



Université Sorbonne Paris Nord

---

# **Système de surveillance cardiaque en temps réel :**

**Architecture streaming avec Kafka et Standard FHIR**

---

*Réalisé par:*

**CUNHA GRILLO LARA**

**ALAOUI MRANI Salma**

**RAFIDISON Nirinah Vanilla**

*Sous la direction de:*

**Monsieur MIHOUBI Mohamed**

*Spécialité: M1 Big Data, Analyse, Business Intelligence*

*Année universitaire: 2025-2026*

# Contenu

## 1. Introduction générale

- 1.1. *Contexte du projet*
- 1.2. *Pourquoi le standard FHIR*
- 1.3. *Objectifs de notre réalisation*

## 2. Infrastructure technique et ingénierie du flux de données

- 2.1. *Configuration de l'Infrastructure via Docker*
- 2.2. *Rôle du Producer dans l'architecture*
- 2.3. *Mise en place de Kafka côté producer*
- 2.4. *Génération des données patient*
- 2.5. *Simulation de valeurs de pression artérielle réalistes*
- 2.6. *Structure JSON FHIR envoyée sur Kafka*
- 2.7. *Logique d'envoi continu sur Kafka*

## 3. Analyse et Traitement des Données consumer

- 3.1. *Rôle du consumer dans le pipeline de données*
- 3.2. *Configuration du Consumer et abonnement au topic Kafka*
- 3.3. *Extraction et lecture des données FHIR*
- 3.4. *Détection automatique des anomalies*
- 3.5. *Traitement et orientation des données*
- 3.6. *Indexation dans Elasticsearch*

## 4. Analyse BI & Visualisation (Kibana Expert)

## 5. Synthèse : Les Forces de l'Architecture Big Data

## 6. Conclusion

## 1. Introduction Générale

### 1.1. Contexte du Projet

Dans le cadre de notre cursus, nous avons été chargés de concevoir un système capable de surveiller en temps réel la pression artérielle des patients. L'enjeu est de taille : dans un milieu hospitalier, chaque seconde compte pour détecter une anomalie critique, comme une crise hypertensive. Notre projet simule donc une chaîne complète de traitement de données de santé, du capteur au chevet du patient jusqu'au tableau de bord du médecin. Notre objectif est de développer une solution complète pour analyser et traiter les données de pression artérielle afin d'améliorer le suivi médical des patients.

### 1.2. Pourquoi le standard FHIR ?

Plutôt que d'utiliser un format de données propriétaire, nous avons choisi de nous appuyer sur le standard international FHIR. C'est ce qui rend notre projet réaliste : les messages que nous générerons (ressources "Observation") sont compatibles avec les systèmes de santé modernes à travers le monde. Cela garantit que nos données sont non seulement structurées, mais aussi interopérables.

Le script Message\_FHIR\_Project.py produit des messages FHIR permettant de simuler les mesures cliniques des patients de notre étude. Chaque message contient l'identifiant du patient, ses valeurs de pression artérielle systolique et diastolique, ainsi que d'autres informations contextuelles liées à son suivi, en respectant la structure et les standards définis par FHIR.

### 1.3 Objectifs spécifiques de notre réalisation

Notre équipe s'est fixée pour mission de monter une infrastructure Big Data capable de gérer quatre étapes clés :

- **La Simulation** : Créer un flux de données crédible pour 5 patients fictifs via Python.
- **Le Streaming** : Faire voyager ces données instantanément grâce à un cluster Kafka.
- **L'Analyse** : Trier automatiquement les patients en fonction de leur état de santé (Normal ou Critique).
- **La Visualisation** : Offrir une interface claire à l'aide d'Elasticsearch et Kibana pour que le personnel soignant puisse agir immédiatement sur les cas urgents.

## 2. Infrastructure Technique et Ingénierie du Flux de Données

La première étape de notre architecture repose sur la capacité à injecter un flux continu de données médicales structurées dans le système de streaming. Dans ce cadre, nous avons conçu et implémenté le Producer Kafka, une brique logicielle en Python agissant comme le point d'entrée vital de l'infrastructure.

### 2.1. Configuration de l'Infrastructure via Docker

Avant de développer le script d'envoi, j'ai mis en place l'environnement technique nécessaire au projet. Nous avons fait le choix de la conteneurisation pour assurer la portabilité du système:

- **Orchestration avec Docker Compose** : nous avons configuré un fichier docker-compose.yml pour lancer simultanément les services Zookeeper, Kafka, Elasticsearch et Kibana.
- **Isolation des services** : Cette méthode nous a permis de travailler sur un cluster local standardisé sans risque de conflits avec les configurations personnelles de nos machines.
- **Validation** : Le bon fonctionnement a été validé par la commande docker ps, confirmant que le broker Kafka était prêt à recevoir des données sur le port 9092.

Pour simuler le fonctionnement de Kafka, nous avons développé deux fichiers distincts, Producer et Consumer, représentant respectivement le producteur et le consommateur de messages. Cette séparation des rôles, plutôt que de centraliser tout le traitement dans un seul fichier, permet de reproduire plus fidèlement le flux de données en temps réel et de clarifier la structure du système.

## 2.2. Rôle du Producer dans l'architecture

Dans l'architecture de surveillance temps réel de la pression artérielle, le Producer joue le rôle de point d'entrée des données dans le système de streaming. Il simule le comportement de capteurs médicaux connectés qui enverraient en continu des mesures de tension vers un cluster Kafka, afin de permettre leur analyse en temps réel par les consumers et les modules d'anomalie.

Plus précisément, le Producer Python génère des messages conformes au standard FHIR (ressource Observation) et les publie sur un topic Kafka dédié aux données de pression artérielle. Cette première brique garantit que le pipeline traite des données structurées et现实的, proches de ce que produiraient de véritables dispositifs médicaux dans un contexte hospitalier.

## 2.3. Mise en place de Kafka côté producer

Côté infrastructure, le producer se connecte à un broker Kafka accessible sur notre localhost. Cette adresse est passée au producer via le paramètre "bootstrap\_servers" de la librairie "kafka-python", ce qui lui permet de publier des messages sur le cluster.

Les messages sont envoyés sur un topic spécifique, qui joue le rôle de canal logique pour toutes les observations de tension artérielle. Afin de garantir un échange de données standardisé, chaque message est réalisé au format JSON avant d'être transmis, grâce à un value\_serializer.

## 2.4. Génération des données patient

Au démarrage, la fonction initialise un ensemble fixe de patients (5 patients), chacun identifié par un UUID unique. À chaque itération, une nouvelle observation est générée pour l'un de ces patients, afin de reproduire le comportement de capteurs médicaux effectuant des mesures régulières. Le même identifiant est utilisé à la fois comme identifiant de la ressource Observation ("id"), et également comme référence du patient, ce qui assure la cohérence interne des données FHIR.

Un horodatage est également généré pour chaque observation à l'aide de Faker, avec une date aléatoire comprise entre la veille et le moment actuel, puis formatée au standard ISO 8601 via “isoformat()”. Cela permet de simuler un flux continu de mesures dans le temps, condition nécessaire pour les analyses temporelles et la détection des tendances dans les pressions artérielles.

## 2.5. Simulation de valeurs de pression artérielle réalistes

Afin de rendre le simulateur crédible sur le plan médical, le script génère des valeurs de pression artérielle systolique et diastolique dans des intervalles différenciés pour les cas normaux et les cas anormaux. Avec une probabilité d'environ 70 %, les valeurs générées sont normales, avec une pression systolique entre 110 et 130 mmHg et une pression diastolique entre 70 et 85 mmHg, ce qui correspond à des tensions typiques chez un adulte sans pathologie aiguë.

Dans environ 30 % des cas, la fonction déclenche volontairement une anomalie en produisant des valeurs plus élevées, la pression systolique est alors tirée entre 141 et 190 mmHg, et la pression diastolique entre 91 et 130 mmHg. Ces plages dépassent les seuils d'hypertension définis (systolique > 140 mmHg, diastolique > 90 mmHg) et permettent de générer des cas à risque qui seront ensuite détectés par le module d'analyse.

## 2.6. Structure JSON FHIR envoyée sur Kafka

Chaque message envoyé sur le topic blood\_pressure\_topic est une ressource FHIR de type Observation structurée au format JSON. La structure générale inclut notamment :

- **resourceType:** "Observation", qui indique le type de ressource FHIR.
- **id:** l'identifiant unique de l'observation, correspondant à l'UUID du patient.
- **status:** "final", signifiant que la mesure est validée et utilisable.
- **code:** un code LOINC ("85354-9") décrivant le "Blood pressure panel", afin de référencer standardisé l'acte de mesure.
- **subject:** la référence au patient sous la forme "Patient/<id>", ce qui relie l'observation à un patient donné.
- **effectiveDateTime:** la date et l'heure de la mesure au format ISO 8601.
- **component:** une liste de deux composants, l'un pour la pression systolique, l'autre pour la pression diastolique.

Chaque composant contient un code et un valueQuantity. Pour la pression systolique, le code "8480-6" avec l'intitulé "Systolic" est utilisé, et la valeur numérique est stockée dans valueQuantity.value avec l'unité "mmHg". Pour la pression diastolique, le code "8462-4" et le libellé "Diastolic" sont employés, toujours avec la même unité. Cette structure est conforme aux bonnes pratiques FHIR pour la représentation des panneaux de pression artérielle.

**Note :** Pour une consultation approfondie de la structure complète et des différents champs techniques utilisés, un exemple exhaustif du message JSON généré par le Producer est disponible sur notre dépôt GitHub.

## 2.7. Logique d'envoi continu sur Kafka

La fonction “main()” orchestre le comportement du producer comme un simulateur de capteurs en temps réel. Une boucle infinie génère périodiquement une nouvelle observation via “generate\_patient()”, puis l’envoie au broker Kafka grâce à la fonction “send-message()”. Ensuite chaque envoi, le script effectue une pause de cinq secondes, ce qui permet de contrôler le débit des flux et de se rapprocher du rythme d’un monitoring continu.

Lors de l’envoi, le producer attend l’accusé de réception (ACK) du broker ce qui permet de s’assurer que le message a bien été écrit dans une partition du topic. En cas d’erreur Kafka, une exception est capturée et un message d’échec est affiché, ce qui facilite le diagnostic. Enfin lorsqu’on interrompt le programme, le producer vide proprement son buffer avec “flush()” puis ferme la connexion avec close(), garantissant une fermeture maîtrisée de la session Kafka.

## 3. Analyse et Traitement des Données consumer

Une fois les messages produits et publiés sur Kafka, le Consumer prend le relais pour l’analyse et le tri des données.

### 3.1. Rôle du Consumer dans le pipeline de données

Le Consumer constitue la seconde brique essentielle de notre pipeline Kafka. Alors que le Producer se charge d’alimenter en continu le flux de données simulées, le Consumer a pour mission de récupérer, analyser et orienter ces données vers leur destination finale. Concrètement, il est capable de trier automatiquement les mesures de tension artérielle en fonction de leur caractère normal ou anormal. Ce module s’inscrit dans une logique de traitement temps réel : chaque message envoyé par le Producer sur le topic Kafka blood\_pressure\_topic est immédiatement consommé par le script Python consumer.py. Celui-ci décode le message JSON au format FHIR, en extrait les valeurs systolique et diastolique, puis applique les règles d’analyse médicale préétablies. Le Consumer assure ainsi le lien entre le flux de données brutes (Kafka) et la base d’analyse (Elasticsearch), tout en archivant localement les valeurs normales.

### 3.2. Configuration du Consumer et abonnement au topic Kafka

Le Consumer a été implémenté en Python à l’aide de la librairie kafka-python. Il se connecte au même broker que le Producer, en utilisant l’adresse localhost:9092 définie par le paramètre bootstrap\_servers. Le groupe de consommateurs a été défini sous le nom bp-consumer-group, ce qui permet à Kafka de gérer la position des messages déjà lus, garantissant qu’aucune observation ne soit traitée deux fois. Lors de son initialisation, le Consumer s’abonne explicitement au topic blood\_pressure\_topic. Dès qu’une nouvelle observation est publiée par le “Producer”, le “Consumer” la reçoit, la décode, puis la passe dans une fonction d’analyse. Le flux est donc continu et synchrone, ce qui reproduit le fonctionnement d’un système médical en surveillance permanente.

### 3.3. Extraction et lecture des données FHIR

Chaque message lu par le Consumer est une ressource FHIR de type *Observation*, contenant la structure complète d'une mesure de pression artérielle.

Le Consumer commence par extraire les deux composants essentiels :

- la **pression systolique**, identifiée par le code LOINC "8480-6",
- et la **pression diastolique**, identifiée par le code "8462-4".

Ces deux valeurs numériques sont ensuite stockées dans des variables Python et converties en nombres flottants pour permettre leur comparaison. Si l'un des deux champs est manquant, l'observation est ignorée afin d'éviter toute erreur de traitement.

Cette phase d'extraction garantit que les règles de détection s'appliquent uniquement sur des données valides, conformes à la structure FHIR initialement définie par le Producer.

### 3.4. Détection automatique des anomalies

La détection d'anomalies repose sur une fonction centrale nommée `detect_anomaly()`. Cette fonction applique un ensemble de règles simples mais efficaces inspirées des recommandations médicales.

Une observation est considérée comme anormale si au moins l'une des conditions suivantes est vérifiée :

- la pression systolique est supérieure à 140 mmHg ou inférieure à 90 mmHg ;
- la pression diastolique est supérieure à 90 mmHg ou inférieure à 60 mmHg.

Lorsque les deux valeurs se situent dans les intervalles de référence ( $90 \leq \text{systolique} \leq 140$  et  $60 \leq \text{diastolique} \leq 90$ ), la tension est jugée normale. La fonction retourne alors un indicateur booléen (True ou False) et une étiquette décrivant le type d'anomalie détectée (par exemple : *hypertension systolique* ou *hypotension diastolique*).

Cette étape permet au Consumer d'assurer un pré-tri automatique, simulant le rôle d'un module d'alerte médicale dans un système de santé connecté.

### 3.5. Traitement et orientation des données

Une fois la classification réalisée, le Consumer applique une logique de tri binaire. Si la mesure est normale, le message est archivé localement dans un fichier unique nommé **normal\_blood\_pressure.jsonl**. Ce format, dit *JSON Lines*, permet d'enregistrer une observation par ligne et de constituer progressivement un historique complet des valeurs normales sans générer de multiples fichiers. Cette solution est particulièrement adaptée à un traitement en continu. Cependant, lorsque la mesure est jugée anormale, le Consumer construit un document simplifié ne conservant que les informations pertinentes : l'identifiant du patient, les valeurs systolique et diastolique, le type d'anomalie, et la date de la mesure. Ce document est ensuite

envoyé à **Elasticsearch** via une requête HTTP. L'objectif est de rendre immédiatement ces données disponibles pour la visualisation et l'analyse sur **Kibana**.

### **3.6. Indexation dans Elasticsearch**

L'indexation correspond à l'étape où les données anormales sont enregistrées dans une base de recherche optimisée. Dans notre projet, les observations anormales sont indexées dans l'index nommé **bp\_anomalies** d'Elasticsearch. Chaque document contient des champs précis : le patient\_id, les deux valeurs de pression, le type d'anomalie et un timestamp. Cette structure permet de faciliter les analyses temporelles et les visualisations de tendance dans Kibana. Une fois les données stockées, il est possible d'interroger Elasticsearch pour retrouver rapidement un patient ou un ensemble d'observations présentant des anomalies similaires. L'indexation assure donc une recherche rapide, une agrégation efficace et une visualisation immédiate dans Kibana, tout en garantissant la scalabilité du système.

## **4. Analyse BI & Visualisation (Kibana Expert)**

Cette section détaille la couche de présentation de notre projet, que nous avons conçue comme un véritable "Centre de Surveillance des Constantes Vitales" à destination des services d'urgence ou de réanimation. Nous sommes partis de l'hypothèse de travail selon laquelle nous fournissons ce tableau de bord à un hôpital afin d'améliorer drastiquement le suivi de leurs patients en temps réel. L'objectif est de transformer le flux brut de l'index des anomalies en indicateurs décisionnels clairs, permettant de passer de la donnée à l'action médicale.

### **4.1. Vue d'ensemble et allocation des ressources**

Dès le premier coup d'œil, le personnel soignant accède à des indicateurs clés via des cartes de métriques mettant en évidence les cas anormaux (ici, 5 anomalies actuellement détectées). Cette volumétrie est complétée par un graphique de répartition (camembert) qui segmente les alertes par typologie. Dans notre capture actuelle, nous observons une répartition parfaitement équilibrée de 50% d'hypertensions systoliques et 50% d'hypertensions diastoliques. D'un point de vue de la gestion hospitalière, cette visualisation est stratégique : elle permet à l'établissement d'allouer ses ressources de manière beaucoup plus ciblée en se concentrant sur les pathologies les plus fréquentes. À grande échelle, cela permet d'ajuster la gestion des stocks de la pharmacie, par exemple en anticipant les commandes de médicaments antihypertenseurs ou de traitements préventifs contre les crises.

### **4.2. Indicateurs de gravité et seuils cliniques**

Pour identifier instantanément l'urgence vitale, nous avons modélisé deux jauge de suivi (Systolique et Diastolique). Contrairement à de simples compteurs, ces jauge intègrent une échelle colorimétrique bidirectionnelle stricte, basée sur les standards médicaux : le vert foncé représente une anomalie marquant une chute des constantes (hypotension), le vert indique une tension normale (tout est sous contrôle), puis l'échelle monte en intensité avec le jaune (pré-

alerte), l'orange (hypertension), et enfin le rouge écarlate pour les urgences absolues.

Actuellement, les jauge affichent des valeurs maximales critiques atteignant 188 mmHg en systolique et 129 mmHg en diastolique, toutes deux situées dans la zone rouge. À leurs côtés, les valeurs moyennes (171.6 et 109.6 mmHg) offrent une vue d'ensemble sur l'état de santé du groupe de patients filtrés. Ce design cognitif réduit drastiquement le temps d'analyse visuelle et alerte le médecin avant même qu'il ne lise les chiffres exacts.

#### **4.3. Analyse des tendances temporelles**

Au-delà de l'alerte instantanée, le tableau de bord intègre une dimension analytique avec une courbe temporelle (Heure d'admission par pas de 10 minutes). Cette analyse des tendances répond au besoin d'identifier les pics de tension par période. Tracer ces données permet de tester et de vérifier des hypothèses cliniques, comme l'influence d'autres facteurs externes sur la pression artérielle (le moment de la journée, le stress d'une admission, ou des événements spécifiques). Si la courbe monte brusquement de manière synchronisée pour plusieurs patients, le personnel soignant peut repérer des phénomènes de "vagues" et anticiper la charge de travail au sein du service.

#### **4.4. Registre de triage et suivi individualisé**

Enfin, la prise de décision opérationnelle est centralisée autour du registre des urgences. Ce tableau liste les 5 patients filtrés par ordre de gravité décroissante. En plaçant le patient présentant une systolique de 188 mmHg en tête de liste, ce tableau agit comme un outil de triage automatique. De plus, ce registre permet un suivi longitudinal des patients. En sélectionnant un ID patient spécifique via les filtres de Kibana, le médecin peut analyser si la pression artérielle d'un individu diminue ou augmente sous l'effet de certains médicaments qui lui ont été administrés au cours de son admission. L'information est ainsi hiérarchisée pour traiter les individus les plus en danger de manière quasi instantanée.

### **5. Synthèse : Les Forces de l'Architecture Big Data**

Notre infrastructure se démarque par sa capacité à allier vitesse de traitement et optimisation des ressources. La principale force de ce projet réside dans sa réactivité en temps réel grâce à l'architecture "Event-Driven" portée par Kafka. Contrairement à une analyse de données classique et statique, notre pipeline traite les messages au fur et à mesure qu'ils sont générés. Ainsi, si un patient subit une crise à un instant précis, l'alerte correspondante traverse le broker et apparaît sur Kibana en un temps record, illustrant toute la puissance du "Streaming" pour des cas d'usage critiques en santé.

Cette vélocité est soutenue par un filtrage intelligent mis en place dès la phase de consommation des données. Le script Python du Consumer agit comme un premier filtre médical essentiel : il écarte les données saines et ne pousse vers la base de données Elasticsearch que les véritables anomalies. Cette approche architecturale, proche de l'Edge Computing, garantit que notre espace

de stockage n'est pas saturé inutilement et que les recherches sur l'index restent ultra-performantes.

## 6. Conclusion

En conclusion, notre projet illustre parfaitement l'intégration efficace de technologies Big Data modernes telles qu'Apache Kafka, la suite Elastic (Elasticsearch & Kibana) et le standard d'interopérabilité santé FHIR, au service du monde médical.

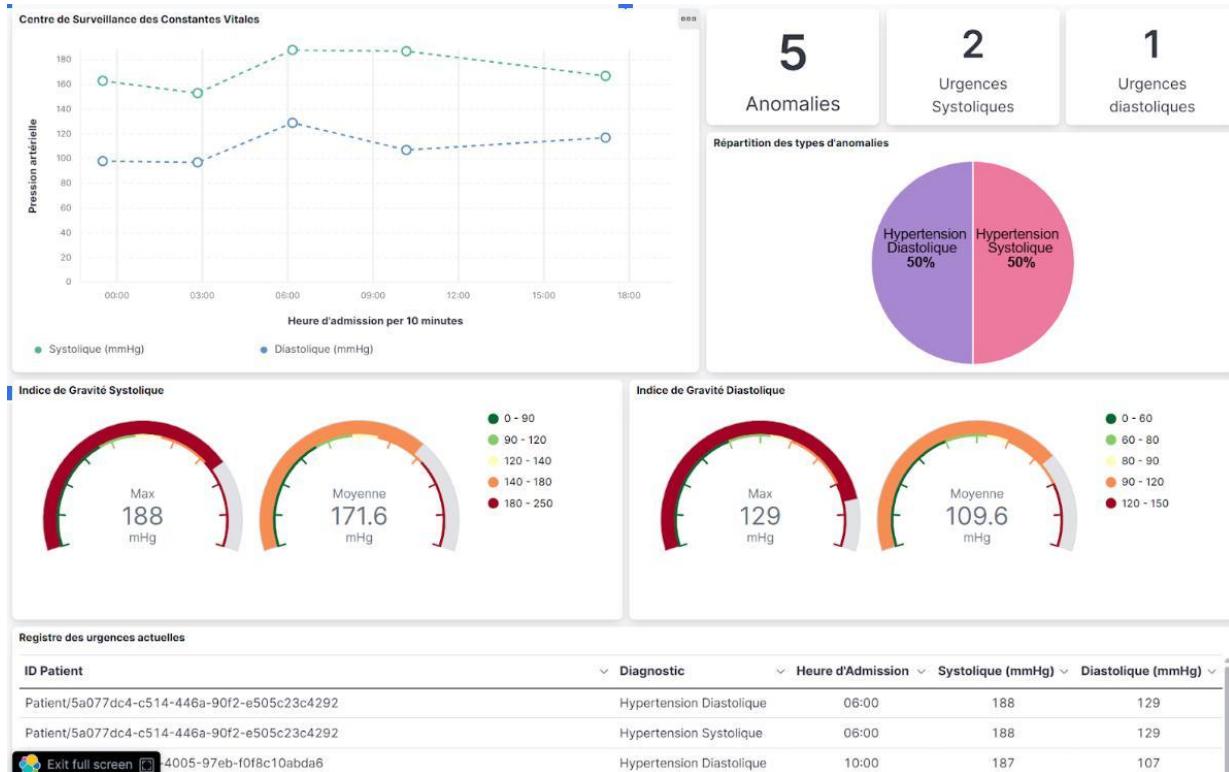
La mise en place de cette architecture a permis de valider une chaîne de traitement complète et temps réel. D'une part, Kafka a prouvé sa capacité à gérer de manière robuste et continue le flux des messages FHIR générés par nos capteurs simulés. D'autre part, le filtrage intelligent assuré par notre script Python a permis d'écartier les données saines, ne transmettant à Elasticsearch que les anomalies. Ce choix architectural garantit un stockage optimisé et des requêtes ultra-performantes.

L'aboutissement visuel de ce pipeline, notre tableau de bord Kibana, démontre qu'il est possible d'aller au-delà du simple stockage de données. En regroupant des visualisations claires, des indicateurs de gravité et des analyses temporelles, nous fournissons aux professionnels de santé un outil clé en main pour suivre les patients, analyser les anomalies artérielles et identifier des tendances.

Bien que la solution actuelle soit pleinement opérationnelle et adaptée aux besoins de triage d'un hôpital, elle ouvre la voie à de nombreuses perspectives d'amélioration. À l'avenir, cette infrastructure pourrait être enrichie en y intégrant des algorithmes de Machine Learning capables d'effectuer des analyses prédictives pour anticiper les crises avant qu'elles ne surviennent, ou encore en couplant le pipeline à un système de notifications (SMS/Email) pour déclencher des alertes médicales immédiates.

Ce projet prouve que la maîtrise des flux de données en continu est aujourd'hui une compétence indispensable pour concevoir les systèmes de santé de demain.

## Annexe



*Dashboard : aperçu des données globales*