External API description

January 2022

1 Introduction

This document describes the key principles of the integration API with the Sensolus platform.

Broadly speaking this API has three goals:

- Extract data from the platform. The data is split in different categories:
    - Time series data: the most low-level observations like coordinates, signal RSSI, matching geo-zones, …. A timeseries element Is fundamentally an entity with a timestamp, a metric type and a value.
    - Aggregated data: here an observation gets grouped in a more complete package. E.g. a location observation groups the coordinates, the source of the coordinates, the trigger for the measurements
    - Alerts
- Inject data into the platform
    - Custom metrics can be inserted in the platform and later used for querying
- Perform administrative actions:
    - Configure tracker with name, tags, third-party ID, Image
    - Define alert rules
    - Define geozones
    - Define geobeacons
    - Define webhooks

With respect to aggregated data there are 2 broad mechanisms to extract the data from the platform:

- Pull: the client needs to periodically poll the system and check what is new and retrieve that data. The protocol being used is REST with JSON encoded data.

Push: the Sensolus platform will push the data as soon as it is available towards the client. REST based push and MQTT is supported. Typically, the push method is preferred if the goal of the external party is to remain completely in sync with the data in the Sensolus platform. It also avoids hitting API quota limits. The pull mechanism can still be used to do an initial sync or to support exceptional querying cases.

2 A few examples

2.1 Get the trackers

Let's start with an example to retrieve the list of trackers:

Request:

```
GET {{SERVER}}/rest/api/v2/devices/{{SERIAL}}?apiKey={{API_KEY}}
```

Response:

```
{
    "name": "Trailer BE-5124",
    "serial": "CWDQHT",
    "sigfoxContractInfo": {
        "contractType": "CONTRACT",
        "activatedAt": "2019-06-04T07:10:38+0000",
        "endsAt": "2020-06-04T07:10:38+0000"
    },
    "status": "ONLINE",
    "sigfoxActivationStatus": "ACTIVATED",
    "batteryInfo": {
        "batteryLevelPercentage": 95,
        "estimatedRemainingBatteryLife": 35,
        "batteryEstimationCalibrated": true,
        "updatedAt": "2019-12-02T15:10:25+0000"
    },
    "firstMessageAt": "2018-11-15T20:20:52+0000",
    "lastSeenAlive": "2019-12-03T07:08:50+0000",
    "productKey": "SNT2 PRO GPS B2 4.2",
    "hwRevisionKey": "SNT2 rev B2 - connect",
    "deviceTags": [],
    "image": {},
    "productName": "SNT2 PRO GPS B2 4.2"
}
```

We can see the following elements in this example:

- Authentication happens with an API key passed as a query parameter
- Responses are always JSON encoded
- Timestamps are ISO-8601 encoded. That will go for all timestamps used in requests and responses

2.2 Retrieve time series info

To know what time series are available, use following API call:

Request:

```
GET {{SERVER}}/rest/api/v2/devices/{{SERIAL}}/
timeseries/types?apiKey={{API_KEY}}
```

Response

```
[
  {
    "key": "DEVICE_STATISTICS_COMM_TOTAL_NUMBER_DI_RETRANSMITS",
    "description": "This timeserie contains number of DI re-
transmits, messages sent from DIQ, not the first transmission from
MSGQ"
  },
  {
    "key": "NETWORK_BASED_LOCATION_SOURCE",
    "description": "The source of the network location. Possible
values are: sigfox_first_receiving_basestation_location |
sigfox_triangulation"
  },
  {
    "key": "SENSOR_DATA_GPS_LOCATION_ADDRESS",
    "description": "The GPS address derived from a GPS location."
  },
  {
    "key": "SENSOR_DATA_WIFI_SCAN_NUMBER_OF_MACS",
    "description": "This timeserie will contain the"
  },
  …
]
```

To get a time series:

```
GET {{SERVER}}/rest/api/v2/devices/{{SERIAL}}/ timeseries?
series={TIME SERIES}&timeFilter={byMessageTime/byInsertTime
}&from={DATE}&to={DATE}&apiKey={{API_KEY}}
```

2.3 Retrieve location info

As a final example let's retrieve the location history for a time window:

Request:

```
GET
{{SERVER}}/rest/api/v2/devices/{{SERIAL}}/data/aggregated/location?ap
iKey={{API_KEY}}&from=2019-09-22T00:00:00%2B0000&to=2019-09-
30T00:00:00%2B0000&timeSorting=DESC&timeFilter=byMessageTime
```

Response

```
[
    {
        "data": [
            {
                "state": "STOP",
                "lat": 51.05985,
                "lng": 3.69419,
                "accuracy": 20,
                "fixTime": 134,
                "source": "gps",
                "geozones": [
                    "all the world"
                ],
                "address": "[ Undecoded ]",
                "id": 100163695,
                "time": "2019-09-29T12:13:00+0000",
                "insertTime": "2019-09-29T12:20:40+0000"
            },
            …
        ],
        "truncated": false,
        "skipped": false,
        "serial": "CWDQHT"
    }
]
```

The request specifies the following:

- the tracker serial
- the type of aggregated data to retrieve, here we want location data
- the time window: from and to
- how we want the data

In this response we observe the following:

- An array of location records. We have not printed the full content of the array in this document.
- At the end 2 Boolean parameters:
    - truncated: there is a maximum number of records we return in a single call. If this flag is true you should reduce the size of the window
    - skipped: not relevant when retrieving data for a single tracker

2.4 Retrieve the active alerts

As a final example let us show how the active alerts can be retrieved:

Request:

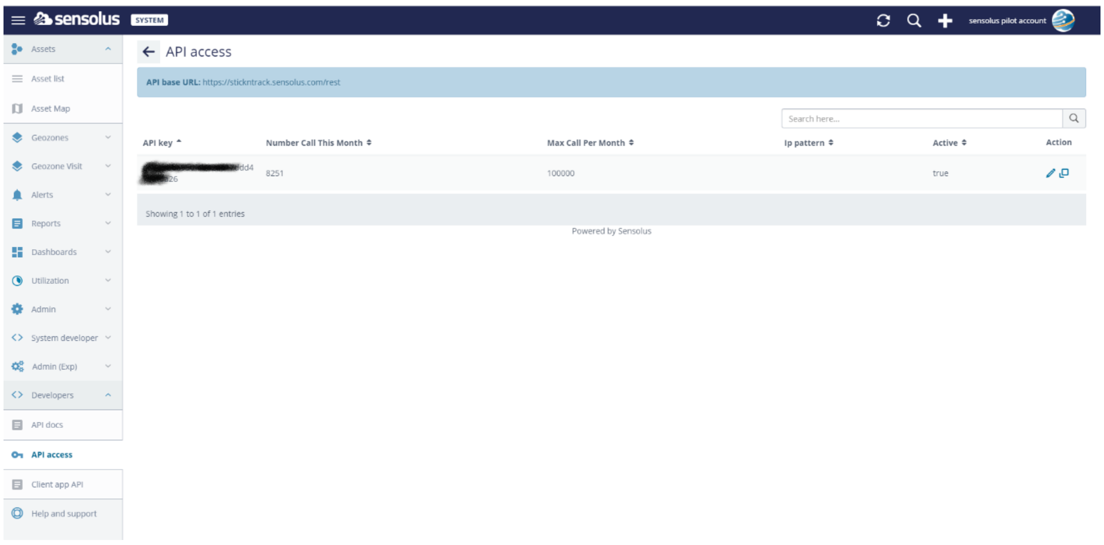
```
GET
{{SERVER}}/rest/api/v1/devices/{{SERIAL}}/alerts/active?apiKey={{API_
KEY}}
```

Response:

```
[
    {
        "date": "2019-09-11T06:06:00+0000",
        "alertType": "GeozoneOutsideAlertType",
        "title": "Phi Test ouside Geozone",
        "alertRule": {
            "id": 861,
            "title": "Phi Test ouside Geozone",
            "description": "Phi Test ouside Geozone",
            "active": true,
            "alertTypeName": "GeozoneOutsideAlertType",
            "definitions": {
                "selectedIds": [
                    696
                ]
            },
            "alertNotifications": [
                {
                    "emails": [],
                    "contacts": [
                        "8a8081f56a4e993b016a4ee2b3d80000"
                    ],
                    "notificationType": "EMAIL"
                }
            ],
            "severity": "REMINDER",
            "monitoredItem": {
                "selectedIds": [],
                "selectType": "ALL",
                "monitoredType": "DEVICE"
            }
        },
        "monitoredEntity": {
            "entityType": "DEVICE",
            "sigfoxDevice": {
                "serial": "CWDQHT",
                "name": "lrnc CWDQHT",
                "lastKnownLat": 51.05461,
                "lastKnownLng": 3.71665,
                "lastLocationUpdate": "2019-12-03T15:15:00+0000",
                "lastSeenAlive": "2019-12-03T15:20:26+0000",
                "gpsSignalAvailable": true,
                "lastMovementDetected": "2019-12-03T15:15:00+0000",
                "status": "ONLINE"
            }
        }
]
```

3 DEV API keys

The API key must be passed for every call to the platform. API calls have a maximum number of requests on a monthly basis and the current usage can be seen in the UI:



API access

API base URL: https://stickntrack.sensolus.com/rest

Search here...

API key	Number Call This Month	Max Call Per Month	Ip pattern	Active	Action
[redacted]dd4 [redacted]26	8251	100000		true	

Showing 1 to 1 of 1 entries

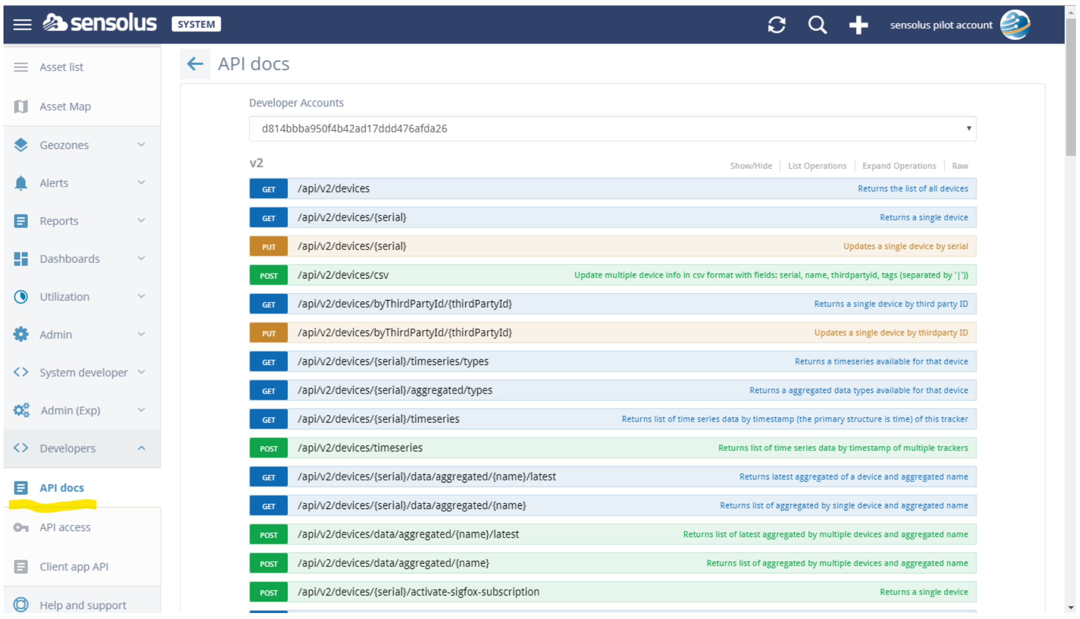
Powered by Sensolus

Everyone with the API key can make calls with it, so you are supposed to keep you're API key in a secure place.

It is possible to limit API calls to a specific subnet only to avoid quota theft.

4 API overview with Swagger

The best way to explore all the API's is to login to the platform and go to the section Developers → API docs:



API docs

Developer Accounts

d814bbba950f4b42ad17ddd476afda26

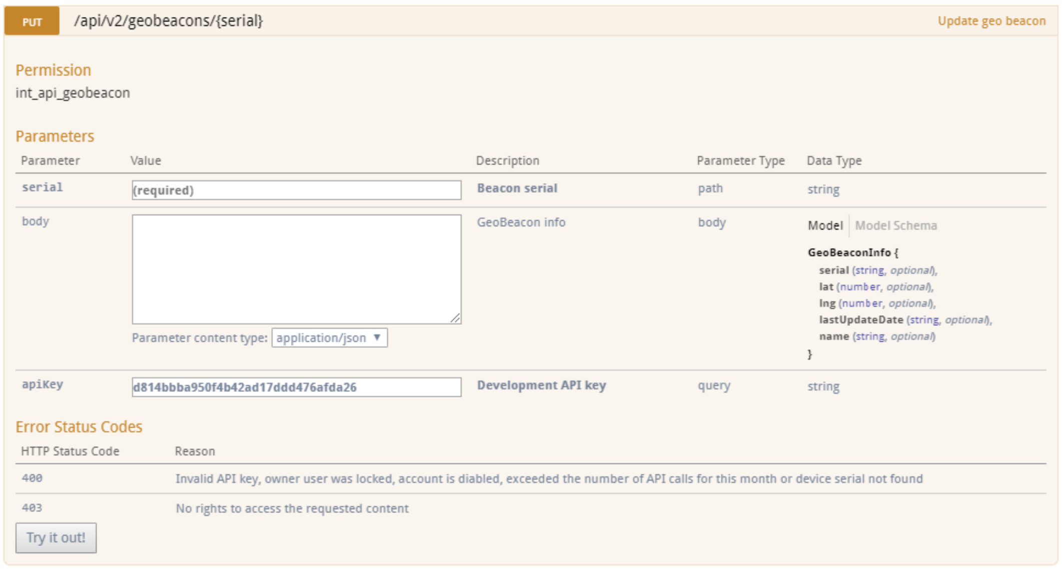
v2

Show/Hide | List Operations | Expand Operations | Raw

GET	/api/v2/devices	Returns the list of all devices
GET	/api/v2/devices/{serial}	Returns a single device
PUT	/api/v2/devices/{serial}	Updates a single device by serial
POST	/api/v2/devices/csv	Update multiple device info in csv format with fields: serial, name, thirdpartyid, tags (separated by '|'))
GET	/api/v2/devices/byThirdPartyId/{thirdPartyId}	Returns a single device by third party ID
PUT	/api/v2/devices/byThirdPartyId/{thirdPartyId}	Updates a single device by thirdparty ID
GET	/api/v2/devices/{serial}/timeseries/types	Returns a timeseries available for that device
GET	/api/v2/devices/{serial}/aggregated/types	Returns a aggregated data types available for that device
GET	/api/v2/devices/{serial}/timeseries	Returns list of time series data by timestamp (the primary structure is time) of this tracker
POST	/api/v2/devices/timeseries	Returns list of time series data by timestamp of multiple trackers
GET	/api/v2/devices/{serial}/data/aggregated/{name}/latest	Returns latest aggregated of a device and aggregated name
GET	/api/v2/devices/{serial}/data/aggregated/{name}	Returns list of aggregated by single device and aggregated name
POST	/api/v2/devices/data/aggregated/{name}/latest	Returns list of latest aggregated by multiple devices and aggregated name
POST	/api/v2/devices/data/aggregated/{name}	Returns list of aggregated by multiple devices and aggregated name
POST	/api/v2/devices/{serial}/activate-sigfox-subscription	Returns a single device

This shows a Swagger view where you can see all API's and experiment with them directly in the browser.

For example, let's look at the request to define a geo beacon:


PUT /api/v2/geobeacons/{serial} — Update geo beacon

Permission
int_api_geobeacon

Parameters

Parameter	Value	Description	Parameter Type	Data Type
serial	(required)	Beacon serial	path	string

body	Parameter content type: application/json	GeoBeacon info	body	Model | Model Schema
				GeoBeaconInfo {
				serial (string, optional),
				lat (number, optional),
				lng (number, optional),
				lastUpdateDate (string, optional),
				name (string, optional)
				}

apiKey	d814bbba950f4b42ad17ddd476afda26	Development API key	query	string

Error Status Codes

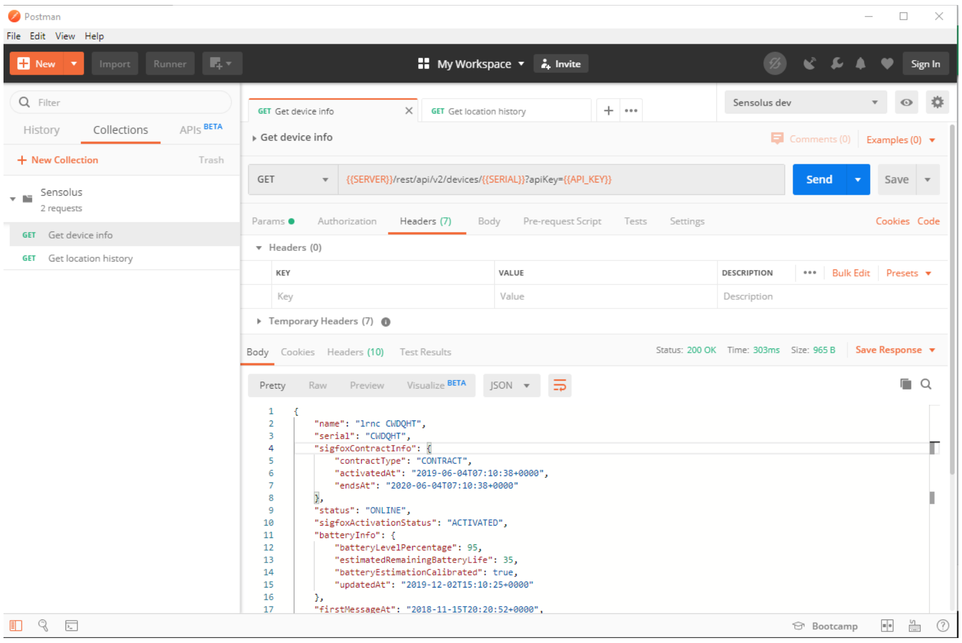
HTTP Status Code	Reason
400	Invalid API key, owner user was locked, account is diabled, exceeded the number of API calls for this month or device serial not found
403	No rights to access the requested content

Try it out!

It will show all the parameters and their expected structure. The goal of this document is not to give an exhaustive overview of every single API call. The API itself is best explored with Swagger.

5 Testing the REST API with Postman

Postman is a well-known tool to test REST clients. It allows you to construct quickly REST calls and see the results of their invocation:



GET Get device info

GET {{SERVER}}/rest/api/v2/devices/{{SERIAL}}?apiKey={{API_KEY}}

Headers (0)

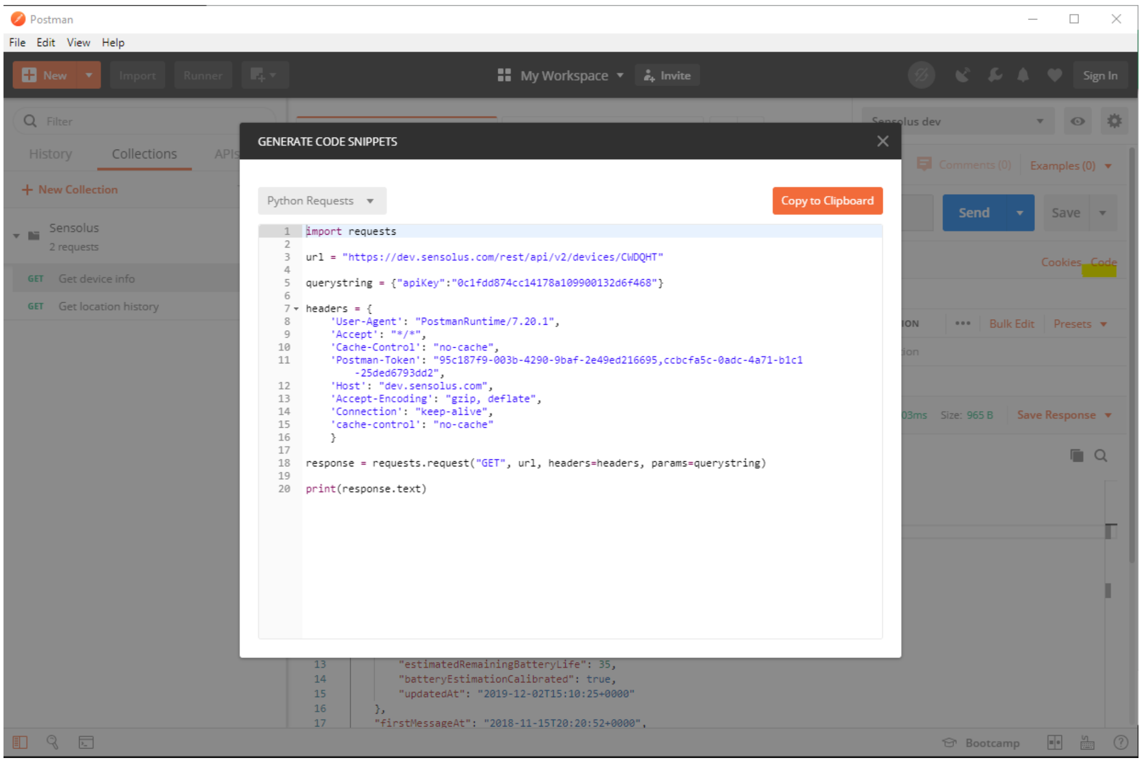
KEY	VALUE	DESCRIPTION
Key	Value	Description

Temporary Headers (7)

Body Cookies Headers (10) Test Results — Status: 200 OK Time: 303ms Size: 965 B

```
{
    "name": "lrnc CWDQHT",
    "serial": "CWDQHT",
    "sigfoxContractInfo": {
        "contractType": "CONTRACT",
        "activatedAt": "2019-06-04T07:10:38+0000",
        "endsAt": "2020-06-04T07:10:38+0000"
    },
    "status": "ONLINE",
    "sigfoxActivationStatus": "ACTIVATED",
    "batteryInfo": {
        "batteryLevelPercentage": 95,
        "estimatedRemainingBatteryLife": 35,
        "batteryEstimationCalibrated": true,
        "updatedAt": "2019-12-02T15:10:25+0000"
    },
    "firstMessageAt": "2018-11-15T20:20:52+0000",
```

One of the nice aspects of Postman is that you can generate client code for various programming languages.



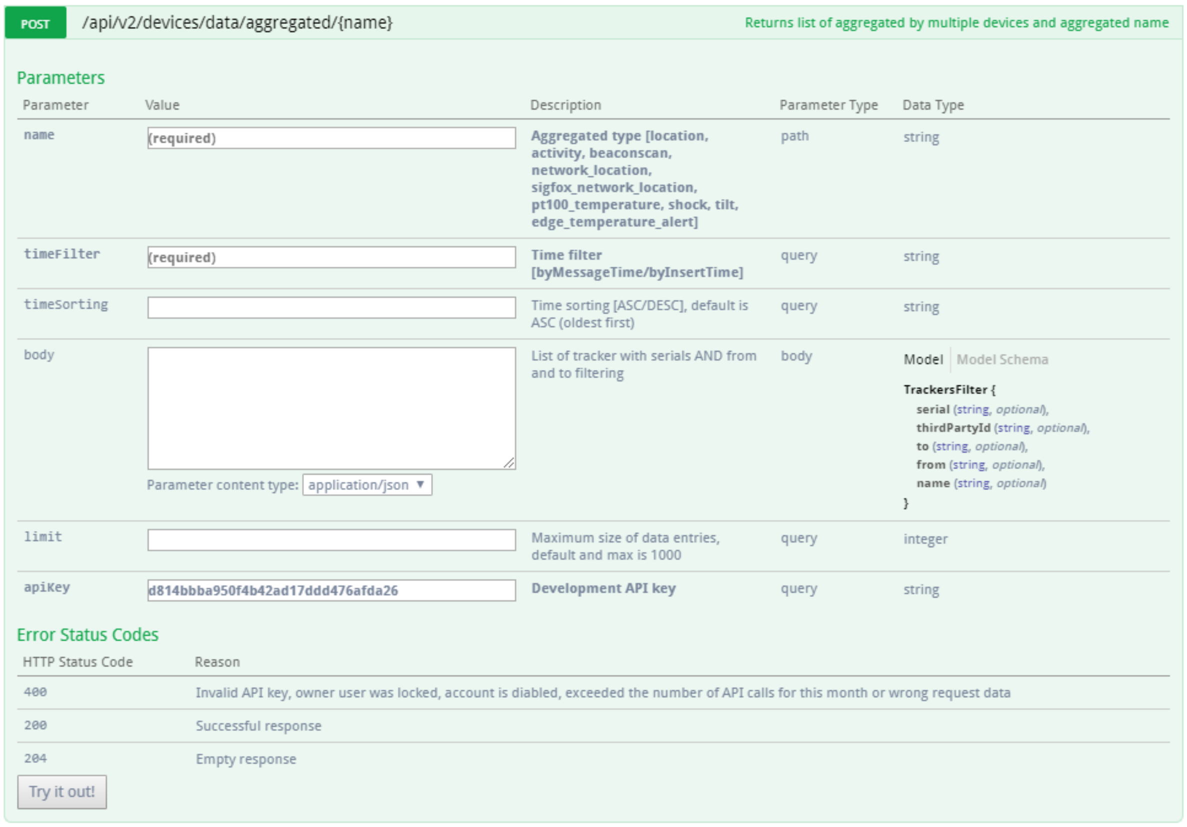
```python
import requests

url = "https://dev.sensolus.com/rest/api/v2/devices/CWDQHT"

querystring = {"apiKey":"0c1fdd874cc14178a109900132d6f468"}

headers = {
    'User-Agent': "PostmanRuntime/7.20.1",
    'Accept': "*/*",
    'Cache-Control': "no-cache",
    'Postman-Token': "95c187f9-003b-4290-9baf-2e49ed216695,ccbcfa5c-0adc-4a71-b1c1
        -25ded6793dd2",
    'Host': "dev.sensolus.com",
    'Accept-Encoding': "gzip, deflate",
    'Connection': "keep-alive",
    'cache-control': "no-cache"
    }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)
```

The Sensolus team publishes a Postman collection file which helps you to get started. Ask your Sensolus contact.

6 Strategy for syncing all data in pull mode

A very common use case is syncing all data across a large set of devices. The simple approach is to retrieve all data for all devices in a big loop by iterating over all devices, but this will quickly exhaust the API quota. However, there is a strategy to do it efficiently.

The solution is based on using the bulk retrieval call:



POST /api/v2/devices/data/aggregated/{name} — Returns list of aggregated by multiple devices and aggregated name

Parameters

Parameter	Value	Description	Parameter Type	Data Type
name	(required)	**Aggregated type [location, activity, beaconscan, network_location, sigfox_network_location, pt100_temperature, shock, tilt, edge_temperature_alert]**	path	string
timeFilter	(required)	**Time filter [byMessageTime/byInsertTime]**	query	string
timeSorting		Time sorting [ASC/DESC], default is ASC (oldest first)	query	string

body	Parameter content type: application/json	List of tracker with serials AND from and to filtering	body	Model | Model Schema
				TrackersFilter {
				serial (string, *optional*),
				thirdPartyId (string, *optional*),
				to (string, *optional*),
				from (string, *optional*),
				name (string, *optional*)
				}

limit		Maximum size of data entries, default and max is 1000	query	integer
apiKey	d814bbba950f4b42ad17ddd476afda26	**Development API key**	query	string

Error Status Codes

HTTP Status Code	Reason
400	Invalid API key, owner user was locked, account is diabled, exceeded the number of API calls for this month or wrong request data
200	Successful response
204	Empty response

The bulk API call allows to pass a set of serials in one shot. As you can see the maximum number of results across all trackers is 1000. That means depending on the number of trackers and the time window the likelihood of having truncated data is large. For the bulk call the response will indicate per serial whether the data is truncated and/or skipped. Truncated means there was more data, but it didn't fit in the response. Skipped means that because of other trackers we already exhausted the budget. In both cases you need to do a new call to retrieve the missing data.

In the request object we can see there is time window per tracker. By incrementally updating this window we can retrieve all data without requesting the same data twice. The following mechanism should be used:

- start with the desired same window for all trackers
- if the tracker data was not truncated and not skipped -> do not include in the next request, we are complete
- if the tracker data was truncated and not skipped: update the time window to exclude what you got already. The update should be done depending on the sort order of the data
- if the data was skipped -> repeat the call with the same time window

This sequence can be repeated until all data is retrieved.

There is one thing which should still discuss and that is the time filter attribute. It has two possible values: *byMessageTime* and byInsertTime. Data from the trackers does not always arrive in real time. As such there is a difference between the time of the message and the time it got inserted in the platform. If the goal is to retrieve all data one should query *byInsertTime*. If you have synced the data *byInsertTime* for a certain window you can be guaranteed there will be no new data for that window. That guarantee is not available if you query by message time. Up to 90 days later data can still be recovered.

7 Push mode

7.1 Principle

A more efficient way to sync all data can be to setup an endpoint to retrieve the data from Sensolus as soon as it arrives. We support two mechanisms for push mode

- HTTP endpoint: we will do a REST call, you can configure the URL and header fields
- MQTT: MQTT is a lightweight protocol that is optimized for networks with small bandwidth and high latency. It supports simple publish/subscribe semantics and is specifically designed with IoT devices in mind. It is an ideal solution for integration of the SNT platform with existing IoT architectures where SNT can act as a hub that reliably captures the Sigfox data, adds the device metadata and publishes them to specific topics based on certain parameters like tags

The data you will receive over both methods looks very similar to the data you retrieve over the pull REST API. A sample is given below:

```
{

"dataType": activity,
"data":
{
      "state": "STOP",
      "lat": 50.926003,
      "lng": 4.0446835,
      "accuracy": 4468,
      "source": "network",
      "id": 798343,
      "time": "2018-07-31T08:47:00+0000",
      "insertTime": "2018-07-31T08:53:48+0000",
      "serial": "RJH963",
      "thirdPartyId": "ABC",
}
```

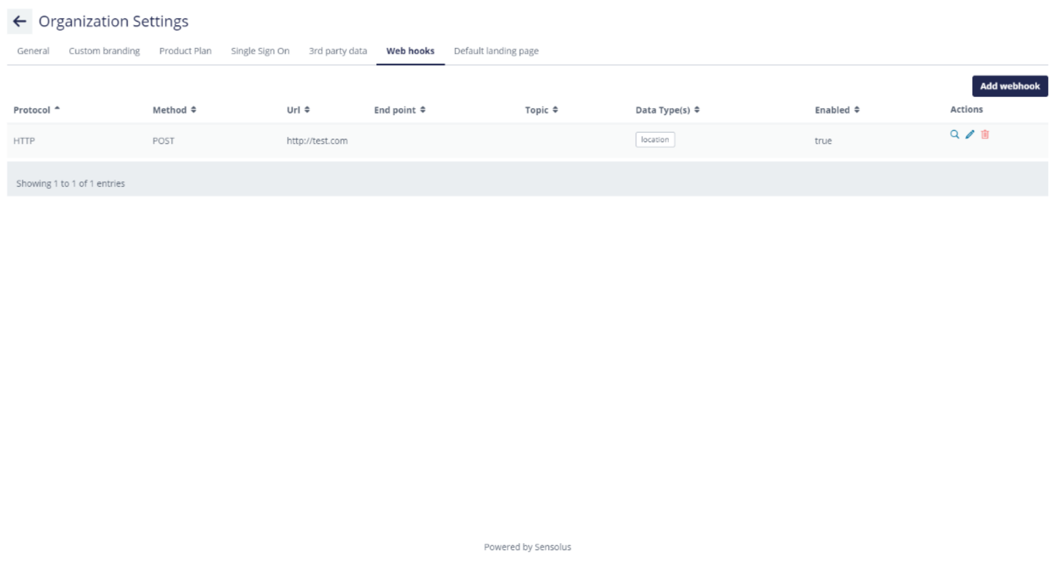
The only difference is that every data item contains 2 extra fields:

- serial
- third party id

We expect the endpoint to return 200 OK. The body will be ignored.

7.2 Configuring webhooks

In the platform push mode is called Webhooks and they can be configured through the UI. Go to the Admin -> Organizations. The Webhook tab will be visible there


Organization Settings

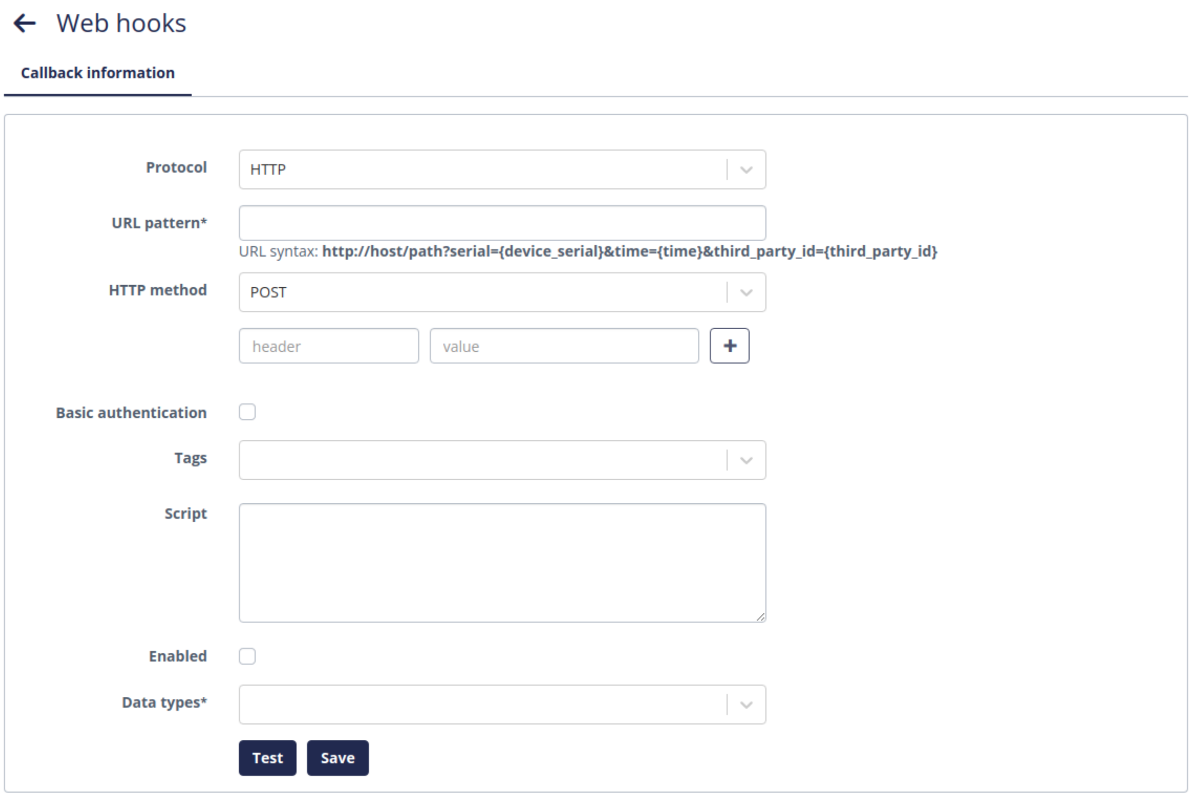
Protocol	Method	Url	End point	Topic	Data Type(s)	Enabled	Actions
HTTP	POST	http://test.com			location	true	

Showing 1 to 1 of 1 entries

Powered by Sensolus

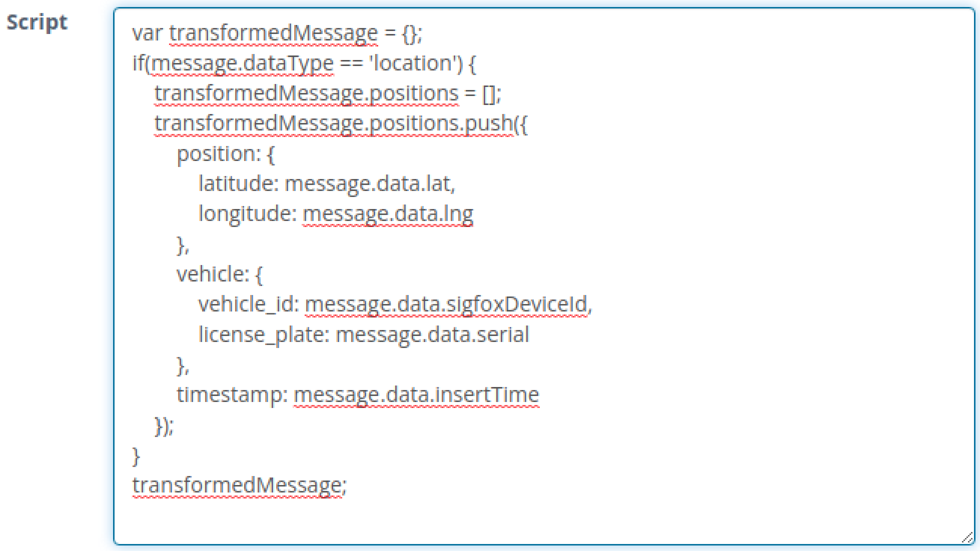
Click add webhook.

For HTTP we have the following configuration:


Web hooks

Protocol	HTTP
URL pattern*	
	URL syntax: **http://host/path?serial={device_serial}&time={time}&third_party_id={third_party_id}**
HTTP method	POST
	header / value
Basic authentication	☐
Tags	
Script	
Enabled	☐
Data types*	

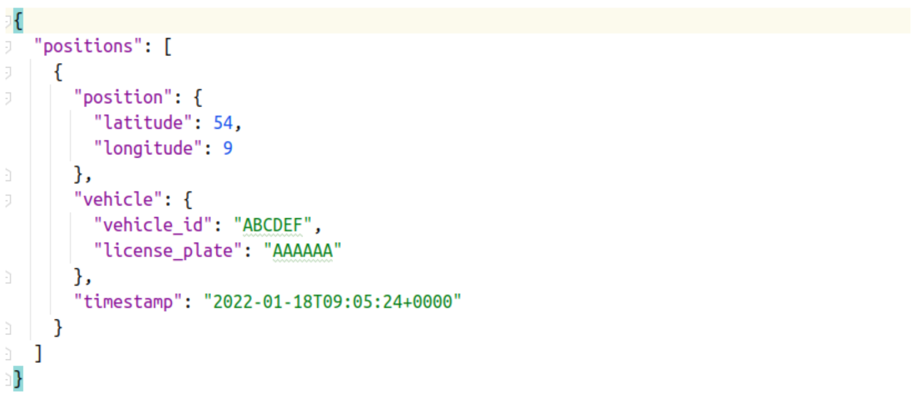
- URL pattern: REST callback URL. May contain placeholders for some parameters like serial, third party id and time (Unix time)
- HTTP method: PUT or POST
- HTTP headers: this can be used to add values like authentication headers
- Enabled: enable or disable webhook
- Data types: select one more of activity, location, pt100_temperature
- Tags: a list of device tags. If not empty, only messages for tagged devices will be sent
- Script: an (optional) transformation script to transform the original message payload in a new payload. This is useful when the webhook server expects a certain predefined format.
  The script should be written in javascript and should evaluate to the expected new format. There is 1 input variable that can be used: *message*. The result of your script is the same as what you would get by calling the javascript eval() method on your script. It will evaluate to the last entered statement in your script. As an example:



Script

```
var transformedMessage = {};
if(message.dataType == 'location') {
  transformedMessage.positions = [];
  transformedMessage.positions.push({
    position: {
      latitude: message.data.lat,
      longitude: message.data.lng
    },
    vehicle: {
      vehicle_id: message.data.sigfoxDeviceId,
      license_plate: message.data.serial
    },
    timestamp: message.data.insertTime
  });
}
transformedMessage;
```

In this case *transformedMessage* is calculated and repeated as a statement on the last line. The result may look something like:

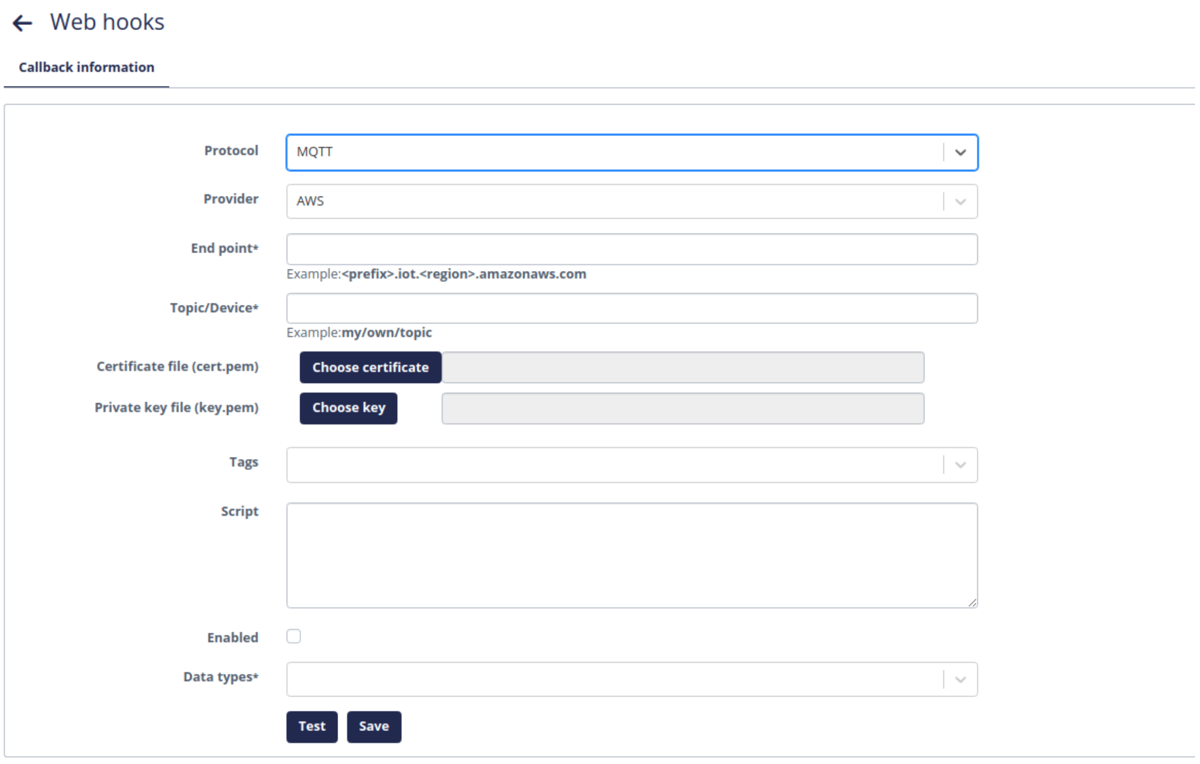


```
{
  "positions": [
    {
      "position": {
        "latitude": 54,
        "longitude": 9
      },
      "vehicle": {
        "vehicle_id": "ABCDEF",
        "license_plate": "AAAAAA"
      },
      "timestamp": "2022-01-18T09:05:24+0000"
    }
  ]
}
```

- Test button: test correct behavior, will send a demo message and show successful response code 200 OK

Save button: save the REST push webhook -> webhook is added to the table (see next)

For MQTT the configuration window looks like:


← Web hooks

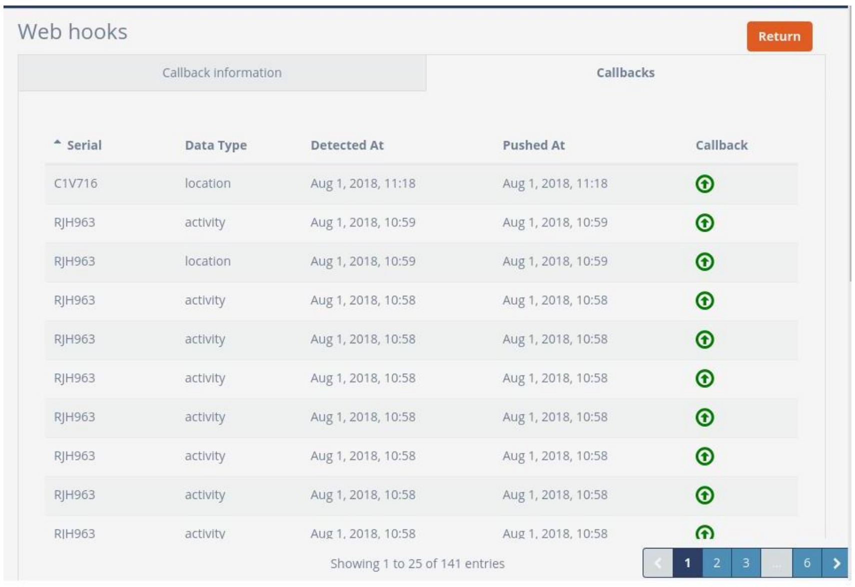
Callback information

Protocol	MQTT
Provider	AWS
End point*	Example:<prefix>.iot.<region>.amazonaws.com
Topic/Device*	Example:my/own/topic
Certificate file (cert.pem)	Choose certificate
Private key file (key.pem)	Choose key
Tags	
Script	
Enabled	☐
Data types*	

- Provider: the IoT hub provider. We currently support Amazon AWS and Microsoft Azure
- End point: host name of the end point. A typical example for AWS IoT core service is given
- Topic/device: name of the topic (AWS) or device (Azure) to which messages should be published
- Certificate file (cert.pem): the certificate file for the mutual SSH authentication
- Private key file (key.pem): the private key file for the mutual SSH authentication
- Enabled: enable or disable webhook
- Data types: select one or more of activity, location, pt100_temperature
- Tags: a list of device tags. If not empty, only messages for tagged devices will be sent
- Script: an (optional) transformation script (see HTTP case above for an example)
- Test button: test correct behavior, will send a demo message and show successful response code 200 OK

Save button: save the REST push webhook -> webhook is added to the table (see next)

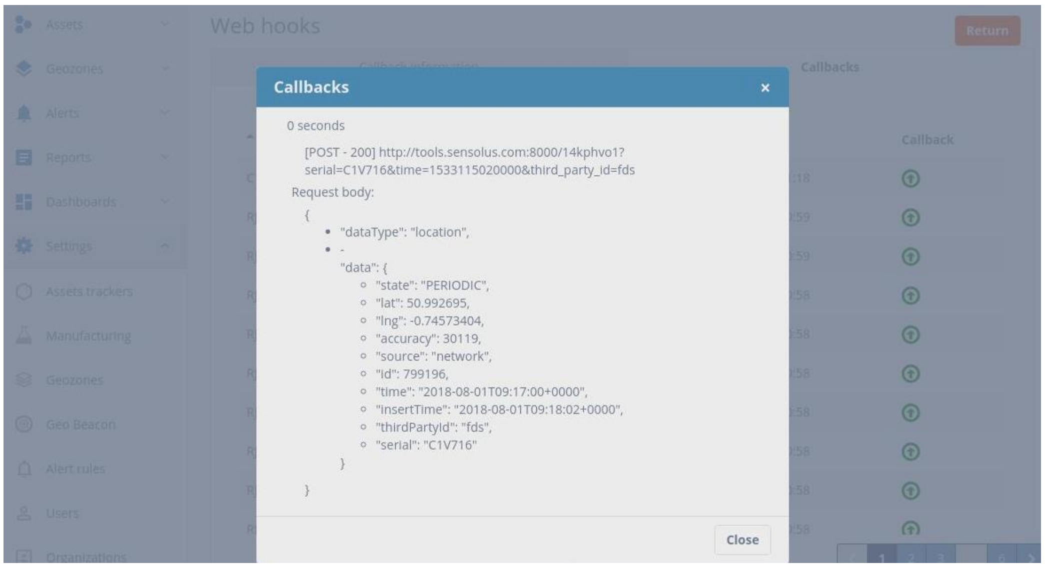
To troubleshoot callbacks the most recent callbacks are visualized in the callbacks tab:



Serial	Data Type	Detected At	Pushed At	Callback
C1V716	location	Aug 1, 2018, 11:18	Aug 1, 2018, 11:18	⊕
RJH963	activity	Aug 1, 2018, 10:59	Aug 1, 2018, 10:59	⊕
RJH963	location	Aug 1, 2018, 10:59	Aug 1, 2018, 10:59	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕
RJH963	activity	Aug 1, 2018, 10:58	Aug 1, 2018, 10:58	⊕

Showing 1 to 25 of 141 entries

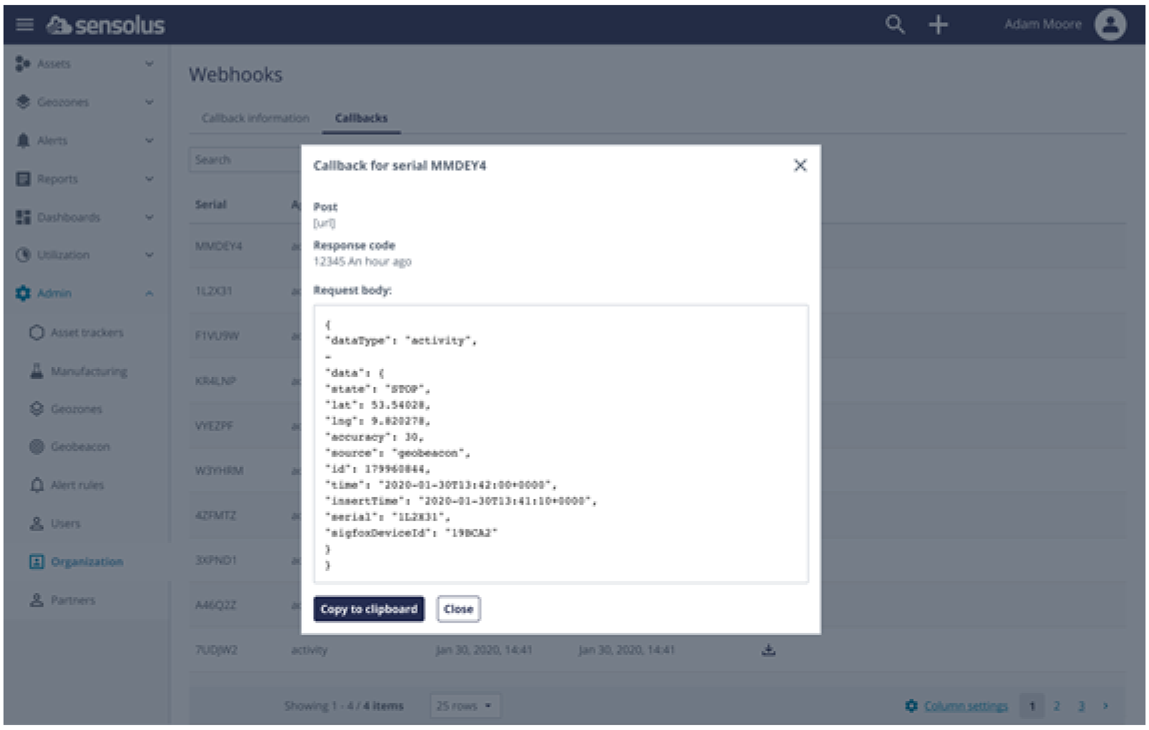
A click on the icon shows the detail of the callback:



```
0 seconds
  [POST - 200] http://tools.sensolus.com:8000/14kphvo1?
  serial=C1V716&time=1533115020000&third_party_id=fds
 Request body:
  {
     • "dataType": "location",
     • -
       "data": {
          ○ "state": "PERIODIC",
          ○ "lat": 50.992695,
          ○ "lng": -0.74573404,
          ○ "accuracy": 30119,
          ○ "source": "network",
          ○ "id": 799196,
          ○ "time": "2018-08-01T09:17:00+0000",
          ○ "insertTime": "2018-08-01T09:18:02+0000",
          ○ "thirdPartyId": "fds",
          ○ "serial": "C1V716"
       }
  }
```

7.3 Webhook retries

When we try to connect to an external system via Webhooks and it doesn't reply 3 times in a row within 30 seconds the system is blacklisted. This is to keep our handling sane because timeouts keep threads busy. With the webhook retries we redo the failed webhooks one hour later once the blacklist has been lifted.


Callback for serial MMDEY4

Post
[url]

Response code
12345 An hour ago

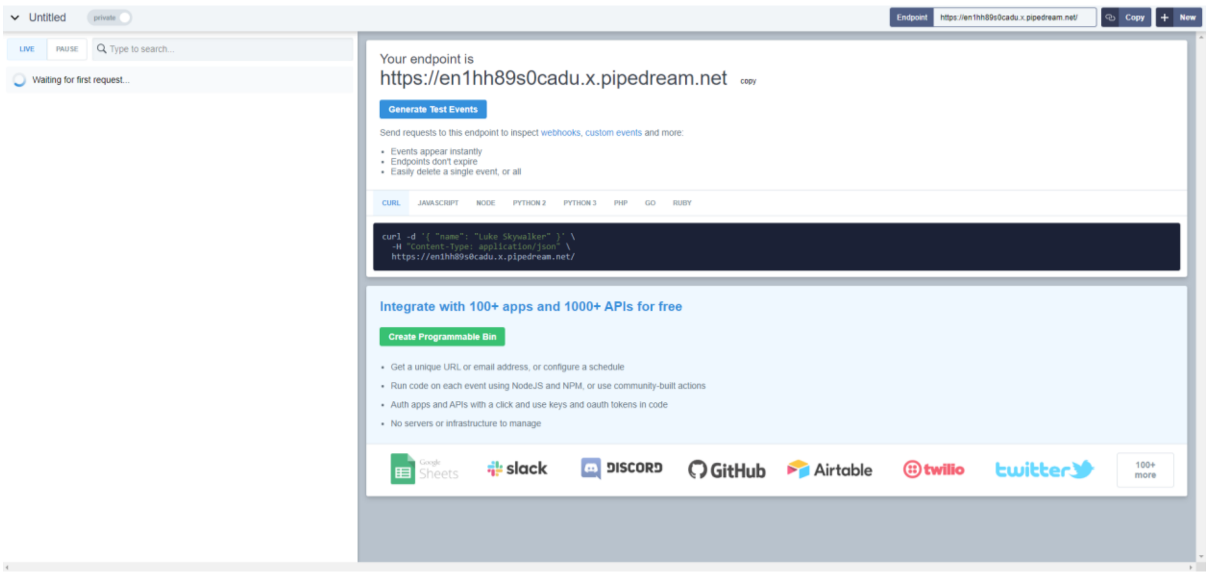
Request body:

```
{
"dataType": "activity",
-
"data": {
"state": "STOP",
"lat": 53.54028,
"lng": 9.820278,
"accuracy": 30,
"source": "geobeacon",
"id": 179960844,
"time": "2020-01-30T13:42:00+0000",
"insertTime": "2020-01-30T13:41:10+0000",
"serial": "1L2X31",
"sigfoxDeviceId": "19BCA2"
}
}
```

7.4 Debugging REST webhooks with RequestBin

The fastest way to see the REST webhooks in action is to use RequestBin (https://requestbin.com) to create a web endpoint which will login all calls.

The RequestBin website will show the endpoint:


Your endpoint is
https://en1hh89s0cadu.x.pipedream.net

Send requests to this endpoint to inspect webhooks, custom events and more:

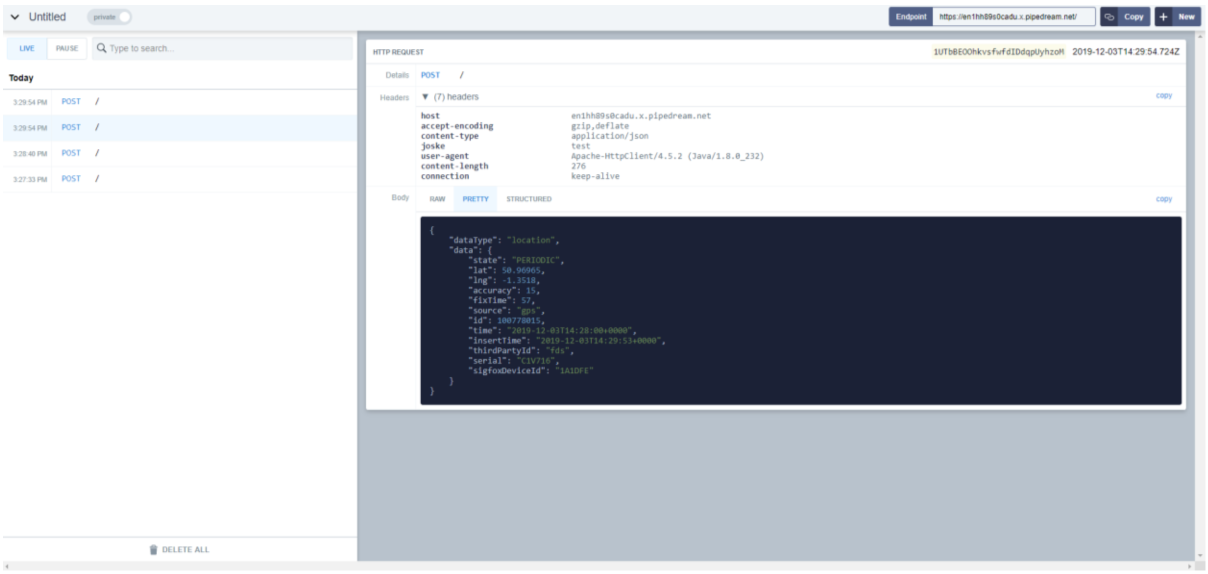
- Events appear instantly
- Endpoints don't expire
- Easily delete a single event, or all

```
curl -d '{ "name": "Luke Skywalker" }' \
  -H "Content-Type: application/json" \
  https://en1hh89s0cadu.x.pipedream.net/
```

Integrate with 100+ apps and 1000+ APIs for free

- Get a unique URL or email address, or configure a schedule
- Run code on each event using NodeJS and NPM, or use community-built actions
- Auth apps and APIs with a click and use keys and oauth tokens in code
- No servers or infrastructure to manage

For every call which arrives a new entry will appear in the list on the left-hand side. Select it to see the details of the call back in the master pane:



Endpoint https://en1hh89s0cadu.x.pipedream.net/

HTTP REQUEST 1UTbBEOOhkvsfwfdIDdqpUyhzoM 2019-12-03T14:29:54.724Z

Details POST /

Headers ▼ (7) headers

```
host                  en1hh89s0cadu.x.pipedream.net
accept-encoding       gzip,deflate
content-type          application/json
joske                 test
user-agent            Apache-HttpClient/4.5.2 (Java/1.8.0_232)
content-length        276
connection            keep-alive
```

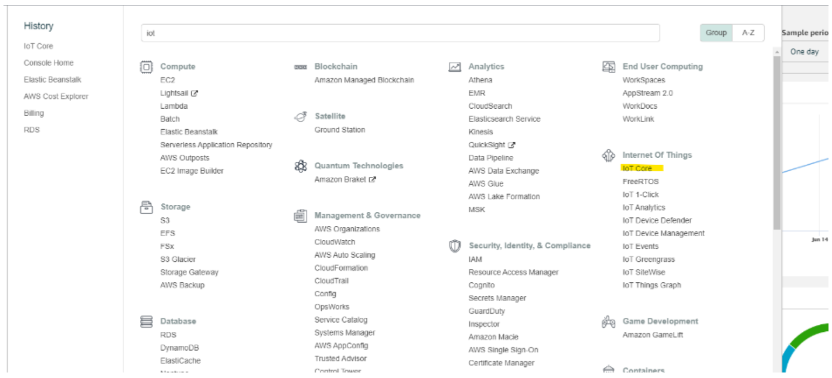
```
{
    "dataType": "location",
    "data": {
        "state": "PERIODIC",
        "lat": 50.96965,
        "lng": -1.3518,
        "accuracy": 15,
        "fixTime": 57,
        "source": "gps",
        "id": 100778015,
        "time": "2019-12-03T14:28:00+0000",
        "insertTime": "2019-12-03T14:29:53+0000",
        "thirdPartyId": "fds",
        "serial": "C1V716",
        "sigfoxDeviceId": "1A1DFE"
    }
}
```

7.5 Testing MQTT webhooks with Amazon MQTT

AWS provides a good environment to test and validate the MQTT functionality. This section describes step by step how to get this configured.

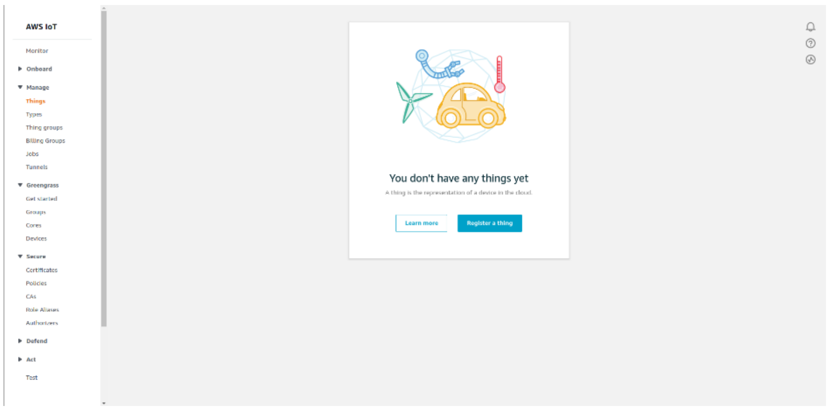
IOT service

Go to the IOT core service

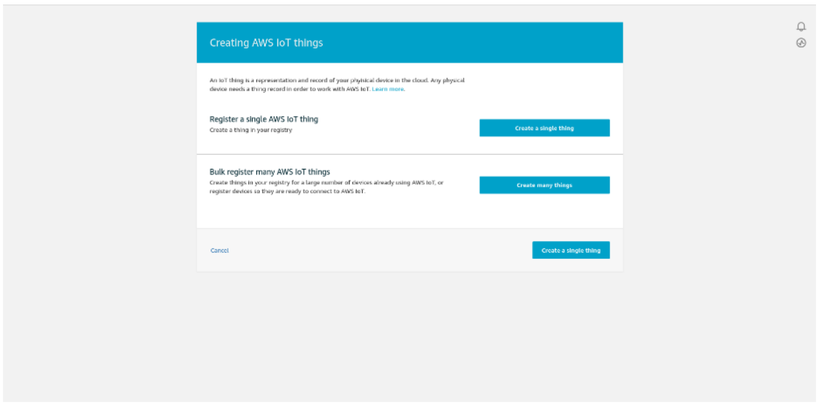


Thing creation

Next go to Manage -> Things. Click on 'register a thing'



Click on 'create a simple thing'



Creating AWS IoT things

An IoT thing is a representation and record of your physical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. Learn more.

Register a single AWS IoT thing
Create a thing in your registry

Create a single thing

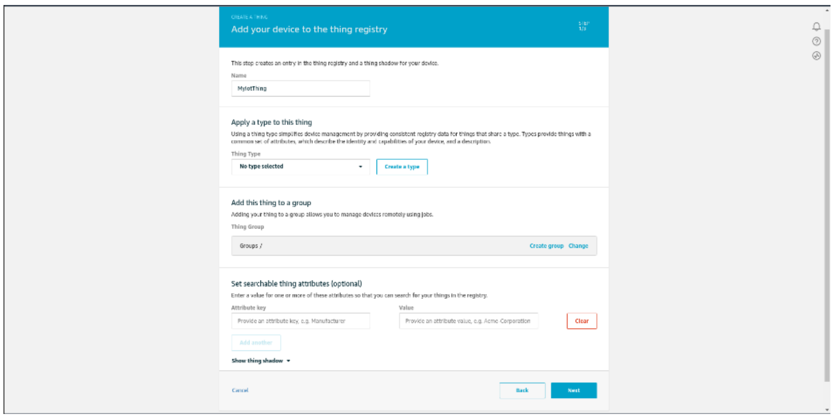
Bulk register many AWS IoT things
Create things in your registry for a large number of devices already using AWS IoT, or register devices so they are ready to connect to AWS IoT.

Create many things

Cancel

Create a single thing

Give a name and go to next



CREATE A THING
Add your device to the thing registry
STEP 1/3

This step creates an entry in the thing registry and a thing shadow for your device.

Name
MyIotThing

Apply a type to this thing
Using a thing type simplifies device management by providing consistent registry data for things that share a type. Types provide things with a common set of attributes, which describe the identity and capabilities of your device, and a description.

Thing Type
No type selected
Create a type

Add this thing to a group
Adding your thing to a group allows you to manage devices remotely using jobs.

Thing Group
Groups /
Create group Change

Set searchable thing attributes (optional)
Enter a value for one or more of these attributes so that you can search for your things in the registry.

Attribute key
Provide an attribute key, e.g. Manufacturer

Value
Provide an attribute value, e.g. Acme-Corporation

Clear

Add another

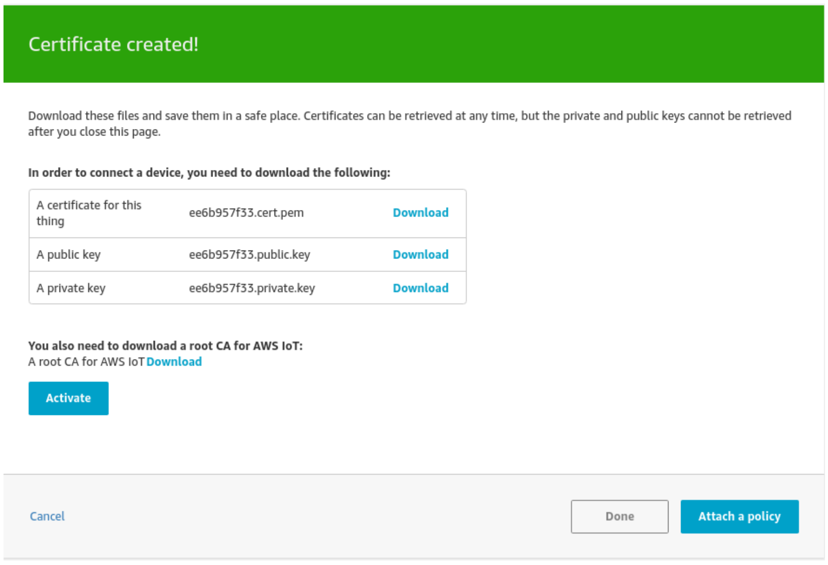
Show thing shadow

Cancel Back Next

Click on the top button to create a certificate with a simple click.

Now you will see a new screen where certificates can be downloaded. We will need the following files:

- the certificate (1st link)
- the private key (3rd link)


Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

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

A certificate for this thing	ee6b957f33.cert.pem	Download
A public key	ee6b957f33.public.key	Download
A private key	ee6b957f33.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT Download

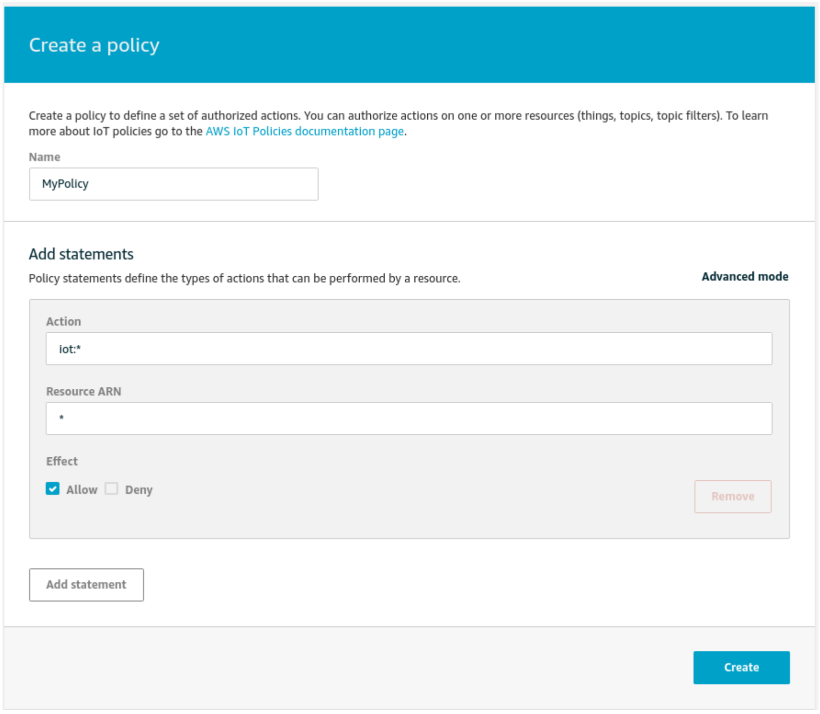
Now click done.

Policy creation

Next, we need to create a policy. X.509 certificates are used to authenticate your device with AWS IoT Core. AWS IoT Core policies are used to authorize your device to perform AWS IoT Core operations, such as subscribing or publishing to MQTT topics. Your device presents its certificate when sending messages to AWS IoT Core. To allow your device to perform AWS IoT Core operations, you must create an AWS IoT Core policy and attach it to your device certificate.

In the left navigation pane, choose Secure, and then choose Policies. Click create a policy.

On the Create a policy page, in the Name field, enter a name for the policy (for example, MyPolicy). Do not use personally identifiable information in your policy names.


Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name: MyPolicy

Add statements

Policy statements define the types of actions that can be performed by a resource.

Action: iot:*

Resource ARN: *

Effect: [✓ Allow] Deny

In the Action field, enter 'iot:*'. In the Resource ARN field, enter *. Select the Allow check box. This allows all clients to connect to AWS IoT Core.

You can restrict which clients (devices) can connect by specifying a client ARN as the resource. The client ARNs follow this format:

```
arn:aws:iot:your-region:your-aws-account:client/<my-client-id>
```

Choose the Add Statement button to add another policy statement. In the Action field, enter iot:*. In the Resource ARN field, enter the ARN of the topic to which your device publishes.

The topic ARN follows this format:

```
arn:aws:iot:your-region:your-aws-account:topic/<your/topic>
```

For example:

```
arn:aws:iot:us-east-1:123456789012:topic/my/topic
```

Finally, select the Allow check box. This allows your device to publish messages to the specified topic.

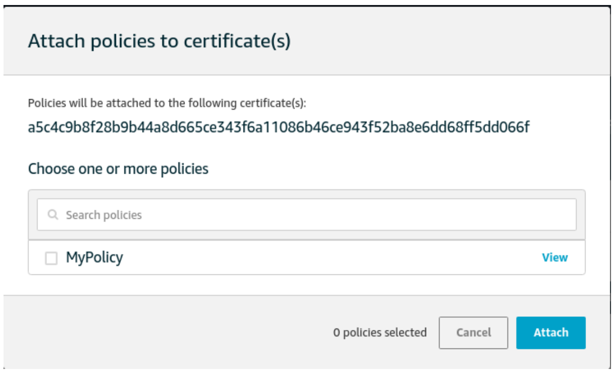
After you have entered the information for your policy, choose Create.

Attach an AWS IoT Core policy to a device certificate

Now that you have created a policy, you must attach it to your device certificate. Attaching an AWS IoT Core policy to a certificate gives the device the permissions specified in the policy.

In the left navigation pane, choose Secure, and then choose Certificates.

In the box for the certificate you created, choose ... to open a drop-down menu, and then choose Attach policy.

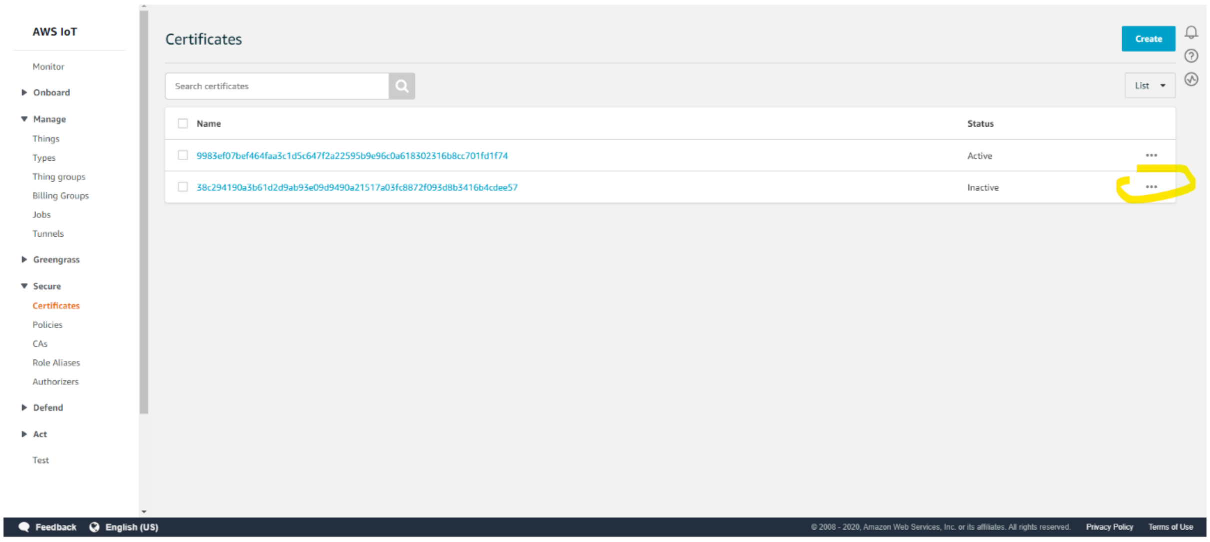


Attach policies to certificate(s)

Policies will be attached to the following certificate(s):
a5c4c9b8f28b9b44a8d665ce343f6a11086b46ce943f52ba8e6dd68ff5dd066f

Choose one or more policies

Finally, also activate the certificate. Select the certificate from the list, click on the 3 dots to get the menu and choose the active action.



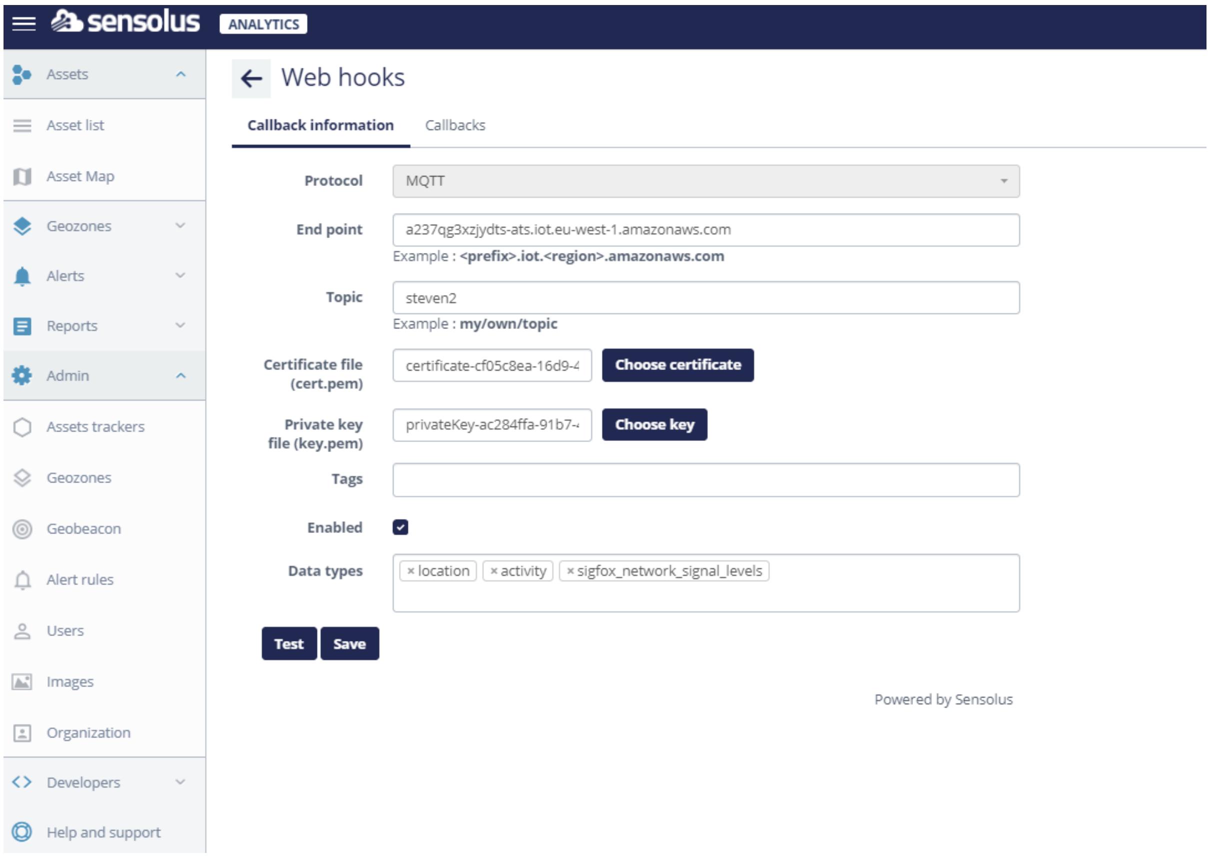
Certificates

Name	Status
9983ef07bef464faa3c1d5c647f2a22595b9e96c0a618302316b8cc701fd1f74	Active
38c294190a3b61d2d9ab93e09d9490a21517a03fc8872f093d8b3416b4cdee57	Inactive

This concludes everything which needs to be done on the AWS side.

Sensolus MQTT configuration

Next fill in the MQTT settings in the Sensolus Web application:



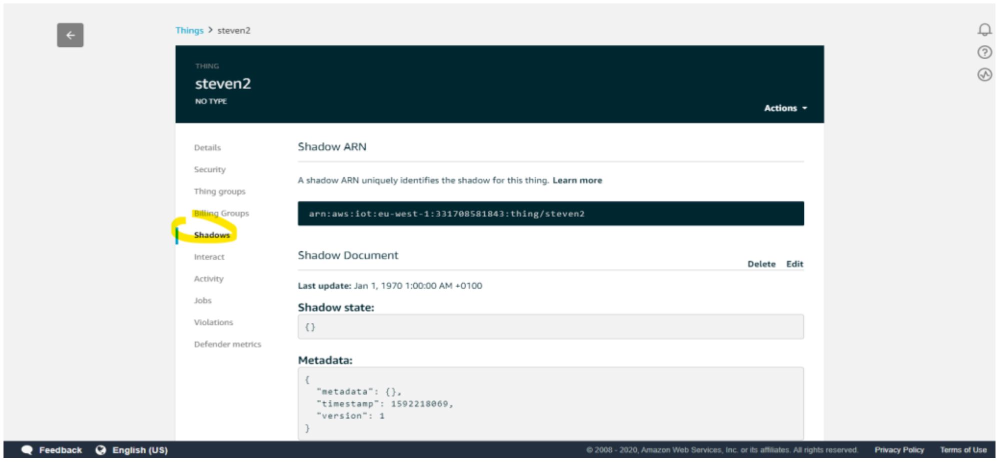
Web hooks

Callback information | Callbacks

Protocol	MQTT
End point	a237qg3xzjydts-ats.iot.eu-west-1.amazonaws.com Example : **<prefix>.iot.<region>.amazonaws.com**
Topic	steven2 Example : **my/own/topic**
Certificate file (cert.pem)	certificate-cf05c8ea-16d9-4 **Choose certificate**
Private key file (key.pem)	privateKey-ac284ffa-91b7- **Choose key**
Tags	
Enabled	✓
Data types	× location × activity × sigfox_network_signal_levels

A few fields need some explanation:

- endpoint: this value can be found in the AWS console under Things -> Shadows:



Shadow ARN

A shadow ARN uniquely identifies the shadow for this thing. **Learn more**

```
arn:aws:iot:eu-west-1:331708581843:thing/steven2
```

Shadow Document

Last update: Jan 1, 1970 1:00:00 AM +0100

Shadow state:

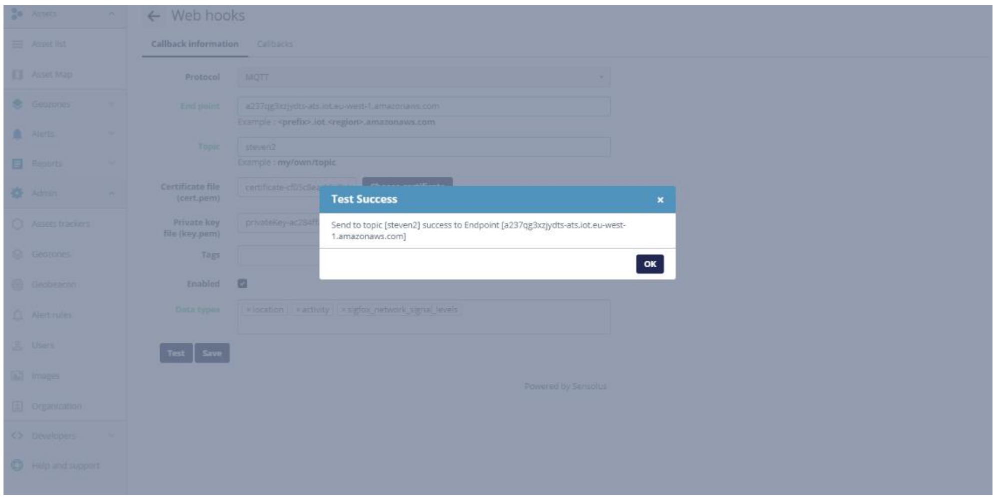
```
{}
```

Metadata:

```
{
  "metadata": {},
  "timestamp": 1592218069,
  "version": 1
}
```

- private and public key: these files where downloaded when the certificate was created.

In the Sensolus application the 'Test' button will now send a dummy message to the topic. If all configuration was done correctly this should give a success message:



Web hooks

Callback information | Callbacks

Protocol: MQTT

End point: a237qg3xzjydts-ats.iot.eu-west-1.amazonaws.com

Example : <prefix>.iot.<region>.amazonaws.com

Topic: steven2

Example : my/own/topic

Enabled: ✓

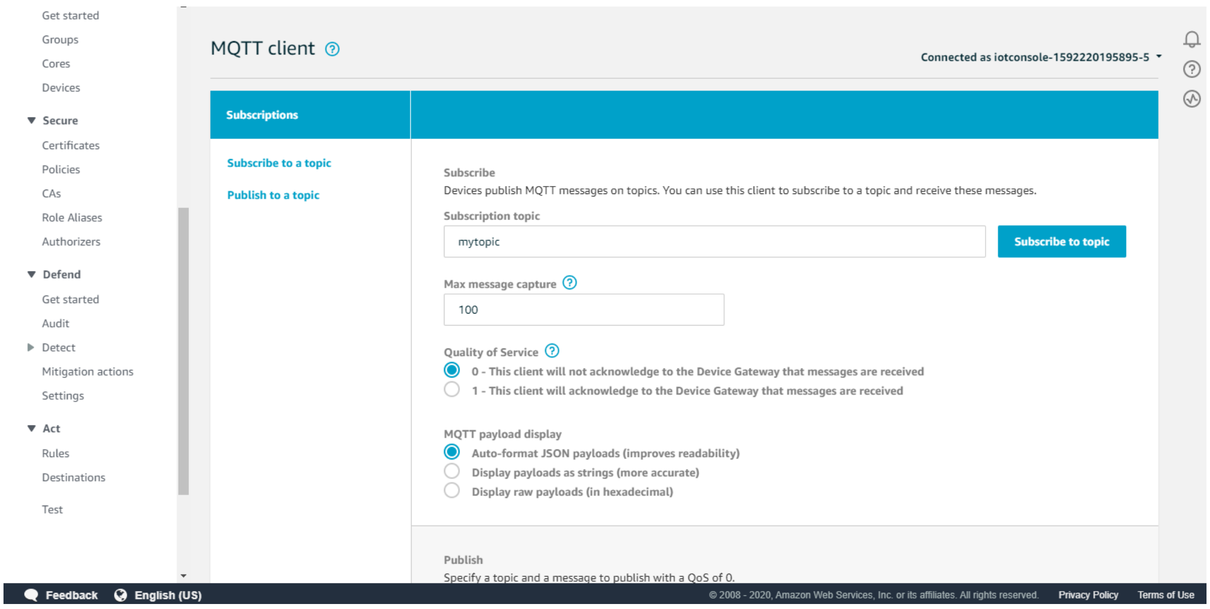
Data types: × location, × activity, × sigfox_network_signal_levels

Test Success

Send to topic [steven2] success to Endpoint [a237qg3xzjydts-ats.iot.eu-west-1.amazonaws.com]

AWS subscribe test

The AWS console can be used to see the messages being published. In the left menu bar select test, fill in the topic name and click 'Subscribe to topic'



MQTT client

Connected as iotconsole-1592220195895-5

Subscribe

Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.

Subscription topic: mytopic

Max message capture: 100

Quality of Service

- [✓ 0 - This client will not acknowledge to the Device Gateway that messages are received]
- 1 - This client will acknowledge to the Device Gateway that messages are received

MQTT payload display

- [✓ Auto-format JSON payloads (improves readability)]
- Display payloads as strings (more accurate)
- Display raw payloads (in hexadecimal)

Publish

Specify a topic and a message to publish with a QoS of 0.

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Now you should see every message published to the topic.

7.6 Testing MQTT webhooks with Azure IoTHub

This section gives a step-by-step explanation on how to configure an MQTT connection to the Microsoft Azure IoT Hub.

More information on the Azure IoT hub can be found here: https://azure.microsoft.com/en-us/services/iot-hub/

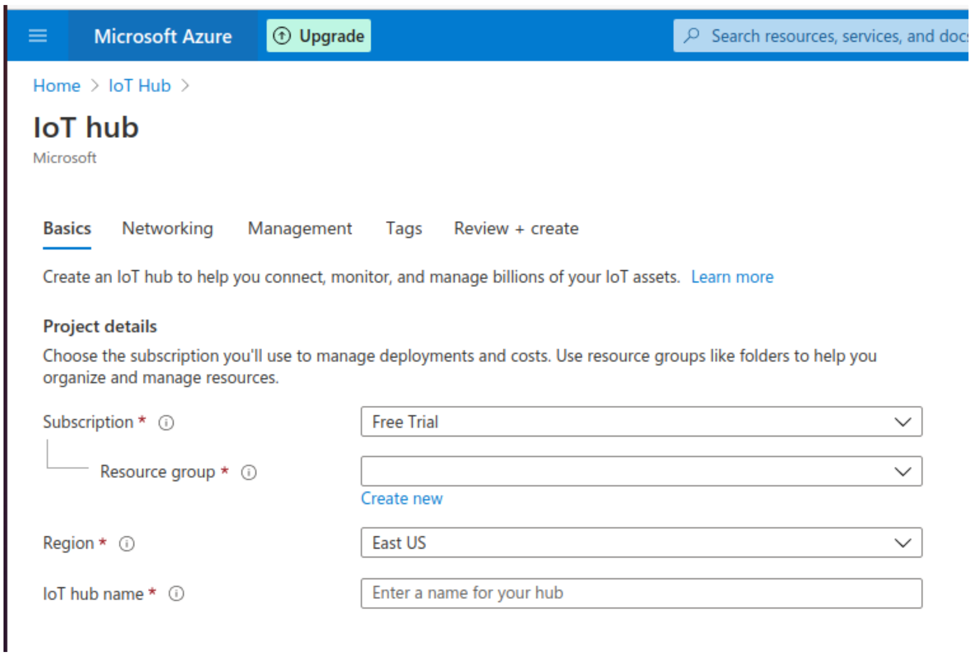
The first step is to create an IoT hub from the Azure portal:

https://portal.azure.com/

Look for IoT hub in the services list or enter IoT hub in the search area.

Create an IoT hub

- you may start with a trial subscription
- just pick a name and a region
- create a new resource group


Microsoft Azure | Upgrade | Search resources, services, and doc

Home > IoT Hub >

IoT hub
Microsoft

Basics | Networking | Management | Tags | Review + create

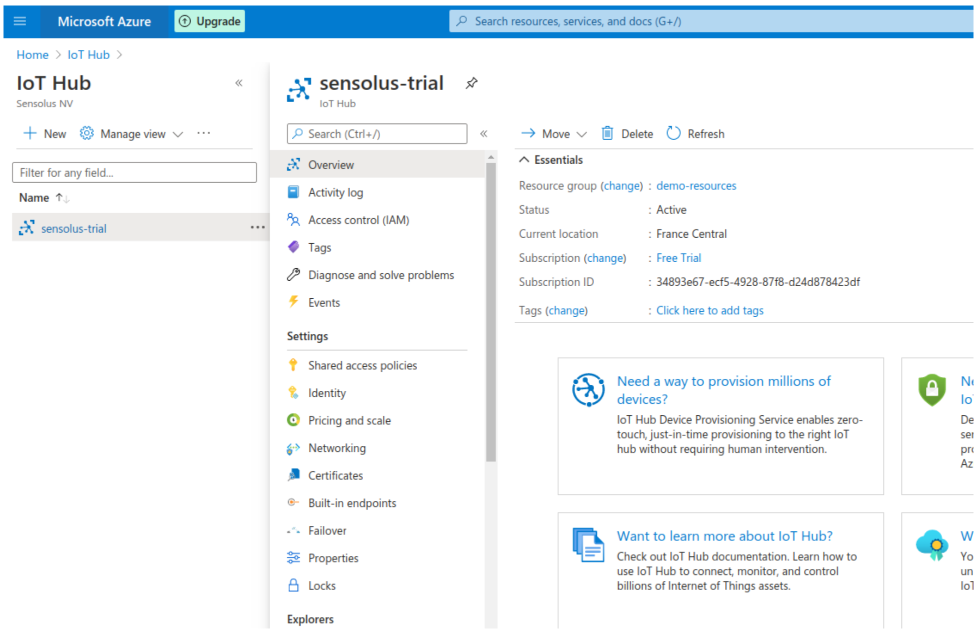
Create an IoT hub to help you connect, monitor, and manage billions of your IoT assets. Learn more

Project details

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription *	Free Trial
Resource group *	Create new
Region *	East US
IoT hub name *	Enter a name for your hub

You should see the following overview page:


Microsoft Azure | Upgrade | Search resources, services, and docs (G+/)

Home > IoT Hub >

IoT Hub
Sensolus NV

New | Manage view | Filter for any field... | Name ↑↓ | sensolus-trial

sensolus-trial
IoT Hub

Search (Ctrl+/) | Overview | Activity log | Access control (IAM) | Tags | Diagnose and solve problems | Events | Settings | Shared access policies | Identity | Pricing and scale | Networking | Certificates | Built-in endpoints | Failover | Properties | Locks | Explorers

Move | Delete | Refresh

Essentials

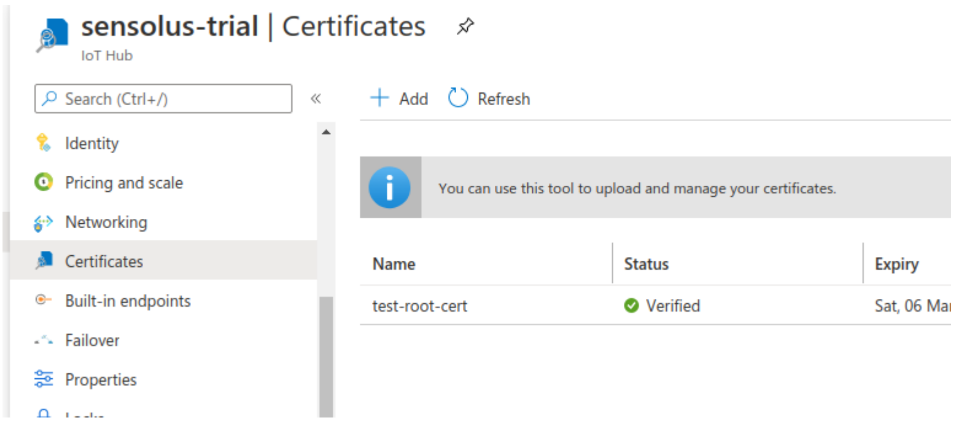
Resource group (change)	: demo-resources
Status	: Active
Current location	: France Central
Subscription (change)	: Free Trial
Subscription ID	: 34893e67-ecf5-4928-87f8-d24d878423df
Tags (change)	: Click here to add tags

Need a way to provision millions of devices?
IoT Hub Device Provisioning Service enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention.

Want to learn more about IoT Hub?
Check out IoT Hub documentation. Learn how to use IoT Hub to connect, monitor, and control billions of Internet of Things assets.

Add a X509 certificate

To safely connect the SNT platform to the IoT hub we will make use of mutual SSL authentication. For this reason you have to add a certificate to your IoT hub.


sensolus-trial | Certificates
IoT Hub

Search (Ctrl+/)
Identity
Pricing and scale
Networking
Certificates
Built-in endpoints
Failover
Properties

+ Add   Refresh

You can use this tool to upload and manage your certificates.

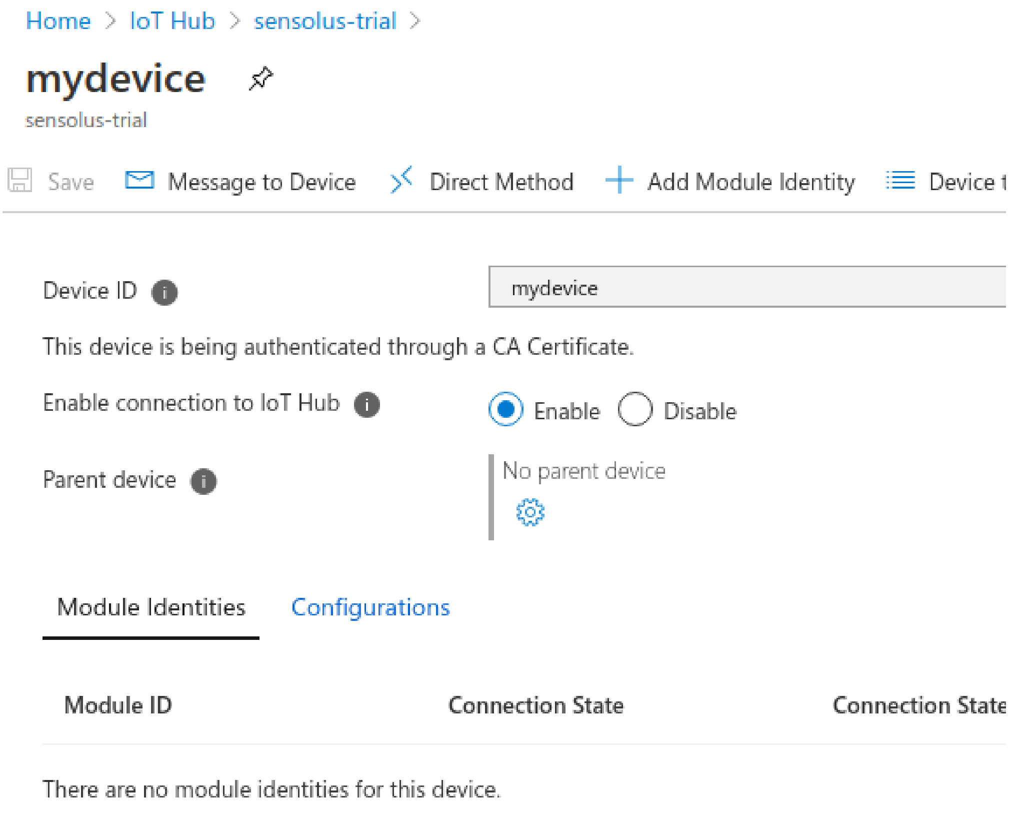
Name	Status	Expiry
test-root-cert	✓ Verified	Sat, 06 Ma

You will have to add a root certificate and also prove that you own it. You can purchase a certificate with a certificate authority but you can also use a self-signed certificate (after all, this is only meant for internal communication).

A nice description and some useful tools such as bash scripts are provided here: https://github.com/Azure/azure-iot-sdk-c/blob/master/tools/CACertificates/CACertificateOverview.md

Create your device

- create a leaf device (edge devices are special devices with custom firmware)
- select the option to authenticate through a CA certificate
- enable connection to IoT hub


Home > IoT Hub > sensolus-trial >

mydevice
sensolus-trial

Save   Message to Device   Direct Method   + Add Module Identity   Device t

Device ID: mydevice

This device is being authenticated through a CA Certificate.

Enable connection to IoT Hub: [✓ Enable] Disable

Parent device: No parent device

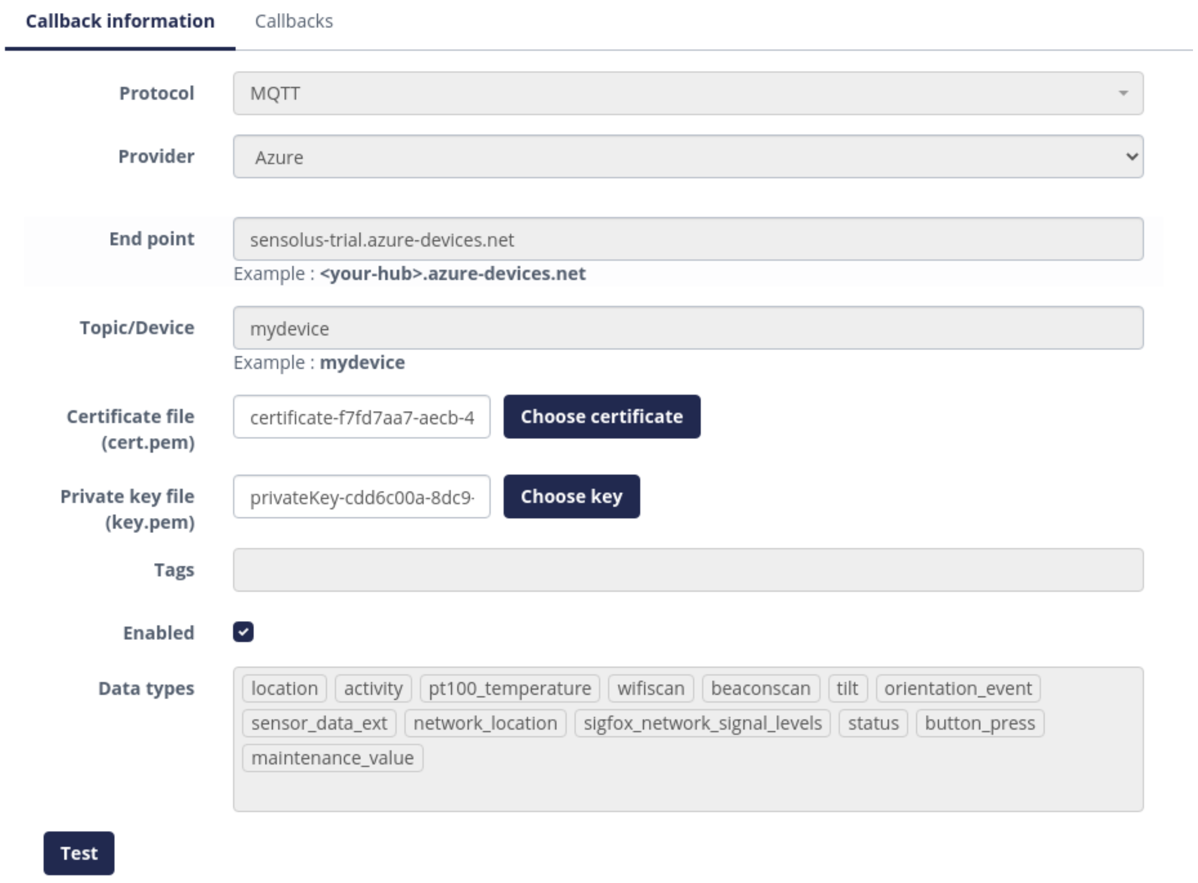
Module Identities   Configurations

Module ID	Connection State	Connection State

There are no module identities for this device.

Sensolus MQTT configuration

Next fill in the MQTT settings in the Sensolus Web application:


Callback information | Callbacks

Protocol	MQTT
Provider	Azure
End point	sensolus-trial.azure-devices.net Example : **<your-hub>.azure-devices.net**
Topic/Device	mydevice Example : **mydevice**
Certificate file (cert.pem)	certificate-f7fd7aa7-aecb-4 [Choose certificate]
Private key file (key.pem)	privateKey-cdd6c00a-8dc9- [Choose key]
Tags	
Enabled	✓
Data types	location, activity, pt100_temperature, wifiscan, beaconscan, tilt, orientation_event, sensor_data_ext, network_location, sigfox_network_signal_levels, status, button_press, maintenance_value

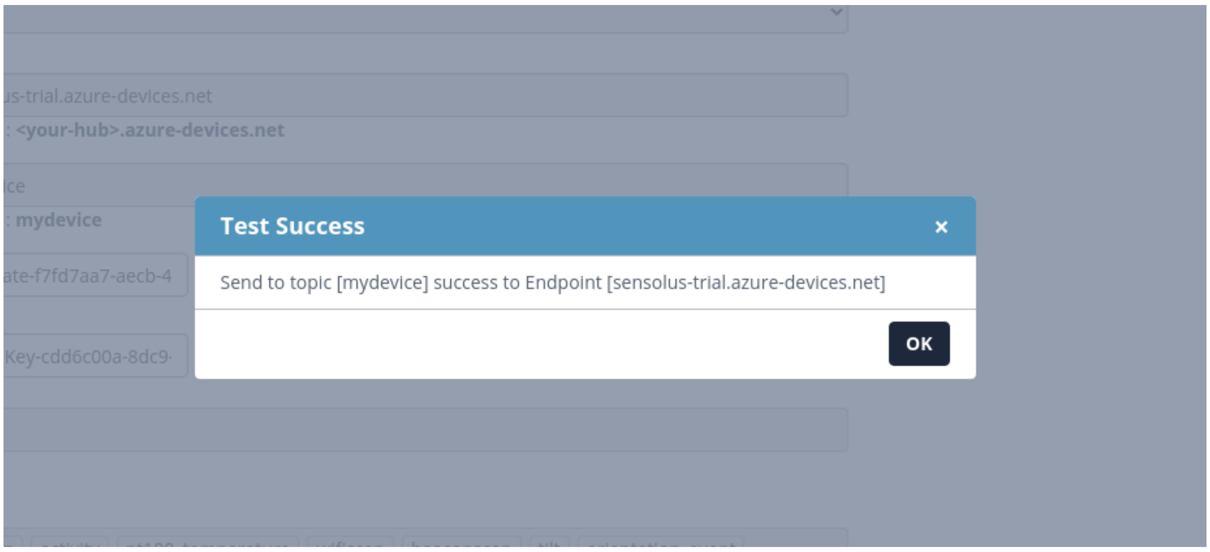
- the end point is just the name of your hub followed by .azure-devices.net
- device is the name of your device
- the certificate file should be a pem file with the certificate. It can be found in the cert folder as <device>.cert.pem if you use the Azure tools. The content looks like this:

```
-----BEGIN CERTIFICATE-----
MIIFfjCCA2agAwIBAgIBAzANBgkqhkiG9w0BAQsFADAqMSgwJgYDVQQDDB9BenVy
ZSBJb1QgSHViIENBIENlcnQgVGVzdCBPbmx5MB4XDTIxMDIwNDE1NTk1MVoXDTIx
...
-----END CERTIFICATE-----
```

- The private key file should be a pem file with the private key. It can be found in the private folder as <device>.key.pem if you use the Azure tools. Content looks like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIIJJwIBAAKCAgEAvwhvebAoHvJNHh+IPFqW9hvcr/gYf22/iwBZDg2k4evkK841
pUPmTPBytnYsVVRAkz/F/2UobVa9PAUCe9I/d+8tFJVuebrJD7DKwU0tPsPcXoab
...
-----END RSA PRIVATE KEY-----
```

In the S to the
topic. I ge:


us-trial.azure-devices.net

: <your-hub>.azure-devices.net

ice

: mydevice

ate-f7fd7aa7-aecb-4

Key-cdd6c00a-8dc9-

Test Success

Send to topic [mydevice] success to Endpoint [sensolus-trial.azure-devices.net]

IoT Hub routing

Once you send messages into IoT Hub, you can consume them on the Event Hub-compatible endpoint of the IoT Hub (https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-d2c#routing-endpoints).

If you need the data in a "real" Event Hub, you can use routing to forward the messages from the IoT Hub into an Event Hub. You can also route messages to a datastore, of course. Please consult the IoT hub documentation.