**Lab 10: Scaling Selenium Tests using AWS Lambda and Step Functions**

Objective:*Run Selenium tests as serverless functions with scalability and orchestration.*

Tasks:

1. Configure a Lambda function to run a Selenium test.

2. Orchestrate multiple tests using AWS Step Functions.

Documentation:

- Introduction to AWS Lambda.

- Basics of AWS Step Functions.

- Scalable and orchestrated Selenium tests on AWS.

Prerequisites:

1- An AWS account with administrative access.

2- Python Automation Course

3- Python Selenium Course

4- Bash Script Deep Dive Course

5- Previous Lab completed

Implementation Documentation:

**Introduction to AWS Lambda**

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the compute resources for you. It allows you to run code without provisioning or managing servers.

**Step 1: Setting Up a Lambda Function for Selenium Tests**

* Log in to your AWS account.
* Go to the Lambda dashboard.
* Create a new Lambda function.
* Configure the function with a trigger (e.g., API Gateway, CloudWatch Events).
* Write your Selenium test script as a Lambda function.
* Ensure to include the necessary dependencies (e.g., Selenium, web driver) in a deployment package.

create two layers of selenium and chromedriver use python 3.7 for them

now create a function main and deploy code in this function

**Step 2: Creating Layers in Lambda**

create new folder name it as lambda selenium

Open terminal in this folder

| pip3.7 install selenium==3.8.0 -t python/lib/python3.7/site-packages pip3.7 install -t selenium/python/lib/python3.7/site-packages selenium==3.8.0 |
| --- |

Run these commands make sure to have python3.7 installed.

#now zip this folder and name it e.g., selenium

**Uploading Selenium to Lambda Layer**

now goto lambda --> layers

create layer

selenium

upload your zip here

x86\_64

Runtimes Python3.7

create layer

Now open terminal for creating zip of Chrome browser and Headless Chromium

| curl -SL https://chromedriver.storage.googleapis.com/2.37/chromedriver\_linux64.zip > chromedriver.zip  curl -SL https://github.com/adieuadieu/serverless-chrome/releases/download/v1.0.0-41/stable-headless-chromium-amazonlinux-2017-03.zip > headless-chromium.zip |
| --- |

zip both these together after unzipping them and rename to chromedriver

**Uploading chrome drivers to Lambda Layer**

now goto lambda --> layers

create layer

chromedriver

upload the zip here

x86\_64

Runtimes Python3.7

create layer

**Creating Lambda Function**

goto lambda, function , create new function

selenium

python3.7

x86\_64

create function

**Configuring Lambda Function memory for smooth sailing**

goto configuration

general configuration

memory to 1024

timeout 1 minutes

save

now click code

scroll down and click edit in Runtime settings

handler put filename.main #in our case it will be like e.g., main.main

save

**Adding layers we created to Lambda Function**

now click add layer

custom layer

selenium

version 1

add

again click add layer

custom layer

chromedriver

version 1

add

**Starting Test in Lambda Function**

now click test

We forgot to create zip of main.py file, goto your machine and open pycharm create a new python file main.py place following script in the file.

| from selenium import webdriver from selenium.webdriver.chrome.options import Options  def main(event, context):  options = Options()  options.binary\_location = '/opt/headless-chromium'  options.add\_argument('--headless')  options.add\_argument('--no-sandbox')  options.add\_argument('--single-process')  options.add\_argument('--disable-dev-shm-usage')   driver = webdriver.Chrome('/opt/chromedriver',chrome\_options=options)    driver.get('https://www.google.com/')  title = driver.title   driver.close();  driver.quit();   response = {  "statusCode": 200,  "body": title  }   return response |
| --- |

# This is a simple selenium script to open chrome driver and get the title of the website it opens, then return status code 200 and the title it got from the website.

**Zipping the file and uploading it**

Now zip this file from the directory where it is placed

Now again goto browser where we left

click code then upload from .zip file

now zip main.py and upload

Now click the test button from the lambda function and check the results.

**Explanation:**

* We define a state machine with two tasks, each representing a Selenium test Lambda function.
* RunTest1 and RunTest2 are the task names.
* The Resource field specifies the ARN of the Lambda function associated with each task.
* End: true signifies that the state is terminal.

Scalable and Orchestrated Selenium Tests on AWS

**Additional Considerations:**

* Ensure that your Lambda functions are configured with the necessary permissions to interact with AWS services (e.g., S3, CloudWatch, SNS).
* Monitor the execution of your state machine and Lambda functions using AWS CloudWatch logs and metrics.

**Conclusion:**

By following these steps, you have successfully set up scalable and orchestrated Selenium tests on AWS using Lambda and Step Functions. This lab demonstrates how to leverage serverless computing and orchestration for automated testing workflows.