

Rapport de Projet de fin d'études :

Etude comparative et mise en place des protocoles applicatifs utilisés dans l'IoT (MQTT, CoAP, AMQP, HTTP, WebSocket, ...)



Filière : Administration des Systèmes et Réseaux

Effectué par :

Meryam RHADI

Oumaima AYYADI

Encadré par : Professeur Omar Moussaoui

Remerciements

On se permet d'adresser nos remerciements à tous ceux qui ont contribué à la réalisation de ce modeste travail.

Tout d'abord on tient à remercier Mr Omar MOUSSAOUI pour ses directives et ses conseils précieux

On vous présente Monsieur, notre reconnaissance et notre joie de nous avoir encadré sur ce projet

Nous remercions également tous ceux qui nous ont aidés de près ou de loin à la réalisation de ce projet.

Table de matière :

Partie théorique :	5
CHAPITRE I : L'internet des Objets (IOT)	5
1. Introduction générale -----	5
2. Définition de l'internet des objets :-----	6
3. Un exemple de IOT :-----	6
CHAPITRE II : Les protocoles applicatifs utilisés dans IOT	7
1. Protocole de Messagerie-----	7
a. MQTT (Message Queuing Telemetry Transport)-----	7
b. AMQP (Advanced Message Queuing Protocol) -----	10
2. Protocole de réseaux :-----	11
a. Websocket-----	11
3. Protocole de transfert web :-----	12
a. CoAP (Constrained Application Protocol) -----	12
b. Interfonctionnement entre CoAP et http -----	14
Partie pratique :	16
CHAPITRE I : Applications utilisées dans la mise en place de IOT	16
1. Node-Red :-----	16
a. Mettre à jour votre système :-----	17
b. Installer Node.js et npm :-----	17
c. Installer Node-RED : -----	18
d. Tester votre installation :-----	18
2. MQTT Explorer : -----	20
3. Contiki : -----	20
a. Installation de contiki : -----	20
CHAPITRE II : Mise en place des protocoles IOT	23
1. MQTT :-----	23
a. Installation de Mosca à l'aide de l'administrateur Node-Red :-----	23
b. Configuration du nœud Mosca :-----	25
c. Exemple 1 :-----	25
d. Exemple 2 : utilisé MQTT avec Dashboard :-----	31
2. COAP ET HTTP :-----	41
Chapitre III : faire une étude comparative	52
Conclusion	54
Bibliographie	55

Table de figure :

<i>Figure 1 : Illustration du fonctionnement du MQTT</i>	8
<i>Figure 2 : Echange des messages en MQTT</i>	9
<i>Figure 3 : Modèle AMQP</i>	10
<i>Figure 4 : La communication entre client et serveur en utilisant HTTP</i>	11
<i>Figure 5 : Comparaison entre l' architecture de COAP et HTTP</i>	12
<i>Figure 6 : Architecture de COAP</i>	13
<i>Figure 7 : Messages CON et ACK</i>	14
<i>Figure 8 : Interfonctionnement entre CAOP et HTTP</i>	15
<i>Figure 9 : Logo de node red</i>	16
<i>Figure 10 : logo de MQTT explorer</i>	20

Partie théorique :

CHAPITRE I : L'internet des Objets (IOT)

1. Introduction générale

L'internet des objets ou **IOT** (pour Internet **O**f Things) est d'abord un concept, et non pas une technologie ou un appareil spécifique. L'internet des Objet, c'est la volonté d'étendre le réseau internet, et donc les échanges des données aux objets du monde physique. Cependant, ces objets connectés à internet peuvent prendre n'importe quelle forme dans le quotidien. On a, par exemple, la voiture connectée, la montre connectée, les lunettes connectées, etc.

Comme l'IOT est en pleine croissance, la liste des objets connectés est donc très étendue et ne cesse de s'agrandir.

Un objet connecté a pour particularité de ne pas fonctionner de manière autonome, mais a la capacité de communiquer et de transmettre des informations à d'autres objets connectés. On parle également des objets connectés comme étant des objets intelligents car il s'agit de communication de machine à machine sans avoir besoin d'être humain comme intermédiaire, et c'est là le plus grand changement puisque jusqu'à présent, les ordinateurs, et donc internet avaient besoin de l'homme pour l'alimenter en données.

Aujourd'hui, l'Internet des objets constitue une passerelle entre le monde physique et le monde virtuel.

2. Définition de l'Internet des objets :

L'Internet des Objets est un réseau de réseaux qui permet, via des systèmes d'identification électronique normalisés unifiés et des dispositifs mobiles sans fil, d'identifier directement et sans ambiguïté des entités numériques et des objets physiques, et ainsi pouvoir récupérer, stocker, transférer et traiter, sans discontinuité entre les mondes physiques et virtuels, les données s'y rattachant.

3. Un exemple de IOT :

Vous rentrez chez vous après le travail, vous êtes stressé, votre rythme cardiaque sera donc plus élevé, votre montre connectée va transmettre cette information aux différents objets connectés, votre radio connectée décidera alors de mettre de la musique douce pour vous détendre, votre téléphone pourra basculer en mode « ne pas déranger ».

Ensuite, la température de votre appartement va aussi s'adapter avec les informations sur le climat extérieur et l'éclairage intérieur de votre appartement s'adaptera en fonction de la luminosité extérieure.

La finalité des objets connectés est de s'adapter non seulement à vos besoins mais aussi à votre environnement.

CHAPITRE II : Les protocoles applicatifs utilisés dans IOT

1. Protocole de Messagerie

Les protocoles de messagerie s'appuient sur un mécanisme de publication et d'abonnement dans lesquels les transferts de données se font de manière asynchrone. On peut trouver dans les protocoles de messagerie : MQTT, AMPQ, etc.

a. MQTT (Message Queuing Telemetry Transport)

C'est un protocole de messagerie de publication et d'abonnement (publish/subscribe) basé sur le protocole TCP/IP, le modèle publish/subscribe (pub/sub) est une alternative au modèle "client/serveur" traditionnel.

Un client, appelé publisher, établit dans un premier temps une connexion de type 'publish' avec le serveur MQTT, appelé broker. Il est reconnu à la fois par le publisher et le subscriber. Son rôle est de filtrer tous les messages entrants et les distribuer en conséquence.

Puis, le publisher transmet les messages au broker sur un canal spécifique, appelé topic. Par la suite, ces messages peuvent être lus par des abonnés, appelés subscribers, qui au préalable ont établi une connexion de type 'subscribe' avec le broker.

Ainsi, la transmission et la consommation des messages se font de manière asynchrone. Le fonctionnement que nous venons de détailler est illustré dans le schéma ci-dessous. Client-A, Client-B et Client-F sont des publishers alors que Client-C, Client-D et Client-E sont des subscribers.

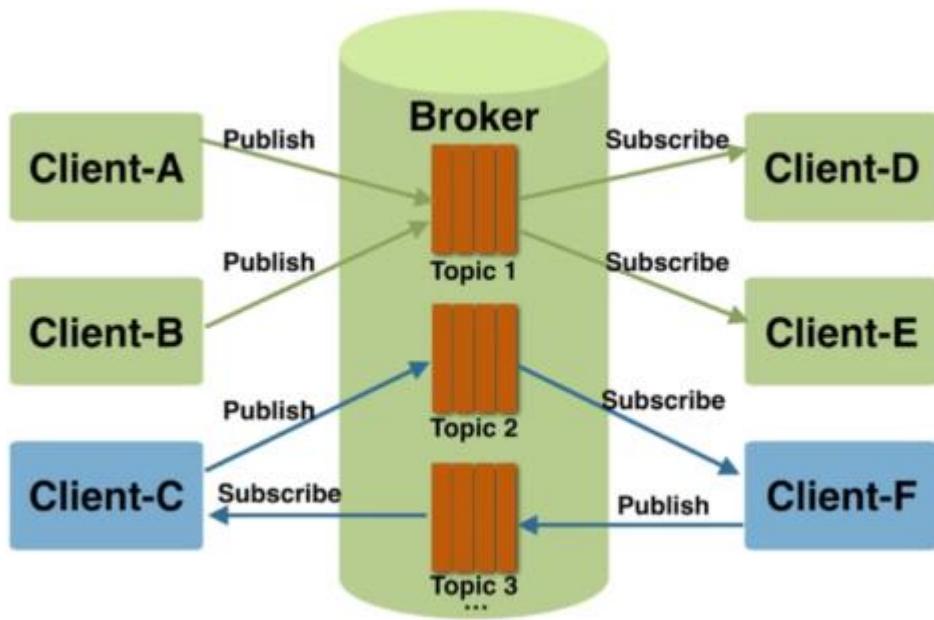


Figure 1 : Illustration du fonctionnement du MQTT

MQTT a la particularité d'être un protocole léger parce que le nombre de messages est restreint et la taille des messages est faible. En effet, chaque message se compose d'un en-tête fixe de 2 octets (spécifiant le type de message et le niveau de qualité de service employés), d'un en-tête variable facultatif, d'une payload (charge utile) limitée à 256 Mo. Il existe trois niveaux de qualité de service (QoS) qui déterminent la façon dont le protocole MQTT gère le contenu. Les clients abonnés peuvent spécifier le niveau de QoS maximal qu'ils souhaitent recevoir. Toutefois, plus le niveau de qualité est élevé et plus cela est gourmand en termes de latence et de bande passante, à cause des répétitions et des accusés de réception supplémentaires.

L'image suivante schématise ce que nous venons d'énoncer. Le publisher transmet des messages de type PUBLISH pour publier une nouvelle donnée et le subscriber utilise un message SUBSCRIBE pour recevoir des données.

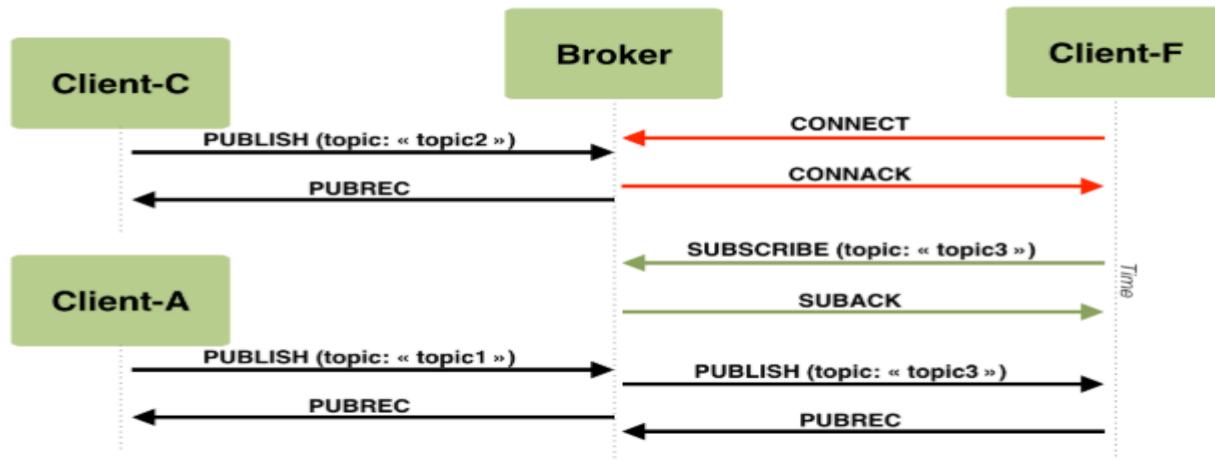


Figure 2 : Echange des messages en MQTT

En résumé, les caractéristiques du protocole MQTT en font un protocole adapté aux réseaux IoT car il répond aux besoins suivants :

- Être adapté aux réseaux à faible bande passante ;
- Idéal pour l'utilisation sur les réseaux sans fils grâce notamment à un nombre limité de messages de petite taille ;
- Faible consommation en énergie car la publication et la consommation des messages est rapide ;
- Nécessite peu de ressources de calculs et de mémoires ;
- Transmet un message à plusieurs entités en une seule connexion TCP.

b. AMQP (Advanced Message Queuing Protocol)

Le protocole AMQP (Advanced Message Queuing Protocol) est un protocole de couche application, pour un environnement middleware orienté message, qui se concentre sur la communication de processus à processus sur les réseaux IP. Le protocole est utilisé pour les systèmes de messagerie client/serveur et pour la gestion de périphériques IoT.

AMQP est efficace, portable, multicanal et sécurisé qui fonctionne bien dans les environnements multiclient, où il permet de déléguer les tâches, et autorise les serveurs à traiter plus rapidement les requêtes immédiates.

Fonctionnement du protocole AMQP :

Le fonctionnement du protocole AMQP est basé sur le même principe que celui de MQTT. Toutefois, la notion de publisher/subsciber est remplacée par celle de producer/consumer. En outre, grâce à un mécanisme interne noté « exchange », AMQP permet de router un message d'un producer vers plusieurs topics. Les critères de routage peuvent se faire de plusieurs façons : inspection du contenu, de l'en-tête, clés de routage, etc. Ainsi, un même message peut être consommé par différents consumers via plusieurs topics.

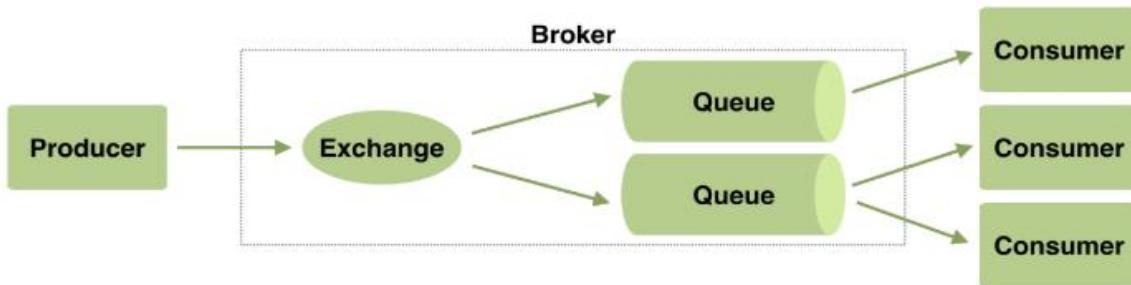


Figure 3 : Modèle AMQP

La file d'attente des messages (Queue) permet de stocker les messages jusqu'à ce qu'ils puissent être traités en profondeur via le logiciel client de restauration. Par conséquent, AMQP est plus adapté aux situations exigeant la fiabilité des scénarios de messageries plus sophistiqués. Ainsi, il est plus destiné aux objets connectés avec des contraintes de communication faibles et des exigences de sécurité importantes.

2. Protocole de réseaux :

a. Websocket

Le protocole Websocket permet l'établissement d'un canal de communication full-duplex en une seule connexion TCP entre un client et un serveur.

À l'origine, ce protocole a été mis en place pour pallier les lacunes du protocole HTTP dans les communications bidirectionnelles entre une application web et des processus serveur. En effet, la communication est asynchrone, c'est-à-dire que le serveur ne peut envoyer une réponse que si le client a, au préalable, envoyé une requête.

websocket permet donc d'établir des échanges de messages temps réel idéal pour les remontées d'alertes, des notifications.

Ce genre de communication est coûteux en termes de ressources matérielles (CPU, mémoires, source d'énergie) et de bande passante. Il n'est donc adapté qu'à des capteurs avec des ressources suffisantes. Il est principalement adapté aux situations de surveillance et d'envoi d'informations en temps réel.



Figure 4 : La communication entre client et serveur en utilisant HTTP

- >la phase de connexion appelée « Handshake », initiée par le client
- >la phase d'échange bidirectionnel de messages
- >la phase de clôture du canal initiée par l'une des deux parties

3. Protocole de transfert web :

Des protocoles basés sur les requêtes http qui utilisent le modèle de communication : demande/réponse, on peut trouver dans les protocoles de transfert web : COAP, HTTP, etc .

a. CoAP (Constrained Application Protocol)

CoAP (Constrained Application Protocol) est prévu pour devenir un protocole d'application omniprésent dans le futur Internet des objets. Il se situe au niveau applicatif de la couche OSI , et il est optimisé pour les périphériques et réseaux contraints utilisés dans les réseaux de capteurs sans fil pour former l'Internet des objets. Basé sur le style architectural REST (style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web) , il permet de manipuler au travers d'un modèle d'interaction client-serveur les ressources des objets communicants et capteurs identifiées par des URI en s'appuyant sur l'échange de requêtes-réponses et méthodes similaires au protocole HTTP. Contrairement au protocole HTTP, qui se base sur la suite TCP/IP, le protocole CoAP se base sur la suite UDP/IPv6/6LowPAN

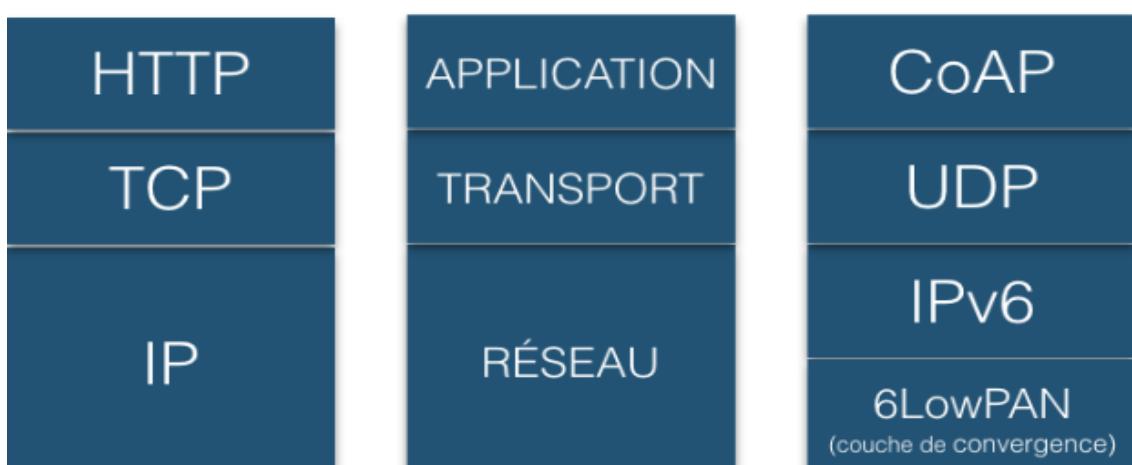


Figure 5 : Comparaison entre l'architecture de COAP et HTTP

L'architecture CoAP :

L'architecture CoAP est divisée en deux couches : une couche message qui apporte la fiabilité et le séquencement des échanges de bout en bout reposant sur UDP, et une couche «Request/Response» qui utilise des méthodes et codes réponses pour les interactions requêtes/réponses.

Il s'agit cependant bien d'un seul et même protocole qui propose ces fonctionnalités dans son entête

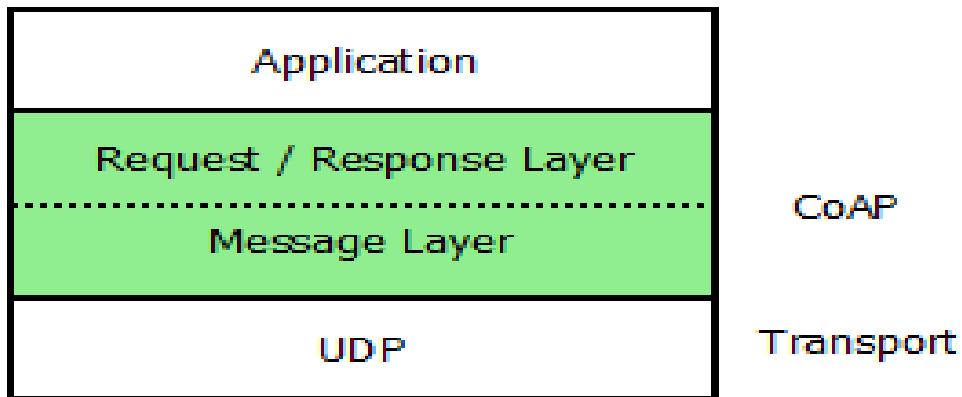


Figure 6 : Architecture de COAP

Une requête est transportée dans un message CON (Confirmable) ou NON (Nonconfirmable), et si la réponse est immédiatement disponible dans le cas d'un message CON, elle est transportée dans un message Acknowledgement (ACK). Si le message ACK est perdu, l'émetteur du message CON le retransmettra. Deux exemples sont montrés concernant une requête de base GET transportée dans un message CON avec la réponse retournée dans un message ACK. Une des réponses indique un succès (2.05) et l'autre indique un échec (4.04). Si le serveur n'est pas en mesure de répondre immédiatement à une requête (e.g., GET) transportée dans un message CON, il répond simplement via un message ACK sans contenu. Lorsque la réponse est prête, le serveur l'émet dans un nouveau message CON (qui devra être acquitté par le client via un message ACK sans contenu).

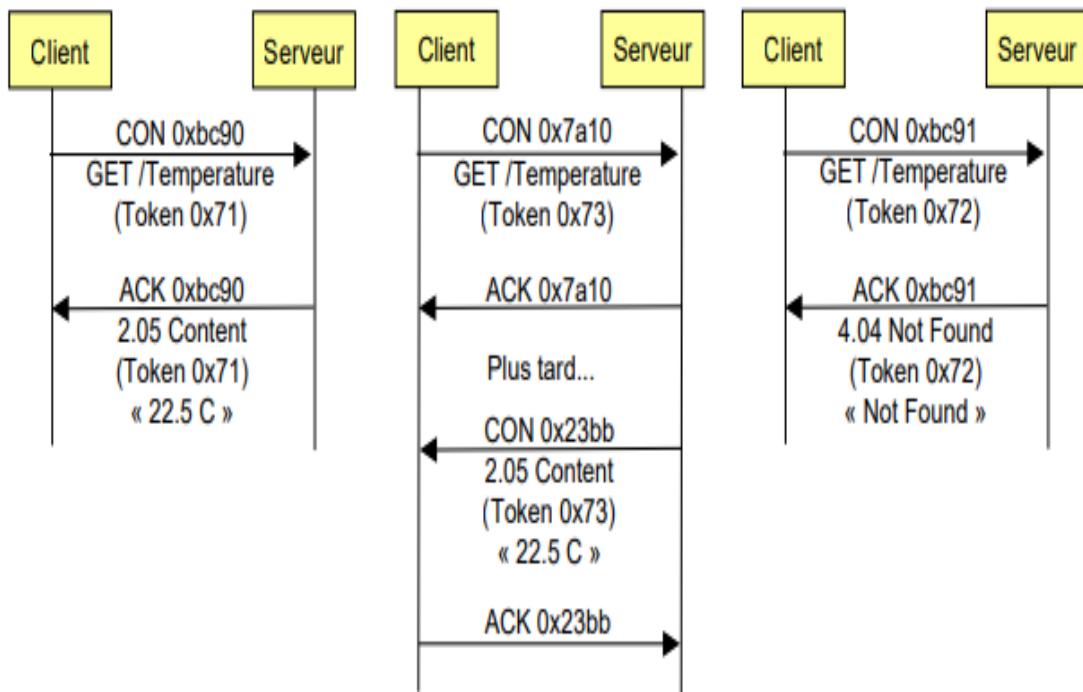


Figure 7 : Messages CON et ACK

b. Interfonctionnement entre CoAP et http

Dans beaucoup de cas, les applications M2M/IoT communiqueront via le protocole HTTP alors que les capteurs/actionneurs supporteront le protocole CoAP compte-tenu des contraintes que leurs sont liés. Dans ce contexte, il est nécessaire de disposer d'un proxy qui disposera en fonction de fonctions de cache pour stocker des informations reçues des capteurs (serveur CoAP) et les mettre à la disposition des applications (client HTTP) tant que ces données leur valeur « max-age » n'a pas expiré. La traduction de protocole HTTP/CoAP réalisée par le proxy est relativement simple car les deux protocoles utilisent les mêmes commandes.

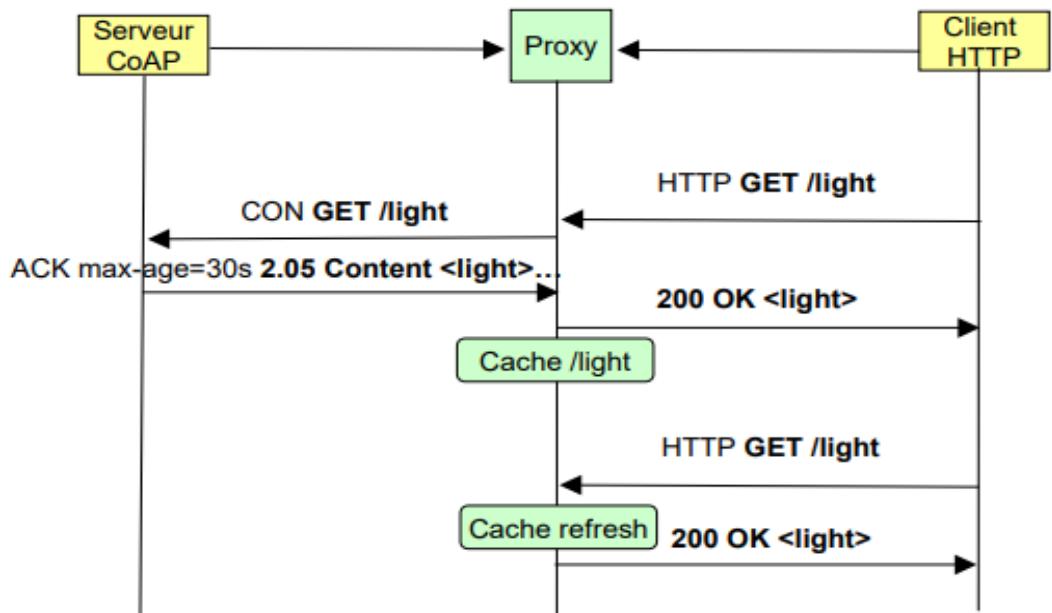


Figure 8 : Interfonctionnement entre CAOP et HTTP

Partie pratique :

CHAPITRE I : Applications utilisées dans la mise en place de IOT

1. Node-Red :

Node-RED est l'un des outils les plus populaires de la révolution IOT. C'est un outil de programmation visuelle basé sur le flux open-source pour câbler les composants dans le cadre de l'Internet des objets. Il permet aux développeurs de tous niveaux de connecter rapidement les E / S physiques, les systèmes basés sur le cloud, les bases de données et la plupart des API dans n'importe quelle combinaison que vous pouvez imaginer. Node-red peut être installé et exécuté sur de très petites machines matérielles, par exemple Raspberry PI, TinkerBoard, etc.

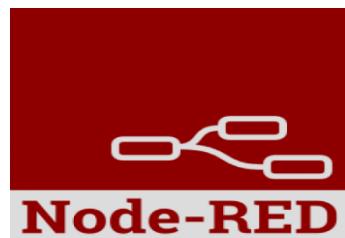


Figure 9 : Logo de node red

a. Mettez à jour votre système :

```
meryam@meryam-virtual-machine:~$ sudo apt-get update  
[sudo] password for meryam:  
Hit:1 http://ma.archive.ubuntu.com/ubuntu bionic InRelease  
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]  
Get:3 http://ma.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]  
Get:4 http://security.ubuntu.com/ubuntu bionic-security/main i386 Packages [459 kB]  
Get:5 http://ma.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]  
Get:6 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [69 1 kB]  
Get:7 http://security.ubuntu.com/ubuntu bionic-security/main Translation-en [22 1 kB]  
Fetched 1,622 kB in 1min 19s (20.6 kB/s)  
Reading package lists... Done
```

Il faut mettre à niveau des packages qui nécessitent des mises à jour dans les référentiels.

b. Installer Node.js et npm :

→ Installer node.js

```
meryam@meryam-virtual-machine:~$ sudo apt-get install nodejs
```

Il est recommandé de vérifier la version de Node.js avant de continuer. Pour vérifier la version de Node.js installée, utilisez la commande suivante :

```
meryam@meryam-virtual-machine:~$ node -v  
v8.10.0
```

→ Installer npm

```
meryam@meryam-virtual-machine:~$ sudo apt-get install npm  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  build-essential dpkg-dev fakeroot g++ g++-7 gcc gcc-7 gyp javascript-common
```

Pour vérifier la version npm installée, utilisez la commande suivante :

```
meryam@meryam-virtual-machine:~$ npm -v  
3.5.2
```

c. Installer Node-RED :

```
meryam@meryam-virtual-machine:~$ sudo npm install -g --unsafe-perm node-red node-red-admin
```

-g : Cela signifie que le package est installé globalement et qu'il sera disponible pour Node.js

–Unsafe-perm : Cela annule toutes les erreurs lors de l'installation. Fondamentalement, le changement d'ID utilisateur / groupe a supprimé lorsqu'un script de package est exécuté, les utilisateurs non root pourront installer des packages.

Node-red-admin : Cela installera le module d'administration qui nous fournira des outils d'administration supplémentaires pour Node-RED.

d. Testez votre installation :

Node-RED utilise le port : 1880 comme port par défaut. Donc, permet au port d'être utilisé par le service avec la commande suivante :

```
meryam@meryam-virtual-machine:~$ sudo ufw allow 1880  
Rules updated  
Rules updated (v6)
```

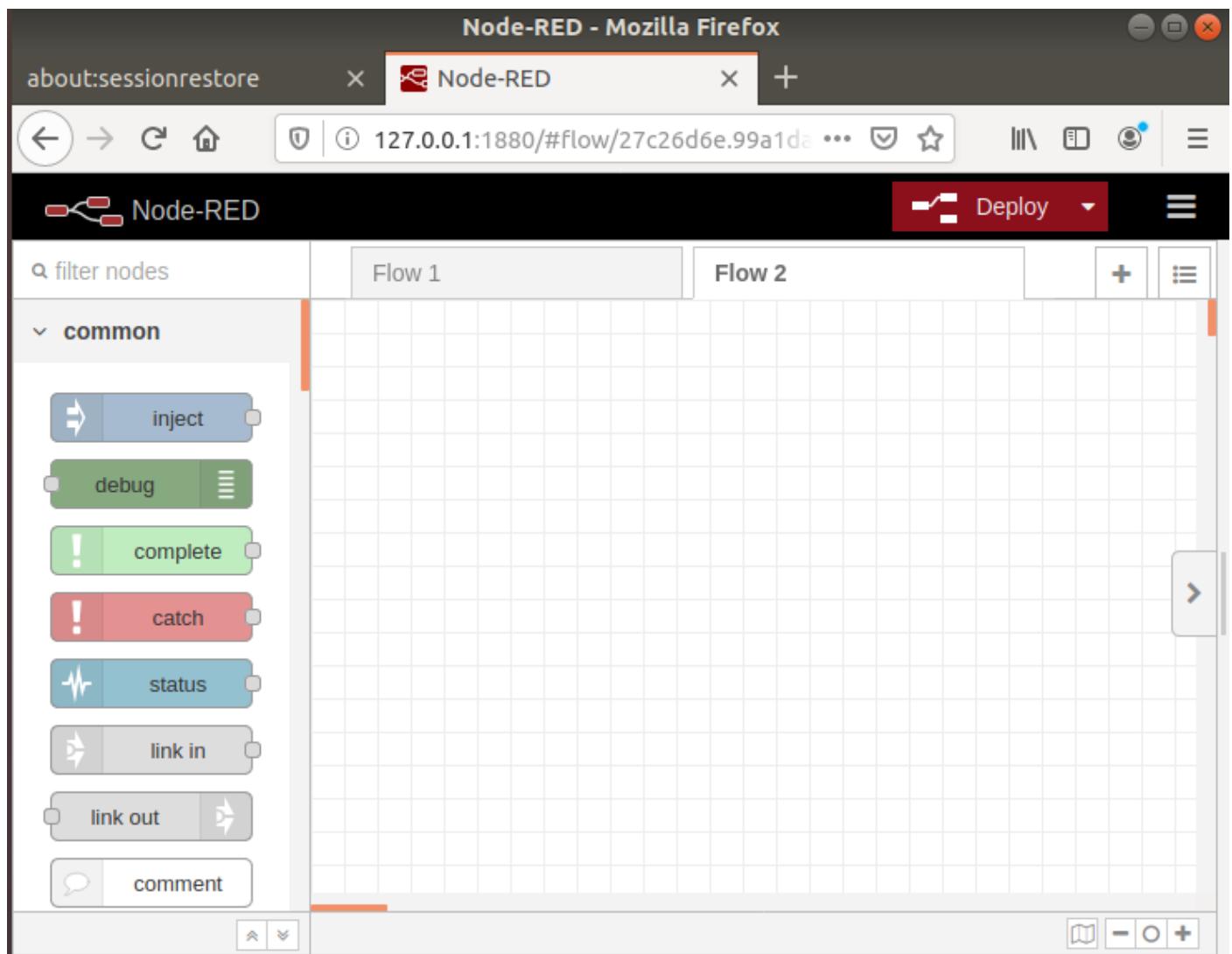
Alors maintenant, nous sommes tous prêts et pouvons commencer à utiliser Node-RED

```
meryam@meryam-virtual-machine:~$ node-red  
8 Apr 18:25:04 - [info]  
  
Welcome to Node-RED  
=====
```

8 Apr 18:25:04 - [info] Node-RED version: v1.0.4
8 Apr 18:25:04 - [info] Node.js version: v8.10.0
8 Apr 18:25:04 - [info] Linux 5.3.0-46-generic x64 LE
8 Apr 18:25:05 - [info] Loading palette nodes
8 Apr 18:25:06 - [info] Settings file : /home/meryam/.node-red/settings.js
8 Apr 18:25:06 - [info] Context store : 'default' [module=memory]
8 Apr 18:25:06 - [info] User directory : /home/meryam/.node-red
8 Apr 18:25:06 - [warn] Projects disabled : editorTheme.projects.enabled=false
8 Apr 18:25:06 - [info] Flows file : /home/meryam/.node-red/flows_meryam-vi
rtual-machine.json
8 Apr 18:25:06 - [info] Creating new flow file
8 Apr 18:25:06 - [warn]

Vous pouvez ouvrir n'importe quel navigateur installé sur votre système et taper sur barre d'url l'adresse suivante :

```
http://localhost:1880
```



2. MQTT Explorer :

MQTT Explorer est un outil qui permet d'intégrer de nouveaux services, des appareils IoT dans un réseau. Cette application s'abonne à toutes les rubriques de votre serveur MQTT et affiche votre hiérarchie de files d'attente de messages, vous permettant d'accéder aux rubriques qui vous intéressent. On a installé l'application depuis Ubuntu software



Figure 10 : logo de MQTT Explorer

3. Contiki

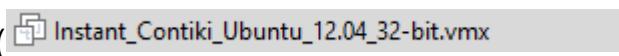
Contiki est un système d'exploitation similaire à Microsoft Windows et Linux . Cependant, pour une raison tout à fait précise et en particulier centrée sur les « questions » au sein de l'IoT. Diverses fonctions d'un système d'exploitation comprennent le contrôle logiciel / technique, la manipulation de ressources utiles, la gestion de la mémoire et la gestion des échanges verbaux. Le but de Contiki OS est de répondre aux exigences des plus petits gadgets avec une saleté intelligente. Les choses doivent pouvoir se communiquer des informations.

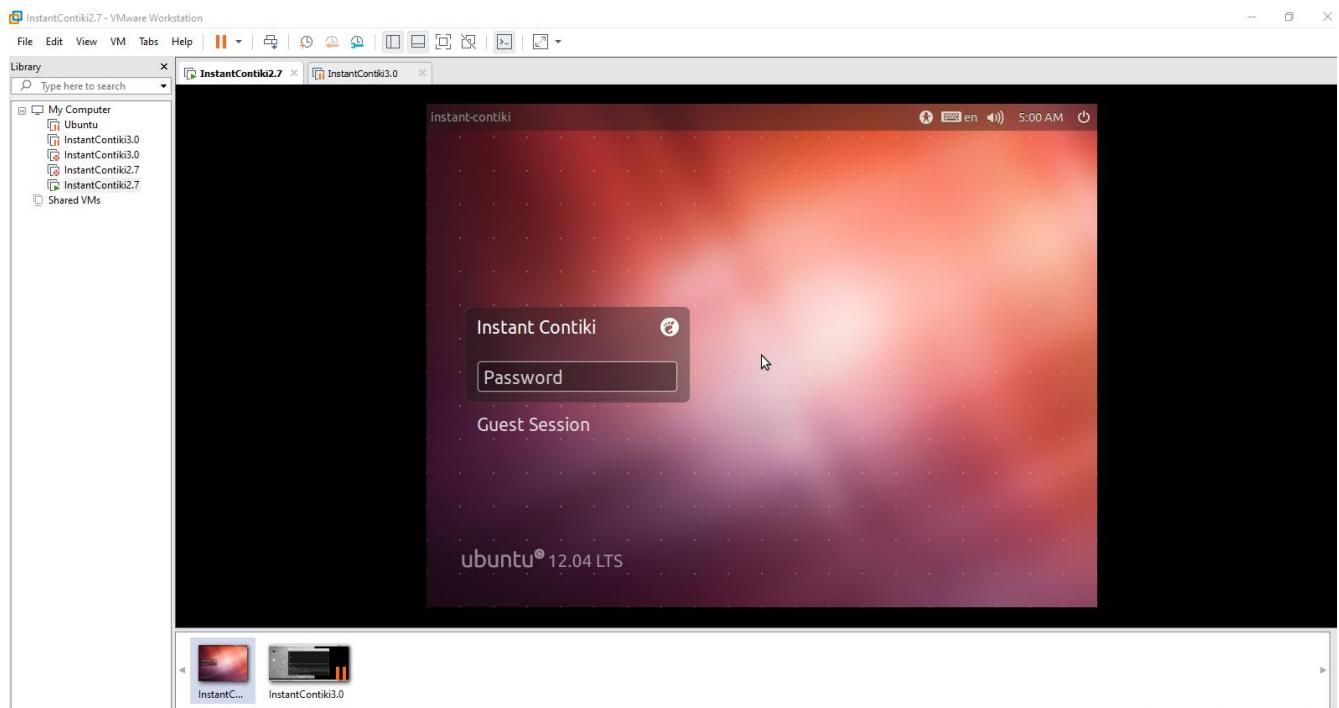
a. Installation de contiki :

Après avoir téléchargé Instant Contiki en suivant ce lien

[<https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%202.7/>]

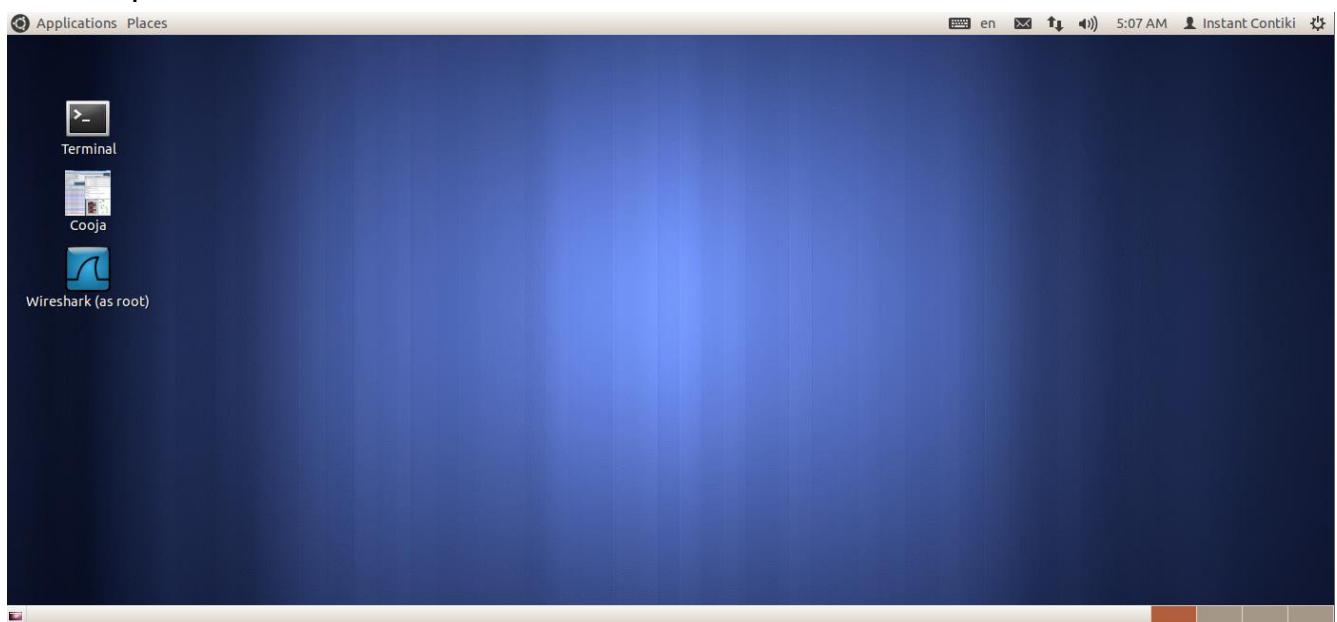
Vous devez également télécharger VMWare Player ou VirtualBox, si vous utilisez Windows comme système d'exploitation hôte. Si vous utilisez MacOS X, vous devez télécharger VMWare Fusion, au lieu de VMWare Player.

Après l'installation de Vmware, vous lancez l'os contiki en exécutant le fichier à extension Vmx ().

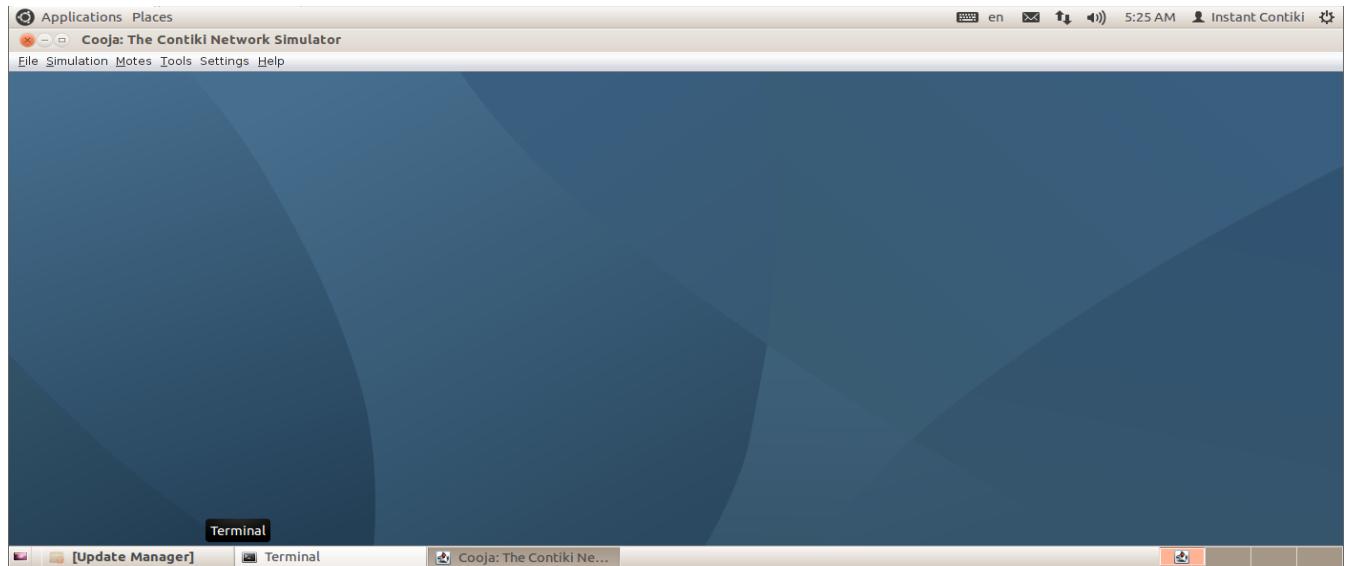


IMPORTANT : Pour vous connecter sur Contiki instantanée : nom d'utilisateur : user ; Mot de passe : user

Ce qui nous mène à l'interface utilisateur :



L'application du simulateur qui nous intéresse est nommé COOJA. En cliquant dessus, elle s'ouvre avec un terminal qui doit rester ouvert tout au long du travail :



A partir de cette étape, vous pouvez créer votre scénario de simulation et vous pouvez le suivre dans la partie pratique.

A screenshot of a terminal window titled 'Terminal'. The window shows the output of a build script. The text is as follows:

```
init:  
clean:  
  [delete] Deleting directory /home/user/contiki/tools/cooja/apps/powertracker/  
build  
compile:  
  [mkdir] Created dir: /home/user/contiki/tools/cooja/apps/powertracker/build  
  [javac] Compiling 1 source file to /home/user/contiki/tools/cooja/apps/power  
tracker/build  
jar:  
  [jar] Building jar: /home/user/contiki/tools/cooja/apps/powertracker/lib/p  
owertracker.jar  
run:  
  [java] INFO [AWT-EventQueue-0] (GUI.java:2846) - External tools default se  
ttings: /external_tools_linux.config  
  [java] INFO [AWT-EventQueue-0] (GUI.java:2876) - External tools user setti  
ngs: /home/user/.cooja.user.properties
```

CHAPITRE II : La mise en place des protocoles IOT

Nous avons choisi les protocoles MQTT , COAP et HTTP pour nos simulations :

1. MQTT :

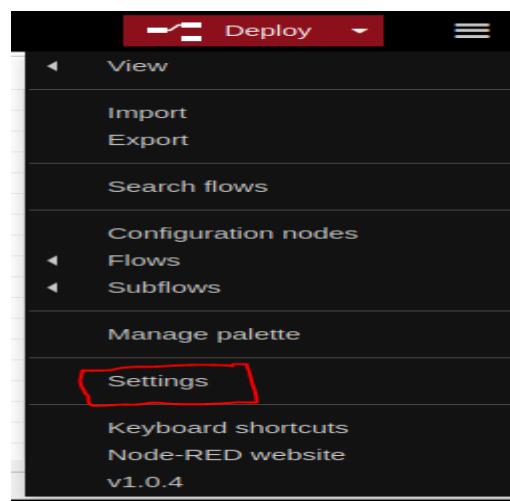
Vous pouvez utiliser Mosca qui est un Broker MQTT écrit en Node.js.



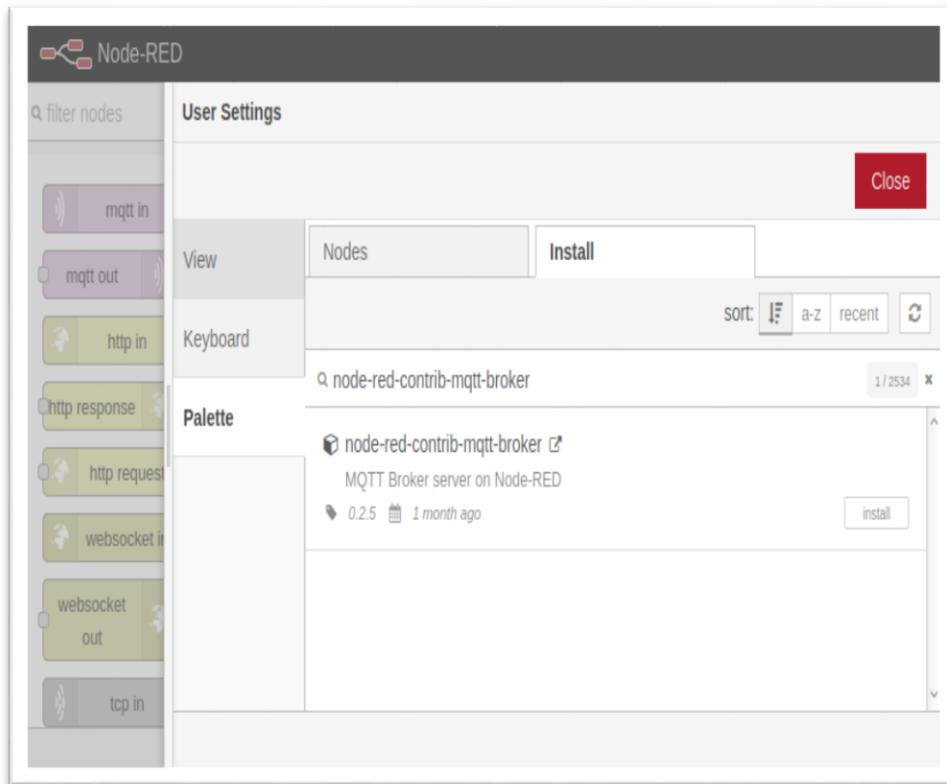
Et parce qu'il ne fait pas partie des nœuds principaux, vous devez l'installer à l'aide du gestionnaire de packages npm ou via le panneau de configuration Admin Node-red.

a. Installation de Mosca à l'aide de l'administrateur Node-Red :

Dans l'administrateur **node-red**, allez dans Setting et après **Palette**.

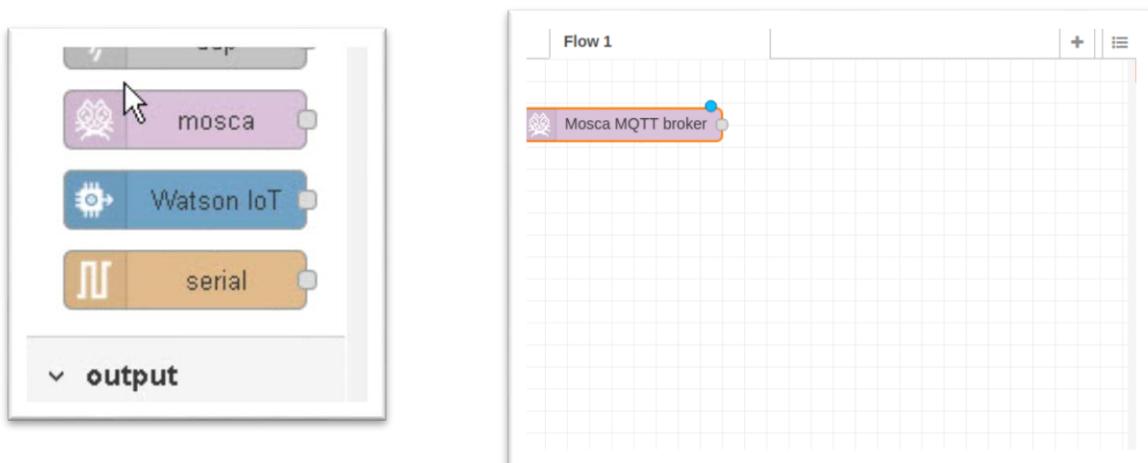


Cliquez sur l'onglet d'installation et recherchez [node-red-contrib-mqtt](#).



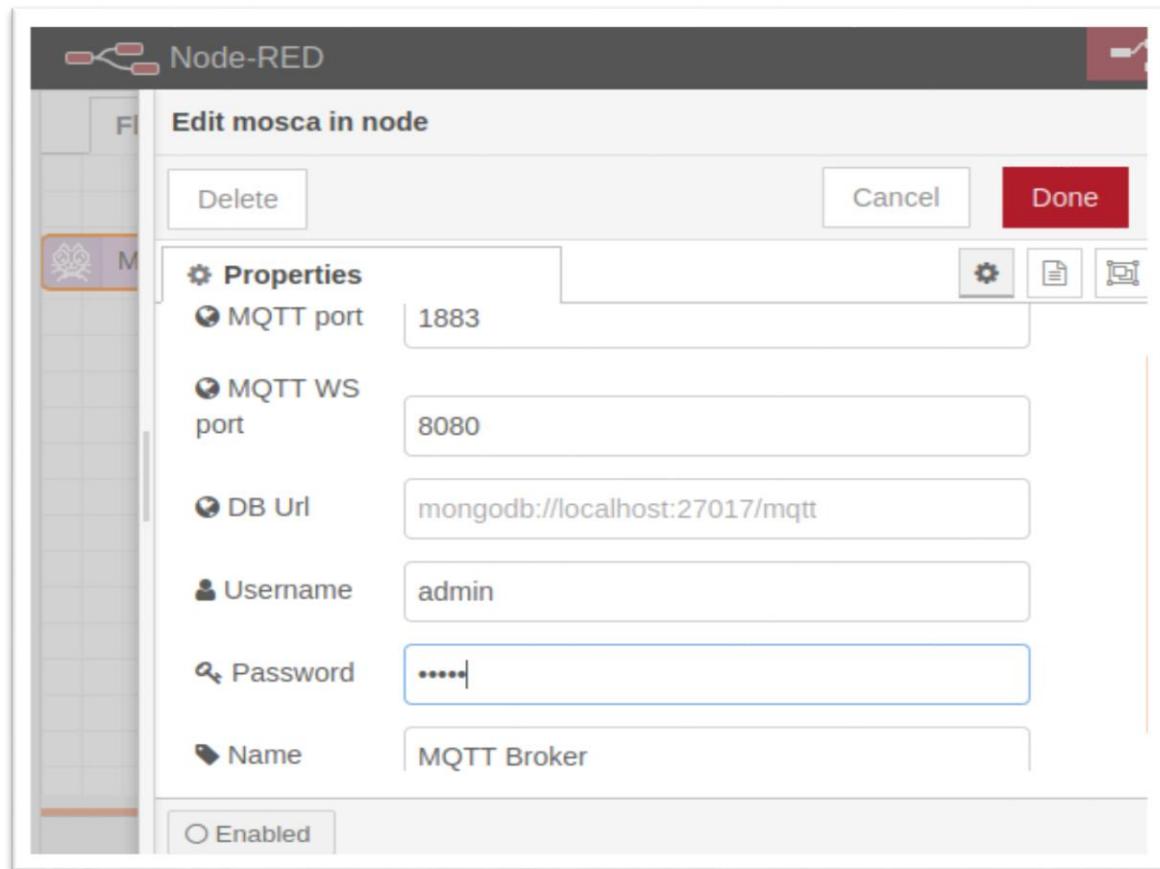
Après l'installation, vous devriez trouver un nouveau nœud appelé mosca dans les catégories d'entrée.

Faites glisser et déposez sur le canevas à utiliser.



b. Configuration du nœud Mosca :

Dans cet exemple, nous allons créer un tableau de bord MQTT pour afficher et contrôler deux capteurs ou appareils MQTT.



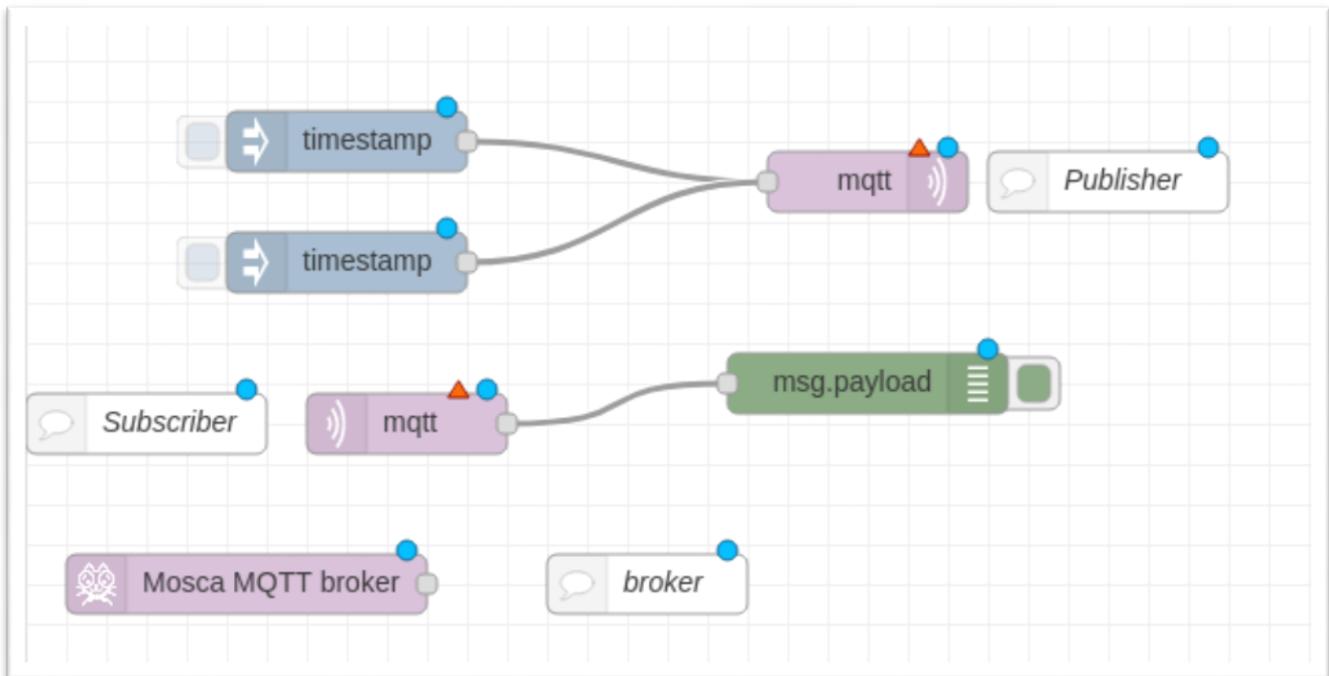
Vous devez définir les ports pour MQTT et MQTT standard sur les Websockets.

Pour les champs de nom d'utilisateur et de mot de passe vous pouvez généralement les laisser vides.

c. Exemple 1 :

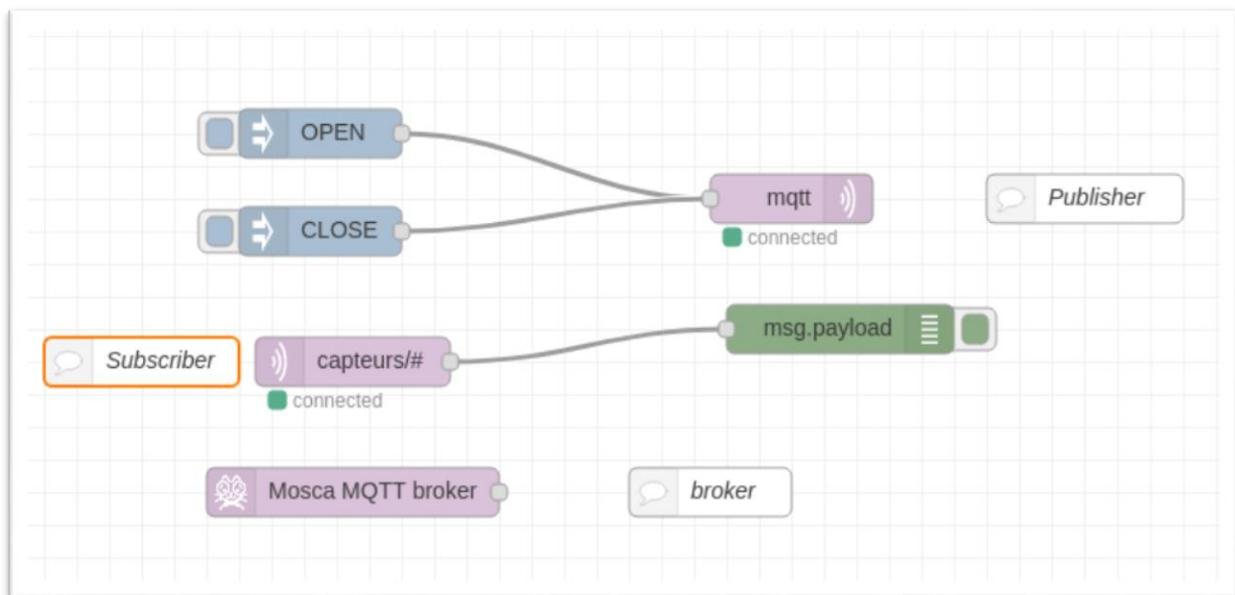
Vous créez un capteur MQTT, qui peut être contrôlé via Internet en envoyant de simples commandes MQTT.

L'utilisation des nœuds d'entrée et de sortie MQTT se fait selon la topologie suivante :



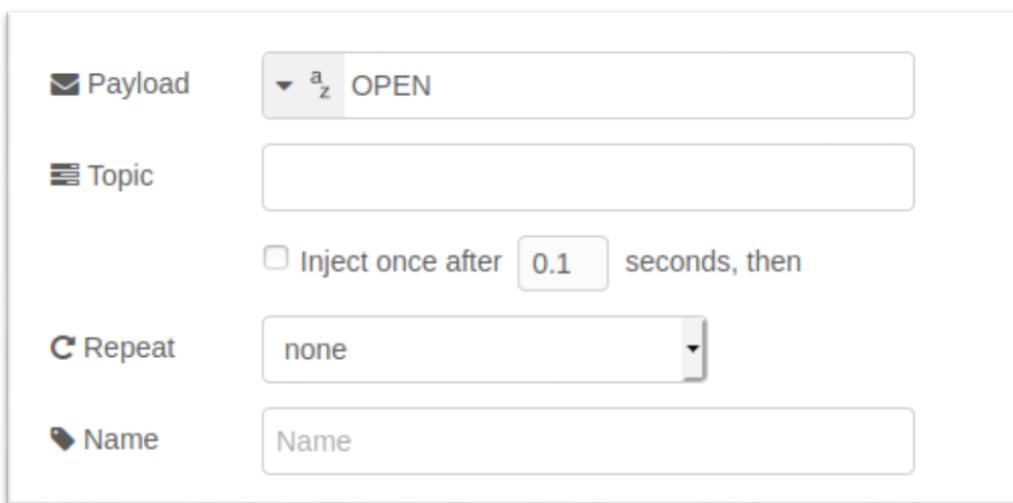
Les triangles en rouge signifient qu'on n'a pas encore configuré les nœuds

Après configuration :

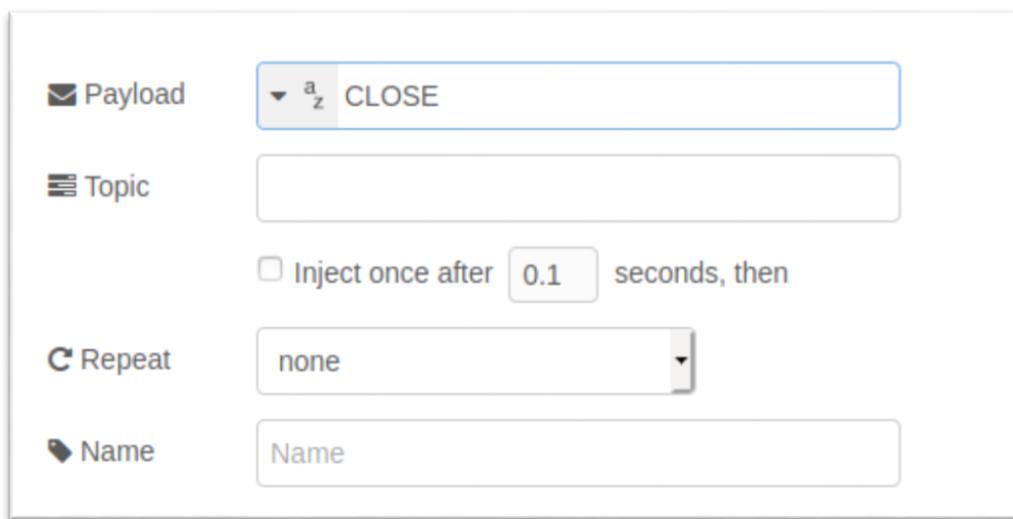


Le nœud Injecter vous permet d'injecter des messages dans un flux, soit en cliquant sur le bouton du nœud, soit en définissant un intervalle de temps entre les injections.

Dans notre exemple on a utilisé 2 injecte nœud : une pour close et l'autre pour open



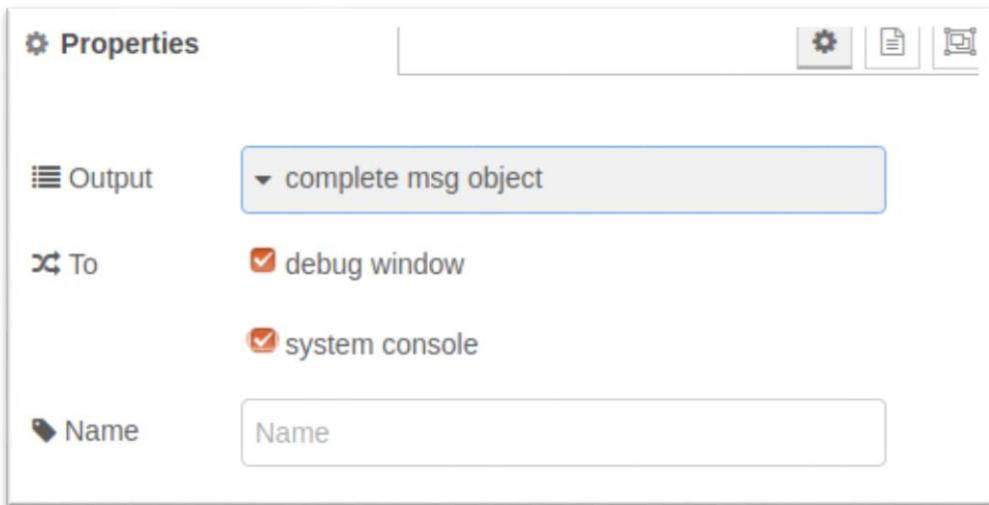
Choisir string dans payload et écrire OPEN



Choisir string dans payload et écrire CLOSE

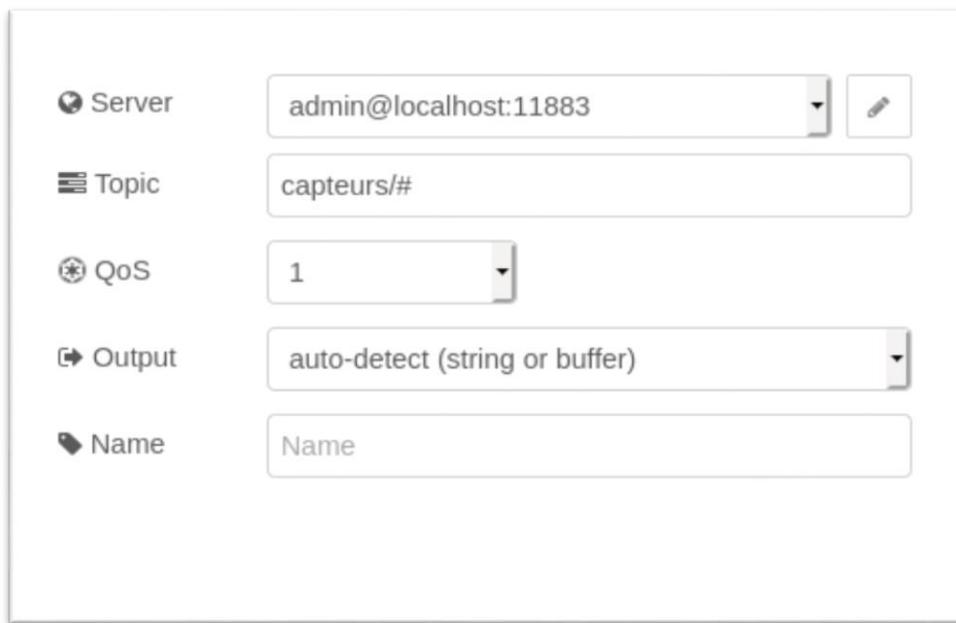
[Le nœud de débogage](#) : entraîne l'affichage de tout message dans la barre latérale de débogage. Par défaut, il affiche simplement la charge utile du message, mais il est possible d'afficher tout l'objet du message.

On va choisir dans output **complete msg object** pour afficher plus de détails et les afficher dans debug de Node Red ainsi que dans le terminal.



MQTT in (Subscriber) :

Pour vous connecter à un courtier ou à un serveur MQTT, vous devez connaître l'adresse IP ou le nom du courtier et le port qu'il utilise (par défaut 1883). Dans notre cas, on utilise port (11883) et adresse IP local



On est abonné au topic **capteurs/#** c'est-à-dire on est abonné à tous les topics qui commence par capteurs.

MQTT out (Publisher) :

Pour publier un message vers un courtier MQTT, vous utilisez le nœud de publication MQTT.

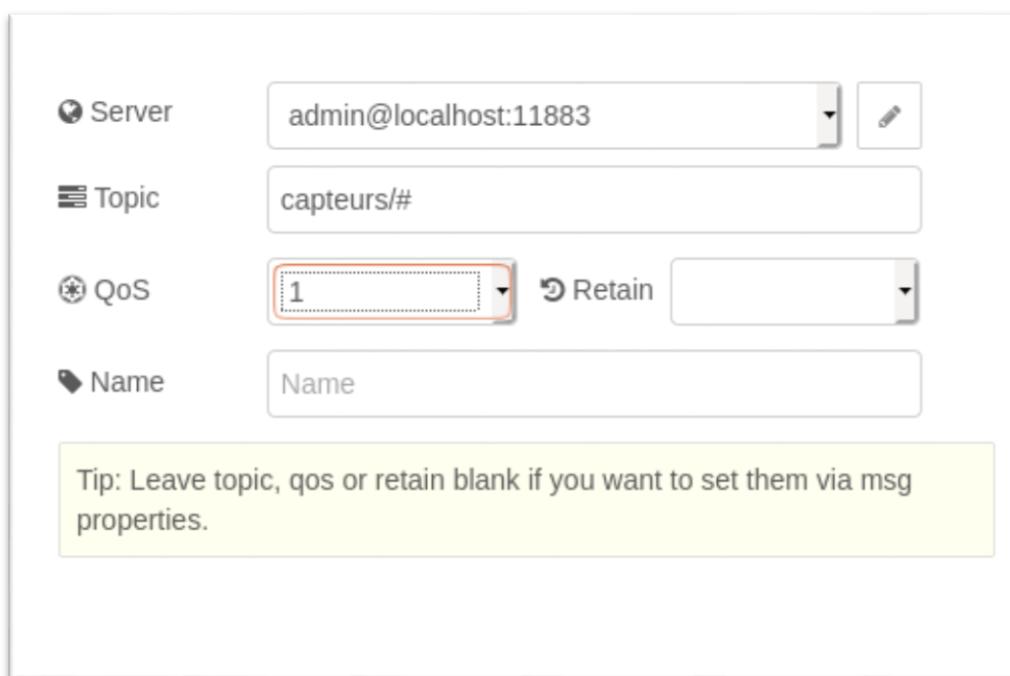
Vous pouvez configurer la qualité de service du message (QOS c'est-à-dire que pour chaque message envoyé, vous pouvez choisir comment le broker doit le gérer et l'indicateur de conservation, il existe 3 types de QOS :

QoS0 : Le message envoyé n'est pas stocké par le Broker. Il n'y a pas d'accusé de réception. Le message sera perdu en cas d'arrêt du serveur ou du client. C'est le mode par défaut

QoS1 : Le message sera livré au moins une fois. Le client renvoie le message jusqu'à ce que le broker envoie en retour un accusé de réception.

QoS2 : Le broker sauvegarde le message et le transmet jusqu'à ce qu'il soit reçu par tous les souscripteurs connectés.

On peut aussi configurer, l'indicateur de conservation. Il peut être soit une chaîne d'objet JavaScript, soit un tampon binaire.



Après la configuration on va faire deploy



Si on revient au terminal, il indique que le broker est connecté

```
13 Apr 12:57:38 - [info] [mqtt-broker:9c5b5719.490f2] Connected to broker: mqtt:
```

Donc maintenant si on clique sur open/close, il s'affiche dans le déboguer de Node Red.

The screenshot shows two messages in the Node-RED debug tab. The first message is a publish event from the topic 'capteurs/#' with payload 'OPEN'. The second message is a publish event from the topic 'capteurs/#' with payload 'CLOSE'.

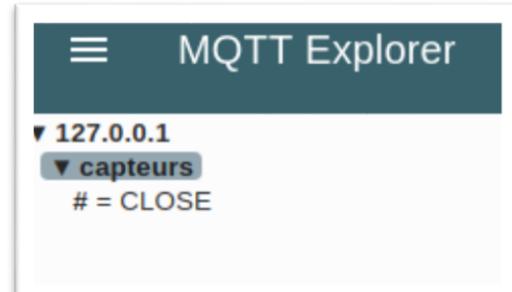
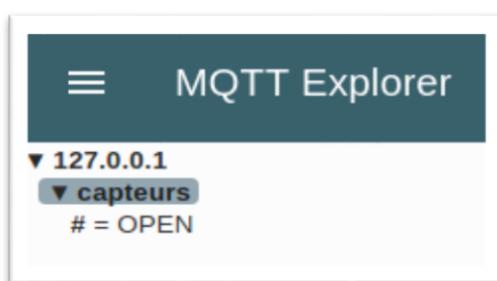
```
4/13/2020, 2:37:21 PM node: 4dae/298.5b4d1c
capteurs/# : msg : Object
  ▼ object
    topic: "capteurs/#"
    payload: "OPEN"
    qos: 1
    retain: false
    _topic: "capteurs/#"
    _msgid: "97e307ca.a54308"

4/13/2020, 2:37:55 PM node: 4dae7298.5b4d1c
capteurs/# : msg : Object
  ▼ object
    topic: "capteurs/#"
    payload: "CLOSE"
    qos: 1
    retain: false
    _topic: "capteurs/#"
    _msgid: "fc968bb0.a57da8"
```

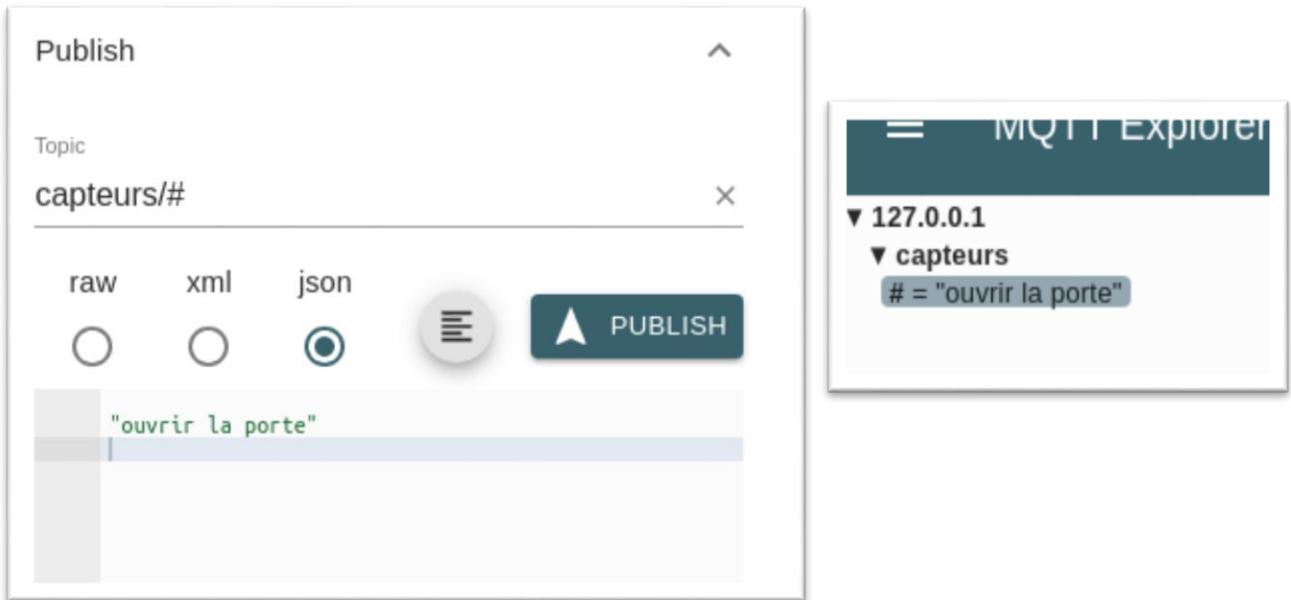
Les informations s'affichent aussi sur le terminal :

```
{ topic: 'capteurs/#',
  payload: 'CLOSE',
  qos: 1,
  retain: false,
  _topic: 'capteurs/#',
  _msgid: 'f0e8f74f.c0b348' }
```

L'utilisation de l'application MQTT Explorer nous permet de voir les messages et aussi d'y répondre, c'est-à-dire que le client pour envoyer un message, ce qui signifie une communication à 2 sens



On envoie un message depuis le client :



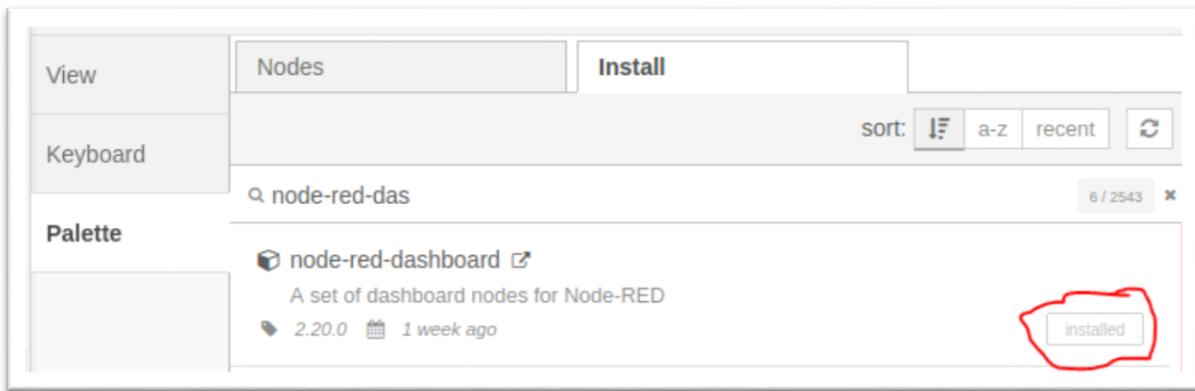
Le message publier par le client s'affiche dans le déboguer :

```
4/13/2020, 3:02:22 PM node: 4dae7298.5b4d1c
capteurs/# : msg : Object
  ▶ object
    topic: "capteurs/#"
    ▶ payload: "«ouvrir la porte»"
    qos: 1
    retain: false
    _topic: "capteurs/#"
    _msgid: "7d64e5b8.527d0c"
```

d. Exemple 2 : utiliser MQTT avec Dashboard :

Installation de Dashboard :

Avec les mêmes étapes qui précèdent pour installer MQTT Broker, on installe Dashboard

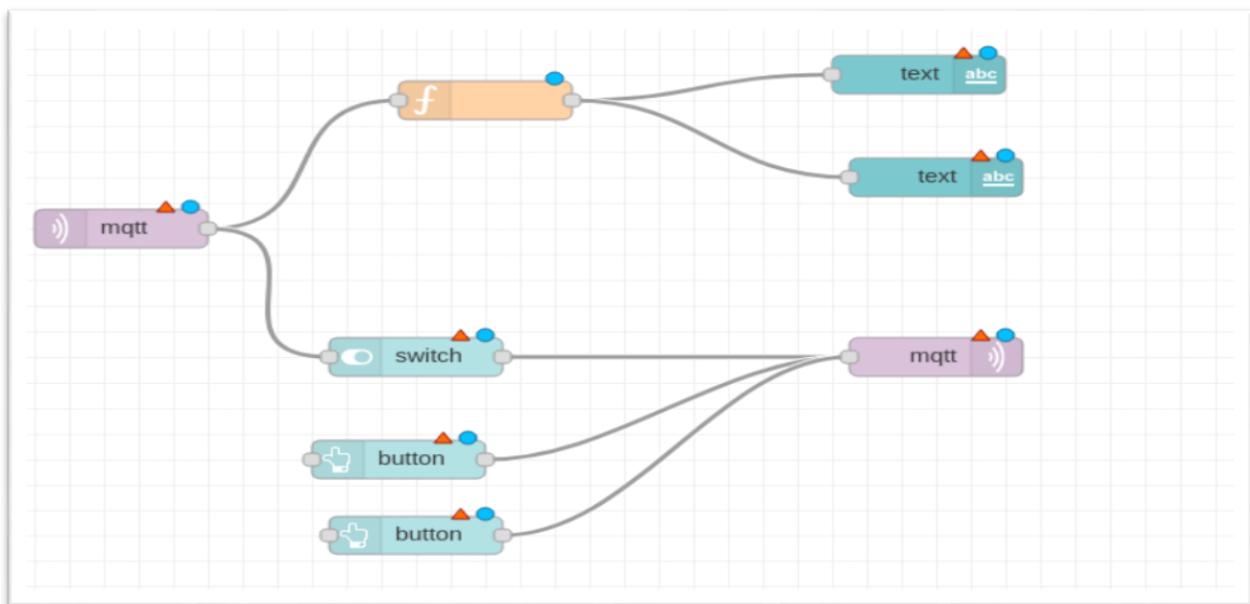


Pour accéder à Dashboard :

127.0.0.1:1880/ui

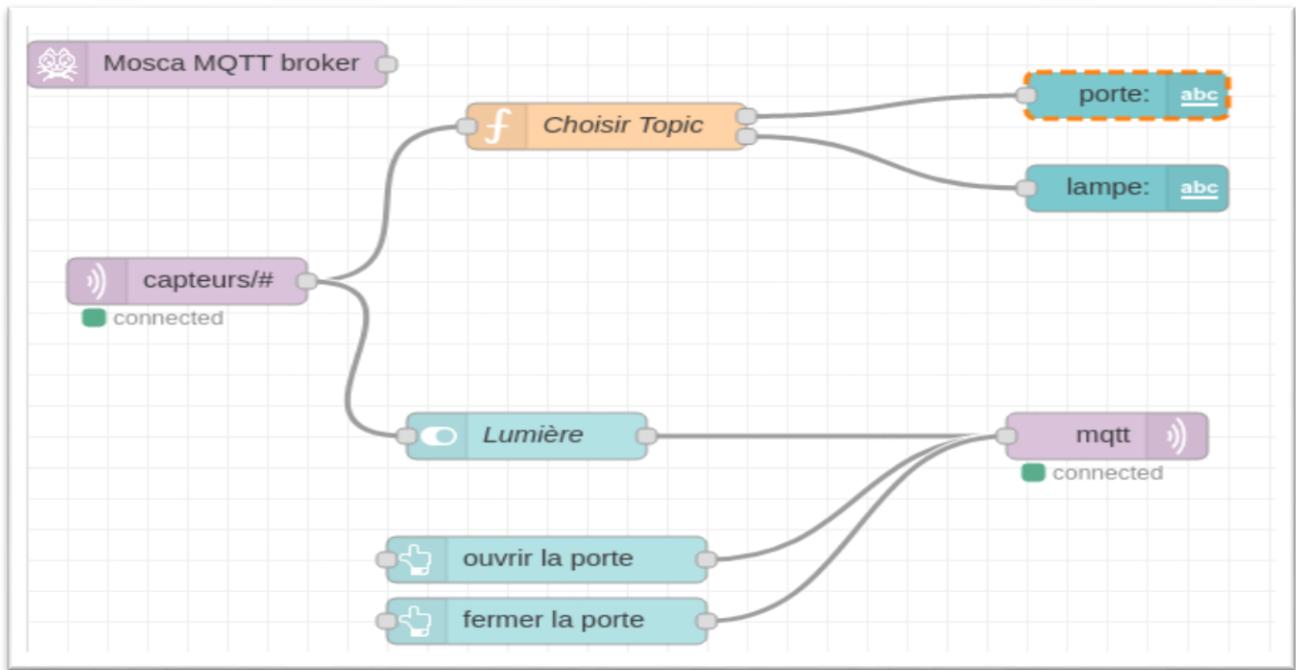
Test de l'installation :

Nous allons créer un tableau de bord MQTT pour afficher et contrôler deux capteurs\appareils MQTT.



Les triangles en rouge indiquent que les nœuds ne sont pas encore configurés et les cercles en bleu indiquent qu'on n'a pas encore déployé notre topologie.

Après configuration :



Les étapes pour configurer :

Les nœuds en bleu sont des Dashboard :

On va créer un onglet **Ma chambre** qui est divisé en deux groupes **contrôler** et **capteurs**

Capteurs contient : porte et lampe

Contrôler contient : fermer la porte, ouvrir la porte et la lumière

- **Bouton switch :** c'est l'interrupteur de l'éclairage

On a réglé Payload pour publier une chaîne de caractères (string) On ou Off selon le cas des messages debug et l'interface de Dashboard, il sera publié exactement dans le topic qu'on a configuré

	Group	[Ma chambre] Controller	<input type="button" value="▼"/>	<input type="button" value="✎"/>
	Size	auto		
	Label	switch		
	Tooltip	optional tooltip		
	Icon	Default	<input type="button" value="▼"/>	
→ Pass though msg if payload matches new state: <input checked="" type="checkbox"/>				
<input type="checkbox"/> When clicked, send:				
	On Payload	<input type="button" value="▼"/>	a _z	On
	Off Payload	<input type="button" value="▼"/>	a _z	Off
	Topic	capteurs/lumière-principale/controller		

- **Bouton pour ouvrir la porte :** on a défini l'Icon depuis un site qui contient le design des icônes, on a choisi la couleur du background et enfin le topic où seront publiées les informations

	Group	[Ma chambre] controller	<input type="button" value="▼"/>	<input type="button" value="✎"/>
	Size	auto		
	Icon	touch_app		
	Label	ouvrir la porte		
	Tooltip	optional tooltip		
	Colour	optional text/icon color		
	Background	#f7bdde		
<input type="checkbox"/> When clicked, send:				
	Payload	<input type="button" value="▼"/>	a _z	Open
	Topic	capteurs/porte-principale/controller		

- **Bouton fermer la porte :** la même chose que le bouton précédent, mais à la place d'open, on va écrire CLOSE

Colour	optional text/icon color
Background	#f7bdde
<input checked="" type="checkbox"/> When clicked, send:	
Payload	a_z Close
Topic	capteurs/porte-principale/controler
→ If msg arrives on input, emulate a button click: <input type="checkbox"/>	
Name	

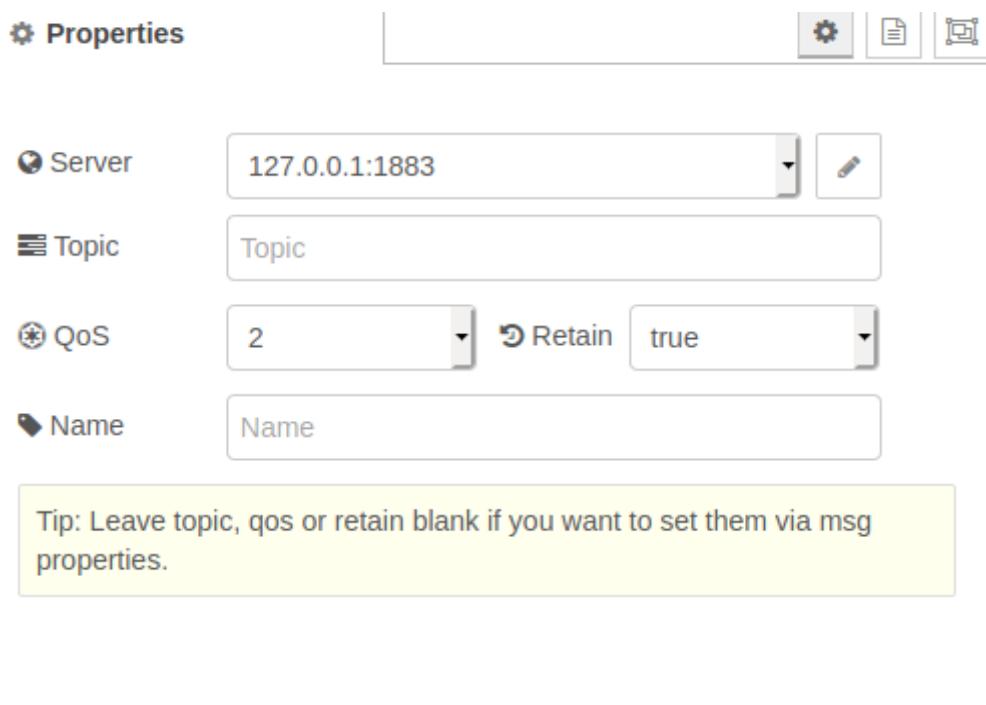
Le rôle de ces 3 Buttons est de publier un message ou de le contrôler en utilisant MQTT

MQTT in nœud (SUBSCRIBE) :

Server	127.0.0.1:1883	<input type="button" value="edit"/>
Topic	capteurs/#	
QoS	2	<input type="button" value="down"/>
Output	auto-detect (string or buffer)	
Name	Name	

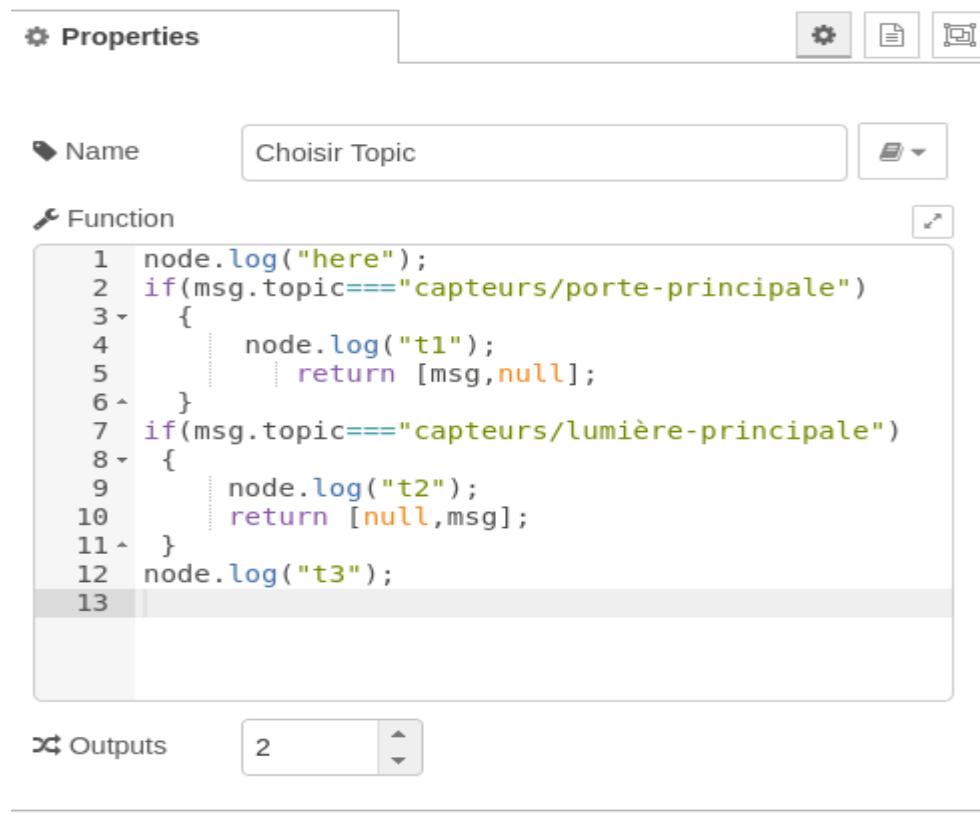
Pour ajouter le serveur(broker) il faut utiliser son adresse IP, et dans le topic on va s'inscrire et écrire : **capteurs/#** c'est-à-dire qu'on va faire Subscribe à tous les topics qui commencent par **capteurs**

MQTT out nœud (PUBLISH) :



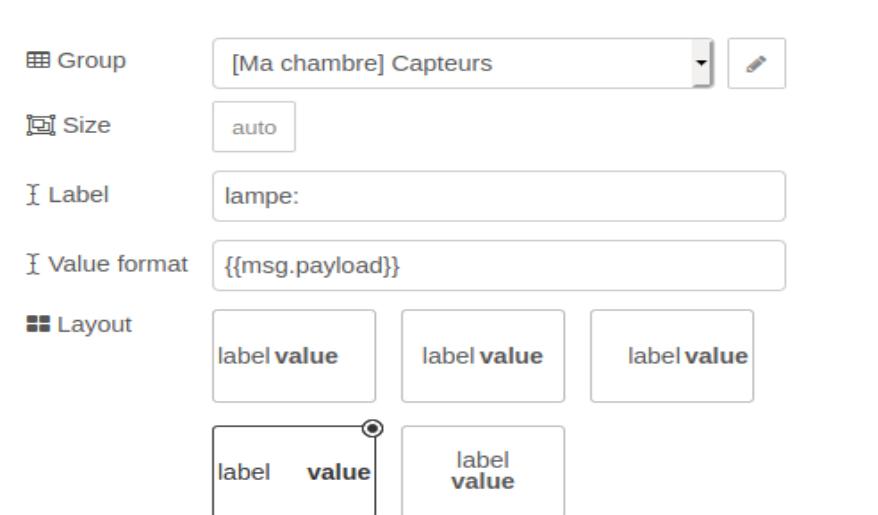
Après la configuration de broker sur MQTT in l'est automatiquement aussi sur MQTT out. On n'a pas ajouté le topic car on utilise le topic qui vient du contrôle bouton de Dashboard

Tout ce qui est envoyé à MQTT in, est renvoyé à la fonction **choisir topic**. On peut les envoyer directement dans les nœuds d'affichage lampe et light, mais ça ne va pas marcher car on va envoyer à la fois des informations de contrôle et de l'état des nœuds. On a donc ajouté la fonction pour filtrer les informations et les envoyer soit à la lampe soit à la porte.

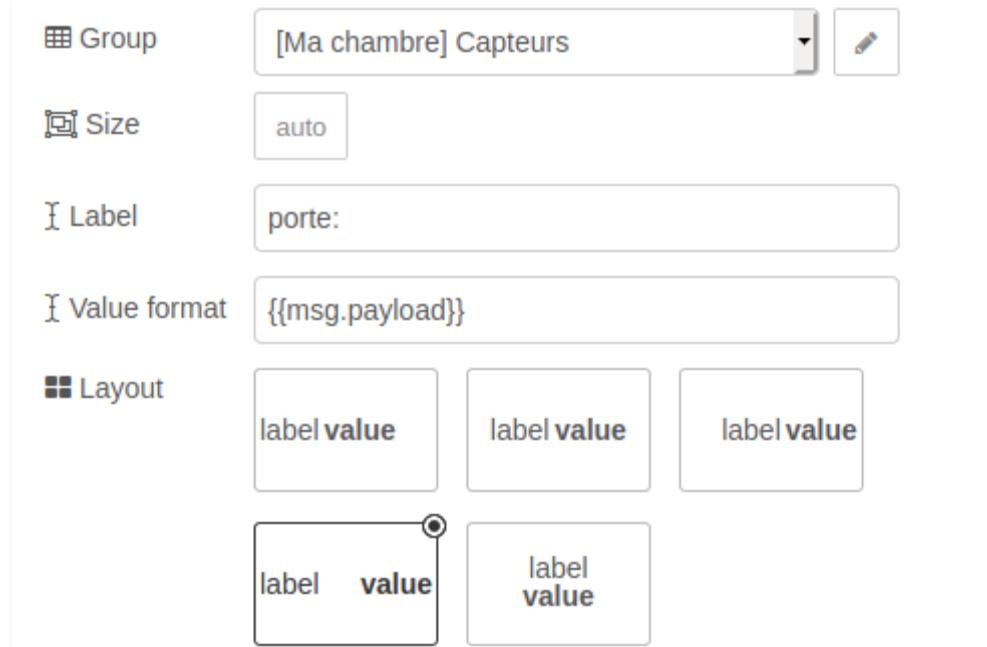


C'est une fonction en java script qui prend les informations d'état capteurs/... et qui transmet ces informations sur l'une des sorties(output) de la fonction. Ainsi, si le message est égal au topic porte, il renvoie le message et la même chose pour le topic lumière.

- **Dans la 1ere sortie de la fonction on a la porte**



- **Et la 2eme sortie** on a la lampe. Toutes les 2 sont dans le groupe capteurs



Finalement, on va cliquer sur Deploy



Dans la page de Dashboard, il sera affiché :

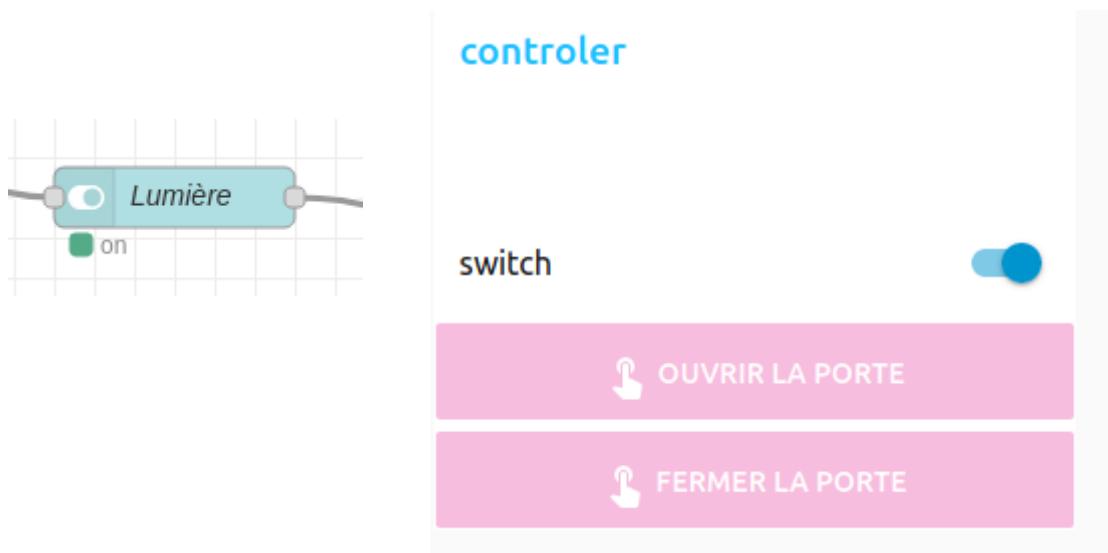
The dashboard has a header 'Ma chambre'. Below it, there is a table with two columns:

Capteurs	controler
lampe:	
porte:	switch <input checked="" type="checkbox"/>

Below the table are two pink buttons:

- OUVRIR LA PORTE** (Open the door)
- FERMER LA PORTE** (Close the door)

Si on tourne le switch ou la porte il s'affiche :

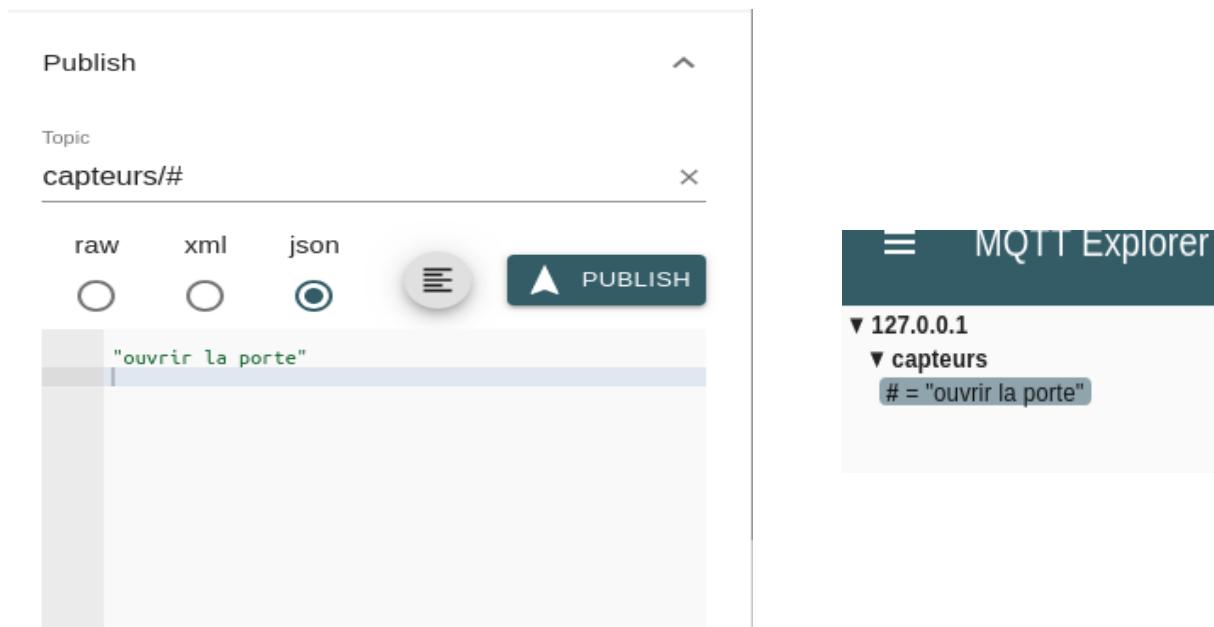


Dans l'interface debugger de node-red :

```
4/13/2020, 4:34:47 PM  node: 4dae7298.5b4d1c
capteurs/lumiere/controler : msg : Object
  ▼ object
    topic: "capteurs/lumiere
    /controler"
    payload: "On"
    qos: 1
    retain: false
    _topic: "capteurs/lumiere
    /controler"
    _msgid: "5f2c2eb5.18646"
```

```
▼ capteurs
  ▼ lumiere
    controler = On
```

On peut aussi envoyer un message depuis le client avec l'interface du client MQTT explorer



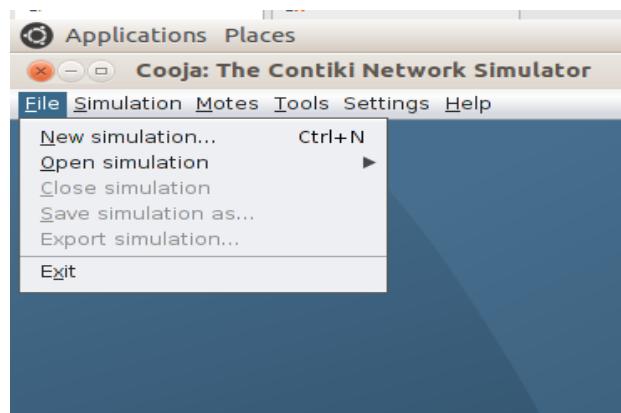
Le message s'affiche dans node-red

```
4/13/2020, 3:02:22 PM  node: 4dae7298.5b4d1c
capteurs/# : msg : Object
  ▼ object
    topic: "capteurs/#"
    ▶ payload: "↔"ouvrir la porte"↔"
    qos: 1
    retain: false
    _topic: "capteurs/#"
    _msgid: "7d64e5b8.527d0c"
```

2. COAP ET HTTP :

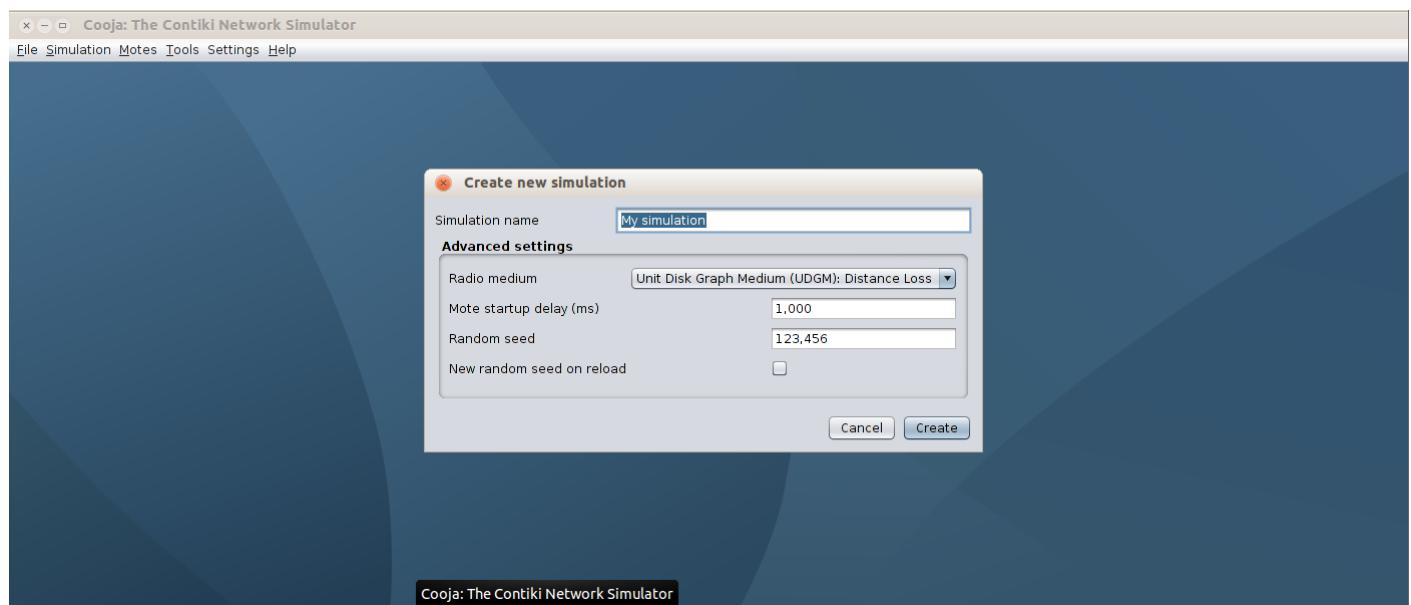
Dans la simulation réalisée avec cooja, on va montrer l'interfonctionnement entre coap et http cités dans la partie théorique, et ce selon les étapes suivantes :

Etape 1 :

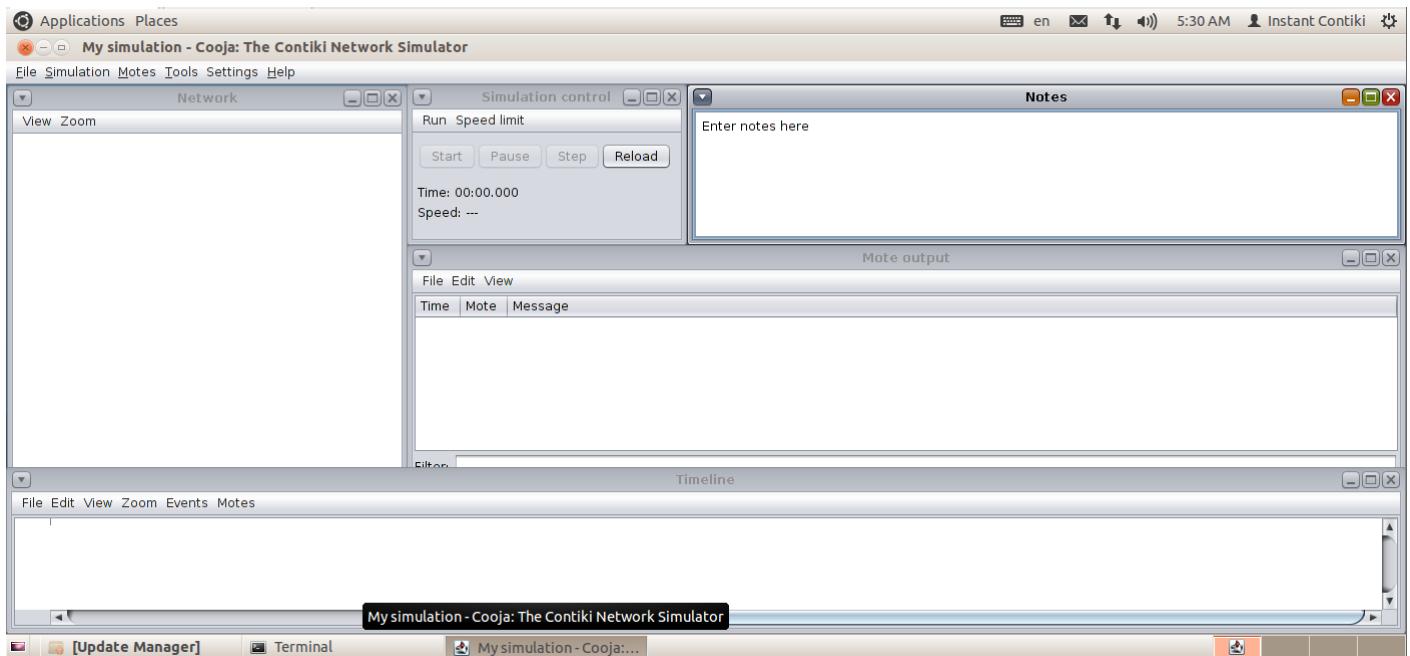


Etape 2 :

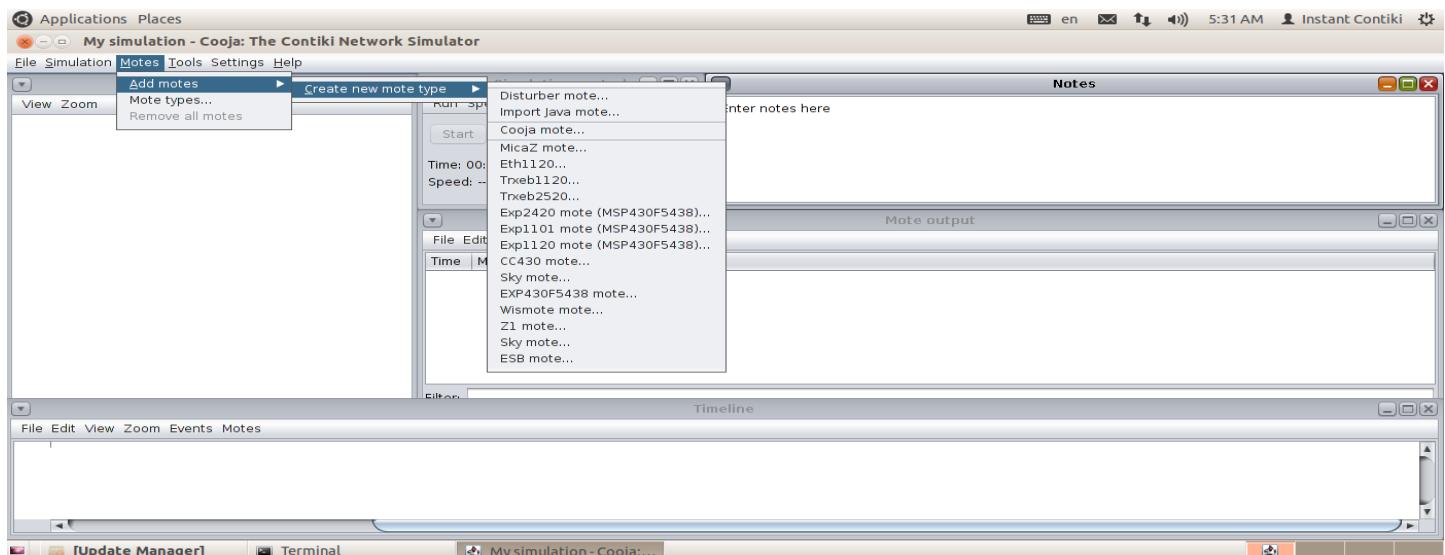
Menu file => nouvelle simulation



Bouton créer qui affiche l'interface suivant :



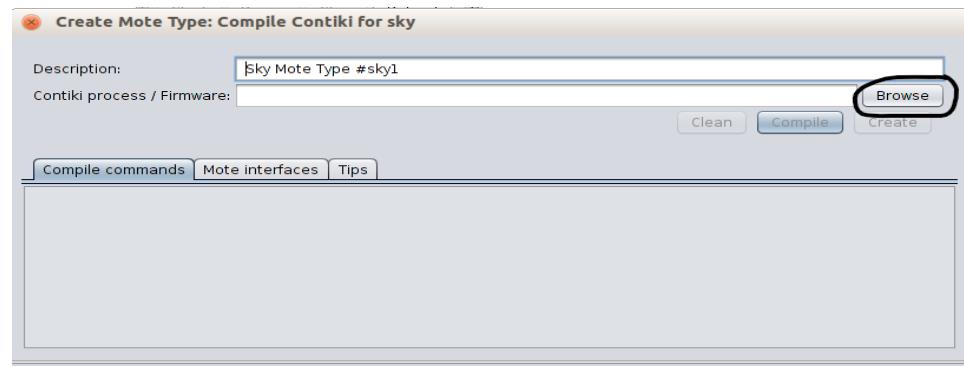
Etape 3 : vous cliquez sur **Motes**



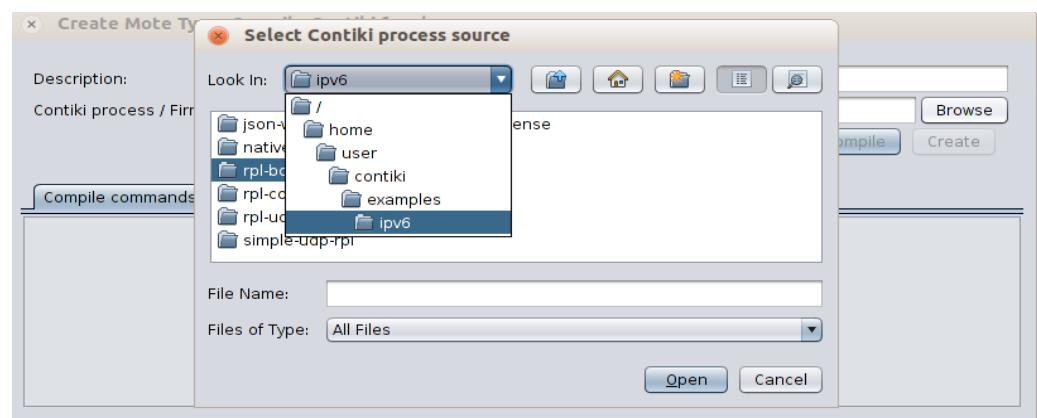
Puis sur **Add motes => create new mote type => sky mote** , de là on doit choisir :

A: pour la création d'un border router sur un fichier border-router .c du sous-dossier rpl-border-router dans le dossier Ipv6

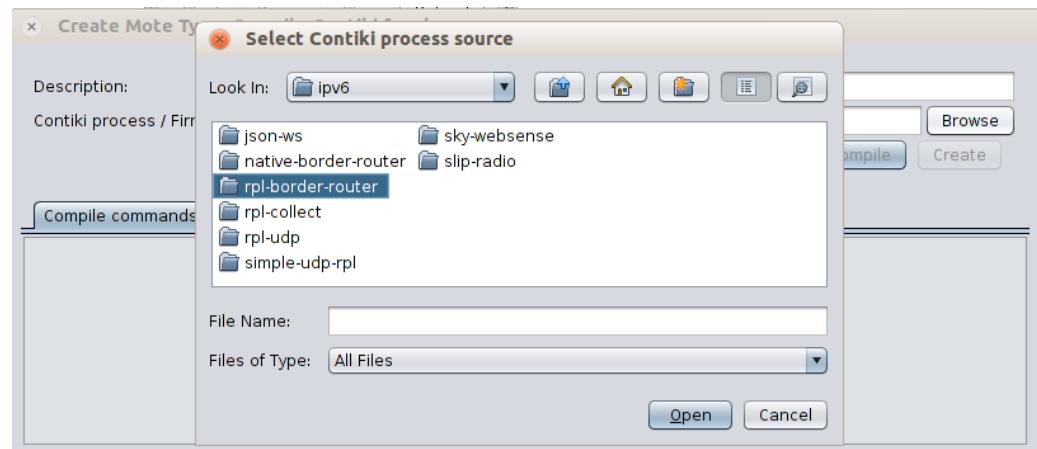
1 :



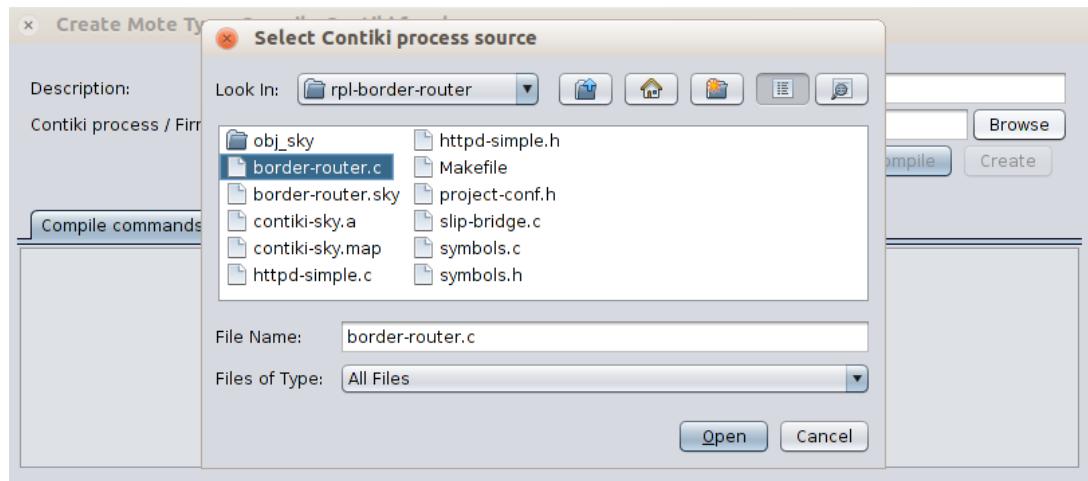
2 :



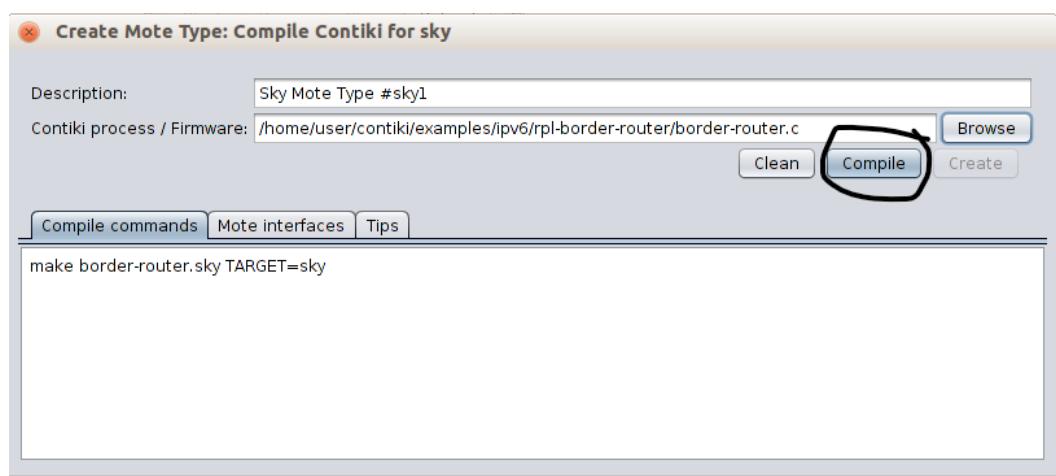
3 :



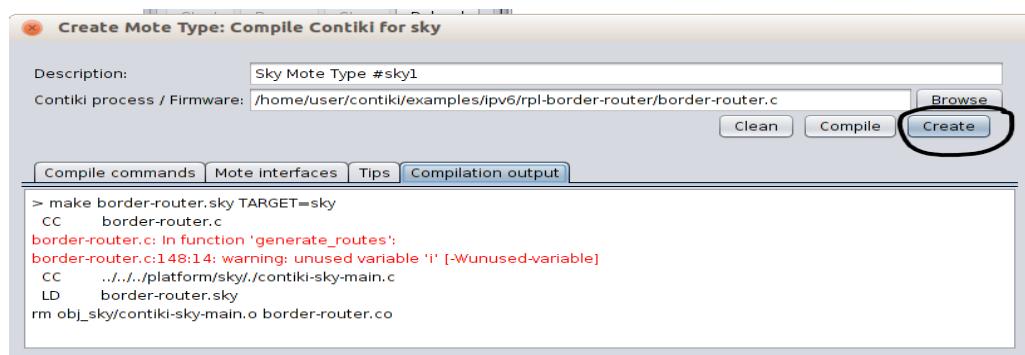
4 :



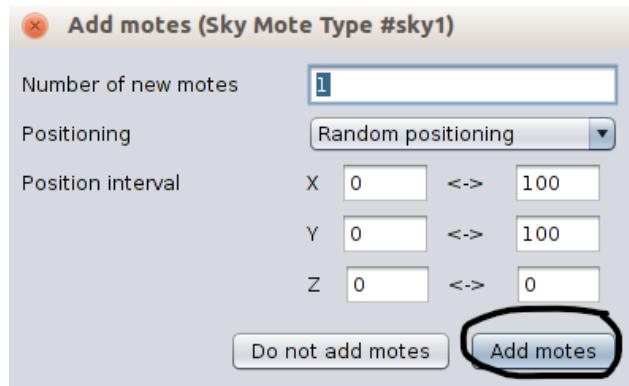
5 : Apres le choix du fichier, on doit le compiler en cliquant sur le bouton compile.



Une Foix compiler, on crée le Mote.

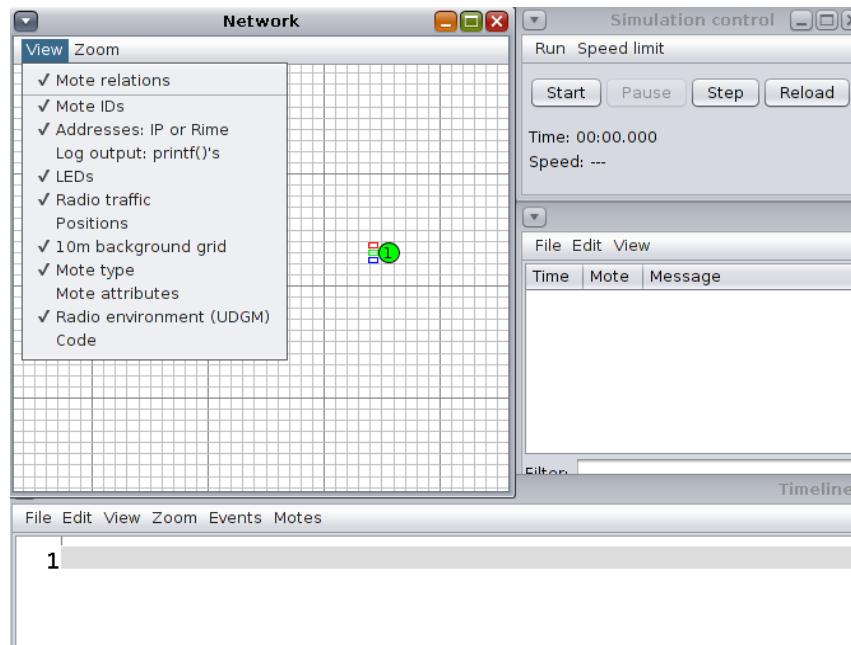


6 :



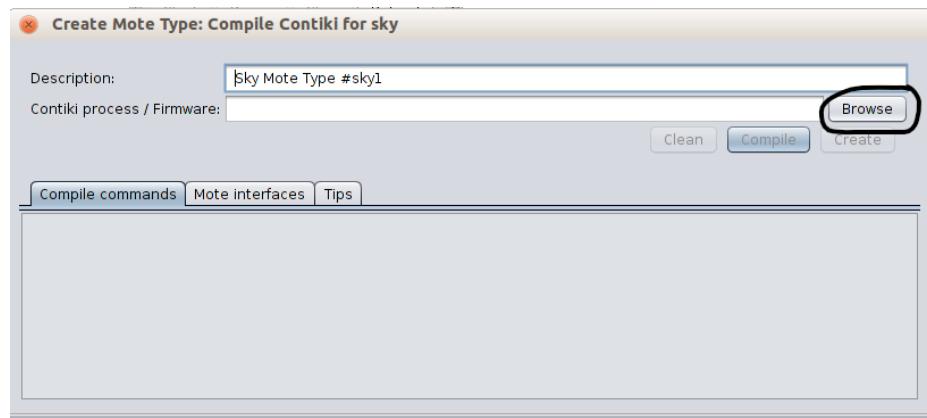
7 :

On choisit le nombre de Motes à ajouter et on valide notre choix, ce qui est l'étape création du border router suivi par les choix des View à afficher

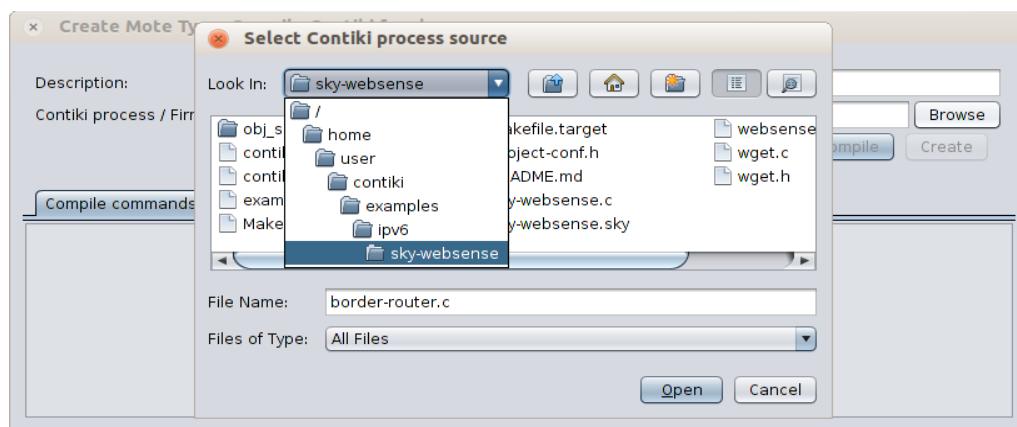


B : Pour la création des capteurs, on choisit le fichier sky-websense.c dans le sous-dossier sky-websense du dossier Ipv6.

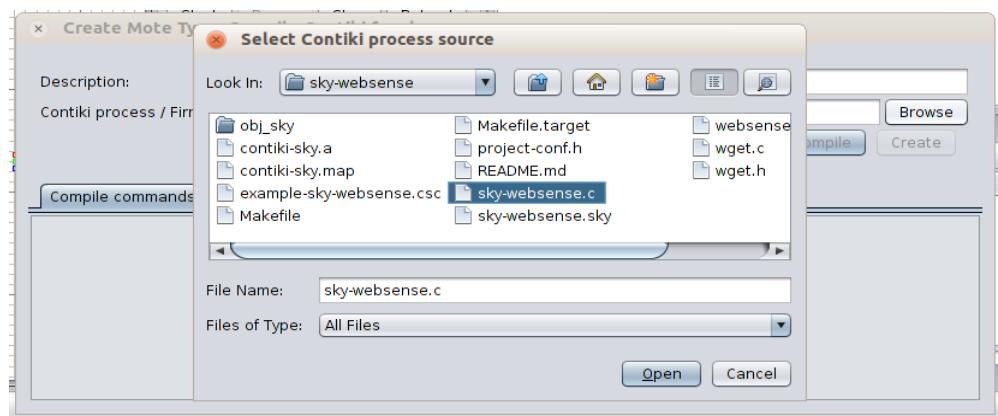
1 :



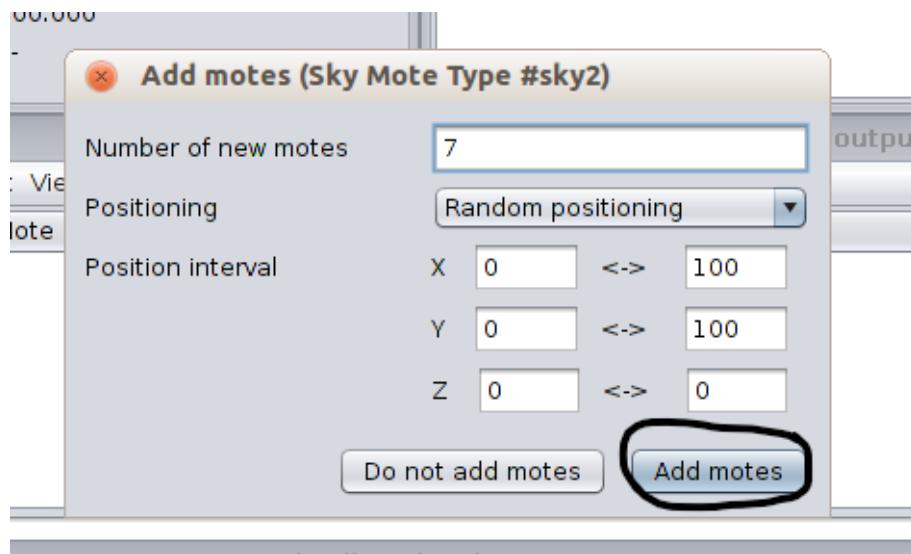
2 :



3 :



4 : A cette étape, le choix est pour 7 capteurs. Ce nombre est facultatif selon votre choix.

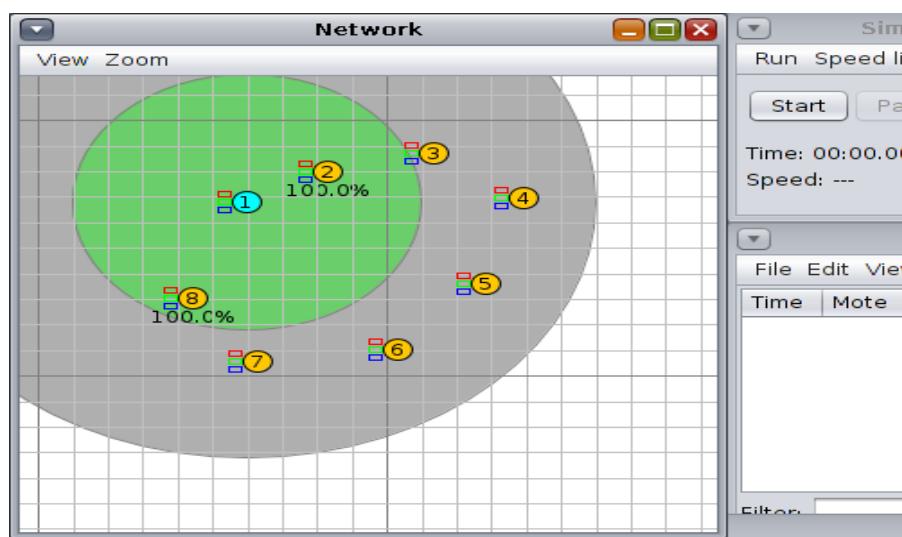


Etape 3 :

Organiser les Motes de façon qu'ils puissent communiquer entre eux. On dit qu'il doivent être dans le même range (le cercle vert qui s'affiche au moment du clic sur le Mote) .

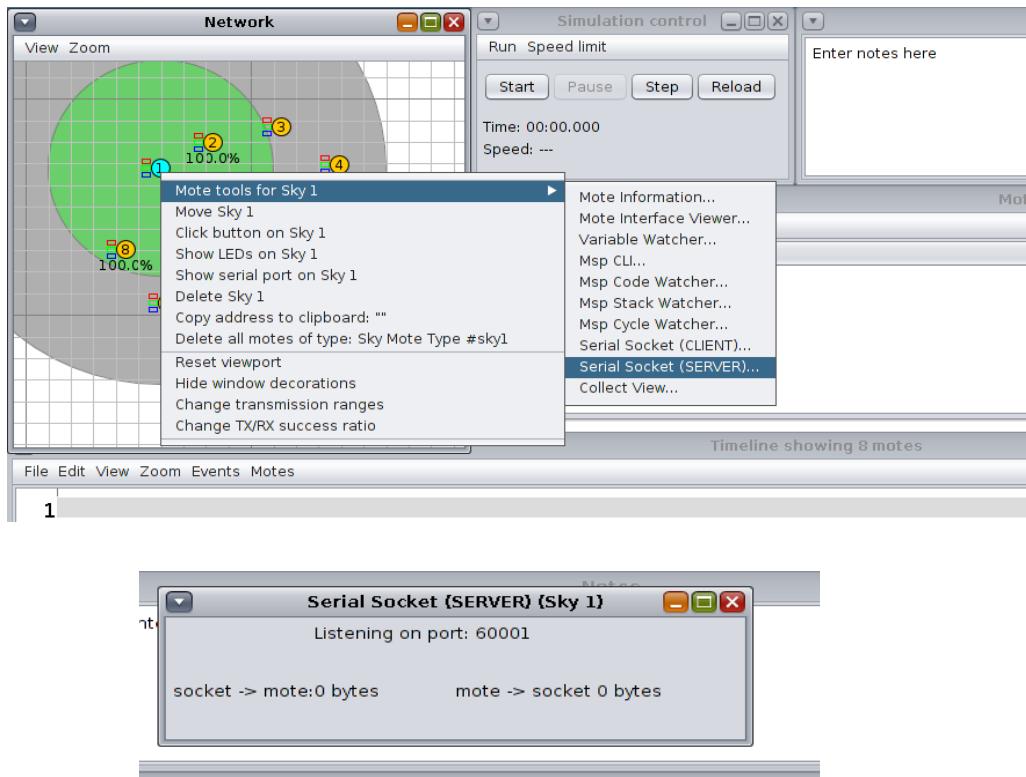
N.B : chaque Mote a son propre range.

Donnons un exemple : Mote 2 et Mote 8 doivent être dans le même range que Mote 1



Etape 4 :

Créer le serial socket server en cliquant sur le bouton gauche de la souris sur le Mote border router (Mote 1) qui affiche le menu contextuel suivant :



Etape 5 :

Ouvrir une fenêtre terminale pour connecter le router par le biais des commandes suivantes :

The screenshot shows a terminal window with the following command history:

```
user@instant-contiki: ~/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help
user@instant-contiki:~/contiki/tools/cooja$ cd ..
user@instant-contiki:~/contiki/tools$ cd ..
cd..: command not found
user@instant-contiki:~/contiki/tools$ cd ..
user@instant-contiki:~/contiki$ cd examples
user@instant-contiki:~/contiki/examples$ cd ipv6
user@instant-contiki:~/contiki/examples/ipv6$ cd rpl-border-router
user@instant-contiki:~/contiki/examples/ipv6/rpl-border-router$ make connect-router-cooja
TARGET not defined, using target 'native'
sudo ../../tools/tunslip6 -a 127.0.0.1 aaaa::1/64
[sudo] password for user:
```

(Le mot de passe est : user)

```

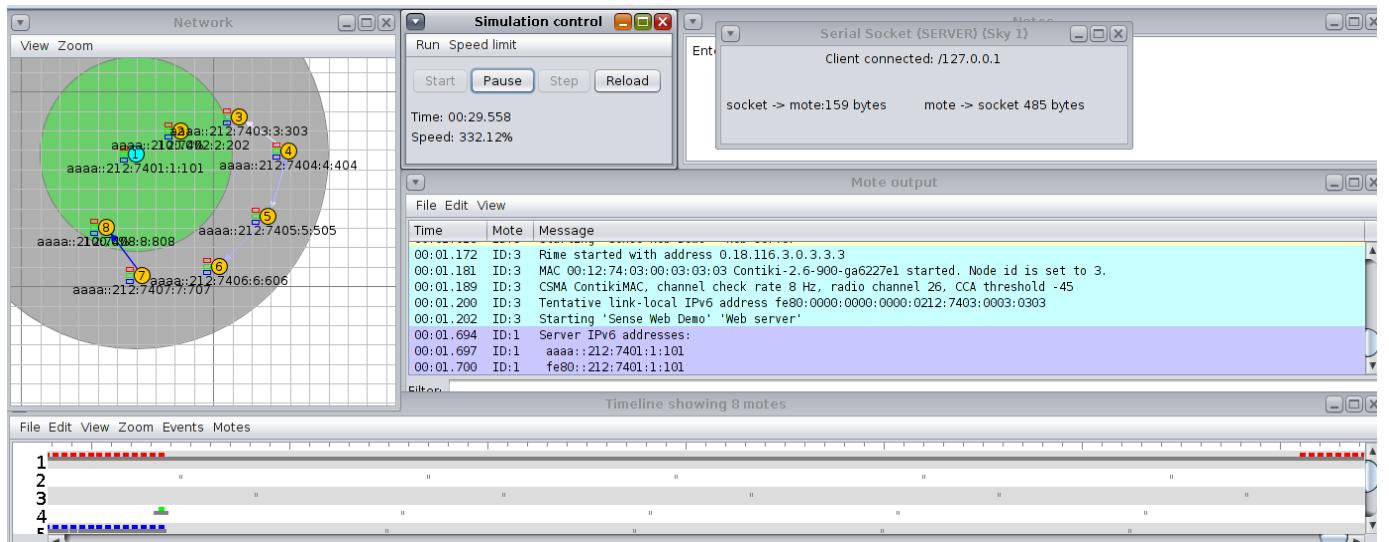
user@instant-contiki: ~/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help
TARGET not defined, using target 'native'
sudo ../../tools/tunslip6 -a 127.0.0.1 aaaa::1/64
[sudo] password for user:
Slip connected to ``127.0.0.1:60001''
opened tun device `/dev/tun0'
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet  addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

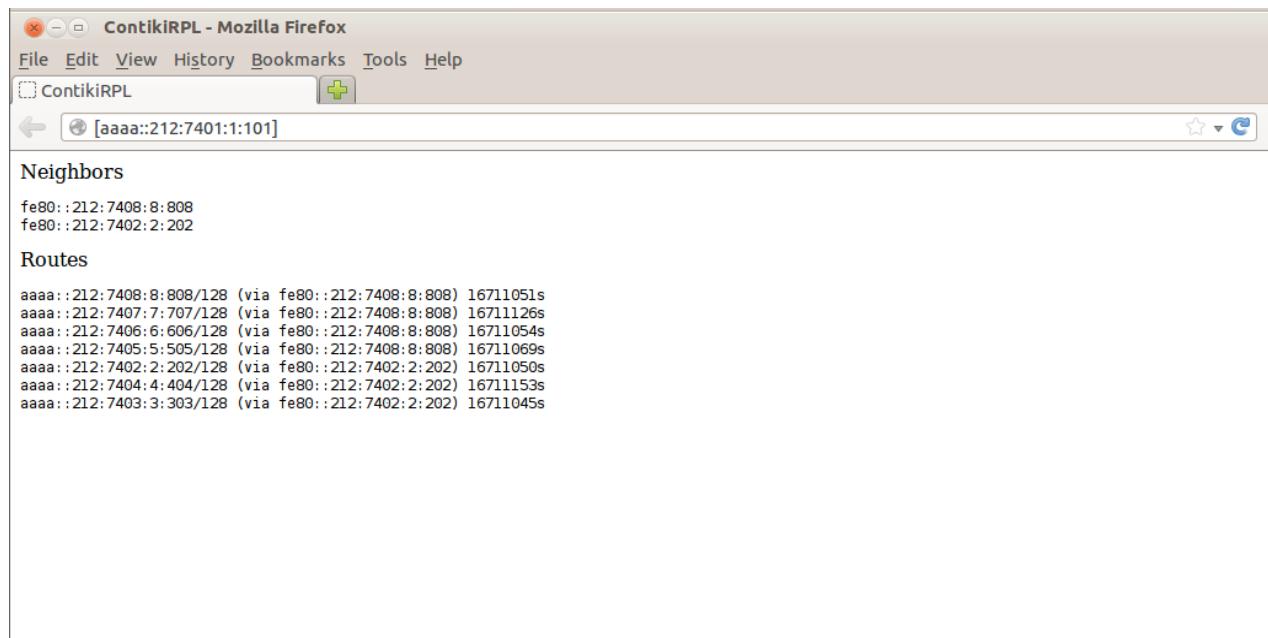
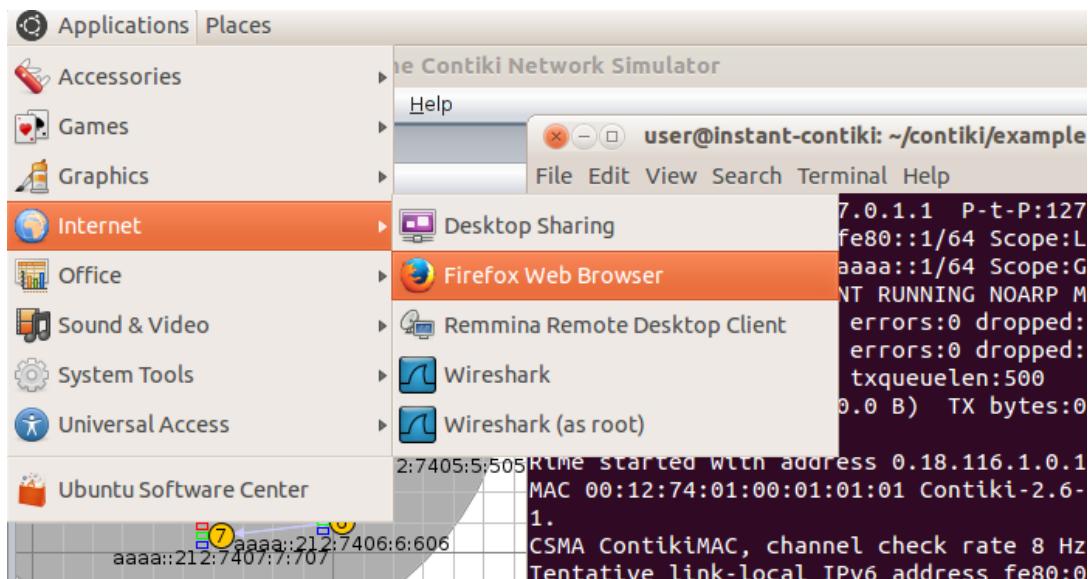
Etape 6 :

De là, dans le simulateur cooja, on démarre le test par clic sur start

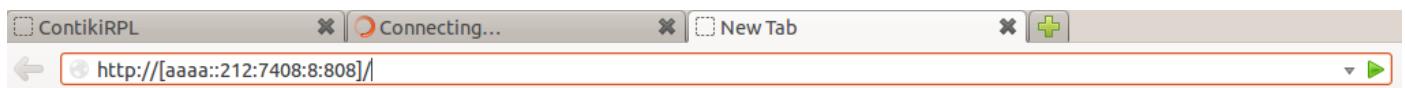


Etape 8 :

Sur le terminal qu'on a ouvert, s'affiche l'adresse IPV6 du serveur qui est généralement : (aaaa ::212 :7402 :1 :101). Cette dernière adresse est copiée, puis insérée dans la barre d'adresse de l'explorateur Firefox.



Ces adresses affichées sont celles des capteurs qu'on peut sélectionner pour y accéder via le navigateur Firefox par le Protocole http



Qui nous donne les capteurs : light et température

Comme dans les exemples suivants :

Capteur numéro 8 :

The screenshot shows a web browser window with two tabs: "ContikiRPL" and "Contiki Web Sense". The address bar shows "[aaaa::212:7408:8:808]". The main content area displays the title "Current readings" and the following data:
Light: 242
Temperature: 24° C

Capteur numéro 7 :

The screenshot shows a web browser window with a single tab. The address bar shows "[aaaa::212:7407:7:707]". The main content area displays the title "Current readings" and the following data:
Light: 211
Temperature: 24° C

Capteur numéro 5 :

The screenshot shows a web browser window with a single tab. The address bar shows "[aaaa::212:7405:5:505]". The main content area displays the title "Current readings" and the following data:
Light: 282
Temperature: 24° C

Chapitre III : faire une étude comparative

Généralement dans IOT les protocoles se diffèrent selon plusieurs critères : temps utilisé, la bande passante, le débit et autres caractéristiques, ce qui met la différence dans le choix du meilleur protocole à l'utiliser dans la mise en place réelle des objets connectés. Dans le but de mieux comparer, entre elles, les topologies qu'on a créées, l'application WireShark est l'outil Ideal. Il nous donnera les détails de transmission des paquets durant une seule communication et par la possibilité du choix du meilleur protocole à appliquer.

MQTT :

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	49074 → 1880 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK...
2	0.000031065	127.0.0.1	127.0.0.1	TCP	74	1880 → 49074 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=...
3	0.000058404	127.0.0.1	127.0.0.1	TCP	66	49074 → 1880 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=381...
4	0.001419812	127.0.0.1	127.0.0.1	HTTP	464	POST /inject/f4b05444.ca1f68 HTTP/1.1
5	0.001446870	127.0.0.1	127.0.0.1	TCP	66	1880 → 49074 [ACK] Seq=1 Ack=399 Win=65152 Len=0 TSval=3...
6	0.004937096	127.0.0.1	127.0.0.1	HTTP	384	HTTP/1.1 200 OK (text/plain)
7	0.006273851	127.0.0.1	127.0.0.1	TCP	66	49074 → 1880 [ACK] Seq=399 Ack=239 Win=65408 Len=0 TSval...
8	0.012026223	127.0.0.1	127.0.0.1	TCP	86	38438 → 11883 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=20 TSva...
9	0.012053387	127.0.0.1	127.0.0.1	TCP	66	11883 → 38438 [ACK] Seq=1 Ack=21 Win=512 Len=0 TSval=381...
10	0.016322249	127.0.0.1	127.0.0.1	TCP	86	11883 → 38438 [PSH, ACK] Seq=1 Ack=21 Win=512 Len=20 TSv...
11	0.017001573	127.0.0.1	127.0.0.1	TCP	70	11883 → 38438 [PSH, ACK] Seq=21 Ack=21 Win=512 Len=4 TSv...
12	0.017183455	127.0.0.1	127.0.0.1	TCP	66	38438 → 11883 [ACK] Seq=21 Ack=25 Win=512 Len=0 TSval=38...
13	0.018040209	127.0.0.1	127.0.0.1	TCP	70	38438 → 11883 [PSH, ACK] Seq=21 Ack=25 Win=512 Len=4 TSv...
14	0.018052997	127.0.0.1	127.0.0.1	TCP	66	11883 → 38438 [ACK] Seq=25 Ack=25 Win=512 Len=0 TSval=38...
15	0.062173794	127.0.0.1	127.0.0.1	TCP	277	1880 → 49046 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=211 TSva...
16	0.062196649	127.0.0.1	127.0.0.1	TCP	66	49046 → 1880 [ACK] Seq=1 Ack=212 Win=507 Len=0 TSval=381...

No.	Time	Source	Destination	Protocol	Length	Info
9	0.012053387	127.0.0.1	127.0.0.1	TCP	66	11883 → 38438 [ACK] Seq=1 Ack=21 Win=512 Len=0 TSval=381...
10	0.016322249	127.0.0.1	127.0.0.1	TCP	86	11883 → 38438 [PSH, ACK] Seq=1 Ack=21 Win=512 Len=20 TSv...
11	0.017001573	127.0.0.1	127.0.0.1	TCP	70	11883 → 38438 [PSH, ACK] Seq=21 Ack=21 Win=512 Len=4 TSv...
12	0.017183455	127.0.0.1	127.0.0.1	TCP	66	38438 → 11883 [ACK] Seq=21 Ack=25 Win=512 Len=0 TSval=38...
13	0.018040209	127.0.0.1	127.0.0.1	TCP	70	38438 → 11883 [PSH, ACK] Seq=21 Ack=25 Win=512 Len=4 TSv...
14	0.018052997	127.0.0.1	127.0.0.1	TCP	66	11883 → 38438 [ACK] Seq=25 Ack=25 Win=512 Len=0 TSval=38...
15	0.062173794	127.0.0.1	127.0.0.1	TCP	277	1880 → 49046 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=211 TSva...
16	0.062196649	127.0.0.1	127.0.0.1	TCP	66	49046 → 1880 [ACK] Seq=1 Ack=212 Win=507 Len=0 TSval=381...

Conclusion

Dans un monde « hyper connecté » via des objets connectés où les usagers sont à la fois émetteurs et récepteurs des données, l'IOT ouvre des champs nouveaux à explorer pour les sciences de l'information et de la communication pour étudier, d'une part, les enjeux sociaux de ces nouveaux bouleversements technologiques et numériques, et pour d'autre part, analyser si les objets connectés répondent à des besoins d'usagers de plus en plus exigeants en matière de service, de communication et d'information. L'Internet des objets doit être traité sous deux aspects. Le premier aspect concerne la réalité industrielle et technologique des objets connectés, comme la gestion de l'entreprise, l'e-administration, l'e-gouvernement, mais aussi la gestuelle : podomètre, direction du regard, GPS, etc. Le second aspect concerne les impacts des objets connectés dans la vie quotidienne dans les domaines suivant : la santé, l'habitat, l'automobile, l'assurance, etc.

Bibliographie

<https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%202.7/>

<https://nodered.org/>

<https://blog.engineering.publicissapient.fr/2018/04/16/internet-des-objets-quels-protocoles-applicatifs-utiliser-1-2/>

https://iot.goffinet.org/iot_protocoles.html