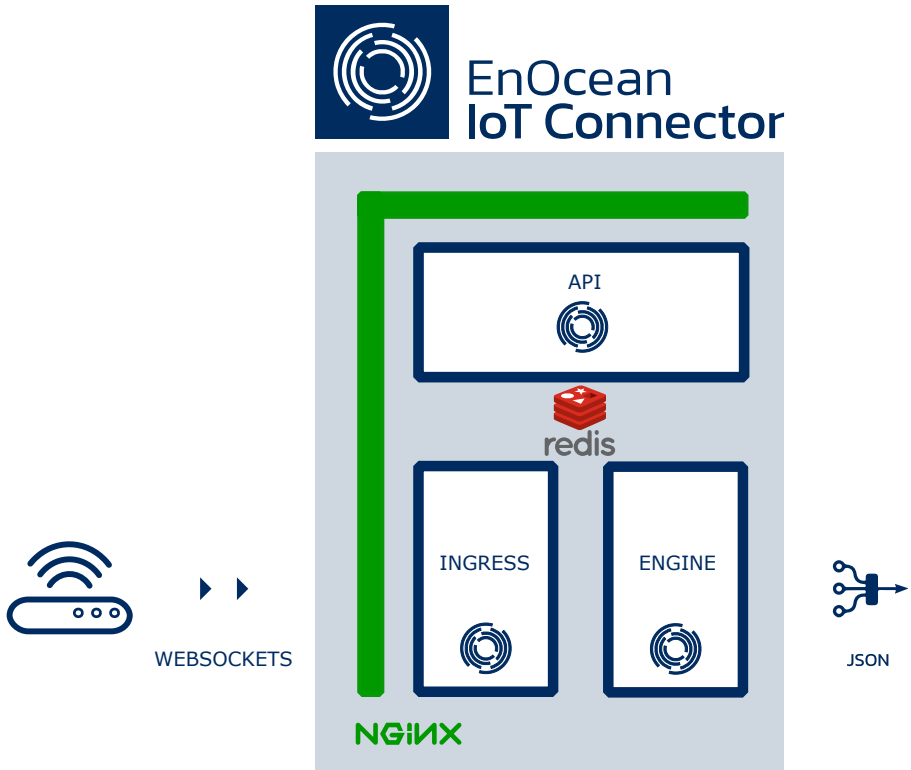
EnOcean IoT Connector

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` / `2021-06-25 15:07:02+02:00`

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		
	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.

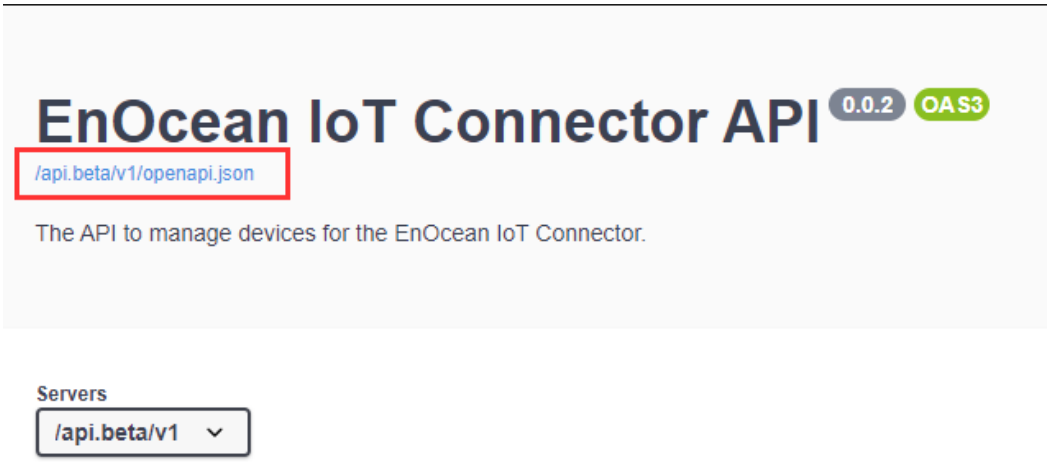
The API

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.

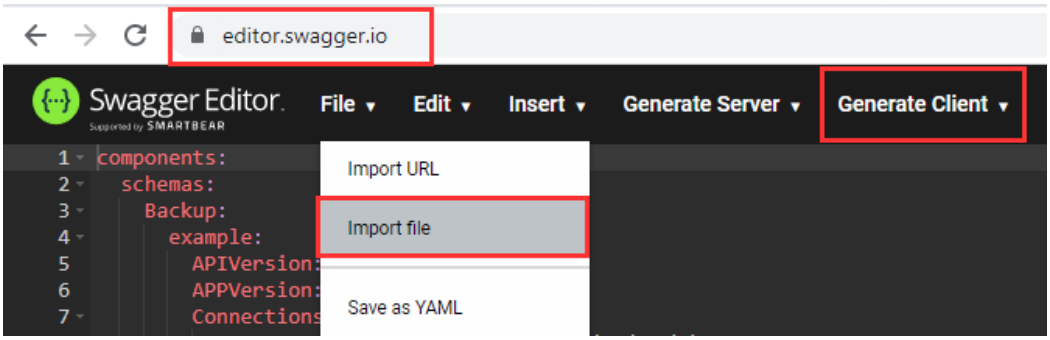
If you specified a volume storage at deployment then all changes done in the API will be persistent even after containers are restarted or updated.

Web UI of management API

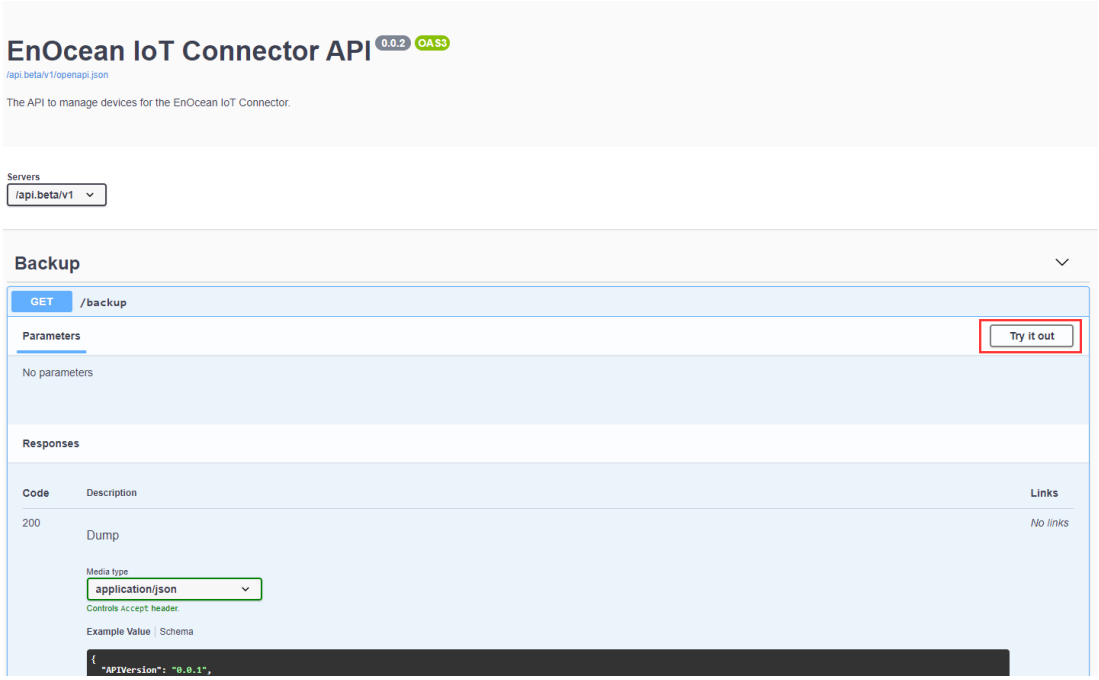
1. Opening the API url on a browser will display the API reference. The URL is `https://<hostname of the container group or IP address>:443`. Example: `https://192.167.1.1:443` or `https://myiotc.eastus.azurecontainer.io:443`
2. If you used a self-signed certificate and did not add it to your browser 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
```

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"
    }
}
```

```
{
    "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",
        "rssi": 80,
        "securityLevel": 0,
        "timestamp": "1611927535.0714777"
    }
}
```


Switch Module

PTM215 battery-less switch module


```
{
    "sensor": {
        "friendlyId": "switch1",
        "id": "feee14ab",
        "location": "Entrance"
    },
    "telemetry": {
        "data": [
            [{
                "key": "energybow",
                "value": "released",
```

```
                "meaning": "Energy Bow released"
            }]
        ],
        "signal": [],
        "meta": {
            "security": [],
            "sensorHealth": [],
            "stats": [{
                "egressTime": "1611927462.4711452"
            }]
        }
    },
    "raw": {
        "data": "f600",
        "sender": "feee14ab",
        "status": "20",
        "subTelNum": 0,
        "destination": "ffffffff",
        "rssi": 71,
        "securityLevel": 0,
        "timestamp": "1611927462.469978"
    }
}
```


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.

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.

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)