Getting started with CoAP for AWS IoT Core

2022-01-22

Getting started with CoAP for AWS IoT Core

This document guides you through the steps to quickly get started with AWS IoT Core for CoAP beta. It covers the following topics:

Preparation (register CoAP device, deploy AWS Lambda function for uplink/downlink processing)

- Send uplink from CoAP client to AWS IoT Core (using CoAP PUT)
- Receive downlink by the CoAP client from AWS IoT Core on request of the CoAP Client (using CoAP GET)

If you experience any issues or unexpected behaviour, have questions or feedback, please contact us at awsiot-coap-beta-support@amazon.com.

Quick summary

The section contains condensed summary of essential steps required to register a device, send a telemetry via uplink and receive a configuration via uplink request/downlink response. This summary is recommended for the users already familiar with CoAP, AWS, AWS CLI and Linux shell. If you prefer to start with an introduction and a detailed explanation of each step, please click here.

1. **Deploy a sample solution for uplink and downlink processing in your AWS account**

   To deploy a sample solution you will use AWS Clouformation to provision AWS Lambda and AWS IoT Rule in your account. That will ensure that AWS cloud sends downlink to the device on each GET uplink from the device. If you prefer to review the CloudFormation template deploying using it, please consult Appendix 1.

   To deploy the stack, please first log into AWS Management Console of an AWS account that is allowlisted for the CoAP beta program. Then click on one of the links below to deploy the stack.

   - Deploy in us-east-1
   - Deploy in eu-west-1
   - Deploy in us-west-2
   - Deploy in ap-southeast-2
   - Deploy in ap-northeast-1

   If you operate your workloads in one of the regions not supported by the beta program, please let us know so we can support you in finding a solution.

2. **Make AWS CLI aware of the APIs of the IoT Core CoAP feature**

   In this guide we will use AWS CloudShell as an environment for running the required AWS CLI commands. Other Linux-based environments like Amazon Linux 2 on EC2, Cloud 9 or MacOS are also expected to work, but were not tested.

   Because the official version of AWS CLI installed AWS CloudShell is not supporting CoAP APIs yet, an additional configuration step is necessary. Please note that this step is a pre-requisite for a successfull participation in the beta programme.

   Please click here to open AWS Cloudshell. Please pay attention to use the same AWS region as in step 2.

   Now please run the following command to install the beta version of the AWS CLI in your CloudShell. You can copy&paste and run the following commands in AWS Cloudshell en block, or do so for each of the commands individually. If you have issues with copy-and-paste, please use Google Chrome as viewer.

   ```
   cd ~
   mkdir -p .aws
   cd .aws
   aws s3 cp  s3://iotwireless-coap-beta/sdk/coap-models.zip .
   unzip coap-models.zip
   cd ~
   ```

3. **Create an input file for the registration of a CoAP device**

   In this step you will create an input file for the CreateWirelessDevce API. Please refer to the documentation of CreateWirelessDevce API in the API reference for the specifiction of the parameters in the input file.

   To create a template for a configuration file, please run the command below:

```
cat > create-wireless-device.json <<EOF
{
 "Type": "CoAP",
  "Name": "MyCoapDevice",
  "Description": "This is my first CoAP device",
  "DestinationName": "coaphelloworld_CoAPDestination_python",
  "Cellular": {
    "Imei": "1000000000000001",
    "DtlsV1_2":
      {"PreSharedKeyIdentity":
       "gettingstartedcoapdeviceimei1000000000000001",
        "PreSharedKeySecret":
   "0x4141414141414141414141414141414141414141414141414141414141414141"
      }
  },
  "WiFi": {
    "MacAddress": "100000000001",
    "DtlsV1_2":
      { "PreSharedKeyIdentity":
        "gettingstartedcoapdevicemac100000000001",
        "PreSharedKeySecret":
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}
    }
}
EOF
```

Now please open the file in the editor:

```
sudo yum install nano -y
nano create-wireless-device.json
```

Define credentials for your CoAP device.

In the editor, please change values for the following parameters to globally unique ones:

- Cellular -> Imei (less or equal then 16 characters, allowed values: `^[0-9]+$`)
- Cellular -> DtlsV1_2 –> PreSharedKeyIdentity (allowed length: 32 - 256 characters, allowed values: `^[a-zA-Z0-9]+$`)
- WiFi -> MacAddress (allowed length: at least 12 characters)
- WiFi -> DtlsV1_2 –> PreSharedKeyIdentity (allowed length: 32 - 256 characters, , allowed values: `^[a-zA-Z0-9]+$`)

If you want, you may remove either Cellular or WiFi element, but not both.

Update preshared keys

For the security reasons, please also modify the following parameters:

- Cellular -> DtlsV1_2 –> PreSharedKey (32 characters)
- WiFi -> DtlsV1_2 –> PreSharedKey (32 characters)

Please note that you can choose from two string formats for the PreSharedKey value:

- If value starts with "0x", then it is interpreted as hexadecimal string, for example "0x41414141414141414141414141414141414141414141414141414141414141414141"
- If value does not start with "0x", then it is interpreted as a string of characters, for example "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

Define the Wireless Destination Name
If you want to use NodeJS uplink processor Lambda function instead of the default Python3 Lambda function, please replace "_python" with "_node" in the DestinationName value.

Note: If you have changed the name of the Cloudformation stack you deployed in the step 2, please ensure that the value of "DestinationName" field is identical to the Output "WirelessDestination-Name" of the Cloudformation stack.

After performing necessary modifications, please save the file.

4. **Register a new CoAP device** Please run:

```
aws iotwireless create-wireless-device \
    --cli-input-json file://create-wireless-device.json
```

Congratulations! Now you can ingest to the CoAP server with DTLS v1.2 PSK endpoint: coaps://coap.<region>.amazonaws.com (e.g., coaps://coap.us-east-1.amazonaws.com or coaps://coap.us-east-1.amazonaws.com).

5. **Optional: build an example CoAP client**

You can skip this step if you want to use your own CoAP client or device.

```
sudo amazon-linux-extras install epel -y
sudo yum install asciidoc mbedtls mbedtls-devel autoconf \
         automake libtool  zlib  zlib-devel -y
git clone https://github.com/obgm/libcoap.git
cd libcoap/
git submodule init
git submodule update
./autogen.sh
./configure  --disable-documentation --enable-examples \
             --with-mbedtls --enable-shared
make all
cd examples
# If everything worked as expected, you should see the file "coap-client
   "
ls coap-client
```

6. **Send uplink and request downlink**

   Before running the commands below, please ensure that:

   - DTLS v.1 PSK credentials are identical to the ones you defined in the step 5.
   - Please ensure that value of the shell variable AWS_REGION matches the region where you registred your CoAP device (e.g. us-east-1).

```
# For the sake of demonstration, we use "Cellular" identity type
./coap-client -m put "coaps://coap.$AWS_REGION.amazonaws.com/telemetry"\
                   -e test\
                   -u gettingstartedcoapdeviceimei1000000000000001\
                   -k AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

# For the sake of demonstration, we use "WiFi" identity type
./coap-client -m get "coaps://coap.$AWS_REGION.amazonaws.com/cfg?t=1" \
                 -u gettingstartedcoapdevicemac100000000001 \
                 -k AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

   Each time you send an uplink, the AWS IoT Rule will be invoked according to the configuration in the WirelessDestination.

   Please note:

   - The message `WARN CoAP Client with Mbed TLS does not support Identity Hint selection` is expected and does not indicate an error.
   - If the message `WARN ***. .. DTLS: Alert '80'` is displayed, this means that the DTLS handshake failed. Please verify if the AWS region matches the region of the CoAP device you registered, and please also verify the credentials.

Detailed guidelines

In the following sections you will find the detailed guidelines to get started with CoAP for AWS IoT Core.

What you are going to build

By following the steps in this guide you will deploy a solution with a following architecture:

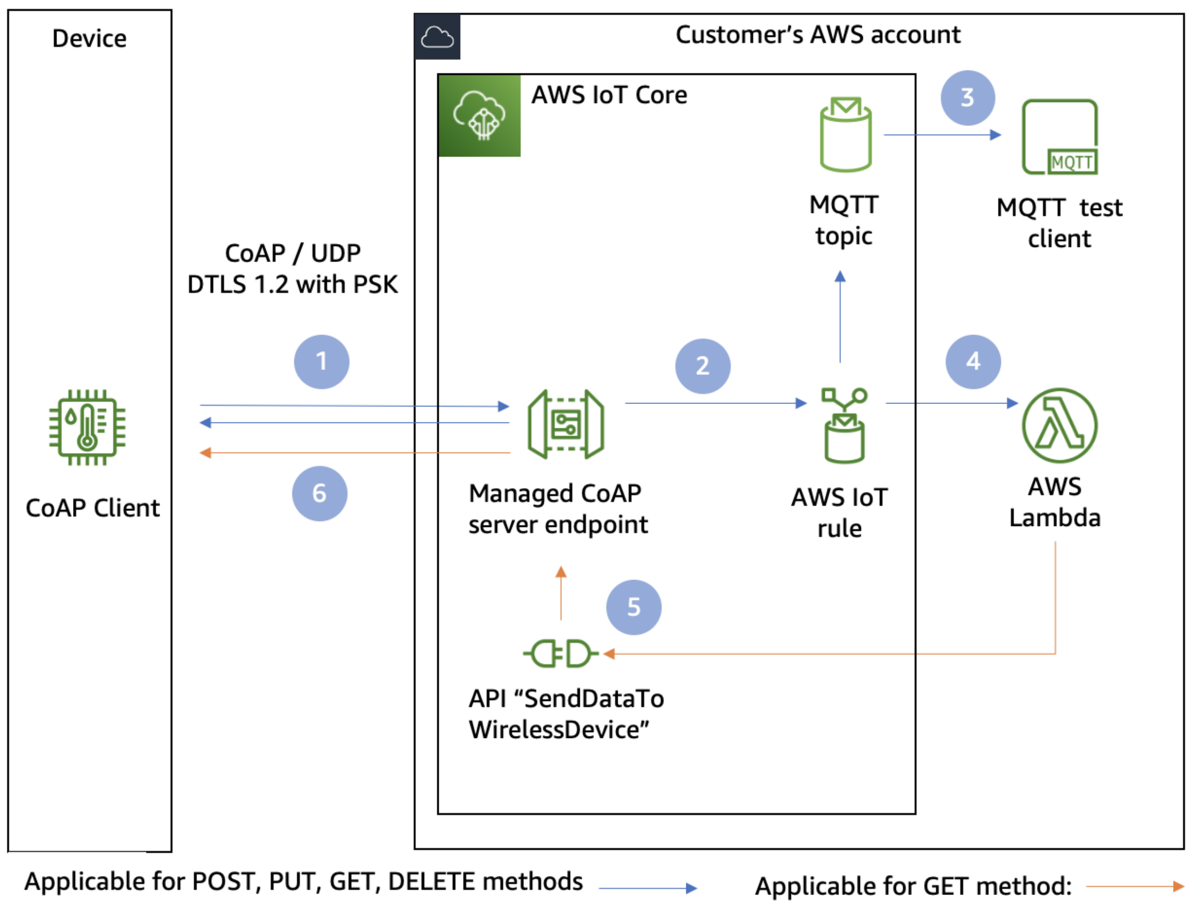


Figure 1: Solution architecture

Please consider the following steps:

1. IoT device sends a confirmable CoAP request message to a managed CoAP endpoint provided by AWS IoT Core. Immediately after the managed CoAP endpoint received CoAP request message, it

sends CoAP acknowledgement message back to the IoT device. *Please note: this acknowledgement message confirms that the message was received by the managed CoAP endpoint. It does not confirm the message was processed by the downstream services.*

2. Managed CoAP endpoint authenticates the device using DTLS 1.2 PSK. After a successfull authentication, Managed CoAP endpoint identifies a Wireless Destination assigned to the identity of the authenticated device. Because in our example the Wireless Destination is configured to process uplink with AWS IoT Rule, the specified AWS IoT Rule is invoked. JSON representation of the received CoAP message is passed to the AWS IoT Rule as an input.
3. For demonstration purposes, AWS IoT Rule publishes the message content as JSON to an MQTT topic. You can view the CoAP message content using AWS IoT MQTT Test Client.
4. AWS IoT Rule invokes AWS Lambda function to process the CoAP message input, however the processing logic depends on the CoAP method:
    - If the CoAP message has method "PUT" or "POST", it is viewed as telemetry ingestion. Processing stops here (in real-world scenarios, telemetry would be stored in some kind of database or object storage)
    - If the CoAP message has method "GET", it is viewed as a request for retrieval of configuration. AWS Lambda function will need to send a response to the device (in real-world scenarios, configuration would be read some from kind of database e.g. DymamoDB or IoT Device shadow).
5. To send a response back to the device, the AWS Lambda function invokes SendDataToWirelessDevice API to send CoAP response back to the device if the device is connected (i.e. UDP socket is open).
6. Managed CoAP server endpoint sends CoAP response message back to the IoT device.

Recommended environment

This guide was tested with AWS CloudShell. AWS CloudShell is a recommended environment for using this guide. Linux distributions like Amazon Linux 2 or Ubuntu are expected to work with possible minor modifications.

Remark: using AWS CloudShell or an EC2 instance introduces an unrealistic testing environment because of lack of NAT gateway between the cellular device and the CoAP endpoint. In real-world scenarios, a NAT gateway operated by the telco provider is between the cellular device and the CoAP endpoint. To ensure that you test in real-world environment, it is recommended to make an additional test with a realistic setup (i.e. either with a cellular device or Notebook behind a NAT gateway).

Step 1: Deploy the solution in your AWS account

To deploy a solution with the architecture described above, please follow the steps below.

1. Log into AWS Management Console of an AWS account that is allowlisted for the CoAP beta program.
2. Deploy Cloudformation stack

   The CloudFormation stack will add the following resources to your AWS account:

   - IoT Rule **coaphelloworld_CoAPUplinkProcessorIoTRulePython** and **coaphelloworld_CoAPUplinkProcess** and the related IAM Role
   - AWS Lambda function **coaphelloworld_CoAPUplinkProcessorLambdaFunctionPython** and **coaphelloworld_CoAPUplinkProcessorLambdaFunctionNode** and the related IAM Role
   - IoT WirelessDestination **coaphelloworld_CoAPWirelessDestination_python** pointing to the IoT Rule and the related IAM Role. **Please ensure to use this exact name when creating a new wireless device.**

   If you prefer to review the CloudFormation template deploying using it, please consult Appendix 1.

   Please click on the following links to start the deployment:

   - Deploy in us-east-1
   - Deploy in us-east-1
   - Deploy in us-west-2
   - Deploy in ap-southeast-2
   - Deploy in ap-northeast-1

3.Review source code (optional)

If you want to review the source code of the AWS Lambda function, please peform the following steps:

- In the “Output” section of the CloudFormation stack, lookup the name of the AWS Lambda function, e.g. ‘coaphelloworld-CoAPUplinkProcessorLambdaFunction-2IO5DeAxRSlt
- Open AWS Lambda console at https://console.aws.amazon.com/lambda/home?#/functions
- Click on the name of the AWS Lambda function. You will find source code in the “Code area”. If you want to build you own solution using this code a blueprint, please contact us at awsiot-coap-beta-support@amazon.com to receive a sample application.

Step 2: Make AWS CLI aware of the APIs of the IoT Core CoAP feature**

In this guide we will use AWS CloudShell as an environment for running the required AWS CLI commands. Other Linux-based environments like Amazon Linux 2 on EC2, Cloud 9 or MacOS are also expected to work, but were not tested.

Because the official version of AWS CLI installed AWS CloudShell is not supporting CoAP APIs yet, an additional configuration step is necessary. Please note that this step is a pre-requisite for a successfull participation in the beta programme.

Please click here to open AWS Cloudshell. Please pay attention to use the same AWS region as in step 2.

Now please run the following command to install the beta version of the AWS CLI in your CloudShell. You can copy&paste and run the following commands in AWS Cloudshell en block, or do so for each of the commands individually. If you have issues with copy-and-paste, please use Google Chrome as viewer.

```
cd ~
mkdir -p .aws
cd .aws
aws s3 cp  s3://iotwireless-coap-beta/sdk/coap-models.zip .
unzip coap-models.zip
cd ~
```

Step 3: Create an input file for the registration of your CoAP device

In this step you will create an input file for the CreateWirelessDevce API. Please refer to the documentation of CreateWirelessDevce API in the API reference for the specifiction of the parameters in the input file.

To create a template for a configuration file, please run the command below:

```
cat > create-wireless-device.json <<EOF
   {
      "Type": "CoAP",
      "Name": "MyCoapDevice",
      "Description": "This is my first CoAP device",
      "DestinationName": "coaphelloworld_CoAPDestination_python",
      "Cellular": {
        "Imei": "1000000000000001",
        "DtlsV1_2":
           { "PreSharedKeyIdentity": "
              gettingstartedcoapdeviceimei1000000000000001",
             "PreSharedKeySecret": "0
                x41414141414141414141414141414141414141414141414141414141414141414141
                "}
      },
      "WiFi": {
        "MacAddress": "100000000001",
        "DtlsV1_2":
          { "PreSharedKeyIdentity": "gettingstartedcoapdevicemac100000000001
             ",
          "PreSharedKeySecret": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}
        }
    }
EOF
```

Now please open the file in the editor:

```
sudo yum install nano -y
nano create-wireless-device.json
```

Define credentials for your CoAP device.
In the editor, please change values for the following parameters to globally unique ones: - Cellular -> Imei (less or equal then 16 characters, allowed values: `^[0-9]+$`) - Cellular -> DtlsV1_2 –> PreSharedKeyIdentity (allowed length: 32 - 256 characters, allowed values: `^[a-zA-Z0-9]+$`) - WiFi -> MacAddress (allowed length: at least 12 characters) - WiFi -> DtlsV1_2 –> PreSharedKeyIdentity (allowed length: 32 - 256 characters, , allowed values: `^[a-zA-Z0-9]+$`)

```
If you want, you may remove either Cellular or WiFi element, but not both.
```

Update preshared keys
For the security reasons, please also modify the following parameters: - Cellular -> DtlsV1_2 –> PreSharedKey (32 characters) - WiFi -> DtlsV1_2 –> PreSharedKey (32 characters)

```
Please note that you can choose from two string formats for the PreSharedKey
    value:
  - If value starts with "0x", then it is interpreted as hexadecimal string,
      e.g. "0
     x41414141414141414141414141414141414141414141414141414141414141414141"
  - If value does not start with "0x", then it is interpreted as a string of
      characters, e.g. "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```

Define the Wireless Destination Name

Please ensure that the value of "DestinationName" field is identical to the Output "WirelessDestination-Nam" of the Cloudformation stack you deployed in the step 2.

After performing necessary modifications, please save the file.

Step 4: Register the CoAP device

After updating the values, please run the following command:

```
aws iotwireless create-wireless-device \
     --cli-input-json file://create-wireless-device.json
```

Expected output:

```
{
    "Arn": "arn:aws:iotwireless:us-east-1:550363093759:WirelessDevice/
       c9e9090f-4377-4840-abf9-ef1c0eadd1c4",
    "Id": "c9e9090f-4377-4840-abf9-ef1c0eadd1c4"
}
```

Please note that the "Id" value is the unique identifier of the registered CoAP device in the current region. You can use this identifier to invoke APIs like SendDataToWirelessDevice, DeleteWirelessDevice, UpdateWirelessDevice, AssociateWirelessDeviceWithThing.

List CoAP devices (optional)

This is only for verification / demonstration purposes. You can skip this step.

```
aws iotwireless list-wireless-devices
```

Step 5 (optional): build an example CoAP client

This guide uses a example CoAP client for the open-source libcoap library with MbedTLS as a DTLS library. You can skip this step if you want to use your own CoAP client.

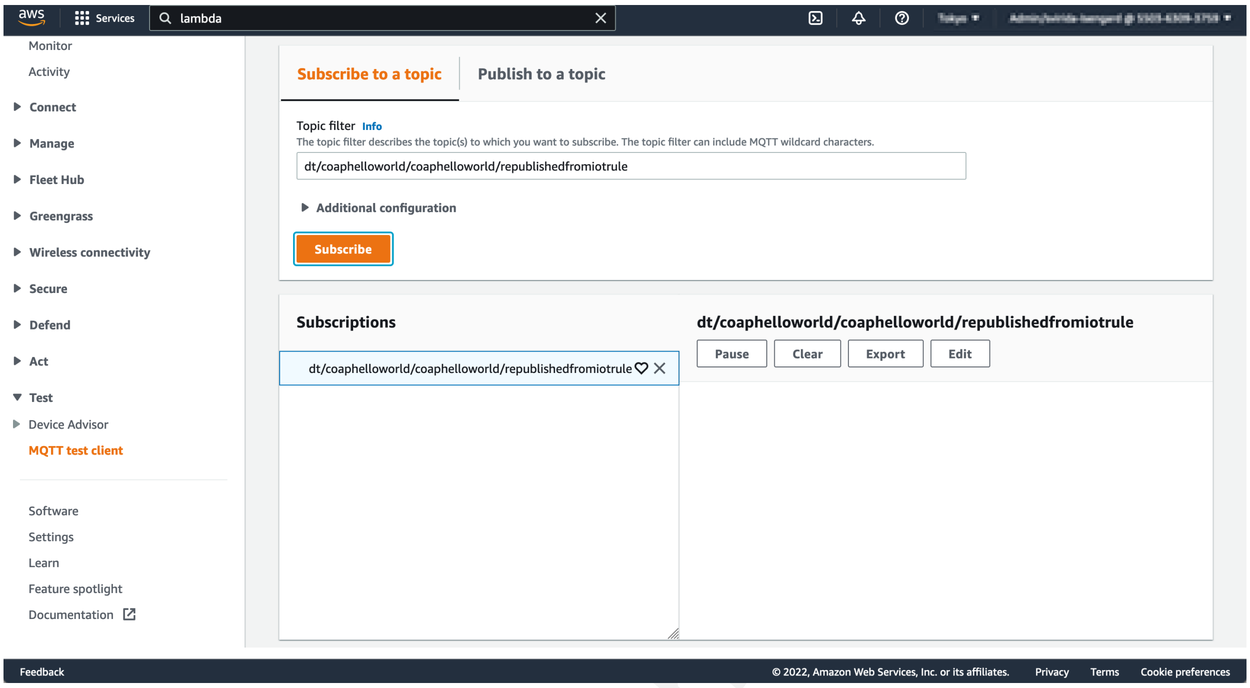
Please perform the following commands to build a CoAP client:

```
sudo amazon-linux-extras install epel -y
sudo yum install asciidoc mbedtls mbedtls-devel autoconf \
          automake libtool  zlib  zlib-devel -y
# This guide assumes that you will be storing the libcoap source code in the
    user home directory.
cd .
git clone https://github.com/obgm/libcoap.git
cd libcoap/
git submodule init
git submodule update
./autogen.sh
./configure  --disable-documentation --enable-examples \
                 --with-mbedtls --enable-shared
make all
```

Please note that these commands were succesfully tested on Amazon Linux 2. Other Linux distributions may require modifications

Step 6: open MQTT Client

To observe the incoming CoAP messages republished to the MQTT Broker by AWS IoT Rule, please open MQTT Test client by clicking here. Please subscribe to the MQTT topic `dt/coaphelloworld/coapdemo/coapdebug`.



Topic filter Info

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

dt/coaphelloworld/coaphelloworld/republishedfromiotrule

Subscriptions

dt/coaphelloworld/coaphelloworld/republishedfromiotrule

dt/coaphelloworld/coaphelloworld/republishedfromiotrule

Figure 2: MQTT Test Client

Step 7: send uplink from CoAP Device to AWS IoT Core with a confirmable PUT request

The image below represents a "happy path" (i.e. successfull execution) of CoAP CON PUT command from the device to the cloud:

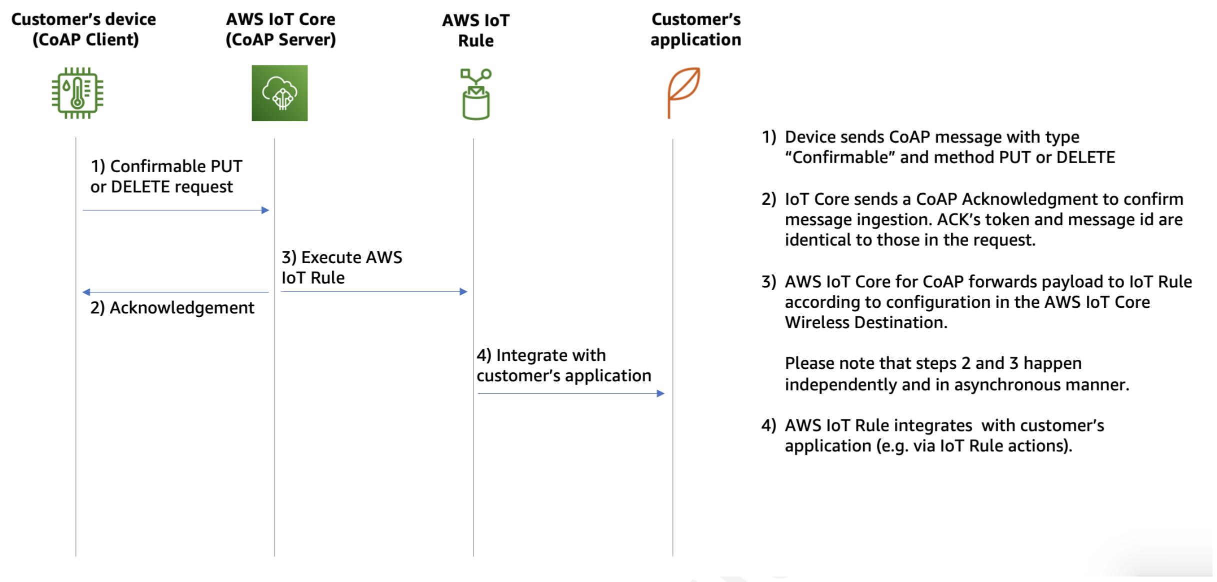

Customer's device (CoAP Client)
AWS IoT Core (CoAP Server)
AWS IoT Rule
Customer's application
1) Confirmable PUT or DELETE request
2) Acknowledgement
3) Execute AWS IoT Rule
4) Integrate with customer's application
1) Device sends CoAP message with type "Confirmable" and method PUT or DELETE
2) IoT Core sends a CoAP Acknowledgment to confirm message ingestion. ACK's token and message id are identical to those in the request.
3) AWS IoT Core for CoAP forwards payload to IoT Rule according to configuration in the AWS IoT Core Wireless Destination.
Please note that steps 2 and 3 happen independently and in asynchronous manner.
4) AWS IoT Rule integrates with customer's application (e.g. via IoT Rule actions).

Figure 3: Uplink from CoAP Device to AWS IoT Core with a confirmable PUT request

Update configuration

- Replace the DTLS PSK credentials with the ones you entered in the file `create-wireless-device.json`
- Adjust the name of the region (e.g. coaps://coap.eu-west-1.amazonaws.com instead of coaps://coap.us-east-1.amazonaws.com) if necessary.

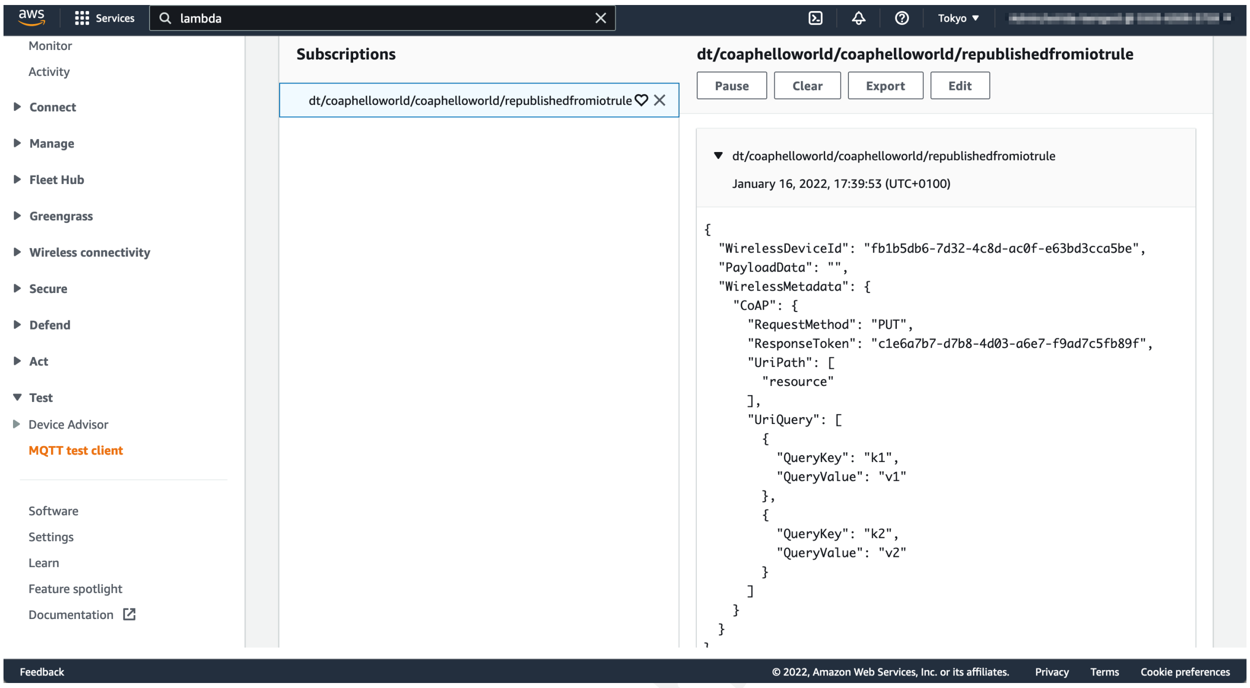
Send the request

Pleas run the following commands:

```
~/libcoap/examples/coap-client -m put "coaps://coap.$AWS_REGION.amazonaws.
    com/telemetry"\
                             -e test\
                             -u gettingstartedcoapdeviceimei1000000000000001\
                             -k AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

View payload from CoAP device using AWS IoT MQTT Test client

Open MQTT Test client and see the data arriving to the cloud on the topic `dt/coaphelloworld/coaphelloworld/republishedfromiotrule`:



dt/coaphelloworld/coaphelloworld/republishedfromiotrule

dt/coaphelloworld/coaphelloworld/republishedfromiotrule

January 16, 2022, 17:39:53 (UTC+0100)

```
{
  "WirelessDeviceId": "fb1b5db6-7d32-4c8d-ac0f-e63bd3cca5be",
  "PayloadData": "",
  "WirelessMetadata": {
    "CoAP": {
      "RequestMethod": "PUT",
      "ResponseToken": "c1e6a7b7-d7b8-4d03-a6e7-f9ad7c5fb89f",
      "UriPath": [
        "resource"
      ],
      "UriQuery": [
        {
          "QueryKey": "k1",
          "QueryValue": "v1"
        },
        {
          "QueryKey": "k2",
          "QueryValue": "v2"
        }
      ]
    }
  }
```

Figure 4: MQTT Test Client

Step 8: send downlink from AWS IoT Core to CoAP device initiated by a non-confirmable GET request

The image below represents a "happy path" (i.e. successfull execution) of CoAP NON GET command from the device to the cloud:

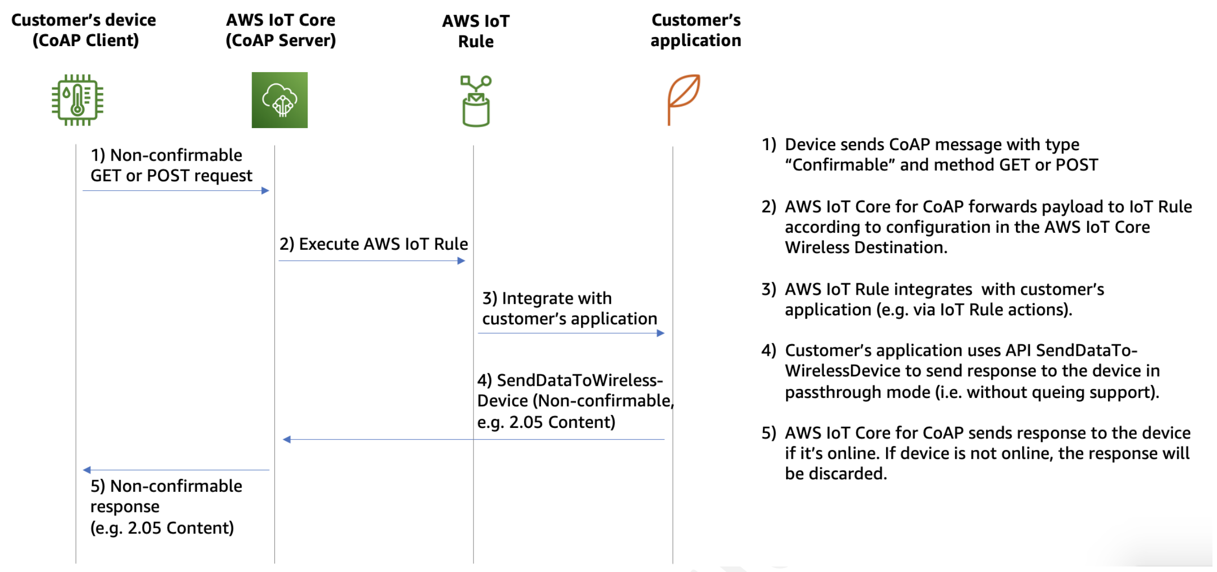

Customer's device (CoAP Client)
AWS IoT Core (CoAP Server)
AWS IoT Rule
Customer's application
1) Non-confirmable GET or POST request
2) Execute AWS IoT Rule
3) Integrate with customer's application
4) SendDataToWireless-Device (Non-confirmable, e.g. 2.05 Content)
5) Non-confirmable response (e.g. 2.05 Content)
1) Device sends CoAP message with type "Confirmable" and method GET or POST
2) AWS IoT Core for CoAP forwards payload to IoT Rule according to configuration in the AWS IoT Core Wireless Destination.
3) AWS IoT Rule integrates with customer's application (e.g. via IoT Rule actions).
4) Customer's application uses API SendDataTo-WirelessDevice to send response to the device in passthrough mode (i.e. without queing support).
5) AWS IoT Core for CoAP sends response to the device if it's online. If device is not online, the response will be discarded.

Figure 5: Downlink from AWS IoT Core to CoAP device initiated by a non-confirmable GET request

Update configuration

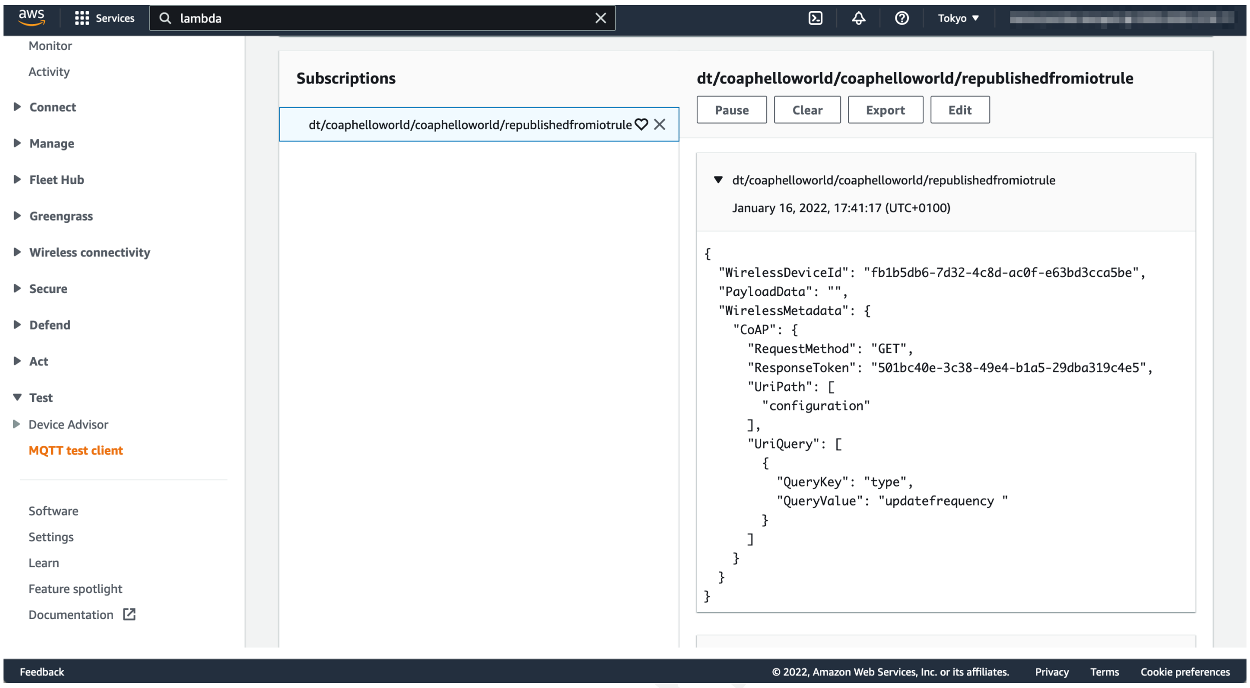
- Replace the DTLS PSK credentials with the ones you entered in the file `create-wireless-device.json`
- Adjust the name of the region (e.g. coaps://coap.eu-west-1.amazonaws.com instead of coaps://coap.us-east-1.amazonaws.com) if necessary.

Send the request

```
~/libcoap/examples/coap-client  -m get "coaps://coap.$AWS_REGION.amazonaws.
    com/cfg?t=1" \
                          -u gettingstartedcoapdevicemac100000000001 \
                          -k AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

View payload from CoAP device using AWS IoT MQTT Test client

Open MQTT Test client and see the data arriving to the cloud on the topic `dt/coaphelloworld/coaphelloworld/republishedfromiotrule`.



dt/coaphelloworld/coaphelloworld/republishedfromiotrule

January 16, 2022, 17:41:17 (UTC+0100)

```
{
  "WirelessDeviceId": "fb1b5db6-7d32-4c8d-ac0f-e63bd3cca5be",
  "PayloadData": "",
  "WirelessMetadata": {
    "CoAP": {
      "RequestMethod": "GET",
      "ResponseToken": "501bc40e-3c38-49e4-b1a5-29dba319c4e5",
      "UriPath": [
        "configuration"
      ],
      "UriQuery": [
        {
          "QueryKey": "type",
          "QueryValue": "updatefrequency "
        }
      ]
    }
  }
}
```

Figure 6: MQTT Test Client

Conclusion and next steps

In this guide we have demonstrated how to register CoAP device in AWS IoT Core, send an uplink from the device, and request a downlink to this device. If you have any feedback, or want to use a sample application from this guide as a blueprint for your own development, please let us know at awsiot-coap-beta-support@amazon.com.

Appendix 1: Sample payloads for the AWS IoT Rule

As a reference for an implementation of uplink processing, please see the following sample payloads. For example, if you implement AWS IoT Rule with AWS Lambda Action, you AWS Lambda function will receive these payloads as an input event.

PUT request

```
{
  "WirelessDeviceId": "ed8d199f-9ab1-4a5d-8df9-198c8b39952c",
  "PayloadData": "dGVzdA==",
  "WirelessMetadata": {
    "CoAP": {
      "RequestMethod": "PUT",
      "ResponseToken": "0c22709f-5c65-42f5-af16-1f15d20b7bf1",
      "UriPath": [
        "telemetry"
      ]
    }
  }
}
```

GET request

```
{
  "WirelessDeviceId": "ed8d199f-9ab1-4a5d-8df9-198c8b39952c",
  "PayloadData": "dGVzdA==",
  "WirelessMetadata": {
    "CoAP": {
      "RequestMethod": "GET",
      "ResponseToken": "de27e797-c687-4176-ad4d-19a6115401aa",
      "UriPath": [
        "command"
      ],
      "UriQuery": [
        {
          "QueryKey": "type",
          "QueryValue": "firmware"
        }
      ]
    }
  }
}
```

POST request

```
{
  "WirelessDeviceId": "ed8d199f-9ab1-4a5d-8df9-198c8b39952c",
  "PayloadData": "dGVzdA==",
  "WirelessMetadata": {
    "CoAP": {
      "RequestMethod": "POST",
      "ResponseToken": "7dc5af83-59f0-48ef-a7ad-9f53bd90d4c0",
      "UriPath": [
        "telmetrywithdownlink"
      ]
    }
  }
}
```

Appendix 2: CloudFormation template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: CoAP payload processor

Globals:
  Function:
    Timeout: 10

Parameters:
  ParameterTopicBaseName:
    Type: String
    Description: Base name of MQTT topic without trailing '/'
    Default: coapdemo

Resources:
  CoAPUplinkProcessorIoTRule:
    Type: AWS::IoT::TopicRule
    Properties:
      Tags:
        - Key: PROJECT
          Value: CoAPHelloWorld
      RuleName: !Sub ${AWS::StackName}_CoAPUplinkProcessorIoTRule
      TopicRulePayload:
        Sql:
          !Sub SELECT *
        AwsIotSqlVersion: '2016-03-23'
        Description: 'CoAP payload processor IoT rule'
        Actions:
        -
          Lambda:
            FunctionArn: !GetAtt CoAPUplinkProcessorLambdaFunction.Arn
        -
          Republish:
            Topic: !Sub "dt/${AWS::StackName}/${ParameterTopicBaseName}/
                republishedfromiotrule"
            RoleArn: !GetAtt CoAPUplinkProcessorIoTRuleRole.Arn

        ErrorAction:
          Republish:
            Topic: !Sub "dt/${AWS::StackName}/${ParameterTopicBaseName}/
                coaperror"
            RoleArn: !GetAtt CoAPUplinkProcessorIoTRuleRole.Arn
  CoAPUplinkProcessorIoTRuleRole:
    Type: AWS::IAM::Role
    Properties:
      Tags:
        - Key: PROJECT
          Value: CoAPHelloWorld

      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        -
          Effect: Allow
          Principal:
            Service:
            - iot.amazonaws.com
          Action:
            - sts:AssumeRole
      Path: /
      Policies:
      -
        PolicyName: CoAPUplinkProcessorIoTRulePolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
          -
            Effect: Allow
            Action:
            - iot:Publish
            Resource: "*"
  LambdaInvocationPermission:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref CoAPUplinkProcessorLambdaFunction
      Action: lambda:InvokeFunction
      Principal: iot.amazonaws.com
      SourceArn: !GetAtt CoAPUplinkProcessorIoTRule.Arn


  CoAPUplinkProcessorLambdaFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      Tags:
        - Key: PROJECT
          Value: CoAPHelloWorld

      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        -
          Effect: Allow
          Principal:
            Service:
            - lambda.amazonaws.com
          Action:
            - sts:AssumeRole
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/
          CloudWatchLambdaInsightsExecutionRolePolicy'
      Policies:
      -
        PolicyName: CoAPUplinkProcessoRolePolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
          -
            Effect: Allow
            Action:
            - iot:Publish
            Resource: "*"

          -
            Effect: Allow
            Action:
            - iotwireless:SendDataToWirelessDevice
            Resource: "*"

          -
            Effect: Allow
            Action:
            - logs:CreateLogGroup
            Resource: !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:*"

          -
            Effect: Allow
            Action:
            - logs:CreateLogStream
            - logs:PutLogEvents
            Resource: !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:
               log-group:/aws/lambda/*"
```

```
CoAPUplinkProcessorLambdaFunction:
  Name: !Sub ${AWS::StackName}_CoAPUplinkProcessorLambdaFunction
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: src
    Handler: app.lambda_handler
    Runtime: python3.7
    Timeout: 10
    Role: !GetAtt CoAPUplinkProcessorLambdaFunctionRole.Arn
    Layers:
      - !Sub "arn:aws:lambda:${AWS::Region}:580247275435:layer:
         LambdaInsightsExtension:14"
    Environment:
      Variables:
        AWS_DATA_PATH: ./models

CoAPWirelessDestination:
  Type: AWS::IoTWireless::Destination
  Properties:
    Description: Process the CoAP uplink payloads
    Expression: !Ref CoAPUplinkProcessorIoTRule
    ExpressionType: RuleName
    Name: !Sub ${AWS::StackName}_CoAPDestination
    RoleArn: !GetAtt AWSIotWirelessDestinationRole.Arn
    Tags:
      - Key: PROJECT
        Value: CoAPHelloWorld
AWSIotWirelessDestinationRole:
  Type: AWS::IAM::Role
  Properties:
    Tags:
      - Key: PROJECT
        Value: CoAPHelloWorld

    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
      -
        Effect: Allow
        Principal:
          Service:
          -  iotwireless.amazonaws.com
        Action:
          - sts:AssumeRole
    Path: /
    Policies:
    -
      PolicyName: CoAPUplinkProcessorIoTRulePolicy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        -
          Effect: Allow
          Action:
          - iot:Publish
          Resource: !Sub "arn:aws:iot:${AWS::Region}:${AWS::AccountId}:
              topic/$aws/rules/*"

        -
          Effect: Allow
          Action:
          - iot:DescribeEndpoint
          Resource: "*"
```

```
Outputs:
  WirelessDestinationName:
    Value: !Ref CoAPWirelessDestination
    Description: Please ensure to reference this name when calling
        CreateWirelessDevice API

  RuleName:
    Value: !Sub ${AWS::StackName}_CoAPUplinkProcessorIoTRule
    Description: This is the name of the IoT Rule which will be invoked on
        each uplink

  LambdaFunctionName:
    Value: !Ref CoAPUplinkProcessorLambdaFunction
    Description: This is the name of the Lambda function which will be
        invoked on each uplink by the IoT Rule
```