National College of Ireland

Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring

MSc Research Project
MSCCLOUD1-A

Harsh Gavhane
Student ID: X23277653

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Harsh Gavhane
Student ID:	X23277653
Programme:	MSCCLOUD1-A
Year:	2025
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	11/08/2025
Project Title:	Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring
Word Count:	773
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:	15th September 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):	

Context-Aware Fuzzy Logic Dispatcher for Real-Time IoT-Based Disaster Monitoring

Harsh Gavhane
X23277653

1 Introduction

The configuration manual helps in step-by-step instruction of implementing an IoT-based disaster event monitoring system that can be constructed with the aid of a hybrid Edge-FogCloud architecture. The system makes use of fuzzy logic in the categorization of data (sensors) as either low, medium or high priority. The high-priority events are executed through **AWS Lambda** and **SNS**; the middle-priority are directed towards **Raspberry Pi 3B+**; the low-priority ones are stored in **AWS DynamoDB**; real-time communication is operated on through **MQTT**, and the main center of coordination is carried out on the **AWS EC2** instance. This guide entails all the hardware and software and configuration required to successfully deploy and test the system.

2 System Requirements

Here, the tools -software and hardware- that are used to develop the hybrid execution environment are explained. It combines cloud, fog, and edge computers so that real-time computational operations and context-aware event prioritization support disaster monitoring. Specifications, both on software and hardware, are given as follows.

2.1 Software Requirements

- **Python 3.10+**: With packages such as `paho-mqtt`, `boto3`, and `requests`.
- **Google Colab**: Used for simulating publisher data and visual debugging.
- **AWS CLI**: Command-line interface to interact with AWS services.
- **AWS Services Used**:
  - **Lambda**: To trigger alerts for high-priority events.
  - **DynamoDB**: To log low-priority event data.
  - **EC2**: Hosts the dispatcher logic.
  - **SNS**: Sends alert emails for high-priority events.
- **Raspberry Pi OS**: With Python 3.9+ and required libraries.
- **Mosquitto MQTT Broker**: For real-time data transmission.

- **Node.js and Express** (Optional): Used to create a REST endpoint on the Raspberry Pi.

2.2 Hardware Requirements

- **Laptop/PC (Development Machine)**: Minimum specifications include an Intel i5 processor, 8GB RAM, and 100GB SSD.
- **Raspberry Pi 3B+**: Used for Fog-layer processing of medium-priority events.
- **Internet Connectivity**: Required for MQTT communication and access to AWS services.

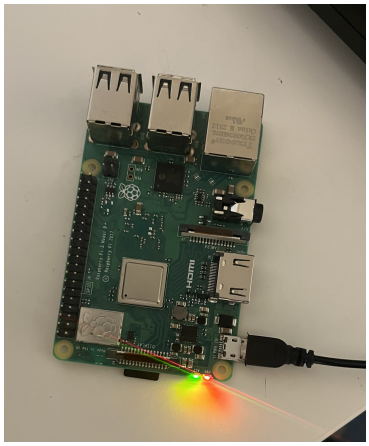

Figure 1: Raspberry Pi 3B+ Hardware Setup

3 System Configuration Steps

This section provides the description of configuration steps in preparing all the components related to the hybrid EdgeFogCloud architecture. The subsections give sequential instructions on how to deploy and integrate the essential modules EC2, MQTT, Lambda, Raspberry Pi, and DynamoDB.

3.1 EC2 Instance Configuration

- Launch an EC2 instance using Ubuntu 22.04 LTS.
- Install Python, pip, and required packages:

```
sudo apt update && sudo apt install python3-pip
pip3 install paho-mqtt boto3 requests
```

- Connect to SSH client.
- Set AWS credentials using:

```
nano ~/.aws/credentials
```

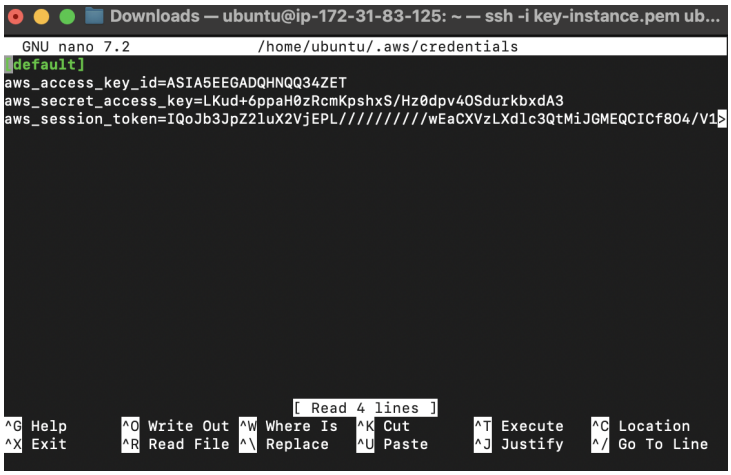

Figure 2: AWS EC2 Instance CLI Running Dispatcher

- Run Dispatcher code using:

  ```
  python3 Dispatcher.py
  ```

3.2 MQTT Broker Configuration

- Use the public MQTT broker: `broker.hivemq.com`.
- Publish and subscribe on topic: `iot/disaster/sensor`.
- Ensure port 1883 is open for communication.

3.3 Lambda Function Setup

- Create a Lambda function named `process_high_priority_event`.
- Use Python 3.12 runtime environment.
- Allow invocation via the EC2 dispatcher using SDK.

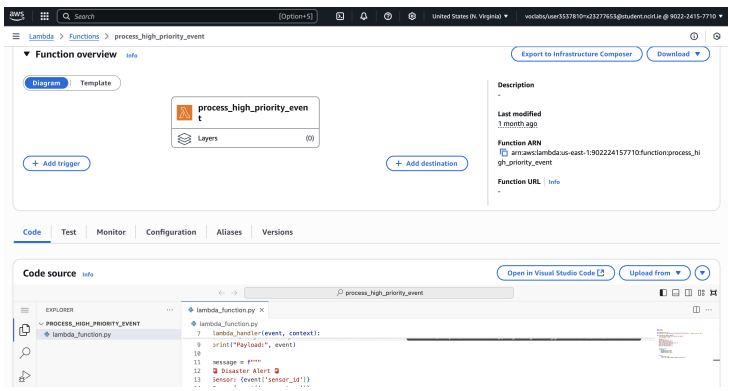

Figure 3: AWS Lambda Function Setup

3.4 Raspberry Pi Configuration

- Deploy a Flask-based REST API on Raspberry Pi at endpoint `/task`.
- Install required libraries using pip.
- Enable SSH for remote access.
- Run the server to handle medium-priority tasks.

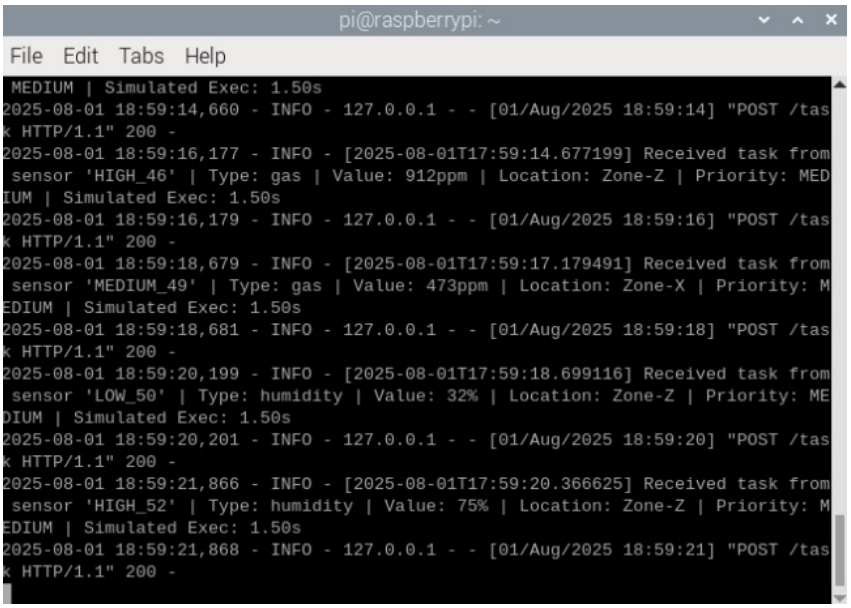

Figure 4: Raspberry Pi Console Receiving Event

3.5 DynamoDB Configuration

- Create a table named `SensorEvents`.
- Define partition key: `sensor_id`, and sort key: `timestamp`.

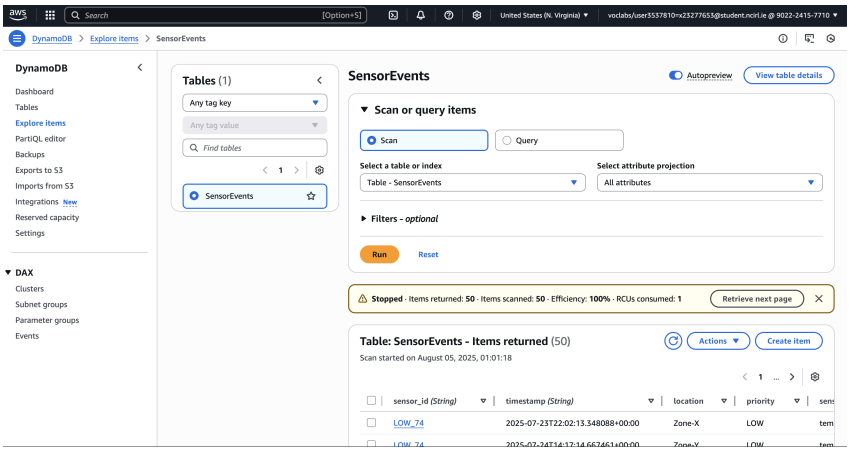

Figure 5: DynamoDB Table with Logged Sensor Data

4 Operational Workflow and Monitoring

In this section, the actual run-time operational characteristic of the configured system will be demonstrated with respect to a main point of interaction.

4.1 Live Dispatcher Output on EC2

After setup complete, dispatcher script is started on EC2 instance. It perpetually listens to MQTT incoming sensor messages, apply fuzzy logic for classification, and forwards them based on priority.

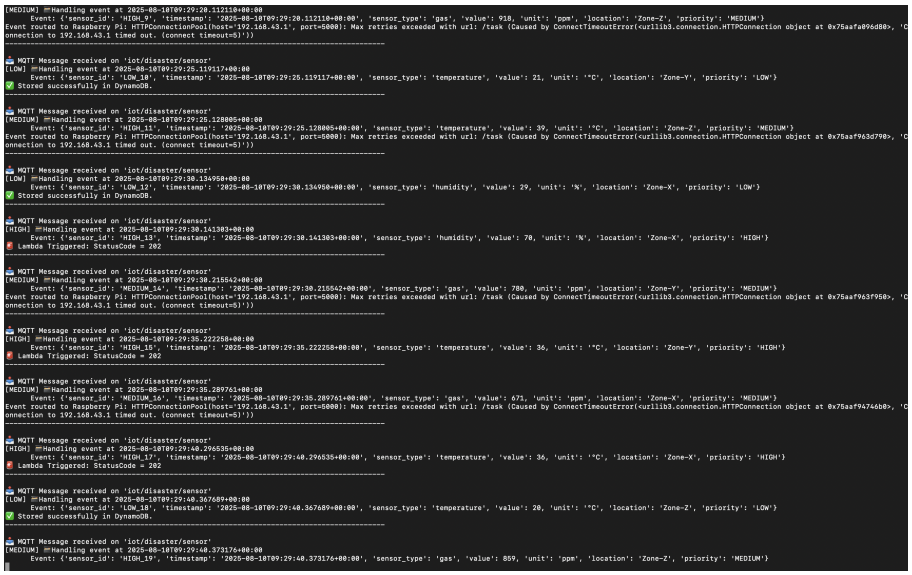

Figure 6: Live Dispatcher Console Output on EC2 Instance

4.2 Google Colab Publisher Execution

The publisher script operates with the help of Google Colab and imitates real-time sensors data of different types and positioning. Such events are published to the MQTT broker and accepted by the dispatcher to assign them to a specific class and direct them.

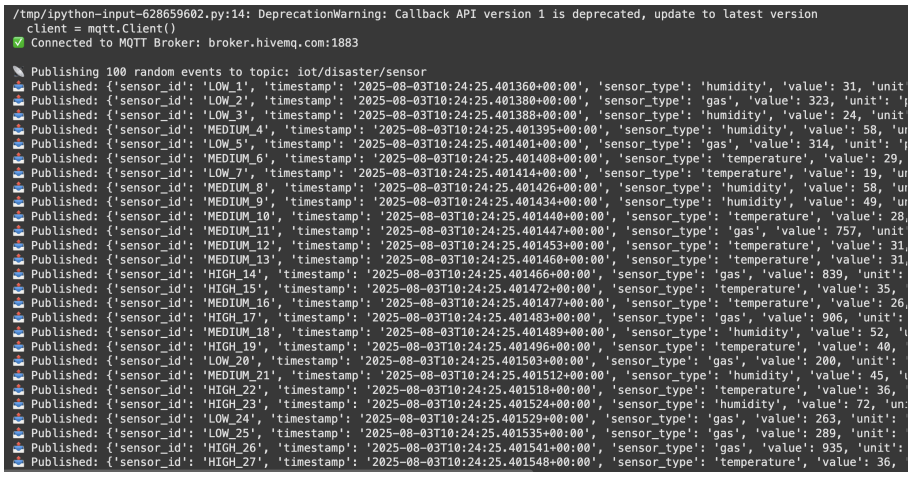

Figure 7: Google Colab Publisher Output Simulating Sensor Data

References

Amazon EC2 - Cloud Compute Capacity - AWS (n.d.).
URL: *https://aws.amazon.com/ec2/*

Download VNC Viewer by RealVNC® (n.d.).
URL: *https://www.realvnc.com/en/connect/download/viewer/*

Google Colab (n.d.).
URL: *https://colab.research.google.com/*

Launch AWS Academy Learner Lab (n.d.).
URL: *https://awsacademy.instructure.com/courses/123758/modules/items/11721528*

Ltd, R. P. (n.d.). Raspberry Pi software.
URL: *https://www.raspberrypi.com/software/*

Python 3.13 documentation (n.d.).
URL: *https://docs.python.org/3/*
.Ltd (n.d.) .*Amazon EC2 - Cloud Compute Capacity - AWS* (n.d.) .*Download VNC Viewer by RealVNC®* (n.d.) .*Google Colab* (n.d.) .*Launch AWS Academy Learner Lab* (n.d.) .*Python 3.13 documentation* (n.d.)