EnOcean IoT Connector

Product Documentation

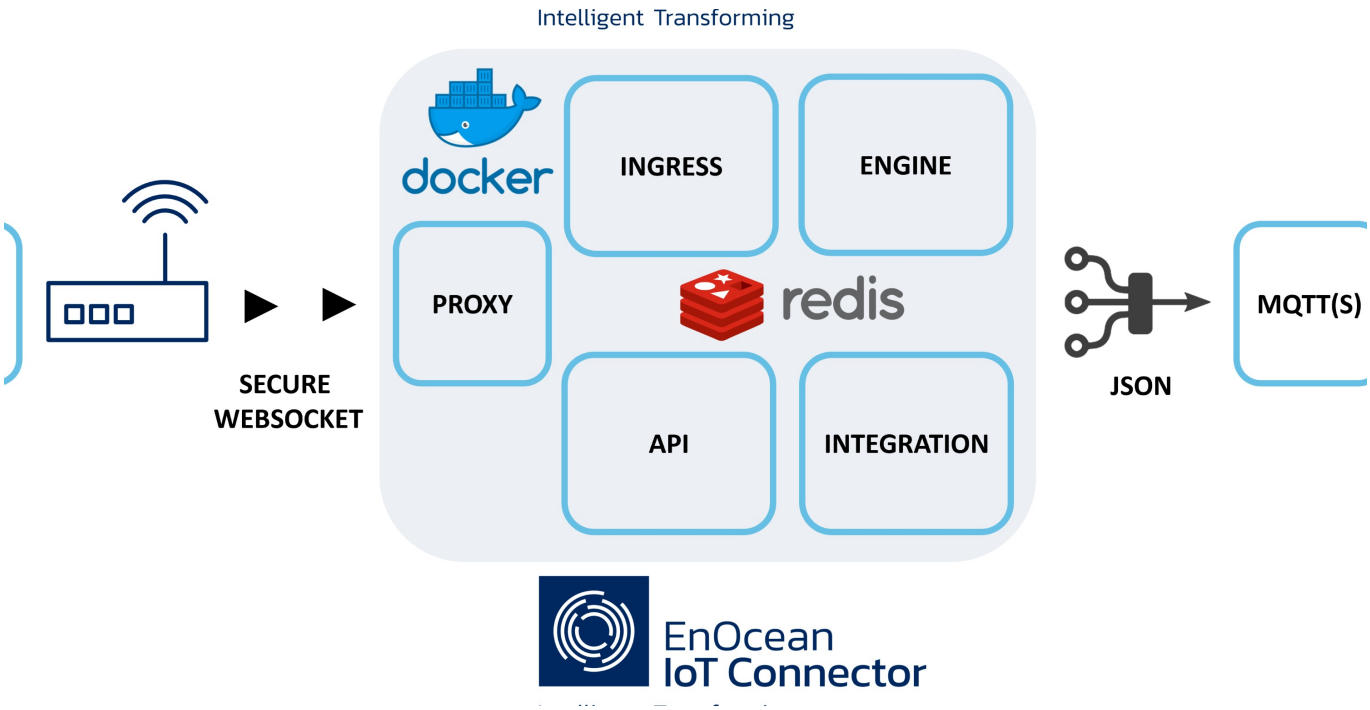
EnOcean GmbH

Table of contents

1. Overview

1.1 EnOcean IoT Connector

The EnOcean IoT Connector (IoTC) allows for the easy processing of the super-optimized EnOcean radio telegrams. The IoTC is distributed as a group of Docker containers. All containers are hosted in the Docker Hub.



The IoTC is composed of the following containers:

1. enocean/iotconnector_ingress
2. enocean/iotconnector_engine
3. enocean/iotconnector_api
4. enocean/iotconnector_integration
5. Redis
6. NGINX

Deploying the IoTC is simple using `docker compose`. For convenience, `docker-compose.yml` files are provided to easily deploy locally (i.e. with Docker) or to Azure Containers Instances (Microsoft Azure cloud account and subscription required).

The IoTC can either be deployed in:

- a public cloud (eg. Azure, AWS)
- private cloud
- on-premise

DOCUMENTATION VERSION/TAG/SHA

`` `/ 2024-01-22 09:39:20+01:00 / f1c3286` ``

1.1.1 Features

Ingress

The ingress controls all incoming traffic from ingress gateways.

- Ingress has a secure web socket support for communication with APs.
- Ingress removes duplicates of data arriving from one sensor via several APs.
- Ingress processes the ESP3 Protocol. Only Packet Type 01 & 10 are supported currently.
- The IoTC currently supports Aruba Access Points as ingress gateways. This list is expanding.

Engine

The IoTC engine completely supports the EnOcean radio protocol standards as defined by the EnOcean Alliance. Additionally, engine evaluates sensor health information, as well as the overall health of EIoTC solution:

- addressing encapsulation
- chaining
- decryption & validation of secure messages
- EEP processing
- information included in signal telegram
- telegram statistics
- health check status

See the Output format description for more details on what the engine can provide.

The following EEPs are supported:

A5	D1	D2	D5	F6
A5-02-05	D1-07-10	D2-01-0B	D5-00-01	F6-02-04
A5-04-01	D1-07-11	D2-01-0F		
A5-04-03	D1-09-01	D2-14-40		
A5-06-02		D2-14-41		
A5-06-03		D2-14-52		
A5-07-01		D2-15-00		
A5-07-03		D2-32-00		
A5-08-01		D2-32-01		
A5-08-02		D2-32-02		
A5-08-03		D2-B1-00		
A5-09-04				
A5-09-09				
A5-10-03				
A5-12-00				
A5-12-01				
A5-14-05				
A5-20-01				
A5-20-06				

A complete description and a list of all existing EEPs can be found here: EEP Viewer. If you are missing an EEP for your application please drop us an email on support-at-enocean-dot-com.

API

The full API Specification is available here or via the web Interface, once the IoTC has been deployed.

The most important features are:

- onboard / update / remove enocean devices
- get most recent data and signal telegrams of a device
- telegram statistic (e.g. count, last seen) for a device and per gateway
- list of connected ingress gateways
- persistent storage of onboarded device - if you specified a volume storage at deployment
- EIoTC health check status

The API container exposes a Web UI for your convenience to see the full documentation and to have a simple client interaction.

Integration

Integration serves as the interface between EIoTC and external systems such as various cloud services that let you build your IoT solution. Currently, we support following integration methods: - MQTT and MQTTS (IoTC acts as a client to an external MQTT broker) - Azure IoT Hub - Azure IoT Central

The output data format is JSON, in accordance to the `key-value` pairs defined by the EnOcean Alliance IP Specification.

NGINX

NGINX is used as a reverse proxy to secure IoTC. It requires valid security certificates for operation.

A `Dockerfile` / `azure.dockerfile` and corresponding dependencies (`start.sh` and `nginx.conf`) are provided at `/deploy/nginx/` in case it needs to be rebuilt or customized.

Redis

Redis is used as a message broker & cache for communication between different containers.

1.1.2 License Agreement and Data Privacy

Please see the License agreement here.

Please see the Data privacy agreement here.

1.1.3 Disclaimer

The information provided in this document describes typical features of the EnOcean software products and should not be misunderstood as specified operating characteristics. No liability is assumed for errors and / or omissions. We reserve the right to make changes without prior notice.

1.2 Prerequisites and used third party libraries

1.2.1 Technical Requirements

The different containers of the IoTC require the Docker environment to run. Specific requirements (i.e. RAM, CPU) depend on the number of connected end points to the IoTC at runtime and their communication frequency. Typical installations (e.g. 100 connected AP, 500 EnOcean end points).

The IoTC was load tested in laboratory conditions with 200 Gateways / APs with trasnfering in total 2000 EnOcean Messages within 10 seconds. No issues or message lose was detected.

For Azure Cloud deployments we recommend to use the `docker-compose.yml` file listed in `azure_deployment` directory.

1.2.2 Used 3rd party components and libraries, OSS Components

Components:

- Redis Community(https://redis.io/)
- Python 3.8 (https://www.python.org/)
- Docker Community (https://docs.docker.com/get-docker/)
- NGINX Community (https://www.nginx.com/)
- Mosquitto (https://mosquitto.org/)

Python Libraries:

- Async Redis (aioredis,https://github.com/aio-libs/aioredis-py, MIT License)
- HIREDIS (hiredis,https://github.com/redis/hiredis,BSD License)
- Licensing (licensing,https://github.com/Cryptolens/cryptolens-python,MIT License)
- Protobuf (protobuf,https://developers.google.com/protocol-buffers/,https://github.com/protocolbuffers/protobuf/blob/master/LICENSE)
- Pydantic (pydantic,https://github.com/samuelcolvin/pydantic/,MIT License)
- Redis (redis,https://github.com/andymccurdy/redis-py,MIT License)
- Tornado (tornado,https://github.com/tornadoweb/tornado,Apache License 2.0)
- Flask (flask,https://flask.palletsprojects.com/en/1.1.x/,BSD=https://flask.palletsprojects.com/en/0.12.x/license/)
- Conexion (conexion,https://github.com/zalando/connexion,https://github.com/zalando/connexion/blob/master/LICENSE.txt)
- Azure (azure,https://github.com/Azure/azure-sdk-for-python,MIT)
- Bitstring (bitstring,https://github.com/scott-griffiths/bitstring,MIT)
- crc8 (crc8,https://github.com/niccokunzmann/crc8,MIT)
- paho-mqtt (paho-mqtt,http://www.eclipse.org/paho/,BSD=https://projects.eclipse.org/projects/iot.paho)
- pycryptodome (pycryptodome, https://github.com/Legrandin/pycryptodome,https://github.com/Legrandin/pycryptodome/blob/master/LICENSE.rst)
- Celery (celery,https://github.com/celery/celery,https://github.com/celery/celery/blob/master/LICENSE)

2. Deployment

2.1 Generating self-signed certificates

> **Warning**
>
> Self-signed certificates are inherently insecure (since they lack a chain of trust). Please contact your IT Admin if you are unsure/ unaware of the consequences of generating & using self-signed certificates. These instructions should be used for development environments only.

For Windows users: Use the `openssl` Docker image to generate a CA, CSR and finally a certificate. **Create a dedicated folder for the process.**

For Linux users: Since most Linux distributions already include `openssl` there is no need to use docker for this step. Simply run the command directly by removing the initial call to docker: `docker run -it --rm -v ${PWD}:/export frapsoft/` . **Create the export directory at root to simplify the process.**

2.1.1 Generate private key for CA authority:

Windows **Linux**

```
docker run -it --rm -v ${PWD}:/export frapsoft/openssl genrsa -des3 -out /export/myCA.key 2048

$ mkdir /export
$ cd /export
$ openssl genrsa -des3 -out /export/myCA.key 2048
```

Complete the fields with the information corresponding to your organization.

2.1.2 Generate root certificate

Windows **Linux**

```
docker run -it --rm -v ${PWD}:/export frapsoft/openssl req -x509 -new -nodes -key /export/myCA.key -sha256 -days 1825 -out /export/myCA.pem

$ openssl req -x509 -new -nodes -key /export/myCA.key -sha256 -days 1825 -out /export/myCA.pem
```

For common name enter the hostname of the deployment or `localhost` for local test deployments.

2.1.3 Generate a key for the certificate going into the connector

Windows **Linux**

```
docker run -it --rm -v ${PWD}:/export frapsoft/openssl genrsa -out /export/dev.localhost.key 2048

$ openssl genrsa -out /export/dev.localhost.key 2048
```

2.1.4 Generate a CSR for the connector

Windows **Linux**

```
docker run -it --rm -v ${PWD}:/export frapsoft/openssl req -new -key /export/dev.localhost.key -out /export/dev.localhost.csr

$ openssl req -new -key /export/dev.localhost.key -out /export/dev.localhost.csr
```

For common name enter the hostname of the deployment or `localhost` for local test deployments.

2.1.5 Create the .ext file

Create a new `localhost.ext` file with the following contents:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
subjectAltName = @alt_names
subjectKeyIdentifier = hash

[alt_names]
DNS.1 = localhost
IP.1 = 192.168.1.2
```

Edit the `localhost.ext` file to match your domain. Make sure the `DNS.1` matches the hostname of your deployment. If you're accessing the deployment via IP address make sure the `IP.1` matches the IP address.

2.1.6 Generate a certificate from CSR for the connector

Windows **Linux**

```
docker run -it -v ${PWD}:/export frapsoft/openssl x509 -req -in /export/dev.localhost.csr -CA /export/myCA.pem -CAkey /export/myCA.key -CAcreateserial -out /export/
dev.localhost.crt  -days 825 -sha256 -extfile /export/localhost.ext

$ openssl x509 -req -in /export/dev.localhost.csr -CA /export/myCA.pem -CAkey /export/myCA.key -CAcreateserial -out /export/dev.localhost.crt  -days 825 -sha256 -
extfile /export/localhost.ext
```

Keep the generated files safe and without access of 3rd parties.

2.2 Deployment Notes

2.2.1 Environment Variables

Mandatory

To deploy the IoTC certain environment variable must be specified. Mandatory paramters are listed below.

Container	Environment Variable	Usage
ingress,engine	IOT_LICENSE_KEY	IoTC license key. Contact your EnOcean sales partner.
ingress	IOT_GATEWAY_USERNAME	Username used for the AP authentication.
ingress	IOT_GATEWAY_PASSWORD	Password used for AP authentication.
ingress	IOT_AUTH_CALLBACK	Authentication callback for APs. The `hostname of the container group instance` + `:8080`. Example: `192.167.1.1:8080` or `myiotc.eastus.azurecontainer.io:8080`
proxy	BASIC_AUTH_USERNAME	User name for basic authentication on the API interface.
proxy	BASIC_AUTH_PASSWORD	Password for basic authentication on the API interface.
engine	INGRESS_USERNAME	Username used for the AP authentication.
engine	INGRESS_PASS	Password used for the AP authentication).
engine	MQTT_CONNSTRING	MQTT broker address:port.

End Point connection

For endpoints one of the integrations needs to be selected.

Container	Environment Variable	Usage
engine	IOT_AZURE_CONNSTRING	The *Connection String* to be use for sending data to the Azure IoT Hub.
	IOT_AZURE_ENABLE	This variable enables the Azure IoT Hub end-point. If this variable is set, the *IOT_AZURE_CONNSTRING* variable must also be set. **If you do not wish to send data to the Azure IoT Hub, don't set this variable, simply leave it out.**
	MQTT_CONNSTRING	The *Connection String* to be use for publishing data to an MQTT broker.
	IOT_ENABLE_MQTT	This variable enables publishing of telemetry into an MQTT broker. **If you do not wish to send data to an MQTT broker, don't set this variable, simply leave it out.**
	IOT_MQTT_CLIENT_ID	MQTT Client ID variable used for the IoTC as client for the MQTT protocol
	MQTT_AUTH	Set to true and specify the basic auth parameters (`MQTT_USERNAME` & `MQTT_PASSWORD`) for MQTT connection.
	MQTT_USERNAME	Username used for MQTT connection. **Required if MQTT_AUTH is true**
	MQTT_PASSWORD	Username used for MQTT connection. **Required if MQTT_AUTH is true**

Reporting behavior

Container	Environment Variable	Usage
engine	GATEWAY_STATS_INTERVAL	Report interval in seconds for Gateway statistics as described here. When setting to 0 or not setting the variable at all, the reports are off.
	SENSOR_STATS_INTERVAL	Report interval in seconds for Sensor statistics as described here. When setting to 0 or not setting the variable at all, the reports are off.
engine	SENSOR_TELEMETRY	Specify a custom MQTT publish path for the sensor telemetry. Default path is listed here. The `ID` identifies a specific device and is represented as by `<ID>` in the custom PATH. e.g. `SENSOR_TELEMETRY="devices/telegram/<ID>` will result in a new path `devices/telegram/aabbccdd`.
	SENSOR_EVENT	Specify a custom MQTT publish path for the sensor meta events. Default path is listed here. The `ID` identifies a specific device and is represented as by `<ID>` in the custom PATH. e.g. `SENSOR_EVENT="devices/event/<ID>` will result in a new path `devices/event/aabbccdd`.
	SENSOR_STATS	Specify a custom MQTT publish path for the sensor meta stats. Default path is listed here. The `ID` identifies a specific device and is represented as by `<ID>` in the custom PATH. e.g. `SENSOR_STATS="devices/stats/<ID>` will result in a new path `devices/stats/aabbccdd`.
	GATEWAY_EVENT	Specify a custom MQTT publish path for the gateway meta events. Default path is listed here. The `MAC` identifies a specific gateway and is represented as by `<MAC>` in the custom PATH. e.g. `GATEWAY_EVENT="ap/event/<MAC>` will result in a new path `ap/event/aabbccddeeff`.
	GATEWAY_STATS	Specify a custom MQTT publish path for the gateway meta events. Default path is listed here. The `MAC` identifies a specific gateway and is represented as by `<MAC>` in the custom PATH. e.g. `GATEWAY_STATS="ap/stats/<MAC>` will result in a new path `ap/stats/aabbccddeeff`.
engine	HEALTH_PUBLISH_INTERVAL	Report interval in seconds for system health

API Behavior

Environment variables that control IoTC API behavior are listed below:

Container	Environment Variable	Usage
api	ONLY_SECURE_DEVICES	Only allow secure devices. If this variable is set only devices with AES key and SLF properties would be allowed into EIoTC.

2.2.2 Overview of Secrets

Secret	Usage
secret-proxy-certificate	Certificate for the NGINX proxy to protect IoTC interfaces.
secret-proxy-key	Private key of the certificate for the NGINX proxy.
mqtt-ca-cert	MQTTS broker CA certificate.
mqtt-client-cert	MQTTS client certificate.
mqtt-client-key	MQTTS client private key.

2.2.3 Ports

The following ports are used:

Service	Description	Port
Management API	Used to commission EnOcean devices into the IoTC. A Swagger UI is available on the root. Supported protocols: `https`.	443 (requests on port 80 will be redirected)
WebSocket Ingress	WebSocket end-point for IoTC compatible gateways. Supported protocols: `wss`.	8080
MQTT (Optional deployment)	Mosquitto MQTT broker. Supported protocols: `mqtt`.	1883

Note

Should different ports mapping be needed please contact EnOcean support for detailed instructions.

2.2.4 License key

To deploy the IoTC a license key is required. You can get a license from the product page or please use the contact form.

Each license is specified for a defined usage. The usage is defined by a maximum number of sensor/gateways which will be processed by the IoTC. If the consumption is reached additional sensors or gateways will be dropped at processing.

You can see the allowed usage of each of your licenses after you log in to the licensing portal. After EnOcean has assigned a license you will receive an invitation e-mail.

Log information about the license status and consumption limit is posted to the console.

License activation

There is a license activation limit. If you deploy the IoTC several times within a very short period (e.g. during testing, debugging), you might experience license activation failed. Please wait for couple of minutes and try again.

The IoT Connector has to communicate with our licensing server periodically to reactivate the license. If the IoT Connector can not successfully activate the license the IoT Connector will cease to process incoming traffic after a defined grace period. The grace period is only valid if the IoT Connector could validate the license at least once. Details are include in the Licensing Agreement.

2.3 Deploy and Connect Devices

2.3.1 1. Step by step deployment

Preparation

1. Clone this repository `git clone https://bitbucket.org/enocean-cloud/iotconnector-docs.git` or download the repository files. This should be downloaded to a directory in which you have edit and execute files rights.
2. Prepare your certificate. If do not have one, you can generate a self-signed certificate, in this case prepare the `"myCA.pem"` file for the Aruba AP.
3. Prepare the `*.crt` and `*.key` file from your CA for the NGINX proxy. If you do not have one, you can generate a self-signed certificate.
4. Find and note the EnOcean ID - EURID (32bit e.g. 04 5F 69 4E) and EEP (e.g. D2-14-41) of the EnOcean sub-gigahertz enabled devices you like to use with the IoTC.

   This information is available:

- On the product label - in text and QR code format
- In NFC memory (check availability with manufacturer)
- In the teach-in telegram.

Optionally find and note also the encryption parameters `AES Key` & `SLF` to use encryption with EnOcean devices. Confirm with manufacturer of the device how to operate the device in secure mode in advance.

Deployment

Decide if you want to deploy the IoTC:

- in a native (local installed) Docker or
- in the Microsoft Azure Container instances - ACI.

> **Note**
>
> Please consider that ACI does not 100% equal to a native docker environment, details for the compose process can looked up here.

Deployment in other cloud platforms is also possible but has not been tested.

Local Deployment Azure Deployment

To deploy the IoTC locally. For example on an PC or Raspberry Pi:

1. Go to the `/deploy/local_deployment/` directory
2. Open the `docker-compose.yml` file and add the following environment variables:

a. **IOT_LICENSE_KEY **

In `ingress` and `engine`. See License key notes for details.

```
ingress:
    image: enocean/iotconnector_ingress:latest
environment:
        - IOT_LICENSE_KEY= #enter license here, be sure not to have empty space after "=" e.g. IOT_LICENSE_KEY=LBIBA-BRZHX-SVEOU-ARPWB

engine:
    image: enocean/iotconnector_engine:latest
    environment:
        - REDIS_URL=redis
        - IOT_LICENSE_KEY= #enter license here, be sure not to have empty space after "=" e.g. IOT_LICENSE_KEY=LBIBA-BRZHX-SVEOU-ARPWB
```

b. **IOT_AUTH_CALLBACK**

The `IOT_AUTH_CALLBACK` is formed by taking the IP address or hostname of your instance + `:8080` . If you are working on a local network with DHCP make sure the IP address stays static.

```
ingress:
    image: enocean/iotconnector_ingress:latest

    environment:
        - IOT_AUTH_CALLBACK= #enter URL here e.g. 192.167.1.1:8080 or myiotc.eastus.azurecontainer.io:8080
```

c. **IOT_GATEWAY_USERNAME** & **IOT_GATEWAY_PASSWORD**

Create a `IOT_GATEWAY_USERNAME` and `IOT_GATEWAY_PASSWORD`. These two environment variables are needed for the connection between Aruba AP and IoTC.

```
ingress:
    image: enocean/iotconnector_ingress:latest

    environment:
        - IOT_GATEWAY_USERNAME= #enter new username for the AP connection to IoTC. e.g. user1
        - IOT_GATEWAY_PASSWORD= #enter new password for the AP connection to IoTC. e.g. gkj35zkjasb5
```

d. **BASIC_AUTH_USERNAME** & **BASIC_AUTH_PASSWORD**

The selected username and password will be used to access the API and its web UI.

```
proxy:
    image: enocean/proxy:latest

    environment:
        - BASIC_AUTH_USERNAME=  #enter new username for API connection of IoTC. e.g. user1
        - BASIC_AUTH_PASSWORD=  #enter new password for API connection to IoTC. e.g. 5a4sdFa$dsa
```

e. **PROXY_CERTIFICATE** & **PROXY_CERTIFICATE_KEY**

Configure the `secrets` for the NGINX proxy with the `.crt`, `.key` files you have prepared.

```
#secrets are defined by docker to keep sensitive information hidden
secrets:
    secret-proxy-certificate:
        file: ../nginx/dev.localhost.crt # specify path to .crt
    secret-proxy-key:
        file: ../nginx/dev.localhost.key # specify path to .key
```

Note

For advanced users, if you need to make changes to the NGINX proxy the `Dockerfile`, `start.sh` and `nginx.conf` are available in the `/deploy/nginx` folder and can be changed and rebuilt as necessary.

f. Select the end-point for the IoTC.

Azure IoT Hub or MQTT client is available. At least one end-point must be enabled.

Azure IoT Hub MQTT

List **IOT_AZURE_CONNSTRING** & **IOT_AZURE_ENABLE**

```
    engine:
        image: enocean/iotconnector_engine:latest
        environment:
```

2.3.2 2. Connect Ingress Gateways

After you have deployed the IoTC connect some APs to it with attached EnOcean USB Dongles.

Connect Aruba AP

Please follow the guides under the section Setup Aruba AP

```
!!! Note
    In general APs will be visible in the list & console only when any EnOcean radio traffic is present. Aruba APs from AOS 8.8.x.x will send an `empty hello message` after few minutes which makes the AP also visible in the list.
```

2.3.3 3. Onboard devices using the API

To see any outputs at the End-points an EnOcean device needs to be onboarded to the IoTC.

Using Web UI of the API

Onboardin can be done with the management API web UI.

1. Open URL in browser `https://<hostname of the container group or IP address>:443`
2. Login using `BASIC_AUTH_USERNAME` & `BASIC_AUTH_PASSWORD`. Specified in environmental variables.
3. Use `POST /device` to add the devices one by one or `POST /backup` all at once.

   Have the `EnOcean ID -> sourceEurid` and `eep` prepared.

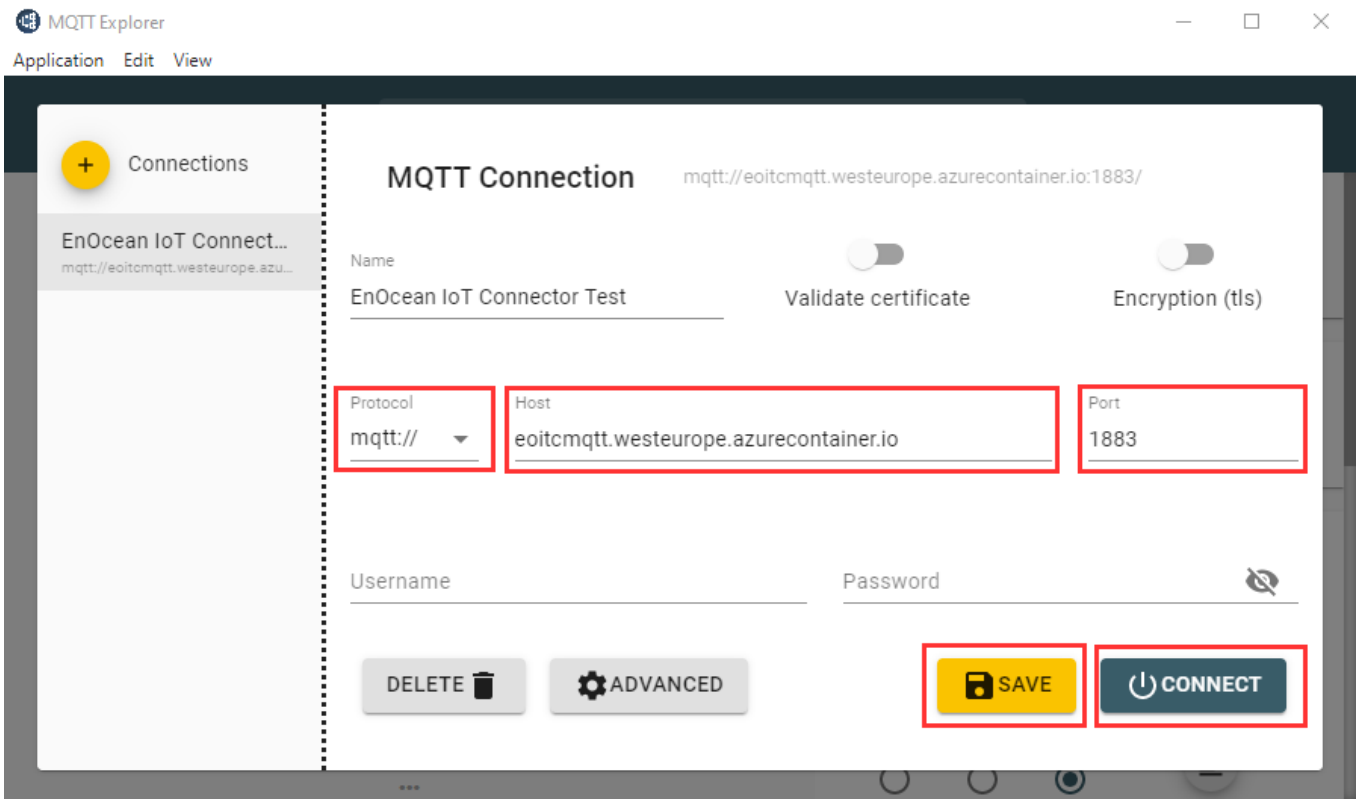
   Additionally specify a `friendlyID` and `location` of the sensor.

   Minimum parameters are:

```
{
  "eep": "A5-04-05",
  "friendlyID": "Room Panel 02",
  "location": "Level 2 / Room 221",
  "sourceEurid": "a1b2c3d4"
}
```

   Check the API Documentation for the complete schema.
4. Check the return code to see if the operation was successful or use `GET /backup` and check if all of your sensors are present.
5. After adding a device you should see any received telegrams from it on the selected end-points. When the first message is received from a new sensor, a message will be logged to the console.

> **Note**
>
> If you have specified to deploy the mosquitto broker as part of the `docker-compose.yml` you can reach it at PORT `:1883` and should see now some messages incoming. The URL will be e.g. `mqtt://192.167.1.1:1883` or `mqtt://myiotc.eastus.azurecontainer.io:1883`
>
> To connect to the broker you can use any kind of MQTT client. e.g. MQTT Explorer.



2.4 Add sensors

A device is commissioned with IoTC using the `POST /device` API with the json schema below. Some parameters are optional depending on if the device is secure included or supports bidirectional communication.

```
{
  "friendlyid": "Room Panel 02",
  "eep": "A5-04-05",
  "deviceType": "sensor",
  "eurid": "A1B2C3D4",
  "destinationEurid": "FFFFFFFF",
  "location": "Level 2 / Room 221",
  "slf": "f3",
  "AESKey": "3A0C1B30B0A822A17A28FD01D77ABDAE"
  "productId": "004900000005",
  "unlockCode": "FFFFFFFE"
}
```

Key	Required or Optional	Type	Description
friendlyid	always required	string	Device name used in IoTC
eep	always required	string	EnOcean Equipment Profile
deviceType	always required	string	Device type defines communication between device and IoTC
eurid	always required	string	EnOcean device specific ID
destinationEurid	always required	string	Used to address communication from device to IoTC. Must always be set to FFFFFFFF
location	always required	string	Device location used in IoTC
slf	required for secure devices	string	Security level format as defined by the EnOcean Alliance. Currently only f3 is supported
AESKey	required for secure devices	string	Device AES key used for secure encryption
product_id	required for bidirectional communication	string	Device product ID, used by IoTC to identify how to configure device for bidirectional communication
unlock_code	required for bidirectional communication	string	May be required to configure device for bidirectional communication

Supported values in `deviceType`

Key	Description
sensor	Devices only sending data to IoTC
bidirectional	Devices supporting bidirectional communication
switch	Wall switch devices based on EnOcean PTM module

3. Setup Aruba AP

3.1 Configuration using Aruba Instant

3.1.1 Required Hardware and Software

- Aruba AP: Aruba AP with USB port.

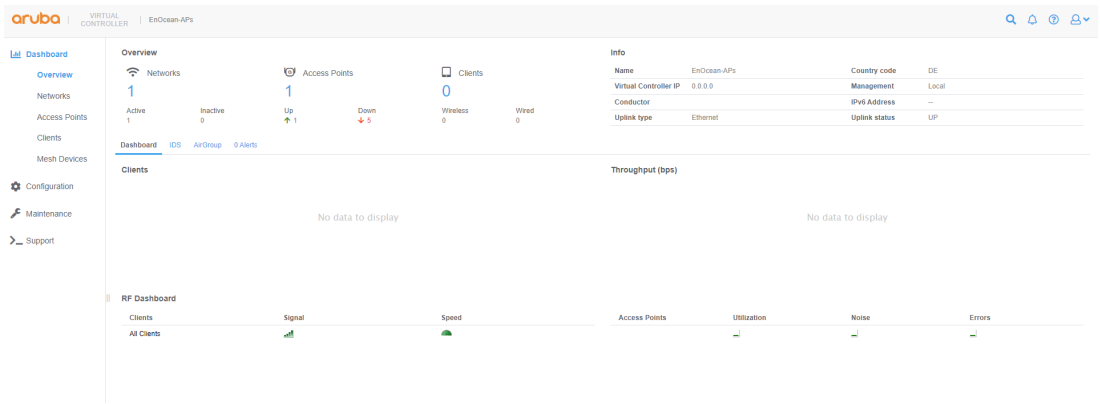
  Check the energy requirements of our Aruba AP to properly operate the USB port.

AP model	USB port (5W)	IPM feature	802.3af (class 3)	802.3at (class 4)	802.3bt (class 5+)	DC power	AC power
AP-303	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-303P	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-304/305	yes	yes	disabled	OK	OK	OK	not supported
AP-314/315	yes	yes	disabled	OK	OK	OK	not supported
AP-324/325	yes	no	disabled	OK	OK	OK	not supported
AP-334/335	yes	yes	disabled	disabled	disabled	OK	not supported
AP-344/345	yes	yes	disabled	disabled	disabled	OK	not supported
AP-504/505	yes	yes	disabled	OK	OK	OK	not supported
AP-514/515	yes	yes	disabled	OK	OK	OK	not supported
AP-534/535	yes	yes	not supported	disabled	OK	OK	not supported
AP-555	yes	yes	not supported	disabled	OK	OK	not supported
AP-203H	no	no	no USB port	no USB port	no USB port	not supported	not supported
AP-303H	yes	yes	disabled	disabled when P	disabled when P	OK	not supported
AP-503H	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-505H	yes	yes	disabled	disabled when P	OK	OK	not supported
AP-203R	yes	no	not supported	not supported	not supported	not supported	OK

- Aruba OS: version **8.10.0.0** or newer (most likely requires update to latest).
- EnOcean USB Stick: USB 300, USB 300U, USB 500 or USB 500U

3.1.2 Step 1: Connect to Instant

Log into the router's web-based management page for Aruba instant.



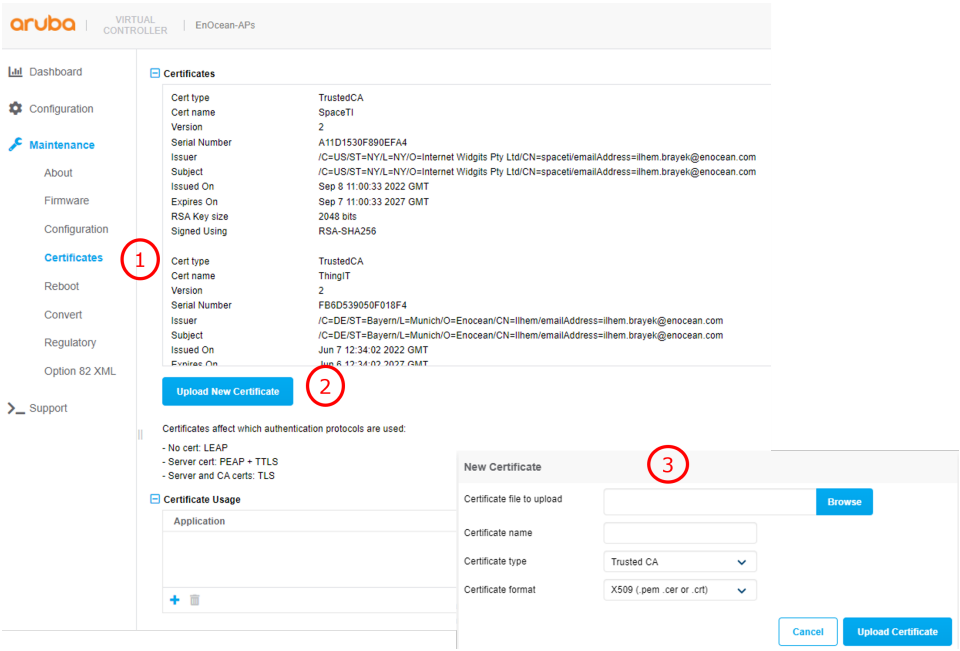
3.1.3 Step 2: Installing Trusted CA Certificates

Note

When using Public Validated Certificates with Aruba Central, the complete certificate path (root + intermediate CAs) must be installed as **one file**.

- Upload your .pem certificate file From **Maintenance -> Certificates -> Upload new certificate**.

  select **Trusted CA** as **certificate type** and **x509(.pem .cer or .crt)** as **Certificate format**.

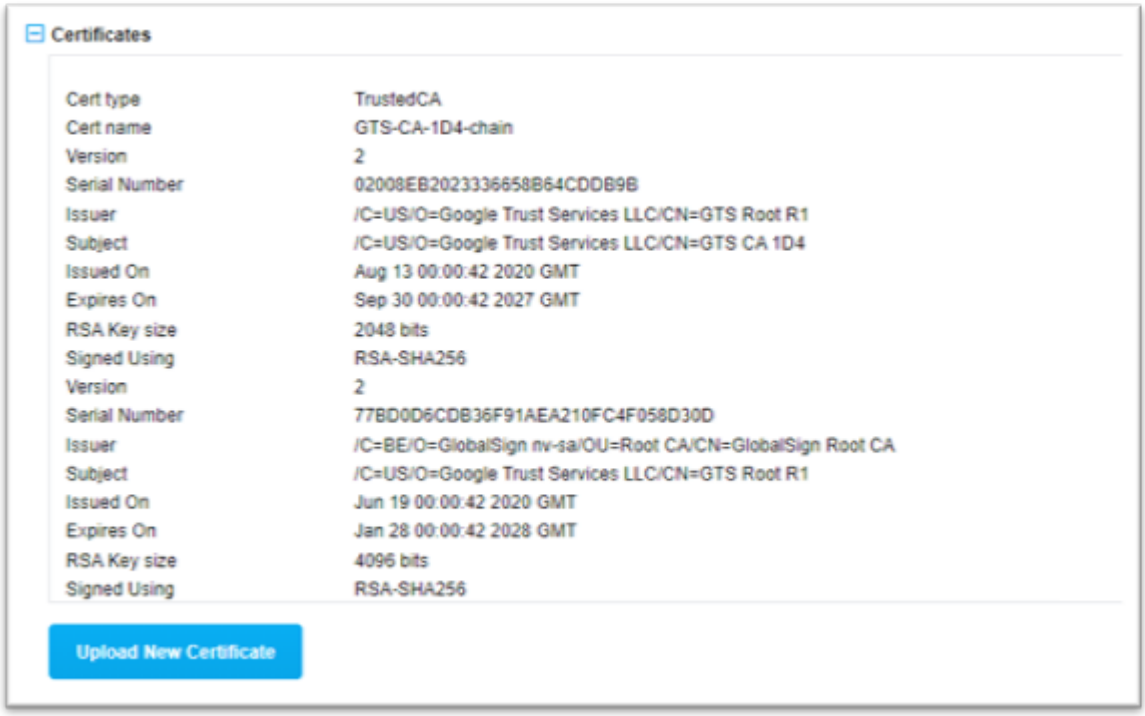


- Browse to certificate. Certificate must be provided as **one file** including complete certificate path (root + intermediate CA). To create the certificate file chain for public validated certificates follow the guide : Using chained certificate.
- Click **Upload Certificate** to save your settings.

Verify Certificate upload.

Verify the certificate is shown on the certificate list:

- Using the UI:

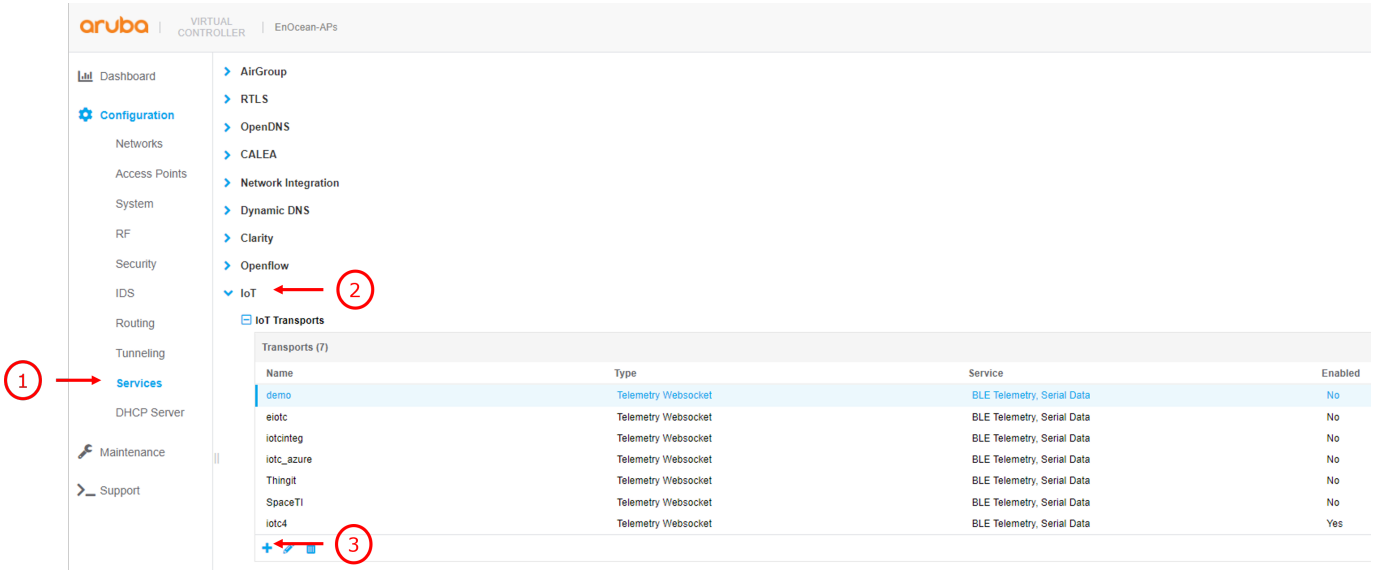


- Using the CLI:

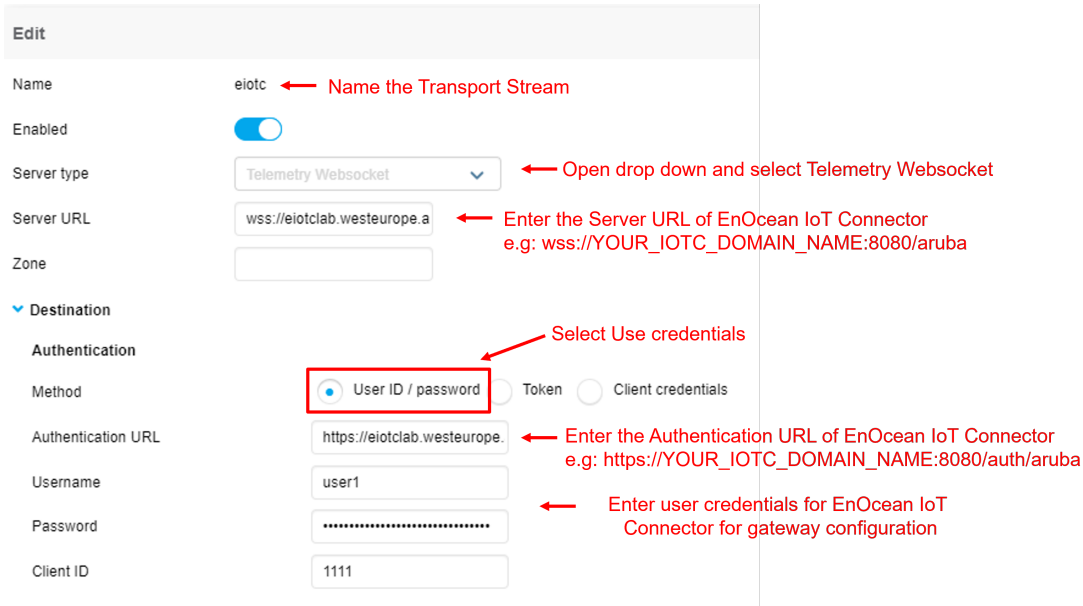
```
(IAP)# show cert all

Cert type:TrustedCA
Cert name:GTS-CA-1D4-chain
Version       :2
Serial Number :02008EB2023336658B64CDDB9B
Issuer        :/C=US/O=Google Trust Services LLC/CN=GTS Root R1
Subject       :/C=US/O=Google Trust Services LLC/CN=GTS CA 1D4
Issued On     :Aug 13 00:00:42 2020 GMT
Expires On    :Sep 30 00:00:42 2027 GMT
RSA Key size  :2048 bits
Signed Using  :RSA-SHA256

Version       :2
Serial Number :77BD0D6CDB36F91AEA210FC4F058D30D
Issuer        :/C=BE/O=GlobalSign nv-sa/OU=Root CA/CN=GlobalSign Root CA
Subject       :/C=US/O=Google Trust Services LLC/CN=GTS Root R1
Issued On     :Jun 19 00:00:42 2020 GMT
Expires On    :Jan 28 00:00:42 2028 GMT
RSA Key size  :4096 bits
Signed Using  :RSA-SHA256

Version       :2
Serial Number :040000000001154B5AC394
Issuer        :/C=BE/O=GlobalSign nv-sa/OU=Root CA/CN=GlobalSign Root CA
Subject       :/C=BE/O=GlobalSign nv-sa/OU=Root CA/CN=GlobalSign Root CA
Issued On     :Sep  1 12:00:00 1998 GMT
Expires On    :Jan 28 12:00:00 2028 GMT
RSA Key size  :2048 bits
Signed Using  :RSA-SHA1
```

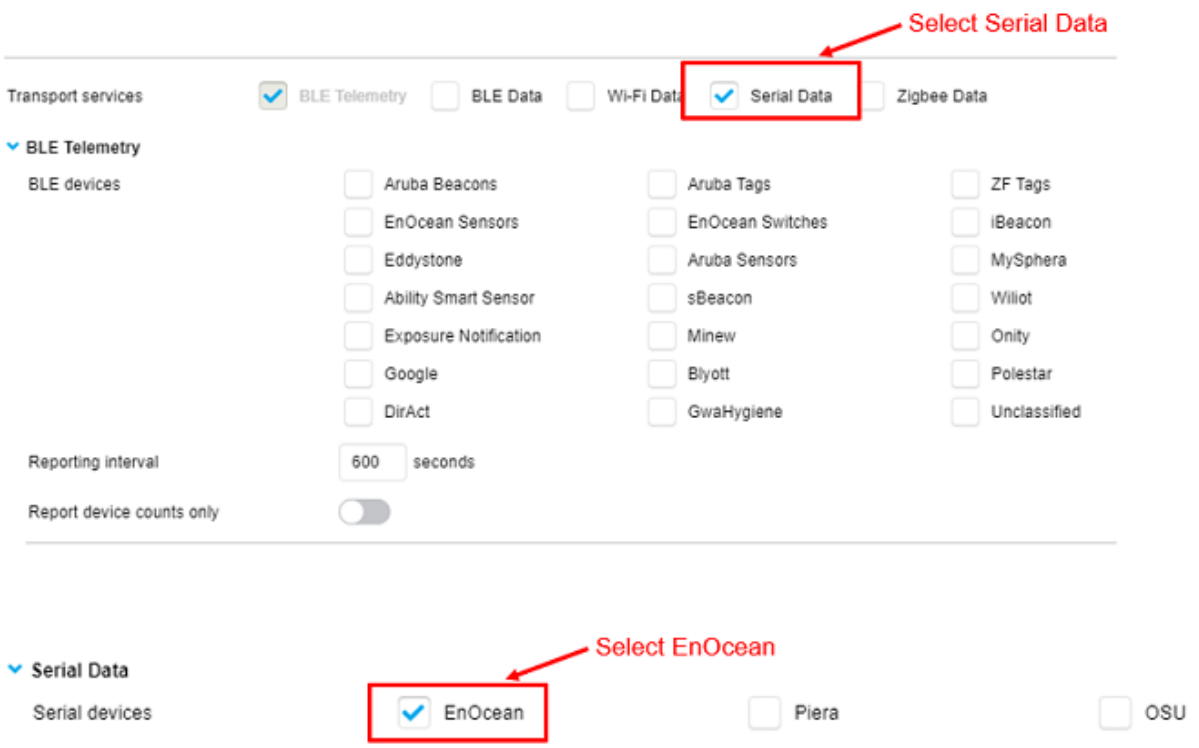
3.1.4 Step 3: IoT transport configuration

- In Aruba Instant select **Configuration** -> **Services** -> **IoT** -> **IoT transports** then add a new transport using the **+** icon:



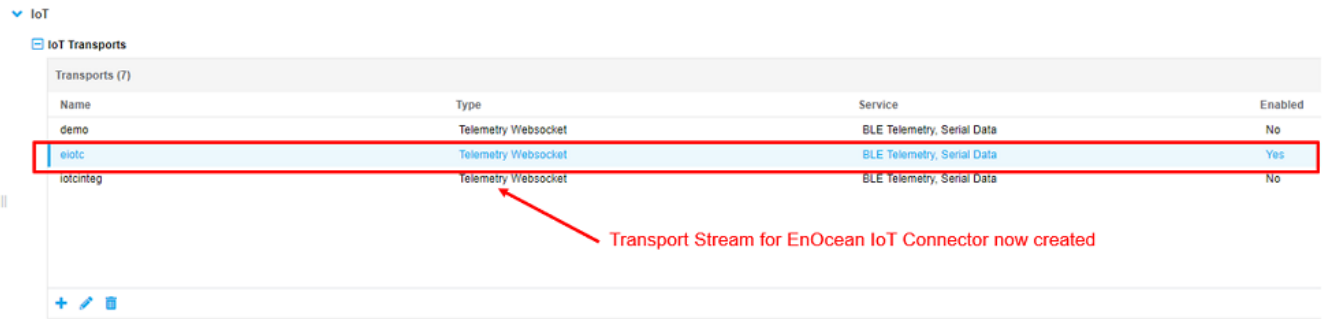
• Enter the following information in the IoT transport popup:





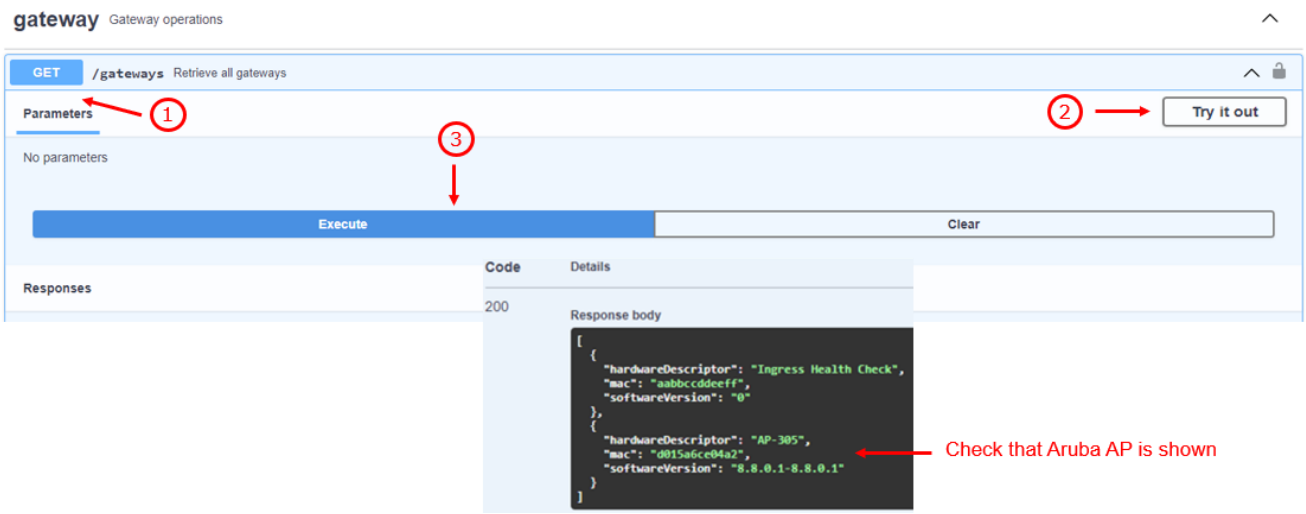
Click **OK** then **Save** to save you configuration.

- Check that you transport stream is enabled:



3.1.5 Step 4: Verify that your Gateway is connected

Login to EnOcean IoT Connector and check gateway status using GET/gateways API



3.2 Configuration using Aruba Controller

3.2.1 Required Hardware and Software

- Aruba AP: Aruba AP with USB port.

  Check the energy requirements of our Aruba AP to properly operate the USB port.

AP model	USB port (5W)	IPM feature	802.3af (class 3)	802.3at (class 4)	802.3bt (class 5+)	DC power	AC power
AP-303	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-303P	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-304/305	yes	yes	disabled	OK	OK	OK	not supported
AP-314/315	yes	yes	disabled	OK	OK	OK	not supported
AP-324/325	yes	no	disabled	OK	OK	OK	not supported
AP-334/335	yes	yes	disabled	disabled	disabled	OK	not supported
AP-344/345	yes	yes	disabled	disabled	disabled	OK	not supported
AP-504/505	yes	yes	disabled	OK	OK	OK	not supported
AP-514/515	yes	yes	disabled	OK	OK	OK	not supported
AP-534/535	yes	yes	not supported	disabled	OK	OK	not supported
AP-555	yes	yes	not supported	disabled	OK	OK	not supported
AP-203H	no	no	no USB port	no USB port	no USB port	not supported	not supported
AP-303H	yes	yes	disabled	disabled when P	disabled when P	OK	not supported
AP-503H	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-505H	yes	yes	disabled	disabled when P	OK	OK	not supported
AP-203R	yes	no	not supported	not supported	not supported	not supported	OK

- Aruba OS: version **8.10.0.0** or newer (most likely requires update to latest).
- EnOcean USB Stick: USB 300, USB 300U, USB 500 or USB 500U

3.2.2 Step 1: Connect to ArubaOS

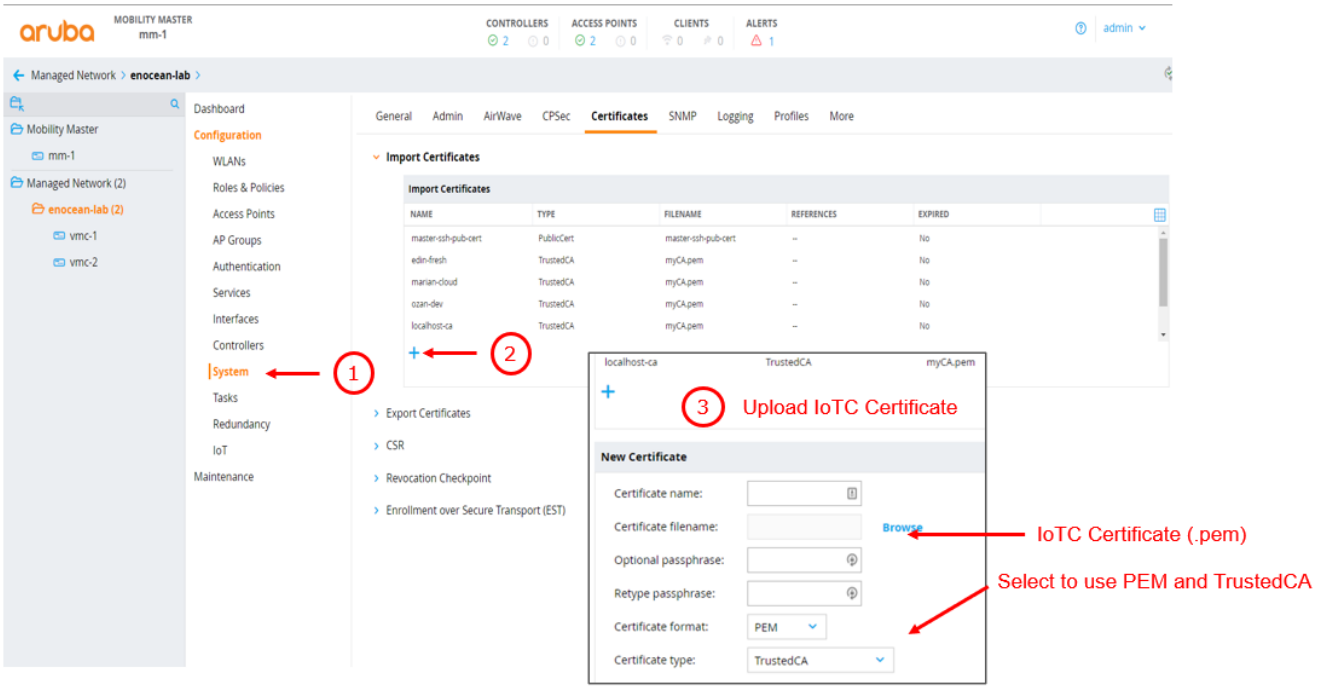
Log into the web-based management page for ArubaOS.

3.2.3 Step 2: Installing Trusted CA Certificates

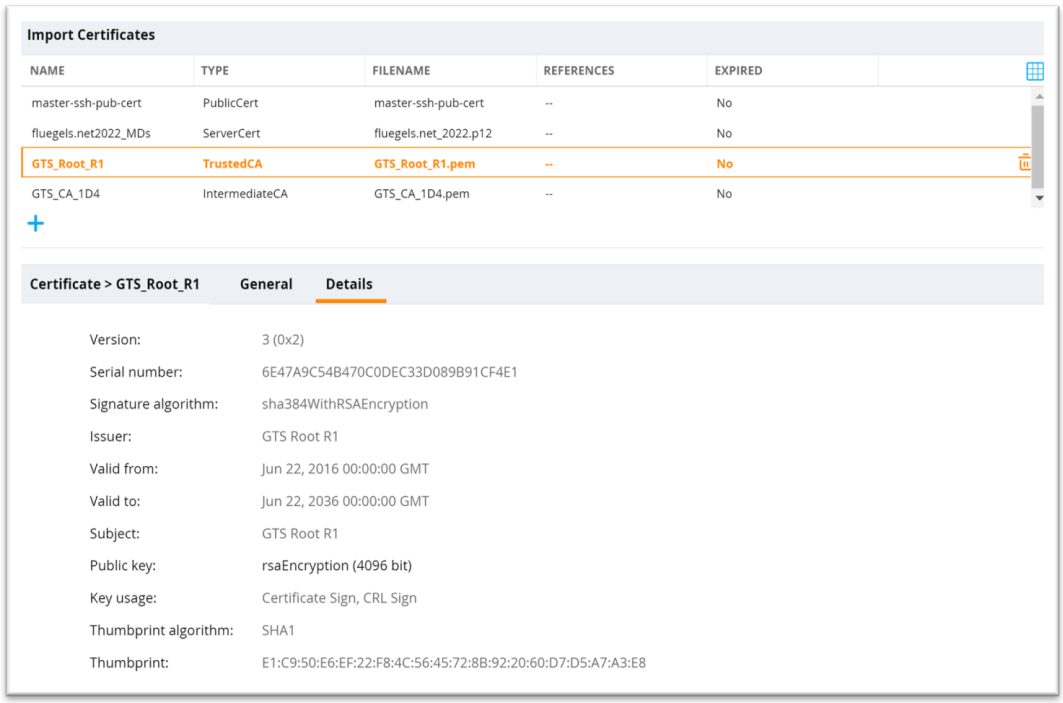
Note

When using Public Validated Certificates with Aruba Controller, the complete certificate path (root + intermediate CAs) must be installed as **separate files**. The server and additional certificates files could be downloaded from https://www.ssllabs.com/ssltest/.

- In your managed network tab select **Configuration -> System -> Certificates** Click the **+** to upload a new certificate. select Trusted CA as **certificate type** and **x509(.pem .cer or .crt)** as Certificate format.



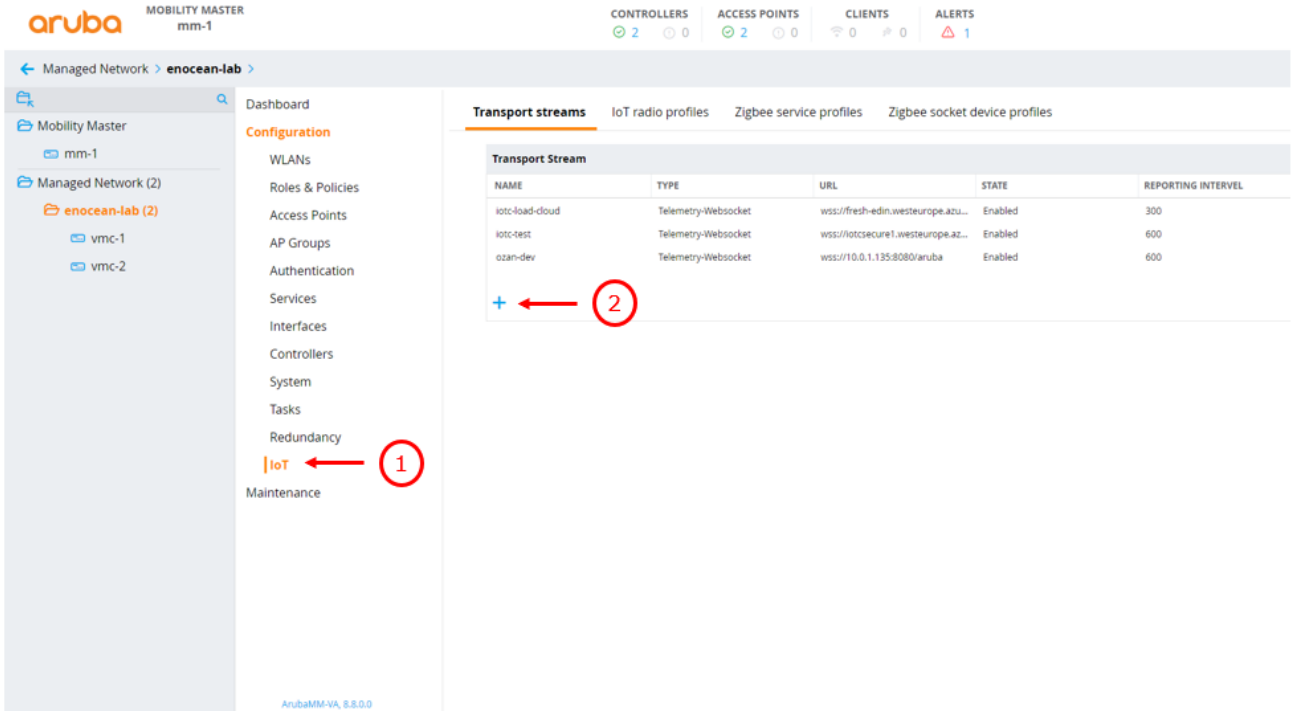
- Click **Upload Certificate** to save your settings.
- Verify that the certificate is shown on the certificates list.

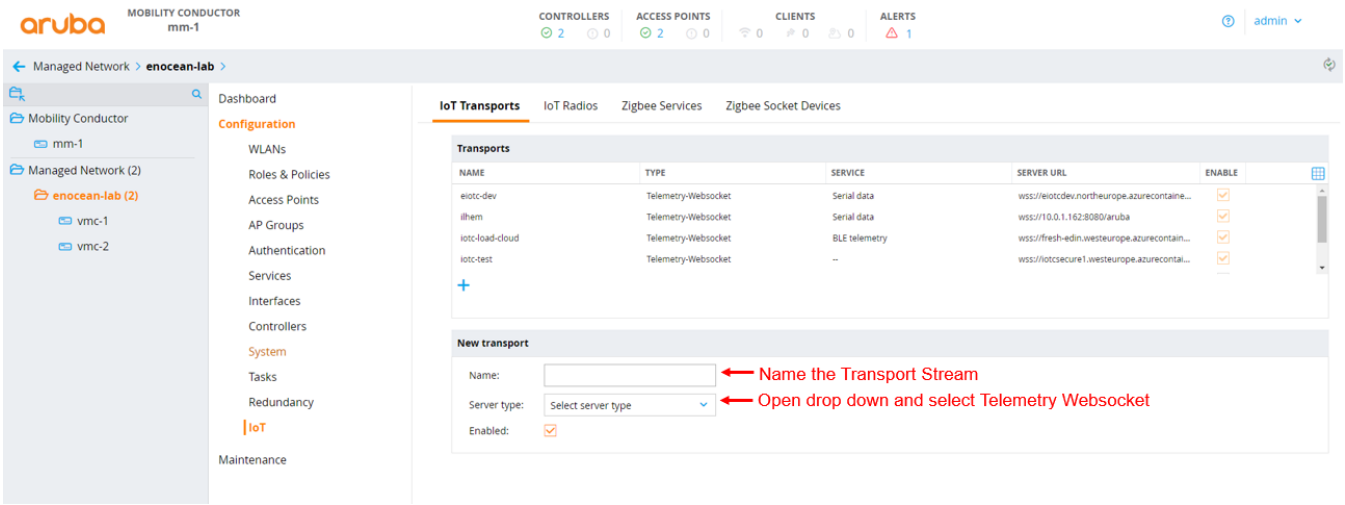


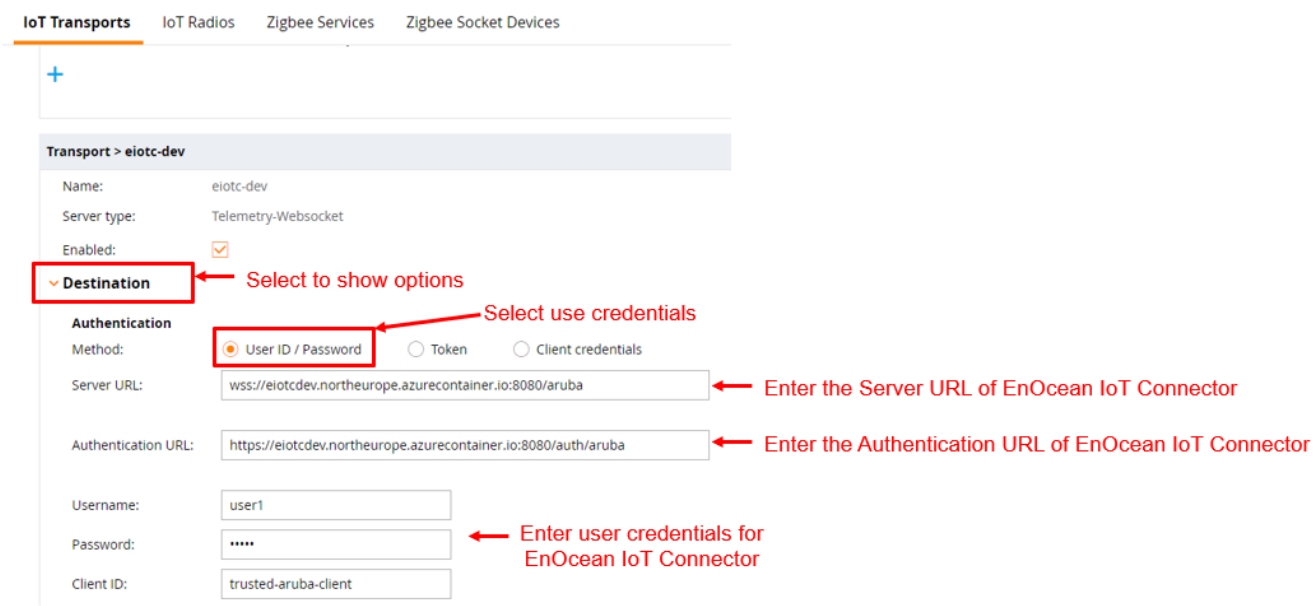
3.2.4 Step 3: IoT transport configuration

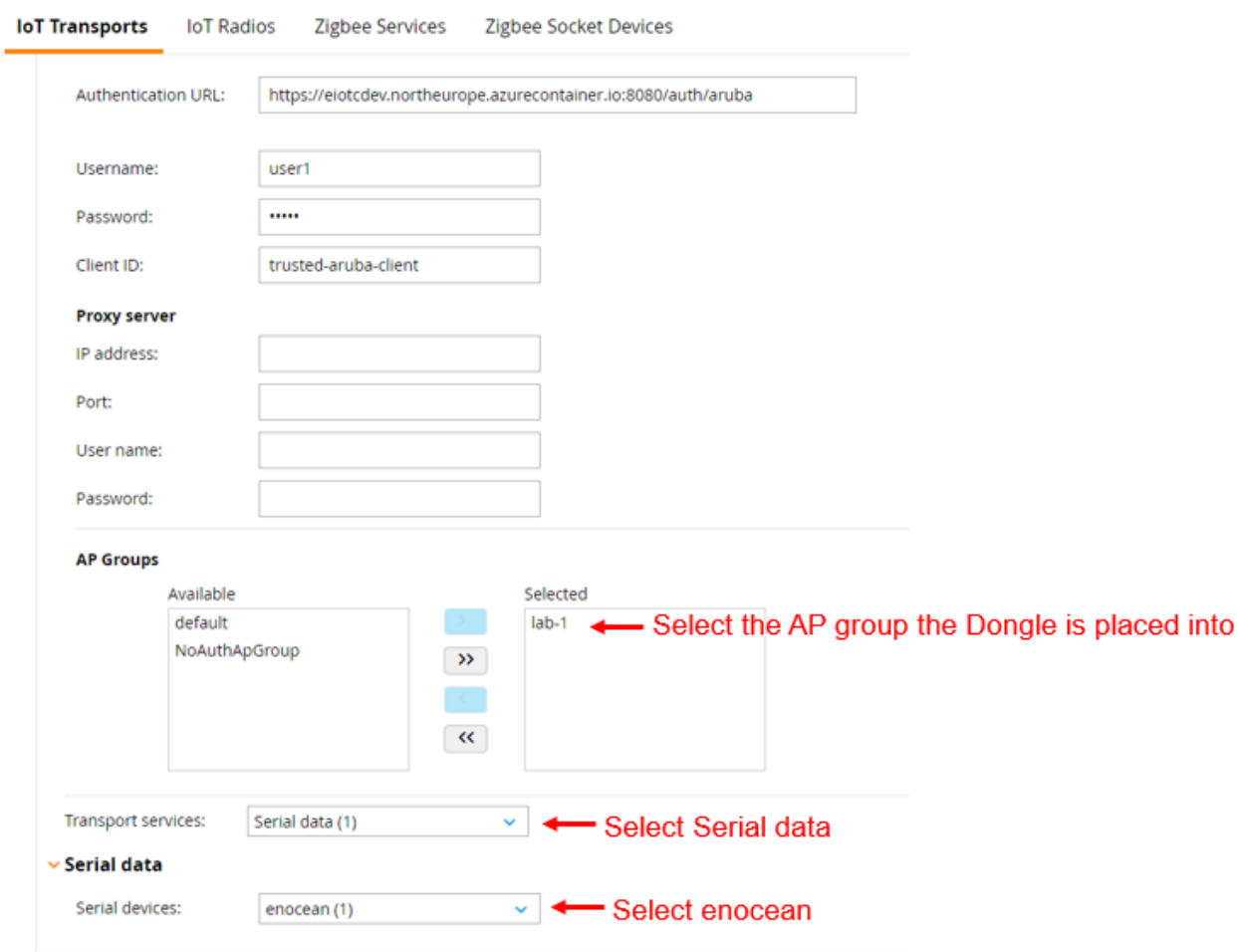
- In Aruba managed network select **Configuration** -> **IoT** -> **Transport streams** then add a new transport using the **+** icon.

  Enter the following information in the IoT transport tab:

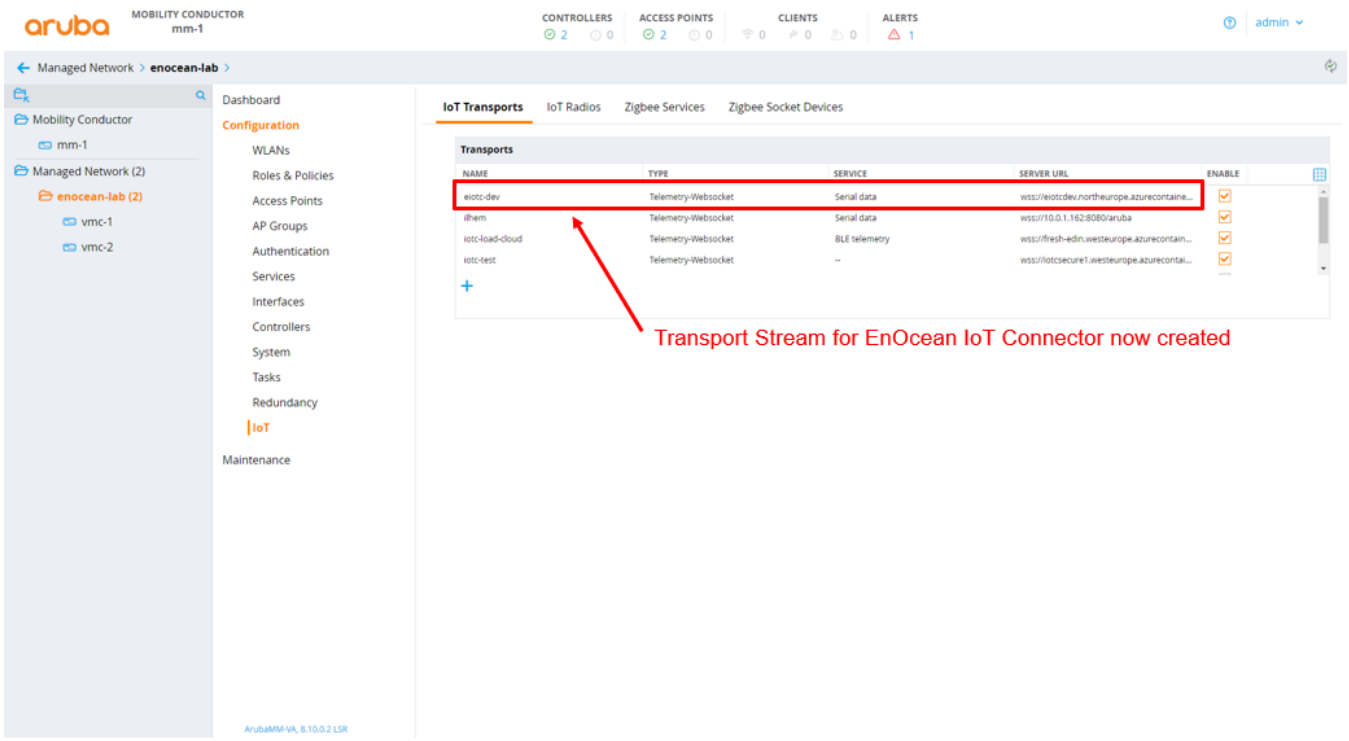






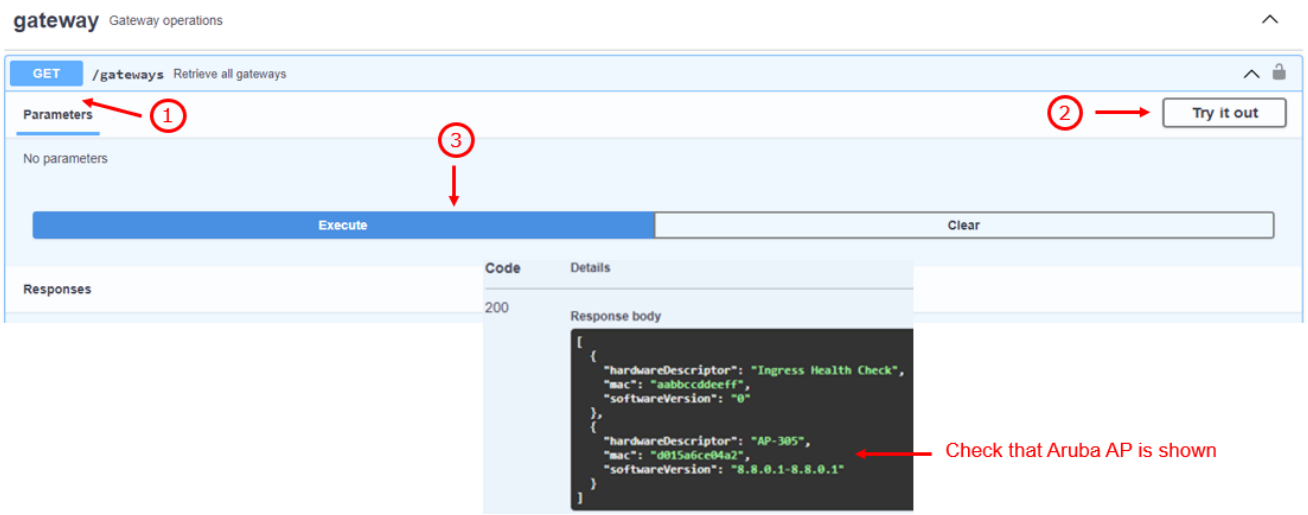


- Check that you transport stream is enabled:



3.2.5 Step 4: Verify that your Gateway is connected

Login to EnOcean IoT Connector and check gateway status using GET/gateways API



3.3 Configuration using Aruba Central

3.3.1 Required Hardware and Software

- Aruba AP: Aruba AP with USB port.

  Check the energy requirements of our Aruba AP to properly operate the USB port.

AP model	USB port (5W)	IPM feature	802.3af (class 3)	802.3at (class 4)	802.3bt (class 5+)	DC power	AC power
AP-303	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-303P	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-304/305	yes	yes	disabled	OK	OK	OK	not supported
AP-314/315	yes	yes	disabled	OK	OK	OK	not supported
AP-324/325	yes	no	disabled	OK	OK	OK	not supported
AP-334/335	yes	yes	disabled	disabled	disabled	OK	not supported
AP-344/345	yes	yes	disabled	disabled	disabled	OK	not supported
AP-504/505	yes	yes	disabled	OK	OK	OK	not supported
AP-514/515	yes	yes	disabled	OK	OK	OK	not supported
AP-534/535	yes	yes	not supported	disabled	OK	OK	not supported
AP-555	yes	yes	not supported	disabled	OK	OK	not supported
AP-203H	no	no	no USB port	no USB port	no USB port	not supported	not supported
AP-303H	yes	yes	disabled	disabled when P	disabled when P	OK	not supported
AP-503H	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-505H	yes	yes	disabled	disabled when P	OK	OK	not supported
AP-203R	yes	no	not supported	not supported	not supported	not supported	OK

- Aruba OS: version **8.10.0.0** or newer (most likely requires update to latest).
- EnOcean USB Stick: USB 300, USB 300U, USB 500 or USB 500U

3.3.2 Step 1: Connect to Aruba Central

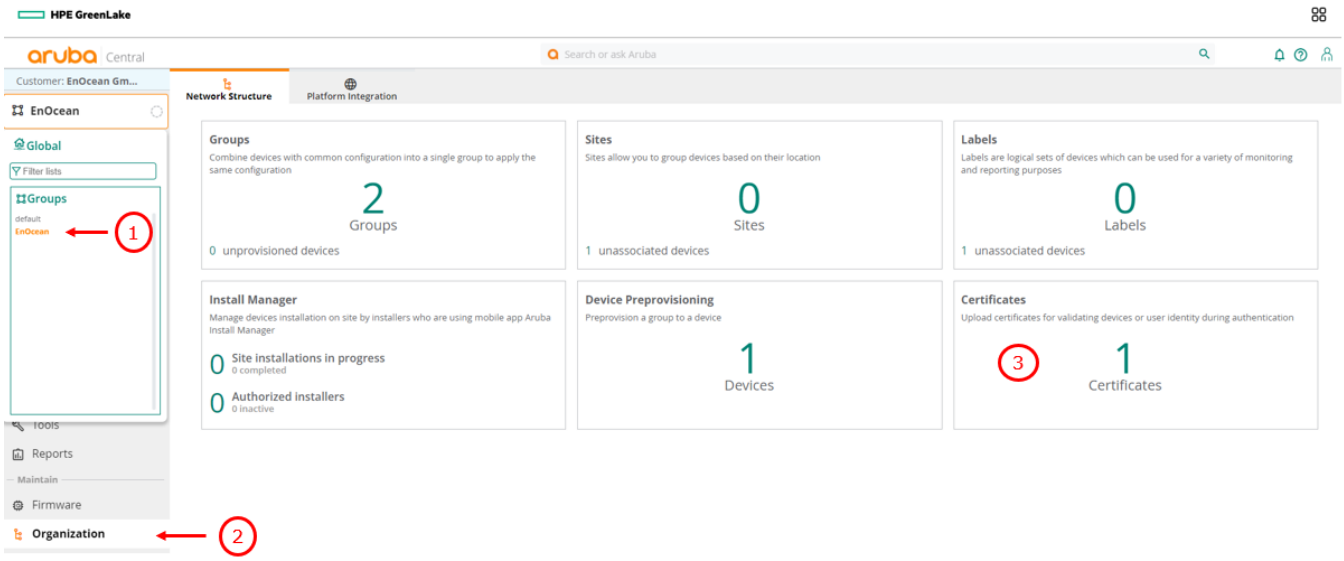
Log into the web-based management page for Aruba Central.

3.3.3 Step 2: Installing Trusted CA Certificates

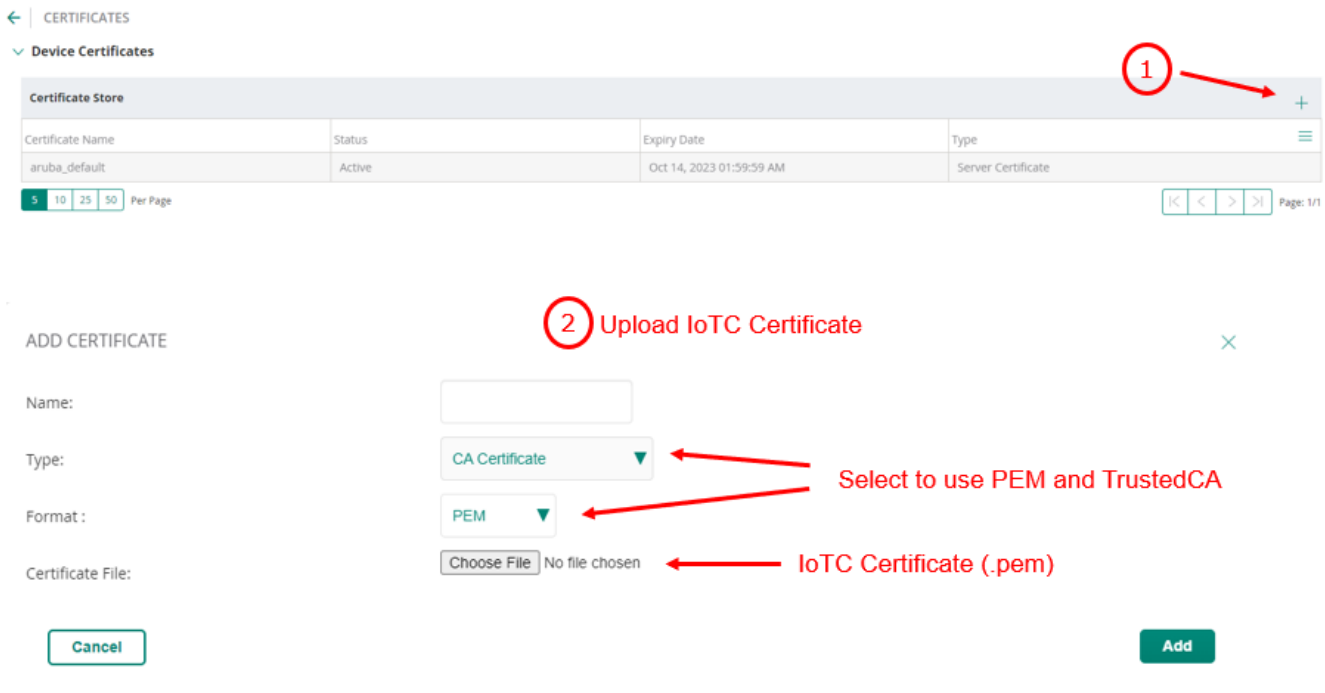
Note

When using Public Validated Certificates with Aruba Central, the complete certificate path (root + intermediate CAs) must be installed as **one file**.

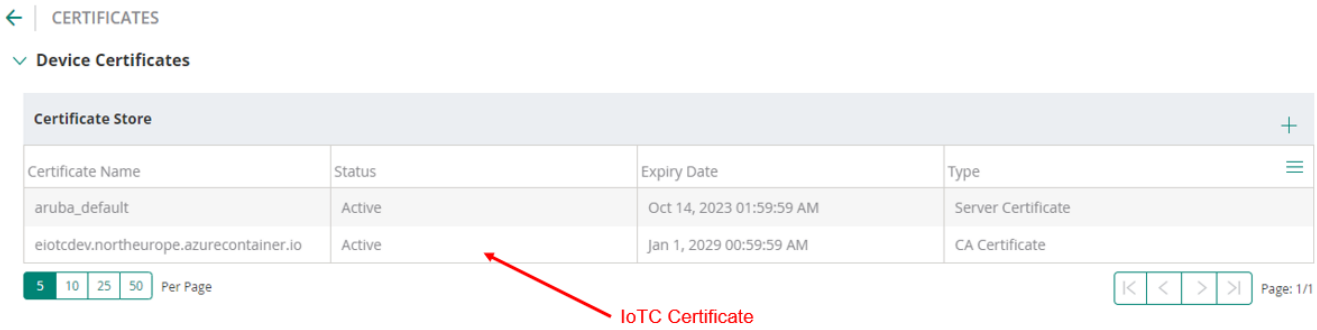
- Upload your .pem certificate file From **Your group** -> **Organization** -> **Certificates**.



- Select **CA certificate** as **certificate type** and **PEM** as **Certificate format** then choose your certificate .pem file. To create the certificate file chain for public validated certificates follow the guide : Using chained certificate.

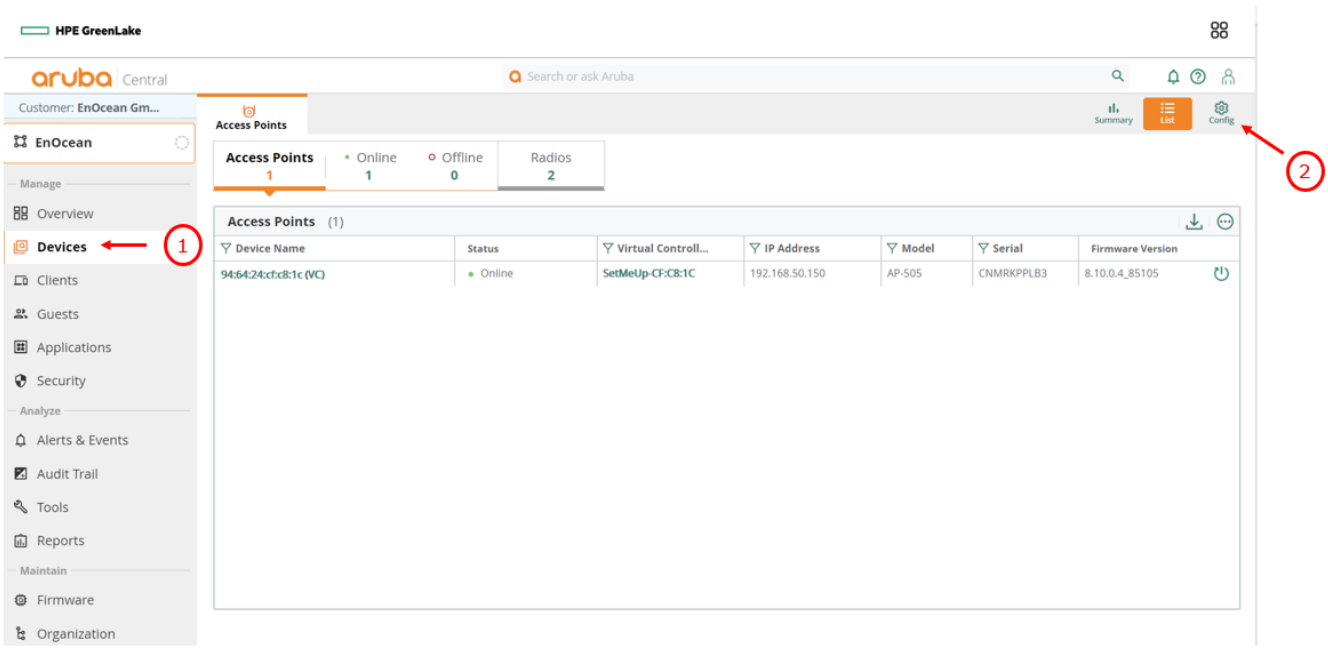


- Click **Add** to save your settings.
- Verify the certificate is shown on the certificate list.

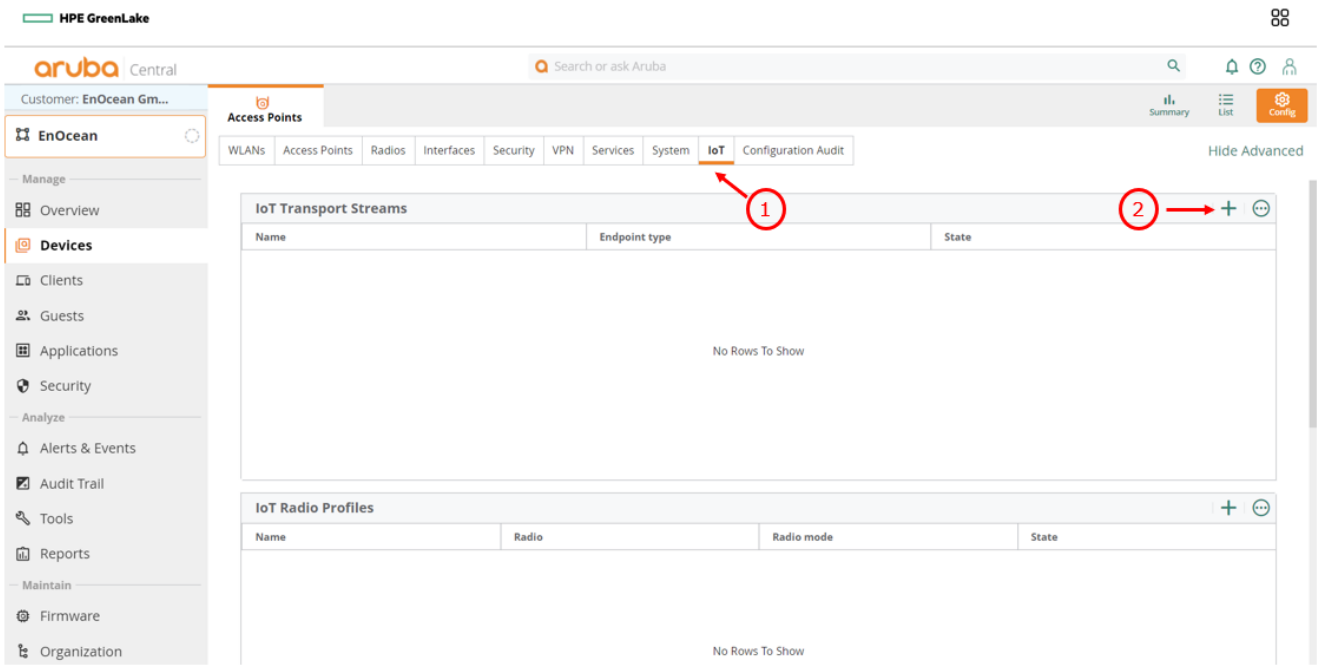


3.3.4 Step 3: IoT transport configuration

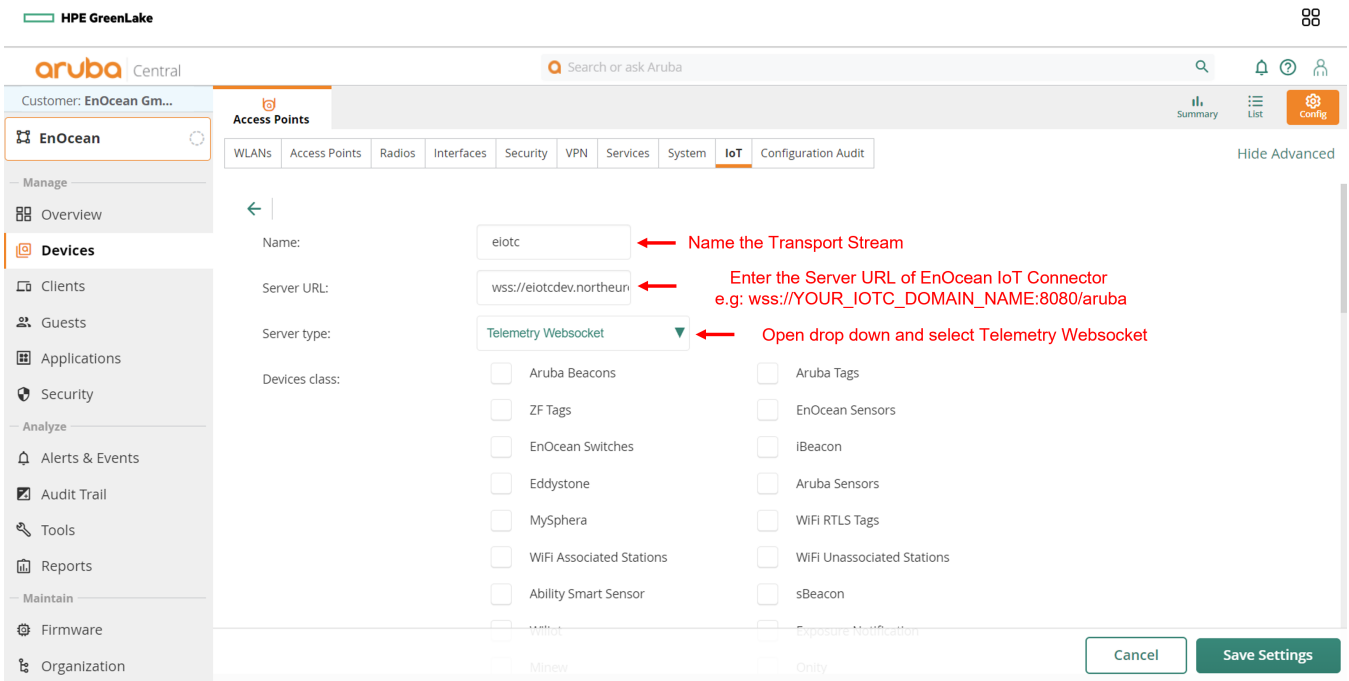
- In your Aruba Central group select **Devices** -> **Config**:

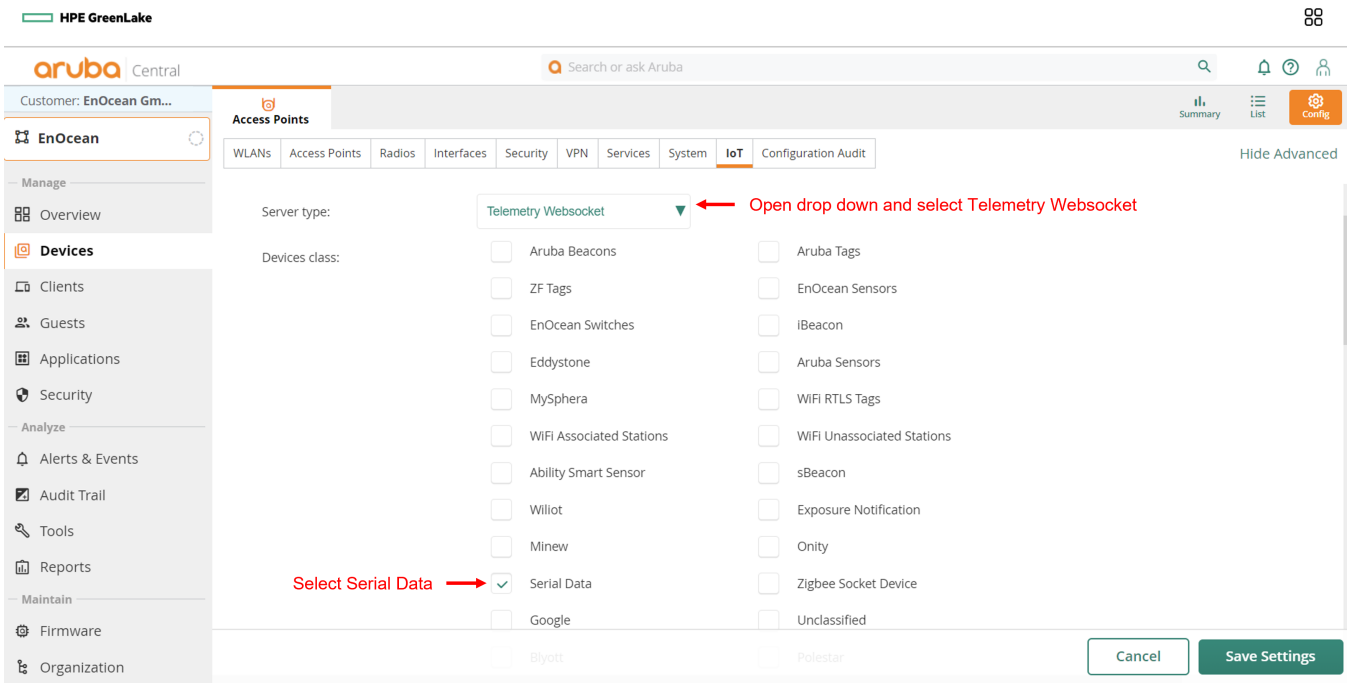


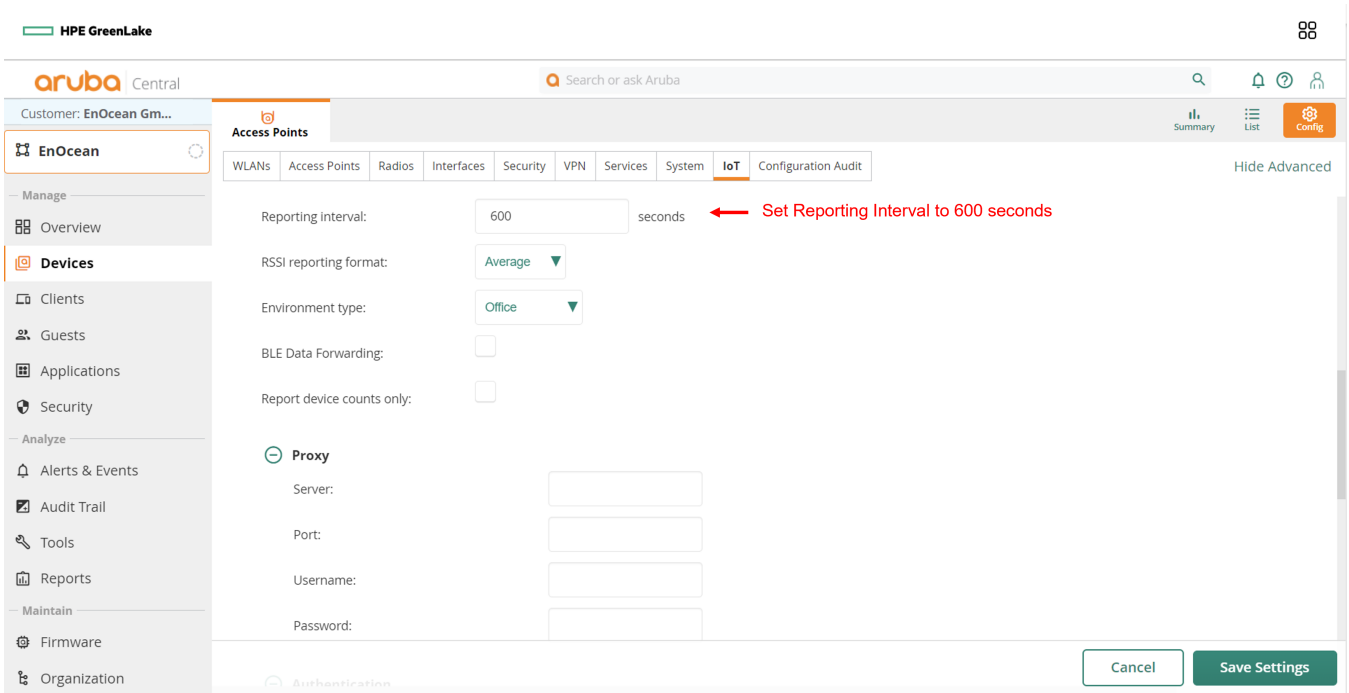
- select **IoT** then add a new IoT transport stream using the **+** icon:

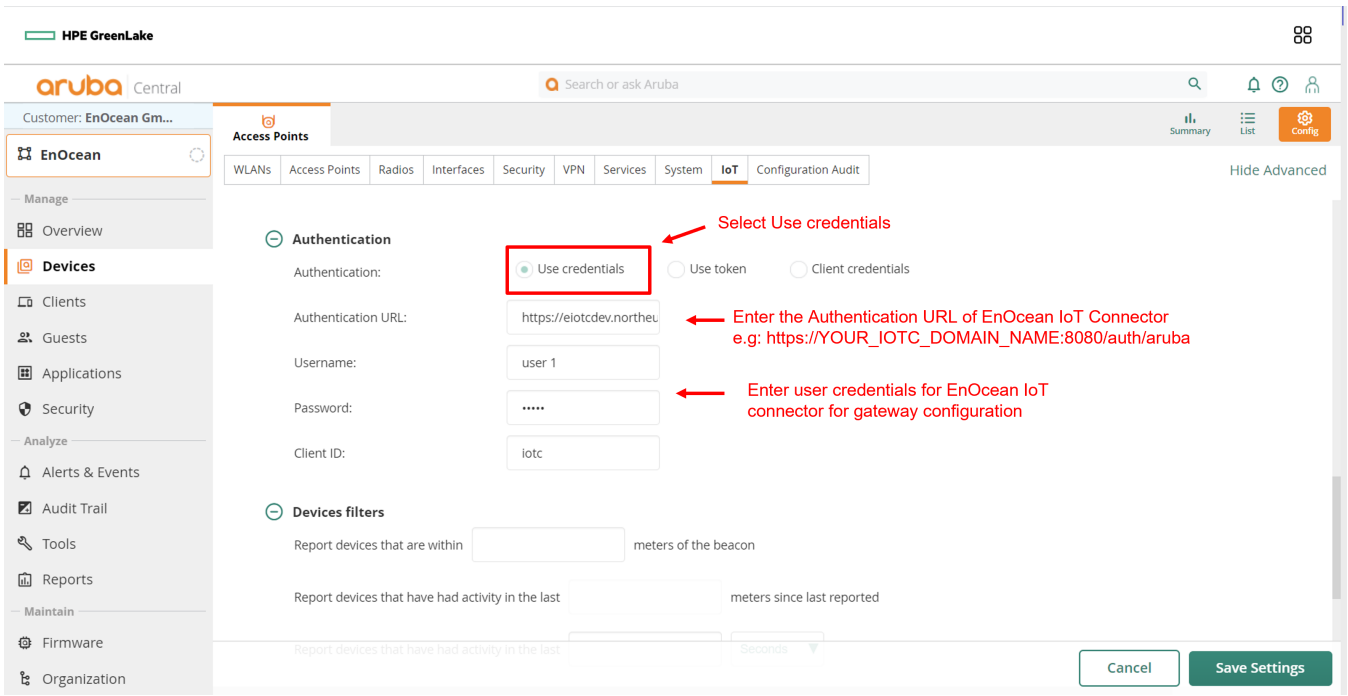


- Enter the following information in the IoT transport tab:



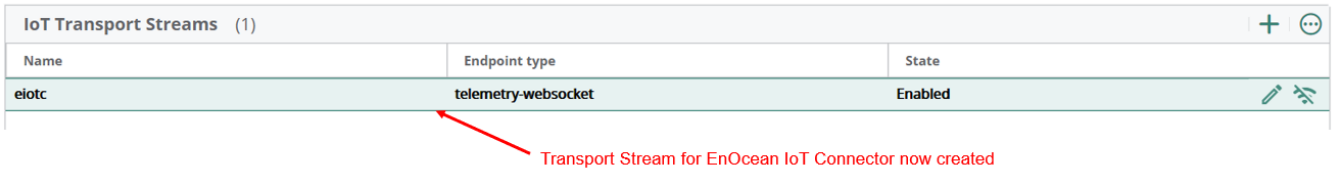






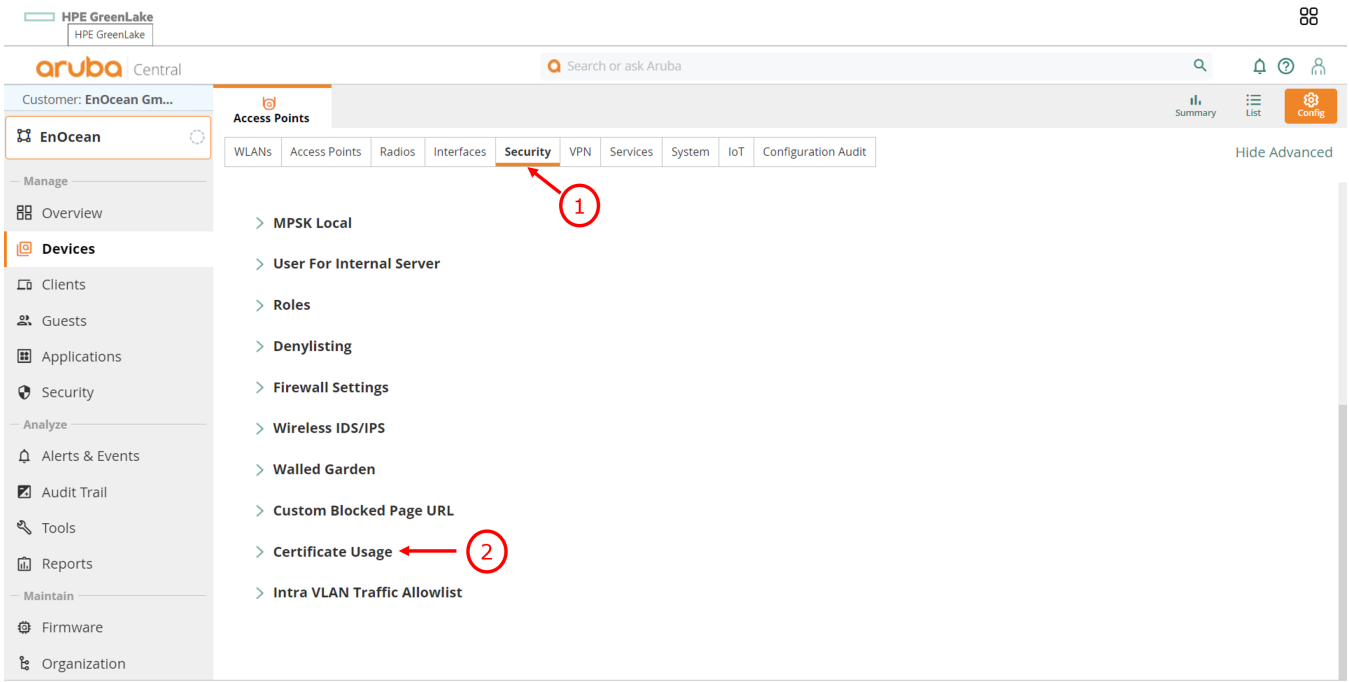
Click **Save settings** to save you configuration.

- Check that you transport stream is enabled:

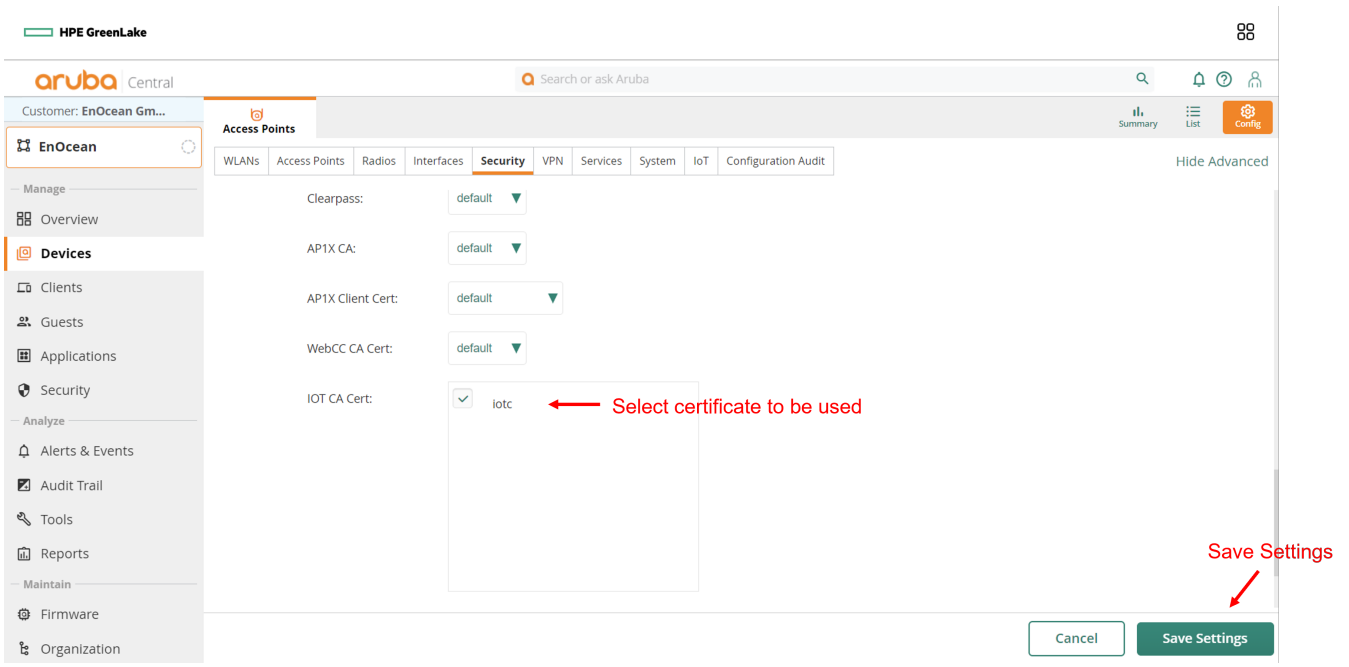


3.3.5 Step 4: Activating the certificate

- Under **Access Points** click **Security** then **Certificate Usage**.

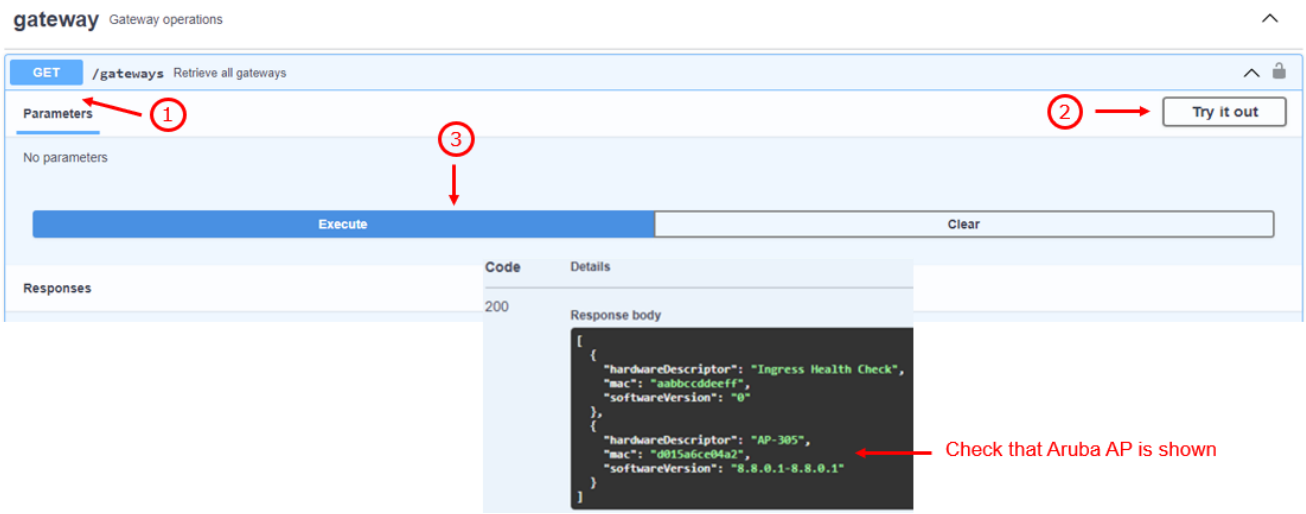


- Tick your certificate then click **Save Settings**.



3.3.6 Step 5: Verify that your Gateway is connected

Login to EnOcean IoT Connector and check gateway status using GET/gateways API



3.4 Notes for Aruba APs configuration through CLI

3.4.1 Required Hardware and Software

- Aruba AP: Aruba AP with USB port.

  Check the energy requirements of our Aruba AP to properly operate the USB port.

AP model	USB port (5W)	IPM feature	802.3af (class 3)	802.3at (class 4)	802.3bt (class 5+)	DC power	AC power
AP-303	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-303P	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-304/305	yes	yes	disabled	OK	OK	OK	not supported
AP-314/315	yes	yes	disabled	OK	OK	OK	not supported
AP-324/325	yes	no	disabled	OK	OK	OK	not supported
AP-334/335	yes	yes	disabled	disabled	disabled	OK	not supported
AP-344/345	yes	yes	disabled	disabled	disabled	OK	not supported
AP-504/505	yes	yes	disabled	OK	OK	OK	not supported
AP-514/515	yes	yes	disabled	OK	OK	OK	not supported
AP-534/535	yes	yes	not supported	disabled	OK	OK	not supported
AP-555	yes	yes	not supported	disabled	OK	OK	not supported
AP-203H	no	no	no USB port	no USB port	no USB port	not supported	not supported
AP-303H	yes	yes	disabled	disabled when P	disabled when P	OK	not supported
AP-503H	no	no	no USB port	no USB port	no USB port	no USB port	not supported
AP-505H	yes	yes	disabled	disabled when P	OK	OK	not supported
AP-203R	yes	no	not supported	not supported	not supported	not supported	OK

- Aruba OS: version **8.10.0.0** or newer (most likely requires update to latest).
- EnOcean USB Stick: USB 300, USB 300U, USB 500 or USB 500U

3.4.2 Adding root certificates

By default the Aruba APs won't be able to connect to the IoT connector using a self-signed certificate. To fix this, it is possible to add an additional certificate by following these steps:

1. Log in into the AP's admin portal.
2. Go to the *Maintenance Section*.
3. Navigate to the *Certificates* sub-menu.
4. Click on *Upload New Certificate*.
5. Choose your root certificate, type in a name, select *Trusted CA* and click *Upload Certificate*.

3.4.3 Configure Aruba AP to forward data to the IoTC

It is highly recommended to set-up the IoT Transport profile on Aruba AP through SSH.

Login into the AP using the same credentials from the web interface:

```
$ ssh <yourUser>@<accesspointIP>
<youruser>@<accesspointIP>s password: <enter password>
```

Replace `yourUser`, `accesspointIP` with your AP's credential's & IP-Address.

After login:

```
show tech-support and show tech-support supplemental are the two most useful outputs to collect for any kind of troubleshooting session.

aa:bb:cc:dd:ee:ff# configure terminal
We now support CLI commit model, please type "commit apply" for configuration to take effect.
aa:bb:cc:dd:ee:ff (config) # iot transportProfile myProfile
```

Replace `myProfile` with your desired profile name.

Now configure the profile:

Aruba OS 8.8.0.0 and newer Aruba OS 8.10.0.0

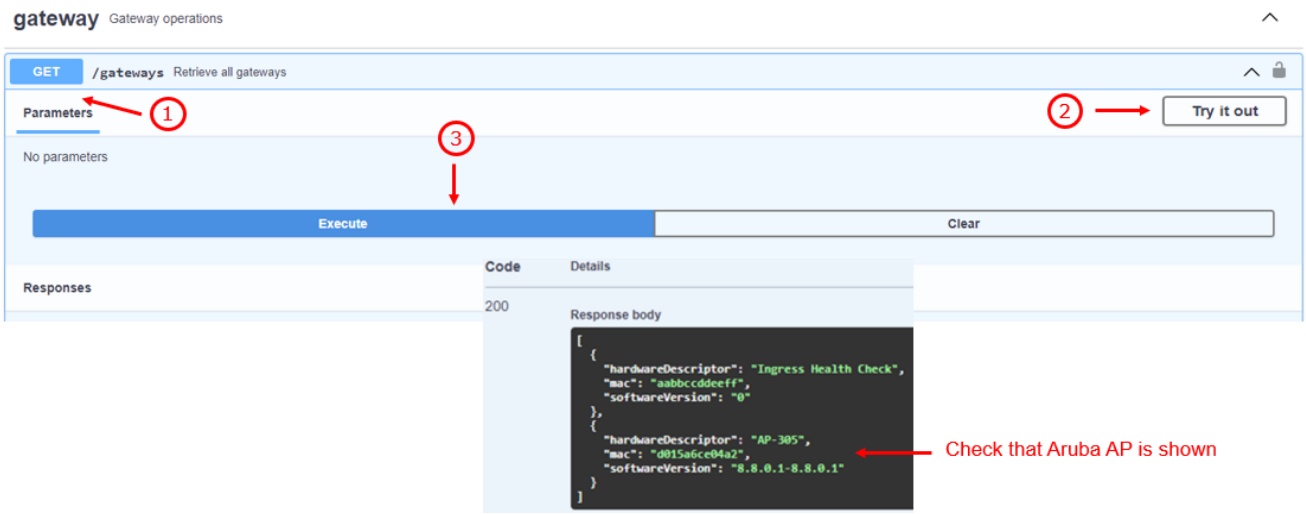
```
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # endpointType telemetry-websocket
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # endpointURL wss://myiotconnector:8080/aruba
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # payloadContent serial-data
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # authenticationURL https://myiotconnector:8080/auth/aruba
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # transportInterval 30
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # authentication-mode password
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # username <aruba_username set using IOT_GATEWAY_USERNAME>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # password <aruba_password set using IOT_GATEWAY_PASSWORD>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # endpointID 1111
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # end
aa:bb:cc:dd:ee:ff# commit apply
committing configuration...

aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # endpointType telemetry-websocket
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # endpointURL wss://myiotconnector:8080/aruba
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # payloadContent serial-data
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # authenticationURL https://myiotconnector:8080/auth/aruba
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # transportInterval 30
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # username <aruba_username set using IOT_GATEWAY_USERNAME>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # password <aruba_password set using IOT_GATEWAY_PASSWORD>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # end
aa:bb:cc:dd:ee:ff# commit apply
committing configuration...
```

Then activate the profile:

```
aa:bb:cc:dd:ee:ff # configure terminal
We now support CLI commit model, please type "commit apply" for configuration to take effect.
aa:bb:cc:dd:ee:ff (config) # iot useTransportProfile myProfile
aa:bb:cc:dd:ee:ff (config) # end
aa:bb:cc:dd:ee:ff # commit apply
committing configuration...
configuration committed.
```

3.4.4 Verify that your Gateway is connected

Login to EnOcean IoT Connector and check gateway status using GET/gateways API



3.5 Notes for Aruba APs Debugging & Troubleshooting

In case the Aruba AP (instant) is not connected to the IoTC i.e. the device is not listed in the gateway list or no EnOcean telegrams are visible on the egress of the IoTC. Try the following steps. Please consider that the commands syntax might change with new Aruba OS releases. The commands were tested with Aruba OS 8.8.x.

1. Show the IoT configuration. Get show and confirm the showed information correspond with the inputs provided before.

```
aa:bb:cc:dd:ee:ff # show iot transportProfile myProfile
```

2. Show & check connected USB devices. Example output is attached. For proper communication an `EnOcean USB` device must be connected to the AP.

```
aa:bb:cc:dd:ee:ff # show usb devices

USB Device Info
---------------
DeviceID  APMac Vendor ID  Product ID  Manufacturer  Product             Version  Serial    Class  Device   Driver    Uptime
--------  ---------------  ----------  ------------  -------             -------  -------   -----  ------   ------    ------
d3adas..  aa:.. 0403       6001        EnOcean GmbH  EnOcean USB 300 DC  2.00     FT55W4A9  tty    ttyUSB0  ftdi_sio  24m34s
```

3. Check the configured IoT Configuration status. `...` represents omitted information.

```
aa:bb:cc:dd:ee:ff # show ap debug ble-relay iot-profile

ConfigID                              : xx

--------------------------Profile[myProfile]--------------------------

authenticationURL                     : ...
serverURL                             : ...
...
--------------------------
TransportContext                      : Connection Established
Last Data Update                      : 2021-06-14 15:01:20
Last Send Time                        : 2021-06-14 15:01:19
TransType                             : Websocket
```

If `TransportContext` displays an error message, please follow up on the meaning of the message. Please consider it can take few seconds to build the connection.

4. To check if EnOcean telegrams are being received and forwarded via the established connection please use the following command and watch if the `Websocket Write Stats` increases after a known EnOcean telegram transmission. Also check for changes in `Last Send Time` . `...` represents omitted information.

```
aa:bb:cc:dd:ee:ff # show ap debug ble-relay report

--------------------------Profile[myProfile]--------------------------

WebSocket Connect Status              : Connection Established
WebSocket Connection Established      : Yes
Handshake Address                     : ...
Refresh Token                         : Not Configured
Access Token                          : ...
Access Token Request by Client at     : 2021-06-14 14:18:32
Access Token Expire at                : 2021-06-14 15:18:32
Location Id                           : ...
Websocket Address                     : ...
WebSocket Host                        : ...
WebSocket Path                        : ...
Vlan Interface                        : Not Configured
Current WebSocket Started at          : 2021-06-14 14:18:42
Web Proxy                             : NA
Proxy Username&password               : NA, NA
Last Send Time                        : 2021-06-14 14:30:35
Websocket Write Stats                 : 8278 (1454156B)
Websocket Write WM                    : 0B (0)
Websocket Read Stats                  : 0 (0B)
```

5. If there are any issues you can get additional log messages by running the following command.

```
aa:bb:cc:dd:ee:ff #  show ap debug ble-relay ws-log myProfile
```

If you struggle with the connection of an Instant Aruba AP please contact the Aruba technical support.

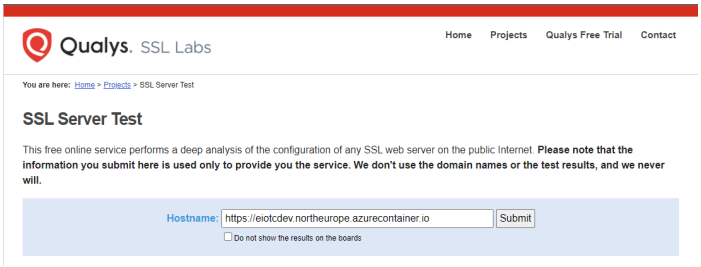
For debugging enterprise connected Aruba AP, via an Aruba Controller please use these commands instead.

```
#Show profiles
show iot transportProfile myProfile

#Show USB devices
show ap usb-device-mgmt all

#Show status and report
show ble_relay iot-profile
show ble_relay report <iot-profile-name>

#Show Log
show ble_relay ws-log <iot-profile-name>
```

3.6 Using chained certificate

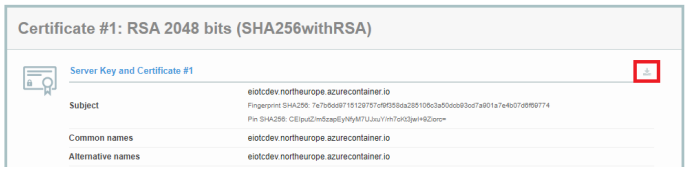
1. Get certificates files:

   In the browser navigate to https://www.ssllabs.com/ssltest/ and paste the URL of your IOTC EnOcean IoT Connector then click **Submit**



2. Create certificate chain:

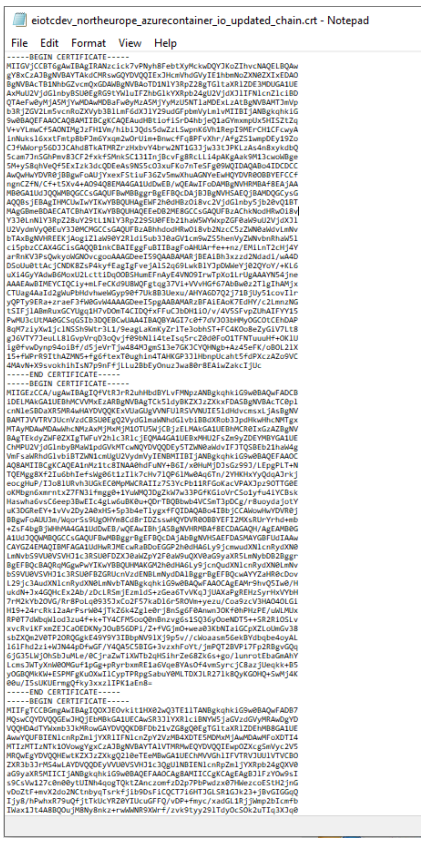
   Download the Server Certificate



   Download the Addtional Certificates

   Create the Certificate Chain File

   Combine the obtained server certificate and the additional certificates file into a single file using a text editor, first in chain file should be your Server's certificate, second one should be the addtional certificates obtained in the previous step. The obtained certificate chain file should look like below:



3. Install the file in Aruba Instant.

4. API Reference

4.1 Swagger API

This is the full API Documentation done in Swagger.

> **Note**
>
> The `Try it out` function only works with a deployed IoTC instance.

4.2 MQTT API

4.2.1 MQTT Topics

Using the MQTT end-point publishes these topics:

PATH	Description
`sensor/[ID]/telemetry`	EnOcean device telemetry of a specific [ID]. Publishing is done every time a valid telegram was processed. Payload consists of JSON file described in here.
`sensor/[ID]/meta/event/`	Event information of a specific [ID]. Publishing is done with a specific event. Reference of possible events and content of JSON files cab be found here.
`sensor/[ID]/meta/stats/`	Statical information about traffic of a specific [ID]. Publishing is done in predefined time internal e.g. 10 min. Interval can be configured. This feature is optional. Configuration is done via ENV variables. Published JSON Payload can be reviewed here.
`gateway/[MAC]/meta/event/`	Event information of a specific gateway [MAC]. Publishing is done with a specific event. Reference of possible events and more can be found here.
`gateway/[MAC]/meta/stats/`	Statical information about traffic of a specific gateway [MAC]. Publishing is done in predefined time internal e.g. 10 min. Interval can be configured. This feature is optional. Configuration is done via ENV variables. Published JSON Payload can be reviewed here.
`system/health/`	Statical information about IoTC health status. Publishing is done in predefined time internal e.g. 10 min. Interval can be configured via ENV variables. Published JSON Payload can be reviewed here.
`device/[ID]/rpc/command`	Topic for queueing new commands. Detailed information can be found here
`device/[ID]/rpc/cancel`	Topic for deleting waiting commands. Detailed information can be found here
`device/[ID]/rpc/result`	Topic for results of requests. Detailed information can be found here

4.2.2 JSON Output Format

All Schemas of all JSON outputs can be found in the download section.

Note

All timestamps in IoTC are in the Unix epoch (or Unix time or POSIX time or Unix timestamp). It is the number of seconds that have elapsed since January 1, 1970. It can be converted into human-readable version quite easy. e.g. use an online convertor.

`timestamp = 1624367607` equals to `GMT: Tuesday, June 22, 2021 1:13:27 PM`

Sensor telemetry

Each output JSON consist of these sections:

- `sensor` - stored information about the sensor provided at onboarding via the API
- `telemetry` - information interpreted by the engine
- `data` - sensor data included in the message and encoded via the EEP
- `signal` - meta information about the sensor and encoded as signal telegram
- `meta/stats` - meta information about the message added by the engine
- `raw` - raw message information
- `rssi` - radio signal strength information. Important to track radio quality

TELEMETRY -> DATA

The data is included in a JSON file as `key-value` pairs following the EnOcean Alliance IP Specification. Example JSON outputs from selected devices are available below.

Multisensor CO2 sensor Switch Module

EnOcean IoT Multisensor


```
{
    "sensor": {
        "friendlyId": "Multisensor 1",
        "id": "04138bb4",
        "location": "Cloud center"
    },
    "telemetry": {
        "data": [{
            "key": "temperature",
            "value": 23.9,
            "unit": "°C"
        }, {
            "key": "humidity",
            "value": 29.0,
            "unit": "%"
        }, {
            "key": "illumination",
            "value": 67.0,
            "unit": "lx"
        }, {
            "key": "accelerationStatus",
            "value": "heartbeat",
            "meaning": "Heartbeat"
        }, {
            "key": "accelerationX",
            "value": -0.13,
            "unit": "g"
        }, {
            "key": "accelerationY",
            "value": 0.08,
            "unit": "g"
        }, {
            "key": "accelerationZ",
            "value": -0.97,
            "unit": "g"
        }, {
            "key": "contact",
            "value": "open",
            "meaning": "Window opened"
        }],
        "signal": [],
        "meta": {
            "stats": [{
                "egressTime": "1611927479.169171",
                "notProcessed": 0,
                "succesfullyProcessed": 6,
                "totalTelegramCount": 6
            }]
        }
    },
    "raw": {
        "data": "d29fce800863b502a620",
        "sender": "04138bb4",
        "status": "80",
        "subTelNum": 0,
        "destination": "ffffffff",
        "rssi": 77,
        "securityLevel": 0,
        "timestamp": "1611927479.166352"
    }
}
```



```
{
    "sensor": {
        "friendlyId": "co2_Hardware2",
        "id": "051b03c9",
        "location": "Hardware 2"
    },
    "telemetry": {
        "data": [{
            "key": "co2",
            "value": 627.45,
            "unit": "ppm"
        }, {
            "key": "learn",
            "value": "notPressed",
            "meaning": "Data telegram"
        }, {
            "key": "powerFailureDetected",
            "value": "False",
            "meaning": "Power failure not detected"
        }],
        "signal": [],
        "meta": {
            "stats": [{
                "egressTime": "1611927535.0731573",
                "notProcessed": 0,
                "succesfullyProcessed": 6,
                "totalTelegramCount": 6
            }]
        }
    },
    "raw": {
        "data": "a500005008",
        "sender": "051b03c9",
        "status": "01",
```

TELEMETRY -> SIGNAL

Selected devices from EnOcean transmit additionally to their data messages also messages about their internal states or events. This messages are known as signal telegrams. Signal telegrams include information about the:

- percentage of remaining energy available in the energy storage
- how much energy is provided via the energy harvester
- availability and status of a back up energy store
- for additional information see the signal telegrams specification and data sheet of your EnOcean product

Example of an energy `MID: 6` signal telegram is below:


```
{
    "sensor": {
        "friendlyID": "0413D759 D2-14-41 SIMU Multisensor",
        "id": "0413d759",
        "location": "Office 265",
        "eep": "d2-14-41",
        "customTag": ""
    },
    "telemetry": {
        "data": [],
        "signal": [{
            "key": "signalIdentifier",
            "value": "0x6",
            "meaning": "Energy status of device"
        }, {
            "key": "energy",
            "value": 56.0,
            "unit": "%"
        }],
        "meta": {
            "stats": [{
                "egressTime": "1638876910.137704",
                "notProcessed": 0,
                "succesfullyProcessed": 6,
                "totalTelegramCount": 6
            }]
        }
    },
    "raw": {
        "uuid": "f521f37c-3a82-42cb-b1cc-c889e946cef3",
        "data": "d00638",
        "sender": "0413d759",
        "status": 128,
        "subTelNum": 1,
        "destination": "ffffffff",
        "rssi": -64,
        "securityLevel": 0,
        "timestamp": "1638876903",
        "subTimestamp": 0,
        "subtelegrams": []
    }
}
```


TELEMETRY -> META

The `meta` section is complementary to `data` and `signal`. The meta section includes the `stats` section as provided by the API for the referenced device. Additionally the egress timestamp is included.

Examples are visible with the above examples with `data` and `signal`.

RAW -> RSSI

The `raw` element includes the radio telegram Information as received by the IoTC. They are mostly included for tracking and debug purposes. The `rssi` is the only one of interest.

The `rssi` radio signal strength information provides important information about connectivity. We recommend to track it and raise and alarm if the level drops or changes significantly.

Sensor meta

EVENT

The IoT Connector provides important information about events that were detected in regard to the sensor status, data transmission or behavior.

There are these types of events:

Type	Event	Description
Security	`MAC_VALIDATION_ERROR`	A received message could not be authentificated with the included CMAC. This could be an indication for a security attack.
	`RLC_REPLAY`	A received message has a lower message sequence counter then the previous. This could e an indication for an replay attack.
	`DEVICE_SEND_NOW_UNSECURE`	A device which was onboarded as secure is now transmitting as non secure. This is an indication of compromision the set security level, possible attack.
Health	`FIRST_TIME_SEND`	An onboarded device transmitted for the first time.
Processing	`EEP_DECODE_ERROR`	The receive message could not be decoded with the specified EEP. This is an indication for an corrupted radio message (if occuring on limited basis) or wrong specified EEP (if occuring pernament).
	`EEP_NOT_FOUND_ERROR`	The specified EEP of an device is not know to the IoTC. Please contact support in such case.

Example of an Health `FIRST_TIME_SEND` message is below:

```
{
  "sensor": {
    "friendlyID": "Multisensor 1",
    "id": "04138d23",
    "location": "Cloud center",
    "eep": "d2-14-41",
    "customTag": ""
  },
  "meta": {
    "events": {
      "security": [],
      "health": [
        {
          "code": "FIRST_TIME_SEND",
          "message": "First time send of device with id=04138d23."
        }
      ],
      "transcoding": []
    },
     "stats": {
         "timestamp": "1637770981"
    }
  }
}
```

STATS

The telegrams `stats` of individual EnOcean devices are posted periodically. This should indicate their operational status and additionally provide operational updates.

Example of an stats message is listed below:

```
{
  "sensor": {
    "friendlyID": "Multisensor 1",
    "id": "04138d23",
    "location": "Cloud center",
    "eep": "d2-14-41",
    "customTag": ""
  },
  "meta": {
     "stats": {
      "lastSeen": "1637827538",
      "notProcessed": 1,
      "succesfullyProcessed": 6,
      "totalTelegramCount": 0
      }
    }
  }
}
```

The content of the `stats` section corresponds to the response of the device telegram statics API request.

Gateway meta

EVENT

Selected AP (e.g. Aruba AP) transmits meta information about their internal states referenced as Gateway Health Updates. The content is similar to the console log messages.

The purpose of this message includes these two use cases: - Still-alive message from the gateway. Know the gateway is operation. - EnOcean USB Dongle information of the gateway. Know the USB Dongle is correctly operating.

Example of an meta event of gateways is listed below:

```
{
  "gateway_info": {
    "mac": "aabbccddeeff",
    "softwareVersion": "8.8.0.0",
    "hardwareDescriptor": "AP-505"
  },
  "stats": {
    "timestamp": "1639039720"
  },
  "usb_info": [
    {
      "usb_identifier": "ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd",
      "usb_health": "healthy"
    }
  ]
}
```

STATS

The telegrams `stats` of individual gateways are posted periodically. This should indicate their operational status and additionally provide operational updates.

Example of an stats message is listed below:

```
{
  "gateway_info": {
    "mac": "aabbccddeeff",
    "softwareVersion": "8.8.0.0",
    "hardwareDescriptor": "AP-505"
  },
  "stats": {
      "lastSeen": "1637827538",
      "notProcessed": 0,
      "succesfullyProcessed": 6,
      "totalTelegramCount": 6
  }

}
```

The content of the `stats` section corresponds to the response of the gateway telegram statics API request.

System health

IoTC periodically checks the status of the containers to validate correct operation of IoTC. After each check, a system health notification is sent on MQTT to notify the application of the IoTC status. The application should examine the health notification and notify the system administrator in case issues are observed. The application should also use the periodic health notification as a keep alive and expect issues if the health notifications are not received.

Example of the system health event is listed below:

```
{
  "gateway_info": [
    {
      "mac": "a1b2c3d4e5f6",
      "status": "healthy",
      "timestamp": "2022-05-18T09:25:33Z"
    }
  ],
  "system_health": {
    "api": "running",
    "ingress": "running",
    "integration": "running",
    "mqtt": "running",
    "redis": "running"
  }
}
```

4.3 API Usage

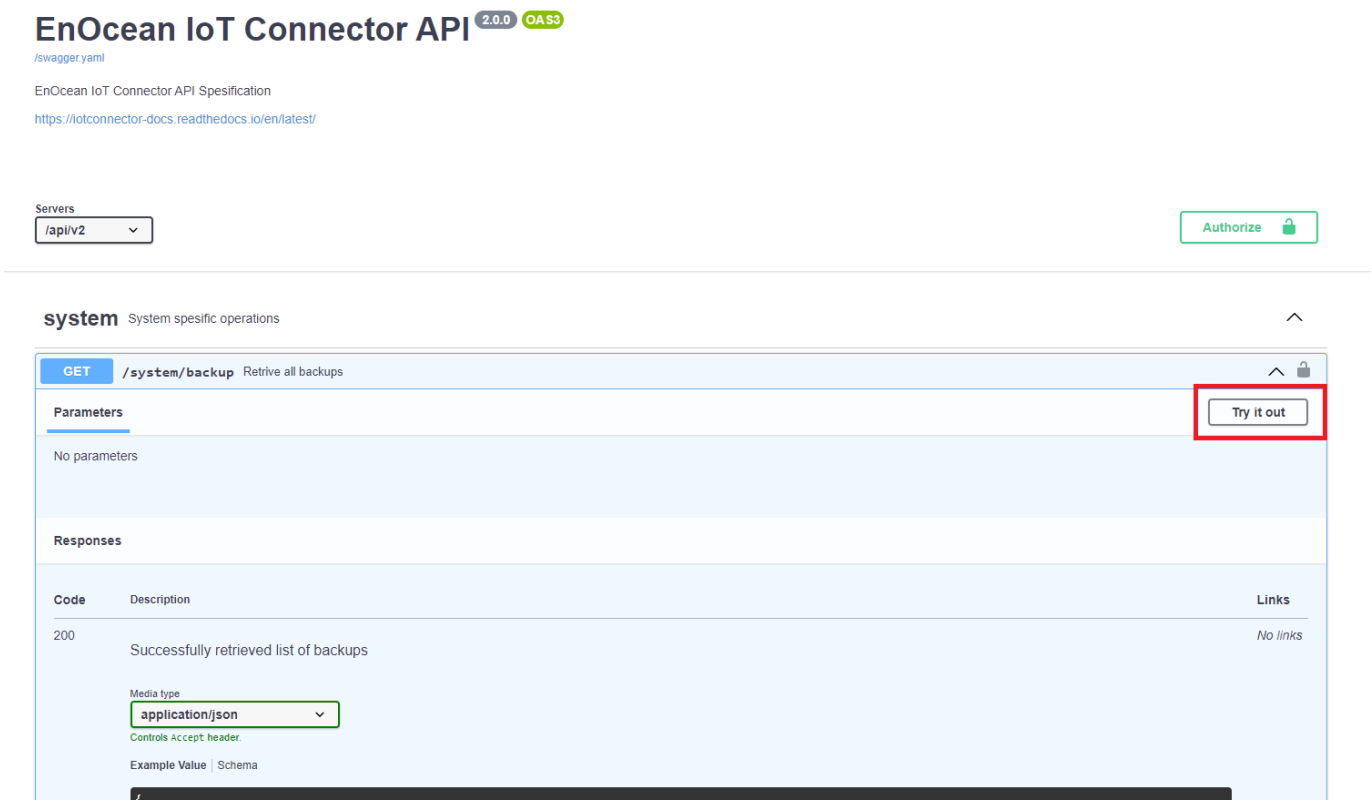
To use the Web UI follow these steps.

1. Opening the API url on a browser will display the API reference. The URL is `https://<hostname of the container group or IP address>:443`. Example: `https://192.167.1.1:443` or `https://myiotc.eastus.azurecontainer.io:443`

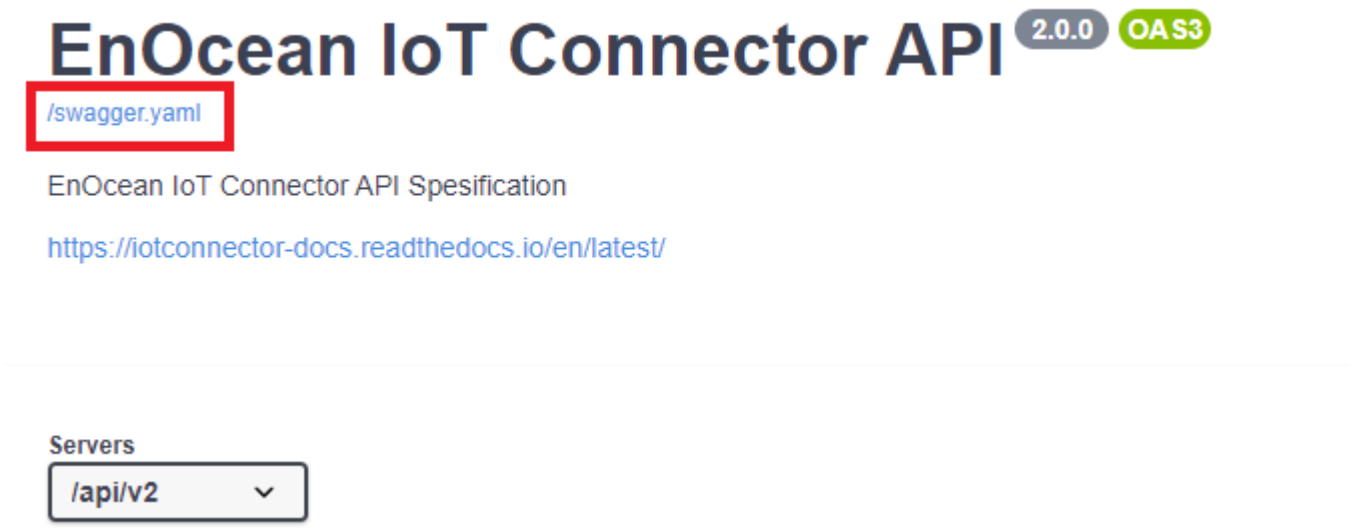
> **Note**
>
> If you used a self-signed certificate and did not add it to your browser you will see a warning, please continue according to your web browser.

2. Login using the `BASIC_AUTH_USERNAME` & `BASIC_AUTH_PASSWORD` you specified in environmental variables.

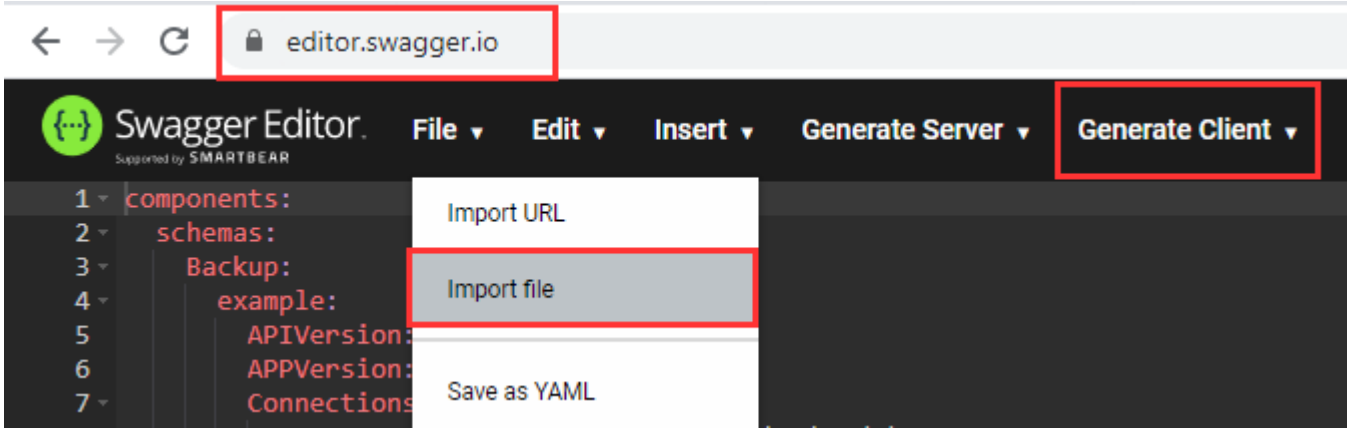
   You can use the `Try it out` function to execute any of the available commands for learning and debugging purposes.



   Download the API Specification as JSON



Go to the editor e.g. online here and generate your client code.



5. Integration

5.1 Enable TLS for MQTT

1. Add lines below to integration container's environment variables in docker-compose.yml

```
- MQTT_USE_TLS=True
- MQTT_USE_TLS_VERIFY=True
```

2. Add secrets to integration container in docker-compose.yml

```
secrets:
    - source: mqtt-ca-cert
      target: /etc/certs/mqtt-ca.crt
      mode: 400
    - source: mqtt-client-cert
      target: /etc/certs/mqtt-mqtt_client.crt
      mode: 400
    - source: mqtt-client-key
      target: /etc/certs/mqtt-mqtt_client.key
      mode: 400
```

3. Expose TLS port of MQTT broker adding line below to ports of mqtt container in docker-compose.yml

```
- "8883:8883"
```

4. Give TLS configuration to MQTT Broker using the provided configuration as volume to mqtt container in docker-compose.yml file.

```
volumes:
    - ./mqtt/config:/mosquitto/config/
```

5. Define certificate files on docker-compose.yml secrets to be used on 2.

```
mqtt-ca-cert:
    file: ./mqtt/certs/ca.crt   # Point your CA Certificate for MQTTS
mqtt-client-cert:
    file: ./mqtt/certs/client.crt  # Point your Client Certificate for MQTTS
mqtt-client-key:
    file: ./mqtt/certs/client.key  # Point your Client Key for MQTTS
```

5.2 Elasticsearch Setup (Linux)

- Step1: download the PGP key

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
```

For v8.*

- Step2: Install apt-transport and "8.* apt repo"

```
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-8.x.list
```

- Step3: Install elasticsearch Save the password that will be generated for the user "elastic"

```
sudo apt-get update && sudo apt-get install elasticsearch
```

- Step4: Enable on boot and start the service.

```
sudo systemctl daemon reload
sudo systemctl start elasticsearch.service
```

- Step5: Confirm by sending a request to "https://localhost:9200" Input the user "elastic" and generated password, you shoul recieve a response like so.

```
{
  "name" : "Cp8oag6",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "AT69_T_DTp-1qgIJlatQqA",
  "version" : {
    "number" : "8.4.1",
    "build_type" : "tar",
    "build_hash" : "f27399d",
    "build_flavor" : "default",
    "build_date" : "2016-03-30T09:51:41.449Z",
    "build_snapshot" : false,
    "lucene_version" : "9.3.0",
    "minimum_wire_compatibility_version" : "1.2.3",
    "minimum_index_compatibility_version" : "1.2.3"
  },
  "tagline" : "You Know, for Search"
}
```

- Step6: Create a user and password for kibana(v8.*) system.

```
/usr/share/elasticsearch/bin/elasticsearch-reset-password -u kibana_system
password="password"
```

- Step7: To access elasticsearch from another machine you have to make the following changes to the configuration file in elasticsearch. go to `/etc/elasticsearch/elasticsearch.yml` and append the following configurations. Elasticsearch v8.* comes with security enabled by default, so all incoming connections either uses basic security authentication or certificates for connections.

```
network.host: 0.0.0.0
http.port: 9200
discovery.seed_hosts: ["0.0.0.0", "[::1]"]
```

Security configuration should be like the sample below.

```
# Enable security features
xpack.security.enabled: true

xpack.security.enrollment.enabled: true

# Enable encryption for HTTP API client connections, such as Kibana, Logstash, and Agents
xpack.security.http.ssl:
 enabled: true
 keystore.path: certs/http.p12

# Enable encryption and mutual authentication between cluster nodes
xpack.security.transport.ssl:
 enabled: true
 verification_mode: certificate
 keystore.path: certs/transport.p12
```

```
 truststore.path: certs/transport.p12
# Create a new cluster with the current node only
# Additional nodes can still join the cluster later
cluster.initial_master_nodes: ["<your-node-name>"]

# Allow HTTP API connections from anywhere
# Connections are encrypted and require user authentication
http.host: 0.0.0.0
```

Go to `/etc/elasticsearch/certs/` and copy "http_ca.crt" you will need it for encryted connection from kibana(v8.*).

For v7.* (linux)

Repeat step1 above. * Step2: Install apt-transport and "7.* apt repo"

```
 sudo apt-get install apt-transport-https
 echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee /etc/apt/sources.list.d/
elastic-7.x.list
```

* Step3: Install elasticsearch

```
 sudo apt-get update && sudo apt-get install elasticsearch
```

* Step4: Enable on boot and start the service.

```
 sudo systemctl daemon reload
 sudo systemctl start elasticsearch.service
```

* Step5: send a request to http://localhost:9200 you should get a response like so.

```
{
 "name" : "Cp8oag6",
 "cluster_name" : "elasticsearch",
 "cluster_uuid" : "AT69_T_DTp-1qgIJlatQqA",
 "version" : {
   "number" : "7.17.6",
   "build_flavor" : "default",
   "build_type" : "tar",
   "build_hash" : "f27399d",
   "build_date" : "2016-03-30T09:51:41.449Z",
   "build_snapshot" : false,
   "lucene_version" : "8.11.1",
   "minimum_wire_compatibility_version" : "1.2.3",
   "minimum_index_compatibility_version" : "1.2.3"
 },
 "tagline" : "You Know, for Search"
}
```

- Step6: To access the cluster from a different machine modify the configuration file at `/etc/elasticsearch/elasticsearch.yml` with the following values; and restart the service.

```
network.host: 0.0.0.0
http.port: 9200
discovery.seed_hosts: ["0.0.0.0", "[::1]"]
```

NOTE

Elasticsearch v7.* does not have security enabled by default, to configure basic security and certificates follow the steps below

Security Setup in Elasticsearch v7*

MINIMAL SECURITY (USERS AND PASSWORDS)

- Step1: Go to `/path/to/elasticseach.yml` and add the following to the file;

  ```
  xpack.security.enabled: true
  ```

- Step2: Create passwords for built-in/new users

  ```
  ./bin/elasticsearch-setup-passwords interactive -u (username)
  ```

  restart the service

SECURED HTTPS TRAFFIC FOR ELASTICSEARCH V7*

Using the elasticsearch certutil tool helps to create the neccesary certificates used by elasticsearch in this mode. Go to the directory where elasticsearch binary executables are, they are usually at `/usr/share/elasticsearch/bin` * Step1: Certificate signing request (CSR).

```
/usr/share/elasticsearch/bin/elasticsearch-certutil http
```

when prompted for CSR type "n" for no. You could say yes if you have an already existing CA authority that you do not control to sign your certificates. when promted to use an existing CA type "n" for no. You could say yes, if you already previously generated one. A new CA will be generated, provide the neccesary details, Hostnames(if available) and ip addresses depending on your setup. A new zip file `/usr/share/elasticsearch/elasticsearch-ssl-http.zip` is created.

- Step2: unzip the file

  ```
  unzip elasticsearch-ssl-http.zip
  ```

  After unziping, two folders are created, one for elasticsearch and one for kibana, they both contain three(3) files, they are the certificate `http.p12`, Readme.txt and a sample-elasticsearch.yml and also `elasticsearch-ca.pem`, Readme.txt and a sample-kibana.yml these files contain next steps for the configuration.

- Step3: copy the certificate to your elasticsearch configuration directory. and add the following to your elasticsearch.yml file

  ```
  xpack.security.http.ssl.enabled: true
  xpack.security.http.ssl.keystore.path: "http.p12"
  ```

  save and restart elasticsearch For Kibana(v7.*) go to kibana.yaml and update "elasticsearch.hosts: ['http://localhost:9200']" to "elasticsearch.hosts: ['https://:9200']" Copy "elasticsearch-ca.pem" file directly into the kibana config directory without renaming it. uncomment and modify `elasticsearch.ssl.certificateAuthorities: $KBN_PATH_CONF/elasticsearch-ca.pem`

5.3 Elasticsearch setup for windows

5.3.1 v8.* setup (windows)

- Step1: download the .zip archive

  ```
  https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.4.2-windows-x86_64.zip
  ```

- Step2: extract it and go to `configs/elasticsearch.yml` uncomment and modify these values.

```
network.host: 0.0.0.0
http.port: 9200
discovery.seed_hosts: ["0.0.0.0", "[::1]"]
```

- step3: start elasticsearch

  ```
  .\bin\elasticsearch.bat
  ```

  password credentails with be output to terminal

- step4: certificate is located at `configs/certs` copy certificate to your kibana config/installation directory

For Kibana(v8.*) go to kibana.yaml and update "elasticsearch.hosts: ['http://localhost:9200']" to "elasticsearch.hosts: ['https://:9200']" Copy the cert from `configs/certs` above into the kibana config directory without renaming it. uncomment and modify `elasticsearch.ssl.certificateAuthorities: $KBN_PATH_CONF/'cert'`.

5.3.2 v7.* setup (windows)

- Step1: download the .zip archive

  ```
  https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.17.6-windows-x86_64.zip
  ```

  Repeat step2 and step3 above.

5.4 Kibana Setup

5.4.1 v8.* (linux)

- Step1: download the pgp key;

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
```

- Step2: Install apt-transport and elastic-repo for v8.*

```
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/
elastic-8.x.list
```

- Step3: install kibana

```
sudo apt-get update && sudo apt-get install kibana
```

you can generate an enrolment token for kibana with the following command;

```
bin/elasticsearch-create-enrollment-token -s kibana   # You can either use this or proceed with the certificate provided by elasticsearch
```

- Step4: configure kibana add the following to your config/kibana.yml

```
server.port: 5601
server.host: "localhost" or 0.0.0.0
server.publicBaseUrl: "http://localhost:5601"
elasticsearch.hosts: ["https://<elasticsearch_ip:9200"]
elasticsearch.username: "kibana_system"    # this username and password is valid when you have created a user "kibana_system" in your elasticsearch node.
elasticsearch.password: "password"
elasticsearch.ssl.certificateAuthorities: [ "path/to/cert/gotten/from/elasticsearch/http_ca.crt" ]
elasticsearch.ssl.verificationMode: certificate
```

- Step5: Enable on boot and start the service

```
sudo systemctl daemon reload
sudo systemctl start kibana.service
```

- Step6: Access kibana

```
You can access kibana on http://<IP:5601
```

5.4.2 v7.* setup (linux)

Same steps for v8.* except step2 and 4 * Step2

```
sudo apt-get install apt-transport-https
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee /etc/apt/sources.list.d/
elastic-7.x.list
```

* Step4: configure kibana add the following to your config/kibana.yml

```
server.port: 5601
server.host: "localhost" or 0.0.0.0
server.publicBaseUrl: "http://localhost:5601"
elasticsearch.hosts: ["https://<elasticsearch_ip:9200"] if security is configured elasticsearch.hosts: ["http://elasticsearch_ip:9200"]
elasticsearch.username: "kibana_system"    # this username and password is valid when you have created a user "kibana_system" in your elasticsearch node.
elasticsearch.password: "password"
elasticsearch.ssl.certificateAuthorities: [ "path/to/cert/gotten/from/elasticsearch/http_ca.crt" ] # comment this out if security is not configured
elasticsearch.ssl.verificationMode: certificate # comment this out if security is not configured
```

5.4.3 v8.* (windows)

- Step1: Download the .zip file

```
https://artifacts.elastic.co/downloads/kibana/kibana-8.4.2-windows-x86_64.zip
```

- Step2: extract the files. and configure Kibana. go to configs/kibana.yaml, from the previous configurations in linux, input the neccesary values
- Step3: start the service

```
.\bin\kibana.bat
```

5.5 Thingsboard Integration

EnOcean IoT Connector can be easily connected to the Thingsboard platform, and you can easily visualize the data.

Thingsboard offers many integration options. The most straight forward is a platform integration with MQTT. This is a Thingsboard Professional Edition feature available for Thingsboard Cloud or own platform instances.

For this purpose we provide the Uplink converter (direction seen from Thingsboard view). The converter takes the default output JSON Files and converts it to the Thingsboard expected format. The converter script can be found here.

5.5.1 Step-by-step integration

Thingsboard itself offers a very detailed manual here and more specific for MQTT here. We tested and deployed it with a thingboard.cloud deployment.

For an IoTC integration into Thingsboard you have to consider:

1. Deploy an IoTC instance and send the data to a MQTT Broker which is publicly available. e.g. Deploy IoTC in Azure Cloud with mosquito broker as described here.
2. Onboard some EnOcean devices. It is Important to confirm that data are sent via MQTT. e.g. confirm your MQTT Explorer is connected to the MQTT Broker.
3. Deploy Thingsboard at your self-managed platform or create yourself a Thingsboard cloud account.
4. Use the Thingsboard integration guide to set up integration for the IoTC.

a. At the step of Uplink converter copy and paste the script in the file here.

b. When asked for a topic filter at integration setup, specify `sensor/+/telemetry` as a default MQTT topics. Finish the remaining steps. For downlink just keep the default generated script from Thingsboard. If you are using the debug mode in integration you should already see some events on communicated telegrams.

5. If everything worked well, you should see devices that have sent a telegram after the integration in your devices list and the profiles. If your devices do not send any telegrams you will not see them in the list; please trigger a transmission to confirm the integration.
6. Create dashboards to visualize the data.

5.6 Azure IoT Central Integration

EnOcean IoT Connector can be easily setup to serve as the data source for Azure IoT Central. Azure IoT Central is an application platfrom-as-a-service (aPaaS) that allows easy development, management, and maintenance of enterprise-grade IoT solutions.

IoT Central offers a variety of integration methods. We use combination of IoT Central device bridge and Azure Service Bus. Device bridge forwards data from devices connected to EnOcean IoTC through to IoT Central application. The device bridge solution provisions several Azure resources into your Azure subscription that work together to transform and forward device messages to IoT Central. At the same time Azure Service Bus is utilized to provide the ability to provision new EnOcean devices directly from Azure IoT Central console.

5.6.1 Step-by-step deployment

Here is step-by-step guide for setting up the device bridge, Service Bus and EnOcean IoTC. This integration uses device bridge provided by Microsoft.

Prepare required keys

1. Get **ID scope** and **SAS Primary key** from IoT Central App > Permissions > Device connection groups > SAS-IoT-Devices.
2. Create API token (**IOT_CENTRAL_TOKEN**) from IoT Central App > Permissions > API Tokens > New.

Deploy the device bridge

1. Go to Azure Custom Deployment.
2. Fill in the required keys from previous step.
3. Review + Create
4. On Azure Portal go to Function App and select your newly deployed function.
5. From Function App menu go to Development Tools > Console and execute commands below

```
cd IoTCIntegration
npm install
```

6. Go to Overview and restart the Function on Azure Portal
7. Go to Functions > IoTCIntegration > Code + Test and "Get function URL". This will be used in iotconnector_engine variables as **IOT_BRIDGE_ADDRESS**

Deploy Azure Service Bus and create a new queue

1. In Azure Portal go to Service Bus and select create a new service bus with basic pricing tier and same resource group from previous step.
2. After the deployment is done, go to resource menu, select Shared access policies>RootManagedSharedAccessKey and note **Primary Connection String**. This will be used as **IOT_CENTRAL_SERVICE_BUS_CONN_STR**.
3. From the main Service Bus menu create new Queue and note its name. This will be used as **IOT_CENTRAL_SERVICE_BUS_QUEUE**.

Create data export on IoT Central

1. Go to IoT Central > Data Export, create a new destination. As a destination type choose Azure Service Bus Queue and input previously obtained **Primary Connection String** and the queue name.
2. In Data Export menu create new export. Select *device lifecycle events* as the type. Input following Enrichment property key:value pairs

```
device_name: Device name
device_temp: Device template name
```

Also set the destination to your previously created export destination.

Modify the docker-compose.yml file to enable integration

- Add variables below to *integration* containers environment variables on docker-compose.yml. Please use provided docker-compose.yml as a guideline.

```
- IOT_ENABLE_IOTCENTRAL=1
- IOT_CENTRAL_ADDRESS= # this is the URL of your IoT Central App
- IOT_CENTRAL_TOKEN=
- IOT_BRIDGE_ADDRESS=
- IOT_CENTRAL_SERVICE_BUS_CONN_STR= # Service bus connection string
- IOT_CENTRAL_SERVICE_BUS_QUEUE= # Name of the service bus queue
- EIOTC_API_URL=https://proxy/api/v2/
- BASIC_AUTH_USERNAME=
- BASIC_AUTH_PASSWORD=
```

5.7 Integrating IoTC with Azure IoT Hub

This documentation provides step-by-step instructions on how to integrate EnOcean IoT Connector with Azure IoT Hub.

EnOcean IoT Connector can be easily setup to serve as the data source for Azure IoT Hub. Azure IoT Hub enables highly secure and reliable communication between your Internet of Things (IoT) application and the devices it manages and it provides a cloud-hosted solution back end to connect virtually any device.

5.7.1 Step-by-step deployment

Step 1: Create a Device in Azure IoT Hub

1. Log in to Azure Portal:
   - Open your web browser and navigate to the Azure Portal.
   - Log in with your Azure account.
2. Navigate to Azure IoT Hub:
   - In the Azure Portal, search for "IoT Hub" and select it from the results.
3. Create a New Device:
   - In the IoT Hub pane, click on "IoT devices" in the left navigation menu.
   - Click the "New" button to add a new device.
   - Fill in the necessary details and click "Save."
4. Get the Connection String:
   - Select the created device.
   - Under the "Device details" tab, copy the "Primary connection string"

Step 2: Update Docker Compose File

- Add variables below to *integration* containers environment variables on docker-compose.yml. Please use provided docker-compose.yml as a guideline.

```
- IOT_AZURE_ENABLE=1
- IOT_AZURE_CONNSTRING=  # specify the connection string
```

Step 3: Deploy and Test

1. Deploy IoTC:
   - Run docker-compose up -d to deploy IoTC with the updated Docker Compose file.
2. Verify Integration:
   - Monitor logs to ensure successful integration with Azure IoT Hub.

6. Bidirectional communication

6.1 Overview

Bidirectional communication was added with IoTC v1.6 to support sending commands from IoTC to EnOcean devices. This allows controlling TRV's and switching ON/OFF light relay switches. Communication from IoTC to EnOcean devices is managed through MQTT using RPC topics (Remote Procedure Call). To support sending data to energy harvesting devices, IoTC also supports queuing commands until the device is ready. The application can manage data in the queue and will get status notifications from IoTC through MQTT RPC topics.

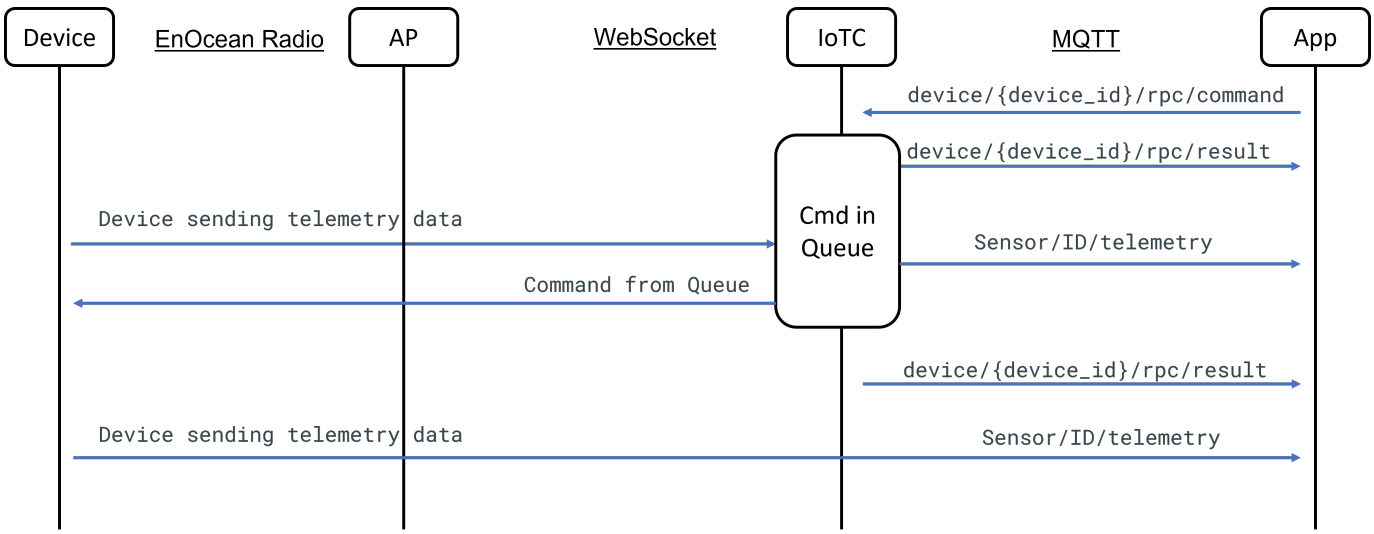
6.1.1 Commissioning of Bidirectional Devices

Bidirectional devices are commissioned using the `POST /devices` API as described here. Once a device is commissioned it must be configured for bidirectional communication. As the configuration procedure is device specific, it is required to follow the configuration procedure described for the specific device. The following list of devices are currently supported:

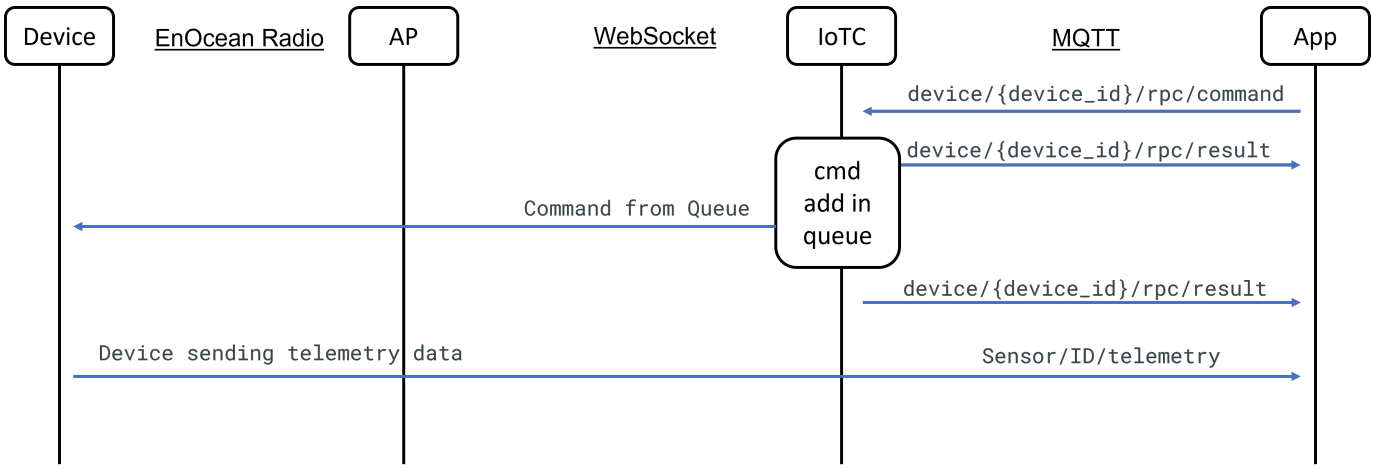
1. Micropelt iTRV MVA004
2. NodOn EnOcean Multifunction Relay Switch
3. NodOn EnOcean Smart Plug
4. OPUS TRV

6.1.2 Controlling bidirectional devices

Communication from IoTC to EnOcean devices is managed through MQTT using RPC topics (Remote Procedure Call). The communication and RPC topic flow is shown in below diagrams. To send a command the application will initiate a request using the `device/{device_id}/rpc/command` topic. IoTC puts the command in the queue and returns a status notification using the `device/{device_id}/rpc/result` topic. The queue will examine if the device is an energy harvesting device or if the command can be sent immediatly to the device. In case the device is an energy harvesting device, IoTC will keep the command in queue until receiving any data from the device like regular telemetry data. Data from the device is used to indicate the device is awake and can receive data. Once data is sent to the device IoTC will notify the application using the `device/{device_id}/rpc/result` topic.

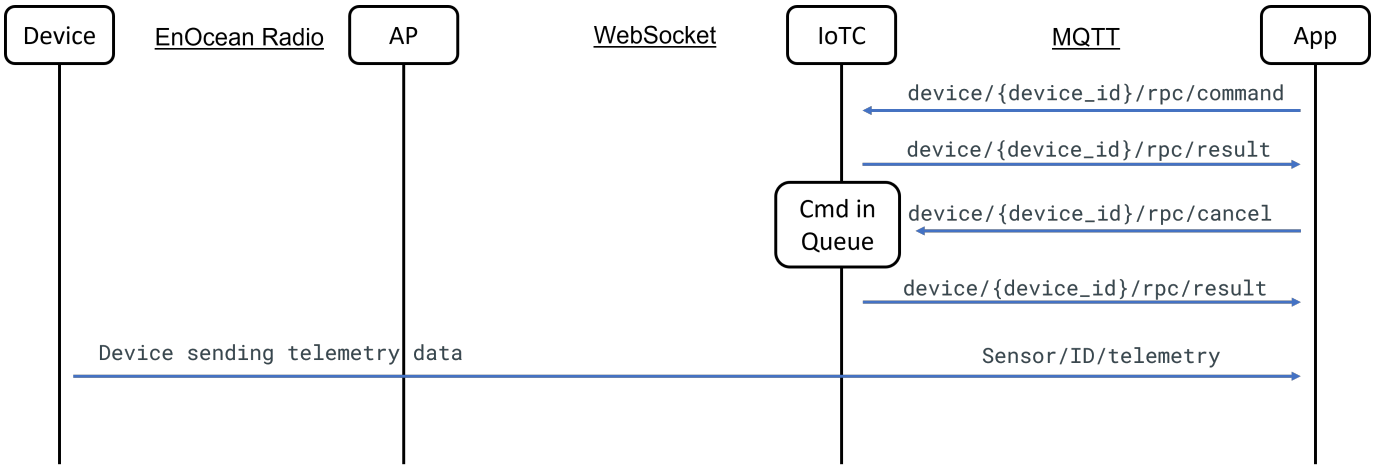


Communication and RPC topic flow for sending data to an energy harvesting device



Communication and RPC topic flow for sending data to an always on device

The application may cancel data in the queue by using the `device/{device_id}/rpc/cancel` topic as shown in below diagram. IoTC will cancel the request and notify the application using the `device/{device_id}/rpc/result` topic.



RPC topic flow for canceling command in queue

6.2 MQTT Interface

Making a requests

A request is made using the `device/{device_id}/rpc/command` topic with command schema as shown below. The command payload is using keys from the JSON representation of the EEP. More information about the payload is available in the description of the devices:

1. Micropelt iTRV MVA004
2. NodOn Multifunction Relay Switch
3. NodOn Smart Plug
4. OPUS TRV

```
{
  "request_id": 1,
  "eep": "A5-20-01",
  "payload": {
    "summerMode": 0,
    "temperature": 0,
    "temperatureSetpoint": 0,
    ...
    }
}
```

key	type and meaning
request_id	integer, unique identifier generated by application and used for tracking requests
eep	string, EnOcean Equipment Profile. This may be different from device's reporting EEP
payload	object, command payload using keys from JSON representation of EEP

Request results

After making a request the application should examine the `device/{device_id}/rpc/result` topic from IoTC for result of the request.

```
{
    "request_id": 1,
    "success": true,
    "message": "Command successfully queued"
}
```

key	type and meaning
request_id	integer, unique identifier of the RPC request
success	boolean, status of request
message	string, explanation of request's current state

If IoTC can properly transcode the request and put the command in queue, the application will receive below result:

```
{
    "request_id": 1,
    "success": true,
    "message": "RPC Command successfully transcoded [buffer=55:00:0A:07:01:EB:A5:00:00:00:08:DE:AD:BE:EF:00:03:FF:FF:FF:FF:FF:00:58]"
}
```

Once data is sent from the queue to the device, the application will receive below result:

```
{
    "request_id": 1,
    "success": true,
    "message": "RPC request sent to device"
}
```

Possible error messages are as below:

Device not added to IoTC

```
{
  "request_id": 1,
  "success": false,
  "message": "Device does not exist"
}
```

Device exists in IoTC but not configured to be controlled

```
{
  "request_id": 1,
  "success": false,
  "message": "Device is not configured for bidirectional communication"
}
```

Invalid json schema

```
{
  "request_id": 1,
  "success": false,
  "message": "JSON decoding error. Given payload is not a valid JSON."
}
```

One of the required keys of EEP is missing

```
{
  "request_id": 1,
  "success": false,
  "message": "Payload doesn't comply with given EEP(A5-20-01). [valve] is missing"
}
```

Device is not bidirectional

```
{
  "request_id": 1,
  "success": false,
  "message": "Device does not support bidirectional communication"
}
```

Canceling existing request

A request may be canceled using the `device/{device_id}/rpc/cancel` topic with below schema.

```
{
  "request_id": 1,
}
```

key	type and meaning
request_id	integer, unique identifier of the RPC request to be canceled

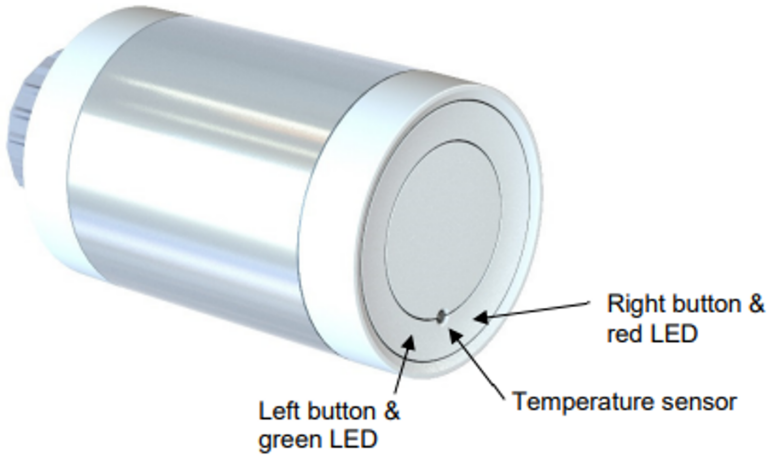
After canceling a request, the application should examine the `device/{device_id}/rpc/result` topic from IoTC for result of the request.

```
{
    "request_id: 1,
    "success": true,
    "message": "RPC command cancelled"
}
```

6.3 Devices

6.3.1 Micropelt iTRV MVA004

Product Name: iTRV MVA004
Product Description: Self-powered Radiator Valve
Product Information page: Link
Product EEP: A5-20-01
Product ID: 004900000005
Product unlock code: FFFFFFFE



Device Commissioning and Configuration

The device is commissioned using the `POST /devices` API as described here. For the Micropelt iTRV MVA004 below specific information must be used in API.

```
{
  ...
  "deviceType": "bidirectional",
  "eep": "A5-20-01",
  "productId": "004900000005",
  "unlockCode": "FFFFFFFE"
}
```

Once the device is commissioned IoTC will automatically configure the device for bidirectional communication. This will be done when receiving telemetry data from the device, which is default sent every one hour.

Controlling Device

IoTC can control iTRV using valve position functionality of the device. To set new valve position use json below in `device/{device_id}/rpc/command` topic. The "valve" parameter can be configured between 0 and 100, as defined in EEP.

```
{
    "request_id": 1,
    "eep": "A5-20-01",
    "payload": {
        "valve": 43,
        "temperature": 0,
        "runInitSequence":0,
        "runLiftSet":0,
        "valveOpen": 0,
        "valveClosed": 0,
        "summerMode": 0,
        "setPointSelection": 0,
        "setPointInverse": 1,
```

```
        "selectFunction": 0,
        "learnBit": 0
    }
}
```

6.3.2 NodOn Multifunction Relay Switch

Product Name: NodOn Multifunction Relay Switch
Product Description: Wireless actuator switch
Product Information page: Link
Product EEP: D2-01-0F
Product ID: 004600010004

Device Commissioning and Configuration

The device is commissioned using the `POST /devices` API as described here. For the NodOn Multifunction Relay Switch below specific information must be used in API.

```
{
  ...
  "deviceType": "bidirectional",
  "eep": "D2-01-0F",
  "productId": "004600010004",
}
```

Once the device is commissioned, IoTC will need to configure the device for bidirectional communication. To enable device configuration, follow below setps:

- Make sure device is factory reset
- Click the button 3 times. Device will go in pairing mode
- IoTC will configure the device automatically
- Device is configured

Controlling Device

To turn ON the device use json below in `device/{device_id}/rpc/command` topic.

```
{
    "request_id": 1,
    "eep": "f6-02-01",
    "payload": {
        "R1": 0,
        "EB": 1,
        "R2": 0,
        "SA": 0
    }
}
```

To turn OFF the device use json below in `device/{device_id}/rpc/command` topic.

```
{
    "request_id": 1,
    "eep": "f6-02-01",
    "payload": {
        "R1": 0,
        "EB": 0,
        "R2": 0,
        "SA": 0
    }
}
```

6.3.3 NodOn Smart Plug

Product Name: NodOn Smart Plug
Product Description: Wireless actuator switch
Product Information page: Link
Product EEP: D2-01-0B
Product ID: 004600010005

Device Commissioning and Configuration

The device is commissioned using the `POST /devices` API as described here. For the NodOn Smart Plug below specific information must be used in API.

```
{
  ...
  "deviceType": "bidirectional",
  "eep": "D2-01-0B",
  "productId": "004600010005",
}
```

Once the device is commissioned, IoTC will need to configure the device for bidirectional communication. To enable device configuration, follow below setps:

- Make sure device is factory reset
- Press the button for 2 seconds. Device will go in pairing mode, led will flash red.
- IoTC will configure the device automatically
- Device is configured

Controlling Device

To turn ON the device use json below in `device/{device_id}/rpc/command` topic.

```
{
    "request_id": 1,
    "eep": "f6-02-01",
    "payload": {
        "R1": 0,
        "EB": 1,
        "R2": 0,
        "SA": 0
    }
}
```

To turn OFF the device use json below in `device/{device_id}/rpc/command` topic.

```
{
    "request_id": 1,
    "eep": "f6-02-01",
    "payload": {
        "R1": 0,
        "EB": 0,
        "R2": 0,
        "SA": 0
    }
}
```

6.3.4 OPUS Micropelt TRV

Product Name: OPUS Micropelt TRV
Product Description: Self-powered Radiator Valve
Product Information page: Link
Product EEP: A5-20-06
Product ID: 00401000003F
Product unlock code: 53C65E34

Device Commissioning and Configuration

The device is commissioned using the `POST /devices` API as described here. For the OPUS Micropelt TRV below specific information must be used in API.

```
{
  ...
  "deviceType": "bidirectional",
  "eep": "A5-20-06",
  "productId": "00401000003F",
  "unlockCode": "53C65E34"
}
```

Once the device is commissioned IoTC will automatically configure the device for bidirectional communication. This will be done when receiving telemetry data from the device, which is default sent every one hour.

Controlling the Device

Opus TRVs manage valve position automatically when used with "Temperature Set Point" mode.

To control a TRV, a RPC command should be sent to respectetive MQTT topics. To `device/{eurid}/rpc/command` topic json below should be sent when a temperature change request is made.

```
{
  "request_id": 1,
  "eep": "a5-20-06",
  "payload": {
    "temperatureSetpoint": 35,
    "temperature": 20,
    "referenceRun": 0,
    "rfConnectionInterval": 0,
    "summerBit": 0,
    "setPointSelection": 1,
    "temperatureSelection": 1,
    "standBy": 0,
    "learnBit": 1
```

```
    }
}
```

Important keys are `temperatureSetpoint` and `temperature`.

temperatureSetpoint is the desired room temperature.

temperature is the current room temperature that is measured by a RCU.

MQTT Topics and expected information can be found here in detail.

7. Troubleshooting

7.1 System health information

IoTC periodically checks the status of the containers to validate correct operation of IoTC. After each check, a system health notification is sent on MQTT to notify the application of the IoTC status. The application should examine the health notification and notify the system administrator in case issues are observed. The application should also use the periodic health notification as a keep alive and expect issues if the health notifications are not received.

The system health notification sent on MQTT can also be requested through API. The system health information is guaranteed to be less than 60 seconds old when requested through the API.

The default system health check and reporting internal is 10 min, which can be configured through the HEALTH_PUBLISH_INTERVAL parameter in the docker-compose file.

A system health endpoint is automatically created with EEP D2-14-41 and ID: **04211939**. This is needed by IoTC to check the ingress and integration containers. This endpoint must be ignored by the application.

7.2 Debugging

7.2.1 Console log

Main debug & info messages from the IoTC can be viewed in the log of the `engine` container.

```
 ______        ____
|  ____|      / __ \
| |__    _ __ | |  | | ___ ___  __ _ _ __
|  __| | '_ \| |  | |/ __/ _ \/ _` | '_ \
| |____| | | | |__| | (_|  __/ (_| | | | |
|______|_| |_|\____/ \___\___|\__,_|_| |_|

 _____   _______      _____                          _
|_   _| |__   __|    / ____|                        | |
  | |  ___ | |______| |     ___  _ __  _ __   ___ ___| |_ ___  _ __
  | | / _ \| |______| |    / _ \| '_ \| '_ \ / _ \ __| __/ _ \| '__|
 _| || (_) | |      | |___| (_) | | | | | | |  __/ (__| || (_) | |
|_____\___/|_|       \_____\___/|_| |_|_| |_|\___|\___|\__\___/|_|

...
INFO::2021-04-19 14:03:29,500::dedupper::Adding gateway with mac='1c28afc2950a' to approved list.
INFO::2021-04-19 14:03:41,505::dedupper::Adding sensor with eurid='04138bb4' to approved list.
INFO::2021-04-19 14:04:41,525::dedupper::Adding sensor with eurid='feee14ab' to approved list.
INFO::2021-04-19 14:07:47,597::dedupper::Adding sensor with eurid='0412d7ef' to approved list.
INFO::2021-04-19 14:08:05,605::dedupper::Adding sensor with eurid='0412d7c3' to approved list.
INFO::2021-04-19 14:08:32,616::dedupper::Adding sensor with eurid='0412d7ab' to approved list.
INFO::2021-04-21 08:16:54,861::dedupper::Adding gateway with mac='d015a6ce04a2' to approved list.
...
```

Other containers post messages to their console as well. To see logs:

- on Docker Desktop: open Docker Desktop -> go to Containers/Apps, find the container group e.g. `local_deployment`, click on the line of interest. e.g. `local_deplomentproxy_1`.
- on Azure Container Instances: Go to Settings -> Container. Select the container e.g. `engine` and select the `Logs` tab.

Most relevant log messages are:

- logs on start up
- logs on device communication
- licensing reports
- health information about Aruba AP
- periodic and continuos summary reports of incoming, processed and outgoing traffic of onboarded devices
- authentication of gateways

Samples of the most important messages and short description are below.

Start up

```
ERROR::...::Azure egress is enabled but connections string is not set.
#Check your docker deployment file for any issues and redeploy the IoTC.
ERROR::...::MQTT is enabled but MQTT_CONNECTION_STRING is empty or invalid.
#Check your docker deployment file for any issues and redeploy the IoTC.
ERROR::...::Could not determined host &/or port from MQTT_CONNECTION_STRING. Exiting with code 0.
#Check your docker deployment file for any issues and redeploy the IoTC.

INFO::...::MQTT client id not set. Using a random one.
#The MQTT client ID can be set in the docker file but this is not mandatory.
```

Sensor communication

```
INFO::...::Adding sensor with XXXXXXXX to approved list.
#A sensor communicated first time to the IoTC. Sensor is added to the approved list (license consumption).
INFO::...::Adding gateway with XXXX-XXXX-XXXX to approved list.
#A gateway is communicating first time to the IoTC. It is added to the approved list (license consumption).

ERROR::...::EEP AA-BB-CC sent by XXXXXXXX is not supported.
#The used EEP of the device is not supported and not processed. Please check if the EEP is correct (in the API). Please contact support@enocean.com for more details.
ERROR::...::EEP AA-BB-CC Parsing error id=XXXXXXXX
#The Telegram was malformed / corrupted, IoTC could not parse the telegram. Should this message repeat itself frequently contact support@enocean.com for advise.
```

```
INFO::...::Replay Exception ....
#A secured device is communicating with an already used sequence counter. Indication of a possible thread.
WARNING::...::CMAC verification failed id=XXXXXXXX
#A message from a secured device could not be authenticated.
WARNING::...::Device XXXXXXXX now unsecure. Dropping.
#A device which was onboarded as secure is not transmitting in standard mode. Possible thread.

WARNING::...::Received telegram from device: XXXXXXXX, but the device is not active. Dropping
#Messages from an onboarded device received but the `active flag` is disabled. Set the flag to active once ready.
```

Licensing reports

Licensing reports are mainly listed in the `engine` container log, but also in the `ingress` container.

`ENGINE` CONTAINER LOG

```
# logs on start up

ERROR::...::License key is not set.
#The key was not set during IoTC deployment in the properties. Check the deployment file and redeploy the IoTC Containers.
WARNING::...::License key XXXXX-YYYYY-XXXXX-YYYYY is not active.
#The License was set properly but it is not active on the licensing server. Contact your sales representative.
ERROR::...::Could not contact the server.”
ERROR::...::Could not contact the server. Error message: {URLException}
ERROR::...::The signature check failed
#There seems to be a a problem with the internet connection to reach the licensing server.

# logs on first or periodic Activation

ERROR::...::Retrying activation in XXX s.
ERROR::...::Activation failed. Tried YYY times to activate.
WARNING::...::Could not activate license key:XXXXX-YYYYY-XXXXX-YYYYY.
# If this issues prevail contact support@enocean.com or your sales representative.

INFO::...:: License Key: XXXXX-YYYYY-XXXXX-YYYYY expired on ...
#Your trial or commercial license expired. It has to be renewed, the IoTC will stop to operate after the grace period. Contact your sales representative.

WARNING::...::Using grace period. Grace period expires in ...
#Your license is not valid and the IoTC runs in the grace period.
WARNING::...::Grace period expired on ...
#The grace period expired. IoTC is not functional.

# logs on license limits

WARNING::...::Package received from gateway XXXX-XXXX-XXXX outside of license allowance. Dropping.
#Your license reached its limit for allowed gateway connections. Please contact your sales representative.
WARNING::...::Sensor XXXXXXX is outside of license allowance or is not learned in. Dropping.
#Your license reached its limit for allowed sensor connections or there is no teach in information. Check if device is listed via the API. Please contact your sales representative,
```

`INGRESS` CONTAINER LOG

```
# logs on start up
ERROR::...::License key is not set. Exiting ingress. #The key was not set during IoTC deployment in the properties. Check the deployment file and redeploy the IoTC Containers.

ERROR::...::License XXXXX-YYYYY-XXXXX-YYYYY activation failed.
ERROR::...::Retrying license activation in XXX s.
# If this issues prevail contact support@enocean.com or your sales representative.

INFO::...::License activation ok. # License check was OK. The ingress will start.
```

Check this page for details on the license key.

Aruba health messages

With the Aruba OS 8.8.0.0 the manufacturer introduces AP health information updates which includes also the EnOcean USB Stick connection status. Health update is send every 120 seconds. The IoT Connector will decode the incoming Health updates and put this information into the ingress container console output. With this messages it is easy to identify should a USB be non operational or removed during operation.

Following descriptors a included in a log message:

- `MAC` The MAC of the Aruba AP.
- `HW_DESC` The Hardware descriptor for the access point.
- `SW_VERSION` The Software version of the access point.
- `USB`
- `ENOCEAN_USB` Hash of unique for the specific EnOcean USB Stick used.
- `USB_HEALTH` Health information showing the status of the USB link.

Healthy example messages are listed here:

`ingress` container

```
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=bbbbbbbbbbbb HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:c3df093db12e57a6e121722ce042f95c USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=bbbbbbbbbbbb HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:c3df093db12e57a6e121722ce042f95c USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=bbbbbbbbbbbb HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:c3df093db12e57a6e121722ce042f95c USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=bbbbbbbbbbbb HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:c3df093db12e57a6e121722ce042f95c USB_HEALTH: healthy
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: ENOCEAN_USB:deb480d77718bbbe5253896b9300acfd USB_HEALTH: healthy
```

Should the EnOcean USB stick be disconnected then this message will appear:

```
INFO::...::Aruba Health message contains no USB info.
INFO::...::Aruba AP Health--> MAC=aaaaaaaaaaaa HW_DESC=AP-505 SW_VERSION=8.8.0.0 USB: N/A USB_HEALTH: N/A
```

Periodic summary

A periodic summary is posted to the `engine` log to summarize the most important statistics about the operation.

```
INFO::...::Total processed Telegrams: 138940
INFO::...::Total learned-in devices: 15
INFO::...::Total MQTT messages: 138935
INFO::...::Total processed Telegrams: 138943
INFO::...::Total learned-in devices: 15
INFO::...::Total MQTT messages: 138938
```

The information provided: - `Total processed Telegrams` represents the incoming telegram counter. In a common EnOcean environment this number will increase continuously, maybe not every report. The increase factor depends on the count of installed devices. A continuously increasing number is a strong signal that the ingress and collection of telegrams is operational. - `Total learned-in devices` summarizes the count of all onboarded devices. This does not mean the devices are actively communicating. - `Total MQTT messages` represent the outgoing telegram counter. A continuously increasing number is a strong signal that the egress interface is working correctly.

Authentication of gateways

`ingress` container

```
# logs at first connect

WARNING::...::Failed to authenticate AP. Payload is not JSON.
#The AP is not compatible
WARNING::...::Incorrect credentials.
#Entered credentials on AP do not match the ones specified in the docker file during deployment. Check it.
WARNING::...::Validation Error. Could not authenticated AP.
#Validation failed, if the issue prevails check the AP for details.


#The HTTPS authentification token expires periodically. Every connected AP will need to renew the token. Following messages will periodically repeat:
INFO::...::Disconnecting client on WS handler: x.x.x.x
INFO::...::AP Authentication request from: x.x.x.x
INFO::...::Connected client on WS handler: x.x.x.x

INFO::...::Disconnecting AP with code 1008 due to invalid token.
#Token is invalid or expired. Should this message repeat check the AP for details.
```

7.3 Best Practices

7.3.1 Cost estimation of runtime in the Azure Container Instances

As long the IoTC is running in the ACI it is subject to repeated cost. The fee depends on the kind of Azure Subscription you have and the consumed resources of the IoTC.

Azure subscriptions

Azure offers different subscription models. It is important to consider the correct model for your operation as there are considerable differences in the cost model.

The pay-as-you-go subscription lets you pay for the services and resources that you use on a monthly basis. A credit or debit card must be attached to the account, and billing for this account is on a monthly basis.

Microsoft has many types of member offers that offer reduced rates for Azure services, like MSDN Platform subscribers and Visual Studio subscribers, to name a few. These types of subscriptions offer substantial discounts over a pay-as-you-go subscription, so it is highly recommended that businesses review and take advantage of any offers for which they may qualify.

LIMITS OF WEBSOCKET CONNECTIONS

Based on the selected subscription model, there might be set limits for maximum parallel websocket connections (APs). Please confirm the quotas for your deployment with Microsoft when planning a commercial usage.

Specify consumed resources

In the docker compose files, you can specify the `reserved` (minimum) and the `limits` (maximum) of resources allocated for containers inside your group at deployment.

Example:

```
    version: "3.9"
    services:
    redis:
        image: redis:alpine
        deploy:
        resources:
            limits:
                cpus: '0.50'
                memory: 50M
            reservations:
                cpus: '0.25'
                memory: 20M
```

ACI will consider this deploy features.

We have entered the typical values in the azure deployment file `/deploy/azure_deployment/docker-compose.yml`. Feel free to edit them based on your considerations. A benchmark for performance tests at specified values can be found here

TRACK CONSUMPTION OF RESOURCES

You can track the actual consumption of your resources with the Azure container monitor and adjust it for your use case and cost plans accordingly. Using the Azure CLI you can also get individual container consumption from the container group, which is much more precise. e.g., use the following command for average CPU Usage:

```
az monitor metrics list --resource (az container show --resource-group **your-resource-group** --name **your-container-group-name** --query id --output tsv) --metric
CPUUsage --dimension containerName --output table
```

Read the Azure documentation for detailed options on the command.

Azure cost calculation

When running the IoTC in Azure, the costs can be evaluated with the Azure cost calculator for **Azure Container Instances**.

Permanent storage or additional components, e.g., Azure Io Hub, need to be considered in pricing separately.

TYPICAL PARAMETERS FOR CALCULATION

Parameter	Typical Value	Notes
Container Groups	1	Assuming you deployed one IoTC
Duration	30 days	How long will the container group (IoTC) run per month
vCPU	1	How many CPU cores will the container group (IoTC) consume in sum. This depends on what you have specified as reserved and limit in the compose file for each container. If the sum of the reservations is bigger than 1, then the group will constantly consume 2 vCPU, which will effectively double the cost. A benchmark for performance tests can be found here
Memory	2GB	Consumed RAM This depends on what you have specified as [reserved and limit].

7.3.2 Performance tests

Load tests in Azure

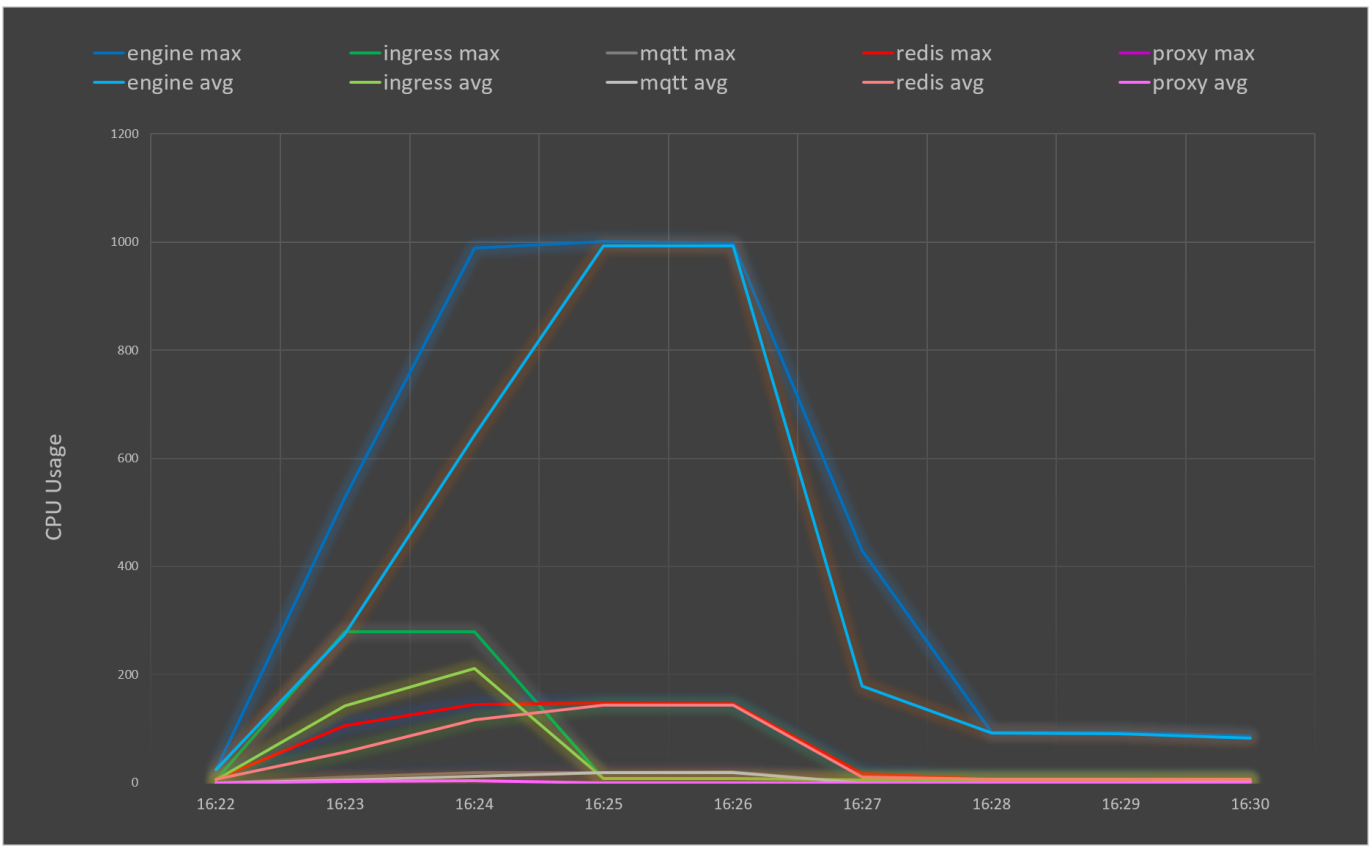
With the typical resources specification for each container, we performed a load test at Azure Container Instances to demonstrate resources usage and prove seamless operation even in heavy load situations. The typical resource parameters are listed in `/deploy/azure_deployment/docker-compose.yml`.

SETUP

Nr. of total sensors	4.000
Nr. of messages per sensor	10
Nr. of total messages	40.000

RESULTS

The egress took approx 4 minutes. All messages have been received.



CPU Usage is measured in milli-cores. One milli-core is 1/1000th of a CPU core, e.g., 500 milli-cores represent the usage of 0.5 CPU core.

CONCLUSION

- Maximum performance of 10 k messages per minute with the specified resources was reached.
- If the engine gets assigned more CPU cores, performance is likely to increase linearly, but 10k messages per minute already references a very huge and busy installation, e.g., international airports with over 20 k EnOcean devices mounted.
- Total consumption in idle time and typical sized installations stays much below 1 vCPU. In high traffic times consumption was greater then 1 vCPU.

8. Update Notes

8.1 Updating EIoTC

IMPORTANT for updating from IoTC v1.3 to IoTC v1.4 see update notes

IMPORTANT for updating from IoTC v1.2 to IoTC v1.3 see update notes

The IoTC application follows a quarterly release cycle for introduction of new features. Updating IoTC to latest release is simple using docker. The following describes the steps for both local and Azure deployments.

1. Make a GET request to /api/v2/system/backup endpoint to generate a system backup
2. Update IoTC using docker

Local Deployment

```
#Stop deployment
docker compose down

#Pull latest versions of docker images
docker compose pull

#Start deployment
docker compose up
```

Azure Deployment

```
#Stop deployment
docker compose down

#Start deployment
docker compose up
```

8.2 Updating from IoTC v1.3 to IoTC v1.4

IOTC version 1.4 provides a significant improvement of the setup, management, and system functionalities to further enhance setup and operation of IOTC. To leverage these new features, IOTC v1.4 introduces an updated API referred to as API v2. IoTC v1.4 also includes a new device storing format together with additional services and environment variables. These changes did not allow to maintain backwards compatibility with IoTC v1.3. It is therefore required to follow below steps when updating from IoTC v1.3 to IoTC v1.4.

8.2.1 Generate system backup file

IoTC v1.4 introduces a new format for storing devices as shown below. It is therefore required to generate and modify an IoTC v1.3 system backup file to make it compatible with IoTC v1.4.

```
{
  "APIVersion": "1.3.0",
  ...
  "Connections": [
    ...
    {
      "sourceEurid": "feee14ab",
      "destinationEurid": "ffffffff",
      "isPTM": false,
      "friendlyID": "Motion Sensor",
      "location": "Meeting Room",
      "customTag": "",
      "eep": "A5-07-03",
      "activeFlag": true,
    },
    ...
  ]
}
```

```
{
  "apiVersion": "1.4.0",
  ...
  "devices": [
    ...
    {
      "eurid": "feee14ab",
      "destinationEurid": "ffffffff",
      "deviceType": "sensor",
      "friendlyid": "Motion Sensor",
      "location": "Meeting Room",
      "customTag": null,
      "eep": "A5-07-03",
      "activeFlag": true,
    },
    ...
  ]
}
```

1. Make a GET request to /api/v1/backup endpoint to generate a system backup
2. Make the following modifications in the system backup file to make it compatible with IoTC v1.4.

   Change the following variables throughout the backup file

   - APIVersion -> apiVersion
   - Connections -> devices
   - sourceEurid -> eurid
   - friendlyID -> friendlyid

   Change isPTM to deviceType acording to below

   - isPTM: false -> deviceType: sensor
   - isPTM: true -> deviceType: switch

8.2.2 Update docker-compose.yml file

IoTC v1.4 requires additional services and environment variables to be added to the docker-compose.yml file. It is therefore required to update the docker-compose.yml file as described below before updating to IoTC v1.4. As reference for adding below services and environment variables it is recommended to download the docker-compose.yml file included in IoTC v1.4.

IoTC v1.4 introduced two new containers; rabbitmq and fluentd. The rabbitmq container is required for both local and Azure deployment whereas fluentd is only required for local deployment. The services must be added as shown below.

```
fluentd:
    user: root
    image: enocean/iotconnector_fluentd:latest
    volumes:
        - /var/lib/docker/containers:/var/lib/docker/containers
        - ./logs:/outputs/
    logging:
        driver: local

rabbitmq:
    image: rabbitmq:3.10-management
    ports:
        - "15672:15672"
        - "5672:5672"
```

In addition to these services environment variable below must be added to “engine”, “integration” and “ingress” services.

```
environment:
   ...
   - RABBITMQ_HOST=rabbitmq
   ...
```

Api service’s configuration changed with API v2

```
api:
    image: enocean/iotconnector_api:latest
    ports:
        - "1887"
    restart: always
    environment:
        - REDIS_URL=redis://redis:6379/0
        - LICENSE_KEY= # License is now required for API as well.
        - DJANGO_SETTINGS_MODULE=eiotc_api.settings.dev
    volumes:
        - ./logs:/var/log/enocean/eiotc
    depends_on:
        - redis
```

Api url on engine configuration must be update to API v2

```
engine:
    ...
    environment:
        ...
        - API_URL=http://api:1887/api/v2
```

8.2.3 Update to API v2

After updating to IoTC v1.4 it is required to align API integration with API v2 by going through each function in the API documentation.

Key changes in API v2 are:

- Organization of all functions in three clearly defined groups (system, devices, gateways) for better structure and easier integration.
- Extension of device functions with {deviceId} to allow addressing a specific device, for instance to request device information for one specific device.
- Support of additional filter options for the GET /devices function to only show devices with a specific EEP, a specific deviceType or at a certain location.
- Extension of gateway functions with {mac} to allow addressing a specific gateway, for instance to request device information for one specific gateway.
- Support for backup functions has been moved into the system group
- The new function GET/system/license has been added to easily verify if IoTC license is valid.
- The new function GET/system/logs has been added to retrieve IoTC logs through the Swagger UI.

8.3 Updating from IoTC v1.2 to IoTC v1.3

IoTC v1.3 introduces a new system health capability and MQTTS. The system health capability reports on IoTC system status for confirmation of proper operation and detection of issues. While MQTTS enables sure communication between IoTC and application. To introduce these features, it's been necessary to extend the docker-compose as described below. These parameters must be added to the IoTC v1.2 docker-compose file before updating to IoTC v1.3

1. The following environment variables must be added to "engine" to accommodate for system health.

```
engine:
    ...
    environment:
        ...
        # newly added variables
        - INGRESS_HOST=ingress
        - INGRESS_PORT=7070
        - INGRESS_USERNAME=  #enter engine user name e.g. user1
        - INGRESS_PASS= #enter engine password e.g. pass1
        - API_URL=http://api:1887/api.beta/v1
        - HEALTH_PUBLISH_INTERVAL=600
```

2. The following environment variables must be added to "secrets" and "integration" to enable MQTTS. This may be skipped if MQTTS is not used.

```
secrets:
    ...
    mqtt-ca-cert:
        file: ./mqtt/config/certs/ca.crt   # Point your CA Certificate
    mqtt-client-cert:
        file: ./mqtt/config/certs/client.crt  # Point your Client Cert
    mqtt-client-key:
    file: ./mqtt/config/certs/client.key  # Point your Client Key
```

```
integration:
    ...
    environment:
        ...
        - MQTT_CONNSTRING=mosquitto:8883
        - MQTT_AUTH=0
        - MQTT_LOCAL_EGRESS_ENABLE=1
        - MQTT_USE_TLS=True
        - MQTT_USE_TLS_VERIFY=True
        ...
    secrets:
        - source: mqtt-ca-cert
            target: /mqtt-ca.crt
        - source: mqtt-client-cert
            target: /mqtt-mqtt_client.crt
        - source: mqtt-client-key
            target: /mqtt-mqtt_client.key
```

8.4 ChangeLog

8.4.1 EnOcean IoT Connector - 1.8.0

General

FEATURES

- Added support for bidirectional communication for following devices:
- Micropelt OPUS TRV: More information about bidirectional communication is available here.
- Added support for the following EEPs:
- D2-01-0F
- D2-01-0B
- D1-07-11
- D1-09-01

8.4.2 EnOcean IoT Connector - 1.7.0 Released

General

FEATURES

- Added support for bidirectional communication for following devices:
- NodOn Multifunction Relay Switch
- NodOn Smart Plug

  More information about bidirectional communication is available here.

8.4.3 EnOcean IoT Connector - 1.6.0 Released

General

FEATURES

- Added support for bidirectional communication, allowing the application to send commands to EnOcean devices. Below list of devices are currently supported:
- Eltako FSVA-230V
- Micropelt TRV More information about bidirectional communication is available here.
- A license grace period is introduced to the license check to avoid disruptions in case IoTC is unable to connect with the license server for a short period of time.

8.4.4 EnOcean IoT Connector - 1.5.0 Released

General

FEATURES

- Added support for Deuta EnoSense CO2 Sensor with EEP D1-09-10

Azure

BUGS

- Fixed Azure deployment issue that was caused by resource reservation rules of Azure ACI
- Fixed issue preventing Azure IoT Hub and MQTT to co-exist. The issue was introduced in IoTC v1.4, when switching to use RabbitMQ

8.4.5 EnOcean IoT Connector - 1.4.0 Released

General

FEATURES

- Updated API to v2
- Moved interservice messaging to RabbitMQ
- Reworked logging to have clear and standardized messages in all containers

API Container

FEATURES

- Added ability to download logs from API

Proxy Container

FEATURES

- Added health check for proxy container

8.4.6 EnOcean IoT Connector - 1.3.1 Released

Engine Container

BUGS

- usbInfo key error fix

8.4.7 EnOcean IoT Connector - 1.3.0 Released

General

FEATURES

- Support for secure MQTT
- Configuration option to only support encrypted data from secure devices
- System health check of IoTC is performed at fixed interval. Health information is published to the MQTT topic and can be accessed via API calls

API Container

FEATURES

- Added PATCH operation to /backup endpoint
- New system health API enpoint

Engine Container

FEATURES

- A5-12-01 EEP added

BUGS

- Minor transcoding error fixed on d2-14-40 and d2-14-41 profiles

Integration Container

FEATURES

- Added MQTTS support with x509 certificates
- Egress functionality has been decoupled from engine and moved to a separate "integration" container

Ingress

FEATURES

- Ingress token expiration time is now configurable

8.4.8 EnOcean IoT Connector - 1.2.0 Released

General

FEATURES

- All container OS images changed from Debian to Alpine
- All containers are now successfully passing SNYK static vulnerability scan.

Azure Container

FEATURES

- Commissioning of EnOcean devices into EIoTC is now possible from Azure IoT Central
- Integration between EIoTC and Azure IoT Central is enabled
- Integration for telemetry exchange between EIoTC and Azure IoT Central

Engine Container

FEATURES

- Commissioning of devices from EIoTC into Azure IoT Central enabled
- Azure IoT Central connection enabled and documented in the references.
- MQTT payload now validated against a schema for each egress message

BUGS

- Improve mqtt egress re-connection timing parameters

Ingress Container

FEATURES

- Optimization of Redis re-connection interval
- /auth/eotunnel endpoint for generic gateways enabled
- Support for generic gateway was added. /auth/eotunnel endpoint used for authentication.

8.4.9 EnOcean IoT Connector - 1.1.0 Released

General

FEATURES

- License check will provides more context why activation failed e.g. internet connection failed
- Timestamp added to MetaEvents of sensors
- Detailed report about performance and resource consumption during common peak traffic load scenarios
- Container versions are now listed in each container separately
- All MQTT JSON payloads are now available as JSON schemas in the download section

BUGS

- Start up of deployments in Azure is more stable and fails gracefully
- Fixed: Summary also enabled for egress into Azure IoT Hub

API Container

BUGS

- Renamed response schemas for telegram / gateway statistics

FEATURES

- Health messages are included are appended the gateway Information structure when using the API

Azure Container

FEATURES

- Redis startup error messages are now handled gracefully

Engine Container

FEATURES

- Added support for Deuta People passing counter ESC.
- Supported EEP List in documentation is autogenerated
- Extended documentation on additional debug/error messages

MQTT Container

FEATURES

- Telegram statistics of sensors are periodically posted on MQTT. Feature can be turned off..
- Topic: "/gateway//health" renamed to "/gateway//meta/event"
- All MQTT topics are configurable with ENV variables at deployment
- Removed sensor health & security from telemetry JSON. Equivalent present in metaEvent
- Telegram statistics gateways are periodically posted on MQTT. Feature is configurable..
- MQTT Topic structure documented in detail
- An example in docker-compose file is provided how to deploy the mosquito broker with TLS certificates and user authentication
- Telegram statistics of posted on MQTT with telemetry
- MQTT basic username and password authentication
- x509 Certificate Connection to MQTT

8.4.10 EnOcean IoT Connector - hotfix-engine-1.0.1 Released

Engine Container

FEATURES

- Fixed. Packet Type 10 Messages were ignored and not processed.

8.4.11 EnOcean IoT Connector - 1.0.0 Released

General

FEATURES

- Minor corrections and improvements, Not influencing the product interface.
- Advanced traffic load tests and stress tests with simulated conditions
- Advanced integration and system tests

8.4.12 Beta Releases

- EnOcean IoT Connector - Beta 0.1.0 Released
- EnOcean IoT Connector - Beta 0.2.0 Released
- EnOcean IoT Connector - hotfix-0.2.2 Released
- EnOcean IoT Connector - hotfix-engine-0.2.3 Released

9. About

9.1 Support

For bug reports, questions or comments, please submit an issue here.

Alternatively, contact support@enocean.com