Up 34 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp              node_app_1
voclabs:~/environment/containers/mysql $
```

28. Import the data into the MySQL database and define a database user.

- Run the following command:

```
voclabs:~/environment/containers/mysql $ docker exec -i mysql_1 mysql -u root -prootpw < my_sql.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
voclabs:~/environment/containers/mysql $
```

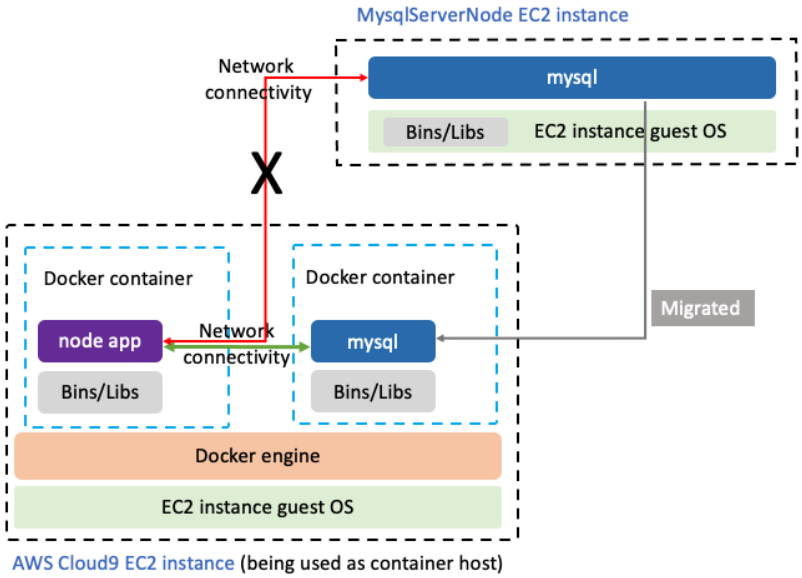
- To create a database user for the node application to use, run the following command:

```
voclabs:~/environment/containers/mysql $ docker exec -i mysql_1 mysql -u root  -prootpw -e "CREATE USER 'nodeapp' IDENTIFIED WITH mysql_native_password BY 'coffee'
; GRANT all privileges on *.* to 'nodeapp'@'%';"
mysql: [Warning] Using a password on the command line interface can be insecure.
voclabs:~/environment/containers/mysql $
```

Task 5: Testing the MySQL container with the node application

Recall that in a previous task you connected to the node application running in the container, but it was connected to the MySQL database that was running on the MysqlServerNode EC2 instance.

In this task, you will update the node application running in the container to point to the MySQL database running in the container. The following diagram shows the migration that you will accomplish in this task:



29. To stop and remove the node application server container, run the following command:

```
voclabs:~/environment/containers/mysql $ docker stop node_app_1 && docker rm node_app_1
node_app_1
node_app_1
```

30. Discover the network connectivity information.

```
voclabs:~/environment/containers/mysql $ docker inspect network bridge
[
    {
        "Name": "bridge",
        "Id": "68d237b6a19603eae68ad583fa20692110a2dda26240b0606f9ccf149e92caf6",
        "Created": "2024-04-19T22:30:58.372982196Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
```

31. Start a new node application Docker container with the specified environment variable **APP_DB_HOST**.

Run the following command, replacing **<ip-address>** with the actual IPv4 address value discovered earlier (without quotes):

```
voclabs:~/environment/containers/mysql $ docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST=172.17.0.3 node_app
be7bd40bc71633d93132628c65dafb7a6b10698abaff5b0f526a37366cdadcb3
voclabs:~/environment/containers/mysql $
```

32. To verify that both containers are running again, run the following command:

```
voclabs:~/environment/containers/mysql $ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS                                                  NAMES
be7bd40bc716   node_app       "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp              node_app_1
ab0c3f23cc5b   mysql_server   "docker-entrypoint.s…"   14 minutes ago       Up 14 minutes       0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql_1
voclabs:~/environment/containers/mysql $
```

Task 6: Adding the Docker images to Amazon ECR

In this final task in the lab, you will add the Docker images that you created to an Amazon Elastic Container Registry (Amazon ECR) repository.

33. Authorize your Docker client for Amazon ECR access by retrieving your AWS account ID.

- In the AWS Management Console, find your AWS account ID under your user name, starting with "voclab/user."

- Return to the AWS Cloud9 terminal and execute the following command, replacing **<account-id>** with your actual AWS account ID:

```
voclabs:~/environment/containers/mysql $ aws ecr get-login-password \
> --region us-east-1 | docker login --username AWS \
> --password-stdin 058264116330.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
voclabs:~/environment/containers/mysql $
```

34. To create the repository, run the following command:

```
voclabs:~/environment/containers/mysql $ aws ecr create-repository --repository-name node-app
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:058264116330:repository/node-app",
        "registryId": "058264116330",
        "repositoryName": "node-app",
        "repositoryUri": "058264116330.dkr.ecr.us-east-1.amazonaws.com/node-app",
        "createdAt": "2024-04-20T02:43:46.209000+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
voclabs:~/environment/containers/mysql $
```

35. Tag the Docker image.

In this step, you will tag the image with your unique **registryId** value to make it easier to manage and keep track of this image.

- Run the following command. Replace <registry-id> with your actual registry ID number.

```
voclabs:~/environment/containers/mysql $ docker tag node_app:latest 058264116330.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
```

- To verify that the tag was applied, run the following command:

```
voclabs:~/environment/containers/mysql $ docker images
REPOSITORY                                                TAG         IMAGE ID       CREATED          SIZE
mysql_server                                              latest      b55e8e7316ad   28 minutes ago   546MB
058264116330.dkr.ecr.us-east-1.amazonaws.com/node-app     latest      b4f56575b062   2 hours ago      84.5MB
node_app                                                  latest      b4f56575b062   2 hours ago      84.5MB
mysql                                                     8.0.23      cbe8815cbea8   3 years ago      546MB
node                                                      11-alpine   f18da2f58c3d   4 years ago      75.5MB
voclabs:~/environment/containers/mysql $
```

36. Push the Docker image to the Amazon ECR repository.

- To push your image to Amazon ECR, run the following command. Replace <registry-id> with your actual registry ID number:

```
voclabs:~/environment/containers/mysql $ docker push 058264116330.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
The push refers to repository [058264116330.dkr.ecr.us-east-1.amazonaws.com/node-app]
4a4a159d8b0c: Pushing [==================================================>]  9.233MB
389adcd08c49: Pushed
9a07659cfd9d: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushing [=========>                                         ]  12.77MB/64.86MB
f1b5933fe4b5: Pushing [=========================================>         ]  4.589MB/5.533MB
```

37. To confirm that the **node-app** image is now stored in Amazon ECR, run the following aws ecr list-images command:

```
voclabs:~/environment/containers/mysql $ aws ecr list-images --repository-name node-app
{
    "imageIds": [
        {
            "imageDigest": "sha256:79b3cb7ed168a99d3d22e966f4f69e1c9baccdb4edb3bbf5d5dde38d2fea6114",
            "imageTag": "latest"
        }
    ]
}
```

Update from the café

Sofía successfully containerized both the web application and backend database of the coffee supplier company acquired by the café owners. She registered the Docker image in Amazon ECR for future deployment. In the next lab, she plans to deploy the application using AWS Elastic Beanstalk. While Sofía containerized the MySQL database, she opted not to push it to Amazon ECR, considering AWS RDS as a better option for database hosting. Sofía is pleased with her progress and looks forward to deploying the application in the upcoming lab.