

# Big Data Analytics

Résumé - Séance 1

Apache Spark

Pr EL GHALI Btihal

18 janvier 2026

## Table des matières

<b>1</b>	<b>Introduction au Big Data</b>	<b>2</b>
1.1	Définition . . . . .	2
1.2	Domaines d'application . . . . .	2
1.3	Types d'analyse de données . . . . .	2
<b>2</b>	<b>Apache Spark</b>	<b>2</b>
2.1	Définition . . . . .	2
2.2	Caractéristiques principales . . . . .	2
2.3	Spark vs Hadoop . . . . .	3
<b>3</b>	<b>Écosystème Spark</b>	<b>3</b>
3.1	Composants principaux . . . . .	3
<b>4</b>	<b>Architecture de Spark</b>	<b>3</b>
4.1	Composants de l'architecture . . . . .	3
4.2	Fonctionnement en 4 étapes . . . . .	4
<b>5</b>	<b>Installation de Spark</b>	<b>4</b>
5.1	Windows . . . . .	4
5.2	Linux . . . . .	4
<b>6</b>	<b>Langage Scala - Notions de base</b>	<b>4</b>
6.1	Caractéristiques . . . . .	4
6.2	Syntaxe de base . . . . .	4
<b>7</b>	<b>Scala vs Python dans Spark</b>	<b>5</b>
7.1	Création de RDD . . . . .	5
7.2	Transformations . . . . .	5
7.3	Actions . . . . .	6
<b>8</b>	<b>Points clés à retenir</b>	<b>6</b>

# 1 Introduction au Big Data

## 1.1 Définition

Le Big Data désigne des ensembles de données complexes et volumineux, provenant de nouvelles sources, qui ne peuvent être traités par des logiciels traditionnels.

## 1.2 Domaines d'application

- **Santé** : Analyse en temps réel de l'état des patients critiques, coordination des greffes
- **Gouvernement** : Sécurité nationale, surveillance des agences militaires et policières
- **Télécommunications** : Réduction du taux de désabonnement, amélioration de l'expérience client
- **Bancaire** : Transactions tolérantes aux pannes, détection de fraudes
- **Marché boursier** : Prédiction des mouvements de portefeuilles, analyse de la demande

## 1.3 Types d'analyse de données

1. **Analyse Descriptive** : Examiner le passé pour comprendre l'avenir (KPI, prospects)
2. **Analyse Diagnostique** : Comprendre pourquoi un événement s'est produit
3. **Analyse Prédictive** : Utiliser des données antérieures pour prévoir l'avenir
4. **Analyse Prescriptive** : La plus complète, combine tous les modèles pour créer des plans axés sur les données

# 2 Apache Spark

## 2.1 Définition

Apache Spark est une infrastructure de traitement parallèle open source permettant d'exécuter des applications analytiques sur des clusters de machines. C'est l'outil incontournable pour l'analyse de données en temps réel.

## 2.2 Caractéristiques principales

- **Polyglotte** : APIs disponibles en Java, Scala, Python et R
- **Vitesse** : Jusqu'à 100× plus rapide que Hadoop MapReduce en mémoire, 10× plus rapide sur disque
- **Formats multiples** : Support de Parquet, JSON, Hive, Cassandra, CSV, RDBMS
- **Calcul temps réel** : Traitement batch et streaming avec faible latence
- **Intégration Hadoop** : Compatibilité fluide avec l'écosystème Hadoop
- **Apprentissage automatique** : MLlib intégré pour le machine learning
- **Évaluation paresseuse** : Utilisation d'un DAG (Directed Acyclic Graph) pour optimiser l'exécution

## 2.3 Spark vs Hadoop

Hadoop	Spark
Traitement par lots (Batch)	Traitement batch et temps réel
Traitement parallèle	100× plus rapide en mémoire
Moins coûteux	10× plus rapide sur disque
Pas de temps réel	Plus coûteux
Tolérant aux pannes	Tolérance aux pannes (RDD)
Évolutivité élevée	APIs interactives (Java, Scala, Python)
Complexe (MapReduce)	Facile

## 3 Écosystème Spark

### 3.1 Composants principaux

- **Spark Core** : Moteur de base pour le traitement parallèle et distribué
  - Gestion de la mémoire et récupération en cas de pannes
  - Planification, distribution et surveillance des tâches
  - Interaction avec les systèmes de stockage
- **Spark SQL** : Module d'intégration du traitement relationnel
  - Interrogation via SQL ou Hive
  - Support de diverses sources de données
  - API DataFrame et API de source de données
- **Spark Streaming** : Traitement des données en temps réel
  - Traitement de flux à haut débit
  - Tolérant aux pannes
- **MLlib** : Bibliothèque d'apprentissage automatique
  - Algorithmes classiques (classification, régression, clustering)
  - Extraction de caractéristiques
  - Pipelines ML
  - Persistance des modèles
- **GraphX** : API pour les graphes et calcul parallèle
  - Resilient Distributed Property Graph
  - Multigraphe orienté avec propriétés

## 4 Architecture de Spark

### 4.1 Composants de l'architecture

Architecture basée sur le modèle Master-Slave avec deux abstractions principales :

- **Driver** : Exécute la fonction main() et gère :
  - Conservation des informations de l'application
  - Réponse aux requêtes utilisateur
  - Analyse, distribution et ordonnancement des tâches
- **Executors** : Responsables de :
  - Exécution du code assigné par le driver

- Rapport de l'état d'avancement au driver
- **SparkSession** : Point d'entrée pour accéder au driver

## 4.2 Fonctionnement en 4 étapes

1. **Soumission** : Le client soumet le code, le driver le convertit en DAG
2. **Planification** : Le driver convertit le DAG en plan d'exécution physique avec des tâches
3. **Négociation** : Le driver négocie les ressources avec le cluster manager qui lance les executors
4. **Exécution** : Le driver surveille les executors et planifie les tâches futures

## 5 Installation de Spark

### 5.1 Windows

1. Installation de Java 17
2. Configuration de JAVA\_HOME et JAVA\_PATH
3. Téléchargement et extraction de Spark 4.0.1
4. Ajout du fichier winutils.exe depuis <https://github.com/cdarlint/winutils>
5. Configuration de SPARK\_HOME et HADOOP\_HOME
6. Ajout de %SPARK\_HOME%\bin au PATH
7. Lancement : `spark-shell`

### 5.2 Linux

```
$ sudo apt update
$ sudo apt install default-jdk
$ wget https://archive.apache.org/dist/spark/...
$ tar xvf spark-3.0.3-bin-hadoop2.7.tgz
$ sudo mv spark-3.0.3-bin-hadoop2.7/ /opt/spark
$ export PATH=$PATH:/opt/spark/bin/
$ spark-shell
```

## 6 Langage Scala - Notions de base

### 6.1 Caractéristiques

- Créé en 2003 par Martin Odersky à l'EPFL
- Fonctionne sur la JVM
- Intersection de la programmation fonctionnelle et orientée objet

### 6.2 Syntaxe de base

Valeurs (immuables)

```
val x = 1 + 1
val x: Int = 1 + 1 // avec type explicite
```

### Variables (mutables)

```
var x = 1 + 1
x = 3 // reassignation possible
```

### Fonctions

```
val addOne = (x: Int) => x + 1
val add = (x: Int, y: Int) => x + y
```

### Méthodes

```
def add(x: Int, y: Int): Int = x + y
def getGreeting(name: String): String = {
  "Hello, " + name
}
```

### Classes

```
class Greeter(prefix: String, suffix: String) {
  def greet(name: String): Unit =
    println(prefix + name + suffix)
}
val greeter = new Greeter("Hello, ", "!")
```

### Classes de cas

```
case class Point(x: Int, y: Int)
val point = Point(1, 2) // sans 'new'
```

## 7 Scala vs Python dans Spark

### 7.1 Création de RDD

#### Collections parallélisées

```
// Scala
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)

# Python
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

#### Depuis un fichier

```
// Scala
val file = sc.textFile("test.txt")

# Python
file = sc.textFile("test.txt")
```

### 7.2 Transformations

```
// Scala
val map = file.map(l => l.split("_"))
val fmap = file.flatMap(l => l.split("_"))

# Python
```

```
file.map(lambda x: x.split(' '))  
file.flatMap(lambda x: x.split(' '))
```

## 7.3 Actions

### WordCount exemple

```
// Scala  
val words = f.flatMap(l => l.split(" "))  
                .map(word => (word, 1))  
words.reduceByKey(_ + _).collect  
  
# Python  
words = f.flatMap(lambda x: x.split(' '))\  
          .map(lambda x: (x, 1))  
words.reduceByKey(add).collect()
```

### Persistence

```
// Scala  
val w = f.flatMap(l => l.split(" "))  
          .map(word => (word, 1)).cache()  
  
# Python  
w = f.flatMap(lambda x: x.split(' '))\  
      .map(lambda x: (x, 1)).cache()
```

## 8 Points clés à retenir

- Spark est 100× plus rapide que Hadoop MapReduce en mémoire
- Architecture basée sur Driver et Executors
- Évaluation paresseuse via DAG pour optimisation
- Support de multiples langages (Scala, Python, Java, R)
- Écosystème complet : SQL, Streaming, ML, GraphX
- RDD comme abstraction de base pour les données distribuées
- Programmation fonctionnelle et immuabilité comme principes fondamentaux