AVIGNON UNIVERSITÉ

Système connecté de surveillance cardiaque (ECG)

Groupe de travail :
Azeddine BELILITA
Youssef DEROUICHE

3 janvier 2026

M2 INFRASTRUCTURES CLOUD SYSTEMES DISTRIBUES (SYRIUS)

ECUE architecture cloud (S-E06-3071)

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

Résumé

Sommaire

1 Introduction

1.1 Contexte

L'Internet des Objets (IoT pour *Internet of Things*) connaît une croissance exponentielle depuis le début de la dernière décennie. Comme souligné dans les notes de cours, les estimations faisaient état de près de 15 milliards d'objets en circulation, avec des projections dépassant les 50 milliards à l'horizon 2020 et au-delà. Ces objets communicants, capables d'interagir entre eux (M2M) et avec des serveurs distants, génèrent des volumes massifs de données capitalisées dans des centres de données pour créer de nouveaux usages.

Parmi ces usages, la **santé connectée (e-santé)** est l'un des secteurs les plus critiques et prometteurs. L'intégration de capteurs biomédicaux communicants permet de passer d'une médecine réactive à une médecine préventive et personnalisée. Dans ce contexte, la surveillance cardiovasculaire est primordiale, les maladies cardiaques restant l'une des principales causes de mortalité dans le monde.

1.2 Problématique : Le défi du Holter connecté

L'électrocardiogramme (ECG) est l'examen de référence pour analyser l'activité électrique du cœur. Si l'ECG de repos en cabinet est utile, il ne permet pas de détecter des anomalies sporadiques (arythmies passagères) qui surviennent lors de la vie quotidienne du patient. Pour cela, les cardiologues utilisent un **Holter**, un dispositif portable qui enregistre l'ECG en continu sur 24 à 48 heures. Cependant, le Holter traditionnel présente une limitation

majeure : **l'absence de temps réel**. Les données sont stockées sur une carte mémoire et ne sont analysées par le médecin qu'une fois l'appareil rendu. Si une anomalie grave survient pendant le port de l'appareil, le médecin n'en est pas informé immédiatement.

La problématique technique centrale de ce projet réside dans la **tension entre mobilité et volume de données** :

- Transmettre l'intégralité d'un signal ECG haute fréquence en temps réel nécessite une bande passante importante (Wi-Fi, 4G), ce qui est très énergivore et limite l'autonomie du dispositif.

- Utiliser des réseaux basse consommation (LPWAN) comme LoRaWAN garantit une grande autonomie et une longue portée, mais la bande passante est trop faible pour transmettre la forme d'onde de l'ECG en continu.

1.3 Objectifs du Projet

Pour répondre à cette problématique, ce projet vise à concevoir et prototyper un **Holter connecté intelligent** reposant sur une architecture iot.

Le système développé propose deux niveaux de surveillance :

1. **Surveillance Locale (Temps Réel) :** Une station locale (basée sur Raspberry Pi) permet au patient ou au personnel soignant à proximité de visualiser la courbe ECG en haute définition et d'enregistrer l'historique complet sans perte de qualité.
2. **Télésurveillance (Alerte Distante) :** Le dispositif embarqué (basé sur Arduino LoRa) analyse le signal en bordure de réseau (*Edge Computing*). Il calcule le rythme cardiaque et n'envoie via le réseau LoRaWAN (The Things Network) que les indicateurs clés (BPM moyen) ou des alertes immédiates en cas d'anomalie critique (tachycardie, bradycardie).

2 État de l'art

Cette section dresse un panorama des solutions existantes pour la surveillance cardiaque et justifie les choix technologiques opérés pour ce prototype, tant au niveau du capteur biomédical que du protocole de communication.

2.1 Les dispositifs de surveillance cardiaque

2.1.1 Holters Classiques vs Dispositifs Connectés

Le Holter ECG traditionnel est un dispositif portatif permettant l'enregistrement continu de l'activité électrique cardiaque sur 24 à 48 heures. Bien qu'il constitue la norme clinique pour le diagnostic des arythmies, il présente une limitation majeure : son fonctionnement est **asynchrone**. Les données sont stockées localement (carte SD ou mémoire flash) et ne sont analysées par le cardiologue qu'après restitution de l'appareil. En cas d'événement cardiaque grave survenant durant l'enregistrement, aucune alerte n'est générée.

À l'inverse, les **Holters connectés (IoT)** représentent une évolution majeure vers la télémédecine. Ils intègrent des modules de communication sans fil permettant :

- La détection d'anomalies en temps réel.
- La transmission immédiate d'alertes aux services de santé.
- Une surveillance prolongée (plusieurs jours/semaines) grâce à une meilleure gestion de l'énergie.

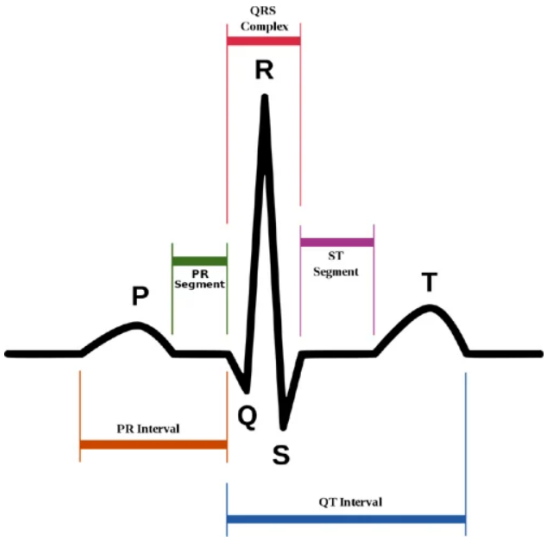
2.1.2 Signal ECG

Le signal ECG brut acquis par ce capteur présente une morphologie caractéristique composée des ondes P, Q, R, S et T :

- **Onde P :** Dépolarisation auriculaire. Son absence ou son aspect "en dents de scie" peut indiquer une fibrillation auriculaire.
- **Complexe QRS :** Contraction des ventricules. Le pic "R" est le point de repère principal pour calculer la fréquence cardiaque.
- **Onde T :** Repolarisation des ventricules.

L'analyse de l'intervalle R-R (temps entre deux pics R consécutifs) est cruciale pour déterminer la **Variabilité de la Fréquence Cardiaque (VFC ou HRV)**. Une variabilité trop faible

ou irrégulière est un indicateur précoce de pathologies comme la tachyarythmie ou le syndrome de Wolff-Parkinson-White.



2.2 Choix du protocole de communication : Pourquoi LoRaWAN ?

Le développement d'un objet connecté médical impose un compromis strict entre débit de données, portée et consommation énergétique.

2.2.1 Comparaison des protocoles sans fil

L'analyse comparative des technologies disponibles montre que les protocoles standards ne répondent pas totalement au besoin de mobilité :

- **Wi-Fi :** Offre un haut débit permettant de transmettre la courbe ECG brute, mais sa consommation électrique est prohibitive pour un appareil sur batterie (plusieurs centaines de mA en transmission). De plus, la portée est limitée à quelques dizaines de mètres.
- **Bluetooth (BLE) :** Très économe en énergie, mais nécessite une passerelle personnelle à proximité immédiate (smartphone du patient), ce qui ajoute une contrainte d'appairage et de dépendance matérielle.
- **LoRaWAN :** Ce protocole LPWAN (*Low Power Wide Area Network*) offre une consommation extrêmement faible (quelques mA en pointe, µA en veille) et une portée de plusieurs kilomètres.

Protocole	Portée	Débit	Autonomie
Wi-Fi	Courte (Indoor)	Très Haut	Faible (Jours)
Bluetooth LE	Très courte	Haut	Moyenne
LoRaWAN	**Longue (km)**	**Très Faible**	**Excellente (Mois)**

Table 1. Comparaison des protocoles IoT pour l'e-santé

2.2.2 Justification du choix LoRaWAN pour la santé

Le choix de LoRaWAN pour ce projet repose sur deux avantages décisifs pour un dispositif médical porté :

- **Pénétration Indoor ("Deep Indoor") :** Utilisant la bande de fréquence ISM 868 MHz (en Europe), les ondes LoRa traversent mieux les obstacles structurels (murs en béton, sous-sols) que le 2.4 GHz du Wi-Fi. Cela est critique pour suivre des patients à l'intérieur d'hôpitaux ou à leur domicile.
- **Autonomie énergétique :** La modulation LoRa permet d'atteindre des portées importantes avec une puissance d'émission minime, garantissant une surveillance sur le long terme sans recharge fréquente.
- **Contrainte induite :** Le faible débit de LoRa (quelques octets par message) interdit la transmission de la courbe ECG en temps réel. Cela impose une architecture de **Edge Computing** : l'Arduino doit analyser le signal, calculer le BPM localement, et n'envoyer que le résultat ou l'alerte via le réseau.

3 Analyse et Spécifications (Cahier des Charges)

La conception du Holter connecté repose sur une double exigence : fournir une analyse clinique précise tout en garantissant une surveillance continue à distance .

3.1 Besoins Fonctionnels Locaux (Station Patient)

La station locale, constituée du binôme Arduino-Raspberry Pi, agit comme une passerelle de diagnostic immédiat. Elle doit répondre aux exigences suivantes :

- **Visualisation ECG Temps Réel :** Le système doit afficher la forme d'onde complète.
- **Stockage et Historisation :** L'intégralité des mesures brutes doit être enregistrée localement (Base de données InfluxDB) pour permettre une lecture "a posteriori" (Replay). Cela pallie l'impossibilité d'envoyer tout l'historique via le réseau LoRaWAN.
- **Configuration des Seuils :** L'utilisateur doit pouvoir définir localement les seuils d'alerte (ex : Tachycardie > 100 BPM, Bradycardie < 50 BPM).

4 Gestion du Réseau LoRaWAN (The Things Stack)

Au cœur de notre infrastructure de communication se trouve **The Things Stack (TTS)**. Il s'agit du serveur réseau qui assure l'interface entre les passerelles physiques (Gateways) et notre plateforme de supervision cloud.

4.1 Rôle Stratégique dans le Projet

Dans l'architecture de notre Holter ECG, The Things Stack remplit trois fonctions critiques :

1. **Sécurité et Authentification :** Il gère la procédure OTAA (Over-The-Air Activation). Il vérifie les clés de sécurité (`AppKey`, `DevEUI`) pour s'assurer que seul notre capteur légitime peut envoyer des données.
2. **Déduplication :** Si le signal du capteur est capté par plusieurs passerelles aux alentours, TTS filtre les doublons pour ne transmettre qu'un seul paquet propre à l'application.
3. **Transformation de Données :** C'est ici que la conversion "Binaire → JSON" est effectuée, rendant les données exploitables par Node-RED et Grafana.

4.2 Le Décodage de Charge Utile (Payload Formatter)

Pour optimiser l'autonomie et le temps d'antenne (Airtime), l'Arduino n'envoie pas du texte, mais une suite de 6 octets hexadécimaux. Nous avons développé un script JavaScript personnalisé directement dans la console TTS pour décoder ce flux.

4.2.1 Analyse du Script de Décodage

Le script `decodeUplink` visible dans la configuration effectue les opérations suivantes sur le tableau `input.bytes` :

- **Reconstruction des Entiers (16 bits) :** L'Arduino découpe les valeurs en deux octets (MSB/LSB). Le script les réassemble par décalage de bits :

```
// Exemple pour le Rythme Cardiaque (Bytes 0 et 1)
const heartRate = (input.bytes[0] << 8) | input.bytes[1];
```

- **Correction d'Unité (HRV) :** Comme indiqué dans les commentaires du code, le capteur mesure la variabilité en microsecondes (μs). Pour une lecture médicale standard, une conversion est appliquée :

```
// Conversion microsecondes -> millisecondes
const hrv_ms = hrv_raw / 1000;
```

- **Sortie JSON :** Le script retourne un objet structuré contenant `heart_rate`, `hrv_ms`, et `ecg_value`.

4.3 Validation du Flux (Live Data)

L'onglet "Live Data" de la console TTS nous permet de valider le fonctionnement de la chaîne d'acquisition en temps réel.

Comme le montre la capture d'écran ci-dessous, nous observons le succès de la transmission :

- **Type d'événement :** "Forward uplink data message" indique que le paquet a été accepté et transmis vers l'application (via MQTT).
- **Payload Décrouvert :** La colonne "Data Preview" confirme que le décodage binaire fonctionne parfaitement. On y lit clairement les valeurs physiologiques : `{ "ecg_value": 263, "heart_rate": 102, "hrv_ms": 49.662 }`.

Cette validation confirme que les données brutes envoyées par le capteur sont correctement transformées en données cliniques avant même d'arriver sur nos serveurs de supervision.

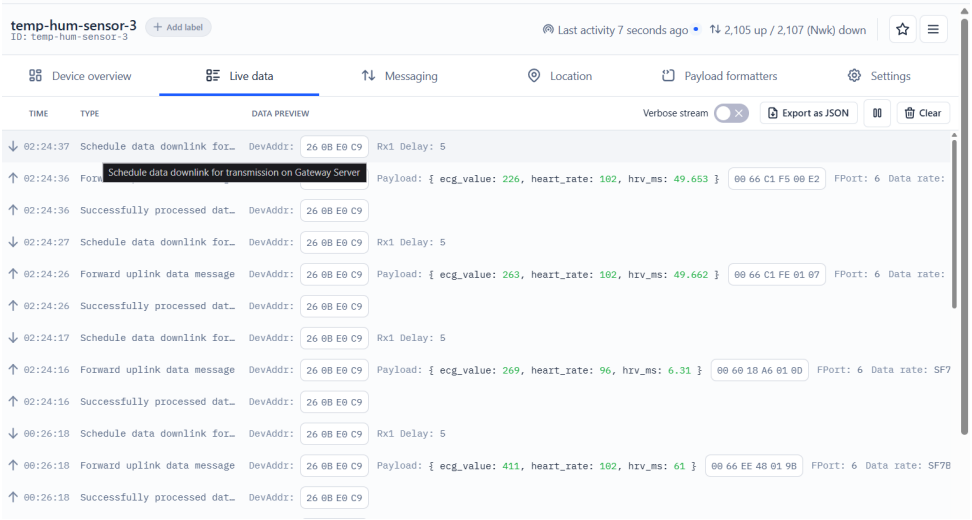


Figure 1. Console The Things Stack : Validation de la réception et du décodage JSON en temps réel

4.4 Besoins Fonctionnels Distants (Cloud & LoRaWAN)

Les fonctionnalités requises sont :

- **Remontée Périodique du BPM :** Transmission de la fréquence cardiaque moyenne calculée à intervalle régulier.
- **Gestion des Alertes (Push) :** En cas de dépassement de seuil une transmission doit être déclenchée *immédiatement*.

Fonctionnalité	Local (RPi)	Distant (LoRa)
Type de Données	Signal Brut (Waveform)	Métadonnées (BPM, Alerte)
Volume	Élevé (500 Bytes/sec)	Faible (5 Bytes/min)
Latence	Faible (Temps réel)	Moyenne (Classe A)
Usage	Diagnostic précis	Monitoring & Alerte

Table 2. Synthèse de la répartition fonctionnelle du système

5 Architecture Matérielle

Afin de valider le fonctionnement du Holter sans dépendre d'un sujet humain en permanence et pour garantir la répétabilité des mesures (scénarios de tachycardie, arythmie), nous avons mis en place une architecture de simulation matérielle.

Le système se compose de trois entités physiques distinctes : le Générateur de Signal (Simulateur), le Dispositif d'Acquisition (le Holter) et l'Infrastructure Réseau (Gateway).

5.1 Le Générateur de Signal ECG (Raspberry Pi 3)

Dans notre configuration, la Raspberry Pi 3 ne joue pas le rôle de passerelle, mais celui de **simulateur physiologique**. Elle remplace le cœur du patient.

5.1.1 Rôle et Fonctionnement

Son rôle est de convertir des données numériques (fichiers d'enregistrements cardiaques réels) en un signal électrique analogique comparable à celui capté par des électrodes.

- **Source de données :** Elle lit des fichiers au format `.csv` contenant des échantillons de signaux ECG réels.
- **Conversion Numérique-Analogique (DAC) :** la Raspberry Pi génère une tension variable simulant l'activité électrique cardiaque.
- **Interface :** Elle dispose d'une interface graphique (ou ligne de commande) permettant de sélectionner le type de pathologie à simuler (Rythme normal, Fibrillation, etc.).

5.2 Le Dispositif Holter (Arduino)

C'est le cœur de notre projet IoT. Il agit comme le dispositif médical porté par le patient.

5.2.1 Rôle et Interconnexion

L'Arduino The Things Uno est relié filairement à la sortie analogique de la Raspberry Pi (Générateur).

- **Acquisition (Entrée A0) :** Le microcontrôleur lit la tension générée par la Raspberry Pi sur son entrée analogique. Il "croit" lire un capteur réel.
- **Traitement :** Le code Arduino analyse ce signal en temps réel pour détecter les pics R (battements) et calculer la fréquence cardiaque (BPM).
- **Transmission (Module LoRa) :** Il encapsule les résultats (BPM, Alertes) et les transmet via son module radio.

5.3 La Passerelle LoRaWAN (LoRa Gateway)

La Gateway est l'élément d'infrastructure qui fait le pont entre le monde radio (IoT) et le monde Internet (IP).

5.3.1 Rôle et Caractéristiques

C'est un équipement physique (boîtier) placé à portée radio de l'Arduino.

- **Démodulation RF :** Elle écoute en permanence les ondes radio sur la bande 868 MHz. Lorsqu'elle capte le signal émis par notre Arduino (modulation LoRa), elle le démodule.
- **Packet Forwarder :** Elle ne décrypte pas et n'analyse pas les données. Son seul rôle est d'encapsuler le paquet radio dans une trame IP (UDP/TCP) et de l'envoyer au serveur réseau (The Things Network) via sa connexion Ethernet ou Wi-Fi.
- **Couverture :** Une seule Gateway peut couvrir plusieurs kilomètres et gérer des milliers de capteurs, ce qui permet de suivre le "patient" (l'Arduino) même s'il est déplacé loin du simulateur, tant qu'il reste alimenté.

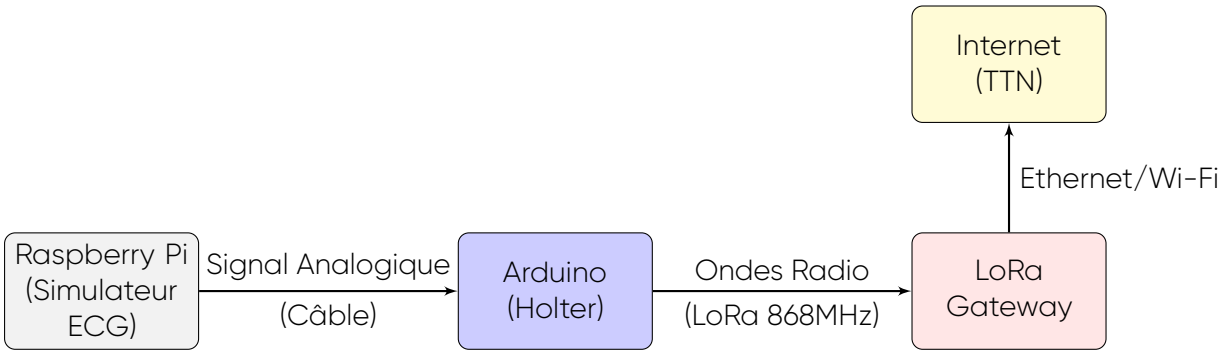


Figure 2. Schéma du banc de test : Simulation, Acquisition et Transmission

6 Architecture Logicielle et Implémentation

6.1 Firmware Embarqué (Arduino C++)

Le code implémenté dans l'Arduino ne se contente pas de relayer l'information ; il agit comme un filtre intelligent. Voici l'explication pas-à-pas de son fonctionnement, insérée dans le contexte du projet.

6.1.1 1. Initialisation et Synchronisation (Setup)

Avant le démarrage de la boucle principale, le système initialise ses repères temporel.

- `Serial.begin(9600);` : Ouvre le canal de communication vers la Raspberry Pi pour le transfert des données en temps réel.
- **Les Chronomètres :** Deux variables temporelles sont initialisées :
  - `instance1 (micros)` : Sert à mesurer la durée précise entre deux battements (pour le HRV).
  - `timer (millis)` : Sert à définir la fenêtre de calcul du rythme cardiaque (10 secondes).

6.1.2 2. Acquisition du Signal (Sampling)

Au début de chaque cycle `loop()`, l'instruction `value = analogRead(A0);` capture la tension instantanée générée par le simulateur ECG.

- **Contexte Projet :** Cette valeur brute (comprise entre 0 et 1023) représente l'activité électrique du cœur à l'instant t. C'est cette donnée qui permet de tracer la courbe "Waveform" sur le Dashboard local.

6.1.3 3. Mécanisme de Détection de Pic (Le Cœur de l'Algorithme)

C'est l'étape critique où le programme distingue un battement cardiaque du bruit de fond.

Le Seuil (Thresholding)

L'algorithme vérifie si le signal dépasse une valeur limite (`threshold = 240`).

$$\text{Si } (Value > 240) \Rightarrow \text{Battement Potentiel}$$

Le Verrouillage Logique (Flag Hysteresis)

Un signal ECG n'est pas un pic parfait d'une durée nulle ; il reste au-dessus du seuil pendant plusieurs millisecondes (environ 80-100ms pour un complexe QRS). Sans protection, la boucle (qui tourne très vite) compterait le même battement 10 ou 20 fois.

- **Mécanisme :** On utilise une variable booléenne `flag`.
- Dès que le seuil est dépassé, on passe `flag` à 1. Tant que le signal ne redescend pas sous le seuil (`else if value < threshold`), le système ignore les nouvelles lectures hautes.
- Cela garantit qu'un seul événement est compté par complexe QRS.

6.1.4 4. Calcul de l'Intervalle R-R (Précision Microseconde)

Lorsqu'un nouveau pic est validé, le code calcule immédiatement le temps écoulé depuis le dernier pic.

$$Intervalle = \text{TempsActuel}(\mu s) - \text{TempsPrecedent}(\mu s) \quad (1)$$

Cette valeur `interval` est fondamentale car elle mesure la régularité du cœur avant même de calculer une moyenne.

6.1.5 5. Calcul du Rythme Cardiaque (BPM)

Pour obtenir une fréquence cardiaque (BPM) stable sans attendre une minute entière à chaque mesure, le code utilise une fenêtre glissante de 10 secondes.

Mécanisme mathématique : Le programme compte les impulsions (`count`) pendant une période définie par `timer_value` (10 000 ms). Une fois les 10 secondes écoulées, le calcul est le suivant :

$$BPM = \text{Nombre de pics sur 10s} \times 6 \quad (2)$$

Exemple : Si on compte 12 battements en 10 secondes : $12 \times 6 = 72$ BPM. Le compteur est ensuite remis à zéro pour la fenêtre suivante.

6.1.6 6. Calcul de la Variabilité (HRV)

La variabilité de la fréquence cardiaque (HRV) est un indicateur de stress ou d'arythmie. Le code utilise une formule simplifiée pour estimer cette variation en temps réel.

Formule implémentée :

$$HRV = \left(\frac{BPM_{moyen}}{60}\right) - \left(\frac{Intervalle_{actuel}}{1\,000\,000}\right) \quad (3)$$

- Le terme $\frac{BPM}{60}$ donne la durée théorique moyenne d'un battement en secondes.
- Le terme $\frac{Intervalle}{1\,000\,000}$ convertit l'intervalle mesuré (en μs) en secondes.
- **Interprétation :** Si le résultat est proche de 0, le cœur bat exactement à la moyenne. Si la valeur oscille fortement, cela indique une arythmie.

Étape	Fonctionnalité	Action / Variable Clé
1	**Initialisation**	Démarrage du port Série et des chronomètres (`millis`, `micros`).
2	**Acquisition**	Lecture de la tension analogique sur la broche `A0`.
3	**Seuillage**	Comparaison du signal avec `threshold` (240).
4	**Anti-rebond**	Vérification et bascule du `flag` pour éviter le comptage multiple.
5	**Intervalle R-R**	Calcul précis du temps écoulé depuis le dernier pic (μs).
6	**Fenêtrage**	Vérification si la fenêtre de temps (10s) est écoulée.
7	**Calcul BPM**	Extrapolation du nombre de pics sur une minute ($count \times 6$).
8	**Calcul HRV**	Comparaison entre la durée théorique et la durée réelle du battement.
9	**Transmission**	Envoi des données formatées (CSV) sur le port Série et LoRaWAN.

Table 3. Résumé séquentiel de l'algorithme de traitement embarqué

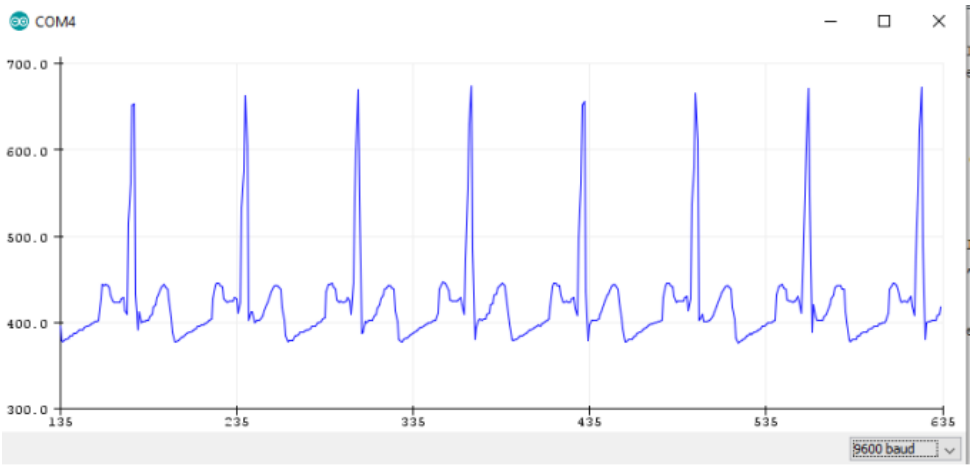


Figure 3. Évolution du nombre d'objets connectés dans le monde (Source : Cours IoT)

7 Infrastructure Serveur et Décodage (The Things Stack)

Une fois le signal LoRa émis par le dispositif Holter, il transite par une infrastructure complexe avant d'être exploitable. Nous utilisons **The Things Stack (TTS)**, une implémentation robuste du standard LoRaWAN.

7.1 Architecture de The Things Stack (TTS)

Comme illustré dans le synoptique global du système, la chaîne de transmission se décompose en plusieurs entités logiques distinctes.

7.1.1 1. La Passerelle (Gateway)

La Gateway est le pont physique entre le protocole radio (LoRa) et le protocole IP (Internet).

- **Rôle :** Elle écoute le spectre 868 MHz. Lorsqu'elle capte une trame valide provenant de notre Arduino, elle encapsule le paquet radio dans un paquet IP (UDP ou TCP) et l'envoie au serveur réseau.

- **Transparence :** La Gateway est "bête" (dumb pipe) : elle ne connaît pas les clés de chiffrement (AppSKey) et ne peut pas lire le contenu médical (BPM, ECG). Elle ajoute simplement des métadonnées de transmission : le *RSSI* (puissance du signal) et le *SNR* (rapport signal/bruit).

7.1.2 2. Le Network Server (NS)

Le serveur réseau est le chef d'orchestre de la communication LoRaWAN.

- **Dé-duplication :** Si plusieurs Gateways reçoivent le même message du Holter, le NS filtre les doublons pour ne garder que le paquet ayant le meilleur signal.
- **Contrôle MAC :** Il gère l'authentification (OTAA) et peut envoyer des commandes au dispositif pour adapter sa vitesse de transmission.

7.1.3 3. L'Application Server (AS)

C'est le composant qui nous intéresse le plus pour le traitement des données.

- **Déchiffrement :** Il possède la clé de session applicative (*AppSKey*) qui lui permet de déchiffrer la charge utile (Payload).
- **Décodage :** Il exécute le *Payload Formatter* pour transformer les octets bruts en objets JSON.
- **Intégration :** Il pousse les données finales via MQTT vers notre tableau de bord Node-RED.

7.2 Protocole de Communication MQTT

Le MQTT (*Message Queuing Telemetry Transport*) est un protocole de messagerie standardisé ISO, conçu spécifiquement pour l'Internet des Objets (IoT). Il est réputé pour sa légèreté et sa faible consommation de bande passante.

7.2.1 Architecture Publier/Abonner

Contrairement au modèle web classique (Client-Serveur) où le client doit demander l'information, MQTT repose sur un modèle événementiel appelé **Publier/Abonner** (*Publish/-Subscribe*). Trois acteurs interviennent :

- **Le Broker :** C'est le serveur central (le "facteur").
- **Le Publisher (Éditeur) :** C'est la source de la donnée.
- **Le Subscriber (Abonné) :** C'est le destinataire (ici, Node-RED).

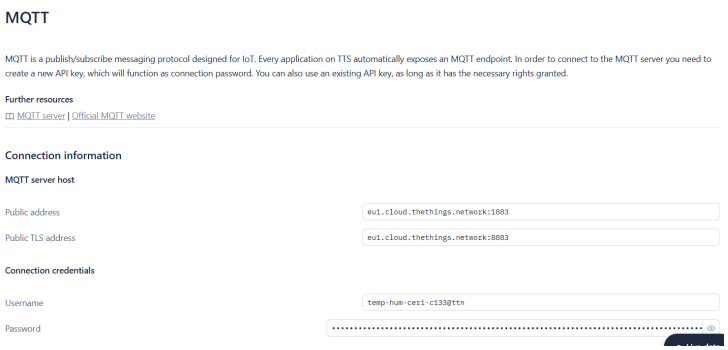
7.2.2 Application au Projet

Dans notre architecture de supervision distante :

1. Le serveur LoRaWAN (TTN) agit comme un pont. Dès qu'il reçoit un paquet radio de l'Arduino, il le publie sur un *Topic* sécurisé `v3/user/devices/sensor/up`.
2. Notre conteneur Node-RED est configuré comme **Subscriber**. Il écoute ce topic en permanence.
3. Dès qu'une donnée arrive, Node-RED la reçoit en temps réel sans avoir besoin d'interroger le serveur en boucle, garantissant une latence minimale pour l'affichage médical.

7.3 Le Payload Formatter (Décodeur Uplink)

Pour économiser le "temps d'antenne" (Airtime) et respecter les contraintes du réseau LoRaWAN, nous n'envoyons pas de texte (JSON ou ASCII) depuis l'Arduino. Nous utilisons un encodage binaire (*Byte Packing*).



L'Application Server doit donc effectuer l'opération inverse : reconstruire les nombres à partir des octets reçus.

7.3.1 Analyse du Script de Décodage

Nous utilisons un décodeur JavaScript personnalisé (*Custom Javascript formatter*) configuré dans la console The Things Stack.

Le script reçoit un tableau d'octets (`input.bytes`) et doit retourner un objet JSON.

1. **Vérification de l'intégrité :** Le code vérifie d'abord que le paquet reçu fait exactement 6 octets. Si ce n'est pas le cas, il retourne une erreur, ce qui évite de traiter des paquets corrompus.

```
if (input.bytes.length !== 6) {
    return { errors: ["Payload length must be 6 bytes"] };
}
```

2. **Reconstruction des Entiers (Bitwise Operations) :** L'Arduino a découpé nos valeurs de 16 bits en deux octets de 8 bits . Le décodeur les rassemble en utilisant un décalage de bits (`<< 8`) et une opération OU logique (`|`).
   - **Heart Rate (Octets 0 et 1) :** `const heartRate = (input.bytes[0] << 8) | input.bytes[1];`
   - **HRV Brut (Octets 2 et 3) :** `const hrv_raw = (input.bytes[2] << 8) | input.bytes[3];`
   - **Valeur ECG (Octets 4 et 5) :** `const ecg = (input.bytes[4] << 8) | input.bytes[5];`
3. **Conversion d'Unités (Post-Processing) :** Une étape critique a été ajoutée pour la cohérence des données médicales. Le microcontrôleur mesure le temps entre deux battements en *microsecondes* (μs) pour une précision maximale. Cependant, pour l'affichage humain, les *millisecondes* (ms) sont préférables.

```
// capteur envoie en microsecondes, pas millisecondes
// Diviser par 1000 pour convertir us -> ms
const hrv_ms = hrv_raw / 1000;
```

4. **Sortie JSON :** Le formateur retourne finalement un objet structuré qui sera transmis via MQTT :

```
return {
    data: {
        heart_rate: heartRate, // en BPM
        hrv_ms: hrv_ms,     // en ms (converti)
        ecg_value: ecg      // valeur ADC brute
    }
};
```

8 Supervision Locale Avancée (Node-RED & InfluxDB)

La station locale (Raspberry Pi 3) constitue le "cerveau" analytique du système Holter. Elle ne se contente pas d'afficher les données brutes ; elle orchestre l'acquisition, le traitement

```javascript
function decodeUplink(input) {
  if (input.bytes.length !== 6) {
    return {
      errors: ["Payload length must be 6 bytes"]
    };
  }

  const heartRate = (input.bytes[0] << 8) | input. bytes[1];
  const hrv_raw = (input.bytes[2] << 8) | input.bytes[3];
  const ecg = (input.bytes[4] << 8) | input.bytes[5];

  // ✅ CORRECTION: Ton capteur envoie en microsecondes, pas millisecondes
  // Diviser par 1000 pour convertir µs → ms
  const hrv_ms = hrv_raw / 1000;

  return {
    data: {
      heart_rate: heartRate,          // en BPM
      hrv_ms: hrv_ms,                 // ✅ MAINTENANT en millisecondes (converti)
      ecg_value: ecg                  // valeur ADC
    }
  };
}
```

Figure 4. Interface de l'Application Server : Code du Payload Formatter Uplink

algorithmique, le stockage historique et la gestion intelligente des alertes.

Cette architecture repose sur l'interaction entre **Node-RED** (orchestration événementielle) et **InfluxDB** (base de données de séries temporelles).

8.1 Architecture du Flux de Données (Flow Node-RED)

Une architecture divisée en cinq blocs fonctionnels majeurs, interconnectés de manière asynchrone.

8.1.1 1. Acquisition et Normalisation (Ingest)

Le point d'entrée du système est le nœud **MQTT In**.

- **Source :** Il s'abonne au topic LoRaWAN Uplink : `v3/temp-hum-ceri-c133@ttn/devices/temp-hum-sensor-3/up`.
- **Données :** Il reçoit un objet JSON complexe contenant la charge utile décodée (`decoded_payload`) ainsi que les métadonnées réseau (RSSI, SNR).

8.2 2. Traitement Algorithmique (Le Processeur de Données)

Le cœur de l'intelligence locale réside dans le nœud de fonction **"Data Processor"**. Ce script JavaScript exécute trois opérations critiques à chaque message reçu.

8.2.1 A. Classification Clinique

Le code analyse le rythme cardiaque (HR) et attribue un statut clinique et une couleur de sévérité, utilisés ensuite par le Dashboard et le système d'alerte.

```javascript

if (hr < 40) {
    status = 'SEVERE BRADYCARDIA';
    severity = 'critical';
} else if (hr < 60) {
    status = 'Bradycardia';
} else if (hr > 150) {
    status = 'SEVERE TACHYCARDIA';
    severity = 'critical';
```

```
}
```

Listing 1. Logique de classification cardiaque

8.2.2 B. Lissage par Moyenne Glissante (Rolling Average)

Pour éviter l'instabilité de l'affichage, le processeur maintient un tampon circulaire des 50 dernières mesures dans la mémoire contextuelle (`flow.get`).

```
// Récupération de l'historique en mémoire
let history = flow.get('ecg_history') || [];
history.push({ hr: hr, ecg: ecg, hrv: hrv });

if (history.length > 50) history.shift();
flow.set('ecg_history', history);

// Calcul de la moyenne
const avg = (arr, key) => arr.reduce((s, v) => s + v[key], 0) / arr.length;
const hr_avg = Math.round(avg(history, 'hr') * 10) / 10;
```

8.3 3. Système d'Alerte Intelligent (Anti-Spam)

La gestion des notifications par email est gérée par une machine à états finis implémentée dans le nœud **"Alert Filter"**.

8.3.1 Filtrage par Changement d'État (Edge Triggering)

Pour éviter de saturer la boîte mail du médecin lorsqu'un patient reste en état pathologique (ex : Tachycardie persistante), le système n'envoie une alerte que lors d'une **transition d'état**.

```
const currentStatus = msg.payload.status;
const lastStatus = flow.get('last_alert_status') || 'NORMAL';

// Si l'état n'a pas changé, on arrête le flux (pas d'email)
if (currentStatus === lastStatus) {
    node.warn(`Même état (${currentStatus}), alerte ignorée`);
    return null;
}

// Sinon, on met à jour l'état et on autorise l'envoi
flow.set('last_alert_status', currentStatus);
return msg;
```

Listing 2. Algorithme de filtrage d'alerte

8.3.2 Génération Dynamique d'Email (HTML)

Le nœud **"Format Email"** génère un courriel responsive. Il injecte dynamiquement des styles CSS (couleur de fond rouge ou orange) selon la variable `severity` et construit un tableau récapitulatif des signes vitaux au moment de l'incident.

8.4 4. Stratégie de Stockage (InfluxDB)

La persistance des données est assurée par le nœud **"InfluxDB Out"**. Contrairement à une base SQL, InfluxDB est optimisée pour l'écriture massive de données temporelles.

Schéma de Données :

- **Measurement :** `ecg_data`
- **Tags (Index) :**

- `record_id` : Identifiant unique de session (`YYYYMMDD_HHMMSS`) généré par le Data Processor.
- `status` : Permet de requêter ultérieurement par pathologie (ex : "SELECT * WHERE status='Tachycardia'").

- **Fields (Valeurs) :** `heart_rate`, `hrv_ms`, `ecg_value`.

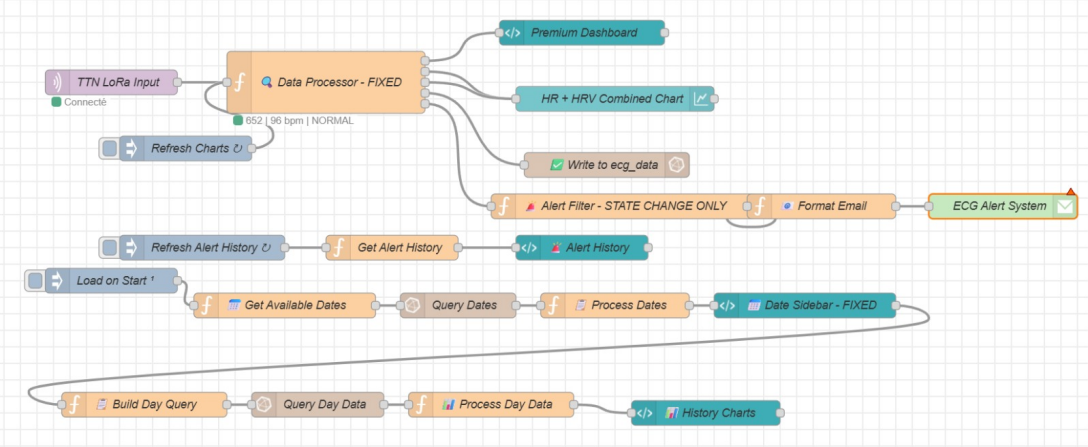


Figure 5. Interface de supervision : Mode Relecture et Analyse Historique

8.5 Dictionnaire des Données (Attributs Stockés)

L'analyse de la table `ecg_data` révèle la structure exacte des données persistées. Voici la définition et l'utilité de chaque colonne stockée :

`time` (Timestamp) L'index primaire généré automatiquement par InfluxDB. Il représente l'instant précis de la réception de la donnée avec une précision à la nanoseconde (ex : `176739...`). C'est la clé de voûte pour l'affichage chronologique des graphiques.

`heart_rate` (Integer) La fréquence cardiaque calculée par l'Arduino en Battements Par Minute (BPM). *Utilité :* Permet de détecter les pathologies (Tachycardie > 100, Bradycardie < 60) et de générer les alertes.

`ecg_value` (Integer) La valeur brute numérisée du signal analogique (ADC 0-1023). *Utilité :* Indispensable pour la fonction "Replay". Elle permet de redessiner la forme d'onde (complexe QRS) a posteriori pour qu'un médecin puisse valider visuellement une anomalie.

`hrv_ms` (Integer) La Variabilité de la Fréquence Cardiaque exprimée en millisecondes. *Utilité :* Indicateur de stress physiologique. Une valeur basse indique un rythme cardiaque très (trop) régulier, souvent signe de stress ou de fatigue.

`year`, `month`, `day_of_month` (Integers) L'année, le mois et le jour de la mesure, stockés dans des colonnes distinctes. *Utilité :* Ces champs, bien que redondants avec le `time`, servent d'index pour accélérer les recherches dans la barre latérale "Historique". Ils permettent de lister instantanément les jours disponibles sans avoir à parser des millions de timestamps.

`hour`, `minute`, `second` (Integers) L'éclatement de l'heure. *Utilité :* Facilite le filtrage horaire (ex : "Montrer toutes les crises survenues entre 22h et 23h") sans calculs complexes côté base de données.

8.6 5. Interface de Relecture Historique (Replay)

Une fonctionnalité avancée du dashboard permet d'explorer les enregistrements passés. Cette logique repose sur la construction dynamique de requêtes InfluxQL (SQL-like).

```
> SELECT * FROM ecg_data ORDER BY time DESC LIMIT 10
name: ecg_data
time                day_of_month ecg_value heart_rate hour hrv_ms minute month second value year
----                ------------ --------- ---------- ---- ------ ------ ----- ------ ----- ----
1767390045033000000 2            229       96         22   17     40     1     44           2026
1767390035034000000 2            231       96         22   28     40     1     34           2026
1767390025032000000 2            216       96         22   6      40     1     24           2026
1767390015044000000 2            189       96         22   28     40     1     14           2026
1767390005051000000 2            185       96         22   39     40     1     4            2026
1767389995054000000 2            179       90         22   50     39     1     54           2026
1767389985068000000 2            334       102        22   39     39     1     44           2026
1767389975062000000 2            212       96         22   17     39     1     34           2026
1767389965070000000 2            187       96         22   28     39     1     24           2026
1767389955061000000 2            184       90         22   61     39     1     14           2026
```

Figure 6. Interface de supervision : Mode Relecture et Analyse Historique

8.6.1 A. Navigation Temporelle

Le nœud **"Get Available Dates"** interroge la base pour lister les jours contenant des données : `SELECT COUNT(heart_rate) FROM ecg_data GROUP BY time(1d)`. Le résultat alimente la barre latérale ("Sidebar") du Dashboard.

8.6.2 B. Reconstruction de Session

Lorsqu'une date est sélectionnée, le nœud **"Build Day Query"** calcule les bornes temporelles exactes en nanosecondes (format natif InfluxDB) pour extraire la journée complète.

```
// Conversion Date -> Nanosecondes
const start_nano = start_of_day.getTime() * 1000000;
const end_nano = end_of_day.getTime() * 1000000;

// Construction de la requête
msg.query = `SELECT heart_rate, hrv_ms, ecg_value
            FROM ecg_data
            WHERE time >= ${start_nano} AND time <= ${end_nano}
            ORDER BY time ASC`;
```

9 Supervision Clinique : Tableaux de Bord et Alertes

L'interface utilisateur (IHM) constitue le point de contact final entre le dispositif technique et le personnel médical. Elle a été conçue pour offrir une lecture immédiate de l'état du patient ("Temps Réel") tout en permettant une analyse a posteriori ("Historique").

9.1 Visualisation Temps Réel

Le tableau de bord principal offre une vue synthétique des signes vitaux instantanés. Il agrège les données provenant du flux MQTT en moins d'une seconde.

Les composants graphiques :

- **Jauges Numériques :** Affichage du Rythme Cardiaque (BPM) et de la Variabilité (HRV) avec un code couleur dynamique (Vert = Normal, Rouge = Critique).
- **Graphique Combiné :** Un tracé linéaire superpose l'évolution du BPM et du HRV sur les 5 dernières minutes, permettant de corréler une hausse du rythme cardiaque avec une chute de la variabilité (signe de stress physiologique).
- **Onde ECG (Waveform) :** Le tracé du signal brut permet au cardiologue de vérifier la présence du complexe QRS et de valider la qualité de la pose des électrodes.

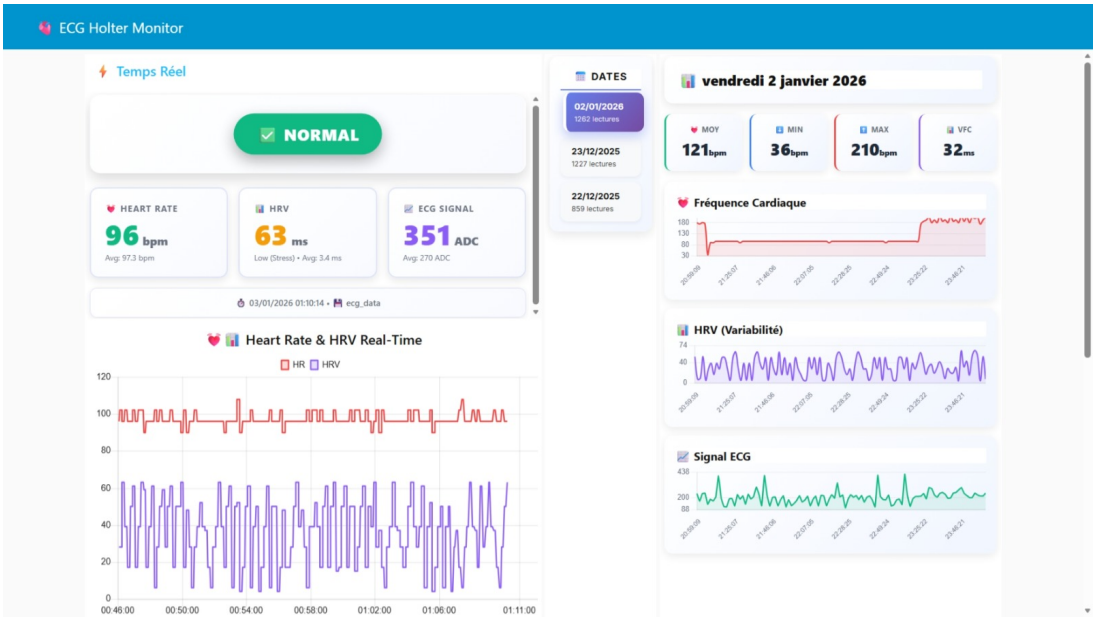


Figure 7. Tableau de bord temps réel : Visualisation simultanée BPM, HRV et Signal ECG

9.2 2. Système de Notification (Alerte Email)

La surveillance passive est assurée par un système de notification "Push". Lorsqu'une anomalie est détectée par la machine à états (ex : Tachycardie soudaine), un courriel riche est généré et envoyé au médecin.

Caractéristiques du rapport d'alerte :

- **Code Visuel :** L'en-tête du mail change de couleur (Orange pour Avertissement, Rouge pour Critique) pour une identification immédiate de l'urgence.
- **Contexte :** Le mail contient l'horodatage précis, l'ID de l'enregistrement et les valeurs moyennes au moment de la crise.
- **Lien :** Il invite le praticien à se connecter au Dashboard pour analyser l'historique complet.

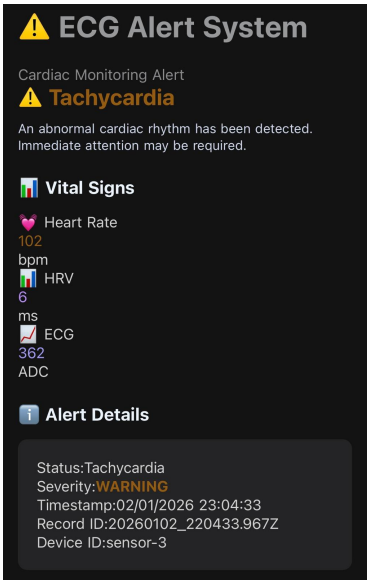


Figure 8. Exemple de notification critique reçue par le personnel soignant

9.3 3. Exploration de l'Historique (Mode Replay)

Pour dépasser la simple surveillance instantanée, l'onglet "Historique" permet de rejouer des séquences passées stockées dans InfluxDB.

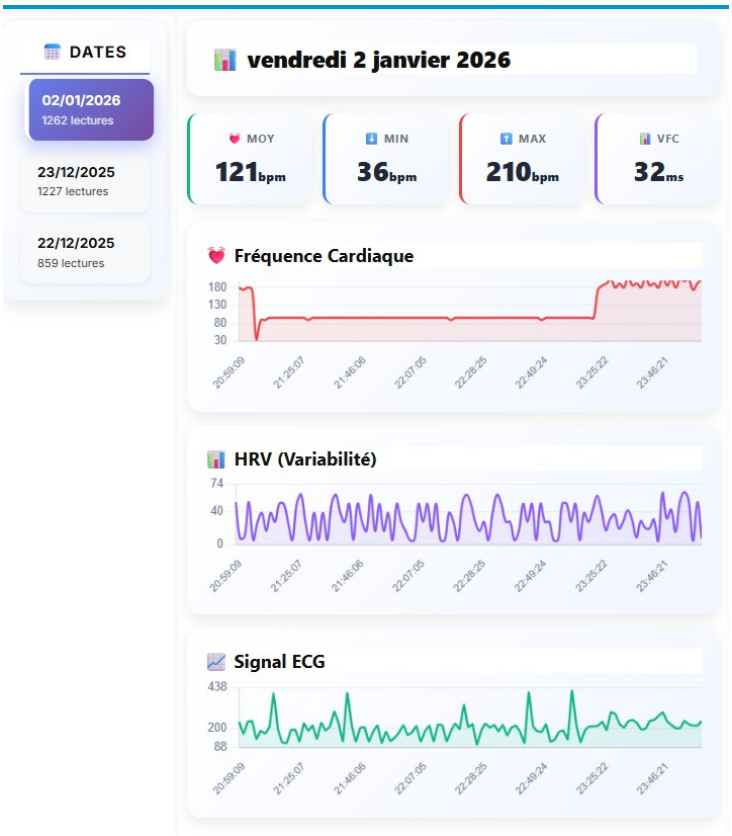


Figure 9. Interface d'historique : Sélection par date et analyse post-crise

10 Supervision Distante et Industrialisation (Architecture Cloud Conteneurisée)

Cette section détaille la migration de notre infrastructure de supervision vers une architecture micro-services conteneurisée, permettant un déploiement reproductible sur tout environnement (Raspberry Pi local, serveur hospitalier, ou cloud public).

10.1 Justification de l'Approche Conteneurisée

L'utilisation de conteneurs Docker répond à plusieurs exigences critiques pour un système de télésurveillance médicale :

- **Isolation des Services :** Chaque composant (logique métier, stockage, visualisation) s'exécute dans son propre environnement cloisonné, éliminant les conflits.
- **Reproductibilité :** L'infrastructure complète est définie "as code". Un nouveau déploiement s'effectue en une seule commande, garantissant une configuration identique entre l'environnement de développement et la production.
- **Persistance des Données Médicales :** Les volumes Docker détachent le cycle de vie des données de celui des conteneurs. Une mise à jour logicielle ne risque jamais d'effacer l'historique ECG des patients.
- **Portabilité Multi-Plateforme :** La même stack peut tourner sur un Raspberry Pi 4 (ARM64), un serveur Linux x86, ou un cluster Kubernetes hospitalier sans modification.

10.2 Architecture Micro-Services : Les Trois Piliers

L'infrastructure repose sur trois services orchestrés par Docker Compose, chacun remplissant un rôle spécifique dans la chaîne de traitement des données cardiaques.

10.2.1 Service 1 : InfluxDB — Le Stockage Temporel

Rôle : Base de données spécialisée dans les séries temporelles.

10.2.2 Service 2 : Node-RED — Le Moteur Logique

Rôle : Orchestrateur de flux (Flow-Based Programming). Il agit comme middleware entre la couche IoT (LoRaWAN/MQTT) et la couche de stockage/visualisation.

Configuration Technique :

- **Persistance des Flux :** Trois volumes critiques sont montés :
  1. `./node_red_data:/data` — Répertoire général (configuration globale, bibliothèques installées).
  2. `./flows.json:/data/flows.json` — Définition des flux logiques (graphe de nœuds).
  3. `./flows_cred.json:/data/flows_cred.json` — Credentials chiffrés (clés API TTN, tokens MQTT).
- **Réseau Docker :** Le paramètre `links: influxdb` et `depends_on: influxdb` garantissent que Node-RED démarre après InfluxDB et peut le résoudre par son nom de service (`http://influxdb:8086`).

10.2.3 Service 3 : Grafana — L'Interface de Supervision Premium

Rôle : Front-end de visualisation temps réel et d'analyse historique, remplaçant le dashboard Node-RED par une interface de niveau industriel.

Configuration Technique :

- **Image :** `grafana/grafana:latest` (version 10.x avec support des transformations avancées).
- **Port Exposé :** 3000 (interface Web).

10.3 Fichier d'Orchestration Complet (docker-compose.yml)

Le fichier suivant définit l'infrastructure complète. Il constitue le *blueprint* reproductible du système :

```yaml
version: '3.8'

services:
  # Service 1 : Base de Données Temporelle
  influxdb:
    image: influxdb:1.8
    container_name: holter_influxdb
    restart: always
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=heart
    volumes:
      - ./influxdb_data:/var/lib/influxdb

  # Service 2 : Moteur Logique
  node-red:
    build: . # Utilise le Dockerfile local
```


```
    container_name: holter_nodered
    restart: always
    user: root
    ports:
      - "1880:1880"
    environment:
      - TZ=Europe/Paris
      - FLOWS=flows. json
    volumes:
      - ./node_red_data:/data
      - ./flows.json:/data/flows.json
      - ./flows_cred.json:/data/flows_cred.json
    links:
      - influxdb
    depends_on:
      - influxdb

  # Service 3 : Visualisation Premium
  grafana:
    image: grafana/grafana:latest
    container_name: holter_grafana
    user: "0" # Correctif permissions Windows
    restart: always
    ports:
      - "3000:3000"
    volumes:
      - grafana_data_vol:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning
      - ./grafana/dashboards:/var/lib/grafana/dashboards
    links:
      - influxdb
```

```
    depends_on:
      - influxdb
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin

volumes:
  grafana_data_vol: # Volume nommé pour persistance Grafana
```

Listing 3. Configuration Docker Compose Complète

10.4 Flux de Données dans l'Architecture Cloud

Le cheminement de l'information depuis le capteur ECG jusqu'à l'interface médicale suit cinq étapes :

1. **Acquisition (Edge Device) :** Le capteur AD8232 échantillonne le signal ECG à 250 Hz. Le microcontrôleur STM32 détecte les pics R et calcule la fréquence cardiaque instantanée.
2. **Transmission (LoRaWAN) :** Les données sont encapsulées dans des trames LoRaWAN (payload de 11 octets) et transmises via le module RAK3172 au réseau The Things Network.
3. **Ingestion Cloud (MQTT → Node-RED) :** Le conteneur Node-RED s'abonne au topic MQTT de TTN (`v3/{app_id}/devices/{dev_id}/up`). Chaque message reçu contient :
   - `payload_raw` : Données binaires encodées en Base64.
   - `metadata` : RSSI, SNR, timestamp d'arrivée.
4. **Traitement (Node-RED Flow) :**
   - **Décodage :** Fonction JavaScript convertit le payload Base64 en valeurs numériques (HR, HRV, ECG).
   - **Détection d'Anomalies :** Algorithme de classification :

$$\text{Status} = \begin{cases} \text{"Bradycardie"} & \text{si } HR < 60 \text{ BPM} \\ \text{"Tachycardie"} & \text{si } HR > 100 \text{ BPM} \\ \text{"Normal"} & \text{sinon} \end{cases}$$

5. **Persistance (InfluxDB Write) :** Le nœud *InfluxDB Out* écrit les données structurées :

```
{
  "measurement": "ecg_data",
  "tags": {
    "device_id": "holter_patient_001",
    "status": "Tachycardie"
  },
  "fields": {
    "heart_rate": 112,
    "hrv": 35,
    "ecg_value": 1.87
  },
  "timestamp": 1735920543000000000
}
```

Listing 4. Structure de Donnée Écrite dans InfluxDB

6. **Visualisation (Grafana Query) :** Grafana interroge InfluxDB toutes les 500 ms avec des requêtes InfluxQL optimisées :

```
SELECT mean("heart_rate")
FROM "ecg_data"
WHERE time > now() - 5m
GROUP BY time(10s) fill(linear)
```

Listing 5. Exemple de Requête Temps Réel

11 Conclusion et Bilan du Projet

11.1 Synthèse des Réalisations (Pas à Pas)

Le tableau ci-dessous résume les étapes techniques franchies pour aboutir au prototype fonctionnel final.

Phase	Réalisations Techniques	Résultat Obtenu
1. Simulation & Acquisition	- Mise en place du générateur de signal (Raspberry Pi). - Connexion Analogique vers Arduino (ADC). - Développement Firmware C++ (Sampling).	Acquisition d'un signal ECG brut stable simulant un patient réel.
2. Edge Computing	- Implémentation de l'algorithme de détection de pics (Thresholding + Hystérésis). - Calcul mathématique du BPM et VFC (HRV).	Transformation d'une donnée brute massive en métadonnées légères (Smart Data).
3. Connectivité LoRaWAN	- Configuration The Things Network (OTAA, AppEUI). - Script de décodage Payload Formatter (JS).	Transmission longue portée basse consommation.
4. Backend & Logique	- Connexion MQTT sécurisée. - Développement des flux Node-RED. - Création de la machine à états pour la gestion des alertes emails.	Réception temps réel (< 2s) et tri intelligent des alertes critiques.
5. Stockage	- Déploiement InfluxDB. - Modélisation du schéma .	Historisation persistante permettant la relecture post-diagnostic.
6. Visualisation et DevOps	- Design du Dashboard Grafana. - Conteneurisation de la stack (Docker Compose). - Provisioning.	Interface professionnelle déployable sur n'importe quel serveur en une commande.

Table 4. Tableau récapitulatif des étapes de réalisation du projet

11.2 Conclusion Générale

Ce projet de Holter Connecté démontre qu'il est aujourd'hui possible de concevoir des dispositifs médicaux de haute précision en utilisant des technologies standardisées et ouvertes (Arduino, LoRaWAN, Docker).

L'architecture hybride proposée (Traitement local + Supervision Cloud) répond parfaitement à la problématique initiale : concilier la mobilité du patient avec la nécessité d'une surveillance médicale rigoureuse. Le prototype final, validé par simulation, est une preuve de concept solide prête pour une éventuelle phase d'industrialisation.