

- In the upper left, choose **File > Save** to save your changes.

63. Review the code.

- On line 18, including the **IndexName** option lets the **scan** operator know that it will be going to the index and not the main table to read the data.
- On line 19, the filter expression processes the records that have been read and shows only records that meet the comparison criteria. In this case, it shows records only if they don't have **out of stock** in the **tags** attribute. This ensures that only items that are available or "on offer" are shown to customers.

```
xxxxxxxxxx

response = table.scan(

    IndexName='special_GSI',

    FilterExpression=Not(Attr('tags').contains('out of stock')))
```

Note: You may be familiar with the older DynamoDB parameter, ScanFilter. This is a **legacy** parameter. FilterExpression should be used instead. FilterExpression does not include all of the operators that were provided with ScanFilter, for example NOT_CONTAINS. This is why you wrapped the comparison inside the **Not()** function.

64. Save the file and run it.

```
xxxxxxxxxx

python3 scan_with_filter.py
```

The output should be similar to the following:

```
xxxxxxxxxx

[{'price_in_cents': Decimal('295'), 'special': Decimal('1'), 'description': 'so good!', 'product_name': 'blueberry jelly doughnut', 'product_id_str':
```

Again, the response is not in the format the website requires. However, when you use this code in a Lambda function in a later lab, you will adjust then.

Update from the café

Sofia is happy with the progress that she has made. The database table is loaded with data, she addressed the café's database backend requirements, and she will soon wire this into the website.

Sofia's next task is to create an API that the website can use.

Sofia decides to stop for the day, but she's looking forward to her next task. She plans to start working on creating a mock API that she can use for testing.

Submitting your work

65. At the top of these instructions, choose **Submit** to record your progress and when prompted, choose **Yes**.
66. 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 is recorded for this lab.

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

Lab complete

Congratulations! You have completed the lab.

68. 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*.

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

© 2021 Amazon Web Services, Inc. and its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.

LAB 6.1

[Version 1.0.5]

Lab 6.1: Developing REST APIs with Amazon API Gateway

Lab overview and objectives

In this lab, you will create a REST application programming interface (API) by using Amazon API Gateway.

After completing this lab, you should be able to:

- Create simple mock endpoints for REST APIs and use them in your website.
- Enable Cross-Origin Resource Sharing (CORS)

Duration

This lab will require approximately **60 minutes** to complete.

AWS service restrictions

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

Scenario

In the *previous lab*, you took on the role of Sofia to build a web application for the café. As part of this process, you created an *Amazon DynamoDB table* that was named *FoodProducts*, where you stored information about café menu items.

You then loaded data that was formatted in JavaScript Object Notation (JSON) into the database table. The table structure looked similar to the following table (one line item of table data is shown as an example):

product_name	description	price_in_cents	product_id	tags	special
apple pie slice	A delicious slice of Frank's homemade pie.	595	a444	[{ "S" : "pie slice" }, { "S" : "on offer" }]	1

In the *previous lab* you also configured code that used the AWS SDK for Python (Boto3) to:

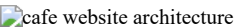
- Scan a DynamoDB table to retrieve product details.
- Return a single item by product name using get-item as a proof of concept
- Create a Global Secondary Index (GSI) called **special_GSI** that you could use to filter out menu items that are on offer and not out of stock.

In *this lab*, you will continue to play the role of Sofia. You will use Amazon API Gateway to configure **mock data endpoints**. There are three that you will create:

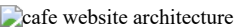
- **[GET]** /products (which will *eventually* invoke a DynamoDB table scan)
- **[GET]** /products/on_offer (which will *eventually* invoke a DynamoDB index scan and filter)
- **[POST]** /create_report (which will *eventually* trigger a batch process that will send out a report)

Then in the lab that follows this one, you will replace the mock endpoints with real endpoints, so that the web application can connect to the DynamoDB backend.

When you *start* the lab, the following resources are pre-created for you in the account.



However, by the *end* of this lab, you will have created the following architecture:



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 AWS Cloud9 environment so that you can create the REST API.

5. From the AWS Management Console, connect to the AWS Cloud9 IDE named **Cloud9 Instance**.

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

- In the same terminal, run the following command:

```
xxxxxxxxxx
```

```
wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/04-lab-api/code.zip -P /home/ec2-user/environment
```

- Notice that a **code.zip** file was downloaded to the AWS Cloud9 instance. The file is listed in the **Environment** window.
- Extract the file:

```
xxxxxxxxxx
```

```
unzip code.zip
```

- Run the script that upgrades the versions of Python and the AWS CLI installed in your IDE environment, and also creates the cafe website in your AWS account.

```
xxxxxxxxxx
```

```
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 to discover this address and then paste the IPv4 address into the command prompt and finish running the script.

7. Verify the version of AWS CLI installed.

- In the AWS Cloud9 Bash terminal (at the bottom of the IDE), run the following command:

```
x
```

```
aws --version
```

The output should indicate that version 2 is installed.

8. Verify that the SDK for Python is installed.

- Run the following command:

```
x
```

```
pip show boto3
```

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

9. 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.amazonaws.com/index.html`
- Verify that the website displays by pasting the full URL into your browser.

10. Note particular website details.

- Notice in the Browse Pastries section that there are two buttons.
- The "on offer" view displays by default and it shows six menu items.
- Select the "view all" view. Notice that many more menu items display.
- Optionally, expose the developer console view in your browser.
 - For example, if you are using *Chrome*, choose View > Developer > **View JavaScript Console**
 - If you are using *Firefox*, choose Tools > Web Developer > **Web Console**.

Analysis: These menu details that display on the website are currently being read out of the **all_products_on_offer.json** and **all_products.json** files that are hosted in your S3 bucket. You also have a copy of these same files in your Cloud9 IDE, in the resources/website directory.

If you are looking at the browser developer console view details, you will also see a log entry written by the main.js file, indicating that hardcoded data is being used. Later in the lab, you will see that the number of menu items that display changes and that the messages in the console change as well.

- Leave the website open in this browser tab, you will return to it towards the end of the lab.

Task 2: Creating the first API endpoint (GET)

In this task, you will create a REST API called *ProductsApi*. You will also create the first of three resources for the API.

The first API resource will be called *products*. It will make a GET request so that the website can retrieve all rows from the *FoodProducts* DynamoDB database table. You will then deploy it in an API Gateway stage that's named *prod*. When a user visits the website, it will make an AJAX request and return a list of café menu items from API gateway (it will return mock data for now).

To complete all these tasks, you will use the SDK for Python.

11. In the AWS Cloud9 navigation pane, expand the **python_3** directory and open the file named **create_products_api.py**.

12. On line 3, replace the (*fill me in*) with the correct value that will create an API Gateway client.

Tip: Consult the [SDK for Python documentation](#) as needed.

13. Take a moment to analyze the first part of what this code will do when you run it:

- **Lines 5-24** create a *REST API* that's named *ProductsApi*, and a *resource* that's named *products*.
- **Lines 28-33** create a *method request* of type *GET* in the *products* resource.

You will analyze what the additional lines of code accomplish later in this task.

14. Run the code.

- **Save** the change to the file.
- Then, in the Bash terminal, go to the directory that contains the Python code file, and run the code.

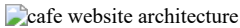
```
xxxxxxxxxx
cd python_3
python create_products_api.py
```

15. Return to the **AWS Management Console** browser tab, and open the API Gateway console.

16. Open the **ProductsApi** that you just created by choosing the link.

17. Choose the **GET** method that you defined.

You should see the details of the GET method execution in a graphical format.



18. Take a moment to study the data flow in the GET method that you defined.

Tip: If you have a screen that is large enough, arrange the browser tabs so that you have the AWS Cloud9 IDE open next to this browser tab. This way, you can see the code and what it produced side-by-side.

- On the left is the *Client*.
- **Lines 28-33** - When you run the *Test*, the *Method Request* is sent to the URL in the Amazon Resource Name (ARN) detail. The request doesn't require any authorization to invoke it.
- **Lines 50-58** - The *Integration Request* of type *MOCK* is invoked, and the mock endpoint receives the data.
- **Lines 35-48** - The mock endpoint invokes the *Integration Response*, which invokes the *Method Response*.
- **Lines 61-92** - The *Method Response* returns the REST API response back to the *Client* that the request originated from.

Analysis: To make it easier during this initial API development phase, you will use *mock data*. When you test the API call, it will not actually connect to the database. Instead, it will return the data that's hardcoded in the *responseTemplate* part of the code (lines 67-91).


This approach reduces the scope of potential errors during testing. You can stay focused in this lab on ensuring that the REST API logic is well defined.

However, the structure of this mock data intentionally matches the data structure that will appear in the next lab when Lambda will be interacting with the database table.

The key values will be mapped to the attributes that are defined in the DynamoDB table (which you created in the previous lab).

Note: attributes in DynamoDB are not primitives. Instead, they are **wrapper objects** (as shown in the example code below). This is why there is a slight difference between the key names in the JSON and the attribute names in DynamoDB.

```
product_name: {
  "S": "vanilla cupcake"
}
```

19. In the API Gateway console, choose the  **TEST** link, then scroll to the bottom and choose the **Test** button.

In the panel on the right, you should see the following response body, response headers, and log information.

```
[
  {
    "product_name_str": "apple pie slice",
    "product_id_str": "a444",
    "price_in_cents_int": 595,
    "description_str": "amazing taste",
    "tag_str_arr": [
      "pie slice",
      "on offer"
    ],
    "special_int": 1
  },
  {
    "product_name_str": "chocolate cake slice",
    "product_id_str": "a445",
    "price_in_cents_int": 595,
    "description_str": "chocolate heaven",
    "tag_str_arr": [
      "cake slice",
      "on offer"
    ]
  },
  {
    "product_name_str": "chocolate cake",
    "product_id_str": "a446",
```

```

    "price_in_cents_int": 4095,
    "description_str": "chocolate heaven",
    "tag_str_arr": [
        "whole cake",
        "on offer"
    ]
}
]

```

Congratulations! You have now successfully created and tested a REST API with a resource that makes a GET request.

Task 3: Creating the second API endpoint (GET)

In this task, you will define another API endpoint of type GET. This endpoint will eventually support calls to `/products/on_offer` from the cafe website and it will return in stock items.

20. In the AWS Cloud9 navigation pane, expand the **python_3** directory and open the file named **create_on_offer_api.py**.

21. Replace `<FMI_1>` and `<FMI_2>` with the correct values so that this code file will add another resource to the API that you defined in the previous task.

- In a browser tab, go to the **API Gateway** console and choose the **ProductsApi** API that you created a moment ago.
- In the panel on the left, choose **Resources**.
- Choose **GET** under products
- In the breadcrumb navigation at the top of the screen (above the Actions menu), you can see APIs > **ProductsAPI** followed by an id in parenthesis.
 - This is the **api_id**.
- On the same line, you will see **/products**, followed by another id in parenthesis.
 - This is the resource **parent_id**

22. Observe the rest of the code.

- The code looks very similar to the `create_products_api` code, because it is also creating a GET resource with a mock data endpoint.
- Notice that the hardcoded data response is formatted as shown below. It returns a single menu item, which will be sufficient since this is mock data.

```

[
  {
    "product_name_str": "apple pie slice",
    "product_id_str": "a444",
    "price_in_cents_int": 595,
    "description_str": "amazing taste",
    "tag_str_arr": [
        "pie slice",
        "on offer"
    ],
    "special_int": 1
  }
]

```

23. Create the API resource.

- **Save** the change to the file.
- Then in the Bash terminal, verify that the current directory is `python_3` and run the code.

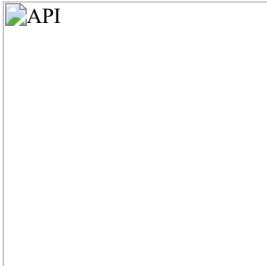
xxxxxxxxxx

python create_on_offer_api.py


24. Observe the results.

- Return to the **AWS Management Console** browser tab, and open the **API Gateway** console.
- Choose the **APIs** link in the breadcrumb navigation above, then on the left, open the **ProductsApi** by choosing the link.

Notice that there is now a nested resource called `/on_offer` under the `/products` resource.



25. Test the `/on_offer` resource.

- Use the  Test link, the same way you tested the first resource in the previous task.
- You should receive a 200 HTML status code response.

xxxxxxxxxx

```
[
  {
    "product_name_str": "apple pie slice",
    "product_id_str": "a444",
    "price_in_cents_int": 595,
    "description_str": "amazing taste",
    "tag_str_arr": [
      "pie slice",
      "on offer"
    ],
    "special_int": 1
  }
]
```

Congratulations! You now have two API resources the website will be able to use.

Task 4: Creating the third API endpoint (POST)

In this task, you will create a third resource for the API, `/create_report`. This resource will be configured at the same level as `/products` (not as a nested resource under `products`).

Café staff who are logged in (authenticated) will later use this API resource to request an *inventory report*.

The report details will be discussed in later labs. However, for now, you will configure the API to support this feature. You will also test that the website can make an Asynchronous JavaScript and XML (AJAX) request.

It is fully expected that the AJAX request will fail when you test it because you haven't configured an authentication mechanism yet. However, you will configure authentication in a later lab.

26. In the AWS Cloud9 IDE, if the `create_products_api.py` file is not already open, open it (you ran this file in Task 2).

27. Next, in the `python_3` directory, also open the `create_report_api.py` file.

28. In the main code editor window, *right-click* the `create_report_api.py` file tab and choose **Split Pane in Two Columns**.

29. Analyze and update the `create_report_api.py` code. Be sure to compare the code in this file to the `create_products_api.py` code while you do the analysis and updates.

- Replace the `<FMI_1>` that appears on line 5 with the correct value.

Tip: You could use the console as you did before to discover the `api_id`. However you can also use the AWS Command Line Interface (AWS CLI) to find the value of the API ID by running the following command:

```
xxxxxxxxxx

aws apigateway get-rest-apis --query items[0].id --output text
```

- Analyze the rest of the `create_report_api` code while comparing it to the `create_products_api` code. The code in the two files looks similar, but they have some differences:

- The `httpMethod` that's invoked is *POST* (instead of GET).
- This code creates a new resource with a *pathPart* of *create_report*, instead of *products*.
- The *product_integration_response* defines three *responseParameters*. In `create_report_api` these parameters do not allow Cross-Origin Resource Sharing (CORS), whereas in `create_products_api` they do allow it.
- The *product_integration_response* also hardcodes a response for testing purposes, though the user is not authenticated. (The purpose of the test is to ensure that the client can receive a response.)

```
xxxxxxxxxx

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

- **Save** the changes to the file.

In the next step, you will run the code in `create_report_api.py`.

30. In the terminal, confirm that you are in the `python_3` directory, and then run the code to create the third endpoint.

```
xxxxxxxxxx


python create_report_api.py
```

31. In the API Gateway console, view the details of the *report* API that you configured.

- Return to the API Gateway console tab and refresh the page.
- Confirm that you are in *ProductsApi*.
- In the navigation pane, confirm that **Resources** is selected, and choose `/create_report > POST`.

You should see the details of the POST method execution.



32. Choose the  **TEST** link, then choose the **Test** button at the bottom of the screen.

In the panel on the right, you should see the following response body, response headers, and log information.



Task 5: Deploying the API

Now that you have defined all three resources in the API, the next step is to deploy the API.

33. Deploy the API.

- Still in the API Gateway console where you have the *ProductsApi* details open, under **Resources** select the root `/`
- From the **Actions** menu, choose **Deploy API** and then fill in the details:.

- Deployment stage: [New Stage].
- Stage name: **prod**
- Stage description: (leave blank)
- Deployment description: (leave blank)

- Choose **Deploy**

Tip: If you see a warning that you do not have ListWebACLs and AssociateWebACL permissions for Web Application Firewall (WAF Regional), you can ignore the message and close it.

34. Copy the **Invoke URL** value to your clipboard. You will use it next.

Task 6: Updating the website to use the APIs

In this final task in the lab, you will update and then test the website files that are hosted on Amazon S3. After you complete these updates, the website will invoke the REST API that you just created.

35. Update the website's **config.js** file.

- In the Cloud9 IDE, open resources/website/**config.js**
- On line 2, replace null with the Invoke URL value you copied a moment ago. Be sure to surround it in quotes.

```
xxxxxxxxxx

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.
- **Save** the change to the file.

36. 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 bucket.
- Tip:** You can find the bucket name in the S3 console, or by running this command:

```
xxxxxxxxxx

aws s3 ls
```

- Notice that this script uploads the config.js file that you just editing the previous step, and uploads it to the S3 bucket.
- Save the change to the file, then run the script.

```
xxxxxxxxxx

python update_config.py
```

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

- ¹If you still have the cafe website open in a browser tab, return to it. If you do not still have it open, reopen it now by following these steps:
 - In the S3 console, choose the bucket that contains your website files.
 - Choose index.html and then copy the Object URL.
 - Load the Object URL in a new browser tab.
- If you have not already done so, expose the developer console view in your browser.
 - For example, if you are using *Chrome*, choose View > Developer > **View JavaScript Console**
 - If you are using *Firefox*, choose Tools > Web Developer > **Web Console**.
- Now, refresh the browser tab to load the changes.

The Cafe website should display.

Note: if you have an issue loading the website, verify that you are still connected to the internet from the same IP address that is hardcoded in the bucket policy.

38. Test and observe details about the website and the application logic.

- Scroll to the top of the Cafe website and choose **login**.

You will receive a message of "No API to call". This is expected. The authentication logic will be implemented in a later lab. The generate report call from the website will also be implemented in a later lab, from the webpage that will load after a successful login.

- Scroll down to the *Browse Pastries* section.

- Notice that **on offer** is chosen by default.

Recall that at the beginning of this lab, you saw six product listings here. However, now you should see only one product listing. That is because the website is now displaying the mock data that you set in the `/products/on_offer` resource you configured in Task 3.

- Choose **view all**.

You should now see three products listed (these match the mock data you set in the `/products` resource you configured in Task 2).

- Observe the log messages printed to the developer console in your browser tab.

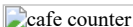
- The **main.js** file has logged that you are now using the API Gateway to get either mock or real data.
- **Note:** You may also see a message in the developer console that a request to `/bean_products` has been blocked by CORS policy. You can ignore this for now, since that is functionality that is introduced in later labs.

- Return to the Cloud9 IDE and open the `resources/website/scripts/main.js` file details in the text editor.

- Recall that at the start of this task, you set the `API_GW_BASE_URL_STR` value in the **config.js** file to match the Invoke URL value that was generated when you deployed the API.
- Here in the **main.js** application code, you can observe how the Cafe application logic notices that you are now using API Gateway to retrieve menu data. So now, when the Cafe application loads menu data, it calls the Invoke URL of the deployed API to retrieve it.

Congratulations! Your website is now making calls to the API that you created and deployed.

Update from the café

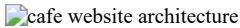
A small icon of a document with a green checkmark, followed by the text "cafe counter".

Sofia is satisfied that she has made progress!

After she successfully set up the website on Amazon S3, Sofia has been excited to improve the website's functionality. Her larger plan is to build a serverless dynamic website with a database backend. Sofia's plan has three major milestones.

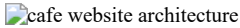
- The *first milestone* was to create a *database backend* to store café data. She accomplished that in the previous lab by using DynamoDB.
- The *second milestone* is to create a *REST API* so that the webpages that are hosted on Amazon S3 can interact with the backend database. Sofia just completed the most difficult part of that task during this lab.

The following diagram summarizes the features that Sofia has built in the last lab and in this lab:



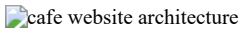
Though the API currently uses mock data, it should be straightforward to replace the mock endpoints with actual endpoints that can communicate with the database.

- The *third milestone* will be accomplished in the *next lab*. Sofia will create AWS Lambda functions. The REST API resources that she created in *this* lab will trigger those Lambda functions to query the DynamoDB table. This database table contains the actual data that she stored in the *previous lab*.



Finally, in later labs in the course, Sofia will use Amazon Cognito to implement the authentication logic that the create_report API call expects.

Sofia knows that she has work to do. For now, though, Sofia decides to celebrate her most recent accomplishment by relaxing with her friends.



Submitting your work

39. 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**.

40. 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.

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

Lab complete

Congratulations! You have completed the lab.

42. 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.

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

© 2021 Amazon Web Services, Inc. and its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.

LAB 7.1

[version_1.0.10]

Lab 7.1: Creating 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.