Thien Tran
Dr. Phani Vadrevu
Cloud Computing
December 1st, 2021

Analysis of Lambda Container Reuse Timeout Threshold

The container computing environment of a Lambda function, as seen with Task-3, has two different execution contexts: cold start and warm start. Cold start is when a new container is provisioned by AWS to handle an event, while warm start is when a free, existing container is used to handle the event. Anyways, the goal of Task-4 is to find the exact threshold in terms of function wait time between an initial cold start and an additional function execution that would cause another cold start as opposed to a warm start. I have determined, based on my quick experiment, that this threshold between cold start and warm start context switch occurs at approximately **5 minutes and 49 seconds** for a Lambda function with *mostly* default attributes after the initial Lambda cold start request. There were many possible ways to gain this information like stated in the hints. The next part of the paper will describe my specific method.

The first thing that I did was that I reused the Lambda function code from Task-2. The code that I created for Task-2, fortunately, had everything necessary to complete the next phases of this task without needing to change anything. Knowing that Lambda is a serverless function with no additional cost regardless of the amount of functions in play, I decided that it would be best to optimize for time, so I decided to launch 180 Lambda functions (to be safe) and invoke the functions sequentially with a minute delay in between, thus making the first invocation wait 1 minute while the last wait 180 minutes and achieving an O(N) time complexity assuming that the context switch happens within N minutes.

I was able to take advantage of Shell-script looping to create and destroy the infrastructure without too much additional work since I was also using automated infrastructure via Boto3 throughout the project. My Lambda function invocations, for both the initial cold start and the sequential invocations, in the shell scripts also appends their responses, which includes an execution context, to their shared response output text files. I then deploy fresh lambda functions and activate the initial invocations to all functions to cold-start them. The first 15 responses to the cold start are shown below:


```
 1 {"statusCode": 200, "lambda_ip_address": "169.254.149.149", "elapsed_time": 0.012567996978759766, "execution_context": "Cold Started"}
 2 {"statusCode": 200, "lambda_ip_address": "169.254.65.85", "elapsed_time": 0.0091552734375, "execution_context": "Cold Started"}
 3 {"statusCode": 200, "lambda_ip_address": "169.254.93.17", "elapsed_time": 0.007018566131591797, "execution_context": "Cold Started"}
 4 {"statusCode": 200, "lambda_ip_address": "169.254.6.241", "elapsed_time": 0.012034893035888672, "execution_context": "Cold Started"}
 5 {"statusCode": 200, "lambda_ip_address": "169.254.136.5", "elapsed_time": 0.01660013198852539, "execution_context": "Cold Started"}
 6 {"statusCode": 200, "lambda_ip_address": "169.254.59.249", "elapsed_time": 0.018029212951660156, "execution_context": "Cold Started"}
 7 {"statusCode": 200, "lambda_ip_address": "169.254.81.65", "elapsed_time": 0.003244161605834961, "execution_context": "Cold Started"}
 8 {"statusCode": 200, "lambda_ip_address": "169.254.96.245", "elapsed_time": 0.007753610610961914, "execution_context": "Cold Started"}
 9 {"statusCode": 200, "lambda_ip_address": "169.254.35.181", "elapsed_time": 0.004555702209472656, "execution_context": "Cold Started"}
10 {"statusCode": 200, "lambda_ip_address": "169.254.30.101", "elapsed_time": 0.005340099334716797, "execution_context": "Cold Started"}
11 {"statusCode": 200, "lambda_ip_address": "169.254.85.77", "elapsed_time": 0.004723072052001953, "execution_context": "Cold Started"}
12 {"statusCode": 200, "lambda_ip_address": "169.254.126.133", "elapsed_time": 0.01134943962097168, "execution_context": "Cold Started"}
13 {"statusCode": 200, "lambda_ip_address": "169.254.233.53", "elapsed_time": 0.010164022445678711, "execution_context": "Cold Started"}
14 {"statusCode": 200, "lambda_ip_address": "169.254.47.193", "elapsed_time": 0.017713308334350586, "execution_context": "Cold Started"}
15 {"statusCode": 200, "lambda_ip_address": "169.254.2.69", "elapsed_time": 0.0033817291259765625, "execution_context": "Cold Started"}
```

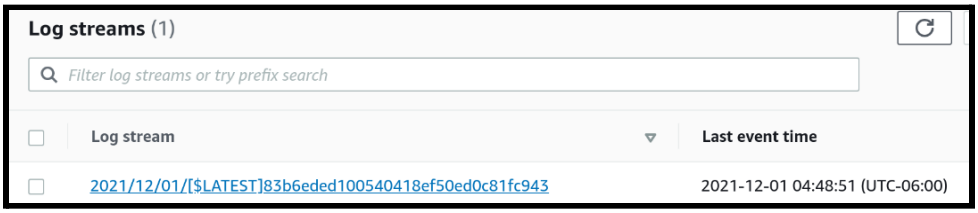
I then begin the sequential invocation sequence. If the execution switch happened somewhere between 0 and 180 minutes, then there should be a line of “Cold Started” contexts followed by a line of “Warm Started” contexts in the output. Below are the first 10 lines of the results:


```
1 {"statusCode": 200, "lambda_ip_address": "169.254.149.149", "elapsed_time": 6.604194641113281e-05, "execution_context": "Warm Started"}
2 {"statusCode": 200, "lambda_ip_address": "169.254.65.85", "elapsed_time": 6.556510925292969e-05, "execution_context": "Warm Started"}
3 {"statusCode": 200, "lambda_ip_address": "169.254.93.17", "elapsed_time": 6.604194641113281e-05, "execution_context": "Warm Started"}
4 {"statusCode": 200, "lambda_ip_address": "169.254.6.241", "elapsed_time": 6.389617919921875e-05, "execution_context": "Warm Started"}
5 {"statusCode": 200, "lambda_ip_address": "169.254.136.5", "elapsed_time": 6.985664367675781e-05, "execution_context": "Warm Started"}
6 {"statusCode": 200, "lambda_ip_address": "169.254.6.241", "elapsed_time": 0.009994268417358398, "execution_context": "Cold Started"}
7 {"statusCode": 200, "lambda_ip_address": "169.254.231.157", "elapsed_time": 0.016592979431152344, "execution_context": "Cold Started"}
8 {"statusCode": 200, "lambda_ip_address": "169.254.88.253", "elapsed_time": 0.0023729801177978516, "execution_context": "Cold Started"}
9 {"statusCode": 200, "lambda_ip_address": "169.254.161.157", "elapsed_time": 0.003760099411010742, "execution_context": "Cold Started"}
10 {"statusCode": 200, "lambda_ip_address": "169.254.111.29", "elapsed_time": 0.008557558059692383, "execution_context": "Cold Started"}
```

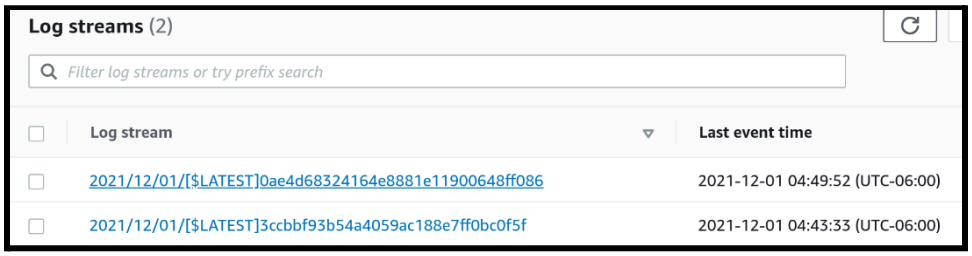

As you can see, everything worked! The first 5 invocations warm-started from a previous cold-started environment, and the last 5 cold-started! I also confirmed it first by noticing that the warm-started contexts have a run time that is significantly lower than the cold-started contexts, and that the IP Addresses in the responses for the warm-started invocations also matched the IP Addresses for the cold-started responses despite the major difference in elapsed time.

And when I went to the CloudWatch logs to get the times to calculate the threshold, sure enough function 5 had one log stream - representing 1 container execution environment - while function 6 had two log streams - showing that it cold-started twice.

Invocation 5



Invocation 6



I then went into the log stream for Invocation 5 and took the true elapsed time from the difference between the time of the very first request to the time of the end of the final request. I also went into the log streams for Invocation 6 and took the elapsed time from the difference between the time of the very first initial request in the earlier log stream and the time of the end of the final request in the later log stream. The context switch happened somewhere in between the elapsed time of the first request (*5 minutes and 19 seconds*) and the elapsed time of the

second request (*6 minutes and 19 seconds*), so I took the average of both of them to be my answer for the approximate threshold time: **5 minutes and 49 seconds**. Cheers!