Analyse de Données Capteurs en Temps Réel via Fog Computing et Federated Learning

Cheikh Bay Oudaa [C16706]
Department of Computer Science, Master 2 AI Machine Learning & Data Science
University of Nouakchott

10 février 2026

1 Introduction

Dans l'ère de l'Internet des Objets (IoT), les appareils et capteurs génèrent d'énormes volumes de données en temps réel. La collecte et le traitement centralisés de ces données posent des défis majeurs, notamment :

- La **bande passante** nécessaire pour transférer toutes les données vers un serveur central.
- La **confidentialité et sécurité** des données sensibles (transactions financières, données personnelles).

Pour surmonter ces limites, ce projet utilise une approche **décentralisée** combinant :

- **Fog Computing** : traitement local des données à la périphérie du réseau.
- **Federated Learning** : entraînement collaboratif d'un modèle de machine learning sans centraliser les données brutes.

L'objectif est de simuler une infrastructure de détection d'anomalies et de fraudes mobiles, capable d'entraîner un modèle distribué et d'afficher les résultats via un **dashboard temps réel**.

2 Architecture du Système

Le système est basé sur une architecture à trois niveaux :

1. **Couche Device (Producteurs)** : simulateurs de capteurs générant des transactions ou données financières. Les données sont envoyées vers Kafka via des topics dédiés.
2. **Couche Fog (Spark Streaming)** : chaque nœud Fog reçoit les données en temps réel, effectue un prétraitement et entraîne localement un modèle de détection.
3. **Couche Cloud (Agrégateur)** : collecte les poids de tous les nœuds Fog, applique l'algorithme FedAvg pour produire un modèle global, et redistribue ce modèle aux nœuds Fog.

Schéma de l'architecture :

Device Nodes → Kafka Topics → Fog Nodes (Spark) → Kafka Topics → Cloud Aggregator → Redistribuer Modèle

3 Technologies Utilisées

— **Apache Kafka** : transport et gestion des flux de données en temps réel.
— **Apache Spark (PySpark)** : traitement analytique et entraînement des modèles.
— **Python** : scripts producteurs et consommateurs.
— **Docker Compose** : orchestration des conteneurs Kafka, Zookeeper et Spark.

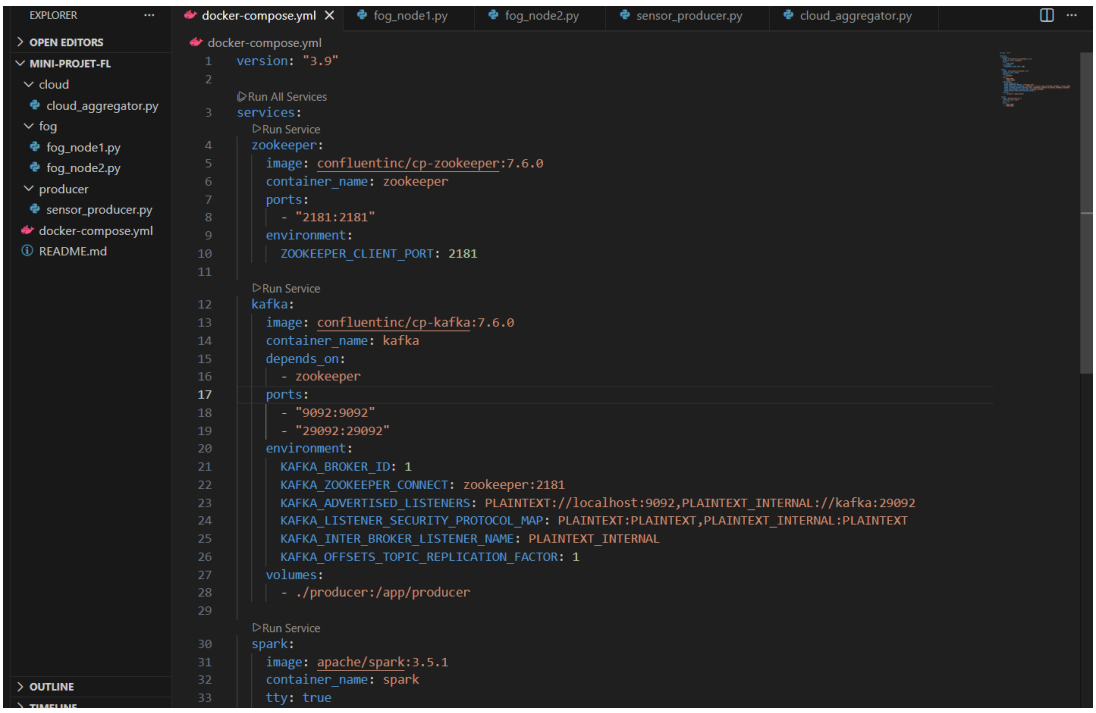

FIGURE 1 – Structure du projet et configuration des services dans l'IDE.

4 Mise en Œuvre

4.1 Phase 1 : Configuration Kafka

Démarrage des conteneurs :

```
docker compose up -d
```

Création des topics Kafka :

```
docker exec -it kafka \
kafka-topics --create --topic sensor-data-node-1 --bootstrap-server
    localhost:9092 --partitions 1 --replication-factor 1

docker exec -it kafka \
kafka-topics --create --topic sensor-data-node-2 --bootstrap-server
    localhost:9092 --partitions 1 --replication-factor 1

docker exec -it kafka \
kafka-topics --create --topic model-weights --bootstrap-server localhost
    :9092 --partitions 1 --replication-factor 1

docker exec -it kafka \
```

```
kafka-topics --create --topic global-model --bootstrap-server localhost
    :9092 --partitions 1 --replication-factor 1
```

4.2 Phase 2 : Producteurs de données (Device Nodes)

Chaque producteur simule des transactions financières et publie dans Kafka :

```
{
  "timestamp":"2026-02-10T17:00:00",
  "value": 123.45,
  "device_id": "node1"
}
```

4.3 Phase 3 : Nœuds Fog (Spark Streaming)

Lecture des topics Kafka et entraînement local :

```
spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "sensor-data-node-1") \
    .load()
```

4.4 Phase 4 : Agrégation Cloud

Application de l'algorithme FedAvg :

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_t^k$$

4.5 Phase 5 : Visualisation (Dashboard)

- Affichage des flux de capteurs en temps réel. - Graphiques des valeurs des capteurs et du modèle. - Tableau des anomalies détectées :

```
st.dataframe(df.tail(10))
st.line_chart(df["value"].tail(50))
```

5 Résultats et Observations

— Les nœuds Fog calculent localement les poids du modèle sans exposer les données brutes.
— Le modèle global converge progressivement grâce à FedAvg.
— Les anomalies sont détectées en temps réel.
— Le dashboard Streamlit permet de suivre les dernières transactions et les anomalies.

6 Conclusion

Ce projet démontre l'efficacité de l'approche décentralisée :
— Traitement local des données via Fog Computing.
— Confidentialité préservée grâce au Federated Learning.
— Gestion efficace des flux en temps réel avec Kafka et Spark.
Perspectives d'amélioration :
— Ajouter des nœuds Fog pour tester la scalabilité.
— Intégrer un modèle de détection de fraude plus complexe.
— Stocker les transactions historiques pour analyses futures.