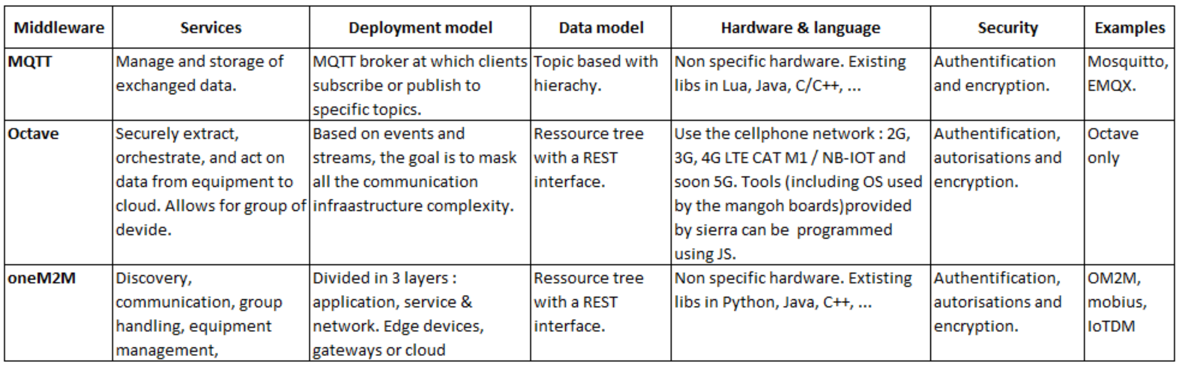
Report Middleware and Service

Baclé Lucas - Géhin Clément

November 2020

1 Introduction

This report will present the work we have done during the introduction to Internet of Things (IoT) middlewares course. IoT connects very diverse components, including the end devices and cloud application. Thus, some middlewares have been developed to act as a bond joining the heterogeneous domains of applications communicating over heterogeneous interfaces.



Middleware	Services	Deployment model	Data model	Hardware & language	Security	Examples
MQTT	Manage and storage of exchanged data.	MQTT broker at which clients subscribe or publish to specific topics.	Topic based with hierachy.	Non specific hardware. Existing libs in Lua, Java, C/C++, ...	Authentification and encryption.	Mosquitto, EMQX.
Octave	Securely extract, orchestrate, and act on data from equipment to cloud. Allows for group of devide.	Based on events and streams, the goal is to mask all the communication infraastructure complexity.	Ressource tree with a REST interface.	Use the cellphone network : 2G, 3G, 4G LTE CAT M1 / NB-IOT and soon 5G. Tools (including OS used by the mangoh boards)provided by sierra can be programmed using JS.	Authentification, autorisations and encryption.	Octave only
oneM2M	Discovery, communication, group handling, equipment management,	Divided in 3 layers : application, service & network. Edge devices, gateways or cloud	Ressource tree with a REST interface.	Non specific hardware. Extisting libs in Python, Java, C++, ...	Authentification, autorisations and encryption.	OM2M, mobius, IoTDM

Figure 1: Middlewares comparison

In the preceding figure, we present the major similarities and differences of the middlewares studied during this course. As we can see each one of them have pros and cons depending of the applications.

2 TP1 - Middleware for the IoT

2.1 State of art MQTT

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

MQTT is a “publish/subscribe” protocol. On one side you have sensors that publish data on a Broker. On the other side you have clients that connect to the Broker by subscribing. This subscribe is called a “topic”, it contains information about how the Broker can contact the client. In this architecture clients and publishers don’t have to know each other, which allows multi scale solutions.

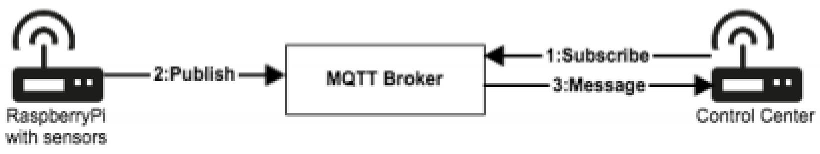


Figure 2: MQTT principle

2.1.2 What is the transport protocol under MQTT? What does it mean in terms of type of communication?

MQTT often uses TCP which means that it is a connected, octet-stream type of communication.

2.1.3 What are the different versions of MQTT?

There are two versions of MQTT :

- MQTT designed for TCP protocol
- MQTT-SN designed for UDP protocol (not really popular)

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

MQTT can send a username and password for client authentication and authorization. However, this information is not encrypted or hashed by default : the password is sent in plain text. To secure the communication we need to use a secure transport (TLS, SSL, ...). Brokers like HiveMQ can authenticate clients with an SSL certificate, so no username and password is needed.

2.1.5 What different topics will be necessary to get the following behavior and what will the connection be in terms of publishing or subscribing?

Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for you 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

We need at least two different topics in order to achieve these goals :

- *MQTT_TOPIC_LUMINOSITY*
- *MQTT_TOPIC_BUTTON*

Both the luminosity sensor and the switch will publish to their respective topics while the light will subscribe to both topics. The light will have the responsibility to decide whether it should be active or not.

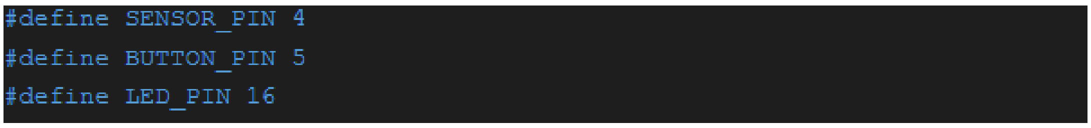
2.1.6 Give the main characteristics of nodeMCU board in term of communication, programming language, Inputs/outputs capabilities

The nodeMCU board is based on a Wi-Fi SoC ESP8266 conceived by Espressif Systems. Its firmware can use the Lua scripting language but in this lab we used the Arduino IDE and an ESP8266 Arduino core to compile an Arduino C/C++ source file for the ESP's machine language. It offers a rather wide array of inputs/outputs including GPIO and specific peripherals like ADC, SPI or UART.

2.2 Lab project : light management application

Our device will show the usage of MQTT through a sample light management application. It will both be the sensor and the actuator involved for the sake of simplicity.

To acquire the data from the sensor and the switch we connected them on the numeric inputs of the card. We connected the variable resistance on D2 and the switch on D1. Then in our code we defined constant as following :



```
#define SENSOR_PIN 4
#define BUTTON_PIN 5
#define LED_PIN 16
```

Figure 3: Pin definition

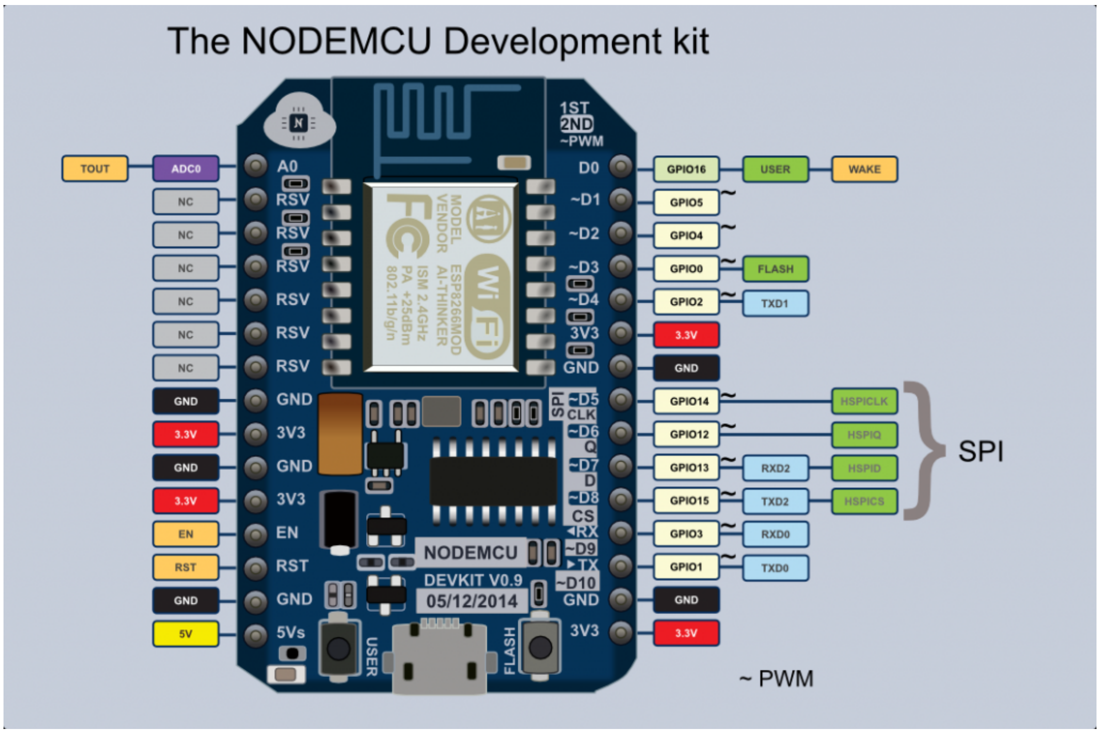
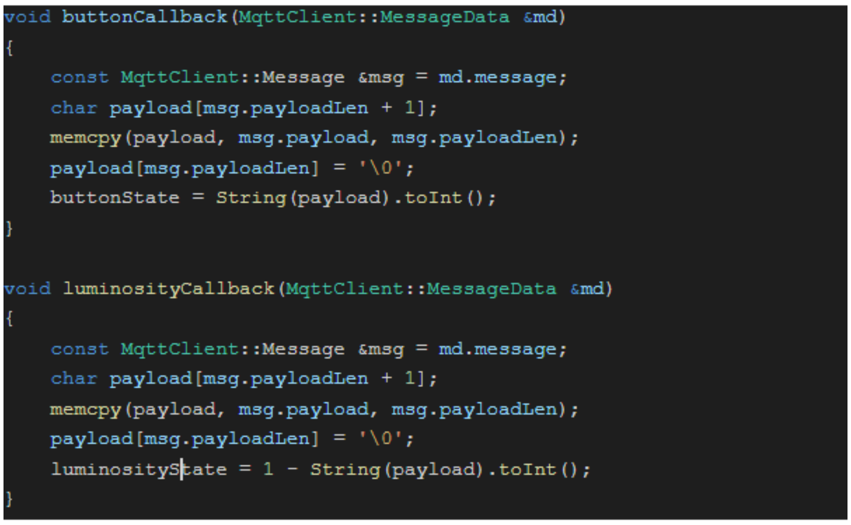


Figure 4: Card Diagram

Using the nodeMCU MQTT implementation we are subscribing to the *MQTT_TOPIC_LUMINOSITY* and *MQTT_TOPIC_BUTTON* topics. A specific callback function is instantiated for each message received by either of these topics.

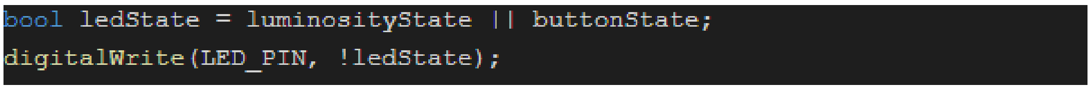


```
void buttonCallback(MqttClient::MessageData &md)
{
    const MqttClient::Message &msg = md.message;
    char payload[msg.payloadLen + 1];
    memcpy(payload, msg.payload, msg.payloadLen);
    payload[msg.payloadLen] = '\0';
    buttonState = String(payload).toInt();
}

void luminosityCallback(MqttClient::MessageData &md)
{
    const MqttClient::Message &msg = md.message;
    char payload[msg.payloadLen + 1];
    memcpy(payload, msg.payload, msg.payloadLen);
    payload[msg.payloadLen] = '\0';
    luminosityState = 1 - String(payload).toInt();
}
```

Figure 5: Callback functions

These callbacks function store the received value in the appropriate global variable. In the luminosity case we needed to subtract 1 to invert the measured value : light must be switched on when the sensor measures no lights.

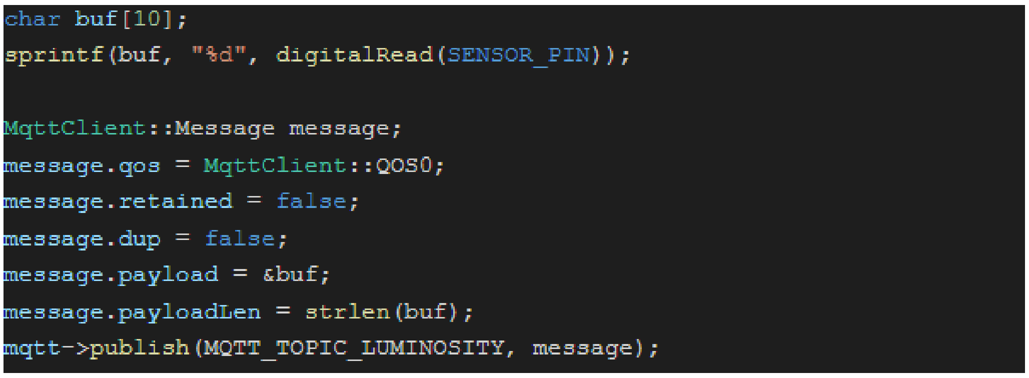
Inside of the main loop, we are running the light logic. The light must be switched on when either the switch has been pressed or the luminosity measured is below a specific threshold.



```
bool ledState = luminosityState || buttonState;
digitalWrite(LED_PIN, !ledState);
```

Figure 6: Light management

In order to update the different topics, the code is measuring the luminosity sensor value and the switch state then publishing it to the aforementioned topics every 30 seconds. We also could have sent a message when either value would have changed.



```
char buf[10];
sprintf(buf, "%d", digitalRead(SENSOR_PIN));

MqttClient::Message message;
message.qos = MqttClient::QOS0;
message.retained = false;
message.dup = false;
message.payload = &buf;
message.payloadLen = strlen(buf);
mqtt->publish(MQTT_TOPIC_LUMINOSITY, message);
```

Figure 7: Publishing the luminosity value

3 TP3 - Middleware for the IoT

3.1 Interact with the REST API

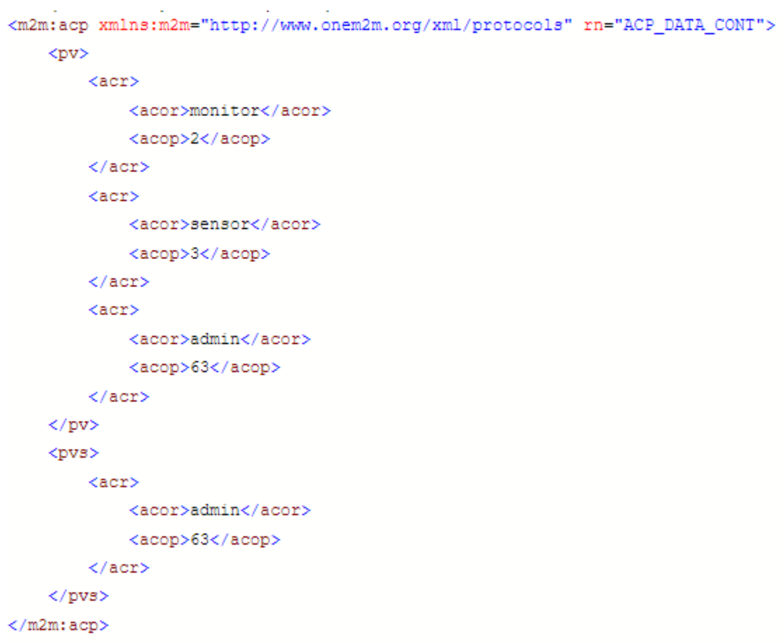
First, using Postman, we have created a collection of request to interact with the REST API. You can find this collection and import it into Postman using this link : `https://www.getpostman.com/collections/9165beca85a2fca6615d`.

3.2 Access Control Management in oneM2M

The aim is to give access to 2 different entities:

- a monitoring application that will only retrieve the data
- a sensor application that will be able to retrieve AND create

To do it we send a POST request with the following body :



```xml
<m2m:acp xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="ACP_DATA_CONT">
    <pv>
        <acr>
            <acor>monitor</acor>
            <acop>2</acop>
        </acr>
        <acr>
            <acor>sensor</acor>
            <acop>3</acop>
        </acr>
        <acr>
            <acor>admin</acor>
            <acop>63</acop>
        </acr>
    </pv>
    <pvs>
        <acr>
            <acor>admin</acor>
            <acop>63</acop>
        </acr>
    </pvs>
</m2m:acp>
```

Figure 8: POST request to control access

3.3 Scenario

In this section we had to realize the following scenario.

3.3.1 Create 3 AE on the MN

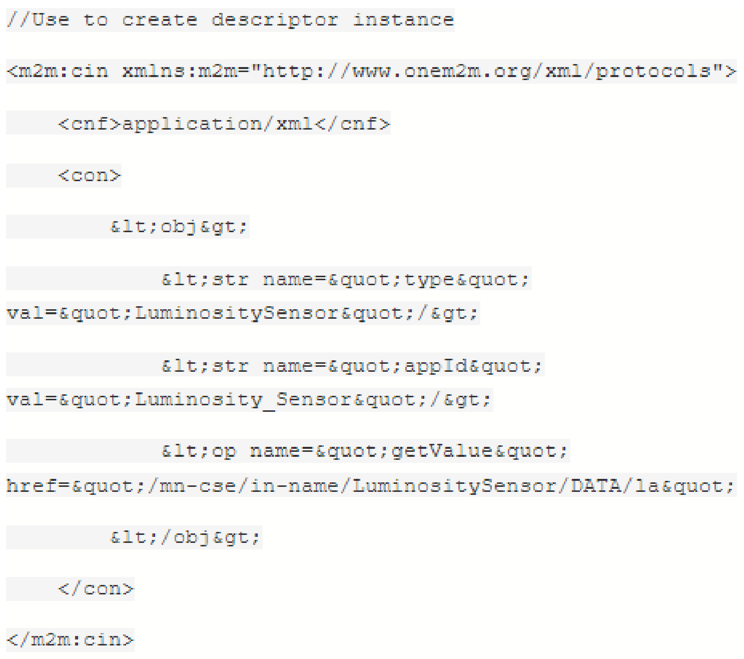
First, we had to create three AE : SmartMeter, LuminositySensor and TemperatureSensor. We also created two containers for each AE (DESCRIPTOR and DATA). To do so, we send three POST requests for each AE we wanted to create. This is an example for the luminosity sensor :

```
//Use to create an AE
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols"
rn="LuminositySensor" >
    <api>app-sensor</api>
    <lbl>Luminosity Sensor</lbl>
    <rr>false</rr>
</m2m:ae>


//Use to create a DESCRIPTOR container

<m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DESCRIPTOR">

</m2m:cnt>


//Use to create a DATA container

<m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DATA">

</m2m:cnt>
```

Figure 9: POST requests use to create the AE of the luminosity sensor

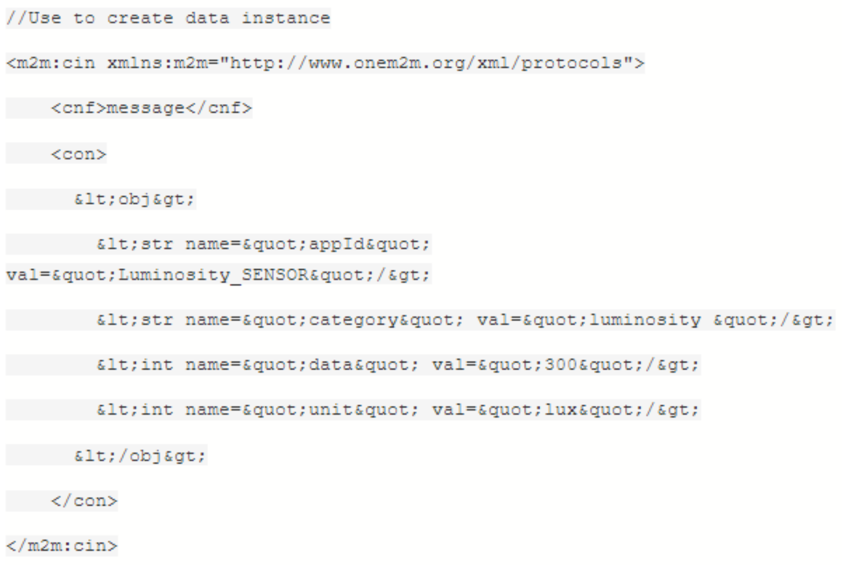
3.3.2 Create the content instances

Then we created instances for each AE in each container using the following code.



```
//Use to create descriptor instance

<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols">

    <cnf>application/xml</cnf>

    <con>

        &lt;obj&gt;

            &lt;str name=&quot;type&quot;
val=&quot;LuminositySensor&quot;/&gt;

            &lt;str name=&quot;appId&quot;
val=&quot;Luminosity_Sensor&quot;/&gt;

            &lt;op name=&quot;getValue&quot;
href=&quot;/mn-cse/in-name/LuminositySensor/DATA/la&quot;

        &lt;/obj&gt;

    </con>

</m2m:cin>
```

Figure 10: POST requests use to create the descriptor instance of the luminosity sensor

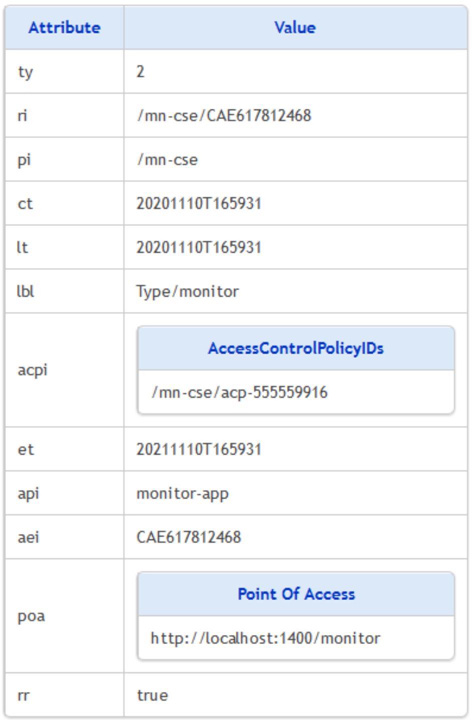


```
//Use to create data instance

<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols">

    <cnf>message</cnf>

    <con>

      &lt;obj&gt;

        &lt;str name=&quot;appId&quot;
val=&quot;Luminosity_SENSOR&quot;/&gt;

        &lt;str name=&quot;category&quot; val=&quot;luminosity &quot;/&gt;

        &lt;int name=&quot;data&quot; val=&quot;300&quot;/&gt;

        &lt;int name=&quot;unit&quot; val=&quot;lux&quot;/&gt;

      &lt;/obj&gt;

    </con>

</m2m:cin>
```

Figure 11: POST requests use to create the data instance of the luminosity sensor

3.3.3 Monitoring Application

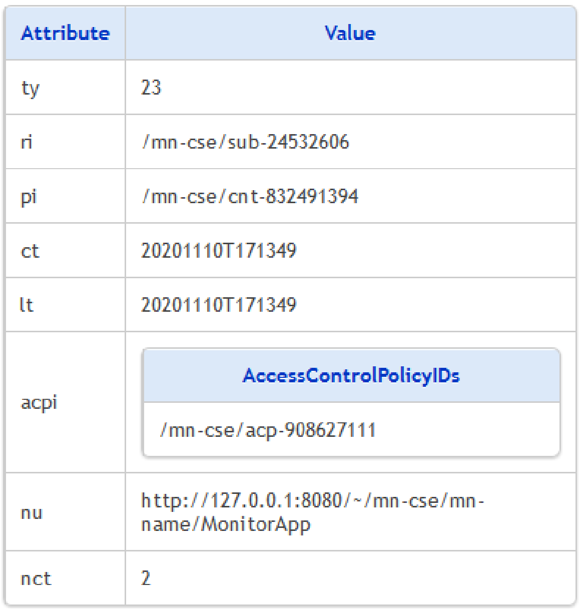
Finally, we used a monitor as a point of access to capture data when they are publish in the data container. To do so, we created a new AE with the rr attribute sets at true. We also define a point of access (POA) where to redirect the data.



Attribute	Value
ty	2
ri	/mn-cse/CAE617812468
pi	/mn-cse
ct	20201110T165931
lt	20201110T165931
lbl	Type/monitor
acpi	AccessControlPolicyIDs: /mn-cse/acp-555559916
et	20211110T165931
api	monitor-app
aei	CAE617812468
poa	Point Of Access: http://localhost:1400/monitor
rr	true

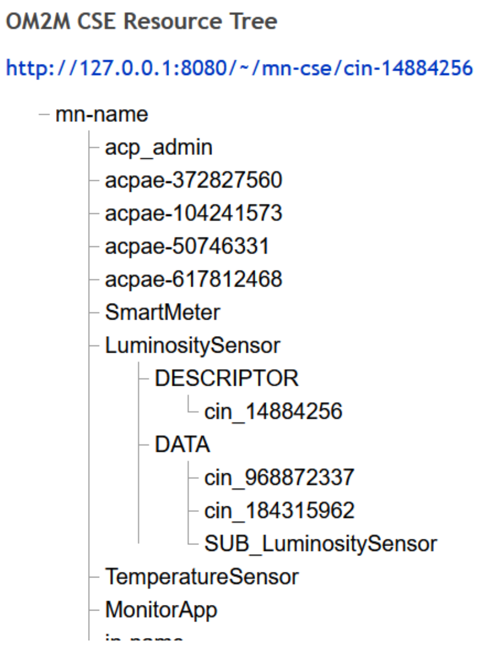
Figure 12: Configuration of the AE of the monitor

Then we subscribe the monitor AE to the luminosity sensor data container.



Attribute	Value
ty	23
ri	/mn-cse/sub-24532606
pi	/mn-cse/cnt-832491394
ct	20201110T171349
lt	20201110T171349
acpi	AccessControlPolicyIDs: /mn-cse/acp-908627111
nu	http://127.0.0.1:8080/~/mn-cse/mn-name/MonitorApp
nct	2

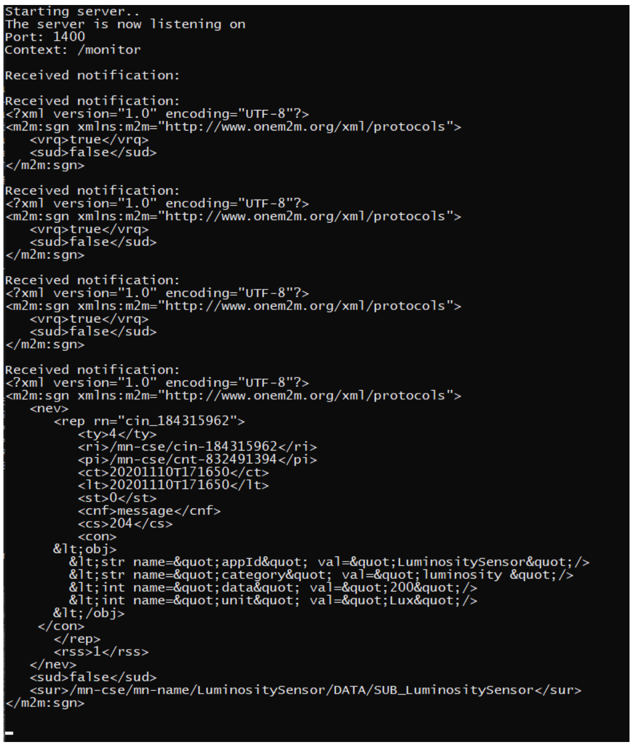
Figure 13: Subscription to the luminosity sensor data container



```
OM2M CSE Resource Tree
http://127.0.0.1:8080/~/mn-cse/cin-14884256
- mn-name
    acp_admin
    acpae-372827560
    acpae-104241573
    acpae-50746331
    acpae-617812468
    SmartMeter
    LuminositySensor
        DESCRIPTOR
            cin_14884256
        DATA
            cin_968872337
            cin_184315962
            SUB_LuminositySensor
    TemperatureSensor
    MonitorApp
```

Figure 14: Tree obtained at the end

Now, when a new instance is created in the data container, the information are also send to the POA given in the monitor AE. So, when we listen on that address we can see the new instance as below.



```
Starting server..
The server is now listening on
Port: 1400
Context: /monitor

Received notification:

Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <vrq>true</vrq>
   <sud>false</sud>
</m2m:sgn>

Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <vrq>true</vrq>
   <sud>false</sud>
</m2m:sgn>

Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <vrq>true</vrq>
   <sud>false</sud>
</m2m:sgn>

Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <nev>
      <rep rn="cin_184315962">
         <ty>4</ty>
         <ri>/mn-cse/cin-184315962</ri>
         <pi>/mn-cse/cnt-832491394</pi>
         <ct>20201110T171650</ct>
         <lt>20201110T171650</lt>
         <st>0</st>
         <cnf>message</cnf>
         <cs>204</cs>
         <con>
      &lt;obj>
        &lt;str name=&quot;appId&quot; val=&quot;LuminositySensor&quot;/>
        &lt;str name=&quot;category&quot; val=&quot;luminosity &quot;/>
        &lt;int name=&quot;data&quot; val=&quot;200&quot;/>
        &lt;int name=&quot;unit&quot; val=&quot;Lux&quot;/>
      &lt;/obj>
    </con>
      </rep>
      <rss>1</rss>
   </nev>
   <sud>false</sud>
   <sur>/mn-cse/mn-name/LuminositySensor/DATA/SUB_LuminositySensor</sur>
</m2m:sgn>

_
```

Figure 15: Display of the monitor in a console

4 TP4 - Fast application prototyping for IoT

4.1 Get sensors values and display them

First, we created a simple system of nodes in order to retrieve values from the work we did in TP1 and TP3, then we displayed them in the debug console.

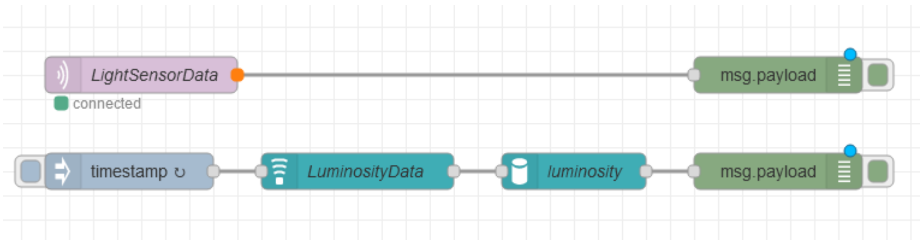
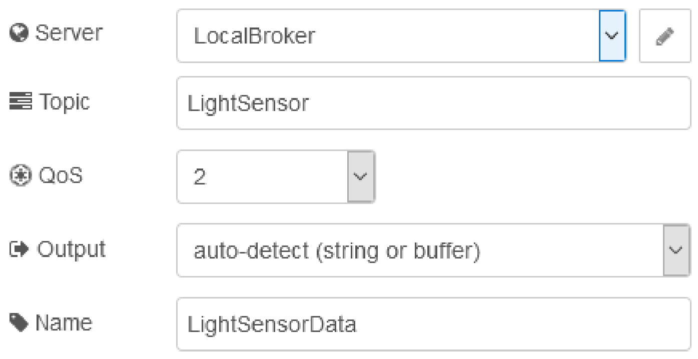
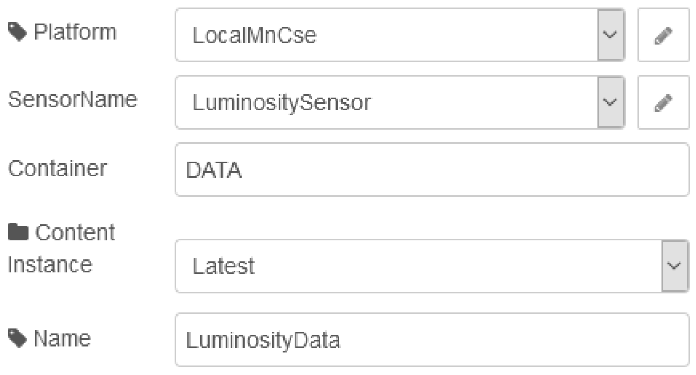


Figure 16: System of nodes to retrieve data from oneM2M and MQTT standard



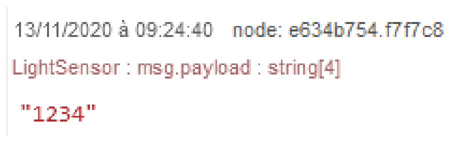
Server: LocalBroker
Topic: LightSensor
QoS: 2
Output: auto-detect (string or buffer)
Name: LightSensorData

Figure 17: Configuration of the MQTT light sensor node



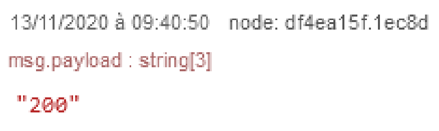
Platform: LocalMnCse
SensorName: LuminositySensor
Container: DATA
Content Instance: Latest
Name: LuminosityData

Figure 18: Configuration of the oneM2M luminosity sensor node



```
13/11/2020 à 09:24:40    node: e634b754.f7f7c8
LightSensor : msg.payload : string[4]
"1234"
```

Figure 19: Data received from MQTT node



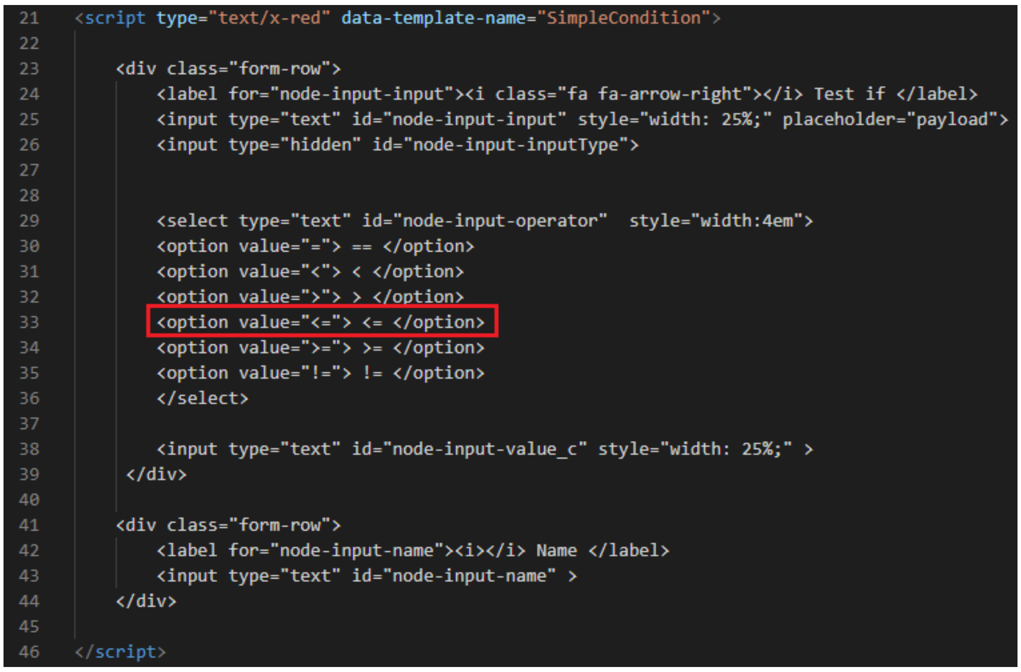
```
13/11/2020 à 09:40:50    node: df4ea15f.1ec8d
msg.payload : string[3]
"200"
```

Figure 20: Data received from oneM2M node

4.2 Sensors and activators

Here, we wanted to display the data received based on a condition. For example, if the quantity of light captured was inferior as a value that we chose.

While working on this part, we found a bug in the code of the simple condition node. There is a space that need to be delete when we choose the option inferior or equal. This space make the case go in default and does not behave as a inferior or equal but as an not equal.



```
<script type="text/x-red" data-template-name="SimpleCondition">

    <div class="form-row">
        <label for="node-input-input"><i class="fa fa-arrow-right"></i> Test if </label>
        <input type="text" id="node-input-input" style="width: 25%;" placeholder="payload">
        <input type="hidden" id="node-input-inputType">


        <select type="text" id="node-input-operator"  style="width:4em">
        <option value="="> == </option>
        <option value="<"> < </option>
        <option value=">"> > </option>
        <option value="<="> <= </option>
        <option value=">="> >= </option>
        <option value="!="> != </option>
        </select>

        <input type="text" id="node-input-value_c" style="width: 25%;" >
     </div>

    <div class="form-row">
        <label for="node-input-name"><i></i> Name </label>
        <input type="text" id="node-input-name" >
    </div>

</script>
```

Figure 21: Error in code corrected

On the figures below you will found our new system, the configuration of the simple node condition and the result with 900 as input.

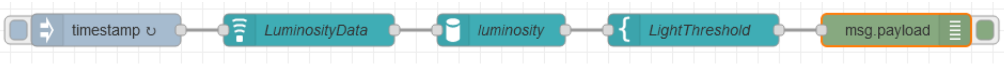
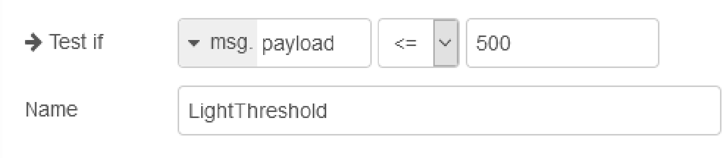


Figure 22: System of node using a simple condition node



Test if msg. payload <= 500

Name LightThreshold

Figure 23: Simple condition node configuration

4.3 Dashboard

Now that we are able to retrieve data and apply simple conditions on them, we wanted to display them on a dashboard. To do so, we replaced debug messages by dashboard nodes.

First, we created a new tabs ”home” and a first group that will contain all of our widgets :

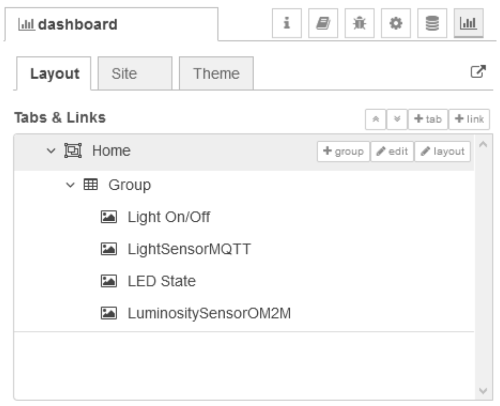


Figure 24: Dashboard nodes organization

Then, we added the following widgets to our node system :

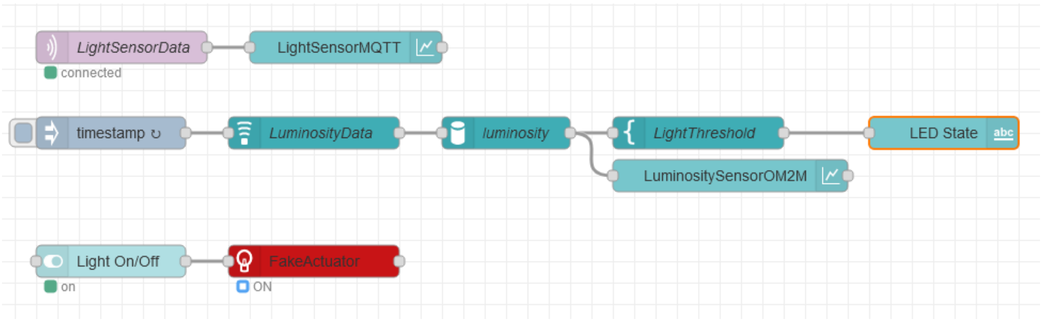


Figure 25: Node system with dashboard nodes

Finally, we obtained a dashboard reachable by a web page :

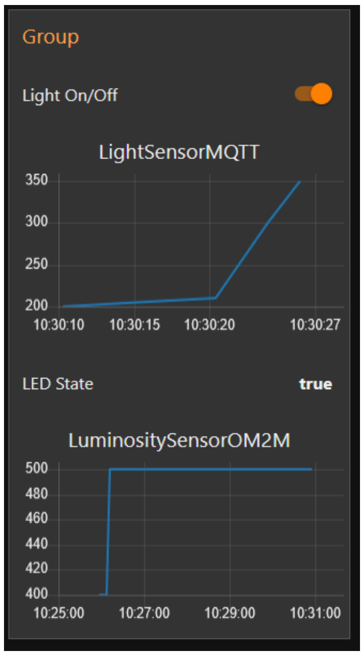


Figure 26: Our dashboard

4.4 Send an email

Finally, we wanted to send an email when the led is on. We used the email node from node-red-node-email that we configured to send an email to an specific email address using the INSA SMTP server.

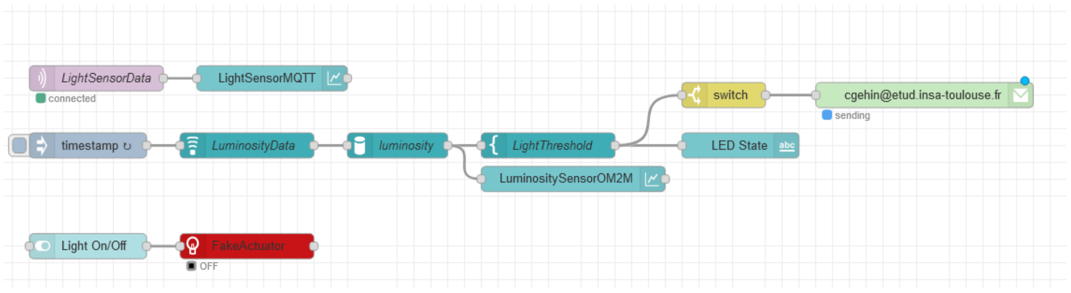
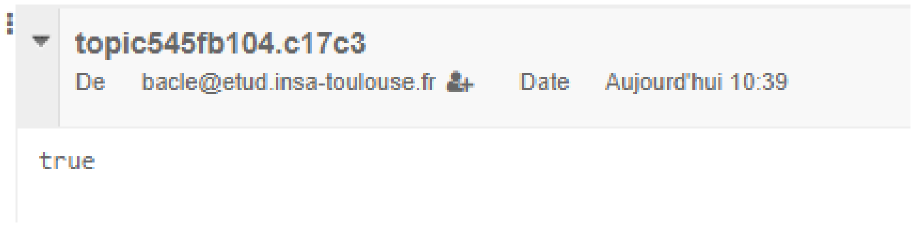


Figure 27: Node system able to send email



topic545fb104.c17c3

De bacle@etud.insa-toulouse.fr Date Aujourd'hui 10:39

```
true
```

Figure 28: Email received with the above system

5 Conclusion

During this classes we learned about different middlewares for Internet of Things. By experimenting we have discovered their similarities and their differences. Overall, we really enjoyed the way this course was presented and would have liked to discover more technologies.