Pipeline Big Data IoT:

Traitement en temps réel de données capteurs

Réalisée par : Ben Imran Ahlam

Encadré par : Mr.Hassan Badir

1. Introduction

L'Internet des Objets (IoT) connaît une croissance exponentielle avec plus de 25 milliards d'appareils connectés générant des volumes massifs de données en continu. Ces flux, caractérisés par leur vélocité et variété, nécessitent des infrastructures adaptées capables de les traiter en temps réel. Les systèmes traditionnels ne peuvent répondre aux exigences de latence et de débit imposées par les applications IoT modernes. Ce projet vise à concevoir et déployer une architecture complète de traitement de données IoT en temps réel, simulant 100 capteurs distribués et démontrant l'ensemble du cycle : ingestion via Kafka, traitement streaming avec Spark, et stockage optimisé dans HDFS.

2. Objectifs du projet

. **Ingestion** : Simuler 100 capteurs IoT générant 1000 événements/seconde via Apache Kafka

. **Traitement** : Implémenter un processeur Spark Streaming avec agrégations par fenêtre de 5 minutes et groupement par région

. **Stockage** : Sauvegarder dans HDFS au format Parquet avec partitionnement par région et compression Snappy

. **Résilience** : Garantir la tolérance aux pannes via checkpointing et réplication

3. Architecture globale

L'architecture du projet repose sur les composants suivants :

1. Apache Zookeeper

Rôle : Coordination et gestion de la configuration distribuée

2. Apache Kafka

Rôle : Message Broker haute performance pour le streaming de données

3. Apache Spark Streaming

Rôle : Moteur de traitement distribué pour les flux de données

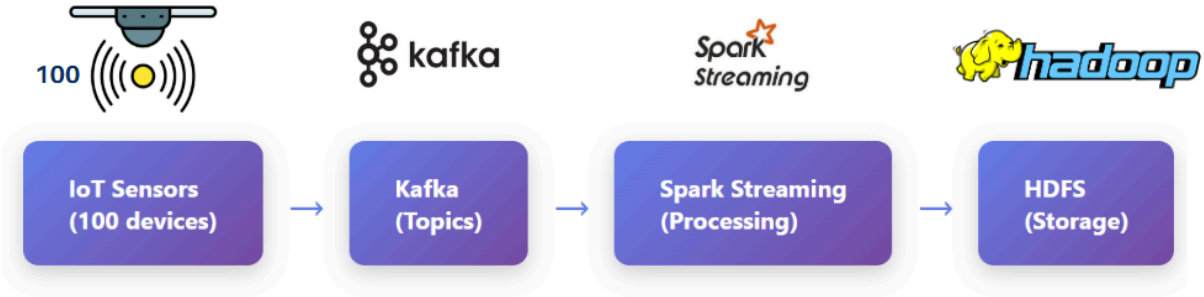
4. HDFS (Hadoop Distributed File System)

Rôle : Système de fichiers distribué pour le stockage à grande échelle

5.Spark Master & Worker

Rôle : Gestion du cluster Spark

- **Spark Master** : Coordonne les jobs et alloue les ressources
- **Spark Worker** : Exécute les tâches de calcul



4. Environnement technique

- Système hôte : Windows
- Conteneurisation : Docker
- Orchestration : Docker Compose
- Langages : Python / Spark SQL
- Format de données : JSON, Parquet

Lancement de tous les services :

```
PS C:\Users\lenovo\iot-bigdata-pipeline> docker-compose up -d
[+] Running 6/6
 ✔ Container spark-master   Started
 ✔ Container namenode       Running
 ✔ Container zookeeper      Running
 ✔ Container kafka          Running
 ✔ Container datanode       Running
 ✔ Container spark-worker   Started
```

Vérification de l'état : *6 conteneurs RUNNING*

```
PS C:\Users\lenovo\iot-bigdata-pipeline> docker ps
>>
CONTAINER ID   IMAGE                                          COMMAND                  CREATED             STATUS                       PORTS
                NAMES
c149679f6d49   apache/spark:3.5.0                             "/opt/entrypoint.sh …"   15 seconds ago      Up 13 seconds
                spark-worker
fdef00fcf98f   apache/spark:3.5.0                             "/opt/entrypoint.sh …"   50 seconds ago      Up 48 seconds                0.0.0.0:7077->7077/tcp, 0.0
.0.0:8080->8080/tcp   spark-master
39579b061471   bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8   "/entrypoint.sh /run…"   About an hour ago   Up About an hour (healthy)   9864/tcp
                datanode
746eece60be3   confluentinc/cp-kafka:7.5.0                    "/etc/confluent/dock…"   About an hour ago   Up About an hour             0.0.0.0:9092->9092/tcp
                kafka
21a0330325b6   confluentinc/cp-zookeeper:7.5.0                "/etc/confluent/dock…"   About an hour ago   Up About an hour             2888/tcp, 0.0.0.0:2181->218
1/tcp, 3888/tcp       zookeeper
9cadee1a271d   bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8   "/entrypoint.sh /run…"   About an hour ago   Up About an hour (healthy)   0.0.0.0:9000->9000/tcp, 0.0
.0.0:9870->9870/tcp   namenode
```

5. Mise en place de Kafka

Kafka est utilisé pour recevoir les données IoT sous forme de messages.

IoT Data Producer (Scala)

```scala
def createProducer(): KafkaProducer[String, String] = {
  val props = new Properties()
  props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092")
  props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
  props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
  props.put(ProducerConfig.ACKS_CONFIG, "all")

  new KafkaProducer[String, String](props)
}
```

5.1 Création du topic

Un topic Kafka dédié est créé pour les données brutes :

- Nom : `iot-sensors-raw`
- Partitions : 4
- Facteur de réplication : 1

Kafka permet ainsi une ingestion scalable et tolérante aux pannes.

```
PS C:\Users\lenovo\iot-bigdata-pipeline> docker exec kafka kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
iot-alerts
iot-sensors-raw
```

6. Stockage avec HDFS

HDFS est utilisé pour stocker les données à grande échelle.

- Répartition des données sur plusieurs nœuds
- Tolérance aux pannes grâce à la réplication
- Adapté aux traitements Big Data

Les données traitées sont stockées dans HDFS sous format optimisé.

```
PS C:\Users\lenovo\iot-bigdata-pipeline> docker exec namenode hadoop fs -ls /iot
Found 2 items
drwxr-xr-x   - root supergroup          0 2025-12-23 14:45 /iot/aggregations
drwxr-xr-x   - root supergroup          0 2025-12-24 08:36 /iot/raw-data
```

7. Traitement des données avec Apache Spark

Apache Spark est utilisé pour :

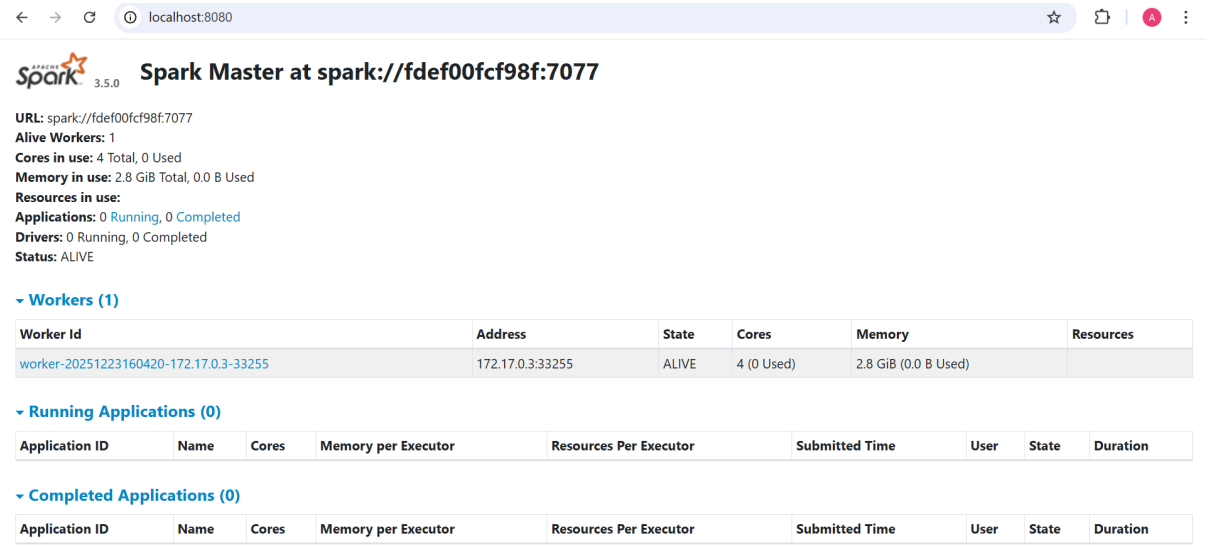
- Lire les données depuis Kafka
- Nettoyer et transformer les données
- Effectuer des agrégations
- Sauvegarder les résultats dans HDFS

Spark offre de meilleures performances que MapReduce grâce au traitement en mémoire.

Spark Streaming Processor (Scala)

```scala
val rawStream = spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "kafka:29092")
  .option("subscribe", "iot-sensors-raw")
  .option("startingOffsets", "earliest")
  .load()

val parsedStream = rawStream
  .selectExpr("CAST(value AS STRING) as json") // Convertir bytes en string
  .select(from_json($"json", sensorSchema).as("data")) // Parser JSON
  .select("data.*") // Extraire les colonnes
  .withColumn("event_time", to_timestamp(from_unixtime($"timestamp" / 1000))) // Timestamp Unix → Timestamp SQL

val windowedAggregations = parsedStream
  .withWatermark("event_time", "10 minutes")
  .groupBy(
    window($"event_time", "5 minutes"),
    $"region"
  )
  .agg(
    avg("temperature").as("avg_temperature"),
    max("temperature").as("max_temperature"),
    count("*").as("sensor_count")
  )
```

- **Spark Master** : http://localhost:8080 (cluster info)



8. Streaming avec Spark Structured Streaming

Le projet utilise **Spark Structured Streaming** pour :

- Consommer les messages Kafka en continu
- Traiter les données quasi en temps réel
- Gérer les fenêtres temporelles et les agrégations

Partitionnement Hive-Style:

Structure créée :

```
root@ce8e90c25b7f:/# hdfs dfs -ls /iot/raw-data
Found 5 items
drwxr-xr-x   - spark supergroup          0 2025-12-24 08:51 /iot/raw-data/_spark_metadata
drwxr-xr-x   - spark supergroup          0 2025-12-24 08:51 /iot/raw-data/region=East
drwxr-xr-x   - spark supergroup          0 2025-12-24 08:51 /iot/raw-data/region=North
drwxr-xr-x   - spark supergroup          0 2025-12-24 08:51 /iot/raw-data/region=South
drwxr-xr-x   - spark supergroup          0 2025-12-24 08:51 /iot/raw-data/region=West
```

Avantages :

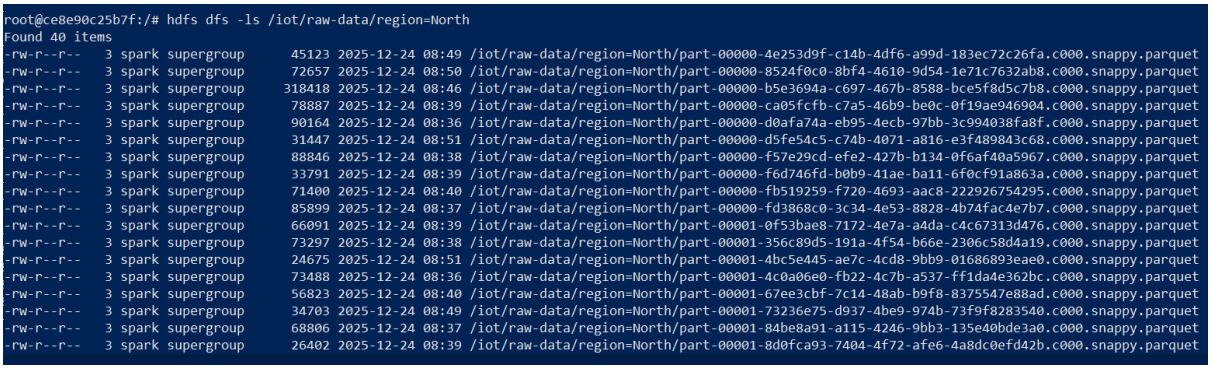
1. **Partition pruning** : WHERE region='North' → lit seulement 1/4 des données
2. **Parallélisme** : Chaque partition peut être lue en parallèle
3. **Organisation logique** : Facile à comprendre et maintenir

9. Format Parquet et partitionnement

Les données finales sont stockées en **Parquet**, car :

- Compression efficace
- Lecture optimisée par colonnes
- Performances élevées pour l'analyse

Le partitionnement améliore la vitesse des requêtes.



```
root@ce8e90c25b7f:/# hdfs dfs -ls /iot/raw-data/region=North
Found 40 items
-rw-r--r--   3 spark supergroup      45123 2025-12-24 08:49 /iot/raw-data/region=North/part-00000-4e253d9f-c14b-4df6-a99d-183ec72c26fa.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      72657 2025-12-24 08:50 /iot/raw-data/region=North/part-00000-8524f0c0-8bf4-4610-9d54-1e71c7632ab8.c000.snappy.parquet
-rw-r--r--   3 spark supergroup     318418 2025-12-24 08:46 /iot/raw-data/region=North/part-00000-b5e3694a-c697-467b-8588-bce5f8d5c7b8.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      78887 2025-12-24 08:39 /iot/raw-data/region=North/part-00000-ca05fcfb-c7a5-46b9-be0c-0f19ae946904.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      90164 2025-12-24 08:36 /iot/raw-data/region=North/part-00000-d0afa74a-eb95-4ecb-97bb-3c994038fa8f.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      31447 2025-12-24 08:51 /iot/raw-data/region=North/part-00000-d5fe54c5-c74b-4071-a816-e3f489843c68.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      88846 2025-12-24 08:38 /iot/raw-data/region=North/part-00000-f57e29cd-efe2-427b-b134-0f6af40a5967.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      33791 2025-12-24 08:39 /iot/raw-data/region=North/part-00000-f6d746fd-b0b9-41ae-ba11-6f0cf91a863a.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      71400 2025-12-24 08:40 /iot/raw-data/region=North/part-00000-fb519259-f720-4693-aac8-222926754295.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      85899 2025-12-24 08:37 /iot/raw-data/region=North/part-00000-fd3868c0-3c34-4e53-8828-4b74fac4e7b7.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      66091 2025-12-24 08:39 /iot/raw-data/region=North/part-00001-0f53bae8-7172-4e7a-a4da-c4c67313d476.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      73297 2025-12-24 08:38 /iot/raw-data/region=North/part-00001-356c89d5-191a-4f54-b66e-2306c58d4a19.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      24675 2025-12-24 08:51 /iot/raw-data/region=North/part-00001-4bc5e445-ae7c-4cd8-9bb9-01686893eae0.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      73488 2025-12-24 08:36 /iot/raw-data/region=North/part-00001-4c0a06e0-fb22-4c7b-a537-ff1da4e362bc.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      56823 2025-12-24 08:40 /iot/raw-data/region=North/part-00001-67ee3cbf-7c14-48ab-b9f8-8375547e88ad.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      34703 2025-12-24 08:49 /iot/raw-data/region=North/part-00001-73236e75-d937-4be9-974b-73f9f8283540.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      68806 2025-12-24 08:37 /iot/raw-data/region=North/part-00001-84be8a91-a115-4246-9bb3-135e40bde3a0.c000.snappy.parquet
-rw-r--r--   3 spark supergroup      26402 2025-12-24 08:39 /iot/raw-data/region=North/part-00001-8d0fca93-7404-4f72-afe6-4a8dc0efd42b.c000.snappy.parquet
```

10. Conteneurisation avec Docker

Chaque composant (**Kafka, Spark, HDFS**) est déployé dans un conteneur Docker.

Avantages :

- Isolation des services
- Déploiement rapide
- Reproductibilité de l'environnement

11. Orchestration avec Docker Compose

Docker Compose permet :

- De lancer tous les services avec une seule commande
- De gérer les réseaux et volumes
- D'éviter les conflits de configuration

12. Résultats obtenus

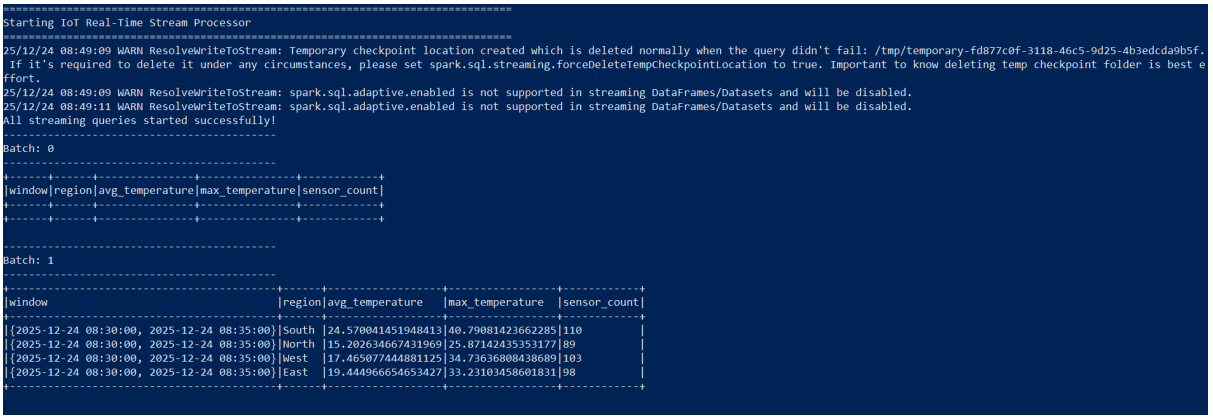
Apres avoir compiler le projet:

```
PS C:\Users\lenovo\iot-bigdata-pipeline> sbt clean assembly
>>
[info] welcome to sbt 1.11.7 (Oracle Corporation Java 21.0.9)
[info] loading settings for project iot-bigdata-pipeline-build from plugins.sbt...
[info] loading project definition from C:\Users\lenovo\iot-bigdata-pipeline\project
[info] loading settings for project iot-bigdata-pipeline from build.sbt...
[info] set current project to IoT-BigData-Pipeline (in build file:/C:/Users/lenovo/iot-bigdata-pipeline/)
[success] Total time: 0 s, completed 23 d├®c. 2025, 21:09:12
[info] compiling 2 Scala sources to C:\Users\lenovo\iot-bigdata-pipeline\target\scala-2.12\classes ...
[warn] multiple main classes detected: run 'show discoveredMainClasses' to see the list
[info] 7 file(s) merged using strategy 'First' (Run the task at debug level to see the details)
[info] 282 file(s) merged using strategy 'Discard' (Run the task at debug level to see the details)
[info] Built: C:\Users\lenovo\iot-bigdata-pipeline\target\scala-2.12\app.jar
[info] Jar hash: af0f17a735c0ca142a98a4ac770bdaf61d8d3dc8
[success] Total time: 140 s (0:02:20.0), completed 23 d├®c. 2025, 21:11:32
←[0J
PS C:\Users\lenovo\iot-bigdata-pipeline>
PS C:\Users\lenovo\iot-bigdata-pipeline>
PS C:\Users\lenovo\iot-bigdata-pipeline> dir target\scala-2.12\
>>


    Répertoire : C:\Users\lenovo\iot-bigdata-pipeline\target\scala-2.12


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        23/12/2025     21:09                classes
d-----        23/12/2025     21:09                sync
d-----        23/12/2025     21:09                update
d-----        23/12/2025     21:09                zinc
-a----        23/12/2025     21:11       71889229 app.jar
```

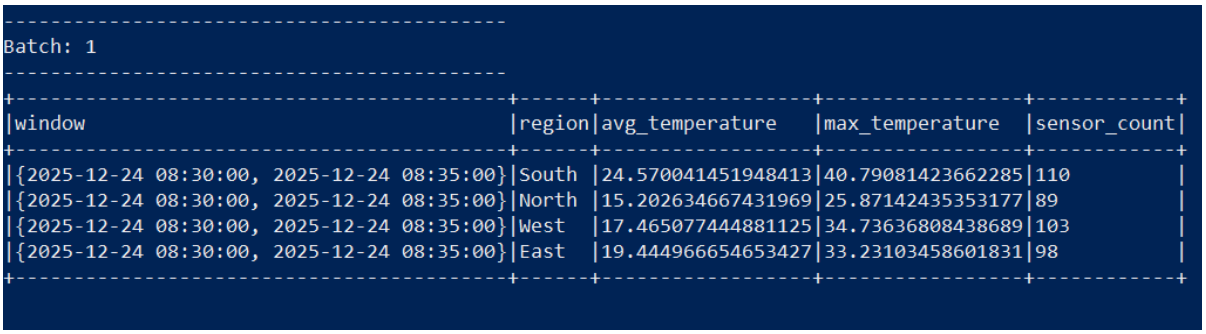
On lance le **IotDataProducer** :

```
PS C:\Users\lenovo\iot-bigdata-pipeline> sbt "runMain IoTDataProducer"
[info] welcome to sbt 1.11.7 (Oracle Corporation Java 21.0.9)
[info] loading settings for project iot-bigdata-pipeline-build from plugins.sbt...
[info] loading project definition from C:\Users\lenovo\iot-bigdata-pipeline\project
[info] loading settings for project iot-bigdata-pipeline from build.sbt...
[info] set current project to IoT-BigData-Pipeline (in build file:/C:/Users/lenovo/iot-bigdata-pipeline/)
[info] running IoTDataProducer
Starting IoT Data Producer - Topic: iot-sensors-raw
Sent 1000 messages
Sent 2000 messages
Sent 3000 messages
Sent 4000 messages
Sent 5000 messages
Sent 6000 messages
Sent 7000 messages
Sent 8000 messages
Sent 9000 messages
Sent 10000 messages
Sent 11000 messages
Sent 12000 messages
Sent 13000 messages
Sent 14000 messages
Sent 15000 messages
Sent 16000 messages
Sent 17000 messages
```

On lance **SparkStreamingProcessor**:



On peut voir les donnees affichees :



- Pipeline fonctionnelle de bout en bout
- Ingestion continue des données IoT
- Traitement et stockage fiables
- Architecture scalable et modulaire

13. Conclusion

Ce projet démontre la mise en place réussie d’une pipeline Big Data IoT moderne. L’utilisation combinée de Kafka, Spark et HDFS permet de gérer efficacement les données massives générées par les objets connectés, tout en assurant performance, fiabilité et évolutivité.