INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Department of Electrical and Computer Engineering
9th semester
T.U. Communication
Protocols for connected objects

INNOVATIVE SMART SYSTEMS

COMPTE-RENDU TP MACHINE-TO-MACHINE

GEI ISS 2020/2021

Marc Kassé

TP 1 : Middleware For the IoT

Le protocole MQTT utilise une architecture « publish/subscribe » en contraste avec le protocole HTTP et son architecture « request/response ».
Le point central de la communication est le broker MQTT en charge de relayer les messages des émetteurs vers les clients. Chaque client s'abonne via un message vers le broker : le « topic » (sorte d'information de routage pour le broker) qui permettra au broker de réémettre les messages reçus des producteurs de données vers les clients. Les clients et les producteurs n'ont ainsi pas à se connaître, ne communiquant qu'au travers des topics. Cette architecture permet des solutions multi-échelles.

MQTT est un basé sur le protocole TCP/IP permet de se connecter au broker en utilisant un port standard (1883) ou un port choisi de manière libre. Il est assez peu énergivore en bande passante.

Le MQTT a été créé par le Dr Andy Stanford-Clark d'IBM et Arlen Nipper d'Arcom – aujourd'hui Eurotech – en 1999.
Bien que le MQTT soit toujours étroitement associé à IBM, il s'agit désormais d'un protocole ouvert qui est supervisé par l'Organisation pour l'avancement des normes d'information structurées (OASIS).
Bien que son nom l'indique, MQTT ne fait pas partie de la série MQSeries originale d'IBM ; cependant, à partir de la version 7.1, il est disponible dans WebSphere MQ. MQTT était auparavant connu sous les noms de protocole SCADA, MQ Integrator SCADA Device Protocol (MQIsdp) et WebSphere MQTT (WMQTT), bien que toutes ces variantes soient tombées en désuétude.
Il existe de nombreuses bibliothèques qui implémentent MQTT.

Les données IoT échangées peuvent s'avérer très critiques, c'est pourquoi il est aussi possible de sécuriser les échanges à plusieurs niveaux :

- Transport en SSL/TLS
- Authentification par certificats SSL/TLS
- Authentification par login/mot de passe

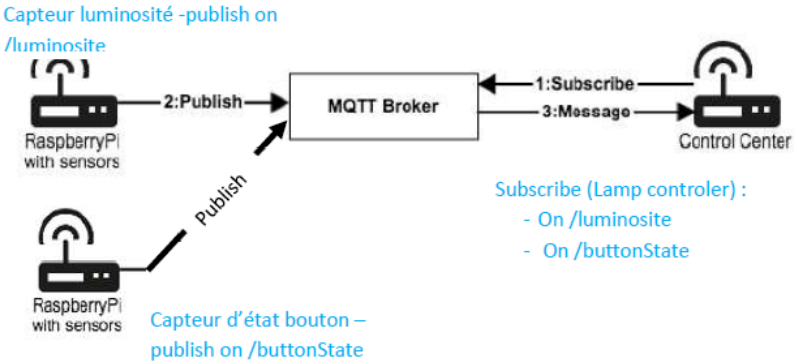
Création d'un équipement IoT avec le module NodeMCU ESP8266 qui utilise MQTT

Les caractéristiques principales de ce module sont :

- Operating Voltage : 3.3V
- Input Voltage : 7-12V
- Digital I/O Pins (DIO) : 16
- Analog Input Pins (ADC) : 1
- Possède des interfaces UARTs, SPIs, et I2Cs
- Clock Speed : 80 MHz
- USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
- Facilement programmable sur Arduino

Nous analysons le code Arduino disponible après avoir inclus la librairie ArduinoMQTT by Oleg K.
La connexion wifi est configurée dans un premier temps puis le client Mqtt client est défini. Si la connexion TCP n'est pas établie entre le broker MQTT et le client (carte Arduino), on effectue une nouvelle tentative.
Une fonction callback Subcription callback est appelée dès qu'un message est reçu. Puis le message est publié sur un topic donné.

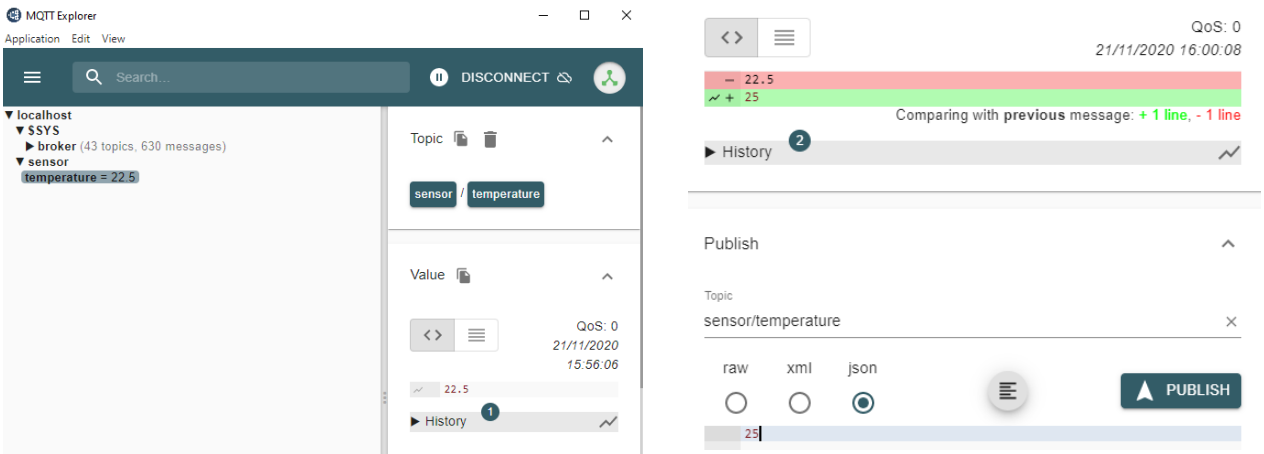
Nous n'avons pas pu avancer plus loin durant la séance suite à un problème de Wifi puis de temps. Néanmoins, nous avons pu nous familiariser avec les commandes de mosquitto et découvrir le principe de base du protocole MQTT. De plus, nous avons réfléchi à l'architecture qu'il aurait fallu implémenter.



Architecture MQTT

```
C:\mosquitto>mosquitto_pub -h localhost -t sensor/temperature -m 22.5
C:\mosquitto>mosquitto_sub -h localhost -t "sensor/temperature"
25
```

Exemple mosquitto : publication dune température sur le topic sensor/température et suscription d'un autre client MQTT au topic « sensor/temperature »



Fenêtres Mosquitto

Pour rappel, l'objectif est de développer une application qui éteint une lampe lorsque que l'intensité lumineuse passe en dessous d'un certain seuil, ou lorsque que l'utilisateur appuie surun bouton.
Il fallait imaginer que les 3 dispositifs (capteur de luminosité, lampe, bouton poussoir) étaient éparés.
La solution est de créer deux topics :

- Le premier sert à communiquer l'intensité lumineuse.
- Le deuxième permet de communiquer l'état du bouton (on/off)

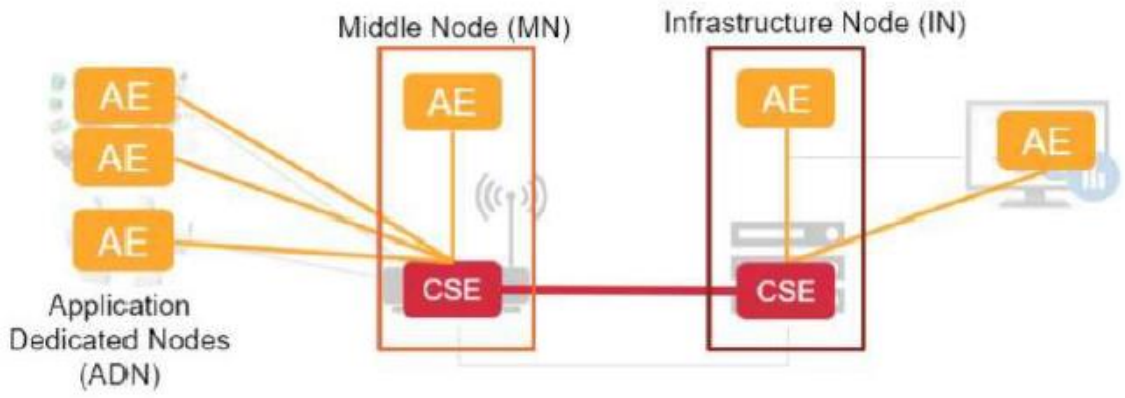
Le client MQTT responsable de l'allumage de la lampe s'abonne aux deux topics, et invoque l'un des deux call back en fonction du topic :

- Le premier allume la lampe si jamais l'intensité lumineuse passe en dessous d'un certain seuil, lors de la réception d'un message sur le topic luminosité.
- Le deuxième allume ou éteint la lampe

TP2 : Middleware for the IoT

Interact with the oneM2M RESTful architecture using Eclipse OM2M

OneM2M est un standard international destiné à la communication “machine to machine” et à l’IoT. L’idée est de définir une architecture globale, indépendante des protocoles ou appareils utilisés afin de faciliter leur intégration dans tous les projets IoT.



Dans le standard OneM2M, il existe deux types d’entités :

- Les AE (applications entities) sont les applications utilisées par le système pour assurer son fonctionnement. Elles sont destinées aussi bien aux objets qu’aux utilisateurs.
- La couche CSE (common service entities) fournit les différents services qui vont permettre d'interagir avec les applications.

Le lien vers les requêtes que j’ai créées se trouvent au lien suivant :
https://www.getpostman.com/collections/4c9170cf5e05ab11a467

La ressource ACP attribue les accès à des ressources oneM2M grâce à la valeur qu’on attribue à AccessControlOperation.

Attribute	Value
rn	acp_sensor
ty	1
ri	/in-cse/acp-925560370
pi	/in-cse
ct	20201106T004236
lt	20201106T004236
pv	AccessControlOriginator: SensorRC — AccessControlOperation: 3 AccessControlOriginator: admin:admin — AccessControlOperation: 63
pvs	AccessControlOriginator: admin:admin — AccessControlOperation: 63

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

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary XML (application/xml)

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

Les accès sont attribués à l’application MY_SENSOR, comme ci-dessous.

Attribute	Value
rn	MY_SENSOR
ty	2
ri	/in-cse/CAE555241203
pi	/in-cse
ct	20201028T175515
lt	20201106T005928
lbl	• Category/temperature • Location/home • Type/sensor
acpi	AccessControlPolicyIDs /in-cse/acp-165530295 /in-cse/acp-925560370
et	20211028T175515
api	app-sensor
aei	CAE555241203
rr	false

PUT http://127.0.0.1:8080/~/in-cse/in-name/MY_SENSOR

```
{
    "m2m:ae": {
        "acpi": [
            "/in-cse/acp-165530295",
            "/in-cse/acp-925560370"
        ]
    }
}
```

Trois applications sont créées avec leurs containers respectifs. La sémantique impose d'avoir un Descriptor et un Data :

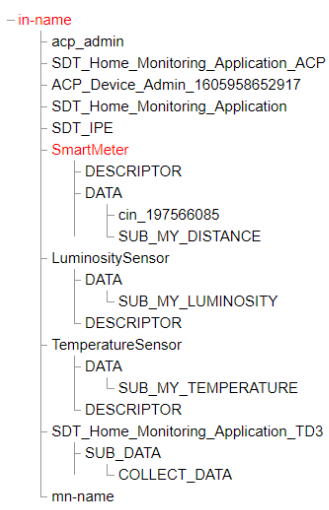
- SmartMeter
  - DATA
  - DESCRIPTOR
- LuminositySensor
  - DATA
  - DESCRIPTOR
- TemperatureSensor
  - DATA
  - DESCRIPTOR

Chaque nouvelle donnée est une instance.

Attribute	Value
rn	cin_251512809
ty	4
ri	/in-cse/cin-251512809
pi	/in-cse/cnt-162049591
ct	20201108T001942
lt	20201108T001942
st	0
cnf	message
cs	2
con	

- LuminositySensor
  - DATA
    - cin_681368958

Une application est créée en mettant en place une architecture publish/subscribe :



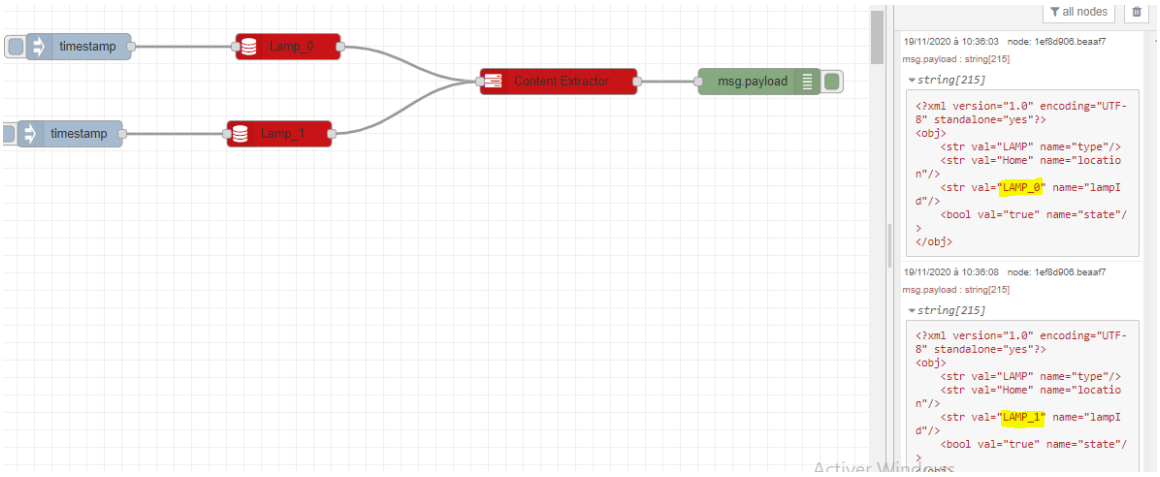
Attribute	Value
rn	SDT_Home_Monitoring_Application_TD3
ty	2
ri	/in-cse/CAE933097326
pi	/in-cse
ct	20201122T092742
lt	20201122T092742
lbl	• ResourceID/SDT_Home_Monitoring_Application • ResourceType/Application
acpi	AccessControlPolicyIDs /in-cse/acp-189558481 /in-cse/acp-573945650
et	20211122T092742
apn	Home Monitoring Application TD3
api	SDT_Home_Monitoring_Application_TD3
aei	CAE933097326
poa	Point Of Access http://localhost:1400/monitor
rr	true

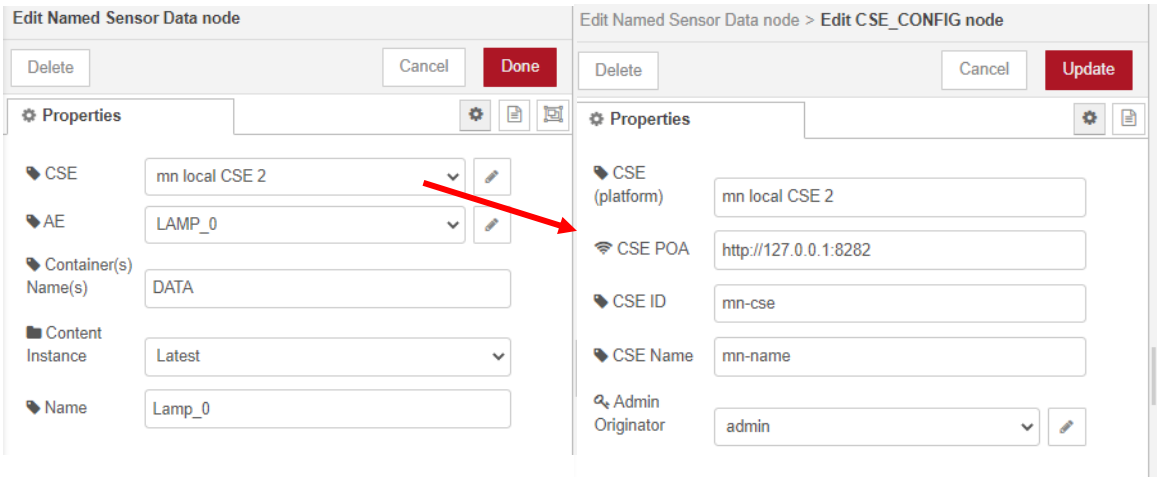
On définit un poa (point of access).

TP 4 - Fast application prototyping for IoT

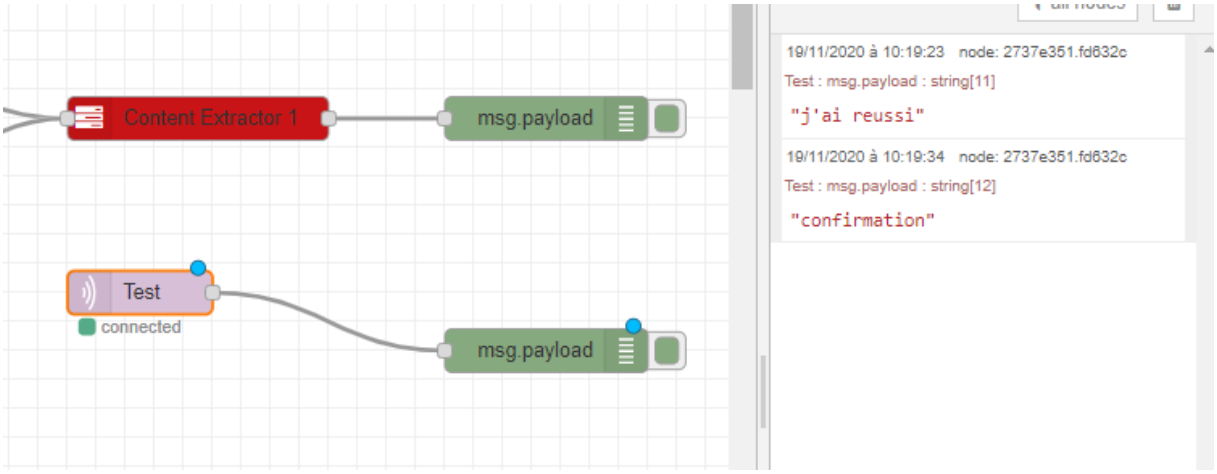
Utilisation de Named Sensor Data et Content Extractor

On remarque que, suite au déclenchement de « timestamp », on extrait le contenu de la dernière constant instance (CI) créé. Dans le contenu, il n'y a pas le numéro ri de la dernière CI. Par contre, on s'aperçoit bien de la distinction entre la lampe 0 et 1.



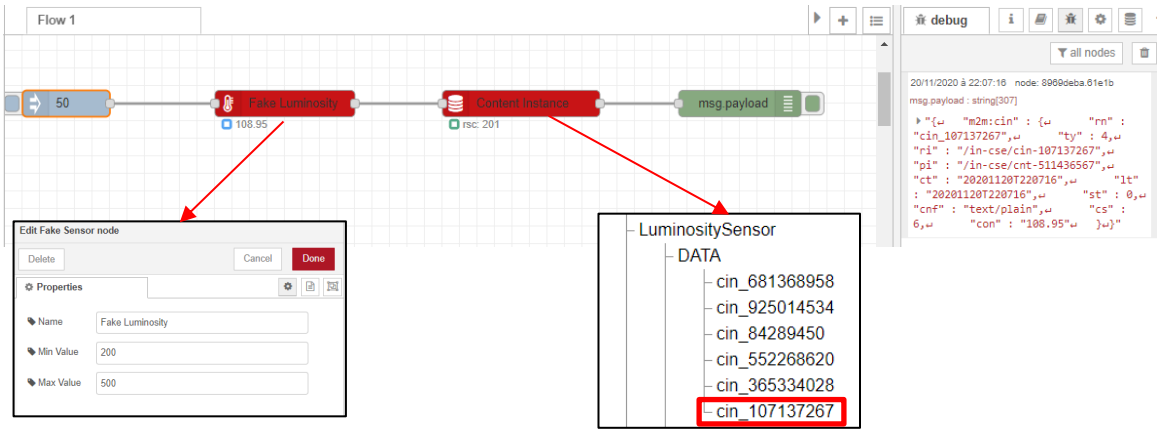


Utilisation d'un MQTT subscribe



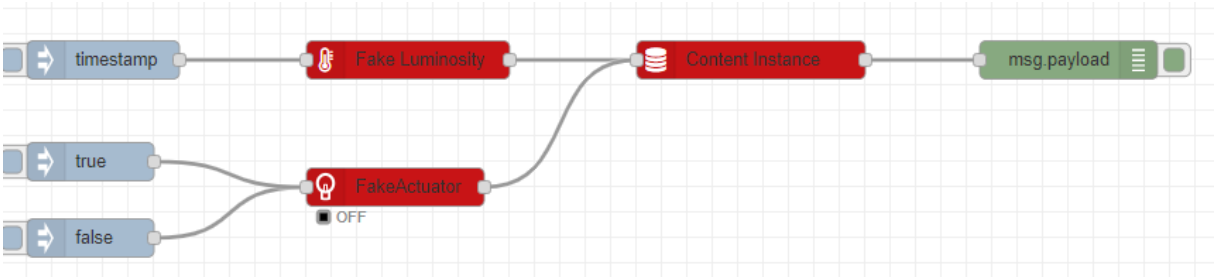
Utilisation fake sensor

Le fake sensor ne joue pas un rôle de filtre, en lui injectant une valeur en dehors de sa zone définie de détection, car il créé un content instance malgré la valeur de 50.



Fake actuator

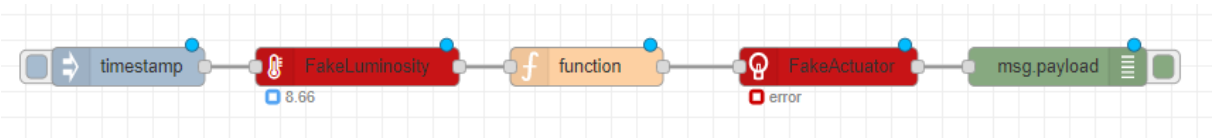
Je peux créer un actionneur qui s'allume en fonctionne de la valeur qui lui est envoyée (true or false). Néanmoins, même à l'état false, une content instance se créé alors que ça ne devrait pas être le cas.



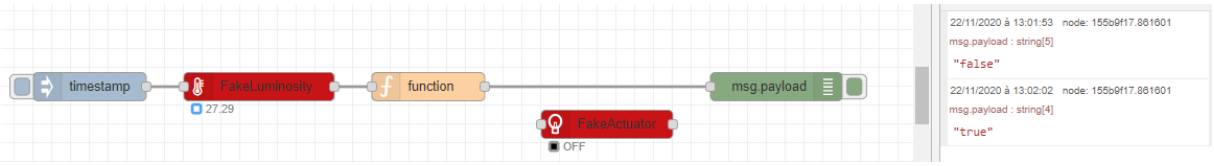
Perform a simple test between luminosity value and a threshold (choose an arbitrary value) using SimpleCondition node.

Vu que mon application ne fonctionne pas très bien, j'ai réalisé l'exercice avec un Fake Actuator.

Le faux actionneur FakeActuator renvoie une erreur :



La condition pourtant marche bien :



Suite à une mauvaise manipulation, j'ai perdu l'accès à node-red et il m'a été impossible de poursuive la suite du TP.

FIN