Rapport de Mini-Projet

Architecture Kafka et Spark
Edge – Fog – Cloud et Federated Learning

Auteur : Daouda Ba

Année : 2026

Table des matières

1 - Introduction

Avec l'explosion des données générées par les objets connectés (IoT), les architectures distribuées sont devenues indispensables. Ce projet vise à mettre en œuvre une architecture **Edge – Fog – Cloud** reposant sur **Apache Kafka** pour la communication et **Apache Spark Structured Streaming** pour le traitement temps réel.

L'objectif est de simuler un scénario inspiré du **Federated Learning**, où les calculs sont réalisés localement (Fog Nodes) avant d'être agrégés dans le Cloud.

2 - Architecture Générale

2.1 Vue globale

L'architecture est composée de trois niveaux :

— **Edge** : capteurs simulés produisant des données

— **Fog** : nœuds intermédiaires traitant localement les données

— **Cloud** : agrégation globale des résultats

2.2 Schéma logique

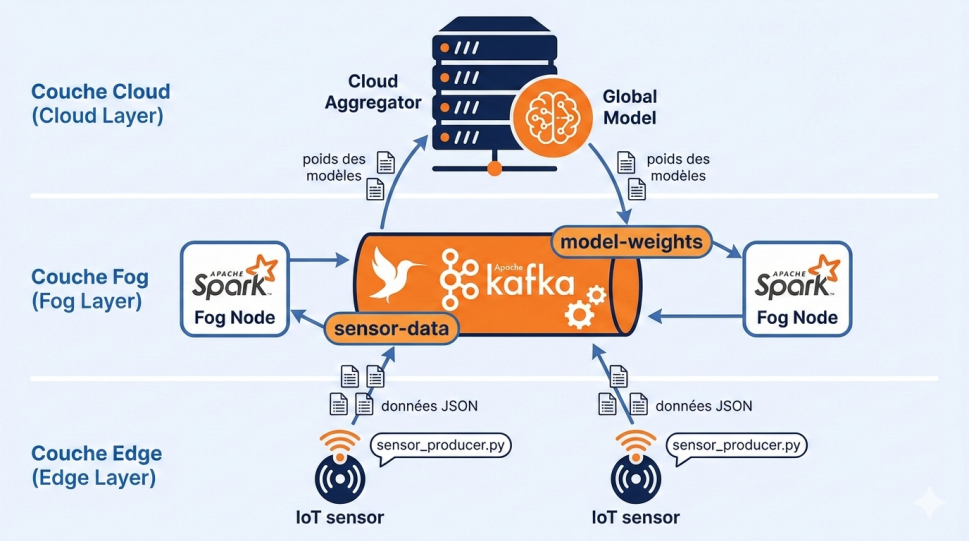

FIGURE 2.1 – Architecture globale du projet

3 - Environnement Technique

3.1 Technologies utilisées

— Apache Kafka
— Apache Spark 3.5.1
— Docker et Docker Compose
— Python (PySpark)
— Zookeeper

3.2 Structure du projet

```
kafka-spark-federated/
|-- docker-compose.yml
|-- producer/
|   |-- sensor_producer.py
|-- fog/
|   |-- fog_node_1.py
|   |-- fog_node_2.py
|-- cloud/
|   |-- aggregator.py
|-- README.md
```

4 - Déploiement avec Docker

4.1 docker-compose.yml

```yaml
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.6.0
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka:7.6.0
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:29092,PLAINTEXT_HOST
    ://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST
    ://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
    PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  spark:
    image: apache/spark:3.5.1
    container_name: spark
    depends_on:
```

```
32        - kafka
33    ports:
34        - "8080:8080"
35        - "4040:4040"
36    volumes:
37        - ./spark:/app
```

Listing 4.1 – Fichier docker-compose.yml

4.2 Commandes principales

```
1 docker compose up -d
2 docker ps
```

5 - Phase 1 : Kafka (Edge)

5.1 Création des topics

```
docker exec -it kafka kafka-topics
--bootstrap-server kafka:29092
--create --topic sensor-data
--partitions 1 --replication-factor 1

docker exec -it kafka kafka-topics
--bootstrap-server kafka:29092
--create --topic model-weights
--partitions 1 --replication-factor 1
```

5.2 Producteur de données

```
import json
import time
from kafka import KafkaProducer
import random

producer = KafkaProducer(
    bootstrap_servers='kafka:29092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

while True:
    data = {
        "sensor_id": "s1",
        "temperature": round(random.uniform(20, 35), 2),
        "vibration": round(random.uniform(0.01, 0.05), 3)
    }
    producer.send("sensor-data", data)
    time.sleep(2)
```

Listing 5.1 – Producteur de capteurs Kafka

6 - Phase 2 : Fog Nodes

6.1 Objectif

Les Fog Nodes consomment les données capteurs, calculent des moyennes locales et produisent un poids représentant un modèle local.

6.2 Code Fog Node

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, avg
from pyspark.sql.types import StructType, StringType, DoubleType,
    IntegerType
import json

spark = SparkSession.builder \
    .appName("FogNode1") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Sch ma des donn es capteurs
schema = StructType() \
    .add("sensor_id", StringType()) \
    .add("temperature", DoubleType()) \
    .add("vibration", DoubleType()) \
    .add("label", IntegerType())

# Lecture Kafka
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "kafka:29092") \
    .option("subscribe", "sensor-data-node-1") \
    .load()

parsed = df.select(
    from_json(col("value").cast("string"), schema).alias("data")
```

```
).select("data.*")

# Calcul du mod le local (moyenne)
model = parsed.groupBy().agg(
    avg("temperature").alias("avg_temp"),
    avg("vibration").alias("avg_vib")
)

def send_weights(batch_df, batch_id):
    if batch_df.count() == 0:
        return

    row = batch_df.collect()[0]
    #weight = (row.avg_temp + 100 * row.avg_vib) / 2
    weight = ((row.avg_temp or 0) +100 * (row.avg_vib or 0)) / 2


    output = {
        "node_id": "fog-1",
        "weight": round(weight, 4),
        "samples": batch_df.count()
    }

    spark.createDataFrame(
        [(json.dumps(output),)],
        ["value"]
    ).selectExpr("CAST(value AS STRING)") \
     .write \
     .format("kafka") \
     .option("kafka.bootstrap.servers", "kafka:29092") \
     .option("topic", "model-weights") \
     .save()

    print("Fog-1 -> poids envoy :", output)

query = model.writeStream \
    .foreachBatch(send_weights) \
    .outputMode("complete") \
    .start()

query.awaitTermination()
```

Listing 6.1 – Fog node 1

6.3 Commande d'exécution

```
docker exec -it spark /opt/spark/bin/spark-submit
--conf spark.jars.ivy=/tmp/ivy
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1
/app/fog/fog_node_1.py
```

6.4 Erreurs rencontrées et corrections

6.4.1 Kafka non trouvé

Erreur :

```
Failed to find data source: kafka
```

Solution : Ajout du package Kafka lors du spark-submit.

6.4.2 Valeurs nulles (NoneType)

Erreur :

```
TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
```

Solution :

```
avg_temp = row.avg_temp or 0
avg_vib = row.avg_vib or 0
weight = (avg_temp + 100 * avg_vib) / 2
```

7 - Phase 3 : Cloud Aggregator

7.1 Objectif

Le Cloud Aggregator reçoit les poids des Fog Nodes et calcule un modèle global.

7.2 Code de l'agrégateur

Code de l'agrégateur (voir fichier cloud/aggregator.py)

7.3 Commande d'exécution

```
docker exec -it spark /opt/spark/bin/spark-submit
--conf spark.jars.ivy=/tmp/ivy
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1
/app/cloud/aggregator.py
```

8 - Résultats et Analyse

Le système fonctionne en temps réel, avec une agrégation correcte des modèles locaux. Le poids global évolue dynamiquement selon les données reçues.

8.1 Captures d'écran

8.1.1 Producteur de données

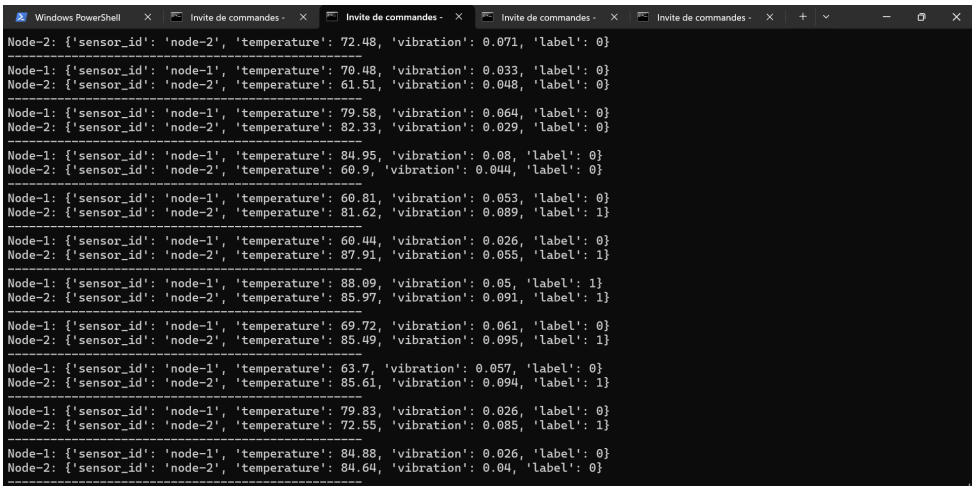

FIGURE 8.1 – Production des données capteurs vers Kafka

8.1.2 Fog Node en exécution

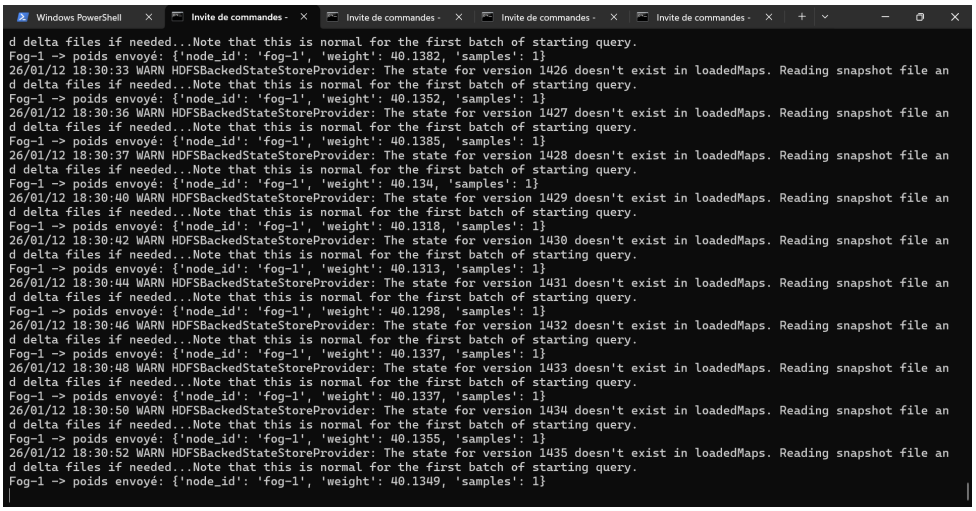

FIGURE 8.2 – Traitement local par le Fog Node

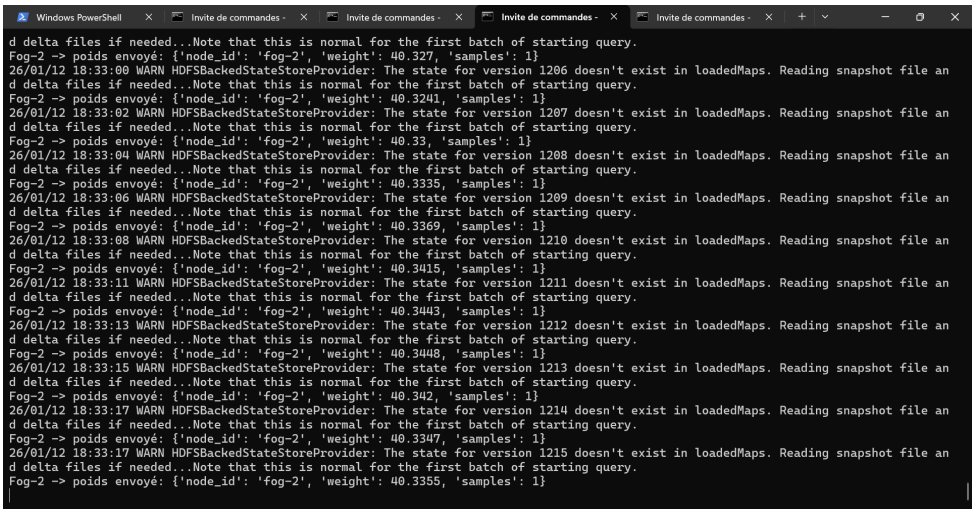

FIGURE 8.3 – Traitement local par le Fog Node

8.1.3 Agrégation Cloud

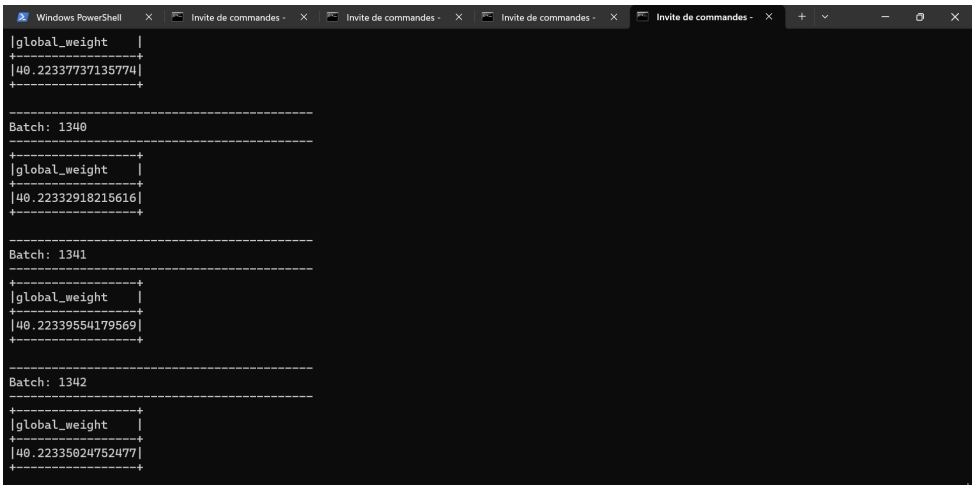

FIGURE 8.4 – Agrégation globale des poids dans le Cloud

8.1.4 Topics Kafka

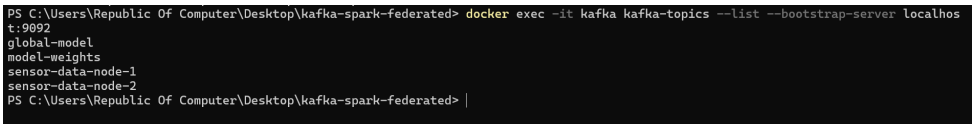

FIGURE 8.5 – Liste des topics Kafka créés

9 - Conclusion

Ce projet démontre l'efficacité de Kafka et Spark pour la mise en œuvre d'architectures distribuées modernes. Il constitue une base solide pour des travaux avancés en Federated Learning, Edge Computing et Big Data temps réel.