

[version_1.0.4]

©2019 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.

Errors or corrections? Contact us at <https://support.aws.amazon.com/#/contacts/aws-training>

Exercise: Lambda

Story So Far

You had a great night out with the team last night. Surprisingly, you feel fresh and bushy-tailed this morning, and are actually keen to get building out the site's back-end.

Steve, the assistant manager isn't in yet, and to be honest you're not really surprised. You noted last night that he really likes his tequila. Anyway, there goes your free latte.

So you head to the coffee maker and start thinking about how you are going to approach the back-end.

At this point you have a working website that points to all three of your APIs, one of which is locked down to logged in users. The problem is that your APIs are returning mock data from API GW, and there is no actual backend functionality yet.

You can either replace these mock endpoints with Lambdas, or use a server (AWS EC2).

As you empty out the coffee grinds, you remember Mike telling you he wants to be using the latest and greatest tech. You think it would probably be a good idea to set this all up "server-less", instead of pointing to EC2.

You muse over the idea of using containers with AWS Fargate, and then as dismiss it, as you look around for coconut milk. It just seems like overkill to use containers for something like this. You figure just throwing a simple script into a Lambda function is more than adequate. Besides Mike would probably still consider it cutting edge anyway (rightfully).

So as you head get back to your desk you decide you are going to create three distinct Lambda functions for each of those API resource endpoints. As you took it pretty easy yesterday pointing and clicking in the AWS console, you figure that you can push yourself today and resume working with the SDKs, and do some coding.

Here is your current game plan.

- You are going to create the back-end scripts for each of the APIs and have them interact with S3 Select. 🙋 *This is done for you by the way albeit a few <FMI>S.*
- Then you are going to package each of these back-end code bundles up to S3, and use the SDKs to create three new Lambda functions using each respective code bundle.
- Then you will test it all via the kiosk, and hope Mike likes what he sees 🤞

There are 3 scripts but only two of them are going to do anything interesting. The 2 GETs will use S3 Select to read the latest Amazon.com reviews file for cameras, and return a subset of these reviews (that are in stock). The `create_report` one, will simply extract the cellphone and IP from the users login (that you set up on Lab 3) and return a standard "report processed" ...for now.

You will learn in this lab:

- How to package up and prepare your Lambda code for deployment from S3.
- How to create Lambda functions using the SDK
- How to update Amazon API Gateway resources to point to functions (instead of just returning mock data).

You will also find out if Steve comes in at all today. You get the feeling staff are taking bets.

Accessing the AWS Management Console

1. At the top of these instructions, click **Start Lab** to launch your lab.
2. A Start Lab panel opens displaying the lab status.
3. Wait until you see the message "**Lab status: ready**", then click the **X** to close the Start Lab panel.
4. At the top of these instructions, click **AWS**

This will open 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, there will typically be a banner or icon at the top of your browser indicating that your browser is preventing the site from opening pop-up windows. Click on the banner or icon and choose "Allow pop ups."

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, to make it easier to follow the lab steps.

Setup

1. Ensure you are in **Cloud9**. Choose **Services** and search for **Cloud9**. You should see an existing IDE called Building_2.0. Click the button **Open IDE**. Once the IDE has loaded, enter the following command into the terminal: *(This command will ensure that you are in the correct path)*

```
cd /home/ec2-user/environment
```

2. You will need get the files that will be used for this exercise. Go to the Cloud9 **bash terminal** (at the bottom of the page) and run the following `wget` command:

```
wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-Building_2.0/lab-4-lambda.zip
```

3. Unzip:


```
unzip lab-4-lambda.zip
```

4. Let's cleanup:

```
rm lab-4-lambda.zip
```

5. Run the `resources/setup.sh` script that will grab the website contents and upload them to the S3 bucket created by our CloudFormation template.

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

 **If you are using Java** you will also need to run the following scripts and commands:

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

Lab Steps

Let's start with the `get_reviews` one

1. Navigate to your respective code folder on the `CMD_LINE`.
2. Open the respective `get_reviews` file and look at the code, then do the same for the `get_reviews_code` file. There are no `<FMI>` entries to do in either of these scripts. However you should read the code to see what it is doing anyway.

⚠ **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to open the parent folder to find the `App.java` file. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `get_reviews` or `get_reviews_code` for node and python, therefore the folder you should navigate to in the java folder is the `get_reviews` or `get_reviews_code` folder. Under that folder you should see a `build.gradle` file - that is how you know you are in the right spot. From there you can drill down into the project structure to find the `App.java` file.

3. Once you have read through the two code files. Head over to the `CMD_LINE` and zip up the `get_reviews_code` file(s) into a zip file called `get_reviews.zip`.

⚠ You'll need to change directories to the correct location (`get_reviews_code`) to run the command.

📎 Use the respective commands for this:

Language	CMD
python_3.6.8	<code>zip get_reviews.zip get_reviews_code.py</code>
node_10.8.1	<code>zip get_reviews.zip get_reviews_code.js</code>
java_8	<code>gradle build</code>

Expected output:

```
adding: get_reviews_code.xx (deflated 44%)
```

Expected output for Java:


```
BUILD SUCCESSFUL in 21s
4 actionable tasks: 4 executed
```

So now we have a zipped up asset with code in it. **Awesome!**

⚠ **If you are using Java** change directories up one level (to do this run `cd ../` in the command line) to the `java_8` folder to go through the next steps

We are now ready to tell Lambda to use this in a newly created function that we are going to call `get_reviews`.

4. You will need to upload this newly created `get_reviews.zip` to a S3 bucket using the respective `uploadToS3` script in your resources folder.

 To save on unnecessary lab steps. I am going to have you shove this zip file in our existing website folder. Lazy I know, in prod you would obviously use a private bucket to store your lambda code.

So again you will need to choose your language specific `uploadToS3` script. As follows:


Language	CMD
python_3.6.8	<code>chmod +rx ../resources/uploadToS3_python.sh && ../resources/uploadToS3_python.sh</code>
node_10.8.1	<code>chmod +rx ../resources/uploadToS3_node.sh && ../resources/uploadToS3_node.sh</code>
java_8.	<code>chmod +rx ../resources/uploadToS3_java.sh && ../resources/uploadToS3_java.sh</code>

You should get an output similar to this:

```
upload: ./get_reviews.zip to s3://c11284a125438u294892t1w468500737961-  
s3bucket-iax7xho9pjdo/get_reviews.zip
```

5. Ok great, you have your code all ready to be consumed by your "about to be created" Lambda function.

You will use the SDK for this bit.

 You don't need to alter anything in that code as there are no `<FMI>`s. You simply need to run the respective script.

You should still be in your correct language folder in the `CMD_LINE`. If so run `get_reviews` from that folder.

 **If you are using Java**, change directories into the `get-reviews` folder that contains a `pom.xml` file.

Like this:

Language	CMD
python_3.6.8	<code>python3 get_reviews.py</code>
node_10.8.1	<code>npm install aws-sdk && node get_reviews.js</code>
java_8	<code>mvn clean install && mvn exec:java - Dexec.mainClass=com.mycompany.app.App</code>

Output should give you


```
Done
```

For Java users, if you see the following warning in the output, please ignore it. You should still see the word `Done` somewhere in the output.

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
```

```
SLF4J: Defaulting to no-operation (NOP) logger implementation
```


```
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

 **NODE users only:** As this was the first time you have execute a node function in this lab, you will notice (above) that you needed to run `npm` to install the `aws sdk`. So you might get some other fluff output / warnings. Just ignore all that, its fine.

6. Choose **AWS Cloud9**. Choose **Go To Your Dashboard**. Choose **Services** and search for **Lambda**. Choose the `get_reviews` function.

7. Choose **Select a test event** next to **Test**. Choose **Configure test events**. Name the event `productid`. Paste in the following:

This ID is a known product ID for a camera. So we can get *REAL* data at last instead of all this mock stuff we've been using up to this point.

 Remember it's API GW that maps `product_id` to `product_id_str`, to demonstrate the mapping feature. Hence this we use `product_id_str` when testing in Lambda directly.

```
{
  "product_id_str": "B00WBJGUA2"
}
```


Choose **Create**. Choose **Test**. You should see something like the following:

```
{
  "product_id_str": "B00WBJGUA2",
  "reviews_arr": [
    {
      "review_headline_str": "Good camera but WIFI access has a major
flaw if you have mutiple Access Points with the same name.",
      "review_body_str": "Both the dropcam and nest cam have an
embarrassingly bad WIFI algorithm when there are multiple access points with
the same name (SID) near it. (I have a tall house and I need multiple WIFI
access points) when you have this situation, the cameras lose connectivity
all the time. The obvious workaround is to dedicate a WIFI access point
specifically for the Nest Cam, which is annoying. why Nest can't or won't
fix this is beyond me. I know of no other WIFI enabled device that is this
&#34;dumb&#34; about WIFI connectivity. Until this is fixed it stays a 3."
    },
    {
      "review_headline_str": "Great product software needs work",
      "review_body_str": "It was easy to setup with a small hiccup
during the scanning of the barcode on the back. I still have issues with the
software not loading correctly on my phone which customer service has said
they are working on fixing. The app hangs quite often when loading it from a
push notification where I either get single spinners or double spinners.<br
/><br />I do wish the monthly/yearly fees for video retention were better or
there was maybe a network based solution for video storage as I would like to
buy more of these and use them as a whole house system but would get quite
pricy"
    }
  ]
}
```

```

    {
      "review_headline_str": "High quality product, great image,
somewhat useless without subscription",
      "review_body_str": "I've had this device for a few weeks now
and I really like it. It was easy to setup and it's easy to use. I already
have a Nest thermostat which I love and I now use the same app (on Android)
to manage the camera. It is really cool to be able to view the camera from
my phone wherever I am. There are some small kinks which seem to need work
in the app. For example, clicking on the notification will open the app and
infinitely try to load the image from the camera history. If you don't pay
for the history it was just infinitely load... you could wait an hour it will
never load an image. You have to back out of the app and open it again to
see the image. Also, the camera should come with at least one day or a few
hours of video history included for free. It would be great to have the
option to cache video history to my own computer or network device. Without
paying the subscription fee you have ZERO video history. You will get a
notification that the camera detected motion.... but you can't see it because
it's usually over before you can open the app. The camera is pretty much
useless without video history... but the prices for history are not cheap.
If you don't mind paying a monthly fee... it's a great device with excellent
build quality and image quality."
    },
    {
      "review_headline_str": "unrefined, buggy",
      "review_body_str": "I was hoping to use this for outdoor
surveillance. Proved to be too difficult to isolate zones where breezy
plants wouldn't trigger unwanted alerts. On one occasion, I received motion
alerts when camera was allegedly off, which made me uncomfortable about when
video was/wasn't being sent to cloud. App had a bad habit of turning off my
motion zones so my alerts were not useful. Camera pours off heat. Seems
overall like an unrefined product not on par with the Nest thermostat which I
own and like."
    }
  ]
}

```

 I think it is worth covering something here that is often overlooked when people move from a mock API Gateway resource to a Lambda function as the back end.

The request that your Lambda receives when you wire it up with API Gateway, depends entirely upon how you set up API Gateway to work with Lambda.

There are 2 main approaches, and you should be aware of them.

Option 1: Choose "Lambda Proxy" and have API Gateway send everything it can possibly extract out of the HTTP request and package it up into an object called "event". Then your Lambda can dig around in that object to get what you need (e.g the query string parameters) and customize a response from the handler. This approach is a hammer to an egg and although less hassle to set up. It really is a bit of overkill in our case where we only want one value called `product_id` from the query string.

OR

Option 2: Tell API Gateway to map a known query string key like `product_id` to the event object in Lambda.

We are going to be using the latter. Yes, it requires you to set up the mapping for this to work, but you have already done it! We already went to the trouble of setting up the mapping when we did the mock `#win`.


Ok, back to the lab ;)


Perfect. We have the `get_reviews` Lambda function all good to go and ready to be wired up to API gateway.

Before we do that let's create next Lambda function for the `get_av_star_rating` API Gateway endpoint. We just need to repeat steps, here we go....

8. Switch back to the **Cloud9** tab. Just like before you will need to zip up then upload your new `get_ratings.zip` to the same S3 bucket using the same respective `upload_ToS3_XXXX` script.

First things first **zip up the code** using the correct command for your language:

 I suggest you read the `get_ratings_code` file so you know what it is doing. It is very similar to the last one you did (`get_reviews`).

 **If you are using Java** the folder structure for java is different from for Node.js or Python. You'll need to open the parent folder to find the `App.java` file. The parent folder name will correspond with the file name for the other languages. In this case, the file is named `get_ratings_code` for node and python, therefore the folder you should navigate to in the java folder is the `get_ratings_code` folder. From there you can drill down into the project structure to find the `App.java` file. Ensure you are in the correct folder on the CLI when you run the gradle build command.

Language	CMD
python_3.6.8	<code>zip get_ratings.zip get_ratings_code.py</code>
node_10.8.1	<code>zip get_ratings.zip get_ratings_code.js</code>
java_8	<code>gradle build</code>


Output will be similar to this:

```
adding: get_ratings_code (deflated 39%)
```

For Java the output would be similar to this:

```
BUILD SUCCESSFUL in 2s
4 actionable tasks: 4 executed
```

Open your respective shell script file in your resources folder called `uploadToS3_XXXXX.sh` in Cloud9 (double click it).

 Before you run this you will need to modify it.

Once you have the file open in Cloud9 and make sure to do the following:

A) *uncomment* out the line that uploads the `get_ratings.zip`

B) *comment* out the `get_reviews` one (as you don't need to re-upload that one)

As in this example (*If you are working with Python of course*):

```

Before:
aws s3 cp ~/environment/python_3.6.8/get_reviews.zip
s3://$bucket/get_reviews.zip
#aws s3 cp ~/environment/python_3.6.8/get_ratings.zip
s3://$bucket/get_ratings.zip
#aws s3 cp ~/environment/python_3.6.8/create_report.zip
s3://$bucket/create_report.zip

After:
#aws s3 cp ~/environment/python_3.6.8/get_reviews.zip
s3://$bucket/get_reviews.zip
aws s3 cp ~/environment/python_3.6.8/get_ratings.zip
s3://$bucket/get_ratings.zip
#aws s3 cp ~/environment/python_3.6.8/create_report.zip
s3://$bucket/create_report.zip

```

Choose **File** and **Save**.

Now run it as follows (just like before). You should still be in your respective language folder in the `CMD_LINE`.

⚠ For Java users, run `cd ../` to back up out directory to successfully run the following command.

Language	CMD
python_3.6.8	<code>../resources/uploadToS3_python.sh</code>
node_10.8.1	<code>../resources/uploadToS3_node.sh</code>
java_8.	<code>../resources/uploadToS3_java.sh</code>

This will give you output similar to this:

```

upload: ./get_ratings.zip to s3://c11284a125438u294892t1w468500737961-
s3bucket-cy38ogpmfrvz/get_ratings.zip

```

9. Now you have your code zipped up and in S3. Its time to create the Lambda function.

This time the Lambda function you are about to create will be employed with the `get_av_star_ratings` API Gateway endpoint (later).

Create your second lambda function using the respective `get_ratings` script:

👤 Read the code in `get_ratings` so you can see what it is doing.

⚠ **If you are using Java**, change directories into the `get_ratings` folder that contains a pom.xml file.

Language	CMD
python_3.6.8	<code>python3 get_ratings.py</code>
node_10.8.1	<code>node get_ratings.js</code>
java_8	<code>mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App</code>


Output should give you:

Done

For Java users, if you see the following warning in the output, please ignore it. You should still see the word `Done` some where in the output.


```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".  
SLF4J: Defaulting to no-operation (NOP) logger implementation  
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further  
details.
```

10. Just like before, head over to the **Lambda** service in the AWS console. Choose **Functions** and this time choose **get_ratings**. Choose **Select a test event** next to **Test**. Choose **Configure test events**. Name the event `productid`. Paste in the following (which is a valid product id):

 Remember it's API GW that maps `product_id` to `product_id_str`, to demonstrate the mapping feature. Hence this we use `product_id_str` when testing in lambda directly.

```
{  
  "product_id_str": "B00WBJGUA2"  
}
```

Choose **Create**. Choose **Test**. The *REAL DATA* output should look similar to the following:


 Note the `statusCode` and `body` keys, that we need for this to work when we hook this up with API gateway.

```
{  
  "product_id_str": "B00WBJGUA2",  
  "average_star_review_float": 3.25  
}
```

Perfect. You are almost done, one more left. We just need to do the `create_report` function now.

11. For `create_report`. Switch back to the **Cloud9** tab. Make sure you are in the right language directory in the `CMD_LINE`:

You are going to go through a similar process as with the other 2 functions.

 **If you are using Java** remember the folder structure is different. Navigate to the `create_report_code` folder on the command line, and ensure the folder you are in contains the `gradle.build` file.

Zip up the code using the correct command for your language:

Language	CMD
python_3.6.8	<code>zip create_report.zip create_report_code.py</code>
node_10.8.1	<code>zip create_report.zip create_report_code.js</code>
java_8	<code>gradle build</code>

Output will be similar to this:

```
adding: create_report_code (deflated 39%)
```

For Java users, output will be similar to this:

```
BUILD SUCCESSFUL in 2s
4 actionable tasks: 4 executed
```

Just like before you will now need to upload this using your uploadToS3 script.

⚠ Again you will need to modify this script first.

Double click the file in Cloud9 and make sure to:

A) *uncomment* out the line that uploads the `get_ratings.zip`

B) *comment* out the reviews one (as you don't need to re-upload that one)

As in this example (*If you are working with Python of course*):

```
Before:
#aws s3 cp ~/environment/python_3.6.8/get_reviews.zip
s3://$bucket/get_reviews.zip
aws s3 cp ~/environment/python_3.6.8/get_ratings.zip
s3://$bucket/get_ratings.zip
#aws s3 cp ~/environment/python_3.6.8/create_report.zip
s3://$bucket/create_report.zip

After:
#aws s3 cp ~/environment/python_3.6.8/get_reviews.zip
s3://$bucket/get_reviews.zip
#aws s3 cp ~/environment/python_3.6.8/get_ratings.zip
s3://$bucket/get_ratings.zip
aws s3 cp ~/environment/python_3.6.8/create_report.zip
s3://$bucket/create_report.zip
```

Choose **File** and **Save**.

Now run it as follows (just like before). You should still be in your respective language folder in the `CMD_LINE`.

⚠ For Java users, run `cd ../` to back up out directory to successfully run the following command.

Language	CMD
python_3.6.8	<code>../resources/uploadToS3_python.sh</code>
node_10.8.1	<code>../resources/uploadToS3_node.sh</code>
java_8.	<code>../resources/uploadToS3_java.sh</code>

This will give you output similar to this:

```
upload: ./create_report.zip to s3://c11284a125438u294892t1w468500737961-
s3bucket-cy38ogpmfrvz/create_report.zip
```

12. Finally, just run your create report script as follows: *(After you have read through it)*

⚠ **If you are using Java**, change directories into the `create_report` folder that contains a `pom.xml` file.

Language	CMD
python_3.6.8	<code>python3 create_report.py</code>
node_10.8.1	<code>node create_report.js</code>
java_8	<code>mvn clean install && mvn exec:java -Dexec.mainClass=com.mycompany.app.App</code>

Output should give you:

```
Done
```

13. Switch back to the **Lambda** tab. Choose **Functions** and choose **create_report**. Choose **Select a test event** next to **Test**. Choose **Configure test events**. Name the event `empty`. Keep the request payload `empty`. If you recall from the prior lab this API endpoint will be authenticated with Cognito which gives us the logged in users phone number automatically ;). We don't need to pass in an additional payload here.

```
{
}
```

Choose **Create**. Choose **Test**. The output should be as follows.

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

OK this is great, let's move on.


Your Lambdas are all working perfectly. However you now need to replace the old MOCKs you had in your API Gateway setup with these 3 functions.

Part 2 - Wire up your Lambda function to API Gateway

1. Choose **Services** and **API Gateway**. Choose the **Fancy-API**.

We will adjust each resource, starting with `get_reviews`.

Choose `/get_reviews` and **GET**. Choose **Integration Request** and for **Integration type** change it to **Lambda Function**.



2. Change the **Lambda Region** to `us-west-2`. Start typing in `get_reviews` which should give you a drop down option. Choose **Save**. Choose **OK**. Then **OK** again though any warnings.
 These modal are tell you that you are about to allow API gateway to interact with Lambda.
3. Test it by choosing **Method Execution**. Choose **TEST**.
4. Paste the following into the **Query Strings** field:

```
product_id=B00WBJGUA2
```


5. Choose **Test**. An error will be present or something similar to the following:

```
{
  "reviews_arr": []
}
```

What do you notice? The response There is are no reviews, and no `product_id`. We know there are reviews for this item, and we know that `product_id` gets mapped to `product_id_str`, so what has happened?

 Well, when you apply a Lambda function to API Gateway. The mapping that was created in the **integration request** gets removed .

Luckily though, we know what the mapping should be. It just needs to be pasted back in. Let's do that now, to fix this.

6. Choose **Method Execution**. Choose **Integration Request** and scroll down to and expand **Mapping Templates**.
7. Choose **When there are no templates defined (recommended)**. Choose **Add mapping template**. Type in `application/json` under **Content-Type**. Choose the checkmark next to it.
8. See it's gone . So paste the following into the text area below it:

```
{
  "product_id_str": "$input.params().queryString.get('product_id')"
}
```

Choose the **Save** button below the text area.

This way, the event object passed from API Gateway to Lambda will end up looking like this, which is what we want.

```
{
  product_id_str: "B00WBJGUA2"
}
```

We also (🔗) need to update the **Integration Response**. Choose **Method Execution** at the top. Choose **Integration Response**. Expand the area by choosing the black arrow next to 200.

Expand the **Header Mappings** section.

Edit the values as follows by clicking on the pencil icon next to each one. Note they are not there any more (🔗).

Edit them as per this table, then submit each one by choosing the respective checkmarks.

Response header	Mapping value	
Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'	
Access-Control-Allow-Methods	'GET'	
Access-Control-Allow-Origin	'*'	

⚠ Important. They are in single quotes.

Click the **Save** button.

Go back to the **get_reviews** dashboard by choosing **Method Execution** at the top left. Choose **TEST**.

Again paste the following into **Query Strings**:

```
product_id=B00WBJGUA2
```

Choose **Test**. **This time** due to your cool new mappings, the output should now show the `product_id_str`, `review_headline_str`, and `review_body_str`.

A bit like this:

```
{
  "product_id_str": "B00WBJGUA2",
  "reviews_arr": [
    {
      .....review info should be in here..
    }
  ]
}
```

Awesome. Fixed!

Now, onto the ratings one.

9. Choose **/get_av_star_rating** and choose **GET**. Choose **Integration Request**. Choose **Lambda Function** and choose `us-west-2` for the **Lambda Region**. In the **Lambda Function** field type in `get_ratings` and choose it from the list.
10. Choose **Save**. Choose **OK** and then **OK** once again to get through the helper modals.
11. As we know that the console will remove your lovely mapping, let's quickly add them back in here. Expand **Mapping Templates** and choose **When there are no templates defined (recommended)**. Choose **Add mapping template**. Type in `application/json` and choose check mark next to it as before.

12. Paste in the following:

```
{
  "product_id_str": "$input.params().querystring.get('product_id')"
}
```

13. Choose **Save** below the text area. Choose **Method Execution** at the top left. Choose **TEST**. Paste the following into **Query Strings**:

```
product_id=B00WBJGUA2
```

Choose **Test**. The output should be similar to the following:

```
{
  "product_id_str": "B00WBJGUA2",
  "average_star_review_float": 3.25
}
```

Again we also need to update the **Integration Response** if we want calls from our website to work ;). Choose **Method Execution** at the top. Choose **Integration Response**. Expand the area by choosing the black arrow next to 200.

Expand the **Header Mappings** section.

Edit the values as follows by clicking on the pencil icon next to each one. Then submit the change by choosing the checkmark.

Response header	Mapping value	
Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'	
Access-Control-Allow-Methods	'GET'	
Access-Control-Allow-Origin	'*'	

⚠ Yes, they are in single quotes.

Choose **Save**.

Awesome. Just one more to do, nearly done!

14. Choose **/create_report** and choose **POST**. Choose **Integration Request**. Choose **Lambda Function** and choose **us-west-2** for the **Lambda Region**. In the **Lambda Function** field type in **create_report** and choose it from the list. Choose **Save**. Choose **OK** and **OK** again to get through the warnings.

Like before you will need to update the **Integration Response**. Choose **Method Execution** at the top. Choose **Integration Response**. Expand the area by choosing the black arrow next to 200.

Expand the **Header Mappings** section.

Edit the values as follows by clicking on the pencil icon next to each one. Then submit the change by choosing the checkmark.

Response header	Mapping value	
Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'	
Access-Control-Allow-Methods	'POST'	
Access-Control-Allow-Origin	'*'	

⚠ Yes, they are in single quotes.

Choose **Save**.

Finally, you need to make sure that your Lambda gets the **header** information (the bearer token stuff). So you need to adjust the mapping for the Cognito Authorization.

We could write a specific mapping statement here to just extract the Authorization header. For speed (👉 *and to expose a cool template that is hammer-2-egg useful*) let's choose a full passthrough template.

Follow these steps to allow for a full passthrough.

Choose **Method Execution**. Choose **Integration Request**. Expand the **Mapping Templates** section. Choose **When there are no templates defined (recommended)**. Choose **Add mapping template**.

Type in `application/json` and choose the check mark next to it.

Under the Generate Template drop down, choose **Method Request passthrough**. 👉 *This is very similar to how Lambda proxy works btw. Pretty helpful rt?*

Choose **Save**.

Now the header will be in the event object discoverable inside our Lambda code.

You won't need to reapply the OPTIONS for `create_report`, these settings do not get overridden when you switch to Lambda (small mercies ;). 👉 You can check if you like; you will see it has `Access-Control-Allow-Credentials` set to `true` and the `Access-Control-Allow-Origin` is your website domain.

15. Choose **Method Execution** and choose **TEST**. Choose **Test** at the bottom. The output should look similar to the following:

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


👉 You might be thinking, wait. Didn't we lock down this API Gateway endpoint down with a **custom authenticator** for Cognito? 😬 I wonder if API Gateway removed that too, when we switched this from MOCK to Lambda?

Good news. No.

All that is actually happening is that the API Gateway test functionality in the AWS console does not take into account authentication 😬. In order to test that this is actually being blocked without a valid bearer token. You need to deploy the API and then test it working via the website (i.e with a valid token) or show it failing with CURL (i.e without a valid token).

Let's watch it fail first 😊. Let's CURL `create_report` and make sure that you can't access without a valid token.

16. We are not going to test with a valid token as we will be doing that shortly when we login via the website. For now let's **deploy the app** and make sure we can't get in via CURL when we don't pass any token in the header.

To deploy. Choose the top resource . Then choose **Actions** and **Deploy API**. Choose **test** for **Deployment stage**. Choose **Deploy**. *Ignore the WAF warnings*. Copy down the **Invoke URL**. It should look *a bit like* this :<https://ppdoo7f7o4.execute-api.us-west-2.amazonaws.com/test>.

17. Switch back to the **Cloud9** tab. We only need to test the `create_report` endpoint via **CURL** via your Cloud 9 `CMD_LINE`.

Replace the `<FMI>` below will be your Invoke URL. (*Lose any trailing slash on you URL*)

```
curl -X <FMI>/create_report
```

Example:

```
curl -X POST https://1j820xv8a3.execute-api.us-west-2.amazonaws.com/test/create_report
```

This should give you REAL data at last via your endpoint, a bit like this:

```
{"message": "Unauthorized"}
```

Perfect, our lockdown is still in place. 🛡️ Despite what the API Gateway test via the console tells us 😊

Finally it is time to test everything via the website. Remember at the start of the lab you ran the setup script? That means you're website is good to go. No further steps are needed as your API is tweaked to use the same URL #winning.

Visit your website URL in your browser. You can grab this URL by navigating from the **AWS Management Console**. Choose **Services** and select **S3**. Choose the bucket. Choose the **Properties** tab. Choose the **Static Website hosting** section. Copy down the **Endpoint URL**.

18. At the website you should be able to test the `get_reviews` and the `get_ratings` functionality and get real data coming back. Check they work. In this lab example use `camera` for the query under **Product Finder**.
19. The function that should fail (as you are not logged in) will be `create_report`.

So choose **REQUEST A REPORT** on the website, and see what it says. It should say **You Need To Be Logged In To Create A Report**

Looks like we need a token before the website will even attempt to call that API. We currently don't have one.

We are going to get a real token to use, and hopefully it will give us the response we want which will be **"Report Processing"**.

In order to get a valid token into the website you need to choose the **Admin Login** at the top left on the website. This will take you to the hosted login page from Cognito.

Use these credentials..


```
username: ricky
password: !FooBar55
```

Once you are logged in successfully you should be redirected back to your website as the callback URL you programmed into Cognito in Lab 3 was `/callback.html`.


20. Now you are back at the site. The website's JavaScript should have stored the credentials from Cognito and allow you to make the AJAX call. You should not longer get the warning when you try and use the `REQUEST A REPORT` button.

This time you should see the right response. So everything worked.

In a few moments you should see this:


```
Report Processing
```

This is a different message than when you tested it earlier right? This is different because the function looks at the Cognito payload (which now exists thanks to your logging in) and can extract information from the user.

 If you are curious you can look at the network panel in chrome dev tools to see that we return all the info back to the browser (just to show it working).

```
cell_str: "+1xxxxxxxxx"
ipv4_str: "xx.182.xxx.6"
message_str: "Report Processing"
name_str: "ricky"
```

Perfect. You now have 3 Lambda functions all working and tested

 Admittedly the `create_report` one doesn't do much in terms of report creation, but you will address that in the next lab, where you will set up all the background processes.

Awesome. You call Mike over to check out the new kiosk app. You show him the live data on the website and how to get reviews and ratings, and then how his managers would log in via Cognito to request a report.

When we see the message "Report Processing". He looks excitedly at his phone for the report. Nothing happens and he looks at you raising his eyebrows.

He is (without words) asking you when that report stuff will be ready.

You stumble on your own thoughts for a second, and then offer up enthusiastically: "Oh right....yeah...end of day tomorrow!".

He smiles and pats you confidently on the shoulder, seeming pleased that you knew what the eyebrow maneuver meant. He walks off still checking his phone, just in case.

Lab Complete

Congratulations! You have completed the lab.

1. Click **End Lab** at the top of this page and then click **Yes** to confirm that you want to end the lab.
2. A panel will appear, indicating that "DELETE has been initiated... You may close this message box now."
3. Click the **X** in the top right corner to close the panel.

For feedback, suggestions, or corrections, please contact us at: <https://support.aws.amazon.com/#/contacts/aws-training>