27/01/2023

Solutions numériques pour l'industrie 4.0

Groupe 13

BERNARD Hippolyte – CHANDRASEGARAM Gagenth – DJEBARRI Dehbia – KICHENASSAMY Kavitha – LECLERC Maxence – MAKON Manyim Ma

ESIEA

Table des matières

I. Définition de l'architecture globale

L'objectif de ce projet est de mettre en place une architecture d'industrie 4.0 semblable à l'architecture suivante :

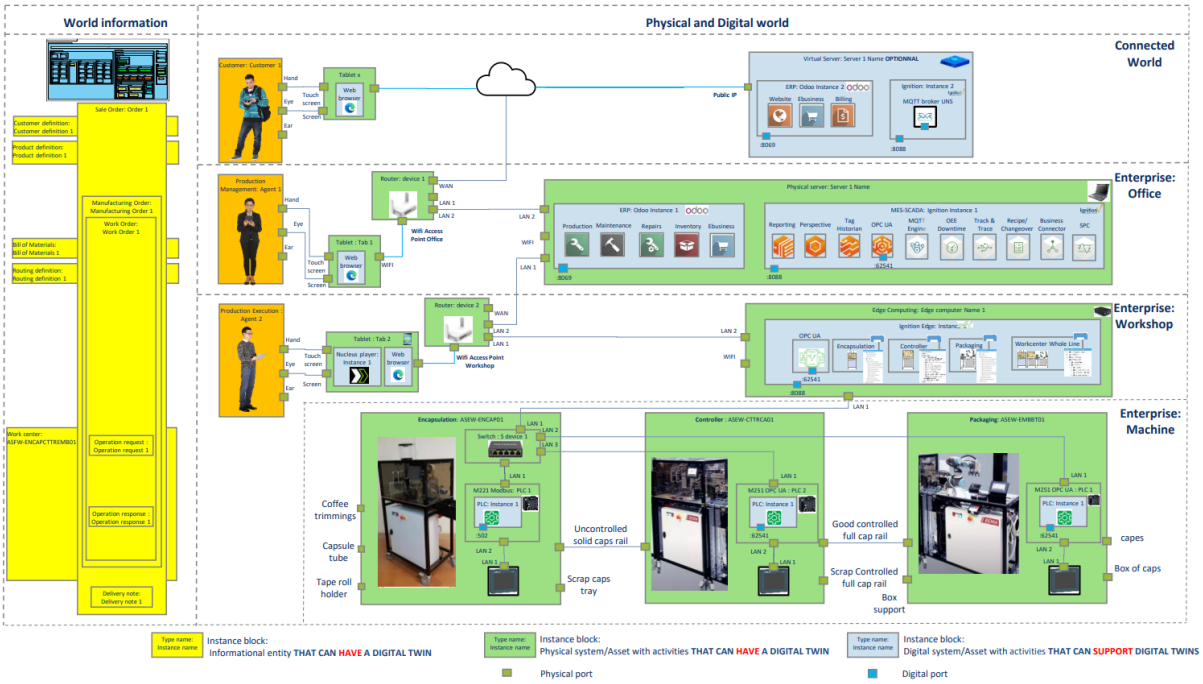

Figure 1 – Architecture de référence

Pour notre projet, nous avions un capteur de température et d'humidité relié à une carte Arduino.

Les données récupérées par le capteur étaient envoyées à un premier broker de messages MQTT sur un microcontrôleur Raspberry.

Une fois les données brutes envoyées au broker, un workflow Node-RED s'occupait de traiter ces informations pour les convertir en Sparkplug B et les envoyer à un second broker installé sur un automate.

Pour finir, un client Ignition récupérait les messages sur le broker de l'automate pour les afficher en temps réel à l'utilisateur.

Grâce à ce système, un utilisateur qui ne se trouve pas dans l'atelier peut accéder en temps réel aux informations d'une machine, comme par exemple la température et l'humidité dans une cabine d'usinage.

II. Mise en place du microcontrôleur

Voici le branchement réalisé entre le capteur DHT 22 et la carte Arduino ESP8266 :

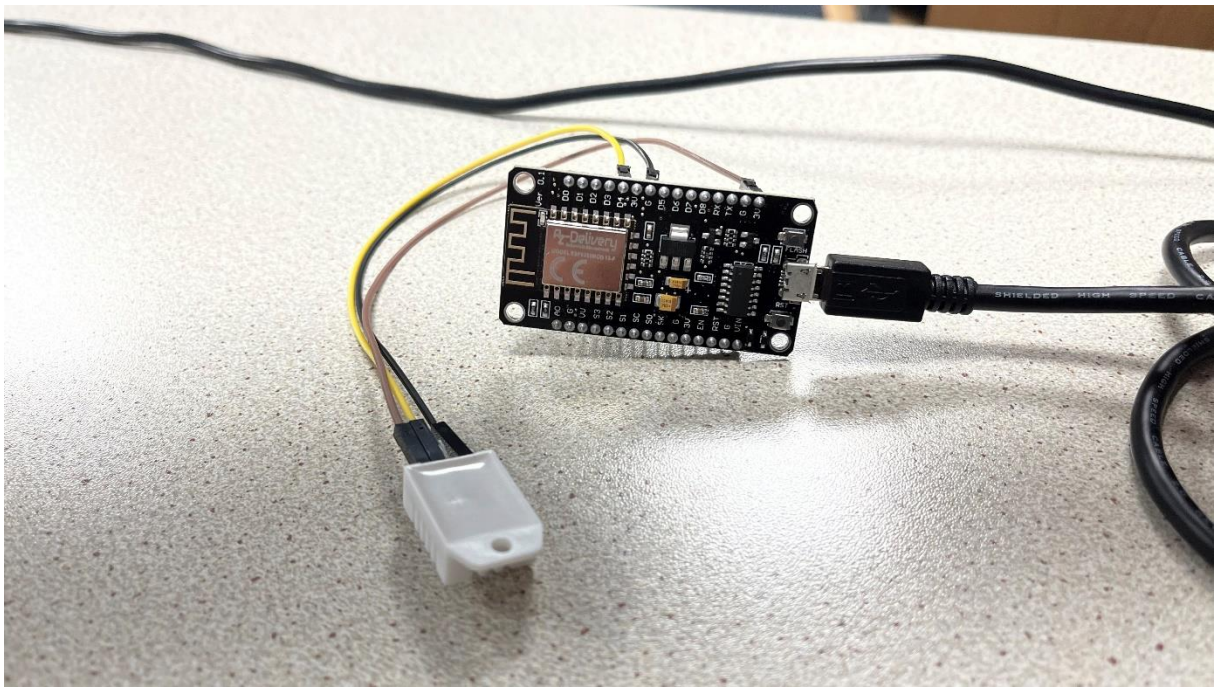

Figure 2 – Branchement du DHT 22 sur la pin 2 de l'ESP8266

La programmation de la carte Arduino se déroule en deux étapes : *setup* et *loop*.

A. Setup

La configuration du microcontrôleur commence par la connexion au réseau wifi local (réseau où est connecté le broker Raspberry).

La découverte automatique du broker sur le réseau local est faite à l'aide de requêtes mDNS, permettant de récupérer l'adresse IP et le port du serveur.

Note : la découverte automatique ne fonctionnant pas, l'adresse IP de la Raspberry a été directement codée dans la configuration.

La configuration se termine par la connexion au serveur MQTT.

B. Loop

La fonction principale du microcontrôleur est composée de trois parties distinctes :

- Récupération de la température et de l'humidité depuis le capteur
- Formatage du message en JSON
- Publication du message au broker

Cette boucle s'effectue toutes les 5 secondes.

```
14:47:42.936 -> {"temperature": 23.80, "humidity": 36.30}
14:47:47.948 ->     -----
14:47:47.948 -> {"temperature": 23.80, "humidity": 36.10}
14:47:52.950 ->     -----
14:47:52.950 -> {"temperature": 23.80, "humidity": 36.10}
14:47:57.965 ->     -----
14:47:57.965 -> {"temperature": 23.90, "humidity": 36.00}
```

Figure 3 – Messages envoyés au broker

Le code se trouve dans le fichier **arduino.ino** (dossier *export*).

III. Configuration de Node Red

Avant de pouvoir exploiter les données, elles doivent être converties au format Sparkplug B. Cette étape est réalisée par le workflow Node-RED suivant :

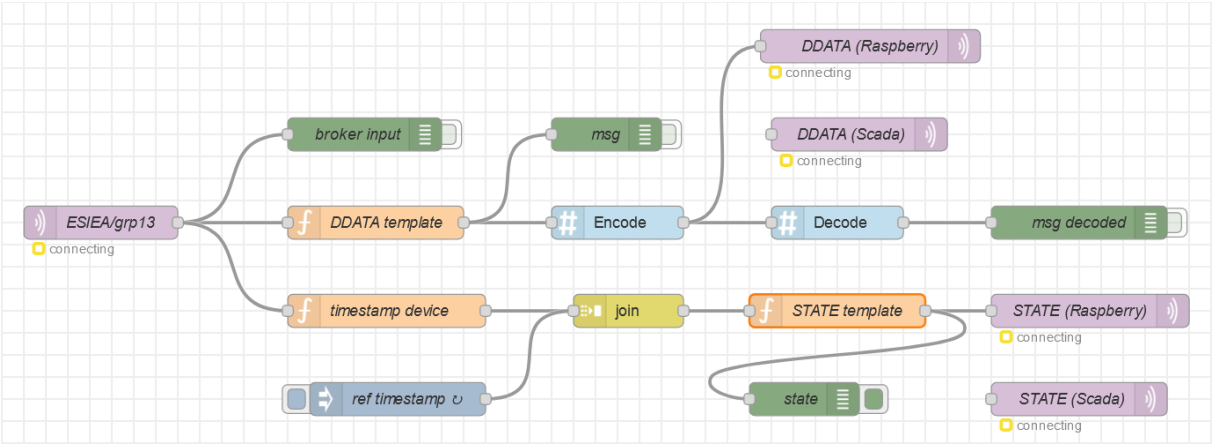

Figure 4 – Workflow Node-RED

Les données d'entrée sont issues du broker MQTT de la carte Raspberry, topic « ESIEA/grp13 ».

A. DDATA

Les données sont formatées pour respecter la norme Sparkplug B avec un timestamp et les données dans le champ « metrics » du payload.

Ce message est ensuite encodé depuis le fichier « payload.proto » pour être envoyé au second broker MQTT de l'automate sur le topic « spBv1.0/ESIEA/DDATA/raspberry/Pdevice_grp13 ».

Note : l'automate étant indisponible, le message a été envoyé au broker de la Raspberry.

B. STATUS

En parallèle de l'envoi des données du capteur, les informations du statut de notre microcontrôleur sont envoyées au broker.

Toutes les 2 secondes est généré un timestamp de référence. Lorsqu'un message arrive en entrée, son timestamp est récupéré et comparé à celui de référence. Ainsi, lorsque notre appareil se déconnecte,

son ancien timestamp est conservé et au bout de 7,5 secondes, le logiciel détecte que l'appareil est hors ligne et passe donc son statut sur « false ».

C. DBIRTH & NDEATH

En plus des messages DDATA et STATUS, des messages DBIRTH et NDEATH ont été paramétrés sur les nodes de sortie MQTT.

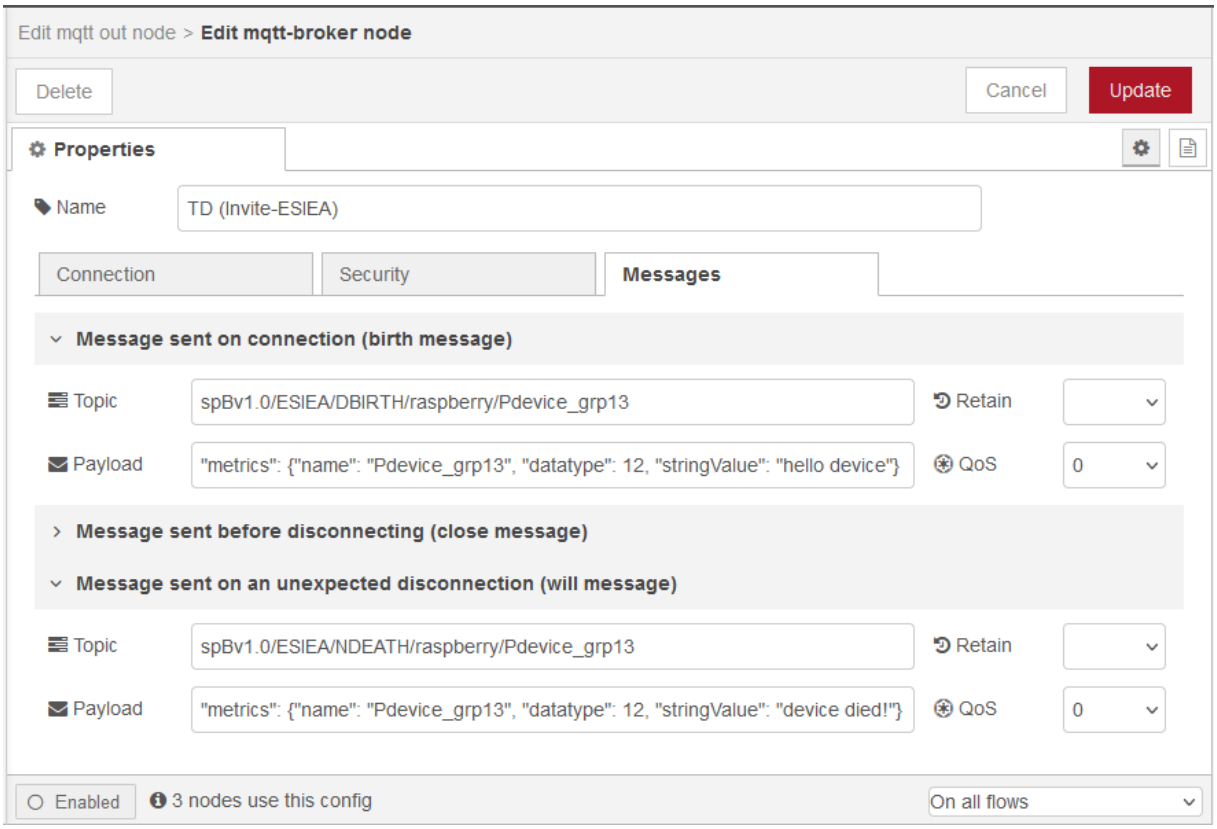

Figure 5 – Paramétrage des messages DBIRTH et NDEATH

La configuration Node-RED utilisée se trouve dans le dossier « *export* », fichier **Node-Red_flows.json**.

D. Visualisation des messages

A l'aide de l'application MQTT Explorer, on peut voir que tous nos messages sont disponibles sur le broker : DDATA, STATE et DBIRTH (NDEATH n'ayant pas été envoyé).

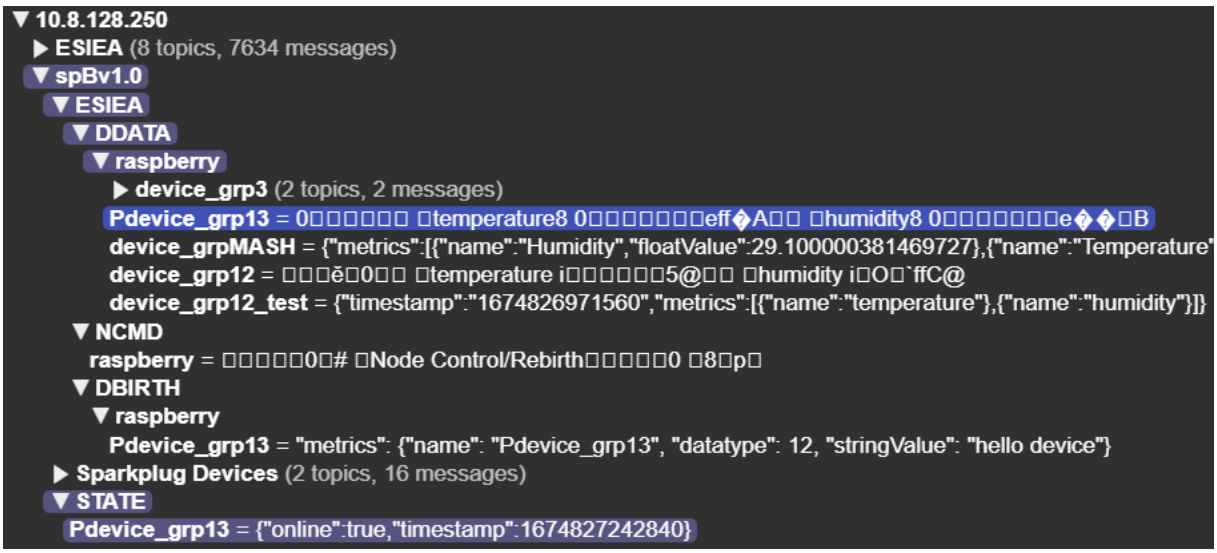

Figure 6 – Messages envoyés au broker

IV. Configuration de Ignition

L'étape finale du projet est de pouvoir visualiser en temps réel les données du capteur depuis Ignition.

Les deux brokers ont donc été ajoutés dans la configuration de serveurs (Raspberry et automate).

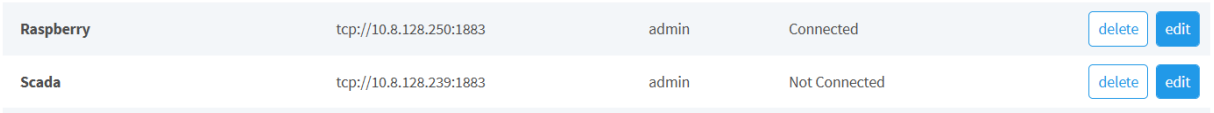

Figure 7 – Serveurs MQTT dans Ignition

Pour afficher les données, nous avons créé le projet « **tp_grp13_p** » (exporté séparément pour ne pas avoir à importer toute la configuration Ignition).

Nous avions initialement créé une page pour visualiser nos données, cependant nous n'avons pas pu lire les messages Sparkplug B contenu dans le broker de la Raspberry.

Le filtre rouge sur le thermomètre et la jauge indique que les données ne peuvent pas être lues.

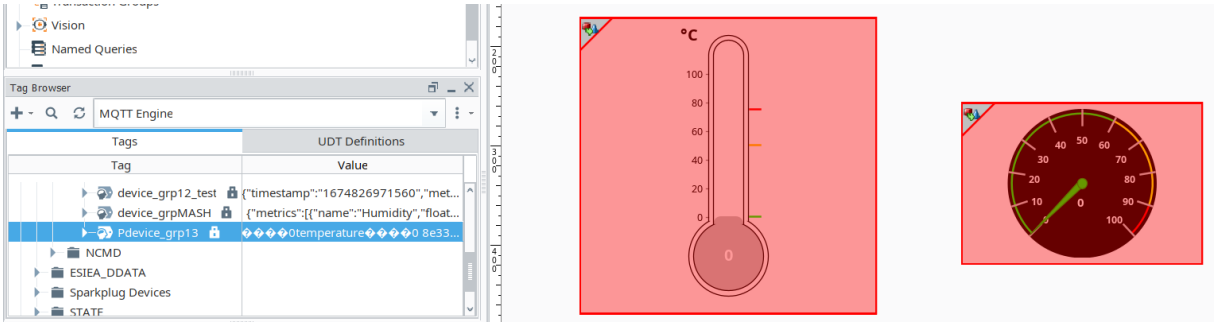

Figure 8 – Erreur à la lecture des données

Nous avons ensuite créé une autre page disponible à l'adresse suivante :
http://localhost:8088/data/perspective/client/tp_grp13_p/pgrp13

Cette page contient ce qu'un utilisateur devrait avoir comme résultat, avec l'historique des températures et la jauge d'humidité en temps réel.

Les données utilisées ne sont qu'à titre d'exemple étant donné les problèmes liés à lecture des messages contenant les vraies données.

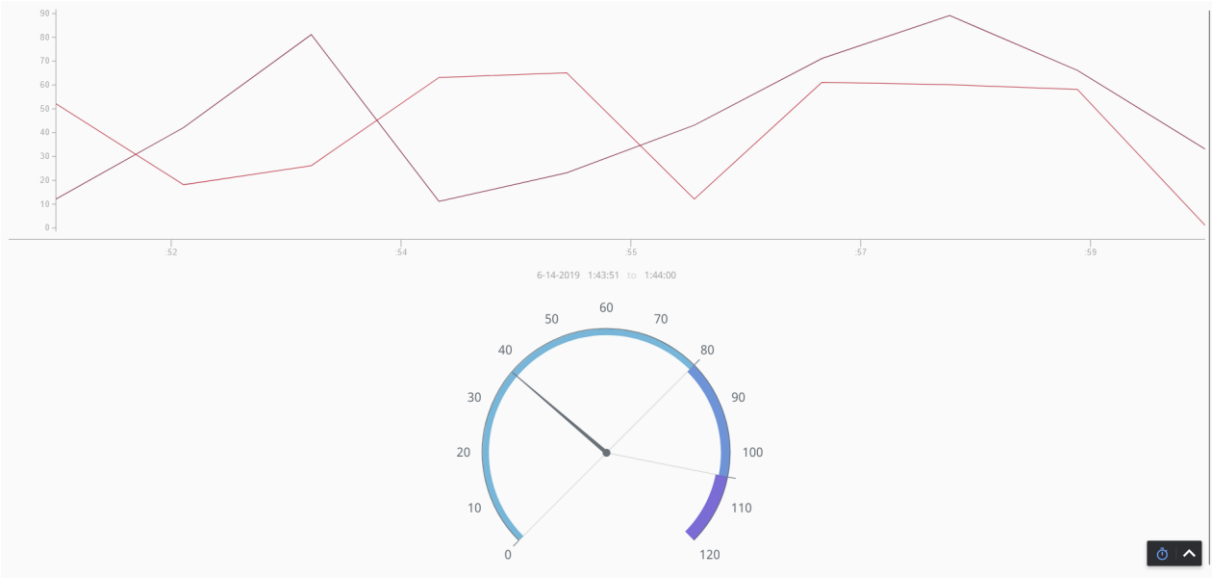

Figure 9 – Page finale de visualisation des résultats