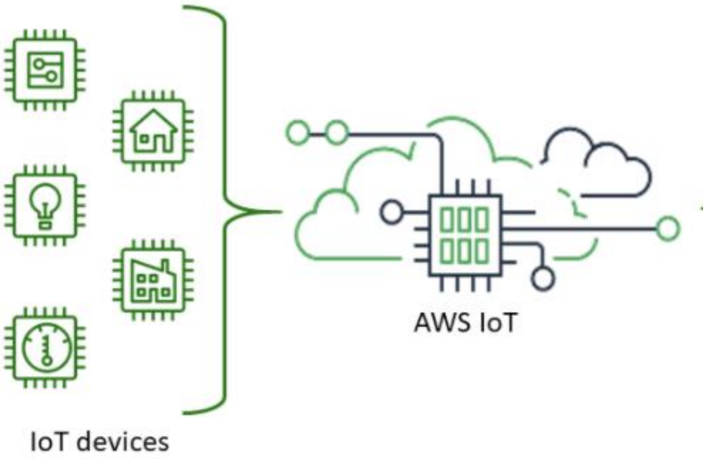
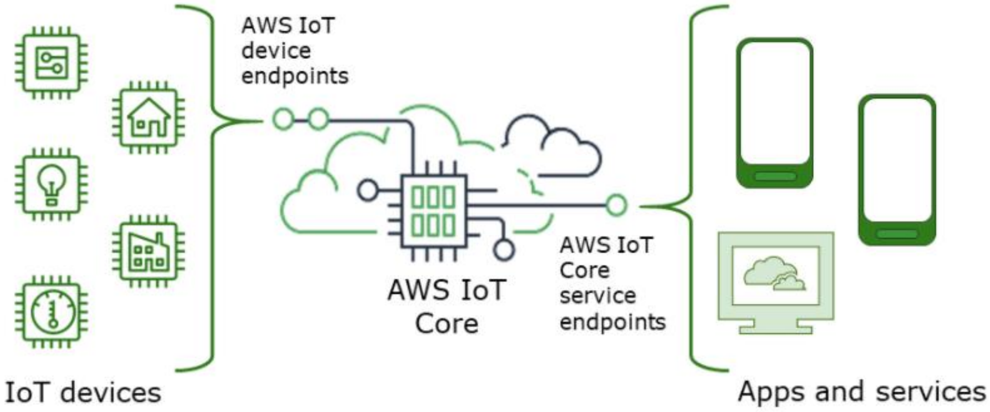
AWS IOT CORE WITH ESP8266

AWS IoT provides the cloud services that connect your IoT devices to other devices and AWS cloud services and lets you select the most appropriate and up-to-date technologies for your solution. AWS IoT can provide support for your compatible devices to facilitate the development and integration of your devices with AWS IoT. To help you manage and support your IoT devices in the field, AWS IoT communication supports **MQTT** and **HTTPS**.



Connecting to AWS IoT Core?

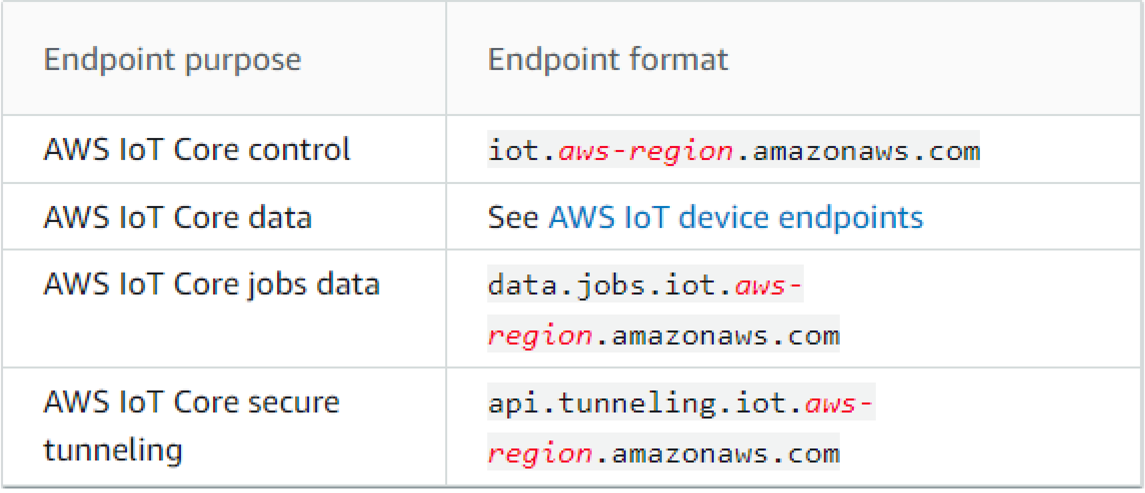
AWS IoT Core supports connections with IoT devices, services, and apps. Devices connect to the AWS IoT Core so they can send data to and receive data from AWS IoT services and other devices. Apps and other services also connect to AWS IoT Core to control and manage the IoT devices and process the data from your IoT solution.



The **AWS IoT** Core service ***endpoints*** provide access to functions that control and manage your AWS IoT solution. Endpoints support communication between your IoT devices and AWS IoT.

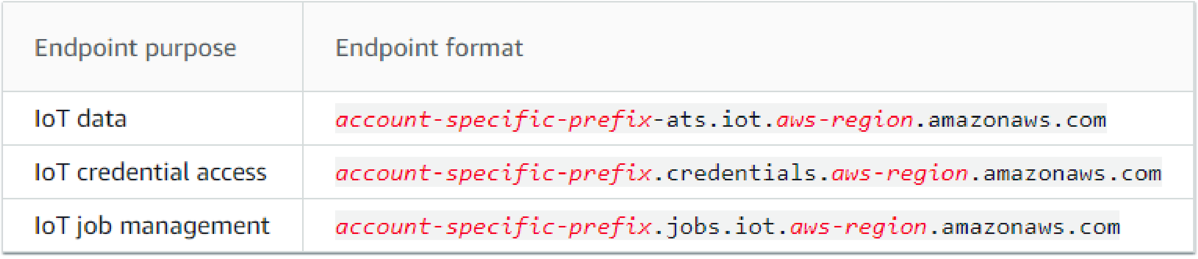
We are using second one,

AWS IoT Core data, from this we can have three more divisions:



Endpoint purpose	Endpoint format
AWS IoT Core control	`iot.aws-region.amazonaws.com`
AWS IoT Core data	See AWS IoT device endpoints
AWS IoT Core jobs data	`data.jobs.iot.aws-region.amazonaws.com`
AWS IoT Core secure tunneling	`api.tunneling.iot.aws-region.amazonaws.com`

From this table we can see that our purpose is ***IoT data***.



Endpoint purpose	Endpoint format
IoT data	`account-specific-prefix-ats.iot.aws-region.amazonaws.com`
IoT credential access	`account-specific-prefix.credentials.aws-region.amazonaws.com`
IoT job management	`account-specific-prefix.jobs.iot.aws-region.amazonaws.com`

So, my endpoint looks like :

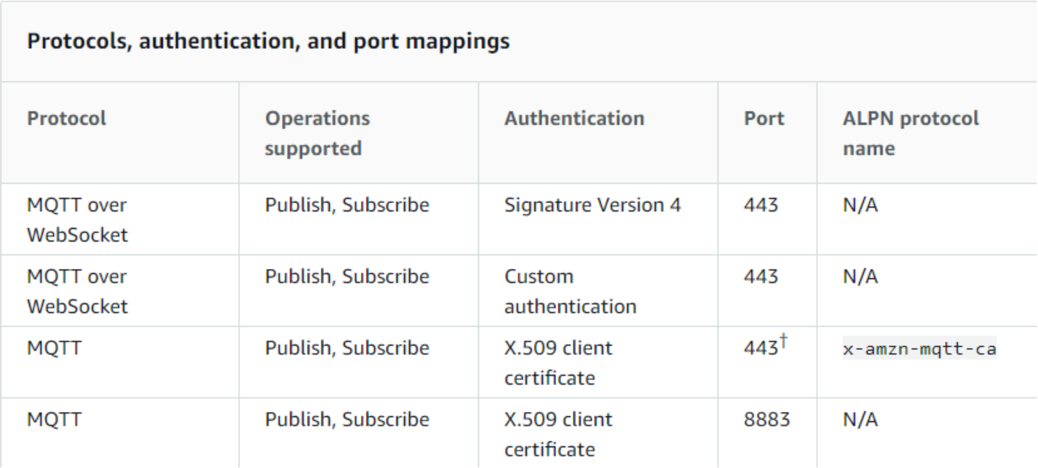
E.g. ***XXXXXXXXXXX-ats.iot.ap-south-1.amazonaws.com***

It's because of my account and its returns an ATS signed data endpoint which is used to send and receive data to and from the message broker, **Device Shadow**, and **Rules Engine** components of **AWS IoT**. AWS IoT manages device communication through a message broker.

Communication and MQTT

Devices and clients publish messages to the message broker and also subscribe to messages that the message broker publishes. Messages are identified by an application-defined **topic**. Device connections to AWS IoT use ***X.509 client certificates*** and ***AWS signature V4*** for authentication. Device communications are secured by TLS version 1.2 and AWS IoT requires devices to send the **Server Name Indication (SNI)** extension when they connect. The AWS IoT Device SDKs help us connect your IoT devices to AWS IoT Core and they support ***MQTT*** and ***MQTT over WSS protocols***.

We have the protocols and port mappings below:



Protocols, authentication, and port mappings

Protocol	Operations supported	Authentication	Port	ALPN protocol name
MQTT over WebSocket	Publish, Subscribe	Signature Version 4	443	N/A
MQTT over WebSocket	Publish, Subscribe	Custom authentication	443	N/A
MQTT	Publish, Subscribe	X.509 client certificate	443†	`x-amzn-mqtt-ca`
MQTT	Publish, Subscribe	X.509 client certificate	8883	N/A

OUR PURPOSE?

Here we need to ***Publish*** and ***Subscribe*** features for data operation. So, we have ***MQTT over WebSocket*** protocol to connect with a secured port of ***443*** and the authentication method of sigv4. So, let's move to ***AWS IoT core*** for console creation.

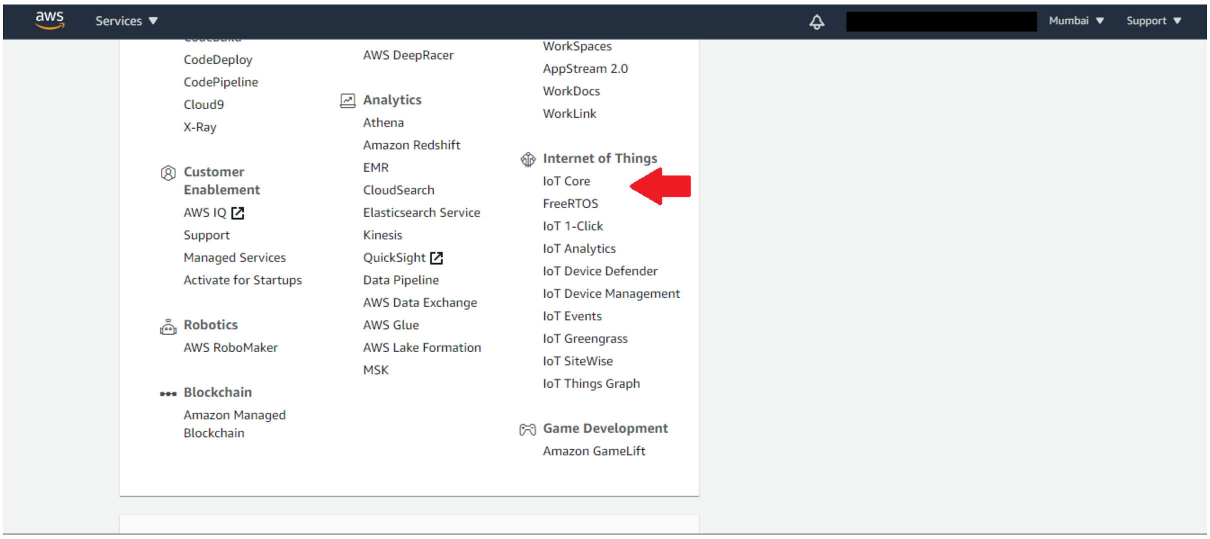
AWS IOT CORE CREATION:

Step 1:

- Go to the Amazon Web Services home page : https://aws.amazon.com/
- Create a new account as personal or professional. Add a payment method for development and academic purposes.
- Log in into AWS console with your credentials.

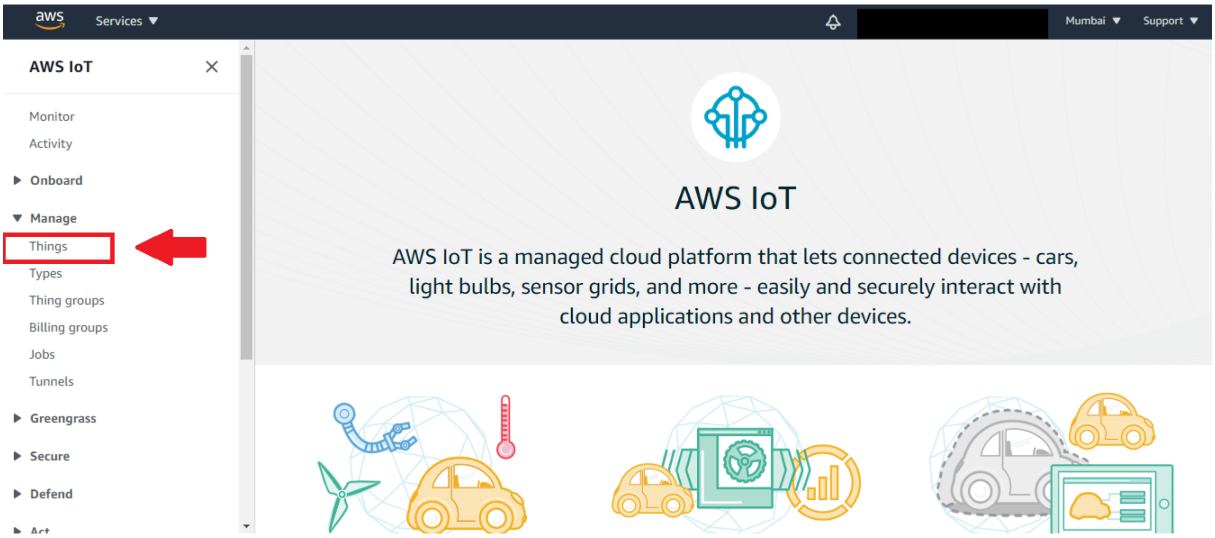
Step 2:

Choose the ***AWS IoT core*** from the given services list by scroll down the window:



Step 3:

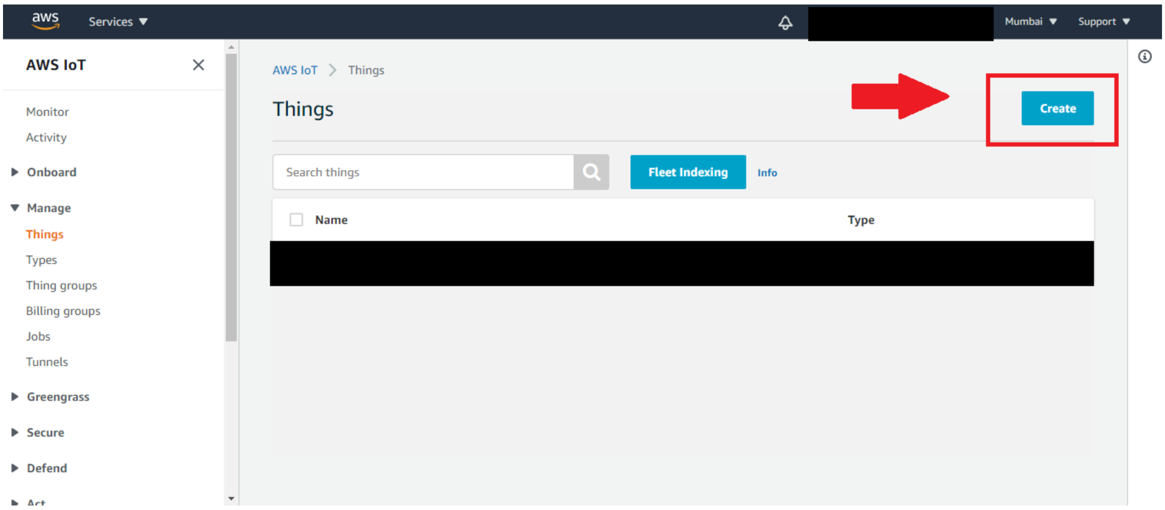
Next, a window will come up like this then you can see the left panel with numerous options. **Click** on the ***Manage*** ⟶ ***Things.*** Prior to that, you have to select the region that you need to use. Here I have selected ***Asia Pacific (Mumbai)ap-south-1*** as my region of services**.**



AWS IoT is a managed cloud platform that lets connected devices - cars, light bulbs, sensor grids, and more - easily and securely interact with cloud applications and other devices.

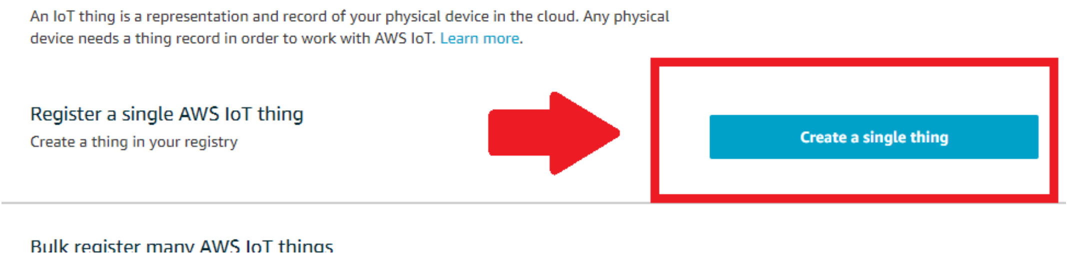
Step 4:

After the selection, you can see a window with ***Create*** option. So, ***click*** on the ***create*** option for creating a thing.



[redacted]

Again, click on ***Create a single thing*** option.

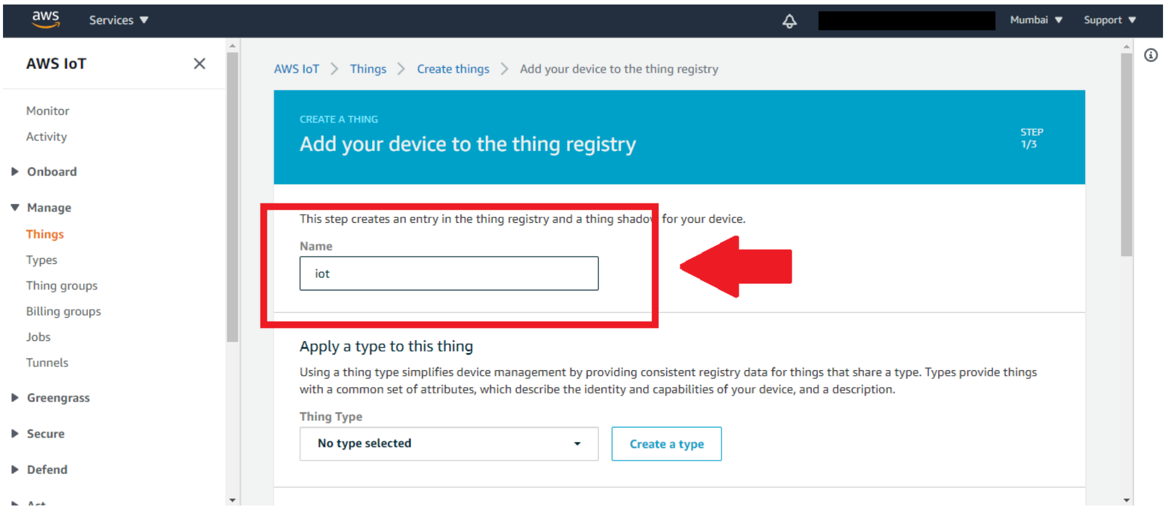


An IoT thing is a representation and record of your physical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. Learn more.

Register a single AWS IoT thing
Create a thing in your registry
Create a single thing
Bulk register many AWS IoT things

Step 5:

After clicking the ***Create a single thing option,*** we have a provision of giving a name to our thing. So here I give it as ***iot***. You can have any name as your wish and **click** next on below for further procedures.



Add your device to the thing registry

This step creates an entry in the thing registry and a thing shadow for your device.

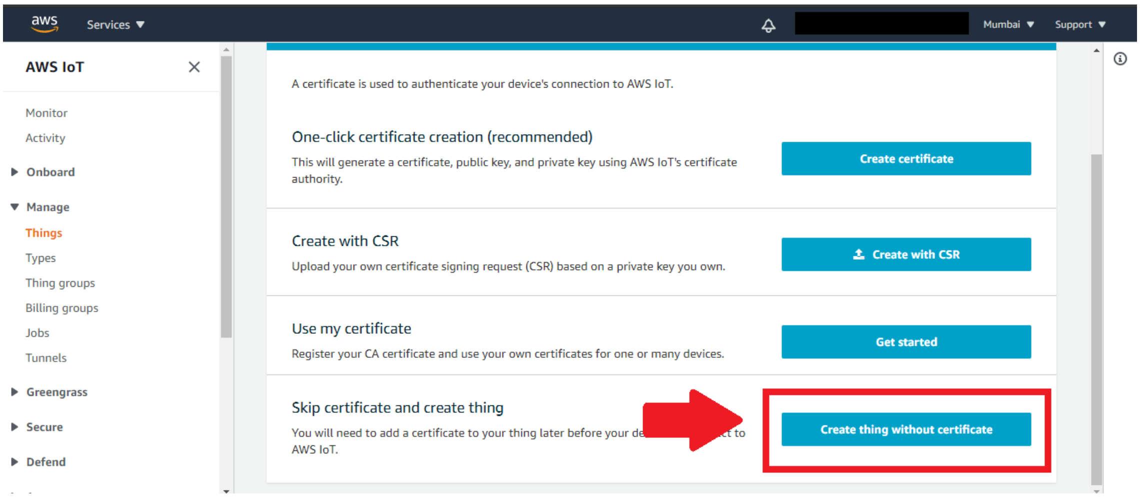
Name: iot

Apply a type to this thing

Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type: No type selected

From here, select ***Create thing without certificate,*** then it will guide us to the ***home*** page where we can see there our created thing on the ***dashboard***.



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.

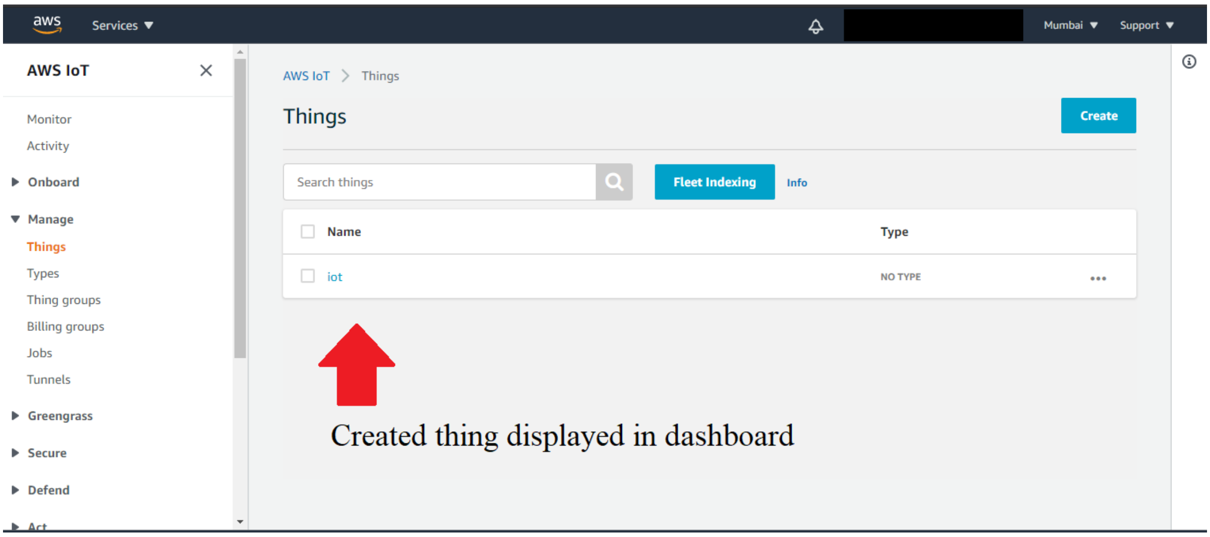
Use my certificate

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 de ct to AWS IoT.

Created thing on the dashboard 👇:

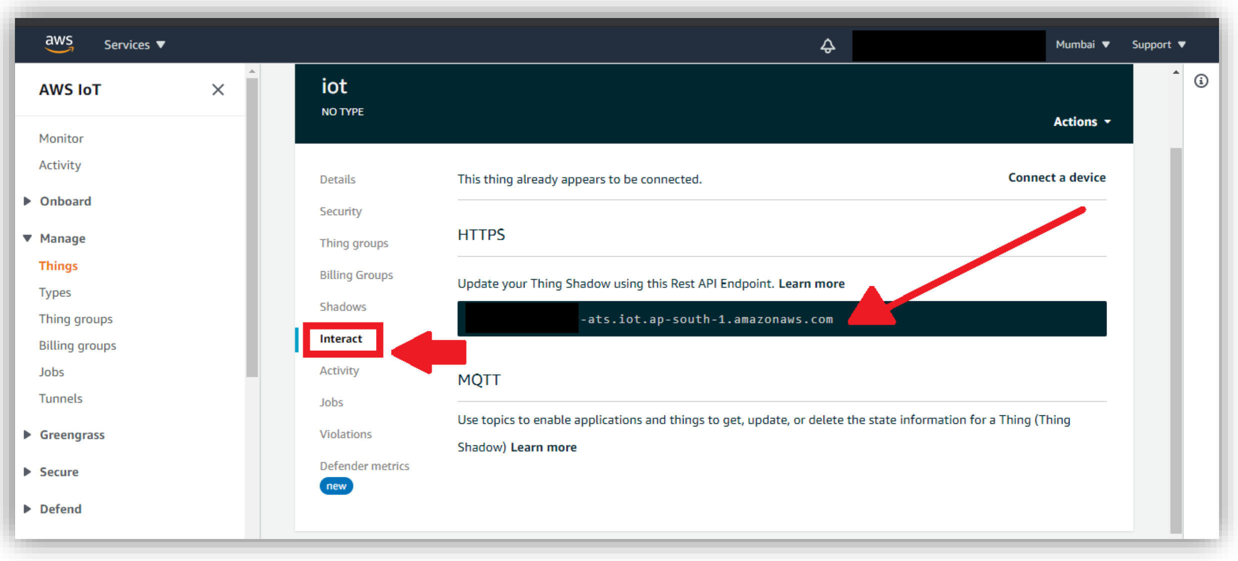


Created thing displayed in dashboard

Step 6:

So now we are look into the connection parameter for ***iot core*** with ***ESP8266***.

1. **Click** on the thing that you have created right now. Here I say ***iot.*** From the window select ***interact*** option. Now you can see our first connection parameter for the source code that is the ***REST API endpoint*** please copy the entire string***.***



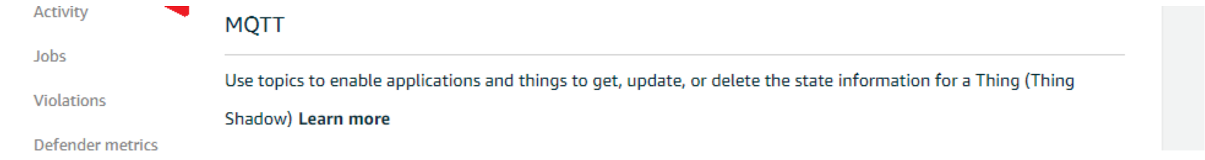
This thing already appears to be connected.

Update your Thing Shadow using this Rest API Endpoint. **Learn more**

```
[redacted]-ats.iot.ap-south-1.amazonaws.com
```

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow) **Learn more**

2. The next one is our ***publish topic*** for iot core. So currently on the MQTT side, you couldn't see that.



Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow) **Learn more**

So, it has a format: ***$aws/things/thing name/shadow/update*** [Provide ***thing name***]

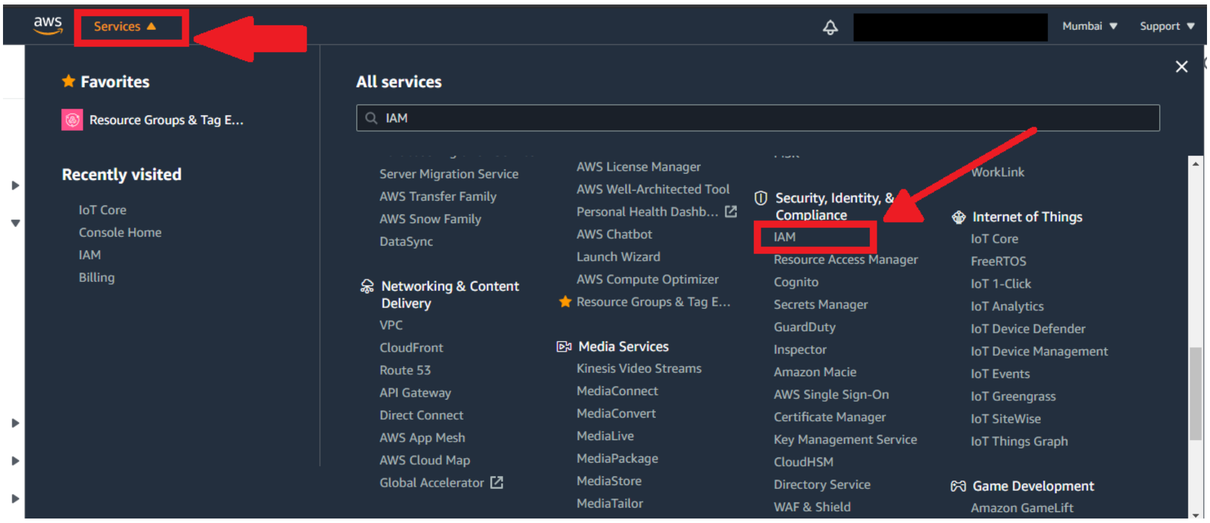
e.g.: ***$aws/things/iot/shadow/update*** please note it for ***source code*** purpose.

DO WE NEED IAM 🙄?

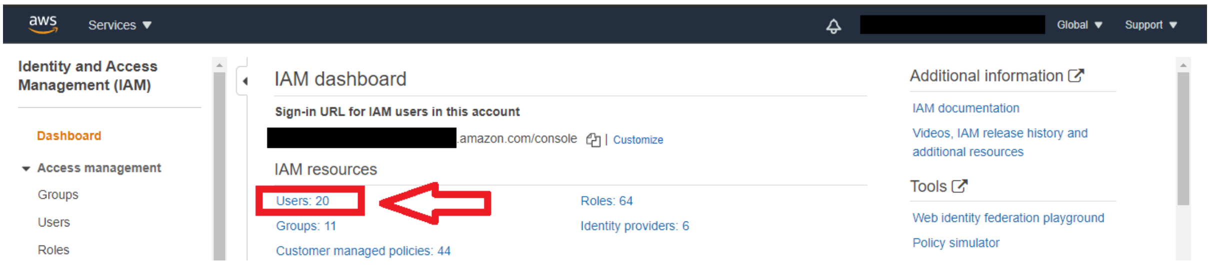
AWS Identity and Access Management (**IAM**) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. Here I opted for a ***root user** method.* But AWS strongly recommends us to practise of using the root user only to create your first **IAM** user.

Step 7:

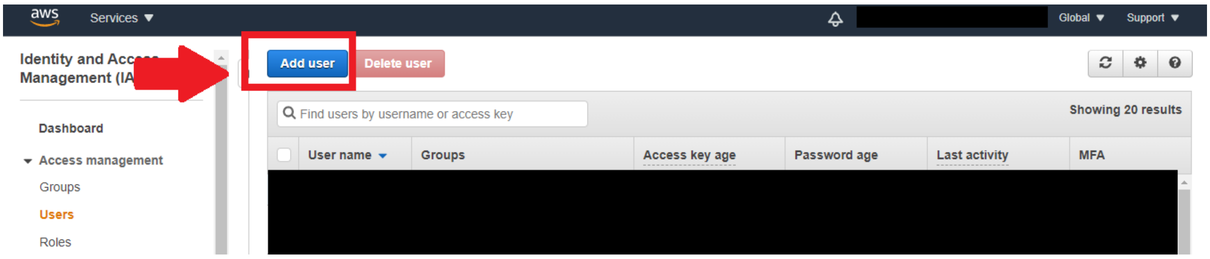
Click on the Services from the top window and select **IAM** from Security, identity & Compliance section.



Select **Users** from the options.

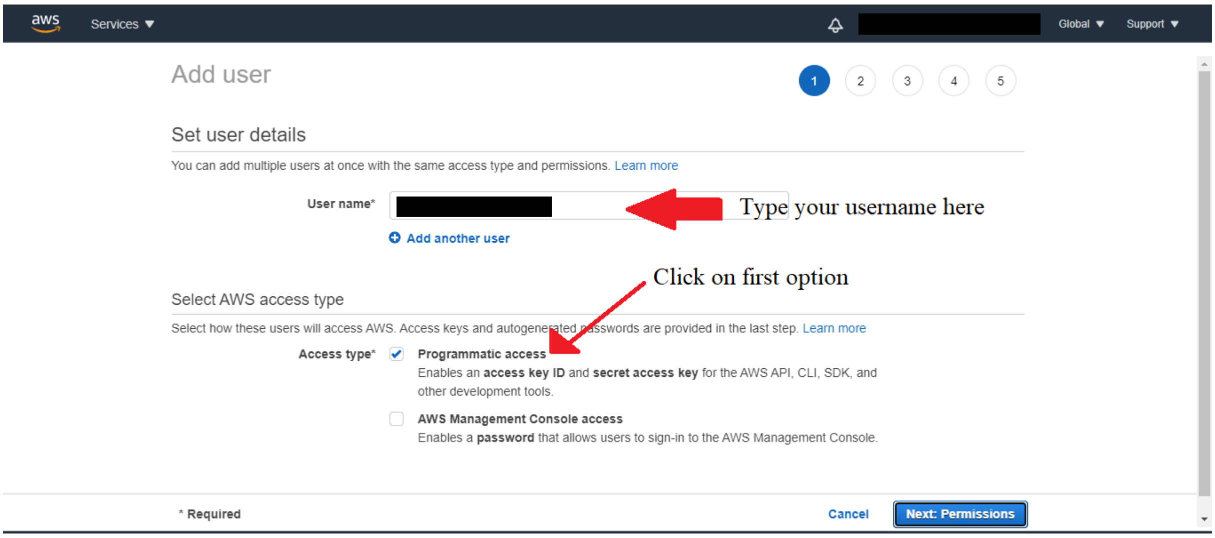

aws
Services ▼
[redacted]
Global ▼
Support ▼
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
IAM dashboard
Sign-in URL for IAM users in this account
[redacted] amazon.com/console | Customize
IAM resources
Users: 20
Groups: 11
Customer managed policies: 44
Roles: 64
Identity providers: 6
Additional information
IAM documentation
Videos, IAM release history and additional resources
Tools
Web identity federation playground
Policy simulator

Select ***Add user*** or you can use it as ***root user*** method.



Step 8:

First type your ***username*** and then give the ***programmatic access*** to the user. Then click ***next: permission.***



[redacted]

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* [redacted]

Type your username here

Add another user

Click on first option

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* ☑ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

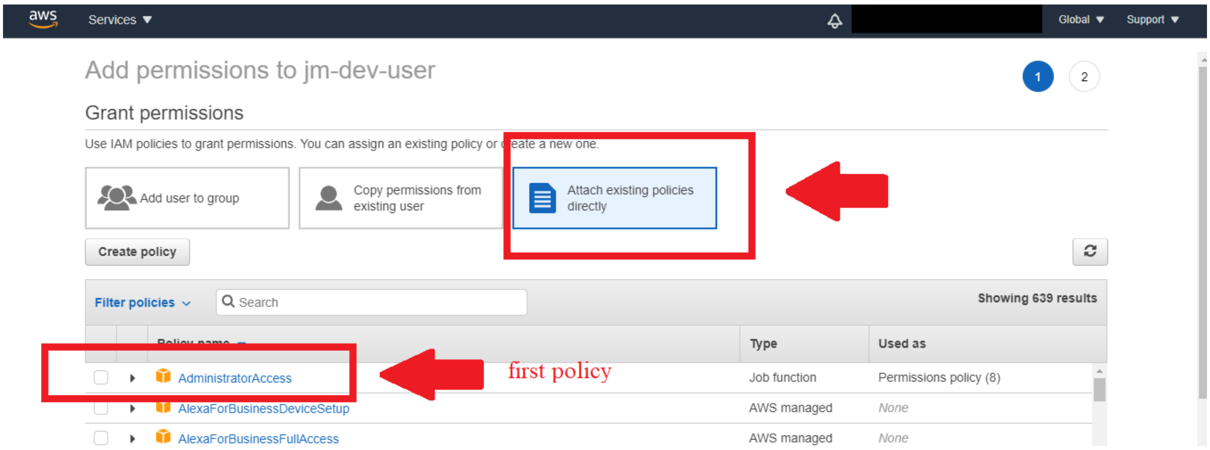
□ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

Cancel **Next: Permissions**

Attach exiting policies for our communication with device. Policies are:

1. **AdministratorAccess**
2. **AWSIoTFullAccess**



[redacted]

Add permissions to jm-dev-user

1 2

Grant permissions

Use IAM policies to grant permissions. You can assign an existing policy or create a new one.

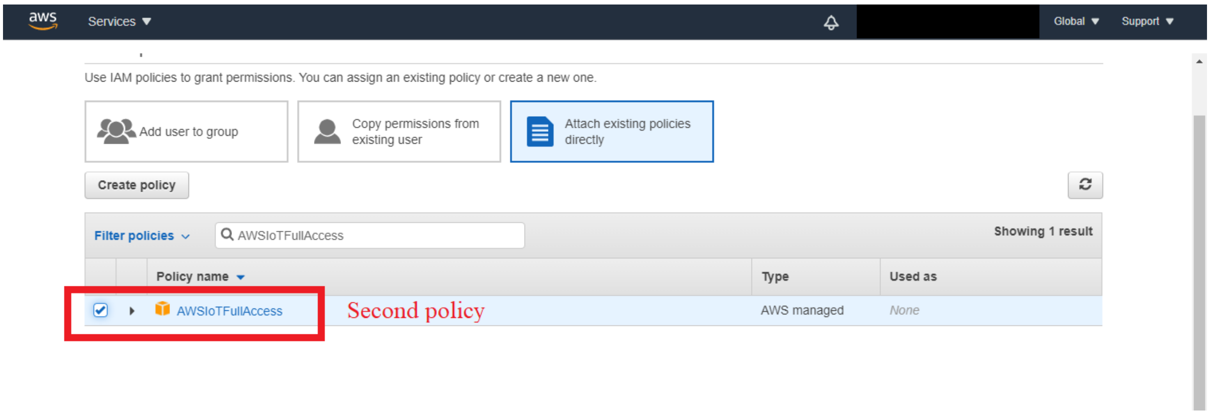
Add user to group | Copy permissions from existing user | Attach existing policies directly

Create policy

Filter policies ⌄ Search — Showing 639 results

	Policy name ▾	Type	Used as
□	AdministratorAccess	Job function	Permissions policy (8)
□	AlexaForBusinessDeviceSetup	AWS managed	None
□	AlexaForBusinessFullAccess	AWS managed	None

first policy



[redacted]

Use IAM policies to grant permissions. You can assign an existing policy or create a new one.

Add user to group | Copy permissions from existing user | Attach existing policies directly

Create policy

Filter policies ⌄ AWSIoTFullAccess — Showing 1 result

	Policy name ▾	Type	Used as
☑	AWSIoTFullAccess	AWS managed	None

Second policy

Then progress to the last page by clicking **next** and **review**. Then you can see that a **.csv** file ready to download, it contains the ***Access key ID*** & ***Secret Access Key*** that we need to use in the ***source code.***

CONNECTING ESP8266 WITH AWS IOT CORE :

So, we have currently ***three*** main sets of credentials here. We have to use it in ***ESP8266*** source code for establishing connection.

From **Step 6: REST API endpoint & Publish topic**

From **Step 8: Key pairs [*Access key ID* & *Secret Access Key*]**

Apart from these we need three more parameters for ESP8266 and IoT core connection. That are:

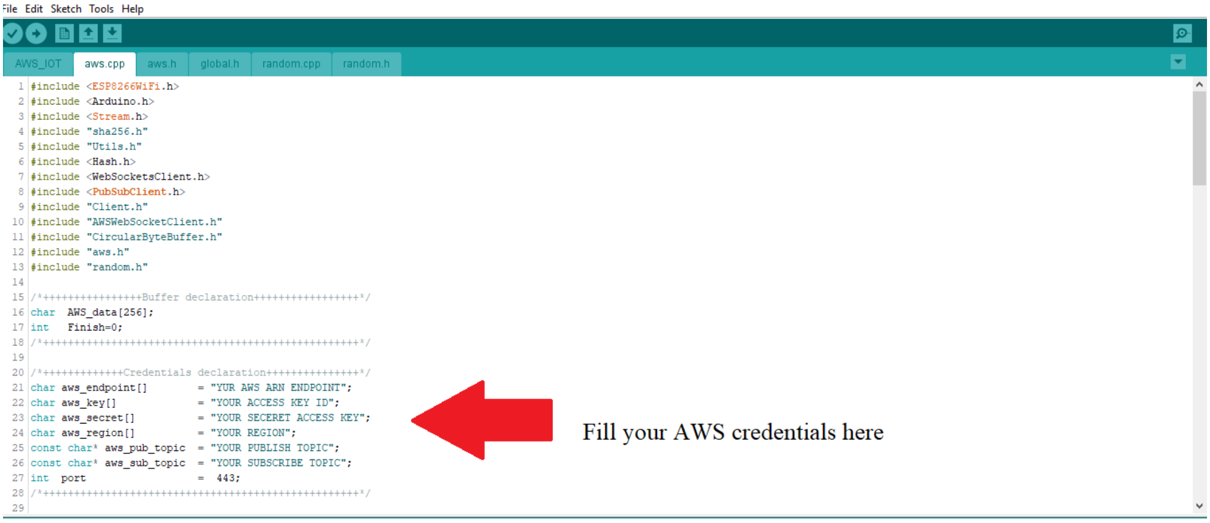
- **Region of services**: My region is **"ap-south-1"** [Paste the region that you are selected]
- **Subscribe topic** **:** You have to give a subscription topic as your wish [Mine is "**incoming**"].
- **Port** **:** For an encrypted communication we need **443** as port number.

So, I can give here an example table for the credentials that based on my ***iot*** named thing creation:

Parameters	Value
REST API endpoint	*************-ats.iot.ap-south-1.amazonaws.com
Publish topic	AK******************
Access key ID	21**************************************
Secret Access Key	ap-south-1
Region of services	$aws/things/iot/shadow/update
Subscribe topic	incoming
Port	443

Table 1.0

Based on your ***thing*** creation there will be slight changes in the credentials. But don't bother that, just follow this pattern for credential identification. Paste the details from ***Table 1.0*** in **aws.cpp** of **ESP8266** source code.

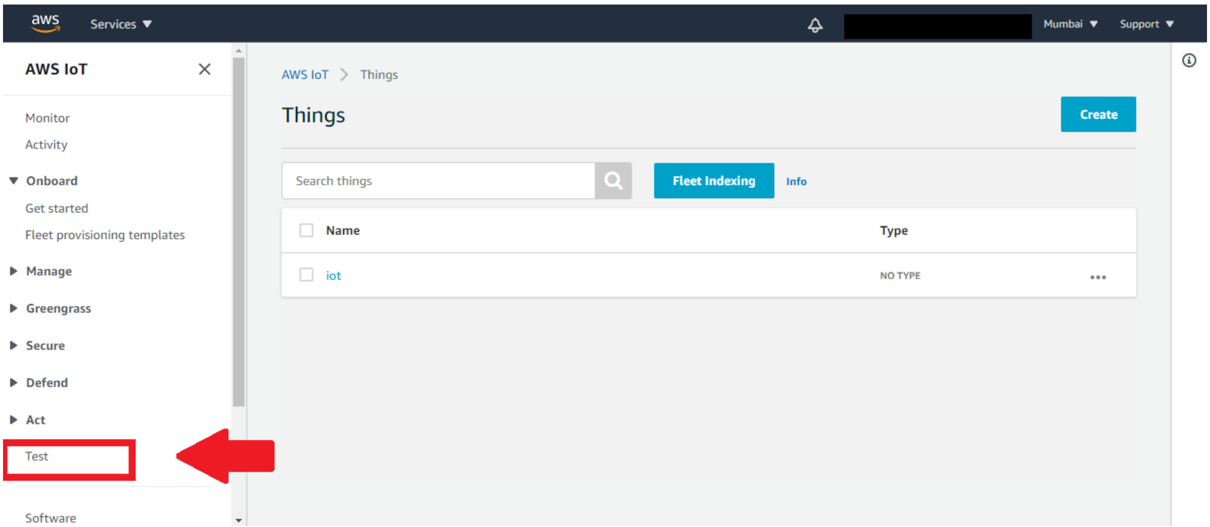


```
#include <ESP8266WiFi.h>
#include <Arduino.h>
#include <Stream.h>
#include "sha256.h"
#include "Utils.h"
#include <Hash.h>
#include <WebSocketsClient.h>
#include <PubSubClient.h>
#include "Client.h"
#include "AWSWebSocketClient.h"
#include "CircularByteBuffer.h"
#include "aws.h"
#include "random.h"

/*+++++++++++++++Buffer declaration++++++++++++++++*/
char  AWS_data[256];
int   Finish=0;
/*++++++++++++++++++++++++++++++++++++++++++++++++++*/


/*++++++++++++Credentials declaration++++++++++++++*/
char aws_endpoint[]        = "YUR AWS ARN ENDPOINT";
char aws_key[]             = "YOUR ACCESS KEY ID";
char aws_secret[]          = "YOUR SECERET ACCESS KEY";
char aws_region[]          = "YOUR REGION";
const char* aws_pub_topic  = "YOUR PUBLISH TOPIC";
const char* aws_sub_topic  = "YOUR SUBSCRIBE TOPIC";
int  port                  =  443;
/*++++++++++++++++++++++++++++++++++++++++++++++++++*/

```

Fill your AWS credentials here

TESTING IoT CORE WITH ESP8266 📢

So now we need to test the whole setup with ***AWS IoT Core*** and ***ESP8266***.For that just click on the **test** option from the left panel as shown in figure.

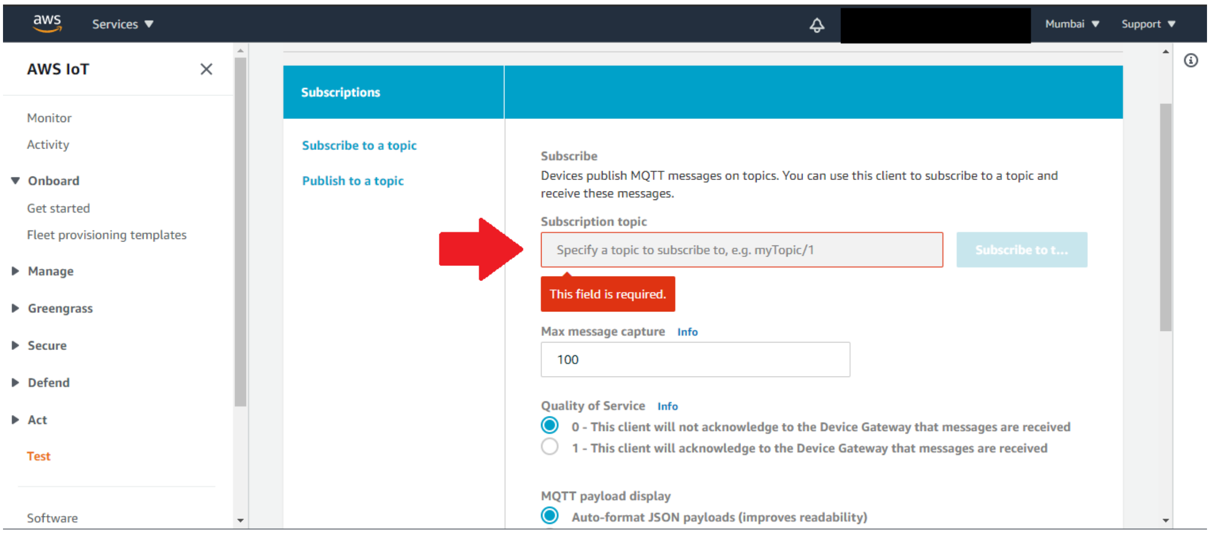


□ Name	Type
□ iot	NO TYPE

A new window will appear like this. Now input our ***subscribe*** and ***publish*** topic to the given slot as follows:

First, we need to give the ***subscribe topic*** for AWS IoT core. We should give our ESP8266 ***publish topic*** here because *IoT core subscribes to the message that published from ESP8266*.So let's go to the subscription side as shown in the figure, we have all data formats in the "***CONNECTING ESP8266 WITH AWS IOT CORE"*** section.

So ESP8266 Publishes in "$aws/things/iot/shadow/update" topic. Let give it into the input slot.



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

This field is required.

Max message capture: 100

Quality of Service

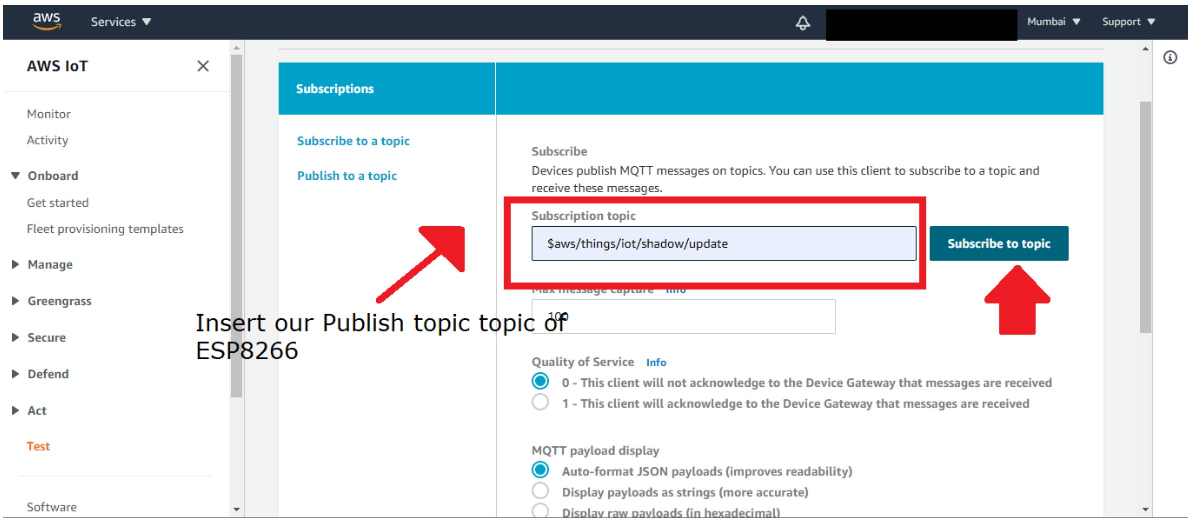
● 0 - This client will not acknowledge to the Device Gateway that messages are received

○ 1 - This client will acknowledge to the Device Gateway that messages are received

MQTT payload display

● Auto-format JSON payloads (improves readability)

Insert the **Publish topic** and **click** the ***Subscribe to topic*** button as shown:



Insert our Publish topic topic of ESP8266

Subscribe

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

Subscription topic: $aws/things/iot/shadow/update

Subscribe to topic

Max message capture Info: 100

Quality of Service Info

● 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

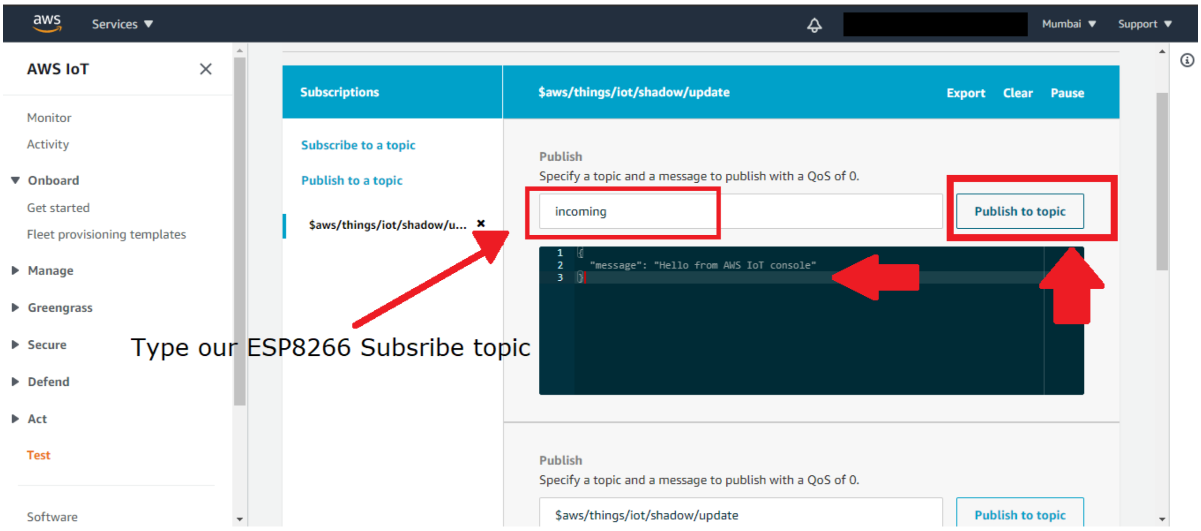
MQTT payload display

● Auto-format JSON payloads (improves readability)

○ Display payloads as strings (more accurate)

○ Display raw payloads (in hexadecimal)

Next, similarly, insert our ESP8266 ***Subscribe topic*** and click the ***Publish to topic*** button**.** In the black window we can see our published message from AWS IoT core to ESP8266 is ***"Hello from AWS IoT console".*** We have to observe it on ESP8266 **serial monitor**.



Type our ESP8266 Subsribe topic

Publish

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

incoming

Publish to topic

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

Publish

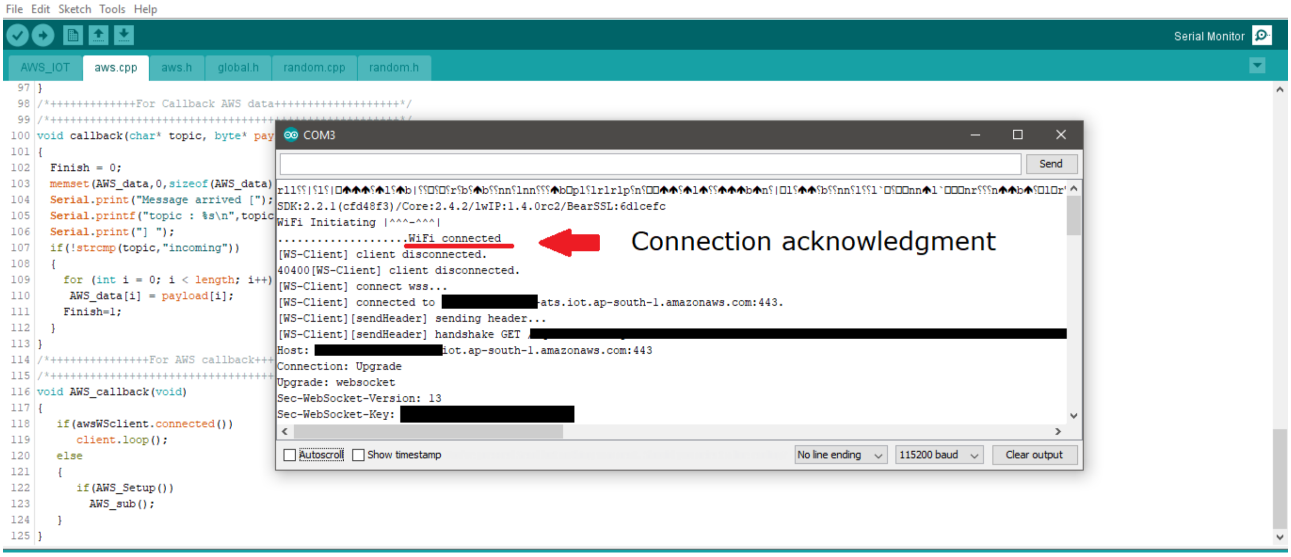
Specify a topic and a message to publish with a QoS of 0.

$aws/things/iot/shadow/update

Publish to topic

SERIAL MONITOR DEBUG 🔍:

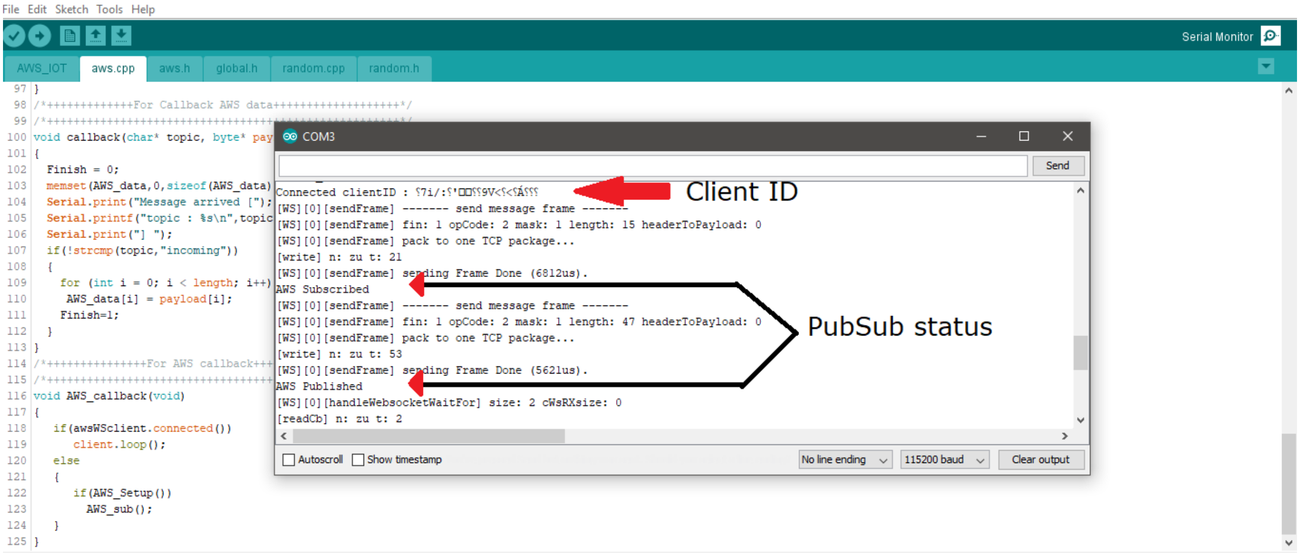
- Download the repository and open **LIB** folder and unzip each library and place it on your arduino library folder.
- Open the code and provide your credentials as in the manner of **Table 1.0** on **aws.cpp** and provide your WIFI router **ssid** and **password** in **global.h.**
- Compile the code and upload it to your ESP8266.
- Restart the module and open the serial monitor with baud rate of **115200**.
- Now you can see the connection acknowledgment as" ***WIFI connected"*** if it is a success as like below:



```
}
/*+++++++++++++For Callback AWS data++++++++++++++++++*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void callback(char* topic, byte* pay
{
  Finish = 0;
  memset(AWS_data,0,sizeof(AWS_data)
  Serial.print("Message arrived [");
  Serial.printf("topic : %s\n",topic
  Serial.print("] ");
  if(!strcmp(topic,"incoming"))
  {
    for (int i = 0; i < length; i++)
     AWS_data[i] = payload[i];
    Finish=1;
  }
}
/*++++++++++++++For AWS callback+++
/*+++++++++++++++++++++++++++++++++
void AWS_callback(void)
{
   if(awsWSclient.connected())
      client.loop();
   else
   {
      if(AWS_Setup())
        AWS_sub();
   }
}
```

```
rll§§|§l§|□♠♠♠§♠l§♠b|§§□§□§r§b§♠b§§nn§lnn§§§♠b□pl§lrlrlp§n§□□♠♠§♠l♠§§♠♠♠b♠n§|□l§♠♠§b§§nn§l§§l`□§□□nn♠l`□□□nr§§§n♠♠b♠§□l□r'
SDK:2.2.1(cfd48f3)/Core:2.4.2/lwIP:1.4.0rc2/BearSSL:6d1cefc
WiFi Initiating |^^^-^^^|
....................WiFi connected
[WS-Client] client disconnected.
40400[WS-Client] client disconnected.
[WS-Client] connect wss...
[WS-Client] connected to [redacted]ats.iot.ap-south-1.amazonaws.com:443.
[WS-Client][sendHeader] sending header...
[WS-Client][sendHeader] handshake GET [redacted]
Host: [redacted]iot.ap-south-1.amazonaws.com:443
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: [redacted]
```

- And we can see the connected ***client ID*** and ***AWS published*** and ***subscribed*** messages ***status*** from esp8266 if ***AWS IoT core*** established successful connection.

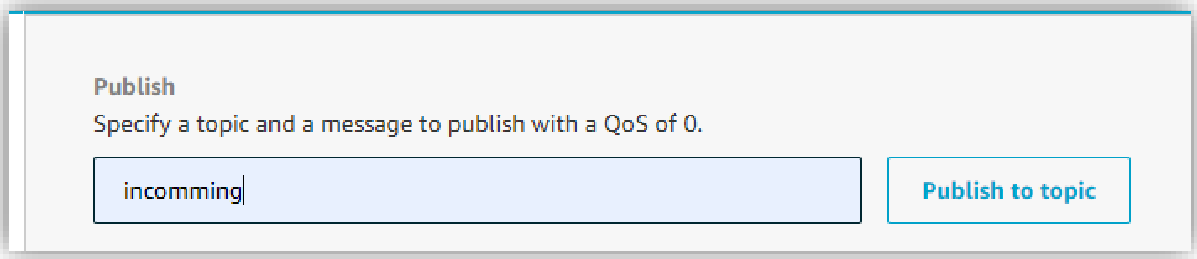


```
}
/*+++++++++++++For Callback AWS data++++++++++++++++++*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void callback(char* topic, byte* pay
{
  Finish = 0;
  memset(AWS_data,0,sizeof(AWS_data)
  Serial.print("Message arrived [");
  Serial.printf("topic : %s\n",topic
  Serial.print("] ");
  if(!strcmp(topic,"incoming"))
  {
    for (int i = 0; i < length; i++)
     AWS_data[i] = payload[i];
    Finish=1;
  }
}
/*++++++++++++++For AWS callback+++
/*+++++++++++++++++++++++++++++++++
void AWS_callback(void)
{
   if(awsWSclient.connected())
      client.loop();
   else
   {
      if(AWS_Setup())
        AWS_sub();
   }
}
```

```
Connected clientID : §7i/:§'□□§§9V<§<§Á§§§
[WS][0][sendFrame] ------- send message frame -------
[WS][0][sendFrame] fin: 1 opCode: 2 mask: 1 length: 15 headerToPayload: 0
[WS][0][sendFrame] pack to one TCP package...
[write] n: zu t: 21
[WS][0][sendFrame] sending Frame Done (6812us).
AWS Subscribed
[WS][0][sendFrame] ------- send message frame -------
[WS][0][sendFrame] fin: 1 opCode: 2 mask: 1 length: 47 headerToPayload: 0
[WS][0][sendFrame] pack to one TCP package...
[write] n: zu t: 53
[WS][0][sendFrame] sending Frame Done (5621us).
AWS Published
[WS][0][handleWebsocketWaitFor] size: 2 cWsRXsize: 0
[readCb] n: zu t: 2
```

Now its good to go for IoT dashboard - serial monitor **Pub Sub** message displays.

When we click ***Publish to topic*** button from the **test** panel of IoT core through the topic *"**incoming**"*

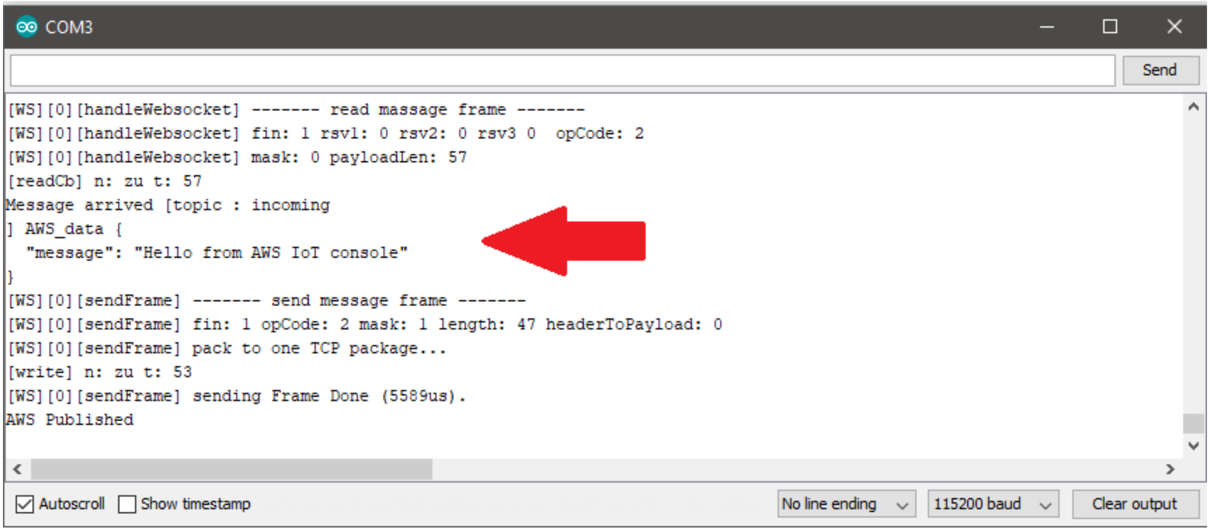


Publish

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

incomming

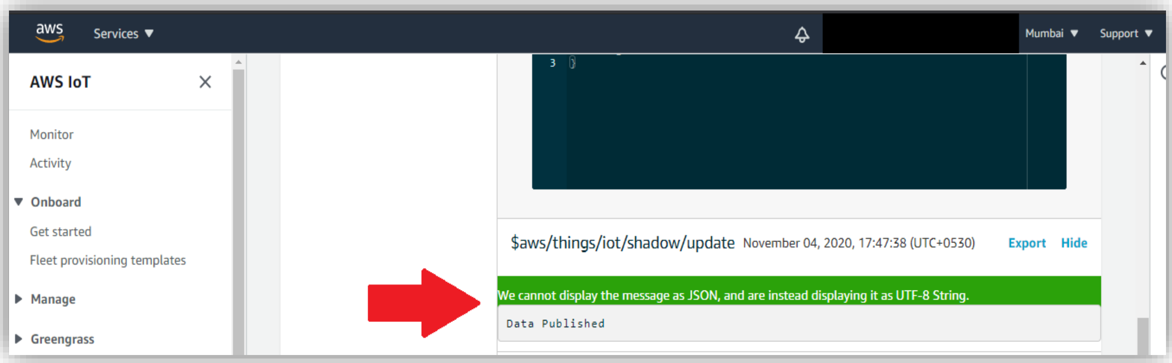
The ***"Hello from AWS IoT console"*** message will publish to ***ESP8266.***You can observe it on serial window***.*** You can edit the message on dashboard as your wish, example***: "Hello".***



```
[WS][0][handleWebsocket] ------- read massage frame -------
[WS][0][handleWebsocket] fin: 1 rsv1: 0 rsv2: 0 rsv3 0  opCode: 2
[WS][0][handleWebsocket] mask: 0 payloadLen: 57
[readCb] n: zu t: 57
Message arrived [topic : incoming
] AWS_data {
  "message": "Hello from AWS IoT console"
}
[WS][0][sendFrame] ------- send message frame -------
[WS][0][sendFrame] fin: 1 opCode: 2 mask: 1 length: 47 headerToPayload: 0
[WS][0][sendFrame] pack to one TCP package...
[write] n: zu t: 53
[WS][0][sendFrame] sending Frame Done (5589us).
AWS Published
```

☑ Autoscroll □ Show timestamp   No line ending   115200 baud

When the message is received/**subscribed**, the **ESP8266** will send an acknowledgment to the **IoT core** as ***"Data published"*** as shown below:



$aws/things/iot/shadow/update November 04, 2020, 17:47:38 (UTC+0530)

We cannot display the message as JSON, and are instead displaying it as UTF-8 String.

```
Data Published
```

So, our **Publish** and **Subscribe** method completed, this is all about the ***AWS IoT Core*** simulation.

Thank you!!