Lab Report : Machine to Machine

Louis Chauvet-Smart
Alexandre Cros
INSA Toulouse - GEI - ISS B2

TP1 : MQTT

•What is the typical architecture of an IoT system based on the MQTT protocol?
publisher-subscriber:
the typical architecture follows this simple structure:

publisher -> Broker (mosquitto in our case) -> subscriber

The typical architecture of an IoT system based on the MQTT protocol is a server centralized architecture. Clients send or subscribe to exchange data with the server.

•What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc ?
MQTT is based on the TCP/IP protocol which allows low bandwidth usage and use equipment that can be battery efficient.

•What are the different versions of MQTT?
The latest version of MQTT is version 5.0, standardized by OASIS. There a 4 main version of MQTT in use today: MQTT v3.1.0, MQTT v3.1.1, MQTT v5, and MQTT-SN, with MQTT 3.1.1 being the most widely used.

•What kind of security/authentication/encryption are used in MQTT?
MQTT data can be secured by SSL authentication.

•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:

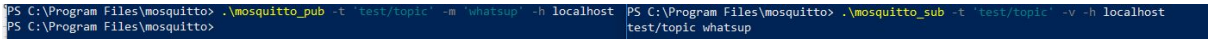
- **you would like to be able to switch on the light manually with the button**
- **the light is automatically switched on when the luminosity is under a certain value**

What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?
Three topics are needed. One for the switch, one for the light and one for the luminosity sensor. Switch and luminosity sensor will publish to the light (they need to change the light state).

Lab work:

Getting mosquitto to work was very finnicky: the series of commands was supposed to be:
.\mosquitto
.\mosquitto_sub -t ‘test/topic’ -v -h localhow
.\mosquitto_pub -t ‘test/topic’ -m ‘hello world’ -h localhost

however that didn't launch mosquitto correctly, even while using powershell as admin. double clicking the executable got it to work in the background though, and it managed to work after a while.



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

communication: ESP8266 Wi-Fi SoC

programming language: the firmware uses the Lua scripting language

I/O: USB and pins

I/O index	ESP8266 pin
0 [*]	GPIO16
1	GPIO5
2	GPIO4
3	GPIO0
4	GPIO2
5	GPIO14
6	GPIO12
7	GPIO13
8	GPIO15
9	GPIO3
10	GPIO1
11	GPIO9
12	GPIO10

•Open the file in the menu: exemples/arduinoMqtt/connectESP8266wificlient and have a look at the different parts ofthe code. Explain those different parts

```
#include <Arduino.h>
#include <ESP8266WiFi.h>

// Enable MqttClient logs
#define MQTT_LOG_ENABLED 1
```

```
// Include library
#include <MqttClient.h>


#define LOG_PRINTFLN(fmt, ...)    logfln(fmt, ##__VA_ARGS__)
#define LOG_SIZE_MAX 128
void logfln(const char *fmt, ...) {
   char buf[LOG_SIZE_MAX];
   va_list ap;
   va_start(ap, fmt);
   vsnprintf(buf, LOG_SIZE_MAX, fmt, ap);
   va_end(ap);
   Serial.println(buf);
}

#define HW_UART_SPEED                                                    115200L
#define MQTT_ID
"TEST-ID"

static MqttClient *mqtt = NULL;
static WiFiClient network;
definitions and parameters for the wifi connexion
// ============== Object to supply system functions ============================
class System: public MqttClient::System {
public:

   unsigned long millis() const {
         return ::millis();
   }

   void yield(void) {
         ::yield();
   }
};

// ============== Setup all objects ================================================
void setup() {
   // Setup hardware serial for logging
   Serial.begin(HW_UART_SPEED);
   while (!Serial);

   // Setup WiFi network
   WiFi.mode(WIFI_STA);
   WiFi.hostname("ESP_" MQTT_ID);
   WiFi.begin("ssid", "passphrase"); connect to specified network (my mobile hotspot)
   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());

  // Setup MqttClient
  MqttClient::System *mqttSystem = new System;
  MqttClient::Logger *mqttLogger = new MqttClient::LoggerImpl<HardwareSerial>(Serial);
  MqttClient::Network * mqttNetwork = new
MqttClient::NetworkClientImpl<WiFiClient>(network, *mqttSystem);
  //// Make 128 bytes send buffer
  MqttClient::Buffer *mqttSendBuffer = new MqttClient::ArrayBuffer<128>();
  //// Make 128 bytes receive buffer
  MqttClient::Buffer *mqttRecvBuffer = new MqttClient::ArrayBuffer<128>();
  //// Allow up to 2 subscriptions simultaneously
  MqttClient::MessageHandlers *mqttMessageHandlers = new
MqttClient::MessageHandlersImpl<2>();
  //// Configure client options
  MqttClient::Options mqttOptions;
  ////// Set command timeout to 10 seconds
  mqttOptions.commandTimeoutMs = 10000;
  //// Make client object
  mqtt = new MqttClient(
        mqttOptions, *mqttLogger, *mqttSystem, *mqttNetwork, *mqttSendBuffer,
        *mqttRecvBuffer, *mqttMessageHandlers
  );
}

// ============== Main loop ================================================
void loop() {
  // Check connection status
  if (!mqtt->isConnected()) {
        // Close connection if exists
        network.stop();
        // Re-establish TCP connection with MQTT broker
        LOG_PRINTFLN("Connecting");
        network.connect("test.mosquitto.org", 1883);
        if (!network.connected()) {
              LOG_PRINTFLN("Can't establish the TCP connection");
              delay(5000);
              ESP.reset();
        }
        // Start new MQTT connection
        MqttClient::ConnectResult connectResult;
        // Connect
        {
              MQTTPacket_connectData options = MQTTPacket_connectData_initializer;
              options.MQTTVersion = 4;
              options.clientID.cstring = (char*)MQTT_ID;
              options.cleansession = true;
              options.keepAliveInterval = 15; // 15 seconds
              MqttClient::Error::type rc = mqtt->connect(options, connectResult);
              if (rc != MqttClient::Error::SUCCESS) {
```

```
                        LOG_PRINTFLN("Connection error: %i", rc);
                        return;
                }
        }
        {
                // Add subscribe here if required
        }
  } else {
        {
                // Add publish here if required
        }
        // Idle for 30 seconds
        mqtt->yield(30000L);
  }
}
```

setting the network to be my phone's hotspot:

```
// Setup WiFi network
WiFi.mode(WIFI_STA);
WiFi.hostname("ESP_" MQTT_ID);
WiFi.begin("louloulacastagne", "12071998");
LOG_PRINTFLN("\n");
LOG_PRINTFLN("Connecting to WiFi");
```

specifying my laptop's ip so the card can connect to the broker:

```
// ============== Main loop ==================================
void loop() {
  // Check connection status
  if (!mqtt->isConnected()) {
    // Close connection if exists
    network.stop();
    // Re-establish TCP connection with MQTT broker
    LOG_PRINTFLN("Connecting");
    network.connect("192.168.43.188", 1883);
    if (!network.connected()) {
      LOG_PRINTFLN("Can't establish the TCP connection");
      delay(5000);
      ESP.reset();
    }
```

I needed to download the profile for the card (esp8266), compile and flash the code to the card, specify the output frequency to the one specified in the code (115200 baud), and the program runs!

```
Connecting to WiFi
.
.
.
.
.
.
Connected to WiFi
IP: 192.168.43.121
Connecting
MQTT - Connect, clean-session: 1, ts: 3129
MQTT - Wait for message, type: 2, tm: 9999 ms
MQTT - Process message, type: 2
MQTT - Connect ack received
MQTT - Connect ack, code: 0
MQTT - Keepalive interval: 12 sec
```

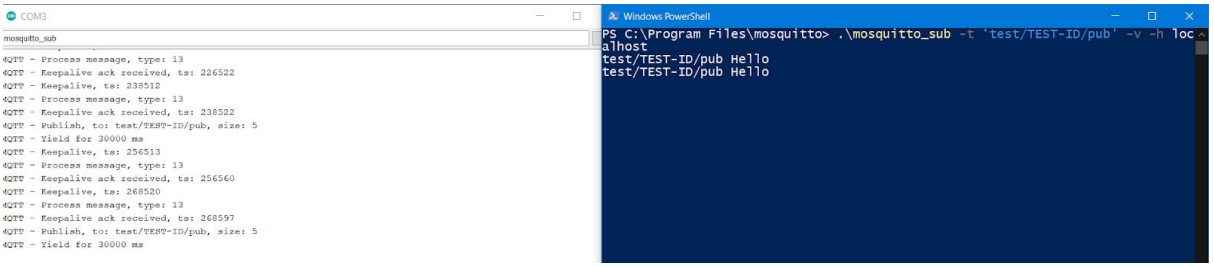
adding publish and subscribe functions to the code :

```
// Subscribe
{
  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;
  }
}
else {
// Publish
{
  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);
}
```

We also add these declarationsneeded for these features

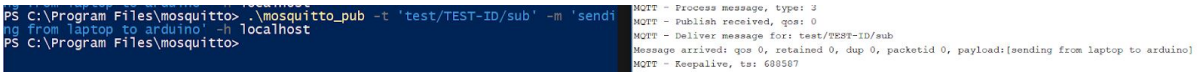
```
// ============== Subscription callback ========================================
void processMessage(MqttClient::MessageData& md) {
  const MqttClient::Message& msg = md.message;
  char payload[msg.payloadLen + 1];
  memcpy(payload, msg.payload, msg.payloadLen);
  payload[msg.payloadLen] = '\0';
  LOG_PRINTFLN(
    "Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
    msg.qos, msg.retained, msg.dup, msg.id, payload
  );
}

#define MQTT_ID                         "TEST-ID"
const char* MQTT_TOPIC_SUB = "test/" MQTT_ID "/sub";
const char* MQTT_TOPIC_PUB = "test/" MQTT_ID "/pub";
```

After flashing the new code to the arduino, it publishes on the topic “test/TEST-ID/pub every few seconds. We listen on that topic on the laptop to see if it properly accesses the broker and publishes:



everything works fine!

To test the other way, we publish a message from the laptop on the topic “test/TEST-ID/sub”:



also works! message received by the arduino

Sadly we did not have enough time to develop the light management app, and even though you took some hardware home to try and finish it in our free time, we couldn’t meet up because of quarantine and had to focus on the rest of the labs.

TP2 : MangOH Presentation

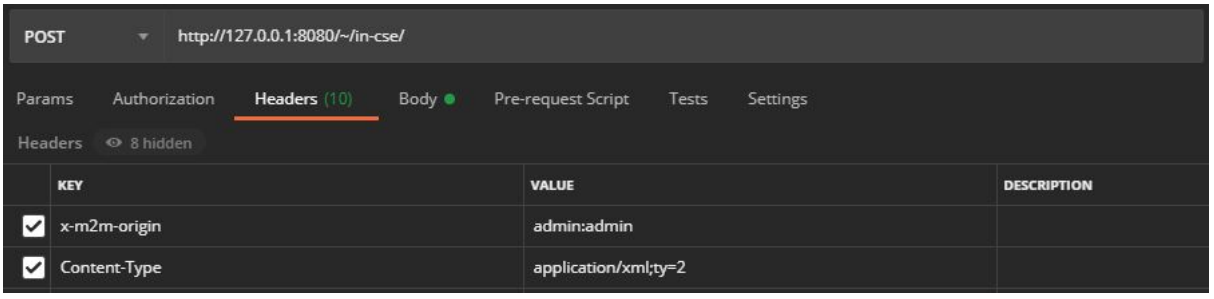
In this lab, we used used Octave, an all-in-one platform developped by Sierra wireless to exract, orchestrate and act on our mangOH Yellow board. The lab was very directed, so we didn’t have much to do on our own, except very basic code to implement actions. This lab was more focused on showing off the features offered by Octave and the board, enabling both edge and cloud capabilities. We used sensors to capture data, used octave’s built-in data filters to only capture what we needed (for example only updating the light sensor’s value when we measured a high variation). We also implemented edge and cloud actions to react to the changes measured by our board. The

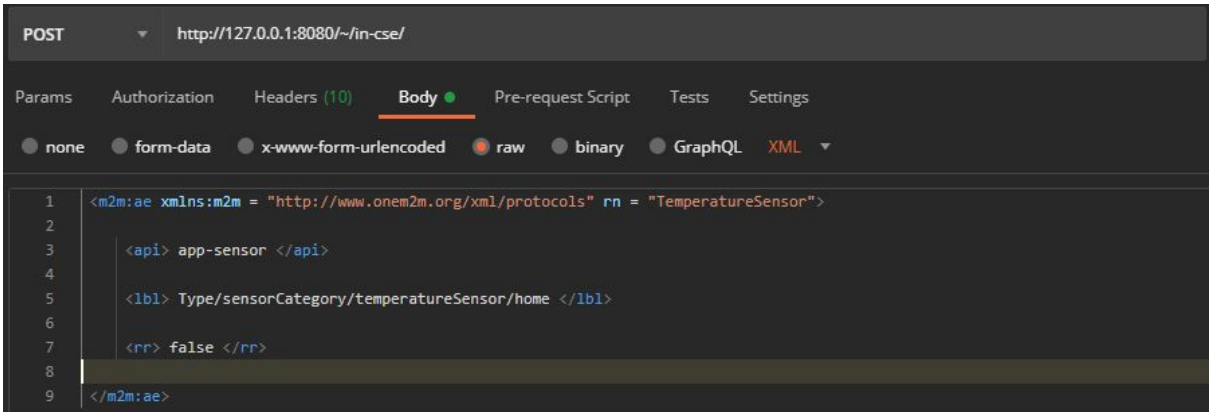
combination of Octave and the mangOH board was impressive, as the board offered a wide array of sensors and features making it a great standalone piece of hardware with its own networking chip, and octave gave us the tools to control every part of the process, from the board to the edge to the cloud.

TP3 : Middleware for the IoT

1) **Create 3 AE on the MN with the following names:**
    a) SmartMeter
    b) LuminositySensor
    c) TemperatureSensor

To create an AE, the following request must be sent :





As the server returns “Status” value “201 created”, AE are created.
Let’s check this on the web interface :

http://localhost:8080/~/in-cse

- in-name
  - acp_admin
  - acpae-932262451
  - acpae-685609287
  - acpae-729516960
  - SmartMeter
  - LuminositySensor
  - TemperatureSensor
  - mn-cse

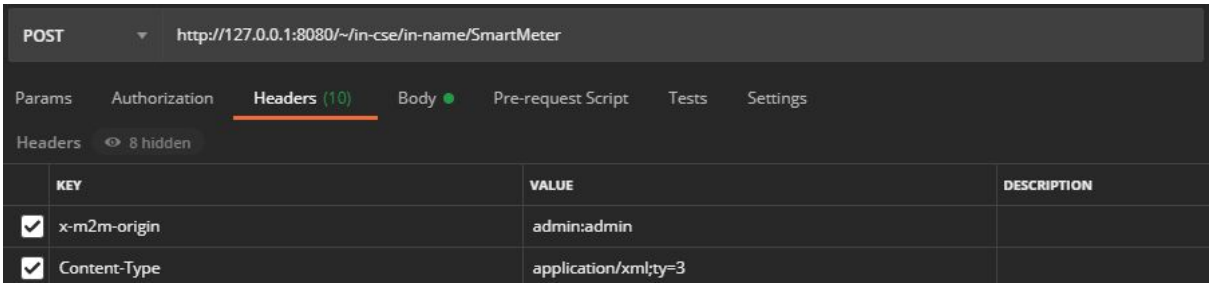
It works !

2) Create the content instances as indicated below:

- The DESCRIPTOR container should have 1 content instance containing the description of the sensor. Use the following oBIX model for the content representation

```
<obj>
    <str name="Type" val="Sensor" />
    <str name="Category" val="Light" />
    <str name="Unit" val="Lux" />
    <str name="Model" val="1142_0" />
    <str name="Location" val="Home" />
    <str name="Manufacturer" val="PHIDGETS" />
    <str name="Consumption Max" val="27 mA" />
    <str name="Voltage Min" val="4.8 V DC" />
    <str name="Voltage Max" val="5.3 V DC" />
    <str name="Operating Temperature Min" val="0 C" />
    <str name="Operating Temperature Max" val="70 C" />
</obj>
```

To create OBSERVER for each AE, the request model to send is the following one :



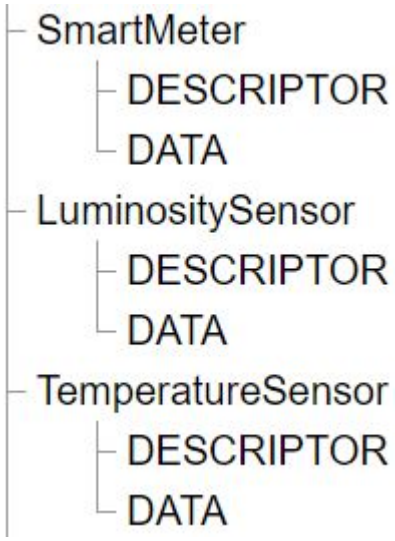
POST http://127.0.0.1:8080/~/in-cse/in-name/SmartMeter

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

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

We can check on the interface that all the containers for each AE have been created :



The body request to create the description is the following :

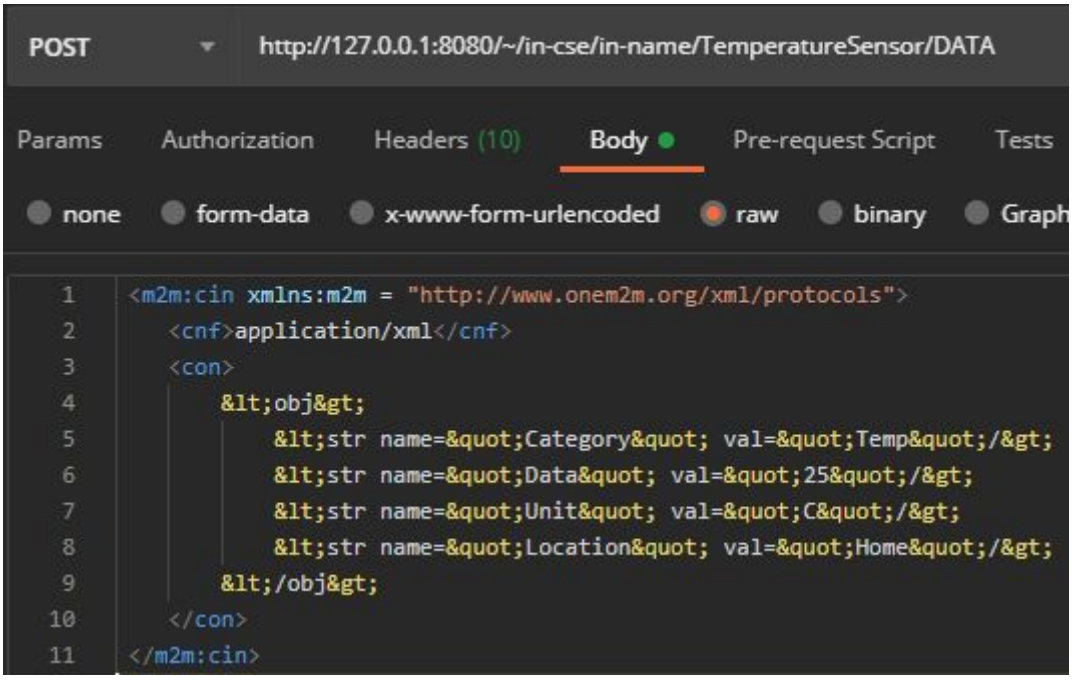
POST http://127.0.0.1:8080/~/in-cse/in-name/TemperatureSensor/DESCRIPTOR

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
<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;Sensor&quot;/&gt;
            &lt;str name=&quot;Category&quot; val=&quot;Temp&quot;/&gt;
            &lt;str name=&quot;Unit&quot; val=&quot;C&quot;/&gt;
            &lt;str name=&quot;Model&quot; val=&quot;1142_0&quot;/&gt;
            &lt;str name=&quot;Location&quot; val=&quot;Home&quot;/&gt;
            &lt;str name=&quot;Manufacturer&quot; val=&quot;PHIDGETS&quot;/&gt;
            &lt;str name=&quot;Consumption Max&quot; val=&quot;27 mA&quot;/&gt;
            &lt;str name=&quot;Voltage Min&quot; val=&quot;4.8 V DC&quot;/&gt;
            &lt;str name=&quot;Voltage Max&quot; val=&quot;5.3 V DC&quot;/&gt;
            &lt;str name=&quot;Operating Temperature Min&quot; val=&quot;0 C&quot;/&gt;
            &lt;str name=&quot;Operating Temperature Max&quot; val=&quot;70 C&quot;/&gt;
        &lt;/obj&gt;
    </con>
</m2m:cin>
```

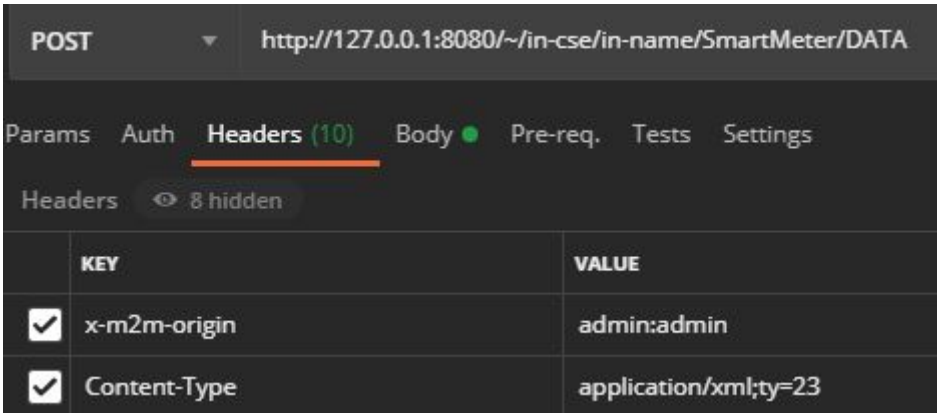
The body request to create a data is the following :



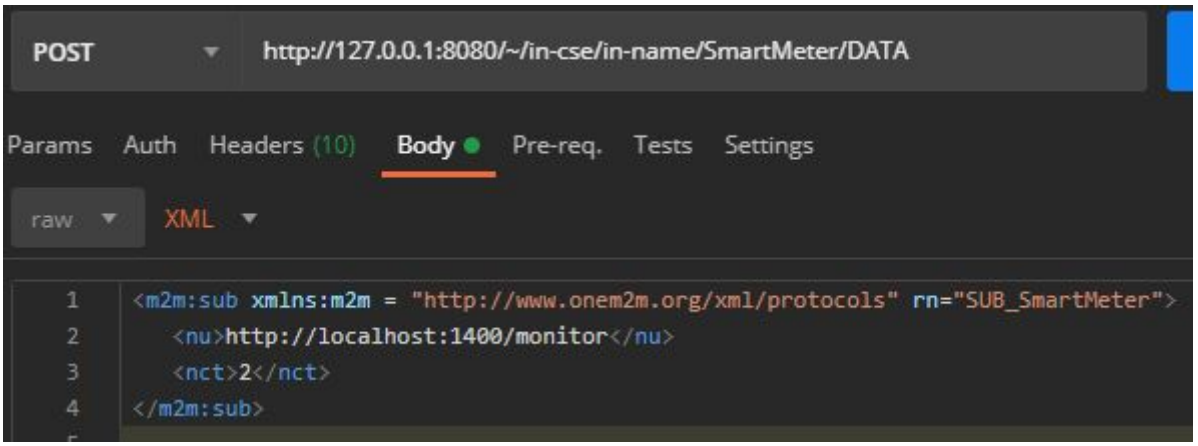
3) Start the monitoring application

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

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>
```

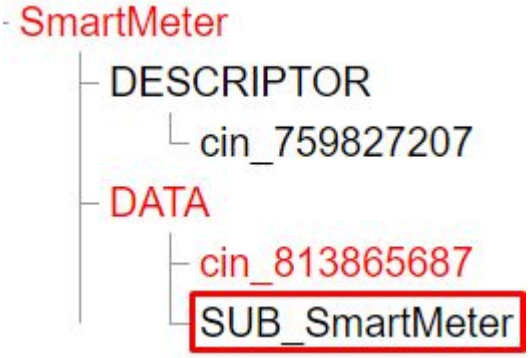
The sub request is header is like :



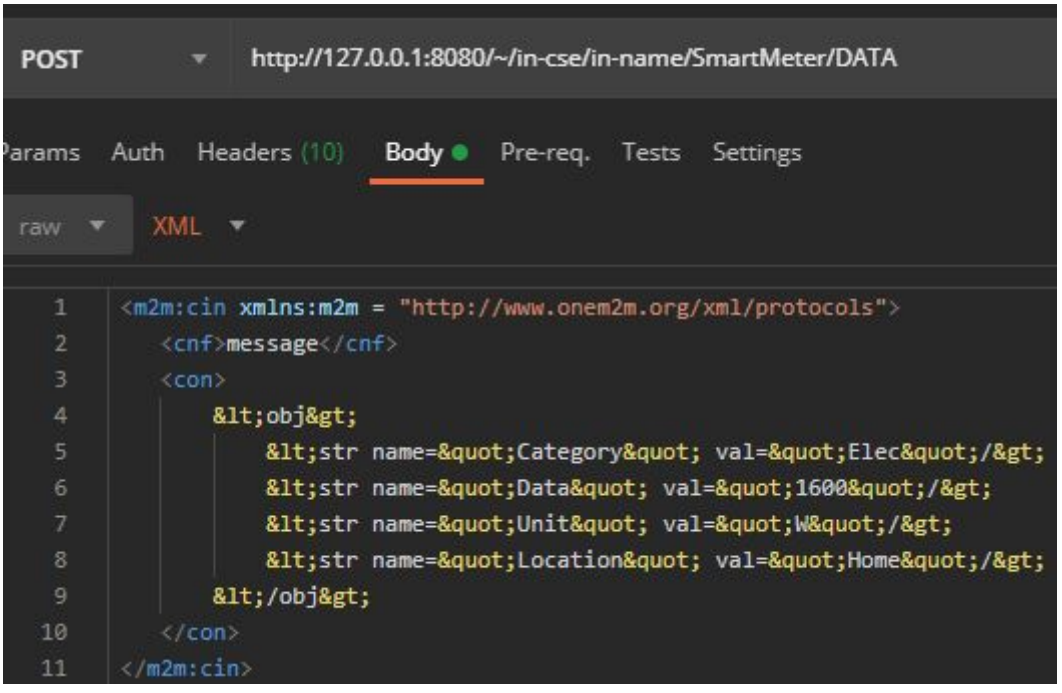
The sub request body is the following one :



We can see on the interface that the sub resource has been created :



To test the monitor subscription we send this data request :



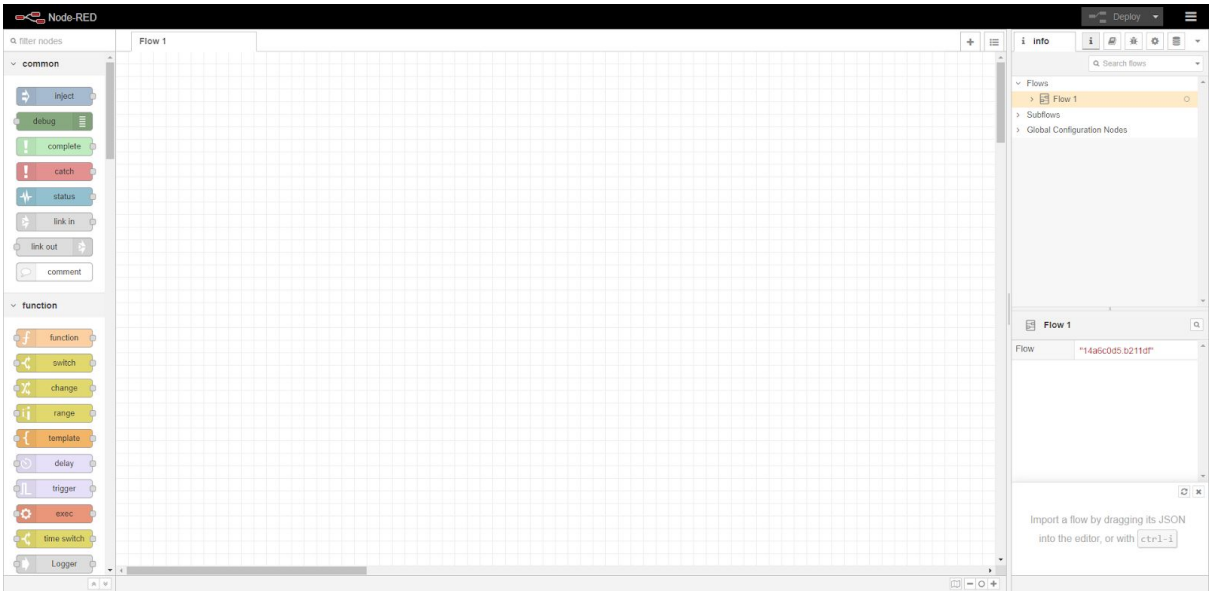
We can on the monitor the following notification :

```
Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <nev>
      <rep rn="cin_942629114">
         <ty>4</ty>
         <ri>/in-cse/cin-942629114</ri>
         <pi>/in-cse/cnt-76143935</pi>
         <ct>20201106T132746</ct>
         <lt>20201106T132746</lt>
         <st>0</st>
         <cnf>message</cnf>
         <cs>207</cs>
         <con>
        &lt;obj>
            &lt;str name=&quot;Category&quot; val=&quot;Elec&quot;/>
            &lt;str name=&quot;Data&quot; val=&quot;1600&quot;/>
            &lt;str name=&quot;Unit&quot; val=&quot;W&quot;/>
            &lt;str name=&quot;Location&quot; val=&quot;Home&quot;/>
        &lt;/obj>
    </con>
      </rep>
      <rss>1</rss>
   </nev>
   <sud>false</sud>
   <sur>/in-cse/in-name/SmartMeter/DATA/SUB_SmartMeter</sur>
</m2m:sgn>
```

The value is 1600, indicating it’s working smoothly.

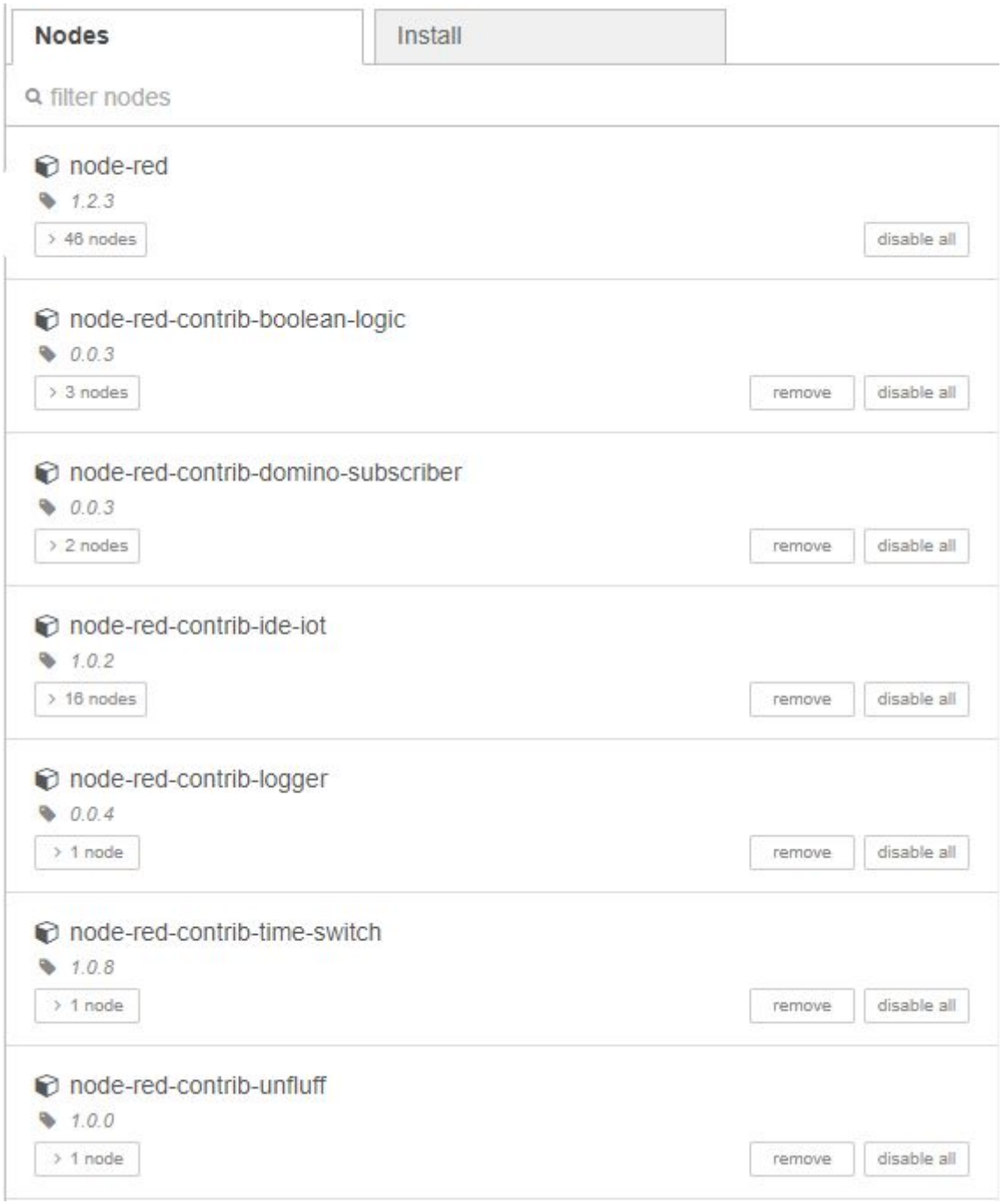
TP4 : Fast application prototyping for IoT

Node-red has been completely set up :



Node-red interface is accessible from the URL http://localhost:8080

We also added all the modules needed :



node-red-dashboard

2.24.0

> 21 nodes

remove disable all

node-red-node-email

1.8.2

> 3 nodes

remove disable all

node-red-node-rbe

0.2.9

> 1 node

disable all

node-red-node-tail

0.1.1

> 1 node

disable all

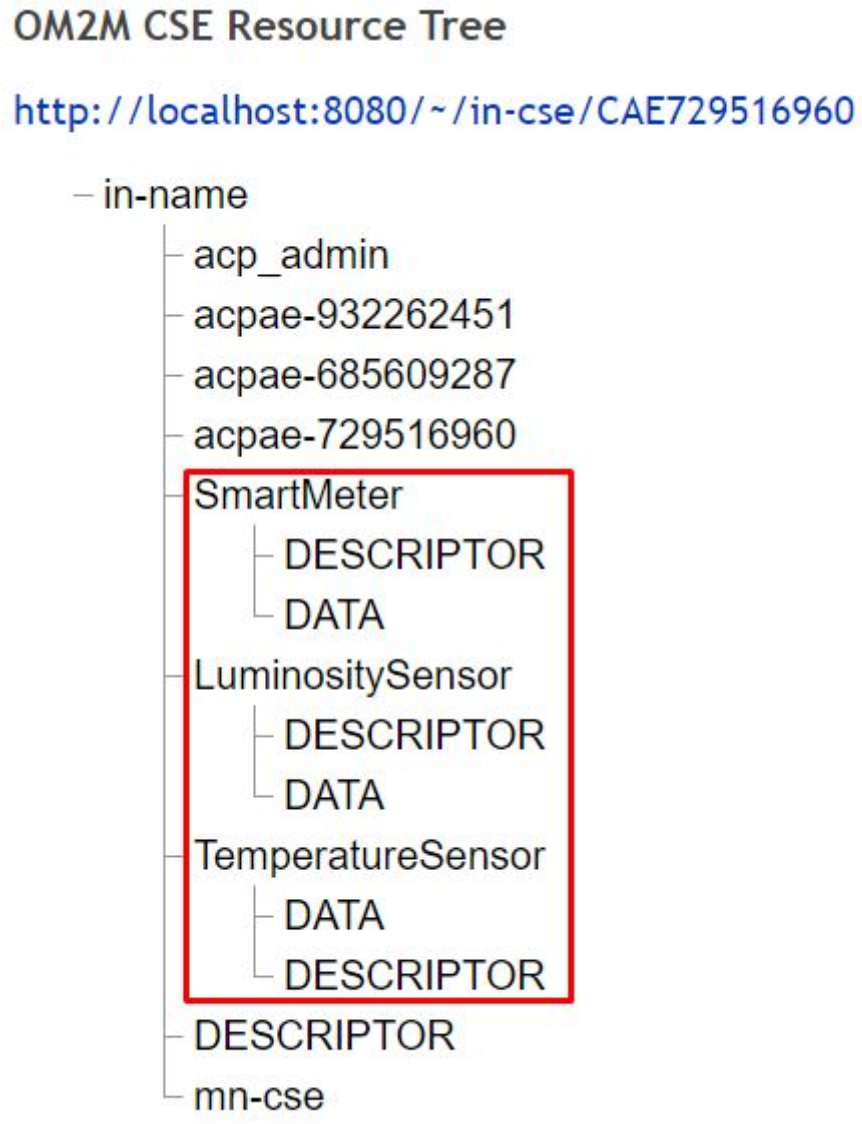
node-red-node-twitter

1.1.6

> 3 nodes

remove disable all

For this exercise we well use all the components from TP3 :



We have all the 3 sensors with “descriptor” and “data” for each one.

While all of the setup phases went well, we hit a brick on the road during the “Application” part of the lab. We were already a little behind since these labs took place just after quarantine hit and we had some trouble getting all the work done. Since we only had a week to finish everything up and write this report, we were not able to finish this lab and develop the the MQTT nodes to get sensor values and use activators. Even though we didn’t get to develop a high level application with node RED, the skills we aquired through labs one to three cover a lot of ground and should be sufficient for the next projects we tackle.