Lab3

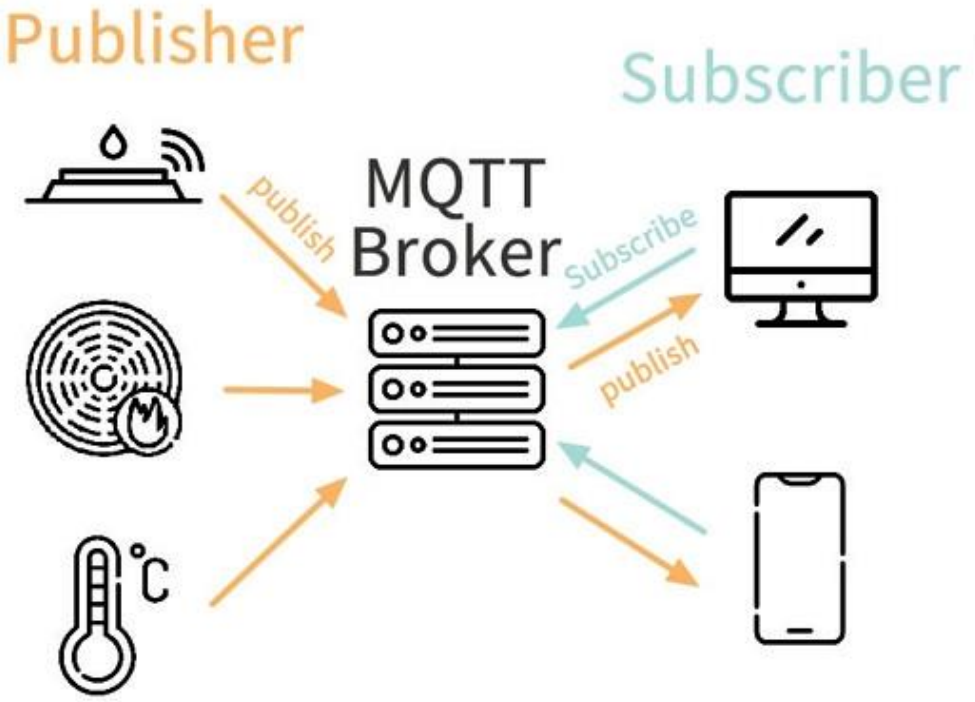
105072123 黃海茵

- **MQTT 運作流程**
  - ✧ **角色**
    1. 分為 Publisher、Subscriber、Broker 三個角色
    2. Publisher 與 Subscriber 之間有 Broker 當作中繼站，所以 Publisher 與 Subscriber 不需要知道彼此的 IP 位址
    3. Publisher 與 Subscriber 之間透過 Topic 來取得需要的資訊

  - ✧ **動作**
    1. Publisher 向 Broker 發送 Publish
    2. Subscriber 向 Broker 發送 Subscribe

  - ✧ **運作方式**

    Subscriber 會向 Broker Subscribe 特定的 Topic，Broker 會判斷 Publisher 所發布的特定 Topic 是否有 Subscriber 訂閱此特定 Topic，如有的話會將此特定 Topic 的訊息傳送給 Subscriber



- QoS 參數
  - ✧ **0： 最多一次傳送（只負責傳送，發送過後就不管數據的傳送情況）**
    當 Publisher 在發送訊息時，指定本次發送 Qos 為 0，則 Broker 可能會沒收到訊息；而 Broker 若有接到訊息，發佈給 Subscriber 時，Subscriber 也可能沒收到訊息。

  - ✧ **1： 至少一次傳送（確認數據交付）**
    當 Publisher 指定 Qos 為 1，Broker 收到訊息後會回傳 PUBACK Message 給 Publisher。Publisher 在發佈訊息後，若沒收到回傳的 PUBACK Message，一定時間後會再重新傳一次相同內容的訊息給 Broker（不論 Broker 是否有到訊息），也因此可能造成 Subscriber 收到的訊息重複。

  - ✧ **2： 正好一次傳送（保證數據交付成功）**
    當 Publisher 指定 Qos 為 2，Broker 為了避免發生 Qos 為 1 時，收到重複訊息的情況，Broker 與 Publisher 會進行兩次的來回確認才將訊息傳遞下去。

MQTT 服務品質 (QoS) 選項

AWS IoT 和 AWS IoT 裝置軟體開發套件支援 MQTT 服務品質 (QoS) 層級 0 和 1。MQTT 通訊協定定義 QoS 的第三個層級，層級 2，但 AWS IoT 不支援。只有 MQTT 通訊協定支援 QoS 功能。HTTPS 不支援 QoS。

此表格說明各個 QoS 層級如何影響訊息發佈至中介裝置的方式，以及訊息中介裝置發佈訊息的方式。

具有 QoS 層級...	訊息為...	評論
QoS 層級 0	傳送零次或更多次	此層級應用於透過可靠通訊連結傳送的訊息，或者可能會錯過沒有問題的訊息。
QoS 層級 1	至少傳送一次，然後重複，直到接收到 PUBACK 回應	在寄件者收到 PUBACK 回應以表示成功傳遞之後，才會將訊息視為完整的。

- **程式流程**
  - ✧ Arduino

```
void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  AT.begin(9600);
  dht.begin();

  command("AT+CSQ", 2000);
  command("AT+QSCLK=0", 2000);
  command("AT+QBAND=1,8", 2000);
  command("AT+CFUN=1", 2000);
  command("AT+CGSN=1", 2000);
  command("AT+CIMI", 2000);
  command("AT+CGATT=0", 2000);
  command("AT+CGDCONT=1,\"IPV4V6\"", 2000);
  command("AT+CGATT=1", 5000);
  command("AT+QICFG=\"dataformat\",1,1", 5000);
  command("AT+QIOPEN=1,0,\"TCP\",140.114.78.132,16619,0,0,0", 5000);

  float h = dht.readHumidity();
  String combine = String(h) + "," + ledState;
  combine = "AT+QISENDEX=0,7," + toHEX(combine);
  const char *result = combine.c_str();
  command(result, 5000);
  String payload = command("AT+QIRD=0,512", 5000);
```

首先是和 server 建立連線，並且傳濕度和 led state 過去，這部分和上次 lab 大同小異，就不贅述。

```
void decodePayload(String str) {
  int cnt = 0;
  String result = "";
  for(int i=0; i<str.length(); i++){
    if(cnt==2)
      result += str[i];
    if(cnt==3)
      break;
    if(str[i] == '\n')
      cnt++;
  }
  Serial.print("Payload: " + result);
  toString(result);

  return;
}
```

這次主要就是多寫了兩個 function，decodePayload 和 toString。decodePayload 用來把 server 回傳的 payload 擷取出需要的部分，然後丟進 toString 中轉成字元。

```
void toString(String hex) {
  int ascii = 0;
  for(int i=0; i<hex.length(); i++){
    if(i%2 == 0){
      if(i > 0)
        Serial.print(char(byte(ascii)));
        ledState = char(byte(ascii) - '0');
      ascii = 0;
    }
    else
      ascii *= 16;

    if ((hex[i]>='0') && (hex[i]<='9'))
      ascii +=  hex[i]-'0';
    if ((hex[i]>='A') && (hex[i]<='F'))
      ascii += hex[i]+10-'A' ;
  }

  return;
}
```

toString 用來把十六進位的 ASCII 碼轉回字元，再存到 ledState 中。

✧ server

```
def mqttcallback(client, userdata, message):
    global currentRing,conn,addr,Lock
    try:
        # [TODO] write callback to deal with MQTT message from Lambda
        response = message.payload.decode("utf-8")
        if 'desired' in json.loads(response)['state']:
            currentRing = int(json.loads(response)['state']['desired']['ring'])
            conn.send(("ring:" + str(currentRing)).encode("utf-8"))
    except Exception as e:
        print(e)
```

這個 function 實作 server 收到 AWS 的狀態，decode 之後，取出字串中有用的部分，再 encode 並轉傳到 BC20。

```
# [TODO] Define ENDPOINT, CLIENT_ID, PATH_TO_CERT, PATH_TO_KEY, PATH_TO_ROOT
ENDPOINT = "a2bu30yj4l66iy-ats.iot.us-east-2.amazonaws.com"
CLIENT_ID = "lab3"
PATH_TO_CERT = "./2398bbc3bc-certificate.pem.crt"
PATH_TO_KEY = "./2398bbc3bc-private.pem.key"
PATH_TO_ROOT = "./root CA 1.txt"
```

```
# [TODO] subscribe AWS topic(s)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/get", 1, mqttcallback)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/accepted", 1, mqttcallback)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/rejected", 1, mqttcallback)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/update", 1, mqttcallback)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/rejected", 1, mqttcallback)
myAWSIoTMQTTClient.subscribe("$aws/things/+/shadow/delta", 1, mqttcallback)
```

這兩部分把原本直接在 MQTT 中手動輸入的資料，自動執行。

```
def on_new_client(clientsocket,addr):
    global currentRing
    while True:
        # [TODO] decode message from Arduino and send to AWS
        recv = clientsocket.recv(7).decode("utf-8")
        print("recv: " + recv)
        humidity = float(recv[0:5])
        currentRing = int(recv[6])
        send = json.dumps({
            "state":{
                "reported":{
                    "humidity":humidity,
                    "ring":currentRing
                }
            }
        })
        myAWSIoTMQTTClient.publish("$aws/things/lab3/shadow/update", send, 1)
    clientsocket.close()
```

這個 function 實作 server 收到 arduino 傳來的資料，decode 後轉成正確的格式，再送到 AWS。

- **遇到困難及解決方式**

  我覺得最近這兩次的 lab，最麻煩也弄比較久的部分都是在字串轉換、處理，每次想要嘗試找一些內建的東西來做，然後都找不到，最後都直接自己寫一個 function，用最原始的方式跑，不知道有沒有更好的方法（至今還沒找到 ☹）

- **操作截圖**

```
105072123@user-System-Product-Name:~/Lab3_CA$ python3 AWS_relay.py
server start at: 0.0.0.0:16619
wait for connection...
connected by ('223.140.66.19', 40933)
recv: 70.20,0
```

```
Sending: AT+QIRD=0,512
Received: AT+QIRD=0,512
+QIRD: 6
72696E673A31

OK

Payload: 72696E673A31
ring:1
```