

## Chapter 4c: Cloud Protocol: MQTT Using AWS

At this end of this chapter, you will understand how to use the MQTT protocol with Amazon Web Services (AWS), including:

- How to write ModusToolbox™ for Connectivity firmware to interact with the AWS IoT Cloud using MQTT.
- How the [AWS Cloud](#) works.
- How to provision "things" (which for semantic reasons, will be notated *thing*, a.k.a. your IoT device, in this chapter) in the AWS IoT Cloud by creating a *thing*, policies and certificates.
- AWS Security.
- How to use a *thing* shadow.
- How to use the AWS IoT test client to subscribe and publish to topics.
- Understand the scope of systems that can be implemented in the AWS Cloud (SNS, Database etc.).

### Table of contents

<b>4c.1</b>	<b>Introduction .....</b>	<b>3</b>
<b>4c.2</b>	<b>Amazon Web Services (AWS) .....</b>	<b>3</b>
<b>4c.3</b>	<b>AWS IoT Introduction .....</b>	<b>3</b>
<b>4c.4</b>	<b>AWS IoT Resources .....</b>	<b>4</b>
4c.4.1	Thing .....	4
4c.4.2	Certificate .....	4
4c.4.3	Policy .....	5
<b>4c.5</b>	<b>AWS IoT Console .....</b>	<b>5</b>
4c.5.1	Creating an AWS IoT account .....	6
4c.5.2	Thing Shadow .....	6
4c.5.3	Topics .....	8
4c.5.4	Device Shadow Topics .....	8
<b>4c.6</b>	<b>Using MQTT with AWS .....</b>	<b>9</b>
4c.6.1	MQTT client .....	9
<b>4c.7</b>	<b>Using HTTPS with AWS .....</b>	<b>9</b>
<b>4c.8</b>	<b>Exercises .....</b>	<b>10</b>
	Exercise 1: Run the AWS tutorial .....	10
	Exercise 2: Create new AWS <i>Thing</i> .....	11
	Exercise 3: Learn how to use the AWS MQTT test client .....	16
	Exercise 4: Run the ModusToolbox™ MQTT client app .....	18
	Exercise 5: ModusToolbox™ AWS MQTT firmware flow .....	20
	Exercise 6: Publish from AWS test MQTT client to toggle kit LED .....	21
	Exercise 7: Get a <i>Thing</i> Shadow from AWS using HTTPS .....	21
<b>4c.9</b>	<b>References .....</b>	<b>22</b>
<b>4c.10</b>	<b>Appendix .....</b>	<b>22</b>
4c.10.1	Exercise 5 Answers .....	22

## Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	Displays file names and paths	<i>sourcefile.hex</i>
[bracketed, bold]	Displays keyboard commands in procedures	[Enter] or [Ctrl] [C]
Menu > Selection	Represents menu paths	File > New Project > Clone
<b>Bold</b>	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the <b>Debugger</b> icon, and then click <b>Next</b> .

## 4c.1 Introduction

Whether you use Amazon Web Services, someone else's cloud, or your own solution, at the heart of it your device will connect and communicate using the AWS IoT Device SDK and MQTT libraries. This chapter will discuss some of the concepts that are important to know when connecting your IoT device to AWS.

## 4c.2 Amazon Web Services (AWS)

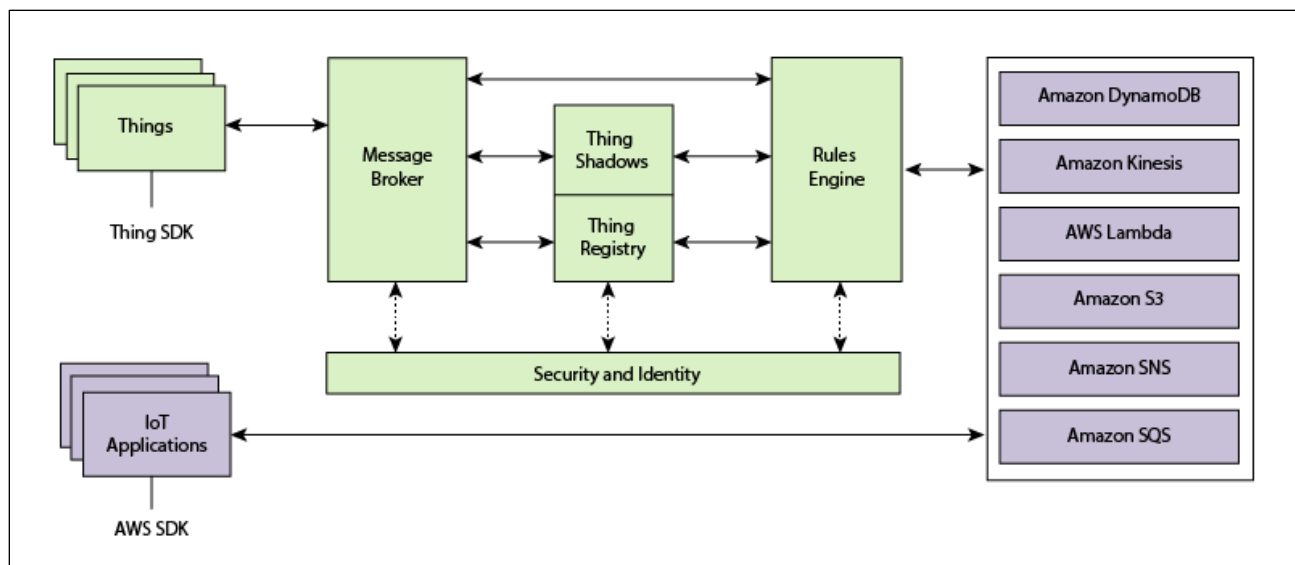
[AWS](#) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality (which makes more money for Amazon than their retail operations). AWS is built from a vast array of both virtual and actual servers and networks as well as a boatload of webserver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

- [AWS IoT](#): A cloud platform that provides Cloud services for IoT devices (the subject of this chapter).
- Amazon Elastic Cloud ([EC2](#)): 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 [Amazon DynamoDB](#), [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.

## 4c.3 AWS IoT Introduction

The AWS IoT Cloud service supports MQTT Message Brokers, HTTP access, **plus** a bunch of server-side functionality that includes (again, this is a partial list because Amazon frequently adds more functionality):

- 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.



## 4c.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.

### 4c.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).

### 4c.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.

### 4c.4.3 Policy

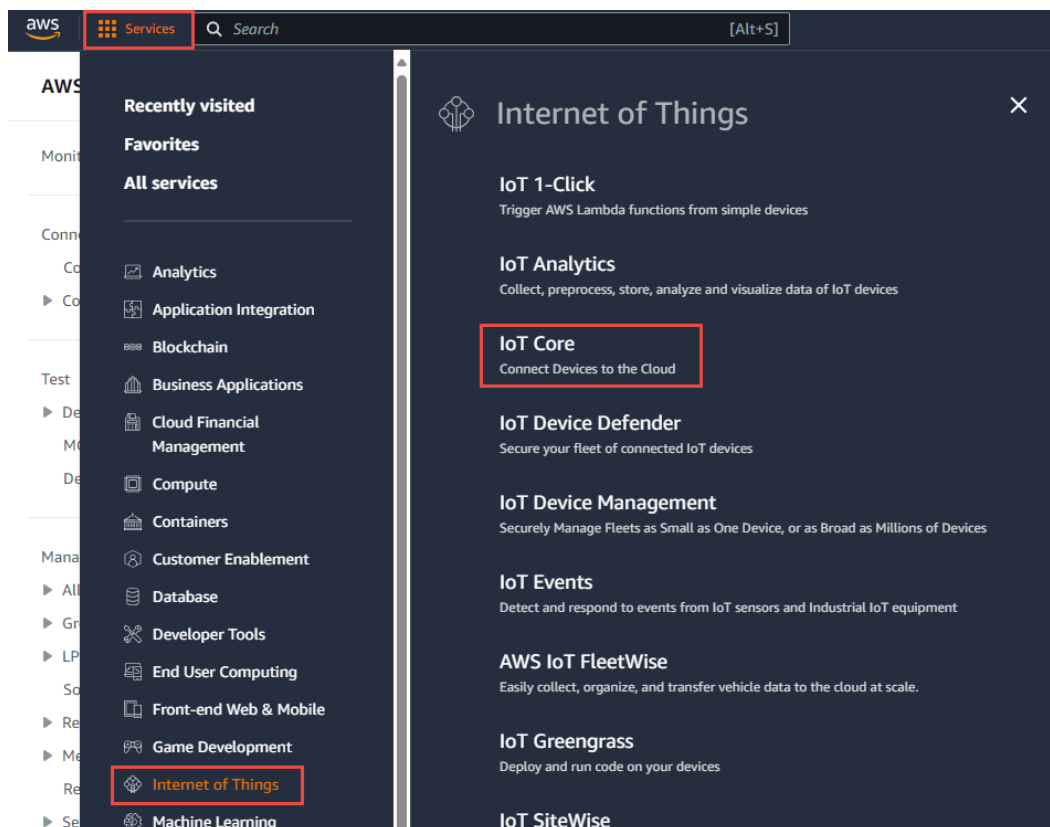
After creating a certificate for your internet-connected *thing*, you must create and attach an AWS IoT policy that will determine what AWS IoT operations the *thing* may perform. AWS IoT policies are JSON documents and they follow the same conventions as AWS Identity and Access Management policies.

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"
    }
  ]
}
```

### 4c.5 AWS IoT Console

The AWS Console can be found at <https://console.aws.amazon.com>. Once you are logged in, click on **Services** > **IoT** > **IoT Core** to access the IoT console.

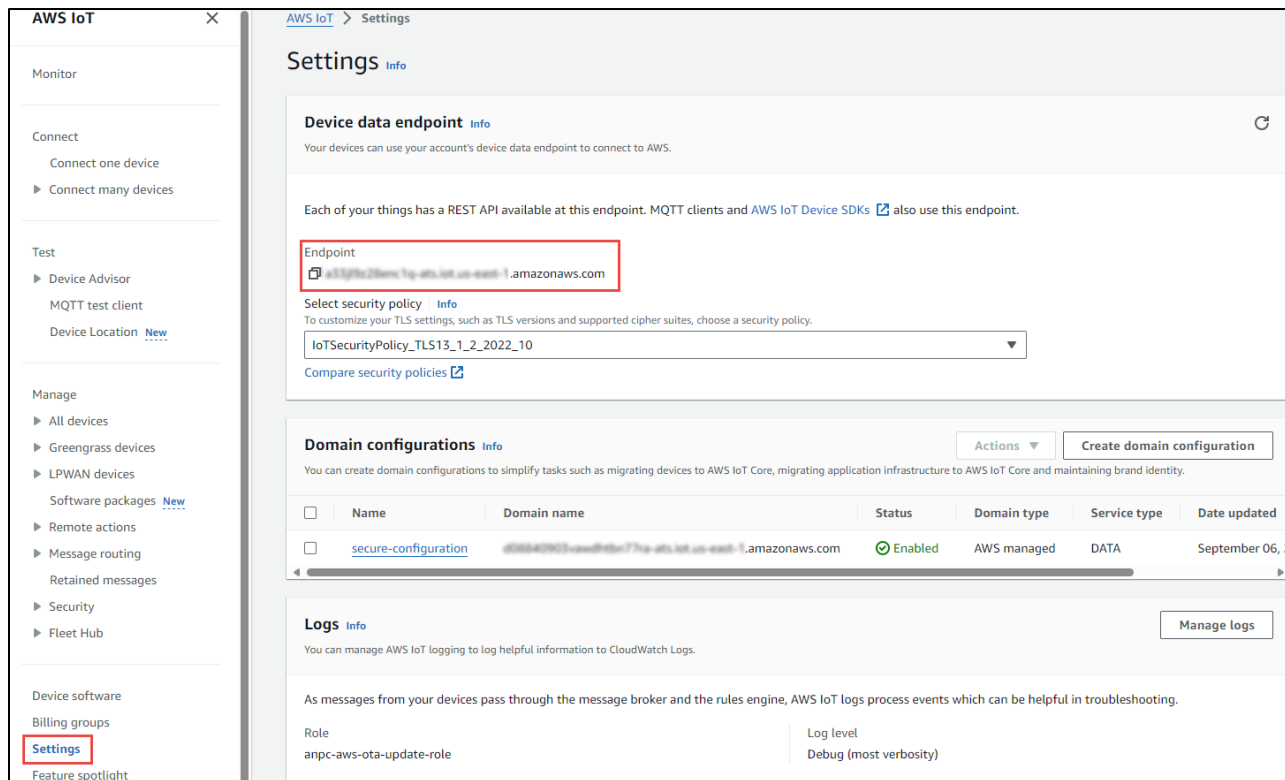


*Note: Screenshots shown throughout this document may not exactly match the AWS site because Amazon updates their site frequently. However, the concepts are the same and you should be able to follow along without too much trouble.*

## 4c.5.1 Creating an AWS IoT account

To create a new AWS account, you need to provide a credit card number. The basic account is free for a year but if you don't cancel before that (or remove your credit card from the Amazon payment options) it will start charging your credit card after the year is up. When you create an AWS IoT account, Amazon will create a new virtual machine for you in the Cloud and will turn on an MQTT Message Broker and an HTTP server on that machine. To connect your device to the machine you will need to know the DNS name of the virtual machine.

To find the virtual machine's DNS name, click on **Settings** at the lower left corner of the AWS IoT console window. The name is listed as the Endpoint.



## 4c.5.2 Thing Shadow

A *thing* shadow (sometimes referred to as a device shadow) is a JSON document (<http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html>) that is used to store and retrieve current state information for a *thing* (device, app, etc.). 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:

### 4c.5.2.1 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 it.
- **Reported:** The reported state of the *thing*. *Things* write to this portion of the document to report their new state. Applications can read this portion of the document to determine the state of a *thing*.

#### 4c.5.2.2 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.

#### 4c.5.2.3 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.

#### 4c.5.2.4 ClientToken:

A string unique to the device that enables you to associate responses with requests in an MQTT environment.

#### 4c.5.2.5 Version:

The document version. Every time the document is updated, this version number is incremented. This is 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": {
    "desired": {
      "color": "BLUE"
    }
  }
}
```

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

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

### 4c.5.3 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.

### 4c.5.4 Device Shadow Topics

Each *thing* that you have will have a group of topics (<https://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html>) 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	A JSON message that you publish to this topic will update the 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 entire 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, 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 /get/accepted with the current state of the shadow or to /get/rejected if the operation is not allowed.
/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/documents" to subscribe to updated documents for all *thing* shadows.

## 4c.6 Using MQTT with AWS

ModusToolbox™ for Connectivity includes an MQTT client library which uses the [AWS IoT Device SDK MQTT client library](#). All features supported by the AWS IoT Device SDK MQTT Library are supported by ModusToolbox™ for Connectivity. There is a configuration file for this library located in *libs/mqtt/cyport/include/iot\_config.h*. You should copy this file to the root directory of your application and use it to adjust default library settings.

In addition to the library, there are several demo applications that can be used as a starting point for using MQTT with AWS.

### 4c.6.1 MQTT client

In this example, the MQTT client RTOS task establishes a connection with the configured MQTT Broker (AWS IoT Core in this example) and creates two tasks - publisher and subscriber. The publisher task publishes messages on a topic when the user button is pressed. The subscriber task subscribes to the same topic and controls the user LED based on the messages received

In the exercises below, we will use the MQTT client project as a starting point to learn the details of using ModusToolbox™ for Connectivity to interact with AWS using MQTT.

## 4c.7 Using HTTPS with AWS

In addition to MQTT, AWS supports a [REST API](#) interface to their cloud. The REST API Endpoint is:

`https://<your_endpoint>:8443/things/<your_thing_name>/shadow`

Note that the port that AWS uses for secure HTTP traffic is 8443 instead of the typical 443.

The connection must have a client verified connection (you need to provide your certificate and private key). After you have a connection you can GET, POST and DELETE the document which is in JSON format.

Here is an example of a CURL connection to AWS:

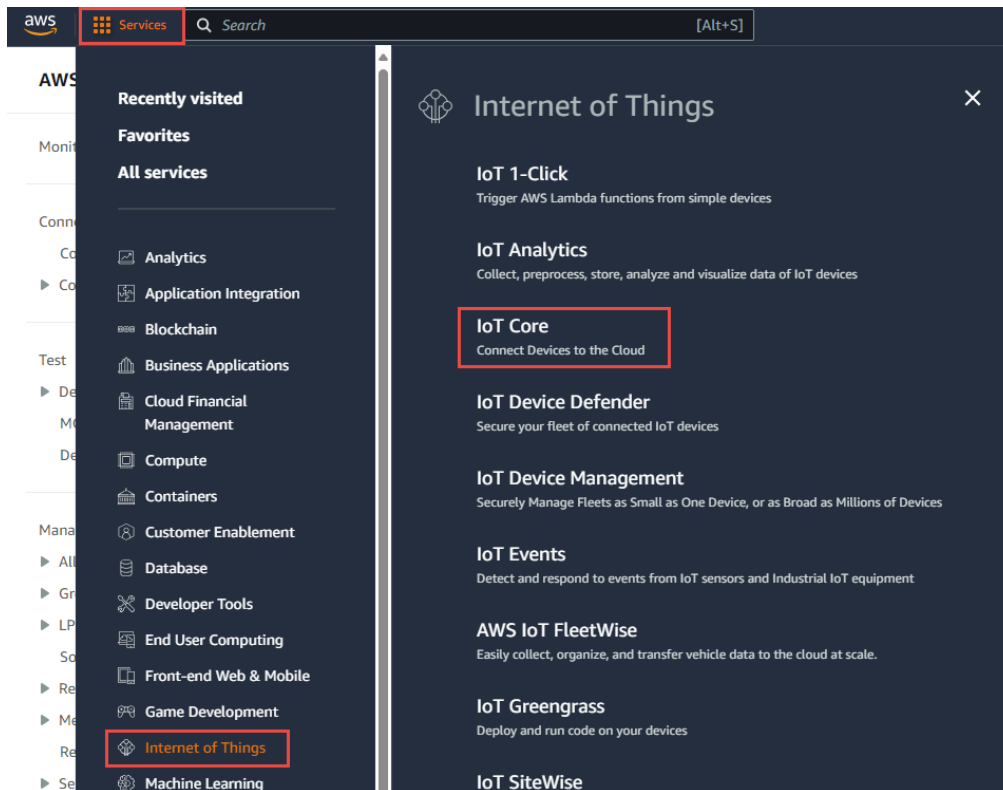
```
CURL -v --cert 6fb5d874d6-certificate.pem --key 6fb5d874d6-private.pem --cacert  
rootca.cer -X GET https://amk6m5lqrxr2u.iot.us-east-  
1.amazonaws.com:8443/things/ww101_39/shadow
```

## 4c.8 Exercises

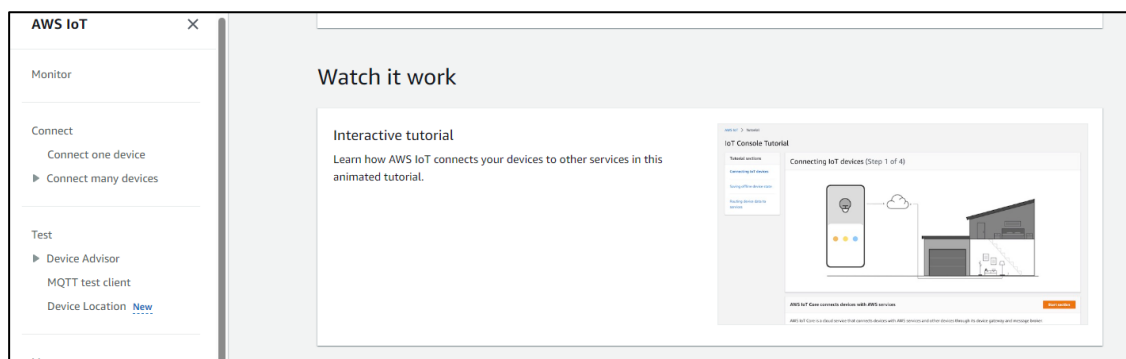
### Exercise 1: Run the AWS tutorial

Run the tutorial on the Amazon IoT Console (<https://console.aws.amazon.com>).

- ☐ 1. Sign up for an AWS account.
- ☐ 2. Once you are logged in, from the **Services** menu, select **IoT Core**:



- ☐ 3. In the **Watch it work** section, click the image to run the interactive tutorial.

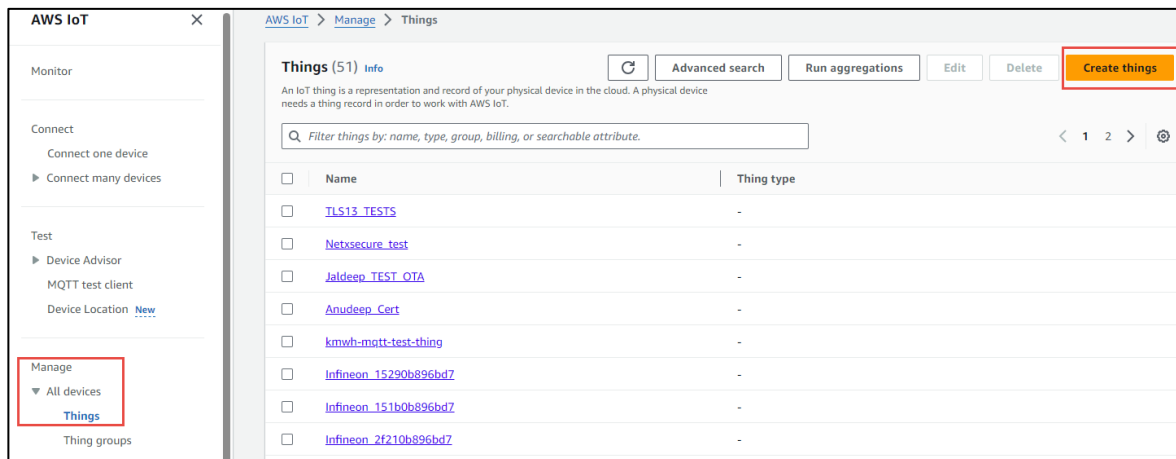


- ☐ 4. Follow the instructions to complete the tutorial.

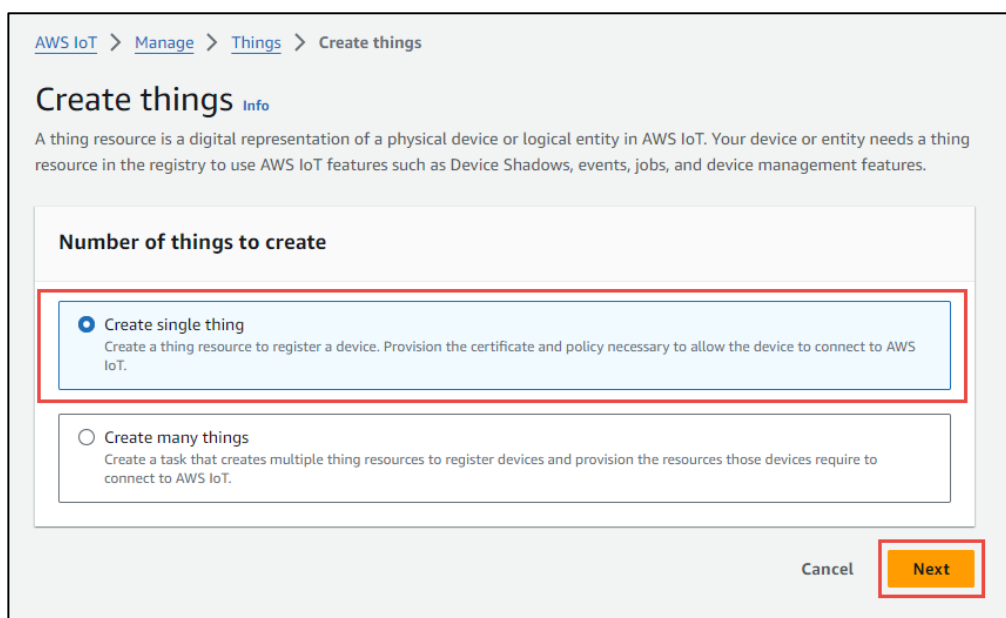
## Exercise 2: Create new AWS Thing

Provision a new *thing* in the AWS IoT Cloud and establish its policy and credentials.

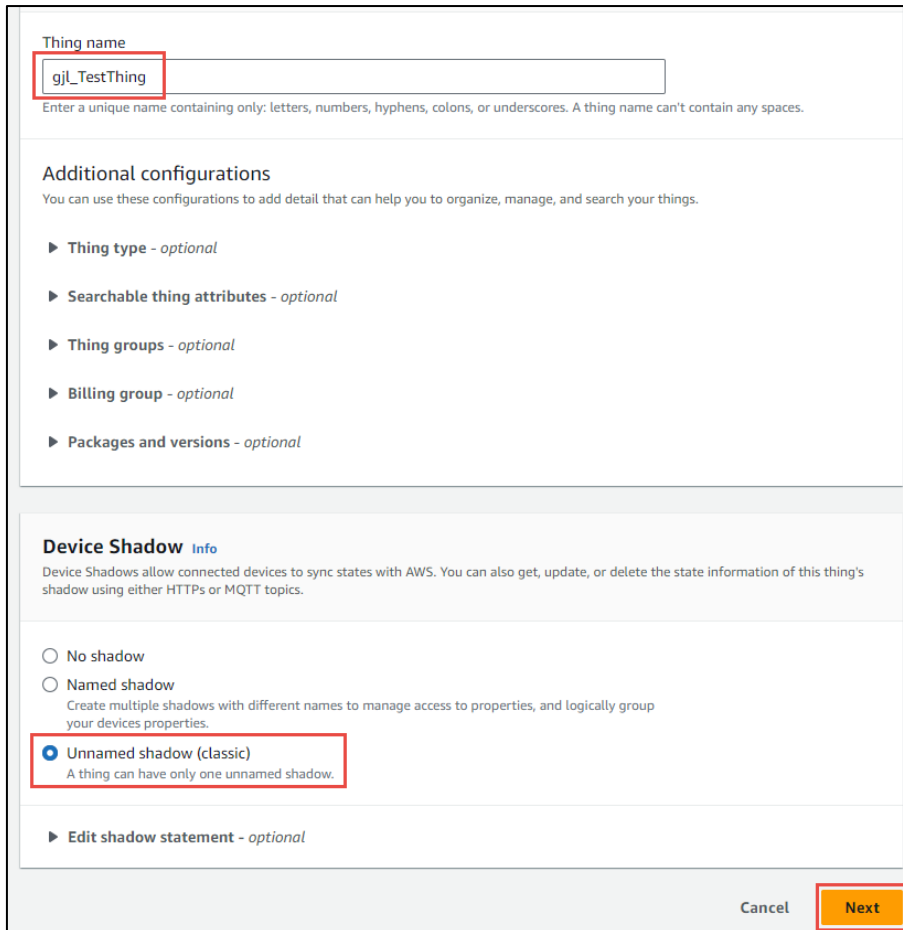
1. From the IoT Console, go to **Manage > All devices > Things**.



2. Click the **Create things** button. Select the option to **Create single thing** and click **Next**.



3. Name your *thing* **<YourInitials>\_TestThing** (or whatever name you choose). Select the option for an **Unnamed shadow**. Click **Next**.



Thing name

gjl\_TestThing

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
- ▶ Packages and versions - 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.

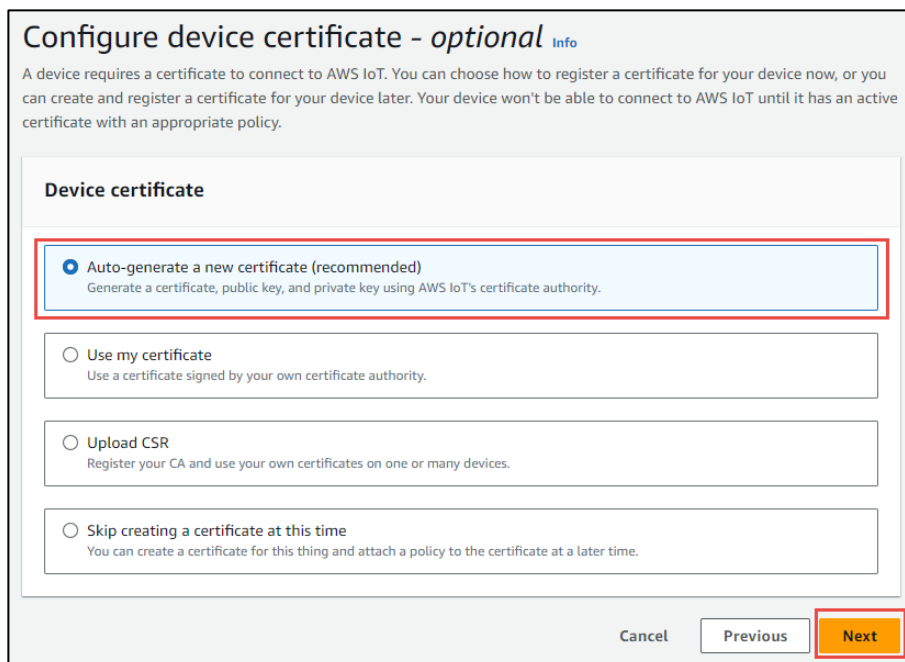
☒ Unnamed shadow (classic)  
A thing can have only one unnamed shadow.

▶ Edit shadow statement - optional

Cancel **Next**



4. Before you can access the broker from your kit you need to create the encryption keys that enable you to identify it as an allowed device. To do this, select **Auto-generate a new certificate** and click Next.



**Configure device certificate - optional** [Info](#)

A device requires a certificate to connect to AWS IoT. You can choose how 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.

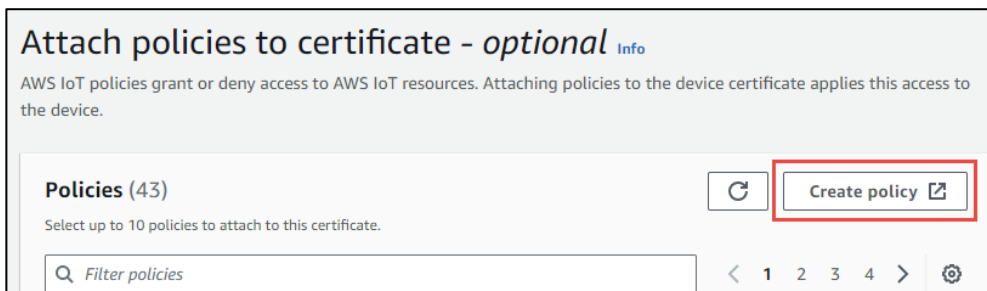
☐ Upload CSR  
Register your CA and use your own certificates on one or many devices.

☐ 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.

Cancel Previous **Next**



- Now you need to attach a policy to the certificate. The policy determines which IoT actions are allowed for the device. Click **Create policy** to create a new one.



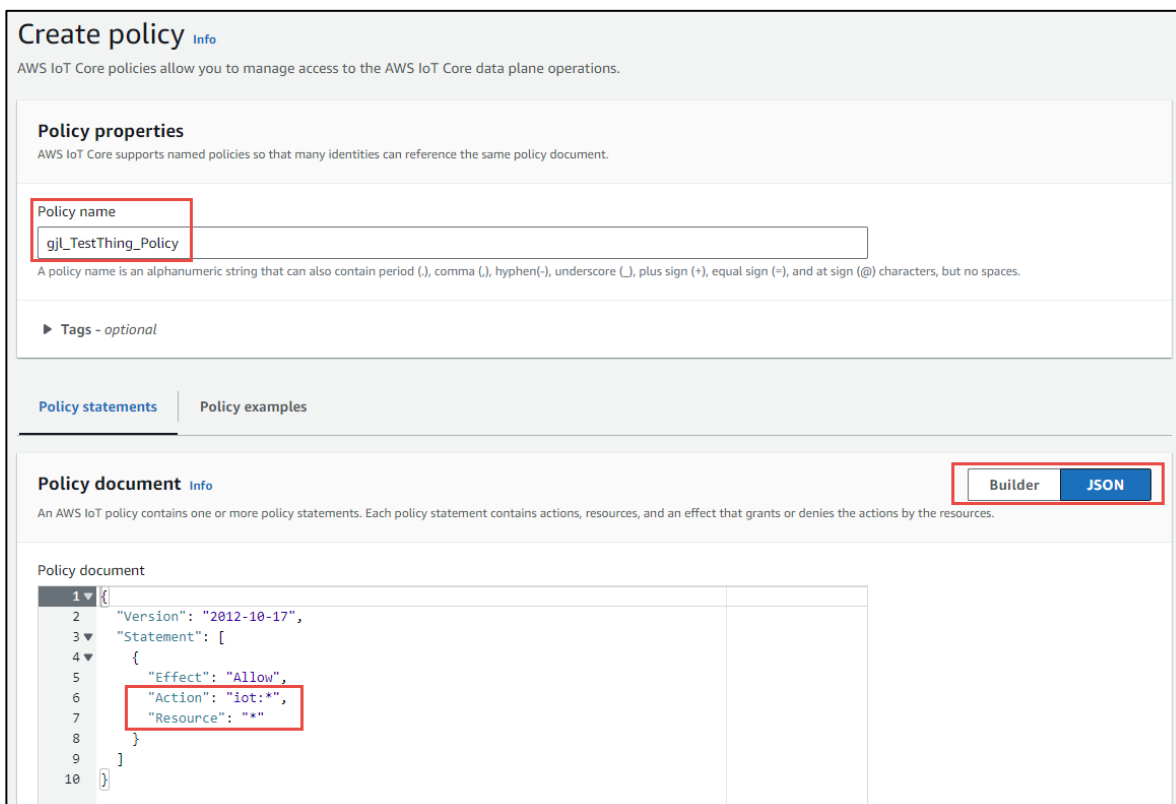
*Note: This will open a new tab where you will create a new policy.*



- Give the policy a name such as **<YourInitials>\_TestThing\_Policy**.



- You can use the **Builder** to setup the policy, or you can switch to **JSON** and enter the policy manually. Let's edit the JSON document manually to allow any IoT action on any resource.



- Once you are satisfied with the policy settings, click **Create**.



- Switch back to the previous browser tab. Select your new policy from the list and click **Create Thing**.

### 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 (1/44)**

↺

Create policy [↗](#)

Select up to 10 policies to attach to this certificate.

🔍 gjl\_test

×

1 match < 1 > ⚙️

<input checked="" type="checkbox"/>	Name
<input checked="" type="checkbox"/>	<a href="#">gjl_TestThing_Policy</a>

Cancel

Previous

Create thing



10. Download the device certificate, public and private key files, and Root CA certificate. We will be using Amazon Root CA 1.

*Note: If you don't download the keys at this step, you **cannot** return. You must download those files now to ensure that the TLS works!*

### 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  
a6b037994ae...te.pem.crt

Deactivate certificate

Download

#### 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  
a6b037994aea99be81f5e40...0cc6807-public.pem.key

Download

Private key file  
a6b037994aea99be81f5e40...cc6807-private.pem.key

Download

#### 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

Download

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

Download

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](#)

Done



11. Click **Done** after you have downloaded the files. You have now created your first AWS *Thing*!



12. Once you get to this point, you should verify:

- You have a *thing* (**Manage > Things**).
- You have a certificate attached to the *thing* (from the *thing*, look at the **Certificates** tab).
- The certificate is Active (look for **Active** in the Status column).
- The certificate has a policy attached to it (click on the Certificate and look at the **Policies** tab).
- The policy allows all IoT actions (iot:\*) for any resource (\*) (click on the Policy).

If any of the above is not true, fix it before proceeding.

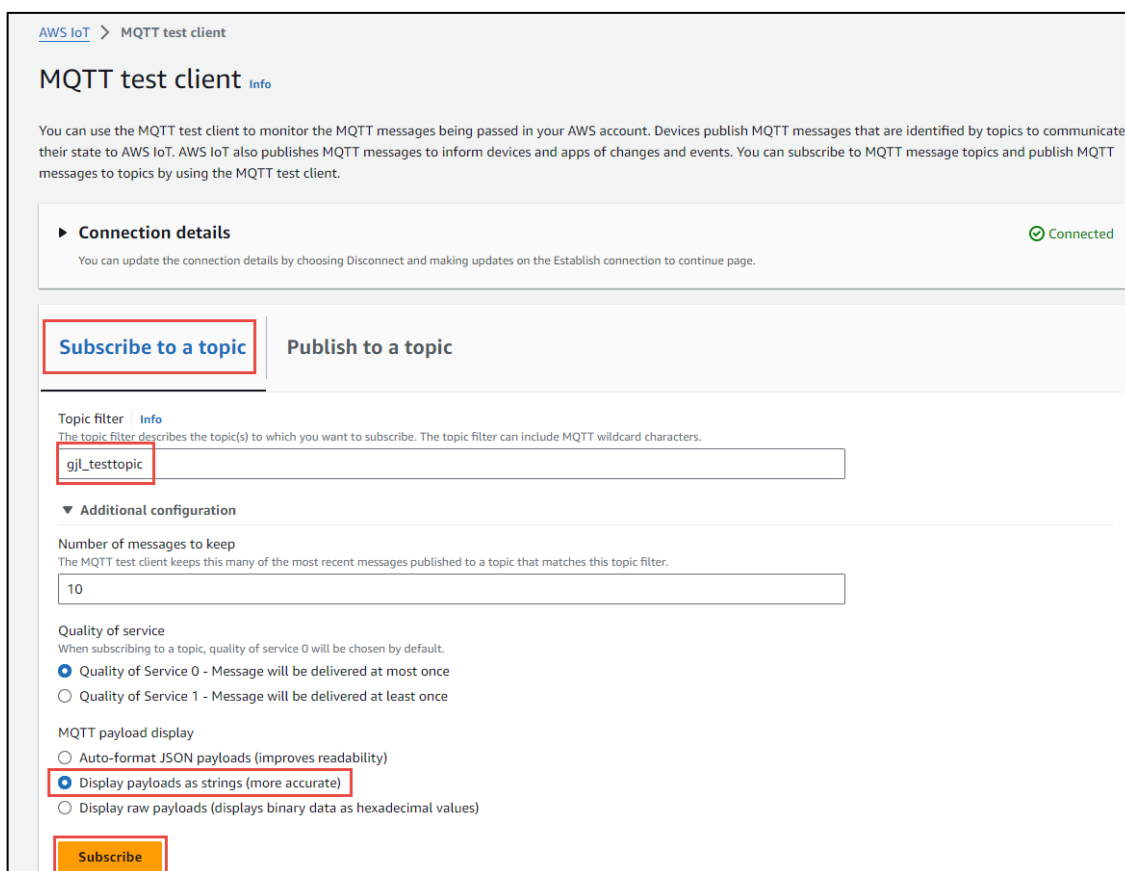
## Exercise 3: Learn how to use the AWS MQTT test client

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. You will use this client to test the later exercises.

To make the subscribe/publish actions more understandable, it is useful to team up with another person for this exercise. Alternately, you can run two tabs in your browser – one to subscribe and one to publish.

### Subscribing to a Topic from the Test client

- ☐ 1. Select **Test > MQTT test client** from the panel on the left side of the screen.
- ☐ 2. Select the **Subscribe to a topic** tab and enter a topic that you want to subscribe to such as **<your\_initials>\_testtopic** in the **Subscription topic** box.
- ☐ 3. Under the **Additional configuration** section, select **Display payloads as strings** and then click **Subscribe**.



AWS IoT > MQTT test client

### MQTT test client Info

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

► **Connection details** Connected

You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

**Subscribe to a topic** **Publish to a topic**

**Topic filter** Info

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

▼ **Additional configuration**

**Number of messages to keep**

The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

**Quality of service**

When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once

☐ Quality of Service 1 - Message will be delivered at least once

**MQTT payload display**

☐ Auto-format JSON payloads (improves readability)

☒ Display payloads as strings (more accurate)

☐ Display raw payloads (displays binary data as hexadecimal values)

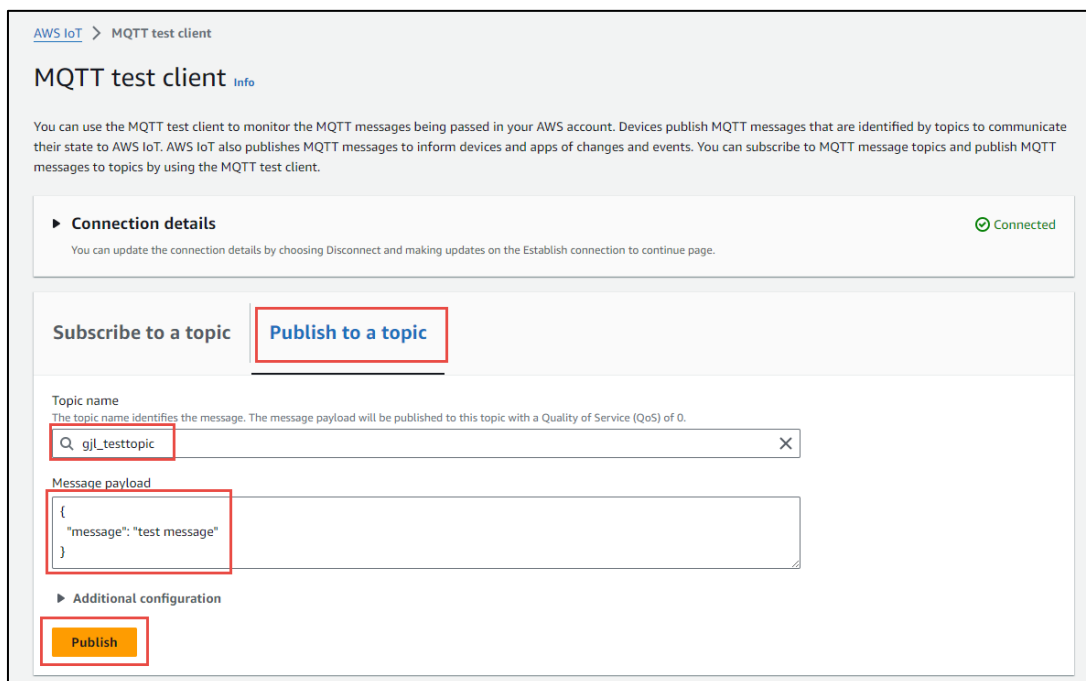
**Subscribe**



## Publishing to a Topic from the Test client

Now that you are subscribed to a topic you can publish messages to that topic from another instance of the MQTT test client.

- ☐ 1. Open another web browser tab, go to <https://console.aws.amazon.com>, open the IoT Core page and go to **Test > MQTT test client**.
- ☐ 2. Select the **Publish to a topic tab** and fill in the name of the topic that you subscribed to earlier. The name must be exactly the same (topic names are case sensitive).
- ☐ 3. Then type in your message and press **Publish to topic**. You can see in the box below I sent a JSON message containing "message": "test message."



AWS IoT > MQTT test client

### MQTT test client Info

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

► **Connection details** Connected

You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

**Subscribe to a topic** **Publish to a topic**

Topic name  
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

gjl\_testtopic

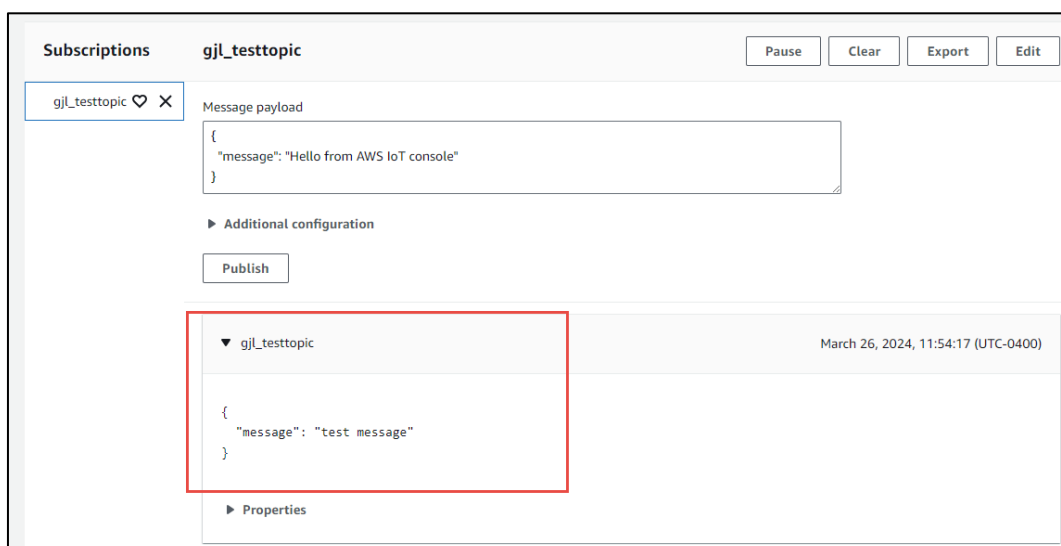
Message payload

```
{
  "message": "test message"
}
```

► **Additional configuration**

**Publish**

- ☐ 4. Go back to the tab with the subscription and check that the published message was sent to the subscriber.



**Subscriptions** **gjl\_testtopic** Pause Clear Export Edit

gjl\_testtopic ♥ ×

Message payload

```
{
  "message": "Hello from AWS IoT console"
}
```

► **Additional configuration**

**Publish**

▼ gjl\_testtopic March 26, 2024, 11:54:17 (UTC-0400)

```
{
  "message": "test message"
}
```

► **Properties**

## Exercise 4: Run the ModusToolbox™ MQTT client app

In this exercise you will create and run the MQTT client application to connect to AWS.

**Note:** The code example has directions to connect either to AWS or to the Mosquitto MQTT test broker at [mosquitto.org](https://mosquitto.org). Refer to the code example's *README.md* if you want to try using the Mosquitto MQTT broker instead of AWS.

- ☐ 1. Create a New Application within the ModusToolbox™ IDE and select the **Wi-Fi MQTT Client** example application. This application has a task to publish and a separate task to subscribe.
- ☐ 2. Modify the *wifi\_config.h* file for your network.

**Note:** *wifi\_config.h* is in the *configs* directory.

- ☐ 3. Copy the device certificate, private key, and Root CA certificate that you generated and saved previously into the proper locations in *mqtt\_client\_config.h*.

They must be formatted as shown below:

```
179 /* PEM-encoded Root CA certificate */
180 #define ROOT_CA_CERTIFICATE \
181 "-----BEGIN CERTIFICATE-----\n" \
182 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPm1jZbyjANBgkqhkiG9w0BAQsF\n" \
183 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBBW\n" \
184 "b24gUm9vdCBDQSAAxMB4XDTE1MDUyNjAwMDAwMFoXDTE1MDUyNjAwMDAwMFow\n" \
185 "MAkGA1UEBhMCVGVhZDANBgNVBAoTBkFtYXNjEzMBcGA1UEAxMQQW1hem9uIFJv\n" \
186 "b3QgQ0EgMTCCASAwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHhK\n" \
187 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAItht0gQ3p0sqTQNr0bVo3bSMgH\n" \
188 "fZM\n" \
189 "906II8c+6zf1tRn4Sliw3te5djgdYZ6k/oI2peVKVURF4fn9tBb6dNqcmzU5L\n" \
190 "qW\n" \
191 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnM3bulHheb4mjUcAwhmahR\n" \
192 "Wa6\n" \
193 "V0Ujw5H5SNz/0egwLX0tdHA114gk957EwW67c4cX8jJGKLhD+rcdqsq08p8kD\n" \
194 "i1L\n" \
195 "93FcXmn/6pUCyziKr1A4b9v7LWlBxcceVOF34GfID5yHI9Y/QCB/IIDeGEw+Oy\n" \
196 "Qm\n" \
197 "jgSubJrIqg0CAwEAaAaCMEAwDwYDVR0TAQH/BAUwAwEB/zA0BgNVHQ8BAf8EBAMC\n" \
198 "n\n" \
199 "AYYwHQYDVIR0B8B8YEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEB\n" \
200 "CwUA\n" \
201 "A4IBAQCYY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbx\n" \
202 "enDI\n" \
203 "U5PMCCjmmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlBI1Bjtt/msv0tadQ1w\n" \
204 "Us\n" \
205 "N+gDS63pYaACbvXy8Mwly7Vu33PqUXHeeE6V/Uq2V8viT096LXFvKw1JbYK8U9\n" \
206 "0vv\n" \
207 "o/ufQJvTMT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1b1dZwgJcJmApzyMZFO6IQ6XU\n" \
208 "n\n" \
209 "SMsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
210 "n\n" \
211 "rqXRfboQnoZsG4q5WTP468SqvG5\n" \
212 "-----END CERTIFICATE-----"
```

You can manually format the strings, as shown above or you can use one of the following methods:

- I. Use the *format\_aws\_certificates.py* Python script to format them simultaneously. The script is in the class material's *Scripts* directory. To use it:
  - a) Place the script and certificates/keys in the same directory.
  - b) The private key file name must end with *private.pem.key*, the certificate file name must end with *certificate.pem.crt*, and the Amazon Root CA must end with *CA1.pem*.
  - c) Open modus-shell, go to the directory with the script and enter:

```
python ./format_aws_certificates.py
```
  - d) Copy/paste the formatted strings from the output window to the proper locations in *mqtt\_client\_config.h*.

- II. Use the *format\_certificate.py* Python script to format them one at a time. The script is in the class material's *Scripts* directory. To use it:
  - e) Put the script and certificates/keys in the same directory.
  - f) Open modus-shell, go to the directory with the script and enter:  
`python ./format_certificates.py <filename>.`
  - g) Copy/paste the formatted strings from the output window to the proper locations in *mqtt\_client\_config.h*.

*Note: The file that you downloaded called <name>-public.pem.key is a public key for your thing. In this case, Amazon already has the public key, so you don't need to provide it.*



4. Set the `#define` for `MQTT_BROKER_ADDRESS` in *mqtt\_client\_config.h* to your broker address.

The correct address can be found by clicking on **Settings** at the lower left corner of the Amazon AWS IoT Core console window. The broker address is listed as the **Endpoint**. For example:

```
49 /* MQTT Broker/Server address and port used for the MQTT connection. */  
50 #define MQTT_BROKER_ADDRESS "amk6m5lqrxr2u-ats.iot.us-east-1.amazonaws.com"
```



5. Change the `#define` for `MQTT_PORT` to 8883. This is necessary because AWS does not use the standard MQTT port.



6. Change the `#define` for `MQTT_SECURE_CONNECTION` to 1. This is necessary because AWS requires a secure connection.



7. Modify the `#define` for `MQTT_PUB_TOPIC` and `MQTT_SUB_TOPIC`.

This is the topic that your device will publish and subscribe to. You can use the topic from the previous exercise with your initials in the name.



8. Modify `MQTT_CLIENT_IDENTIFIER` to the name of the *thing* that you created previously.

*Note: This a good practice to prevent conflicts between multiple devices on a broker – every device connected to a broker MUST have a client identifier. However, there is a function defined in *mqtt\_task.c* that will append a timestamp onto your client identifier prefix to prevent multiple clients with identical identifiers.*



9. Build and program your project.



10. Open the serial port and watch your terminal session.



11. Subscribe to your topic using the AWS MQTT Test client.

When you press the button on your device, you should see updates to the topic in the test window.

---

## Exercise 5: ModusToolbox™ AWS MQTT firmware flow

Explain in detail the firmware flow for the publisher task by answering the following questions:

- ☐ 1. How do the AWS library functions (e.g. `IotMqtt_PublishSync`) get into your project?
- ☐ 2. What function is called when the button is pressed?
- ☐ 3. How does the button callback unlock the publisher task?
- ☐ 4. Are all messages sent to the AWS IOT MQTT Message Broker required to be in JSON format?
- ☐ 5. What steps are required to get an AWS connection established?
- ☐ 6. What function is called to send data to the server?

## Exercise 6: Publish from AWS test MQTT client to toggle kit LED

Publish messages using the AWS test MQTT client to test the subscribe task in the application.



1. Determine what string needs to be sent to turn the light on or off.

*Note: Look in the source code to find the string that is being looked for when a message is received.*

*Note: If you are successful, the LED on your device should turn on/off.*

## Exercise 7: Get a *Thing* Shadow from AWS using HTTPS



1. Create a project that uses HTTPS to connect to AWS. Get the shadow of your *thing* and print it to the UART.

*Note: Start with the [httpbin.org](http://httpbin.org) GET project using TLS from the HTTP chapter.*

*Note: In addition to initializing the root certificate for AWS, you will need to read in a valid thing certificate and private key to initialize the TLS identity. This step is necessary because AWS will validate that your kit is authorized to connect to the broker and interact with your thing.*

*Note: If you are using the CY8CPROTO-062-4343W or CY8CPROTO-062S2-43439 kit, you must add `CY_WIFI_HOST_WAKE_SW_FORCE=0` to the `DEFINES` in the Makefile in order to disable the Wi-Fi host wake feature. This is necessary because the user button and host wake pin from the Wi-Fi device are connected to the same PSoC pin.*

*The following code can be pasted into the Makefile below the existing `DEFINES`.*

```
# The CY8CPROTO-062-4343W and CY8CPROTO-062S2-43439 kits share
# the same GPIO for the user button (USER_BTN1) and the Wi-Fi host
# wake up pin (used by the PSoC to wake up the Wi-Fi device. Since this
# example uses the GPIO for interfacing with the user button, the
# SDIO interrupt to wake up the host is disabled by setting
# CY_WIFI_HOST_WAKE_SW_FORCE to '0'.
#
# If you wish to enable this host wake up feature, Comment out
# DEFINES+=CY_WIFI_HOST_WAKE_SW_FORCE=0. If you want this feature on
# one of the kits that share pins, change the GPIO pin for USER_BTN
# in design/hardware & comment out DEFINES+=CY_WIFI_HOST_WAKE_SW_FORCE=0.
DEFINES+=CY_WIFI_HOST_WAKE_SW_FORCE=0
```

## 4c.9 References

Resources	Link
AWS Developers Guide	<a href="http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html">http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html</a>
AWS IOT Getting Started	<a href="https://aws.amazon.com/iot/getting-started/">https://aws.amazon.com/iot/getting-started/</a>
A nice PowerPoint about MQTT	<a href="http://www.slideshare.net/PeterREgli/mq-telemetry-transport">http://www.slideshare.net/PeterREgli/mq-telemetry-transport</a>
MQTT Topic Naming Best Practices	<a href="http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices">http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices</a>
AWS Forum	<a href="https://forums.aws.amazon.com/forum.jspa?forumID=210">https://forums.aws.amazon.com/forum.jspa?forumID=210</a>

## 4c.10 Appendix

Answers to the questions asked in the exercises above are provided here.

### 4c.10.1 Exercise 5 Answers

1. How do the AWS library functions (e.g. `IotMqtt_PublishSync`) get into your project?

The `#include` for `iot_mqtt.h` in the C file cause the AWS library functions to be included in the project.

2. What function is called when the button is pressed?

```
isr_button_press()
```

3. How does the button callback unlock the publisher task?

It notifies the publisher task by calling:

```
xTaskNotifyFromISR(publisher_task_handle, new_device_state,  
SetValueWithoutOverwrite, &xHigherPriorityTaskWoken);
```

4. Are all messages sent to the AWS IOT MQTT Message Broker required to be in JSON format?

No, but messages that affect the shadow have to be JSON.

5. What steps are required to get an AWS connection established?

Connect to the network

Initialize the Cypress Secure Sockets and MQTT libraries

Set up network info and connection info

Open the connection using `IotMqtt_Connect()`

6. What function is called to send data to the server?

```
IotMqtt_PublishSync()
```

#### Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

**Published by**  
**Infineon Technologies AG**  
**81726 Munich, Germany**

**© 2024 Infineon Technologies AG.**  
**All Rights Reserved.**

#### IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office ([www.infineon.com](http://www.infineon.com)).

#### WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.