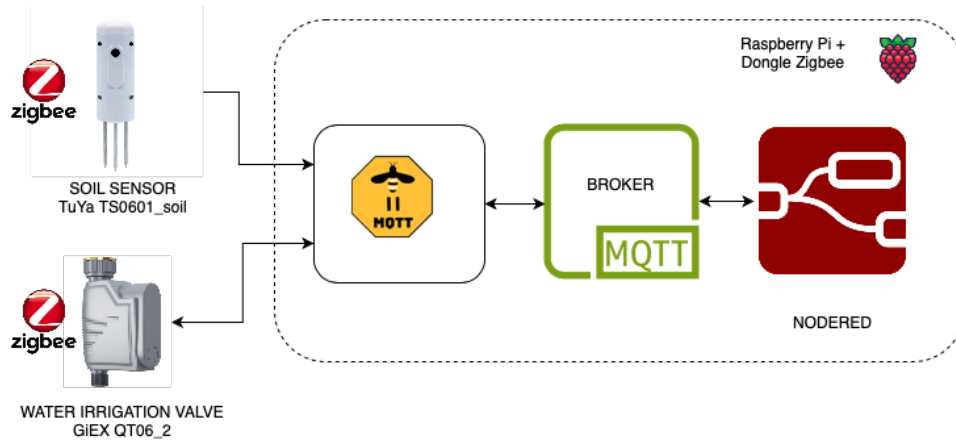


TP1 ZIGBEE

SYSTEME D'IRRIGATION INTÉRIEURE



1. MATERIEL

- 1 raspberry pi
- 1 dongle USB/zigbee
 - sonoff zigbee 3.0 (<https://amz.run/6LG0>)
 - Ou dongle CC2531 Zigbee Clé USB + **firmware pour OpenHAB ioBroker FHEM zigbee2mqtt** avec antenne SMA Boîtier Noir (<https://amz.run/6LFw>)
- Un capteur d'humidité du sol TuYa 0601_soil
- Une électrovanne d'irrigation GiEX QT06_2

2. INSTALLATION SUR RASPBERRY PI

2.1. Installation de node-red : ouvrir une console sur le raspberry pi et saisir la commande

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

Cela installe nodejs, npm et node-red.

Si node-red a été correctement installé, nodejs et npm le sont aussi. On peut vérifier avec les commandes

```
# Verify that the correct nodejs and npm (automatically installed with nodejs)
# version has been installed
node --version # Should output v18.X or more
```

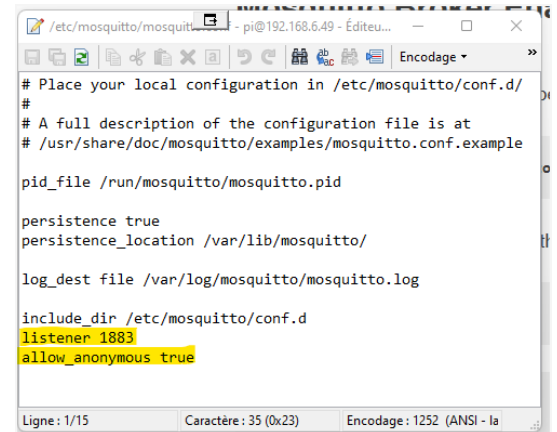
```
npm --version # Should output 10.X or more
```

2.2. Installation du broker mosquitto :

```
sudo apt update
sudo apt install -y mosquitto mosquitto-
clients
sudo systemctl enable mosquitto.service
```

Éditer le fichier `/etc/mosquitto/mosquitto.conf` et ajouter les 2 lignes suivantes à la fin :

```
listener 1883
allow_anonymous true
```



Redémarrer le service mosquitto :

```
sudo systemctl restart mosquitto
```

2.3. Installation de zigbee2mqtt

2.3.1 Détermination du port USB du dongle zigbee

Connecter le dongle ZigBee et déterminer le port série.

```
ls -l /dev/serial/by-id
```

exemple de réponse avec un dongle CC2531

```
total 0
lrwxrwxrwx. 1 root root 13 Oct 19 19:26 usb-
Texas_Instruments_TI_CC2531_USB_CDC_0X00124B0018ED3DDF-if00 -> ../../ttyACM0
dans ce cas le dongle sera localisé à /dev/ttyACM0
```

exemple de réponse avec un dongle sonoff ZigBee 3.0

```
total 0
lrwxrwxrwx 1 root root 13 26 janv. 16:17 usb-
Silicon_Labs_Sonoff_Zigbee_3.0_USB_Dongle_Plus_0001-if00-port0 -> ../../ttyUSB0
Dans ce cas le dongle sera localisé à /dev/ttyUSB0
```

2.3.2 Installation de zigbee2mqtt

Création du dossier d'installation pour zigbee2mqtt et affectation de l'utilisateur pi comme propriétaire

```
sudo mkdir /opt/zigbee2mqtt
sudo chown -R ${USER}: /opt/zigbee2mqtt
```

Clonage du dépôt Zigbee2Mqtt

```
git clone --depth 1 https://github.com/Koenkk/zigbee2mqtt.git
/opt/zigbee2mqtt
```

Installation de dépendances

```
cd /opt/zigbee2mqtt
npm ci
```

À la fin cela indique combien de packages ont été installés.

Éditer le fichier de configuration (avec nano ou autre)

Il est localisé ici : **/opt/zigbee2mqtt/data/configuration.yaml**

```
homeassistant: false
permit_join: true
mqtt:
  base_topic: zigbee2mqtt
  server: mqtt://localhost
serial:
  # emplacement du dongle Zigbee
  port: /dev/ttyACM0 # ou /dev/ttyUSB0
advanced:
  network_key: GENERATE
frontend: true
```

Source : https://github.com/bouhenic/FormationIOT/blob/main/journée2/TP1Zigbee/Configuration_Tp1.yaml

Lancement de zigbee2mqtt :

```
cd /opt/zigbee2mqtt
npm start
```

Si le démarrage se passe sans encombre on voit ce type de messages dans la console :

```
info 2023-01-27 16:27:24: Logging to console and directory:
'/opt/zigbee2mqtt/data/log/2023-01-27.16-27-24' filename: log.txt
info 2023-01-27 16:27:24: Starting Zigbee2MQTT version 1.29.2 (commit #1402139)
info 2023-01-27 16:27:24: Starting zigbee-herdsman (0.14.83-hotfix.0)
info 2023-01-27 16:27:30: zigbee-herdsman started (resumed)
info 2023-01-27 16:27:30: Coordinator firmware version:
'{"meta":{"maintrel":3,"majorrel":2,"minorrel":6,"product":0,"revision":20211115,"transportrev":2},"type":"zStack12"}'
```

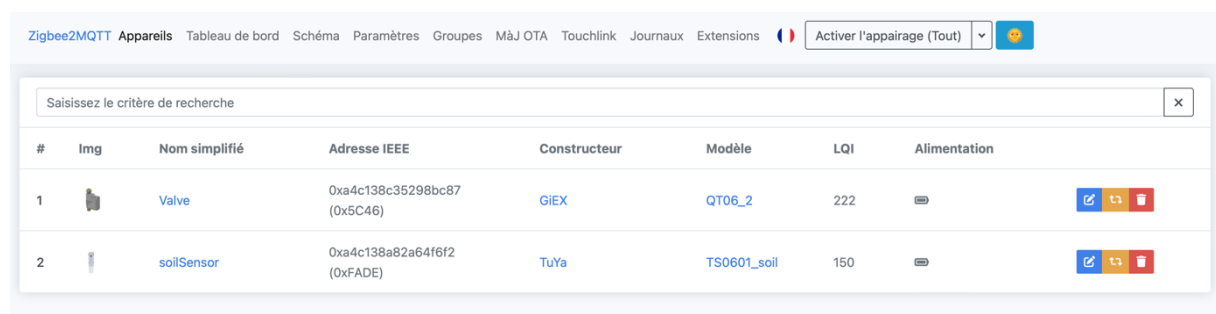
Pour un démarrage comme un service avec systemctl voir ici :

https://www.zigbee2mqtt.io/guide/installation/01_linux.html#optional-running-as-a-daemon-with-systemctl





3 SERVEUR WEB DE CONFIGURATION (FRONTEND)


Si zigbee2mqtt est lancé, il est accessible à l'adresse du Raspberry sur le port 8080 :

http://<IP_RASPBERRY>:8080



The screenshot shows the Zigbee2MQTT web interface. At the top, there's a navigation bar with links: Appareils, Tableau de bord, Schéma, Paramètres, Groupes, MâJ OTA, Touchlink, Journaux, Extensions, and a language selector (FR). There's also a button to 'Activer l'appairage (Tout)'. Below the navigation bar is a search bar labeled 'Saisissez le critère de recherche'. The main content area displays a table of devices with the following columns: #, Img, Nom simplifié, Adresse IEEE, Constructeur, Modèle, LQI, and Alimentation. Two devices are listed: 1. Valve (GIEX, QT06_2, LQI 222) and 2. soilSensor (TuYa, TS0601_soil, LQI 150). Each device row has a small icon, a simplified name, the IEEE address, manufacturer, model, LQI value, and a battery status icon. There are also action buttons (edit, delete, etc.) for each device.

#	Img	Nom simplifié	Adresse IEEE	Constructeur	Modèle	LQI	Alimentation
1		Valve	0xa4c138c35298bc87 (0x5C46)	GIEX	QT06_2	222	
2		soilSensor	0xa4c138a82a64f6f2 (0xFADE)	TuYa	TS0601_soil	150	

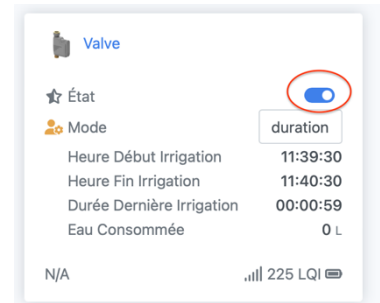
Pour découvrir un nouveau dispositif il suffit de cliquer sur **Activer l'appairage (tout)** et de mettre le dispositif à inscrire en mode découverte. Il apparait avec son adresse IEEE. On peut le renommer en cliquant sur .

Il est possible de piloter les dispositifs depuis cette page web en choisissant l'onglet **Tableau de bord**.

Chaque fois qu'une action est déclenchée sur un dispositif, les données sont publiées sur le broker.

Par exemple :

Le fait de cliquer sur l'interrupteur d'état pour le dispositif nommé Valve génère le message suivant dans le fichier log.



```
Zigbee2MQTT:info 2024-02-17 11:42:07: MQTT publish: topic 'zigbee2mqtt/Valve',  
payload '{"battery":100,"cycle_irrigation_interval":0,"cycle_irrigation_num_times":0,"irrigation_end_time":"11:43:03","irrigation_start_time":"11:42:03","irrigation_target":60,"last_irrigation_duration":"00:01:00","linkquality":222,"mode":"duration","state":"ON","water_consumed":0}'
```

Ce qui montre qu'un message a été publié sur le topic **zigbee2mqtt/Valve**

Le message est un fichier Json qui une fois mis en forme donne

```
{  
  "battery": 100,  
  "cycle_irrigation_interval": 0,  
  "cycle_irrigation_num_times": 0,  
  "irrigation_end_time": "11:40:30",  
  "irrigation_start_time": "11:39:30",  
  "irrigation_target": 60,  
  "last_irrigation_duration": "00:01:00",  
  "linkquality": 222,  
  "mode": "duration",  
  "state": "ON",  
  "water_consumed": 0  
}
```

Il s'agit d'un objet qui indique l'état de la batterie, le mode de fonctionnement (durée ou capacité, la durée d'arrosage, la dernière durée d'arrosage, la qualité du lien (qualité de la réception radio) et enfin l'état de l'électrovanne : ON ce qui signifie qu'elle est en arrosage. On peut suivre le journal des événements en cliquant sur l'onglet **Journaux**.

```
Info 2024-02-17 11:50:46 MQTT publish: topic 'zigbee2mqtt/Valve', payload  
'{"battery":100,"cycle_irrigation_interval":0,"cycle_irrigation_num_times":0,"irrigation_end_time":"11:51:14","irrigation_start_time":"11:50:14","irrigation_target":60,"last_irrigation_duration":"00:01:00","linkquality":225,"mode":"duration","state":"ON","water_consumed":0}'
```

4 PILOTAGE PAR MQTT

Fonctions exposées

Pour chaque dispositif, il faut se rendre sur la page de documentation qui lui est associée. Par exemple pour une électrovanne de la marque GiEx, la documentation donne :

- Battery (numéric) : Batterie restante en %, peut prendre jusqu'à 24 heures avant d'être rapportée. La valeur peut être trouvée dans l'état publié sur la propriété de la batterie. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur. La valeur minimale est 0 et la valeur maximale est 100. L'unité de cette valeur est le %.
- State (binary) : État. La valeur peut être trouvée dans l'état publié sur la propriété d'état. Il n'est pas possible de lire (/get) cette valeur. Pour écrire (/set) une valeur, publiez un message sur le sujet zigbee2mqtt/NOM_AMICAL/set avec le payload {"state": NOUVELLE_VALEUR}. Si la valeur est égale à ON, l'état est ON, si OFF, OFF.
- Mode (énum) : Mode d'irrigation. La valeur peut être trouvée dans l'état publié sur la propriété de mode. Il n'est pas possible de lire (/get) cette valeur. Pour écrire (/set) une valeur, publiez un message sur le sujet zigbee2mqtt/NOM_AMICAL/set avec le payload {"mode": NOUVELLE_VALEUR}. Les valeurs possibles sont : duration, capacity.
- Cycle irrigation num times (numeric) : Nombre de fois que l'irrigation en cycle se produit, réglé sur 0 pour un cycle unique. La valeur peut être trouvée dans l'état publié sur la propriété cycle_irrigation_num_times. Il n'est pas possible de lire (/get) cette valeur. Pour écrire (/set) une valeur, publiez un message sur le sujet zigbee2mqtt/NOM_AMICAL/set avec le payload {"cycle_irrigation_num_times": NOUVELLE_VALEUR}. La valeur minimale est 0 et la valeur maximale est 100.
- Irrigation start time (numeric) : Heure de début de la dernière irrigation. La valeur peut être trouvée dans l'état publié sur la propriété irrigation_start_time. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur.
- Irrigation end time (numeric) : Heure de fin de la dernière irrigation. La valeur peut être trouvée dans l'état publié sur la propriété irrigation_end_time. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur.
- Last irrigation duration (numeric) : Durée de la dernière irrigation. La valeur peut être trouvée dans l'état publié sur la propriété last_irrigation_duration. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur.
- Water consumed (numeric) : Consommation d'eau de la dernière irrigation. La valeur peut être trouvée dans l'état publié sur la propriété water_consumed. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur. L'unité de cette valeur est le litre.
- Irrigation target (numeric) : Objectif d'irrigation, durée en secondes ou capacité en litres (selon le mode), réglé sur 0 pour laisser la vanne ouverte indéfiniment, pour des raisons de sécurité, l'objectif sera forcé à un minimum de 10 secondes en mode durée. La valeur peut être trouvée dans l'état publié sur la propriété irrigation_target. Il n'est pas possible de lire (/get) cette valeur. Pour écrire (/set) une valeur, publiez un message sur le sujet zigbee2mqtt/NOM_AMICAL/set avec le payload {"irrigation_target": NOUVELLE_VALEUR}. La valeur minimale est 0 et la valeur maximale est 43200. L'unité de cette valeur est secondes ou litres.
- Cycle irrigation interval (numeric) : Intervalle entre les cycles d'irrigation. La valeur peut être trouvée dans l'état publié sur la propriété cycle_irrigation_interval. Il n'est pas possible de lire (/get) cette valeur. Pour écrire (/set) une valeur, publiez un message sur le sujet zigbee2mqtt/NOM_AMICAL/set avec le payload {"cycle_irrigation_interval": NOUVELLE_VALEUR}. La valeur minimale est 0 et la valeur maximale est 43200. L'unité de cette valeur est sec.
- Linkquality (numeric) : Qualité du lien (force du signal). La valeur peut être trouvée dans l'état publié sur la propriété linkquality. Il n'est pas possible de lire (/get) ou d'écrire (/set) cette valeur. La valeur minimale est 0 et la valeur maximale est 255. L'unité de cette valeur est lqi.

Test de mqtt avec mosquitto :

Commande et configuration de l'électrovanne :

- State (binary) :

Le state ne peut pas être lu avec /get. P Peut être lu avec un subscribe sur le topic zigbee2mqtt/Nom_amical après une publication de valeur.

```
Ex : mosquitto_sub -h localhost -t zigbee2mqtt/Valve
```

L'état de l'électrovanne peut être activée avec :

```
mosquitto_pub -h localhost -t zigbee2mqtt/Valve/set -m  
'{"state":"OFF"}'
```

- Mode (enum) :

C'est le mode de fonctionnement (durée d'arrosage ou quantité d'arrosage). Ne peut pas être lu avec /get. Peut être lu avec un subscribe sur le topic zigbee2mqtt/Valve.

```
mosquitto_sub -h localhost -t zigbee2mqtt/Valve
```

le mode peut être modifié par :

```
mosquitto_pub -h localhost -t zigbee2mqtt/Valve/set -m  
'{"mode":"duration"}'
```

- Irrigation target (numeric) :

Permet le réglage de la durée d'arrosage et de la quantité d'eau à arroser.

La valeur minimale est 0 et la valeur maximale est 43200. L'unité de cette valeur est en secondes ou en litres.

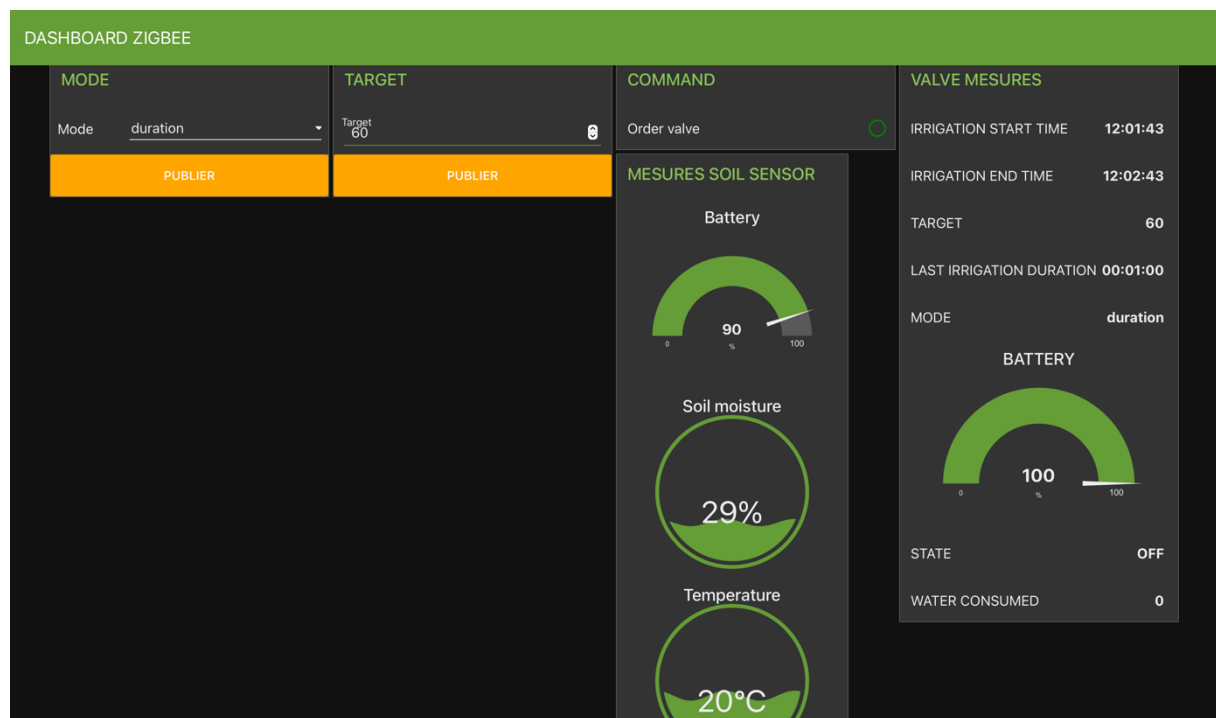
```
mosquitto_pub -h localhost -t zigbee2mqtt/Valve/set -m  
'{"irrigation_target":60}'
```

Lecture des mesures issues du capteur d'humidité :

Ne peut pas être lu avec /get. Le capteur publie les valeurs à chaque nouvelle mesure. Il suffit de s'abonner au topic : **zigbee2mqtt/soilSensor**.

```
mosquitto_sub -h localhost -t zigbee2mqtt/soilSensor
```

5 UTILISATION DE NODE-RED



L'objectif est de concevoir le dashoard ci-dessus :

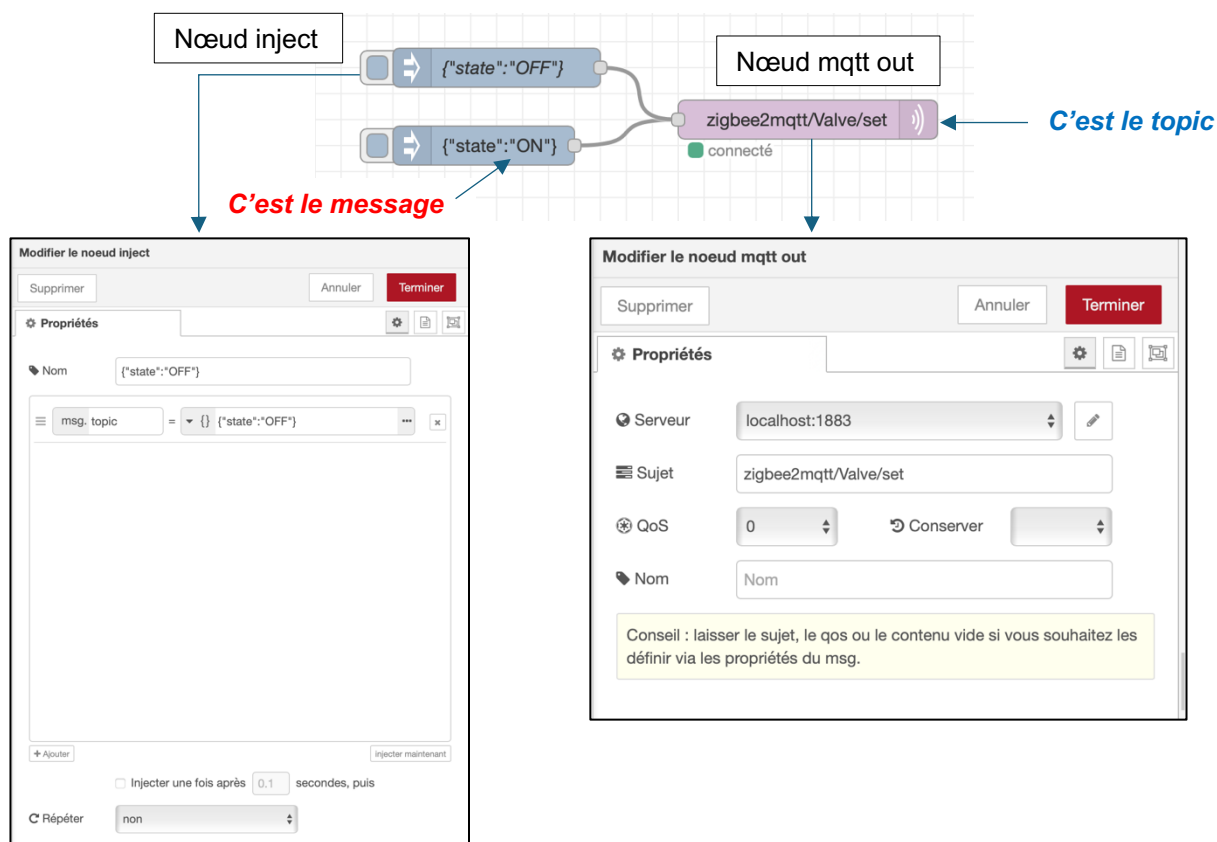
Côté capteur, on affiche :

- Le niveau de la batterie.
- L'humidité du sol.
- La température.

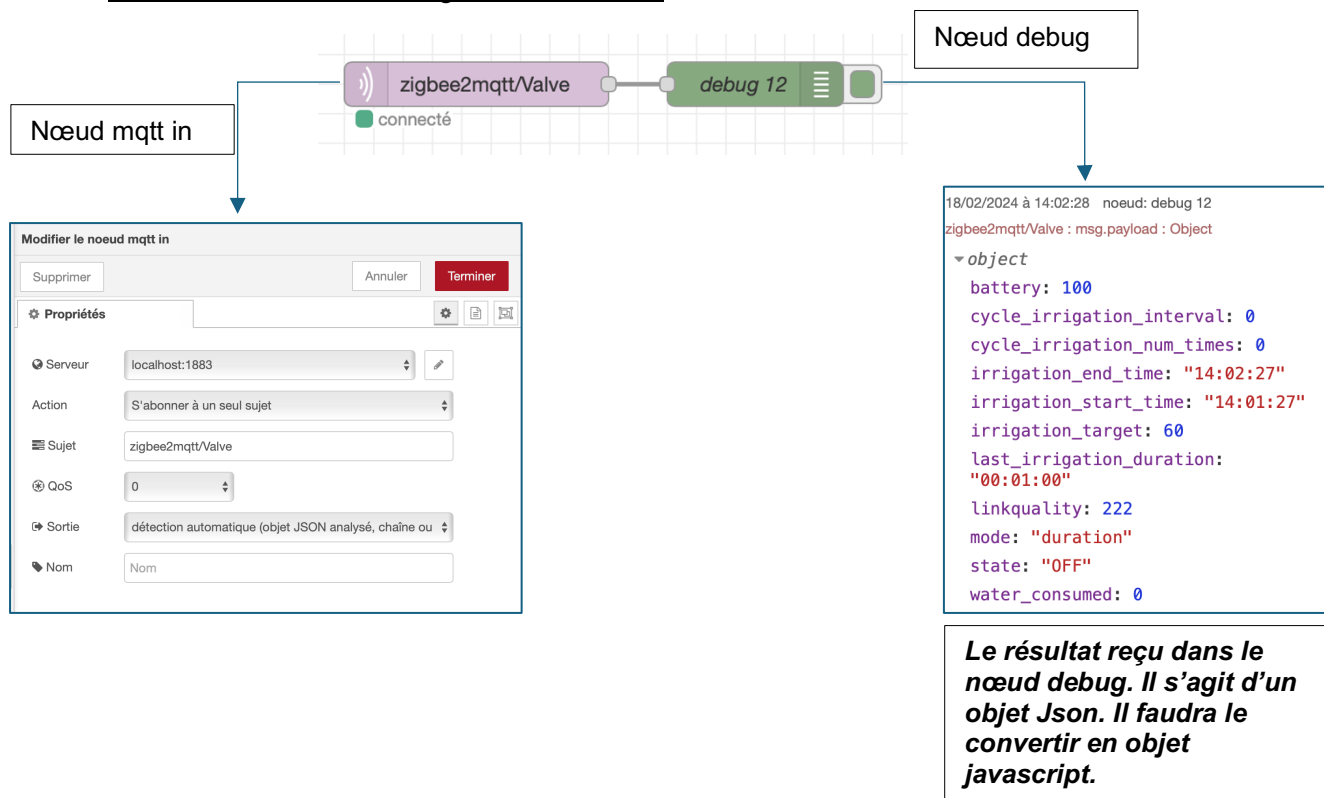
Côté électrovanne, on affiche :

- Le niveau de la batterie.
- L'état de l'électrovanne.
- L'eau consommée.
- Le mode de fonctionnement (durée ou quantité).
- La durée d'arrosage.
- Les temps de début et de fin du dernier arrosage.
- Un switch déclenchera l'arrosage.
- Un dropdown associé à un bouton permettront le choix du mode.
- Un input text et un bouton permettront la saisie de la durée d'arrosage ou de la quantité d'eau.

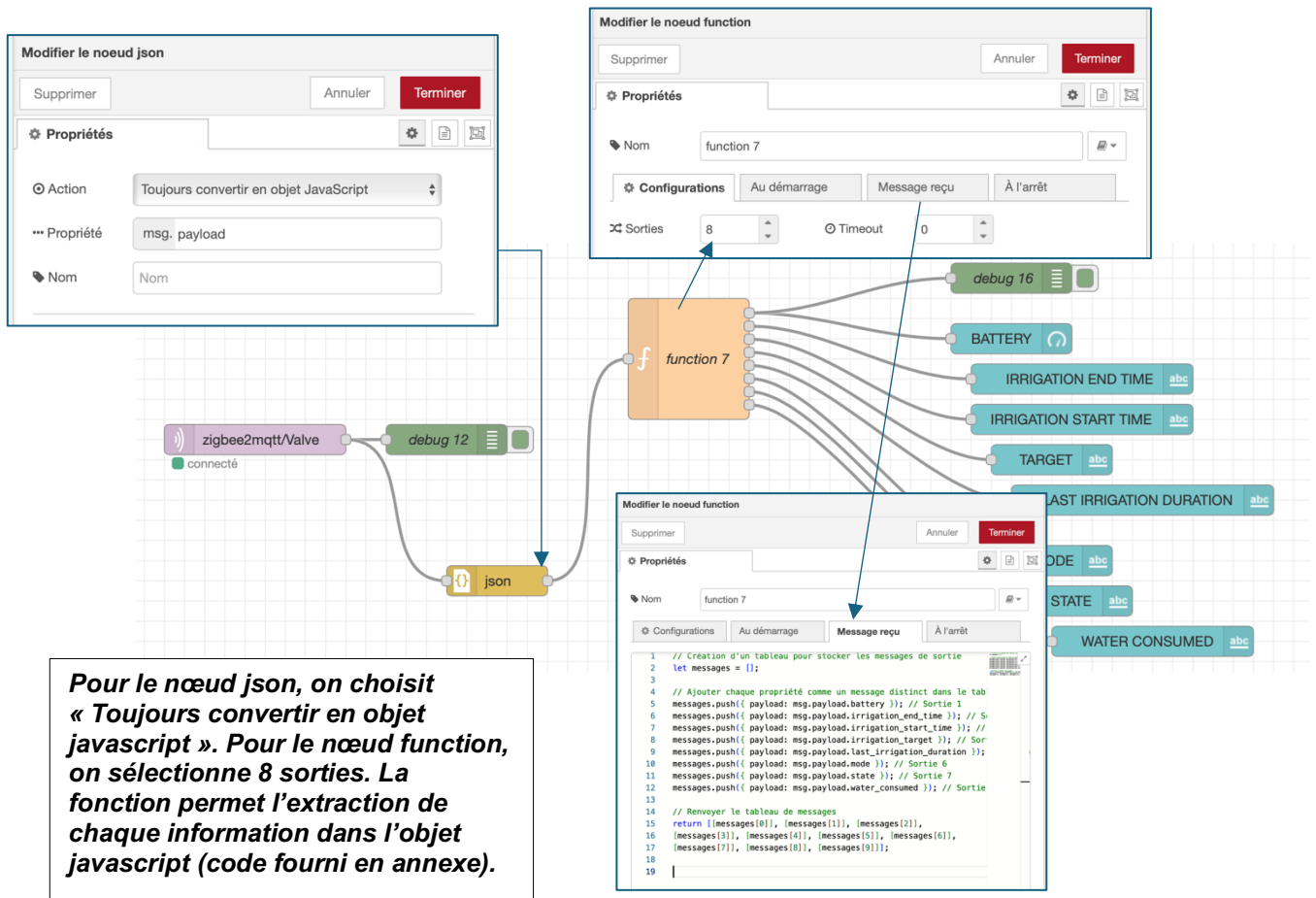
Test du publish, commande de l'électrovanne :



Test du subscribe, affichage des mesures :

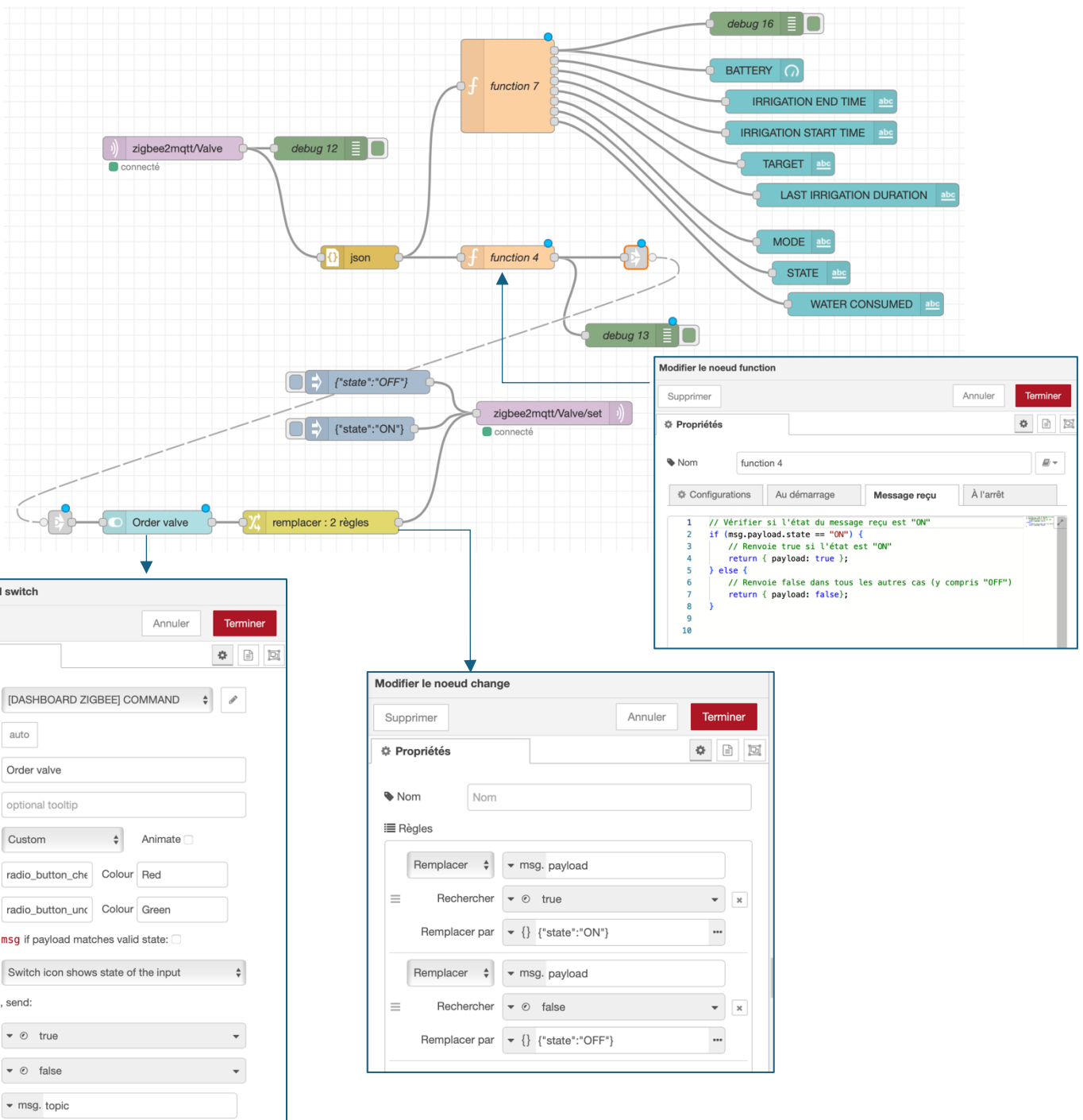


On complète avec un nœud json, un nœud function et des nœuds dashboard.



Source function7 :
<https://github.com/bouhenic/FormationIOT/blob/main/journée2/TP1Zigbee/function7.js>

Ajout de la commande de l'électrovanne depuis le dashboard



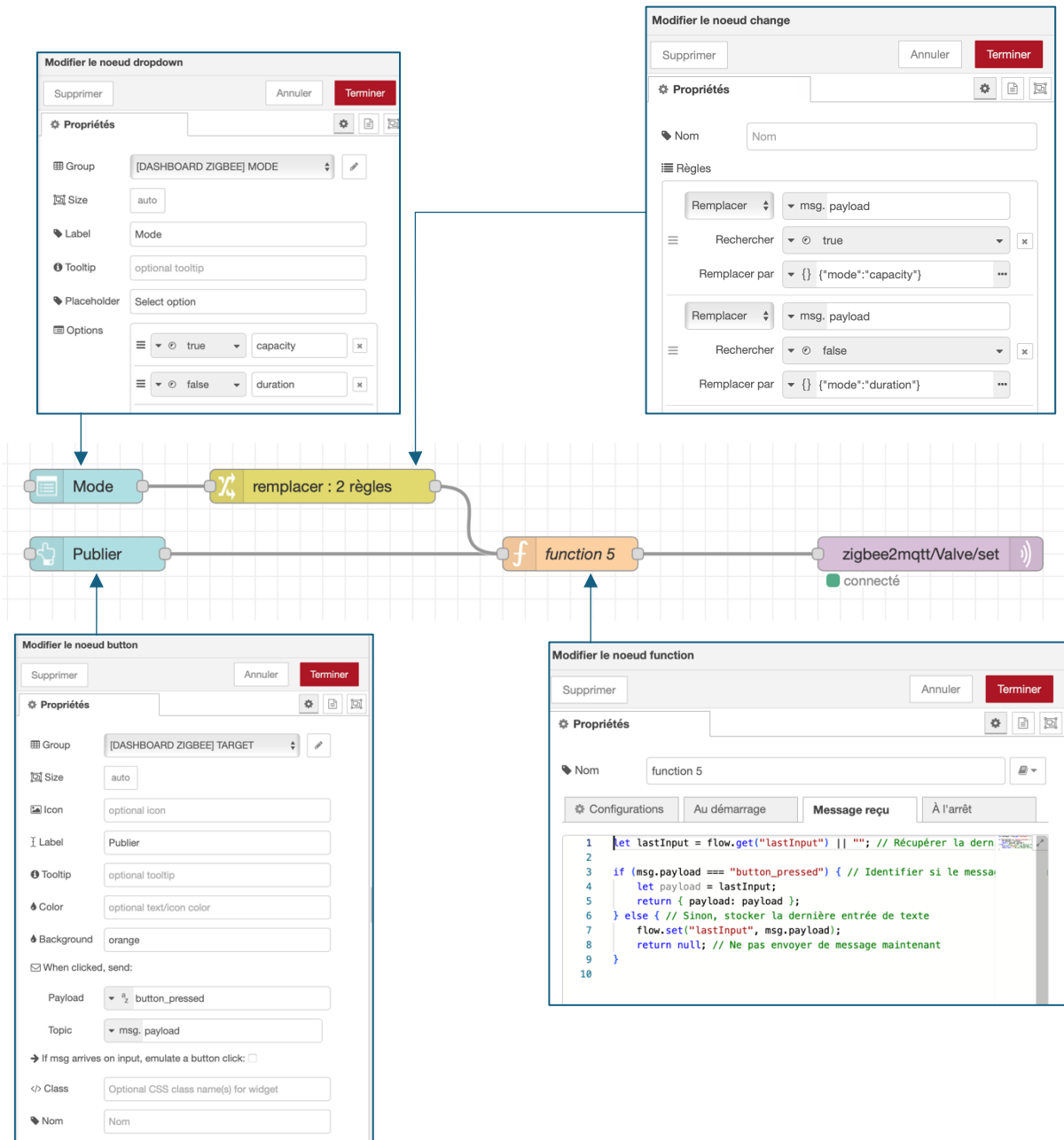
L'objectif ici est de remplacer la commande de l'électrovanne par un switch. Le switch renvoie une information sur son état, c'est la raison pour laquelle on récupère l'état de l'électrovanne depuis le mqtt en subscribe. La fonction « fonction4 » met en forme le payload pour le switch. Le nœud « change » crée les messages de commande de l'électrovanne.

Source function4 :

<https://github.com/bouhenic/FormationIoT/blob/main/journée2/TP1Zigbee/function4.js>

Choix et publication du mode (duration/capacity) sur la dashboard

L'ensemble des nœuds suivants va permettre la publication du mode choisi (duration ou capacity). On utilisera un nœud « dropdown » et un bouton de publication.



Source function5 :

<https://github.com/bouhenic/FormationIOT/blob/main/journée2/TP1Zigbee/function5.js>

L'ensemble des nœuds suivants permettra la publication de la valeur de réglage de l'arrosage. Tout dépend si le mode choisi est « duration » ou « capacity ». Dans le cas de duration, l'unité est la seconde. Dans le cas de capacity, l'unité est le litre.

Modifier le nœud text input

Supprimer Annuler Terminer

Propriétés

Group: [DASHBOARD ZIGBEE] TARGET

Size: auto

Label: Target

Tooltip: optional tooltip

Mode: number Delay (ms): 300

→ If msg arrives on input, pass through to output: ☐

Send value on focus leave: ☐

When changed, send:

Payload: Current value

Topic: msg. payload

Class: Optional CSS class name(s) for widget

Name:

Setting Delay to 0 waits for Enter or Tab key, to send input.

Modifier le nœud function

Supprimer Annuler Terminer

Propriétés

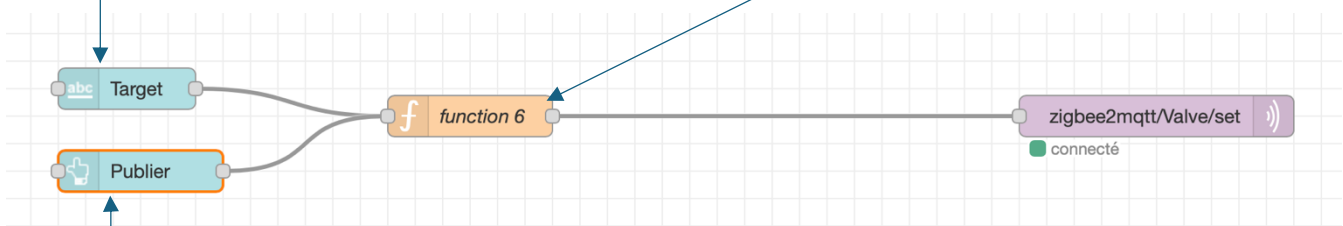
Nom: fonction 6

Configurations: Au démarrage Message reçu À l'arrêt

```

1 let lastInput2 = flow.get("lastInput2") || ""; // Récupérer la de
2
3 if (msg.payload === "button_pressed") { // Identifier si le messa
4 // Utiliser la valeur stockée comme irrigation_target
5 let irrigationTarget = lastInput2;
6 // Créer et retourner un objet avec la propriété irrigation_t
7 return { payload: { "irrigation_target": irrigationTarget } };
8 } else { // Sinon, stocker la dernière entrée de texte
9 flow.set("lastInput2", msg.payload);
10 return null; // Ne pas envoyer de message maintenant
11 }
12

```



Modifier le nœud button

Supprimer Annuler Terminer

Propriétés

Group: [DASHBOARD ZIGBEE] TARGET

Size: auto

Icon: optional icon

Label: Publier

Tooltip: optional tooltip

Color: optional text/icon color

Background: orange

When clicked, send:

Payload: button_pressed

Topic: msg. payload

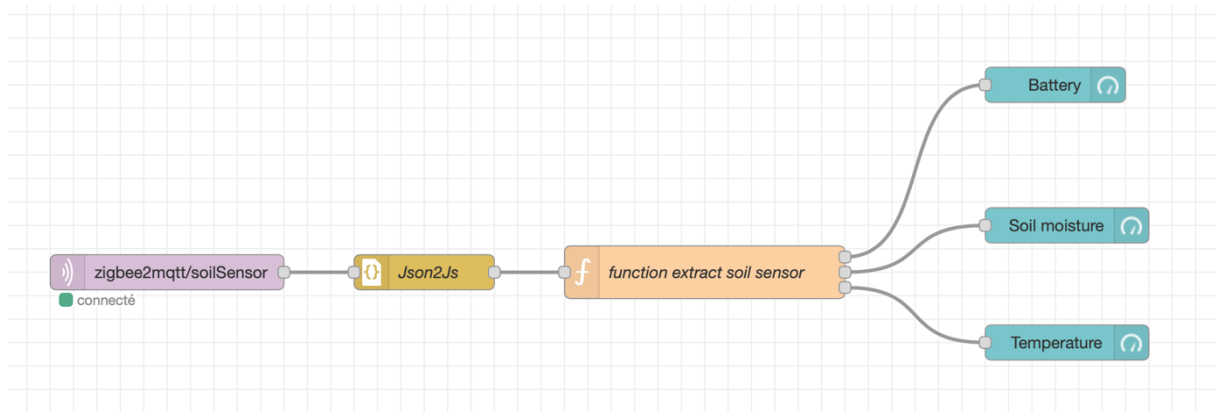
→ If msg arrives on input, emulate a button click: ☐

Class: Optional CSS class name(s) for widget

Nom: Nom

Source :
<https://github.com/bouhenic/FormationIOT/blob/main/journée2/TP1Zigbee/function6.js>

L'ensemble des nœuds suivants permet l'abonnement au topic du capteur d'humidité du sol et l'affichage des mesure sur le dashboard.



Source :

<https://github.com/bouhenic/FormationIOT/blob/main/journée2/TP1Zigbee/extractSoilSensor.js>

EXEMPLE DE FICHIER DE CONFIGURATION.YAML

```
homeassistant: false
permit_join: false
mqtt:
  base_topic: zigbee2mqtt
  server: mqtt://localhost
serial:
  port: /dev/ttyUSB0
frontend: true
advanced:
  log_output:
    - console
# mettre 675X ou X est le numéro du groupe
pan_id: 6755
device_options:
  legacy: false
blocklist:
#01
  - '0xa4c138c35298bc87'
  - '0xa4c138a82a64f6f2'
#02
  - '0xa4c1381520886a8c'
  - '0xa4c138fcfd1ab05'
#03
  - '0x90fd9ffffe14c33d'
  - '0x000d6ffffe16ec06'
  - '0xa4c138de63548d15'
#04
  - '0xd0cf5efffe18cb8c'
  - '0xa4c1389c73550ed1'
  - '0x90fd9ffffe193cfc'
devices: {}
```

CODE DES FONCTIONS :

Function 7 :

```
// Création d'un tableau pour stocker les messages de sortie
let messages = [];

// Ajouter chaque propriété comme un message distinct dans le tableau
messages.push({ payload: msg.payload.battery }); // Sortie 1
messages.push({ payload: msg.payload.irrigation_end_time }); // Sortie 2
messages.push({ payload: msg.payload.irrigation_start_time }); // Sortie 3
messages.push({ payload: msg.payload.irrigation_target }); // Sortie 4
messages.push({ payload: msg.payload.last_irrigation_duration }); // Sortie 5
messages.push({ payload: msg.payload.mode }); // Sortie 6
messages.push({ payload: msg.payload.state }); // Sortie 7
messages.push({ payload: msg.payload.water_consumed }); // Sortie 8

// Renvoyer le tableau de messages
return [[messages[0]], [messages[1]], [messages[2]],
[messages[3]], [messages[4]], [messages[5]], [messages[6]], [messages[7]], [messages[8]], [messages[9]]];
```

Function 4 :

```
// Vérifier si l'état du message reçu est "ON"
if (msg.payload.state == "ON") {
    // Renvoie true si l'état est "ON"
    return { payload: true };
} else {
    // Renvoie false dans tous les autres cas (y compris "OFF")
    return { payload: false };
}
```

Function 5 :

```
let lastInput = flow.get("lastInput") || ""; // Récupérer la dernière entrée stockée

if (msg.payload === "button_pressed") { // Identifier si le message provient du bouton
    let payload = lastInput;
    return { payload: payload };
} else { // Sinon, stocker la dernière entrée de texte
    flow.set("lastInput", msg.payload);
    return null; // Ne pas envoyer de message maintenant
}
```

Fonction 6 :

```
let lastInput2 = flow.get("lastInput2") || ""; // Récupérer la dernière entrée stockée

if (msg.payload === "button_pressed") { // Identifier si le message provient du bouton
  // Utiliser la valeur stockée comme irrigation_target
  let irrigationTarget = lastInput2;
  // Créer et retourner un objet avec la propriété irrigation_target définie
  return { payload: { "irrigation_target": irrigationTarget } };
} else { // Sinon, stocker la dernière entrée de texte
  flow.set("lastInput2", msg.payload);
  return null; // Ne pas envoyer de message maintenant
}
```

Fonction extract soil sensor :

```
// Création d'un tableau pour stocker les messages de sortie
let messages = [];

// Ajouter chaque propriété comme un message distinct dans le tableau
messages.push({ payload: msg.payload.battery }); // Sortie 1
messages.push({ payload: msg.payload.soil_moisture }); // Sortie 2
messages.push({ payload: msg.payload.temperature }); // Sortie 3

// Renvoyer le tableau de messages
return [[messages[0]], [messages[1]], [messages[2]]];
```

Source :

Formation IOT (François Riotte)