“

IoT Blog

Future is already here!

UNCATEGORIZED

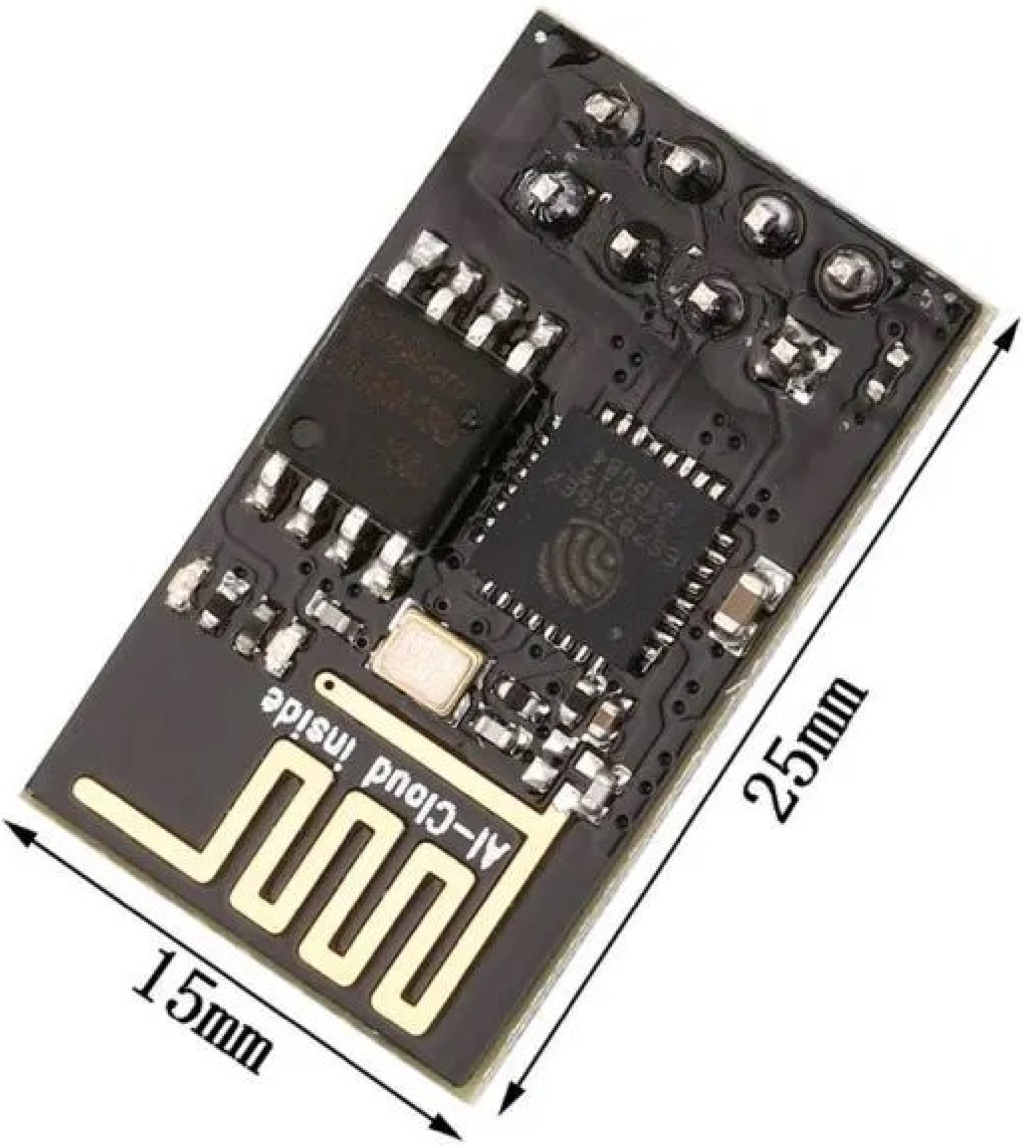
Connect ESP 8266 to AWS MQTT using Miropython

Date: January 10, 2019

This post provides the steps to connect Amazon AWS MQTT using Micropython running on ESP 8266.

— THE PROCEDURE APPLICABLE TO THE NODEMCU DEVELOPMENT MODULES AND ESP32.

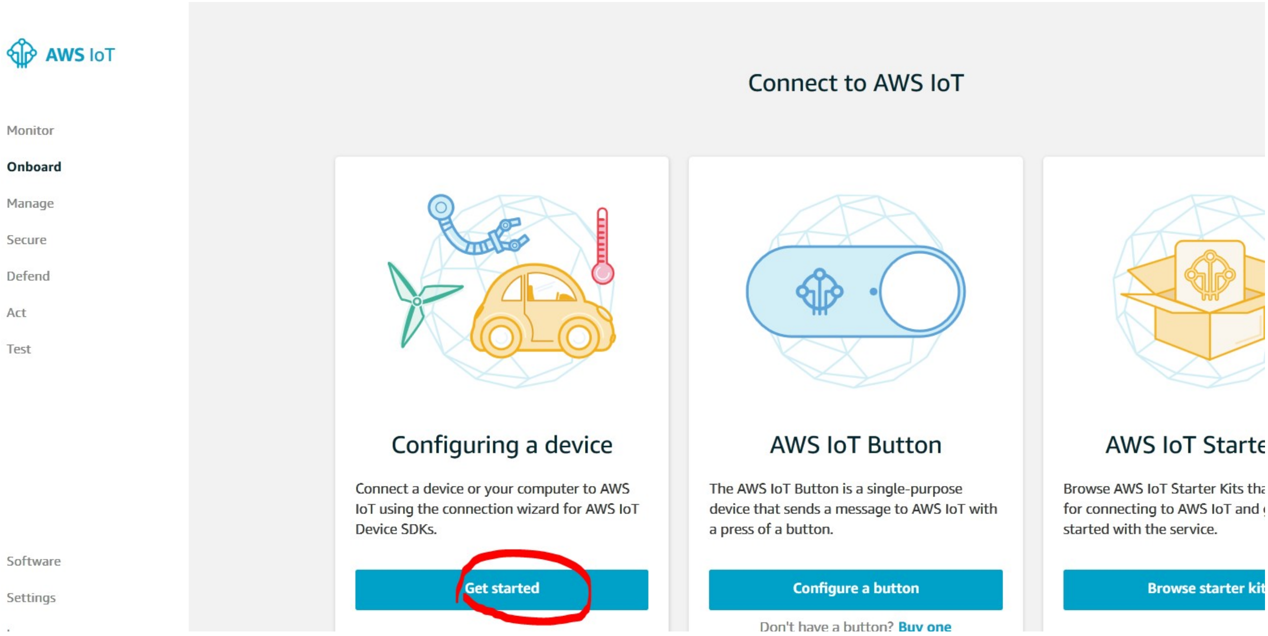
ESP 8266-01 is a one of the least expensive WiFi-enabled ESP modules: https://en.wikipedia.org/wiki/ESP8266. There are many similar 8266 based modules with integrated USB adapter like NodeMCU a little bit larger in size.



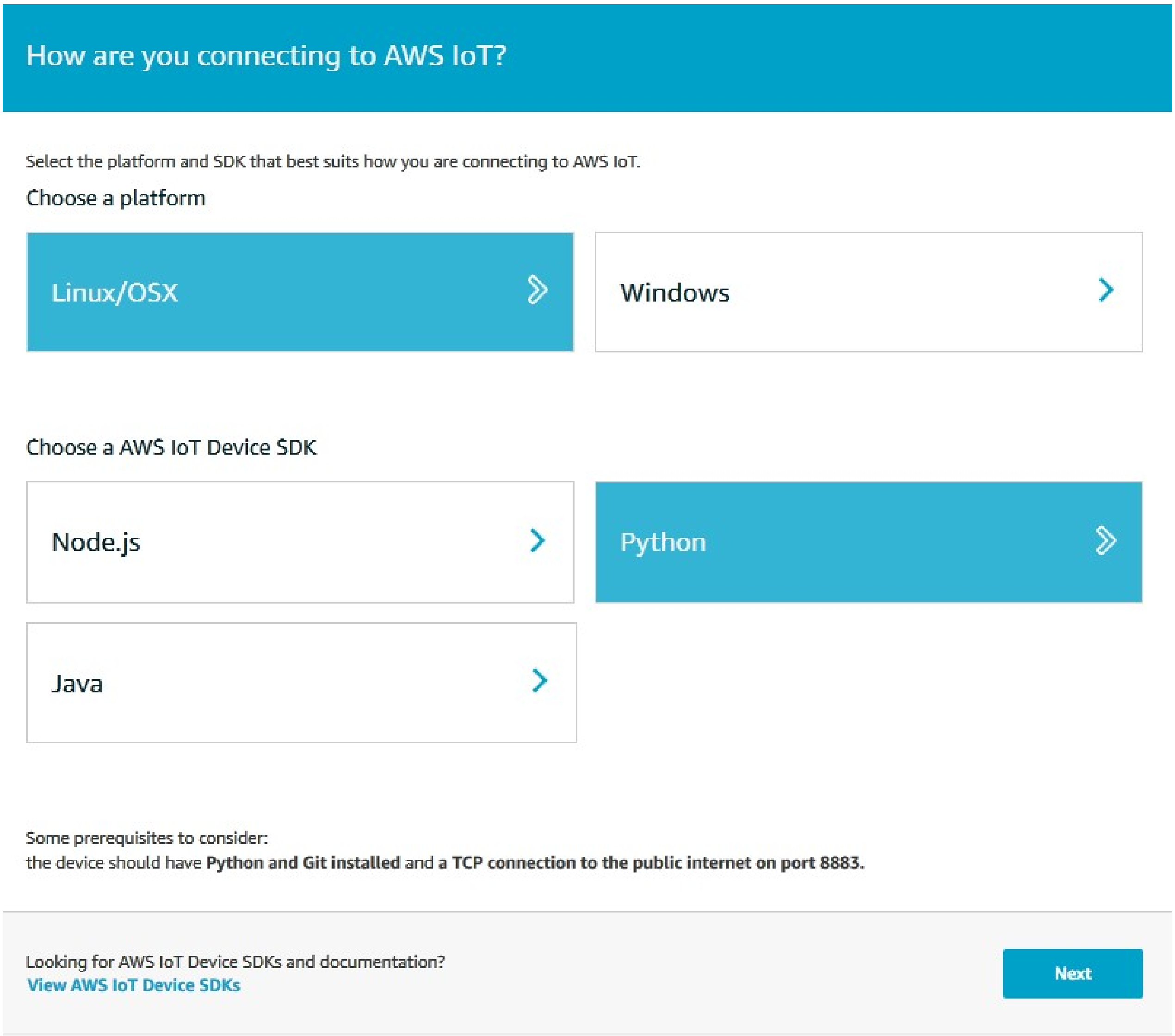
8266 is quite capable running all different kinds of applications. One area where the functionality is still limited is in secured communications. There are many examples on the internet how to connect ESP 8266 to generic MQTT using plan sockets, web sockets, or simplified TLS connection. This doesn't work directly with Amazon AWS MQTT. There is a way is to connect using the Arduino-specific instructions from here: https://www.youtube.com/watch?v=WJuXv5usXcY. This is a valid workaround but it is using intermediate API with shared API secret key. This limit the application to AWS API only. In this post we will connect to AWS MQTT directly using x.509 client certificate. The same approach may be useful for other secured MQTT providers.

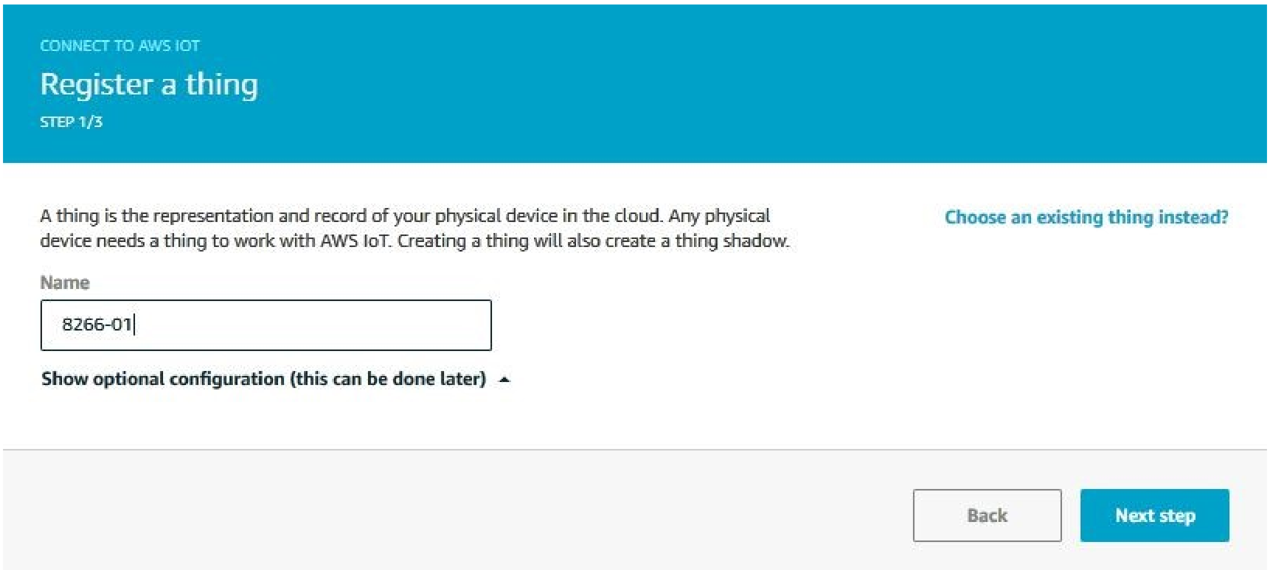
Let's start form creation AWS IoT thing. If you don't have AWS account it's easy sign up

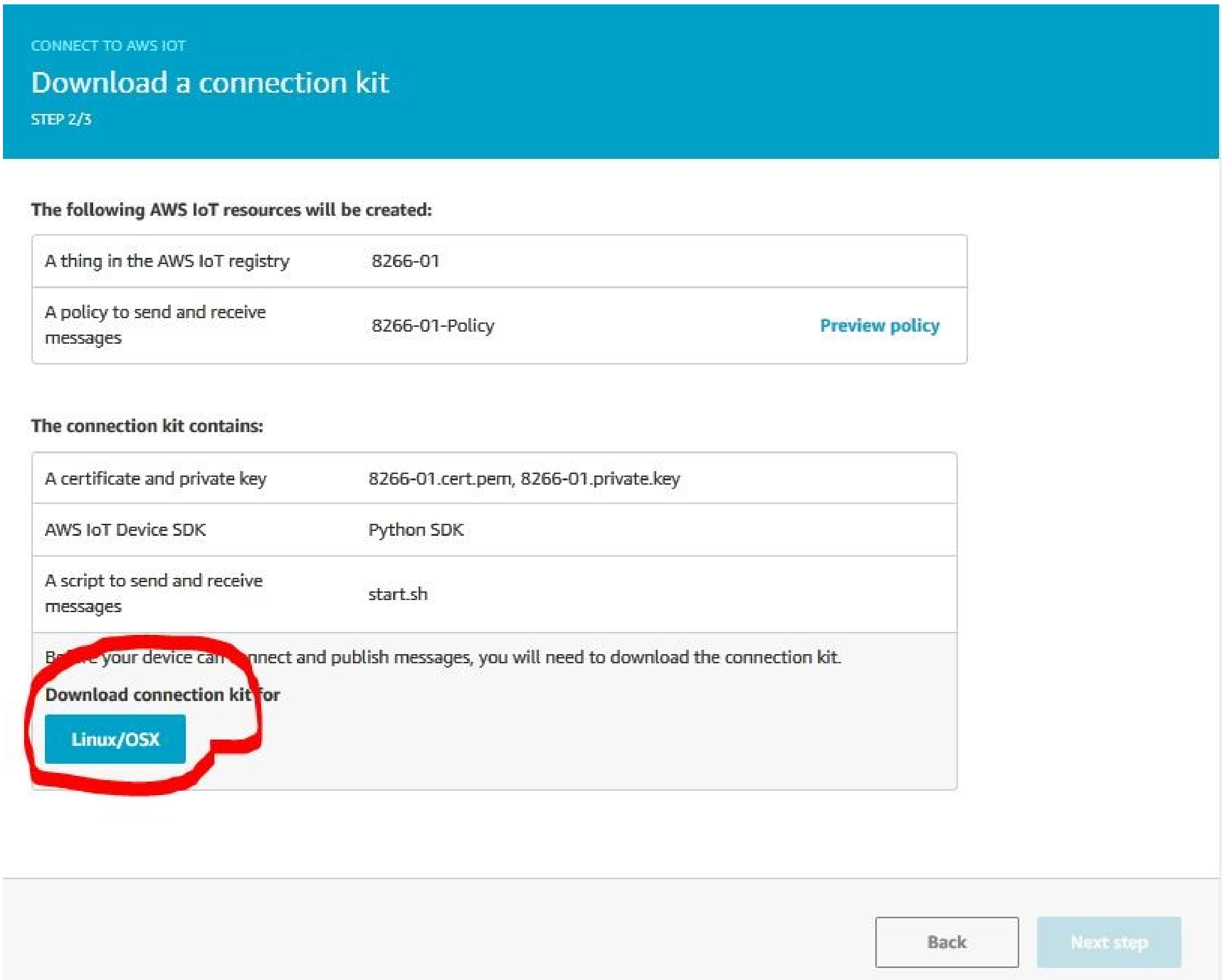
Go to your AWS console https://xxxxxxxxxxxx.signin.aws.amazon.com/console and select **IoT->Onboard**.

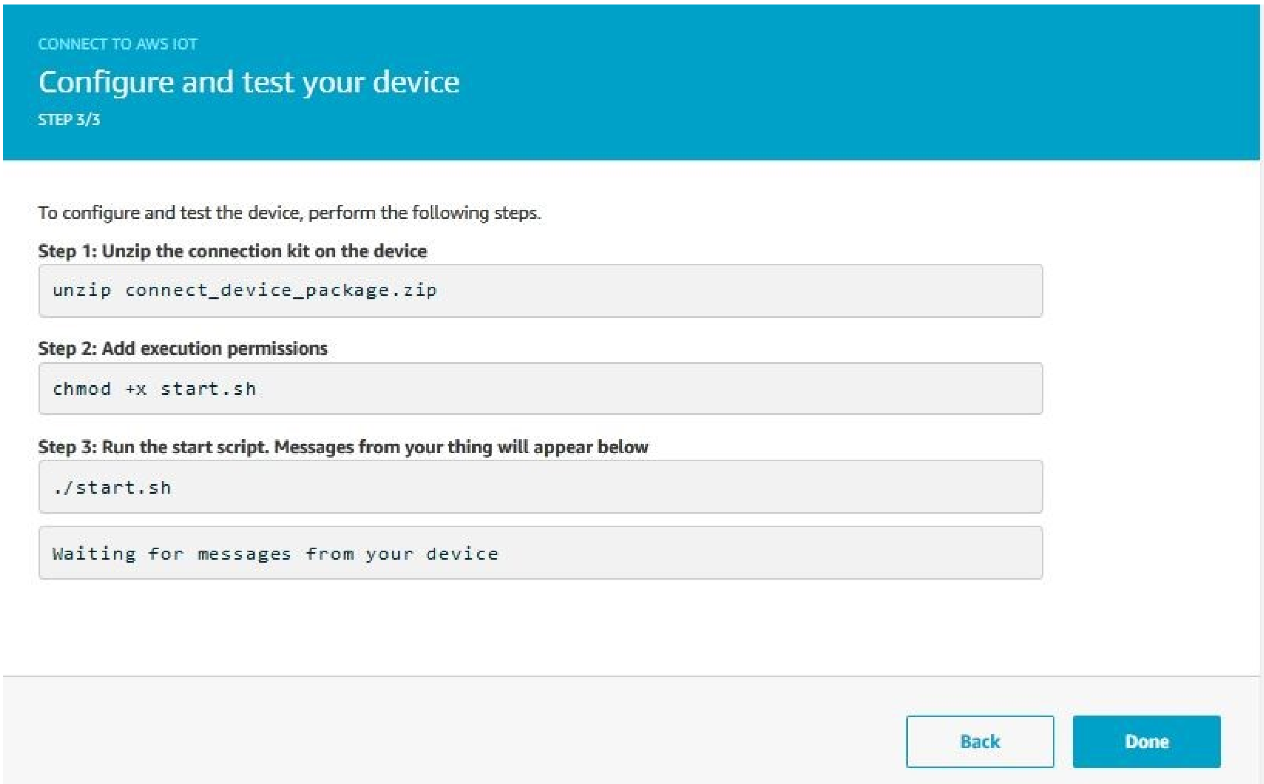


Select your OS and Python connection kit to speed-up the cloud thing registration.




CONNECT TO AWS IOT
Register a thing
STEP 1/3
A thing is the representation and record of your physical device in the cloud. Any physical device needs a thing to work with AWS IoT. Creating a thing will also create a thing shadow.
Choose an existing thing instead?
Name
8266-01
Show optional configuration (this can be done later)
Back
Next step


CONNECT TO AWS IOT
Download a connection kit
STEP 2/3
The following AWS IoT resources will be created:
A thing in the AWS IoT registry
8266-01
A policy to send and receive messages
8266-01-Policy
Preview policy
The connection kit contains:
A certificate and private key
8266-01.cert.pem, 8266-01.private.key
AWS IoT Device SDK
Python SDK
A script to send and receive messages
start.sh
your device can
nnect and publish messages, you will need to download the connection kit.
Download connection kit for
Linux/OSX
Back
Next step



When you clicked "Download Connection Kit" button you will get zip file "connect_device_package.zip" which contains AWS MQTT X.509 client certificate, private key, AWS Root CA cert, and a script to poll the AWS MQTT Python client source from the github.com . There are essentially four lines in the script:

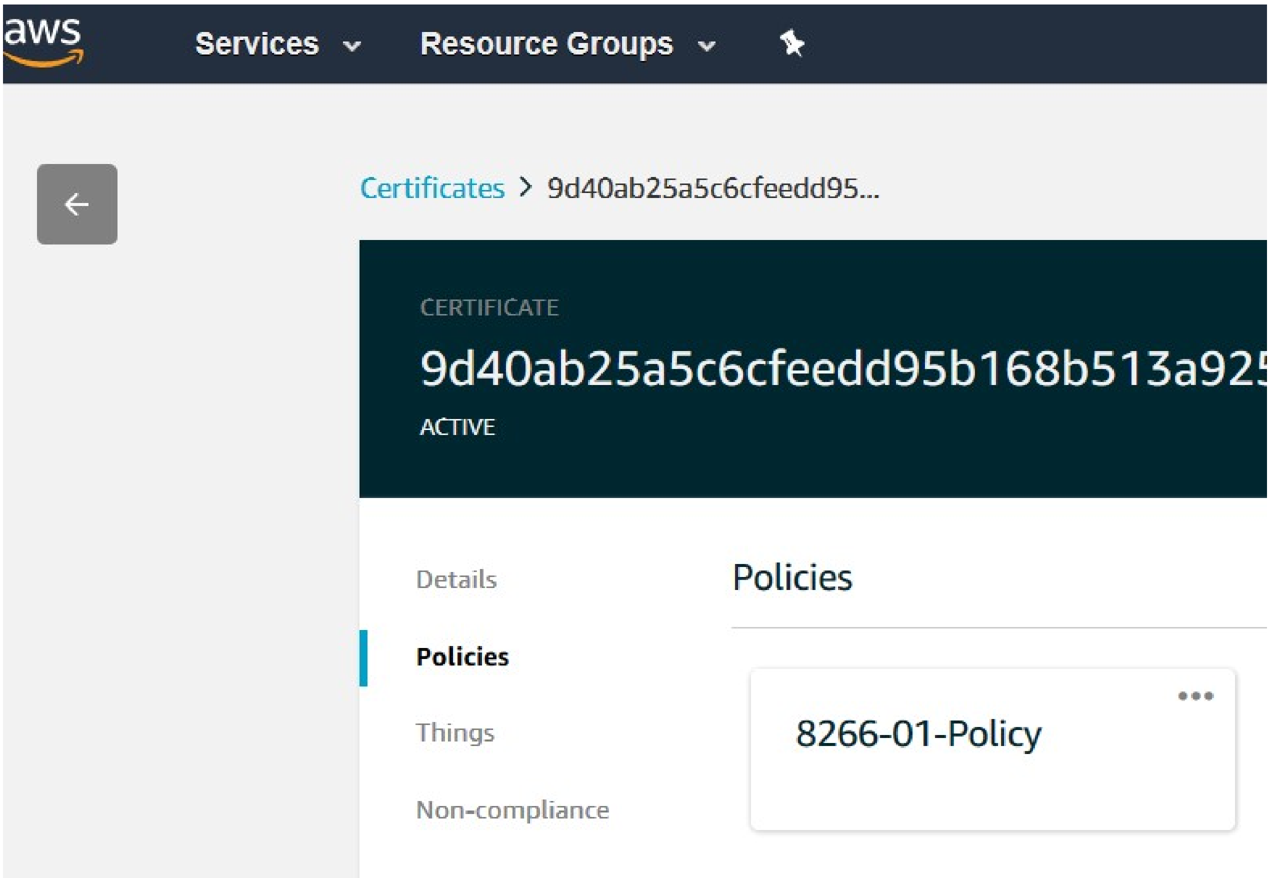
git clone https://github.com/aws/aws-iot-device-sdk-python.git

python setup.py install

curl https://www.amazontrust.com/repository/AmazonRootCA1.pem > root-CA.crt

python aws-iot-device-sdk-python/samples/basicPubSub/basicPubSub.py -e xxxxxxxxxxxxxx-ats.iot.us-east-2.amazonaws.com -r root-CA.crt -c 8266-01.cert.pem -k 8266-01.private.key

On AWS IoT side there is new IoT thing has been created, called "8266-01" or whatever name you have used. Also AWS has created client cert for the new IoT thing and a new security policy with permissions publish/subscribe/update shadow. You can check the policy document on AWS and harden it if needed.



Once the script has been executed you can experiment and run the basicPubSub.py on your computer to see the AWS MQTT messages going live.

Interesting part is that there is no new user/policy created under AWS IAM. The certificate by itself represents the IoT identity so there is less administration needed if you need to manage millions of things.

Let's run the setup:

C:\dev\Blog\connect_device_package>**git clone https://github.com/aws/aws-iot-device-sdk-python.git**
Cloning into 'aws-iot-device-sdk-python'…
remote: Enumerating objects: 258, done.
remote: Total 258 (delta 0), reused 0 (delta 0), pack-reused 258R
Receiving objects: 100% (258/258), 186.66 KiB | 5.18 MiB/s, done.
Resolving deltas: 100% (103/103), done.

C:\dev\Blog\connect_device_package\aws-iot-device-sdk-python>**python setup.py install**
running install
running build
running build_py
creating build
creating build\lib
creating build\lib\AWSIoTPythonSDK
copying AWSIoTPythonSDK\MQTTLib.py -> build\lib\AWSIoTPythonSDK
…
…
yte-compiling C:\dev\Anaconda\Anaconda3\envs\aws-cli\Lib\site-packages

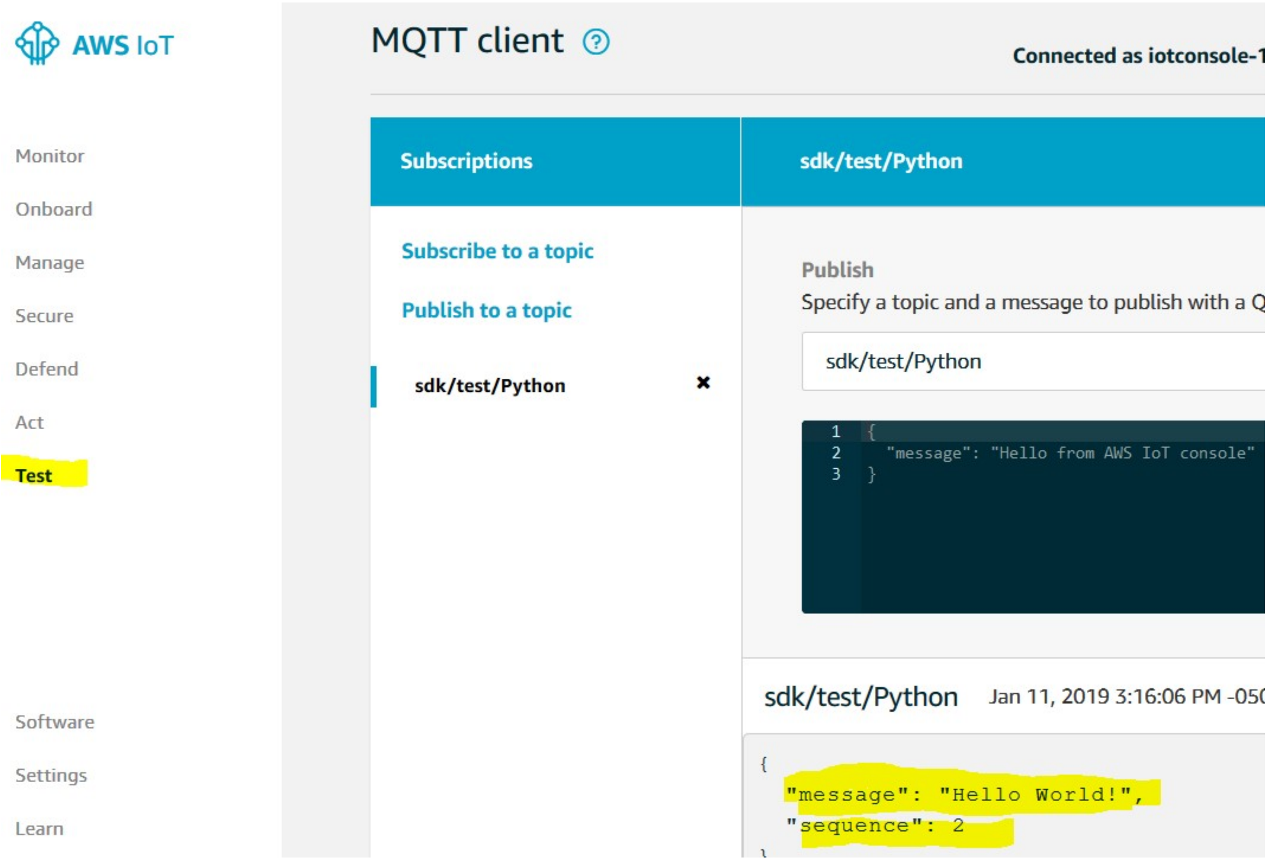
running install_egg_info
Writing C:\dev\Anaconda\Anaconda3\envs\aws-cli\Lib\site-packages\AWSIoTPythonSDK-1.4.2-py3.7.egg-info

C:\dev\Blog\connect_device_package>**curl https://www.amazontrust.com/repository/AmazonRootCA1.pem > root-CA.crt**
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 1188 100 1188 0 0 1434 0 –:–:– –:–:– –:–:– 1434

C:\dev\Blog\connect_device_package>**python aws-iot-device-sdk-python/samples/basicPubSub/basicPubSub.py -e xxxxxxxxxxxxx-xxx.iot.xx-xxx-x.amazonaws.com -r root-CA.crt -c 8266-01.cert.pem -k 8266-01.private.key**
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG – Initializing MQTT layer…
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG – Registering internal event callbacks to MQTT layer…
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – MqttCore initialized
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Client id: basicPubSub
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Protocol version: MQTTv3.1.1
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Authentication type: TLSv1.2 certificate based Mutual Auth.
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring endpoint…
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring certificates…
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring reconnect back off timing…
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Base quiet time: 1.000000 sec
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Max quiet time: 32.000000 sec
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Stable connection time: 20.000000 sec
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring offline requests queueing: max queue size: -1
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring offline requests queue draining interval: 0.500000 sec
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring connect/disconnect time out: 10.000000 sec
2019-01-11 14:21:53,276 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Configuring MQTT operation time out: 5.000000 sec
2019-01-11 14:21:53,291 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Performing sync connect…
2019-01-11 14:21:53,291 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Performing async connect…
2019-01-11 14:21:53,291 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO – Keep-alive: 600.000000 sec

Event consuming thread started
2019-01-11 14:21:53,291 – AWSIoTPythonSDK.core.protocol.mqtt_core – DEBUG –
Passing in general notification callbacks to internal client…
2019-01-11 14:21:53,291 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Filling in fixed event callbacks: CONNACK, DISCONNECT, MESSAGE
2019-01-11 14:21:53,525 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Starting network I/O thread…
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Produced [connack] event
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Dispatching [connack] event
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
No need for recovery
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Invoking custom event callback…
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO –
Performing sync subscribe…
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Adding a new subscription record: sdk/test/Python qos: 1
2019-01-11 14:21:53,604 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Filling in custom suback event callback…
2019-01-11 14:21:53,650 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Produced [suback] event
2019-01-11 14:21:53,650 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Dispatching [suback] event
2019-01-11 14:21:53,666 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Invoking custom event callback…
2019-01-11 14:21:53,666 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
This custom event callback is for pub/sub/unsub, removing it after invocation…
2019-01-11 14:21:55,680 – AWSIoTPythonSDK.core.protocol.mqtt_core – INFO –
Performing sync publish…
2019-01-11 14:21:55,680 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Filling in custom puback (QoS>0) event callback…
2019-01-11 14:21:55,712 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Produced [puback] event
2019-01-11 14:21:55,712 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Dispatching [puback] event
2019-01-11 14:21:55,727 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
Invoking custom event callback…
2019-01-11 14:21:55,727 – AWSIoTPythonSDK.core.protocol.internal.clients – DEBUG –
This custom event callback is for pub/sub/unsub, removing it after invocation…
2019-01-11 14:21:55,743 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Produced [message] event
2019-01-11 14:21:55,743 – AWSIoTPythonSDK.core.protocol.internal.workers – DEBUG –
Dispatching [message] event
Received a new message:
b'{"message": "Hello World!", "sequence": 0}'
from topic: sdk/test/Python

Local client working! Go to AWS console->IoT->Test (for example https://us-east-2.console.aws.amazon.com/iot/home?region=us-east-2#/test) and subscribe to MQTT



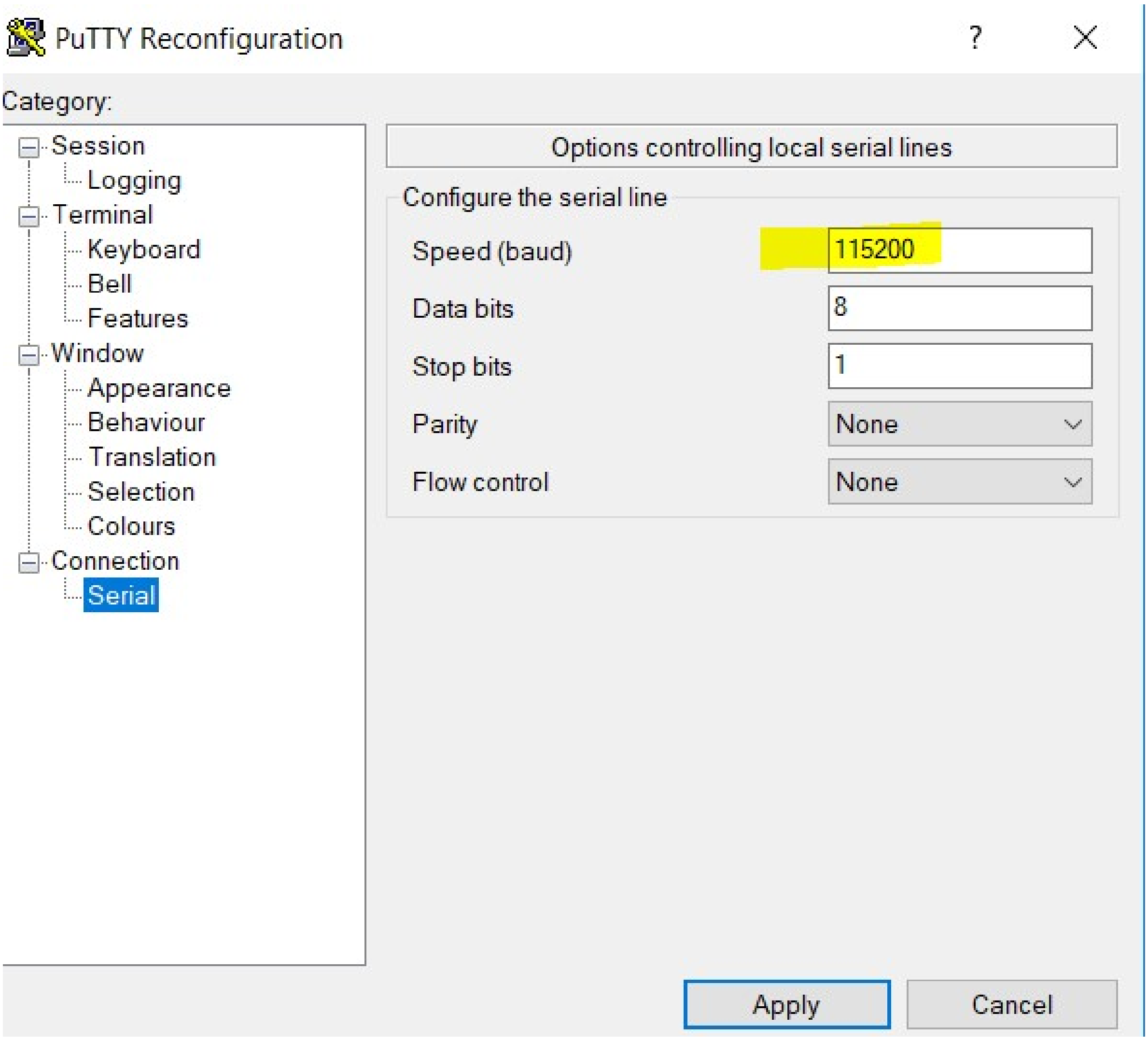
You will see the messages coming from the client you are running locally on your computer.

In the ideal world that's the end of story since the "Python can run everywhere". We could drop basicPubSub.py on ESP8266 and run, but no luck yet.

If you try to upload the AWS client as-is you will get a bunch of errors. ESP8266 just is too small to be able swallow all the dependencies. On another side let's think positive. We do have AWS MQTT Python client, all the client certs, and a registered IoT "thing" inside the AWS cloud. All of them working just fine together.

Since we need the real 8266 hardware module at this point, please make sure you have Micropython 1.9.4 installed on 8266. There is a lot of useful info on "how to install Micropython" on the internet. For esp8266-01 specifically there is my post at https://awsiot.wordpress.com/2019/01/09/micropython-on-esp-8266. Assuming you have Micropython installed, the procedure below will be valid for the most esp8266 and esp32 modules including popular NodeMCU esp8266-12E

Let's connect 8266 to the USB port and play with the serial terminal to get comfortable. There are many serial terminals available on the internet, I will use PuTTY. Serial port number and port speed are two important settings you need to keep an eye on. Port number may change depending on other USB devices connected to your computer. Although my system installed USB drivers automatically you may need to install a driver for your particular chipset. The basic hardware test is to list COM ports on your computer before plugging in the board and after. If you see new COM port, this is your adapter. Make a note which COM port number it is. On my system the COM port number is COM6 so make sure update the command line using your port.



Make sure COM port speed set to 115200 and COM port matches to your adapter port#.

Once you get serial terminal connected you will see MicroPython REPL prompt: >>>. You can run Python commands right from there.

```
MicroPython v1.9.4-8-ga9a3caad0 on 2018-05-11; ESP module with
ESP8266
 Type "help()" for more information.


>>>gc.mem_free()
      12352
```

this is useful to see the amount of free RAM available

>>> **import machine; machine.reset()**

This command will restart 8266. As far I remember after every restart Micropython will execute boot.py and then main.py located on built-in flash file system.

Also we'll need **ampy** which is Adafruit MicroPython Tool to copy files between the our computer and esp8266. https://github.com/adafruit/ampy

ampy “ls” command:

C:\dev\Blog>**ampy -p COM6 ls**
/boot.py

There are also “put”, “get”, and “mkdir” commands which may be very useful. On some boards there is additional flash installed inside the /flash directory, but on esp8266-01 MicroPython firmware didn’t create any directories. Let’s create “flash” directory for consistency:

C:\dev\Blog\connect_device_package>**ampy -p COM6 mkdir flash**

C:\dev\Blog>**ampy -p COM6 ls**
/boot.py
/flash

In order to proceed with AWS MQTT test on esp8266 we need the following steps:

- Connect to WIFI
- Initialize MQTT Client using AWS certs
- Connect MQTT Client
- Publish some message we can see on AWS IoT test console.

Let’s create a new Python file called main.py.

We will connect 8266 to WIFI using example from esp 8266 SDK

https://docs.micropython.org/en/latest/wipy/tutorial/wlan.html

```
def connect_wifi(ssid, pw):
    from network import WLAN
    from network import STA_IF
    import machine

    wlan = WLAN(STA_IF)
    nets = wlan.scan()
    if(wlan.isconnected()):
        wlan.disconnect()
    wlan.connect(ssid, pw)
    while not wlan.isconnected():
        machine.idle() # save power while waiting
        print('WLAN connection succeeded!')
        break
    print("connected:", wlan.ifconfig())
```

Now let’s create MQTT client for AWS. There is a post on micropython.org https://forum.micropython.org/viewtopic.php?t=5166 regarding MQTT connection using esp32 module. Basically there are two lines of code:

```
client = MQTTClient(client_id="aws-mqtt",  server="XXXXXXXXXXX.XX-
central-1.amazonaws.com", port=8883,  keepalive=4000, ssl=True,
ssl_params={ "key":key, "cert":cert,  "server_side":False })

client.connect()
```

Our MQTT publish method will call client.publish() API which takes MQTT topic and message – no magic there.

```
def pub_msg(msg):
    global mqtt_client
    try:
        mqtt_client.publish(MQTT_TOPIC, msg)
        print("Sent: " + msg)
    except Exception as e:
        print("Exception publish: " + str(e))
        raise
```

Our AWS Client need to read client cert and private key from the file system. There are many examples which instead use file path but this code didn't work for me with Micropython 1.9.

```
with open(KEY_FILE, "r") as f:
        key = f.read()
print("Got Key")
with open(CERT_FILE, "r") as f:
        cert = f.read()
print("Got Cert")
```

There is resulting main.py file on GitHub: https://gist.github.com/StanS-AWS/a243ffac4fd19a3a8ab243633aa322db. You need to set:

- AWS endpoint URL
- WiFi SSID
- Wifi password

> *You can find your AWS endpoint URL inside the connect_device_package.zip with basicPubSub example:*
>
> *python aws-iot-device-sdk-python/samples/basicPubSub/basicPubSub.py -e* ***xxxxxxxxxxx-xxx.iot.xx-xxxx-x.amazonaws.com*** *-r root-CA.crt -c 8266-01.cert.pem -k 8266-01.private.key*

Let's update main.py and upload the files to esp8266:

```
ampy -p COM6 put main.py
ampy -p COM6 put 8266-01.cert.pem /flash/cert
```

There are three files in total – updated main.py, client certificate, and private key you have received from AWS inside the zip file.

We need to reset esp 8266 to start main.py. Connect the serial terminal and reset the module:
>>> **import machine; machine.reset()**

This code ***works*** on ESP32 but a no-go on 8266. There is an MQTT connect error message ” invalid key “.

```
Connecting WIFI
 connected: ('192.168.100.6', '255.255.255.0', '192.168.100.1',
'192.168.100.1')
 Connecting MQTT
 Got Key
 Got Cert
 Cannot connect MQTT: invalid key
 invalid key
 MicroPython v1.9.4-8-ga9a3caad0 on 2018-05-11; ESP module with
ESP8266
 Type "help()" for more information.
```

After some digging it turned-out Micropython on 8266 using different TLS library under the hood. In order to perform client cert authentication we need an extra step. If you don’t have OpenSSL you’ll need to install OpenSSL from here: https://wiki.openssl.org/index.php/Binaries

Run following two commands with OpenSSL

```
openssl x509 -in 8266-01.cert.pem -out 8266-01.cert.der -outform DER
openssl rsa -in 8266-01.private.key -out 8266-01.key.der -outform
DER
```

This will convert the certificates into binary .der format. Now upload the certs into 8266 /flash directory:

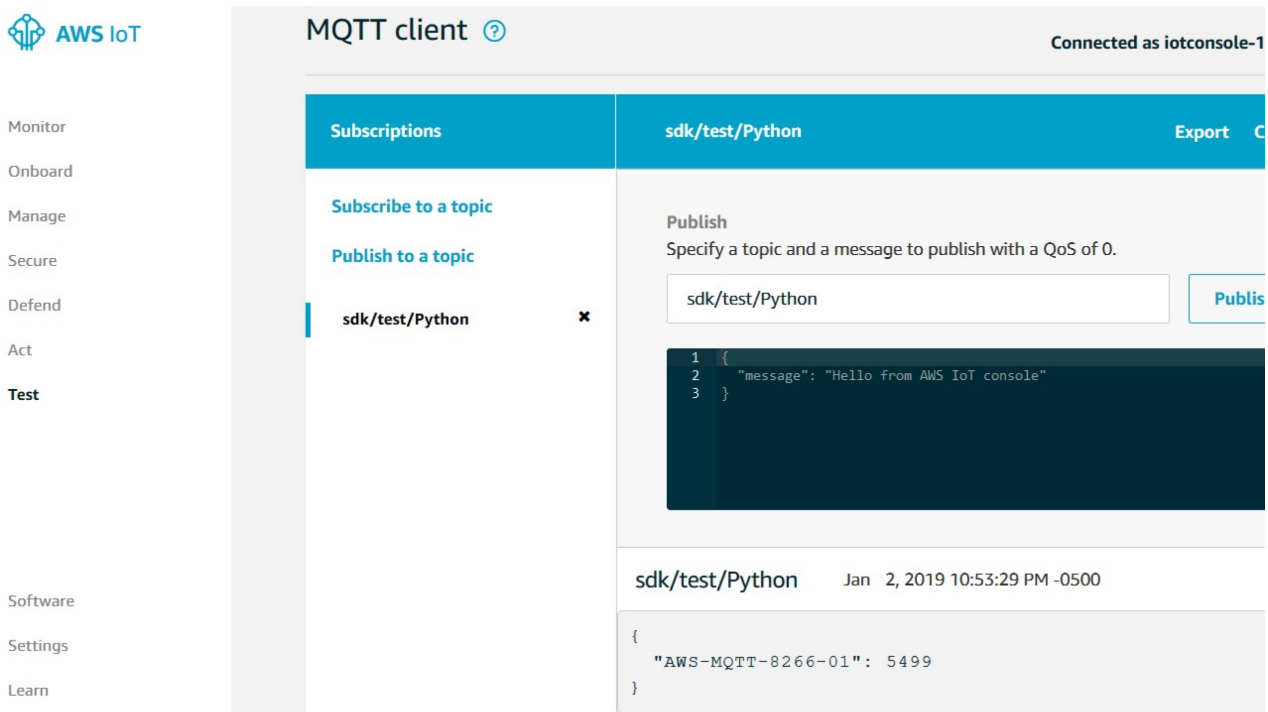
```
ampy -p COM6 put 8266-01.cert.der  /flash/cert
ampy -p COM6 put 8266-01.key.der  /flash/key
```

Go to AWS IoT test console and subscribe to **sdk/test/Python** topic. After that open serial terminal and reset the module:

import machine; machine.reset() . If everything going well you will see something like that:

```
Connecting WIFI
 connected: ('192.168.100.6', '255.255.255.0', '192.168.100.1',
'192.168.100.1')
 Connecting MQTT
 Got Key
 Got Cert
 MQTT Connected
 Publishing
 Sent: {"AWS-MQTT-8266-01":5499}
 OK
 MicroPython v1.9.4-8-ga9a3caad0 on 2018-05-11; ESP module with
ESP8266
 Type "help()" for more information.
 >>>
```

On AWS test console you will see the test message we just sent:



Congratulations, we just published message on AWS MQTT using Micropython on esp8266. There are few notes:

- Connection takes long time to succeed, for example my 8266-01 module takes about 40 seconds to publish the message after restart.
- There is no AWS endpoint CA validation which is limitation of the current TLS library. On another side AWS will authenticate all the client certs as expected.
- We have reused AWS pre-configured policy which defines the list of allowed MQTT topics and ClientId. In our test the topic set to **sdk/test/Python,** and **ClientId** set to **basicPubSub**. In the real application you will need to change the topic so make sure to update the policy on ASW side accordingly.
- I only tested with esp8266-01 and esp32 but the same steps most likely will work with NodeMCU clones.