


©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: Optimizing

Story So Far

Looking over the top of your laptop's screen at the clock on the wall. You take a deep breath. Look down at your Cloud9 environment and think to yourself "I've got this .

Your app works, but you need to show Mike that you have some form of monitoring in place. Ideally, prove the value of said monitoring.

You figure that if can show an actual measurable improvement from the current system. That will be the icing on the cake.

Here is your current game plan.

- Using just one API `get_reviews` you will set up Metrics using AWS X-ray.
- You will improve the performance of the Lambda function by working with context re-use.
- You will deploy Amazon API Gateway Cache and demonstrate the improvements with data.

You will learn in this lab:

- How to set up and work with AWS X-Ray.
- The benefits of re-using Lambda Context.
- How to leverage Amazon API Gateway caching.

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**

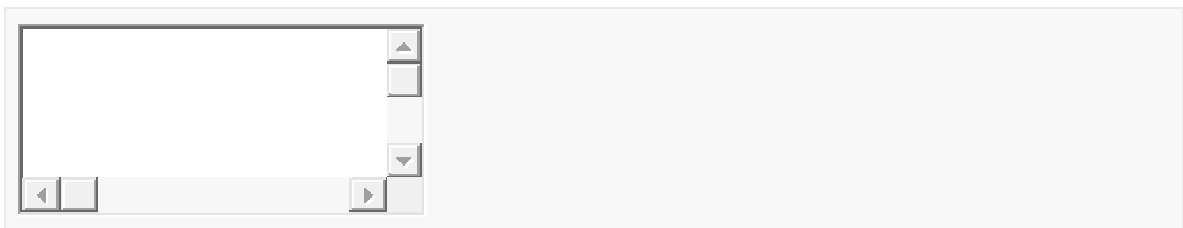
This will open the AWS Management Console in a new browser tab. The system will automatically log you in.

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.

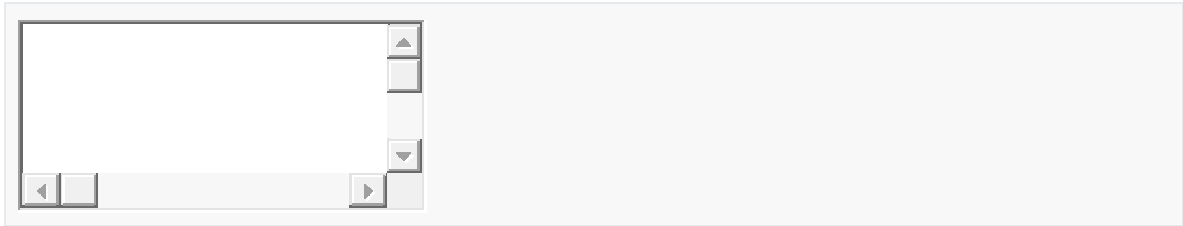
Setup

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).



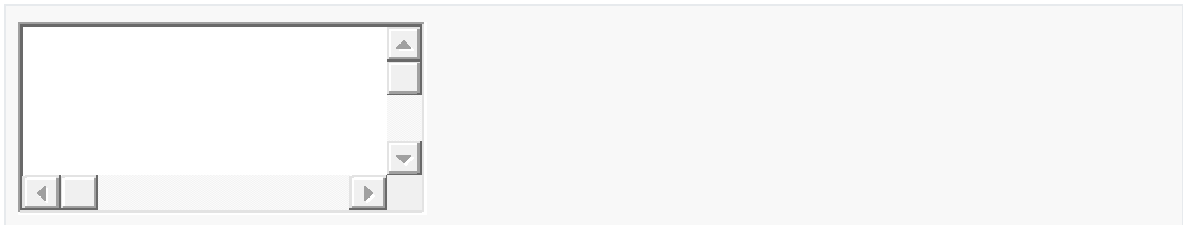
```
cd /home/ec2-user/environment
```

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:



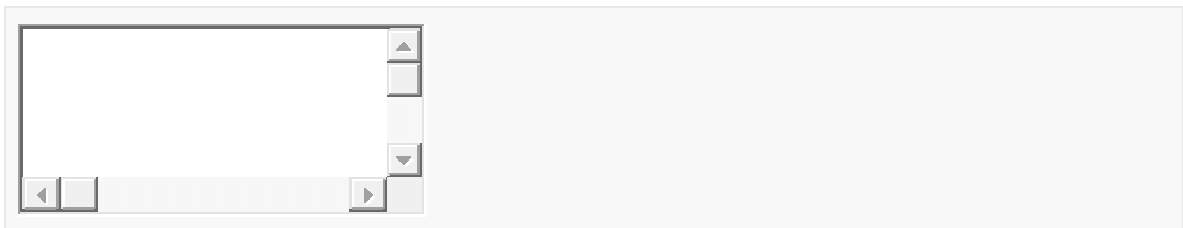
```
wget https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-Building_2.0/lab-6-optimizing.zip
```

3. Unzip:



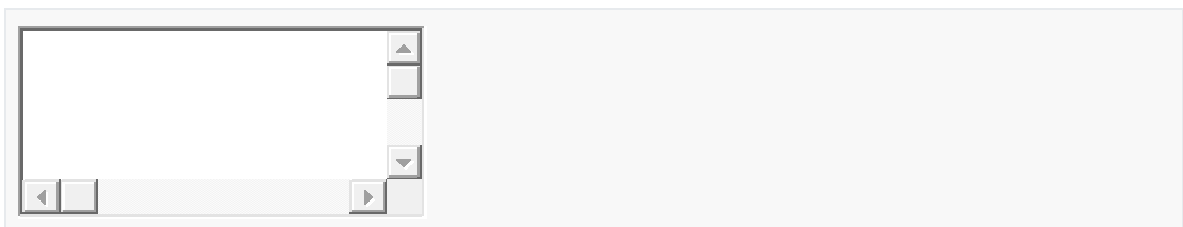
```
unzip lab-6-optimizing.zip
```

4. Let's cleanup:



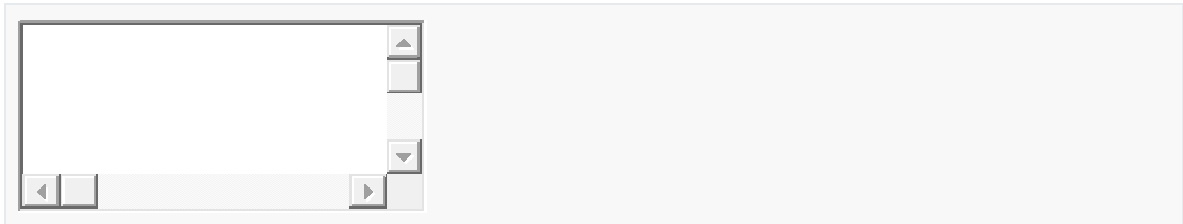
```
rm lab-6-optimizing.zip
```

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
```

6. This will prompt you to provide your email, cellphone and IPv4. You can get your Ipv4 from <https://www.whatismyip.com/>. Please use your real email and cellphone (no + country code needed).
7. ⚠ **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

We now wish to optimize the app, however we don't want to optimize every single part of the app as that would be a whole course.

For the purposes of this Lab we will take one of our GET APIs, namely `get_reviews` and see if we can optimize its performance.

Add instrumentation

Without seeing what is happening we are flying blind!

We need to capture information on how this API is performing.

1. Choose **AWS Cloud9** and choose **Go To Your Dashboard**. Choose **Services** and search for **Lambda**.
2. Choose the **get_reviews** function. Scroll down to **Monitoring tools**. Choose **Edit**. Under **AWS X-Ray** enable **Active tracing**.
3. Choose **Save**.
4. Copy the **Callback URL** that was output from the setup script. Paste it into the browser replacing `/callback.html` with `/index.html`.
5. Where it says **Start typing your search phrase...** enter in something like `camera`. Choose reviews.
6. Switch back to the **Lambda** tab. Choose **Services** and **AWS X-Ray**. You should see some data in the **Service map**. If you don't see anything change to **Last 1 minute** at the top right and refresh.

7. Choose **Traces** and choose one from the **Trace list** to drill down to see more details.
8. You can also look at additional detail under **Analytics**. Feel free to close the X-Ray tab.

You should see something similar to this.

In this example you can see it takes 6.5 seconds. 🤖♂ *Yours will likely be different.*

Code 1 - Re-structure the app, make use of context

Now you are going to change some of the code in your `get_reviews` function, to see if a code change can improve on that time

1. Navigate to your respective code folder on the `CMD_LINE`.

⚠ **If you are using Java** You'll need to navigate to the parent folder on the command line for the program to run. The parent folder name will correspond with the file name for the other languages. Under that folder you should see a `build.gradle` file - that is how you know you are in the right spot :).

2. Open the respective `get_reviews_code` file and look at the code. There are no `<FMI>` entries to do in either of these scripts.

⚠ **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_code` for node and python, therefore the folder you should navigate to in the java folder is the `get_reviews_code` folder. From there you can drill down into the project structure to find the `App.java` file.

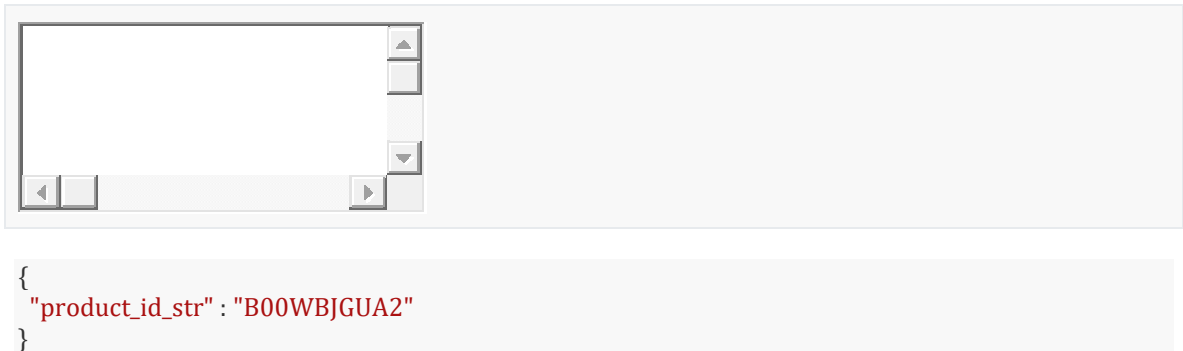
You will see the AWS service client/resource object for you respective language `var` block is inside the handler.

By creating the connection to Amazon S3 inside of a handler, or inside of a method/function, it must be created every time the code is run. We can improve this by leveraging the context's ability to be reused. Objects


declared outside of the function's handler method remain initialized, providing additional optimization when the function is invoked again.

To prove this point. Let's run the Lambda function with the service objects declared *in* the handler first then change the code and run it again. Then we can view the difference in performance in the X-Ray console.

3. From **Cloud9** choose **AWS Cloud9** in the upper left hand corner and choose **Go To Dashboard**. Choose **Services** and **Lambda**. Choose the **get_reviews** function.
4. Create a new test for this Lambda function by selecting dropdown next to the **Test** button and select **Configure Test Events**.
5. Name the test event `test`
6. Paste the following into the text area:



7. Choose **Create**.
8. Invoke the Lambda function by pressing the **Test** button with the `test` event selected in the drop down. The `get-reviews` function should execute successfully.
9. Now press the **Test** button a second time, invoking the function again. This should also be successful. This second test will be important when it comes time to look at the traces in X-Ray. The first invocation will be much longer than the second due to execution context re-use. More on that in a bit ;)
10. It's time to update the code to take advantage of context reuse. Go back to your **Cloud9** tab.
11. Under your respective code folder, you should see a file called `code_1`. Open this file and review the code.

 Notice how the AWS connections to the service through the client objects are declared at the top, along with any other objects that could be re-used across invocations. By placing these declarations here it should cut down on

the execution time of the Lambda function if a context is available to be re-used.

12. Copy the code from the `code_1` file and paste it into the `get_reviews_code` file for your language in the **Lambda** tab. Essentially, replace the old code with the new code. Choose **Save**.

⚠ If you are using Python you may need to alter the Handler name in basic settings to `index.lambda_handler` and the Runtime to `Python3.8` and rename your file `index.py`

⚠ If you are using Java, you need to alter the Handler name in the basic settings to `com.mycompany.app.App::handleRequest`, the Runtime to `java8`, the memory to `448`, and the timeout to `90`.

15. Now that your function code has been updated. Let's test it **twice** again, just like you did before. Go back to the **Lambda** tab and make sure you select the `get_reviews` function.
16. The test event we created before should still be there. Press the **Test** button with the `test` event selected from the drop down. Your Lambda function should execute successfully.
17. Press the **Test** button again to invoke the Lambda function a second time.

Your Lambda function should execute successfully this time as well.

18. Now here's the fun part. Switch back to the **AWS X-Ray** tab. Choose **Traces**. Change to **Last 5 minutes** and refresh.
19. This page will show you a `Trace list` section and you should see multiple traces for the `get_reviews` Lambda function. Sort these traces by clicking on the **Age** tab and sort so the oldest traces are at the top and the newest are at the bottom.

📊 For the oldest trace, look at the **Response Time** tab. This trace was when we executed the Lambda function the first time. Now compare that response time with the response time of the trace directly below it. The second trace should be much faster than the first. This is because of execution context re-use. Even without us declaring our objects outside of the handler, you can see there is still some performance increase just from the first invocation to the second. This is because the first time there was a cold start, and the second time there was not.

Now, review the third traces response time. Notice how it is back to being a longer response time. That is because when we updated the source code for Lambda it had to do another cold start to create a new execution context for it with the new code in place. This time the code had the objects declared outside the handler method. Compare this response time with the fourth trace. Notice how the fourth trace is the fastest.

Compare the second trace and fourth traces response times. Since neither of them have cold starts, you can objectively compare what impact taking the objects out the handler/methods had on performance. The most recent invocation should show a performance increase from the second invocation.

See the difference? It's only a minor change right but now we have our dependencies being reused over and over on for the same handler.

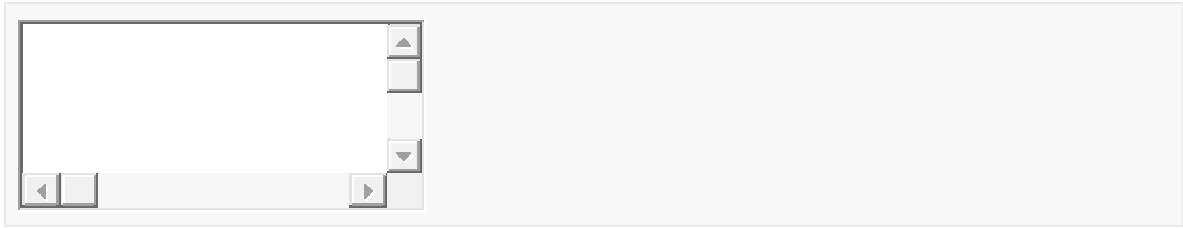
As in this example image below, you can see the more recent calls with the newer code using the "context reuse things" are faster than before.

Step 3 API Gateway cache

One more optimization to finish off.

You are going to add a cache in front of your endpoint using Amazon API Gateway Cache.

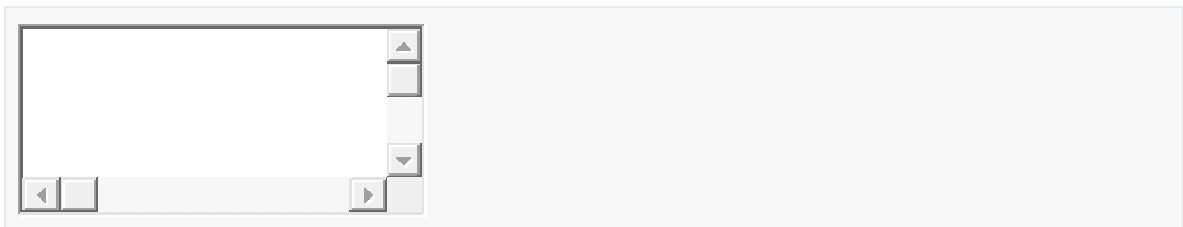
1. Choose **Services** and search for **API Gateway**. Choose **Fancy-API**.
2. Choose **Stages** and choose **test**.
3. Choose the Logs/Tracing tab and check the **Enable X-Ray Tracing** checkbox. Click **Save Changes**.
4. Choose the Settings tab. **Enable API cache**. Set it to **0.5GB**. Choose **Save Changes**.
5. Expand **test**. Choose **get_reviews/GET**.
6. Choose **Override for this method**. Choose **Enable Method Cache**. Choose **Save Changes**.
7. Navigate back to the **test** stage and you should now see the following text under the Settings tab where you enabled caching before:



Cache status:CREATE_IN_PROGRESS

It takes a few minutes to create the cache instance for API Gateway.

8. Refresh the page every few minutes until you see the Cache status on the **Settings** tab change to:



Cache status: AVAILABLE

Now we will test the `get_reviews` API from the website, and view the traces in X-ray.


9. Open a new tab and open the website.

If you don't know the endpoint for your website (this is the **Callback URL** we copied earlier) and changed to `/index.html`.

10. Search for a product review. For example, type `camera` in the search box and click the **reviews** button on a 1-2 different products. This will send a request to API Gateway, which will then trigger the `get_reviews` lambda function.
11. Press the `reviews` button again *on the same item(s) you just did in the last step*.

This should submit a second API call for that product ID to the `get_reviews` API. API Gateway should pull the data from the cache the second time.


12. Switch back to the **API Gateway** tab. Choose **Services** and **X-Ray** and choose **Traces**.
13. You should see traces that have the website URL listed under the **URL** tab. Sort the traces by age, so you can easily compare them.

 Compare the latest trace against the trace that was recorded from the first time you submitted the request from the website. The first time a request comes in for reviews on a product, the data doesn't exist in the API Gateway cache, and therefore the Lambda function is executed. The second time, the data is in the cache and therefore the Lambda function is not executed, but instead the data is just retrieved from the cache.

You should see improvements in response times from trace to trace as it used the cache.

14. To prove that the cache was used, and the Lambda function was not executed, click on the latest trace.

This will take you to a page that shows you details on this trace, including the service map.

 Notice how the service map does not include a node for AWS Lambda for the `get_reviews` function. That is because the API Gateway cache was used.

As you can see from this example the first one was in line with what we expect, a few hundred ms, then it gets into better "double digits" once the cache is kicking in. Your traces will be different.

Feel free to look at the trace data available to you here, and click around- see what cool data you can find :)

In closing, some ideas for further optimization.

We only optimized one API here.

If you were really doing this in production, you would want to apply similar concepts to the other GET APIs such as `get_ratings`.

As far as the POST API goes (`create_report`) you would not want to apply API GW caching to that, as in contrast to GETs, our POST is using token authorization.

However you could still apply the same type of context reuse strategies.

You could potentially take metrics even further and have your Lambda function provide custom sample payloads to X-Ray.

That's it! All done!

As I mentioned in my lab intro, you can go on and on with optimizing and tweeting, but as long as you are implementing metrics and attacking the biggest impact areas first, you are on the right track.

***Congratulations, you have completed the project for Mike. He, along with all of his staff are delighted with what you have built for them, in such a short period of time!**

GREAT JOB

I really hope you enjoyed all of these labs, and learned a thing or two on the way. ..and if you were doing, thanks for playing along with the story.

See you all again, in the next course.

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>