

Chapter 3: Amazon Web Services (AWS) IoT Core

After completing this chapter, you will understand the AWS IoT Core services, MQTT, and how to create Things, Policies, and Certificates.

3.1	INTRODUCTION	1
3.2	MQTT	1
3.3	AMAZON WEB SERVICES	2
3.4	AWS IOT INTRODUCTION	3
3.5	AWS IOT RESOURCES.....	5
3.5.1	THING.....	5
3.5.2	CERTIFICATE	5
3.5.3	POLICY	5
3.6	THING SHADOW	5
3.7	TOPICS.....	7
3.8	DEVICE SHADOW TOPICS	8
3.9	CREATING THINGS	9
3.9.1	TO CREATE A POLICY:.....	9
3.9.2	TO CREATE A THING:	11
3.10	TRANSFORM KEYS INTO “C”	17
3.11	PEMFILETOCSTRING.HTML	18
3.12	TEST CLIENT	20
3.12.1	SUBSCRIBING TO A TOPIC FROM THE TEST CLIENT	20
3.12.2	PUBLISHING TO A TOPIC FROM THE TEST CLIENT	21
3.13	GREENGRASS.....	22

3.1 Introduction

Whether you use Mbed, Amazon FreeRTOS, or connect directly to AWS using your own (or Cypress) firmware, at the heart of it your device will connect and communicate using the AWS IoT Core using MQTT. This chapter will discuss some of the concepts that are important to know when connecting your IoT device to AWS.

3.2 MQTT

MQTT is a lightweight messaging protocol that allows a device to **Publish Messages** to a specific **Topic** on a **Message Broker**. The Message Broker will then relay the message to all devices that are **Subscribed** to that **Topic**.

The format of the messages being sent in MQTT is unspecified. The message broker does not know (or care) anything about the format of the data and it is up to the system designer to specify an overall format of the data. All that being said, [JavaScript Object Notation \(JSON\)](#) has become the lingua franca of IoT.

A Topic is simply the name of a message queue e.g. "mydevice/status" or "mydevice/pressure". The name of a topic can be almost anything you want but by convention is hierarchical and separated with slashes "/".

Publishing is the process by which a client sends a message as a blob of data to a specific topic on the message broker.

A Subscription is the request by a device to have all messages published to a specific topic sent to the client.

A Message Broker is just a server that handles the tasks:

- Establishing connections (MQTT Connect)
- Tearing down connections (MQTT Disconnect)
- Accepting subscriptions to a Topic from clients (MQTT Subscribe)
- Turning off subscriptions (MQTT Unsubscribe)
- Accepting messages from clients and pushing them to the subscribers (MQTT Publish)

MQTT provides three levels of Quality of Service (QOS):

- Level 0: At most once (each message is delivered once or never)
- Level 1: At least once (each message is certain to be delivered, possibly multiple times)
- Level 2: Exactly once (each message is certain to arrive and do so only once)

MQTT operates on TCP Ports 1883 for non-secure and 8883 for secure (TLS).

Cloud providers that support MQTT include Amazon AWS and IBM Bluemix.

3.3 Amazon Web Services

From <https://aws.amazon.com/what-is-aws/>

AWS is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality (which makes more money for Amazon than their retail operations). AWS is built from a vast array of both virtual and actual servers and networks as well as a boatload of webserver software and administrative tools including (partial list):

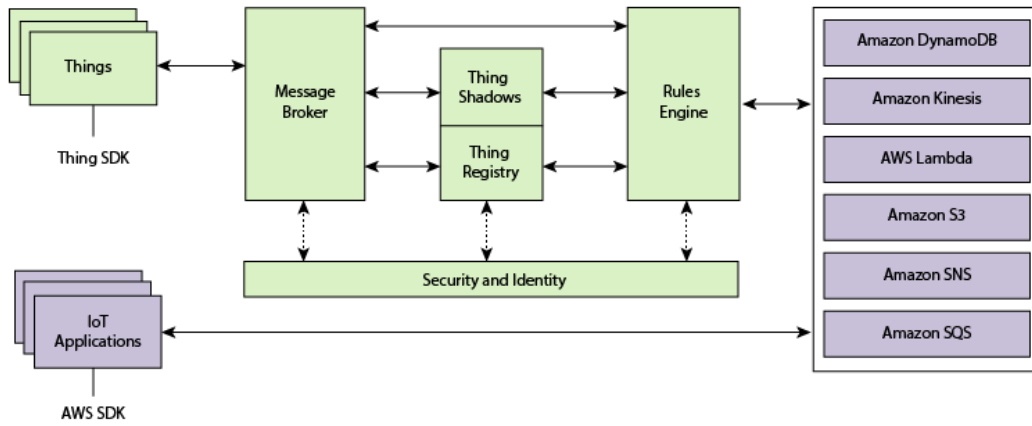
- [AWS IoT Core](#): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud ([EC2](#)/[EC3](#)): A virtualized compute capability, basically Linux, Windows etc. servers that you can rent.
- [Amazon Lambda](#): A Cloud service that enables you to send event driven tasks to be executed.
- Storage: Large fast file systems called [Amazon S3](#) & [AWS Elastic File System](#).

- Databases: Large fast databases called [Dynamo DB](#), [Amazon Relational Database \(RDS\)](#), [Amazon Aurora](#).
- Networking: Fast, fault tolerant, load balanced networks with entry points all over the world.
- Developer tools: A unified programming API supporting the AWS platform supporting a bunch of different languages.
- [Amazon Simple Notification System \(SNS\)](#): A platform to send messages including SMS and Email.
- [Amazon Simple Queueing Services \(SQS\)](#): A platform to send messages between servers (NOT the same thing as MQTT messages).
- [Amazon Kinesis](#): A platform to stream and analyze "massive" amounts of data. This is the plumbing for AWS IoT.

3.4 AWS IoT Introduction

The AWS IoT Cloud service supports MQTT Message Brokers, HTTP access, **plus** a bunch of server-side functionality that provides:

- Message Broker: A virtual MQTT Message Broker.
- A virtual HTTP Server.
- Thing Registry: A web interface to manage the access to your *things*.
- Security and identity: A web interface to manage the certificates and rules about *things*. You can create encryption keys and manage access privileges.
- A "shadow": An online cache of the most recent state of your *thing*.
- Rules Engine: An application that runs in the cloud that can subscribe to Topics and take programmatic actions based on messages – for example, you could configure it to subscribe to an "Alert" topic, and if a *thing* publishes a warning message to the alert topic, it uses Amazon SNS to send a SMS Text Message to your cell phone
- IoT Applications: An SDK to build Web pages and cell phone Apps.



3.5 AWS IoT Resources

There are three types of resources in AWS: *Things*, *Certificates*, and *Policies*. The second exercise will take you step by step through the process to create each of them.

3.5.1 Thing

A *thing* is a representation of a device or logical entity. It can be a physical device or sensor (for example, a light bulb or a switch on a wall). It can also be a logical entity like an instance of an application or a physical entity that does not connect to AWS IoT but can be related to other devices that do (for example, a car that has engine sensors or a control panel).

3.5.2 Certificate

AWS IoT provides mutual authentication and encryption at all points of connection so that data is never exchanged between *things* and AWS IoT without a proven identity. AWS IoT supports X.509 certificate-based authentication. Connections to AWS use certificate-based authentication. You should attach policies to a certificate to allow or deny access to AWS IoT resources. A root CA (certification authority) certificate is used by your device to ensure it is communicating with the actual Amazon Web Services site. You can only connect your *thing* to the AWS IoT Cloud via TLS.

3.5.3 Policy

When you create your internet-connected *thing*, you must create and attach an AWS IoT policy that will determine what AWS IoT operations the *thing* may perform. AWS IoT policies are JSON documents and they follow the same conventions as AWS Identity and Access Management policies.

You can specify permissions for specific resources such as topics and shadows. Here is an example of a Policy created for a new *thing* that allows any IoT action for any resource.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [ "iot:*" ],
      "Resource": [ "*" ],
      "Effect": "Allow"
    }
  ]
}
```

3.6 Thing Shadow

<http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html>

A *thing* shadow (sometimes referred to as a device shadow) is a JSON document that is used to store and retrieve current state information for a *thing* (device, app, and so on). The *Thing* Shadows service maintains a *thing* shadow for each *thing* you connect to AWS IoT. You can use *thing* shadows to get and set the state of a *thing* over MQTT or HTTP, regardless of whether the *thing* is currently connected to the Internet. Each *thing* shadow is uniquely identified by its name.

The JSON Shadow document representing the device has the following properties:

- state:
 - desired: The desired state of the *thing*. Applications can write to this portion of the document to update the state of a *thing* without having to directly connect to a *thing*.
 - reported: The reported state of the *thing*. *Things* write to this portion of the document to report their new state. Applications read this portion of the document to determine the state of a *thing*.
- metadata: Information about the data stored in the state section of the document. This includes timestamps, in Epoch time, for each attribute in the state section, which enables you to determine when they were updated.
- timestamp: Indicates when the message was transmitted by AWS IoT. By using the timestamp in the message and the timestamps for individual attributes in the desired or reported section, a *thing* can determine how old an updated item is, even if it doesn't feature an internal clock.
- clientToken: A string unique to the device that enables you to associate responses with requests in an MQTT environment.
- version: The document version. Every time the document is updated, this version number is incremented. Used to ensure the version of the document being updated is the most recent.

An example of a shadow document looks like this:

```
{
  "state" : {
    "desired" : {
      "color" : "RED",
      "sequence" : [ "RED", "GREEN", "BLUE" ]
    },
    "reported" : {
      "color" : "GREEN"
    }
  },
  "metadata" : {
    "desired" : {
      "color" : {
        "timestamp" : 12345
      },
      "sequence" : {
        "timestamp" : 12345
      }
    },
    "reported" : {
      "color" : {
        "timestamp" : 12345
      }
    }
  },
  "version" : 10,
  "clientToken" : "UniqueClientToken",
  "timestamp": 123456789
}
```

If you want to update the Shadow, you can publish a JSON document with just the information you want to change to the correct topic. For example, you could do:

```
{
  "state" : {
    "reported" : {
      "color": "BLUE"
    }
  }
}
```

Note that spaces and carriage returns are optional, so the above could be written as:

```
{"state":{"reported":{"color": "BLUE"}}}
```

3.7 Topics

You can interact with AWS using either MQTT or HTTP. While topics are an MQTT concept, you will see later that topic names are important even when using HTTP to interact with *thing* shadows. The AWS Message Broker will allow you to create Topics with almost any name, with one exception: Topics named "\$aws/..." are reserved by AWS IoT for specific functions.

As the system designer, you are responsible for defining what the topics mean and do in your system. Some [best practices](#) include:

1. Don't use a leading forward slash.
2. Don't use spaces.
3. Keep the topic short and concise.
4. Use only ASCII characters.
5. Embed a unique identifier e.g. the name of the *thing*.

For example, a good topic name for a temperature sensing device might be: myDevice/temperature.

3.8 Device Shadow Topics

<https://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html>

Each *thing* that you have will have a group of topics of the form "\$aws/things/<thingName>/shadow/<type>" which allow you to publish and subscribe to topics relating to the shadow. The specific shadow topics that exist are:

MQTT Topic Suffix <type>	Function
/update	The JSON message that you publish to this topic will become the new state of the shadow.
/update/accepted	AWS will publish a message to this topic in response to a message to /update indicating a successful update of the shadow.
/update/documents	When a document is updated via a publish to /update, the complete new document is published to this topic.
/update/rejected	AWS will publish a message to this topic in response to a message to /update indicating a rejected update of the shadow.
/update/delta	After a message is sent to /update, the AWS will send a JSON message if the desired state and the reported state are not equal. The message contains all attributes that don't match.
/get	If a <i>thing</i> publishes a message to this topic, AWS will respond with a message to either /get/accepted or /get/rejected with the current state of the shadow.
/get/accepted	
/get/rejected	
/delete	If a <i>thing</i> publishes a message to this topic, AWS will delete the shadow document.
/delete/accepted	AWS will publish to this topic when a successful /delete occurs.
/delete/reject	AWS will publish to this topic when a rejected /delete occurs.

The update topic is useful when you want to update the state of a *thing* on the cloud. For example, if you have a *thing* called "myThing" and want to update a value called "temperature" to 25 degrees in the state of the thing, you would publish (for MQTT) or POST (for HTTP) using the following topic and message:

topic: \$aws/things/myThing/shadow/update

message: {"state":{"reported":{"temperature":25}}}

Once the message is received, the MQTT message broker will publish to the /accepted and /documents topics with the appropriate information.

If you are using the MQTT test server to subscribe to topics, you can use "#" as a wildcard at the end of a topic to subscribe to multiple topics. For example, you can use "\$aws/things/theThing/shadow/#" to subscribe to all shadow topics for the *thing* called "theThing".

You can also use "+" as a wildcard in the middle of a topic to subscribe to multiple topics. For example, you can use "\$aws/things/+shadow/update" to subscribe to update topics for all *thing* shadows.

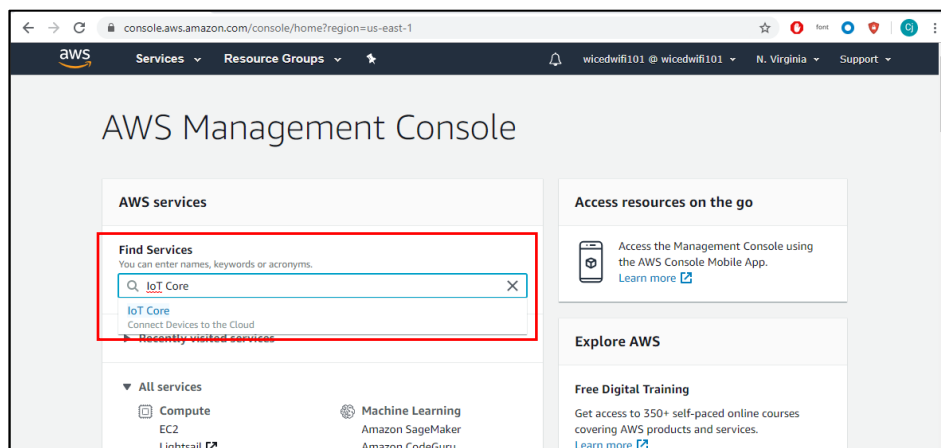
3.9 Creating Things

At the end of the thing creation process, AWS will ask you which policy to attach to your thing. You can have a generic policy that applies to multiple things, or you can have a specific policy that applies to each thing. This is a security architecture decision that the applications developer is responsible for.

3.9.1 To create a policy:

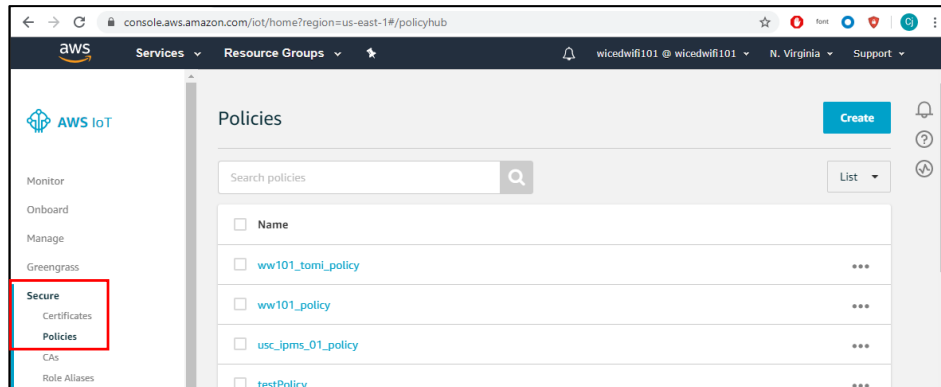


1. Go to <https://aws.amazon.com/console/> and login using the credentials provided.
2. If needed, click the "AWS" icon in the top left.
3. Under **Find Services**, search for and select **IoT Core**.

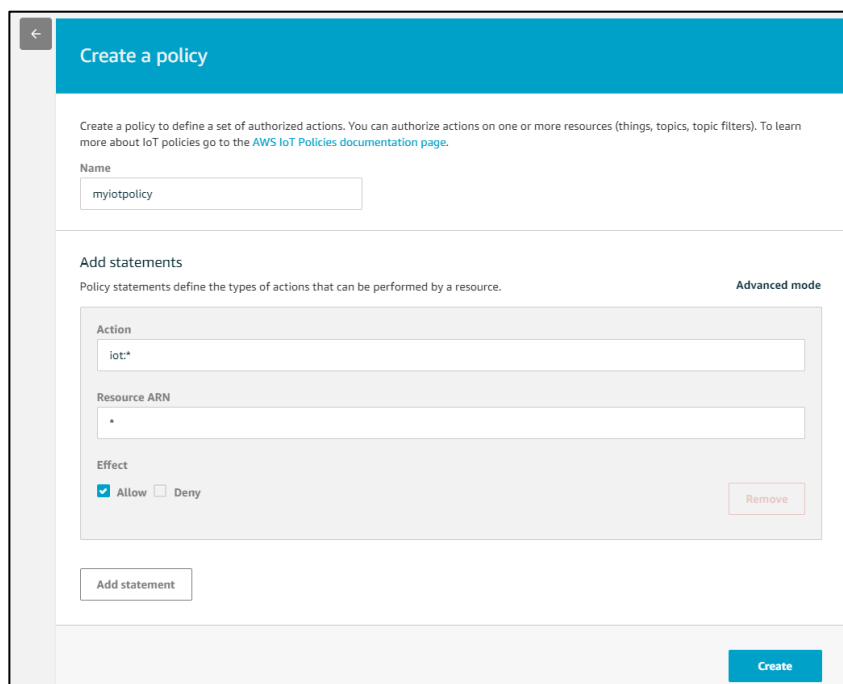




4. On the left, click on **Secure > Policies**. You will be able to look (and search) all of the policies attached to your account.



5. To create a new policy, click **Create**. On the “Create a Policy” page:



- a. Give the policy a **Name** (in this case myiotpolicy). Your name should reflect your security architectural objectives.
- b. Then assign the specific **Action**. In this case I am using “iot:*”, which means this policy will allow all IoT actions (i.e connect, publish).
- c. Finally specify which **Resource** that this policy applies to. To simplify things I use “*” meaning all resources. However, there are a complicated set of rules that let you constrain the resource based on certificates, connection types etc.

You can read about those rules here:

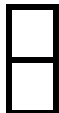
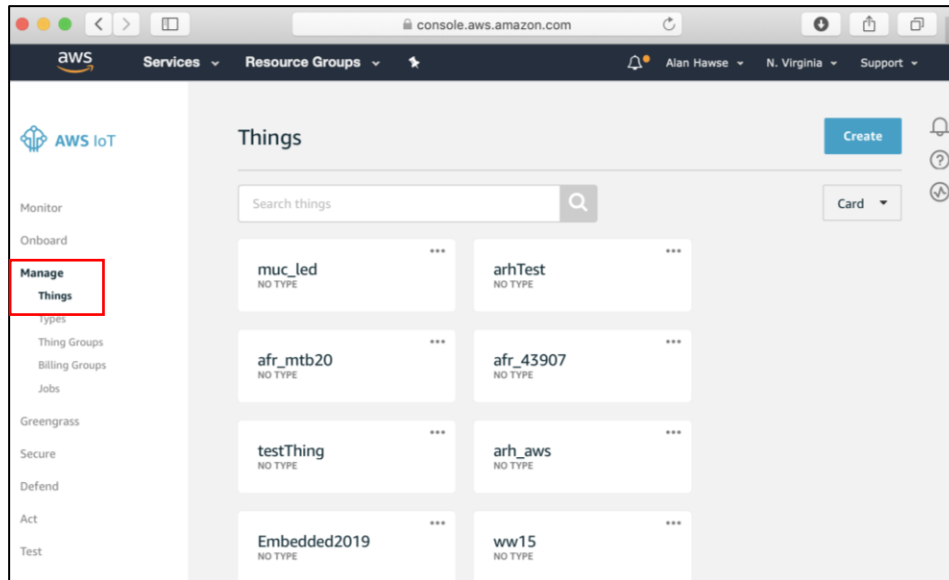
<https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html>

- d. Click **Create** at the bottom to make the policy.

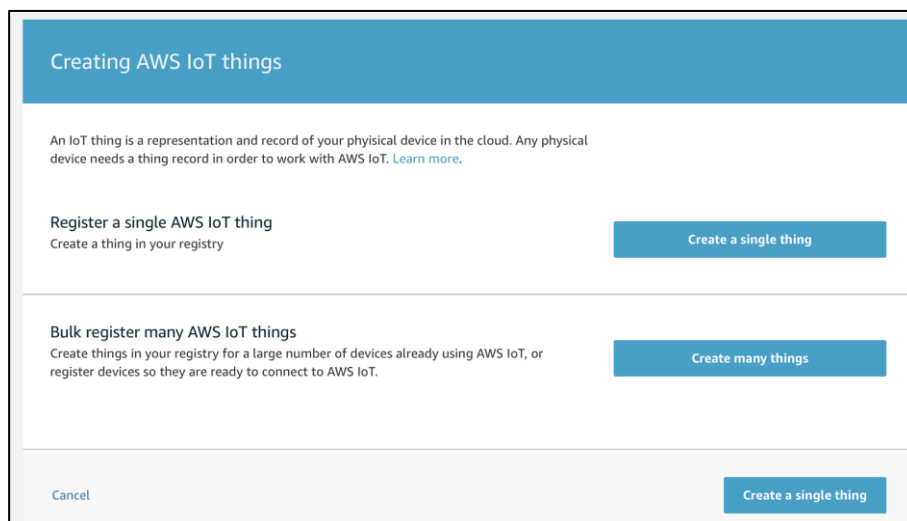
3.9.2 To create a thing:



1. Go back to **IOT Core** and select **Manage > Things**.



2. Once there, click **Create** to make a new thing.
3. On the “Creating AWS IoT things” page, choose **Create a single thing**.





4. Give it a unique name and click **Next**.

CREATE A THING

STEP 1/3

Add your device to the thing registry

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

Name

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	Value	
<input type="text" value="Provide an attribute key, e.g. Manufacturer"/>	<input type="text" value="Provide an attribute value, e.g. Acme-Corporation"/>	<div>Clear</div>
<div>Add another</div>		

Show thing shadow ▾

Cancel

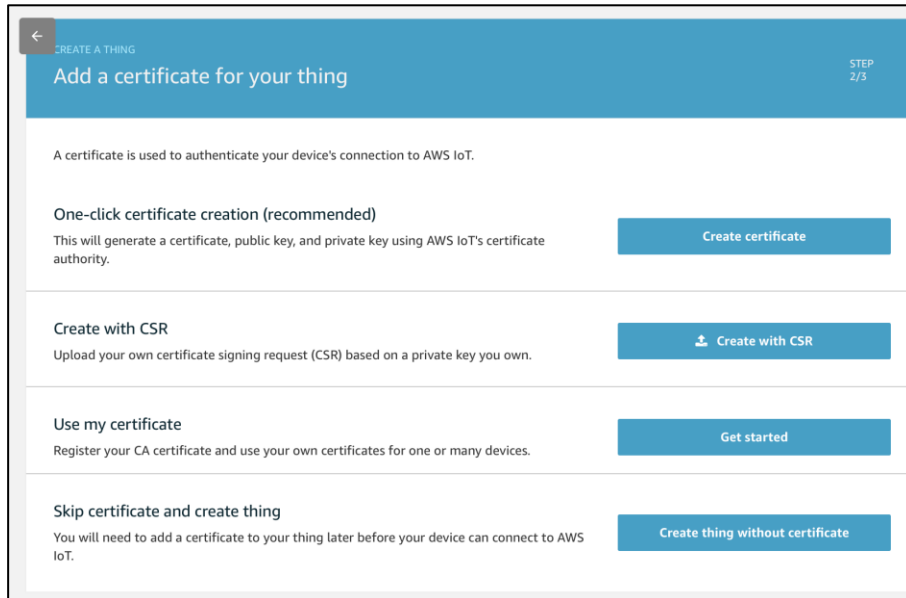
Back

Next

The security associated with a thing is controlled using X.509 certificates. Each thing should have a certificate attached to it. The easiest (and probably best) thing to do is let AWS create and manage certificates for you.



5. Click **Create certificate**.



← CREATE A THING STEP 2/3

Add a certificate for your thing

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.

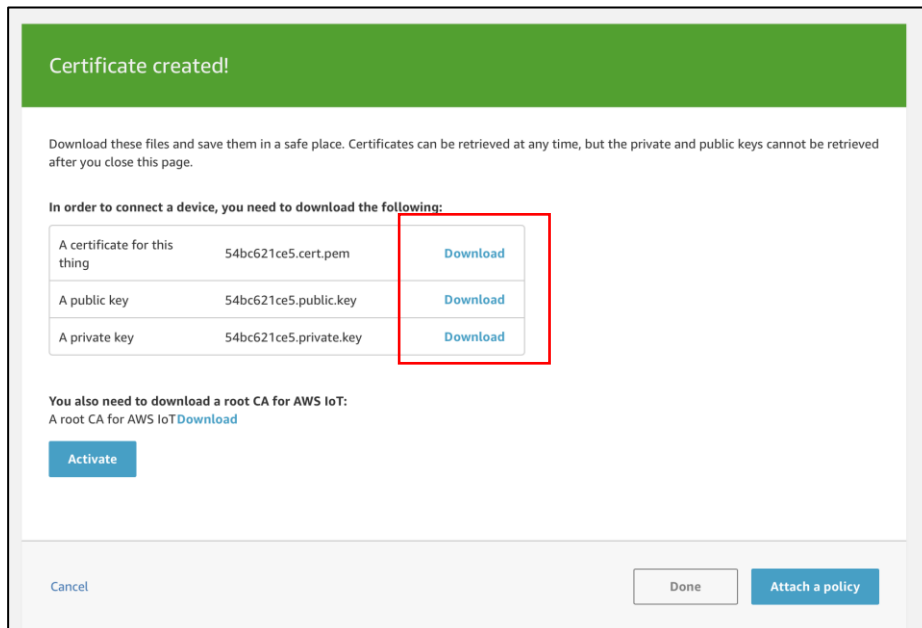
Create with CSR
 Upload your own certificate signing request (CSR) based on a private key you own.

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.



6. This will create an AWS signed certificate (public key), a public key and a private key. Now, you **MUST** download all three files from this screen as it will be the last time you have the opportunity. **Do this now!**



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	54bc621ce5.cert.pem	Download
A public key	54bc621ce5.public.key	Download
A private key	54bc621ce5.private.key	Download

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

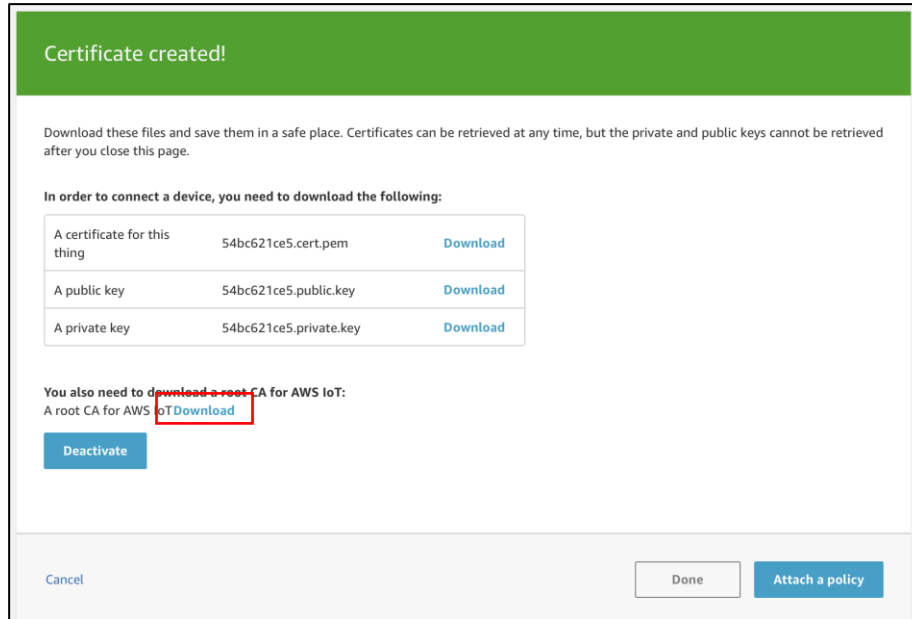
[Activate](#)

Cancel Done [Attach a policy](#)

Note: All three are required for every new certificate.



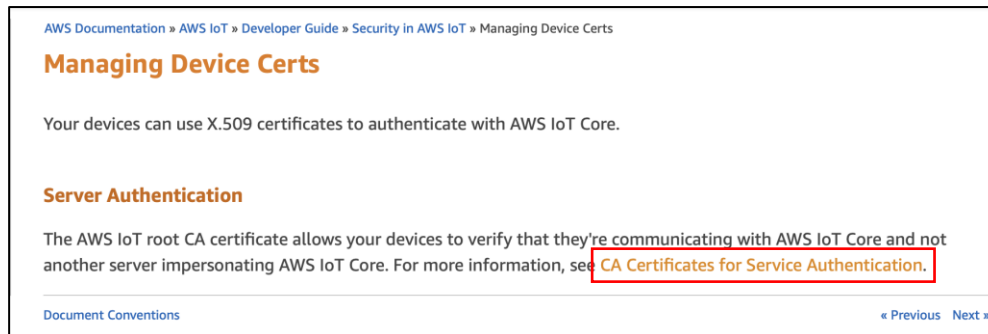
7. Also, click **Activate** to enable the certificate, and the page indicates that the Certificate was created.



When making a TLS connection, it is best to verify the certificate of the other side (Amazon). To do this, you need to include the AWS root certificate.

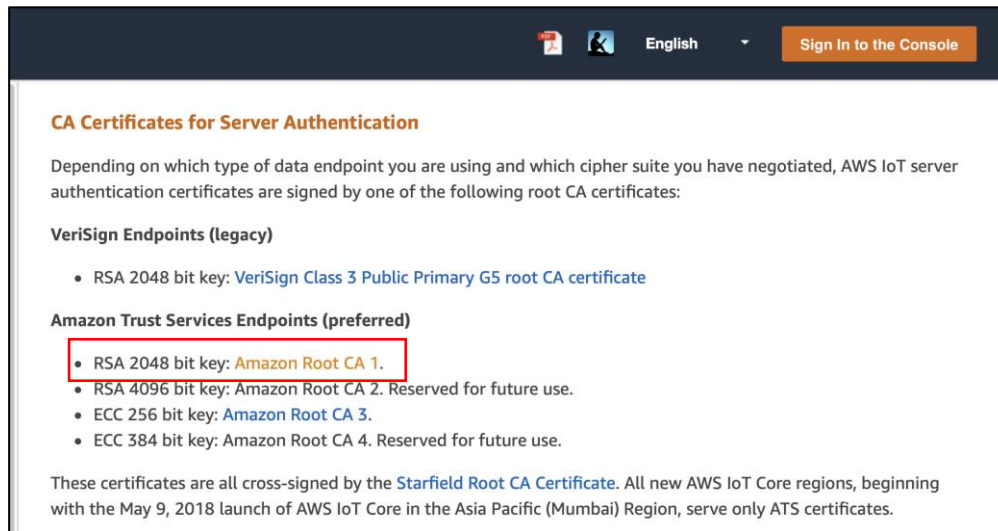


8. Right-click on the **Download** link to open the “Managing Device Certs” page in another window.
9. Click on “CA Certificates for Service Authentication” to view the certificates.

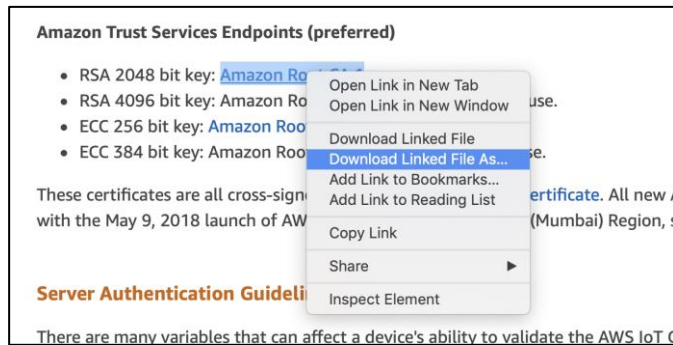




10. Right-click on the **RSA2048 Amazon Root CA1** link.



11. Save the file as appropriate for your web browser.



Note: You only need to download this once since it will be the same for any project that you create.

The certificate and keys for your Thing allow AWS to validate the identity of your thing. The root CA will allow your firmware to validate that the connection to AWS is legitimate (i.e. you didn't connect to an AWS imposter).



12. The next step in the process is to attach a policy (the one we created way back at the start). Click **Attach Policy**.

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	54bc621ce5.cert.pem	Download
A public key	54bc621ce5.public.key	Download
A private key	54bc621ce5.private.key	Download

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

[Deactivate](#)

Cancel
Done
Attach a policy



13. On the “Add a policy for your thing” page, search for your policy (if there is a long list). Click the check box, then click **Register Thing**.

CREATE A THING STEP 3/3

Add a policy for your thing

Select a policy to attach to this certificate:

☒ myIoTPolicy
 [View](#)

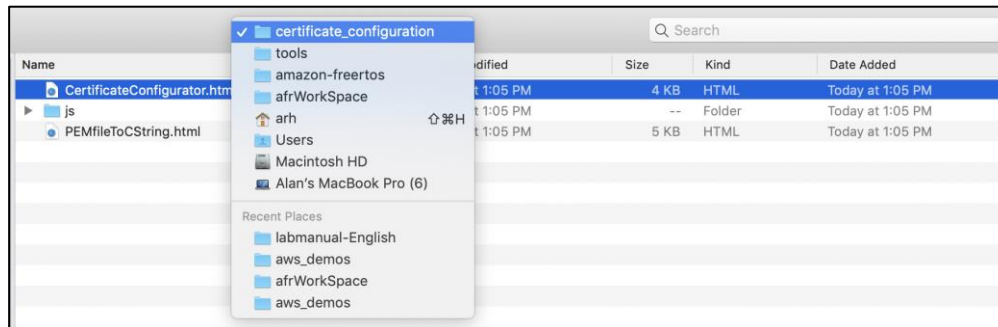
1 policy selected
Register Thing

3.10 Transform Keys Into “C”

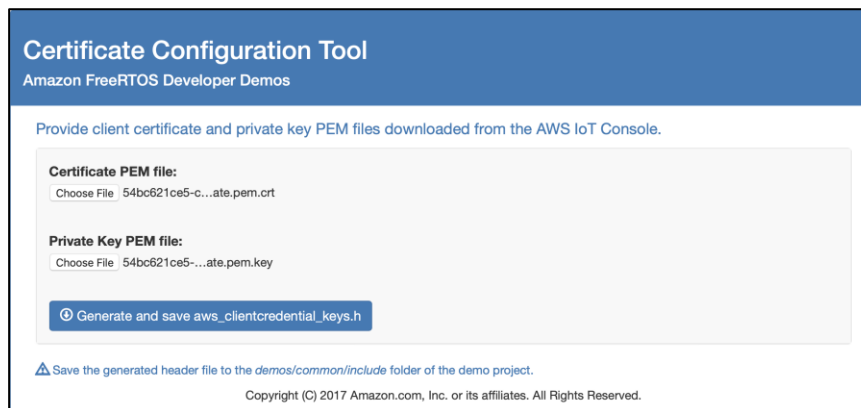
Once you have your certificates and keys you need to turn them into a format that is usable by your middleware. MbedTLS uses a string format. The easiest way to change your certificate into a C-File is to use the utility provided by Amazon FreeRTOS.



1. First, go to the “amazon-freertos/tools/certificate_configuration” directory and open the CertificateConfigurator.html in a web browser.



2. Select your Certificate and Private Key files. Then click **Generate and save...**



This will make a C-header file named “aws_clientcredential_keys.h” and download it to your PC in the amazon-freertos/demos/include directory (the path on the screen is wrong). This is just a normal C-header file with #defines for the keyCLIENT_CERTIFICATE_PEM, keyCLIENT_PRIVATE_KEY_PEM. And this file is named exactly the right thing for the Amazon FreeRTOS demo.

The file will look like this:

```
aws-keys — emacs aws_clientcredential_keys.h — 10
#ifndef AWS_CLIENT_CREDENTIAL_KEYS_H
#define AWS_CLIENT_CREDENTIAL_KEYS_H

#include <stdint.h>

/*
 * PEM-encoded client certificate.
 *
 * Must include the PEM header and footer:
 * "-----BEGIN CERTIFICATE-----\n"
 * "...base64 data...\n"
 * "-----END CERTIFICATE-----"
 */
#define keyCLIENT_CERTIFICATE_PEM \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDWjCCCAKGAwIBAgIWAQY9SFEqYUMZK13nNVHcP9lncTKMA0GCSqGSIb3DQEB\n" \
"CwUAME0xSzB3BgNVBAsMQkFtYXpvaXB3ZWlgaU2VydmljZXMQTz1BbW6b24uY29t\n" \
"IEluYy4gTD1ZW0dGx1IFNUPVdhc2hpbmd0b24gQz1VUzAeFw0xOTExODQy\n" \
"NTdaFw00TEYmZmZU5NTlaMB4xHDAaBgNVBAMME0FXUyBjb1QgQ2VydG1maWNh\n" \
"dGUwggEiMA0GCSqGSIb3DQEBAAQAAIBDwAwggEKAoIBAQCUIvafPz0bVqUf1J\n" \
"JYrZc6zDuaC+R+dA0ILBS7KlmonTAf18p8rt3zDwQnYCr+90BzX+IuN+QvXN+wi8\n" \
"qVEF1HecZNYUOLK0yW9IOq3X82e1/A7Vz90rx55ELYEBV/lvX/9x/1eyPyMQqxAj\n" \
"HU6/UHiVz48p691Qf4q1WKHC9KVGtKwSpf/gQcokxkF9SjEYNTusdrirxCeNjA\n" \
"QdUYq/9Yr4rsN61v6KbBy0q/Xq1tsXKEjxvbiUNrZfvQXSCacVx/n0uKyQ1TF2ae\n" \
"/OZ8slj+uv1lgI5aaGep+haBbTF36u/LL3RivucHQTUPutUJ4f2+cR71HZNZ655p\n" \
"mFo/AgMBAAGjYDBEMB8GA1UdIwQYMBaAFG2Mj+2L1qJaX6x2zRmErLQ91w/MB0G\n" \
"A1UdDgQWB8T2ZYnmFW31qcUyWUmc77LV9506zAMBgNVHRMBAf8EAjAAMA4GA1Ud\n" \
"DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAQCAQEAWS321YPxa+cvHu0RdQrLdpVE\n" \
"arhPTCj6QS/VyJ19sKi2KUqdiUBh/280ZxE3cRzYZ22QIK6f2PwNvmaFwN8qi5IAe\n" \
"cw7HQWqPSWi0gDLz1BYip5AUoq10/6h++LuMrd7Z1GdiG0V2QuT1ygdrd7qP1PA\n" \
"1Y5KNH00Iv4aLzuN1YKR6crpGnD80Y+incVmVuHZZF21jk+8sCx0Dw9MtJN1JkL\n" \
"yeMBbezamg4KRL602PLpDubLGgag/P1G7kHEyScwxOvwyE5Tr+qbV1Gqn4aW3pmm\n" \
"QDf49XCEJafq534M3klutUfND6pG/LKujQo4dncw4E8rZPu0eHQNvH0a8JGE1w=\n" \
"-----END CERTIFICATE-----"
```

This file will also work just fine in Mbed OS.

3.11 PEMfileToCString.html

You can also use the AWS Tool called “PEMfileToCString.html” to convert any (single) PEM file into a C-String.



1. Go to the “amazon-freertos/tools/certificate_configuration” directory and open the PEMfileToCString.html file in a web browser.
2. For example, to convert the Root Certificate for Amazon, can click **Choose File**.

PEM to C String Conversion Tool

Use with Amazon FreeRTOS Developer Tests

Select a PEM key or PEM certificate file to be converted into the format required by aws_clientcredential_keys.h

PEM Certificate or Key:

Choose File no file selected

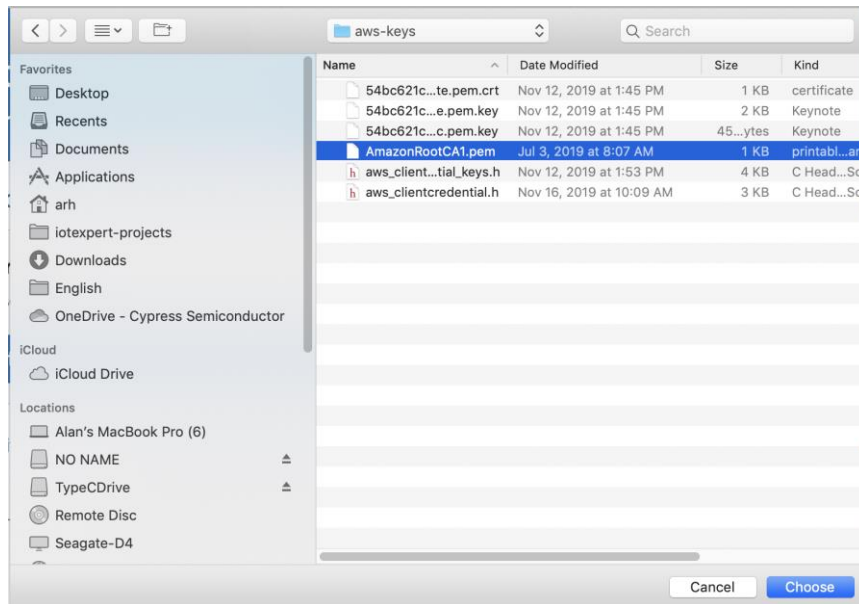
Display formatted PEM string to be copied into aws_clientcredential_keys.h

Copy the key or certificate into the appropriate variable in tests/common/include/aws_clientcredential_keys.h file of the test project.

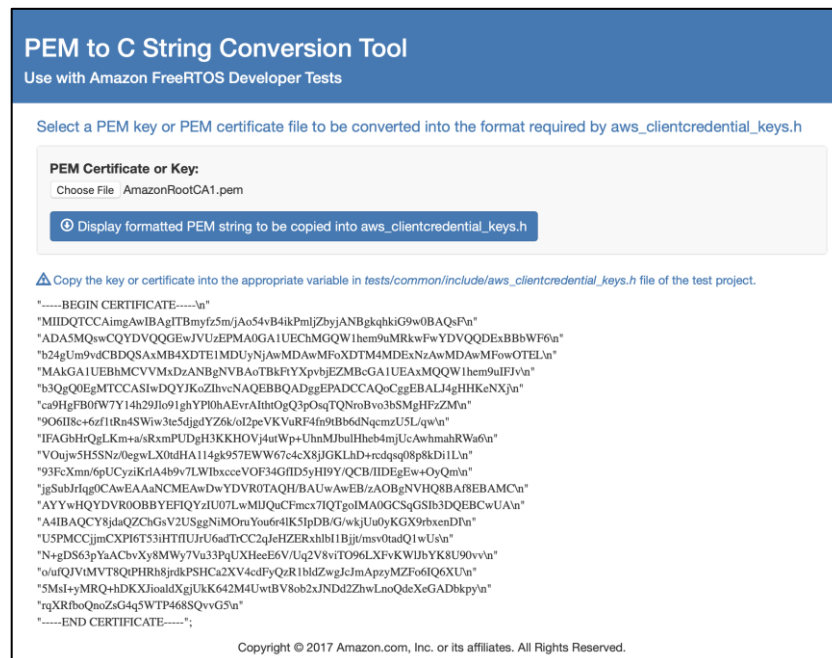
Copyright © 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.



3. Select the AmazonRootCA1.pem file and click **Choose**.



4. Then click **Display formatted PEM string...**



- ```

/*
 * PEM-encoded Sign-Join-In-Time Registration (JITR) certificate (optional).
 *
 * If used, must include the PEM header and footer:
 *
 * "-----BEGIN CERTIFICATE-----\n"
 * "...base64 data...\n"
 * "-----END CERTIFICATE-----"
 */

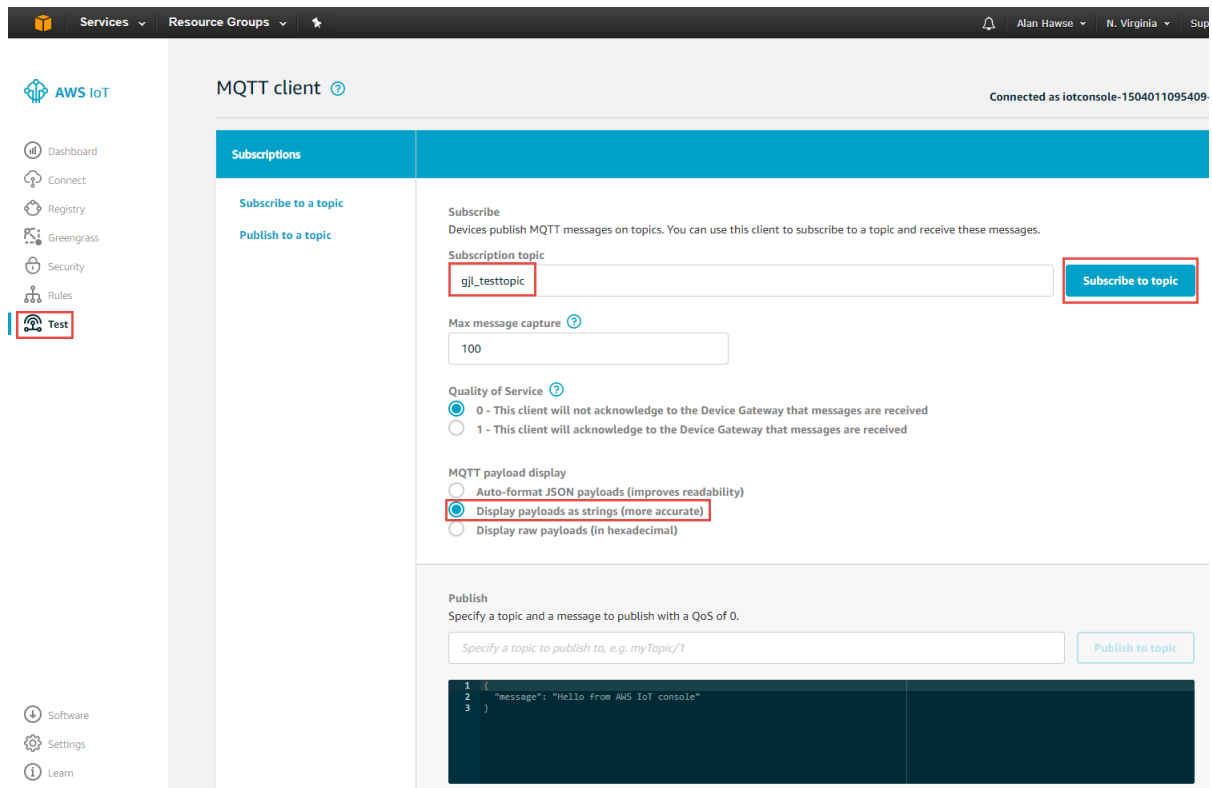
#define keyJITR_DEVICE_CERTIFICATE_Authority_PEM \
 "-----BEGIN CERTIFICATE-----\n"

/*
 * "MIIDDTCCAImAwIBAIIbmVzfMzEwIAo5A/B4ikPmLiZbyiANBkzhkhiG9w0BA0sFvN"
 * "ADA5M0sYcwDQ0V00G6eJXUzE2MDUwAUEiAECMGOWIhem9MURk0eFzVD00DExBbbfWELn"
 * "b24Um9yZCBDO5A5M0sYcwDQ0V00G6eJXUzE2MDUwAUEiAECMGOWIhem9MURk0eFzVD00DExBbbfWELn"
 * "MAkGA1UEBmVCMVczdA0N0BnVA0BAkTFYXqvbFE2MzBGAUUEiAECMGOWIhem9MURk0eFzVD00DExBbbfWELn"
 * "b30Q0eMTCVCCzDQ0V00G6eJXUzE2MDUwAUEiAECMGOWIhem9MURk0eFzVD00DExBbbfWELn"
 * "ca9hAFB0fW714h291091u9YlP8AeYrAit1th003p0aStOnRvbo3b5MhKcFzMN"
 * "9061T8c-6z1r148S4Wj3t5ed1q4dYz6K10ZpeVkuVRf4n9t86BdMazcmU5L/qwN"
 * "IFAGbHr4dKmta+sRexMPudh3K9KH9v4i4utUw+UhnM1UwLthbeH4dUcAwmAhRwa6Gn"
 * "Vou1uSH5SN3/8eawXk4h14d4K357EwM67CdXh8JCGkHnd+cdos+o8B8kD1L1"
 * "Xm1u0CvYkIkd4b57V1Wb3cxv9F34G1FDSyht19Y0CB/TIDEf+e+o0mN"
 * "Jo5ubJrL0o9CAEAA5NCAeYwDVR8T0A0/BAluAwMEFz/ZA0BNH00B8A7BERACmN"
 * "AYYhWEDVR80BEEFF10zU87Lw4U7qCqFmcXj/BQtoIAm8Q5SaG531300BECuAJAN"
 * "ADAIBAQCY81d04ZChGSvZU5SaQNM0Jm0Yur6Ar4K5tDB/G/kw1u0YkGX9rxbenDtN"
 * "U5PMCC1imXP16T53H7fUJ3rU6adTCrC2a3eHZErxh1B18i1t/msv0d401w5N"
 * "N+o4p65a3YAcBv8YwMw/V73p3UAlHeE6G/U6V2Y81vT096LXFkVw1BYK9U89vrvN"
 * "o/uf0JTVJTMvBT01PHR81h1dRPSHCA2XV4dF6Zv2r1bL2wGJicMazpW706106XJN"
 * "5MsT+VMR0+HDK0J1oadXg1Uk642M4UwTBV80b2q3JNDd2ZhwLno0dExG6AdbkpVn"
 * "raXRFbo0Noz43J5adP7468S0vG5N"
 *
 * -----END CERTIFICATE-----
 */

```

The AWS website has an MQTT Test Client that you can use to test publishing and subscribing to topics. Think of it as a terminal window into your message broker, or as a generic IoT *thing* that can publish and subscribe.

1. Select "Test" from the panel on the left of the screen.
2. Enter a topic that you want to subscribe to such as **<your\_initials>\_testtopic** in the "Subscription topic" box.
3. Select "Display payloads as strings", and click on **Subscribe to topic**.
4. Make sure to put your initials or some other unique string in the topic if you are using the class AWS account. If not, you may see messages from someone else publishing to the same topic.

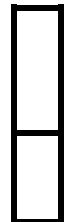


The screenshot shows the AWS IoT MQTT client interface. On the left sidebar, the 'Test' option is selected. The main area is titled 'MQTT client' and shows a 'Subscriptions' section with a 'Subscribe to topic' button. The 'Subscription topic' field contains 'gjl\_testtopic'. Below this, the 'Max message capture' is set to 100. The 'Quality of Service' is set to 0. The 'MQTT payload display' section has three options: 'Auto-format JSON payloads (improves readability)', 'Display payloads as strings (more accurate)' (which is selected), and 'Display raw payloads (in hexadecimal)'. The 'Publish' section shows a message being published to the topic 'gjl\_testtopic/1'. The message content is displayed in a code block as follows:

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

### 3.12.2 Publishing to a Topic from the Test Client

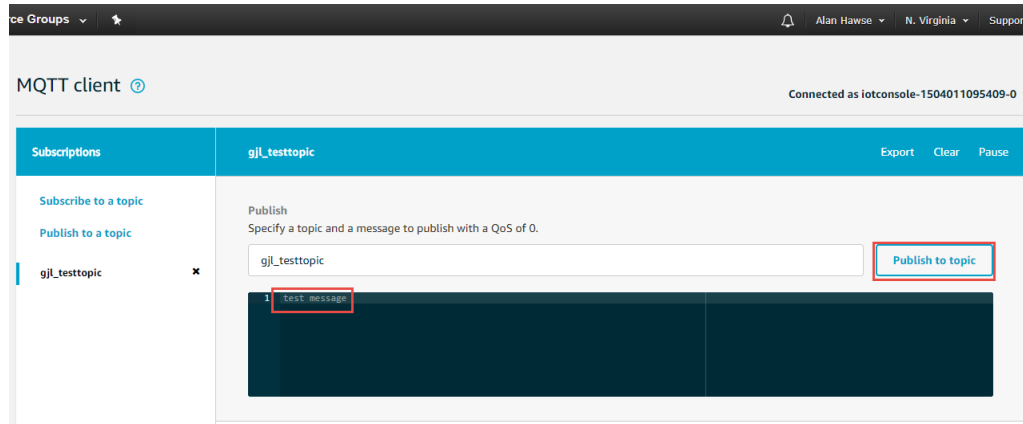
Now that I am subscribed to a topic, I can publish messages to that topic from another instance of the MQTT test client.



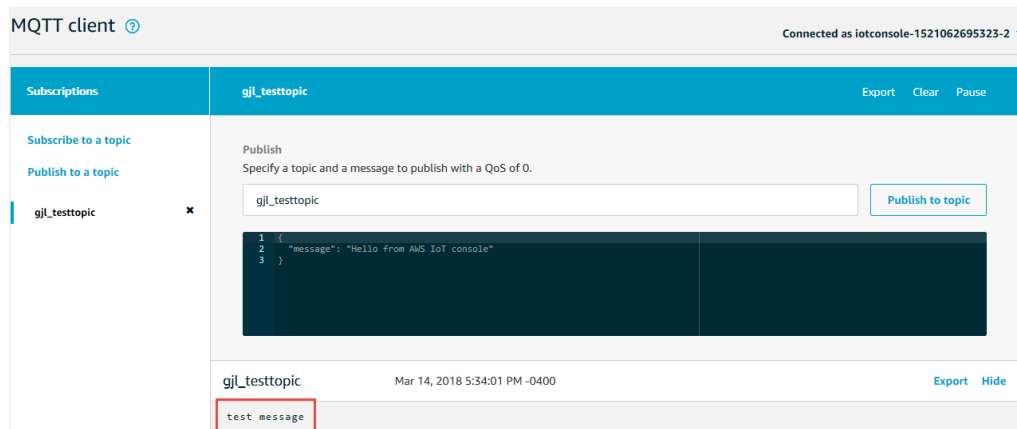
1. First, open another web browser tab, login to the AWS account, and go to the Test page. (Note: team up with another student if you can so that one person subscribes and the other publishes to the same topic).
2. Scroll down to the "Publish" section of the page and fill in the name of the topic that you subscribed to earlier. The name must be exactly the same (including case).



3. Then type in your message and press "Publish to topic". You can see in the box below I sent "test message".



4. Now, go back to the tab with the subscription and see that the published message was sent to the subscriber.



### 3.13 GreenGrass

AWS IoT GreenGrass is available at <https://aws.amazon.com/greengrass/>

From the Webpage, AWS IoT GreenGrass is described as follows:

AWS IoT Greengrass seamlessly extends AWS to edge devices so they can act locally on the data they generate, while still using the cloud for management, analytics, and durable storage. With AWS IoT Greengrass, connected devices can run AWS Lambda functions, Docker containers, or both, execute predictions based on machine learning models, keep device data in sync, and communicate with other devices securely – even when not connected to the Internet.

The following image (also from the AWS website) shows how devices can be used in the AWS IoT GreenGrass system:

