

# Middleware for IoT

HATIBI Mohammed Amine  
Anka Soubaai Abdelmajid

5 ISS

# Contents

## Lab 1 & 2 : ESP8266 & MQTT

1) Aim of the lab.....	3
2) MQTT.....	3
3) Install and the broker.....	5
4) Creation of an IoT device with the nodeMCU board that uses MQTT communication.....	5
5) Creation of a simple application.....	7
6) Conclusion.....	9

## Lab 3 : oneM2M REST API

1) The Application.....	11
2) Initialisation.....	11
3) AE Resources.....	12
4) Container resource for each AE.....	15
5) Content instance.....	17
6) Subscriptions creation.....	19
7) Simulation.....	22
8) Conclusion.....	23

## Lab 4 : Fast prototyping of IoT application with node-RED

1) Aim of the Lab.....	24
2) Node-Red.....	24
3) Applications.....	28
4) Simple test : Display Sensor Values.....	28
a) Sensors and activators.....	29
b) Dashboard.....	30

Conclusion.....	31
-----------------	----

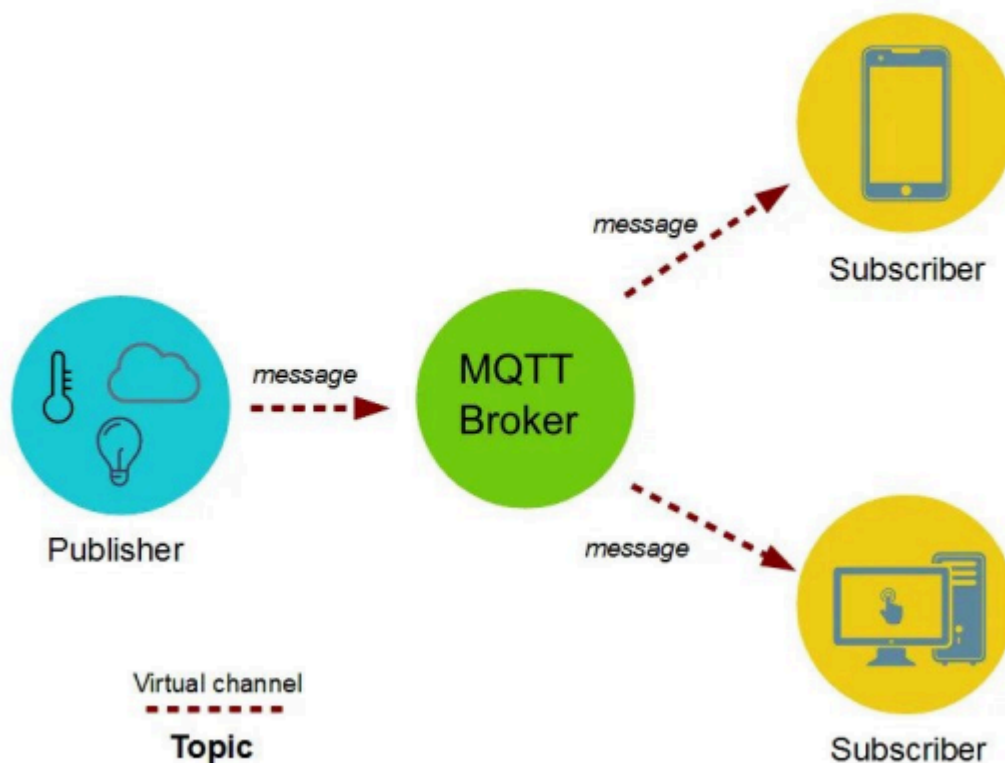
# Lab 1 & 2 : ESP8266 & MQTT

## 1) Aim of the Lab :

The goal of this lab is to explore the capabilities of the MQTT protocol for IoT. To do that, we will first make a little state of the art about the main characteristics of MQTT, then we will install several software on our laptop to manipulate MQTT. Finally, we will develop a simple application with an IoT device (ESP8266) using the MQTT protocol to communicate with a server on our laptop

## 2) MQTT :

- What is the typical architecture of an IoT system based on the MQTT protocol ?



In MQTT architecture, there are two types of systems: clients and brokers. A broker is the server that the clients communicate with. The broker receives communications from clients and sends those communications on to 2 other clients. Clients do not communicate directly with each other, but rather connect to the broker. Each client may be either a publisher, a subscriber, or both.

MQTT is an event-driven protocol. There is no periodic or ongoing data transmission. This keeps transmission to a minimum. A client only publishes when there is information to be sent, and a broker only sends out information to subscribers when new data arrives.

- What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc ?

MQTT runs on top of TCP/IP using a PUSH/SUBSCRIBE topology. This lightweight protocol allows it to be implemented on limited latency networks. In terms of communication, this means that it can support diverse application scenarios for IoT devices and services. And in terms of bandwidth usage, it is designed to work in a network with a limited bandwidth and low power.

- What are the different versions of MQTT?

There are two different variants of MQTT and several versions.

- MQTT v3.1.0
- MQTT v3.1.1
- MQTT v5
- MQTT-SN

There is very little difference between v3.1.0 and 3.1.1, and the MQTT v5 version is the latest one.

- What kind of security/authentication/encryption are used in MQTT?

The original goal of the MQTT protocol was to make the smallest and most efficient data transmission possible over expensive, unreliable communication lines.

However, there are some security options available at the cost of more data transmission and a larger footprint.

- SSL/TLS – Running on top of TCP/IP, the obvious solution for securing transmissions between clients and brokers is the implementation of SSL/TLS. Unfortunately, this adds substantial overhead to the otherwise lightweight communications.
- Network security – If the network itself can be secured, then the transmission of insecure MQTT data is irrelevant. In this case, security issues would have to occur from inside the network itself, perhaps via a bad actor or another form of network penetration.
- Username and password – MQTT does allow usernames and passwords for a client to establish a connection with a broker

- Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for your house with this behavior:
  - you would like to be able to switch on the light manually with the button
  - the light is automatically switched on when the luminosity is under a certain value
 What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?

The two topics needed to get this behavior are : Button State and Luminosity. The button and the light sensor are going to be the publishers, the light is going to be a subscriber and will subscribe to the two topics.

### 3) Install and test the broker :

We first installed mosquitto, and ran the broker with the file.exe. Then, we test the command mosquitto\_pub and mosquitto\_sub

- Command mosquitto\_pub :

*mosquitto\_pub -m "test message" -t button/jaune*

We use the command to send a message in a topic "jaune" contained in the button.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19041.508]
(c) 2020 Microsoft Corporation. Tous droits réservés.

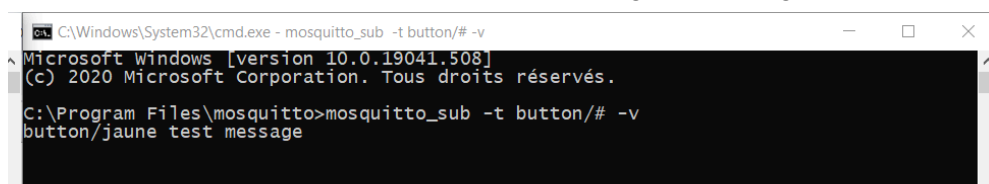
C:\Program Files\mosquitto> mosquitto_pub -m "test message" -t button/jaune

C:\Program Files\mosquitto>
```

- Command mosquitto\_sub :

*mosquitto\_sub -t button/# -v*

We use the command to subscribe and listen if a message is arriving.



```
C:\Windows\System32\cmd.exe - mosquitto_sub -t button/# -v
Microsoft Windows [version 10.0.19041.508]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Program Files\mosquitto> mosquitto_sub -t button/# -v
button/jaune test message
```

### 4) Creation of an IoT device with the nodeMCU board that uses MQTT communication :

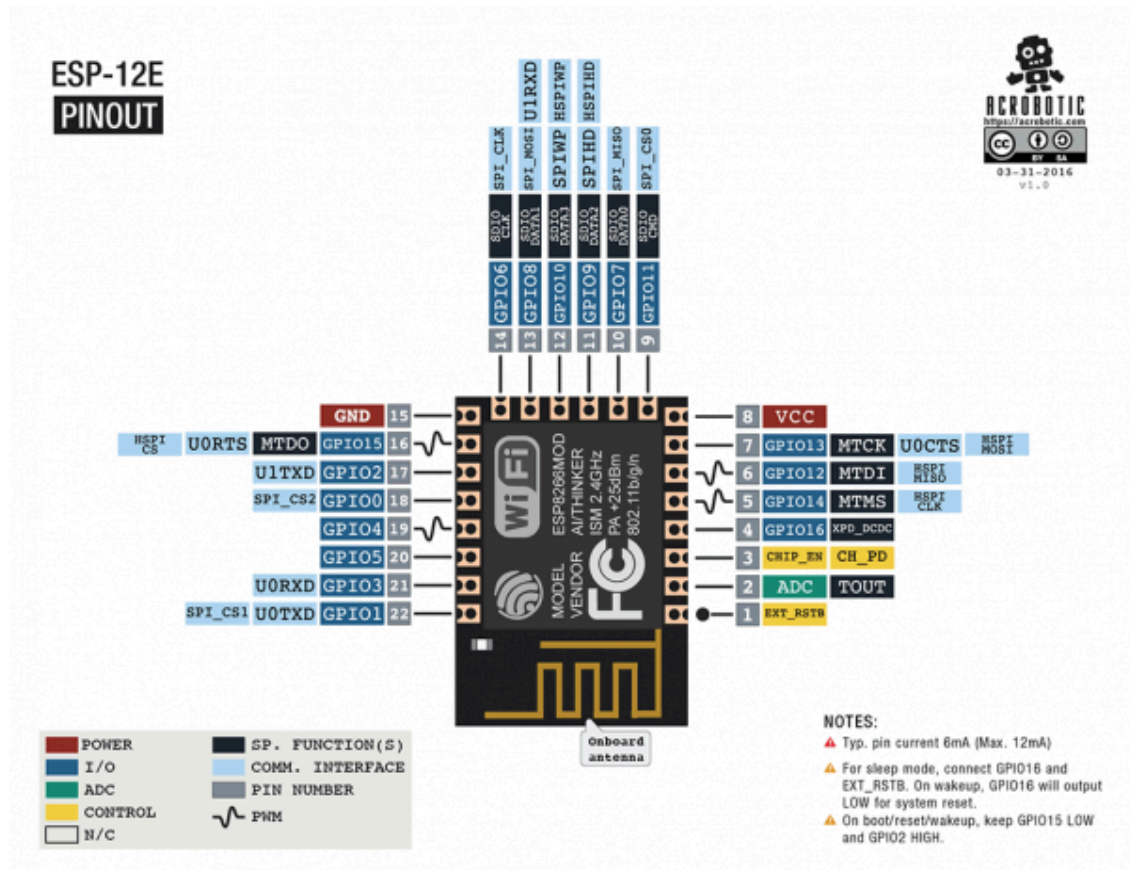
We installed the Arduino IDE and opened an example that we can find on the library arduinoMqtt, which is named connectESP8266wifiClient.

- Give the main characteristics of nodeMCU board in terms of communication, programming language, Inputs/outputs capabilities.

The ESP8266 is a microcontroller 32 bits, which integrates the IEEE 802.11 b/g/n Wi-Fi standard.

- In terms of communication: ESP8266 is a WIFI microchip and the nodeMCU board has 4 pins available for SPI communication.
- In terms of programming language: Different programming languages can be used : **Commandes AT, ESP8266 SDK, Lua (NodeMCU), C/C++ (Arduino), MicroPython, Javascript.** In this lab, we use Arduino.
- The nodeMCU board needs between 3.0V and 3.6V to operate
- In terms of inputs/outputs capabilities: NodeMCU has 16 input/output pins on its board (GPIO)

Here is an image retracing the inputs/outputs capabilities :



We send a message from the wireless router to the electronic card, then the message is sent to the broker that we have installed on the computer.

The mosquitto broker manages a set of topics. A topic can be described as a queue of messages.

Basically the broker centralizes the information sent by the publishers so that everything is available for every subscriber. The publisher is a sensor and the subscriber a client, like a server on the cloud.

## 5) Creation of a simple application :

Blocs that we have modified from the example :

### **Bloc Configuration**

```
// Setup WiFi network
WiFi.mode(WIFI_STA);
WiFi.hostname("ESP_" MQTT_ID);
WiFi.begin("Cisco38658", "");
LOG_PRINTFLN("\n");
LOG_PRINTFLN("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    LOG_PRINTFLN(".");
}
LOG_PRINTFLN("Connected to WiFi");
LOG_PRINTFLN("IP: %s", WiFi.localIP().toString().c_str());
```

### **Bloc Sub**

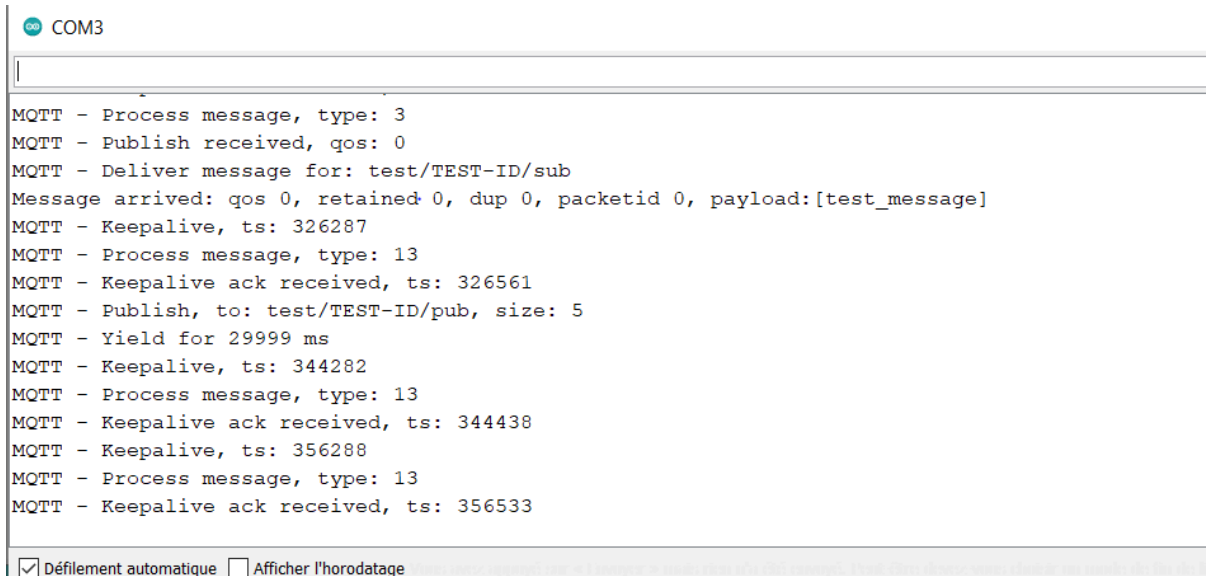
```
{
    MqttClient::Error::type rc = mqtt->subscribe(
        MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
    );
    if (rc != MqttClient::Error::SUCCESS) {
        LOG_PRINTFLN("Subscribe error: %i", rc);
        LOG_PRINTFLN("Drop connection");
        mqtt->disconnect();
        return;
    }
}
```

### **Bloc Pub**

```
{
    const char* buf = "Hello";
    MqttClient::Message message;
    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;
    message.payload = (void*) buf;
    message.payloadLen = strlen(buf);
    mqtt->publish(MQTT_TOPIC_PUB, message);
}
```

Results that we can observe in the broker Mosquitto :

### Mosquitto\_sub

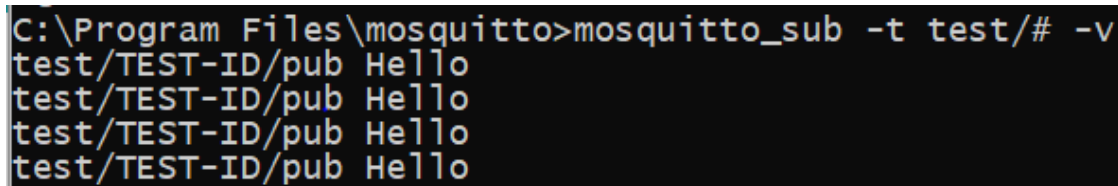


```
COM3

MQTT - Process message, type: 3
MQTT - Publish received, qos: 0
MQTT - Deliver message for: test/TEST-ID/sub
Message arrived: qos 0, retained 0, dup 0, packetid 0, payload:[test_message]
MQTT - Keepalive, ts: 326287
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 326561
MQTT - Publish, to: test/TEST-ID/pub, size: 5
MQTT - Yield for 29999 ms
MQTT - Keepalive, ts: 344282
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 344438
MQTT - Keepalive, ts: 356288
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 356533

☒ Défilement automatique ☐ Afficher l'horodatage
```

### Mosquitto\_pub



```
C:\Program Files\mosquitto>mosquitto_sub -t test/# -v
test/TEST-ID/pub Hello
test/TEST-ID/pub Hello
test/TEST-ID/pub Hello
test/TEST-ID/pub Hello
```

In the part where we would like to be able to switch on the light manually with the button, we've decided that the light will be on when the button is pressed, and on the contrary switched off when the button is not pressed. We created a function for this. In a temporary variable, we stock the value ridden of the state of the button and then treat each case separately. We publish a different message depending on the state of the button/LED.

It has been well observed that when we press the button, the light is switched on, and that when the button is not pressed, the light is not switched on.

In the part where the light is automatically switched on when the luminosity is under a certain value and on the contrary switched off when the luminosity is over the limit, we chose 100 Lux as the limit. In the same way as the first part, we have created a function that treats each situation. We stock the ridden value from the light sensor and then act accordingly. It has been well observed that when we cover the luminosity sensor, the light is switched on, and that when the captor detects enough light, the led is switched off.



## 6) Conclusion :

This lab allowed us to fully understand the MQTT protocol and also how it is important for IoT.

We started by understanding the main characteristics of MQTT, then we installed several software on our laptop to be able to manipulate MQTT. Finally, we developed a simple application with an IoT device using the MQTT protocol.

## Lab 3 : oneM2M REST API :

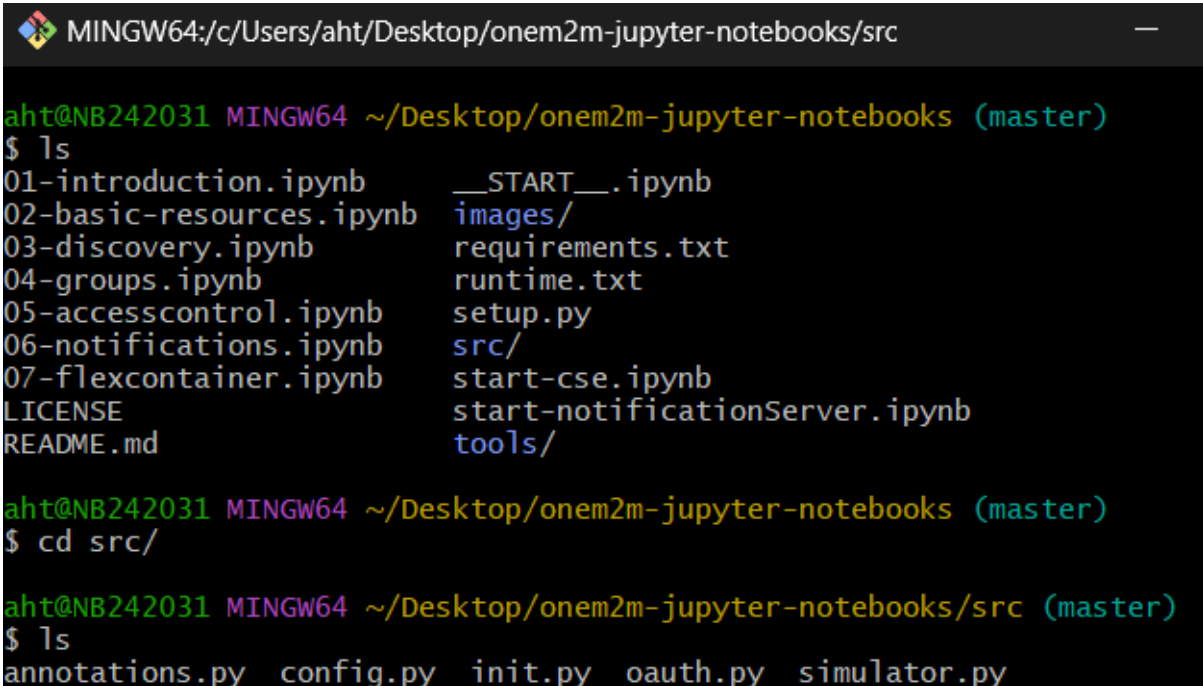
The objective is to develop an application that simulates the behavior of a device created during labs 1 and 2 on the ESP8266. The focus of these labs is to manipulate the service layer of oneM2M and various resources.

We will utilize an oneM2M stack named ACME and interact with the ACME CSE using a Jupyter Notebook. The assumption is that an IN-CSE has been installed, configured, and run on their laptop from GitHub. Additionally, the Jupyter Notebook has been installed, and all six chapters have been completed.

Given the flexibility in their approach, the program will be built using the Notebook library to facilitate the creation of REST requests seamlessly.

We used a separate Python virtual environment from the global environment because we encountered an issue with opening the notebook initially. To do this, we use the command **'python -m venv venv'**.

First , we start by navigating to the directory dedicated to the notebook .



```
MINGW64:/c/Users/aht/Desktop/onem2m-jupyter-notebooks/src
aht@NB242031 MINGW64 ~/Desktop/onem2m-jupyter-notebooks (master)
$ ls
01-introduction.ipynb      __START__.ipynb
02-basic-resources.ipynb  images/
03-discovery.ipynb        requirements.txt
04-groups.ipynb           runtime.txt
05-accesscontrol.ipynb    setup.py
06-notifications.ipynb    src/
07-flexcontainer.ipynb    start-cse.ipynb
LICENSE                   start-notificationServer.ipynb
README.md                 tools/

aht@NB242031 MINGW64 ~/Desktop/onem2m-jupyter-notebooks (master)
$ cd src/

aht@NB242031 MINGW64 ~/Desktop/onem2m-jupyter-notebooks/src (master)
$ ls
annotations.py  config.py  init.py  oauth.py  simulator.py
```

## The Application:

We possess devices equipped with a button, a light, and a luminosity sensor. Our goal is to develop a smart home system with the following functions:

- Manual activation of the light using the button.
- Automatic activation of the light when the luminosity falls below a specific threshold.

To achieve this, we aim to create an application that simulates this behavior through a RESTful IoT API provided by a oneM2M Common Services Entity (CSE).

## Initialisation

Firstly, we need to initiate a CSE within the notebook.

### CSE - Runtime

#### Starting the CSE ¶

Executing the following command will start a CSE inside the notebook.

```
# Increase the width of the notebook to accomodate the log output
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# Change to the CSE's directory and start the CSE
# Ignore the error from the %cd command
%cd -q tools/ACME
%run -m acme -- --headless
```

```
C:\Users\ah\AppData\Local\Programs\Python\Python311\Lib\site-packages\IPython\core\magics\osm.py:417: UserWarning: using dhist requires you to install the 'pickleshare' library.
  self.shell.db['dhist'] = compress_dhist(dhist)[-100:]
[ACME] 0.10.2 - An open source CSE Middleware for Education
```

CSE started

Next, we proceed with importing essential modules and configurations, setting up the CSE.



THE IoT STANDARD

### Initialization

The section does import necessary modules and configurations, and prepares the CSE for this notebook.

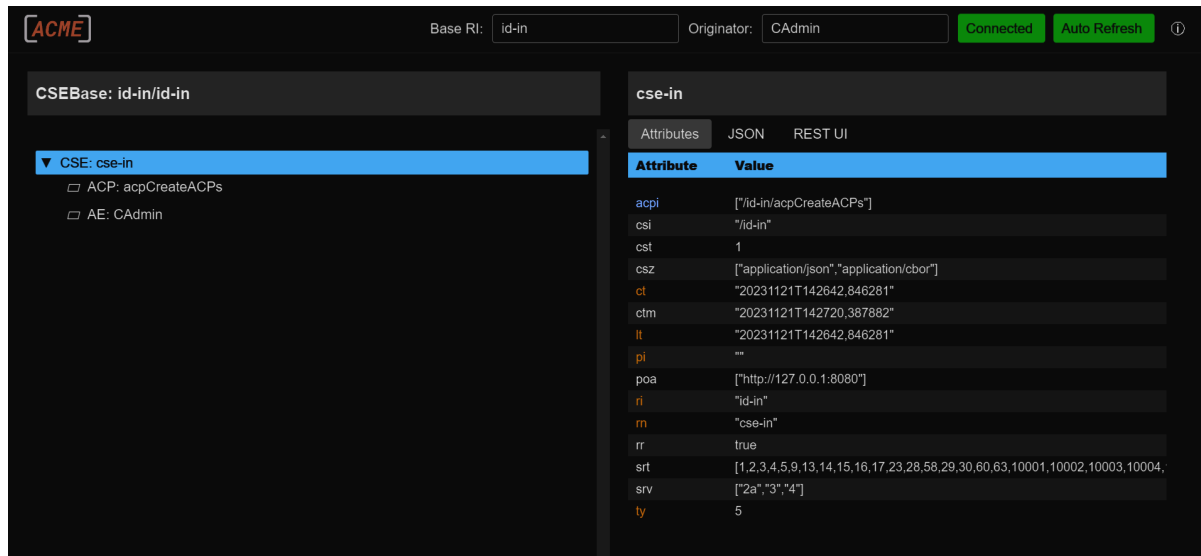
```
[1]: %run src/init.py introduction
```

Configuration Ready (cse-in)

#### Initial Resource Tree

```
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
```

Now, within the webUI, our CSE is visible, albeit currently empty.



## AE Resources

In this section, our objective is to establish our AE resources. Specifically, we require the creation of three distinct AEs:

- A button
- A light
- A luminosity sensor

These entities are designated within the Notebook as follows:

```
[2]: CREATE (
# CREATE request

# Create the AE resource under the CSEBase
to = cseBaseName,

# Request Parameters
originator = 'Cmyself',
requestIdentifier = '123',
releaseVersionIndicator = '3',
resourceType = Type.AE,

# Request Body
primitiveContent =
{
  'm2m:ApplicationEntity': {
    'resourceName': 'Button',
    'App-ID': 'Nbutton',
    'requestReachability': True,
    'supportedReleaseVersions': [ '3' ]
  }
}
)
```

# Assign an originator ID, must start with 'C'  
# Unique request identifier  
# Release version indicator  
# Type of the resource: AE  
# AE can receive requests  
# Supports oneM2M release 3

```
[3]: CREATE (                                     # CREATE request

# Create the AE resource under the CSEBase
to           = cseBaseName,

# Request Parameters
originator   = 'Clight',                         # Assign an originator ID, must start with 'C'
requestIdentifier = '123',                       # Unique request identifier
releaseVersionIndicator = '3',                   # Release version indicator
resourceType = Type.AE,                         # Type of the resource: AE

# Request Body
primitiveContent =
{
    'm2m:ApplicationEntity': {
        'resourceName':      'Light',
        'App-ID':            'Nlight',
        'requestReachability': True,             # AE can receive requests
        'supportedReleaseVersions': [ '3' ]       # Supports oneM2M release 3
    }
}

)
```

```
[4]: CREATE (                                     # CREATE request

# Create the AE resource under the CSEBase
to           = cseBaseName,

# Request Parameters
originator   = 'CSensor',                       # Assign an originator ID, must start with 'C'
requestIdentifier = '123',                       # Unique request identifier
releaseVersionIndicator = '3',                   # Release version indicator
resourceType = Type.AE,                         # Type of the resource: AE

# Request Body
primitiveContent =
{
    'm2m:ApplicationEntity': {
        'resourceName':      'Sensor',           # Name of the resource
        'App-ID':            'Nsensor',          # Application ID, must start with 'N'
        'requestReachability': True,             # AE can receive requests
        'supportedReleaseVersions': [ '3' ]       # Supports oneM2M release 3
    }
}

)
```

As of now, CSE Resource Tree contains the three AE :

#### ► oneM2M Response

##### CSE Resource Tree

```
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├─ Button -> m2m:ApplicationEntity | resourceID=Cmyself
├─ Light -> m2m:ApplicationEntity | resourceID=Clight
└─ Sensor -> m2m:ApplicationEntity | resourceID=CSensor
```

The server returns "Status" value "201 (CREATED)" this means that the AE is created .

## HTTP Response

201 (CREATED)

Headers			Response Content   Body	
HTTP Header	Request Attribute	Value		
X-M2M-RSC	Response Status Code	2001	{ "m2m:ApplicationEntity": { "resourceName": "Sensor", "App-ID": "Nsensor", "requestReachability": true, "supportedReleaseVersions": [ "3" ], "resourceID": "CSensor", "parentID": "id-in", "creationTime": "20231121T143322,049972", "lastModifiedTime": "20231121T143322,049972", "expirationTime": "20241120T143322,049972", "resourceType": 2, "AE-ID": "CSensor" } }	
X-M2M-RI	Request Identifier	123		
X-M2M-RVI	Release Version Indicator	3		
X-M2M-OT	Originating Timestamp	20231121T143322,052973		
Content-Type		application/json		

We can See in the server that AE:Button, AE:Light, AE:Sensor we created :

ACME

Base RI: id-in

Originator: CAdmin

Connected

Auto Refresh

CSEBase: id-in/id-in

▼ CSE: cse-in

ACP: acpCreateACPs

AE: Button

AE: CAdmin

AE: Light

AE: Sensor

cse-in

Attributes

JSON

REST UI

Attribute	Value
acpi	["/id-in/acpCreateACPs"]
csi	"/id-in"
cst	1
csz	["application/json","application/cbor"]
ct	"20231121T143307,411211"
ctm	"20231121T144135,138549"
lt	"20231121T143307,411211"
pi	""
poa	["http://127.0.0.1:8080"]
ri	"id-in"
rn	"cse-in"
rr	true
srt	[1,2,3,4,5,9,13,14,15,16,17,23,28,58,29,30,60,63,10001,10002,10003,10004]
srv	["2a","3","4"]
ty	5

## Container resource for each AE

In this stage, we'll incorporate a container resource for each AE. A container, within the oneM2M resource hierarchy, serves as a framework capable of accommodating multiple data instances as subordinate resources. These containers will store the data pertinent to each AE:

- For the button and the light, it will retain their respective statuses (ON and OFF).
- In the case of the luminosity sensor, it will capture and store the luminosity data.

```
[5]: CREATE (                                     # CREATE request

    # Create the container resource under the AE
    to                                     = cseBaseName + '/Button',

    # Request Parameters
    originator                             = 'Cmyself',      # Set the originator
    requestIdentifier                       = '123',          # Unique request identifier
    releaseVersionIndicator                 = '3',            # Release version indicator
    resourceType                           = Type.Container, # Type of the resource: container

    # Request Body
    primitiveContent =
    {
        'm2m:Container': {
            'resourceName': 'Button_Status' # Set the resource name
        }
    },
)
```

```
[6]: CREATE (                                     # CREATE request

    # Create the container resource under the AE
    to                                     = cseBaseName + '/Light',

    # Request Parameters
    originator                             = 'Clight',       # Set the originator
    requestIdentifier                       = '123',          # Unique request identifier
    releaseVersionIndicator                 = '3',            # Release version indicator
    resourceType                           = Type.Container, # Type of the resource: container

    # Request Body
    primitiveContent =
    {
        'm2m:Container': {
            'resourceName': 'Light_Status' # Set the resource name
        }
    },
)
```

```
[7]: CREATE (                                     # CREATE request

# Create the container resource under the AE
to                                     = cseBaseName + '/Sensor',

# Request Parameters
originator                           = 'CSensor',      # Set the originator
requestIdentifier                     = '123',          # Unique request identifier
releaseVersionIndicator               = '3',            # Release version indicator
resourceType                         = Type.Container,  # Type of the resource: container

# Request Body
primitiveContent =
{
    'm2m:Container': {
        'resourceName': 'Sensor_Status' # Set the resource name
    }
},
)
```

As we can see the server return 201 , it's mean that the AE was created :

201 (CREATED)

Headers			Response Content   Body
HTTP Header	Request Attribute	Value	
X-M2M-RSC	Response Status Code	2001	<pre>{   "m2m:Container": {     "resourceName": "Sensor_Status",     "resourceID": "cnt5850733832422414360",     "parentID": "CSensor",     "creationTime": "20231121T145817,345613",     "lastModifiedTime": "20231121T145817,345613",     "expirationTime": "20241120T145817,345613",     "resourceType": 3,     "currentNrOfInstances": 0,     "currentByteSize": 0,     "stateTag": 0   } }</pre>
X-M2M-RI	Request Identifier	123	
X-M2M-RVI	Release Version Indicator	3	
X-M2M-OT	Originating Timestamp	20231121T145817,351715	
Content-Type		application/json	

The CSE Resource Tree contains three AE and for each AE a container :

```
CSE Resource Tree
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├── Button -> m2m:ApplicationEntity | resourceID=Cmyself
│   └── Button_Status -> m2m:Container | resourceID=cnt8954325963567000354
├── Light -> m2m:ApplicationEntity | resourceID=Clight
│   └── Light_Status -> m2m:Container | resourceID=cnt6937413823686619564
└── Sensor -> m2m:ApplicationEntity | resourceID=CSensor
    └── Sensor_Status -> m2m:Container | resourceID=cnt5850733832422414360
```

**Note:** Each AE could be accompanied by a container descriptor to detail every sensor, for instance. However, in this scenario, we've solely generated containers for data.

**Access Control Policy (ACP):** These policies link access control with credentials, outlining rules for resource access. In our scenario, we opted for simplicity, hence the decision not to establish any ACP. However, in a real-world context, we could enforce restrictions on users, their permitted operations, and even aspects like date, time, geo-location, network addresses, and beyond.



## Content instance

To set up the containers for each AE, we'll input specific data. The button will start as "OFF" since it's initially inactive. The light will also be set to "OFF" as it's currently unlit, while the luminosity sensor will begin at 100Lux.

```
[8]: CREATE (                                     # CREATE request

    # Create the content instance resource under the container
    to           = cseBaseName + '/Button/Button_Status',

    # Request Parameters
    originator    = 'Cmyself',                    # Set the originator
    requestIdentifier = '123',                     # Unique request identifier
    releaseVersionIndicator = '3',                 # Release version indicator
    resourceType  = Type.ContentInstance, # Type of the resource: contentInstance

    # Request Body
    primitiveContent =
        {
            'm2m:ContentInstance': {
                'contentInfo': 'text/plain:0',    # Media type of the content
                'content': 'OFF'                  # The content itself
            }
        }
)

[9]: CREATE (                                     # CREATE request

    # Create the content instance resource under the container
    to           = cseBaseName + '/Light/Light_Status',

    # Request Parameters
    originator    = 'Clight',                    # Set the originator
    requestIdentifier = '123',                     # Unique request identifier
    releaseVersionIndicator = '3',                 # Release version indicator
    resourceType  = Type.ContentInstance, # Type of the resource: contentInstance

    # Request Body
    primitiveContent =
        {
            'm2m:ContentInstance': {
                'contentInfo': 'text/plain:0',    # Media type of the content
                'content': 'OFF'                  # The content itself
            }
        }
)
```

```
[10]: CREATE (                                     # CREATE request

    # Create the content instance resource under the container
    to                               = cseBaseName + '/Sensor/Sensor_Status',

    # Request Parameters
    originator                       = 'CSensor',           # Set the originator
    requestIdentifier                 = '123',               # Unique request identifier
    releaseVersionIndicator           = '3',                 # Release version indicator
    resourceType                      = Type.ContentInstance, # Type of the resource: contentInstance

    # Request Body
    primitiveContent =
        {
            'm2m:ContentInstance': {
                'contentInfo': 'text/plain:0',    # Media type of the content
                'content': 'OFF'                  # The content itself
            }
        }
)
```

Each container within every AE has been initialized with data:

► oneM2M Response

CSE Resource Tree

```
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├── Button -> m2m:ApplicationEntity | resourceID=Cmyself
│   └── Button_Status -> m2m:Container | resourceID=cnt3084946363728832120
│       └── cin_WYIRcty17C -> m2m:ContentInstance (text/plain:0) | resourceID=cin4910411114126238706
├── Light -> m2m:ApplicationEntity | resourceID=Clight
│   └── Light_Status -> m2m:Container | resourceID=cnt4237441171208088387
│       └── cin_EK39ISIE4b -> m2m:ContentInstance (text/plain:0) | resourceID=cin3811897741664638424
└── Sensor -> m2m:ApplicationEntity | resourceID=CSensor
    └── Sensor_Status -> m2m:Container | resourceID=cnt6796548186437469066
        └── cin_1gBB9bLZhr -> m2m:ContentInstance (text/plain:0) | resourceID=cin2851287498569978930
```



Base RI: id-in

Originator: CAdmin

Connected

Auto Refresh



ContentInstance: id-in/cin3811897741664638424

- ▼ CSE: cse-in
  - ACP: acpCreateACPs
  - AE: Button
  - AE: CAdmin
  - ▼ AE: Light
    - ▼ CNT: Light\_Status
      - CIN: cin\_EK39ISIE4b
  - AE: Sensor

cse-in/Light/Light\_Status/cin\_EK39ISIE4b

Attributes JSON REST UI

Attribute	Value
cnf	"text/plain:0"
con	"OFF"
cs	3
ct	"20231121T15:19:50.439739"
et	"20241120T15:19:50.439739"
lt	"20231121T15:19:50.439739"
pi	"cnt4237441171208088387"
ri	"cin3811897741664638424"
rn	"cin_EK39ISIE4b"
st	1
ty	4

**Group Creation:** Hypothetically, if multiple lights were present, we could create a group enabling simultaneous activation of all lights by pressing a button or retrieving the most recent states of all lights. However, since we're dealing with only one light in our application, this functionality isn't required.

## Subscriptions creation

Subscription resources serve to track resource creations, updates, and additional events. They also encompass the setup for notifications, specifying when and how these notifications are dispatched to a designated target.

When a user generates a subscription resource, they configure the parameter "notification content type" (abbreviated as nct) to a value of 1. This value signifies that all attributes of the subscribed resource will be relayed as notifications to the subscriber.

For instance, if the Button AE establishes a subscription resource, it includes a notification URI set to the resource identifier of the Button AE. This setup ensures that the Button AE receives notifications whenever there are alterations in the content instance of the Button\_Status container.

Initially, the notification server needs activation. To engage with notifications, running a notification server becomes imperative, as it handles two distinct types of requests:

- Upon the creation of a new <Subscription>, a verification request is dispatched to the notification server. This request aims to validate the subscription's authenticity.
- Whenever a monitored resource undergoes changes, the server is alerted. These alterations are then transmitted or "pushed" to the notification server for processing.

```
[*]: # Increase the width of the notebook to accomodate the log output
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# change to the CSE's directory and start the CSE
%cd -q tools/NotificationServer
%run ./NotificationServer.py

C:\Users\ankas\ACME-oneM2M-CSE\env\Lib\site-packages\IPython\core\magics\osm.py:417: UserWarning: using dhists requires you to install the `pickleshare` library.
  self.shell.db['dhists'] = compress_dhists(dhists)[-100:]
[ACME] - Notification Server

Notification server started.
Listening for http(s) connections on port 9999.
```

Now we can start the subscription to the AE containers

```
CREATE (                                     # CREATE request

# Create subscription resource under the container
to                                     = cseBaseName + '/Sensor/Sensor_Status',

# Request Parameters
originator                           = 'CSensor', # Set the originator
requestIdentifier                     = '123',      # Unique request identifier
releaseVersionIndicator              = '3',        # Release version indicator
resourceType                         = Type.Subscription, # Type of the resource: Subscription

# Request Body
primitiveContent =
{
    'm2m:Subscription': {
        'resourceName' : 'Subscription',
        'notificationURI' : [ notificationURL ],
        'notificationContentType' : 1,
        'eventNotificationCriteria' : {
            'notificationEventType': [ 1, 2, 3, 4 ]
        }
    }
}
)
```

```
: CREATE (                                     # CREATE request

# Create subscription resource under the container
to                                     = cseBaseName + '/Button/Button_Status',

# Request Parameters
originator                           = 'Cmyself', # Set the originator
requestIdentifier                     = '123',      # Unique request identifier
releaseVersionIndicator              = '3',        # Release version indicator
resourceType                         = Type.Subscription, # Type of the resource: Subscription

# Request Body
primitiveContent =
{
    'm2m:Subscription': {
        'resourceName' : 'Subscription',
        'notificationURI' : [ notificationURL ],
        'notificationContentType' : 1,
        'eventNotificationCriteria' : {
            'notificationEventType': [ 1, 2, 3, 4 ]
        }
    }
}
)
```

```
[13]: CREATE (                                     # CREATE request

# Create subscription resource under the container
to                                     = cseBaseName + '/Light/Light_Status',

# Request Parameters
originator                           = 'CLight', # Set the originator
requestIdentifier                     = '123',      # Unique request identifier
releaseVersionIndicator              = '3',        # Release version indicator
resourceType                         = Type.Subscription, # Type of the resource: Subscription

# Request Body
primitiveContent =
{
    'm2m:Subscription': {
        'resourceName' : 'Subscription',
        'notificationURI' : [ notificationURL ],
        'notificationContentType' : 1,
        'eventNotificationCriteria' : {
            'notificationEventType': [ 1, 2, 3, 4 ]
        }
    }
}
)
```

As we can see the server return 201 , it's mean that the AE was created :


## HTTP Response

201 (CREATED)

Headers			Response Content   Body	
HTTP Header	Request Attribute	Value		
X-M2M-RSC	Response Status Code	2001		
X-M2M-RI	Request Identifier	123		
X-M2M-RVI	Release Version Indicator	3		
X-M2M-OT	Originating Timestamp	20231121T211505,286308		
Content-Type		application/json		
			<pre>{   "m2m:Subscription": {     "resourceName": "Subscription",     "notificationURI": [       "http://localhost:9999"     ],     "notificationContentType": 1,     "eventNotificationCriteria": {       "notificationEventType": [         1,         2,         3,         4       ]     },     "resourceID": "sub2177458812731170548",     "parentID": "cnt3879308459773967174",     "creationTime": "20231121T211503,200435",     "lastModifiedTime": "20231121T211503,200435",     "expirationTime": "20241120T211503,200435",     "resourceType": 23   } }</pre>	

From both the CSE Resource Tree and the server interface, we have the capability to oversee and track all the containers.

### ► oneM2M Response

 CSE Resource Tree

```
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├── Button -> m2m:ApplicationEntity | resourceID=Cmyself
│   └── Button_Status -> m2m:Container | resourceID=cnt3505830373884372999
│       ├── cin_96vqzxRjXW -> m2m:ContentInstance (text/plain:0) | resourceID=cin8997091762271056629
│       └── Subscription -> m2m:Subscription | resourceID=sub5456090952994540413
├── Light -> m2m:ApplicationEntity | resourceID=Clight
│   └── Light_Status -> m2m:Container | resourceID=cnt3879308459773967174
│       ├── cin_j6dVNB0C30 -> m2m:ContentInstance (text/plain:0) | resourceID=cin1245780101210205866
│       └── Subscription -> m2m:Subscription | resourceID=sub2177458812731170548
└── Sensor -> m2m:ApplicationEntity | resourceID=CSensor
    └── Sensor_Status -> m2m:Container | resourceID=cnt1748421551701084926
        ├── cin_EjUmKafgRW -> m2m:ContentInstance (text/plain:0) | resourceID=cin903553530116994588
        └── Subscription -> m2m:Subscription | resourceID=sub3976823730100991959
```

▼ CSE: cse-in

- ACP: acpCreateACPs
- ▼ AE: Button
  - ▼ CNT: Button\_Status
    - CIN: cin\_U4CHJEaBJE
    - SUB: Subscription
- AE: CAdmin
- ▼ AE: Light
  - ▼ CNT: Light\_Status
    - CIN: cin\_3YKEC5SMS6
    - SUB: Subscription
- ▼ AE: Sensor
  - ▼ CNT: Sensor\_Status
    - CIN: cin\_FN8aYa3arQ
    - SUB: Subscription

Attributes JSON REST UI

Attribute	Value
ct	"20231121T212252,555682"
enc	{"net":[1,2,3,4]}
et	"20241120T212252,555682"
lt	"20231121T212252,555682"
nct	1
nu	["http://localhost:9999"]
pl	"cnt0984982039152811597"
ri	"sub4860726553518718065"
rm	"Subscription"
ty	23

For instance, within the notification server, we're able to observe the subscription of an AE to its respective container. Here's an illustration: the Button AE sends a subscription request for the Button\_Status. The server responds with a "200" status code, indicating successful creation of the subscription.

```
Notification server started.  
Listening for http(s) connections on port 9999.  
### Received Notification (http)  
Host: localhost:9999  
User-Agent: ACME 0.10.2  
Accept-Encoding: gzip, deflate  
Accept: */*  
Connection: keep-alive  
Date: Tue, 21 Nov 2023 21:14:50 GMT  
Content-Type: application/json  
cache-control: no-cache  
X-M2M-Origin: /id-in  
X-M2M-RI: 7208286784573669355  
X-M2M-RVI: 4  
X-M2M-OT: 20231121T211450,616106  
Content-Length: 83
```

```
"POST / HTTP/1.1" 200
```

```
{  
  "m2m:sgn": {  
    "vrq": true,  
    "sur": "/id-in/sub3976823730100991959",  
    "cr": "CSensor"  
  }  
}
```

```
### Sent Notification Response (http)  
HTTP/1.0 200 OK  
Server: BaseHTTP/0.6 Python/3.12.0  
Date: Tue, 21 Nov 2023 21:14:52 GMT  
X-M2M-RSC: 2000  
X-M2M-RI: 7208286784573669355
```

## Simulation

Within the IN-CSE, our setup involves three AEs: a Button, a Light, and a Luminosity sensor. To replicate the ESP8266 use case, we could develop a Python program. This program would continuously monitor the button's status. Upon detecting a new instance, it retrieves its content via the RETRIEVE function. Subsequently, the IN-AE initiates a new content instance creation process, updating the light's state accordingly. This process involves creating a new content instance, setting the light status to 'ON.' Conversely, if the button status switches to 'OFF,' a new content instance is generated, this time altering the light status to 'OFF.'

The application has been developed, and the Python code is deposited in a Git repository located at <https://github.com/Abdel211/M2M.git>.

The same analogy applies to the second application involving the Luminosity sensor. In this case, we could design an 'if' function that toggles the light based on reaching a maximum brightness threshold. This function would rely on monitoring the Luminosity sensor's level, enabling us to adjust the Light Status by adding distinct content instances depending on the situation.

## Conclusion

This project allowed us to apply the different concepts of oneM2M that we learned in the MOOC. Furthermore, we were able to test our knowledge by creating our first AEs, containers and Content Instances. We also learned how to retrieve the latest created Content Instances resource and to update or delete the container. We were also able to test the Discovery function and its labels, the creation of groups, the Access Control and the Subscriptions and Notifications procedure. We were also able to create our own IN-CSE system based on the use case we dealt with in the previous projects.

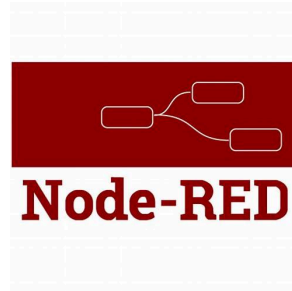
# Lab 4 : Fast prototyping of IoT application with node-RED:

## 1. Aim of the Lab

The goal of this final session is to bring together everything we have accomplished in TP1, TP2, and TP3 to create a fully functional application that interacts with various real and virtual devices. We will be focusing on the following tasks: deploying a high-level application, implementing a concrete architecture with real devices, and using NODE-RED to speed up development. The aim is to create a real-world scenario where devices are connected to multiple technologies, but by using the oneM2M standard and the concept of Interworking Proxy Entities, we can simplify the process by only needing to manage one protocol and eliminating the need for specific interfaces for each device. oneM2M will handle the interoperability between all technologies at the middleware level.

## 2. Node-Red :

To be able to develop a high-level application , we can use a tool called NODE-RED.



Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a visual, drag-anddrop interface for wiring the Internet of Things together without having to write any code. It allows for easy integration of different devices and services, and can be used to create automation flows and dashboards. Node-RED is built on Node.js, and can run on-premises or in the cloud.

In the sequence of commands executed during the practical session, the initial step involved checking the versions of Node.js and npm through the commands ``node -v`` and ``npm -v`` respectively. This preliminary check ensured that the required tool versions were installed and available for subsequent actions.



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Users\ankas\OneDrive\Bureau> node -v
v20.10.0
PS C:\Users\ankas\OneDrive\Bureau> npm -v
10.2.3
PS C:\Users\ankas\OneDrive\Bureau> npm install -g --unsafe-perm node-red
[██████████] | reify:@node-red/editor-api: timing reify:loadBundles

```

Following this, the installation of Node-RED was initiated using the command `npm install -g --unsafe-perm node-red`. This command facilitated the global installation of Node-RED within the system, granting accessibility to Node-RED from any directory or location within the system. The use of the `--unsafe-perm` option might have been employed to circumvent potential permission-related issues during installation, ensuring a seamless installation process without encountering permission restrictions.

```

PS C:\Users\ankas\OneDrive\Bureau> npm install -g --unsafe-perm node-red
added 303 packages in 21s

45 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New patch version of npm available! 10.2.3 -> 10.2.4
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.4
npm notice Run npm install -g npm@10.2.4 to update!
npm notice
PS C:\Users\ankas\OneDrive\Bureau>

```

Following the installation of Node-RED, the next step involved configuring the execution policy within the PowerShell environment. The command `Set-ExecutionPolicy RemoteSigned -Scope Process` was executed during the session. This command is commonly utilized in PowerShell on Windows systems to adjust the execution policy for the current session.

```
PS C:\Program Files\nodejs> Set-ExecutionPolicy RemoteSigned -Scope Process
```

Subsequently, the command `node-red` was executed. This command initiated the Node-RED runtime environment, launching the Node-RED server. Running this command facilitated access to the Node-RED editor interface through a web browser.

```
PS C:\Program Files\nodejs> node-red -v
24 Nov 09:01:21 - [info]

Welcome to Node-RED
=====

24 Nov 09:01:21 - [info] Node-RED version: v3.1.0
24 Nov 09:01:21 - [info] Node.js version: v20.10.0
24 Nov 09:01:21 - [info] Windows_NT 10.0.22621 x64 LE
24 Nov 09:01:23 - [info] Loading palette nodes
24 Nov 09:01:24 - [info] Settings file : C:\Users\ankas\.node-red\settings.
js
24 Nov 09:01:24 - [info] Context store : 'default' [module=memory]
24 Nov 09:01:24 - [info] User directory : C:\Users\ankas\.node-red
24 Nov 09:01:24 - [warn] Projects disabled : editorTheme.projects.enabled=false
24 Nov 09:01:24 - [info] Flows file : C:\Users\ankas\.node-red\flows.json
24 Nov 09:01:24 - [info] Creating new flow file
24 Nov 09:01:24 - [warn]
```

Afterwards, the LOM2M-Node-Red was installed from the GitLab repository available at <https://gitlab.irit.fr/sepia-pub/lightom2m>. The node-red source files dedicated to the oneM2M architecture were obtained from the directory `.../lightom2m/src/node-red`. In this specific instance, the directory path used was `C:\Users\ankas\Downloads\lightom2m-main-src-node-red\lightom2m-main-src-node-red\src\node-red`.

These nodes were instrumental in facilitating interactions not only with LOM2M but also with any Common Services Entity (CSE) using JSON. Detailed documentation regarding the usage and functionalities of these nodes can be found at <https://gitlab.irit.fr/sepia-pub/lightom2m/-/wikis/LOM2M/Node-RED>

This addition helps outline the specific steps you took to access and utilize the node-red source files dedicated to the oneM2M architecture from the provided GitLab repository and highlights their significance in enabling interactions with LOM2M and other CSEs using JSON.

```
PS C:\Users\ankas\.node-red> npm install C:\Users\ankas\Downloads\lightom2m-
main-src-node-red\lightom2m-main-src-node-red\src\node-red

added 1 package, and audited 3 packages in 805ms

found 0 vulnerabilities
PS C:\Users\ankas\.node-red>
PS C:\Users\ankas\.node-red>
```

Following the successful installation of Node-RED and the integration of LOM2M nodes into the environment, the Node-RED server was initiated using the command `node-red`. This command started the Node-RED runtime environment, enabling the utilization of the integrated nodes and providing access to the Node-RED editor interface through a web browser. The activation of Node-RED via this command facilitated the commencement of application development and interaction with LOM2M components within the Node-RED environment.

```
PS C:\Users\ankas\.node-red> node-red
24 Nov 09:24:49 - [info]

Welcome to Node-RED
=====

24 Nov 09:24:49 - [info] Node-RED version: v3.1.0
24 Nov 09:24:49 - [info] Node.js version: v20.10.0
24 Nov 09:24:49 - [info] Windows_NT 10.0.22621 x64 LE
24 Nov 09:24:50 - [info] Loading palette nodes
24 Nov 09:24:51 - [warn] -----
----
24 Nov 09:24:51 - [warn] [node-red-contrib-lom2m/Named ACP] Error: Cannot find module 'request'
Require stack:
- C:\Users\ankas\Downloads\lightom2m-main-src-node-red\lightom2m-main-src-node-red\src\node-red\acp-name.js
- C:\Users\ankas\AppData\Roaming\npm\node_modules\node-red\node_modules\node-red\registry\lib\loader.js
- C:\Users\ankas\AppData\Roaming\npm\node_modules\node-red\node_modules\node-red\registry\lib\index.js
- C:\Users\ankas\AppData\Roaming\npm\node_modules\node-red\node_modules\node-red\runtime\lib\nodes\index.js
- C:\Users\ankas\AppData\Roaming\npm\node_modules\node-red\node_modules\node-red\runtime\lib\index.js
- C:\Users\ankas\AppData\Roaming\npm\node_modules\node-red\lib\red.js

24 Nov 09:24:51 - [warn] Encrypted credentials not found
24 Nov 09:24:51 - [info] Server now running at http://127.0.0.1:1880/
24 Nov 09:24:51 - [info] Starting flows
24 Nov 09:24:51 - [info] Started flows
```

Subsequent to the Node-RED setup, the focus shifted towards initiating a Common Services Entity (CSE) within the notebook environment. This involved the importation of essential modules and configurations necessary for establishing the CSE infrastructure.

### Starting the CSE

Executing the following command will start a CSE inside the notebook.



```
[*]: # Increase the width of the notebook to accommodate the log output
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# Change to the CSE's directory and start the CSE
# Ignore the error from the %cd command
%cd -q tools/ACME
%run -m acme -- --headless

C:\Users\ankas\AppData\Local\Programs\Python\Python312\Lib\site-packages\IPython\core\magics\osm.py:417: UserWarning: using dhist requires you to
install the 'pickleshare' library.
self.shell.db['dhist'] = compress_dhist(dhist)[-100:]
[ACME] 0.10.2 - An open source CSE Middleware for Education

CSE started
```

The initiation process encompassed importing requisite modules and configuring settings to set up the CSE environment. This step was fundamental in preparing the groundwork for subsequent interactions and functionalities within the notebook environment

```
[*]: %run src/init.py basic
```

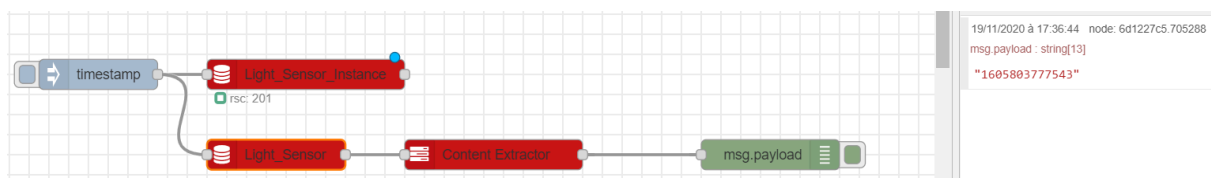
**Configuration Ready (cse-in)**

### 3. Applications

Our objective encompassed the development of a sophisticated application leveraging Node-RED, Eclipse OM2M / LOM2M, and MQTT within a genuine IoT architecture. This involved integrating these technologies to create a high-level application capable of handling real-time interactions and data exchange among various IoT components.

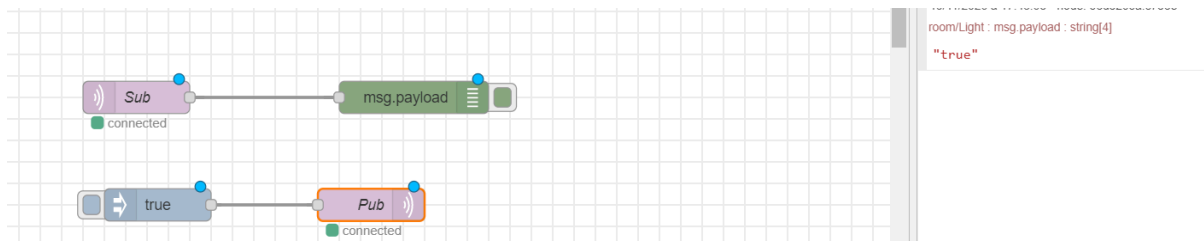
#### 3.1 Simple test : Display Sensor Values :

We initiated our practical exploration by creating a basic application to retrieve and display sensor data. Using Node-RED, we configured nodes to fetch simulated sensor values or connect to real sensors via protocols like MQTT or HTTP. After retrieving the sensor data, we employed processing nodes to extract specific values for display. These values were then directed to a display node, such as a debug node for console visualization or a user interface node for dashboard presentation. This simple application served as an initial step in handling sensor data, showcasing Node-RED's ability to manage and visualize information from various IoT devices.



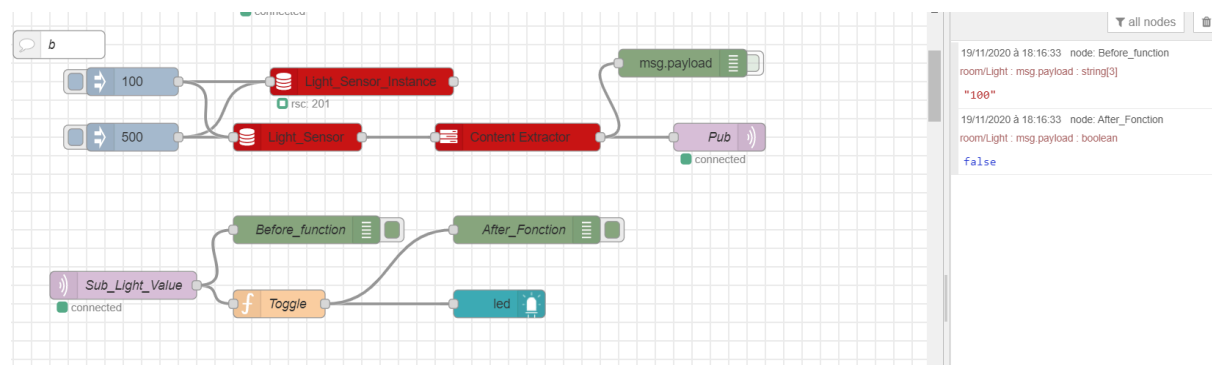
Attribute	Value
rn	cin_817972049
ty	4
ri	/in-cse/cin-817972049
pi	/in-cse/cnt-496344905
ct	20201119T173644
lt	20201119T173644
st	0
cnf	application/json
cs	13
con	1605803804525

As depicted in the images provided, we generated a value through the ContentInstance and subsequently visualized it by employing the Content Extractor associated with the Light Sensor. Validation of the accuracy of the data was performed via the OM2M web page established during Lab 3. Subsequently, leveraging the functionality of the two MQTT nodes, we successfully transmitted a message utilizing the MQTT subscriber.

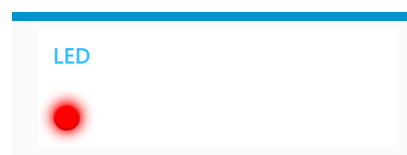


### 3.2 Sensors and activators :

In the subsequent section, we engineered a system centered around the modulation of light input, utilizing a subscription model to establish a connection between the data. This linkage facilitated the activation of either a Green or Red LED contingent upon the varying light values received.



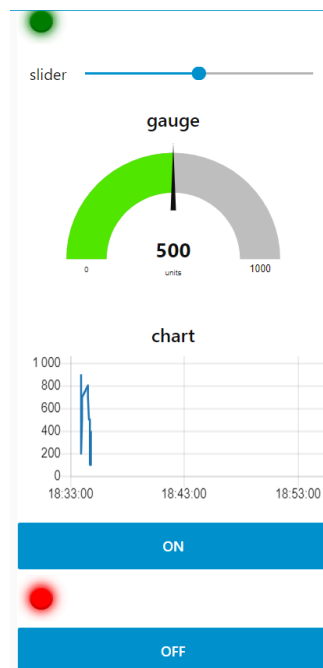
As an illustration, when the value surpasses 250 lux, our configuration dictates the LED to illuminate in green; conversely, for values below this threshold, the LED displays in red. To demonstrate, when assigning a light value of 100 lux, the anticipated outcome is:



### 3.3 Dashboard :

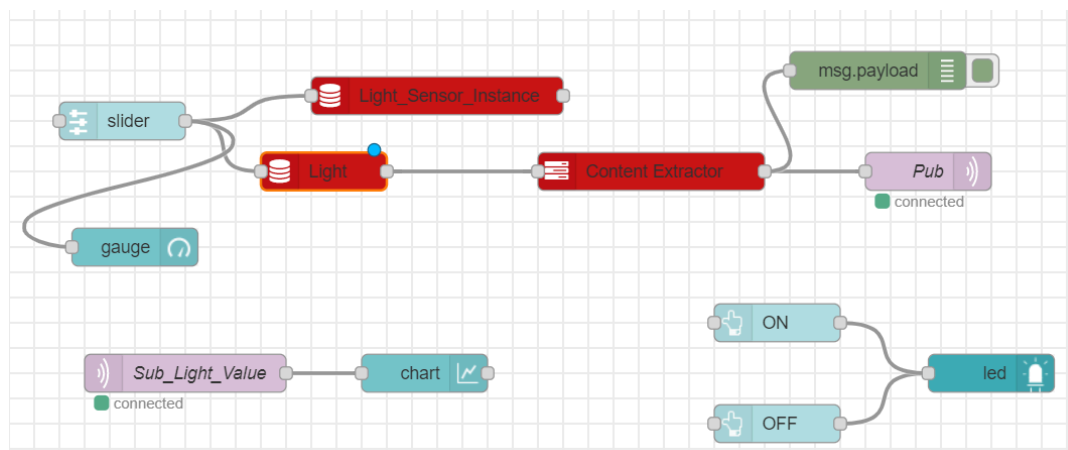
In this section, we crafted a dashboard interface integrating multiple graphs to visualize sensor data. Additionally, we incorporated interactive buttons allowing for the control of LED activation and deactivation.

We structured the dashboard system by employing the dashboard palette, resulting in the following setup:



Within this dashboard, the slider function is utilized to transmit varying light values to the light sensor, simulating changes in light intensity. The initial LED indicator conveys whether the light values exceed 250 lux.

Subsequently, the chart visually displays the values sent to the light sensor, while two buttons are employed to control the activation of the second LED. The Node-RED flow corresponding to this setup is as follows:



#### 4. Conclusion

In conclusion, Node-RED emerges as a versatile platform enabling data retrieval, utilization in actuation, and even dashboard creation. Its strength lies in the extensive array of plugins it offers, facilitating diverse system representation for both dashboards and workflow configurations. The abundance of nodes available ensures adaptability to various technologies prevalent in the market, such as MQTT nodes and more.

However, one potential drawback pertains to the dashboard design interface. While Node-RED provides powerful functionalities, alternative tools like Blynk might offer simpler and more enjoyable interfaces, especially for the creation of mobile application-based dashboards.

Moreover, although due to time constraints we were unable to implement the section involving the utilization of other nodes in Node-RED to connect with email, social networks, etc., the understanding of this process remains within our grasp. By integrating specialized nodes available within Node-RED, one can potentially establish connections with email services, social media platforms, and more, expanding the application's functionality and reach beyond traditional IoT interactions.