**MQTT protocol in Python**

**What is MQTT in IoT?**

**Internet of Things (IoT) and machine-to-machine (M2M) technologies need to use a messaging and connectivity protocol in order to exchange information from a remote location.**

**A few desirable features of such a protocol are:**

* **Small code footprint (to make it easy to implement in small devices)**
* **Low power consumption**
* **Low bandwidth consumption**
* **Low latency**

**MQTT fulfills all of these requirements and has the momentum of the big public clouds—Amazon Web Services, Microsoft Azure, and Google Cloud Platform—behind it. In this article, we’ll explore why MQTT is the most popular choice of messaging protocol for IoT products.**

## **What is MQTT?**

* **MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol that was developed by IBM and first released in 1999. It uses the pub/sub pattern and translates messages between devices, servers, and applications.**
* **The MQTT protocol was initially created in order to link sensors on oil pipelines with communications satellites, with an emphasis on minimal battery loss and bandwidth consumption.**
* **Since its inception, MQTT has continued to undergo development, with version 5.0 arriving in May 2018. Version 3.1.1 was submitted to the OASIS consortium in 2013 and accepted as an ISO standard.**

## **How Does MQTT Work?**

### **MQTT architecture**

**The connected devices in the MQTT protocol are known as “clients,” which communicate with a server referred to as the “broker.” The broker handles the task of data transmission between clients.**

**Whenever a client (known as the “publisher”) wants to distribute information, it will publish to a particular topic, the broker then sends this information to any clients that have subscribed to that topic (known as “subscribers”).**

**The publisher does not need any data on the number or the locations of subscribers. In turn, subscribers do not need any data about the publisher. Any client can be a publisher, subscriber, or both. The clients are typically not aware of each other, only of the broker that serves as the intermediary. This setup is popularly known as the “pub/sub model.”**

### **MQTT messages**

**When a client wants to send data to the broker, this is known as a “publish.” When a client wants to receive data from the broker, it will “subscribe” to a topic or topics. When a client subscribes to a certain topic, it will receive all messages published on that topic going forward.**

**Along with the message itself, the publisher also sends a QoS (Quality of Service) level. This level defines the guarantee of delivery for the message. These QoS levels are as follows:**

* **At most once: When the message is published, the broker will only receive the message “at most once.” This level should not be used for mission-critical information since it carries the risk that the subscribers will not receive the message.**
* **At least once: The publisher continues to resend the message until it receives an acknowledgment from the broker regarding the particular message. In other words, it’s more important that the message is received than it is to ensure it is only received once. This is probably the most commonly used QoS level.**
* **Exactly once: The publisher and broker work together to ensure the broker will receive and act on a message exactly once. This requires some additional overhead in the form of a four-part handshake. Although this is the safest QoS level, it is also the slowest and therefore only used when necessary.**

## **How Does MQTT Work in IoT Projects?**

**In this section, we’ll discuss how you can use MQTT in an IoT project, using one of our recent clients as an example.An automotive battery company wanted to offer “fresher” batteries to sell to its customers nationwide. This meant implementing a “first in, first out” strategy so that batteries wouldn’t sit on the shelf for too long.Of course, this required the company to track the dates of arrival of the stock on their shelves. In need of a trusted IoT partner, the company reached out to Very.**

**Very helped install IoT sensors on the company’s batteries and on their shelves. These sensors transmit information via MQTT to**[**Amazon Web Services (AWS)**](https://www.verypossible.com/insights/aws-iot-core-elixir-and-nerves-a-crash-course)**in the cloud. Each battery has a signal-emitting device that sends a Bluetooth signal to convey its presence on the rack. In addition, a single battery-powered hub wakes up once per day in order to transmit information to AWS using MQTT (as well as the TLS protocol for**[**secure transmission**](https://www.verypossible.com/insights/how-to-apply-security-best-practices-to-iot-firmware-development)**).**

**The MQTT protocol is the de-facto standard for IoT messaging. Standardized by OASIS and ISO, MQTT publish/subscribe protocol provides a scalable and reliable way to connect devices over the Internet. Today, MQTT is used by many companies to connect millions of devices to the Internet.**

### **Companies are using MQTT because:**

* **It requires minimal resources since it is lightweight and efficient**
* **Support bi-directional messaging between device and cloud**
* **Can scale to millions of connected devices**
* **Support reliable message delivery through 3 QoS levels**
* **Works well over unreliable networks**
* **Security enabled, so it works with TLS and common authentication protocols**

## **The Benefits of MQTT**

* **Lightweight code footprint: Devices need only a few lines of code in order to get up and running with the MQTT protocol.**
* **Minimized data packets: MQTT is very energy-efficient. This is great if a device is battery-powered or has little CPU power.**
* **Speed: MQTT operates in real time, with no delays outside of QoS.**
* **Ease of implementation: MQTT already has libraries in programming languages such as**[**Elixir**](https://www.verypossible.com/insights/what-is-elixir)**and**[**Python**](https://www.verypossible.com/insights/structuring-serverless-applications-with-python)**.**
* **Last will and testament: If a client unexpectedly disconnects, you can set message instructions to be sent to all subscribers in order to remedy the situation.**
* **Retained messages: Each topic can have one retained message that a client automatically receives when it subscribes (like a pinned post on social media).**

## **Common Alternatives to MQTT**

### **XMPP**

**XMPP (Extensible Messaging and Presence Protocol) is a communications protocol based on the XML language for storing and transporting data. It is frequently used to power instant messaging services such as Jabber.**

### **HTTP(S)**

**HTTP (Hypertext Transfer Protocol) and its extension HTTPS (Hypertext Transfer Protocol Secure) are communications protocols that are the foundation of the World Wide Web. However, they are stateless and carry more overhead per transmission than MQTT. In addition, HTTPS has a lower throughput than MQTT, meaning that you can’t send as many messages in the same period of time.**

## **Conclusion**

**MQTT plays a crucial role in making IoT projects more “low-lift” in terms of technical specifications while achieving the desired connections among devices, servers, and application**

## **The use of Python MQTT**

### **Connect to the MQTT broker**

**This article will use**[**the free public MQTT broker**](https://www.emqx.com/en/mqtt/public-mqtt5-broker)**provided by EMQ X. This service is based on**[**MQTT IoT cloud platform**](https://www.emqx.com/en/cloud)**to create. The accessing information of the broker is as follows:**

* **Broker: broker.emqx.io**
* **TCP Port: 1883**
* **Websocket Port: 8083**

#### **Import the Paho MQTT client**

**from paho.mqtt import client as mqtt\_client**

#### **Set the parameter of MQTT Broker connection**

**Set the address, port and topic of MQTT Broker connection. At the same time, we call the Python function random.randint to randomly generate the MQTT client id.**

**broker = 'broker.emqx.io'**

**port = 1883**

**topic = "/python/mqtt"**

**client\_id = f'python-mqtt-{random.randint(0, 1000)}'**

**# username = 'emqx'**

**# password = 'public'**

#### **Write the MQTT connect function**

**Write the connect callback function on\_connect. This function will be called after connecting the client, and we can determine whether the client is connected successfully according to rc in this function. Usually, we will create an MQTT client at the same time and this client will connect to broker.emqx.io.**

**def connect\_mqtt():**

**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)**

**# Set Connecting Client ID**

**client = mqtt\_client.Client(client\_id)**

**client.username\_pw\_set(username, password)**

**client.on\_connect = on\_connect**

**client.connect(broker, port)**

**return client**

### **Publish messages**

**First, we define a while loop. In this loop, and we will set the MQTT client publish function to send messages to the topic /python/mqtt every second.**

**def publish(client):**

**msg\_count = 0**

**while True:**

**time.sleep(1)**

**msg = f"messages: {msg\_count}"**

**result = client.publish(topic, msg)**

**# result: [0, 1]**

**status = result[0]**

**if status == 0:**

**print(f"Send `{msg}` to topic `{topic}`")**

**else:**

**print(f"Failed to send message to topic {topic}")**

**msg\_count += 1**

### **Subscribe to messages**

**Write the message callback function on\_message. This function will be called after the client received messages from the MQTT Broker. In this function, we will print out the name of subscribed topics and the received messages.**

**def subscribe(client: mqtt\_client):**

**def on\_message(client, userdata, msg):**

**print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")**

**client.subscribe(topic)**

**client.on\_message = on\_message**

### **The complete code**

**The code of publishing messages**

**# python 3.6**

**import random**

**import time**

**from paho.mqtt import client as mqtt\_client**

**broker = 'broker.emqx.io'**

**port = 1883**

**topic = "python/mqtt"**

**# generate client ID with pub prefix randomly**

**client\_id = f'python-mqtt-{random.randint(0, 1000)}'**

**# username = 'emqx'**

**# password = 'public'**

**def connect\_mqtt():**

**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)**

**client = mqtt\_client.Client(client\_id)**

**client.username\_pw\_set(username, password)**

**client.on\_connect = on\_connect**

**client.connect(broker, port)**

**return client**

**def publish(client):**

**msg\_count = 0**

**while True:**

**time.sleep(1)**

**msg = f"messages: {msg\_count}"**

**result = client.publish(topic, msg)**

**# result: [0, 1]**

**status = result[0]**

**if status == 0:**

**print(f"Send `{msg}` to topic `{topic}`")**

**else:**

**print(f"Failed to send message to topic {topic}")**

**msg\_count += 1**

**def run():**

**client = connect\_mqtt()**

**client.loop\_start()**

**publish(client)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**run()**

**The code of subscribing to messages**

**# python3.6**

**import random**

**from paho.mqtt import client as mqtt\_client**

**broker = 'broker.emqx.io'**

**port = 1883**

**topic = "python/mqtt"**

**# generate client ID with pub prefix randomly**

**client\_id = f'python-mqtt-{random.randint(0, 100)}'**

**# username = 'emqx'**

**# password = 'public'**

**def connect\_mqtt() -> mqtt\_client:**

**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)**

**client = mqtt\_client.Client(client\_id)**

**client.username\_pw\_set(username, password)**

**client.on\_connect = on\_connect**

**client.connect(broker, port)**

**return client**

**def subscribe(client: mqtt\_client):**

**def on\_message(client, userdata, msg):**

**print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")**

**client.subscribe(topic)**

**client.on\_message = on\_message**

**def run():**

**client = connect\_mqtt()**

**subscribe(client)**

**client.loop\_forever()**

**if \_\_name\_\_ == '\_\_main\_\_':**

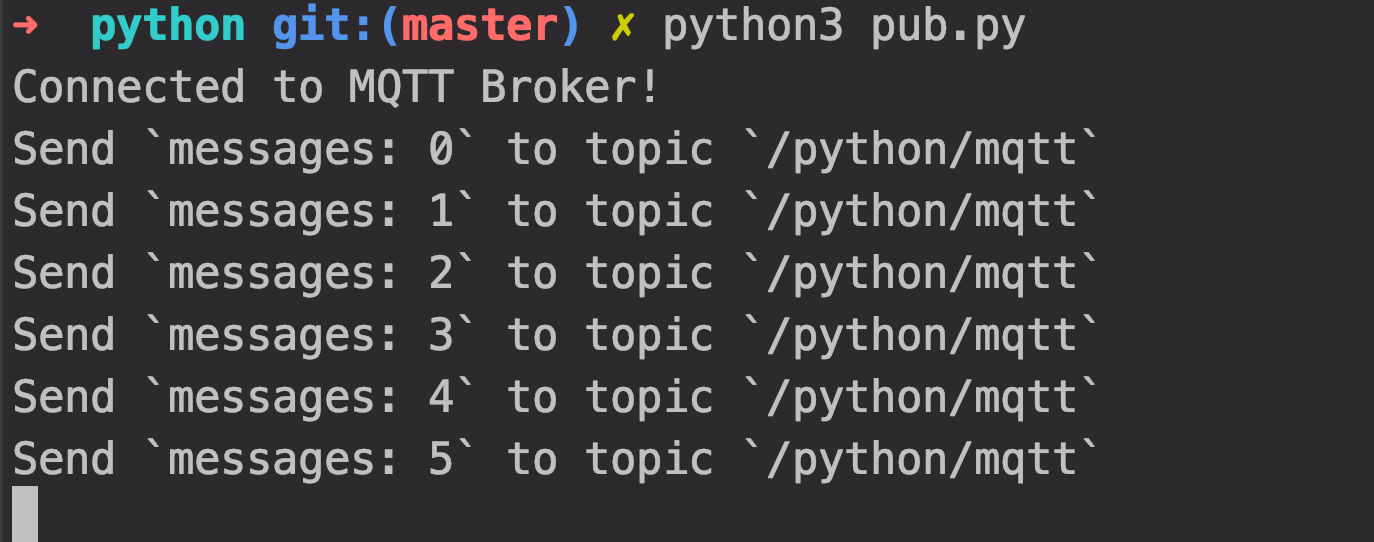
**run()**

## **Test**

#### **Publish messages**

**Run the code of publishing messages, we will see that the client connects successfully and publishes messages successfully**

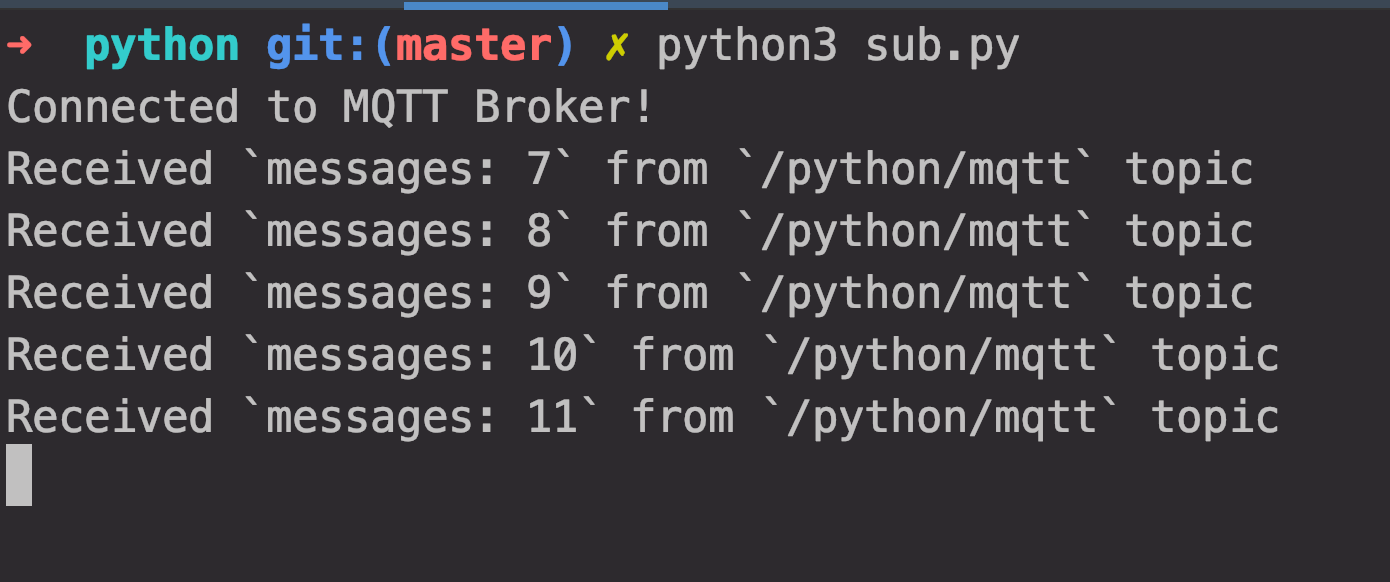
**python3 pub.py**

****

#### **Subscribe to messages**

**Run the code of subscribing to messages, we will see that the client connects successfully and receives the published messages successfully**

**python3 sub.py**

****