IOT

PROJET

Contrôle environnemental pour station spatiale (ECLSS)

Livrable 3

Remy MOREEL

Léo SCHAEFFER

Axel DELAR

Mohammed EL ORFALI

CESI
ÉCOLE D'INGÉNIEURS

Table des matières

1. Introduction

Après trois semaines d'analyse et de conception, ce livrable marque le passage de la théorie à la pratique. L'objectif n'est plus seulement de proposer une architecture cohérente, mais de démontrer la viabilité technique du système à travers un prototype fonctionnel.

Dans ce cadre, nous avons réalisé un Proof of Concept (POC) opérationnel mettant en œuvre une chaîne complète de communication entre capteurs, une passerelle Edge et notre consommateur, en nous appuyant sur un protocole applicatif adapté aux objets contraints. Le choix s'est porté sur MQTT, en raison de sa légèreté, de son modèle de communication publication/souscription.

Ce livrable 3 présente :

- Une démonstration fonctionnelle du flux de données (capteurs – Broker MQTT – consommateur)
- Un schéma d'architecture de communication
- Une note protocolaire justifiant les choix techniques réalisés

Enfin, il y aura également une partie sur la sécurité du prototype. Des mesures de base, inspirées des recommandations de la norme ETSI EN 303 645, ont été intégrées afin de garantir un minimum de protection pour notre dispositif.

2. Présentation de l'équipe

Voici notre équipe pour la réalisation de ce projet :

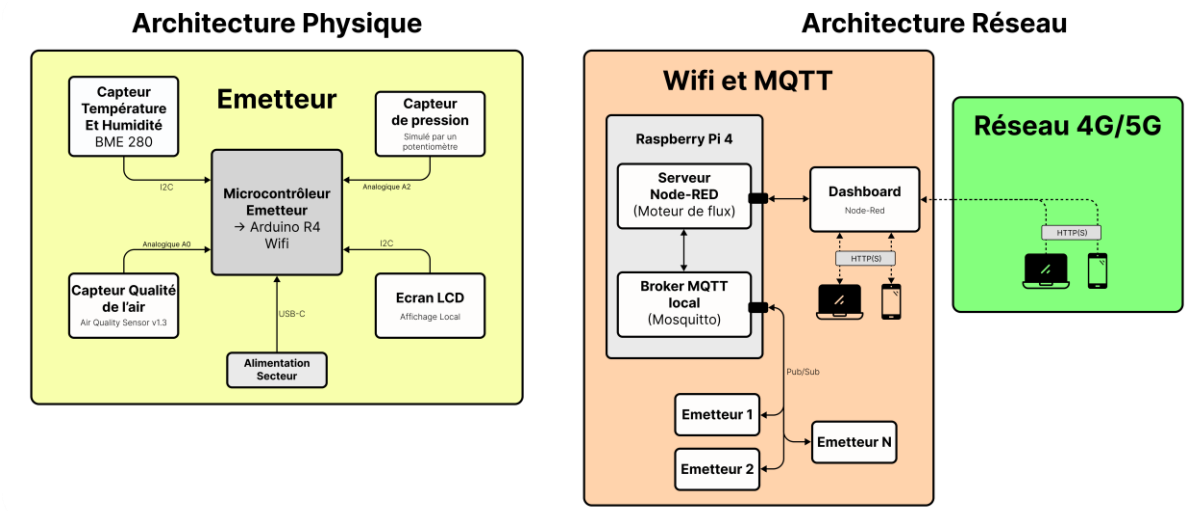
Mohammed EL ORFALI
Chef de projet

Léo SCHAEFFER

Axel DELAR

Remy MOREEL

3. Schéma d'architecture de communication



Ce schéma montre l'architecture globale de votre prototype, en séparant la partie physique (les objets contraints) et la partie réseau autour de MQTT.

Dans le bloc « Architecture Physique », on voit les cartes Arduino émettrices avec leurs capteurs (température/humidité BME280, qualité de l'air, pression) et l'écran LCD, tous reliés au microcontrôleur Arduino R4 WiFi qui mesure les grandeurs et publie les données.

Dans le bloc « Wifi et MQTT », la Raspberry Pi 4 joue le rôle de passerelle Edge : elle héberge le broker Mosquitto et le serveur Node-RED qui reçoit les messages, les traite et alimente le dashboard.

Enfin, le bloc « Réseau 4G/5G » illustre l'accès distant : le dashboard Node-RED peut être consulté depuis un smartphone ou un autre PC via HTTPS, pour visualiser en temps réel les mesures issues des Arduino publiées en MQTT.

Le déroulé complet de ce flux, depuis la mesure sur les capteurs jusqu'à l'affichage sur le dashboard, sera détaillé dans la suite du rapport.

4. Note protocolaire

I) Choix protocole MQTT

Dans le cadre de ce projet, le protocole MQTT (Message Queuing Telemetry Transport) a été retenu pour assurer la communication entre les capteurs, la passerelle Edge et les consommateurs de données. MQTT est particulièrement adapté aux objets contraints, grâce à sa légèreté, à sa faible consommation de bande passante et à sa simplicité de mise en œuvre.

Le protocole repose sur un modèle publication / souscription, qui permet de découpler les producteurs de données (capteurs) des consommateurs (applications, dashboards). Ce découplage facilite l'évolutivité du système et limite les dépendances entre les composants, ce qui est particulièrement pertinent dans un contexte IoT.

Dans notre architecture, le broker MQTT Mosquitto est déployé localement sur une Raspberry Pi, qui joue le rôle de passerelle Edge. Cette passerelle centralise les échanges, assure la réception des données issues des capteurs et leur redistribution vers les consommateurs.

II) Modèle de communication : publication/souscription

Les cartes Arduino, représentant les objets contraints du système, publient périodiquement les mesures issues des capteurs (température, pression, humidité, qualité de l'air). Ces données sont envoyées vers le broker MQTT sans connaissance directe des consommateurs.

Le dashboard Node-RED s'abonne aux topics correspondants afin de recevoir les données en temps réel. Ce modèle permet :

- D'ajouter ou de retirer des consommateurs sans modifier les capteurs,
- De centraliser le traitement au niveau de la passerelle Edge,
- De limiter les échanges inutiles sur le réseau.

Pour le moment, la communication est principalement unidirectionnelle (capteurs → consommateurs). Toutefois, l'architecture prévoit également une communication descendante (passerelle → capteurs), notamment pour le pilotage à distance d'actionneurs tels qu'un buzzer.

III) Organisation des topics MQTT

Les topics ont été organisés selon une logique par type de donnée, avec un préfixe commun permettant d'identifier le projet :

- eclss/temperature
- eclss/pression
- eclss/humidite
- eclss/airquality
- eclss/buzzer/salle1
- eclss/buzzer/salle2

Les mesures provenant de plusieurs salles sont publiées sur un même topic, accompagnées d'un identifiant de l'Arduino émetteur dans la charge utile du message. Le tri et la séparation des données sont ensuite réalisés au niveau de Node-RED, qui filtre les messages en fonction de cet identifiant.

Ce choix nous permet :

- De limiter le nombre de topics à gérer,
- De simplifier la configuration côté capteurs,
- De centraliser la logique de traitement et de routage sur la passerelle Edge.

Les topics eclss/buzzer/salleX sont réservés à une évolution future du système, permettant l'envoi de commandes depuis le dashboard vers les capteurs afin d'activer des actionneurs à distance.

IV) Qualité de service (QoS)

Les messages MQTT sont publiés avec un niveau de qualité de service QoS 1. Ce niveau garantit que chaque message est reçu au moins une fois par le broker et les abonnés, grâce à un mécanisme d'accusé de réception. En contrepartie, ce mode peut entraîner une duplication ponctuelle des messages, qui est gérée au niveau applicatif.

Ce choix constitue un compromis entre fiabilité et légèreté. Les données issues des capteurs sont envoyées à une fréquence d'une seconde et ne sont pas critiques au sens strict, mais une perte répétée de messages pourrait dégrader la lisibilité des mesures sur le dashboard. Le QoS 1 permet ainsi d'améliorer la fiabilité des échanges tout en conservant une charge réseau et une complexité limitée, compatibles avec un prototype IoT.

Ce niveau de QoS est également cohérent avec une architecture Edge locale, où la latence reste faible et où le surcoût induit par les accusés de réception demeure acceptable.

V) Gestion des sessions et des messages retain

Les messages MQTT sont publiés sans option retain. Ainsi, seuls les nouveaux messages sont transmis aux abonnés, et aucune valeur précédente n'est automatiquement envoyée lors de la connexion d'un nouveau client. Ce comportement est adapté à un système où les données sont mises à jour très fréquemment et où les valeurs passées présentent peu d'intérêt.

Les clients MQTT utilisent une clean session activée. Les capteurs étant en fonctionnement continu, hormis lors de redémarrages ponctuels, il n'est pas nécessaire de conserver des sessions persistantes ou des messages en attente. Ce choix simplifie la gestion des connexions et reste cohérent avec le caractère non critique des données.

5. Sécurité “baseline”

D’un point de vue sécurité et cybersécurité de notre dispositif, on se base sur la norme ETSI EN 303 645. Celle-ci définit un socle dit « baseline » d’exigences en termes de cybersécurité et de protection des données, pour les objets connectées (soit, l’IoT).

La norme met à disposition 13 mesures :

- **Pas de mots de passe universels par défaut** : Il est évident que nous n’utilisons aucun mot de passe par défaut (123456, P@ssw0rd, admin, etc.). Chaque mot de passe utilisé dans le projet est généré aléatoirement et stocker en local dans un fichier secrets.h (non disponible sur le repository GitHub).
- **Gérer les signalements de vulnérabilités** : En d’autres termes, si un incident est repéré par un client ou autres, sur un objet connecté que nous avons conçu, nous avons le devoir d’y remédier.
- **Maintenir le logiciel à jour** : Si des mises à jour de composants, ou des failles critiques dans le logiciel sont détectées, nous avons le devoir d’en faire une mise à jour de sécurité.
- **Stocker de façon sûre les paramètres sensibles** : Cela se rapporte à ce qui à été mentionné dans le point 1. Toutes nos données confidentiels et paramètres (Pin des capteurs, mots de passe Wi-Fi, port MQTT, Topics de communication) sont stockées dans un fichier secrets.h local.
- **Communiquer de façon sûre** : Toutes nos communications sont protégées par le protocole TLS au niveau du transport, ainsi que par un chiffrement HMAC au niveau applicatif. En faisant cela, les envois/réceptions de messages sont sécurisées.
- **Minimiser la surface d’attaque exposée** : En d’autres termes, limité le nombre de connexions et les données envoyées. Dans notre cas, nous faisons une sélection des données avant l’envoie via MQTT dans un souci de sécurité et d’optimisation.
- **Assurer l’intégrité logicielle** : Pour ce cas, nous sommes entrain de mettre en place une blockchain, qui permettra la vérification de l’intégrité.
- **Sécuriser les données personnelles** : Dans notre cas, nous ne stockons pas de données personnelles.
- **Résilience aux pannes** : Nous avons choisi pour ce point d’avoir nos émetteurs sur secteur, avec une batterie de secours en cas de coupure, de même que pour le routeur que l’on utilise pour les communications en local. Ainsi, s’il y a un problème nous stockons les données en local, afin de les renvoyées lors du rétablissement.

- **Exploiter la télémétrie (logs) de manière utile** : Pour ce point, nous avons pour projet d'instaurer une analyse prédictive sur notre serveur Node-Red, afin de pourvoir anticiper des pannes, de mauvaises données, etc.
- **Faciliter la suppression des données utilisateur** : Non concernées car nous ne stockons pas de données utilisateurs.
- **Rendre l'installation et la maintenance simples** : Pour cela, nous avons conçu un boitier permettant l'accès facile à tous les composants. De même, ces composants sont facilement trouvables et remplaçable.
- **Valider les entrées** : Par exemple, prévoir les injections SQL. Nous ne sommes pas concernées par ce point car nous n'attendons aucune entrée utilisateur.

Par ces différents points, nous garantissons la sécurité aux utilisateurs de notre prototype.

6. PoC

Pour ce livrable, il nous était demandé de fournir une vidéo de démonstration, un Proof of Concept du projet. Pour cela, nous avons tourné une vidéo contenant une démo de flux de données, d'un Arduino émetteur vers le serveur sur Raspberry Pi, utilisant le protocole MQTT.

Cette vidéo est disponible dans le dossier Moodle de ce livrable et contient les différents détails de notre configuration MQTT (topics, QoS, journal du Broker).

Dans un premier temps, la vidéo présente les deux cartes Arduino sur lesquelles sont connectés les différents capteurs. Ces cartes représentent les objets contraints du système et sont chargées de mesurer les données physiques.

Ensuite, nous affichons sur le PC le moniteur série d'une des cartes Arduino. Cela permet de visualiser en temps réel les valeurs mesurées par les capteurs ainsi que les données qui sont envoyées vers le réseau, afin de vérifier le bon fonctionnement de la collecte et de l'émission des messages.

La vidéo se concentre ensuite sur la Raspberry Pi 4, qui joue le rôle de passerelle Edge. Celle-ci héberge le broker MQTT. Nous affichons le journal du broker en temps réel, ce qui permet d'observer directement les messages échangés, les connexions des clients et les publications sur les différents topics. À ce stade, l'ensemble des topics est affiché afin d'avoir une vision globale des échanges, même s'il serait possible de se limiter à un topic spécifique selon les besoins.

Nous présentons ensuite le point d'accès Wi-Fi, qui permet d'interconnecter l'ensemble des équipements du prototype : les cartes Arduino, la passerelle Edge et les dispositifs consommateurs. Ce réseau local constitue le support de communication du système.

Enfin, la vidéo montre l'ordinateur client sur lequel est affiché le dashboard Node-RED. Pour des raisons de simplicité de configuration, le dashboard est temporairement hébergé sur cet ordinateur, mais il sera intégré directement à la Raspberry Pi dans la version finale. On peut alors observer que les jauges du dashboard se mettent à jour en temps réel à chaque nouvelle donnée reçue via MQTT, ce qui valide le bon fonctionnement de la chaîne complète.