DEVELCO
PRODUCTS

Squid.link gateway AWS IoT Core:
Getting started

Revised 25.01.2022

Content

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.

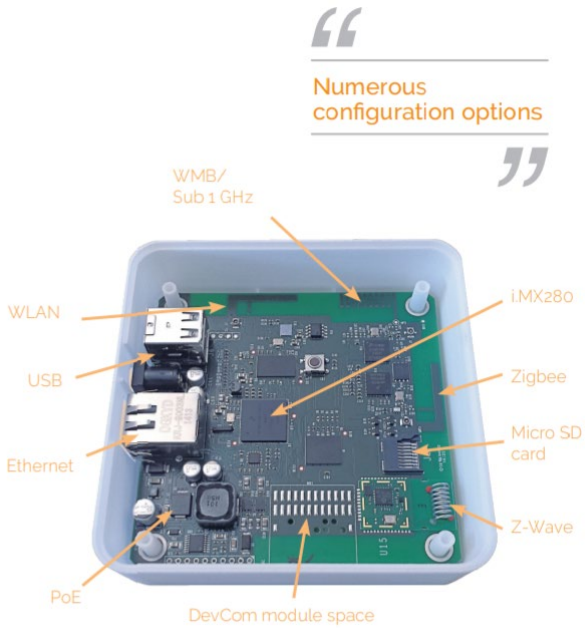


Figure 1 Squid.link gateway

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

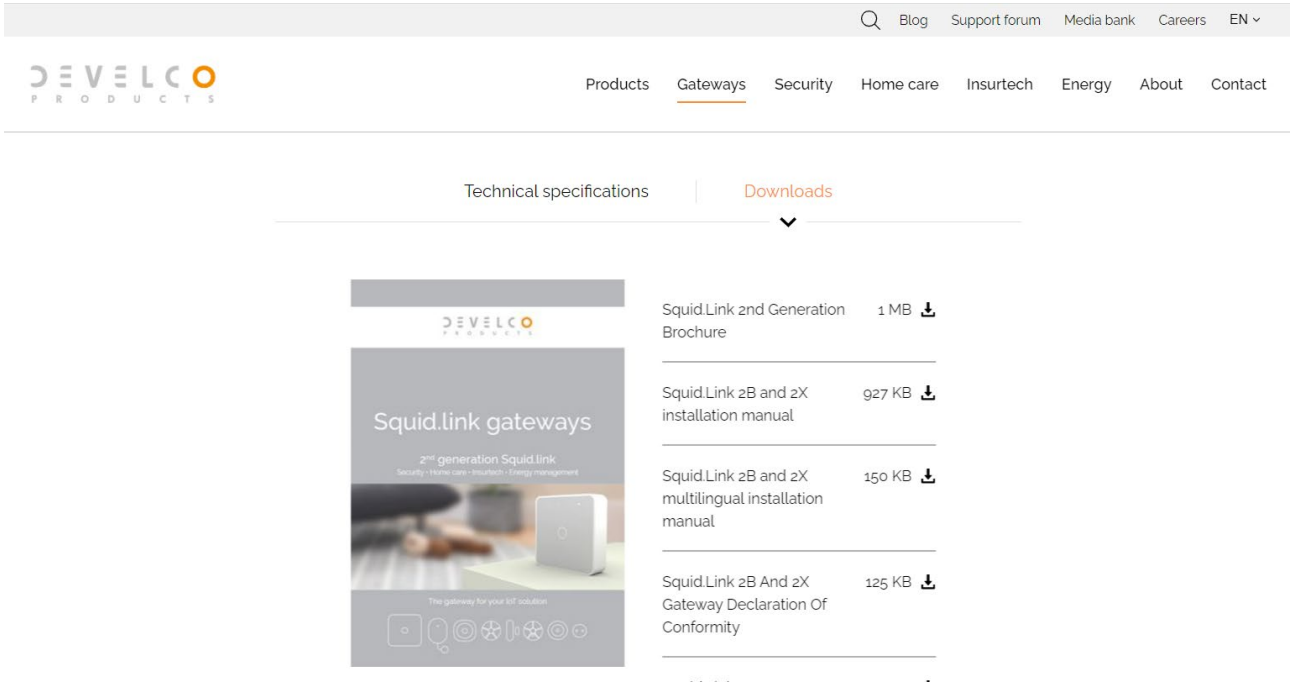


Figure 2 Squid.link gateways installation manual

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 logicaly 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.

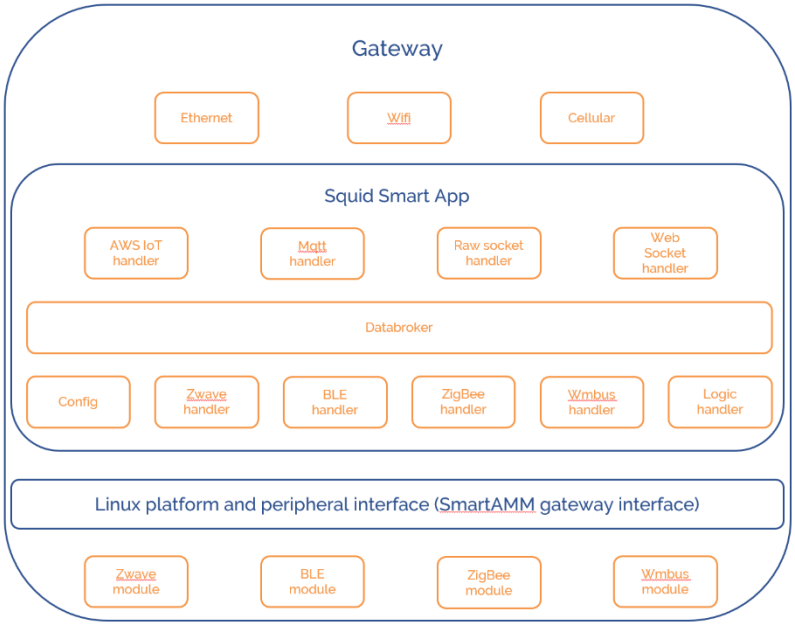


Figure 3 Gateway Architecture with Squid Smart APP as middleware

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

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/

To get the list of logical endpoints on a specific device send a GET request to:
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://api.squid.link/

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

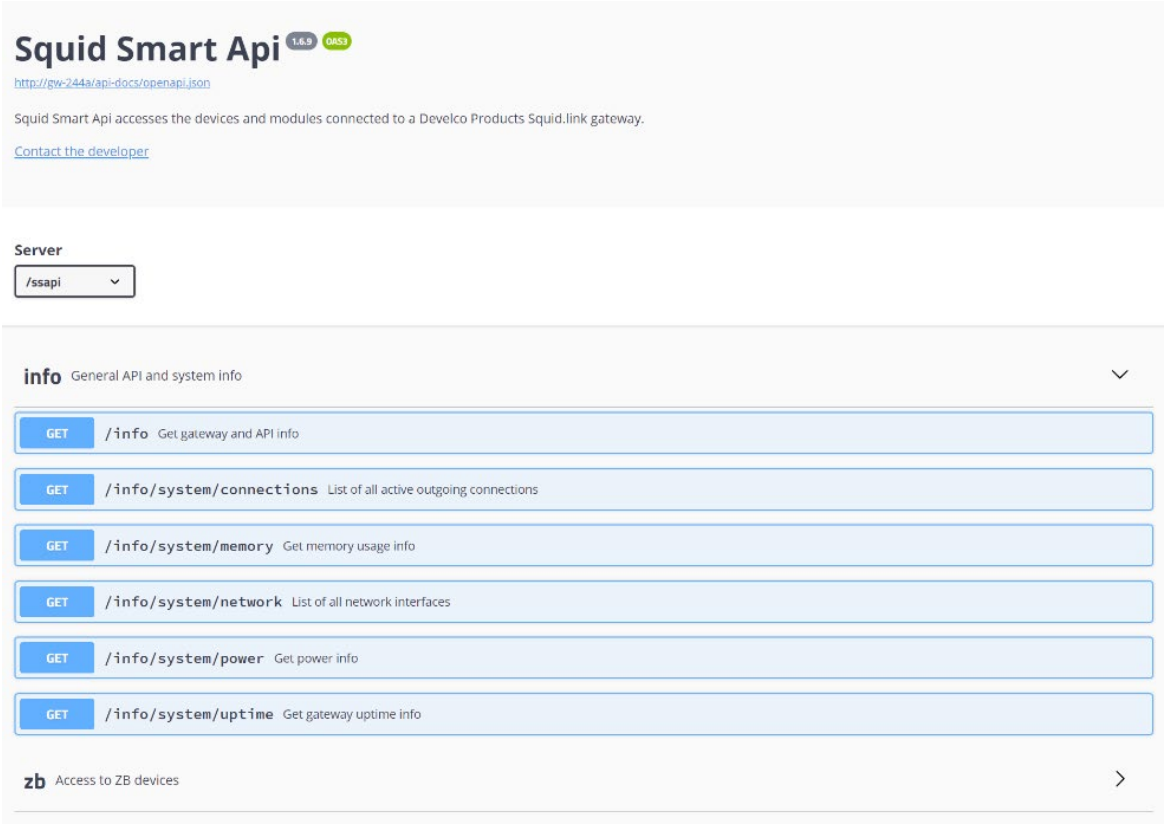


Figure 4 Squid Smart API

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)

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.

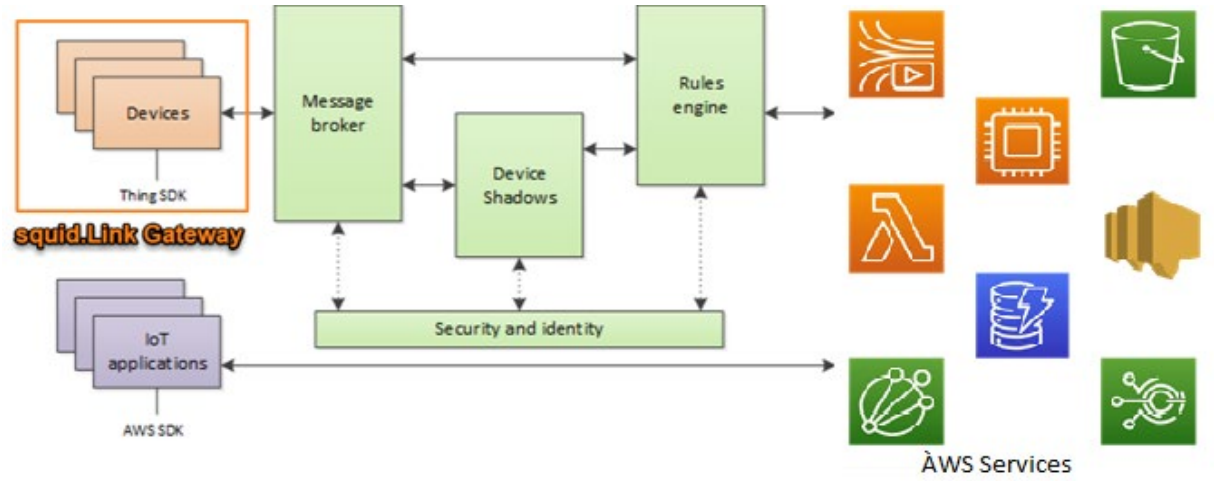


Figure 6 AWS IoT Core system [1]

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 recommend 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.

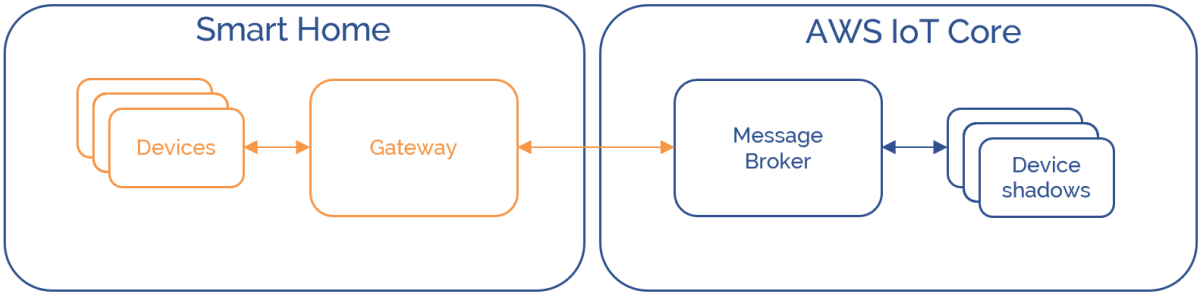


Figure 7 Logical representation of a smart home with IoT devices connected to the cloud via a gateway and AWS IoT Core

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)

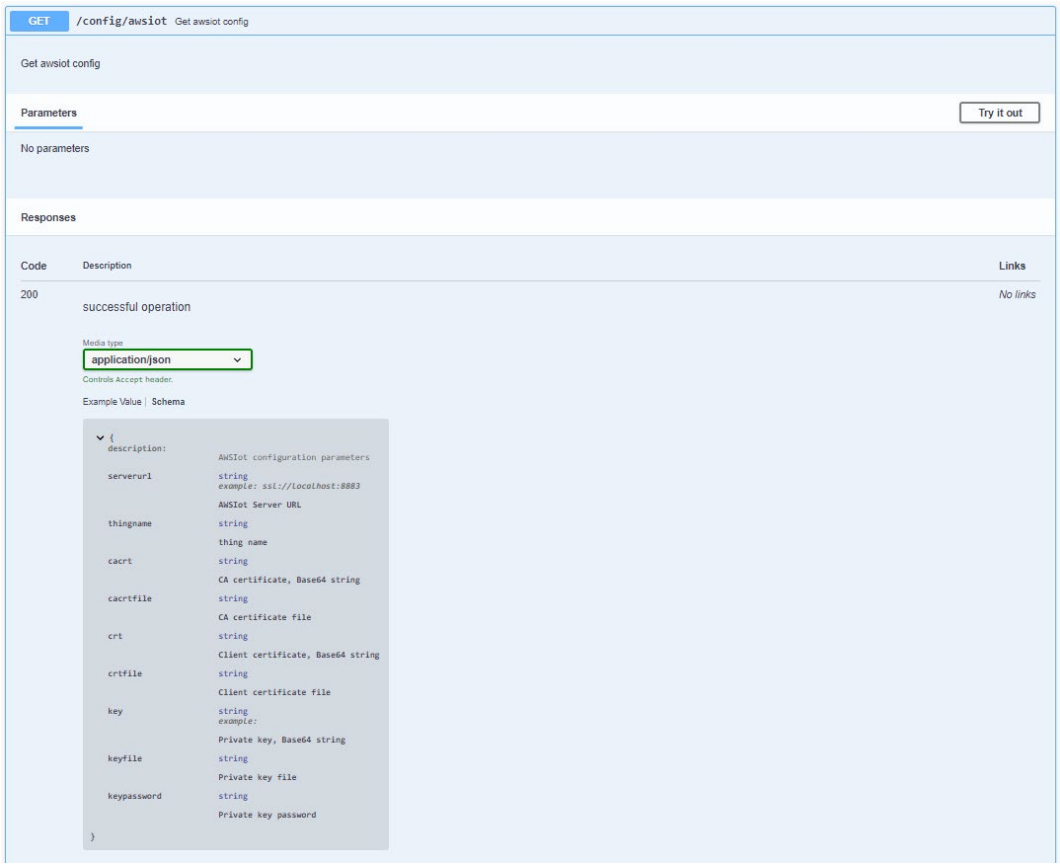


Figure 8 AWS IoT configuration parameters

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.

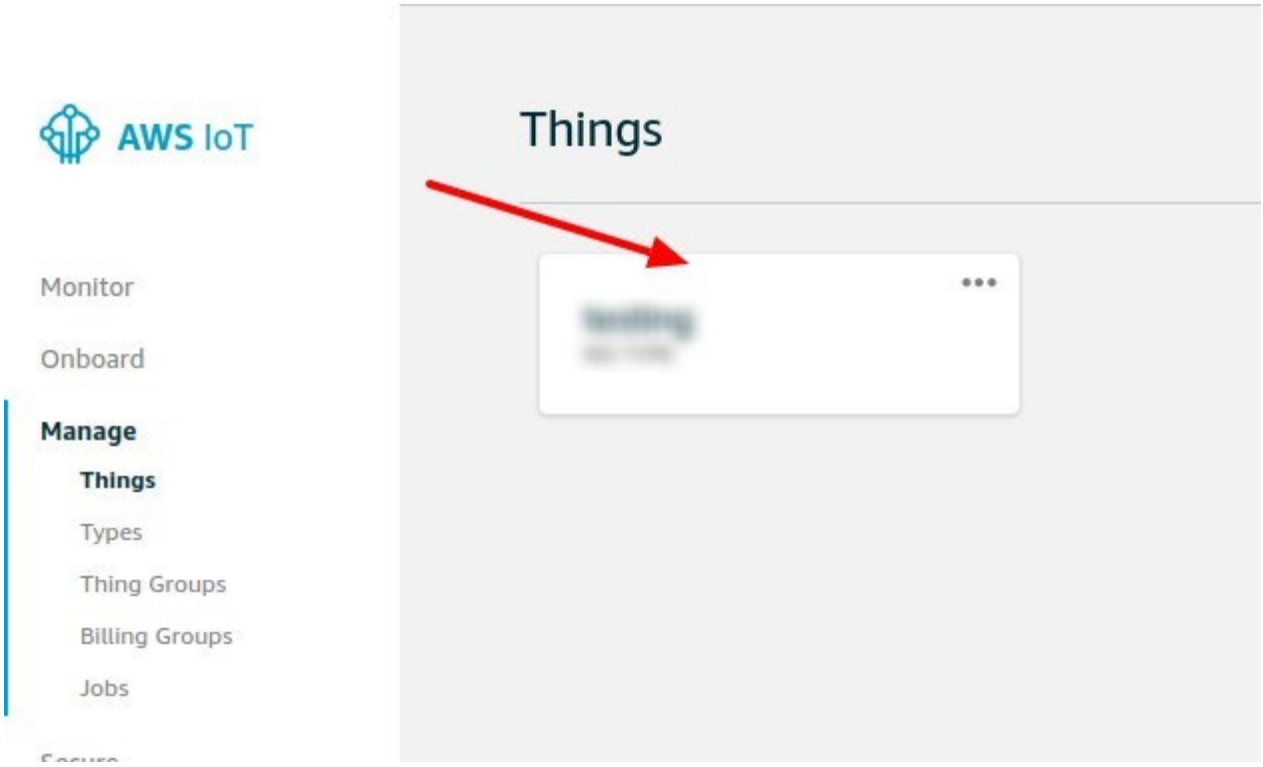


Figure 9 Creating a thing

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.

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

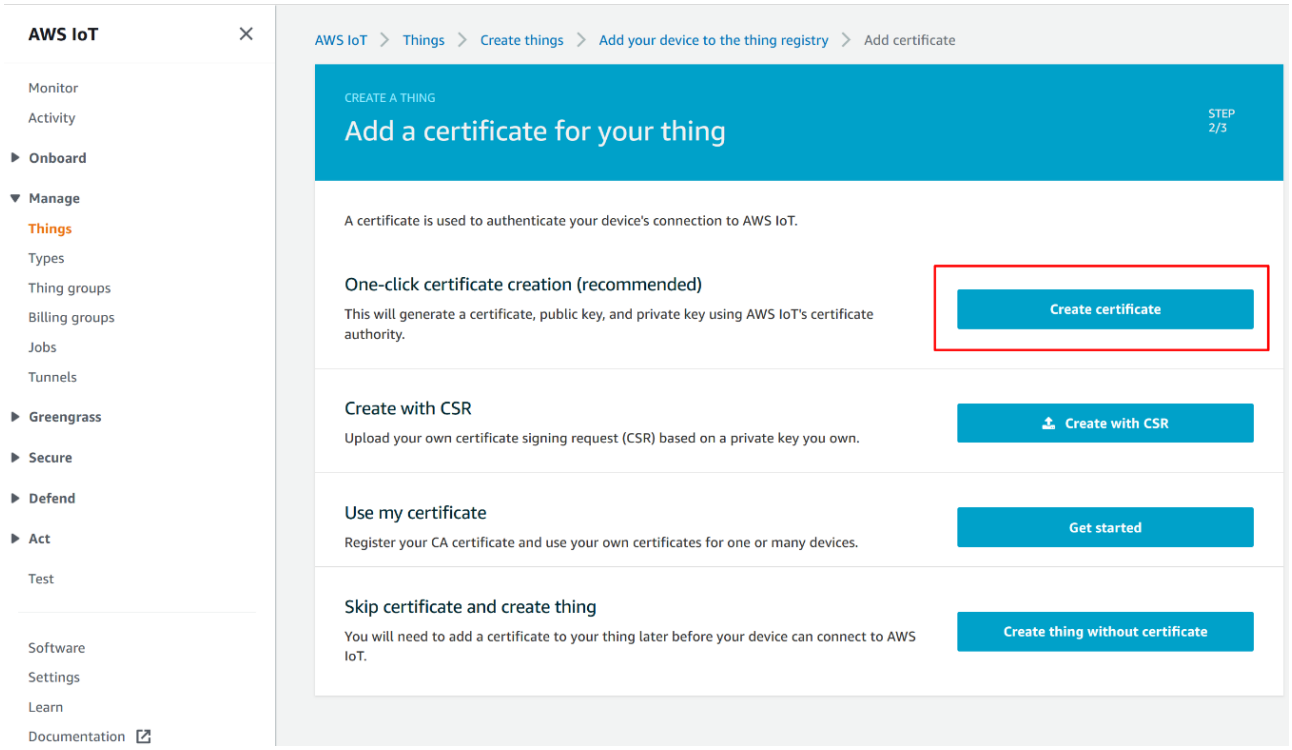


Figure 10 creating certificate

6. Choose the Download links to download the certificate, private

Important

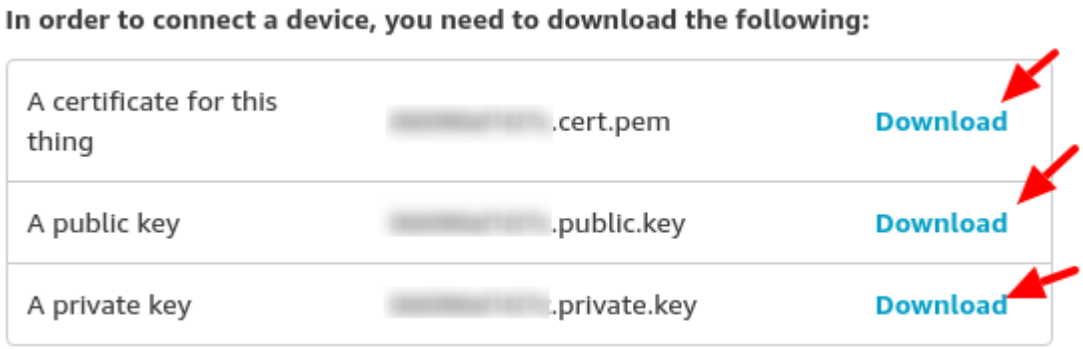


Figure 11 Downloading Certificates

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

7. Choose Activate.
8. Choose Attach a policy. See 4.2.3 on steps to create a policy
9. For Add a policy for your thing, choose a policy (se 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}"
}
]
}

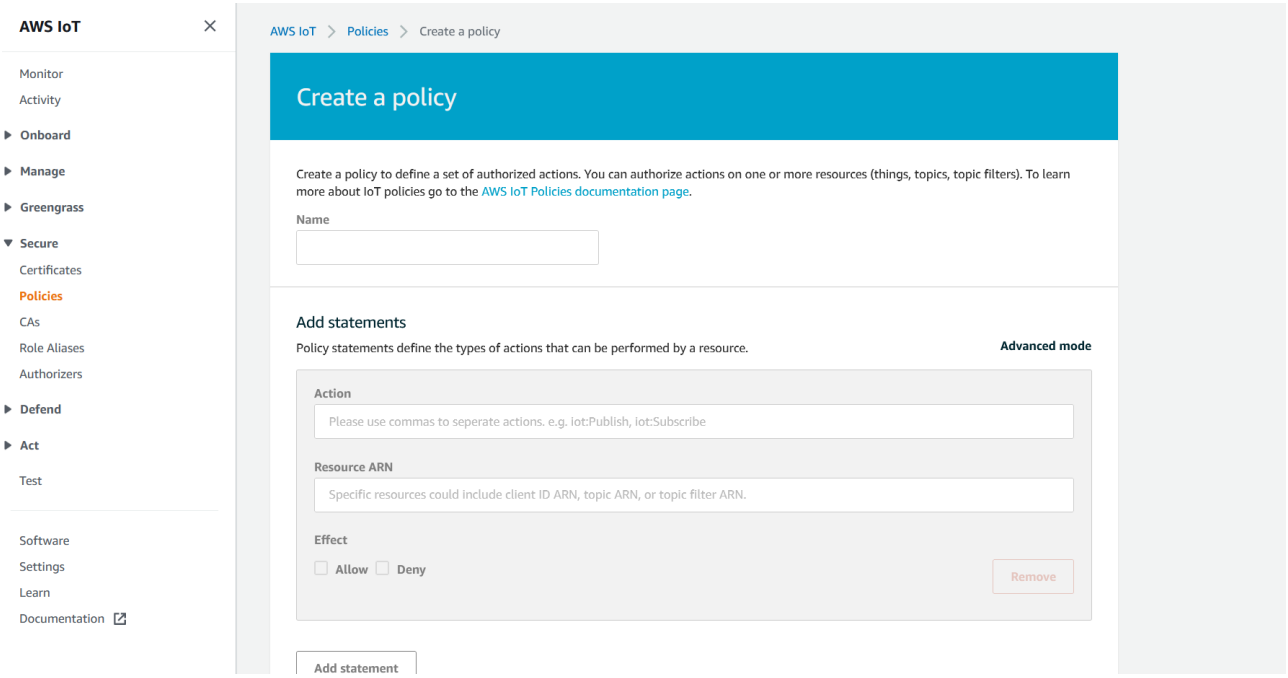


Figure 12 Creating a policy

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

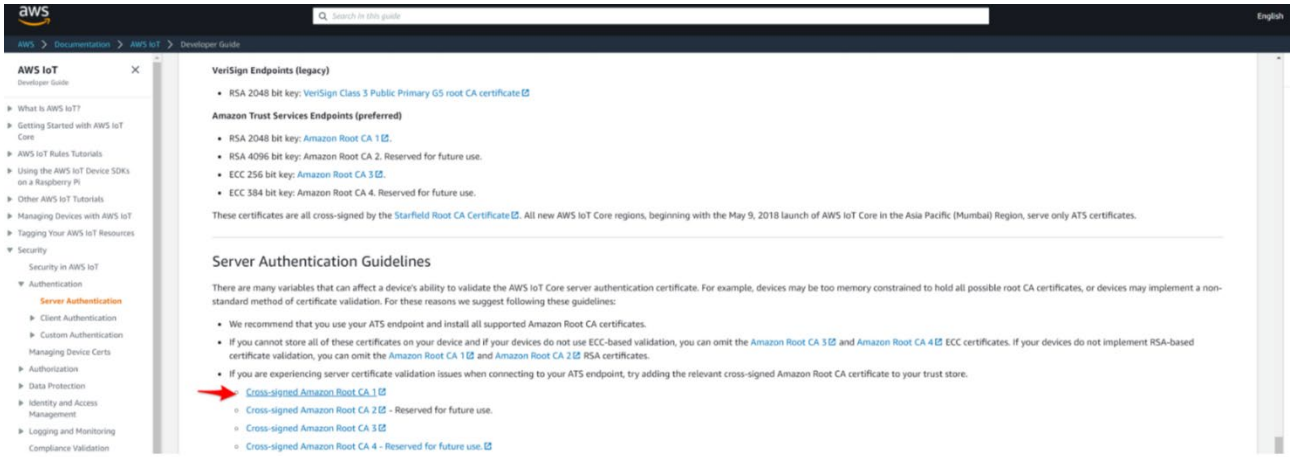


Figure 13 downloading CA certificate

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.

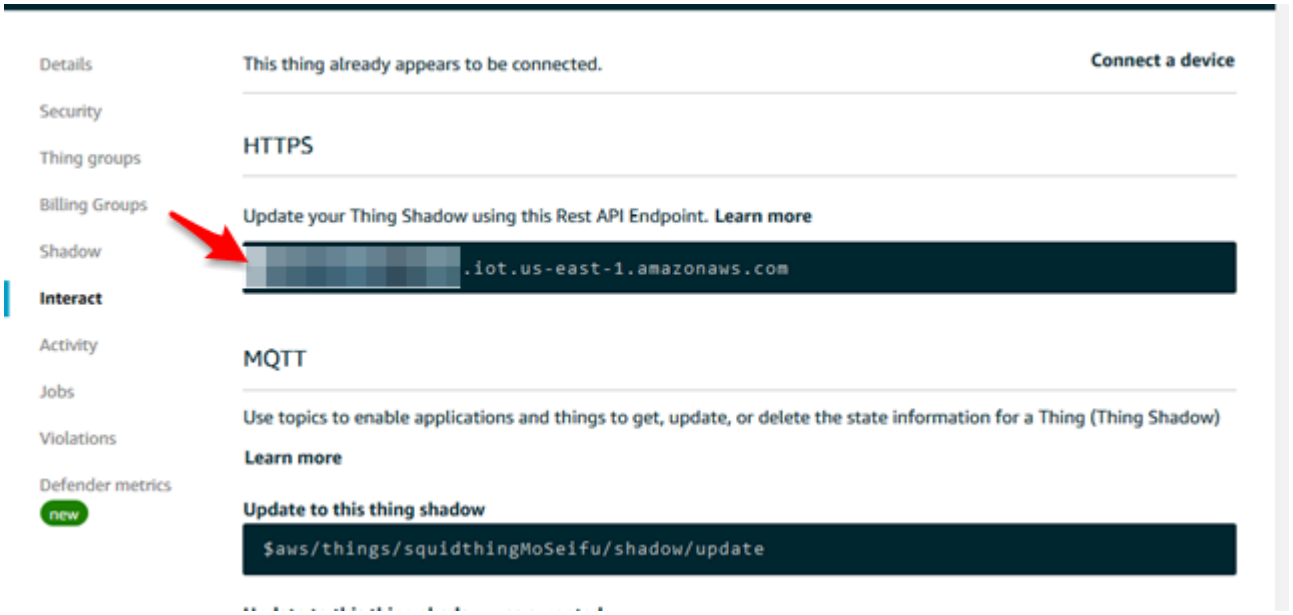


Figure 14 Rest API Endpoint for the thing shadow

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

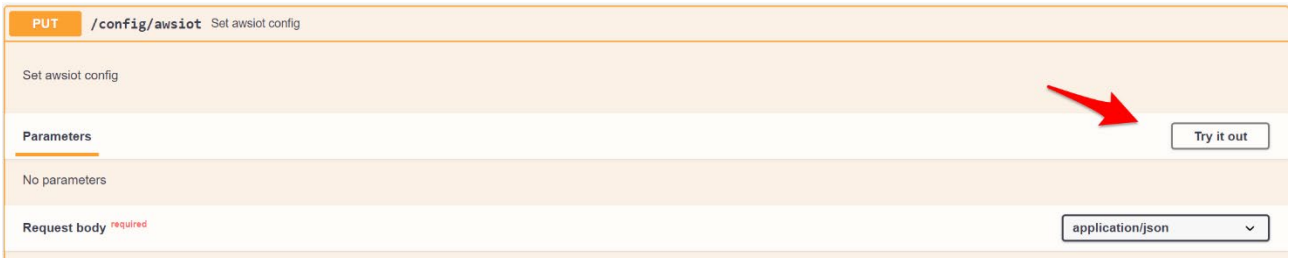


Figure 15 Set awsiot config

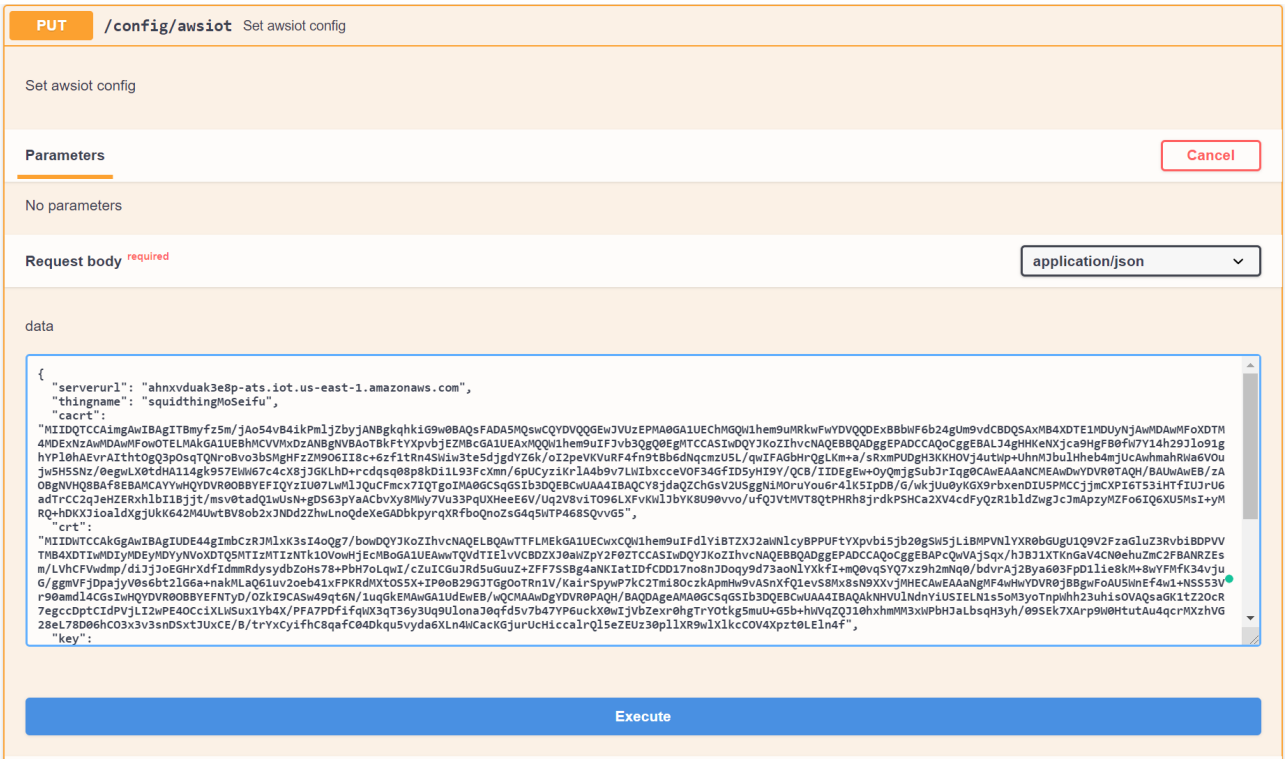


Figure 16 Executing the configuration remember to put corresponding configuration parameters

If connection is successful, you can see the following message at /config/awsiot/status

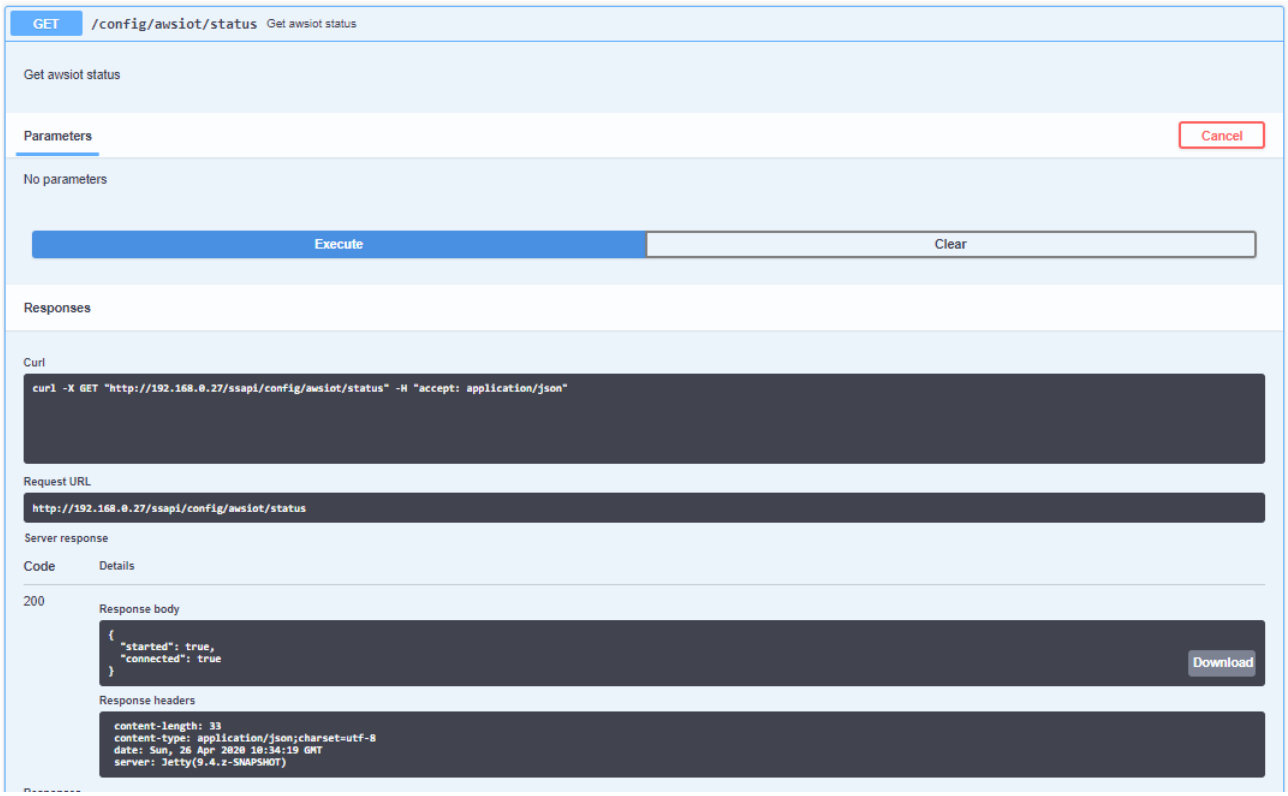


Figure 17 Successful connection

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 privilage.

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

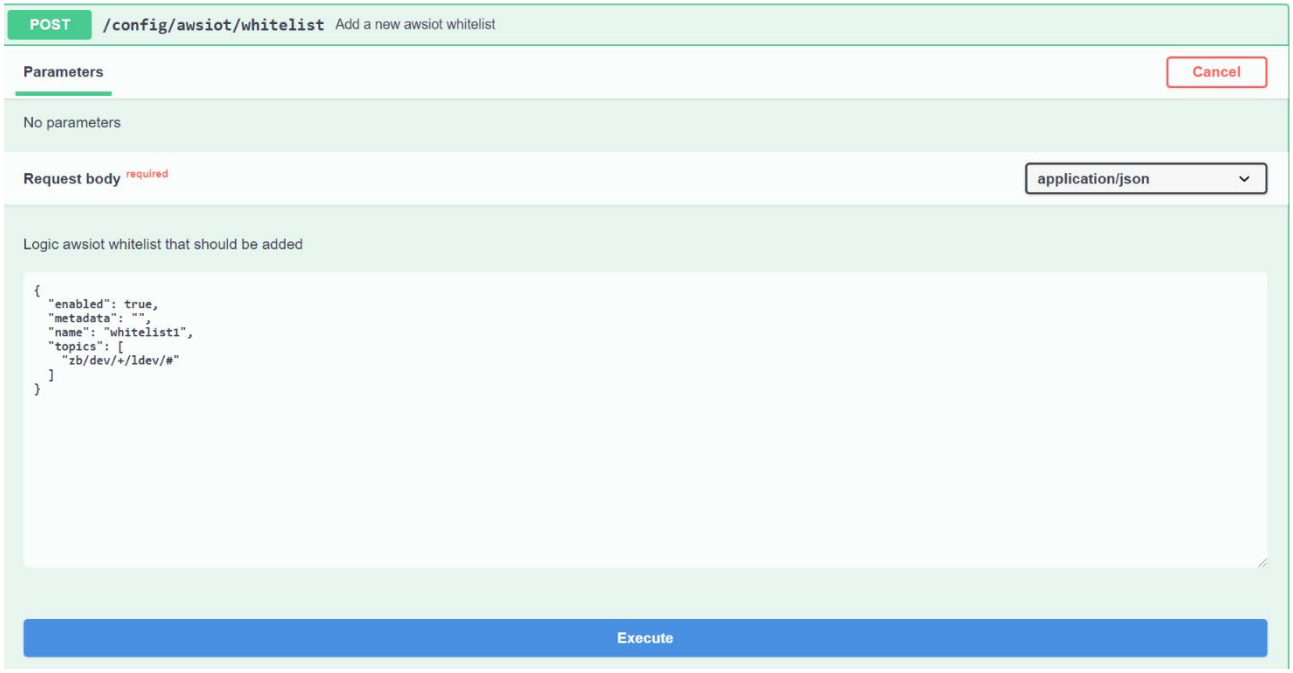


Figure 18 Whitelisting devices

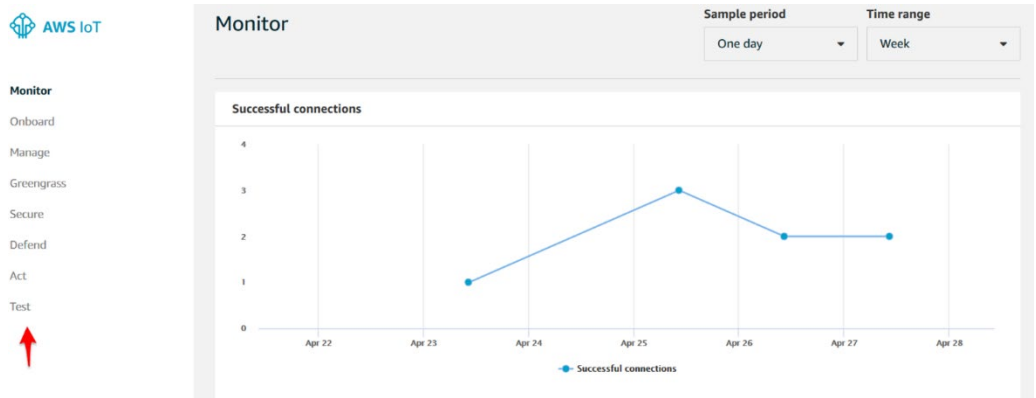
5 Interoperability with Device Shadows over MQTT

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.

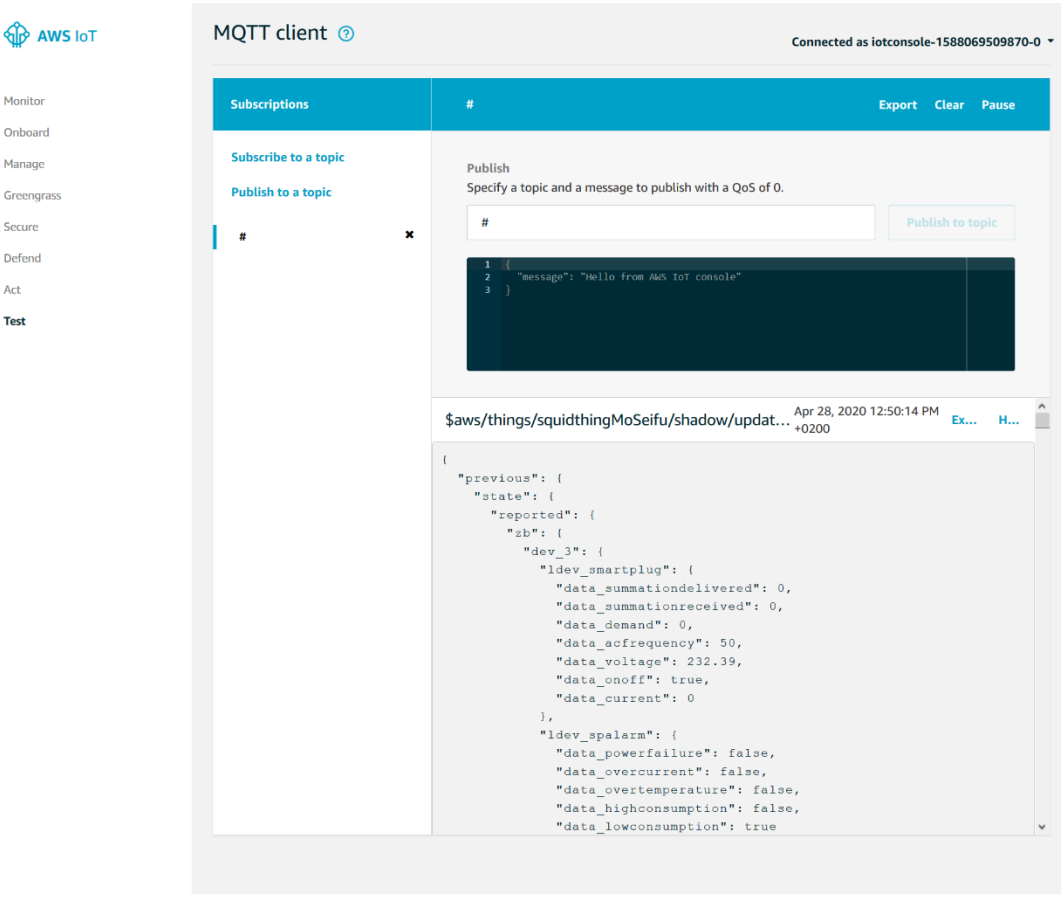
In this section we demonstrate how to view device MQTT messages with the AWS IoT MQTT client:

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



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/

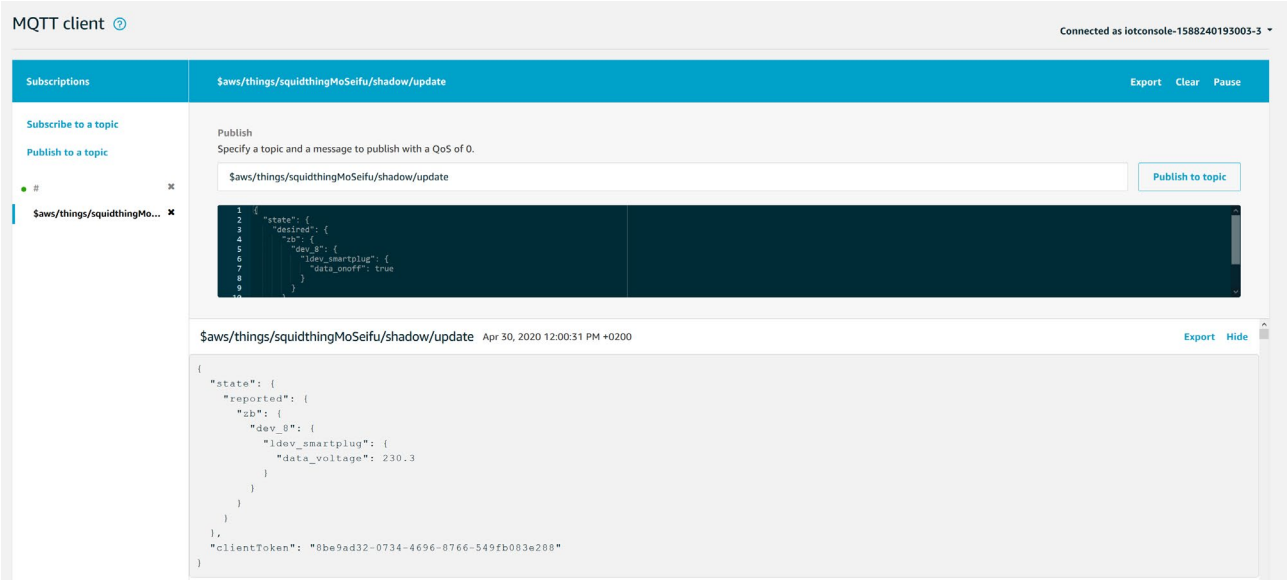
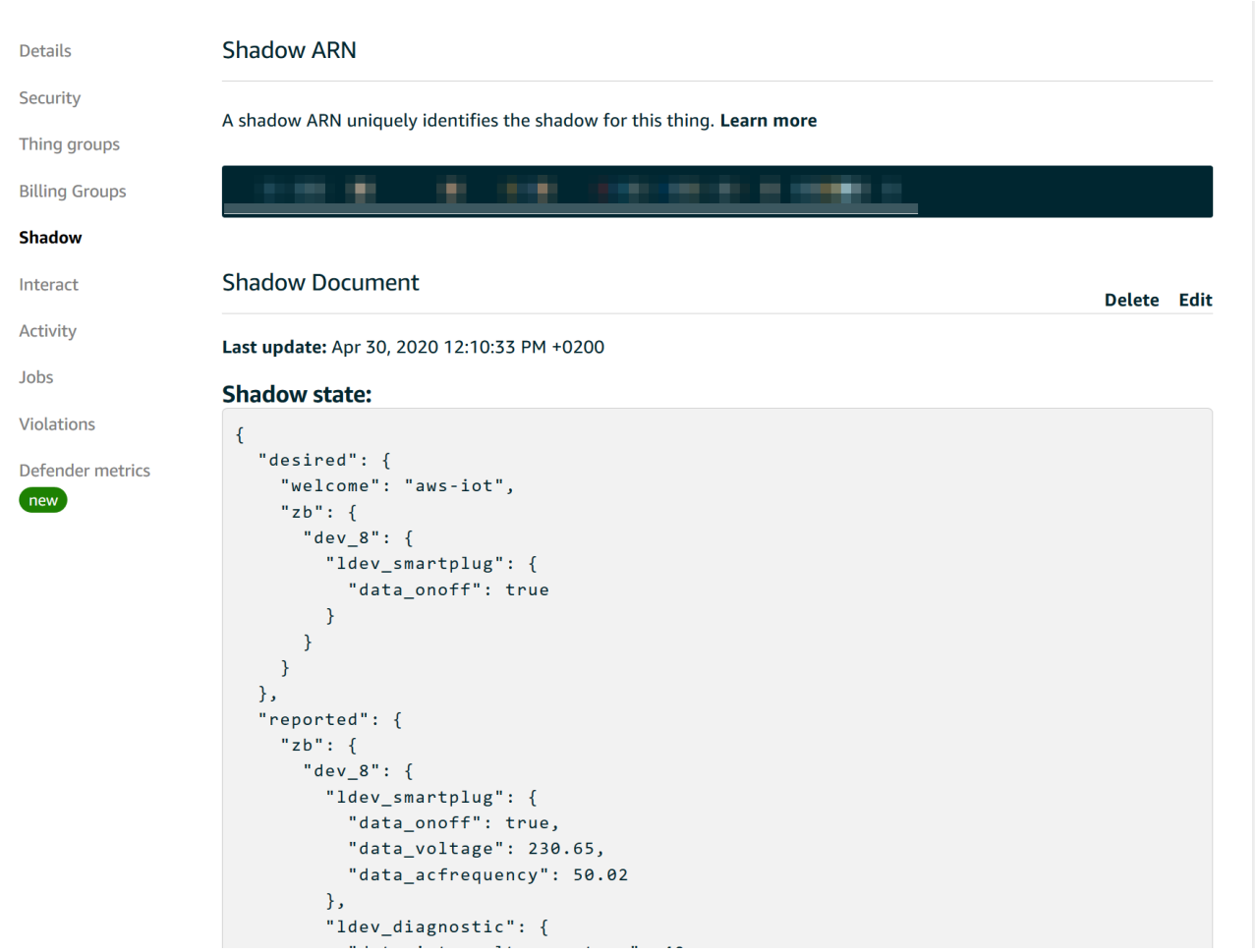


Figure 19 Publish to a topic

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
       }
     }
    }
  }
}
```



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..