

# Etapes du pipeline

## Guide des étapes des pipelines de données Big Data

### Introduction

Un **pipeline de données** est une série d'étapes automatisées permettant de collecter, transformer, enrichir et stocker des données dans un environnement Big Data. Chaque étape lit des données à partir d'une source, effectue des transformations spécifiques, puis écrit le résultat sur un **Data Lake** basé sur **HDFS (Hadoop Distributed File System)**. Ces résultats peuvent être réutilisés par d'autres étapes du pipeline.

Ce guide décrit les principales étapes d'un pipeline de données :

1. **Ingestion des données**
  2. **Nettoyage et validation des données**
  3. **Enrichissement des données**
- 

## 1. Ingestion des données

### 1.1. Définition

L'**ingestion des données** est la première étape d'un pipeline. Elle consiste à **collecter des données** à partir de différentes sources : bases de données, API, fichiers CSV, logs, flux en temps réel, etc.

### 1.2. Types d'ingestion

- **Batch Ingestion** : Traitement des données en lots à des intervalles réguliers.
- **Stream Ingestion** : Traitement des données en continu (ex. : Kafka, Flink).


### 1.3. Exemple d'ingestion avec PySpark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("DataIngestion").getOrCreate()
```

```
# Lecture d'un fichier CSV
df = spark.read.csv("hdfs:///datalake/raw_data.csv", header=True,
inferSchema=True)

# Écriture dans le Data Lake HDFS (bronze layer)
df.write.parquet("hdfs:///datalake/bronze/ingested_data.parquet")
```



---

## 2. Nettoyage et validation des données

### 2.1. Définition

Après l'ingestion, les données brutes peuvent contenir des erreurs, des doublons ou des valeurs manquantes. L'étape de **nettoyage et validation** vise à :

- Supprimer les doublons
- Gérer les valeurs nulles
- Valider les types de données
- Appliquer des règles de qualité des données

### 2.2. Techniques courantes

- Suppression des valeurs nulles : `dropna()`
- Remplacement des valeurs manquantes : `fillna()`
- Détection des anomalies : validation de schéma

### 2.3. Exemple de nettoyage avec PySpark

```
# Suppression des doublons
df_clean = df.dropDuplicates()

# Remplacement des valeurs manquantes
df_clean = df_clean.fillna({"age": 0, "city": "Unknown"})

# Validation des types de données
from pyspark.sql.functions import col
df_clean = df_clean.withColumn("age", col("age").cast("Integer"))

# Écriture des données nettoyées dans HDFS (silver layer)
df_clean.write.parquet("hdfs:///datalake/silver/cleaned_data.parquet")
```



## 3. Enrichissement des données

### 3.1. Définition

L'**enrichissement des données** consiste à ajouter des informations supplémentaires provenant d'autres sources pour **améliorer la valeur analytique** des données. Cela peut inclure des jointures, des calculs dérivés ou des intégrations de données externes.

### 3.2. Exemples d'enrichissement

- **Jointures** avec des bases de référence (ex. : codes postaux, données démographiques)
- **Calcul de nouvelles métriques** (ex. : indicateurs de performance)
- **Ajout de données issues d'APIs externes**

### 3.3. Exemple d'enrichissement avec PySpark

```
# Chargement des données nettoyées et des données externes
df_clean = spark.read.parquet("hdfs:///datalake/silver/cleaned_data.parquet")
df_external = spark.read.csv("hdfs:///datalake/external/demographics.csv",
                             header=True)

# Jointure pour enrichir les données
df_enriched = df_clean.join(df_external, on="city", how="left")

# Calcul d'un nouvel indicateur
df_enriched = df_enriched.withColumn("income_per_person", col("income") /
                                     col("population"))

# Écriture des données enrichies dans HDFS (gold layer)
df_enriched.write.parquet("hdfs:///datalake/gold/enriched_data.parquet")
```



## 4. Architecture du Data Lake : Bronze, Silver, Gold

Un pipeline de données suit souvent une architecture en **trois couches** :

- **Bronze (Raw Layer)** : Données brutes, ingérées sans modification.
- **Silver (Clean Layer)** : Données nettoyées et validées.
- **Gold (Enriched Layer)** : Données enrichies prêtes pour l'analyse ou le reporting.

Toutes ces couches sont stockées dans **HDFS** pour assurer la scalabilité et la résilience.

---

## 5. Exemple de Pipeline Complet

```
# Ingestion
raw_data = spark.read.csv("hdfs:///datalake/raw_data.csv", header=True,
inferSchema=True)
raw_data.write.parquet("hdfs:///datalake/bronze/ingested_data.parquet")

# Nettoyage
clean_data = raw_data.dropDuplicates().fillna({"age": 0}).withColumn("age",
col("age").cast("Integer"))
clean_data.write.parquet("hdfs:///datalake/silver/cleaned_data.parquet")

# Enrichissement
external_data = spark.read.csv("hdfs:///datalake/external/data.csv",
header=True)
enriched_data = clean_data.join(external_data, on="id", how="left")
enriched_data.write.parquet("hdfs:///datalake/gold/enriched_data.parquet")
```



---

## 6. Bonnes pratiques pour les pipelines de données

- **Automatisation des pipelines** avec des outils comme Apache Airflow
  - **Surveillance de la qualité des données** à chaque étape
  - **Gestion des erreurs** pour assurer la robustesse des workflows
  - **Optimisation des performances** via le partitionnement et la parallélisation
  - **Utilisation efficace de HDFS** pour la gestion du stockage distribué
- 

## Conclusion

Un pipeline de données efficace repose sur des étapes bien définies : **ingestion**, **nettoyage/validation**, et **enrichissement**. Chaque étape contribue à garantir des données de qualité, prêtes pour des analyses avancées et des prises de décision stratégiques, le tout en s'appuyant sur la robustesse d'**HDFS** comme solution de stockage distribué.