



Squid.link gateway AWS IoT Core: Getting started

Revised 25.01.2022

Content

1	Cautionary notes	4
2	Getting started with Squid.link gateways.....	5
2.1	Installation manual.....	5
3	Squid Smart APP API intro	6
3.1	API Documentation.....	7
3.2	Adding devices to the gateway	8
4	AWS IoT Core handler	9
4.1	AWS IoT Core	9
4.2	Configuring AWS IoT	11
4.2.1	Set up your AWS Account	11
4.2.2	Creating a thing and certificates	12
4.2.3	Creating a policy.....	14
4.2.4	Ca certificate	16
4.2.5	AWS IoT server URL.....	16
4.3	Configuring Squidsmart api AWS IoT Handler.....	17
4.4	Whitelisting devices.....	18
5	Interoperability with Device Shadows over MQTT	19
6	Updates.....	22
7	Contact Information	22
8	References	22

Copyright © Develco Products A/S

All rights reserved.

Develco Products assumes no responsibility for any errors, which may appear in this manual. Furthermore, Develco Products reserves the right to alter the hardware, software, and/or specifications detailed herein at any time without notice, and Develco Products does not make any commitment to update the information contained herein.

All the trademarks listed herein are owned by their respective owners.

1 Cautionary notes

Develco Products A/S reserves the right to make changes to any product to improve reliability without further notice. Develco Products A/S does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under patent rights or the rights of third parties.

2 Getting started with Squid.link gateways

The Squid.link gateways are an open Linux platform including multiple wireless networks for communication with IoT devices like smart meters, smart sensors, smart plugs, smart thermostats etc. The gateways are modular and can handle many different wireless protocols at the same time.

The Squid.link gateways have options for ZigBee, Z-Wave, Wireless M-Bus, Bluetooth Classic, Bluetooth Low Energy (BLE), and WLAN HAN networks. Communication with servers and e.g. smartphones can be established via WLAN, Ethernet (to local modem) or cellular networks. The price is extremely competitive since you will only pay for your selected communication modules.

The gateway includes processor power to implement even very complex local intelligence. The memory options leave room for data storage and logging. You are no longer dependent on one vendor but can combine your Home Area network exactly the way you prefer.

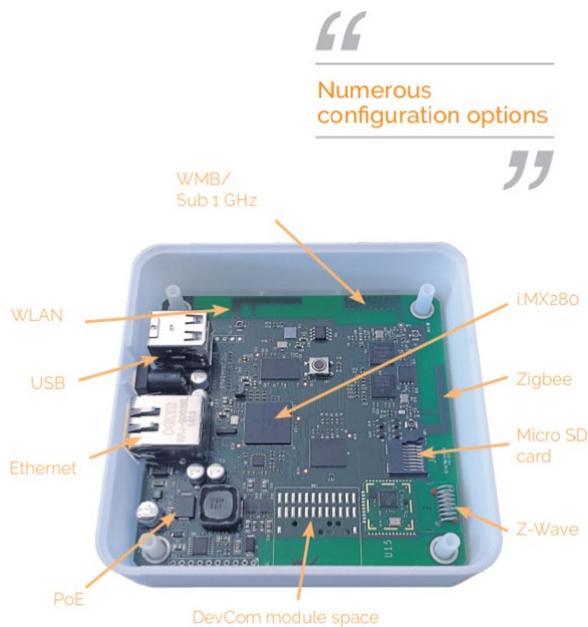
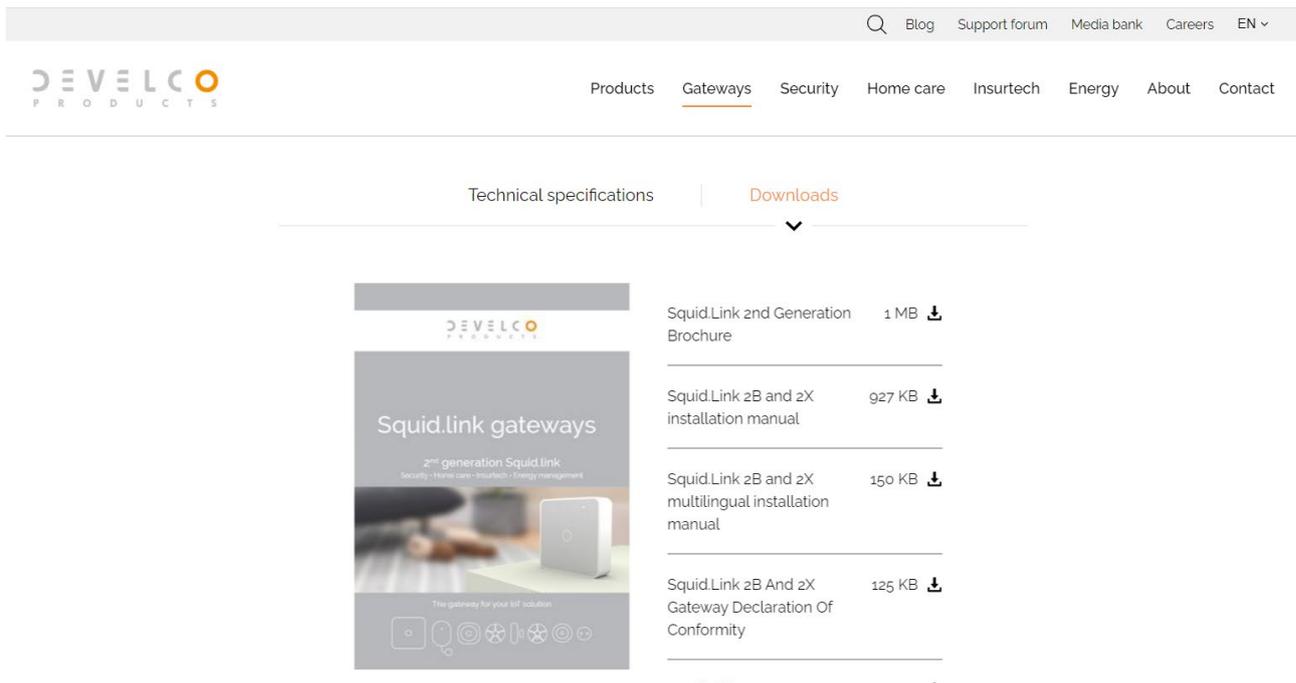


Figure 1 Squid.link gateway

The Squid.link gateway is configured when put into production. You can also design the appearance of the gateways the way you want. There are numerous design and color options that can be tailored according to your needs. Further description of the Squid.link gateways can be found using the link - [Squid.link Gateway](#)

2.1 Installation manual

The installation manual for the gateway can be found on our website. Please choose your selected gateway and go to the download section - [Squid.link gateway installation manual](#)



DEVELCO PRODUCTS

Products Gateways Security Home care Insurtech Energy About Contact

Technical specifications | Downloads

Squid.link gateways
2nd generation Squid.link
Security · Home care · Insurance · Energy management
The gateway for your IoT solution

Squid.Link 2nd Generation Brochure	1 MB	↓
Squid.Link zB and zX installation manual	927 KB	↓
Squid.Link zB and zX multilingual installation manual	150 KB	↓
Squid.Link zB And zX Gateway Declaration Of Conformity	125 KB	↓

Figure 2 Squid.link gateways installation manual

3 Squid Smart APP API intro

The Squid Smart App is a middleware application that allows developers to communicate with smart home appliances in a protocol-agnostic fashion which is achieved by displaying all devices as a resource in the REST API. The application should run on the Develco Products Squid.link gateways. The application is prepared for communicating with devices using different wireless technologies. Currently, the application supports ZigBee, Wireless M-Bus, and Bluetooth technologies.

The Squid Smart App offers the REST API, and each device linked to the gateway is modelled as an API resource and it relies on easy-to-read templates to identify which resources are accessible. The datapoints that belong logically together are grouped in a logical device. The Squid Smart App offers various ways of connection to the cloud/server side system by using MQTT, raw socket, web socket and AWS IoT handlers. The choice of connection is defined based on customer and use case requirement. Chapter 4 elaborates more on AWS IoT core handler.

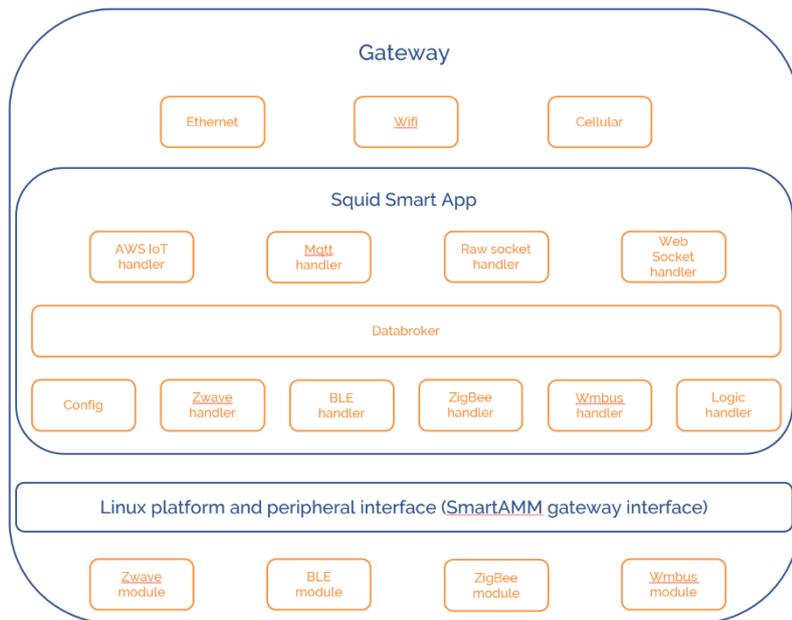


Figure 3 Gateway Architecture with Squid Smart APP as middleware

To illustrate the concept there is an example below showing the URL of the onoff datapoint of a smartplug. In this case it is device 1 and the name of the logical device is "smartplug". {hostname} should be replaced by the hostname or IP of the gateway (E.g.,hostname can be gw-244A, where 244A is the last four digits of gateway serial number at the back of the gateway). In the default configuration the API is available on port 80. If the application has been configured to use another port, remember to include the port number in the URL.

[http://\[hostname\]/ssapi/zb/dev/1/ldev/smartplug/data/onoff](http://[hostname]/ssapi/zb/dev/1/ldev/smartplug/data/onoff)

The logical device named Smart Plug also contains datapoints for reading current, voltage and consumed energy. The device also has a logical device named diagnostic which contains datapoints for reading network link strength, etc.

The application has been designed so that it's possible to discover which resources are available. To get a list of devices send a GET request to:

[http://\[hostname\]/ssapi/zb/dev/](http://[hostname]/ssapi/zb/dev/)

To get the list of logical endpoints on a specific device send a GET request to:

[http://\[hostname\]/ssapi/zb/dev/{device no}/ldev/](http://[hostname]/ssapi/zb/dev/{device no}/ldev/)

3.1 API Documentation

The API doc is also available directly on the gateway or internet. It can be accessed at the following URLs:

[http://\[hostname\]/api-docs/interactive/index.html#/](http://[hostname]/api-docs/interactive/index.html#/)

<http://api.squid.link/>

The interactive documentation will allow you to try out the API calls directly from the browser.

Figure 4 Squid Smart API

The API can be accessed over HTTP or by connecting to a WebSocket or a raw socket. When using HTTP the message pattern is request/response. The WebSocket and raw socket wrap the REST API and adds support for push messages. Messages are pushed when a resource is: added, removed or updated.

The raw socket can be access on port 10000 and only from localhost. The WebSocket and HTTP interface is available on port 80. The port number can be changed in the configuration file.

3.2 Adding devices to the gateway

The ZigBee handler takes care of adding and discovering devices. When a device has been discovered and a matching template has been found the handler will add all the appropriate resources to the API.

A ZigBee device can be added in three ways:

1. Adding with EUI
 - a. This is done by sending a POST request with the eui of the ZigBee device to `/zb/dev`. It is recommended to also include the installcode for improved security.

2. Enable scan mode
 - a. Scan mode is enabled by sending a PUT request to /zb. By setting the autoAdd option to false any found device will be added to the list of prospects. The devices which should be added to the API should be added afterwards using method 1. Alternatively, the devices can be added automatically by setting autoAdd to true.
 - b. The list of prospects can be read by sending a GET request to /zb. If connected to a WebSocket or a raw socket the prospects list will be pushed every 5 seconds while the scan is active.
3. Adding with barcode
 - a. Devices can be added similar to method 1. Only not adding the eui and install code but instead the string read from the device barcode (currently supports DP and EnOcean barcode formats)

POST /zb/dev Add a new ZigBee device

Parameters Try it out

No parameters

Request body required application/json

ZigBee device that should be added to the gateway.

Example Value | Model

```
{
  "barcode": "|0015BC002F000398|02B45383C263F60AD358B13DD9EC4A807619|",
  "eui": "0015BC002F000398",
  "installcode": "02B45383C263F60AD358B13DD9EC4A807619",
  "type": "Smart plug"
}
```

Figure 5 Adding a zigbee device using API

4 AWS IoT Core handler

4.1 AWS IoT Core

AWS IoT enables Internet-connected devices to connect to the AWS Cloud and lets applications in the cloud interact with Internet-connected devices. Common IoT applications either collect and process telemetry from devices or enable users to control a device remotely.

The block diagram below gives an overview of the AWS IoT system. The goal is to integrate a Squid.link gateway with AWS IoT Core to facilitate access to all connected devices underneath the gateway.

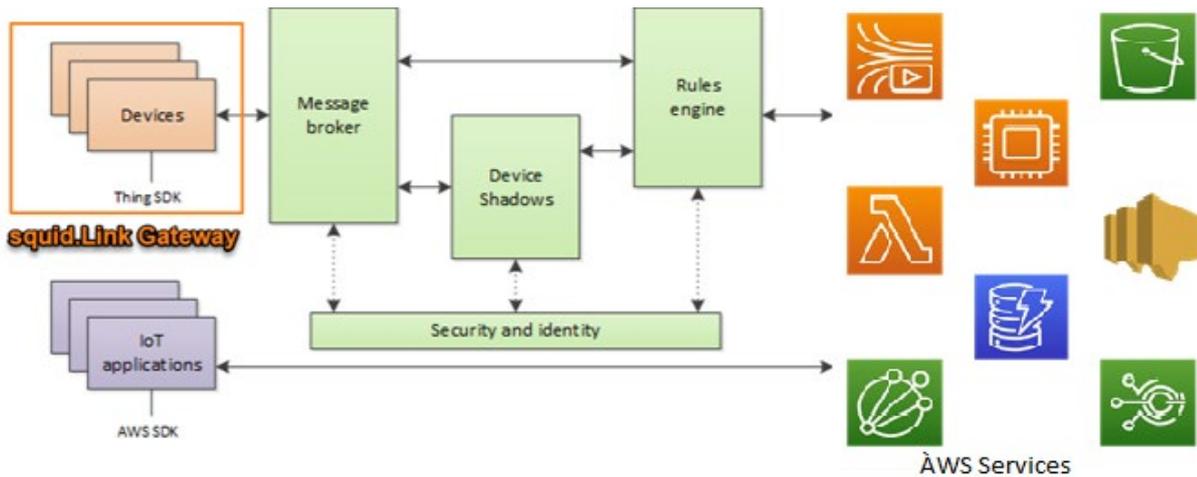


Figure 6 AWS IoT Core system [1]

The state of each device connected to AWS IoT is stored in a device shadow. The Device Shadow service manages device shadows by responding to requests to retrieve or update device state data. The Device Shadow service makes it possible for devices to communicate with applications and for applications to communicate with devices.

Communication between a device and AWS IoT is protected through the use of X.509 certificates. AWS IoT can generate a certificate or we can use our own. In either case, the certificate must be registered and activated with AWS IoT, and then copied onto your device. When a device communicates with AWS IoT, it presents the certificate to AWS IoT as a credential. It is recommended that all devices that connect to AWS IoT have an entry in the registry. The registry stores information about a device and the certificates that are used by the device to secure communication with AWS IoT.

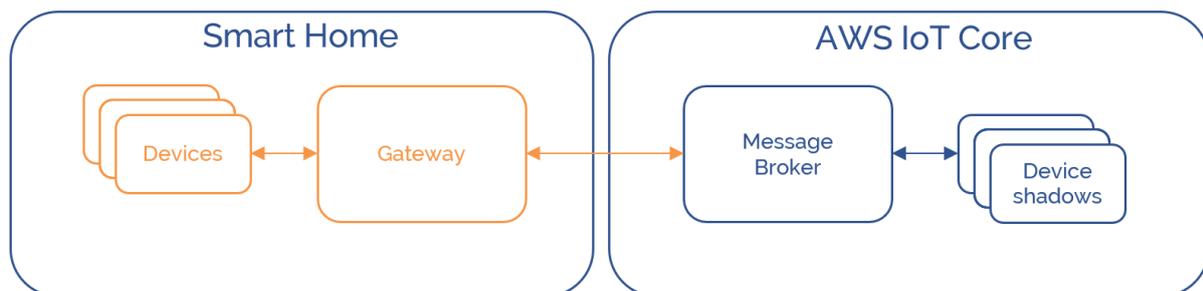


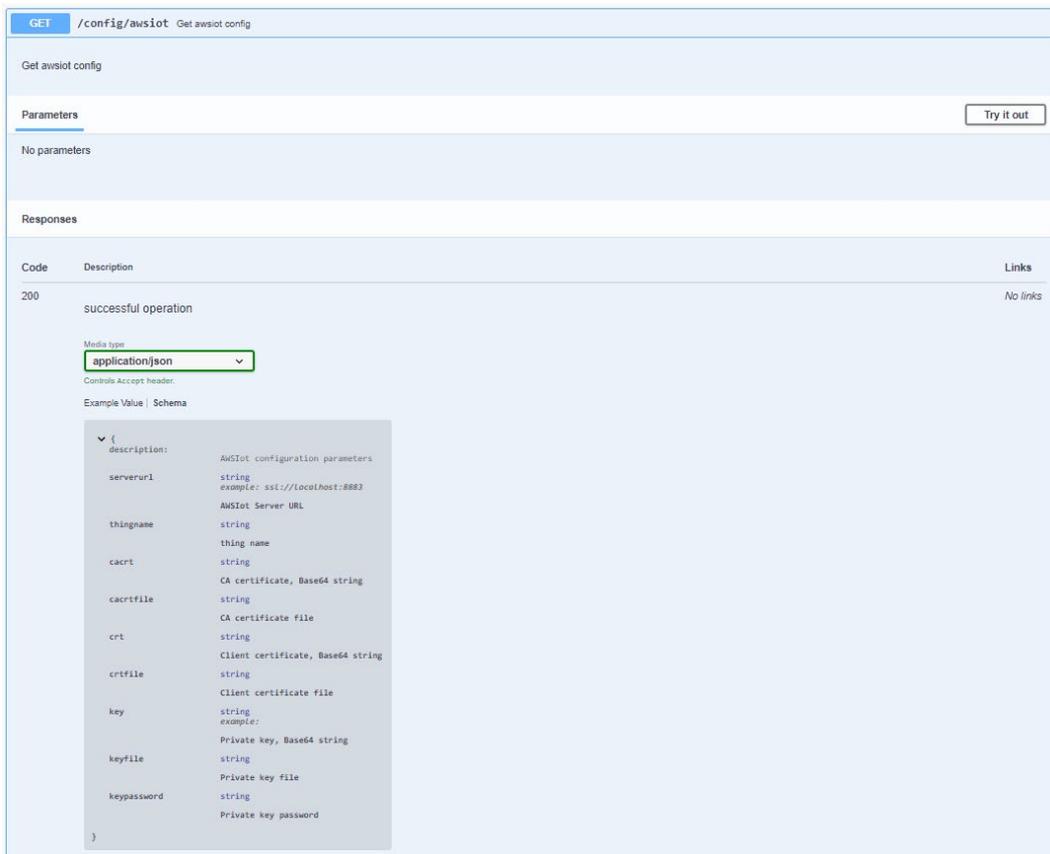
Figure 7 Logical representation of a smart home with IoT devices connected to the cloud via a gateway and AWS IoT Core

You can create rules that define one or more actions to perform based on the data in a message. For example, you can insert, update, or query a DynamoDB table or invoke a Lambda function. Rules use expressions to filter messages. When a rule matches a message, the rules engine triggers the action using the selected properties. Rules also contain an IAM role that grants AWS IoT permission to the AWS resources used to perform the action [1].

To configure AWS IoT handler on the gateway, the following configuration parameters are required :

- AWS IoT server URL
- Thing name from AWS IoT core
- CA certificate from AWS IoT core
- Client certificate from AWS IoT core
- Private key from AWS IoT core

See [http://\[hostname\]/api-docs/interactive/index.html#/config/getAWSIoTApiConfig](http://[hostname]/api-docs/interactive/index.html#/config/getAWSIoTApiConfig)



GET /config/awsiot Get awsiot config

Get awsiot config

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	successful operation	No links

Media type:

Content Accept Header:

Example Value | Schema

```

{
  description: AWSIoT configuration parameters
  serverurl: string
  example: ssl://localhost:8883
  description: AWSIoT Server URL
  thingname: string
  description: thing name
  caert: string
  description: CA certificate, Base64 string
  cacrtfile: string
  description: CA certificate file
  crt: string
  description: Client certificate, Base64 string
  crtfile: string
  description: Client certificate file
  key: string
  example:
  Private key, Base64 string
  keyfile: string
  description: Private key file
  keypassword: string
  description: Private key password
}

```

Figure 8 AWS IoT configuration parameters

Section 4.2 shows how to get the configuration parameters from AWS IoT

4.2 Configuring AWS IoT

4.2.1 Set up your AWS Account

Before you use AWS IoT Core for the first time, complete the following tasks:

1. Sign up for an AWS account
Go to <https://portal.aws.amazon.com/billing/signup> and follow the instructions

2. Create a user and grant permissions

You can find step by step instructions using the link below -

<https://docs.aws.amazon.com/iot/latest/developerguide/setting-up.html#create-iam-user>

3. Open the AWS IoT console

Go to <https://console.aws.amazon.com/iot/home>

If you already have an AWS account and an IAM user for yourself, you can use them and skip ahead to Open the AWS IoT console [2].

4.2.2 Creating a thing and certificates

First of all you have to create a "Thing" in the "Manage" / "Things" section on the AWS IoT Service. Follow the steps below:

Create a thing in the AWS IoT registry to using the steps below

1. In the [AWS IoT console](#), in the navigation pane, choose Manage, and then choose Things.

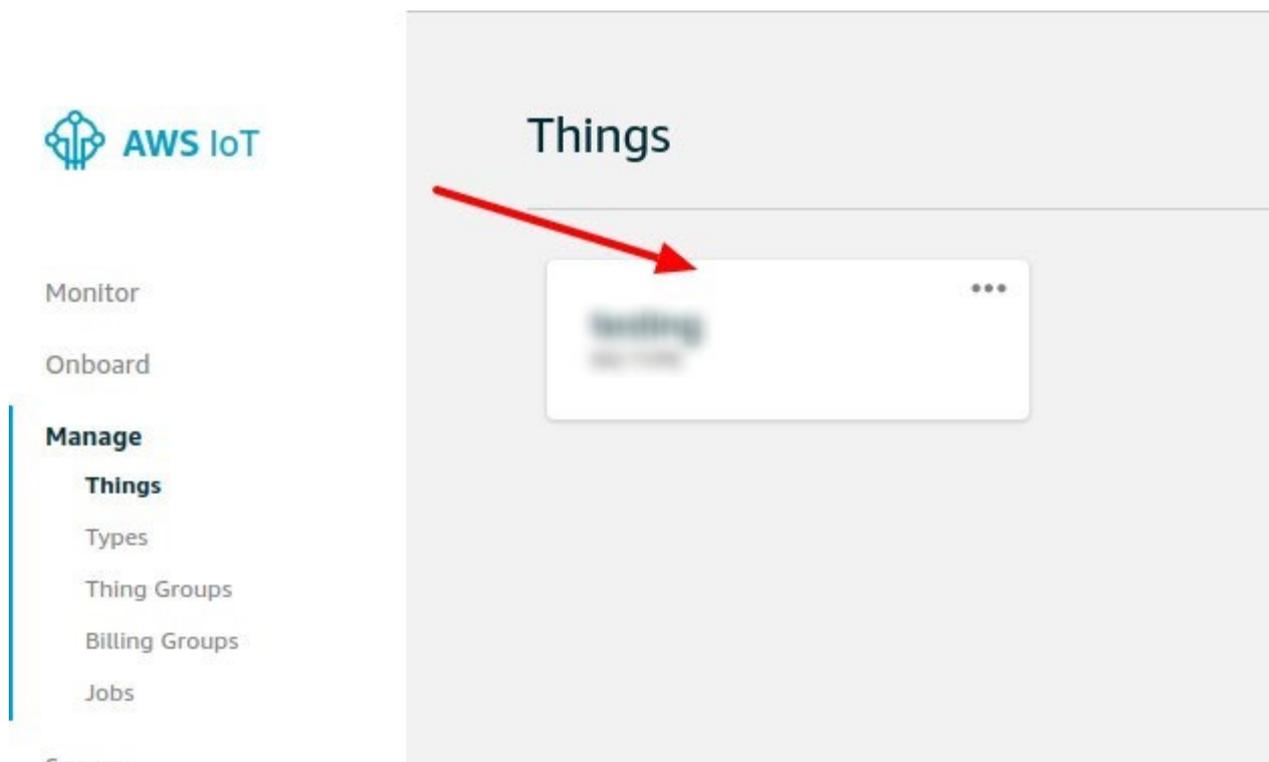


Figure 9 Creating a thing

2. If a You don't have any things yet dialog box is displayed, choose Register a thing. Otherwise, choose Create.
3. On the Creating AWS IoT things page, choose Create a single thing.
4. On the Add your device to the device registry page, enter a name for your IoT thing (for example, Squid.Link 1, and then choose Next. You can't change the name of a thing after you create it. To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

- On the Add a certificate for your thing page, choose Create certificate.

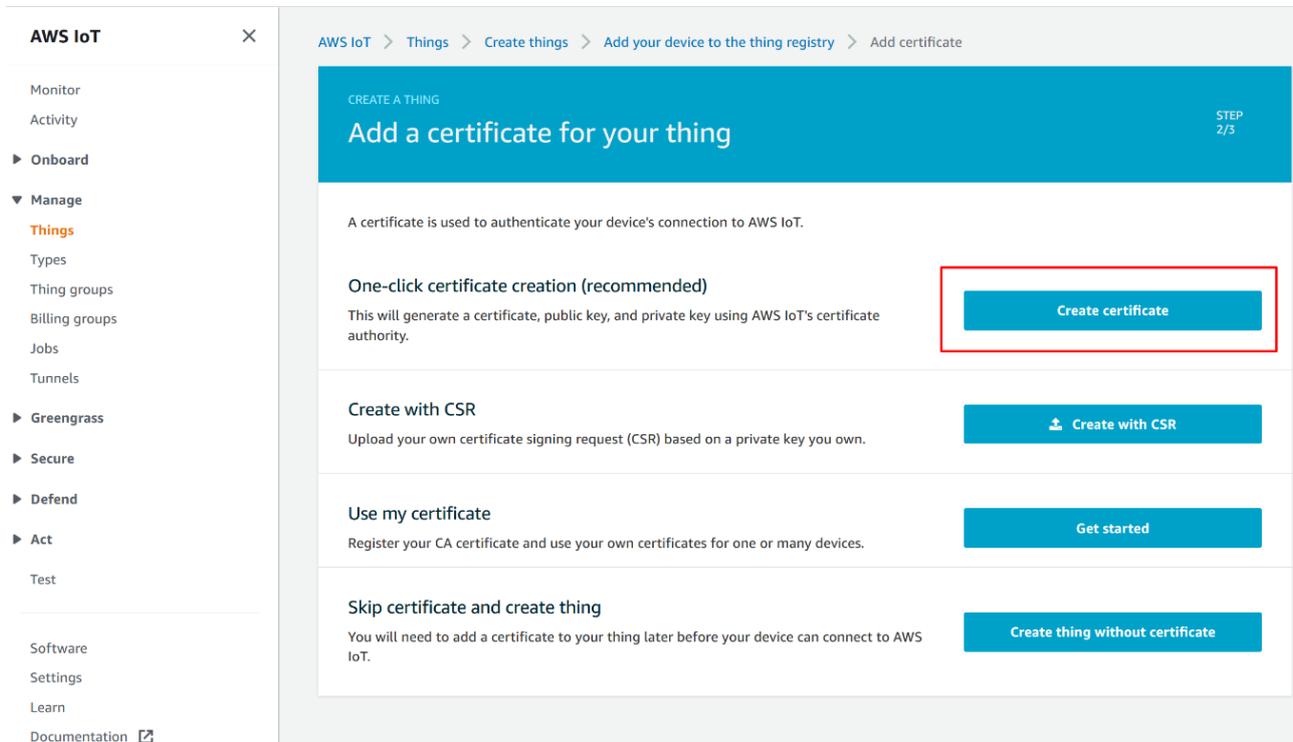


Figure 10 creating certificate

- Choose the Download links to download the certificate, private
- Important

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

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

Figure 11 Downloading Certificates

This is the only time you can download your certificate and private key.

- Choose Activate.
- Choose Attach a policy. See 4.2.3 on steps to create a policy
- For Add a policy for your thing, choose a policy (see section 4.4.2), and then choose Register Thing.

More can be found at : <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>

4.2.3 Creating a policy

Follow the steps below to create a policy

1. In the AWS IoT console, if a Get started button appears, choose it. Otherwise, in the navigation pane, expand Secure, and then choose Policies.
2. If a You don't have any policies yet dialog box appears, choose Create a policy. Otherwise, choose Create.
3. Enter a name for the AWS IoT policy
4. In the Add statements section, fill the policy statements in Json format. You can past the following Json sample.

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "iot:Connect",
"Resource": "arn:aws:iot:<region>:<account id>:client/${iot:Connection.Thing.ThingName}"
},
{
"Effect": "Allow",
"Action": "iot:Publish",
"Resource": [
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/update",
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/delete",
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/get",
"arn:aws:iot:<region>:<account id>:topic/${iot:Connection.Thing.ThingName}/*"
]
},
{
"Effect": "Allow",
"Action": "iot:Receive",
"Resource": [
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/update/*",
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/delete/*",
"arn:aws:iot:<region>:<account id>:topic/$aws/things/${iot:Connection.Thing.ThingName}/shadow/get/*",
"arn:aws:iot:<region>:<account id>:topic/${iot:Connection.Thing.ThingName}/*"
]
},
{
"Effect": "Allow",
"Action": "iot:Subscribe",
"Resource": [
"arn:aws:iot:<region>:<account id>:topicfilter/$aws/things/${iot:Connection.Thing.ThingName}/shadow/update/*",
```

```

"arn:aws:iot:<region>:<account id>:topicfilter/$aws/things/${iot:Connection.Thing.ThingName}/shadow/delete/*",
"arn:aws:iot:<region>:<account id>:topicfilter/$aws/things/${iot:Connection.Thing.ThingName}/shadow/get/*",
"arn:aws:iot:<region>:<account id>:topicfilter/${iot:Connection.Thing.ThingName}/*"
]
},
{
"Effect": "Allow",
"Action": [
"iot:GetThingShadow",
"iot:UpdateThingShadow",
"iot:DeleteThingShadow"
],
"Resource": "arn:aws:iot:<region>:<account id>:thing/${iot:Connection.Thing.ThingName}"
}
]
}
}

```

Figure 12 Creating a policy

All devices in your fleet must have with privileges that authorize intended actions only, which include (but not limited to) AWS IoT MQTT actions such as publishing messages or subscribing to topics with specific scope and context. The specific permission policies can vary for your use cases. Identify the permission policies that best meet your business and security requirements.

To find more sample policies, please go to <https://docs.aws.amazon.com/iot/latest/developerguide/example-iot-policies.html>

To learn more about AWS IoT security best practices, please check

<https://docs.aws.amazon.com/iot/latest/developerguide/security-best-practices.html>

4.2.4 Ca certificate

Ca certificate can be downloaded from:

<https://docs.aws.amazon.com/iot/latest/developerguide/server-authentication.html>

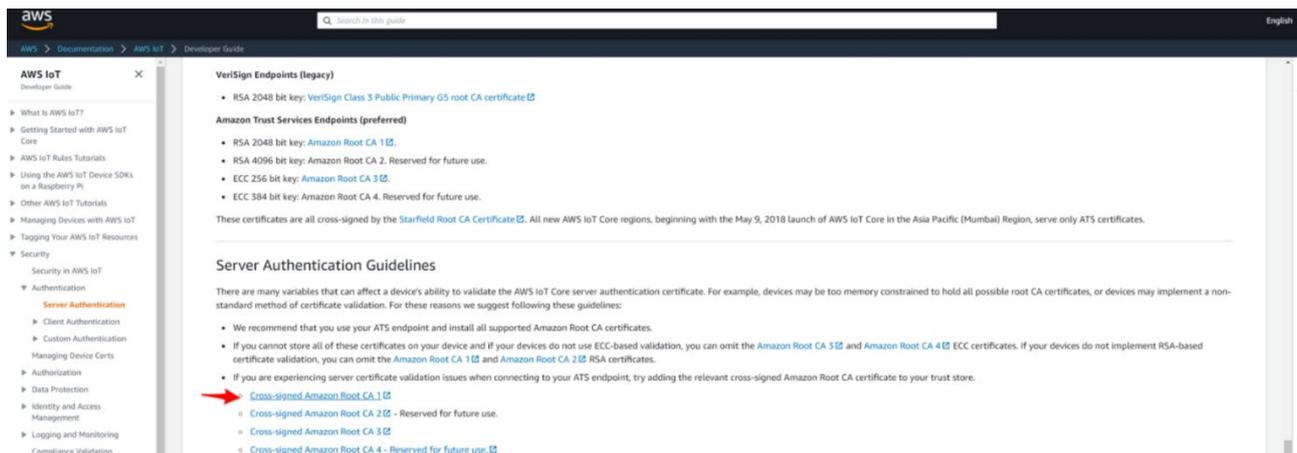


Figure 13 downloading CA certificate

4.2.5 AWS IoT server URL

To find AWS IoT server URL one should go to the created thing and select interact you will find REST API Endpoint for the thing shadow.

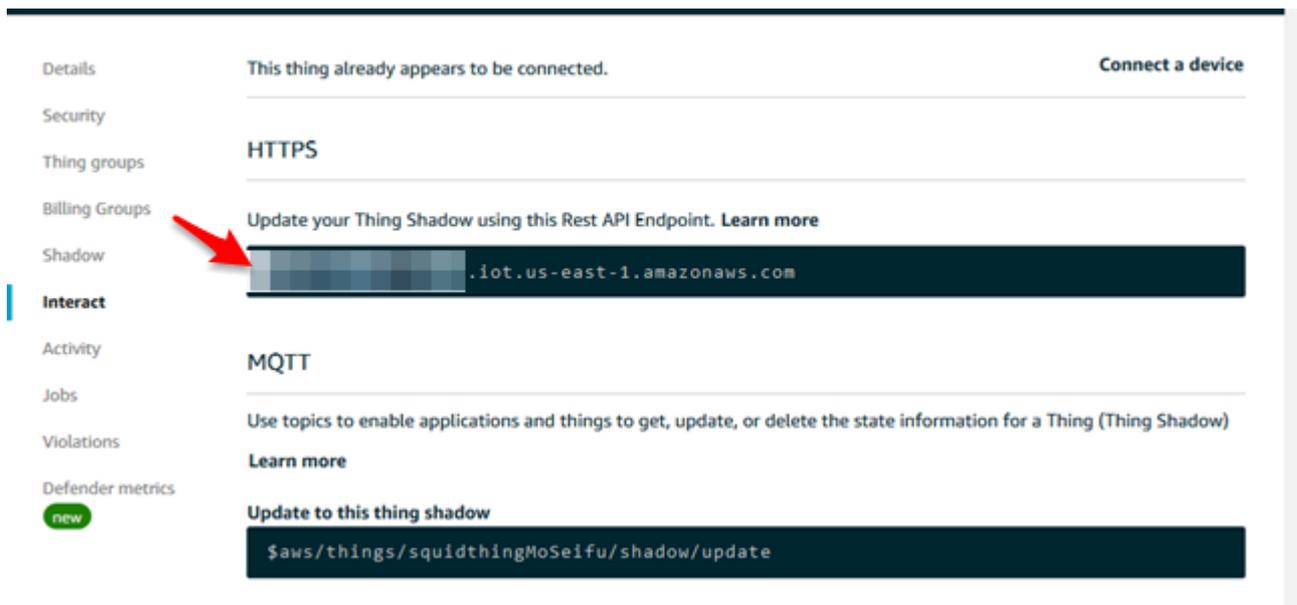
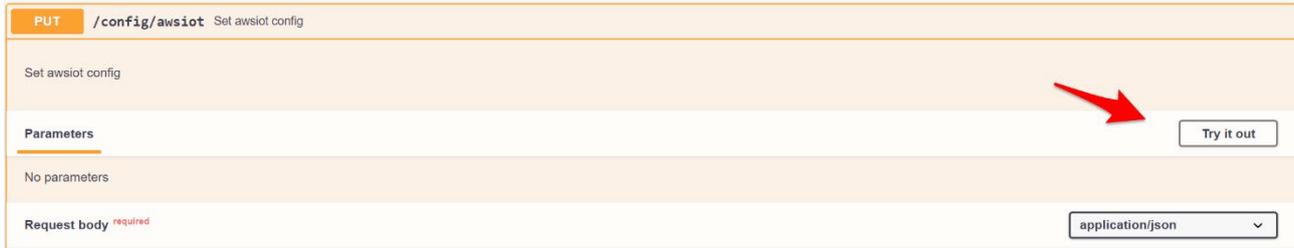


Figure 14 Rest API Endpoint for the thing shadow

4.3 Configuring Squidsmart api AWS IoT Handler

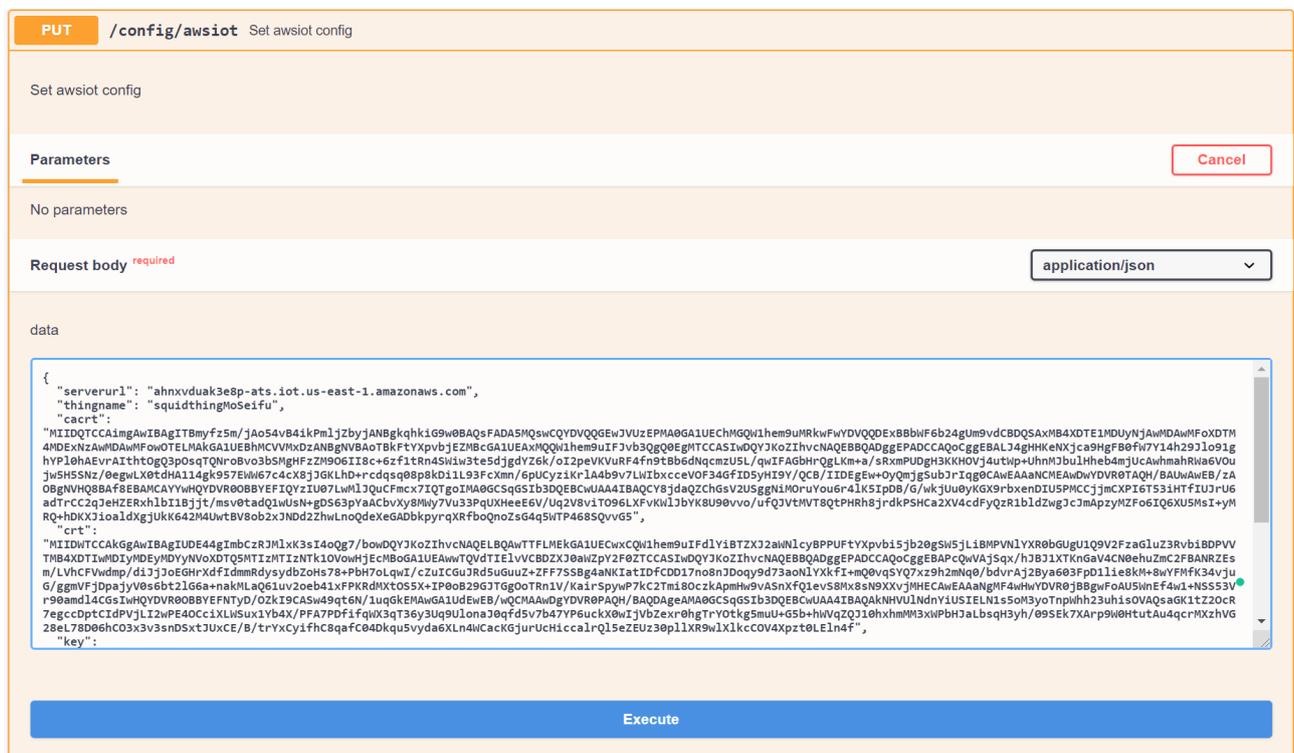
Now we have all the configuration parameters needed from the above steps. This section demonstrates how to configure the AWS IoT Handler by using the parameters.

First step is to go to `/config/awsiot` of the Squidsmart api and press try it out and paste config information and then execute buttons



The screenshot shows the API configuration interface for the endpoint `/config/awsiot`. The method is `PUT`. There is a `Try it out` button with a red arrow pointing to it. The `Request body` is set to `application/json`.

Figure 15 Set awsiot config



The screenshot shows the API configuration interface for the endpoint `/config/awsiot`. The method is `PUT`. The `Request body` is set to `application/json`. The `data` field contains the following JSON object:

```
{
  "serverurl": "ahnxvduak3e8p-ats.iot.us-east-1.amazonaws.com",
  "thingname": "squidthingMoSeifu",
  "caert": "
\"MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB41kPm1jzbyjANBgkqhkiG9w0BAQsFADA5MQswCQYDVQGEwZjVuzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDEwBBbW50b24gUm9vdCB0Q3AxMjB4XDT01MDUyNjAwMDAwMFoXDTM0MDExZjA5MjB4MDFMakGA1UEBmMjVudXZANBgNVBAoTBkF0YXN0b24gUm9vdCB0Q3AxMjB4MDFMakGA1UEAxMQW1hem9uIFJvb3Q0Q0EgMTCCASiW0QY3koZiHvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKcXJca9HGF80fw7Y14h29Jlo91gHYPI8hAevrA1tHt0gQ3p0sqTQNr0bvo3b5MghFzZM906II8c+6zftRn45iW3te5djdYZ6k/oI2peVKURF4fn9tEB6dNqcmzU5L/qwTFAGbHrQgLKm+a/sRxpUDgh3KHOVj4utWp+UhnHjbu1Hheb4mjUcAwHmahRiaw6Voujw5H5SNz/0egwLX0tdHA114gk957EwM67c4cx8jJGKLHD+rcdsq08p8kd11L93Fcxm/6pUCyziKr1A4b9v7LwIbxcceVof346fID5YH9Y/QCB/IIDEgEw+OyQmjg5ubJrIqg0CAwEAAnCMEAwDwYDVR0TAQH/BAUwAwEB/zA0BGNVHQBBA78EBAMCAYYHQYDVR0BBYEFYQYzIU07LWm1JQuCFmxc7IQTgoIMA0GCSqGSIb3DQEBQwAA4IBAQC8Y8jdaQZchGsv2U5ggN1M0ruYou6r4Lk5IPDB/g/wkjuU0yK9X9rbxendIUSPMCCjJmCXPi6T531HTFIUJrU6adTfrc2zJehZERxh1b1BjJt/ms0tadQ1u5N+gD563pYaAcbvXy8Mhy7Vu33PqUXHee6V/luq2V8v1T096LXFvKw1JbYK8U90vvo/ufQ3VtHVT8QtPHR8jrdkPShca2XV4cdfyQzr1b1dZwg7c3maPzyMZF06IQ6XUS5SI+yMRC+hDKXJ3oa1dXgJuk642M4UwtBV80b2x3NDd2ZhwLnoQdexGAdbkpyrqKRfboQnoz2564q5WTP4685Qvvg5\",
  "cert": "
\"MIIDTCCAkGAgIBAgIUDE44gImbCzRJM1kX3sI40Qg7/bowDQY3koZiHvcNAQELBQAwTFLMEKGA1UECwxQW1hem9uIFJvb3Q0Q0EgMTCCASiW0QY3koZiHvcNAQEBBQADggEPADCCAQoCggEBAPcQwVAj5sq/h3B1XTKnGv4CN0ehuZmC2FBANRZesm/LVhCFVwdmp/diJjJoEGHrXdfIdmmRdysydbZoHs78+PbH7oLQwI/cZuICguJRd5Guuz+ZF75SBg4aNIaTIDfCDD17no8nJDoqy9d73aoNLYkFI+mQ0vqSYQ7xz9h2mNq0/bdvrAj2Bya603Fpd11ie8kM+8wYFFMfK34vjuG/ggmVfDpaJyV0s6bt2Lg6a+nakMLaQ61uv2oeb41xPKRDMxtOS5X+IP80B29GJTg0oTrn1V/KairSpywP7kC2Tm180czApmHw9vASnXfQ1evS8Mx8sN9XXvJMHECAwEAAnGMF4wHwYDVR0BBgwFoAU5WnEf4w1+NSS53Vr90amD14Ccs2WqYDVR0BBYEFNTYD/OZk19CAs49qt6N/IuqkEPAAwGA1UEwEB/wCMAAwDgYDVR0RBAQH/BAQDAgeAA0GCSqGSIb3DQEBQwAA4IBAQAkNHVUIndNY1US1ELN1s50M3yotNpWhh23uh150VQsagK1T220cR7egc0ptCldPvJLI2wPE40CC1XLM5ux1VbX/FFA79DF1fjw03qT36y3Ug9U1ona30qf5v7b47P6ucX8wIvBZeXr0h9TfY0tkg5muU+GSb-hhwVgQ31ghxhmMh3xwPbH7alsq3yh/09SEK7XAr9p90tutAu4qcrMXZhgV28eL7800eC03x3v3sNdSxtJUXE/B/trYxyifhc8qafC04dkqu5vyda6XLN4wCack6JurUchiCca1rQ15eZUz30p1XR9w1XkC0V4Xpz20E1n4f\",
  "key": "
\"key\": \"key\"
\"MIIDTCCAkGAgIBAgIUDE44gImbCzRJM1kX3sI40Qg7/bowDQY3koZiHvcNAQELBQAwTFLMEKGA1UECwxQW1hem9uIFJvb3Q0Q0EgMTCCASiW0QY3koZiHvcNAQEBBQADggEPADCCAQoCggEBAPcQwVAj5sq/h3B1XTKnGv4CN0ehuZmC2FBANRZesm/LVhCFVwdmp/diJjJoEGHrXdfIdmmRdysydbZoHs78+PbH7oLQwI/cZuICguJRd5Guuz+ZF75SBg4aNIaTIDfCDD17no8nJDoqy9d73aoNLYkFI+mQ0vqSYQ7xz9h2mNq0/bdvrAj2Bya603Fpd11ie8kM+8wYFFMfK34vjuG/ggmVfDpaJyV0s6bt2Lg6a+nakMLaQ61uv2oeb41xPKRDMxtOS5X+IP80B29GJTg0oTrn1V/KairSpywP7kC2Tm180czApmHw9vASnXfQ1evS8Mx8sN9XXvJMHECAwEAAnGMF4wHwYDVR0BBgwFoAU5WnEf4w1+NSS53Vr90amD14Ccs2WqYDVR0BBYEFNTYD/OZk19CAs49qt6N/IuqkEPAAwGA1UEwEB/wCMAAwDgYDVR0RBAQH/BAQDAgeAA0GCSqGSIb3DQEBQwAA4IBAQAkNHVUIndNY1US1ELN1s50M3yotNpWhh23uh150VQsagK1T220cR7egc0ptCldPvJLI2wPE40CC1XLM5ux1VbX/FFA79DF1fjw03qT36y3Ug9U1ona30qf5v7b47P6ucX8wIvBZeXr0h9TfY0tkg5muU+GSb-hhwVgQ31ghxhmMh3xwPbH7alsq3yh/09SEK7XAr9p90tutAu4qcrMXZhgV28eL7800eC03x3v3sNdSxtJUXE/B/trYxyifhc8qafC04dkqu5vyda6XLN4wCack6JurUchiCca1rQ15eZUz30p1XR9w1XkC0V4Xpz20E1n4f\",
  \"key\": \"key\"
\"MIIDTCCAkGAgIBAgIUDE44gImbCzRJM1kX3sI40Qg7/bowDQY3koZiHvcNAQELBQAwTFLMEKGA1UECwxQW1hem9uIFJvb3Q0Q0EgMTCCASiW0QY3koZiHvcNAQEBBQADggEPADCCAQoCggEBAPcQwVAj5sq/h3B1XTKnGv4CN0ehuZmC2FBANRZesm/LVhCFVwdmp/diJjJoEGHrXdfIdmmRdysydbZoHs78+PbH7oLQwI/cZuICguJRd5Guuz+ZF75SBg4aNIaTIDfCDD17no8nJDoqy9d73aoNLYkFI+mQ0vqSYQ7xz9h2mNq0/bdvrAj2Bya603Fpd11ie8kM+8wYFFMfK34vjuG/ggmVfDpaJyV0s6bt2Lg6a+nakMLaQ61uv2oeb41xPKRDMxtOS5X+IP80B29GJTg0oTrn1V/KairSpywP7kC2Tm180czApmHw9vASnXfQ1evS8Mx8sN9XXvJMHECAwEAAnGMF4wHwYDVR0BBgwFoAU5WnEf4w1+NSS53Vr90amD14Ccs2WqYDVR0BBYEFNTYD/OZk19CAs49qt6N/IuqkEPAAwGA1UEwEB/wCMAAwDgYDVR0RBAQH/BAQDAgeAA0GCSqGSIb3DQEBQwAA4IBAQAkNHVUIndNY1US1ELN1s50M3yotNpWhh23uh150VQsagK1T220cR7egc0ptCldPvJLI2wPE40
```

GET /config/awsiot/status Get awsiot status

Get awsiot status

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X GET "http://192.168.0.27/ssapi/config/awsiot/status" -H "accept: application/json"
```

Request URL

```
http://192.168.0.27/ssapi/config/awsiot/status
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "started": true, "connected": true }</pre> <p>Response headers</p> <pre>content-length: 33 content-type: application/json;charset=utf-8 date: Sun, 26 Apr 2020 10:34:19 GMT server: Jetty(9.4.z-SNAPSHOT)</pre>

Figure 17 Successful connection

4.4 Whitelisting devices

In order to enable ZigBee devices to send datapoint to shadow, one has to post whitelist logic. For example the following logic shows how to enable all ZigBee data points. It is important to remember that AWS IoT handler works under the principle of denying all by default, principle of least privilege.

```
{
  "enabled": true,
  "metadata": "",
  "name": "zibgee datapoints only",
  "topics": [
    "zb/dev+/ldev/#"
  ]
}
```

POST /config/awsiot/whitelist Add a new awsiot whitelist

Parameters Cancel

No parameters

Request body required application/json

Logic awsiot whitelist that should be added

```

{
  "enabled": true,
  "metadata": "",
  "name": "whitelist1",
  "topics": [
    "zb/dev/+/1dev/#"
  ]
}

```

Execute

Figure 18 Whitelisting devices

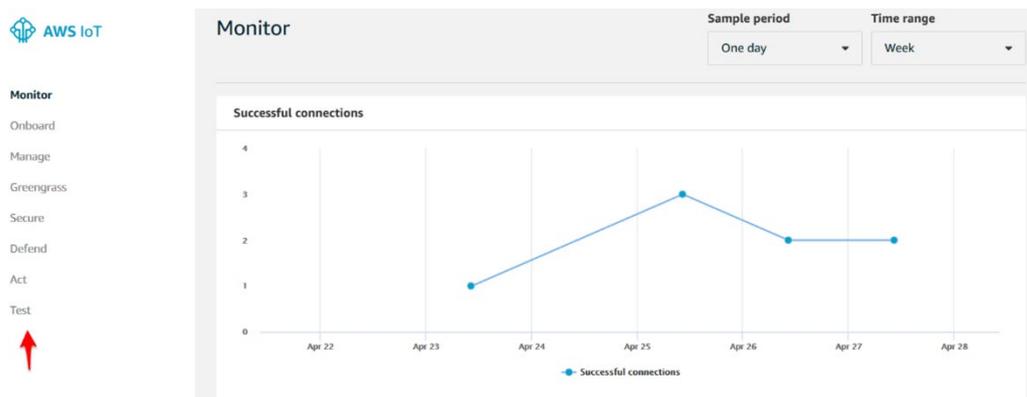
5 Interoperability with Device Shadows over MQTT

The above steps show how to configure the AWS IoT handler to communicate with AWS IoT Core and check for successful connection. Current state information of the device can be seen from the device shadow.

In this section we demonstrate how to view device MQTT messages with the AWS IoT MQTT client:

To view MQTT messages follow the following steps:

- 1) In the AWS IoT console, choose Test in the left navigation pane



- 2) Subscribe to the topic on which your IoT thing publishes. To get all updates you can use # as in the example below

The screenshot shows the AWS IoT MQTT client interface. On the left, there is a navigation menu with options: Monitor, Onboard, Manage, Greengrass, Secure, Defend, Act, and Test. The main area is titled "MQTT client" and shows "Connected as iotconsole-1588069509870-0". The interface is divided into two main sections: "Subscriptions" and "Publish".

In the "Subscriptions" section, there is a "Subscribe to a topic" and a "Publish to a topic" button. Below these, there is a list of subscriptions with a header "#".

In the "Publish" section, there is a "Publish" button and a "Specify a topic and a message to publish with a QoS of 0." instruction. Below this, there is a text input field with the topic "#". To the right of the input field is a "Publish to topic" button. Below the input field, there is a code editor showing a JSON message:

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

Below the code editor, there is a message log showing a received message from the topic "\$aws/things/squidthingMoSeifu/shadow/updates" at "Apr 28, 2020 12:50:14 PM +0200". The message content is a JSON object:

```
{
  "previous": {
    "state": {
      "reported": {
        "zb": {
          "dev_3": {
            "ldev_smartplug": {
              "data_summationdelivered": 0,
              "data_summationreceived": 0,
              "data_demand": 0,
              "data_acfrequency": 50,
              "data_voltage": 232.39,
              "data_onoff": true,
              "data_current": 0
            },
            "ldev_spalarm": {
              "data_powerfailure": false,
              "data_overcurrent": false,
              "data_overtemperature": false,
              "data_highconsumption": false,
              "data_lowconsumption": true
            }
          }
        }
      }
    }
  }
}
```

- 3) Publish on topic: On the MQTT client page, in the Publish section, in the Specify a topic and a message to publish field, enter - \$aws/things/mqtttest/shadow/update/

The screenshot shows the AWS IoT MQTT client interface. On the left, there is a navigation menu with options: Monitor, Onboard, Manage, Greengrass, Secure, Defend, Act, and Test. The main area is titled "MQTT client" and shows "Connected as iotconsole-1588240193003-3". The interface is divided into two main sections: "Subscriptions" and "Publish".

In the "Subscriptions" section, there is a "Subscribe to a topic" and a "Publish to a topic" button. Below these, there is a list of subscriptions with a header "#". One subscription is visible: "\$aws/things/squidthingMoSeifu/shadow/update".

In the "Publish" section, there is a "Publish" button and a "Specify a topic and a message to publish with a QoS of 0." instruction. Below this, there is a text input field with the topic "\$aws/things/squidthingMoSeifu/shadow/update". To the right of the input field is a "Publish to topic" button. Below the input field, there is a code editor showing a JSON message:

```
1 {
2   "state": {
3     "desired": {
4       "zb": {
5         "dev_3": {
6           "ldev_smartplug": {
7             "data_onoff": true
8           }
9         }
10      }
11    }
12  }
```

Below the code editor, there is a message log showing a received message from the topic "\$aws/things/squidthingMoSeifu/shadow/update" at "Apr 30, 2020 12:00:31 PM +0200". The message content is a JSON object:

```
{
  "state": {
    "reported": {
      "reported": {
        "dev_3": {
          "ldev_smartplug": {
            "data_voltage": 230.3
          }
        }
      }
    }
  },
  "clientToken": "08be9ad32-0734-4696-8766-549fb083e288"
}
```

Figure 19 Publish to a topic

The device shadow can also be used to get and set the state of the device over MQTT . To set state of the device one has to put desired parameters by editing the shadow document.

For example the below json command changes 'data_onoff' state of the smart plug to true, (to switch on the smart plug).

```
"state": {
  "desired": {
    "zb": {
      "dev_8": {
        "ldev_smartplug": {
          "data_onoff": true
        }
      }
    }
  }
}
```

Details

Shadow ARN

Security

A shadow ARN uniquely identifies the shadow for this thing. [Learn more](#)

Thing groups

Billing Groups

Shadow

Interact

Shadow Document

Delete Edit

Activity

Last update: Apr 30, 2020 12:10:33 PM +0200

Jobs

Shadow state:

Violations

Defender metrics

new

```
{
  "desired": {
    "welcome": "aws-iot",
    "zb": {
      "dev_8": {
        "ldev_smartplug": {
          "data_onoff": true
        }
      }
    }
  },
  "reported": {
    "zb": {
      "dev_8": {
        "ldev_smartplug": {
          "data_onoff": true,
          "data_voltage": 230.65,
          "data_acfrequency": 50.02
        },
        "ldev_diagnostic": {
```

6 Updates

Updates for the Squid Smart App are available for download at the Pre-built Rootfs page.

<https://supportforum.develcoproducts.com/squidlink/prebuilt-rootfs>

7 Contact Information

Technical support: Please contact Develco Products for support.
products@develcoproducts.com

Sales: Please contact Develco Products for information on prices, availability, and lead time.
info@develcoproducts.com

8 References

[1] AWS, "https://aws.amazon.com/iot-core/," [Online]. [Accessed 2020].

[2] "AWS IoT Core," [Online]. Available: <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>.