# Hadoop & HDFS

# 1. Définition

**Hadoop** est un framework open-source conçu pour manipuler de très grands volumes de données (Big Data) de manière distribuée.
Il repose sur **3 piliers principaux :**

1. **HDFS** – Le système de fichiers distribué
2. **YARN** – Le gestionnaire de ressources et d'exécution
3. **MapReduce** – Le modèle de calcul parallèle *(moins utilisé aujourd'hui mais fondamental à comprendre)*

Hadoop permet de stocker **énormément de données** (plusieurs To/Petaoctets) et de les traiter **en parallèle** sur un cluster de machines. Il est donc**, tolérant aux pannes**, et un traitement **scalable**.

**HDFS (Haddoop Distibuted File System)** est un **système de fichiers distribué**, conçu pour stocker des fichiers **énormes** en les découpant et en les distribuant sur plusieurs machines d'un cluster. Il fonctionne selon l'idée suivante :
*Un fichier est découpé en blocs (par défaut 128 Mo).*
*Ces blocs sont copiés plusieurs fois (réplication) et dispersés sur plusieurs machines pour garantir la disponibilité et la tolérance aux pannes.*

## 1.1 .L' architecture HDFS

L'architecture repose sur deux types de nœuds :

### 1. NameNode

- Le chef du cluster HDFS
- Contient la **métadonnée**
    - Quels fichiers existent
    - Où sont stockés les blocs
    - Qui a accès à quoi
- Ne stocke **pas** les données, uniquement les informations.

👉 **C'est un point central : s'il tombe, le cluster devient inutilisable.**

### 2. DataNodes

- Machines qui stockent **les blocs de données** physiques
- Reçoivent les ordres du NameNode
- Renvoient des rapports réguliers de santé (Heartbeats)

👉 Ce sont les "disques durs" du cluster.

### Fonctionnement d'un stockage dans HDFS

1. On veut stocker un fichier
2. Hadoop coupe le fichier en **blocs** (ex : 128 Mo)
3. Chaque bloc est **mis sur différents DataNodes**
4. Le NameNode enregistre où se trouvent les blocs

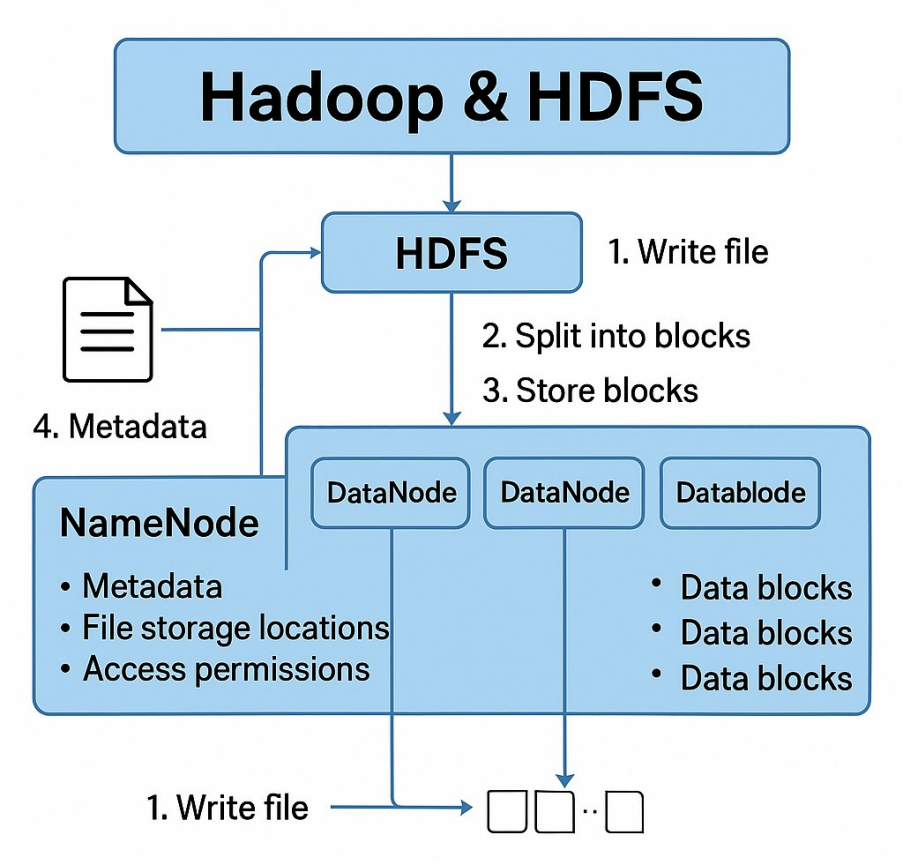
✔ **Réplication**
Par défaut, le facteur de réplication est **3**, pour éviter la perte de données.
Ex : Un fichier de 1 Go → ~8 blocs → chaque bloc a 3 copies → 24 blocs stockés au total.

---

## Les Avantages d'HDFS

✔ **Tolérance aux pannes**
Si un DataNode tombe, les blocs sont toujours disponibles ailleurs.
✔ **Scalabilité horizontale**
Tu ajoutes des machines → tu augmentes la capacité.
✔ **Haut débit**
HDFS est optimisé pour lire/écrire de très gros fichiers (GB/TB/PB).
✔ **Proximité des données**
Hadoop essaie de faire tourner les traitements **là où les données sont stockées** (« *data locality* »).
→ Cela évite de déplacer les fichiers, beaucoup plus lent.

---




## Limitations d'HDFS

- ❌ Pas prévu pour les petits fichiers (il préfère les gros fichiers)
- ❌ Pas fait pour les milliers de petites écritures aléatoires
- ❌ Un seul NameNode → point de fragilité (sauf en HA – High Availability)

## Commandes HDFS utiles (similaires à Linux)

| Commande | Fonction |
|---|---|
| hdfs dfs -ls / | Lister |
| hdfs dfs -put localfile /hdfs/path | Envoyer un fichier vers HDFS |
| hdfs dfs -get /hdfs/path localfile | Télécharger |
| hdfs dfs -mkdir /path | Créer un répertoire |
| hdfs dfs -rm -r /path | Supprimer |

# 1.2. Pourquoi utiliser docker ?

## 2.1. Installation beaucoup plus simple

Sans Docker :

- Il faut installer Java, Hadoop, configurer les fichiers XML, SSH, les chemins, etc.
- La moindre erreur de config = ça ne démarre pas.

Avec Docker :

- Tu télécharges une image déjà prête (namenode, datanode…).
- Un docker compose up → tu as un cluster HDFS fonctionnel.
- Parfait pour un TP ou un environnement de test.
-

## 2.2. Environnement identique pour tout le monde

En formation / promo :

- Sans Docker, chacun a sa propre config (Ubuntu, Windows, versions différentes…).
- Résultat : les bugs ne sont jamais les mêmes.

Avec Docker :

- Même image pour tout le monde → **même version de Hadoop, même config**.
- Le formateur sait que les commandes fonctionneront pareil sur toutes les machines.

## 2.3. Isolation de l'environnement

- Hadoop et HDFS ne touchent pas trop au système de fichiers "réel" de ta machine.
- Tout est "conteneurisé" → facile à supprimer si tu as tout cassé :

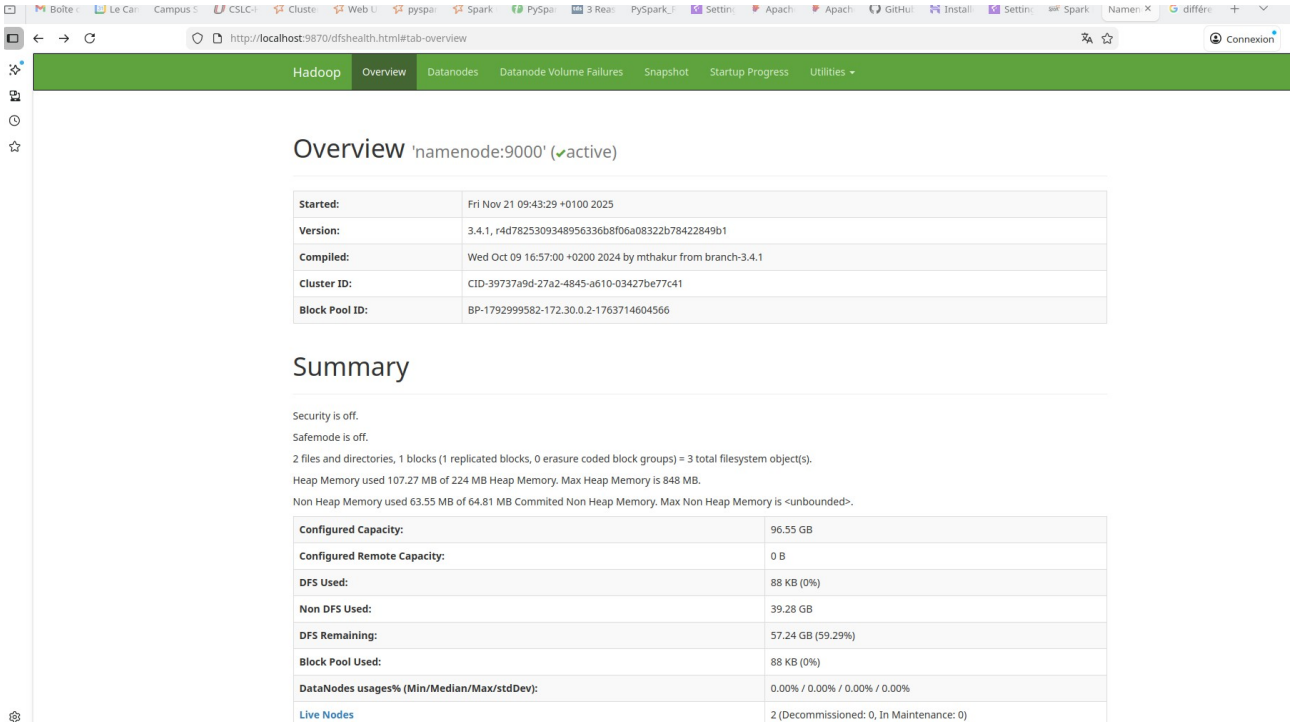
Comment installer docker
suivre la procédure de ce lien :
https://www.hostinger.com/fr/tutoriels/installer-docker-sur-ubuntu

## Conclusion :

- installation Docker,
- récupération l'image apache/hadoop:3.4.1,
- démarrage un mini-cluster HDFS (1 NameNode + 2 DataNodes) avec *docker compose up -d*,
- vérification :
    - la WebUI sur le port 9870 >

  *ssh -L 9870:192.168.4.171:9870 mvana@51.91.85.211*
  puis *http://localhost:9870*
    - les commandes hdfs dfs (ls, put, cat…).

```
mvana@promo-ds4-gra9-2:~/hadoop-cluster$ cd ~/hadoop-cluster
sudo docker compose up -d
WARN[0000] /home/mvana/hadoop-cluster/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 25/25
 ✔ namenode Pulled
   ✔ 8ba884070f61 Pull complete
   ✔ 93cf21fbbef2 Pull complete
   ✔ c778ba593b2f Pull complete
   ✔ 782cc61249e1 Pull complete
   ✔ 62371aa21ed7 Pull complete
   ✔ a99f0329f9e1 Pull complete
   ✔ a3de4f0aebed Pull complete
   ✔ 126875c5019e Pull complete
   ✔ 95cfbfe67739 Pull complete
   ✔ e8adfb95322d Pull complete
   ✔ 73bf0366d627 Pull complete
   ✔ 08af2d7d7ec2 Pull complete
   ✔ 086c6a12104a Pull complete
   ✔ 13a0fd9e312b Pull complete
   ✔ 34a986ee3c28 Pull complete
   ✔ b08b5eed2a97 Pull complete
   ✔ 36b23ba8b2ef Pull complete
   ✔ 6503fca807cb Pull complete
   ✔ 4f4fb700ef54 Pull complete
   ✔ cb57cff56e54 Pull complete
   ✔ 2f91676ace52 Pull complete
   ✔ 3da47e30e0b8 Pull complete
 ✔ datanode1 Pulled
 ✔ datanode2 Pulled
[+] Running 4/4
 ✔ Network hadoop-cluster_hdfs_network  Created
 ✔ Container namenode                   Started
 ✔ Container datanode2                  Started
 ✔ Container datanode1                  Started
```



# 2 — Les fichiers dans HDFS

Dans cette partie, j'ai utilisé l'outil en ligne de commande fourni par HDFS pour manipuler des fichiers dans le système distribué. Toutes les commandes ont été exécutées dans le conteneur *namenode* de mon cluster Docker.

## ◆ Affichage du contenu de la racine du HDFS

Je commence par afficher le contenu de la racine du système HDFS :
*hdfs dfs -ls*
À ce moment-là, la racine ne contient généralement aucun fichier, ce qui me permet de partir sur un espace propre.

## ◆ Création d'un dossier data_test

Je crée ensuite un dossier dans HDFS nommé **data_test**, qui me servira de répertoire de travail :
*hdfs dfs -mkdir /data_test*

Je vérifie immédiatement sa présence :
*hdfs dfs -ls*

```
mvana@promo-ds4-gra9-2:~/hadoop-cluster$ sudo docker exec -it namenode bash
[root@namenode hadoop]# hdfs dfs -ls /

2025-11-21 10:04:57,312 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:04:57,313 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:04:57,313 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:04:57,314 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:04:57,327 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:04:58,048 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolver, no QOP found in configuration
r.protection
Found 1 items
-rw-r--r--   2 root supergroup         13 2025-11-21 08:48 /test_hdfs.txt
2025-11-21 10:04:58,300 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]#
[root@namenode hadoop]# hdfs dfs -mkdir /data_test
2025-11-21 10:05:11,084 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:05:11,085 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:05:11,085 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:05:11,086 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:05:11,096 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0

2025-11-21 10:05:11,801 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolver, no QOP found in configuration
r.protection
2025-11-21 10:05:12,091 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - /data_test: masked={ masked: rwxr-xr-x, unmasked: rwxrwxrwx }
2025-11-21 10:05:12,122 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]#
[root@namenode hadoop]# hdfs dfs -ls /

2025-11-21 10:06:44,612 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:06:44,614 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:06:44,614 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:06:44,614 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:06:44,626 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:06:45,380 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolver, no QOP found in configuration
r.protection
Found 2 items
drwxr-xr-x   - root supergroup          0 2025-11-21 10:05 /data_test
-rw-r--r--   2 root supergroup         13 2025-11-21 08:48 /test_hdfs.txt
2025-11-21 10:06:45,661 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]#
```

## ◆ Lecture des manuels des commandes put et get

Avant de manipuler des fichiers, je prends le temps de consulter les pages d'aide des commandes essentielles :
*hdfs dfs -help put*
*hdfs dfs -help get*

Grâce à cette lecture, je comprends que :

- **put** me permet d'ajouter un fichier depuis l'hôte ou le conteneur vers HDFS.
- **get** me permet d'extraire un fichier depuis HDFS vers mon système local.

## ◆ Ajout du CSV des arbres de Paris

Après avoir copier le fichier arbreParis.csv dans namenode (fait manuellement dans VsCode)

```
[root@namenode hadoop]# hdfs dfs -help put
-put [-f] [-p] [-l] [-d] [-t <thread count>] [-q <thread pool queue size>] <localsrc> ... <dst> :
  Copy files from the local file system into fs. Copying fails if the file already
  exists, unless the -f flag is given.
  Flags:

  -p                            Preserves timestamps, ownership and the mode.
  -f                            Overwrites the destination if it already exists.
  -t <thread count>             Number of threads to be used, default is 1.
  -q <thread pool queue size>   Thread pool queue size to be used, default is
                                1024.
  -l                            Allow DataNode to lazily persist the file to disk.
                                Forces replication factor of 1. This flag will
                                result in reduced durability. Use with care.
  -d                            Skip creation of temporary file(<dst>._COPYING_).
[root@namenode hadoop]# hdfs dfs -help get
-get [-f] [-p] [-crc] [-ignoreCrc] [-t <thread count>] [-q <thread pool queue size>] <src> ... <localdst> :
  Copy files that match the file pattern <src> to the local name. <src> is kept.
  When copying multiple files, the destination must be a directory.
  Flags:

  -p                            Preserves timestamps, ownership and the mode.
  -f                            Overwrites the destination if it already exists.
  -crc                          write CRC checksums for the files downloaded.
  -ignoreCrc                    Skip CRC checks on the file(s) downloaded.
  -t <thread count>             Number of threads to be used, default is 1.
  -q <thread pool queue size>   Thread pool queue size to be used, default is
                                1024.
```

*ls -l /opt/hadoop/data/nameNode*

J'ajoute ensuite le fichier CSV contenant les données des arbres de Paris dans HDFS.
Je commence par copier le fichier dans le conteneur *namenode*, puis je l'envoie dans HDFS :

*hdfs dfs -put /opt/hadoop/data/nameNode/arbresParis.csv /data_test/*
*hdfs dfs -ls /data_test*

```
[root@namenode hadoop]#
[root@namenode hadoop]# hdfs dfs -ls /data_test
2025-11-21 10:20:29,278 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:20:29,280 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:20:29,280 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:20:29,280 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:20:29,292 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:20:30,037 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolver, no QOP found in configuration for dfs.data.transfe
r.protection
Found 1 items
```

*hdfs dfs -cat /data_test/arbresParis.csv | head*

```
[root@namenode hadoop]# hdfs dfs -cat /data_test/arbresParis.csv | head
2025-11-21 10:20:32,013 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:20:32,015 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:20:32,015 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:20:32,015 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:20:32,029 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:20:32,865 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolver, no QOP found in configuration for dfs.data.transfe
r.protection
2025-11-21 10:20:33,188 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Connecting to datanode 172.30.0.3:9866
2025-11-21 10:20:33,213 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient  - SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2025-11-21 10:20:33,227 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient  - SASL client skipping handshake in unsecured configuration for addr = /172.30.0.3, datanodeId = Datanode
InfoWithStorage[172.30.0.3:9866,DS-df43e90f-560c-42b7-a418-12e12737b0e9,DISK]
Geo point;idbase;domanialite;arrondissement;complement adresse;numero;adresse;circonference en cm;hauteur en m;stade développement;pépinière;genre;espèce;varieteoucultivar;date de plantation;libellé Français;ID Base;
ID arbre;Site;Adresse;Complément d'adresse;Arrondissement;Domanialité;Dénomination usuelle;Dénomination botanique;Autorité taxonomique;Année de plantation;Qualification remarquable;Résumé;Descriptif;Numéro de délibér
ation;Date de la délibération;Label national;Panonceau;Photo 1;Copyright 1
cat: Unable to write to output stream.
[root@namenode hadoop]#
```

## ◆ Ce que je constate

En examinant le fichier, je remarque plusieurs choses importantes :

1. **HDFS stocke le fichier, mais ne le modifie pas** : son contenu est identique à celui du fichier source.
2. **HDFS découpe le fichier en blocs internes**, mais cela n'est pas visible directement pour l'utilisateur : il conserve la même apparence logique.
3. **HDFS n'est pas un système de fichiers classique** : il ne permet pas d'ouvrir des fichiers lourds facilement avec des commandes POSIX comme less ou head -n.

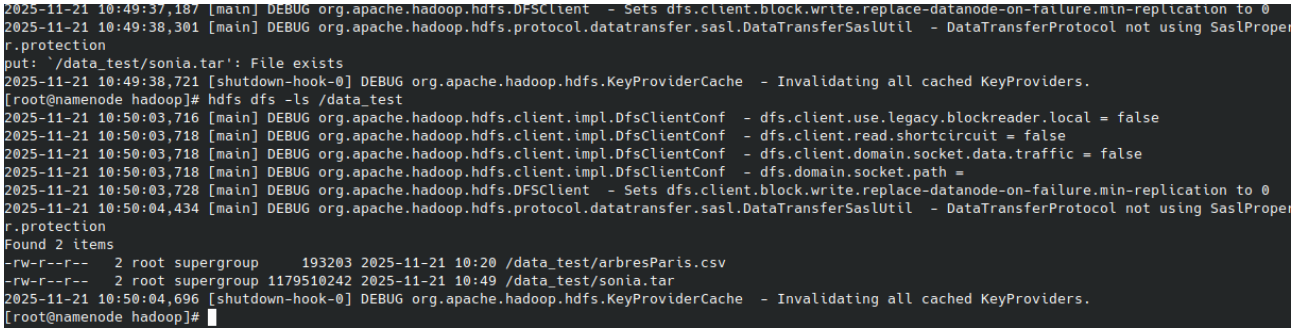
Après avoir ajouté le fichier CSV des arbres de Paris dans HDFS, je poursuis l'exercice en manipulant une archive contenant plusieurs livres. Dans mon cas, j'utilise le fichier sonia.tar.

## ◆ Ajout du fichier sonia.tar

Je commence par copier l'archive sonia.tar depuis mon l'endroit où il est situé sur le serveur vers le conteneur *namenode*.
*sudo docker cp ~/Module_BigData/sonia.tar namenode:/tmp/sonia.tar*

Une fois le fichier présent dans /tmp, je l'ajoute dans le système HDFS :
*sudo docker exec -it namenode bash*
*hdfs dfs -put /tmp/sonia.tar /data_test/*

(cela a pris un petit moment pour copier… pas mal de lignes ont défilé)
on regarde si le fichier est bien présent dans HDFS :
*hdfs dfs -ls /data_test*




## ◆ Ce que je constate en ajoutant l'archive

En ajoutant l'archive dans HDFS, je constate que :

- HDFS accepte parfaitement le fichier sonia.tar, même s'il s'agit d'un fichier volumineux.
- Le fichier apparaît dans le dossier /data_test, comme n'importe quel autre fichier.
- En revanche, je ne peux **pas** consulter son contenu depuis HDFS.
  Il est impossible d'extraire l'archive directement avec des commandes comme tar -x, car HDFS n'est pas un système de fichiers classique.

En résumé :
➡ **HDFS stocke l'archive, mais ne peut pas l'ouvrir ni la manipuler.**
➡ **Le fichier est traité comme un simple binaire.**

## ◆ Suppression de l'archive

Pour continuer l'exercice, je supprime le fichier sonia.tar du dossier /data_test :
*hdfs dfs -rm /data_test/sonia.tar*

hdfs dfs -ls /data_test

```
[root@namenode hadoop]# hdfs dfs -rm /data_test/sonia.tar
2025-11-21 10:55:42,619 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:55:42,620 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:55:42,621 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:55:42,621 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:55:42,633 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:55:43,342 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPro
r.protection
Deleted /data_test/sonia.tar
2025-11-21 10:55:43,703 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]# hdfs dfs -ls /data_test
2025-11-21 10:55:50,686 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 10:55:50,687 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 10:55:50,687 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 10:55:50,687 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 10:55:50,700 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 10:55:51,503 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPro
r.protection
Found 1 items
-rw-r--r--   2 root supergroup     193203 2025-11-21 10:20 /data_test/arbresParis.csv
2025-11-21 10:55:51,789 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]#
```

## 🔹 Ajout de tous les livres dans HDFS (mode récursif)

Les **modes récursifs** d'une commande HDFS (ou d'une commande Linux en général) permettent de **traiter un dossier et *tout* ce qu'il contient**, c'est-à-dire :

- les fichiers dans ce dossier,
- les sous-dossiers,
- les fichiers dans les sous-dossiers,
- etc.

👉 **Récursif = on descend dans toutes les couches d'un dossier automatiquement.**

Dans HDFS, ce comportement est utilisé principalement avec les commandes **put**, **get**, **mkdir**, **ls**, et **rm**.

Le mode récursif (-R) permet d'appliquer une commande HDFS à un dossier et à l'ensemble de son contenu (fichiers + sous-dossiers). Je l'ai utilisé pour envoyer en une seule commande tous les livres dans HDFS, puisque le dossier contenait de nombreux fichiers. Sans l'option -R, la commande put ne peut pas traiter un dossier.

Exemple sur le dossier books_split

remettre le fichier sonia.tar dans HDFS et autre exemple avec books_split/

***mvana@promo-ds4-gra9-2:~/Module_BigData$*** *sudo docker cp  books_split namenode:/tmp/books_split*

## 🔹 Mode récursif avec put

La commande :

*hdfs dfs -put /tmp/sonia.tar /data_test/*

signifie :

*J'ajoute le dossier soniar.tar dans HDFS, et j'ajoute aussi tous les fichiers et sous-dossiers qu'il contient.*

HDFS va envoyer dans /data_test/ :

- tous les fichiers du dossier sonia.tar/
- tous les sous-dossiers (s'il y en a)
- et tous les fichiers dans ces sous-dossiers

Sans -R, la commande échoue si tu lui donnes un dossier.

NB : Je ne peux pas utiliser l'option -R (récursive) avec un simple fichier comme sonia.tar, car cette option est destinée aux répertoires. Avec un fichier, j'utilise simplement hdfs dfs -put, et je réserve -R pour l'ajout d'un dossier complet contenant plusieurs fichiers et sous-dossiers.

```
[root@namenode hadoop]# hdfs dfs -ls -R /data_test
2025-11-21 13:02:33,445 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 13:02:33,446 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 13:02:33,447 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 13:02:33,447 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 13:02:33,457 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 13:02:34,239 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPro
r.protection
-rw-r--r--   2 root supergroup     193203 2025-11-21 10:20 /data_test/arbresParis.csv
2025-11-21 13:02:34,525 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]# hdfs dfs -put -R /tmp/sonia.tar /data_test/^C
[root@namenode hadoop]# hdfs dfs -put /tmp/sonia.tar /data_test/
```

Pour ajouter l'ensemble des fichiers du dossier books_split dans HDFS, j'ai d'abord copié ce dossier dans le conteneur namenode, puis j'ai utilisé la commande
*hdfs dfs -put -R /tmp/books_split/ /data_test/*.
Sur la version de Hadoop que j'utilise, l'option -R n'est pas disponible pour la commande put, ce qui m'a d'abord donné une erreur (Illegal option -R).
J'ai ensuite constaté que, lorsqu'on passe un dossier en paramètre à put, HDFS ajoute automatiquement tous les fichiers qu'il contient, sans qu'il soit nécessaire d'ajouter une option récursive.

## ◆ Mode récursif avec ls

Tu peux lister **toute la structure** d'un dossier :
*hdfs dfs -ls /data_test*
*hdfs dfs -ls /data_test/books_split | head*
Ça affiche :

- les sous-répertoires,
- les fichiers,
- et parfois les permissions.

```
2025-11-21 13:22:53,917 [main] DEBUG org.apache.hadoop.hdfs.DFSOutputStream  - Closing an already closed stream. [Stream:true, streamer:true]
2025-11-21 13:22:53,933 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]# hdfs dfs -ls /data_test
2025-11-21 13:24:52,824 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 13:24:52,827 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 13:24:52,827 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 13:24:52,828 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 13:24:52,848 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 13:24:53,809 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolve
r.protection
Found 2 items
-rw-r--r--   2 root supergroup     193203 2025-11-21 10:20 /data_test/arbresParis.csv
drwxr-xr-x   - root supergroup          0 2025-11-21 13:22 /data_test/books_split
2025-11-21 13:24:54,155 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]# hdfs dfs -ls /data_test/books_split | head
2025-11-21 13:24:55,895 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 13:24:55,897 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 13:24:55,897 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 13:24:55,898 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 13:24:55,910 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 13:24:56,568 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolve
r.protection
Found 710 items
-rw-r--r--   2 root supergroup        125 2025-11-21 13:21 /data_test/books_split/book_0000.txt
-rw-r--r--   2 root supergroup    2118455 2025-11-21 13:21 /data_test/books_split/book_0001.txt
-rw-r--r--   2 root supergroup     391079 2025-11-21 13:22 /data_test/books_split/book_0002.txt
[root@namenode hadoop]#
```

## ◆ Mode récursif avec rm

La commande  rm ou rmdir
exemple *hdfs dfs -rm -R /data_test/books_split/*

```
Deleted /data_test/books_split
2025-11-21 13:27:58,032 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]# hdfs dfs -ls /data_test
2025-11-21 13:28:09,316 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.use.legacy.blockreader.local = false
2025-11-21 13:28:09,319 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.read.shortcircuit = false
2025-11-21 13:28:09,319 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.client.domain.socket.data.traffic = false
2025-11-21 13:28:09,319 [main] DEBUG org.apache.hadoop.hdfs.client.impl.DfsClientConf  - dfs.domain.socket.path =
2025-11-21 13:28:09,332 [main] DEBUG org.apache.hadoop.hdfs.DFSClient  - Sets dfs.client.block.write.replace-datanode-on-failure.min-replication to 0
2025-11-21 13:28:10,090 [main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil  - DataTransferProtocol not using SaslPropertiesResolve
r.protection
Found 1 items
-rw-r--r--   2 root supergroup     193203 2025-11-21 10:20 /data_test/arbresParis.csv
2025-11-21 13:28:10,376 [shutdown-hook-0] DEBUG org.apache.hadoop.hdfs.KeyProviderCache  - Invalidating all cached KeyProviders.
[root@namenode hadoop]#
```

# 3 — Le fameux job sur les textes (version adaptée : arbres de Paris)

Dans cette partie, j'ai réalisé un job Spark, mais au lieu d'utiliser des textes, j'ai choisi de travailler sur le fichier arbresParis.csv, qui contient les données sur les arbres de Paris.
Comme l'image Docker namenode ne contient pas Spark, j'ai exécuté le job Spark **depuis ma machine virtuelle**, où Spark était déjà installé.
Ensuite, j'ai transféré les résultats dans HDFS.

### 1. Téléversement des données dans HDFS

Le fichier arbresParis.csv était déjà présent dans HDFS sous le chemin /data_test/arbresParis.csv

### 2. Adaptation du job Spark

J'ai écrit un script Python job_arbres.py pour :

- lire le fichier,
- calculer le nombre d'arbres par arrondissement,
- produire un jeu de résultats intermédiaires,
- puis un résultat final (top 5 des arrondissements ayant le plus d'arbres),
- enregistrer ces résultats dans des dossiers locaux.
-

Exécution :
*/home/mvana/bigdata/bin/python /home/mvana/Module_BigData/job_arbres.py*
le job a été exécuté avec succès et produit dans les dossiers :
*resultats/intermediaire_nb_arbres/*
*resultats/final_top5/*

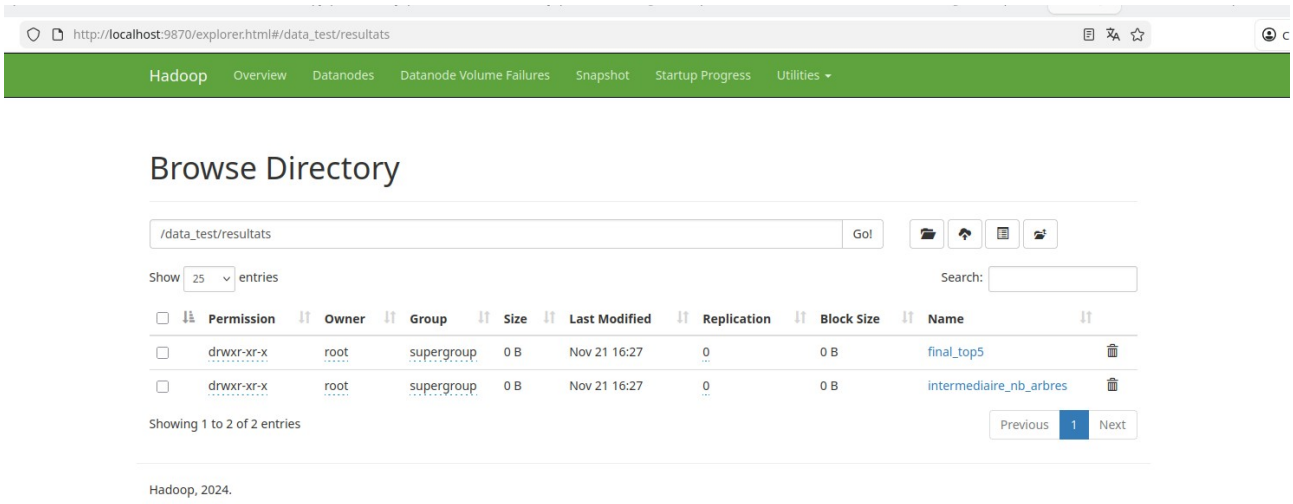
### 3. Transfert des résultats dans HDFS

J'ai commencé par copier le dossier résultats dans le conteneur namenode :
*sudo docker cp /home/mvana/Module_BigData/resultats namenode:/tmp/resultats*

Ensuite, à l'intérieur du conteneur, j'ai créé un dossier cible dans HDFS :
*hdfs dfs -mkdir -p /data_test/resultats*

Et j'ai importé les fichiers :
*hdfs dfs -put /tmp/resultats/intermediaire_nb_arbres /data_test/resultats/*
*hdfs dfs -put /tmp/resultats/final_top5 /data_test/resultats/*

J’ai vérifié la présence des fichiers :
*hdfs dfs -ls -R /data_test/resultats*
Les résultats sont bien présents dans HDFS (cf




## 3 — Clustering

Pour cette dernière partie, j’ai choisi d’appliquer une méthode de clustering non pas sur des textes, mais sur le jeu de données des arbres de Paris (arbresParis.csv). L’objectif que je me suis fixé est de regrouper les arbres en fonction de caractéristiques simples : l’arrondissement, la circonférence du tronc et la hauteur.
Je commence par lancer mon environnement Python dédié au big data,
ssh mvana@51.91.85.211
cd ~/Module_BigData
source ~/bigdata/bin/activate
puis j’exécute mon script Spark :
/home/mvana/bigdata/bin/python job_clustering_arbres.py

Dans ce script, j’instancie d’abord une SparkSession. Ensuite, je charge le fichier CSV :

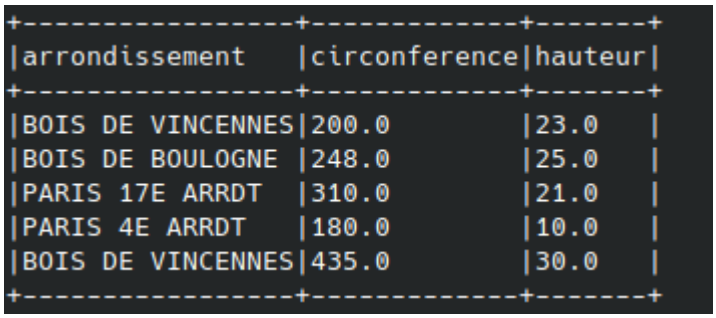
```
df_raw = (
    spark.read
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv")
)
```

Spark m’indique qu’il y a 186 lignes dans le fichier. Je sélectionne ensuite uniquement les colonnes dont j’ai besoin pour le clustering : l’arrondissement, la circonférence en centimètres et la hauteur en mètres. Je profite de cette étape pour nettoyer un peu les données.
Je commence par renommer les colonnes et les mettre dans un format plus pratique :

```
df = df_raw.select(
    F.trim(F.col("arrondissement3")).alias("arrondissement"),
```

```
    F.col("circonference en cm").cast("double").alias("circonference"),
    F.col("hauteur en m").cast("double").alias("hauteur")
)
```

Je filtre ensuite les lignes problématiques : je retire les arrondissements vides et les lignes où la circonférence ou la hauteur sont nulles. Après ce filtrage, Spark m'indique qu'il reste 185 lignes "propres". J'affiche un extrait pour vérifier le résultat :



```
+-----------------+-------------+-------+
|arrondissement   |circonference|hauteur|
+-----------------+-------------+-------+
|BOIS DE VINCENNES|200.0        |23.0   |
|BOIS DE BOULOGNE |248.0        |25.0   |
|PARIS 17E ARRDT  |310.0        |21.0   |
|PARIS 4E ARRDT   |180.0        |10.0   |
|BOIS DE VINCENNES|435.0        |30.0   |
+-----------------+-------------+-------+
```

À ce stade, je dois préparer les données pour les donner à un algorithme de clustering de Spark MLlib, en l'occurrence KMeans. L'arrondissement est une variable catégorielle (texte comme "PARIS 17E ARRDT" ou "BOIS DE VINCENNES"), alors que la circonférence et la hauteur sont numériques. Je commence donc par transformer l'arrondissement en indice numérique avec StringIndexer :

```
indexer = StringIndexer(
    inputCol="arrondissement",
    outputCol="arrondissement_index"
)
df_indexed = indexer.fit(df).transform(df)
```

Ensuite, j'assemble mes trois variables explicatives dans un seul vecteur de features grâce à VectorAssembler :

```
assembler = VectorAssembler(
    inputCols=["circonference", "hauteur", "arrondissement_index"],
    outputCol="features"
)
data = assembler.transform(df_indexed).select("arrondissement", "circonference", "hauteur", "features")
```

Une fois les features prêtes, je configure et j'entraîne mon modèle de clustering KMeans. Je choisis par exemple 4 clusters (k = 4) de manière empirique, ce qui permet d'avoir un découpage pas trop fin ni trop grossier :

```
kmeans = KMeans(k=4, seed=42, featuresCol="features", predictionCol="cluster")
model = kmeans.fit(data)
result = model.transform(data)
```

Je demande ensuite à Spark d'afficher quelques lignes pour vérifier que chaque arbre a bien été affecté à un cluster.
On voit par exemple que certains arbres de bois (Vincennes, Boulogne) se retrouvent dans le même cluster, tandis que d'autres arrondissements parisiens se regroupent selon des profils de circonférence et de hauteur.

```
Fichier  Édition  Affichage  Signets  Modules externes  Configuration  Aide
+-----------------+-------------+-------+
|arrondissement   |circonference|hauteur|
+-----------------+-------------+-------+
|BOIS DE VINCENNES|200.0        |23.0   |
|BOIS DE BOULOGNE |248.0        |25.0   |
|PARIS 17E ARRDT  |310.0        |21.0   |
|PARIS 4E ARRDT   |180.0        |10.0   |
|BOIS DE VINCENNES|435.0        |30.0   |
+-----------------+-------------+-------+
only showing top 5 rows
Lance le KMeans avec k=4 ...
25/11/24 12:33:51 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
25/11/24 12:33:52 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
25/11/24 12:33:52 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
25/11/24 12:33:53 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
25/11/24 12:33:54 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
Centres des clusters (circonference, hauteur) :
  Cluster 0 : [136.74358974  11.33333333]
  Cluster 1 : [382.89393939  24.92424242]
  Cluster 2 : [563.59259259  29.85185185]
  Cluster 3 : [233.9245283   16.33962264]
Quelques lignes avec le cluster assigné :
25/11/24 12:33:55 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
+-----------------+-------------+-------+-------+
|arrondissement   |circonference|hauteur|cluster|
+-----------------+-------------+-------+-------+
|BOIS DE VINCENNES|200.0        |23.0   |3      |
|BOIS DE BOULOGNE |248.0        |25.0   |3      |
|PARIS 17E ARRDT  |310.0        |21.0   |1      |
|PARIS 4E ARRDT   |180.0        |10.0   |0      |
|BOIS DE VINCENNES|435.0        |30.0   |1      |
|PARIS 8E ARRDT   |437.0        |28.0   |1      |
|PARIS 14E ARRDT  |140.0        |5.0    |0      |
|BOIS DE BOULOGNE |250.0        |20.0   |3      |
|PARIS 5E ARRDT   |415.0        |24.0   |1      |
|PARIS 7E ARRDT   |483.0        |25.0   |2      |
+-----------------+-------------+-------+-------+
only showing top 10 rows
25/11/24 12:33:55 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: arrondissement, circonference en cm, hauteur en m
 Schema: arrondissement3, circonference en cm, hauteur en m
Expected: arrondissement3 but found: arrondissement
CSV file: file:///home/mvana/Module_BigData/Iteration2/arbresParis.csv
Résultats sauvegardés dans : /home/mvana/Module_BigData/resultats_clustering_arbres
```

Une fois satisfait du résultat, je décide de sauvegarder ce DataFrame de sortie sur le disque local, dans un répertoire dédié :

```
output_path = "/home/mvana/Module_BigData/resultats_clustering_arbres"
result.select("arrondissement", "circonference", "hauteur", "cluster") \
    .coalesce(1) \
    .write.mode("overwrite") \
```

*.option("header", "true") *
*.csv(output_path)*

Spark confirme l'écriture, et le répertoire contient un fichier `part-00000-...csv` avec la liste des arbres et leur cluster associé.
Comme l'exercice me demande de stocker les résultats dans HDFS, je dois ensuite les envoyer dans le cluster Hadoop qui tourne dans Docker. Pour cela, je commence par copier le dossier de résultats depuis la machine hôte vers le conteneur du NameNode :
Spark confirme l'écriture, et le répertoire contient un fichier `part-00000-...csv` avec la liste des arbres et leur cluster associé

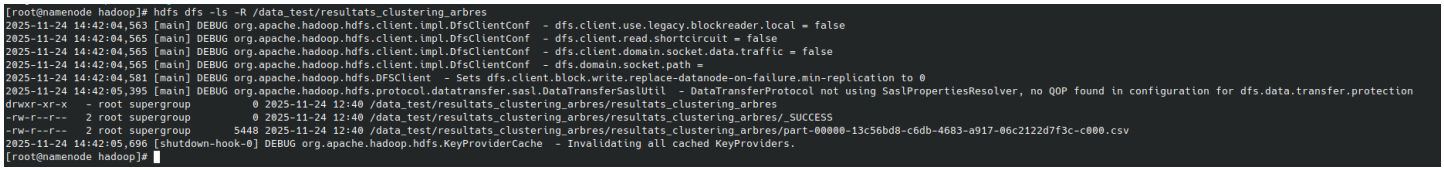
Comme l'exercice me demande de stocker les résultats dans HDFS, je dois ensuite les envoyer dans le cluster Hadoop qui tourne dans Docker. Pour cela, je commence par copier le dossier de résultats depuis la machine hôte vers le conteneur du NameNode :
*sudo docker cp /home/mvana/Module_BigData/resultats_clustering_arbres *
*namenode:/tmp/resultats_clustering_arbres*

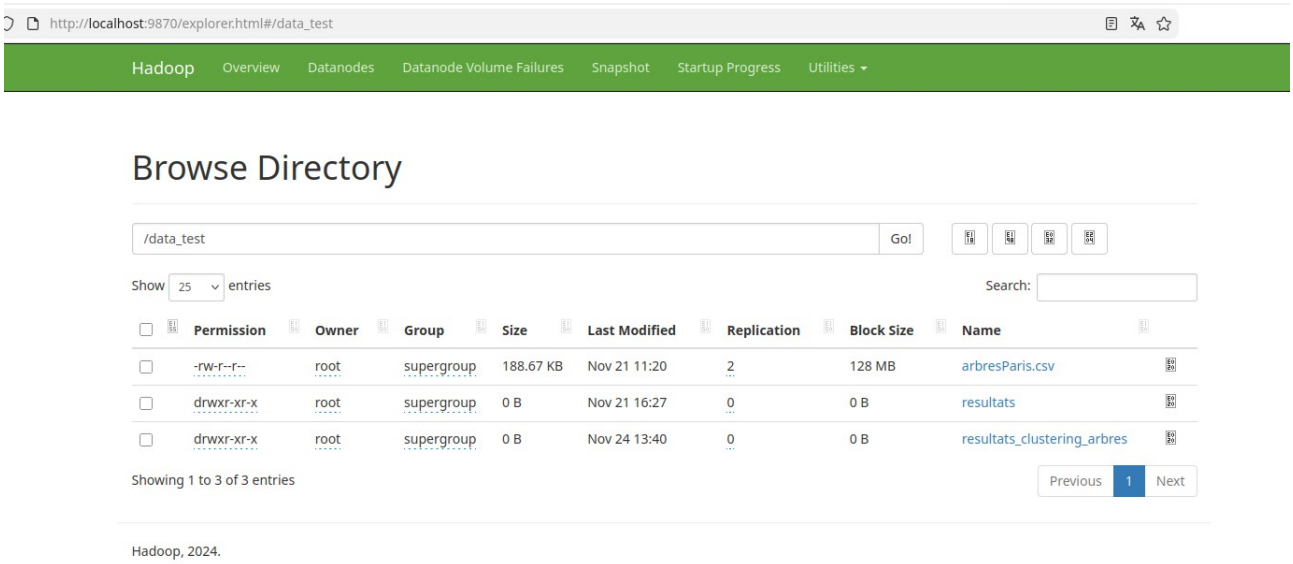
Une fois dans le conteneur `namenode`, je bascule dans un shell :
*sudo docker exec -it namenode bash*

À l'intérieur, je veux créer un répertoire HDFS pour accueillir ces résultats. Au début, le NameNode était en safe mode, ce qui m'empêchait de créer des répertoires ou de faire des `put`. J'ai donc commencé par désactiver le safe mode :
*hdfs dfsadmin -safemode leave*

Après ça, je crée le dossier cible dans HDFS :
*hdfs dfs -mkdir -p /data_test/resultats_clustering_arbres*
Puis je copie dans HDFS le dossier qui se trouve dans `/tmp` du conteneur :
*hdfs dfs -put /tmp/resultats_clustering_arbres /data_test/resultats_clustering_arbres*
Enfin, je vérifie que tout est bien présent dans HDFS :
*hdfs dfs -ls -R /data_test/resultats_clustering_arbres*

Je vois bien le répertoire et les fichiers `part-00000-...` ainsi que le fichier `_SUCCESS`.






Conclusion => J’ai ainsi pu dans cette partie :

- Préparé les données des arbres (sélection, nettoyage, typage).
- Appliqué un clustering KMeans avec Spark MLlib.
- Sauvegardé les résultats localement.
- Puis transféré et stocké ces résultats dans HDFS via mon cluster Hadoop dans Docker.

Cette partie me montre concrètement comment enchaîner prétraitement, modèle de machine learning et persistance des résultats dans un environnement big data combinant Spark et HDFS.