National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Ashish Oli
Student ID: x23102926

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ashish Oli
Student ID:	x23102926
Programme:	Cloud Computing
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	959
Page Count:	7

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:	Ashish Oli
Date:	11th August 2024

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

Ashish Oli
x23102926

1 Prerequisites

1.1 Software Requirements

- Node.js: Version 14.x or higher
- npm: Version 6.x or higher (comes with Node.js)
- Docker: Latest version
- Git: For version control
- wrk: HTTP benchmarking tool
- ghz: gRPC benchmarking tool
- SSH Client: For connecting to AWS EC2 instances

1.2 Hardware Requirements

- AWS EC2 instances - for this project I have used free tier (t2.micro free)
- Minimum 2 GB RAM for local testing

1.3 Setup cloud account

1. Create an AWS Account: Please follow the below link to setup an AWS account - Create AWS account
2. Creating an EC2 instance: Please follow the link given to create an ec2 instance - Create an Ec2 instance

NOTE: Please store your pem keys provided by AWS safely with correct permission for later to use it to ssh into the server.

2 Environment Setup

This section provides a detailed guide for setting up the project environment, pulling code from repositories, running the servers, and performing benchmarks. The steps are organized for clarity, ensuring that each part of the setup is correctly implemented.

2.1 Environment Setup

NOTE: This setup is for both Local setup and cloud instance setup.

NOTE: This setup is for configuring, running and testing application both manually(through code) and using docker image from dockerhub.

If you are running this directly in cloud environment using docker, then you don't need to setup node environment as the docker image will be packed with the required environment. You can skip the *Node.js Environment* steo in sub section 2.1.1.

2.1.1 Node.js Environment

To set up the Node.js environment, follow these steps:

```
# Install Node.js, npm, and nvm
sudo apt install -y nodejs npm

# For installing nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# Switch to the latest LTS version using nvm
nvm install --lts
nvm use --lts

# Verify installation
node --version
npm --version
```

2.1.2 Docker Environment

Set up Docker by executing the following commands:

```
# Install Docker
sudo apt install docker.io

# Add Docker to the user group to avoid using sudo with Docker commands
sudo usermod -aG docker $USER

# Verify Docker installation
docker --version
```

2.1.3 Homebrew for Tool Installation

Install Homebrew to manage additional tooling:

```
/bin/bash -c \
"$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2.2 Cloning and Running the Code

2.2.1 Cloning the Repositories

Clone the repositories for both HTTP and gRPC servers:

HTTP Server:

```
git clone https://github.com/oliashish/http-server-primary
git clone https://github.com/oliashish/http-server-replica
```

gRPC Server:

```
git clone https://github.com/oliashish/gRPC-server-primary
git clone https://github.com/oliashish/gRPC-server-replica
```

2.2.2 Running the Servers Manually

Navigate to the directory of each server and run the following commands:

NOTE: Run the replicas first to obtain IP and service running for primary.

For HTTP Replica:

```
cd replica
npm install
node replica.js
```

For HTTP Primary:

```
cd primary
npm install
node primary.js
```

For gRPC Replica:

```
cd replica
npm install
node replica.js
```

For gRPC Primary:

```
cd primary
npm install
node primary.js
```

2.3 Setting Up with Docker

Use Docker to set up the environment with publicly available images:

```
docker run -d --rm --name=http-replica -p 3001:3001 http-replica
docker run -d --rm --name=http-primary -p 3000:3000 http-primary
docker run -d --rm --name=gRPC-replica -p 50051:50051 gRPC-replica
docker run -d --rm --name=gRPC-primary -p 50050:50050 gRPC-primary
```

2.4 Testing the Code

2.4.1 Testing the HTTP Server

You can test the HTTP server using a web client like Postman, ThunderClient, or ‘cURL‘:

```
curl -X POST -d '{"task": "Test string for checking word and character count"}'\
http://localhost:3000/execute
```

2.4.2 Testing the gRPC Server

For gRPC server testing, use ‘grpcurl‘ as normal web clients won’t work:

```
brew install grpcurl

# Test gRPC API
grpcurl -plaintext -import-path ./ -proto tasks.prot\
-d '{"task": "Test string for checking word and character count"}' \
localhost:50051 TaskService.Execute
```

2.5 Benchmarking Setup

2.5.1 Benchmarking HTTP Server with wrk

Install ‘wrk‘ for benchmarking the HTTP server:

```
sudo apt-get install build-essential libssl-dev git -y
git clone https://github.com/wg/wrk.git wrk
cd wrk
sudo make
sudo cp wrk /usr/local/bin
```

Create a ‘post.lua‘ file in the root directory of the HTTP primary server project:

```
wrk.method = "POST"
wrk.body   = '{"task": "Test string for checking word and character count"}'
wrk.headers["Content-Type"] = "application/json"
```

Run the ‘wrk‘ command:

```
wrk -t12 -c400 -d30s -s post.lua <your domain or local IP>/execute
```

2.5.2 Benchmarking gRPC Server with ghz

Install ‘ghz‘ for benchmarking the gRPC server:

```
brew install ghz
```

Run the ‘ghz‘ command:

```
ghz --insecure --proto ./tasks.proto --call TaskService.Execute \
-d '{"task": "Test string for checking word and character count"}' \
--concurrency=50 --connections=20 --duration=30s \
<your domain or local IP>
```

2.6 Analyzing Benchmark Results

Pipe the results to a file for analysis:

```
wrk -t12 -c400 -d30s -s post.lua \
<your domain or local IP>/execute > http-results.txt

ghz --insecure --proto ./tasks.proto --call TaskService.Execute \
-d '{"task": "Test string for checking word and character count"}' \
--concurrency=50 --connections=20 --duration=30s \
<your domain or local IP> > gRPC-results.txt
```

Now you can analyze the results to understand the performance and efficiency of the HTTP and gRPC servers.

3 Analyzing results

Now let use see what is the report structure of the benchmarking tools that we have and how it will look like when you test it on your local machine

1. **wrk**:

   When you run wrk for benchmarking with following command:

   ```
   wrk -t12 -c400 -d30s -s post.lua <your domain or local IP>
   ```

   You’ll see a similar result structure as below.

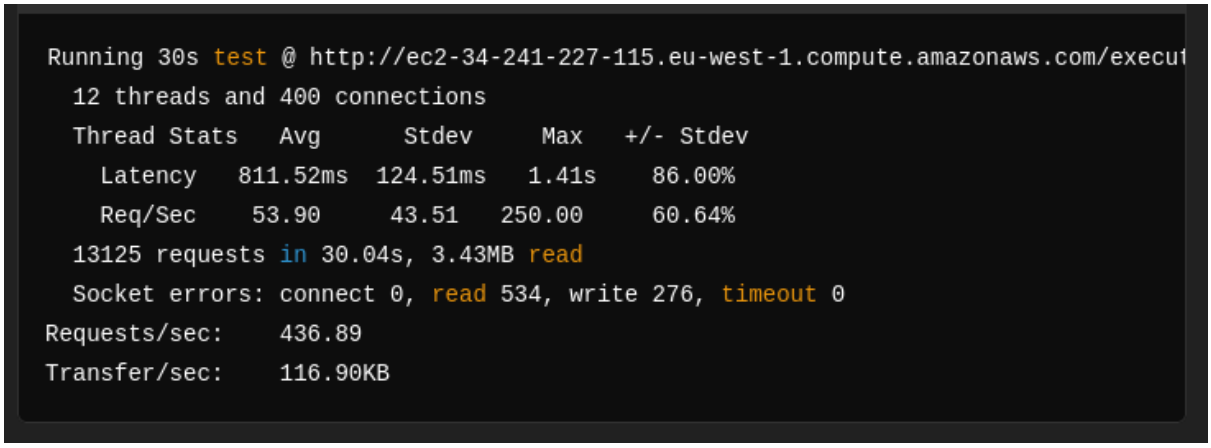

Figure 1: wrk benchmarking results structure

- number of connections
- latency - avg, min, max
- throughput - avg, min, max
- socket errors
- number of request
- request/sec
- transfer/sec

2. ghz:

   When you run the benchmarking for gRPC using ghz you'll see the following output format

   - Summary: request summary with count, slowest, fastest, average etc.
   - Response time: of different reequest in different timestamp
   - Latency distribution; infomation abuot the latency in different request
   - overall status: of OK and Unavailable stamp

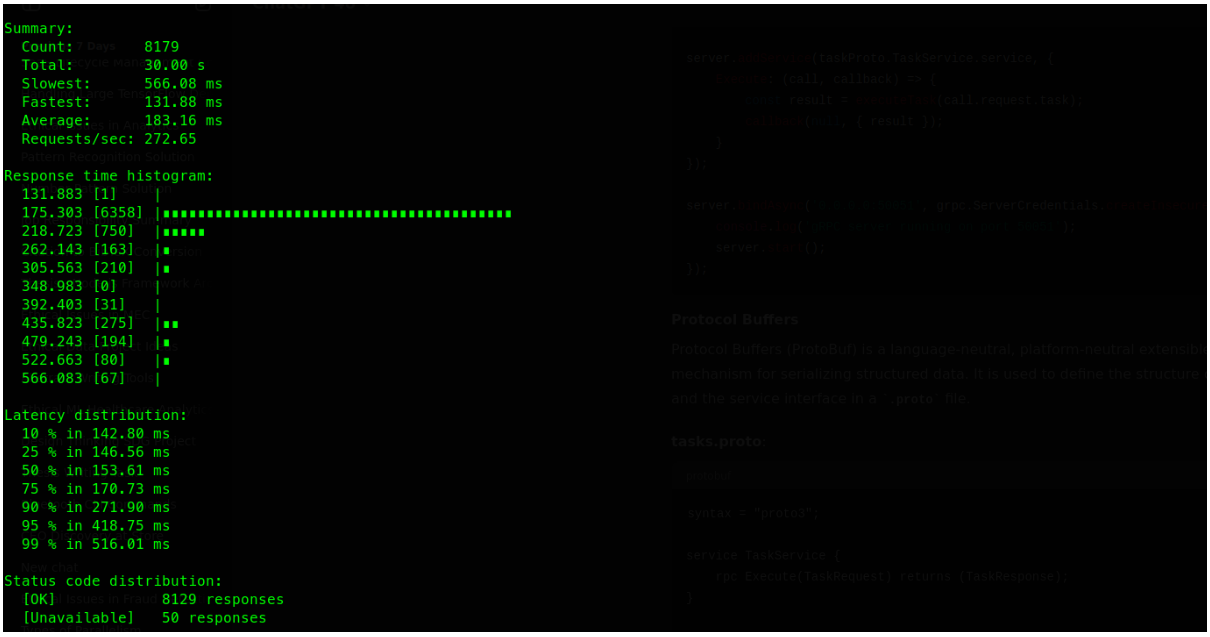

Figure 2: ghz benchmarking results structure