

# **SAE23 : Mettre en place une solution informatique pour l'entreprise**

*Solution Docker*

**Auteurs : Pierre CHAVEROUX, Ange GIUNTINI, Gaspard  
BERSOULLÉ, Sylvio GASPAROTTO**

## Sommaire

<b>Introduction</b>	<b>1</b>
<b>1. Configuration de l'environnement de travail - Docker</b>	<b>2</b>
1.1. Installation de Docker	2
1.2. Docker n°1 - Installation d'un broker de test MQTT	3
1.3. Docker n°2 - Installation de Node-Red	4
1.4. Docker n°3 : Installation de InfluxDB	5
1.5. Docker n°4 - Installation et configuration de Grafana	6
<b>2. Première approche de visualisation des données : Node-Red Dashboard</b>	<b>7</b>
2.1. Installation de Node-Red Dashboard	7
2.2. Noeud 'mqtt in' : Récupération des données de capteurs	8
2.3. Noeud 'function' : Filtrage des données	11
<b>3. Deuxième approche de virtualisation des données : InfluxDB et Grafana</b>	<b>15</b>
3.1. Création de la base de donnée InfluxDB	15
3.2. Implémentation de InfluxDB dans Node-Red	16
3.3. Visualisation des données à l'aide de Grafana Dashboards	17
<b>4. Visualisations et résultats</b>	<b>19</b>
4.1. Flow Node-Red	19
4.2. Node-Red Dashboard	20
4.3. Grafana Dashboard	21
<b>Conclusion</b>	<b>22</b>

## Introduction

Ce rapport a pour but de constituer un guide dans la configuration de l'environnement de travail Docker pour le développement d'une application d'automatisation dans le contexte de la SAE23 à l'IUT de Blagnac. L'objectif de ce projet est de créer un système de collecte, de stockage et de visualisation des données provenant de capteurs en temps réel dans l'IUT de Blagnac.

La configuration de l'environnement de travail est réalisée sur une machine virtuelle Ubuntu, dérivée d'Ubuntu, dans un environnement Docker. Docker est une technologie de conteneurisation qui permet de créer, distribuer et exécuter des applications de manière portable et isolée. Les conteneurs Docker encapsulent les applications et leurs dépendances, offrant ainsi une meilleure utilisation des ressources et facilitant le déploiement des applications.

Ce rapport aura pour but de détailler les différentes étapes de configuration, notamment l'installation de Docker et la mise en place des différents conteneurs Docker nécessaires au projet, tels qu'un broker MQTT, Node-Red, InfluxDB et Grafana. Chaque étape sera expliquée en détail, du téléchargement des images Docker à la configuration des conteneurs.

La première approche de visualisation des données sera réalisée à l'aide de Node-Red Dashboard, un environnement de développement visuel permettant de créer des applications d'automatisation. Nous détaillerons l'installation de Node-Red Dashboard et présenterons les fonctionnalités offertes pour la création de graphiques interactifs.

Ensuite, nous aborderons la deuxième approche de visualisation des données en utilisant InfluxDB, une base de données de séries chronologiques, et Grafana, une plateforme d'analyse et de visualisation des données en temps réel.

Finalement, nous conclurons cet écrit en observant les résultats obtenus via des captures d'écrans des différents outils de visualisation des données des capteurs de l'IUT.

## 1. Configuration de l'environnement de travail - Docker

L'environnement de travail sera la VM disponible sur Webetud. Nous travaillerons sous Lubuntu, un système d'exploitation léger dérivé de Ubuntu. Après l'installation et la création de la machine virtuelle dans VMWare Workstation Pro 16, il est nécessaire de mettre à jour le système d'exploitation avec les commandes suivantes :

```
apt upgrade
apt dist-upgrade
apt autoremove && apt autoclean && apt clean
```

L'exécution des commandes précédentes est nécessaire à la suite de la configuration et à la mise en place des Dockers.

### 1.1. Installation de Docker

Docker est une solution de conteneurisation qui permet de créer, distribuer et exécuter des applications de manière portable et isolée. Les conteneurs Docker encapsulent les applications et leurs dépendances. Docker utilise une technologie de virtualisation légère qui partage le noyau du système d'exploitation hôte, offrant ainsi une meilleure efficacité des ressources. Les conteneurs Docker sont rapides à démarrer et à arrêter, ce qui facilite le déploiement des applications. Nous chercherons donc ici à utiliser cette solution de conteneurisation pour répondre aux besoins du client et satisfaire le cahier des charges.

On commence par installer les paquets d'application Docker avec : '**apt install docker.io**'  
On ajoute l'utilisateur 'pierre' au groupe docker pour faciliter l'exécution des commandes Dockers dans le terminal de commande (plus besoin d'être en mode 'root'). Pour ce faire, il suffit de taper la commande : '**usermod -aG docker pierre**'.

On redémarre la machine pour prendre en compte les paramètres avec la commande '**reboot**'. Il est possible de vérifier la bonne configuration du Docker avec la commande:

**'docker run hello-world'.**

Il s'affiche alors dans le terminal parmi les résultats de la commande la phrase :

*'Hello from Docker !*

*This message shows that our installation appears to be working correctly'*

Cette phrase est le témoin d'une bonne configuration. Le docker créé à partir de l'image hello-world étant inutile à notre projet, nous pouvons le supprimer avec la commande :

```
pierre@SAE23-Docker:~$ docker rm 36245fd6702b
36245fd6702b      → 36245fd6702b étant l'ID du conteneur
```



On peut maintenant commencer l'installation des dockers utiles à notre projet.

## 1.2. Docker n°1 - Installation d'un broker de test MQTT

Un broker MQTT est un serveur intermédiaire dans le protocole MQTT (Message Queuing Telemetry Transport), utilisé pour la communication machine à machine. Il facilite la transmission des messages entre les clients MQTT en agissant en tant que point central. Les clients se connectent au broker pour publier des messages (publish) ou s'abonner à des messages (subscribe). Le broker reçoit les messages publiés et les distribue aux clients abonnés. Ce type de communication est particulièrement adapté aux applications IoT et M2M.

Nous allons donc ici chercher à installer un docker supportant le broker MQTT Mosquitto. Bien que l'on récupérera les données sur le broker de l'iut à l'adresse suivante : **mqtt.iut-blaganc.fr**, nous utiliserons ce broker local à des fins de tests. Pour se faire, on commence par récupérer l'image de mosquitto avec la commande : **'docker pull ansi/mosquitto'**



Nous allons créer un nouveau docker avec l'image fraîchement récupérée :

**docker run -d -p 1883:1883 --name mosquittoRT ansi/mosquitto**

La commande se décompose en plusieurs champs d'arguments :

- -d : Exécute le conteneur en arrière-plan (mode détaché).
- -p 1883:1883 : Mappe le port 1883 de la machine hôte sur le port 1883 du conteneur.
- --name mosquittoRT : Donne le nom "mosquittoRT" au conteneur.

Il ne faut pas oublier de télécharger les paquets clients de Mosquitto, sans quoi la publication et la réception des données partagées sur le broker ne pourra pas être validée. Le téléchargement des paquets s'effectue de cette façon : **'apt install mosquitto-clients'**.

On peut rapidement contrôler le fonctionnement de ce docker en effectuant le test suivant :

- Publication d'une donnée :

**pierre@SAE23-Docker:~\$ mosquitto\_pub -h localhost -p 1883 -t message/SAE23 -m "La SAE23, quel super projet !"**

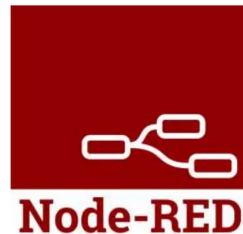
- Réception d'une donnée (en lançant l'écoute du topic avant de publier la donnée):

**pierre@SAE23-Docker:~\$ mosquitto\_sub -h localhost -p 1883 -t message/SAE23**  
**La SAE23, quel super projet !**

Le docker MQTT est donc fonctionnel et nous aidera à faire des tests de récupération de données sur le broker de l'iut.

### 1.3. Docker n°2 - Installation de Node-Red

Node-RED est un environnement de développement visuel utilisé pour créer des applications d'automatisation. Il est très souvent utilisé en IoT car il fournit une architecture de travail basé sur des nœuds qui peuvent être connectés et configurés pour créer des scénarios complexes. L'objectif ici est de créer un docker supportant le service Node-Red.



**A visual tool for wiring the internet of things**

On récupère l'image node-red sur le repository officiel avec :

**docker pull nodered/node-red**

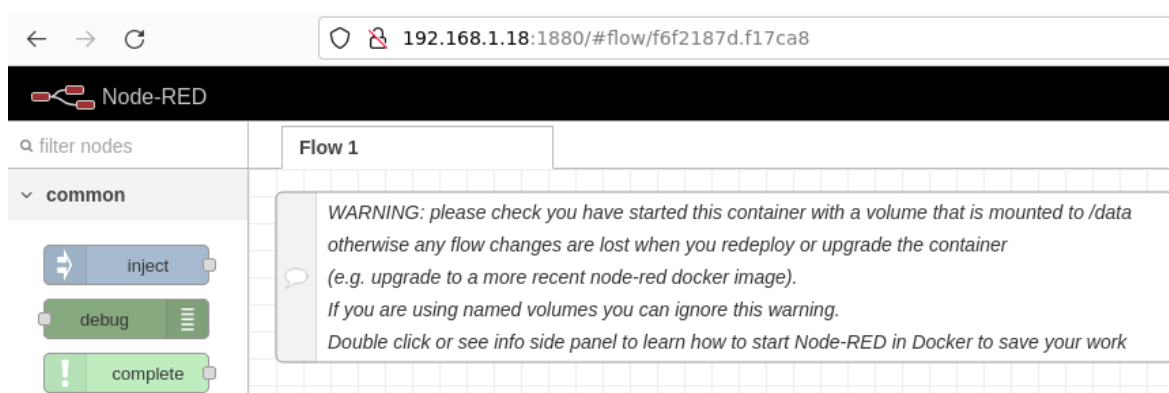
Une fois fait, on crée le docker nommé noderedRT avec la commande suivante :

**docker run -d -p 1880:1880 -v volNodeRed:/data --name noderedRT nodered/node-red**

De la même façon que précédemment à partir de l'image Node-Red téléchargée on démarre le conteneur en arrière-plan, on mappe le port 1880 de la machine hôte sur le port 1880 du conteneur. On connecte ensuite le volume nommé volNodeRed au répertoire /data du conteneur.

Une fois installé et démarré, le docker propose un site de création de projet à l'adresse suivante : <http://192.168.1.18:1880>

On retrouve bien les possibilités de création précédemment explicités :



#### 1.4. Docker n°3 : Installation de InfluxDB

InfluxDB est une base de données de séries chronologiques (time series database) open-source, conçue pour stocker, interroger et visualiser des données qui évoluent avec le temps. Elle est optimisée pour la collecte et la gestion de données temporelles provenant de capteurs, d'appareils IoT, de mesures météorologiques, offrant des fonctionnalités telles que la rétention des données à long terme, le partitionnement et des requêtes performantes basées sur le temps. Nous aurons besoin de cette base pour stocker les données récupérée depuis le broker MQTT.



On commence par récupérer l'image influxdb:1.8 avec la commande :

```
docker pull influxdb:1.8
```

Ensuite et de la même façon que précédemment, on démarre le conteneur en arrière-plan en mappant le port 8086 de la machine hôte sur le port 8086 du conteneur. Le volume nommé volinfluxdbRT est concaténé au répertoire /var/lib/influxdb du conteneur. Pour faire cette manipulation,, on effectue la commande suivante :

```
docker run -d -p 8086:8086 --name influxdbRT -v volinfluxdbRT:/var/lib/influxdb  
docker.io/library/influxdb:1.8
```

Nous utiliserons les fonctionnalités de influxdb plus tard. Pour l'instant, nous allons chercher à implémenter influxdb dans Node-Red. Pour ce faire, il suffit d'installer un nouveau module nommé **node-red-confib-influxdb**.

Une fois le docker installé, nous allons pouvoir chercher à configurer notre base de données. Pour se faire, on tape la commande suivante pour entrer dans le docker tout en restant au sein de la VM : **docker exec -it influxdbRT influx**. On peut alors interagir avec l'outil pour créer des tables, les modifier, ...

Voici l'affichage dans le terminal que nous propose le docker :

```
root@a8332eb9cd36:/# influx  
Connected to http://localhost:8086 version 1.8.10  
InfluxDB shell version: 1.8.10  
>
```

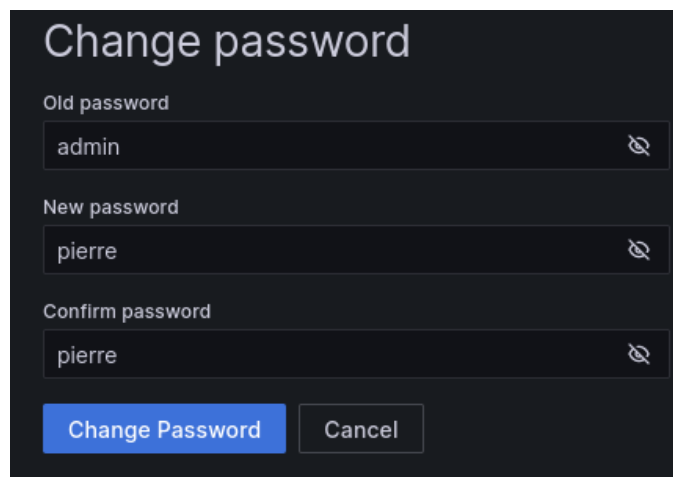
Nous reviendrons plus tard sur la création de la base qui va servir à récupérer les données des capteurs pour chaque bâtiment de l'iut.

### 1.5. Docker n°4 - Installation et configuration de Grafana

Grafana est une plateforme open-source d'analyse et de visualisation des données en temps réel. Elle permet de créer des tableaux de bord interactifs et personnalisables pour surveiller et analyser des données provenant de différentes sources. C'est grâce à cet outil que nous allons rendre la visualisation des données pour l'utilisateur agréable et intuitif. Nous dédions donc comme précédemment un docker à cet outil. Pour installer ce dernier, on récupère le docker avec la commande :

```
docker run -d -p 3000:3000 --name grafanaRT -v volGrafana:/var/lib/grafana grafana/grafana
```

Une fois le docker installé et démarré, il est possible de se rendre sur la page web grafana avec l'adresse suivante : <http://192.168.1.18:3000>. Le couple d'identifiant mot de passe utilisé est par défaut le suivant : **admin/admin**. On modifie le mot de passe pour plus de sécurité tel que :



Une fois que tous les dockers sont installés, il est possible de tous les démarrer simultanément avec la commande :

```
pierre@SAE23-Docker:~$ docker start $(docker ps -a -q)  
ef1d3ffb172d  
a8332eb9cd36  
b7f3461fb3ee  
c772d5716011
```

Cette commande est intéressante car on stocke en bash le résultat de la commande (**ps -a -q**) dans la variable temporaire '\$'. Cette commande permet de récupérer les ID des dockers et de démarrer chacun d'eux avec la commande **docker start**.





## 2. Première approche de visualisation des données : Node-Red Dashboard

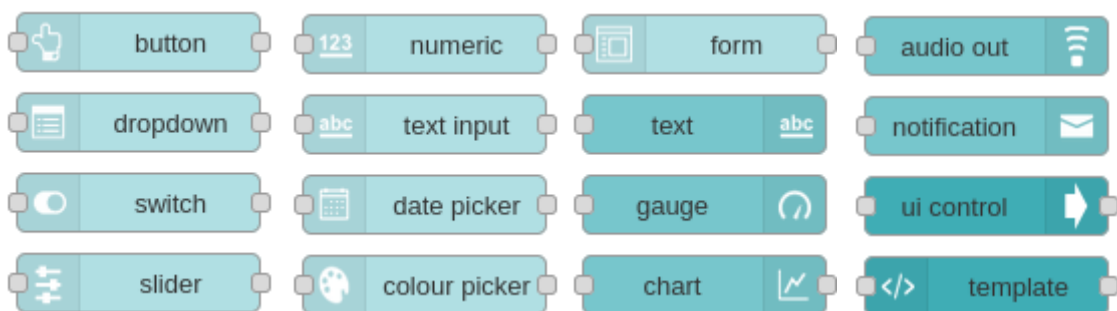
Au cours de la première partie, nous nous sommes contentés d'installer les différents dockers dont nous avons besoin. Cette deuxième partie aura pour objectif de traiter la première partie de la problématique, à savoir comment afficher les données récupérées via le broker avec un affichage à la volée de ces données au fur et à mesure que ces dernières arrivent. Pour ce faire, nous utiliserons le module Node-Red Dashboard.

### 2.1. Installation de Node-Red Dashboard

L'installation de Node-Red Dashboard est très simple. Il suffit de suivre les quelques étapes suivantes :

1. Se rendre dans le menu (symbolisé par trois barres horizontales)
2. Cliquer Manage Palette puis sur Install
3. Rechercher le module 'node-red-dashboard' et installer ce dernier

Une fois la manipulation faite et après avoir rechargé la page Web, de nouvelles options sont disponibles dans la catégorie dashboard :



Ces modules vont permettre de créer des graphiques divers. On peut se connecter au site internet associé à Node-Red Dashboard pour visualiser les graphiques sur le site suivant :

<http://192.168.1.18:1880/ui/>

## 2.2. Noeud 'mqtt in' : Récupération des données de capteurs

Le broker MQTT de l'iut envoie les valeurs de capteur pour chaque salle dans chaque bâtiment à intervalle régulier. Nous allons ici chercher à créer un nœud dans Node-Red permettant de récupérer les données telles qu'elles sont envoyées au format brut par le broker.

Avant de configurer ce nœud, il faut déterminer quel topic nous allons écouter sur le broker pour récupérer les données qui nous intéressent. Plusieurs choix sont possibles. Tout d'abord, on pourrait imaginer récupérer les données pour des salles distinctes. On peut par exemple prétendre à récupérer les données de la salle E209 avec la commande suivante :

```
pierre@SAE23-Docker:~$ mosquitto_sub -h mqtt.iut-blagnac.fr -p 1883 -t Student/by-room/E209/data  
{"topic": "Student/by-room/E209/data", "payload": {"temperature": 25.9, "humidity": 46, "activity": 0, "co2": 386, "tvoc": 240, "illumination": 66, "infrared": 19, "infrared_and_visible": 68, "pressure": 988.3}, "deviceName": "AM107-39", "devEUI": "24e124128c011464", "room": "E209", "floor": 2, "Building": "E"}}, "qos": 0, "retain": false, "_msgid": "1da0f593632e6020"}
```

Cette solution est très contraignante car elle nous oblige à effectuer une requête par salle, donc quatre requêtes pour répondre au minimum demandé dans le cahier des charges.

Nous utiliserons donc une autre méthode bien plus utile : nous allons récupérer toutes les données de toutes les salles de tous les bâtiments. Nous ferons donc l'écoute du topic suivant :

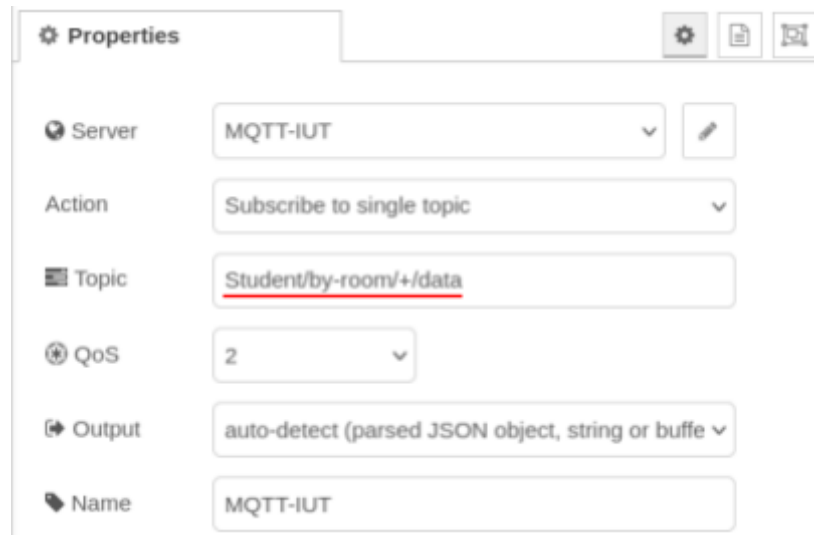
```
pierre@SAE23-Docker:~$ mosquitto_sub -h mqtt.iut-blagnac.fr -p 1883 -t Student/by-room/#/data
```

En utilisant '#', on demande au nœud Node-Red de récupérer toutes les données de tous les topics situés entre 'by-room' et 'data'. Ainsi, on peut symboliser l'astuce de la façon suivante :

**Student/by-room/#/data**  
  
=  
  
**Student/by-room/E001/data**  
**+ Student/by-room/E002/data**  
**+ Student/by-room/B112/data**  
**+ Student/by-room/C006/data**  
  
...

Ajoutons le nœud nommé 'mqtt in' situé dans la catégorie 'Network' en le faisant glisser dans l'espace de travail. En double-cliquant sur le nœud on le configure de la façon suivante :

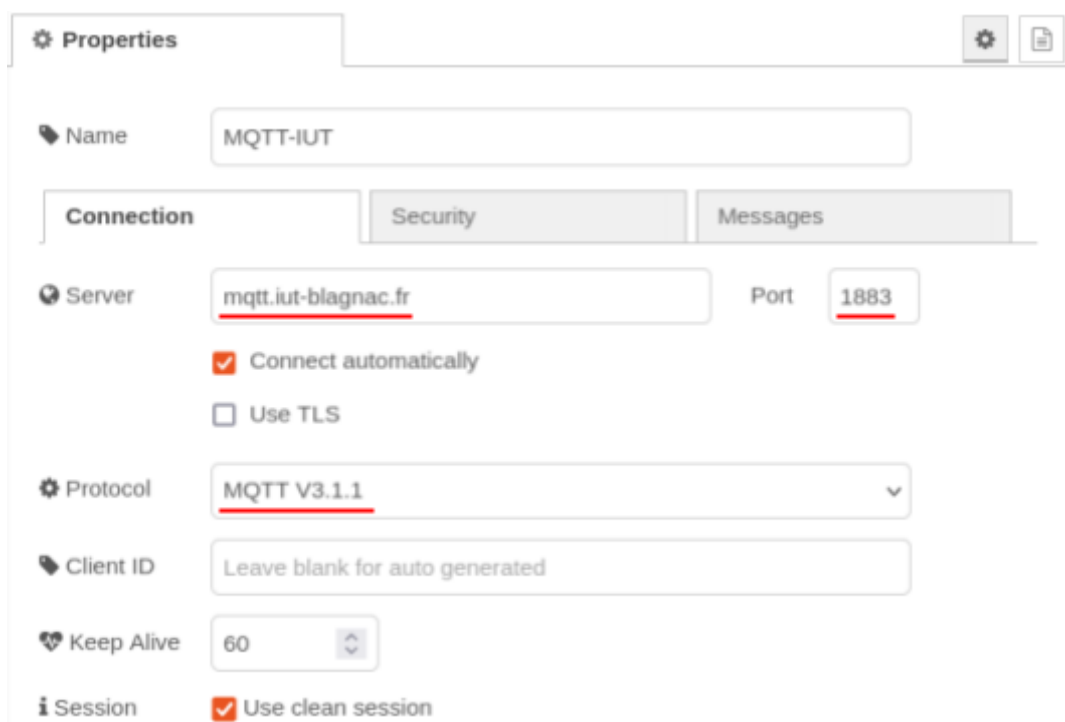
- Configuration générale du nœud :



The screenshot shows the 'Properties' configuration window for a node. The fields are as follows:

- Server:** MQTT-IUT
- Action:** Subscribe to single topic
- Topic:** Student/by-room/+/data
- QoS:** 2
- Output:** auto-detect (parsed JSON object, string or buffer)
- Name:** MQTT-IUT

- Configuration de la partie serveur (connexion au broker MQTT) :

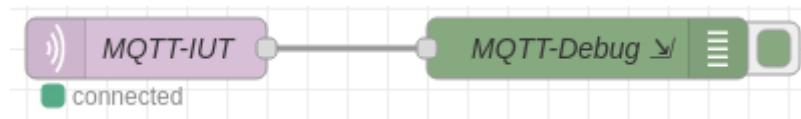


The screenshot shows the 'Properties' configuration window for a node, specifically the 'Connection' tab. The fields are as follows:

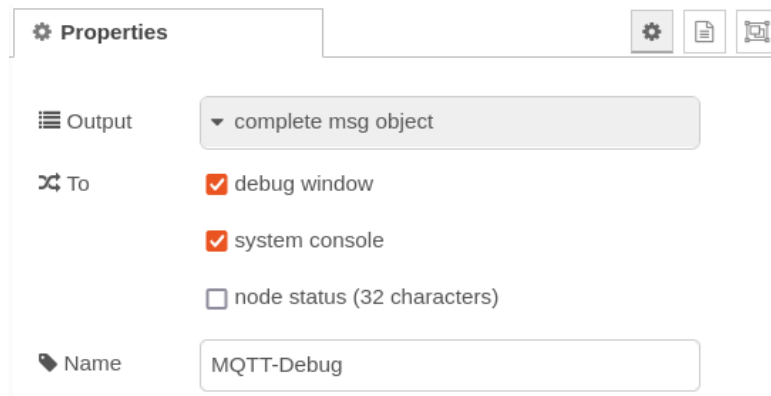
- Name:** MQTT-IUT
- Connection Tab:**
  - Server:** mqtt.iut-blagnac.fr
  - Port:** 1883
  - ☒ Connect automatically
  - ☐ Use TLS
  - Protocol:** MQTT V3.1.1
  - Client ID:** Leave blank for auto generated
  - Keep Alive:** 60
  - Session:** ☒ Use clean session

L'adresse du broker mqtt est la suivante : mqtt.iut-blagnac.fr (193.54.227.55). Le port de transmission des données est le 1883 et le protocole MQTT est la V3.1.1.

On ajoute en sortie du nœud de récupération de données 'MQTT-IUT' un autre nœud de type 'debug', pour observer la sortie de 'MQTT-IUT'. On nomme de noeud 'MQTT-Debug'.



Le nouveau noeud est très simple à installer :



On peut alors observer en sortie la donnée suivante de la salle B112 :

```
29/05/2023 19:40:21 node: MQTT-Debug
Student/by-room/B112/data : msg : Object
▶ { topic: "Student/by-room/B112/data", payload:
  "[{temperature:25.7,humidity:52...", qos: 0,
  retain: false, _msgid: "d8b6c8ac62bb4b7a" }
```

Ce test nous permet de valider que les données sont correctement reçues.

On se rend cependant compte que la donnée .json comporte de très nombreuses valeurs, correspondantes par exemple à la température ou encore au taux d'humidité. Pour trier ces valeurs et n'extraire que certaines d'entre elles, il nous faudra filtrer ce paquet de données avec des fonctions.

### 2.3. Noeud 'function' : Filtrage des données

Dans cette partie, nous allons chercher à trier les données de capteurs issues du broker MQTT, dans le but de rendre cette donnée exploitable pour la création de graphiques. Avant cela, nous allons nous intéresser au format des données renvoyées par le nœud 'MQTT-IUT'.

Dans Node-Red, les données sont récupérées dans la structure de payload **msg.payload**. Le contenu du message (**msg**) est stocké dans ce qui est appelé un payload, une charge utile en français.

D'autre part, les données récupérées par le broker mqtt sont au format .json et plus particulièrement d'un tableau .json de deux cases. On peut simplifier le format de la donnée récupérée par l'approximation suivante : la première case du tableau contient les mesures du capteur, alors que la deuxième case contient les informations sur le capteur. On peut donc symboliser la donnée tel que : **[{ Donnée },{ Informations }]**

Exemple réel de format de la donnée récupérée :

```
[{"temperature":25.9,"humidity":46,"activity":0,"co2":386,"tvoc":240,"illumination":66,"infrared":19,"infrared_and_visible":68,"pressure":988.3},{ "deviceName":"AM107-39", "devEUI":"24e124128c011464", "room":"E209", "floor":2, "Building":"E"}]
```

Les valeurs des différents types de capteurs sont accessibles de la façon suivante :

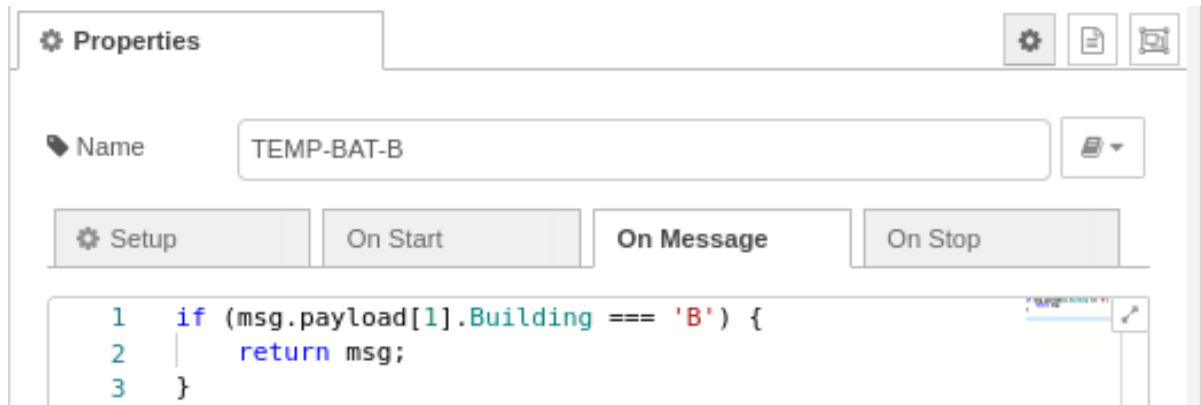
- Accès dans première case du tableau à la valeur de température :  
**msg.payload[0].temperature** → Équivalente ici à 25.9
- Accès dans deuxième case du tableau à la salle équipée du capteur :  
**msg.payload[1].room** → Ici la salle E209

Maintenant que nous avons compris comment les données étaient organisées et récupérées, nous allons pouvoir aborder le tri des données à l'aide de fonctions. Le but avec ces fonctions est de récupérer le contenu d'un champ de mesure en particulier pour une mesure donnée.

Trois fonctions différentes sont créées : la première sert à filtrer le message en fonction du bâtiment, la deuxième par salle et la dernière par type de valeur. Ici aussi, il aura été question de choix. En effet, il aurait été possible de créer une seule fonction effectuant tous ces tests, et ainsi de gagner en optimisation, mais il serait devenu très compliqué de se repérer dans Node-Red. Nous aurons l'occasion de justifier ce choix quand cette première approche de visualisation sera terminée.

- Première fonction : filtrage par bâtiment ~

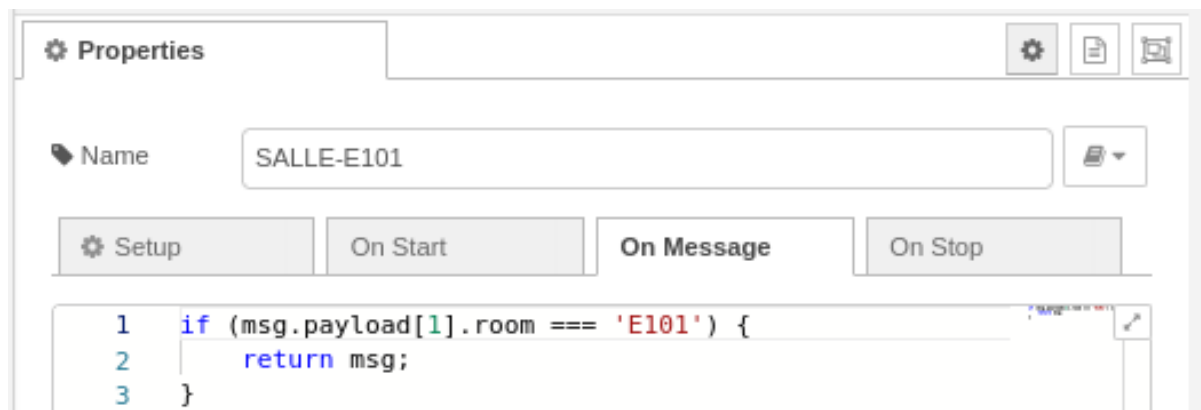
La fonction de filtrage par bâtiment est une fonction simple à configurer de la façon suivante :



La fonction sert donc à laisser passer le message 'msg' si la condition est respectée, ici si le bâtiment duquel provient le capteur est le bâtiment B. On remarquera que l'on retourne directement le message sans y apporter de modification.

- Deuxième fonction : filtrage par salle ~

La fonction de tri par salle est très similaire à celle de tri par bâtiment :

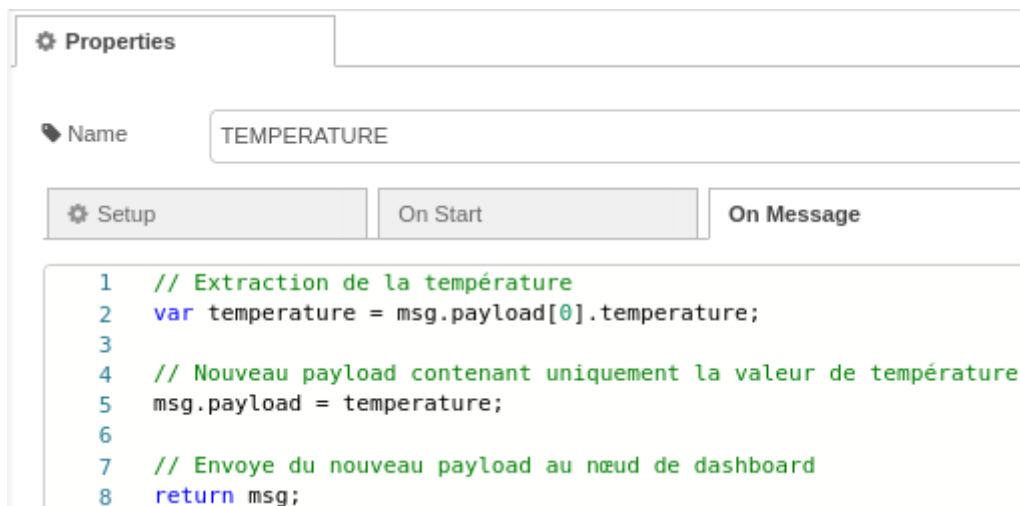


Ce n'est plus ici le champ 'Building' qui est testé mais le champ 'room' du paquet msg.payload.

On peut noter qu'il aurait été possible de concaténer les deux fonctions de tris (par bâtiments et salle) dans une unique fonction. Cette manipulation nous aurait permis de gagner en temps de calcul, mais aurait été un mauvais choix car aurait rendu le flow Node-Red difficile à analyser et assez illisible.

- Troisième fonction : filtrage sur le type de donnée

Cette dernière et troisième fonction est la plus intéressante des trois à expliquer. En effet, nous allons chercher à récupérer un champ en particulier de données dans le paquet .json brut récupéré sur le broker.

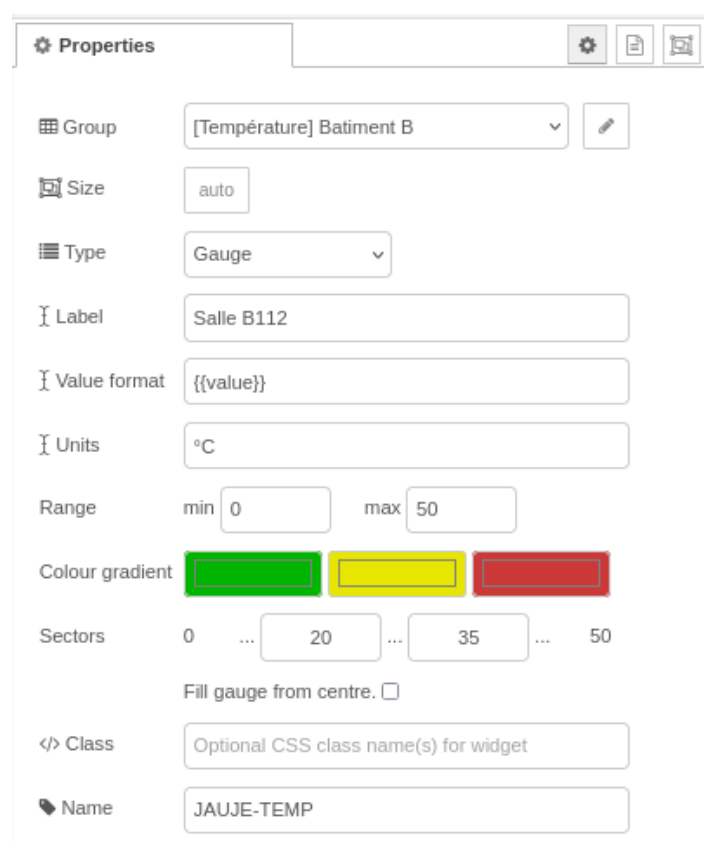


```

1 // Extraction de la température
2 var temperature = msg.payload[0].temperature;
3
4 // Nouveau payload contenant uniquement la valeur de température
5 msg.payload = temperature;
6
7 // Envoie du nouveau payload au nœud de dashboard
8 return msg;

```

On commence par récupérer dans une variable température la valeur de température contenu dans le paquet initial. Ensuite, on écrase le paquet initial pour n'y insérer qu'une seule valeur (décimale ou non). Le paquet fraîchement créé est ensuite envoyé vers une jauge permettant d'afficher la valeur. On configure la jauge de cette façon :



Properties

Group: [Température] Batiment B

Size: auto

Type: Gauge

Label: Salle B112

Value format: {{value}}

Units: °C

Range: min 0 max 50

Colour gradient: [Green] [Yellow] [Red]

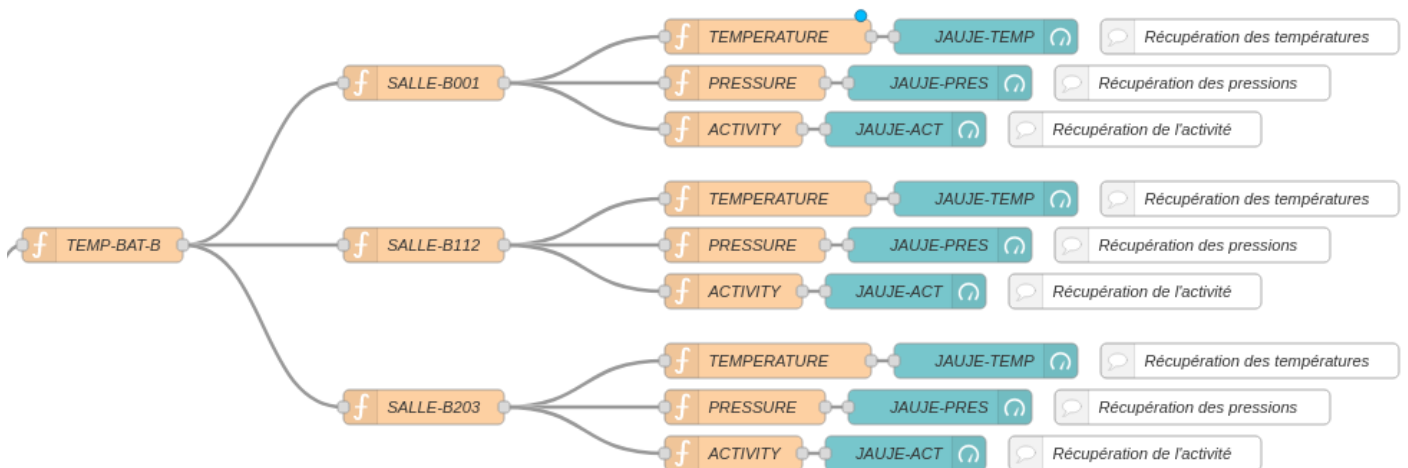
Sectors: 0 ... 20 ... 35 ... 50

Fill gauge from centre: ☐

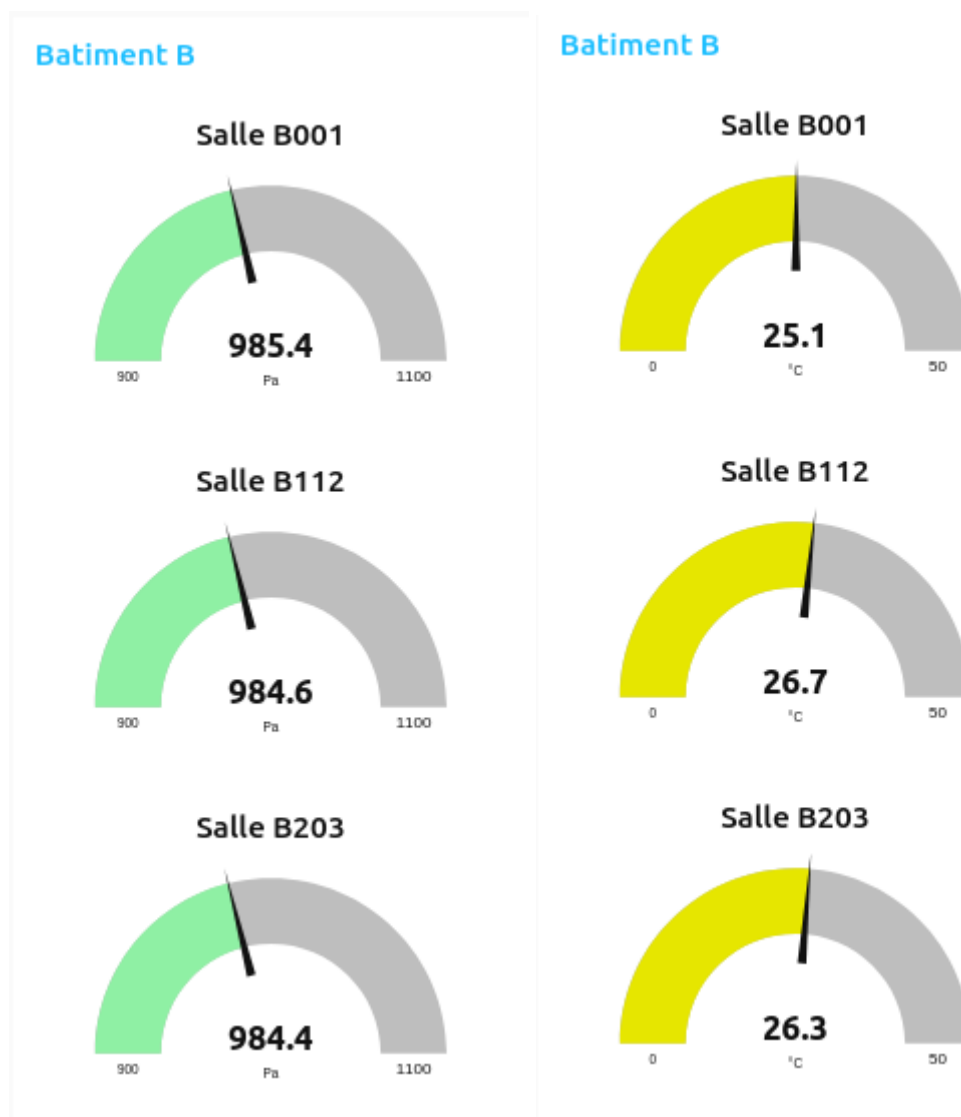
Class: Optional CSS class name(s) for widget

Name: JAUJE-TEMP

Pour le bâtiment B, on peut observer le flow suivant dans Node-Red :



Par exemple, le résultat est le suivant dans Node-Red Dashboard pour les valeurs de pression et de température :





### 3. Deuxième approche de virtualisation des données : InfluxDB et Grafana

#### 3.1. Création de la base de donnée InfluxDB

Nous allons ici chercher à créer la base de données qui va recueillir les données de capteurs. Pour se faire, on lance le docker et on rentre dans le terminal bash de ce dernier avec les commandes :

```
pierre@SAE23-Docker:~$ docker start a8332eb9cd36
a8332eb9cd36
pierre@SAE23-Docker:~$ docker exec -it a8332eb9cd36 /bin/bash
root@a8332eb9cd36:/#
```

Pour entrer dans InfluxDB, il suffit de taper la commande '**influx**' dans le terminal bash. Une fois fait, on peut créer la base de donnée avec :

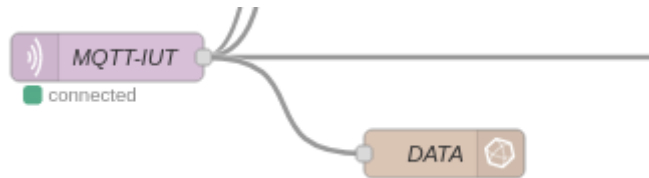
```
root@a8332eb9cd36:/# influx
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
> CREATE DATABASE SAE23_Sensor
> SHOW DATABASES
name: databases
name
----
_internal
SAE23_Sensor
```

Une fois la base créée, on peut ajouter un utilisateur nommé 'sae23' et ayant comme mot de passe 'sae23'. Cette utilisateur dispose de tous les droits et sera plus tard nécessaire pour se connecter à la base de donnée :

```
CREATE USER sae23 WITH PASSWORD 'sae23' WITH ALL PRIVILEGES
```

### 3.2. Implémentation de InfluxDB dans Node-Red

Maintenant que la base de données est créée, nous allons de nouveau utiliser Node-Red pour injecter les données récupérées via le nœud broker dans SAE23\_Sensor. Cette étape est assez simple car elle ne nécessite que l'implémentation du nouveau nœud 'InfluxDB in'. On nommera ce nœud 'DATA' :



Une fois ajouté dans le flow Node-Red, il ne reste plus qu'à le configurer de la façon suivante :

The image shows two screenshots of the Node-Red configuration interface. The top screenshot is for the 'DATA' node, and the bottom screenshot is for the 'SAE23\_Sensor' node. Both screenshots show the 'Properties' tab with various configuration fields.

**DATA Node Properties:**

- Name: DATA
- Server: [v1.8-flux] SAE23\_Sensor
- Database: SAE23\_Sensor
- Measurement: sensor\_data
- Time Precision: Seconds (s)
- Retention Policy: (empty)
- Tip: If no retention policy is specified, autogen will be assumed.

**SAE23\_Sensor Node Properties:**

- Name: SAE23\_Sensor
- Version: 1.8-flux
- URL: http://172.17.0.3:8086
- Username: sae23
- Password: (masked with dots)
- Verify server certificate: ☒

### 3.3. Visualisation des données à l'aide de Grafana Dashboards

Maintenant que toutes les données sont récupérées par le broker mqtt et envoyées dans la base de données, nous allons chercher à exploiter cette dernière avec l'outil de visualisation graphique Grafana. Cet outil étant très complet, nous n'allons pas ici comment configurer chaque graphique.

Il y a plusieurs étapes à suivre pour utiliser Grafana.

Premièrement, il est nécessaire de créer un compte. Une fois fait, il faut se connecter à la base de donnée SAE23 en suivant la procédure suivante :

- Se connecter à son compte utilisateur
- Ajouter une source de données en cliquant sur Add your first data source.
- Sélectionnez InfluxDB puis remplissez les champs URL (ne pas oublier le port), Database, User et Password avec les valeurs appropriées.
- Cliquez sur Save & Test. Vous devez obtenir le message Data source is working.

Voici un aperçu de la configuration effectuée :

SAE23-InfluxDB

Type: InfluxDB

Settings

Alerting supported

Name: SAE23-InfluxDB Default ☒

Query Language

InfluxQL

HTTP

URL: http://172.170.3:8086

Allowed cookies: New tag (enter key to add) Add

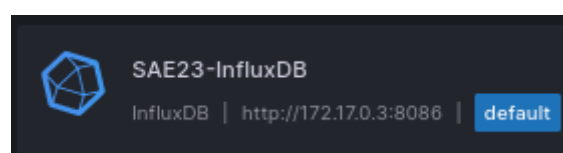
Timeout: Timeout in seconds

Database: SAE23\_Sensor

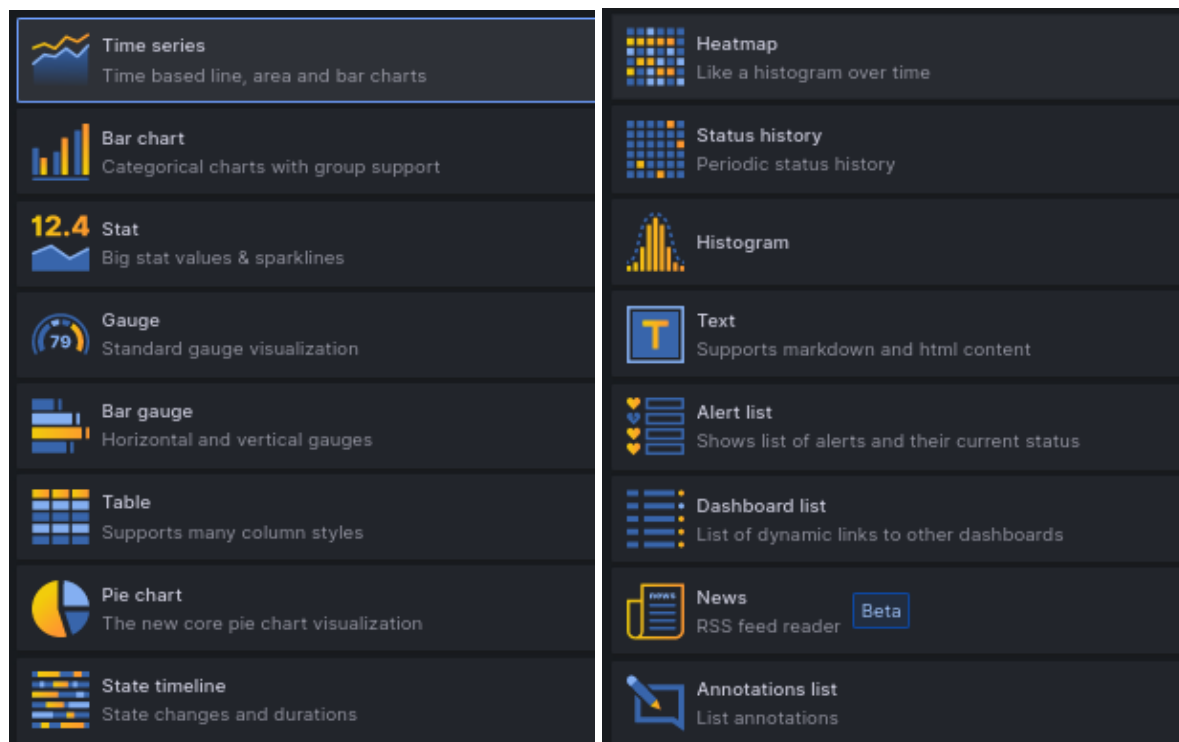
User: sae23

Password: configured Reset

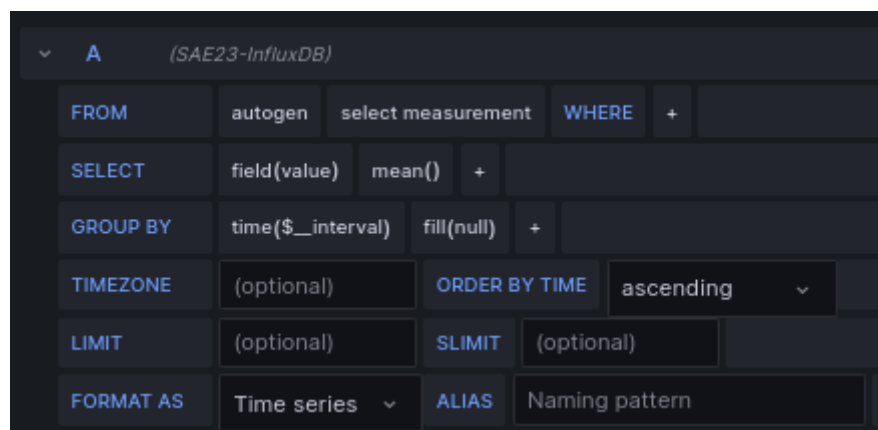
Une fois cette manipulation effectuée, vous devriez voir apparaître dans la liste des connectés a des base de données celle fraîchement configurée :



Il est alors possible à l'aide de créer des graphiques dans différents dashboards. Il existe un choix étendu de templates tels que les suivants (non-exhaustif) :



Les graphiques sont configurables à l'aide d'un menu 'option déroulant mais surtout d'un assistant graphique pour générer des requêtes InfluxDB sur la base de données :

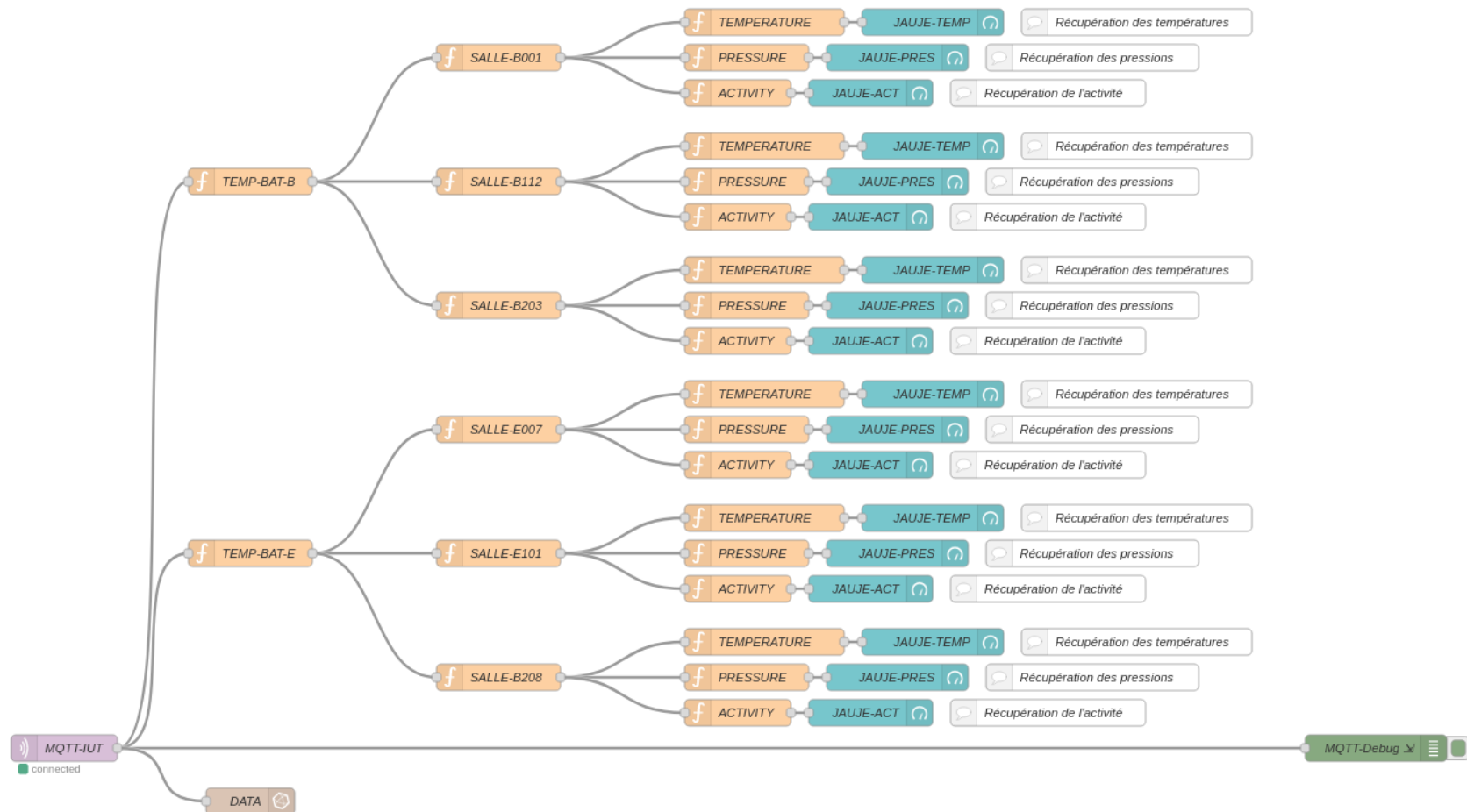


Maintenant que tous les outils sont fonctionnels, vous avez la possibilité de mettre en place de beaux dashboard pour visualiser les données récupérées par vos capteurs.

## 4. Visualisations et résultats

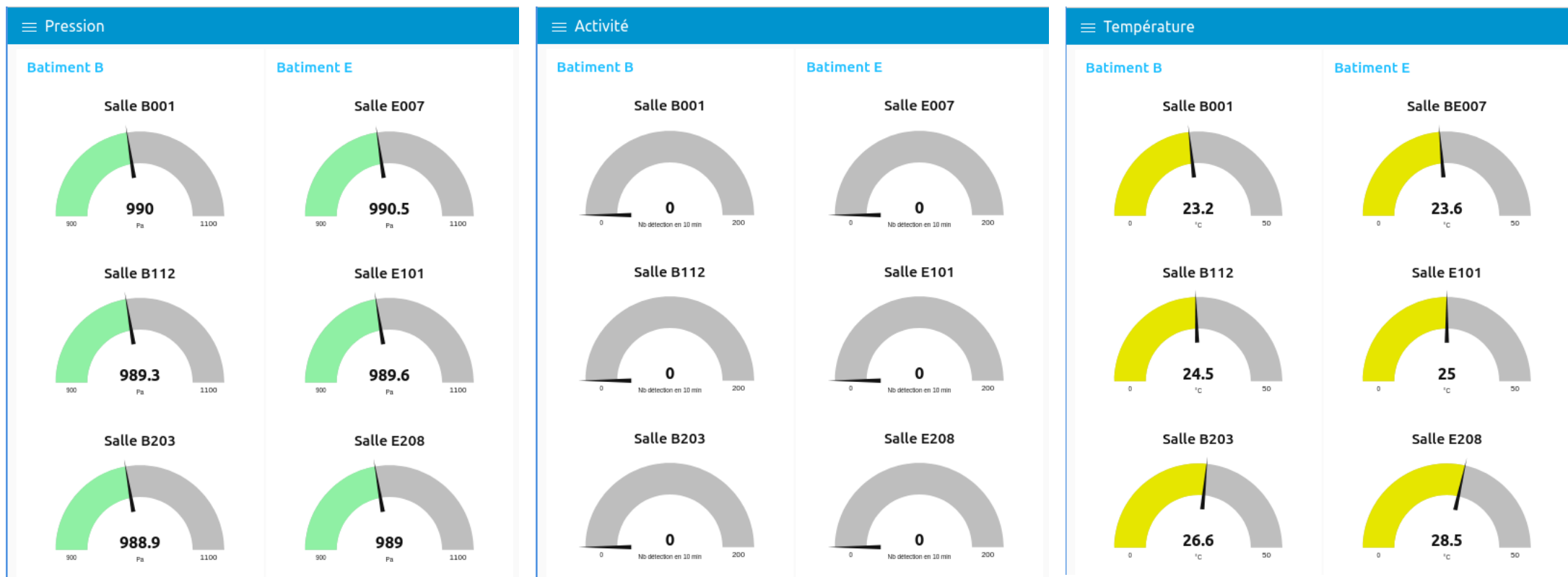
La dernière partie de ce rapport aura pour objectif de montrer les différents résultats obtenus en fin de projet concernant les approches de visualisation de données.

### 4.1. Flow Node-Red



## 4.2. Node-Red Dashboard

Il aura été créé dans Node-Red Dashboard 3 pages de visualisation de données : pour la température, la pression et l'activité dans les salles. Les screens-shots ayant été réalisés un samedi, il est normal que l'activité dans les salles de classe soit nulle (étudiants et enseignants en weekend).



Le choix de la visualisation via des jauges a été le plus judicieux étant donné le type de valeur récupérée.

#### 4.3. Grafana Dashboard

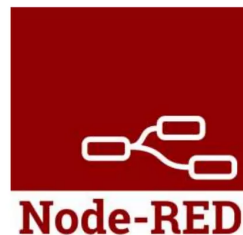
Grafana Dashboard est l'outil qui nous a permis de visualiser le mieux les données, via des graphiques détaillés et personnalisables :  
A noter que les valeurs de longitude et latitude de la salle du conseil de l'IUT serviront de repère pour placer l'IUT sur la carte du monde.



## Conclusion

En conclusion, nous arrivons au terme de ce projet avec deux solutions de visualisation pour les données de capteurs de l'IUT de Blagnac. Grâce à la solution docker et à l'utilisation d'outil d'automatisation (Node-Red), de base de donnée (InfluxDB) et de visualisation (Grafana et Node-Red Dashboard), les utilisateurs ayant accès à ces données peuvent avoir une idée claire des métriques des différents bâtiments et salles de l'IUT.

Ce travail, constituant le premier exercice de la SAE23, aura permis de nous familiariser avec des outils IoT qui pourront nous être utiles dans nos futures missions en tant qu'administrateur réseau (ou autres métiers des réseaux et télécommunications). Une autre solution de visualisation des données, proposant un panel de données à visualiser bien plus complet, constituera le second exercice de cette SAE.



**A visual tool for wiring the internet of things**

