National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Siddhesh Manik Gaikwad
Student ID: x23355824

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Siddhesh Manik Gaikwad

Student ID: X23355824

Programme: MSc Cloud Computing **Year:** 2025

Module: Research Project

Lecturer: Rashid Mijumbi

Submission Due Date: 28th January 2026

Project Title: Performance Comparison of AWS Lambda and Azure Functions Using Spring Boot and Quarkus Java Framework

Word Count: 1145 **Page Count:** 4

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: Siddhesh Manik Gaikwad

Date: 28th January 2026

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):	

Configuration Manual

Siddhesh Manik Gaikwad
Student ID: x23355824

1 Introduction

This document acts as a complete reference for configuring, building, deploying, and testing all four serverless applications created for the project titled, "Performance Comparison between AWS Lambda and Azure Functions Built using Spring Boot and Quarkus". It has been implemented in such a way as to automate as much as possible, in order to provide standardisation and repeatability.
Using this manual will provide you with everything you need to reproduce the entire experimental set up including compiling the source code to executing performance tests using the actual cloud endpoints.

2 System Requirements

Before proceeding, ensure your local development environment meets the following requirements:

- **Operating System:** A Unix-like environment (Linux, macOS, or Windows with WSL) or Windows with PowerShell 7+ and Git Bash. The automation scripts are cross-platform.
- **Git:** Version 2.25 or later, for cloning the project repository.
- **Java Development Kit (JDK):** Java 17 or later (Java 21 is recommended to match the Azure runtime).
- **Apache Maven:** Version 3.8 or later, for building the Java projects.
- **AWS CLI:** Version 2.x, configured with credentials for your AWS account.
- **Azure CLI:** Version 2.x, logged into your Azure account.
- **Postman (Optional):** For manual API testing and verification. The provided postman-collection.json can be imported.

3 Initial Setup and Configuration

First, clone the project repository to your local machine:
git clone <repository-url>
cd <repository-directory>
The automation scripts rely on the AWS and Azure CLIs being correctly configured.
AWS CLI Configuration:

1. Ensure you have an IAM user with programmatic access (Access Key ID and Secret Access Key) and sufficient permissions to create IAM roles, Lambda functions, and API Gateways.
2. Configure the AWS CLI by running:

aws configure

3. Enter your Access Key ID, Secret Access Key, and a default region (e.g., eu-north-1). The deployment scripts will override the region, but a default is required.

Azure CLI Configuration:

1. Log into your Azure account by running:

az login

2. Follow the instructions in your web browser to authenticate.

3. Set the subscription you wish to use for the deployment:

az account set --subscription "<your-subscription-name-or-id>"

4 Project Structure Overview

The project is organized into several key directories:

```
/
├── deployment-scripts/     # Contains shell scripts for deploying each function
├── load-testing/           # Contains scripts for performance testing and analysis
├── spring-boot-aws-lambda/ # Source code for the Spring Boot on AWS project
├── spring-boot-azure-functions/ # Source code for the Spring Boot on Azure project
├── quarkus-aws-lambda/     # Source code for the Quarkus on AWS project
├── quarkus-azure-functions/ # Source code for the Quarkus on Azure project
├── run.sh                  # Master script runner for all operations
└── ... (other configuration files)
```

5 The run.sh Master Script

Major operations are conducted from the run.sh script. Thus, this script is a common interface to build, deploy, and test. The script detects the operating system automatically and runs the appropriate scripts for the back end (Bash or PowerShell).

Usage:

./run.sh <command>

Available Commands:

- **build-all:** Compiles all four Java projects and creates the deployable artifacts.
- **deploy-all:** Deploys all four applications to their respective cloud platforms.
- **quick-test**: Runs the 10-request performance test against all deployed endpoints.
- **analyze-results:** Parses the CSV files generated by quick-test and produces summary reports.

6 Step-by-Step Execution Workflow

For the reproduction of the entire set of experiments, execute the following steps:

This step, alongside the use of Maven to compile the Java source code, packages each application into a deployable artifact.

1. Go to the project root directory.
2. Run the build-all command:

./run.sh build-all

3. The individual scripts will build the respective set of four projects. After successful completion, the deployable artifacts will now rest in the target/ directory of each project folder (i.e., for example, spring-boot-aws-lambda/target/). It may take several minutes.

The provisioning of the required cloud infrastructure and application code deployment will be done by the AWS and Azure CLIs in the following step.

Note: This step will create resources in your cloud accounts that may incur costs.

4. Make sure you have completed the CLI configuration in section 3.2.
5. From the root directory of the project, execute the deploy-all command:
6. ./run.sh deploy-all
7. The script will confirm before proceeding; type yes to confirm.
8. The script will deploy the four functions in order, which is quite a long procedure possibly taking around 20-30 minutes, particularly on the first run.
9. Upon completion, for every function, four text files would be created in the deployment-scripts/ directory, each one containing the public URL (e.g.

springboot-aws-endpoint.txt). These URLs are automatically used by the testing scripts.

This step will execute a "quick test" protocol, where it will send 10 requests to each endpoint and collect performance data.

From the project's root directory, run the quick-test command:

1. ./run.sh quick-test
2. The script will read the deployed endpoint URLs from the files generated during deployment.
3. It will then sequentially test all four endpoints, showing the response time in the console for each request.
4. After completing its execution, results will be dumped into CSV files inside the load-testing/results/ directory; four individual files (i.e., quick-test-SpringBoot-AWS.csv) and one summary file (i.e., quick-test-summary.csv) will be available.

This step will take raw CSV data and make a human-readable summary and markdown report.

10. From the project's root directory, run the analyze-results command:

1. ./run.sh analyze-results
2. The script will read the CSV files from the load-testing/results/ directory.
3. It will print a comparative summary table in the console and create an extensive markdown report called PERFORMANCE_REPORT.md in the results directory.

7 Manual Verification (Using Postman)

You can manually verify that each endpoint is working correctly using a tool like Postman or curl.

1. Open the endpoint files in the deployment-scripts/ directory to get the URLs.
2. Import the postman-collection.json file (located in the root of the project) into Postman. This collection contains pre-configured requests for all four endpoints.
3. Send a GET request to any of the endpoint URLs. You should receive an HTTP 200 OK response with a JSON body matching the schema described in the research report, for example:


```
{
 "message": "Performance test response",
 "timestamp": "2025-12-01T10:30:45.123Z",
 "framework": "Spring Boot",
 "platform": "AWS Lambda",
 "computation": 12470
}
```


8 Cleaning Up Resources

To avoid ongoing cloud costs, you must delete the resources created by the deployment scripts.

- **AWS Cleanup:**
    1. Go to the AWS Lambda console, and delete the springboot-performance-test and quarkus-performance-test functions.
    2. Go to the Amazon API Gateway console, and delete the springboot-perf-api and quarkus-perf-api APIs.
    3. Go to the AWS IAM console, and delete the lambda-execution-role-springboot and lambda-execution-role-quarkus roles.
- **Azure Cleanup:**
    1. The easiest way to clean up all Azure resources is to delete the entire resource group.
    2. Run the following Azure CLI command:

```
az group delete --name thesis-performance-rg --yes --no-wait
```

This command will delete the resource group and all resources within it, including the Function Apps and Storage Accounts.