KENNESAW STATE
UNIVERSITY

CPE 4040: Data Collection and Analysis

Lab 3: Raspberry Pi with Cloud MQTT Broker

Learning Objectives:

1. Set up a cloud based MQTT Broker.
2. Implement a MQTT subscriber code in Python to connect to MQTT broker and subscribe to topics.
3. Remotely control Raspberry Pi digital output using a cloud platform.

Hardware Requirement:

1. LEDs and resistors
2. Breadboards

About Paho MQTT:

The Paho Python Client provides a client class with support for MQTT v5.0, MQTT v3.1.1, and v3.1 on Python 3.7+. This enables your Raspberry Pi to both Publish and Subscribe to MQTT messages. For details, please check the website:

https://www.eclipse.org/paho/index.php?page=clients/python/index.php.

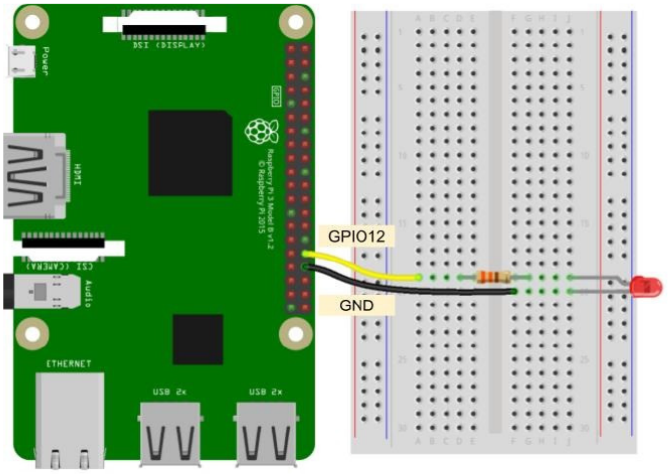
An explanation of the subscriber script used in this assignment can be found in the Appendix.

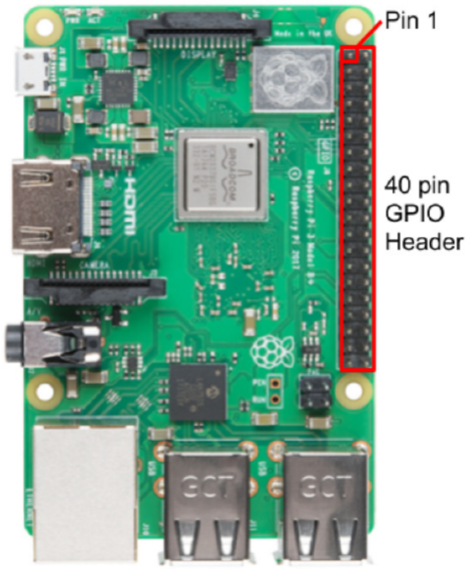
Lab Procedure:

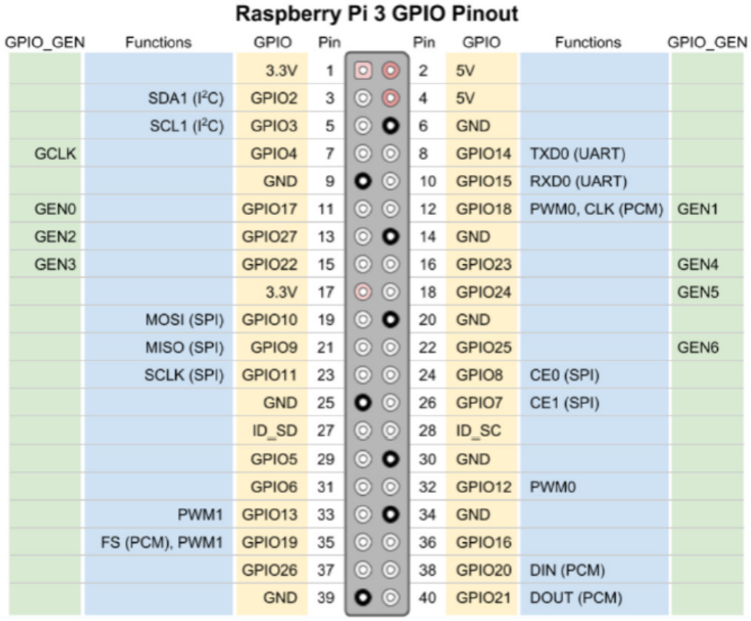
1. Power up your Raspberry Pi and open Remote Desktop Connection on your laptop and connect to the Raspberry Pi.
2. After login, open a terminal window and install the Paho-MQTT package in Python.

```
pip3 install paho-mqtt
```

3. Connect an LED with a serial resistor (330 Ω to 1 kΩ) to GPIO12 on the extension connector of your Raspberry Pi, as shown below. Don't forget to connect GND! **Remember to capture a photo of your hardware setup**.







Raspberry Pi 3 GPIO Pinout

GPIO_GEN	Functions	GPIO	Pin	Pin	GPIO	Functions	GPIO_GEN
		3.3V	1	2	5V		
	SDA1 (I^{2}C)	GPIO2	3	4	5V		
	SCL1 (I^{2}C)	GPIO3	5	6	GND		
GCLK		GPIO4	7	8	GPIO14	TXD0 (UART)	
		GND	9	10	GPIO15	RXD0 (UART)	
GEN0		GPIO17	11	12	GPIO18	PWM0, CLK (PCM)	GEN1
GEN2		GPIO27	13	14	GND		
GEN3		GPIO22	15	16	GPIO23		GEN4
		3.3V	17	18	GPIO24		GEN5
	MOSI (SPI)	GPIO10	19	20	GND		
	MISO (SPI)	GPIO9	21	22	GPIO25		GEN6
	SCLK (SPI)	GPIO11	23	24	GPIO8	CE0 (SPI)	
		GND	25	26	GPIO7	CE1 (SPI)	
		ID_SD	27	28	ID_SC		
		GPIO5	29	30	GND		
		GPIO6	31	32	GPIO12	PWM0	
	PWM1	GPIO13	33	34	GND		
	FS (PCM), PWM1	GPIO19	35	36	GPIO16		
		GPIO26	37	38	GPIO20	DIN (PCM)	
		GND	39	40	GPIO21	DOUT (PCM)	

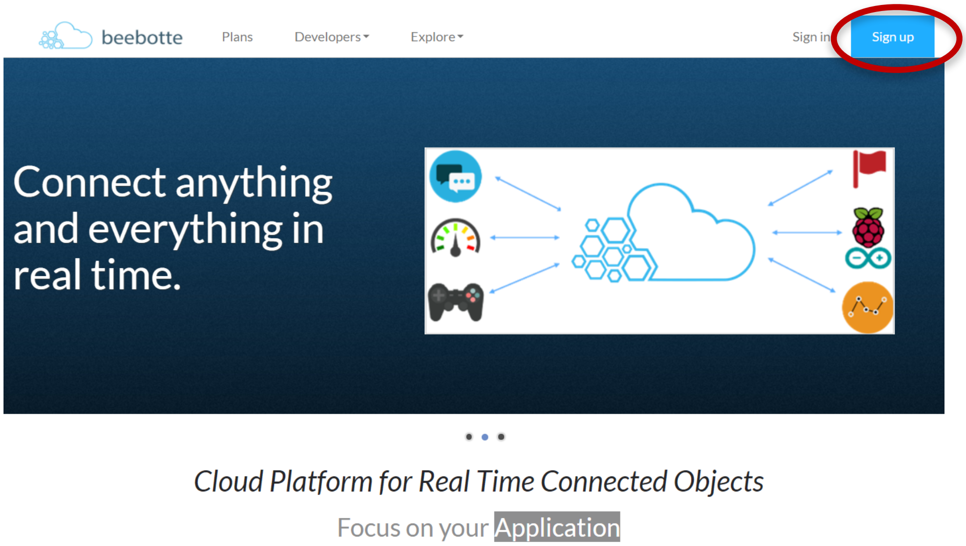
4. Create a file (`digitalOut.py`) with the python script below to test the control of the digital output. **Capture a short video while executing the script.**

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

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT)
GPIO.output(12, GPIO.HIGH)
time.sleep(1)
GPIO.output(12, GPIO.LOW)
```

5. Go to www.beebotte.com cloud platform and create a free account.

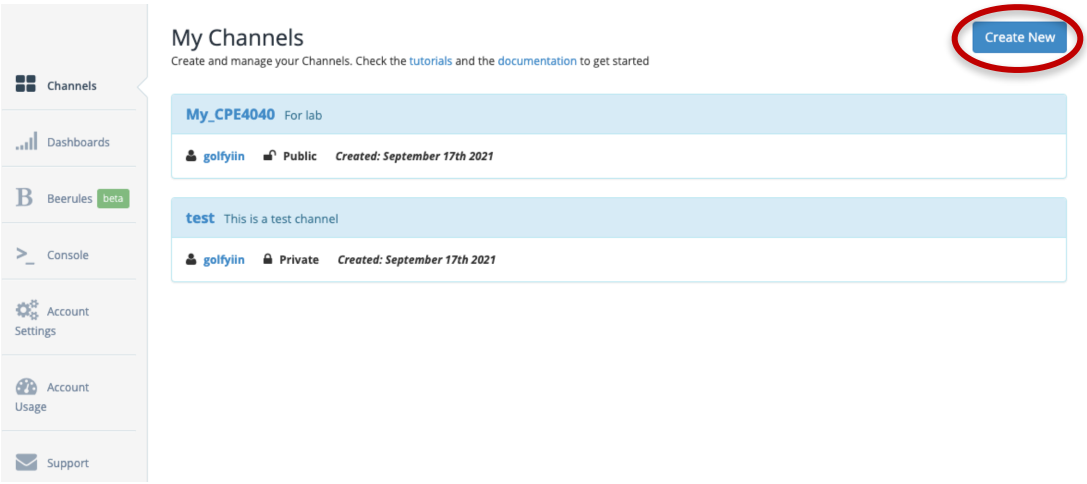


Connect anything and everything in real time.

Cloud Platform for Real Time Connected Objects

Focus on your Application

6. After logging in, go to “My Channels” and create a channel with the name of your choice (e.g., My_CPE4040). For resource name, enter “sensor” and create the new channel.



My Channels

Create and manage your Channels. Check the tutorials and the documentation to get started

My_CPE4040 For lab

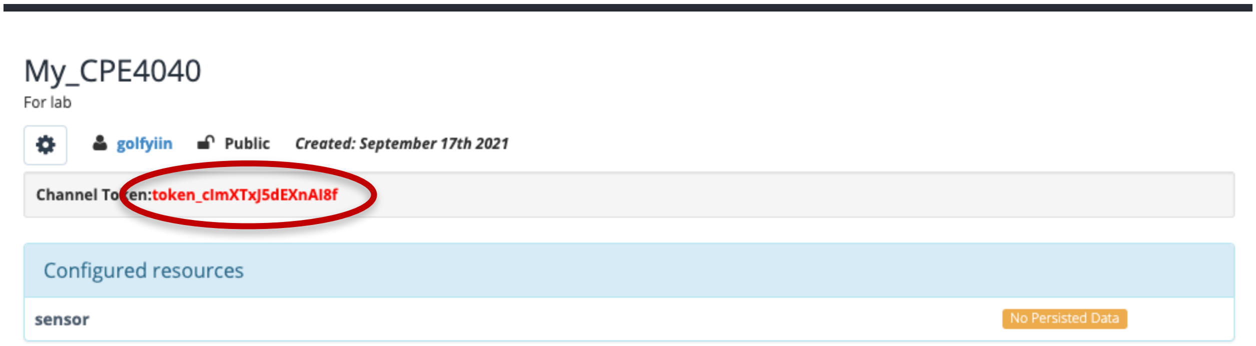
golfyiin Public *Created: September 17th 2021*

test This is a test channel

golfyiin Private *Created: September 17th 2021*

7. Click on the channel you just created and save the "Channel Token" for later use in your subscriber Python code.

   Question: What is Channel Token used for?



My_CPE4040

For lab

golfyiin Public *Created: September 17th 2021*

Channel Token:token_clmXTxJ5dEXnAI8f

Configured resources

sensor No Persisted Data

8. Create a new Python program by entering the following codes:

```
sudo nano subscriber.py
```

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO

# This is the GPIO settings
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(12, GPIO.OUT)

# This is the MQTT Connection Settings
host = "mqtt.beebotte.com"
port = 1883
password = ""
username = "<token_XXXXXXXXXXXX>"
channel = "<channel name>"
resource = "<resource name>"
defStr = "Temperature"
topic = channel + "/" + resource

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe(topic,1)

def on_message(client, userdata, msg):
    if defStr in str(msg.payload.decode()):
        print("Message Received:", str(msg.payload.decode()))
    elif "ledON" in str(msg.payload.decode()):
        print("LED ON!")
    elif "ledOFF" in str(msg.payload.decode()):
        print("LED OFF!")
```

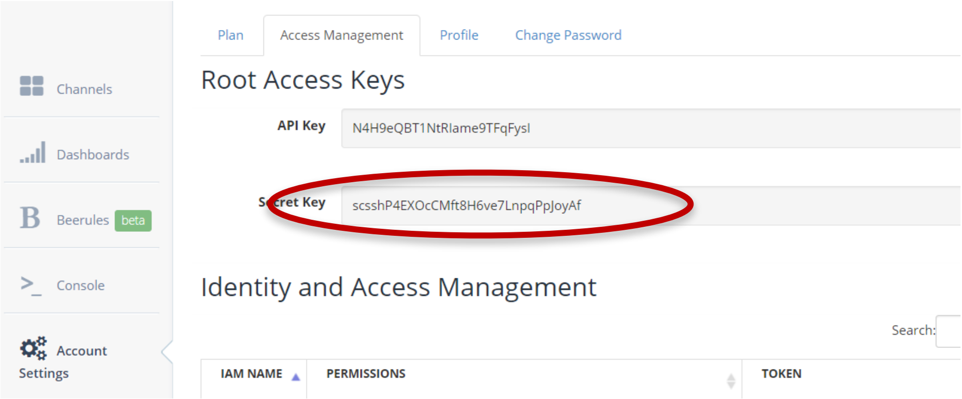
```
client = mqtt.Client()
client.username_pw_set(username)
client.connect(host,port,60)

client.on_connect = on_connect
client.on_message = on_message

client.loop_forever()
```

<u>Note</u>: Do not forget to type in your own token string, channel name and resource name in the code. Be careful about copy-and-paste the code. You need to keep proper indentations!

9. Run the code in a Terminal using **Python3**. If the connection is established, a "connected" message with result code 0 should appear on the screen. If it is a different result code, you should double check the MQTT connection parameters and run again.
10. In the Beebotte cloud platform, go to the "Account Settings" menu and click the "Access Management" tab. Make a copy of your "Secret Key".



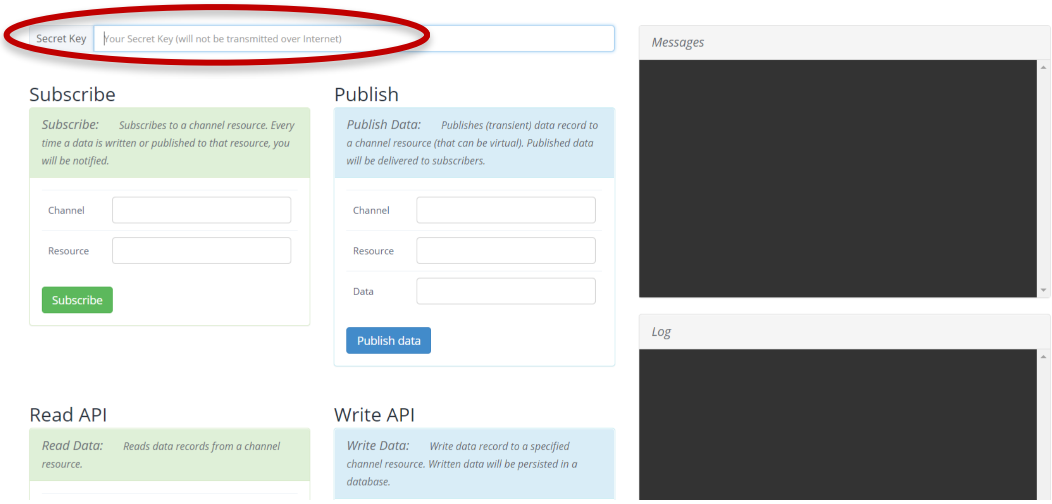
Root Access Keys

API Key: N4H9eQBT1NtRlame9TFqFysl

Secret Key: scsshP4EXOcCMft8H6ve7LnpqPpJoyAf

Identity and Access Management

11. Go to the "Console" menu and paste your Secret Key in the box at the top. Now you are ready to test the MQTT Subscribe and Publish functions. **<u>Question</u>**: Why is this key needed?



Subscribe: *Subscribes to a channel resource. Every time a data is written or published to that resource, you will be notified.*

Publish Data: *Publishes (transient) data record to a channel resource (that can be virtual). Published data will be delivered to subscribers.*

Read Data: *Reads data records from a channel resource.*

Write Data: *Write data record to a specified channel resource. Written data will be persisted in a database.*

12. To test your channel in the Beebotte cloud, fill out the channel name and the resource name in the Subscribe section and click the “Subscribe” button. Fill out the same information in the Publish section with arbitrary data and click the “Publish” button. Observe the messages starting to appear inside the Message window.

13. The `subscriber.py` code filters the messages starting with “Temperature:” (**<u>Question:</u>** How do we know it?) Write an appropriate message (i.e. “Temperature:50”) in the data field of the Publish section and test to see it on Raspberry Pi. Try to publish **at least five messages** with and without the proper format.

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.

    Modify the `subscriber.py` script to turn on and off the LED within the message handling function. Run the script again, publish the LED control messages, and observe the status of the LED. Record a video that captures the LED being turned on and off. **Attach the modified code in the lab report**.

Lab Report:

Please follow the guidelines to prepare your lab report and remember to include the following in the report:

- A clear and well-lit photo of your hardware setup
- Modified codes
- Answers to the questions asked in some of the steps

Submit the report via D2L Assignment, in PDF format.

Appendix

Lab 3: subscriber.py code analysis

In [ ]:

```
!pip3 install paho-mqtt
```

Client

You can use the client class as an instance, within a class or by subclassing. The general usage flow is as follows:

- Create a client instance using **Client( )** constrcutor
- Connect to a broker using one of the **connect*( )** functions
- Call one of the **loop*( )** functions to maintain network traffic flow with the broker
- Use **subscribe( )** to subscribe to a topic and receive messages
- Use **publish( )** to publish messages to the broker
- Use **disconnect( )** to disconnect from the broker

Callbacks will be called to allow the application to process events as necessary. These callbacks are described below.

In [ ]:

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO

# This is the GPIO settings
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT)

# This is the MQTT Connection Settings
host = "mqtt.beebotte.com"
port = 1883
password = ""
username = "<token_XXXXXXXXXXXX>"
channel = "<channel name>"
resource = "<resource name>"
defStr = "Temperature"
topic = channel + "/" + resource


## When a client issues a CONNECT request, the broker should send an
## acknowledgment (CONNACK).
## This mechanism is referred to as Callback and we usually create a function to
## handle this callback

## The callback for when the client receives a CONNACK response from the server.
## rc = Connection Return Code
##  0: Connection successful
##  1: Connection refused - incorrect protocol version
##  2: Connection refused - invalid client identifier
##  3: Connection refused - server unavailable
##  4: Connection refused – bad username or password
##  5: Connection refused – not authorised
##  6–255: Currently unused.

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.
    # Syntax: subscribe(topic, qos=0)

    client.subscribe(topic,1)

## The callback for when a PUBLISH message is received from the server.
## The binary message msg.payload is decoded into a string or JSON

def on_message(client, userdata, msg):
    if defStr in str(msg.payload.decode("utf-8")):    ## Receiving Temp reading
        print("Message Received:", str(msg.payload.decode("utf-8")))
    elif "ledON" in str(msg.payload.decode()):        ## For RPi LED control
        print("LED ON!")
    elif "ledOFF" in str(msg.payload.decode()):
        print("LED OFF!")


## MQTT Client constructor
## Syntax: Client(client_id="", clean_session=True, userdata=None,
##         protocol=MQTTv311, transport="tcp")

client = mqtt.Client()

## The MQTT broker requires username and password

client.username_pw_set(username)

## The connect() function connects the client to a broker.
## Syntax: connect(host, port=1883, keepalive=60)

client.connect(host, port, 60)

client.on_connect = on_connect
client.on_message = on_message

## The loop() function will read the receive and send buffers and process
## the messages it finds.
## On the receive side it looks at the messages, and depending on the message
## type, it will trigger the appropriate callback function, e.g., on_connect
## or on_message

client.loop_forever()
```

In [ ]: