



Search ...



Connect ESP32 MicroPython to AWS IoT

Lesson Contents

1. AWS IoT

- 1.1. Things

- 1.2. Certificates

- 1.3. Policy

2. ESP32

- 2.1. Drivers

- 2.2. MicroPython Firmware

- 2.3. Python

- 2.4. Wireless

- 2.5. File Management

- 2.6. Hardware Button

- 2.7. Install Certificate

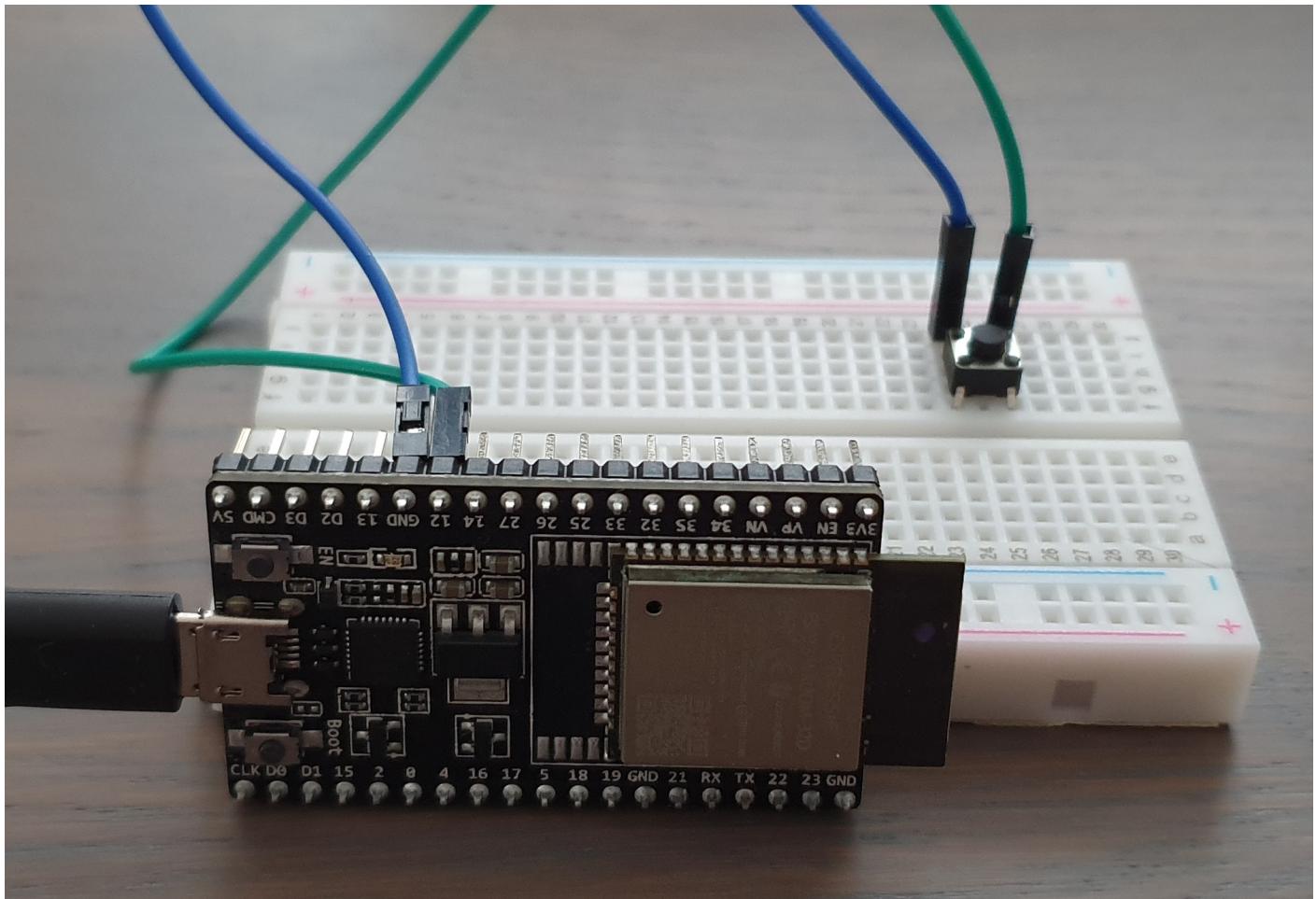
- 2.8. MQTT

- 2.9. Act

3. Conclusion

Most IoT tutorials use something like a Raspberry Pi as the “thing”. The Raspberry Pi is a great device but when it comes to IoT, it feels too much like a computer to me. It has GPIO pins so you can connect different hardware but it runs a full Linux distribution (Raspbian, based on Debian).

I wanted to try a “real” IoT device and decided on an [ESP32 from Espressif](#). The ESP32 is a small device, which has a Wi-Fi & Bluetooth Chip, and plenty of GPIO pins. It supports a range of firmwares including Mongoose OS, Zerynth, ESP Easy, FreeRTOS, and MicroPython.



To make it a true IoT device, I added a hardware button. We are going to connect this ESP32 to AWS IoT and configure the ESP32 so it sends a message to AWS IoT when we press the button.

This lesson is lengthy, especially with all the AWS IoT screenshots. If you get lost, you can use the index below:

1. AWS IoT

Let's start with AWS. There are three components we need:

- Thing
- Certificate
- Policy

The thing is our ESP32 and it requires a certificate for authentication. The policy defines what our thing can do.

1.1. Things



Services ▾

Resource Groups ▾



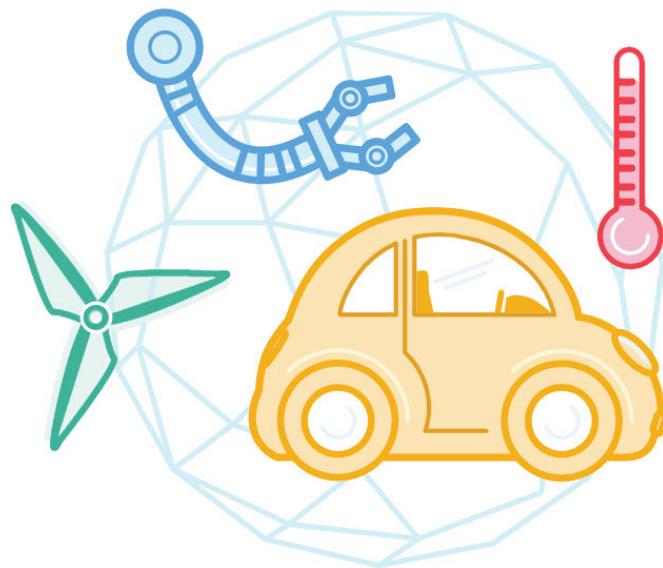
History

[Console Home](#)[IoT Events](#)[IoT 1-Click](#)[IoT Things Graph](#)[IoT Core](#)[CloudWatch](#)**Internet Of Things****IoT Core**[Amazon FreeRTOS](#)[IoT 1-Click](#)[IoT Analytics](#)[IoT Device Defender](#)[IoT Device Management](#)[IoT Events](#)[IoT Greengrass](#)[IoT SiteWise](#)[IoT Things Graph](#)

Under **Manage**, choose **Things**:

The screenshot shows the AWS IoT Things management interface. On the left, there's a sidebar with a yellow circular icon at the top. Below it, the 'AWS IoT' logo is displayed. The sidebar contains several navigation items: 'Monitor', 'Onboard', 'Manage' (which is highlighted with a red box), 'Things' (also highlighted with a red box), 'Types', 'Thing Groups', 'Billing Groups', and 'Jobs'. To the right of the sidebar, the main content area has a title 'Things' and a search bar labeled 'Search things' with a magnifying glass icon. A vertical grey scrollbar is positioned between the sidebar and the main content area.

You'll see the following screen because you don't have any things yet:



You don't have any things yet

A thing is the representation of a device in the cloud.

[Learn more](#)[Register a thing](#)

Select **register a thing** and choose **create a single thing**:

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

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.

Create a single thing

Create many things

Create a single thing

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

We'll give our thing a name. I'll call it "esp32":

CREATE A THING

Add your device to the thing registry

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

Name
esp32

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 [Create a type](#)

Add this thing to a group
Adding your thing to a group allows you to manage devices remotely using jobs.

Thing Group
Groups / [Create group](#) [Change](#)

Set searchable thing attributes (optional)
Enter a value for one or more of these attributes so that you can search for your things in the registry.

Attribute key Provide an attribute key, e.g. Manufacturer	Value Provide an attribute value, e.g. Acme-Corporation	Clear
Add another		

Show thing shadow

Cancel **Back** **Next**

Lessons ↗

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



1.2. Certificates

The console asks us if we want to create a certificate. We need one so select the **Create certificate** option:

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.

Create certificate

Create with CSR

Get started

Create thing without certificate

In the following screen you will see the certificates that were created:

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	4b7db071de.cert.pem	Download
A public key	4b7db071de.public.key	Download
A private key	4b7db071de.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT [Download](#)

[Activate](#)

[Cancel](#) [Done](#) [Attach a policy](#)

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Save the certificate, public key, and private key. We'll need them later. We'll also need the root certificates so click on the download button next to *A root CA for AWS IoT*. This takes you to the X.509 Certificate and AWS IoT page:

Server Authentication

Server certificates allow your devices to verify that they're communicating with AWS IoT and not another server impersonating AWS IoT. Service certificates must be copied onto your device and referenced when devices connect to AWS IoT. For more information, see the [AWS IoT Device SDKs](#).

AWS IoT server certificates are signed by one of the following 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.

Download all the certificates you see here. We'll need them later.

1.3. Policy





Monitor

Onboard

Manage

Greengrass

Secure

Certificates

Policies

CAs

Role Aliases

Lessons ⓘ



You don't have any policies yet

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

[Learn more](#)

[Create a policy](#)

Our policy requires two actions:

- iot:Connect
- iot:Publish

The iot:Connect action grants permission to connect to AWS IoT with client id "esp32" and the iot:Publish action restricts the device to publishing on a topic named "esp32".

Here's what our actions look like:

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
esp32-policy

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-1:<account-id>:client/esp32

Effect
 Allow Deny

Action
iot:Publish

Resource ARN
arn:aws:iot:us-east-1:<account-id>:topic/esp32

Effect
 Allow Deny

Add statement

Create

Hit the **Create** button and head back to **Secure > Certificates**:

IMPORTANT

With the new AWS, go to Policy examples and select the second to last Publish/Subscribe policy (the one that says Exclude topic from allowed topics or Excluir tema de temas permitidos)

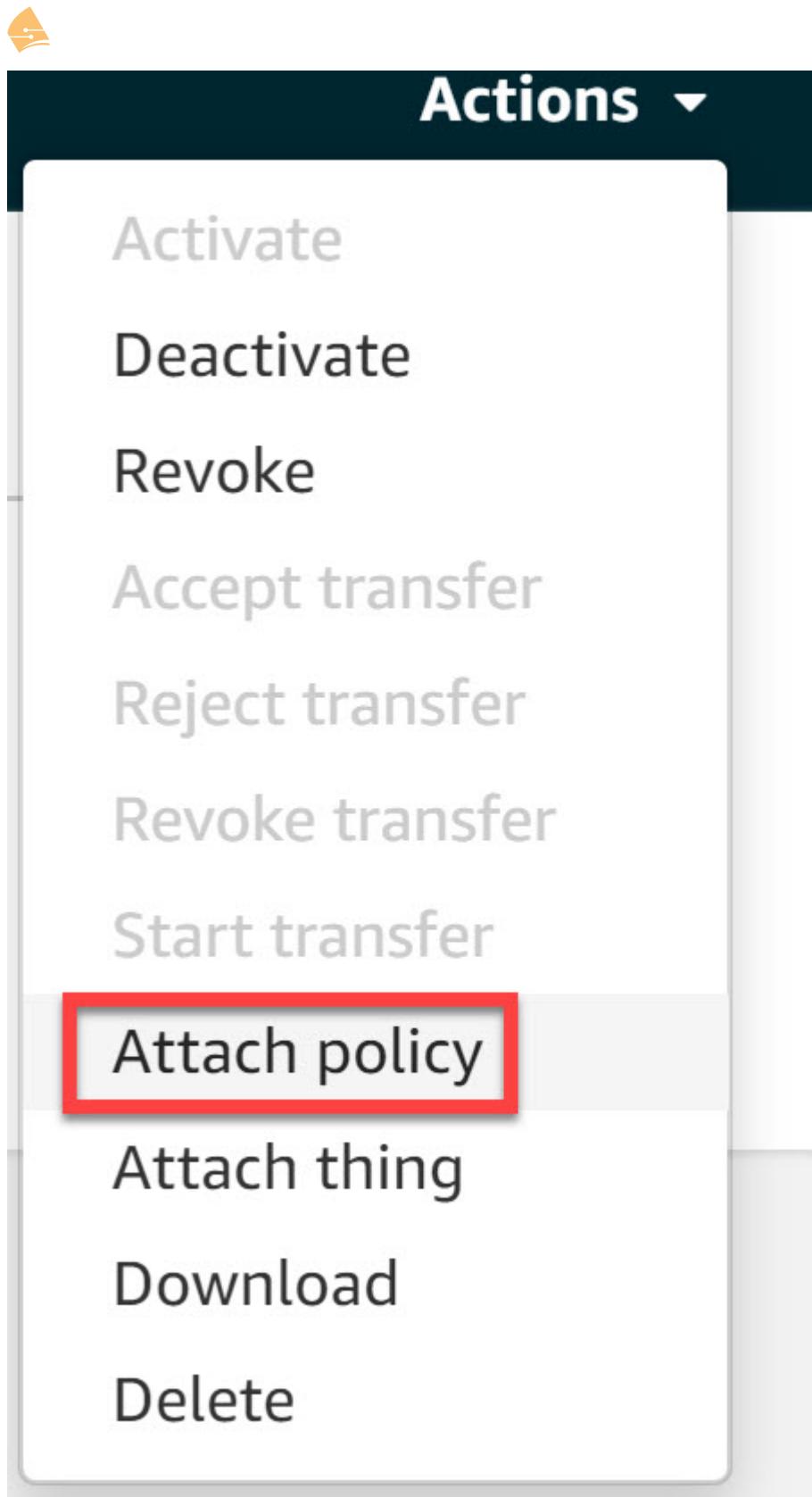
Then, simply replace \${iot:Connection.Thing.ThingName} for the name of the thing (i.e. esp32 in this example)

The screenshot shows the AWS IoT Certificates page. On the left, there's a sidebar with navigation links: Monitor, Onboard, Manage, Greengrass, Secure (which is selected), Certificates, Policies, CAs, Role Aliases, and Authorizers. The main area displays a search bar and a list of certificates. One certificate, with the ID '4b7db071de5178246...' and status 'ACTIVE', is highlighted with a red box.

In the upper right corner, select **Actions**:

The screenshot shows the AWS IoT Certificate details page for the certificate '4b7db071de5178246f64131be80400ce92f4251929373c0f7681cbaac2954c15'. The 'Actions' button in the top right corner is highlighted with a red box. The page displays the Certificate ARN (arn:aws:iot:us-east-1:151709868703:cert/4b7db071de5178246f64131be80400ce92f4251929373c0f7681cbaac2954c15) and various certificate details like Issuer, Subject, Create date, Effective date, and Expiration date.

Now choose the **Attach policy** option:



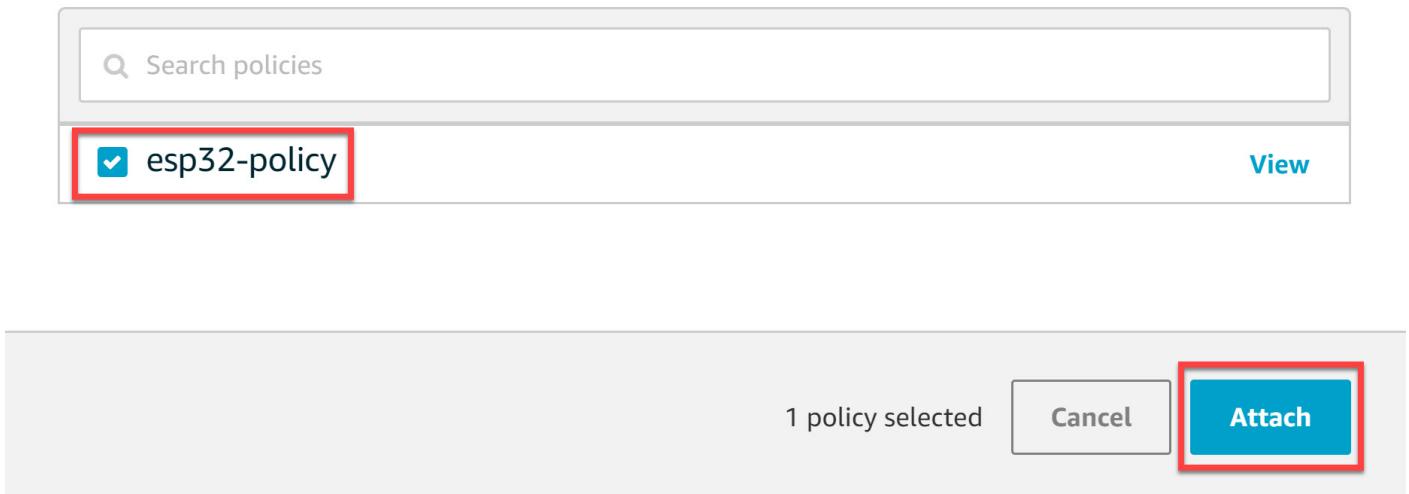
Select the "esp32-policy" we created and click on the **Attach** button:



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

4b7db071de5178246f64131be80400ce92f4251929373c0f7681cbaac2954c15

Choose one or more policies



There is one more thing to do. We need the REST API Endpoint. Go to **Manage > Things**:

A screenshot of the AWS IoT Things management page. The left sidebar shows navigation options like Monitor, Onboard, Manage (with Things selected), Types, Thing Groups, Billing Groups, Jobs, Greengrass, Secure, Defend, Act, Test, Software, Settings, and Learn. The main area is titled "Things" and shows a list with one item: "esp32 NO TYPE". A red box highlights the "esp32" entry. The top navigation bar includes links for Services, Resource Groups, and Support, along with user information: renemolenaar @ 1517-0986-8... and N. Virginia. The bottom footer includes Feedback, English (US), Privacy Policy, Terms of Use, and a Lessons link.

Look for the Rest API Endpoint under **Interact**:

THING
esp32
NO TYPE

Actions ▾

Details This thing already appears to be connected. Connect a device

Security

Thing Groups

Billing Groups

Shadow

Interact

Activity

Jobs

Violations

Defender metrics

HTTPS

Update your Thing Shadow using this Rest API Endpoint. [Learn more](#)

a3gp6dog57u3bg-ats.iot.us-east-1.amazonaws.com

MQTT

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

[Learn more](#)

Update to this thing shadow

\$aws/things/esp32/shadow/update

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Write down the REST API Endpoint. In my case it's:

a3gp6dog57u3bg-ats.iot.us-east-1.amazonaws.com

IMPORTANT

For the new AWS, go to **MQTT test client** or **Cliente de prueba de MQTT**, then click on **Connection details** and **Connection Point**. That link is the Rest API Endpoint.

2. ESP32

Time to get our hands dirty with the ESP32.

2.1. Drivers

I'm using Windows 10. The first time you connect the ESP32 with the USB cable, your computer won't recognize the device. Go to Device Manager and you will see an unknown device:

The screenshot shows the Windows Device Manager interface. At the top, there's a navigation bar with File, Action, View, Help, and several icons for back, forward, search, and help. Below the navigation bar, the main pane displays a tree view of system components under the heading "XPS15". The components listed are: Audio inputs and outputs, Batteries, Biometric devices, Bluetooth, Cameras, Computer, Disk drives, Display adapters, Firmware, Human Interface Devices, Keyboards, Memory technology devices, Mice and other pointing devices, Monitors, Network adapters, Other devices, Print queues, Processors, and Security devices. The "Other devices" node is expanded, and its child node, "CP2102N USB to UART Bridge Controller", is highlighted with a red rectangular border. A vertical scroll bar is visible on the right side of the main pane.

- > Audio inputs and outputs
- > Batteries
- > Biometric devices
- > Bluetooth
- > Cameras
- > Computer
- > Disk drives
- > Display adapters
- > Firmware
- > Human Interface Devices
- > Keyboards
- > Memory technology devices
- > Mice and other pointing devices
- > Monitors
- > Network adapters
- > Other devices
 - CP2102N USB to UART Bridge Controller**
- > Print queues
- > Processors
- > Security devices



CP2102N USB to UART Bridge Controller Properties

X

General

Driver

Details

Events



CP2102N USB to UART Bridge Controller

Device type: Other devices

Manufacturer: Unknown

Location: Port #0001.Hub #0001

Device status

The drivers for this device are not installed. (Code 28)

There are no compatible drivers for this device.

To find a driver for this device, click Update Driver.

Update Driver...

OK

Cancel

Lessons ↺

Choose the first option to automatically search for the driver:



← Update Drivers - CP2102N USB to UART Bridge Controller

How do you want to search for drivers?

→ **Search automatically for updated driver software**

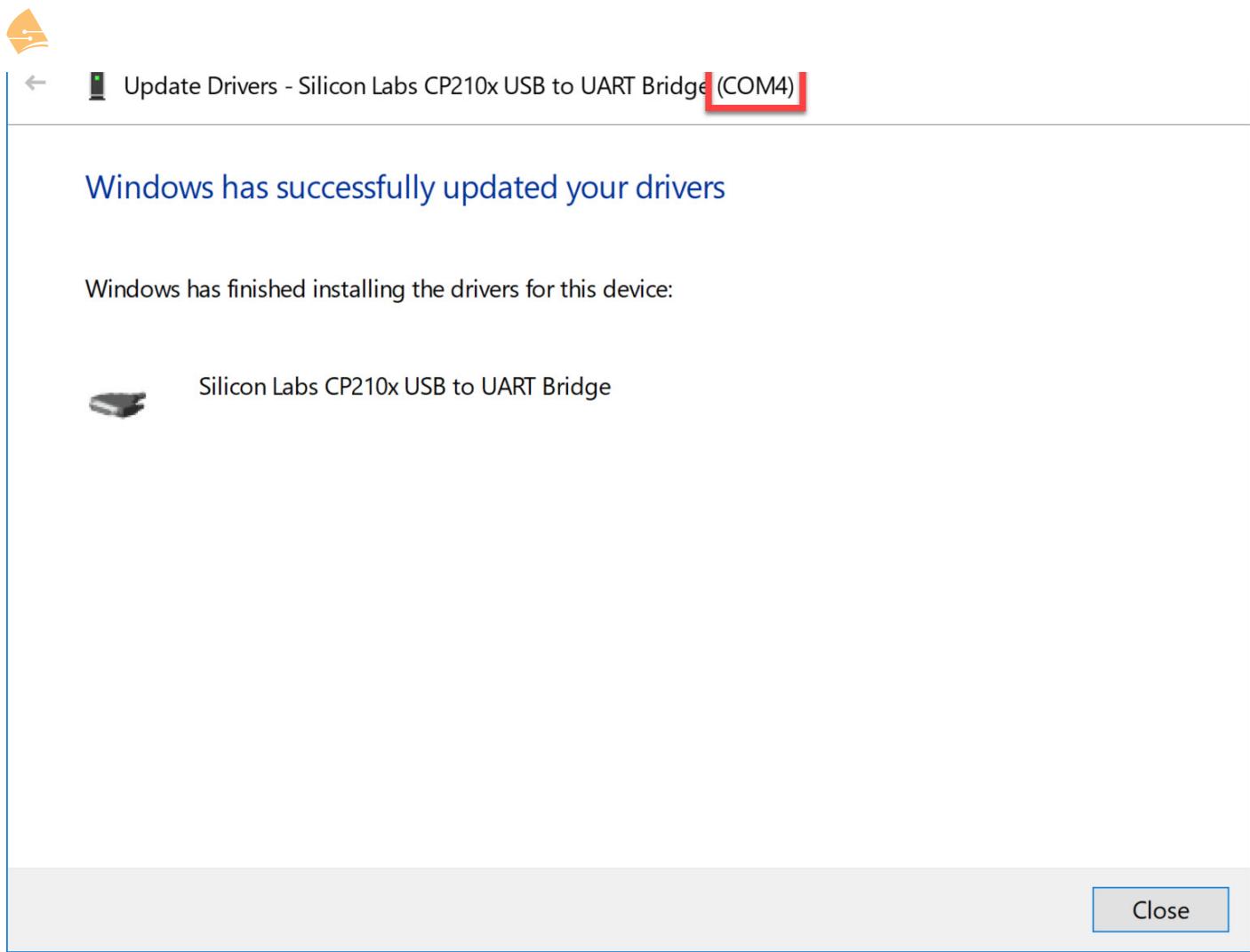
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

→ **Browse my computer for driver software**

Locate and install driver software manually.

Cancel

This won't take long. Once the driver is downloaded it will also show you the COM port number:



In my case, it's COM4.

2.2. MicroPython Firmware

The ESP32 supports different firmwares. My personal favorite is MicroPython. I use Python for many things so it's great that I can use it for the ESP32 as well.

MicroPython includes a small subset of the Python 3 standard library and is optimized to run on microcontrollers.

I downloaded the latest standard firmware (esp32-20190611-v1.11-44-g8b18cfede.bin) and saved it to my disk. To install the firmware, we need the esptool application. This tool allows us to communicate with the ROM bootloader of the ESP32.

We can install it with pip:

```
pip install esptool
```



```
C:\Users\renemolenaar\AppData\Local\Programs\Python\Python37-32\Scripts
```

First, we need to erase the flash. We do this with the following command:

```
esptool.py.exe --chip esp32 -p com4 erase_flash
esptool.py v2.6
Serial port com4
Connecting.....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 24:0a:c4:c1:0f:50
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 8.7s
Hard resetting via RTS pin...
```

Excellent. We now have an empty flash. Let's install the firmware we just downloaded:

```
esptool.py.exe --chip esp32 -p com4 write_flash -z 0x1000
c:\users\renemolenaar\Downloads\esp32-20190611-v1.11-44-g8b18cfede.bin
esptool.py v2.6
Serial port com4
Connecting.....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 24:0a:c4:c1:0f:50
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 1169696 bytes to 731891...
Wrote 1169696 bytes (731891 compressed) at 0x00001000 in 64.9 seconds (effective 144.2 kbit/s)...
```



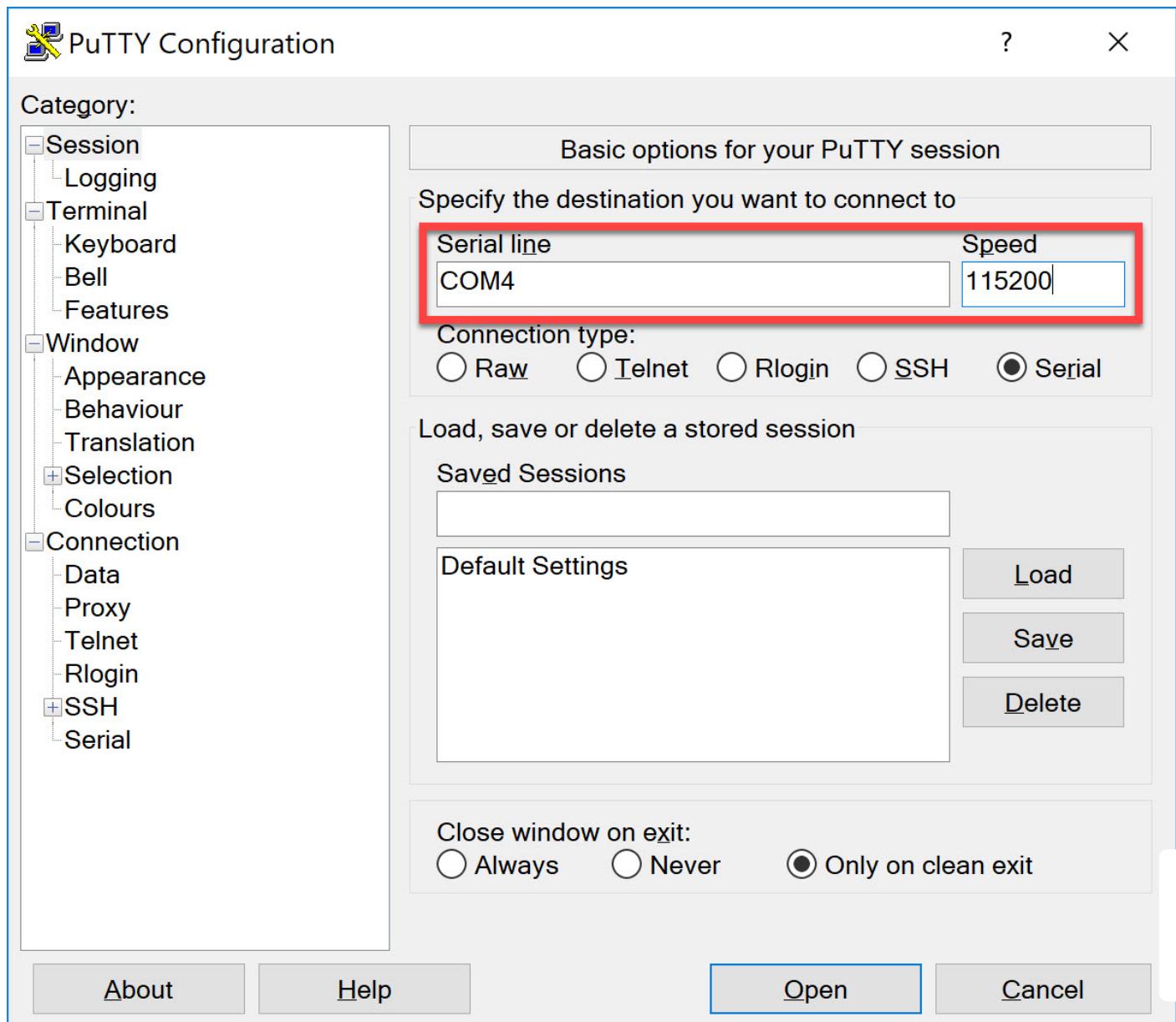
Leaving...

Hard resetting via RTS pin...

Our ESP32 now has the MicroPython firmware.

2.3. Python **NOTE: We can use PyMakr instead (see Micropython tutorial)**

Let's see if we can connect to our ESP32. We can use Putty for this. Enter the COM port number we found in Device Manager and set the speed to **115200**:



Once you open the console, you see the following prompt:



Awesome. We now have access to Python on our ESP32!

2.4. Wireless

Before we can connect our ESP32 “thing” to the Internet, we need connectivity. I’ll use the Wi-Fi chip for this. First we need to activate the Wi-Fi chip:

```
>>> sta_if = network.WLAN(network.STA_IF)
I (268050) wifi: wifi driver task: 3ffe2d70, prio:23, stack:3584, core=0
I (285637) wifi: wifi firmware version: 38e2484
I (285637) wifi: config NVS flash: enabled
I (285637) wifi: config nano formating: disabled
I (285637) system_api: Base MAC address is not set, read default base MAC address from
BLK0 of EFUSE
I (285647) system_api: Base MAC address is not set, read default base MAC address from
BLK0 of EFUSE
I (285677) wifi: Init dynamic tx buffer num: 32
I (285677) wifi: Init data frame dynamic rx buffer num: 32
I (285677) wifi: Init management frame dynamic rx buffer num: 32
I (285687) wifi: Init static rx buffer size: 1600
I (285687) wifi: Init static rx buffer num: 10
I (285697) wifi: Init dynamic rx buffer num: 32
```

You can verify whether the interface is active or not:

```
>>> sta_if.active(True)
W (433087) phy_init: failed to load RF calibration data (0x1102), falling back to full
calibration
I (433237) phy: phy_version: 4007, 9c6b43b, Jan 11 2019, 16:45:07, 0, 2
I (433247) wifi: mode : sta (24:0a:c4:c1:0f:50)
True
I (433247) wifi: STA_START
```

Above, you can see that it shows “True”. Let’s scan for wireless networks:



```
sta_if.scan()
```

```
I (45584) network: event 1
[(b'MY_WIFI_SSID', b'\xfc\xec\xda\x17\xcd', 1, -64, 3, False), (b'Ziggo24978',
b'\x9e\x8a\x0b', 6, -67, 3, False), (b'Ziggo', b'rZ\x9e\x8a\x0b', 6, -67, 5, False),
(b'Leo Gasten', b'z\x8a q\x0b', 1, -75, 0, False), (b'MY_WIFI_SSID',
b'\xfc\xec\xda;\x89w', 11, -80, 3, False), (b'Leo', b'x\x8a q\x0b', 1, -85, 3, False),
(b'ziggo-ap-1909866', b'\xd8\xb6\xb7\xd0)\x00', 11, -88, 3, False), (b'AP_804094029',
b'\xbct\xf9\xf89\xb0', 1, -90, 3, False), (b'Seli', b'\xc0\xc1\xc0.\xac', 1, -92, 4,
False), (b'SMA1992052369', b'\xdc\xef\xca\x97\xb6\x81', 11, -94, 4, False)]
```

My wireless network is called "MY_WIFI_SSID". Let's connect to it:

```
>>> sta_if.connect('MY_WIFI_SSID', 'MY_WIFI_PASSWORD')
>>> I (507797) wifi: new:&lt1,0>, old:&lt1,0>, ap:&lt255,255>, sta:&lt1,0>, prof:1
I (508367) wifi: state: init -> auth (b0)
I (508367) wifi: state: auth -> assoc (0)
I (508377) wifi: state: assoc -> run (10)
I (508387) wifi: connected with MY_WIFI_SSID, channel 1, bssid = fc:ec:da:17:58:cd
I (508387) wifi: pm start, type: 1

I (508387) network: CONNECTED

I (512267) event: sta ip: 10.82.100.135, mask: 255.255.255.0, gw: 10.82.100.254
I (512267) network: GOT_IP
```

We are now connected to the wireless network. We can verify if the ESP32 is connected with the following command:

```
>>> sta_if.isconnected()
True
```

Our thing is now connected, great!

2.5. File Management

Our ESP32 is connected to the wireless network but every time you disconnect the USB cable, you have to type in all wireless commands again. To prevent this from happening, we can add all Python commands in a Python file and store it on the ESP32.



We can install Ampy with pip:

```
pip install adafruit-ampy
```

Ampy ends up in a sub-directory of our user directory:

```
C:\Users\renemolenaar\AppData\Local\Programs\Python\Python37-32\Scripts
```

Let's check the file system of our ESP32:

```
ampy -p com4 ls  
/boot.py
```

There is only one file called "boot.py". Let's view its contents:

```
ampy -p com4 get boot.py  
# This file is executed on every boot (including wake-boot from deepsleep)  
#import esp  
#esp.osdebug(None)  
#import webrepl  
#webrepl.start()
```

Like the comment above states, the boot.py file is executed on every boot. We can use this to include code that we want to run whenever we start our ESP32.

Let's create a short Python function that connects to our wireless network. Save the code below in a boot.py file:

```
def connect():  
    import network  
    sta_if = network.WLAN(network.STA_IF)  
    if not sta_if.isconnected():  
        print('connecting to network...')
```



```
sta_if.active(True)
sta_if.connect('MY_WIFI_SSID', 'MY_WIFI_PASSWORD')
while not sta_if.isconnected():
    pass
print('network config:', sta_if.ifconfig())
```

Now we copy our boot.py file to the ESP32:

```
ampy -p com4 put c:\Users\renemolenaar\Downloads\boot.py
```

Restart your ESP32, open the CLI and run the connect() function:

```
>>> connect()
I (18115) wifi: wifi driver task: 3ffe2eb8, prio:23, stack:3584, core=0
I (18115) wifi: wifi firmware version: 38e2484
I (18115) wifi: config NVS flash: enabled
I (18115) wifi: config nano formating: disabled
I (18115) system_api: Base MAC address is not set, read default base MAC address from
BLK0 of EFUSE
I (18125) system_api: Base MAC address is not set, read default base MAC address from
BLK0 of EFUSE
I (18155) wifi: Init dynamic tx buffer num: 32
I (18155) wifi: Init data frame dynamic rx buffer num: 32
I (18155) wifi: Init management frame dynamic rx buffer num: 32
I (18155) wifi: Init static rx buffer size: 1600
I (18165) wifi: Init static rx buffer num: 10
I (18165) wifi: Init dynamic rx buffer num: 32
connecting to network...
I (18225) phy: phy_version: 4007, 9c6b43b, Jan 11 2019, 16:45:07, 0, 0
I (18235) wifi: mode : sta (24:0a:c4:c1:0f:50)
I (18235) wifi: STA_START
I (18355) wifi: new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1
I (18925) wifi: state: init -> auth (b0)
I (18925) wifi: state: auth -> assoc (0)
I (18935) wifi: state: assoc -> run (10)
I (18945) wifi: connected with MY_WIFI_SSID, channel 1, bssid = fc:ec:da:17:58:cd
I (18955) wifi: pm start, type: 1
```



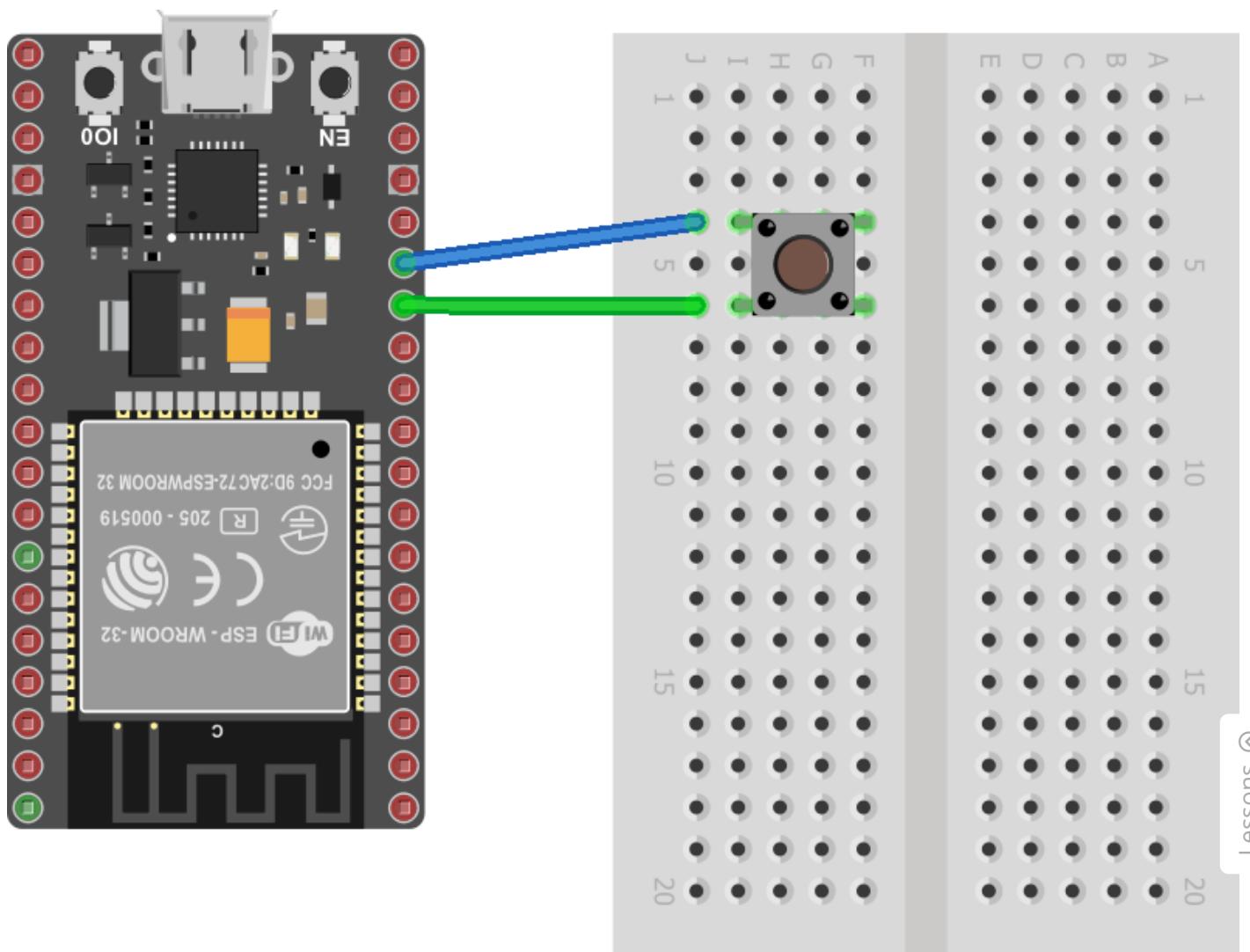
```
I (19555) network: GOT_IP  
network config: ('10.82.100.135', '255.255.255.0', '10.82.100.254', '10.82.100.253')
```

Excellent. With a single command we can now connect our ESP32 to the wireless network.

- ! You could also add `connect()` to the `boot.py` file so that the ESP32 connects to your wireless network when it boots.

2.6. Hardware Button

I connected a hardware button to my ESP32 through a breadboard. I used GPIO pin 12 and the GND pin next to it. Here's an overview:





```
>>> import machine  
>>> button = machine.Pin(12, machine.Pin.IN, machine.Pin.PULL_UP)
```

When we read the value of the button variable, we get the status:

```
>>> button.value()  
1
```

When the button is unpressed, the value is 1. Let's press it and re-read the value:

```
>>> button.value()  
0
```

Excellent. We can read the status of our hardware button.

2.7. Install Certificate

Now we need to install the certificates we downloaded from AWS IoT. We can do this with Ampy:

```
ampy -p com4 put c:\users\renemolenaar\Downloads\4b7db071de-certificate.pem.crt  
ampy -p com4 put c:\users\renemolenaar\Downloads\4b7db071de-private.pem.key  
ampy -p com4 put c:\users\renemolenaar\Downloads\4b7db071de-public.pem.key  
ampy -p com4 put c:\users\renemolenaar\Downloads\AmazonRootCA1.pem  
ampy -p com4 put c:\users\renemolenaar\Downloads\AmazonRootCA3.pem
```

Let's check if our files are on the ESP32 file system:

```
ampy -p com4 ls  
/4b7db071de-certificate.pem.crt  
/4b7db071de-private.pem.key  
/4b7db071de-public.pem.key  
/AmazonRootCA1.pem  
/AmazonRootCA3.pem  
/boot.py
```



2.8. MQTT

We now have everything we need to connect our ESP32 to AWS IoT. We'll use MQTT for this. First, we import the MQTT library:

```
>>> from umqtt.robust import MQTTClient  
I (197871) modsocket: Initializing
```

MQTT requires a number of settings:

- certificate file: the certificate we created in AWS IoT for our thing.
- private key: the private key from the certificate file.
- MQTT client ID: this matches the name of our thing in AWS IoT.
- MQTT port number: default port number
- MQTT topic: the topic we configured in AWS IoT.
- MQTT host: the restpoint API endpoint for our thing.

Let's set all these values and read the certificate files into variables:

```
>>> certificate_file = "/4b7db071de-certificate.pem.crt"  
>>> private_key_file = "/4b7db071de-private.pem.key"  
>>> mqtt_client_id = "esp32"  
>>> mqtt_port = 8883  
>>> mqtt_topic = "esp32/publish"  
>>> mqtt_host = "a3gp6dog57u3bg-ats.iot.us-east-1.amazonaws.com"  
>>> mqtt_client = None  
  
>>> with open(private_key_file, "r") as f:  
...     private_key = f.read()  
...  
>>> with open(certificate_file, "r") as f:  
...     certificate = f.read()
```

We can now configure a MQTT client that uses all the above variables:

```
mqtt_client = MQTTClient(client_id= mqtt_client_id, server= mqtt_host, port= mqtt_port, keepalive= 5000,  
    ssl=True, ssl_params={"cert":certificate, "key":private_key, "server_side":False})  
    keepalive=5000, ssl=True, ssl_params={"cert":certificate, "key":private_key,  
    "server_side":False})
```

Let's try to connect through MQTT:

```
>>> mqtt_client.connect()
```

We want to make sure AWS IoT receives our MQTT messages so head over to AWS IoT and look for the **Test** section:



Monitor

Onboard

Manage

Greengrass

Secure

Defend

Act



The screenshot shows the AWS IoT MQTT client interface. On the left sidebar, under the 'Test' section, there are several options: Monitor, Onboard, Manage, Greengrass, Secure, Defend, Act, Software, Settings, and Learn. The main area is titled 'MQTT client' and shows the 'Subscriptions' tab. A red box highlights the 'Subscribe to a topic' button and the 'Subscription topic' input field which contains 'esp32/publish'. Another red box highlights the 'Subscribe to topic' button. Below this, there are sections for 'Max message capture' (set to 100), 'Quality of Service' (set to 0 - This client will not acknowledge to the Device Gateway that messages are received), and 'MQTT payload display' (set to Auto-format JSON payloads (improves readability)). The 'Publish' section shows a text input field with the following JSON message:

```

1  {
2     "message": "Hello from AWS IoT console"
3  }

```

Let's do a quick test to make sure we receive a message. Try the following line:

```
>>> mqtt_client.publish(mqtt_topic, "test message")
```

This sends the “test message” to AWS IoT and we should see it in the console:

The screenshot shows the AWS IoT MQTT client console. It displays a message entry for the topic 'esp32/publish' at the time 'Jun 12, 2019 9:56:05 AM +0200'. The message content is 'test message'. Above the message, there is a green bar with the text 'We cannot display the message as JSON, and are instead displaying it as UTF-8 String.' To the right of the message entry, there are 'Export' and 'Hide' buttons. On the far right edge of the screen, there is a vertical 'Lessons' sidebar.

Excellent. How about we go one step further? Whenever I push the button, I want to see a message. I can do this with the following code:



```
import time
```

```
>>> while True:  
...     if button.value() == 0:  
...         mqtt_client.publish(mqtt_topic, "button pushed!")  
...         time.sleep(1)
```

Whenever I push the button, the value is 0. When that happens, I want to send the “button pushed!” message. The `time.sleep(1)` part is a quick fix to add a one-second delay. If I don’t add this, we will spam AWS IoT non-stop with MQTT messages.

I can see my messages in the console:

esp32/publish	Jun 12, 2019 9:59:42 AM +0200	Export	Hide
We cannot display the message as JSON, and are instead displaying it as UTF-8 String.			
<code>button pushed!</code>			
esp32/publish	Jun 12, 2019 9:59:41 AM +0200	Export	Hide
We cannot display the message as JSON, and are instead displaying it as UTF-8 String.			
<code>button pushed!</code>			
esp32/publish	Jun 12, 2019 9:59:37 AM +0200	Export	Hide
We cannot display the message as JSON, and are instead displaying it as UTF-8 String.			
<code>button pushed!</code>			

This is looking great. A press of the button on our thing and the data ends up in AWS IoT.

2.9. Act

Our data is in AWS but it doesn’t do anything yet. With your data in Amazon’s cloud platform, there are so many options to choose from. I’ll give you a quick overview of the possibilities. Look in the **Act** section of AWS IoT:



Monitor

Onboard

Manage

Greengrass

Secure

Defend



Test



Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1
```

And then pick one of the possible actions:

IMPORTANT

Follow this tutorial to upload the files to an AWS S3 bucket:

https://www.youtube.com/watch?v=cwp7RyfOlsI&ab_channel=CodeonCloud



Select an action

Select an action.

<input type="radio"/>	 Insert a message into a DynamoDB table DYNAMODB
<input type="radio"/>	 Split message into multiple columns of a DynamoDB table (DynamoDBv2) DYNAMODBV2
<input type="radio"/>	 Send a message to a Lambda function LAMBDA
<input type="radio"/>	 Send a message as an SNS push notification SNS
<input type="radio"/>	 Send a message to an SQS queue SQS
<input type="radio"/>	 Send a message to an Amazon Kinesis Stream AMAZON KINESIS
<input type="radio"/>	 Republish a message to an AWS IoT topic AWS IOT REPUBLISH
<input type="radio"/>	 Store a message in an Amazon S3 bucket S3
<input type="radio"/>	 Send a message to an Amazon Kinesis Firehose stream AMAZON KINESIS FIREHOSE
<input type="radio"/>	 Send message data to CloudWatch CLOUDWATCH METRICS
<input type="radio"/>	 Change the state of a CloudWatch alarm CLOUDWATCH ALARMS
<input type="radio"/>	 Send a message to the Amazon Elasticsearch Service AMAZON ELASTICSEARCH
<input type="radio"/>	 Send a message to a Salesforce IoT Input Stream SALESFORCE IOT
<input type="radio"/>	 Send a message to IoT Analytics IOT ANALYTICS
<input type="radio"/>	 Send a message to an IoT Events Input IOT EVENTS
<input type="radio"/>	 Start a Step Functions state machine execution STEP FUNCTIONS

[Cancel](#) [Configure action](#)

 [Feedback](#)  [English \(US\)](#) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

For example, you could:

Lessons 

- Store your data in DynamoDB.
- Send a text message or e-mail through SNS.
- Run a serverless application through AWS Lambda.

3. Conclusion



- How to upload the MicroPython firmware to your ESP32.
- How to connect your ESP32 to a wireless network.
- How to create a “thing” in Amazon AWS IoT with certificates and a policy.
- How to send data from your ESP32 to AWS IoT.

I hope you enjoyed this lesson. If you have any questions feel free to leave a comment!

Previous Lesson

IoT Fog and Edge Computing

Next Lesson

Evolving Technologies Practice

Exam



Tags: Cloud, IoT

Forum Replies

A

aleemiiui

From where we can get the device to practice and whats the cost of device , currently I am in Saudia Arabia



lagapides

Hello Muhammad

If you do a quick search online, you will see that the device is available on many sites including eBay. I'm not sure what the shipping capabilities are for each of these retailers for Saudi Arabia, but you will have to look at the policies of each provider.

I hope this has been helpful!

Laz

Lessons



ReneMolenaar

Hello Vic,



```
/4b7db071de-certificate.pem.crt  
/4b7db071de-private.pem.key
```

You could use the IoT root CA certificates to verify that you are communicating with the actual AWS IoT servers, and some server impersonating them. In this example, I didn't.

Rene



ReneMolenaar

Hi Christian,

Did you also try the one I used?

(esp32-20190611-v1.11-44-g8b18cfede.bin)

Just to make sure it's a software issue, not a hardware issue.

It is possible to convert certificate types. You could try to convert the AWS IOT X.509 certificates to DER:

```
openssl x509 -in input.crt -out input.der -outform DER  
openssl x509 -in input.der -inform DER -out output.pem -outform PEM
```

You can convert any certificate with openssl but if you are trying this in a lab and want a more convenient solution, there are [online GUI converters](#).

Rene



lagapides

Hello Imran

Thanks for sharing that! It's so important for the community to share information that will make it easier for others to implement the lessons.

Thanks again!

Laz



8 more replies! Ask a question or join the discussion by visiting our Community Forum

[Disclaimer](#) [Privacy Policy](#) [Support](#) [About](#)

© 2013 - 2023 NetworkLessons.com 51974