Renesas RA Family

RA AWS MQTT/TLS Cloud Connectivity Solution

Introduction

This application note describes the IoT Cloud connectivity solution in general, provides a brief introduction to IoT Cloud providers like Amazon Web Services (AWS), and covers the FSP MQTT/TLS module and its features. 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.

This application note enables developers to effectively use the FSP MQTT/TLS modules in end-product design. Upon completion of this guide, developers will be able to add “AWS Core MQTT”, “Mbed TLS”, and “secure sockets on FreeRTOS plus TCP” using the Ethernet interface, configure them correctly for the target application, and write code using the included application example code as a reference for an efficient starting point.

References to detailed API descriptions, and other application projects that demonstrate more advanced uses of the module, are in the *FSP User’s Manual* (available at: https://renesas.github.io/fsp/), which serves as a valuable resource in creating more complex designs.

This MQTT/TLS AWS Cloud Connectivity solution is supported on the EK-RA6M3.

Applies to:

- RA6M5 MCU Group
- RA6M4 MCU Group
- RA6M3 MCU Group
- RA6M2 MCU Group
- RA6M1 MCU Group

Required Resources

To build and run the MQTT/TLS application example, the following resources are needed.

Development tools and software

- e^2 studio IDE v2022-01 or later (renesas.com/us/en/software-tool/e-studio)
- Flexible Software Package (FSP) 3.6.0 or later (renesas.com/us/en/software-tool/flexible-software-package-fsp)
  - Installer for e^2 studio and FSP can be found at ehttps://github.com/renesas/fsp
- SEGGER RTT Viewer V 7.60e or later (https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/)

Hardware

- Renesas RA™ EK-RA6M3 kit (renesas.com/ra/ek-ra6m3)
- PC running Windows® 10 and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari)
- Micro USB cables (included as part of the kit)
- Ethernet cable (CAT5/6)
- Ethernet switch or router with an Ethernet port which has access to the Internet.

Prerequisites and Intended Audience

This application note assumes that the user is adept in operating the Renesas e^2 studio ISDE with Flexible Software Package (FSP). If not, we recommend reading and following the procedures in the *FSP User Manual* sections for ‘Starting Development’ including ‘Debug the Blinky Project’. Doing so enables familiarization with e^2 studio and FSP and validates proper debug connection to the target board. In addition, this application note assumes prior knowledge of MQTT/TLS and its communication protocols.

The intended audience is users who want to develop applications with MQTT/TLS modules using Renesas RA™ RA6 MCU Series.

Note: If you are a first-time user of e^2 studio and FSP, we highly recommend you install e^2 studio and FSP on your system in order to run the Blinky Project and to get familiar with the e^2 studio and FSP development environment before proceeding to the next sections.

Note: This Application Project and App Note can only use versions FSP V3.6.0 or later. Since the AWS MQTT client module is deprecated starting FSP V3.0.0, this application note, which uses Core MQTT, should be used as a reference for operating the equivalent AWS Core MQTT module.

Prerequisites

1. Access to online documentation available in the Cloud Connectivity References section
2. Access to latest documentation for identified Renesas Flexible Software Package
3. Prior knowledge of operating e^2 studio and built-in (or standalone) RA Configurator
4. Access to associated hardware documentation such as User Manuals, Schematics, and so forth

Using this Application Note

Section 1 of this document covers the General Overview of Cloud Connectivity, AWS IOT Core, MQTT and TLS protocols, and device certificates and keys used in Cloud Connectivity.

Sections 2 to 5 cover the use of the FSP configurator to add the Core MQTT, Secure Sockets, Integrated TLS client and MQTT components to the project.

Sections 6.1 and 6.2 cover Cloud connectivity Application Project architecture and its software components overview.

Section 6.3 covers the step-by-step procedure to recreate the bundled Application Project using the FSP configurator.

Note: For a quick validation using the provided application project, you can skip the above sections and go to section 6.5 for instructions on importing, building, and running the Application Project on the EK board. You are still required to provide necessary user credentials for the application as described in sections 6.4, 6.5, and 6.6 before validation can be done using the steps described in the section 6.7.

Section 6.4 covers the cloud side configuration required to run the Application Project.

Section 6.5 covers the importing, building, and running the Application Project on the EK board.

Section 6.6 covers the user specific credentials to run the application.

Section 6.7 covers the validation of the Application Project from the board and from the cloud.

Contents

1. Introduction to Components for Cloud Connectivity

1.1 General Overview

The Internet-of-Things (IoT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. The 'things' in this definition are objects in the physical world (physical objects) or information world (virtual) that can be identified and integrated into communication networks. In the context of the IoT, a 'device' is a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage, and data processing. Communication is often performed with providers of network-hosted services, infrastructure, and business applications to process/analyze the generated data and manage the devices. Such providers are called Cloud Service Providers. While there are many manufacturers for devices and cloud service providers, for the context of this application note, the device is a Renesas RA Microcontroller (MCU) connecting to services provided by Amazon Web Services (AWS) for IoT.

1.2 Cloud Service Provider

AWS IoT is a platform that enables users to connect devices to AWS services and other devices, secure data and interactions, process, and act upon device data, and enable applications to interact with devices even when they are offline. As a Cloud Service Provider, AWS IoT provides the ability to:

- Connect and manage devices
- Secure device connections and data
- Process and act upon device data
- Read and set device state at any time

Figure 1 summarizes the features provided by AWS IoT.

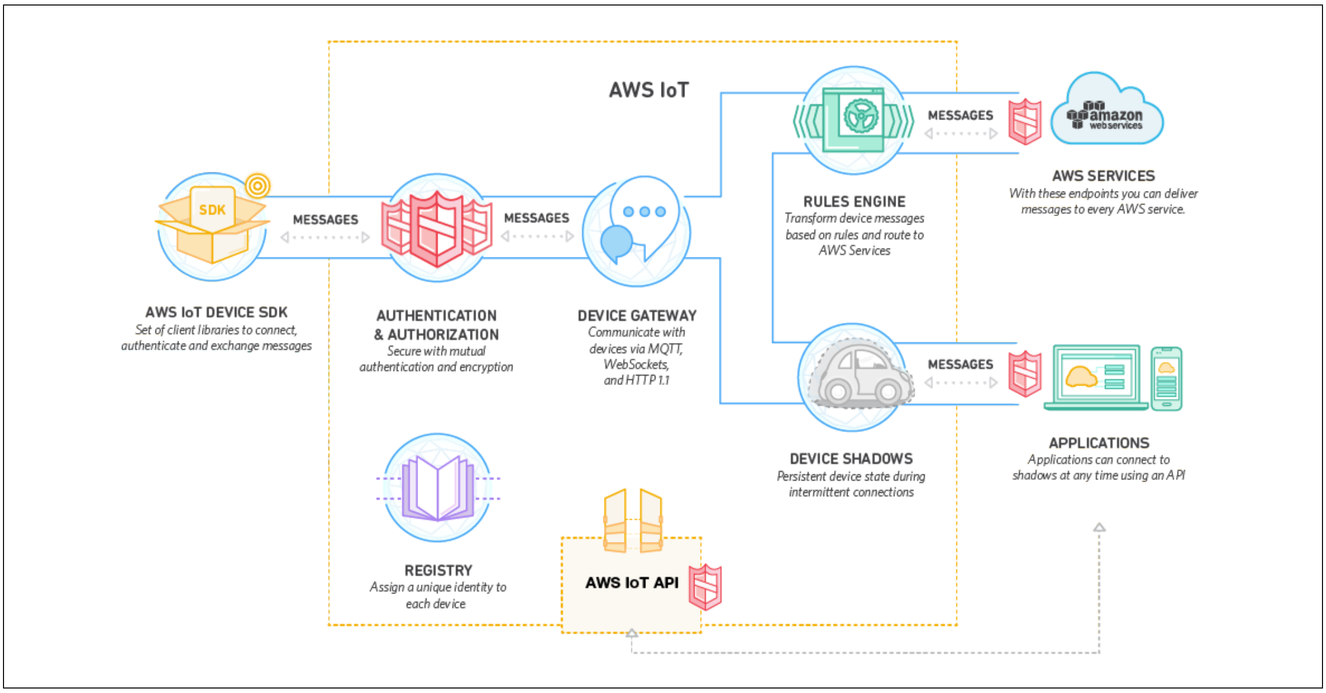

AWS IoT
SDK
MESSAGES
MESSAGES
MESSAGES
MESSAGES
amazon web services
AWS SERVICES
With these endpoints you can deliver messages to every AWS service.
RULES ENGINE
Transform device messages based on rules and route to AWS Services
AWS IoT DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
AUTHENTICATION & AUTHORIZATION
Secure with mutual authentication and encryption
DEVICE GATEWAY
Communicate with devices via MQTT, WebSockets, and HTTP 1.1
DEVICE SHADOWS
Persistent device state during intermittent connections
APPLICATIONS
Applications can connect to shadows at any time using an API
REGISTRY
Assign a unique identity to each device
AWS IoT API

Figure 1. AWS IoT Features, Service Components, and Data Flow Diagram

A key feature provided by AWS is the AWS IoT Software Development Kit (SDK) written in C, which allows devices such as sensors, actuators, embedded microcontrollers, or smart appliances; to connect, authenticate, and exchange messages with AWS IoT using the MQTT, HTTP, or WebSocket's protocols. *This application note focuses on configuring and using the AWS IoT Device SDK and the included MQTT protocol available through the Renesas Flexible Software Package (FSP) for Renesas RA MCUs.*

1.3 AWS IoT Core

AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT Core can support billions of devices and trillions of messages. It can process and route messages to AWS endpoints and to other devices reliably and securely. With AWS IoT Core, customer applications can keep track of all devices, all the time, even when they are not connected.

AWS IoT Core addresses security concerns for the infrastructure by implementing mutual authentication and encryption. AWS IoT Core provides automated configuration and authentication upon a device's first connection to AWS IoT Core, as well as end-to-end encryption throughout all points of connection, so that data is never exchanged between devices and AWS IoT Core without proven identity.

This application note focuses on complementing the security needs of AWS IoT Core through installing a proven identity for the RA MCU by storing a X.509 certificate and asymmetric cryptography keys in Privacy Enhanced Mail (PEM) format in the on-board flash. The RA MCU has on-chip security features, such as Key Wrapping, to protect the private key associated with the public key and the certificate associated with the device[1]. Additionally, RA MCUs can also generate asymmetric keys using features of the Secure Cryptography Engine (SCE) and API available through the FSP. The SCE accelerates symmetric encryption/decryption of data between the connected device and AWS IoT, allowing the ARM Cortex-M processor to perform other application specific computations.

1.4 MQTT Protocol Overview

This application note features Message Queuing Telemetry Transport (MQTT) as it is a lightweight communication protocol specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements. MQTT uses a publish/subscribe architecture which is designed to be open and easy to implement, with up to thousands of remote clients capable of being supported by a single server. These characteristics make MQTT ideal for use in constrained environments where network bandwidth is low or where there is high latency and with remote devices that might have limited processing capabilities and memory. *The RA MCU device in this application note implements a Core MQTT which communicates with AWS IoT and exchanges example telemetry information, such as MCU temperature and MCU GPIO status.*

1.5 TLS Protocol Overview

The primary goal of the Transport Layer Security (TLS) protocol is to provide privacy and data integrity between two communicating applications or endpoints. AWS IoT mandates use of secure communication. Consequentially, all traffic to and from AWS IoT is sent securely using TLS. TLS protocol version 1.2 or later ensures the confidentiality of the application protocols supported by AWS IoT. A variety of TLS Cipher Suites are supported. This application note configures the RA Flexible Software Package for the MCU based device to provide the following capabilities and AWS IoT negotiates the appropriate TLS Cipher Suite configuration to maximize security.

Table 1. TLS Capabilities in RA FSP

Secure Crypto Hardware Acceleration	Supported
Key Format Supported	AES, ECC, RSA
Hash	SHA-256
Cipher	AES
Public Key Cryptography	ECC, ECDSA, RSA
Message Authentication Code (MAC)	HKDF

On top of these supported features, Mbed Crypto middleware also supports a variety of features which can be enabled through the RA Configurator. Refer to the FSP User's Manual section for the Crypto Middleware (rm_psa_crypto).

[1] This application note does not focus on using Key Wrapping for securely storing the private key for devices deployed in a production environment.

1.6 Device Certificates, CA, and Keys

Device Certificates, Certificate Authorities (CA), and Asymmetric Key Pairs create the foundation for trust needed for a secure environment. The background information on these commonly used components in AWS is as follows:

A *digital certificate* is a document in a known format that provides information about the identity of a device. The X.509 standard includes the format definition for public-key certificate, attribute certificate, certificate revocation list (CRL), and attribute certificate revocation list (ACRL). X.509-defined certificate formats (X.509 Certificates) are commonly used on the internet and in AWS IoT for authenticating a remote entity/endpoint, that is, a Client and/or Server. In this application note, an X.509 certificate and asymmetric cryptography key pair (public and private keys) are generated from AWS IoT and installed (during binary compilation) into the RA MCU device running the Core MQTT to establish a *known identity*. In addition, a root Certification Authority (CA) certificate is also downloaded and used by the device to authenticate the connection to the AWS IoT gateway.

Certification authority (CA) certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two CAs[2]. The root CA certificate allows devices to verify that they're communicating with AWS IoT Core and not another server impersonating AWS IoT Core.

The public and private keys downloaded from AWS IoT use RSA algorithms for encryption, decryption, signing and verification[3]. These key pairs, and certificates are used together in the TLS process to:

1. Verify device identity.
2. Exchange symmetric keys, for algorithms such as AES, for encrypting and decrypting data transfers between endpoints.

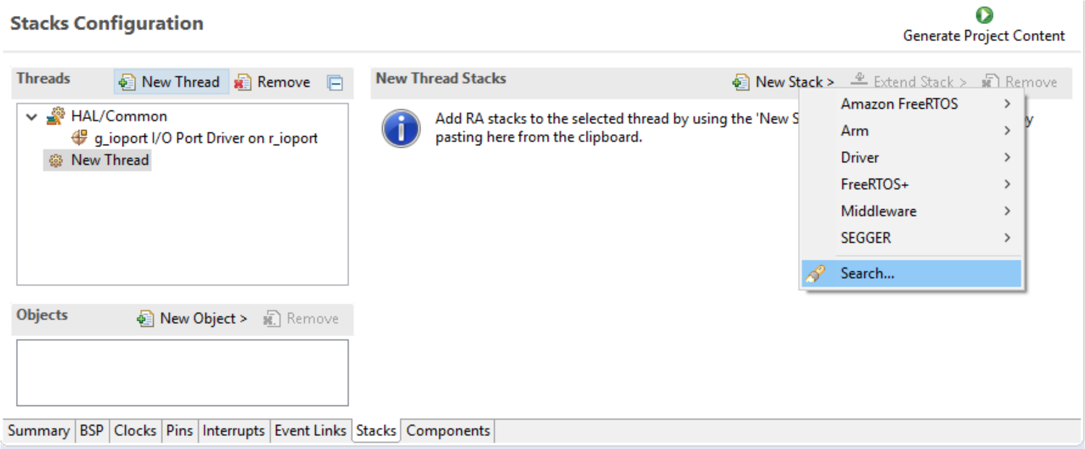
2. AWS Core MQTT with RA FSP

The AWS MQTT library included in RA FSP can connect to either AWS MQTT or to any third party MQTT broker such as Mosquitto. The complete documentation for the library can be found on the AWS IoT Device SDK C: MQTT website. Primary features supported by the library are:

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server or any MQTT broker.
- Non-secure MQTT connections to Mosquitto servers.[4]

The AWS Core MQTT can be directly imported into a Thread Stack and is configured through the RA Configuration Perspective. To add the AWS Core MQTT to a new thread in e^2 studio, open `Configuration.xml` with the RA Configuration. While ensuring that the correct thread is selected on the left, use the tab for **Stacks > New Stack > Search** and search for the keyword AWS Core MQTT.

[2] The root CA certificate provided by AWS IoT is signed by Digital Guardian.
[3] Public Key length used is 2048 bits.
[4] Recommended for local server testing and not for production/deployment.



Add RA stacks to the selected thread by using the 'New S
pasting here from the clipboard.

Figure 2. AWS Core MQTT Module Selection

Adding the AWS Core MQTT Stack results in the default configuration with *some unmet dependencies*, as shown below in Figure 3.

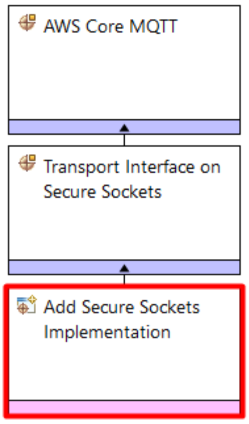

AWS Core MQTT
Transport Interface on Secure Sockets
Add Secure Sockets Implementation

Figure 3. AWS Core MQTT Stack View

While the AWS Core MQTT stack shown contains a lot of dependencies and configurable properties, most default settings can be used as-is. The following change is needed to meet all unmet dependencies (marked in red) for the AWS Core MQTT stack added to a new project (as shown above):

Enable Mutex and Recursive Mutex usage support as needed by IoT SDK and FreeRTOS in the created Thread properties.

Note: **Properties > Common > General > Use Mutexes > Enabled**
Properties > Common > General > Use Recursive Mutexes > Enabled

Upon completion of the above step, the AWS Core MQTT is ready to accept a Secure Socket Implementation, which has dependencies on using a TLS Session and an underlying TCP/IP implementation.

Additional documentation on the AWS Core MQTT is available in the FSP User's Manual under **RA Flexible Software Package Documentation > API Reference > Modules > AWS Core MQTT**.

3. Secure Sockets Implementation

The AWS Secure Sockets module provides an API that is based on the widely used BSD Sockets. While the RA FSP contains a Secure Socket Implementation for both Wi-Fi and Ethernet, this application and application note focuses on the use of an on-board Ethernet.

Secure Sockets can be added to the Thread Stack by clicking on **Add Secure Sockets Implementation > New > Secure Sockets on WiFi or Secure Sockets on FreeRTOS Plus TCP (for Ethernet)**.

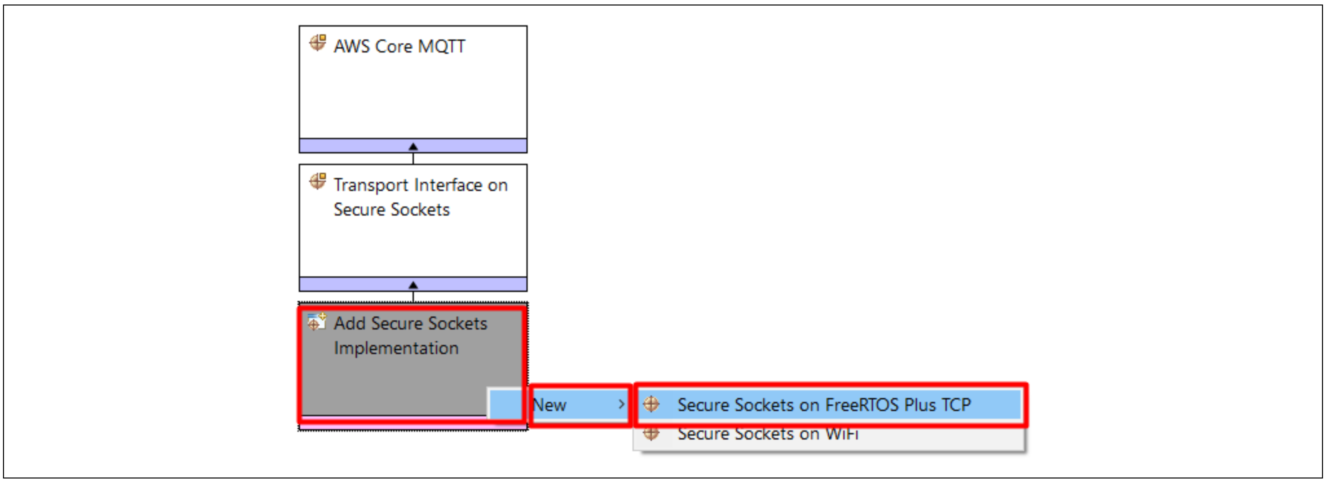

AWS Core MQTT
Transport Interface on Secure Sockets
Add Secure Sockets Implementation
New
Secure Sockets on FreeRTOS Plus TCP
Secure Sockets on WiFi

Figure 4. Adding Secure Socket to the Core MQTT Module

Upon addition, the needed stack is complete and has unmet dependencies for the dependent modules.

Now hover the cursor over the red blocks and the error will pop up. Make the appropriate settings.

- For the **Buffer Allocation** error: Choose the heap implementation using **New Stack > RTOS > FreeRTOS Heap 4**. Also, set **Dynamic Memory allocation** using **Application Thread > Properties > Memory Allocation > Support Dynamic Allocation > Enabled**.
- For the **MbedTLS(Crypto Only)** error, the MBEDTLS_ECDH_C in MbedTLS must be defined when using MbedTLS. Set by selecting **the MbedTLS(Crypto Only) module and navigating tp properties > Public Key Cryptography(PKC|ECC| MBEDTLS_ECDH_C > Define**.
- For the **RTOS Heap memory** error, set Heap Memory allocation using **Application Thread > Properties > Memory Allocation > Total Heap Size > 0x40000**
- The flash file system for the persistent storage is needed. This can be added by clicking on **Add AWS PKCS11 PAL > New > AWS PKCS11 PAL on LittleFS**.
- Add heap by navigating to the BSP tab. Then go to **Properties > | RA Common | Heap Size of 0x1000. This is required to do malloc with LittleFS** and other standard library functions.

After all the appropriate settings to take care of the errors due to the missing configuration, the new configurator snapshot looks clean with no errors as shown below in Figure 5.

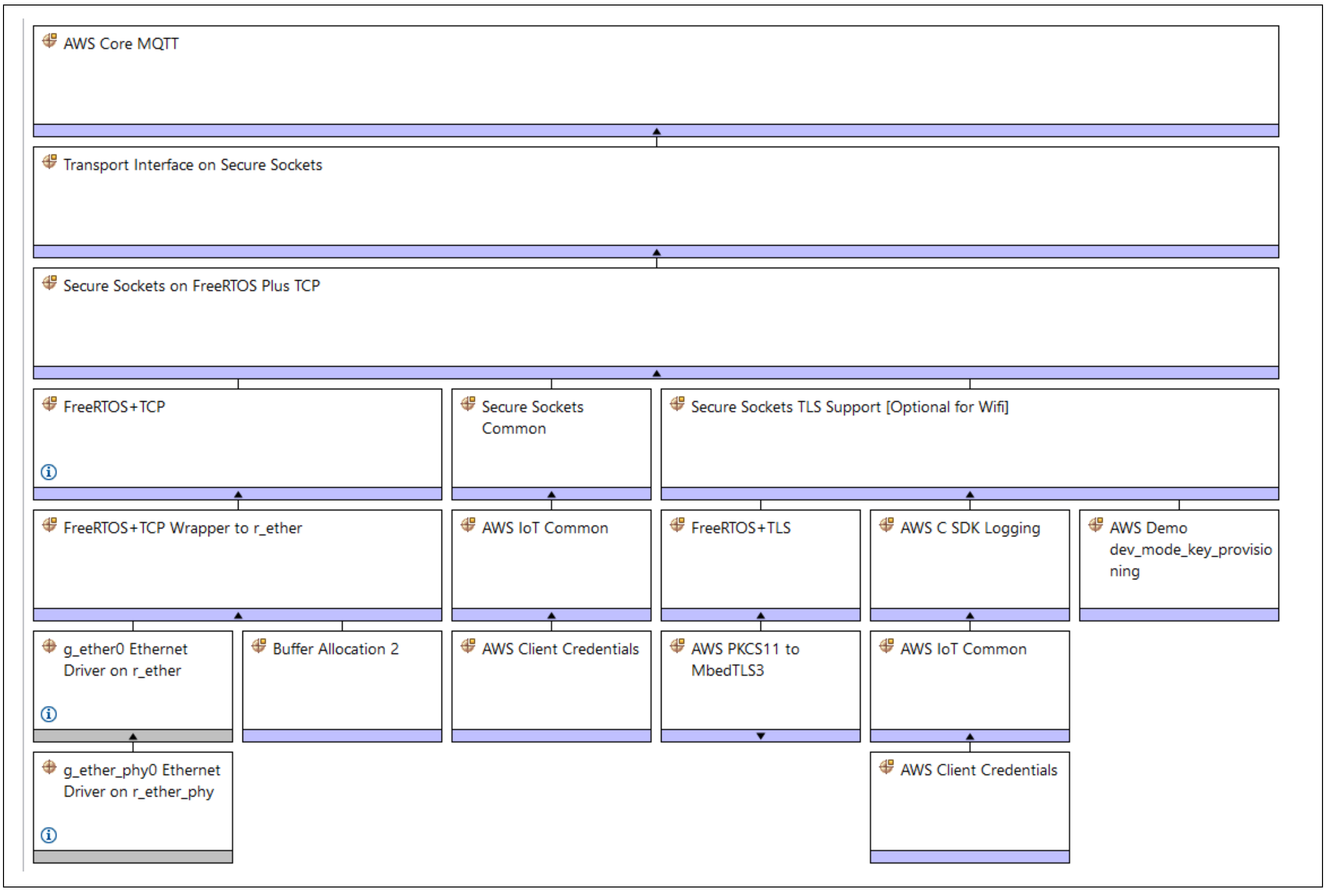

AWS Core MQTT
Transport Interface on Secure Sockets
Secure Sockets on FreeRTOS Plus TCP
FreeRTOS+TCP
Secure Sockets Common
Secure Sockets TLS Support [Optional for Wifi]
FreeRTOS+TCP Wrapper to r_ether
AWS IoT Common
FreeRTOS+TLS
AWS C SDK Logging
AWS Demo dev_mode_key_provisioning
g_ether0 Ethernet Driver on r_ether
Buffer Allocation 2
AWS Client Credentials
AWS PKCS11 to MbedTLS3
AWS IoT Common
g_ether_phy0 Ethernet Driver on r_ether_phy
AWS Client Credentials

Figure 5. Expanded Secure Socket Module

Additional documentation on AWS Secure Sockets is available in the FSP User's Manual under **RA Flexible Software Package Documentation > API Reference > Modules > AWS Secure Sockets**.

4. Mbed TLS

Arm® Mbed™ TLS is Arm's implementation of the TLS protocols as well as the cryptographic primitives required by those implementations. Mbed TLS is also solely used for its cryptographic features even if the TLS/SSL portions are not used.

Secure Socket TLS Support uses FreeRTOS+TLS which eventually uses Mbed TLS. Use of Mbed TLS requires configuration and operation of the Mbed Crypto module which in turn operates the SCE on the MCU.

The following underlying mandatory changes are needed by a project using the Secure Sockets on FreeRTOS+TLS module:

1. Use FreeRTOS heap implementation scheme 4 (first fit algorithm with coalescence algorithm) or scheme 5 (first fit algorithm with coalescence algorithm with heap spanning over multiple non-adjacent/non-contiguous memory regions.
2. Enable support for dynamic memory allocation in FreeRTOS.
3. Enable Mbed TLS platform memory allocation layer.
4. Enable the Mbed TLS generic threading Layer that handles default locks and mutexes for the user and abstracts the threading layer to use an alternate thread-library.
5. Enable Elliptic Curve Diffie Helleman library.
6. Change FreeRTOS Total Heap Size to a value greater than 0x1500.

Additional documentation on the Mbed TLS is available in the FSP User's Manual under **RA Flexible Software Package Documentation > API Reference > Modules > Crypto Middleware (rm_psa_crypto)**.

5. MQTT Module APIs Usage

The AWS Core MQTT is documented online. Table 2 lists APIs provided by AWS Core MQTT that are used as a part of the Application Example.

Table 2. MQTT Module APIs

MQTT_Init	Initializes an MQTT context
MQTT_Connect	Establishes an MQTT session
MQTT_Subscribe	Sends MQTT SUBSCRIBE for the given list of topic filters to the broker
MQTT_Publish	Publishes a message to the given topic name
MQTT_Ping	Sends an MQTT PINGREQ to broker
MQTT_Unsubscribe	Sends MQTT UNSUBSCRIBE for the given list of topic filters to the broker
MQTT_Disconnect	Disconnect an MQTT session
MQTT_ProcessLoop	Loop to receive packets from the transport interface. Handles keep-alive
MQTT_ReceiveLoop	Loop to receive packets from the transport interface. Does not handle keep-alive
MQTT_GetSubAckStatusCodes	Parses the payload of an MQTT SUBACK packet that contains status codes corresponding to topic filter subscription requests from the original subscribe packet
MQTT_Status_strerror	Error code to string conversion for MQTT statuses.
MQTT_PublishToResend	Get the packet ID of next pending publish to be resent

6. Cloud Connectivity Application Example

6.1 Overview

This application project demonstrates the use of APIs available through the Renesas FSP-integrated modules for Amazon IoT SDK C, Mbed TLS module, Amazon FreeRTOS, and HAL Drivers operating on Renesas RA MCUs. Network connectivity is established using Ethernet. The application running on a Renesas Evaluation Kit also serves as a reference system for the operation of Core MQTT, Mbed TLS/Crypto, and Ethernet configuration, using the FSP configurator. The application may be used as a starting point for inspiring other customized cloud-based solutions using Renesas RA MCUs. In addition, it marginally demonstrates the operation and setup of cloud services available through the cloud service provider.

The upcoming sub-sections show step-by-step creation of a device and security credentials policies as required by the AWS IOT on the cloud side to communicate with the end devices. The example, accompanying this documentation, demonstrates Subscribe and Publish messaging between and Core MQTT and MQTT Broker, periodic publication of temperature data, asynchronous publication of a “User Push Button” event from the MCU to the Cloud. The device is also subscribed to receive actuation events (LED ON/OFF) from the Cloud, thereby showing two-way control.

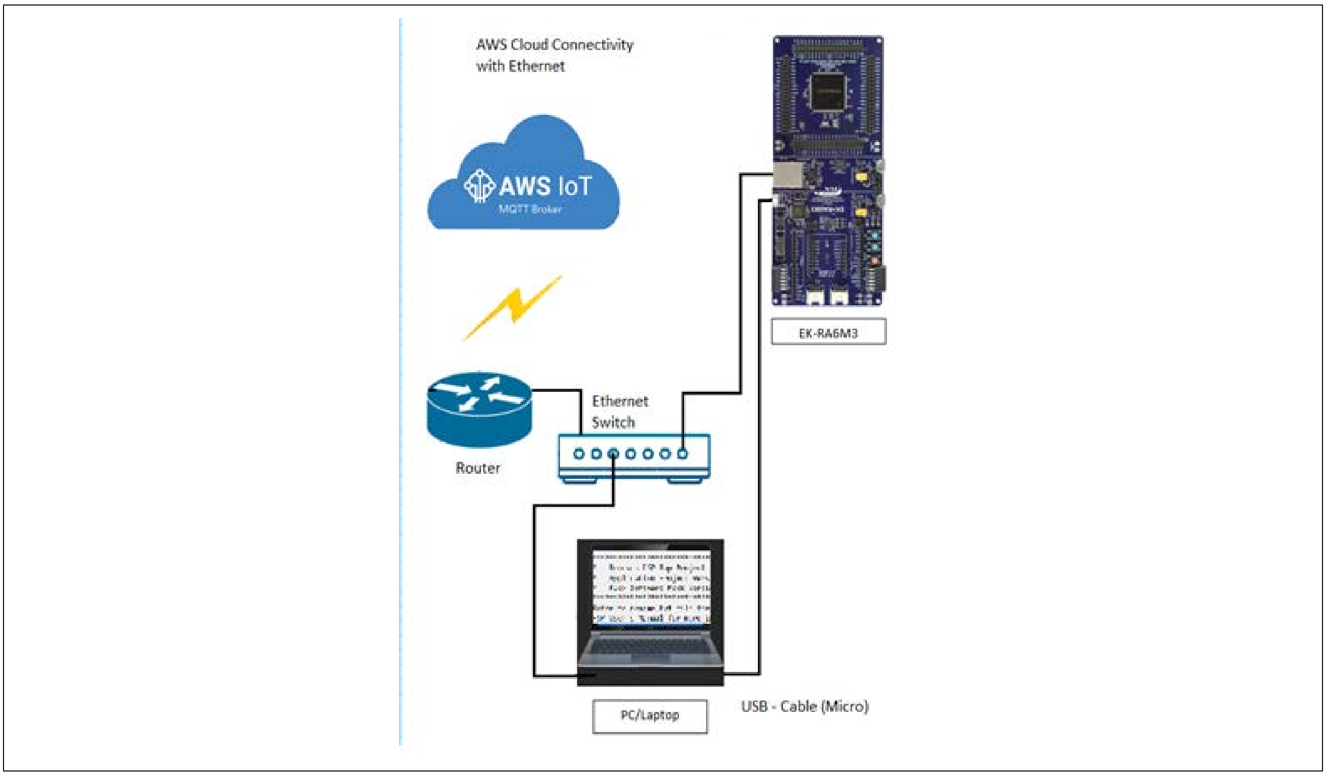


Figure 6. Application Projects High-Level Overview for Ethernet

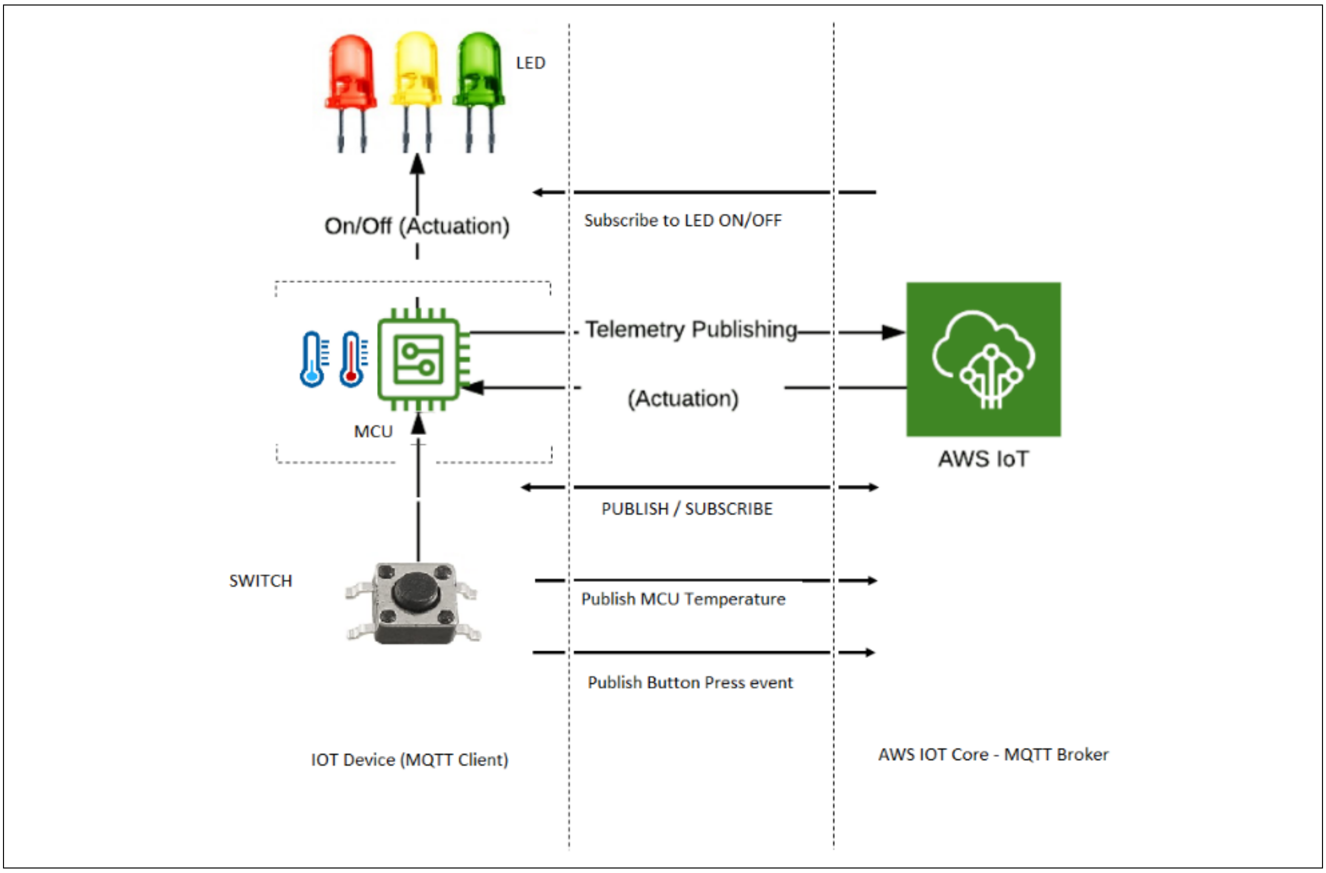


Figure 7. MQTT Publish/Subscribe to/from AWS IoT Core

6.2 MQTT/TLS Application SW Architecture Overview

The following files from this application project serve as a reference

Table 3. Files Used in Application Project

No.	Filename	Purpose
1	`src/application_thread_entry.c`	Contains data structures, functions, and main thread used in a Cloud Connectivity application.
2	`src/common_utils.h`	Contains macros, data structures, and functions commonly used across the project.
3	`src/hal_entry.c`	Unused file automatically generated by FSP. This file is used for non-RTOS based projects.
4	`src/mqtt_demo_helpers.c`	Contains data structures and functions used in thenmqtt interface for Cloud Connectivity.
5	`src/ mqtt_demo_helpers.h`	Accompanying header for exposing functionality provided by `mqtt_demo_helpers.c`.
6	`src/SEGGER_RTT/SEGGER_RTT.c`	Implementation of SEGGER real-time transfer (RTT) which allows real-time communication on targets which support debugger memory accesses while the CPU is running.
7	`src/SEGGER_RTT/SEGGER_RTT.h`	
8	`src/SEGGER_RTT/SEGGER_RTT_Conf.h`	
9	`src/SEGGER_RTT/SEGGER_RTT_printf.c`	
10	`src/usr_config.h`	To customize the user configuration to run the application.
11	`src/usr_hal.c`	Contains data structures and functions used for the Hardware Abstraction Layer (HAL) initialization and associated utilities.
12	`src/usr_hal.h`	Accompanying header for exposing functionality provided by `usr_hal.c.`
13	`src/usr_data.h`	Accompanying header file for the application thread.
14	`src/usr_network.c`	Contains data structures and functions used to operate the FreeRTOS TCP/IP and Ethernet Module. This file is for Ethernet-specific use.
15	`src/usr_network.h`	Accompanying header for exposing functionality provided by `usr_network.c.` This file is for Ethernet-specific use.
16	`src/ backoffAlgorithm/backoff_algorithm.c`	Retry algorithms with random back off for the next retry attempt
17	`src/ backoffAlgorithm/backoff_algorithm.h`	Retry algorithms with random back off for the next retry attempt header file
18	`src/subcription_manager/ mqtt_subscription_manager.c`	MQTT Subscription manager, which handles the callback
19	`src/subcription_manager/ mqtt_subscription_manager.h`	Associated header file for MQTT Subscription manager, which handles the callback.

Note: `application_thread_entry.c` is the file auto generated as part of the project creation. Users are required to add code this file.

Note: To run the application with the supplied code, `application_thread_entry.c` is available part of this App note bundle can be merged or overwritten with the autogenerated file.

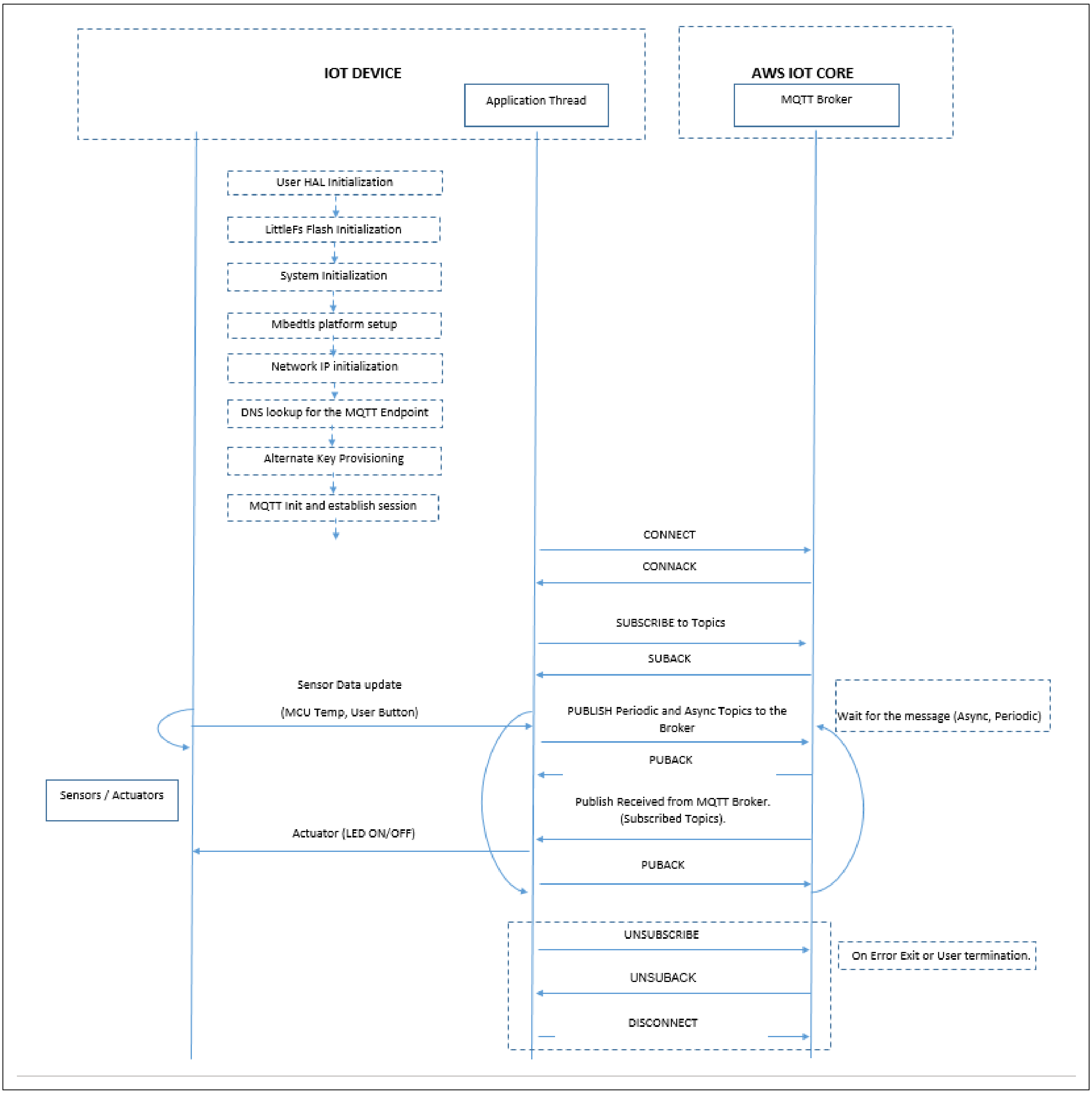

IOT DEVICE
Application Thread
AWS IOT CORE
MQTT Broker
User HAL Initialization
LittleFs Flash Initialization
System Initialization
Mbedtls platform setup
Network IP initialization
DNS lookup for the MQTT Endpoint
Alternate Key Provisioning
MQTT Init and establish session
CONNECT
CONNACK
SUBSCRIBE to Topics
SUBACK
Sensor Data update
(MCU Temp, User Button)
PUBLISH Periodic and Async Topics to the Broker
Wait for the message (Async, Periodic)
PUBACK
Sensors / Actuators
Publish Received from MQTT Broker. (Subscribed Topics).
Actuator (LED ON/OFF)
PUBACK
UNSUBSCRIBE
On Error Exit or User termination.
UNSUBACK
DISCONNECT

Figure 8. Application Example Implementation Details

6.3 Creating the Application Project using the FSP Configurator

This section shows complete steps to create the project from the start using the e^2 studio and FSP configurator. The table below shows the step-by-step process in creating the Project. It is assumed that the user is familiar with the e^2 studio and FSP configurator. Launch the installed e^2 studio for the FSP.

Table 4. Step-by-step Details for Creating the Application Project for Ethernet and Wi-Fi (EK-RA6M3)

	Step	Intermediate Steps
1	**Project Creation:**	**File → New → C/C++ Project**
2	**Project Template:**	**Templates for New RA C/C++ Project → Renesas RA C/C++ Project → Next**
3	**e^2 studio - Project Configuration (RA C Executable Project) →**	**Project Name (Name for the Project)** Note: Input your desired name for the project -> **Next**
4	**Device Selection →**	FSP Version: **3.6.0**
		Board: **EK-RA6M3**

	Step	**Intermediate Steps**
		Device: **R7FA6M3AH3CFC**
		Language: C
5	Select Tools	Toolchain: **GNU ARM Embedded** (Default)
		Toolchain version: **(**10.3.1.20210824**)**
		Debugger: **J-Link ARM** **Next→FreeRTOS (v10.4.3-LTS.Patch.2+fsp.3.6.0)**
6	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: **FreeRTOS(v3.6.0)** → **Next**
	Project Template	**Project Template Selection: FreeRTOS – Minimal – Static Allocation → Finish**
7	**Stacks Tab (Part of the FSP Configurator)→**	**Threads → New Thread**
8	**Config Thread Properties→**	
		Symbol: **application_thread**
		Name: **Application Thread**
		Stack size: **32768 Bytes**
		Priority: **1**
		Thread Context: **NULL**
		Memory Allocation: **Static**
9	**Generic RTOS configs under thread (Additional configuration on top of the Default Config provided by FSP)**	
	Common → General	Use Mutexs: **Enabled**
		Use Recursive Mutexes: **Enabled**
		Max Task Name Len: **32**
	Common → Memory Allocation	Support Dynamic Allocation: **Enabled**
		Total Heap Size: **0x40000** Note: For Ethernet application Total Heap Size: 0x40000 is required
10	**Add the Heap Implementation in HAL/Common**	
	New Stack →	**RTOS → FreeRTOS Heap 4**
11	**Adding the AWS MQTT Wrapper Module to the Application Thread**	
	Note: Now the Newly created thread (Application thread) is ready to add new Stack (Here the AWS MQTT Wrapper is added)	
	New Stack →	**Networking → AWS MQTT Wrapper**
13	**Adding the AWS MQTT Wrapper module brings in the AWS Core MQTT module to the configurator. Now the dependency modules are required to be added.**	
13a	Under the “Add transport Interface”, add	**New → AWS Transport Interface on Secure Sockets**
13b	Under the “Add secure socket Implementation”, add	**New → AWS Secure Socket on FreeRTOS Plus TCP**
14	**Adding persistent storage support for AWS PKCS11 and resolve the error in the configurator by selecting the Heap size in the BSP Tab. Right-click on pink highlighted stack to**	
	Add AWS PKCS11 PAL module →	**New → AWS PKCS11 PAL on LittleFS**
	BSP Tab → RA Common→	Heap size : **0x1000**
15	**Some dependency related to TLS Support are needed to be resolved to remove the error in the FSP configurator by modifying the “MbedTLS(Crypto Only)” Property Settings.**	
	Common → Platform →	MBEDTLS_PLATFORM_MEMORY: **Define**
	Common → General →	MBEDTLS_THREADING_C: **Define**
	Common → General →	MBEDTLS_THREADING_ALT: **Define**
	Common → Public Key Cryptography (PKC) →	ECC → MBEDTLS_ECDH_C: **Define**

	Step	Intermediate Steps
19	**FreeRTOS + TCP Configuration** Note: This is only applicable for the Ethernet application project. Most of the default settings remain the same, except few of the default configuration needs to be changed	
	Common →	**DHCP callback function → Enable**
		DHCP Send Discover After Auto IP: Enable
		Let TCP use windowing mechanism → Enable
20	**Ethernet Driver Configuration**	
	General →	**Flow control functionality → Enable**
	Buffers →	**Number of Tx Buffers → 4**
		Number of Rx Buffers → 4
21	**Ether PHY Driver**	
	Module g_ether_phy0 →	**PHY LSI Address → 1**
22	**Adding the HAL Modules as required for the Application Project: Here, ADC, Timer0, Timer1, External IRQ and ELC Modules are used for MCU Temperature, 30 Seconds periodic timer, 1 second Periodic Heartbeat Monitor Timer, Push button switches, and Event linking of ADC Data read respectively.**	
	HAL/Common Stacks → New Stack	**→ Input → External IRQ Driver on r_icu**
	Property Settings for r_icu	Name: **pushButtonS1**
		Channel: **13**
		Trigger: **Rising**
		Digital Filtering: **enabled**
		Digital Filtering Sample Clock (PCLK/64)
		Pin Interrupt Priority: **Priority 10**
		Callback: **pb_callback**
	HAL/Common Stacks → New Stack	**→ Input → External IRQ Driver on r_icu**
	Property Settings for r_icu	Name: **pushButtonS2**
		Channel: **12**
		Trigger: **Rising**
		Digital Filtering: **enabled**
		Digital Filtering Sample Clock (PCLK/64)
		Pin Interrupt Priority: **Priority 10**
		Callback: **pb_callback**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	Property Settings for r_gpt → General	Name: **gpt**
		Channel: **0**
		Mode: **Periodic**
		Period: **30**
		Period Unit: **Seconds**
	Interrupts:	Callback: **NULL**
		Overflow/Crest Interrupt Priority: **Priority 10**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	Property Settings for r_gpt → General	Name: **g_hb_timer**
		Channel: **1**
		Mode: **Periodic**
		Period: **1000**
		Period Unit: **MilliSeconds**
	Interrupts:	Callback: **g_hb_timer_cb**
		Overflow/Crest Interrupt Priority: **Priority 10**
	HAL/Common Stacks → New Stack	**→ System → ELC Driver on r_elc**

	Step	**Intermediate Steps**
	Property Settings for r_elc → Module g_elc Driver on r_elc	Name: **g_elc**
	HAL/Common Stacks → New Stack	**→ Analog → ADC Driver on r_adc**
	Property Settings for r_adc → General	Name: **adc**
		Unit: **0**
		Resolution: **12-bit**
		Alignment: **Right**
		Clear after read: **On**
		Mode: **Single Scan**
		Double-trigger: **Disabled**
	Property Settings for r_adc → Input →	Channel Scan Mask : **Temperature Sensor**
	Property Settings for r_adc → Interrupt →	Normal/Group A Trigger: **GPT0 COUNTER OVERFLOW(overflow)**
		Callback: **adc_mcu_temp_callback**
		Scan End Interrupt Priority: **Priority 10**
23	**Modifying the Pin configuration as required for the Application Projects (UART related Pins, Flow Control Pins, Disable Multi functionality pins as needed to use alternative pin functions).** **Note: The GLCDC Pins are shared with SCI9 (PMOD 1), In this Application Project GLCDC is not used, so disable it if it is enabled by default.**	
	Pins Tab in the FSP configurator →	**Peripherals→ Graphics: GLCDC → GLCDC0**
		Operation Mode: **Disabled**
24	**Modifying the BSP Settings - RA Common for (Main stack and Heap Settings)**	
	Property Settings for RA Common	Main stack size(bytes): **0x1000**
		Heap size (bytes): **0x1000**
25	**Adding FreeRTOS Objects for the Application (Topic Queue needs to be created for the application – Message Queue)**	
	Stacks Tab → Objects →	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_topic_queue**
		Item Size (Bytes): **64**
		Queue Length (Items): **16**
		Memory Allocation: **Static**

The above configuration is a prerequisite to generate the required stack and features for the cloud connectivity application provided with this application note. Once the **Generate Project Content** button is clicked, it generates the source code for the project. The generated source code contains the required drivers, stacks, and middleware. The user application files must be added into the `src` folder.

Note: FSP-generated code needs to be called/used from the application, while some of the middleware needs to be called exclusively as part of the application for proper initialization. For instance, the `Mbedtls_platform_setup ()` call initializes the SCE and TRNG. `SYSTEM_Init()` initializes and prepares the Crypto and Socket libraries.

For the validation of the created project, the same source files listed in the section **MQTT/TLS Application SW Architecture Overview (Table 3)** may be added. Users are required to add the directory path and subdirectory for proper compilation. Refer the enclosed project for more details.

The details of the configurator from the default settings to changed settings are described in the following sections, including the reason for change.

6.4 MQTT/TLS Configuration

This section describes the MQTT and TLS module configuration settings that are done as part of this application example.

The following table lists changes made to a default configuration populated by the RA Configurator.

Table 5. Default Configuration for EK-RA6M3

Property	Original Value	Changed Value	Reason for Change
Application Thread			
Common → General → Use Mutexes	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → Memory Allocation → Support Dynamic Allocation	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → Memory Allocation → Total Heap Size	0	0x40000	Heap required for the FreeRTOS, AWS IOT SDK, Mbed TLS
AWS IoT Common			
Platform Name	Unknown	AWS Cloud Connectivity	This value is user selectable and can be set to any value.
Mbed TLS (Crypto Only)			
Platform → Mbedtls_platform_memory	Undefine	Define	This selection is required in order to support the Mbed_tls.
General → Mbedtls_threading_alt	Undefine	Define	This selection is required in order to support the Mbed_tls to plug in any thread library.
General → Mbedtls_threading_c	Undefine	Define	This selection is required in order to support the Mbed_tls to abstracts the threading layer to allow easy plugging in any thread-library.
Public Key Cryptography → ECC → Mbedtls_ecdh_c	Undefine	Define	This selection is required in order to support the Mbed_tls to enable the ECDH module.
LittleFS (Heap Selection)			
BSP → RA Common Heap Size	0x0	0x1000	Heap selection for Heap 3 and below needs to be done here.
FreeRTOS + TCP Configuration			
Common → DHCP Callback function	Disable	Enable	Callback for DHCP handling
Let TCP use windowing mechanism →	Disabled	Enable	Turns on the TCP Flow control
Ethernet Driver Configuration			
BSP → RA Common Heap Size	0x0	0x1000	Heap selection for Heap 3 and below needs to be done here.
General → Flow control functionality →	Disable	Enable	Flow control selection for Ethernet
Buffers → Number of Tx Buffers →	1	4	Tx Buffer for Data Transmit
Buffers → Number of Rx Buffers →	1	4	Rx Buffer for Data Reception

Property	Original Value	Changed Value	Reason for Change
Ethernet PHY Driver			
Module g_ether_phy0 →	0	1	Select the LSI PHY Address

6.4.1 IoT Cloud Configuration (AWS)

6.4.1.1 AWS IoT Policies

AWS IoT Core policies are JSON (JavaScript Object Notation) documents that authorize a device to perform AWS IoT Core operations. AWS IoT defines a set of policy actions describing the operations and resources for which access can be granted or denied. 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.

```
{
 “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 connected to AWS IoT Core. Thing shadows may be used 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. A user account must be created on the AWS IoT Cloud service before continuing to the next section.

To create an AWS account, open to the following link in a web browser:
https://portal.aws.amazon.com/billing/signup#/start

Fill in the required details and create a user account.

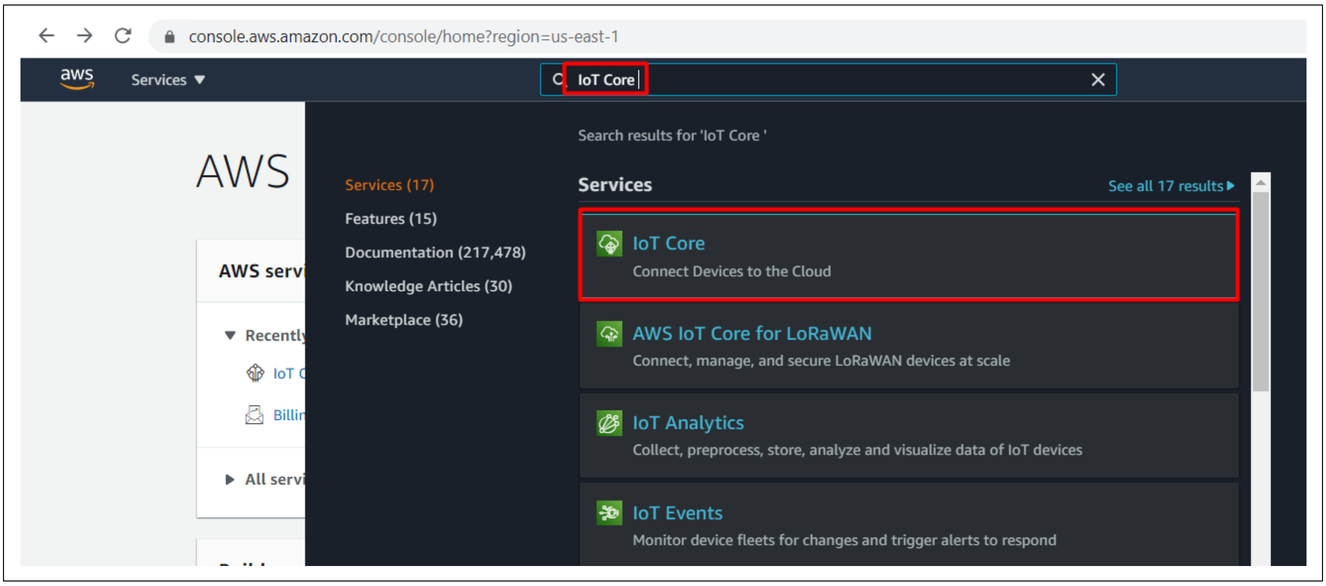
Note: While creating the project, certificates and policies, the screenshots may look slightly different from what is shown in the document and users need to use navigation in the AWS IoT core environment to find corresponding attributes while working on this project.

6.4.2 Creating a Device on AWS IoT Core

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

6.4.2.1 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**.



Connect Devices to the Cloud

Connect, manage, and secure LoRaWAN devices at scale

Collect, preprocess, store, analyze and visualize data of IoT devices

Monitor device fleets for changes and trigger alerts to respond

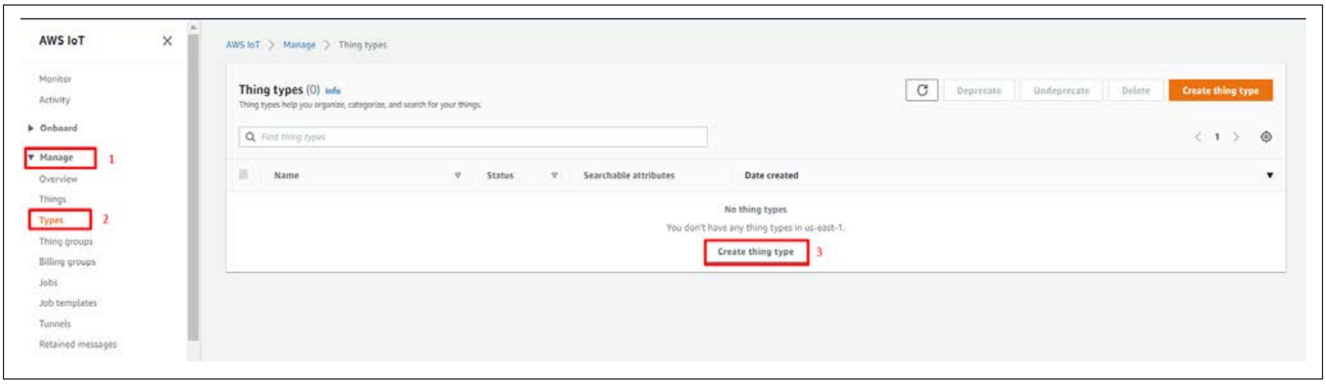
Figure 9. RA Cloud Connectivity AWS IOT Core Selection

6.4.2.2 Create a Thing Type

1. Start creating a device "Thing Type" by selecting **Manage** on the AWS IOT console as shown in the snapshot.

   Note: Be sure to select the appropriate region in the AWS console on the top right corner. A Thing created in one region will not be seen in another region.

2. Now select **Types**.
3. Next, select **Create a thing type** to create a thing type.

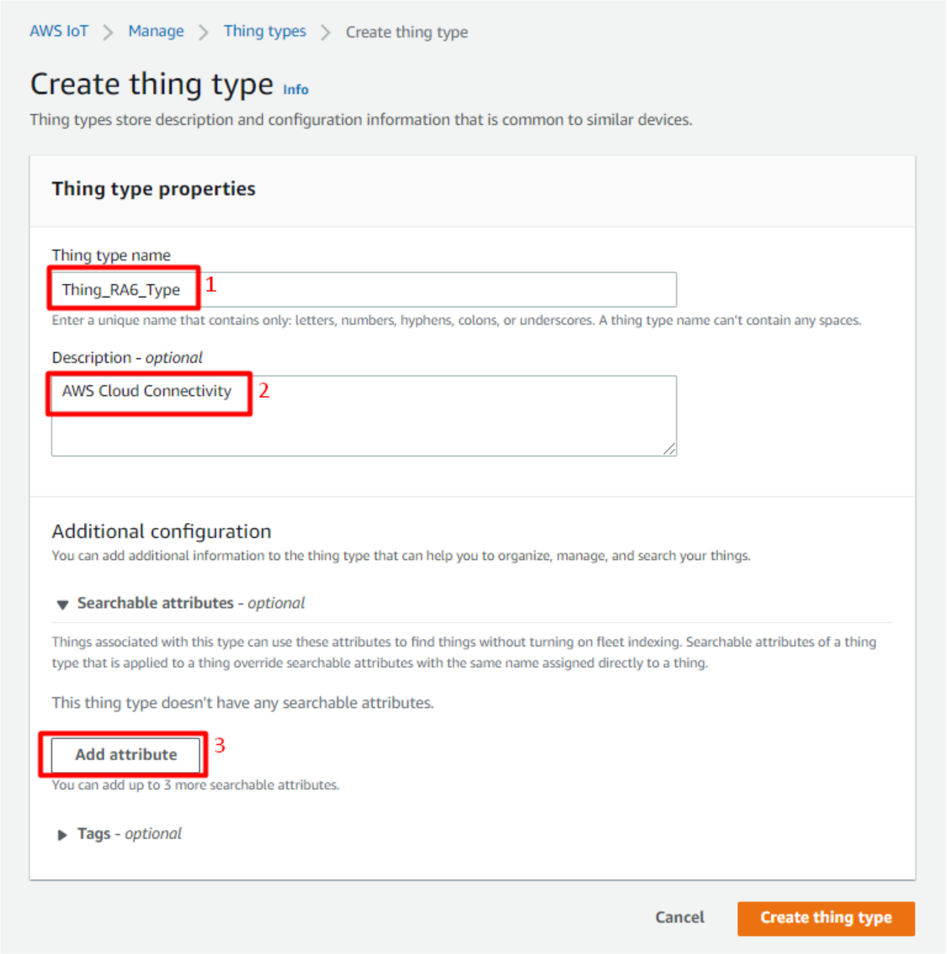


Thing types help you organize, categorize, and search for your things.

No thing types

You don't have any thing types in us-east-1.

Figure 10. Creating Thing Types



AWS IoT > Manage > Thing types > Create thing type

Create thing type Info

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

Thing type properties

Thing type name

Thing_RA6_Type 1

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

Description - *optional*

AWS Cloud Connectivity 2

Additional configuration

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

▼ Searchable attributes - *optional*

Things associated with this type can use these attributes to find things without turning on fleet indexing. Searchable attributes of a thing type that is applied to a thing override searchable attributes with the same name assigned directly to a thing.

This thing type doesn't have any searchable attributes.

Add attribute 3

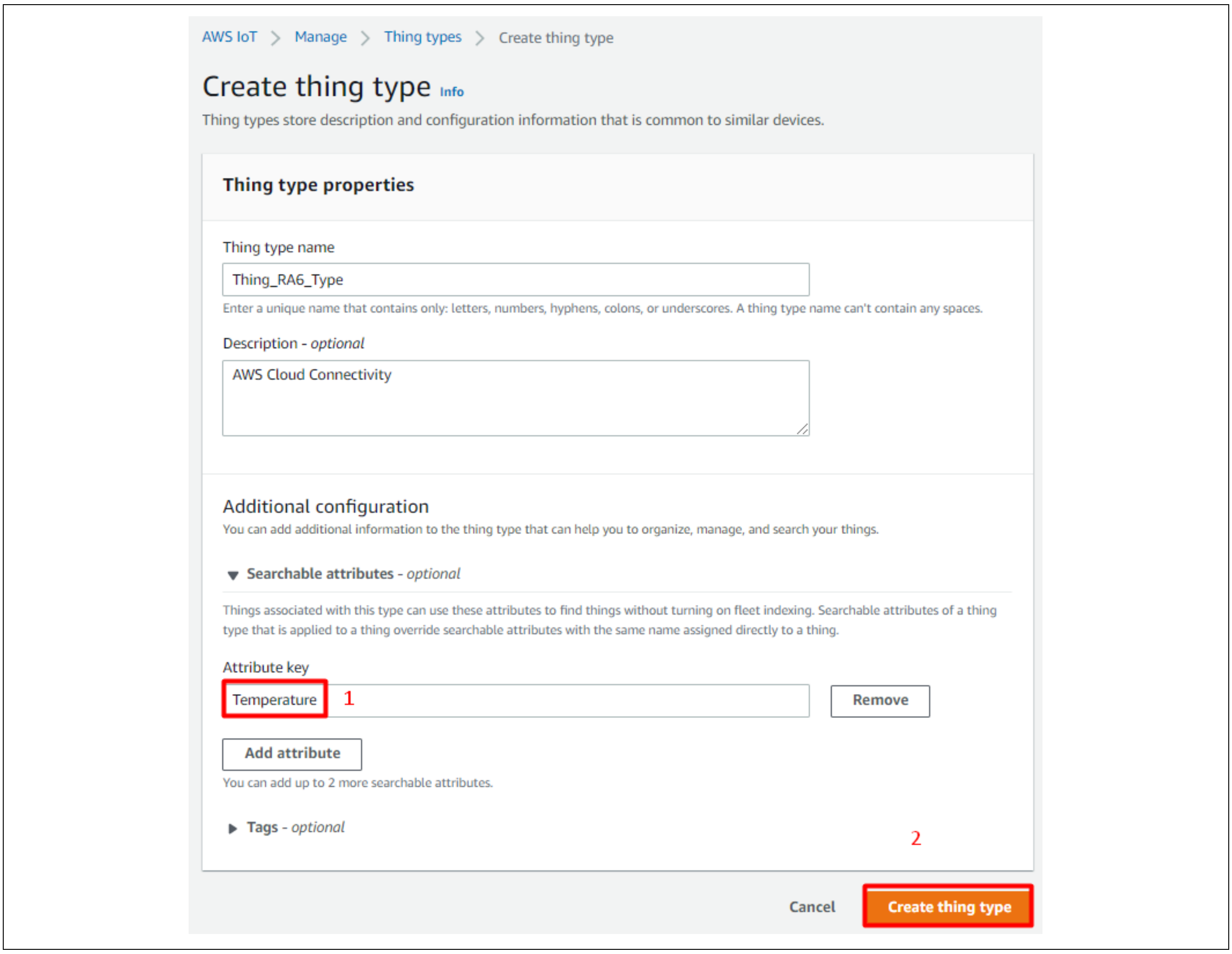
You can add up to 3 more searchable attributes.

▶ Tags - *optional*

Cancel Create thing type

Figure 11. Create a Thing Type

4. Add **Thing_RA6_Type** as the Thing type name and **AWS Cloud Connectivity** as the optional description.
5. Click on **Add Attribute**, which will open window for adding **Temperature** as the Attribute Key. Then click on **Create thing type**.



AWS IoT > Manage > Thing types > Create thing type

Create thing type Info

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

Thing type properties

Thing type name: Thing_RA6_Type

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

Description - *optional*: AWS Cloud Connectivity

Additional configuration

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

Searchable attributes - *optional*

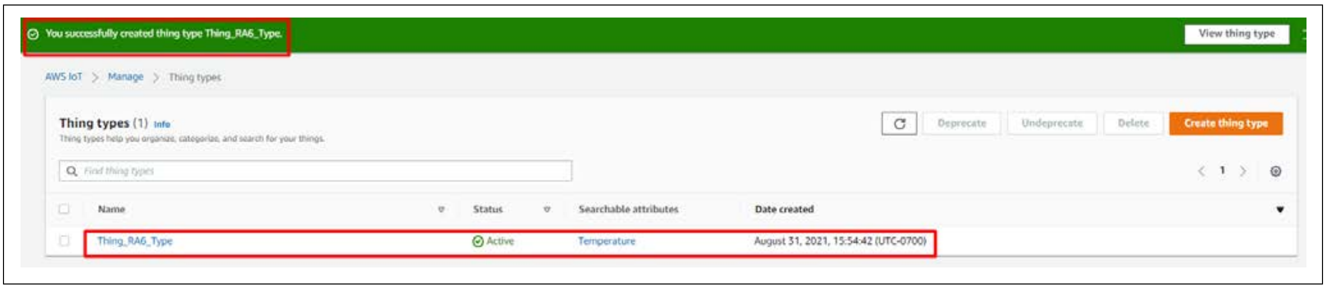
Things associated with this type can use these attributes to find things without turning on fleet indexing. Searchable attributes of a thing type that is applied to a thing override searchable attributes with the same name assigned directly to a thing.

Attribute key: Temperature

You can add up to 2 more searchable attributes.

Figure 12. Create a Thing type

6. The newly created thing type will look as shown in Figure 12. This Thing type name is being used while creating the thing in the next section 6.4.2.3.



You successfully created thing type Thing_RA6_Type.

AWS IoT > Manage > Thing types

Thing types (1) Info

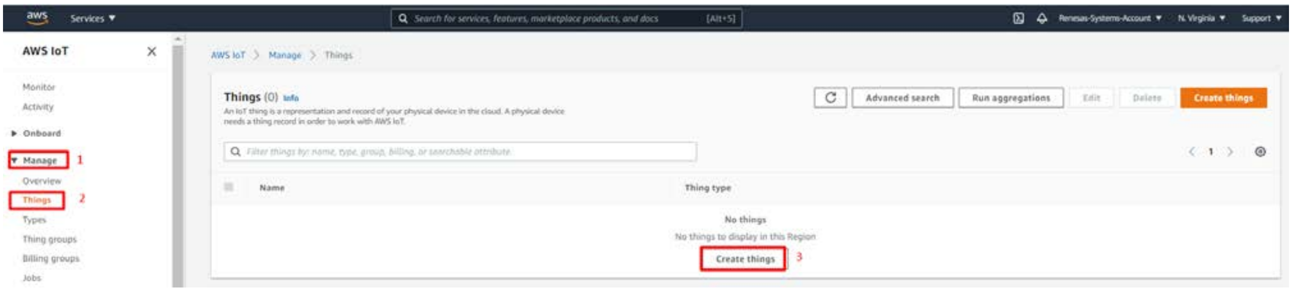
Thing types help you organize, categorize, and search for your things.

Name	Status	Searchable attributes	Date created
Thing_RA6_Type	Active	Temperature	August 31, 2021, 15:54:42 (UTC-0700)

Figure 13. Newly Created Thing Type

6.4.2.3 Create a Thing

1. Start creating a Thing by selecting **Manage** as shown in Figure 14.
2. Now select **Things**.
3. Next, select **Create a thing** to create a thing.



Things (0) Info

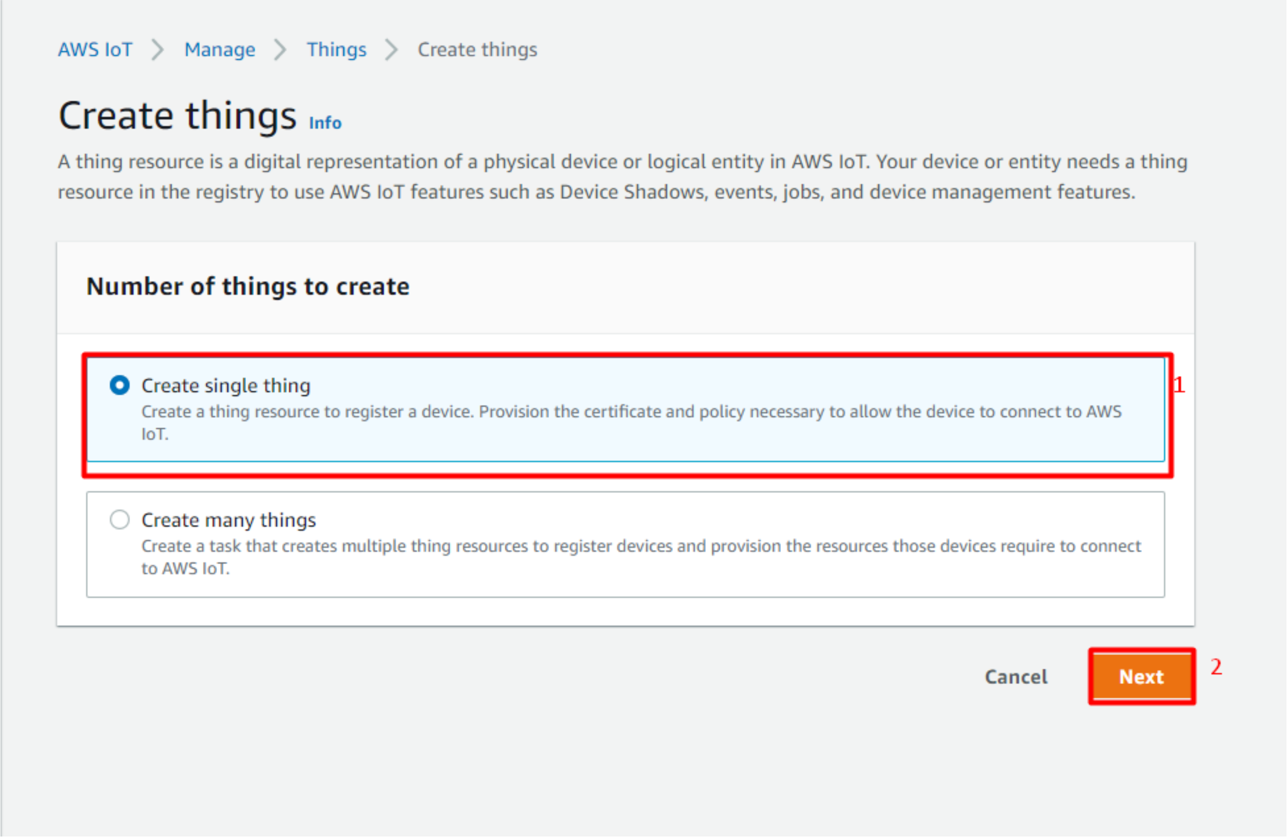
An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

No things

No things to display in this Region

Figure 14. Create a Thing

4. Next, select the **Create a single thing** button. Then click **Next**.



Create things Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Number of things to create

● Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

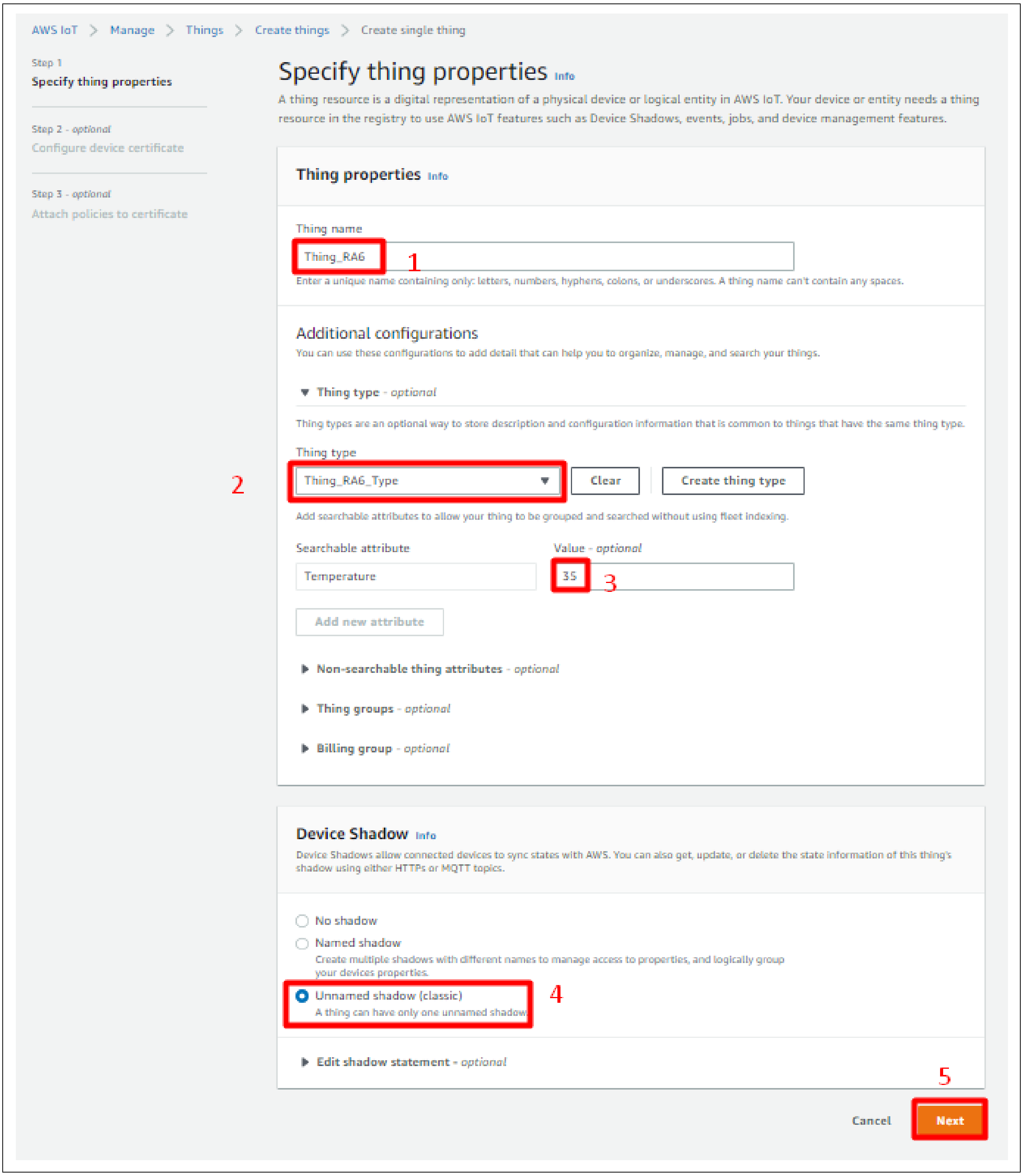
○ Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

Figure 15. Create a Single Thing

5. Enter the **Thing Name.** In the example, a Thing by name **Thing_RA6** is being created.

Note: Remember to store the **Thing Name**. This information is required for future reference during configuration**.**

6. Choose a Thing type by clicking the **Thing type** dropdown and select **Thing_RA6_Type** and choose the attribute value (In this case, **Temperature 35**) and **Unnamed Shadow** for **Device Shadow**.



AWS IoT > Manage > Things > Create things > Create single thing

Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

Specify thing properties Info

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name
Thing_RA6
Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

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

▼ Thing type - *optional*

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

Thing type
Thing_RA6_Type ▼ Clear Create thing type

Add searchable attributes to allow your thing to be grouped and searched without using fleet indexing.

Searchable attribute: Temperature
Value - *optional*: 35

Add new attribute

▶ Non-searchable thing attributes - *optional*

▶ Thing groups - *optional*

▶ Billing group - *optional*

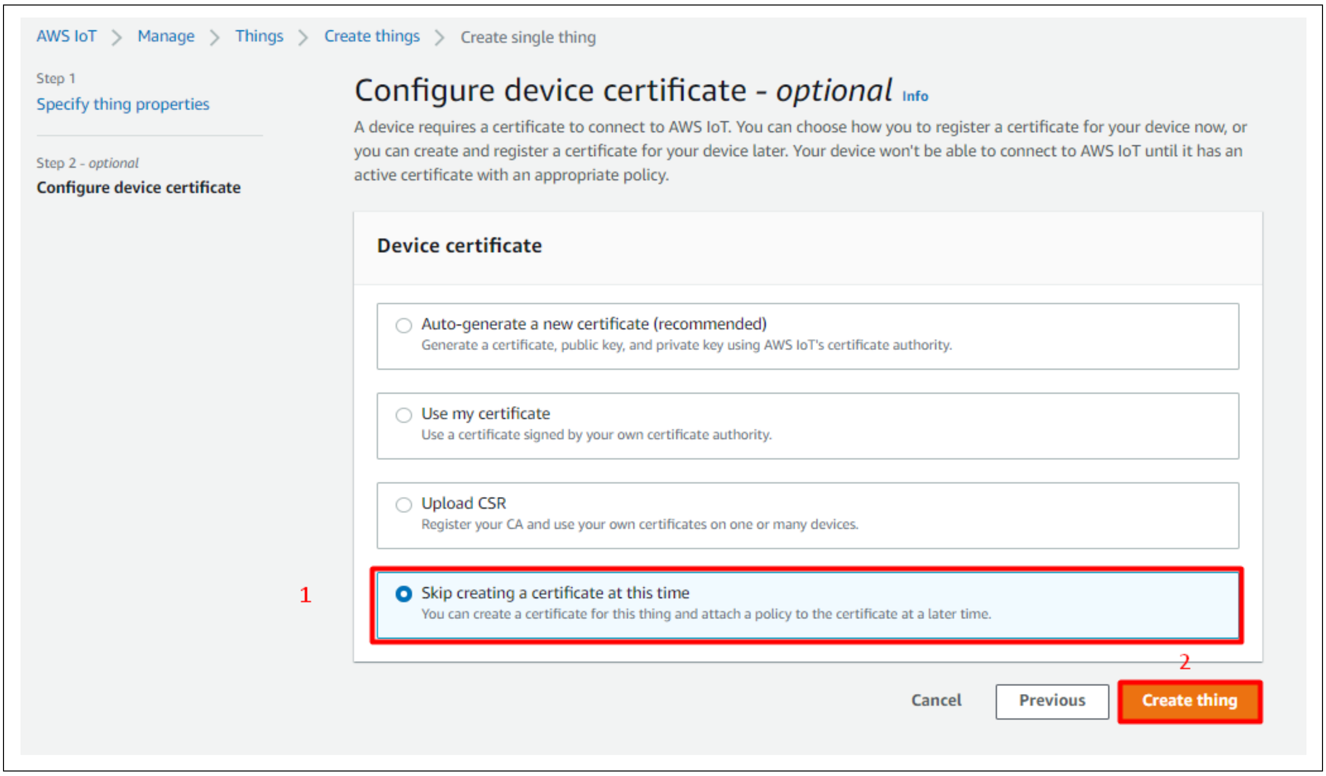
Device Shadow Info
Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

○ No shadow
○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.
● Unnamed shadow (classic)
A thing can have only one unnamed shadow.

▶ Edit shadow statement - *optional*

Figure 16. Create a Thing

7. Click **Next**, which will take you to device certificate configuration. Choose **Skip creating a certificate at this time** and click **Create thing**.



AWS IoT > Manage > Things > Create things > Create single thing

Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Configure device certificate - *optional* Info

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

Device certificate

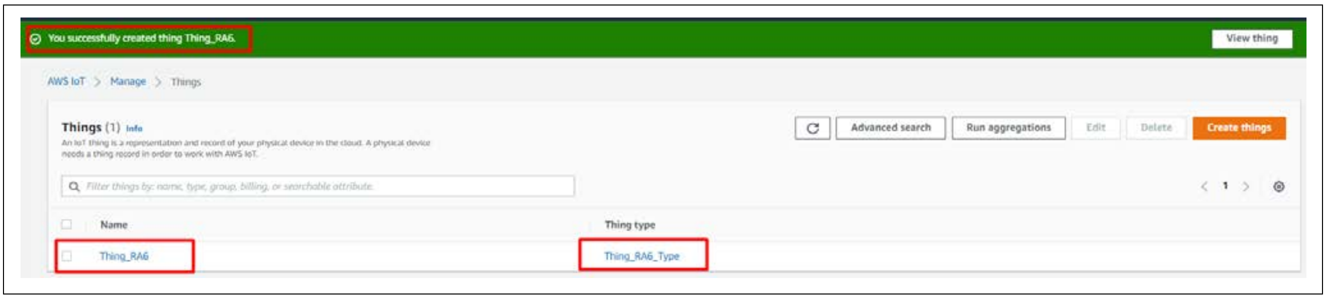
○ Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

○ Use my certificate
Use a certificate signed by your own certificate authority.

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

● Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Figure 17. Create a Thing Without Certificate



You successfully created thing Thing_RA6.

AWS IoT > Manage > Things

Things (1) Info
An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

Name	Thing type
Thing_RA6	Thing_RA6_Type

View thing
Advanced search
Run aggregations
Edit
Delete
Create things
Filter things by: name, type, group, billing, or searchable attribute.
1

Figure 18. Newly Created Thing Without Certificate

6.4.2.4 Create a Policy

1. Start creating the policy required for the Thing you just created by selecting **Secure** and **Policies** from the AWS IOT console as shown in Figure 19.
2. Now click on **Create a Policy**.

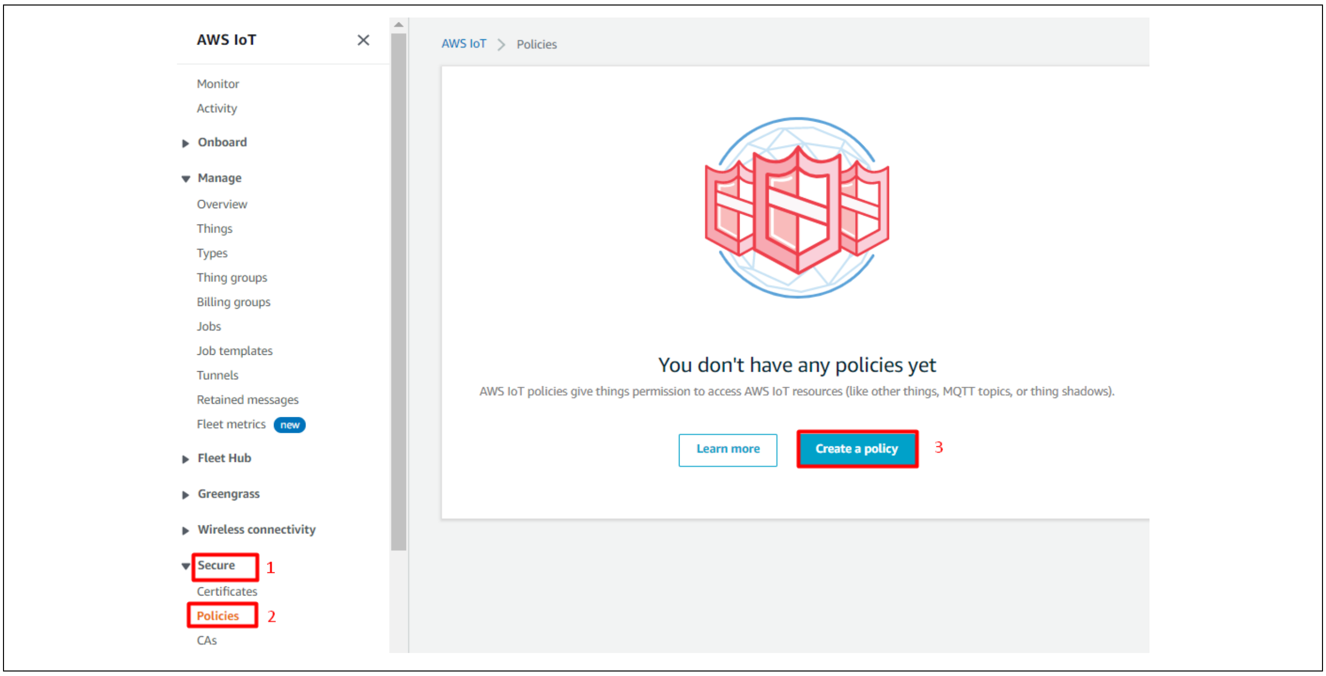
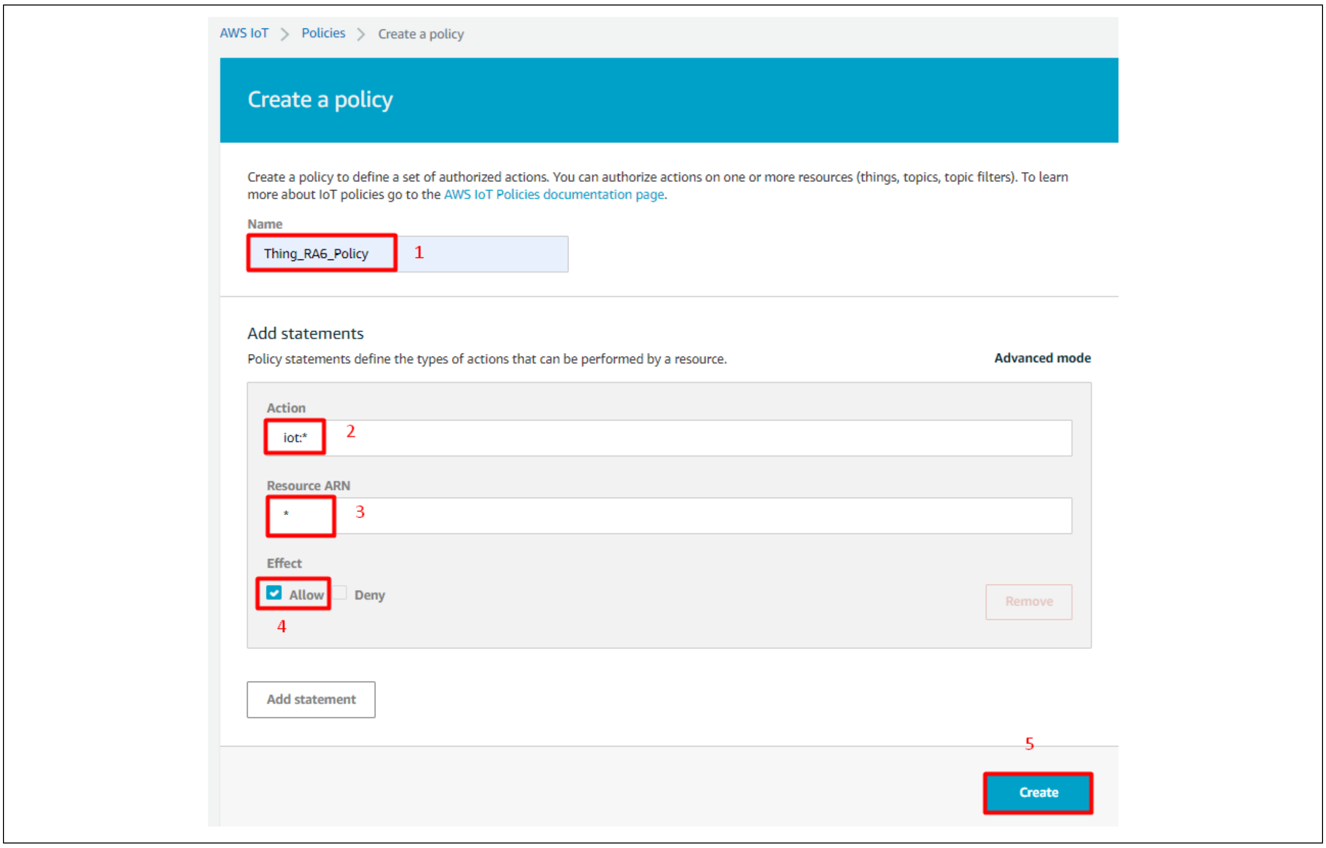


Figure 19. Create a New Policy

3. Next, input **Thing_RA6_Policy** as **Name**
4. Set **Action** for the policy to **iot:***
5. Set **Resource ARN** to *****
6. Set **Effect** to **Allow** as shown in Figure 20 and click on **Create**.



Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name: Thing_RA6_Policy

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

Action: iot:*

Resource ARN: *

Effect: ☑ Allow □ Deny

Figure 20. Create a New Policy

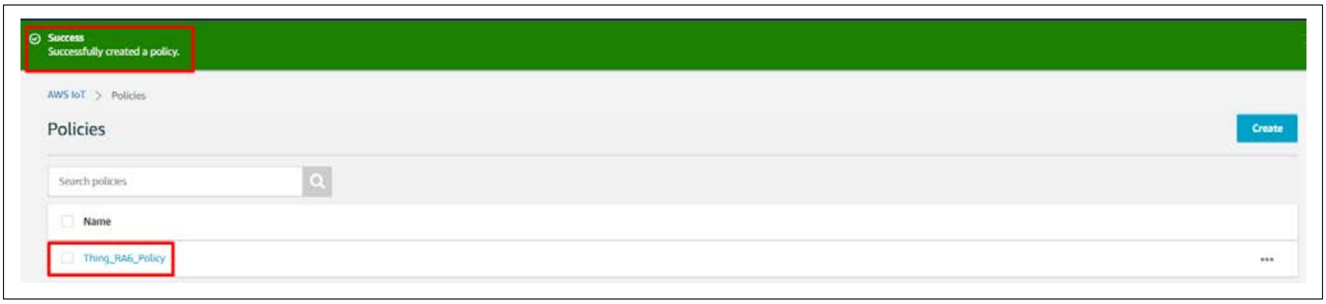


Figure 21. Created Thing New Policy

6.4.3 Generating Device Certificate and Keys

At this point, it is assumed that the AWS IoT Thing and Thing Policy has been created using the above instructions. Now we need to create device key and certificates for the AWS IoT Thing (Thing) created.

1. From the AWS IOT console, select **Secure** and then **Certificates**,
2. On the new window click on **Create a Certificate** as shown in the Figure 22.

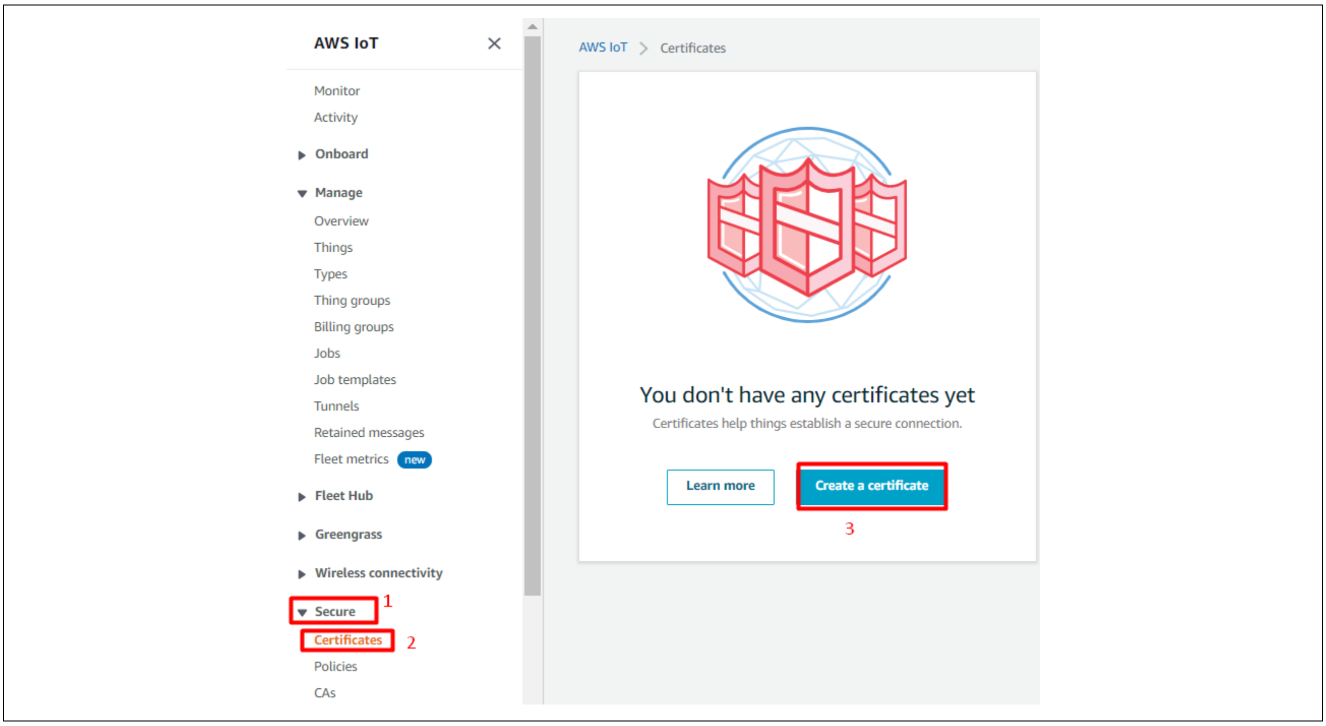
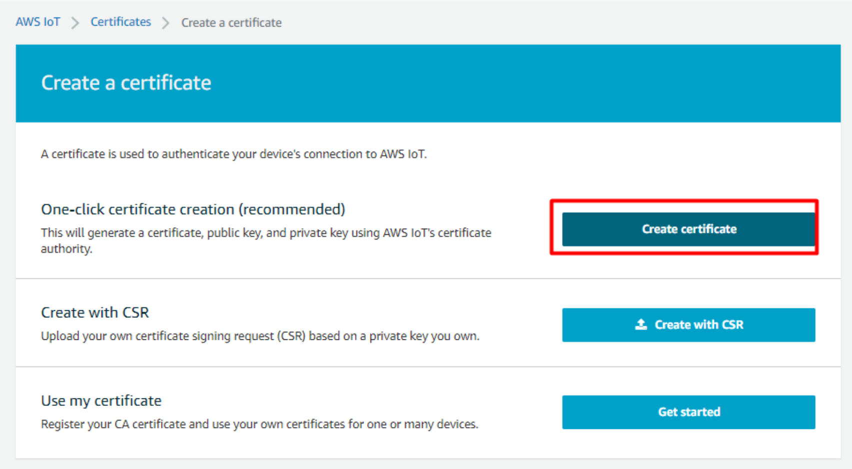


Figure 22. Creating New Certificate

3. On the new **Create a Certificate** window, click on **One-click certificate creation (recommended)** as shown in the Figure 23.



Create a certificate

A certificate is used to authenticate your device's connection to AWS IoT.

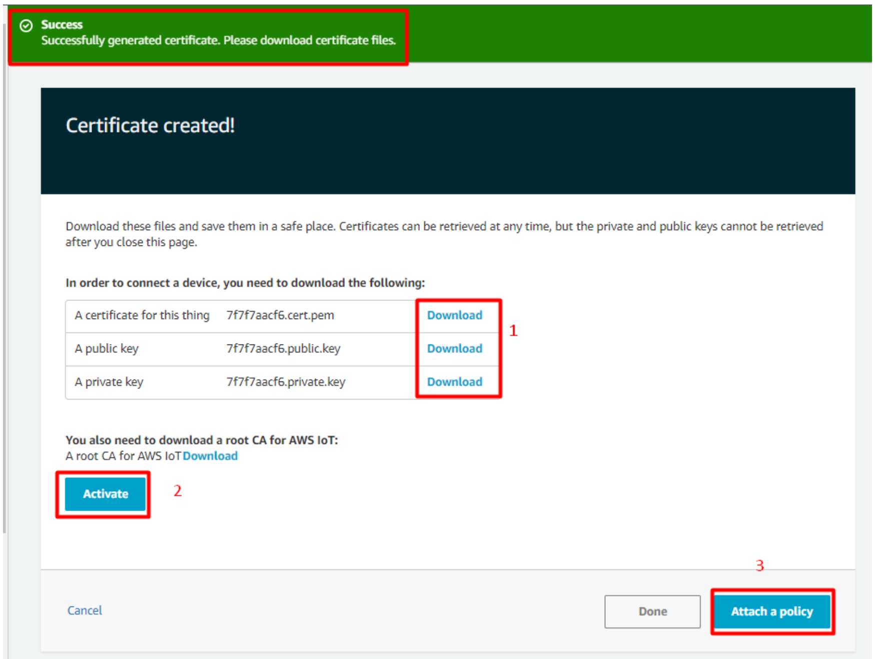
One-click certificate creation (recommended)
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.

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

Use my certificate
Register your CA certificate and use your own certificates for one or many devices.

Figure 23. Create a new Certificate

4. Newly created certificates and keys need to be downloaded. These will be used in the application project. Details of using them in the application are described in the upcoming sections. Download and save it on your computer.
5. Activate the certificate by clicking the **Activate** button.
6. Finally, click **Attach a policy** to attach the newly created policy to this certificate.



Success
Successfully generated certificate. Please download certificate files.

Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

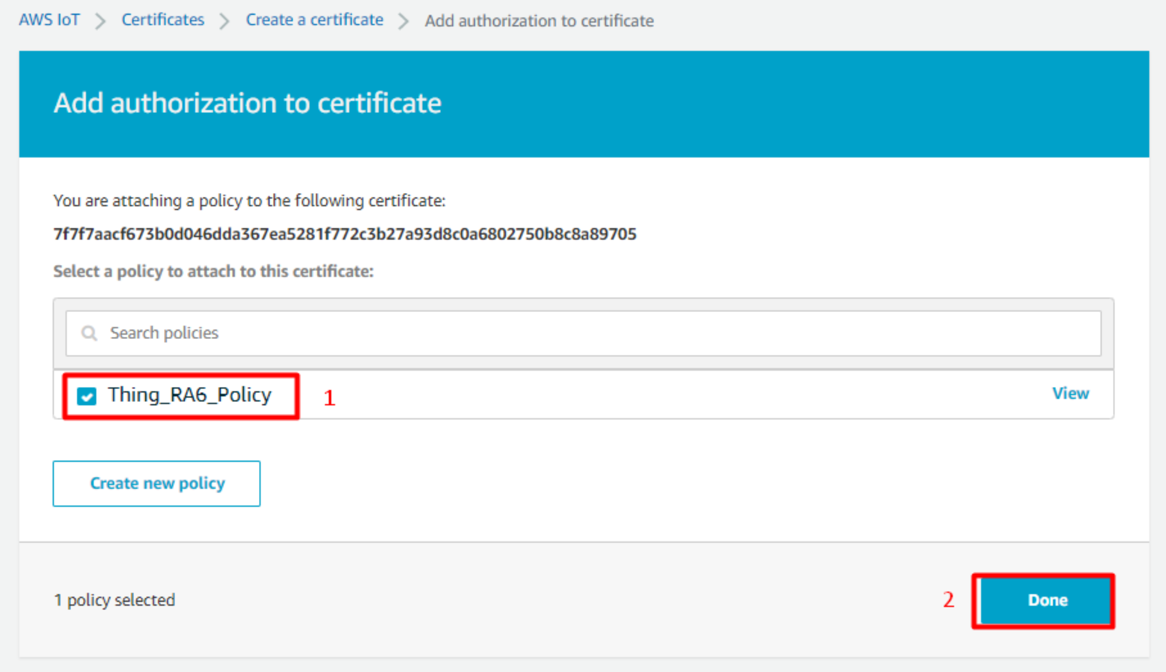
In order to connect a device, you need to download the following:

A certificate for this thing	7f7f7aacf6.cert.pem	Download
A public key	7f7f7aacf6.public.key	Download
A private key	7f7f7aacf6.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT Download

Figure 24. Device Certificate and Keys

7. In the new window, select the newly created policy **Thing_RA6_Policy** and click **Done** as shown in Figure 25.



AWS IoT > Certificates > Create a certificate > Add authorization to certificate
Add authorization to certificate

You are attaching a policy to the following certificate:

7f7f7aacf673b0d046dda367ea5281f772c3b27a93d8c0a6802750b8c8a89705

Select a policy to attach to this certificate:

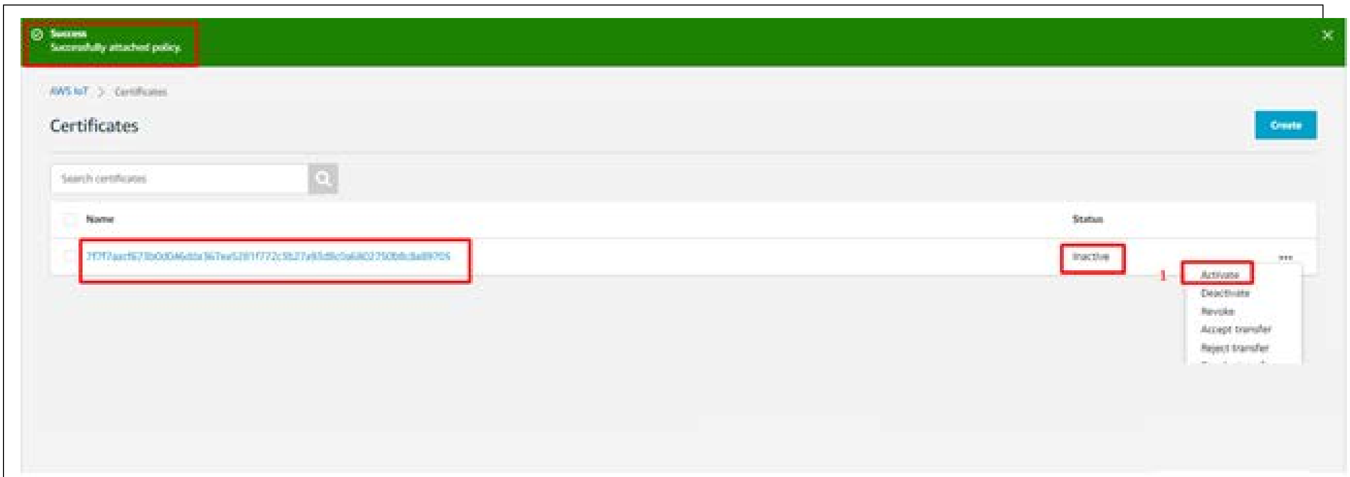
Search policies
☑ Thing_RA6_Policy
1
View
Create new policy

1 policy selected

Figure 25. Attach Policy to the Certificate

6.4.4 Activate the Certificate

1. In the opened window, you will notice the successfully attached policy.
2. The newly created certificate needs to be activated in order to use it. To activate the certificate, just click on **Activate** as shown in Figure 26.



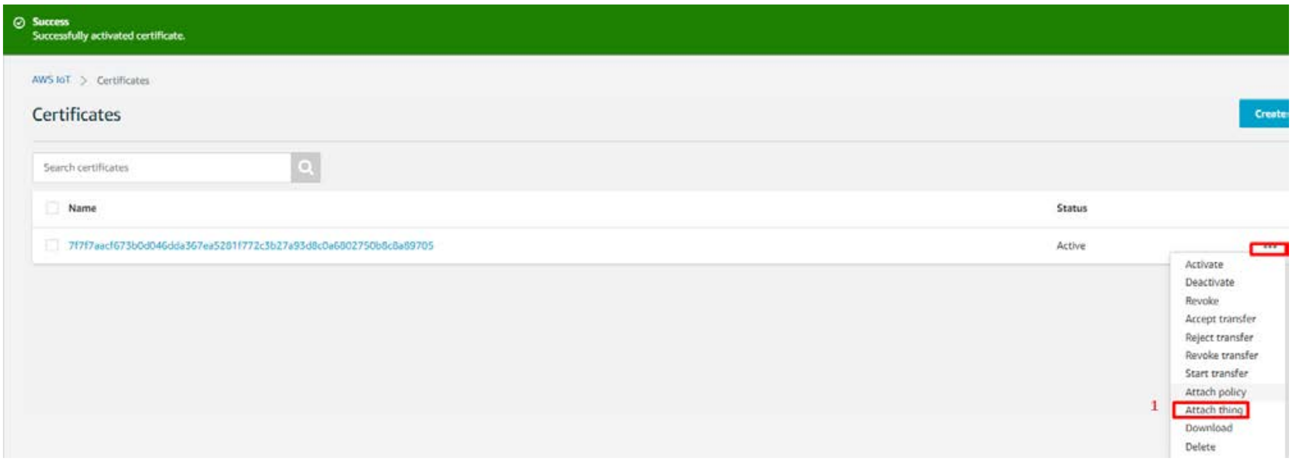
Success
Successfully attached policy.

Name	Status
7f7f7aacf673b0d046dda367ea5281f772c3b27a93d8c0a6802750b8c8a89705	Inactive

Figure 26. Activate the Certificate Created

6.4.5 Attach Thing to certificate

1. On the same dropdown menu, you can click **Attach thing**. This will attach the newly created thing ”Thing RA6” to the certificate.



Success
Successfully activated certificate.

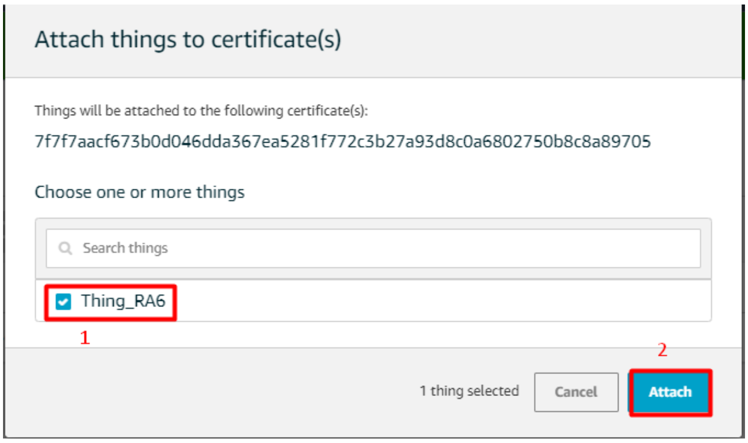
AWS IoT > Certificates

Certificates

Name	Status
7f7f7aacf673b0d046dda367ea5281f772c3b27a93d8c0a6802750b8c8a89705	Active

Figure 27. Attach Thing to Certificate

2. In the pop-up window, choose the newly created Thing **Thing_RA6** and click **Attach**.



Attach things to certificate(s)

Things will be attached to the following certificate(s):
7f7f7aacf673b0d046dda367ea5281f772c3b27a93d8c0a6802750b8c8a89705

Choose one or more things

☑ Thing_RA6

1 thing selected

Figure 28. Attach Newly Created Thing to Certificate

6.5 Running the MQTT/TLS Application Example

Note: If the project has been created with the instructions given in the previous section (6.3) using the FSP configurator, skip importing the project and directly go to Loading the Executable Binary into the Target MCU (6.5.2). However, to quickly import and evaluate the application project archived with this document, browse the sub-sections below on importing, building, and loading sections.

6.5.1 Importing, Building and Loading the Project

6.5.1.1 Importing

This project can be imported into e^2 studio using the instructions provided in the RA FSP User’s Manual. See Section *Starting Development > e2 studio ISDE User Guide > Importing an Existing Project into e2 studio ISDE.*

6.5.1.2 Building the Latest Executable Binary

Upon successfully importing and/or modifying the project into e^2 studio IDE, follow the instructions provided in the RA FSP User’s Manual to build an executable binary/hex/mot/elf file. See Section *Starting Development > e2 studio ISDE User Guide > Tutorial: Your First RA MCU Project > Build the Blinky Project.*

Note: The attached Application Project Example may produce an error connecting to the AWS cloud if test credentials are used during the run. Refer to the upcoming section 6.6, Connecting to AWS IoT, to enter the credentials for the device created per section 6.4.2, Creating a Device on AWS IoT Core.

6.5.2 Loading the Executable Binary into the Target MCU

The executable file may be programmed into the target MCU through any one of three means.

6.5.2.1 Using a Debugging Interface with e^2 studio

Instructions to program the executable binary are found in the latest RA FSP User Manual. See Section *Starting Development > e2 studio ISDE User Guide > Tutorial: Your First RA MCU Project > Debug the Blinky Project.*

This is the preferred method for programming as it allows additional debugging functionality available through the on-chip debugger.

6.5.2.2 Using J-Link Tools

SEGGER J-Link Tools such as J-Flash, J-Flash Lite, and J-Link Commander can be used program the executable binary into the target MCU. Refer to User Manuals UM08001 and UM08003 on www.segger.com.

6.5.2.3 Using Renesas Flash Programmer

The Renesas Flash Programmer provides usable and functional support for programming the on-chip flash memory of Renesas microcontrollers in each phase of development and mass production. The software supports all RA MCUs and the software user's manual is available online.

6.5.3 Powering up the Board

To connect power to the board, connect the USB cable to the EK-RA6M3 board's J10 connector (DEBUG_USB) and the other end to the PC USB port. Then run the debug application, using the following instructions.

Reset the board assembly associated with this application note to the default electrical jumper settings as specified in the board's hardware user's manual, before proceeding with the next set of instructions.

6.5.3.1 Deviation from Default Jumper Settings

The following are deviations from default board settings that should be performed prior to applying power to evaluate the application.

Table 6. Jumper Settings

Board Name	Jumper settings
EK-RA6M3	No change necessary

For this Ethernet-based cloud connectivity application project and application note, the user is required to connect the Ethernet cable to the RJ45 Ethernet connector on the board.

6.5.3.2 Power-on Behavior

Upon successful configuration and downloading of the image to the target RA MCU, the following behavior should be observed upon application of power:

1. The power LED on the RA MCU target assembly lights up.
2. The J-Link LED will be blinking based on the activity when it is connected.
3. The User LEDs (BLUE, GREEN, RED) are used to indicate the status of the application from the start of initialization to continuous status of running.

6.6 Connecting to AWS IoT

This section describes the steps to be followed to connect the device to the AWS IoT.

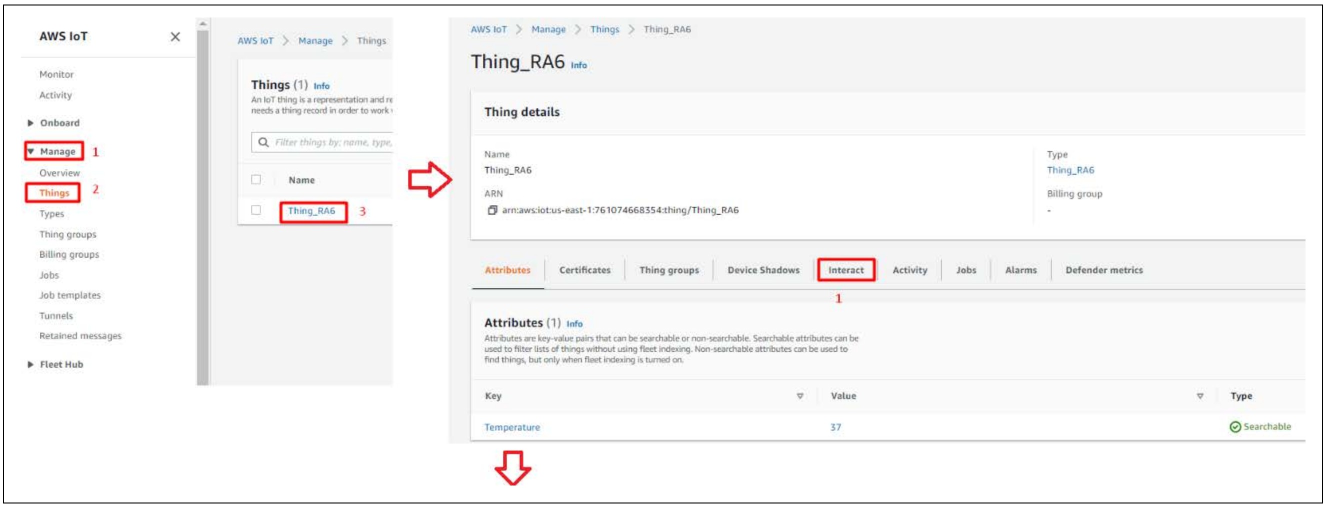
Note: Firewalls in the network may prevent connectivity to AWS IoT. Configure the network to allow access to the MQTT Port 8883.

6.6.1 AWS IoT Credentials

Default credentials for connectivity to AWS IoT are provided in the file `usr_config.h` These should be updated to use the credentials generated per the guidelines provided in section Creating a Device on AWS

IoT Core. MQTT ENDPOINT is obtained from the AWS IoT Core. The screenshots for the reference to capture the Endpoint information are as shown in Figure 29 and Figure 30.

```
  #define USR_MQTT_ENDPOINT "aoh5lvd4o23ku-ats.iot.us-east-1.amazonaws.com"
```



An IoT thing is a representation and re needs a thing record in order to work

Name: Thing_RA6

ARN: arn:aws:iot:us-east-1:761074668354:thing/Thing_RA6

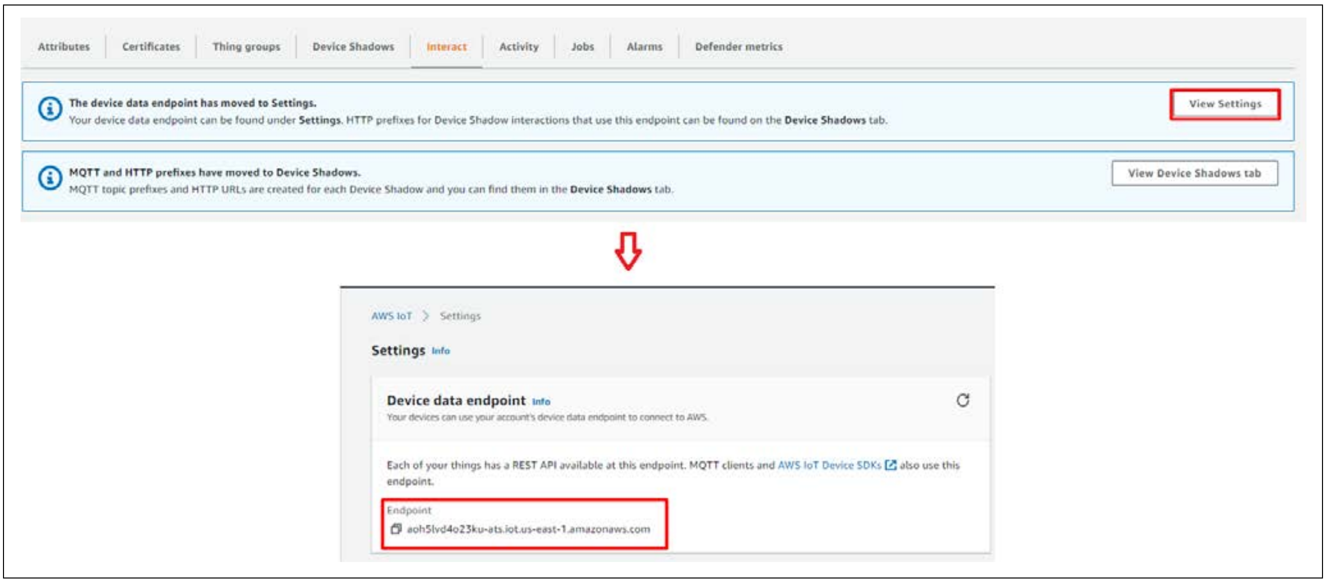
Type: Thing_RA6

Billing group: -

Attributes are key-value pairs that can be searchable or non-searchable. Searchable attributes can be used to filter lists of things without using fleet indexing. Non-searchable attributes can be used to find things, but only when fleet indexing is turned on.

Key	Value	Type
Temperature	37	Searchable

Figure 29. Getting User MQTT Endpoint



The device data endpoint has moved to Settings.
Your device data endpoint can be found under **Settings**. HTTP prefixes for Device Shadow interactions that use this endpoint can be found on the **Device Shadows** tab.

MQTT and HTTP prefixes have moved to Device Shadows.
MQTT topic prefixes and HTTP URLs are created for each Device Shadow and you can find them in the **Device Shadows** tab.

Your devices can use your account's device data endpoint to connect to AWS.

Each of your things has a REST API available at this endpoint. MQTT clients and AWS IoT Device SDKs also use this endpoint.

Endpoint: aoh5lvd4o23ku-ats.iot.us-east-1.amazonaws.com

Figure 30. User MQTT Endpoint

The downloaded asymmetric key pairs and certificates generated from AWS need to be included into the source code (`usr_config.h`) for the application example by converting each line into the format required by the AWS SDK as shown below. Refer `src/usr_config.h` from the attached project for more details.

Note: You can even use the Offline tool provided by AWS FreeRTOS for converting the Certificates and Keys PEM file to C string. The tool `certificate_configuration.zip` is attached as part of the bundle. Unzip the tool and, from the Certificate configuration tool page, provide the client certificate and private key PEM files downloaded from the AWS IoT Console. This will generate and saves the certificate and Private key in the c string format as required by the AWS SDK.

Downloaded Certificate/Key format from AWS	Desired Format for the AWS SDK
`-----BEGIN PUBLIC KEY-----`	`“-----BEGIN PUBLIC KEY-----\n”\`

```
#define CLIENT_CERTIFICATE                    “Populate the Client Certificate”
#define CLIENT_KEY                            “Populate the Client Key”
```

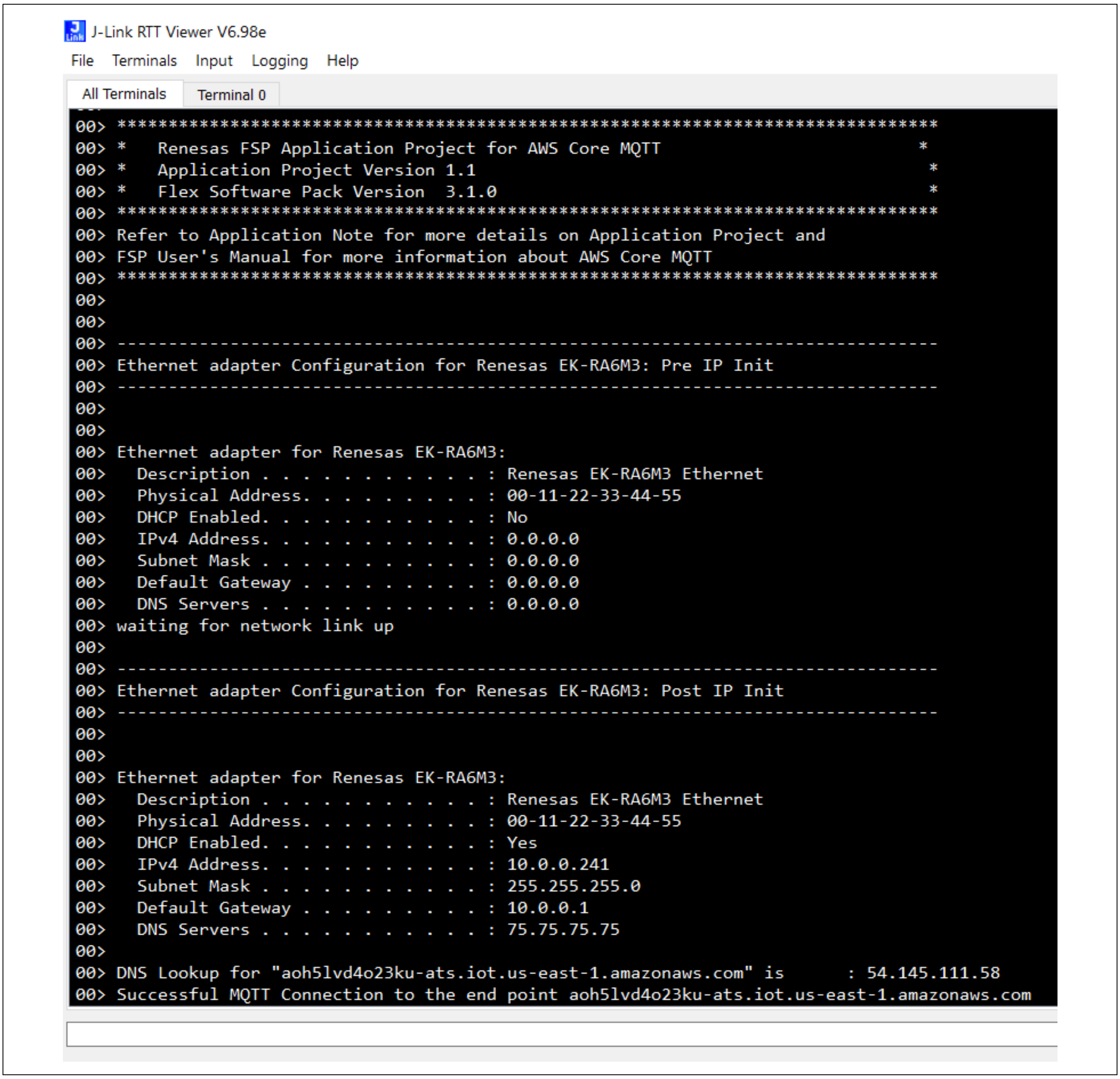
Note: From the downloaded files, client cert is the "xxxxx-certificate.pem.crt" file and the client key is the "xxxx-private.pem.key" file

Note: For TLS communication, some of the cloud service providers offer Mutual Authentication and some cloud service providers offer server-only authentication. The current application project uses Mutual Authentication to connect to the AWS cloud where both server and clients verify each other. But there are a few cloud service providers such as Adafruit and Huawei where Mutual Authentication is not used. Instead server-only authentication is used where after the handshake, the server sends the client its public key and a digitally signed certificate signed by a CA. If the client has this CA's public key, it can decrypt the certificate and establish trust with the server. When developing applications, users are required to know what kind of authentication is being used by the TLS.

6.7 Verifying the Application Project

This section describes the steps to verify this application example’s functions.

When the target RA MCU is successfully programmed with the application example binary and the board is powered up, a SEGGER J-Link RTT Console such as J-Link RTT Viewer V6.98e or later should display output similar to the output shown in Figure 31.

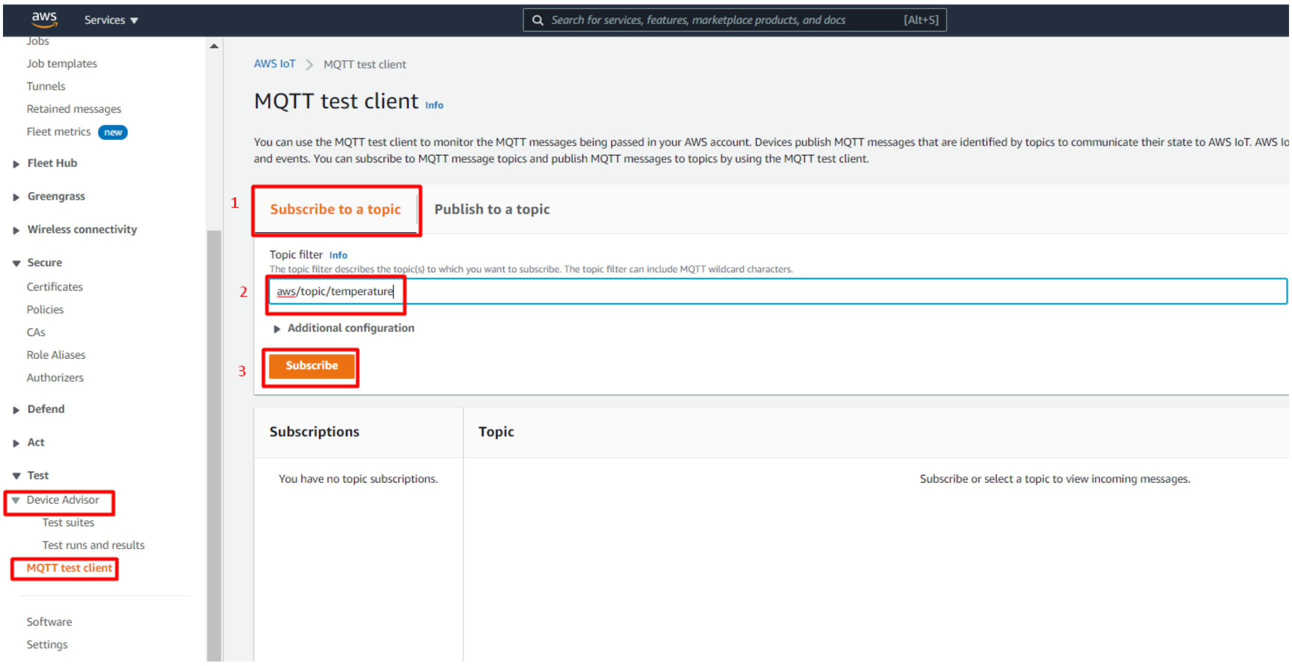


```
00> ********************************************************************************
00> *   Renesas FSP Application Project for AWS Core MQTT                         *
00> *   Application Project Version 1.1                                            *
00> *   Flex Software Pack Version  3.1.0                                          *
00> ********************************************************************************
00> Refer to Application Note for more details on Application Project and
00> FSP User's Manual for more information about AWS Core MQTT
00> ********************************************************************************
00>
00>
00> --------------------------------------------------------------------------------
00> Ethernet adapter Configuration for Renesas EK-RA6M3: Pre IP Init
00> --------------------------------------------------------------------------------
00>
00>
00> Ethernet adapter for Renesas EK-RA6M3:
00>   Description . . . . . . . . . . . : Renesas EK-RA6M3 Ethernet
00>   Physical Address. . . . . . . . . : 00-11-22-33-44-55
00>   DHCP Enabled. . . . . . . . . . . : No
00>   IPv4 Address. . . . . . . . . . . : 0.0.0.0
00>   Subnet Mask . . . . . . . . . . . : 0.0.0.0
00>   Default Gateway . . . . . . . . . : 0.0.0.0
00>   DNS Servers . . . . . . . . . . . : 0.0.0.0
00> waiting for network link up
00>
00> --------------------------------------------------------------------------------
00> Ethernet adapter Configuration for Renesas EK-RA6M3: Post IP Init
00> --------------------------------------------------------------------------------
00>
00>
00> Ethernet adapter for Renesas EK-RA6M3:
00>   Description . . . . . . . . . . . : Renesas EK-RA6M3 Ethernet
00>   Physical Address. . . . . . . . . : 00-11-22-33-44-55
00>   DHCP Enabled. . . . . . . . . . . : Yes
00>   IPv4 Address. . . . . . . . . . . : 10.0.0.241
00>   Subnet Mask . . . . . . . . . . . : 255.255.255.0
00>   Default Gateway . . . . . . . . . : 10.0.0.1
00>   DNS Servers . . . . . . . . . . . : 75.75.75.75
00>
00> DNS Lookup for "aoh5lvd4o23ku-ats.iot.us-east-1.amazonaws.com" is      : 54.145.111.58
00> Successful MQTT Connection to the end point aoh5lvd4o23ku-ats.iot.us-east-1.amazonaws.com
```

Figure 31. Welcome Screen on the Console

On the cloud side, go to IoT Core and select **Test**, then choose **MQTT test client**. Subscribe to a topic (In this application **aws/topic/temperature** and **aws/topic/switch_status** are the topics for subscription). You may observe events on the dashboard for the subscribed topics as shown below.

Note: Temperature messages are synchronous (every 30 seconds). Switch statuses are asynchronous. User can press S1 or S2 button to verify the switch status on the Dashboard.



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

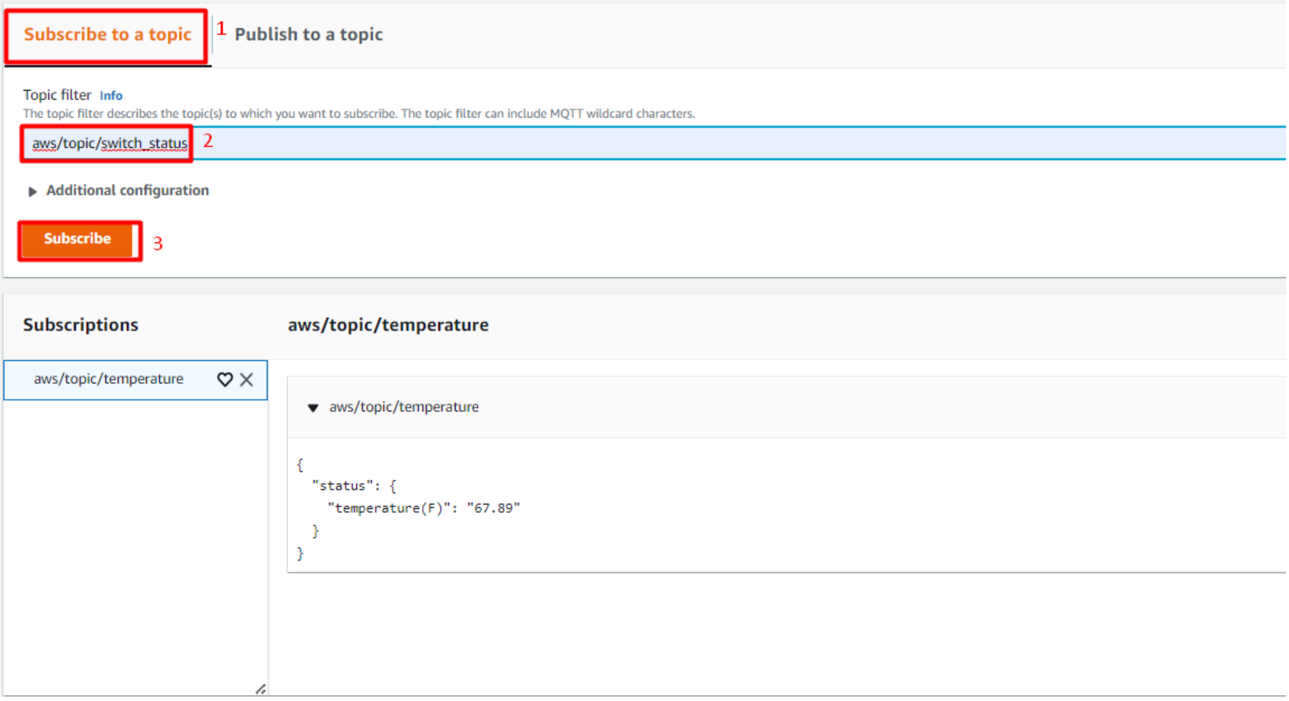
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

Topic filter: aws/topic/temperature

You have no topic subscriptions.

Subscribe or select a topic to view incoming messages.

Figure 32. Subscribe to a Temperature Topic Messages on the AWS IoT Screen

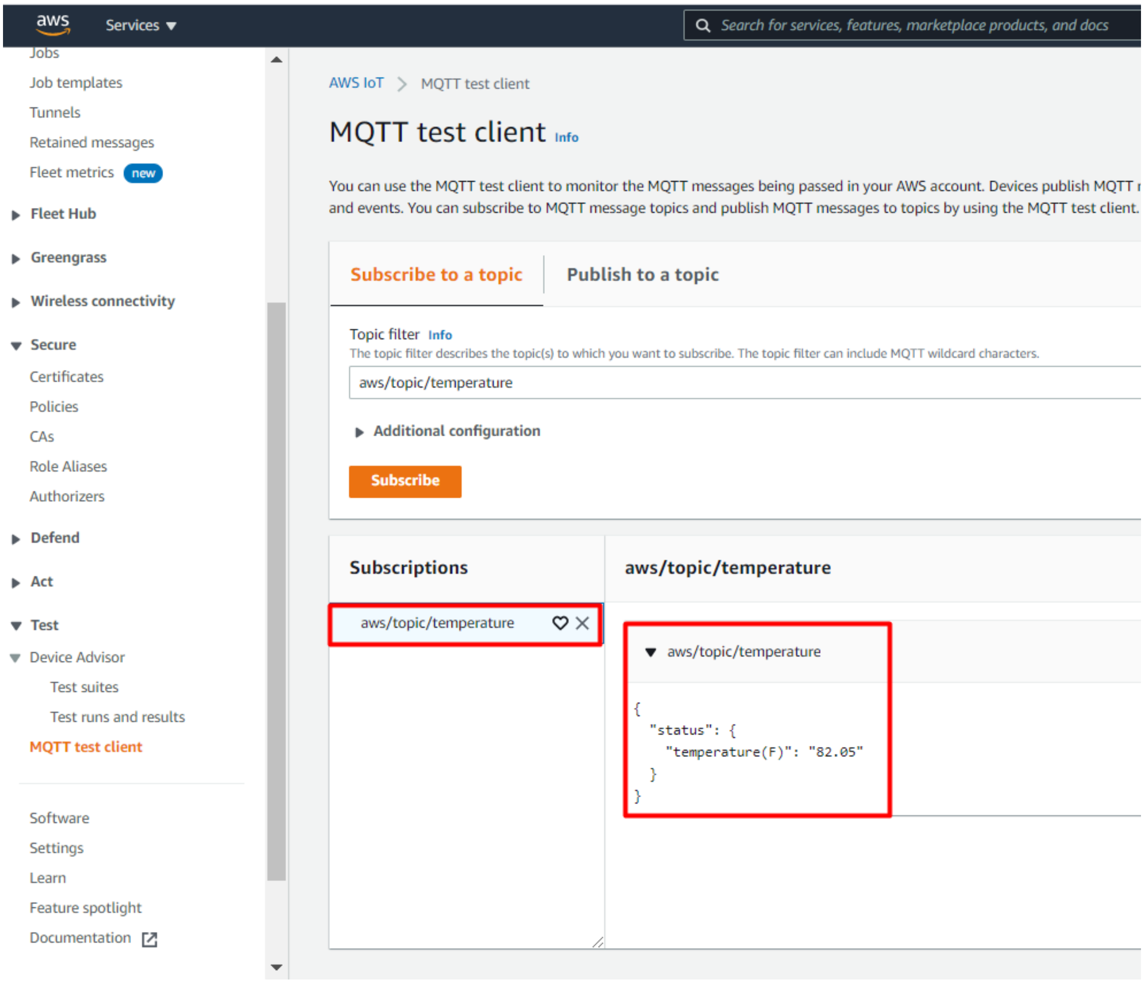


The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

Topic filter: aws/topic/switch_status

```
{
  "status": {
    "temperature(F)": "67.89"
  }
}
```

Figure 33. Subscribe to a Switch status Topic Messages on the AWS IoT Screen



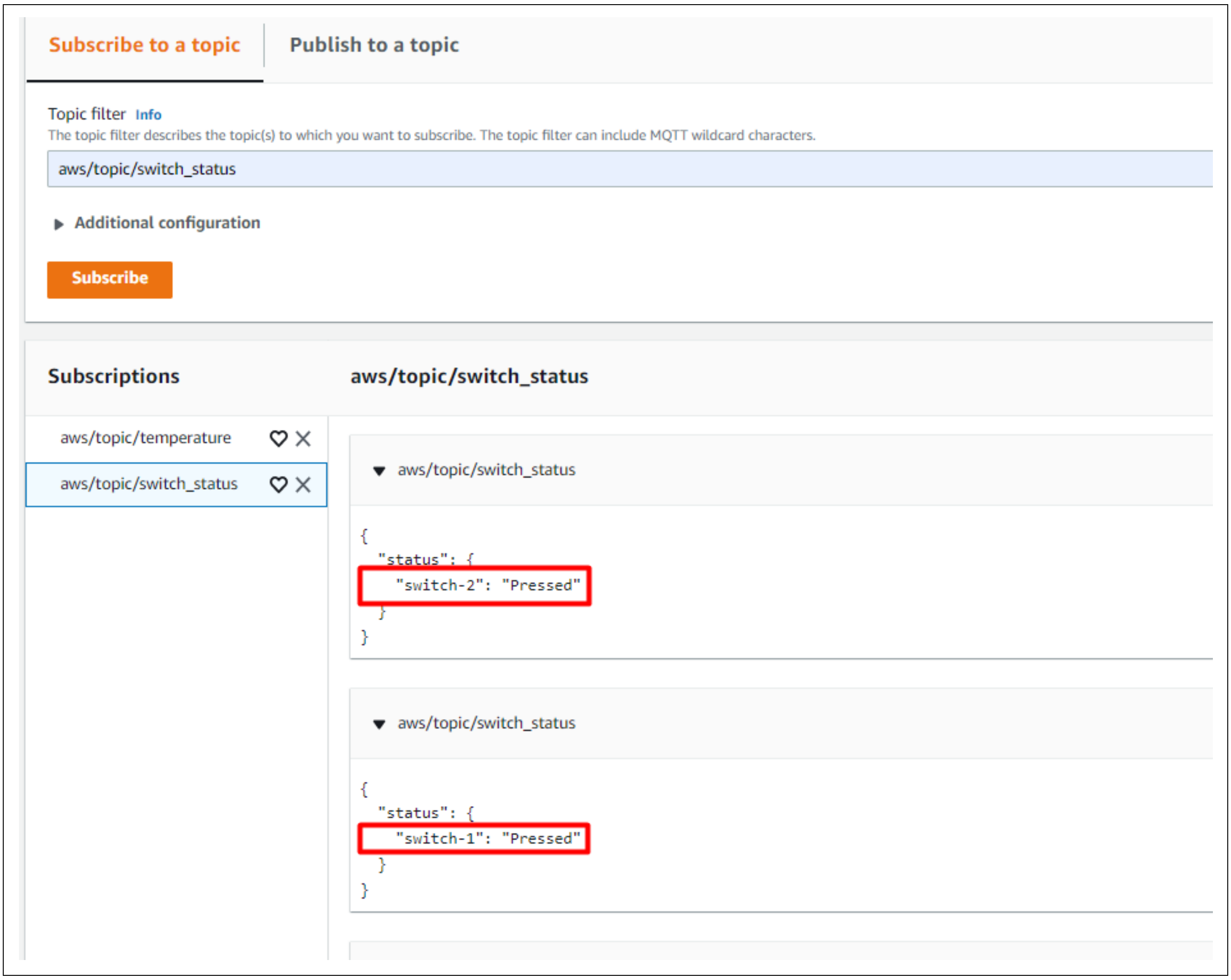
You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devices publish MQTT r and events. You can subscribe to MQTT message topics and publish MQTT messages to topics by using the MQTT test client.

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

Topic filter: aws/topic/temperature

```
{
  "status": {
    "temperature(F)": "82.05"
  }
}
```

Figure 34. Subscribed Messages on the AWS IoT Screen - Temperature



```
{
  "status": {
    "switch-2": "Pressed"
  }
}
```

```
{
  "status": {
    "switch-1": "Pressed"
  }
}
```

Figure 35. Subscribed Messages on the AWS IoT Screen - Switch Status

To publish the message from the cloud to the device to turn the LEDs ON/OFF, select **Publish to a topic** and type the specific topic (**aws/topic/led**) and message in the respective window and click **Publish to a topic** as shown below. Follow the steps 1-4 as shown in Figure 36.

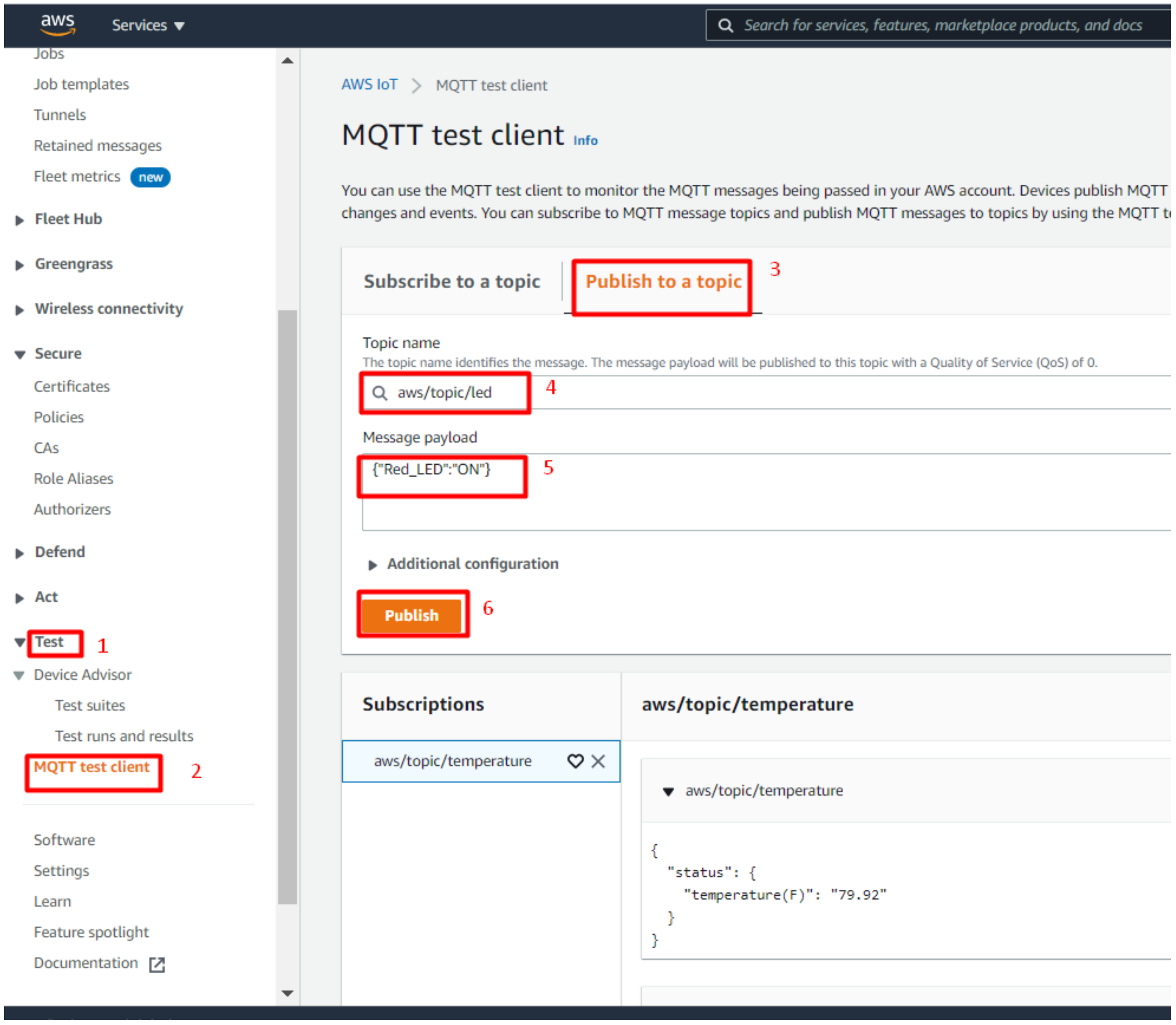
The actuation of the requested LED can be seen on the demo board.

Note: The messages under the Message column are **case sensitive. U**sers 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. The parser on the device is sensitive to message JSON format. Do not include any extra spaces in the JSON message. If the message alone is seen on the console without the LED actuation, please check for the spaces introduced in the message.

Table 7. Messages for Toggling User LED

LED State	Message
RED LED ON	{"Red_LED":"ON"}
RED LED OFF	{"Red_LED":"OFF"}



MQTT test client

You can use the MQTT test client to monitor the MQTT messages being passed in your AWS account. Devices publish MQTT changes and events. You can subscribe to MQTT message topics and publish MQTT messages to topics by using the MQTT t

Topic name

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

aws/topic/led

Message payload

{"Red_LED":"ON"}

aws/topic/temperature

▾ aws/topic/temperature

```
{
  "status": {
    "temperature(F)": "79.92"
  }
}
```

Figure 36. Publishing the Messages from the AWS IoT Screen

Note: The LEDS are also used for other user indications in this Application. So, you may see that the LED status changes after you set the LED status from the MQTT GUI **Publish to a topic** due to the following events.

- Blinking red LED indicates that MQTT message failure activity is seen. This could be Message Publish or Subscribe failure.
- Solid red LED indicates the initialization failed. The console log will have the appropriate log for the error (if this is not turned on by the user as part of the publish message).
- Green LED blinks periodically to indicate the heartbeat of the system and healthy network connectivity.
- Blue LED toggles based on the MQTT activity (default: temperature data every 30 seconds). Here the blue LED activity indicates that successful MQTT messages are being exchanged.

7. MQTT/TLS Module Next Steps

- For setting up a client using a device certificate signed by a preferred CA certificate, refer to the link: https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html
- For 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

8. Bibliography

[1] International Telecommunication Union, "ITU-T Y.4000/Y.2060 (06/2012)," 15 06 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559.

[2] Amazon Web Services, "AWS IoT Core Features," [Online]. Available: https://www.amazonaws.cn/en/iot-core/features/.

[3] Amazon Web Services, "AWS IoT Core," [Online]. Available: https://www.amazonaws.cn/en/iot-core/.

[4] W. T. L. L. O. S. R. N. S. R. X. G. K. N. K. S. F. M. K. D. L. I. R. Valerie Lampkin, Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, IBM Redbooks, 2012.

[5] I. E. T. Force, "The Transport Layer Security (TLS) Protocol Version 1.2," [Online]. Available: https://tools.ietf.org/html/rfc5246.

[6] Amazon Web Services, "AWS IoT Security," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-security.html.

[7] Amazon Web Services, "Transport Security in AWS IoT," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html.

[8] International Telecommunication Union, "X.509 (10/19) Summary," 10 2019. [Online]. Available: https://www.itu.int/dms_pubrec/itu-t/rec/x/T-REC-X.509-201910-I!!SUM-HTM-E.htm.

[9] Eclipse Foundation, "Eclipse Mosquitto™ - An open source MQTT broker," [Online]. Available: https://mosquitto.org/.

[10] Amazon Web Services, "AWS IoT Device SDK C: MQTT," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/index.html.

[11] R. Barry, "Mastering the FreeRTOS™ Real Time Kernel," in *A Hands-On Tutorial Guide*, 2016.

[12] A. I. D. S. C. Documentation, "AWS IoT Device SDK C: MQTT Functions," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/mqtt_functions.html.

[13] Amazon, "Configuring the FreeRTOS Demos," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/userguide/freertos-configure.html.

[14] "Amazon FreeRTOS mbedTLS," [Online]. Available: https://github.com/aws/amazon-freertos/blob/master/libraries/3rdparty/mbedtls/utils/mbedtls_utils.c.

[15] Renesas Electronics Corporation, "Renesas Flash Programmer (Programming GUI) - Documentation," [Online]. Available: https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html#documents.

[16] Silex Technology, Inc., "SX-ULPGN-EVK - Evaluation kit for Ultra-Low-Power Hostless Wi-Fi IoT Module," [Online]. Available: https://www.silextechnology.com/connectivity-solutions/embedded-wireless/sx-ulpgn-evk.

9. Known Issues

Details about the known FSP and tool related issues can be found at this link: https://github.com/renesas/fsp/issues

Website and Support

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

RA Product Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug.17.21	—	First release document
1.01	Sep.17.21	—	Updated new AWS snapshots
1.02	Mar.21.22	---	Updated the AWS Wrapper module Specific configs and review comments

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

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/.