Experiment No. 4
Title: MQTT using HiveMQ cloud and R-Pi
Aim: Write a program on Arduino / Raspberry Pi subscribe to MQTT broker for temperature data and print it.

Hardware/Software: Raspberry Pi

Theory:

MQTT is simple, lightweight messaging protocol used to establish communication between multiple devices. It is TCP-based protocol relying on the publish-subscribe model. This communication protocol is suitable for transmitting data between resource-constrained devices having low bandwidth and low power requirements. Hence this messaging protocol is widely used for communication in IoT Framework.

Publish-Subscribe Model :
This model involves multiple clients interacting with each other, without having any direct connection established between them. All clients communicate with other clients only via third party known as Broker.

MQTT Client and Broker :
Clients publish messages on different topics to broker. The broker is the central server that receives these messages and filters them based on their topics. It then sends these messages to respective clients that have subscribed to those different topics.

Hence client that has subscribed to a specific topic receives all messages published on that topic.

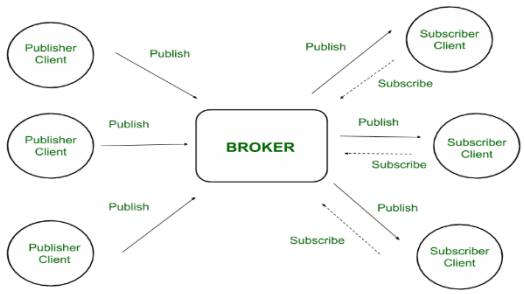


Figure – Publish-Subscribe Model

Here the broker is central hub that receives messages, filters them, and distributes them to appropriate clients, such that both message publishers, as well as subscribers, are clients.

Advantages :

1. Easy Scalability –
   This model is not restricted to one-to-one communication between clients. Although the publisher client sends a single message on specific topic, broker sends multiple messages to all different clients subscribed to that topic. Similarly, messages sent by multiple such publisher clients on multiple different topics will be sent to all multiple clients subscribed to those topics.
   Hence one-to-many, many-to-one, as well as many-to-many communication is possible using this model. Also, clients can publish data and at the same time receive data due to this two-way communication protocol. Hence MQTT is considered to be bi-directional protocol. The default unencrypted MQTT port used for data transmission is 1883. The encrypted port for secure transmission is 8883.
2. Eliminates insecure connections –
   In a complex system where multiple devices are connected with each other, each device not only has to manage its connections with other devices but also has to ensure that these connections are secure. But in the publish-subscribe model, the broker becomes central server managing all security aspects. It is responsible for the authentication and authorization of all connected clients.
3. Lightweight Communication –
   Data transmission is quick, efficient, and lightweight because MQTT messages have small code footprint. These control messages have a fixed header of size 2 bytes and payload message up to size 256 megabytes.

Topics :

In MQTT, topic is UTF-8 string that the broker uses to filter messages for each individual connected client. Each topic consists of one or more different topic levels. Each topic level is separated by forward slash also called topic level separator. Both topics and levels are casesensitive.

MQTT HIVEMQ

HiveMQ is an MQTT broker and a client based messaging platform designed for the fast, efficient and reliable movement of data to and from connected IoT devices. It uses the MQTT protocol for instant, bi-directional push of data between your device and your enterprise systems.

The biggest advantage of HiveMQ Cloud: There's no need for an always-on server that one must self-manage. Yes, Mosquitto runs on an inexpensive Raspberry Pi. Still, we require it to keep it up-and-running if we use it as the "central hub" to distribute data. Another big advantage of

HiveMQ Cloud: It is free to connect up to 100 devices! Even for the most enthusiastic maker, that's a lot of microcontrollers or computers! You can find example code for many programming languages from HiveMQ.

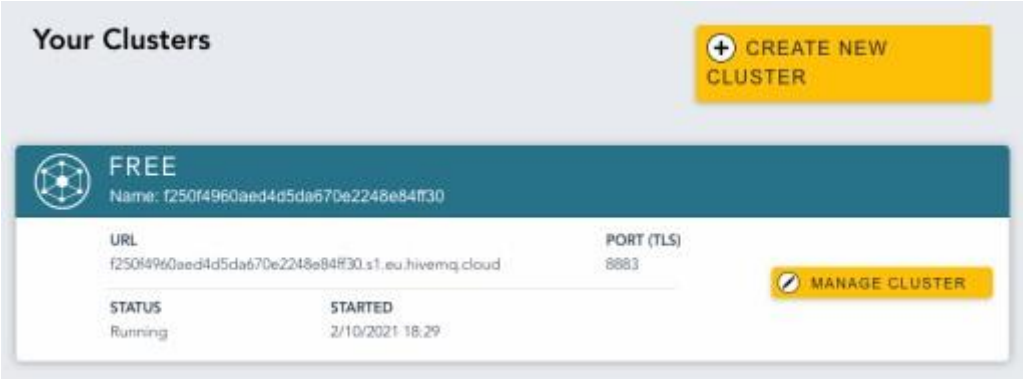
Setting Up a HiveMQ Cloud Account

To start with a free HiveMQ Cloud account.

1. Go to HiveMQ Cloud Sign Up, click on ***Sign Up Now*** and create a login.

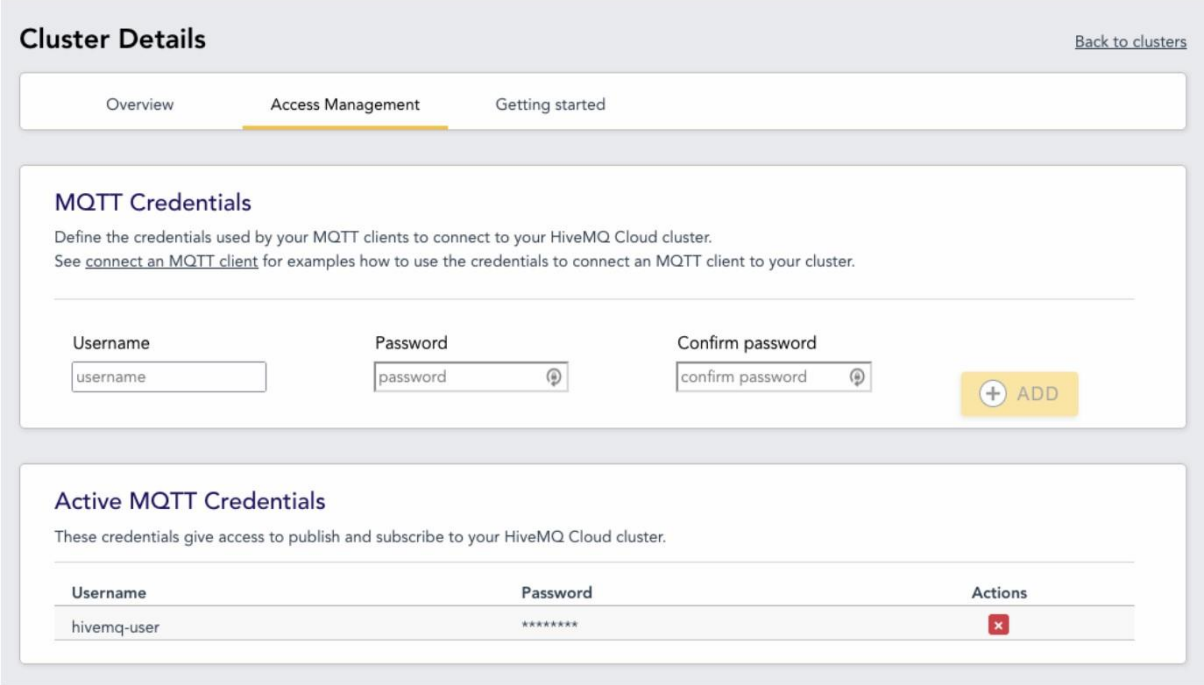
2. Once logged in, you will be presented with your HiveMQ MQTT cluster.

3. Click on the ***Manage Cluster*** button. You will see the details of your HiveMQ Cloud instance.



4. Go to the ***Access Management*** section and create a username and password for the credentials, which we will use in our application.

Lab Manual: Edge and Fog Computing



https://www.emqx.com/en/blog/use-mqtt-with-raspberry-pi

Procedure:
Program:

MQTT Client program
Install MQTT Broker Server
sudo apt-get install mosquitto

Command line clients in case for debugging

```
# sudo apt-get install mosquitto-clients -y


# Install the MQTT Publisher
# sudo pip3 install paho-mqtt


import os import sys import
time import board import
adafruit_dht import
paho.mqtt.client as mqtt import
json


# Initial the dht device, with data pin connected to: dhtDevice =
adafruit_dht.DHT11(board.D19, use_pulseio=False)


sensor_data = {'temperature': 0, 'humidity': 0}


Server = '127.0.0.1'


client = mqtt.Client()


client.connect(Server, 1883, 60)


client.loop_start()


if __name__ == '__main__':
while True:        try:
          # Print the values to the serial port
temperature = dhtDevice.temperature
humidity = dhtDevice.humidity
print("Temp: {:.1f} C    Humidity: {}% "
.format( temperature, humidity))
time.sleep(2.0)
sensor_data['temperature'] = temperature
sensor_data['humidity'] = humidity
```

```
        # Sending humidity and temperature data to ThingsBoard
client.publish('test_channel', json.dumps(sensor_data), 1)


        time.sleep(5)

      except RuntimeError as error:
        # Errors happen fairly often, DHT's are hard to read, just keep going
print(error.args[0])          time.sleep(2.0)          continue


      except KeyboardInterrupt:
        client.loop_stop()
client.disconnect()
        print ('Exiting Program')
exit()
```

HiveMQ MQTT Server Program

```
# Install the MQTT Publisher
# sudo pip3 install paho-mqtt


# open the MQTT browser client in web browser #
http://www.hivemq.com/demos/websocket-client/


import os import sys import
time import board import
adafruit_dht import
paho.mqtt.client as mqtt import
json


# Initial the dht device, with data pin connected to: dhtDevice =
adafruit_dht.DHT11(board.D19, use_pulseio=False)


sensor_data = {'temperature': 0, 'humidity': 0}


MQTTServer = 'broker.mqttdashboard.com'
```

```
client = mqtt.Client()

client.connect(MQTTServer, 1883, 8000)

client.loop_start()

if __name__ == '__main__':
while True:        try:
        # Print the values to the serial port            temperature =
dhtDevice.temperature            humidity = dhtDevice.humidity
print("Temp: {:.1f} C    Humidity: {}% ".format( temperature, humidity))
time.sleep(2.0)            sensor_data['temperature'] = temperature
sensor_data['humidity'] = humidity


        # Sending humidity and temperature data to HIVEMQ
client.publish('RPI4_MQTT', json.dumps(sensor_data), 1)

        time.sleep(5)

    except RuntimeError as error:
        # Errors happen fairly often, DHT's are hard to read, just keep going
print(error.args[0])            time.sleep(2.0)            continue


    except KeyboardInterrupt:
        client.loop_stop()
client.disconnect()
        print ('Exiting Program')
        exit()
```

Conclusion:

To implement a program on a Raspberry Pi that subscribes to an MQTT broker for temperature data and prints it, you can leverage the lightweight MQTT protocol, ideal for IoT devices with limited resources. The MQTT protocol's publish-subscribe model allows the Raspberry Pi to act as a client, subscribing to specific topics (e.g., temperature) and receiving relevant messages from the broker. The HiveMQ Cloud broker can be used for this purpose, as it simplifies management by eliminating the need for an always-on local server. Once set up, the Raspberry Pi can use Python's Paho MQTT library to connect to the broker, subscribe to the desired topic, and print incoming temperature data in real-time. This setup is efficient, scalable, and well-suited for IoT applications where reliable, low-bandwidth communication is crucial.