

Rapport de Mini-Projet

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

Auteur : Daouda Ba

Année : 2026

Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | Architecture Générale | 4 |
| 2.1 | Vue globale | 4 |
| 2.2 | Schéma logique | 4 |
| 3 | Environnement Technique | 5 |
| 3.1 | Technologies utilisées | 5 |
| 3.2 | Structure du projet | 5 |
| 4 | Déploiement avec Docker | 6 |
| 4.1 | docker-compose.yml | 6 |
| 4.2 | Commandes principales | 7 |
| 5 | Phase 1 : Kafka (Edge) | 8 |
| 5.1 | Création des topics | 8 |
| 5.2 | Producteur de données | 8 |
| 6 | Phase 2 : Fog Nodes | 9 |
| 6.1 | Objectif | 9 |
| 6.2 | Code Fog Node | 9 |
| 6.3 | Commande d'exécution | 10 |
| 6.4 | Erreurs rencontrées et corrections | 11 |
| 6.4.1 | Kafka non trouvé | 11 |
| 6.4.2 | Valeurs nulles (NoneType) | 11 |
| 7 | Phase 3 : Cloud Aggregator | 12 |
| 7.1 | Objectif | 12 |
| 7.2 | Code de l'agrégateur | 12 |
| 7.3 | Commande d'exécution | 12 |

| | | |
|----------|---------------------------------|-----------|
| 8 | Résultats et Analyse | 13 |
| 8.1 | Captures d'écran | 13 |
| 8.1.1 | Producteur de données | 13 |
| 8.1.2 | Fog Node en exécution | 14 |
| 8.1.3 | Agrégation Cloud | 15 |
| 8.1.4 | Topics Kafka | 15 |
| 9 | Conclusion | 16 |

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

Capteurs --> Kafka(sensor-data)

|

+-----+-----+

|

|

Fog Node 1

Fog Node 2

|

|

+-----+-----+

|

Kafka(model-weights)

|

Cloud Aggregator

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

```
1 services:
2   zookeeper:
3     image: confluentinc/cp-zookeeper:7.6.0
4     container_name: zookeeper
5     environment:
6       ZOOKEEPER_CLIENT_PORT: 2181
7       ZOOKEEPER_TICK_TIME: 2000
8     ports:
9       - "2181:2181"
10
11   kafka:
12     image: confluentinc/cp-kafka:7.6.0
13     container_name: kafka
14     depends_on:
15       - zookeeper
16     ports:
17       - "9092:9092"
18       - "29092:29092"
19     environment:
20       KAFKA_BROKER_ID: 1
21       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
22       KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:29092,PLAINTEXT_HOST
23       ://0.0.0.0:9092
24       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST
25       ://localhost:9092
26       KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
27       PLAINTEXT_HOST:PLAINTEXT
28       KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
29       KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
30
31   spark:
32     image: apache/spark:3.5.1
33     container_name: spark
34     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

```
1 docker exec -it kafka kafka-topics
2 --bootstrap-server kafka:29092
3 --create --topic sensor-data
4 --partitions 1 --replication-factor 1
5
6 docker exec -it kafka kafka-topics
7 --bootstrap-server kafka:29092
8 --create --topic model-weights
9 --partitions 1 --replication-factor 1
```

5.2 Producteur de données

```
1 import json
2 import time
3 from kafka import KafkaProducer
4 import random
5
6 producer = KafkaProducer(
7     bootstrap_servers='kafka:29092',
8     value_serializer=lambda v: json.dumps(v).encode('utf-8')
9 )
10
11 while True:
12     data = {
13         "sensor_id": "s1",
14         "temperature": round(random.uniform(20, 35), 2),
15         "vibration": round(random.uniform(0.01, 0.05), 3)
16     }
17     producer.send("sensor-data", data)
18     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

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import from_json, col, avg
3 from pyspark.sql.types import StructType, StringType, DoubleType,
   IntegerType
4 import json
5
6 spark = SparkSession.builder \
7     .appName("FogNode1") \
8     .getOrCreate()
9
10 spark.sparkContext.setLogLevel("WARN")
11
12 # Sch ma des donn es capteurs
13 schema = StructType() \
14     .add("sensor_id", StringType()) \
15     .add("temperature", DoubleType()) \
16     .add("vibration", DoubleType()) \
17     .add("label", IntegerType())
18
19 # Lecture Kafka
20 df = spark.readStream \
21     .format("kafka") \
22     .option("kafka.bootstrap.servers", "kafka:29092") \
23     .option("subscribe", "sensor-data-node-1") \
24     .load()
25
26 parsed = df.select(
27     from_json(col("value").cast("string"), schema).alias("data")
```

```

28 ).select("data.*")
29
30 # Calcul du mod le local (moyenne)
31 model = parsed.groupBy().agg(
32     avg("temperature").alias("avg_temp"),
33     avg("vibration").alias("avg_vib")
34 )
35
36 def send_weights(batch_df, batch_id):
37     if batch_df.count() == 0:
38         return
39
40     row = batch_df.collect()[0]
41     #weight = (row.avg_temp + 100 * row.avg_vib) / 2
42     weight = ((row.avg_temp or 0) + 100 * (row.avg_vib or 0)) / 2
43
44
45     output = {
46         "node_id": "fog-1",
47         "weight": round(weight, 4),
48         "samples": batch_df.count()
49     }
50
51     spark.createDataFrame(
52         [(json.dumps(output),)],
53         ["value"]
54     ).selectExpr("CAST(value AS STRING)") \
55     .write \
56     .format("kafka") \
57     .option("kafka.bootstrap.servers", "kafka:29092") \
58     .option("topic", "model-weights") \
59     .save()
60
61     print("Fog-1 -> poids envoy :", output)
62
63 query = model.writeStream \
64     .foreachBatch(send_weights) \
65     .outputMode("complete") \
66     .start()
67
68 query.awaitTermination()

```

Listing 6.1 – Fog node 1

6.3 Commande d'exécution

```
1 docker exec -it spark /opt/spark/bin/spark-submit
2 --conf spark.jars.ivy=/tmp/ivy
3 --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1
4 /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 :

```
1 avg_temp = row.avg_temp or 0
2 avg_vib = row.avg_vib or 0
3 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

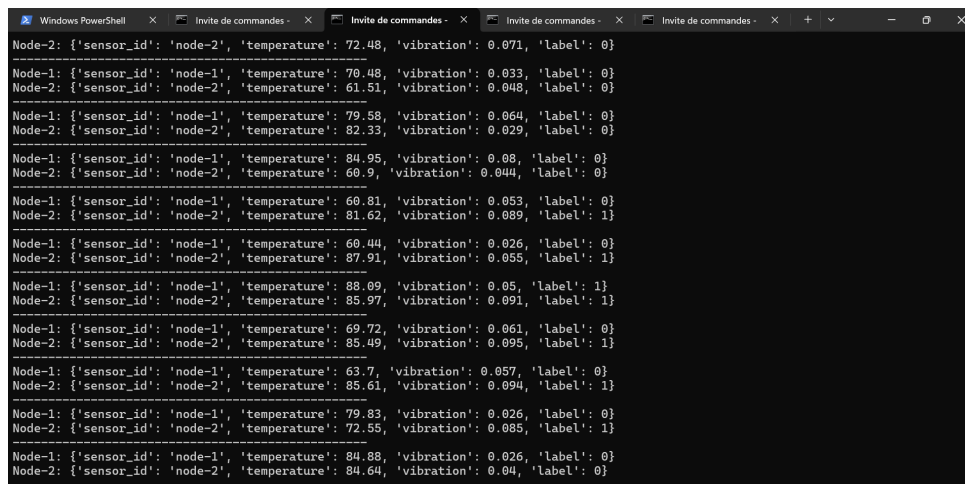
```
1 docker exec -it spark /opt/spark/bin/spark-submit
2 --conf spark.jars.ivy=/tmp/ivy
3 --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1
4 /app/cloud/aggregator.py
```

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

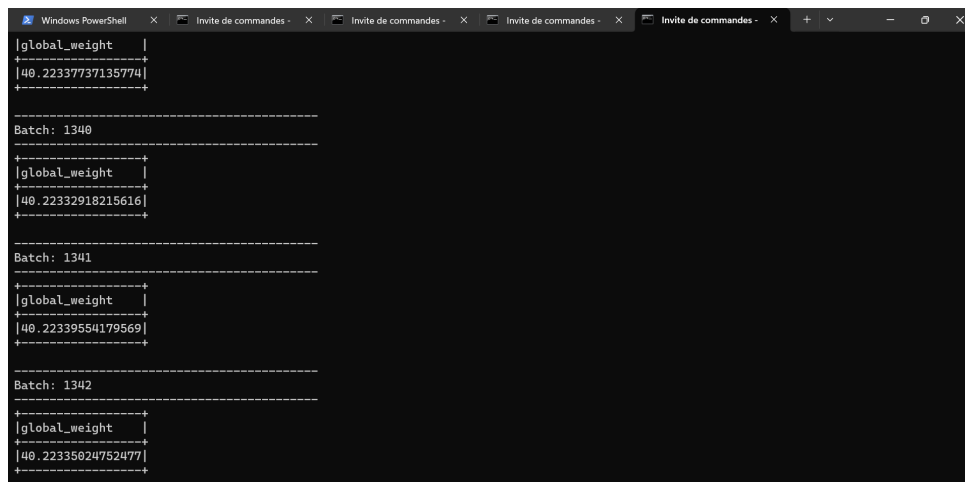
8.1.1 Producteur de données



```
Node-2: {'sensor_id': 'node-2', 'temperature': 72.48, 'vibration': 0.071, 'label': 0}
Node-1: {'sensor_id': 'node-1', 'temperature': 70.48, 'vibration': 0.033, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 61.51, 'vibration': 0.048, 'label': 0}
Node-1: {'sensor_id': 'node-1', 'temperature': 79.58, 'vibration': 0.064, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 82.33, 'vibration': 0.029, 'label': 0}
Node-1: {'sensor_id': 'node-1', 'temperature': 84.95, 'vibration': 0.08, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 60.9, 'vibration': 0.044, 'label': 0}
Node-1: {'sensor_id': 'node-1', 'temperature': 60.81, 'vibration': 0.053, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 81.62, 'vibration': 0.089, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 60.44, 'vibration': 0.026, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 87.91, 'vibration': 0.055, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 88.09, 'vibration': 0.05, 'label': 1}
Node-2: {'sensor_id': 'node-2', 'temperature': 85.97, 'vibration': 0.091, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 69.72, 'vibration': 0.061, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 85.49, 'vibration': 0.095, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 63.7, 'vibration': 0.057, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 85.61, 'vibration': 0.094, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 79.83, 'vibration': 0.026, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 72.55, 'vibration': 0.085, 'label': 1}
Node-1: {'sensor_id': 'node-1', 'temperature': 84.88, 'vibration': 0.026, 'label': 0}
Node-2: {'sensor_id': 'node-2', 'temperature': 84.64, 'vibration': 0.04, 'label': 0}
```

FIGURE 8.1 – Production des données capteurs vers Kafka

8.1.3 Agrégation Cloud



```
Windows PowerShell X Invite de commandes X Invite de commandes X Invite de commandes X Invite de commandes X + - X

|global_weight |
+-----+
|40.22337737135774|
+-----+

Batch: 1340

+-----+
|global_weight |
+-----+
|40.22332918215616|
+-----+

Batch: 1341

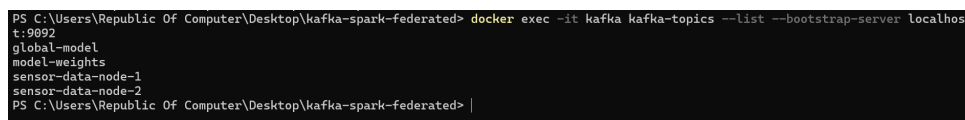
+-----+
|global_weight |
+-----+
|40.22339554179569|
+-----+

Batch: 1342

+-----+
|global_weight |
+-----+
|40.22335024752477|
+-----+
```

FIGURE 8.4 – Agrégation globale des poids dans le Cloud

8.1.4 Topics Kafka



```
PS C:\Users\Republic Of Computer\Desktop\kafka-spark-federated> docker exec -it kafka kafka-topics --list --bootstrap-server localhost:9092
global-model
model-weights
sensor-data-node-1
sensor-data-node-2
PS C:\Users\Republic Of Computer\Desktop\kafka-spark-federated> |
```

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.