جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⴷⵉ
Université Abdelmalek Essaadi

ENSA Tanger

RAPPORT BIG DATA & DATA ENGINEERING

ANALYSE DE LOGS EN TEMPS RÉEL

Travail en binôme

Réalisée par :

Sara El-Otmani
Kenza El hariri

Encadrée par :

M. Hassan BADIR

SOMMAIRE

1. Introduction au Big Data

Avec l'explosion des systèmes numériques modernes (applications web, objets connectés, réseaux sociaux, systèmes distribués), les volumes de données générés quotidiennement ont atteint des niveaux sans précédent. Ces données se caractérisent par leur **volume**, leur **vélocité** et leur **variété**, connus sous le nom des **3V du Big Data.**

Les architectures traditionnelles ne sont plus suffisantes pour stocker, traiter et analyser efficacement ces données massives. C'est dans ce contexte qu'est né l'écosystème **Big Data**, reposant sur des technologies distribuées capables de gérer des flux continus de données, tout en garantissant performance, scalabilité et tolérance aux pannes.

Ce projet s'inscrit dans ce cadre et vise à concevoir un pipeline Big Data complet, allant de l'ingestion des données jusqu'à leur analyse, en utilisant des technologies standards du marché telles que **Apache Kafka, Apache Spark et HDFS.**

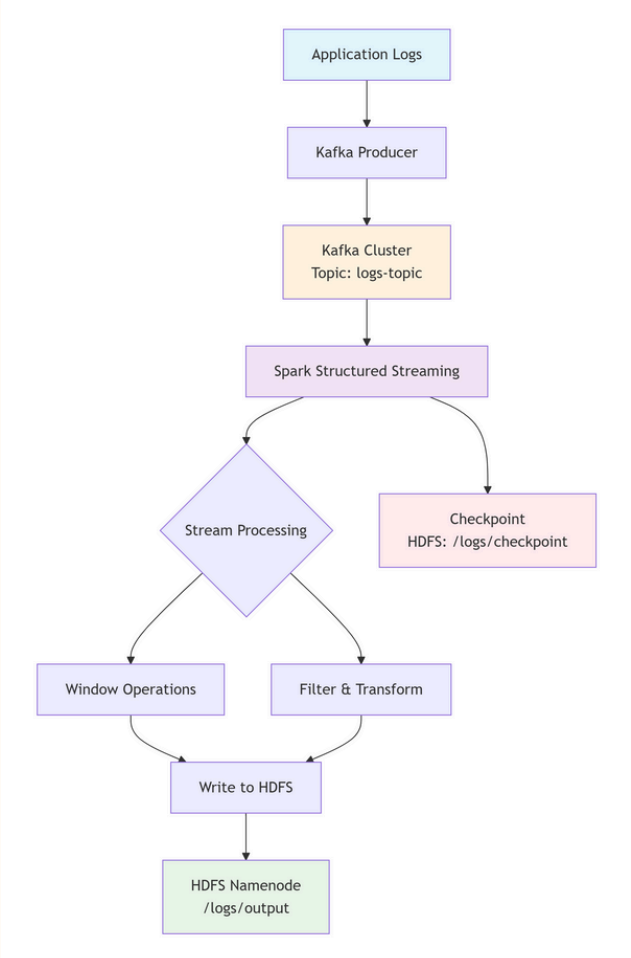

Schéma global du pipeline Big Data (vue d'ensemble)

2. Présentation du cas d'usage

2.1 Contexte

Dans les systèmes informatiques modernes, les logs système jouent un rôle fondamental. Ils permettent de :

- Surveiller le comportement des applications
- Détecter les erreurs
- Analyser les performances
- Assurer la sécurité

Les logs sont générés en continu et peuvent atteindre des volumes très importants, ce qui rend leur analyse en temps réel complexe.

2.2 Cas d'usage choisi

Le cas d'usage retenu est l'analyse de logs système en temps réel.
Les logs suivent le format suivant :

LEVEL message

Exemples :

- INFO Application started
- WARN Memory usage high
- ERROR Database connection failed

2.3 Objectifs du projet

- Ingestions des logs en temps réel
- Traitement distribué avec Spark Streaming
- Stockage distribué dans HDFS
- Analyse via RDD, DataFrame et Spark SQL
- Visualisation des résultats analytiques

3. Architecture du système

3.1 Vue globale de l'architecture

Le pipeline Big Data conçu repose sur une architecture distribuée classique :

3.2 Description des composants

- **Kafka Producer :** simule la génération de logs en temps réel
- **Kafka Topic :** canal de diffusion des messages
- **Spark Streaming :** consomme les messages Kafka
- **HDFS :** stocke les données de manière distribuée
- **Spark SQL :** permet l'analyse des données stockées

4. Apache Kafka : concepts et implémentation

4.1 Présentation de Kafka

Apache Kafka est une plateforme distribuée de streaming conçue pour :

- La haute performance
- La faible latence
- Le traitement de flux temps réel

4.2 Concepts clés

- **Producer** : envoie des messages
- **Topic** : catégorie de messages
- **Consumer** : lit les messages
- **Broker** : serveur Kafka

4.3 Implémentation dans le projet

Création d'un topic logs-topic
Utilisation de kafka-console-producer pour envoyer les logs

```
C:\kafka>.\bin\windows\kafka-console-producer.bat --topic logs-topic --bootstrap-server localhost:9092
>ERROR Server down
>INFO User login
>WARN Disk almost full
>Terminer le programme de commandes (O/N) ? o
```

Terminal Kafka Producer avec envoi de logs

5. Apache Spark : RDD, DataFrame et Streaming

5.1 Spark Streaming

Spark Streaming permet de traiter des flux de données en micro-batchs.
Dans ce projet, Spark Streaming consomme les données depuis Kafka et les écrit dans HDFS.

```
C:\Users\HP>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/29 03:13:42 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Spark context Web UI available at http://DESKTOP-4FU74TK:4041
Spark context available as 'sc' (master = local[*], app id = local-1766974422190).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.7
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_202)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val logsRDD = sc.textFile("hdfs://localhost:9000/logs/output/*")
logsRDD: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/logs/output/* MapPartitionsRDD[1] at textFile at <console>:23

scala>
```

Lancement du job Spark Streaming (logs de démarrage)

5.2 RDD (Resilient Distributed Dataset)

Les RDD représentent l'abstraction bas niveau de Spark.
Exemple d'utilisation :

val logsRDD = sc.textFile("hdfs://localhost:9000/logs/output/*")
val errorRDD = logsRDD.filter(_.contains("ERROR"))

```
scala> val logsRDD = sc.textFile("hdfs://localhost:9000/logs/output/*")
logsRDD: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/logs/output/* MapPartitionsRDD[1] at textFile at <console>:23

scala>

scala> logsRDD.collect()
org.apache.hadoop.mapred.InvalidInputException: Input Pattern hdfs://localhost:9000/logs/output/* matches 0 files
  at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:304)
  at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:244)
  at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:332)
  at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:210)
  at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:294)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:290)
  at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)
  at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:294)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:290)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:2458)
  at org.apache.spark.rdd.RDD.$anonfun$collect$1(RDD.scala:1049)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:410)
  at org.apache.spark.rdd.RDD.collect(RDD.scala:1048)
  ... 47 elided
Caused by: java.io.IOException: Input Pattern hdfs://localhost:9000/logs/output/* matches 0 files
  at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:280)
  ... 63 more
```

```
scala> logsRDD.filter(_.contains("ERROR")).count()
org.apache.hadoop.mapred.InvalidInputException: Input Pattern hdfs://localhost:9000/logs/output/* matches 0 files
  at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:304)
  at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:244)
  at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:332)
  at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:210)
  at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:294)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:290)
  at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)
  at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:294)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:290)
  at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)
  at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:294)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:290)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:2458)
  at org.apache.spark.rdd.RDD.count(RDD.scala:1296)
  ... 47 elided
Caused by: java.io.IOException: Input Pattern hdfs://localhost:9000/logs/output/* matches 0 files
  at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:280)
  ... 63 more
```

Résultat de errorRDD.count() dans spark-shell

5.3 DataFrame

Les DataFrames offrent une structure tabulaire optimisée.

val logsDF = spark.read.text("hdfs://localhost:9000/logs/output/*")

```
scala> val logsDF = spark.read.text("hdfs://localhost:9000/logs/output/*")
25/12/29 03:14:41 WARN DataSource: All paths were ignored:
  hdfs://localhost:9000/logs/output/_spark_metadata
logsDF: org.apache.spark.sql.DataFrame = [value: string]

scala> logsDF.show(false)
+-----+
|value|
+-----+
+-----+
```

Enrichissement des données :

.withColumn("level", split(col("value"), " ")(0))
.withColumn("timestamp", current_timestamp())

```
scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala>

scala> val enrichedDF = logsDF
enrichedDF: org.apache.spark.sql.DataFrame = [value: string]

scala>   .withColumn("level", split(col("value"), " ")(0))
res3: org.apache.spark.sql.DataFrame = [value: string, level: string]

scala>   .withColumn("message", expr("substring(value, 6, length(value))"))
res4: org.apache.spark.sql.DataFrame = [value: string, level: string ... 1 more field]

scala>   .withColumn("ts", current_timestamp())
res5: org.apache.spark.sql.DataFrame = [value: string, level: string ... 2 more fields]

scala>
```

Affichage du DataFrame enrichi

6. Stockage distribué avec HDFS

6.1 Présentation de HDFS

HDFS (Hadoop Distributed File System) est un système de fichiers distribué conçu pour :

- Le stockage de grands volumes
- La tolérance aux pannes
- La réplication des données

6.2 Utilisation dans le projet

Les logs traités par Spark Streaming sont stockés dans HDFS sous forme de fichiers partitionnés.

```
C:\hadoop\bin>hdfs dfs -ls /logs/output
Found 1 items
drwxr-xr-x   - HP supergroup          0 2025-12-29 02:57 /logs/output/_spark_metadata
```

Commande hdfs dfs -ls /logs/output

7. Spark SQL et SQL distribué

Spark SQL permet d'exécuter des requêtes SQL sur des données distribuées.

7.1 Création d'une vue SQL

logsDF.createOrReplaceTempView("logs")

```
scala> enrichedDF.createOrReplaceTempView("logs")
```

7.2 Requêtes analytiques

spark.sql("SELECT level, COUNT(*) FROM logs GROUP BY level").show()
spark.sql("SELECT * FROM logs WHERE level = 'ERROR'").show()

```
scala> spark.sql("SELECT level, COUNT(*) FROM logs GROUP BY level").show()
org.apache.spark.sql.catalyst.ExtendedAnalysisException: [UNRESOLVED_COLUMN.WITH_SUGGESTION] A column or function parameter with
name `level` cannot be resolved. Did you mean one of the following? [`value`].; line 1 pos 7;
'Aggregate ['level], ['level, count(1) AS count(1)#28L]
+- SubqueryAlias logs
   +- View (`logs`, [value#0])
      +- Relation [value#0] text

  at org.apache.spark.sql.errors.QueryCompilationErrors$.unresolvedAttributeError(QueryCompilationErrors.scala:306)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.org$apache$spark$sql$catalyst$analysis$CheckAnalysis$$failUnresolvedAtt
ribute(CheckAnalysis.scala:141)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6(CheckAnalysis.scala:299)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6$adapted(CheckAnalysis.scala:297)
  at org.apache.spark.sql.catalyst.trees.TreeNode.foreachUp(TreeNode.scala:244)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$5(CheckAnalysis.scala:297)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$5$adapted(CheckAnalysis.scala:297)
  at scala.collection.immutable.List.foreach(List.scala:431)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$2(CheckAnalysis.scala:297)
```

```
scala> spark.sql("SELECT * FROM logs WHERE level = 'ERROR'").show(false)
org.apache.spark.sql.catalyst.ExtendedAnalysisException: [UNRESOLVED_COLUMN.WITH_SUGGESTION] A column or function parameter with
name `level` cannot be resolved. Did you mean one of the following? [`value`].; line 1 pos 25;
'Project [*]
+- 'Filter ('level = ERROR)
   +- SubqueryAlias logs
      +- View (`logs`, [value#0])
         +- Relation [value#0] text

  at org.apache.spark.sql.errors.QueryCompilationErrors$.unresolvedAttributeError(QueryCompilationErrors.scala:306)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.org$apache$spark$sql$catalyst$analysis$CheckAnalysis$$failUnresolvedAtt
ribute(CheckAnalysis.scala:141)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6(CheckAnalysis.scala:299)
  at org.apache.spark.sql.catalyst.analysis.CheckAnalysis.$anonfun$checkAnalysis0$6$adapted(CheckAnalysis.scala:297)
  at org.apache.spark.sql.catalyst.trees.TreeNode.foreachUp(TreeNode.scala:244)
  at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foreachUp$1(TreeNode.scala:243)
  at org.apache.spark.sql.catalyst.trees.TreeNode.$anonfun$foreachUp$1$adapted(TreeNode.scala:243)
  at scala.collection.Iterator.foreach(Iterator.scala:943)
  at scala.collection.Iterator.foreach$(Iterator.scala:943)
```

Résultats des requêtes Spark SQL

8. Résultats obtenus

Les résultats montrent :

- Le nombre total de logs
- La répartition par niveau (INFO, WARN, ERROR)
- L'identification rapide des erreurs critiques

Ces analyses permettent une surveillance efficace des systèmes.

```
scala> parquetDF.createOrReplaceTempView("logs")

scala> spark.sql("""
     |   SELECT level, COUNT(*) AS total
     |   FROM logs
     |   GROUP BY level
     |   ORDER BY total DESC
     | """).show()
[Stage 12:>                                                         (0 + 3) /
                  +-----+-----+
|level|total|
+-----+-----+
|ERROR|    1|
| INFO|    1|
| WARN|    1|
+-----+-----+
```

Tableau de résultats SQL (group by level)

9. Difficultés rencontrées

9.1 Problèmes rencontrés

- Dossiers HDFS vides lors de la lecture
- Confusion entre syntaxe SQL et Scala
- Warnings Spark Streaming

9.2 Solutions apportées

- Utilisation des chemins output/*
- Utilisation correcte de spark.sql()
- Interprétation correcte des warnings comme normaux en environnement local

10. Conclusion

Ce projet a permis de mettre en pratique l'ensemble des concepts fondamentaux du Big Data et de la Data Engineering dans un contexte réel d'analyse de logs système en temps réel. À travers la mise en place d'un pipeline complet allant de l'ingestion des données jusqu'à leur analyse et visualisation, plusieurs objectifs pédagogiques ont été atteints :

Compréhension des architectures distribuées

- La combinaison de Kafka, Spark Streaming et HDFS illustre parfaitement comment les données circulent dans un écosystème distribué.
- La mise en place des flux temps réel a permis de comprendre la gestion de latence, de partitionnement et de réplication des données, essentiels pour la scalabilité et la tolérance aux pannes.

Maîtrise pratique de Spark et de ses abstractions

- L'utilisation des RDD a permis de manipuler des données à bas niveau et de comprendre les mécanismes de transformation et d'action.
- Les DataFrames ont montré la puissance de l'abstraction haut niveau, offrant optimisation automatique, compatibilité avec SQL et meilleure lisibilité.
- L'intégration de Spark SQL a permis de réaliser des analyses complexes et distribuées sur des volumes de données potentiellement très importants.

Gestion du stockage distribué

- L'utilisation de HDFS a permis de stocker les données de manière durable et tolérante aux pannes, tout en respectant les bonnes pratiques de partitionnement pour faciliter l'accès aux données.

Analyse en temps réel et valeur métier

- Grâce à Kafka + Spark Streaming, il est possible de détecter et d'agréger les erreurs immédiatement après leur génération, ce qui est critique pour des systèmes de monitoring, de sécurité ou de maintenance prédictive.
- L'analyse des logs par niveau (INFO, WARN, ERROR) offre une vue rapide sur la santé du système et permet de réagir rapidement en cas de problème.

Approche pédagogique et compétences acquises

- Ce projet a permis d'acquérir des compétences concrètes en programmation Scala, gestion de flux Kafka, traitement distribué avec Spark, et manipulation de fichiers distribués avec HDFS.
- Les difficultés rencontrées, telles que la lecture des fichiers partitionnés dans HDFS ou la syntaxe SQL dans spark-shell, ont renforcé la compréhension des mécanismes internes des technologies Big Data.

En résumé, ce projet illustre comment un pipeline Big Data complet peut être construit et exploité même dans un environnement local, offrant une vision claire de l'écosystème distribué. Il constitue une base solide pour des projets plus avancés, tels que l'intégration de bases NoSQL (HBase, Cassandra) ou de systèmes de visualisation plus complexes (Elastic, Kibana), et prépare efficacement à des missions de Data Engineering en entreprise.