
Flow Nexus Documentation

Jonas Remmert, Akarshan Kapoor

Jul 08, 2024

CONTENTS

Contents	i
1 Motivation	2
2 Vision	3
3 Overview	4
3.1 Features	4
3.2 Key Components	4
3.2.1 IoT Devices	4
3.2.2 LwM2M Server	5
4 Architecture	6
4.1 Django	6
4.1.1 Build and Run	6
4.1.2 Database Model	7
4.2 Leshan	8
4.2.1 Build and Run	8
4.3 Communication, Interfaces	8
4.3.1 Data Flow: Endpoint -> Backend	9
4.3.2 Data Flow: Backend -> Endpoint	10
4.3.3 Asynchronous Communication	10
4.3.4 Example Communication	11
5 Build and Deploy	12
5.1 Development Setup	12
5.2 Container Environment	13
6 Simulation	15
6.1 IoT Devices with Zephyr	15
6.1.1 LwM2M Client Simulation	15
7 Documentation	17
7.1 Build the documentation	17
8 Glossary	18
8.1 Additional Terms and Definitions	18
9 Internal Django	19
10 Weblog	20
10.1 Update Documentation & Add Weblog	20
10.2 Project Starts	20
Index	21

Welcome to Flow Nexus, your comprehensive solution for small to medium scale IoT deployments. This platform is designed to streamline the setup, configuration, and use of IoT devices, making it easy to manage your IoT ecosystem.

MOTIVATION

This project aims to provide a minimal and standalone Open Source IoT platform. The focus is on systems with a deployment of up to a few thousand devices; it does not aim to be scalable to millions of devices. The goal is to build a system that receives, for example, regular temperature samples from remote IoT devices. These temperature samples will be sent to a backend and stored in a database. The data is then visualized in a web application.

VISION

Our vision for Flow Nexus is to create a user-friendly, efficient, and robust Open Source platform for IoT management. We aim to simplify the complexities of IoT deployments, making them accessible and manageable for businesses of all sizes.

OVERVIEW

This documentation describes a local server-based IoT system that leverages the Lightweight Machine to Machine (LwM2M) protocol to communicate between IoT devices running Zephyr OS and a backend server using Django. The system does not depend on external cloud services and is designed to operate fully within a local environment.

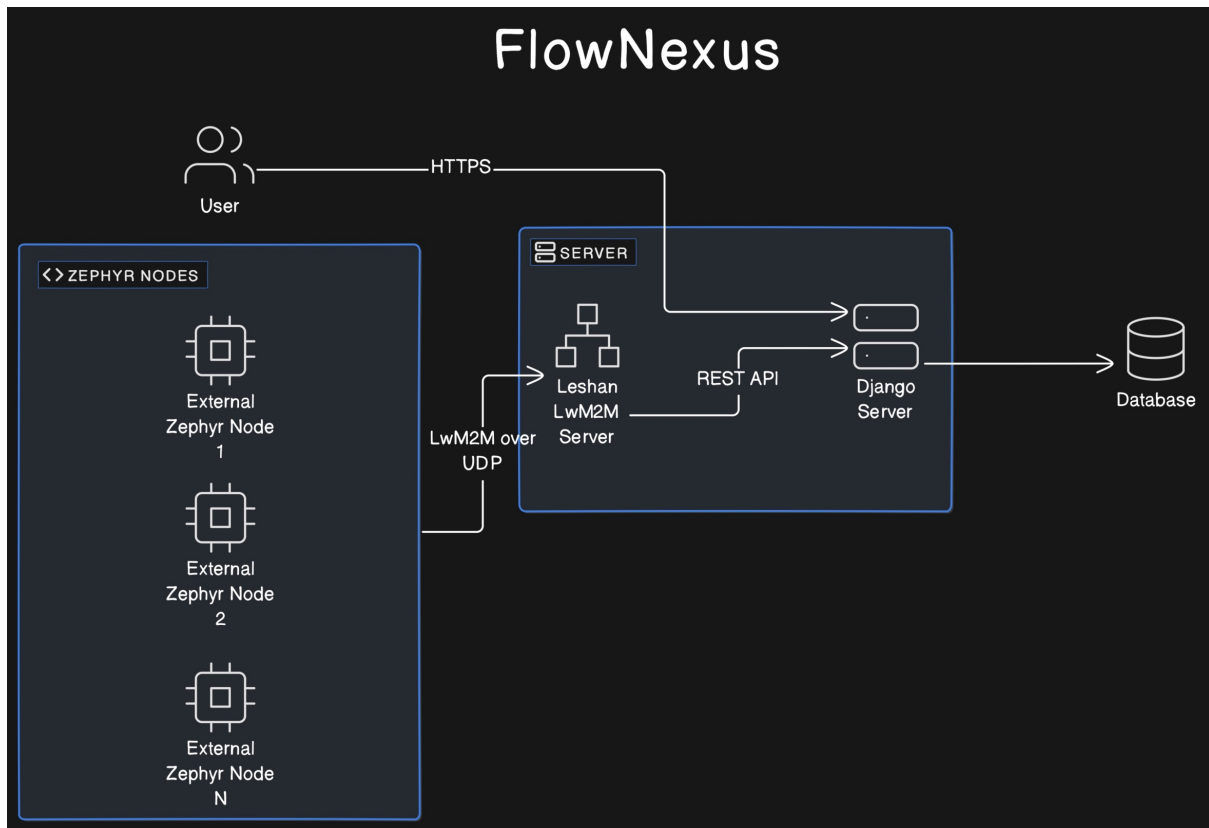
3.1 Features

- No dependencies of external services like AWS, MQTT brokers or similar. The system has to be able to run in a local environment.
- The main focus is the LwM2M protocol and the communication between Zephyr and the server.
- The system should show how to add data to a database and visualize it in a web application.
- The web application has to support a secure login and basic user management.
- Certificate based authentication and encryption.
- Server to read and observe resources from IoT Device.
- Server to write resources to IoT Devices.
- OTA Update.
- Receive logs from IoT Devices.

3.2 Key Components

3.2.1 IoT Devices

These are typically resource-constrained devices (limited energy and storage capabilities) that run Zephyr OS. They communicate with the server using the UDP protocol to ensure constant connectivity, even in low power states. These devices are programmed to handle tasks like temperature sensing, location tracking, or any other telemetry based on Zephyr applications. They communicate via the UDP protocol to maintain a continuous link without needing frequent reconnections. The IoT device has to support LwM2M. Mainly systems that run Zephyr should be compatible. The application can e.g. be used with a dedicated nRF9160 device or via a simulation like `native_sim` or in Renode. The nRF9160 is a low power LTE-M and NB-IoT SoC that runs Zephyr OS. UDP is used as transport protocol as it allows to keep devices connected even when the device is sleeping for extended periods (TCP would require a new connection setup typically after a few minutes)



3.2.2 LwM2M Server

The LwM2M server plays a critical role in managing communications with IoT devices. It handles data transmission, device monitoring, and control commands, ensuring that devices can report their telemetry data, receive configuration updates, and execute commands sent by the server. The server's efficient management of these tasks is essential for maintaining reliable communication with numerous IoT devices, especially those with limited power and processing resources.

ARCHITECTURE

This section describes the architecture of individual components and how they interact with each other.

4.1 Django

The Django server hosts the REST API, manages the database, and provides a web interface for data visualization and user management. This server acts as the backend infrastructure that supports the entire IoT ecosystem by:

- **REST API:** Facilitates communication between the IoT devices and the server. The API endpoints handle requests for data uploads from devices, send control commands, and provide access to stored telemetry data.
- **Database Management:** Stores all the telemetry data, device information, user accounts, and configuration settings. The database ensures that data is organized, searchable, and retrievable in an efficient manner.
- **Web Interface:** Offers a user-friendly interface for data visualization and management. Through this interface, users can monitor device status, visualize telemetry data in real-time, manage user accounts, and configure device settings.

4.1.1 Build and Run

The Django server can also run locally, without the need of a docker container. Make sure to create a virtual environment and install the requirements:

```
host:~$ source venv/bin/activate
host:~$ cd flownexus_workspace/lwm2m_server/server/django
host:lwm2m_server/server/django$ pip install -r requirements.txt
host:lwm2m_server/server/django$ ./django_start.sh
```

The Django server should now be up and running under the following URL: <http://localhost:8000/admin>. The admin login is `admin` and the password

Run Unit Tests

There are unit tests, that test the deserializer, which parses the Json payload from the ReST API. You can run the unit tests with the following command:

```
host:~/flownexus_workspace/lwm2m_server/server/django$ python manage.py test_
↳sensordata
Found 2 test(s).
Creating test database for alias 'default'...
-----
```

(continues on next page)

(continued from previous page)

```
Ran 2 tests in 0.008s
```

```
OK
```

```
Destroying test database for alias 'default'...
```

4.1.2 Database Model

The database model is the core of the Django server. It aims to store information according to the LwM2M resource model. The advantage is that data can be stored in a generic way and the server can be extended with new resources without changing the database schema.

The server application logic only has to handle higher level events. Those events are situations where multiple resources are associated (e.g. Temperature, Pressure, Humidity, Acceleration). Those multiple resources are linked in the event, together with a timestamp. The event itself is represented by the database model in a generic way, several custom event types can be created by the application logic.

An Entity Relationship Diagram (ERD) is a visual representation of the database schema. It is automatically generated from the Django models. `sensordata` is the Django app that contains the application logic.

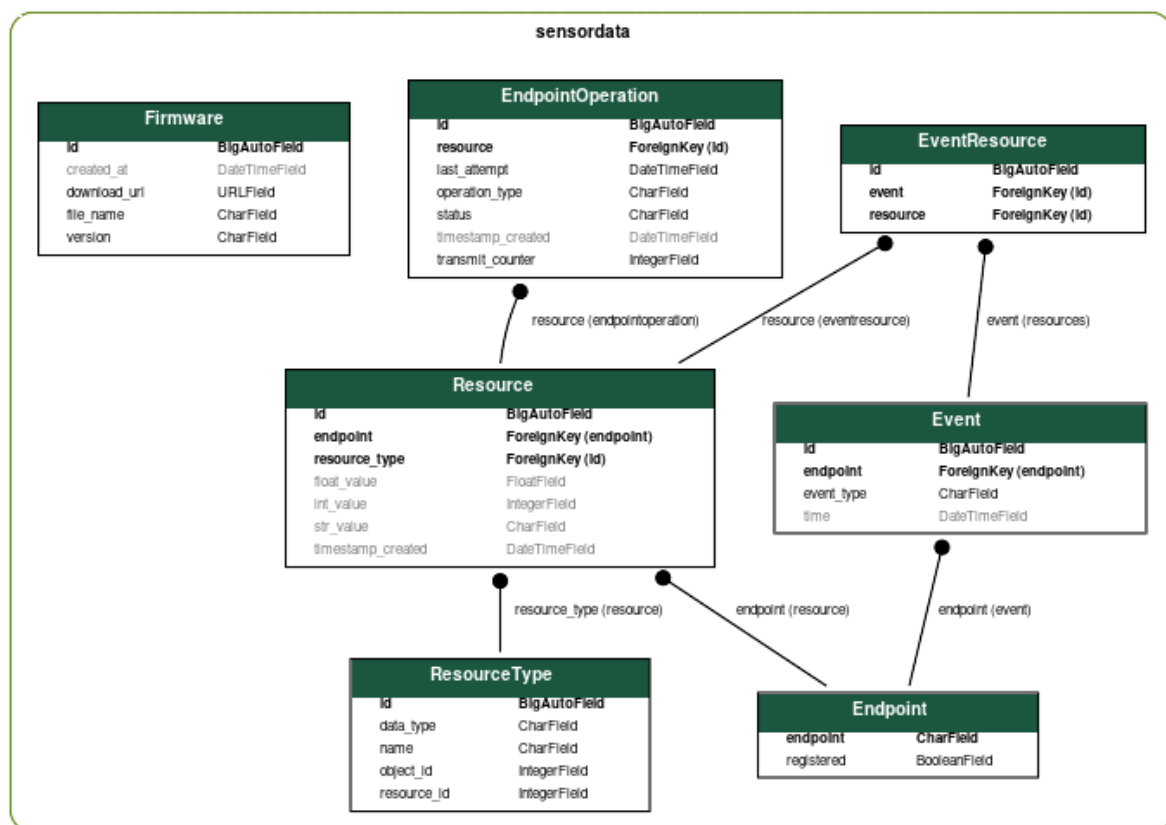


Fig. 1: Entity Relationship Diagram generated from Django models

Endpoint

IoT devices using the LwM2M protocol in the network, identifiable by a unique name, e.g. `urn:imei:123456789012345`.

ResourceType

Defines resource data points comprehensively, annotating each with a unique object-resource ID combination, a descriptive name, and specifying the expected data type.

Resource

Captures individual data values from IoT devices, annotated with timestamps, applicable data types, and linked to both the device and resource type for which the data pertains.

Event

Serves as a collection point for significant occurrences reported by devices, including composite events defined by enclosing object IDs. The server application logic has to generate events based on matching timestamps or received composite notifications from devices. Although individual resources within an event may have different timestamps, the event itself encapsulates a single timestamp.

EventResource

Acts as a junction table forming a many-to-many relationship between events and their constituent resources, enabling flexible association without direct modification to the core events or resources tables.

EndpointOperation

Represents actionable commands or processes targeted at endpoints, tracking the operation type, status, and scheduling through timestamps, also detailing the transmission attempts and last action.

Firmware

Stores metadata about firmware binaries that are available for devices to download and install. Each record includes a version identifier, the name of the file, a URL from where the device can retrieve the firmware, and timestamps for tracking when each firmware record was created and last updated.

4.2 Leshan

4.2.1 Build and Run

The Leshan server can also run locally, without the need of a docker container.

```
host:~$ sudo apt update
host:~$ sudo apt install openjdk-17-jdk maven
host:~$ source venv/bin/activate
host:~$ cd flownexus_workspace/lwm2m_server/server/leshan
host:lwm2m_server/server/leshan$ ./leshan_build_run.sh
```

The Leshan server should now be up and running under the following URL: <http://localhost:8080>.

4.3 Communication, Interfaces

The communication between IoT devices and Leshan is specified by the OMA LwM2M standard:

- [LwM2M core specification v1.1.1](#)
- [LwM2M transport binding v1.1.1](#)

The standard describes how the LwM2M server (Leshan) works, however, it does not describe how to connect a backend server to Leshan. The backend is responsible for storing the data in a database and implementing application logic. A frontend can access the data in the database and visualize outward facing user interfaces. Leshan acts as a gateway between Endpoints and the backend. There should be no application specific logic implemented in Leshan.

In order to communicate and exchange data, both components (Leshan LwM2M Server and Django) post data to each other's ReST APIs. Communication is typically triggered by IoT devices sending data or the user/application requesting data from devices.

4.3.1 Data Flow: Endpoint -> Backend

All communication from Endpoints to the server flows through Leshan. Leshan interprets this data according to the LwM2M protocol and generates a ReST API call to the backend server. The backend server then stores the data in the database.

Django hosts the ReST API that Leshan posts to. Serializers, a part of Django, deserialize the incoming data and store it in the database according to the database model.

Leshan Data Format

There are two types of data that Leshan sends to the backend, single resource and composite resource format. The two ReST API endpoints that Leshan posts to are available under the following URLs:

- /leshan_api/resource/single: single resource format.
- /leshan_api/resource/composite: composite resource format.

For more details, please check the [Internal Django API documentation](#).

Listing 1: Single Resource Format (3303/0/5700)

```
{
  "ep": "qemu_x86",
  "obj_id": 3303,
  "val": {
    "kind": "singleResource",
    "id": 5700,
    "type": "FLOAT",
    "value": "24.899181214836236"
  }
}
```

Listing 2: Composite Resource Format (3/0/0..17)

```
{
  "ep" : "qemu_x86",
  "val" : {
    "instances" : [ {
      "kind" : "instance",
      "resources" : [ {
        "kind" : "singleResource",
        "id" : 0,
        "type" : "STRING",
        "value" : "Zephyr"
      }, {
        "kind" : "multiResource",
        "values" : {
          "0" : "1",
          "1" : "5"
        }
      },
      {
        "kind" : "singleResource",
        "id" : 17,
        "type" : "STRING",
        "value" : "qemu_x86"
      }
    ] ,
    "id" : 0
  } ],
}
```

(continues on next page)

(continued from previous page)

```

    "kind" : "obj",
    "id" : 3
  }
}
```

The marked lines in the composite resource format show where Object ID, Instance ID and Resource ID are located. A composite resource format can consist of multiple Object IDs.

Warning: Currently multiResources are not supported and will be ignored. MultiResource datatypes are e.g. used for Voltage range (min, max).

Registration Updates

4.3.2 Data Flow: Backend -> Endpoint

Endpoints often operate in queue mode, meaning they are not always online. The LwM2M Server is aware of the current status of a device (Online/Offline) and communicates this status to the backend server. Leshan does not queue pending data that should be sent to the device when it comes online. The backend server must handle this by itself so it has to have a representation of the current status of each device as well as the data to be send. The resource table **DeviceOperation** is used to store pending operations that should be sent to the endpoint while it is online.

Once an endpoint updates it's registration (LwM2M Update Operation) Leshan notifies the backend. The backend checks the **DeviceOperation** table for pending operations and sends them to the device by posting to the Leshan hosted ReST API. Leshan keeps the post call open until the device acknowledges the operation or a timeout is generated. Endpoints can be slow to respond (several Seconds), so the backend has to handle the ReST API call in an asynchronous manner. By only sending data to endpoints while they are online, the backend can be sure that the ReST API calls are not open for a long time.

4.3.3 Asynchronous Communication

Given that endpoints are comparably slow to respond, handling communication asynchronously is essential for efficient operation. This can be effectively managed using Celery, a distributed task queue. When Leshan notifies the backend of an endpoint status update, Celery can be used to handle the long-running API calls, ensuring that the backend remains responsive and scalable. As the backend communicates with many endpoints simultaneously, an efficient queuing mechanism is essential to ensure that the system remains responsive and scalable.

Before the backend executes the API call, it updates the endpointOperation status to **SENDING**, indicating an ongoing operation.

Once the API call is complete the database will be updated with the result (e.g. **CONFIRMED**, **FAILED**, **QUEUED**) depending on the result of the request. The **FAILED** status is assigned after 3 attempts. Retransmissions are triggered when the endpoint updates it's registration the next time.

4.3.4 Example Communication

The following example shows how the backend server can send a firmware download link resource **Package URI 5/0/1** to an endpoint:

1. User creates new **DeviceOperation**: resource path 5/0/1, value **https://url.com/fw.bin**.
2. Backend checks endpoint online status.
3. If endpoint is offline, no further action is taken right away.
4. Endpoint comes online, Leshan sends update to the backend.
5. Backend checks **DeviceOperation** table for pending operations for the endpoint.
6. Finds pending operation, send resource to endpoint via the Leshan ReST API.
7. Pending operation is marked **completed** if the endpoint acknowledges the operation.

BUILD AND DEPLOY

5.1 Development Setup

The build instructions in the documentation are tested for a native Linux Machine. For MacOS or Windows consider creating a docker container build. One of the developers uses the following *devcontainer.json* build environment:

```
{
  "name": "Ubuntu",
  "image": "mcr.microsoft.com/devcontainers/base:jammy",
  "runArgs": [
    "--cap-add=NET_ADMIN",
    "--cap-add=MKNOD",
    "--device=/dev/net/tun",
    "--sysctl=net.ipv6.conf.all.disable_ipv6=0",
    "--sysctl=net.ipv6.conf.default.disable_ipv6=0"
  ],
  "postCreateCommand": "apt-get update && apt-get install -y iproute2 && echo 'IPv6 is enabled.'",
  "remoteUser": "root"
}
```

Before you we start with any development here are a few things you should get configured:

- Get the Zephyr SDK downloaded and configured in your root directory. You can find the instructions [here](#).
- Setup a virtual environment for the project.

```
host:~$ sudo apt update && sudo apt upgrade
host:~$ sudo apt install python3-pip python3.10-venv
host:~$ python3.10 -m venv venv
host:~$ source venv/bin/activate
host:~$ pip install --upgrade pip && pip install west
host:~$ west init -m https://github.com/jonas-rem/lwm2m_server --mr main flownexus_
↪workspace
host:~$ cd flownexus_workspace
host:~/flownexus_workspace$ west update
```

5.2 Container Environment

Both components run in a Docker container. The Leshan server is running in a `openjdk:17-slim` container and the Django server is running in a `python:3.11-slim` container. This allows for an easy and reproducible setup of the server.

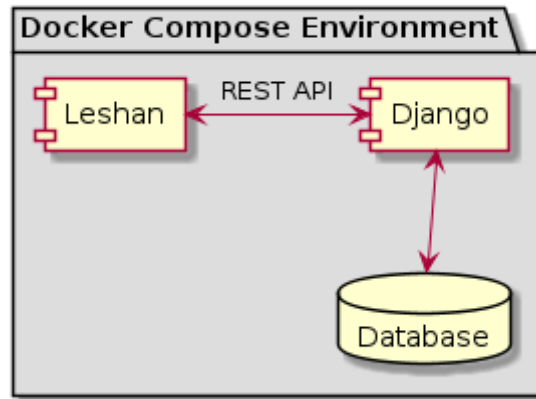


Fig. 1: Both components running in one machine using Docker Compose

The following diagram shows the Docker Compose environment. The file `docker-compose.yml` defines the services and their configuration. The file `Dockerfile.leshan` defines the Leshan container and the file `Dockerfile.django` defines the Django container.

Warning: Make sure to change the password to the admin console as well as other settings like `SECRET_KEY`, `DEBUG` flag in a production environment!

The container can be build and started with the following commands:

```

host:~/flownexus_workspace/lwm2m_server/server$ docker compose build
[+] Building 0.5s (20/20) FINISHED                                docker:default
=> [leshan internal] load build definition from Dockerfile.leshan    0.0s
=> [leshan internal] load metadata for docker.io/library/openjdk:17-slim 0.4s
=> [django internal] load build definition from Dockerfile.django    0.0s
=> [django internal] load metadata for docker.io/library/python:3.11-sli 0.4s
=> [leshan 1/5] FROM docker.io/library/openjdk:17-slim@sha256:aaa3b3cb27 0.0s
=> [django 1/5] FROM docker.io/library/python:3.11-slim@sha256:d11b9bd5e 0.0s
=> CACHED [leshan 2/5] WORKDIR /leshan                               0.0s
=> CACHED [leshan 3/5] COPY . /leshan/                               0.0s
=> CACHED [leshan 4/5] RUN apt-get update && apt-get install -y mave 0.0s
=> CACHED [leshan 5/5] RUN chmod +x /leshan/leshan_build_run.sh     0.0s
=> => exporting layers                                              0.0s
=> => writing image sha256:a017577ba2b175374148f5c3f128ac117ba5436ceaeff 0.0s
=> => naming to docker.io/library/server-leshan                     0.0s
=> CACHED [django 2/5] WORKDIR /django                               0.0s
=> CACHED [django 3/5] COPY . /django/                               0.0s
=> CACHED [django 4/5] RUN pip install --no-cache-dir -r /django/require 0.0s
=> CACHED [django 5/5] RUN chmod +x /django/django_start.sh        0.0s
=> => writing image sha256:1c88f1227753b08cf994c4e61d5cdcf97d68f260c99ad 0.0s
=> => naming to docker.io/library/server-django                     0.0s

```



```
host:~/flownexus_workspace/lwm2m_server/server$ docker compose up
[+] Running 2/0
  ✓ Container server-leshan-1 Created                                0.0s
  ✓ Container server-django-1 Created                                0.0s
Attaching to django-1, leshan-1
[..]
django-1 | Starting development server at http://0.0.0.0:8000/
leshan-1 | [main] INFO org.eclipse.leshan.server.LeshanServer - CoAP over UDP
↳ endpoint based on Californium library available at coap://0.0.0.0:5683.
leshan-1 | LeshanServer started
^CGracefully stopping... (press Ctrl+C again to force)
[+] Stopping 2/2
  ✓ Container server-django-1 Stopped                                10.3s
  ✓ Container server-leshan-1 Stopped                                10.5s
```

SIMULATION

6.1 IoT Devices with Zephyr

As device management protocol LwM2M is used, Zephyr offers a LwM2M client at `subsys/net/lib/lwm2m`. This LwM2M client sample application implements the LwM2M library and establishes a connection to an LwM2M server. The example can be build with the following command:

```
host:~/flownexus_workspace/lwm2m_server$ west build -b nrf9161dk_nrf9160_ns fw_test/  
↳lwm2m_client -p  
host:~/flownexus_workspace/lwm2m_server$ west flash --recover
```

6.1.1 LwM2M Client Simulation

The Zephyr application can run in simulation mode. This allows to test all components locally. Once leshan and Zephyr are running, the Zephyr application can be started in emulation with the following command:

```
host:~/flownexus_workspace/lwm2m_server$ ./zephyr_build_run_sim.sh  
*** Booting nRF Connect SDK zephyr-v3.5.0-3024-g7c3e830729b7 ***  
[00:00:00.000,000] <dbg> net_lwm2m_engine: lwm2m_engine_init: LWM2M engine socket_  
↳receive thread started  
[00:00:00.000,000] <dbg> net_lwm2m_obj_security: security_create: Create LWM2M_  
↳security instance: 0  
[00:00:00.000,000] <dbg> net_lwm2m_obj_server: server_create: Create LWM2M server_  
↳instance: 0  
[00:00:00.000,000] <dbg> net_lwm2m_obj_device: device_create: Create LWM2M device_  
↳instance: 0  
[00:00:00.010,000] <dbg> net_lwm2m_obj_firmware: firmware_create: Create LWM2M_  
↳firmware instance: 0  
[00:00:00.010,000] <inf> net_config: Initializing network  
[00:00:00.010,000] <inf> net_config: IPv4 address: 192.0.2.1
```

You should see the following output in the docker console or in the most recent log file in `server/logs/`:

```
host:lwm2m_server/docker_compose$ leshan-1 | LeshanServer started  
leshan-1 | new device registered: qemu_x86  
leshan-1 | Onboarding qemu_x86  
leshan-1 | Resources:  
leshan-1 | </3>  
leshan-1 | </3/0>  
leshan-1 | </3/0/0>  
leshan-1 | </3/0/1>  
[..]
```

Additionally you can see the device in the Django server under `http://localhost:8000/admin/sensordata/endpoint/`. You should see that the LAST UPDATED field contains a recent timestamp.

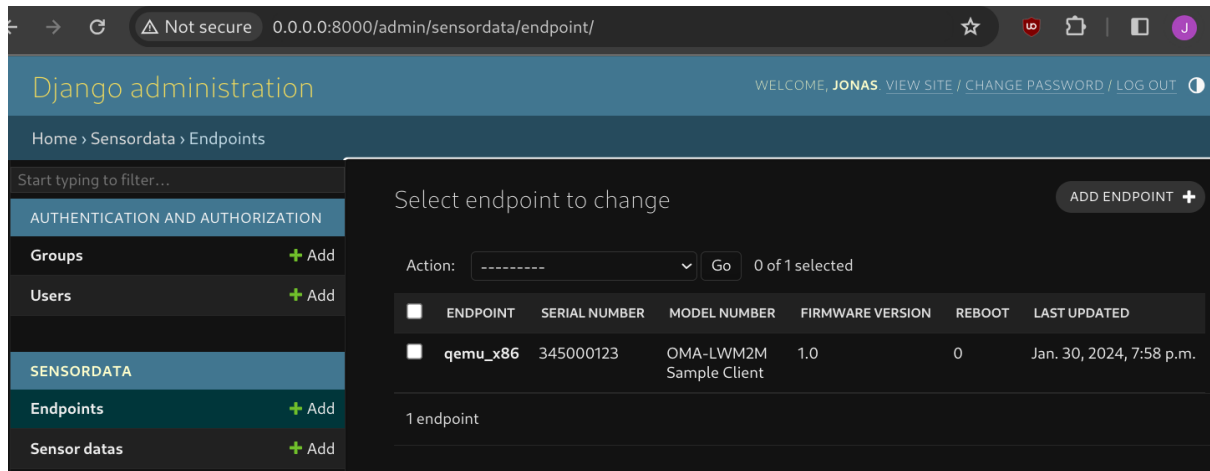


Fig. 1: Endpoints table in Django

DOCUMENTATION

7.1 Build the documentation

```
host:~$ sudo apt-get install default-jre plantuml graphviz
host:~$ source venv/bin/activate
host:~$ cd flownexus_workspace/lwm2m_server/doc
host:lwm2m_server/doc$ pip install -r requirements.txt
host:lwm2m_server/doc$ tox -e py3-html
```

Open the generated index.html in the doc/build directory in your browser.

GLOSSARY

Terms are usually explained and indexed directly in the documentation chapters. Additionally, some general terms are defined here to provide a quick reference. The terms defined here, as well as the indexed terms from the documentation chapters can be referenced in the documentation by using the `:term:` role.

8.1 Additional Terms and Definitions

LwM2M

Lightweight Machine to Machine protocol.

Leshan

LwM2M server implementation as one main component of the system.

Backend

Django instance that is responsible for the REST API, database and visualization.

Frontend

Web application that visualizes the data from the backend, integrated into Django.

INTERNAL DJANGO

The API Documentation is available at https://jonas-rem.github.io/lwm2m_server.

10.1 Update Documentation & Add Weblog

#2

Date

May 15, 2024

Author

<https://github.com/Kappuccino111>

Tags

Add, Update

- Switched the new documentation to *Sphinx-Book-Theme* for better readability and navigation.
 - Added a new section for the weblog to keep track of updates and changes in the project.
-

10.2 Project Starts

#1

Date

January 1, 2024

Author

<https://github.com/jonas-rem>

Tags

Update

- The project has started and the initial documentation has been added.
-

INDEX

B

Backend, 18

E

Endpoint, 7

EndpointOperation, 8

Event, 8

EventResource, 8

F

Firmware, 8

Frontend, 18

L

Leshan, 18

LwM2M, 18

R

Resource, 8

ResourceType, 7