School of Engineering and Technology
University of Washington – Tacoma
Instructor: Wes Lloyd

Extra Credit – Serverless Function-as-a-Service (FaaS) Stress Microservice

Version 0.10

Due Date: Friday March 22nd, 2019 @ 11:59 pm, tentative

Objective

The purpose of the extra credit assignment is to develop an original AWS Lambda function in Java using the FaaS Inspector framework. The function should run for at least 10 to 30 seconds, and should create resource stress. The function should create either: CPU stress, memory stress, disk I/O stress, or network I/O stress. CPU stress, for example, can be generated by implementing a complex algorithm such as Fibonacci. Memory stress can be created by writing code that creates and operates on large dynamically sized arrays. Disk I/O stress can be created by reading and/or writing files under the local "/tmp" file system. Network I/O stress can be created by downloading or uploading data from a remote server. The function should be deployed to AWS Lambda, and a public http REST end point should be configured and made available using the AWS API Gateway. The "callservice.sh" and "partestcpu.sh" scripts should be updated to call your new function.

For instructions on developing the AWS Lambda function with the FaaS inspector framework, please review TCSS 562 Tutorial #4 at:
http://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS562_f2018_tutorial_4.pdf

Please complete the tutorial as needed to complete the assignment. The tutorial will not be graded, but the new Lambda function that creates either CPU, memory, disk, or network stress **will be**. Your specific task is to create a Lambda function that will force the creation and use of no fewer than 100 run time containers on AWS Lambda when running the partestcpu.sh script on your laptop or EC2 VM client. Clients should have no more than 4-6 CPU cores.

The number of run time containers, appears as the last line of the partestcpu.sh script. The example below shows that not enough stress has been created. The number of containers must be 100. To ensure the use of 100 runtime containers, the function duration should probably be more than 10 seconds.

```
$ ./partestcpu.sh 100 100

.
.
.

containers,newcontainers,recycont,hosts,recyvms,avgruntime,avgssruntime,avglatency,runs_per_container,runs_per_cont_stdev,runs_per_host,runs_per_host_stdev,totalcost
83,83,0,78,0,6926,1456,5470,1.205,0.211,1.282,0.331,$0.0073
```

For the stress function, choose one resource: CPU, memory, disk, or network

Develop original AWS Lambda service in Java using the FaaS Inspector framework with performance bound by CPU, memory, disk, or network

Stress code can be found online, and integrated and deployed as a Lambda function, or original stress code can be written from scratch.

Functions that simply sleep (perform no operations, and are idle), to obtain 100 containers will not receive credit. The goal is to write or find code to develop a Lambda function that performs original and/or creative operations that actually exercise Lambda's available resources.

The function, should not have external dependencies, or require significant effort to deploy.

What to Submit

To submit the assignment, capture the output of **`partestcpu.sh 100 100`** and submit a CSV or Excel XLSX file to Canvas. Additionally, submit your complete project source tree.

From the "faas_inspector" directory, create a tar gzip file:

```
~/git/faas_inspector$ tar czf disk-stress-function.tar.gz *
```

This source tree should include updated versions of callservice.sh and partestcpu.sh to work you're your function under the faas_inspector/lambda/java_template/test directory.

Grading Rubric

This assignment will be scored out of 20 points.
Maximum credit will be granted for solutions that do a good job with at least 3 of the 4:

#1- generate stress against 100 runtime containers in one call using a standard 2-4 CPU client
#2- incorporate originality in terms of the stress test
#3- include documentation that describes what the stress function is, and what it does
#4- exposes request JSON parameters to support configuration the behavior of the stress function.

Document History:
v.10 Initial version