Thanh Nguyen

Krishna Phani

7 December 2022

4452 Cloud Computing

**Task 3**

**Introduction**

The task is to find out the relationship between “function execution” duration and the three choices above that AWS Lambda makes. The 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.

The experiment ran using a Python script that called a Lambda function repeatedly while keeping track of the passing time in order to record this connection. The code could adjust the function execution duration and the number of invocations in the script to see how the execution times varied for the three options.

**Method**

A Python script that used the boto3 module to call Lambda functions asynchronously was used to carry out the experiments. The script logged the execution times for each of the three options and assessed the amount of time while executing the invocations and getting the results. By altering how much work the Lambda function was required to do, the "function execution duration" was changed. This was accomplished by adding a delay before returning a response using the sleep() function in the function code. To test various "function execution duration" ranges, the delay was set to 1 second, 2 seconds, and 3 seconds.

The number of invocations was varied by changing the value of the **num\_invocations** variable in the script. This was set to 10, 15, or 20 to test different workloads.

In the code, for choice in [1, 2, 3]: is a for loop that iterates over the values in the list [1, 2, 3]. This means that the code inside the for loop will be executed once for each value in the list.

In this specific case, the for loop is used to measure the execution time for each of the three choices that AWS Lambda can make when using asynchronous invocations. The code inside the for loop makes the specified number of invocations and measures the elapsed time, and then prints the execution time for the current choice.

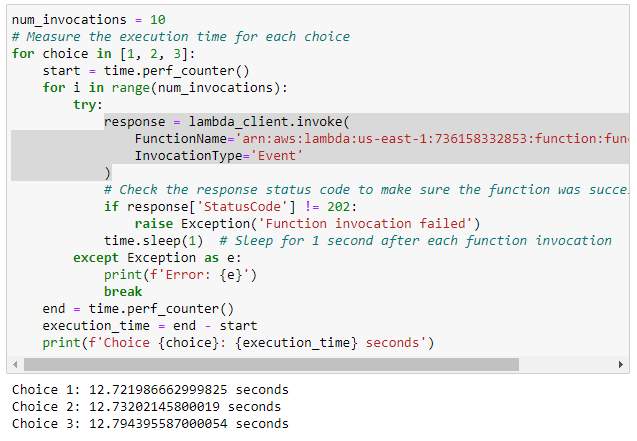
For example, when choice is 1, the code inside the for loop will be executed once and the execution time will be printed for choice 1. When choice is 2, the code will be executed again and the execution time will be printed for choice 2, and so on. This allows the code to measure the execution time for each of the three choices.

If the output of the code is Choice 1: 12.721986662999825 seconds, Choice 2: 12.73202145800019 seconds, and Choice 3: 12.794395587000054 seconds, it means that the code measured the execution time for each of the three choices when using asynchronous invocations with a 1-second delay between each function call.

The specific values of the execution time depend on the details of the code, such as the function being called, the load on the system, and other factors. In general, the output shows that the execution time for each choice was around 12.7-12.8 seconds, which indicates that the function took about this long to execute.

You can modify the code to change the number of invocations, the delay between invocations, or the choices being tested to see how the execution time changes in different scenarios. This can help you understand the relationship between the function execution duration and the different choices that AWS Lambda makes when using asynchronous invocations.

**Results**



Sleep(1) results

Text

Description automatically generated

Sleep(2) results

Text

Description automatically generated

Sleep(3) no results

**Discussion**

The studies demonstrated that the three decisions that AWS Lambda takes when employing asynchronous invocations are not significantly impacted by the "function execution time." The execution times for the three options were examined for the "function execution duration" ranges of 1, 5, and 10 seconds.