

Lab : Middleware for IoT

Rémi Goulard - Pauline Dupuy

Part 1 : ESP8266 & MQTT protocol

Objective of the lab : In this lab we will become familiar with a protocol very useful in an IoT environment : MQTT. After discovering the main MQTT characteristics, we will use various softwares to develop an application on a ESP8266, communicating with a server through MQTT.

1. What is MQTT protocol

- Typical architecture of an IoT system based on the MQTT protocol :

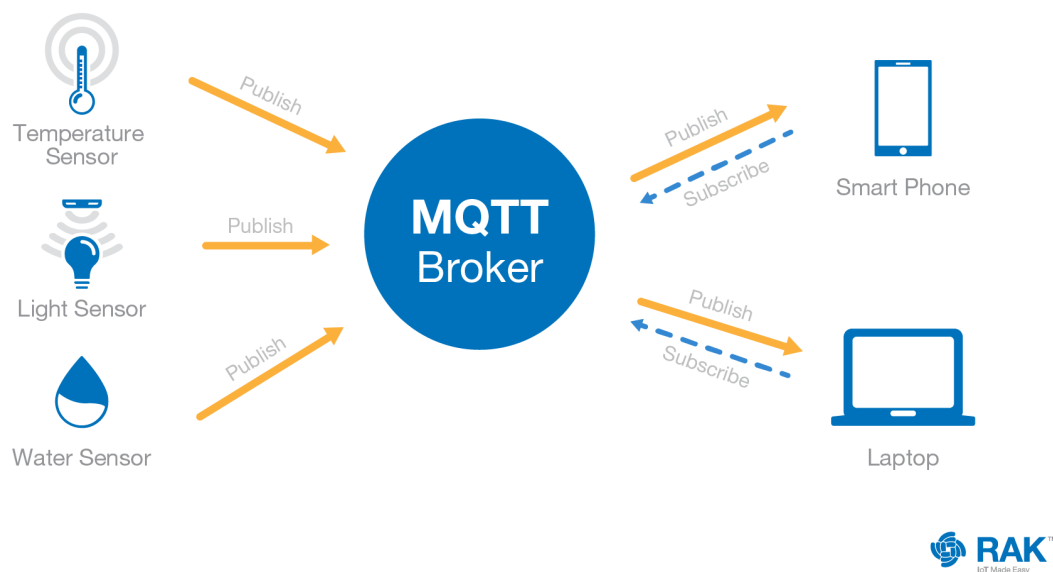


Figure 1 : typical IoT system architecture based on MQTT

MQTT based IoT architecture is based on the subscribe/publish model. The architecture is composed of 3 categories of entities :

- **Publishers** : devices like sensors which publish their data on a specific channel called 'topic'
- **Subscribers** : devices like laptops which can subscribe to a 'topic' to know all the informations published on it
- **Brokers** : devices in the middle, responsible for transmitting messages published on topics to all the subscribed entities.

- IP protocol under MQTT and characteristics :

MQTT is based on TCP/IP => TCP is a communication protocol with connection, so it ensures a minimum of QoS (Quality of Service). MQTT is not heavy in terms of bandwidth usage, it supports a maximum payload of 256MB.

- What are the different versions of MQTT :

MQTT was created in 1999 and had known 4 versions so far :

- MQTT 1 : 1999
- MQTT 3 : 2010
- MQTT 4 : 2018
- MQTT 5 : 2019

Those are the official releases of MQTT, but as an open protocol it's possible to find a lot of specific versions of MQTT on repositories such as GitHub.

Therefore you can also find various MQTT agents, such as [ActiveMQ \(en\)](#) / [ejabberd](#) / JoramMQ, [OW2 JORAM](#) / Mosquitto and many more.

- Security - authentication - encryption used in MQTT :

As in many other fields, security and integrity of data are very important when creating an IoT system. To protect the customer, protocols used have to present some security features.

In the case of the MQTT protocol, it is based on TCP/IP, and can therefore take advantage of SSL/TLS (Secure Socket Layer / Transport Layer Security) to add some security. Indeed, more authentication is required. But this security enhancement is not always possible, as it implies more costs.

- MQTT architecture to automatically switch on a light bulb when the luminosity is under a threshold.

In this case we have a button, a connected light bulb and a luminosity sensor.

We consider there is a broker between those devices. The button and the sensor would publish on 1 topic : 'light status'.

If the luminosity goes beyond the threshold => sensor publishes on the 'light status' and the light turns on/off.

If someone presses the button => button publish on 'light status' and the light turns on/off.

The bulb on its side would be subscribed to the topic, and turn on/off according to what is published by the button and the sensor.

2. Installation of the broker and tests

For this lab we use the Mosquitto broker version of MQTT, available on Windows operating system.

The first step is to download and test the mosquito broker :

```
C:\Program Files\mosquitto>mosquitto_sub -t TP1  
ExempleMessage
```

Figure 2 : subscription to topic 'TP1' using mosquitto

```
C:\Program Files\mosquitto>mosquitto_pub -t TP1 -m ExempleMessage
```

Figure 3 : message published on topic 'TP1'

On the previous figures, we can see the 3 different steps :

- First terminal subscribe to 'TP1' topic and waits for a message to be published
- Second terminal publish a message on this topic : 'ExempleMessage'
- First terminal receives the message published on the topic it follows => the broker correctly transmitted the message.

3. Creation of an IoT device on ESP8266 communicating with MQTT.

In this part, we will use an external device (ESP8266) to simulate the publish/subscribe devices.

On the ESP8266 we will use the NodeMCU board. This board supports Wi-Fi chips supporting IP, that allows us to use MQTT protocol. To use the ESP8266 as a subscribe/publish device, we will use Arduino IDE and C++ language.

According to its technical reference, ESP8266 gives access to 16 GPIO (Input/output), from GPIO00 to GPIO15.

The first step is to configure the NodeMCU board on Arduino IDE, so our ESP8266 can act as our publish/subscribe devices :

- Add the board NodeMCU 09 (ESP-12 module) on Arduino IDE
- Add the ArduinoMqtt library (by Oleg Kovalenko)
- Open the file in the menu:
examples/arduinoMqtt/connectESP8266wifi.c and modify the code to connect to our own Network (my smartphone) :

```
const char* ssid = "Remi's phone";  
const char* password = "okayokay";
```

Figure 4 : Network connexion setup

```
WiFi.begin(ssid, password);
```

Figure 5 : Network connexion phase

- Modify the address of the MQTT server to be able to connect to the broker on your laptop :

```
//my computer IP address
const char* mqtt_server = "192.168.56.1"

network.connect(mqtt_server, 1883);
```

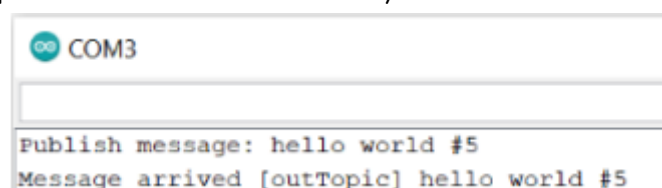
Figure 6-7 : MQTT server address modification & connexion

- Add a publish/subscribe behavior in the ESP8266 device :

```
// Subscribe
{
  MqttClient::Error::type rc = mqtt->subscribe(
    outTopic, MqttClient::QOS0, processMessage
  );
  if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTF("Subscribe error: %i", rc);
    LOG_PRINTF("Drop connection");
    mqtt->disconnect();
    return;
  }
}
else {
  // Publish
  {
    const char* buf = "Hello World";
    MqttClient::Message message;
    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;
    message.payload = (void*) buf;
    message.payloadLen = strlen(buf);
    mqtt->publish(outTopic, message);
  }
}
```

Figure 8 : ESP8266 publish/subscribe behavior

- Finally we test our device behavior and look on the IDE console the messages published and received by the board :



The screenshot shows the Arduino IDE console window with the title 'COM3'. It displays two lines of text: 'Publish message: hello world #5' and 'Message arrived [outTopic] hello world #5'.

Figure 9 : ESP8266 published/subscribed messages

4. Creation of a simple application

In this part of the lab, we use the previous knowledge about connecting to a network, and using an ESP8266 to publish/subscribe to topics to implement the exemple from part 2.

The following figure describes the architecture behavior :

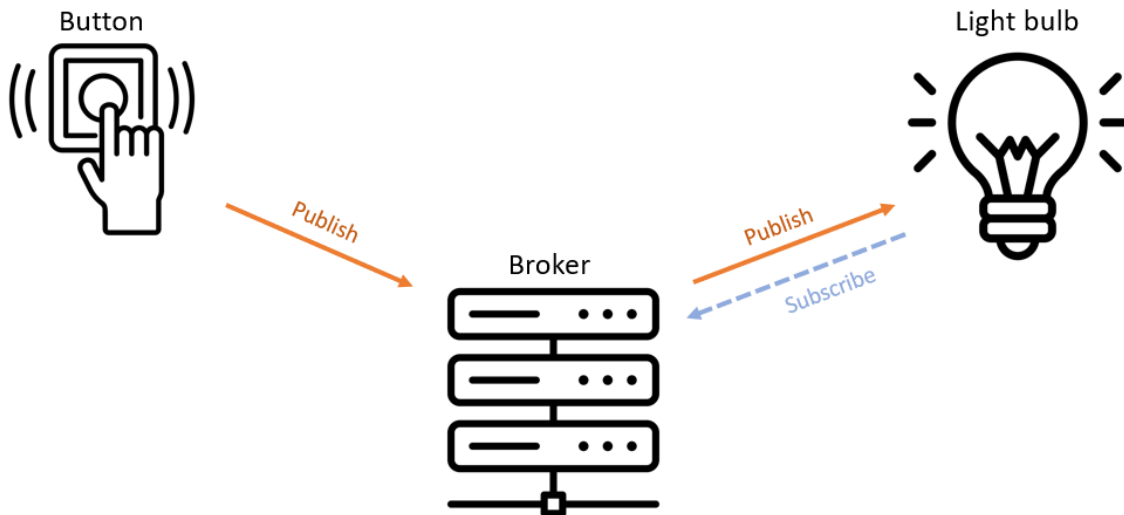


Figure 10 : application architecture

In this case, the light bulb will be represented by the inter LED of the ESP8266. First, we create a callback() function that reads the message on the wanted topic and turn on the LED if the message is '1' and off if '0' :

```
// ===== Callback () =====
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    for (int i = 0; i<length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();

    //turn up the LED on message '1'
    if ((char)payload[0]=='1') {
        digitalWrite(BUILTIN_LED, LOW); //turn on if volt level is low (inversed on ESP8266)
    } else {
        digitalWrite(BUILTIN_LED, HIGH); //turn the lLED off
    }
}
```

Figure 11 : callback() function => to turn the LED on/off

Now we have a function to turn the LED on and off, we setup the intern LED :

```
//setup the LED  
pinMode(BUILTIN_LED, OUTPUT);
```

Figure 12 : setup the LED

Finally, we add in the loop part some code to publish a message ('1' or '0') on the topic, depending on the button state :

```
if (digitalRead(buttonPin)==HIGH) {  
    snprintf (msg, MSG_BUFFER_SIZE, "1", value);  
    client.publish("outTopic",msg);  
} else {  
    snprintf (msg, MSG_BUFFER_SIZE, "0", value);  
    client.publish("outTopic", msg);  
}
```

Figure 13 : setup the LED

The LED is correctly working.

5. Conclusion

During the lab, we discovered the MQTT protocol, its characteristics, how to implement an MQTT architecture using mosquitto as a broker, and finally how to create an application on the nodeMCU board that uses MQTT.

Part 2 : Middleware based on oneM2M standard

Objective of the lab : In this lab, we want to focus on oneM2M standard. We will use one implementation of oneM2M called ACME to manipulate the service layer of the standard, and understand how to create and use various resources.

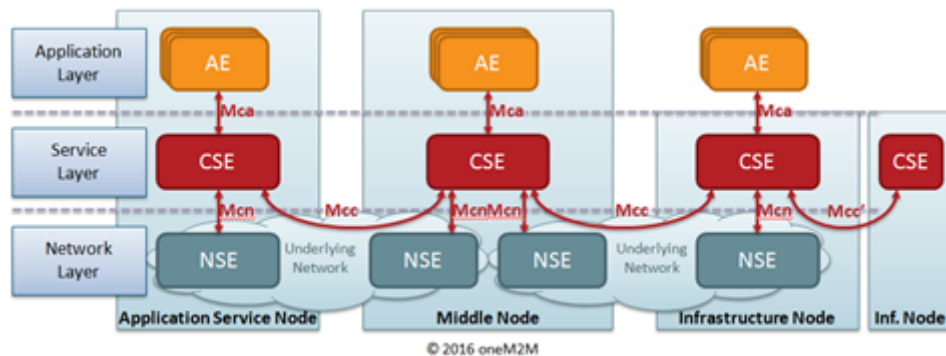


Figure 14 : oneM2M standard architecture

1. Installation and test of ACME for oneM2M standard

As there are various implementations of oneM2M standard, we will use the one called ACME, disponible at the address ["https://github.com/ankraft/ACME-oneM2M-CSE"](https://github.com/ankraft/ACME-oneM2M-CSE).

To test our implementation, we launch an instance and use the console command to show the resource tree.

```
goalard@insa-20665:~/Bureau/5A/Middleware_for_IoT/ACME-oneM2M-CSE$ python3 -m acme
```

```
[ACME] 2023.10.1 - An open source CSE Middleware for Education
```

Figure 15 : ACME IN-CSE instance => 'python3 -m acme'

Resource Tree

```
cse-in -> m2m:cb (csi=/id-in) | ri=id-in
├─ acpCreateACPs -> m2m:acp | ri=acpCreateACPs
└─ CAdmin -> m2m:ae | ri=CAdmin
```

Figure 16 : IN-CSE resource tree => 't' console command

We have correctly configured the ACME package and are now able to create CSE's and simulate a oneM2M standard architecture.

2. ACME CSE manipulation through notebook

Now we have a working oneM2M environment working via ACME, it is required to understand how to programmatically interact with a oneM2M CSE, create and update resources. For this purpose, we follow a series of Notebooks available at the address :

["https://github.com/ankraft/onem2m-jupyter-notebooks"](https://github.com/ankraft/onem2m-jupyter-notebooks).

We are not going to give too many details about the Notebook, but instead give a few clues about what we learnt in each one.

To open a Notebook, we use the following command :

```
goalard@insa-20665:~/Bureau/5A/Middleware_for_IoT/onem2m-jupyter-notebooks$ jupyter notebook --NotebookApp.token='' 01-introduction.ipynb
```

Figure 17 : command to launch the wanted notebook (here 01...)

List of ACME notebooks :

- **Chapter 1 : Introduction**
Retrieve a <CSEBase> resource : gives all informations about the CSE
- **Chapter 2 : Basic resources**
How to create and use resources : <AE>|<Container>|<ContentInstance>
=> use HTTP requests ⇔ Retrieve / Create / Update / Delete

```
CSE Resource Tree
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├─ acpCreateACPs -> m2m:AccessControlPolicy | resourceID=acpCreateACPs
├─ CAdmin -> m2m:ApplicationEntity | resourceID=CAdmin
├─ Notebook-AE -> m2m:ApplicationEntity | resourceID=Cmyself
├─ Container -> m2m:Container | resourceID=cnt3539575881735828455
├─ Notebook-AE-Test -> m2m:ApplicationEntity | resourceID=CmyselfTP
├─ Container-TP -> m2m:Container | resourceID=cnt7447610874859301366
└─ cin_AX5nS9Z6Tr -> m2m:ContentInstance | resourceID=cin4376931543156560605
```

Figure 18 : chapter 2 tree with our code to add resources (cf Annexes)

- **Chapter 3 : Discovery**
Use label to dynamically discover resources => easier to find a few resources at the same time
- **Chapter 4 : Groups**
Create groups containing various resources (allowed to be different types) to access them at the same time => easier to retrieve / update

```
CSE Resource Tree
cse-in -> m2m:CSEBase (CSE-ID=/id-in) | resourceID=id-in
├─ acpCreateACPs -> m2m:AccessControlPolicy | resourceID=acpCreateACPs
├─ CAdmin -> m2m:ApplicationEntity | resourceID=CAdmin
├─ StreetLight-AE-1 -> m2m:ApplicationEntity | resourceID=Cmyself
├─ Light-Container-1 -> m2m:Container | resourceID=cnt4121499088734830112
├─ StreetLight-AE-2 -> m2m:ApplicationEntity | resourceID=Cmyself2
├─ Light-Container-2 -> m2m:Container | resourceID=cnt1328092263083093884
└─ Lights-Group -> m2m:Group | resourceID=grp3193501732509733708
```

Figure 19 : chapter 4 resource tree with groups

- **Chapter 5 : Access control policy**
Use <ACP> to control the accesses from originators to resources.
Initially, only the creator of a resource can access it (to do actions) but the <ACP> can change that.
Careful : when <ACP> is created, the original creator 'loses' its right on its child resources, its rights have to be explicit in the <ACP>.
- **Chapter 6 : Notifications**
Subscribe to changes of resources and receive notifications.

3. Simulated device application

In the last part of this lab, we use ACME to create an application that simulates the behavior of our button - light system.
The simulated application turns on/off a light depending on the value of the button (0 or 1). We use various resources : <AE> <Containers> <ContentInstances> according to the figure below :

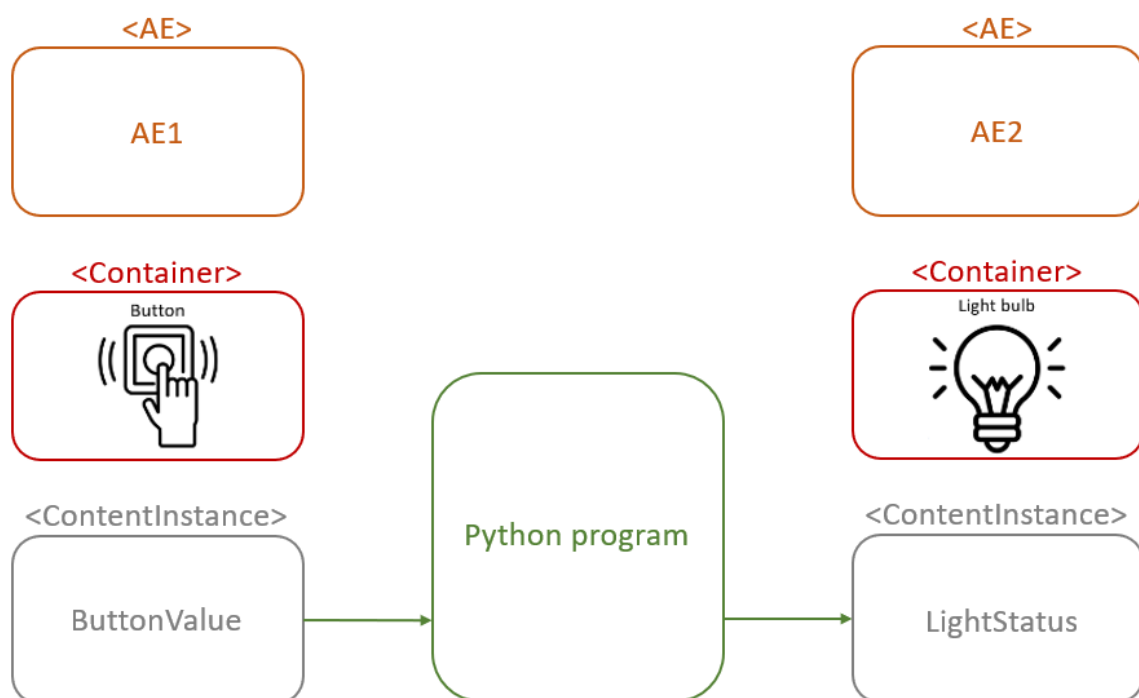


Figure 20 : simulated application oneM2M architecture

The first step is to create those resources. To do it, we implement 3 different python codes :

- CreateAE :

```
# Methode to create an Application entity
def createAE1():
    payload = '{ \
        "m2m:ae": { \
            "rn" : "AE1", \
            "api" : "NAE1", \
            "rr" : true, \
            "srv" : ["3"] \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=2",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in",data=payload,headers=_headers)
    handleResponse(r)
```

Figure 21: python code to create an AE (AE1 here)

- CreateContainer :

```
# Methode to create a Container
def createButton():
    payload = '{ \
        "m2m:cnt": { \
            "rn" : "Button" \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=3",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE1",data=payload,headers=_headers)
    handleResponse(r)
```

Figure 22 : python code to create a Container (Button here)

- CreateContentInstance :

```
# Methode to create a ContainerInstance
def createButtonValue():
    payload = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "0" \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE1/Button",data=payload,headers=_headers)
    handleResponse(r)
```

Figure 23 : python code to create a ContentInstance (ButtonValue here)

We use the previous codes to create our system described by the following resource tree :

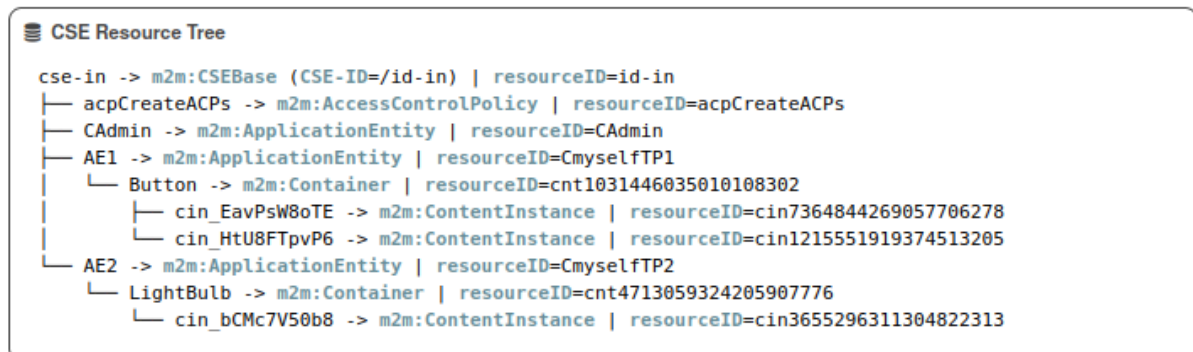


Figure 24 : system resource tree

Now we have our devices, we focus on creating a python code that emulates the behavior of the system. To do so, we will add 5 more functions (see annexe).

- SwitchButtonValueON : change button value from '0=>1'
- SwitchButtonValueOFF : change button value from '1=>0'
- SwitchLightStatusON : change light status from "off=>on"
- SwitchLightStatusOFF : change light status from "on=>off"
- FindButtonValue : returns the button value, to know if light status should be changed
- FindLightStatus : returns the status of the light => to check if the previous functions are working or not

Finally we create a little python program to test a use case of our functions and see if it really emulates the system :

```
createAE1()
createAE2()
createButton()
createBulb()
createButtonValue() # original value is '0'
createLightStatus() # original status is 'off'

while 1 :
    # Switch light to 'on' by pressing the button
    SwitchButtonValueON()
    x=FindButtonValue()
    buttonvalue=x[44] # recuperate the value of the button at the time
    if buttonvalue==1 :
        SwitchLightStatusON()

    # Switch light to 'off' by pressing the button
    SwitchButtonValueOFF()
    x=FindButtonValue()
    buttonvalue=x[44] # recuperate the value of the button at the time
    if buttonvalue==0 :
        SwitchLightStatusOFF()

    time.sleep(5)
```

Figure 25 : system emulation through oneM2M standard

In this use case, the start state of the system is :

- button to '0'
- light status to 'off'

Then we change the button value to '1' and see if the light status moved to 'on'
=> it does !

We used a **while 1** loop so any kind of use cases could be implemented instead of the one we developed.

Part 3 : Fast application prototyping for IoT

Objectives of the lab : In this lab, we want to use all the notions seen in previous labs to create a complete application that interacts with several real and virtual devices. The idea is to base ourselves on oneM2M to make various devices with different interfaces communicate using only one type of protocol. The other objective is to learn how to use RED NODE to develop faster.

1. Node RED configuration

Node RED is a programming tool used for wiring hardware devices, API's and online services. It provides a browser based editor with a huge range of nodes easy to use and wire.

To use Node RED, we first install a working Node.js framework, then use the command `"npm install -g --unsafe-perm node-red"` to add the Node RED package.

The second step is to install "LOM2M-Node-red", a package that gives access to various nodes, that will be the base for our oneM2M architecture.

Once all the installation is done, we can access the Node-RED editor at the address <http://localhost:1880>.

2. Applications using Node RED

We will deploy warriors applications to see how Node RED is working :

1. Get sensors values and display them :

We need 5 nodes :

- **Names Sensor Data** => takes information about the CSE (AE/Container/ContainerInstance) to know which CSE to retrieve data from.
- **Timestamp** => trigger to activate sensor data node.
- **Content Extractor** => retrieve information from 'sensor data' node.
- **Debug** => act as a display for retrieved data
- **MQTT** => to post/retrieve the light value on ESP8266

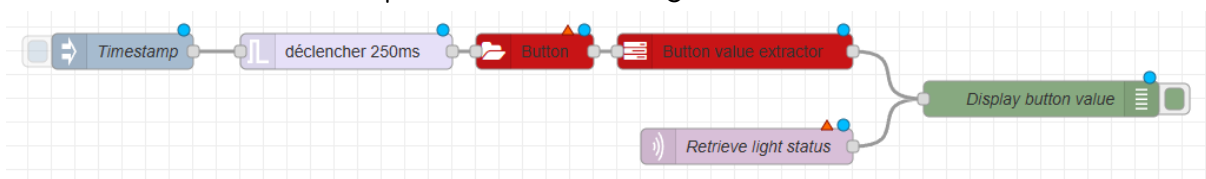


Figure 26 : data visualization system => simulated device (up) VS real one (down)

2. Sensor and actuator :

For this exemple, we retrieve on the corresponding topic the luminosity value measured on the ESP8266. Depending on this value, we use a switch to decide either or not to turn on a LED. This choice is sent on another topic, on which the ESP8266 is listening.

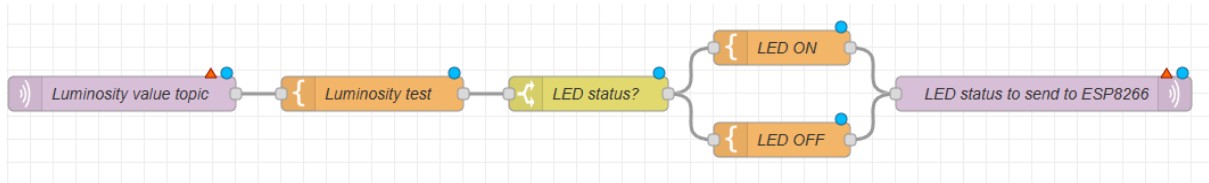


Figure 27 : luminosity retrieve & LED status command

3. Dashboard :

Create a dashboard that plots values of sensors with different graphs and buttons to switch on/off leds.

We create a dashboard, on which we plot the values of the luminosity sensor, and a light that represents the LED status of ESP8266. Lastly, we put a switch, and use a chart node to represent the value sent to the ESP8266 ('1' to turn on the LED and '0' to turn it off). This chart node is yellow when '1' is sent and gray when '0' is sent. The button is also an input, as it publishes on the topic.

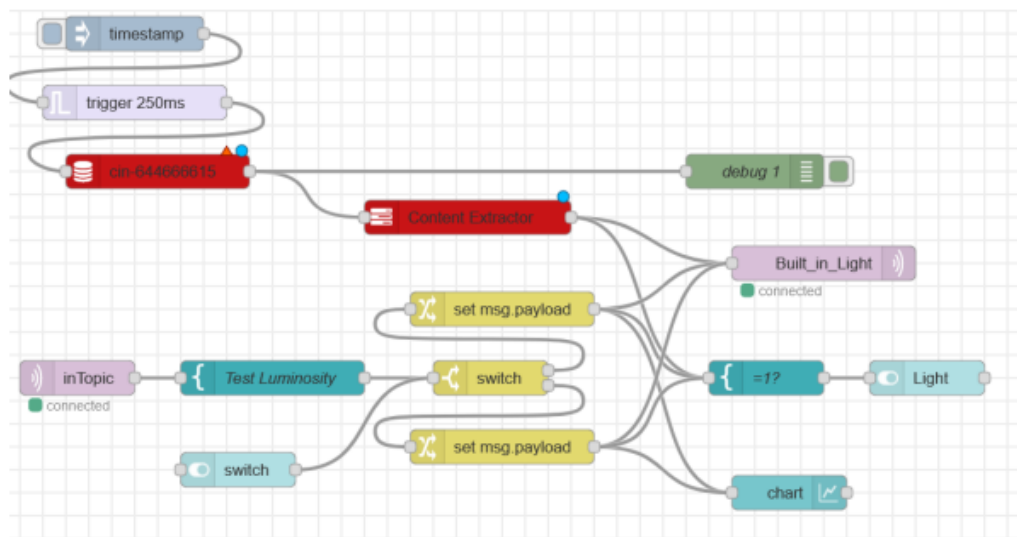


Figure 27 : Application flow

Application test : we put a trigger at 18, and publish 2 messages, the first with a value under 18 (15) and the second with a value over 18 (25). We are suppose to see on the chart the light turns off then on :

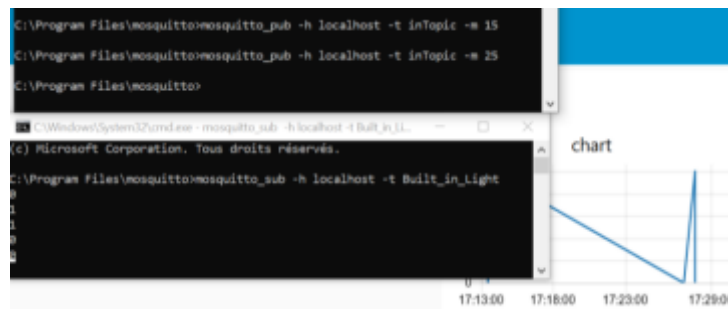


Figure 27 : Dashboard visualization => down then up => ok !

3. Conclusion

In this lab, we have tried to mix all we have seen in the previous labs. We also discovered Node RED, to have a faster integration of IoT devices communication, either real or simulated devices. Node RED is a powerful tool as it's open source => various node packages developed by the community and constant improvement. On the other hand, its web browser editor seems to be a constraint, as it might be difficult to simulate complex projects with a lot of devices (too complex to see), which limits its scalability.

Annexes

Annexe 1: code for chapter 2

```
import sys
import requests
import json

def handleResponse(r):
    print (r.status_code)
    print (r.headers)
    print (r.text)
    return;

# Methode to create an Application entity
def createAE():
    payload = '{ \
        "m2m:ae": { \
            "rn" : "Notebook-AE-Test", \
            "api" : "Ncabecou", \
            "rr" : true, \
            "srv" : ["3"] \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=2",
        "Accept": "application/json",
        "X-M2M-Origin": "CmyselfTP",
        "X-M2M-RI": "123",
        "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in",data=payload,headers=_headers)
    handleResponse(r)

# Methode to create a Container
def createCT():
    payload = '{ \
        "m2m:cnt": { \
            "rn" : "Container-TP" \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=3",
        "Accept": "application/json",
        "X-M2M-Origin": "CmyselfTP",
        "X-M2M-RI": "123",
        "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/Notebook-AE-Test",data=payload,headers=_headers)
    handleResponse(r)

# Methode to create a ContainerInstance
def createCTI():
    payload = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "cabecou" \
        } \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
        "Accept": "application/json",
        "X-M2M-Origin": "CmyselfTP",
        "X-M2M-RI": "123",
        "X-M2M-RVI": "3"
    }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/Notebook-AE-Test/Container-TP",data=payload,headers=_headers)
    handleResponse(r)

createAE()
createCT()
createCTI()

RETRIEVE (                                     # RETRIEVE request

    # Target the CSEBase itself
    to = cseBaseName,

    # Request Parameters
    originator = defaultOriginator, # Set the originator
    requestIdentifier = '123',       # Unique request identifier
    releaseVersionIndicator = '3',   # Release version indicator
)
```


Annexe 2 : python functions from TP2 application

```
def SwitchButtonValueON():
    # delete existing value '0'
    payload1 = '{ \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
                }
    json.dumps(json.loads(payload1,strict=False), indent=4)
    r = requests.delete("http://127.0.0.1:8080/cse-in/AE1/Button/la",data=payload1,headers=_headers)
    handleResponse(r)
    # create new value '1'
    payload2 = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "1" \
        } \
    }'
    create_headers = {"Content-Type": "application/json;ty=4",
                      "Accept": "application/json",
                      "X-M2M-Origin": "CmyselfTP1",
                      "X-M2M-RI": "123",
                      "X-M2M-RVI": "3"
                      }
    json.dumps(json.loads(payload2,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE1/Button",data=payload2,headers=create_headers)
    handleResponse(r)
```

```
def SwitchButtonValueOFF():
    # delete existing value '1'
    payload1 = '{ \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
                }
    json.dumps(json.loads(payload1,strict=False), indent=4)
    r = requests.delete("http://127.0.0.1:8080/cse-in/AE1/Button/la",data=payload1,headers=_headers)
    handleResponse(r)
    # create new value '0'
    payload2 = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "0" \
        } \
    }'
    create_headers = {"Content-Type": "application/json;ty=4",
                      "Accept": "application/json",
                      "X-M2M-Origin": "CmyselfTP1",
                      "X-M2M-RI": "123",
                      "X-M2M-RVI": "3"
                      }
    json.dumps(json.loads(payload2,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE1/Button",data=payload2,headers=create_headers)
    handleResponse(r)
```

```

def SwitchLightStatudON():
    # delete the 'off' status
    payload1 = '{ \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP2",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
                }
    json.dumps(json.loads(payload1,strict=False), indent=4)
    r = requests.delete("http://127.0.0.1:8080/cse-in/AE2/LightBulb/la",data=payload1,headers=_headers)
    handleResponse(r)
    # create the 'on' status
    payload2 = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "on" \
        } \
    }'
    create_headers = {"Content-Type": "application/json;ty=4",
                      "Accept": "application/json",
                      "X-M2M-Origin": "CmyselfTP2",
                      "X-M2M-RI": "123",
                      "X-M2M-RVI": "3"
                      }
    json.dumps(json.loads(payload2,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE2/LightBulb",data=payload2,headers=create_headers)
    handleResponse(r)

```

```

def SwitchLightStatusOFF():
    # delete the 'on' status
    payload1 = '{ \
    }'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP2",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
                }
    json.dumps(json.loads(payload1,strict=False), indent=4)
    r = requests.delete("http://127.0.0.1:8080/cse-in/AE2/LightBulb/la",data=payload1,headers=_headers)
    handleResponse(r)
    # create the 'off' status
    payload2 = '{ \
        "m2m:cin": { \
            "cnf" : "text/plain:0", \
            "con" : "off" \
        } \
    }'
    create_headers = {"Content-Type": "application/json;ty=4",
                      "Accept": "application/json",
                      "X-M2M-Origin": "CmyselfTP2",
                      "X-M2M-RI": "123",
                      "X-M2M-RVI": "3"
                      }
    json.dumps(json.loads(payload2,strict=False), indent=4)
    r = requests.post("http://127.0.0.1:8080/cse-in/AE2/LightBulb",data=payload2,headers=create_headers)
    handleResponse(r)

```

```

def FindButtonValue():
    payload = '{} '
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP1",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"
                }
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.get("http://127.0.0.1:8080/cse-in/AE1/Button/la",data=payload,headers=_headers)
    handleResponse(r)
    return r.text

```

```
def FindLightStatus():
    payload = '{}'
    _headers = {"Content-Type": "application/json;ty=4",
                "Accept": "application/json",
                "X-M2M-Origin": "CmyselfTP2",
                "X-M2M-RI": "123",
                "X-M2M-RVI": "3"}
    json.dumps(json.loads(payload,strict=False), indent=4)
    r = requests.get("http://127.0.0.1:8080/cse-in/AE2/Lightbulb/la",data=payload,headers=_headers)
    handleResponse(r)
    return r.text
```

Annexe 2 : Node RED flows

```
[
  {
    "id": "cc608cb54f52e12e",
    "type": "tab",
    "label": "Flow 1",
    "disabled": false,
    "info": "",
    "env": []
  },
  {
    "id": "9189b3b17df66a89",
    "type": "debug",
    "z": "cc608cb54f52e12e",
    "name": "debug 1",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "statusVal": "",
    "statusType": "auto",
    "x": 660,
    "y": 160,
    "wires": []
  },
  {
    "id": "0f41e27dee5aa9f6",
    "type": "mqtt in",
    "z": "cc608cb54f52e12e",
    "name": "",
    "topic": "inTopic",
    "qos": "2",
    "datatype": "auto-detect",
    "broker": "40bfe271dd677eb3",
    "nl": false,
    "rap": true,
    "rh": 0,
    "inputs": 0,
    "x": 70,
    "y": 340,
    "wires": [
      [
        "6bc205d64233b5c4"
      ]
    ]
  },
  {
    "id": "e0b25407fb58d5a5",
    "type": "Named Sensor Data",
    "z": "cc608cb54f52e12e",
    "name": "cin-644666615",
    "cseConfig": "09c69cc5628665fe",
    "seConfig": "d49340b1080344f1",
    "cntName": "DATA",
    "cin": "la",
    "x": 140,
    "y": 160,
    "wires": [
      [
        "ec469c7287aab180",
        "9189b3b17df66a89"
      ]
    ]
  },
  {
    "id": "ec469c7287aab180",
    "type": "Content Extractor",
    "z": "cc608cb54f52e12e",
    "name": "Content Extractor",
    "x": 450,
    "y": 200,
    "wires": [
      [
        "0cb728ab7d3cb4fe",
        "337f92796e8c4481"
      ]
    ]
  },
  {
    "id": "a7ebcb6f0ccefe38",
    "type": "trigger",
    "z": "cc608cb54f52e12e",
    "name": "",
    "op1": "1",
    "op2": "0",
    "op1type": "val",
    "op2type": "val",
    "duration": "250",
    "extend": "false",
    "overrideDelay": "false",
    "units": "ms",
    "reset": "",
    "bytopic": "all",
    "topic": "topic",
    "outputs": 1,
    "x": 120,
    "y": 100,
    "wires": [

```

```

        [
            "e0b25407fb58d5a5"
        ]
    ],
    {
        "id": "0ab9a4cbl1a27bb0c",
        "type": "inject",
        "z": "cc608cb54f52e12e",
        "name": "",
        "props": [
            {
                "p": "payload"
            },
            {
                "p": "topic",
                "vt": "str"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "topic": "",
        "payload": "",
        "payloadType": "date",
        "x": 120,
        "y": 40,
        "wires": [
            [
                "a7ebcb6f0ccefe38"
            ]
        ]
    },
    {
        "id": "6bc205d64233b5c4",
        "type": "SimpleCondition",
        "z": "cc608cb54f52e12e",
        "operator": "<",
        "value_c": "20",
        "inputType": "msg",
        "input": "payload",
        "name": "Test Luminosity",
        "x": 240,
        "y": 340,
        "wires": [
            [
                "3c5a5b5a27f83535"
            ]
        ]
    },
    {
        "id": "0cb728ab7d3cb4fe",
        "type": "mqtt out",
        "z": "cc608cb54f52e12e",
        "name": "",
        "topic": "Built_in_Light",
        "qos": "",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correl": "",
        "expiry": "",
        "broker": "40bfe271dd677eb3",
        "x": 720,
        "y": 240,
        "wires": []
    },
    {
        "id": "3c5a5b5a27f83535",
        "type": "switch",
        "z": "cc608cb54f52e12e",
        "name": "",
        "property": "payload",
        "propertyType": "msg",
        "rules": [
            {
                "t": "true"
            },
            {
                "t": "false"
            }
        ],
        "checkall": "true",
        "repair": false,
        "outputs": 2,
        "x": 430,
        "y": 340,
        "wires": [
            [
                "5be72a421cbade60"
            ],
            [
                "8ca922f2c0e56199"
            ]
        ]
    },
    {
        "id": "5be72a421cbade60",
        "type": "change",
        "z": "cc608cb54f52e12e",
        "name": "",
    }

```

```

      "rules": [
        {
          "t": "set",
          "p": "payload",
          "pt": "msg",
          "to": "1",
          "tot": "str"
        }
      ],
      "action": "",
      "property": "",
      "from": "",
      "to": "",
      "reg": false,
      "x": 440,
      "y": 280,
      "wires": [
        [
          "0cb728ab7d3cb4fe",
          "213f82a5df501a49",
          "337f92796e8c4481"
        ]
      ]
    },
    {
      "id": "8ca922f2c0e56199",
      "type": "change",
      "z": "cc608cb54f52e12e",
      "name": "",
      "rules": [
        {
          "t": "set",
          "p": "payload",
          "pt": "msg",
          "to": "0",
          "tot": "str"
        }
      ],
      "action": "",
      "property": "",
      "from": "",
      "to": "",
      "reg": false,
      "x": 440,
      "y": 400,
      "wires": [
        [
          "0cb728ab7d3cb4fe",
          "213f82a5df501a49",
          "337f92796e8c4481"
        ]
      ]
    },
    {
      "id": "34d7df065f5fa897",
      "type": "ui switch",
      "z": "cc608cb54f52e12e",
      "name": "",
      "label": "switch",
      "tooltip": "",
      "group": "4fd5ccc30c319ce7",
      "order": 0,
      "width": 0,
      "height": 0,
      "passthru": true,
      "decouple": "false",
      "topic": "topic",
      "topicType": "msg",
      "style": "",
      "onvalue": "true",
      "onvalueType": "bool",
      "onicon": "",
      "oncolor": "",
      "offvalue": "false",
      "offvalueType": "bool",
      "officon": "",
      "offcolor": "",
      "animate": false,
      "className": "",
      "x": 210,
      "y": 420,
      "wires": [
        [
          "3c5a5b5a27f83535"
        ]
      ]
    },
    {
      "id": "213f82a5df501a49",
      "type": "ui chart",
      "z": "cc608cb54f52e12e",
      "name": "",
      "group": "4fd5ccc30c319ce7",
      "order": 1,
      "width": 0,
      "height": 0,
      "label": "chart",
      "chartType": "line",
      "legend": "false",
      "xformat": "HH:mm:ss",
      "interpolate": "linear",
      "nodata": "",
      "dot": false,

```

```

    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 690,
    "y": 440,
    "wires": [
        []
    ]
},
{
    "id": "b082b17f40268ac8",
    "type": "ui_switch",
    "z": "cc608cb54f52e12e",
    "name": "",
    "label": "Light ",
    "tooltip": "",
    "group": "4fd5ccc30c319ce7",
    "order": 2,
    "width": 2,
    "height": 0,
    "passthru": true,
    "decouple": "false",
    "topic": "topic",
    "topicType": "msg",
    "style": "",
    "onvalue": "true",
    "onvalueType": "bool",
    "onicon": "fa-fire",
    "oncolor": "yellow",
    "offvalue": "false",
    "offvalueType": "bool",
    "officon": "fa-fire",
    "offcolor": "grey",
    "animate": true,
    "className": "",
    "x": 810,
    "y": 340,
    "wires": [
        []
    ]
},
{
    "id": "337f92796e8c4481",
    "type": "SimpleCondition",
    "z": "cc608cb54f52e12e",
    "operator": "=",
    "value_c": "1",
    "inputType": "msg",
    "input": "payload",
    "name": "=1?",
    "x": 670,
    "y": 340,
    "wires": [
        [
            "b082b17f40268ac8"
        ]
    ]
},
{
    "id": "40bfe271dd677eb3",
    "type": "mqtt-broker",
    "name": "",
    "broker": "localhost",
    "port": "1883",
    "clientId": "",
    "autoConnect": true,
    "usetls": false,
    "protocolVersion": "4",
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willPayload": "",
    "willMsg": {},
    "userProps": "",
    "sessionExpiry": ""
}

```

```

    },
    {
      "id": "09c69cc5628665fe",
      "type": "CSE_CONFIG",
      "cse": "MN LOM2M",
      "poa": "http://localhost:8080",
      "cseId": "mn-cse",
      "cseName": "mn-name",
      "adminOriginator": "024201ff2c7ac58b"
    },
    {
      "id": "d49340b1080344f1",
      "type": "AE_CONFIG",
      "aeName": "CAE371386530"
    },
    {
      "id": "4fd5ccc30c319ce7",
      "type": "ui_group",
      "name": "Switch",
      "tab": "6ddb4c724768617a",
      "order": 1,
      "disp": true,
      "width": "6",
      "collapse": false,
      "className": ""
    },
    {
      "id": "024201ff2c7ac58b",
      "type": "ORIGINATOR_CONFIG",
      "originatorName": "admin",
      "originator": "admin"
    },
    {
      "id": "6ddb4c724768617a",
      "type": "ui_tab",
      "name": "Light",
      "icon": "dashboard",
      "order": 1,
      "disabled": false,
      "hidden": false
    }
  ]
}

```