



Weather Machine – Mini Project

Jordan Gribben

CMP408: System Internals and Cybersecurity

BSc Ethical Hacking Year 4

2021/22

1 INTRODUCTION

1.1 BACKGROUND

The purpose of this project is to build a full-stack system that will display weather data on the Raspberry Pi Zero. This will be achieved by a city being entered that will then be transferred to the Pi via MQTT, once the Pi has received this communication it will receive weather data for that city using an online weather source, this data will then be displayed on the Pi. The security of this device will also be considered by having relevant security features.

1.2 OBJECTIVES

The objectives of this project are as follows:

- Create a webpage using an amazon web services (AWS) EC2 instance that allows users to input a city.
- Publish this data using an MQTT broker.
- Retrieve this data using the Raspberry Pi, get the weather data for the city retrieved and display it on the Pi.
- Use LEDs to show the current state of the Pi.
- Implement security features

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

As discussed in the background, this project's purpose is to take in a city and displaying that city's weather data. The interfaces designed for this project are purposefully simplistic, this is done for accessibility, the simplistic layout of the webpage helps users have a clear understanding of how to use the page.

Once a user has submitted a city from the webpage the weather data for that city will display on the Raspberry Pi, figure A shows the various processes used by this project to achieve this.

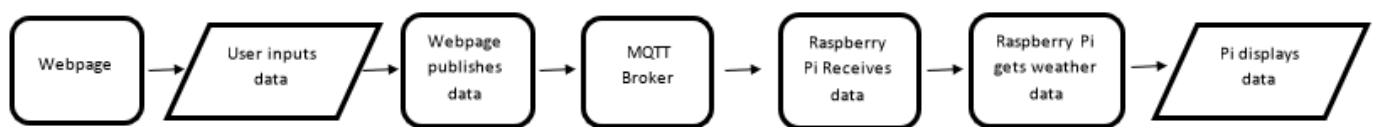


Figure A - Project Flow Diagram

The project can be split into two key sections these being the Raspberry Pi setup/hardware, and the AWS EC2 instance which acts as both the webpage and the MQTT broker. This report details the setup of the Raspberry Pi and the AWS EC2 instance so that the creation of this weather machine can be recreated.

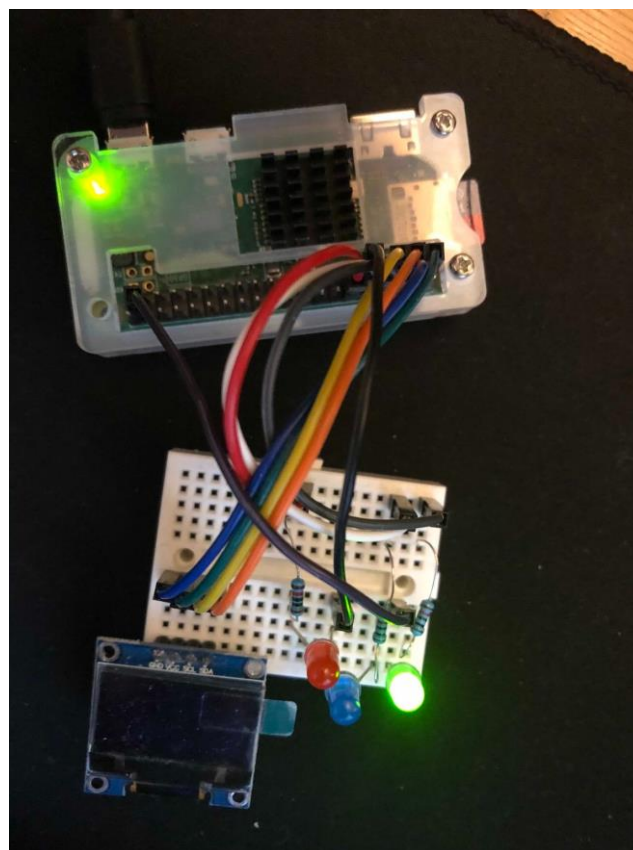
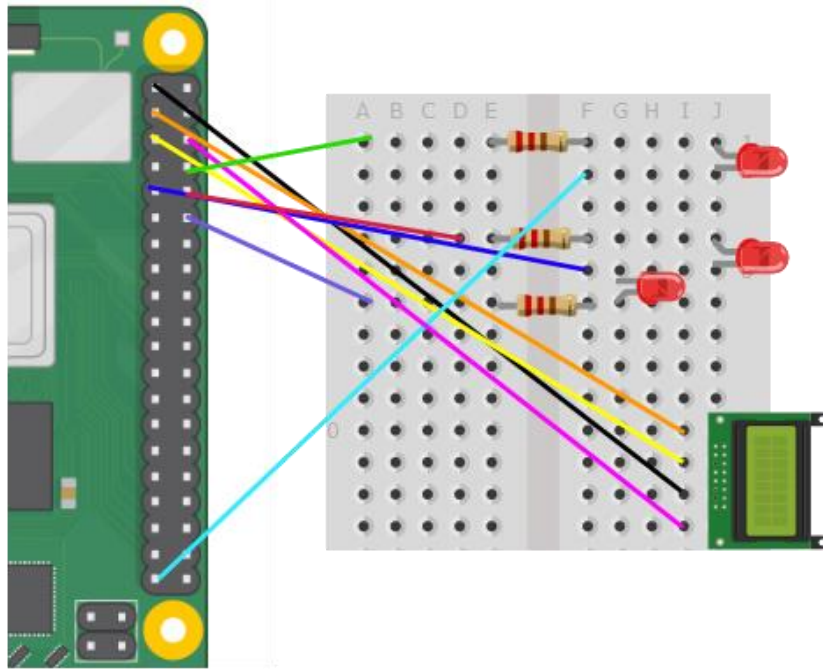
2.2 RASPBERRY PI ZERO

When setting up the Pi the first stage was to allow SSH connection, this can be done by adding a file named SSH to the boot location of the Pi. With SSH connection enabled, the Pi was able to be accessed by SSH on the IP 192.168.1.195, this allows for the Pi to be used without having to have it connected to a display (Raspberry Pi, 2021).

For this project the Pi was connected to various peripherals, these being 3 LEDs, and a 0.96 inch display. Figures B & C show a diagram of the setup as well as a picture of it set up. As shown in these figures, the LEDs are connected to pins 8, 10 and 12 being GPIO 14, 15, 18 respectively, each LED is connected to these pins via a resistor on the breadboard. Additionally, the short leg of each LED is connected to one of the Pi's ground connections with 2 of the LEDs sharing one ground cable.

The 0.96 inch display connects to the Pi with a 3v3 power going to the displays VCC and a ground cable going to the displays GND. Additionally, GPIO 2 and 3 are connected to the display utilizing the SDA and SCL GPIO pins of the pi, these are wired by connecting SDA to SDA and SCL to SCL.

Within the Python code the GPIO mode is set to BCM, due to this the pins are referenced by their GPIO numbers rather than physical pin number.



For the 0.96 inch display to be used Adafruit's python library was installed (Adafruit, 2021), this library allowed for objects to be displayed such as text. Additionally, the Python Pillow library was also installed to further make use of the display (Pillow, 2021).

A python script was created on the Pi, this script is used to subscribe to the MQTT server allowing the Pi to receive any information published on the AWS hosted webpage. Once a message has been received the Pi then uses OpenWeather (OpenWeather, 2021) to retrieve the weather data based off the city that has been entered, the weather data is then displayed on both the terminal and 0.96 inch display. This python script can be found in appendix A.

Using the Pi's GPIO pins 3 LED's have been connected, these are to show the various states of the weather machine. Green shows the Pi is waiting on data, blue shows the Pi is displaying data and Red shows that the city could not be found.

2.3 AMAZON EC2

An Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance was set up, this instance was running Linux Ubuntu. The instance was set up with a RSA key, this was done to ensure only those that have access to this key can SSH into the instance (Amazon, 2021), helping keep the instance secure. The PPK file for this key was downloaded allowing for an SSH connection to be made via PuTTY to the AWS instance.

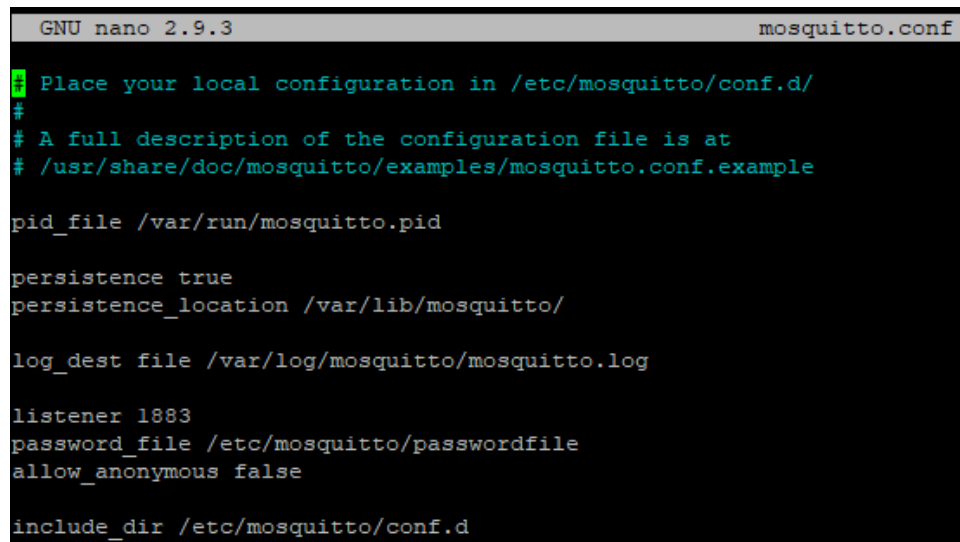
A new security group was created for this instance, this was to ensure only ports essential to this project were open. The open ports include, 22 for SSH connection, 80 for HTTP, and 1883 for MQTT, figure D shows the security group created for this instance.

Security group rule ID	Port range	Protocol	Source
sgr-0aaab73ecdb72610c	1883	TCP	0.0.0.0/0
sgr-0fbea3cefc6376a49	80	TCP	0.0.0.0/0
sgr-0e476e1b7294adbab	1883	TCP	::/0
sgr-01e95f49995d70a41	22	TCP	0.0.0.0/0

Figure D - AWS Instance Security Group

With the instance set up the MQTT broker Mosquitto was installed, Mosquitto was chosen as the broker due to it being lightweight, making it perfect for a simple IoT device (Eclipse, 2021). Mosquitto will allow for messages to be carried from the EC2 instance to the Pi, with the messages being published and the Pi subscribing to them. The broker was initially tested using MQTT explorer, this was to see if messages could successfully be published on the broker and received on the Pi before creating the webpage.

With the broker successfully able to publish and subscribe messages, usernames and passwords were added to the broker to help secure it. This was done by creating a username and password file with the following command “mosquitto_passwd -b passwordfile user password user”, this command created a user called “user” and prompted for a password to be created. With this file successfully created the contents of “mosquitto.conf” had to be updated to accommodate for this change (Eclipse, 2021). The updated content found within “mosquitto.conf” can be seen in figure E. Mosquitto was then restarted to ensure that the update to this file was registered, and further testing was done using MQTT explorer to ensure messages could only be published and subscribed too if the username and password was given.

A screenshot of a terminal window showing the configuration file /etc/mosquitto/mosquitto.conf being edited with GNU nano 2.9.3. The file contains several configuration parameters for the Mosquitto MQTT broker, including persistence settings, log file location, listener port, and authentication file path.

```
GNU nano 2.9.3 mosquitto.conf

# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

listener 1883
password_file /etc/mosquitto/passwordfile
allow_anonymous false

include_dir /etc/mosquitto/conf.d
```

Figure E - mosquitto.conf

A webpage was then set up on the EC2 instance using flask (Flask, 2021), this page was coded in python. This was chosen as it would allow for publishing to MQTT to be coded within the same file rather than having php enabled in html. This webpage can be seen in figure F.

Weather Machine Mini Project

Enter a City:

Figure F - Webpage

The webpage is simplistic by design, featuring the title for the project along with an input box prompting the user to enter a city. Once the user enters a city and presses submit the city the user entered is published via MQTT and if successfully publish the webpage will display ‘message published’. The Pi is then able to receive this message and display the appropriate data. The code for webpage can be seen in appendix B.

To further improve the security of this project input sanitization was added to the webpage using the “pybluemonday” python library (DevOps Tools, 2021). This is done when the city is inputted on the webpage to ensure that the user input is both safe for the Pi and webpage. By having input sanitization, it also helps protect against cross site scripting attacks. The code snippet in figure G shows the input sanitization.

```
city = strict.sanitize(request.form.get("city"));
```

Figure G - Input Sanitization

3 CONCLUSION

To conclude, this project turns a Raspberry Pi Zero into a basic weather machine meeting all the objectives set out in the introduction. The Pi successfully communicated with the broker on AWS to receive the data it needs. Additionally, the security aspect of this project is covered due to features such as RSA keys so only authorised parties can SSH into the EC2 instance, usernames and passwords on the MQTT broker and input sanitisation.

4 FUTURE WORK

If given more time or resources were available for this project the project additional work could be carried out. The following is potential future work for this project.

- Linux Kernel module (LKM) – A LKM could be added to control the LED's connected to the Pi. Allowing them to flash a certain colour if the Pi is retrieving data or a different colour if it cannot find a valid city.
- Database – A AWS database could be utilised, this could allow the Pi to published the weather data and have it stored on a database.
- Security Features - Additional security features could be added to make the project more secure such as running on HTTPS over HTTP.

5 REFERENCES

Adafruit, 2021. *Installing CircuitPython Libraries on Raspberry Pi*. [Online]
Available at: <https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi>

[Accessed 18 December 2021].

Amazon, 2021. *Amazon EC2 key pairs and Linux instances*. [Online]

Available at: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html>

[Accessed 19 December 2021].

DevOps Tools, 2021. *A library for sanitizing HTML very quickly via bluemonday*. [Online]

Available at: <https://pythonawesome.com/a-library-for-sanitizing-html-very-quickly-via-bluemonday/>

[Accessed 10 January 2022].

Eclipse, 2021. *Eclipse Mosquitto*. [Online]

Available at: <https://mosquitto.org/>

[Accessed 5 December 2021].

Eclipse, 2021. *mosquitto.conf man page*. [Online]

Available at: <https://mosquitto.org/man/mosquitto-conf-5.html>

[Accessed 8 January 2022].

Flask, 2021. *User's Guide*. [Online]

Available at: <https://flask.palletsprojects.com/en/2.0.x/>

[Accessed 28 December 2021].

OpenWeather, 2021. *Weather API*. [Online]

Available at: <https://openweathermap.org/api>

[Accessed 27 December 2021].

Pillow, 2021. *Pillow 9.0.0*. [Online]

Available at: <https://pypi.org/project/Pillow/>

[Accessed 18 December 2021].

Raspberry Pi, 2021. *Raspberry Pi Documentation*. [Online]

Available at: <https://www.raspberrypi.com/documentation/computers/configuration.html>

[Accessed 15 December 2021].

6 APPENDICES

APPENDIX A - RASPBERRY PI PYTHON SCRIPT

```
#import required modules
#Reference, paho
#based on: https://pypi.org/project/paho-mqtt/#client
import requests, json
import RPi.GPIO as GPIO
from time import sleep
import board
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306
import paho.mqtt.client as mqtt

GPIO.setmode(GPIO.BCM)
GPIO.setup(14, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)

broker = "18.233.152.78"

def Draw(text, textW, textH, textD):
    #Reference, Adafruit
    #based on: https://learn.adafruit.com/monochrome-oled-breakouts/python-usage-2
    # Define the Reset Pin
    oled_reset = digitalio.DigitalInOut(board.D4)

    # Change these
    # to the right size for your display!
    WIDTH = 128
    HEIGHT = 64
    BORDER = 5

    # Use for I2C.
    i2c = board.I2C()
    oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3C, reset=oled_reset)

    # Clear display.
    oled.fill(0)
    oled.show()

    font_size = 14

    # Create blank image for drawing.
```

```
# Make sure to create image with mode '1' for 1-bit color.  
image = Image.new("1", (oled.width, oled.height))
```

```
# Get drawing object to draw on image.  
draw = ImageDraw.Draw(image)  
# Load default font.  
font = ImageFont.load_default()
```

```
(font_width, font_height) = font.getsize(text)
```

```
draw.text(  
    (oled.width//2 - font_width//2, 0),  
    text,  
    font=font,  
    fill=255,  
)
```

```
(font_width, font_height) = font.getsize(textW)
```

```
draw.text(  
    (oled.width//2 - font_width//2, 0 + font_size),  
    textW,  
    font=font,  
    fill=255,  
)
```

```
(font_width, font_height) = font.getsize(textH)
```

```
draw.text(  
    (oled.width//2 - font_width//2, 0 + font_size*2),  
    textH,  
    font=font,  
    fill=255,  
)
```

```
(font_width, font_height) = font.getsize(textD)
```

```
draw.text(  
    (oled.width//2 - font_width//2, 0 + font_size*3),  
    textD,  
    font=font,  
    fill=255,  
)
```

```
# Display image  
oled.image(image)  
oled.show()  
GPIO.output(15, GPIO.HIGH)
```

```
sleep(1)
GPIO.output(15, GPIO.LOW)
sleep(1)
GPIO.output(15, GPIO.HIGH)
sleep(1)
GPIO.output(15, GPIO.LOW)
```

```
def Weather(client, userdata, message):
```

```
    GPIO.output(14, GPIO.LOW) #Once message recieved light off
```

```
    AWScity = str(message.payload.decode("utf-8","ignore"))
```

```
    #Reference, Weather API code
```

```
    #based on: https://www.geeksforgeeks.org/python-find-current-weather-of-anycity-using-openweathermap-api/
```

```
    # Enter your API key here
```

```
    api_key = "85540bda151e89fc35d48ac7d5e2056b"
```

```
    # base_url variable to store url
```

```
    base_url = "http://api.openweathermap.org/data/2.5/weather?"
```

```
    # Give city name
```

```
    city_name = AWScity
```

```
    # complete_url variable to store
```

```
    # complete url address
```

```
    complete_url = base_url + "appid=" + api_key + "&q=" + city_name
```

```
    # get method of requests module
```

```
    # return response object
```

```
    response = requests.get(complete_url)
```

```
    # json method of response object
```

```
    # convert json format data into
```

```
    # python format data
```

```
    x = response.json()
```

```
    # Now x contains list of nested dictionaries
```

```
    # Check the value of "cod" key is equal to
```

```
    # "404", means city is found otherwise,
```

```
    # city is not found
```

```
    if x["cod"] != "404":
```

```
        # store the value of "main"
```

```
        # key in variable y
```

```
        y = x["main"]
```

```

# store the value corresponding
# to the "temp" key of y
current_temperature = y["temp"]

# store the value corresponding
# to the "humidity" key of y
current_humidity = y["humidity"]

# store the value of "weather"
# key in variable z
z = x["weather"]

# store the value corresponding
# to the "description" key at
# the 0th index of z
weather_description = z[0]["description"]

current_temperature = round(current_temperature - 273.15, 2)

# print following values
print( city_name + " Weather" +
      "\n Temperature (in Celcius) = " +
          str(current_temperature) +
      "\n humidity (in percentage) = " +
          str(current_humidity) +
      "\n description = " +
          str(weather_description))

text = (city_name + " Weather" )
textW = ("Temperature: " + str(current_temperature))
textH = ("Humidity: " + str(current_humidity))
textD = (str(weather_description))

Draw(text, textW, textH, textD)

```

else:

```

print(" City Not Found ")
GPIO.output(18, GPIO.HIGH) #if city cannot be found flash red led
sleep(1)
GPIO.output(18, GPIO.LOW)
sleep(1)
GPIO.output(18, GPIO.HIGH)
sleep(1)
GPIO.output(18, GPIO.LOW)

```

```

GPIO.output(14, GPIO.HIGH)

```

```
client = mqtt.Client()

client.username_pw_set("user", password="test")

client.on_message=Weather

client.connect(broker)
client.subscribe("Citydata")
client.loop_forever()
```

APPENDIX B – WEBPAGE PYTHON SCRIPT

```
#Reference, paho
#based on: https://pypi.org/project/paho-mqtt/#client
from flask import Flask, request, render_template
from pybluemonday import StrictPolicy
import paho.mqtt.client as mqtt
import requests, json, sys, time, ssl

strict = StrictPolicy()

broker = "127.0.0.1"
broker_port = 1883

# The callback for when the client receives a CONN-ACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection and reconnect then
    subscriptions will be renewed.
    client.subscribe("Citydata")

def on_publish(client, userdata, mid):
    print("Message Published")

def send_to_MQTT(msg):
    client = mqtt.Client()
    client.username_pw_set("user", password="test")
    client.on_connect = on_connect
    client.on_publish = on_publish

    client.connect(broker, port=broker_port, keepalive=60)
    client.publish("Citydata", payload=msg, qos=0, retain=False)
    client.disconnect()

app = Flask(__name__, template_folder = "./")
```

```
@app.route("/", methods=["POST"])
def send():
    print(request.form)
    city = strict.sanitize(request.form.get("city"));
    print (city)
    if not city:
        return "Invalid input"
    send_to_MQTT(city)
    return (city + " Published")
@app.route("/", methods=["GET"])
def home():
    return render_template("index.html")
app.run(host="0.0.0.0", port=80)
```