THE UNIVERSITY of NEW ORLEANS

CSCI 4452/5452 CLOUD COMPUTING

Fall 2022

Assigned: Oct 29

Credits: 20 course points

Due: NOV 30 @ 11:59PM

Assignment-2

In some demo classes, we will see that some under-the-hood decisions that AWS Lambda service makes can potentially have an impact on our data processing setup (especially, if we are relying on global variables that persist across invocations).

In this assignment, our goal is to try and infer how AWS is making these decisions regarding spawning of new Execution Contexts for the Lambda service.

Task-1

Your first task is to create a simple AWS Lambda function that echoes a string argument it receives during invocation. During invocation, the function should also return the IP address of the server on which the Lambda service is running. Your Lambda function should also print the received string argument and the IP address (this printed log will of course be available to view on AWS Cloud Logs).

Deliverables: You need to include the following in your submission report.

(1) The AWS Lambda function you created.
(2) The AWS CLI command or AWS SDK code that you used to do the function invocation.
(3) Two screenshots showing the function invocation and its result both on your client machine as well as on the AWS Cloud Logs.

Task-2

The second task is to find out the time it takes for your Task-1 code to run during "cold starts" / "warm starts". A "cold start" is when AWS Lambda is forced to spin up a new Execution Context in order to process a requested function invocation. A "warm start" is when AWS Lambda makes use of an already pre-existing Execution context for a requested invocation. There will be a measurable timing difference between the two modes. Your goal is to measure the response time of an AWS Lambda function after at least 10 cold starts and at least 10 warm starts and then compare the two categories of times.

Hints:

(1) For a cold start, you can make slight changes to your Task-1 code, re-deploy it and then do the first function invocation with the help of some simple time measurement set-up on your client machine. Make sure that you do not use the "Test" feature provided on AWS GUI as this will end up invoking your function and might lead to a warm start instead of a cold start". You can continue to make such slight changes for more cold starts.
(2) For a warm start, you can first make a cold start as above and then make a series of invocations to the same cold-started function and measure the response times.

Deliverables:

(1) Describe the process you used for setting up cold and warm starts. Please include any code / commands you used for this.
(2) Show the raw results of response time for all the cold starts and warm starts and their mean and median values.
(3) Draw a box plot graph depicting the cold and warm start response times. You may use Excel / Matplotlib or any other suitable tool for this. Please submit additional code (if any) for this.

Task-3

Task-2 involved using synchronous function invocations to distinguish between cold and warm starts of Lambda functions. In this task, you will only use asynchronous invocations.

You can make a series of N back-to-back asynchronous invocation requests (N = 10) programmatically to a newly deployed function. Now, AWS Lambda has three choices:

(1) Create N Execution contexts for each of the N requests.
(2) Create more than 1 but less than N Execution contexts for processing each of the requests.
(3) Create only 1 Execution context for processing all the N requests serially.

Your task is to find out the relationship between "function execution" duration and the three choices above that AWS Lambda makes. For example, if the "function execution" duration is very short, we can suspect that AWS might prefer to do (3) instead of going for the more expensive option of (1).

Hints:

(1) You can easily vary the execution time of a function by using a sleep() function that most programming languages support.
(2) Note that the default time out value (of only 3 seconds) might interfere with the experiments in this task and you might need to change it.
(3) There is no need to receive the response from the Lambda function on the client for this task. You can print the IP addresses in AWS Cloud Logs (as in Task-1) and infer the execution context used by AWS Lambda using the IP address.
(4) You can make the string argument being passed for each invocation be incremented serially. If you make this string argument printed in the Cloud Logs (as in Task-1), this will help you map each printed output log to the request you made.

Deliverables:

(1) A report describing the results of your experiments. The report should state clearly what "function execution duration" ranges lead to each of the 3 choices made by AWS. The units for "function execution duration" need not be more granular than 1 second.
(2) For clarity, you should also draw a graph that visually shows the relationship between number of execution contexts, say, on Y-axis and the "function execution duration" on X-axis.
(3) Your report should also include all the IP addresses used by AWS for deployment during each of the invocations in your reports.

Task-4

The final task is to attempt to infer how long AWS Lambda keeps a cold-started execution context around to use it for a future warm-start. Let's call this "Lambda Context Preservation" (LCP) time. Task-2 would have given you an ability to clearly distinguish between cold and warm starts based solely on response time. You will now re-use your setup of Task-2 to do this task.

Hints:

(1) You can use a binary-search like technique to zero in on the LCP time being used by AWS. You can do a cold start of a new function and then wait for some time, say, 2 hours. After this time, you will attempt to do another function invocation and infer whether this is a warm start. If it is not, it implies that the LCP time is less than 2 hours. You can then use a wait time of 1 hour for another new function and repeat this process.
(2) You might want to script the above process after the times get a little smaller in order to ease the process. It might be easier to do this by pre-deploying X amount of different Lambda functions and then using them iteratively in the binary search process. However, this is only a suggestion, and you are free to do this entire binary search process manually too.

Note: You do not need to have a granularity of more than 1 minute for the final answer for this task.

Deliverables:

(1) A report describing the results of your experiments. The report should clearly state all the wait times that led to cold starts, and all wait times that led to a warm start throughout your experimentation. If needed, a simple binary tree diagram might help easily visualize these results, but this is completely optional and carries no additional points.
(2) If any additional scripts have been written for conducting the experiments, please include them in the submission.

5452 / Graduate Student Requirement

5452/Graduate students are required to complete Tasks 1-4 **twice** using two AWS supported Lambda function languages (Java, Go, PowerShell, Node.js, C#, Python, and Ruby). Note that this means you should submit **two sets of deliverables and conduct all experiments twice.** Any client-side SDK-based code that is used need not be changed between the two iterations. So, it is OK to use the same boto3-based client-side code to invoke a Java Lambda function as well as a Python Lambda function, for example.

Implementation & Submission

- You are free to choose any language(s) that you feel comfortable with.
- **Submission**: Create a **private repository** named `csci4452-f22` on `gitlab.cs.uno.edu` (use the UNO Active Directory Login). Please add me as a collaborator with at least **Developer** access. My e-mail address for gitlab is **kkvadrev@uno.edu**
    - Make sure that the repository is named **exactly** as specified (this includes 5452 students)
    - Create a folder in your repo called `assignment-2` and place all your deliverables for the 4 tasks there.
- **The submission deadline is a hard one.**
- This is an **individual** take-home assignment. You may passively consult any available sources on the Internet—that is, you can read anything but you cannot solicit advice in any form. **All work must by yours only.**