NodeMCU ESP8266 with AWS IoT Core using Arduino IDE & MQTT

In this tutorial, I will tell you How to connect **NodeMCU ESP8266** with **AWS IoT Core** using **Arduino IDE** and **MQTT Protocol**. In this process, we will see how to **create a thing in AWS IoT core**, generating **certificates** and **policy**, How all **AWS IoT core** credentials are converted to **.der format** and directly downloaded into the **NodeMCU ESP8266 SPIFFS file system**. We will be developing a sketch that will marry ESP8266 and AWS IoT Core using MQTT Protocol.

This sketch requires the certificate files to be uploaded to the **device's flash rather than storing them in line with the script**. This allows the same **script to be used by multiple devices**, that each read the required files from their internal flash storage.

Introduction to AWS IoT Core:

The Internet of Things (IoT) is being integrated with almost every device nowadays. There are a number of hardware and software IoT platforms available in the market for building IoT based applications. In my previous article, I have explained how to interface DHT22 with NodeMCU and post the Temperature and Humidity to the Thingspeak web server. Likewise, we can interface sensors to the hardware development kits like ESP32, ESP8266, Raspberry Pi, Particleboards(Aargon, Boron, Xenon) and post data to the clouds like Thingspeak, Ubidots, AWS IoT Core, Microsoft Azure.

Amazon is not only in e-commerce but also focusing on IoT and providing cloud-based service named AWS IoT. Here, **AWS IOT stands for Amazon Web Service Internet of Things**. This service allows us to connect our devices to the internet for processing, operating and exchanging data securely. Along with AWS IoT, the Amazon Web Services also provides tons of other features like virtual machine deployment, web-hosting, etc.

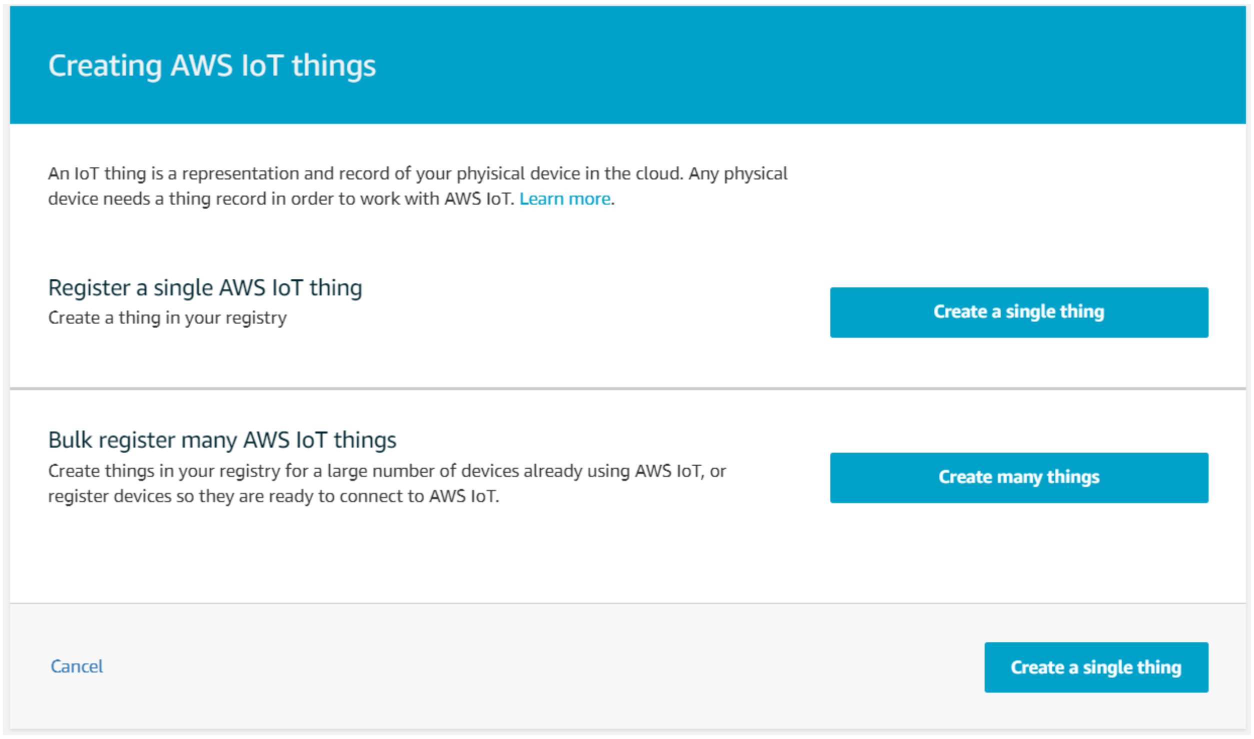
Steps involved in this tutorial:

1. **Creating a Thing in the AWS, generating a certificate and attaching a policy to it.**
2. **Converting AWS credential(Certificate, Private Key, Root CA) from .pem to .der format**
3. **Installing ESP8266 sketch data upload tool in Arduino IDE**
4. **Arduino sketch and modifications according to the thing.**
5. **Uploading AWS certificates & code to the NodeMCU ESP8266**
6. **Testing/Subscription of things on Amazon Web Services(AWS).**
7. **Results & Data Logging.**

1. Creating a Thing in the AWS IoT Core, generating a certificate and attaching a policy to it.

Setting up the AWS environment for these devices is pretty simple. check the following:

login to the AWS Management Console & search for IoT core in the Amazon Services, Find services search bar will help you in this regard. After getting into the IoT Core section, tap on the tab called "Manage" from the AWS IoT menu which is on the left side, tap on the register thing button if you haven't added any devices till now. If you have previously added things just tap on the button named "Create" which is on the top right corner beside the iot-notifications Icon.


Creating AWS IoT things

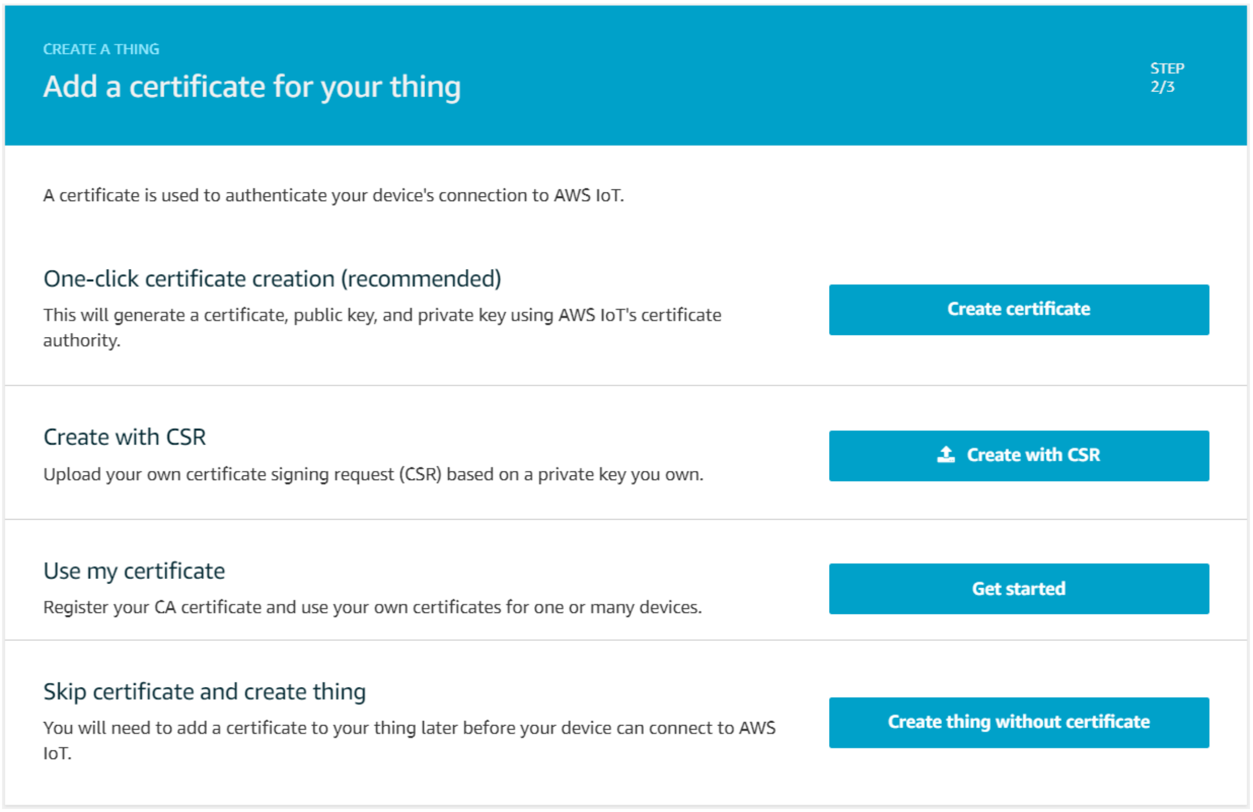
An IoT thing is a representation and record of your phyisical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. Learn more.

Register a single AWS IoT thing
Create a thing in your registry

Bulk register many AWS IoT things
Create things in your registry for a large number of devices already using AWS IoT, or register devices so they are ready to connect to AWS IoT.

Setting up the AWS environment for these devices is pretty simple. check the following: Amazon AWS

and login to the AWS Management Console & search for IoT core in the Amazon Services, Find services search bar will help you in this regard. After getting into the IoT Core section, tap on the tab called "Manage" from the AWS IoT menu which is on the left side, tap on the register thing button if you haven't added any devices till now. If you have previously added things just tap on the button named "Create" which is on the top right corner beside the iot-notifications Icon.



CREATE A THING

Add a certificate for your thing

STEP 2/3

A certificate is used to authenticate your device's connection to AWS IoT.

One-click certificate creation (recommended)

This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

Create with CSR

Upload your own certificate signing request (CSR) based on a private key you own.

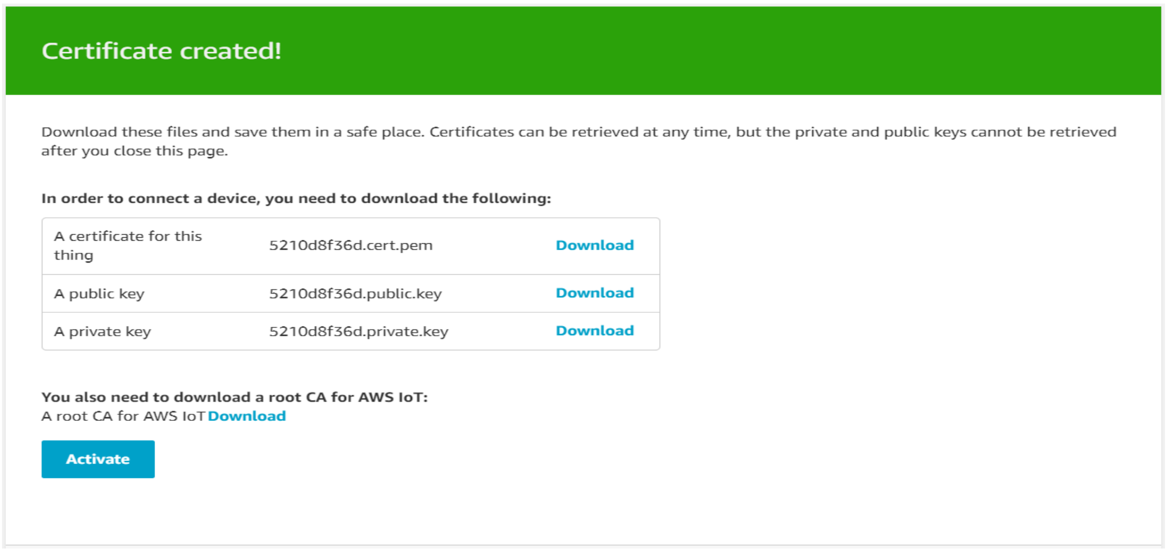
Use my certificate

Register your CA certificate and use your own certificates for one or many devices.

Skip certificate and create thing

You will need to add a certificate to your thing later before your device can connect to AWS IoT.

Use create a certificate.



Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	5210d8f36d.cert.pem	Download
A public key	5210d8f36d.public.key	Download
A private key	5210d8f36d.private.key	Download

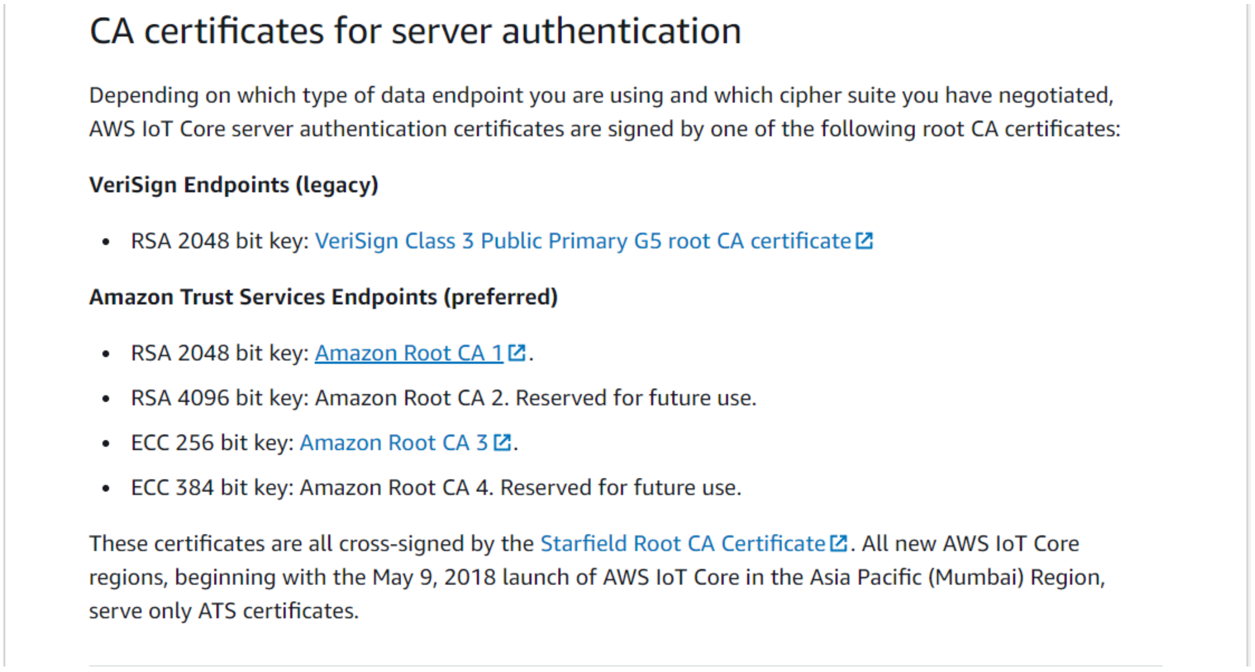
You also need to download a root CA for AWS IoT:

A root CA for AWS IoT Download

Next point is to create and attach a policy to the certificate, authorizing the authenticated device to perform IoT actions on IoT resources. for this tap on the “secure” tab from the AWS IoT menu which is on the left side, later go for the policies section. Now tap on the button named “Create” which is on the top right corner beside the iot-notifications Icon. give your policy name and fill the fields(Action, Resource ARN) with a star “*” and check to Allow for Effect option then press the “create” button.

Now tap on the certificates section which is right above the policies section, You will see a certificate which you have created earlier, tap on the three dots and choose to attach the policy, a pop will come showing your existing policies, check on the recent policy that you have created and attach. That’s it you have successfully created a thing, generated a certificate and attached policy to it.

You need to Download A certificate for a thing(option 1), private Key(option 3) and click on a root CA for AWS IOT Download and a page will be redirected and need to download RSA 2048 bit Key.(Save as a Link). Keep all the 3files a side in a proper file.



CA certificates for server authentication

Depending on which type of data endpoint you are using and which cipher suite you have negotiated, AWS IoT Core server authentication certificates are signed by one of the following root CA certificates:

VeriSign Endpoints (legacy)

- RSA 2048 bit key: VeriSign Class 3 Public Primary G5 root CA certificate

Amazon Trust Services Endpoints (preferred)

- RSA 2048 bit key: Amazon Root CA 1.
- RSA 4096 bit key: Amazon Root CA 2. Reserved for future use.
- ECC 256 bit key: Amazon Root CA 3.
- ECC 384 bit key: Amazon Root CA 4. Reserved for future use.

These certificates are all cross-signed by the Starfield Root CA Certificate. All new AWS IoT Core regions, beginning with the May 9, 2018 launch of AWS IoT Core in the Asia Pacific (Mumbai) Region, serve only ATS certificates.

2.Converting AWS IoT Core credential(Certificate, Private Key, Root CA) from .pem to .der format

There are two main methods for encoding certificate data.

- **DER = Binary encoding for certificate data**
- **PEM = The base64 encoding of the DER-encoded certificate, with a header and footer lines added.**

DER

DER: (Distinguished Encoding Rules) is a subset of BER encoding providing for exactly one way to encode an ASN.1 value. DER is intended for situations when a unique encoding is needed, such as in cryptography, and ensures that a data structure that needs to be digitally signed produces a unique serialized representation.

PEM

PEM: (Privacy-enhanced Electronic Mail) Simply a US-ASCII by base64 encoded DER certificate, certificate request, or PKCS#7, enclosed between typical PEM delimiters. ie "—–BEGIN CERTIFICATE—–" and "—–END CERTIFICATE—–". PEM is an abbreviation for Privacy Enhanced Mail (RFC 1421 – RFC 1424), an early standard for securing electronic mail (IRTF, IETF). PEM never has been widely adopted as Internet Mail Standard but has become a staple standard in x509 pki (also called pkix)
=> Since our ESP8266 will not understand base64 encoding we will convert that certificate to Binary.

OpenSSL on Windows

If you're using Windows, you can install one of the many OpenSSL open-source implementations: the one we can recommend is Win32 OpenSSL by Shining Light Production, available as *light* or *full* version, both compiled in x86 (32-bit) and x64 (64-bit) modes. You can install any of these versions, as long as your system supports them.
IMPORTANT: OpenSSL for Windows requires the Visual C++ 2008 Redistributables runtime in order to work.
OpenSSL is basically a console application, meaning that we'll use it from the command-line: after the installation process completes, it's important to check that the installation folder (C:\Program Files\OpenSSL-Win64\bin for the 64-bit version) has been added to the system PATH (Control Panel > System> Advanced > Environment Variables): if it's not the case, we strongly recommend to manually add it, so that you can avoid typing the complete path of the executable every time you'll need to launch the tool.
Once OpenSSL will be installed, we'll be able to use it to convert our SSL Certificates in various formats.

- **Now go to the folder where all the certificates are downloaded, The AWS certificate will be something like this "xxxxxxxxxx-certificate.pem.crt.txt" So now just rename that document to "xxxxxxxxxx-certificate.pem.crt".**
  The following commands will convert the downloaded device certificate files to the correct format for this script.

  ```
  > openssl x509 -in xxxxxxxxxx-certificate.pem.crt -out cert.der -
  outform DER
  > openssl rsa -in xxxxxxxxxx-private.pem.key -out private.der -outform
  DER
  > openssl x509 -in AmazonRootCA1.pem -out ca.der -outform DER
  ```

  Replace "xxxxxxxxxx" with your certificate name and AmazonRootCA1 will remain the same because there is no change
  Example:

  ```
  > openssl x509 -in 2b495edf21-certificate.pem.crt -out cert.der -
  outform DER > openssl rsa -in 2b495edf21-private.pem.key -out
  private.der -outform DER
  > openssl x509 -in AmazonRootCA1.pem -out ca.der -outform DER

  Note: if you don't have openssl You can download from the official
  website as in .exe Format.
  ```

  After executing these commands, you will see certificates save in the same folder with .der format, copy these DER-format files into a folder called `data`

3.Installing ESP8266 sketch data upload tool in Arduino IDE

Before going to ESP8266 sketch data upload tool make sure, your machine has the latest Arduino IDE installed locally, with the ESP8266 plugin installed. If you don't know how to install ESP8266 plugin,

check out this article: NodeMCU programming with Arduino.

Now Let's install Arduino ESP8266 filesystem uploader which packs the sketch data folder into the SPIFFS filesystem image, and uploads the image to ESP8266 flash memory.

- **Download the tool archive"ESP8266FS-0.4.0.zip" from the Git hub releases page.**
- **In your Arduino sketchbook directory, create `tools` directory if it doesn't exist yet. You can find the location of your sketchbook directory in the Arduino IDE at File > Preferences > Sketchbook location.**
- **Unpack the tool into `tools` directory (the path will look like `<sketchbook directory>/tools/ESP8266FS/tool/esp8266fs.jar`).**
- **Restart Arduino IDE.**
- **Select "tools > ESP8266 Sketch Data Upload" will be there.**
  Credits of the tool : Hristo Gochkov.

4.Arduino sketch and modifications according to the thing.

The following connects your NodeMCU ESP8266 to AWS IoT server then:

– publishes "hello world" to the topic "outTopic" every two seconds to the server.

– subscribes to the topic "inTopic", printing out any messages that are coming from the server.

Open a new sketch file of Arduino IDE, Copy & Paste the below code into that and save with some file name. i.e "ESP8266_AWS_IoTCore"

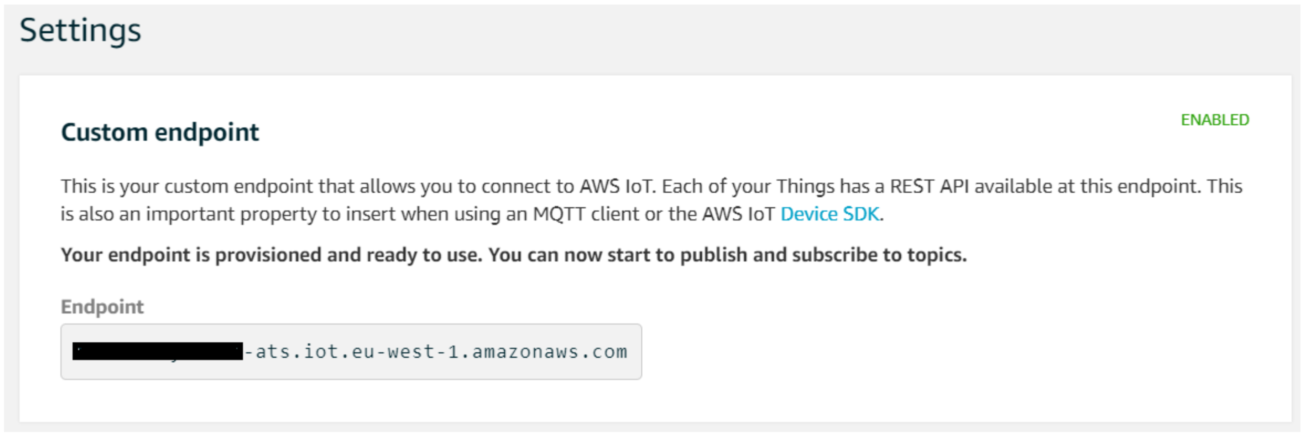
You have to make sure that the WiFi username and password are provided which is available in the range.

```
const char* ssid = "Wifi_Name";
const char* password = "Wifi_password";
```

Also, change the AWS_endpoint which is the address of the MQTT broker for your AWS account in a specific region.

```
const char* AWS_endpoint = "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com";
//MQTT broker ip
```

`You can find the custom endpoint in settings tab of AWS IOT CORE`



Settings

Custom endpoint

ENABLED

This is your custom endpoint that allows you to connect to AWS IoT. Each of your Things has a REST API available at this endpoint. This is also an important property to insert when using an MQTT client or the AWS IoT Device SDK.

Your endpoint is provisioned and ready to use. You can now start to publish and subscribe to topics.

Endpoint

```
[redacted]-ats.iot.eu-west-1.amazonaws.com
```

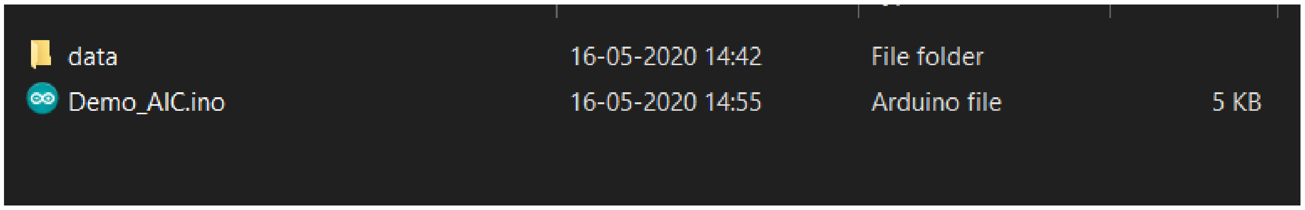
Need to change the endpoint in Code of arduino as shown below.

```
const char* ssid = "hologram";
const char* password = "09876543";

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org");

const char* AWS_endpoint = "xxxxxxxxxxxxxx-ats.iot.us-west-2.amazonaws.com"; //MQTT broker ip

void callback(char* topic, byte* payload, unsigned int length) {
Serial.print("Message arrived [");
Serial.print(topic);
Serial.print("] ");
for (int i = 0; i < length; i++) {
Serial.print((char)payload[i]);
}
```

In the place of XXXXXXXXXXXXXX you need to add your endpoint.

5.Uploading AWS certificates & code to the NodeMCU ESP8266

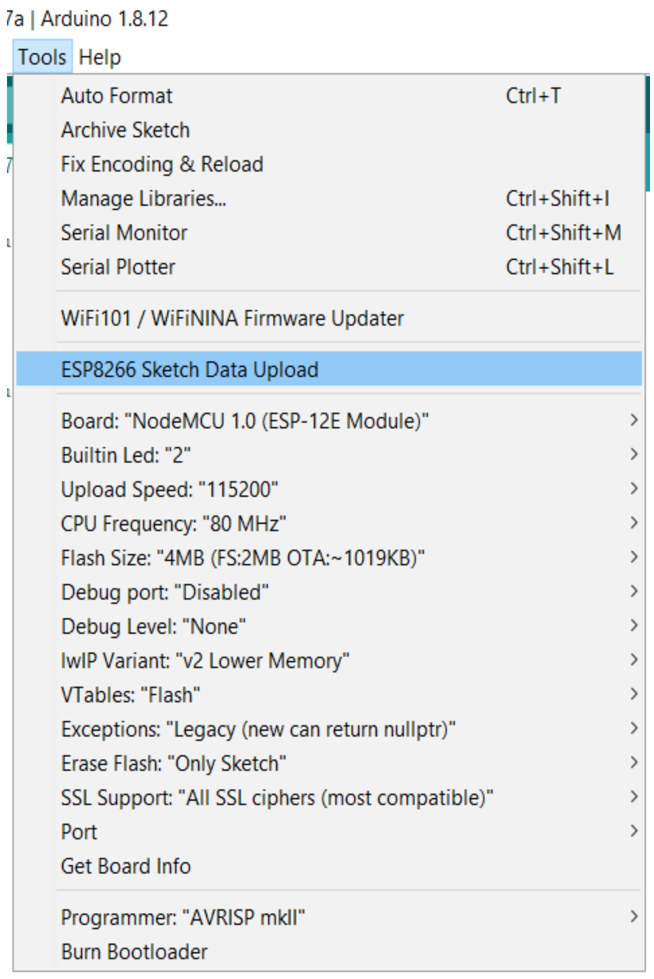
Make sure that folder `data` should sit alongside with your Arduino code as shown below, for that you can move `data` folder into the Arduino code Folder.



data	16-05-2020 14:42	File folder	
Demo_AIC.ino	16-05-2020 14:55	Arduino file	5 KB

Now open your Arduino code for ESP8266 and Select *Tools > ESP8266 Sketch Data Upload* menu item. This should start uploading the files into the ESP8266 flash file system. When done, the IDE status bar will display the SPIFFS Image Uploaded message. It might take a few minutes for large file system sizes.

Note: Make sure you have selected a board, port, and closed Serial Monitor.

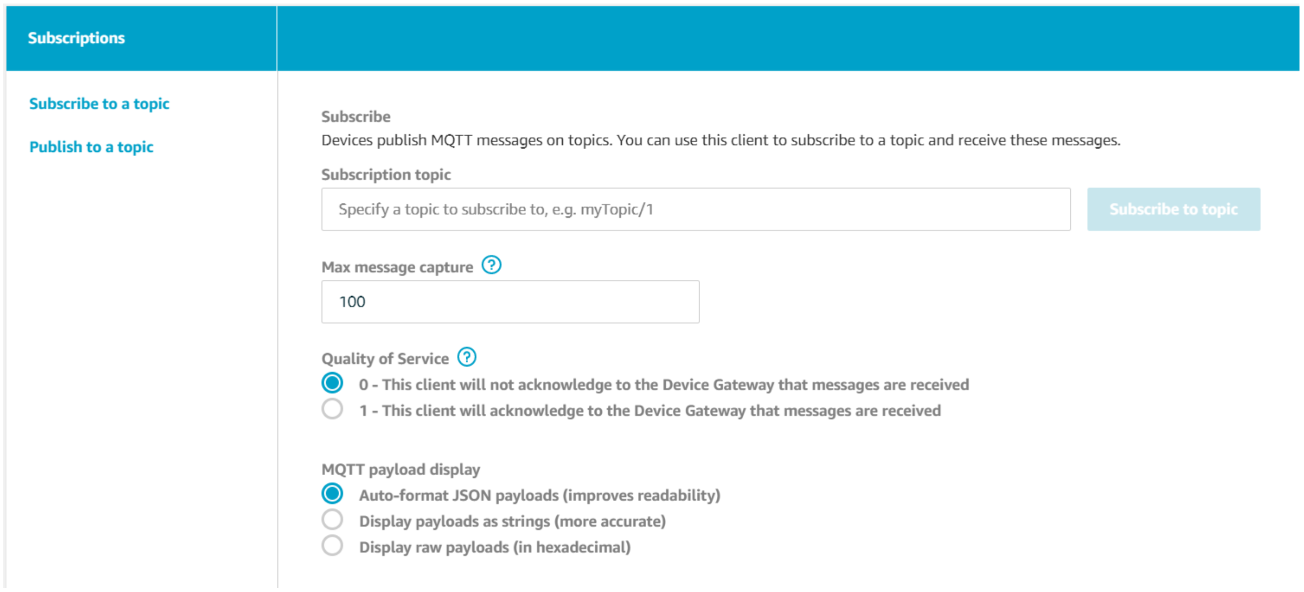

7a | Arduino 1.8.12
Tools
Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
ESP8266 Sketch Data Upload
Port
Get Board Info
Burn Bootloader

Board: "NodeMCU 1.0 (ESP-12E Module)"
Builtin Led: "2"
Upload Speed: "115200"
CPU Frequency: "80 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Debug port: "Disabled"
Debug Level: "None"
IwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Legacy (new can return nullptr)"
Erase Flash: "Only Sketch"
SSL Support: "All SSL ciphers (most compatible)"
Programmer: "AVRISP mkII"

The above mentioned configuration must be followed.
=> To save time while uploading to these devices (both code and especially SPIFFS), set the upload baud rate as high as does not produce errors. That was 921600 for my devices, up from the default of 115200, saving a lot of time while iterating.
=> If you get errors when trying to upload the certificates to the device flash, you may not have SPIFFS enabled from the Arduino > Tools menu. Any of the SPIFFS storage size values will work, the certificates take up significantly less than 1 MB. All my devices had 4 MB of flash. If you ever change the board type then this setting is lost; I wiped my certificates several times while re-uploading code while I hadn't noticed this, resulting in connection errors.
Once the Certificates are uploaded successfully, Now go for uploading of the Arduino sketch "ESP8266_AWS_IoTCore" the NodeMCU board by clicking on the upload button of Arduino IDE.
=> Once the script is uploaded you can view the output on the Serial Monitor. For debugging any potential issues with the certificates and policies, I used the command on this page. This checks the chain of trust between the client certificate and root CA, and checks that a TLS connection can be established. Potential issues could arise from the certificate not being activated in the AWS Console, or the device clock not having synced with the NTP server which prevents certificate validation.

6.Testing/Subscription of thing on Amazon Web Services (AWS IoT Core).

It's time to get back to the AWS IoT Core and subscribe or the topic.This Subscription topic bar can be found in the MQTT client section of the Test tab of your AWS IoT column. Just type outTopic and push subscribe to the topic button as shown below.



Subscribe
Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.

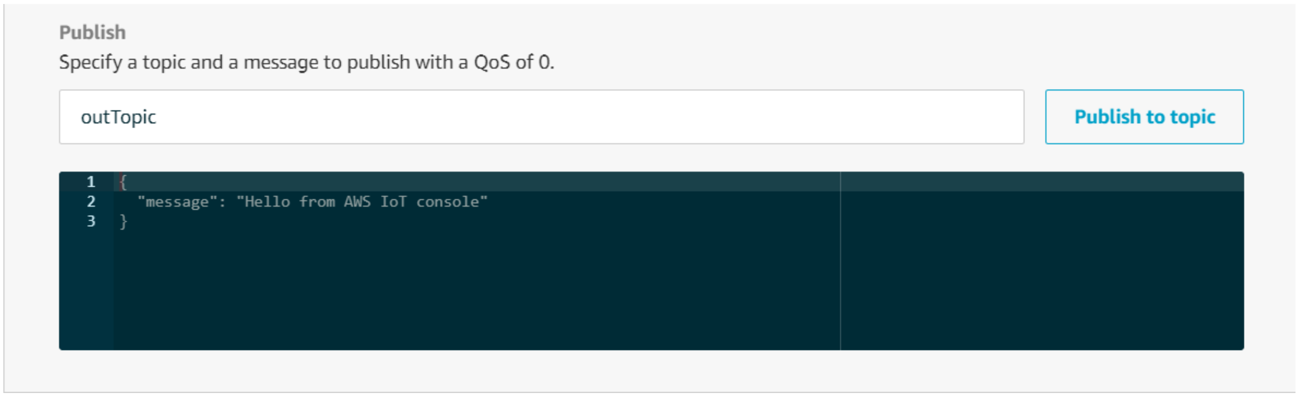
Subscription topic: Specify a topic to subscribe to, e.g. myTopic/1

Max message capture: 100

Quality of Service
[✓ 0 - This client will not acknowledge to the Device Gateway that messages are received]
1 - This client will acknowledge to the Device Gateway that messages are received

MQTT payload display
[✓ Auto-format JSON payloads (improves readability)]
Display payloads as strings (more accurate)
Display raw payloads (in hexadecimal)

The above shown is the test panel on AWS IOT Core.



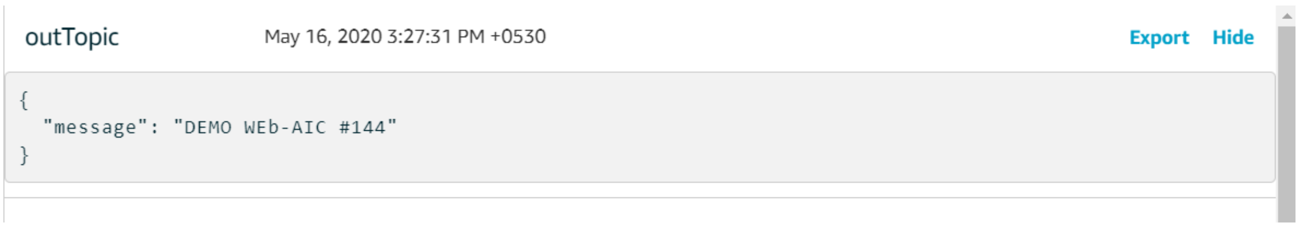
Publish
Specify a topic and a message to publish with a QoS of 0.

outTopic

```
{
  "message": "Hello from AWS IoT console"
}
```

7.Results & Data Logging.

Hopefully, your device is now connected to your WiFi network, has authenticated itself with AWS, has the permissions to connect and publish to an IoT topic, and you're seeing messages in MQTT client like this.



May 16, 2020 3:27:31 PM +0530

```
{
  "message": "DEMO WEb-AIC #144"
}
```

Need to Type "outTopic" in the given space and need to click on the Publish the topic button after all successful Publication on Data you can observe the data transfer in outTopic Bar as shown above. This Would be The Final Output.

For Arduino Code : https://github.com/manishkotni/AWS-IOT_CORE