Wireless Networks

A Project By

Vaibhav Gandhi - 60003180056
Vedant Gandhi - 60003180060
Smit Chatwani - 60003180052

Literature on Publisher-Subscriber Model using MQTT connection

Publish-subscribe model

The publish-subscribe model is different from the traditional client-server model. It separates the client (publisher) that sends the message from the client (subscriber) that receives the message. The publisher and the subscriber do not need to establish direct contact. We can either let multiple publishers publish messages to one subscriber, or let multiple subscribers receive messages from one publisher at the same time. The essence of it is that an intermediate role called a broker is responsible for all message routing and distribution. The traditional client-server model can achieve similar results, but it cannot be as simple and elegant as the publish-subscribe model.

The advantage of the publish-subscribe model is the decoupling of publishers and subscribers. This decoupling is manifested in the following two aspects:

- Spatial decoupling. Subscribers and publishers do not need to establish a direct connection, and new subscribers do not need to modify the publisher's behavior when they want to join the network.
- Time decoupling. Subscribers and publishers do not need to be online at the same time. Even if there are no subscribers, it does not affect publishers to publish messages.
- Synchronization decoupling: Operations on both components do not need to be interrupted during publishing or receiving.

In summary, the pub/sub model removes direct communication between the publisher of the message and the recipient/subscriber. The filtering activity of the broker makes it possible to control which client/subscriber receives which message. The decoupling has three dimensions: space, time, and synchronization.

Message Filtering

The broker plays a pivotal role in the pub/sub process as he needs to accurately and efficiently forward the desired messages to the subscribers.The broker has several filtering options:

OPTION 1: SUBJECT-BASED FILTERING

This filtering is based on the subject or topic that is part of each message. The receiving client subscribes to the broker for topics of interest. From that point on, the broker ensures that the receiving client gets all messages published to the subscribed topics. In general, topics are strings with a hierarchical structure that allow filtering based on a limited number of expressions.

OPTION 2: CONTENT-BASED FILTERING

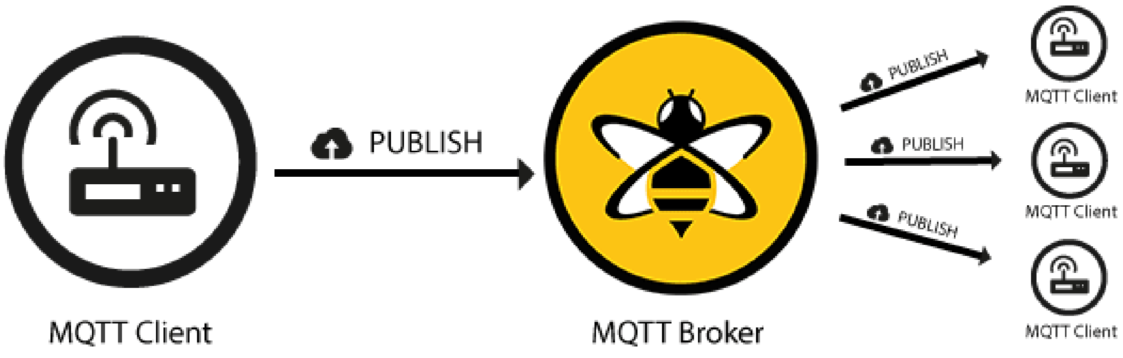
In content-based filtering, the broker filters the message based on a specific content filter-language. The receiving clients subscribe to filter queries of messages for which they are interested. A significant downside to this method is that the content of the message must be known beforehand and cannot be encrypted or easily changed.

OPTION 3: TYPE-BASED FILTERING

When object-oriented languages are used, filtering based on the type/class of a message (event) is a common practice. For example, a subscriber can listen to all messages, which are of type Exception or any sub-type.

MQTT

The MQTT protocol distributes messages based on topics rather than message content. Each message contains a topic, and the broker does not need to parse user data. This provides the possibility to implement a general, business-independent MQTT broker. Users can also encrypt their data at will, which is very useful for WAN communication.



Compared with message queuing, MQTT does not require the topic to be created explicitly before publishing or subscribing. The only possible adverse effect is that the client may use the wrong topic without knowing it, but the benefit of flexible deployment is higher than that.

MQTT is not a message queue, although many behaviors and characteristics of the two are very close, such as using a publish-subscribe model. The scenarios they face are significantly different. Message queues are mainly used for message storage and forwarding between server-side applications. In this kind of scenario, the data volume is often large but the access volume is small. MQTT is targeted at the IoT field and the mobile Internet field. The focus of such scenarios is massive device access, management and messaging. In practical scenarios, the two are often used in combination. For example, MQTT Broker first receives data uploaded by IoT devices, and then forwards these data to specific applications for processing through message queues.

To handle the challenges of a pub/sub system, MQTT has three Quality of Service (QoS) levels. You can easily specify that a message gets successfully delivered from the client to the broker or from the broker to a client. However, there is the chance that nobody subscribes to the particular topic. If this is a problem, the broker must know how to handle the situation.

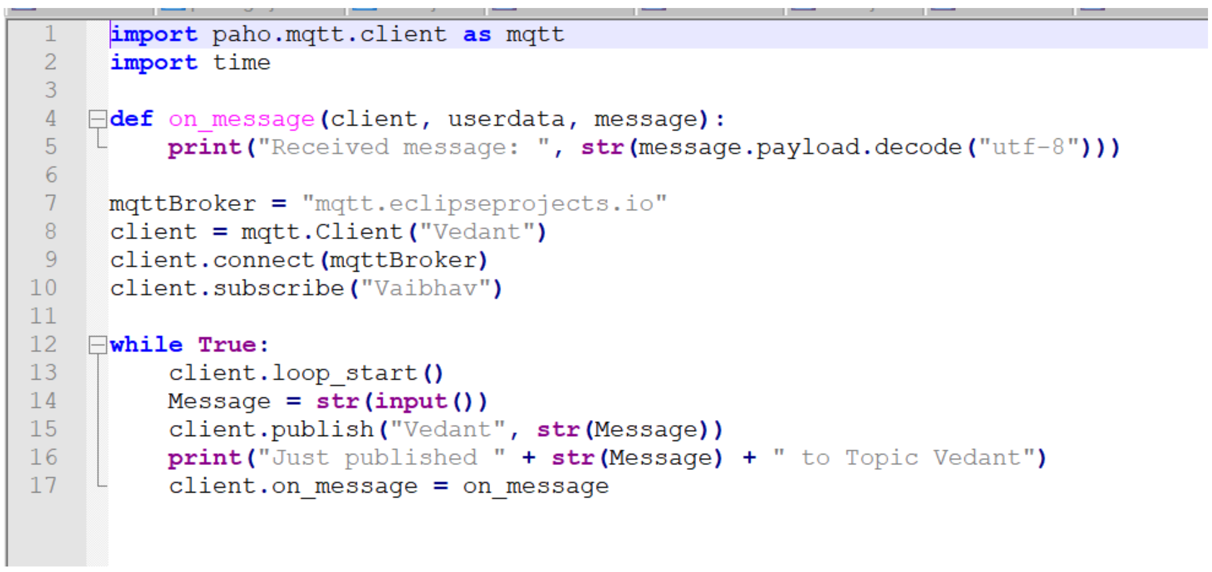
Methodology

Implementation:

This project successfully implements a publisher-subscriber communication model using the MQTT protocol. The project implemented makes use of 2 servers that act as the subscriber and publisher. The goal was for the publisher to send an email to the subscriber along with a confirmatory message signifying a successful MQTT connection

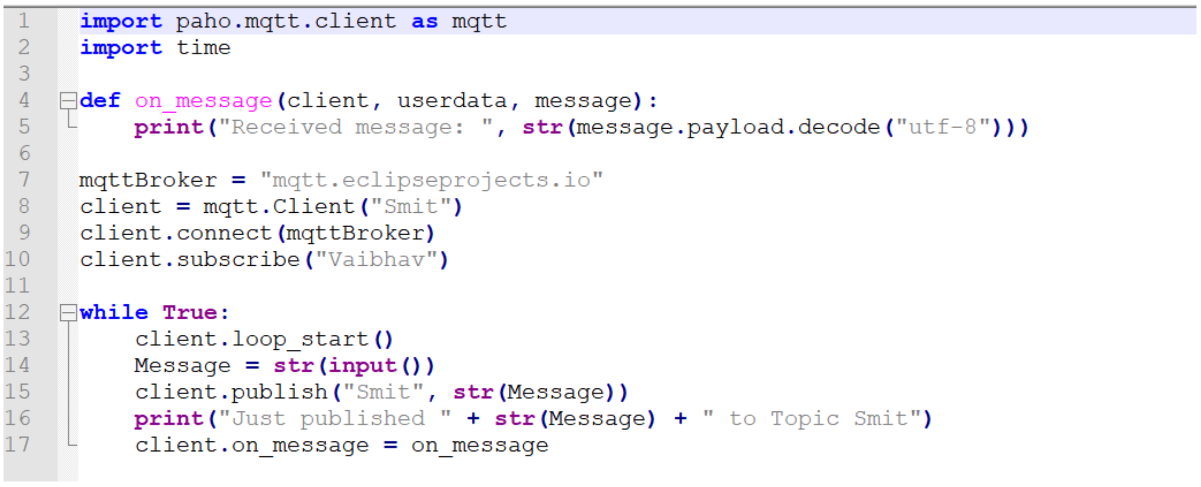
For local host:-

Code:



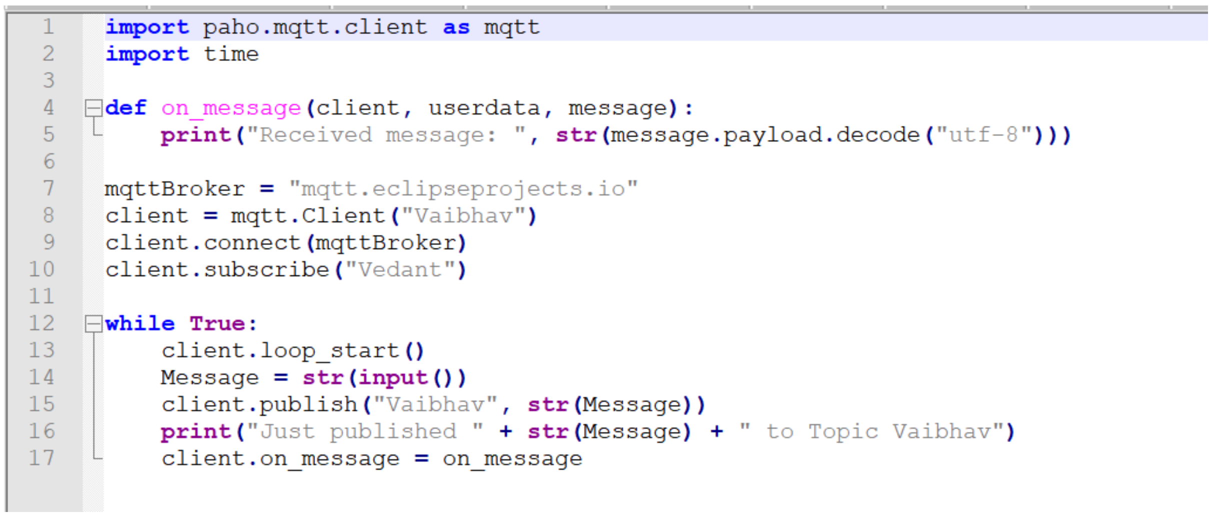
```python
import paho.mqtt.client as mqtt
import time

def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))

mqttBroker = "mqtt.eclipseprojects.io"
client = mqtt.Client("Vedant")
client.connect(mqttBroker)
client.subscribe("Vaibhav")

while True:
    client.loop_start()
    Message = str(input())
    client.publish("Vedant", str(Message))
    print("Just published " + str(Message) + " to Topic Vedant")
    client.on_message = on_message
```

Fig- mqtt_subscriber.py



```python
import paho.mqtt.client as mqtt
import time

def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))

mqttBroker = "mqtt.eclipseprojects.io"
client = mqtt.Client("Smit")
client.connect(mqttBroker)
client.subscribe("Vaibhav")

while True:
    client.loop_start()
    Message = str(input())
    client.publish("Smit", str(Message))
    print("Just published " + str(Message) + " to Topic Smit")
    client.on_message = on_message
```

Fig- mqtt_subscriber2.py



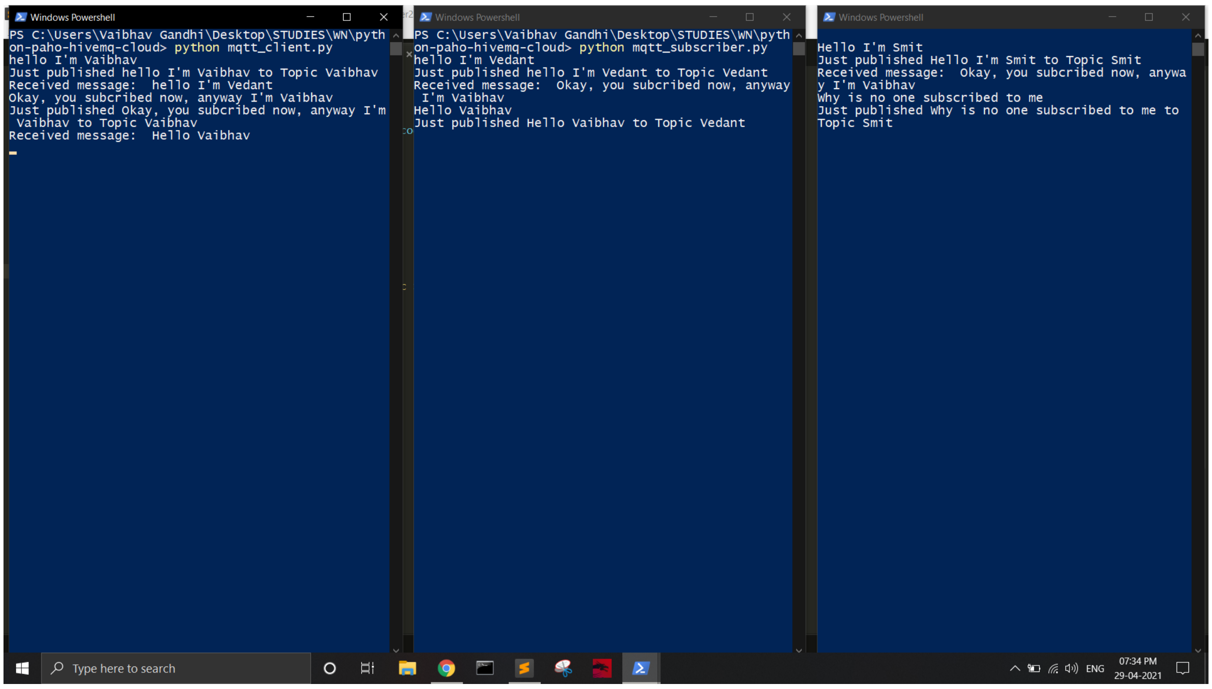
```python
import paho.mqtt.client as mqtt
import time

def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))

mqttBroker = "mqtt.eclipseprojects.io"
client = mqtt.Client("Vaibhav")
client.connect(mqttBroker)
client.subscribe("Vedant")

while True:
    client.loop_start()
    Message = str(input())
    client.publish("Vaibhav", str(Message))
    print("Just published " + str(Message) + " to Topic Vaibhav")
    client.on_message = on_message
```

Fig- mqtt_client.py

Vaibhav is subscribed by clients Vedant and Smit

Vedant is subscribed by client Vaibhav only

Smit is not subscribed by anyone



```
PS C:\Users\Vaibhav Gandhi\Desktop\STUDIES\WN\pyth
on-paho-hivemq-cloud> python mqtt_client.py
hello I'm Vaibhav
Just published hello I'm Vaibhav to Topic Vaibhav
Received message:  hello I'm Vedant
Okay, you subcribed now, anyway I'm Vaibhav
Just published Okay, you subcribed now, anyway I'm
 Vaibhav to Topic Vaibhav
Received message:  Hello Vaibhav
```

```
PS C:\Users\Vaibhav Gandhi\Desktop\STUDIES\WN\pyth
on-paho-hivemq-cloud> python mqtt_subscriber.py
hello I'm Vedant
Just published hello I'm Vedant to Topic Vedant
Received message:  Okay, you subcribed now, anyway
 I'm Vaibhav
Hello Vaibhav
Just published Hello Vaibhav to Topic Vedant
```

```
Hello I'm Smit
Just published Hello I'm Smit to Topic Smit
Received message:  Okay, you subcribed now, anywa
y I'm Vaibhav
Why is no one subscribed to me
Just published Why is no one subscribed to me to
Topic Smit
```

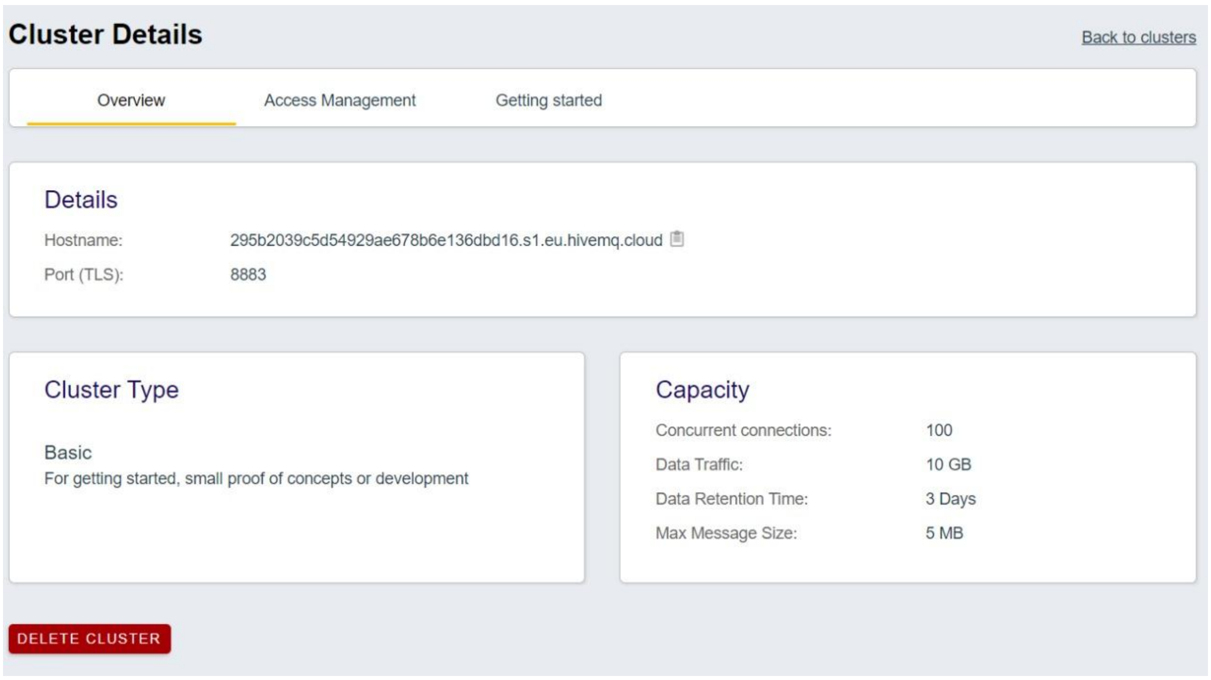
So whenever Vaibhav messages Vedant and Smit will receive them.

When Vedant messages Vaibhav will receive the messages as he has subscribed to Vedant.

When Smit messages, no one receive his messages as no one has subscribed to Smit.

For Cloud:-

HiveMQ cluster is used as the broker



Cluster Details

Details

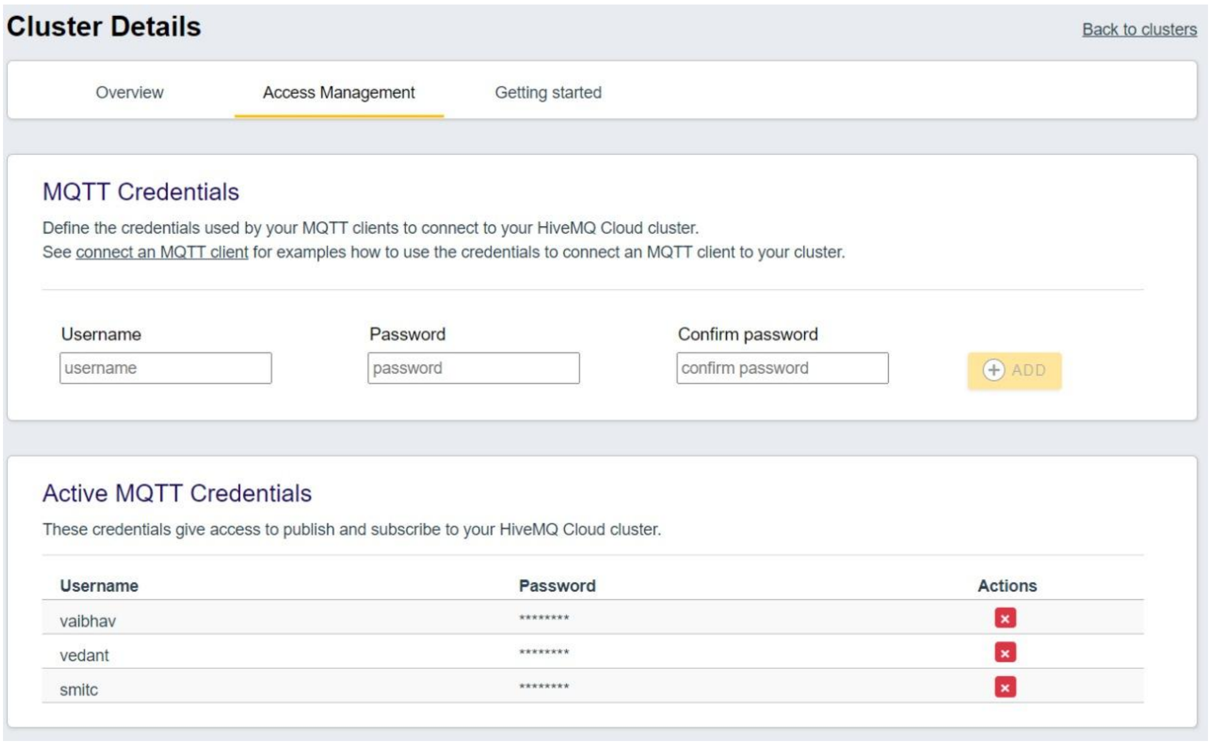
Hostname:	295b2039c5d54929ae678b6e136dbd16.s1.eu.hivemq.cloud
Port (TLS):	8883

Cluster Type

Basic
For getting started, small proof of concepts or development

Capacity

Concurrent connections:	100
Data Traffic:	10 GB
Data Retention Time:	3 Days
Max Message Size:	5 MB



Cluster Details

MQTT Credentials

Define the credentials used by your MQTT clients to connect to your HiveMQ Cloud cluster.
See connect an MQTT client for examples how to use the credentials to connect an MQTT client to your cluster.

Username	Password	Confirm password
username	password	confirm password

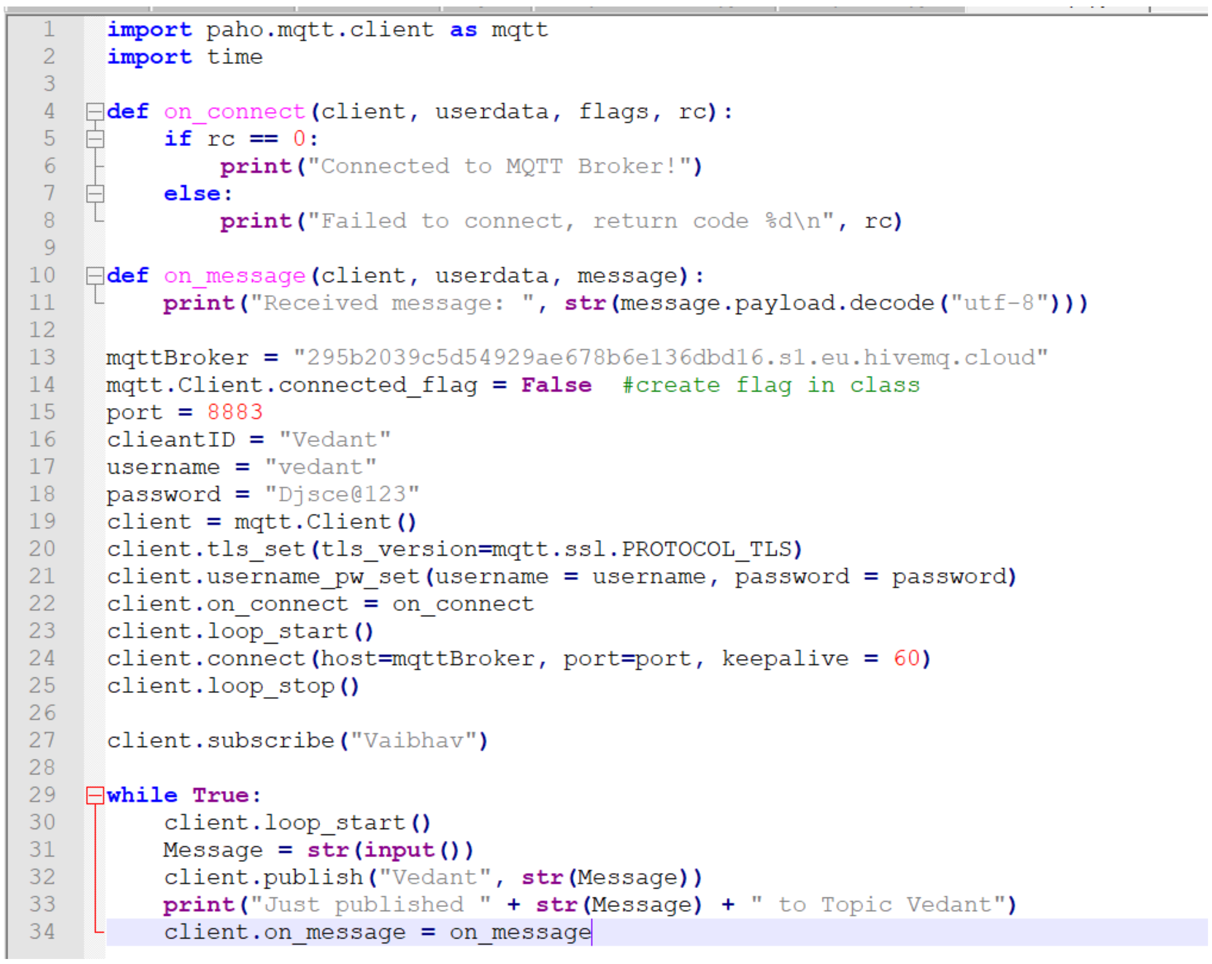
Active MQTT Credentials

These credentials give access to publish and subscribe to your HiveMQ Cloud cluster.

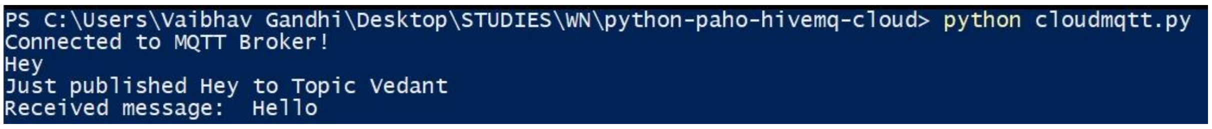
Username	Password	Actions
vaibhav	********	✕
vedant	********	✕
smitc	********	✕

Code:-

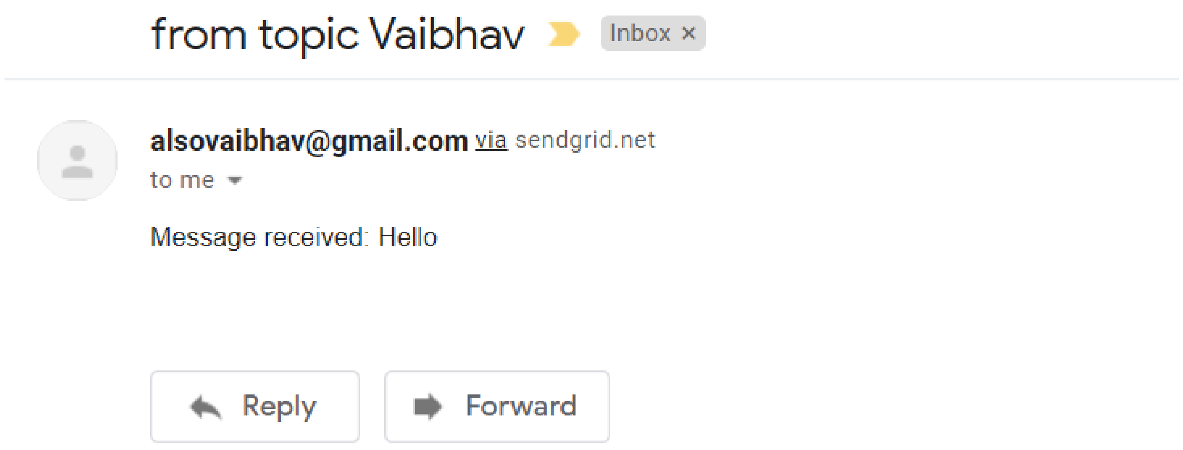
```
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)

def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))

mqttBroker = "295b2039c5d54929ae678b6e136dbd16.s1.eu.hivemq.cloud"
mqtt.Client.connected_flag = False  #create flag in class
port = 8883
clieantID = "Vaibhav"
username = "vaibhav"
password = "Djsce@123"
client = mqtt.Client()
client.tls_set(tls_version=mqtt.ssl.PROTOCOL_TLS)
client.username_pw_set(username = username, password = password)
client.on_connect = on_connect
client.loop_start()
client.connect(host=mqttBroker, port=port, keepalive = 60)
client.loop_stop()

client.subscribe("Vedant")

while True:
  client.loop_start()
  Message = str(input())
  client.publish("Vaibhav", str(Message))
  sg =
SendGridAPIClient('SG.7yUXG8ufSZSEGyPPwmhlBQ.Iwcg6zXusL1D-mo-9j-iyU9D
GC6DICqqO93feNT2rFc')
  from_email = Email("alsovaibhav@gmail.com")  # Change to your verified sender
  to_email = To("vedantgandhi24@gmail.com")  # Change to your recipient
  subject = "from topic Vaibhav"
  content = Content("text/plain", "Message received: "+str(Message))
  mail = Mail(from_email, to_email, subject, content)

  # Get a JSON-ready representation of the Mail object
  mail_json = mail.get()

  # Send an HTTP POST request to /mail/send
  response = sg.client.mail.send.post(request_body=mail_json)
  print(response.status_code)
  print(response.headers)
  print("Just published " + str(Message) + " to Topic Vaibhav")
  client.on_message = on_message
```



```python
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)

def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))

mqttBroker = "295b2039c5d54929ae678b6e136dbd16.s1.eu.hivemq.cloud"
mqtt.Client.connected_flag = False  #create flag in class
port = 8883
clieantID = "Vedant"
username = "vedant"
password = "Djsce@123"
client = mqtt.Client()
client.tls_set(tls_version=mqtt.ssl.PROTOCOL_TLS)
client.username_pw_set(username = username, password = password)
client.on_connect = on_connect
client.loop_start()
client.connect(host=mqttBroker, port=port, keepalive = 60)
client.loop_stop()

client.subscribe("Vaibhav")

while True:
    client.loop_start()
    Message = str(input())
    client.publish("Vedant", str(Message))
    print("Just published " + str(Message) + " to Topic Vedant")
    client.on_message = on_message
```



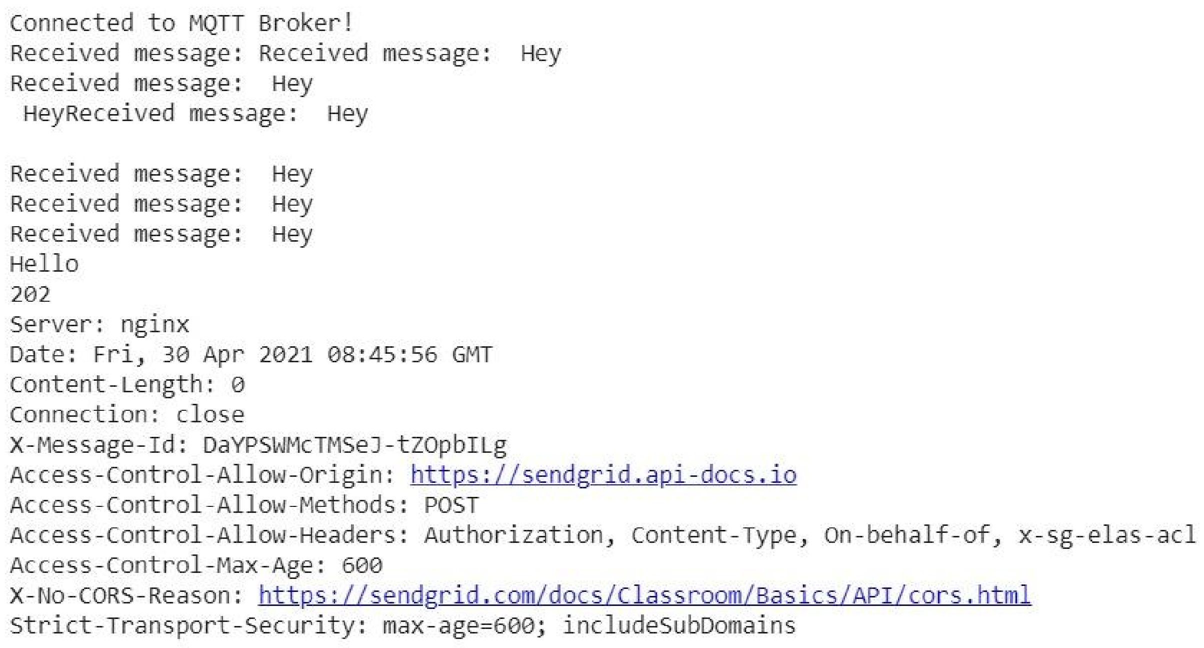
```
PS C:\Users\Vaibhav Gandhi\Desktop\STUDIES\WN\python-paho-hivemq-cloud> python cloudmqtt.py
Connected to MQTT Broker!
Hey
Just published Hey to Topic Vedant
Received message:  Hello
```



from topic Vaibhav

alsovaibhav@gmail.com via sendgrid.net

Message received: Hello



```
Connected to MQTT Broker!
Received message: Received message:  Hey
Received message:  Hey
 HeyReceived message:  Hey

Received message:  Hey
Received message:  Hey
Received message:  Hey
Hello
202
Server: nginx
Date: Fri, 30 Apr 2021 08:45:56 GMT
Content-Length: 0
Connection: close
X-Message-Id: DaYPSWMcTMSeJ-tZOpbILg
Access-Control-Allow-Origin: https://sendgrid.api-docs.io
Access-Control-Allow-Methods: POST
Access-Control-Allow-Headers: Authorization, Content-Type, On-behalf-of, x-sg-elas-acl
Access-Control-Max-Age: 600
X-No-CORS-Reason: https://sendgrid.com/docs/Classroom/Basics/API/cors.html
Strict-Transport-Security: max-age=600; includeSubDomains
```

References

- https://www.hivemq.com/blog/mqtt-client-library-paho-python/
- https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/
- https://stackoverflow.com