Lab 8.2: Running Containers on a Managed Service

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**

Duration

This lab will require approximately 90 minutes to complete.

AWS service restrictions

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

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.

When you *start* the lab and complete Task 1, your architecture will look like the following diagram.

By the *end* of this lab, you will have deployed the website code to Elastic Beanstalk and migrated the data to Aurora Serverless. Your architecture will then look like the following diagram.

Accessing the AWS Management Console

1. **At the top of these instructions, choose Start Lab to launch your lab.**

A Start Lab panel opens, and it displays the lab status.

Tip: If you need more time to complete the lab, choose the Start Lab button again to restart the timer for the environment.

2. **Wait until you see the message *Lab status: ready*, then close the Start Lab panel by choosing the X.**

3. **At the top of these instructions, choose AWS.**

This opens the AWS Management Console in a new browser tab. The system will automatically log you in.

Tip: If a new browser tab does not open, a banner or icon is usually at the top of your browser with a message that your browser is preventing the site from opening pop-up windows. Choose the banner or icon and then choose Allow pop ups.

4. **Arrange the AWS Management Console tab so that it displays along side these instructions. Ideally, you will be able to see both browser tabs at the same time so that you can follow the lab steps more easily.**

Tip: If you want the lab instructions to display across the entire browser window, you can hide the terminal in the browser panel. In the top-right area, clear the Terminal checkbox.

Task 1: Preparing the development environment

In this first task, you will configure your AWS Cloud9 environment so that you can use Python and the AWS Command Line Interface (AWS CLI) to interact with AWS services.

5. **In the AWS Management Console, connect to the AWS Cloud9 integrated development environment (IDE) named *Cloud9 Instance*.**
6. **Download and extract the files that you need for this lab.**
    - **In the same terminal, run the following command:**

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

    - **Notice that a code.zip file was downloaded to the AWS Cloud9 instance. The file is listed in the left navigation pane.**
    - **To extract the files, run the following command:**

unzip code.zip

7.
8. **Run a script to upgrade the version of Python and the AWS CLI that are installed on the Cloud9 instance. The script will also recreate the work that you completed in earlier labs into this AWS account.**

Note: This script populates the Amazon Simple Storage Service (Amazon S3) bucket with the café website code and configures the bucket policy as you did in the Amazon S3 lab. The script also creates an Amazon DynamoDB table and populates it with data as you did in the DynamoDB lab. The script recreates the REST API that you created in the API Gateway lab. Finally, the script pushes the Docker containers for the coffee suppliers application to Amazon ECR.

    - **Set permissions on the script so that you can run it, and then run the script:**

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

    - **When prompted for an IP address, enter the IPv4 address that the internet uses to contact your computer. You can find your IPv4 address at https://whatismyipaddress.com.**

Note: Keep in mind if a VPN is in use, or this is behind a proxy or other complicated network infrastructure this may not work correctly. *Finding your public IP behind a VPN or proxy is outside the scope of these instructions.*

Note: The IPv4 address that you set is the one that will be used in the bucket policy. Only requests that originate from the IPv4 address you identify will be allowed to load the website pages. Do not set the address to 0.0.0.0 because the S3 bucket's block public access settings will prevent access.

Note the metadata settings on the objects stored in the S3 bucket, and then verify that you can access the café website.

8. **Verify the version of AWS CLI and the version of SDK for Python that are installed.**
    - **In the AWS Cloud9 Bash terminal (at the bottom of the IDE), run the following command:**

aws --version

The output should indicate that version 2 is installed.

    - **Verify that the SDK for Python is installed:**

pip show boto3

Note: If you see a message about not using the latest version of pip, ignore the message.

9. **Confirm access to the café website.**
    - **Navigate to the Amazon S3 console.**
    - **Choose the link for the bucket that has *-s3bucket* in the name.**
    - **Choose the Properties tab, then scroll down to the Static website hosting section.**
    - **Choose the URL that appears under Bucket website endpoint.**

The café website displays in a new browser tab. If it doesn't, see the following troubleshooting tip.

Troubleshooting tip: If you are using a VPN, the IP address returned by whatismyipaddress.com might not allow access to the café website. If you can't connect to the café website, disconnect from VPN. Find your IP address again using whatismyipaddress.com, and then run setup.sh again. Then, stay off of the VPN for the duration of this lab.

10. **Open your preferred text editor, and copy and paste the following text into a new file.**

Cloud9 VPC ID:

Cloud9 Availability Zone:

Cloud9 subnet ID:

extraSubnetForRds subnet ID:

Cloud9 security group ID:

Database endpoint:

Repository URI:

Elastic Beanstalk URL:

As you progress through the lab steps, you will save the IDs of several resources. You will use this information later when you deploy your application to Elastic Beanstalk.

Task 2: Configuring the subnets for Amazon RDS and Elastic Beanstalk to use

Sofía knows that she needs at least two subnets in her virtual private cloud (VPC) to use an Amazon RDS database. She also needs two subnets to deploy the application using Elastic Beanstalk if she wants to be able to use cross-zone load balancing. She decides to use the same subnets for both the Aurora Serverless database and the Elastic Beanstalk EC2 instances.

Even if only a single application instance and database instance will be used initially, more than one subnet is required for this configuration. Further, each subnet used in an Amazon RDS or load-balanced Elastic Beanstalk deployment must be in a different Availability Zone.

When this lab environment launched, a VPC was created, but it only has one subnet. In this task, you will review the existing subnet and create a second one. You will also ensure that both subnets have a route to the public internet.

11. **Navigate to the Amazon VPC console.**
    - **Return to the AWS Management Console browser tab.**
    - **From the Services menu, choose VPC.**
    - **In the left navigation pane, choose Your VPCs.**
    - **Select the checkbox for *Cloud9 VPC*, and then copy the VPC ID into your text editor.**
12. **Review the Availability Zone for the *Cloud9 subnet*.**
    - **In the left navigation pane, choose Subnets.**
    - **Select the checkbox for *Cloud9 subnet*.**
    - **In the bottom pane, locate the Availability Zone and copy the value into your text editor.**

The Availability Zone looks similar to the following: *us-east-1a*

- **Copy the Subnet ID value into your text editor as the Cloud9 subnet ID.**

13. **Create a second subnet to provide high availability for the database and application in the future:**
    - **Choose Create subnet and configure the following:**
      - **VPC ID: Choose Cloud9 VPC**
    - **Subnet name: Enter extraSubnetForRds**
      - **Availability Zone: Choose a different Availability Zone than the one that the *Cloud9 subnet* is using**
    - **IPv4 CIDR block: Enter 10.16.2.0/24**

- Choose Create subnet.

- Configure the subnet to automatically assign a public IP address:
  - Select the checkbox for *extraSubnetForRds*.
  - From the Actions menu, select Edit subnet settings.
  - Select the checkbox for Enable auto-assign public IPv4 address.
  - Choose Save.
- In a text editor, record the Subnet ID as the extraSubnetForRds subnet ID.

14. Update the route table for the *extraSubnetForRds* subnet.
    - Select the checkbox for *extraSubnetForRds*.
    - In the bottom pane, choose the Route table tab, and review the route table.

Notice that this route table does not include a path to the public internet (0.0.0.0/0). For this configuration to work, you need to update the route table association for this subnet.

- Choose Edit route table association.

The subnet is currently configured to use the *Main route table*.

- For Route table ID, choose the other route table.
- Choose Save then choose the Route table tab.

In the Routes section, you now find two entries that are similar to the following:

Destination	Target
10.16.0.0/16	local
0.0.0.0/0	igw-xxxxxxxxx

15. Because Elastic Beanstalk must be able to reach the public internet, the route table for the *Cloud9 subnet* and *extraSubnetForRds* must include a route to 0.0.0.0/0 using the Internet Gateway (igw-xxxxxxxxxx). Elastic Beanstalk will be able to create an EC2 instance in either of these subnets. This route will allow the instance to reach the outside world regardless of the subnet that Elastic Beanstalk uses to deploy the application.

Next, you will create a new database that the coffee suppliers application will use.

Task 3: Setting up an Aurora Serverless database

Instead of running the database on an EC2 instance or in a Docker container, the application will use the Aurora Serverless managed service as the data platform.

In this task, you will create a new Aurora Serverless instance.

15. **Create an Aurora Serverless database instance.**
    - **In the search box to the right of Services, search for and choose RDS.**
    - **In the Create database panel under the Resources panel, choose Create database.**
        - **In the Choose a database creation method section, choose Standard create.**
        - **In the Engine options section, configure the following:**
        - **Engine type: Choose Aurora(MySQL-Compatible)**
        - **Version: Choose Aurora MySQL 3.07.0 (compatible with MySQL 8.0.36)**
        - **In the Templates section, choose Dev/Test.**
        - **In the Settings section, configure the following:**
            - **DB cluster identifier: Enter supplierdb**
        - **Under Credentials management, choose Self managed.**
        - **Ensure that Auto generate a password is NOT checked**
        - **Master password: Enter coffee_beans_for_all**

📓 **Note: This is the password for the database super user that is named *admin*. You will use this password later in this assignment.**

- **Confirm password: Re-enter the password**
- **In the Instance configuration section, configure the following:**
    - **DB instance class: Choose Serverless v2**
- **Also note the default Capacity range settings (Minimum 2 ACU, Maximum 16 ACU) but do not change them.**
- **In the Connectivity section, configure the following:**
    - **Virtual Private Cloud (VPC): Choose Cloud9 VPC**
    - **Under VPC security group (firewall), choose Existing VPC security groups**
    - **Remove the default security group**
    - **From the Dropdown, add the security group with Cloud9-Instance in the name**
- **Under RDS Data API, Select Checkbox for *Enable the RDS Data API*.**
- **Under Monitoring > Performance Insights**
    - **UnCheck/DeSelect *Enable Performance Insights (cluster level)***

- **Expand Additional configuration (Enhanced Monitoring)**
    - **Uncheck/DeSelect *Enable Enhanced Monitoring***
- **Expand Additional configuration**
- **For Initial database name, enter suppliers**
- **Keep rest of the parameters at their default values and Choose Create database.**

Note: If you see a pop-up similar to *Suggested add-ons for supplierDB*, choose Close.

Note: Wait for the database to get created, it will take a few minutes.

16. **Find the endpoint for your database.**
    - **Choose the supplierdb hyperlink displayed at the top.**
    - **From the DB identifier , select supplierdb-instance-1.**
        - **Note: Wait for the Status to become *Available*.**
    - **On the In the bottom pane, in the Connectivity & security section**
        - **Copy the Endpoint name value to your text editor.**

The endpoint is similar to *supplierdb.cluster-xxxxxxxxxxx.us-east-1.rds.amazonaws.com*.

17. **Find the ID for the database security group.**
    - **In the Security section, choose the VPC security groups hyperlink.**
    - **On the Security Groups page, in the bottom pane, on the Details tab, copy the Security group ID value to your text editor.**

The security group ID is similar to *sg-123456acbde*.

While the database is still being created, move to the next task and review your container image.

Task 4: Reviewing the container image

In a previous lab, you created a Docker image of the coffee suppliers application, which was written in Node.js. You then uploaded the node application image to Amazon ECR.

In this task, you will review the details of the image that was uploaded to Amazon ECR.

18. **Review the Docker image in the console.**
    - **In the search box to the right of Services, search for and choose Elastic Container Registry.**

- **For the cafe/node-web-app repository, copy the URI value to your text editor as the Repository URI.**
- **Choose the cafe/node-web-app hyperlink, and review the details.**

19. **Review this same Docker image information in the AWS Cloud9 terminal.**
    - **Return to the AWS Cloud9 browser tab.**
    - **Run the following command:**

aws ecr describe-repositories

The output is similar to the following:


```
{
  "repositories": [
    {
      "repositoryArn": "arn:aws:ecr:us-east-1:10101010101:repository/cafe/node-web-app",
      "registryId": "10101010101",
      "repositoryName": "cafe/node-web-app",
      "repositoryUri": "10101010101.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app",
      "createdAt": "2021-05-18T15:47:50+00:00",
      "imageTagMutability": "MUTABLE",
      "imageScanningConfiguration": {
        "scanOnPush": false
      },
      "encryptionConfiguration": {
        "encryptionType": "AES256"
      }
    }
  ]
}
```


📓 **Note: Your repositoryArn, repositoryUri, and createdAt timestamps will be different from this example output.**

20.

21. **Next, to inspect the *cafe/node-web-app* image, run the following command in the AWS Cloud9 terminal:**

aws ecr describe-images --repository-name cafe/node-web-app

The output is similar to the following:


```
{
   "imageDetails": [
      {
         "registryId": "455065222927",
         "repositoryName": "cafe/node-web-app",
         "imageDigest":
"sha256:f73f6be26ca663a96885f91e2dc1cab873d0296c3bfa6ec0e03709fa9dea7a11",
         "imageTags": [
            "latest"
         ],
         "imageSizeInBytes": 27708083,
         "imagePushedAt": "2021-05-18T16:49:03+00:00",
         "imageManifestMediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "artifactMediaType": "application/vnd.docker.container.image.v1+json"
      }
   ]
}
```


Now you know how to find details about a container hosted in Amazon ECR using the console or the AWS CLI.

Task 5: Configuring communication between the container and the database

Now it's time to test connectivity between the containerized application and the Aurora Serverless database. Initially, you will test the container connectivity from your AWS Cloud9 environment.

In Task 1, the setup.sh script set up the Docker image in your AWS Cloud9 environment. You do not need to pull the image from Amazon ECR again.

In a previous lab, you used a command similar to the following to start your container:

docker run -d --name node-web-app-1 -p 3000:3000 -e APP_DB_HOST="<ip-address>" cafe/node-web-app

This time, instead of using the IP address of a database host, you will set the value for APP_DB_HOST to the database endpoint that you saved in a previous task.

21. **Start your container.**
    - **In the AWS Cloud9 terminal, enter the following command. Replace <db-endpoint> with the Database endpoint value from your text editor.**

docker run -d --name node-web-app-1 -p 80:3000 -e APP_DB_HOST="<db-endpoint>" cafe/node-web-app

The updated command looks similar to the following:

docker run -d --name node-web-app-1 -p 80:3000 -e APP_DB_HOST="supplierdb.cluster-cltkajxxxxxx.us-east-1.rds.amazonaws.com" cafe/node-web-app

    - **Run the updated command.**

The output is similar to the following:

ab3a49274fa2f154d583cd6f5cf934d03229647644699a3e38c53f5678df0246

22. **Note: Notice that the *docker run* command mapped your container host port 80 to container port 3000. When you deploy with Elastic Beanstalk later in the lab, a Classic Load Balancer will map traffic and health checks to host port 80.**
23.
24. **To test the container application from the AWS Cloud9 terminal, run the following command:**

curl http://localhost:80

The response is similar to the following:

```
<!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>

<div class="container">
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <img src="/img/espresso.jpg" width="200">
    <div><a class="navbar-brand page-title" href="/supplier">Coffee suppliers</a></div>
```

```
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
          <li class="nav-item active">
            <a class="nav-link" href="/">Home</a>
            <a class="nav-link" href="/suppliers">Suppliers list</a>
          </li>
        </ul>
      </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>
</html>
```

23. **Configure the security group of the AWS Cloud9 EC2 instance.**

As in a previous lab, you need to add a rule so that you can view the website on port 80:

- **Return to the AWS Management Console browser tab.**
- **In the search box to the right of Services, search for and choose EC2.**
- **In the left navigation pane, choose Security Groups.**
- **Select the checkbox for the security group that has *Cloud9-Instance* in the name.**
- **In the lower pane, choose the Inbound rules tab.**
- **Choose Edit inbound rules.**
- **Add an entry for the application port:**
  - **Choose Add rule**
  - **Type: Choose HTTP**
  - **Source: Choose My IP**

For the application to communicate with the database, add another rule that allows inbound communication on port 3306.

- **Add an entry for the database port:**
  - **Choose Add rule**
  - **Type: Choose MYSQL/Aurora**
  - **Source: Choose Custom**
  - **From the Source search box, choose the *Cloud9-Instance* security group that you are currently editing.**

This rule is self referencing to allow any call that originates from this security group to communicate with the database.

- **Choose Save rules.**

Now, your application can connect to your Amazon RDS instance from AWS Cloud9 as you test your work.

24. **Locate the public IP address of your AWS Cloud9 instance.**
    - **In the left navigation pane, choose Instances.**
    - **Choose the Cloud9-Instance.**
    - **In the bottom pane, choose the Details tab, and copy the Public IPv4 address value to your clipboard.**
25. **In a new browser tab, paste the Public IPv4 address in the address bar.**

The coffee suppliers web application displays.

26. **Choose List of suppliers.**

The following error message displays on the page:

Although you have set up the container and Amazon RDS networking successfully, the error occurs because additional setup is needed in the database itself. Keep this browser tab open; you will use it again in the next task.

In the next task, you will finish setting up the Amazon RDS instance.

Task 6: Creating the application database objects

The coffee suppliers application expects to communicate with a database named *COFFEE*, a database user named *nodeapp*, and a database table named *suppliers*. None of these database objects have been created in the *supplierdb* cluster yet. That is why the application is returning an error message.

In this task, you will the create database objects that the application requires.

27. **Connect to the Amazon RDS query editor.**
    - **Return to the AWS Management Console browser tab.**
    - **In the search box to the right of Services, search for and choose RDS.**
    - **In the left navigation pane, choose Query Editor, and configure the following:**
        - **Database instance or cluster: Choose supplierdb**
        - **Database username: Choose Add new database credentials**
        - **Enter database username: Enter admin**
        - **Enter database password: Enter coffee_beans_for_all**
        - **Enter the name of the database or schema: Enter suppliers**
        - **Choose Connect to database.**

📓 **Note: If you receive an error message, choose Connect to database again and the connection should be successful.**

28. **Create the database objects that support the coffee suppliers application.**
    - **On the Editor tab, delete the contents of the text area, and paste in the following code:**

```
CREATE USER "nodeapp" IDENTIFIED WITH mysql_native_password BY "coffee";

CREATE DATABASE COFFEE;

USE COFFEE;

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;

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));

- **Choose Run.**

The Output pane looks similar to the following:

29. **Verify that the *suppliers* table is empty.**
    - **Replace the contents in the text area with the following code:**

use COFFEE;

select * from suppliers

- **Choose Run.**

The Result set pane looks similar to the following, which indicates that the table is empty.

- **Keep this browser tab open on the Query Editor page.**

30. **Try again to access the supplier information from the web application.**
    - **Return to the browser tab that is connected to the coffee suppliers application.**
    - **Refresh the page.**

Instead of an error message, the page displays an empty suppliers table and an Add a new supplier button.

31.

32. **Choose Add a new supplier and add a new item to the table.**

33. **View the record in the query editor.**
    - **Return to the Amazon RDS query editor browser tab.**
    - **To run the query again, choose Run.**

The Result set pane displays the record that you added using the web application.

Congratulations! You have learned how to set up a new Aurora Serverless instance. You have also successfully tested the node application locally on AWS Cloud9.

Now, you need to load the supplier data into the database and deploy the node application to EC2 instances using Elastic Beanstalk.

Task 7: Seeding the database with supplier data

The supplier has provided Sofía with the latest SQL dump from their supplier database. This data needs to be loaded into the Aurora Serverless instance.

In this task, you will load the supplier data into the *COFFEE* database.

33. **Review the supplier data.**
    - **Return to the AWS Cloud9 browser tab.**
    - **To change the directory to resources, run the following command:**

cd ~/environment/resources

    - **From the Environment pane in Cloud9, open the *resources/coffee_db_dump.sql* file and review its contents.**

34. **Connect to the database.**
    - **Run the following command in the terminal window. Replace <db-endpoint> with the Database endpoint value from your text editor.**

mysql -h <db-endpoint> -P 3306 -u admin -p

    - **When prompted for the password, enter coffee_beans_for_all**

The output looks similar to the following:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 11

Server version: 5.7.12 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

- **To verify that you are connected to the correct database, run the following command:**

use COFFEE; select * from suppliers;

The command returns the record or records that you added through the web application. The output looks similar to the following:

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

mysql> select * from suppliers

-> ;

Database changed

```
+----+-----------+----------------+---------+-------+------------------+-----------+
| id | name      | address        | city    | state | email            | phone     |
+----+-----------+----------------+---------+-------+------------------+-----------+
|  1 | Sofia     | 1234 Some Lane | Anytown | NV    | sofia@example.com | 123456789 |
+----+-----------+----------------+---------+-------+------------------+-----------+
```

1 row in set (0.00 sec)

35. **To use the SQL dump to update your database with the information from the supplier, run the following command at the mysql prompt:**

source coffee_db_dump.sql

The output displays several lines similar to the following:

..

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

..truncated

36. **Review the data that was loaded into the database.**

- **To query the *suppliers* table, run the following command at the mysql prompt:**

use COFFEE; select * from suppliers;

The query returns the following records:

```
+----+--------------------------------------+-----------------+-----------+-------+----------------------+-------------+
| 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 |
+----+--------------------------------------+-----------------+-----------+-------+----------------------+-------------+
```

The supplier also provided coffee bean information in a table called *beans*. You will use this data on the café website too.

- **To query the *beans* table, run the following command at the mysql prompt:**

select * from beans;

The query returns the following records:

```
+----+-------------+----------+----------------+-------+-----------------------------------------------------+----------+
| 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 |

+----+-------------+----------+----------------+-------+---+----------+

- **To return to the terminal shell, type exit**

37. **View the updated supplier information from the application webpage.**
    - **Return to the browser tab that is connected to the coffee suppliers application.**
    - **Refresh the page.**

The suppliers are now listed on the page, which looks similar to the following:

Your database is now up to date with the most recent information from the supplier, and you are ready to use Elastic Beanstalk to deploy the application.

Elastic Beanstalk

So far, Sofía has been deploying the web application manually. The data layer has been migrated to a reliable data platform, and application functionality with the new database has been successfully tested. She is now ready to deploy the node container using Elastic Beanstalk.

For now, she will use Elastic Beanstalk to deploy the container on a single Amazon EC2 instance. For this deployment to work, she needs a policy that allows the EC2 instance created by Elastic Beanstalk to communicate with Amazon ECR. Elastic Beanstalk will also automatically deploy a Classic Load Balancer that maps requests to port 80 of the EC2 instance that hosts the container. In the future, this environment can be scaled to meet customer demand.

First, you need to create an AWS Identity and Access Management (IAM) policy and an associated role. This extra step is required because you are using a Docker environment.

Task 8: Review the IAM policy and role for Elastic Beanstalk

When Elastic Beanstalk deploys the application, it will launch a new EC2 instance. Then, the EC2 instance will pull the Docker image for the coffee suppliers application from Amazon ECR and start the container. You must use an IAM policy and associated role to give the EC2 instance permission to get images from Amazon ECR.

In this task, you will create the IAM policy and role that EC2 needs to access Amazon ECR.

38. **Review the Elastic Beanstalk IAM policy that will be used with your Docker environment.**
    - **Return to the AWS Management Console browser tab.**
    - **From the Services menu, choose IAM.**
    - **In the left navigation pane, choose Policies.**
    - **In the search text box, enter aws-elasticbeanstalk-ec2-instance-policy then select the hyperlink of the policy.**
    - **In the Permissions tab, choose the JSON option and review the policy statement:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "AllowEbAuth"
```

```
    },
    {
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "AllowPull"
    },
    {
      "Effect": "Allow",
      "Action": [
        elasticbeanstalk:Put*
      ],
      "Resource": "*"
    }
  ]
}
```

Notice that the Sid named *AllowEbAuth* allows the EC2 instances to retrieve the authorization token. The other Sid, *AllowPull*, allows the EC2 instances to download the docker image from the repository. The last statement in the policy allows the Elastic Beanstalk service to report instance statistics to the beanstalk application.

39. **Review the role that will be used by the Elastic Beanstalk Docker environment.**
    - **In the left navigation pane, choose Roles.**
    - **In the search text box, enter aws-elasticbeanstalk-ec2-role.**
    - **In the Trust relationships tab, notice that the EC2 service, *ec2.amazonaws.com*, is defined as a Trusted entity.**
    - **In the Permissions tab, locate the policy that is associated with the Role.**

You will find that this role is using the *aws-elasticbeanstalk-ec2-instance-policy* policy you just examined.

Elastic Beanstalk will use this role for all of the instances that it launches. Initially, you will start with just one instance.

Task 9: Creating an Elastic Beanstalk application

With the necessary permissions, networking configuration, and services in place, it's time to deploy the Elastic Beanstalk application. First, you will deploy a sample application. Then, you will update the application to use the coffee suppliers website image.

40. **Create a sample Elastic Beanstalk application.**
    - **Return to the AWS Cloud9 browser tab.**
    - **To change to the *environment* directory, run the following command:**

```
cd ~/environment
```

    - **To create a new folder called *bean*, run the following command:**

```
mkdir bean
```

    - **To change to the *bean* directory, run the following command:**

```
cd bean
```

    - **To create a sample application named *MyNodeApp*, run the following command:**

```
aws elasticbeanstalk create-application --application-name MyNodeApp
```

The output from the previous command looks similar to the following:

```
{
  "Application": {
    "ApplicationArn": "arn:aws:elasticbeanstalk:us-east-
1:455065222927:application/MyNodeApp",
    "ApplicationName": "MyNodeApp",
    "DateCreated": "2021-05-19T17:45:40.748Z",
    "DateUpdated": "2021-05-19T17:45:40.748Z",
    "ConfigurationTemplates": [],
    "ResourceLifecycleConfig": {
      "VersionLifecycleConfig": {
        "MaxCountRule": {
          "Enabled": false,
```

```
            "MaxCount": 200,
            "DeleteSourceFromS3": false
          },
          "MaxAgeRule": {
            "Enabled": false,
            "MaxAgeInDays": 180,
            "DeleteSourceFromS3": false
          }
        }
      }
    }
}
```

41. **Create an Elastic Beanstalk environment.**
    - **In AWS Cloud9, in the *bean* folder, create a new file named *options.txt* and open it.**
    - **Paste the following text into the *options.txt* file:**

```
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "<FMI_1>"
  },
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "VPCId",
    "Value": "<FMI_2>"
```

```
  },
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "Subnets",
    "Value": "<FMI_3>,<FMI_4>"
  },
  {
    "Namespace": "aws:elasticbeanstalk:application:environment",
    "OptionName": "APP_DB_HOST",
    "Value": "<FMI_5>"
  }
]
```

- In the *options.txt* file, replace the placeholders with the values that you saved in your text editor as follows:
  - <FMI_1>: Cloud9 security group ID
  - <FMI_2>: Cloud9 VPC ID
  - <FMI_3>: Cloud9 subnet ID
  - <FMI_4>: extraSubnetforRds subnet ID
  - <FMI_5>: Database endpoint

The updated file will look similar to the following. Note that your values will be different.

```
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-0e52adxxxxxxxx"
  },
```

```
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "VPCId",
    "Value": "vpc-09d7xxxxxxxxx"
  },
  {
    "Namespace": "aws:ec2:vpc",
    "OptionName": "Subnets",
    "Value": "subnet-005ca7bedxxxxxxx,subnet-04a1eec73xxxxxxx"
  },
  {
    "Namespace": "aws:elasticbeanstalk:application:environment",
    "OptionName": "APP_DB_HOST",
    "Value": "supplierdb.cluster-cltkaxxxxxxx.us-east-1.rds.amazonaws.com"
  }
]
```

- Save the file.
- To identify the currently available solution stack for Amazon Linux 2, run the following command in the AWS Cloud9 terminal:

```
aws elasticbeanstalk list-available-solution-stacks|grep 'running Docker'
```

The output contains the solution stack name and looks similar to the following. Note that the version number might be higher:

```
64bit Amazon Linux 2 v4.0.5 running Docker
```

Tip: Use the result that runs on Amazon Linux 2.

- In the AWS Cloud9 terminal, make sure you are in the *bean* directory.
- Run the following command. Replace the <solution-stack-name> placeholder with the name of the available solution stack:

```
aws elasticbeanstalk create-environment --application-name MyNodeApp --environment-name MyEnv --solution-stack-name "<solution-stack-name>"  --region us-east-1 --option-settings file://options.txt
```

The updated command looks similar to the following:

aws elasticbeanstalk create-environment --application-name MyNodeApp --environment-name MyEnv --solution-stack-name "64bit Amazon Linux 2 v4.0.5 running Docker" --region us-east-1 --option-settings file://options.txt

The output looks similar to the following:

{

"EnvironmentName": "MyEnv",

"EnvironmentId": "e-b966fxxxxx",

"ApplicationName": "MyNodeApp",

"SolutionStackName": "64bit Amazon Linux 2 v3.6.4 running Docker",

"PlatformArn": "arn:aws:elasticbeanstalk:us-east-1::platform/Docker running on 64bit Amazon Linux 2/3.6.4",

"DateCreated": "2023-12-01T19:44:17.565000+00:00",

"DateUpdated": "2023-12-01T19:44:17.565000+00:00",

"Status": "Launching",

"Health": "Grey",

"Tier": {

"Name": "WebServer",

"Type": "Standard",

"Version": "1.0"

},

"EnvironmentArn": "arn:aws:elasticbeanstalk:us-east-1:13557xxxxxxxx:environment/MyNodeApp/MyEnv"

}

Tip: If necessary, type q to return to the command prompt.

42. **To stop the Docker container that is running on the AWS Cloud9 instance, run the following command:**

docker stop node-web-app-1 && docker rm node-web-app-1

43. **Review your Elastic Beanstalk environment in the AWS Management Console.**
    - **Return to the AWS Management Console tab.**
    - **From the Services menu, choose Elastic Beanstalk.**
    - **For the MyEnv environment, the Health displays *Pending*.**

- **At the bottom under the Events tab.**

The environment creation will take several minutes, but it's interesting to watch what is happening. The following screenshot provides an example of what the console displays:

44.

- **At the top navigation area, under MyEnv you can see the status of the environment.**
- **Wait for the environment to be ready before moving to the next step. When the environment is ready, Health displays *Ok* under an image of a check mark, as shown in the following screenshot:**

45.

46. Your Elastic Beanstalk application is now up and running. Try it out!

44. Test the sample Elastic Beanstalk application.

- **At the top of the page, locate the domain for this environment.**

The domain looks similar to the following: *MyEnv.eba-xxxxxxx.us-east-1.elasticbeanstalk.com*

- **Choose the domain hyperlink to open it in a new browser tab.**

The sample application displays and looks like the following:

45.

46. You have deployed the sample application, and all of the required scaffolding is in place. Now, you only need to tell Elastic Beanstalk to use your code.

45. Update the application with the code for the coffee suppliers application.

The code is in the image that you uploaded to Amazon ECR. For now, you will deploy the application to Elastic Beanstalk manually. In a later lab, you will learn to automate deployments.

- **On your local computer, create a new text file called *Dockerrun.aws.json*.**
- **Paste the following text into the file:**

{

"AWSEBDockerrunVersion": "1",

"Image": {

"Name": "<FMI_1>",

"Update": "true"

},

"Ports": [{ "ContainerPort" : 3000 }]

}

Note: The configuration file uses port 3000 instead of port 80. Elastic Beanstalk will proxy port 80 to your container port 3000 on single docker container environments such as this one.

- **In the file, replace the <FMI_1> placeholder with the Repository URI value from your text editor.**
- **Save the changes to the file.**
- **Return to the AWS Management Console browser tab.**
- **On the MyEnv environment page within the Elastic Beanstalk console, choose Upload and deploy.**
- **Navigate to and choose the *Dockerrun.aws.json* file.**
- **For Version label, append the letter a to the default MyNodeApp-version-1**
- **Keep the current deployment preferences.**
- **Choose Deploy and observe the deployment process.**

If you want to follow what is happening, observe the Recent events section.

After a few minutes, the page updates and looks like the following:

46. **Test the Elastic Beanstalk deployment of the coffee suppliers application.**
    - **Copy the MyEnv domain URL to your text editor as the Elastic Beanstalk URL.**
    - **Choose the domain hyperlink to open it in a new browser tab.**

The coffee suppliers application opens.

- **Choose List of suppliers.**

The page displays the same list of suppliers that you loaded into the database earlier in the lab.

The application does not have a button or link to display the *beans* database information yet. However, to test the *beans* database query, you can update the application URL.

- **In the browser address bar, replace /suppliers in the URL with /beans.**

The updated URL looks similar to the following: *http://myenv.eba-xxxxxxxxx.us-east-1.elasticbeanstalk.com/beans*

The page displays the inventory details of the available coffee bean varieties.

- **In the browser address bar, append the URL with .json**

The updated URL looks similar to the following: *http://myenv.eba-xxxxxxxxx.us-east-1.elasticbeanstalk.com/beans.json*

The page now displays the same inventory details but in JSON format. You will use this URL in the API Gateway configuration because the café website code expects to receive data formatted in JSON.

Way to go! Using Elastic Beanstalk, you have launched the coffee suppliers application code in a container that is hosted on an EC2 instance. The EC2 instance is fronted by a Classic Load Balancer, and Auto Scaling is enabled. Even though you are only using one instance, the addition of Auto Scaling adds high availability to your application. If the EC2 instance that is running your container fails, a new instance will automatically be launched and configured to replace it. Using the Classic Load Balancer opens up additional options for future scaling and management of connections to the coffee suppliers application.

Next, you will configure API Gateway with a new resource that will call the coffee suppliers application.

Task 10: Configuring the API Gateway proxy

Well done! You now have a working Elastic Beanstalk environment. You can access supplier information and coffee bean inventory using the coffee suppliers application.

However, you need to be able to access the coffee bean inventory directly from the café website. Good news! You can leverage the URL that returns the beans data in JSON format (*beans.json*) to add this functionality.

47. **Find the Amazon S3 URL for the café website.**
    - **In the browser tab with these lab instructions, choose Details.**
    - **Next to AWS, choose Show.**
    - **Save the WebsiteURL value to your text editor.**

48. **Review the coffee suppliers integration with the application.**
    - **In a new browser tab, paste the WebsiteURL.**
    - **From the menu in the upper-left corner, choose Buy Coffee.**

Notice that the Buy Coffee section displays a message that this section is coming soon. Once you set up API Gateway to call the Elastic Beanstalk application, that will change.

49. **Create an API Gateway resource.**
    - **Return to the AWS Management Console browser tab.**
    - **From the Services menu, choose API Gateway.**
    - **Choose the ProductsApi hyperlink.**
    - **In the Resources pane, keep the top-level "/" resource selected, as shown in the following image:**

    - **Choose Create resource, and configure the following:**
        - **Resource Path: Keep the default / selection**
        - **Resource Name: Enter bean_products**

🤦♂ **Note: Ensure the resource name value has an underscore.**

        - **Select CORS (Cross Origin Resource Sharing)**

📓 **Note: CORS is required to enable communication between your website and the endpoint.**

        - **Choose Create resource.**

50. **Create an API Gateway method.**
    - **In the Resources pane, ensure that the /bean_products resource is selected, as shown in the following image:**

    - **In the Methods panel, choose Create method.**
    - **For Method type choose GET.**
    - **Configure the following:**
        - **Integration type: Choose HTTP**
        - **Toggle HTTP proxy integration to on**
        - **For HTTP method choose GET**
        - **For Endpoint URL, enter the following. Replace the <FMI_1> placeholder with the Elastic Beanstalk URL from your text editor:**

http://<FMI_1>/beans.json

The updated URL looks similar to the following: *http://MyEnv.eba-3xxxxxxxx.us-east-1.elasticbeanstalk.com/beans.json*

The following image shows the options that you should select on this page (your Endpoing URL will be different):

- **Choose Create method.**

- **Choose the Test tab.**
- **In the Test method panel, choose Test at the bottom of the page.**

The response output looks similar to the following:

51. **Deploy the API changes.**
    - **In the Resources pane, choose the top-level "/" resource.**
    - **Choose Deploy API.**
    - **For Stage, choose prod.**
    - **Choose Deploy.**

Note: If you see an error message that you do not have permissions for Web Application Firewall, ignore the message.

52. **Test the new API resource in the café application.**
    - **Return to the browser tab where you opened the WebsiteURL. Reload the page if you already had it open.**
    - **From the menu in the upper-left corner, choose Buy Coffee.**

This section of the café application now displays the coffee bean inventory information.

Great job! You have configured a new API Gateway endpoint that is used to integrate the coffee bean inventory from the coffee suppliers application with the customer-facing café website.

Update from the café

The café is thrilled! You have deployed the coffee suppliers application using managed services. You even deployed the application's database to a serverless platform, which will scale and shrink with utilization. This configuration will be able to grow with the café. In addition, you were able to integrate the coffee bean inventory with the main café website so that customers can see what's in stock.

Congratulations! Now you know how to manually deploy a single container, update code, and deploy an application using Amazon ECR, Amazon RDS, and Elastic Beanstalk.

Later in the course, you will learn to automate deployments, rather than using a manual process as outlined in this lab. You will use a full DevOps code pipeline to build the container on every change to the code.

Submitting your work

53. **At the top of these instructions, choose Submit to record your progress and when prompted, choose Yes.**

Tip: If you previously hid the terminal in the browser panel, expose it again by selecting the Terminal check box. This action will ensure that the lab instructions remain visible after you choose Submit.

54. **If the results don't display after a couple of minutes, return to the top of these instructions and choose Grades**

Tip: You can submit your work multiple times. After you change your work, choose Submit again. Your last submission is what will be recorded for this lab.

55. **To find detailed feedback on your work, choose Details followed by View Submission Report.**

Lab complete

Congratulations! You have completed the lab.

56. **Choose End Lab at the top of this page, and then select Yes to confirm that you want to end the lab.**

A panel indicates that *DELETE has been initiated... You may close this message box now.*

57. **Select the X in the top-right corner to close the panel.**