KENNESAW STATE UNIVERSITY

CPE 4040: Data Collection and Analysis, Spring 2024

Laboratory Report #3

Raspberry Pi with Cloud MQTT Broker

Team Members: Anindita Deb and Damisi Kayode

Electrical and Computer Engineering

Kennesaw State University

Faculty: Dr. Jeffrey L Yiin

Date of Lab Session: February 12, 2024

I. Objective

The objective of the lab is to set up a cloud based MQTT Broker. This is followed by the implementation of a MQTT subscriber code in Python to connect to MQTT broker and subscriber to topics (in reference to the previous lab). Additionally, we will be remotely controlling the Raspberry Pi's digital output using a cloud platform.

II. Material List

Hardware:

Raspberry Pi (4)

Breadboard

LEDs

Resistors (330 ohm to 1k ohm)

(We chose to use a 330ohm resistor in our case)

Software:

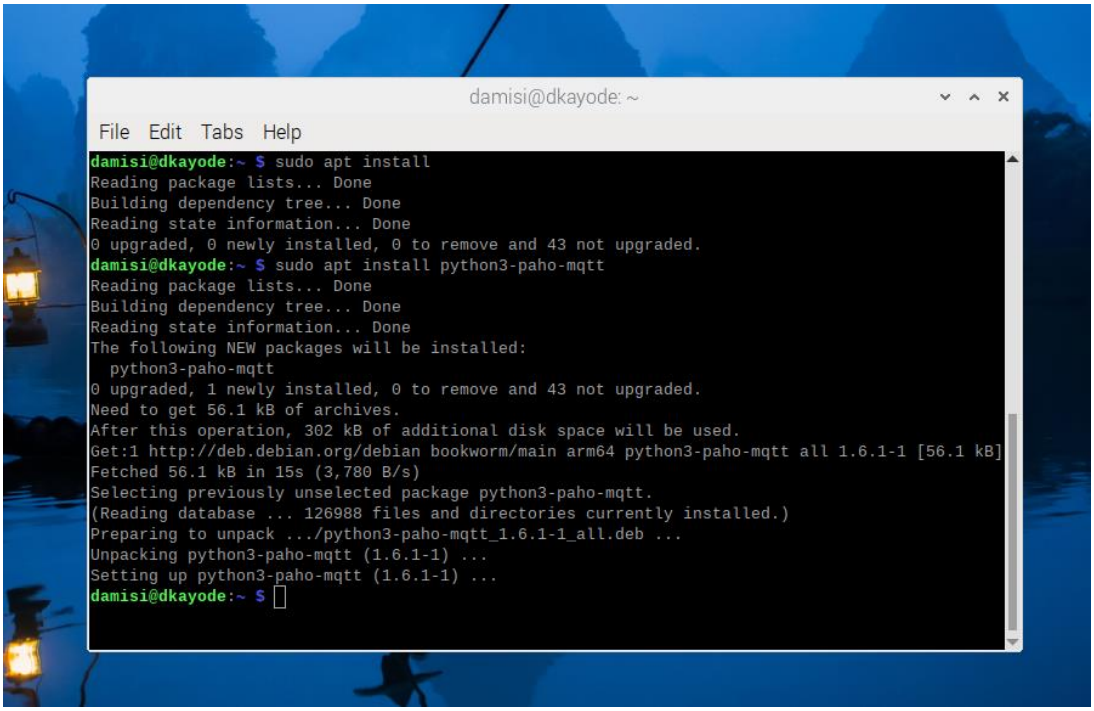
PC

Remote Desktop Connection

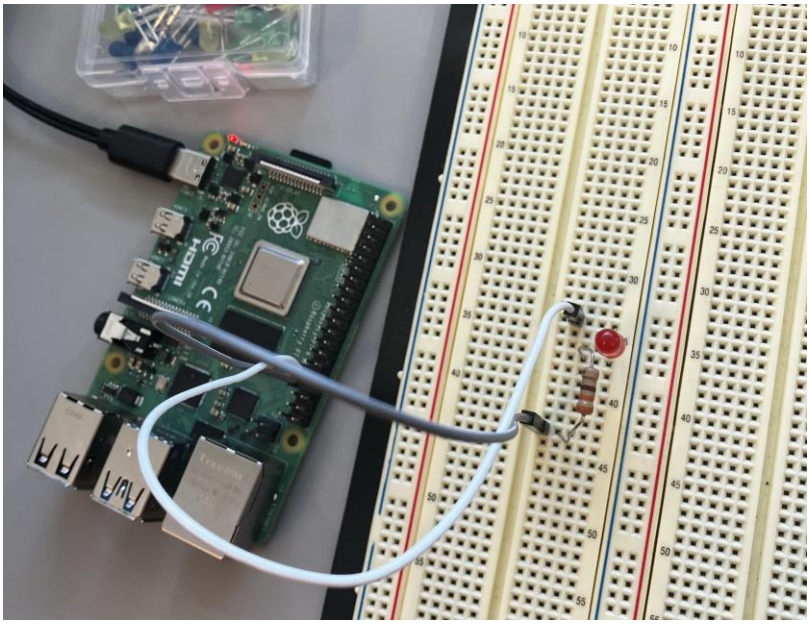
Paho MQTT

III. Lab Procedures and Results

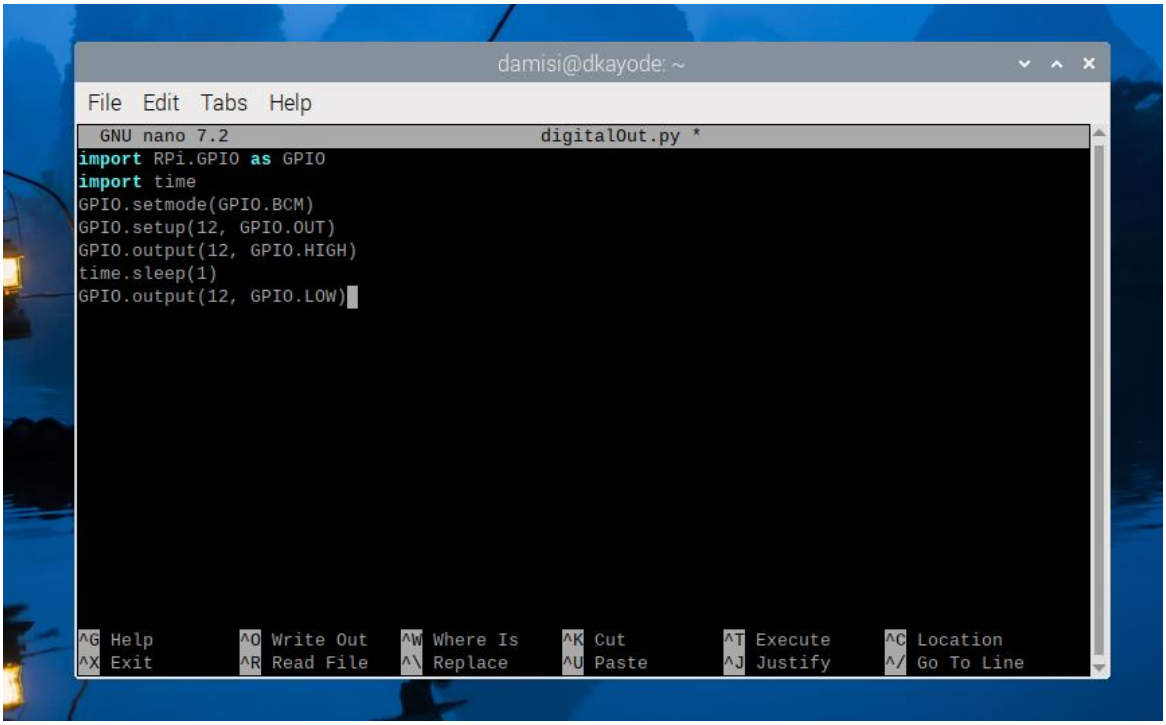
1. We power up our Raspberry Pi and connect to our laptop to the Pi through Remote Desktop Connection.
2. After login, install Paho-MQTT package in Python.



3. We connect an LED with a serial resistor 330 ohms to GPIO12 on the extension connector of your Raspberry Pi as shown in the lab procedure diagrams.



4. We create a file (digitalOut.py) with the python script below to test the control of the digital output.



Question: Please explain the script line-by-line, from GPIO.setmode(GPIO.BCM)and down? What does this code do to the LED?

GPIO.setmode(GPIO.BCM) tells us that we are referring to the Broadcom SOC channel numbers install of the physical board numbers.

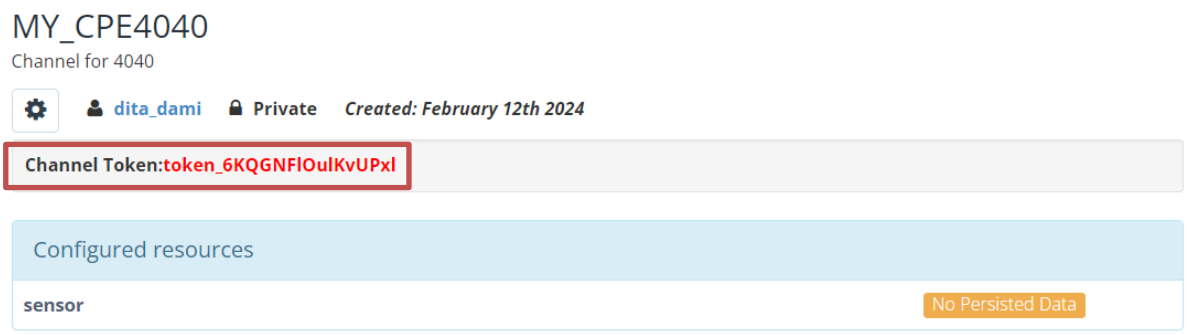
GPIO.setup(12,GPIO.OUT) line sets up pin 12 as an output pin indicating that we'll be sending signals from the Raspberry Pi to control an LED.

GPIO.output(12,GPIO.HIGH) turns the signal to this pin on so the LED in turn lights up.

Time.sleep(1) keeps the status of the board in that way for 1 second (so in this case the pin 12 LED stays on).

GPIO.output(12,GPIO.LOW) turns the signal to pin 12 off and in turn the LED turns off.

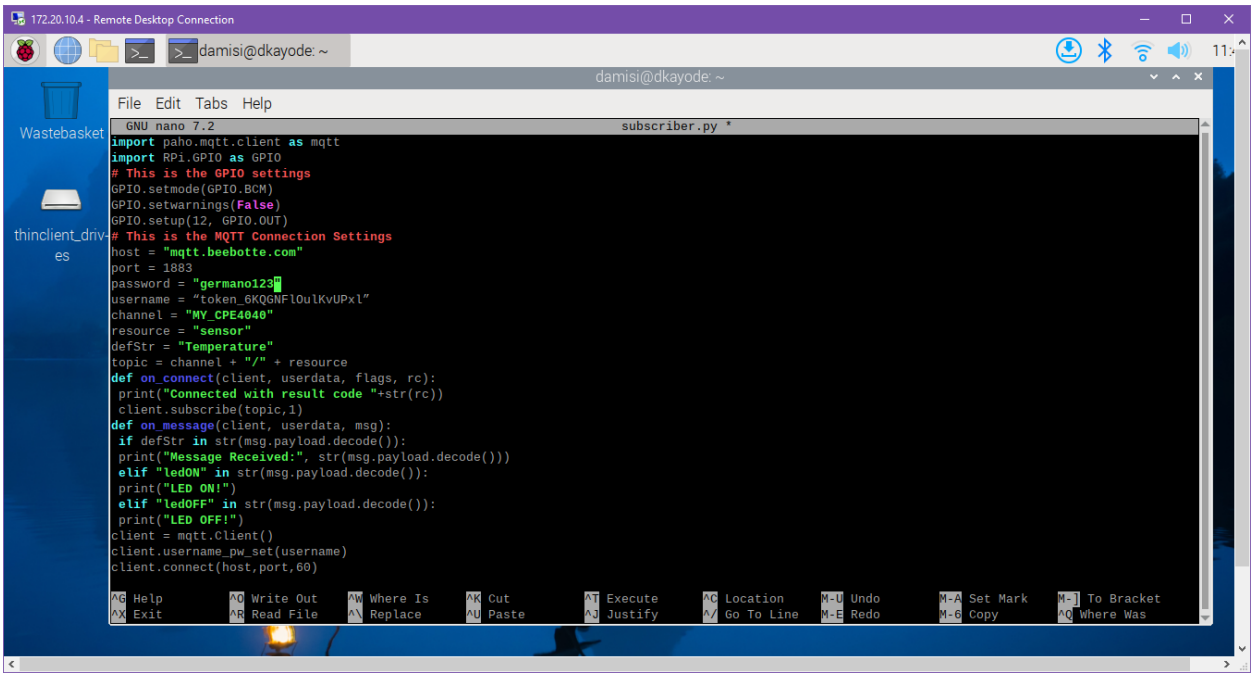
5. We go to www.beebotte.com cloud platform and create a free account.
6. After logging in, we go to “My Channels” and create a channel with the name of your choice (for example: My_CPE4040). For the resource name, we enter “sensor” and create the new channel.
7. We select the channel we just created and save the Channel Token for later use in our subscriber Python Code.



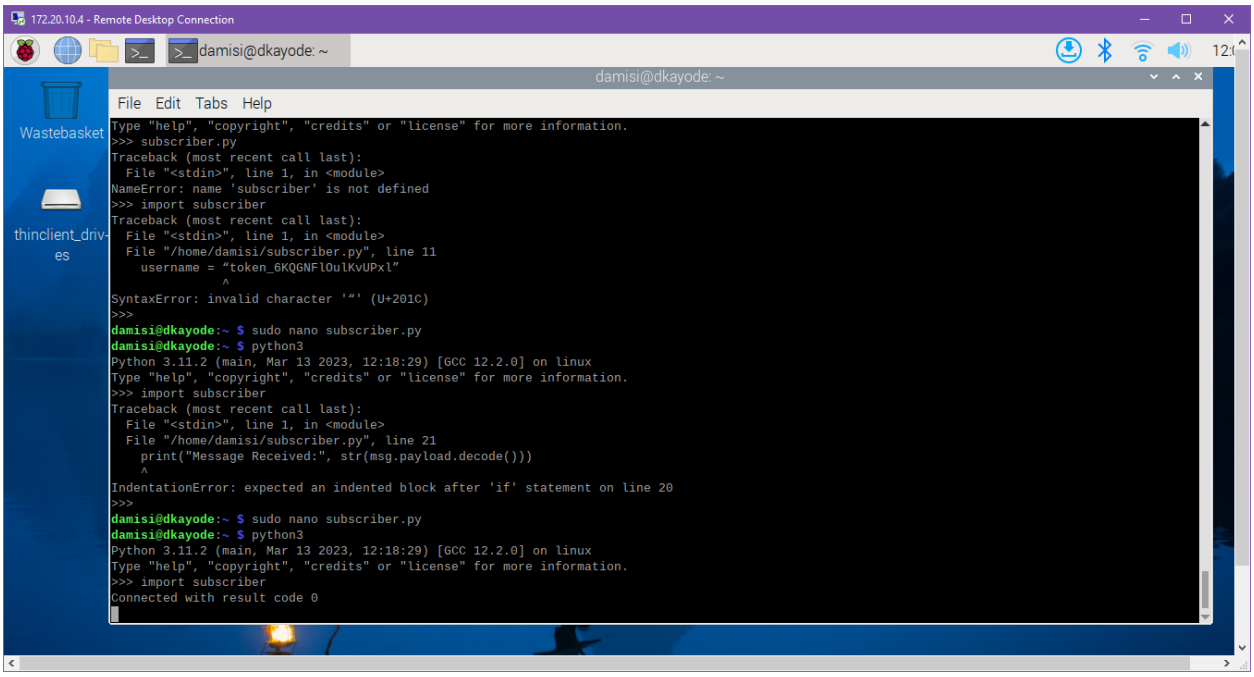
Question: What is Channel Token used for? It's a unique identified associated with a specific channel where data is being sent or received.

8. We make a new python program by entering the following codes:

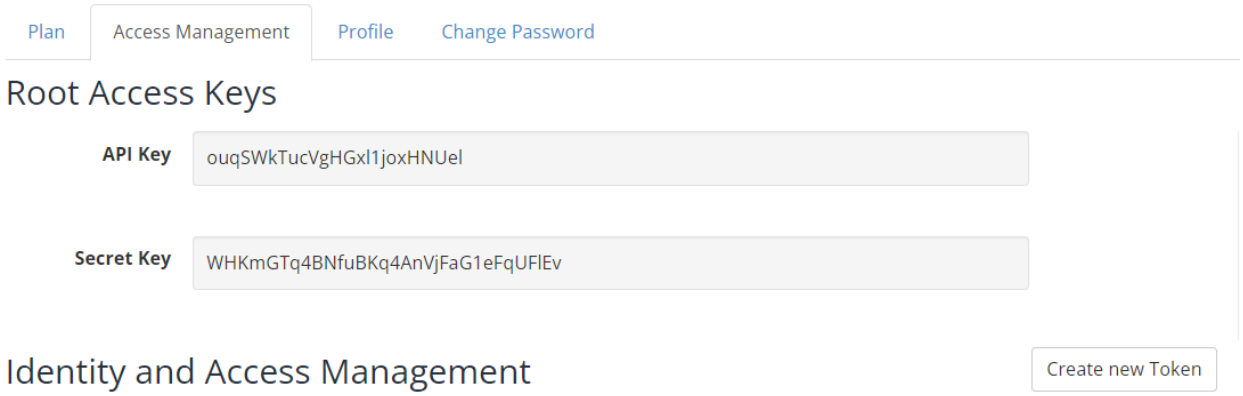
Sudo nano subscriber.py



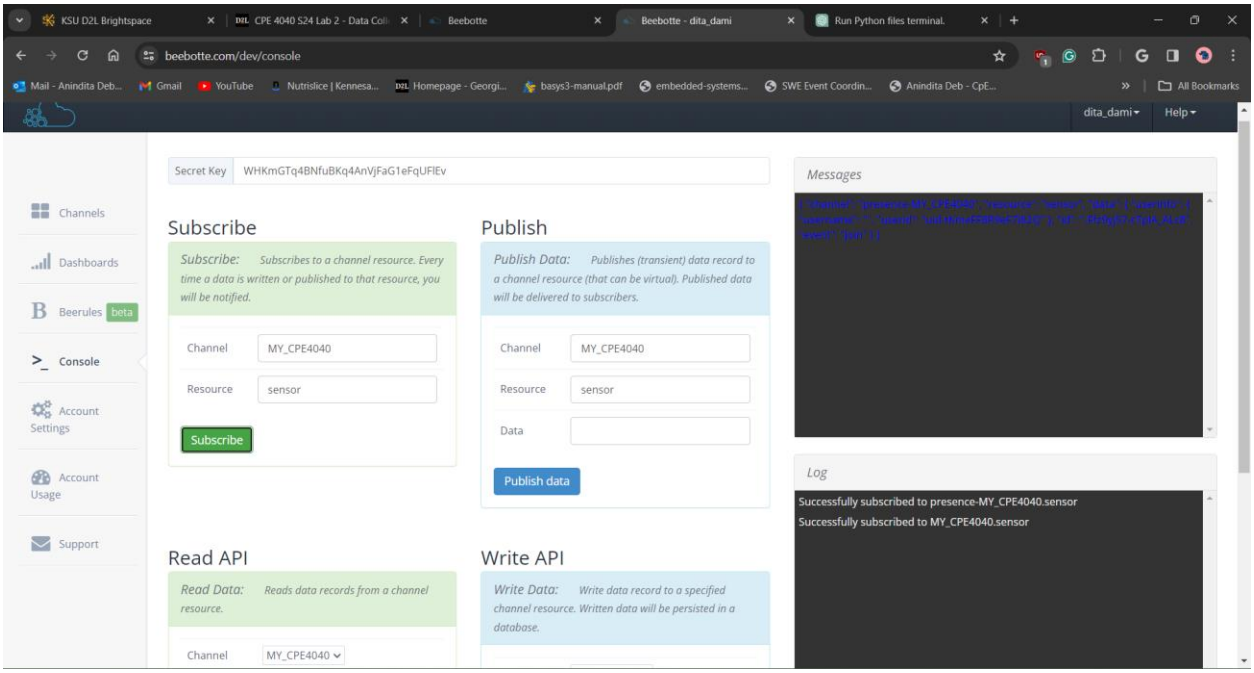
9. We run the code in a terminal using Python3 and if the connection is established a "connected" message with a result 0 should appear on screen.



10. In the Beebotte cloud platform, we go to the account settings menu and click the “access management” tab and make a copy of our secret key.

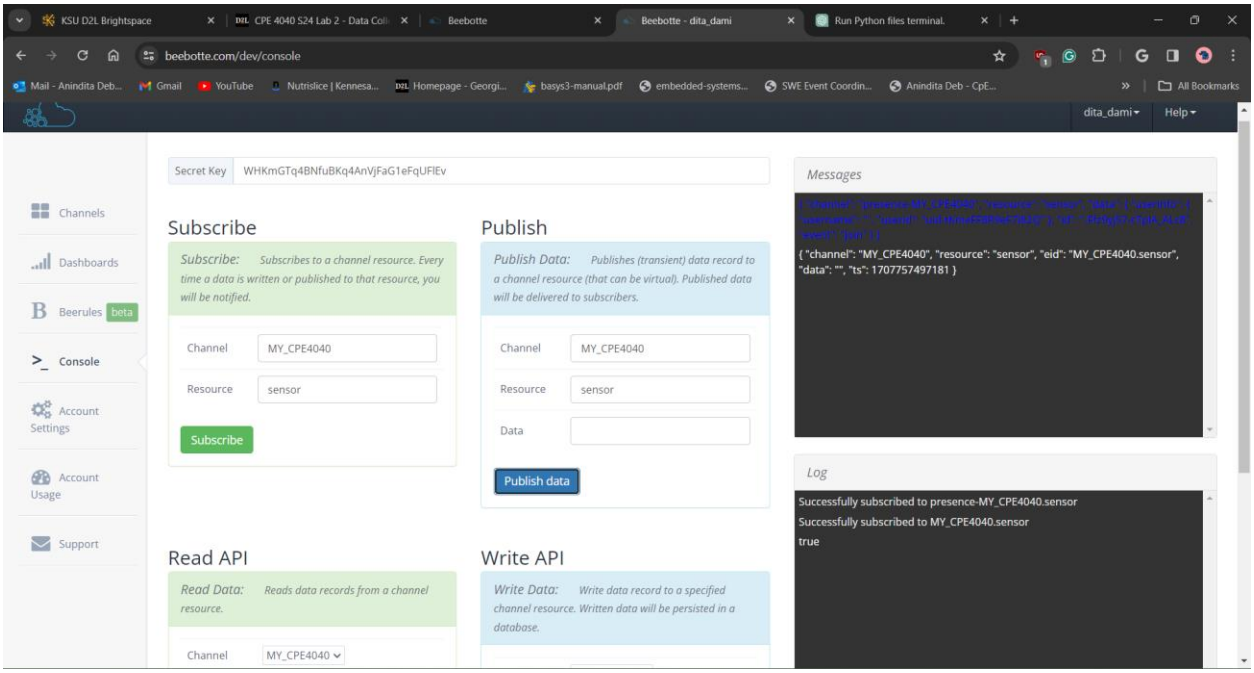


11. We go to console menu and paste our secret key into the box at the top. Now, we’re ready to test the MQTT subscriber and publish functions.



Question: Why is this key needed? Secret keys are used to ensure the connection is secure and prevents unauthorized access to the channel by the devices trying to receive or transmit data.

12. To test our channel in the Beebotte cloud, we fill out the channel name and the resource name in the subscribe section and click the “subscribe” button. We fill out the same information in the publish section with arbitrary data and click the “Publish” button.

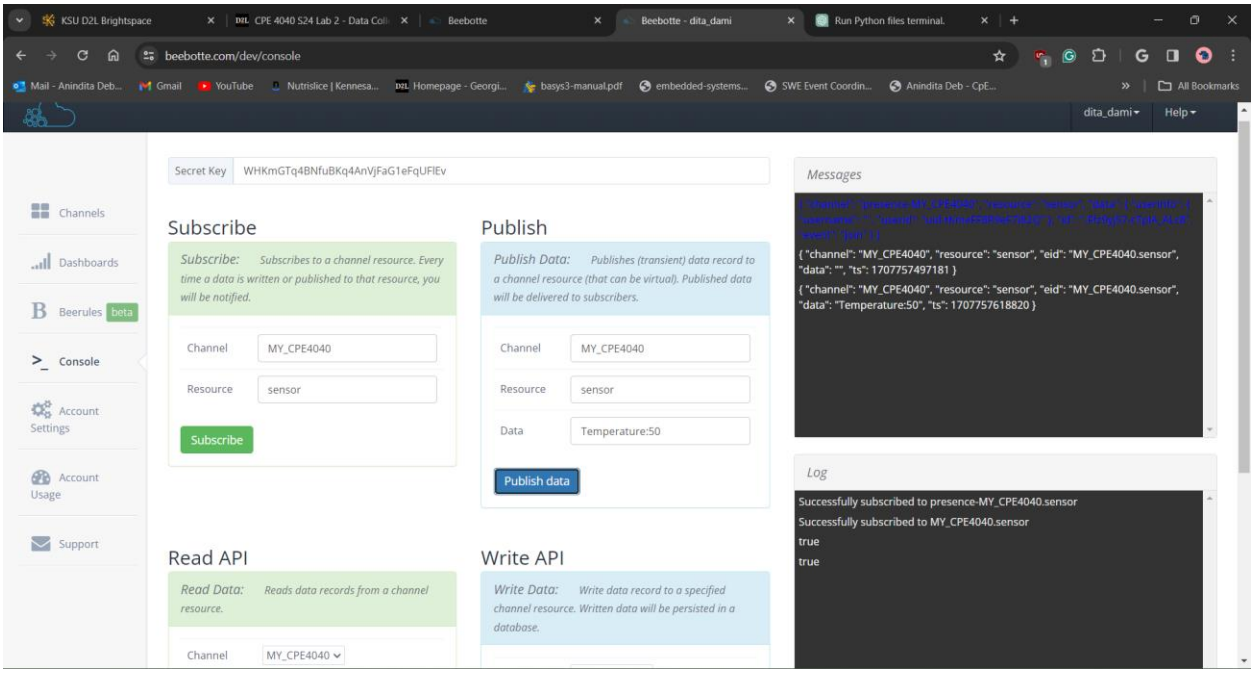


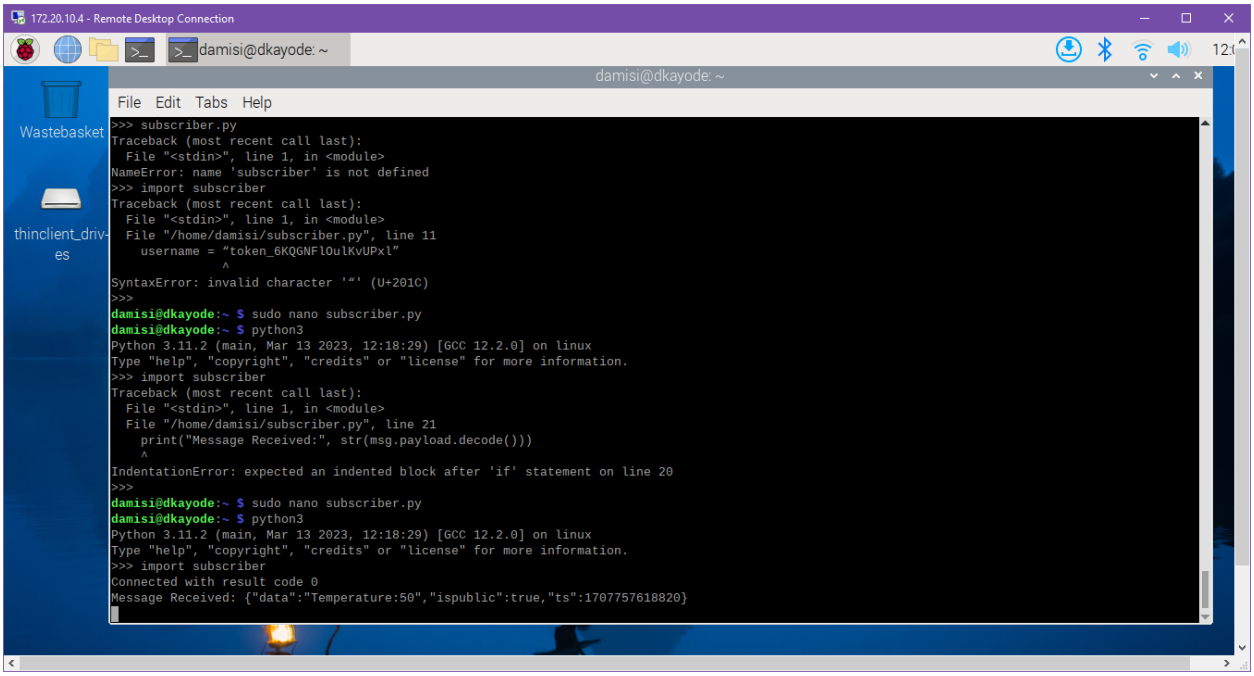
13. The subscriber.py code filters the messages starting with “Temperature”.

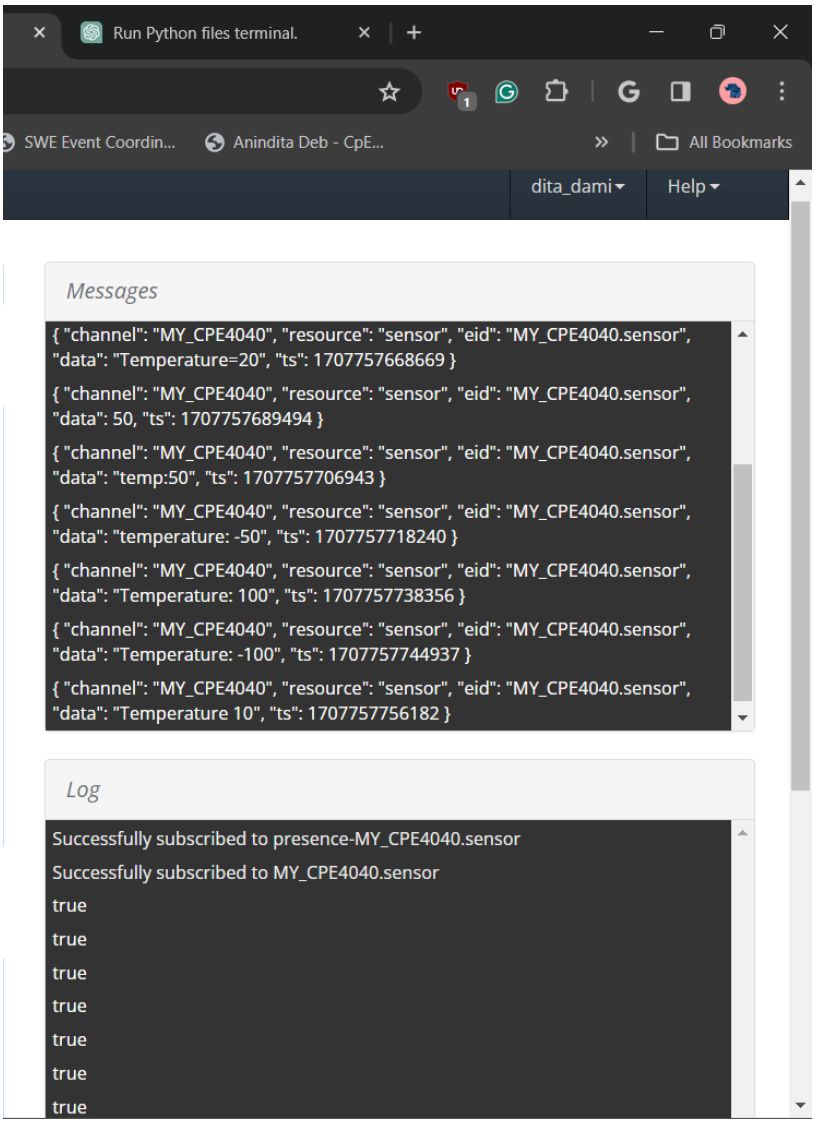
    Question: How do we know it? We can see the line of code from subscriber, but in addition to that, we can also observe the results when ‘Temperature’ is not spelled with a capital or isn’t written out in full that it doesn’t receive the message on the raspberry pi terminal.

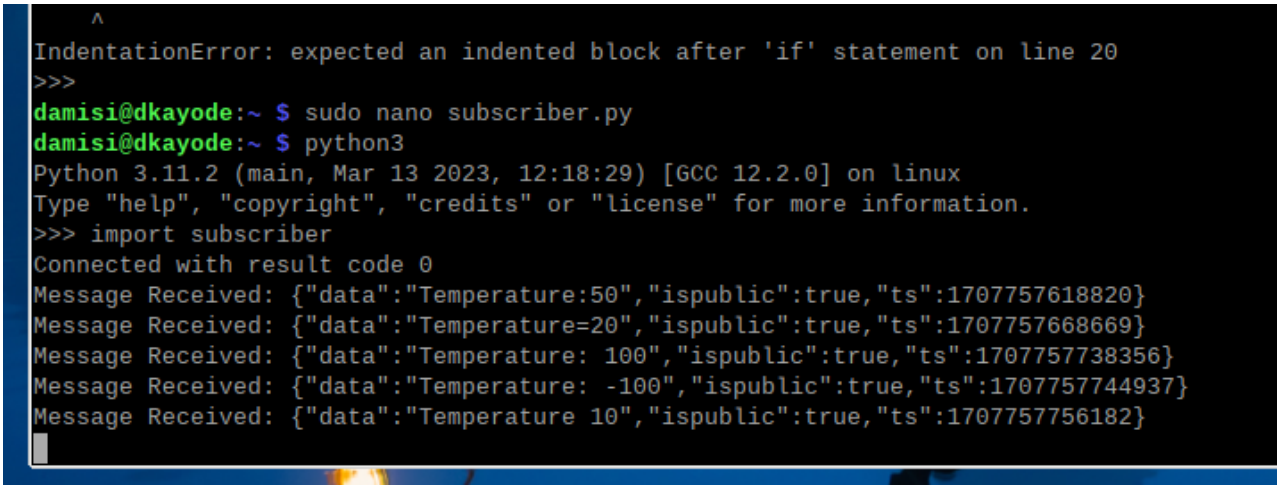
    Write an appropriate message (“Temperature: 50”) in the data field of the publish section and test to see it on the Raspberry Pi.

    We try 5 messages (with/without) proper format.

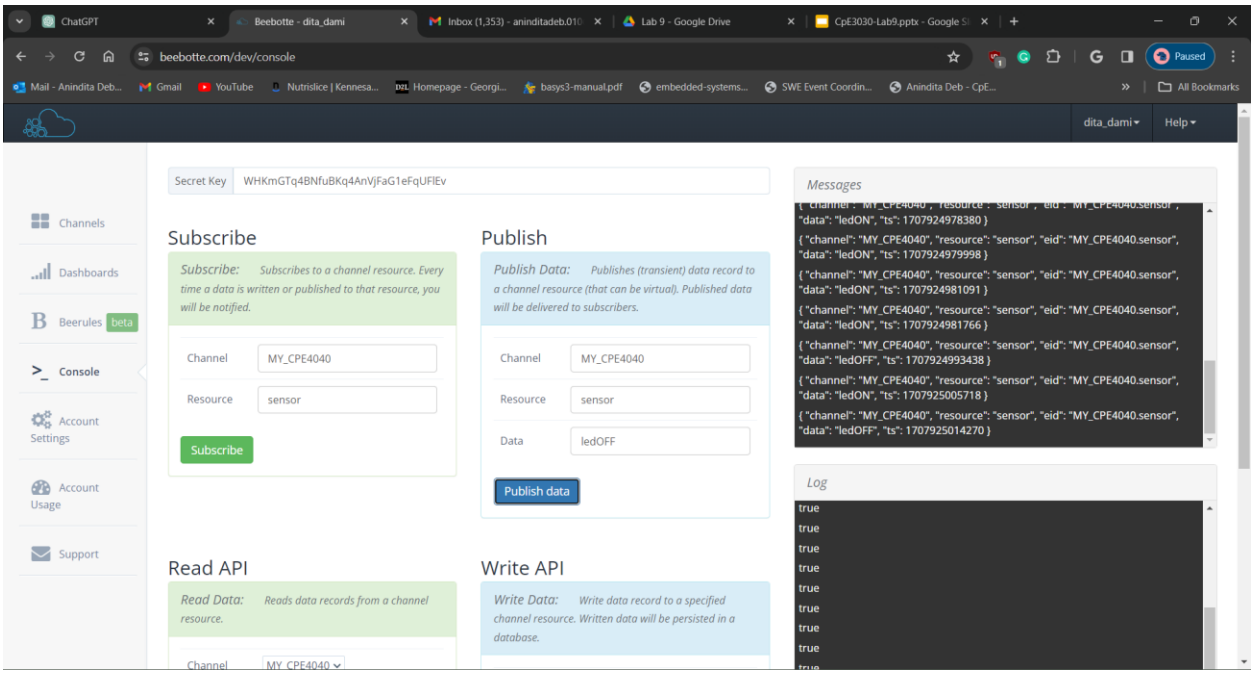

beebotte.com/dev/console
dita_dami
Help
Secret Key
WHKmGTq4BNfuBKq4AnVjFaG1eFqUFlEv
Channels
Dashboards
Beerules
beta
Console
Account Settings
Account Usage
Support
Subscribe
Subscribe: Subscribes to a channel resource. Every time a data is written or published to that resource, you will be notified.
Channel
MY_CPE4040
Resource
sensor
Subscribe
Publish
Publish Data: Publishes (transient) data record to a channel resource (that can be virtual). Published data will be delivered to subscribers.
Channel
MY_CPE4040
Resource
sensor
Data
Temperature:50
Publish data
Messages
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "", "ts": 1707757497181 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "Temperature:50", "ts": 1707757618820 }
Log
Successfully subscribed to presence-MY_CPE4040.sensor
Successfully subscribed to MY_CPE4040.sensor
true
true
Read API
Read Data: Reads data records from a channel resource.
Channel
MY_CPE4040
Write API
Write Data: Write data record to a specified channel resource. Written data will be persisted in a database.

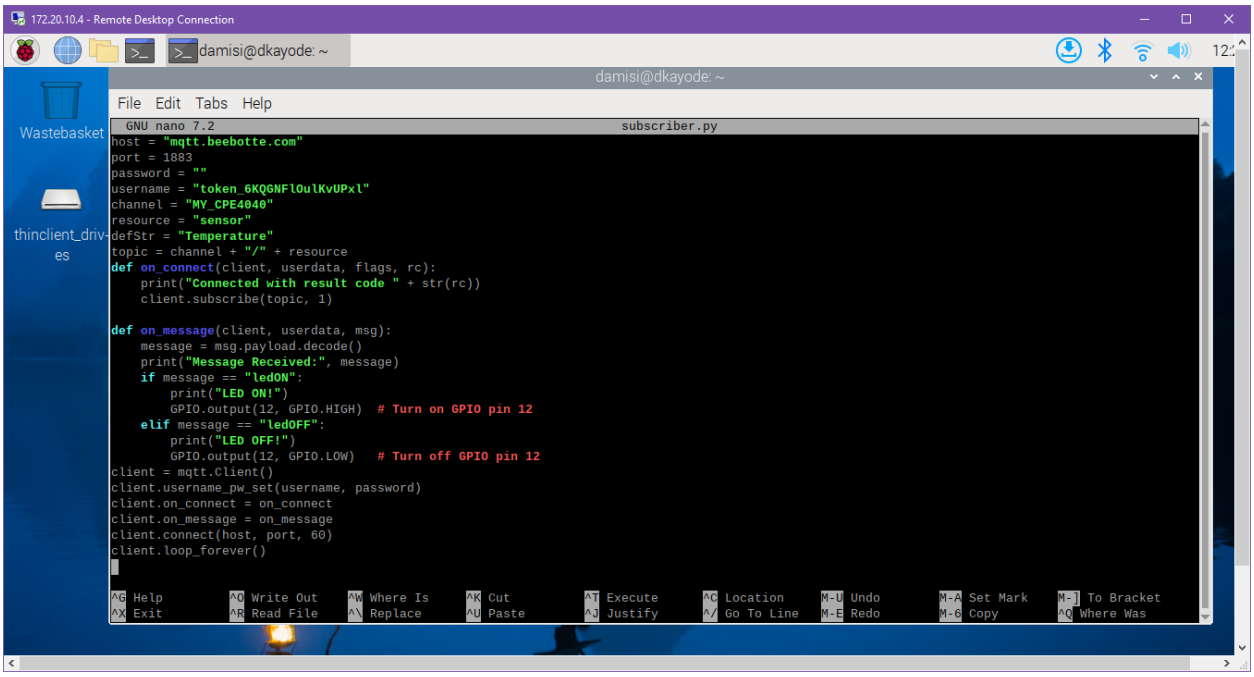

172.20.10.4 - Remote Desktop Connection
damisi@dkayode: ~
damisi@dkayode: ~
File Edit Tabs Help
Wastebasket
thinclient_drives
>>> subscriber.py
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'subscriber' is not defined
>>> import subscriber
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/home/damisi/subscriber.py", line 11
username = “token_6KQGNFlOulKvUPxl”
^
SyntaxError: invalid character '“' (U+201C)
>>>
damisi@dkayode:~ $ sudo nano subscriber.py
damisi@dkayode:~ $ python3
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import subscriber
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/home/damisi/subscriber.py", line 21
print("Message Received:", str(msg.payload.decode()))
^
IndentationError: expected an indented block after 'if' statement on line 20
>>>
damisi@dkayode:~ $ sudo nano subscriber.py
damisi@dkayode:~ $ python3
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import subscriber
Connected with result code 0
Message Received: {"data":"Temperature:50","ispublic":true,"ts":1707757618820}


Run Python files terminal.
SWE Event Coordin...
Anindita Deb - CpE...
All Bookmarks
dita_dami
Help
Messages
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "Temperature=20", "ts": 1707757668669 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": 50, "ts": 1707757689494 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "temp:50", "ts": 1707757706943 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "temperature: -50", "ts": 1707757718240 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "Temperature: 100", "ts": 1707757738356 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "Temperature: -100", "ts": 1707757744937 }
{ "channel": "MY_CPE4040", "resource": "sensor", "eid": "MY_CPE4040.sensor", "data": "Temperature 10", "ts": 1707757756182 }
Log
Successfully subscribed to presence-MY_CPE4040.sensor
Successfully subscribed to MY_CPE4040.sensor
true
true
true
true
true
true
true


^
IndentationError: expected an indented block after 'if' statement on line 20
>>>
damisi@dkayode:~ $ sudo nano subscriber.py
damisi@dkayode:~ $ python3
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import subscriber
Connected with result code 0
Message Received: {"data":"Temperature:50","ispublic":true,"ts":1707757618820}
Message Received: {"data":"Temperature=20","ispublic":true,"ts":1707757668669}
Message Received: {"data":"Temperature: 100","ispublic":true,"ts":1707757738356}
Message Received: {"data":"Temperature: -100","ispublic":true,"ts":1707757744937}
Message Received: {"data":"Temperature 10","ispublic":true,"ts":1707757756182}

14. The subscriber script also filters two distinct messages: ledON and ledOFF. Upon receiving the ledON message, the script will trigger an action to turn the LED on, and the text "LED ON" should be displayed in the Terminal to confirm the action. Conversely, when the ledOFF message is received, the script will turn the LED off, with "LED OFF" appearing in the Terminal as confirmation.
    We modify the subscriber.py script to turn on and off the LED within the message handling function. We run the script again, publish the LED control messages, and observe the status of the LED.





IV. Conclusion

- This was very straightforward yet again and it was pretty cool to see how I (Anindita) had my pc running the Beebotte site and was able to control the LED on the raspberry pi running on Damisi' pc/hotspot. It reminds me very much of a lab from device networks.
- We ran into a very simple problem during the lab which was just not getting the LED to turn on for quite a while. It was only after we realized that we didn't ground it correctly that everything was working smoothly afterwards.
- I can't say I have any suggestions for this lab, it was simple enough that we were able to complete it without too much issue.