

## Renesas Synergy™ Platform

# Synergy MQTT/TLS AWS Cloud Connectivity Solution

---

## Introduction

This application note describes IoT Cloud connectivity solution in general, introduces you briefly to IoT Cloud providers, like Amazon Web Services (AWS), and covers the Synergy MQTT/TLS module, its features, and operational flow sequence (Initialization/Data flow). The application example provided in the package uses AWS IoT Core. The detailed steps in this document show first-time AWS IoT Core users how to configure the AWS IoT Core platform to run this application example demonstration.

This application note enables you to effectively use the Synergy MQTT/TLS modules in your own design. Upon completion of this guide, you will be able to add the MQTT/TLS module to your own design, configure it correctly for the target application, and write code using the included application example code as a reference and efficient starting point.

References to detailed API descriptions, and other application projects that demonstrate more advanced uses of the module, are in the *Synergy Software Package (SSP) User's Manual*, which serves as a valuable resource in creating more complex designs.

This Synergy MQTT/TLS AWS Cloud Connectivity solution is supported on AE-CLOUD1 and AE-CLOUD2 kits.

## Required Resources

To build and run the MQTT/TLS application example, you need:

### Development tools and software

- e<sup>2</sup> studio ISDE v7.5.1 or later, or IAR Embedded Workbench® for Renesas Synergy™ v8.23.3 or later, available at [www.renesas.com/synergy/tools](http://www.renesas.com/synergy/tools).
- Synergy Software Package (SSP) 1.7.8 or later ([www.renesas.com/synergy/ssp](http://www.renesas.com/synergy/ssp))
- Synergy Standalone Configurator (SSC) 7\_3\_0 or later ([www.renesas.com/synergy/ssc](http://www.renesas.com/synergy/ssc))
- SEGGER J-link® USB driver ([www.renesas.com/synergy/jlinksynergy](http://www.renesas.com/synergy/jlinksynergy))

### Hardware

- Renesas Synergy™ AE-CLOUD1 kit ([www.renesas.com/synergy/ae-cloud1](http://www.renesas.com/synergy/ae-cloud1)), which includes Wi-Fi board; and, AE-CLOUD2 kit ([www.renesas.com/synergy/ae-cloud2](http://www.renesas.com/synergy/ae-cloud2)), which includes a Pillar board, Wi-Fi board and BG96 Cellular shield.  
Note: A CAT-M1, NB-IoT, or EGPRS SIM card should be procured separately for cellular functionality.
- Renesas Synergy™ Application Example kit PMOD based Wi-Fi Module ([www.renesas.com/synergy/kits/ae-wifi1](http://www.renesas.com/synergy/kits/ae-wifi1)).
- PC running Windows® 10; the Tera Term console or similar application, and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari).
- Micro USB cables
- Ethernet cable

## Prerequisites and Intended Audience

This application note assumes that you have some experience with the Renesas e<sup>2</sup> studio ISDE and Synergy Software Package (SSP). Before you perform the procedures in this application note, follow the procedure in the *SSP User Manual* to build and run the Blinky project. Doing so enables you to become familiar with e<sup>2</sup> studio and SSP, and validates that the debug connection to your board functions properly. In addition, this application note assumes you have some knowledge of MQTT/TLS and its communication protocols.

The intended audience is users who want to develop applications with MQTT/TLS modules using Renesas Synergy™ S5 or S7 MCU Series.

## Contents

1. Introduction to Cloud Connectivity .....	4
1.1 Overview.....	4
1.2 Major Components .....	4
1.3 Cloud Provider Overview.....	5
1.3.1 Amazon Web Services IoT Core .....	5
1.3.1.1 Key Features .....	5
1.4 MQTT Protocol Overview .....	6
1.5 TLS Protocol Overview.....	7
1.5.1 Device Certificates, and Keys .....	8
1.5.1.1 Device Certificates .....	8
1.5.1.2 Loading Certificates onto your Device .....	8
1.5.2 Device Security Recommendations .....	8
2. Synergy MQTT/TLS Cloud Solution.....	9
2.1 MQTT Client Overview .....	9
2.2 Design Considerations .....	9
2.2.1 Supported Features.....	9
2.2.2 Operational Flow Sequence .....	10
2.3 TLS Session Overview .....	10
2.3.1 Design Considerations .....	10
2.3.2 Supported Features.....	11
2.3.3 Operational Flow Sequence .....	11
2.3.3.1 TLS Handshake.....	11
2.3.3.2 Initialization Flow Sequence.....	12
2.3.3.3 Data Communication Flow Sequence.....	13
3. MQTT/TLS Application Example.....	13
3.1 Application Overview.....	13
3.2 Software Architecture Overview .....	14
3.2.1 Console Thread .....	14
3.2.2 MQTT Thread.....	15
3.2.3 MQTT Rx Thread.....	15
3.3 IoT Cloud Configuration (AWS).....	15
3.3.1.1 AWS IoT Policies.....	15
3.3.2 Creating a Device on AWS IoT Core.....	16
3.3.2.1 Open AWS IoT Core Service .....	16
3.3.2.2 Create a Thing.....	16
3.3.3 Generating Device Certificate and Keys .....	23
3.3.4 Creating a Policy for your Device .....	26
3.3.5 Connecting the Certificate to the Policy .....	28

4.	Running the MQTT/TLS Application .....	30
4.1	Importing, Building, and Loading the Project .....	30
4.2	Manually Adding the Board Support Package for the AE-CLOUD1/AE-CLOUD2 Kit .....	30
4.3	Powering up the Board .....	31
4.4	Connect to AWS IoT Cloud .....	31
4.4.1	Configuration Wizard Menu .....	32
4.4.1.1	Network Interface Selection .....	32
4.4.1.2	AWS IoT Core Configuration .....	39
4.4.1.3	Dump Previous Configuration .....	45
4.4.2	Demo Start/Stop Command .....	46
4.5	Verifying the Demo .....	46
4.5.1	Starting the Synergy Cloud Connectivity Demonstration .....	46
4.5.2	Opening the MQTT Client on AWS IoT Core .....	47
4.5.3	Publishing the MQTT Message from AWS MQTT Client .....	50
4.5.4	Stopping the Synergy Cloud Connectivity Demonstration .....	51
4.6	Customizing the demo delays .....	51
5.	Next Steps .....	51
6.	MQTT/TLS Reference .....	51
7.	Known Issues and Limitations .....	51
	Revision History .....	53

## 1. Introduction to Cloud Connectivity

### 1.1 Overview

Internet of Things (IoT) is a sprawling set of technologies described as connecting everyday objects, like sensors or smart phones, to the World Wide Web. IoT devices are intelligently linked together to enable new forms of communication between things and people, and among things.

These devices, or things, connect to the network. Using sensors, they provide the information they gather from the environment or allow other systems to reach out and act on the world through actuators. In the process, IoT devices generate massive amounts of data, and cloud computing provides a pathway, enabling data to travel to its destination.

### 1.2 Major Components

The IoT Cloud Connectivity Solution includes the following major components:

1. Devices or Sensors
2. Gateway
3. IoT Cloud services
4. End user application/system

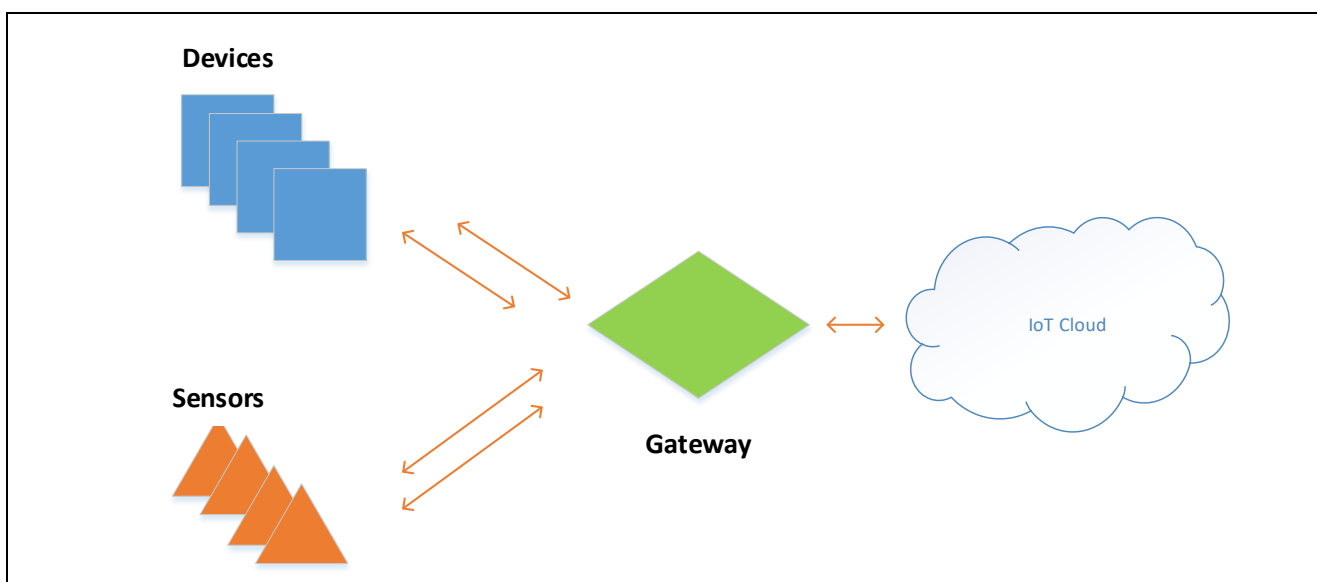


Figure 1. IoT Cloud Connectivity Architecture

#### Devices or Sensors

A device includes hardware and software that interact directly with the world. Devices connect to a network to communicate with each other, or to centralized applications. Devices may connect to the Internet either directly or indirectly.

#### Gateway

A gateway enables devices that are not directly connected to the Internet to reach cloud services. The data from each device is sent to the Cloud Platform, where it is processed and combined with data from other devices, and potentially with other business-transactional data. Most of the common communication gateways support one or more communication technologies such as Wi-Fi, Ethernet, or Cellular.

#### IoT Cloud

Many IoT devices produce lots of data. You need an efficient, scalable, affordable way to manage those devices, handle all that information, and make it work for you. When it comes to storing, processing, and analyzing data, especially big data, it is hard to surpass the cloud.

### 1.3 Cloud Provider Overview

#### 1.3.1 Amazon Web Services IoT Core

The AWS IoT Core service provides secure, bi-directional communication between IoT devices and the AWS Cloud over MQTT, HTTPS, and Web Sockets, enabling you to collect telemetry from multiple things, store the data, and analyze it.

AWS IoT Core is authenticated using TLS mutual authentication with X.509 certificates. Once a certificate is provisioned and activated, it can be installed on a device that then uses the certificate for all requests to device gateway.

The following diagram summarizes the service components and the flow of data.

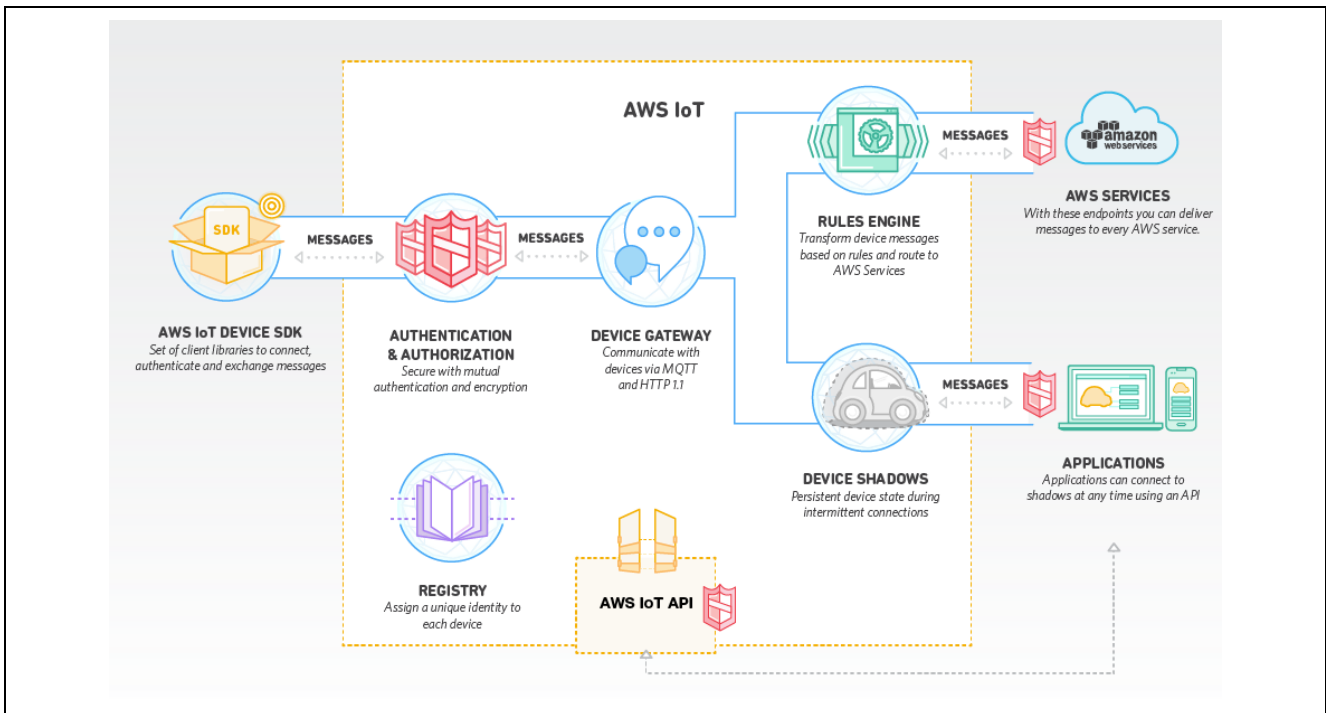


Figure 2. AWS IoT Cloud Solution

##### 1.3.1.1 Key Features

###### (1) AWS IoT Device SDK

The AWS IoT Device SDK enables your devices to connect, authenticate, and exchange messages with the IoT Core using MQTT, HTTPS, or Web Sockets protocols. The AWS IoT Device SDK supports C, C++, Java, Node.JS, Python, and Arduino Yun. It includes the client libraries, the developer guide, and the porting guide for manufacturers.

###### (2) Device Gateway

The AWS IoT Device Gateway enables devices to securely and efficiently communicate with the AWS IoT Core. The Device Gateway can exchange messages using a publish-subscribe model, enabling one-to-one and one-to-many communications. With this one-to-many communication pattern, the AWS IoT Core makes it possible for a connected device to broadcast data to multiple subscribers for a given topic.

The Device Gateway supports MQTT, Web Sockets, and HTTPS 1.1 protocols. The Device Gateway scales automatically to support over a billion devices without provisioning infrastructure.

###### (3) Authentication and Authorization

AWS IoT Core offers mutual authentication and encryption at all points of connection, so that data is never exchanged between devices and the AWS IoT Core without a proven identity. The AWS IoT Core supports the AWS method of authentication (called 'SigV4'), X.509 certificate-based authentication, and customer created token-based authentication (through custom authorizers). Connections using HTTPS can use any of these methods, while connections using MQTTs use certificate-based authentication, and connections using Web Sockets can use SigV4 or custom authorizers.

**(4) Registry**

The Registry establishes an identity for devices and tracks metadata such as the devices' attributes and capabilities. To each device, the Registry assigns a unique identity that is consistently formatted, regardless of the type of device or how it connects. The Registry also supports metadata that describes the capabilities of a device, for example, whether a sensor reports temperature data, and if the temperature scale is Fahrenheit or Celsius.

**(5) Device Shadows**

With the AWS IoT Core, you can create a persistent, virtual version, or a "shadow," of each device that includes the device's latest state. Applications, or other devices, can read these messages and interact with the device. Device Shadows persist in reporting the last state and the desired future state of each device, even when the device is offline. You can retrieve the device's last reported state, or set a desired future state, through the API, or using the Rules Engine.

**(6) Rules Engine**

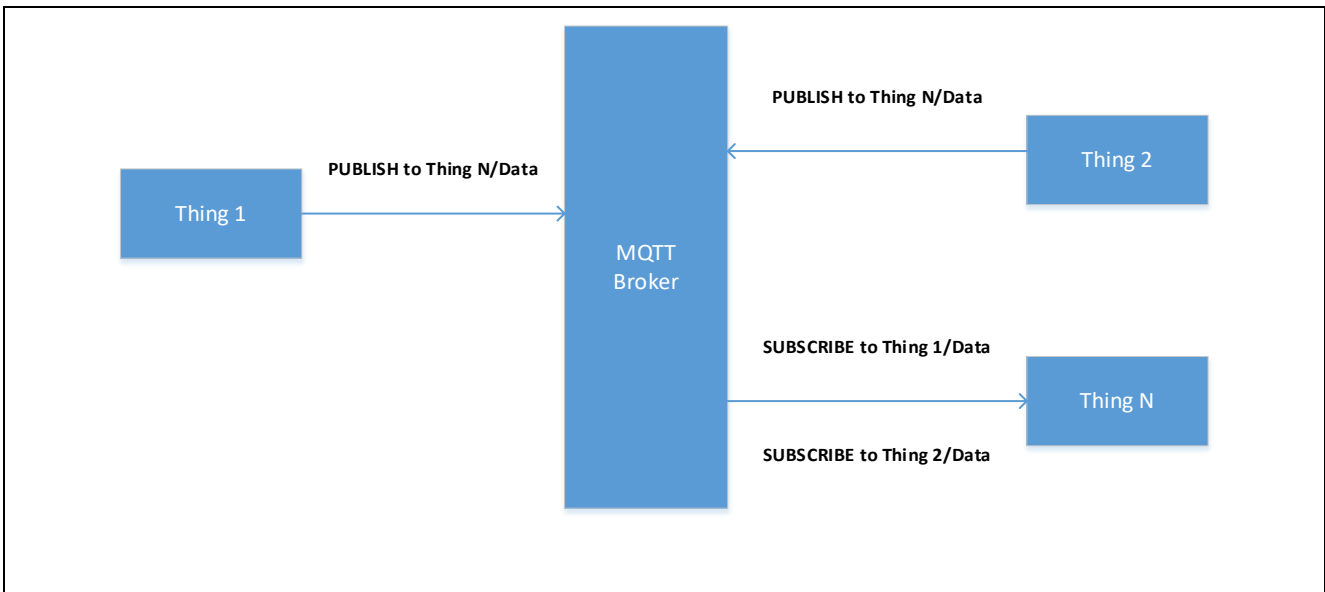
The Rules Engine makes it possible to build IoT applications that gather, process, analyze, and act on data generated by connected devices at a global scale, without having to manage any infrastructure. The Rules Engine evaluates inbound messages published into the AWS IoT Core and then transforms and delivers them to another device or a cloud service, based on business rules you define. A rule can apply to data from one or many devices, and it can take one or many actions in parallel.

The Rules Engine can also route messages to AWS endpoints, including AWS Lambda, Amazon Kinesis streams, Amazon S3, Amazon DynamoDB, Amazon CloudWatch, and Amazon Elasticsearch with built-in Kibana integration. External endpoints can be reached using AWS Lambda, Amazon Kinesis, and Amazon Simple Notification Service (SNS).

**1.4 MQTT Protocol Overview**

MQTT stands for MQ Telemetry Transport. MQTT is a Client Server publish-subscribe messaging transport protocol. It is an extremely light weight, open, simple messaging protocol, designed for constrained devices, as well as low-bandwidth, high-latency, or unreliable networks. These characteristics make it ideal for use in many situations, including constrained environments, such as communication in Machine to Machine (M2M) and IoT contexts, where a small code footprint is required, and/or network bandwidth is at a premium.

A MQTT client can publish information to other clients through a broker. A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for authentication, authorization of clients, as well as delivering published messages to any of its clients who subscribe to the topic. In this publisher/subscriber model, multiple clients may publish data with the same topic. A client will receive the messages published if the client subscribes to the same topic.



**Figure 3. MQTT Client Publish/Subscribe Model**

In this model, there is no direct connection between a publisher and the subscriber. To handle the challenges of a pub/sub system, the MQTT generally uses quality of service (QoS) levels. There are three QoS levels in MQTT:

- At most once (0)
- At least once (1)
- Exactly once (2)

#### At most once (0)

A message will not be acknowledged by the receiver or stored and redelivered by the sender.

#### At least once (1)

It is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once. The sender will store the message until it gets an acknowledgment in form of a PUBACK command message from the receiver.

#### Exactly once (2)

It is guaranteed that each message is received only once by the counterpart. It is the safest and the slowest quality of service level. The guarantee is provided by two flows there and back, between sender and receiver.

AWS IoT Core does not support QoS level 2.

## 1.5 TLS Protocol Overview

Transport Layer Security (TLS) protocol and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over a computer network.

The TLS/SSL protocol provides privacy and reliability between two communicating applications. It has the following basic properties:

**Encryption:** The messages exchanged between communicating applications are encrypted to ensure that the connection is private. Symmetric cryptography mechanism such as AES (Advanced Encryption Standard) is used for data encryption.

**Authentication:** A mechanism to check the peer's identity using certificates.

**Integrity:** A mechanism to detect message tampering and forgery to ensure that connection is reliable. Message Authentication Code (MAC) such as Secure Hash Algorithm (SHA) is used to ensure message integrity.

TLS/SSL uses TCP but provides secure communication for application layer protocols, such as HTTP and MQTT.

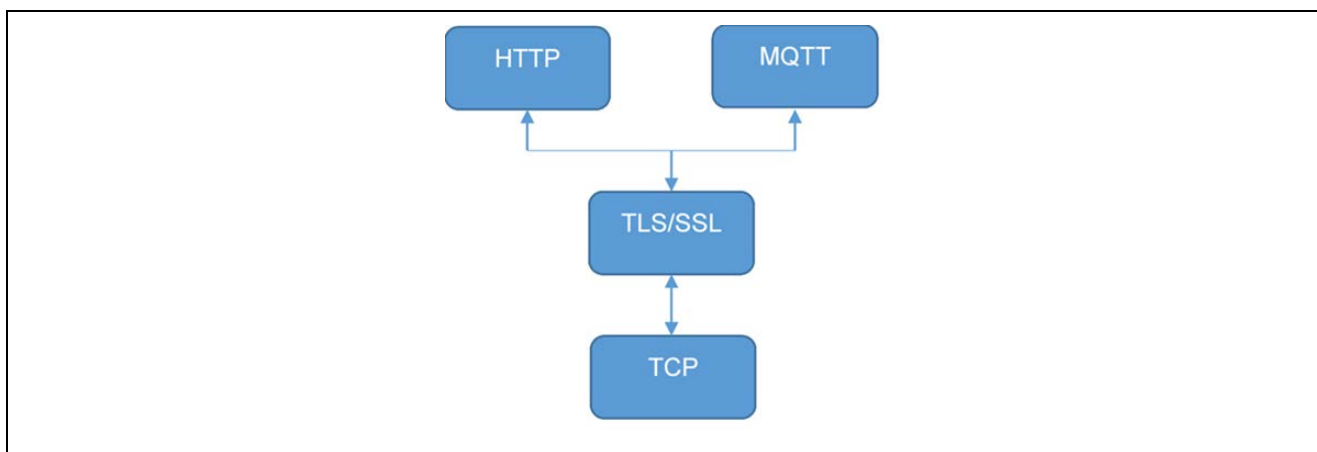


Figure 4. TLS/SSL Hierarchy

### 1.5.1 Device Certificates, and Keys

Devices certificates, public and private keys, and the ways they can be generated are discussed in this section.

#### 1.5.1.1 Device Certificates

Security is a critical concern when deploying and managing IoT devices. In general, each of the IoT devices needs an identity before they can communicate with the cloud. Digital certificates are the most common method for authenticating a remote host in TLS. Essentially, a digital certificate is a document with specific formatting that provides identity information for a device.

TLS normally uses a format called X.509, a standard developed by the International Telecommunication Union, though other formats for certificates may be used, if the TLS hosts can agree on the format to be used. X.509 defines a specific format for certificates and various encoding that can be used to produce a digital document. Most X.509 certificates used with TLS are encoded using a variant of ASN.1, another telecommunication standard. Within ASN.1 there are various digital encodings, but the most common encoding for TLS certificates is the Distinguished Encoding Rules (DER) standard. DER is a simplified subset of the ASN.1 Basic Encoding Rules (BER) that is designed to be unambiguous, making parsing easier.

Though DER-formatted binary certificates are used in the actual TLS protocol, they may be generated and stored in a number of different encodings, with file extensions such as `.pem`, `.crt`, and `.p12`. PEM is the most common of the alternative certificate encodings. The PEM format (from Privacy-Enhanced Mail) is a base64 encoded version of the DER encoding.

Depending on your application, you may generate your own certificates, be provided with certificates by a manufacturer or government organization, or purchase certificates from a commercial certificate authority.

#### 1.5.1.2 Loading Certificates onto your Device

To use a digital certificate in your NetX™ Secure application, you must first convert your certificate into a binary DER format, and optionally, convert the associated private key into a binary format; typically, a PKCS#1-formatted, DER-encoded RSA key. Once converted, it is up to you to load the certificate and the private key on to the device. Possible options include using a flash-based file system or generating a C array from the data (using a tool such as “xxd” from Linux® with the “-i” option), and then compiling the certificate and key into your application as constant data.

Once your certificate is loaded on the device, you can use the TLS API to associate your certificate with a TLS session.

### 1.5.2 Device Security Recommendations

The following security recommendations are not enforced by Cloud IoT Core but help you secure your devices and connections.

- The private key should be kept secret.
- Use TLS 1.2 when communicating with IoT Cloud to verify that the server certificate is valid using root certificate authorities.
- Each device should have a unique public/private key pair. If multiple devices share a single key and one of those devices is compromised, an attacker could impersonate all the devices that have been configured with that one key.
- Keep the public key secure when registering it with Cloud IoT Core. If an attacker can tamper with the public key and trick the provisioner into swapping the public key and registering the wrong public key, the attacker will subsequently be able to authenticate on behalf of the device.
- The key pair used to authenticate the device to Cloud IoT Core should not be used for other purposes or protocols.
- Depending on the device’s ability to store keys securely, key pairs should be rotated periodically. When practical, all keys should be discarded when the device is reset.
- If your device runs an operating system, make sure that you have a way to securely update it. Android Things provides a service for secure updates. For devices that do not have an operating system, ensure that you can securely update the device’s software if security vulnerabilities are discovered after deployment.



## 2. Synergy MQTT/TLS Cloud Solution

### 2.1 MQTT Client Overview

The NetX Duo MQTT Client module provides high-level APIs for a Message Queuing Telemetry Transport (MQTT) protocol-based client. The MQTT protocol works on top of TCP/IP and therefore the MQTT client is implemented on top of the NetX Duo IP and NetX Duo Packet pool. NetX Duo IP attaches itself to the appropriate link layer frameworks, such as Ethernet, Wi-Fi, or cellular.

The NetX Duo MQTT client module can be used in normal or in secure mode. In normal mode, the communication between the MQTT client and broker is not secure. In secure mode, the communication between the MQTT Client and broker is secured using the TLS protocol.

### 2.2 Design Considerations

- By default, the MQTT client does not use TLS; communication is not secure between a MQTT client and the broker.
- The Synergy MQTT client does not add the NetX Duo TLS session block. It only adds the NetX Duo TLS common block. This block defines/controls the common properties of NetX secure.
- It is the responsibility of the user/application code to create the TLS session, configure the security parameters, and load the relevant certificates manually under the TLS setup callback provided by `nxd_mqtt_client_secure_connect ()` API.

#### 2.2.1 Supported Features

NetX Duo MQTT Client supports the following features:

- Compliant with OASIS MQTT Version 3.1.1 Oct 29th, 2014. The specification can be found at <http://mqtt.org/>.
- Provides an option to enable/disable TLS for secure communication using NetX Secure in SSP.
- Supports QoS and provides the ability to choose the levels that can be selected while publishing the message.
- Internally buffers and maintains the queue of received messages.
- Provides a mechanism to register callback when a new message is received.
- Provides a mechanism to register callback when connection with the broker is terminated.

### 2.2.2 Operational Flow Sequence

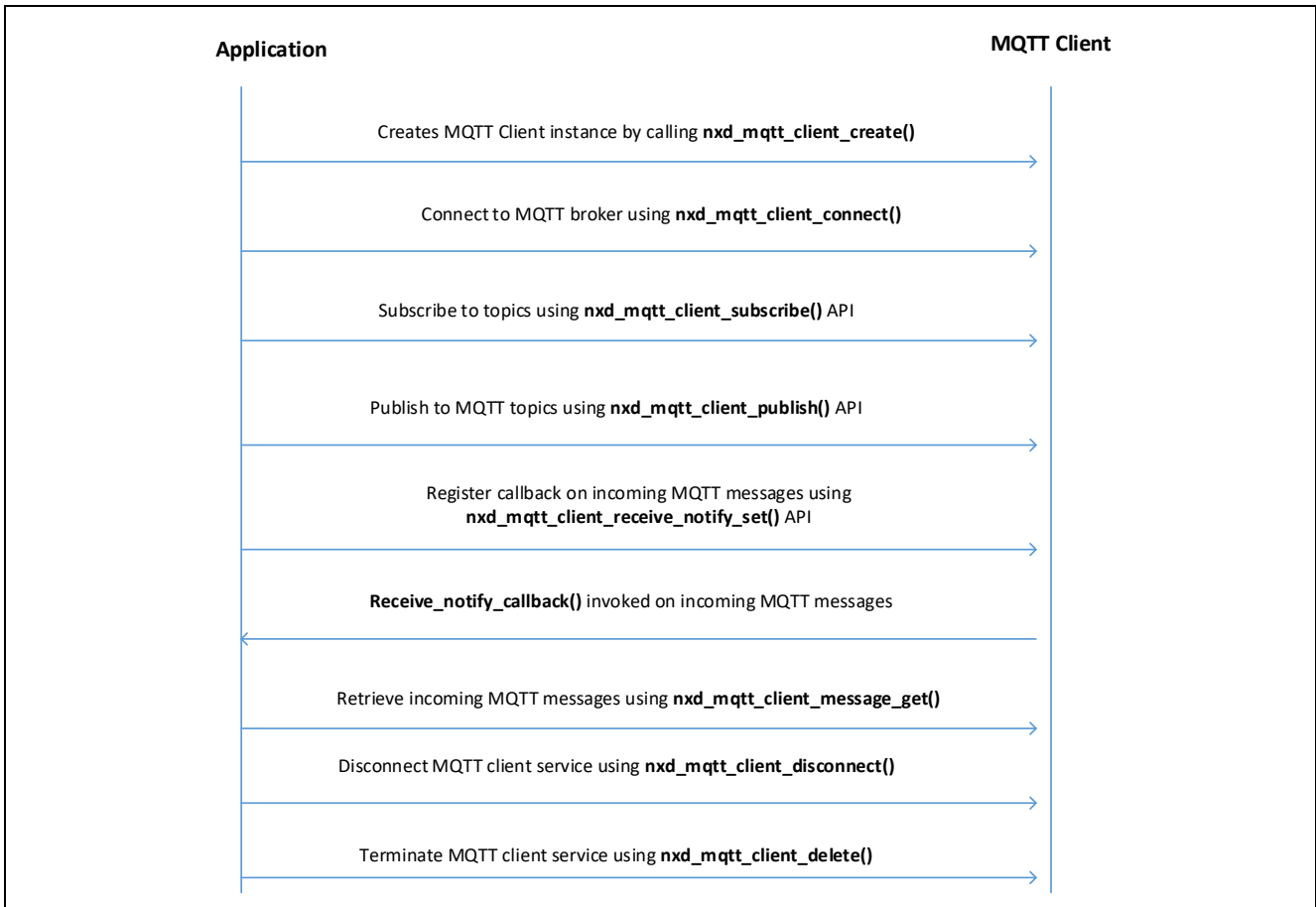


Figure 5. Synergy MQTT Client Flow Sequence

## 2.3 TLS Session Overview

The NetX Duo TLS session module provides high-level APIs for the TLS protocol-based client. It uses services provided by the Synergy Crypto Engine (SCE) to carry out hardware-accelerated encryption and decryption.

The NetX Duo TLS Session module is based on Express Logic’s NetX Secure which implements the Secure Socket Layer (SSL) and its replacement, TLS protocol, as described in RFC 2246 (version 1.0) and 5246 (version 1.2). NetX Secure also includes routines for the basic X.509 (RFC 5280) format. NetX Secure is intended for applications using ThreadX RTOS in the project.

### 2.3.1 Design Considerations

- NetX Secure TLS performs only basic path validation on incoming server certificates. Once the basic path validation is complete, TLS then invokes the certificate verification callback supplied by the application.
- It is the responsibility of the application to perform any additional validation of the certificate. To help with the additional validation, NetX Secure provides X.509 routines for common validation operations, including DNS validation and Certificate Revocation List checking.
- Software-based cryptography is processor-intensive. NetX Secure software-based cryptographic routines are optimized for performance, but depending on the power of the target processor, may result in very long operations. When hardware-based cryptography is available, it should be used for optimal performance of the NetX secure TLS.
- Due to the nature of embedded devices, some applications may not have the resources to support the maximum TLS record size of 16 KB. NetX Secure can handle 16 KB records on devices with sufficient resources.

### 2.3.2 Supported Features

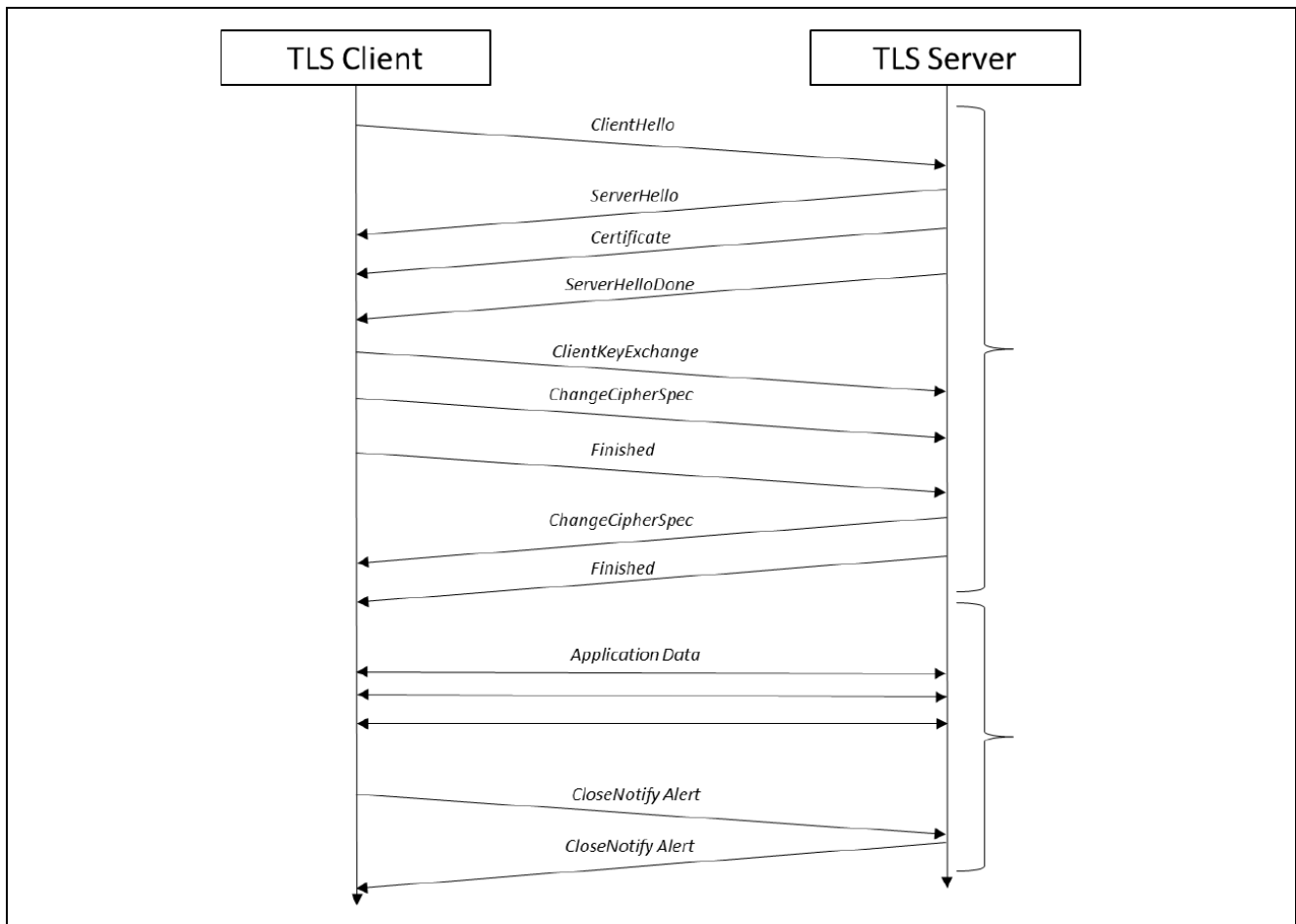
- Support for RFC 2246, TLS Protocol Version 1.0
- Support for RFC 5246, TLS Protocol Version 1.2
- Support for RFC 5280 X.509 PKI Certificates (v3)
- Support for RFC 3268 AES Cipher suites for TLS
- RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
- RFC 2104 HMAC: Keyed-Hashing for Message Authentication
- RFC 6234 US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
- RFC 4279 Pre-Shared Key Cipher suites for TLS

### 2.3.3 Operational Flow Sequence

This section describes the TLS handshake operational sequence.

#### 2.3.3.1 TLS Handshake

The following figure shows a typical TLS handshake between the TLS Server and Client.



**Figure 6. TLS Handshake**

- A TLS handshake begins when the TLS client sends a **ClientHello** message to a TLS server, indicating its desire to start a TLS session.
- The message contains information about the encryption the client would like to use for the session, along with information used to generate the session keys.
- The TLS server responds to the **ClientHello** with a **ServerHello** message, indicating a selection from the encryption options provided by the client.
- It is followed by a Certificate message, in which the server provides a digital certificate to authenticate its identity to the client.
- Finally, the server sends a **ServerHelloDone** message to indicate that it has no more messages to send.

- Once Client has received all the server’s messages, it has enough information to generate the session keys. TLS does this by creating a shared bit of random data called the Pre-Master Secret, which is a fixed size and is used as a seed to generate all the keys needed once encryption is enabled.
- The Pre-Master Secret is encrypted using the public key algorithm (such as RSA) specified in the Hello messages and the public key provided by the server in its certificate.
- The encrypted Pre-Master Secret is sent to the server in the **ClientKeyExchange** message. The server, upon receiving the **ClientKeyExchange** message, decrypts the Pre-Master Secret using its private key and proceeds to generate the session keys in parallel with the TLS client.
- Once the session keys are generated, all further messages can be encrypted using the private-key algorithm (such as AES) selected in the Hello messages. One final un-encrypted message called **ChangeCipherSpec** is sent by both the client and server to indicate that all further messages will be encrypted.
- The first encrypted message sent by both the client and server is also the final TLS handshake message, called **Finished**. This message contains a hash of all the handshake messages received and sent. This hash is used to verify that none of the messages in the handshake have been tampered with or corrupted.
- Now the application begins sending and receiving data. All data — sent by either side — is first hashed using the hash algorithm chosen in the Hello messages, and then encrypted using the chosen private -key algorithm with the generated session keys.
- Finally, a TLS session can only be successfully ended if either the Client or Server chooses to do so. Both the client and server must send and process a **CloseNotify** alert for a successful session shutdown.

### 2.3.3.2 Initialization Flow Sequence

A typical TLS session initialization flow sequence is shown in the following figure.

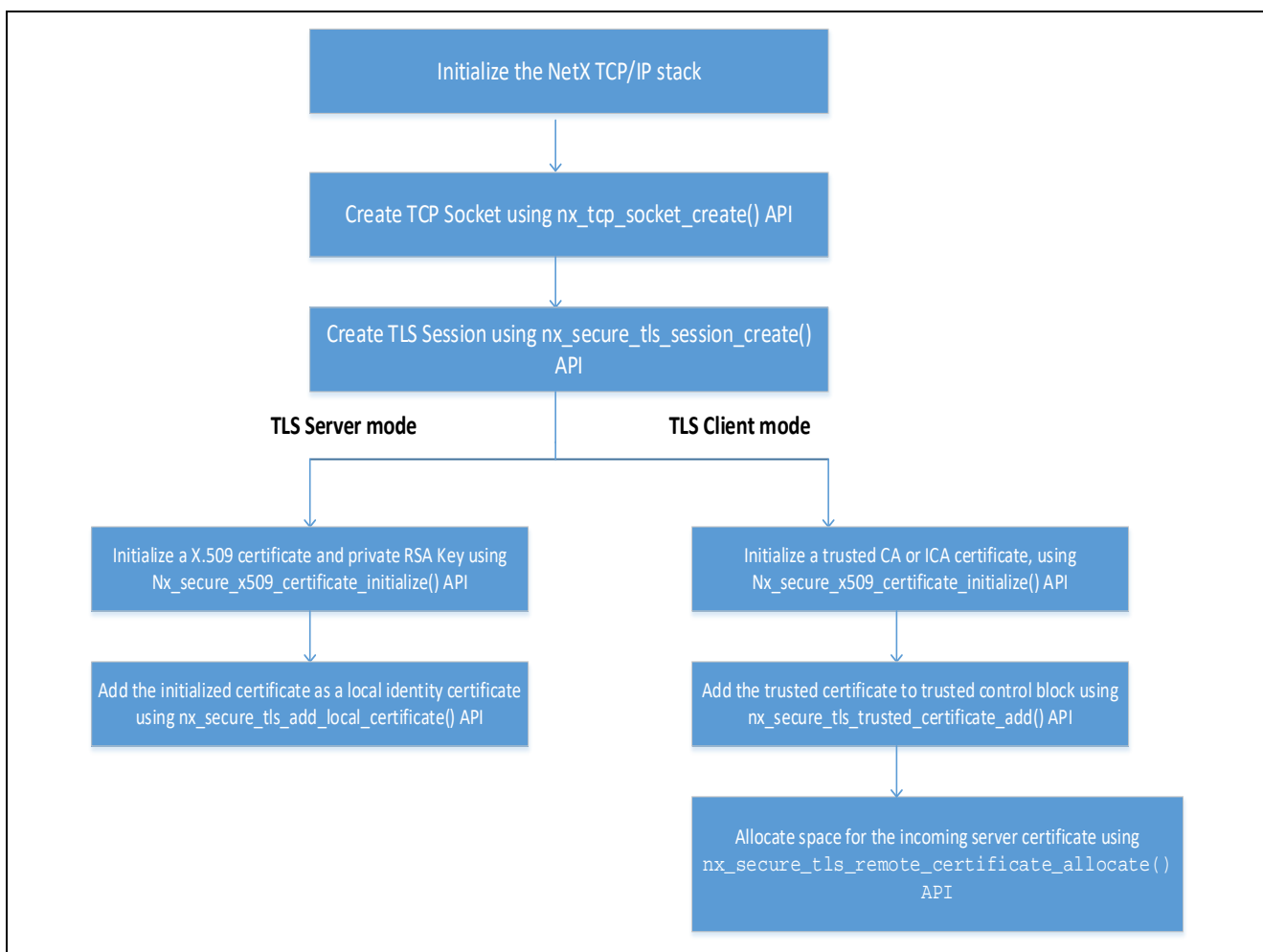


Figure 7. Synergy TLS Session Initialization

### 2.3.3.3 Data Communication Flow Sequence

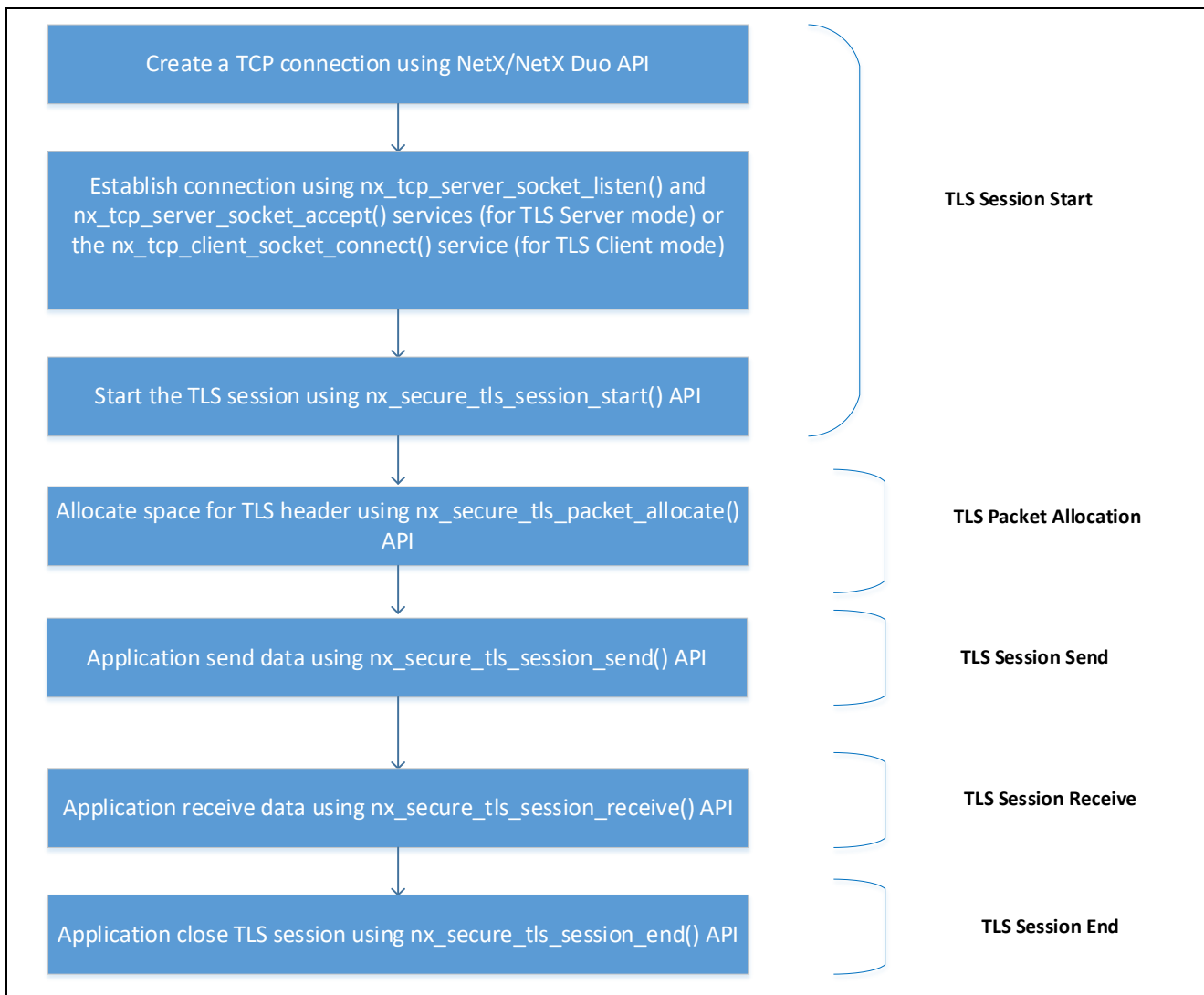


Figure 8. Synergy TLS Session Data Flow Sequence

## 3. MQTT/TLS Application Example

### 3.1 Application Overview

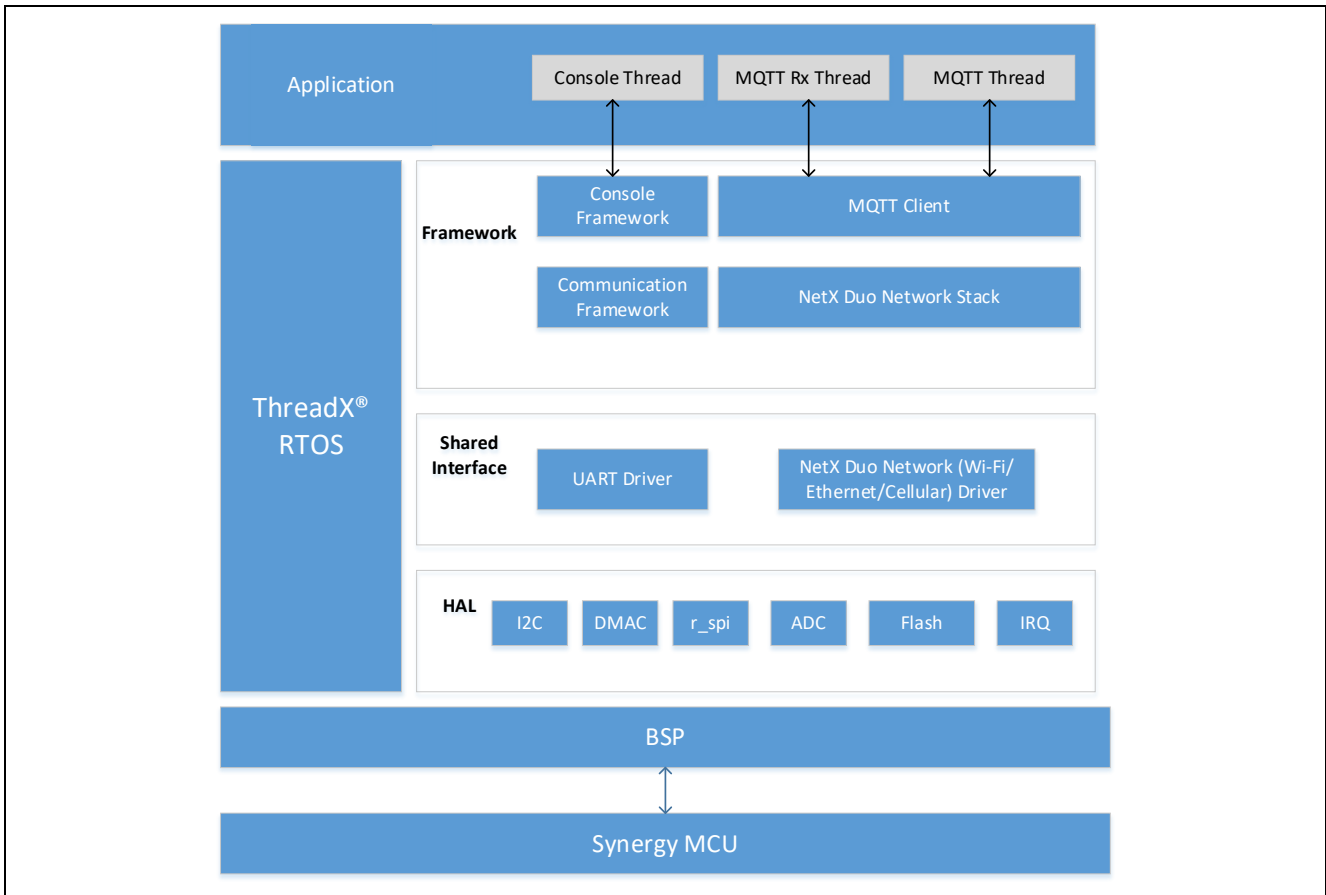
This example application project demonstrates the Renesas Synergy™ IoT Cloud Connectivity solution using the onboard Synergy MQTT/TLS modules. For demonstration purposes, this application uses Amazon Web Services (AWS) as the cloud provider. Ethernet or Wi-Fi or Cellular (supported only on AE-CLOUD2 kit) is used as the primary communication interface between the Thing and AWS IoT Core.

In this example, the AE-CLOUD1 and AE-CLOUD2 kit acts as an MQTT node/Thing, connects to the AWS IoT Core, and it periodically reads on-board sensor values and publishes this information to the AWS IoT Core. It also subscribes to its User LED state MQTT topic. You can turn the User LEDs ON/OFF by publishing the LED state remotely. This application reads the updated LED state and turns the User LEDs ON/OFF.

The steps here use the MQTT Client from AWS IoT Core to subscribe to the MQTT topics published by AE-CLOUD1/AE-CLOUD2 kit. Follow the instructions in section 3.3 to setup the MQTT client on AWS IoT Core and run this demonstration. However, you are free to use any known MQTT client to subscribe to the MQTT topics published by AE-CLOUD1/AE-CLOUD2 Synergy MCU kit.

### 3.2 Software Architecture Overview

Figure 9 shows the software architecture for the Synergy Cloud Connectivity Application Example Project.



**Figure 9. Synergy Cloud Connectivity Application Software Architecture**

The main software components of this application are:

- MQTT Client
- NetX Duo IP Stack and its underlying driver components for Ethernet, Cellular and Wi-Fi.
- Console Framework

This application contains the following application threads:

- Console Thread
- MQTT Thread
- MQTT Rx Thread

#### 3.2.1 Console Thread

This thread handles the function related to Command Line Interface (CLI). It uses the console framework, which in-turn, uses the communication framework and its underlying USBX CDC device module components.

This thread reads the user inputs and stores them in the internal data flash. The stored information is read later by the MQTT Thread when it tries to run the Synergy Cloud connectivity demo.

This thread presents you with the following CLI command options.

- `Cwiz`
- `Demo start/stop`

#### **Cwiz** command option

Using this command option, you can select the following configurations:

- Network interface such as Ethernet, Wi-Fi, and its associated IP mode (DHCP/Static).

- IoT cloud selection (AWS).
- Dump the existing configuration from flash.
- Exit the menu.

#### Demo start/stop command option

Using this command option, you can run/stop the Synergy Cloud Connectivity Demonstration.

### 3.2.2 MQTT Thread

The MQTT thread is the main control thread that handles the following major functions:

- Initialize communication interface (Ethernet/Wi-Fi).
- Initialize IoT Cloud interface.
- Read sensor data and publish the data periodically on MQTT topics.
- Update the user LED state based on the type of MQTT message received.

On wakeup, this thread periodically checks (every 5 seconds) for user input event flag state set once you enter the demonstration start/stop command on the CLI. If the demonstration start command is issued from the CLI, this thread will read the pre-configured user information from internal flash and check its validity. If the content is valid, it then starts the Synergy Cloud connectivity demonstration. If a demo stop command is issued, it de-initializes the IoT cloud interface.

### 3.2.3 MQTT Rx Thread

This thread handles the incoming MQTT messages from the MQTT broker. On receiving the new MQTT message, the user callback `receive_notify_callback()` will be invoked by the MQTT thread. This callback in turn sets the semaphore on which the MQTT Rx Thread is polling periodically.

On receiving the new MQTT message, it uses the `nxd_mqtt_client_message_get()` API to read the message, parses it, and acts on it based on the type of the message received.

## 3.3 IoT Cloud Configuration (AWS)

### 3.3.1.1 AWS IoT Policies

AWS IoT Core policies are JSON (JavaScript Object Notation) documents that authorize your device to perform AWS IoT Core operations. AWS IoT defines a set of policy actions describing the operations and resources for which you can grant or deny access. For example:

- `IoT:Connect` represents permission to connect to the AWS IoT message broker.
- `IoT:Subscribe` represents permission to subscribe to an MQTT topic or topic filter.
- `IoT:GetThingShadow` represents permission to get a thing shadow.

### JSON

JSON is an open standard, lightweight, data-interchange format. As a text document, it is easy for users to read and write, and for machines to parse and generate.

JSON is completely language independent, using conventions that are familiar to C-family programmers, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. The following example shows a JSON script used to turn on an LED.

```
{
  "state": {
    "desired": {
      "LED_value": "On"
    }
  }
}
```

### AWS IoT Thing Shadow

A Thing Shadow (also referred to as a Device Shadow) is a JSON document used to store and retrieve current state information for a Thing (device, application, and so on).

The Thing Shadow service maintains a thing shadow for each thing you connect to AWS IoT Core. You can use thing shadows to get and set the state of a thing over MQTT or HTTP, regardless of whether the thing is connected to the Internet. Each thing shadow is uniquely identified by its name.

### Amazon Web Services Signup

Amazon Web Services offers a free account (12 months) for each user. It is expected that you create an account on the AWS IoT Cloud service before continuing to the next section.

To create an AWS account, open to the following link in your web browser:

<https://portal.aws.amazon.com/billing/signup#/start>

Fill in the required details and create a user account.

Note: While creating the project, certificates and policies, the screenshots may look slightly different from what is shown in the document and users need to navigation in the AWS IoT core environment to find corresponding attributes while working on this project.

### 3.3.2 Creating a Device on AWS IoT Core

The following steps show you how to create a device on the IoT Core user account. It is assumed that you created a user account in the AWS IoT Core and have followed the AWS signup procedure.

#### 3.3.2.1 Open AWS IoT Core Service

1. Connect to the AWS IoT service by typing **IoT Core** in the AWS services search bar.
2. Click **IoT Core**.

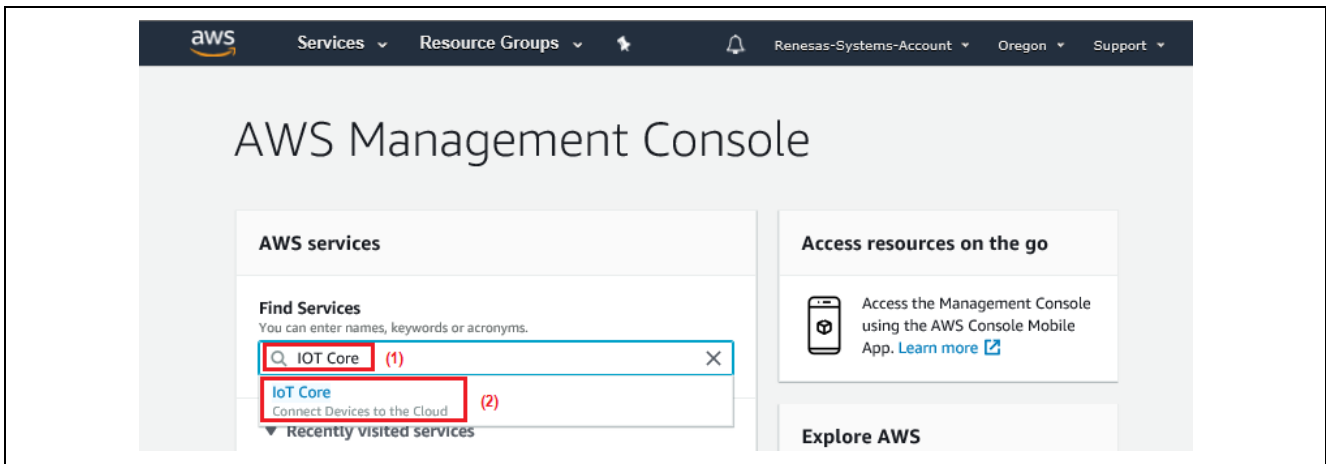
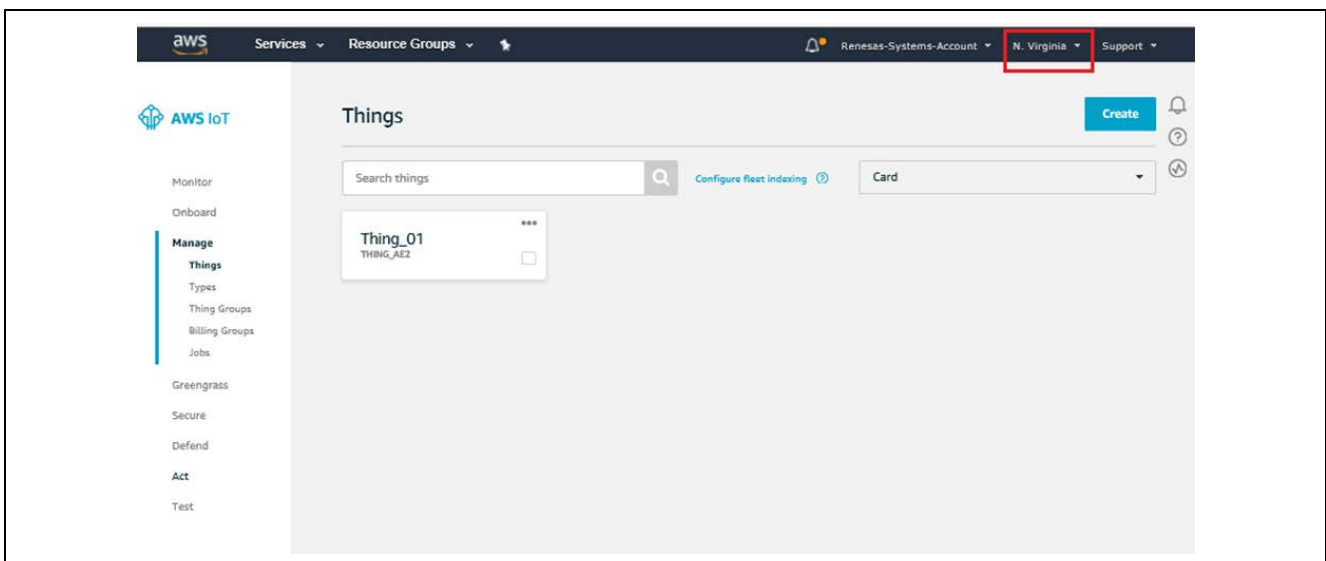


Figure 10. Synergy Cloud Connectivity Application Software Architecture

#### 3.3.2.2 Create a Thing

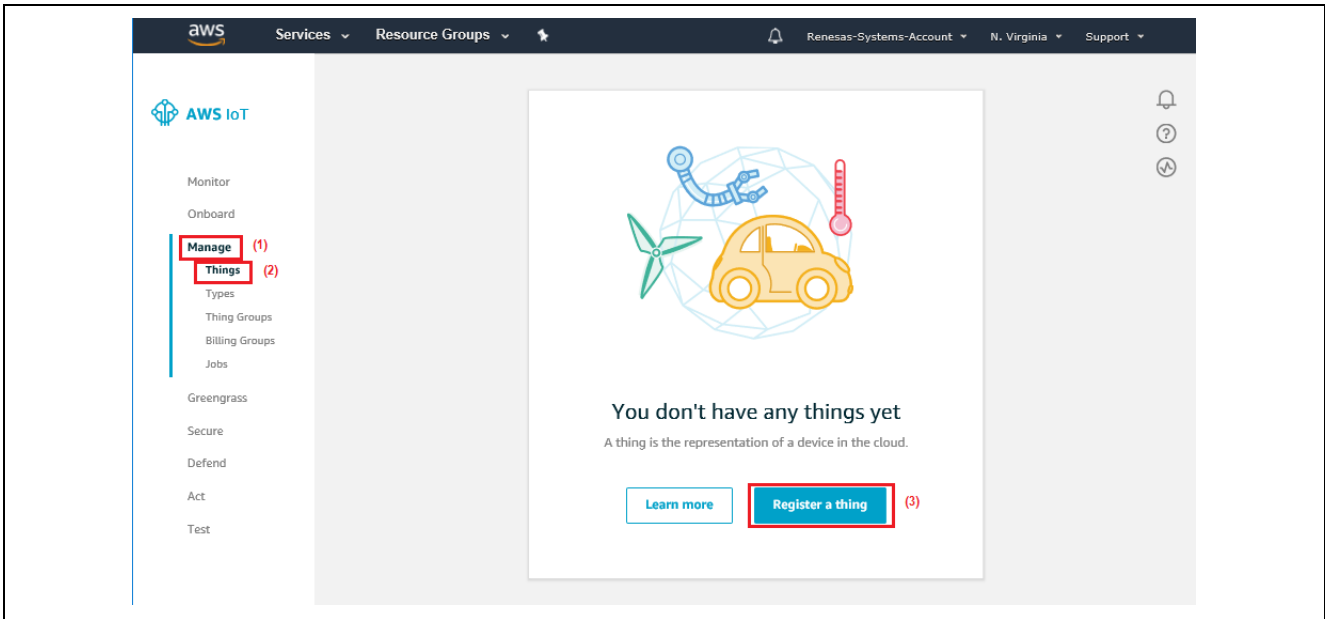
1. Start creating your device by selecting **Manage**. Be sure to select the appropriate region in the AWS console on the top right corner.

Note: A Thing created in one region will not be seen in another region.

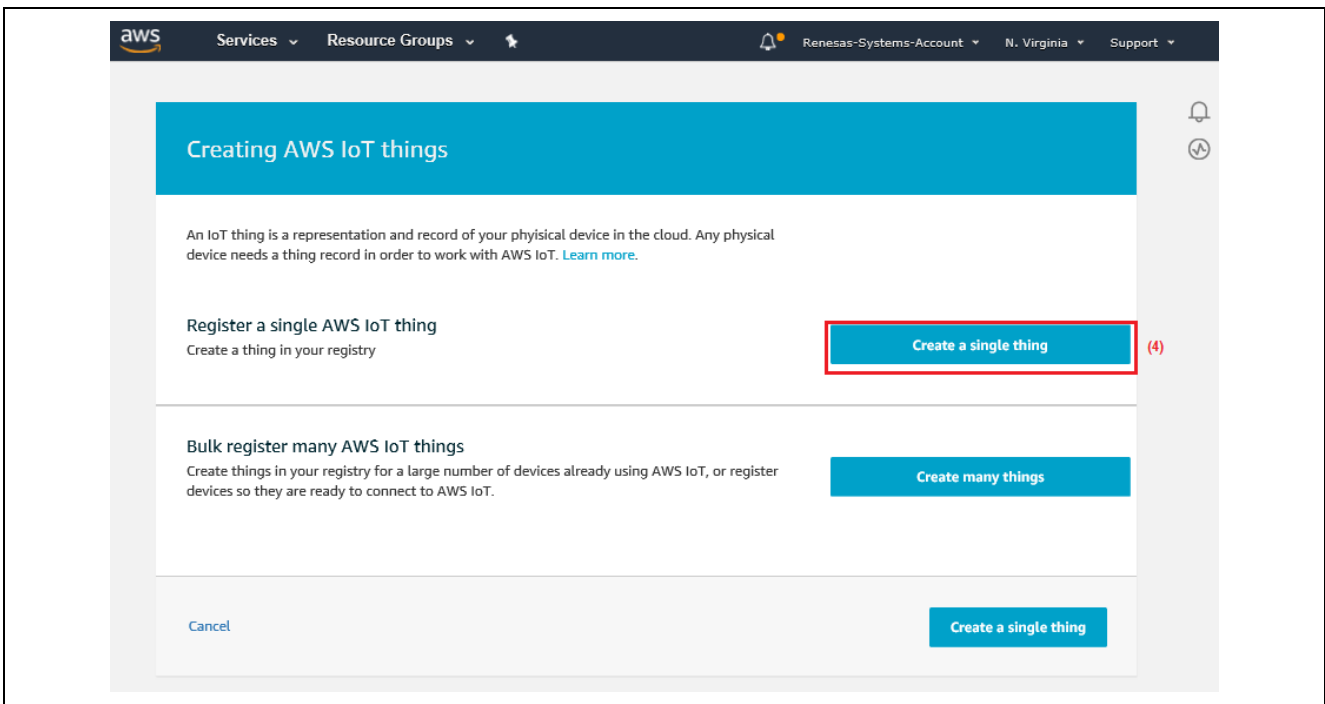




2. Now select **Things**.
3. Next, select **Register a thing** to create a thing.



4. Then select the **Create a single thing** button.



5. Enter the **Thing Name**. In the example, a Thing by name `Thing_01` is created.

Note: Remember to store the **Thing Name**. This information is passed to firmware using the serial console during configuration.

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**

Thing\_01 (5)

---

**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.

<b>Attribute key</b>	<b>Value</b>
Provide an attribute key, e.g. Manufacturer	Provide an attribute value, e.g. Acme-Corporation
	Clear

Add another

**Show thing shadow** ▾

Cancel

Back

Next

6. Create a Thing type by clicking the **Create a type** button.

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

(6)

---

**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	
Provide an attribute key, e.g. Manufacturer	Provide an attribute value, e.g. Acme-Corporation	Clear
<div style="border: 1px solid #0099cc; padding: 5px; display: inline-block; color: #0099cc;">Add another</div>		

**Show thing shadow** ▾

Cancel

Back

Next

7. Enter the **Type Name** and **Description**. Add the attributes by clicking the button **Add another** in the **Set searchable thing attributes** section.

Create a thing type

This will help you organize, categorize, and search for your things.

**Name**

Thing\_AE2 (7)

**Description**

AE-Cloud2 kit (7)

---

**Set searchable thing attributes**

You can define up to three attributes for a thing type. Things associated with this type can be searched by using these fields.

Add another (7)

---

**Tags**

Apply tags to your resources to help organize and identify them. A tag consists of a case-sensitive key-value pair. [Learn more](#) about tagging your AWS resources.

<p><b>Tag name</b></p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">                 Provide a tag name, e.g. Manufacturer             </div>	<p><b>Value</b></p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">                 Provide a tag value, e.g. Acme-Corporation             </div>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Add another

Cancel
Create thing type

8. Add the attribute key and click the **Create thing type** button.

## Create a thing type

This will help you organize, categorize, and search for your things.

**Name**

**Description**

Temperature

(8)

Remove

C..
Cancel
Create thing type (8)

9. Select the **Thing Type** and enter the attribute value. Click the **Next** button.

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**

Thing\_AE2 (9)

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
temperature	<div style="display: flex; align-items: center;"> <div style="border: 2px solid red; padding: 1px; margin-right: 5px;">25</div> <span>(9)</span> </div>

---

**Set non-searchable thing attributes (optional)**

You can use thing attributes to describe the identity and capabilities of your device.

Attribute key	Value	
Provide an attribute key, e.g. Manufacturer	Provide an attribute value, e.g. Acme-Corporation	<div style="border: 1px solid #ccc; padding: 2px 10px; color: #A52A2A; font-weight: bold;">Clear</div>
<div style="border: 1px solid #ccc; padding: 2px 10px; color: #00AEEF; font-weight: bold; display: inline-block;">Add another</div>		

**Show thing shadow** ▼

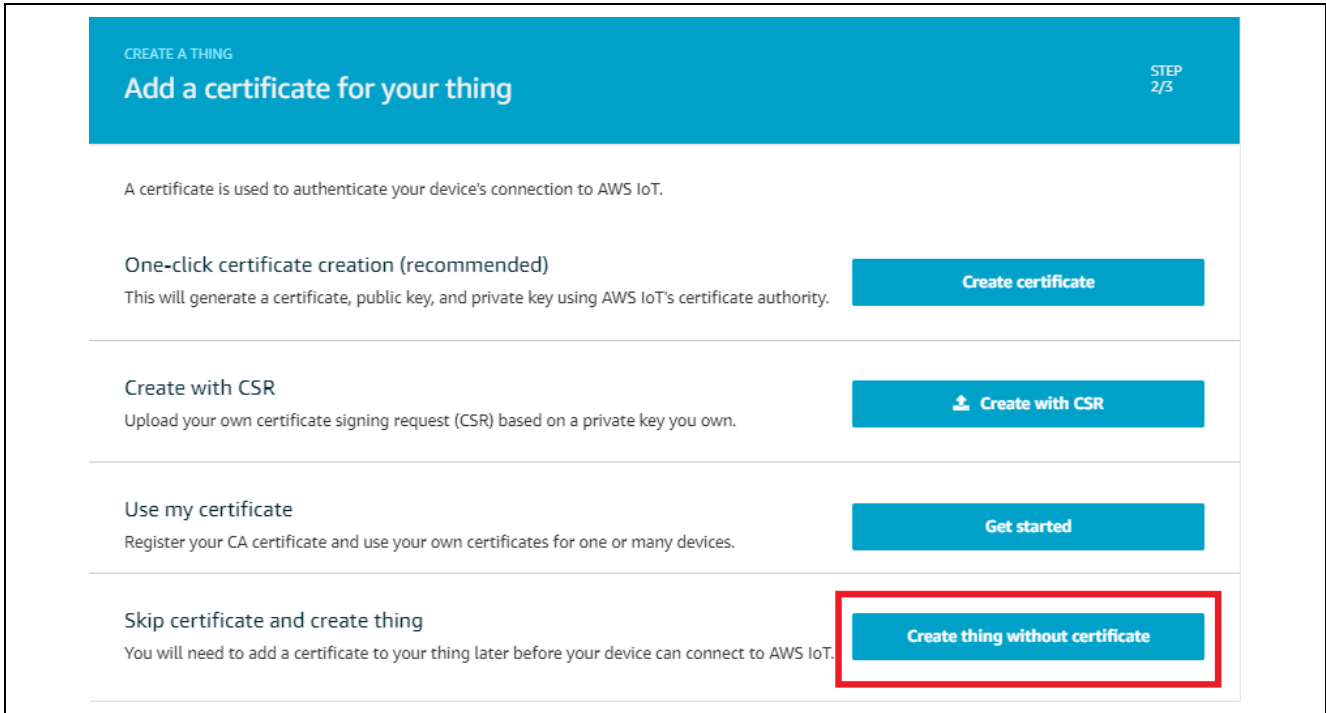
Cancel

Back

Next

(9)

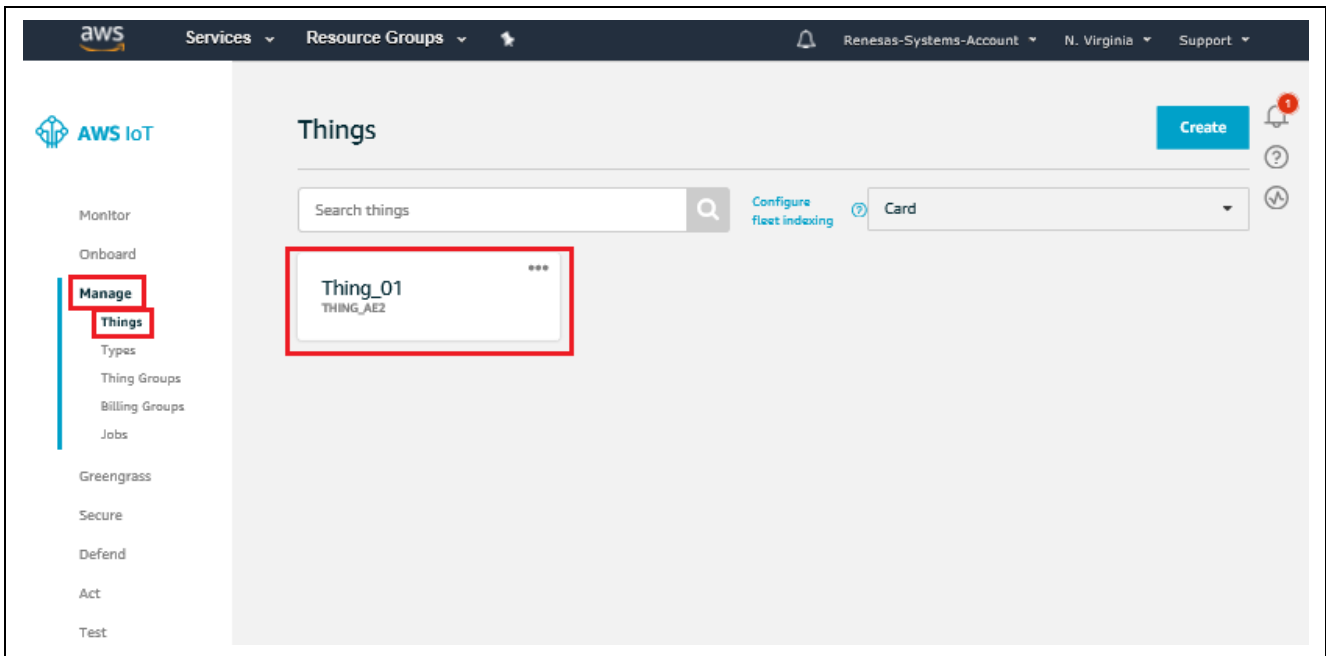
10. Click the option, **Create thing without certificate**, to create a thing in AWS IoT.



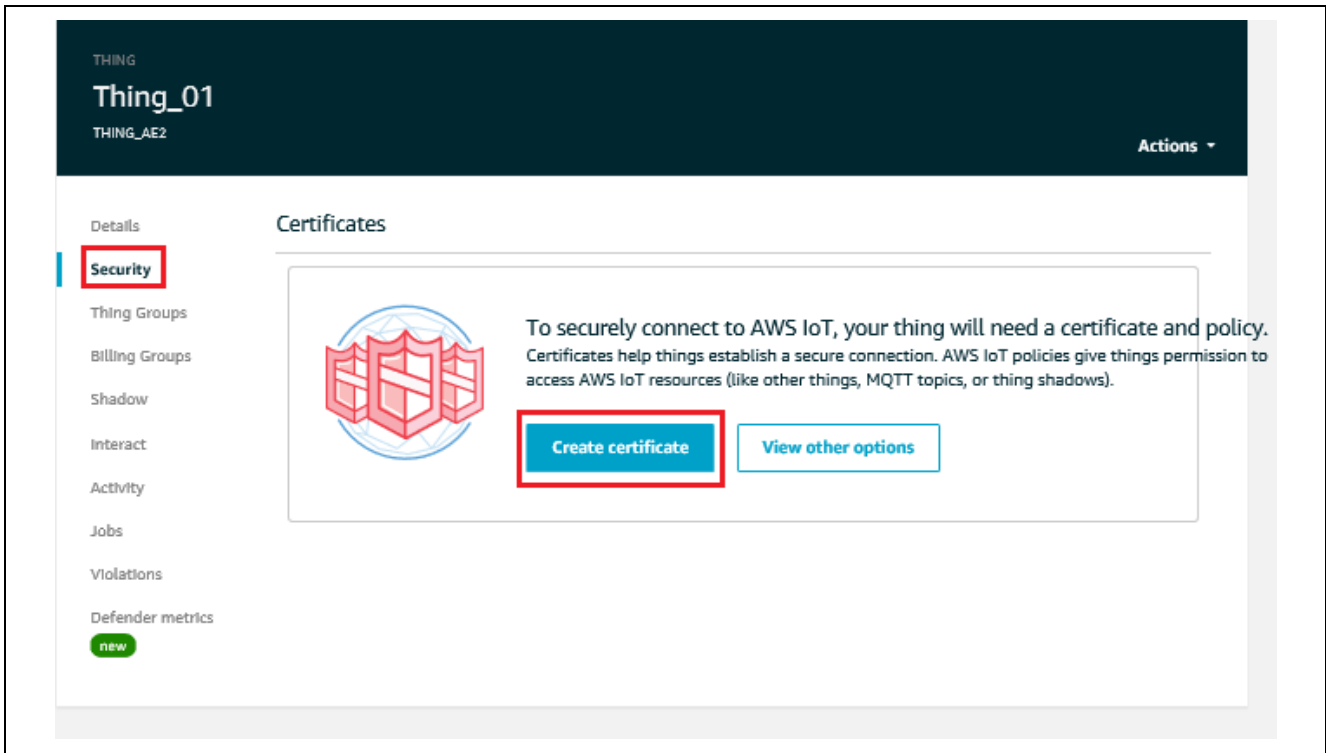
### 3.3.3 Generating Device Certificate and Keys

At this point, it is assumed that the AWS IoT Thing has been created following the above instructions. Now you can generate device certificates and keys for the AWS IoT Thing (Thing) created.

The Thing you created appears in the **Things** section, as shown in the following screen.



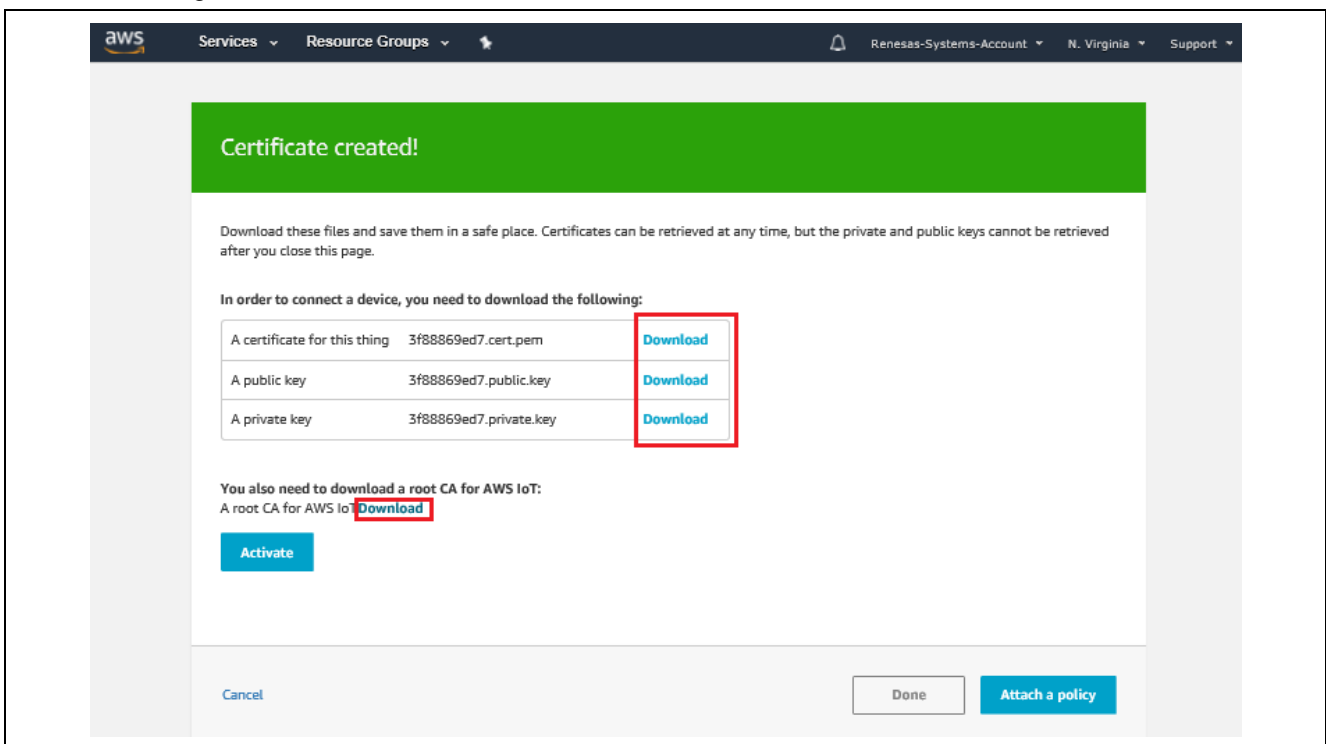
1. Click the Thing you created. It will open in a new window with the Thing information.  
In the example, the Thing created is called `Thing_01`.
2. Go to the **Security** tab and click **Create certificate** button, as shown in the following screen.



It generates the following for the Thing you created, as shown in the following screen.

- A device certificate
- A public key
- A private key
- A root CA for AWS IoT

3. To download certificates, click the **Download** button next to each of the certificates and keys, as shown in the following screen.





- Click the **Download** button. The root CA for AWS opens the link <https://docs.aws.amazon.com/iot/latest/developerguide/managing-device-certs.html#server-authentication>. Click **RSA 2048 bit key: Amazon Root CA 1**, as shown in the following figure to open the certificate in a new tab.

Note: Certificate generated by right clicking and downloading it to the PC might not work.

AWS Documentation » AWS IoT » Developer Guide » Security and Identity for AWS IoT » AWS IoT Authentication » X.509 Certificates » X.509 Certificates and AWS IoT

## X.509 Certificates and AWS IoT

### Client Authentication

AWS IoT can use AWS IoT-generated certificates or certificates signed by a CA certificate for device authentication. Certificates generated certificates signed by a CA certificate are set when the certificate is created.

#### Note

We recommend that each device be given a unique certificate to enable fine-grained management including certificate revocation.

Devices must support rotation and replacement of certificates in order to ensure smooth operation as certificates expire.

To use a certificate that is not created by AWS IoT, you must register a CA certificate. All device certificates must be signed by the CA cert

You can use the AWS IoT console or CLI to perform the following operations:

- Create and register an AWS IoT certificate.
- Register a CA certificate.
- Register a device certificate.
- Activate or deactivate a device certificate.
- Revoke a device certificate.
- Transfer a device certificate to another AWS account.
- List all CA certificates registered to your AWS account.
- List all device certificates registered to your AWS account.

For more information about the CLI commands to use to perform these operations, see [AWS IoT CLI Reference](#).

For more information about using the AWS IoT console to create certificates, see [Create and Activate a Device Certificate](#).

### Server Authentication

Server certificates allow your devices to verify that they're communicating with AWS IoT and not another server impersonating AWS IoT. : SDKs. AWS IoT server certificates are signed by one of the following CA certificates:

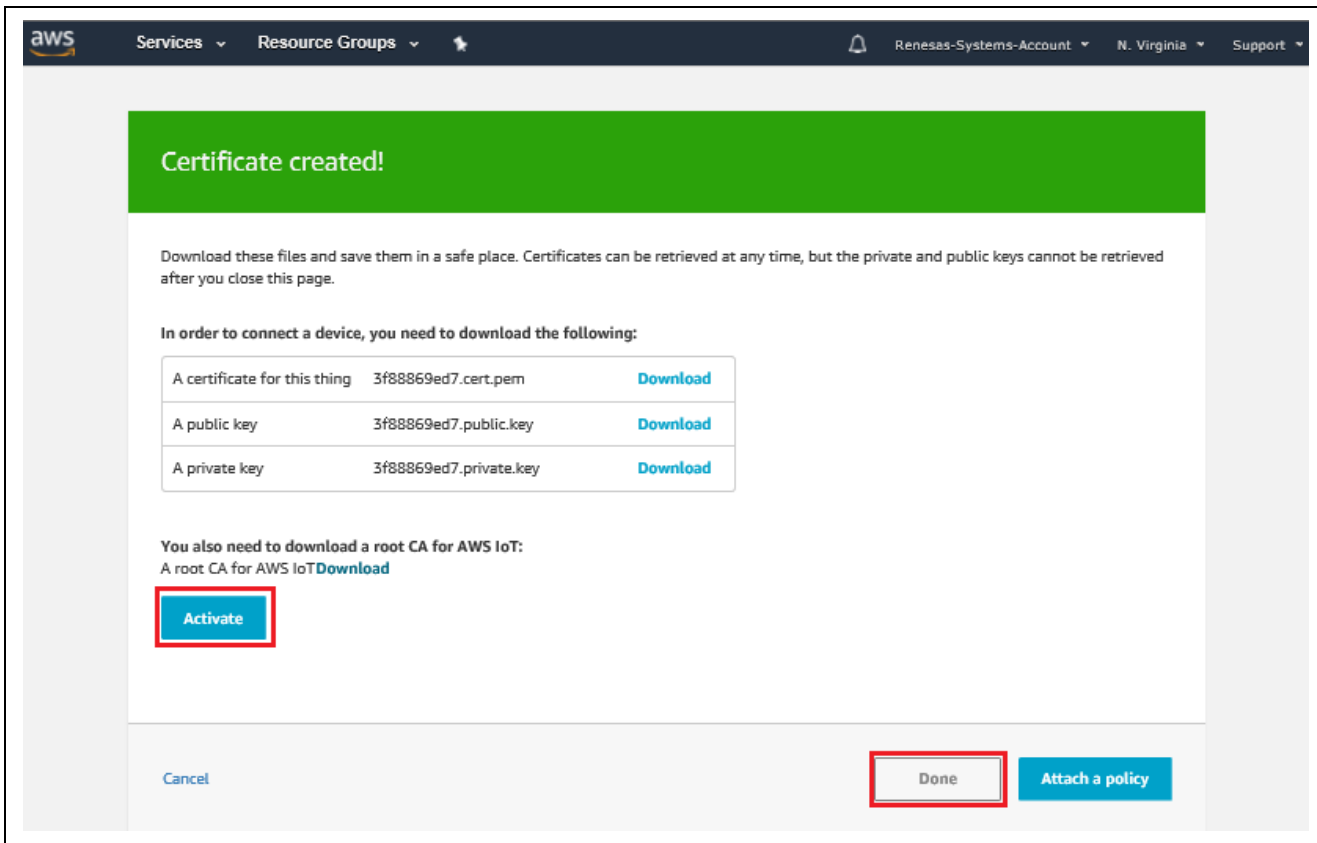
#### VeriSign Endpoints (legacy)

- RSA 2048 bit key: [VeriSign Class 3 Public Primary G5 root CA certificate](#)

#### Amazon Trust Services Endpoints (preferred)

- RSA 2048 bit key: [Amazon Root CA 1](#)
- RSA 4096 bit key: [Amazon Root CA 2](#)
- ECC 256 bit key: [Amazon Root CA 3](#)
- ECC 384 bit key: [Amazon Root CA 4](#)

5. Go back to the AWS console and click the **Activate** button to activate the certificate just created.
6. After certificate activation, click the **Done** button to complete the certification/keys creation.



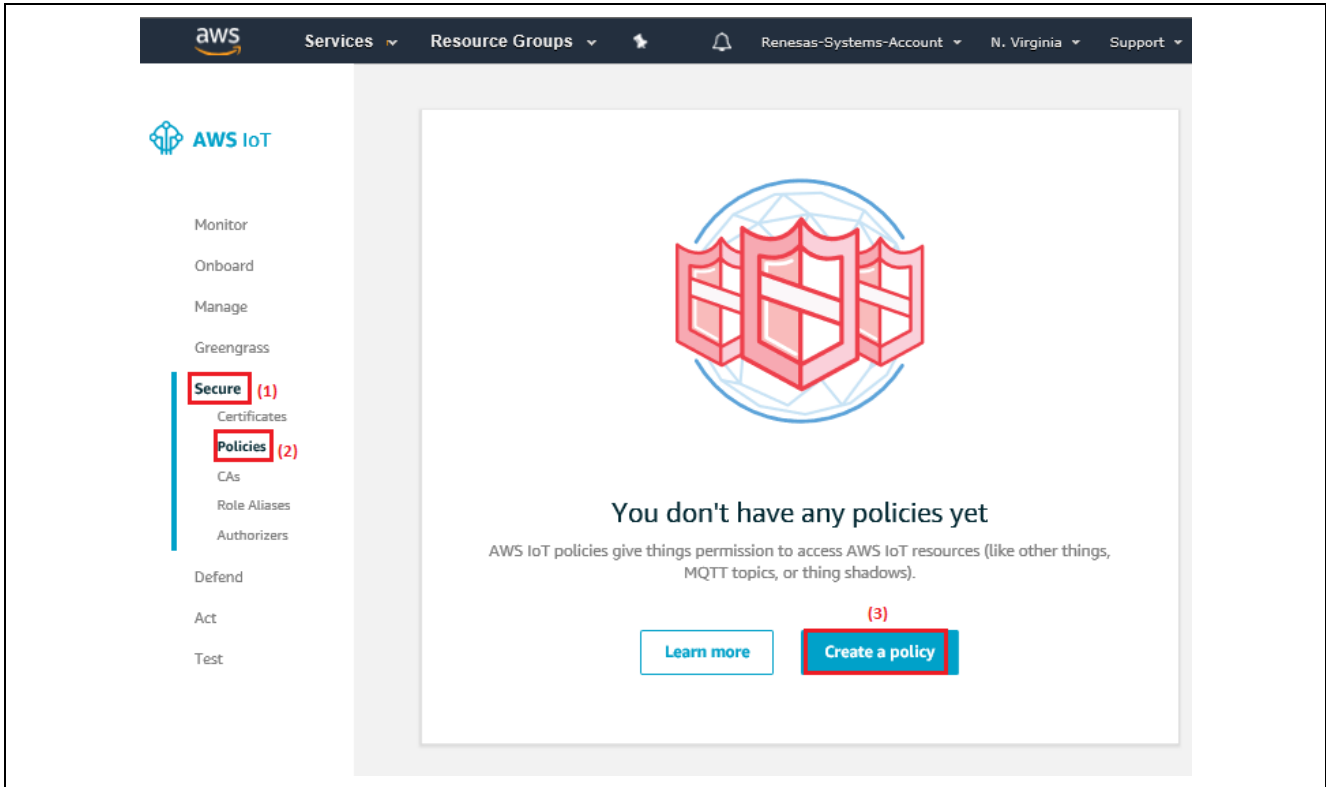
Note: Since October 2018, AWS has recommended that all users create an Amazon Trust Services (ATS) endpoint and load these CA certificates onto their devices. The Amazon Root CA1 can be downloaded from: <https://docs.aws.amazon.com/iot/latest/developerguide/managing-device-certs.html>. For users who created endpoints prior to October 2018 and are still using them to test this AP, it is recommended to use the `rootCA.pem` file given as part of this package.

### 3.3.4 Creating a Policy for your Device

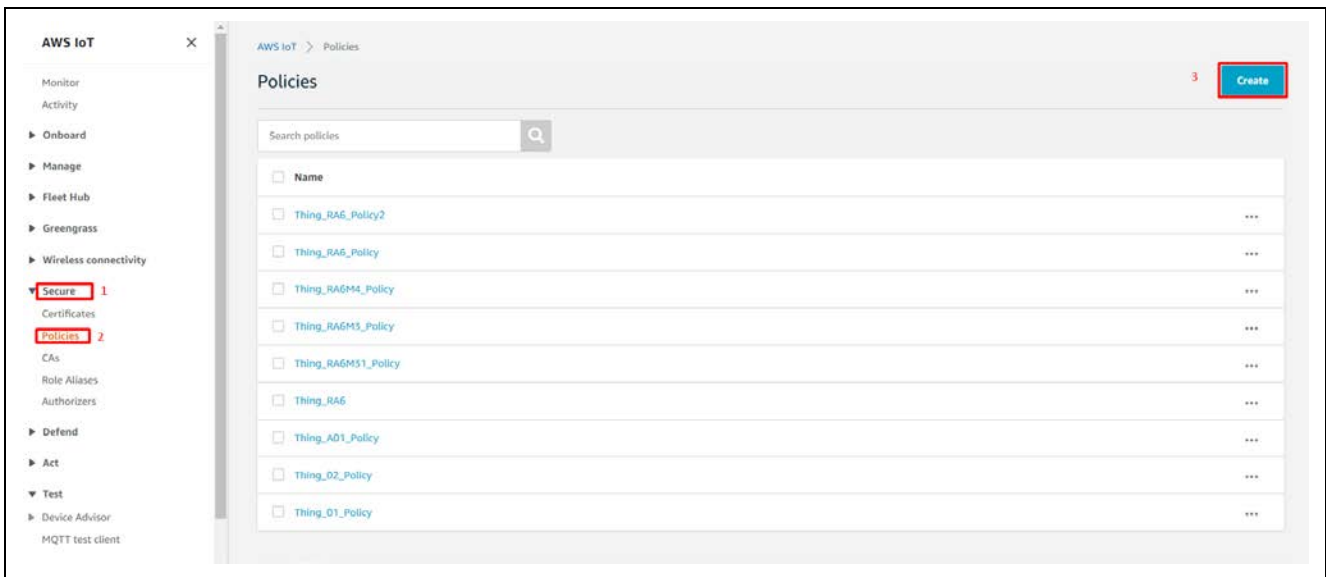
To create a policy, go back to the Thing Hub.

1. Click the **Secure** option shown in the following screen.
2. Click the **Policies** option; it opens a window to create a new policy.

3. Click the **Create** button in the top right corner of the policies window to create new policy.



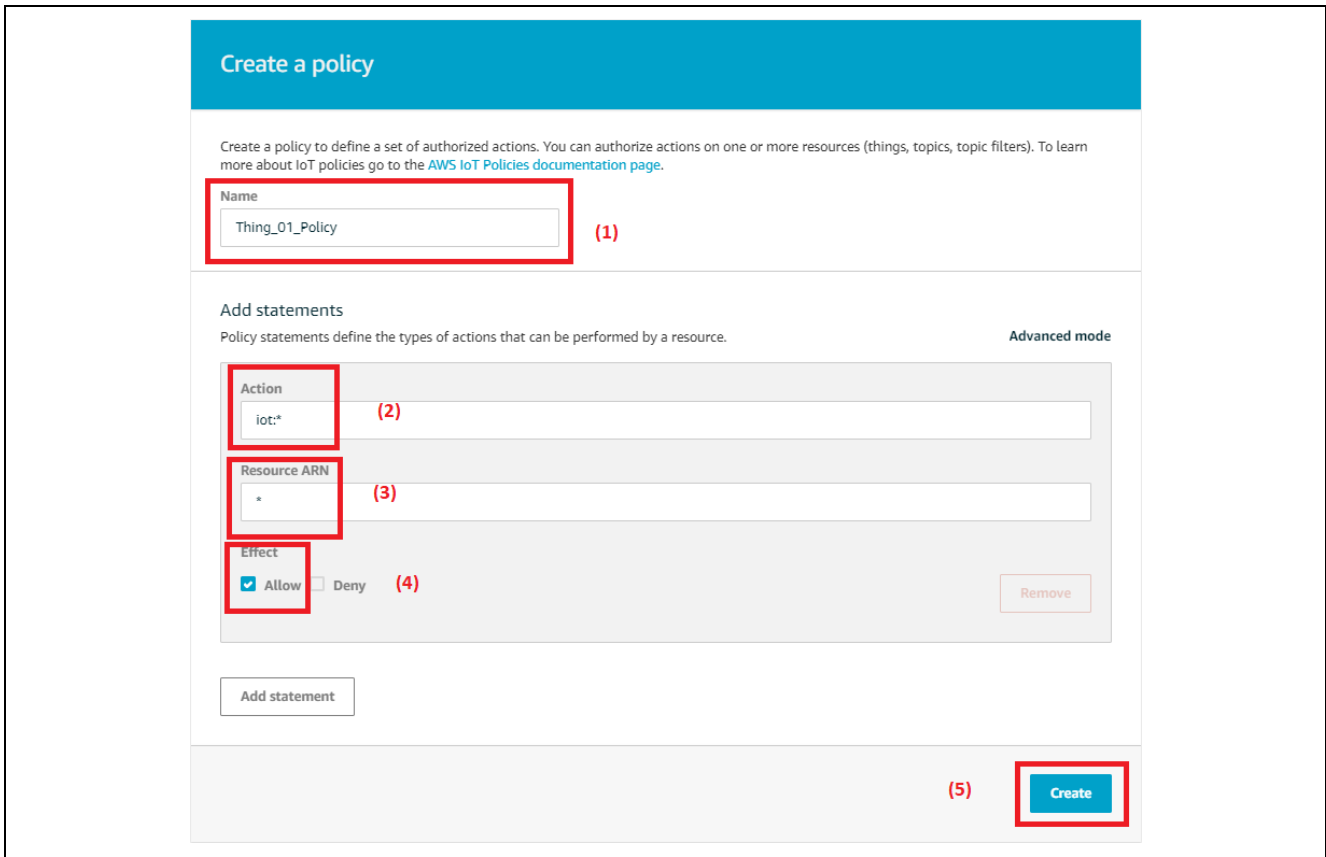
Note: If the Thing and Policy were created in the past, the Policy creation GUI snapshot may look different and is similar to the snapshot shown as follows.



4. Enter the **Name** for your policy in the Name box as shown in the following screen.
5. Under Action, type: **iot:\***
6. Under Resource ARN, type: \*

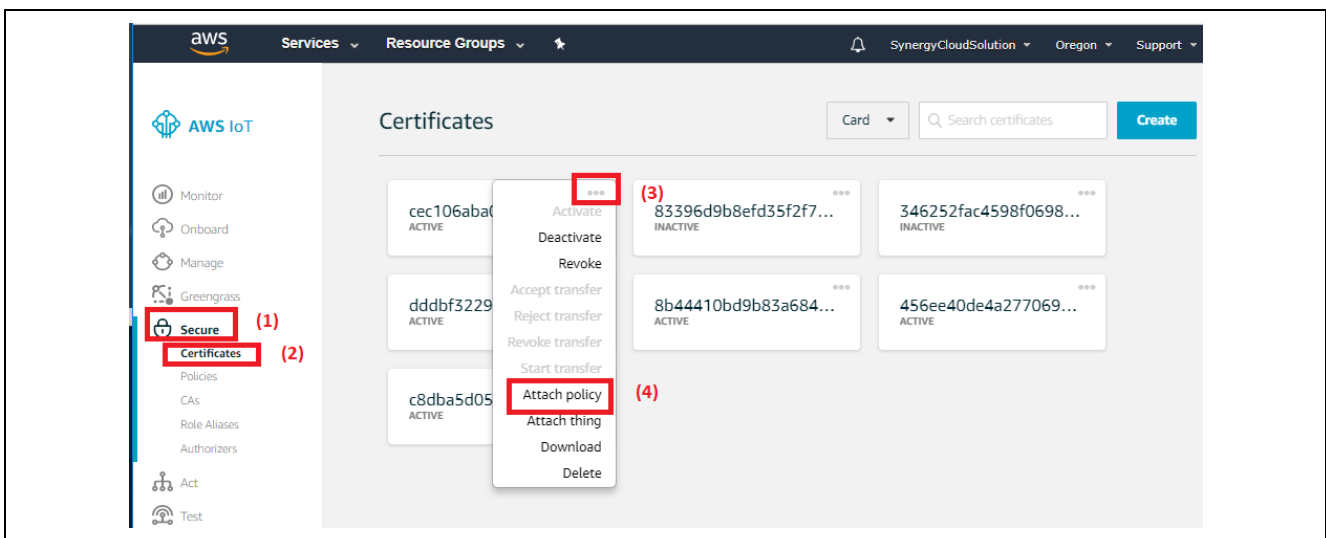
Note: The examples in this document are intended only for development environments. All devices in your fleet must have credentials with privileges that authorize only intended actions on specific resources. The specific permission policies can vary for your use case. Identify the permission policies that best meet your business and security requirements. For more information, refer to [Example policies](#) and [Security best practices](#).

7. Click **Allow**.
8. Click **Create**. Your policy has now been created.

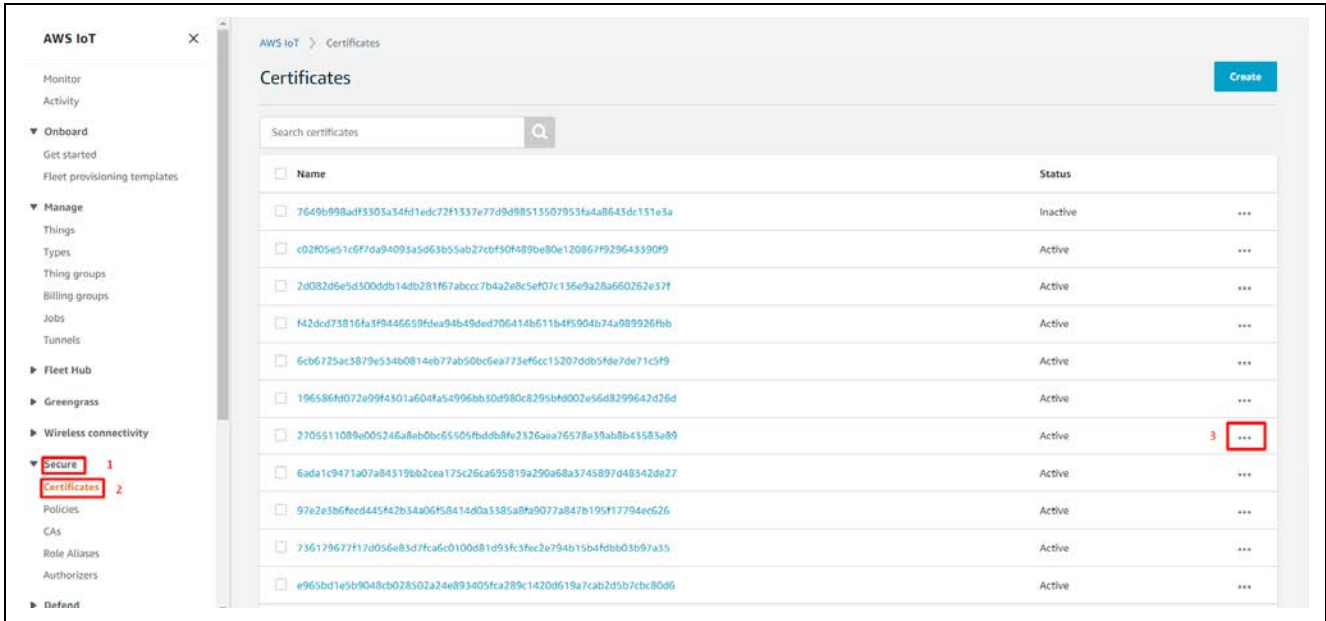


### 3.3.5 Connecting the Certificate to the Policy

1. Click the **Secure** option as shown in the following screen. Then, click the **Certificates** option. It will open a window listing the device certificates created in your AWS IoT Core service.
2. Choose the certificate you had created for your Thing. This can be done by clicking the “...” in the top right corner of your certificate.
3. Click the **Attach policy** option from the drop-down menu.

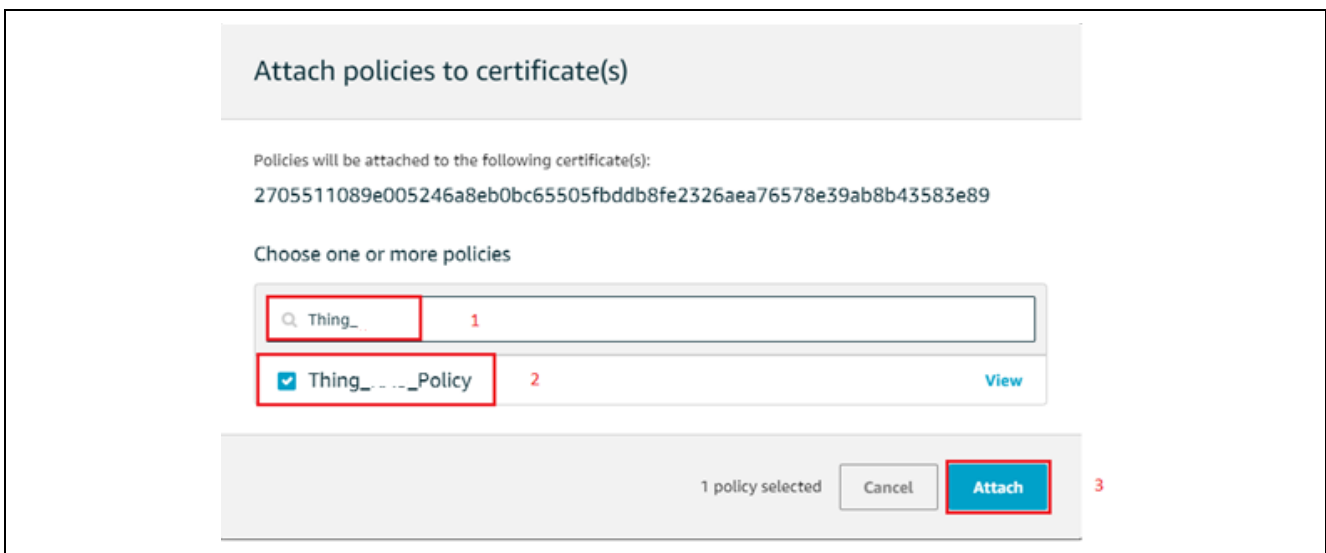


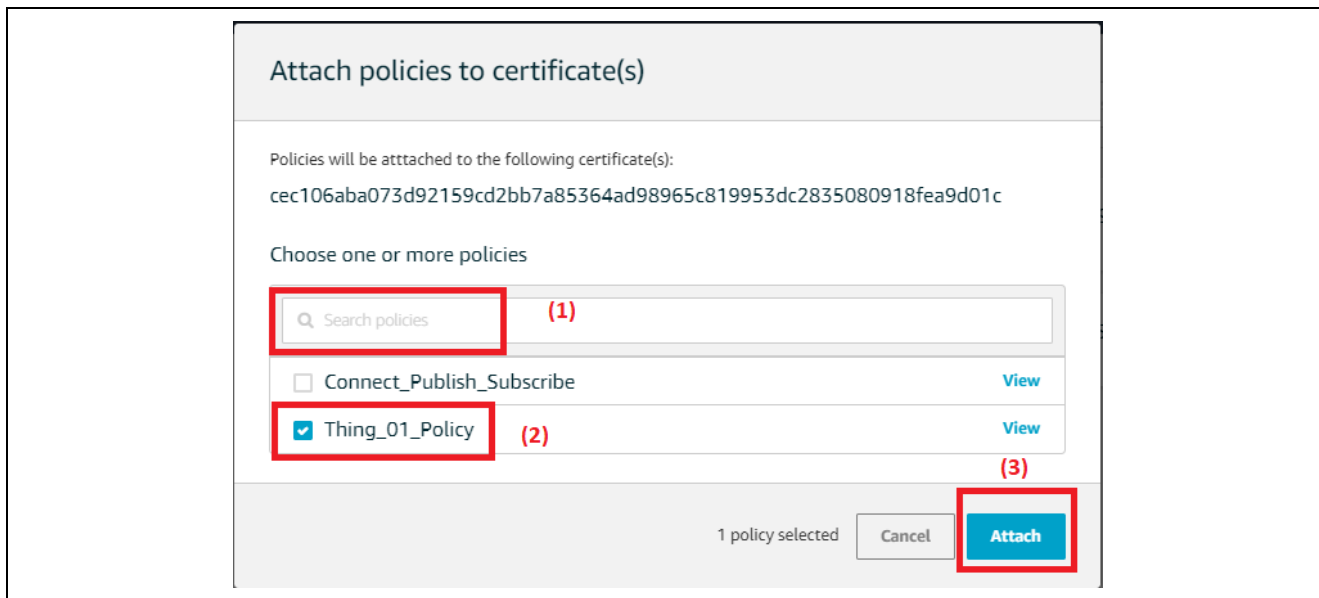
Note: If the Thing and Policy were created in the past on this account, you may notice the existing certificates. The Certificate attached to the policy GUI snapshot may look different, similar to the snapshot shown in Figure below



4. Search for the policy on the **Search policies** window.
5. Choose the policy from the list and click the **Attach** button, as shown in the following screen.
6. Your policy has now been attached to your device certificate.

Note: If the Thing and Policy were created on an existing AWS account which already has existing Things, the Certificate attached to the policy GUI snapshot may look different and is similar to the snapshot shown in the below Figure.





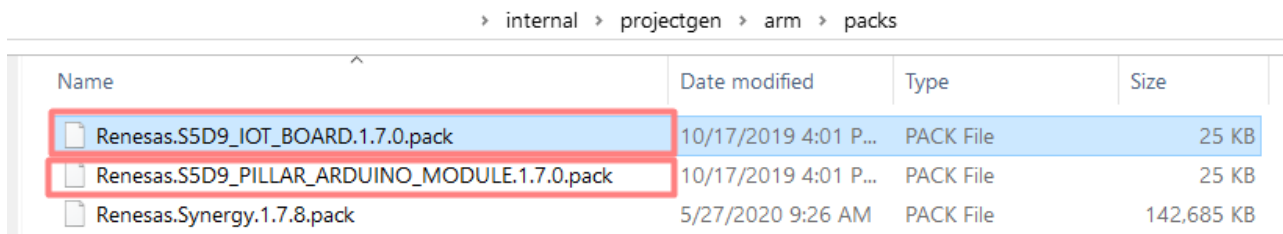
## 4. Running the MQTT/TLS Application

### 4.1 Importing, Building, and Loading the Project

See the *Renesas Synergy™ Project Import Guide* ([r11an0023eu0121-synergy-ssp-import-guide.pdf](http://r11an0023eu0121-synergy-ssp-import-guide.pdf)), included in this package, for instructions to import the project into e<sup>2</sup> studio, build and run the project.

### 4.2 Manually Adding the Board Support Package for the AE-CLOUD1/AE-CLOUD2 Kit

- From the project bundle, locate the BSP files, **Renesas.S5D9\_PILLAR\_ARDUINO\_MODULE.1.7.0.pack** for AE-CLOUD2 and **Renesas.S5D9\_IOT\_BOARD.1.7.0.pack** for AE-CLOUD1.
- For e<sup>2</sup> studio users: Copy the files shown in the following figure to the e<sup>2</sup> studio packs folder, `C:\Renesas\e2studio_v7.5.1\internal\projectgen\arm\packs`.



**Figure 11. Load BSP Pack for AE-CLOUD1/AE-CLOUD2 Kit**

- For IAR users: Copy the files to SCC \packs folder, `C:\Renesas\Synergy\ssc_v7.3.0_ssp_v1.7.8\internal\projectgen\arm\packs`

Note: If e<sup>2</sup> studio and IAR SSC are installed in any other location, the same information needs to be provided to copy the pack.

### 4.3 Powering up the Board

To connect power to the board, connect the SEGGER J-Link® debugger to the PC, connect the board to the PC USB port, and run the debug application, using the following instructions.

1. For AE-CLOUD2, connect the micro USB end of the supplied USB cable to the AE-CLOUD2 board’s J6 connector (DEBUG\_USB).  
Connect the other end of the USB cable to the USB port on your workstation.  
Note: The kit contains a SEGGER J-Link® On-board (OB). J-Link provides full debug and programming for the AE-CLOUD2 board.
2. For the AE-CLOUD1 board, connect the J-Link Lite, supplied with the kit, to the J2 connector on the AE-CLOUD1 and to the 10-pin header on the J-Link lite using the supplied 10-pin, flat-ribbon cable.
3. Attach the PMOD-based GT-202 Wi-Fi module in the PMOD connector.

4. For the AE-CLOUD2 kit, connect the BG96 Cellular shield on the AE-CLOUD2 Arduino Connector. Next, attach the Cellular antenna to the LTE antenna connector, and then the GPS antenna to the GNSS antenna connector on the BG96 shield and attach the PMOD-based GT-202 Wi-Fi module in the PMOD connector all the time in spite of running the demo using ethernet/cellular interface. *This redundant connection is necessary as a workaround to a known issue in SSP v1.6.3 and v1.7.0.*

5. Connect the second micro USB cable as follows:  
— AE-CLOUD2/ AE-Cloud1 board’s J9 connector  
Connect the other end of the USB cable to the USB port on your workstation. This connection is necessary for the serial console.

### 4.4 Connect to AWS IoT Cloud

The following instructions show how to run the Synergy Cloud connectivity application project and connect to the AWS IoT Cloud.

Note: At this stage, it is assumed you completed the instructions in section 3.3 to create an AWS IoT account, set up your device on the AWS IoT Core, and downloaded the device certificates and keys.

- Section 4.4 shows how the command line interface can be used to configure the boards; depending upon the desired interface for cloud connectivity.
- While running the application on these boards, connectivity is done using one interface at a time (Ethernet or Wi-Fi, or Cellular). Users are required to configure only the desired interface to run the application. For example, if you use Ethernet, then Wi-Fi or Cellular does not need to be configured and vice-versa.
- Note that the CLI snapshots shown in some cases may not be applicable to all the boards, such as Cellular not being applicable to AE-CLOUD1 board.

**Table 1. Kit Connectivity Options (only one interface supported at a time)**

Board	Ethernet	Wi-Fi	Cellular
AE-CLOUD1	Supported	Supported	Not Supported
AE-CLOUD2	Supported	Supported	Supported

1. Connect the USB Device port of the kit to the test PC. The port will be automatically detected as an USB Serial device in case of Windows 10 PC.  
In case of Windows 7/8 PC, refer to the following installation guide to load the Synergy USB CDC driver: [www.renesas.com/en-us/products/synergy/software/add-ons/usb-cdc-drivers.html](http://www.renesas.com/en-us/products/synergy/software/add-ons/usb-cdc-drivers.html)  
<https://en-support.renesas.com/knowledgeBase/16977397>.
2. Open the serial console application, such as Tera Term, to connect it to the AE-CLOUD1/AE-CLOUD2 kit. The default Tera Term settings are 8-N-1, and the baud rate is 9600.

3. Press **Enter**. The following command prompt and CLI information appears on the serial console.

```

*****
* Renesas Synergy AWS IoT Cloud Connectivity Application
*
* FW version 1.3.0 - Jan 28 2021, 15:32:58
*
* Synergy Software Package Version: 1.7.8
*****
>
    
```

Figure 12. Command Prompt

4. Press the **?** key on your keyboard to display the available CLI command options shown in the following figure.

```

>?
Help Menu
  cwiz : Network/Cloud Configuration Menu
        Usage: cwiz

  demo : Start/Stop Synergy Cloud Connectivity Demo
        Usage: demo <start>/<stop>
    
```

Figure 13. Help Menu

#### 4.4.1 Configuration Wizard Menu

Enter command `cwiz`. Press the enter key in the serial console to enter the configuration menu. The `cwiz` command is used to configure the Network interfaces, the AWS IoT Core Service, and to dump the previous configuration stored in the internal flash.

```

COM34 - Tera Term VT
File Edit Setup Control Window Help
?
Help Menu
  cwiz : Network/Cloud Configuration Menu
        Usage: cwiz

  demo : Start/Stop Synergy Cloud Connectivity Demo
        Usage: demo <start>/<stop>

>cwiz
##### Main Menu #####
1. Network Interface Selection
2. AWS IoT Core Configuration
3. Dump previous configuration from flash
4. Exit
Please Enter Your Choice:>
    
```

Figure 14. Configuration Menu

##### 4.4.1.1 Network Interface Selection

From the configuration menu, press **1** key to configure the Network Interface. It lists the available network interface options in this application project. Currently this application supports Ethernet, Wi-Fi, Cellular (in case of AE-CLOUD2 kit) communication interfaces.

Note: The user can select only one network interface at a time. For instance, when Ethernet is selected, Wi-Fi and Cellular are not available and vice-versa. To change the network interface, the demonstration should be stopped, and the new interface should be selected through `cwiz`.

If the user uses the same network interface between power cycles, then the network interface selection can be skipped since the interface and credentials are stored in the flash.



For example, if the user selects **Ethernet Network Interface Configuration** as the interface for cloud connectivity before and after a power cycle, the user can go directly to section 4.4.1.2, AWS IOT Core Configuration. The same applies for Wi-Fi or Cellular as well.

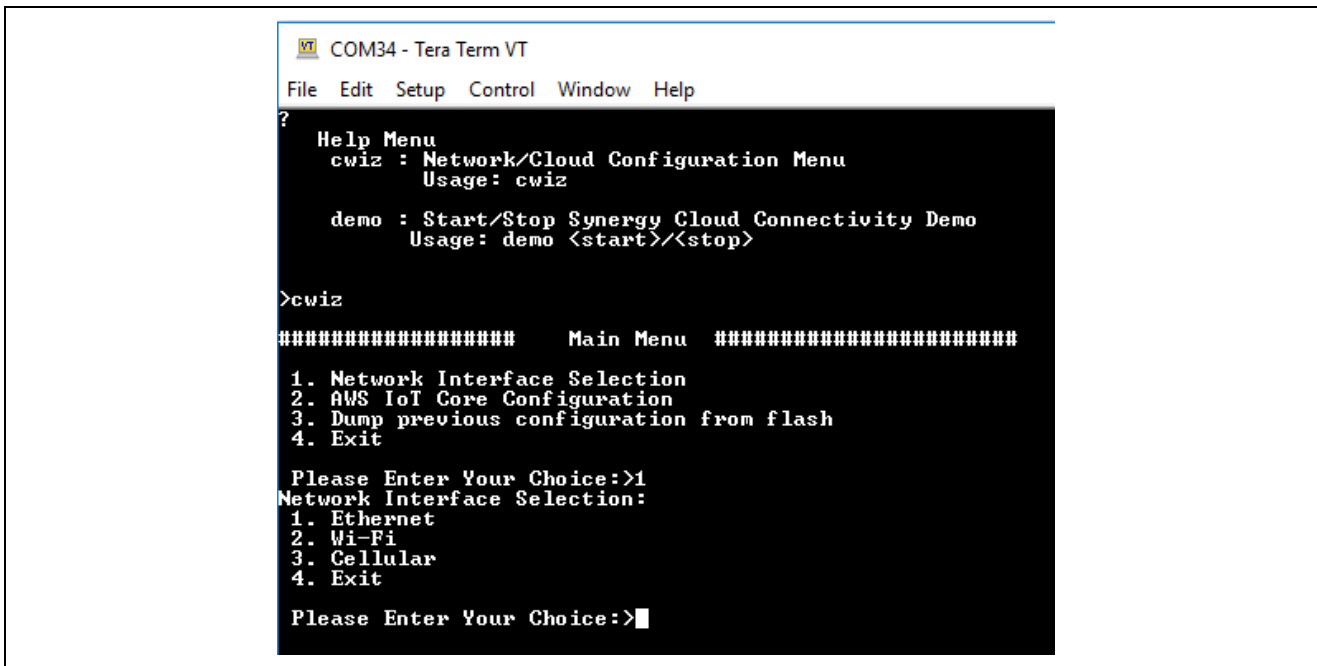


Figure 15. Network Interface Selection Menu

(1) Ethernet Network Interface Configuration

From the **Network Interface Selection** menu, press 1 to select the Ethernet Network Configuration.

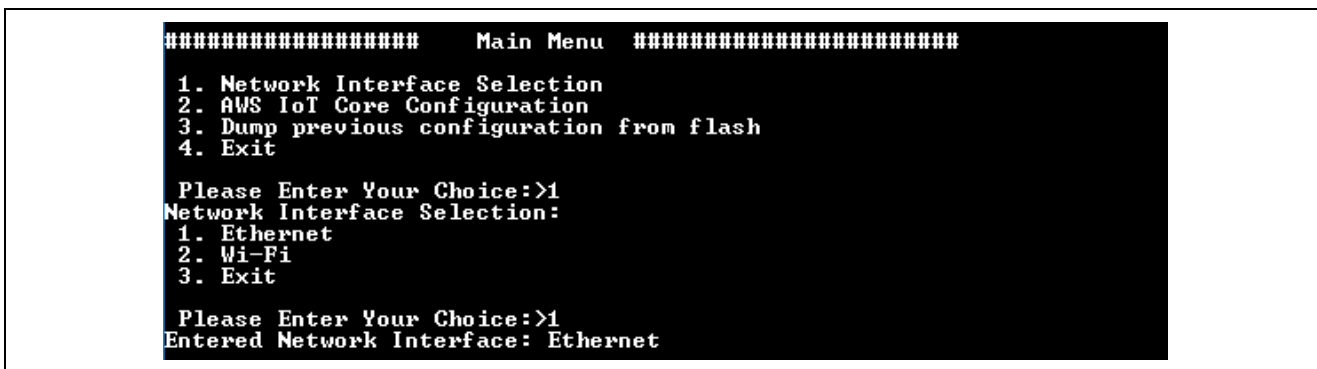


Figure 15. Ethernet Network Interface Selection

You see the submenu where you choose the IP Address Configuration mode from the available options (DHCP/Static). Choose the **IP Address Configuration** mode. The selected Ethernet configuration setting is stored in internal flash; it is used at a later stage, when communication is initialized.

**Note:** The Ethernet Static IP configuration does not work with the default project. User needs to add the NetX Duo source code to the project. This issue will be fixed in future releases.

```

Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>1
Entered Network Interface: Ethernet

Enter IP Address Configuration Mode
 1. Static
 2. DHCP
Please Enter Your Choice
>2
Entered IP Configuration Mode: DHCP
Network Configuration stored in flash
    
```

Figure 16. Ethernet Network Interface Configuration Menu – DHCP Configuration

```

Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>1
Entered Network Interface: Ethernet

Enter IP Address Configuration Mode
 1. Static
 2. DHCP
Please Enter Your Choice
>1
Entered IP Configuration Mode: Static

Enter the IP Address:
>143.103.16.80
Enter Network Mask:
>255.255.255.128
Enter Gateway:
>143.103.16.2
Enter DNS:
>143.103.10.62
Network Configuration stored in flash
    
```

Figure 17. Ethernet Network Interface Configuration Menu – Static IP Configuration

(2) Wi-Fi Network Interface Configuration

From the **Network Interface Selection** menu, press **2** to select the Wi-Fi Network Configuration.

```

COM34 - Tera Term VT
File Edit Setup Control Window Help
?
Help Menu
cwiz : Network/Cloud Configuration Menu
Usage: cwiz

demo : Start/Stop Synergy Cloud Connectivity Demo
Usage: demo <start>/<stop>

>cwiz
##### Main Menu #####
 1. Network Interface Selection
 2. AWS IoT Core Configuration
 3. Dump previous configuration from flash
 4. Exit

Please Enter Your Choice:>1
Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>
    
```

Figure 18. Wi-Fi Network Interface Configuration Menu

You are given the option to enter Wi-Fi Configuration settings, such as **SSID**, **Pass key**, **Security type**, and **IP Address Configuration mode**.

The selected Wi-Fi configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.

```
Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>2
Entered Network Interface: Wi-Fi

Wi-Fi Configuration
Enter the SSID associated with the Network
>rea-guestwifi
Enter the passphrase
>xxxxxxxxxxxx
Enter Security Type
 1. WEP
 2. WPA
 3. WPA2
 4. None
Please Enter Your Choice
>3
Entered Security Type: WPA2

Enter IP Address Configuration Mode
 1. Static
 2. DHCP
Please Enter Your Choice
>2
Entered IP Configuration Mode: DHCP
Network Configuration stored in flash
```

Figure 19. Wi-Fi Configuration – DHCP Configuration Mode

```
Network Interface Selection:
 1. Ethernet
 2. Wi-Fi
 3. Cellular
 4. Exit

Please Enter Your Choice:>2
Entered Network Interface: Wi-Fi

Wi-Fi Configuration
Enter the SSID associated with the Network
>rea-guestwifi
Enter the passphrase
>xxxxxxxxxxxx
Enter Security Type
 1. WEP
 2. WPA
 3. WPA2
 4. None
Please Enter Your Choice
>3
Entered Security Type: WPA2

Enter IP Address Configuration Mode
 1. Static
 2. DHCP
Please Enter Your Choice
>1
Entered IP Configuration Mode: Static

Enter the IP Address:
>192.168.1.62
Enter Network Mask:
>255.255.255.0
Enter Gateway:
>192.168.1.254
Enter DNS:
> 4.2.2.4
Network Configuration stored in flash
```

Figure 20. Wi-Fi Configuration – Static IP Configuration

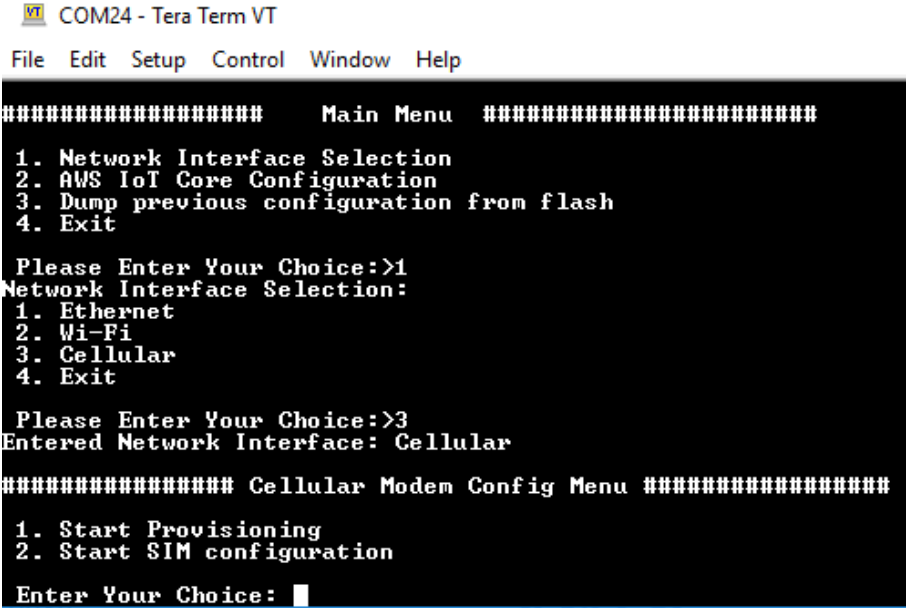
**(3) Cellular Network Interface Configuration (Applicable for AE-CLOUD2 only)**

From the **Network Interface Selection** menu, press **3** to select the Cellular Network Configuration.

You will be given the two choices:

- Option 1: Enter **1** in case of SIM provisioning. In this case, it is assumed that you already pre-configured the SIM card. You can only enter the APN, context ID and PDP type of the SIM.
- Option 2: Enter **2** in case of SIM configuration. This option is ideal if you need to configure the SIM card using the AT shell interface. For example, setting the Scan Mode, IoT OpMode and so forth, on the SIM.

Note: After cellular connects to the cloud using Option 1, user cannot return to the AT shell configuration window using Option **2** anymore.



```

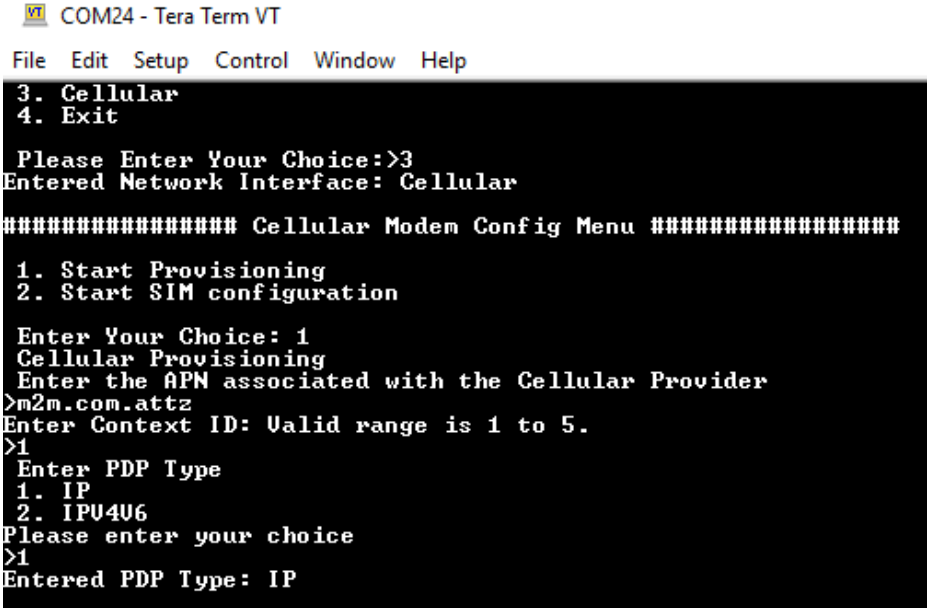
COM24 - Tera Term VT
File Edit Setup Control Window Help
##### Main Menu #####
1. Network Interface Selection
2. AWS IoT Core Configuration
3. Dump previous configuration from flash
4. Exit
Please Enter Your Choice:>1
Network Interface Selection:
1. Ethernet
2. Wi-Fi
3. Cellular
4. Exit
Please Enter Your Choice:>3
Entered Network Interface: Cellular
##### Cellular Modem Config Menu #####
1. Start Provisioning
2. Start SIM configuration
Enter Your Choice: █

```

Figure 21. Cellular Configuration

**(a) Start Provisioning Option**

In the **cellular modem configuration menu**, choose option **1** to enter the **Start provisioning** sub-menu.



```

COM24 - Tera Term VT
File Edit Setup Control Window Help
3. Cellular
4. Exit
Please Enter Your Choice:>3
Entered Network Interface: Cellular
##### Cellular Modem Config Menu #####
1. Start Provisioning
2. Start SIM configuration
Enter Your Choice: 1
Cellular Provisioning
Enter the APN associated with the Cellular Provider
>m2m.com.attz
Enter Context ID: Valid range is 1 to 5.
>1
Enter PDP Type
1. IP
2. IPU4U6
Please enter your choice
>1
Entered PDP Type: IP

```

Figure 22. Cellular Modem Provisioning Menu

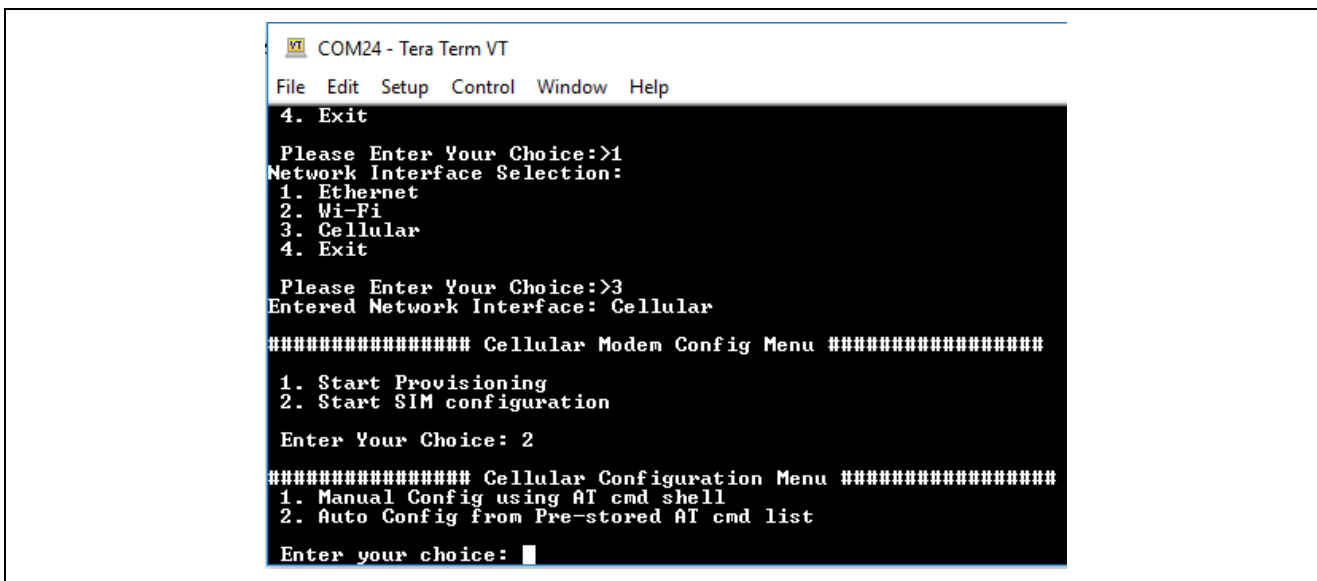
You are given the option to enter Cellular Configuration settings, such as APN, Context ID, PDP type.

The selected Cellular configuration setting is stored in the internal flash to be used at a later stage, when the communication is initialized.

**(b) Start SIM Configuration Option**

**Note:** Use this menu item to exercise configuration of a new SIM card and identify the proper settings. This menu item cannot connect the device to the cloud even if “demo start” is issued. After finishing SIM card configuration using this menu item, user needs to go back to the main menu and choose option **1 Start Provisioning** to connect the device to cloud.

In the **Cellular modem configuration** menu, choose option **2** to enter the Start SIM Configuration sub-menu.

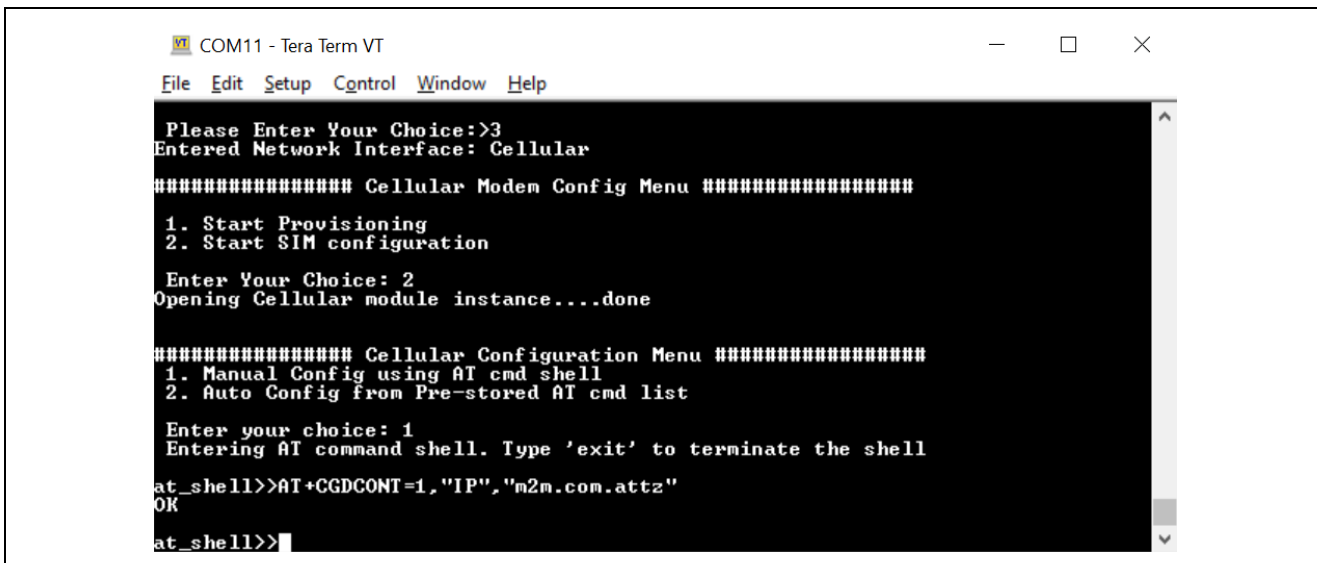


**Figure 23. Cellular Configuration Menu**

You can either choose option 1 to enter Manual Configuration mode using AT command shell or choose option 2 to enter Auto Configuration from a pre-stored AT command list. You generate this list at the end of the previous Manual configuration.

**Manual Configuration using AT Command Shell**

In case you select option 1 in the Cellular Configuration Menu, enter the following AT command shell. You can experiment with various AT commands to configure the SIM cards.



**Figure 24. AT Command Shell**

See the following knowledge base article provided by Renesas as a baseline to provision the SIM card using the BG96 Cellular modem:

<https://en.na4.teamsupport.com/knowledgeBase/18027787>

To exit the AT command shell, enter the command **exit** or **EXIT**. You will be asked whether to save the AT command, as shown in the following screen.

```

COM24 - Tera Term VT
File Edit Setup Control Window Help
1. Ethernet
2. Wi-Fi
3. Cellular
4. Exit

Please Enter Your Choice:>3
Entered Network Interface: Cellular

##### Cellular Modem Config Menu #####

1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: 1
Entering AT command shell. Type 'exit' to terminate the shell
at_shell>>exit
Do you wish to store the AT commands for your carrier? [Y/N]:

```

If you chose to save the AT commands, which can be later used to auto configure the new SIM cards, enter **Y**. When you do, you will be asked to enter the AT command details, as shown in the following screen.

**Note:** Only the commands you entered after you choose **Y** for the above query will be saved. See the following example.

```

at_shell>>exit
Do you wish to store the AT commands for your carrier? [Y/N]: Y
***** Start Inserting AT Commands. Type exit to terminate!!! *****

AT Command: at+cgdcont=1,"IP","m2m.com.attz"
Response <case sensitive>: OK
Response Wait time in MilliSeconds: 1000
Retry Count: 5
Retry Delay in milli-seconds : 100

#####
AT Command: at+cgdcont=1,"IP","m2m.com.attz"
Response string: OK
Response Wait time: 1000
Retry Count: 5
Retry Delay: 100

#####
Do you Want to save this AT Command ? [y/n]: Y
AT Command: exit

```

**Auto Configuration from pre-stored AT command list**

If you chose option **2** in the Cellular Configuration Menu, enter the Auto configuration from the following pre-stored AT command list menu.

```
##### Cellular Modem Config Menu #####
1. Start Provisioning
2. Start SIM configuration

Enter Your Choice: 2
Opening Cellular module instance...done

##### Cellular Configuration Menu #####
1. Manual Config using AT cmd shell
2. Auto Config from Pre-stored AT cmd list

Enter your choice: 2

#####

Command: at+cgdcont=1,"IP","m2m.com.attz"

Resp:
OK
```

**Figure 25. Auto configuration from pre-stored AT command list**

The pre-stored AT commands will be sent to the cellular modem and their responses will be displayed in the console window.

Note: In case of repeated failures to register to the network, increase the AT command retry count. Set the appropriate network scan sequence in the Synergy Configurator of the project, then generate and rebuild the project.

**4.4.1.2 AWS IoT Core Configuration**

From the **Main Menu**, press **2** and press **Enter** to configure the AWS IoT Core service as shown in the following screen.

```
*****
>?
Help Menu
cwiz : Network/Cloud Configuration Menu
Usage: cwiz

demo : Start/Stop Synergy Cloud Connectivity Demo
Usage: demo <start>/<stop>

>cwiz
##### Main Menu #####
1. Network Interface Selection
2. AWS IoT Core Configuration
3. Dump previous configuration from flash
4. Exit

Please Enter Your Choice:>2
1. AWS IoT Core Setting Menu
2. Device Certificate/Keys Setting Menu
3. Exit

Please Enter Your Choice:>█
```

**Figure 26. AWS IoT Core Configuration Menu**

**(1) AWS IoT Core Setting Menu**

From the AWS IoT Core configuration menu, press **1** and hit **Enter** to configure the AWS IoT Core settings, as shown in the following screen.

```

*****
>?
Help Menu
cwiz : Network/Cloud Configuration Menu
      Usage: cwiz

demo : Start/Stop Synergy Cloud Connectivity Demo
      Usage: demo <start>/<stop>

>cwiz
#####      Main Menu      #####
1. Network Interface Selection
2. AWS IoT Core Configuration
3. Dump previous configuration from flash
4. Exit

Please Enter Your Choice:>2
1. AWS IoT Core Setting Menu
2. Device Certificate/Keys Setting Menu
3. Exit

Please Enter Your Choice:>1

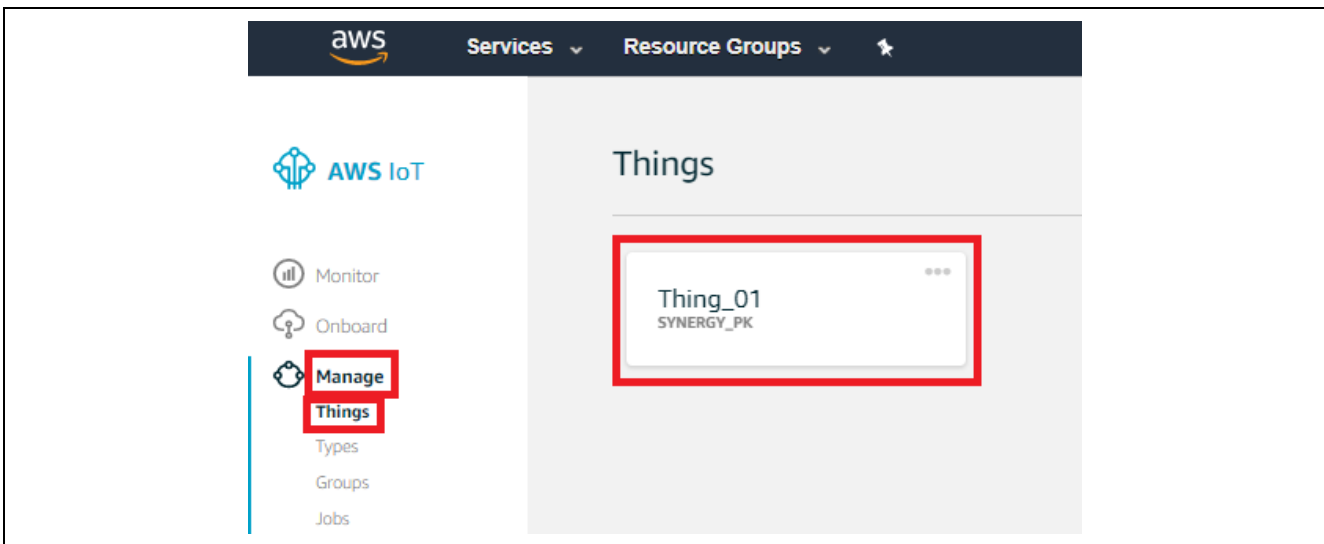
AWS Cloud Settings Menu
1. Enter AWS Endpoint information:
2. Enter AWS Thing Name:
3. Exit

Please Enter Your Choice:>
    
```

In the AWS IoT core configuration menu, you have the option of entering AWS Endpoint information and the AWS Thing name.

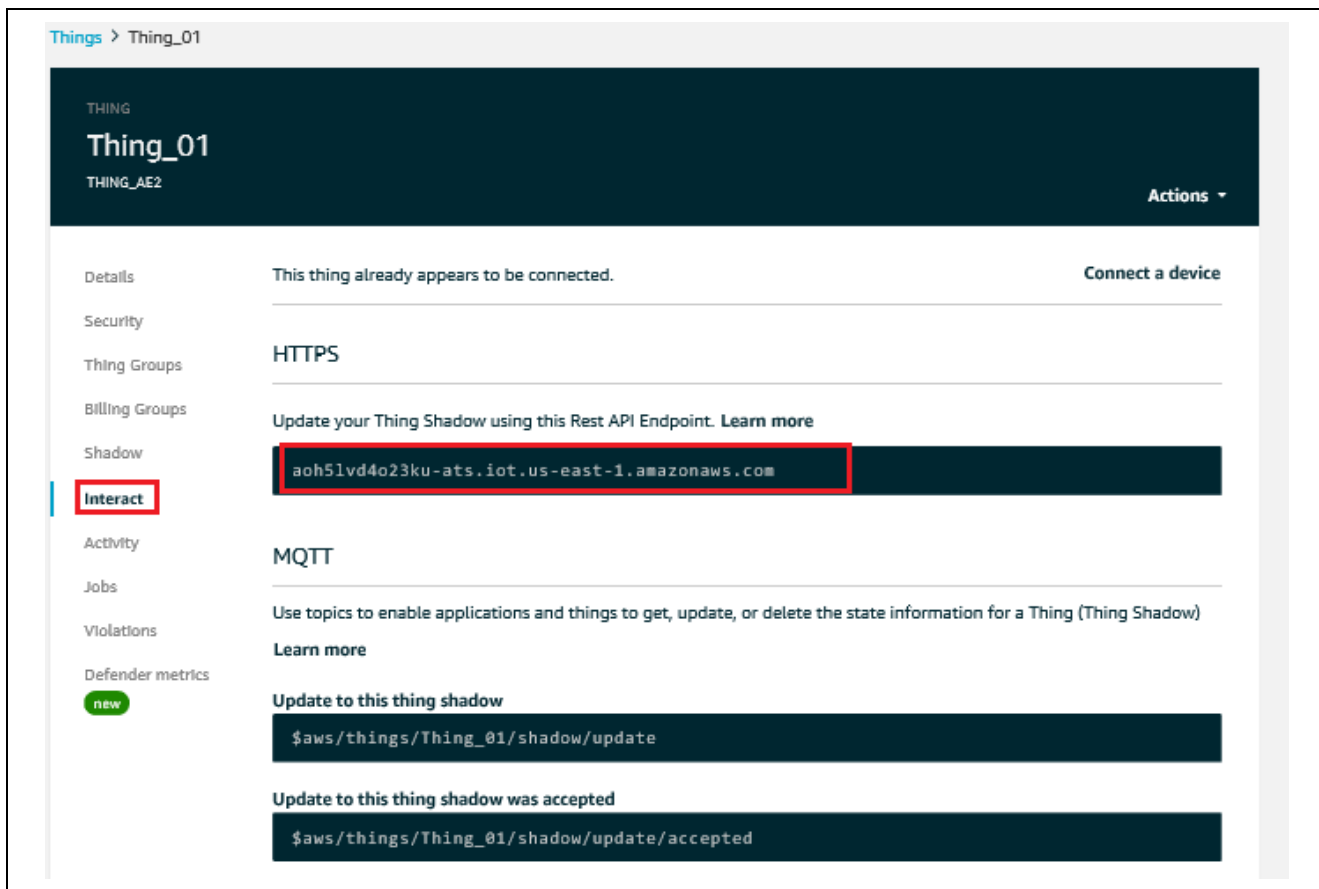
To locate the AWS IoT endpoint information for your MQTT thing, use the following steps.

1. Open the thing you had created for this application. The thing created can be found under **Manage** tab, as the following screen shows. Click the Thing name to open it.





- Open the Thing, then go to the **Interact** tab shown below. The AWS IoT Endpoint address can be found under the Rest API Endpoint block.



The selected configuration setting is stored in internal flash; it is used at a later stage during the AWS IoT Core connection.

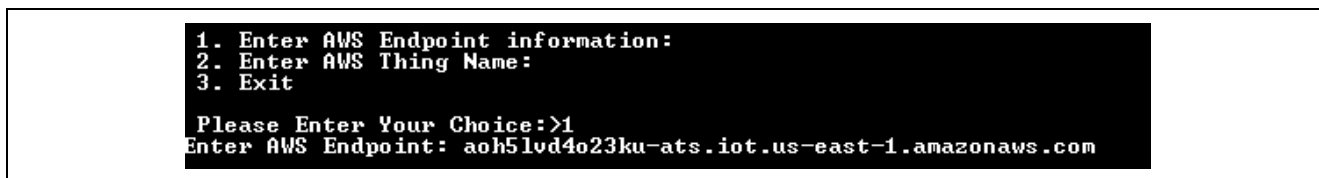
Note: If any user created the MQTT Thing and policies in the AWS console before October 2018 and is still using them to test this AP, they need to manually change the MQTT Endpoint as the shown below and use the `rootCA.pem` file given as part of this package.

**New MQTT Endpoint name:** `a12rqr6dvovqfi-ats.iot.us-west-2.amazonaws.com`

**Modified MQTT Endpoint name:** `a12rqr6dvovqfi.iot.us-west-2.amazonaws.com`

Note: If the MQTT Thing and policies are created new, and in order to use it for testing the AP, no need to change the endpoint info as well user doesn't need to use the attached `rootCA.pem`. Instead use the new Endpoint as is and root CA can be used as referenced in the section 3.3.3 (4) (Amazon root CA1).

- At the prompt, enter **1** to enter AWS Endpoint Information.
- Paste the Endpoint information in the CLI as shown.



- At the prompt, enter **2** to enter AWS Thing Name.
- Enter the name of the thing that is created in section 3.

```
AWS Cloud Settings Menu
1. Enter AWS Endpoint information:
2. Enter AWS Thing Name:
3. Exit

Please Enter Your Choice:>2
Enter AWS Thing Name: Thing_01
```

## (2) Certificate/Keys Setting Menu

Exit out of the AWS Cloud setting menu by entering **3** at the prompt. Exiting brings the CLI to the following AWS Core configuration menu.

```
AWS Cloud Settings Menu
1. Enter AWS Endpoint information:
2. Enter AWS Thing Name:
3. Exit

Please Enter Your Choice:>3
Device Certificate information stored in flash

1. AWS IoT Core Setting Menu
2. Device Certificate/Keys Setting Menu
3. Exit

Please Enter Your Choice:>
```

From the **AWS IoT Core configuration menu**, press **2** and **Enter** key to configure the Device Certificate/Keys settings.

Note: This step should be carried out sequentially. All certificates should be entered sequentially. Any change to one of the certificates requires all the certificates to be re-entered in the same order. Be sure to copy the labels indicating the beginning and end of the certificate.

In the Device Certificate/Keys settings menu, you have the option of entering the root CA, device certificates, and device private key in the `.pem` format.

Enter 1 at the prompt and paste the root CA certificate downloaded in section 3.3.3.

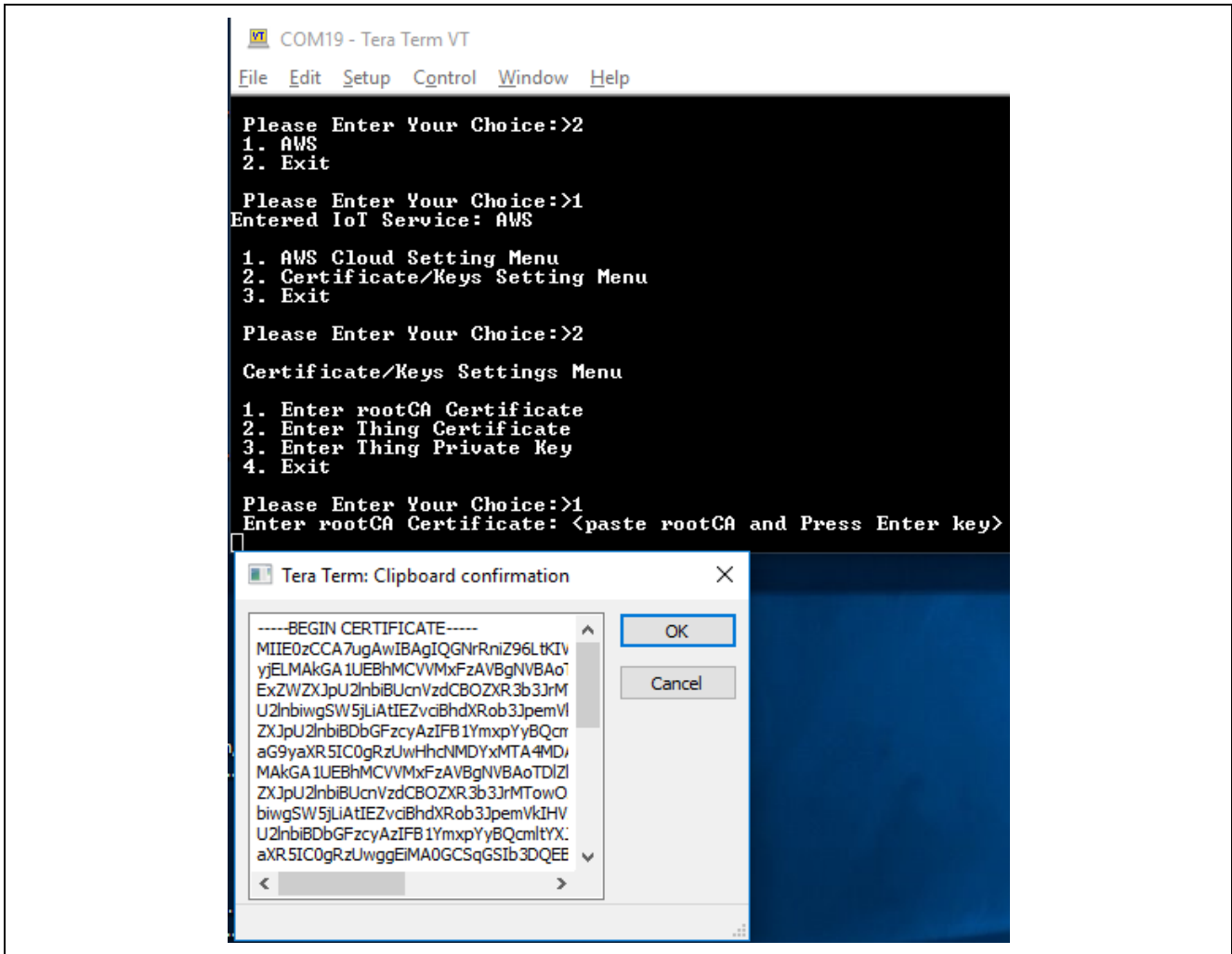


Figure 27. Certificates/Keys Setting Menu

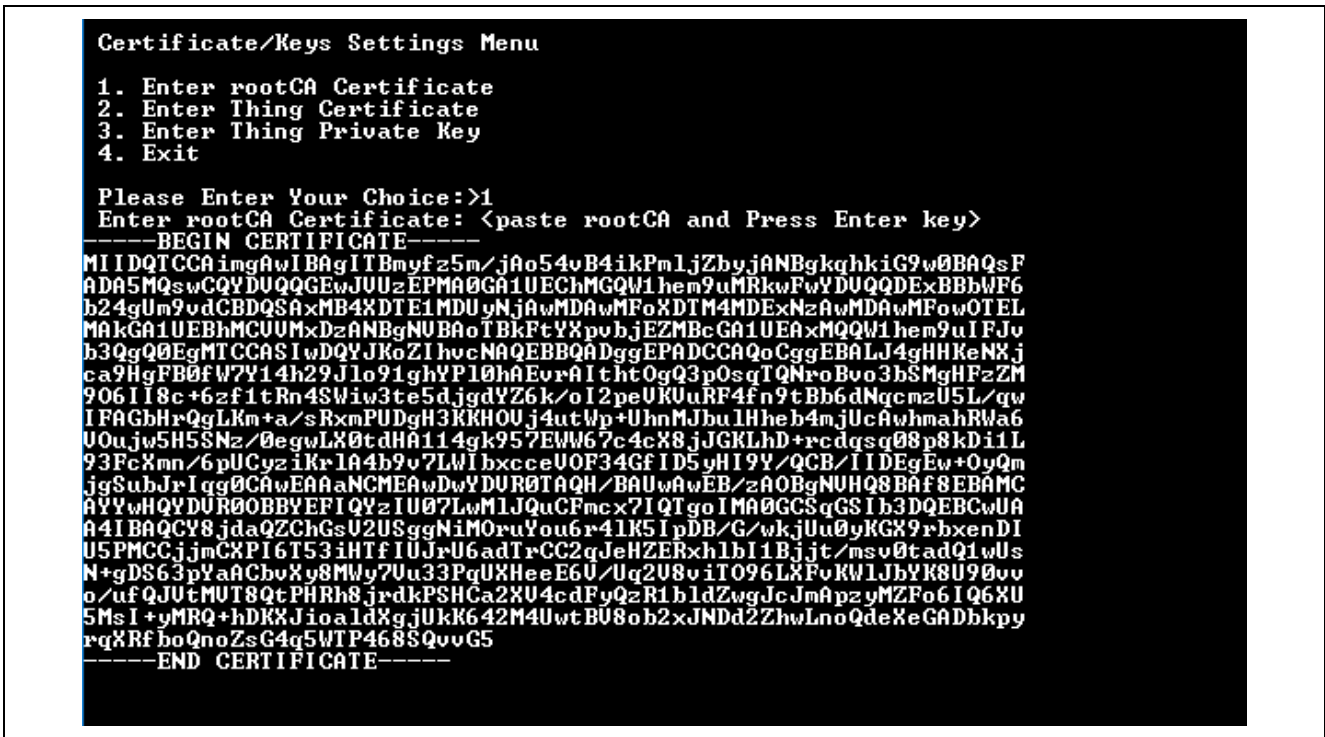
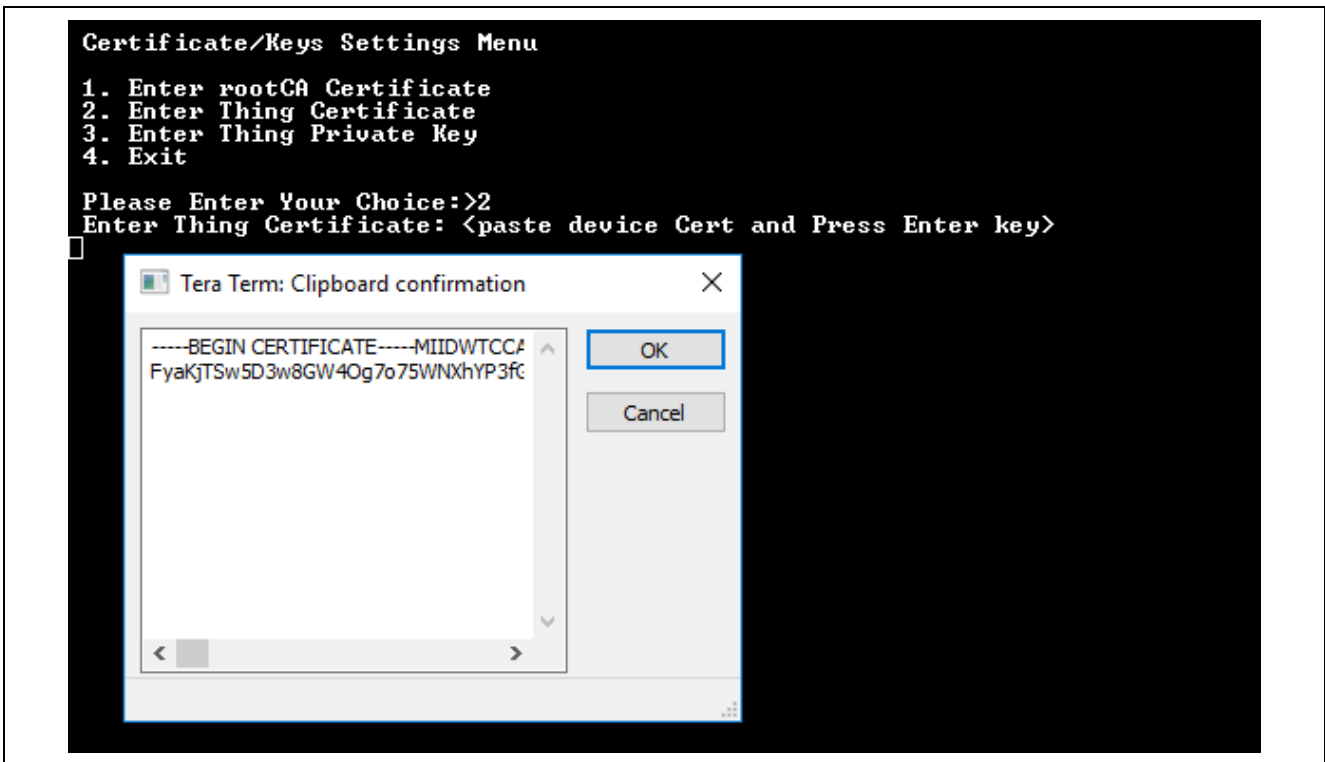


Figure 28. Entering Root CA Certificate

Press **Enter** to return to the Certificate Settings Menu.

Enter **2** to enter the Thing Certificate.

Open the xxxx-certificate.pem.crt file, downloaded in section 3.3.3, in a text editor. Copy and paste the certificate including the labels to the CLI. Press **OK**.



Again, press **Enter** to return to the Certificate Settings Menu and enter **3** to enter the Thing Private Key.

Open the xxxxxxxx-private.pem.key, downloaded in section 3.3.3, in a text editor. Copy and paste the certificate including the labels to the CLI.



The selected configuration setting is stored in the internal flash; it is used at a later stage during the AWS IoT Core connection.

Exit to the Main Menu by choosing the Exit option from the menu.

### 4.4.1.3 Dump Previous Configuration

From the Main menu, choose option 3 to display the pre-selected network, the AWS IoT core Service Configuration options you selected from the internal flash, as shown in the following screen.

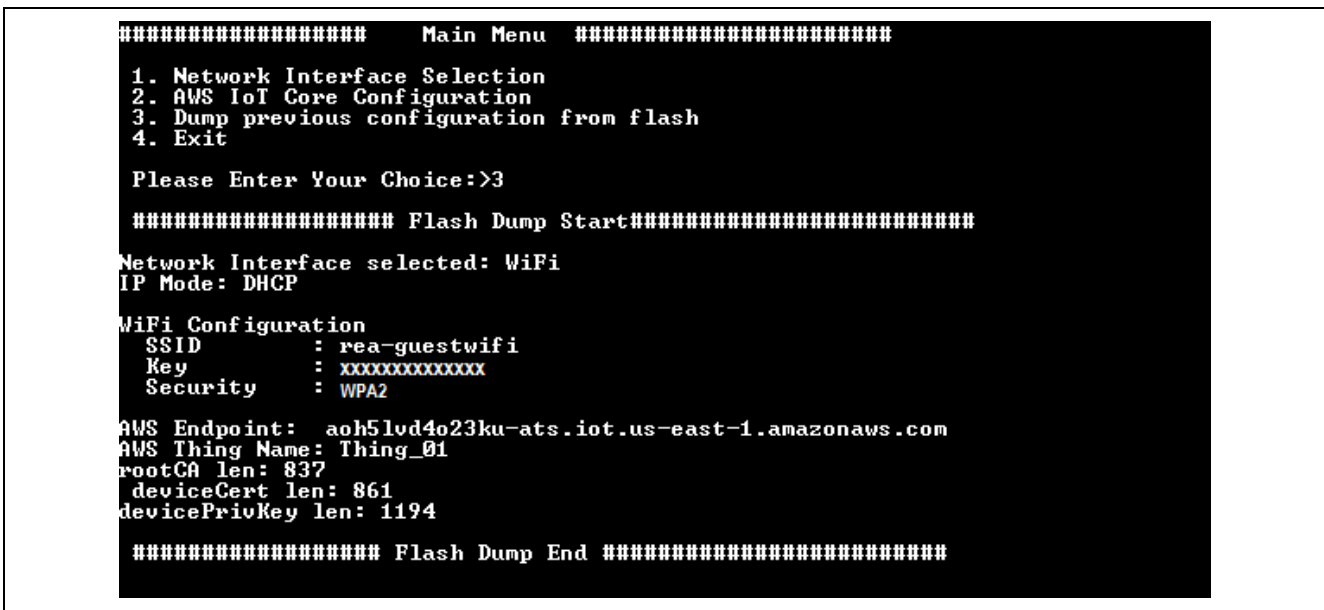


Figure 29. Dump Configuration Menu

**Note:** Ensure that all cloud information is present in the dump from flash. Any empty space in the cloud information indicates that the cloud information is stored incorrectly.

Note: The first-time project is flashed onto the board; since there is no information stored in flash, dumping the data using `cwiz` command may result in garbage being displayed on the CLI. Power cycle the board and configure before proceeding to dump the data.

Exit out of the **Main** menu by entering **4** at the prompt.

#### 4.4.2 Demo Start/Stop Command

From the CLI console, enter `demo start` command to start the Synergy Cloud Connectivity Application Demonstration.

```
*****
>?
Help Menu
  cwiz : Network/Cloud Configuration Menu
        Usage: cwiz

  demo : Start/Stop Synergy Cloud Connectivity Demo
        Usage: demo <start>/<stop>

>|
```

Figure 30. Help Menu

The application framework reads your pre-configured selection options for the network interface, the IoT Service from the internal flash, and checks for its validity. If the content is valid, it then initializes the network interface and establishes a MQTT connection with the AWS IoT Core.

This application wakes up periodically (every 5 seconds) and checks for your input event flag state. The flag state is set once you have entered the `demo start/stop` command on the CLI. This application does the following functions periodically until you enter the `demo stop` command.

1. Initialize communication interface (Ethernet/Wi-Fi/Cellular).
2. Initialize IoT Cloud interface.
3. Read sensor data and publish them periodically on MQTT topics.
4. Updates your LED state based on the type of MQTT message received.

If the `demo stop` command is issued, it de-initializes the IoT Cloud interface, which in turn stops MQTT messages from publishing and clears any pending MQTT messages from its internal queue.

Note: Once the demo starts running (`demo start` command issued), the `cwiz` command should not be used until the demo is stopped using the `demo stop` command.

## 4.5 Verifying the Demo

The following instructions verify the functionality of this Synergy Cloud Connectivity Application Project.

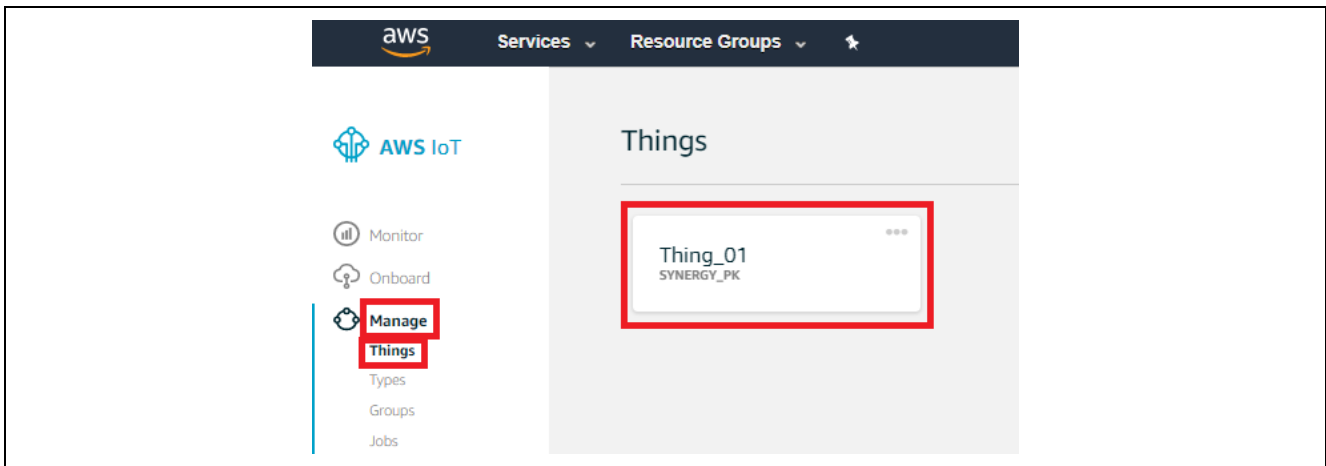
### 4.5.1 Starting the Synergy Cloud Connectivity Demonstration

Use the `demo start` command to run this application demonstration from the serial console.

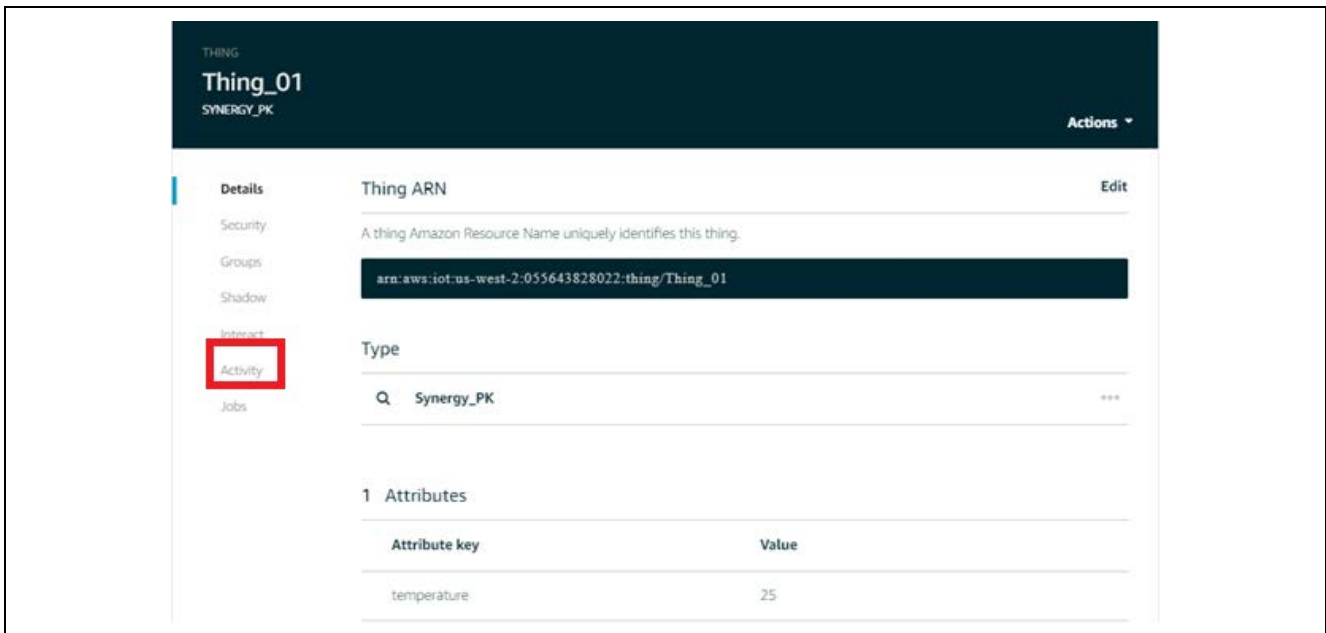
Once you run `demo start`, it begins to configure the network interface, establishes a connection with AWS IoT Core, and starts publishing sensor data periodically (every 5 seconds).

### 4.5.2 Opening the MQTT Client on AWS IoT Core

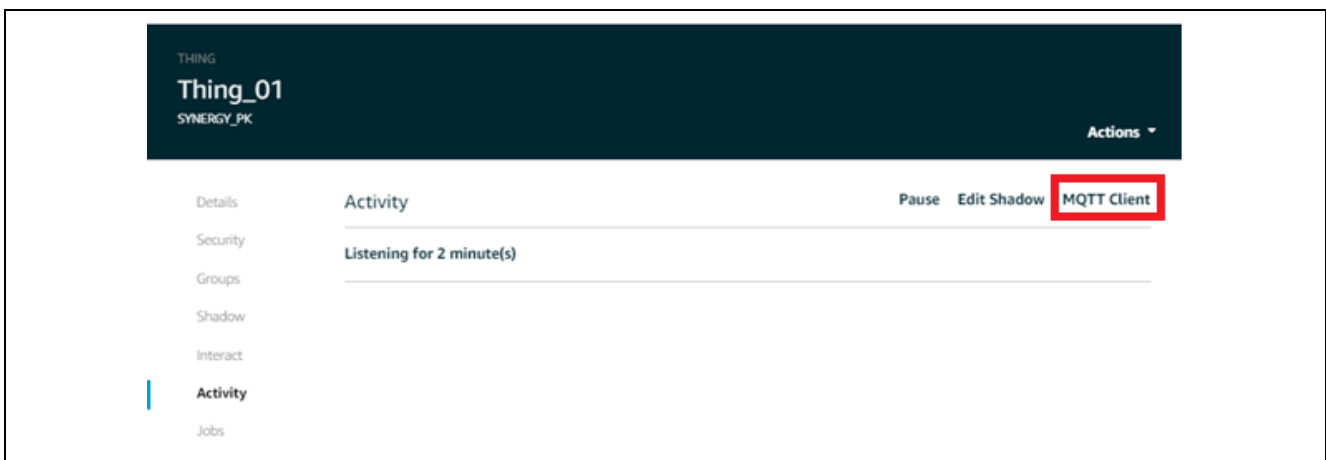
1. Login to your AWS IoT Core account and select your Thing that you created in section 3.3.



2. Click the **Activity** tab shown in the following screen.



3. Click **MQTT Client** in the following screen on the top right corner of the window.



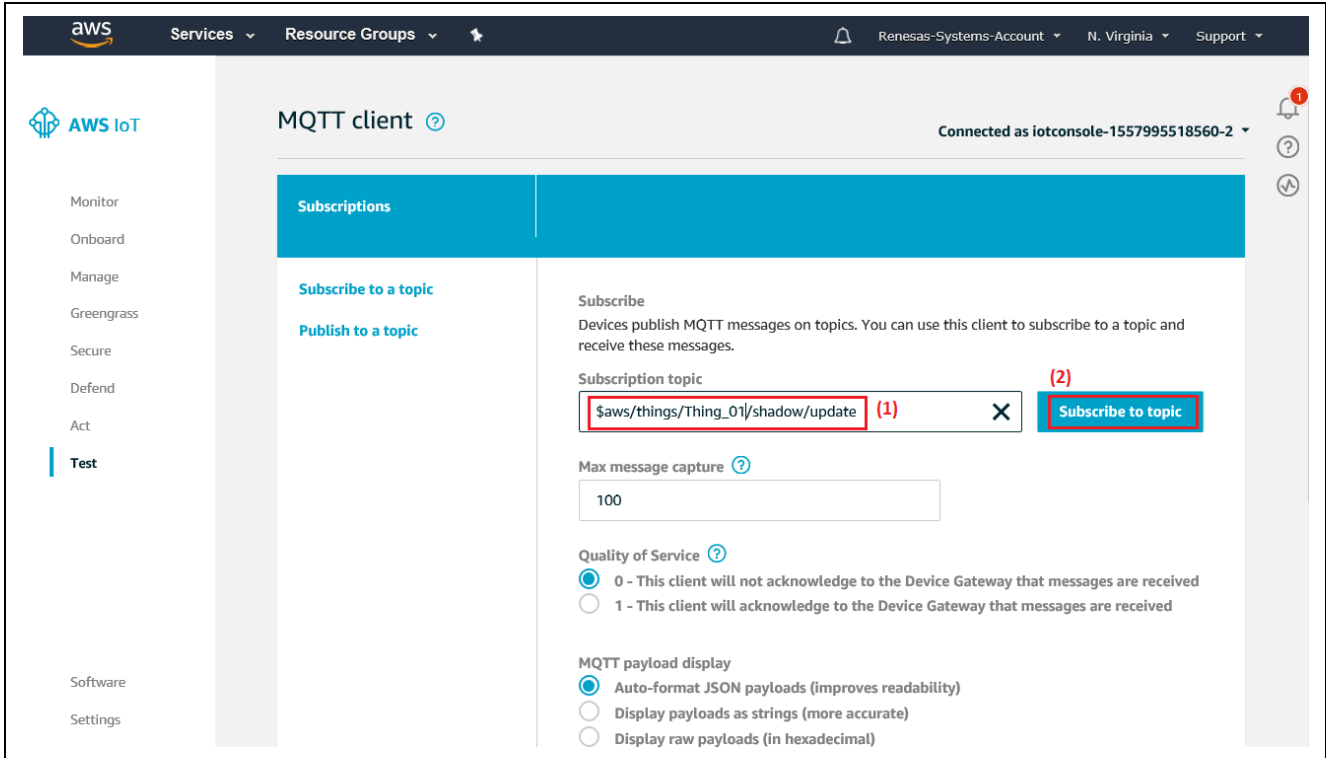
The MQTT Client window opens on the AWS IoT Core. You can subscribe to the topics you want to listen and publish your MQTT message on the topic you want to publish.

4. In this application, the user can subscribe to 2 different topics –for sensor data and for LED states. The subscription topics include two topics:

**User LED state:**

`$aws/things/<your thing name>/shadow/update`

Type the Subscription topic in the **Subscription topic box** as shown in the following screen. Click the **Subscribe to topic** button. The AWS MQTT Client starts listening to the topic to which you subscribed.

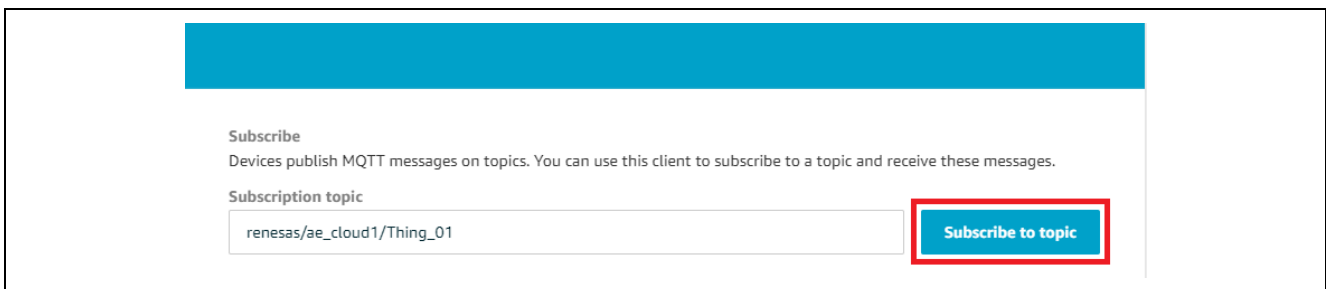


**Sensor data (telemetry data):**

Click **Subscribe to a topic** one more time to bring up below interface and input the sensor data subscription topic.

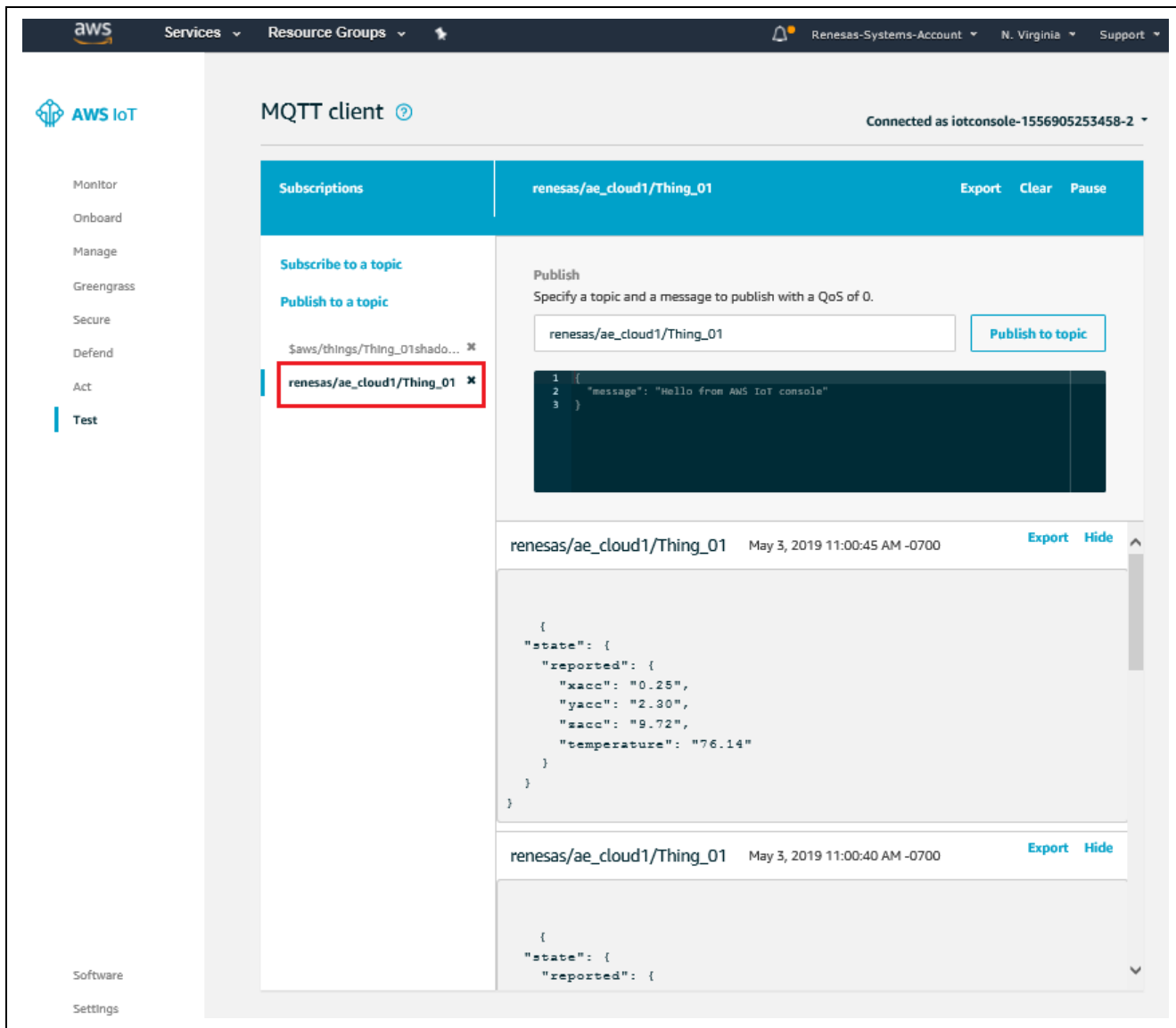
For AE-Cloud1: `renesas/ae_cloud1/<your thing name>`

For AE-Cloud2: `renesas/ae_cloud2/<your thing name>`



When you subscribe to the sensor data topic, you can see the sensor data getting updated every 5 seconds. If there are multiple topics, click the topic (in the left pane shown in the following screen) to see the updates.





5. You can also enter the MQTT topic that is to be published by the cloud in the **Publish box**, as shown in the following screen.

For this application, the following topic is used to publish User LED state:

**User LED state is published in the following topic:**

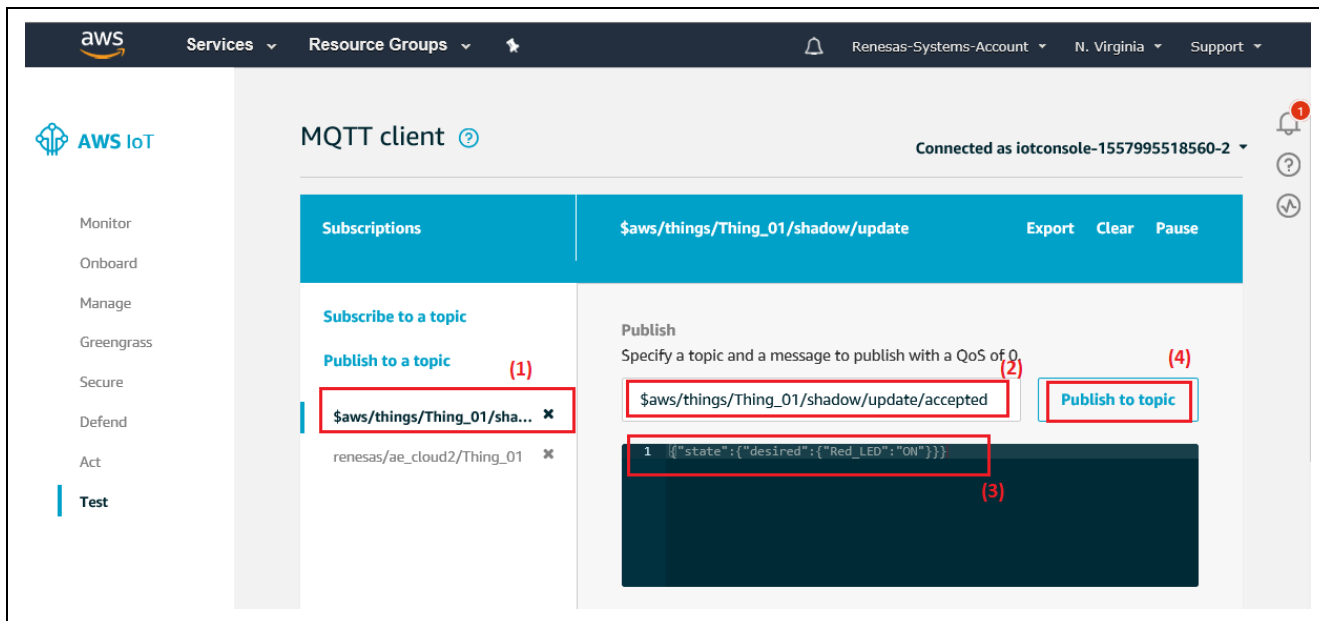
\$aws/things/<your thing name>/shadow/update/accepted

Note: There is no publish topic for sensor data.

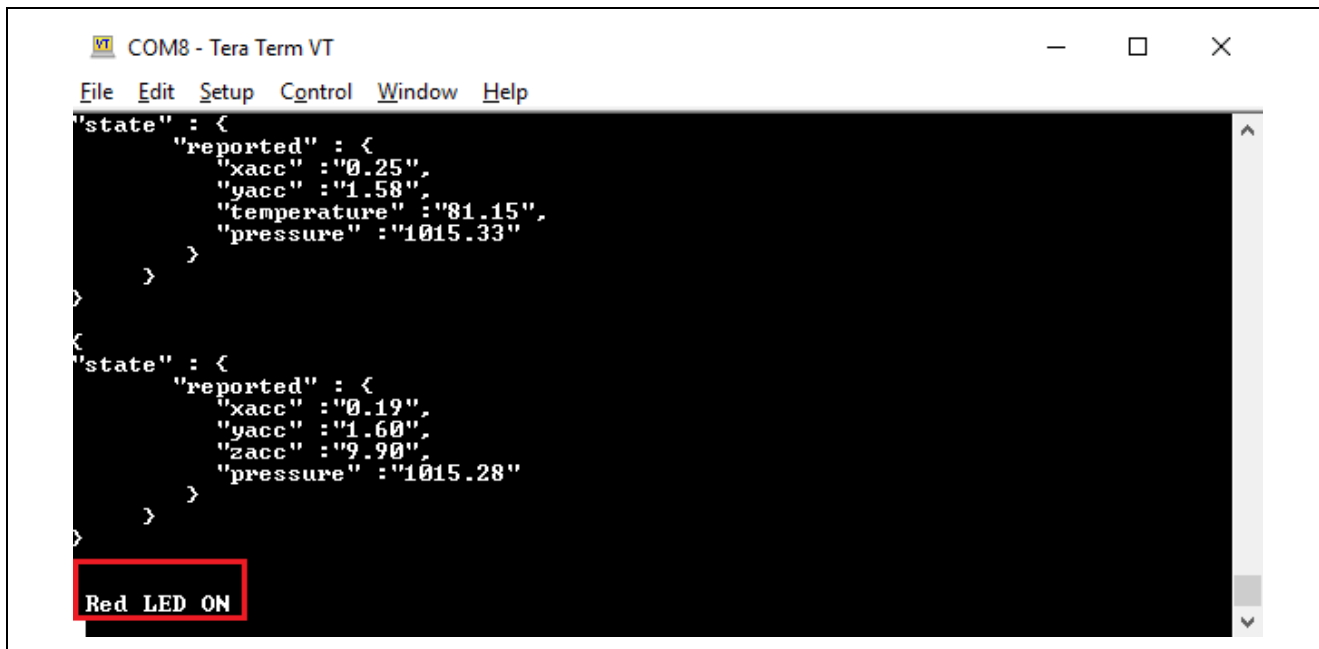
6. Enter the MQTT message in JSON format in the message window below the Publish button.

Refer to section 4.5.3 for details about the different JSON messages used in the demonstration.

7. Click the **Publish to topic** button to publish your MQTT message. Any MQTT clients who are subscribed to your MQTT publish topics will get your published message.



The published message can be seen on the CLI. The corresponding LED should be turned ON or OFF as indicated in the JSON message.



### 4.5.3 Publishing the MQTT Message from AWS MQTT Client

You can publish the MQTT message to turn your LEDs ON/OFF on the AE-CLOUD1/AE-CLOUD2 kit by using messages from the following table. These messages indicate the states for Red, Green, and Yellow LEDs.

Note: The Message under Message Column are **case sensitive**; users need to take care of this while using them to turn the LEDs ON/OFF.

Only enter one message at a time. Copy the message 'as-is' and do not include any extra spaces.

**Table 2. Turning the User LED ON/OFF on your AE-CLOUD1/AE-CLOUD2 kit**

LED State	Message
RED LED ON	{"state":{"desired":{"Red_LED":"ON"}}
RED LED OFF	{"state":{"desired":{"Red_LED":"OFF"}}
YELLOW LED ON	{"state":{"desired":{"Yellow_LED":"ON"}}
YELLOW LED OFF	{"state":{"desired":{"Yellow_LED":"OFF"}}
GREEN LED ON	{"state":{"desired":{"Green_LED":"ON"}}
GREEN LED OFF	{"state":{"desired":{"Green_LED":"OFF"}}

#### 4.5.4 Stopping the Synergy Cloud Connectivity Demonstration

To stop the demonstration, enter the `demo stop` command. Issuing this command de-initializes the IoT Cloud interface, stops it from publishing MQTT messages, and clears any pending MQTT messages from its internal queue. The demonstration can be restarted by typing `demo start` command.

```
>demo stop
De-initializing IoT Service... >done
De-initializing N/W Interface... Done
>
```

**Figure 31. Application demo stop sequence**

#### 4.6 Customizing the demo delays

This application supports failure recovery for a period specified by certain macros. To increase or reduce the time for recovery modify the following in the source code (`MQTT_Config.h`):

**Table 3. Delay settings**

Macro	Purpose	Set to
MQTT_UPDATE_DELAY	Delay between the updates pushed to the cloud.	5 seconds
IOT_NW_RETRY_DELAY	Retry for Network failures	5 seconds
NETWORK_RETRY_CNT	Retry count for network connectivity in case of network failures.	10 seconds
IOT_SERVICE_RETRY_DELAY	Delay between retries for IoT Service connectivity in case of failures.	Currently set to 100
IOT_SERVICE_RETRY_CNT	Retry count for IoT Service.	5 seconds

### 5. Next Steps

- Renesas Synergy Module Guides collateral: [www.renesas.com/synergy/applicationprojects](http://www.renesas.com/synergy/applicationprojects)
- Setting up a client using a device certificate signed by your CA certificate, refer to the link: <https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html>
- Using a self-signed certificate to configure AWS, refer to the link: <https://developer.amazon.com/docs/custom-skills/configure-web-service-self-signed-certificate.html>

For other links to the Synergy Gallery, development tools, and utilities, see the Website and Support section.

### 6. MQTT/TLS Reference

- SSP v1.7.8 User's Manual ([www.renesas.com/synergy/ssp](http://www.renesas.com/synergy/ssp)).
- AWS IoT documentation (<https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>).

### 7. Known Issues and Limitations

1. Restarting using `demo start` command may fail on certain networks when using Ethernet or Wi-Fi.
2. Occasional outages in cloud connectivity can be noticed during the demo due to changes in the cloud server. Contact the Renesas support team for questions.

## Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	<a href="http://www.renesas.com/synergy/software">www.renesas.com/synergy/software</a>
Synergy Software Package	<a href="http://www.renesas.com/synergy/ssp">www.renesas.com/synergy/ssp</a>
Software add-ons	<a href="http://www.renesas.com/synergy/addons">www.renesas.com/synergy/addons</a>
Software glossary	<a href="http://www.renesas.com/synergy/softwareglossary">www.renesas.com/synergy/softwareglossary</a>
Development tools	<a href="http://www.renesas.com/synergy/tools">www.renesas.com/synergy/tools</a>
Synergy Hardware	<a href="http://www.renesas.com/synergy/hardware">www.renesas.com/synergy/hardware</a>
Microcontrollers	<a href="http://www.renesas.com/synergy/mcus">www.renesas.com/synergy/mcus</a>
MCU glossary	<a href="http://www.renesas.com/synergy/mcuglossary">www.renesas.com/synergy/mcuglossary</a>
Parametric search	<a href="http://www.renesas.com/synergy/parametric">www.renesas.com/synergy/parametric</a>
Kits	<a href="http://www.renesas.com/synergy/kits">www.renesas.com/synergy/kits</a>
Synergy Solutions Gallery	<a href="http://www.renesas.com/synergy/solutionsgallery">www.renesas.com/synergy/solutionsgallery</a>
Partner projects	<a href="http://www.renesas.com/synergy/partnerprojects">www.renesas.com/synergy/partnerprojects</a>
Application projects	<a href="http://www.renesas.com/synergy/applicationprojects">www.renesas.com/synergy/applicationprojects</a>
Self-service support resources:	
Documentation	<a href="http://www.renesas.com/synergy/docs">www.renesas.com/synergy/docs</a>
Knowledgebase	<a href="http://www.renesas.com/synergy/knowledgebase">www.renesas.com/synergy/knowledgebase</a>
Forums	<a href="http://www.renesas.com/synergy/forum">www.renesas.com/synergy/forum</a>
Training	<a href="http://www.renesas.com/synergy/training">www.renesas.com/synergy/training</a>
Videos	<a href="http://www.renesas.com/synergy/videos">www.renesas.com/synergy/videos</a>
Chat and web ticket	<a href="http://www.renesas.com/synergy/resourcelibrary">www.renesas.com/synergy/resourcelibrary</a>

**Revision History**

Rev.	Date	Description	
		Page	Summary
1.00	Sep.24.18	—	Initial version
1.01	Feb.06.19	—	Updated links and section 4, Running the MQTT/TLS Application
1.02	Jun.27.19	—	Updated portions of section 4, Running the MQTT/TLS Application
1.03	Oct.21.19	—	Updated for SSP v1.7.0
1.04	Nov.01.19	—	Updated section 4.3, step 4
1.05	Jan 29.2021		Fixed the Base64 length issue

## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
  - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
  - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://www.renesas.com)

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:  
[www.renesas.com/contact/](http://www.renesas.com/contact/).