Configuration Manual for Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems

MSc Research Project
Cloud Computing

Olajide Olawuyi
Student ID: x23214619

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Olajide Olawuyi		
Student ID:	X23214619		
Programme:	Msc Cloud Computing	**Year:**	2024/2025
Module:	**Research Project**		
Lecturer:			
Submission Due Date:	13th December, 2024		
Project Title:	Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems		
Word Count:	6,127	**Page Count:**	

Signature: [illegible]

Date: 10th December, 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 for Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems

Olajide Olawuyi
Student ID: x23214619

1 Introduction

This documentation is aimed at creating a comprehensive manual to demonstrate all research project implementation setup for carrying out this research on "Performance Optimization and Scalability of Cloud-Based Data Processing Pipelines for Fire Emergency Response Systems". This research is primarily focused on optimizing the performance (efficiency and scalability) of a cloud based data processing pipeline design and built for fire emergency response system. To archive the set objectives of research, three system components were developed namely IoT-enabled system for processing and transmitting IoT data from installation site to the cloud platform, the machine learning component for processing and analysing sensor data and detecting fire signals and the monitor web application for visualizing the state of all IoT-enabled systems. Together, these systems components are couple as Fire Emergency Response System (FERS).

The IoT-enabled system is a localized IoT data streaming system used to stream read sensor data generated from the installation site and propagate the data to the cloud via EMQX publicly available MQTT broker. The stream IoT data is further processed and analysed by the machine learning component to label the data as fire or non-fire signal visualized from the monitor web application.

The machine component is composed of four (4) trained supervised machine learning models namely Decision Tree (DT) model, Logistic Regression (LR) model, Random Forest (RF) model and Support Vector Machine (SVM) model. These model used to classify or label streamed IoT data as fire and non-fire signal.

The monitor web application is used to monitor all configure and install IoT-enabled system and tracking the state or label of the signal being transmitted in real-time.

The rest of this documentation is organized as follows: section 2 will discuss the hardware specification and the software specification of the computer device used to carrying out the research implementation. Section 3 will discuss the software, programming languages libraries and their installation to create the implementation setup for carrying out research. Section 4 discusses the implementation of the system components namely the IoT-enables system, machine learning component and the monitor application. Finally, section 5 will discuss the concluding statement of this document.

2 Hardware and Software System Specification

This section will discuss implementation system configuration with respect to hardware and software resources.

2.1 Hardware Configuration

Three (3) different hardware machine were configured to implement the different system component implemented to archive the set objectives this study. The MacBook Pro machine was the primary development machine hosting the virtual machine and for build the machine learning models used for analysis the IoT sensor data, Ubuntu 22.04 Desktop Virtual Machine used to implement and setup the IoT-enabled system and the AWS EC2 instance for hosting and deploying the Monitor web Application.

2.1.1 MacBook Pro

The primary hardware for used for the implementation research study is MacBook Pro and it is equipped with the following specifications

- 8GB of RAM
- 256GB of ROM
- Apple M2 Chip

2.1.2 Ubuntu 22.04 Desktop Virtual Machine

The virtual machine used to implement and run the IoT-enabled system and it is equipped with the following specifications

4GB of RAM

- 30GB of ROM
- 4 Core of CPU x86_64 architecture

2.1.3 AWS EC2 Instance

The AWS EC2 instance used to implement and run monitor web application and it is equipped with the following specifications

- 1GB of RAM
- 8GB of ROM
- Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz

2.2 Software Requirements

The software requirements for the solution proposed will be discussed with respect to the implementation machine (MacBook Pro), the virtual machine for the IoT-enabled system and the AWS EC2 for hosting the monitor web application.

2.2.1 MacBook Pro

The MacBook Pro machine is the primary device used in the development of most components of this research study. Mac OS Sequoia (version 15) is the primary software used carry out this research and other system application software includes

- Universal Turing machine (UTM) virtual machine - UTM employs Apple's Hypervisor virtualization framework to run ARM64 operating systems on Apple Silicon at near native speeds. This virtual machine is used to power the Ubuntu 22.04 Desktop used to implement the IoT-enabled System.
- Ubuntu 22.04.5 Desktop LTS ISO File – This is the installation file used to install Ubuntu on the UMT virtual machine.
- Git – This is a distributed version control system that versions of files. It is used to maintain the different source code used to implement the system components and backup the source all through the development period.
- Anaconda – This is an open source data science and artificial intelligence distribution platform for Python and R programming language.
- Python – Python is a high-level general-purpose programming language used in this instance to build the machine learning component of this project.
- Jupyter Notebook – is a project to develop open-source software, web-based Python IDE, ideal for coding, visualization, and documentation in data science and education.

2.2.2 IoT-Enabled System

The IoT-enabled system is implemented in an Ubuntu 22.04 install in Universal Turing machine virtual host running MacBook Pro. The below are a list of system software and application software used to setting up IoT-enabled system

- Ubuntu 22.04 Desktop OS – This is the underlying system software used to implement the IoT-enabled system. All others application software listed were install and executed.
- Contiki-NG - This is a specialized operating system designed for Internet of Things (IoT) devices with limited resources. It features a standards-compliant, low-power IPv6 communication stack that facilitates Internet connectivity.
- C programming – This programming language is extensively used to develop and compile applications and system components. It allows for developing to write efficient, low-level code for resource-constrained IoT devices while leveraging the modularity and flexibility of Contiki-NG.
- Visual Studio Code - This is a code editor created by Microsoft that offers features like debugging, syntax highlighting, smart code completion, snippets, code refactoring, and integrated Git version control.
- Mosquitto MQTT Broker - is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.
- Mosquitto MQTT Client - It enables devices and applications to publish and subscribe to messages over a network using MQTT protocol, making it ideal for Internet of Things (IoT) use cases.

2.2.3 AWS EC2 Instance

The AWS EC2 instance in the cloud was provision using Ubuntu 24.04 server and run the monitor web application. The below are a list of system software and application software used to setting up IoT-enabled system

- Ubuntu 24.04 Server OS - This is the underlying system software used develop and run the monitor web application. All others application software listed were install and executed.
- Python – The web application was developed in Python programming language using fastapi libraries.

- JavaScript, HTML and CSS – These technologies are used to develop the frontend component of the web applications.

3 Compiling and Running the IoT-Enabled System

The IoT-enabled system components such MQTT client application and IoT Data Controller are develop using Contiki-NG OS and the c programming and as such will require compilation and execution of compile code. This section provide instruction used to compile and run the application components.

IoT-enabled system source code is contained ***fire-emergency-response-system*** directory. The directory contains 3 other directories namely ***application_ui***, ***data_io*** and ***mqtt-client***. The application_ui directory contains IoT Data Controller application source code and mqtt-client directory contains MQTT client application source code.

3.1 Compilation instruction for MQTT Client Application

To compile MQTT client application source code, navigate into ***mqtt-client*** directory and open the terminal point to the current directory. On the opened terminal run

```
make clean
```

The make clean command is used to remove previously compiled code from the build directory which contains all compilation resource use to support file need to run the compiled MQTT client application.

To compile MQTT client application, run

```
make TARGET=native
```

This command will create a build directory in the current directory and generate mqtt-client.native file and all supporting files and resource need to execute mqtt-client application stored in the build folder.

3.2 Compilation instruction for IoT Data Controller Application

To compile IoT Data controller application source code, navigate into ***application_ui*** directory and open the terminal point to the current directory. The ***application_ui*** contains

application-ui.c source code for the IoT Data Controller UI application. The ***application-ui.c*** source code depends on the source code in the ***data_io*** directory. To compile application-ui.c code, a linker between the application-ui.c source and the source code in the data_io directory. To begin compilation, on the opened terminal run

```
gcc -I../data_io -c ../data_io/cJSON.c -o cJSON.o
```

This command compiles the source file ***cJSON.c*** in the ***data_io*** directory into an object file ***cJSON.o***. The compiler looks for header files in the ***data_io*** directory and produces an intermediate file ***cJSON.o*** that contains the compiled machine code for ***cJSON.c***. This file can later be linked with other object files to create an executable.

```
gcc -I../data_io -c ../data_io/data-processor.c -o data-processor.o
```

This command compiles the source file ***data-processor.c*** in the ***data_io*** directory into an object file ***data-processor.o***. The compiler looks for header files in the ***data_io*** directory and produces an intermediate file ***data-processor.o*** that contains the compiled machine code for ***data-processor.c***. This file can later be linked with other object files to create an executable.

```
gcc `pkg-config --cflags gtk+-3.0` -I../data_io -c application-ui.c -o
application-ui.o
```

This command uses ***pkg-config*** to load compiler flags for including the GTK+ 3.0 library headers. Adds ***data_io*** directory for custom headers and compiles the ***application-ui.c*** source file into an object file with the name ***application-ui.o***. The resulting object file can be linked with other object to create an executable.

```
gcc application-ui.o data-processor.o cJSON.o `pkg-config --libs gtk+-
3.0` -o my_application
```

This command links the compiled object files (***application-ui.o***, ***data-processor.o***, ***cJSON.o***) with the GTK+ 3.0 libraries to produce an executable named ***my_application***. It ensures that all required symbols (functions, variables, etc.) are resolved from both the object files and the

GTK+ 3.0 library. The resulting executable (***my_application***) can now be run, and it will likely use GTK+ for its graphical user interface.

3.3 Run IoT-Enabled System

To run the IoT-enabled system, several different services and applications must be executed concurrently. The following actions or steps must the execute serially

Open a new terminal and run the following commands

```
sudo kill $(sudo lsof –t –i:1883)
```

This is used to terminate any process that is running on the network port 1883. This is important so that the mosquito broker can use the port.

```
sudo systemctl stop mosquito
```

This command stops the Mosquitto MQTT broker if it is running. The service is no longer active, and any applications or devices attempting to connect to it will fail until the service is restarted.

```
sudo mosquitto  -c /etc/mosquitto/mosquitto.conf
```

This command is used run Mosquitto broker in the foreground (if not run as a service). It uses the configuration settings defined in /etc/mosquitto/mosquitto.conf. The broker will be ready to handle MQTT client connections and publish/subscribe operations.

On a second terminal, run

```
mosquitto_sub -h localhost -p 1883 -t "Local/FERS/IoT-Data/json" -u
"fers_user_account" -P "0000000000"
```

The mosquitto_sub command connects to the MQTT broker running locally on localhost:1883. It subscribes to the topic "Local/FERS/IoT-Data/json". It uses the specified username ***fers_user_account*** and password ***0000000000*** for authentication. Once connected, the command waits and outputs any messages published to the specified topic in real-time.

On a third terminal, navigate to ***mqtt-client*** directory using the ***cd*** command. To run the MQTT client application, run

```
sudo ./build/native/mqtt-client.native
```

This is used to run the MQTT client application from the Contiki-NG OS that published IoT sensor data periodically every 5 seconds.

Finnally, on a fourth terminal, navigate to ***application_ui*** directory using the ***cd*** command. To run the IoT Data Controller application, run

```
gcc -I../data_io -c ../data_io/cJSON.c -o cJSON.o && gcc -I../data_io -c
../data_io/data-processor.c -o data-processor.o && gcc pkg-config --cflags
gtk+-3.0 -I../data_io -c application-ui.c -o application-ui.o && gcc
application-ui.o data-processor.o cJSON.o pkg-config --libs gtk+-3.0 -o
my_application && ./my_application
```

The provided command sequence compiles and links a C program consisting of three source files: cJSON.c (for JSON handling), data-processor.c (for data processing), and application-ui.c (for the graphical user interface). Each file is compiled into an object file (cJSON.o, data-processor.o, and application-ui.o) with necessary include paths and GTK+ 3.0 compiler flags. The object files are then linked together with GTK+ 3.0 libraries to create an executable named my_application. Finally, the executable is run, launching the program, which likely includes a graphical user interface and processes JSON data using the compiled functionality. This workflow demonstrates a typical modular C development process with external library integration.

3.4 Real Time Monitor Web Application

The monitor web application is used to monitor the state of all active IoT-enabled system (also known as IoT node) that is publishing data in real time. The monitor web application can be accessed live on

http://34.242.232.105:8000/

4 Conclusion

This report has provided a detailed description of the implementation of the essential components that makes the Fire Emergency Response System (FERS). The system was made up of three subsystems: the IoT-enabled system, the machine learning component, and the monitoring web application. Detailed explanations, accompanied by code snippets on how the system can be setup and operated.

The outcomes of this study are repeatable, providing a baseline for researchers who wish to reproduce the system using the methods outline in this report. By following these instructions, subsequent implementations can produce consistent and reliable results. This report serves as a comprehensive guide for setting up and operate the scalable and efficient fire emergency response systems.

.

References

Contiki-NG Documentation — Contiki-NG documentation (no date). https://docs.contiki-ng.org/en/develop/.

Osy (no date) *UTM*. https://mac.getutm.app/.

Wikipedia contributors (2024a) *Anaconda (Python distribution)*. https://en.wikipedia.org/wiki/Anaconda_(Python_distribution).

Wikipedia contributors (2024b) *Git*. https://en.wikipedia.org/wiki/Git.

Wikipedia contributors (2024c) *Project Jupyter*. https://en.wikipedia.org/wiki/Project_Jupyter.

Wikipedia contributors (2024d) *Python (programming language)*. https://en.wikipedia.org/wiki/Python_(programming_language).

Wikipedia contributors (2024e) *Visual Studio Code*. https://en.wikipedia.org/wiki/Visual_Studio_Code.