Pipeline de Streaming pour la Surveillance de Capteurs Environnementaux

1. Présentation Générale du Projet

Ce document décrit la **conception** et la **mise en œuvre** d'un pipeline de streaming de données en **temps réel** dédié à la surveillance de capteurs environnementaux.

Les données traitées incluent notamment :

- **Température**
- **Humidité**
- **CO_2**
- **Pression atmosphérique**
- **État de la batterie**

Le système repose sur des technologies modernes de traitement de flux : **Kafka**, **Spark Structured Streaming**, **PostgreSQL** et **Streamlit**.

2. Objectifs et Enjeux du Système

Le pipeline répond à plusieurs besoins critiques liés à la gestion de données en continu :

- Surveillance instantanée grâce au traitement des données en temps réel dès leur arrivée.
- Détection rapide des anomalies telles que les valeurs extrêmes, pannes et warnings.
- Analyse fiable des tendances à moyen et long terme.
- Agrégation temporelle des données sur des fenêtres définies.
- Stockage persistant et fiable des résultats.
- Visualisation interactive et exploitable pour la prise de décision.

3. Composants Techniques du Pipeline

Composant	Rôle	Justification du choix
Producteur Kafka	Simule la génération continue des données de capteurs	Permet de créer un flux réaliste, continu et configurable, reproduisant un environnement proche de la production.
Kafka	Bus de messages distribué, tolérant aux pannes	Garantit haute disponibilité, persistance des messages et capacité à absorber les pics de charge, idéal pour le streaming temps réel.

Spark Structured Streaming	Traitement distribué des flux en micro-batches	Offre un modèle de traitement temps réel scalable, gère les fenêtres temporelles et les agrégations, tout en restant simple à intégrer avec l'écosystème Spark.
PostgreSQL	Stockage des données brutes et agrégées	Base relationnelle fiable et performante, permettant des requêtes analytiques avancées et une séparation claire entre données temps réel et historiques.
Dashboard Streamlit	Visualisation interactive des données	Permet de créer rapidement des interfaces interactives avec Python, facilitant l'exploration des données en temps réel et agrégées.
Docker et docker-compose	Conteneurisation des applications et services	Orchestrer l'ensemble des services afin de garantir la portabilité et la reproductibilité de l'environnement.

4. Architecture Globale du Système

L'architecture du pipeline est **séquentielle, modulaire et découplée**, organisée en quatre grandes étapes :

1. Production des données de capteurs
2. Ingestion et traitement via Kafka et Spark
3. Persistance dans PostgreSQL
4. Visualisation via Streamlit
5. Portabilité et la reproductibilité du projet sur différents environnements via Docker

Cette organisation garantit une **scalabilité élevée**, une **tolérance aux pannes** et une **bonne maintenabilité**.

5. Traitement des Données en Streaming

5.1 Préparation des Données et Gestion du Temps

Avant tout traitement métier, les données subissent les étapes suivantes :

- Consommation Kafka sous forme de micro-batches
- Parsing JSON avec un schéma explicite
- Gestion du temps événementiel (Event Time)
- Application d'un watermark pour gérer les données en retard et limiter l'état en mémoire

Cette approche garantit la **cohérence temporelle** des calculs.

5.2 Stratégie de Traitement à Double Flux

Le flux principal est scindé en **deux traitements parallèles**, chacun répondant à un besoin spécifique.

5.2.1 Flux de Données Brutes (Raw Stream)

Objectif
Fournir une vision **instantanée** de l'état des capteurs afin de permettre une réaction immédiate face aux événements détectés.

Traitement
Les données subissent peu de transformations et sont écrites directement dans la table *sensor_data*, garantissant une faible latence et une disponibilité rapide de l'information.

Cas d'usage

- Surveillance en temps réel
- Génération d'alertes immédiates
- Analyse de l'état courant du système

5.2.2 Flux de Données Agrégées (Windowed Stream)

Objectif
Réduire le bruit des données et analyser les tendances de fond sur des périodes temporelles définies.

Traitement
Application de fenêtres temporelles glissantes avec des calculs d'agrégation tels que :

- Calcul des moyennes (température, humidité, CO_2, niveau de batterie)
- Comptage des alertes et des pannes

Les résultats agrégés sont stockés dans la table *sensor_data_agg*.

Cas d'usage

- Analyse de tendances
- Étude des corrélations entre variables
- Suivi global et synthétique de l'état du système

6. Stockage et Visualisation

6.1 Persistance des Données avec PostgreSQL

La séparation des données permet d'améliorer la lisibilité du système ainsi que les performances globales. La table **sensor_data** contient les données brutes reflétant l'état instantané des capteurs, tandis que la table **sensor_data_agg** stocke les données agrégées destinées à l'analyse historique et au suivi des tendances.

6.2 Tableau de Bord Interactif avec Streamlit

Le dashboard est structuré en **deux vues complémentaires**.

6.2.1 Vue Temps Réel

Question clé : *Que se passe-t-il maintenant ?*

Contenu

- Indicateurs clés de performance (KPI) instantanés
- Alertes affichées en direct
- Séries temporelles à granularité fine
- Visualisation spatiale de la répartition des capteurs

Philosophie

Cette vue privilégie la **réactivité** et la rapidité d'interprétation, même si les données peuvent être bruitées. Elle est principalement destinée à un usage opérationnel.

6.2.2 Vue Agrégée

Question clé : *Quelle est la tendance ?*

Contenu

- Moyennes calculées par fenêtre temporelle
- Évolution des variables clés
- Répartition et fréquence des alertes
- Analyses comparatives entre capteurs ou périodes

Philosophie

Cette vue met l'accent sur des **visualisations stables, lisibles et interprétables**, facilitant l'analyse décisionnelle et stratégique.

7. Aspects Opérationnels et Choix d'Architecture

7.1 Supervision et Observabilité

Le système est supervisé à l'aide des interfaces natives de **Apache Spark**, notamment la **Spark Master UI** et la **Spark Application UI**.
Ces outils offrent une visibilité détaillée sur l'exécution des traitements et permettent de suivre en temps réel :

- La latence de bout en bout du pipeline
- La durée d'exécution des micro-batches
- L'utilisation des ressources (CPU et mémoire)
- L'état et la disponibilité des exécuteurs

Cette supervision facilite l'identification des goulots d'étranglement, l'analyse des performances et la détection précoce d'éventuelles anomalies dans le traitement des flux.

7.2 Décisions Techniques Clés

Plusieurs choix techniques ont été effectués afin d'assurer la performance, la lisibilité et la fiabilité du système :

- **Micro-batching** : permet d'optimiser le débit et la stabilité du traitement tout en conservant une latence maîtrisée.
- **Séparation des données brutes et agrégées** : distinction claire entre les usages temps réel et analytiques, améliorant la lisibilité et les performances.
- **Fenêtres temporelles** : utilisées pour lisser le bruit des données et stabiliser les métriques dans le temps.
- **Dashboard scindé** : séparation entre vue temps réel et vue agrégée afin d'offrir une meilleure expérience utilisateur et une interprétation adaptée aux différents besoins.

8. Conclusion Générale

Ce projet démontre une implémentation complète et cohérente d'un pipeline de streaming temps réel, de la génération des données à leur visualisation. Il met en pratique les concepts fondamentaux du streaming moderne, notamment le temps événementiel, le micro-batching, le windowing, la tolérance aux pannes ainsi que la séparation des usages temps réel et analytiques.