**Aim** :- Create Lambda functions using the AWS SDK for Python.

**Lab overview and objectives**

In this lab, you will use the AWS SDK for Python (boto3) to create AWS Lambda functions. Calls to the REST API that you created in the earlier Amazon API Gateway lab will initiate the functions. One of the Lambda functions will perform either an Amazon DynamoDB database table scan or an index scan. Another Lambda function will return a standard acknowledgment message that you will enhance later in a lab where implement Amazon Cognito.

After completing this lab, you should be able to:

* Create a Lambda function that queries a DynamoDB database table.
* Grant sufficient permissions to a Lambda function so that it can read data from DynamoDB.
* Configure REST API methods to invoke Lambda functions using Amazon API Gateway.

**AWS service restrictions**

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

**Scenario**

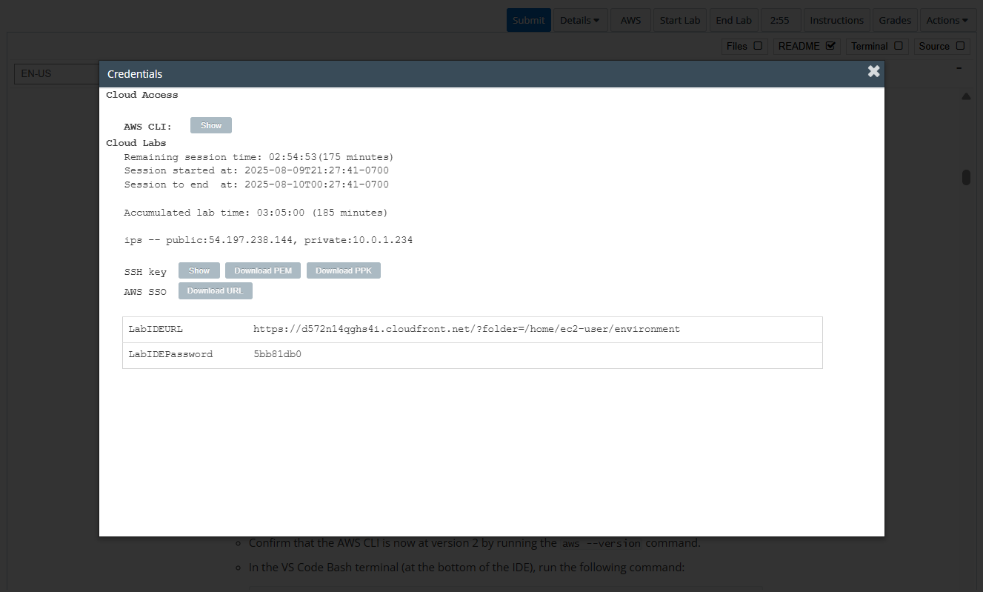
The café is eager to launch a dynamic version of their website so that the website can access data stored in a database. Sofía has been making steady progress toward this goal.

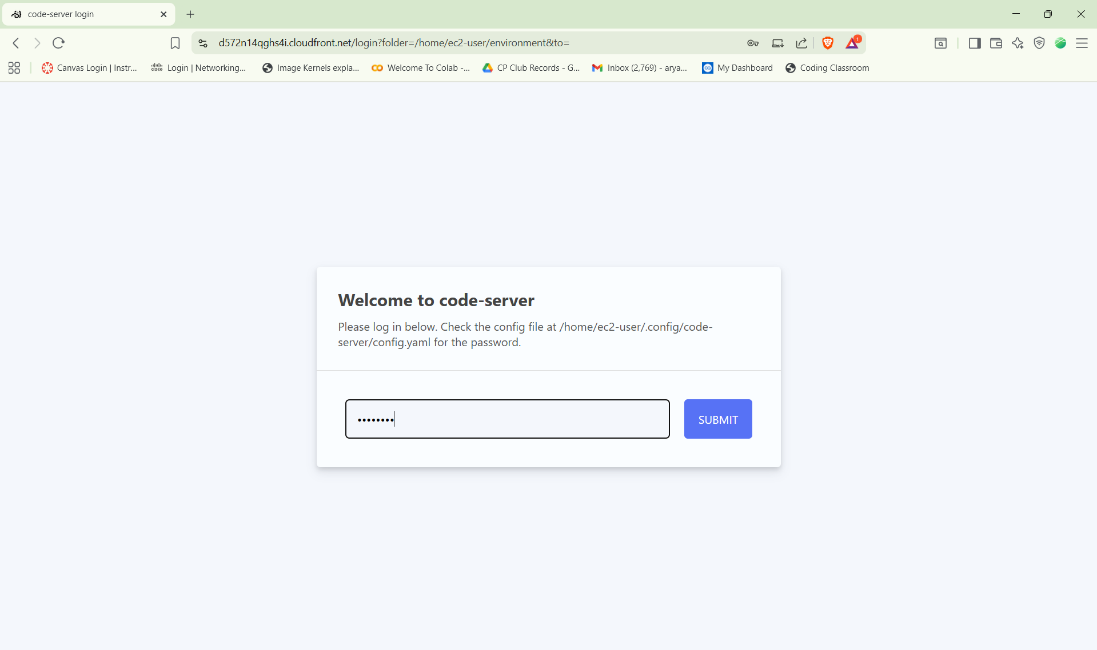
In a previous lab, you played the role of Sofía and created a DynamoDB database. The database table contains café menu details, and an index holds menu items that are flagged as specials. Then, in another lab, you created an API to add the ability for the website to receive mock data through REST API calls.

In this lab, you will again play the role of Sofía. You will replace the mock endpoints with functional endpoints so that the web application can connect to the database. You will use Lambda to bridge the connection between the GET APIs and the data stored in DynamoDB. Finally, for the POST API call, Lambda will return an updated acknowledgment message.

**Task 1: Preparing the lab**

Connect to the VS Code IDE.

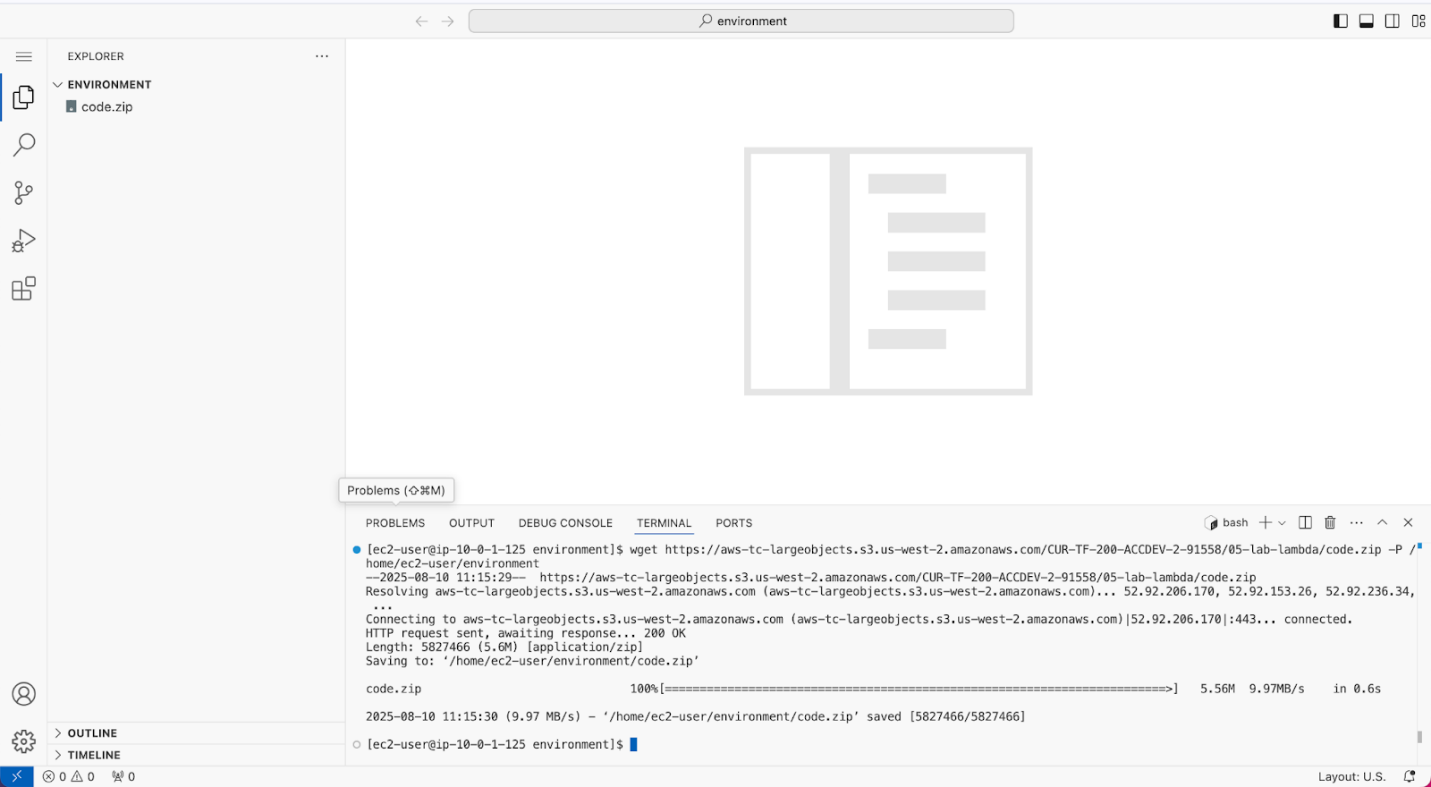
1. At the top of these instructions, choose Details followed by **AWS: Show**
2. Copy values from the table **similar** to the following and paste it into an editor of your choice for use later.
   1. **LabIDEURL**
   2. **LabIDEPassword**
3. In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
4. 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.

****

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

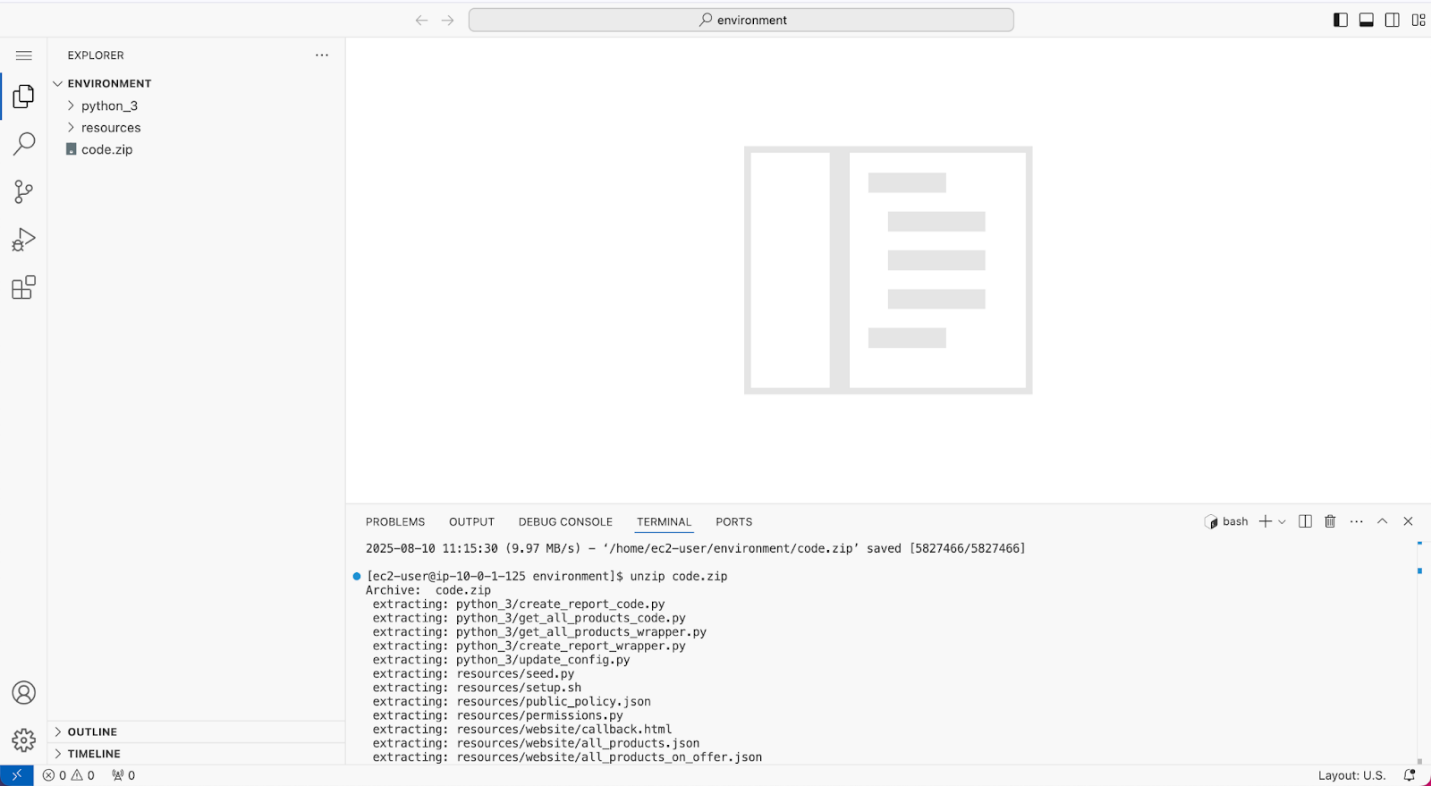
* In the VS Code bash terminal (located at the bottom of the IDE), run the following commands:

**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**



1. You should see that the **code.zip** file was downloaded to the VS Code IDE and is now in the left navigation pane.

* Extract the file by running the following command:

unzip code.zip

1. Run a script that upgrades the version of the AWS CLI installed on the VS Code IDE.

* To set permissions on the script and then run it, run the following commands in the Bash terminal:

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

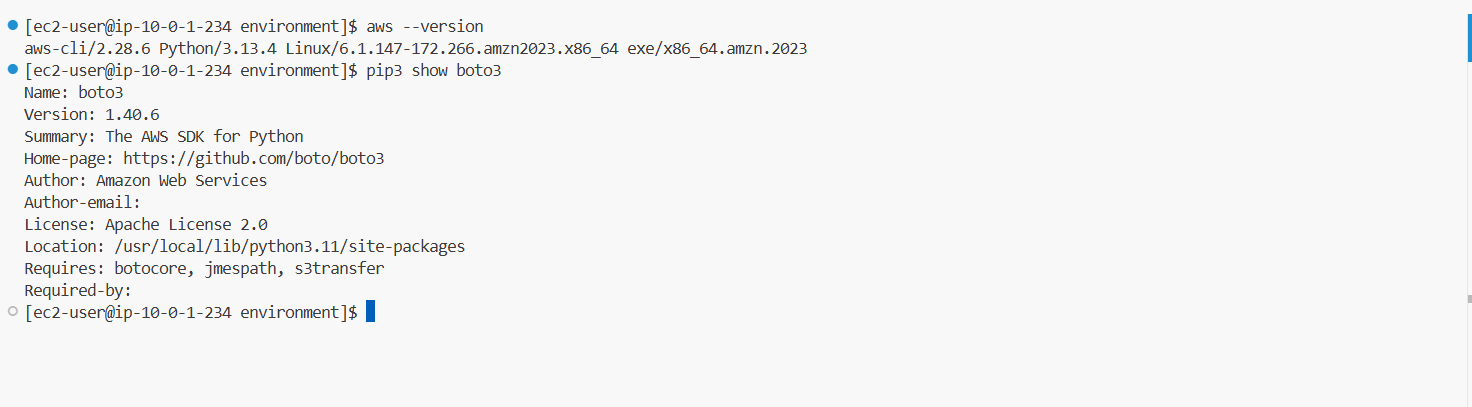
The script will prompt you for the **IP address** by which your computer is known to the internet.

Use [www.whatismyip.com](http://www.whatismyip.com/) to discover this address and then paste the IPv4 address into the command prompt and finish running the script.



1. Verify the AWS CLI version and also verify that the SDK for Python is installed.

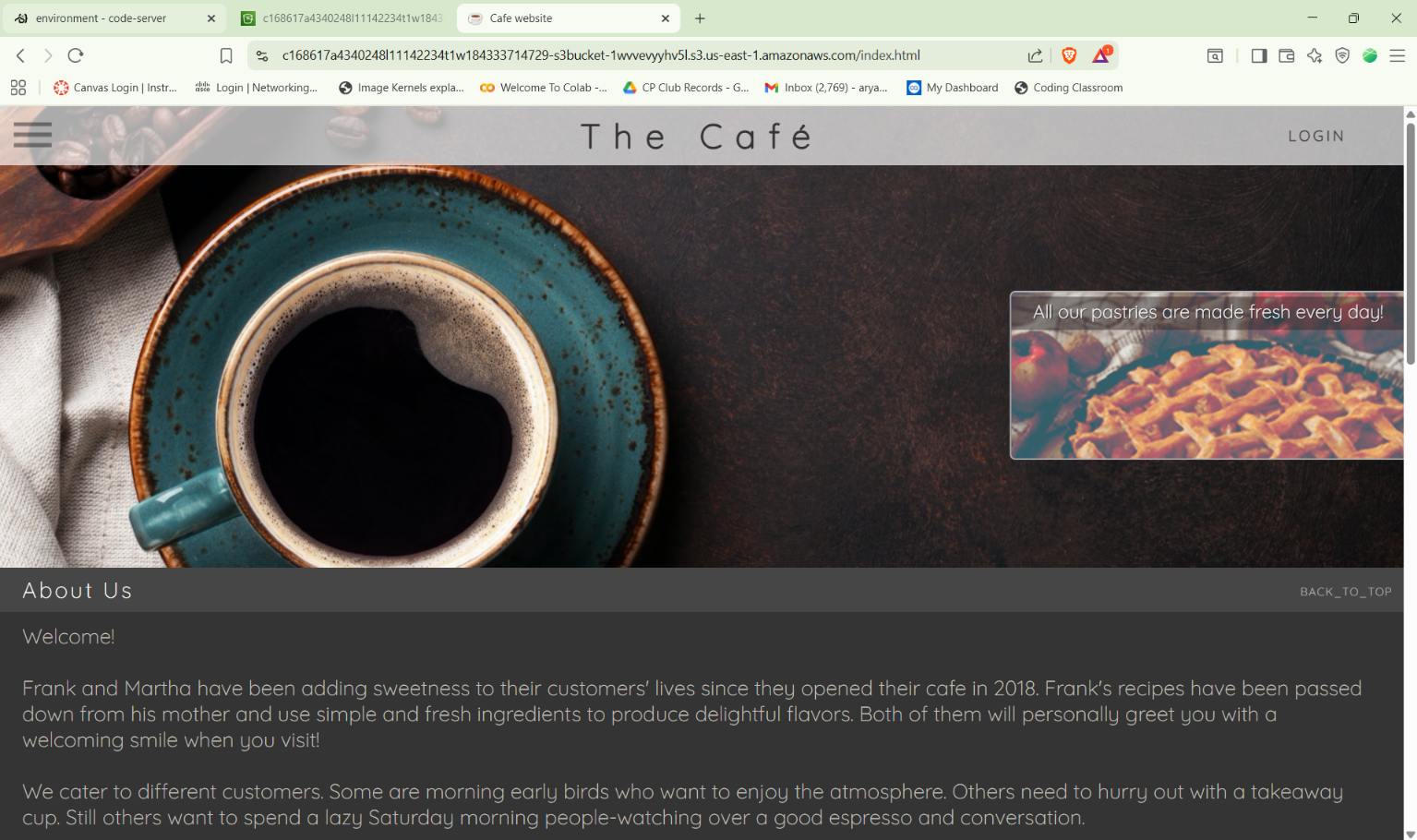
* Confirm that the AWS CLI is now at version 2 by running the **aws --version** command.
* In the VS Code Bash terminal (at the bottom of the IDE), run the following command:

**pip3 show boto3**

1. Verify that the cafe website can be loaded in a browser tab.
   * Load the website in a browser tab.
     + In a browser tab, open the Amazon S3 console.
     + Choose your bucket name, and then choose **Objects**.

If the files that the script just uploaded do not display, choose the refresh icon to view them.

* + - Choose the **index.html** file.
    - Copy the **Object URL**. It will be in the following format. https://<bucket-name>.s3.amazon.com/index.html
    - Verify that the website displays by pasting the full URL into your browser.



1. Take a moment to see what resources the script created.
   * Confirm that an S3 bucket is hosting the café website files:
     + On the AWS Management Console, in the search bar at the top, search for and select s3 to open the Amazon S3 console.
     + From the **General purpose buckets**, 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.

* + Confirm that DynamoDB has the menu data stored in a table:
    - On the AWS Management Console, in the search bar at the top, search for and select DynamoDB to open the Amazon DynamoDB console.
    - Browse 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.
  + Confirm that the ProductsApi REST API was defined in API Gateway:
    - On the AWS Management Console, in the search bar at the top, search for and select API Gateway to open the Amazon API Gateway console.
    - 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.

1. Copy the invoke URL for the API to your clipboard.
   * 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.

* + Copy the **Invoke URL** value that displays at the top of the page.

You will use this value in the next step.

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

* In the VS Code IDE browser tab, open resources/website/**config.js**.
* 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.
* 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**

**};**

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

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

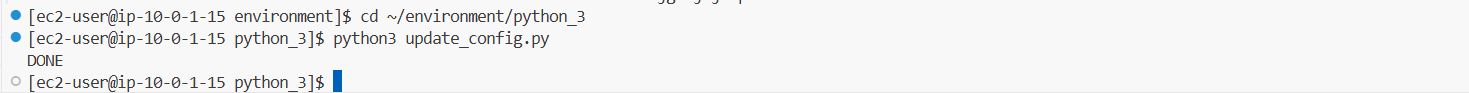
* Open python\_3/**update\_config.py** in the text editor.
* Replace the <FMI\_1> placeholder with the name of your S3 bucket.
* **Tip**: Find the bucket name in the S3 console, or run the following command:

**aws s3 ls**

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

**cd ~/environment/python\_3**

**python3 update\_config.py**

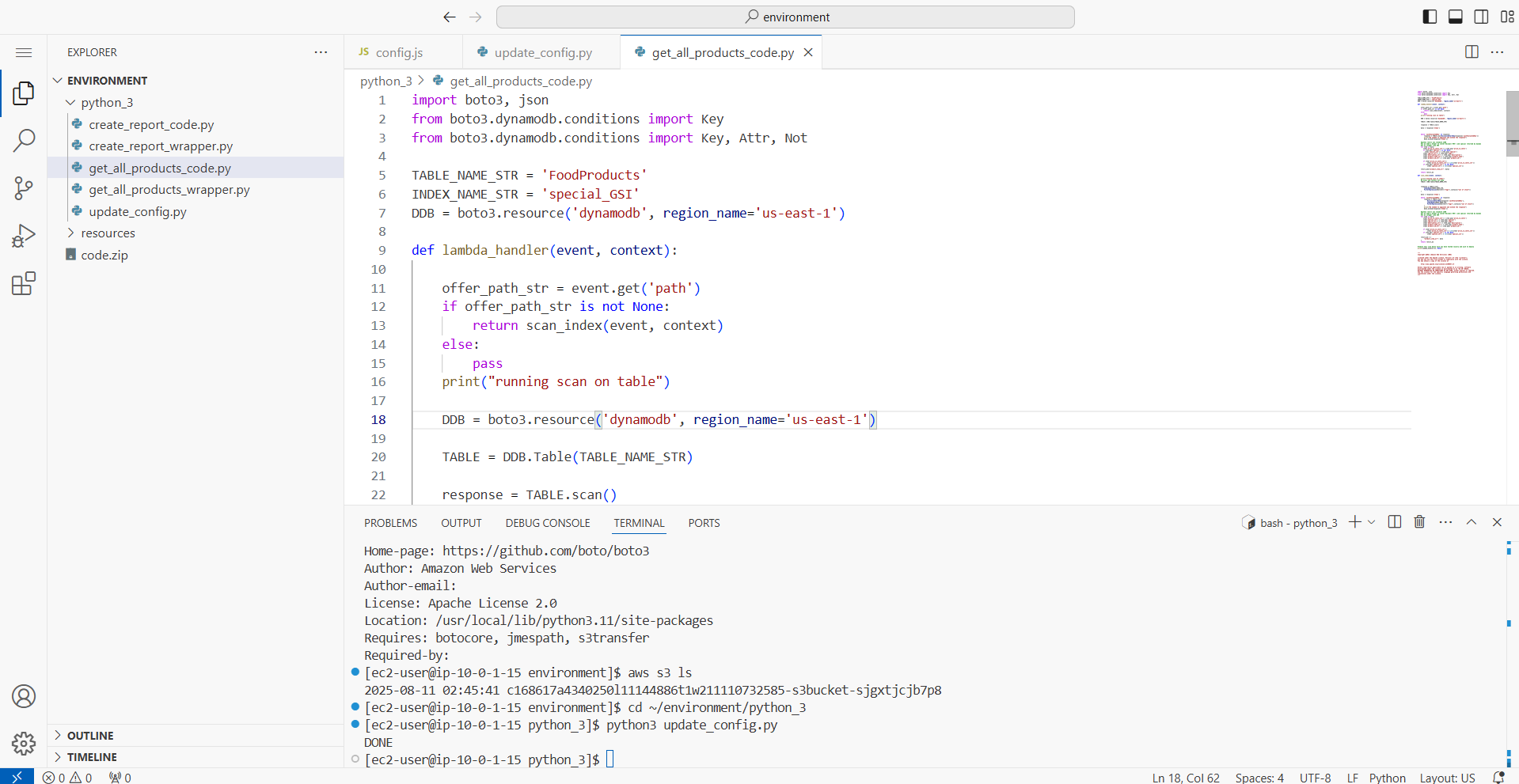


**Task 2: Creating a Lambda function to retrieve data from DynamoDB**

1. Observe and edit the Python code that you will use in the Lambda function.
   * In the VS Code IDE file browser, browse to and open python\_3/**get\_all\_products\_code.py**.
   * 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.

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



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

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

cd ~/environment/python\_3

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

**python3 get\_all\_products\_code.py**

****

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

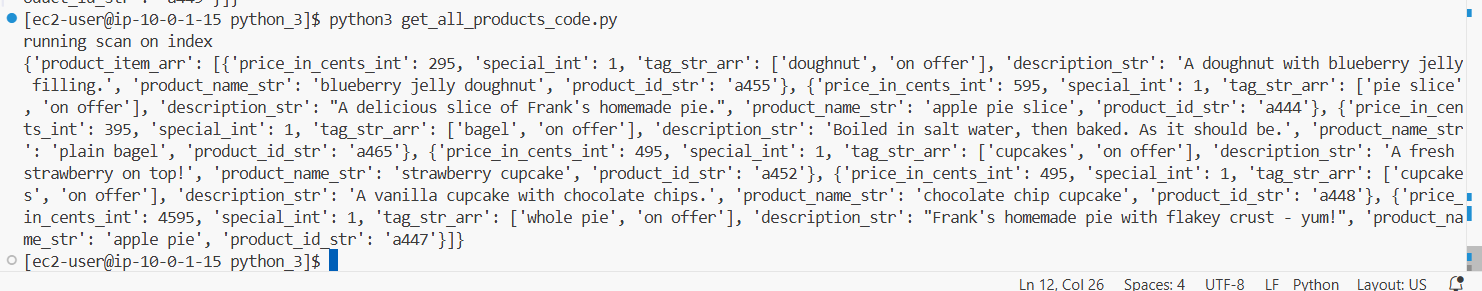
* In the **get\_all\_products\_code.py** file, line 12 has the following code:

**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.
* 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:**

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

**python3 get\_all\_products\_code.py**

1. Package the code and store it in an S3 bucket.
   * A bucket with -s3bucket- in the name was created for you when you started the lab.
   * Verify that your VS Code IDE terminal is in the **python\_3** directory.

**cd ~/environment/python\_3**

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

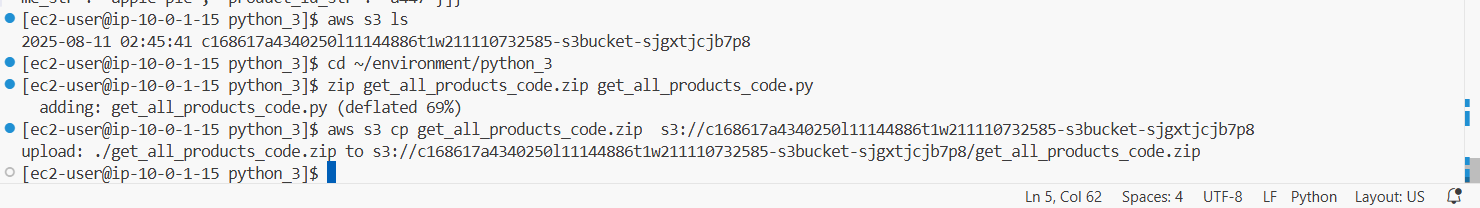
**zip get\_all\_products\_code.zip get\_all\_products\_code.py**

* + Next, to retrieve the name of your S3 bucket, run the following command:

**aws s3 ls**

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

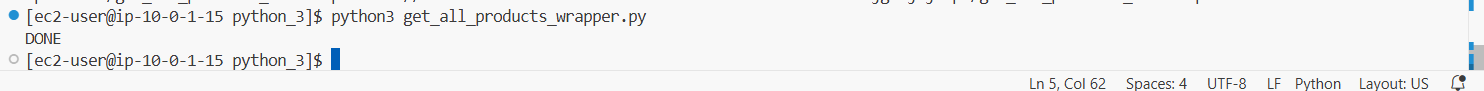
**aws s3 cp get\_all\_products\_code.zip s3://<bucket-name>**



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

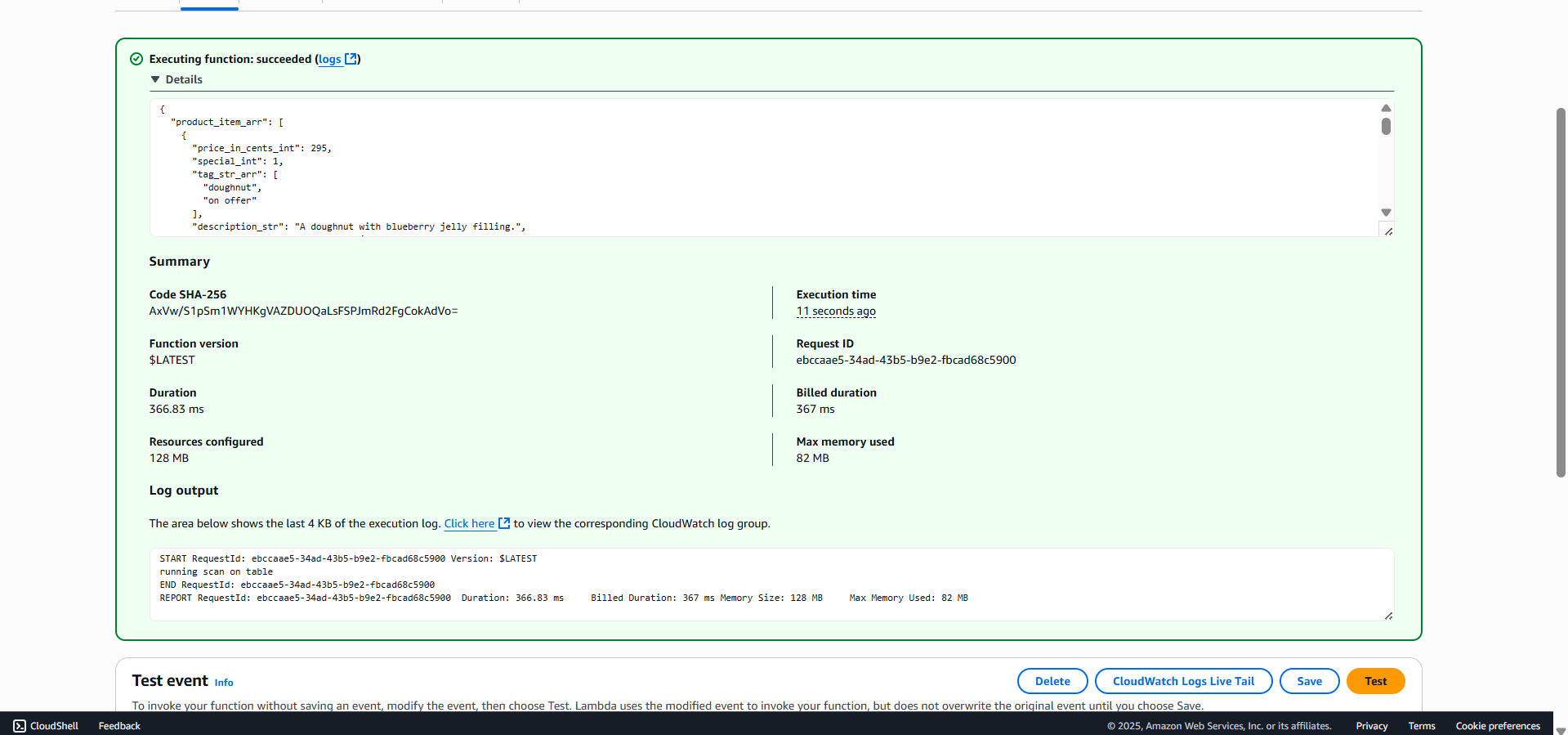
**python3 get\_all\_products\_wrapper.py**

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

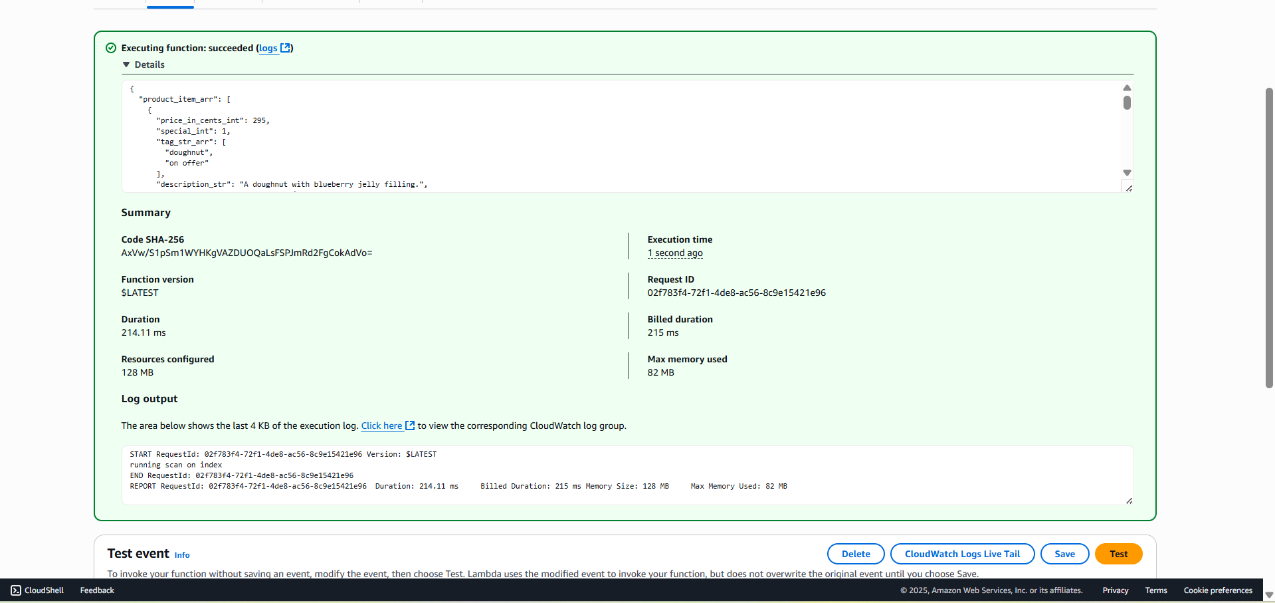


1. Observe the function that you created and test it.

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



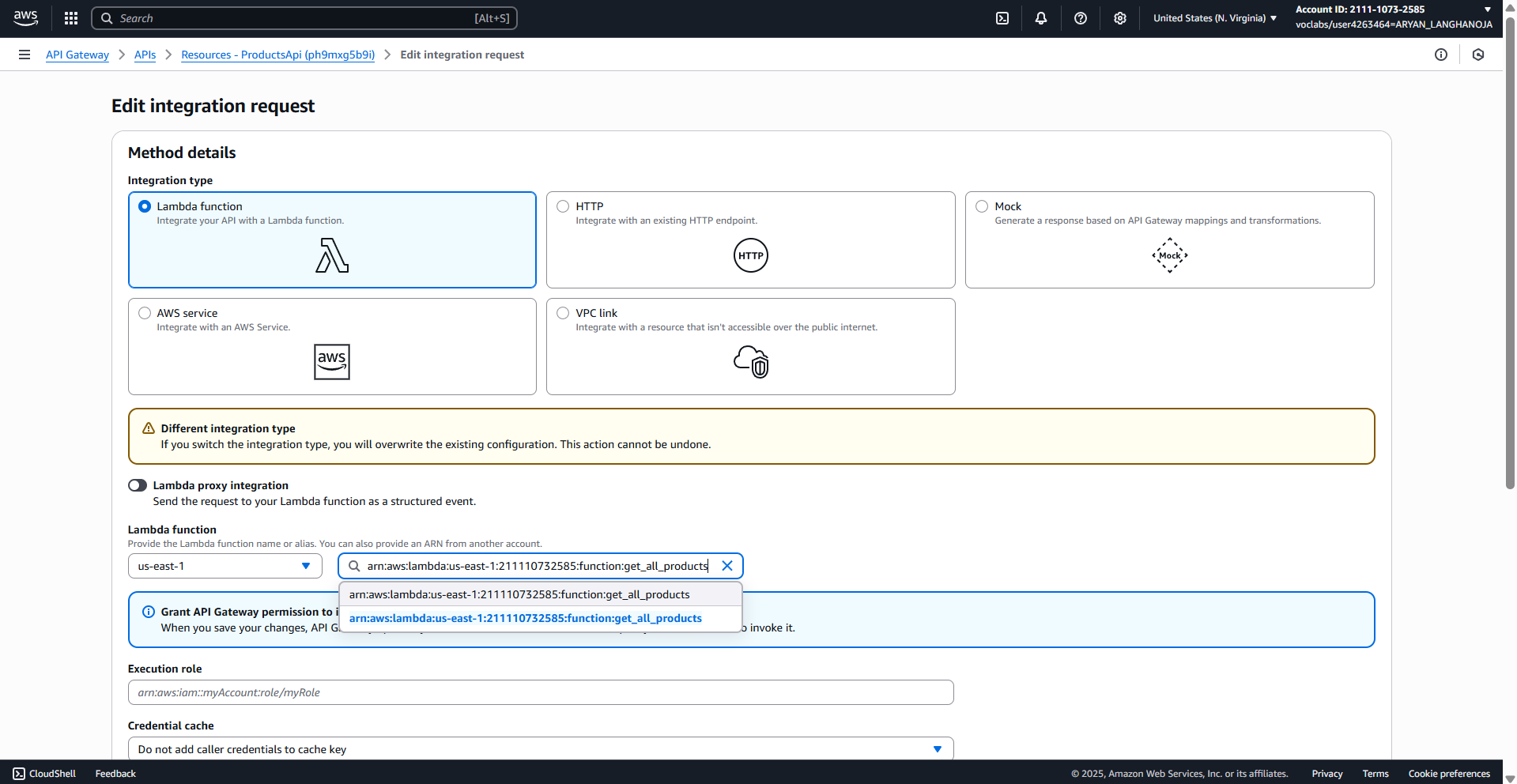
1. Create a new test event that is called **onOffer**.

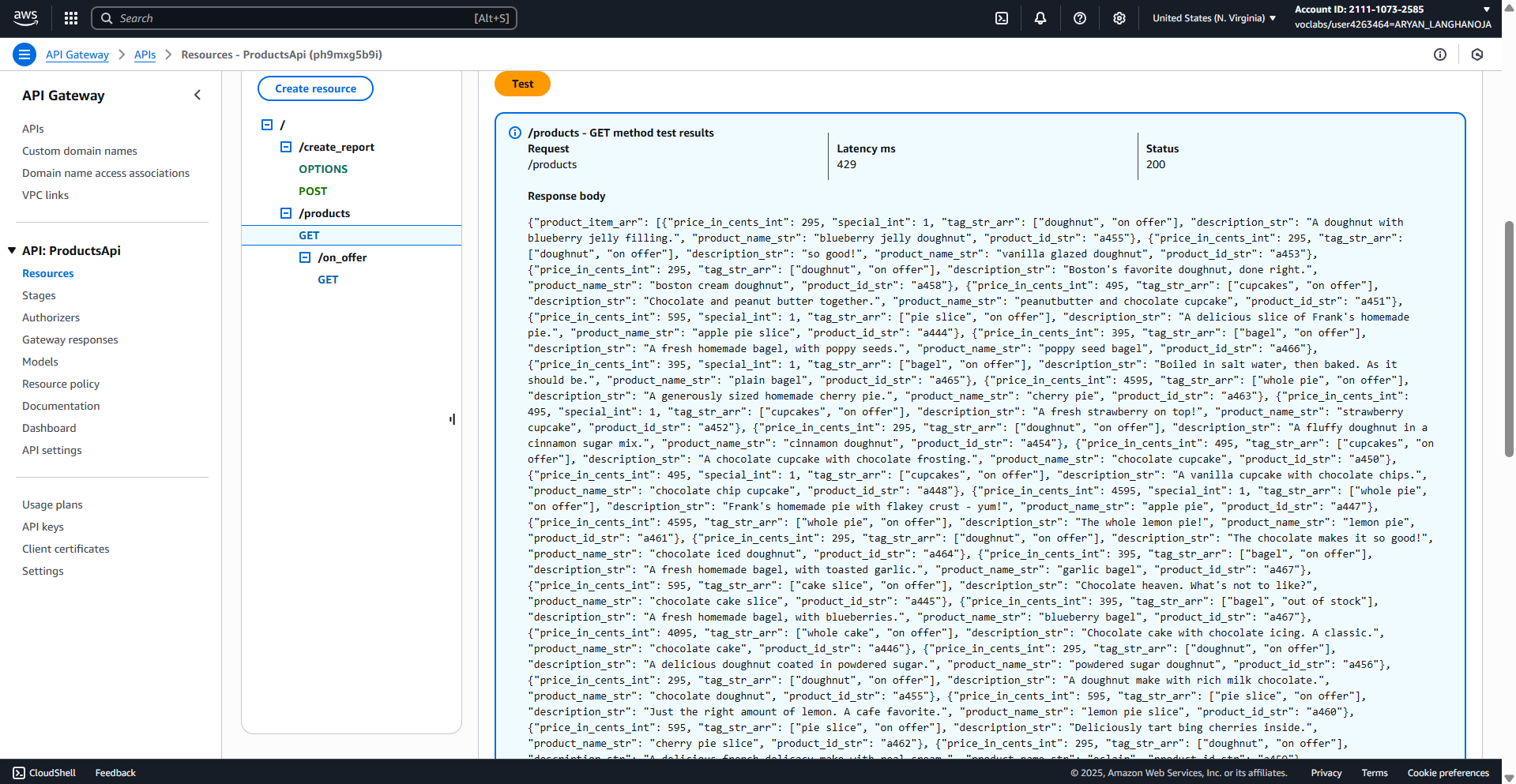
* In the **Code source** panel, open the **Test** menu (choose the arrow icon), and choose **Configure test event**.
* Choose **Create new event**.
  + For **Event name**, enter onOffer
  + In the code editor panel, replace the existing code with the following:

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

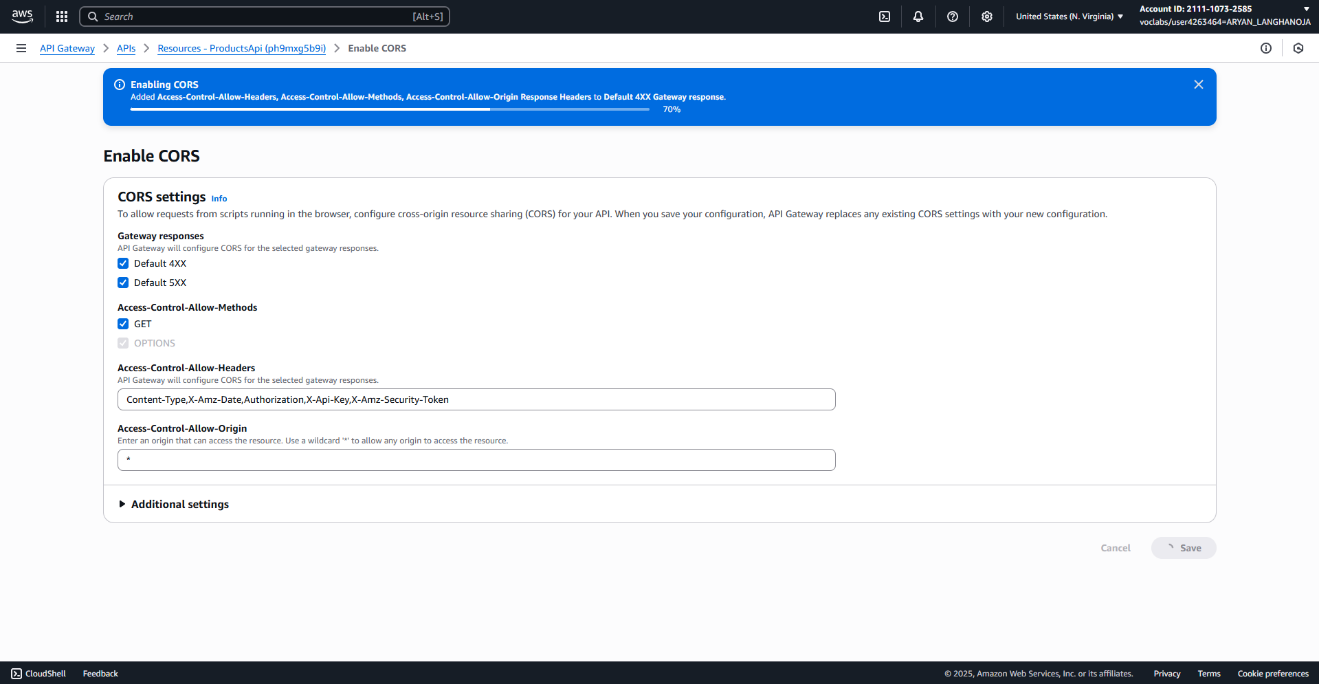
1. Replace the mock endpoint with the Lambda function.

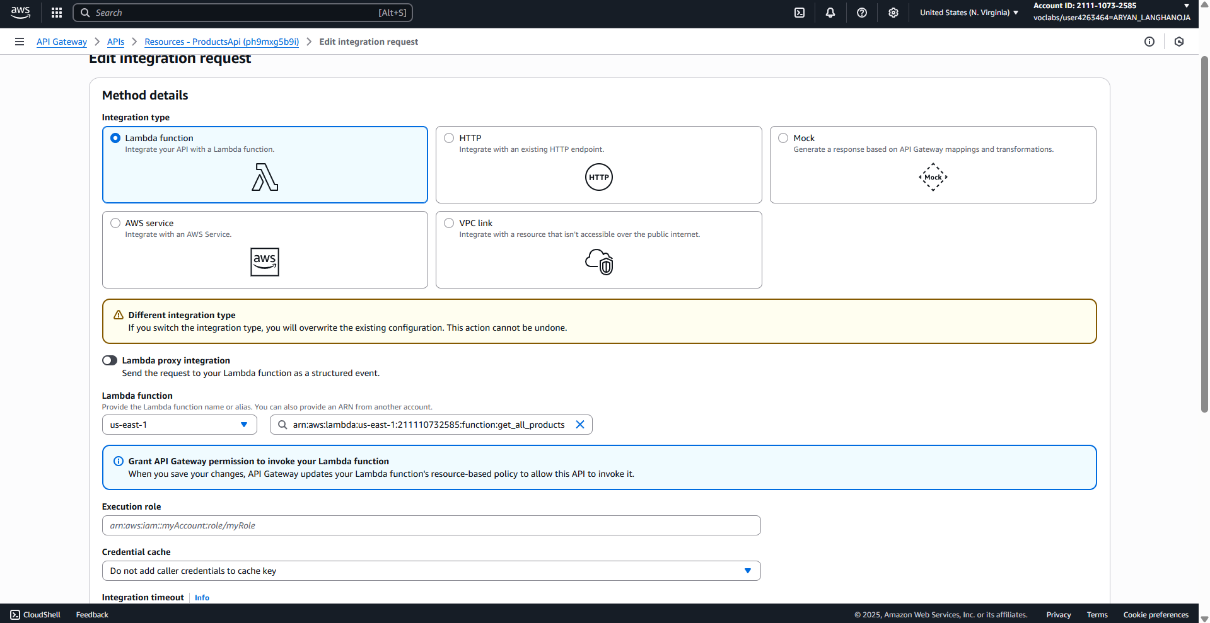
* At the top of the page. Ensure that the **GET** method is still selected under **/products**.
* Choose **Integration Request** and **Edit**:
* Integration type: **Lambda Function**
* Lambda Region: **us-east-1**
* Lambda Function: get\_all\_products
* Choose **Save**
* Choose **Save**.
* Notice on the right side of the page that the method is no longer calling a "Mock Endpoint". Instead, it is calling your Lambda function.

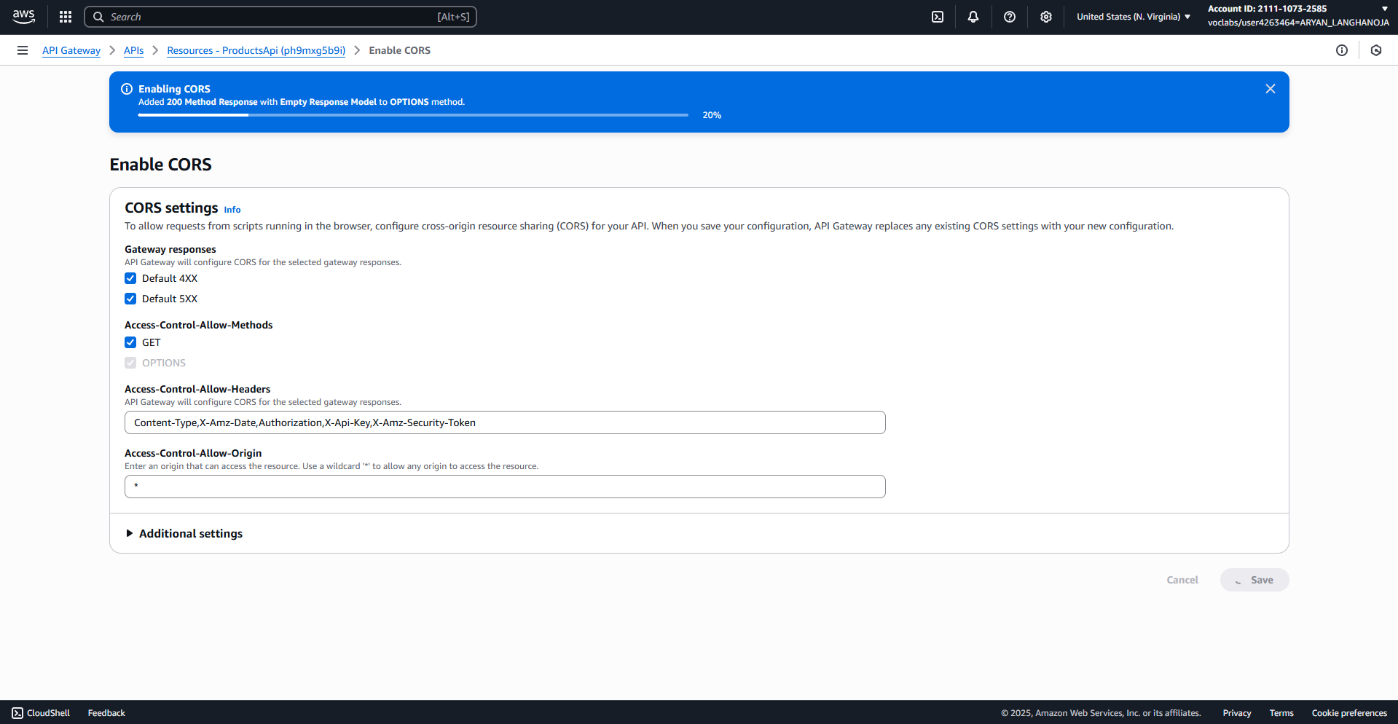




1. Re-enable CORS on the /products API resource.
   * Choose **/products** so that it is highlighted.
   * Select the **GET** method.
   * Choose button **Enable CORS** from the top.
   * Select **Default 4XX** and **Default 5XX** under **Gateway responses**.
   * Select **GET** under **Access-Control-Allow-Methods**.
   * Choose **Save**.



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



1. Configure the /on\_offer integration request details.
   * Choose the **GET** method for **/products/on\_offer**.
   * Choose **Integration Request**.
   * Choose **Edit**. Expand **Mapping Templates**.
   * Choose **Add mapping template**.
   * In the Content-Type box, enter the following text:

application/json

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

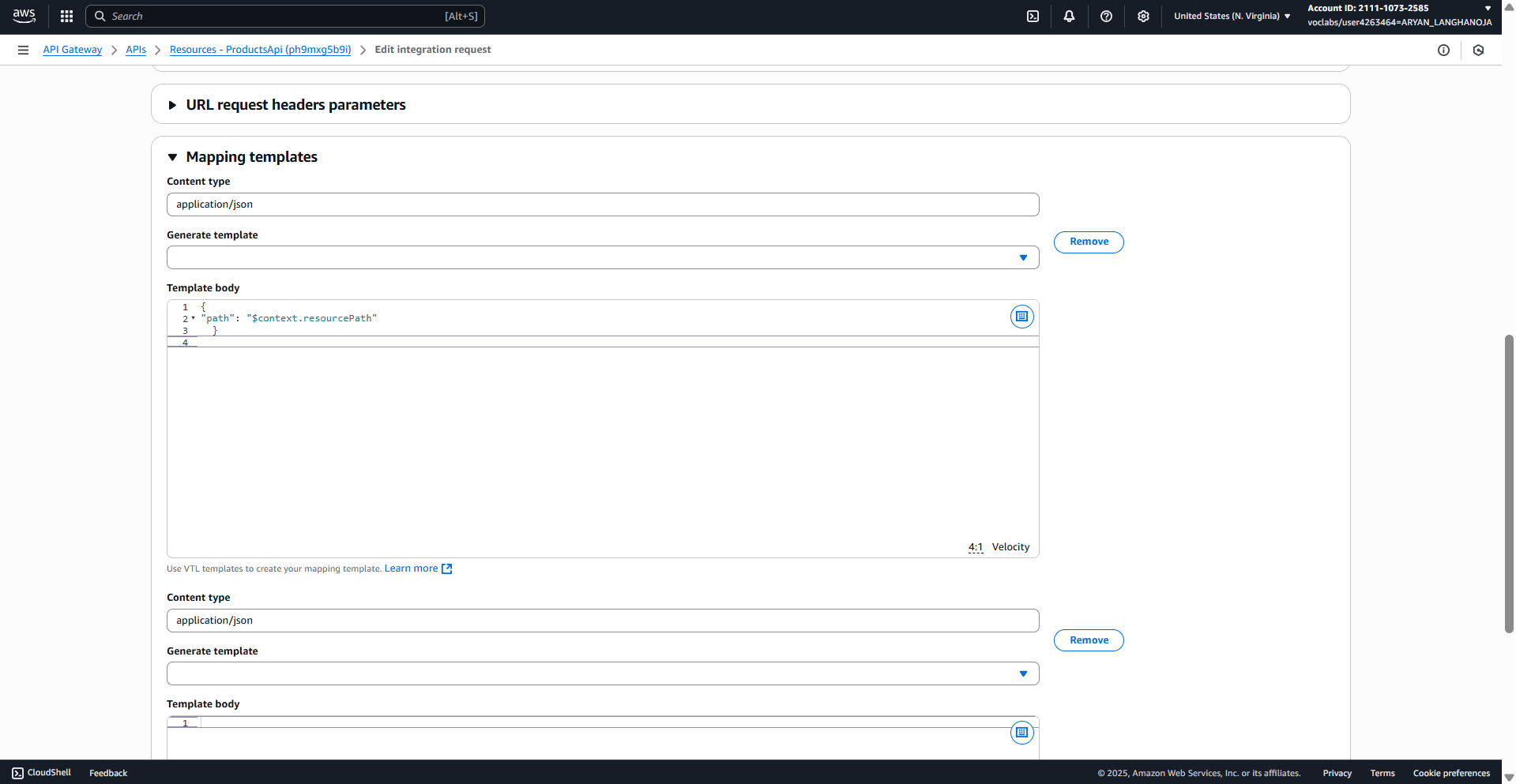
{

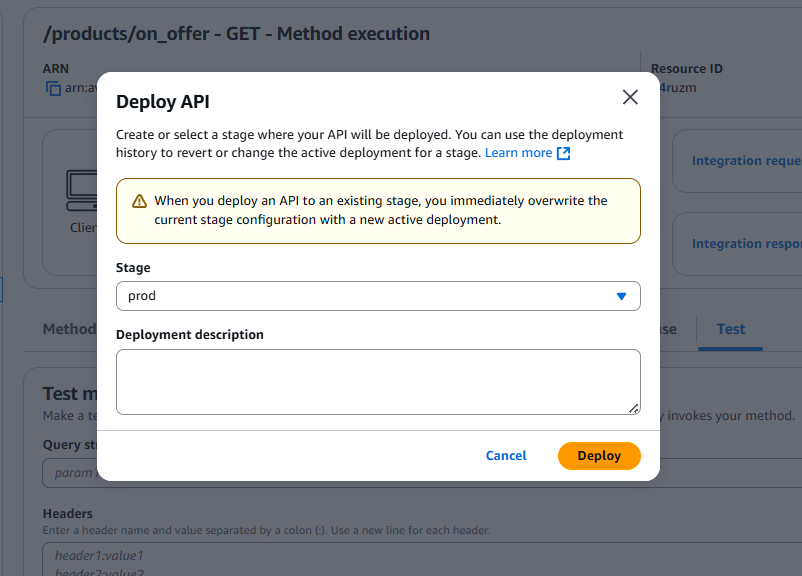
"path": "$context.resourcePath"

}

This will evaluate to /products/on\_offer. For now, the code simply checks for the existence of the variable.

* + Choose **Save** at the bottom of the page.



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

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

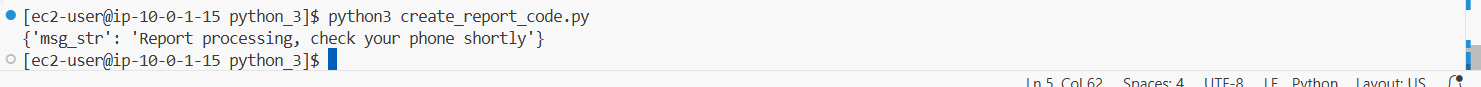
1. Observe and test the Python code that you will use in the Lambda function.

* 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.
* Run the following command in the terminal:

**python3 create\_report\_code.py**

* 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.

1. Edit the wrapper code that you will use to create the Lambda function.
   * Browse to and open python\_3/**create\_report\_wrapper.py**.
   * 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.

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

1. Package the code and store it in the S3 bucket.
   * To place a copy of your code in a .zip file, run the following command:

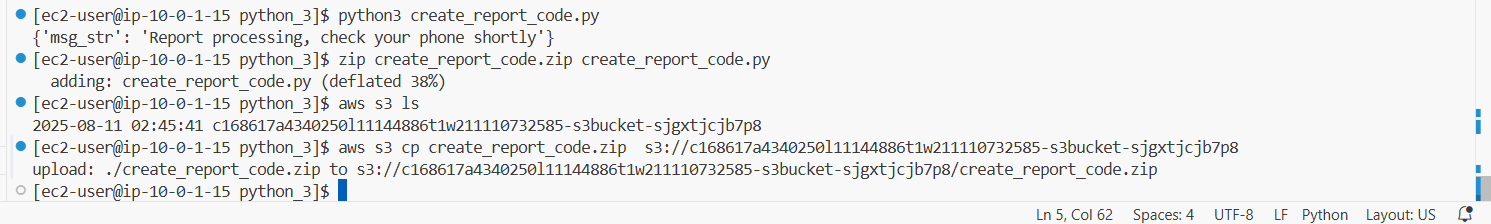
**zip create\_report\_code.zip create\_report\_code.py**

* + 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>**

* + Verify that the command succeeded.

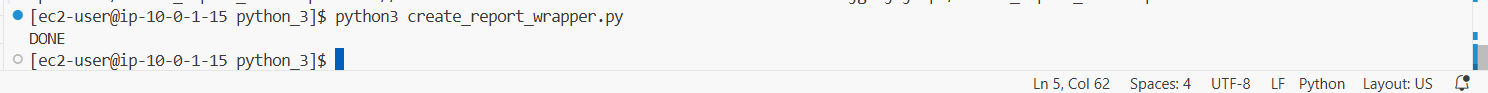
The following is the output: **upload: ./create\_report\_code.zip to s3://<bucket-name>/create\_report\_code.zip**



1. 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.



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

1. Test the existing POST method for /create\_report.
   * Browse to the API Gateway console.
   * 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".

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

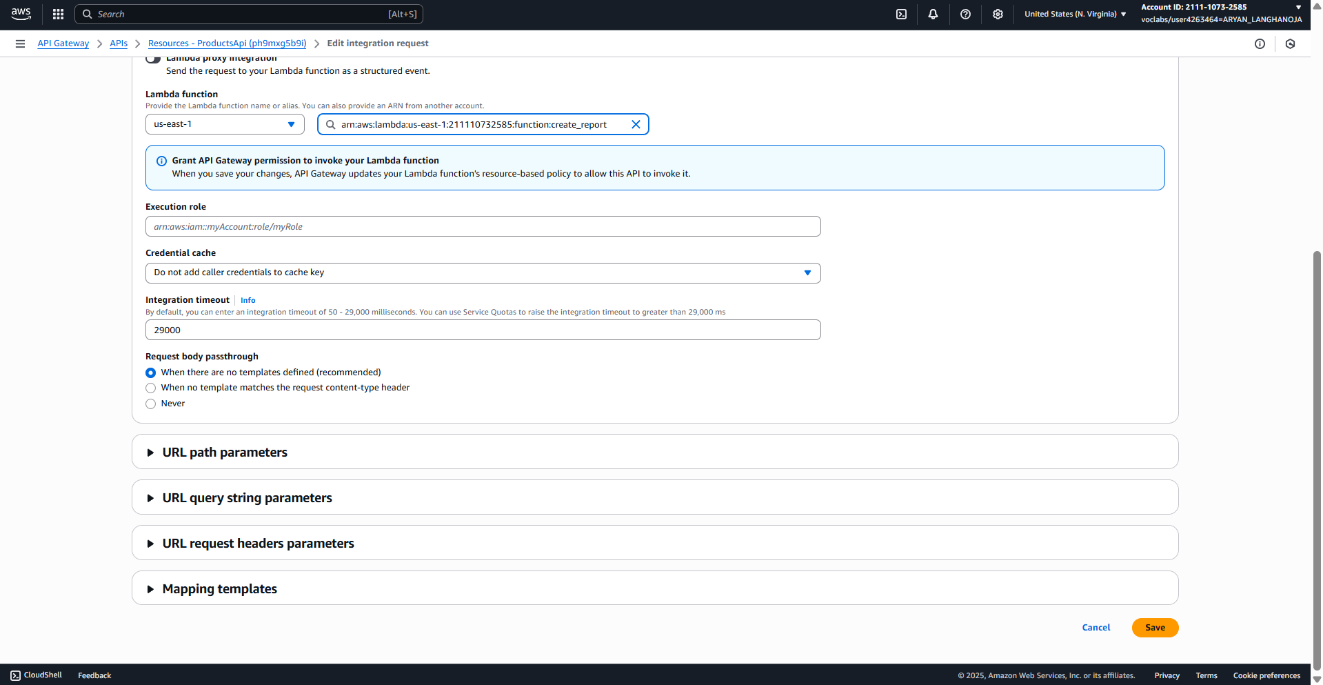
Verify that the Response Body correctly returns the mock data (note the lowercase "r" in the results), as in the following example:

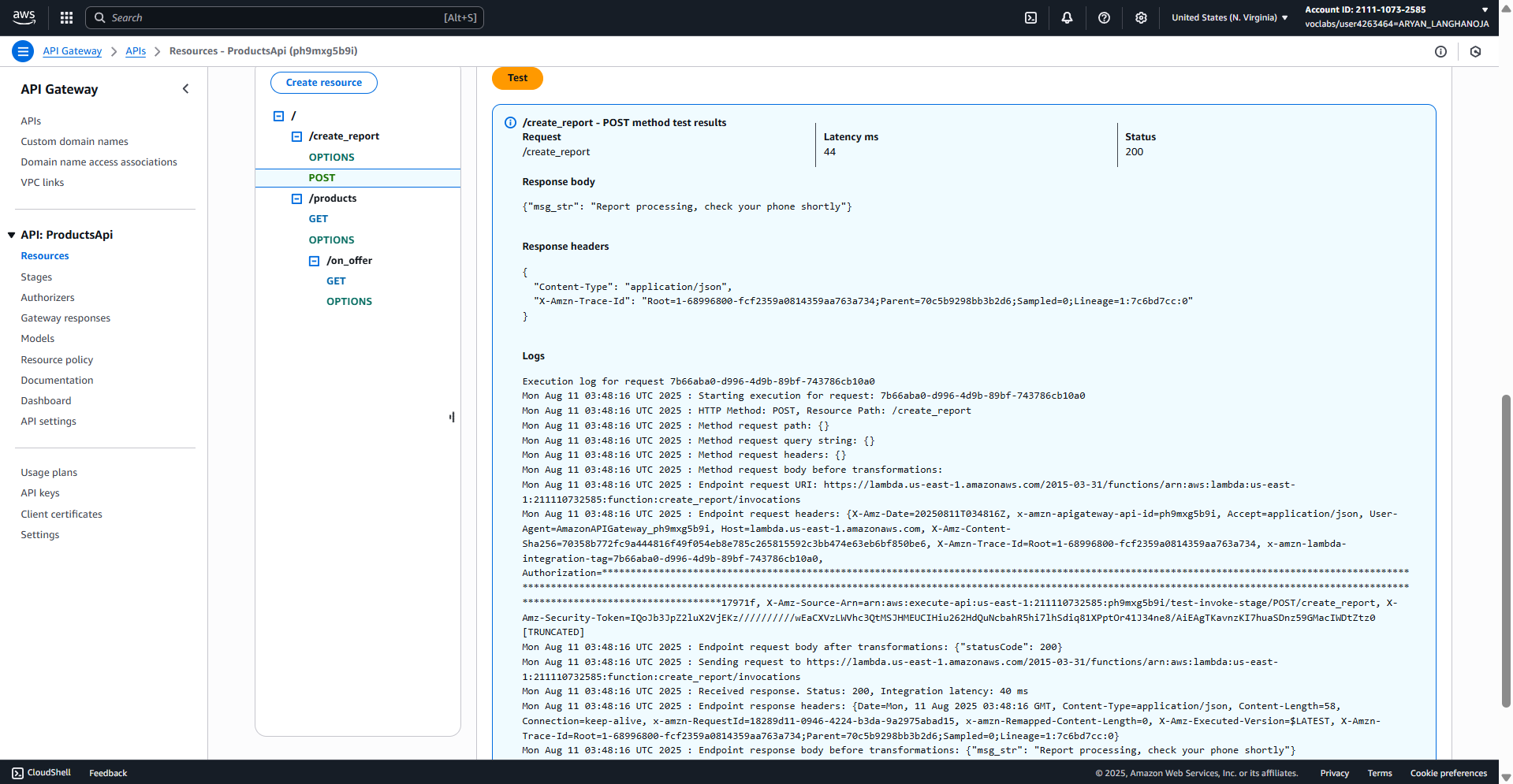
{

 "message\_str": "report requested, check your phone shortly."

}

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

****

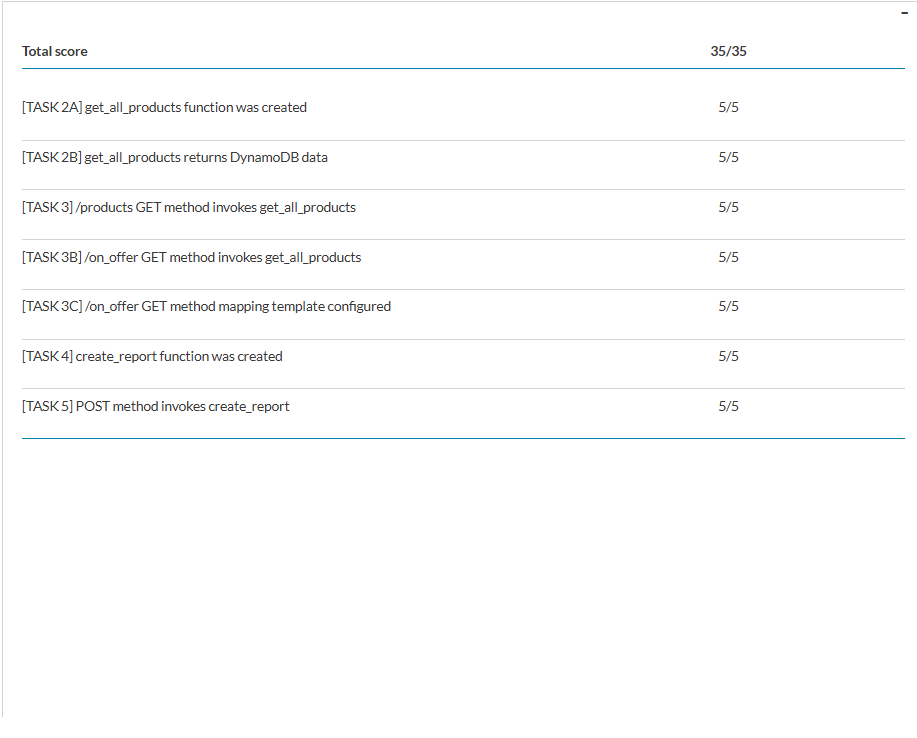


1. Deploy the API.
   * In the Resources panel, choose the API root /.
   * Choose Deploy API.
   * For Deployment stage, choose prod, and then choose Deploy.

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

**Conclusion:-**

* In this Lab I Learned the AWS Lambda Function Service.
* I had integrated the REST API of GET Method /product with lambda function using python.
* I had intgrrated the REST API of GET Method /product/on\_offer with lambda function using python.
* I had integrated the REST API of POST Method /create\_report with lambda functon using python.
* I had re-deployed that API

**Result :-**