National College of Ireland

Lambda Authorizer Benchmarking Tool Configuration Manual

MSc Research Project
Cloud Computing

Cornelius
Student ID: 21126747

School of Computing
National College of Ireland

Supervisor: Dr. Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Cornelius
Student ID:	21126747
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Shivani Jaswal
Submission Due Date:	01/02/2023
Project Title:	Lambda Authorizer Benchmarking Tool Configuration Manual
Word Count:	1334
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	26th January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Lambda Authorizer Benchmarking Tool Configuration Manual

Cornelius
21126747

Table of Content

1 Prerequisites

The purpose of this application is to calculate the performance of Lambda Authorizer-enabled serverless functions. It was created as part of the MSc in Cloud Computing Research Project at the National College of Ireland. First of all, NodeJS and NPM are required for this project, and their installation of them is straightforward. The next step is to install the AWS CLI and AWS SAM CLI. Make sure the AWS CLI profile is configured with an active AWS account. Furthermore, in order to perform benchmarking process, the user must install programming language runtimes: Python 3.9, Go 1.x and Java 11. In addition to Java, the user needs to install Maven as well. Ensure that all the prerequisites are correctly installed by running the commands as shown in Figure 1.

```
$ npm -v && node -v
8.19.2
v18.11.0

$ aws --version && sam --version
aws-cli/2.8.5
SAM CLI, version 1.60.0

$ python --version && go version
Python 3.9.14
go1.19.2

$ java -version && mvn -version
openjdk64-11.0.11
Apache Maven 3.8.6
```

```
tyranade@Corneliuss-MacBook-Pro ~ % npm -v && node -v
8.19.2
v18.11.0
tyranade@Corneliuss-MacBook-Pro ~ % aws --version && sam --version
aws-cli/2.8.5 Python/3.10.8 Darwin/22.1.0 source/arm64 prompt/off
SAM CLI, version 1.60.0
tyranade@Corneliuss-MacBook-Pro ~ % aws configure list
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key     ****************63NV shared-credentials-file
secret_key     ****************93NY shared-credentials-file
    region                eu-west-1      config-file    ~/.aws/config
tyranade@Corneliuss-MacBook-Pro ~ % python --version && go version
Python 3.9.14
go version go1.19.2 darwin/arm64
tyranade@Corneliuss-MacBook-Pro ~ % java -version && mvn -version
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment AdoptOpenJDK-11.0.11+9 (build 11.0.11+9)
OpenJDK 64-Bit Server VM AdoptOpenJDK-11.0.11+9 (build 11.0.11+9, mixed mode)
Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
Maven home: /opt/homebrew/Cellar/maven/3.8.6/libexec
Java version: 11.0.11, vendor: AdoptOpenJDK, runtime: /Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home
Default locale: en_ID, platform encoding: UTF-8
OS name: "mac os x", version: "10.16", arch: "x86_64", family: "mac"
```

Figure 1: Results of the Version Checkup

2 Development Environment

The author's machine and system to develop and run the Lambda Authorizer Benchmarking Tool are described in Figure 2.

```
$ system_profiler SPSoftwareDataType
```

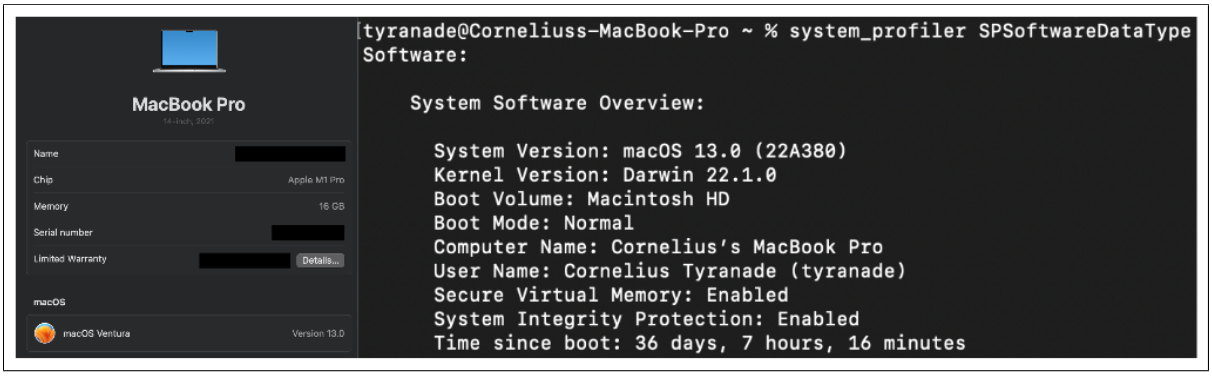

Figure 2: Author's Machine and Operating System

3 Installation

Please read the prerequisites before installation.

The first step is to clone the repository locally and enter the downloaded folder as per the screenshot in Figure 3.

```
$ git clone https://github.com/cornelius-tyranade/lambda-authorizer-
    ↪ benchmarking-tool.git
$ cd lambda-authorizer-benchmarking-tool
```

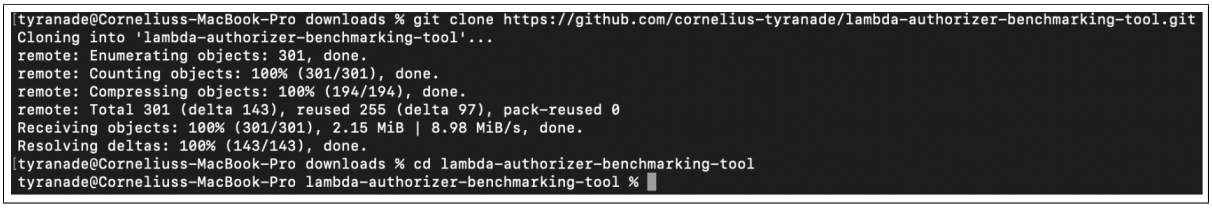

Figure 3: Lambda Authorizer Benchmarking Tool Repository Cloning Result

3.1 Dependencies

The *package.json* file contains a list of dependencies that need to be installed by running the command below:

```
$ npm install
```

Below are a short explanation of used dependencies and the source of each dependency:

1. Artillery. Perform multiple serverless performance tests quickly.
2. Command-exists-promise. Verify if a specific command exists in the system.
3. Commander. Quick solution for NodeJs command-line interfaces development.
4. Config. Simple key-values file configuration manager.
5. Replace-in-file. Replace marked text synchronously in one or more files.
6. ShellJS. Call shell commands in NodeJs application.

Figure 4 illustrates how NodeJS application dependencies are installed and set up:

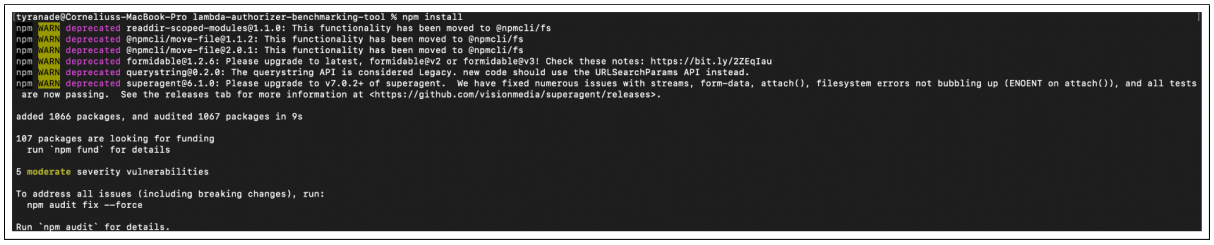

Figure 4: NPM Dependencies Installation Result

3.2 AWS SAM Configurations

This application uses AWS SAM as serverless application builder, so the user need to run command below to configure *samconfig.toml*. However, **DO NOT** type **"Y"** when the terminal asks to start deployment. Leave the input blank, and the capital letter value will be used by default as illustrate in Figure 5.

```
$ sam deploy --guided
```

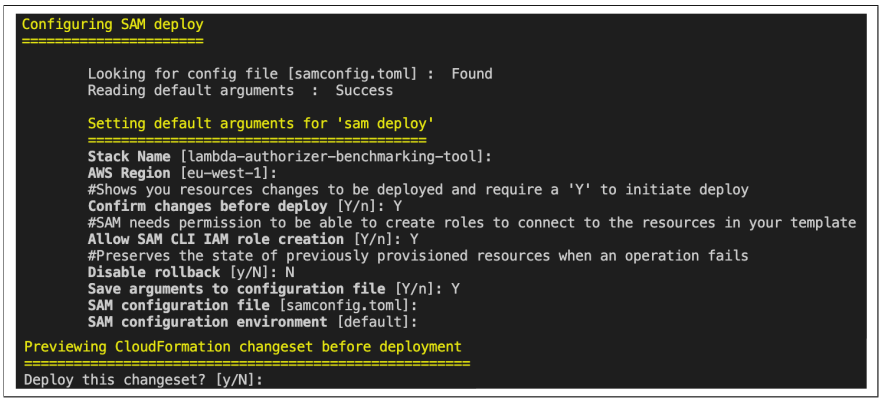

Figure 5: AWS SAM Quick Setup

Alternatively, the user can directly modify existing *samconfig.toml* as per Figure 6.

```
version = 0.1
[default]
[default.deploy]
[default.deploy.parameters]
stack_name = "lambda-authorizer-benchmarking-tool"
s3_bucket = "aws-sam-cli-managed-default-samclisourcebucket-xxxxx"
s3_prefix = "lambda-authorizer-benchmarking-tool"
region = "eu-west-1"
confirm_changeset = true
capabilities = "CAPABILITY_IAM"
image_repositories = []
```

Figure 6: Existing AWS SAM Configurations

4 Usage

In this part, the author explains every single option that can be used in the Lambda Authorizer Benchmarking Tools application. A total of six main options are available in this application, including Option Help, Option Clean, Option Deploy, Option Test, Option Report, and Option Logs Insight. In addition, the user can combine more than one option when running the application.

4.1 Option Help

This option explains what feature options are available in the Lambda Authorizer Benchmarking Tool. As seen in Figure 7, this option has the following effect.

```
$ node labt -h
$ node labt --help
```

```
Usage: labt [options]

==================================
Lambda Authorizer Benchmarking Tool
==================================

Options:
  -v, --version                        display version
  -c, --clean                          remove current stack if exist
  -d, --deploy                         start the deployment process
  -t, --test [identifers...]           start the performance test
  -r, --report [identifers...]         generate Artillery performance test report in HTML
  -li, --logs-insight [identifers...]  generate AWS CloudWatch logs insight query result in JSON
  -h, --help                           display help for command
```

Figure 7: Option Help Result

4.2 Option Version

Lambda Authorizer Benchmarking Tool version can be found by selecting this option.

```
$ node labt -v
$ node labt --version
```

4.3 Option Clean

This option removes the currently installed AWS CloudFormation Lambda application stack. It also logs the removing process into *serverless-apps-builder/logs* folder as *stage_delete.txt*.

```
$ node labt -c
$ node labt --clean
```

4.4 Option Deploy

This option deploys all scenarios according to what is written in the *template.yaml* after building the scenarios code.

```
$ node labt -d
$ node labt --deploy
```

Upon completion, each scenario's URLs and identifiers are extracted into *urls.json* and *identifiers.json*, respectively.

```
// urls.json
[
  "https://<url>/v1/req-auth-go?QueryString1=queryValue1",
  "https://<url>/v1/tkn-auth-go -H \"AuthorizationToken: <bearer>\"",
  "https://<url>/v1/req-auth-python?QueryString1=queryValue1",
  "https://<url>/v1/req-auth-node?QueryString1=queryValue1",
  "https://<url>/v1/tkn-auth-node -H \"AuthorizationToken: <bearer>\"",
  "https://<url>/v1/req-auth-java?QueryString1=queryValue1",
  "https://<url>/v1/tkn-auth-python -H \"AuthorizationToken: <bearer>\"",
  "https://<url>/v1/tkn-auth-java -H \"AuthorizationToken: <bearer>\""
]

// identifiers.json
[
  "requestAuthorizerGo",
  "tokenAuthorizerGo",
  "requestAuthorizerPython",
  "requestAuthorizerNode",
  "tokenAuthorizerNode",
  "requestAuthorizerJava",
  "tokenAuthorizerPython",
  "tokenAuthorizerJava"
]
```

Also, *stage_build.txt* and *stage_deploy.txt* log files are generated during this process. It shown in Figure 8.

Figure 8: Option Deploy Result

4.5 Option Test

This option instructs the system to run performance testing repeatedly for a certain duration. The duration and rate of performance testing can be set from within the *default.json* file in the *config* folder. As an example, the duration and rate values are ten. The system will call the function ten times a second for ten seconds.

```
$ node labt -t
$ node labt --test
$ node labt -t "tokenAuthorizerPython"
$ node labt -t "requestAuthorizerPython" "tokenAuthorizerPython"
```

It also generated Artillery configuration in *config/artillery* folder. One of the examples of test results is *requestAuthorizerGo.yml*

```
config:
  target: https://xxxxx.execute-api.eu-west-1.amazonaws.com/v1/
  phases:
    - duration: 10
      arrivalRate: 10
      name: benchmarking

scenarios:
  - name: requestAuthorizerGo
    flow:
      - get:
          url: /req-auth-go?QueryString1=queryValue1
          expect:
            - statusCode: 200
```

The output of this process is placed in the *outputs/artillery* folder in JSON form. For instance, *requestAuthorizerGo.json* contents below.

```
{
  "aggregate": {
    ...
    "firstCounterAt": 1669747575150,
    "firstHistogramAt": 1669747575742,
    "lastCounterAt": 1669747585079,
    "lastHistogramAt": 1669747585079,
    "firstMetricAt": 1669747575150,
    "lastMetricAt": 1669747585079,
    "period": 1669747580000,
    "summaries": {
      "http.response_time": {
        "min": 55,
        "max": 539,
        "count": 100,
        "p50": 68.7,
        "median": 68.7,
        "p75": 85.6,
        "p90": 94.6,
        "p95": 108.9,
        "p99": 497.8,
        "p999": 497.8
      },
    },
  },
  ...
}
```

4.6 Option Report

Using this option, it will generate an HTML report using the JSON file in *outputs/artillery*. Report HTML generation result can be found in section Artillery reports.

```
$ node labt -r
$ node labt --report
$ node labt -r "tokenAuthorizerNode"
$ node labt -r "requestAuthorizerNode" "tokenAuthorizerNode"
```

4.7 Option Logs Insight

The AWS CloudWatch logs record each function' activities when the user run performance tests. This option queries several important output parameters from logs. The results of this output are placed in the *outputs/logs_insight* folder.

```
$ node labt -li
$ node labt --logs-insight
$ node labt -li "tokenAuthorizerGo"
$ node labt -li "tokenAuthorizerGo" "tokenAuthorizerJava"
```

Two files are generated as a result. Below is a sample of the Logs Insight query result:

```
// query_id_overview.json
{
  "queryId": "ad83de51-9e1a-4dbd-8baa-742527399491"
}

// query_result_overview.json
{
  "results": [
    [
      {
        "field": "functionName",
        "value": "requestAuthorizerGo"
      },
      {
        "field": "memorySize",
        "value": "128"
      },
      {
        "field": "coldStarts",
        "value": "3"
      },
      ...
    ],
    ...
  ],
  ...
}
```

4.8 Combination of Options

This application can accept flag combinations of more than one input. With the following command, the user can perform a clean deployment, then run the test and create an HTML report, as well as generate the results of a Logs Insight query at once.

```
$ node labt -c -d -t -r -li
```

5 Configurations

To simplify setting application variables, the author separated the configuration settings into a file called *default.json* within the *config* directory.

5.1 Artillery

Below is the configuration used in the application when running the Test options:

1. Variable *artillery.duration* determines how long the performance test runs for each scenario.
2. Variable artillery.rate specifies how many API calls are in one second.

```
{
  "artillery": {
    "duration": 10,
    "rate": 10
  }
}
```

5.1.1 Templates

When the user runs the option test, the application will generate Artillery configurations in the *folder/artillery* based on the two templates provided, *artillery_request.yml* and *artillery_token.yml*.

```
#artillery_request.yml
config:
  target: ${endpoint}
  phases:
    - duration: ${duration}
      arrivalRate: ${rate}
      name: benchmarking

scenarios:
  - name: ${identifier}
    flow:
      - get:
          url: ${postfix-url}?QueryString1=queryValue1
          expect:
            - statusCode: 200
```

```yaml
#artillery_token.yml
config:
  target: ${endpoint}
  phases:
    - duration: ${duration}
      arrivalRate: ${rate}
      name: benchmarking

scenarios:
  - name: ${identifier}
    flow:
      - get:
          url: ${postfix-url}
          headers:
            AuthorizationToken: "Bearer allow"
          expect:
            - statusCode: 200
```

5.2 Logs Insight

Meanwhile, this is the configuration used in the application when running the Logs Insight option:

1. Variable *logsInsight.timeRange* decides how many minutes before the current time are in order to fetch the logs.
2. Variable l*ogsInsight.waitTimeQuery* sets the waiting time (in seconds) before getting actual Logs Insight query results.

```json
{
  "logsInsight": {
    "timeRange": 1440,
    "waitTimeQuery": 15
  }
}
```

6 Reports

The Lambda Authorization Benchmarking Tool has two different output group reports. The mechanism for generating them is also different. Artillery JSON and HTML reports use the Artillery framework to calculate them. Meanwhile, the Logs Insight JSON report is obtained by querying directly into the AWS CloudWatch logging system. This section displays the report output from the options Test and Logs Insight after 100 calls to each scenario function.

6.1 Artillery

Artillery HTML generation result sample can be found in *outputs/artillery*. One of the most important outputs in this HTML is response time. This result is basically the sum of the performance calculation of calling a Lambda Authorizer-enabled serverless function. Figure 9 illustrates the result from *requestAuthorizerGo.json.html*.

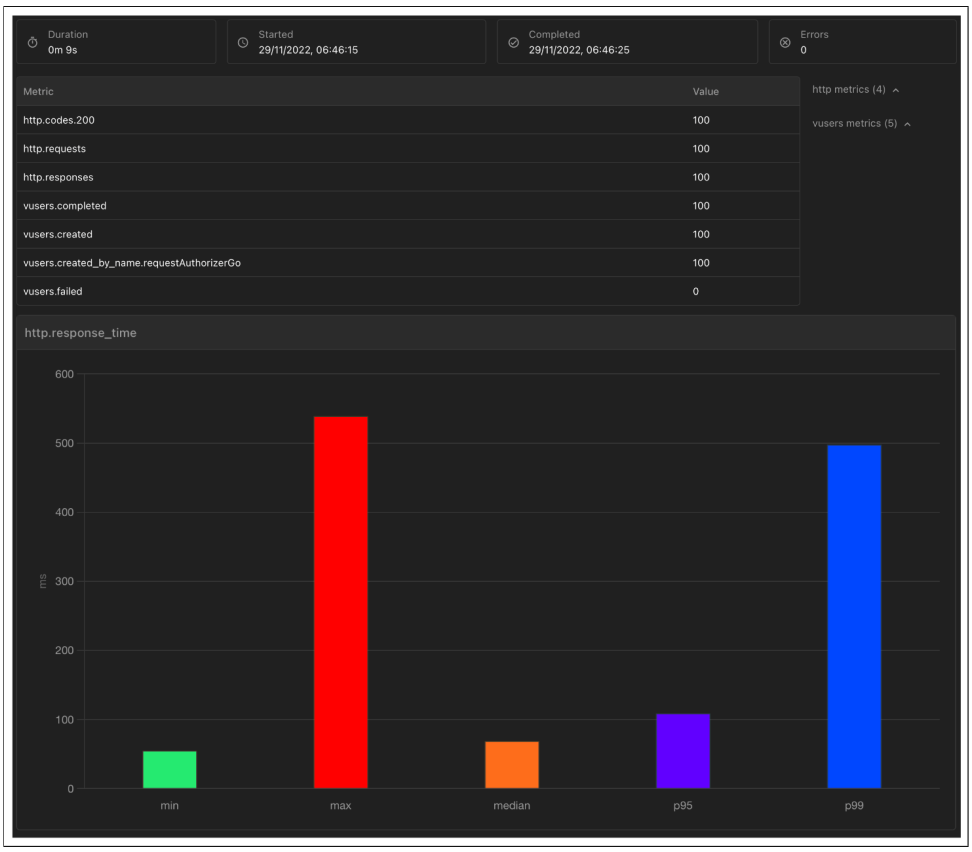

Figure 9: Artillery HTML Report Result

6.2 Logs Insight

There are seven query results generated by Logs Insight queries in JSON format. The output results can be seen in the *outputs/logs_insight* folder. The author converts the JSON results into tables as a more friendly way to view them.

1. Request-Token Access Control Overview Query (Figure 10).

#	functionName	coldStar…	minInitDurat…	maxInitDurat…	minDurati…	maxDurati…	minMemoryReque…	maxMemoryUse…	overProvisionedMemoryMB
1	requestAuthorizerGo	3	82.96	255.78	7.48	8.27	27.6566	28.6102	93.4601
2	requestAuthorizerJava	83	466.67	739.91	244.3	358.61	82.9697	83.9233	38.147
3	requestAuthorizerNode	3	157.98	223.68	10.43	14.14	54.3594	55.3131	66.7572
4	requestAuthorizerPython	3	110.04	131.03	1.6	1.85	34.3323	34.3323	87.738
5	tokenAuthorizerGo	3	77.4	85.51	1.96	17.31	27.6566	27.6566	94.4138
6	tokenAuthorizerJava	64	376.39	584.48	395.04	577.48	77.2476	78.2013	43.869
7	tokenAuthorizerNode	3	141.87	165.6	4.65	19.99	54.3594	54.3594	67.7109
8	tokenAuthorizerPython	5	102.28	116.82	1.27	1.82	34.3323	34.3323	87.738

Figure 10: Overview Request-Token Query Result

2. Request Access Controlled Max Init Duration Query (Figure 11).

#	functionName	coldStar…	minInitDurat…	maxInitDuration
1	requestAuthorizerPython	3	110.04	131.03
2	requestAuthorizerNode	3	157.98	223.68
3	requestAuthorizerGo	3	82.96	255.78
4	requestAuthorizerJava	83	466.67	739.91

Figure 11: Request Max Init Duration Query Result

3. Request Access Controlled Max Duration Query (Figure 12).

#	functionName	minDurati…	maxDuration
1	requestAuthorizerPython	1.6	1.85
2	requestAuthorizerGo	7.48	8.27
3	requestAuthorizerNode	10.43	14.14
4	requestAuthorizerJava	244.3	358.61

Figure 12: Request Max Duration Query Result

4. Request Access Controlled Max Used Memory Query (Figure 13).

#	functionName	minMemoryReque…	maxMemoryUse…	overProvisionedMemoryMB
1	requestAuthorizerGo	27.6566	28.6102	93.4601
2	requestAuthorizerPython	34.3323	34.3323	87.738
3	requestAuthorizerNode	54.3594	55.3131	66.7572
4	requestAuthorizerJava	82.9697	83.9233	38.147

Figure 13: Request Max Used Memory Query Result

5. Token Access Controlled Max Init Duration Query (Figure 14).

#	functionName	coldStar…	minInitDurat…	maxInitDuration
1	tokenAuthorizerGo	3	77.4	85.51
2	tokenAuthorizerPython	5	102.28	116.82
3	tokenAuthorizerNode	3	141.87	165.6
4	tokenAuthorizerJava	64	376.39	584.48

Figure 14: Token Max Init Duration Query Result

6. Token Access Controlled Max Duration Query (Figure 15).

#	functionName	minDurati…	maxDuration
1	tokenAuthorizerPython	1.27	1.82
2	tokenAuthorizerGo	1.96	17.31
3	tokenAuthorizerNode	4.65	19.99
4	tokenAuthorizerJava	395.04	577.48

Figure 15: Token Max Duration Query Result

7. Token Access Controlled Max Used Memory Query (Figure 16).

#	functionName	minMemoryReque…	maxMemoryUse…	overProvisionedMemoryMB
1	tokenAuthorizerGo	27.6566	27.6566	94.4138
2	tokenAuthorizerPython	34.3323	34.3323	87.738
3	tokenAuthorizerNode	54.3594	54.3594	67.7109
4	tokenAuthorizerJava	77.2476	78.2013	43.869

Figure 16: Token Max Used Memory Query Result

7 Advanced Topics

This section is added to explain how to make advanced customization to applications that involve changing templates, configurations, and codes.

7.1 Add/Modify New Runtime Scenarios

If the user wants to add new language runtimes as a scenario, the user needs to modify the script inside *serverless-apps-builder/template.yaml*. As mentioned in the Installation section, this application uses AWS SAM as a serverless application builder. The AWS SAM full developer guide can be found here.

Below is the example template to add a new language runtime:

```
### Lambda <Request|Token> Authorizer <Runtime>
<Request|Token>Authorizer<Runtime>Resource:
  Type: AWS::ApiGateway::Resource
  Properties:
    ParentId: !GetAtt AppApi.RootResourceId
    PathPart: "<req|tkn>-auth-<runtime>"
    RestApiId: !Ref AppApi
```

```
# GET Method with Lambda authorizer <request|token> enabled
<Request|Token>Authorizer<Runtime>Get:
  Type: AWS::ApiGateway::Method
  Properties:
    RestApiId: !Ref AppApi
    ResourceId: !Ref <Request|Token>Authorizer<Runtime>Resource
    HttpMethod: GET
    AuthorizationType: CUSTOM
    AuthorizerId: !Ref AuthorizersLambda<Request|Token><Runtime>
    Integration:
      Type: AWS_PROXY
      IntegrationHttpMethod: POST
      Uri:
        !Join [
          "",
          [
            "arn:aws:apigateway:",
            !Ref AWS::Region,
            ":lambda:path/2015-03-31/functions/",
            !GetAtt App<Request|Token>Authorizer<Runtime>Function.
              ↪ Arn,
            "/invocations",
          ],
        ]

# Lambda <Request|Token> Authorizer
AuthorizersLambda<Request|Token><Runtime>:
  Type: AWS::ApiGateway::Authorizer
  Properties:
    Name: AuthorizersLambda<Request|Token><Runtime>
    Type: REQUEST
    RestApiId: !Ref AppApi
    IdentitySource: <method.request.querystring.QueryString1|method.request.
      ↪ header.AuthorizationToken>
    AuthorizerResultTtlInSeconds: 0
    AuthorizerUri:
      !Join [
        "",
        [
          "arn:aws:apigateway:",
          !Ref AWS::Region,
          ":lambda:path/2015-03-31/functions/",
          !GetAtt <Request|Token>Authorizer<Runtime>Function.Arn,
          "/invocations",
        ],
      ]
```

```
# App<Request|Token>Authorizer<Runtime> function
App<Request|Token>Authorizer<Runtime>Function:
  Type: AWS::Serverless::Function
  Properties:
    FunctionName: app<Request|Token>Authorizer<Runtime>
    Description: <Request|Token> Authorizer <Runtime> Application
    Runtime: <runtimeVersion>
    CodeUri: scenarios/<runtime>
    Handler: app<Request|Token>Authorizer.lambda_handler
    MemorySize: 128
    Timeout: 3

# <request|token>Authorizer<Runtime> function
<Request|Token>Authorizer<Runtime>Function:
  Type: AWS::Serverless::Function
  Properties:
    FunctionName: <request|token>Authorizer<Runtime>
    Description: <Request|Token> Authorizer <Runtime>
    Runtime: <runtimeVersion>
    CodeUri: scenarios/<runtime>
    Handler: <request|token>Authorizer.lambda_handler
    MemorySize: 128
    Timeout: 3

# Permission to allow App<Request|Token>Authorizer<Runtime>Function
   ↪ invocation from API Gateway
App<Request|Token>Authorizer<Runtime>Permission:
  Type: AWS::Lambda::Permission
  Properties:
    FunctionName: !Ref App<Request|Token>Authorizer<Runtime>Function
    Action: lambda:InvokeFunction
    Principal: apigateway.amazonaws.com
    SourceArn: !Sub arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${
        ↪ AppApi}/*/GET/<req|tkn>-auth-<runtime>

# Permission to allow <Request|Token>Authorizer<Runtime>Function
   ↪ invocation from API Gateway
<Request|Token>Authorizer<Runtime>FunctionPermission:
  Type: AWS::Lambda::Permission
  Properties:
    FunctionName: !Ref <Request|Token>Authorizer<Runtime>Function
    Action: lambda:InvokeFunction
    Principal: apigateway.amazonaws.com
    SourceArn: !Sub arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${
        ↪ AppApi}/authorizers/${AuthorizersLambda<Request|Token><Runtime>
        ↪ }
```

```
### Deployment and Output Setup
Deployment:
  Type: AWS::ApiGateway::Deployment
  DependsOn:
    - <Request|Token>Authorizer<Runtime>Get
  Properties:
    RestApiId: !Ref AppApi

Outputs:
# API Gateway endpoint to be used during performance tests
<Request|Token>Authorizer<Runtime>:
  Description: <Request|Token> Authorizer <Runtime> Get Endpoint
  Value: !Sub "https://${AppApi}.execute-api.${AWS::Region}.amazonaws.com/
    ↪ v1/<req|tkn>-auth-<runtime> <\?QueryString1=queryValue1| -H \"
    ↪ AuthorizationToken: Bearer allow\">"
```

7.2 Modify Artillery Performance Test Tests

As explained in section Artillery reports, the artillery performance test uses templates from folder *config/templates* to generate actual Artillery performance configuration files in folder *config/artillery*. Users can change the contents of the template as the user wants. Detailed information about artillery templates is available here.

7.3 Add/Modify Logs Insight Queries

It is necessary to change the code in *labt.js* if the user wants to modify or add to the Log Insight query. Users can read the document here to learn the AWS CloudWatch Logs Insight query language. Below is the code section that the user needs to modify:

```
// runAwsCloudWatchLogsInsight method
let commandQuery<queryTitle> = 'aws logs start-query ' +
       '--log-group-names ' + <identifierParams|identifierRequestParams|
          ↪ identifierTokenParams> +
       ' --start-time ' + startTime +
       ' --end-time ' + endTime +
       ' --query-string \'<your_query>\'' +
       ' > outputs/logs_insight/query_id_<queryTitle>.json';

// Add the code afterward
executeLogsInsight(<identifierParams|identifierRequestParams|
   ↪ identifierTokenParams> !== '' ? <queryTitle> : '', <errorMessage>,
   ↪ <delayMessage>, "<queryTitle>.json");
```