Assignment - IOT Experiments

Bishwajit Kumar Poddar

Msc MI

Roll - 211110

1. Configure Raspberry Pi on the system and explore its GUI
2. Get Values for IR, LDR, PIR, UltraSonic, Rain, and Sound sensors using Raspberry.
3. Uploading values of IoT sensors in the Cloud
4. Implement IoT Core service to deploy sensor values on AWS Cloud.
5. Establish wireless communication between 2 Raspberry pi using NRF and Bluetooth modules.
6. Establish Wireless communication between 2 Arduinos using NRF and Bluetooth modules.
7. Establishing client server communication
8. Sniffing messages communicated between client and server
9. Configure ESP8266 using Aurdino IDE and configure GPIOs to Run the Hellp World Program (Using LED)
10. Configure ESP8266 using Aurdino IDE and display characters in the Seven Segment Display.

Configure Raspberry Pi on the system and explore its GUI

- To setup raspberry pi we need to flash official / arm images to raspberry. To do so, we can use the raspberry pi imager which is a cloned tool of etcher. It will download the raspberry pi image and flash it to the microsd card which will be used to boot Raspberry pie.

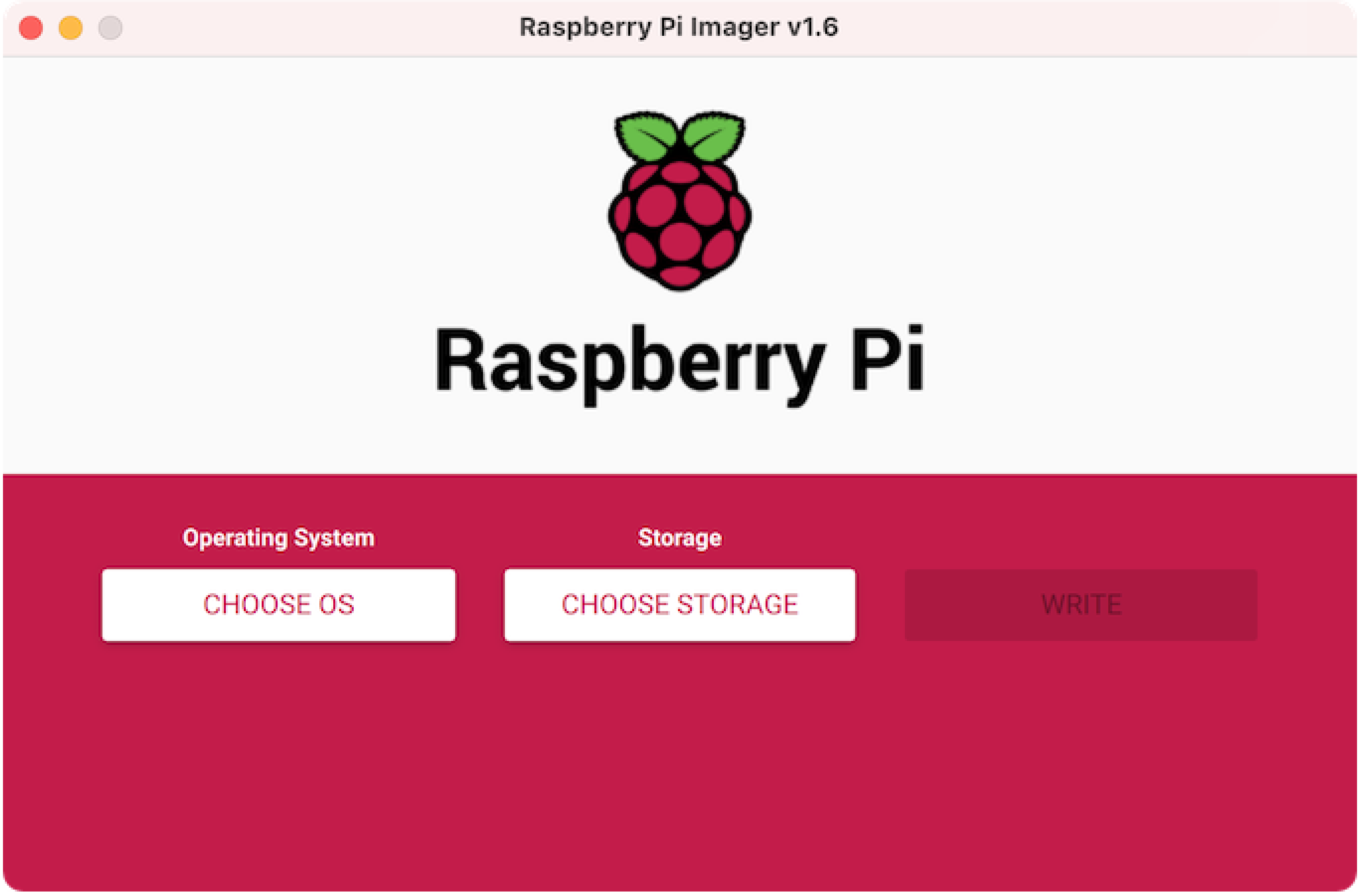


Figure: Raspberry pi imager

- Choose the OS that you want to run on raspberry pi. And choose the storage which is the microsd card which will be inserted. after selecting click on write and image will be written to the micro-sd card. Just plug the micro-sd to raspberry pi and give power and it will boot on will show raspberry OS booting screen.
- Explore the OS. Open terminal and use `apt update` command to update the libraries.

Get Values for IR, LDR, PIR, UltraSonic, Rain, and Sound sensors using Raspberry.

1. IR

- This sensor emits IR beam and in return senses the light which is bouncing back to it.
- If a object is close enough then this sensor can detect the object and set -off alarm. But for this experiment the python code only gives a bash output.

Souce Code

```
import RPi.GPIO as GPIO
import time

sensor = 16
#buzzer = 18
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(sensor,GPIO.IN)
#GPIO.setup(buzzer,GPIO.OUT)

#GPIO.output(buzzer,False)
print("IR Sensor Ready.....")
print(" ")


while True:
if not GPIO.input(sensor):
    #print(GPIO.input(sensor))
    #GPIO.output(buzzer,True)

    print("Object Detected")
    time.sleep(0.5)
else:
    print("Object not Deteceted")
    time.sleep(0.5)
```

2. LDR

- Light sensor detects light and give output as binary if its bright or not.

Source Code

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
LIGHT_PIN = 24
GPIO.setup(LIGHT_PIN, GPIO.IN)
lOld =  not GPIO.input(LIGHT_PIN)
print('Starting up the LIGHT Module (click on STOP to exit)')
time.sleep(0.2)
while True:
    if GPIO.input(LIGHT_PIN) <= 10:
        if GPIO.input(LIGHT_PIN):
        print ('\u263e')
        else:
        print ('\u263c')
    lOld = GPIO.input(LIGHT_PIN)
    time.sleep(0.2)
```

3. PIR

- its senses movement in the environment using IR data.

Source Code

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.IN)         #Read output from PIR motion sensor
GPIO.setup(3, GPIO.OUT)         #LED output pin
while True:
    i=GPIO.input(11)
    if i==0:                 #When output from motion sensor is LOW
        print("No intruders",i)
        GPIO.output(3, 0)  #Turn OFF LED
        time.sleep(0.1)
    elif i==1:               #When output from motion sensor is HIGH
        print("Intruder detected",i)
        GPIO.output(3, 1)  #Turn ON LED
        time.sleep(0.1)
```

4. UltraSonic

- For this experiment we were given ultrasonic sensor and resistors to calculate the distance from sensor to any object.
    - Ultrasonic send a sound signal.
    - It bounces from the object.
    - From the time the sound wave came back we calculate the distance

Source Code

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO_TRIGGER = 18
GPIO_ECHO = 24

GPIO.setup(GPIO_TRIGGER , GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    GPIO.output(GPIO_TRIGGER, True )
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False )

    starttime = time.time()
    stoptime = time.time()

    while GPIO.input(GPIO_ECHO)==0:
        starttime = time.time()

    while GPIO.input(GPIO_ECHO)==1:
        stoptime = time.time()
    timeescaped = stoptime - starttime

    distance = (timeescaped * 34300) /2
    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print(f'Measured distance = {round(dist)}')
            # add the requets code here + imported
            time.sleep(0.2)

    except KeyboardInterrupt:
        print('why you stopped ???????????????????')
        GPIO.cleanup()
```

5. Rain Sensor

- This sensor uses a simple circuit with many iron plates with minimum distance. if a water drops fall on the plate then its connects the circuit and thus gives a positive result. As water with minerals are good conductors of electricity. Water used in this filtered water (regular drinking)

Source Code

```
from time import sleep
from gpiozero import Buzzer, InputDevice

#buzz   = Buzzer(13)
no_rain = InputDevice(18)

def buzz_now(iterations):
    for x in range(iterations):
        #buzz.on()
        sleep(0.1)
        buzz.off()
        sleep(0.1)

while True:
    if not no_rain.is_active:
        print("It's raining - get the washing in!")
        #buzz_now(5)
        # insert your other code or functions here
        # e.g. tweet, SMS, email, take a photo etc.
    sleep(1)
```

i.e. The buzz codes are commented out as we were not given any buzzer. But pin 13 can be used for that as per the code

6. Sound Sensor

- Detects the noise in the environment and eleminates the background noise from there it gives any noise greater than the ambient noise.

Source Code

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

#GPIO SETUP
channel = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel, GPIO.IN)

def callback(channel):
    if GPIO.input(channel):
            print "Sound Detected!"
    else:
            print "Sound Detected!"

GPIO.add_event_detect(channel, GPIO.BOTH, bouncetime=300)  # let us
know when the pin goes HIGH or LOW
GPIO.add_event_callback(channel, callback)  # assign function to GPIO
PIN, Run function on change

# infinite loop
while True:
        time.sleep(1)
```

Uploading values of IoT sensors in the Cloud

- To upload the values we will the basic http model of Request Reponse model Request Response model *Figure basic Request Response Model*
- To achive this we will use a simple python library which comes with preinstalled http request reponse model as a web framework.
- Create simple web url in flask app which can access the post requests made by the IOT devices and give positive reponse.

Python flask webframework code

```
from flask import Flask,request,jsonify,render_template,redirect

app = Flask(__name__)
gloabl_variable = 0

@app.route("/")
def hello_world():
    return redirect('/multidata')
    # return render_template('auto_update.html')

@app.route('/get_multiple_data')
def get_multidata():
    return jsonify(datas[::-1])

@app.route('/multidata')
def multidata():
    return render_template('multiple_ready.html')

@app.route('/set_data',methods=['GET','POST'])
def set_data():
    print(request.args.to_dict())
    global gloabl_variable
    gloabl_variable = request.args.get('distance')
    return 'thank you'

if __name__=='__main__':
    app.run(host='0.0.0.0',debug= True)
```

> Here the template that is renders

basic html page syntax

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Auto update page</title>
</head>
<body>
    <center>
        <h1>Distance</h1>
        <h2 id = 'rondechaka'>0</h2>
    </center>
    <script>

        var element = document.getElementById("rondechaka");
        setInterval(function() {
            fetch('get_data')
                .then(res => res.json())
                .then(data => element.innerHTML=data.distance)
        }, 500);
    </script>
</body>
</html>
```

- Host this in the AWS EC2 server
- use the amazon ec2 server ip address to post the data from IOT using requests module.

Example request module code

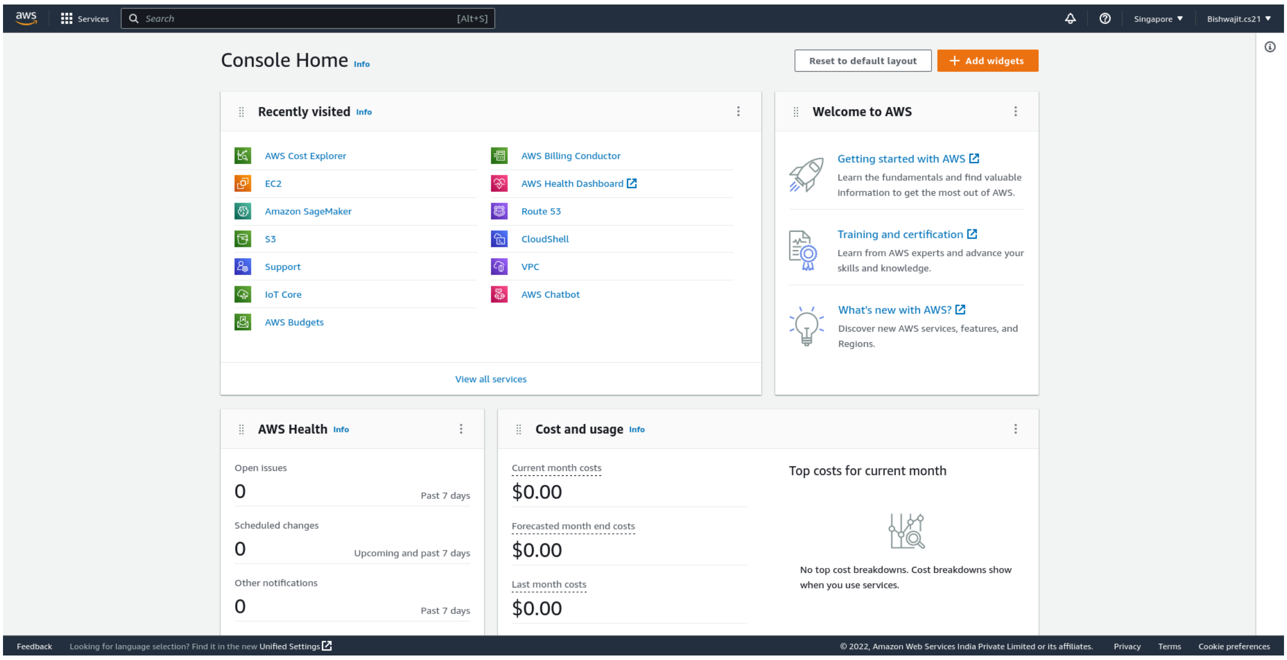
```
import requests
import random
import time
for i in range(100):
    x = requests.get(f'http://10.10.27.169:5000/set_data?distance=
{random.randint(0,99)}')
    print(x)
    time.sleep(1)
```

Implement IoT Core service to deploy sensor values on AWS Cloud

- For this experiment we'll be using MQTT. MQTT is a lightweight, publish-subscribe, machine to machine network protocol for Message queue/Message queuing service. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth.

- AWS cloud give IoT core option and MQTT service with free SSL encryption support.

- First go to `aws.amazon.com`



Learn the fundamentals and find valuable information to get the most out of AWS.

Learn from AWS experts and advance your skills and knowledge.

Discover new AWS services, features, and Regions.

Open issues 0 Past 7 days

Scheduled changes 0 Upcoming and past 7 days

Other notifications 0 Past 7 days

Current month costs $0.00

Forecasted month end costs $0.00

Last month costs $0.00

Top costs for current month

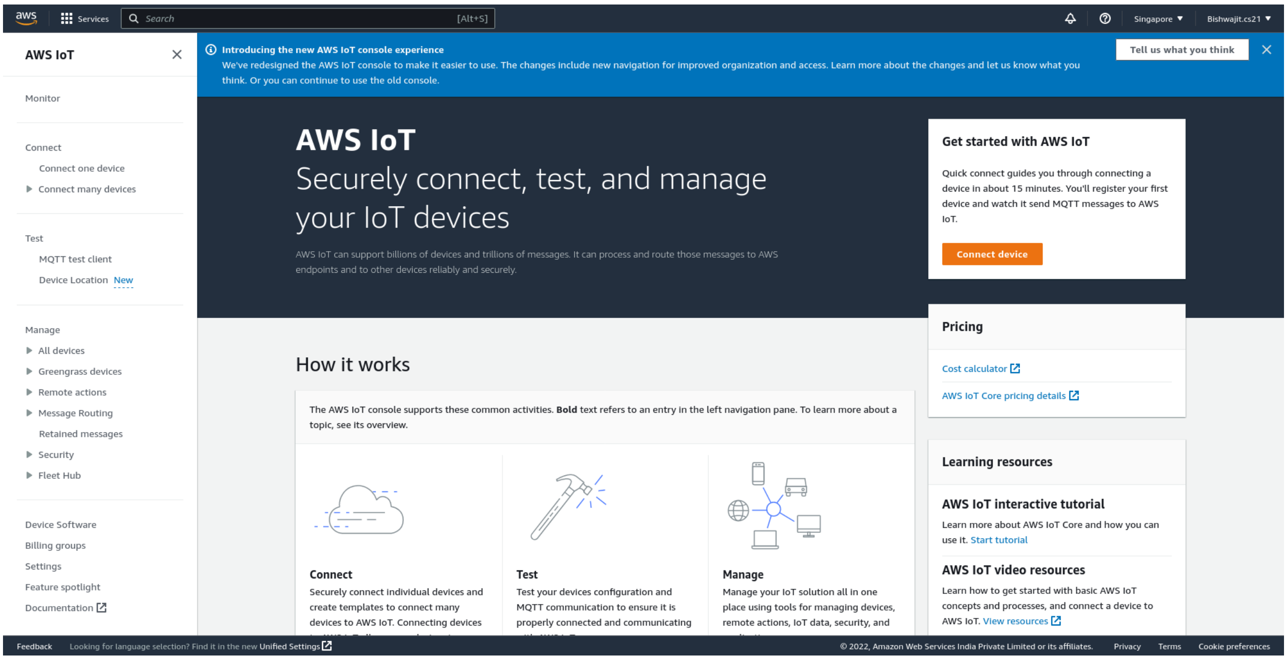
No top cost breakdowns. Cost breakdowns show when you use services.

Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services India Private Limited or its affiliates.

Figure: AWS home webpage

- From here go to `IoT Core`



Introducing the new AWS IoT console experience
We've redesigned the AWS IoT console to make it easier to use. The changes include new navigation for improved organization and access. Learn more about the changes and let us know what you think. Or you can continue to use the old console.

AWS IoT
Securely connect, test, and manage your IoT devices

AWS IoT can support billions of devices and trillions of messages. It can process and route those messages to AWS endpoints and to other devices reliably and securely.

Get started with AWS IoT
Quick connect guides you through connecting a device in about 15 minutes. You'll register your first device and watch it send MQTT messages to AWS IoT.

Pricing
Cost calculator
AWS IoT Core pricing details

How it works
The AWS IoT console supports these common activities. **Bold** text refers to an entry in the left navigation pane. To learn more about a topic, see its overview.

Connect
Securely connect individual devices and create templates to connect many devices to AWS IoT. Connecting devices

Test
Test your devices configuration and MQTT communication to ensure it is properly connected and communicating

Manage
Manage your IoT solution all in one place using tools for managing devices, remote actions, IoT data, security, and

Learning resources
AWS IoT interactive tutorial
Learn more about AWS IoT Core and how you can use it. Start tutorial
AWS IoT video resources
Learn how to get started with basic AWS IoT concepts and processes, and connect a device to AWS IoT. View resources

Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services India Private Limited or its affiliates.

Figure: IOT core webpage

- From here go to `All device > Things`

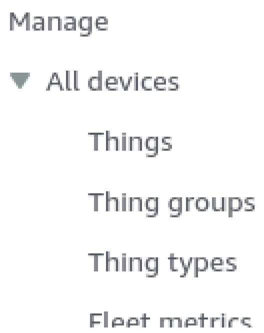
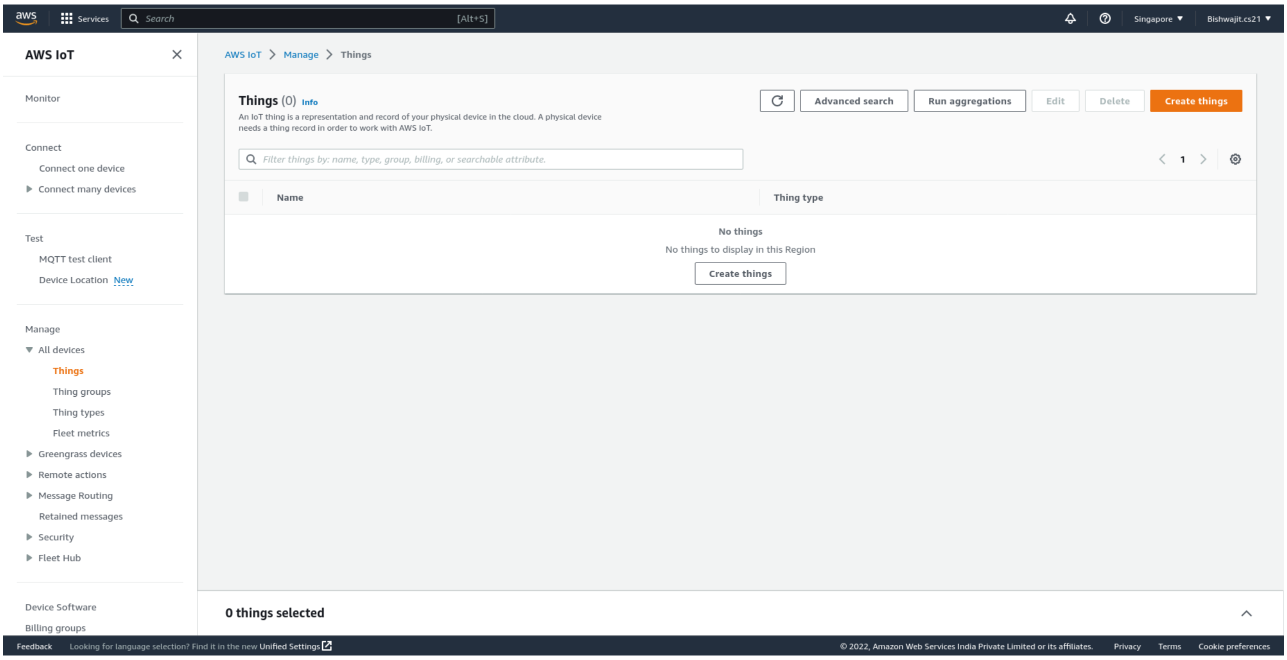


Figure: ALl device to things navigator

- click on things and this page will appear



Things (0) Info
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.

Filter things by: name, type, group, billing, or searchable attribute.

Name	Thing type

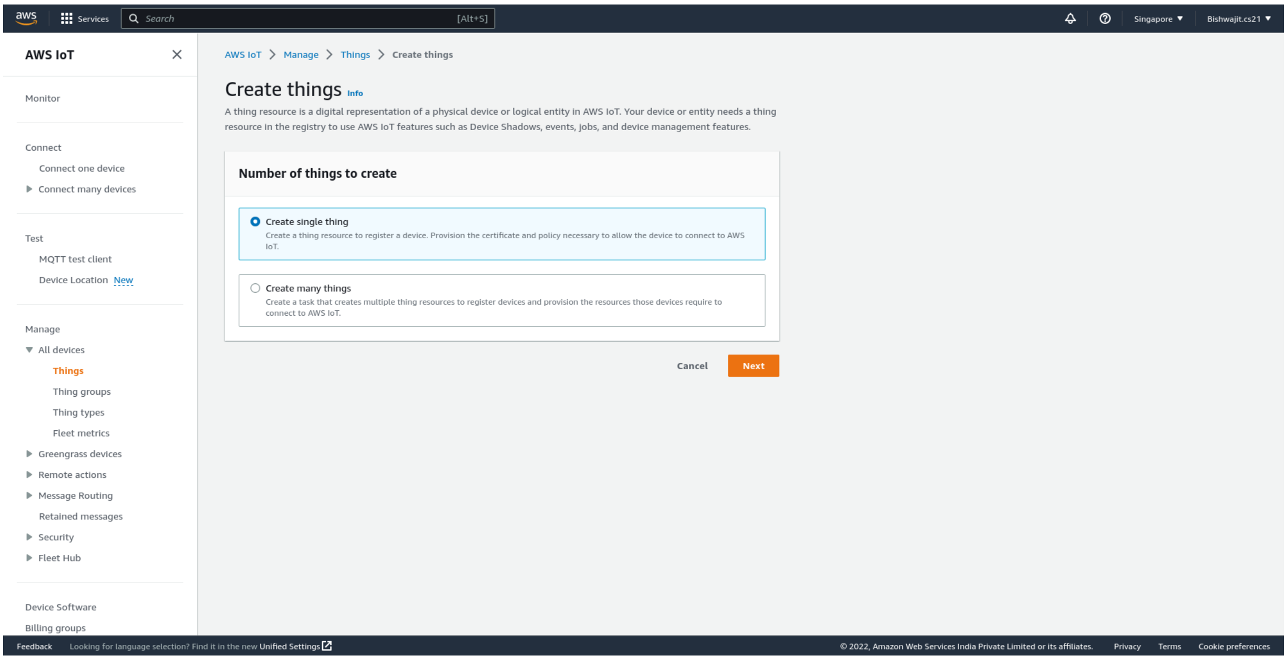
No things
No things to display in this Region

0 things selected

Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services India Private Limited or its affiliates.

Figure: Things page in AWS IoT Core

- Click on create thing and after that this page will appear



Create things Info

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.

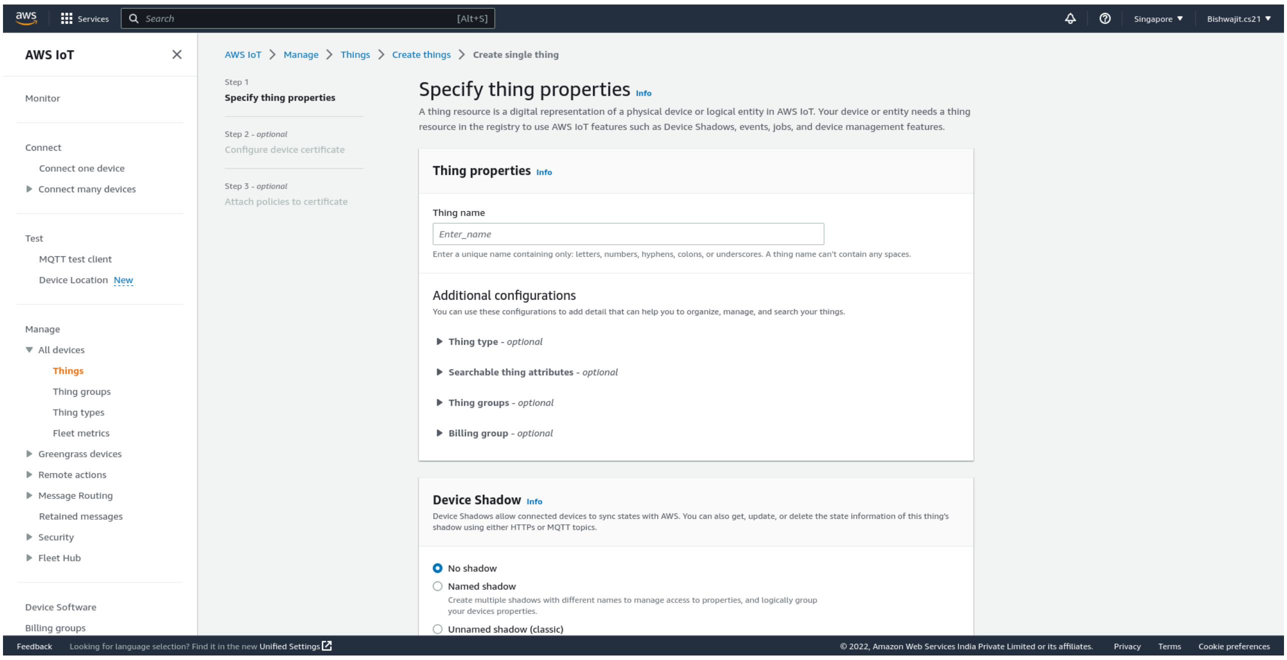
Number of things to create

● 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: AWS create thing page with many and single things creation

- Click on next and specify the name



Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

Specify thing properties Info

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 Info

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

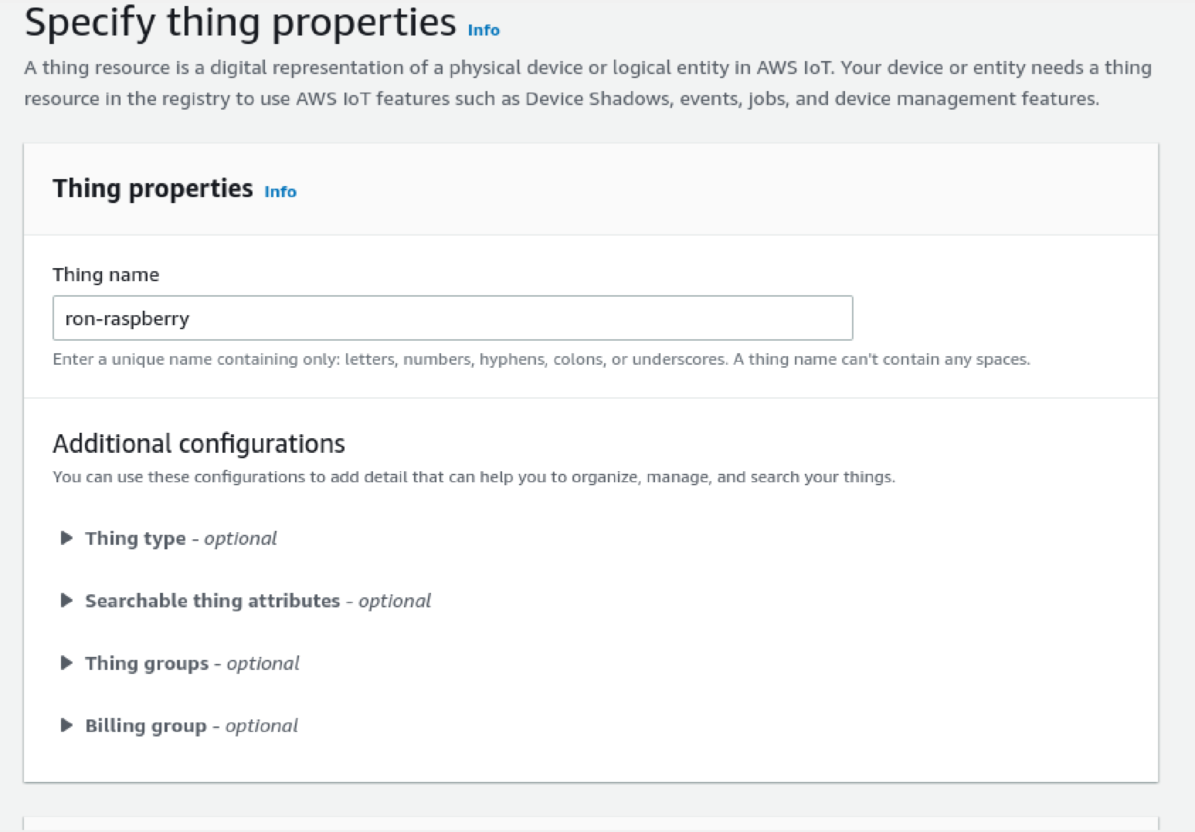
Additional configurations
You can use these configurations to add detail that can help you to organize, manage, and search your things.

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

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)

> For my Case I am using `ron-raspberry`



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

ron-raspberry

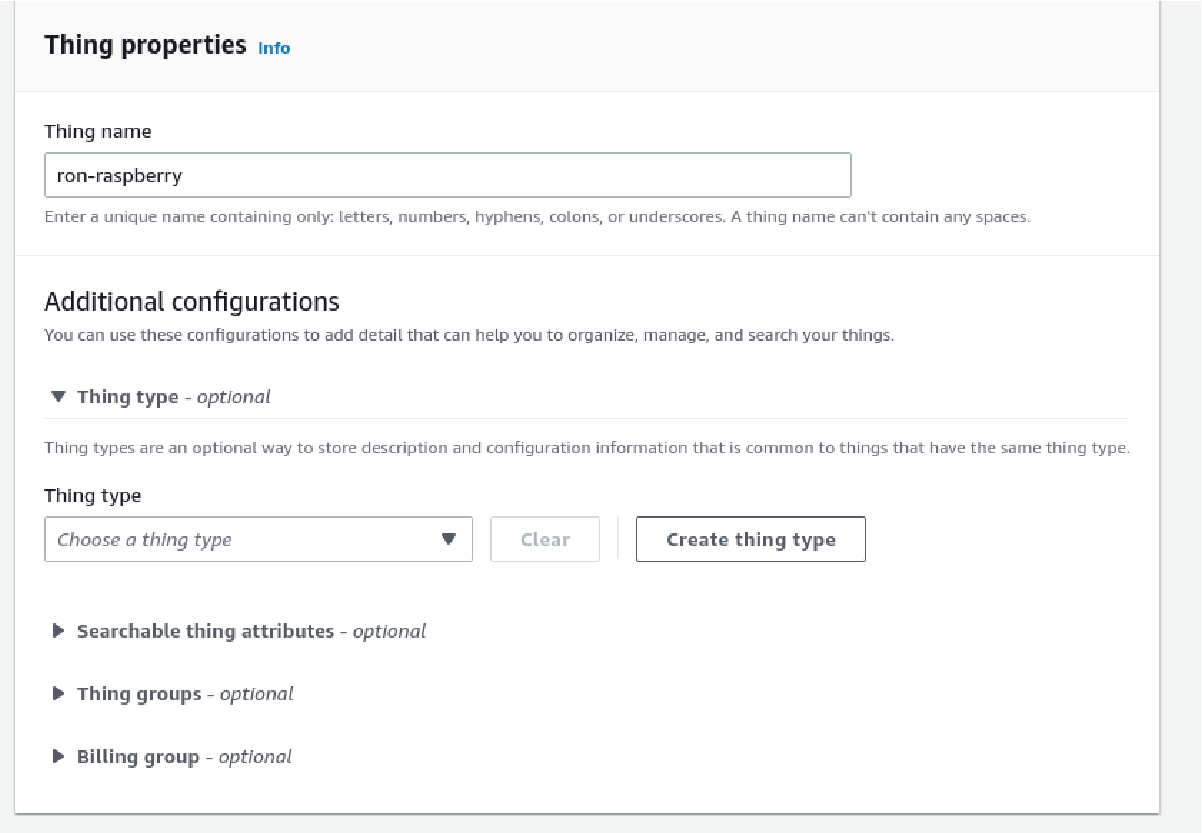
Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

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

- Now in the thing type
  - create a thing type



Thing properties

Thing name

ron-raspberry

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.

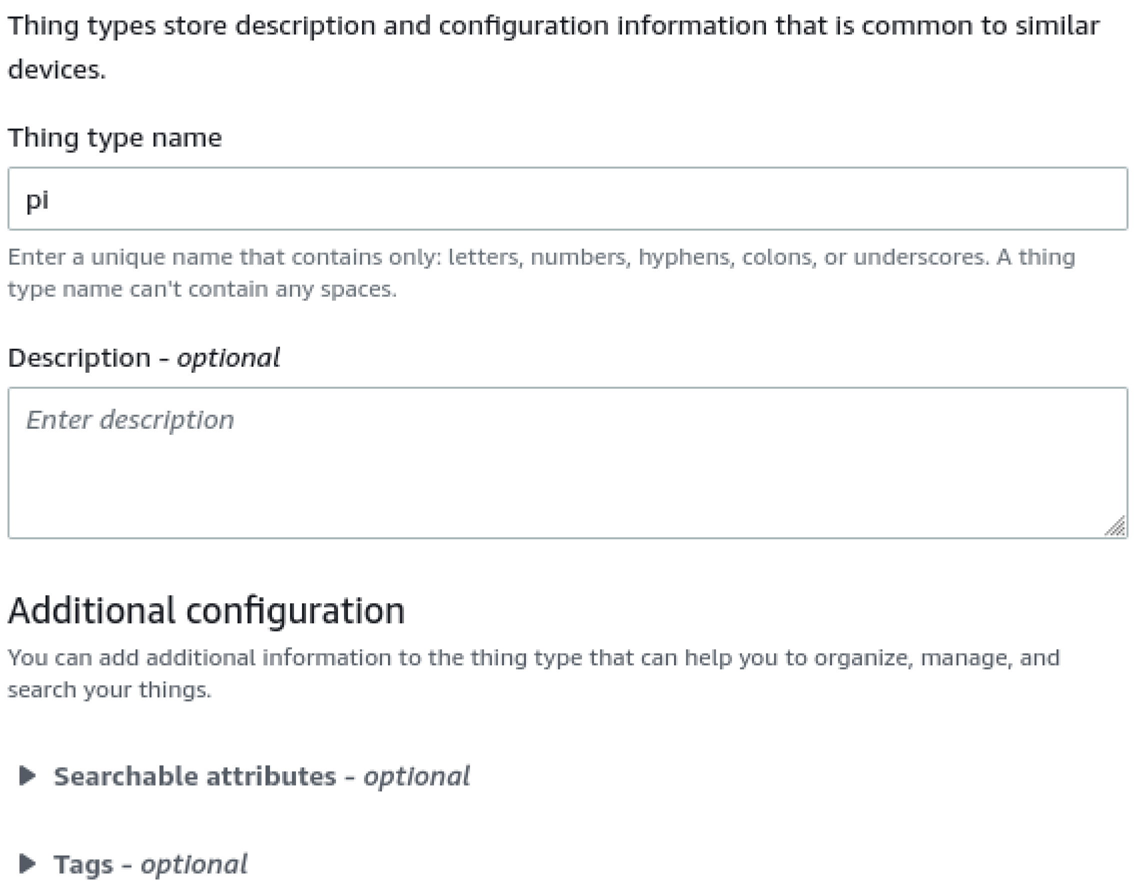
Thing types are an optional way to store description and configuration information that is common to things that have the same thing type.

Thing type

Choose a thing type ▼
Clear
Create thing type
▶ Searchable thing attributes - optional
▶ Thing groups - optional
▶ Billing group - optional

  - specify the name as `pi`

> Change the thing type if needed in this case we are working with raspberry pi model 4 B



Thing types store description and configuration information that is common to similar devices.

Thing type name

pi

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

Description - *optional*

Enter description

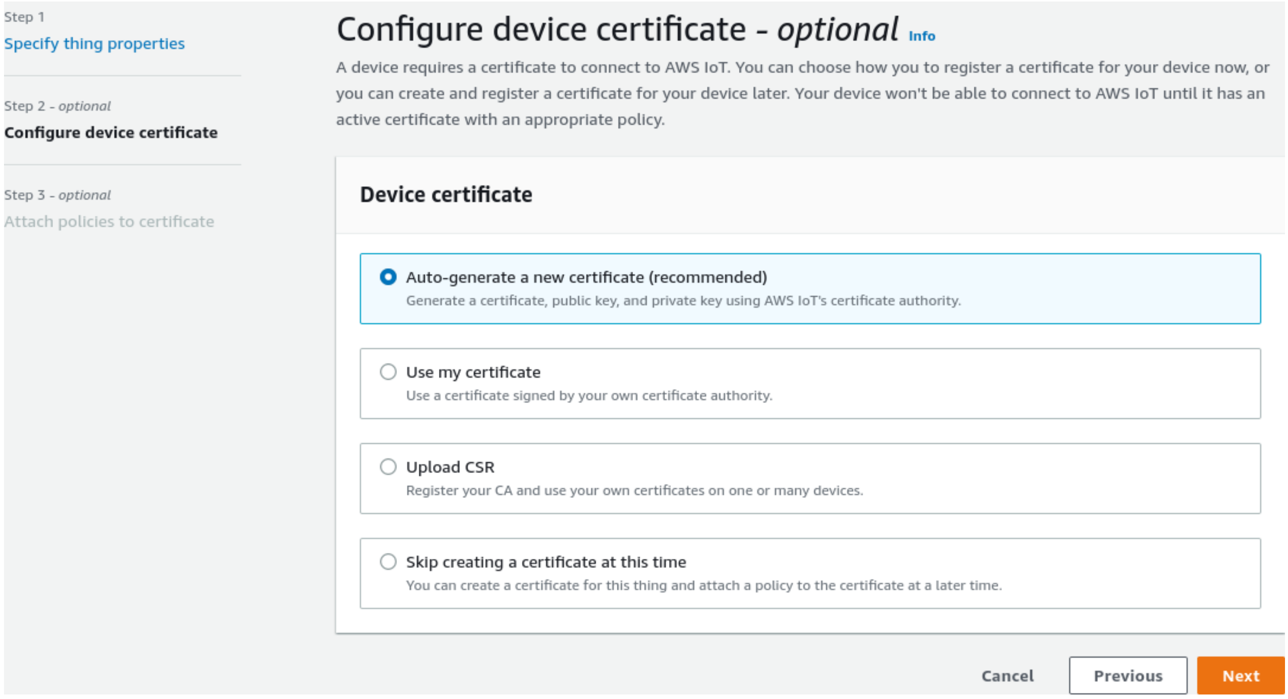
Additional configuration

You can add additional information to the thing type that can help you to organize, manage, and search your things.

▶ **Searchable attributes** - *optional*

▶ **Tags** - *optional*

- Go to next and you will get a certification generation gateway



Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

Configure device certificate - *optional*

A device requires a certificate to connect to AWS IoT. You can choose how you to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

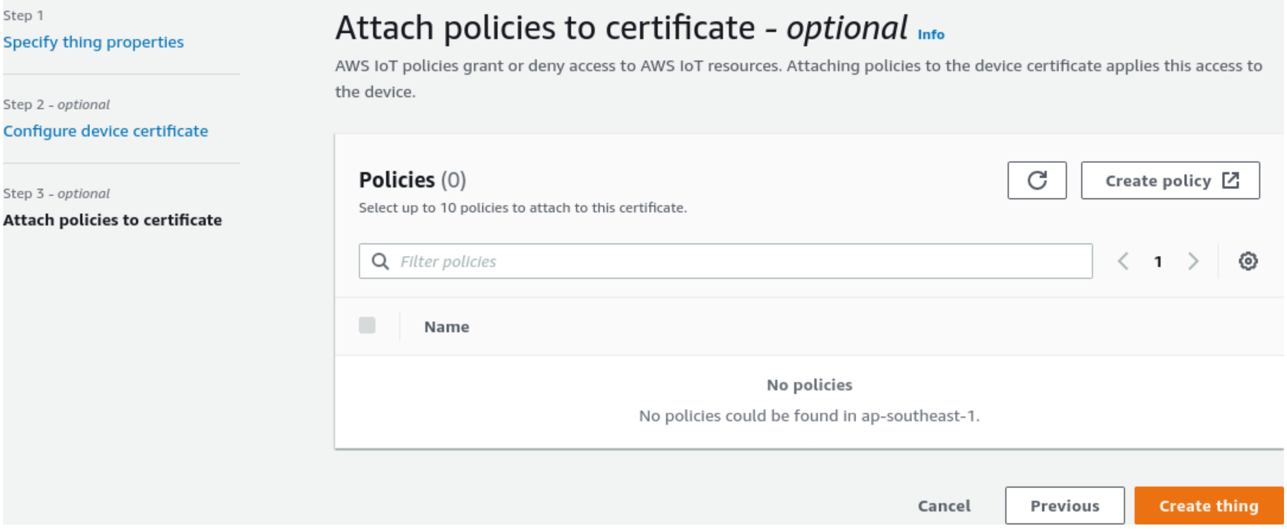
● Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

○ 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.

- Click on auto generate and go to next



Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

Policies (0)

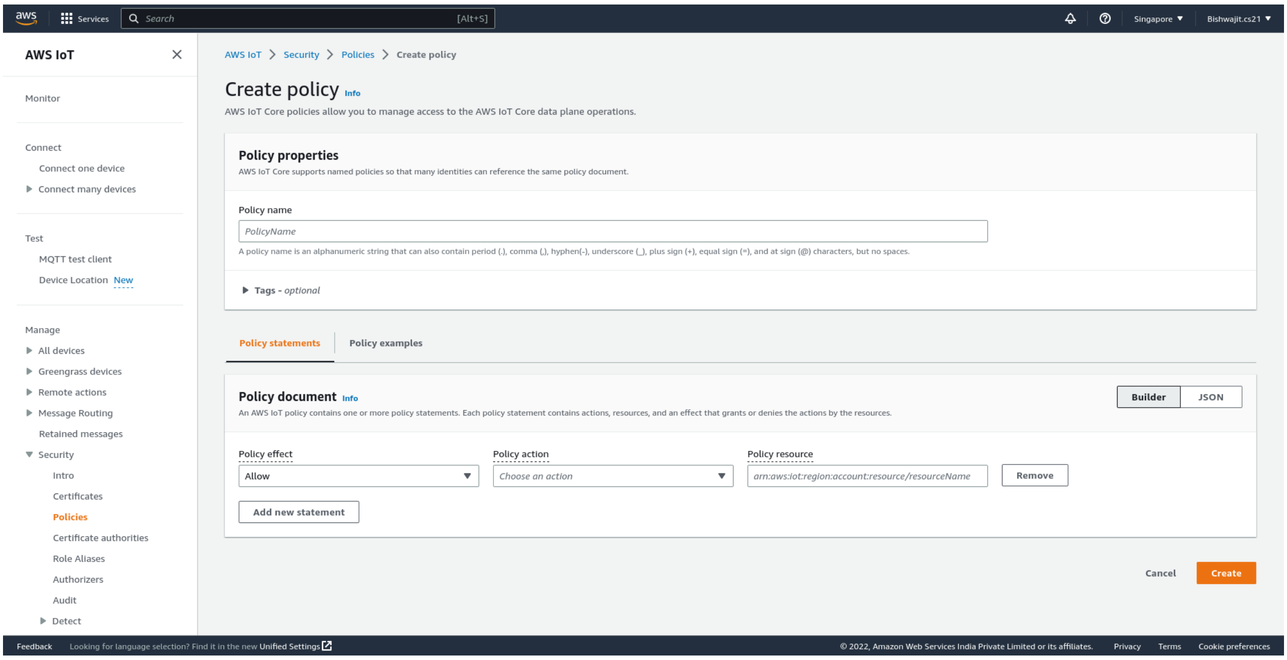
Select up to 10 policies to attach to this certificate.

Filter policies

No policies

No policies could be found in ap-southeast-1.

- Go to create policy where a new tab will open where you can create the policy and attactch with raspberry pi



Create policy Info

AWS IoT Core policies allow you to manage access to the AWS IoT Core data plane operations.

Policy properties

AWS IoT Core supports named policies so that many identities can reference the same policy document.

Policy name: *PolicyName*

A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

▶ Tags - *optional*

Policy document Info

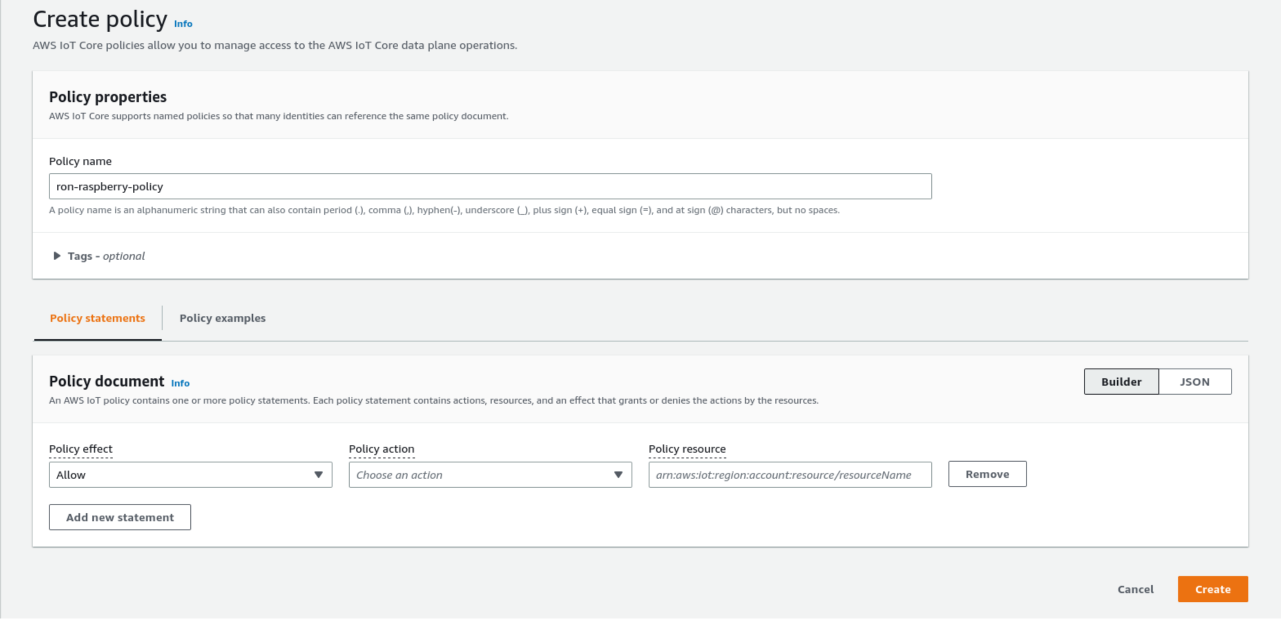
An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect: Allow

Policy action: *Choose an action*

Policy resource: *arn:aws:iot:region:account:resource/resourceName*

- Give a name for my case I am using `ron-raspberry-policy`



Create policy Info

AWS IoT Core policies allow you to manage access to the AWS IoT Core data plane operations.

Policy properties

AWS IoT Core supports named policies so that many identities can reference the same policy document.

Policy name: ron-raspberry-policy

A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

▶ Tags - *optional*

Policy document Info

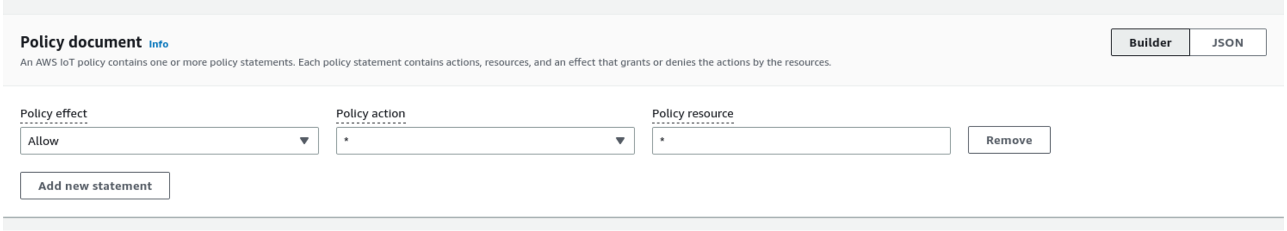
An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect: Allow

Policy action: *Choose an action*

Policy resource: *arn:aws:iot:region:account:resource/resourceName*

- after that set the polcy action to * and policy resource to *



Policy document Info

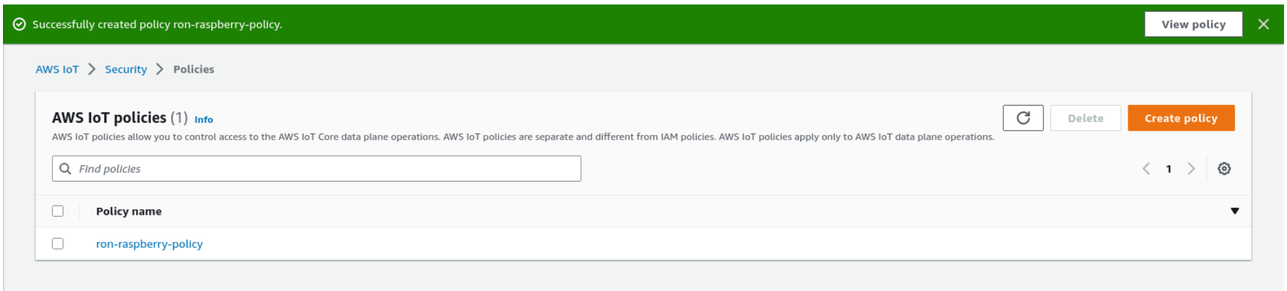
An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect: Allow

Policy action: *

Policy resource: *

- Now you will see this screen where it confirms policy is created



Successfully created policy ron-raspberry-policy.

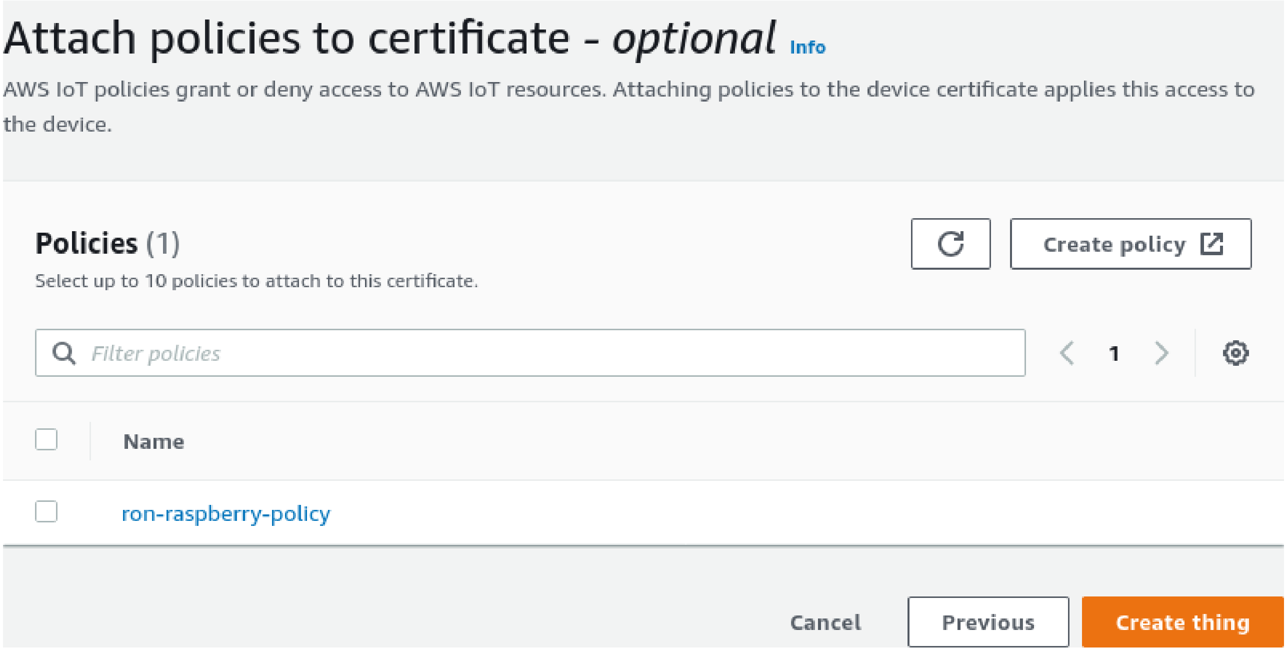
AWS IoT > Security > Policies

AWS IoT policies (1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations.

	Policy name
□	ron-raspberry-policy

- now close the TAB and go to previous tab where you will find this page



Attach policies to certificate - *optional* Info

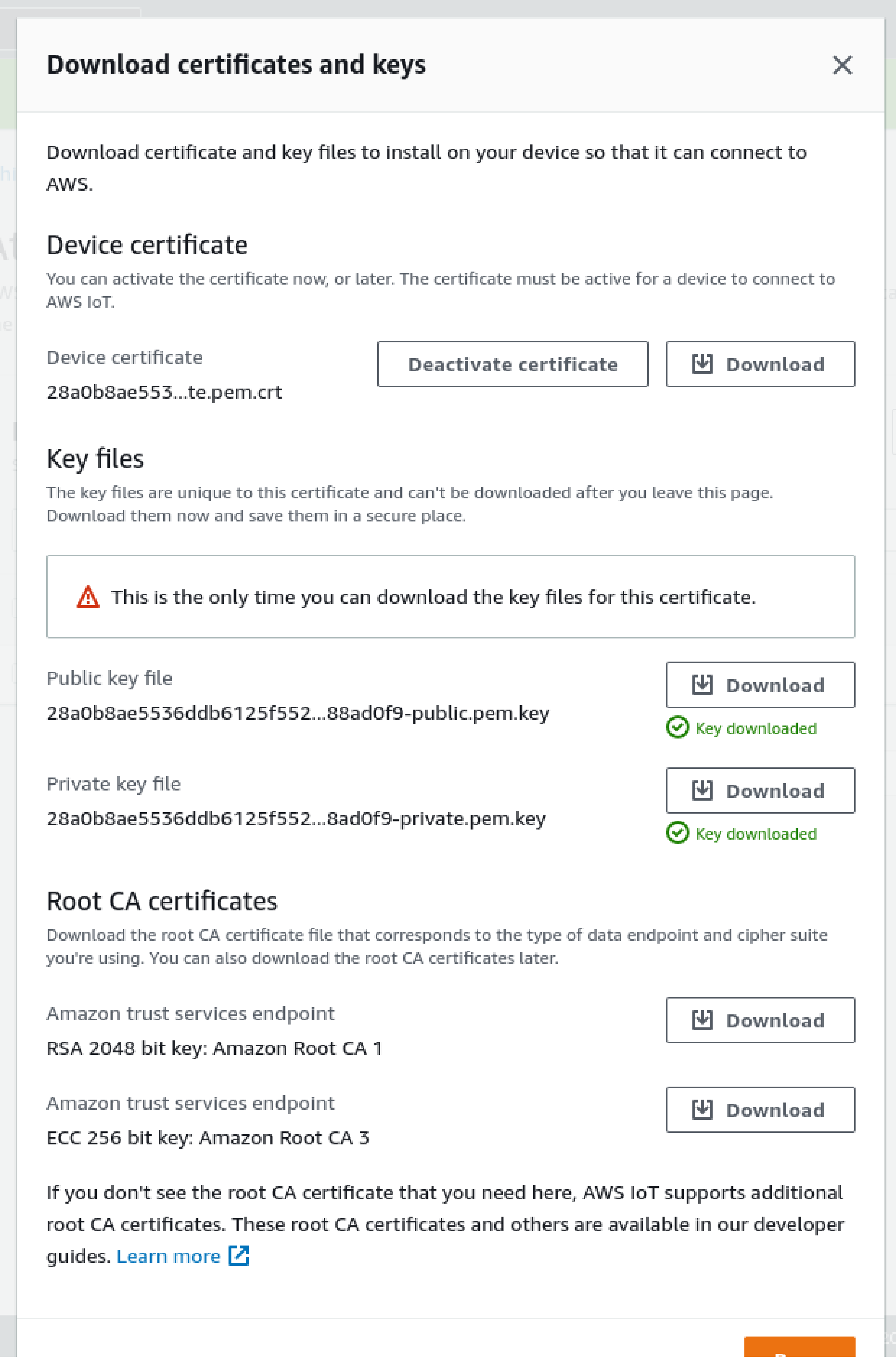
AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

Policies (1)

Select up to 10 policies to attach to this certificate.

	Name
□	ron-raspberry-policy

- select `ron-raspberry-policy` and then `create thing` and you will get this screen



Download certificates and keys

Download certificate and key files to install on your device so that it can connect to AWS.

Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate
28a0b8ae553...te.pem.crt

Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file
28a0b8ae5536ddb6125f552...88ad0f9-public.pem.key

Private key file
28a0b8ae5536ddb6125f552...8ad0f9-private.pem.key

Root CA certificates

Download the root CA certificate file that corresponds to the type of data endpoint and cipher suite you're using. You can also download the root CA certificates later.

Amazon trust services endpoint
RSA 2048 bit key: Amazon Root CA 1

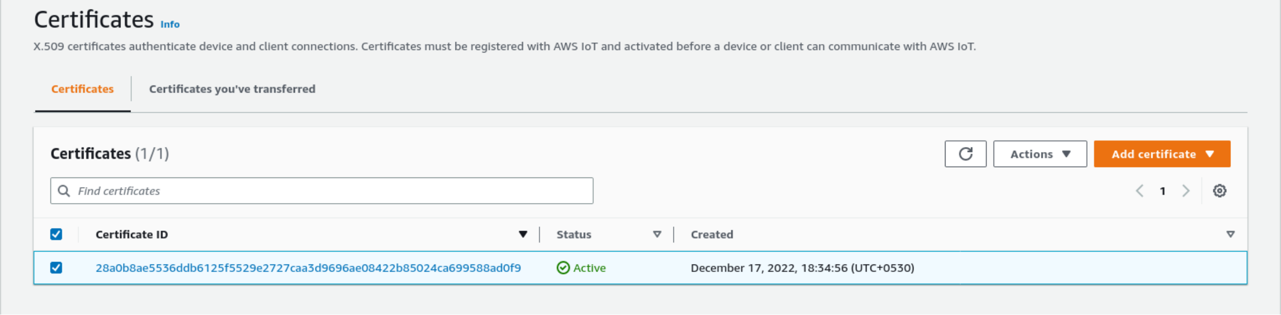
Amazon trust services endpoint
ECC 256 bit key: Amazon Root CA 3

If you don't see the root CA certificate that you need here, AWS IoT supports additional root CA certificates. These root CA certificates and others are available in our developer guides. Learn more

- Now as shown download device `certificate + Public key + private key`

- Then click on `Done`

- Go to certificates and attach to thing you created.



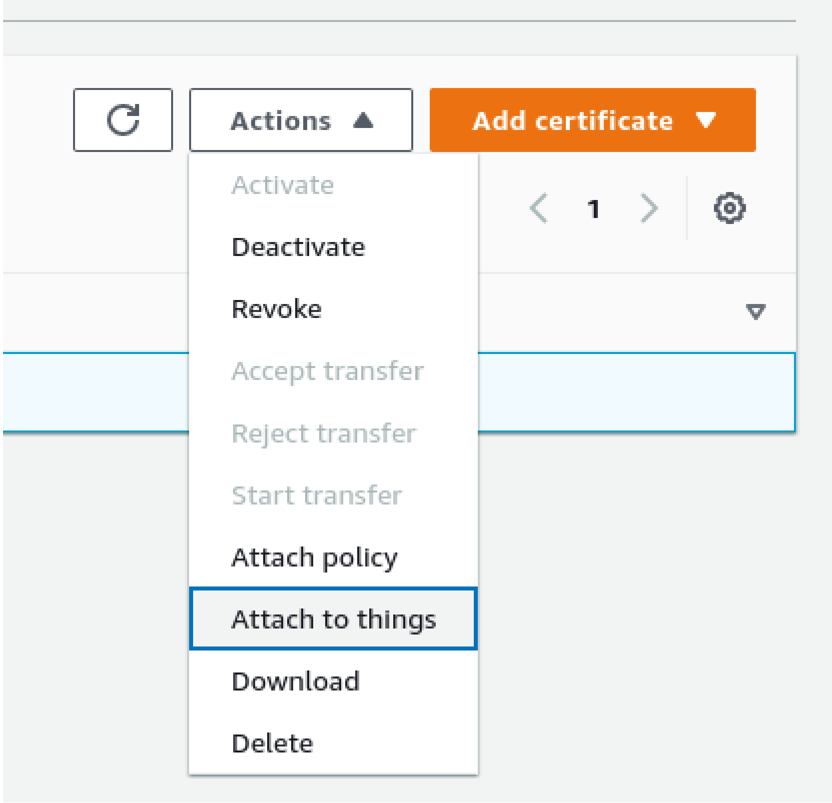
Certificates Info

X.509 certificates authenticate device and client connections. Certificates must be registered with AWS IoT and activated before a device or client can communicate with AWS IoT.

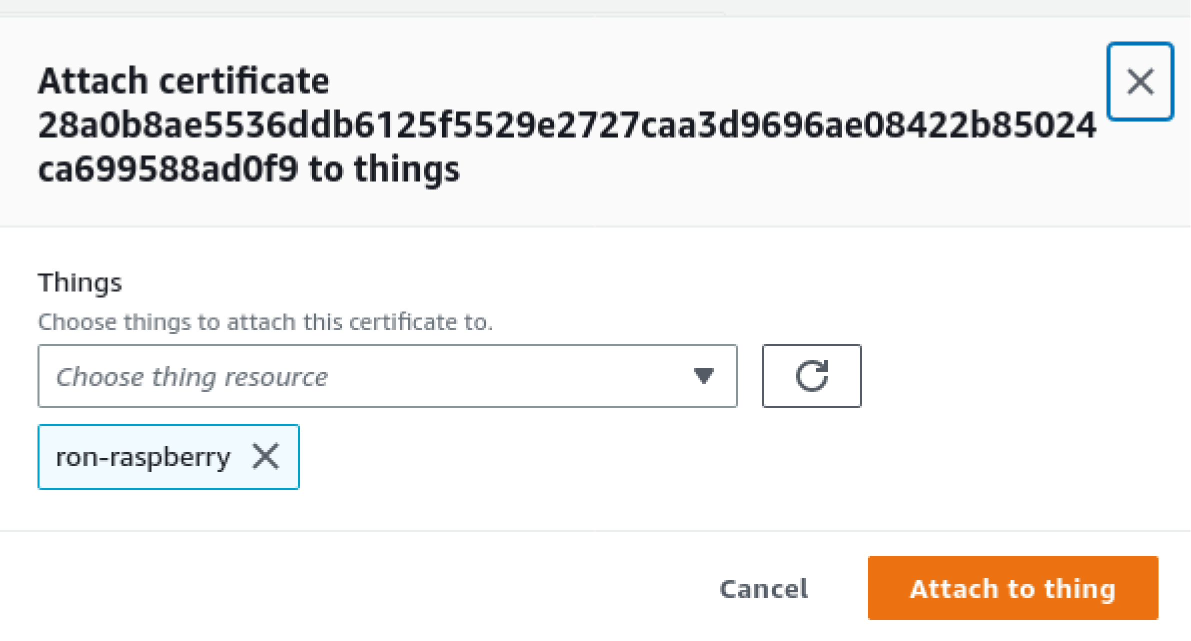
Certificates (1/1)

☑	Certificate ID	Status	Created
☑	28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9	Active	December 17, 2022, 18:34:56 (UTC+0530)

  - Select action and then select `Attatch to a thing`



  - Type your thing name select and click on attatch to thing

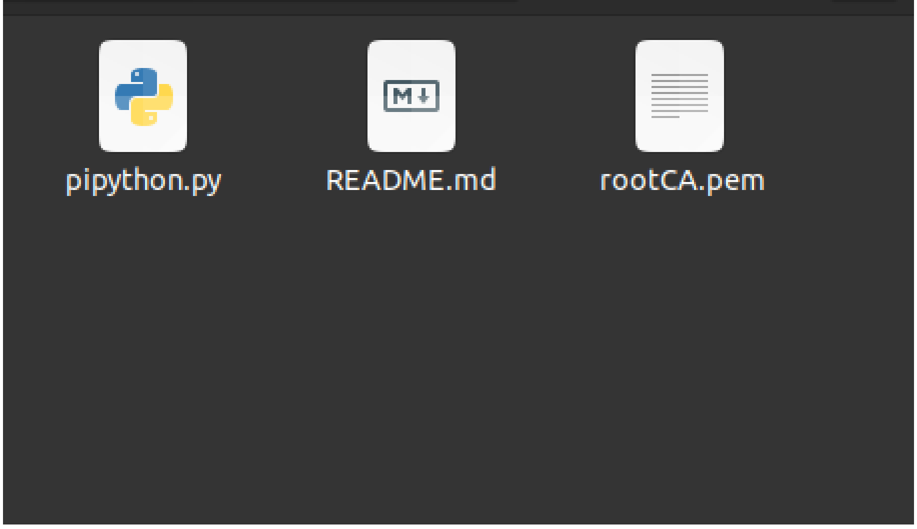


Attach certificate 28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9 to things

Things

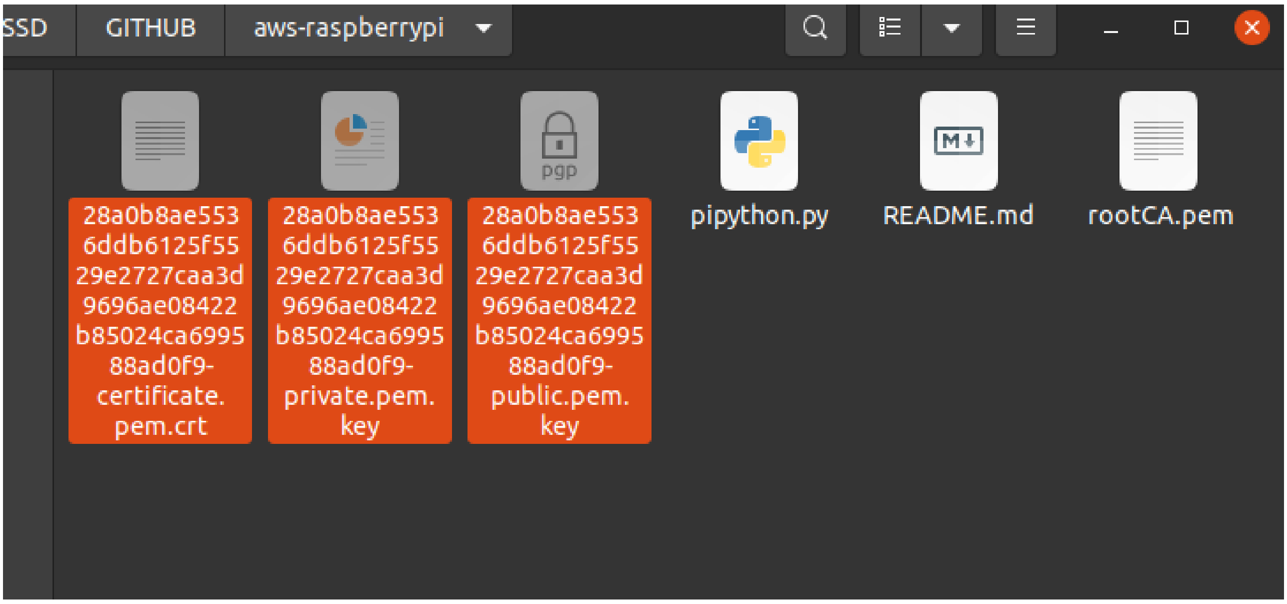
Choose things to attach this certificate to.

- Clone the repo `https://github.com/binaryupdates/aws-raspberrypi`



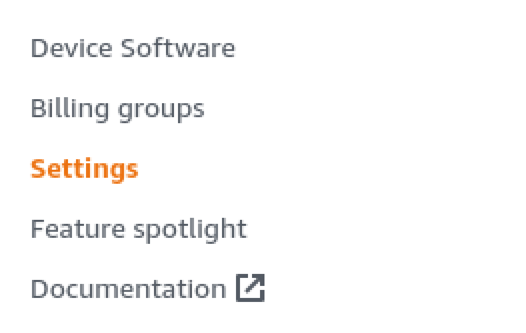
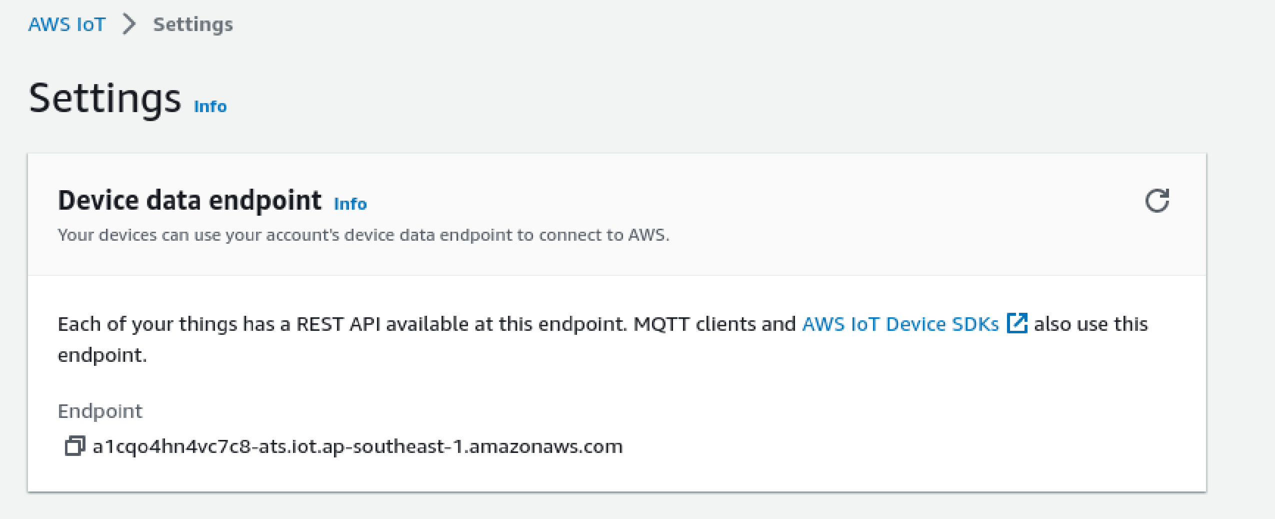
- This repo has only this files

- now COPY the three files here


SSD
GITHUB
aws-raspberrypi
28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9-certificate.pem.crt
28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9-private.pem.key
pgp
28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9-public.pem.key
pipython.py
README.md
rootCA.pem

- Now change the piython.py to this code where change file of certificate and change the of private key

```
import time
import paho.mqtt.client as mqtt
import ssl
import json
import _thread
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))


client = mqtt.Client()
client.on_connect = on_connect
client.tls_set(ca_certs='./rootCA.pem',
certfile='./28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9-certificate.pem.crt',
keyfile='./28a0b8ae5536ddb6125f5529e2727caa3d9696ae08422b85024ca699588ad0f9
-private.pem.key', tls_version=ssl.PROTOCOL_SSLv23)
client.tls_insecure_set(True)
client.connect("a1cqo4hn4vc7c8-ats.iot.ap-southeast-1.amazonaws.com", 8883,
60) #Taken from REST API endpoint - Use your own.


def intrusionDetector(Dummy):
    while(1):
        x = GPIO.input(21)
        if(x==0):
            print("Just Awesome")
            client.publish("device/data", payload="Hello from
BinaryUpdates!!" , qos=0, retain=False)
        time.sleep(5)

_thread.start_new_thread(intrusionDetector,("Create intrusion Thread",))

client.loop_forever()
```

> Here the client.connect URL you can find in the aws its different from others.
> Go to settings in aws and in the enpoint you will find the url of yours
>
> 
>
>
> From here you will see this appear
>
> 
>
>
> Your devices can use your account's device data endpoint to connect to AWS.
>
> Each of your things has a REST API available at this endpoint. MQTT clients and AWS IoT Device SDKs also use this endpoint.
>
> Endpoint
>
> a1cqo4hn4vc7c8-ats.iot.ap-southeast-1.amazonaws.com
>
> Copy the end point and paste in

```
client.connect("a1cqo4hn4vc7c8-ats.iot.ap-southeast-1.amazonaws.com", 8883,
60) #Taken from REST API endpoint - Use your own.
```

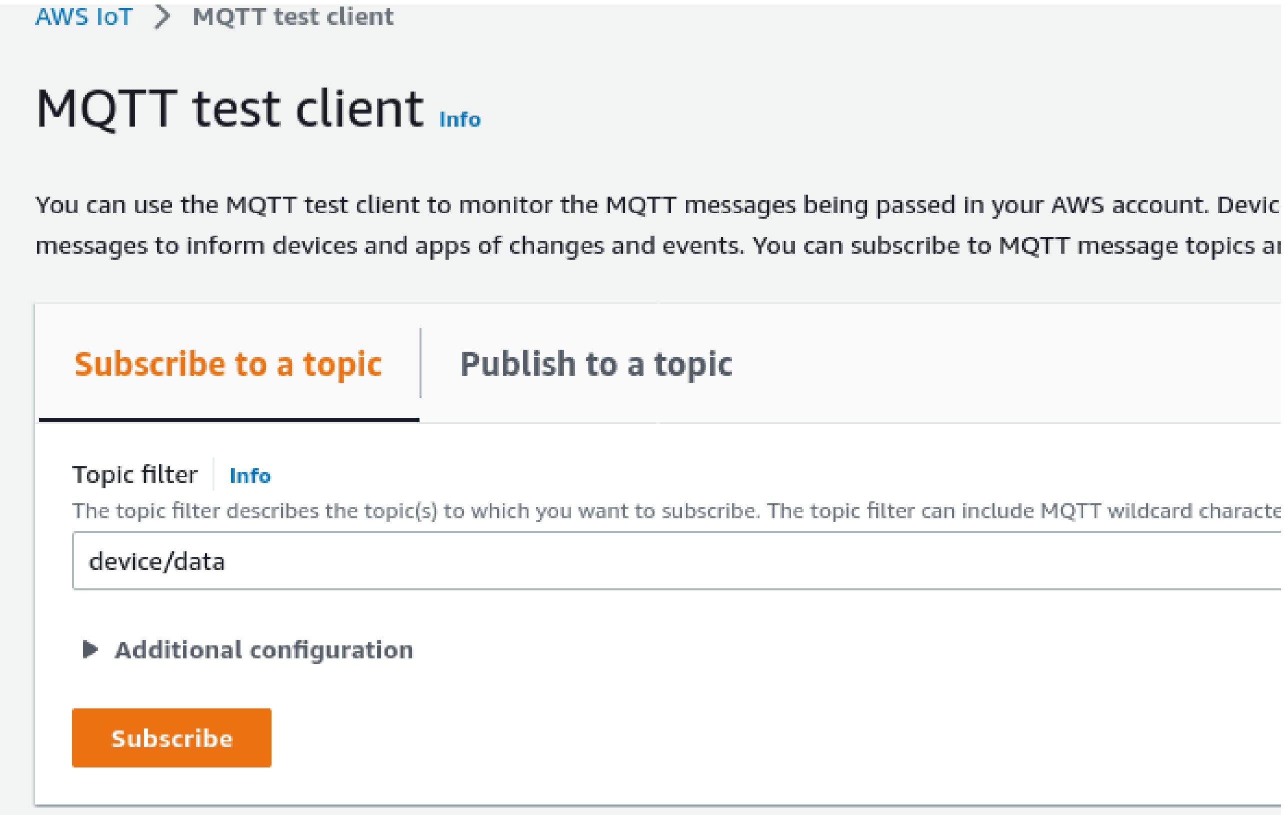
- you need to install paho-mqtt pypi

```
pip install paho-mqtt
```

- run the code and you will see this kind of output

```
(pt_venv) ron@ron-linux:~/SECONDARY_SSD/GITHUB/aws-raspberrypi$ python3
pipython.py
Just Awesome
Connected with result code 0
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
Just Awesome
```

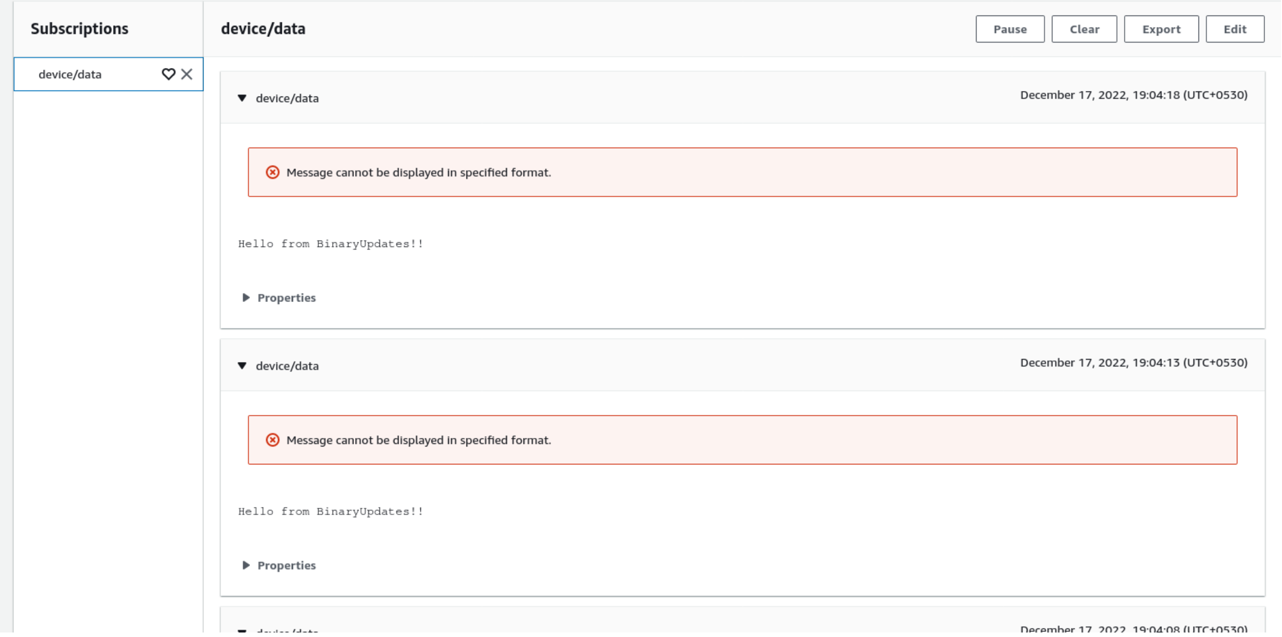
- TO if its working go to AWS and click on MQTT Test Client and type the topic name `device/data` and click on subscribe



You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devic
messages to inform devices and apps of changes and events. You can subscribe to MQTT message topics ar

Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characte
device/data

- The output will be like this



device/data

December 17, 2022, 19:04:18 (UTC+0530)

Message cannot be displayed in specified format.

```
Hello from BinaryUpdates!!
```

device/data

December 17, 2022, 19:04:13 (UTC+0530)

Message cannot be displayed in specified format.

```
Hello from BinaryUpdates!!
```

December 17, 2022, 19:04:08 (UTC+0530)

> IOT message is publishing

Establish wireless communication between 2 Raspberry pi using NRF and Bluetooth modules.

> add data

Establish Wireless communication between 2 Arduinos using NRF and Bluetooth modules.

> add data

Establishing client server communication

> add data

Sniffing messages communicated between client and server

- For this experiemnt we have used wireshark.
    - install wireshark in ubuntu
        - Run this commands one by one

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt update
sudo apt install wireshark
```

- run wireshark using sudo command

```
sudo wireshark
```

- another way is to use spacy
- run this command to install python scapy

```
sudo apt install python scapy
```

- copy and paste this code

```
#!/usr/bin/env python
import scapy.all as scapy
import argparse
from scapy.layers import http
def get_interface():
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--interface", dest="interface",
help="Specify interface on which to sniff packets")
    arguments = parser.parse_args()
    return arguments.interface

def sniff(iface=None):
    scapy.sniff(store=False, prn=process_packet)

def process_packet(packet):
    if packet.haslayer(http.HTTPRequest):
        print(f"[+] Http Request >>   {packet[http.HTTPRequest].Host}
to destination {packet[scapy.IP].src}  port number
{packet[scapy.TCP].sport} location : {packet[http.HTTPRequest].Path}")
        if packet.haslayer(scapy.Raw):
            load = bytes(packet[scapy.Raw].load)
            print(load)
            keys = ["username", "password", "pass", "email"]
            # for key in keys:
            #     if key in load:
            #         print("\n\n\n[+] Possible password/username >> "
+ load + "\n\n\n")
            #         break

# iface = get_interface()
sniff()
```

- run using sudo
  - For example

```
sudo python3 sniffer.py
```

Configure ESP8266 using Aurdino IDE and configure GPIOs to Run the Hellp World Program (Using LED)

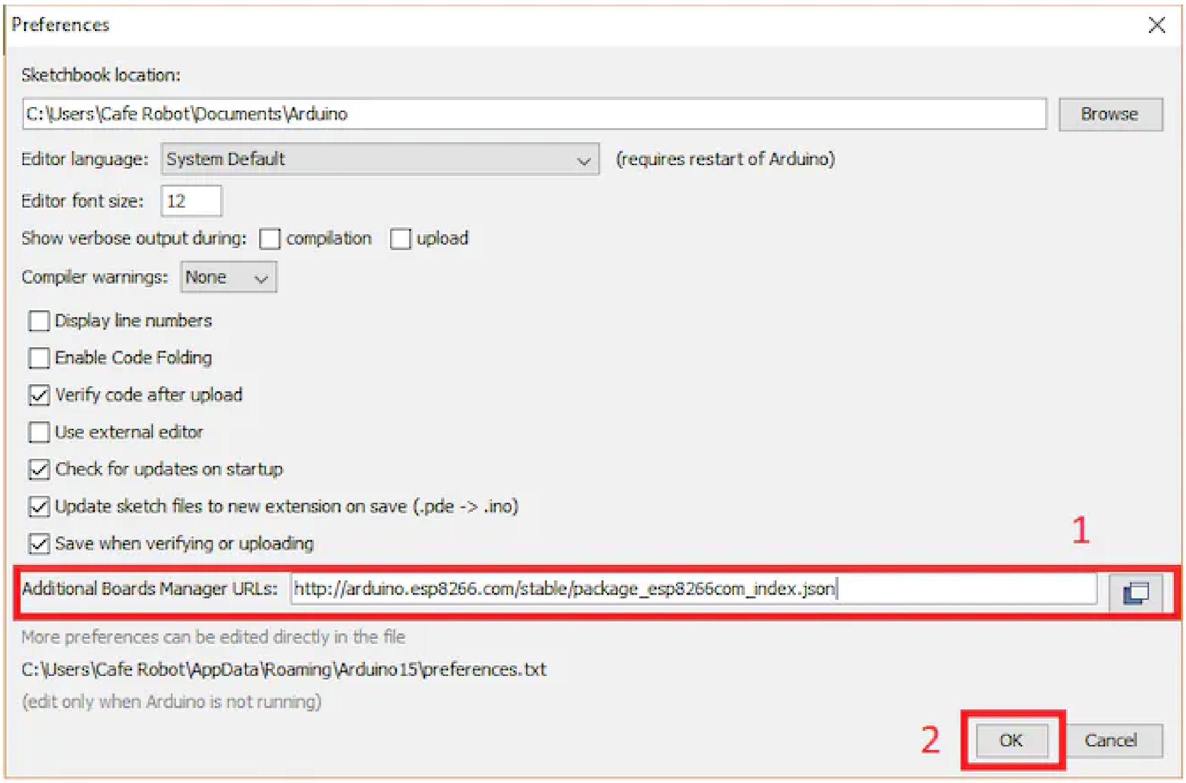
- The ESP8266 board comes with built in LED. Which can be used to blink with a delay of 250ms.
- After connecting the ESP8266 to we need to import/download ESP8266 binaries so that machine can compile the code compatible with the binaries in the ESP8266

> The steps the bellow / if already done not needed

  - COPY the link bellow

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

  - Choose Preferences in the File menu and enter the copied code in Additional Board Manager URLs part. Then press OK.



Preferences

Sketchbook location:

C:\Users\Cafe Robot\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Show verbose output during: □ compilation □ upload

Compiler warnings: None

□ Display line numbers

□ Enable Code Folding

☑ Verify code after upload

□ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

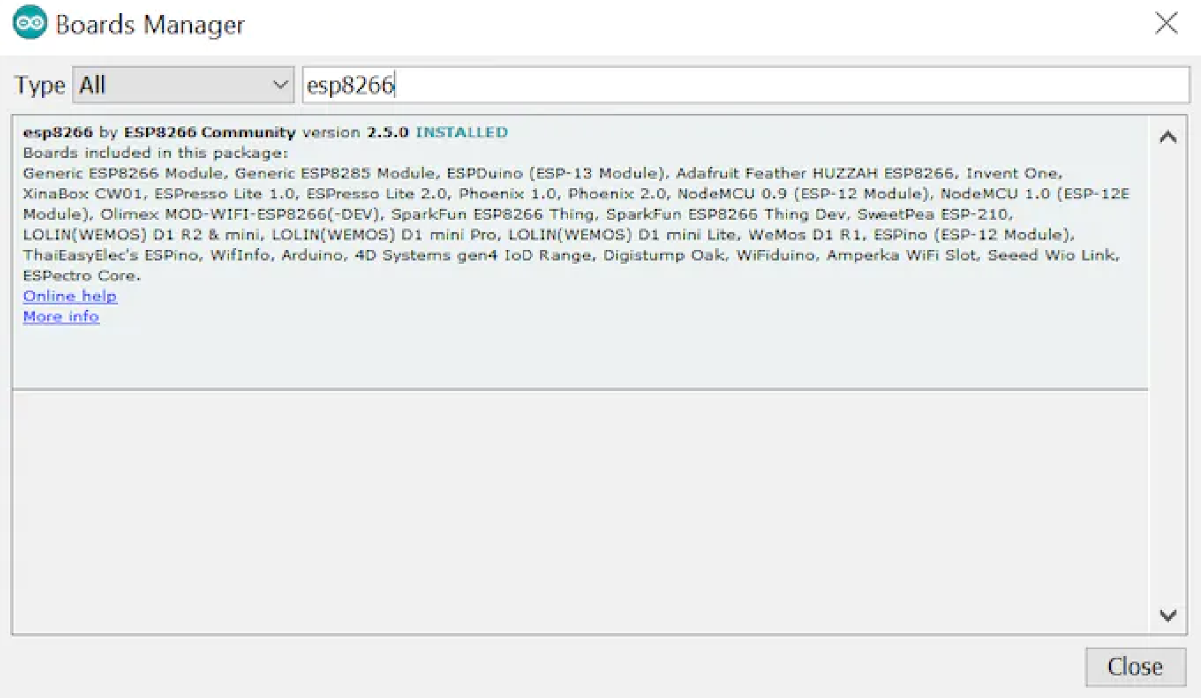
Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json

More preferences can be edited directly in the file

C:\Users\Cafe Robot\AppData\Roaming\Arduino15\preferences.txt

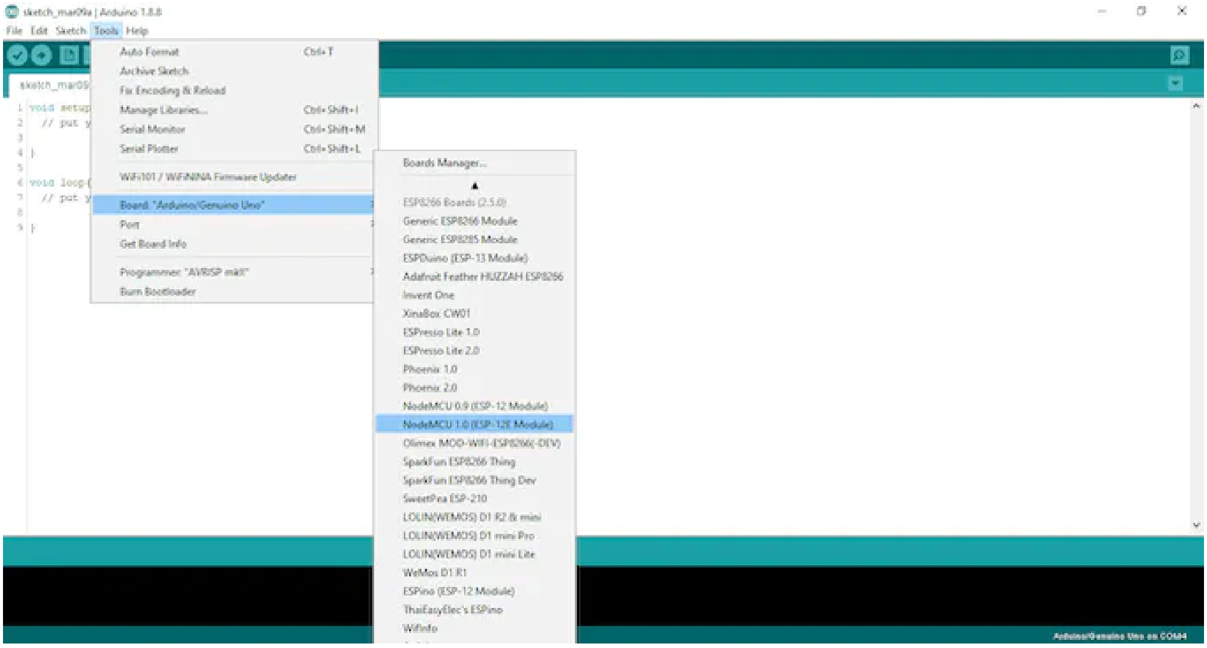
(edit only when Arduino is not running)

  - Search the word ESP8266 in Boards>boards manager from Tools menu. Then install ESP8266 boards. After complete installation, you will see the INSTALLED label on ESP8266 boards.



esp8266 by **ESP8266 Community** version **2.5.0** INSTALLED
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core.
Online help
More info

- After these two steps, you can see ESP8266 based boards such as NodeMCU in your Arduino IDE boards list, and you can choose your desired board to upload the code.

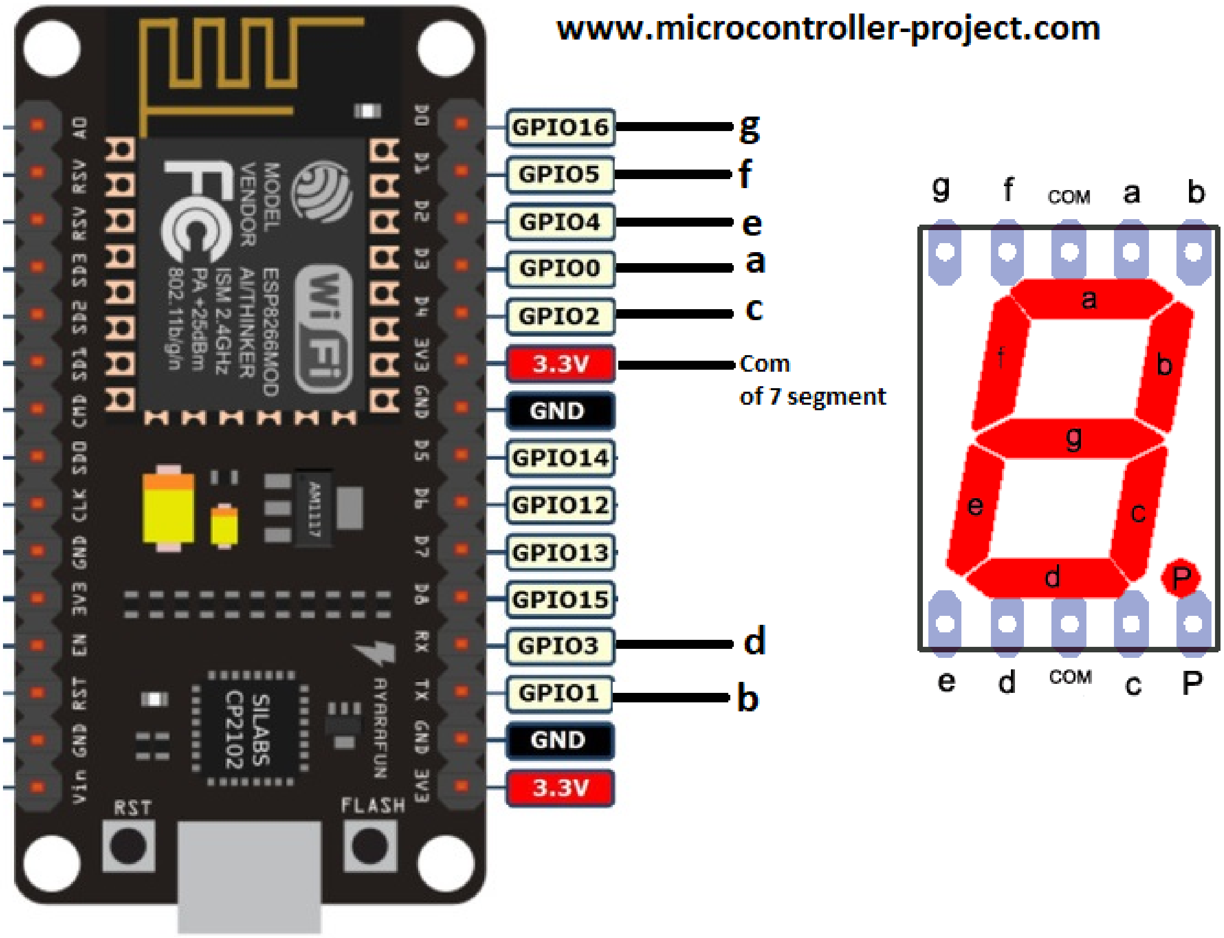

sketch_mar09a | Arduino 1.8.8
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
ESP8266 Boards (2.5.0)
Generic ESP8266 Module
Generic ESP8285 Module
ESPDuino (ESP-13 Module)
Adafruit Feather HUZZAH ESP8266
Invent One
XinaBox CW01
ESPresso Lite 1.0
ESPresso Lite 2.0
Phoenix 1.0
Phoenix 2.0
NodeMCU 0.9 (ESP-12 Module)
NodeMCU 1.0 (ESP-12E Module)
Olimex MOD-WIFI-ESP8266(-DEV)
SparkFun ESP8266 Thing
SparkFun ESP8266 Thing Dev
SweetPea ESP-210
LOLIN(WEMOS) D1 R2 & mini
LOLIN(WEMOS) D1 mini Pro
LOLIN(WEMOS) D1 mini Lite
WeMos D1 R1
ESPino (ESP-12 Module)
ThaiEasyElec's ESPino
WifInfo
Arduino/Genuino Uno on COM4

- Copy and run this simple code in your audrino IDE

```
void setup() {
// initialize digital pin LED_BUILTIN as an output.
pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the
voltage level)
delay(250);                       // wait for a second
digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
delay(250);                       // wait for a second
}
```

Configure ESP8266 using Aurdino IDE and display characters in the Seven Segment Display.

- Connect the seven segment display in this manner



- then run this code

Source Code

```
#include <ESP8266WiFi.h>

//const char* ssid = "Your SSID";
//const char* password = "Your Wifi Password";

//Seven segment pins attachecd with nodemcu pins
int a = 0;  //Gpio-0 with a of 7 segment display
int b = 1;  //Gpio-1 with b of 7 segment display
int c = 2;  //Gpio-2 with c of 7 segment display
int d = 3;  //Gpio-3 with d of 7 segment display
int e = 4;  //Gpio-4 with e of 7 segment display
int f = 5;  //Gpio-5 with f of 7 segment display
int g = 16; //Gpio-16 with g of 7 segment display

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);
}

void loop() {
  digitalWrite(a, LOW);
  delay(250);
  digitalWrite(b, LOW);
  delay(250);
  digitalWrite(c, LOW);
  delay(250);
  digitalWrite(d, LOW);
  delay(250);
  digitalWrite(e, LOW);
  delay(250);
  digitalWrite(f, LOW);
  delay(250);
  digitalWrite(g, LOW);
  delay(250);
  digitalWrite(a,HIGH);
  digitalWrite(b,HIGH);
  digitalWrite(c,HIGH);
  digitalWrite(d,HIGH);
  digitalWrite(e,HIGH);
  digitalWrite(f,HIGH);
  digitalWrite(g,HIGH);
  delay(250);
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);       //Displaying 1
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  delay(250);
  digitalWrite(a, LOW);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);      //Displaying 2
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, LOW);
  delay(250);
  digitalWrite(a, LOW);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);      //Displaying 2
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, LOW);
  delay(250);
  digitalWrite(a, LOW);
  digitalWrite(b, LOW);
  digitalWrite(c, LOW);
  digitalWrite(d, LOW);      //Displaying 3
  digitalWrite(e, HIGH);
```

```
    digitalWrite(f, HIGH);
    digitalWrite(g, LOW);
    delay(250);
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);     //Displaying 4
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    delay(250);
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);
    digitalWrite(e, HIGH);     //Displaying 5
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    delay(250);
     digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);    //Displaying 6
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    delay(250);
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);   //Displaying 7
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    delay(250);
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);   //Displaying 7
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    delay(250);
    digitalWrite(a, LOW);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);    //Displaying 9
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    delay(250);
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
```

```
  digitalWrite(g,HIGH);
  delay(250);

}
```