

Chapter 3: AWS MQTT Flow

After completing this chapter, you will understand how to use the AWS flow.

3.1	MQTT	1
3.2	AMAZON WEB SERVICES	2
3.3	AWS IOT INTRODUCTION	3
3.4	AWS IOT RESOURCES.....	4
3.4.1	THING.....	4
3.4.2	CERTIFICATE	4
3.4.3	POLICY	4
3.5	THING SHADOW	4
3.6	TOPICS.....	6
3.7	DEVICE SHADOW TOPICS	7
3.8	CREATING THINGS	8
3.9	TRANSFORM KEYS INTO “C”	17
3.10	PEMFILETOCSTRING.HTML	18
3.11	TEST CLIENT	21
3.11.1	SUBSCRIBING TO A TOPIC FROM THE TEST CLIENT	21
3.11.2	PUBLISHING TO A TOPIC FROM THE TEST CLIENT	22
3.12	GREENGRASS.....	23

3.1 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 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.

3.2 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):

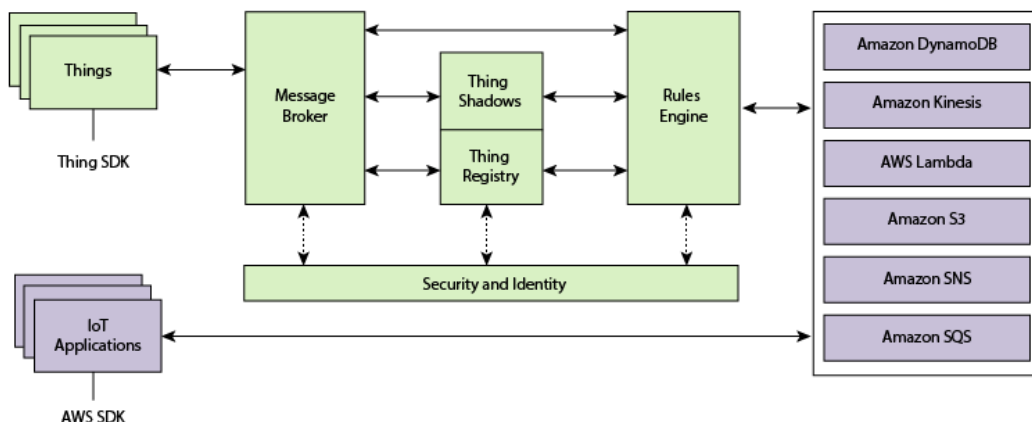
- [AWS IoT](#): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud ([EC2](#)/[EC3](#)): A virtualized compute capability, basically Linux, Windows etc. servers that you can rent.
- [Amazon Lambda](#): A Cloud service that enables you to 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 a MQTT messages).

- [Amazon Kinesis](#): A platform to stream and analyze "massive" amounts of data. This is the plumbing for AWS IoT.

3.3 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.



3.4 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.

3.4.1 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).

3.4.2 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.

3.4.3 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"
    }
  ]
}
```

3.5 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"}}}
```

3.6 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.

3.7 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.

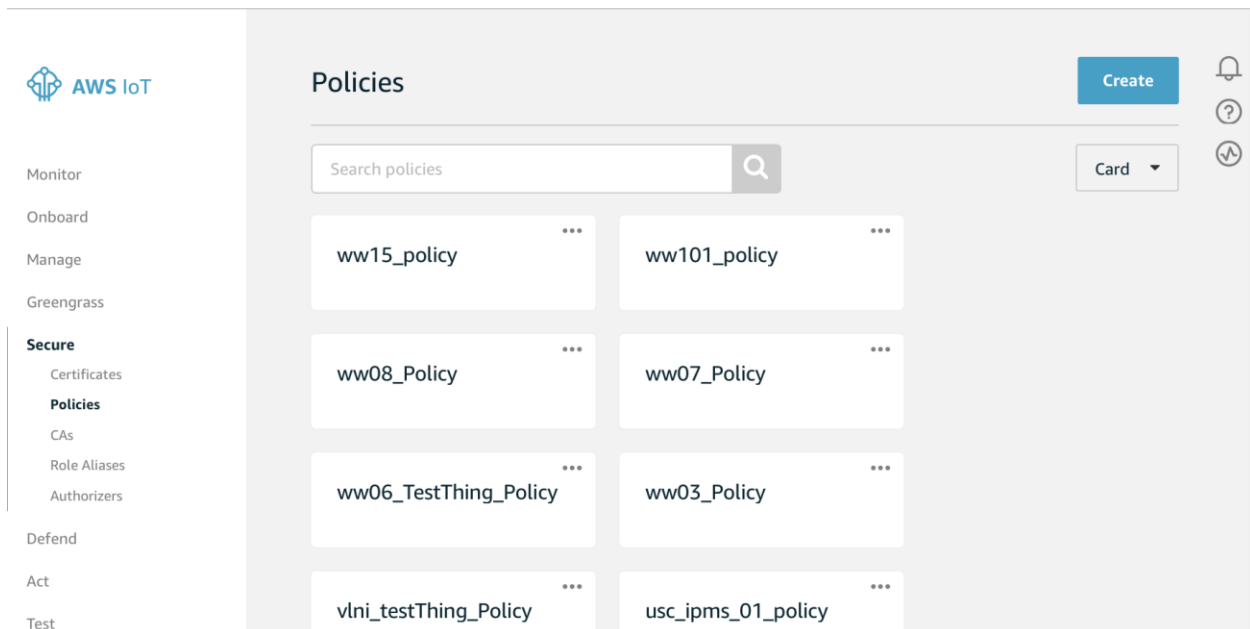
If you are using the MQTT test server to subscribe to topics, you can use "#" as a wildcard at the end of a topic to subscribe to multiple topics. For example, you can use "\$aws/things/theThing/shadow/#" to subscribe to all shadow topics for the *thing* called "theThing".

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.

3.8 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.

To create a policy go to the AWS IoT console and click on **Secure > Policies**. You will be able to look (and search) all of the policies attached to your account. To create a new policy click **Create**.



Give the policy a Name (in this case IoTPolicy). Your name should reflect your security architectural objectives. Then assign the specific "actions". In this case I am saying that this policy will allow all IoT actions i.e connect, publish. Finally specify which Resource that this policy applies to. To simplify things I say "*" 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>

Click **Create** to make the policy.

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

Add statements

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

Action

Resource ARN

Effect

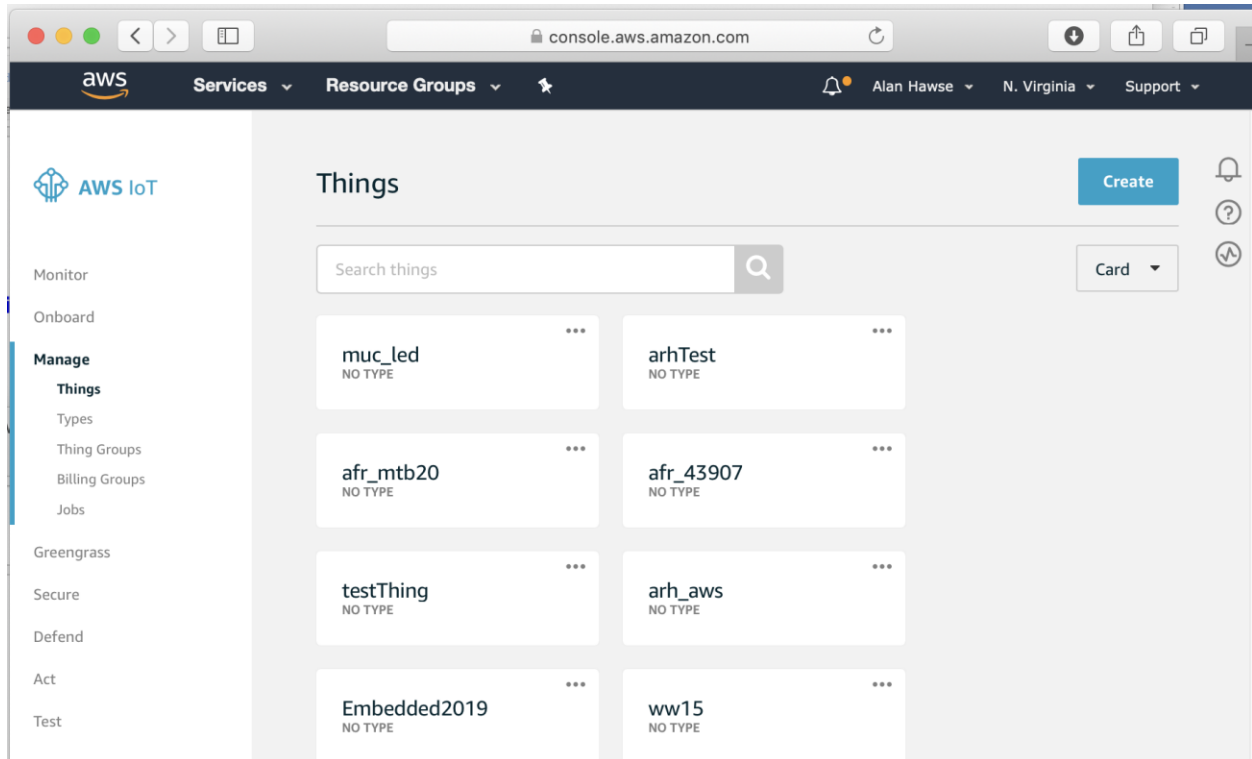
☒ Allow ☐ Deny

Remove

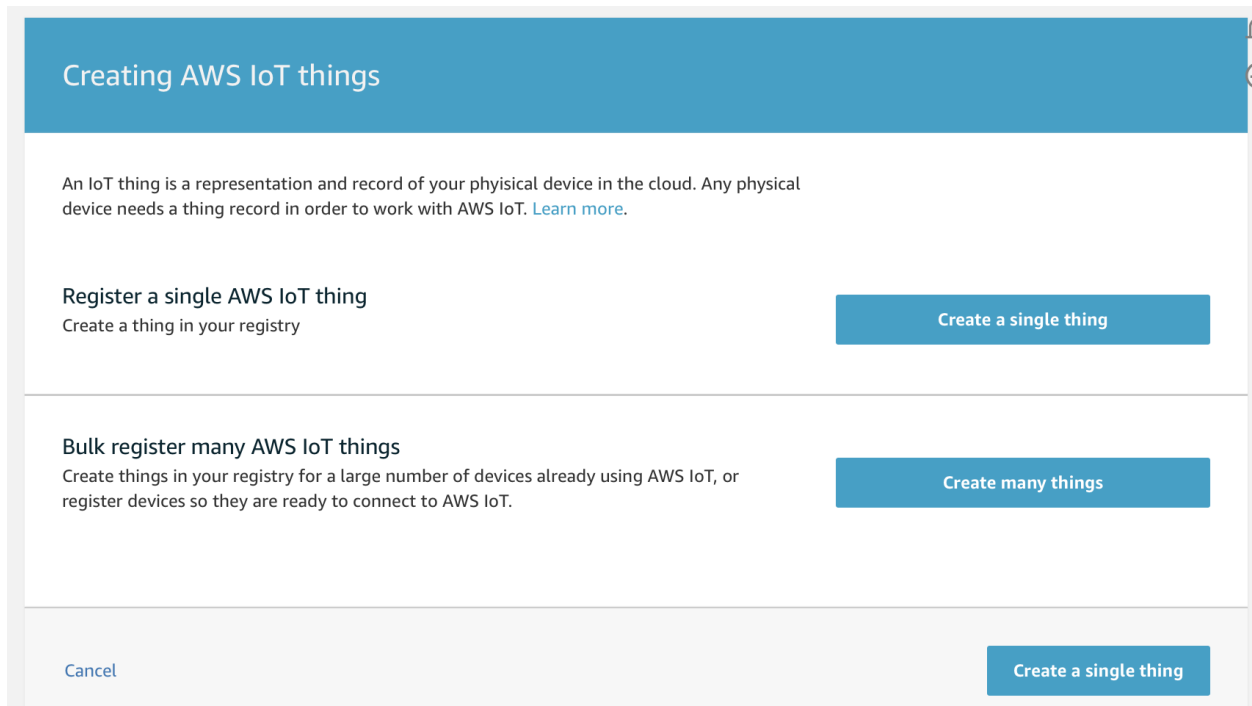
Add statement

Create

Now you need to go to “Manage > Things”. Once there, click **Create** to make a new thing.



Choose “Create a single thing”.



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>		

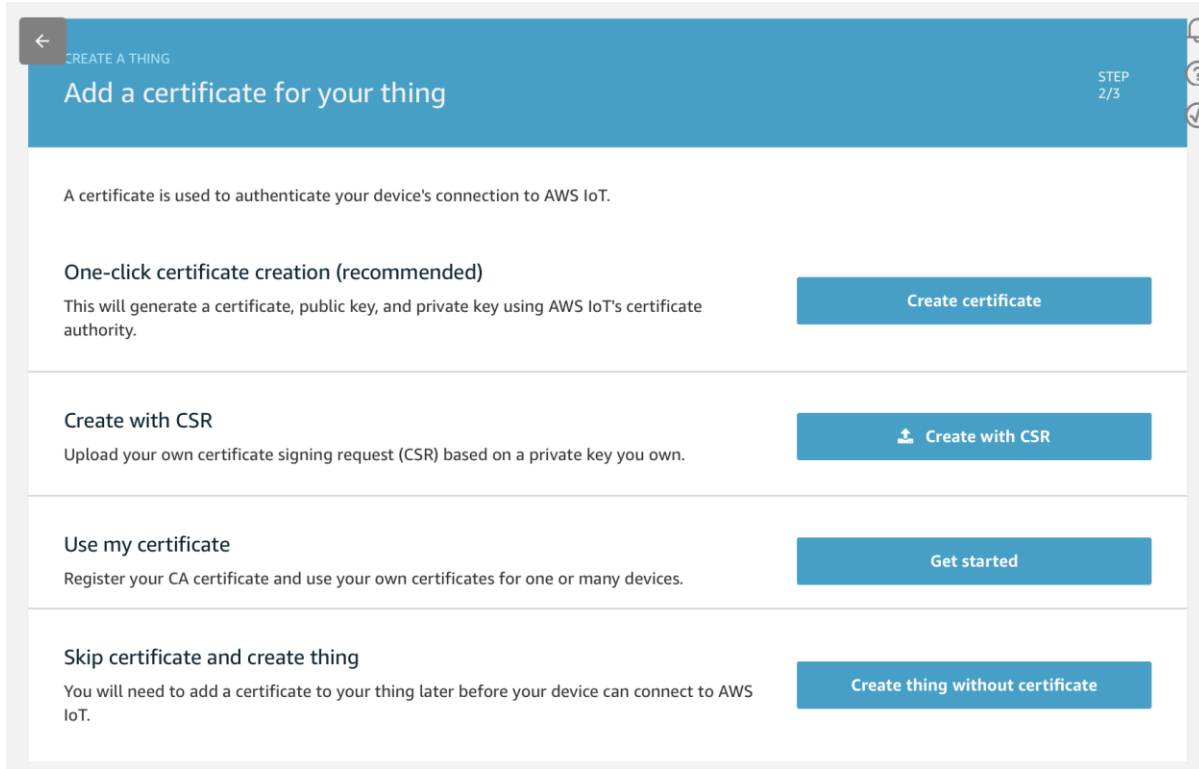
Show thing shadow ▾

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. Click **Create certificate**.



The screenshot shows the 'Add a certificate for your thing' page in the AWS IoT console. The page has a blue header with a back arrow, 'CREATE A THING', and 'STEP 2/3'. Below the header, a message states: 'A certificate is used to authenticate your device's connection to AWS IoT.' There are four options, each with a description and a button:

- One-click certificate creation (recommended)**: This will generate a certificate, public key, and private key using AWS IoT's certificate authority. Button: **Create certificate**
- Create with CSR**: Upload your own certificate signing request (CSR) based on a private key you own. Button: **Create with CSR**
- Use my certificate**: Register your CA certificate and use your own certificates for one or many devices. Button: **Get started**
- Skip certificate and create thing**: You will need to add a certificate to your thing later before your device can connect to AWS IoT. Button: **Create thing without certificate**

This will create an AWS signed certificate (public key), a public key and a private key.

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](#)

Now, you MUST download these files from this screen as it will be the last time you have the opportunity. **Do this now!**

Click **Activate** to enable the certificate.

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](#)

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. You can get this from the **Download** button. Right click and open this in another window. Then click **CA Certificates for Service Authentication**.

[AWS Documentation](#) » [AWS IoT](#) » [Developer Guide](#) » [Security in AWS IoT](#) » Managing Device Certs

Managing Device Certs

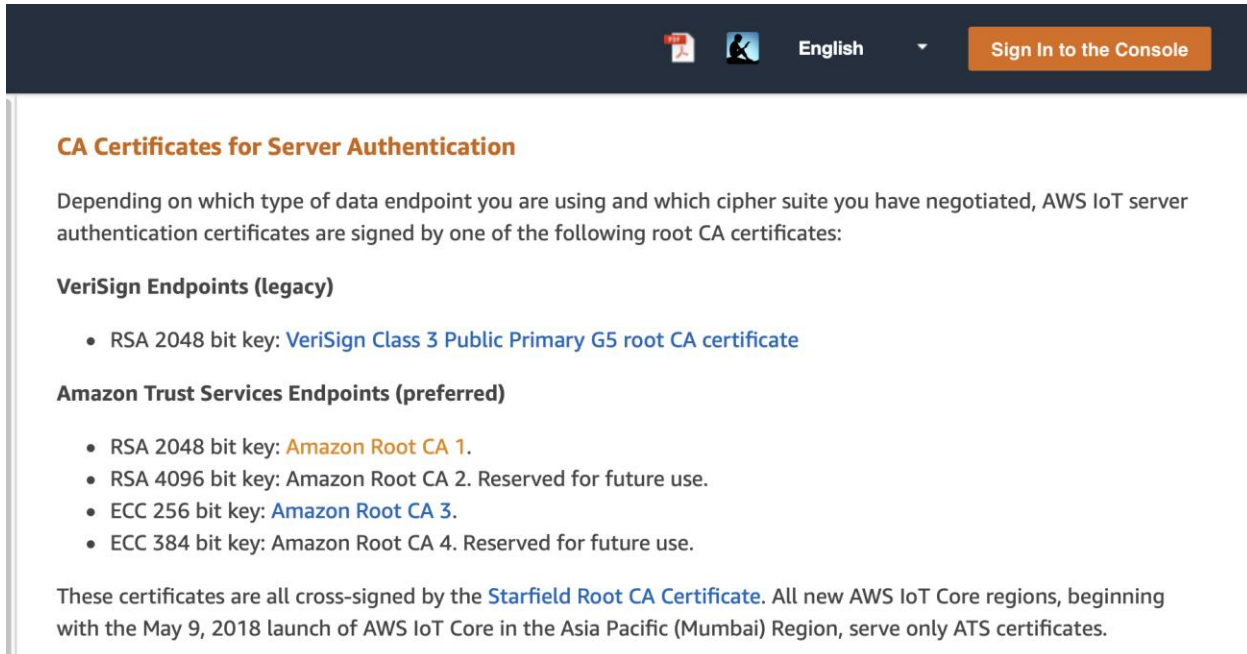
Your devices can use X.509 certificates to authenticate with AWS IoT Core.

Server Authentication

The AWS IoT root CA certificate allows your devices to verify that they're communicating with AWS IoT Core and not another server impersonating AWS IoT Core. For more information, see [CA Certificates for Service Authentication](#).

[Document Conventions](#)[« Previous](#) [Next »](#)

You want to download the RSA2048 Amazon Root CA1.



CA Certificates for Server Authentication

Depending on which type of data endpoint you are using and which cipher suite you have negotiated, AWS IoT server authentication certificates are signed by one of the following root 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.

These certificates are all cross-signed by the [Starfield Root CA Certificate](#). All new AWS IoT Core regions, beginning with the May 9, 2018 launch of AWS IoT Core in the Asia Pacific (Mumbai) Region, serve only ATS certificates.

Right-click and save the file as appropriate for your web browser.

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.

These certificates are all cross-signed by the [Starfield Root CA Certificate](#). All new AWS IoT Core regions, beginning with the May 9, 2018 launch of AWS IoT Core in the Asia Pacific (Mumbai) Region, serve only ATS certificates.

Server Authentication Guidelines

There are many variables that can affect a device's ability to validate the AWS IoT C

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](#)

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:

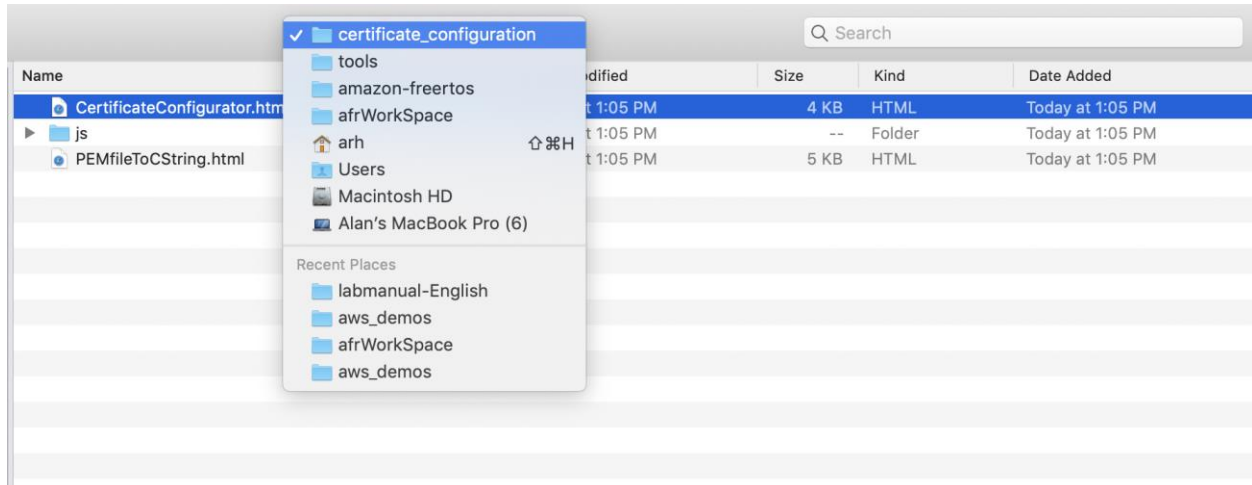
☒ myIoTPolicy [View](#)

1 policy selected

[Register Thing](#)

3.9 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. First, open: `amazon-freertos/tools/certificate_configuration/CertificateConfigurator.html` in a web browser.



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

Certificate Configuration Tool

Amazon FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

Certificate PEM file:
 54bc621ce5-c...ate.pem.crt

Private Key PEM file:
 54bc621ce5-...ate.pem.key

Save the generated header file to the `demons/common/include` folder of the demo project.

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

This will make a C-file (and download it to your PC). The file will be called “aws_clientcredential_keys.h” 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" \
"MIIDWjCCAkKgAwIBAgIWAOKY9SFEqyUMZK13nNVHcP9lncTKMA0GCSqGSIb3DQEBA\n" \
"CwUAME0xSzBjBgNVBAsMQkFtYXpvcbiBXZWIGU2VydmljZXMGTz1BbWf6b24uY29t\n" \
"IEIuYy4gTD1TWf0dGx1IFNUPVdhc2hpbmd0b24gQz1VUzAeFw0xOTExMTIxODQy\n" \
"NTdaFw00TEYmZyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBjb1QgQ2VydG1maWNh\n" \
"dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCuIvafPz0bVqUff1JJ\n" \
"JYrZc6zDuaC+R+dA0ILBS7KLmonTaf18p8rt3zDwQnYCr+90BzX+IuN+QvXN+wi8\n" \
"qVEF1HecZNYU01K0yW9I0q3X82e1/A7Vz90rx55E1YEBV/1vX/9x/1eyPyMQqxAj\n" \
"HU6/UHiVz48p69lQf4q1WKHC9KVGtkWSpf/gQcokxkF9SjEYZNTusdrirxCeNjA\n" \
"QdUYq/9Yr4rsN61v6KbBy0q/Xq1tsXKEjxvbiUNrZfvQXSCacVx/n0uKyQ1TF2ae\n" \
"/OZ8slj+uvllgI5aaGep+haBbTF36u/LL3RivuchQTUPutUJ4f2+cR7iHZNZ655p\n" \
"mFo/AgMBAAGjYDBEMB8GA1UdIwQYMBaAFG2MJ+2L1qJaX6x2zRmERrLQ91w/MB0G\n" \
"A1UdDgQWB8T2ZYNmFW31qcUyWUQmc77LV9506zAMBgNVHRMBAf8EAjAAMA4GA1Ud\n" \
"DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCQAQEAWS321YPxa+cvHu0RdQrLdpVE\n" \
"aRhpTCJ6QS/VyJI9sKi2KUqdiUBh/280ZxE3cRzYZ22IK6f2PWnVmaFWn8qi5IAe\n" \
"c7WHQwWPSWI0gDLz1BYip5AUoql0/6h++LuMrd7Z1GdiG0V2QuT1ygdrrd7qP1Pat\n" \
"1Y5kNH00oIv4aLzuN1YKR6crpGnD80Y+incVmVuHZZF21jk+8sCx0Dw9MtJN1JkL\n" \
"yeMBbezamg4KRL602PLpDubLGgag/PlG7kHEySCwx0vwyE5Tr+qbV1Gqn4aW3pmm\n" \
"QDf49XCEJafq534M3klutUfND6pG/LKujQo4dncw4E8rZPu0eHQNVHoa8JGE1w==\n" \
"-----END CERTIFICATE-----"
```

This file will also work just fine in Mbed OS.

3.10 PEMfileToCString.html

You can also use the AWS Tool called “[PEMfileToCString.html](https://amazon-freertos/tools/certificate_configuration/PEMfileToCString.html)” to convert any (single) PEM file into a C-String. In a browser, open: amazon-freertos/tools/certificate_configuration/PEMfileToCString.html

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:

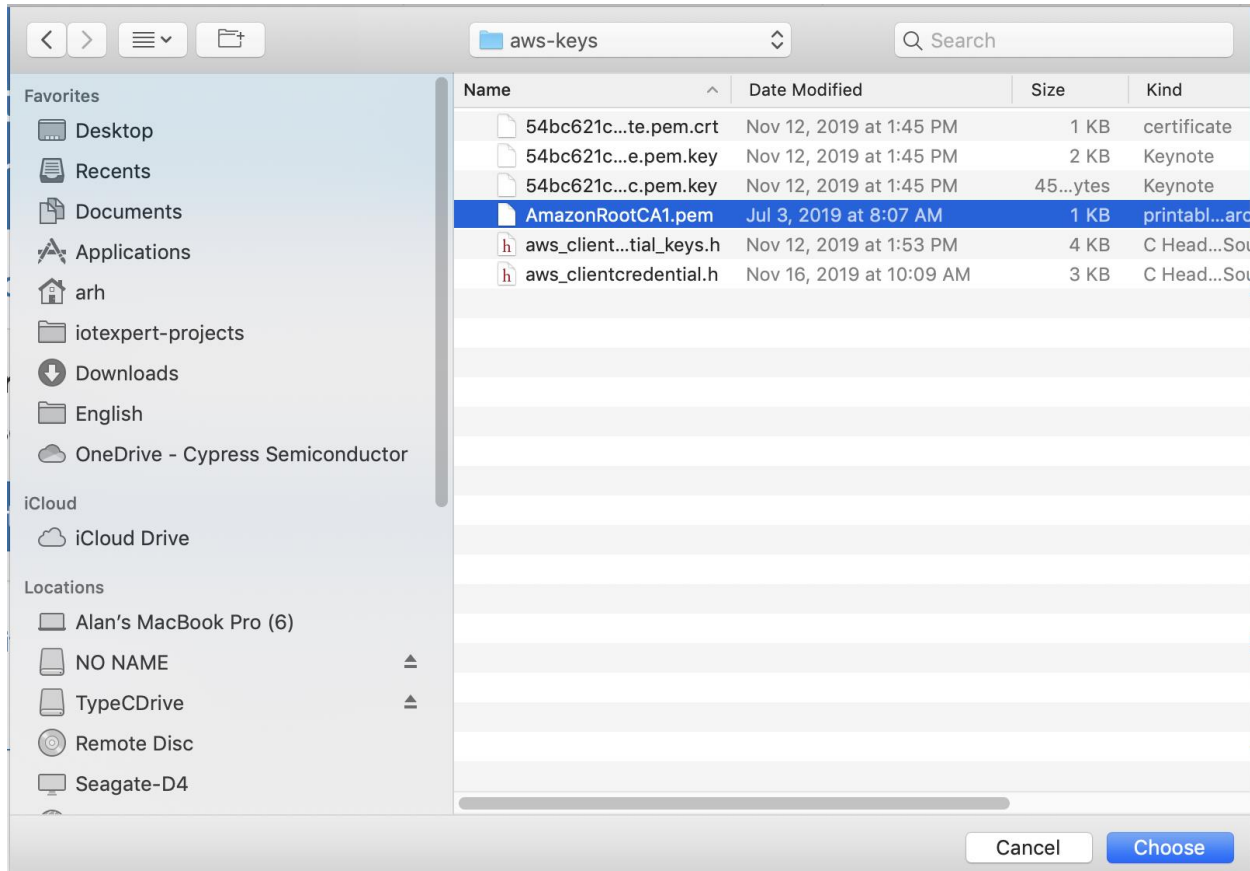
no file selected

 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.

For example, to convert the Root Certificate for amazon you can click **Choose File** and select the AmazonRootCA1.pem.



Then click **Display formatted...**

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 AmazonRootCA1.pem

④ 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.

-----BEGIN CERTIFICATE-----
"MIIDQTCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n"ADAM5QSCaimgQYDVQGEwJVUzEPMA0GA1UEChMGMGQW1hem9uMRkwFwYDVQDEeBBBWF6\n"b24gUm9vdCBDQ05AaM4XDTE1MDU1j2pAwMDAwMFoXDTM0MDE4NzAwMDAwMFowOTEL\n"MAkGA1UEBhMCVVMxZDANBgNVBAoTBkFtYXpjb2ZMCGA1UEAxMQQW1hem9uIFJv\n"b3QgQ0E0E0MTCcASiWdDQYJKoZIhvcNAQEBBQADggEPADCCAQcCgGEBALJ4gHHK\n"eNXj\n"ca9Hg6R0fW7Y14h29Jl01ghYPl0hAEvraltht0Kv3pOsqTQNb0v30BSMgHfZ\n"9061f8c+6z1rtn4SWin3te5d9gYZ6k/0l2peVqV3dtn6dNqcmz5U/qw\n"IFAGbHrQgLkm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUeAwchmahR\n"WA6\n"VOujw5X5SNz0egwLX0tdHA114g957WE676c4X8jGKLhd+rdcsq808kDj1L\n"93fCeHm5n/6pUCyziKrlA4b9v7LWlbcxevOF34GfId5yHI9Y/QCB/IIDeG\n"EW+Oym\n"jSubJrlq0G/AwEAAncMEAdwYDVR0TAQH/BAUwAwEB/zAOBnW7HQB8A8EBAMC\n"n"AYYwHQYDVR0OBBYEFiQYzIU07LwMIJQUcfmcx7lQTgoIMA0GCSGSib3DQEB\n"CwUA\n"44BAQK8jydaQZChGsV2USggNiMOruY0u6r4IK5lpDB/G/wkjU0uYKGX9r\n"bxeDn\n"U5PMCCjxpXCI6T5iHTfUjUr6adTrCC2jeHEZERuHl1Bjtlmsv0tadQ1wUS\n"n"U+G563pYAACbvX8MWY7Vu33Pq0L2Uq24Xee6V7/Uq128wiT06GXFvKWlY\n"bYK8U90vv\n"o/ufQJvMT8QPHR8jrdkPSHCa2XV4cdFyQzR1bldZwgJcImApzyMZF06fQ6XU\n"n"5Msl+yMRQ+hDKXjioaldXgJkUk642M4UwtB780b2xJNd2ZhwLnoQdeX\n"eGADbkpy\n"rqRxfboQnoZsG4q5WTP468SQvvG5\n"-----END CERTIFICATE-----

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

This string can then be copy/pasted into your keys file (or any other C-file) For instance you could add this to the file `aws_clientcredenital_keys.h`.

```

/*
 * PEM-encoded Just-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"
"MIIDOTCCAImqAwIBAgITBmVfz5m/iAo54vB4ikPmliZbyiANBakahkiG9w0BAQsF\n"
"ADA5MQswCQYDVQ0GEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDV00DExBBbW6\n"
"b24aUm9vdCBDOsAqMB4XDTE1MDUvNiAwMDAwMFoXDTE1MDUvNiAwMDAwMFoXDTE1\n"
"MAkGA1UEBhMCVVMxOzANBgNVBAoTBkFtYXovb1EzMBCGA1UEAxMQQW1hem9uIFJv\n"
"b30aQ0EaMTCCASiW00YJKoZIhvcNAQEBBQADgQEPADCCAAQoCqEBAJ4aHHKeNXi\n"
"ca9HqFB0fw7Y14h29Jl091ahYP10hAEvrAItht0a03p0saT0NroBvo3bSMaHFzZM\n"
"906II8c+6zf1tRn4SWiw3te5djadYZ6k/oI2peVKVuRF4fn9tBb6dNacmzU5L/qw\n"
"IFAGbHr0aLkm+a/sRxmPUDaH3KKH0Vj4utWo+UhnMJbu1Hheb4m1UcAwhmahRWa6\n"
"V0uiw5H5SNz/0eawLX0tdHA114ak957EwW67c4cX8iJGKLhD+rcdasa08p8kDi1L\n"
"93FcXmn/6bUCvziKrlA4b9v7LWIbxcceV0F34GfID5vHI9Y/OCB/IIDEqEw+0y0m\n"
"iaSubJrIaa0CAwEAaAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zA0BqNVH08BAf8EBAMC\n"
"AYYwHQYDVR00BBYEFIOYzIU07LwMLJQUcFmcx7IOTqoIMA0GCSaGSIB3DQEB\n"
"A4IBAQC8ida0ZChGsV2USaaNiM0ruYou6r4LK5IoDB/G/wk1Uu0vKGX9rbxenDI\n"
"U5PMCCimCXPI6T53iHTfIUJrU6adTrCC2aJeHZERxhlbI1B1it/msv0tad01wUs\n"
"N+aDS63pYaACbvXv8Mwv7Vu33PaUXHeeE6V/Uq2V8viT096LXFvKwLJbYK8U90vv\n"
"o/ufoJvTMT80tPHRh8irdkPSHCa2XV4cdFv0zR1bldZwaJcJmApzvMZFo6I06XU\n"
"5MsI+vMRQ+hDKXJioaldXaiUkK642M4UwtBV8ob2xJNDd2ZhwLno0deXeGADbkpv\n"
"raXRfbo0noZsG4a5WTP468S0vvG5\n"
"-----END CERTIFICATE-----"

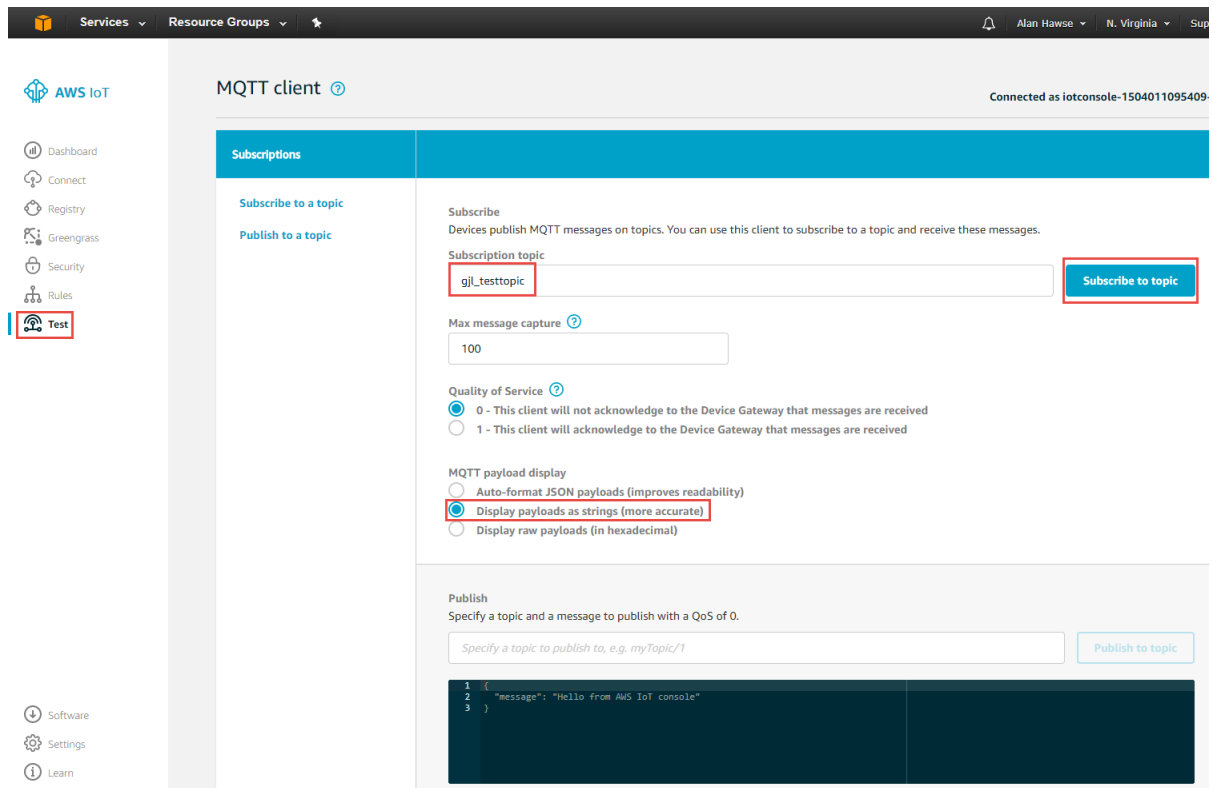
```

3.11 Test Client

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.

3.11.1 Subscribing to a Topic from the Test Client

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.



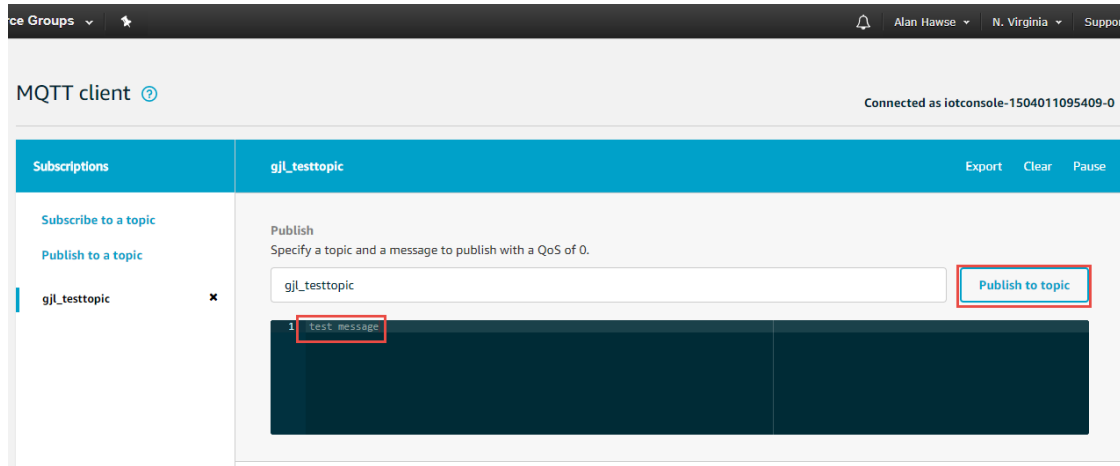
The screenshot shows the AWS IoT MQTT client interface. On the left sidebar, the 'Test' option is highlighted with a red box. The main content area is titled 'MQTT client'. Under the 'Subscriptions' tab, the 'Subscribe to topic' section shows a 'Subscription topic' field with the value 'gjl_testtopic' and a 'Subscribe to topic' button, both highlighted with red boxes. Below this, the 'MQTT payload display' section has three radio button options: 'Auto-format JSON payloads (improves readability)', 'Display payloads as strings (more accurate)' (which is selected and highlighted with a red box), and 'Display raw payloads (in hexadecimal)'. At the bottom, the 'Publish' section has a text input field with the placeholder 'Specify a topic to publish to, e.g. myTopic/1' and a 'Publish to topic' button.

3.11.2 Publishing to a Topic from the Test Client

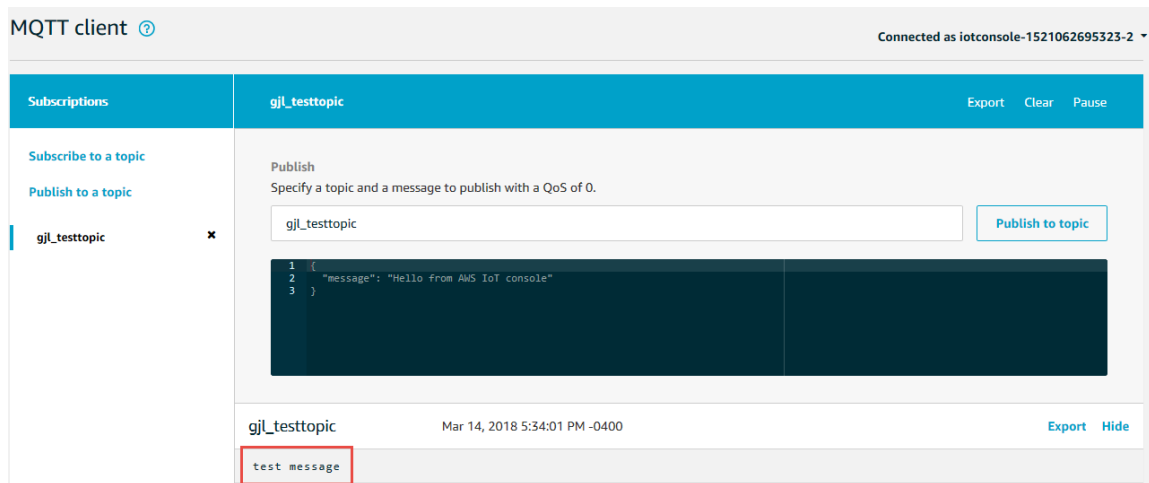
Now that I am subscribed to a topic, I can publish messages to that topic from another instance of the MQTT test client.

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).

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



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



3.12 GreenGrass