Master Spécialisé

Big Data

COMPTE RENDU DE PROJET

– BGDIA707 –
INTRODUCTION AU FRAMEWORK HADOOP

REALISÉ PAR :

Moussa Caudanna YEO – Pety Ialimita RAKOTONIAINA – Yassine CHAFAI – Victor RIVIÈRE – Mathieu SAUVEUR

SOMMAIRE

INTRODUCTION

Le domaine du Big Data a révolutionné la manière dont les entreprises gèrent et analysent d'énormes volumes de données. Dans le cadre du cours "Introduction à Hadoop" au sein du Mastère Big Data, gestion et analyse de données massives, nous avons entrepris un projet ambitieux visant à explorer les potentialités des technologies de traitement distribué.

Notre objectif initial était de maîtriser l'installation et la configuration de l'écosystème Hadoop (tels que HBase, Zookeeper, HDFS et YARN et Spark en cluster). Les ordinateurs mis à notre disposition au sein de l'école, tous connectés au même réseau, ont été consacrés à l'établissement de notre cluster.

Notre deuxième objectif était l'application concrète de ces technologies pour traiter une base de données substantielle, en l'occurrence, une masse imposante de données sur les prix des vols, totalisant 3.3 Go. Dans cette optique, nous avons concentré nos efforts sur l'exploration de ces données.

Au fil de ce rapport, nous procéderons à une exploration détaillée, abordant les différents aspects de notre projet. Nous commencerons par dévoiler les sources de téléchargement des logiciels essentiels puis nous décrivons chaque étape de la mise en place de l'écosystème Hadoop, Zookeeper et Spark ainsi que leur configuration en cluster. Par la suite, nous verrons comment nous avons pu utiliser ces logiciels dans notre projet d'analyse des prix des vols. Enfin, nous dévoilerons les résultats obtenus au terme de notre analyse.

PARTIE 1 : INSTALLATION ET CONFIGURATION DE L'ENVIRONNEMENT DE TRAVAIL BIG DATA

I. PREREQUIS

A) CONFIGURATION MATERIELLE

Pour ces projets nous allons utiliser 5 machines différentes disponible sur le réseau de l'école.

Après avoir taper la commande "sudo lshw", on constate la configuration suivante :

- CPU : AMD EPYC-Milan Processor, 1 core, capacité 2 GHz
- RAM : 8 GB
- ROM : 80 GB (commande df -h)

B) CONFIGURATION LOGICIELLE

1. Système d'exploitation

Nous sommes sur des machines utilisant UBUNTU en version 22.04. (commande : « lsb_release -a »).

2. Java

Les versions de Java installées sur les différentes machines sont openjdk "1.8.0_392" (commande : « java -version »).

3. Autres dépendances nécessaires

Il est important que les programme OpenSSH soit installé sur les machines afin de pouvoir créer des connexions entre elles. Dans notre cas nous utilisons la version OpenSSH_8.9p1 Ubuntu-3ubuntu0.3, OPENSSL 3.0.2 15 Mar 2022. (commande : « ssh -v localhost »)

II. INSTALLATION DE HADOOP

A) TELECHARGEMENT ET EXTRACTION

Dans la machine virtuelle de chaque futur cluster, on télécharge la version stable et la plus récente de Hadoop :

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

Ensuite l'installation de Hadoop passe par la décompression du logiciel. Sur ce, le code utilisé est le suivant :

```
tar -xzvf hadoop-x.y.z.tar.gz -C /chemin/de/destination
```

Une fois le fichier est décompressé, il est permis de configurer Hadoop.

Avant d'entrer dans les configurations des fichiers, il faut configurer préalablement l'environnement pour que le cluster système sache où trouver les fichiers exécutables. Il 'agit aussi des configurations de Hadoop proprement dit. La configuration de la variable d'environnement **HADOOP_HOME** est une étape essentielle dans ce processus :

HADOOP_HOME=/path/to/hadoop export HADOOP_HOME

Les étapes ci-dessus ne modifient le PATH et HADOOP_HOME que pour la session de terminal actuelle. Pour rendre ces modifications permanentes, il faut les ajouter au fichier de profil d'utilisateur, comme .bashrc. Pour ce faire, on y ajoute la ligne suivante :

export PATH=$PATH:$HADOOP_HOME/bin

On teste maintenant si Hadoop est effectivement installé avec : **hadoop version**

Ce qui affiche la version de **3.3.6.**

B) CONFIGURATION DE DEPART DE HADOOP

La configuration se poursuit dans chaque nœud comme suit.

1. *Configuration clé de Hadoop*

Une fois installé, il faut donc configurer en premier les fichiers de configuration Java de Hadoop. Ils se trouvent dans le répertoire **etc/hadoop/**. Les principaux fichiers à configurer sont :

- **core-site.xml** : Configuration générale de Hadoop, comme le système de fichiers par défaut et les paramètres de port.
- **hdfs-site.xml** : Configuration spécifique au système de fichiers Hadoop (HDFS), comme les paramètres de réplication et de stockage.
- **yarn-site.xml** : Configuration spécifique à YARN, le gestionnaire de ressources de Hadoop.
- **mapred-site.xml** : Configuration spécifique à MapReduce.

Parlons une à une des configurations que nous avons faites.

Pour **core-site.xml**, cette configuration définit le système de fichiers par défaut pour Hadoop. Il indique où les données doivent être stockées et gérées dans le cluster HDFS. Pour plus de précision, voyons les paramètres de configuration :

- **Balise <configuration>** : Cette balise englobe l'ensemble des propriétés de configuration dans le fichier XML. Toutes les propriétés doivent être définies à l'intérieur de cette balise.

- **Balise <property>** : Chaque paramètre de configuration individuel est encapsulé dans une balise <property>. Hadoop utilise ces balises pour lire et appliquer les configurations.

- **Balise <name>** : Cette balise contient le nom de la propriété de configuration. Dans ce cas, fs.defaultFS est le nom de la propriété.

- **fs.defaultFS** : Cette propriété spécifie le nom du système de fichiers par défaut que Hadoop doit utiliser. Si un chemin dans Hadoop ne spécifie pas un schéma, il sera traité en fonction de ce qui est défini dans fs.defaultFS.

- **Balise <value>** : Cette balise contient la valeur attribuée à la propriété. Ici, hdfs://localhost:9000 est la valeur assignée.

- **hdfs://localhost:9000** : On utilise Hadoop en mode standalone. Celà veut dire que hadoop est utilisé dans la machine locale pour des tests ou de développement.

Pour **hdfs-site.xml** :

- **Balise <name>** : dfs.replication est le nom de la propriété.

- **Balise <value>** : 1 est la valeur attribuée.

- **Description** : Cette propriété définit le facteur de réplication par défaut pour HDFS. Le facteur de réplication est le nombre de copies d'un fichier qui sont conservées dans le cluster HDFS. Un facteur de 1 signifie qu'il n'y aura qu'une seule copie de chaque bloc de données. Ce qui n'est pas recommandé pour la production car cela ne fournit aucune redondance. Mais dans un environnement de test comme ici, cette situation s'adapte bien.

En gros, la configuration de dfs.replication affecte directement la durabilité et la disponibilité des données.

Pour **yarn-site.xml** :

- **<name>yarn.nodemanager.aux-services</name>** : Ce paramètre indique les services auxiliaires qui seront exécutés par NodeManager. mapreduce_shuffle est un service auxiliaire commun qui permet le tri et le transfert des données pendant les phases de shuffle et de reduce dans MapReduce.

- **<name>yarn.nodemanager.env-whitelist</name>** : Cette propriété spécifie les variables d'environnement qui seront autorisées dans les conteneurs lancés par le NodeManager de YARN. Les valeurs listées, comme **JAVA_HOME**, **HADOOP_COMMON_HOME**, et bien d'autres comprennent les chemins et configurations essentiels pour les différents composants de Hadoop. Elles assurent que les applications exécutées sous YARN ont accès à toutes les ressources. Il s'agit aussi des configurations nécessaires pour leur fonctionnement optimal.

Pour **mapred-site.xml** :

- **mapreduce.framework.name** : Définit YARN comme le gestionnaire de ressources pour les tâches MapReduce.
- **mapreduce.application.classpath** : Spécifie les chemins où MapReduce cherche les classes et bibliothèques nécessaires à son exécution.

Une fois que tous ces fichiers marchent à merveilles dans chaque nœud, on peut passer au démarrage du cluster Hadoop.

2. *Formatage de HDFS*

Pour la première installation avec un nouveau système de fichiers HDFS, il faut formater HDFS. Cette étape est cruciale car elle initialise le système de fichiers. Elle est aussi nécessaire avant le premier démarrage du NameNode.

La commande de formatage : [hdfs]$ $HADOOP_HOME/bin/hdfs namenode -format

3. *Démarrage des points clés*

Pour **HDFS** :

- Lancement du NameNode HDFS sur le nœud désigné.
- Commande : [hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start namenode
- Sur chaque nœud DataNode, il faut lancer le DataNode HDFS.
- Commande : [hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start datanode
- Pour démarrer tous les processus HDFS simultanément, il faut utiliser le script start-dfs.sh

Pour **YARN** :

- Démarrage du ResourceManager YARN sur le nœud désigné.
- Commande : [yarn]$ $HADOOP_HOME/bin/yarn --daemon start resourcemanager
- Sur chaque nœud de travail, il faut lancer le NodeManager YARN.
- Commande : [yarn]$ $HADOOP_HOME/bin/yarn --daemon start nodemanager
- On démarre tous les processus YARN simultanément.
- Commande : [yarn]$ $HADOOP_HOME/sbin/start-yarn.sh

Démarrage du serveur WebAppProxy :

- Le serveur WebAppProxy joue un rôle clé dans la sécurisation, la gestion et la simplification de l'accès aux applications YARN dans un cluster Hadoop.

- Commande : [yarn]$ $HADOOP_HOME/bin/yarn --daemon start proxyserver

4. *Vérification du démarrage de Hadoop*

Après avoir démarré les services, il est bon de vérifier que tout fonctionne correctement. Sur ce, nous pouvons utiliser l'interface utilisateur Web de Hadoop ou des commandes de vérification d'état.

Accès aux Interfaces Utilisateur Web de Hadoop :

Chaque composant de Hadoop fournit une interface web accessible via un navigateur. Les URL typiques sont :

- NameNode HDFS : http://<namenode-host>:9870

Il faut remplacer <namenode-host> par l'adresse IP de NameNode.

- ResourceManager YARN : http://<resourcemanager-host>:8088

Il faut remplacer <resourcemanager-host> par l'adresse IP de ResourceManager.

Ici, le réglage de proxy sur l'ordinateur peut être nécessaire pour qu'il y ait adéquation.

Commandes de Vérification d'État :

- **Pour vérifier NameNode, DataNode, et SecondaryNameNode** :jps

- Rechercher les noms des processus dans la liste des processus Java en cours d'exécution.

- **Pour vérifier ResourceManager et NodeManager** : Même commande jps sur les nœuds où ces services devraient fonctionner.

- **Pour vérifier JobHistory Server** : Utiliser également jps sur le nœud où le JobHistory Server est déployé.

Surveillance des Ressources Système :

- **Utilisation de htop** :

Exécutez top ou htop (si installé) : htop

- Cet outil affiche une liste des processus en cours avec l'utilisation de la CPU, de la mémoire, et d'autres statistiques système.

Combinaison des Méthodes :

- L'utilisation des interfaces utilisateurs web pour une vue d'ensemble et des détails spécifiques à Hadoop.

- L'utilisation des commandes comme **jps**, **top**, et **htop** pour des vérifications plus techniques et des surveillances au niveau du système.

5. *Vérification du démarrage de Hadoop*

Dans la pratique, les étapes suivantes sont transversales : démarrage, vérification, stop, configuration. Les trois processus sont déjà décrits sauf le stop.

Quand la vérification a montré une défaillance de fonctionnement dans certains éléments de Hadoop, on fait le stop. Tout l'ensemble de code de démarrage s'applique lorsqu'on veut stop mais il nous faut juste remplacer les « start » en « stop » dans les commandes.

C) DEMARRAGE DU CLUSTER DE HADOOP

1. *Création de l'host file*

Pour que chaque nœud communiquer entre eux par nom, il faut éditer le fichier /etc/hosts pour ajouter les adresses IP privées des 05 serveurs :

```
 GNU nano 6.2                          hosts
192.168.3.149   node-master
192.168.3.216   node1
192.168.3.53    node2
192.168.3.142   node3
192.168.3.196   node4
```

2. *Configuration sur le node master*

La distribution des paires de clés d'authentification pour l'utilisateur Hadoop est un processus destiné à configurer une connexion SSH sécurisée entre le nœud maître et les nœuds de travail d'un cluster Hadoop. Voici un résumé des étapes :

Génération d'une Clé SSH sur le Nœud Maître :

- Il faut se connecter à la machine master suivant : 192.168.3.149

- Master génère par la suite une clé SSH en utilisant la commande « **ssh-keygen -b 4096** ». Lors de la création de cette clé, il ne faut pas entrer de mot de passe pour permettre une communication automatique et sans interruption de maître Hadoop.

Distribution de la Clé Publique du Nœud Maître aux Nœuds de Travail :

Cette étape est cruciale dans le processus de mise en place d'une authentification SSH sans mot de passe pour un cluster Hadoop. Elle permet au nœud maître de communiquer de manière sécurisée et automatique avec les nœuds de travail.

- **Affichage et Copie de la Clé Publique du Nœud Maître** : la clé publique se situe à l'endroit suivant **/home/hadoop/.ssh/id_rsa.pub**. Cette clé est celle qui sera utilisée par les nœuds de travail pour authentifier le nœud maître. Pour ce faire, on suit l'étape suivante.

- **Création et Remplissage du Fichier master.pub sur les Nœuds de Travail** : Sur chaque nœud de travail, on crée un fichier nommé **master.pub** dans le répertoire **/home/hadoop/.ssh**. On y colle la clé publique du nœud maître. Ce fichier sert de moyen pour stocker la clé publique du nœud maître sur chaque nœud de travail.

Établir la localisation de namenode :

Pour les différents nœuds, on applique la configuration suivante sur **core-site.xml**. Notons d'abord que **core-site.xml** indique où les données doivent être stockées et gérées dans le cluster HDFS. Pour plus de précision, voyons les paramètres de configuration :

- **Balise <value>** : Cette balise contient hdfs://192.168.3.51:9000 mais non plus hdfs://localhost:9000.

- Ce changement signifie que le NameNode de Hadoop est désormais configuré pour fonctionner sur une machine spécifique dans le réseau, accessible via l'adresse IP 192.168.3.51. Ce réglage est essentiel pour passer d'un environnement Hadoop local à un environnement distribué.

Établir la localisation de HDFS :

- **On a ajouté la Balise <name>** : dfs.datanode.data.dir.

- **Balise <value>** : /home/ubuntu/data/dataNode est la valeur attribuée.

- **Description** : Cette propriété spécifie le chemin du répertoire où les Datanodes HDFS stockent leurs blocs de données. Le chemin ~/dataNode_dir indique que chaque Datanode stockera ses blocs dans un répertoire dataNode_dir dans le répertoire personnel de l'utilisateur qui exécute le Datanode. Il est important de s'assurer que ce chemin est valide et que l'utilisateur Hadoop a les permissions nécessaires pour écrire dans ce répertoire.

- **On ajoute aussi Balise <name>dfs.namenode.name.dir</name>** : C'est comme dans data node, ce paramètre spécifie le chemin du répertoire où le NameNode stocke les métadonnées du système de fichiers HDFS. Ce répertoire contiendra des informations essentielles telles que la liste des blocs de données, leur emplacement sur les différents DataNodes, etc.

dfs.datanode.data.dir, elle détermine l'emplacement physique du stockage des données.

Définir YARN comme planificateur de tâche :

On modifie le fichier mapred-site.xml en définissant YARN comme le cadre par défaut pour les opérations MapReduce :

- **Ajout de <name>yarn.app.mapreduce.am.env</name>** : Ce paramètre définit les variables d'environnement pour l'Application Master (AM) de MapReduce. HADOOP_MAPRED_HOME=$HADOOP_HOME indique que l'AM doit utiliser la

même installation Hadoop que celle spécifiée dans la variable d'environnement HADOOP_HOME.

- **Ajout de <name>mapreduce.map.env</name> et <name>mapreduce.reduce.env</name>** : Ces paramètres définissent les variables d'environnement pour les tâches map et reduce, respectivement. Ils sont également définis pour utiliser HADOOP_MAPRED_HOME=$HADOOP_HOME, garantissant la cohérence de l'environnement pour toutes les composantes de MapReduce.

- **Ajout de <name>yarn.app.mapreduce.am.resource.mb</name>** : Ce paramètre spécifie la quantité de mémoire, en mégaoctets, allouée à l'Application Master de MapReduce. Dans votre cas, il est réglé sur 512 MB.

- **Ajout de <name>mapreduce.map.memory.mb</name>** : Ce paramètre définit la quantité de mémoire, en mégaoctets, allouée à chaque tâche map. Vous l'avez configuré pour utiliser 256 MB.

Configuration de YARN :

- **Ajout de <name>yarn.acl.enable</name>** : Ce paramètre détermine si les contrôles d'accès (ACL) sont activés pour les applications YARN. Une valeur de 0 (comme dans votre configuration) signifie que les ACL sont désactivées. Cela peut avoir des implications sur la sécurité et l'accès aux ressources, alors assurez-vous que cela correspond à vos besoins de sécurité et de gouvernance des données.

- **<name>yarn.resourcemanager.hostname</name> :** On la même machine maitre ici par rapport à hdfs. Il s'agit de celle qui a l'adresse suivante : 192.168.3.216.

- Notons que nous avons fait ce choix pour la raison suivante.

- **Point Centralisé pour les Ressources et les Données** : Le nœud avec cette adresse IP agit comme le nœud maître pour à la fois la gestion des ressources (YARN) et le stockage des données (HDFS).

- **Simplification de la Configuration** : Cela peut simplifier la configuration et la gestion du cluster, car les composants clés sont centralisés.

- **Considérations de Performance et de Fiabilité** : Nous avons un cluster petit, cette configuration peut être suffisante.

- **Scalabilité** : Un cluster Hadoop peut commencer avec une configuration simple sur un nombre limité de machines, mais il est conçu pour être facilement extensible.

- **Ajout de <name>yarn.nodemanager.aux-services</name>** : Ce paramètre indique les services auxiliaires qui seront exécutés par NodeManager. mapreduce_shuffle est un service auxiliaire commun qui permet le tri et le transfert des données pendant les phases de shuffle et de reduce dans MapReduce.

3. Configuration des workers

Dans le fichier **~/hadoop/etc/hadoop/workers**, on établit la liste des workers :

```
 GNU nano 6.2                                    workers
92.168.3.149
192.168.3.142
192.168.3.53
192.168.3.196
```

4. *Configuration des mémoires allouées*

Pour **Yarn** **/home/hadoop/hadoop/etc/hadoop/yarn-site.xml** :

- **yarn.nodemanager.resource.memory-mb (1536 MB**) : Définit la mémoire totale (1536 MB) que le NodeManager peut allouer pour les conteneurs sur un nœud.

- **yarn.scheduler.maximum-allocation-mb (1536 MB)** : Fixe la limite supérieure de mémoire (1536 MB) qu'un conteneur unique peut utiliser.

- **yarn.scheduler.minimum-allocation-mb (128 MB)** : Établit la limite inférieure de mémoire (128 MB) allouable à un conteneur.

- **yarn.nodemanager.vmem-check-enabled (false)** : Désactive la vérification de la mémoire virtuelle par le NodeManager

Pour **mapred-site.xml**, on ajoute :

- **<name>yarn.app.mapreduce.am.resource.mb</name>** : Ce paramètre spécifie la quantité de mémoire, en mégaoctets, allouée à l'Application Master de MapReduce. Dans notre cas, il est réglé sur 512 MB.

- **<name>mapreduce.map.memory.mb</name>** : Ce paramètre définit la quantité de mémoire, en mégaoctets, allouée à chaque tâche map. Nous l'avons configuré pour utiliser 256 MB.

III. INSTALLATION DE ZOOKEEPER

A) TÉLÉCHARGEMENT ET EXTRACTION

Dans la machine virtuelle de chaque futur cluster, on télécharge la version stable et la plus récente de Zookeeper :

```
wget https://dlcdn.apache.org/zookeeper/zookeeper-3.8.3/apache-zookeeper-3.8.3-bin.tar.gz
```

Ensuite l'installation de Zookeeper passe par la décompression du dossier. Sur ce, le code utilisé est le suivant :

```
tar -xvzf apache-zookeeper-3.8.3-bin.tar.gz
```

Une fois que le fichier est décompressé, il est permis de configurer Zookeeper.

B) CONFIGURATION DE ZOOKEEPER

Dans un premier temps nous allons configurer le fichier **zoo.dfg**. Dans ce fichier on y indique certaines limites de traitement comme le nombre de serveurs ou le temps de tick entre eux… Mais aussi l'adresse des machines appartenant au cluster Hadoop.

```
tickTime=2000
dataDir=/home/ubuntu/zookeeper-data
maxClientCnxns=60
initLimit=10
syncLimit=5
clientPort=2181
server.1=192.168.3.216:2888:3888
server.2=192.168.3.149:2888:3888
server.3=192.168.3.53:2888:3888
server.4=192.168.3.142:2888:3888
server.5=192.168.3.196:2888:3888
```

On remarque que nous avons indiqué le chemin vers un fichier qui n'existe pas encore, on doit donc créer le dossier **/home/ubuntu/data/zookeeper**.

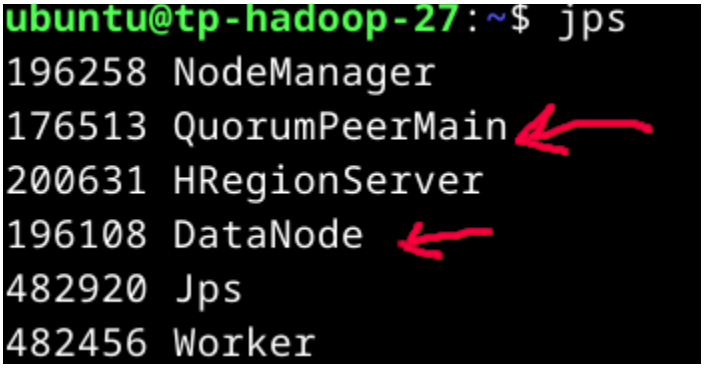
Dans ce même dossier nous devons créer un fichier avec la commande « **sudo nano ~/data/zookeeper/myid** » et y inscrire sur les machines le numéro attribué à son IP dans le dossier **zoo.cfg**. Par exemple dans le fichier myid de la machine ayant l'IP 192.168.3.196 le numéro à indiquer dans le fichier est 5.

C) DÉMARRAGE DE L'ENSEMBLE ZOOKEEPER

Afin de démarrer les Zookeeper sur chaque machine, nous devons exécuter les commandes "bin/zkServer.sh start" sur chaque machine.

D) VÉRIFICATION DE L'INSTALLATION

Quand les serveurs sont démarrés sur chaque machine, on peut vérifier si tous les nœuds sont bien en état de marche. En tapant jps, les nœuds relatifs à ZooKeeper tel que **QuorumPeerMain**, **DataNode**, etc doivent être présents.



Afin de savoir qui est le leader il suffit de lancer la commande suivante sur chaque machine, si la sortie de mode est 'leader' alors le serveur est le leader

```
ubuntu@tp-hadoop-27:~$ ~/apache-zookeeper-3.8.3-bin/bin/zkServer.sh status
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /home/ubuntu/apache-zookeeper-3.8.3-bin/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
```

Ici la machine tp-hadoop-27 est la machine leader pour Zookeeper.

IV. INSTALLATION DE HBASE

A) TÉLÉCHARGEMENT ET EXTRACTION

Dans la machine virtuelle de chaque futur cluster, on télécharge la version stable et la plus récente de HBase :

```
wget https://dlcdn.apache.org/hbase/stable/hbase-2.5.5-hadoop3-bin.tar.gz
```

Ensuite l'installation de HBase passe par la décompression du dossier. Sur ce, le code utilisé est le suivant :

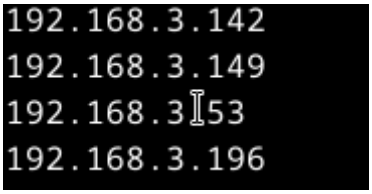
```
tar -xvzf hbase-2.5.5-hadoop3-bin.tar.gz
```

Une fois que le fichier est décompressé, il est permis de configurer HBase.

B) CONFIGURATION DE HBASE

La première étape de configuration va être de désactiver le Zookeeper installer de base avec HBase, en effet, HBase intègre un Zookeeper par défaut. Il faut donc mettre dans le fichier hbase-env.sh la valeur export **HBASE_MANAGES_ZK=false**, afin d'utiliser notre propre Zookeeper.

Aussi dans le fichier **conf/regionservers**, il faut inscrire les **ip** des machines slaves :



En suite 2 configurations différentes sont à effectuer en fonction des machines slaves et de la machine Master.

Dans les machines slaves :

Mettre dans le fichier hbase-site.xml.

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>false</value>
</property>
<property>
  <name>hbase.tmp.dir</name>
  <value>./tmp</value>
</property>
<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
</property>
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://192.168.3.196:2181/hbase</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
</configuration>
```

Alors que dans la machine Master il faut mettre dans ce même fichier les instructions suivantes :

```
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://192.168.3.216:9000/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.tmp.dir</name>
    <value>./tmp</value>
  </property>
  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>false</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>192.168.3.53,192.168.3.216,192.168.3.142,192.168.3.149,192.168.3.196</value>
  </property>
  <property>
    <name>hbase.zookeeper.peerport</name>
    <value>2888</value>
  </property>
  <property>
    <name>hbase.zookeeper.leaderport</name>
    <value>3888</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2181</value>
  </property>
  <property>
    <name>hbase.master</name>
    <value>192.168.3.216:16000</value>
  </property>
  <property>
    <name>hbase.master.port</name>
    <value>16000</value>
  </property>
</configuration>
```

C) DÉMARRAGE DE HBASE

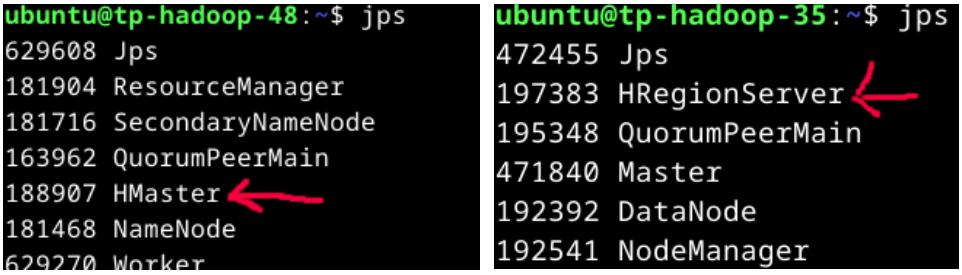
La commande pour démarrer HBase : **bin/start-hbase.sh**

La commande pour rentrer dans le shell de HBase : **bin/hbase shell**

La commande pour arrêter HBase : **bin/stop-hbase.sh**

D) VÉRIFICATION DE L'INSTALLATION

Quand un HBase est démarré l'exécution de la commande JPS doit afficher HRegionServer sur toutes les machines ainsi que HMaster sur la machine Master.



Ici la machine tp-hadoop-48 est le master HBase et la machine 35 un slave.

Afin de vérifier si HBase fonctionne correctement on peut accéder à l'interface via le l'adresse, http://192.168.3.216:16010/master-status.

V. INSTALLATION DE SPARK

A) TÉLÉCHARGEMENT ET EXTRACTION

Dans la machine virtuelle de chaque futur cluster, on télécharge la version stable et la plus récente de Spark :

```
wget https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
```

Ensuite l'installation de Spark passe par la décompression du dossier. Sur ce, le code utilisé est le suivant :

```
tar -xvzf spark-3.5.0-bin-hadoop3.tgz
```

Une fois que le fichier est décompressé, il est permis de configurer Spark.

B) CONFIGURATION DE SPARK

Dans le fichier il faut rajouter les instructions suivantes :

```
# Spécifiez l'adresse IP du nœud maître Spark
export SPARK_MASTER_HOST="192.168.3.142"

# Configurez la mémoire allouée aux travailleurs Spark
export SPARK_WORKER_MEMORY="2g"

# Configurez le nombre de cœurs par travailleur
export SPARK_WORKER_CORES="2"

# Configurez le nombre d'instances de travailleurs par machine
export SPARK_WORKER_INSTANCES="1"

# Spécifiez les répertoires locaux pour le stockage temporaire des données Spark
export SPARK_LOCAL_DIRS="/var/spark/local"

# Autres configurations Spark...
```

Sur la copie d'écran le master à l'adresse **192.168.3.142** dans notre cas le master final sera l'adresse finissant par 216.

Aussi le fichier **spark-default.conf** doit être complété de la sorte :

```
# Example:
# spark.master                     spark://master:7077
# spark.eventLog.enabled           true
# spark.eventLog.dir               hdfs://namenode:8021/directory
# spark.serializer                 org.apache.spark.serializer.KryoSerializer
# spark.driver.memory              5g
# spark.executor.extraJavaOptions  -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"
spark.master yarn
spark.driver.memory 1g
spark.yarn.am.memory 1g
spark.executor.memory 3120mb
spark.executor.memoryOverhead=1G
spark.driver.memoryOverhead=1G
spark.eventLog.enabled true
spark.eventLog.dir hdfs://192.168.3.216:9000/spark-logs
spark.history.provider org.apache.spark.deploy.history.FsHistoryProvider
spark.history.fs.logDirectory hdfs://192.168.3.216:9000/spark-logs
spark.history.fs.update.interval 10s
spark.history.ui.port 18080
spark.hadoop.fs.defaultFS hdfs://192.168.3.216:9000
spark.hadoop.conf.dir /home/ubuntu/hadoop-3.3.6/etc/hadoop
```

C) DÉMARRAGE DU CLUSTER SPARK

Pour démarrer le cluster Spark, il suffit que le Master rentre la commande : **./sbin/start-all.sh**

De même pour arrêter le cluster il faut rentrer sur le Master la commande : **./sbin/stop-all.sh**

D) VÉRIFICATION DE L'INSTALLATION

Si toutes les machines sont bien configurées, après entrer la commande JPS le terme "Worker" ou "Master" doit apparaître.

Aussi afin de vérifier si les opérations sur Spark se passe correctement, on peut accéder à l'interface utilisateur via les adresses :

- http://192.168.3.216:8080/ (Spark Master UI)
- http://192.168.3.216:4040/ (Spark application UI)

PARTIE 2 : EXPLORATION DES DONNEES MASSIVES : CAS DES PRIX DE BILLET D'AVION

I. PREPARATION DES DONNEES

Afin d'importer nos données de kaggle à l'une de nos machines nous avons dû télécharger l'environnement Kaggle avec la commande : **pip install kaggle**.

Ensuite il nous a fallu générer un token afin d'utiliser l'API. Après avoir créer un fichier .json contenant le token généré, nous avons pu utiliser la commande : **kaggle datasets download -d dilwong/flightprices**

Après avoir téléchargé le fichier .zip de 7 GB. Nous l'avons unzippé afin de découvrir un fichier de 30GB.

Malheureusement nous avons eu des difficultés à exécuter notre script spark avec pour l'analyse des 30 GB de données, les workers rencontrant des limites de mémoire pour le traitement. Nous avons donc récupéré 10% du dataset, ce qui correspond à un fichier de 3.380 GB de données, pour effectuer le traitement.

Une extraction des 10% a donc été récupérée et stockée sur le HDFS :

```
ubuntu@tp-hadoop-48:~/hadoop-3.3.6$ ./bin/hdfs dfs -ls
Found 10 items
drwxr-xr-x   - ubuntu supergroup          0 2023-12-11 17:06 .sparkStaging
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 12:48 books
-rw-r--r--   1 ubuntu supergroup  3461616129 2023-12-11 16:39 extrait_flight.csv
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 12:51 grep-temp-2016522988
drwxr-xr-x   - ubuntu supergroup          0 2023-10-20 13:10 input
-rw-r--r--   1 ubuntu supergroup 31091834438 2023-12-11 14:49 itineraries.csv
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 12:49 output
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 13:03 output12
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 12:58 output3
drwxr-xr-x   - ubuntu supergroup          0 2023-10-11 13:00 output5
ubuntu@tp-hadoop-48:~/hadoop-3.3.6$
```

II. SCRIPT PYSPARK

A) CREATION DU SCRIPT

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation
```

Les bibliothèques nécessaires sont importées, notamment **Matplotlib**, **Seaborn**, **pandas**, et les composants de **PySpark**. Une session Spark est initialisée avec le nom "desc".

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

df = spark.read.csv(path="extrait_flight.csv", sep=",", header=True, inferSchema=True)
df = df.na.fill("UNKNOWN")
pandas_df = df.select("baseFare").toPandas()
```

Les données sont chargées depuis le fichier CSV stocké dans le système distribué dans un DataFrame Spark. Les valeurs nulles dans le DataFrame sont remplies avec la chaîne "UNKNOWN". Une colonne du DataFrame Spark est extraite et convertie en un DataFrame Pandas pour une visualisation plus facile avec les bibliothèques standard de Python.

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.boxplot(x=pandas_df["baseFare"])
plt.title("Box Plot pour le prix de base")
plt.savefig("boxplot.png")
```

Un diagramme à moustaches (box plot) est créé pour la colonne "baseFare" à l'aide de **Seaborn**. Les réglages visuels sont ajustés avec **Seaborn** et **Matplotlib**. Le diagramme est sauvegardé en tant que fichier image.

```
departures_by_airport = df.groupBy("startingAirport").count().\
    orderBy(F.desc("count")).limit(10).toPandas()

# Nombre d'arrivées par aéroport
arrivals_by_airport = df.groupBy("destinationAirport").count().\
    orderBy(F.desc("count")).limit(10).toPandas()

# Nombre total de vols par aéroport (départs + arrivées)
total_flights_by_airport = df.groupBy("startingAirport", "destinationAirport").count()\
    .withColumnRenamed("count", "totalFlights")
total_flights_by_airport = total_flights_by_airport.groupBy("startingAirport")\
    .agg(F.sum("totalFlights").alias("totalFlights")).orderBy(F.desc("totalFlights")).limit(10).toPandas()
```

Le nombre de départs par aéroport, le nombre d'arrivées et le nombre total de vols (départs + arrivées) par aéroport est calculé en utilisant des fonctions **PySpark**.

Les résultats sont convertis en un DataFrame **Pandas** pour la visualisation avec **Seaborn**. Seuls les dix premiers aéroports sont sélectionnés pour l'analyse.

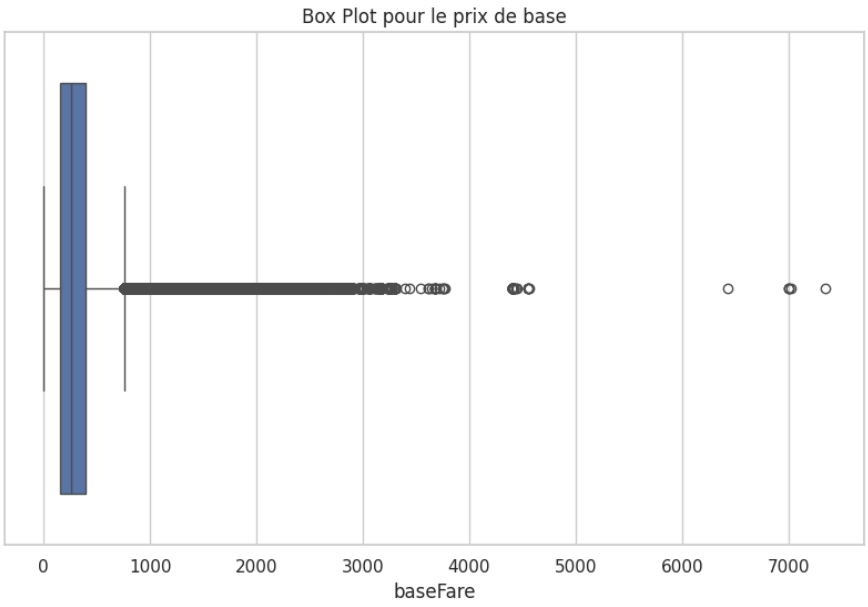
```
numeric_cols = ["baseFare", "totalFare", "seatsRemaining"]

# Assembler les colonnes en un vecteur
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="features")
df_numeric = assembler.transform(df)

# Calculer la matrice de corrélation
correlation_matrix = Correlation.corr(df_numeric, "features").head()

# Convertir la matrice de corrélation en Pandas DataFrame
corr_matrix_pd = pd.DataFrame(correlation_matrix[0].toArray(), columns=numeric_cols, index=numeric_cols)
```

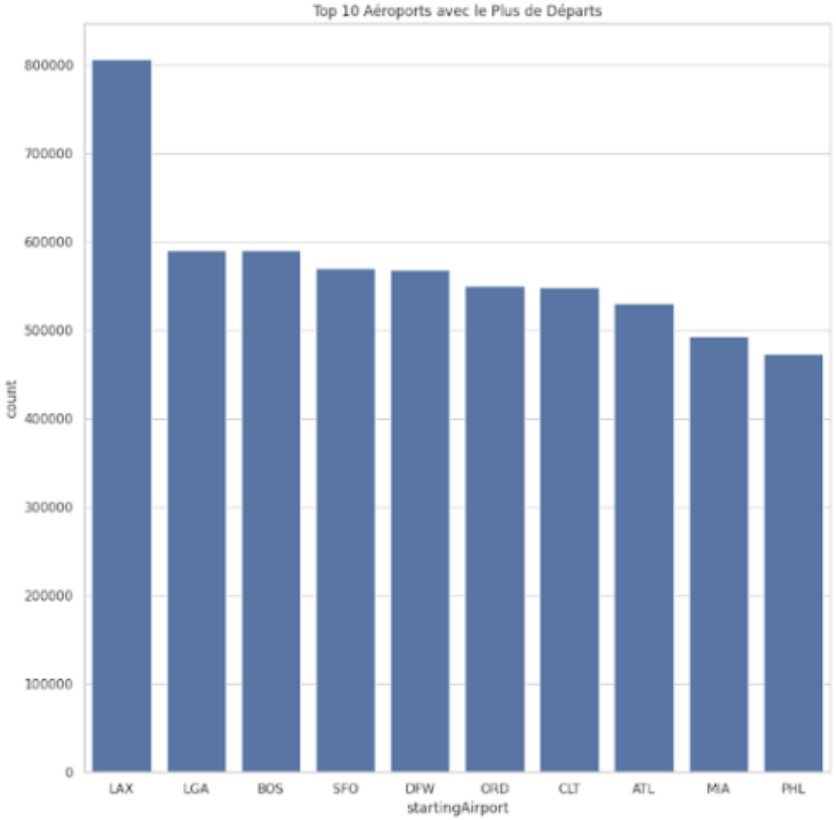
Les colonnes numériques pertinentes sont sélectionnées. Les colonnes sont assemblées en un vecteur à l'aide de **PySpark**. La matrice de corrélation est calculée avec **PySpark**, puis est convertie en un DataFrame Pandas pour une meilleure visualisation.

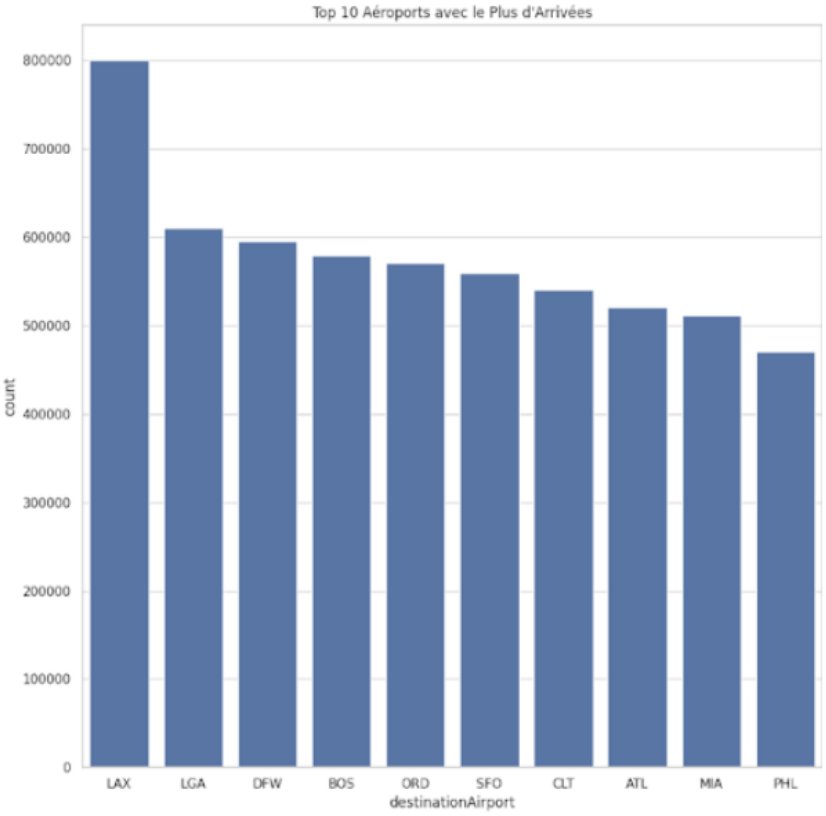
B) RESULTATS

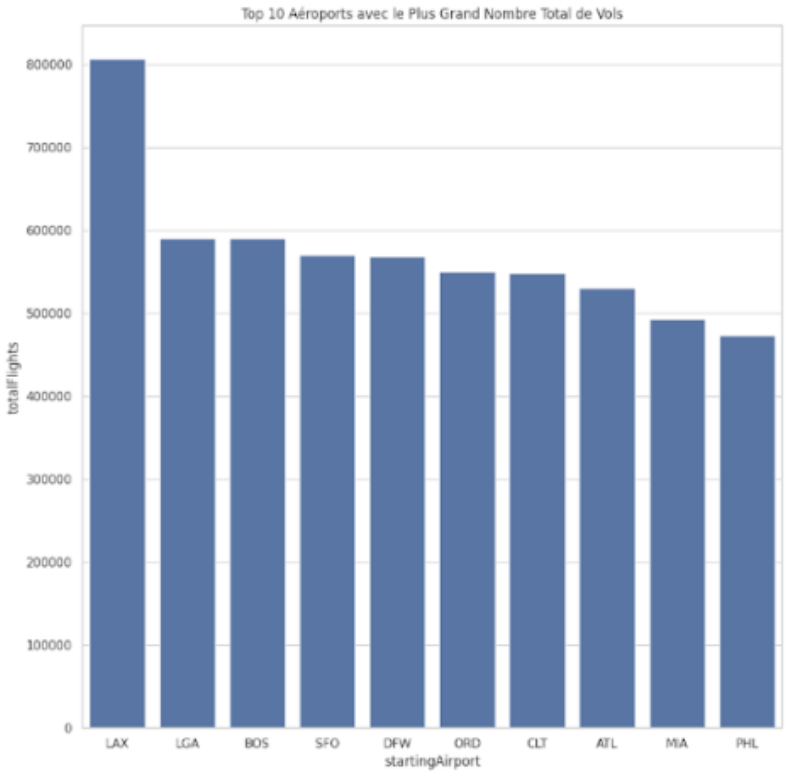
1. *Résultats graphiques du script*

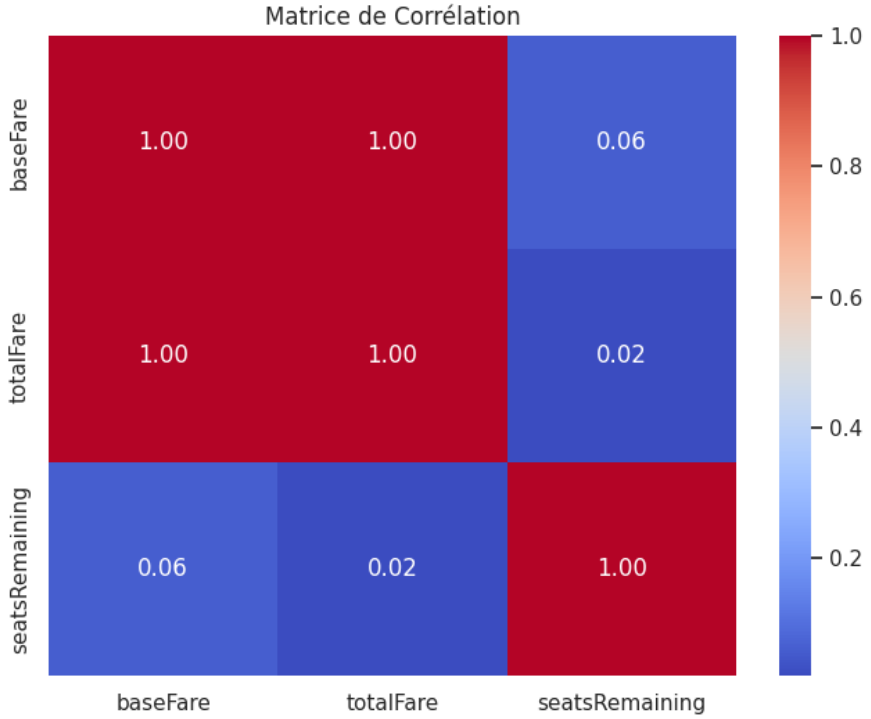
Le script expliqué dans la section précédente a donc pour but de faire une analyse descriptive des données que nous lui avons soumis. Des graphes ont donc été générés, et présentent les résultats de notre analyse prédictive :


Box Plot pour le prix de base
0
1000
2000
3000
4000
5000
6000
7000
baseFare


Top 10 Aéroports avec le Plus de Départs
count
800000
700000
600000
500000
400000
300000
200000
100000
0
LAX
LGA
BOS
SFO
DFW
ORD
CLT
ATL
MIA
PHL
startingAirport


Top 10 Aéroports avec le Plus d'Arrivées
count
800000
700000
600000
500000
400000
300000
200000
100000
0
LAX
LGA
DFW
BOS
ORD
SFO
CLT
ATL
MIA
PHL
destinationAirport

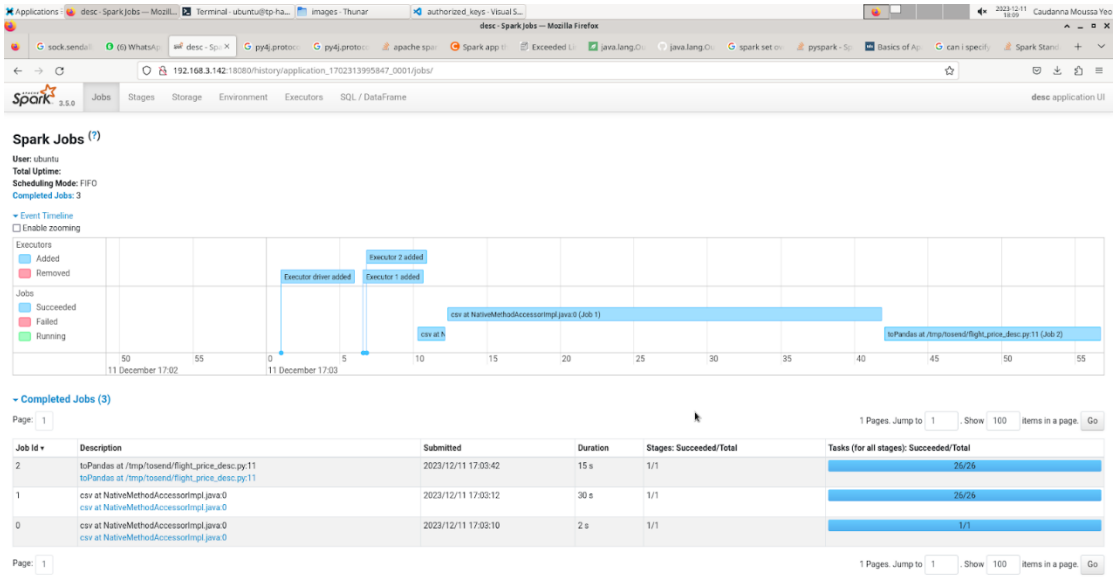

Top 10 Aéroports avec le Plus Grand Nombre Total de Vols
totalFlights
800000
700000
600000
500000
400000
300000
200000
100000
0
LAX
LGA
BOS
SFO
DFW
ORD
CLT
ATL
MIA
PHL
startingAirport

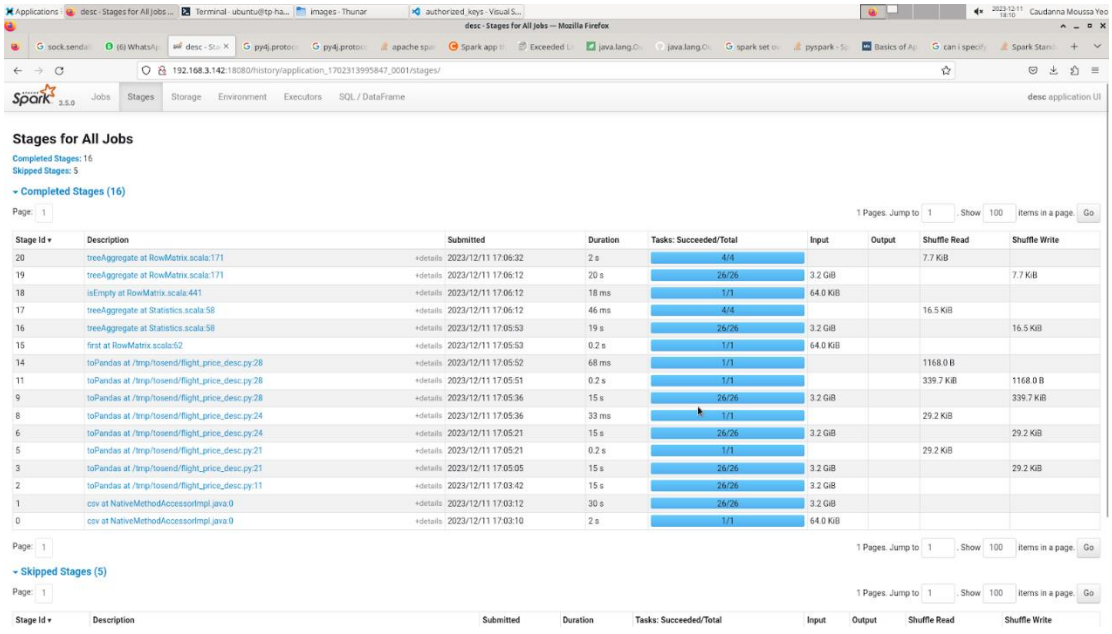


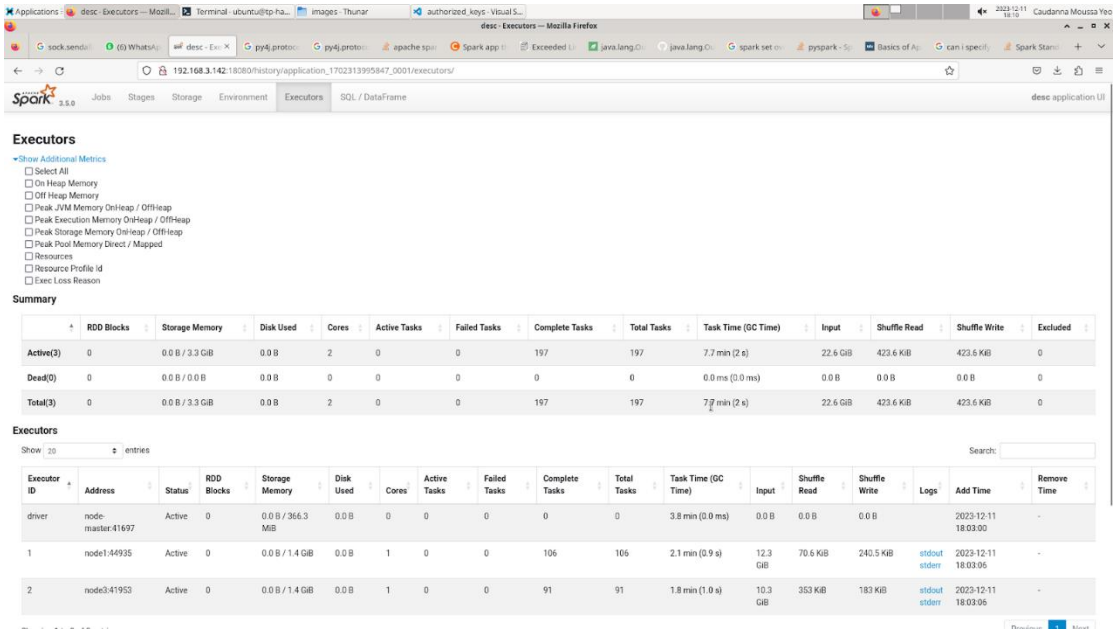
Les graphes précédents montrent la distribution du prix des billets d'avion, les différents aéroports qui accueillent le plus de vols, et les corrélations qu'il existe entre le prix de base, le prix total et le nombre restant de sièges.

2. *Résultat de l'exécution*

L'exécution du script a été monitorée grâce à l'interface web de Spark, disponible à l'adresse http://192.168.3.142:18080/ (le port 18080 ayant été spécifié dans les configurations). Quelques captures sont présentées ci-dessous :


desc - Spark Jobs — Mozilla Firefox
192.168.3.142:18080/history/application_1702313995847_0001/jobs/
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
desc application UI
Spark Jobs (?)
User: ubuntu
Total Uptime:
Scheduling Mode: FIFO
Completed Jobs: 3
Event Timeline
Enable zooming
Executors
Added
Removed
Executor 2 added
Executor driver added
Executor 1 added
Jobs
Succeeded
Failed
Running
csv at NativeMethodAccessorImpl.java:0 (Job 1)
toPandas at /tmp/tosend/flight_price_desc.py:11 (Job 2)
11 December 17:02
11 December 17:03
Completed Jobs (3)
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
2
toPandas at /tmp/tosend/flight_price_desc.py:11
2023/12/11 17:03:42
15 s
1/1
26/26
1
csv at NativeMethodAccessorImpl.java:0
2023/12/11 17:03:12
30 s
1/1
26/26
0
csv at NativeMethodAccessorImpl.java:0
2023/12/11 17:03:10
2 s
1/1
1/1


desc - Stages for All Jobs — Mozilla Firefox
192.168.3.142:18080/history/application_1702313995847_0001/stages/
Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
desc application UI
Stages for All Jobs
Completed Stages: 16
Skipped Stages: 5
Completed Stages (16)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
20 treeAggregate at RowMatrix.scala:171 2023/12/11 17:06:32 2 s 4/4 7.7 KiB
19 treeAggregate at RowMatrix.scala:171 2023/12/11 17:06:12 20 s 26/26 3.2 GiB 7.7 KiB
18 isEmpty at RowMatrix.scala:441 2023/12/11 17:06:12 18 ms 1/1 64.0 KiB
17 treeAggregate at Statistics.scala:58 2023/12/11 17:06:12 46 ms 4/4 16.5 KiB
16 treeAggregate at Statistics.scala:58 2023/12/11 17:05:53 19 s 26/26 3.2 GiB 16.5 KiB
15 first at RowMatrix.scala:62 2023/12/11 17:05:53 0.2 s 1/1 64.0 KiB
14 toPandas at /tmp/tosend/flight_price_desc.py:28 2023/12/11 17:05:52 68 ms 1/1 1168.0 B
11 toPandas at /tmp/tosend/flight_price_desc.py:28 2023/12/11 17:05:51 0.2 s 1/1 339.7 KiB 1168.0 B
9 toPandas at /tmp/tosend/flight_price_desc.py:28 2023/12/11 17:05:36 15 s 26/26 3.2 GiB 339.7 KiB
8 toPandas at /tmp/tosend/flight_price_desc.py:24 2023/12/11 17:05:36 33 ms 1/1 29.2 KiB
6 toPandas at /tmp/tosend/flight_price_desc.py:24 2023/12/11 17:05:21 15 s 26/26 3.2 GiB 29.2 KiB
5 toPandas at /tmp/tosend/flight_price_desc.py:21 2023/12/11 17:05:21 0.2 s 1/1 29.2 KiB
3 toPandas at /tmp/tosend/flight_price_desc.py:21 2023/12/11 17:05:05 15 s 26/26 3.2 GiB 29.2 KiB
2 toPandas at /tmp/tosend/flight_price_desc.py:11 2023/12/11 17:03:42 15 s 26/26 3.2 GiB
1 csv at NativeMethodAccessorImpl.java:0 2023/12/11 17:03:12 30 s 26/26 3.2 GiB
0 csv at NativeMethodAccessorImpl.java:0 2023/12/11 17:03:10 2 s 1/1 64.0 KiB
Skipped Stages (5)



L'exécution du script a donc été effectuée en environ **4 minutes** sur un jeu de données de **3.3 GB**, en utilisant deux workers, ceux présents sur les machines **192.168.3.149** et **192.168.3.142**.

CONCLUSION