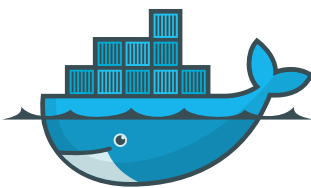


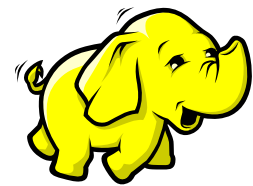
Rapport de Projet



WSL LINUX



DOCKER



**HADOOP
/SPARK**

Préparé par :
Firas Kahlaoui

E-mail:
kahlaouifiras2017@gmail.com

SOMMAIRE

<u>Introduction & Objectifs</u>	3
<u>Installation de Docker</u>	4
<u>Téléchargement d'Images Docker et Déploiement de Conteneurs</u>	6
<u>MapReduce dans un Conteneur Docker</u>	8
<u>Analyse de Données avec Spark</u>	13
<u>Les enseignements tirés de ce projet</u>	20

INTRODUCTION & OBJECTIFS

- **1.1 INTRODUCTION**

Dans le paysage toujours changeant du traitement de données à grande échelle, les entreprises et les organisations cherchent constamment des moyens d'optimiser l'efficacité de leurs flux de travail d'analyse. Dans cette quête d'efficacité, l'adoption de technologies telles que Docker et Apache Spark devient de plus en plus courante.

Docker, avec sa capacité à encapsuler des applications dans des environnements légers et portables appelés conteneurs, offre une solution efficace pour la gestion des environnements de développement et de production. D'autre part, Apache Spark, un moteur de traitement de données rapide et polyvalent, permet l'analyse de données en temps réel et en batch à grande échelle.

- **1.2 OBJECTIFS**

1. Installation de Docker
2. Téléchargement d'Images Docker et Déploiement de Conteneurs
3. MapReduce dans un Cluster Multi-nœuds Hadoop
4. Analyse de Données avec Spark

INSTALLATION DE DOCKER

- Mise à jour de l'index des paquets : Avant d'installer Docker, assurez-vous que l'index des paquets est à jour en exécutant la commande suivante :

```
$ sudo apt update
```

- Installation de Docker : Une fois l'index des paquets mis à jour, vous pouvez installer Docker en utilisant la commande `apt install`. Assurez-vous d'ajouter `sudo` pour exécuter la commande avec des privilèges administratifs :

```
$ sudo apt install docker.io
```

- Vérification de l'installation : Après l'installation, vous pouvez vérifier si Docker a été installé avec succès en exécutant la commande suivante pour afficher la version installée :

```
$ docker --version
```

```
firas@FirasKahlaoui:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
firas@FirasKahlaoui:~$ |
```

- Démarrage du service Docker : Docker ne démarre pas automatiquement après l'installation. Vous devez démarrer le service Docker en utilisant la commande suivante :

```
$ sudo systemctl start docker
```

- Vérification de l'état du service Docker : Vous pouvez vérifier l'état du service Docker pour vous assurer qu'il fonctionne correctement en exécutant la commande suivante :

```
$ sudo systemctl status docker
```

```
firas@FirasKahlaoui:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2024-04-24 17:30:46 CET; 5min ago
     TriggeredBy: ● docker.socket
        Docs: https://docs.docker.com
    Main PID: 263 (dockerd)
       Tasks: 21
      Memory: 125.0M
    CGroup: /system.slice/docker.service
            └─263 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

- Activation du démarrage automatique de Docker au démarrage du système (optionnel) :

```
$ sudo systemctl enable docker
```

- Ajouter l'utilisateur au groupe Docker : Utilisez la commande usermod pour ajouter votre utilisateur actuel au groupe Docker. Remplacez username par le nom de votre utilisateur :

```
$ sudo usermod -aG docker username
```

```
$ groups username
```

```
firas@FirasKahlaoui:~$ groups
firas adm dialout cdrom floppy sudo audio dip video plugdev netdev docker
firas@FirasKahlaoui:~$ |
```

TÉLÉCHARGEMENT D'IMAGES DOCKER ET DÉPLOIEMENT DE CONTENEURS

- Télécharger l'image docker uploadée sur dockerhub de Lilia Sfaxi:

```
$ docker pull liliasfaxi/hadoop-cluster:latest
```

- Créez un réseau pour interconnecter les trois conteneurs :

```
$ docker network create --driver=bridge hadoop
```

- Trois conteneurs sont créés et démarrés à l'aide de l'image Docker 'liliasfaxi/hadoop-cluster:latest', où 'hadoop-master' est configuré comme nœud principal avec les ports 9870, 8088, 7077 et 16010 exposés, tandis que 'hadoop-worker1' et 'hadoop-worker2' agissent comme nœuds de travail avec le port 8042 exposé, tous connectés au réseau 'hadoop'.

```
$ docker run -itd --net=hadoop -p 9870:9870 -p 8088:8088 -p 7077:7077 -p 16010:16010 --name hadoop-master --hostname hadoop-master liliasfaxi/hadoop-cluster:latest
```

```
$ docker run -itd -p 8040:8042 --net=hadoop --name hadoop-worker1 --hostname hadoop-worker1 liliasfaxi/hadoop-cluster:latest
```

```
$ docker run -itd -p 8041:8042 --net=hadoop --name hadoop-worker2 --hostname hadoop-worker2 liliasfaxi/hadoop-cluster:latest
```

```
firas@FirasKahlaoui:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker-hadoop_resourcemanager  latest            26b8b3f49b7d      6 days ago        1.37GB
docker-hadoop_historyserver    latest            067c5a74099d      6 days ago        1.37GB
docker-hadoop_nodemanager1     latest            20da93d06681      6 days ago        1.37GB
docker-hadoop_datanode1        latest            2be2f954b285      6 days ago        1.37GB
docker-hadoop_datanode2        latest            2be2f954b285      6 days ago        1.37GB
docker-hadoop_datanode3        latest            2be2f954b285      6 days ago        1.37GB
docker-hadoop_namenode         latest            501a8da25521      6 days ago        1.37GB
firas_image              latest            72ea37c3f331      3 weeks ago        5.11GB
liliasfaxi/hadoop-cluster      latest            34d8f3b55a11      3 months ago      4.89GB
apache/hadoop              3.3.3            7e0307af801f      10 months ago      1.04GB
bde2020/hadoop-base            2.0.0-hadoop3.2.1-java8  a89a06d383e8      4 years ago        1.37GB
firas@FirasKahlaoui:~$ |
```

- La liste des conteneurs Docker en cours d'exécution sur le système peut être consultée en utilisant la commande :

\$ docker ps

```
firas@FirasKahlaoui:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
8d406046cbb9   liliasfaxi/hadoop-cluster:latest    "sh -c 'service ssh _"  3 weeks ago   Up 5 seconds  0.0.0.0:8041→8042/tcp, :::8
041→8042/tcp
8b501bc99995   liliasfaxi/hadoop-cluster:latest    "sh -c 'service ssh _"  3 weeks ago   Up 5 seconds  0.0.0.0:8040→8042/tcp, :::8
040→8042/tcp
1b7d126d2c6b   liliasfaxi/hadoop-cluster:latest    "sh -c 'service ssh _"  3 weeks ago   Up 14 seconds 0.0.0.0:7077→7077/tcp, :::7
077→7077/tcp, 0.0.0.0:8088→8088/tcp, :::8088→8088/tcp, 0.0.0.0:9870→9870/tcp, :::9870→9870/tcp, 0.0.0.0:16010→16010/tcp, :::160
10→16010/tcp   hadoop-master
firas@FirasKahlaoui:~$ |
```

MAPREDUCE DANS UN CONTENEUR DOCKER

- Pour commencer avec MapReduce dans un conteneur Docker, nous allons d'abord copier les fichiers nécessaires dans le conteneur hadoop-master.
- Ces fichiers comprennent purchases.txt, qui sera utilisé comme source de données pour le travail MapReduce, ainsi que les scripts mapper.py et reducer.py pour effectuer le MapReduce.

```
$ docker cp [nom-du-fichier] [nom-du-conteneur]:[destination]
```

```
firas@FirasKahlaoui:~/Data$ ls
purchases.txt purchases.txt:Zone.Identifier
firas@FirasKahlaoui:~/Data$ docker cp purchases.txt hadoop-master:/Data/
Successfully copied 211MB to hadoop-master:/Data/
firas@FirasKahlaoui:~/Data$ |
```

```
firas@FirasKahlaoui:~/MapReduce$ ls
Mapper.py Mapper.py:Zone.Identifier Reducer.py Reducer.py:Zone.Identifier
firas@FirasKahlaoui:~/MapReduce$ docker cp Mapper.py hadoop-master:/mapper.py
Successfully copied 2.05kB to hadoop-master:/mapper.py
firas@FirasKahlaoui:~/MapReduce$ docker cp Reducer.py hadoop-master:/reducer.py
Successfully copied 2.56kB to hadoop-master:/reducer.py
firas@FirasKahlaoui:~/MapReduce$ |
```

- Pour accéder au shell du conteneur hadoop-master, vous pouvez utiliser la commande suivante :

```
$ docker exec -it hadoop-master /bin/bash
```

```
firas@FirasKahlaoui:~/MapReduce$ docker exec -it hadoop-master /bin/bash
root@hadoop-master:~# cd /
root@hadoop-master:~# ls
Data      Result    bin       dev       home      lib32     libx32    media     opt       reducer.py  run      srv      tmp      var
MapReduce Spark_Scala boot      etc       lib       lib64     mapper.py mnt       proc      root       sbin     sys     usr
root@hadoop-master:~# |
```


- Pour démarrer Hadoop dans le conteneur, utilisez la commande suivante :

```
# ./start-hadoop.sh
```

```
root@hadoop-master:~# ./start-hadoop.sh

Starting namenodes on [hadoop-master]
hadoop-master: Warning: Permanently added 'hadoop-master' (ED25519) to the list of known hosts.
hadoop-master: WARNING: HADOOP_NAMENODE_OPTS has been replaced by HDFS_NAMENODE_OPTS. Using value of HADOOP_NAMENODE_OPTS.
Starting datanodes
WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_DIR.
hadoop-worker1: Warning: Permanently added 'hadoop-worker1' (ED25519) to the list of known hosts.
hadoop-worker2: Warning: Permanently added 'hadoop-worker2' (ED25519) to the list of known hosts.
hadoop-worker1: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_DIR
.
hadoop-worker1: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
hadoop-worker2: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_DIR
.
hadoop-worker2: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
Starting secondary namenodes [hadoop-master]
hadoop-master: Warning: Permanently added 'hadoop-master' (ED25519) to the list of known hosts.
hadoop-master: WARNING: HADOOP_SECONDARYNAMENODE_OPTS has been replaced by HDFS_SECONDARYNAMENODE_OPTS. Using value of HADOOP_SECONDARYNAMENODE_OPTS.

Starting resourcemanager
Starting nodemanagers
hadoop-worker1: Warning: Permanently added 'hadoop-worker1' (ED25519) to the list of known hosts.
hadoop-worker2: Warning: Permanently added 'hadoop-worker2' (ED25519) to the list of known hosts.
```

- Pour tester les services Hadoop, utilisez la commande 'jps' dans le conteneur hadoop-master et hadoop-worker1 et hadoop-worker2

```
root@hadoop-master:~# jps
627 ResourceManager
404 SecondaryNameNode
938 Jps
172 NameNode
root@hadoop-master:~# |
```

```
firas@FirasKahlaoui:~/MapReduce$ docker exec -it hadoop-worker1 /bin/bash
root@hadoop-worker1:~# jps
197 NodeManager
346 Jps
76 DataNode
root@hadoop-worker1:~# |
```

```
firas@FirasKahlaoui:~/MapReduce$ docker exec -it hadoop-worker2 /bin/bash
root@hadoop-worker2:~# jps
197 NodeManager
346 Jps
76 DataNode
root@hadoop-worker2:~# |
```

- Maintenant tout fonctionne bien, tous les services Hadoop sont opérationnels. Nous pouvons démarrer le travail de MapReduce.
- commencer par créer un répertoire dans le système de fichiers Hadoop (HDFS):

```
# hadoop fs -mkdir /myInput
```

- Puis charger le fichier purchases.txt dans ce répertoire :

```
# hadoop fs -put purchases.txt /myInput/
```

```
root@hadoop-master:~# hadoop fs -mkdir /myInput
root@hadoop-master:~# cd
root@hadoop-master:~# cd /Data/
root@hadoop-master:/Data# ls
purchases.txt
root@hadoop-master:/Data# hadoop fs -put purchases.txt /myInput/
root@hadoop-master:/Data# |
```

- nous pouvons consulter le système de fichiers Hadoop HDFS en utilisant localhost:9870

Browse Directory

/myInput

Go!

Show

25

entries

Search:

<div><input type="checkbox"/></div>	<div>Permission</div>	<div>Owner</div>	<div>Group</div>	<div>Size</div>	<div>Last Modified</div>	<div>Replication</div>	<div>Block Size</div>	<div>Name</div>	
<div><input type="checkbox"/></div>	<div>-rw-r--r--</div>	<div>root</div>	<div>supergroup</div>	<div>201.52 MB</div>	<div>Apr 24 18:43</div>	<div>2</div>	<div>128 MB</div>	<div>purchases.txt</div>	<div><div></div></div>

Showing 1 to 1 of 1 entries

Previous

1

Next

- nous aurons besoin du hadoop streaming jar que vous pouvez trouver dans le répertoire suivant:

```
$HADOOP_HOME/share/hadoop/tools/lib/
```

- Maintenant, nous pouvons démarrer notre travail MapReduce avec la commande suivante:

```
# hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-file ./mapper.py -mapper "python3 mapper.py" \
-file ./reducer.py -reducer "python3 reducer.py" \
-input /myInput/purchases.txt \
-output /outResult
```

```
root@hadoop-master:/# hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -file ./mapper.py -mapper "python3 mapper.py" -file ./reducer.py -reducer "python3 reducer.py" -input /myInput/purchases.txt -output /outResult
2024-04-24 17:58:18,090 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar40093230841304244/] [] /tmp/streamjob8149355113720878529.jar tmpDir=null
2024-04-24 17:58:19,755 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at hadoop-master/172.18.0.2:8032
2024-04-24 17:58:20,118 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at hadoop-master/172.18.0.2:8032
2024-04-24 17:58:20,600 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1713979427978_0001
2024-04-24 17:58:21,863 INFO mapred.FileInputFormat: Total input files to process : 1
2024-04-24 17:58:21,924 INFO net.NetworkTopology: Adding a new node: /default-rack/172.18.0.4:9866
2024-04-24 17:58:21,927 INFO net.NetworkTopology: Adding a new node: /default-rack/172.18.0.3:9866
2024-04-24 17:58:22,016 INFO mapreduce.JobSubmitter: number of splits:2
2024-04-24 17:58:22,292 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1713979427978_0001
2024-04-24 17:58:22,296 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-04-24 17:58:22,577 INFO conf.Configuration: resource-types.xml not found
2024-04-24 17:58:22,578 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-04-24 17:58:23,477 INFO impl.YarnClientImpl: Submitted application application_1713979427978_0001
2024-04-24 17:58:23,581 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1713979427978_0001/
2024-04-24 17:58:23,590 INFO mapreduce.Job: Running job: job_1713979427978_0001
2024-04-24 17:58:38,031 INFO mapreduce.Job: Job job_1713979427978_0001 running in uber mode : false
2024-04-24 17:58:38,042 INFO mapreduce.Job: map 0% reduce 0%
2024-04-24 17:58:58,369 INFO mapreduce.Job: map 67% reduce 0%
2024-04-24 17:58:59,420 INFO mapreduce.Job: map 100% reduce 0%
2024-04-24 17:59:15,536 INFO mapreduce.Job: map 100% reduce 100%
2024-04-24 17:59:15,561 INFO mapreduce.Job: Job job_1713979427978_0001 completed successfully
```

```
2024-04-24 17:59:15,712 INFO streaming.StreamJob: Output directory: /outResult
root@hadoop-master:/# |
```

- Après que le travail MapReduce a été terminé avec succès, je peux maintenant obtenir les fichiers de sortie en utilisant cette commande :

```
# hadoop fs -get /outResult .
```

```
root@hadoop-master:/# ls
Data      Result   bin      dev      home     lib32    libx32   media    opt       proc      root     sbin     sys      usr
MapReduce Spark_Scala boot     etc      lib      lib64    mapper.py mnt      outResult reducer.py run      srv      tmp      var
root@hadoop-master:/# cd outResult/
root@hadoop-master:/outResult# ls
_SUCCESS part-000000
root@hadoop-master:/outResult# |
```

- Le résultat du job MapReduce se trouve dans le fichier part-00000. Nous pouvons le visualiser en utilisant la commande suivante:

```
# vi part-00000
```

```
Albuquerque      10052311.419999966
Anaheim          10076416.359999955
Anchorage        9933500.400000004
Arlington        10072207.970000006
Atlanta          9997146.700000083
Aurora           9992970.91999996
Austin           10057158.900000017
Bakersfield      10031208.920000002
Baltimore        10096521.450000018
Baton Rouge      10131273.229999963
Birmingham       10076606.520000014
Boise            10039166.74
Boston           10039473.280000059
```

ERREURS RENCONTRÉES

- Lors de l'exécution du job MapReduce, il y avait un problème dans le fichier mapred-site.xml qui a provoqué l'échec de la commande:

```
root@hadoop-master:/# hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -file ./mapper.py -mapper "python3 ma
pper.py" -file ./reducer.py -reducer "python3 reducer.py" -input /myInput/purchases.txt -output /outResult1
2024-04-24 18:30:30,180 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
2024-04-24 18:30:30,429 ERROR conf.Configuration: error parsing conf mapred-site.xml
```

SOLUTION



- La solution a été de reconfigurer le fichier mapred-site.xml:

```
# cd $HADOOP_CONF_DIR/
# vi mapred-site.xml
```

ANALYSE DE DONNÉES AVEC SPARK

- Commencez par démarrer Hadoop et vérifiez que tous les services fonctionnent correctement dans les conteneurs (cluster Hadoop):

```
root@hadoop-master:~# jps
627 ResourceManager
404 SecondaryNameNode
938 Jps
172 NameNode
root@hadoop-master:~#
```

- Ensuite, démarrez Spark en utilisant la commande :

```
# spark-shell
```

```
root@hadoop-master:~# spark-shell  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
24/04/24 19:16:35 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.  
Spark context Web UI available at http://hadoop-master:4040  
Spark context available as 'sc' (master = yarn, app id = application_1713986122086_0001).  
Spark session available as 'spark'.  
Welcome to
```



```
 version 3.5.0  
  
Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 1.8.0_392)  
Type in expressions to have them evaluated.  
Type :help for more information.
```

```
scala>
```

- Nous allons effectuer notre analyse sur l'ensemble de données des Salaires des Ingénieurs: Tendances et Facteurs Impactants (2020-2024) provenant de Kaggle.

Dataset Link

LES COLONNES DE DATASET

- **work_year**: L'année au cours de laquelle les données salariales ont été collectées (par exemple, 2024).
- **experience_level**: Le niveau d'expérience de l'employé (par exemple, MI pour niveau intermédiaire).
- **employment_type**: Le type d'emploi (par exemple, TP pour temps plein). **job_title**: Le titre du poste (par exemple, Data Scientist).
- **salary**: Le montant du salaire. **salary_currency**: La devise dans laquelle le salaire est libellé (par exemple, USD pour dollars américains).
- **salary_in_usd**: Le montant du salaire converti en dollars américains.
- **employee_residence**: Le pays de résidence de l'employé (par exemple, AU pour Australie).
- **remote_ratio**: Le ratio indiquant le niveau de travail à distance (0 pour aucun travail à distance).
- **company_location**: L'emplacement de l'entreprise (par exemple, AU pour Australie).
- **company_size**: La taille de l'entreprise (par exemple, S pour petite).

LES KPIS

- **KPIs pour le salaire :**

- Moyenne des salaires par année de travail.

- Écart type des salaires pour évaluer la dispersion des données salariales.

- **Salaire par Niveau d'Expérience** : Comparer les salaires selon les différents niveaux d'expérience.
- **Salaire par Titre de Poste** : Explorer les différences de salaires en fonction des titres de poste.
- **Salaire par Type d'Emploi** : Comparer les salaires entre les emplois à temps plein, à temps partiel, contractuels, etc.
- **Salaire par Taille de l'Entreprise** : Analyser comment la taille de l'entreprise affecte les salaires.
- **Impact du Télétravail sur le Salaire** : Étudier si les arrangements de télétravail affectent les niveaux de salaire.


```
scala> val filePath = "file:///home/firas/salaries.csv"
```

```
scala> val filePath = "file:///home/firas/salaries.csv"
filePath: String = file:///home/firas/salaries.csv
```

- Maintenant, voici la commande pour lire un fichier CSV depuis HDFS :

```
scala> val df = spark.read.option("header", "true").csv(filePath)
```

```
scala> val df = spark.read.option("header", "true").csv(filePath)
df: org.apache.spark.sql.DataFrame = [work_year: string, experience_level: string ... 9 more fields]
```

- Moyenne des salaires par année de travail:

```
scala> val averageSalaryByYear = df.groupBy("work_year").agg(avg("salary").alias("average_salary"))
```

```
scala> val averageSalaryByYear = df.groupBy("work_year").agg(avg("salary").alias("average_salary"))
averageSalaryByYear: org.apache.spark.sql.DataFrame = [work_year: string, average_salary: double]

scala> averageSalaryByYear.show()
+-----+-----+
|work_year|average_salary|
+-----+-----+
|2020|356637.4533333333|
|2022|167705.96193353474|
|2023|159319.8009156004|
|2021|559350.9908256881|
|2024|152319.73718267796|
+-----+-----+
```

- Écart type des salaires pour évaluer la dispersion des données salariales :

```
scala> val salaryStdDev = df.agg(stddev("salary").alias("salary_std_dev"))
```

```
scala> val salaryStdDev = df.agg(stddev("salary")).alias("salary_std_dev"))
salaryStdDev: org.apache.spark.sql.DataFrame = [salary_std_dev: double]

scala> salaryStdDev.show()
+-----+
| salary_std_dev |
+-----+
| 340601.7023392354 |
+-----+
```

- En moyenne, les salaires ont varié considérablement au fil des années, avec une moyenne de 356 637 \$ en 2020, en baisse à 152 319 \$ en 2024. De plus, l'écart-type des salaires, qui mesure la dispersion des données autour de la moyenne, est évalué à 340 601 \$, indiquant une certaine volatilité dans les niveaux de rémunération des ingénieurs au cours de cette période.
- Salaire par Niveau d'Expérience : Comparer les salaires selon les différents niveaux d'expérience.

```
scala> val salaryByExperience = df.groupBy("experience_level").agg(avg("salary"))
```

```
scala> val salaryByExperience = df.groupBy("experience_level").agg(avg("salary"))
salaryByExperience: org.apache.spark.sql.DataFrame = [experience_level: string, avg(salary): double]

scala> salaryByExperience.show()
+-----+-----+
|experience_level| avg(salary)|
+-----+-----+
| EX | 208196.86573146292 |
| MI | 157148.50944333995 |
| EN | 126019.35102350265 |
| SE | 168892.2990987608 |
+-----+-----+
```

- Ces résultats suggèrent que le niveau d'expérience est un facteur clé dans la détermination des niveaux de rémunération, avec une augmentation générale des salaires à mesure que l'expérience augmente.

- Salaire par Titre de Poste : Explorer les différences de salaires en fonction des titres de poste.

```
scala> val salaryByJobTitle =
df.groupBy("job_title").agg(avg("salary").alias("average_salary")).orderBy(desc("average_salary"))
```

```
scala> val salaryByJobTitle = df.groupBy("job_title").agg(avg("salary").alias("average_salary")).orderBy(desc("average_salary"))
salaryByJobTitle: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [job_title: string, average_salary: double]

scala> salaryByJobTitle.show()
+-----+-----+
| job_title | average_salary |
+-----+-----+
|Principal Data Architect| 3000000.0|
|Lead Machine Learning Engineer| 1566200.0|
|Manager Data Manager| 1562500.0|
|Head of Machine Learning| 1146000.0|
|Lead Data Analyst| 1095833.3333333333|
|Lead Data Scientist| 771578.9090909091|
|AI Programmer| 764400.8571428572|
|BI Data Analyst| 645526.3157894737|
|Head of Data Science| 584630.4166666666|
|Applied Machine Learning| 554842.8571428572|
|Big Data Engineer| 492300.0|
|Data Science Technician| 375000.0|
|AI Research Engineer| 360871.4285714286|
|Machine Learning Engineer| 359653.3333333333|
|Data Science Manager| 338386.01639344264|
|Analytics Engineer| 325000.0|
|NLP Engineer| 316713.3333333333|
|Product Data Analyst| 291700.0|
|AI Software Engineer| 287640.0|
|Managing Director| 280000.0|
+-----+-----+
only showing top 20 rows
```

- Cette analyse des salaires par titre de poste révèle une distribution variée des rémunérations, avec des salaires moyens atteignant jusqu'à 3 000 000 \$ pour un poste de "Principal Data Architect". Les postes de direction et les rôles spécialisés dans les domaines de l'intelligence artificielle et de l'apprentissage automatique occupent les premières positions en termes de rémunération moyenne.
- Salaire par Type d'Emploi : Comparer les salaires entre les emplois à temps plein, à temps partiel, contractuels, etc.

```
scala> val salaryByEmploymentType = df.groupBy("employment_type").agg(avg("salary"))
```

```
scala> val salaryByEmploymentType = df.groupBy("employment_type").agg(avg("salary"))
salaryByEmploymentType: org.apache.spark.sql.DataFrame = [employment_type: string, avg(salary): double]

scala> salaryByEmploymentType.show()
+-----+-----+
|employment_type|    avg(salary)|
+-----+-----+
|      FT      |163768.6581576703|
|      PT      |109374.36842105263|
|      CT      |115664.14285714286|
|      FL      | 430182.5|
+-----+-----+
```

- Les employés à temps plein (FT) ont un salaire moyen de 163 768 \$, tandis que ceux à temps partiel (PT) gagnent en moyenne 109 374 \$. Les travailleurs contractuels (CT) ont un salaire moyen légèrement plus élevé, à 115 664 \$. En revanche, les employés bénéficiant de contrats de type freelance (FL) affichent un salaire moyen beaucoup plus élevé, à 430 182 \$. Ces résultats soulignent l'impact du type d'emploi sur les niveaux de rémunération, avec des écarts significatifs entre les différents types de travailleurs.
- Salaire par Taille de l'Entreprise : Analyser comment la taille de l'entreprise affecte les salaires.
 - Groupez les salaires par taille de l'entreprise (petite, moyenne, grande) et examinez les différences.

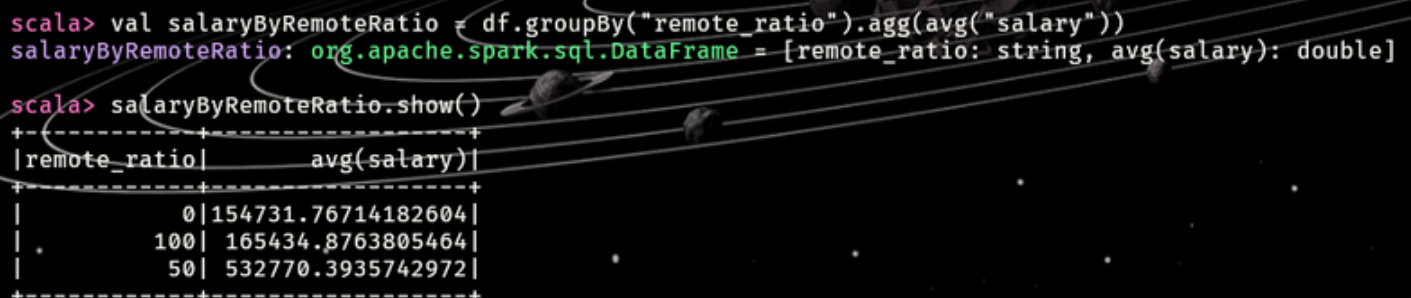
```
scala> val salaryByCompanySize = df.groupBy("company_size").agg(avg("salary"))
```

```
scala> val salaryByCompanySize = df.groupBy("company_size").agg(avg("salary"))
salaryByCompanySize: org.apache.spark.sql.DataFrame = [company_size: string, avg(salary): double]

scala> salaryByCompanySize.show()
+-----+-----+
|company_size|    avg(salary)|
+-----+-----+
|      M      |152453.491092481|
|      L      |308653.12427745666|
|      S      |284437.69680851063|
+-----+-----+
```

- Cette analyse des salaires, catégorisée par taille d'entreprise, révèle des différences marquées dans les rémunérations moyennes. Les entreprises de taille moyenne (M) offrent un salaire moyen de 152 453 \$, tandis que les entreprises de petite taille (S) et grande taille (L) offrent des salaires moyens plus élevés, à 284 437 \$ et 308 653 \$ respectivement. Ces résultats indiquent une corrélation entre la taille de l'entreprise et les niveaux de rémunération, avec les entreprises de plus grande taille ayant tendance à offrir des salaires plus élevés en moyenne.
- Impact du Télétravail sur le Salaire : Étudier si les arrangements de télétravail affectent les niveaux de salaire.
 - Comparez les salaires des employés travaillant entièrement à distance avec ceux des employés travaillant sur site, en calculant les moyennes ou médianes pour chaque groupe.

```
scala> val salaryByRemoteRatio = df.groupBy("remote_ratio").agg(avg("salary"))
```



```
scala> val salaryByRemoteRatio = df.groupBy("remote_ratio").agg(avg("salary"))
salaryByRemoteRatio: org.apache.spark.sql.DataFrame = [remote_ratio: string, avg(salary): double]
scala> salaryByRemoteRatio.show()
+-----+-----+
|remote_ratio|    avg(salary)|
+-----+-----+
|          0|154731.76714182604|
|         100|165434.8763805464|
|          50|532770.3935742972|
+-----+-----+
```

- Pour les employés travaillant exclusivement sur site (ratio de travail à distance de 0), le salaire moyen s'élève à 154 731 \$. En revanche, pour ceux travaillant à distance à plein temps (ratio de travail à distance de 100), le salaire moyen atteint 165 434 \$. Une différence notable se présente pour les travailleurs bénéficiant d'un ratio de travail à distance de 50, avec un salaire moyen considérablement plus élevé de 532 770 \$. Ces résultats suggèrent que le niveau de travail à distance peut influencer de manière significative les niveaux de rémunération.

LES ENSEIGNEMENTS TIRÉS DE CE PROJET

1- INSTALLATION DE DOCKER :

- Familiarisation avec les principales commandes Docker
- Apprentissage des concepts de base de la virtualisation et de la gestion des conteneurs.
- Acquisition de compétences en gestion des images et des conteneurs Docker.

2- TÉLÉCHARGEMENT D'IMAGES DOCKER ET DÉPLOIEMENT DE CONTENEURS :

- Compréhension du processus de recherche, téléchargement et utilisation d'images Docker depuis des registres publics.
- Pratique du déploiement et de la gestion de conteneurs pour exécuter des applications isolées dans des environnements contrôlés.

3- MAPREDUCE DANS UN CLUSTER MULTI-NŒUDS HADOOP :

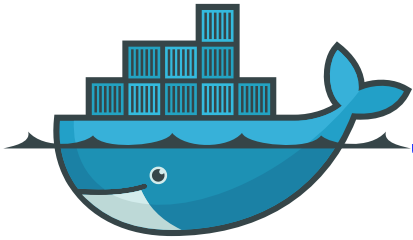
- Apprentissage des principes et du fonctionnement de MapReduce pour le traitement distribué des données.
- Développement de compétences en programmation MapReduce pour la manipulation efficace de grands ensembles de données.

4- ANALYSE DE DONNÉES AVEC SPARK :

- Compréhension des concepts clés de traitement de données distribuées avec Apache Spark.
- Expérimentation avec les fonctionnalités de Spark pour l'analyse, le traitement et la manipulation de données massives.

»» Ces enseignements fournissent une base solide pour la compréhension et la maîtrise des technologies Docker, Hadoop et Spark, ainsi que pour leur utilisation dans des projets d'analyse de données complexes.

HADOOP CLUSTER AVEC DOCKER



docker

