MQTT paho client with python

I'm not going to provide the solution to the exercise, as this would give away part of the solution for the graded assignment, but I'll give you a working python application that should show you have to use the Paho mqtt client library from python.

You can use the following links to help understand the code below:
- Connecting to the Watson IoT platform using MQTT
- MQTT messaging for the Watson IoT platform
- Python paho-mqtt package documentation

To start with you need to install the client library, which can be done on the command line by issuing the following command:

```
pip install paho-mqtt
```

Then enter the following code and save to file mqttexample.py. Modify the connection details to match your environment and device registration as indicated by the comments

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

# UPDATE THE FOLLOWING TO MATCH YOUR ENVIRONMENT
# ----------
# Gateway details
orgID = 'abc123'
deviceType = 'rasPi'
deviceID = 'rasPi01'
deviceToken = 'passw0rd'
# -----------
# Device details when publishing/subscribing as gateway
# Here the gateway device type and id are used to send data from gateway device
devType = 'rasPi'
devID = 'rasPi01'
# THE CODE BELOW WON'T NEED CHANGING

deviceAuth = 'use-token-auth'
host = orgID + '.messaging.internetofthings.ibmcloud.com'
# clientID must follow platform format or will fail to connect:
# If you are connecting as a device use the following clientID
# clientID = 'd:' + orgID + ':' + deviceType + ':' + deviceID
# If you are connecting as a gateway use the following clientID
clientID = 'g:' + orgID + ':' + deviceType + ':' + deviceID

#Topics must follow platform format (see docs for details)
#Publish and subscribe as a device
# publishTopic = 'iot-2/evt/status/fmt/json'
# commandTopic = 'iot-2/cmd/display/fmt/json'
#Publish and subscribe as a gateway
publishTopic = 'iot-2/type/'+devType+'/id/'+devID+'/evt/status/fmt/json'
commandTopic = 'iot-2/type/'+devType+'/id/'+devID+'/cmd/display/fmt/json'

messageToPublish = {'d' : { 'temp' : 25.0, 'hum' : 49} }
connected = False

def on_connect(client, userdata, flags, rc):
    global connected
    print("client commected:", rc)
    connected = True
```

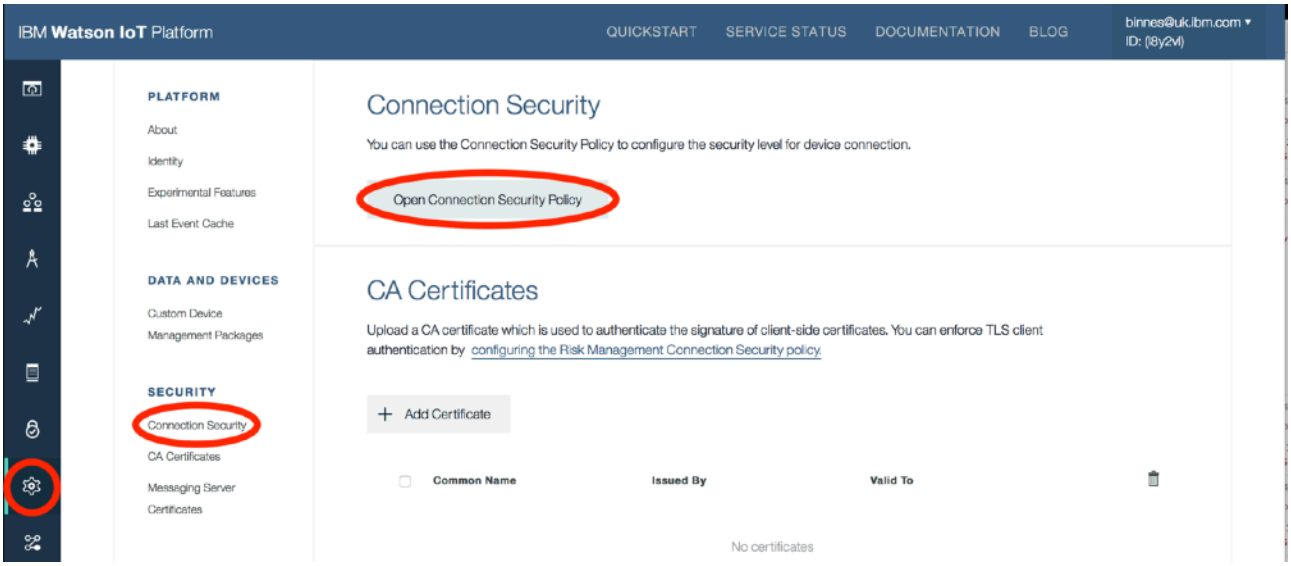
```
def on_publish(client, userdata, mid):
    print("message published")

#This is the callback when a message is received
def on_message(client, userdata, message):
    print("message received=", str(message.payload.decode("utf-8")))
    print("message topic=", message.topic)

#Create the client, set the authentication details and register callbacks
client = mqtt.Client(clientID, protocol=mqtt.MQTTv311)
client.username_pw_set(deviceAuth, deviceToken)
client.on_connect = on_connect
client.on_publish = on_publish
client.on_message = on_message

#Connect the client and start the handler loop
client.connect(host, keepalive=60)
client.loop_start()
while (False == connected):
    time.sleep(1)

#subscribe to receive commands
client.subscribe(commandTopic, qos=1)
for i in range(5):
    client.publish(publishTopic, payload=json.dumps(messageToPublish), qos=1,
retain=False)
    time.sleep(2)
time.sleep(20)
#stop the handler loop
client.loop_stop()
```

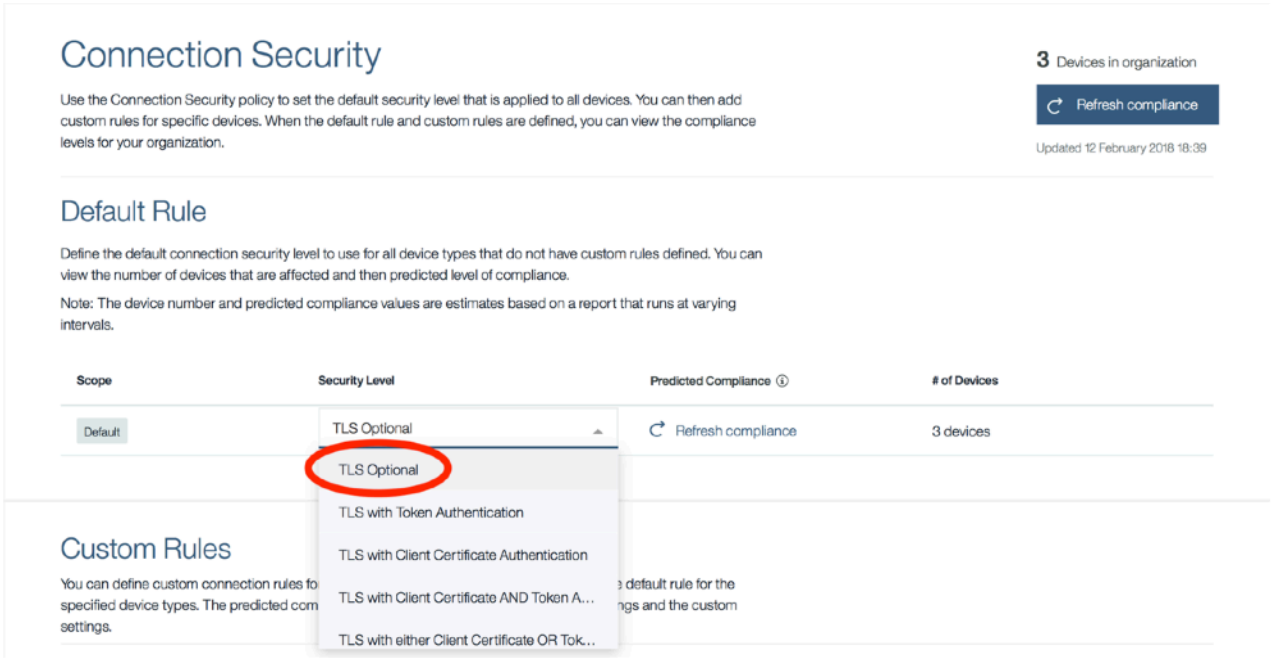
When you run this code you may be able to connect or you may get a return code 5 - not authorised:

```
python mqttexample.py
('client commected:', 5)
```

By default the IBM platform IoT service requires that you use TLS secured connections, so the code above will only create an unsecured connection, which is not authorised.

To test the above code you can configure your IoT service to accept insecure connections. To do this navigate to the IoT console in the IBM Cloud web user interface, select the settings section then Connection Security and finally Open Connection Security.





Select “TLS Optional” for the default scope Security Level. Save the change then try the application again and you should now be able to connect to the IoT platform and see the messages being published from the device.

Enabling TLS

You do not want to be running your IoT platform without secure connections, so you should enable TLS in your applications. To do this you need to download the server certificate from here and place the file messagiing.pem in the same directory as the python file.

When you have the certificate you can then modify the application as shown below - the changed lines are in **bold:**

```
import time, json, os, ssl
import paho.mqtt.client as mqtt

# UPDATE THE FOLLOWING TO MATCH YOUR ENVIRONMENT
# ----------
# Gateway details
orgID = 'abc123'
deviceType = 'rasPi'
deviceID = 'rasPi01'
deviceToken = 'passw0rd'
# -----------
# Device details when publishing/subscribing as gateway
devType = 'rasPi'
devID = 'rasPi01'
# THE CODE BELOW WON'T NEED CHANGING

deviceAuth = 'use-token-auth'
host = orgID + '.messaging.internetofthings.ibmcloud.com'
# clientID must follow platform format or will fail to connect:
# If you are connecting as a device use the following clientID
# clientID = 'd:' + orgID + ':' + deviceType + ':' + deviceID
# If you are connecting as a gateway use the following clientID
clientID = 'g:' + orgID + ':' + deviceType + ':' + deviceID

# Server certificate file
caFile = os.path.dirname(os.path.abspath(__file__)) + "/messaging.pem"

#Topics must follow platform format (see docs for details)
```

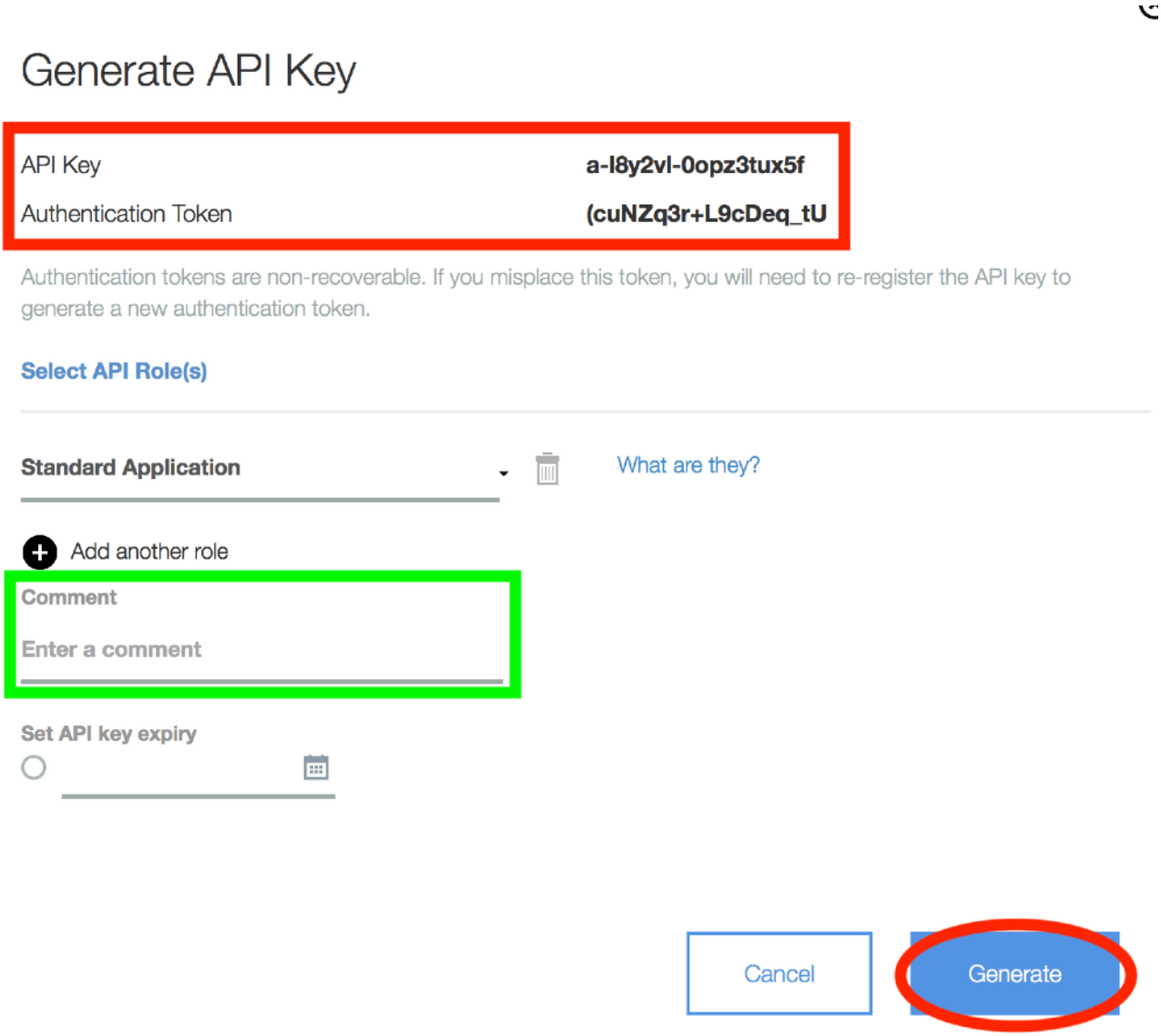
```
#Publish and subscribe as a device
# publishTopic = 'iot-2/evt/status/fmt/json'
# commandTopic = 'iot-2/cmd/display/fmt/json'
#Publish and subscribe as a gateway
publishTopic = 'iot-2/type/'+devType+'/id/'+devID+'/evt/status/fmt/json'
commandTopic = 'iot-2/type/'+devType+'/id/'+devID+'/cmd/display/fmt/json'

messageToPublish = {'d' : { 'temp' : 25.0, 'hum' : 49} }
connected = False

def on_connect(client, userdata, flags, rc):
    global connected
    print("client commected:", rc)
    connected = True

def on_publish(client, userdata, mid):
    print("message published")

#This is the callback when a message is received
def on_message(client, userdata, message):
    print("message received=", str(message.payload.decode("utf-8")))
    print("message topic=", message.topic)

#Create the client, set the authentication details and register callbacks
client = mqtt.Client(clientID, protocol=mqtt.MQTTv311)
client.username_pw_set(deviceAuth, deviceToken)
client.tls_set(ca_certs=caFile, certfile=None, keyfile=None,
               cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2)
client.on_connect = on_connect
client.on_publish = on_publish
client.on_message = on_message

#Connect the client and start the handler loop
client.connect(host, port=8883, keepalive=60)
client.loop_start()
while (False == connected):
    time.sleep(1)

#subscribe to receive commands -
client.subscribe(commandTopic, qos=1)
for i in range(5):
    client.publish(publishTopic, payload=json.dumps(messageToPublish), qos=1,
retain=False)
    time.sleep(2)
time.sleep(20)
#stop the handler loop
client.loop_stop()
```

The code above will connect with a secure connection, so you can go and reconfigure your IoT platform to require **TLS with token Authentication** fo the default connection security rule.

MQTT client for a Node.js connecting as an application

To test the above python code I've created a small Node.JS applications, which connects as an application to the Watson IoT platform. An application authenticates to the platform using an API key, which you must generate from the Watson IoT platform console. To generate a key navigate to the Apps section, then press the Generate API Key button. This will generate a new key.

You need to record the API Key and Authentication Token, as the Authentication Token content cannot be recovered once the Generate button is pressed. You can also add a comment to identify where the key is being used and there are also options to set a validity period for the key of a key should only remain valid for a specific period:



Once you have the key then create a new folder and copy the code below. You will need a copy of the messaging.pem certificate file which you downloaded for the python application and then there are 2 files to create, package.json and server.js:

package.json:

```
{
        "name": "mqttNode",
        "version": "0.0.1",
        "description": "Sample code to demonstrate MQTT connection to Watson IoT
platform",
        "main": "server.js",
        "dependencies": {
                "mqtt": "^2.8.2"
        }
}
```

server.js:

```
var fs = require('fs');
var mqtt = require('mqtt')

// UPDATE THE FOLLOWING TO MATCH YOUR ENVIRONMENT
var orgID = 'abc123';
var applicationID = 'testApp'; // user defined string
var applicationKey = 'a-abc123-n7vh9a5d8y'; // Application key
var applicationToken = '5qY!rfb?oVFBWBM?)M';
//  ------------
var devType = 'rasPi'
var devID = 'rasPi01'
// THE CODE BELOW WON'T NEED CHANGING


var CA = fs.readFileSync(__dirname + '/messaging.pem');
var PORT = 8883;
var HOST = orgID+'.messaging.internetofthings.ibmcloud.com';
var PROTOCOL = 'ssl';
var CLIENTID = 'a:'+orgID+':'+applicationID;
var SERVER = PROTOCOL+'://'+HOST+':'+PORT;

var SUBTOPIC = 'iot-2/type/'+devType+'/id/'+devID+'/evt/+/fmt/json';
var CMDTOPIC = 'iot-2/type/'+devType+'/id/'+devID+'/cmd/display/fmt/json'

var options = {
  port: PORT,
  host: HOST,
  protocolId: 'MQTT',
  protocolVersion: 4,
  clientId: CLIENTID,
  ca: CA,
  keepalive: 60,
  username: applicationKey,
  password: applicationToken,
  rejectUnauthorized: true,
  clean: true
};

console.log("Connecting to " + SERVER +' using client ID '+CLIENTID);
var client = mqtt.connect(SERVER, options);


client.on('connect', function () {
  console.log("Connected");
  client.subscribe('iot-2/type/+/id/+/evt/+/fmt/json');
});

client.on('message', function (topic, message) {
```

```
  console.log('Message received on topic '+topic+' : '+message.toString());
  client.publish(CMDTOPIC,
      new Buffer(JSON.stringify({'screen':JSON.parse(message).d.temp})));
});

client.on('error', function(error) {
  console.log("Error : " + error.toString());
});
```

You need to update the variables at the top of server.js to match your environment. The values for applicationKey and applicationToken are from the API key you generated. The devType and devID should match the values you used in the python application, then whenever the python application publishes a message the Node.JS application will send a command back to the python application.

To run the Node.JS application you need to run **npm install** from a command line within the folder containing the package.json file, this will download the necessary prerequisite packages, then run the application using **npm start.**

With the Node.JS application running you can now open up another command window and run the python application, where you should see the 2 applications communicating via the Watson IoT platform, with all connections to the platform being secured TLS connections.