ENEB 341 - Internet of Things

Due date: 17 December, 2021

Robin Godinho

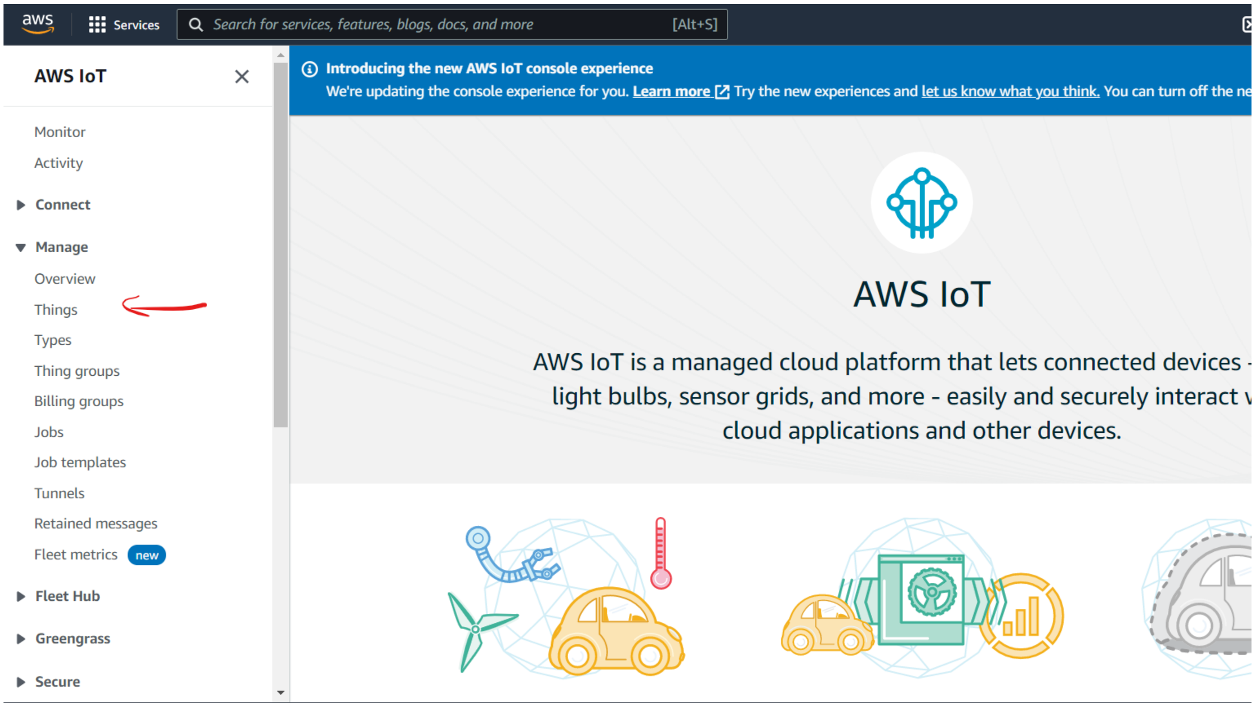
Abstract

The purpose of the following lab was to serve as an introduction into one of the world's most powerful cloud services, Amazon Web Services. This lab required us to become familiar with one of the many services offered by AWS. For this lab, our main focus was with the AWS 'IoT core.' The objective for the lab was to make use of the ESP32 Arduino board, in order to send and receive messages from the cloud, using the AWS IoT core service. The following lab incorporated the creation of a 'Thing,' which came with a certificate, device private key, Amazon Root CA 1 and lastly the creation of a policy.

AWS Setup

1. ‘Creating a Thing’

-In the homepage, select the ‘services’ drop-down menu and select the ‘IoT Core.’
-In the ‘IoT Core’ homepage, go to the left-hand options. Select the drop-down menu called ‘Manage’ and select ‘Things.’



Introducing the new AWS IoT console experience

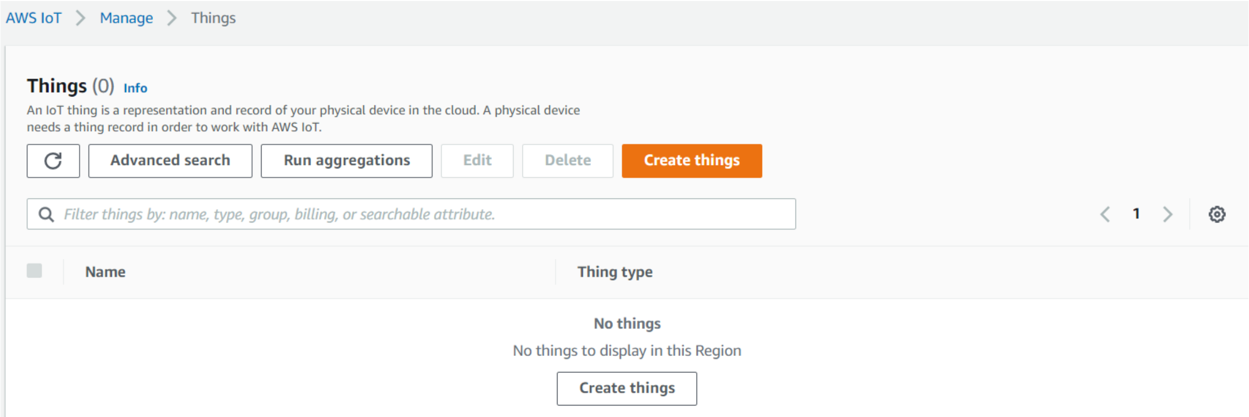
We're updating the console experience for you. Learn more Try the new experiences and let us know what you think. You can turn off the ne

AWS IoT

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

Figure 1: Creating a ‘Thing’ in the IoT core

Once you have clicked on ‘Things,’ the following menu should show up and click on the orange icon called ‘create things.’



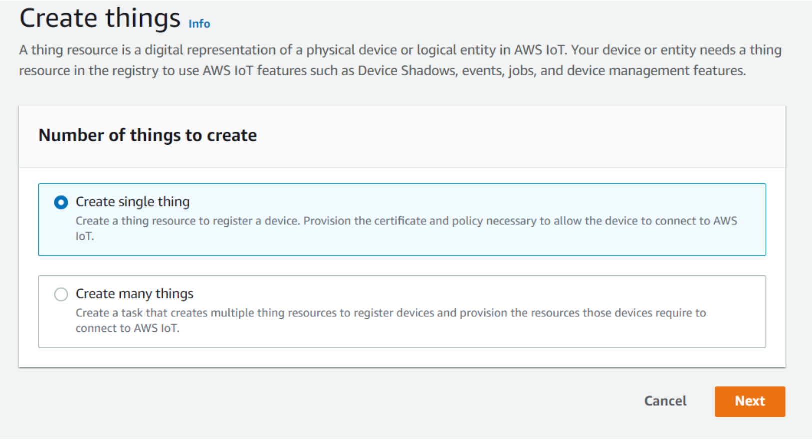
An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

No things to display in this Region

Figure 2: Create things

A thing is a representation of our physical device on AWS. Once you click on ‘Create things’, you will be asked several questions, in the sequence below:

- Number of things to create – Click on ‘Create single thing’ for now, and click ‘Next’
- Thing name – Give it a suitable name, like esp32_0
- Leave out the optional additional configurations
- In the Device Shadow section, select ‘No shadow’ and click ‘Next’. We’ll cover device shadows in a separate article.
- For the Device Certificate section, click on the recommended ‘Auto-generate a new certificate’, and click on ‘Next’
- In the Policies section, do nothing for now (we will create and attach a policy to this thing in the immediate next section).
- Click on ‘Create thing’.



Create things

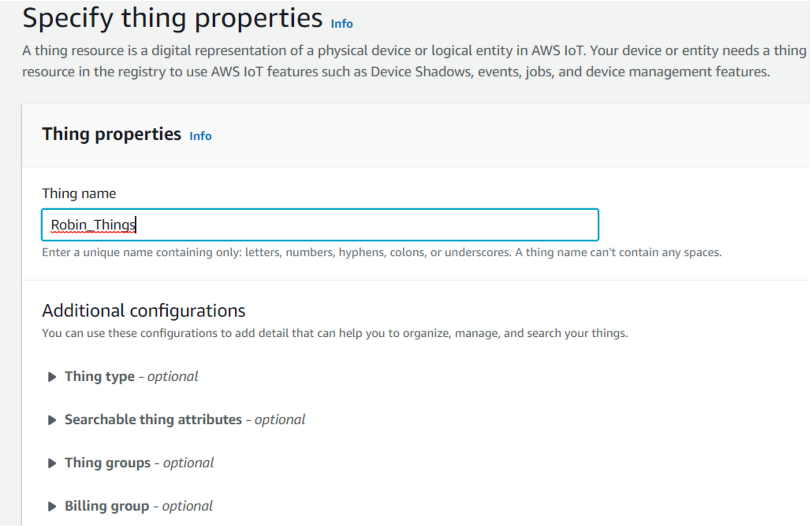
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Number of things to create

● Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

○ Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

Figure 3: Create a single thing



Specify thing properties

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties

Thing name

Robin_Things

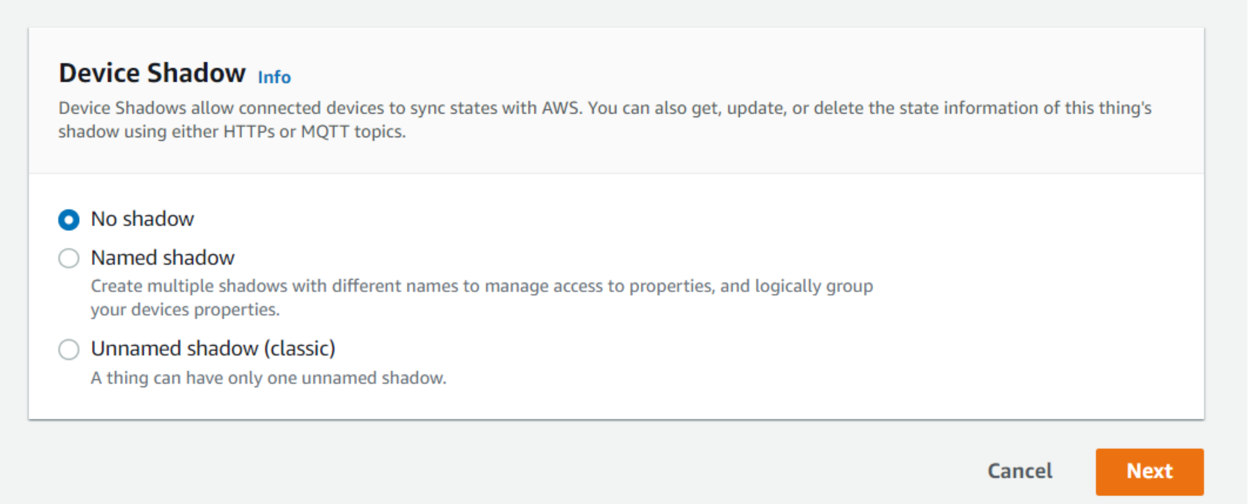
Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

- **Thing type** - *optional*
- **Searchable thing attributes** - *optional*
- **Thing groups** - *optional*
- **Billing group** - *optional*

Figure 4: Create a name for your Thing
Note: Leave out the additional configurations



Device Shadow

Info

Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

● No shadow

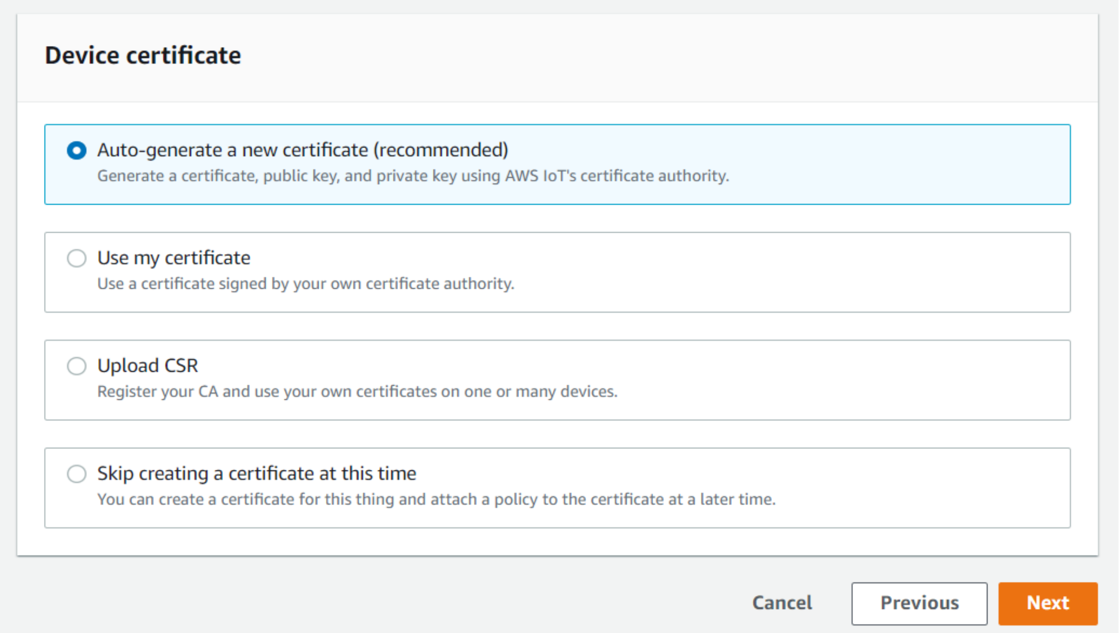
○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

○ Unnamed shadow (classic)
A thing can have only one unnamed shadow.

Figure 5: Select no Shadow for the connected AWS

2. Configure device certificate

- Select the 'Auto-generate a new certificate' and click next.



Device certificate

● Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

○ Use my certificate
Use a certificate signed by your own certificate authority.

○ Upload CSR
Register your CA and use your own certificates on one or many devices.

○ Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Figure 6: Auto-generate a new certificate

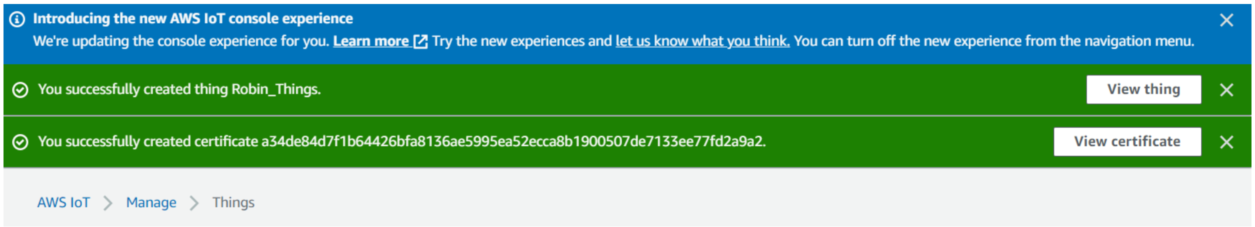
Skip the attach policy to the certificate section for now. Click on 'create now.'

3. Download certificates and keys

-For the following section, be sure to download only the highlighted files.

1. Device certificate
2. Private Key file
3. Amazon Root CA 1

Once you have downloaded the following files, click on “DONE” and the Thing should be created.



Introducing the new AWS IoT console experience
We're updating the console experience for you. Learn more Try the new experiences and let us know what you think. You can turn off the new experience from the navigation menu.

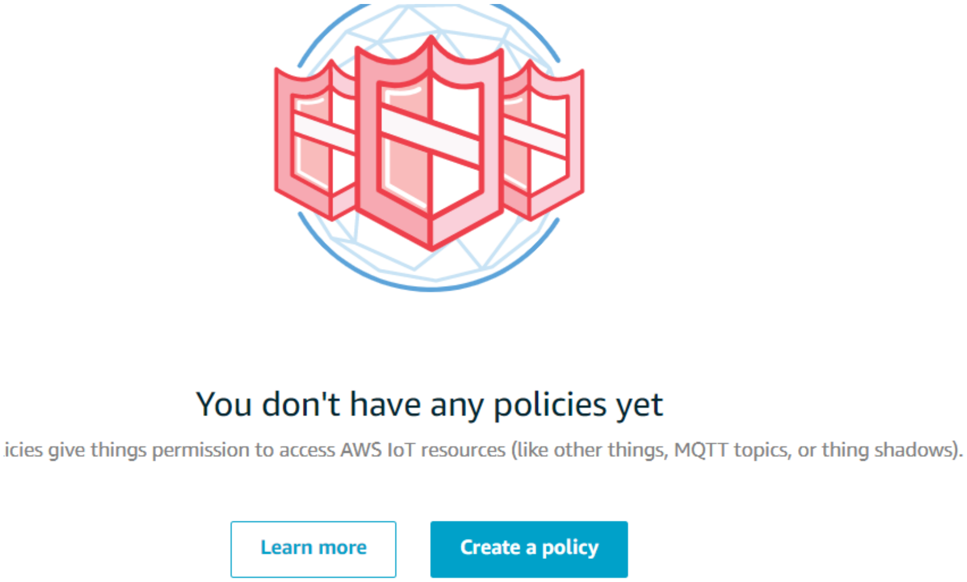
You successfully created thing Robin_Things.

You successfully created certificate a34de84d7f1b64426bfa8136ae5995ea52ecca8b1900507de7133ee77fd2a9a2.

Figure 7: The Thing has been created successfully

4. Attach a policy

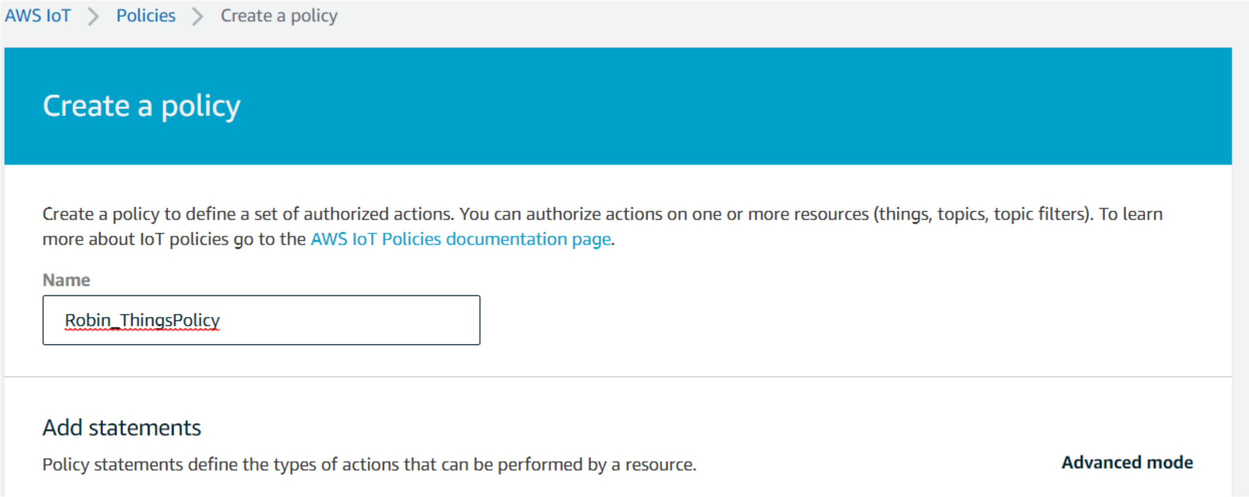
In the left-side menu, click on Secure -> Policies, and click on ‘Create’



You don't have any policies yet

icies give things permission to access AWS IoT resources (like other things, MQTT topics, or thing shadows).

Figure 8: Create a Policy under the ‘secure’ tab



Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name

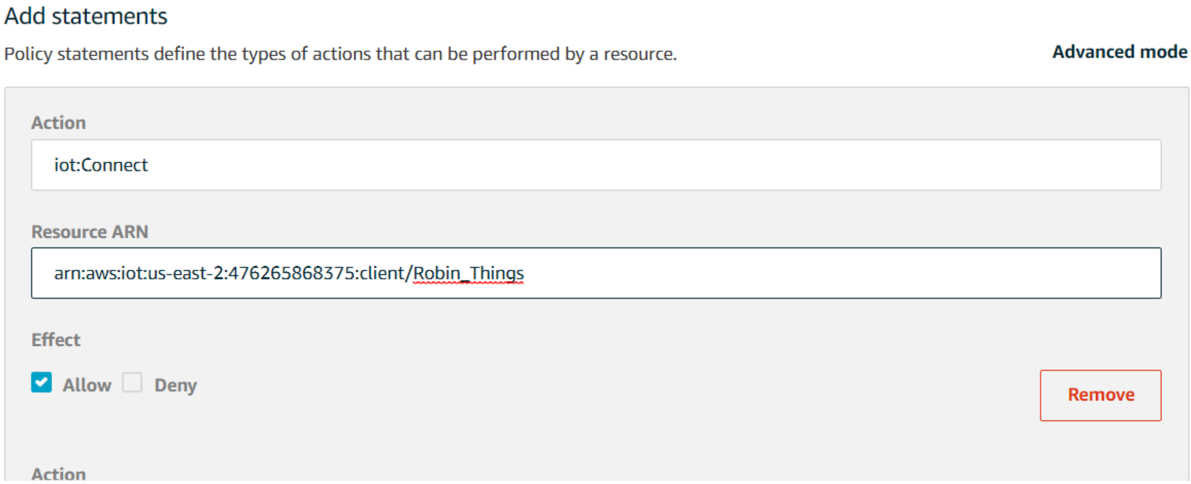
Robin_ThingsPolicy

Add statements

Policy statements define the types of actions that can be performed by a resource.

Figure 9: Create a suitable name for the Policy

In the form that opens up, give the policy a suitable name, like ‘ESP32 Policy’, and then add the following statements:
iot:Connect, iot:Subscribe, iot:Receive, iot:Publish
Note: Separate each action with a comma



Add statements

Policy statements define the types of actions that can be performed by a resource.

Action

iot:Connect

Resource ARN

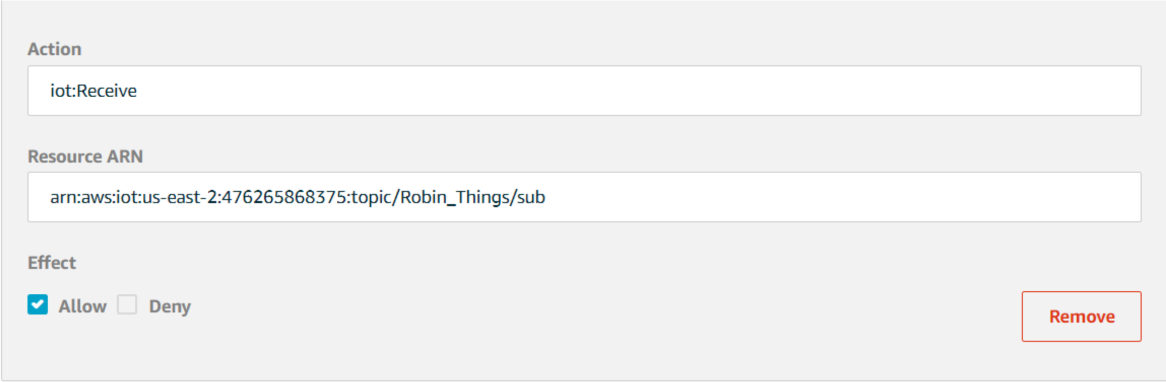
arn:aws:iot:us-east-2:476265868375:client/Robin_Things

Effect

☑ Allow □ Deny

Figure 10: IoT: Connect action

A pop-up will show up. Select the policy that you have recently created



Action

iot:Receive

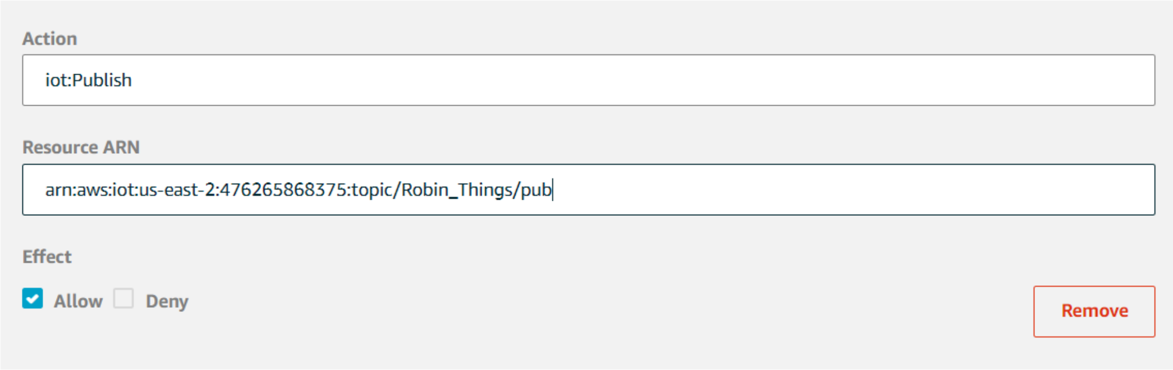
Resource ARN

arn:aws:iot:us-east-2:476265868375:topic/Robin_Things/sub

Effect

☑ Allow □ Deny

Figure 11: IoT: Receive action



Action

iot:Publish

Resource ARN

arn:aws:iot:us-east-2:476265868375:topic/Robin_Things/pub

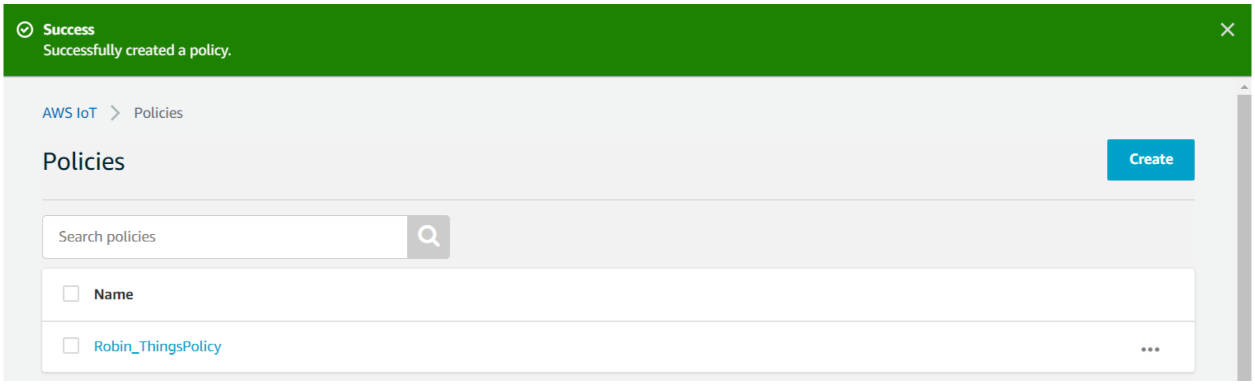
Effect

☑ Allow □ Deny

Figure 12: IoT: Publish action

We essentially give our thing the permission to connect, subscribe to ‘esp32/sub’ topic, receive messages from ‘esp32/sub’ topic, and publish to ‘esp32/pub’ topic.

In the ‘Effect’ section, check ‘Allow’ for each statement. After adding the statements, click on ‘Create’.

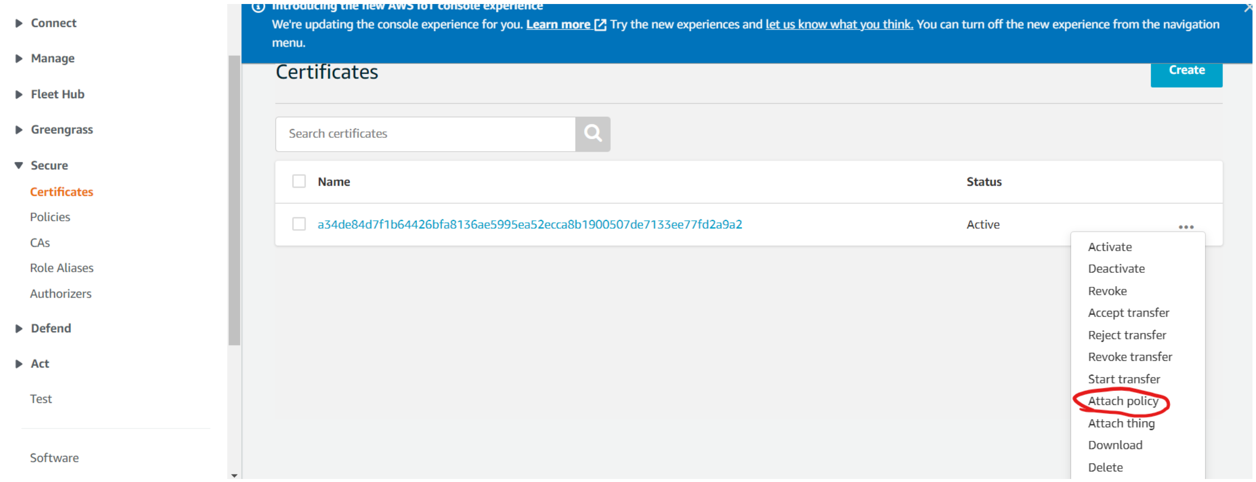


Success
Successfully created a policy.

□ Name
□ Robin_ThingsPolicy

Figure 13: Creation of the policy was successful

Next, within the ‘Secure’ menu on the left, click on Certificates. Click on the certificate of your device, go to ‘Actions’, and click on ‘Attach policy’.



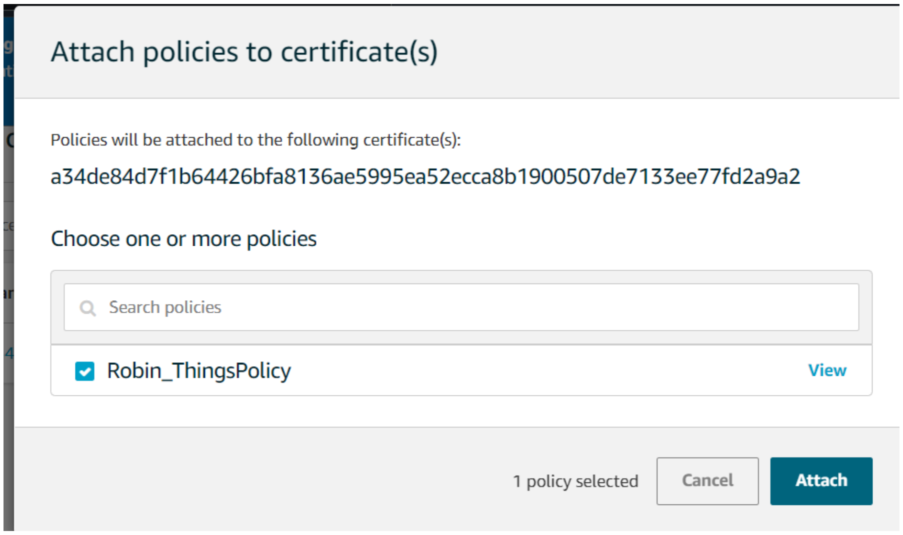
Introducing the new AWS IoT console experience

We're updating the console experience for you. Learn more Try the new experiences and let us know what you think. You can turn off the new experience from the navigation menu.

Certificates

Name	Status
a34de84d7f1b64426bfa8136ae5995ea52ecca8b1900507de7133ee77fd2a9a2	Active

Figure 14: Attaching the policy



Attach policies to certificate(s)

Policies will be attached to the following certificate(s):

a34de84d7f1b64426bfa8136ae5995ea52ecca8b1900507de7133ee77fd2a9a2

Choose one or more policies

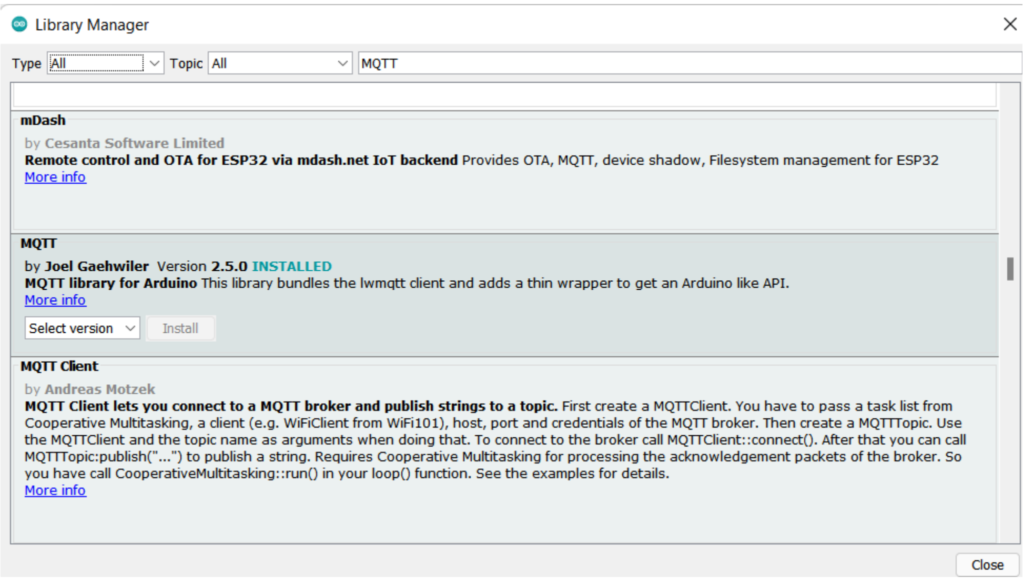
☑ Robin_ThingsPolicy

1 policy selected

Figure 15: Attach the policy

5. ESP32 Arduino Setup

The first step is to download the MQTT library from the Arduino library.



MQTT

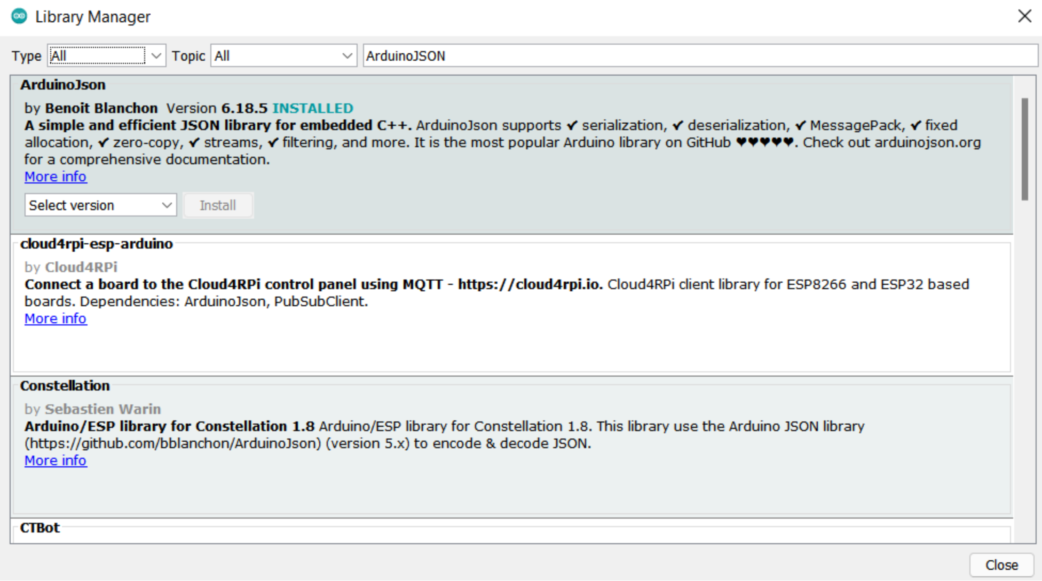
mDash
by Cesanta Software Limited
Remote control and OTA for ESP32 via mdash.net IoT backend Provides OTA, MQTT, device shadow, Filesystem management for ESP32
More info

MQTT
by **Joel Gaehwiler** Version **2.5.0** **INSTALLED**
MQTT library for Arduino This library bundles the lwmqtt client and adds a thin wrapper to get an Arduino like API.
More info

MQTT Client
by Andreas Motzek
MQTT Client lets you connect to a MQTT broker and publish strings to a topic. First create a MQTTClient. You have to pass a task list from Cooperative Multitasking, a client (e.g. WiFiClient from WiFi101), host, port and credentials of the MQTT broker. Then create a MQTTTopic. Use the MQTTClient and the topic name as arguments when doing that. To connect to the broker call MQTTClient::connect(). After that you can call MQTTTopic:publish("...") to publish a string. Requires Cooperative Multitasking for processing the acknowledgement packets of the broker. So you have call CooperativeMultitasking::run() in your loop() function. See the examples for details.
More info

Figure 16: Search and download the MQTT

Next, you need to download the ArduinoJSON library. In the library manager, search for ArduinoJSON and install the library by Benoit Blanchon.



ArduinoJSON

ArduinoJson
by **Benoit Blanchon** Version **6.18.5** **INSTALLED**
A simple and efficient JSON library for embedded C++. ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ MessagePack, ✔ fixed allocation, ✔ zero-copy, ✔ streams, ✔ filtering, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation.
More info

cloud4rpi-esp-arduino
by Cloud4RPi
Connect a board to the Cloud4RPi control panel using MQTT - https://cloud4rpi.io. Cloud4RPi client library for ESP8266 and ESP32 based boards. Dependencies: ArduinoJson, PubSubClient.
More info

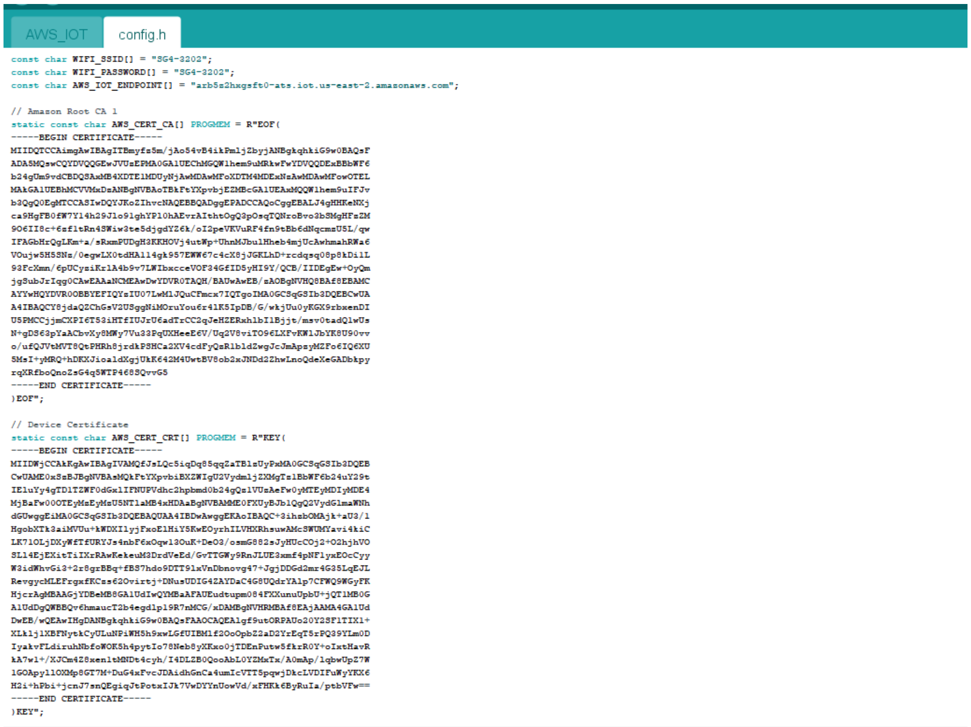
Constellation
by Sebastien Warin
Arduino/ESP library for Constellation 1.8 Arduino/ESP library for Constellation 1.8. This library use the Arduino JSON library (https://github.com/bblanchon/ArduinoJson) (version 5.x) to encode & decode JSON.
More info

CTBot

Figure 17: Search and download the ArduinoJSON

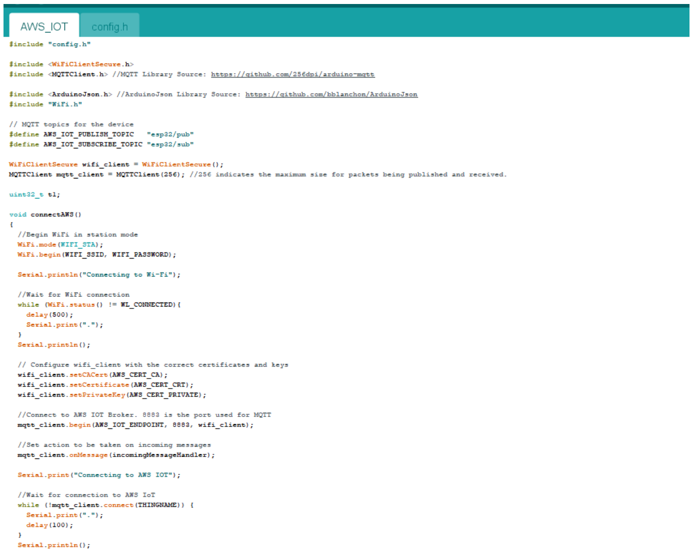
6. Download the AWS_IOT.ion and the config.h files

Before connecting the ESP32 to your computer, it is important to open and attach all of the downloaded files, such as the ‘Thing's’ certificate, device key and the AWS certificate CA to the code. Ensure that the WIFI SSID is attached as well. This will enable the device to read and send data to the cloud.



```
const char WIFI_SSID[] = "SG4-3202";
const char WIFI_PASSWORD[] = "SG4-3202";
const char AWS_IOT_ENDPOINT[] = "arb5z2hxgsft0-ats.iot.us-east-2.amazonaws.com";

// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
)EOF";

// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
MIIDWjCCAkKgAwIBAgIVAMQfJsLQc5iqDq85qqZaTBlzUyPxMA0GCSqGSIb3DQEB
CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t
IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0yMTEyMDIyMDE4
MjBaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh
dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC+3ihzbOMAjk+aU3/1
HgobXTk3aiMVUu+kWDXIlyjFxoElHiY5KwEOyrhILVHXRhsuwAMcSWUMYavi4kiC
LK71OLjDXyWfTfURYJs4nbF6xOqw13OuK+DeO3/osmG882sJyHUcCOj2+O2hjhVO
SL14EjEXitTiIXrRAwKekeuM3DrdVeEd/GvTTGWy9RnJLUE3xmf4pNF1yxEOcCyy
W3idWhvGi3+2r8grBBq+fBS7hdo9DTT91xVnDbnovg47+JgjDDGd2mr4G35LqEJL
RevgycMLEFrgxfKCzs62Ovirtj+DNusUDIG4ZAYDaC4G8UQdrYAlp7CFWQ9WGyFK
HjcrAgMBAAGjYDBeMB8GA1UdIwQYMBaAFAUEudtupm084FXXunuUpbU+jQT1MB0G
A1UdDgQWBBQv6hmaucT2b4egdlp19R7nMCG/xDAMBgNVHRMBAf8EAjAAMA4GA1Ud
DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEAlgf9utORPAUo20Y2SF1TIX1+
XLklj1XBFNytkCyULuNPiWH5h9xwLGfUIBMlf2OoOpbZ2aD2YrEqT5rPQ39YLm0D
IyakvFLdiruhNbfoWOK5h4pytIo78Neb8yXKxo0jTDEnPutw5fkrR0Y+oIxtHavR
kA7wl+/XJCm4Z8xenltMNDt4cyh/I4DLZB0QooAbL0YZMxTx/A0mAp/lqbwUpZ7W
1GOApyllOXMp8GT7M+DuG4xFvcJDAidhGnCa4umIcVTT5pqwjDkcLVDIFuWyYKX6
H2i+hPbi+jcnJ7snQEgiqJtPotxIJk7VwDYYnUowVd/xFHKk6ByRuIa/ptbVFw==
-----END CERTIFICATE-----
)KEY";
```

Figure 18: Altered code for the config.h file

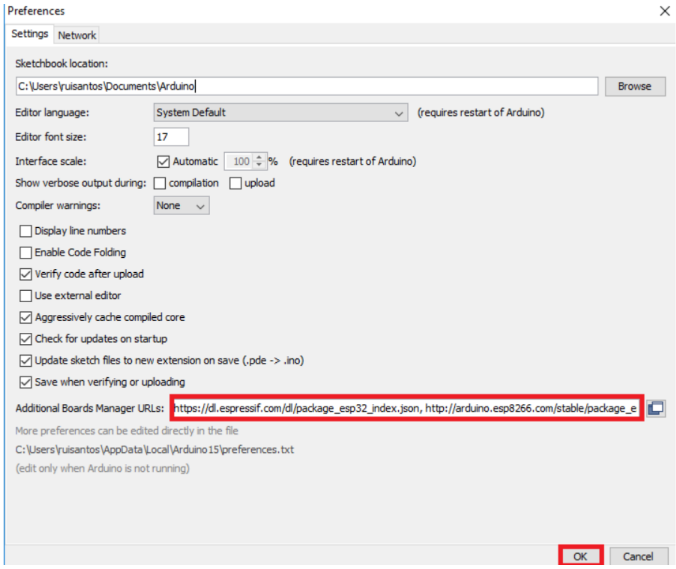


```
#include "config.h"

#include <WiFiClientSecure.h>
#include <MQTTClient.h> //MQTT Library Source: https://github.com/256dpi/arduino-mqtt

#include <ArduinoJson.h> //ArduinoJson Library Source: https://github.com/bblanchon/ArduinoJson
#include "WiFi.h"

// MQTT topics for the device
#define AWS_IOT_PUBLISH_TOPIC   "esp32/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp32/sub"

WiFiClientSecure wifi_client = WiFiClientSecure();
MQTTClient mqtt_client = MQTTClient(256); //256 indicates the maximum size for packets being published and received.

uint32_t t1;

void connectAWS()
{
  //Begin WiFi in station mode
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  //Wait for WiFi connection
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  // Configure wifi_client with the correct certificates and keys
  wifi_client.setCACert(AWS_CERT_CA);
  wifi_client.setCertificate(AWS_CERT_CRT);
  wifi_client.setPrivateKey(AWS_CERT_PRIVATE);

  //Connect to AWS IOT Broker. 8883 is the port used for MQTT
  mqtt_client.begin(AWS_IOT_ENDPOINT, 8883, wifi_client);

  //Set action to be taken on incoming messages
  mqtt_client.onMessage(incomingMessageHandler);

  Serial.print("Connecting to AWS IOT");

  //Wait for connection to AWS IoT
  while (!mqtt_client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  Serial.println();
```

Figure 19: Code for the AWS_IOT file

7. Installing the ESP32 Arduino IDE

The first step to connecting the ESP32 IDE, is to go to **Files->Preferences.**
In the 'Additional Board URL's' field, copy and paste the following URL and then press OK.



Preferences

Settings | Network

Sketchbook location:
C:\Users\ruisantos\Documents\Arduino
Browse

Editor language: System Default (requires restart of Arduino)
Editor font size: 17
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Show verbose output during: □ compilation □ upload
Compiler warnings: None

□ Display line numbers
□ Enable Code Folding
☑ Verify code after upload
□ Use external editor
☑ Aggressively cache compiled core
☑ Check for updates on startup
☑ Update sketch files to new extension on save (.pde -> .ino)
☑ Save when verifying or uploading

Additional Boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_e

More preferences can be edited directly in the file
C:\Users\ruisantos\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

OK Cancel

Figure 18: ESP32 Arduino URL

Open the Boards Manager. Go to **Tools > Board > Boards Manager**…

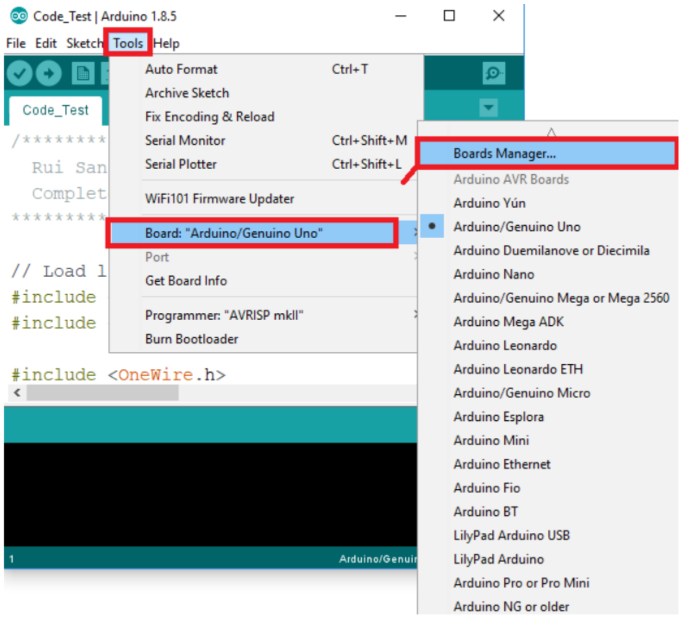
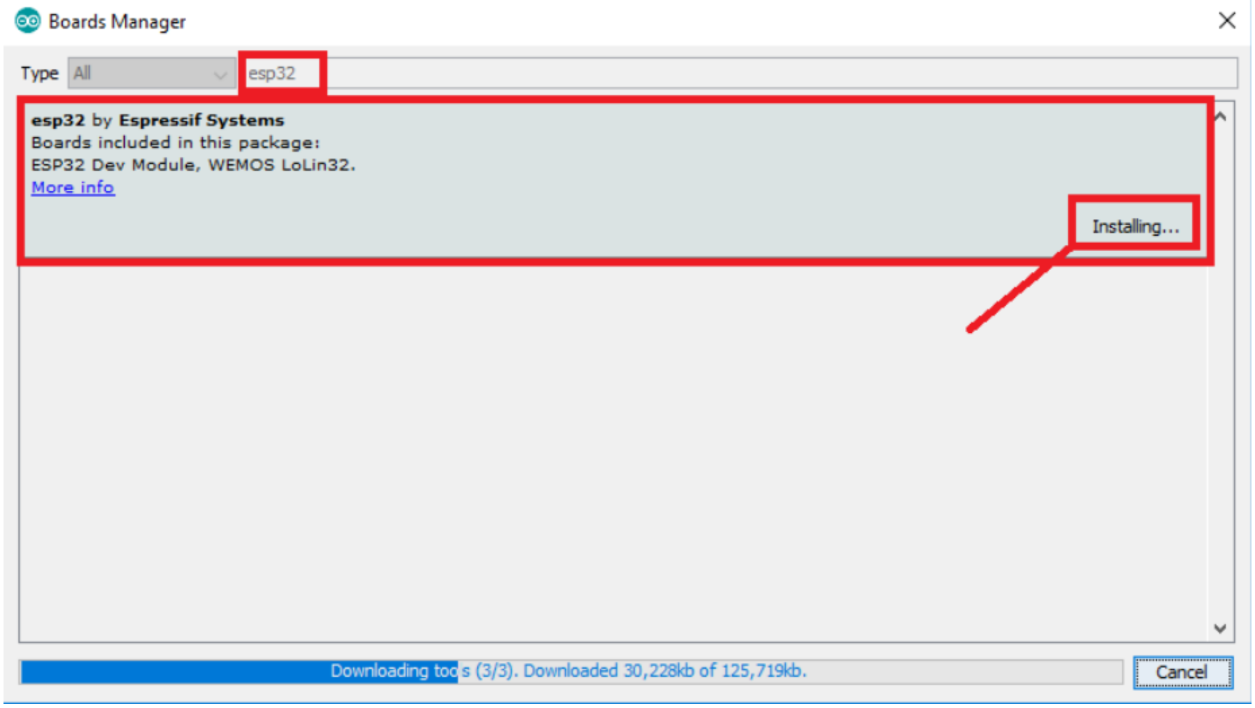


Figure 19: Board Management

Search for **ESP32** and press install button for the "**ESP32 by Espressif Systems**"



Boards Manager

Type All esp32

esp32 by **Espressif Systems**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

Installing...

Downloading tools (3/3). Downloaded 30,228kb of 125,719kb.

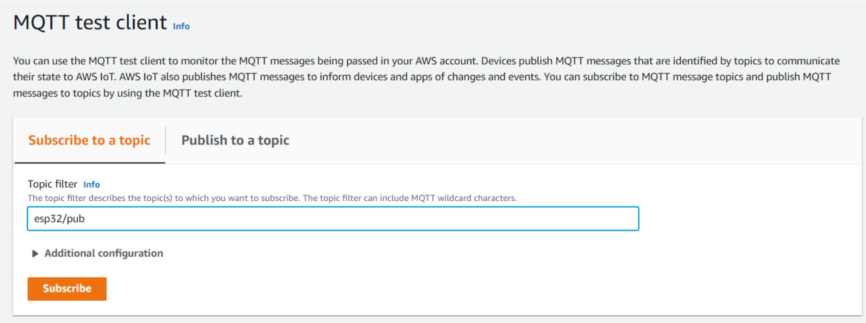
Cancel

Figure 20: Installing the ESP32 board library

Select your Board in Tools > Board menu (in my case it's the **DOIT ESP32 DEVKIT V1**) Make a port selection in the **Tools** option. Ensure that 'Port 3' is selected.

7.Testing

In order to test, go to the AWS IoT Console, and from the left side menu, select '*Test*'. The 'MQTT test client' portal will open up. Think of this like the broker interface. You can subscribe to a topic here, and also publish to a topic. Since our ESP32 is publishing to the 'esp32/pub' topic, let us subscribe to that topic and see the incoming messages.



MQTT test client Info

You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devices publish MQTT messages that are identified by topics to communicate their state to AWS IoT. AWS IoT also publishes MQTT messages to inform devices and apps of changes and events. You can subscribe to MQTT message topics and publish MQTT messages to topics by using the MQTT test client.

Subscribe to a topic | Publish to a topic

Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
esp32/pub

▶ Additional configuration

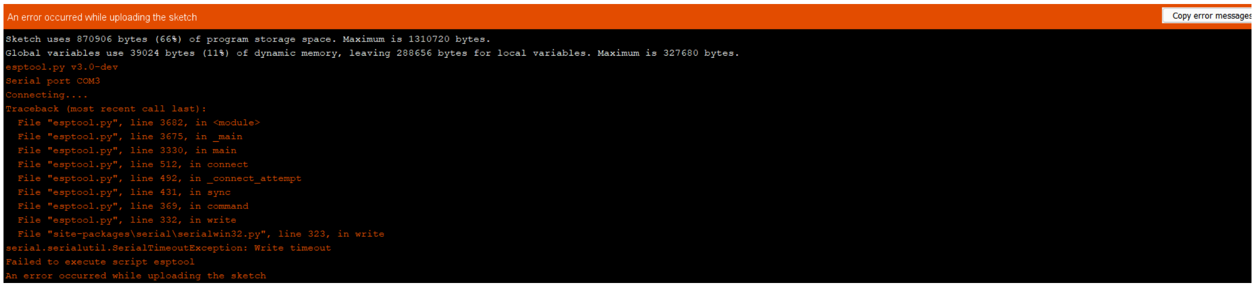
Subscribe

Figure 21: MQTT test

Once you have successfully connected the ESP32 and verified that the code is working, it is now time to run the program.

Conclusion

Unfortunately, the ESP32 was unable to connect to the ports of the computer, as there were no drivers that are compatible with the ESP32 devKit model. This may be due to the more advanced computer models which come with the Windows 11 or later software. In order to run the program and send the data, make use of a Windows 10 or less recent software.



```
Sketch uses 870906 bytes (66%) of program storage space. Maximum is 1310720 bytes.
Global variables use 39024 bytes (11%) of dynamic memory, leaving 288656 bytes for local variables. Maximum is 327680 bytes.
esptool.py v3.0-dev
Serial port COM3
Connecting....
Traceback (most recent call last):
  File "esptool.py", line 3682, in <module>
  File "esptool.py", line 3675, in _main
  File "esptool.py", line 3330, in main
  File "esptool.py", line 512, in connect
  File "esptool.py", line 492, in _connect_attempt
  File "esptool.py", line 431, in sync
  File "esptool.py", line 369, in command
  File "esptool.py", line 332, in write
  File "site-packages\serial\serialwin32.py", line 323, in write
serial.serialutil.SerialTimeoutException: Write timeout
Failed to execute script esptool
An error occurred while uploading the sketch
```

Figure22: Runtime error due to port drivers