

## **Experiment 05: Creating Lambda Functions Using the AWS SDK for Python**

### **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** check box.

### **Task 1: Configuring the development environment**

In this first task, you will configure your Visual Studio Code Integrated Development Environment (VS Code IDE) so that you can create Lambda functions.

5. Connect to the VS Code IDE.
- o At the top of these instructions, choose **Details** followed by **AWS: Show**
  - o Copy values from the table **similar** to the following and paste it into an editor of your choice for use later.
    - **LabIDEURL**
    - **LabIDEPassword**
  - o In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
  - o On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.
6. Download and extract the files that you will need for this lab.
- o In the same VS Code IDE terminal, run the following command:

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/05-lab-lambda/code.zip -P /home/ec2-user/environment
```

- o The code.zip file is downloaded to the VS Code IDE. The file is listed in the left navigation pane.
- o Extract the file:

```
unzip code.zip
```

7. Run the script that re-creates 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 the DynamoDB table and populates it with data as you did in the DynamoDB lab. Finally, the script re-creates the REST API that you created in the API Gateway lab.

- o To set permissions on the script and then run it, run the following commands:

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

- o When prompted for an IP address, enter the IPv4 address that the internet uses to contact your computer. You can find this IP address from <https://whatismyipaddress.com>.

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

8. To verify the SDK for Python is installed, run the following command in the VS Code IDE terminal:

```
pip3 show boto3
```

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

9. Take a moment to see what resources the script created.
  - o Confirm that an S3 bucket is hosting the café website files:
    - In the *Your environments* browser tab, browse to the Amazon S3 console, and choose the name of the bucket that was created.
    - Choose **index.html** and copy the **Object URL**.
    - Load the URL in a new browser tab.

The café website displays. Currently, the website is accessing the hard-coded menu data that is stored in S3 to display the menu information.

**Tip:** Notice that several menu items are listed in the **Browse Pastries** section of the page.

- o Confirm that DynamoDB has the menu data stored in a table:
  - Browse to the DynamoDB console.
  - Choose **Tables** and choose the **FoodProducts** table.
  - Choose **Explore table items** and confirm that the table is populated with menu data.

- Choose **View table details** and then on the **Indexes** tab, confirm that an index named **special\_GSI** was created.
- o Confirm that the ProductsApi REST API was defined in API Gateway:
  - Browse to the API Gateway console.
  - Choose the name of the **ProductsApi** API.
  - The API has a **GET** method for **/products** and a **GET** method for **/products/on\_offer**.
  - Finally, the API has **POST** and **OPTIONS** methods for **/create\_report**.
  - From the lower pane, you can use the **TEST** menu and *Test* each method to ensure that they are returning the mock data that you used in the previous lab. Each method should return a 200 HTML status code.

10. Copy the invoke URL for the API to your clipboard.

- o In the API Gateway console, in the left panel, choose **Stages** and then choose the **prod** stage.

**Note:** If you see a warning that you do not have ListWebACLs and AssociateWebACL permissions, ignore the warning.

- o Copy the **Invoke URL** value that displays at the top of the page. You will use this value in the next step.

11. Update the website's config.js file.

- o In the VS Code IDE browser tab, open resources/website/**config.js**.
- o On line 2, replace null with the invoke URL value that you copied a moment ago. Also, be sure to surround the URL in double quotation marks.
- o The file now looks like the following example, but you will have a different value for `<some-value>`:

```

window.COFFEE_CONFIG = {
API_GW_BASE_URL_STR: "https://<some-value>.execute-api.us-east-
1.amazonaws.com/prod",
COGNITO_LOGIN_BASE_URL_STR: null
};

```

- o Verify that /prod appears at the end of the URL with no trailing slash.
- o Close the file by choosing **X** from the top. (Your changes are saved automatically).

12. Update and then run the update\_config.py script.

- o Open python\_3/**update\_config.py** in the text editor.

- o Replace the <FMI\_1> placeholder with the name of your S3 bucket.

**Tip:** Find the bucket name in the S3 console, or run the following command:

```
aws s3 ls
```

- o Notice that this script will upload the config.js file that you just edited to the S3 bucket.
- o Close the file by choosing **X** from the top. (Your changes are saved automatically).
- o To run the script, run the following commands:

```
cd ~/environment/python_3
python3 update_config.py
```

13. Load the latest café webpage with the developer console view exposed.

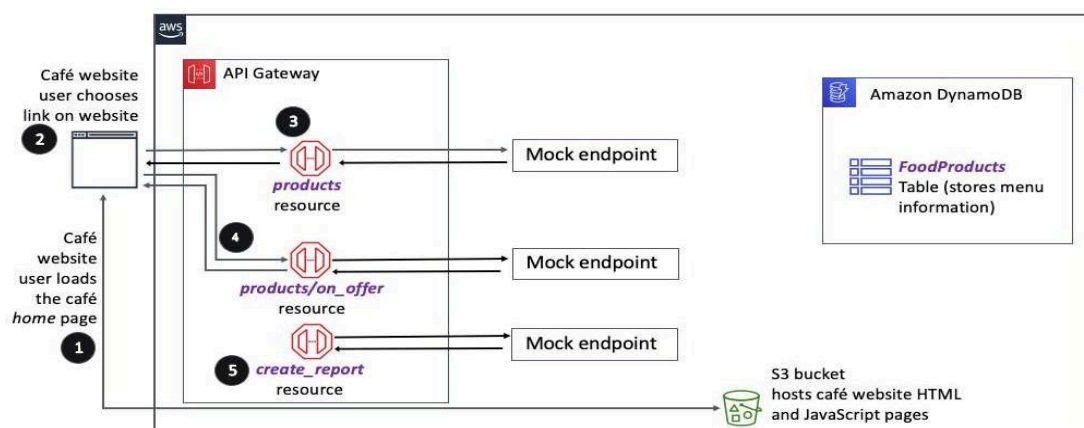
- o If you still have the café website open in a browser tab, return to it. If you do not have the website open, reopen it now by following these steps:
  - In the Amazon S3 console, choose the name of the bucket that contains your website files.
  - Choose **index.html** and then copy the **Object URL** value.
  - Load the object URL in a new browser tab.
- o Refresh the browser tab to load the

changes. The café website displays.

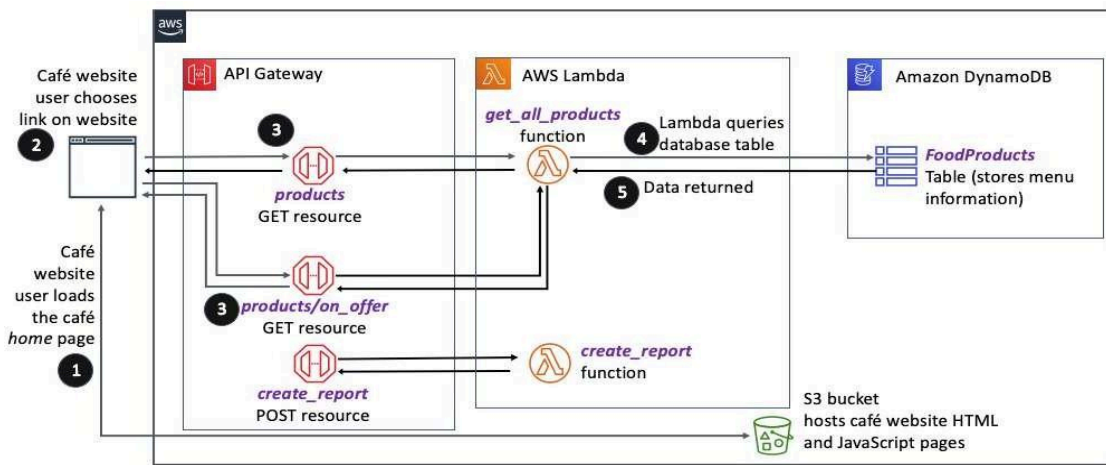
- o Observe the **Browse Pastries** section of the webpage. Notice that only one menu item now displays.

This indicates that you are no longer viewing hard-coded data. Instead, the website returns the mock data just as you configured it to do in the previous lab.

Your AWS account is now configured as shown in the following diagram:



By the end of this lab, you will have created Lambda functions that the API will invoke. At that point, your account resources and configurations will look like the following diagram:



## Task 2: Creating a Lambda function to retrieve data from DynamoDB

The first Lambda function that you create will respond to any `/products` GET requests from the website. The function will replace the mock endpoint that you created for the `products` API resource in the previous lab.

14. Observe and edit the Python code that you will use in the Lambda function.

- o In the VS Code IDE file browser, browse to and open `python_3/get_all_products_code.py`.
- o Replace the `<FMI_1>` placeholder and the `<FMI_2>` placeholder with the proper values.

**Tip:** To find the missing values in the code, return to the DynamoDB console.

- o Notice that the code does the following:
  - It creates a boto3 client to interact with the DynamoDB service.
  - It reads the items out of the table and returns the menu data.
  - It also scans the table index and filters for items that are in stock.
- o Close the file by choosing **X** from the top. (Your changes are saved automatically).

15. Test the code *locally* in VS Code IDE.

- o To ensure that you are in the correct folder, run the following command:

```
cd ~/environment/python_3
```

- o To run the code locally in the VS Code IDE terminal, run the following command:

```
python3 get_all_products_code.py
```

- o If the code runs successfully, the first part of the table data is returned in the terminal output, formatted as a JSON document as shown here.

running scan on table

```
{'product_item_arr': [{'price_in_cents_int': 295, 'special_int': 1, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A doughnut with blueberry jelly filling.', 'product_name_str': 'blueberry jelly doughnut', 'product_id_str': 'a455'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'so good!', 'product_name_str': 'vanilla glazed doughnut', 'product_id_str':
```

```
'a453'}, {'price_in_cents_int': 295, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': "Boston's favorite doughnut, done right.", 'product_name_str': 'boston cream doughnut', 'product_id_str': 'a458'}, ...truncated for brevity
```

16. Modify a setting in the code and test it again.

- o In the `get_all_products_code.py` file, line 12 has the following code:

```
if offer_path_str is not None:
```

**Analysis:** If the `offer_path_str` variable is not found, the condition fails and runs a scan of the table.

- o To verify that this logic is working, temporarily reverse this condition. Remove the word `not` from this line of code, so that it looks like the following:

```
if offer_path_str is None:
```

- o Close the file by choosing **X** from the top. (Your changes are saved automatically).
- o Run the code again:

```
python3 get_all_products_code.py
```

- o The output resembles the following example:

```
running scan on index
{'product_item_arr': [{'price_in_cents_int': 295, 'special_int': 1, 'tag_str_arr': ['doughnut', 'on offer'], 'description_str': 'A doughnut with blueberry jelly filling.', 'product_name_str': 'blueberry jelly doughnut', 'product_id_str': 'a455'}, {'price_in_cents_int': 595, 'special_int': 1, 'tag_str_arr': ['pie slice', 'on offer'], 'description_str': "A delicious slice of Frank's homemade pie.", 'product_name_str': 'apple pie slice', 'product_id_str': 'a444'}, {'price_in_cents_int': 395, 'special_int': 1, 'tag_str_arr': ['bagel', 'on offer'], 'description_str': 'Boiled in salt water, then baked. As it should be.', 'product_name_str': 'plain bagel', 'product_id_str': 'a465'}, ...truncated for brevity
```

- o Notice that fewer menu items were returned this time. This was the intended result. The website calls to the REST API will implement the code so that customers can filter the menu for "on offer" menu items.

Now that you know that the code logic works, reverse the code change and then define the Lambda function that will run the code.

17. Reverse the change that you made to the code.

- o On line 12, add the word `not` back to the code so that it again reads as the following:

```
if offer_path_str is not None:
```

18. Comment out the last line of the code and save the file.

- o To comment out the last line of code, use a `#`. The line should look like the following:

```
#print(lambda_handler({}, None))
```

- o Close the file by choosing **X** from the top. (Your changes are saved automatically).

19. Locate the IAM role that the Lambda function will use, and copy the role Amazon Resource Number (ARN) value.

- o Browse to the IAM console, and choose **Roles**.
- o In the search box search for and select the **LambdaAccessToDynamoDB** role that has been created for you.

Notice that this role provides read only access to DynamoDB. The policy provides enough access for Lambda to read the data that is stored in DynamoDB.

- o Copy the **Role ARN**

value. You will use this in the next step.

20. Edit the wrapper code that you will use to create the Lambda function.

- o Return to the VS Code IDE file browser.
- o Browse to and open `python_3/get_all_products_wrapper.py`.
- o On line 5, replace `<FMI_1>` with the role ARN value that you copied.
- o Close the file by choosing **X** from the top. (Your changes are saved automatically).

21. Observe what the `get_all_products_wrapper.py` code will accomplish when it is run:

- o Creates a Lambda boto3 client.
- o Sets the name of the role that the Lambda function should use.
- o References the location of the S3 bucket that has the code that the Lambda function should run (the code you just zipped and placed in Amazon S3).
- o Uses all of this information to create a Lambda function. The function definition identifies Python 3.8 as the runtime.

22. Package the code and store it in an S3 bucket.

- o A bucket with `-s3bucket-` in the name was created for you when you started the lab.
- o Verify that your VS Code IDE terminal is in the **python\_3** directory.

```
cd ~/environment/python_3
```

- o To place a copy of your code in a .zip file, run the following command:

```
zip get_all_products_code.zip get_all_products_code.py
```

- o Next, to retrieve the name of your S3 bucket, run the following command:

```
aws s3 ls
```

- o Finally, to place the .zip file in the bucket, run the following command. Replace `<bucket-name>` with the actual bucket name that you retrieved:

```
aws s3 cp get_all_products_code.zip s3://<bucket-name>
```

- o Verify that the command

succeeded. The response looks like the following:

```
upload: ./get_all_products_code.zip to s3://<bucket-name>/get_all_products_code.zip
```

23. To create the Lambda function, run the following command:

```
python3 get_all_products_wrapper.py
```

The output of the command shows DONE, confirming that the code ran without errors.

24. Observe the function that you created and test it.

- o Browse to the Lambda console.
- o Choose the name of the **get\_all\_products** function that you just created.
- o In the **Code source** panel, open (double-click) the **get\_all\_products\_code.py** file to display the code.
- o Choose **Test**.
- o For **Event name**, enter Products
- o Keep all of the other default test event values, and choose

**Save.** The test event is saved.

- o Choose **Test** again.

A tab that shows the results of your test displays, with a response that shows the data returned from the DynamoDB table. The following is an example:

```
{
  "product_item_arr": [
    {
      "price_in_cents_int": 295,
      "special_int": 1,
      "tag_str_arr": [
        "doughnut",
        "on offer"
      ],
      "description_str": "A doughnut with blueberry jelly filling.",
      "product_name_str": "blueberry jelly doughnut",
      "product_id_str": "a455"
    },
    {
      "price_in_cents_int": 295,
      "tag_str_arr": [
        "doughnut",
        "on offer"
      ],
      ... Truncated for brevity
    }
  ]
}
```

25. Create a new test event that is called **onOffer**.

- o In the **Code source** panel, open the **Test** menu (choose the arrow icon), and choose **Configure test event**.
- o Choose **Create new event**.
  - For **Event name**, enter onOffer
  - In the code editor panel, replace the existing code with the following:

```
{
```



```
"path": "on_offer"  
}
```

- o Choose **Save**.
- o Choose **Test**.

This time, the results only display the items that are on offer and not out of stock.

- o Scroll to the bottom of the test results to the function logs. You see the log message running scan on index.

As you can see, your Lambda function is now successfully retrieving data from the DynamoDB table. You have also observed that this one Lambda function can be used to return *all* menu items *or* only the ones that are on offer.

Congratulations on achieving this milestone!

The next step is to configure the REST API to invoke this Lambda function whenever anyone requests product data on the website. You will also need to find a way to pass the path variable to Lambda when customers choose to view only the on offer products.

### Task 3: Configuring the REST API to invoke the Lambda function

First, you tested the code locally in VS Code IDE to ensure that it worked. Then, you deployed the code as a Lambda function and tested that it worked as deployed. In this task, you will configure the `/products` and `/products/on_offer` REST API functions to invoke the `get_all_products` Lambda function so that the code can be invoked from the café website.

26. Test the existing GET `/products` resource.

- o Browse to the API Gateway console.
- o Choose the **ProductsApi** API, and choose the **GET** method for

`/products`. Notice on the right side of the page that the method is still accessing a "Mock Endpoint".

- o Choose **Test**, and then choose **Test** at the bottom of the page.
- o Verify that the Response Body correctly returns the mock data.

27. Replace the mock endpoint with the Lambda function.

- o At the top of the page. Ensure that the **GET** method is still selected under `/products`.
- o Choose **Integration Request** and **Edit**:
  - Integration type: **Lambda Function**
  - Lambda Region: **us-east-1**
  - Lambda Function: `get_all_products`
  - Choose **Save**
- o Choose **Save**.

Notice on the right side of the page that the method is no longer calling a "Mock Endpoint". Instead, it is calling your Lambda function.

28. Test the /products GET API call one more time by selecting

**Test.** The call returns output similar to the following:

```
{
  "product_item_arr": [
    {
      "price_in_cents": 295,
      "special": 1,
      "description": "A doughnut with blueberry jelly filling.",
      "product_name": "blueberry jelly doughnut",
      "product_id_str": "a455",
      "tags": [
        "doughnut",
        "on offer"
      ]
    },
    {
      "price_in_cents": 295,
      "description": "so good!",
      "product_name": "vanilla glazed doughnut",
      "product_id_str": "a453",
      "tags": [
        "doughnut",
        "on offer"
      ]
    },
    ...truncated for brevity(26 items)
  ]
}
```

29. Analyze the results.

- o When you switched the integration endpoint from the mock endpoint to the Lambda function, the response headers that permitted Cross-Origin Resource Sharing (CORS) were removed.
- o If you scroll down past the Response Data, the **Response Headers** section now shows only the following:

```
{"Content-Type":["application/json"],"X-Amzn-Trace-Id":["Root=1-63066f30-6fcb49923559a3bc52eed2ce;Sampled=0"]}
```

The response header does not contain the CORS information that you need because API Gateway resides in a different subdomain (us-east-1.amazonaws.com) than the S3 bucket (s3.amazonaws.com).

You could manually add the CORS configuration back to this resource using the AWS SDK. However, API Gateway has a feature that makes it simple to permit CORS.

30. Re-enable CORS on the /products API resource.

- o Choose **/products** so that it is highlighted.
- o Select the **GET** method.
- o Choose button **Enable CORS** from the top.
- o Select **Default 4XX** and **Default 5XX** under **Gateway responses**.

- o Select **GET** under **Access-Control-Allow-Methods**.
- o Choose **Save**.

31. Test the /products GET API call one more time.

- o Choose the **GET** method for **/products**.
- o Choose **Test**, and then choose **Test** at the bottom of the page.
- o Scroll down to the **Response Headers** section again.
- o This time, Access-Control-Allow-Origin is included. The following is an example of the output (your data will have a different value for Root):

```
{"Access-Control-Allow-Origin":["*"],"Content-Type":["application/json"],"X-Amzn-Trace-Id":["Root=1-63066fa3-6e4218d81cb85b1dc0b78d05;Sampled=0"]}
```

32. Using the same approach, update the /on\_offer GET API method.

- o Choose the **ProductsApi** API, and choose the **GET** method for **/on\_offer**.
- o Choose **Integration Request** and configure:
  - Integration type: **Lambda Function**
  - Lambda Region: **us-east-1**
  - Lambda Function: **get\_all\_products**
- o Choose **Save**.
- o Choose **/on\_offer** so that it is highlighted.
- o Choose **Enable CORS**.
- o Select **Default 4XX** and **Default 5XX** under **Gateway responses**.
- o Select **GET** under **Access-Control-Allow-Methods**.
- o Choose **Save**.

33. Test the /on\_offer GET API call.

- o Choose the **GET** method for **/products/on\_offer**.
- o Choose **Test**, and then choose **Test** at the bottom of the page.
- o Scroll down to the **Response Headers** section. Notice that CORS is enabled in the headers.
- o Scroll back up to the **Response Body** section. Do you notice an

issue? The Lambda function is returning all of the menu item, not just the specials.

This is because the conditional check for on\_offer\_str is not working.

Recall the relevant code:

```
offer_path_str = event.get('path')
if offer_path_str is not None:
```

**Analysis:** You need to set the API logic for `/products/on_offer` to pass the path to the event object that Lambda uses. The next step explains how to do that.

34. Configure the `/on_offer` integration request details.

- o Choose the **GET** method for `/products/on_offer`.
- o Choose **Integration Request**.
- o Choose **Edit**. Expand **Mapping Templates**.
- o Choose **Add mapping template**.
- o In the Content-Type box, enter the following text:

```
application/json
```

- o Under **Generate template** choose **Method request passthrough**.
- o Replace the text with the following:

```
{  
  "path": "$context.resourcePath"  
}
```

This will evaluate to `/products/on_offer`. For now, the code simply checks for the existence of the variable.

- o Choose **Save** at the bottom of the page.

35. Test the GET method for `/on_offer` again.

- o Choose **Test**, and then choose **Test** at the bottom of the page.
- o Verify that only the on offer items (six items) are

returned. Excellent! Now that this filter is working, you can deploy the

API.

36. Deploy the API.

- o In the **Resources** panel, choose the API root `/`.
- o Choose **Deploy API**.
- o For **Deployment stage**, choose **prod**, and then choose **Deploy**.

Congratulations! You have successfully updated the REST API so that it invokes a single Lambda function that can now filter for on offer items. With this logic, you do not need to create and maintain two separate Lambda functions to handle this functionality.

## Task 4: Creating a Lambda function for report requests in the future

In this task, you will complete steps that are similar to what you just did. However, this time you will create the Lambda function for the `/create_report` POST action.

37. Observe and test the Python code that you will use in the Lambda function.

- o Back in the VS Code IDE, browse to and open `python_3/create_report_code.py`.

Notice that this code does not do much yet. It simply returns a message. In a later lab, you will implement more useful logic to actually create a report; however, this code will suffice for now.

- o Run the following command in the terminal:

```
python3 create_report_code.py
```

- o The output returned in the terminal looks like the following:

```
{'msg_str': 'Report processing, check your phone shortly'}
```

Notice the capitalized "R" in the word Report. The mock data contained a lowercase "r" instead. This difference is how you can know that the website is accessing the Lambda function and not the mock data.

38. Comment out the last line of the code in the `create_report_code.py` file.

- o To comment out the last line of code, use a `#`. The line should look like the following:

```
#print(lambda_handler(None, None))
```

39. Edit the wrapper code that you will use to create the Lambda function.

- o Browse to and open `python_3/create_report_wrapper.py`.
- o On line 5, replace the `<FMI_1>` placeholder with the `LambdaAccessToDynamoDB Role ARN` value.

**Tip:** You may need to return to the IAM console to copy the Role ARN value.

- o Close the file by choosing **X** from the top. (Your changes are saved automatically).

40. Package the code and store it in the S3 bucket.

- o To place a copy of your code in a `.zip` file, run the following command:

```
zip create_report_code.zip create_report_code.py
```

- o To place the `.zip` file in the bucket, run the following command. Replace `<bucket-name>` with the actual bucket name:

```
aws s3 cp create_report_code.zip s3://<bucket-name>
```

- o Verify that the command

succeeded. The following is the output:

```
upload: ./create_report_code.zip to s3://<bucket-name>/create_report_code.zip
```

41. Finally, to create the Lambda function, run the following command:

```
python3 create_report_wrapper.py
```

The output of the command is **DONE**, confirming that the code ran without errors.

42. Observe the `create_report` function that you created and test it.

- o Browse to the Lambda console.
- o Choose the name of the **create\_report** function that you just created.
- o In the **Code source** panel, open (double-click) the **create\_report\_code.py** file to display the code.
- o Choose **Test**.
- o For **Event name**, enter `ReportTest`

- o Keep all of the other default test event values, and choose

**Save.** The test event is saved.

- o Choose **Test** again.

A tab that shows the results of your test displays, with a response that shows the message hardcoded into the function code. The following is an example:

```
{  
  "msg_str": "Report processing, check your phone shortly"  
}
```

Note the capitalized "R" instead of the lowercase "r" that is used in the mock data. Therefore, you know that the website is accessing the Lambda function and not the mock data.

## Task 5: Configuring the REST API to invoke the Lambda function to handle reports

Recall that in a previous task you configured both GET methods in your API to invoke the first Lambda function that you created. In this task, you will complete similar steps to configure the POST method in the API to invoke the new create\_report Lambda function.

43. Test the existing POST method for /create\_report.

- o Browse to the API Gateway console.
- o Choose the **ProductsApi** API, and choose the **POST** method for

**create\_report**. Notice on the right side of the page that the method is still accessing a "Mock Endpoint".

- o Choose **Test**, and then choose **Test** at the bottom of the page.

Verify that the **Response Body** correctly returns the mock data (note the lowercase "r" in the results), as in the following example:

```
{  
  "message_str": "report requested, check your phone shortly."  
}
```

44. Replace the mock endpoint with the Lambda function.

- o Ensure that the **POST** method is still selected.
- o Choose **Integration Request** and **Edit**:
  - Integration type: **Lambda Function**
  - Lambda Region: **us-east-1**
  - Lambda Function: create\_report
  - Choose **Save**.

Notice on the right side of the page that the POST method is no longer calling a "Mock Endpoint". Instead, it is calling the Lambda function.

- o Test the method again.

The returned data looks like the following example. Note that you are now seeing a capitalized "R" instead of the lowercase "r" from the mock data:

```
{"msg_str": "Report processing, check your phone shortly"}
```

**Note:** You could configure CORS on this method as you did for the other API methods. However, you do not need to because the website will not invoke this URL until a later lab when you enable authentication.

45. Deploy the API.

- o In the **Resources** panel, choose the API root /.
- o Choose **Deploy API**.
- o For **Deployment stage**, choose **prod**, and then choose **Deploy**.

Congratulations! You have successfully updated the REST API so that it invokes the create\_report Lambda function.

## Task 6: Testing the integration using the café website

In this final task, you will test both API calls (/products and /products/on\_offer) through the website.

46. Load the café website.

- o Return to the browser tab where you have the café website open, and refresh the page.
  - **Note:** To find the page again, browse to the Amazon S3 console. Choose the bucket name, choose **index.html**, and copy the **Object URL** value. Load the URL in a new browser tab.

The café website displays. The website is now accessing the menu data that is stored in DynamoDB.

47. Test the menu items filter on the website.

- o Scroll down to the **Browse Pastries** section of the page. By default, only the "on offer" menu items display.
- o Choose **view all** and verify that more menu items are returned.

If everything displays correctly, that means that your CORS configuration is working properly.

48. Edit a menu item price in the DynamoDB table, and verify that the change is reflected on the website.

- o Select your favorite "on offer" menu item and note the current price.
- o In a different browser tab, go to the DynamoDB console and load the **FoodProducts** table items.
- o To open the menu item's record, choose the **product\_name** hyperlink for that item.
- o Change the **price\_in\_cents** value to a different three-digit or four-digit number.
- o Save the change.
- o Reload the café website, and verify that the price change is reflected on the website.

## Submitting your work

49. 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** checkbox. This action will ensure that the lab instructions remain visible after you choose **Submit**.

50. 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.

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

## **Lab complete**

Congratulations! You have completed the lab.

52. 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.

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



## Experiment 06: Migrating a Web Application to Docker Containers

### 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 would like the lab instructions to display across the entire browser window, you can hide the terminal in the browser panel by unchecking the Terminal checkbox in the top right.

### Task 1: Preparing the development environment

In this first task, you will connect to VS Code IDE and configure the environment to support the development that you will work on during the rest of the lab.

5. Connect to the VS Code IDE.
- o At the top of these instructions, choose Details followed by **AWS: Show**
  - o Copy values from the table for the following and paste it into an editor of your choice for use later.
    - **LabIDEURL**
    - **LabIDEPassword**
  - o In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
  - o On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.
6. Download and extract the files that you will need for this lab.
- o In the same terminal, run the following command:

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/06-lab-containers/code.zip -P /home/ec2-user/environment
```

- o The code.zip file is downloaded to the VS Code IDE. The file is listed in the left navigation pane.
- o Extract the file:

```
unzip code.zip
```

**Note:** You will use the downloaded and extracted files later in this lab.

7. Run a script that ensures you can get full credit when you choose to submit this lab. It also will upgrade the version of Python and the AWS CLI that are installed on the VS Code IDE.

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

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

- o Verify that the script completed without error.
8. Verify the version of AWS CLI installed.
    - o In the VS Code IDE Bash terminal (at the bottom of the IDE), run the following command:

```
aws --version
```

The output should indicate that version 2 is installed.

9. Verify that the SDK for Python is installed.
  - o Run the following command:

```
pip3 show boto3
```

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

## Task 2: Analyzing the existing application infrastructure

In this task, you will analyze the current application infrastructure.

10. Open the existing coffee supplier application in a browser tab.
  - o Return to the browser tab labeled **Your environments**, and navigate to the EC2 console.
  - o Choose **Instances**.

Notice that three instances are running.

- o One instance is the VS Code IDE that you used in the previous task.

- o The other two instances (MySQLServerNode and AppServerNode) support the application that you will containerize in this lab.
- o Choose the **AppServerNode** instance, and copy the **Public IPv4 address** value.
- o Open a new browser tab and navigate to the IP

address. The coffee suppliers website displays.

**Note:** The page uses http:// instead of https://. Your browser might indicate that the site is not secure, because it does not have a valid SSL/TLS certificate. You can ignore the warning in this development environment.

#### 11. Test the web application functionality.

- o Choose **List of suppliers** and then choose **Add a new supplier**.
- o 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](mailto:nwolf@example.com)
  - **Phone:** 4155551212
- o Choose **Submit**.

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

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

**Submit.** Notice that the change was saved in the record.

#### 12. Analyze the web application code.

- o A copy of the application code that is installed on the AppServerNode EC2 instance is also available in your VS Code IDE.
  - Return to the VS Code IDE browser tab.
  - In the file browser in the left navigation pane, expand the **resources** directory, and then expand the **codebase\_partner** directory to see the application code.
  - **Optional:** If you are interested to know how the code is configured on the AppServerNode, you can connect to the instance and view the code there as well. To do this, in the terminal next to these instructions, run the following commands, to connect to the EC2 instance and see the files that are installed (replace <public-ip-address> with the actual IPv4 address of the AppServerNode instance).

```
ssh -i ~/.ssh/labsuser.pem ubuntu@<public-ip-address>
cd resources/codebase_partner
ls -l
```

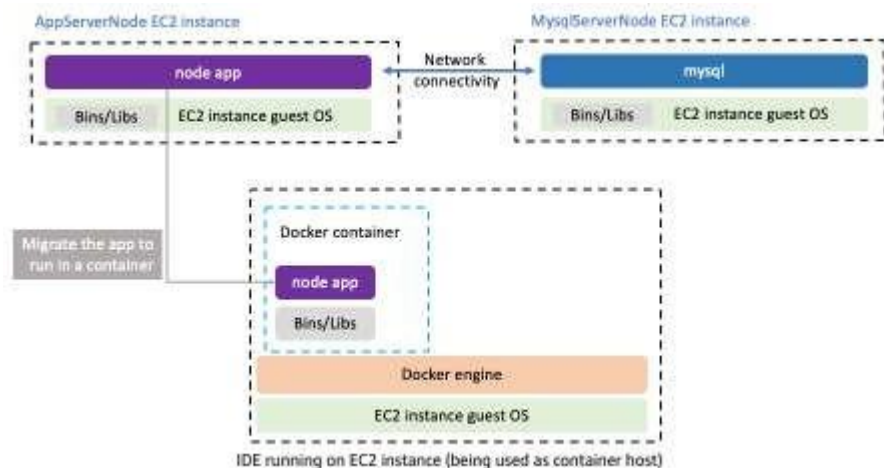
- o For this lab, it is not necessary for you to understand the details of how the application was built. However, the following details might be of interest to you:
  - The application was built with Express, which is a framework for building web applications.
  - The application runs on port 80 and is coded in node.js.
  - To install the application directly on the guest OS of the AppServerNode Ubuntu Linux EC2 instance, node.js and the node package manager (npm) were installed. Then, the code that you can see in **resources/codebase\_partner** was placed on the server, and the connection to the application's database was configured.

### 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 instead run in a Docker container. The Docker container is portable and could run on any OS that has the Docker engine installed.

For convenience, you will run the container on the same EC2 instance that hosts the VS Code IDE that you are using. You will use this IDE to build the Docker image and launch the Docker container.

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



13. Create a working directory for the node application, and move the source code into the new directory.
  - o In the VS Code IDE, to create and navigate to a directory to store your Docker container code, run the following commands:

```
mkdir containers
cd containers
```

- o To create and navigate to a directory named **node\_app** inside of the **containers** directory, run the following commands:

```
mkdir node_app  
cd node_app
```

- o To move the code base, which you copied earlier, into the new **node\_app** directory, run the following command:

```
mv ~/environment/resources/codebase_partner ~/environment/containers/node_app
```

#### 14. Create a Dockerfile.

**Note:** A *Dockerfile* is where you provide instructions to Docker to build an image. A Docker *image* is a template that has instructions to create a *container*.

- o To create a new Dockerfile named **Dockerfile** in the **node\_app/codebase\_partner** directory, run the following command:

```
cd ~/environment/containers/node_app/codebase_partner  
touch Dockerfile
```

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

```
FROM node:11-alpine  
RUN mkdir -p /usr/src/app  
WORKDIR /usr/src/app  
COPY . .  
RUN npm install  
EXPOSE 3000  
CMD ["npm", "run", "start"]
```

- o Save the changes.

**Analysis:** This Dockerfile code specifies that an Alpine Linux distribution with node.js runtime requirements should be used to create the image. The code also specifies that the container should allow network traffic on TCP port 3000. Finally, the code specifies that the application should be run and started when the container launches.

#### 15. Build an image from the Dockerfile.

- o In the VS Code IDE Bash terminal, run the following command:

```
docker build --tag node_app .
```

- o The output is similar to the following:

```
Sending build context to Docker daemon 9.007MB
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
Step 2/7 : RUN mkdir -p /usr/src/app
--> Running in 768899b8cc6f
Removing intermediate container 768899b8cc6f
--> e3d5cc4cafd7
Step 3/7 : WORKDIR /usr/src/app
--> Running in c16e1d316a6c
Removing intermediate container c16e1d316a6c
--> 9557a073a12b
Step 4/7 : COPY . .
--> 373727287dc7
Step 5/7 : RUN npm install
--> Running in 4ef97681cff6
npm WARN coffee_api@1.0.0 No repository field.
audited 78 packages in 0.862s
found 0 vulnerabilities
Removing intermediate container 4ef97681cff6
--> 847f6f8474c5

Step 6/7 : EXPOSE 3000
--> Running in 39ff9456b6a8
Removing intermediate container 39ff9456b6a8
--> 1e7614a93ae1
Step 7/7 : CMD ["npm", "run", "start"]
--> Running in ff6310d7bdbd
Removing intermediate container ff6310d7bdbd
--> a5886f101e12
Successfully built a5886f101e12
Successfully tagged node_app:latest
```

**Note:** Ignore any minor warnings in the output.

#### 16. Verify that the Docker image was created.

- o To list the Docker images that your Docker client is aware of, run the following command:

```
docker images
```

- o The output is similar to the following:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
node_app	latest	39501e06b5e9	6 seconds ago	84.5MB
node	11-alpine	f18da2f58c3d	4 years ago	75.5MB

- o Notice that the **node\_app** line item was created only a few seconds ago.

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

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

```
docker run -d --name node_app_1 -p 3000:3000 node_app
```

**Analysis:** This command launches a container with the name **node\_app\_1**, using the **node\_app** image that you created as the template. The -d argument specifies that it should run in the background and print the container ID. The -p specifies to publish container port 3000 to the host (VS Code IDE) port 3000.

- o The terminal returns the container ID. It is a long string of letters and numbers.
- o To view the Docker containers that are currently running on the host, run the following command:

```
docker container ls
```

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

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

```
curl http://localhost:3000
```

- o The webpage looks 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">
    
    <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>

```

**Analysis:** This demonstrates that the application is running and available on TCP port 3000. However, you want to interact with this as a website, and you don't yet know if the application is able to connect to the database.

19. Adjust the security group of the VS Code IDE instance to allow network traffic on port 3000 from your computer.

**Note:** Because you are using the VS Code IDE instance to run the container, you must open TCP port 3000 for inbound traffic.

- o Return to the AWS Management Console browser tab, and navigate to the EC2 console.
  - o Locate and select the **Lab IDE** instance.
  - o Choose the **Security** tab, and choose the security group hyperlink.
  - o Choose the **Inbound rules** tab, and choose **Edit inbound rules**.
  - o Choose **Add rule** and configure the following:
    - Type: **Custom TCP**
    - Port range: **3000**
    - Source: **My IP**
  - o Choose **Save rules**.
20. Access the web interface of the application, which is now running in a container.
    - o In the EC2 console, choose **Instances** and choose the **Lab IDE** instance.

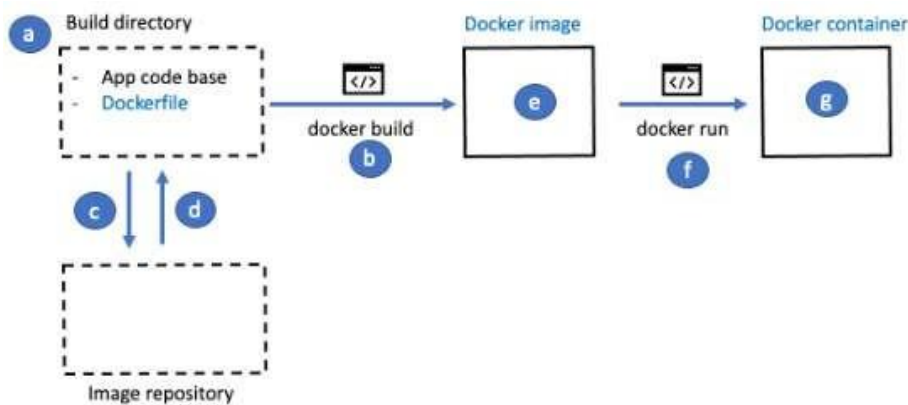


- o On the **Details** tab, copy the **Public IPv4 address** value.
- o Open a new browser tab. Paste the IP address into the address bar, and add :3000 at the end of the address.

The web application loads in the browser. You have seen this page before; however, you previously accessed the application that was running directly on the AppServerNode EC2 instance guest OS. This time you accessed the application running in a container on the VS Code IDE instance's Docker hypervisor.

## 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.



- You copied the code base into a directory, which acted as your build area [a]. You also created a Dockerfile that provided instructions for how to create a Docker image. That Dockerfile specified a FROM instruction that identified a starter image to use.
- You then ran the docker build command [b]. Docker read the Dockerfile and requested the starter image from an image repository [c]. The image repository returned the starter image file [d].
- The docker build command then finished building the image according to the instructions in the Dockerfile, which resulted in the Docker image [e].
- Finally, you ran the docker run command [f] to run a Docker container [g].

21. Analyze the database connection issue.

o 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**.

- o Return to the VS Code IDE browser tab.
- o Open the **config.js** file in the **containers/node\_app/codebase\_partner/app/config/** directory.

The top of the file contains the following code:

```
let config = {
  APP_DB_HOST: "3.82.161.206",
  APP_DB_USER: "nodeapp",
  APP_DB_PASSWORD: "coffee",
  APP_DB_NAME: "COFFEE"
}
```

**Analysis:** The application code checks for an environmental variable to learn how to connect to the MySQL database. As you see in the code, the settings include a hardcoded IP address for the location of the MySQL database host.

- o However, the application code also contains the following logic:

```
object.keys(config).forEach(key => {
  if(process.env[key] === undefined){
    console.log(`[NOTICE] value for key '${key}' not found in ENV, using default
    value. See app/config/config.js`)
  } else {
    config[key] = process.env[key]
  }
});
```

This checks for the existence of an environmental variable. If one is found, that value overrides the placeholder (hardcoded) APP\_DB\_HOST address.

When you visited the web application earlier—the version that was directly installed on the EC2 instance guest OS—the node instance passed an environment variable (the IPv4 address of the MySQLServerNode EC2 instance) to the application.

However, you did not launch your node\_app\_1 container with that environment variable. Therefore, the node application defaulted to the hardcoded 3.82.161.206 IP address, which does not match the IP address of the MySQL instance.

- o Establish a terminal connection to the container to observe the settings.
  - To find the container ID, run the following command:

```
docker ps
```

- To connect your terminal to the container, run the following commands, one at a time. Replace `<container-id>` with the actual container ID value that you just retrieved:

```
docker exec -ti <container-id> sh
whoami
```

Your terminal is now connected to the container as the root user.

- o To observe the environment variables that are present in the node user's environment, run the following commands:

```
su node
env
```

Notice that the **APP\_DB\_HOST** variable is not present.

- o To disconnect from the container, run the following commands:

```
exit
exit
```

The first exit command makes you the root user again. The second command disconnects you from the container.

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

- o 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.

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

```
docker stop node_app_1 && docker rm node_app_1
```

- o To verify that the application is no longer running, run the following curl command:

```
curl http://localhost:3000
```

The output indicates a failure to connect to the application (*Connection refused*).

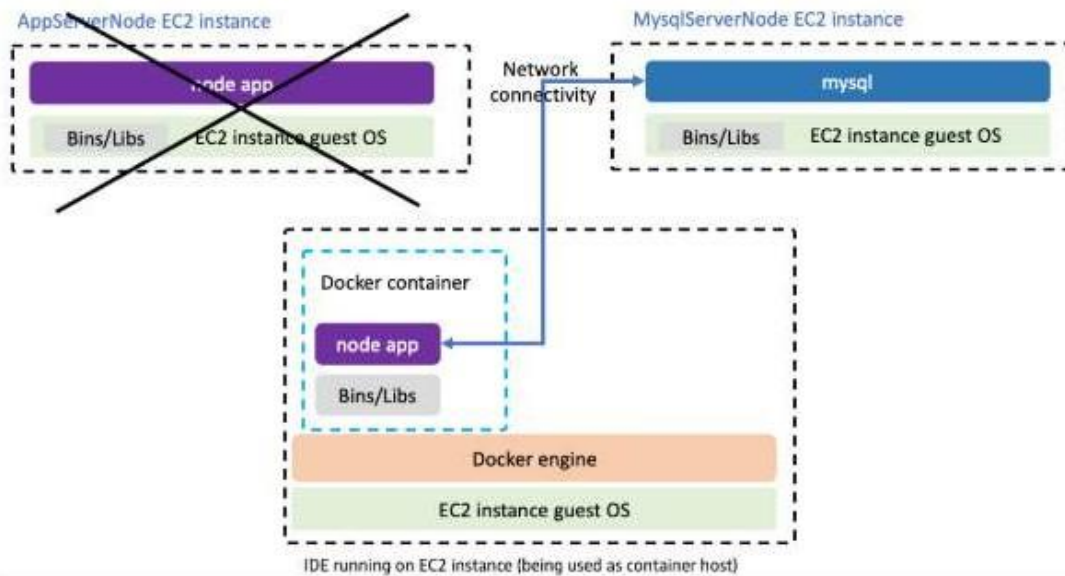
**Tip:** If you refresh the web application in the browser (the version that is running as a container on the VS Code IDE), you will find that the application no longer loads.

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

- o Return to the EC2 console, and copy the **Public IPv4 address** value of the **MysqlServerNode** EC2 instance.
- o Return to the VS Code IDE Bash terminal.
- o 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:

```
docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST="<ip-address>"
node_app
```

When you pass in the network location of the database as an environment variable, you give the node application the information that it needs to establish network connectivity to the database, as illustrated in the following diagram:



O  
w

and,

24. Verify that the database connection is now working.

- o Try to access the web application again.

1. If you still have the page open, refresh the browser tab. with  
Otherwise, to navigate to the application in a new browser tab, go to

`http://<LabIDE-public-ip>:3000/!`

`public-ip>`

the actual public IPv4 address of your VS Code IDE).

- o The application is working. Choose **List of suppliers** to go to the

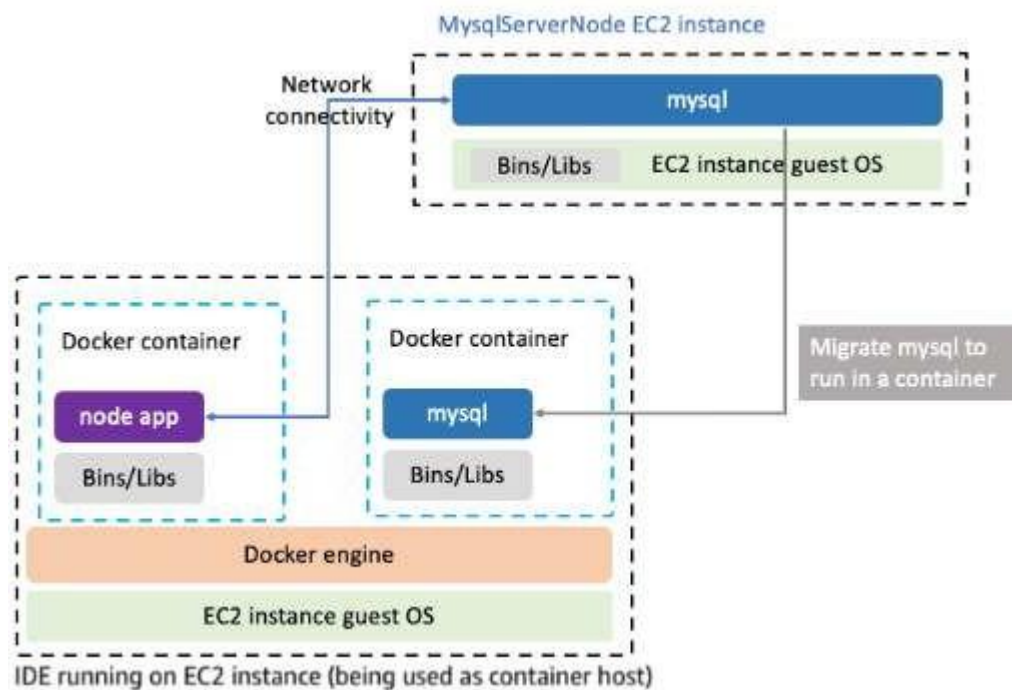
`http://<LabIDE-public-ip>:3000/!`

page.

- o The page displays the supplier entry that you created earlier. This indicates that your container is connecting to the MySQLServerNode EC2 instance where that data is stored.

Congratulations! You have successfully migrated the node application to a container. Also, the application container (**node-app\_1**) is now able to successfully establish a network connection with the MySQL database, which is still running on an EC2 instance.

## 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 above diagram shows the migration that you will accomplish in this task:

25. Create a mysqldump file from the data that is currently in the MySQL database.
  - o Return to the VS Code IDE, and close any file tabs that are open in the text editor.
  - o Choose **File** > **New File** and then paste the following code into the new file:

```
mysqldump -P 3306 -h <mysql-host-ip-address> -u nodeapp -p --databases COFFEE >
../../my_sql.sql
```

- o Next, go to the EC2 console and copy the **Public IPv4 address** value of the **MysqlServerNode** instance.
- o Return to the text file in VS Code IDE and replace **<mysql-host-ip-address>** in the code with the IP address that you copied.
- o In the terminal, to ensure that you are in the correct directory, run the following command:

```
cd /home/ec2-user/environment/containers/node_app/codebase_partner
```

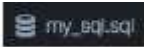
- o Finally, copy the command that you created in the text editor into the terminal and run the command.

Your command will look similar to the following example, but your IP address will be different:

```
mysqldump -P 3306 -h 100.27.45.2 -u nodeapp -p --databases COFFEE >
../../my_sql.sql
```

- o The mysqldump utility prompts you for a password. Enter the following password:

```
coffee
```

If successful, the terminal does not show any output. However, in the left navigation panel, notice that the  file now appears in the **containers** directory.

**Tip:** To see the new file, you might need to choose the settings icon in the upper-right corner of the file tree panel and then choose **Refresh File Tree**.

26. Open the mysqldump file and observe the contents.

- o Open the **my\_sql.sql** file in the VS Code IDE editor.
- o 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.
- o 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.

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

```
cd /home/ec2-user/environment/containers
mkdir mysql
cd mysql
```

28. Create a Dockerfile.

- o To create a new Dockerfile, run the following command:

```
touch Dockerfile
```

- o To move the sqldump file into the new **mysql** directory, run the following command:

```
mv ../../my_sql.sql .
```

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

```
FROM mysql:8.0.23
COPY ./my_sql.sql /
EXPOSE 3306
```

- o Save the changes.

**Analysis:** This Dockerfile code specifies that a new Docker image should be created by starting with an existing mysql Docker image. Then, the sqldump file, which you created in a previous step, is copied to the container. The code also specifies that the container should allow network traffic on TCP port 3306, which is the standard MySQL port for network communication.

29. Attempt to free up some disk space on the VS Code IDE instance by removing unneeded files.

- o Run the following command:

```
docker rmi -f $(docker image ls -a -q)
```

**Note:** You can safely ignore any *Error response from daemon* messages that display.

- o Finally, run the following command:

```
sudo docker image prune -f && sudo docker container prune -f
```

**Note:** In some cases, the total reclaimed space might be zero; however, you might see that some unneeded containers were deleted.

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

```
docker build --tag mysql_server .
```

The output is similar to the following:

```

Sending build context to Docker daemon 5.12kB
Step 1/3 : FROM mysql:8.0.23
8.0.23: Pulling from library/mysql
f7ec5a41d630: Pull complete
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
5726073179b6: Pull complete
eadfac8b2520: Pull complete
f5936a8c3f2b: Pull complete
cca8ee89e625: Pull complete
6c79df02586a: Pull complete
Digest: sha256:6e0014cdd88092545557dee5e9eb7e1a3c84c9a14ad2418d5f2231e930967a38
Status: Downloaded newer image for mysql:8.0.23
---> cbe8815cbea8
Step 2/3 : COPY ./my_sql.sql /
---> b71b3be6d378
Step 3/3 : EXPOSE 3306

---> Running in 975ff38ce91c
Removing intermediate container 975ff38ce91c
---> 13c22244afbb
Successfully built 13c22244afbb
Successfully tagged mysql_server:latest

```

31. Verify that the Docker image was created.

- To list the Docker images that your Docker client is aware of, run the following command:

```
docker images
```

- The output is similar to the following:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql_server	latest	13c22244afbb	About a minute ago	546MB
node_app	latest	6148dceae9d0	3 hours ago	82.7MB
mysql	8.0.23	cbe8815cbea8	3 weeks ago	546MB
node	11-alpine	f18da2f58c3d	23 months ago	75.5MB

- Notice the **mysql\_server** line item, which was created only a few minutes ago.



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

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

```
docker run --name mysql_1 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=rootpw -d
mysql_server
```

**Analysis:** This command launches a container with the name **mysql\_1**, using the **mysql\_server** image that you created as the template. The **-e** parameter passes an environment variable.

- o The terminal returns the container ID for the container.
- o To view the Docker containers that are currently running on the host, run the following command:

```
docker container ls
```

Two containers are now running. One hosts the node application, and the other hosts the MySQL database.

- o Note the *CONTAINER ID* of the **mysql\_1**, you will need this later.

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

- o Run the following command:

⚠ Note that space is *not* included between **-p** and **rootpw** in the command.

```
sed -i '1d' my_sql.sql
docker exec -i mysql_1 mysql -u root -prootpw < my_sql.sql
```

Ignore the warning about using a password on the command line interface being insecure.

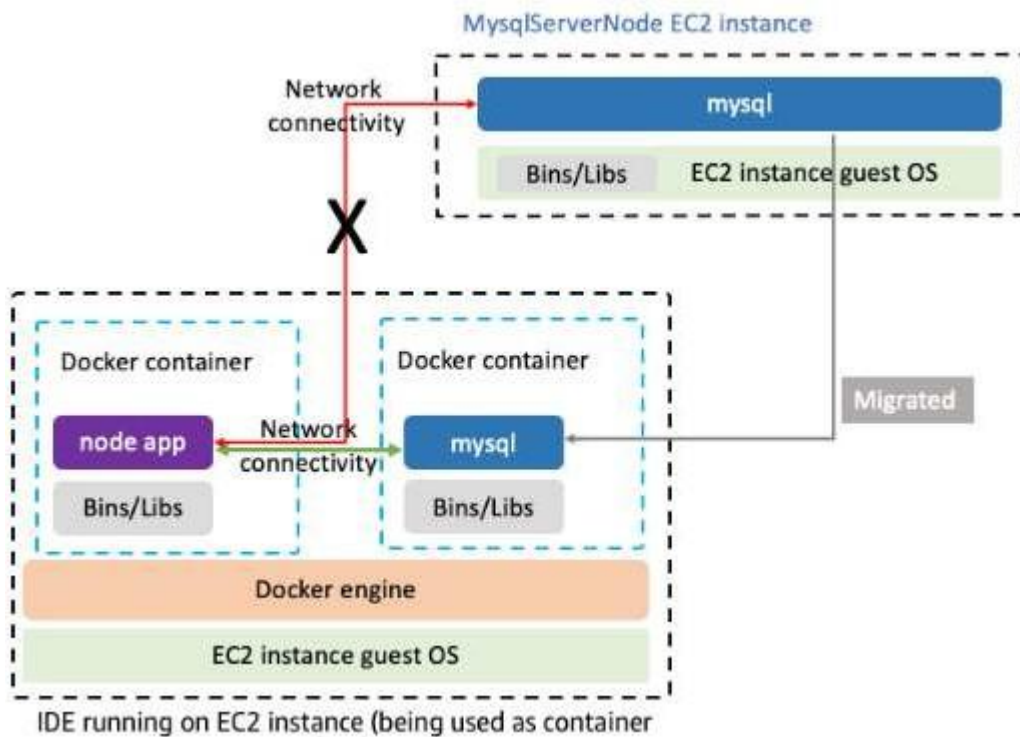
- o To create a database user for the node application to use, run the following command:

```
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'@'%';"
```

## 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:



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

```
docker stop node_app_1 && docker rm node_app_1
```

35. Discover the network connectivity information.

- o To find the IPv4 address of the **mysql\_1** container on the network, run the following command:

```
docker inspect <CONTAINER ID>
```

The following example output shows only a portion of the output:

```
    "NetworkSettings": {
      "Bridge": "",
      "SandboxID":
"11c6c262578c1de828ca85c2691d039a236eb1c4d69d3569feb4e91da035dff4",
      "SandboxKey": "/var/run/docker/netns/11c6c262578c",
      "Ports": {
        "3306/tcp": [
          {
            "HostIp": "0.0.0.0",
            "HostPort": "3306"
          },
          {
            "HostIp": "::",
            "HostPort": "3306"
          }
        ],
        "33060/tcp": null
      },
      "HairpinMode": false,
```

```

        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID":
"02a66a897f5c13e0f6e5578282b837b7a68dc9971e09649f33a2a9afb3ae06c2",
        "Gateway": "172.17.0.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "172.17.0.3",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "MacAddress": "02:42:ac:11:00:03",
        "Networks": {
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "MacAddress": "02:42:ac:11:00:03",
                "NetworkID":
"dcecd17963569a9f56ff5354db686cc1db7214a7677a92ff39b3b88d509fa7f9",
                "EndpointID":
"02a66a897f5c13e0f6e5578282b837b7a68dc9971e09649f33a2a9afb3ae06c2",
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.3",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "DriverOpts": null,
                "DNSNames": null
            }
        }
    }
}

```

In the example output, the `IPAddress` used by the **mysql\_1** container is 172.17.0.3.

⚠ The IP address used by *your* **mysql\_1** container is different.

36. Start a new node application Docker container.

- o In this step, you run the Docker command to start a new container from the **node\_app** Docker image. However, this time you pass the **APP\_DB\_HOST** environment variable to the container environment.
- o Run the following command. Replace `<ip-address>` with the actual IPv4 address value that you just discovered. You do *not* need to surround the IP address in quotes.

```
docker run -d --name node_app_1 -p 3000:3000 -e APP_DB_HOST=<ip-address> node_app
```

The container starts as it did previously.

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

```
docker ps
```

38. Test the application.

- o Open the web application that is running as a container on the VS Code IDE.

The address for the web application is <http://<LabIDE-public-ip-address>:3000>

- o In the coffee suppliers application, choose **List of suppliers** to verify that the database is connected.

If you see the supplier entry that you created previously, and the entry contains the change that you made to the street name (for example, Container Street), then that is confirmation that you have successfully connected the node\_app running in the container to the database running in the other container.

One of the important benefits of running an application on containers is the portability and scalability that doing so provides.

Sofia has now proven that she can launch functional containers from each of the two Docker images that she created, she is ready to move this solution into production.

**NOTE:** to get full credit when you submit your lab, leave the two Docker containers running.

## 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.

39. Authorize your Docker client to connect to the Amazon ECR service.

- o Discover your AWS account ID.
  - In the AWS Management Console, in the upper-right corner, choose your user name. Your user name begins with **voclab/user**.
  - Copy the **My Account** value from the menu. This is your AWS account ID.
- o Next, return to the VS Code IDE Bash terminal.
- o To authorize your VS Code IDE Docker client, run the following command. Replace **<account-id>** with the actual account ID that you just found:

```
aws ecr get-login-password \
--region us-east-1 | docker login --username AWS \
--password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com
```

A message indicates that the login succeeded.

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

```
aws ecr create-repository --repository-name node-app
```

The response data is in JSON format and includes a **repositoryArn** value. This is the URI that you would use to reference your image for future deployments.

The response also includes a **registryId**, which you will use in a moment.

41. 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.

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

```
docker tag node_app:latest <registry-id>.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
```

The command does not provide a response.

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

```
docker images
```

- o This time, notice that the **latest** tag was applied and the image name includes the remote repository name where you intend to store it. The following image provides an example of the output:

```
voclabs:~/environment/containers/mysql $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql_server	latest	13c22244afbb	3 hours ago	546MB
642015801240.dkr.ecr.us-east-1.amazonaws.com/node_app	latest	6148dcae9d0	6 hours ago	82.7MB
node_app	latest	6148dcae9d0	6 hours ago	82.7MB
mysql	8.0.23	cbe8015cbea8	3 weeks ago	546MB
node	11-alpine	f18da2f58c3d	23 months ago	75.5MB

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

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

```
docker push <registry-id>.dkr.ecr.us-east-1.amazonaws.com/node-app:latest
```

- o The output is similar to the following:

```
The push refers to repository [642015801240.dkr.ecr.us-east-1.amazonaws.com/node-app]
006e0ec54dba: Pushed
59762f95cb06: Pushed
22736f780b31: Pushed
d81d715330b7: Pushed
1dc7f3bb09a4: Pushed
dcaceb729824: Pushed
f1b5933fe4b5: Pushed
latest: digest:
sha256:f75b60adddb8d6343b9dff690533a1cd1fbb34ccce6f861e84c857ba7a27b77d size: 1783
```

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

```
aws ecr list-images --repository-name node-app
```

The command returns information about the image that you just uploaded. The output is similar to the following:

```
{
  "imageIds": [
    {
      "imageDigest":
"sha256:f75b60adddb8d6343b9dff690533a1cd1fbb34ccce6f861e84c857ba7a27b77d",
      "imageTag": "latest"
    }
  ]
}
```

## Submitting your work

44. 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** ☐ checkbox. This action will ensure that the lab instructions remain visible after you choose **Submit**.

45. 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.

46. To find detailed feedback on your work, choose [Details](#) followed by ► **View Submission Report**.

## Lab complete

Congratulations! You have completed the lab.

47. 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.

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



## **Experiment 07: Caching Application Data with ElastiCache, Caching with Amazon CloudFront, Caching Strategies**

### **Accessing the AWS Management Console**

1. At the top of these instructions, choose **Start Lab**.
  - o The lab session starts.
  - o A timer displays at the top of the page and shows the time remaining in the session.

**Tip:** To refresh the session length at any time, choose **Start Lab** again before the timer reaches 0:00.

- o Before you continue, wait until the circle icon to the right of the [AWS](#) link in the upper-left corner turns green.
2. To connect to the AWS Management Console, choose the **AWS** link in the upper-left corner, above the terminal window.
    - o A new browser tab opens and connects you to the console.

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

3. if you see a message  
`The credentials in your login link were invalid... To logout, click here.` link, then double-click the [AWS](#) link above these instructions again.
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** check box.

### **Task 1: Preparing the development environment**

In this first task, you will configure your Visual Studio Code Integrated Development Environment (VS Code IDE) so that you can use Python and the AWS Command Line Interface (AWS CLI) to interact with AWS services.

5. Connect to the VS Code IDE.

- o At the top of these instructions, choose Details followed by **AWS: Show**
  - o Copy values from the table for the following and paste it into an editor of your choice for use later.
    - **LabIDEURL**
    - **LabIDEPassword**
    - In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
    - On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.
6. Download and extract the files that you need for this lab.
- o In the same terminal, run the following command:

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/08-lab-db-caching/code.zip -P /home/ec2-user/environment
```

- o Notice that a **code.zip** file was downloaded to the VS Code IDE. The file is listed in the Environment window.
- o To extract the files, run the following command:

```
unzip code.zip
```

7. Run a script to upgrade the AWS CLI that are installed on the VS Code IDE. It will also recreate the work that you completed in earlier labs into this AWS account.

**Note:** This script deploys the architecture that you created across the previous labs. This includes populating and configuring the Amazon Simple Storage Service (Amazon S3) bucket that the café website uses. The script creates the Amazon DynamoDB table and populates it with data. The script recreates the REST API that you created in the Amazon API Gateway lab. Finally, the script deploys the containerized coffee suppliers AWS Elastic Beanstalk application.

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

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

- o 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:** 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 it to 0.0.0.0 because the S3 bucket's block public access settings will prevent access.

8. Verify the version of AWS CLI installed.
- o In the VS Code IDE Bash terminal (at the bottom of the IDE), run the following command:

```
aws --version
```

The output should indicate that version 2 is installed.

9. Verify that the SDK for Python is installed.

- o Run the following command:

```
pip3 show boto3
```

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

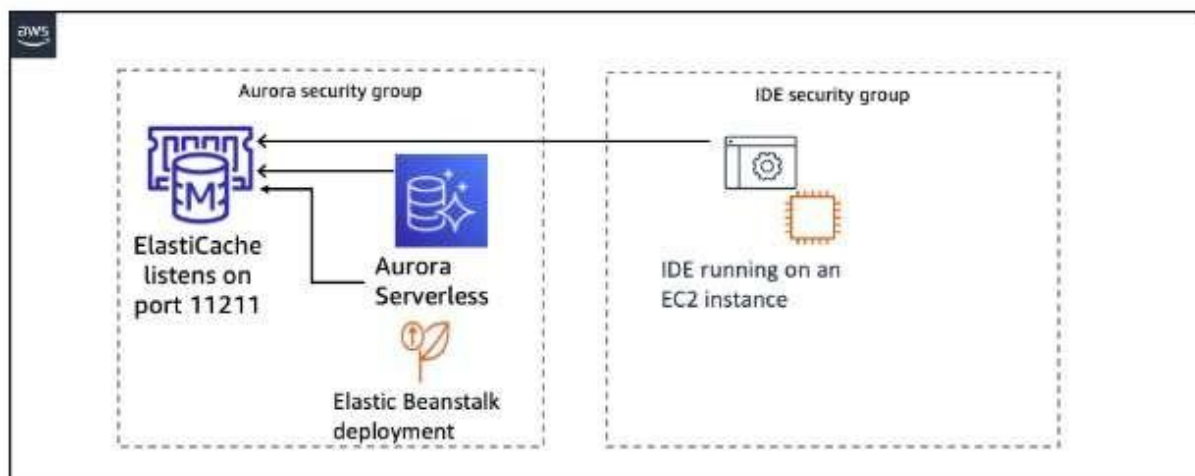
- o Keep the VS Code IDE open in your browser. You will use it later in this lab.

## Task 2: Configuring the subnets for ElastiCache to use

The Memcached database must be able to communicate with the Aurora Serverless database. In addition, the application needs to be able to access both the Memcached database and the Aurora Serverless database.

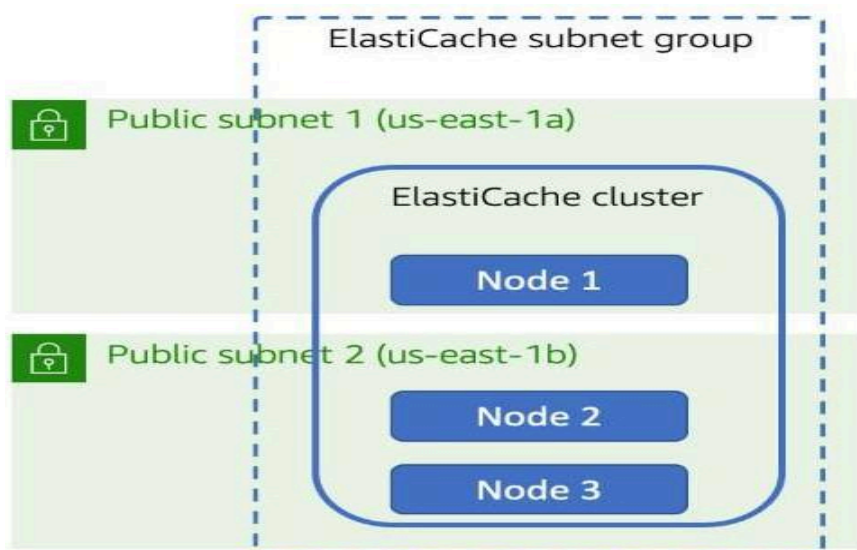
In this task, you will act as Olivia to configure a security group to enable the necessary communication between the application, database, and cache. You will use the security group that the Aurora Serverless cluster is already using rather than creating an additional security group.

10. Configure the security group that the Aurora Serverless instance will use, as shown in the following diagram.



- o Return to the AWS Management Console browser tab.
- o From the **Services** menu, choose **EC2**.
- o In the left navigation pane, choose **Security Groups**.
- o Select the checkbox for the security group that has *Aurora* in the name.
- o In the lower pane, choose the **Inbound rules** tab.
- o Choose **Edit inbound rules**.

- Add an entry to enable access to the Memcached port from all sources that are assigned to the *Aurora* security group:
    - Choose **Add rule**
    - **Type:** Choose **Custom TCP**
    - **Port range:** Enter 11211
    - **Source:** Choose the security group with *Aurora* in the name. This is the same group that you are currently editing.
  - o Add an entry to enable access to the Memcached port from all sources that are assigned to the *Lab IDE SG* security group:
    - Choose **Add rule**
    - **Type:** Choose **Custom TCP**
    - **Port range:** Enter 11211
    - **Source:** Choose the security group with *Lab IDE SG* in the name
  - o Choose **Save rules**.
11. Configure the database subnet group that the Aurora Serverless instance will use, as shown in the following diagram.



- o From the **Services** menu, choose **ElastiCache**.
- o In the left navigation pane, choose **Subnet Groups**.
- o Choose **Create Subnet Group**.
  - Configure the following settings:
    - **Name:** Enter ElastiCacheSubnetGroup
    - **Description:** Enter Subnet Group for ElastiCache
    - **VPC ID:** Choose **IDE VPC**

- Set up the first Availability Zone:
  - **Availability Zone or Outpost:** Choose the first Availability Zone
  - **Subnet ID:** Choose the available subnet ID
  - Choose **Add**
- Set up the second Availability Zone:
  - **Availability Zone or Outpost:** Choose the second Availability Zone
  - **Subnet ID:** Choose the available subnet ID
  - Choose **Add**
- Choose **Create**.

Next, you will deploy an ElastiCache cluster.

### Task 3: Creating the ElastiCache cluster

As with the database deployment in a previous lab, the café has chosen to deploy a managed service instead of manually installing software in a container or on an EC2 instance. ElastiCache offers two engine options, Memcached and Redis. Olivia decided to use Memcached for the coffee suppliers application because the application does not require advanced features that Redis offers. Memcached will be a simple way to get started with caching, and the café could move to Redis later if needed.

In this task, you will again act as Olivia to create an ElastiCache for Memcached cluster.

12. Create an ElastiCache for Memcached instance.
  - o Remain in the ElastiCache console.
  - o In the left navigation pane, choose **Memcached**.
  - o Choose **Create**.
  - o Configure the ElastiCache cluster with the following settings:
    - In the **Create your Amazon ElastiCache cluster** section:
      - **Cluster engine:** Choose **Memcached**
    - In the **Location** section, choose **Amazon Cloud**.
    - In the **Memcached settings** section:
      - **Name:** Enter MemcachedCache
      - **Engine version compatibility:** Choose **1.6.6**
      - **Port:** Enter 11211
      - **Parameter group:** Choose **default.memcached.1.6**
      - **Node type:** Choose **cache.r6g.large (13.07GiB)**

- **Number of nodes:** Enter 3
- In the **Advanced Memcached settings** section:
  - **Subnet group:** Enter ElasticCacheSubnetGroup
  - **Availability zones placement:** Choose **Select zones** and then configure the following zones:
    - **Node1:** Choose the first option in the list (for example, **us-east-1a**)
    - **Node2:** Choose the second option in the list (for example, **us-east-1b**)
    - **Node3:** Choose the second option in the list
  - **Security groups:** Configure the following:
    - Choose the edit (pencil) icon.
    - Deselect the *default* security group.
    - Select the security group that has *Cluster* in the name.
    - Choose **Save**.
- Choose **Create**.

The Memcached cluster will take up to 5 minutes to become available. Move to the next task while the cluster is being created.

You need to make sure that the cache is working correctly. In the next few tasks, you continue as Olivia to create some proof-of-concept Python code to test different data operations.

## Task 4: Loading records into the cache from the database

The first thing you want to test is a simple read operation. Reading from cache is faster than reading from the database, so the coffee suppliers application will try to read from the cache first.

Initially, the cache will be empty and will be loaded when the database is queried. This cache management strategy is called *lazy loading*. Rather than proactively loading the data into cache, data is loaded after the first time that it is requested from the database. You will also configure the time to live (TTL) to limit how long the data will persist in the cache.

As you progress through the scripts, you will notice that the application code must control when the cache gets updated and how long data persists in cache.

13. To change to the directory that holds the proof-of-concept scripts, return to the VS Code IDE Bash terminal, and run the following command:

```
cd ~/environment/python_3
```

14. To install the libraries that Python needs to interact with the database and the cache, run the following commands:

```
sudo dnf install -y mariadb105-devel gcc python3-devel
sudo pip3 install PyMySQL
sudo pip3 install pymemcache
```

In the scripts, you will need to define the connection to both the Aurora database and the Memcached cache. In the following steps, you find the endpoints for these resources.

15. Find the ElastiCache for Memcached endpoint.

- o In the VS Code IDE Bash terminal, run the following command:

```
aws elasticache describe-cache-clusters
```

The output should be similar to the following:

```
{
  "cacheClusters": [
    {
      "CacheClusterId": "memcachedcache",
      "ConfigurationEndpoint": {
        "Address": "memcachedcache.xxxxxxx.cfg.use1.cache.amazonaws.com",
        "Port": 11211
      },
      ...
    }
  ]
}
```

- o In the **ConfigurationEndpoint** section, make a note of the **Address** value. This is the ElastiCache endpoint.

**Note:** You may need to type q to exit back to the terminal.

16. Find the Aurora Serverless cluster endpoint.

- o In the VS Code IDE Bash terminal, run the following command:

```
aws rds describe-db-cluster-endpoints
```

The output should be similar to the following:

```
{
  "DBClusterEndpoints": [
    {
      "DBClusterIdentifier": "supplierdb",
      "Endpoint": "supplierdb.cluster-xxxxxxxxxx.us-east-1.rds.amazonaws.com",
      "Status": "available",
      "EndpointType": "WRITER"
    }
  ]
}
```

- o Make a note of the **Endpoint** value.

17. Review the script that will be used to test loading data into the cache.

- o In the VS Code IDE window, expand the *python\_3* directory, and open the *find\_all.py* file.
- o Review the code.

This code returns all of the coffee bean inventory information.

First, the script queries the Memcached cache. If the *all\_beans* key is found in the cache, the data returned is printed to the console.

```
data = memcached_client.get('all_beans')
```

If no data is found in the cache, the script then queries the Aurora Serverless database.

```
db_query = "SELECT * FROM beans"
mycursor = mydb.cursor()
mycursor.execute(db_query)
```

Any records returned from the database are also added to the cache. That way, the next time that the script is run, the data will be returned more quickly as it will be served from the cache instead of the database.

```
memcached_call = memcached_client.set('all_beans', output_json, TTL_INT)
```

**Note:** Note the time to live (TTL) configuration, which is set by the *TTL\_INT* variable. The *all\_beans* key automatically expires from the cache if it has not been accessed after 3 minutes.

18. Update the script to define the database and cache connections.

- o In the *find\_all.py* file, replace *<FMI\_1>* with the ElastiCache endpoint


address. The updated code should be similar to the following:

```
memcached_client =
base.Client(('memcachedcache.xxxxxxx.cfg.use1.cache.amazonaws.com', 11211))
```

- o Replace *<FMI\_2>* with the Aurora

endpoint. The updated code should be similar to the following:

```
mydb = pymysql.connect(
    "supplierdb.cluster-xxxxxxxxxx.us-east-1.rds.amazonaws.com",
    "nodeapp",
    "coffee",
    "COFFEE"
)
```

- o From the navigation pane, choose  menu, then choose **File > Save**.

19. Test the *find\_all.py* script.



- o First, ensure that the status of the Memcached cluster is *available* before you continue.
- o In the VS Code IDE Bash terminal, run the following command:

```
python3 find_all.py
```

It might take several seconds for the query to respond the first time that you run it.

The output should be similar to the following:

```
current time:- 2021-07-06 20:04:51.715942
Finding all items
Data not found in cache. Retrieving data from the database:
[
  {
    "id": 1,
    "supplier_id": 1,
    "type": "Arabica",
    "product_name": "Best bean",
    "price": "18.00",
    "description": "Delicious, smooth coffee.",
    "quantity": 1000
  },

  {
    "id": 2,
    "supplier_id": 1,
    "type": "Robusta",
    "product_name": "Great bean",
    "price": "12.00",
    "description": "Full bodied, good to the last drop.",
    "quantity": 800
  },

  ..truncated for brevity...

Setting the result in cache
True
current time:- 2021-07-06 20:05:17.784851
```

You can tell from the output that the cache was empty when the script ran. The message also confirms that the *beans* data has been written to the cache.

Make a note of the timestamps from when the script started and when it ended. What do you think will happen when you run the script again before the 3-minute TTL expires?

- o In the VS Code IDE Bash terminal, run the command again:

```
python3 find_all.py
```

The output should be similar to the following:

```
current time:- 2021-07-06 20:06:13.352247
Finding all items
Data returned from cache:
[
  {
    "id": 1,
    "supplier_id": 1,
    "type": "Arabica",
    "product_name": "Best bean",
    "price": "18.00",
    "description": "Delicious, smooth coffee.",
    "quantity": 1000
  },
  {
    "id": 2,
    "supplier_id": 1,
    "type": "Robusta",
    "product_name": "Great bean",
    "price": "12.00",
    "description": "Full bodied, good to the last drop.",
    "quantity": 800
  },
  ..truncated for brevity..
current time:- 2021-07-06 20:06:13.409011
```

Notice that the data is now being retrieved from the cache instead of the database. If you compare the start and end timestamps between the previous run and this one, you will find that the data is coming back much more quickly.

Next, you will test an update to a record. The rest of the test scripts will be very similar to *find\_all.py*.

## Task 5: Updating data

The next operation you need to test is an update to an item in the database. You know that the application will try to read data from the cache before looking for data in the database. This means that you must ensure that the cache continues to return current information when records change in the database. Otherwise, the cache becomes stale, and the application can provide inaccurate results. Imagine a customer thinking that they have ordered an item when it was really out of stock. You don't want unhappy customers. Keep the cache current to prevent this from happening.

20. Review the script that will be used to test an update to a record in the Aurora database.

- o In the VS Code IDE window, expand the *python\_3* directory, and open the *update\_item.py* file.
- o Review the code to understand what it does.

Notice that instead of sending a SELECT statement to the database, the cursor sends an UPDATE statement. The bean item will be updated with the values defined in the *vals* variable.

```
db_query = "UPDATE beans SET supplier_id=%s, type=%s, product_name=%s, price=%s,
description=%s, quantity=%s WHERE id=%s"
vals = (1, "Arabica Arabica", "Best bean EVER", "28.00", "so delicious, smooth
coffee.", 800, 1)
```

The cursor sends both the query and the replacement values to the database.

```
mycursor.execute(db_query, vals)
```

Also notice that, instead of reading the *all\_beans* key from the cache, the cache is deleted.

```
memcached_call = memcached_client.delete('all_beans')
```

This code is using the *write-through* caching strategy because it updates the cache when the database receives a change. This strategy ensures that the data in the cache does not become stale. In this example, the key is completely removed from the cache. The cache will be refreshed with current database records the next time that the *find\_all.py* script is run.

**Note:** Depending on how cache keys are structured, updating an item in the cache rather than purging the key could also be a good option. In this simplified example, the key contains all items, which makes updating a single item more cumbersome.

21. Test the *update\_item.py* script.

- o Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
- o Run the *update.py* script:

```
python3 update_item.py
```

The output should be similar to the following:

```
Updating a bean item
Updated item in RDS
FLUSH the CACHE as it is stale
Purged the cache: True
```

22. Now, to test the cache refresh.

- o Run the *find\_all.py* script:

```
python3 find_all.py
```

The output should be similar to the following:

```
Finding all items
Data not found in cache. Retrieving data from the database:
[
  {
    "id": 1,
    "supplier_id": 1,
    "type": "Arabica Arabica",
    "product_name": "Best bean EVER",
    "price": "28.00",
    "description": "So delicious, smooth coffee.",
    "quantity": 800
  },
  ..truncated for brevity..
```

o Run the script again to ensure that the data is being returned from the cache. The output should be similar to the following:

```
Finding all items
Data returned from cache:
[
  {
    "id": 1,
    "supplier_id": 1,
    "type": "Arabica Arabica",
    "product_name": "Best bean EVER",
    "price": "28.00",
    "description": "So delicious, smooth coffee.",
    "quantity": 800
  },
  {
    "id": 2,
    ..truncated for brevity..
```

Well done! You've created a script that updates the database and also ensures that the cache stays up to date.

Next, you will create a script to test adding a new coffee item to the database.

## Task 6: Creating a new record

In this task, you will create a new item in the database. You will use the *write-through* caching strategy to prevent the cache from becoming stale.

23. Review the script that will be used to test creating a record in the Aurora database.

- o In the VS Code IDE window, expand the *python\_3* directory, and open the *create\_item.py* file.

- o Review the code to understand what it is doing.

The key difference between *update\_item.py* and *create\_item.py* is the database query. The *create\_item.py* script adds a record instead of updating an existing one.

```
db_query = "INSERT INTO beans (supplier_id, type, product_name, price,
description, quantity) VALUES (%s, %s, %s, %s, %s, %s)"
vals = (1, 'Java Java', 'Worlds greatest bean', '38.00', 'Nutty tasting coffee.', 400
)
```

24. Test the *create\_item.py* script.

- o Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
- o Run the *update.py* script:

```
python3 create_item.py
```

The output should be similar to the following:

```
Adding a new bean item
1 record(s) inserted into RDS.
FLUSH the CACHE as it is stale
Purged the cache: True
```

25. Test the cache refresh again.

- o Run the *find\_all.py* script:

```
python3 find_all.py
```

The output should be similar to the following:

```
Finding all items
Data not found in cache. Retrieving data from the database:
[
  {
    "id": 1,
    "supplier_id": 1,
    "type": "Arabica Arabica",
    "product_name": "Best bean EVER",
    "price": "28.00",
    "description": "So delicious, smooth coffee.",
    "quantity": 800
  },
  {
    "id": 2,
    ..truncated for brevity..

```

```

  {
    "id": 15,
    "supplier_id": 1,
    "type": "Java Java",
    "product_name": "worlds greatest bean",
    "price": "38.00",
    "description": "Nutty tasting coffee.",
    "quantity": 400
  }
]
Setting result in cache
True

```

- o Run *find\_all.py* again to see the new item return from the cache:

```
Finding all items
FROM CACHE: [ { id: 1,
  supplier_id: 1,
  type: 'Arabica Arabica',
  product_name: 'Best bean EVER',
  price: '28.00',
  description: 'So delicious, smooth coffee.',
  quantity: 800 },

..truncated for brevity..

{ id: 15,
  supplier_id: 1,
  type: 'Java Java',
  product_name: 'worlds greatest bean',
  price: '38.00',
  description: 'Nutty tasting coffee.',
  quantity: 400 } ]
```

**Note:** If the data is not returning from the cache wait a minute and run the command again.

Nicely done!

In the final test, you will remove a record from the database.

## Task 7: Deleting a record

The final proof-of-concept script tests deleting an item from the database. You will again use the *write-through* strategy to ensure the cache stays in sync with the database.

26. Review the script that will be used to test deleting a record from the Aurora database.

- o In the VS Code IDE window, expand the *python\_3* directory, and open the *delete\_item.py* file.
- o Review the code:

```
db_query = "DELETE FROM beans WHERE id=%s"
val = (14,)
```

Again, the main difference in this script is in the database query. The SQL statement deletes (removes) an item from the database.

You should also recognize the caching pattern in the code. Any time a change is made to the database, the cache will be purged to force a clean refresh during the next read operation.

27. Test the *delete\_item.py* script.

- o Update the *<FMI\_1>* and *<FMI\_2>* placeholders as you did previously.
- o Run the *delete\_item.py* script:

```
python3 delete_item.py
```

The output should be similar to the following:

```
Deleting a bean item
1 record(s) deleted from RDS.
FLUSH the CACHE as it is stale
Purged the cache: True
```

You can test the *find\_all.py* script again to verify that the record was removed from the database and that the cache is refreshing as expected.

Awesome job! You have created and tested a few scripts to prove that the cache and database are working properly.

Nikhil has started helping Sofia with the café's application code. You are going to send this proof-of-concept code to Nikhil so that he can follow these examples and incorporate caching into the coffee suppliers application Node.js code.

## Task 8: Updating the application container

Nikhil quickly updated the coffee suppliers application. His code expanded on the examples from the proof of concept. He restructured the code, and now rather than having a script for each type of operation (SELECT, INSERT, UPDATE, DELETE), one script handles all database calls, and another interacts with the cache. He also modified the code to use an environment variable to identify the ElastiCache for Memcached endpoint.

You will perform the following steps as Nikhil to review the application code that interacts with the cache. Then, you will update the application.

28. Review the new code. Notice how similarly Node.js and Python work with Memcached. Also, recognize the changes that have been made to optimize cache utilization.

**Note:** Don't worry about understanding everything that is happening in the Node.js script. Focus on the larger patterns. It can be helpful to follow the messages that are included in the code.

- o In VS Code IDE, in the Environment window, expand the *resources* directory, and open the *bean.controller\_2.js* file.
- o Locate and review the following lines of code:

```
const memcachedHost = process.env.MEMC_HOST || "localhost";
const memcachedPort = '11211';
const cacheTtlInSec = 300;
```

The code above sets up the endpoint, port, and Time to Live settings for the Memcached connection. The *MEMC\_HOST* value will be populated by an environment variable that you will set up later.



```
return res.render("bean-list-all", {beans: JSON.parse(data), cache_msg: "results from cache"});
```

```
res.render("bean-list-all", {beans: data, cache_msg: "results from db"});
```

The two lines of code above were added to display a note on the website about whether the response came from cache or the database.

```
memcached.set('beans_' + data.id, JSON.stringify(data), cacheTtlInSec, function (err) {
```

The line of code above optimizes the cache usage by storing individual items. To return all items, you can still use the *all\_beans* key. However, to return a single item, you can now use a key that is associated with a specific inventory item. This limits the amount of data that must be read and returned over the network to the application.

```
exports.findOne = (req, res) => {
```

Finally, you added code to select individual keys when it is not necessary to return everything.

29. To update the application code, run the following command:

```
mv ~/environment/resources/bean.controller_2.js \
~/environment/resources/codebase_partner/app/controller/bean.controller.js
```

30. To rebuild and update the Docker image, run the following commands:

```
cd ~/environment/resources/codebase_partner/
npm install
docker build --tag node_app .
```

31. Authorize your Docker client to connect to the Amazon Elastic Container Registry (Amazon ECR) service.

- o Discover your AWS account ID:
  - In the AWS Management Console, in the upper-right corner, choose your user name, which begins with **voclabs/user**.
  - Copy the **My Account** value from the menu. This is your AWS account ID.
- o Return to the VS Code IDE Bash terminal.
- o To authorize your VS Code IDE Docker client, run the following command. Replace with the actual account ID that you just found in the console:

```
aws ecr get-login-password \
--region us-east-1 | docker login --username AWS \
--password-stdin <account-id>.dkr.ecr.us-east-1.amazonaws.com
```

A message indicates that the login succeeded.

32. Push the updated container image to Amazon ECR.

- o To find the URI for the repository, run the following command:

```
aws ecr describe-repositories --query repositories[][repositoryUri] --output text
```

The output should be similar to the following:

```
xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app
```

- o Next, to tag the Amazon ECR image, run the following command. Replace *<FMI\_1>* with the repository URI:

```
docker tag node_app:latest <FMI_1>:latest
```

The updated command should be similar to the following:

```
docker tag node_app:latest xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/cafe/node-
web-app:latest
```

- o Now, to push the image to Amazon ECR, run the following command. Again, replace *<FMI\_1>* with the repository URI:

```
docker push <FMI_1>:latest
```

The updated command should be similar to the following:

```
docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/cafe/node-web-app:latest
```

The output should be similar to the following:

```
8df1a11d2c34: Pushed
3a21fd1e1e85: Pushed
78c260a1577d: Layer already exists
d81d715330b7: Layer already exists
1dc7f3bb09a4: Layer already exists
dcaceb729824: Layer already exists
f1b5933fe4b5: Layer already exists
latest: digest:
sha256:8a9137a92347982bbf368ae500131cf8c7f2c69d227dfe7d7c529c81cba1b5ee size: 1786
```

These messages verify that the existing image was updated.

33. Test your Elastic Beanstalk website.

- o On the search bar at the top, search for and select Elastic Beanstalk.
- o In the list of environments, choose the **MyEnv** link.
- o On the **MyEnv** page, choose the URL directly under **MyEnv** to open the currently deployed application.

The application opens in a new browser tab.

- o Append `/beans` to the URL in the application browser tab.

**Note:** If you receive an error page, wait a few seconds and reload the *beans* page. This error can occur because the database did not respond quickly enough. It should work on the second try.

The application is still using the old code and needs to be updated to use the container that you just pushed to the repository.

- o Keep this browser tab open because you will return to it shortly.

Recall that the updated Node.js application requires an environment variable to set the Memcached endpoint.

34. Configure the new Memcached environment variable, *MEMC\_HOST*.

- o Return to the browser tab with the **MyEnv** page.
- o In the left navigation pane, choose **Configuration**.
- o In the **Software** category, choose **Edit**.
- o Scroll down to the **Environment properties** section, and add the following entry:
  - **Name:** Enter `MEMC_HOST`
  - **Value:** Enter the same Memcached endpoint that you used when testing the Python scripts
- o Choose **Apply**.

A message indicates that *Elastic Beanstalk is updating your environment*.

It takes a few minutes for the environment to be updated. Once the build has finished, you can continue.

- o Revisit the browser tab with the coffee suppliers application, and refresh the page.

A note now appears under **All beans** to indicate when the results were returned from the database and when they were returned from the cache.

35. Test the code update from the café website.

- o Return to the browser tab with the **MyEnv** page.
- o From the **Services** menu, choose **S3**.
- o Locate the bucket that has *s3bucket* in the name and choose the associated link.

- o Choose the **Properties** tab and then scroll down to the **Static website hosting** section.
- o Choose the website endpoint link to open the café's website in a new tab.
- o On the café's website, choose the menu, and then choose **Buy Coffee**.

The list of beans is returned and is the same as the list that was returned when you were testing from the command line.

Congratulations! You have created an ElastiCache for Memcached cluster, configured network security to allow calls to the cache, and updated the application code to use the cache to front end the database call. These actions will speed up performance for the café's customers.

## Submitting your work

36. To record your progress, choose **Submit** at the top of these instructions.

37. When prompted, choose **Yes**.

After a couple of minutes, the grades panel appears and shows you how many points you earned for each task. If the results don't display after a couple of minutes, choose **Grades** at the top of these instructions.

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

38. To find detailed feedback about your work, choose **Submission Report**.

## Lab complete

Congratulations! You have completed the lab.

39. At the top of this page, choose **End Lab**, and then choose **Yes** to confirm that you want to end the lab.

A message panel indicates that the lab is terminating.

40. To close the panel, choose **Close** in the upper-right corner.

## Experiment 08: Implementing CloudFront for Caching and Application Security

### 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** check box.

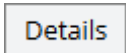
### Task 1: Preparing the development environment

In this first task, you will configure your VS Code IDE. You will also run the script to recreate the work you completed in previous labs.

5. Before proceeding to the next step, verify that the AWS CloudFormation stack creation process for the lab has successfully completed.
- o In a new browser tab, navigate to the **CloudFormation** console.
  - o In the left navigation pane, choose **Stacks**.
  - o For the stack with "ACD\_2.0" in the **Description** column, verify that the **Status** says **CREATE\_COMPLETE**.

⚠ If it doesn't show CREATE\_COMPLETE yet, wait until it does. Because this stack is creating an Amazon Relational Database Service (Amazon RDS) database instance, it might take about 5 minutes to complete.

6. Connect to the VS Code IDE.

- o At the top of these instructions, choose  followed by **AWS: Show**

- o Copy values from the table for the following and paste it into an editor of your choice for use later.

- **LabIDEURL**

- **LabIDEPassword**

- o In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
- o On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.

7. Download and extract the files that you need for this lab.

- o In the same terminal, run the following command:

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/09-lab-cloudfront/code.zip -P /home/ec2-user/environment
```

- o The code.zip file is downloaded to the VS Code IDE. The file is listed in the left navigation pane.
- o Extract the file:

```
unzip code.zip
```

8. Run a script to upgrade the version of Python and the AWS CLI that are installed on the VS Code IDE. The script also recreates the work that you completed in earlier labs into this AWS account.

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

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

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

- o Verify that the script did not encounter any errors:
  - If you see any "An error occurred" lines just before the "done" line, the script might have been run too soon after you started the lab. Run the setup.sh script again to clear errors. If you don't see any error lines, you can assume that the script ran successfully.

**Analysis:** The CloudFormation template that ran when you started this lab created resources in the AWS account. The script you ran also created resources. Between the two of them, the following resources have been created to replicate what you built in previous labs:

- o An S3 bucket with an associated bucket policy. The bucket contains the café website code.
  - o An Amazon DynamoDB table populated with menu data.
  - o A REST API configured using Amazon API Gateway.
  - o An AWS Lambda function that retrieves data from DynamoDB when invoked.
  - o A Memcached cluster that caches supplier data from Amazon Aurora Serverless for the suppliers application. A café/node-web-app Docker image, which is stored in the Amazon Elastic Container Registry (Amazon ECR).
  - o An AWS Elastic Beanstalk environment and application that runs an EC2 instance named *MyEnv*. The EC2 instance hosts a Docker container created from the Docker image that is stored in Amazon ECR.
  - o An Aurora Serverless database running MySQL on Amazon RDS, which contains the *supplierdb* database, which stores coffee supplier information.
9. Verify the version of AWS CLI installed.
- o In the VS Code IDE Bash terminal (at the bottom of the IDE), run the following command:

```
aws --version
```

The output should indicate that version 2 is installed.

10. Verify that the SDK for Python is installed.

- o Run the following command:

```
pip3 show boto3
```

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

- o Keep the VS Code IDE open in your browser. You may use it later in this lab.
11. Note the metadata settings on the objects stored in the S3 bucket, and then verify that you can access the café website.
- o Navigate to the Amazon S3 console.
  - o Choose the link for the bucket that has *-s3bucket* in the name.
  - o Choose the **index.html** link.
  - o Scroll down to the **Metadata** section.

Two key-value pairs are listed. The **Cache-Control** key-value pair has a value of **max-age=0**. This was set when you ran *setup.sh* in an earlier step. Line 21 of that script ran the command `aws s3 cp ./resources/website s3://$bucket/ --recursive --cache-control "max-age=0"` to set this metadata value on every file that it uploaded to the bucket. You'll learn about the significance of this setting later in the lab.

- o At the top of the page, open the **Object URL** in a new browser tab.

The café website displays. 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.

- o Keep the website open in this browser tab, because you will return to it later in this lab.

## Task 2: Configuring a distribution for static website content

In this task, you will configure the café website, which is hosted on Amazon S3, to be available through a CloudFront distribution. With the distribution, you can enable HTTPS access to the website.

**Detailed analysis:** Recall from earlier labs that static website hosting, which would make the site available at `https://s3-website.amazon.com`, is *not* configured on the café website S3 bucket. Instead, you access the website by loading the Object URL of the `index.html` file from the bucket, which is in the form `http://s3.amazonaws.com/index.html`. Therefore, the site is not currently accessed using HTTPS. Sofia hasn't worried about this until now, because she knew she would integrate other AWS services into the website design and would use CloudFront as part of the final application design.

12. Begin to configure a CloudFront distribution for the café website hosted on Amazon S3.

- o Navigate to the CloudFront console.
- o Choose **Create a CloudFront distribution**.

**Tip:** You will configure several settings for the distribution over the next few steps. Pay careful attention not to miss any steps, and don't finish creating the distribution until you are specifically instructed to.

13. Configure the **Origin** settings for the distribution.

- o **Note:** If a setting is not specified in the following steps, use the default.
- o **Origin domain:** Search for and choose the S3 bucket that has `-s3bucket` in the name. This bucket contains the website code.
- o In the **Origin access** settings.
  - Choose **Legacy access identities**.
  - Choose **Create new OAI**.
  - **Name:** Enter `access-identity-cafe-website`
  - Choose **Create**.
  - **Bucket policy:** Choose **Yes, update the bucket policy**.
- o In the **Add custom header** section, choose **Add header** and configure:
  - **Header name:** Enter `cf`
  - **Value:** Enter `1`



**Note:** The custom header is not an important requirement for this lab, but it will be useful in a later lab.

14. Configure the **Default cache behavior** settings.

- o **Path pattern:** Keep the **Default (\*)** setting. This means that all requests will go to the origin.
- o **Viewer protocol policy:** Choose **Redirect HTTP to HTTPS**. This will help users find the website, even if they load the HTTP URL.
- o **Allowed HTTP methods:** Keep the default **GET, HEAD** setting.
- o **Restrict viewer access:** Keep the default **No** setting. This will be a public website.

15. Configure the **Cache key and origin requests** settings.

- o Choose **Legacy cache settings**.
- o Keep the settings for **Headers**, **Query strings**, and **Cookies** as **None**.
- o **Object caching:** Choose **Use origin cache headers**.

**Note:** Recall the metadata key-value pair with key **Cache-Control** and value **max-age=0**. This metadata is set on all of the objects stored in the origin S3 bucket. By choosing **Use origin cache headers** for this distribution, you ensure that this distribution will inherit this setting applied to all of the objects in the Amazon S3 origin.

- o Under **Web Application Firewall (WAF)** choose **Do not enable security protections**.

16. Configure the **Settings** section of the distribution.

- o **Price class:** Keep the default **Use all edge locations** setting.
- o **Alternative domain name (CNAME):** Do not enter anything.

**Note:** You could specify a unique domain name here; however, this lab doesn't have one.

- o **Custom SSL certificate:** Do not enter anything.

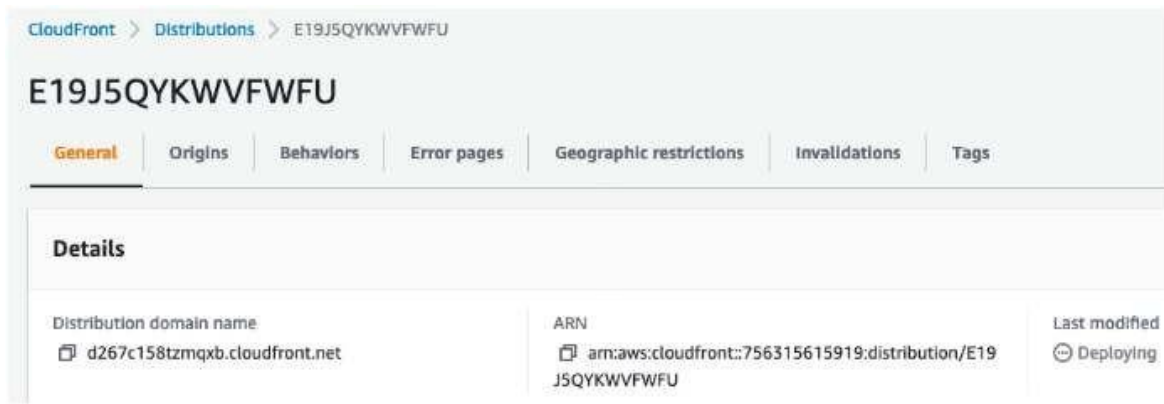
**Note:** If you owned a domain and had a TLS or SSL certificate from a certificate authority, you would upload it here. For this lab, you will use the default CloudFront certificate (\*.cloudfront.net), so you do not need to specify a custom certificate.

- o **Supported HTTP versions:** Select **HTTP/2**.
- o **Default root object:** Enter **index.html**
- o **IPv6:** Off

**Note:** This is the café website homepage hosted in the S3 bucket.

17. Finally, choose **Create distribution**.

While the distribution is being created, **Last modified** at the top of the page displays *Deploying* as shown in the following image.



**Note:** It might take 5 to 15 minutes for the deployment to complete. However, you can continue to the next step in the lab.

18. Update the bucket policy.

- o In a new browser tab, navigate to the Amazon S3 console.
- o Choose the link for the bucket that has *-s3bucket* in the name.
- o Choose the **Permissions** tab.
- o In the **Bucket policy** section, choose **Edit**.
- o In the **Policy** code section, delete lines 4 through 17, which are highlighted in the following image.

```

1 {
2   "Version": "2008-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Principal": "*",
7       "Action": "s3:GetObject",
8       "Resource": [
9         "arn:aws:s3:::c24306a3394781912243t1w756315615919-s3bucket-ipkihimkc4x9/*",
10        "arn:aws:s3:::c24306a3394781912243t1w756315615919-s3bucket-ipkihimkc4x9"
11      ],
12      "Condition": {
13        "IpAddress": {
14          "aws:SourceIp": "172.125.76.41/32"
15        }
16      }
17    },
18    {
19      "Sid": "DenyOneObjectIfRequestNotSigned",
20      "Effect": "Deny",
21      "Principal": "*",

```

- o At the bottom of the page, choose **Save changes**.

19. Retest access to the café website after the update to the bucket policy.

- o Return to the browser tab where the café website is open, and refresh the browser page.

You no longer see the site. Instead, you see an *AccessDenied* error similar to the following image.

```

- <Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>ZGBBEF5HCWWJJB2G</RequestId>
- <HostId>
  HBMGHF7xvk5nmv2CWqYG+p+brl2HzGrm+c1QgVNQMYc7VZ2456lQxAkvjJLgYcVPr4qGHG06+nQ=
</HostId>
</Error>

```

**Note:** This outcome is expected, because you removed the lines from the bucket policy that granted `s3:GetObject` access for requests coming in from your IP address. You don't want anyone to access the website directly using the Amazon S3 URL. Instead, you want users to access the site through CloudFront.

- o Close the browser tab where the café website is open.
20. Verify that the CloudFront distribution is now enabled.
- o Return to the CloudFront console.
  - o In the left navigation pane, choose **Distributions**.
  - o Verify that the distribution you created now has a **Status** of *Enabled*, as shown in the following image.



⚠ If the status is not *Enabled* yet, wait until it is before moving to the next step.

21. Test the CloudFront distribution.
- o Choose the link for the distribution ID.
  - o Copy the **Distribution domain name** URL and open it in a new browser tab.

**Note:** The URL ends in *.cloudfront.net*.

The café website displays.

- o Try to load the website on a mobile device that is *not* connected to the same network as your computer. For example, the device is connected to the internet through a cellular connection and not the same WiFi network that your computer is connected to.

Notice that you can still access the site.

**Tip:** If you don't have a mobile device, an alternate way to test that the café website is available from another network is to use the VS Code IDE. In the terminal, run `wget <distribution-domain-name>` where `<distribution-domain-name>` is the distribution URL. If the HTTP request returns an HTTP status code of 200, then the site is available from the VS Code IDE, which runs on a different network than your computer.

Congratulations! You successfully created a CloudFront distribution. The website is running on a secure HTTPS connection, and you secured the site so that it is only available through CloudFront.

### Task 3: Securing network access to the distribution using AWS WAF

In this task, you will configure the website so that it can only be accessed from a specific IP address range again. In the café scenario, this is the IP address range that Sofia uses to connect to the internet.

Recall that the S3 bucket policy enforced this previously. Now that the website is configured to be accessed through CloudFront, Sofia needs to find an alternative way to implement this. She will use the AWS WAF service to create an access control list (ACL) to restrict access to the café website.

22. First, create an IP set for your IP address.

- o In the AWS Management Console, search for waf and choose **WAF & Shield**.
- o In the left navigation pane, choose **IP sets**.
- o Choose **Create IP set** and configure:
  - **IP set name:** Enter office
  - **Description:** Enter office IP
  - **Region:** Choose **Global (CloudFront)**.
  - **IP addresses:** Enter <ip-address>/32 where <ip-address> is your public IPv4 address, as identified by [whatismyipaddress.com](https://whatismyipaddress.com).

**Note:** Be sure to include the /32 at the end of the IP address.

- Choose **Create IP set**.

23. Begin to create a web ACL.

- o In the left navigation pane, choose **Web ACLs**.
- o Choose **Create web ACL**.
- o In the **Web ACL details** section, configure:
  - **Resource type:** Choose **CloudFront distributions**.
  - **Name:** Enter cafe-website-office-only-during-dev
  - **Description:** Enter Allow access to the cafe website through CloudFront from the cafe office
  - **CloudWatch metric name:** Enter cafe-website-office-only-during-dev
- o In the **Associated AWS resources** section, configure:
  - Choose **Add AWS resources**.
  - Select the CloudFront distribution that you created.
  - Choose **Add**.
  - Select the CloudFront distribution again, and then choose **Next**.

24. Add a rule to the web ACL configuration to allow requests from the *office* IP set.
  - o In the **Rules** section, choose **Add rules, Add my own rules and rule groups**.
  - o **Rule type**: Choose **IP set**.
  - o **Name**: Enter `only_office_please`
  - o **IP set**: Choose the *office* IP set that you just created.
  - o **IP address to use as the originating address**: Keep the default **Source IP address** setting.
  - o **Action**: Choose **Allow**.
  - o Choose **Add rule**.
25. Update the new web ACL rule to block any requests that don't match the rule.
  - o In the **Rules** section, select the `only_office_please` rule.
  - o In the **Default web ACL action** section, for **Default action**, choose **Block**.
  - o Choose **Next**.
26. Set the rule priority, configure metrics, and create the web ACL.
  - o Choose the `only_office_please` rule.
  - o Choose **Next**.
  - o Keep all of the default metrics settings, and choose **Next** again.
  - o Review the settings, and at the bottom of the page, choose **Create web ACL**.
27. Confirm that the web ACL configuration has been applied to the CloudFront distribution.
  - o Return to the CloudFront console.
  - o In the left navigation pane, choose **Distributions**.
  - o **Note**: The **Last modified** column might display *Deploying* for the distribution. The deployment will complete within about 5 minutes; however, you can proceed to the next step without waiting.
  - o Choose the link for the distribution ID.
  - o In the **Security** section, notice that the distribution now has an **AWS WAF** value.
28. Test the AWS WAF configuration that was applied to the CloudFront distribution.
  - o Return to the browser tab where the café website is open, and refresh the browser page.

The website displays, because your computer's IP address is in the IP set that you specified when you configured the web ACL.

**Tip:** To open the café website again, navigate to the CloudFront console. Open the details page for the distribution and locate the **Distribution domain name**. Enter that URL into a new browser tab.

- o Next, try to load the website on a mobile device that is *not* connected to the same network as your computer. For example, the device is connected to the internet through a cellular connection and not the same WiFi network that your computer is connected to.

You cannot access the site through a different network now.

**Tip:** If you don't have a mobile device, an alternate way to test is to use the VS Code IDE. In the terminal, run `wget <distribution-domain-name>` where `<distribution-domain-name>` is the distribution URL. If the HTTP request returns an HTTP status code of 403, then the site is not available from the VS Code IDE, which runs on a different network than your computer.

Congratulations! You have blocked direct access to the website through Amazon S3, configured a global CDN, and also secured the website to prevent anyone who isn't using your IP address from viewing the café website during this development phase.

**Analysis:** At this point, because your CloudFront distribution URL has an obscure value, someone would need to both guess the URL and use your IP address to access the website. Doing this would be difficult for anyone other than yourself. However, to truly secure the site, you should configure a login system. In a later lab in this course, you will use the Amazon Cognito service to implement authentication and to secure parts of the website.

## Task 4: Securing a REST API endpoint using AWS WAF

The café website is now configured so that it can only be viewed from the café office network (your IP address). However, the REST API URLs that the website's AJAX use are not yet secured and could be invoked from anywhere on the internet.

In this task, you will secure access to one of the REST API endpoints.

29. Create a *regional* AWS WAF IP set.

- o Return to the WAF & Shield console.
- o In the left navigation pane, choose **IP sets**.
- o Choose **Create IP set** and configure:
  - **IP set name:** Enter `office_regional`
  - **Description:** Enter IP of the office for API Gateway
  - **Region:** Choose **US East (N. Virginia)**.
  - **IP addresses:** Enter `<ip-address>/32` where `<ip-address>` is your public IPv4 address, as identified by `whatismyipaddress.com`.

**Note:** Be sure to include the `/32` at the end of the IP address.

- o Choose **Create IP set**.

30. Begin to create a *regional* web ACL.

- o In the left navigation pane, choose **Web ACLs**.
- o Choose **Create web ACL**.
- o In the **Web ACL details** section, configure:

- **Resource type:** Choose **Regional resources**.
  - **Region:** Choose **US East (N. Virginia)**.
  - **Name:** Enter `website-api-gw-office-only-during-dev`
  - **Description:** Enter `To allow us to access the API GW calls used by the website from the office`
  - **CloudWatch metric name:** Enter `website-api-gw-office-only-during-dev`
  - o In the **Associated AWS resources** section, configure:
    - Choose **Add AWS resources**.
    - For Resource type, select **Amazon API Gateway**.
    - Select the **ProductsApi - prod** API Gateway resource.
    - Choose **Add**.
    - Select the **ProductsApi - prod** resource again, and then choose **Next**.
31. Add a rule to the web ACL configuration to allow requests from API Gateway.
- o In the **Rules** section, choose **Add rules, Add my own rules and rule groups**.
  - o **Rule type:** Choose **IP set**.
  - o **Name:** Enter `ip_for_apigw`
  - o **IP set:** Choose the *office\_regional* IP set that you just created.
  - o **IP address to use as the originating address:** Keep the default **Source IP address** setting.
  - o **Action:** Choose **Allow**.
  - o Choose **Add rule**.
32. Update the new web ACL rule to block any requests that don't match the rule.
- o In the **Rules** section, select the **ip\_for\_apigw** rule.
  - o In the **Default web ACL action** section, for **Default action**, choose **Block**.
  - o Choose **Next**.
33. Set the rule priority, configure metrics, and create the web ACL.
- o Choose the **ip\_for\_apigw** rule.
  - o Choose **Next**.
  - o Keep all of the default metrics settings, and choose **Next** again.
  - o Review the settings, and at the bottom of the page, choose **Create web ACL**.
34. When the web ACL creation process is complete, check the resources associated with the ACL.
- o Choose the link for the *website-api-gw-office-only-during-dev* ACL, which you just created.

- o Choose the **Associated AWS resources** tab.
- o Confirm that the *ProductsApi - prod* resource is listed. If it isn't, add it:
  - Choose **Add AWS resources**.
  - Choose the *ProductsApi - prod* resource.
  - Choose **Add**.

35. Test the new ACL from your computer.

- o In a new browser tab, go to the API Gateway console.
- o Choose the link for the **ProductsApi**.
- o In the left navigation pane, choose **Stages**.
- o In the **Stages** navigation pane, expand the **prod** stage.
- o Under **/bean\_products**, choose **GET**.
- o Copy the **Invoke URL**, which has the format *https://execute-api.us-east-1.amazonaws.com/prod/bean\_products*, and load the URL in a new browser tab.

A JSON-formatted document with product information displays. This is the expected behavior.

36. Test the new ACL from another network.

- o On the browser tab on your *computer* where you just opened the invoke URL, open the context menu (right-click) for the page and choose **Create QR Code for this page** as shown in the following image.



**Tip:** These instructions are for the Google Chrome browser. Other browsers might not provide this feature. If this feature is not available, see [Alternative steps](#).

- o A pop-up window displays a QR code as shown in the following image. Keep the window open and continue to the next step.





- Verify that your mobile device is *not* connected to the same network as your computer.
- On your mobile device, use the camera app or a QR Code reader app to scan the QR Code on your computer.
- The app prompts you to load the link.

The page displays `{"message": "Forbidden"}`. This is the expected behavior.

- **Alternative steps:** If you don't have a mobile device or cannot create a QR code from your browser, an alternate way to test is to use the VS Code IDE. In the terminal, run `wget <invoke-URL>` where `<invoke-URL>` is the **Invoke URL** for GET for the `/bean_products` API. It should return a 403 Forbidden response message.

Congratulations! The AJAX URI that the café website uses is now also secured so that it can only be invoked from the café network (your computer). In a later lab, you will secure access to the coffee suppliers web application.

**Analysis:** To use AWS WAF to make the suppliers application available only from the café's office IP address, Sofia would need to use an Application Load Balancer. To use a load balancer, she would need to upgrade the Elastic Beanstalk application. However, because the suppliers website is not meant for public access, even when the website moves to production, this method would not be sufficient to try to secure the website. Sofia decides to leave the suppliers website configuration as it is for now. In a later lab, she will implement authentication to properly secure that part of the site

## How the café plans to use a CloudFront function

**Note:** This section explains how the café will use a CloudFront function. You don't have any steps to complete until you get to the next task.

Frank is pleased that the café website has been secured and cannot be accessed outside of the café's office network during this development phase. However, it seems that an effective cloud developer's work is never done because Frank has one more request.

He can't decide which promotional image to use on the homepage. He likes the apple pie image, but he also likes the pictures of his homemade pastries. He has asked Sofia (you) if it's possible to randomly display an image from a collection when the site is loaded.

Sofia first considered using JavaScript code on the client side to randomize the image. However, she chatted with Faythe, the AWS developer who often comes into the café to get a morning coffee. Faythe mentioned that Sofia could use the CloudFront Functions feature to achieve the same result.

Sofia could configure a CloudFront function to be invoked each time a request comes into (or out of) CloudFront. Therefore, she could manage the randomize logic at the edge rather than bloating the website with randomizing code logic.

To invoke the function, Sofia can add code such as the following to the website:

```
function changeImage(){
    var pastry_name_str = "apple_pie";

    if(document.cookie !== "" && document.cookie.split('=')[1] !== ""){
        pastry_name_str = document.cookie.split("=")[1];
    }
    $('[data-role2='special_highlight']")
        .css({
            "background-image": "url(/images/items/" + pastry_name_str + ".png)"
        });
}
```

This code checks for the existence of a cookie, which CloudFront Functions will add. If the cookie exists, the website will dynamically swap the image to display.

The CloudFront Functions code has already been written and made available for you in the *resources/website/scripts/main.js* file, so you don't need to update the website code in this lab. However, you will need to send a special cookie to the CloudFront distribution. The cookie will reference an image that can be used to override the default image on the website. If the cookie isn't found, the website will display the apple pie image.

## Task 5: Configuring a CloudFront function on the website

In this final task, you will create a new CloudFront function. You will add code to the function to run each time that the café website is loaded. You will then test the website to verify that the desired result has been achieved.

37. Create a CloudFront function.
  - o Navigate to the CloudFront console.
  - o In the left navigation pane, choose **Functions**.
  - o Choose **Create function**.
  - o For **Name**, enter `random_image_header`

- o Choose **Create function**.
- o In the **Function code** section, replace the existing code with the following code:
- In the code you pasted, replace the **distro\_str** value of dp880v3pwdoto on line 15, with your unique distribution string value.

**Tip:** To find this value, if you have the café website open in your browser, copy the portion of the site's URL between *https://* and *.cloudfront.net*. An alternative is to look up the value in the CloudFront console.

- To apply the update, choose **Save changes**.

```
attr_str = "Secure: Path=/: Domain=" + distro_str + ".cloudfront.net: Expires=" +
function handler(event) {
  var
    pastry_name_arr = [
      "apple_pie",
      "apple_pie_slice",
      "chocolate_chip_cupcake",
      "strawberry_cupcake",
      "blueberry_jelly_doughnut",
      "plain_bagel"
    ],
    random_int = Math.floor(Math.random() * (pastry_name_arr.length)),
    response = event.response,
    date = new Date(),
    attr_str = "",
    distro_str = "dp880v3pwdoto"; //change this

    date.setTime(+ date + (365 * 86400000)); //24 \* 60 \* 60 \* 100
```

- Analyze the function code. Notice that it generates a response cookie that contains a value from the `pastry_name_array`.

38. Test the function.

- o Choose the **Test** tab.
- o For **Event Type**, choose **Viewer Response**.
- o Keep the other default settings, and at the bottom of the page, choose **Test function**.
- o A results area appears at the bottom of the page. Confirm that the **Status** is *200 OK*. The output should show that a single random item from the `pastry_name_array` was returned in the cookie.

**Note:** No log results are expected.

39. Publish the CloudFront function, and associate it with the CloudFront distribution.

- o Choose the **Publish** tab.
- o Choose **Publish function**.

A message at the top of the page indicates that the *random\_image\_header* function was successfully published.

- o In the **Associated distributions** section, choose **Add association** and configure:
  - **Distribution**: Choose the distribution.
  - **Event type**: Choose **Viewer Response**.
  - **Cache behavior**: Select **Default (\*)**.
  - Choose **Add association**.
- o In the left navigation pane, choose **Distributions**. Notice that the distribution is being deployed again.

40. Test the functionality on the café website.

- o Return to the café website browser tab, and refresh the page a few times. Each time you refresh, the graphic that appears to the right of the cup of coffee should change. Congratulations! You have successfully implemented a CloudFront function on the website.

## Task 6: Adjusting the cache duration

Recall from earlier in the lab that all objects in the website S3 bucket have a metadata key-value for **Cache-Control** with the value set to **max-age=0**. In addition, recall that when you configured the CloudFront distribution, you chose to use the legacy cache settings and let the origin cache headers determine the object caching behavior. You also configured the CloudFront distribution with a path pattern of \*, which means that the distribution will forward all object requests to the origin (the S3 bucket).

During development, these configurations helped you to immediately notice the effect of any changes made. However, now that the distribution has been tested and found to be stable, Sofia has decided to adjust the caching settings.

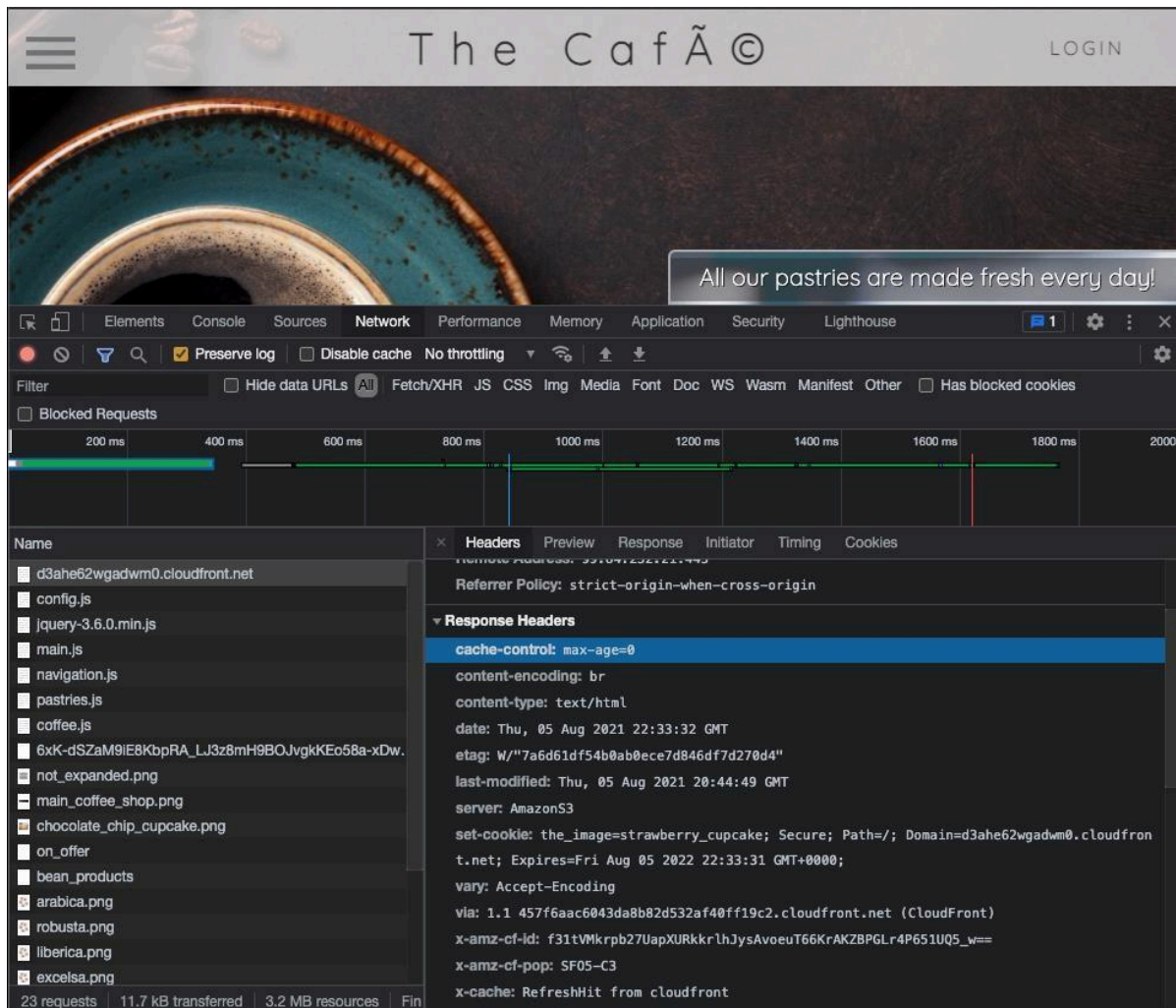
41. Confirm the cache settings that are in place on the café website.
- o Return to the café website browser tab.
  - o Open the context menu (right-click) on the page and choose **Inspect** (if using Chrome) or **Inspect Element** (if using Firefox).
  - o In the developer tools area that appears, choose the **Network** tab.
  - o Next, refresh the café website by using the browser's refresh icon. A list of all the files and accessed locations displays in the developer tools area.
  - o At the top of the file list, choose the CloudFront distribution URL (the URL ends in *cloudfront.net*).

**Note:** Depending on your browser, you might need to choose **All** to see a cloudfront.net URL entry.

- o Analyze the **Response Headers** information.
  - Notice the header **cache-control: max-age=0**.

- Also notice the header **x-cache: RefreshHit from cloudfront**. (In Firefox, it says **Miss from cloudfront**.) This means that a matching cache key wasn't found in the cache.

The following image shows the Response Headers area in the browser developer tools.



Sofia has a functional CloudFront distribution, but the configuration isn't taking advantage of the caching features. She could remove the Cache-Control metadata from all of the S3 objects and then update the behavior settings in the CloudFront distribution to use the CloudFront *CachingOptimized* managed policy. That would apply a default time to live (TTL) of 86,400 seconds (24 hours) to the requested objects. However, that cache setting would be too long for testing purposes.

Sofia decides to begin by applying a caching file expiration time of 3 minutes for testing purposes. She decides to make this change by updating the origin response headers. In this case, the origin is the S3 bucket, and the response headers are configured by updating the Cache-Control key-value pairs set on the Amazon S3 objects. After Sofia finishes testing, she can remove the Cache-Control metadata from the objects in the S3 bucket and use the CloudFront cache settings to control the cache file expiration behavior.

You need to use AWS CLI for this.

42. Edit the Cache-Control header set on each object in the S3 bucket.
  - o Go to the VS Code IDE bash terminal and run the following command:

```

BUCKET_NAME=$(aws s3api list-buckets --query "Buckets[?contains(Name,
's3bucket')].Name" --output text)
echo $BUCKET_NAME
aws s3 ls s3://$BUCKET_NAME/ | awk '{print $4}' | xargs -I {} aws s3api copy-
object \
    --bucket $BUCKET_NAME \
    --copy-source $BUCKET_NAME/{} \
    --key {} \
    --cache-control "max-age=180" \
    --metadata-directive REPLACE

```

This command updates *cache-control* setting for all objects in the bucket. Notice the multiple files getting updated and output similar to the following:

```

{
  "ServerSideEncryption": "AES256",
  "CopyObjectResult": {
    "ETag": "\"82e9c8cb72608751aee8077bc72df441\"",
    "LastModified": "2025-01-10T00:06:21+00:00"
  }
}

```

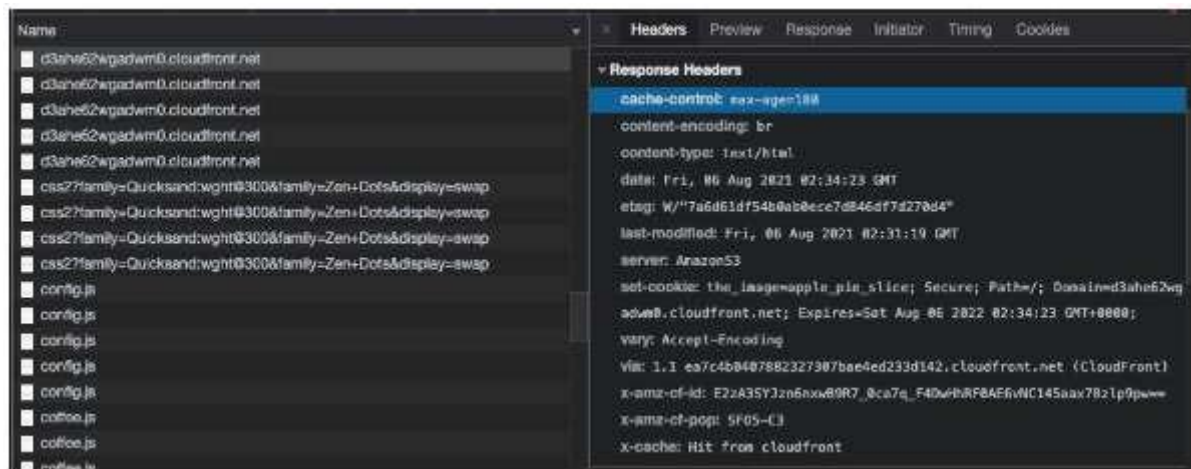
- o Navigate to the Amazon S3 console.
- o Choose the link for the bucket that has *-s3bucket* in the name.
- o Select hyperlink for any file to open its properties
- o Go to the **Metadata** section, you should see a *Cache-Control* property with a value *max-age=180*

43. Test the effects of updating the caching settings on the CloudFront origin (the S3 bucket).

- o Reload the café website.
- o In the developer tools area, notice that the new max-age setting of 180 seconds was applied to the cache.
- o Notice that x-cache now says **Hit from cloudfront**, as shown in the following image.

A hit indicates that the cache key matches the request, and the object was served from the CloudFront edge cache.

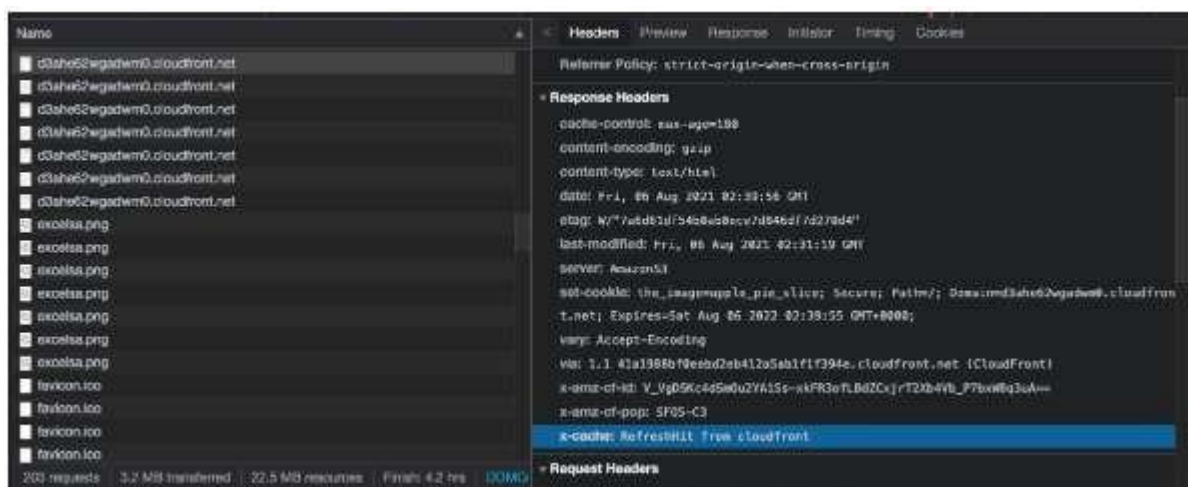




44. Wait 3 minutes, and then test again.

- o Reload the café website.
- o Notice that x-cache now says **RefreshHit from cloudfront**, as shown in the following image.

A refresh hit means the objects were in the edge cache, but the cached objects were expired. Therefore, CloudFront had to get a new copy from the origin.



- o Refresh the webpage a few times to observe how it behaves.
- o Also notice that the image at the top right of the page still changes on each page refresh, even though the page is fully cached.

## Submitting your work

45. 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** checkbox. This action will ensure that the lab instructions remain visible after you choose **Submit**.

46. 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.

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

## **Lab complete**

Congratulations! You have completed the lab.

48. 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.*

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



## Experiment 10: Automating Application Deployment Using a CI/CD Pipeline

### 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** check box.

### Task 1: Preparing the development environment

5. Before you can start working on this lab, you must import some files and install some packages in the Visual Studio Code Integrated Development Environment (VS Code IDE) that was prepared for you.
- o Connect to the VS Code IDE.
    - At the top of these instructions, choose Details followed by **AWS: Show**
    - Copy values from the table for the following and paste it into an editor of your choice for use later.
      - **LabIDEURL**
      - **LabIDEPassword**
      - In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
      - On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.
6. Before proceeding to the next step, verify that the AWS CloudFormation stack creation process for the lab has successfully completed.

- o In a new browser tab, navigate to the CloudFormation console.
- In the left navigation pane, choose **Stacks**.
- For *all three* stacks, verify that the **Status** says *CREATE\_COMPLETE*.

⚠ If any stack doesn't yet have that status, wait until it does. It might take about 12 minutes to complete from the time that you chose **Start Lab**.

7. Download and extract the files that you need for this lab.

- o In the VS Code IDE bash terminal (at the bottom of the IDE), run the following command:

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/13-lab-ci-cd/code.zip -P /home/ec2-user/environment
```

- o Notice that a **code.zip** file was downloaded to the VS Code IDE. The file is listed in the Environment window.
- o Extract the file:

```
unzip code.zip
```

8. Run a script that upgrades the version of the AWS CLI installed on the VS Code IDE. The script also re-creates the work that you completed in earlier labs into this AWS account.

**Note:** This script deploys the architecture that you created across the previous labs. This includes populating and configuring the S3 bucket that the café website uses. The script creates the Amazon DynamoDB table and populates it with data. The script also re-creates the REST API that you created in the Amazon API Gateway lab and deploys the containerized coffee suppliers AWS Elastic Beanstalk application.

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

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

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

- o When prompted for an email address, enter one that you have access to as you complete the lab.

9. Verify the version of AWS CLI installed.

- o In the VS Code IDE Bash terminal, run the following command:

```
aws --version
```

The output should indicate that version 2 is installed.

10. Verify that the SDK for Python is installed.

- o Run the following command:

```
pip3 show boto3
```

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

Keep the VS Code IDE open in your browser. You will use it later in this lab.

## Task 2: Creating a CodeCommit repository

When a developer manages code in their local working environment, it is difficult to track and manage changes. This approach also limits the ability for multiple developers to collaborate on the same codebase.

AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. CodeCommit makes it easy for teams to securely collaborate on code with contributions encrypted in transit and at rest.

In this task, you will create a CodeCommit repository to host the café website code.

11. Create a CodeCommit repository to host the codebase.

- o In the search bar at the top, search and select CodeCommit to open the AWS CodeCommit service console.
- o From the navigation pane, choose **Create repository**.
- o Configure the following settings:
  - **Repository name:** Enter front\_end\_website
  - **Description:** Enter Repository for the cafe website front end
  - Keep the rest of the default settings.
- o Choose **Create**.

12. Create a test file.

- o In the **front\_end\_website** section, choose **Create file**.
- o Copy and paste the following code into the **front\_end\_website** text box:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test page</title>
  </head>
  <body>
    <h1>
      This is a sample HTML page.
    </h1>
  </body>
</html>
```

- o In the **Commit changes to main** section, configure the following settings:
  - **File name:** Enter test.html
  - **Author name:** Enter your name.
  - **Email address:** Enter the same email address that you used in Task 1.
  - **Commit message:** Enter This is my first commit.
  - Choose **Commit changes**.

13. Review your commit.

- o In the left navigation pane, under **Repositories**, choose **Commits**.
- o Choose the link for the commit ID. Only one should be listed.
- o Review the information about this commit.

**Note:** On this page, you see the author of the commit, the commit message, the date of the commit, and the name and contents of the file that was added.

14. Add a comment to the committed file.

- o In the **test.html** section, hover on the plus sign (+) icon on line 4.
- o To the left of the line number, a comment icon appears. Choose the icon.

**Note:** The comment icon is a small box that contains an ellipsis.

- o In the **New comment** text box, enter: I must add a better title at some point. The following image shows the comment text box.



- o Choose **Save**.

Now that you have created a CodeCommit repository to host and manage your code changes, you can use it as a source to automate publishing updates to the café website.

### Task 3: Creating a pipeline to automate website updates

So far, you have been running commands from VS Code IDE to update the code in the S3 bucket for the website. In this task, you will configure a CI/CD pipeline by using CodePipeline to automate the website updates. The pipeline will use the code in your CodeCommit repository to deploy changes to the website's S3 bucket.

15. Return to the VS Code IDE.
16. In the Explorer section, expand *Environment*, which is located in the upper-left corner.
17. Expand the **resources** folder, and open the file named **cafe\_website\_front\_end\_pipeline.json**.

This file defines configuration that will be used to deploy your new pipeline. Review the following code snippets to understand how the pipeline is configured.

The following lines declare the AWS Identity and Access Management (IAM) role that will be associated with the pipeline.

```

"pipeline": {
  "roleArn": "arn:aws:iam::<FMI_1>:role/RoleForCodepipeline",

```

The following code snippet defines the *Source* that your pipeline will use to create and update your application. In this case, the source is your CodeCommit repository, *front\_end\_website*. Note that the pipeline will be configured to use the *main* branch.

```

{
  "name": "Source",
  "actions": [
    {
      "inputArtifacts": [],
      "name": "Source",
      "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "version": "1",
        "provider": "CodeCommit"
      },
      "outputArtifacts": [
        {
          "name": "MyApp"
        }
      ],
      "configuration": {
        "RepositoryName": "front_end_website",
        "BranchName": "main"
      },
      "runOrder": 1
    }
  ]
}

```

The following section of code defines the *Deploy* stage. The deployment will update code in Amazon S3. The *configuration* settings define details about the deployment target. In this case, this section defines the S3 bucket name, configures files to be extracted from a .zip file, and sets a caching policy.

```
{
  "name": "Deploy",
  "actions": [
    {
      "inputArtifacts": [
        {
          "name": "MyApp"
        }
      ],
      "name": "Cafewebsite",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "version": "1",
        "provider": "S3"
      },
      "outputArtifacts": [],
      "configuration": {
        "BucketName": "<FMI_2>",
        "Extract": "true",
        "CacheControl": "max-age=14"
      },
      "runOrder": 1
    }
  ]
}
```

The final section of the code defines the *artifactStore*. This is the S3 bucket where CodePipeline artifacts will be stored.

```
"artifactStore": {
  "type": "S3",
  "location": "codepipeline-us-east-1-<FMI_1>-website"
},
"name": "cafe_website_front_end_pipeline",
"version": 1
}
```

18. Update the `cafe_website_front_end_pipeline.json` file:

- o Replace the two `<FMI_1>` placeholders with the your AWS account ID.

**Note:** To find your account ID, run the following command: `aws sts get-caller-identity`

- o Replace the `<FMI_2>` placeholder with the name of the bucket that has *s3bucket* in the name.

**Note:** To retrieve a list of the S3 buckets in your account, run the following command: `aws s3 ls`

- o From the navigation pane, choose menu, then choose **File > Save**.

19. To create the pipeline, run the following commands.

```
cd ~/environment/resources
aws codepipeline create-pipeline --cli-input-json file://cafe_website_front_end_pipeline.json
```

The command returns output similar to the following example:

```
{
  "pipeline": {
    "name": "cafe_website_front_end_pipeline",
    "roleArn": "arn:aws:iam::111122223333:role/RoleForCodepipeline",
    "artifactStore": {
      "type": "S3",
      "location": "codepipeline-us-east-1-111122223333-website"
    },
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "name": "Source",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "provider": "CodeCommit",
              "version": "1"
            }
          }
        ]
      }
    ]
  }
}
```

You may need to press the Q key to return to the command prompt.

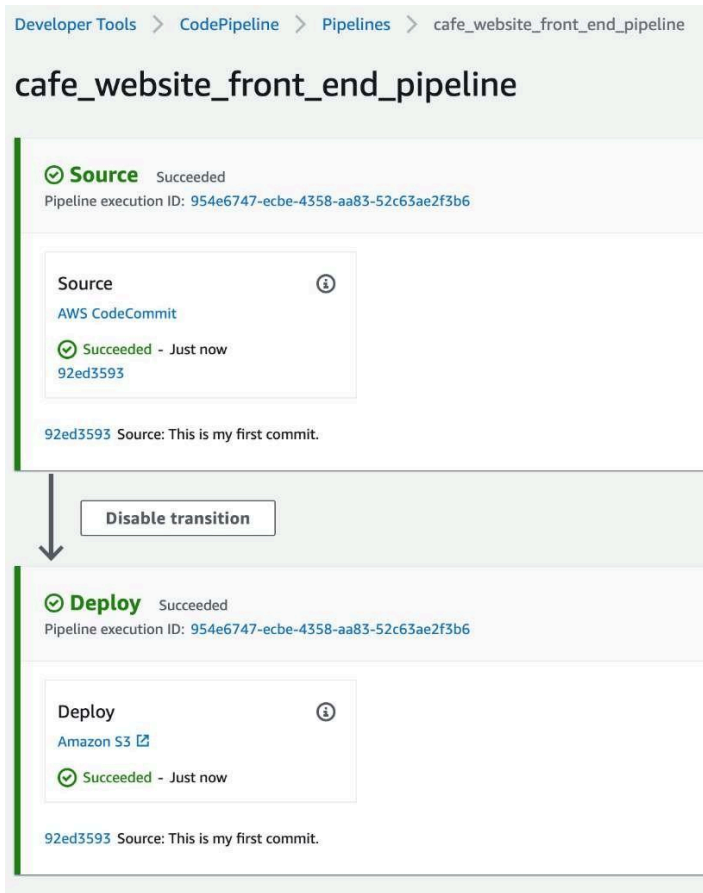
If you receive an error, double check your account ID and the bucket name that you used to update the placeholders. Correct any typos in the `cafe_website_front_end_pipeline.json` file and run the command again.

20. Navigate to the CodePipeline console.

The **Pipelines** section lists the **cafe\_website\_front\_end\_pipeline** Pipeline.

21. Choose the **cafe\_website\_front\_end\_pipeline** hyperlink and review the pipeline status, as shown in the following image.





The pipeline should deploy successfully. Code was deployed using CodeCommit as the source and the café website S3 bucket as the target. This means the bucket should have been updated with the *test.html* file.

22. Verify the automated deployment.

- o Return to the VS Code IDE bash terminal.
- o To find your Amazon CloudFront distribution domain name, run the following command:

```
aws cloudfront list-distributions --query DistributionList.Items[0].DomainName --output text
```

- o Update the following URL by replacing *<cloudfront\_domain>* with the value that was returned by the previous command: [https://<cloudfront\\_domain>/test.html](https://<cloudfront_domain>/test.html)

The updated URL is similar to <https://aaabbb111222.cloudfront.net/test.html>.

- o Open a new browser tab, and enter the URL that you just created. You reach a sample webpage similar to the following:



**This is a sample HTML page.**

- o Keep this browser tab open. You will return to it later.

Well done! Now, when you update the repository, the pipeline will automatically update your website.

Next, you will clone the repository to your VS Code IDE. This will provide the ability to edit the files locally and synchronize with your centralized CodeCommit repository.

#### Task 4: Cloning a repository in VS Code IDE

It's more efficient to edit code in an IDE than it is to use the CodeCommit console. In this task, you will clone a local copy of the repository in your VS Code IDE work environment.

23. Retrieve the SSH clone URL for your repository.

- o Navigate to the CodeCommit console.
- o In the navigation pane, choose **Repositories**.
- o In the **Clone URL** column, choose **Clone HTTPS(GRC)** to copy the URL to your clipboard.

Note: URL should look similar to *codecommit::us-east-1://front\_end\_website*

24. Clone your repository using the SSH URL.

- o Return to the VS Code IDE terminal.
- o Run the following command to clone the repository to your VS code IDE, replace the *<<Clone URL>>* with the value you copied.

```
cd ..  
git clone <<Clone URL>>
```

Note the output similar to below

```
Cloning into 'front_end_website'... remote: Counting objects: 3, done.Unpacking objects: 100% (3/3),  
290 bytes | 290.00 KiB/s, done
```

- o Note that the file is also visible in the explorer




You now have a local clone of your repository that can be edited using your IDE. Next, you will learn how to manage your local copy of the repository and synchronize changes with CodeCommit.

## Task 5: Exploring the Git integration with the VS Code IDE

You can interact with the repository through the command line, but VS Code provides Git integration in the IDE to make it easier to manage your repository. In this task, you will perform simple repository management operations by using this integration.

25. Explore repository branch management.

- o Locate the branch icon,  which is located in the lower-left corner of the IDE.
  - Source Control option is opened.
- o From the top, choose three dots to the right of *SOURCE CONTROL*.
- o From the menu displayed, choose **Source Control Repositories**
- o Your repository *front\_end\_website* is

displayed. The IDE is currently set up to communicate with the **main** branch.

26. Explore and edit the repository files.


Remember that the repository was cloned to the local folder *environment/front\_end\_website*. You can open repository files in your IDE to view and edit them.

- o From the explorer menu, expand the **front\_end\_website** folder to reveal the **test.html** file, which is shown in the following image.




- o Open the **test.html** file and edit the page title on line 4. Replace the current title with the following text:

Best test page ever.

- o Save your changes.
- o Number **1** now appears next to **branch** icon in the left pane. This indicates that changes have been made to the code that need to be committed to the branch.
- o Hover over  and you will notice a message **\*Source Control 1 pending change\***

**Note:** This is because you made a change to the file but it is not yet committed to the repository main branch.

27. Commit your changes to the **main** branch.

- o Choose the  icon to open the **Source Control**.

**Note:** You don't have to use the IDE integration. You could also use Git from the command line. For example, if you enter the following command, you should find the *test.html* file listed in the output.

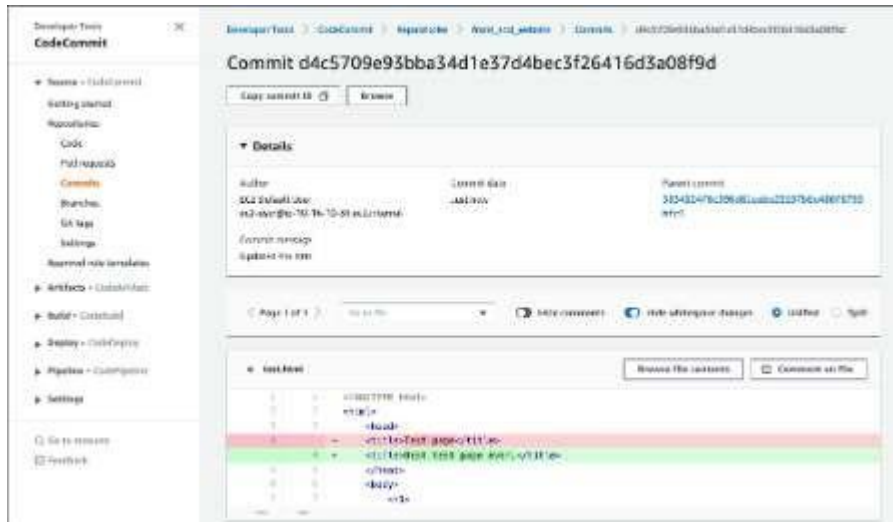
- In the **Message** text box, enter Updated the title as shown in the following image.

**Note:** Notice that one file, *test.html*, is listed under **Changes**. This is the same file that was returned by the *git status* command.

- On the **Commit** button, choose *more actions* which is located to the right.
- Choose *Commit & Push*, then choose **Yes**.
- This will push the code and trigger the code pipeline which deploys new version of the file to website.

28. Review the changes in CodeCommit.

- o Navigate to the CodeCommit console.
- o Choose the **front\_end\_website** repository link.
- o In the navigation pane, under **Repositories**, choose **Commits**.
- o Choose the link for the commit ID with the most recent commit date.
- o Go to the **test.html** section, and notice the highlighted lines, which are shown in the following image.

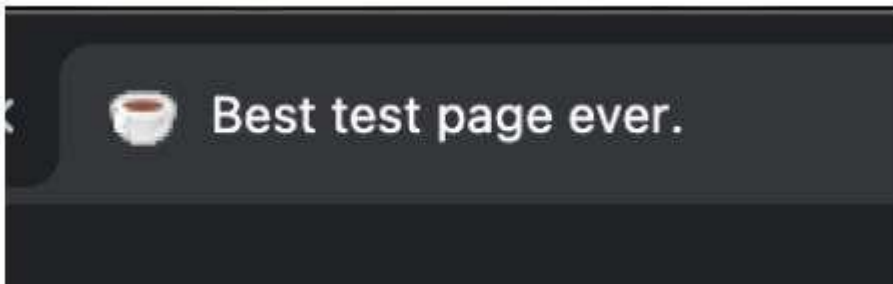


The first highlighted line is preceded by a minus sign (-) and highlighted in red. This shows the contents of the line before the change was made. The second highlighted line is preceded by a plus sign (+) and highlighted in green. This line shows the current content of the line.

Great work! You have confirmed that VS Code IDE is able to push changes to your CodeCommit repository.

29. Now, return to the tab that contains the café website and refresh the page.

Notice that the browser tab title has changed, as shown in the following image. This proves that the pipeline deployed the changes that you committed from your local repository.



**Note:** You might need to refresh the page a few times before you see the new tab title.

Now that you have learned to manage your local repository and synchronize it with CodeCommit, it's time to update the repository with the actual café website code.

## Task 6: Pushing the café website code to CodeCommit

In this task, you will first remove the test.html file. Then, you will update the local repository with the café website code. Finally, you will verify that your pipeline built the café website on S3. You will also verify that *max-age* is set to 14 to confirm that the latest changes are being applied and the caching was updated.

30. Return to the VS Code IDE tab.

31. On the explorer, Under *Environment* folder, expand the **front\_end\_website** folder and delete the **test.html** file, as shown in the following image.



32. In the VS Code IDE bash terminal, run the following commands.

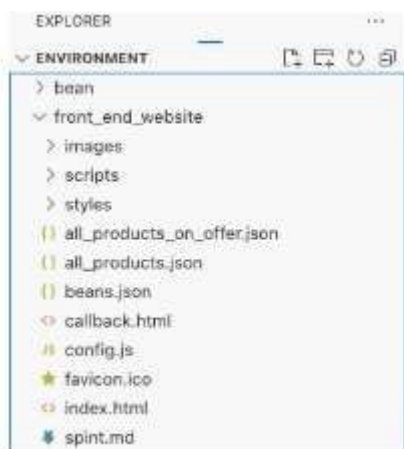
The second command will copy all of the contents under the *website* folder to the *front\_end\_website* folder.

```
cd ~/environment
cp -r ./resources/website/* front_end_website
```

33. To delete the *website* folder, so that there is one source of truth, run the following command.

```
rm -r ./resources/website
```

Now, your local repository has the folder structure shown in the following image:



34. Commit your changes.

- o Choose the *Branch* icon (now it is showing number of new files to be pushed).
- o Enter the following commit message: Providing the website
- o On the **Commit** button, choose *more actions* which is located to the right.
- o Choose *Commit & Push*, then choose **Yes**.
- o To the right of **front\_end\_website**, choose the options icon and choose **Commit**.

- o The **Source control Graph** in the lower pane shows history of changes done to the repository.

35. Return to the browser tab that shows the *test.html* page.

36. Update the URL by removing **test.html** from the address, and then submit the updated URL.

The updated address is similar to the following example: <https://aaabbb111222.cloudfront.net>

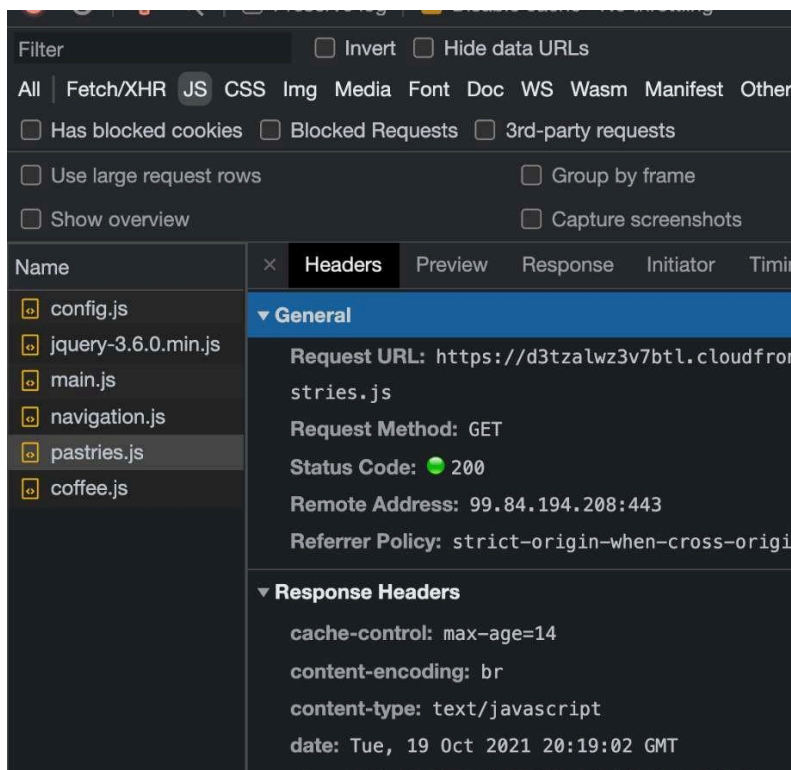
The browser now displays the café website. Well done!

37. Verify that your pipeline applied the cache-control setting, which was configured in Task 3.

- o Remain on the café website page, and open your browser's developer tools.

**Note:** To open the developer tools, open your browser's context menu (right-click) and choose **Inspect**.

- o Choose the **Network** tab, and then refresh the webpage.
- o Choose **pastries.js**.
- o Choose the **Headers** tab, and locate the **Response Headers** section, as shown in the following image.



Notice that the **cache-control** value is set to *max-age=14*, which indicates that pipeline updated the cache settings. This means that the website is being built from the most recent repository update.

**Note:** If **cache-control** is set to *max-age=0*, the pipeline might still be applying the update to the S3 bucket. Wait a few seconds, refresh the page, and choose **pastries.js** again.

Congratulations! You have moved the codebase to a secure and scalable managed service. You have also ensured that the website will stay up to date with the latest improvements.

## Submitting your work

38. 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** checkbox. This action will ensure that the lab instructions remain visible after you choose **Submit**.

39. 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.

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

## Lab complete

Congratulations! You have completed the lab.

41. 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.*

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