

Manipulation d'Apache Spark en utilisant un cluster de containers Docker

**Binôme : - Imane TOUIBA
- Zahra KASMOUTI**

I. Lancement du cluster Spark avec Hadoop (constitué d'un Namenode et de deux datanodes) :

1) Téléchargement des fichiers nécessaires :

Nous avons d'abord téléchargé le répertoire hadoop-main qui contient tous les fichiers nécessaires pour la configuration d'un cluster Hadoop et Spark à partir du lien : <https://gitlab.com/dounia.zaidouni/hadoop>.

2) Lancement des conteneurs :

Nous avons ensuite lancé les conteneurs en effectuant ces deux commandes :

```
$ cd hadoop-main/
```

```
$ docker-compose -f docker-compose_spark.yml up -d
```

```
imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker-compose -f docker-compose_spark.yml up -d
Pulling spark-master (douniazaidouni/spark-master:1.0.0)...
1.0.0: Pulling from douniazaidouni/spark-master
396c31837116: Pulling fs layer
34836d4b51d9: Downloading [=====] 396c31837116: Downloading [>-----]
7116: Downloading [==> 396c31837116: Downloading [=====] 396c31837116: Do
wnloading [=====] 396c31837116: Downloading [=====] 396c31837116: ]
956.7kB/2.798MB
396c31837116: Downloading [=====] 396c31837116: Downloading [=====] 396c31837116: Do
wnloading [=====] 396c31837116: Downloading [=====] 396c31837116: Do
wnloading [=====] 396c31837116: Downloading [=====] 396c31837116: ]
2.308MB/2.798MB

396c31837116: Downloading [=====] 396c31837116: Pull complete
34836d4b51d9: Pull complete
791e39ec18d2: Pull complete
86080a9fa15b: Pull complete
11a9824b74a2: Pull complete
b4ccb230fbdc: Pull complete
Digest: sha256:d32792a8efb29b89d059f886e62016e8c2a4bc83f6c6b3bb3dda784d9fe74540
Status: Downloaded newer image for douniazaidouni/spark-master:1.0.0
Pulling spark-worker-1 (douniazaidouni/spark-worker:1.0.0)...
1.0.0: Pulling from douniazaidouni/spark-worker
396c31837116: Already exists
34836d4b51d9: Already exists
791e39ec18d2: Already exists
1d1369a6fd16: Already exists
86080a9fa15b: Already exists
11a9824b74a2: Already exists
284dd9d654e3: Pull complete
Digest: sha256:e7521243560e63654d653f7f0a6b883243689f26a44098c8d4a55b367ee67ceb
```

```

284dd9d654e3: Pull complete
Digest: sha256:e7521243560e63654d653f7f0a6b883243689f26a44098c8d4a55b367ee67ceb
Status: Downloaded newer image for douniazaidouni/spark-worker:1.0.0
historyserver is up-to-date
resourcemanager is up-to-date
datanode2 is up-to-date
namenode is up-to-date
nodemanager1 is up-to-date
datanode1 is up-to-date
nodemanager2 is up-to-date
Creating spark-master ... done
Creating spark-worker-2 ... done
Creating spark-worker-1 ... done
imane@imane-VirtualBox:~/Downloads/hadoop-main$ █

```

3) Accès au conteneur spark-master :

Nous avons accédé au container du namenode de notre cluster Spark avec Hadoop en lançant la commande suivante :

\$ docker exec -it spark-master /bin/bash

```

imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it spark-master /bin/bash
bash-5.0# ls
bin etc          finish-step.sh  lib    master.sh  mnt   proc   run   spark  sys  usr  wait-for-step.sh
dev execute-step.sh  home        lib64   media     opt   root   sbin  srv   tmp   var
bash-5.0# █

```

Nous avions également testé l'accès au terminal Scala et python à partir du namenode :

- Terminal Scala : # cd spark/ et # ./bin/spark-shell

```

imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it spark-master /bin/bash
bash-5.0# ls
bin etc          finish-step.sh  lib    master.sh  mnt   proc   run   spark  sys  usr  wait-for-step.sh
dev execute-step.sh  home        lib64   media     opt   root   sbin  srv   tmp   var
bash-5.0# cd spark/
bash-5.0# ./bin/spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/01/03 12:34:50 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using built-in
asses where applicable
Spark context Web UI available at http://900a82d7cab9:4040
Spark context available as 'sc' (master = local[*], app id = local-1735907697202).
Spark session available as 'spark'.
Welcome to

      ____/ \
     / \ \ \ \ \
     \ \ \ \ \ \ \
      / \ \ \ \ \
     / \ / \ / \ / \
     /_ /_ /_ /_ /_ \_ \
                           version 3.3.0

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_275)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

```

Pour quitter le terminal Scala nous utilisons la commande quit

- Terminal Python : # ./bin/pyspark

```

bash-5.0# ./bin/pyspark
Python 3.7.10 (default, Mar 2 2021, 09:06:08)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/01/03 12:36:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using built-in Java classes where applicable
Welcome to
    ____/
   / \ \_ \ \_ \ \_ \ \_ \ \_ \
  /_ / . _ \_, _ / / _ \ \_ \
 /_/
Using Python version 3.7.10 (default, Mar 2 2021 09:06:08)
Spark context Web UI available at http://900a82d7cab9:4040
Spark context available as 'sc' (master = local[*], app id = local-1735907817040).
SparkSession available as 'spark'.
>>>

```

Pour quitter le terminal Scala nous utilisons la commande exit()

II. Manipulation des RDDs en utilisant le terminal pyspark en local :

1) Enregistrement et chargement des Textfile :

Tout d'abord nous avons créé un fichier : ValeursINPT.txt dans le répertoire : /spark en utilisant l'outil vi : bash-5.0# vi ValeursINPT.txt

Ensuite nous avons accédé au terminal Python pour pouvoir profiter des méthodes fournis par la classe SparkContext :

- Création du RDD à partir de ValeursINPT.txt :

```
>>> mydata = sc.textFile("file:/usr/local/spark/ValeursINPT.txt")
```

- Compte du nombre de lignes dans la RDD :

```
>>> mydata.count()
```

```

Using Python version 3.7.10 (default, Mar 2 2021 09:06:08)
Spark context Web UI available at http://900a82d7cab9:4040
Spark context available as 'sc' (master = local[*], app id = local-1735908164653).
SparkSession available as 'spark'.
>>>
>>> mydata = sc.textFile("file:/spark/ValeursINPT.txt")
>>> for line in mydata.collect():
...     print(line)
...
Nos valeurs à l INPT sont :
Numérique par nature.
Renouvellement permanent.
Innovation et entrepreneuriat.
Ouverture sur l écosystème.
>>> mydata.count()
6
>>> █

```

- Filtrage de RDD :

```
>>> mydata_filt = mydata.filter(lambda s: s.startswith('N'))
```

- Sauvegarde de la RDD filtrée sous forme d'un fichier .txt :

```

>>> mydata_filt.saveAsTextFile("/spark/values_starts_withN")
- Compte du nombre de lignes de RDD filtrée :
>>> mydata_filt.count()
>>>
>>> mydata_filt = mydata.filter(lambda s: s.startswith('N'))
>>> mydata_filt.saveAsTextFile("/spark/values_starts_withN")
>>> mydata.count()
6
>>> mydata_filt.count()
2
>>>

```

NOTE : dans le répertoire /spark un répertoire nommé values_starts_withN est créé suite à la transformation du filtrage, ce répertoire contient le fichier part-00000 qui contient les lignes qui commencent par N.

```

bash-5.0# ls
LICENSE   R           RELEASE      bin    data    jars    licenses  python  values_starts_withN
NOTICE   README.md  ValeursINPT.txt  conf  examples  kubernetes  logs    sbin    yarn

```

2) Enregistrement et chargement des SequenceFiles :

- Création d'une RDD à partir d'une séquence :
- >>> rdd = sc.parallelize(range(1, 4)).map(lambda x: (x, "a" * x))
- Sauvegarde de RDD en tant que SequenceFile :
- >>> rdd.saveAsSequenceFile("/spark/fileseq1")
- Tri et affichage des éléments du SequenceFile :
- >>> sorted(sc.sequenceFile("/spark/fileseq1").collect())

```

Spark context available as 'sc' (master = local[*], app id = local-17359085061
SparkSession available as 'spark'.
>>> rdd = sc.parallelize(range(1, 4)).map(lambda x: (x, "a" * x))
>>> rdd.saveAsSequenceFile("/spark/fileseq1")
>>> sorted(sc.sequenceFile("/spark/fileseq1").collect())
[(1, 'a'), (2, 'aa'), (3, 'aaa')]
>>>

```

Notons que dans le répertoire /spark un répertoire nommé : fileseq1 est créé, ce répertoire doit contenir les fichiers :

```

bash-5.0# cd fileseq1/
bash-5.0# ls
_SUCCESS  part-00000
bash-5.0#

```

3) Utilisation d'une fonction nommée :

Tout d'abord, nous avons définit une fonction qui retourne la majuscule d'une string :

```

>>> def toUpper(s):
...     return s.upper()
...
Nous avons ensuite utiliser la méthode map pour appliquer la fonction
toUpper() à chaque ligne de notre RDD :
>>> mydata = sc.textFile("/spark/ValeursINPT.txt")
>>> mydataupper = mydata.map(toUpper)
>>> for line in mydataupper.collect():
...     print(line)
Spark context available as 'sc' (master = local[*], app id = local-1735908735282).
SparkSession available as 'spark'.
>>> def toUpper(s):
...     return s.upper()
...
>>> mydata = sc.textFile("/spark/ValeursINPT.txt")
>>> mydataupper = mydata.map(toUpper)
>>> for line in mydataupper.collect():
...     print(line)
...
NOS VALEURS A L INPT SONT :
NUMERIQUE PAR NATURE.
RENOUVELLEMENT PERMANENT.
INNOVATION ET ENTREPRENEURIAT.
OUVERTURE SUR L ECOSYSTEME.
>>>

```

4) Utilisation d'une fonction anonyme :

Tout d'abord, nous avons copié le fichier poeme.txt du local vers le conteneur spark-master en lançant la commande : (dans le terminal de la machine hôte)

```
$ docker cp /home/zaidouni/Documents/hadoop-main/poeme.txt
spark-master:/
```

Ensuite nous avons relancé le conteneur spark-master puis accéder au terminal python :

- création de RDD à partir de poeme.txt :
- >>> mydata = sc.textFile("/poeme.txt")
- Utilisation d'une fonction anonyme pour transformer la RDD :
- >>> mydata_upper = mydata.map(lambda line:
line.upper()).take(5)
- Affichage de la RDD transformée :
- >>> for line in mydata_upper:
... print(line)

```

SparkSession available as 'spark'.
>>> mydata = sc.textFile("/poeme.txt")
>>> mydata_upper = mydata.map(lambda line: line.upper()).take(5)
>>> for line in mydata_upper:
...     print(line)
...
CELUI QUI CROYAIT AU CIEL
CELUI QUI NY CROYAIT PAS
TOUS DEUX ADORAIENT LA BELLE
PRISONNIERE DES SOLDATS
LEQUEL MONTAIT A LECHELLE
>>>

```

5) Utilisation de « parallelize » :

Dans cette section nous avons utilisé `parallelize()` afin de transformer une liste de nombres entiers en une RDD. Ensuite nous avons utilisé l'action `reduce()` pour compter la somme des nombres entiers :

```

>>> data = [10, 20, 30, 40, 50, 100, 250]
>>> distData = sc.parallelize(data)
>>> total = distData.reduce(lambda a,b: a + b)
>>> print (total)
500
>>> █

```

6) Utilisation de « wholeTextFiles » :

Tout d'abord dans le répertoire `/spark`, nous avons créé un répertoire nommé : `json_files` (en utilisant la commande : `mkdir json_files`). Ensuite nous avons créé deux fichier `file1.json` et `file2.json` dans ce répertoire.

- Création d'une RDD à travers le répertoire `json_files` :


```
>>> myrdd1 = sc.wholeTextFiles("/spark/json_files")
```
- Transformation de la RDD avec une fonction anonyme :


```
>>> myrdd2 = myrdd1.map(lambda a: json.loads(a[1]))
```
- Affichage des éléments de la RDD transformée :


```
>>> for record in myrdd2.take(2):
...     print(record.get("firstName",None))
...
Fred
Barney
```

```
SparkSession available as 'spark'.
>>> import json
>>> myrdd1 = sc.wholeTextFiles("/spark/json_files")
>>> myrdd2 = myrdd1.map(lambda a: json.loads(a[1]))
>>> for record in myrdd2.take(2):
...     print(record.get("firstName",None))
...
[Stage 0:>
[Stage 0:=====
    Fred
Barney
```

7) Utilisation de « flatMap » et « distinct »:

Dans cette section nous avons utilisé flatMap pour transformer une RDD dont les lignes sont les lignes de poeme.txt en une RDD dont les lignes sont les mots de poeme.tx :

```
>>> mydata = sc.textFile("/poeme.txt")
>>> mynewdata = mydata.flatMap(lambda line: line.split(' ')).distinct()
>>> for line in mynewdata.collect():
...     print(line)
>>> mydata = sc.textFile("/poeme.txt")
>>> mynewdata = mydata.flatMap(lambda line: line.split(' ')).distinct()
>>> for line in mynewdata.collect():
...     print(line)
...
[Stage 1:>
[Stage 2:>
    celui
    qui
    croyait
    au
    ciel
    ny
    pas
    tous
    deux
    adoraient
    la
    belle
    prisonniere
    des
    soldats
    lequel
    montait
    a
    lechelle
    et
    guettait
```

8) Utilisation de « subtract » et « zip »:

- Utilisation de subtract() :

```
>>> mydata = ["Chicago", "Boston", "Paris", "San Francisco",
    "Tokyo"]
>>> rdd1 = sc.parallelize(mydata)
>>> data = ["San Francisco", "Boston", "Amsterdam", "Mumbai",
    "McMurdo Station"]
>>> rdd2 = sc.parallelize(data)
>>> newrdd = rdd1.subtract(rdd2)
>>> for line in newrdd.collect():
...     print(line)
...
>>> mydata = ["Chicago", "Boston", "Paris", "San Francisco", "Tokyo"]
>>> dd1 = sc.parallelize(mydata)
>>> rdd1 = sc.parallelize(mydata)
>>> mydata = ["Chicago", "Boston", "Paris", "San Francisco", "Tokyo"]
>>> rdd1 = sc.parallelize(mydata)
>>> data = ["San Francisco", "Boston", "Amsterdam", "Mumbai", "McMurdo Stat
>>> rdd2 = sc.parallelize(data)
>>> newrdd = rdd1.subtract(rdd2)
>>> for line in newrdd.collect():
...     print(line)
...
[Stage 3:<=====] [Stage 4:>
Chicago
Paris
Tokyo
>>>
```

- Utilisation de zip() :

```
>>> ziprdd = rdd1.zip(rdd2)
>>> for line in ziprdd.collect():
...     print(line)
...

```

```
>>> ziprdd = rdd1.zip(rdd2)
>>> for line in ziprdd.collect():
...     print(line)
...
('Chicago', 'San Francisco')
('Boston', 'Boston')
('Paris', 'Amsterdam')
('San Francisco', 'Mumbai')
('Tokyo', 'McMurdo Station')
>>>
```

9) Utilisation de intersection et union :

```
>>> interdd = rdd1.intersection(rdd2)
>>> for line in interdd.collect():
...     print(line)
...
San Francisco
Boston
>>> unionrdd = rdd1.union(rdd2)
>>> for line in unionrdd.collect():
...     print(line)
...
Chicago
Boston
Paris
San Francisco
Tokyo
San Francisco
Boston
Amsterdam
Mumbai
McMurdo Station
>>> █
```

III. Exécution du programme « Word Count » en utilisant le terminal interactif Scala et Python :

1) Dépôt du poeme.txt dans le HDFS :

Tout d'abord, il faut copier le fichier poeme.txt dans le container du namenode en tapant :

```
$ docker cp /home/zaidouni/Documents/hadoop-main/poeme.txt namenode:/
```

- Accéder au conteneur du namenode :
\$ docker exec -it namenode /bin/bash
- Copiez le fichier d'entrée dans HDFS :
\$ hdfs dfs -put poeme.txt /
\$ hdfs dfs -ls /

```
imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it namenode /bin/bash
root@79c2d3facca6:/# hdfs dfs -ls /
Found 6 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
-rw-r--r--  3 root supergroup 1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup  0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup  0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup  0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup  0 2025-01-03 11:58 /tmp
root@79c2d3facca6:/#
```

2) Exécution du code du « Word Count » :

Pour exécuter le programme du « Word Count » nous avons utilisé le terminal spark-shell, en tapant le code suivant dans le terminal scala :

```
scala> val lines = sc.textFile("hdfs://namenode:9000/poeme.txt")
scala> val words = lines.flatMap(_.split("\\s+"))
scala> val wc = words.map(w => (w, 1)).reduceByKey(_ + _)
scala> wc.saveAsTextFile("hdfs://namenode:9000/file1.count")
```

```
scala> val lines = sc.textFile("hdfs://namenode:9000/poeme.txt")
lines: org.apache.spark.rdd.RDD[String] = hdfs://namenode:9000/poeme.txt MapPartitionsRDD[1] at textFile at <console>:23

scala> val words = lines.flatMap(_.split("\\s+"))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:23

scala> val wc = words.map(w => (w, 1)).reduceByKey(_ + _)
wc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:23

scala> wc.saveAsTextFile("hdfs://namenode:9000/file1.count")
```

- Nous avons accédé au container du namenode et affiché le contenu du HDFS :
\$ docker exec -it namenode /bin/bash
\$ hdfs dfs -ls /

```

imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it namenode /bin/bash
root@79c2d3facca6:/# hdfs dfs -ls /
Found 6 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
-rw-r--r--  3 root supergroup 1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup  0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup  0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup  0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup  0 2025-01-03 11:58 /tmp
root@79c2d3facca6:/#

```

- Nous avons ensuite afficher le contenu du fichier créé dans le répertoire file1.count :

```

root@79c2d3facca6:/# hdfs dfs -cat /file1.count/part-00000
(jura,1)
(ils,1)
(violoncelle,1)
(comment,1)
(rose,1)
(soldats,1)
(que,2)
(celui,20)
(levres,1)
(bretagne,1)
(ses,1)

```

Exécution du programme Word Count en utilisant le terminal python :

```

Spark context available as 'sc' (master = local[*], app id = local-1736371864873).
SparkSession available as 'spark'.
>>> words = sc.textFile("hdfs://namenode:9000/poeme.txt").flatMap(lambda line: line.split(" "))
>>> wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)
>>> wordCounts.saveAsTextFile("hdfs://namenode:9000/file2.count")
>>>

```

NOTE : répertoire file2.count contenant un fichier part-00000 est créé dans HDFS :

```
imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it namenode /bin/bash
root@79c2d3facca6:/# hdfs dfs -ls /
Found 8 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
drwxr-xr-x  - root supergroup    0 2025-01-08 21:26 /file1.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:31 /file2.count
-rw-r--r--  3 root supergroup   1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup    0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup    0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup    0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup    0 2025-01-03 11:58 /tmp
root@79c2d3facca6:/# hdfs dfs -cat /file2.count/part-00000
('celui', 20)
('qui', 25)
('croyait', 20)
('au', 11)
('ciel', 10)
('ny', 10)
('pas', 11)
('tous', 3)
('deux', 6)
('l', 1)
```

IV. Exécution du programme « Word Count » en utilisant un script python en mode local et en mode distribué :

1) Pour un lancement en local :

Dans le répertoire /spark, nous avons créé un fichier « word_count.py » contenant un script python qui compte le nombre de mots en local.

Ensuite, nous avons lancé la commande suivante pour soumettre le script word_count.py pour exécution en local :

bash-5.0# ./bin/spark-submit word_count.py

Voici une suite de commandes que nous avons lancé dans le container Namenode pour visualiser part-00000 du file0.count :

```
imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it namenode /bin/bash
root@79c2d3facca6:/# hdfs dfs -ls /
Found 9 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
drwxr-xr-x  - root supergroup    0 2025-01-08 21:36 /file0.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:26 /file1.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:31 /file2.count
-rw-r--r--  3 root supergroup  1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup    0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup    0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup    0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup    0 2025-01-03 11:58 /tmp
root@79c2d3facca6:/# hdfs dfs -cat /file0.count/part-00000
('celui', 20)
('qui', 25)
('croyait', 20)
('au', 11)
('ciel', 10)
('ny', 10)
('pas', 11)
('tous', 3)
('deux', 6)
```

2) Pour un lancement en mode distribué :

Dans le répertoire /spark, créez un fichier « word_count_dist.py » contenant un script python qui compte le nombre de mots en mode distribué.

Pour lancer ce script en mode distribué, nous avons utilisé la commande suivante :

```
bash-5.0# spark-submit --master spark://spark-master:7077
word_count_dist.py
```

Voici une suite de commandes que nous avons lancé dans le container Namenode pour visualiser part-00000 du file3.count :

```

imane@imane-VirtualBox:~/Downloads/hadoop-main$ docker exec -it namenode /bin/bash
root@79c2d3faccac6:/# hdfs dfs -ls /
Found 10 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
drwxr-xr-x  - root supergroup   0 2025-01-08 21:36 /file0.count
drwxr-xr-x  - root supergroup   0 2025-01-08 21:26 /file1.count
drwxr-xr-x  - root supergroup   0 2025-01-08 21:31 /file2.count
drwxr-xr-x  - root supergroup   0 2025-01-08 21:43 /file3.count
-rw-r--r--  3 root supergroup 1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup   0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup   0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup   0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup   0 2025-01-03 11:58 /tmp
root@79c2d3faccac6:/# hdfs dfs -cat /file3.count/part-00000
('ils', 1)
('sont', 2)
('en', 2)
('le', 6)
('grabat', 1)
('que', 2)
('gele', 1)
('les', 4)
('rats', 1)
('croyait', 20)

```

V. Implémentation du « Word Count » comme application Spark Batch en utilisant Java.

→ Installation d'Apache Maven :

Teste de la configuration maven :

```

900a82d7cab9:/opt/apache-maven-3.5.0-bin# mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T19:39:06Z)
Maven home: /opt/apache-maven-3.5.0-bin/apache-maven-3.5.0
Java version: 1.8.0_275, vendor: IcedTea
Java home: /usr/lib/jvm/java-1.8-openjdk/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.8.0-51-generic", arch: "amd64", family: "unix"
900a82d7cab9:/opt/apache-maven-3.5.0-bin#

```

Création d'un projet Maven en utilisant les paramètres suivants :

```

ype-quickstart-1.1.jar (6.2 kB at 55 kB/s)
Define value for property 'groupId': inpt.myapp
Define value for property 'artifactId': myapp
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' inpt.myapp: :
Confirm properties configuration:
groupId: inpt.myapp
artifactId: myapp
version: 1.0-SNAPSHOT
package: inpt.myapp
Y: : Y
[INFO] -----

```

A la fin de la génération du projet maven, un message de « Build success » s'affiche :

```
Y: : Y
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0-SNAPSHOT
[INFO] -----
[INFO] Parameter: basedir, Value: /opt/apache-maven-3.5.0-bin
[INFO] Parameter: package, Value: inpt.myapp
[INFO] Parameter: groupId, Value: inpt.myapp
[INFO] Parameter: artifactId, Value: myapp
[INFO] Parameter: packageName, Value: inpt.myapp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /opt/apache-maven-3.5.0-bin/myapp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 31.440 s
[INFO] Finished at: 2025-01-08T22:45:10Z
[INFO] Final Memory: 18M/60M
[INFO] -----
900a82d7cab9:/opt/apache-maven-3.5.0-bin#
```

Utilisation de mvn pour compiler :

```
Downloaded: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.jar (208 kB at 274 kB/s)
[INFO] Building jar: /opt/apache-maven-3.5.0-bin/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 42.570 s
[INFO] Finished at: 2025-01-08T22:09:49Z
[INFO] Final Memory: 19M/60M
[INFO] -----
900a82d7cab9:/opt/apache-maven-3.5.0-bin/myapp#
```

Affichage de l'arborescence du projet avec la commande **tree** :

```
900a82d7cab9:/opt/apache-maven-3.5.0-bin# tree myapp/
myapp/
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       └── inpt
│   │           └── myapp
│   │               └── App.java
│   └── test
│       └── java
│           └── inpt
│               └── myapp
│                   └── AppTest.java
└── target
    ├── classes
    │   └── inpt
    │       └── myapp
    │           └── App.class
    ├── maven-archiver
    │   └── pom.properties
    └── maven-status
        └── maven-compiler-plugin
            ├── compile
            │   └── default-compile
            │       ├── createdFiles.lst
            │       └── inputFiles.lst
            └── testCompile
                └── default-testCompile
                    ├── createdFiles.lst
                    └── inputFiles.lst
```

→ Reconfiguration du projet Maven :

Tout d'abord, nous avons rajouté dans le fichier « pom.xml » les dépendances nécessaires pour notre application Spark Batch.

Ensuite, nous avons renommé App.java en WordCountTask.java et après nous avons supprimé le contenu ancien en le remplaçant par un code qui compte les mots .

Nous devons donc recompiler notre projet :

\$ mvn package

```
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myapp ---
[INFO] Building jar: /opt/apache-maven-3.5.0-bin/myapp/target/myapp-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 14.996 s
[INFO] Finished at: 2025-01-08T23:02:48Z
[INFO] Final Memory: 34M/81M
[INFO] -----
```

→ Lancement du programme avec spark-submit :

```
900a82d7cab9:/spark# ./bin/spark-submit --class inpt.myapp.WordCountTask /opt/apache-maven-3.5.0-bin/myapp/target/1.0-SNAPSHOT.jar hdfs://namenode:9000/poeme.txt hdfs://namenode:9000/results
25/01/08 23:07:40 INFO SparkContext: Running Spark version 3.3.0
25/01/08 23:07:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using built-in
classes where applicable
25/01/08 23:07:41 INFO ResourceUtils: =====
25/01/08 23:07:41 INFO ResourceUtils: No custom resources configured for spark.driver.
25/01/08 23:07:41 INFO ResourceUtils: =====
25/01/08 23:07:41 INFO SparkContext: Submitted application: inpt.myapp.WordCountTask
25/01/08 23:07:41 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name:
mount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap
: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
25/01/08 23:07:41 INFO ResourceProfile: Limiting resource is cpu
25/01/08 23:07:41 INFO ResourceProfileManager: Added ResourceProfile id: 0
25/01/08 23:07:42 INFO SecurityManager: Changing view acls to: root
25/01/08 23:07:42 INFO SecurityManager: Changing modify acls to: root
```

Pour afficher le contenu du « results », nous devons accéder au container du namenode et afficher le contenu du HDFS :

```
$ hdfs dfs -ls /
```

```
imane@imane-VirtualBox:~/Downloads$ docker exec -it namenode /bin/bash
root@79c2d3facca6:/# hdfs dfs -ls /
Found 11 items
drwxrwxrwt  - root root          0 2025-01-03 12:04 /app-logs
drwxr-xr-x  - root supergroup    0 2025-01-08 21:36 /file0.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:26 /file1.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:31 /file2.count
drwxr-xr-x  - root supergroup    0 2025-01-08 21:43 /file3.count
-rw-r--r--  3 root supergroup 1669 2025-01-03 11:48 /poeme.txt
drwxr-xr-x  - root supergroup    0 2025-01-03 12:04 /result2
drwxr-xr-x  - root supergroup    0 2025-01-08 23:08 /results
drwxr-xr-x  - root supergroup    0 2025-01-03 11:59 /results2
drwxr-xr-x  - root supergroup    0 2025-01-03 11:47 /rmstate
drwx-----  - root supergroup    0 2025-01-03 11:58 /tmp
root@79c2d3facca6:/#
```

```
$ hdfs dfs -cat /results/part-00000
```

```
root@79c2d3facca6:/# hdfs dfs -cat /results/part-00000
(jura,1)
(ils,1)
(violoncelle,1)
(comment,1)
(rose,1)
(soldats,1)
(que,2)
(celui,20)
(levres,1)
(bretagne,1)
(ses,1)
(quelle,1)
(de,5)
(les,4)
(cruelle,1)
(tombe,1)
(ruisnelle,1)
(combat,1)
(fideles,1)
(au,11)
(prefere,1)
(glas,1)
(un,4)
(quil,1)
```