National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Yashashwini Suresh
Student ID: x23251263

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Yashashwini Suresh
Student ID:	x23251263
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	8198
Page Count:	6

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:	10th December 2025

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

Yashashwini Suresh
x23251263

1 Introduction

This configuration document gives the entire technical infrastructure to set up, deploy, test and recreate the experimentation done on the MSc research project, Evaluating Sustainability of Virtual machines, Containers, and Serverless computing on AWS. The manual describes the environment setup, system requirements, project structure in a folder, EC2, Docker, and AWS Lambda deployment, as well as, how to run load tests and gather energy-related data.

This manual is aimed at achieving 100 percent reproducibility so that future researchers may also be able to replicate all experiments, such as deploying microservices, load testing them with Locust, extracting CloudWatch metrics, and calculating sustainability (Energy, Water, Efficiency).

2 Scope

The following configuration manual is that of:

- Deploying AWS models such as EC2, Docker-onEc2 and Lambda
- Foundation which is an infrastructure deployment done with Terraform.
- Installing and executing of microservice as a base for other models.
- Image building and ECR process.
- Creation of lambda packaging and creation of function URL.
- Locust (Low, Medium, High workload profiles) load test.
- CPU, invocation count, and duration AWS CloudWatch metric retrieval.
- Calculations required to carry out sustainability analysis: Energy, Water, and Efficiency.

Strengthening Security or monitoring the enterprise, CI/CD pipelines, and multi-cloud deployments have not been included in the document.

3 Project Objective

The configuration set up is aimed at providing a controlled and repeatable benchmarking in three compute models:

1. EC2 Virtual machines (persistent infrastructure)
2. Docker Containers with EC2 (lightweight virtualisation)
3. The AWS Lambda Serverless Functions (event-driven compute) are used.

In this test, to be able to give an evaluation it is necessary to constantly deploy the same FastAPI microservice exposing:

- Fibonacci processing CPU -bound, helps in computing the fibonacci series load.
- io for minimalistic JSON deserialization or serialization file read.

It is also possible to configure automated metric extraction and formula based sustainability calculation in the manual.

4 System Requirements

Hardware Requirements	**Software Requirements**
8 GB RAM minimum (Local system)	Windows 10/11 / macOS / Ubuntu
Dual-core CPU or higher	Python 3.11+
10 GB available disk space	Docker Desktop
Stable internet connection	AWS CLI v2
GPU not required	Terraform v1.6+
	VS Code + AWS Toolkit (optional)
	fastapi, uvicorn, boto3, locust, python-dotenv

Table 1: Hardware and Software Requirements

5 Installation Requirements

5.1 Cloning Repository:

```
git clone https://github.com/<your-repo>/GreenTechnology_Research.git
cd GreenTechnology_Research
```

5.2 Creating Virtual Env:

```
python3 -m venv .venv
source .venv/bin/activate      # for macOS or Linux
```

or use below

```
.venv\Scripts\activate         # for Windows
```

5.3 Installing Dependencies:

```
pip install -r requirements.txt
```

5.4 Structure of the Project:

```
GreenTechnology_Research/
├── services/
│   └── app/
│       ├── main.py          # this is where FastAPI microservice code is.
│       ├── requirements.txt
│       └── Dockerfile
├── infra/
│   └── aws/
│       └── terraform/
│           ├── main.tf      # EC2 setup done in here
│           ├── ecs_ec2.tf   # Docker/ECS setup for image
│           ├── lambda.tf    # AWS Lambda setup
│           ├── variables.tf
│           ├── outputs.tf
│           └── provider.tf
├── load_tests/
│   ├── locustfile.py
│   ├── low.json / medium.json / high.json
├── scripts/
│   ├── lambda_load_test.py
│   ├── collect_cloudwatch.py
│   ├── calculate_energy.py
├── results/
│   ├── raw/                 # Load test CSV output
│   └── processed/           # Final aggregated results
└── README.md
```

Figure 1: structure of this project

6 Deployment and Config Setup

6.1 EC2 deployment by FastAPI and Uvicorn:

Step 1: **setting up Terraform** Edit - variables file as per your region, type of instance and name of your project if needed.

```
region = "us-east-1"
instance_type = "t3.medium"
project = "sust-compute-study"
```

Step 2: **Deploy the EC2**

```
terraform init
terraform apply -auto-approve
```

This output gives you two things:

- EC2 Public IP - when you perform apply, it provides you the IP in the end use it for further steps.
- `App URL(http://<ip>:8080/docs) - Apply the IP you obtained here`

Step 3: **Verify the Deployment of EC2**

```
curl -I http://<public-ip>:8080/docs
```

6.2 Docker on EC2 Deployment

Step 1: **Building Image for Docker**

```
docker build -t sust-micro:v1 services/app
```

Step 2: **Authenticate to ECR to check the login info**

```
aws ecr get-login-password | docker login ...
```

Step 3: **Pushing Image**

```
docker tag sust-microservice:v1 <ACCOUNT_ID>.dkr.ecr.us-east-
1.amazonaws.com/sust-microservice:v1
```

Push the Image to the Account_ID:

```
docker push <ACCOUNT_ID>.dkr.ecr.us-east-1.amazonaws.com/sust-microservice:v1
```

Step 4: **To Run Container on EC2**

```
docker run -d -p 8080:8080 --ulimit nofile=65535:65535 sust-microservice:v1
```

6.3 Lambda Deployment on AWS

Step 1: **To Compute, first step is to Zip**

```
cd services/lambda/compute
zip -r compute.zip .
```

Step 2: **Creating the Lambda Function**

```
  aws lambda create-function \
--function-name sust-compute \
--runtime python3.11 \
--handler handler.lambda_handler \
--zip-file fileb://compute.zip \
--role arn:aws:iam::<ACCOUNT_ID>:role/LabRole
```

Step 3: **Test URL Function using below:**

```
  aws lambda create-function-url-config \
--function-name sust-compute \
--auth-type NONE
```

7 Testing using Loads

7.1 EC2 & Docker Load run tests:

```
  locust -f load_tests/locustfile.py --headless \
-u 50 --spawn-rate 5 --host http://<IP> -t 6m --csv=results/ec2_low_run1
```

Workloads:

- Low: Uses 50 users for the load
- Medium: 200 users for the load
- High: 800 users for the load

Repeating 10 runs per workload gives you an average of 10 runs which will be helpful for varied circumstances.

7.2 Running Loads for Lambda:

```
  python scripts/lambda_load_test.py \
--url $COMPUTE_URL \
--name lambda_low_run1 \
--duration 120 \
--delay 0.5 \
--compute
```

likewise run for multiple loads, change the name - *run1, *run2 etc. Change the Duration and Delay according to Med and High load as below:

- Medium Load: duration 120; delay 0.2
- High Load: duration 180; delay 0.1
- Same goes for io runs, just by changing –compute to –io

8 Metrics Collection

To collect the values, like cpu percentage that the load has used, use the below in terminal to get the results.

For EC2 and Docker values: we will now use the stat first to get the end time of the load and edit the cloudwatch collection code according to the stat end time: stat -c '%y' results/'ec2 or docker file name'*_stats.csv

```
  aws cloudwatch get-metric-statistics \
--namespace AWS/EC2 \
--metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=$INSTANCE_ID \
--start-time <start> --end-time <end> \
--period 60 \
--statistics Average
```

For Lambda Metrics:

```
  aws cloudwatch get-metric-statistics \
--namespace AWS/Lambda \
--metric-name Duration ...
```

9 To Calculate the Energy and Water

9.1 For Docker and EC2:

$$E_{kWh} = \frac{CPU\% \times 60W \times 360s}{100 \times 3600}$$

$$Water = E \times 1.8$$

Where: P is 60W • T=360 secs • WUE=1.8 L/kWh

9.2 For Lambda:

$$Energy_{\lambda} = \frac{Invocations \times Duration_{ms}}{1000} \times 7W$$

10 Guide to Troubleshoot errors:

Issue	Cause	Fix
503 from ALB	Target group not healthy	Check health checks or the container port
Lambda returns 429	Too many requests	Reduce delay time and workload and rerun
Docker errors "address already in use"	Port conflict	Stop previous container: `docker ps` → `docker rm`

Table 2: troubleshoot guide

11 Summary

This configuration manual makes it possible to entirely reproduce all the experiments in the research project. The steps will make EC2, Docker, and Lambda environments deployments consistent, as well as perform precise load execution with Locust and extract reliable metrics of the CloudWatch, to maintain sustainability analysis.