



ECOLE NATIONALE  
DES SCIENCES  
APPLIQUEES  
TANGER



---

# Hadoop - MapReduce

---

*Réalisé par :*

Hamaoui AYOUB

Nafar Belal

*Encadrant :*

M. Hassan Badir

17 décembre 2019

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Outils . . . . .	2
1.1.1	Docker . . . . .	2
1.1.2	Image Docker Cloudera QuickStart . . . . .	2
1.2	Préparations de l'environnement . . . . .	2
1.2.1	Importation de l'image Cloudera QuickStart . . . . .	2
1.2.2	Exécution d'un conteneur Cloudera QuickStart . . . . .	2
<b>2</b>	<b>Atelier 1 : Map Reduce (Titanic Data Analysis)</b>	<b>5</b>
2.1	Énoncé du problème 1 . . . . .	5
2.1.1	Source code . . . . .	5
2.1.2	Étape 1 : Transfere des fichiers java vers conteneur cloudera/quickstart . . . . .	6
2.1.3	Étape 2 : Compilation du Job . . . . .	6
2.1.4	Étape 3 : Exécution du Job . . . . .	7
2.2	Énoncé du problème 2 . . . . .	8
2.2.1	Source code . . . . .	8
2.2.2	Étape 1 : Transfere des fichiers java vers conteneur cloudera/quickstart . . . . .	9
2.2.3	Étape 2 : Compilation du Job . . . . .	9
2.2.4	Étape 3 : Exécution du Job . . . . .	10
<b>3</b>	<b>Atelier 2 : Hadoop - MapReduce (WordCount)</b>	<b>11</b>
3.1	Présentation . . . . .	11
3.2	Préparation des données . . . . .	11
3.2.1	Etape 1 : Importer des données dans HDFS . . . . .	11
3.3	WordCount en JAVA . . . . .	13
3.3.1	Le Driver . . . . .	13
3.3.2	Le Mapper . . . . .	14
3.3.3	Le Reducer . . . . .	14
3.3.4	Transfere des fichiers java vers conteneur cloudera/quickstart . . . . .	15
3.4	Compilation et exécution du Job . . . . .	16
3.4.1	Étape 1 : Compilation du Job . . . . .	16
3.4.2	Étape 2 : Exécution du Job . . . . .	16
3.4.3	Résultats . . . . .	17
<b>4</b>	<b>Bibliographie</b>	<b>19</b>

# 1 Introduction

## 1.1 Outils

### 1.1.1 Docker

**Docker** est différent des autres plates-formes, car il utilise des conteneurs Linux. La plupart des "machines virtuelles" fonctionnent en isolant ou en simulant l'accès au matériel de l'hôte afin qu'un système d'exploitation invité complet puisse s'exécuter dessus. Les conteneurs Linux, cependant, fonctionnent en partitionnant les ressources du système d'exploitation hôte : ils ont leur propre vue du système de fichiers et d'autres ressources, mais ils s'exécutent sur le même noyau. Ceci est similaire aux prisons BSD ou aux zones Solaris. Docker fournit des outils, un format d'emballage et une infrastructure autour des conteneurs Linux et des technologies associées.

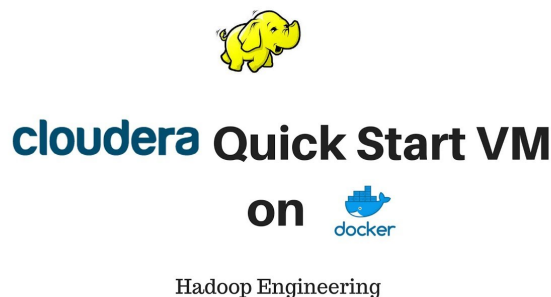


FIGURE 2 – Cloudera et Docker

### 1.1.2 Image Docker Cloudera QuickStart

Les machines virtuelles **Cloudera QuickStart** et cette image Docker sont des déploiements à nœud unique de la distribution 100% open source de Cloudera, notamment Apache Hadoop et Cloudera Manager. Ce sont des environnements idéaux pour découvrir Hadoop, essayer de nouvelles idées, tester et faire la démonstration des applications.

## 1.2 Préparations de l'environnement

### 1.2.1 Importation de l'image Cloudera QuickStart

Vous pouvez importer l'image Cloudera QuickStart depuis Docker Hub :

```
1 docker pull cloudera/quickstart
```

### 1.2.2 Exécution d'un conteneur Cloudera QuickStart

Pour exécuter un conteneur à l'aide de l'image, vous devez connaître le nom ou le hachage de l'image. Si vous avez suivi les instructions d'importation ci-dessus, le nom pourrait être **cloudera/quickstart :latest** (ou autre chose si vous avez téléchargé plusieurs versions). Le hachage est également imprimé dans le terminal lorsque vous importez, ou vous pouvez rechercher les hachages de toutes les images importées avec :

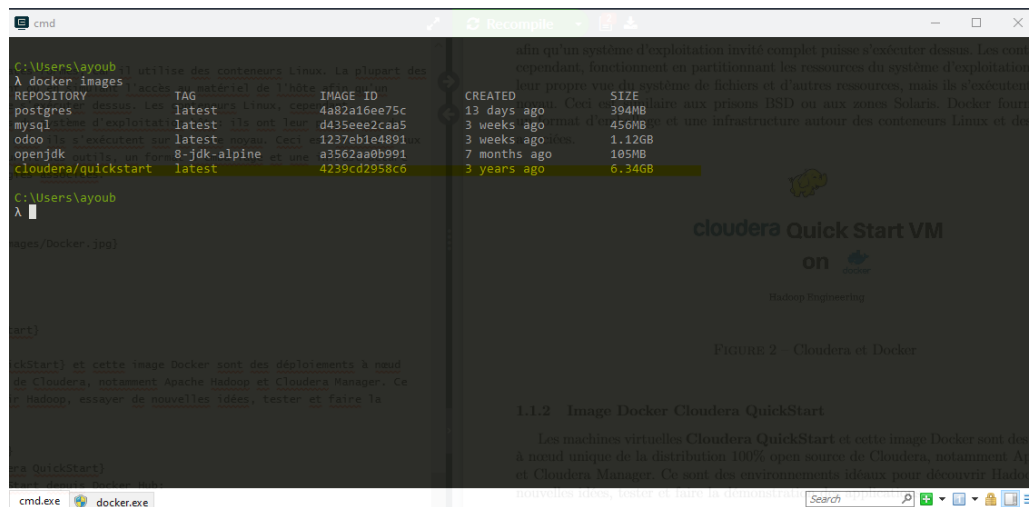


FIGURE 3 – Commande : docker images

Une fois que vous connaissez le nom ou le hachage de l'image, vous pouvez l'exécuter :

```
1 docker run --name quickstart.cloudera --hostname=quickstart.cloudera -d --
  privileged=true -t -i -p 8888:8888 -p 7180:7180 -p 8181:80 docker.io/
  cloudera/quickstart:latest /usr/bin/docker-quickstart
```

Listing 1 – Docker sous windows

ou bien :

```
1 docker run --name quickstart.cloudera --hostname=quickstart.cloudera -d --
  privileged=true -t -i docker.io/cloudera/quickstart:latest /usr/bin/
  docker-quickstart
```

Listing 2 – Docker sous Linux

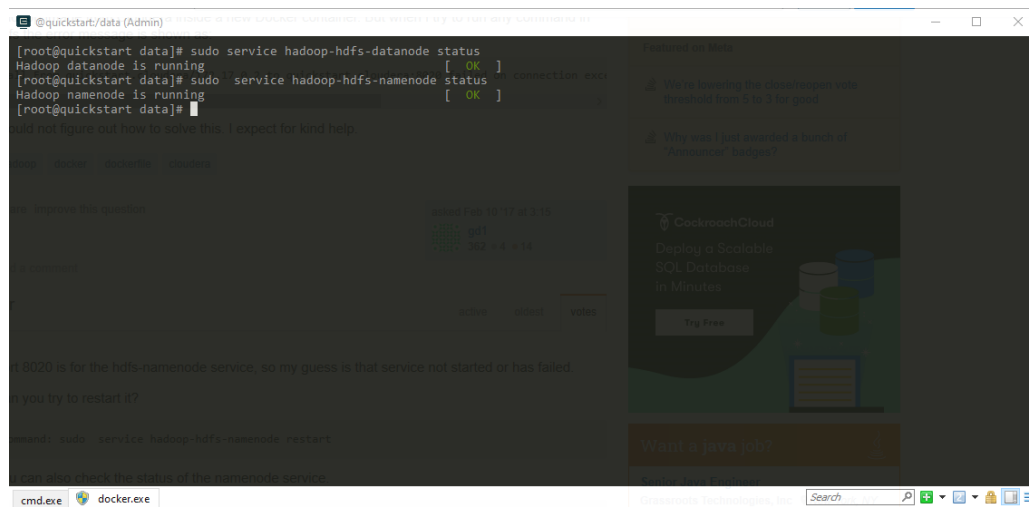
(hostname ne doit pas être modifié, sinon certains services ne démarrent pas correctement)

Pour détacher le tty sans quitter le shell, utilisez la séquence d'échappement **Ctrl+p + Ctrl+q**

**IMPORTANT** : pour Windows, vous devez également exposer un ou plusieurs ports lors des commandes d'exécution du docker en ajoutant les options -p 8888 : 8888 -p 7180 : 7180 -p 8181 : 80 (host\_port : container\_port). Voir ci-dessous pour les ports disponibles.

**REMARQUE** : il faudra plusieurs minutes pour démarrer tous les services et commencer à répondre aux ports http.

- Si vous recevez des problèmes de synchronisation d'horloge, essayez d'exécuter manuellement `/etc/init.d/ntpd start` pour synchroniser l'heure.
- Parfois **DataNode** et **NameNode** ne fonctionne pas, pour les faire fonctionner lancer les commandes :
  - `sudo service hadoop-hdfs-datanode start`
  - `sudo service hadoop-hdfs-namenode restart`



The screenshot shows a terminal window titled '@quickstart/data (Admin)'. The user runs the following commands:

```

[root@quickstart data]# sudo service hadoop-hdfs-datanode status
Hadoop datanode is running
[root@quickstart data]# sudo service hadoop-hdfs-namenode status
Hadoop namenode is running
[root@quickstart data]#
    
```

The terminal output shows 'Hadoop datanode is running' and 'Hadoop namenode is running' with status '[ OK ]'. Below the terminal, there is a Cloudera Quickstart interface with a dark theme. It includes a 'Featured on Data' section with a 'Try Free' button, a 'CockroachCloud' section with a 'Try Free' button, and a 'Want a java job?' section. The terminal window is running on a system with 'cmd.exe' and 'docker.exe' in the taskbar.

FIGURE 4 – SHELL CLOUDERA QUICKSTART

## 2 Atelier 1 : Map Reduce (Titanic Data Analysis)

### 2.1 Énoncé du problème 1

Dans cet énoncé du problème, nous trouverons l'âge moyen des hommes et des femmes décédés dans la tragédie du Titanic.

#### 2.1.1 Source code

Maintenant, à partir du **Mapper**, nous allons dériver :

- Le genre comme clé
- Age comme valeur

Ces valeurs seront transmises à la phase de mélange et de tri et seront ensuite envoyées à la phase de réduction où l'agrégation des valeurs est effectuée.

Mapper code :

```
1 public class Average_age {
2     public static class Map extends Mapper<LongWritable, Text, Text,
3         IntWritable> {
4         private Text gender = new Text();
5         private IntWritable age = new IntWritable();
6         public void map(LongWritable key, Text value, Context context )
7             throws IOException, InterruptedException {
8             String line = value.toString();
9             String str[]=line.split(",");
10            if(str.length>6){
11                gender.set(str[4]);
12                if((str[1].equals("0")) ){
13                    if(str[5].matches("\\d+")){
14                        int i=Integer.parseInt(str[5]);
15                        age.set(i);
16                    }
17                }
18            }
19            context.write(gender, age);
20        }
21    }
22 }
```

### Reducer Code :

```

1 public static class Reduce extends Reducer<Text,IntWritable, Text,
  IntWritable> {
2     public void reduce(Text key, Iterable<IntWritable> values, Context
  context) throws IOException, InterruptedException {
3         int sum = 0;
4         int l=0;
5         for (IntWritable val : values) {
6             l+=1;
7             sum += val.get();
8         }
9         sum=sum/l;
10        context.write(key, new IntWritable(sum));
11    }
12 }
    
```

### Configuration Code :

```

1 job.setMapOutputKeyClass(Text.class);
2 job.setMapOutputValueClass(IntWritable.class);
    
```

Vous pouvez télécharger l'intégralité du code source discuté à partir du lien ci-dessous :  
<https://github.com/kiran0541/Map-Reduce>

### 2.1.2 Étape 1 : Transfere des fichiers java vers conteneur cloudera/quickstart

On utilisons les commandes :

```

1 docker cp C:\Users\ayoub\OneDrive\Documents\TP_BIG_DATA\Titanic\CodeJava\
  Average_age.java quickstart.cloudera:/dataTitanic/
2
3
4 docker cp C:\Users\ayoub\OneDrive\Documents\TP_BIG_DATA\Titanic\TitanicData.
  txt quickstart.cloudera:/dataTitanic/
    
```

### 2.1.3 Étape 2 : Compilation du Job

Les commandes à utiliser sont les suivantes (en vert) :

```

1 [root@quickstart dataTitanic]# ls
2 Average_age.java  TitanicData.txt
3
4 [root@quickstart dataTitanic]# hadoop fs -mkdir dataTitanic
5
6
7 [root@quickstart dataTitanic]# hadoop fs -copyFromLocal /dataTitanic/
  TitanicData.txt dataTitanic
8
9
10 [root@quickstart dataTitanic]# hadoop fs -ls dataTitanic
11 Found 1 items
12 -rw-r--r--  1 root supergroup      61111 2019-12-17 20:09 dataTitanic/
  TitanicData.txt
13
14
    
```

```
15 [root@quickstart dataTitanic]# javac *.java -cp $(hadoop classpath)
16 Note: Average_age.java uses or overrides a deprecated API.
17 Note: Recompile with -Xlint:deprecation for details.
18
19
20 [root@quickstart dataTitanic]# jar cvf titanicAvg.jar *.class
21 added manifest
22 adding: Average_age.class(in = 1818) (out= 885)(deflated 51%)
23 adding: Average_age$Map.class(in = 2042) (out= 865)(deflated 57%)
24 adding: Average_age$Reduce.class(in = 1659) (out= 709)(deflated 57%)
```

### 2.1.4 Étape 3 : Exécution du Job

Façon d'exécuter le fichier Jar pour obtenir le résultat de la première déclaration de problème :(en vert)

```
1 [root@quickstart dataTitanic]# hadoop jar titanicAvg.jar Average_age
2 19/12/17 20:14:38 INFO client.RMPProxy: Connecting to ResourceManager at
3 /0.0.0.0:8032
4 ...
5
6 [root@quickstart dataTitanic]# hadoop fs -tail avg_out/part-r-00000
7 female 28
8 male 30
```

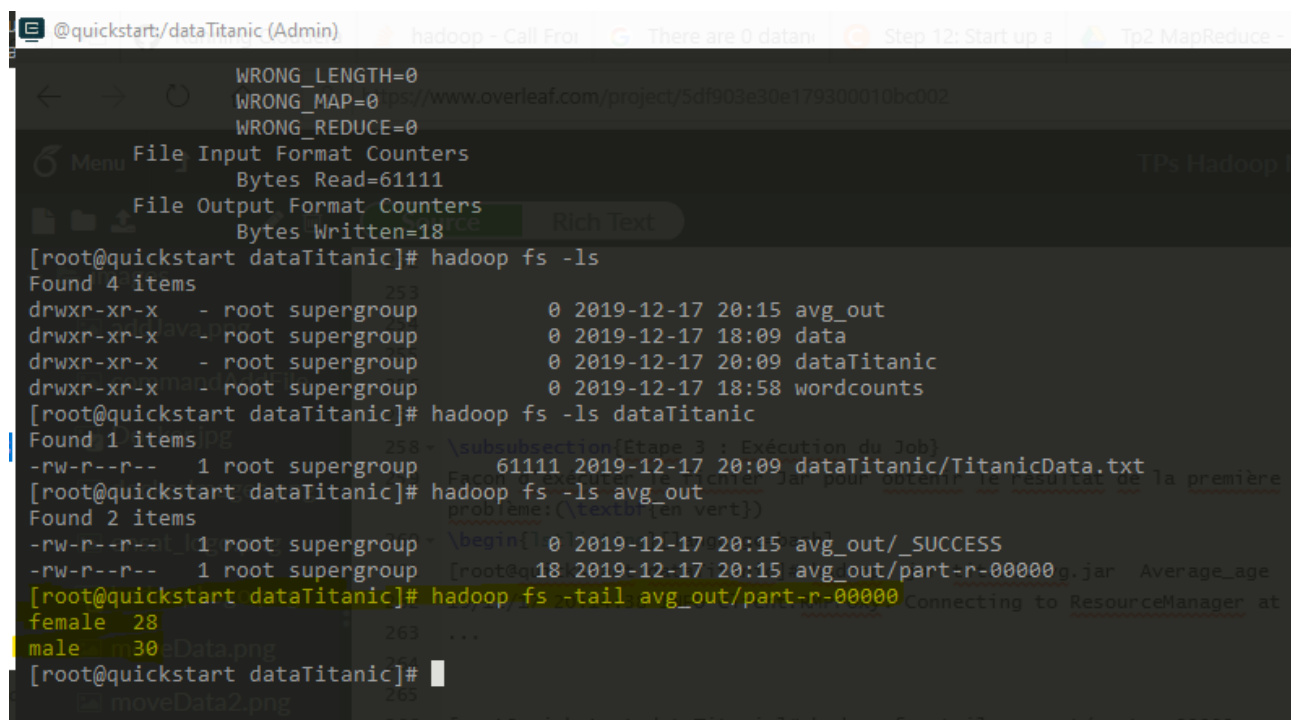


FIGURE 5 – Avg



## 2.2 Énoncé du problème 2

Dans cet énoncé du problème, nous trouverons le nombre de personnes décédées ou ayant survécu dans chaque classe avec leur sexe et leur âge.

### 2.2.1 Source code

Maintenant, à partir du **Mapper**, nous voulons obtenir la clé composite qui est la combinaison de la classe de passagers et du sexe et de leur âge et de la valeur int finale '1' en tant que valeurs qui seront transmises à la phase de mélange et de tri et qui seront ensuite envoyées au réducteur. phase où l'agrégation des valeurs est effectuée.

Mapper code :

```

1 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable
  > {
2     private Text people = new Text();
3     private final static IntWritable one = new IntWritable(1);
4     public void map(LongWritable key, Text value, Context context )
      throws IOException, InterruptedException {
5
6         String line = value.toString();
7         String str[]=line.split(",");
8         if(str.length>6){
9             String survived=str[1]+" "+str[4]+" "+str[5];
10            people.set(survived);
11            context.write(people, one);
12        }
13    }
14 }
    
```

Reducer code :

```

1 public static class Reduce extends Reducer<Text,IntWritable, Text,
  IntWritable> {
2
3     public void reduce(Text key, Iterable<IntWritable> values, Context
      context) throws IOException, InterruptedException {
4         int sum = 0;
5         for (IntWritable val : values) {
6             sum += val.get();
7         }
8         context.write(key, new IntWritable(sum));
9     }
10
11 }
    
```

### 2.2.2 Étape 1 : Transfere des fichiers java vers conteneur cloudera/quickstart

On utilisons les commandes :

```
1 docker cp C:\Users\ayoub\OneDrive\Documents\TP_BIG_DATA\Titanic\CodeJava\
  Female_survived.java quickstart.cloudera:/dataTitanic/
2
3 docker cp C:\Users\ayoub\OneDrive\Documents\TP_BIG_DATA\Titanic\TitanicData.
  txt quickstart.cloudera:/dataTitanic/
```

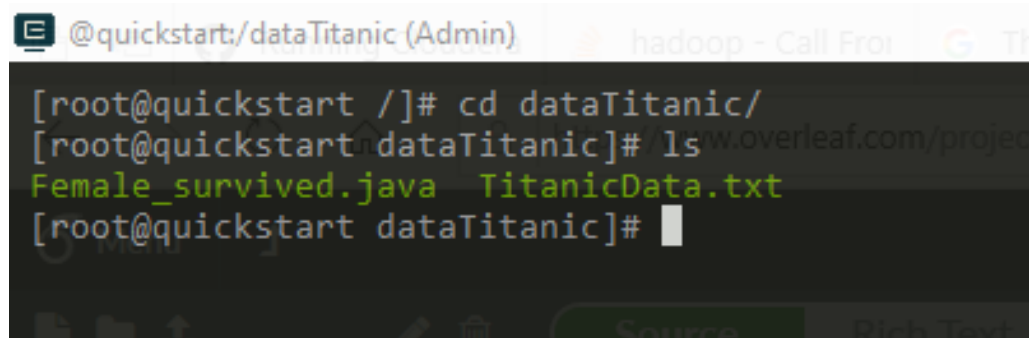


FIGURE 6 – Dossier dataTitanic

### 2.2.3 Étape 2 : Compilation du Job

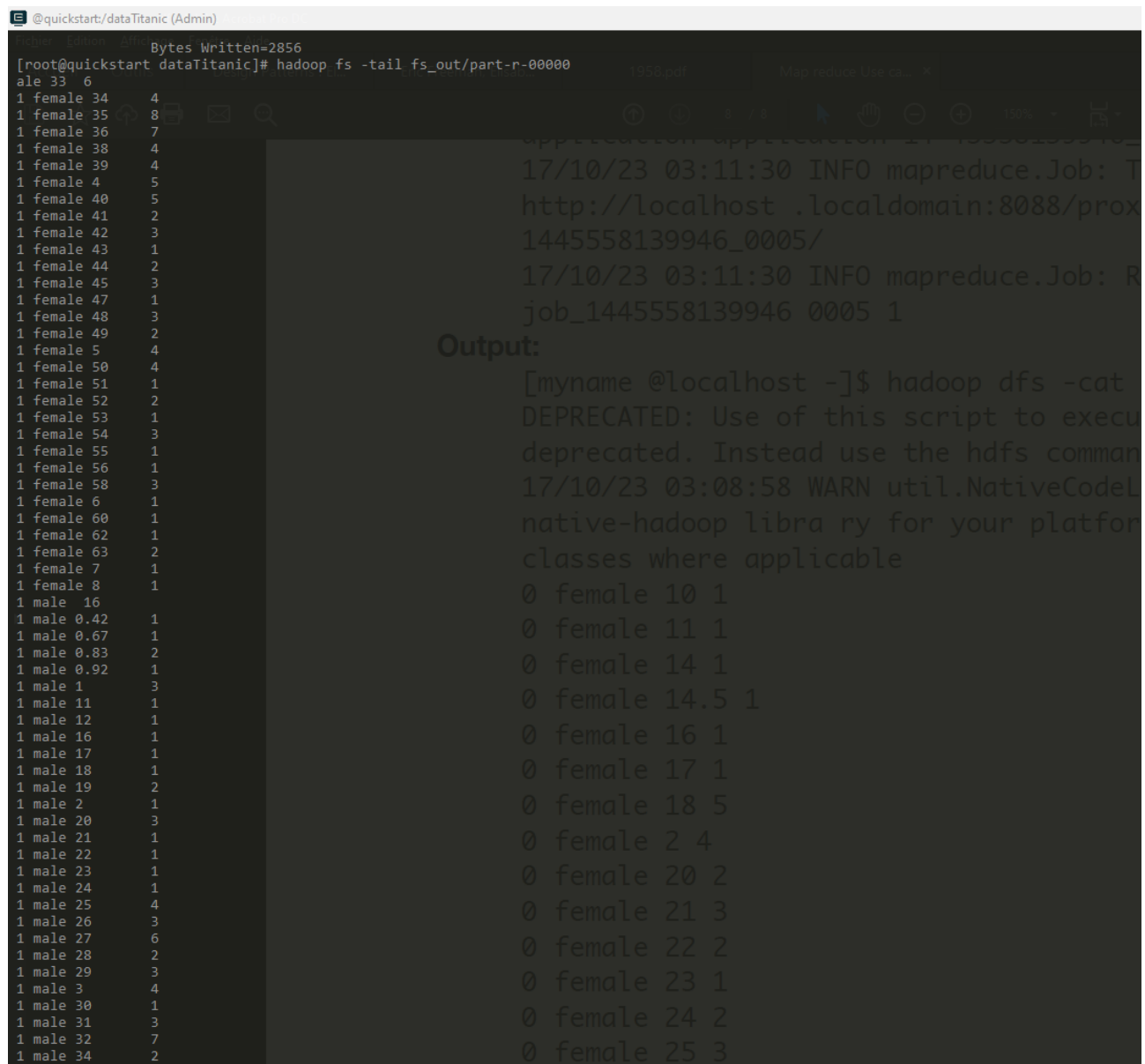
Les commandes à utiliser sont les suivantes (en vert) :

```
1 [root@quickstart dataTitanic]# ls
2 Female_survived.java TitanicData.txt
3
4 [root@quickstart dataTitanic]# hadoop fs -mkdir dataTitanicFS
5
6
7 [root@quickstart dataTitanic]# hadoop fs -copyFromLocal /dataTitanic/
  TitanicData.txt dataTitanicFS
8
9
10 [root@quickstart dataTitanic]# hadoop fs -ls dataTitanicFS
11 Found 1 items
12 -rw-r--r-- 1 root supergroup 61111 2019-12-17 20:09 dataTitanic/
  TitanicData.txt
13
14
15 [root@quickstart dataTitanic]# javac *.java -cp $(hadoop classpath)
16 Note: Average_age.java uses or overrides a deprecated API.
17 Note: Recompile with -Xlint:deprecation for details.
18
19
20 [root@quickstart dataTitanic]# jar cvf titanicFS.jar *.class
21 added manifest
22 adding: Average_age.class(in = 1818) (out= 885)(deflated 51%)
23 adding: Average_age$Map.class(in = 2042) (out= 865)(deflated 57%)
24 adding: Average_age$Reduce.class(in = 1659) (out= 709)(deflated 57%)
```

### 2.2.4 Étape 3 : Exécution du Job

Façon d'exécuter le fichier Jar pour obtenir le résultat de la première déclaration de problème :(en vert)

```
1 [root@quickstart dataTitanic]# hadoop jar titanicFS.jar Female_survived
   dataTitanicFS fs_out
2 19/12/17 20:14:38 INFO client.RMProxy: Connecting to ResourceManager at
   /0.0.0.0:8032
3 ...
4
5
6 [root@quickstart dataTitanic]# hadoop fs -tail avg_out/part-r-00000
7 female 28
8 male 30
```



```
@quickstart/dataTitanic (Admin)
root@quickstart dataTitanic]# hadoop fs -tail fs_out/part-r-00000
19/12/17 20:14:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
...
[root@quickstart dataTitanic]# hadoop fs -tail avg_out/part-r-00000
female 28
male 30
```

FIGURE 7 – Female survived

## 3 Atelier 2 : Hadoop - MapReduce (WordCount)

### 3.1 Présentation

WordCount est surnommé le "Hello World !" de Hadoop. Ce programme compte le nombre d'occurrences de chaque mot dans un corpus représentant l'ensemble des oeuvres de Shakespeare. Le développement et la mise en oeuvre d'un programme Hadoop comprennent en général les phases suivantes :

- Préparation des données.
- Importation des données dans HDFS.
- Ecriture du programme Hadoop et validation en environnement de test.
- Exécution du programme Hadoop en environnement de production.
- Récupération et analyse des résultats.

### 3.2 Préparation des données

La totalité de l'oeuvre de Shakespeare est stockée dans un seul fichier au format Plain Text UTF-8 provenant du projet Gutenberg (<http://www.gutenberg.org/>). Ce fichier peut être téléchargé à l'adresse <http://www.gutenberg.org/ebooks/100>. Si cette adresse ne fonctionne pas, lancer une recherche Google sur l'expression download shakespeare works.

Le fichier téléchargé est enregistré sur le bureau sous le nom pg100.txt.

Le fichier se présente sous la forme d'un ensemble de lignes, chaque ligne se terminant par un signe de nouvelle ligne ( \n ). Chaque ligne est composée de mots séparés par un espace ou est vide.

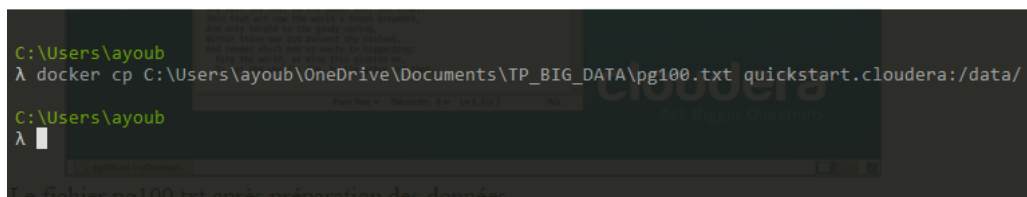
Le fichier pg100.txt après préparation des données

Dans un projet Hadoop, une phase de préparation des données peut être nécessaire, bien que pas toujours obligatoire.

#### 3.2.1 Etape 1 : Importer des données dans HDFS

Pour importer le fichier pg100.txt dans HDFS, procédez de la manière suivante :

- Ouvrez le Terminal (SHELL CLOUDERA QUICKSTART)
- Crée un sous-répertoire **data**
- Pour passer fichier pg100.txt à notre conteneur cloudera/quickstart lancer la commande :
  - `docker cp [PATH_TO_FILE]/pg100.txt quickstart.cloudera:[PATH_TO_FOLDER]/data/`



```
C:\Users\ayoub
λ docker cp C:\Users\ayoub\OneDrive\Documents\TP_BIG_DATA\pg100.txt quickstart.cloudera:/data/
C:\Users\ayoub
λ
```

FIGURE 8 – Ajouter fichier pg100.txt

Vérifier que fichier est bien été crée

```
[root@quickstart /]# ls
bin boot data dev etc home lib lib64 lost+found media mnt opt packer_files proc root sbin selinux srv sys tmp usr var
[root@quickstart /]# cd data
[root@quickstart data]# ls
pg100.txt
[root@quickstart data]#
```

FIGURE 9 – Vérifier que fichier pg100.txt est bien crée

Les commandes à utiliser sont les suivantes :

```
1 [root@quickstart data]# hadoop fs -mkdir data
2
3 [root@quickstart data]# hadoop fs -ls
4 Found 1 items
5 drwxr-xr-x - root supergroup 0 2019-12-17 18:08 data

1 [root@quickstart /]# hadoop fs -copyFromLocal /data/pg100.txt data
2
3 [root@quickstart /]# hadoop fs -ls data
4 Found 1 items
5 -rw-r--r-- 1 root supergroup 5582655 2019-12-17 18:09 data/pg100.txt
```

Les lignes qui prouvent que le répertoire data a bien été créé et que le fichier pg100.txt a bien été copié apparaissent en gras et en vert.

```
[root@quickstart data]# hadoop fs -mkdir data
[root@quickstart data]# hadoop fs -ls
Found 1 items
drwxr-xr-x - root supergroup 0 2019-12-17 18:08 data
[root@quickstart data]# cd ..
[root@quickstart /]# hadoop fs -copyFromLocal /data/pg100.txt data
[root@quickstart /]# hadoop fs -ls data
Found 1 items
-rw-r--r-- 1 root supergroup 5582655 2019-12-17 18:09 data/pg100.txt
[root@quickstart /]#
```

FIGURE 10 – Les commandes pour ajouter fichier

## 3.3 WordCount en JAVA

### 3.3.1 Le Driver

Le driver est un programme java qui s'exécute généralement sur la machine cliente (donc pas dans le Cluster Hadoop). Il permet de configurer le job puis de le soumettre au cluster Hadoop pour exécution.

Le code du Drive de WordCount se trouve ci-après. Le fichier **WordCountDriver.java** :

```
1
2 import org.apache.hadoop.fs.Path;
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
6 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
7 import org.apache.hadoop.mapreduce.Job;
8
9 public class WordCountDriver{
10     public static void main(String[] args) throws Exception{
11         if(args.length != 2){
12             System.out.printf("Format de la ligne de commande: WordCount <input
13             dir> <output dir>\n");
14             System.exit(-1);
15         }
16
17         Job job = new Job();
18         job.setJarByClass(WordCountDriver.class);
19
20         job.setJobName("Word Count");
21
22         FileInputFormat.setInputPaths(job, new Path(args[0]));
23         FileOutputFormat.setOutputPath(job, new Path(args[1]));
24
25         job.setMapperClass(WordCountMapper.class);
26         job.setReducerClass(WordCountReducer.class);
27
28         job.setMapOutputKeyClass(Text.class);
29         job.setOutputValueClass(IntWritable.class);
30
31         boolean success = job.waitForCompletion(true);
32         System.exit(success ? 0 : 1);
33     }
34 }
```

### 3.3.2 Le Mapper

Le Mapper est un programme java exécuté en parallèle sur plusieurs noeuds esclaves (**slave nodes**) du cluster Hadoop, chaque instance étant un Mapper. Chaque mapper compte le nombre d'occurrences d'un mots dans une partie des oeuvres de Shakspeare (les reducers se chargent de synthétiser le travail des mappers).

Le code du Mapper de WordCount se trouve ci-après. Le développeur Hadoop rédige son programme comme s'il ne devait s'appliquer qu'à un seul enregistrement. Le passage d'un enregistrement à un autre et la gestion de la fin de fichier sont pris en charge par Hadoop « en coulisse »

Le fichier **WordCountMapper.java** :

```

1
2 import java.io.IOException;
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.LongWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Mapper;
7
8 public class WordCountMapper extends Mapper<LongWritable, Text, Text,
9     IntWritable>{
10
11     @Override
12     public void map(LongWritable key, Text value, Context context) throws
13         IOException, InterruptedException{
14
15         String line = value.toString();
16
17         for(String word : line.split("\\W+")){
18             if(word.length() > 0){
19                 context.write(new Text(word), new IntWritable(1));
20             }
21         }
22     }
23 }
```

### 3.3.3 Le Reducer

Le Reducer est un programme Java exécuté en parallèle sur plusieurs noeuds esclaves (**slave nodes**) du Cluster Hadoop, chaque instance étant un Reducer. Chaque Reducer :

- Se voit affecter par Hadoop un sous-ensemble de l'ensemble des mots constituant les oeuvres de Shakespeare.
- Est chargé, pour chaque mot de ce sous-ensemble, de cumuler les comptages de ce même mot issus, le cas échéant, de différents mappers.

**Remarque :** Il convient de garder dans l'esprit que les différentes occurrences d'un mot donné des oeuvres de Shakspeare, par exemple l'article « the », pourront être prises en compte par plusieurs mappers. Par contre, toutes les occurrences de « the » seront envoyées par Hadoop à un seul et même Reducer, même si elles proviennent de mappers différents, c'est ce qui permet de garantir qu'aucune occurrence de « the » ne sera oubliée.

Le fichier **WordCountReducer.java** :

```

1
2 import java.io.IOException;
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Reducer;
6
7 public class WordCountReducer extends Reducer<Text, IntWritable, Text,
8     IntWritable>{
9
10    @Override
11    public void reduce(Text key, Iterable<IntWritable> values, Context context)
12        throws IOException, InterruptedException{
13
14        int wordCount = 0;
15        for(IntWritable value : values){
16            wordCount = wordCount + value.get();
17        }
18
19        context.write(key, new IntWritable(wordCount));
20    }
21 }

```

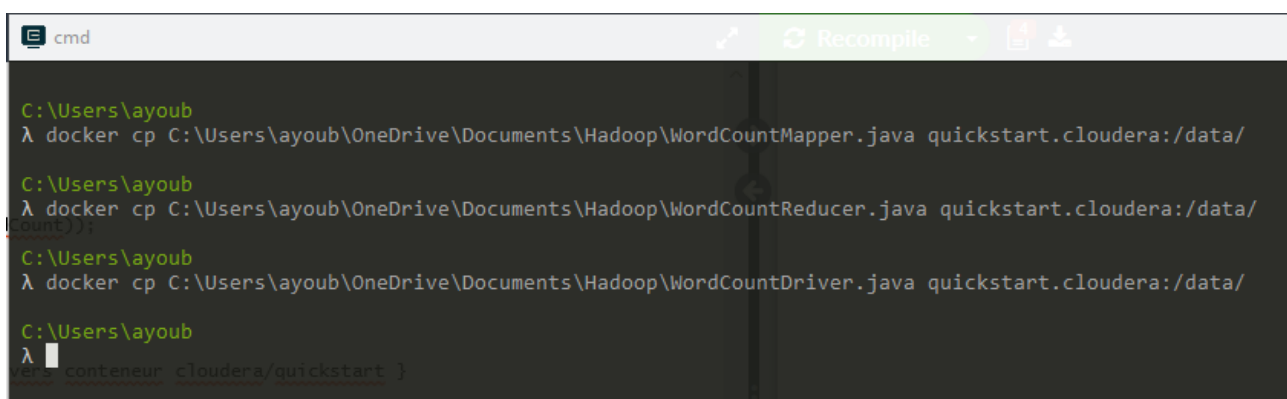
### 3.3.4 Transfere des fichiers java vers conteneur cloudera/quickstart

On utilise les commandes :

```

1 docker cp [PATH\_To\_FileJAVA]\WordCountMapper.java quickstart.cloudera:/
2   data/
3
4 docker cp [PATH\_To\_FileJAVA]\WordCountReducer.java quickstart.cloudera:/
5   data/
6
7 docker cp [PATH\_To\_FileJAVA]\WordCountDriver.java quickstart.cloudera:/
8   data/

```



```

cmd
C:\Users\ayoub
λ docker cp C:\Users\ayoub\OneDrive\Documents\Hadoop\WordCountMapper.java quickstart.cloudera:/data/

C:\Users\ayoub
λ docker cp C:\Users\ayoub\OneDrive\Documents\Hadoop\WordCountReducer.java quickstart.cloudera:/data/

C:\Users\ayoub
λ docker cp C:\Users\ayoub\OneDrive\Documents\Hadoop\WordCountDriver.java quickstart.cloudera:/data/

C:\Users\ayoub
λ

```

FIGURE 11 – Ajouter les fichiers JAVA



## 3.4 Compilation et exécution du Job

### 3.4.1 Étape 1 : Compilation du Job

Pour compiler le programme WordCount, procédez de la manière suivante :

- Ouvrez le Terminal et positionnez-vous au niveau du dossier data
- Vérifiez que les trois fichiers Java (driver, mapper et reducer) sont présents.
- Compilez le driver, le mapper et le reducer.
- Vérifiez que les trois fichiers compilés (.class) sont présents.
- Transformez les trois fichiers compilés en un fichier JAR exécutable.

Les commandes à utiliser sont les suivantes (**en vert**) :

```

1 [root@quickstart /]# cd data
2
3
4 [root@quickstart data]# ls
5 pg100.txt  WordCountDriver.java  WordCountMapper.java  WordCountReducer.java
6
7
8 [root@quickstart data]# javac *.java -cp $(hadoop classpath)
9 Note: WordCountDriver.java uses or overrides a deprecated API.
10 Note: Recompile with -Xlint:deprecation for details.
11
12 [root@quickstart data]# ls
13 pg100.txt          WordCountDriver.java  WordCountMapper.java
14 WordCountReducer.java
15 WordCountDriver.class  WordCountMapper.class  WordCountReducer.class
16
17 [root@quickstart data]# jar cvf wc.jar *.class
18 added manifest
19 adding: WordCountDriver.class(in = 1576) (out= 852)(deflated 45%)
20 adding: WordCountMapper.class(in = 1709) (out= 711)(deflated 58%)
21 adding: WordCountReducer.class(in = 1602) (out= 668)(deflated 58%)
22
23 [root@quickstart data]# hadoop jar wc.jar WordCountDriver data wordcounts
24 19/12/17 18:46:23 INFO client.RMProxy: Connecting to ResourceManager at
25 /0.0.0.0:8032

```

Le programme WordCount est prêt à être exécuté. Le fichier JAR exécutable est nommé wc.jar

### 3.4.2 Étape 2 : Exécution du Job

Pour exécuter le programme WordCount, procédez de la manière suivante :

- Ouvrez le Terminal et positionnez-vous au niveau du dossier **data**.
- Lancez l'exécution de WordCountDriver.jar en précisant le répertoire en entrée (dans lequel se situe le fichier des oeuvres complètes de Shakespeare) et le répertoire en sortie (dans lequel seront stockés les fichiers de résultats- un fichier par reducer), puis patientez...
- Une fois le job terminé, affichez les résultats.

Les commandes à utiliser sont les suivantes :

```
1 [root@quickstart /]# cd data
2
3
4 [root@quickstart data]# hadoop jar wc.jar WordCountDriver data wordcounts
5 19/12/17 18:46:23 INFO client.RMProxy: Connecting to ResourceManager at
6 /0.0.0.0:8032
...
```

La commande permettant de lancer WordCount s'analyse de la façon suivante :

- **hadoop** : la commande à exécuter est une commande Hadoop, pas Linux.
- **jar** : la commande à exécuter est jar, qui lance l'exécution d'un fichier JAR.
- **wc.jar** : le fichier à exécuter est wc.jar.
- **WordCountDriver** : nom de la classe à appeler pour lancer le job.
- **data** : répertoire contenant les données en entrée.
- **wordcounts** : répertoire contenant les résultats en sortie.

L'exécution du job donne lieu à l'affichage de nombreux messages.

### 3.4.3 Résultats

Visualisez le contenu du fichier qui contient les résultats lancer les commandes :

```
1 [root@quickstart data]# hadoop fs -ls wordcounts
2 Found 2 items
3 -rw-r--r-- 1 root supergroup 0 2019-12-17 18:58 wordcounts/
4 -rw-r--r-- 1 root supergroup 301374 2019-12-17 18:58 wordcounts/part-r-
5 -00000
6
7 [root@quickstart data]# hadoop fs -tail wordcounts/part-r-00000
```

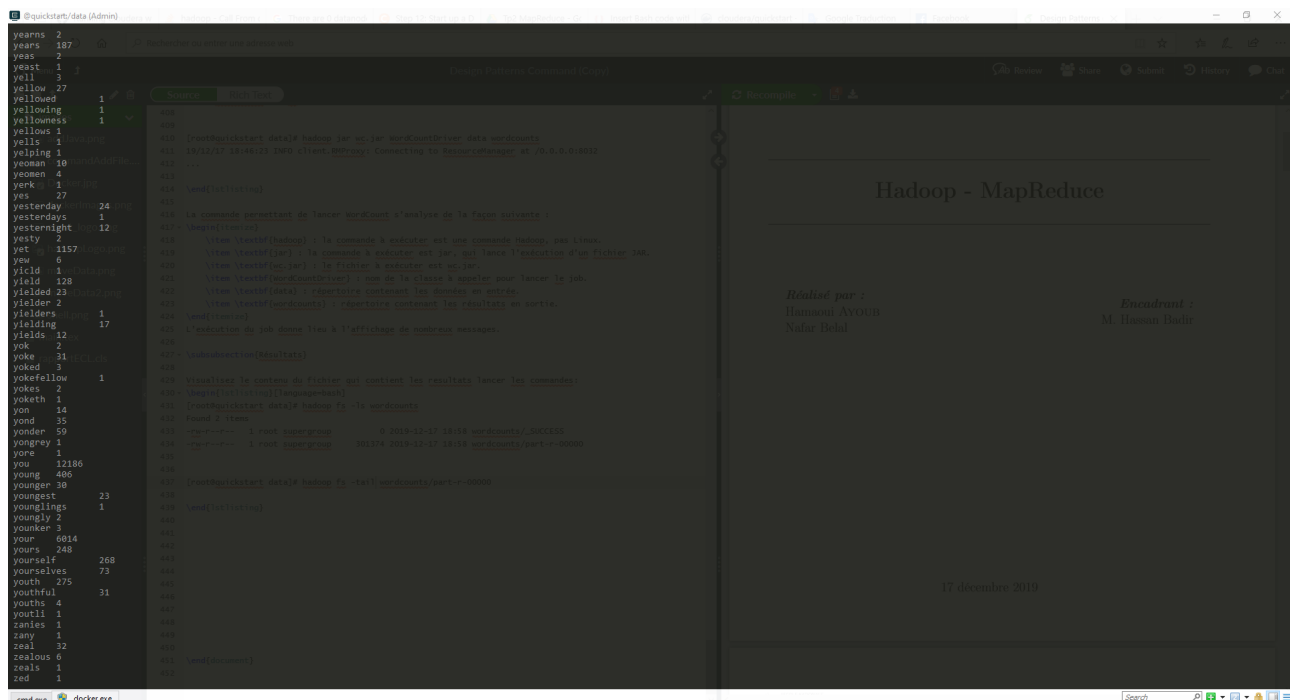


FIGURE 12 – Résultats

```

thankful 19
thankfully 7
thankfulness 5
thanking 1
thankings 3
thankless 3
thanks 123
thanksgiving 2
that 8343
thatch 5
thaw 6
thaws 1
the 23288
theatre 5
thee 3162
theft 14
thefts 2
thein 1
their 1935
theirs 29
    
```

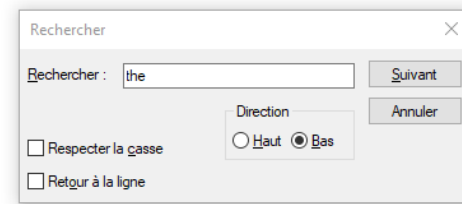


FIGURE 13 – "the" apparaît 23 288 fois

## 4 Bibliographie

- TP\_2 MapReduce, M. Hassan Badir
- Map reduce Use case – Titanic Data Analysis, Kiran, August 9, 2017, <https://acadgild.com/blog/analyzing-titanic-data-hadoop-mapreduce>
- <https://hub.docker.com/r/cloudera/quickstart/>
- [https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/cdh\\_sg\\_datanode\\_start.html](https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/cdh_sg_datanode_start.html)
- <https://stackoverflow.com/questions/42150883/call-from-quickstart-cloudera-172-17-0->
- <https://gist.github.com/davideicardi/21aabb65faaa3c655770cf0ccbac6564>