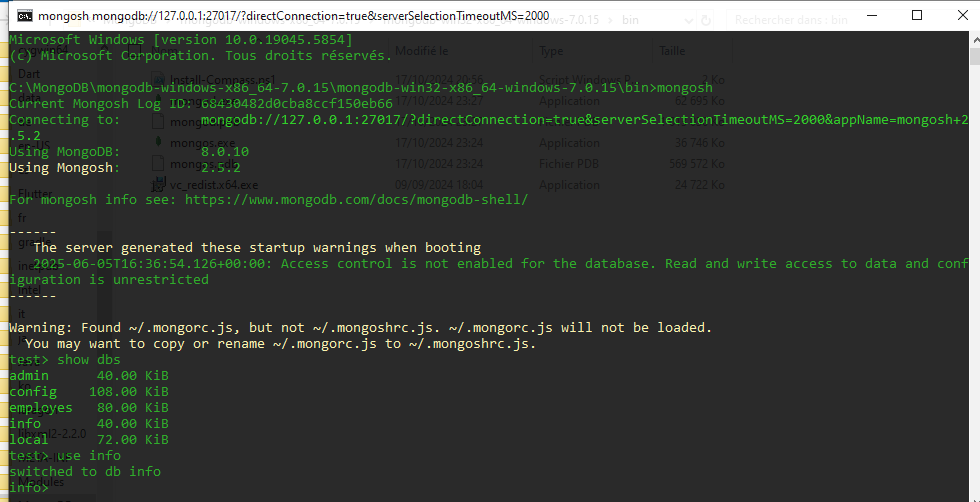
AichetouMoctar-C20136

TP1 MONGDB

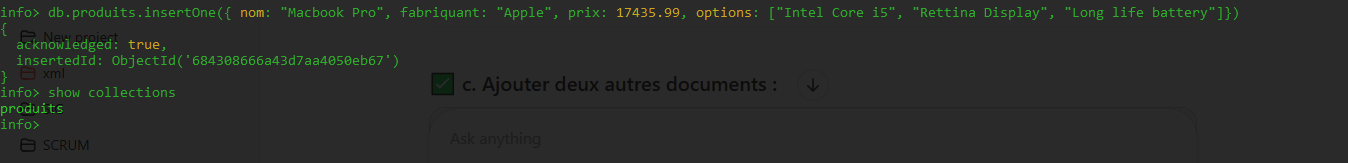
**Exercice 01**

Création d’une BD Mongo et opération CRUD

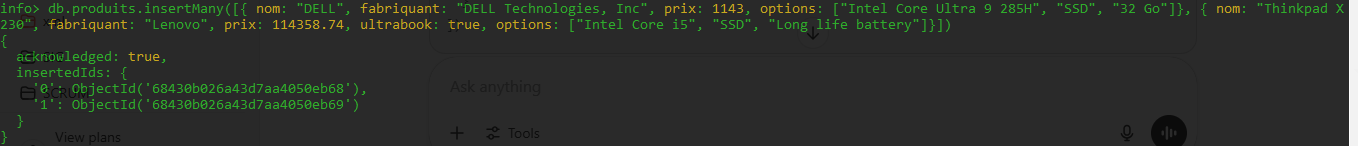
a. Créer une base de données nommée info et vérifiez qu'elle est sélectionnée



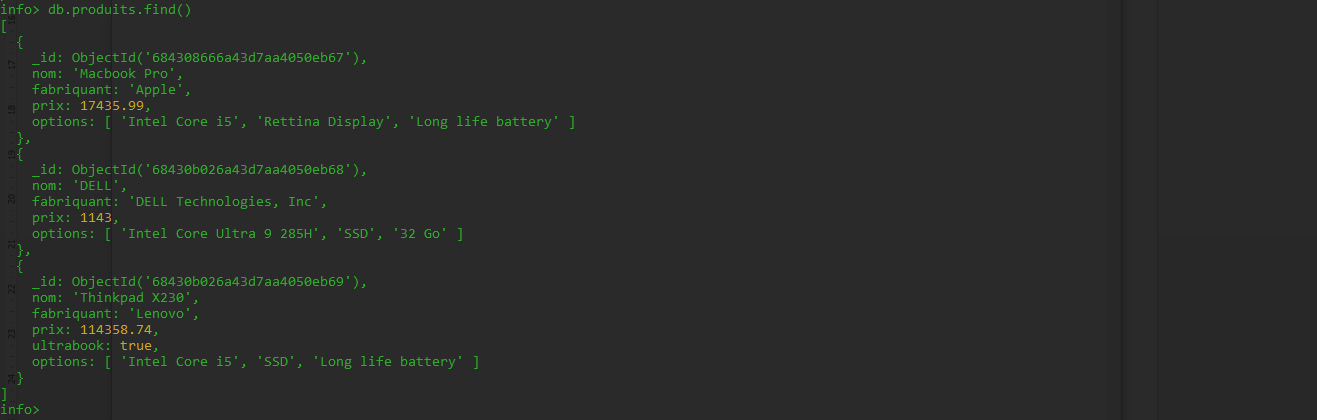
b. Créer une nouvelle collection nommée produits et y insérer le document suivant



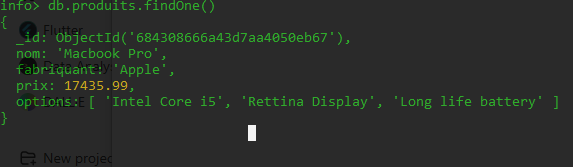
c. Rajouter deux autres documents dans produits

d. Effectuez les requêtes de lecture suivantes:

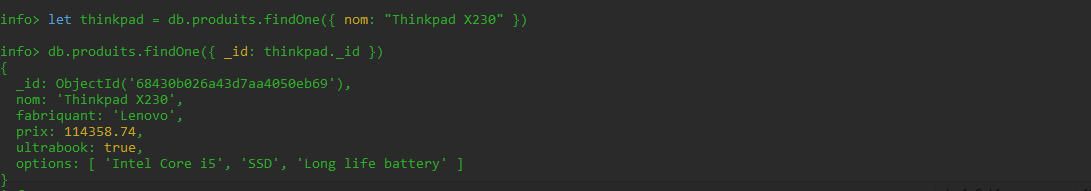
- Récupérer tous les produits.



- Récupérer le premier produit



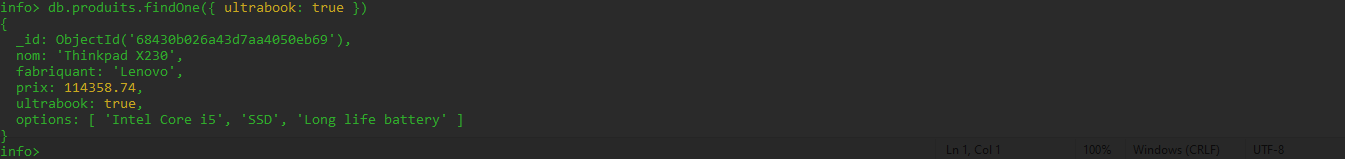
- Trouver l’id du Thinkpad et faites la requête pour récupérer ce produit avec son id.



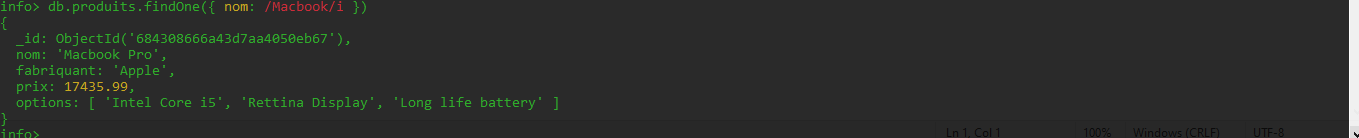
- Récupérer les produits dont le prix est supérieur à 13723 DA



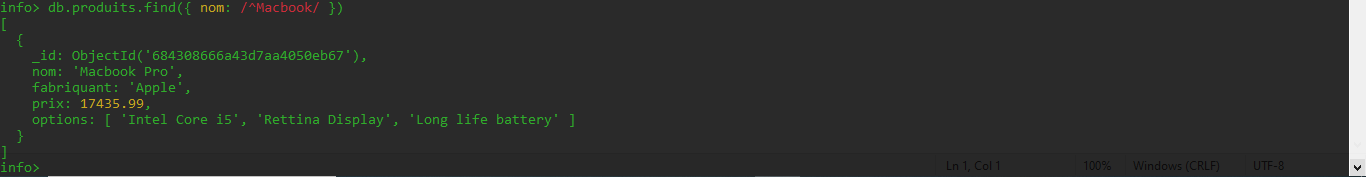
- Récupérer le premier produit ayant le champ ultrabook à true



- Récupérer le premier produit dont le nom contient Macbook



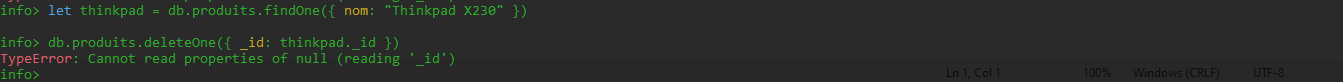
- Récupérer les produits dont le nom commence par Macbook



- Supprimer le produit dont le fabricant est Lenovo.



- Supprimer le Lenovo X230 en utilisant uniquement son id.



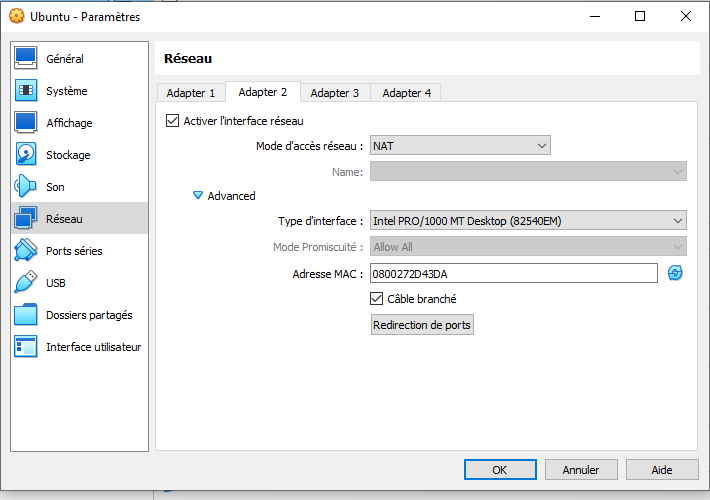
**Rapport – Mise en place de la plateforme Hadoop et Exécution d’un programme MapReduce**

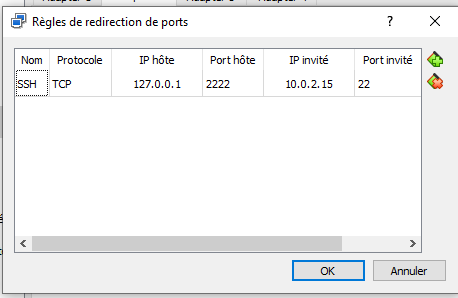
Partie 1 : Préparation de la plateforme de développement

**Étape 1 : Préparer la machine virtuelle Ubuntu**

Utilisation de VirtualBox pour lancer une machine Ubuntu, avec réseau configuré en NAT + redirection de port pour permettre SSH depuis la machine hôte.

* Installation de **VirtualBox**.
* Importation d’une VM Ubuntu (ou installation manuelle).
* Configuration du réseau en **mode NAT** avec **redirection de port** :
  + Port hôte : 2222 ➝ Port invité : 22 (SSH)
  + Adresse : 127.0.0.1
  + Adresse de machine virtuelle: 10.0.2.15



****

**Étape 2 : Activer SSH dans la VM**

Permet d’accéder à la VM depuis un terminal externe ou PuTTY. On installe openssh-server puis on démarre le service SSH.

**Dans la machine virtuelle :**

bash

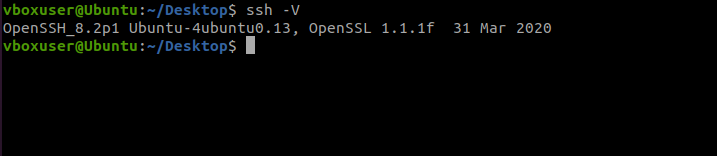
CopyEdit

sudo apt update

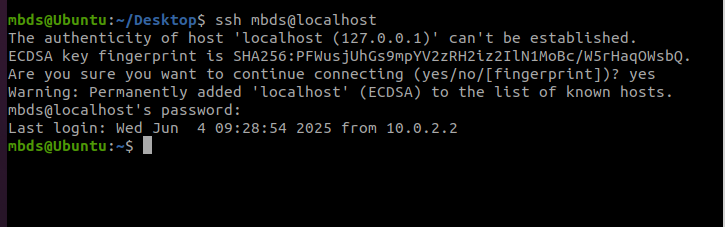
sudo apt install openssh-server -y

sudo systemctl enable ssh

sudo systemctl start ssh

**Test de connexion SSH local :**

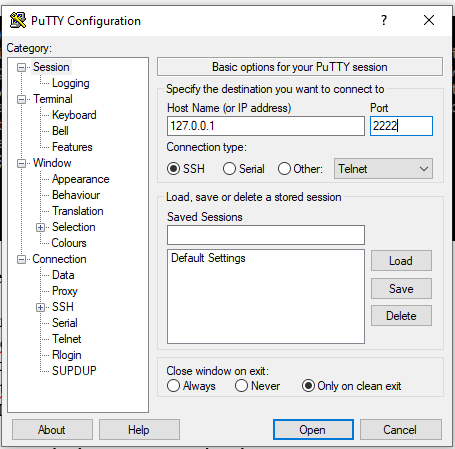
ssh mbds@localhost

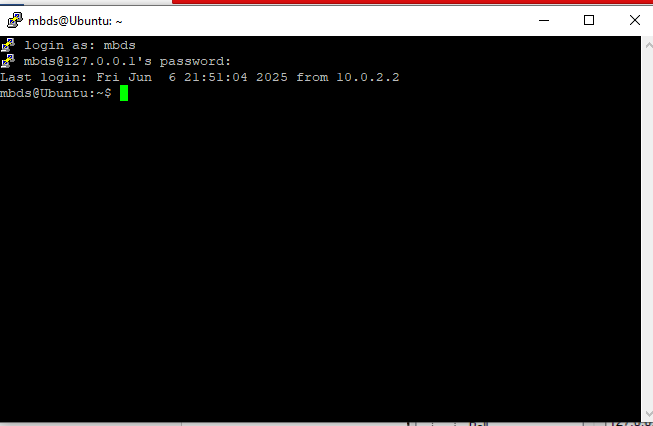


#### ****Étape 3 : Connexion SSH depuis la machine hôte****

#### PuTTY est un client SSH sous Windows. On se connecte à la VM via localhost sur le port 2222.

* Utilisation de **PuTTY** :
  + Hôte : localhost
  + Port : 2222
  + Utilisateur : mbds
  + Mot de passe : password





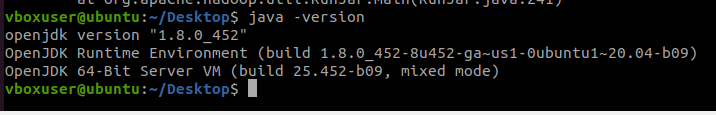
**Étape 4 : Installer Java (prérequis pour Hadoop)**

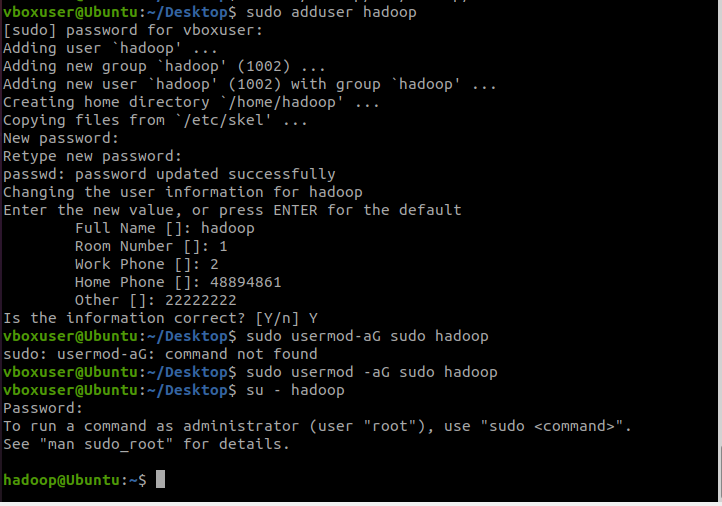
Java est indispensable pour Hadoop. On installe Java 8 avec apt et on vérifie avec java -version.

sudo apt update

sudo apt install openjdk-8-jdk -y

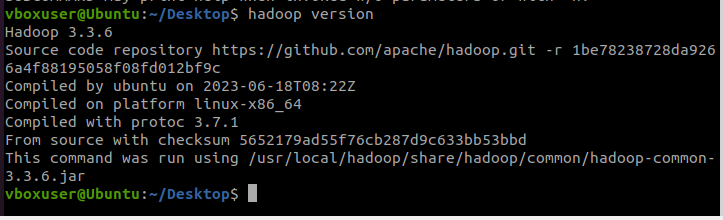
java –version

Créer un utilisateur hadoop (optionnel mais recommandé:

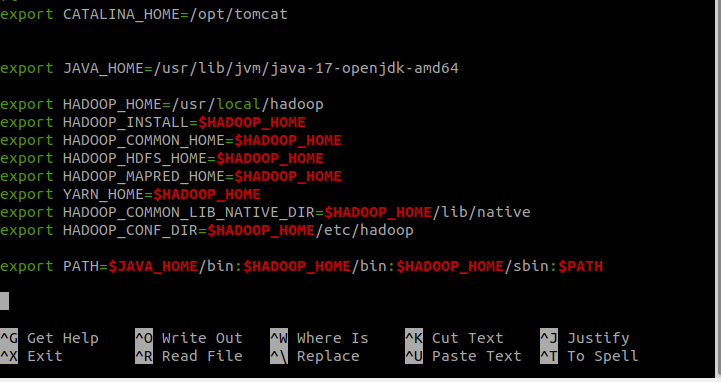
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

tar -xvzf hadoop-3.3.6.tar.gz

mv hadoop-3.3.6 ~/hadoop

 *Ajoute ceci dans ~/.bashrc (ou ~/.profile)***:**

nano ~/.bashrc



**Puis recharge le profil :**

source ~/.bashrc

#### ****Étape 6 : Configurer Hadoop (mode pseudo-distribué)****

Dans ~/hadoop/etc/hadoop/, éditer :

core-site.xml :

*core-site.xml*

*<configuration>*

*<property>*

*<name>fs.defaultFS</name>*

*<value>hdfs://localhost:9000</value>*

*</property>*

*</configuration>*

*hdfs-site.xml*

*<configuration>*

*<property>*

*<name>dfs.replication</name>*

*<value>1l</value>*

*</property>*

*</configuration>*

*mapred-site.xml*

*<configuration>*

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*

*</configuration>*

*yarn-site.xml*

*<configuration>*

*<property>*

*<name>yarn.nodemanager.aux-services</name>*

*<value>mapreduce\_shuffle</value>*

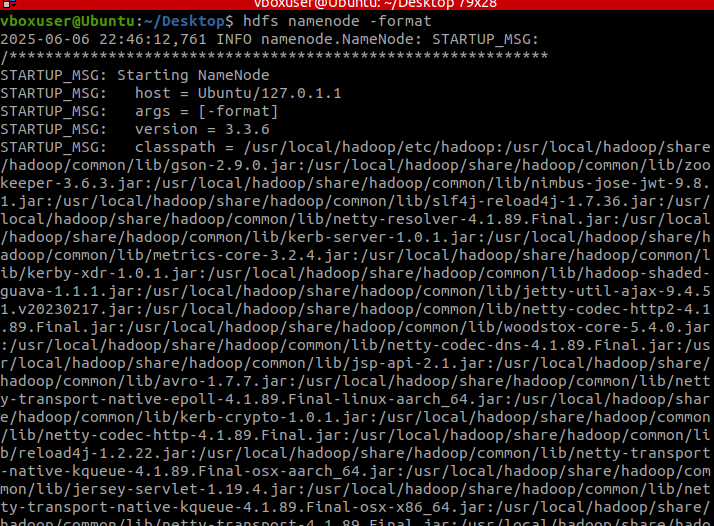
*</property>*

*</configuration>*

**Étape 7 : Formater le système de fichiers Hadoop (HDFS)**

On initialise le système de fichiers Hadoop avec hdfs namenode -format. À faire une seule fois.

hdfs namenode –format

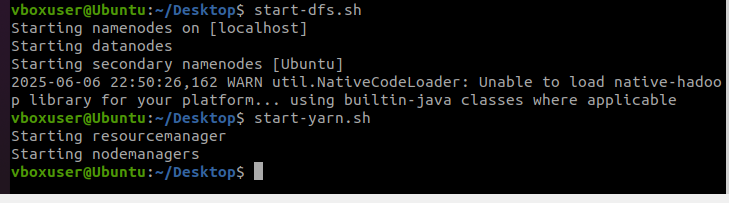


**Étape 8** : Démarrer Hadoop

Lancement des services Hadoop HDFS et YARN avec start-dfs.sh et start-yarn.sh.

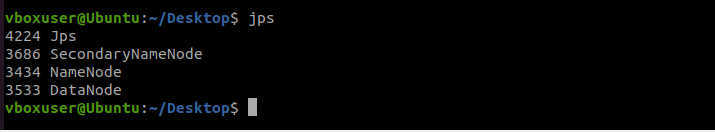
start-dfs.sh

start-yarn.sh



**Étape 9** : Tester Hadoop

**Jps**



**Teste HDFS :**

Après avoir démarré Hadoop avec start-dfs.sh et start-yarn.sh, nous avons vérifié que le système de fichiers HDFS est opérationnel.  
La commande suivante a été exécutée pour créer un répertoire :

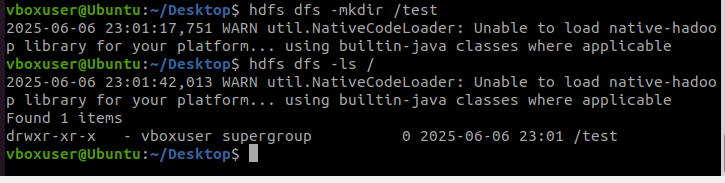
hdfs dfs -mkdir /test

Ensuite, nous avons confirmé sa présence avec :

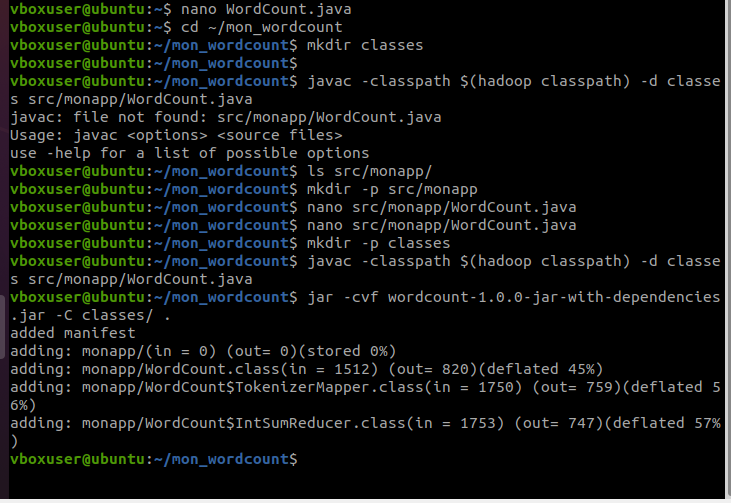
hdfs dfs -ls /

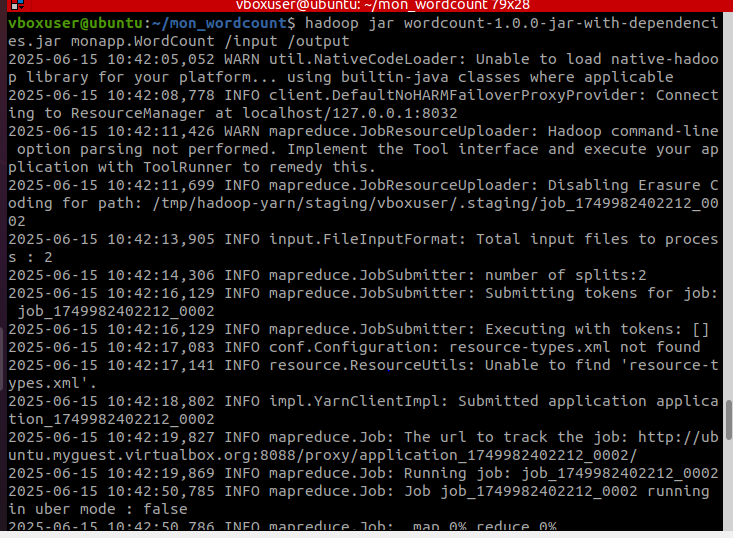
Ce qui a donné l’affichage suivant :

drwxr-xr-x - vboxuser supergroup 0 2025-06-06 23:01 /test

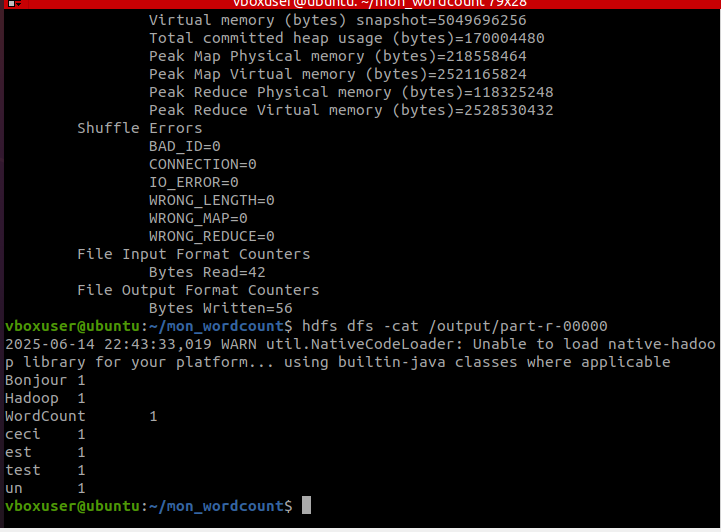


Cela montre que Hadoop HDFS fonctionne correctement malgré le message d’avertissement NativeCodeLoader, qui peut être ignoré.





Programme Mapreduce qui s'appelle wordcount (comptage des mots ), apartir du fichier .jar que tu as compilé, sur des fichiers situés dans le dossiers /input et qui vont produire des resultats dans /output.



Nous a permis de consulter les resultats finaux du traitement MapReduce.

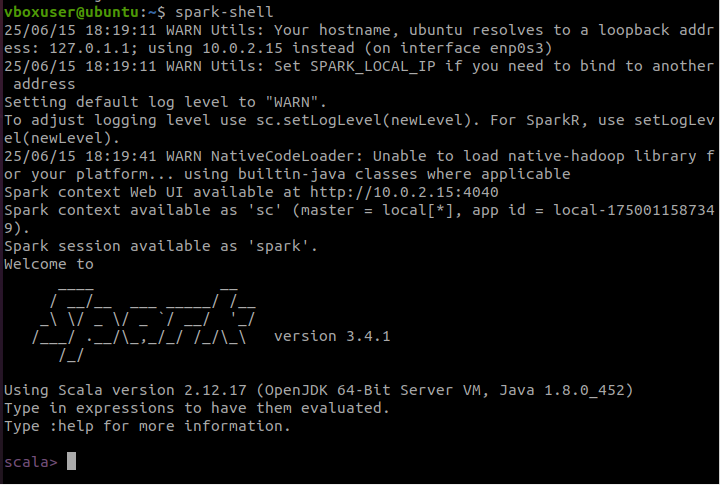
Le fichier contient la listes des mots présent dans le texte d'entrée , avec les nombres d'auccurrence de chacun.

Ces resultats confirme le bon fonctionnement du programme WordCount développé par hadoop.

TP3:SPARK:

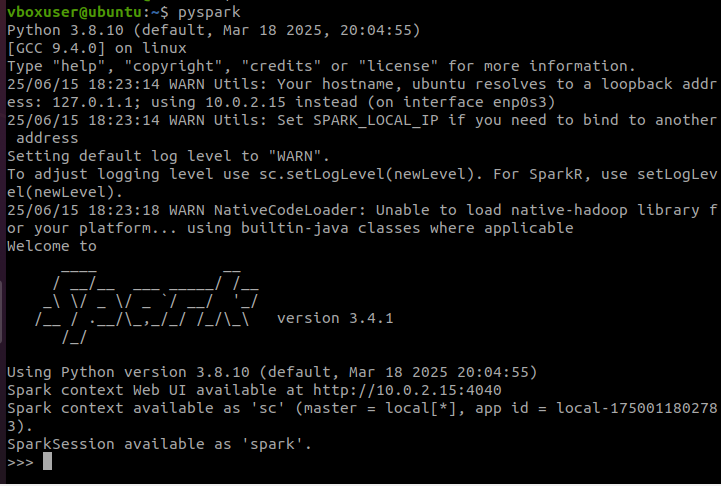
Premier étapes Préparation de l'environnement:

Elle install spark version 3.4.1



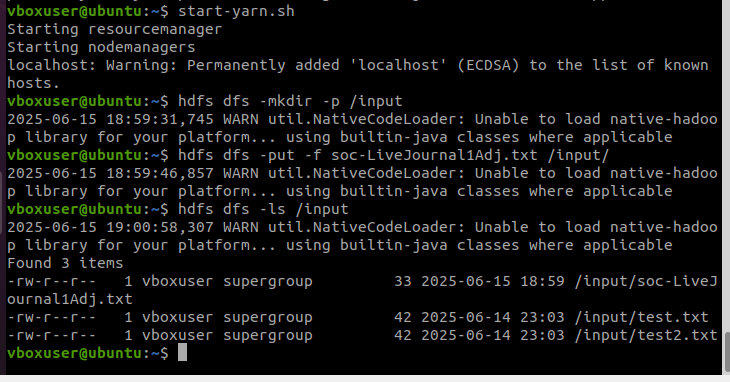
On exécute le commande spark-shell pour ouvrir l'interface de spark

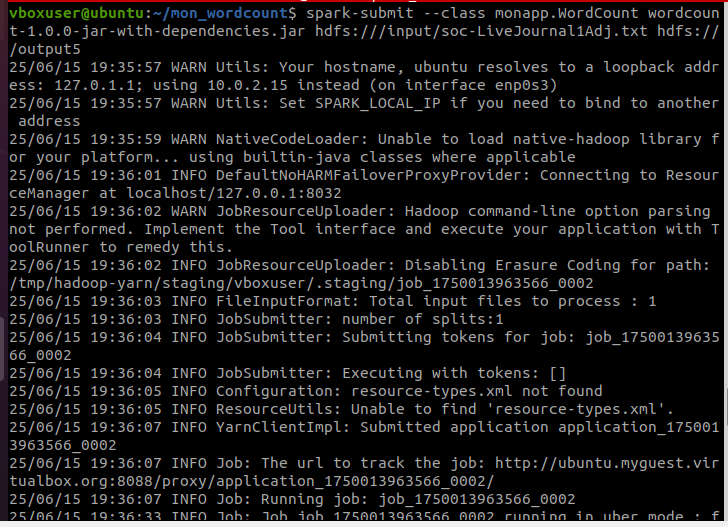
Cela permet de **lancer une session interactive Apache Spark** avec le langage **Scala**. Ce mode est utilisé pour tester rapidement des commandes Spark et manipuler des données directement depuis le terminal.



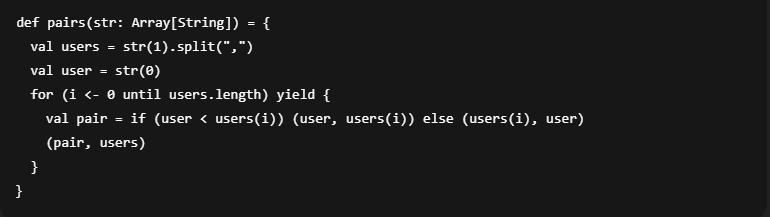
Ici on exécute le commande pyspark

Cela permet de **lancer une session interactive de Spark avec Python**, appelée **PySpark**. Ce mode est utilisé pour exécuter du code Spark écrit en Python directement depuis le terminal.





# Rôle de la fonction `pairs`



Cette fonction prend une ligne (séparée par tabulation) comme un tableau `[user, liste\_d\_amis]`. Pour chaque ami du `user`, elle crée un couple `(user, ami)` en s’assurant de garder l’ordre croissant pour éviter les doublons (ex: (2,3) pas (3,2) et (2,3)). Elle retourne des paires `(user1, user2)` associées à la liste d’amis complète du user.  
  
But : Préparer les données pour ensuite faire l'intersection des amis entre les deux utilisateurs.

Pour chaque paire (A, B), on veut comparer leurs listes d’amis et trouver l’intersection (= amis en commun).

## **Explication du bloc suivant**

var ans = ""

val p2 = p1.map(...).filter(...).flatMap(...).collect()

ans = ans + "0\t4\t" + p2.mkString(",") + "\n"

### **But de ce code :**

Extraire et afficher **les amis en commun entre des utilisateurs spécifiques** (ex: 0 et 4).

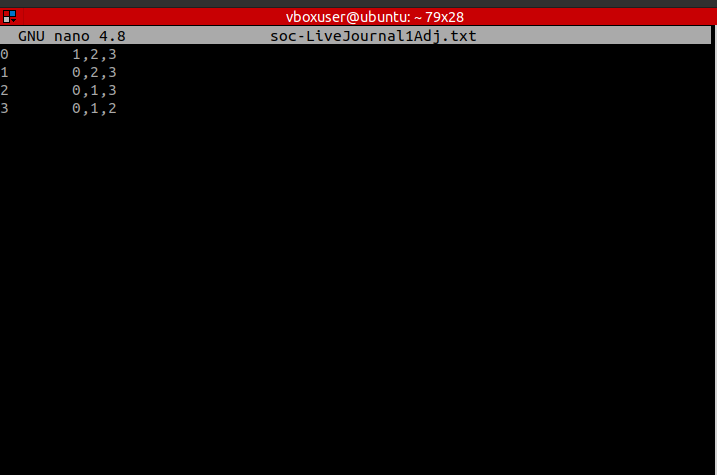
**Étapes expliquées :**

1. p1.map(x => x.split("\t")) : découpe chaque ligne du résultat pour obtenir les trois parties : A, B, amis.
2. filter(x => x(0) == "0" && x(1) == "4") : ne garde que les lignes correspondant à la paire d’utilisateurs souhaitée.
3. flatMap(x => x(2).split(",")) : transforme la liste des amis communs en tableau.
4. collect() : récupère le résultat dans le programme principal.
5. mkString(",") : formate le résultat en chaîne de caractères.

**3 Role du code spark:**

Ce programme Spark lit une liste d’amis depuis un fichier, puis génère toutes les paires d’utilisateurs pour calculer leurs amis en commun à l’aide de la fonction **intersect**. Les résultats sont formatés et sauvegardés dans deux fichiers : un avec toutes les paires (**output**), l’autre avec des cas ciblés (output1). Le programme utilise les transformations Spark classiques comme **map**, **flatMap**, **filter** et **reduceByKey**.

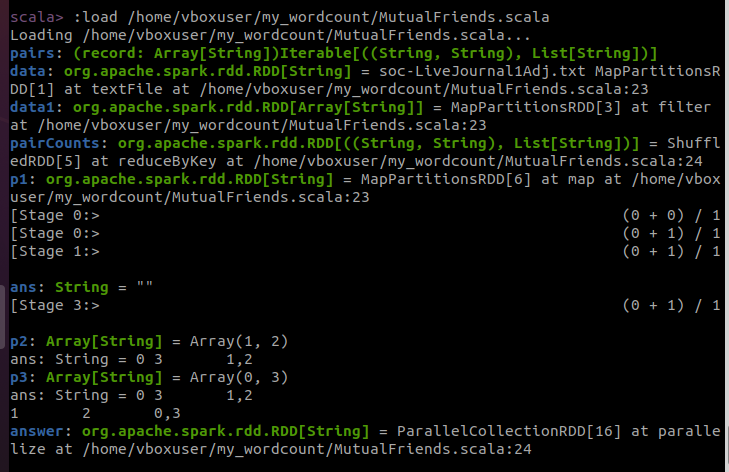
Premièrement elle crée les fichier soc-LiveJournal1Adj.txt qui contient les ID des utilisateur et son amis.



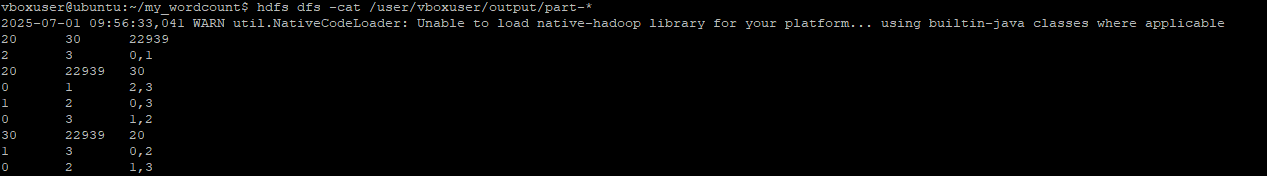
Tu es dans le **shell Scala Spark (spark-shell)**, et tu viens de **charger et exécuter ton fichier MutualFriends.scala**, situé dans /home/vboxuser/my\_wordcount/.

load /home/vboxuser/my\_wordcount/MutualFriends.scala

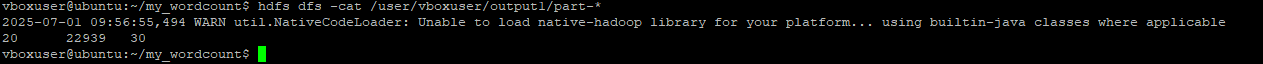
Tu as lancé ton fichier contenant tout le code pour traiter les amis en commun avec Spark.



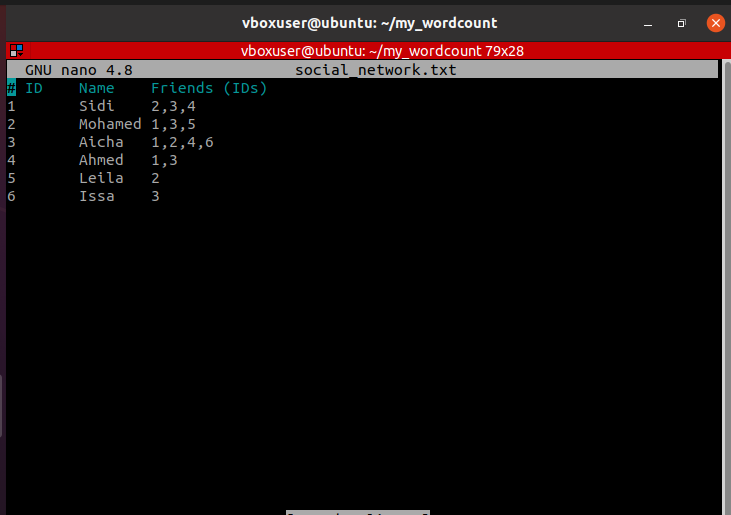
Cette commande affiche le fichier de sortie généré par Spark contenant toutes les paires d’utilisateurs amis, accompagnées de leur liste d’amis en commun. Chaque ligne est structurée comme UserA\tUserB\tListeDesAmisCommuns. Ce fichier est obtenu en appliquant p1.saveAsTextFile("output") après avoir effectué l’intersection des listes d’amis pour chaque paire dans Spark.



Ce fichier output1 contient uniquement les résultats ciblés pour des paires spécifiques d’utilisateurs. Par exemple, il affiche les amis en commun des utilisateurs (20,22939). Ces résultats ont été extraits du résultat global (output) à l’aide de filtres dans le programme Spark, puis enregistrés dans un fichier séparé à l’aide de saveAsTextFile("output1").



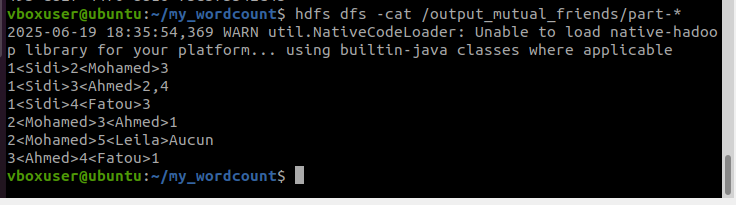
Dans le cadre de ce projet, nous avons créé un fichier texte nommé social\_network.txt qui représente le graphe social à analyser. Ce fichier contient la liste des utilisateurs et leurs relations d'amitié, sous la forme suivante :



À l’issue de l’exécution du script PySpark sur HDFS, nous avons affiché les résultats des amis communs entre les paires d’utilisateurs à l’aide de la commande :

hdfs dfs -cat /output\_mutual\_friends/part-\*

Le contenu retourné présente les amis communs pour chaque paire d’utilisateurs, par exemple :



Le projet a été déposé sur GitHub dans un dépôt public intitulé mutual-friends-spark, accessible à l’adresse suivante :  
 [**https://github.com/AichetouMoctar/mutual-friends-spark**](https://github.com/AichetouMoctar/mutual-friends-spark)  
Le dépôt contient le script PySpark principal (mutual\_friends.py), les fichiers de test et les résultats, ainsi qu’un README.md documentant toutes les étapes du projet.

