DA16200/DA16600

Getting Started with AWS® IoT Core

The DA16200/DA16600 is a highly integrated ultra-low power Wi-Fi system on chip (SoC) that allows you to develop a complete Wi-Fi solution on a single chip. This document is a DA16200/DA16600 manual intended to help new or existing developers quickly get started using AWS® IoT Core.

Contents

Figures

Tables

1. Terms and Definitions

AP	Access Point
API	Application Programming Interface
AWS	Amazon Web Services
DPM	Dynamic Power Management
DTIM	Delivery Traffic Indication Map
IoT	Internet of Things
MCU	Micro-Controller Unit
OTA	Over the Air
SDK	Software Development Kit
TIM	Traffic Indication Map

2. References

[1] DA16200MOD, Datasheet, Renesas Electronics.

[2] DA16600MOD, Datasheet, Renesas Electronics.

[3] UM-WI-056, DA16200 DA16600 FreeRTOS Getting Started Guide, User Manual, Renesas Electronics.

[4] UM-WI-042, DA16200 DA16600 Provisioning Mobile App for Android/iOS, User Manual, Renesas Electronics.

Note 1 References are for the latest published version, unless otherwise indicated.

3. AWS IoT

The DA16200MOD/DA16600MOD is a full offload SoC for IoT applications such as security systems, door locks, and smart applications. This section provides procedures on how to configure AWS IoT for communicating with the DA16200/DA16600 IoT device.

3.1 Configure AWS IoT

This section describes how to set up requirements before using AWS IoT. To connect a device to the AWS IoT server, the following components are required:

1. Sign up AWS account and permissions.
2. Connect devices to AWS IoT.
3. Configure Amazon Cognito user pools and identity pools.
4. Set up Amazon IAM.
5. Create S3 bucket.

3.1.1 Sign Up for AWS Account

To create an AWS account and grant permissions:

1. Go to AWS website and create a free account (https://portal.aws.amazon.com/).
2. Create an administrative user for performing daily administrative tasks.
3. Open the AWS IoT console to get started with AWS IoT.

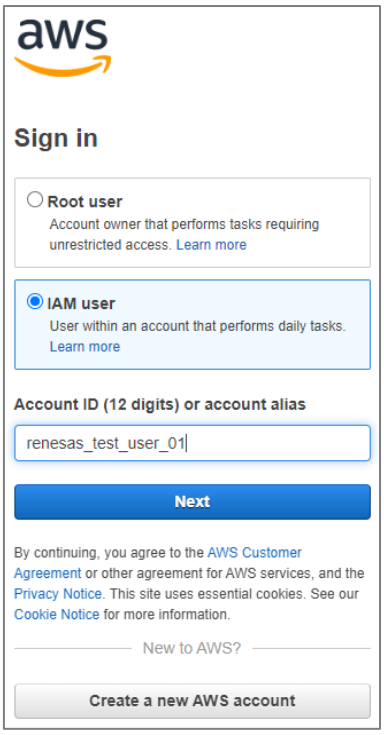

Figure 1. Sign up for AWS account

NOTE
If you do not have an AWS account, Renesas Electronics can provide a Thing name that has already been created for testing.

3.1.2 Connect Devices to AWS IoT

You can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of IoT solutions through AWS IoT console. Prior to sending data to and receiving data from AWS IoT server, you should register a device first.

3.1.2.1 Register a Device in Thing Registry

In the Thing Registry, the devices connected to the AWS IoT server are represented by Things. The Thing Registry allows keeping records of all devices that are connected to an AWS IoT account.

To register a device in the Thing Registry:

1. On the AWS IoT console, on the navigation pane, expand **Registry**.
2. Expand **All devices** and click **Things** > **Create things**.

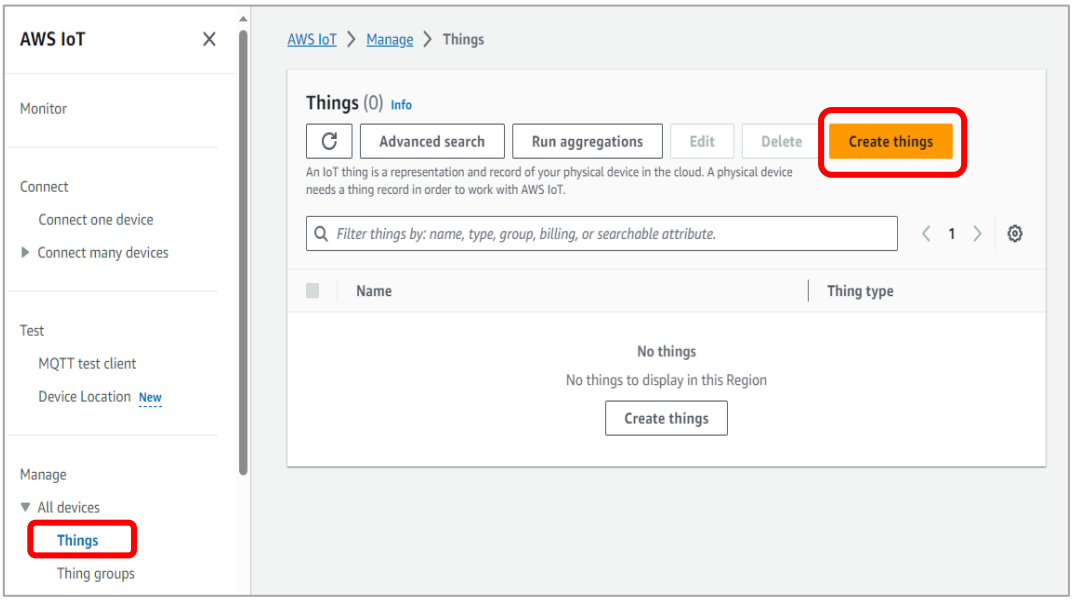

Figure 2. Register things

3. Select **Create single thing**.

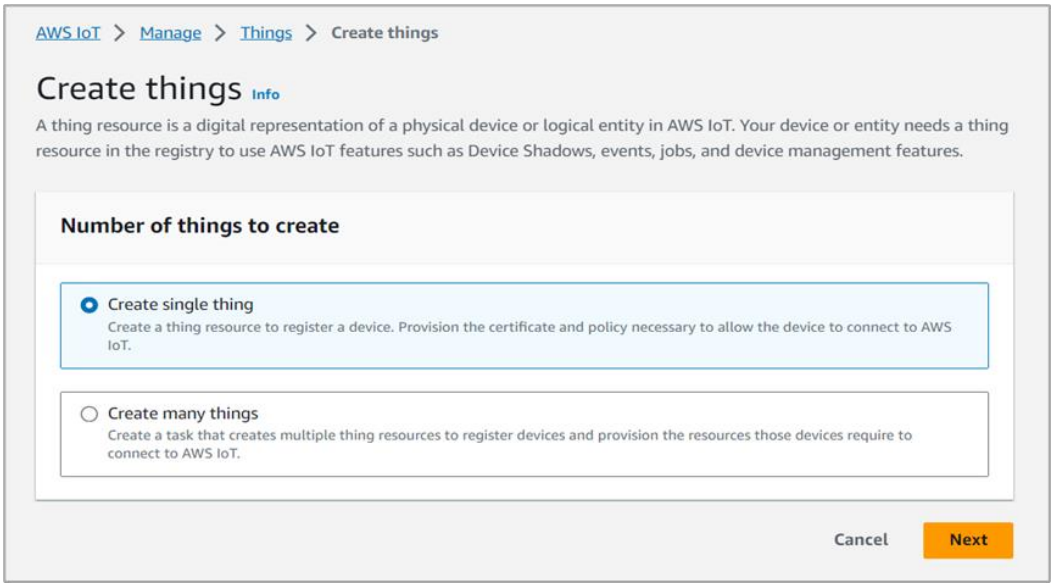

Figure 3. Create single thing

4. To add the device to the Thing Registry, in the **Thing name** field, enter a device name, for example, "MyTestDoorLock", and under **Device Shadow**, select **Unnamed shadow (classic)** and click **Next**.

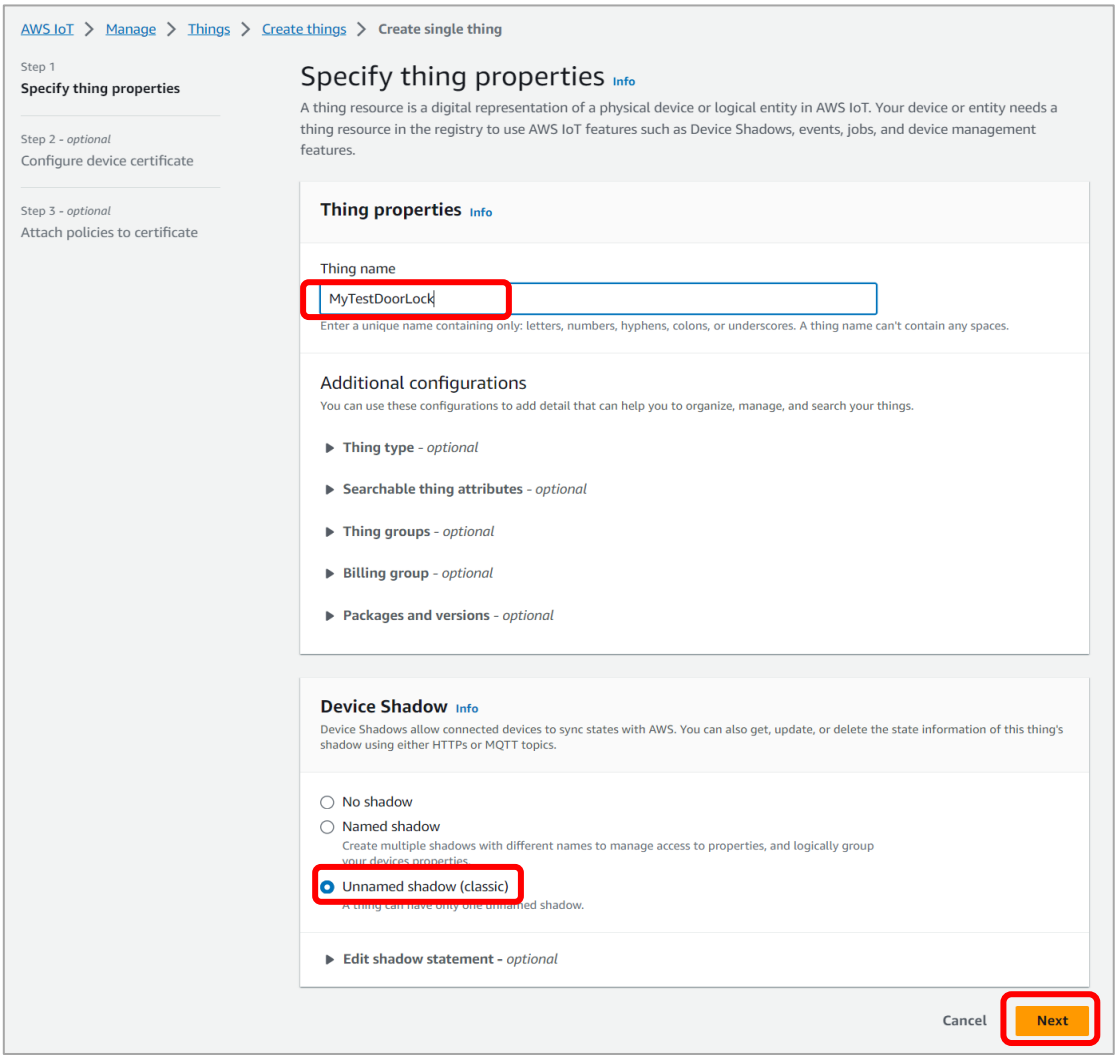

Figure 4. Thing name

5. Select **Skip creating a certificate at this time** and click **Create thing**.

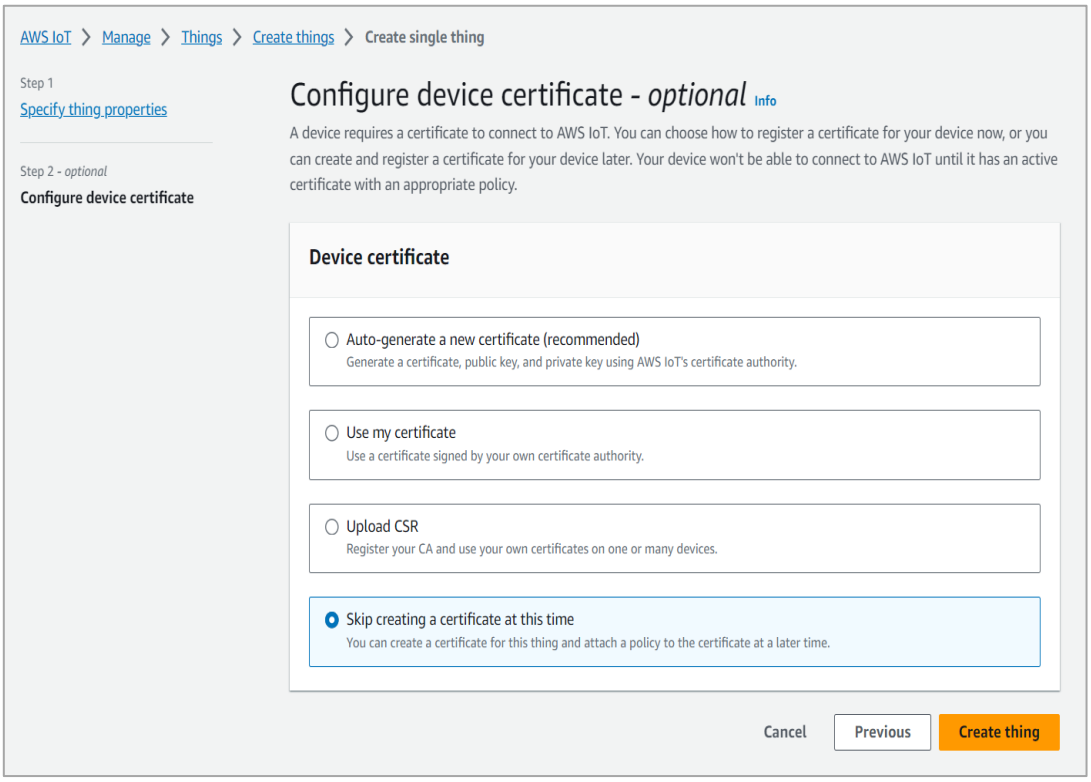

Figure 5. Thing without certificate

Now you have the thing created to perform the test and it is named **MyTestDoorLock**.

6. In the **Things** list, click the created thing.

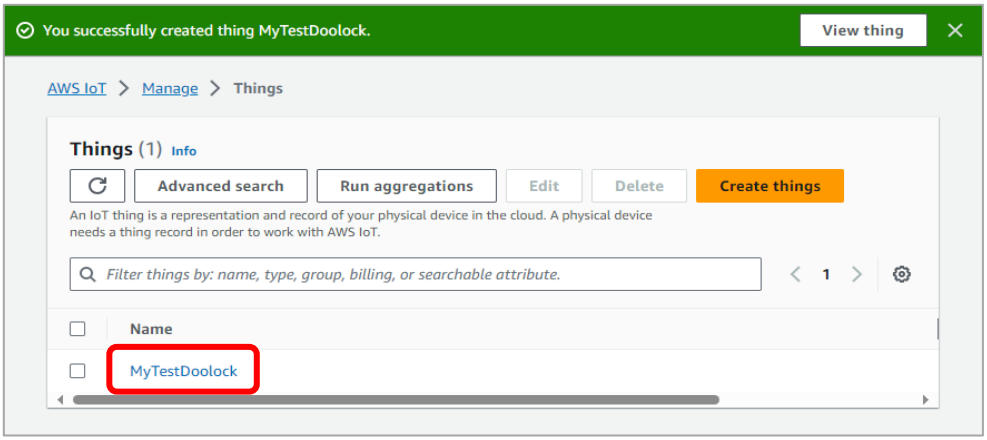

Figure 6. Created thing

Then the thing details appear.

7. For the shadow function of the thing, select the **Device Shadows** and click **Classic shadow**.

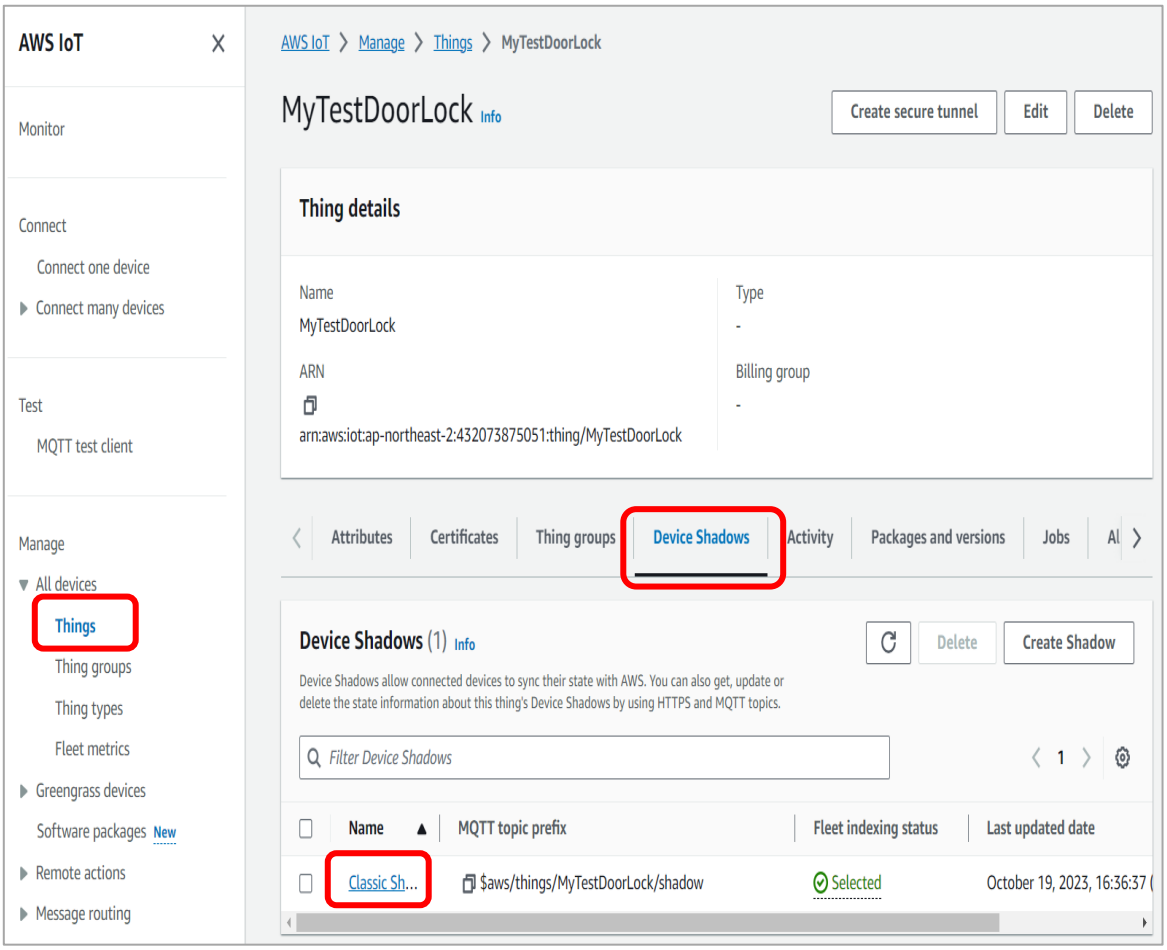

Figure 7. Classic shadow

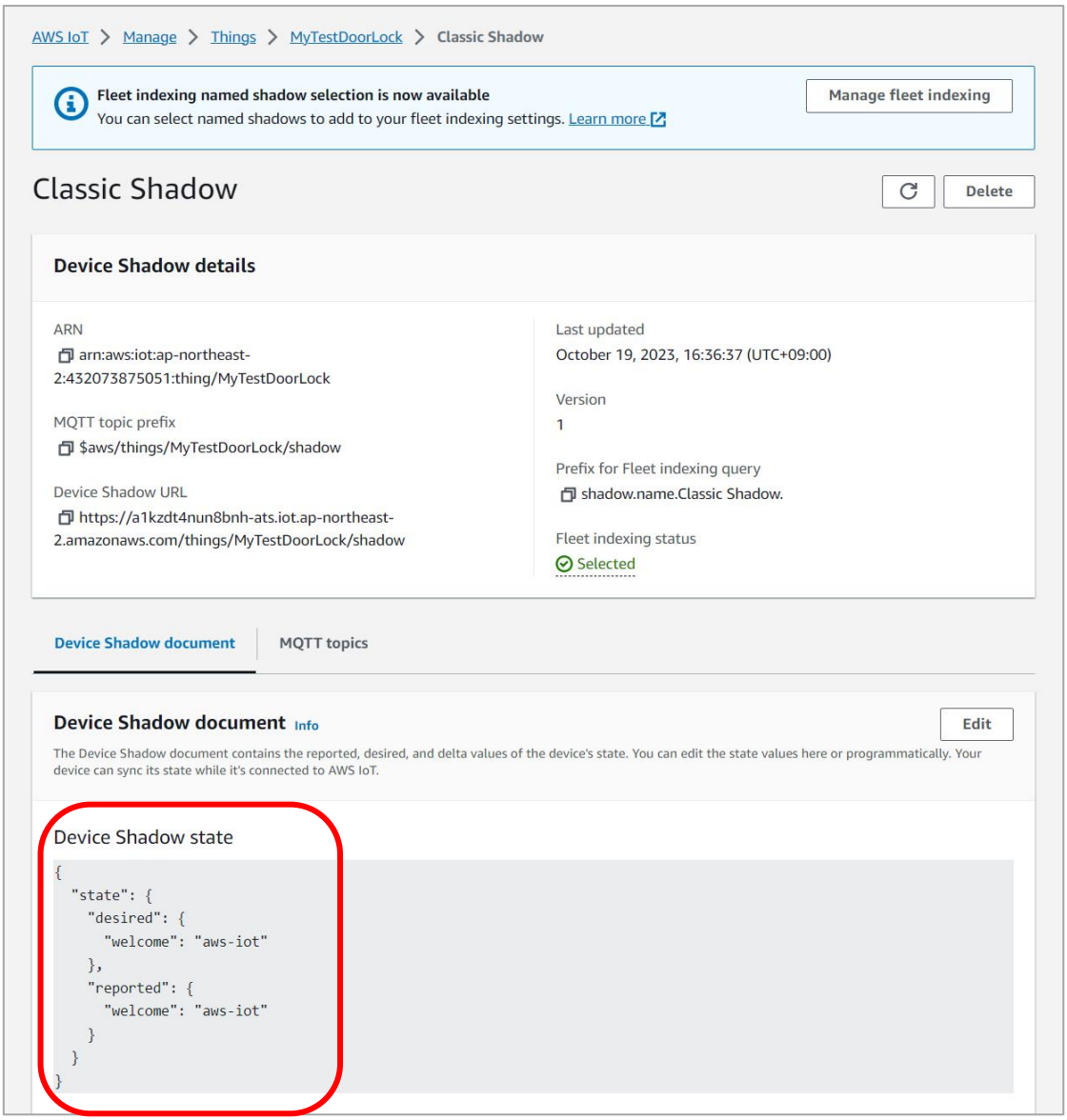

Figure 8. Device shadow document

For more information on device shadows for AWS IoT, visit AWS IoT Device Shadow service (https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html).

3.1.2.2 Create and Activate Device Certificate

The communication between the device and the AWS IoT web service is protected by X.509 certificates. You can let the AWS IoT generate a certificate or you can use your own X.509 certificate. This section shows that AWS IoT generates the X.509 certificate.

You should activate the certificates before use. To create and activate a device certificate:

1. On the navigation pane, expand **Security** and click **Certificates,** and then click **Add certificate** > **Create certificate**.

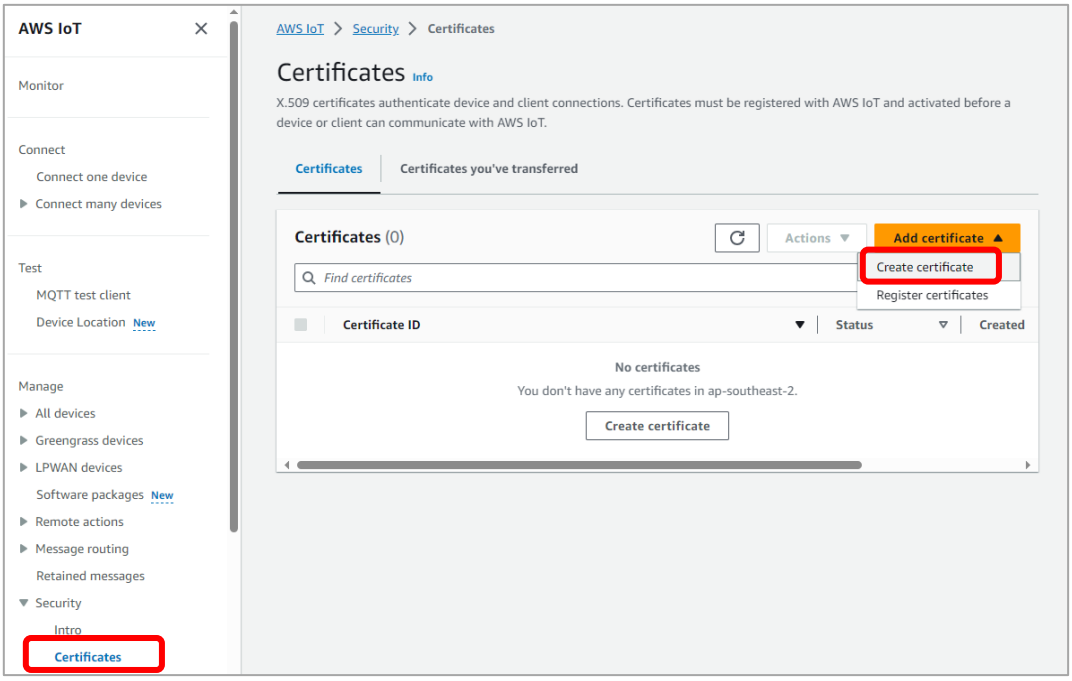

Figure 9. Create certificates

2. Select **Auto-generate new certificate (recommended)** and **Activate**, and then click **Create**.

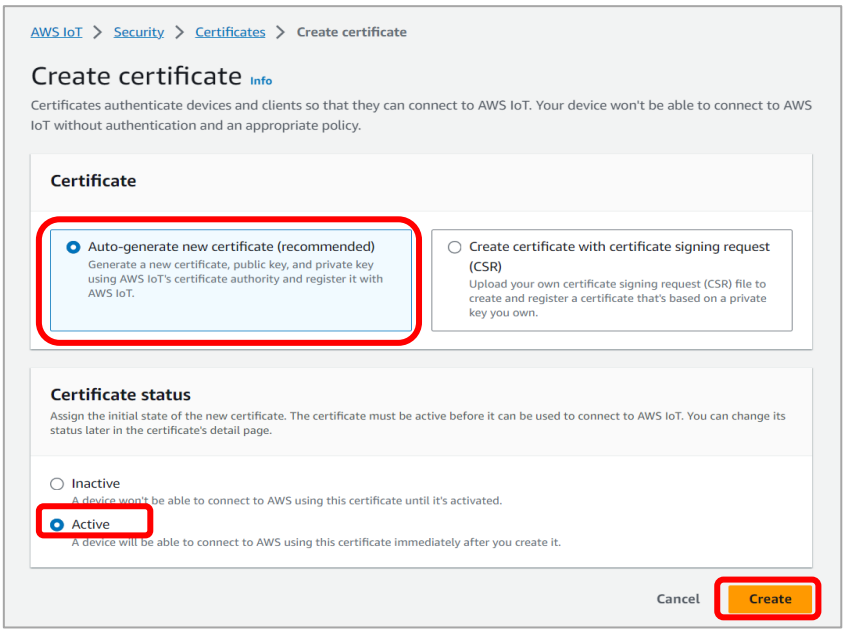

Figure 10. Create certificates (continued)

3. There are three required certificates to download.
   On the **Certificate Created** page, to download the device certificate, private key, and root CA certificates for AWS IoT, and then save the downloads to your computer, click **Download**.

NOTE
You must save the certificate files before leaving this page. After leaving this page in the console, you no longer have access to the certificate files. Renesas recommends that Device certificate, Private key file, and Root CA should be downloaded in sequential order.

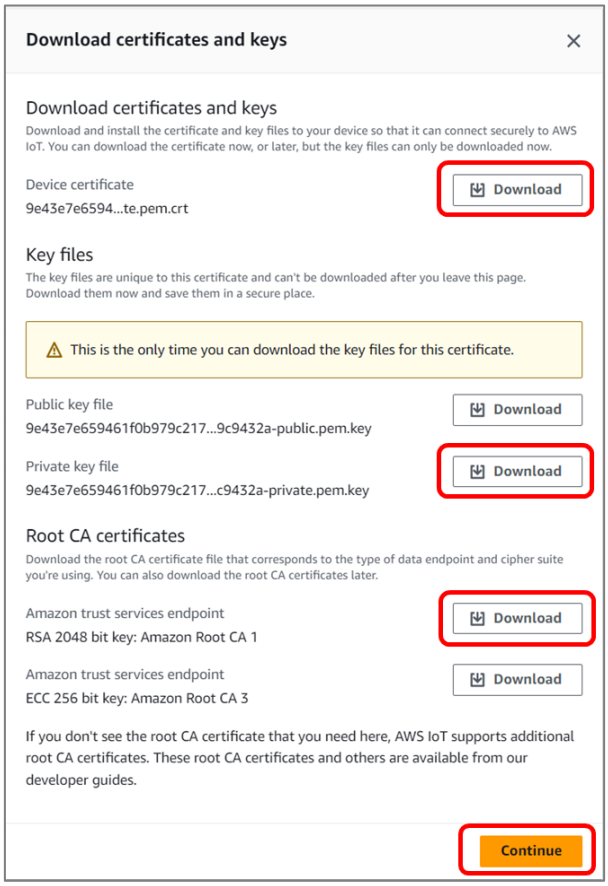

Figure 11. Download certificates and keys

For Root CA, visit the AWS Docs site (https://docs.aws.amazon.com/iot/latest/developerguide/server-authentication.html#server-authentication-certs). Root CA certificates are subjected to expiration and/or revocation.

The certificate status should be **Active** in the list of certificates.

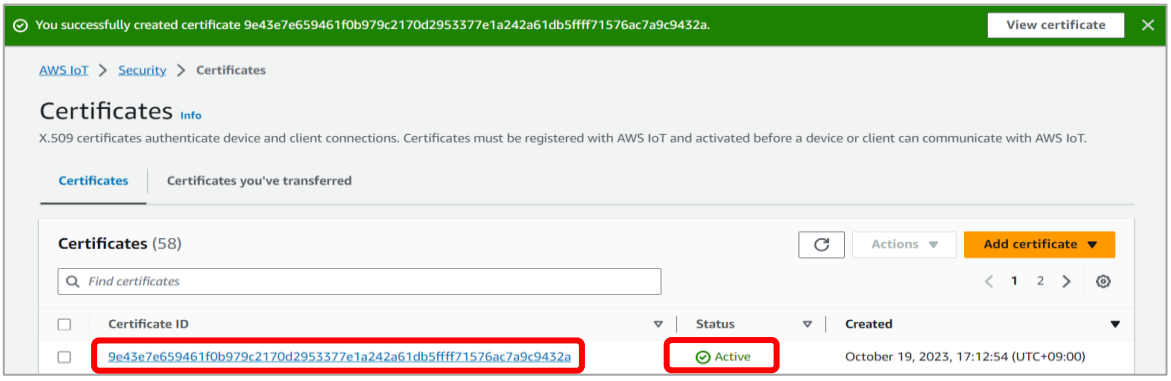

Figure 12. Activate certificate

3.1.2.3 Create Policy

The X.509 certificates are used to authenticate the device with the AWS IoT. The AWS IoT policies are used to authorize the device for AWS IoT operations, such as subscribing or publishing to MQTT topics. The device displays its certificate only while connecting to the AWS IoT.

To allow the device for AWS IoT operations, you should create an AWS IoT policy and attach that policy to the device certificate.

To create an AWS IoT policy:

1. On the navigation pane, expand **Security** and click **Policies** > **Create policy**.

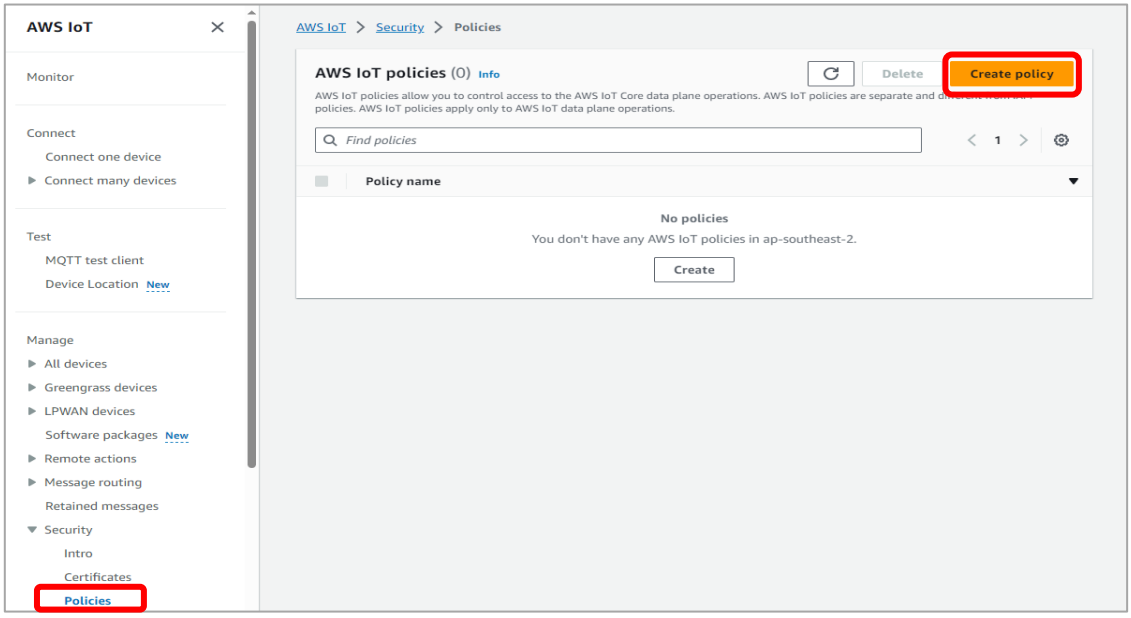

Figure 13. Create policy

2. On the **Create policy** page:
   a. In the **Policy name** field, under **Policy properties**, enter a name for the policy (for example, MyTestPolicy). Renesas strongly recommends not using personally identifiable information in policy names.

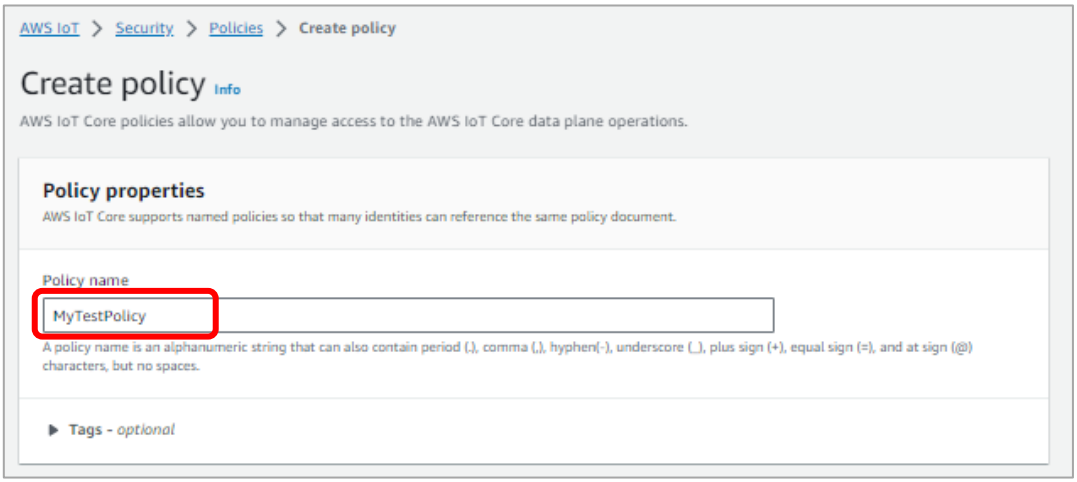

Figure 14. Add policy name

b. Under **Policy document**, select JSON, and then copy and paste the following JSON statement:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*"
    }
  ]
}
```

c. After entering the required information, click **Create**.

NOTE
The examples in this document are intended only for development environments. All devices in your production fleet must have credentials with privileges that authorize only intended actions on specific resources. The specific permission policies may vary depending on use cases. Identify the permission policies that best meet the business and security requirements. For more information, see Example Policies and Security Best practices in AWS IoT.

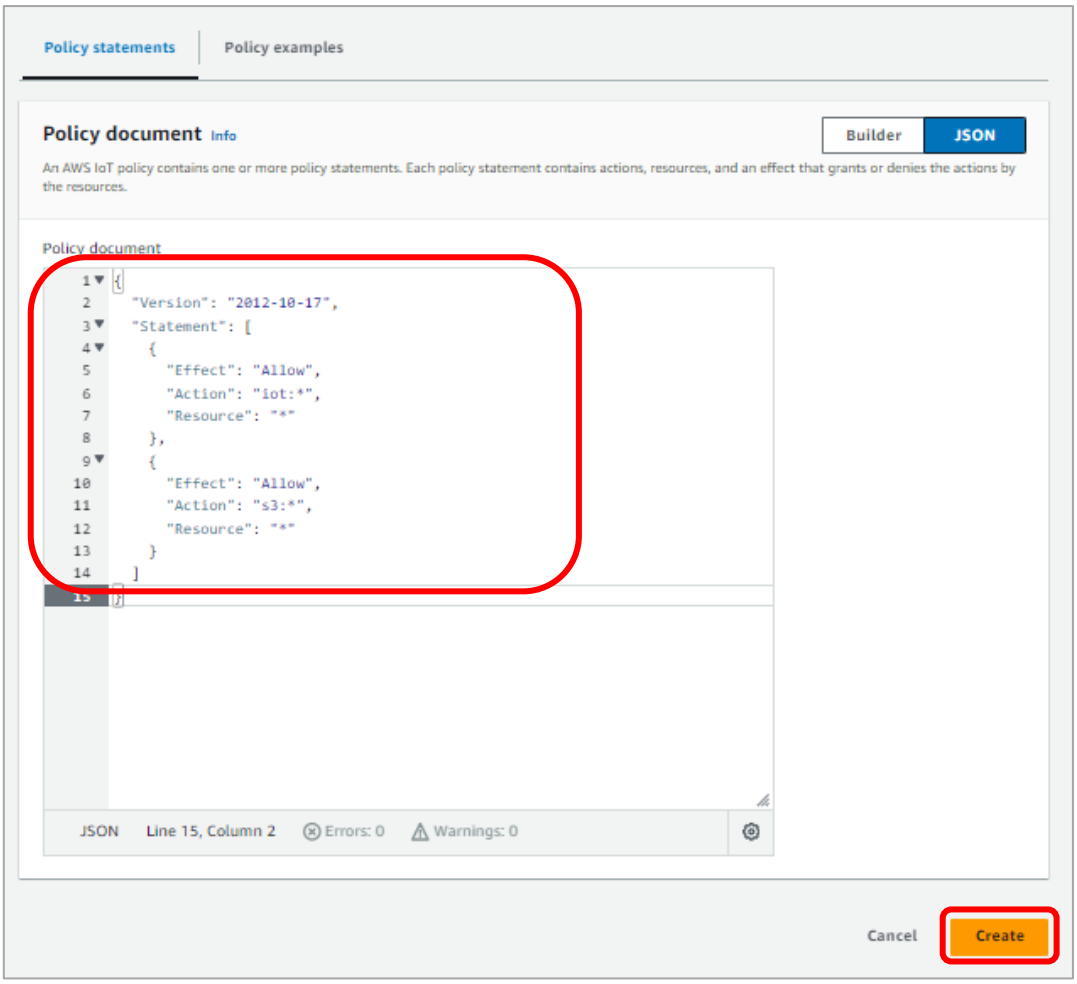

Figure 15. Enter JSON policy statement

3. To view the created policies, expand **Security** and click **Policies**.

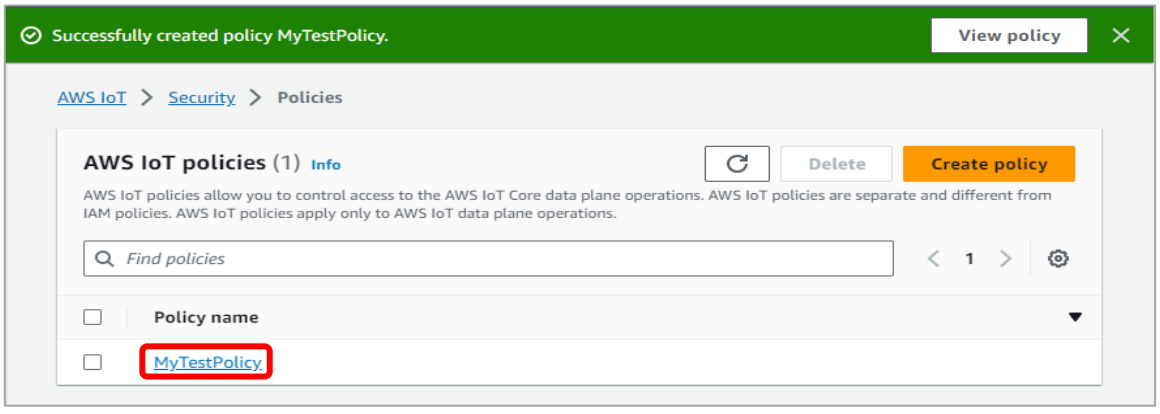

Figure 16. Created policy

4. Click the policy to view the details. Figure 17 shows an example of the selected policy content.

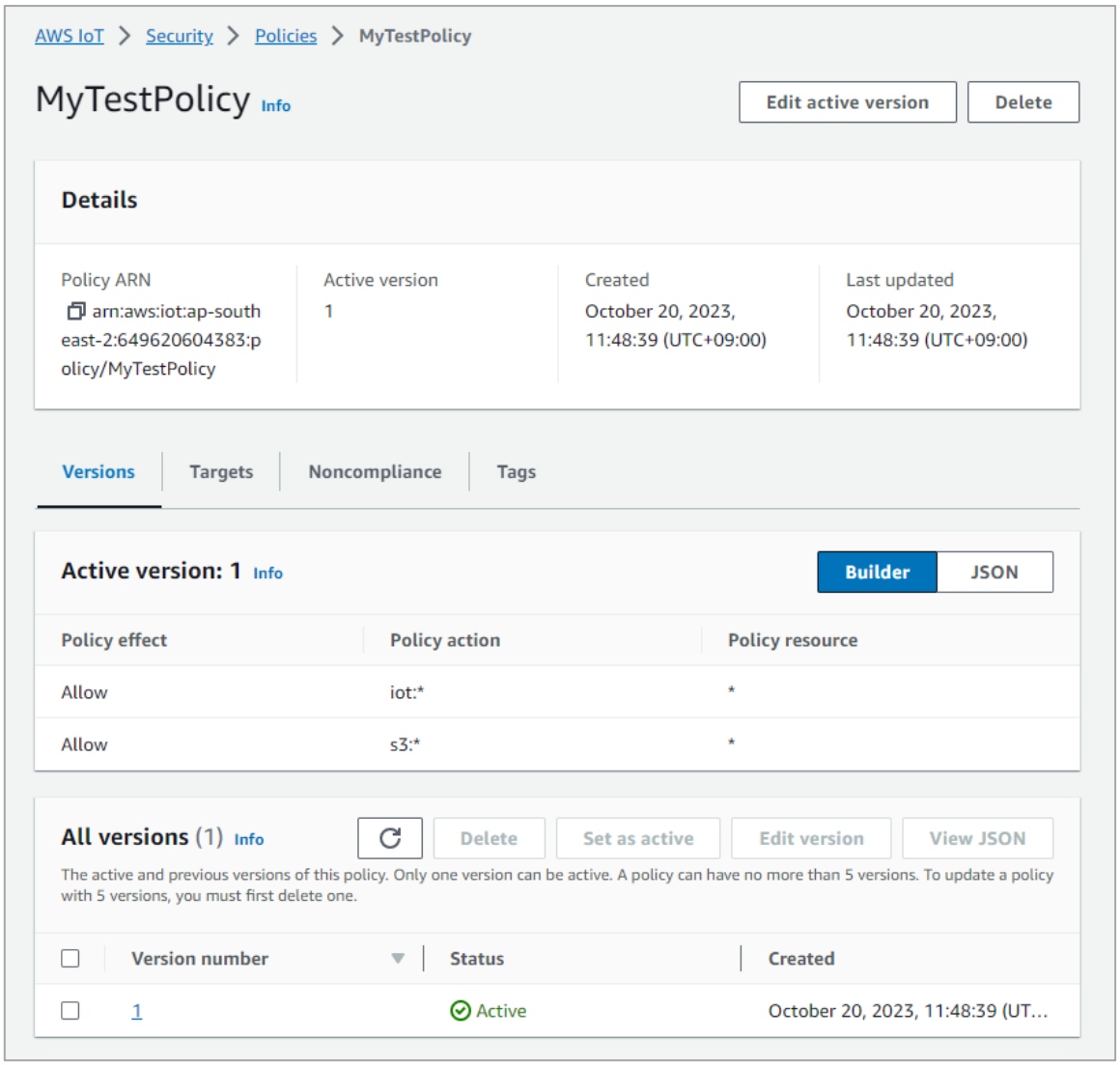

Figure 17. Check created policy

3.1.2.4 Attach Certificate to Thing and Policy

After an AWS IoT policy is created, you must attach that policy to the device certificate. The attachment of an AWS IoT policy to a certificate gives the device the permissions that are specified in the policy.

To attach the AWS IoT Policy to a device certificate:

1. Go to the certificate created by you, select **Policies,** and click **Attach policies**.

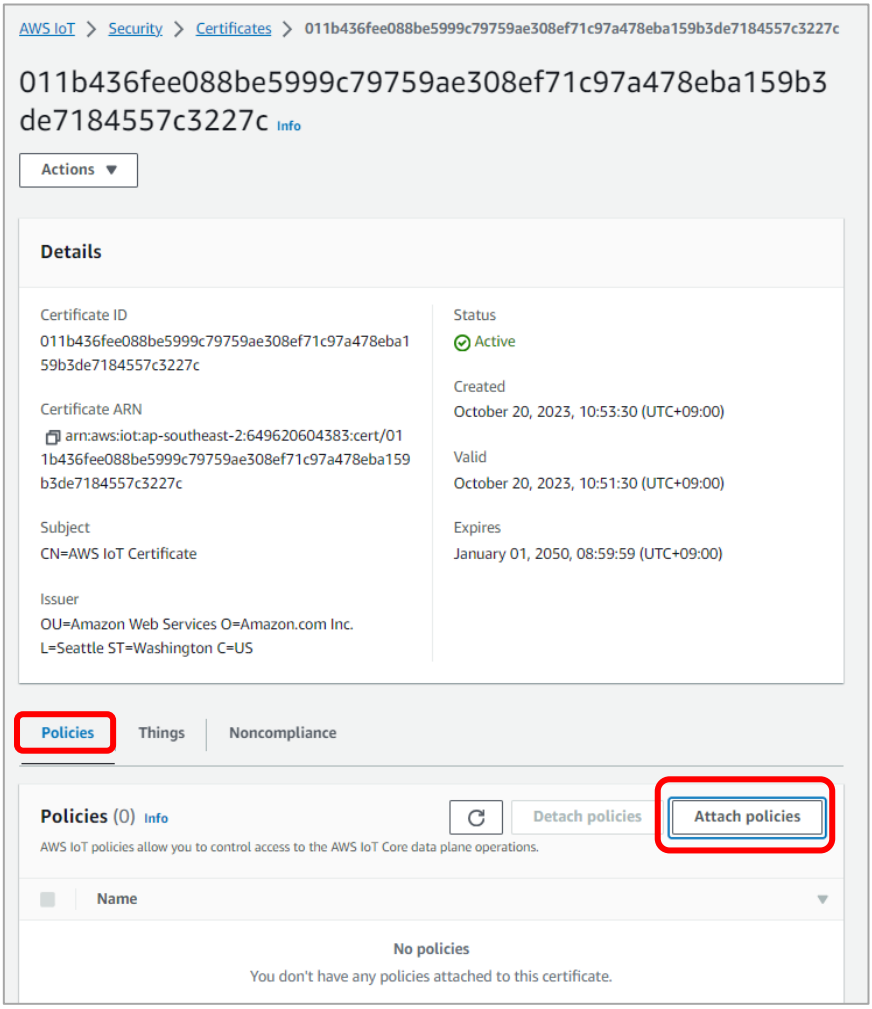

Figure 18. Policies

2. Select the created policy and click **Attach policies**.

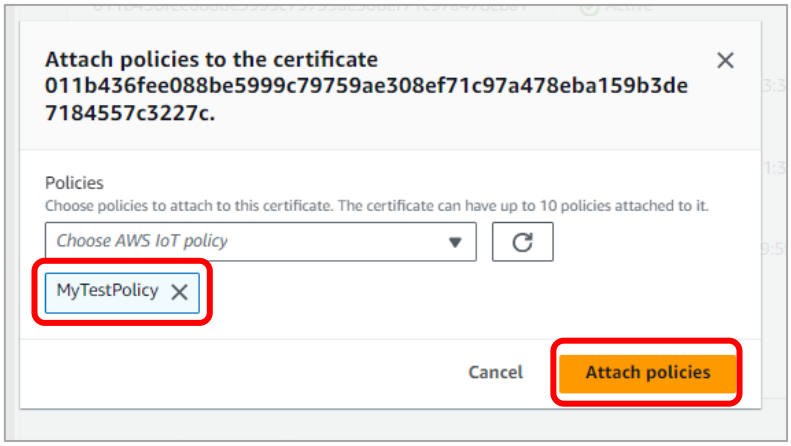

Figure 19. Attach policy

NOTE
A device should have a certificate, private key, and root CA certificate to authenticate with the AWS IoT. Renesas recommends that you attach the device certificate to the thing that represents the device in AWS IoT. This allows you to create AWS IoT policies that grant permissions based on certificates attached to things.

3. Go to the certificate created by you, select **Things** and click **Attach to things**.

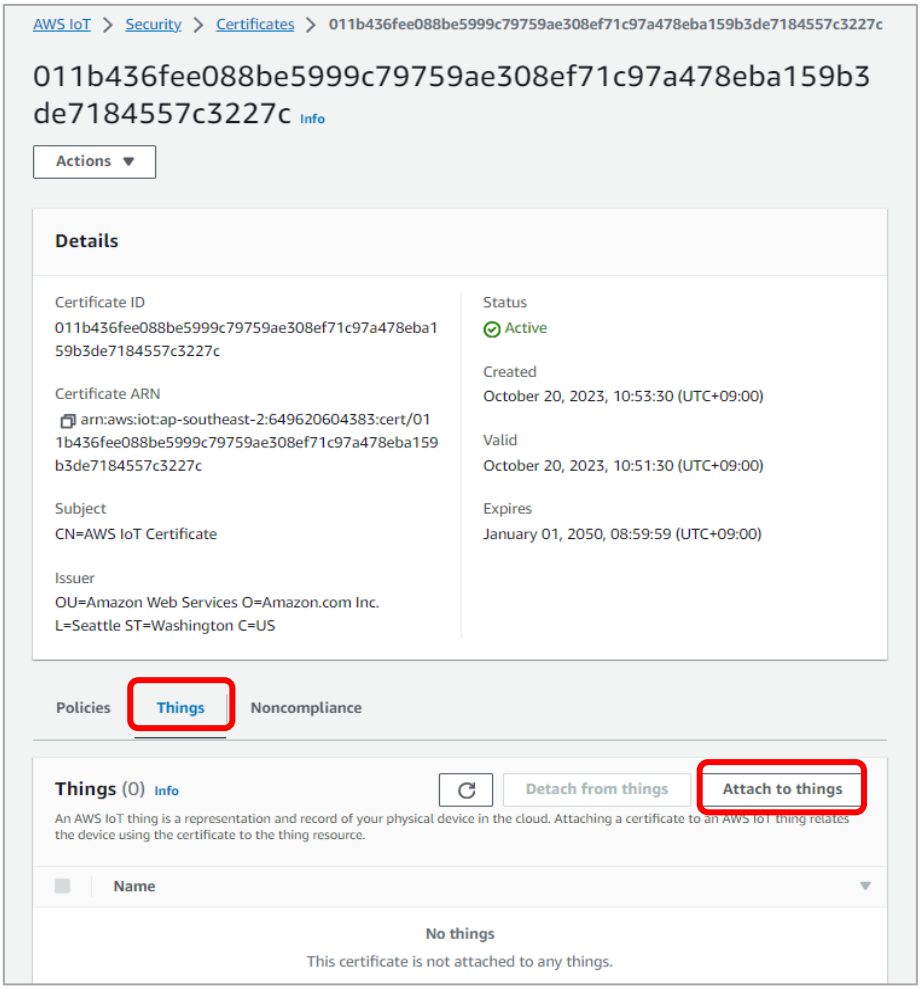

Figure 20. Attach things to certificate

4. Select the box of the thing that was created and click **Attach to thing**.

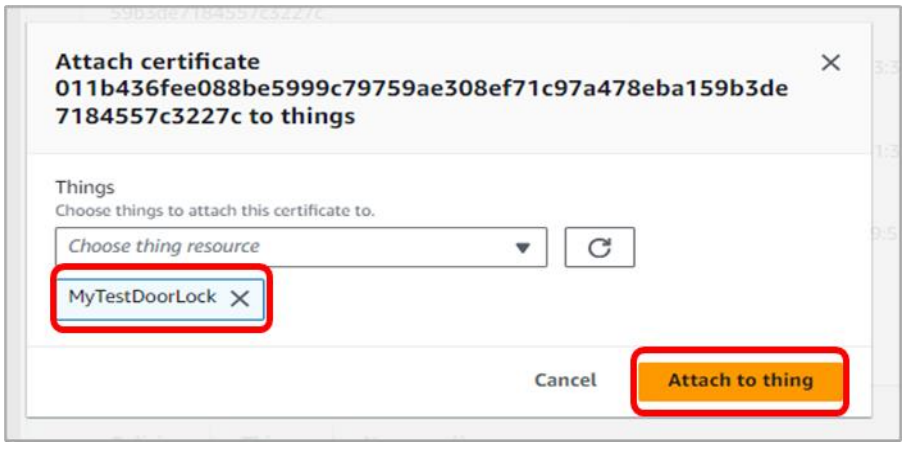

Figure 21. Attach to thing

3.1.2.5 Store Events in S3 Bucket

To store log files for the Door lock:

NOTE
On how to create Amazon S3 bucket, see Section 3.1.5.

1. Select **AWS console** > **AWS IoT Core**, expand **Message routing** and click **Rules** > **Create rule**.

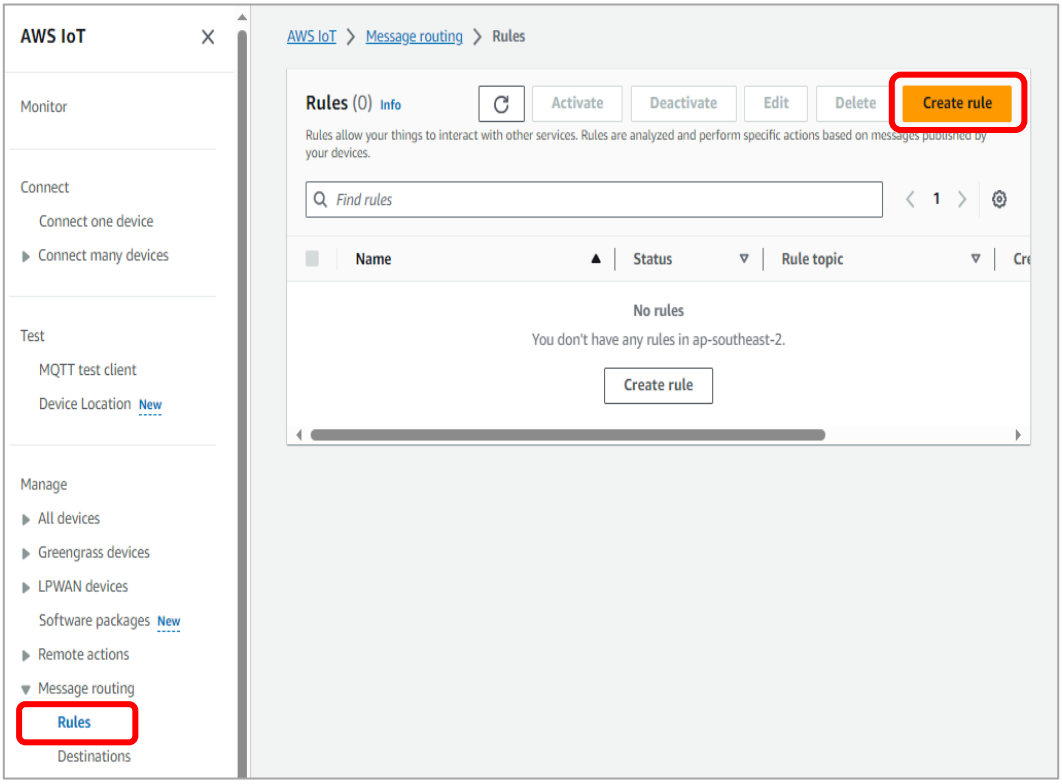

Figure 22. Create rule

2. Enter a rule name and click **Next**.

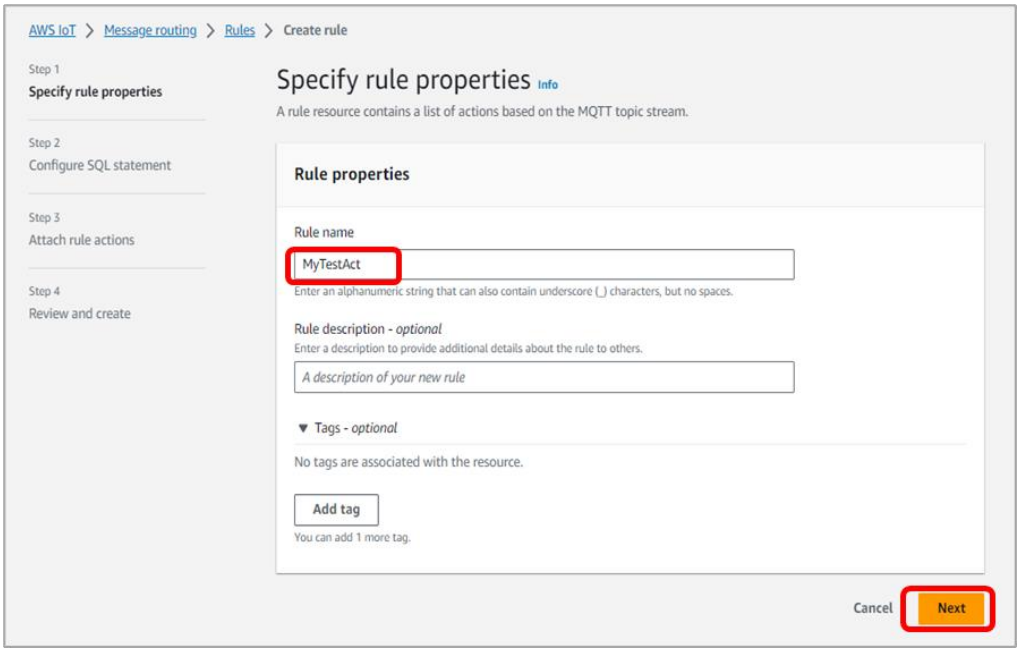

Figure 23. Specify rule name

3. Copy and paste the following SQL statement in the SQL statement box and click **Next**.

```
SELECT * FROM '$aws/things/Yourthingname/shadow/update'
WHERE state.reported.doorStateChange > 0 OR state.reported.temperature > 70 OR
state.reported.doorBell > 0
```

Note that the thing name is now **MyTestDoorLock**.

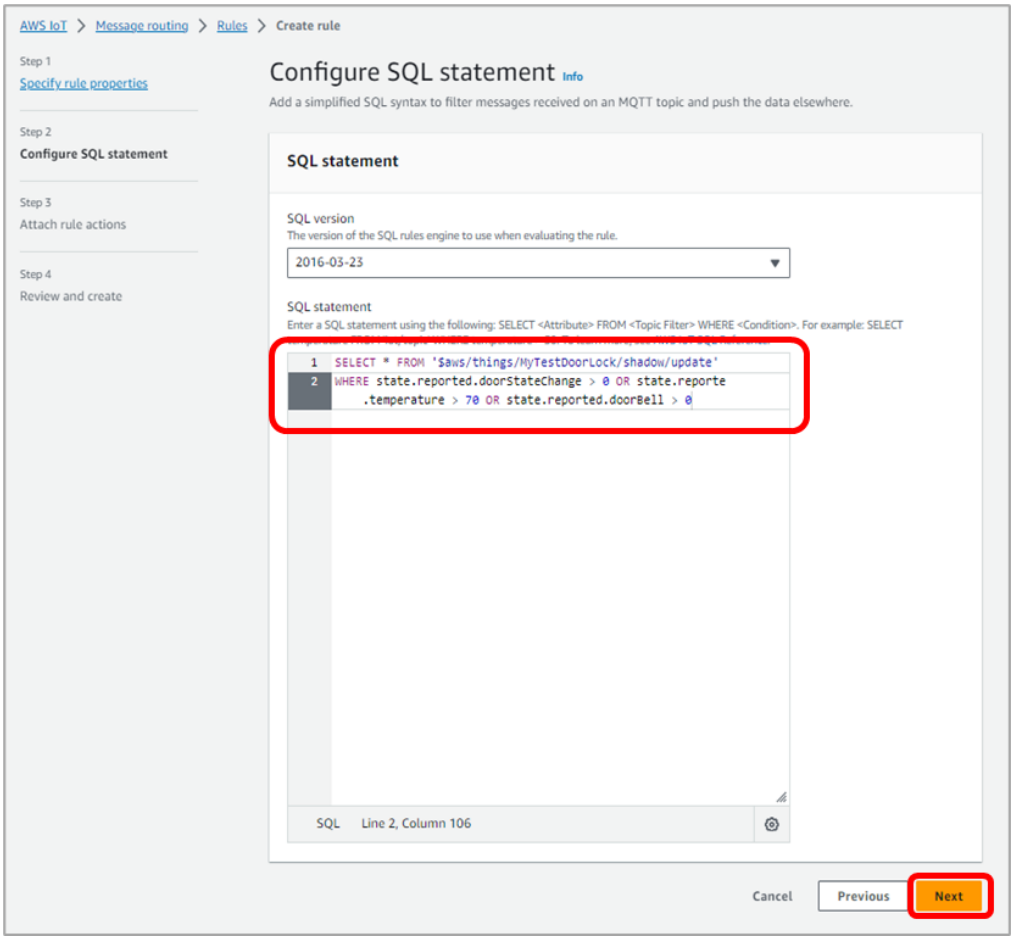

Figure 24. Configure SQL statement

4. Under **Rule actions**, in the **Action 1** list, select **S3 bucket**.

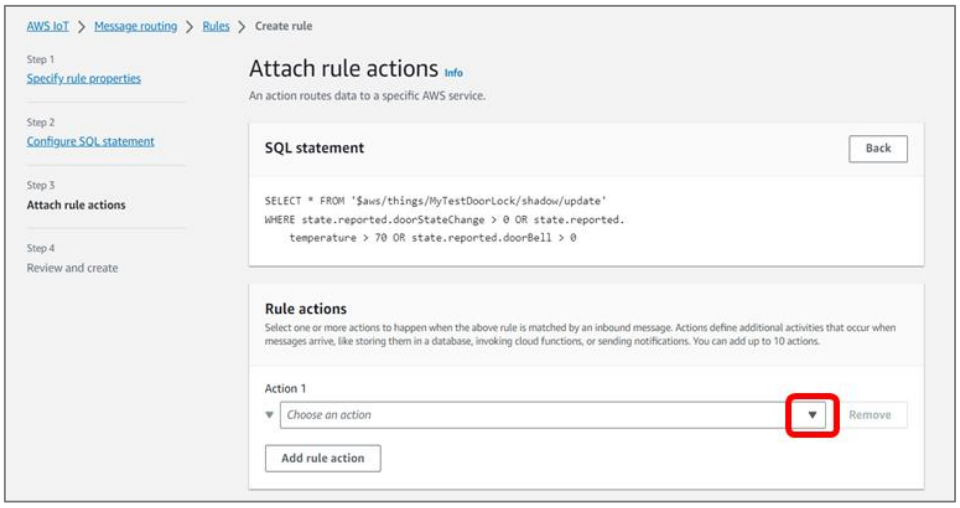

Figure 25. Attach rule actions

5. Click **Browse S3** and in the Key field, enter **${timestamp()}**. Then, click **Create new role**.

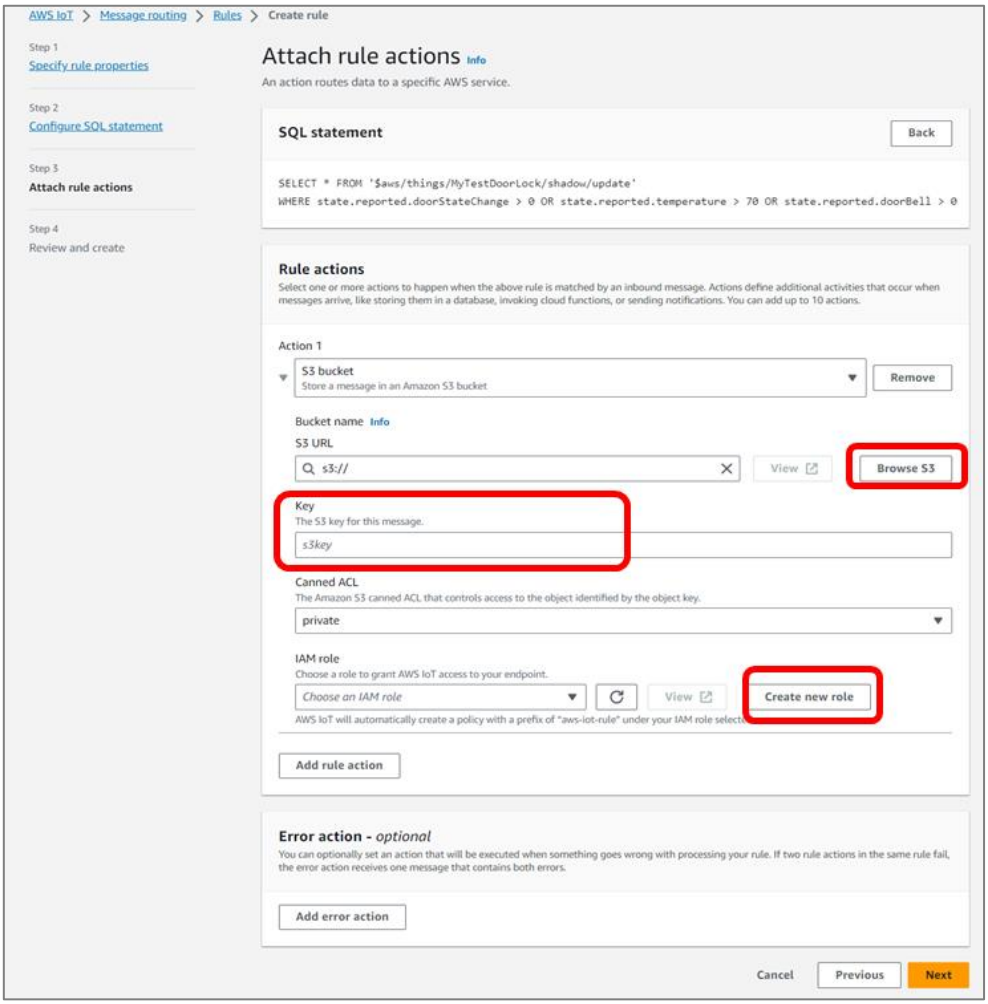

Figure 26. Attach rule actions (continued)

6. In the **Role name** field, enter an IAM role name and click **Create**.

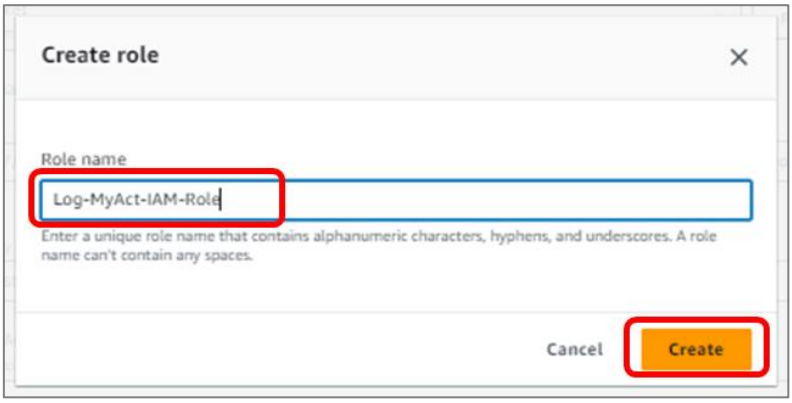

Figure 27. Create IAM role to save log files

7. Review the entered information and click **Create**.

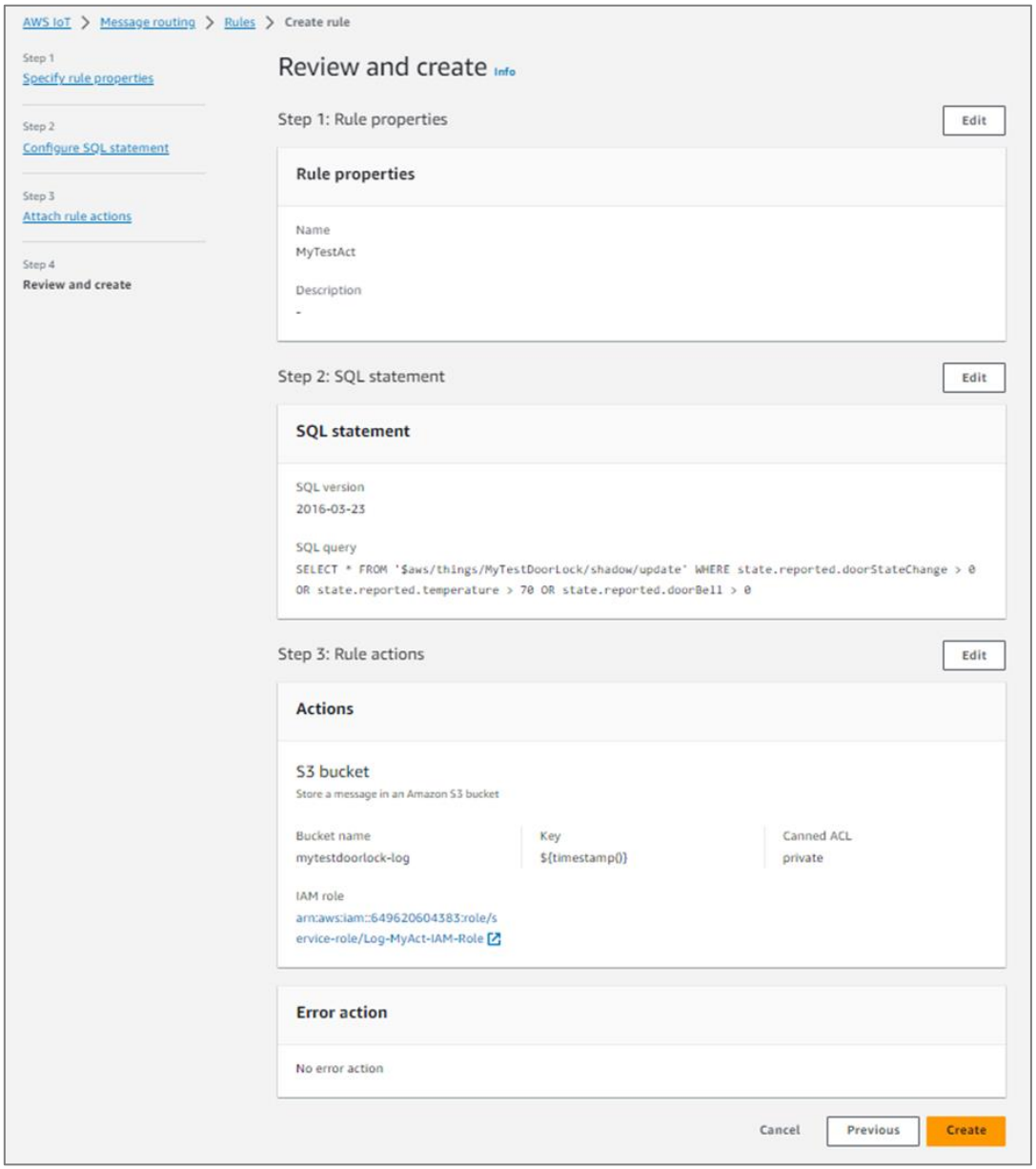

Figure 28. Review rules

8. The created rules should appear in the list of policies.

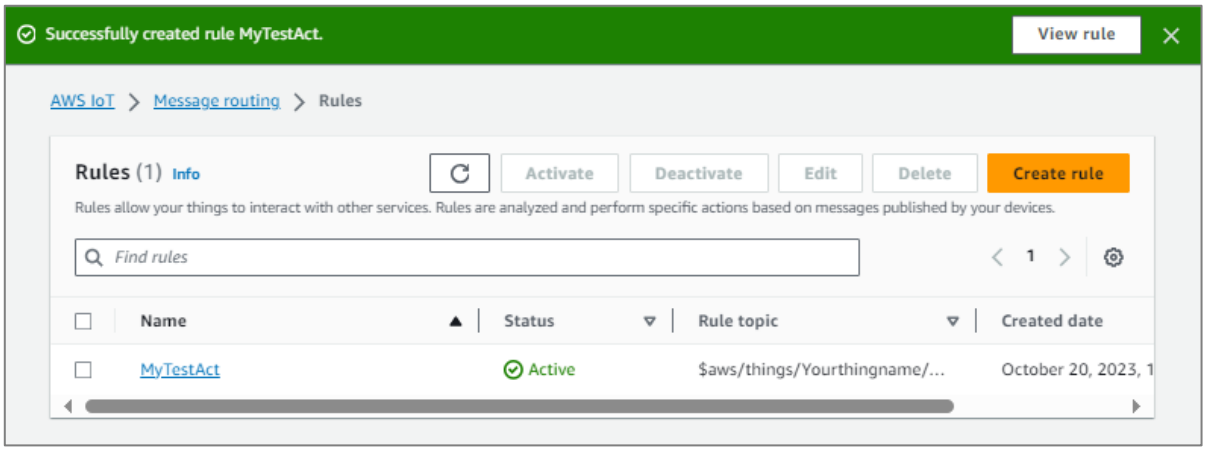

Figure 29. Created rule

9. Go to the IAM console and select **Roles** and review the created roles.

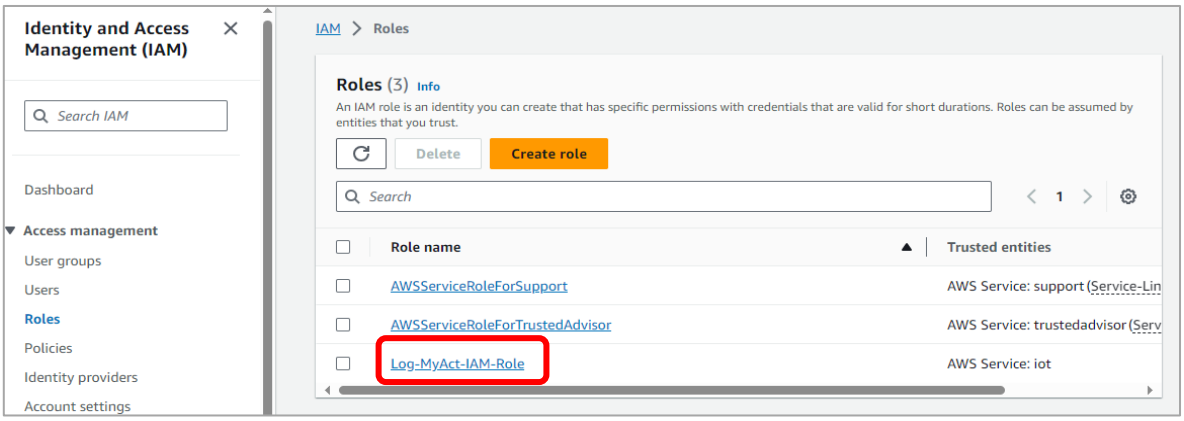

Figure 30. Created role

10. Choose the created role name and click **Attach policies**.

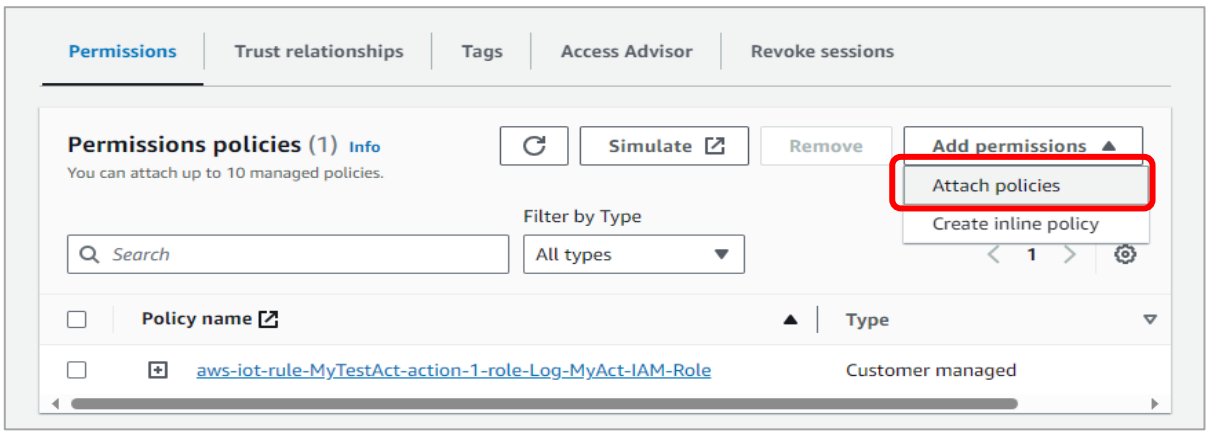

Figure 31. Attach policy to role

11. Search for the policy name of **AWSIoTFullAccess** and click **Add permissions**. Do the same thing for **AmazonS3FullAccess.**

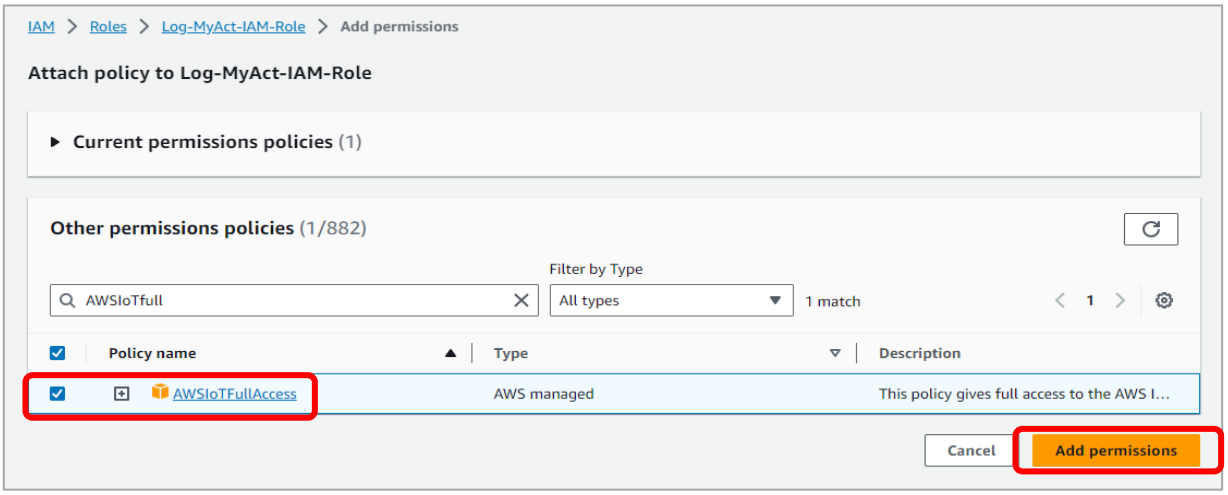

Figure 32. AWSIoTFullAccess policy

NOTE
AWSIoTFullAccess and AmazonS3FullAccess policies are not recommended for production.

When the policies are added, the execution roles should look similar to Figure 33.

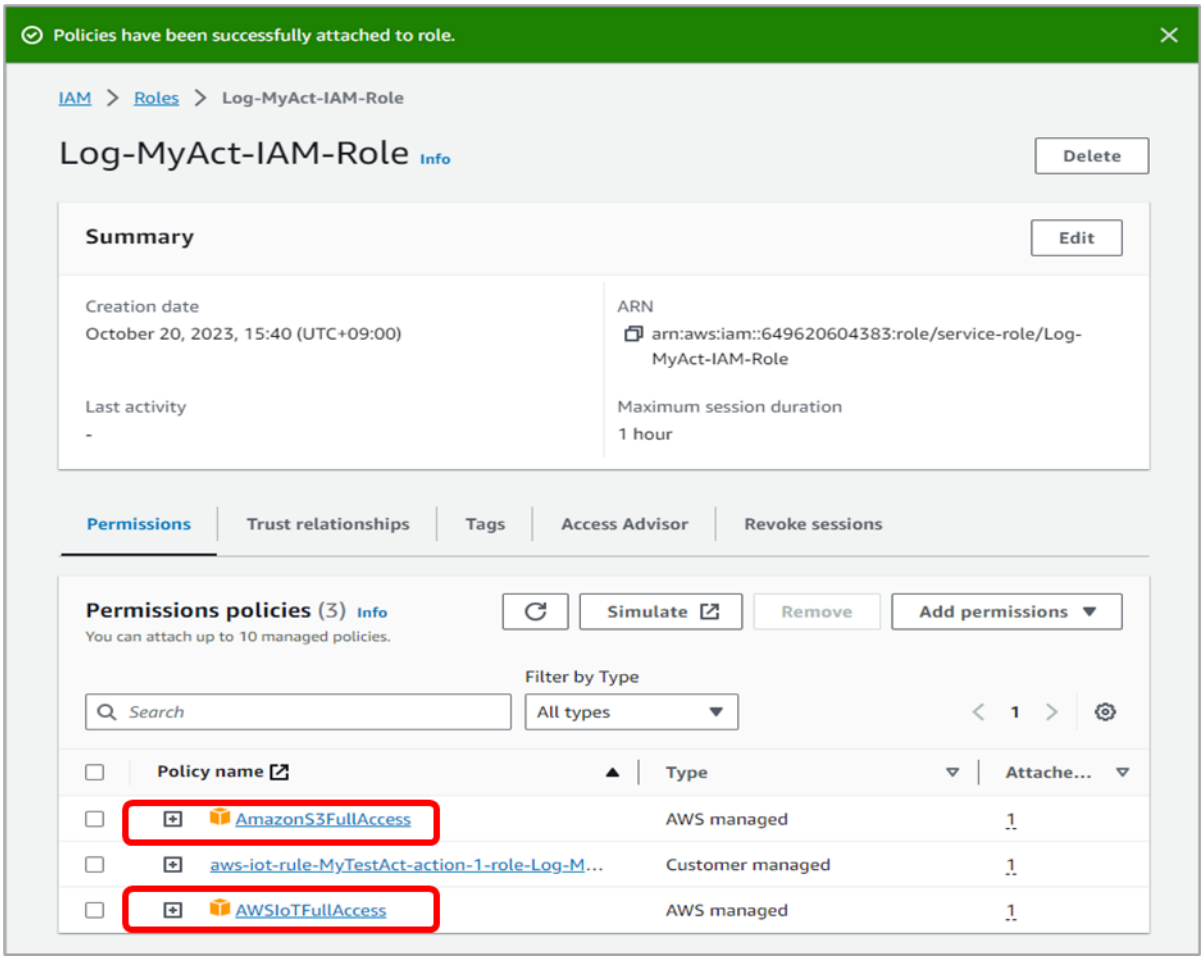

Figure 33. Attached policies

3.1.3 Configure Amazon Cognito

Amazon Cognito provides authentication, authorization, and user management for web and mobile apps. Users can sign in directly with a username and password or through a third party such as Facebook, Amazon, or Google.

The two main components of Amazon Cognito are **user pools** and **identity pools**. User pools are directory of users that provide sign-up and sign-in options for app users. Identity pools provide AWS credentials to grant users access to other AWS services. Identity pools and user pools can be used separately or together. For more information, visit AWS Docs site (https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html).

3.1.3.1 Create User Pools

1. Go to the Amazon Cognito console and click **Create user pool**.

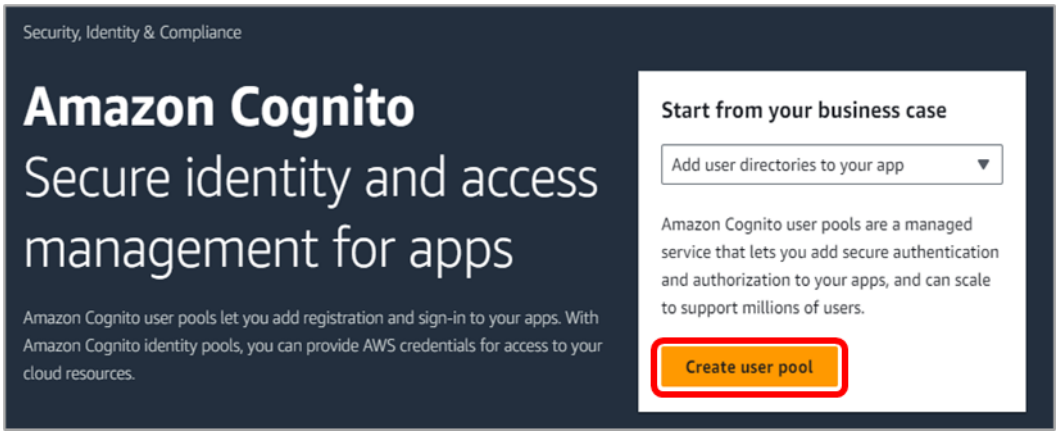

Figure 34. Create user pool

2. On the **Configure sign-in experience** page, select the **Email** checkbox, and click **Next**.

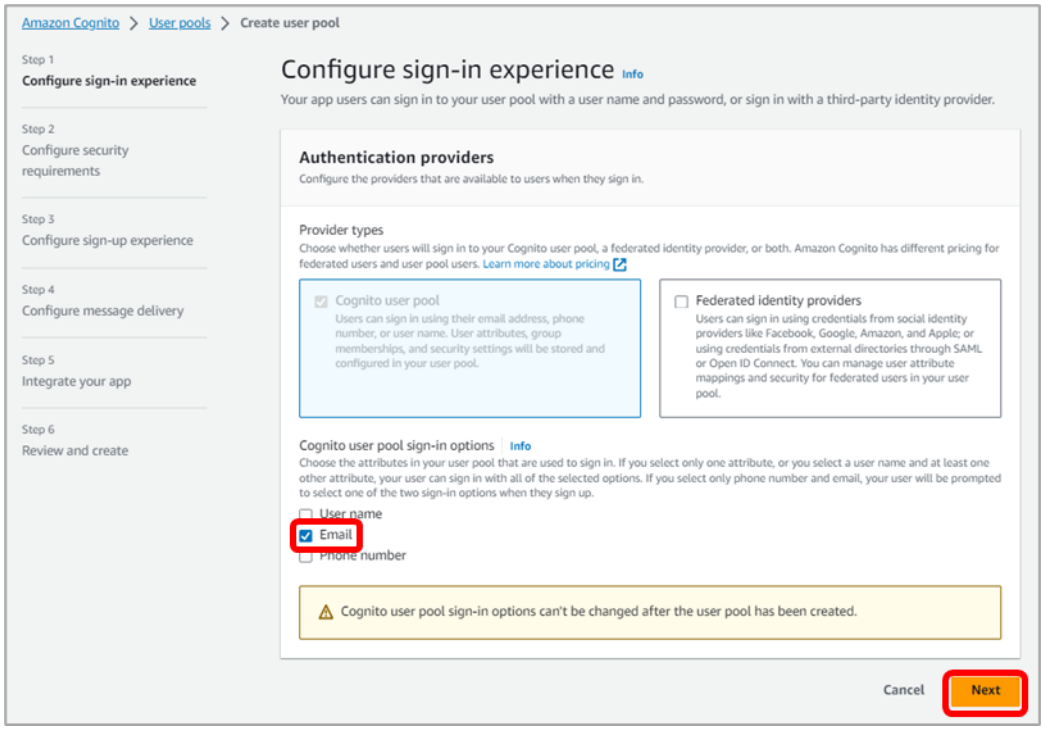

Figure 35. Configure sign-in options

3. Select **Cognito defaults** as password policy mode. Then, select **No MFA** and **Email only**, and click **Next**.

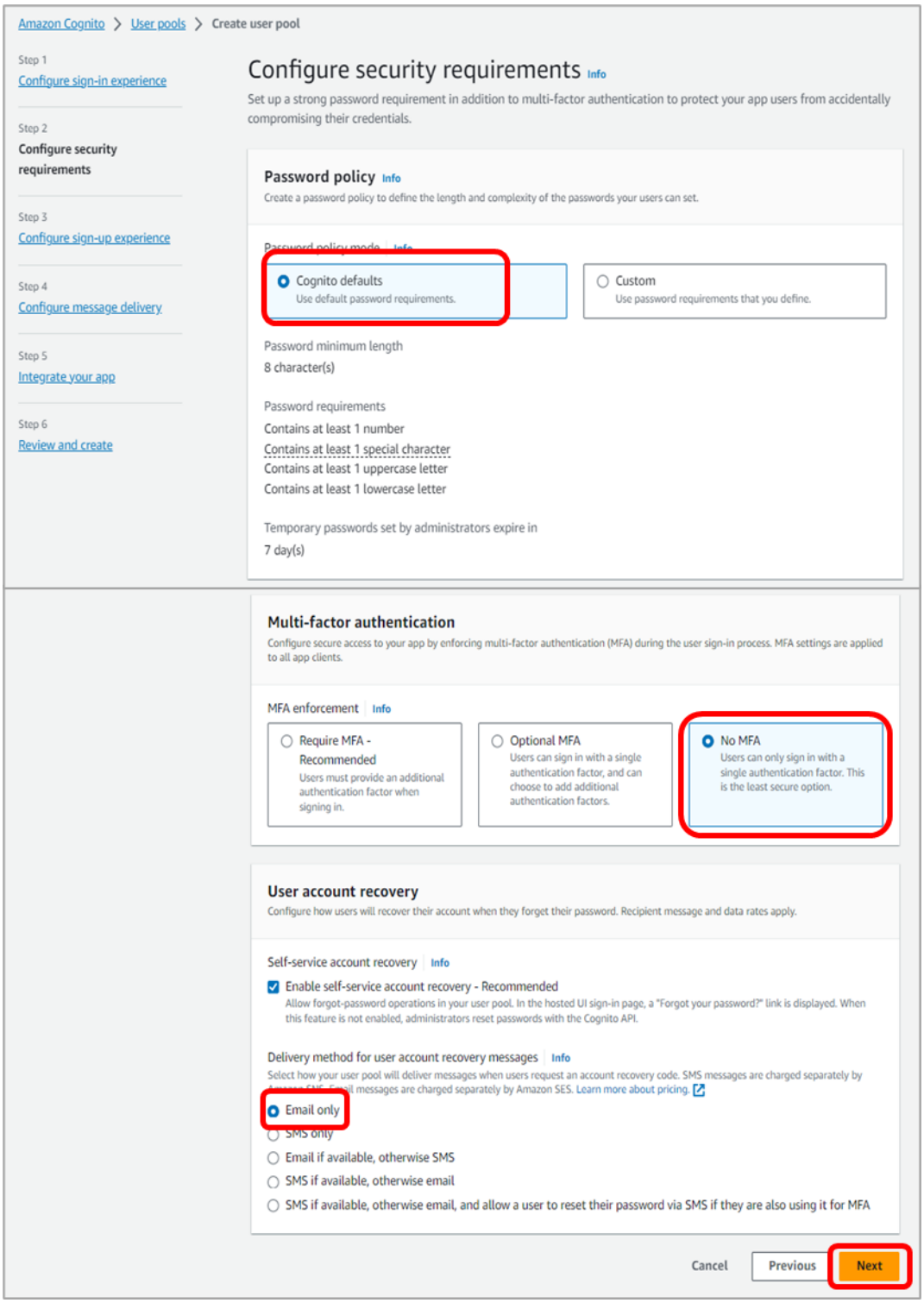

Figure 36. Configure security requirements

4. Configure sign-up as shown in Figure 37.

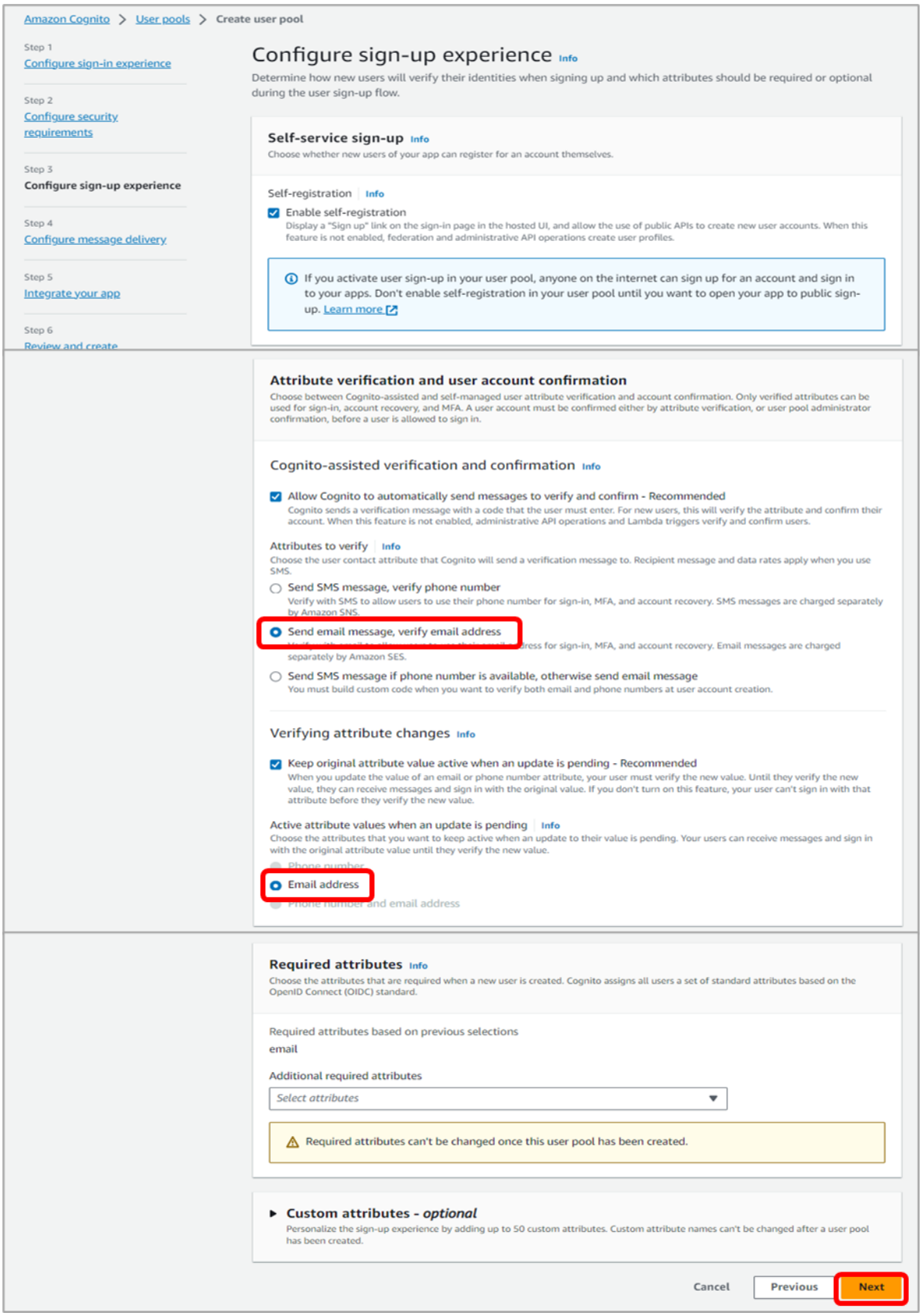

Figure 37. Configure sign-up experience

5. Select **Send email with Cognito**.

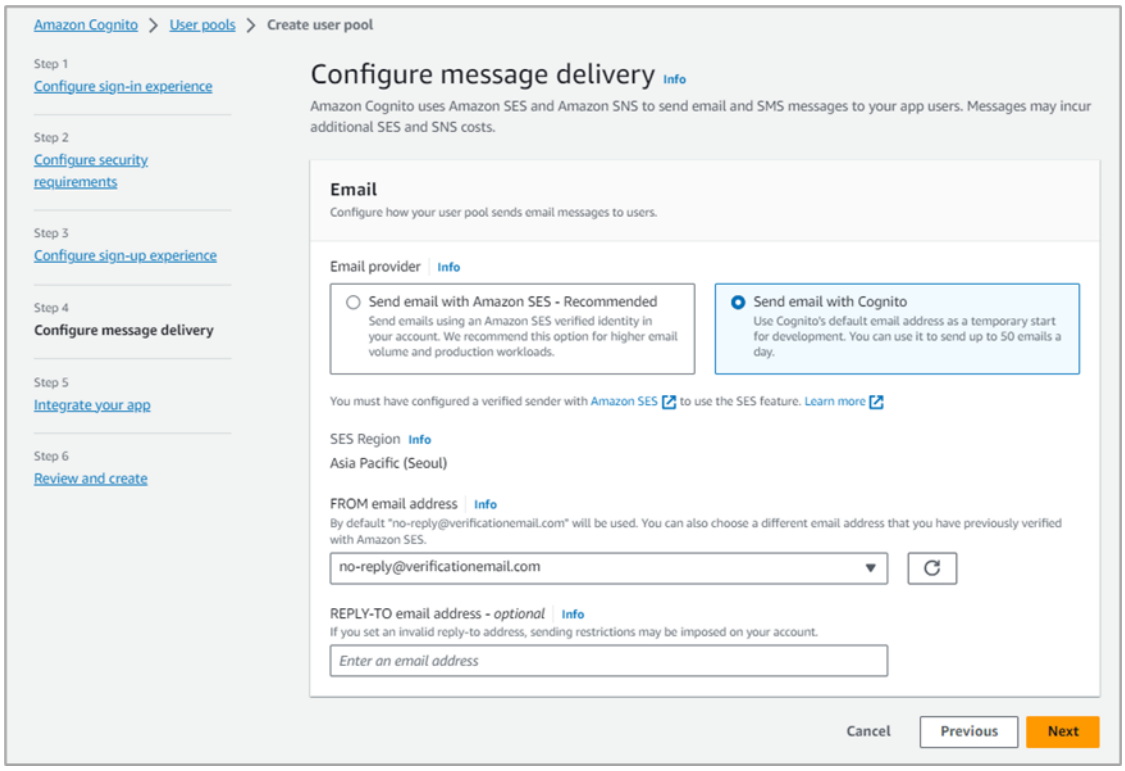

Figure 38. Configure message delivery

6. On the **Integrate your app** page, enter required items as shown in Figure 39 and click **Next**.

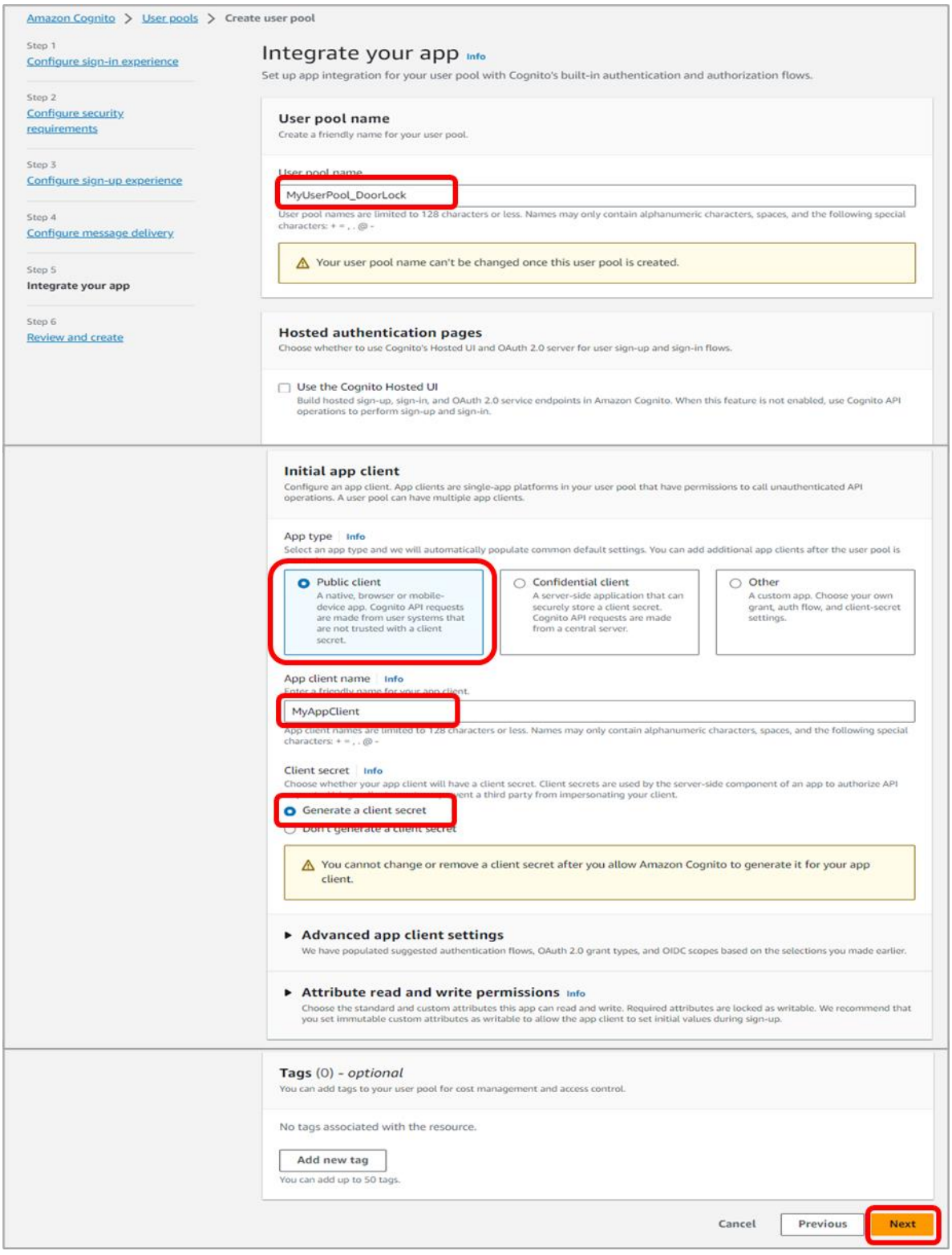

Figure 39. Integrate app client

7. On the **Review and create** page, review the entered information, and click **Create user pool**. Then, the created user pool should appear in the list.

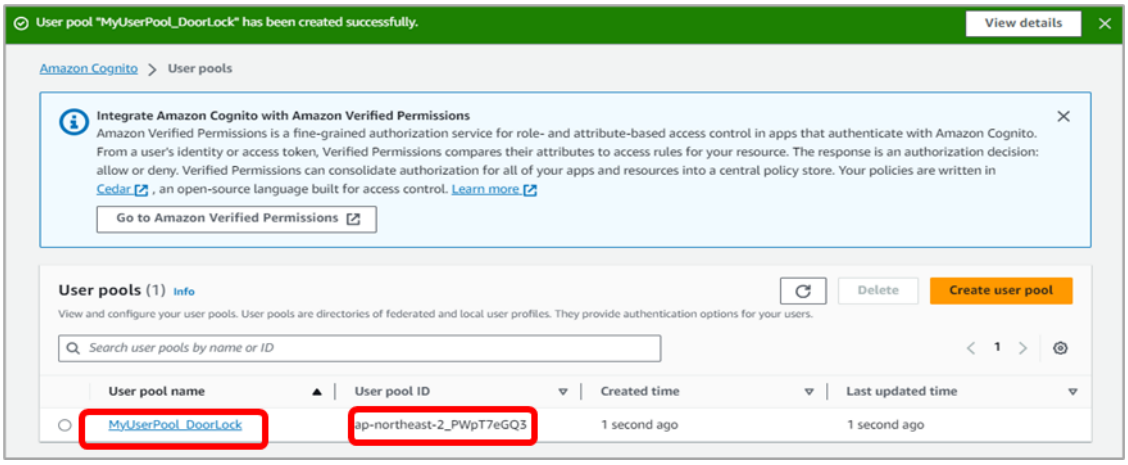

Figure 40. Created user pool

3.1.3.2 Create Identity Pools

1. Go to the Amazon Cognito console. Choose **Identity pools** and click **Create identity pool**.

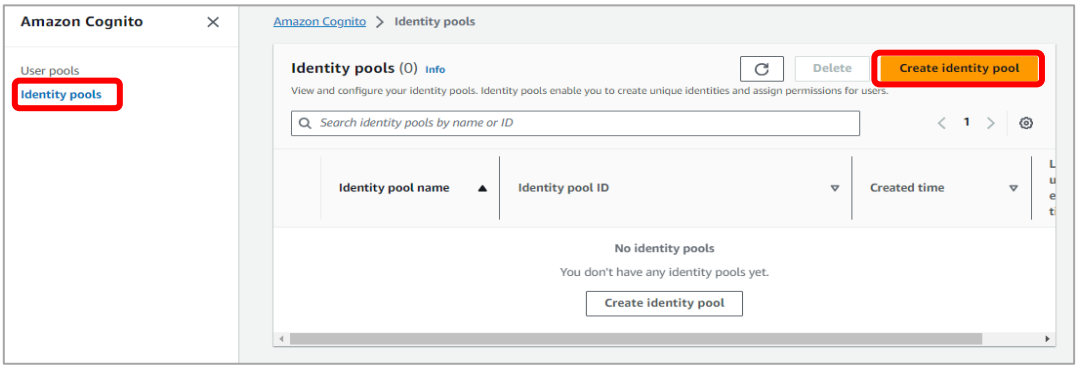

Figure 41. Create identity pool

2. On the **Configure identity pool trust** page, select **Guest access** and click **Next**.

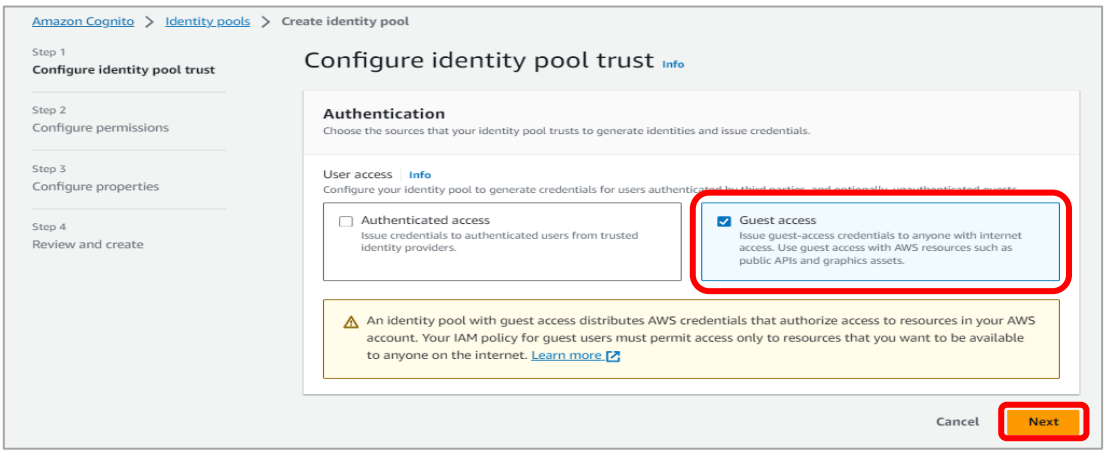

Figure 42. Create identity pool trust

3. On the **Configure permissions** page, select **Create a new IAM role,** enter an **IAM role name**, and then click **Next**.

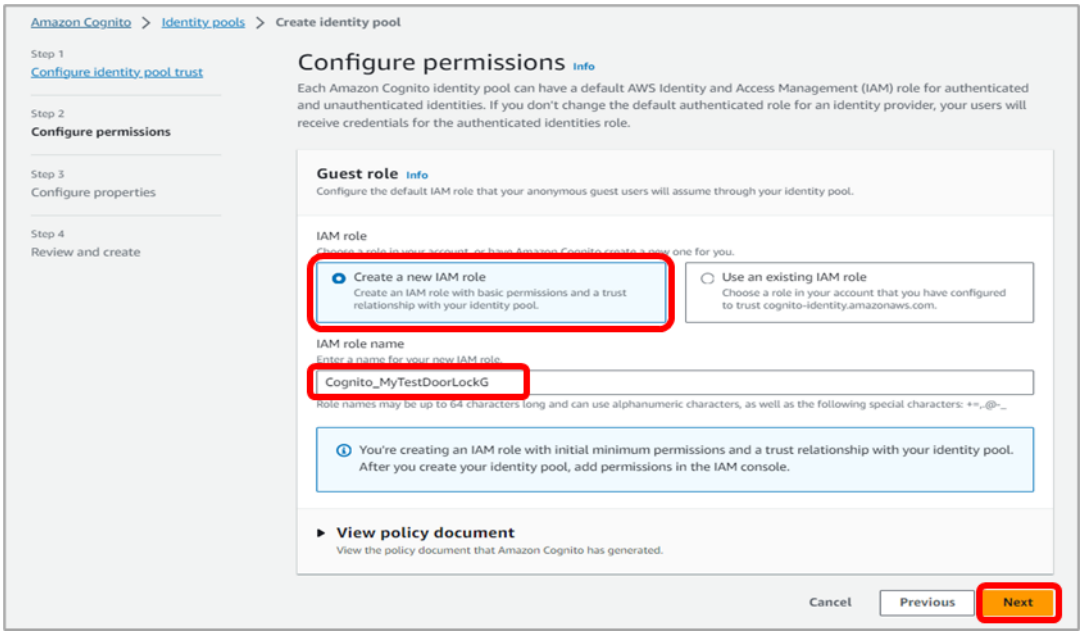

Figure 43. Configure permissions

4. Enter an Identity pool name and click **Next**.

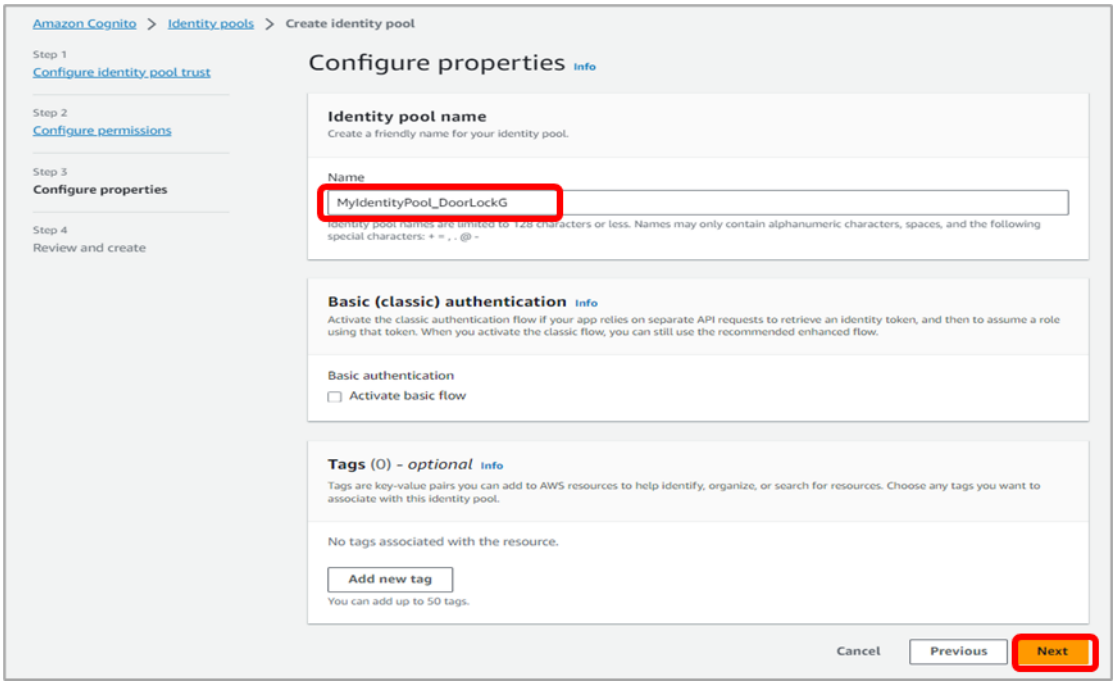

Figure 44. Configure properties

5. Review the selected items and click **Create identity pool**. Then, the created identity pools appear in the list.

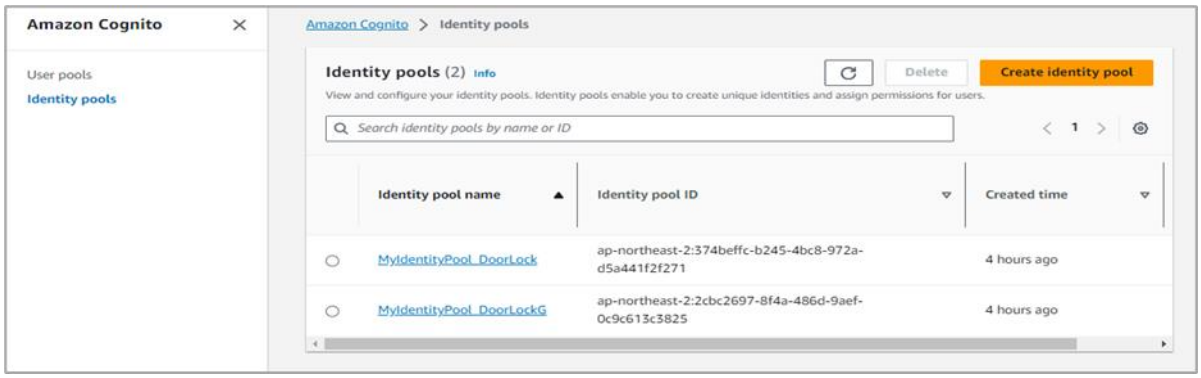

Figure 45. Created identity pools

3.1.4 Set Up AWS IAM

While creating an identity pool, you should update the IAM roles that the users assume. When a user logs in to the app, Amazon Cognito generates temporary AWS credentials for the user. These temporary credentials are associated with a specific IAM role. The IAM role lets users define a set of permissions to access AWS resources. For more information, visit https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html.

- The roles in Cognito_MyTestDoorlockG are created automatically when the federation identity is created via Cognito Identity Pool.
- The device only needs an unauthorized role.

To set up AWS IAM:

1. Go to the IAM console and select **Cognito_MyTestDoorLockG**.

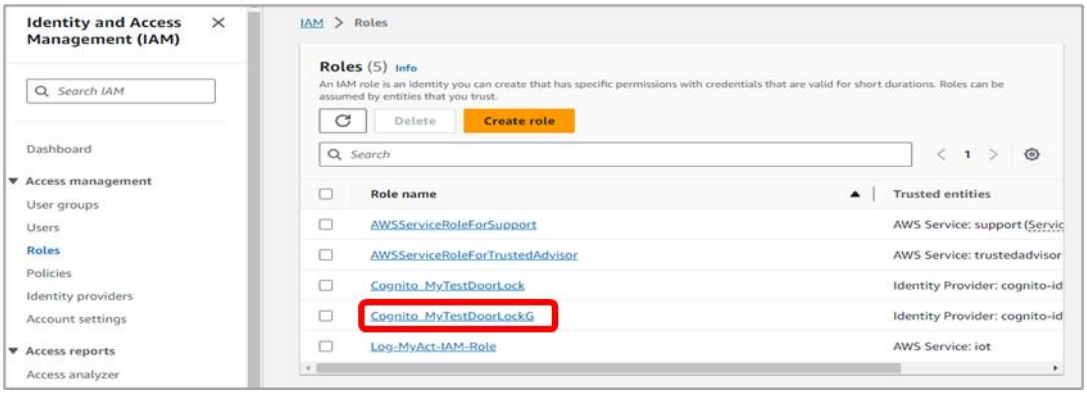

Figure 46. IAM role

2. Expand **Add permissions** and click **Attach policies**.

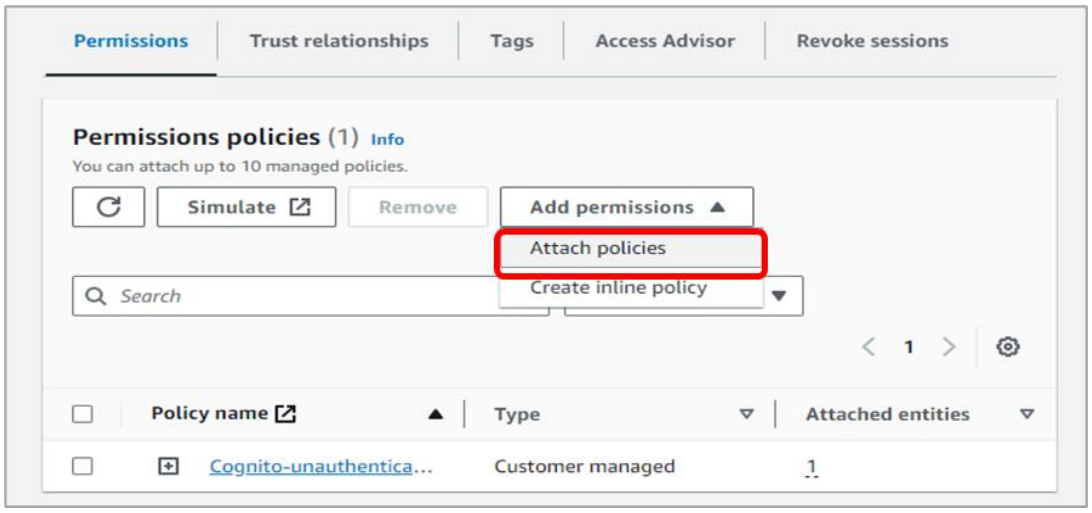

Figure 47. Attach policies

3. Search for the policy name of **AWSIoTFullAccess** and click **Add permissions**.

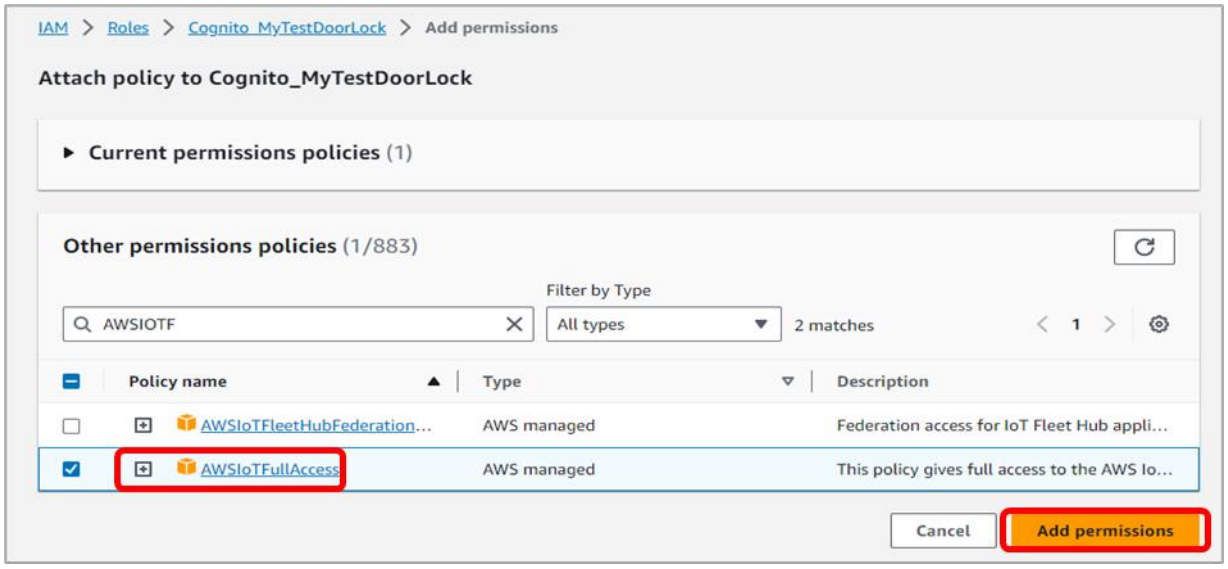

Figure 48. AWSIoTFullAccess policy

NOTE
AWSIoTFullAccess policy is not recommended for production.

4. Search for the policy name of **AmazonS3FullAccess** and click **Add permissions**.

Figure 49. AmazonS3FullAccess policy

NOTE
AmazonS3FullAccess policy is not recommended for production.

5. The attached policies appear in the list.

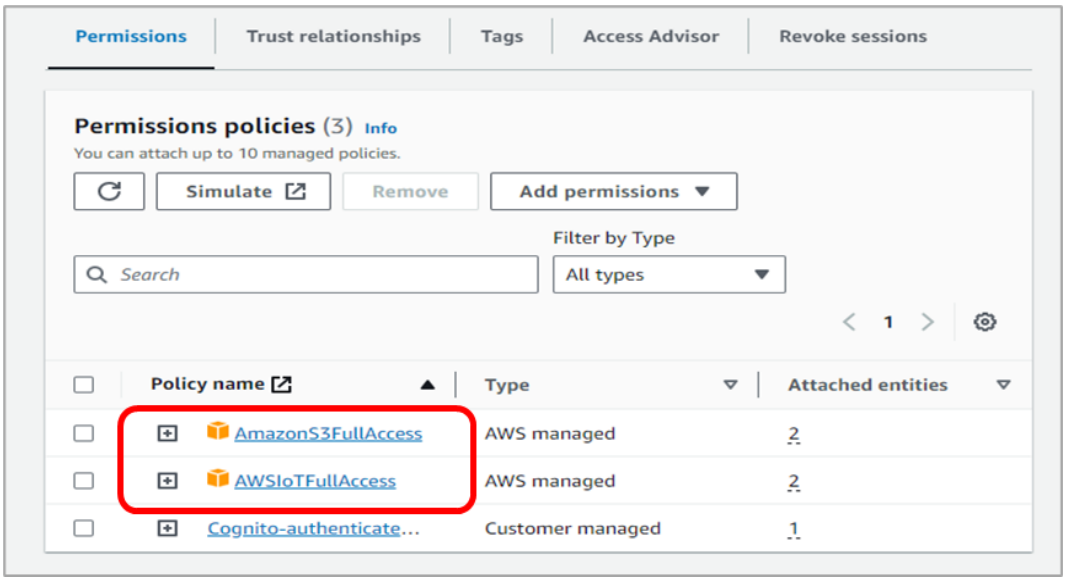

Figure 50. Attached policies

3.1.5 Create Amazon S3 Bucket

Every object in Amazon S3 is stored in a bucket. Before storing data in Amazon S3, you need to create a bucket. To create S3 bucket:

1. In the Amazon S3 console, in the left navigation pane, select **Buckets**, and click **Create bucket**.
2. On the **Create bucket** page, in the **Bucket name** field, type a bucket name.
3. For **Region**, choose the AWS region where you want the bucket to reside, and click **Create bucket**.

   When Amazon S3 successfully creates the bucket, the console displays an empty bucket in the **Buckets** pane.

4. Door Lock Reference Application

Door lock reference application is available on the official website.

NOTE
Go to the Renesas website (https://www.renesas.com/us/en/products/wireless-connectivity/wi-fi/low-power-wi-fi) and scroll down to the Software Downloads section. Find "AWS IoT Reference" or type it in the search box, and then select the reference package and download.

4.1 Reference DA16200/DA16600 SDK Setting

4.1.1 Edit Endpoint

1. Change the AWS_USER_MQTT_HOST in the **app_aws_user_conf.h** file as follows:
   - AWS IoT > Settings

```
#define AWS_USER_MQTT_HOST    "(account-specific-prefix).iot.(aws-region).amazonaws.com"
```

2. Build the SDK and then update the image.

4.1.2 Edit Thing Name

4.1.2.1 Edit Thing Name with Console Command

You can directly rename things using the console commands without the need to build an SDK. Renesas recommends using console commands over other methods for changing a thing name.

If the test board is running, run the factory command first, and then proceed to NVRAM as follows.

```
[/DA16200] #
[/DA16200] # nvram
Command-List is changed, "NVRAM"
[/DA16200/NVRAM] #
[/DA16200/NVRAM] # setenv APP_THINGNAME USER_THING_NAME  // write user thing name
[/DA16200/NVRAM] # getenv                                // read user thing name
[/DA16200/NVRAM] # unsetenv APP_THINGNAME                // remove user thing name
[/DA16200/NVRAM] #
```

After that, complete the provisioning process.

```
[/DA16200] # nvram
    Command-List is changed, "NVRAM"
[/DA16200/NVRAM] # setenv APP_THINGNAME APP-DOORLOCK-1
[/DA16200/NVRAM] # getenv

Total length (411)
APP_THINGNAME (STR,15) ....... APP-DOORLOCK-1
N1_Profile (STR,02) .......... 1
N1_mode (STR,02) ............. 2
SYSMODE (STR,02) ............. 1
N1_ssid (STR,17) ............. "Renesas_DA16200"
N1_psk (STR,13) .............. "1234567890"
N1_proto (STR,04) ............ RSN
N1_key_mgmt (STR,08) ......... WPA-PSK
```

```
country_code (STR,03) ........ KR
1:IPADDR (STR,09) ............ 10.0.0.1
1:NETMASK (STR,14) ........... 255.255.255.0
1:GATEWAY (STR,09) ........... 10.0.0.1
1:DNSSVR (STR,08) ............ 8.8.8.8
USEDHCPD (STR,02) ............ 1
DHCPD_IPCNT (STR,03) ......... 10
DHCPD_TIME (STR,05) .......... 3600
DHCPD_S_IP (STR,09) .......... 10.0.0.2
DHCPD_E_IP (STR,10) .......... 10.0.0.11
DHCPD_DNS (STR,08) ........... 8.8.8.8
[/DA16200/NVRAM] #
```

4.1.2.2 Edit Thing Name in Configuration File

If the thing name does not exist in NVRAM, the predefined name located in the first header is stored in NVRAM.

Change AWS_USER_MY_THING_NAME in the **app_thing_manager.h** file, then build the SDK and update the image:

```
/*
*   USER Thing name define
*   Generic SDK default : "DA16200"
*   AWS IOT default : "IOT-SENSOR-46" or "FAE-DOORLOCK-4" or "assigned_thing_name"
*/
#define APP_USER_MY_THING_NAME  "FAE-DOORLOCK-4"
```

4.1.3 Edit Image File Name for OTA

To test the OTA update, edit the **app_aws_user_conf.h** file in the DA16200 SDK and modify the file names to match the image file names that are uploaded to the Amazon S3 bucket.

```
#if defined(__BLE_COMBO_REF__)
#define RTOS_NAME                       "DA16600_FRTOS-GEN01.img"
#define BLE_NAME                                "DA16600_BLE_OTA.img"
#else
#define RTOS_NAME                       "DA16200_FRTOS-GEN01.img"
#endif
```

4.1.4 Connect Certificates to Thing

To authenticate the device with AWS IoT, the device must contain the **Root CA**, **Client Certificate,** and **Client Private Key**. For more information, see https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html.

To add these certificates to the device, edit **app_aws_certi.h** and insert the certificates downloaded from AWS as follows:

```
#define democonfigROOT_CA_PEM   "-----BEGIN CERTIFICATE-----\n" \
"MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
"ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
"b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
. . .
"o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
"5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
"rqXRfboQnoZsG4q5WTP468SQvvG5\n" \
"-----END CERTIFICATE-----\n"

#define democonfigCLIENT_CERTIFICATE_PEM        "-----BEGIN CERTIFICATE-----\n" \
"MIIDWjCCAkKgAwIBAgIVAIqSKvd/Qq2E9ZleQWN2Gk/iPw2GMA0GCSqGSIb3DQEB\n" \
"CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t\n" \
"IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0xODEyMDYwNjQw\n" \
. . .
"TcaCwbJQy2XprqPpBo3ZuWqmSi55uslXj+2B4XgPZutim++8J7DHQbfHAGZwiAFN\n" \
"90TNlhZBdI87Ga07p0db03KcBQs8dBMaABC0RK39LqJ5ZdQMT/Owx0+iO2Be7w30\n" \
"7o06zCQB2A0nmfvAR8gSuImIBfKz2I1xQX5+CO4wes8RH5pNIOK2QrKgr9NJkA==\n" \
"-----END CERTIFICATE-----\n"

#define democonfigCLIENT_PRIVATE_KEY_PEM        "-----BEGIN RSA PRIVATE KEY-----\n" \
"MIIEpAIBAAKCAQEA2fwGze8cV4ALJcgdeGR1fzD1I66YD0p62x3C8ITqSiC6B4iz\n" \
"ugk6n17/6cXF8odFAh6adTxet5tL5mGLgLnkYFtt7Iyj10T8hpxT1Yxp7TYZRblw\n" \
"Fl9fptPRi5KncVhs9sICqJEmvKTDv6LUwIlefrofMv+6uX7gEhssGUeVnrrR/Mo8\n" \
. . .
"EOP11QKBgQCDnAVbfrXC+4S5UNwxGHw4cZJwAvOkkeApV3WlBSZFbbGzIxrVy79O\n" \
"7ETTGfSAbksUljV+2HZZVSXtgsCS/fzsFjMWYpeNRX3+9wtFfGCfxoygGW0JvOyY\n" \
"kg61geirHUDYgog9XzGKATXc3K/m7JdyOcWdbf54nhzcEqjRv1DhCA==\n" \
"-----END RSA PRIVATE KEY-----\n" \
```

4.2 Reference Application in DA16200/DA16600

The following components shown in Figure 51 are required to run the application in DA16200/DA16600 via an Internet connection and AWS IoT server:

- AWS IoT reference application package
- DA16200/DA1660 EVB
- Router: Connection to internet
- Mobile device: Android/iOS application
- AWS account.

Figure 51. Architecture of AWS IoT

Install the mobile application by searching for **DA16200** or **DA16600** on the Google Pay Store or the Apple App Store on the mobile devices.

Provisioning is required for connection between DA16200/DA16600 and Router before connecting DA16200/DA16600 with AWS IoT hub. The provisioning can be done with the Renesas Wi-Fi Provisioning app on either an Android or iOS device. For details on how to install and provision the mobile app, see Ref. [4]. When provisioning is completed, select AWS IoT to open AWS application on mobile device.

4.2.1 Open Door

Figure 52 shows message flows of opening the door.

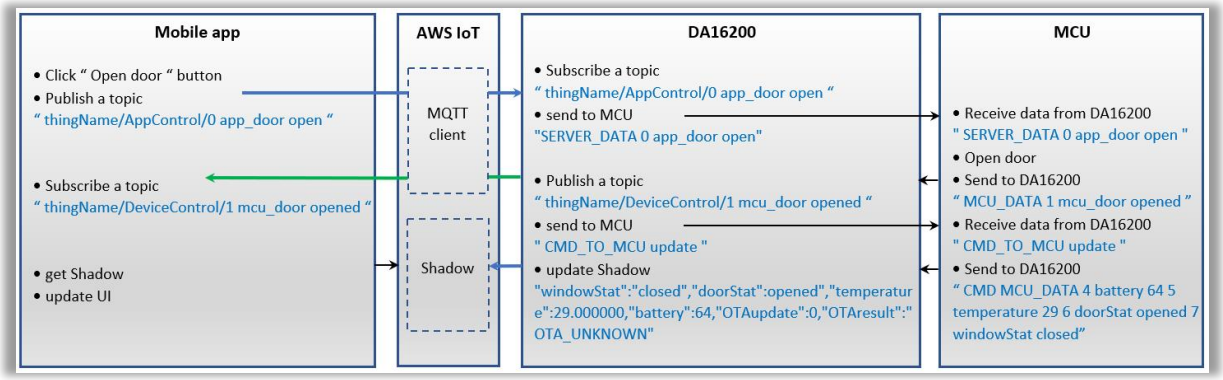

Figure 52. Message flows of opening door

The operation of **opening door** in Android app is shown in Figure 54.

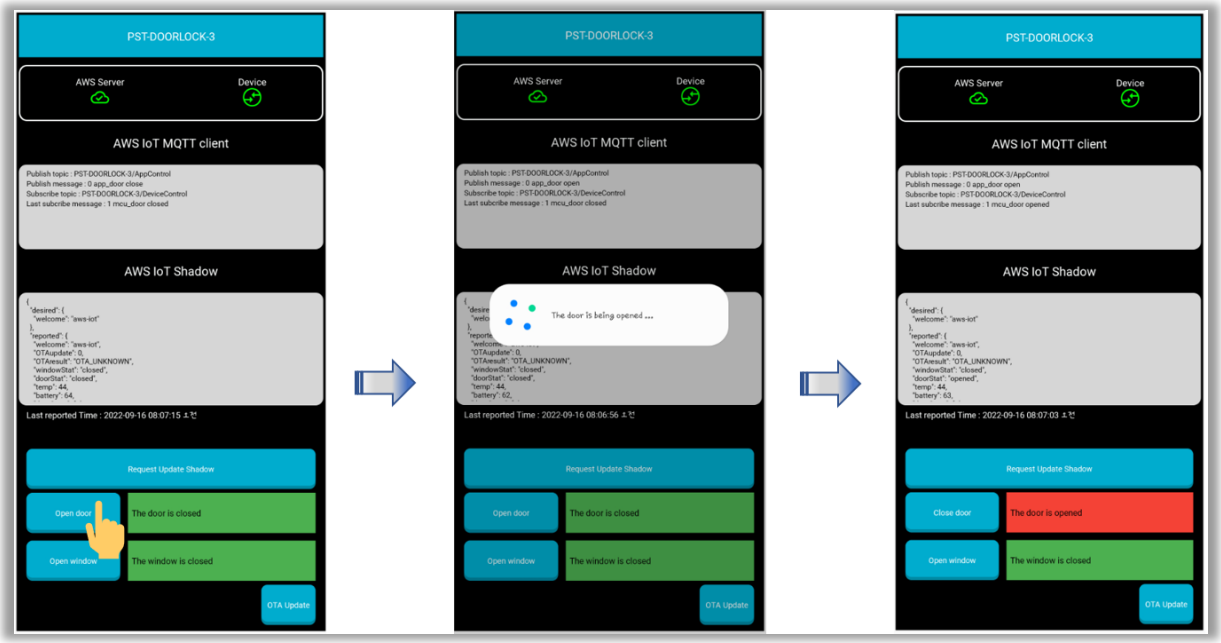

Figure 53. Open dooring on mobile app

Figure 54. Shadow state when door is open

When the operation of opening door is completed, the console logs of the DA16200 appear as follows:

```
Count : 0, cmdNum = 4
mqtttype = 1
index(=3) matched
data type(shadow) = 0
call update sensor(need to be set variable): battery = 63

Count : 1, cmdNum = 5
mqtttype = 1
index(=2) matched
data type(shadow) = 2
call update sensor(need to be set variable): temperature = 28.000000
```

```
Count : 2, cmdNum = 6
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = opened

Count : 3, cmdNum = 7
mqtttype = 1
index(=0) matched
data type(shadow) = 1
call update sensor(need to be set variable): windowStat = closed

release response

*********************************************************************************
publish (shadow sensor update) OK -     payload:
"{"state":{"reported":{"windowStat":"closed","doorStat":"opened","temperature":28.0
00000,"battery":63,"OTAupdate":0,"OTAresult":"OTA_UNKNOWN"}},"clientToken":"PST-DOORLOCK-3-0"}"
```

4.2.2 Close Door

Figure 55 shows message flows of closing door.

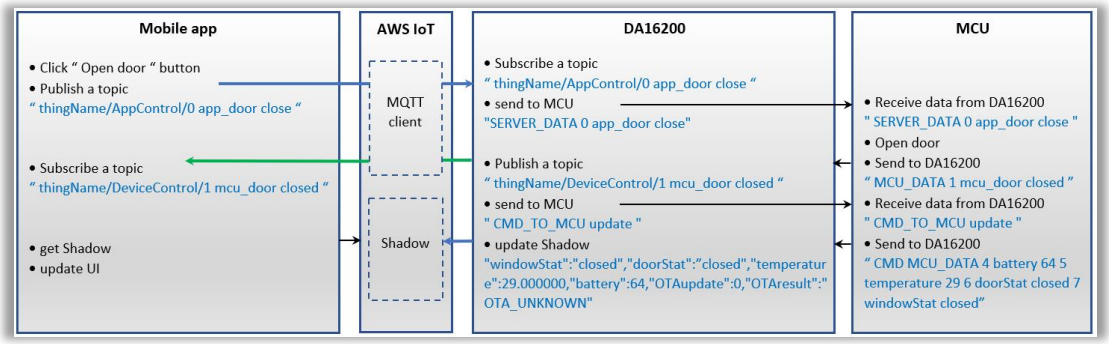

Figure 55. Message flows of closing door

The operation of **closing door** in Android app is shown in Figure 56.

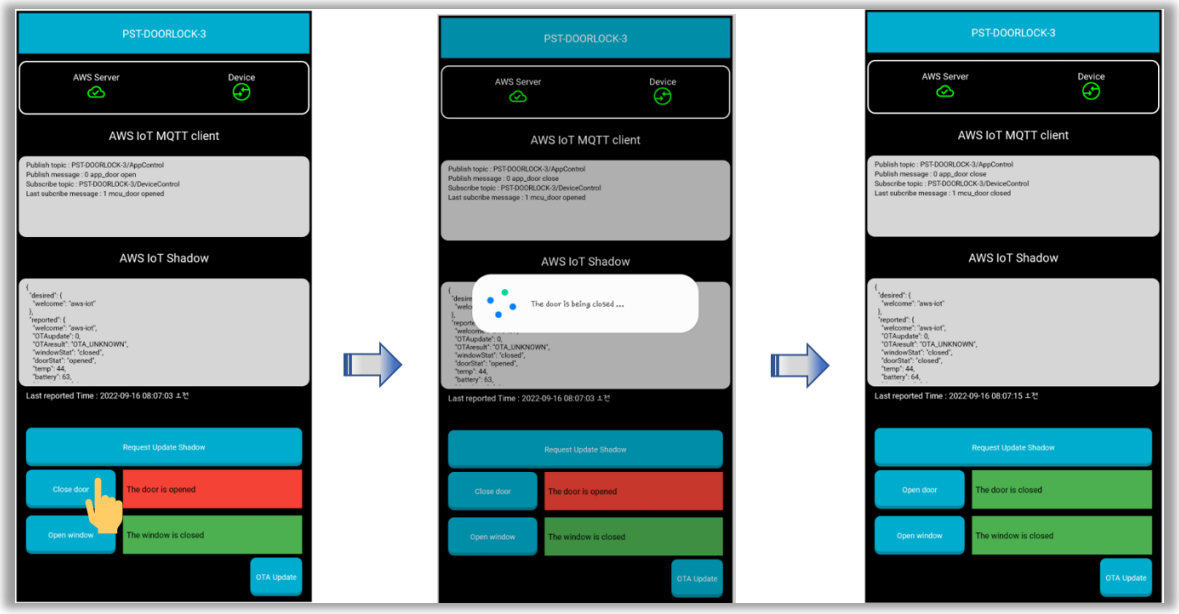

Figure 56. Closing door on mobile app

Figure 57 shows the state of Shadow on the AWS IoT Hub when the operation for closing door is completed.

Device Shadow state


```
{
  "state": {
    "desired": {
      "welcome": "aws-iot"
    },
    "reported": {
      "welcome": "aws-iot",
      "OTAupdate": 0,
      "OTAresult": "OTA_UNKNOWN",
      "windowStat": "closed",
      "doorStat": "closed",
      "temp": 44,
      "battery": 76,
      "doorState": false,
      "openMethod": "app",
      "doorStateChange": 1,
      "DoorOpenMode": 0,
      "temperature": 41,
```


Figure 57. Shadow state when door is closed

When the operation of closing door is completed, the console logs of the DA16200/DA16600 appears as follows:

```
Count : 0, cmdNum = 4
mqtttype = 1
index(=3) matched
data type(shadow) = 0
call update sensor(need to be set variable): battery = 76
```

```
Count : 1, cmdNum = 5
mqtttype = 1
index(=2) matched
data type(shadow) = 2
call update sensor(need to be set variable): temperature = 41.000000

Count : 2, cmdNum = 6
mqtttype = 1
index(=1) matched
data type(shadow) = 1
call update sensor(need to be set variable): doorStat = closed

Count : 3, cmdNum = 7
mqtttype = 1
index(=0) matched
data type(shadow) = 1
call update sensor(need to be set variable): windowStat = closed

release response

********************************************************************************
publish (shadow sensor update) OK -     payload:
"{"state":{"reported":{"windowStat":"closed","doorStat":"closed","temperature":41.0
00000,"battery":76,"OTAupdate":0,"OTAresult":"OTA_UNKNOWN"}},"clientToken":"PST-DOORLOCK-3-0"}"
```

4.3 Reference Application in Host MCU

Application in host MCU can control DA16200/DA16600 and connection between the host MCU and mobile phone through AWS IoT server using AT commands. Figure 58 shows the AWS IoT using firmware images for AT commands and host MCU.

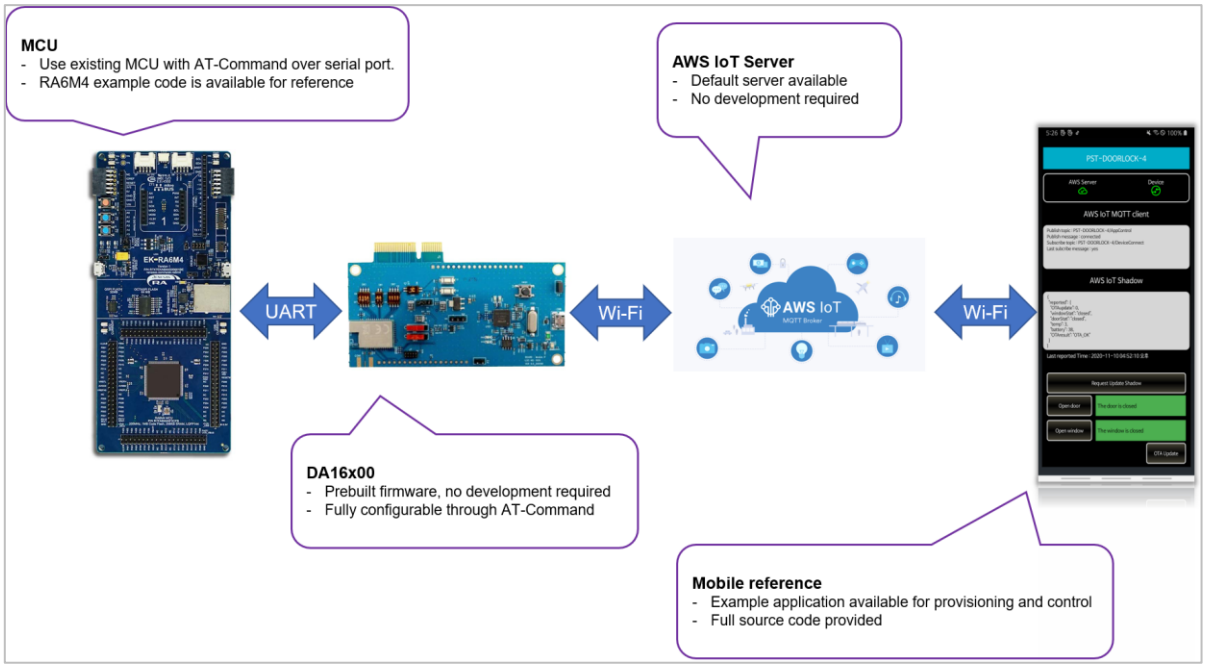

Figure 58. AWS IoT using firmware images for AT commands and host MCU

4.3.1 Download Package for Door Lock Reference Application in Host MCU

A firmware image for AT command and application in MCU are available on the official Renesas website (https://www.renesas.com/us/en/products/wireless-connectivity/wi-fi/low-power-wi-fi).
The contents of the package are the following:

- \DA16200 or \DA16600
  - Firmware images for the DA16200/DA16600 Wi-Fi devices.
  - Tera Term script for downloading the firmware images to the DA16200/DA16600 Wi-Fi device.
- \DA16200\Script (\DA16600\Script)
  - Tera Term script that demonstrates how to use AT commands for AWS IoT using a personal computer and the DA16200/DA16600.
  - Getting Stared with AT commands for AWS IoT
    - Introduces the DA16200/DA16600 AT commands for AWS IoT and describes how to set up the development environment and test the examples.
    - Describes how to connect an external host to the DA16200/DA16600 EVK for using the AT commands for AWS IoT.
    - Describes the AT commands for AWS IoT command list.
- \MCU
  - Sample project based on the RA6M4 development environment which demonstrates how to use AT commands for AWS IoT.

4.3.2 Hardware Connections between DA16200/DA16600 and Host MCU

The hardware components shown in Figure 59 are required to run door lock reference application using AT commands and host MCU:

- DA16200/DA16600 EVK
- EK-RA6M4 board

- Windows laptop or personal computer.

In addition, the following hardware connections are required for each operation:

- UART0: Programming firmware images and monitoring logs from DA16200/DA16600.
- UART1 or UART2: AT command interface between MCU and DA16200/DA16600.
- GPIO from the MCU to the DA16200/DA16600 to wake up the DA16200/DA16600 from DPM Low-power mode (DPM LPM).
- GPIO from the DA16200/DA16600 to host MCU to wake up the MCU in Sleep mode.

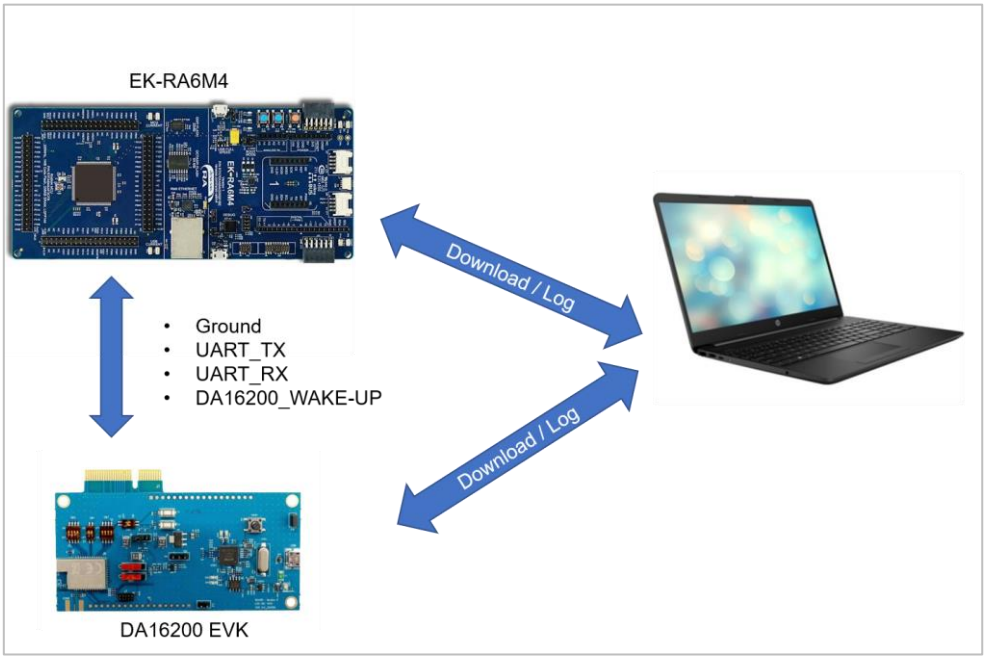

Figure 59. Hardware configuration

Table 1 shows the pin connections between the DA16200/DA16600 EVK and the EK-RA6M4 board.

Table 1. Pin connection

Function	DA16200 EVK		DA16600 EVK		EK-RA6M4 Board	
	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name
Ground	J3.18	GND	J2.12	GND	J24-7	GND
UART_TX	J4.11	TX1/GPIOA_4	J2.2	TX2/GPIOC_6	J23-2	D1/TXD
UART_RX	J4.12	RX1/GPIOA_5	J2.4	RX2/GPIOC_7	J23-1	D0/RXD
DA16200_WAKE_UP	J3.11	RTC_WAKE_UP2	SW1	RTC_WAKE_UP2	J23-6	D5/PWM
MCU_WAKE_UP	J4.18	GPIOA_11	J2.9	GPIOA_11	None	None

4.3.2.1 UART Connection for AT Commands

Table 2 shows the default configuration of UART1 (DA16200 EVB) or UART2 (DA16600 EVB) for AT commands.

Table 2. Default configuration for UART1 or UART2

Settings	Value
Baud Rate	115200
Data Bits	8

Settings	Value
Parity	None
Stop Bits	1
Flow Control (HW/SW)	None

The DA16200 EVB uses GPIOA_4 and GPIOA_5 for UART1 TX and UART1 RX, and the DA16600 EVB uses GPIOC_6 and GPIOC_7 for UART2 TX and UART2 RX by default. In addition, GND needs to be connected to the host MCU as shown in Figure 60.

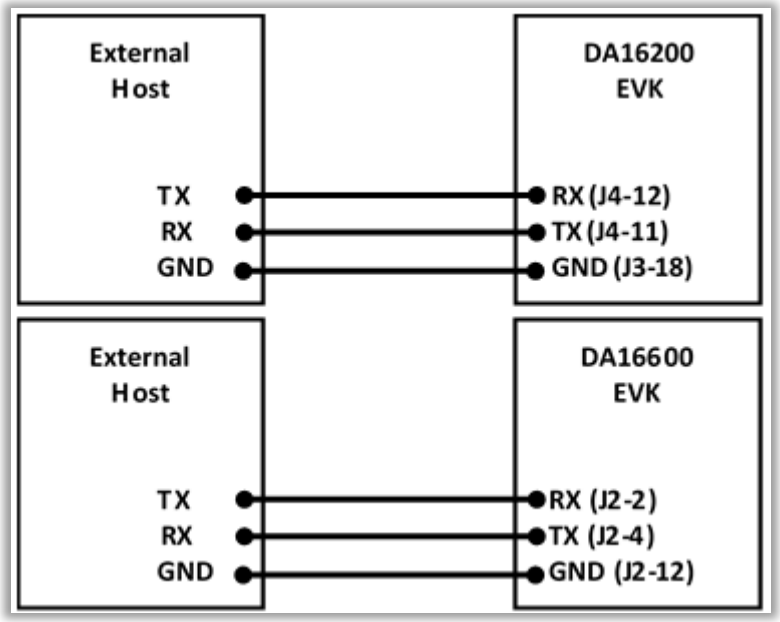

Figure 60. Default UART hardware connection

If the GPIO pin configuration is changed using AT commands, other connections for UART1 can be used as shown in Figure 61. The following AT command is used for GPIOA_2 for UART1 TX and GPIOA_3 for UART1 RX. Table 3 shows the pin combination for UART1.

```
AT+AWS=SET NV_PIN_BMUX BMUX_UART1d            // GPIOA_2 and GPIOA_3 for UART1
```

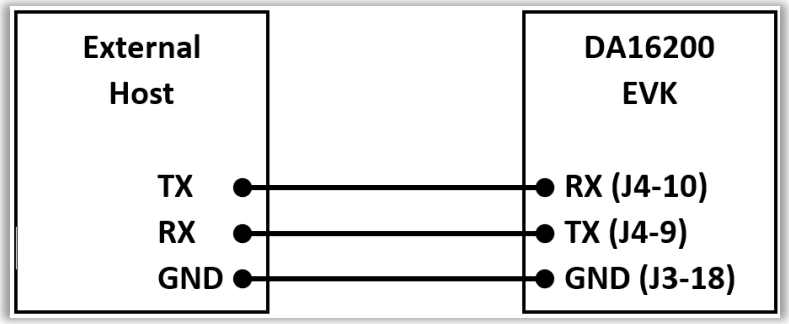

Figure 61. Sample example of UART1 connection

Table 3. UART1 pin configuration

PIN Mux	GPIO	Signal Name
PIN_AMUX	GPIOA_0	TX
	GPIOA_1	RX
PIN_BMUX	GPIOA_2	TX

PIN Mux	GPIO	Signal Name
	GPIOA_3	RX
PIN_CMUX	GPIOA_4	TX
	GPIOA_5	RX
PIN_DMUX	GPIOA_6	TX
	GPIOA_7	RX

When Dynamic Power Management (DPM) mode is enabled and DA16200/DA16600 is in DPM LPM, the host MCU must wake up the DA16200/DA16600 from DPM LPM using RTC_WAKE_UP. Then, the host MCU can send or receive data over the network in DPM Fully Functional Mode (FFM). The wake-up event is triggered when the GPIO pin of the host MCU changes from Low to High and then back to Low.

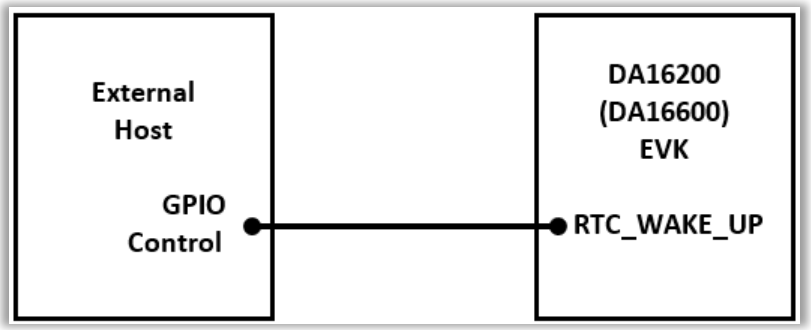

Figure 62. Hardware connection for waking up DA16200/DA16600

The host MCU may be in Sleep mode when DA16200/DA16600 wakes up from DPM LPM and needs to send responses to the host MCU. In this scenario, the DA16200/DA16600 needs to wake up the host MCU from sleep using GPIO as shown in Figure 62. This connection is not required if the host MCU does not use sleep mode. GPIOA_11 is available on DA16200/DA16600 EVB for waking up host MCU by default (see Figure 63) and it can be configured using the following AT commands:

```
AT+AWS SET APP_MCU_WKAEUP_PORT GPIO_UNIT_A            // GPIO_A port
AT+AWS SET APP_MCU_WKAEUP_PIN GPIO_PIN11        // GPIO_11
```

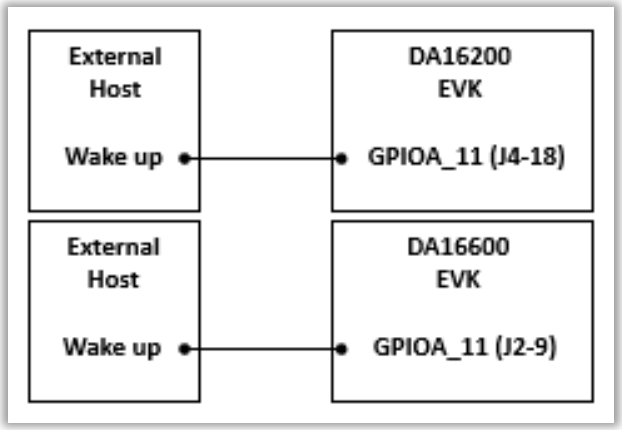

Figure 63. Default pin configuration for waking up host MCU

Other GPIOs in the DA16200 EVB can be used for waking up the host MCU as shown in Table 4. For example, GPIOC_6 can be configured for waking up host MCU using the follwoing AT commands (see Figure 64):

```
AT+AWS SET APP_MCU_WKAEUP_PORT GPIO_UNIT_C            // GPIO_C port
AT+AWS SET APP_MCU_WKAEUP_PIN GPIO_PIN6       // GPIO_6
```

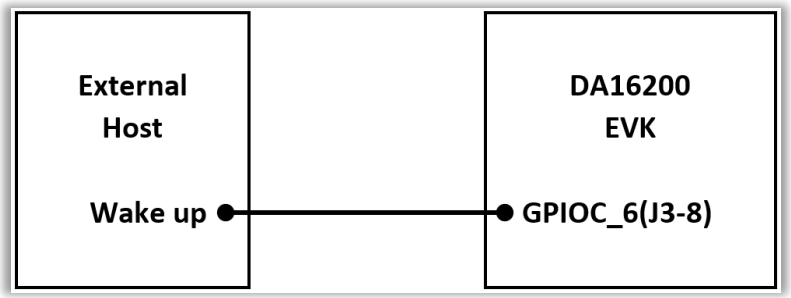

Figure 64. Another pin configuration for waking up host MCU

Table 4. GPIO pin configuration

Port	PIN Mux	GPIO
GPIO_UNIT_A	PIN_AMUX	GPIOA_0
		GPIOA_1
	PIN_BMUX	GPIOA_2
		GPIOA_3
	PIN_CMUX	GPIOA_4
		GPIOA_5
	PIN_DMUX	GPIOA_6
		GPIOA_7
	PIN_EMUX	GPIOA_8
		GPIOA_9
	PIN_FMUX	GPIOA_10
		GPIOA_11
GPIO_UNIT_C	PIN_UMUX	GPIOC_6
		GPIOC_7
		GPIOC_8

4.3.3 Programming Firmware Images for DA16200/DA16600

When using an EVB for the first time, the firmware must be updated to the latest version. For more details, see Ref [3]. After programming the firmware image, factory reset is required to enter the AWS IoT configuration setting mode. This can be done by pushing the "Factory_RST" button for 5 seconds as shown in Figure 65 and Figure 66.

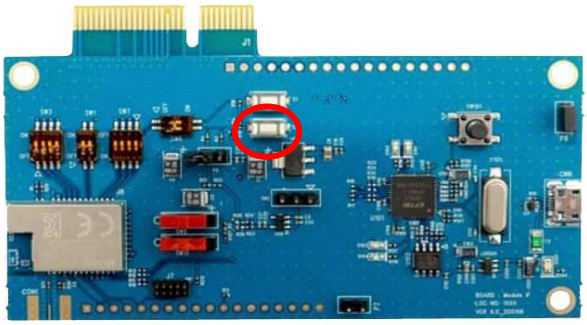

Figure 65. Factory reset button on DA16200 EVB

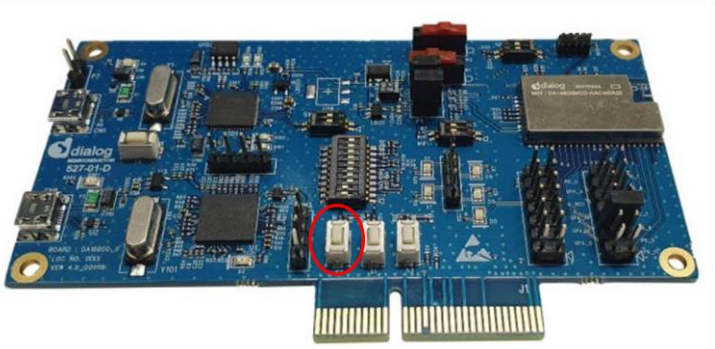

Figure 66. Factory reset button on DA16600 EVB

The logs from DA16200 are as follows:

```
[/DA16200]#
Factory reset ready.


Factory Reseting....


DA16200 concurrent factory reset AP mode = 1 ("AP_ONLY")....


....[app_set_customer_ap_configure] set AP config mode = 0


apps_reboot_ap_mode  Customer configuration ...
.
 default_ssid = "Dialog_DA16200" ..., ap_config_param->ssid_name


 PW = 1234567890


 PW = 1234567890  completed
.
apps_reboot_ap_mode  IPADDR_CUSTOMER...
....
apps_reboot_ap_mode  customer_dhcpd_flag == DHCPD_CUSTOMER..
......
OK
```

The logs from DA16600 are as follows:

```
[/DA16600]#
Factory reset ready.


Factory Reseting....
Set STA Mode ...


Rebooting....


Reset BLE ...


Wakeup source is 0x0
[dpm_init_retmemory] DPM INIT CONFIGURATION(1)


        ***************************************************
        *             DA16600 SDK Information
        * -------------------------------------------------
```

```
        *
        * - CPU Type        : Cortex-M4 (120 MHz)
        * - OS Type         : FreeRTOS 10.4.3
        * - Serial Flash    : 4 MB
        * - SDK Version     : V3.2.8.0 AWS-ATCMD Doorlock Ref. QFN GEN
        * - F/W Version     : FRTOS-GEN01-01-f017bfdf51-006558
        * - F/W Build Time  : Sep  5 2023 17:17:05
        * - Boot Index      : 0
        *
        ******************************************************

gpio wakeup enable 00000402
[combo] dpm_boot_type = 0

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
[combo] BLE_BOOT_MODE_0
[combo] BLE FW VER to transfer ....
>>> v_6.0.14.1114.3 (id=1) at bank_1
[combo] BLE FW transfer done

System Mode : Station Only (0)
>>> Start DA16X Supplicant ...
>>> DA16x Supp Ver2.7 - 2022_03
>>> MAC address (sta0) : d4:3d:39:40:72:16
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)

>>> UART2 : Clock=80000000, BaudRate=115200
>>> UART2 : DMA Enabled ...
[UART ready notification]
<<< GAPM_DEVICE_READY_IND
AWS IoT dev_name="DA16200", len=7
IoT dev_name="DA16200", len=7
[combo] Advertising...

[/DA16600] #
```

After the factory reset, the DA16200/DA16600 is now ready to enter the AWS IoT Configuration Settings.

4.3.4 Configure Components for Testing

The following information are required for testing the application with AWS IoT server:

- Unique thing name

The information can be set in the source code for host MCU or using the provided scripts in the downloaded package. For how to run the macro script, see Ref. [3]. The scripts are located in the following location:

\DA16x00_img\script\doorlock.ttl

```
;In order to use this script on DA16200, the console should be prompt
;after setting the DA16200 to STA mode, SNTP client enable, and no DPM mode in easy setup through the console.
;set configurations with DA16200's console

;set features
sendln "user"
;set board type
sendln "SET APP_BOARD_FEATURE EVK"
mpause 400
;set your thingname
;sendln "SET APP_THINGNAME FAE-DOORLOCK-4”
mpause 400
;set broker address
sendln "SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com"
mpause 400
```

The MCU source code can be found in the following file:

\MCU\RA6M4\Src\atcmd\at_cmd.c.

```
#define MAX_RETRY_SEND_COUNT      10

/* AWS features, configurations, and certification keys */
const char* cmd_set_cfg[MAX_CFG_NUM] =
{
  "\r\nAT+"PLATFORM" SET AWS_USE_FP 0\r\n",
  "\r\nAT+"PLATFORM" SET APP_BOARD_FEATURE EVK\r\n",
  "\r\nAT+"PLATFORM" SET APP_THINGNAME FAE-DOORLOCK-4\r\n",
  "\r\nAT+"PLATFORM" SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com\r\n",
  "\r\nAT+"PLATFORM" SET APP_LPORT 1883\r\n",
  "\r\nAT+"PLATFORM" SET APP_SUBTOPIC /AppControl\r\n",
  "\r\nAT+"PLATFORM" SET APP_PUBTOPIC /DeviceControl\r\n",
  "\r\nAT+"PLATFORM" CFG 0 app_door 1 2\r\n",            /* mcu sub.       str */
  "\r\nAT+"PLATFORM" CFG 1 mcu_door 1 0\r\n",            /* mcu pub.       str */
  "\r\nAT+"PLATFORM" CFG 2 app_window 1 2\r\n",          /* mcu sub.       str */
  "\r\nAT+"PLATFORM" CFG 3 mcu_window 1 0\r\n",          /* mcu pub.       str */
  "\r\nAT+"PLATFORM" CFG 4 battery 0 1\r\n",             /* shadow  int */
```

```
  "\r\nAT+"PLATFORM" CFG 5 temperature 2 1\r\n",          /* shadow  float */
  "\r\nAT+"PLATFORM" CFG 6 doorStat 1 1\r\n",             /* shadow  str */
  "\r\nAT+"PLATFORM" CFG 7 windowStat 1 1\r\n",           /* shadow  str */
  "\r\nAT+"PLATFORM" CFG 8 app_shadow 1 2\r\n",           /* mcu sub.       str */
  "\r\nAT+"PLATFORM" CFG 9 mcu_shadow 1 0\r\n",           /* mcu pub.       str */
  "\r\nAT+"PLATFORM" SET SLEEP_MODE 3\r\n",
  "\r\nAT+"PLATFORM" SET USE_DPM 1\r\n",
  "\r\nAT+"PLATFORM" SET RTC_TIME 1740\r\n",
  "\r\nAT+"PLATFORM" SET DPM_KEEP_ALIVE 30000\r\n",
  "\r\nAT+"PLATFORM" SET USE_WAKE_UP 0\r\n",
  "\r\nAT+"PLATFORM" SET TIM_WAKE_UP 10\r\n",
  "\r\nAT+"PLATFORM" SET APP_MCU_WKAEUP_PORT GPIO_UNIT_A\r\n",    /* GPIO_UNIT_A or
GPIO_UNIT_C */
  "\r\nAT+"PLATFORM" SET APP_MCU_WKAEUP_PIN GPIO_PIN11\r\n"      /* GPIO_PIN0 ~ GPIO_PIN11
or GPIO_PIN6~GPIO_PIN8 */
};
```

4.3.5 Test without Host MCU

If the host MCU is not available, the AWS IoT commands can be tested with the script provided in the downloaded package.

Door lock for two-way communication:

`\DA16x00_img\script\doorlock.ttl.`

NOTE
The example script only supports initial value setting. To fully verify the operation of the AT commands, use the host MCU for interacting with the server and application.

4.3.6 Test with Host MCU

The e²studio is required for building source code for host MCU and programing the images to host MCU. Visit the Renesas website (https://www.renesas.com/us/en/software-tool/e-studio) for downloading and installing the e²studio. After installing the e²studio, complete the following steps for building and programming.

1. Import the `project` file to `\MCU\RA6M4\`.

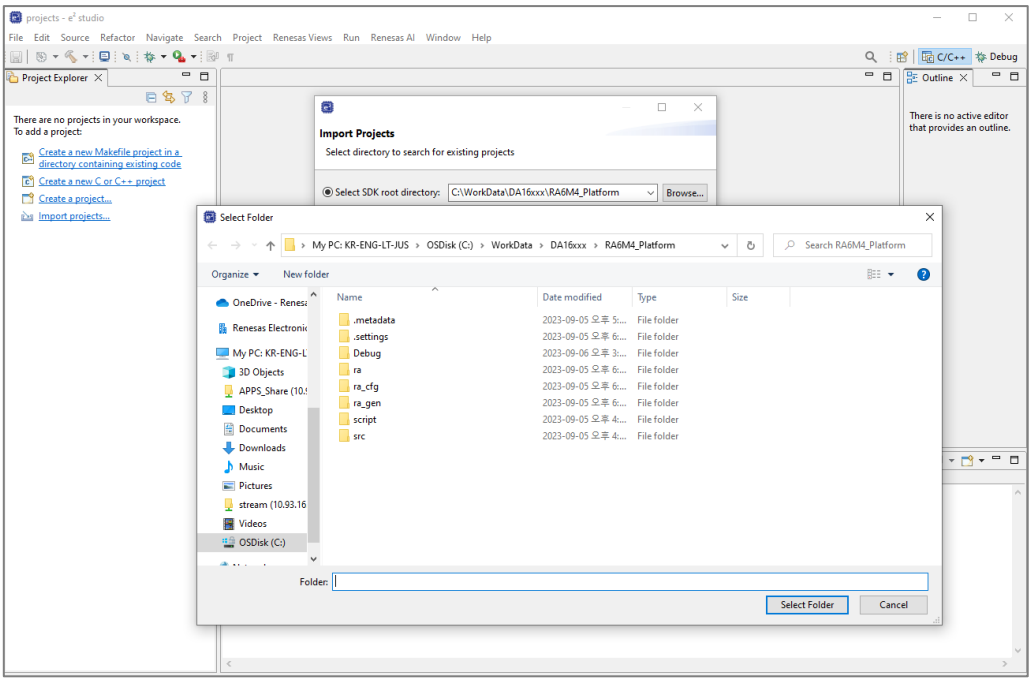

Figure 67. e²studio project file

NOTE
When connecting to the RA6M4 MCU for the first time or changing the configuration, complete the step 2 to set up the FSP configuration.

2. To set FSP configuration of the RA6M4 MCU, select **configurations.xml**.

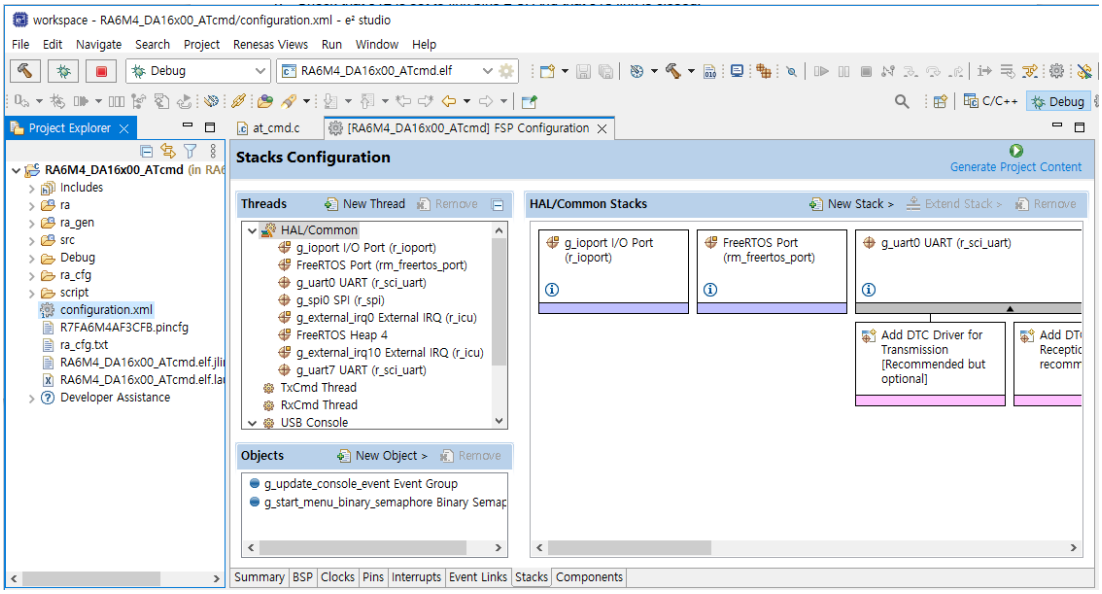

Figure 68. FSP configuration

3. Use the thing name received from the FAE to test without setting up a server.
4. Change the thing name to the received name.

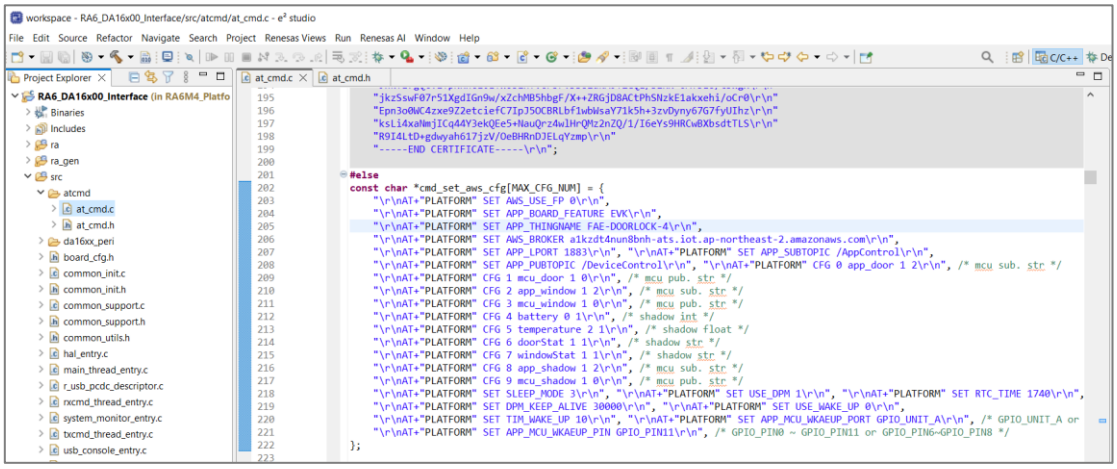

Figure 69. Thing name in MCU source code

5. To build a new project, select **project > Build Project**.

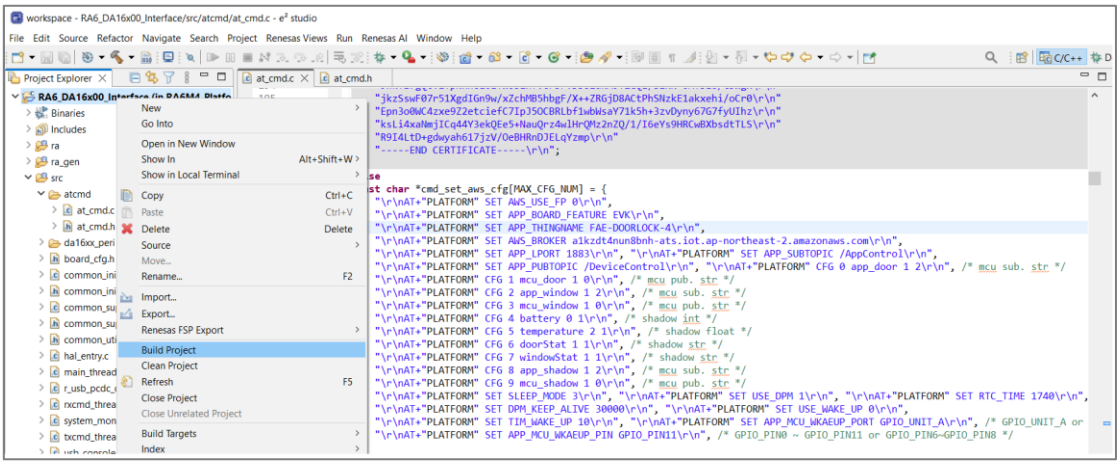

Figure 70. Build project

6. To set the connection to the RA6M4 MCU, select **Debug Configurations**.

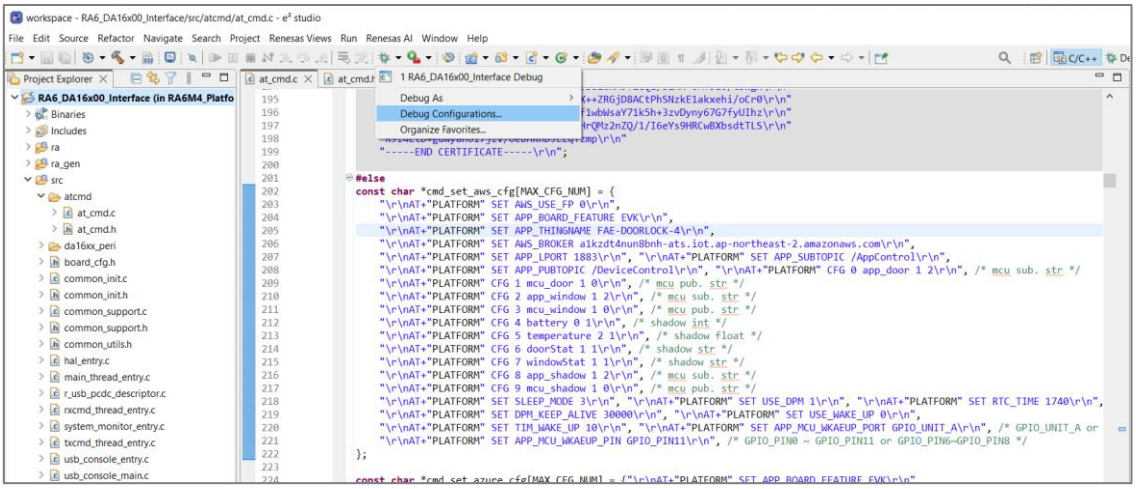

Figure 71. Debug configurations

7. On the **Debugger** tab, change the configuration as shown in Figure 72, and then click **Apply** > **Debug**.

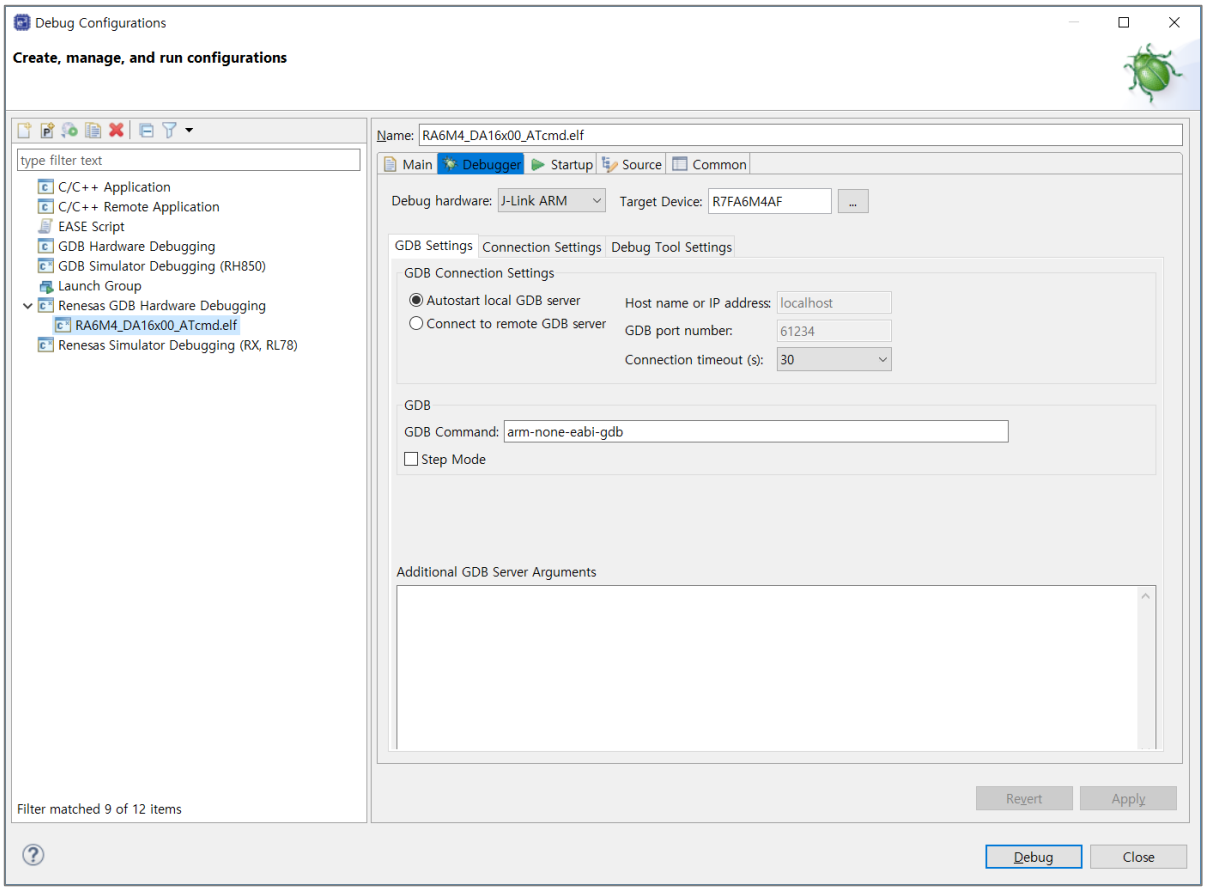

Figure 72. Set debug configurations

The following shows the console output of the DA16200 after a factory mode reset.

```
Soft-AP is Ready (d4:3d:39:10:d5:07)


>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
[UART ready notification]
[http_server_task] HTTP-Server Start!!
=========================================
[ AWS-IOT AT COMMAND ]
[ aws_shadow_dpm_auto_start]
AWS_IOT on Station Mode for "FAE-DOORLOCK-4"
=========================================
[pal_app_dpm_auto_start] mcu_wakeup_port=-1, mcu_wakeup_pin=0x0
default set to mcu_wakeup_port=0, mcu_wakeup_pin=0x800


Root CA: X
Certificate: X
Private Key: X
```

```
nvram read string(thingname) error
invalid APP feature...can't start APP Platform thread...check again
.. UART ready
```

The following shows the console output of the DA16200 when setting the AWS IoT configuration via AT commands from an MCU.

```
================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CFG 3 mcu_window 1 0
================================================


================================================
Att[3] number   : 3
Att[3] name       : mcu_window
Att[3] data type: 1
Att[3] MQTT type: 0
================================================


================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CFG 4 battery 0 1
================================================


================================================
Att[4] number   : 4
Att[4] name       : battery
Att[4] data type: 0
Att[4] MQTT type: 1
================================================



================================================
argc num = 2
argv[0]: AT+AWS
argv[1]: CFG 5 temperature 2 1
================================================


================================================
Att[5] number   : 5
Att[5] name       : temperature
```

```
Att[5] data type: 2

Att[5] MQTT type: 1

================================================
```

The following shows the console output of the DA16200 after the Soft AP has been configured and it is waiting to be provisioned by the mobile application.

```
Soft-AP is Ready (d4:3d:39:10:d5:07)

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
[UART ready notification]
[http_server_task] HTTP-Server Start!!
========================================
[ AWS-IOT AT COMMAND ]
[ aws_shadow_dpm_auto_start]
AWS_IOT on Station Mode for "FAE-DOORLOCK-4"
========================================
[pal_app_dpm_auto_start] mcu_wakeup_port=0, mcu_wakeup_pin=0x800

Root CA: O
Certificate: O
Private Key: O

subscribe index=0, name=app_door
subscribe index=2, name=app_window
newNode index=4
newNode index=5
newNode index=6
newNode index=7
subscribe index=8, name=app_shadow
shadow item count = 4, (integer#=1, string#=2, float#=1)
current shadowConut = 4
pkey=windowStat, pdata=test
current shadowConut = 3
pkey=doorStat, pdata=test
current shadowConut = 2
pkey=temperature, pdata=16.500000
current shadowConut = 1
pkey=battery, pdata=2700

AWS_IOT AP Mode  FAE-DOORLOCK-4

+ATPROV=STATUS 1
================================================
[Start Provisioning with TCP/TLS] .. Soft AP Mode
================================================
[app_provision_switch_client_thread] Create...(status=0) [10]
[app_provision_TCP_server_thread] Create ...
[app_provision_TLS_server_thread] Create TLS...

>>> Start Provisioning Server (TLS) ...
Wait Accept (TLS)...
[app_find_home_ap] Wi-Fi Scan request success.
[app_find_home_ap:518] (0) iptime_justin / 3 / -34 / 2447
[app_find_home_ap:518] (1) AP-101-201 / 3 / -66 / 2432
[app_find_home_ap:518] (2) SK_WiFiGIGA551A_2.4G / 3 / -78 / 2422
[app_find_home_ap:518] (3) SK_WiFiGIGA551A / 3 / -79 / 2422
[app_find_home_ap:518] (4) SK_WiFi3801 / 3 / -94 / 2412
[app_find_home_ap:518] (5) NIS-HomeAP11N / 0 / -74 / 2447
[app_provision_TCP_server_thread] socket().. status=1
 Wait Accept...
```

4.4 Mobile App Demo

Install the mobile application by searching for **DA16200** or **DA16600** on the Google Pay Store or the Apple App Store on the mobile devices.

4.4.1 Open Door

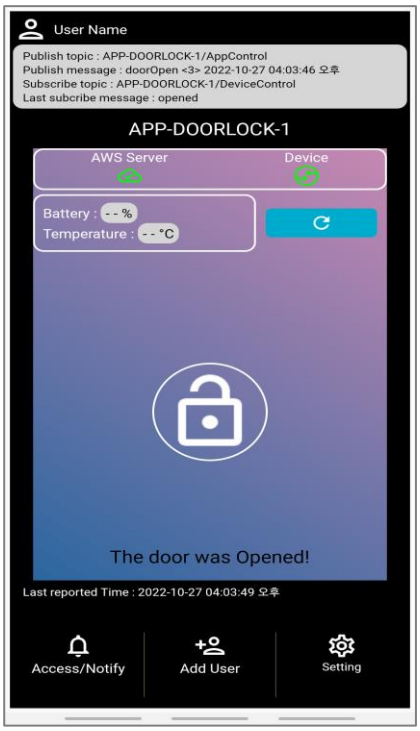

Figure 73. Opened status on application

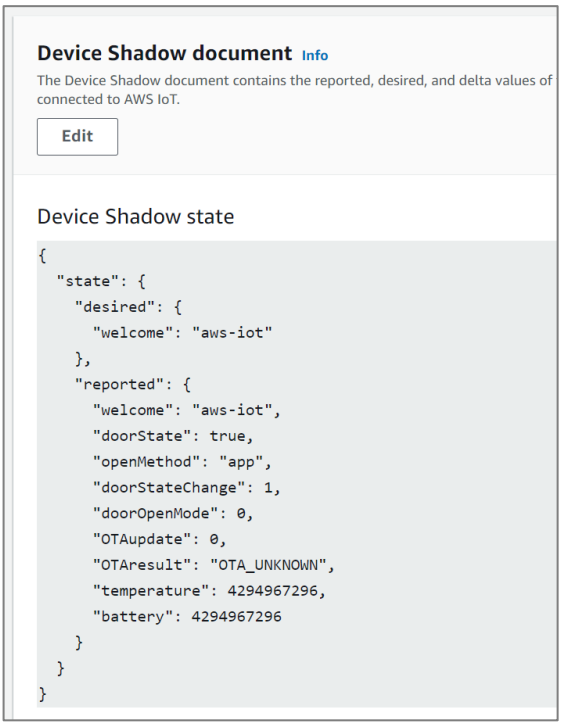

Figure 74. Opened status on AWS IoT console

- [Current Status]
  - **Opened**, Battery: _ _%, Temperature: _ _ °C (Real values are displayed on door lock ref. board)
  - Mobile APP (User): **Opened image button**
  - AWS (Server)
    - "doorState": **true**
    - "temperature": 4294967296
    - "battery": 4294967296

NOTE
A value of 4294967296 for the temperature or battery fields indicates the function is not available.

- DA16200 (Thing): The status of the device is displayed as shown in the **red text**.

```
INFO] [DoorLockDemo] [prvEventCallback:728]

Incoming Publish Topic Name: (Command) APP-DOORLOCK-1/AppControl matches subscribed topic.

Incoming Publish Message : doorOpen


open comm

[openControl]


[INFO] [DoorLockDemo] [controlDoorLock:1555] publish (command response) OK -  payload: "opened"

DEBUG:   [aws_dpm_app_door_work:1974] previous MQTT result = 0, doorLock CMD (=1: 0-idle, 1-open, 2-
close, 3-auto close)
```

```
================================================================================


[INFO] [DoorLockDemo] [aws_dpm_app_door_work:2030] publish (shadow doorlock update) OK -        payload:
"{"state":{"reported":{"doorState":true,"openMethod":"app","doorStateChange":1,"doorOpenMode":0,"OTAupdat
e":0,"OTAresult":"OTA_UNKNOWN"}}}"
********************************************************************************


last user Timer ID = 5
last doorOpenFlag state: "true"
last FOTA Stat: 0
last FOTA Url: ""
```

4.4.2 Close Door

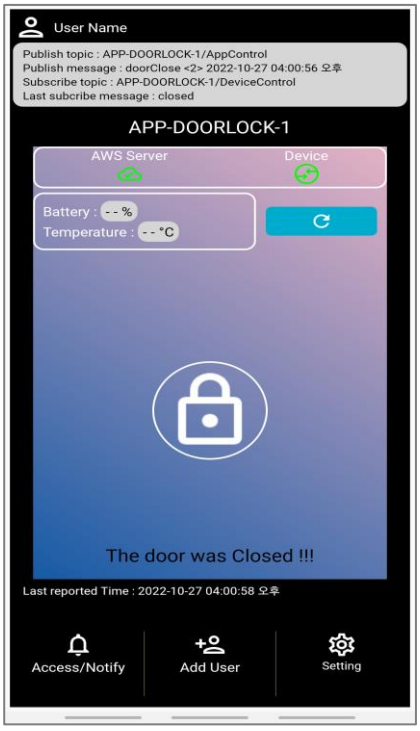

Figure 75. Closed status on application

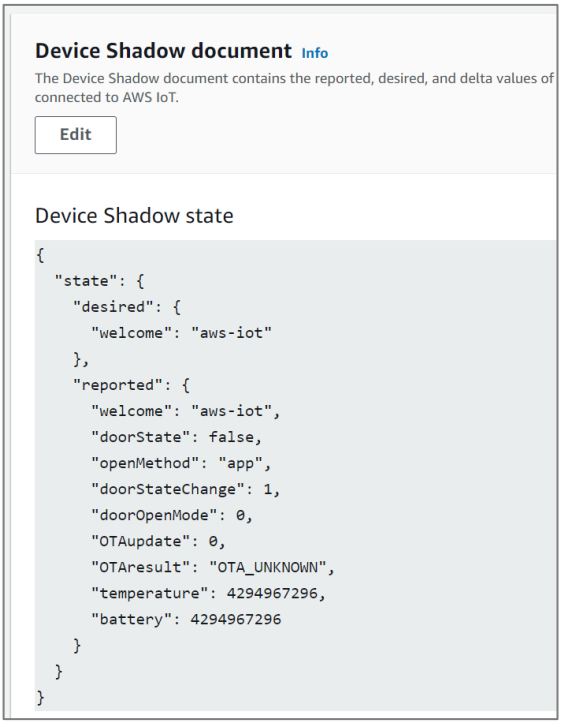

Figure 76. Closed status on AWS IoT console

- [Current Status]
  - **Closed**, Battery: _ _%, Temperature: _ _ °C (Real values are displayed on door lock ref. board)
  - Mobile APP (User): **Closed image button**
  - AWS (Server)
    - "doorState": **false**
    - "temperature": 4294967296
    - "battery": 4294967296

NOTE
A value of 4294967296 for the temperature or battery fields indicates the function is not available.

- DA16200 (Thing): The status of the device is displayed as shown in the **red text**.

```
[INFO] [DoorLockDemo] [prvEventCallback:728]
```

```
Incoming Publish Topic Name: (Command) APP-DOORLOCK-1/AppControl matches subscribed topic.
Incoming Publish Message : doorClose


close comm
[closeControl]


[INFO] [DoorLockDemo] [controlDoorLock:1555] publish (command response) OK -  payload: "closed"
DEBUG:   [aws_dpm_app_door_work:1974] previous MQTT result = 0, doorLock CMD (=2: 0-idle, 1-open, 2-
close, 3-auto close)


================================================================================


[INFO] [DoorLockDemo] [aws_dpm_app_door_work:2030] publish (shadow doorlock update) OK -        payload:
"{"state":{"reported":{"doorState":false,"openMethod":"app","doorStateChange":1,"doorOpenMode":0,"OTAupda
te":0,"OTAresult":"OTA_UNKNOWN"}}}"
********************************************************************************


last user Timer ID = 5
last doorOpenFlag state: "false"
last FOTA Stat: 0
```

5. OTA Update

Over the Air (OTA) is the process of updating the DA16200/DA16600 firmware image through Wi-Fi using an AWS S3 bucket.

Figure 77 shows the setting up process of the OTA update.

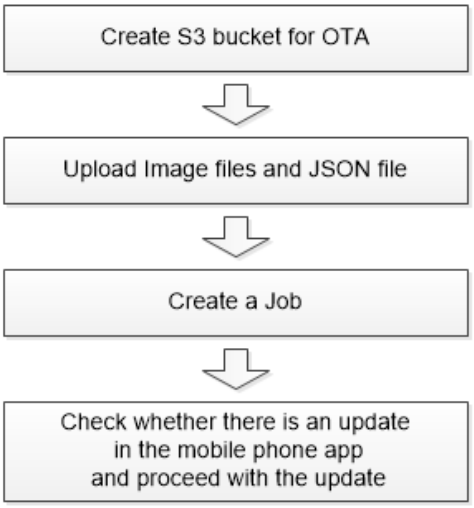

Figure 77. OTA update

5.1 Create S3 Bucket

For OTA update, create a new bucket in S3:

1. In the Amazon S3 console, click **Create bucket**.

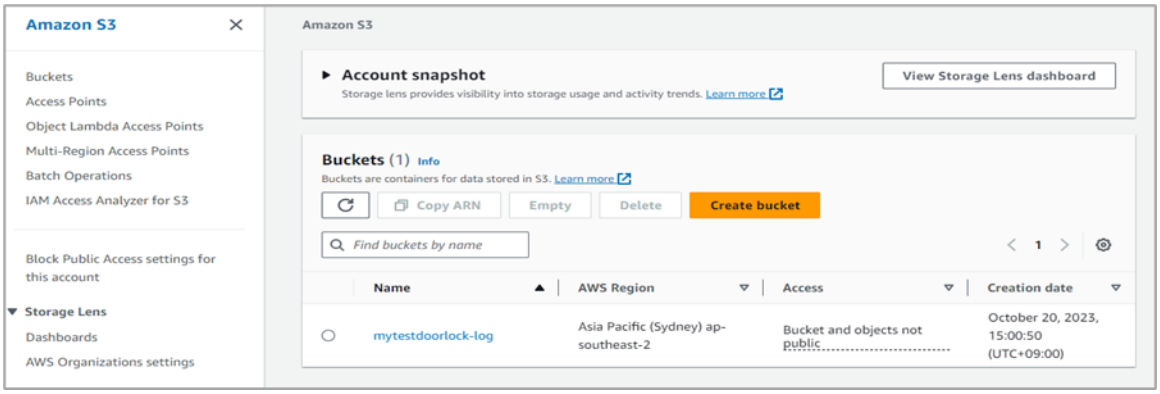

Figure 78. Create bucket for OTA update

2. Enter a Bucket name, apply the settings as shown in Figure 79–Figure 81, and click **Create bucket**.

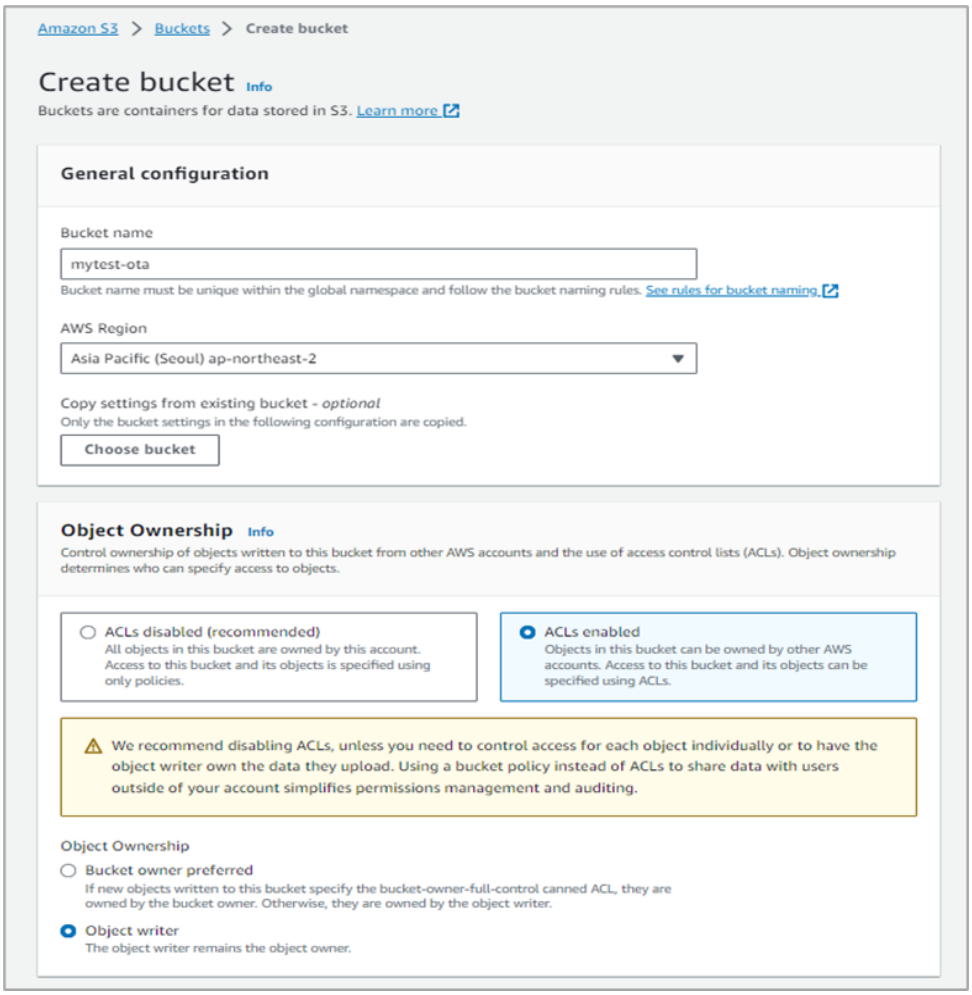

Figure 79. Bucket configuration – general and object ownership

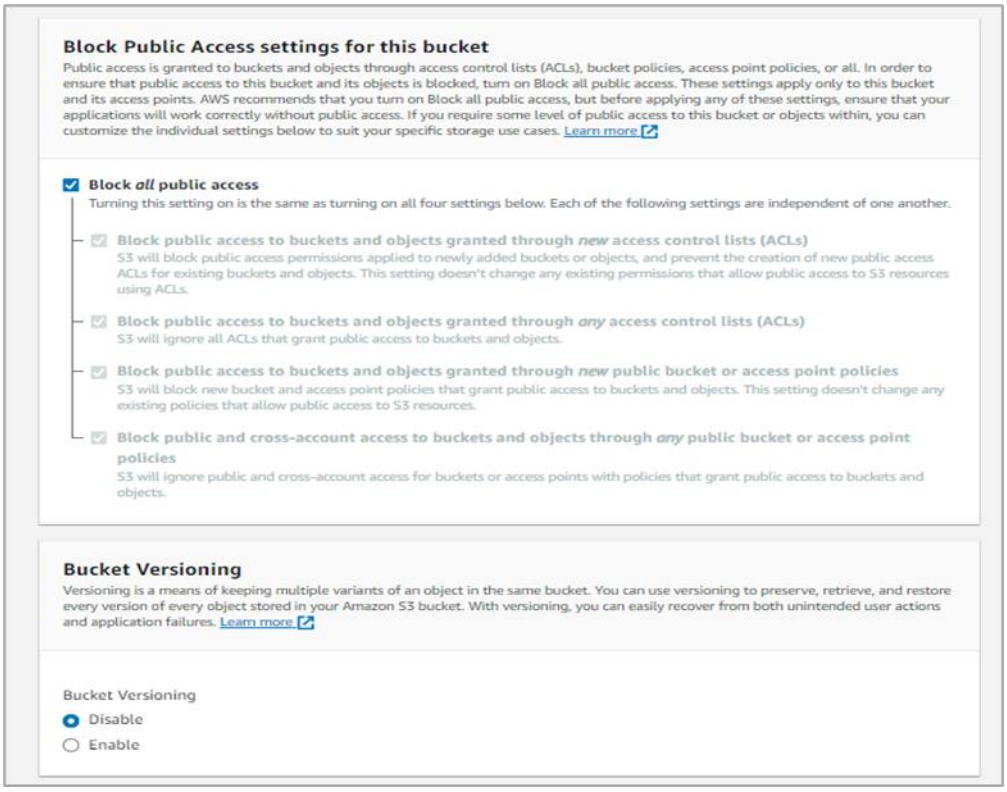

Figure 80. Bucket configuration – public access and versioning

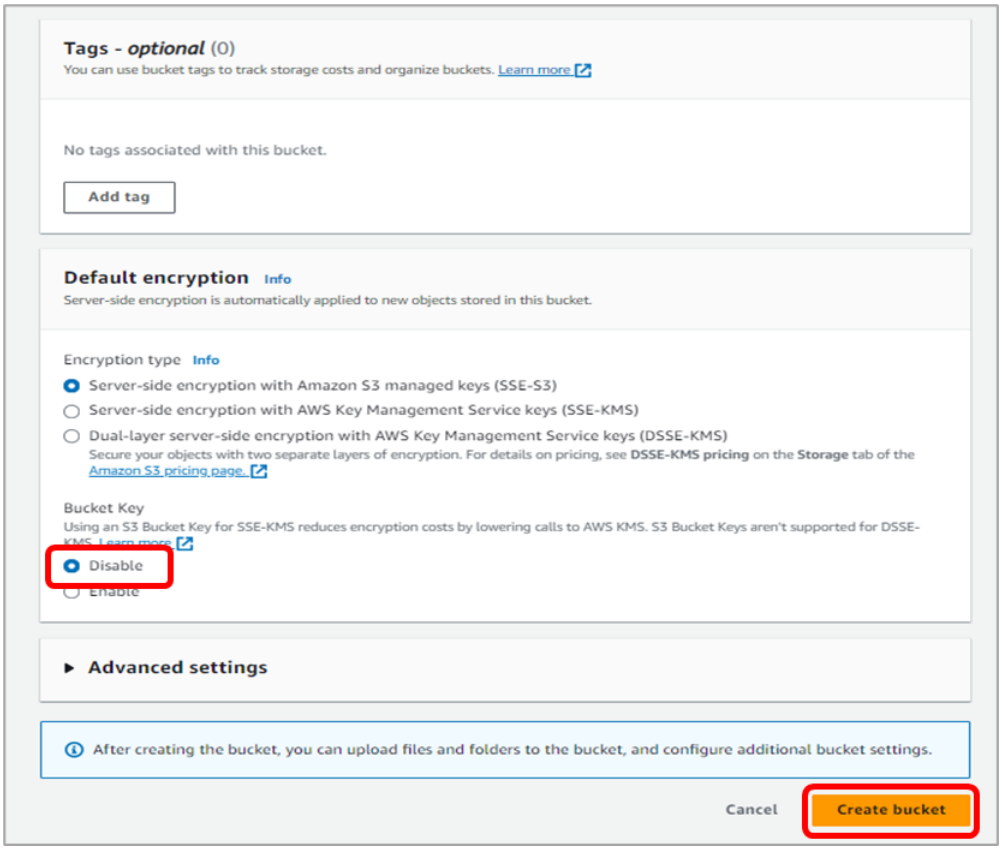

Figure 81. Bucket configuration – bucket key

3. Select the created bucket in the Buckets list.

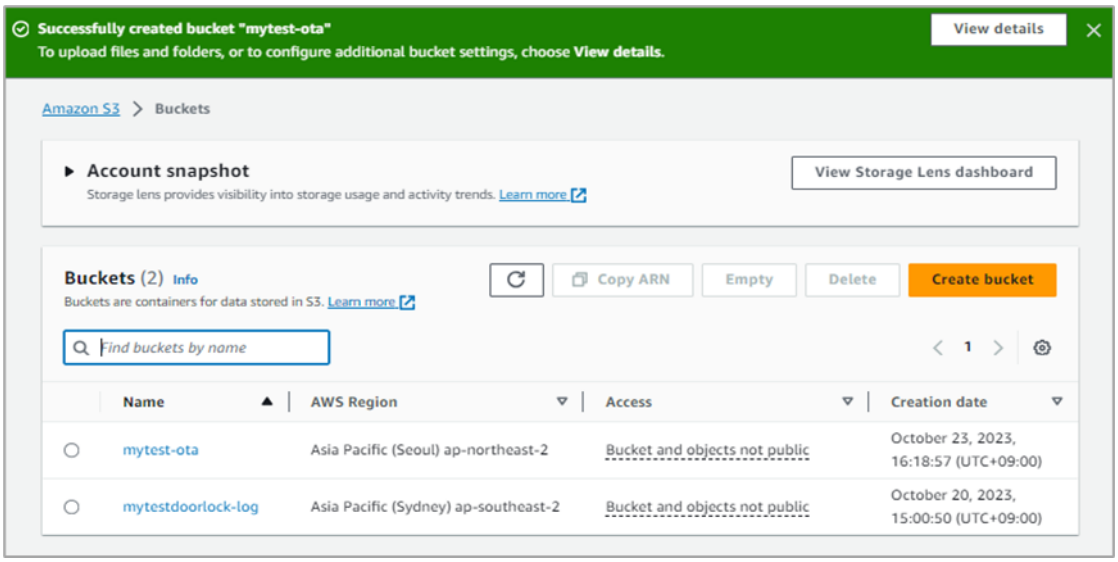

Figure 82. Created buckets for OTA

4. Click the **Permissions** tab, and then click **Edit**.
   This bucket must be modified for public access in the next step.

NOTE
Use public buckets for development environments only.

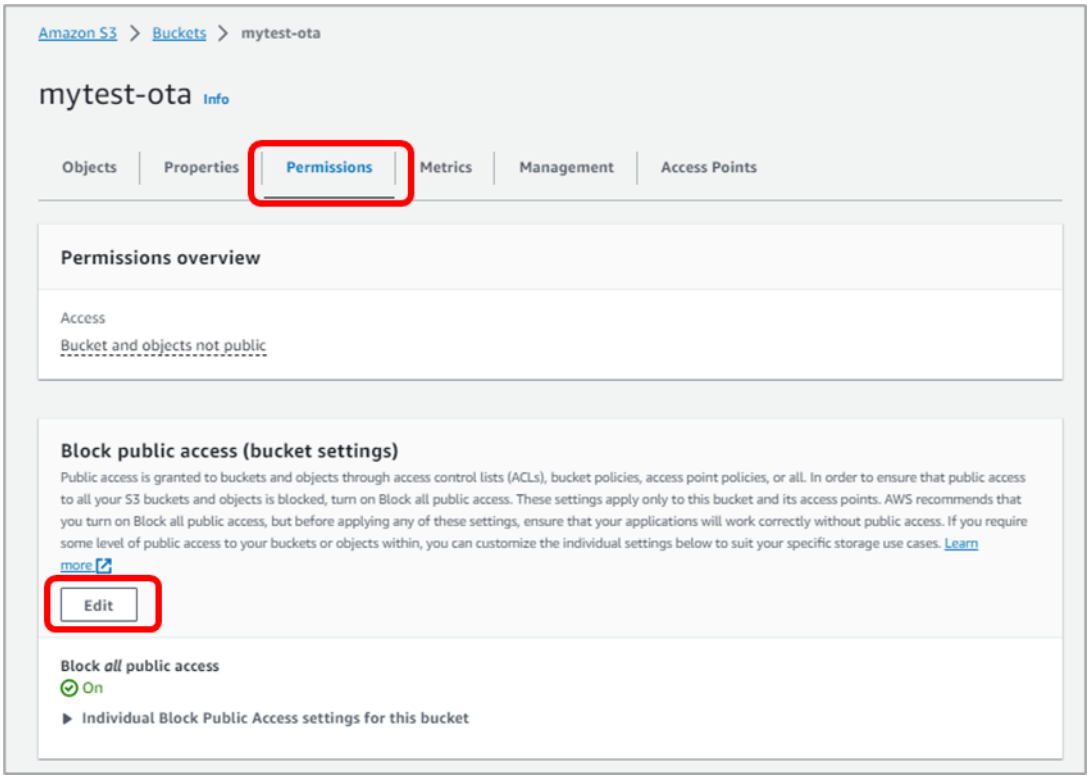

Figure 83. Edit bucket for public access

5. Clear all checkboxes, and then click **Save changes**.

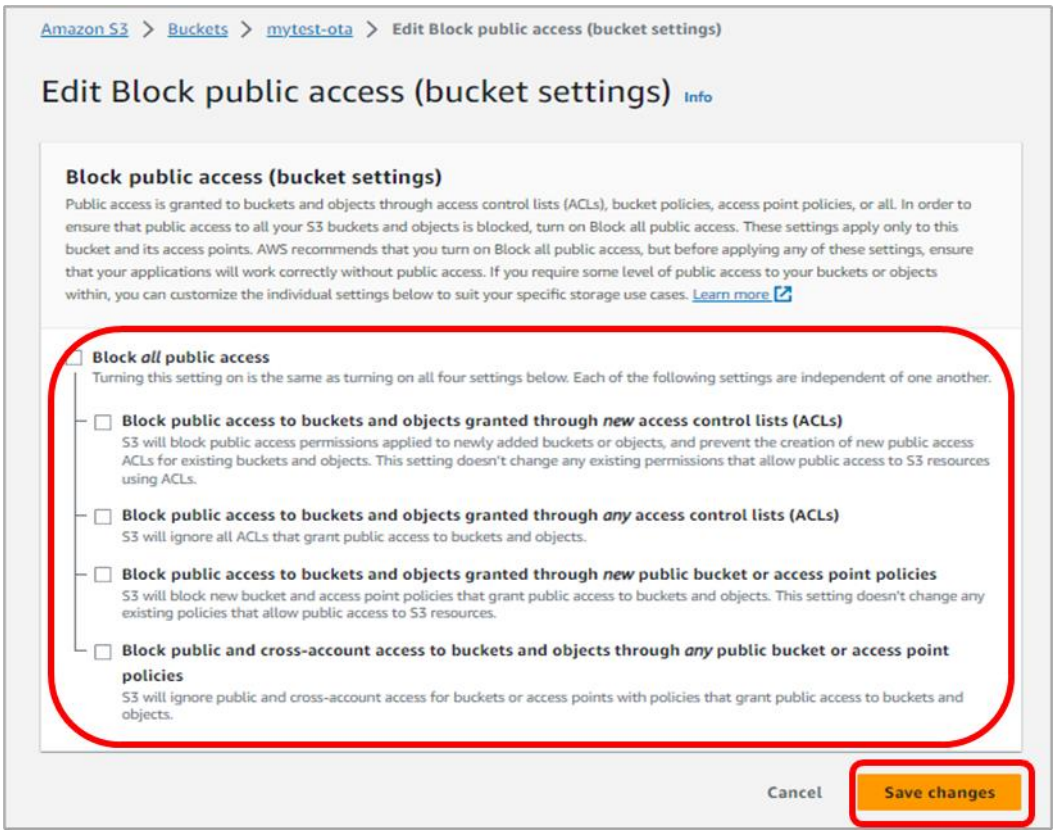

Figure 84. Public access settings for bucket

6. To save the settings, enter *confirm*, and then click **Confirm**.

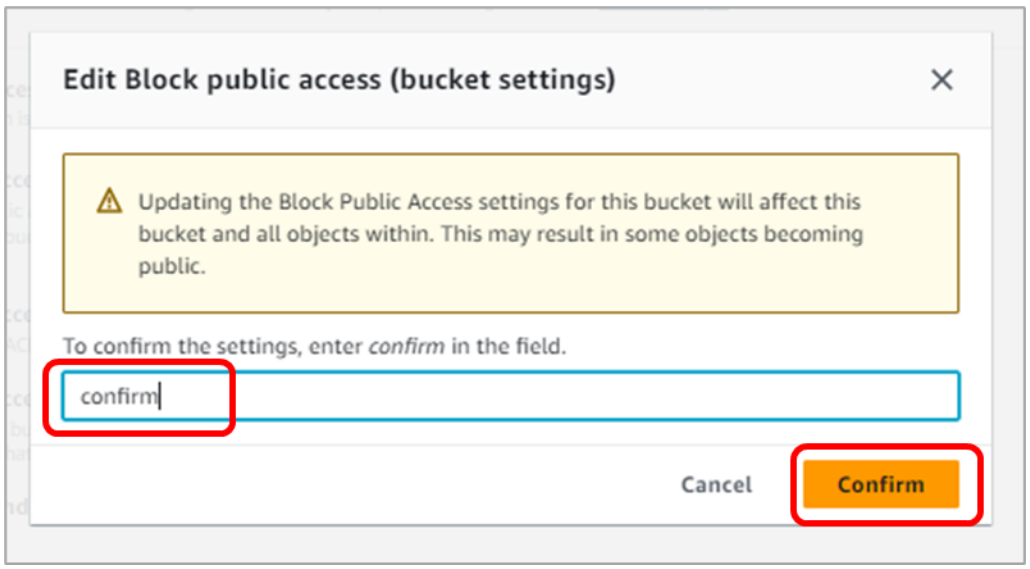

Figure 85. Confirm settings

7. On the **Permissions** tab, verify that all block options of public access are off.

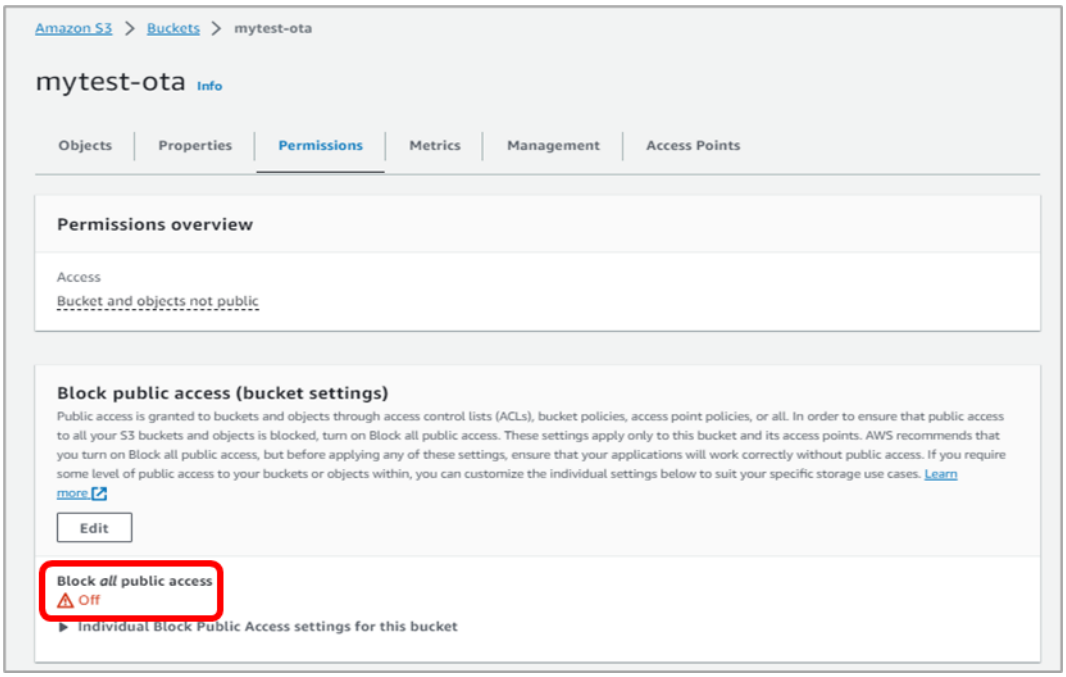

Figure 86. Settings updated

8. Click **Access Control List (ACL)** > **Edit** and next to **Everyone**, select **Read** Bucket ACL, and then click **Save changes**.

NOTE
On how to avoid ACLs, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/about-object-ownership.html

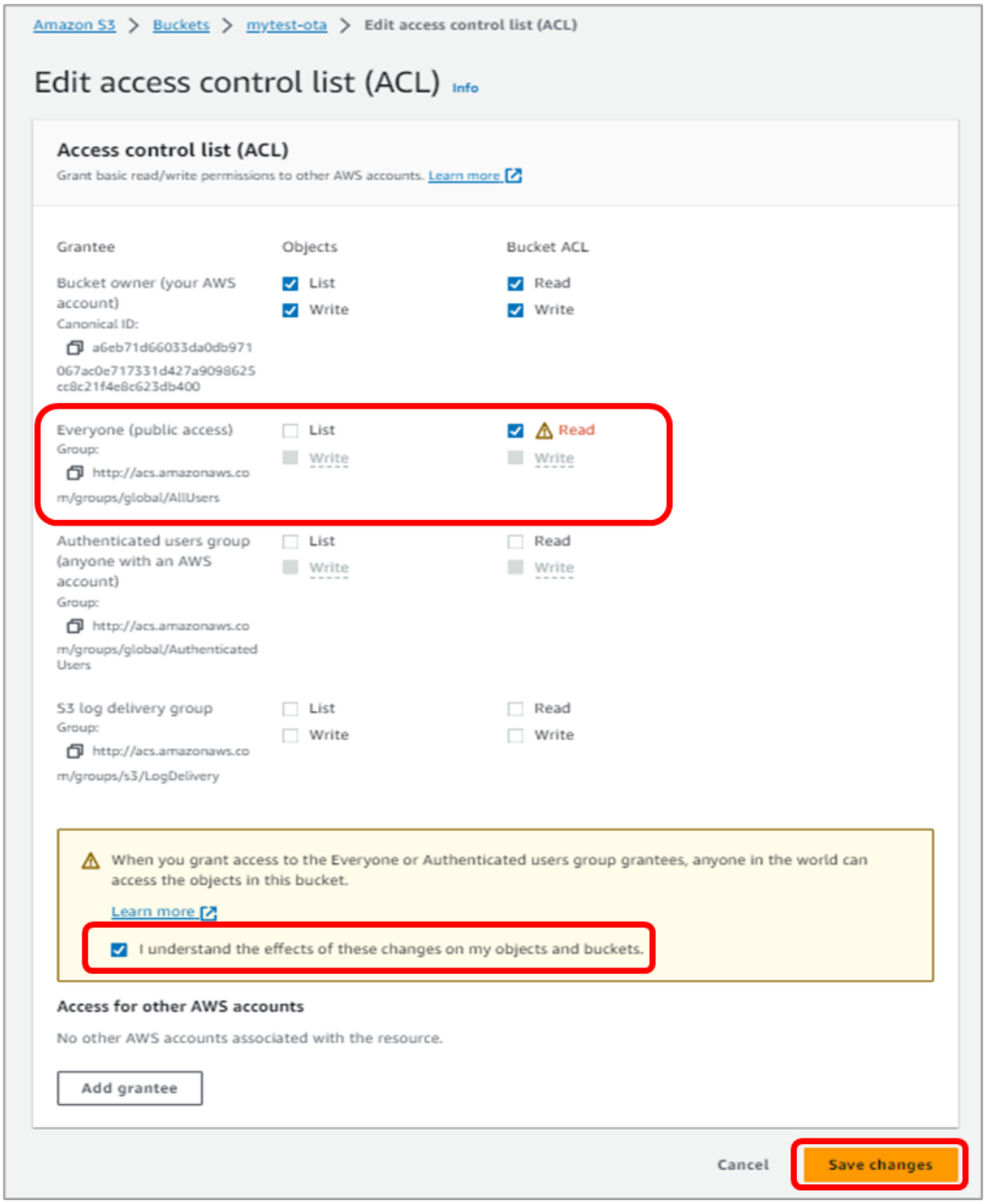

Figure 87. Public access for everyone

9. The bucket policy must be added as shown in Figure 88 and Table 5.
   "User Bucket Name" in Table 5 is the name of the S3 bucket created for an OTA update.

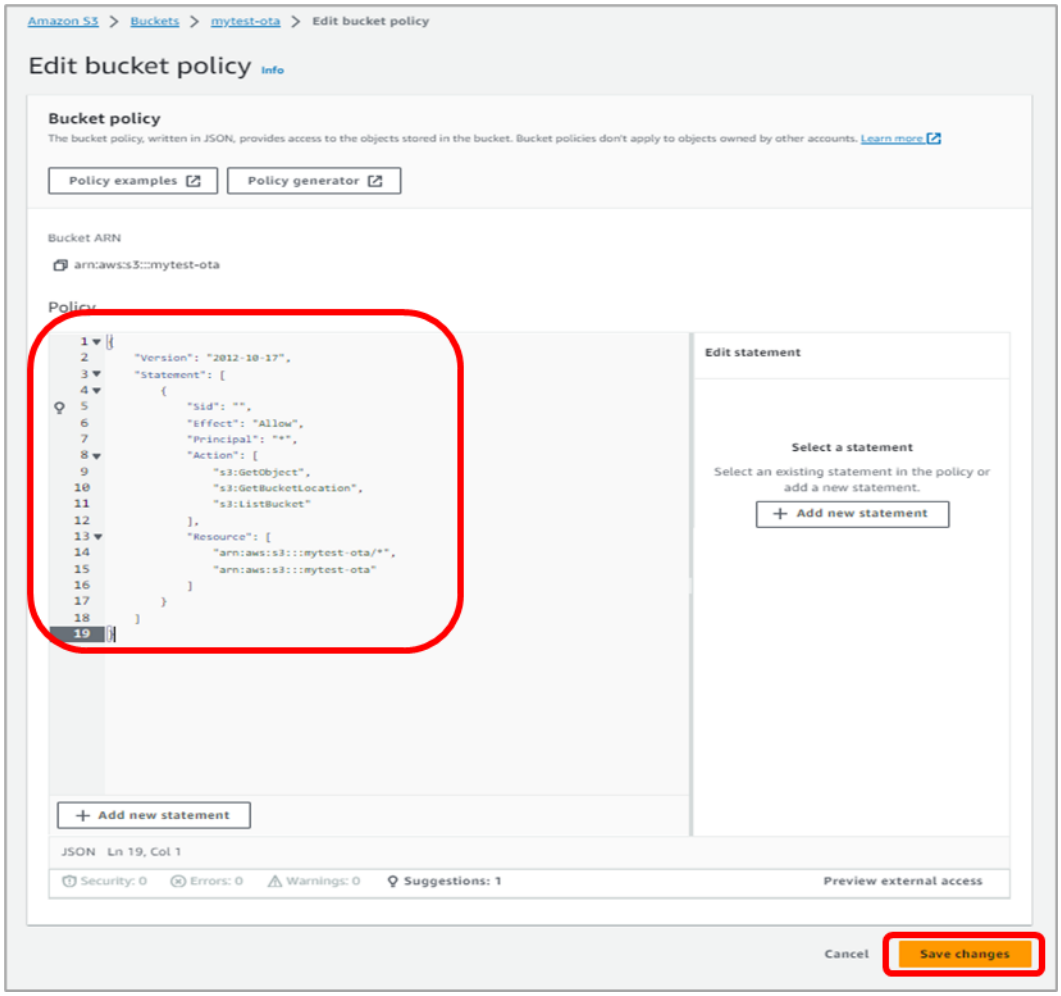

Figure 88. Bucket policy editor

Table 5. Bucket policy JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::User Bucket Name/*",
```

```
                "arn:aws:s3:::User Bucket Name"
            ]
        }
    ]
}
```

5.2 Upload Image File and JSON File

1. Rename the image files as follows:
   - RTOS Image: DA16200_FRTOS-GEN01.img
2. To be able to upload the Image files and JSON file for an OTA update, click **Upload**.

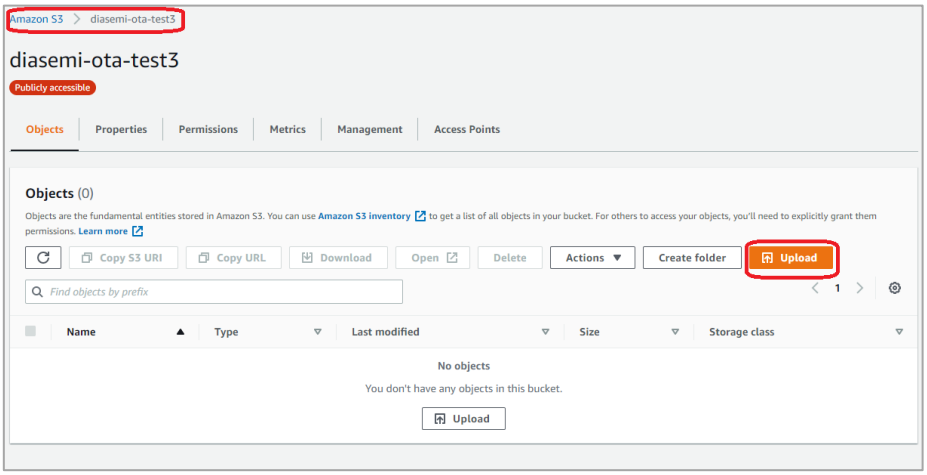

Figure 89. Upload files

3. Drag and drop or add files to upload.
4. There is one IMG file for a DA16200 OTA update, and the JSON file is a path setting file for the update. The important thing is that the names of the two files for the update should be the same as in Figure 90.

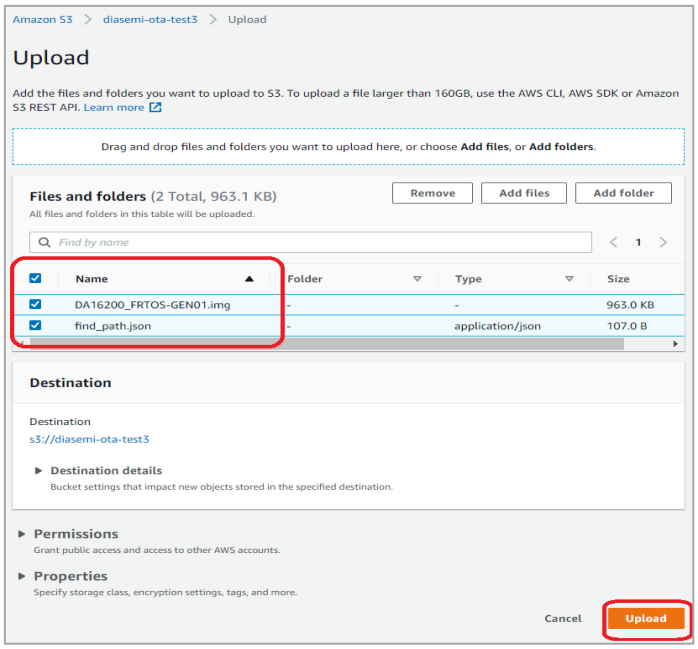

Figure 90. Ready to upload

The JSON information for an OTA update is as follows:

```
{ "operation":"install",
"Source":"https:// User Bucket Name.s3.ap-northeast-2.amazonaws.com/" }
```

"User Bucket Name" is the name of the S3 bucket created for an OTA update.

The URL policy of the "Source" can be changed by AWS.

5. Click the uploaded file name to check it.

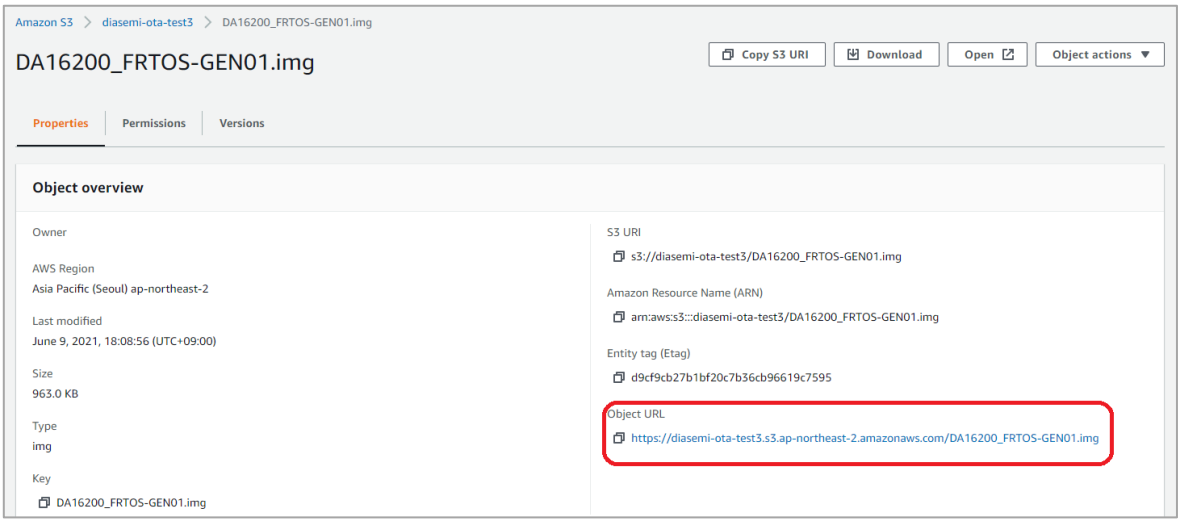

Figure 91. URL of source

6. Check if the files are uploaded correctly. You can delete and/or re-upload files to the bucket on the **Actions** tab.

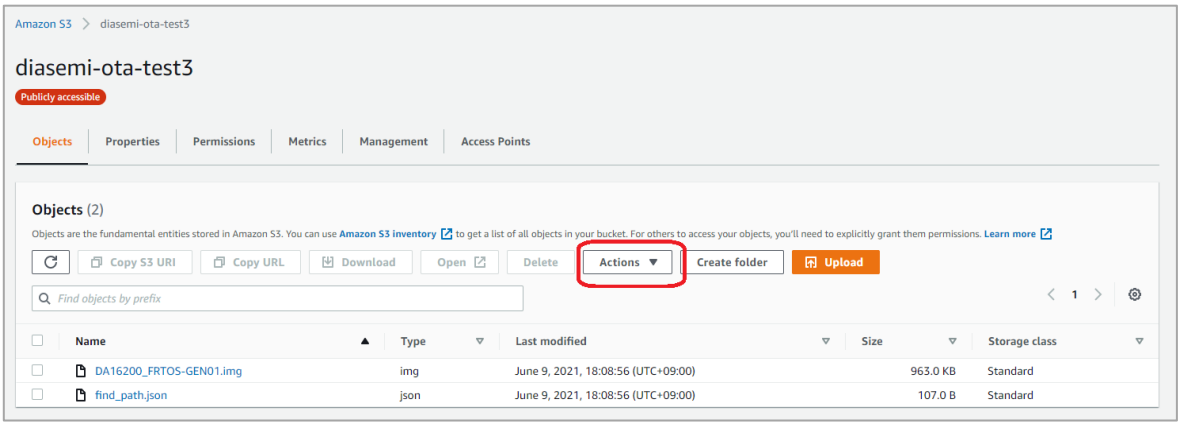

Figure 92. Uploaded files

As a result, a publicly accessible bucket is created.

Figure 93. Completed setup for OTA update

5.3 Create Job

AWS IoT Jobs is a service that allows you to define a set of remote operations that are sent to and executed on one or more devices connected to AWS IoT.

For an OTA update, go to the **IoT Core** service page in AWS Management Console. OTA is the process of replacing a product with a newer version of the same product. A Job must be created and registered to do an OTA update. It is a task to access the file uploaded to the bucket of the S3 service. If the server operator registers this Job at the desired time, the test thing proceeds with the OTA update.

1. In the AWS Management Console, go to **IoT core** > **Manage > Remote Actions** > **Jobs,** and click **Create job**.Figure 94. Create job

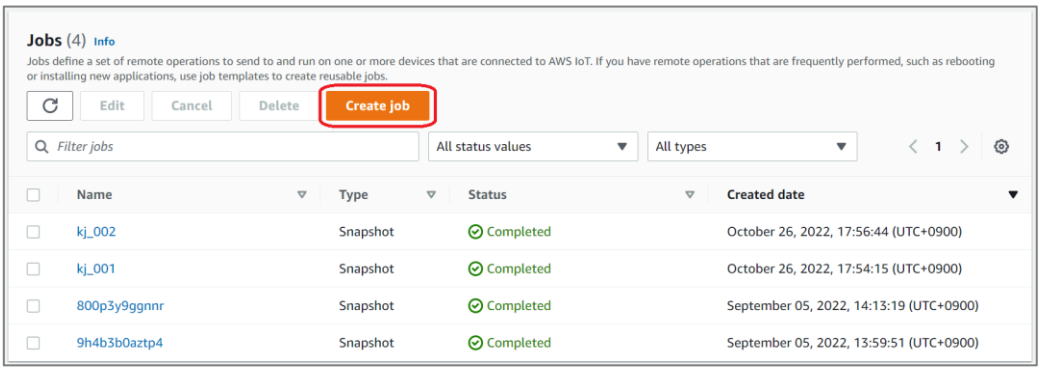

Figure 94. Create job

2. Select **Create custom job** and click **Next**.

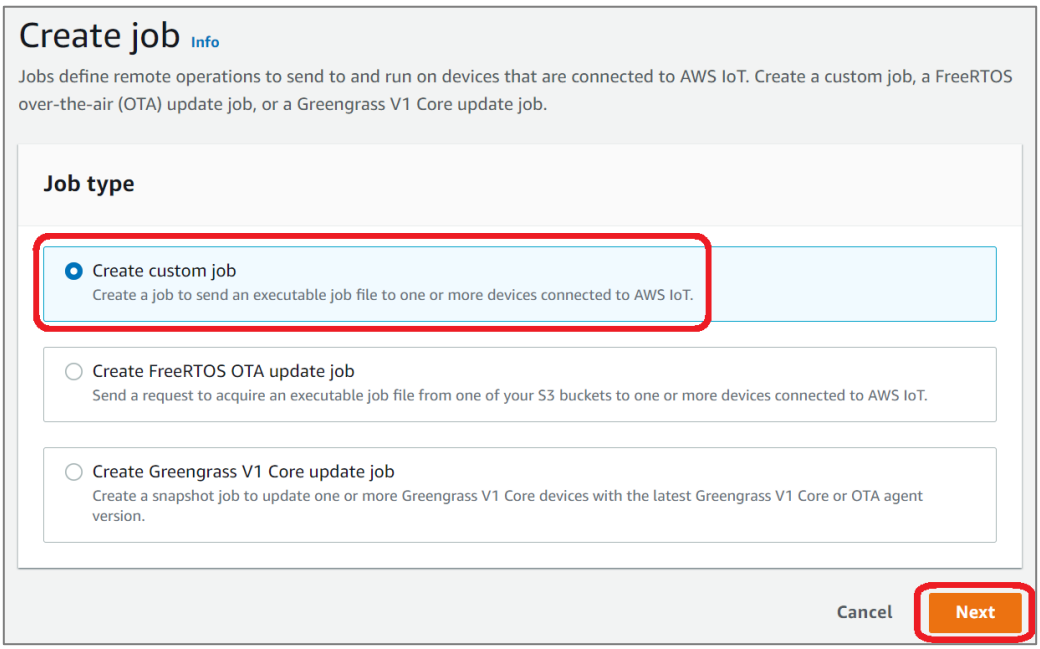

Figure 95. Create custom job

3. In the **Name** field, enter the job name and click **Next**.

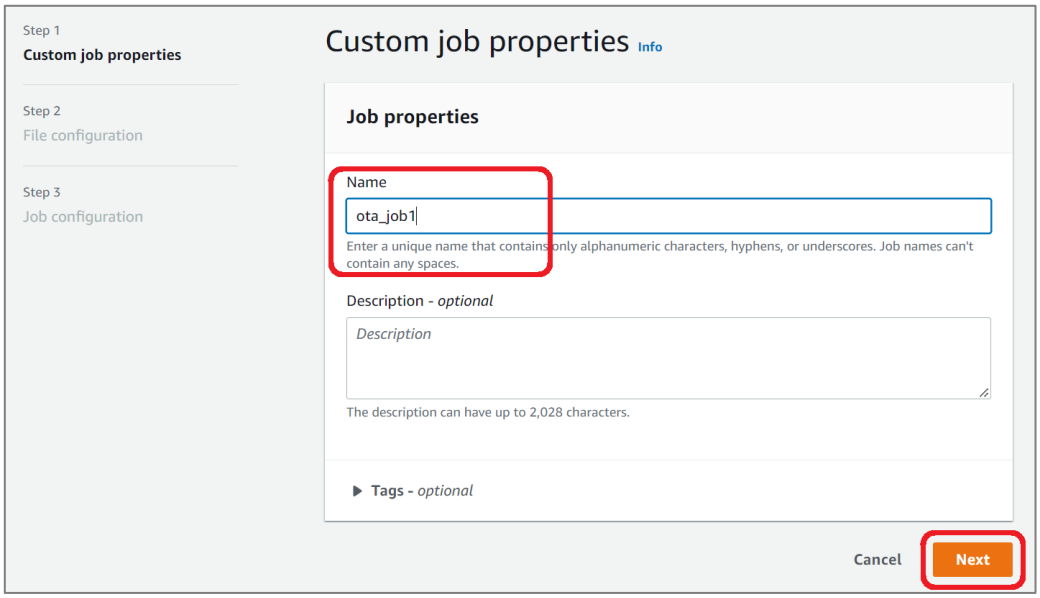

Figure 96. Enter job name

4. Select the devices to update. The thing to select is available in the list of options.

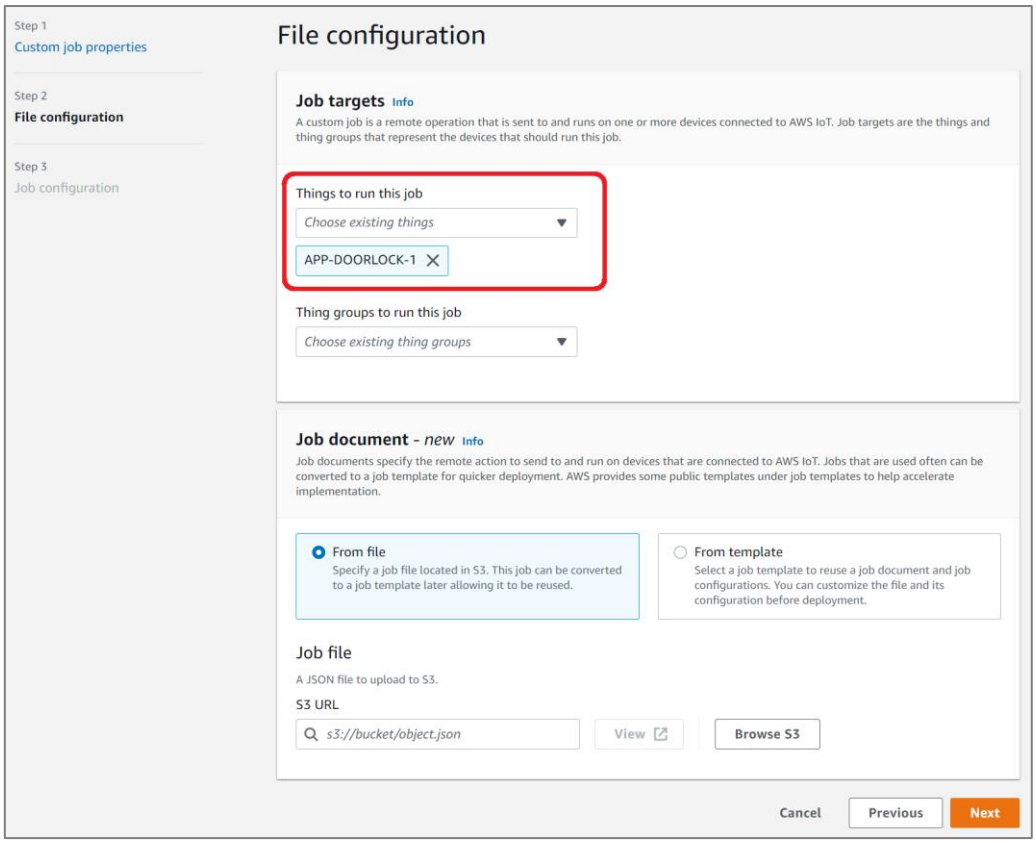

Figure 97. Select thing for OTA update

5. Under **Job file**, click **Browse S3** and select the S3 URL and click **Next**.

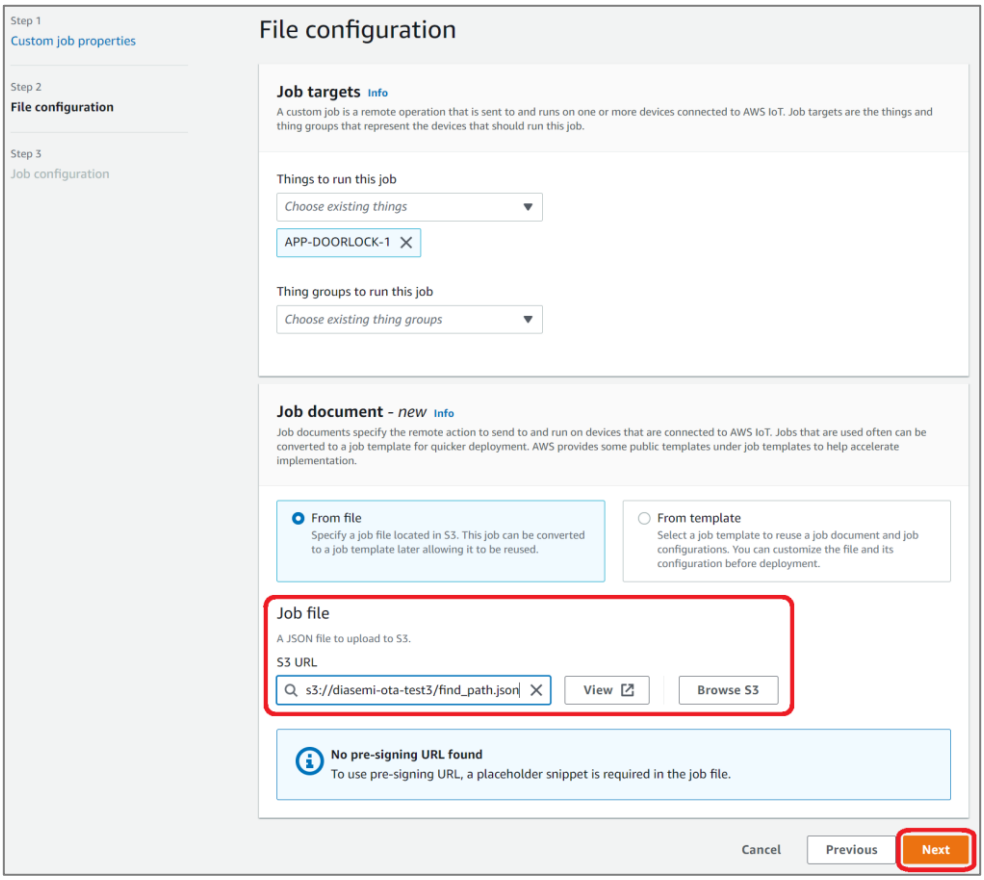

Figure 98. Select JSON for OTA update

6. Under **Job run type**, select **Snapshot** and click **Submit**.

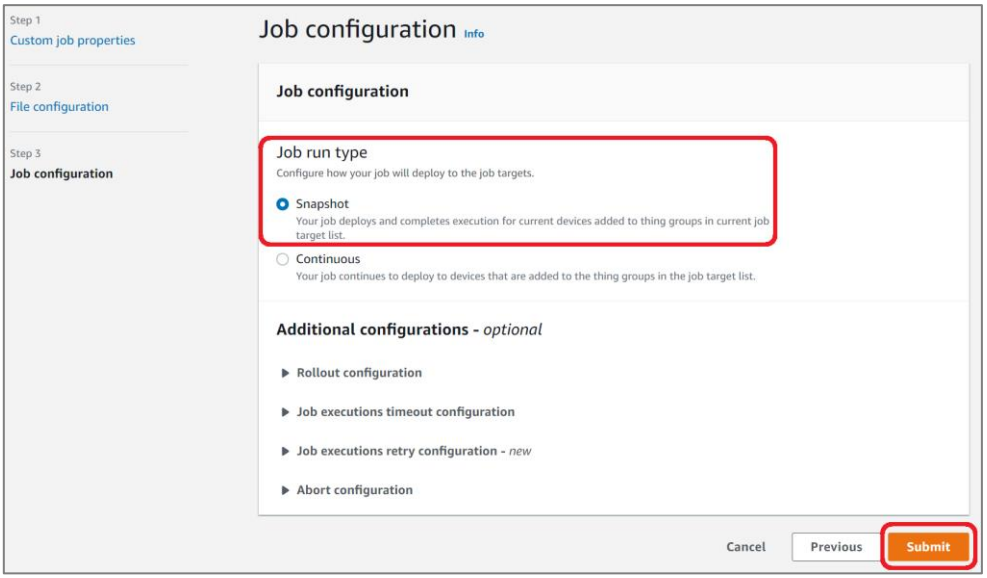

Figure 99. Job run type

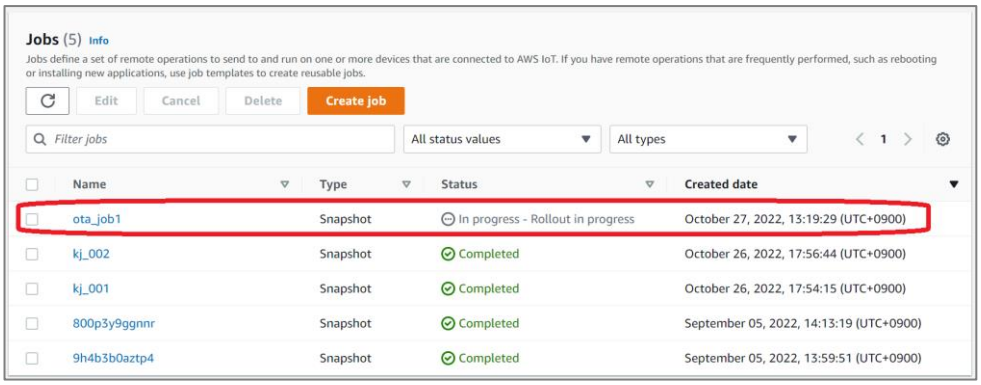

Figure 100. Job being created

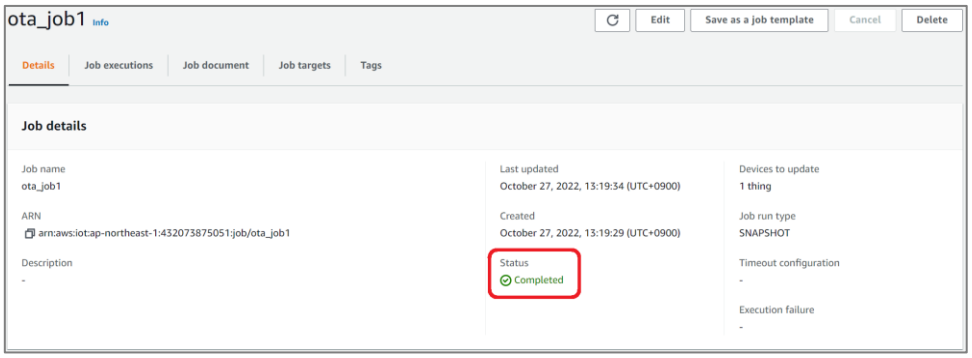

Figure 101. Successfully created job

5.4 Execute OTA Update

When a job is created successfully, the device receives the job details as follows:

```
[dpmAPPManager] DM_NEED_CONNECTION

DM_NEED_CONNECTION


[INFO] [DoorLockDemo] [aws_dpm_app_connect:2267] Establishing MQTT session with provisioned
certificate...

recv timeout(=2000 ms) set OK (socket=0)

hostName = "a1kzdt4nun8bnh-ats.iot.ap-northeast-1.amazonaws.com", flag to re-query (=0)

host IP from RTM = "54.178.218.11"

TCP connection OK to "a1kzdt4nun8bnh-ats.iot.ap-northeast-1.amazonaws.com"

[INFO] [DoorLockDemo] [aws_dpm_app_connect:2317] Sucessfully established connection with provisioned
credentials.

[Make AWS-Thing-Name]

[NVRAM] AWS Thing name : [APP-DOORLOCK-1] (len=14)

[NVRAM] [APP-DOORLOCK-1/DeviceConnect][APP-DOORLOCK-1/AppControl][APP-DOORLOCK-1/DeviceControl]

[INFO] [DoorLockDemo] [aws_dpm_app_subscription:1939] subscription info: total(default:4, tried:4), OK(4)

current RTM user Timer ID = 5

current RTM temperature(str): 0.000000

current RTM battery(str): 0.000000

current RTM doorOpen state: "false"

current RTM doorOpenMode : 0

current RTM FOTAFlag: 1

current RTM FOTA url : "https://diasemi-ota-test3.s3.ap-northeast-2.amazonaws.com/"

[dpmAPPManager] DM_RTC_WAKEUP

DM_WAKEUP_TIMER (tid=5)


DEBUG:   [aws_dpm_app_sensor_work:2104] read values from sensor if available


================================================================================

recv timeout(=120 ms) set OK (socket=0)

[INFO] [DoorLockDemo] [aws_dpm_app_sensor_work:2162] publish (shadow sensor update) OK -        payload:
"{"state":{"reported":{"doorState":false,"temperature":4294967296.000000,"battery":4294967296.000000}}}"

********************************************************************************


last temperature: Not available

last battery: Not available

Sleep mode 3: KA timer interval(=1800 sec)

DM_FINISH_DEVICE


recv timeout(=20 ms) set OK (socket=0)

[dpm_keepalive_timer_register] RTC interval (=1780 secs), mode (=0)

>>> Start DPM Power-Down !!!
```

Note

- When a Job for an OTA update is created, you can see the URL of the S3 bucket accessed through JSON in the console. Also, the setting icon changes in the Mobile application. See the console message and Figure 102.
- The temperature and battery value displayed as 4294967296 indicates that it is not available.

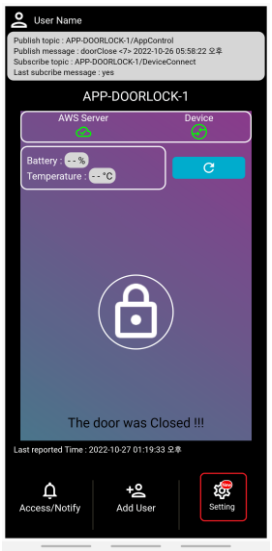

Figure 102. Successful job for OTA update in mobile app

The update is executed when you click the **Update** button on the Setting screen. The console and the Android application show the progress status during the OTA update. When the update is completed, the thing restarts and in the Android device, the update notification disappears (Figure 103).

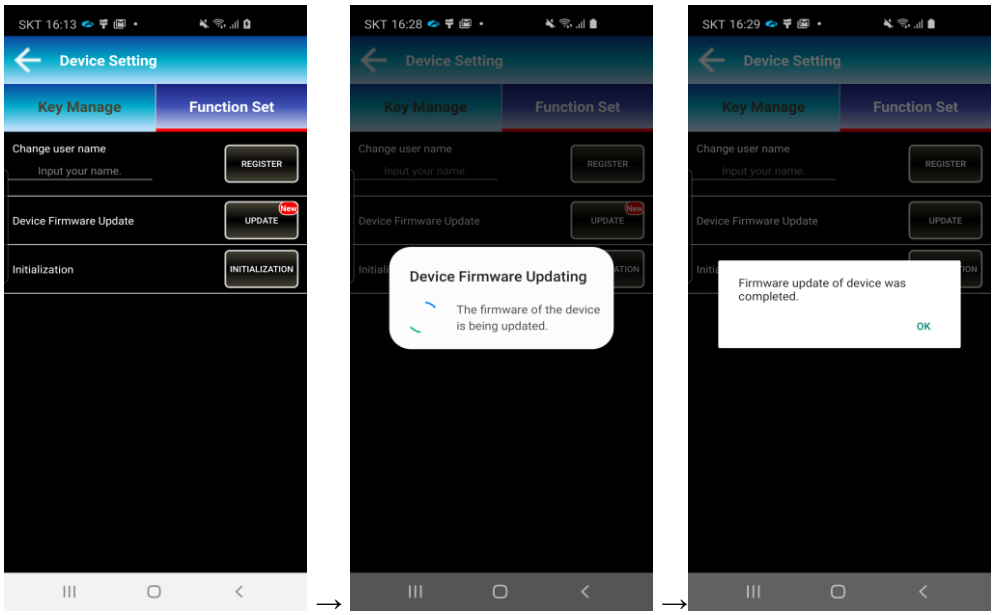

Figure 103. Execute OTA update in Android app

The following example shows the console message when an update is being performed.

```
last user Timer ID = 5
last doorOpenFlag state: "false"
last FOTA Stat: 2
last FOTA Url: "https://diasemi-ota-test3.s3.ap-northeast-2.amazonaws.com/"
URL for updating: "https://diasemi-ota-test3.s3.ap-northeast-2.amazonaws.com/"
```

```
save URL info & reboot for OTA

Wakeup source is 0x0

...

...

...

DEBUG:   [aws_ota_fw_update:3532]  RTOS url https://diasemi-ota-test2.s3.ap-northeast-
2.amazonaws.com/DA16200_FRTOS-GEN01.img

>>> SNTP Server: pool.ntp.org (106.247.248.106)

>>> SNTP Time sync : 2022.10.26 - 08:56:58

> Server FW version : FRTOS-GEN01-01-56c232799-004457

   >> HTTP(s) Client Downloading... 100 %(1202848/1202848 Bytes)

- OTA Update : <RTOS> Download - Success

DEBUG:   [app_ota_fw_download_complete_notify:3375] RTOS download finish. (0x00)

- OTA: Renewing with new F/W

- OTA: RTOS

        > Same Version : FRTOS-GEN01-01-56c232799-004457

>>> RTOS is updated and system reboots. (New boot_idx=0) !!!

DEBUG:   [app_ota_fw_renew_notify:3497] Succeeded to replace with new FW.

- OTA: Reboot after 0 secs ...

Wakeup source is 0x0

[dpm_init_retmemory] DPM INIT CONFIGURATION(1)
```

Appendix A Provisioning

The DA16200 supports a provisioning feature called Soft AP mode for an easy network configuration. Provisioning with the **mobile network data off** on your mobile phone and Wi-Fi turned on. When provisioning is complete, turn on your mobile data again. Figure 104 shows the workflow of the provisioning process.

- Press the **Factory Reset** button for about 5 seconds. Start the Android application and touch the **START** button to find the wanted AP.

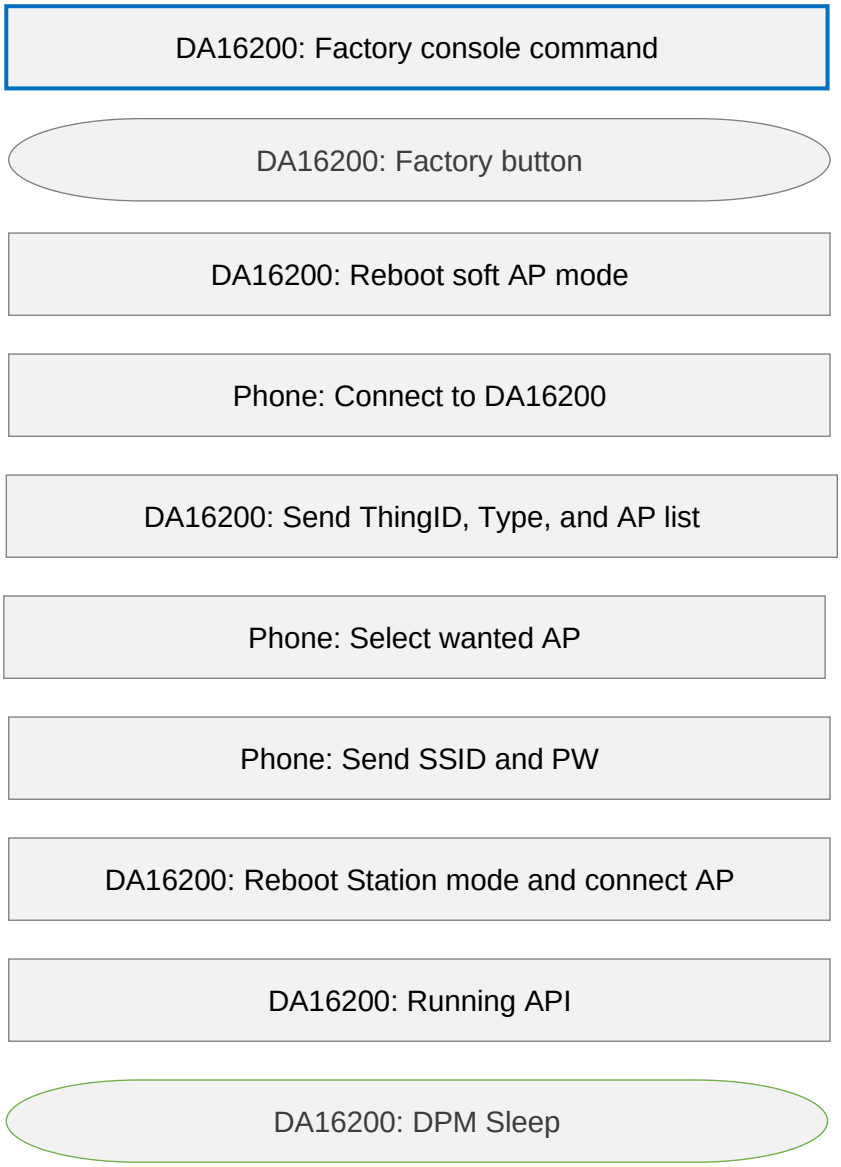

Figure 104. Provisioning flow

A.1 Android Application

```
System Mode : Soft-AP (1)

>>> DHCP Server Started
>>> Start DA16X Supplicant ...
>>> DA16x Supp Ver2.7 - 2022_03
>>> Add SoftAP Inteface (softap1) ...
>>> MAC address (softap1) : d4:3d:39:11:5e:73
>>> softap1 interface add OK
>>> AP Operating Channel: 1(2412)

>>> Network Interface (wlan1) : UP
BSS Isolate Disabled

Soft-AP is Ready (d4:3d:39:11:5e:73)

=======================================
[ APP-IOT Doorlock ]
[ aws_shadow_dpm_auto_start]
AWS_IOT on Station Mode for "APP-DOORLOCK-1"
=======================================

AWS_IOT AP Mode  APP-DOORLOCK-1

=================================================
[Start Provisioning with TCP/TLS] .. Soft AP Mode
=================================================

[app_provision_switch_client_thread] Create...(status=0) [10]
[app_provision_TCP_server_thread] Create ...
[app_provision_TLS_server_thread] Create TLS...

>>> Start Provisioning Server (TLS) ...
Wait Accept (TLS)...

> Wi-Fi Scan request success.
(0) KT_GIGA_2G_505 / 3 / -25 / 2412
(1) TP-LINK_AECC / 3 / -40 / 2412

[app_provision_TCP_server_thread] socket().. status=1
 Wait Accept...
```

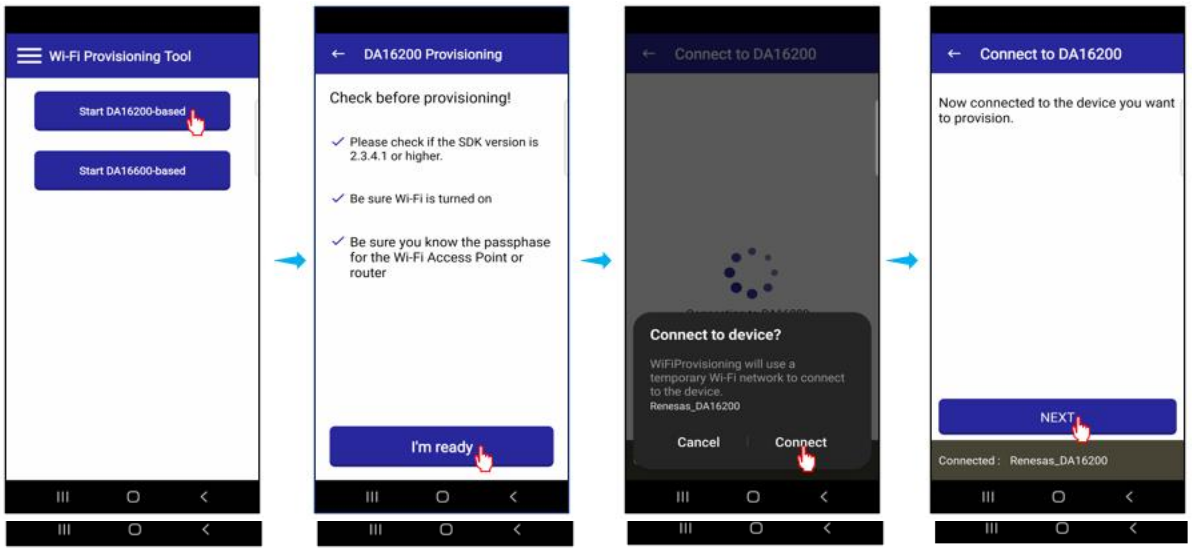

Figure 105. Provisioning from mobile app

```
[dpmAPPManager] DM_NEED_CONNECTION
DM_NEED_CONNECTION


[INFO] [DoorLockDemo] [aws_dpm_app_connect:2267] Establishing MQTT session with provisioned
certificate...
recv timeout(=2000 ms) set OK (socket=0)
hostName = "a1kzdt4nun8bnh-ats.iot.ap-northeast-1.amazonaws.com", flag to re-query (=0)
host IP = "52.69.14.255"
TCP connection OK to "a1kzdt4nun8bnh-ats.iot.ap-northeast-1.amazonaws.com"
recv timeout(=120 ms) set OK (socket=0)
[INFO] [DoorLockDemo] [aws_dpm_app_connect:2317] Sucessfully established connection with provisioned
credentials.
[Make AWS-Thing-Name]
[NVRAM] AWS Thing name : [APP-DOORLOCK-1] (len=14)
[NVRAM] [APP-DOORLOCK-1/DeviceConnect][APP-DOORLOCK-1/AppControl][APP-DOORLOCK-1/DeviceControl]
[INFO] [DoorLockDemo] [aws_dpm_app_subscription:1939] subscription info: total(default:4, tried:4), OK(4)
current RTM user Timer ID = 0
current RTM temperature(str): 0.000000
current RTM battery(str): 0.000000
current RTM doorOpen state: "false"
current RTM doorOpenMode : 0
current RTM FOTAFlag: 0
current RTM FOTA url : ""
[dpmAPPManager] DM_BOOT_WAKEUP
DM_WAKEUP_BOOT


[INFO] [DoorLockDemo] [connectionReadyInform:1598] publish (command response) OK -  payload: "yes"
[closeControl]
```

```
================================================================================

[INFO] [DoorLockDemo] [aws_dpm_app_door_work:2030] publish (shadow doorlock update) OK -        payload:
"{"state":{"reported":{"doorState":false,"openMethod":"app","doorStateChange":1,"doorOpenMode":0,"OTAupda
te":0,"OTAresult":"OTA_UNKNOWN"}}}"
********************************************************************************


last user Timer ID = 0

last doorOpenFlag state: "false"

last FOTA Stat: 0

last FOTA Url: ""


Sleep mode 3: KA timer interval(=1800 sec)


DM_FINISH_DEVICE


recv timeout(=20 ms) set OK (socket=0)

[dpm_keepalive_timer_register] RTC interval (=1780 secs), mode (=0)

>>> Start DPM Power-Down !!!
```

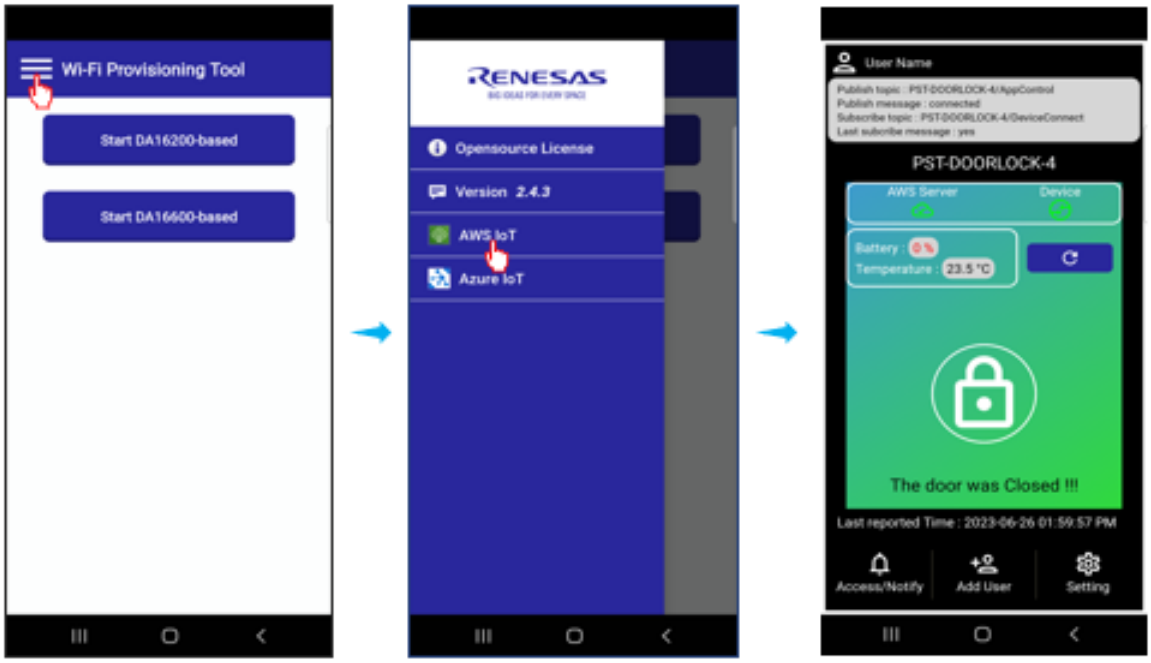

Figure 106. Running AWS IoT application from mobile app

Appendix B AT Commands for AWS IoT

B.1 Operating Modes

There are three operating modes:

- Setting Mode for features configuration.
- Provisioning Mode for network connection.
- Communication Mode for running.

B.1.1 Setting Mode

After uploading the image and rebooting, the DA16200/DA16600 enters Setting mode. In this mode, all AWS IoT settings can be configured using the SET command and a specific topic can be configured using the CFG command. For proper operation of AWS IoT, the TLS certificate keys must be set. All configuration data is stored before calling the factory reset command (Figure 107).

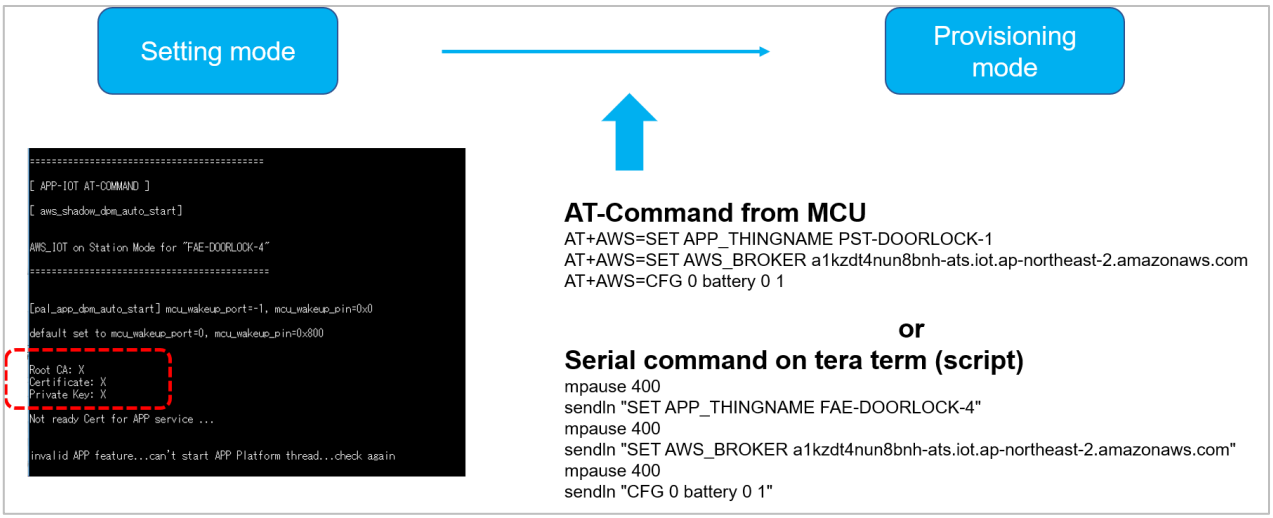

Figure 107. Setting mode

B.1.2 Provisioning Mode

In provisioning mode, the DA16200/DA16600 can be provisioned using an Android or iOS device. During provisioning, the MCU only receives a report on the provisioning status. When provisioning is complete, the DA16200/DA16600 enters Communication mode automatically after rebooting (Figure 108).

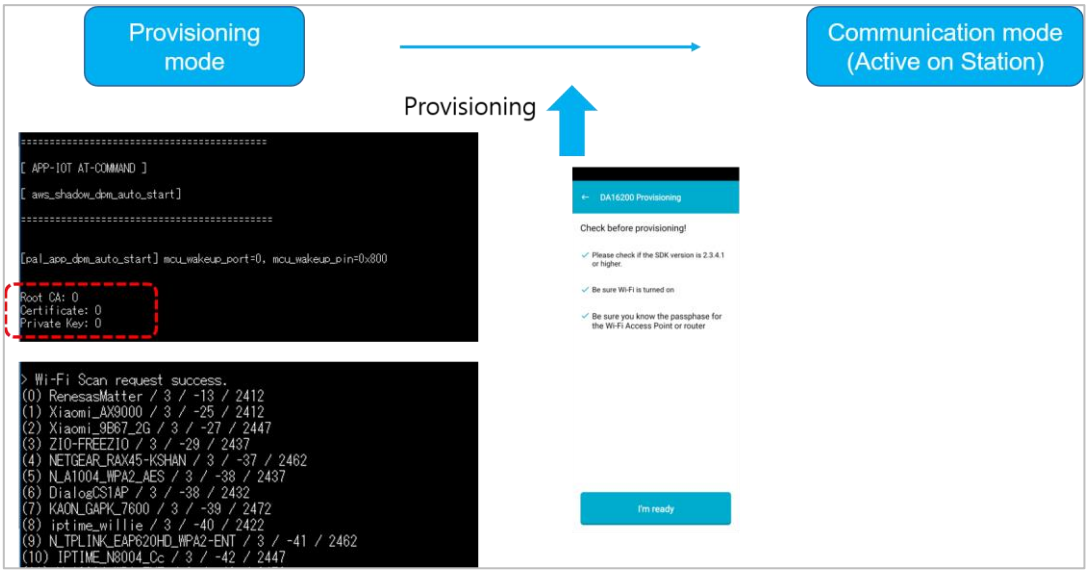

Figure 108. Provisioning mode

B.1.3 Communication Mode

The DA16200/DA16600 Communication Mode is used by the MCU to communicate (send and receive) topic values with an AWS server (Figure 109).

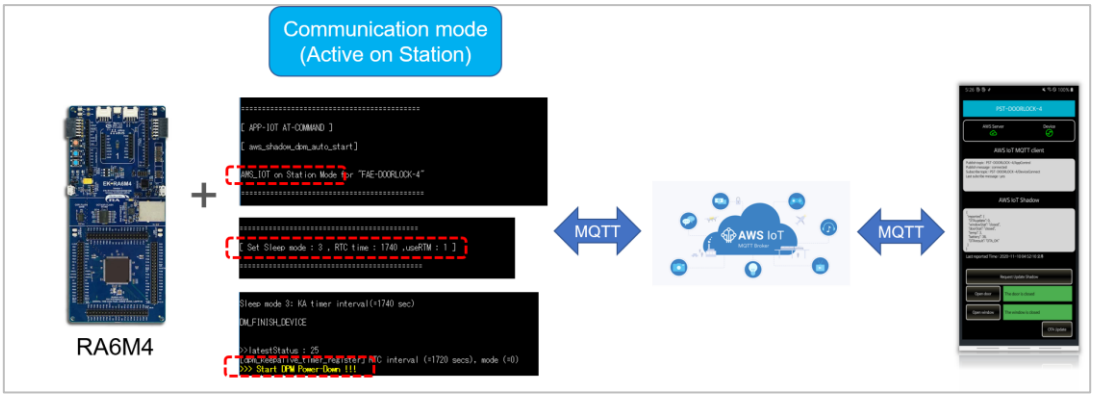

Figure 109. Communication mode

B.2 Configuring Topic to Publish, Subscribe, and Shadow

B.2.1 Configure Topics

- Topics are configured as shown in Table 6.
- The MCU and Mobile App should be configured based on the topics shown in Table 6.
- The MCU pushes the topics in Table 6 to the DA16200/DA16600 using AT command.

The DA16200 facilitates the communication between the MCU and phone as shown in Figure 110.

Table 6. Configuration of topics

Number	Name	Value Type	CMD Type	Value
0	app_door	1: String	2: Subscribe	"open"/"close"
1	mcu_door	1: String	0: Publish	"opened"/"closed"
2	battery	0: Integer	1: Shadow	Battery value (0~100)
3	temperature	2: Float	1: Shadow	Temperature value
4	doorStat	1: String	1: Shadow	"opened"/"closed"
5	windowStat	1: String	1: Shadow	"opened"/"closed"

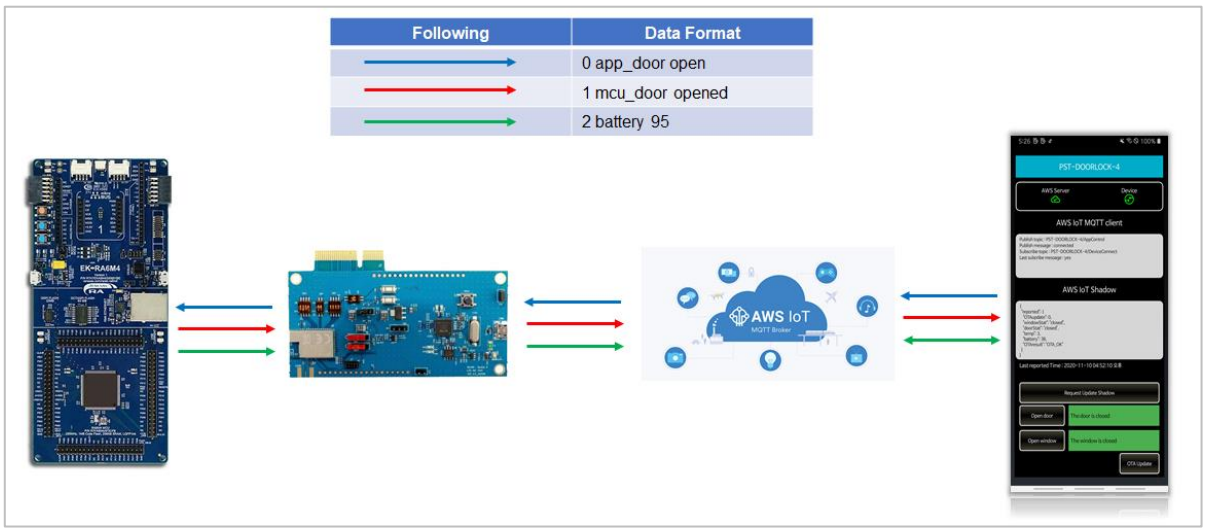

Figure 110. Communication between MCU and phone

B.3 AT Command List

B.3.1 Basic Set

Table 7. Basic set of MCU to DA16200/DA16600

Head	Main	Sub	Parameters
AT+AWS=	SET	APP_THINGNAME	Set the device thing name. Used to choose a device by its thing name during provisioning.
		AWS_BROKER	Set the broker address.
		APP_LPORT	Set the local port.
		APP_SUBTOPIC	Set subscriber topic name, and the default is "/AppControl".
		APP_PUBTOPIC	Set subs topic name, and the default is "/DeviceControl".
		SLEEP_MODE	Set sleep mode.

Head	Main	Sub	Parameters
			1 – not connected sleep. DA16200/DA16600 will wake up only by RTC_PWR_KEY. 2 – not connected sleep. DA16200/DA16600 will wake up by RTC. 3 – connected sleep. The connection is retained even during DPM.
		USE_DPM	Define the operation of sleep mode 3. 0 – no DPM. Used during debug. 1 – DPM mode.
		RTC_TIME	Set the wake-up time for Sleep mode 2.
		DPM_KEEP_ALIVE	Set the keep-alive time between the IoT device and the AP. Default value is 30*1000 microseconds.
		USE_WAKE_UP	Set the wake-up time for full-boot mode. Default value is set to 0 (0 = unused).
		TIM_WAKE_UP	Set the period to check a beacon frame from the AP. Default value is set to10.
		AWS_USE_FP	Not used command. 0 – Default value. 1 – Not in use.
EX) AT+AWS=SET APP_THINGNAME AssignedThingName AT+AWS=SET AWS_BROKER a1kzdt4nun8bnh-ats.iot.ap-northeast-2.amazonaws.com			

B.3.2 TLS Certificate

Table 8. TLS from MCU to DA16200/DA16600

Start Code	Sub Code	Type	End Code
\x1b	C0,	Root CA. Self-Signed, well known. Has root certificate public key. Signed by root certificate private key.	\x03
	C1,	Certificate key. Has own public key. Signed by root certificate private key. Use root certificate public key to prove authenticity.	
	C2,	Private key. Has own public key. Signed by certificate private key. Use certificate 1 public key to prove authenticity.	
EX) send "\x1b" over UART send "C0,-----BEGIN CERTIFICATE-----\n" "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo ….. over UART send "\x03"			

B.3.3 PIN MUX

Table 9. PIN MUX from MCU to DA16200/DA16600

Head	Main	Sub	Parameters
AT+AWS=	SET	NV_PIN_AMUX	AMUX_UART1d 4 /* UART1(RXD, TXD) */ AMUX_GPIO 9 /* GPIOA [1:0] */

Head	Main	Sub	Parameters
		NV_PIN_BMUX	BMUX_UART1d 4 /* UART1(RXD, TXD) */ BMUX_GPIO 8 /* GPIOA [3:2] */
		NV_PIN_CMUX	CMUX_UART1d 6 /* UART1(RXD, TXD) */ CMUX_GPIO 8 /* GPIOA [5:4] */
		NV_PIN_DMUX	DMUX_UART1d 4 /* UART1(RXD, TXD) */ DMUX_GPIO 8 /* GPIOA [7:6] */
		NV_PIN_EMUX	EMUX_GPIO 8 /* GPIOA [9:8] */
		NV_PIN_FMUX	FMUX_GPIO 6 /* GPIOA [11:10] */
		NV_PIN_UMUX	UMUX_GPIO 2 /* GPIOC [8:6] */
		APP_MCU_WKAEUP_PORT	GPIO_UNIT_A 0 GPIO_UNIT_C 2 /*Support only GPIO 6,7,8 */
		APP_MCU_WKAEUP_PIN	GPIO_PIN0 ~ GPIO_PIN11
		UART_CFG	[baud-rate]
Note: Default pin mux is BMUX Ex) use GPIOA2 and GPIOA3 for UART1, and GPIOA9 for MCU wakeup AT+AWS=SET NV_PIN_BMUX BMUX_UART1d AT+AWS=SET NV_PIN_EMUX EMUX_GPIO AT+AWS=SET APP_MCU_WKAEUP_PORT GPIO_UNIT_A AT+AWS=SET APP_MCU_WKAEUP_PIN GPIO_PIN9			

B.3.4 Configure Data as Topics

Table 10. Configuration data from MCU to DA16200/DA16600

Head	Main	Sub	Parameters
AT+AWS=	CFG	[number] [name] [value-type] [MQTT-type]	▪ number: Index to identify the saved topic. Increase by 1 when setting a new topic. Max value is 10 (total supported topics is 10). ▪ name: String specifying the topic name. ▪ value-type 0 – Integer type. 1 – String type. 2 – Float type. ▪ MQTT-type 0 – Publish: The prompt command is used to send a value from the MCU to the phone. For example, door state = true/false. 1 – Shadow: The value is sent to the device twin and will be updated on the phone the next time it is connected. 2 – Subscribe: The prompt command is used to send a value from the phone to the MCU. For example, door open command.
Ex) AT+AWS=CFG 0 doorStat 1 1 AT+AWS=CFG 1 battery 2 1 AT+AWS=CFG 2 door_open 0 2			

B.3.5 Command – MCU to DA16200/DA16600

Table 11. Command of MCU to DA16200/DA16600

Head	Main	Sub	Description
AT+AWS=	CMD	FACTORY_RESET	Reset the AWS IoT configuration to the factory default. All values stored in NVRAM are cleared. Use the "SET" and "CFG" commands to set the AWS IoT configuration.
		RESET_TO_AP	Switch to AP mode keeping the values set in NVRAM. The previous values in NVRAM will be kept.
		GET_STATUS	Get the current AWS IoT status. The MCU can read the current status from the DA16200/DA16600 at any time.
		RESTART	Reboot the device keeping the current mode and status.
		MCU_DATA	Used by the MCU to set a CFG parameter in the DA16200/DA16600.The value must be the same format as defined by the CFG setting. Parameters: [number] [name] [value]
Ex) AT+AZU=CMD FACTORY_RESET AT+AZU=CMD MCU_DATA 1 mcu_door opened			

B.3.6 Command – DA16200/DA16600 to MCU

Table 12. Command of DA16200/DA16600 to MCU

Head	Main	Parameters	Description
+AWSIOT	SERVER_DATA	[number] [name] [value]	Used by the DA16200/DA16600 to set a CFG parameter in the MCU. The value must be the same format as defined by the CFG setting.
+AWSIOT	CMD_TO_MCU	update	Used by the DA16200/DA16600 to request the status of devices such as sensors, batteries, and doors from the MCU. The DA16200/DA16600 maintains the values obtained from the MCU and forwards them when requested by an external phone app or by an MQTT ping-pong wake-up event.
Ex) +AWSIOT SERVER_DATA 0 door_control open +AWSIOT CMD_TO_MCU update			

B.3.7 DA16200/DA16600 Status – DA16200/DA16600 to MCU

Table 13. Status from DA16200/DA16600 to MCU

Status	Value	Parameters
IDLE	-1	Initial state of AWS-IoT application. Sent when a system error occurs. For example, network connection failure.
Done factory reset	0	Sent after completes factory reset by "CMD FACTORY_RESET".
Boot Ready	1	Sent when entering AWS-IoT application mode.
Need configuration	5	Sent if there is no setting. MCU should set and configure with the SET and CFG command.
Start AP mode	10	Sent when being started to AP mode. Need to process provisioning with Phone.
Network OK	15	Sent when it is OK to connect AP without problem.

Status	Value	Parameters
Network fail	16	Sent when it fails to connect AP with any problem. Normally, it happens during provisioning failure by the wrong SSID or PW. Need to go to AP mode by MCU send "RESET_TO_AP" command.
Start STA	20	Not defined yet.
Done STA	25	Sent when entering Sleep mode for DPM.
MCUOTA	30	Sent when MCU OTA starts processing.
EX) +AWSIOT STATUS 15		

Appendix C Troubleshooting

C.1 Operational Issue

When UI buttons are not visible or not showing up properly while using the mobile app, try to uninstall and install the app again. The first time running the mobile app after reinstalling it, make sure that the app can access the location of the device as described in Test Provisioning on Android/iPhone sections of Ref. [4].

Revision History

Revision	Date	Description
1.6	July 22, 2024	▪ Modified Note to provide customer with thing name for testing instead of providing the AWS login credentials. ▪ Added Section 4.1. ▪ Editorial changes.
1.5	Jan 26, 2024	Added Troubleshooting section.
1.4	Nov 30, 2023	Merged documents: ▪ UM-WI-016 DA16200 Door Lock Application Using AWS IoT. ▪ UM-WI-017 DA16200 AWS IoT Server Setup. ▪ UM-WI-038 DA16200 DA16600 Getting Started with AWS IoT Using AT Commands.
1.3	Aug 18, 2023	▪ Changed IDE to e2studio. ▪ Editorial update.
1.2	Dec 1, 2022	Edited as direct link of documents.
1.1	Nov 4, 2022	Modify hyperlink of the documents.
1.0	Oct 13, 2022	Initial version.

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Renesas Electronics' suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

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-us/.

Trademarks

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