EnOcean IoT Connector

Product Documentation

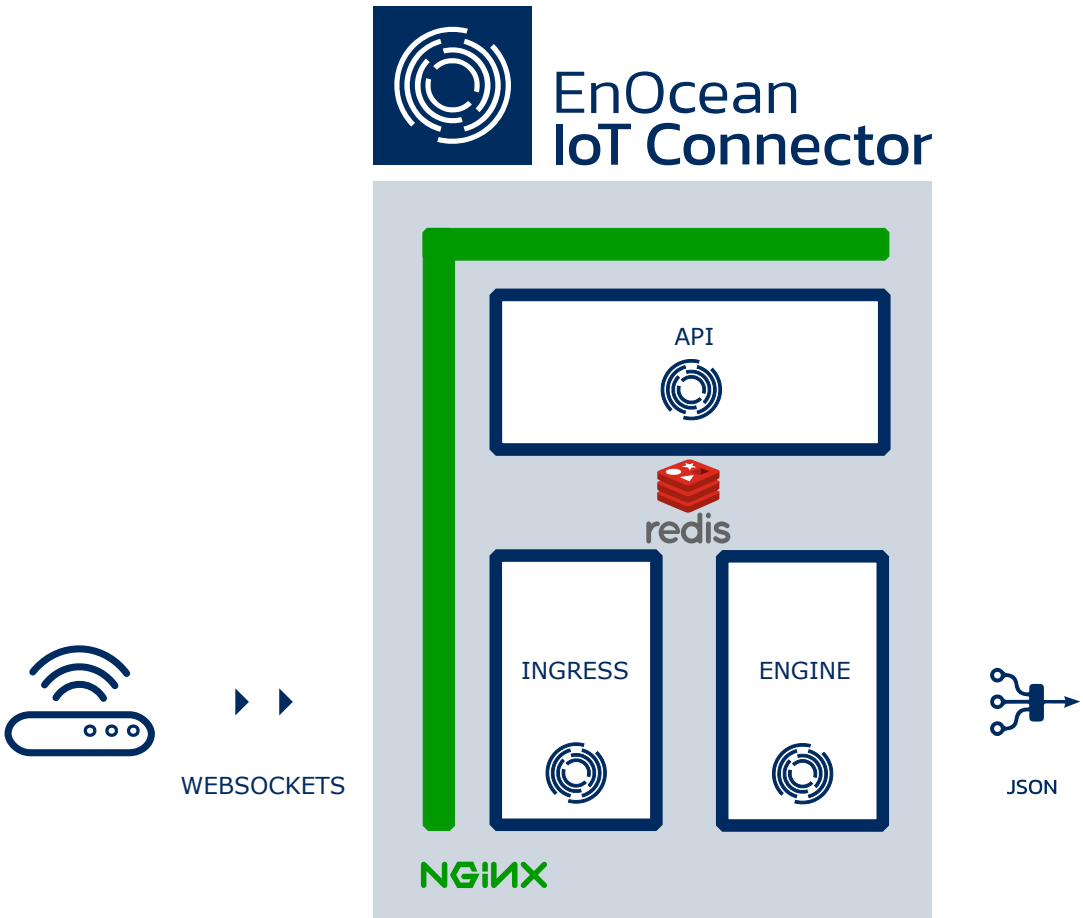
EnOcean GmbH

Table of contents

1. Product Description

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. Redis
5. 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)
- private cloud
- on-site

IoTC containers are built for `linux/arm/v7`, `linux/arm64` and `linux/amd64`

This guide will explain the basic functionality and cover the basic deployment steps and configuration options.

DOCUMENTATION VERSION/TAG

`1.0.4` / `2021-07-03 10:42:25+02:00`

1.1.1 Features

Ingress

The ingress controls all incoming traffic from ingress gateways.

- The IoTC currently supports Aruba Access Points as ingress gateways.
- Communication is executed via secure web sockets only. Secure web sockets use SSL encryption. A manual how to add a certificate to an Aruba AP is listed here.
- It detects duplicates - i.e. filter if two or more ingress gateways received the same radio signal, and makes sure each signal is processed only once.
- Processes the ESP3 Protocol. Only Packet Type 01 is currently supported.

Engine

The IoTC engine completely supports the EnOcean radio protocol standards as defined by the EnOcean Alliance. Including:

- addressing encapsulation
- chaining
- decryption & validation of secure messages
- EEP processing

Additionally the IoTC evaluates sensor health information:

- information included in signal telegram
- telegram statistics

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

BUILT-IN END-POINTS

Available end-points are MQTT and the Azure IoT Hub. The output data format is JSON, in accordance to the `key-value` pairs defined by the EnOcean Alliance IP Specification.

SUPPORTED ENOCEAN EQUIPMENT PROFILES (EEP)

The following EEPs are supported:

F6 Profiles	A5 Profiles	D2 Profiles	D5 Profiles
F6-03-02	A5-02-05	D2-14-40	D5-00-01
	A5-04-01	D2-14-41	
	A5-04-03	D2-15-00	
	A5-06-02	D2-32-00	
	A5-06-03	D2-32-01	
	A5-07-01	D2-32-02	
	A5-07-03	D2-B1-00	
	A5-08-01		

F6 Profiles	A5 Profiles	D2 Profiles	D5 Profiles
	A5-08-02		
	A5-08-03		
	A5-09-04		
	A5-09-09		
	A5-12-00		
	A5-14-05		

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 let us know.

API

The API is used to onboard EnOcean Devices into the IoTC.

The most important features are:

- most recent data and signal telegrams from a device
- get past telegrams to get past health
- telegram statistic (e.g. count, last seen) for a device and per gateway
- list of connected ingress gateways
- persistent storage of onboarded device - if volume is selected.
- Include `friendlyID`, `location` or any `custom parameter` for each onboarded device
- All onboarded devices can be retrieved via `GET /backup` or uploaded via `POST /backup`.
- Open API Standard 3 supporting the automatic generation of clients in several languages.
- `Active flag` to enable/disable telegram processing for a particular device.

The API exposes a UI interface for your convenience. Once the IoTC connector has been deployed, the full API specification is available via the UI web Interface.

NGINX

NGINX is used as a proxy to protect the interface of the IoTC. The user is required to provide a certificate for usage.

A `Dockerfile` and corresponding dependencies (`start.sh` and `nginx.conf`) in `enocean/proxy` is provided incase the proxy needs to be rebuilt or customized.

redis

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

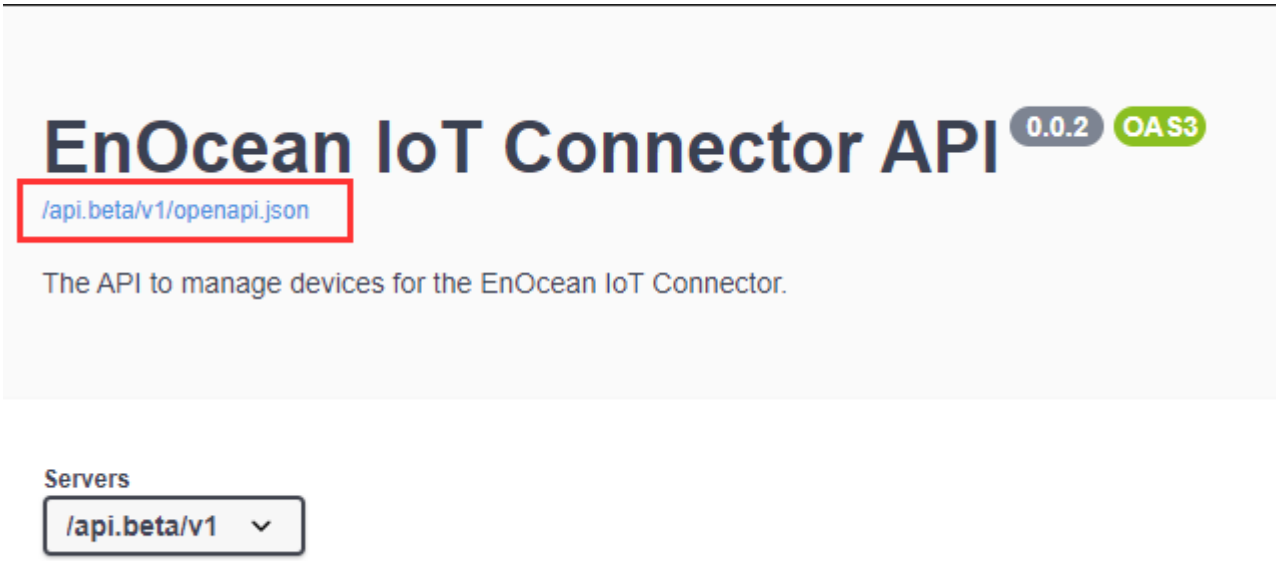
1.1.2 The API

The API is OpenAPI compliant, supporting the automatic generation of clients in several languages. The full API Specification is available here or via the web Interface, once the IoTC has been deployed.

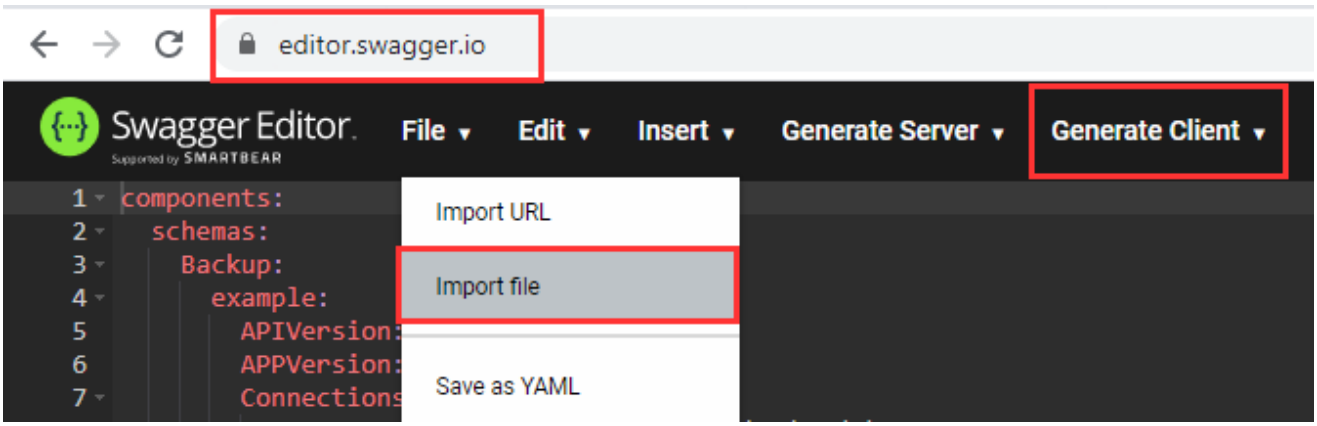
If you specified a volume storage at deployment then all changes done in the API will be persistent even after containers are restarted or updated.

Web UI of management API

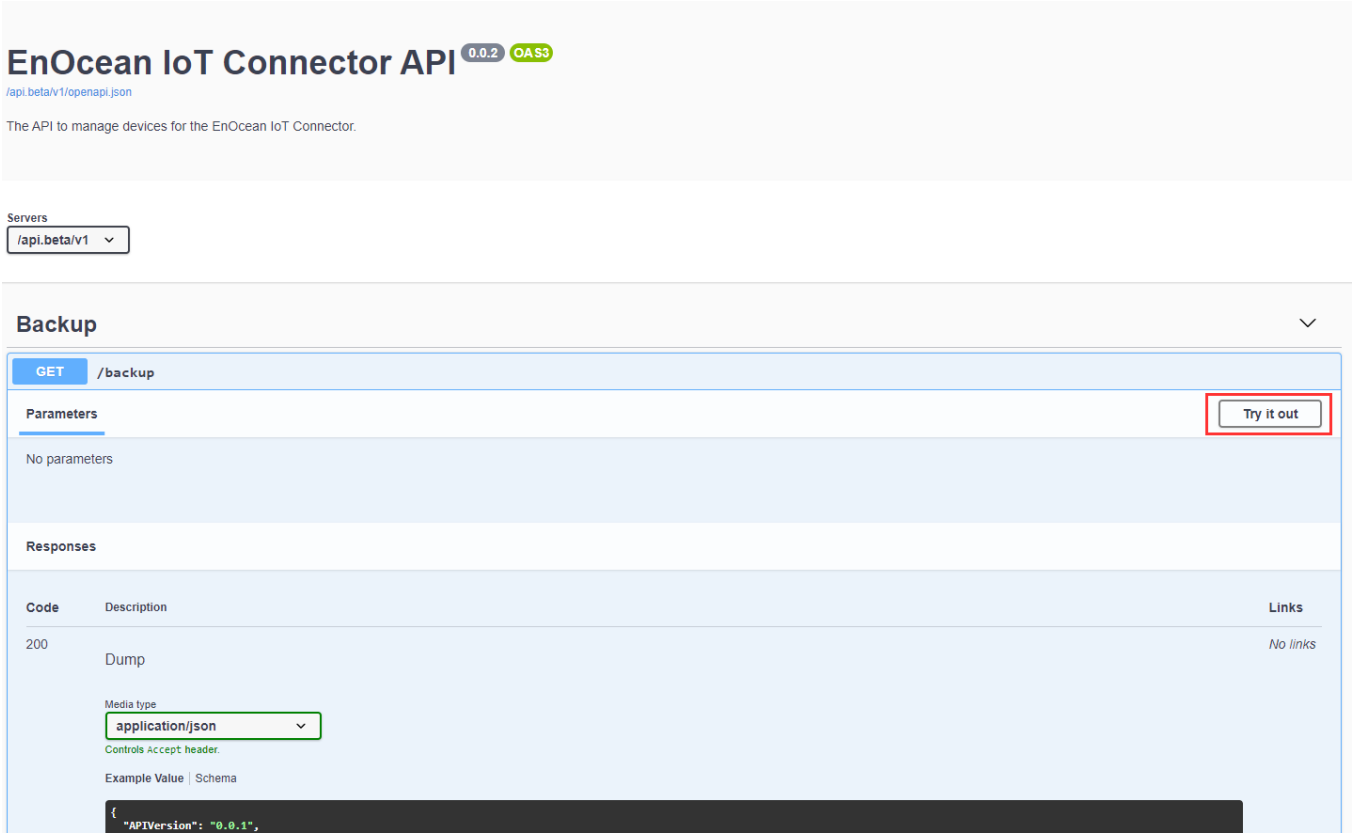
1. Opening the API url on a browser will display the API reference. The URL is `https://<hostname of the container group or IP address>:443`. Example: `https://192.167.1.1:443` or `https://myiotc.eastus.azurecontainer.io:443`
2. If you used a self-signed certificate and did not add it to your browser you will see a warning, please continue according to your web browser.
3. Login using the `BASIC_AUTH_USERNAME` & `BASIC_AUTH_PASSWORD` you specified in environmental variables.
4. The API complies with Open API Standard 3.
   a. Download the API Specification as JSON



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



5. You can use the `Try it out` function to execute any of the available commands.



Telegram statistics - sensor & gateway statistics

The API provides telegram statistics of the individual devices and per ingress gateway.

Gateway statistics

Calling `GET /gateways/metadata/statistics/telegrams` returns the statics per gateway


```
[
    {
    "device": {
      "hardwareDescriptor": "AP-305",
      "mac": "d01546c204a2",
      "softwareVersion": "8.7.1.1-8.7.1.1"
    },
    "stats": {
      "lastSeen": "1619210924",
      "notProcessed": 0,
      "succesfullyProcessed": 78662,
      "totalTelegramCount": 78662
    }
  },
  {
    "device": {
      "hardwareDescriptor": "AP-305",
      "mac": "24f27f551bf4",
      "softwareVersion": "8.7.1.0-8.7.1.0"
    },
    "stats": {
      "lastSeen": "1619210928",
      "notProcessed": 0,
      "succesfullyProcessed": 91526,
      "totalTelegramCount": 91526
    }
  }
]
```


EnOcean Device statistics

Calling `GET /devices/metadata/statistics/telegrams?sourceID=051b03c9&destinationID=FFFFFFFF` returns statistics for a individul EnOcean device `sourceID=051b03c9&destinationID=FFFFFFFF`.


```
[
  {
    "device": {
      "activeFlag": "true",
      "customTag": "",
      "destinationEurid": "ffffffff",
      "eep": "a5-09-09",
      "friendlyID": "co2_Hardware2",
      "isPTM": "false",
      "location": "Hardware 2",
      "sourceEurid": "051b03c9"
    },
    "stats": {
      "lastSeen": "1619210854",
      "notProcessed": 0,
      "succesfullyProcessed": 1057,
      "totalTelegramCount": 1057
    }
  }
]
```


The `stats` section is defined as:

```
TelegramStatistics:
      properties:
        lastSeen:
          description: Timestamp of last valid telegram from device in UTC seconds.
          type: string
        notProcessed:
          description: Count of not processed telegrams due to various reasons & NOT forwarded on egress.
          type: integer
        succesfullyProcessed:
          description: Count of succesfully processed telegrams & forwarded on egress.
          type: integer
        totalTelegramCount:
          description: Total count of received telegrams.
          type: integer
```

1.1.3 End-points

Available end-points are MQTT or Azure IoT Hub.

Output Format

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.

> **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`

Multisensor

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": {
            "security": [],
            "sensorHealth": [],
            "stats": [{
                "egressTime": "1611927479.169171"
            }]
        }
    },
    "raw": {
        "data": "d29fce800863b502a620",
        "sender": "04138bb4",
        "status": "80",
        "subTelNum": 0,
        "destination": "ffffffff",
        "rssi": 77,
        "securityLevel": 0,
        "timestamp": "1611927479.166352"
    }
}
```


CO2 sensor


```
{
    "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": {
            "security": [],
            "sensorHealth": [],
            "stats": [{
                "egressTime": "1611927535.0731573"
            }]
        }
    },
    "raw": {
        "data": "a500005008",
        "sender": "051b03c9",
        "status": "01",
        "subTelNum": 0,
        "destination": "ffffffff",
```

Each output JSON consist of three 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` -raw sensor health data included in the message and encoded as signal telegram
  - `meta` - meta information about the message added by the engine
- `raw` - raw message information
- `rssi` - radio signal strength information. Important to track radio quality

Sensor Health Information

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

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.

General operation can be checked by the `lastSeen` parameter provided by the API. Some devices have a periodic communication pattern. Checking deviations / fluctuations in the pattern can help to detect issues before quickly.

> **Note**
>
> EnOcean plans to provide a more automated Sensor health tracing and issue detection and reporting. Please see product roadmap.

1.1.4 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) can be run at common embedded platforms on the market e.g. RPi gen 4.

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

1.1.5 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)
- pyinstaller (pyinstaller,https://github.com/pyinstaller/pyinstaller,https://github.com/pyinstaller/pyinstaller/blob/develop/COPYING.txt)

1.1.6 License Agreement and Data Privacy

Please see the License agreement here.

Please see the Data privacy agreement here.

1.1.7 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 Support

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

Alternatively, contact support@enocean.com

We will aim to provide fast support for alpha customers. The current release of EnOcean IoT Connector is currently in beta, meaning testing, bug hunting & optimization is still ongoing. We thank you for you understanding.

1.2.1 Debugging

Console Log Messages

Main debug & info messages from the IoTC can be viewed in the log of the engine container. Other containers post messages to the console as well. To see these: e.g. open Docker Desktop -> go to Containers/Apps, find `local_deployment`, click on the line of interest. e.g. `local_deplomentproxy_1`.

The main engine log messages are:

- a gateway was added
- an onboarded sensor transmitted messages for the first time
- licensing information
- device transmitted with an unsupported EEP

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

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

Version: 0.1.0 0.1.0 0.1.0
...
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.
...
```

2. Deploy the IoTC

2.1 Deployment Notes

2.1.1 Overview of Environment Variables

To deploy the IoTC certain environment variable must be specified, these are listed below:

Environment Variable	Usage	Required?
IOT_LICENSE_KEY	IoTC license key. Contact your EnOcean sales partner.	**Yes**
IOT_ARUBA_USERNAME	Username used for the Aruba AP authentication.	**Yes**
IOT_ARUBA_PASSWORD	Password used for Aruba AP authentication.	**Yes**
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`	**Yes**
BASIC_AUTH_USERNAME	User name for basic authentication on the API interface.	**Yes**
BASIC_AUTH_PASSWORD	Password for basic authentication on the API interface.	**Yes**
IOT_AZURE_CONNSTRING	The *Connection String* to be use for sending data to the Azure IoT Hub.	This variable is required if the variable *IOT_AZURE_ENABLE* is set.
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.**	No
MQTT_CONNSTRING	The *Connection String* to be use for publishing data to an MQTT broker.	This variable is required if the variable *MQTT_LOCAL_EGRESS_ENABLE* is set.
MQTT_LOCAL_EGRESS_ENABLE	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.**	No

2.1.2 Overview of required Secrets

Secret	Usage	Required?
secret-proxy-certificate	Certificate for the NGINX proxy to protect IoTC interfaces.	**Yes**
secret-proxy-key	Private key of the certificate for the NGINX proxy.	**Yes**

2.1.3 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)

Service	Description	Port
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.

2.1.4 License key

To deploy the IoTC a license key is required. Please contact EnOcean for a license key for a trial or commercial usage.

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.

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

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.

2.2 Deploy and Connect Devices

2.2.1 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 locally installed Docker or deploy in the Microsoft Azure Container instances. Deployment on other cloud platforms is also possible but has not been tested.

Local 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_ARUBA_USERNAME & IOT_ARUBA_PASSWORD**

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

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

    environment:
        - IOT_ARUBA_USERNAME=   #enter new username for Aruba AP         connection to IoTC. e.g. user1
        - IOT_ARUBA_PASSWORD=   #enter new password for Aruba 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/testing_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

List **IOT_AZURE_CONNSTRING & IOT_AZURE_ENABLE**.

```
    engine:
        image: enocean/iotconnector_engine:latest
            environment:
            # Comment this section out, should Azure egress not be desired.
                - IOT_AZURE_ENABLE=1
```

2.2.2 2. Connect Ingress Gateways

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

Connect Aruba AP

Check that the Aruba AP corresponds to the required SW and HW.

1. Upload to the Aruba APs the `*.pem` file you have prepared.
2. Connect the Aruba AP.
3. You can check if the AP got connected via the management API by using `GET /gateways`. You can use the build in Web UI or your HTTPS client.

   Response body example:

```
[
    {
      "hardwareDescriptor": "AP-305",
      "mac": "24f27fca1ba4",
      "softwareVersion": "8.7.1.0-8.7.1.0"
    },
    {
      "hardwareDescriptor": "AP-505",
      "mac": "1c28afc2950a",
      "softwareVersion": "8.8.0.0-8.8.0.0"
    }
]
```

   Or check the engine console.

> **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.

2.2.3 3. Onboard devices using the API

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

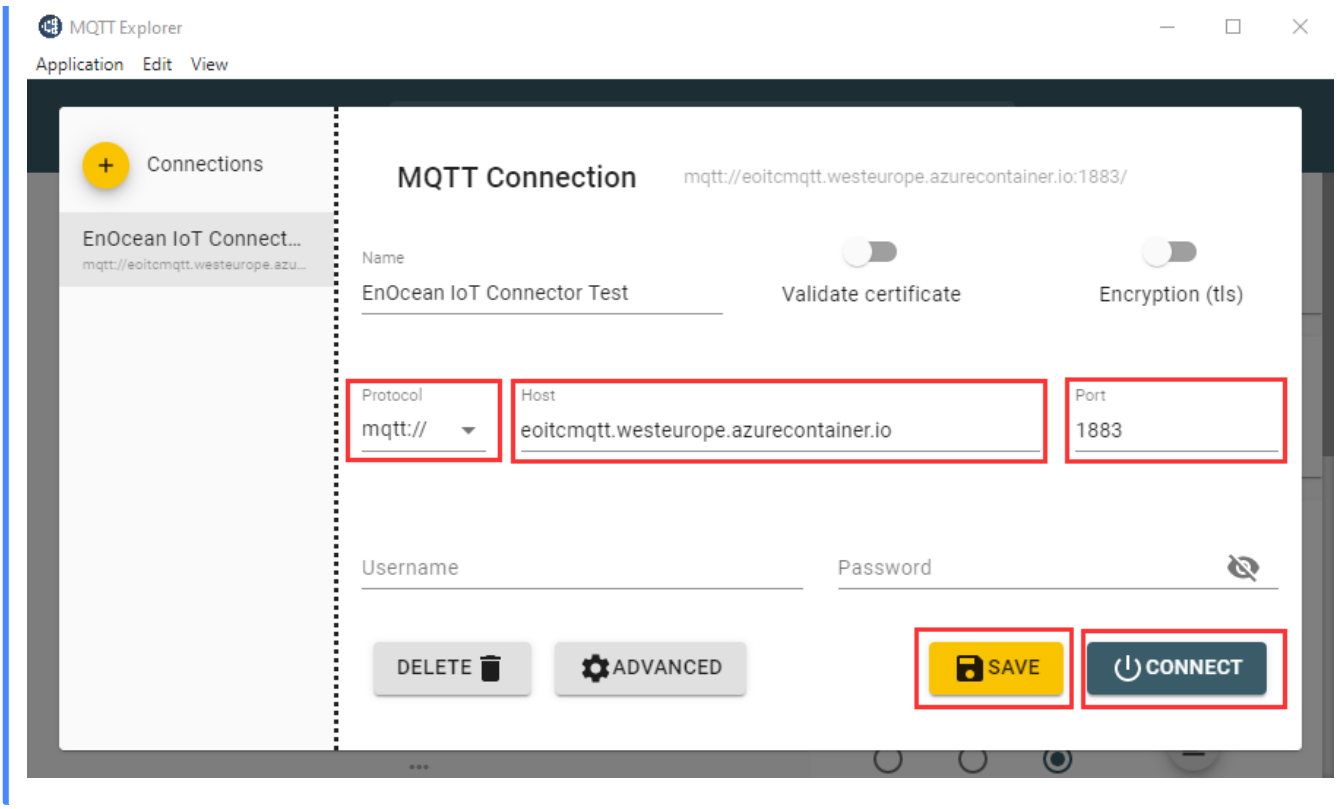
   Additionally specify a `friendlyID` and `location` of the sensor.

   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.



3. Additional Notes

3.1 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.**

3.1.1 Generate private key for CA authority:

For Windows users:

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

For Linux users:

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

Complete the fields with the information corresponding to your organization.

3.1.2 Generate root certificate

```
docker run -it --rm -v ${PWD}:/export frapsoft/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.

3.1.3 Generate a key for the certificate going into the connector

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

3.1.4 Generate a CSR for the connector

```
docker run -it --rm -v ${PWD}:/export frapsoft/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.

3.1.5 Create the .ext file

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

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
#keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names
subjectKeyIdentifier = hash

[alt_names]
DNS.1 = localhost
```

Edit the `localhost.ext` file to match your domain. Make sure the `DNS.1` matches the hostname of your deployment.

3.1.6 Generate a certificate from CSR for the connector

```
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
```

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

3.2 Notes for Aruba APs

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

3.2.2 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*.

3.2.3 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

```
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_ARUBA_USERNAME>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # password <aruba_password set using IOT_ARUBA_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...
```

Aruba OS 8.7.0.0

```
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") # username <aruba_username set using IOT_ARUBA_USERNAME>
aa:bb:cc:dd:ee:ff (IoT Transport Profile "myProfile") # password <aruba_password set using IOT_ARUBA_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.
```

3.2.4 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>
```

4. Release Notes

4.1 Documentation Changes

4.1.1 18.06.2021

- change style to mkdocs and moved to rtd domain
- added tabs for styles
- added download section
- added full API Documentation in swagger

4.1.2 14.06.2021

- update for debug and troubleshooting information on Aruba APs.

4.1.3 10.05.2021

- depends on correction in docker compose files
- Specific documentation for RPi / Linux users
- Specific commands for AOS 8.7.x.x included
- Updated information on licensing
- Updated information on generation for API Source code
- Updated energy profiles of Aruba APs

4.2 EnOcean IoT Connector - Beta 0.2.0 Not released

4.2.1 General

Features

- New structure of the technical product documentation
- Processing Aruba Health messages. Showing status of connected USB and AP Status.
- Create HTML from markdown
- Technical documentation is now available on: https://iotconnector-docs.readthedocs.io/
- Added UI with extra container to simulate the incoming traffic and gateways
- Solution now available on Azure Marketplace https://azuremarketplace.microsoft.com/de-de/marketplace/apps/enoceangmbh1606401683119.iotc-saas
- Automated build scripts
- Persistent storage all configuration & runtime enable. After IoTC

Bugs

- Some logging messages where using root logger instead of instance logging
- Deleted devices are removed from licensing count at runtime
- Workaround for arm/v7 platform, because it does not correctly hash ca-certificates
- Introduced Technical Documentation Versioning

4.2.2 API Container

Bugs

- UI redirect includes a slash /api.beta/v1/ui/ at the end to avoid unnecessary redirect
- Sanitized string outputs / inputs

4.2.3 Engine Container

Bugs

- Unknown EEPs handled gracefully

Features

- EEP D2-14-52 supported
- Console Debug Output/Log

4.2.4 Ingress Container

Bugs

- Support & Documentation extended for older Aruba OS Versions (8.7.x)
- Aruba APs will appear in the gateway list even without EnOcean traffic (3-5 min delay) - feature based on Aruba AP.

Features

- Change password for from IOT_ARUBA_ *to INGRESS_*
- Support for ESP3 Packet Type 10 - required for Japan region

• Support for generic APs on ingress

4.3 EnOcean IoT Connector - Version Beta 0.1.0

Bug

- Switched fields friendlyID and location on output.

Features

- Cryptolens Licensing added.
- Optimize web sockets and use Secure web sockets.
- Allow users to provide a certificate + key for NGINX.
- Allow APs to connect using secure web sockets.
- Receive respective sensor health data (parsed signal telegram) triggered by signal telegram
- Add, remove & update sensors via API.
- Add tags to onboarded sensors
- Enable activated flag for devices
- Authenticate with the API.
- Get Gateways list via the API.
- Add CT Clamp EEPs
- Query last 5 data telegrams incl. RSSI
- Get telegram statistics per Gateway / per Device.

Container Hotfixes Version Beta 0.1.0

API HOTFIXES

API - Hotfix 0.1.3

- PUT devices fixed

API - Hotfix 0.1.2

- Source ID forced conversion to lowercase
- Boolean correction from Redis without quotes

API - Hotfix 0.1.1

- REDIS save of device configuration on new device interaction

ENGINE HOTFIXES

Engine - Bugfix 0.1.3

- Removing of deleted devices at runtime
- Additional debug messages

Engine - Hotfix 0.1.2

- Additional debug messages for device onboarding
- Rehashing of user certificates after update forced
- Processing empty message as hello to complete onboarding

Engine - Hotfix 0.1.1

- Improved licensing performance

INGRESS HOTFIXES

Ingress - Bugfix 0.1.3

- Rehashing of user certificates after update forced

Ingress - Hotfix 0.1.2

- Processing empty message as hello to complete onboarding
- Support for AOS 8.7.x.x - Client ID is optional

Ingress - Hotfix 0.1.1

- Improved licensing performance, retry on fail