National College of Ireland

Configuration Manual

MSc Research Project
Master of Science in Cloud Computing

Ann Mariya Jojo
Student ID: x23241535

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Ann Mariya Jojo
Student ID:	x23241535
Programme:	Master of Science in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1300
Page Count:	9

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:	Ann Mariya Jojo
Date:	25th January 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

Ann Mariya Jojo
x23241535

1 Introduction

The documents consist of each steps that needs to be followed for setting up and executing the load balancing experiment.There are two cases mainly that is traditional VM based approach and Case 2 that is Implementation of Hybrid GWO-PSO algorithm on AWS Lambda for optimization.

2 Case 1 Experiment

The following prerequisites are required before setting up the experiment.

- An AWS account with the permission to create and access the EC2 instances, load balancers,Lambda and IAM services,
- For traffic simulation the artillery is required for Case 1 and Case 2.
- Boto3, numpy, requests libraries and python 3.9 is requirred.

2.1 Creating EC2 Instances

For the creation of EC2 instances first navigate into EC2 dashboad in AWS account.The instance types is t2 micro and the AMI is Amazon Linux 2.The security groups are which allows inbound HTTP that isp ort 80 from 0.0.0.0/0 and SSH that is port 22 from your IP that is for debugging. For SSH access a Key Pair is also required.

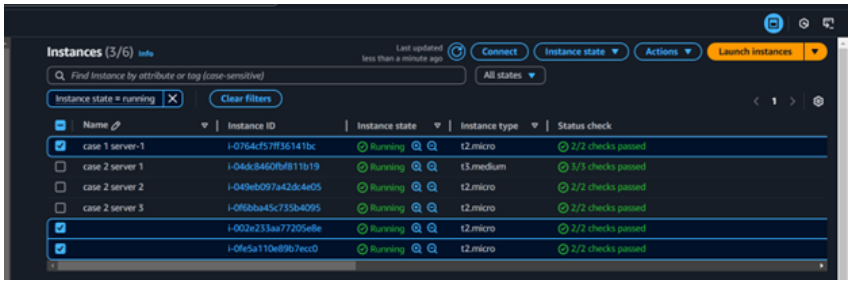

Figure 1: The EC2 instances for case 1

For installing and running the Apache Web server the commands used are given below;

- sudo yum update -y

- sudo yum install -y httpd
- sudo systemctl start httpd
- sudo systemctl enable httpd
- echo "Welcome to Case 1 Server" — sudo tee /var/www/html/index.html

2.2 Setting up of Application Load balancer

The scheme used is Internet facing and the listeners is port 80 that is HTTP.The target groups to add the EC2 instances which is created earlier and to configure the path for health check Wee and Liu (2010) up.By accessing the DNS name in a browser it can be verified that Application Load balancer DNS is reachable or not.

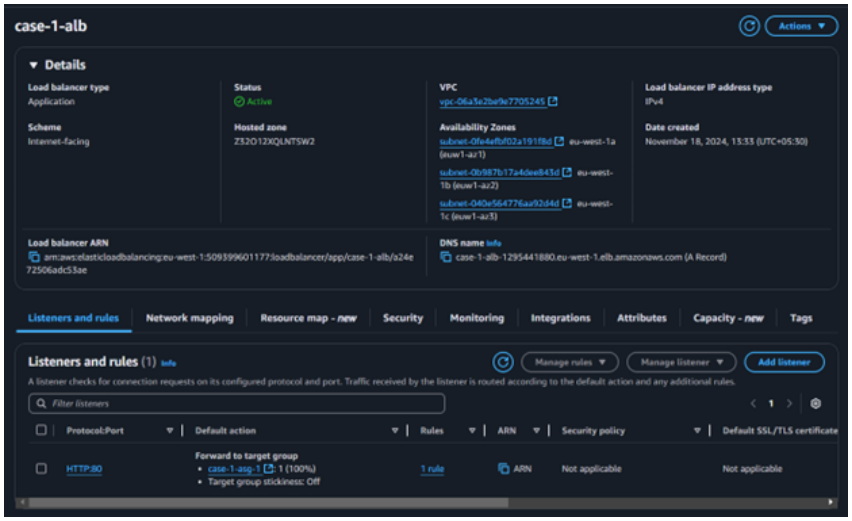

Figure 2: Setting up of ALB

2.3 Configuration of Auto scaling Group

The next step is creating the Auto scaling group for that navigate to the Auto Scaling section in AWS accountArvindhan and Anand (2019).The target group that is created previously is attached and number of instances for minimum and maximum is also set.Make sure that auto scaling policy which is triggered is based on CPU utilization.

2.4 Simulation of traffic with the help of Artillery

Artillery is installed on the local machine using the command npm install -g artillery@latest and the configuration file created in the name of case1-test.yml and run the simulation for output.The given below is test case 1 in figure 4 .

3 Case 2 Experiment

The prerequisites for setting up the Case 2 experiment is already mentioned on the prerequisites of first experiment.

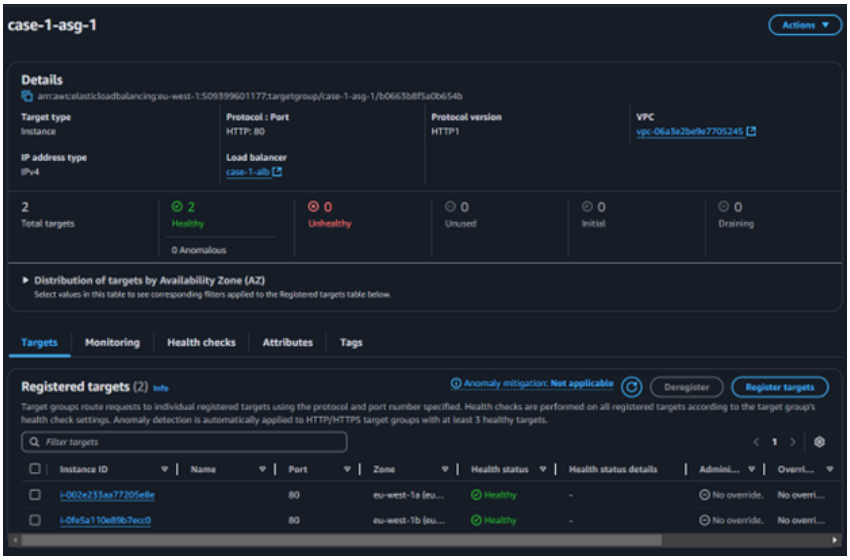

Figure 3: Auto Scaling group Configuration

```yaml
config:
  target: "http://case-1-alb-1295441880.eu-west-1.elb.amazonaws.com"
  phases:
    - duration: 300   # Run for 5 minutes
      arrivalRate: 20 # 20 requests per second
scenarios:
  - flow:
      - get:
          url: "/"
```

Figure 4: case1-test.yml

3.1 Creating EC2 instances

The same steps that is mentioned for Case 1 scenario is followed here also . The number of instances created is three.

Figure 5: EC2 instances for case 2 scenario

3.2 Configuration of AWS Lambda

The Lambda function is created the runtime is python 3.9, memory is 1024 MB and the time out is 30 seconds.The IAM role attached for the function are:

- AWSLambdaBasicExecutionRole
- AmazonEC2ReadOnlyAccess

- CloudWatchFullAccess

The script of the GWO-PSO is added with the help of lambda handler and deploy the lambda function.

3.3 Application Load balancer is configured for Lambda

The application load balancer is created as mentioned in the scenario of Case 1 .The same step is followed for the Case 2 also and added a Lambda target group. Also attach the Lambda function as target.The health check path is configured and also to the Artillery test script the DNS endpoint is added.

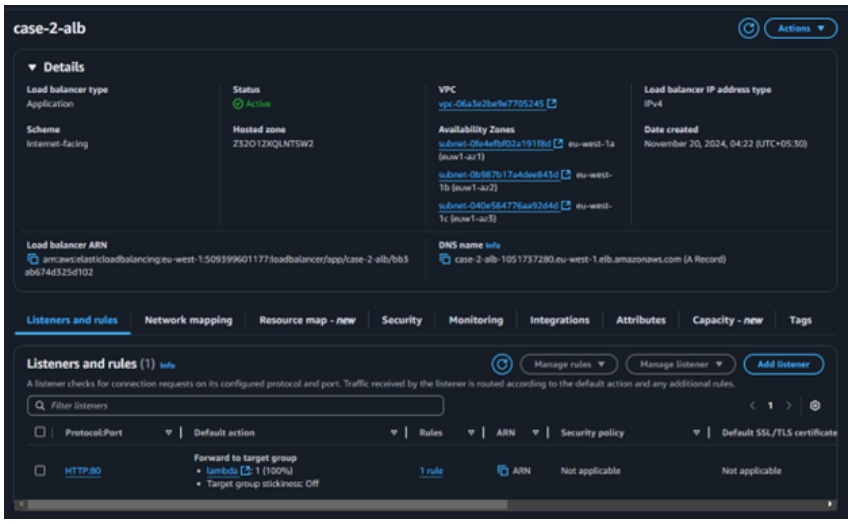

Figure 6: Case 2 ALB

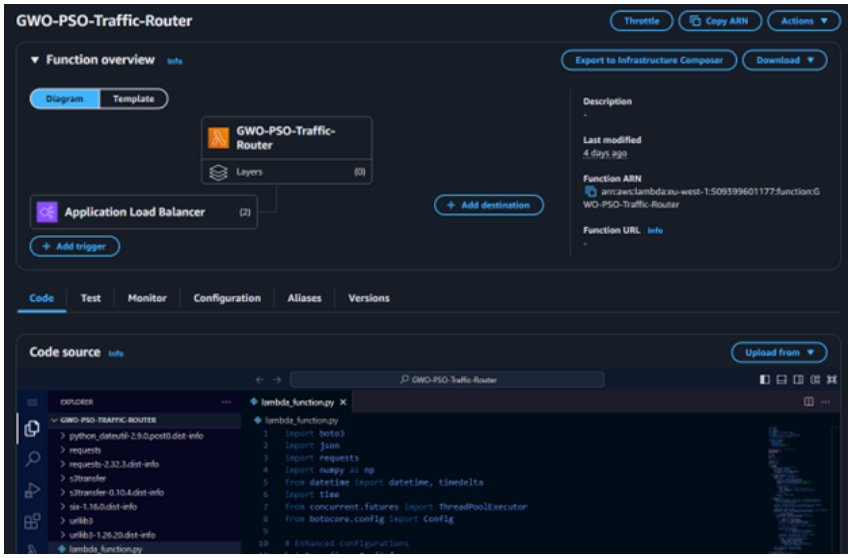

Figure 7: Lambda function of Case 2

3.4 Lambda function Testing

As mentioned the scenario in case 1. The same way yaml file is created for case 2 scenario and updated the artillery configuration fileLin et al. (2018).The name of the file case2-test.yml file. The script is given below and the simulation is executed with the help of the command artillery run case2-test.yml.

```yaml
config:
  target: "http://case-2-alb-1051737280.eu-west-1.elb.amazonaws.com"
  phases:
    - duration: 300
      arrivalRate: 20
scenarios:
  - flow:
      - get:
          url: "/lambda"
```

Figure 8: case2-test.yml

3.5 On Server setting up of Artillery

For running artillery the SSH into the server is designated.The Node.js and artillery is installed.The given below are the commands.

- curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh — bash
- source /.bashrc
- nvm install –lts
- npm install -g artillery@latest

The configuration files of artillery that is case1-test.yml and case2-test.yml is copied to server and test is executed with the help of commands that is given above.

3.6 IAM Roles required

The IAM role required for the EC2 instances is CloudWatchAgentServerPolicy.This policy is required mainly for monitoring.The policy required for Lambda functions are given below;

- AmazonEC2FullAccess
- AmazonEC2ReadOnlyAccess
- AWSLambdaBasicExecutionRole
- AWSLambdaVPCAccessExecutionRole
- CloudWatchFullAccess
- CloudWatchFullAccessV2

3.7 Validation of Result

From the artillery output he result is collected and the metrics mainly errors, throughput, execution time and response time are evaluated for both scenarios and it is showed in figure 7 and 8.

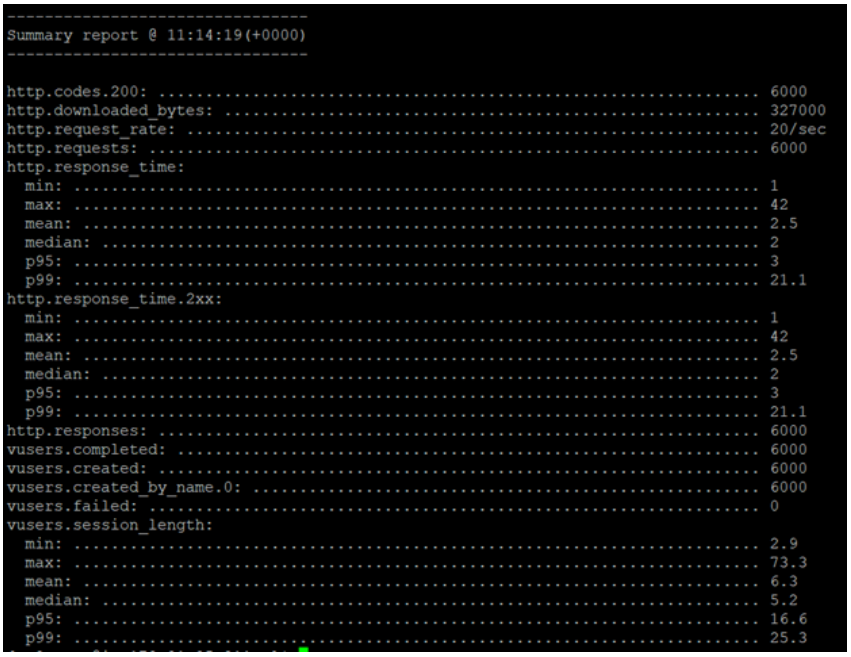

Figure 9: Metrics collection of Case 1

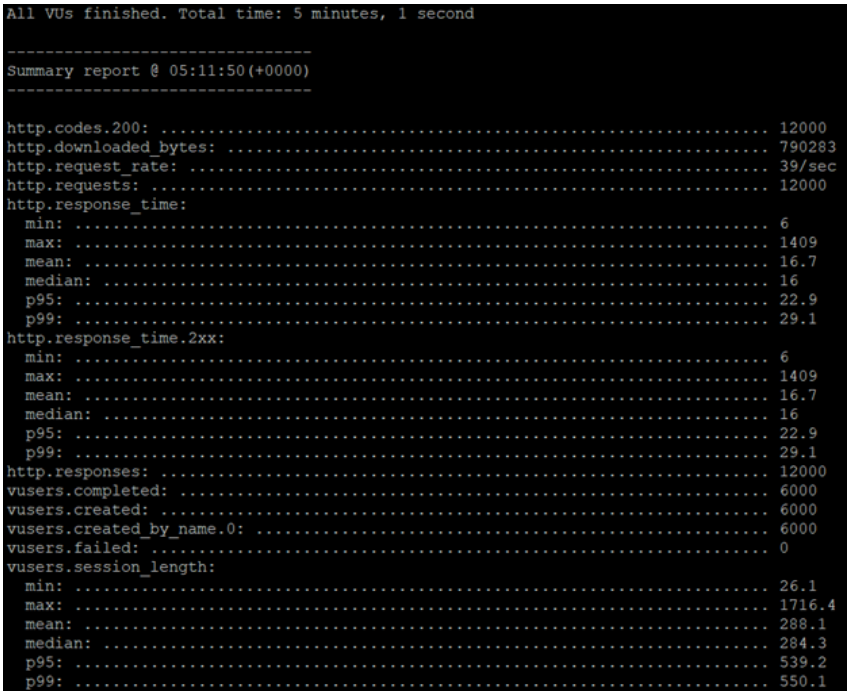

Figure 10: Metrics collection of case 2

3.8 GWO-PSO hybrid algorithm code

The given below is the code :

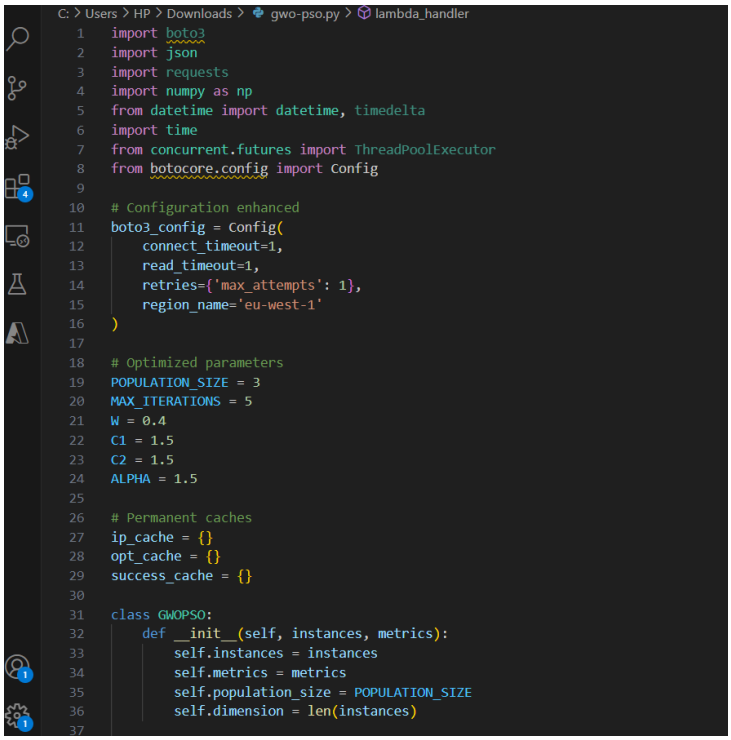

Figure 11: Hybrid algorithm implementation

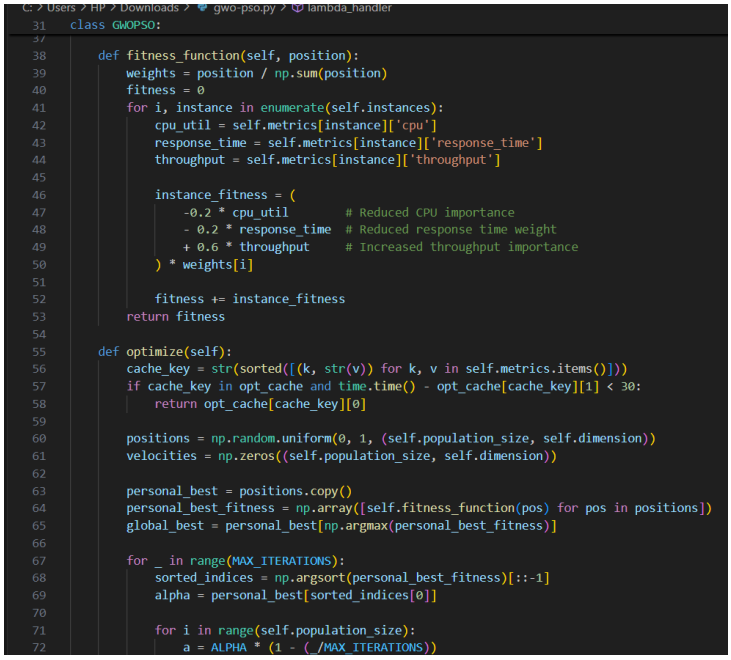

Figure 12: Hybrid algorithm implementation

```
class GWOPSO:
    def optimize(self):
            for i in range(self.population_size):
                a = ALPHA * (1 - (_/MAX_ITERATIONS))
                A1 = a * (2 * np.random.random(self.dimension) - 1)
                C1_wolf = 2 * np.random.random(self.dimension)

                X1 = alpha - A1 * np.abs(C1_wolf * alpha - positions[i])

                r1, r2 = np.random.random(2)
                velocity = (W * velocities[i] +
                            C1 * r1 * (personal_best[i] - positions[i]) +
                            C2 * r2 * (global_best - positions[i]))

                positions[i] = positions[i] + 0.7 * X1 + 0.3 * velocity
                positions[i] = np.clip(positions[i], 0, 1)

                fitness = self.fitness_function(positions[i])
                if fitness > personal_best_fitness[i]:
                    personal_best[i] = positions[i]
                    personal_best_fitness[i] = fitness
                    if fitness > self.fitness_function(global_best):
                        global_best = positions[i]

        result = global_best / np.sum(global_best)
        opt_cache[cache_key] = (result, time.time())
        return result

def lambda_handler(event, context):
    start_time = time.time()

    if event.get('path') == '/health':
        return {
            'statusCode': 200,
            'body': json.dumps({
                'status': 'healthy',
```

Figure 13: Hybrid algorithm implementation

```
    if event.get('path') == '/health':
        return {
            'statusCode': 200,
            'body': json.dumps({
                'status': 'healthy',
                'timestamp': datetime.utcnow().isoformat()
            })
        }

    instances = [
        'i-04dc8460fbf811b19',
        'i-049eb097a42dc4e05',
        'i-0f6bba45c735b4095'
    ]

    metrics = {}
    for instance in instances:
        metrics[instance] = get_instance_metrics(instance)

    optimizer = GWOPSO(instances, metrics)
    weights = optimizer.optimize()

    sorted_instances = [instances[i] for i in np.argsort(weights)[::-1]]

    for instance_id in sorted_instances:
        try:
            target_ip = get_instance_ip(instance_id)

            if event.get('path') == 'lambda':
                execution_time = np.random.uniform(1.5, 2.5)
                time.sleep(0.1)

                return {
                    'statusCode': 200,
                    'body': json.dumps({
```

Figure 14: Hybrid algorithm implementation

Figure 15: Hybrid algorithm implementation

References

Arvindhan, M. and Anand, A. (2019). Scheming an proficient auto scaling technique for minimizing response time in load balancing on amazon aws cloud, *International Conference on Advances in Engineering Science Management & Technology (ICAESMT)-2019, Uttaranchal University, Dehradun, India*.

Lin, W.-T., Krintz, C., Wolski, R., Zhang, M., Cai, X., Li, T. and Xu, W. (2018). Tracking causal order in aws lambda applications, *2018 IEEE international conference on cloud engineering (IC2E)*, IEEE, pp. 50–60.

Wee, S. and Liu, H. (2010). Client-side load balancer using cloud, *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 399–405.