RX Family

Provisioning Procedure for IoT Devices

Introduction

IoT device provisioning is required in order to connect to AWS IoT, a cloud service provided as part of Amazon Web Services™ (AWS). As used here, the term “provisioning” refers to the process of generating, utilizing, and managing authentication information such as things, private keys, and device certificates. Provisioning requires consideration of matters such as how to write authentication information to products as part of the manufacturing process (initial installation) and how to manage (protect) and update key data. These types of data are stored in the on-chip flash memory of RX Family MCUs. Since it is extremely difficult to modify the provisioning method for IoT devices afterward, the above-mentioned consideration must begin at the product development stage so that verification can be completed by the mass production phase.

Of the various provisioning methods provided by AWS, this document describes the “fleet provisioning method,” which automates provisioning during the manufacturing process and when the device is initially used.

Deploying the fleet provisioning method eliminates the need to devote time and effort to cumbersome provisioning procedures while making the provisioning process more secure and convenient.

What you will learn in this application note

- ✓ Overview of provisioning methods provided by AWS.
- ✓ How to realize fleet provisioning using demo and confirm operation. The steps to run the demo will be explained from "4 Running the Fleet Provisioning Demo".

The contents of this document are sufficient to implement provisioning, but if followed unmodified will result in important data saved as part of the provisioning processing, such as the private key and device certificate, being stored as “clear text” (unencrypted text) in the on-chip flash memory of the RX Family MCU. This means that if there is a security hole in a user program programmed to the RX Family MCU that allows arbitrary areas of memory to be read, the provisioning data in the flash memory could be accessed, possibly allowing an attacker to perform an unauthorized login to the user’s AWS account.

Using the Trusted Secure IP (TSIP) module of the RX Family MCU enables the private key and device certificate to be stored in encrypted form, greatly reducing the danger of unauthorized access to the provisioning data. For details of the TSIP module, see the page linked to below.

https://www.renesas.com/software-tool/trusted-secure-ip-driver

We strongly encourage using the TSIP module to boost security.

It is possible to reduce the risk of unauthorized access to provisioning data by improving software quality, but this approach can never completely eliminate it. In particular, if there are defects in the software of IoT devices, which are vulnerable to threats posed by attackers, it is recommended that firmware update functionality be used to apply corrections in a timely manner. For more information on firmware updates, please refer to the application note Renesas MCU Firmware Update Design Policy (R01AN5548).

Operating Environment

The operation described in this application note has been confirmed on the following environment.

Integrated development environment	e^2 studio 2023-10
Board	CK-RX65N
Toolchain	CC-RX Compiler v3.05.00
Emulator	E2OB (E2 Lite On Board) module of CK-RX65N

Before applying the contents of this application note to another MCU, a review of product-specific settings matching the specifications of the MCU should be made and adequate evaluation performed.

Related Application Notes

Information on documents related to this application note is provided below. Refer to these documents as necessary.

- Renesas MCU Firmware Update Design Policy (R01AN5548)
- RX Family How to implement FreeRTOS OTA by using Amazon Web Services on RX65N (R01AN5549)
- Firmware Integration Technology User's Manual (R01AN1833)
- RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)

Information about boards, related programs, and development tools needed to develop RX cloud solutions is summarized on the page linked to below.

https://www.renesas.com/rx-cloud

Also, the following information publicly released by AWS may be of use. (The first two items are only available in Japanese.)

- Provisioning authentication information to devices in AWS IoT
  Video: https://youtu.be/gcJwNEQ2eLY
  Document: https://pages.awscloud.com/rs/112-TZM-766/images/EV_iot-deepdive-aws2_Sep-2020.pdf
- Document on fleet provisioning templates
  https://docs.aws.amazon.com/iot/latest/developerguide/provision-template.html
- Document on AWS IoT Core policies
  https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html
- AWS IoT API reference document: CreateCertificateFromCsr
  https://docs.aws.amazon.com/iot/latest/apireference/API_CreateCertificateFromCsr.html
- Provisioning devices that don't have device certificates using fleet provisioning
  https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html
- How to automate onboarding of IoT devices to AWS IoT Core at scale with Fleet Provisioning
  https://aws.amazon.com/blogs/iot/how-to-automate-onboarding-of-iot-devices-to-aws-iot-core-at-scale-with-fleet-provisioning/

Contents

1. Terminology

The following terms are used in this document.

Table 1.1 List of Terms

Term	Meaning
AWS	A suite of cloud computing services provided by Amazon Web Services, Inc.
FreeRTOS	An open-source real-time operating system for embedded systems.
Provisioning	Device provisioning. Certification of a device to enable communication with AWS IoT Core.
Fleet provisioning	Functionality that implements automated provisioning of IoT devices when they are turned on for the first time.

2. Device Provisioning

IoT device provisioning refers to the process of generating a unique ID (such as an X.509 certificate or private key) for a device, registering the unique ID with an AWS IoT endpoint, and linking the necessary access privileges (IoT policies, etc.) to enable the device to connect securely to AWS IoT and other cloud-based applications. (See Figure 2.2)

Device provisioning on AWS IoT makes use of AWS IoT Core functionality such as just-in-time-registration (JITR) and just-in-time-provisioning (JITP) to automate the process of registering the identity of each device in the AWS cloud and linking it with the necessary permissions, making it easy the perform provisioning for multiple devices. However, the process of securely generating a unique ID and writing it to each device is the responsibility of the user, and for OEM vendors manufacturing large numbers of devices, this process can involve manual operations and be quite time consuming.

Fleet provisioning, which is described in this document, is one way to deal with this issue.

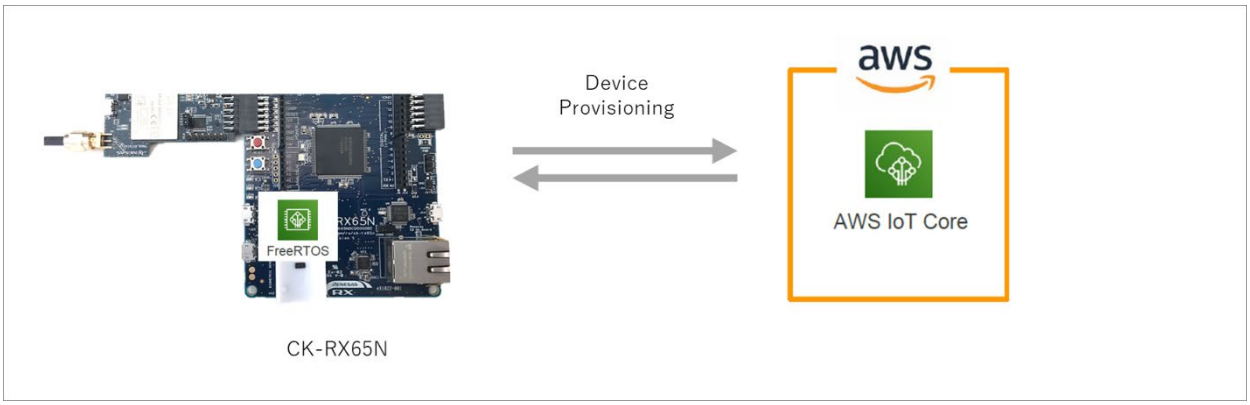


Figure 2.1 Device Provisioning

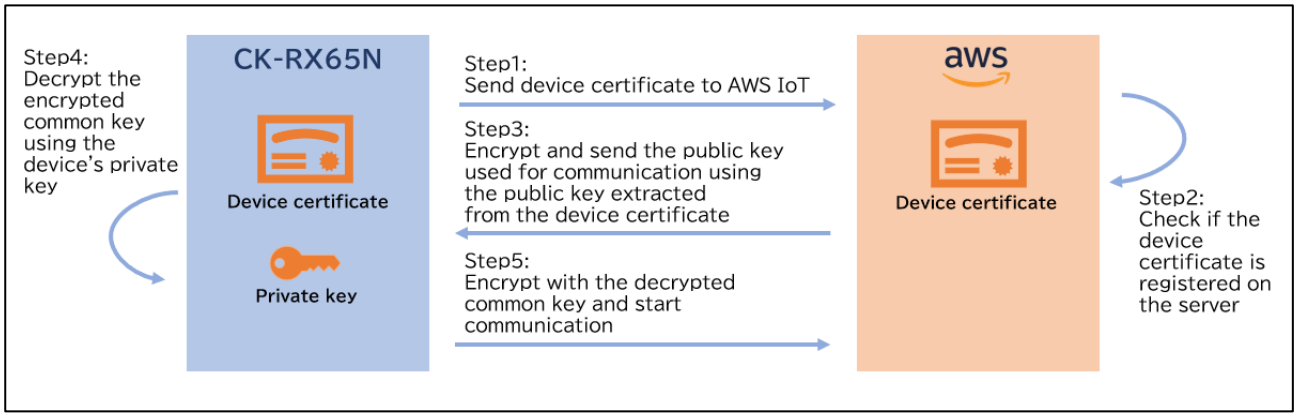


Figure 2.2 IoT Device Provisioning

2.1 Provisioning Methods of AWS IoT

AWS IoT allows the user to select from the provisioning methods listed below.

AWS allows the user to select the device provisioning method that best matches their application. Multiple provisioning methods are available to accommodate market demand and a variety of use cases. The following document describes now the various provisioning methods work as well as their advantages and disadvantages in order to assist users in making a selection. We recommend referencing this document when considering the different provisioning methods.

https://pages.awscloud.com/rs/112-TZM-766/images/EV_iot-deepdive-aws2_Sep-2020.pdf#page=115

[Provisioning Methods of AWS IoT]

1. Private key and certificate issuance and pre-registration by AWS IoT (registration at time of device kitting)
2. Certificate issuance and pre-registration by AWS IoT (registration at time of device kitting)
3. Fleet provisioning registration (Described in this document.)
4. Certificate issuance by your own certification authority and pre-registration on AWS IoT
5. Certificate issuance by your own certification authority and registration by JITR
6. Certificate issuance by your own certification authority and registration by JITP
7. Registration of a certificate from an unregistered certification authority (multi-account registration)

When confirming the operation of FreeRTOS at the preliminary stages when considering mass production, the simplest approach is “private key and certificate issuance and pre-registration by AWS IoT.” In this case a private key certificate is issued and the source code is converted on AWS, and the resulting source code is embedded in the source code of FreeRTOS. However, it is difficult to embed individual certificates during manufacturing using this method. For this reason, this document focuses on fleet provisioning, which does not require use of a certification authority and imposes the lightest workload during mass production.

Note: A part of RX Family MCUs incorporate a Trusted Secure IP (TSIP) module. When the TSIP is used, an on-chip random number generator is used to generate an RSA or elliptic curve cryptosystem key pair, and the public key is extracted and sent to a user-specified certification authority, which appends and returns a certificate. This enables implementation of JITR or JITP. This method provides strong security while reducing the implementation cost, and it should be considered for practical use moving forward.

2.2 Fleet Provisioning Method

Fleet provisioning is a procedure in which provisioning takes place when each IoT device is started for the first time. Generally speaking, it can be implemented in either of the following two ways.

1. Provisioning by claim (approach using provisioning claim certificates)
2. Provisioning by trusted user (mobile or web app user, etc.)

In addition, either of the following two procedures can be used to obtain the individual certificates and private keys used for fleet provisioning.

A) Having the AWS certification authority generate a new individual certificate and private key and send it to the device (CreateKeysAndCertificate).
B) Generating a key pair on the device internally and sending a certificate signature request (CSR) to AWS to have them generate only an individual certificate and send it to the device (CreateCertificateFromCsr).

This document describes the implementation of a fleet provisioning demo that combines 1. and B). (See Figure 2.6.) The provisioning method presented in this document provides the following advantages.

Advantages:

- The device's private key never leaves the device.
- There is no need to establish a connection between the manufacturing plant and AWS IoT.
- There is no need to put in place a structure for issuing individual certificates or registering devices.

On the other hand, it also has the following disadvantages. It is necessary to be aware of both the advantages and the disadvantages when using this provisioning method.

Disadvantages:

- It is necessary to take into account the possibility that the provisioning claim certificate could leak to an unauthorized party.
- It is necessary to implement functionality on the device to issue a provisioning request and receive a response.

2.3 Provisioning by Claim (Approach Using Provisioning Claim Certificates)

Each device can be manufactured with an embedded provisioning claim certificate and private key. If these credentials have been registered with AWS IoT, AWS IoT can exchange them for a unique device certificate that can then be used in the normal operation of the device. This process consists of the steps listed below.

The design of provisioning by claim assumes a scenario in which all the devices are manufactured using a common provisioning claim certificate. The provisioning claim certificate only allows each device to do the following.

1. Establish an initial connection to AWS IoT Core.
2. Verify identity.
3. Use data communication as described below to request an ID to which the necessary permissions have been assigned.

The provisioning claim certificate common to all the devices is written to each device, along with the initial software, at a site such as the manufacturing plant. If the device already contains an individual private key, it can send a provisioning claim certificate to be signed by AWS IoT Core and a certificate signature request (CSR). (See Figure 2.6.)

In addition to the provisioning claim certificate presented by each device, fleet provisioning can make use of Lambda-based provisioning hooks to verify the attributes of devices. Examples of device attributes include serial number, MAC ID, and device location. We recommend that you consider making use of Lambda functions in provisioning transactions as a way to automate acceptance or rejection of the provisioning status of individual devices based on the custom attributes sent during this process.

(The demo project described in this application note does not make use of Lambda functions.)

Refer to the page linked to below for information on using AWS Lambda for provisioning.

https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html

“Using pre-provisioning hooks with the AWS CLI”

2.3.1 Overview of Provisioning by Claim (Using Provisioning Claim Certificate)

When the device is powered on and capable of establishing a network connection, one of the following workflows is executed.

Figure 2.5 and Figure 2.6 show the workflows for the CreateKeysAndCertificate method and CreateCertificateFromCsr method, respectively.

Also, you can confirm the details of the AWS IoT Fleet Provisioning Demo workflow (CreateCertificateFromCsr method), on which the fleet provisioning demo described in this document is based, by visiting the page linked to below.

https://aws.github.io/aws-iot-device-sdk-embedded-C/latest/docs/doxygen/output/html/fleet_provisioning_demo.html

1. Using the claim certificate written to the device beforehand, the device connects to AWS IoT Core via a secure TLS 1.2 connection. If the device contains a CSR, this is presented along with the provisioning claim certificate.
2. The certificate is linked to an extremely restrictive policy that only provides access to IoT topics linked to the fleet provisioning process.
3. The fleet provisioning service returns a token providing “proof of ownership” to securely isolate the transaction and a valid certificate and private key payload. The token will be called later to activate the certificate. If a CSR was presented, it is used to generate the certificate.
4. The device sends a MQTT request to AWS IoT Core and presents the ownership token, the name of the fleet provisioning template created by the account owner, and (optionally) device attributes for provisioning validation. It is recommended that Lambda-based provisioning hooks be used to enable additional validation, such as checking the device’s serial number or MAC ID against a pre-approved list.
5. The fleet provisioning template is acted upon, the provisioning transaction takes place, and the results are returned. Typically, these results may include verification by Lambda function of device attributes, certificate activation, production policy attachment, and thing or group creation (optional).
6. Based on the results of the provisioning transaction, the status of the new certificate is returned. If the transaction was successful, the provisioning claim certificate is deprecated or rotated for the “production” certificate. If the transaction is denied, an “access denied” error is returned to the device.

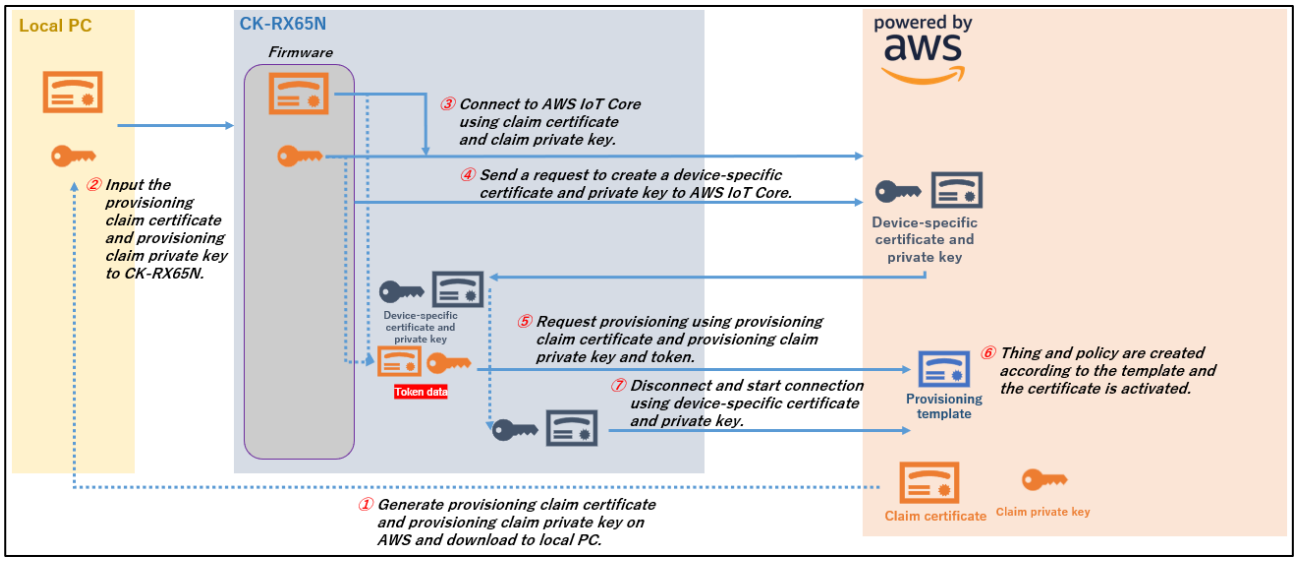


Figure 2.3 Workflow of Provisioning by Claim Using CreateKeysAndCertificate Method

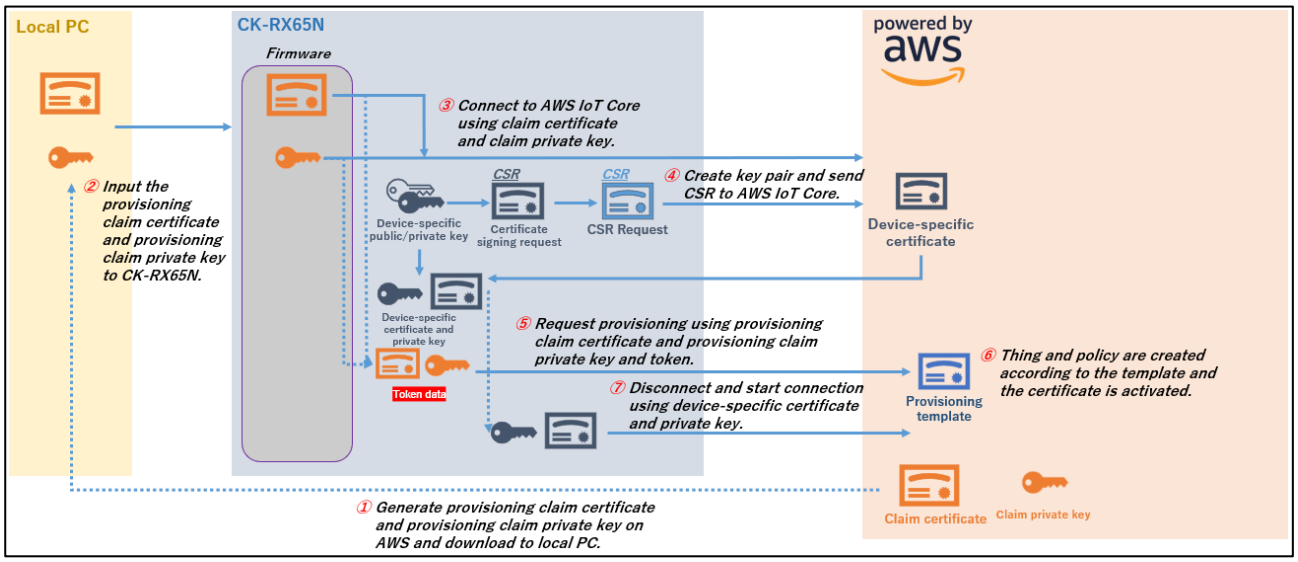


Figure 2.4 Workflow of Provisioning by Claim Using CreateCertificateFromCsr Method

2.3.2 Determining a Unique Thing Name

When sending a request to MQTT during fleet provisioning, the device's serial number can be included in the payload to ensure that each device has a unique thing name that does not duplicate an existing one.

Generally speaking, one of the following two methods is used to determine the serial number.

1. A random value generated by a random number generator or an ID value unique to the device is used as the serial number.
2. A Lambda-based provisioning hook and Amazon S3 or a user-specified database are used to change a temporarily assigned serial number to a unique serial number.

The example described in this document makes use of method 1. The unique ID assigned to each RX Family MCU is used to prevent duplication of thing names.

Figure 2.5 Using a Random Value or Unique ID to Determine the Thing Name

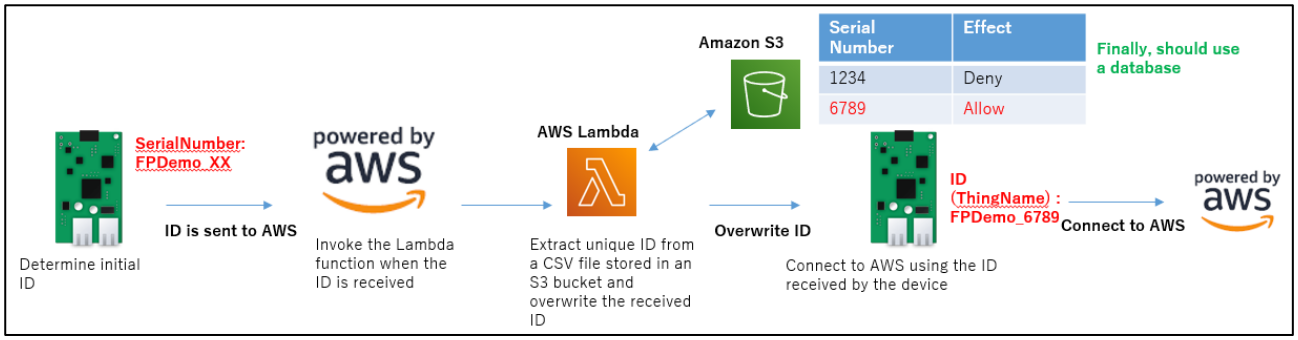


Figure 2.6 Using Amazon S3 or a Database to Determine the Thing Name

3. Preparation

This section and those that follow describe the sequence of steps from importing the project accompanying this application note to running the fleet provisioning demo on the CK-RX65N board.

3.1 Hardware Environment

The components of the hardware environment for the demo project are listed below.

Table 3.1 Hardware Components

Item	Product Name	Provider	Description
Board	CK-RX65N	Renesas Electronics Corporation	RX65N Cloud Kit
PC	PC running Windows 10 (recommended)	—	Host PC for demo

3.2 Software Environment

The components of the software environment for the demo project are listed below.

Table 3.2 Components

Item	Product Name	Version	Description
Integrated development environment	e^2 studio	2023-10	—
Toolchain	CC-RX	v3.05.00	—
Communication software	Tera Term	Ver 4.106	For displaying logs
Emulator	E2OB (E2 Lite On Board) module of CK-RX65N	—	—

3.3 Tera Term Settings

The demo uses Tera Term to display log output. Configure the settings in Tera Term as shown below.

Table 3.3 Tera Term Settings

Item	Setting
Baud rate	115,200
Data length	8 bits
Parity	None
Stop bits	1 bit
Flow control	None

3.4 FreeRTOS Project

Figure 3.1 shows the software components of the demo project.

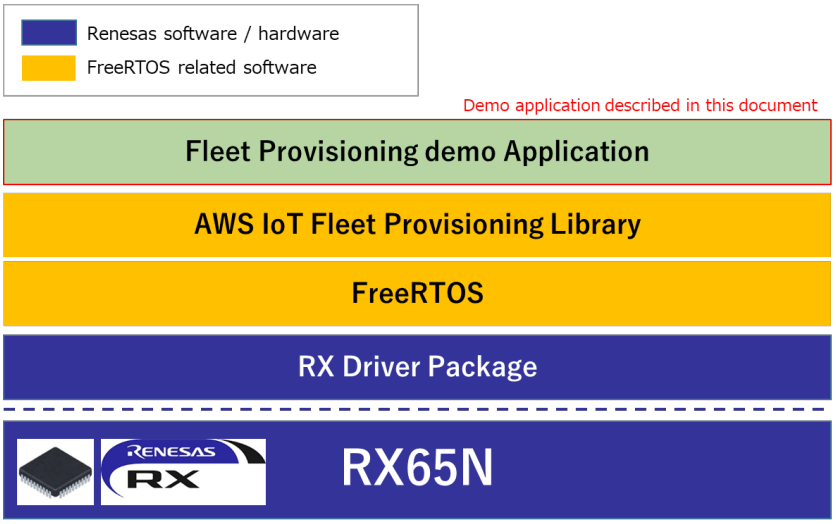


Figure 3.1 Components of Demo Project Accompanying This Application Note

The AWS IoT Fleet Provisioning Library for FreeRTOS is used to implement fleet provisioning functionality. RX Driver Package, FreeRTOS, AWS IoT Fleet Provisioning Library, and the demo application are available from the repository linked to below.

Demo application: iot-reference-rx : FreeRTOS reference repository

4. Running the Fleet Provisioning Demo

How to run the fleet provisioning demo application is described below.

4.1 Preparing the Running Environment

First, prepare the environment on which the demo will run. Figure 4.1 shows an example using the CK-RX65N board. Either a wired (Ethernet) or wireless (cellular) communication interface can be used to connect to AWS.

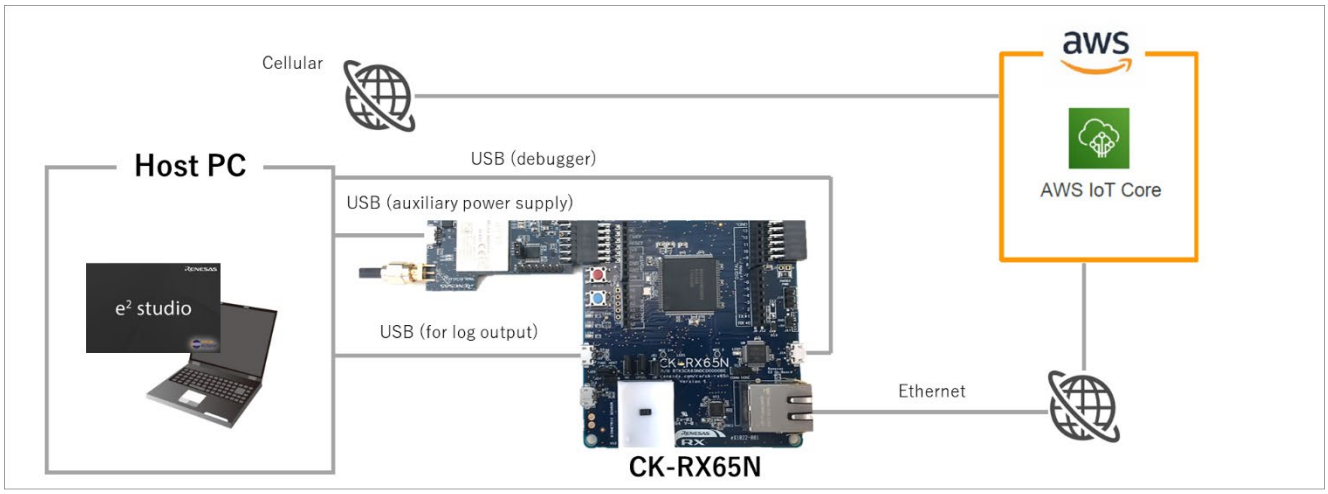


Figure 4.1 Demo Running Environment

4.2 AWS Preparation

An AWS account is required to run the fleet provisioning demo application. If you do not have an account, start by creating an account and logging in to the console. Note that the screenshots of the AWS console appearing in this application note are current as of September 2023.

AWS top page (https://aws.amazon.com/)

① Select **Sign In to the Console** → **Get Started for Free** to create a new account.

② Click **Sign In to the Console** and sign in.

③ Select **Services** → **Internet of Things** → **IoT Core** to open the AWS IoT console.

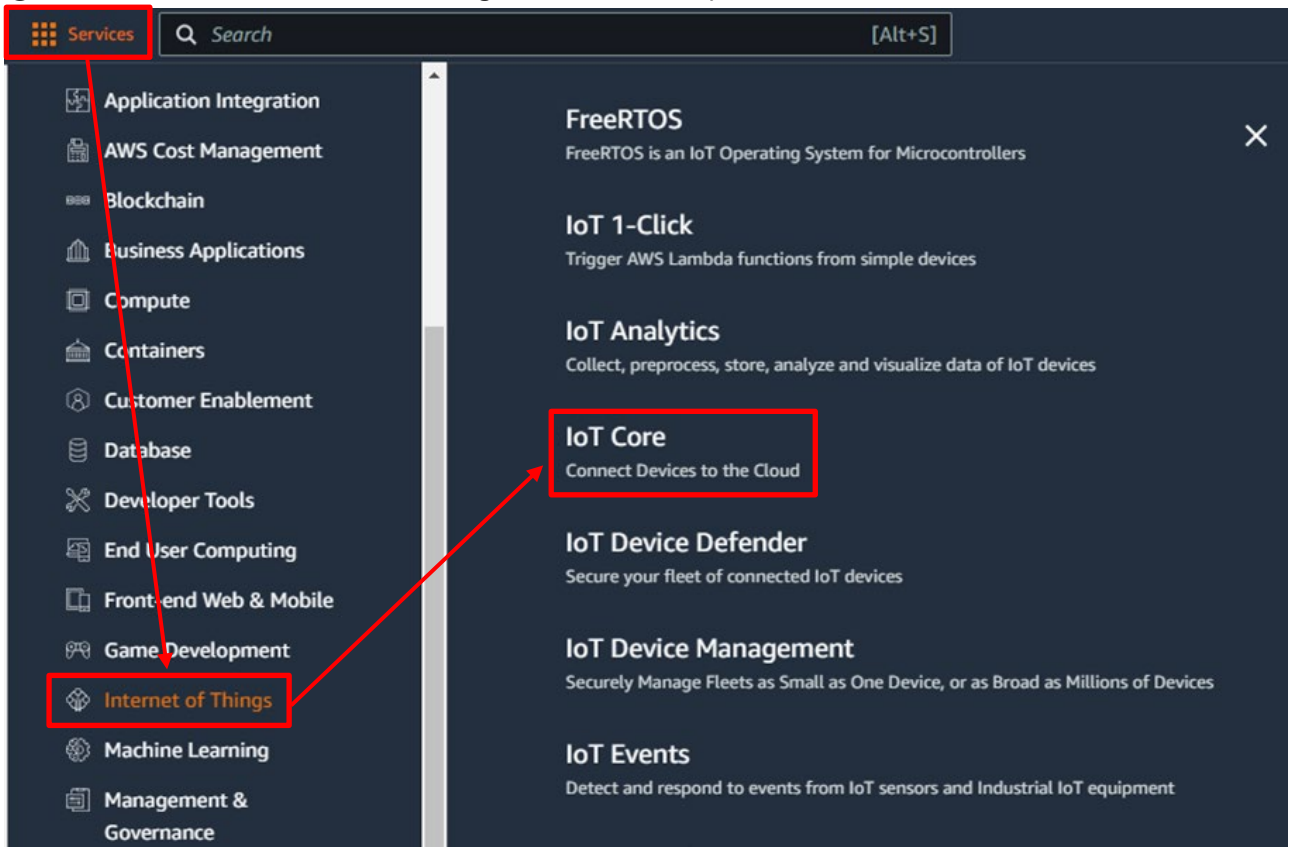


Figure 4.2 AWS Console

4.3 AWS Settings for Fleet Provisioning

It is necessary to configure AWS settings in order to run the fleet provisioning demo.

1. Policy settings
2. Generating a claim certificate and claim key pair
3. Creating a fleet provisioning template

4.3.1 Policy Settings

Follow the steps below to create AWS IoT Core policies. The first policy you create will be used when fleet provisioning is run.

Select **Security** → **Policies** and then click the **Create policy** button.

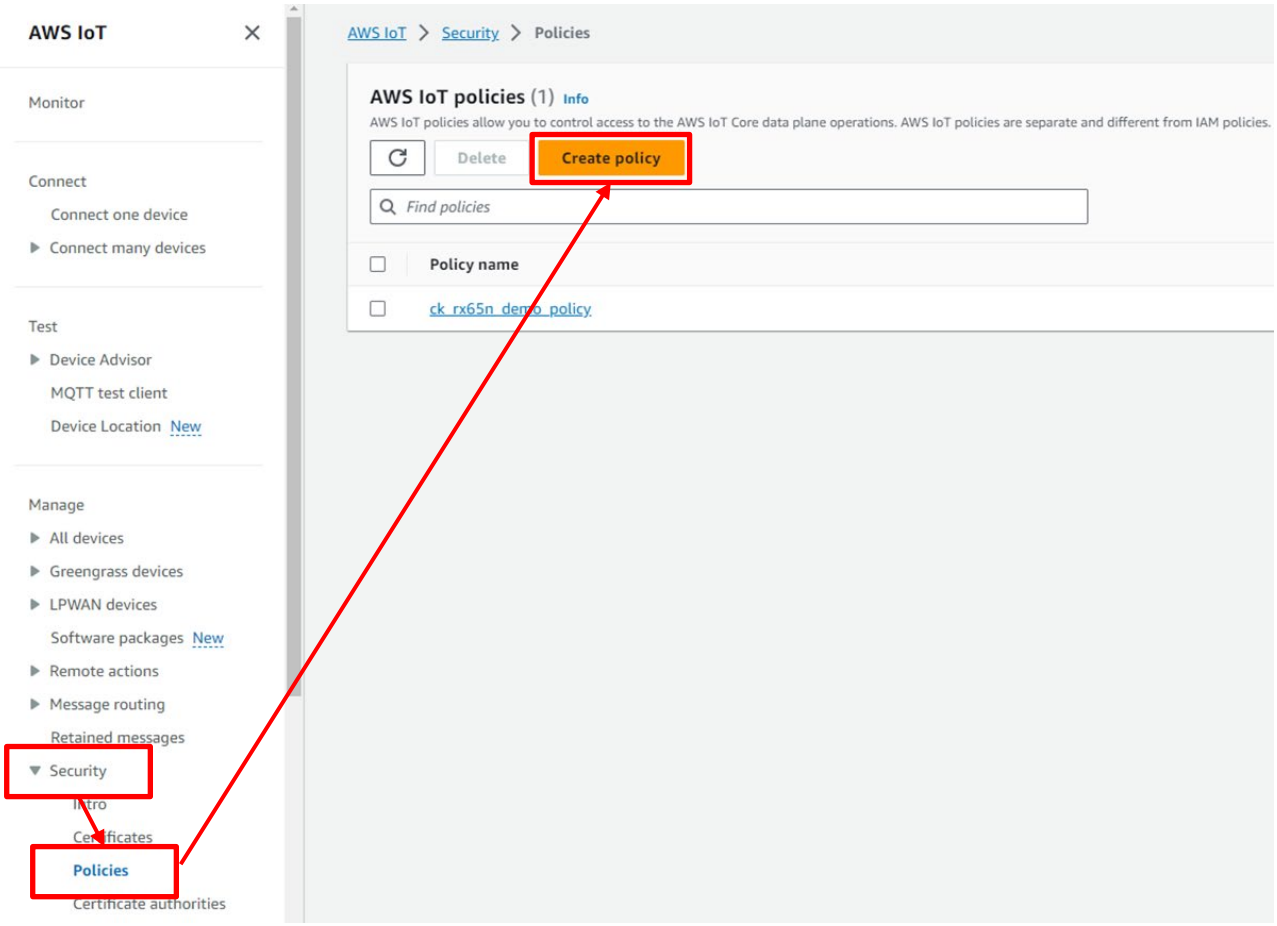


Figure 4.3 Creating an AWS IoT Policy (1)

In the **Policy name** field, enter a policy name of your choice.

Click the **JSON** button to display the policy document input field, then copy and paste the policy document shown in Figure 4.5 into the input field. When copying and pasting the policy document in Figure 4.5, make the following changes:

・Change “ap-northeast-1” to match the region used.

・Change <account id> to your own account ID (account ID is the 12-digit number after @ that is displayed by clicking on the account name in the upper right corner, excluding the hyphen)

Click the **Create** button to create the policy.

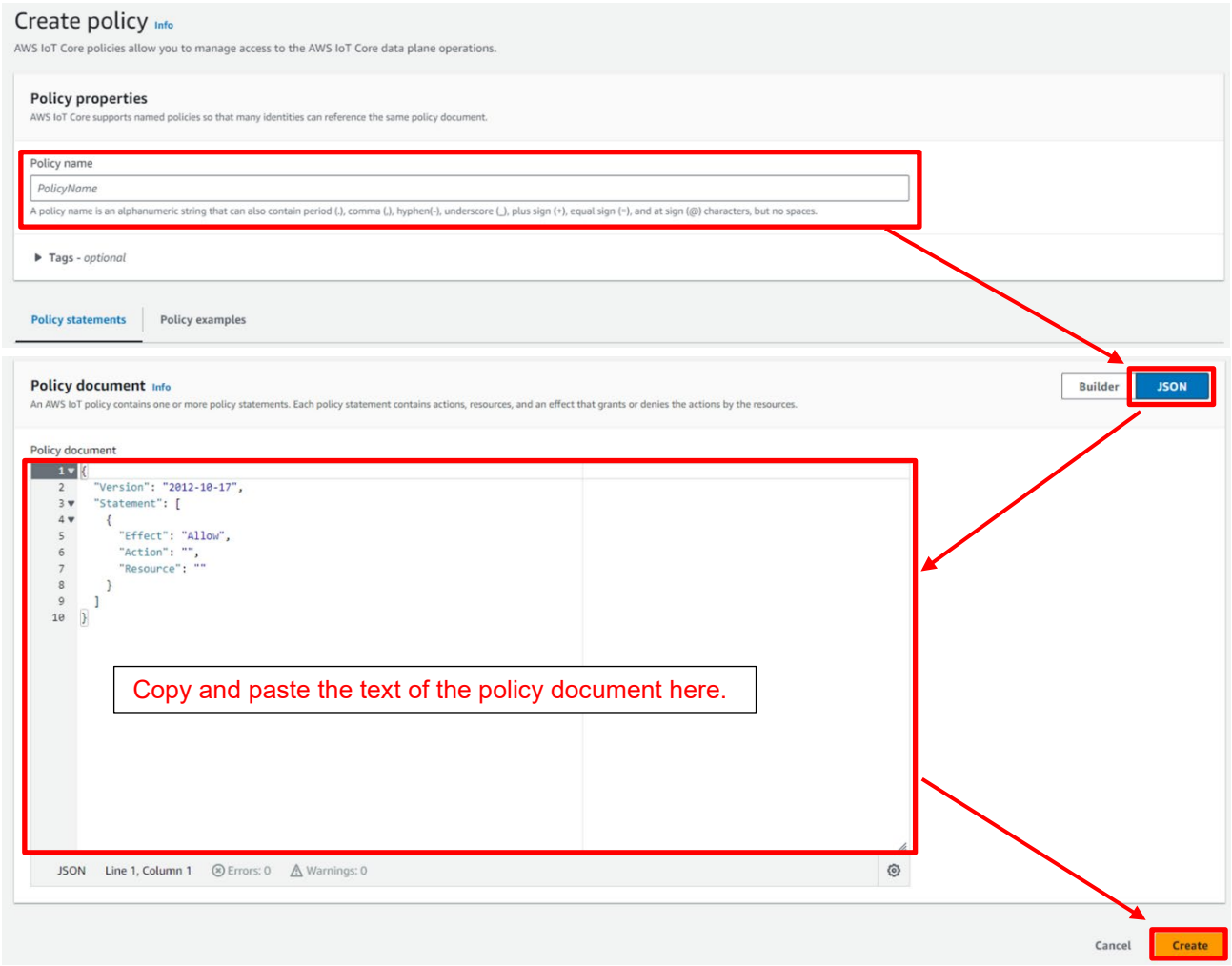


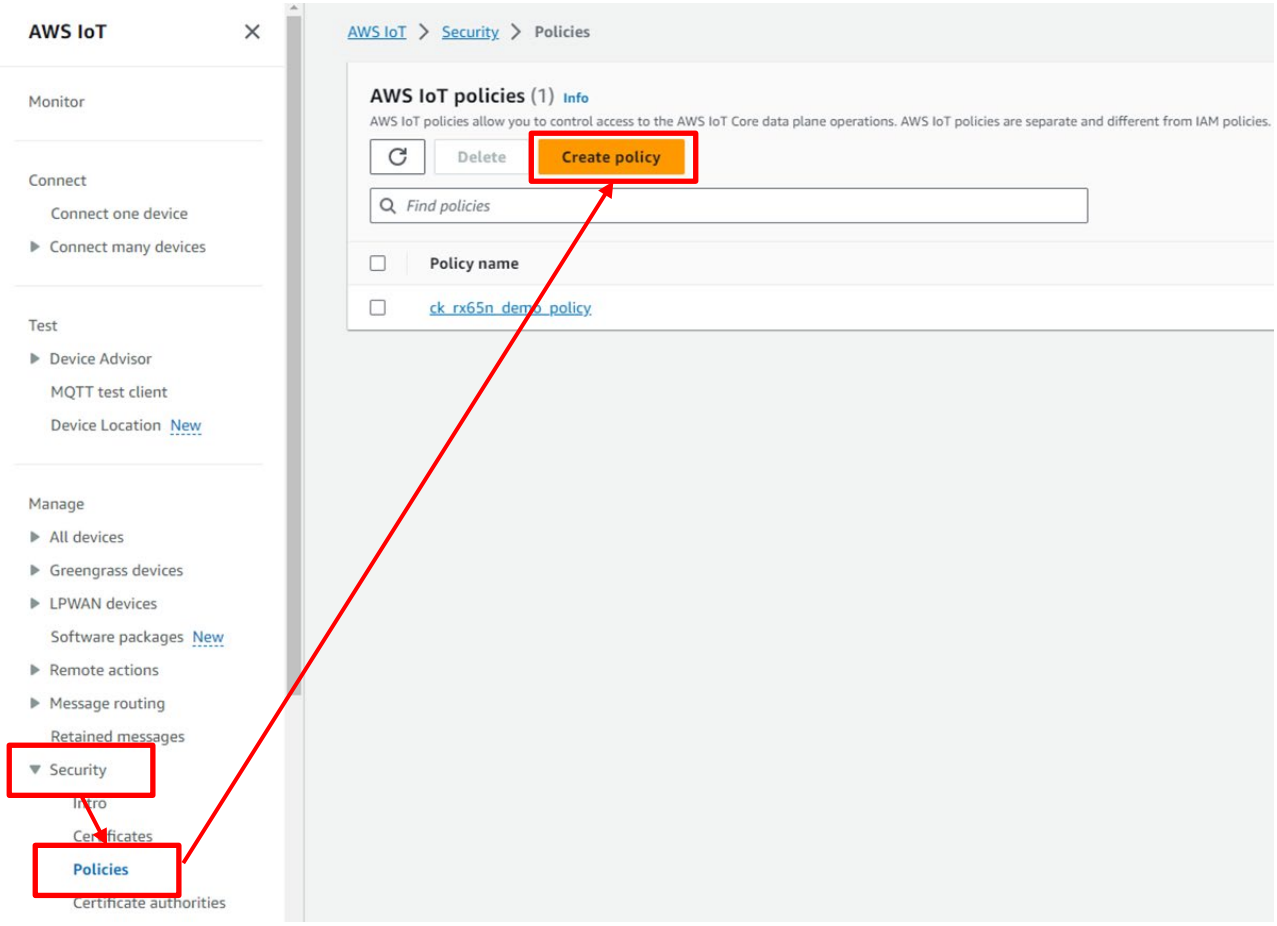
Figure 4.4 Creating an AWS IoT Policy (2)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive",
        "iot:RetainPublish"
      ],
      "Resource": [
        "arn:aws:iot:ap-northeast-1:<account id>:topic/$aws/certificates/create-from-csr/*",
        "arn:aws:iot:ap-northeast-1:<account id>:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:ap-northeast-1:<account id>:topicfilter/$aws/certificates/create-from-csr/*",
        "arn:aws:iot:ap-northeast-1:<account id>:*"
      ]
    }
  ]
}
```

Figure 4.5 Policy Document

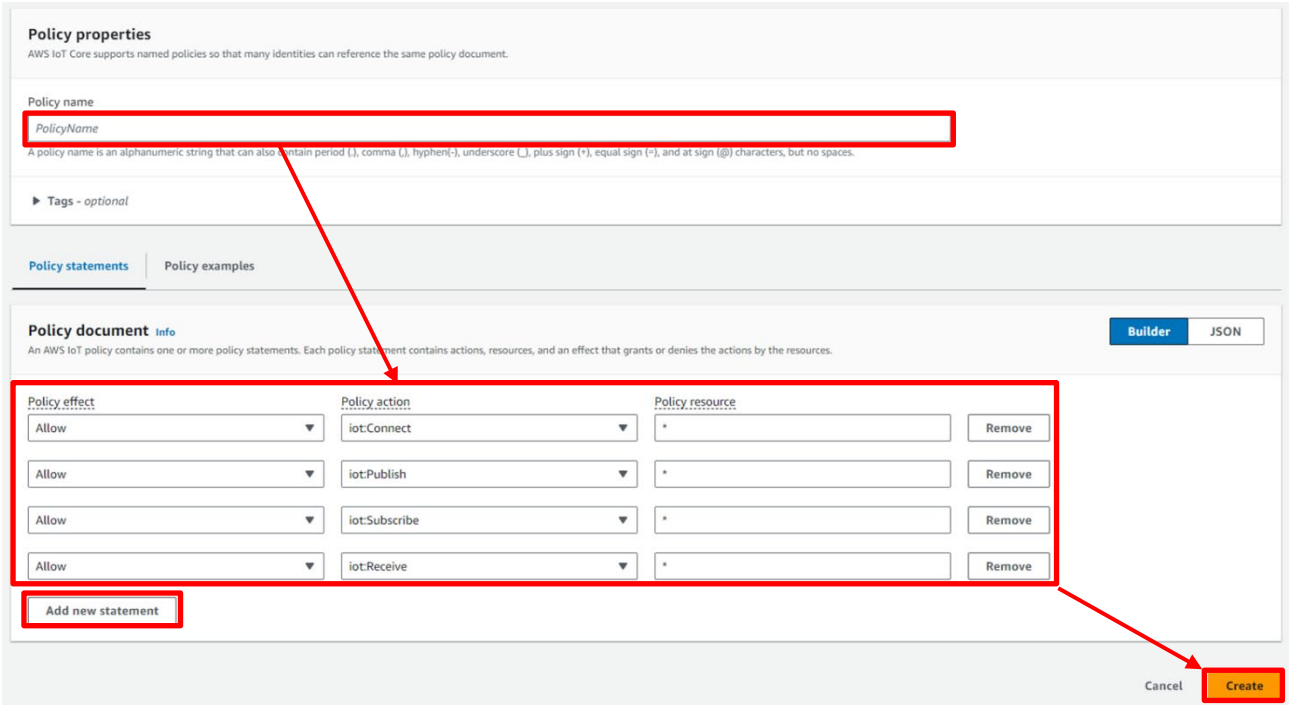
Next, create a policy that will be attached to things created after fleet provisioning is run.

Select **Security** → **Policies** and then click the **Create policy** button.



In the **Policy name** field, enter a policy name of your choice.

For **Policy action** under **Policy document**, select **Allow** for **iot:Connect**, **iot:Publish**, **iot:Subscribe**, and **iot:Receive**. For **Policy resource** enter the wildcard character (*) to allow all resources. By default you can configure one statement. Click the **Add new statement** button to add additional statements as needed.



4.3.2 Generating a Claim Certificate and Claim Key Pair

Generate a provisioning claim certificate and provisioning claim key pair for use in fleet provisioning.

Select **Security** → **Certificates** and then click **Add certificate** → **Create certificate**.

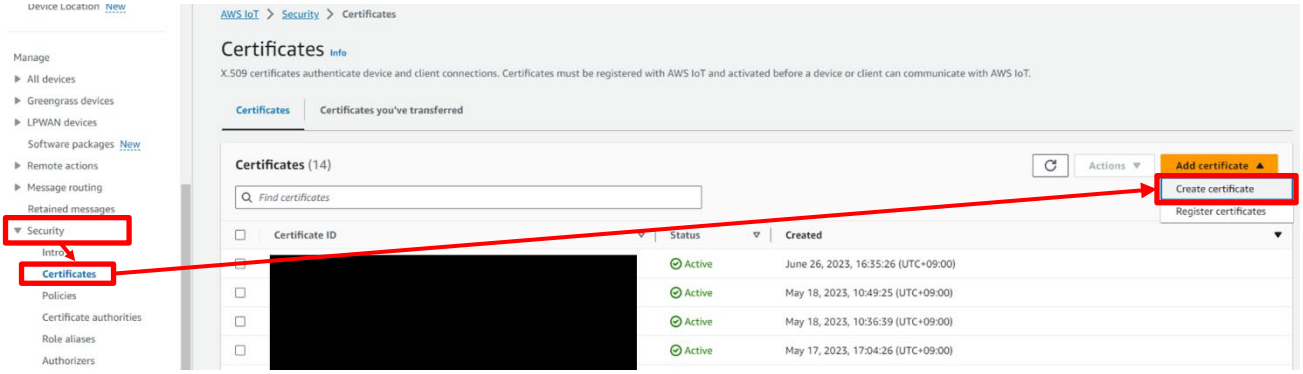


Figure 4.6 Creating a Certificate

Click **Auto-generate new certificate (recommended)** → **Create**.

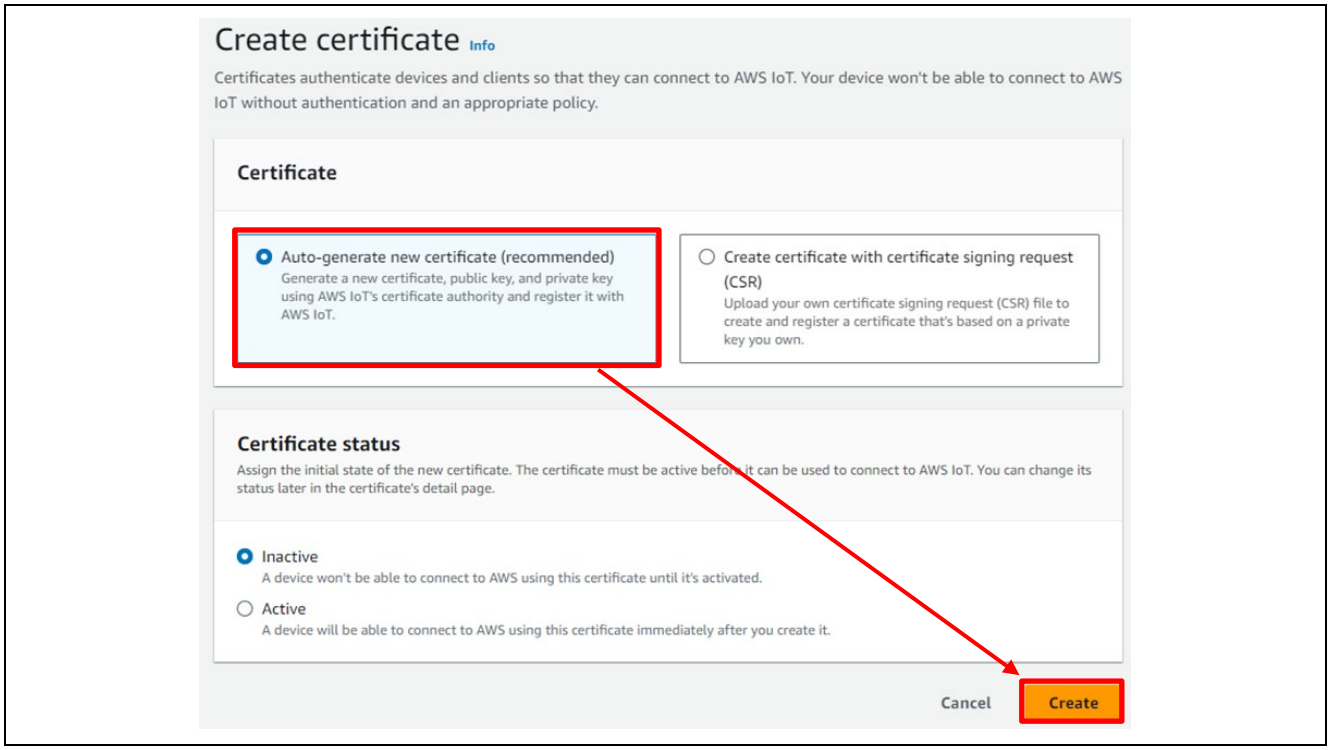


Figure 4.7 Creating a Certificate Automatically

Download the newly created certificate ① and key pair ②③, then click the **Continue** button.

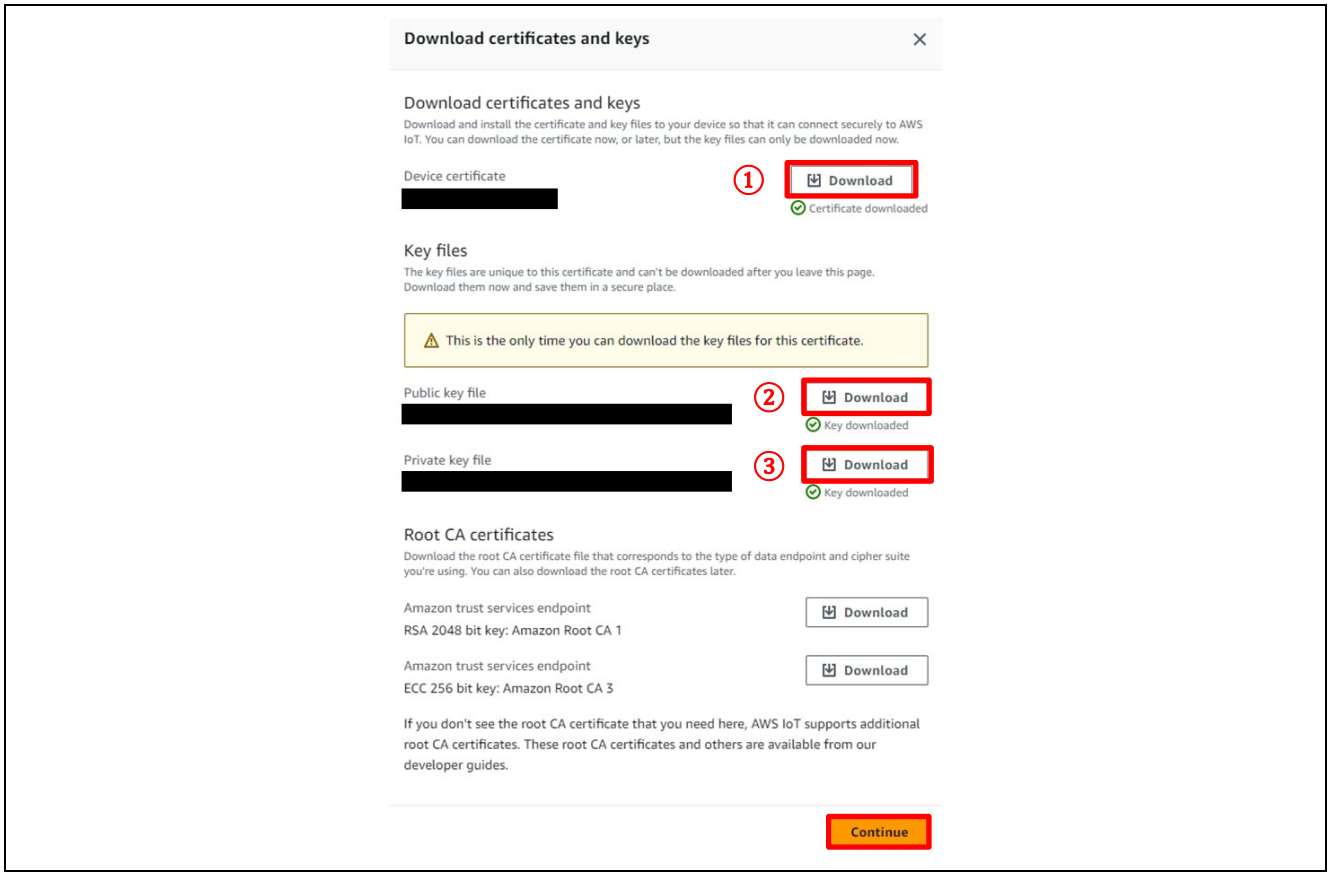


Figure 4.8 Downloading the Certificate and Key Pair

On the AWS console, select **Security** → **Certificates** and select the newly generated certificate ID.

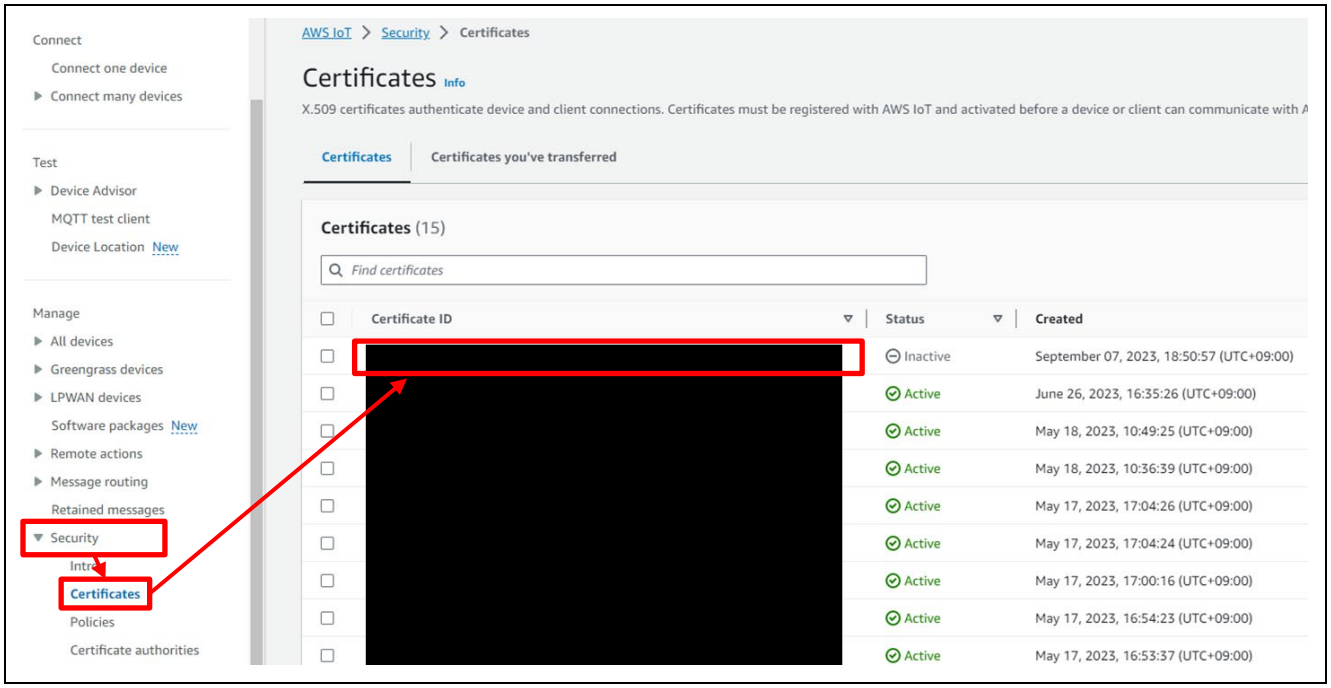


Figure 4.9 Certificate Settings

Click **Actions** → **Activate** to activate the certificate. Also click the **Attach policies** button.

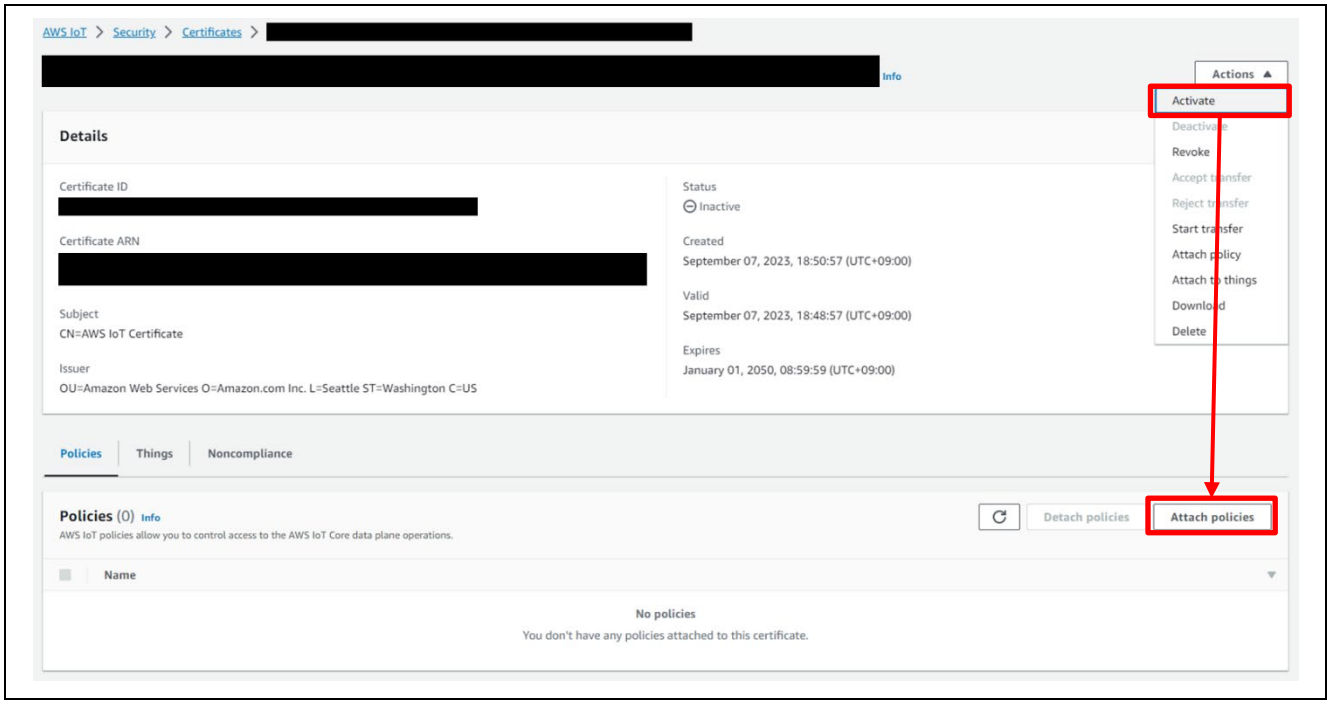


Figure 4.10 Certificate Settings: Attach Policies (1)

Clicking the **Attach policies** button opens the dialog box shown in Figure 4.11.

Select the policy to be used when fleet provisioning is run, created in 4.3.1, Policy Settings, and then click the **Attach policies** button to attach it to the certificate.

This completes the settings related to generation of the claim certificate and claim key pair.

Figure 4.11 Certificate Settings: Attach Policies (2)

4.3.3 Creating a Fleet Provisioning Template

Select **Connect many devices** → **Connect many devices**, then click the **Create provisioning template** button.

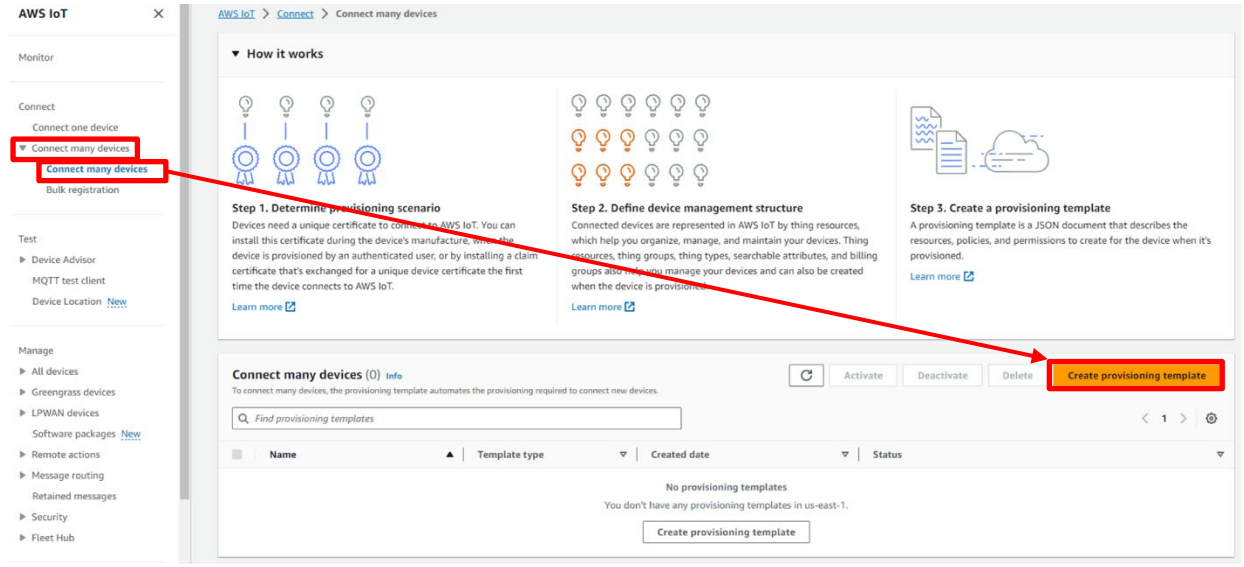


Figure 4.12 Creating a Provisioning Template (1)

Select **Provisioning devices with claim certificates**, then click the **Next** button.

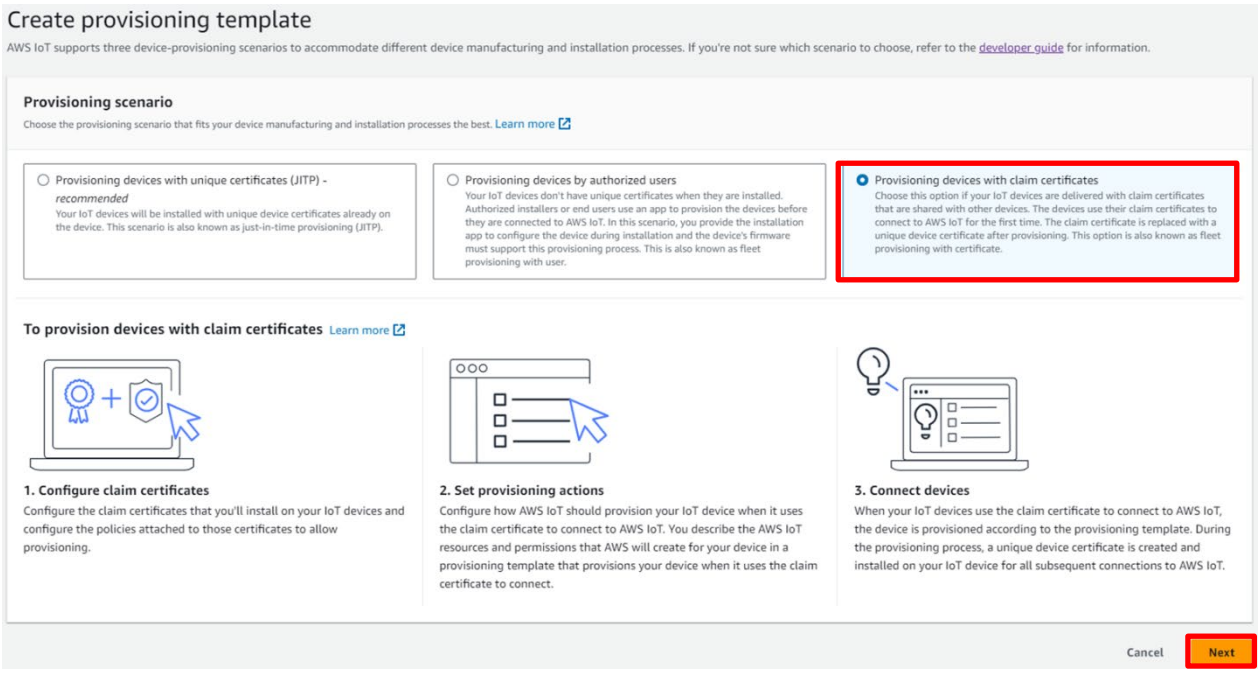


Figure 4.13 Creating a Provisioning Template (2)

On the template creation screen, specify the provisioning template status, template name, and provisioning role. For **Provisioning template status** select **Active**, and enter the name of the provisioning template. Then click the **Create new role** button and enter the role name.

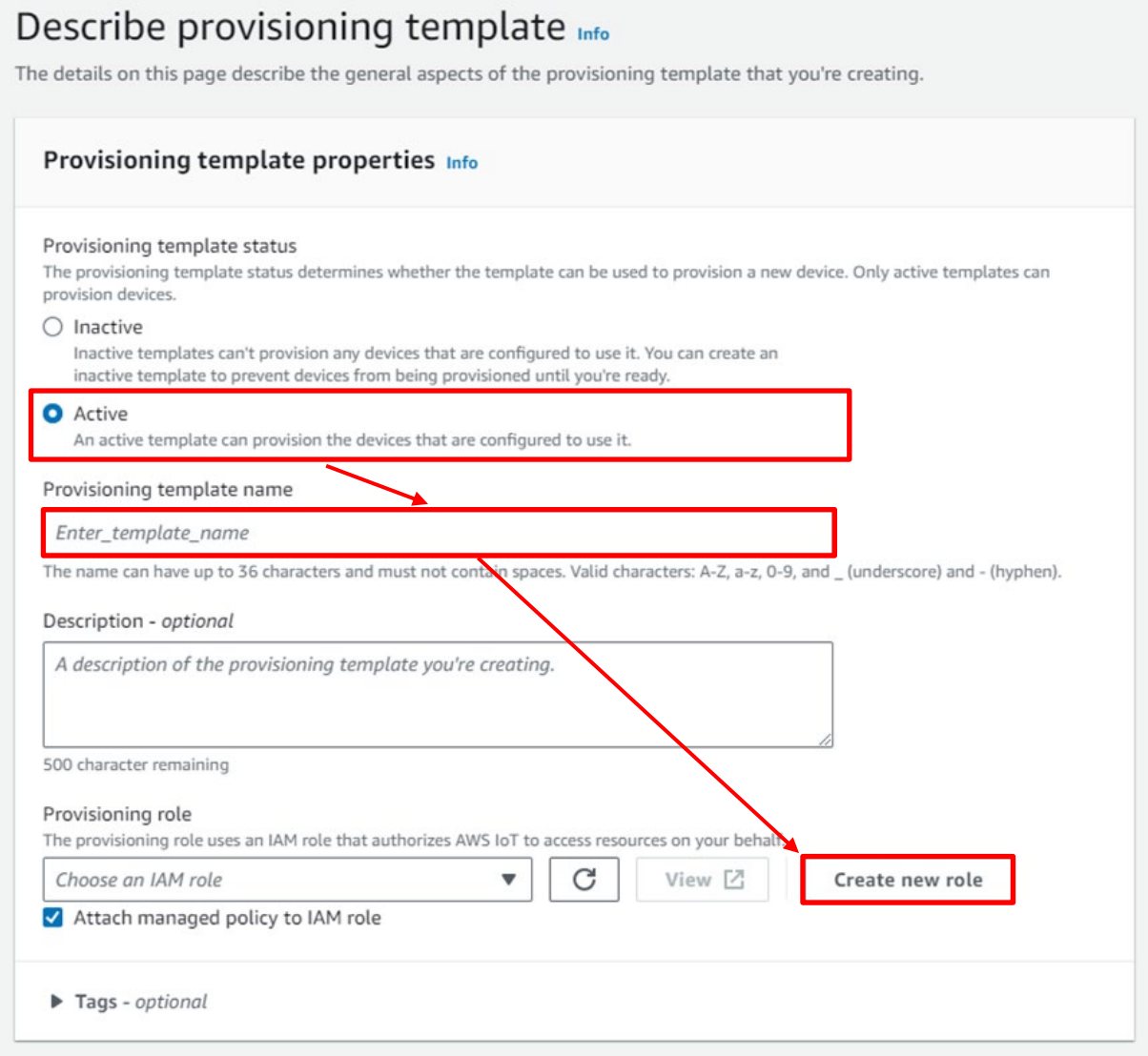


Figure 4.14 Creating a Provisioning Template (3)

For **Claim certificate policy**, select the policy to be used when fleet provisioning is run, created in 4.3.1, for **Claim certificate**, select the certificate created in 4.3.2, and click the **Next** button.

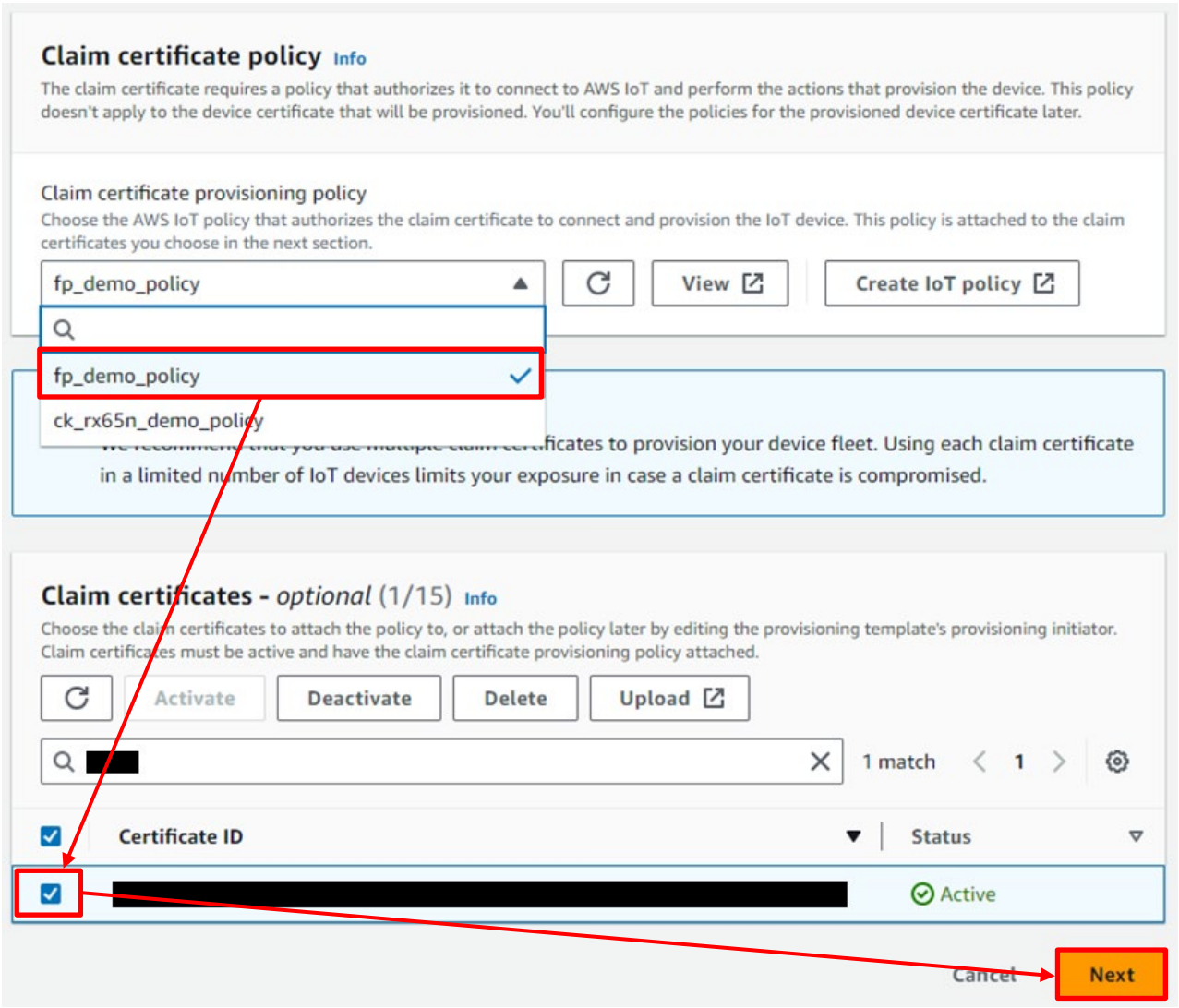


Figure 4.15 Creating a Provisioning Template (4)

For **Pre-provisioning actions**, select **Don't use a pre-provisioning action**. Also, under **Automatic thing creation**, turn on **Automatically create a thing resource when provisioning a device**, and if necessary enter a character string of your choice as the thing name prefix. The thing name registered with AWS will be generated from this character string and the serial number set by the program. After entering the prefix, click the **Next** button.

Note: The demo does not use pre-provisioning actions. Refer to the page linked to below for information on using pre-provisioning actions.
https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html
"Using pre-provisioning hooks with the AWS CLI"

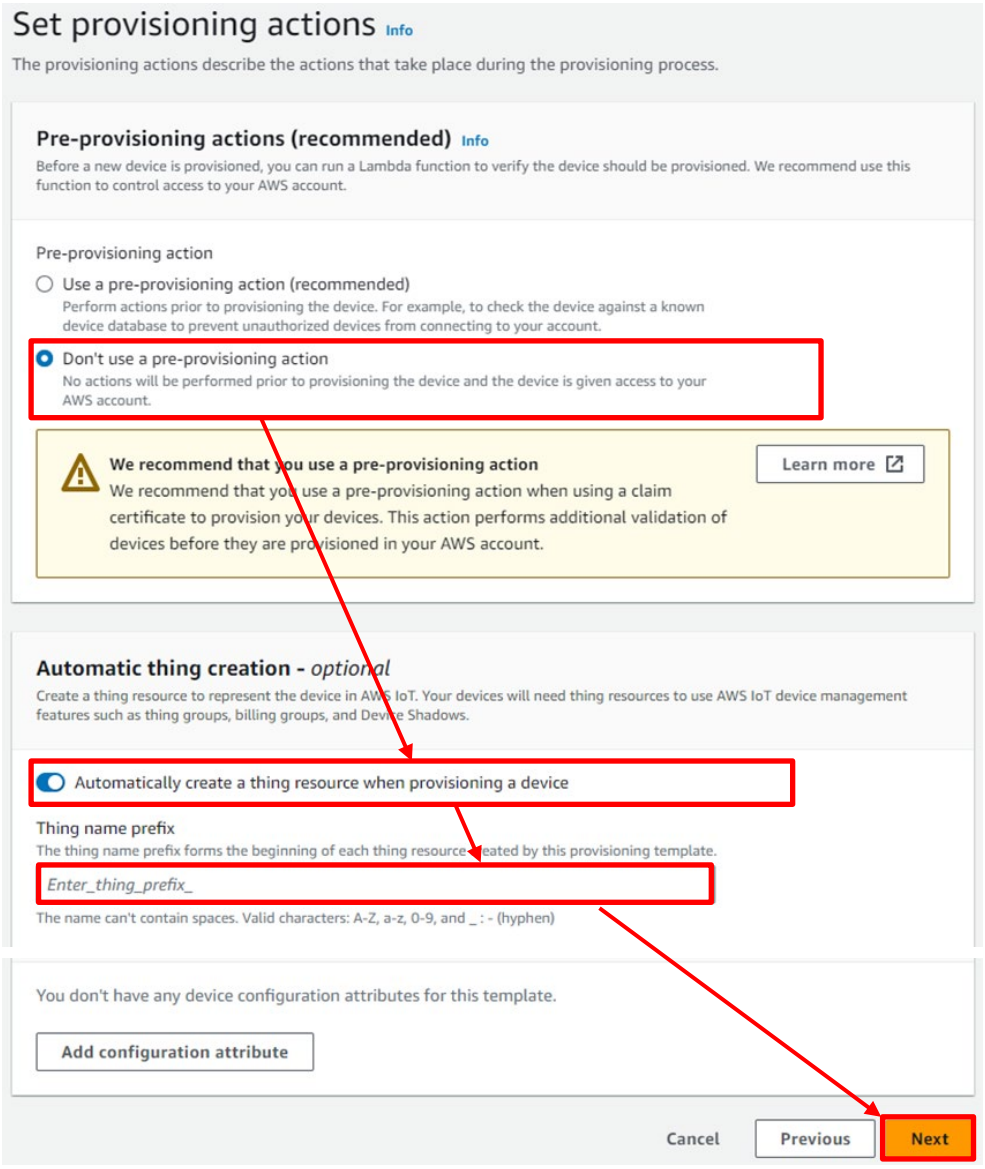


Figure 4.16 Creating a Provisioning Template (5)

For **Set device permissions**, check the box next to the policy attached to newly created things, which was created in 4.3.1, then click the **Next** button.

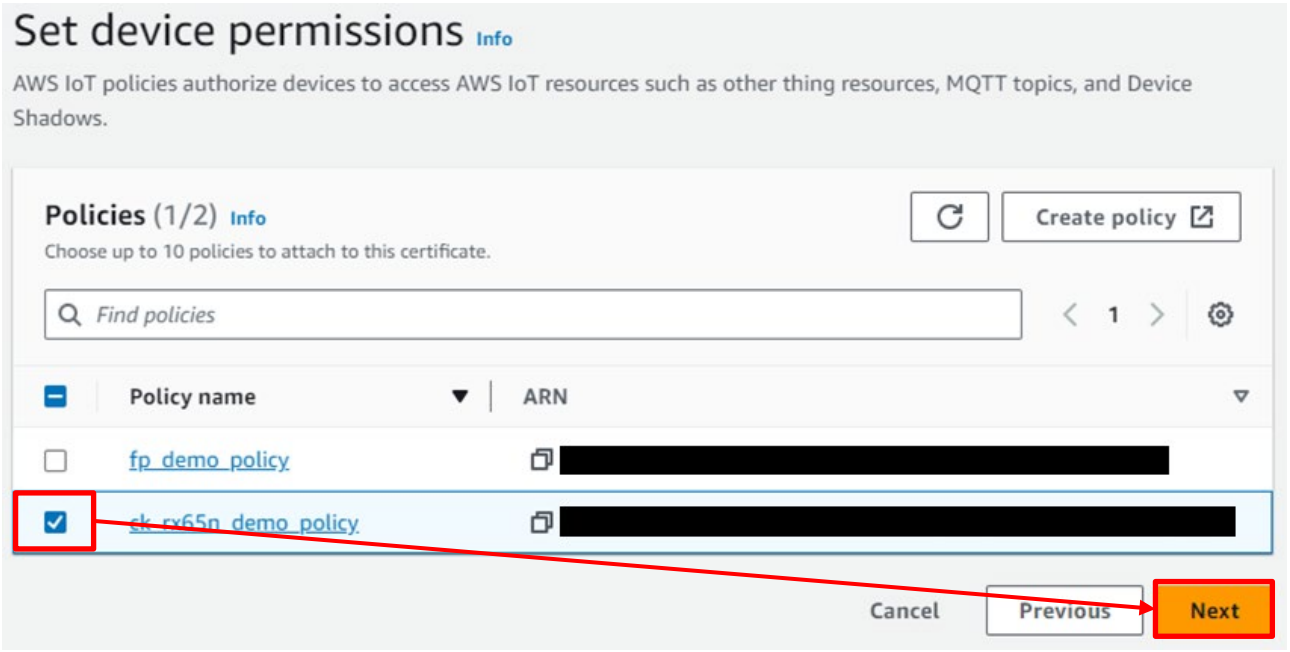


Figure 4.17 Creating a Provisioning Template (6)

Click the **Create template** button to complete the process of creating a fleet provisioning template.

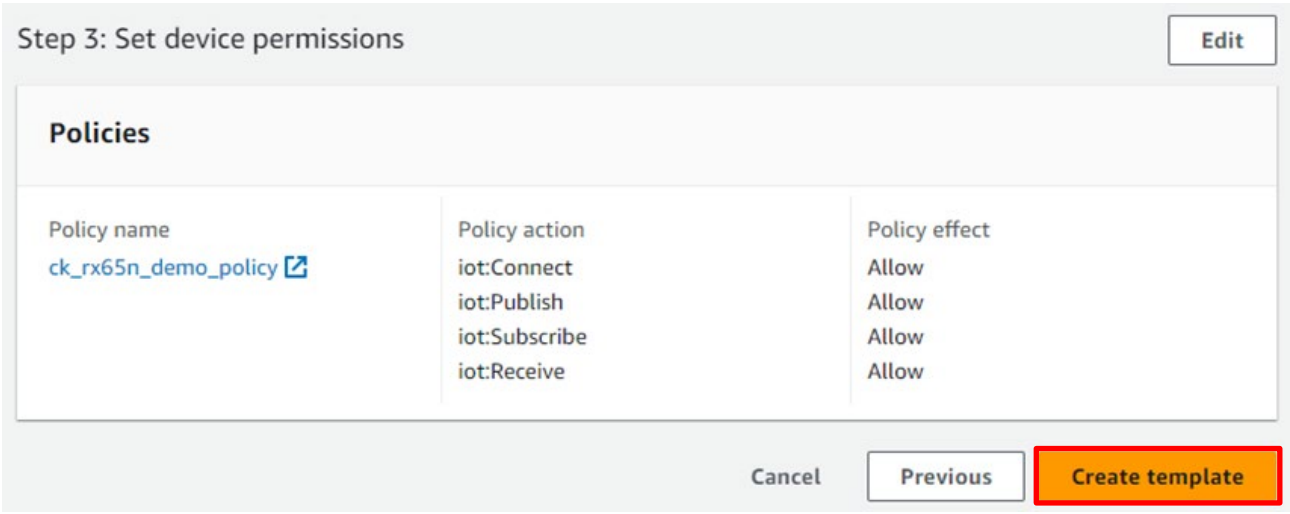


Figure 4.18 Creating a Provisioning Template (7)

4.4 Deploying the Demo Project

Clone the GitHub repository linked to below to a Git client of your choice and then import it into e² studio. Clone iot-reference-rx to a folder of your choice.

Demo application: [iot-reference-rx : FreeRTOS reference repository]

Note: Due to a limitation affecting e² studio, the path name of the “folder of your choice” (including the folder name) must be no more than 35 characters long. If you specify a path name of 36 or more characters, an error will result when you attempt to build the project.

From the top left of the menu bar of e² studio, select **File** → **Import** → **General** → **Existing Projects into Workspace**, then use the browse button next to the **Select root directory** to navigate to and select **Projects\aws_ryz014a_ck_rx65n\e2studio_ccrx**. Figure 4.19 shows an example in which the CK-RX65N board (Cellular version) is used. Substitute different values as appropriate for “ck_rx65n” or “ryz014a” if you are using a different board or a different communication interface.

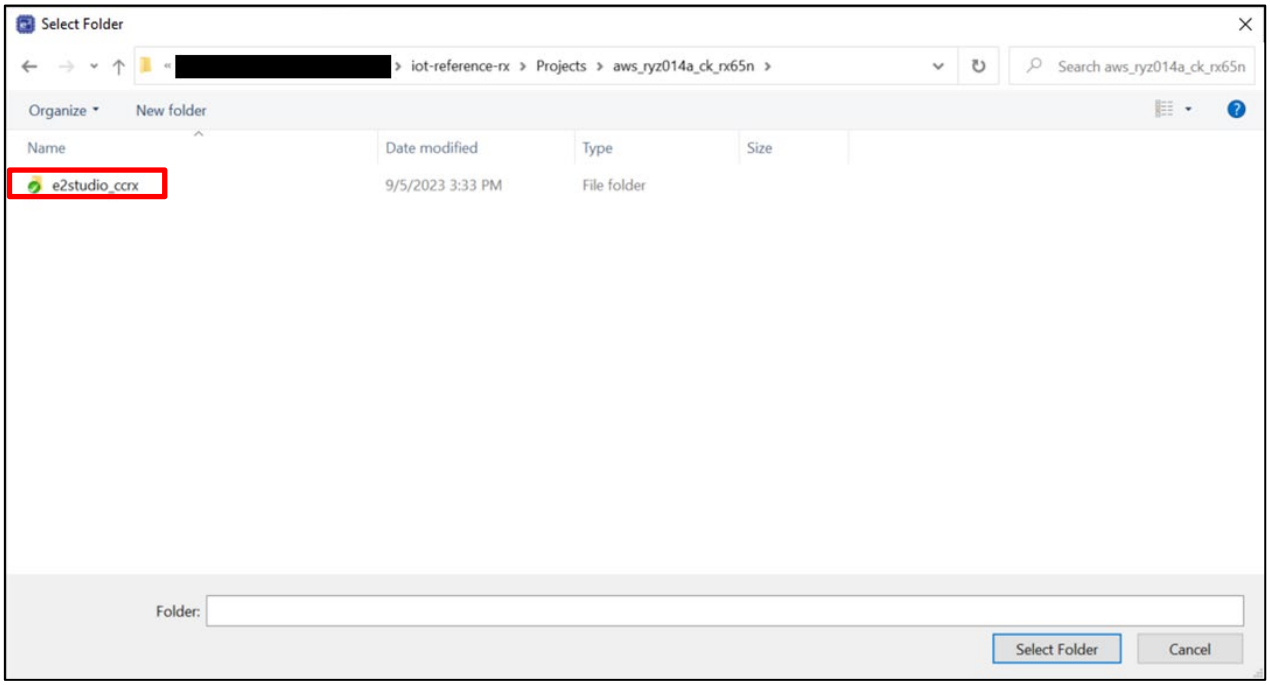


Figure 4.19 Demo Project

4.5 FreeRTOS Settings

You will need to make a modification to the program in order to run the demo.

4.5.1 Modifying the Configuration File

From the **Project Explorer** panel in e² studio, open aws_ryz014a_ck_rx65n/src/frtos_config/demo_config.h and change the value of **ENABLE_FLEET_PROVISIONING_DEMO** to **1**.

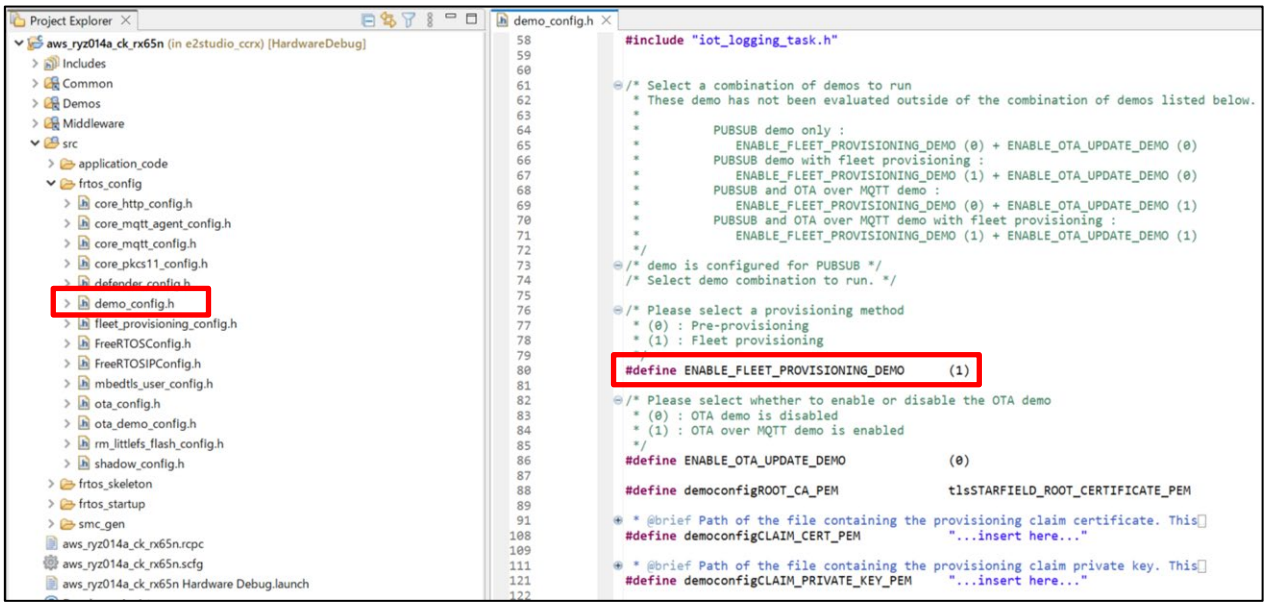


Figure 4.20 Location of Modification in demo_config.h

4.5.2 Cellular information settings

From the **Project Explorer** panel in e2 studio, open aws_ryz014a_ck_rx65n/aws_ryz014a_ck_rx65n.scfg and launch the Smart Configurator. (Figure 4.21)

Select the **Components** tab in the Smart Configurator and select **Middleware** → **Generic** → **r_cellular** from the Components. Set each item of **Access point name**, **Access point login ID**, **Access point password** and **SIM card PIN code** according to the SIM card you are using. If there is no content to enter, leave it blank. (Figure 4.22)

After entering the Cellular information, click the **Generate Code** button to apply the settings to the program.

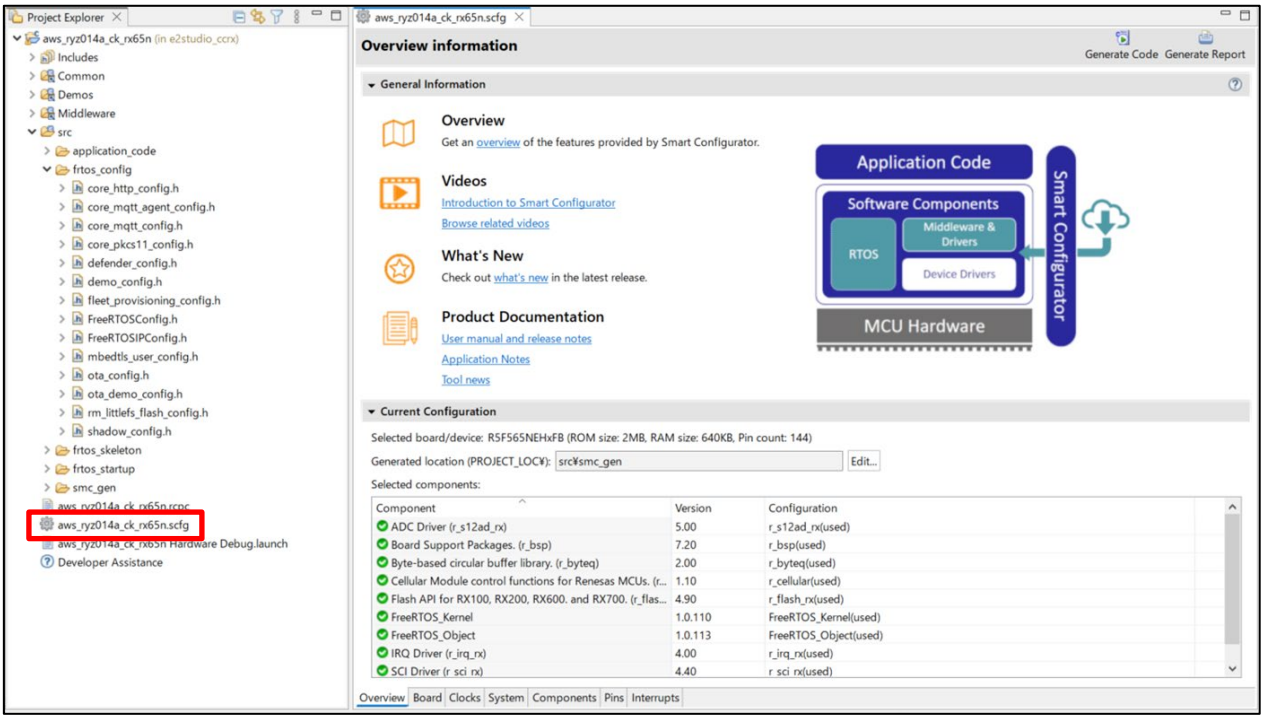


Figure 4.21 Launch the Smart Configurator

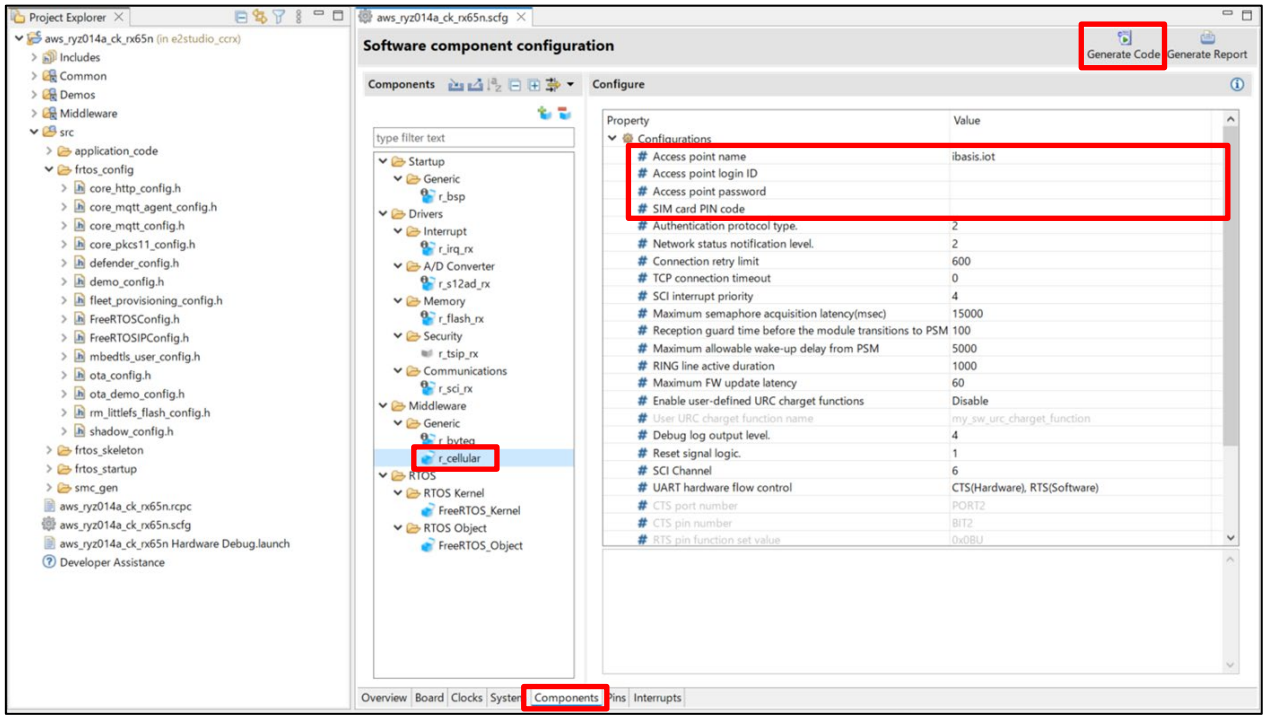


Figure 4.22 Entering Cellular information

4.6 Building and Running the Program

Build the project, program it to the device, and run the demo.

First, on the **Project Explorer** panel, right-click aws_ryz014a_ck_rx65n and select **Build Project** to build the project.

Next, select **Run** → **Debug Configurations…** from the e^2 studio menu to open the Debug Configurations window. In the list at the left of the Debug Configurations window, select **Renesas GDB Hardware Debugging** → **aws_ryz014a_ck_rx65n Hardware Debug**. Then select the **Debugger** tab followed by the **Connection Settings** tab (indicated by arrows in Figure 4.23).

Check to make sure that the settings of the items enclosed by red frames in Figure 4.21 match those shown, then click the **Debug** button to download to the device the executable data produced by building the project.

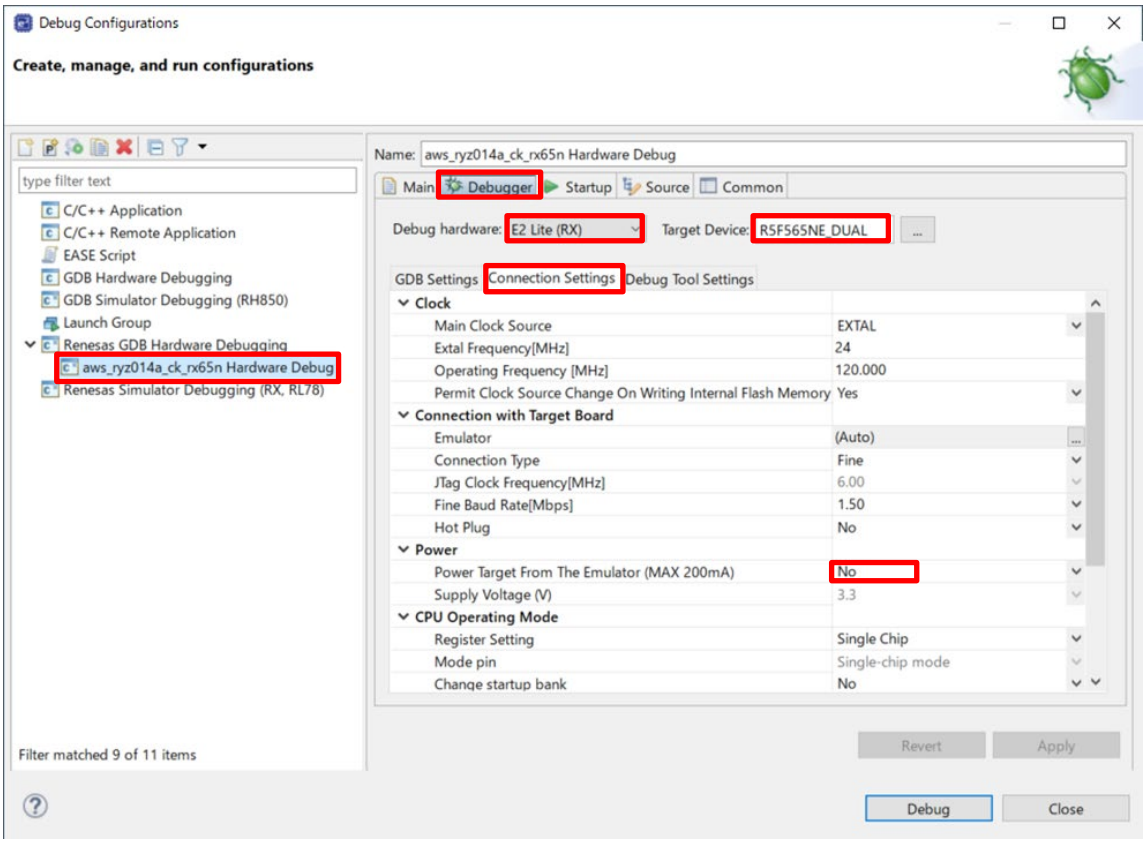


Figure 4.23 Debug Configurations

Launch Tera Term in order to enter the claim certificate, claim private key, endpoint, and provisioning template name.

After Tera Term starts, select **Serial** and **USB Serial Device**, then click the **OK** button.

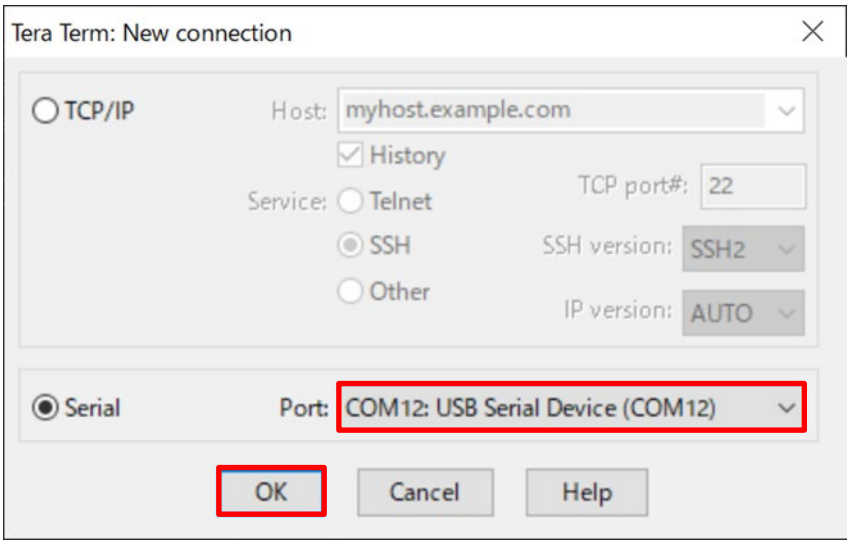


Figure 4.24 Initial Window when Tera Term Starts

Select **Setup** → **Serial port…** from the menu, configure the serial port setting items enclosed by red frames as shown, and then click the **New setting** button.

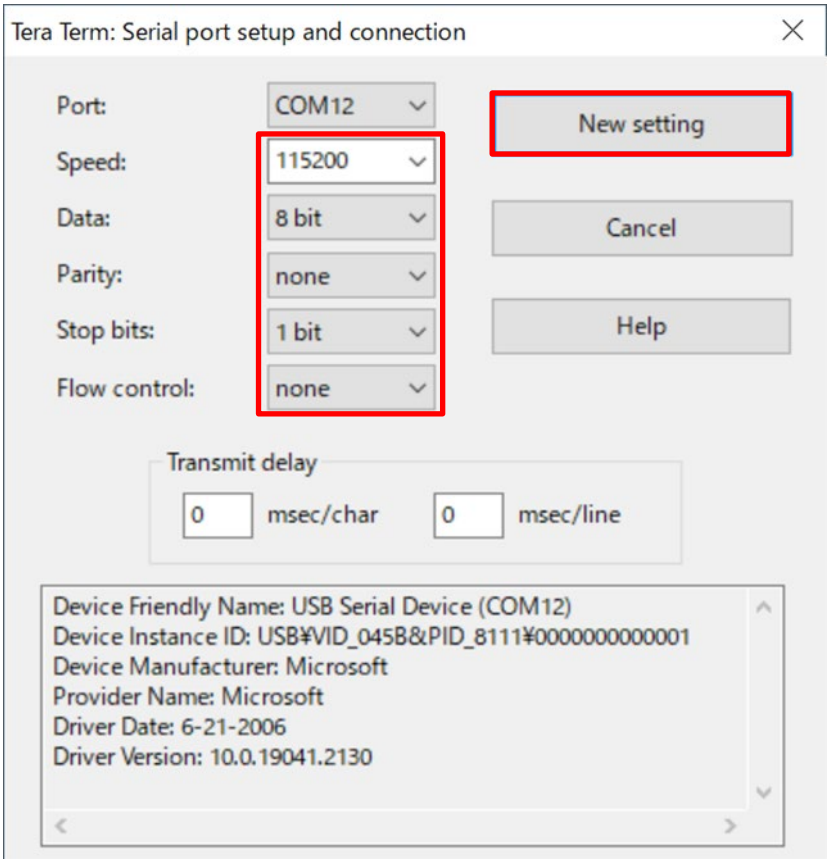


Figure 4.25 Serial Port Setup

Select **Setup** → **Terminal…** from the menu, set **Receive:** to **AUTO** and **Transmit:** to **CR+LF** as shown in the red frames, and then click the **OK** button.

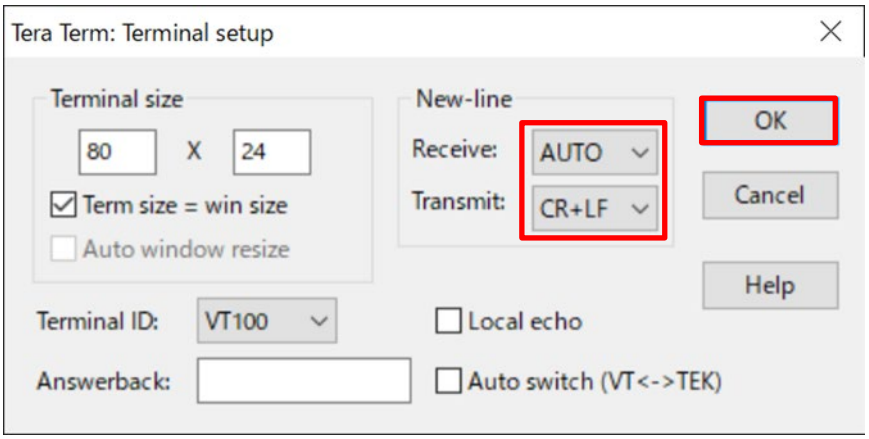


Figure 4.26 Terminal Setup

From the AWS IoT console, select **MQTT test client**, enter **#** under **Topic filter**, and click the **Subscribe** button.

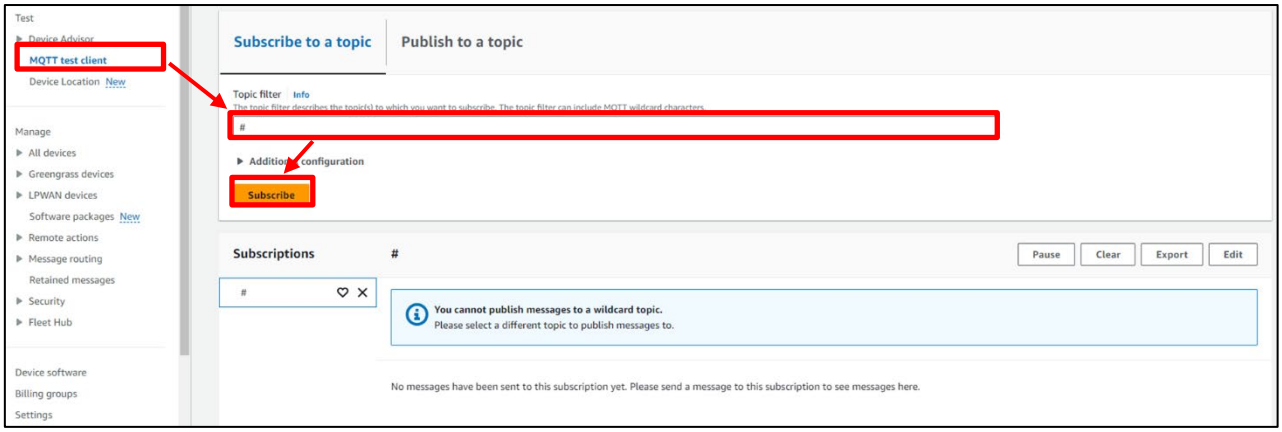


Figure 4.27 MQTT Test Client Settings

In e^2 studio, press **Resume** (F8) to display the text output shown below in Tera Term. Within 10 seconds, type **CLI** in Tera Term and press the Enter key.

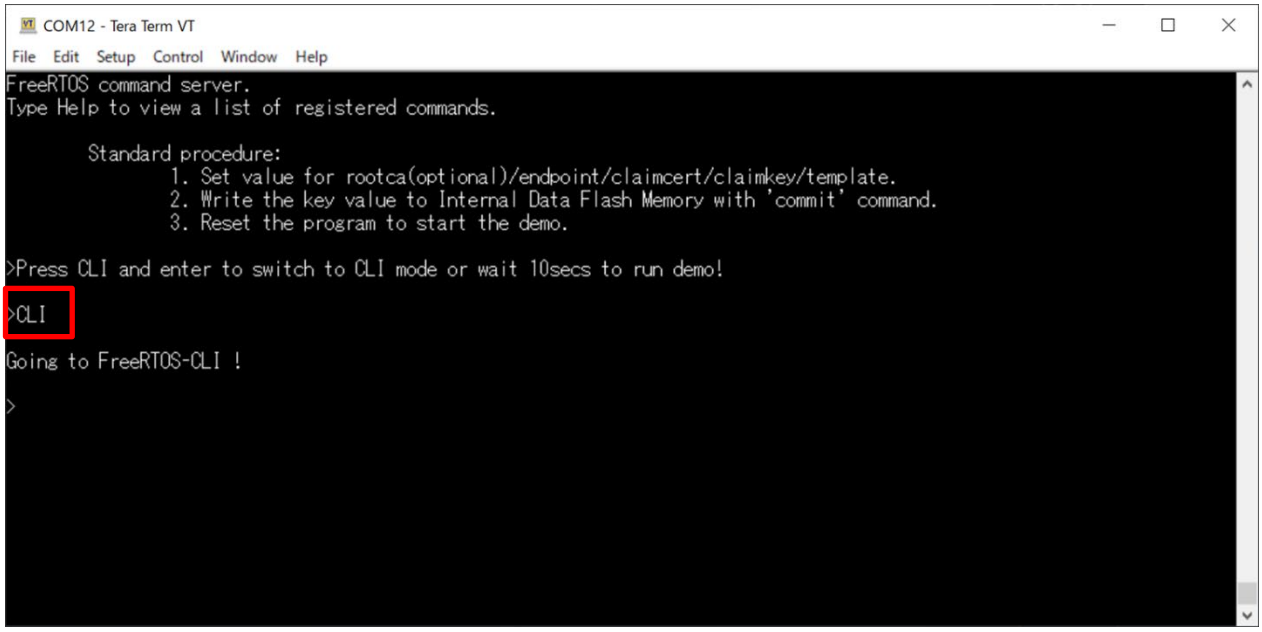


Figure 4.28 Entering Information Using CLI (1)

It is possible that information may have been stored already if the demo was run previously, so type **format** in Tera Term and press the Enter key.

This causes all stored information to be erased.

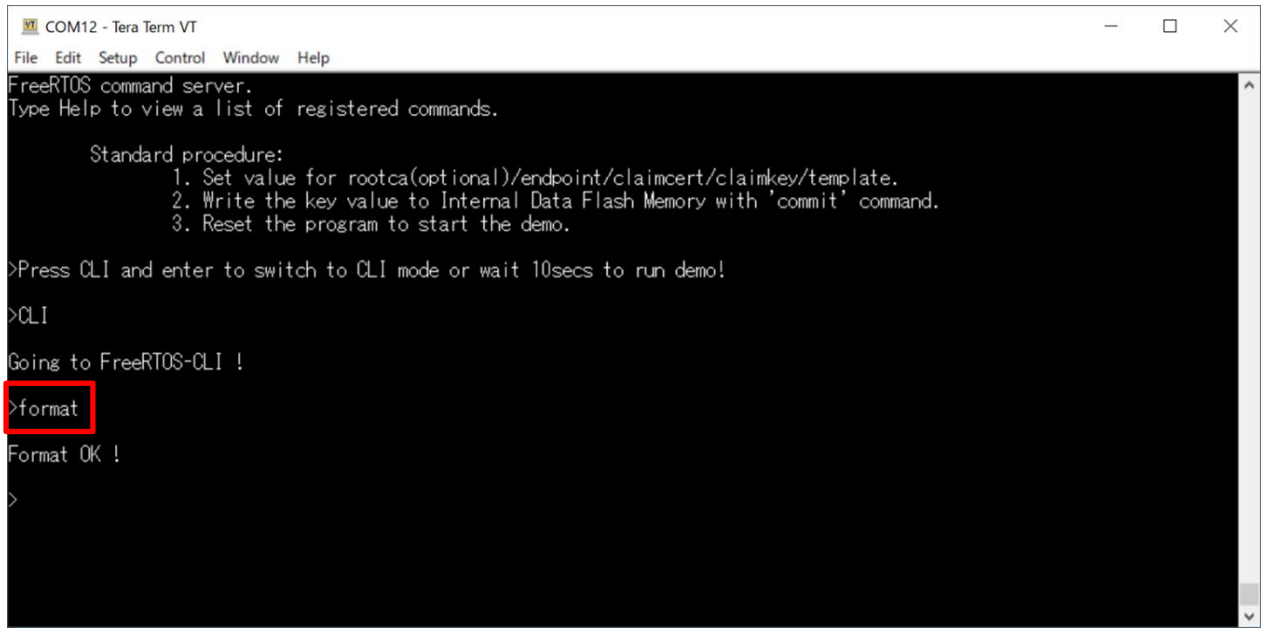


Figure 4.29 Entering Information Using CLI (2)

To enter the endpoint, type **conf set endpoint <endpoint>** in Tera Term and press the Enter key.

For **<endpoint>**, enter the value in the format **xxxxxxxxxx.amazonaws.com** that is displayed for **Endpoint** when you select **Settings** → **Device data endpoint** on the AWS IoT console.

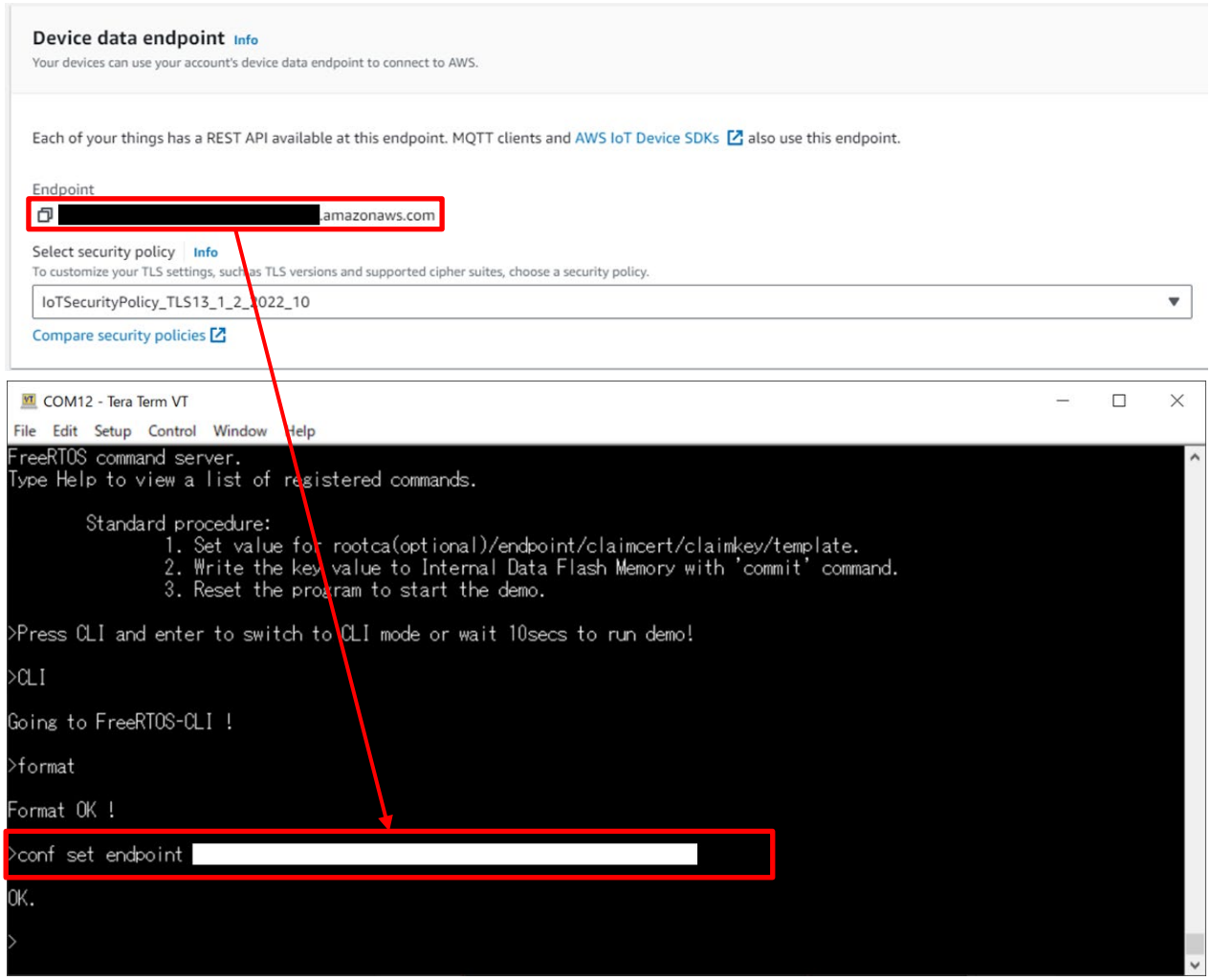


Figure 4.30 Entering Information Using CLI (3)

To enter the provisioning template name, type **conf set template <template_name>** in Tera Term and press the Enter key.

For **<template_name>**, enter the name of the provisioning template created in 4.3.3.

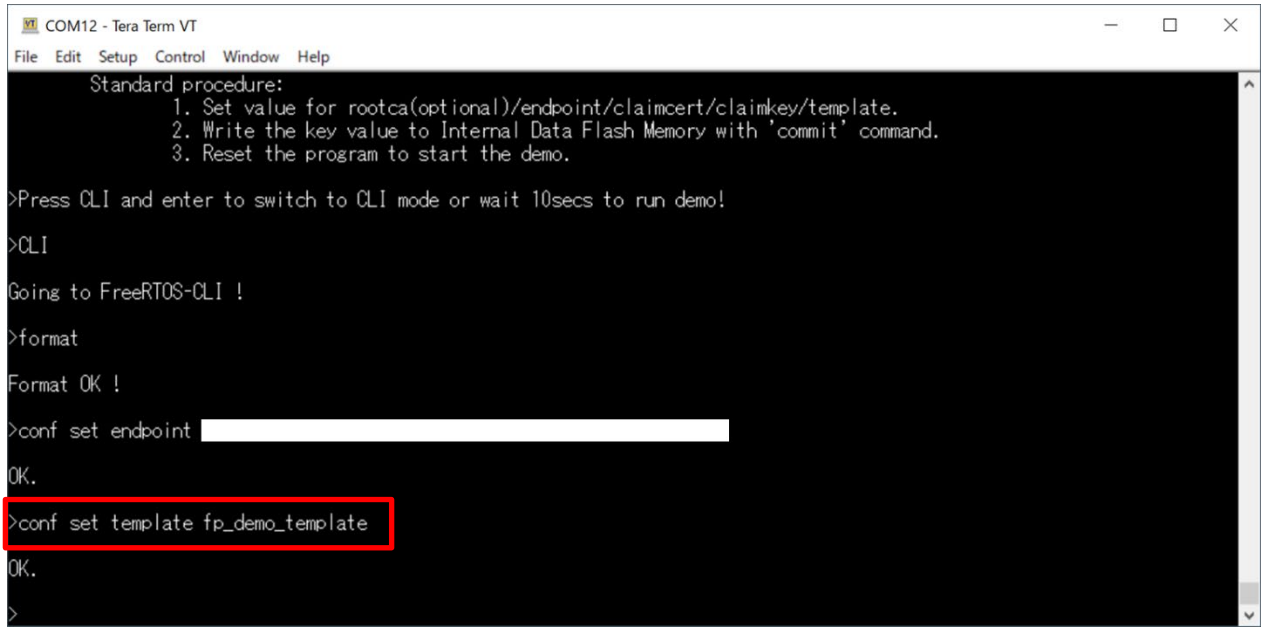


Figure 4.31 Entering Information Using CLI (4)

To enter the provisioning claim certificate, type **conf set claimcert** in Tera Term. Next, drag and drop the provisioning claim certificate file (**xxxx-certificate.pem.crt**) created in 4.3.2 onto the Tera Term window (**Send File**). Finally, press the Enter key in Tera Term.

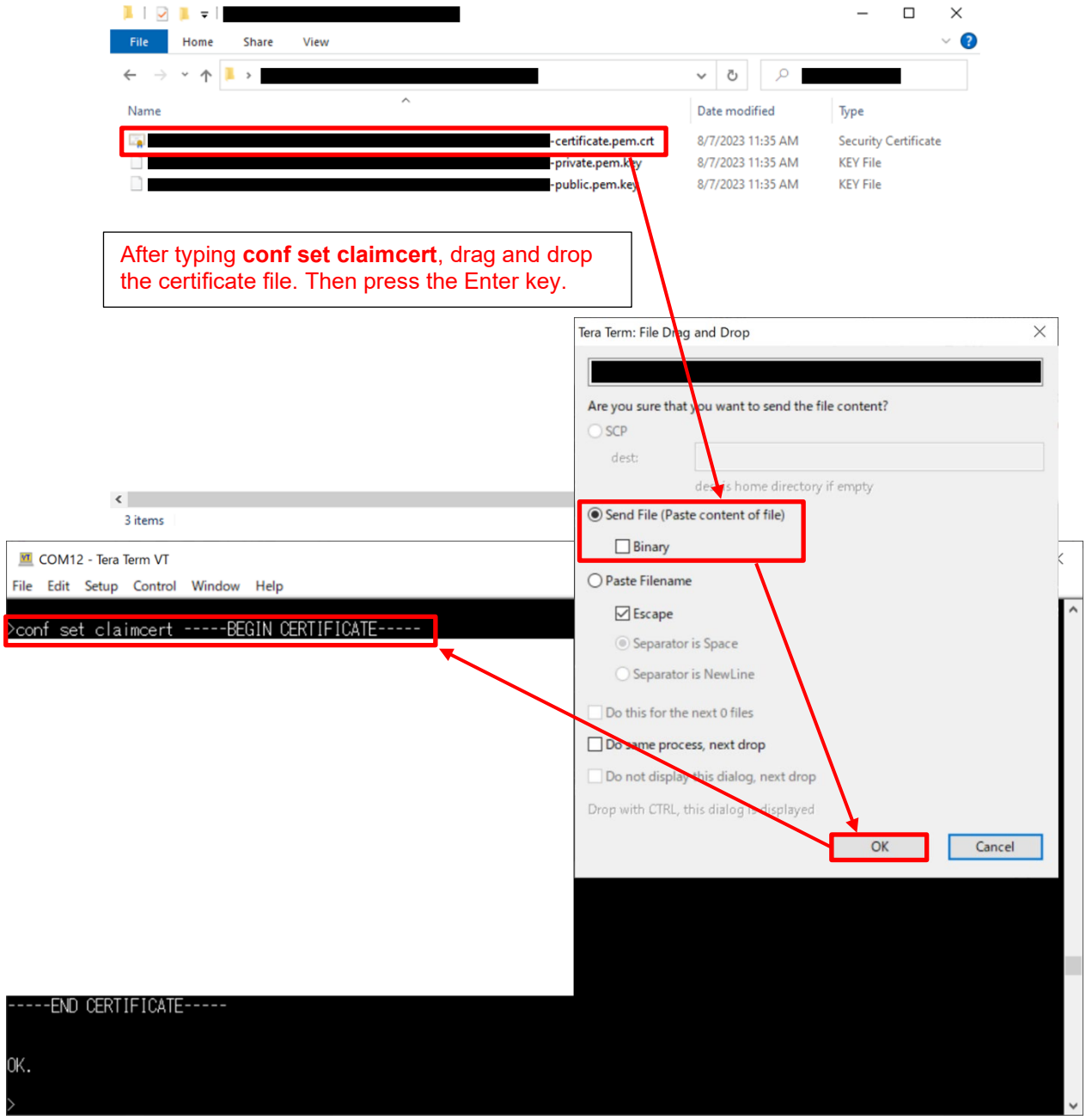


Figure 4.32 Entering Information Using CLI (5)

To enter the provisioning claim private key, type **conf set claimkey** in Tera Term. Next, drag and drop the provisioning claim private key file (**xxxx-private.pem.key**) created in 4.3.2 onto the Tera Term window (**Send File**). Finally, press the Enter key in Tera Term.

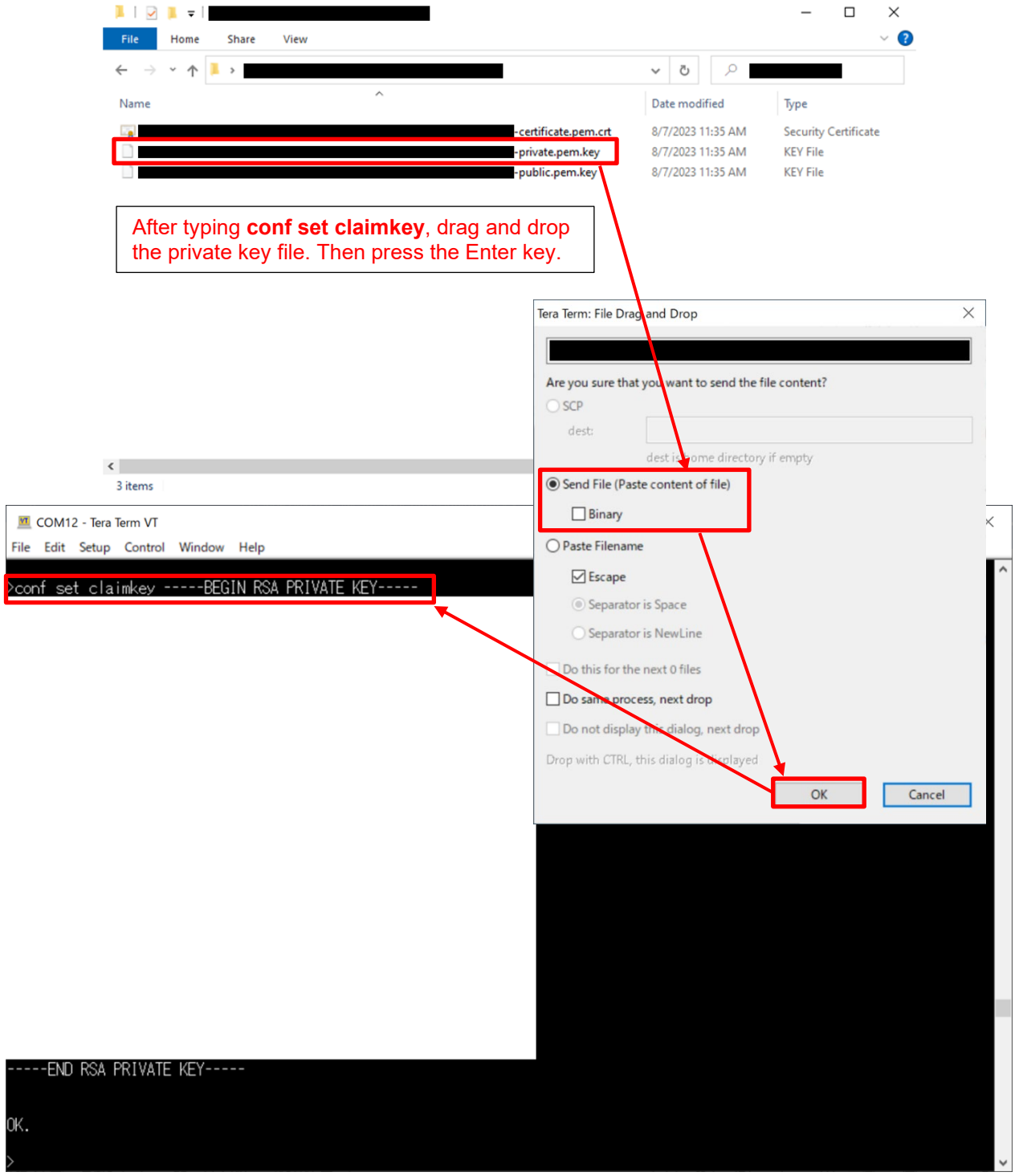


Figure 4.33 Entering Information Using CLI (6)

To store the information entered up to this point in the data flash memory, type **conf commit** in Tera Term and press the Enter key.

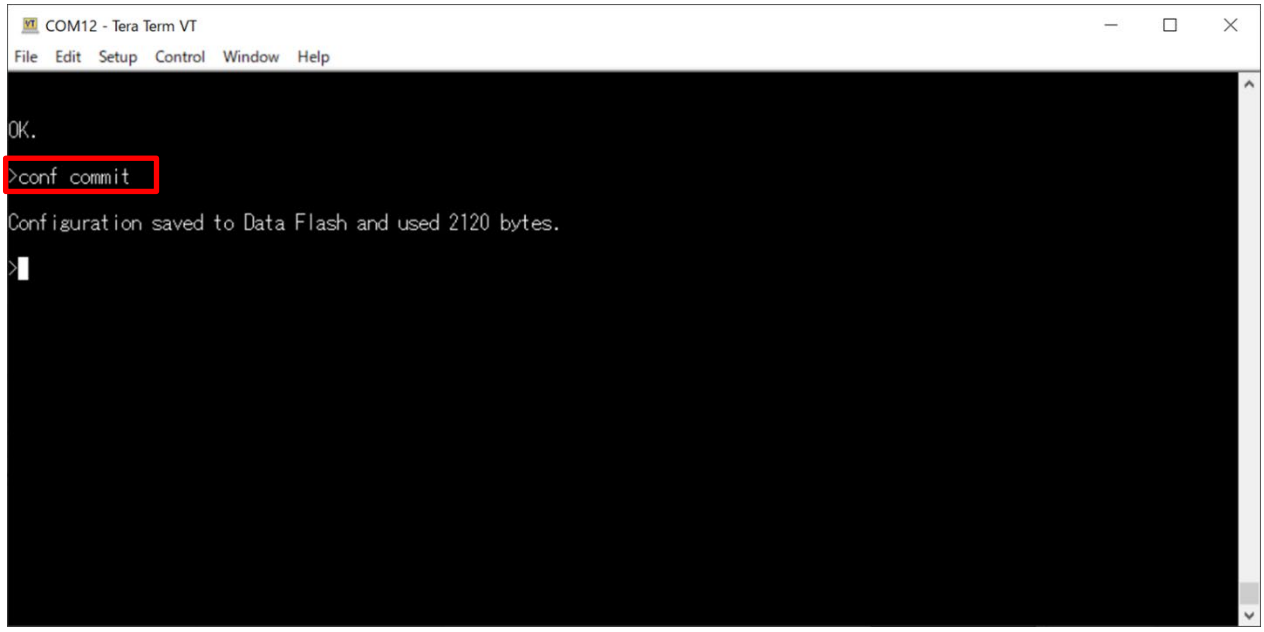


Figure 4.34 Entering Information Using CLI (7)

To start the demo, type **reset** in Tera Term and press the Enter key. If nothing is entered in Tera Term for 10 seconds after the reset, the demo starts.

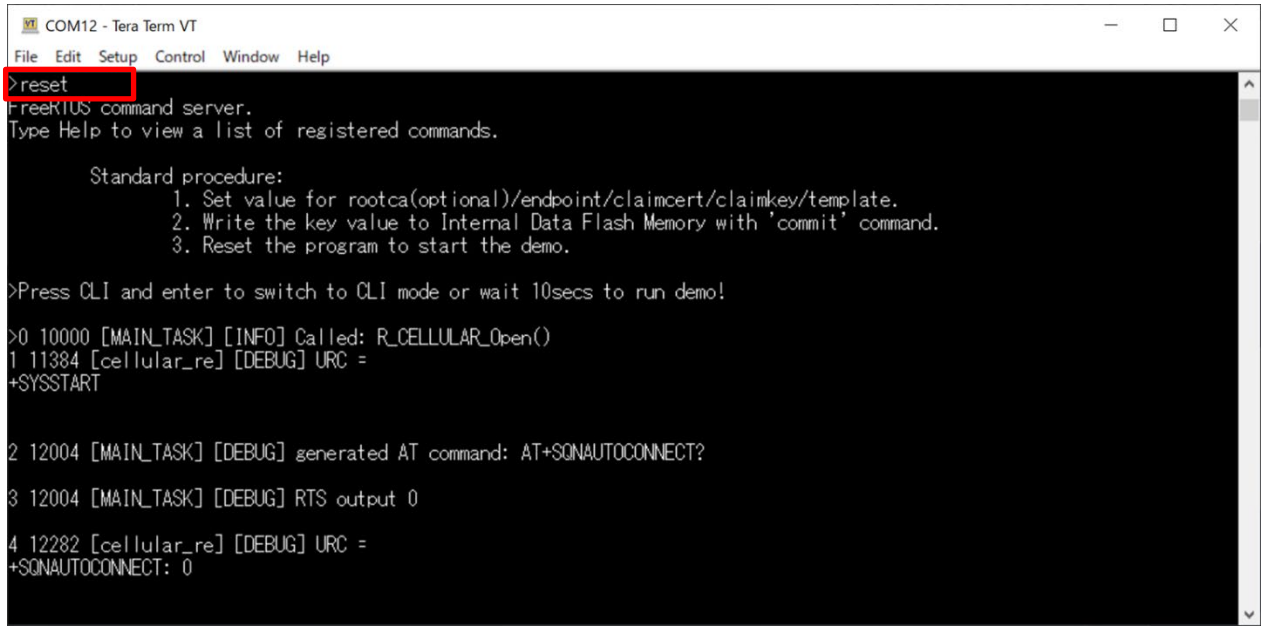


Figure 4.35 Entering Information Using CLI (8)

4.7 Confirming the Results of Running the Demo

Figure 4.36 shows a log file produced by running the fleet provisioning demo.

(The log is displayed in Tera Term.)

If the text string “Demo completed successfully.” appears at the end of the log, the fleet provisioning demo completed successfully. Successful completion of the demo means that a new thing has been registered on AWS IoT Core and an individual device certificate assigned to it.

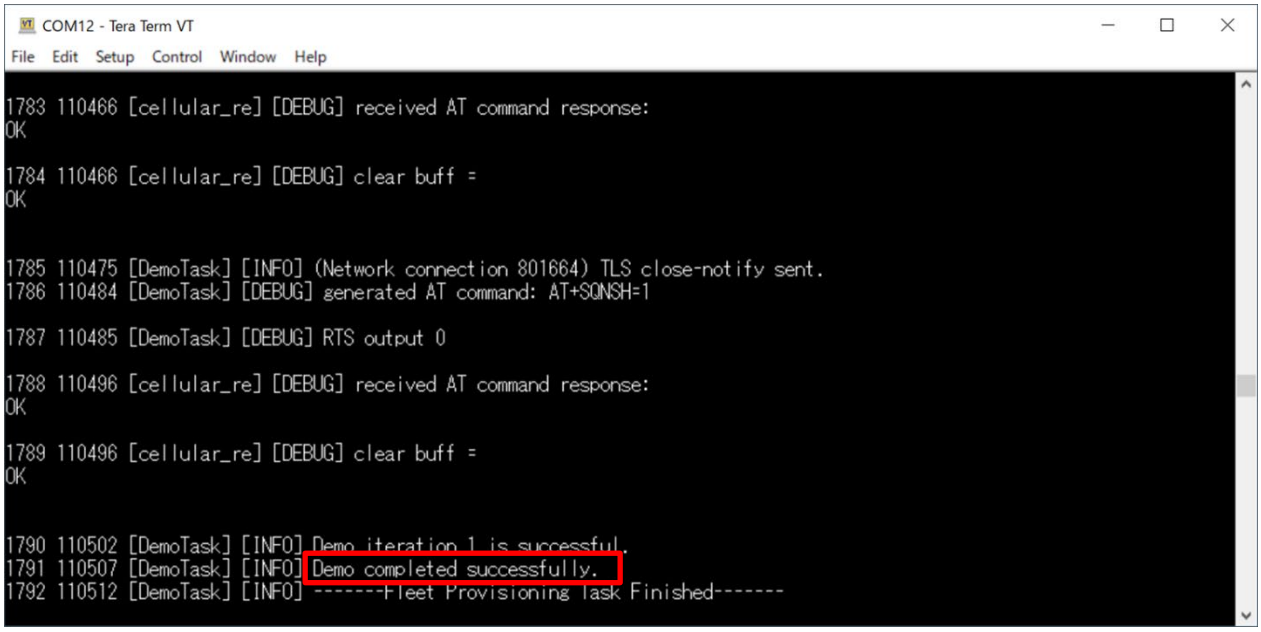


Figure 4.36 Log Produced when Fleet Provisioning Demo Completes Successfully

After running the fleet provisioning demo, you can use the individual device certificate and private key obtained from AWS to run the PubSub demo. Check to confirm that the text string “Successfully sent QoS 0 publish to topic:” appears in the log as shown in Figure 4.37.

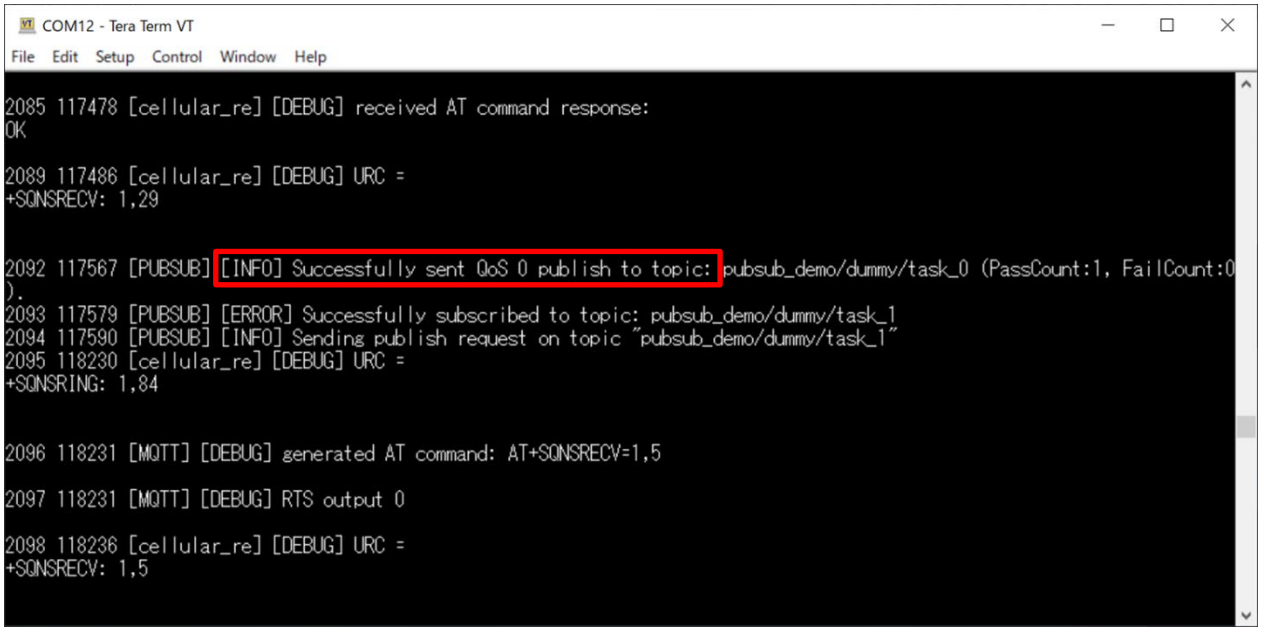


Figure 4.37 Log Produced when PubSub Demo Completes Successfully

You can also check MQTT messages sent to AWS from CK-RX65N by selecting **MQTT test client** from the AWS IoT console.

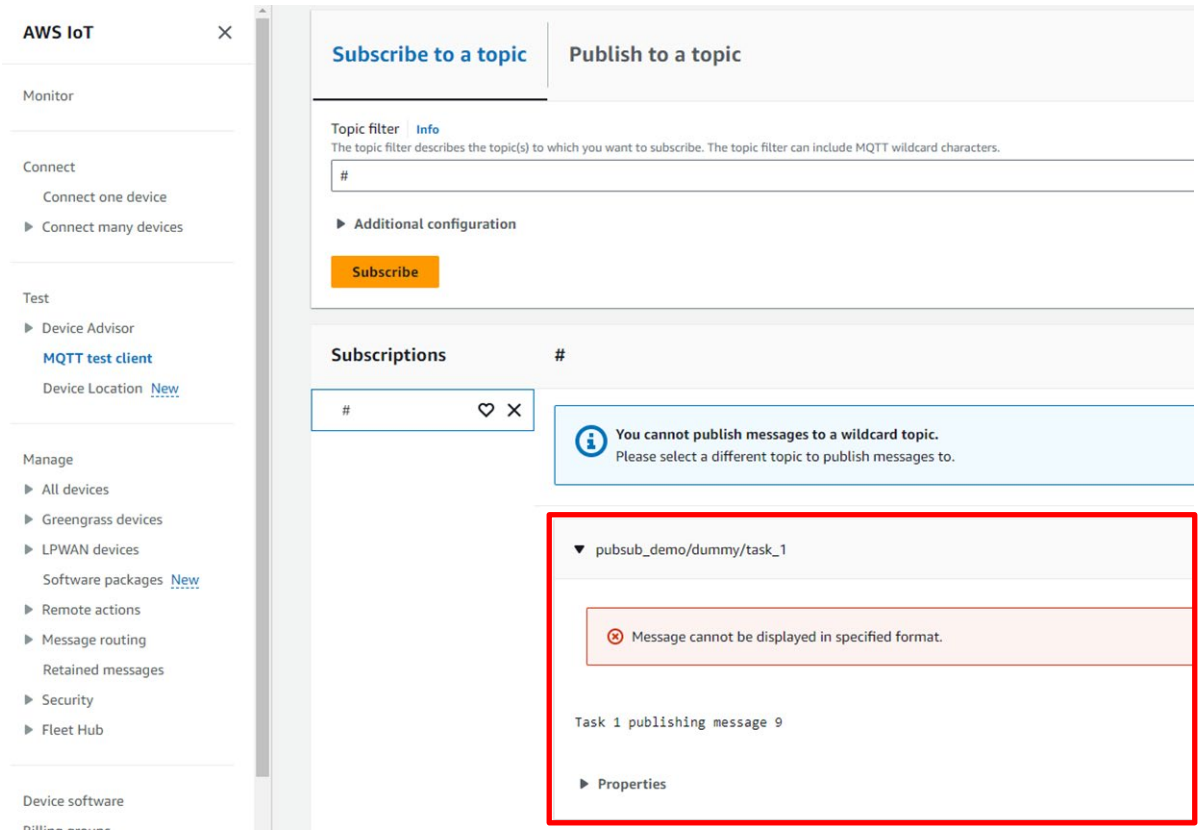


Figure 4.38 MQTT Test Client after Successful Completion of PubSub Demo

You can check on the thing registered by the fleet provisioning demo from the AWS IoT console.

Under **All devices**, select **Things**. The thing (shown as “FPDemoID_xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx” in Figure 4.39) will have been registered if the demo completed successfully.

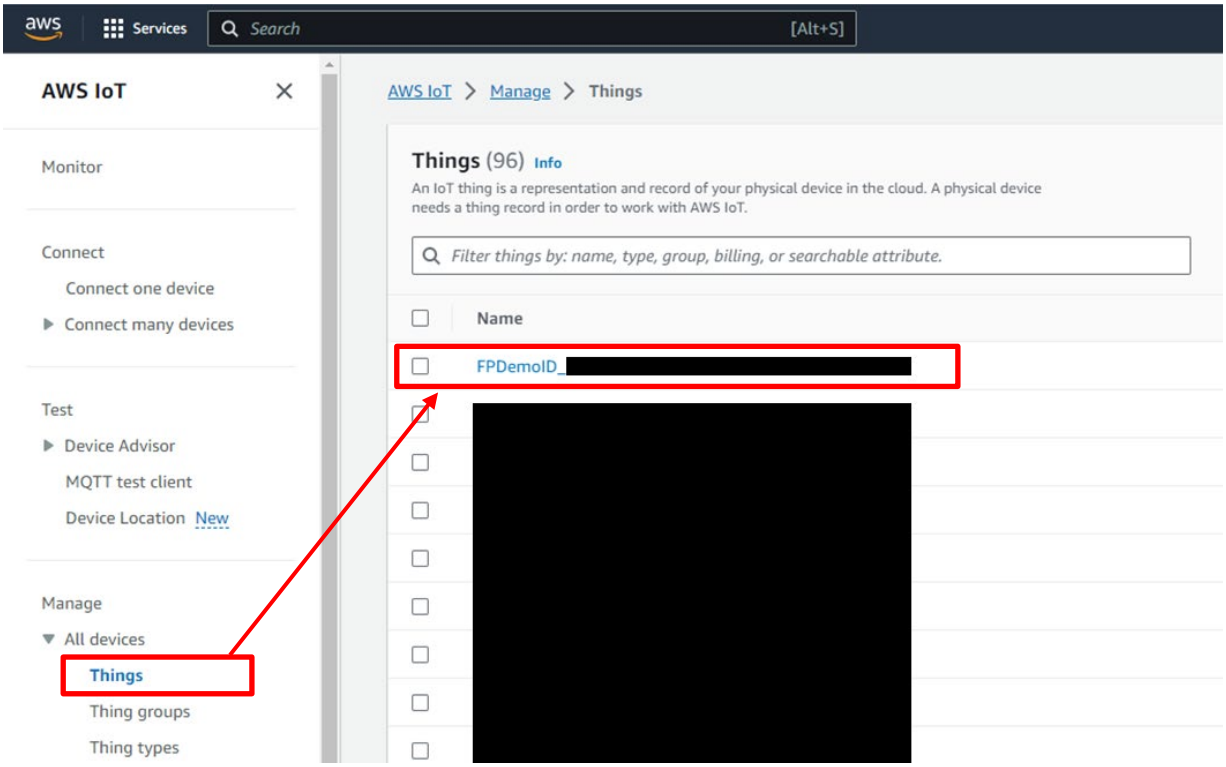


Figure 4.39 Confirming the Results of Running the Demo (1)

By checking the registered things, you can confirm that the individual device certificate generated and assigned by fleet provisioning (**Certificate ID** in Figure 4.40) has been attached and activated.

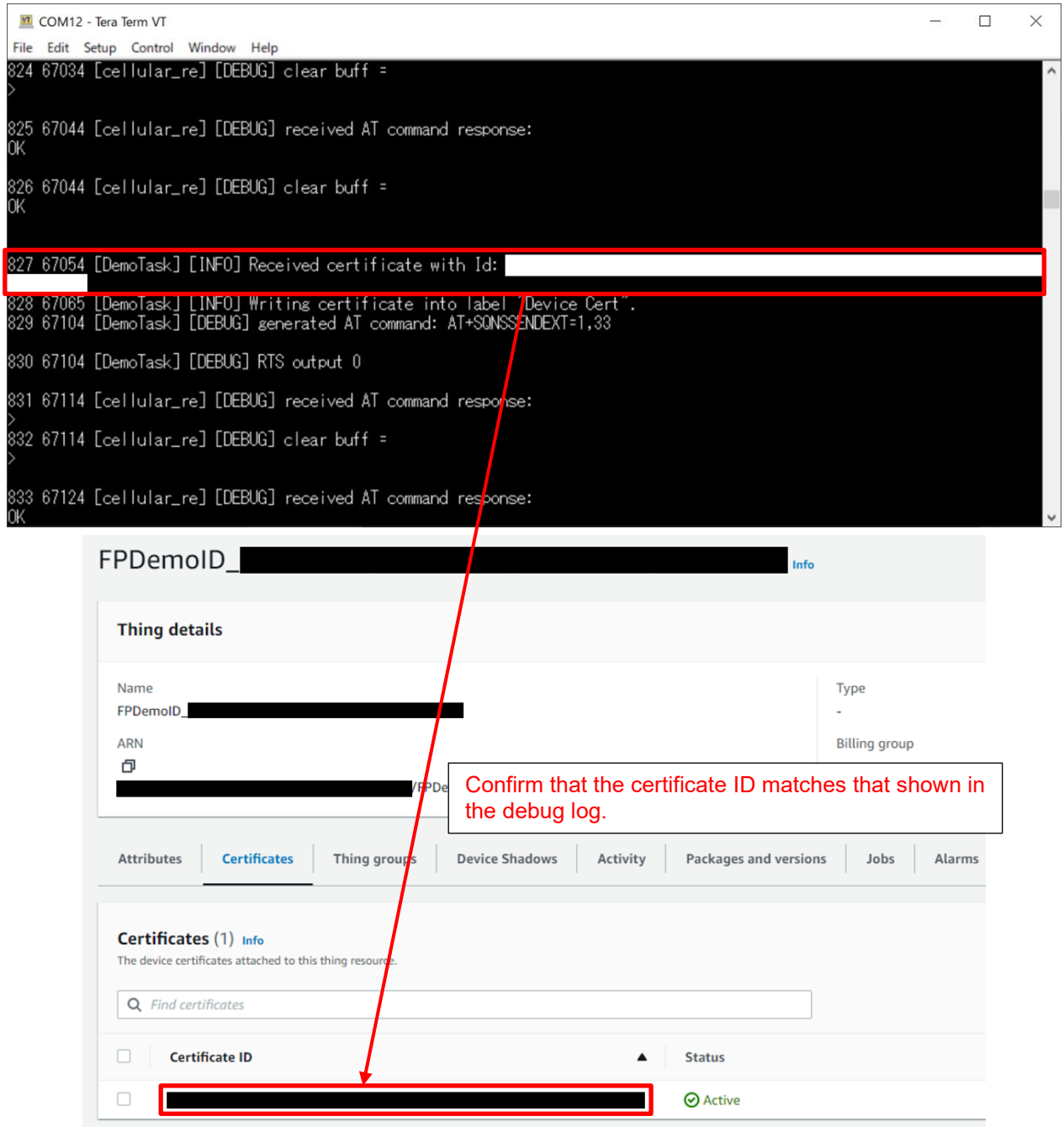


Figure 4.40 Confirming the Results of Running the Demo (2)

5. Conclusion

As mentioned earlier, there are multiple provisioning methods, and there are also various ways to enhance security. Nowadays, it is essential to select and deploy an appropriate provisioning method that matches the actual application in the target market, the scale of the system (number of devices), and the required level of security.

However, it is not a simple matter to maintain, manage, and operate a secure manufacturing facility in-house in order to implement provisioning functionality. This is why the fleet provisioning method had gained so much attention as an approach to the device provisioning process, and this is probably why market demand for this method is growing rapidly.

The provisioning method described in this document is only one example, so it will not satisfy the requirements of all users. Nevertheless, we think the information presented here will help deepen the reader's understanding of the advantages and disadvantages of deployment. It is our hope that this document will help users build convenient and practical production lines.

6. Websites and Support Information

AWS re:Post : https://repost.aws

Renesas FreeRTOS GitHub : https://github.com/renesas/iot-reference-rx

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 30, 2023	—	First edition issued
			Below, intentionally left blank.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.