[https://github.com/andfanilo/cloudera-quickstartvm-tutorial?tab=readme-ov-file](https://github.com/andfanilo/cloudera-quickstartvm-tutorial?tab=readme-ov-file" \t "_blank)

**MapReduce using Java**

1. **Loading data**

In hadoop, make a new directory called sampledata/, and verify that it was created.

hdfs dfs -mkdir sampledata

hdfs dfs -ls

hdfs dfs -chmod 777 sampledata

upload the data file from the local file system to the HDFS using the following commands:

hdfs dfs -put data.txt sampledata

hdfs dfs -ls sampledata

hdfs dfs -cat sampledata/data.txt

1. **Coding**

**Start your Java project**

Create a directory to hold the three Java files that you will be making and make it accessible.

mkdir dossier1

cd dossier1

**Create the Java file for the mapper class (WC\_Mapper.java)**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class WC\_Mapper extends MapReduceBase implements Mapper**<LongWritable**,Text,Text,IntWritable**>**{

    private final static IntWritable one = new IntWritable(1);

    private Text word = new Text();

    public void map(LongWritable key, Text value,OutputCollector**<Text**,IntWritable**>** output,

           Reporter reporter) throws IOException{

        String line = value.toString();

        StringTokenizer  tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()){

            word.set(tokenizer.nextToken());

            output.collect(word, one);

        }

    }

}

**Create the reducer class (WC\_Reducer.java)**

 import java.io.IOException;

    import java.util.Iterator;

    import org.apache.hadoop.io.IntWritable;

    import org.apache.hadoop.io.Text;

    import org.apache.hadoop.mapred.MapReduceBase;

    import org.apache.hadoop.mapred.OutputCollector;

    import org.apache.hadoop.mapred.Reducer;

    import org.apache.hadoop.mapred.Reporter;

    public class WC\_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,

     Reporter reporter) throws IOException {

    int sum=0;

    while (values.hasNext()) {

    sum+=values.next().get();

    }

    output.collect(key,new IntWritable(sum));

    }

    }

**Create the Runner class (WC\_Runner.java)**

import java.io.IOException;

    import org.apache.hadoop.fs.Path;

    import org.apache.hadoop.io.IntWritable;

    import org.apache.hadoop.io.Text;

    import org.apache.hadoop.mapred.FileInputFormat;

    import org.apache.hadoop.mapred.FileOutputFormat;

    import org.apache.hadoop.mapred.JobClient;

    import org.apache.hadoop.mapred.JobConf;

    import org.apache.hadoop.mapred.TextInputFormat;

    import org.apache.hadoop.mapred.TextOutputFormat;

    public class WC\_Runner {

        public static void main(String[] args) throws IOException{

            JobConf conf = new JobConf(WC\_Runner.class);

            conf.setJobName("WordCount");

            conf.setOutputKeyClass(Text.class);

            conf.setOutputValueClass(IntWritable.class);

            conf.setMapperClass(WC\_Mapper.class);

            conf.setCombinerClass(WC\_Reducer.class);

            conf.setReducerClass(WC\_Reducer.class);

            conf.setInputFormat(TextInputFormat.class);

            conf.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.setInputPaths(conf,new Path(args[0]));

            FileOutputFormat.setOutputPath(conf,new Path(args[1]));

            JobClient.runJob(conf);

        }

    }

1. **Compile the three Java files**

cd dossier1

javac \*.java $(hadoop classpath)

“javac -cp "/usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/*:/usr/lib/hadoop-hdfs/\*" \*.java“

ls -l

WC\_Mapper.class

WC\_Reducer .class

WC\_Runner .class

Create a Java Archive File (jar cf, avec c = create, f = file) from the three class files. Then list the manifest (jar tf) of the archive file:

jar cf Results.jar \*.class

ls

jar tf \*.jar

The Java Archive File was created in the directory where the .java and .class files reside. But when we use Hadoop MapReduce to run the jar, Hadoop does not like to have the .class files in the same directory.

Therefore you want to move the file to the parent directory, where we will run it in the next step:

cp \*.jar ..

cd ..

1. **Run the JAR file**

hadoop jar ./Results.jar WC\_Runner sampledata/data.txt sampledata/Result\_out

1. **Vérification**

hdfs dfs -cat sampledata/Result\_out/part-00000

**Avec eclipse**

Lancer eclipse

Réécris les trois classes java

WC\_Mapper.java

WC\_Reducer .java

 WC\_Runner .java

**Suivez cette vidéo**

<https://www.youtube.com/watch?v=VzKGdM4hc74&ab_channel=BigData101>

**Mapreduce word count with python**

<https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/>

**Step 1:**Create a file (*data.txt* ) in a directorie and add some data to it.

cd Documents/

touch data.txt # touch is used to create an empty file

nano data.txt # to edit the file

cat data.txt # to see the content of the file

**Step 2:** Create a **mapper.py**file that implements the mapper.

It will read the data from STDIN and will split the lines into words,

it will generate an output of each word with its individual count.

touch mapper.py fichier vide

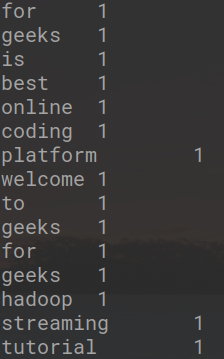
cat mapper.py

Copy the below code to the *mapper.py* file.

|  |
| --- |
| #!/usr/bin/env python    # import sys because we need to read and write data to STDIN and STDOUT  import sys  # reading entire line from STDIN (standard input)  for line in sys.stdin:      # to remove leading and trailing whitespace      line = line.strip()      # split the line into words      words = line.split()        # we are looping over the words array and printing the word      # with the count of 1 to the STDOUT      for word in words:       # write the results to STDOUT (standard output);          # what we output here will be the input for the Reduce step, i.e. the input for reducer.py          print '%s\t%s' % (word, 1) |

**Test du mapper en local (sur linux)**

cat data.txt | python mapper.py



**Step 3:**Create a ***reducer.py***file that implements the reducer logic.

It will read the output of mapper.py from STDIN(standard input)

It will aggregate the occurrence of each word

It will write the final output to STDOUT.

touch reducer.py

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# read the entire line from STDIN

for line in sys.stdin:

    # remove leading and trailing whitespace

    line = line.strip()

    # splitting the data on the basis of tab we have provided in mapper.py

    word, count = line.split('\t', 1)

    # convert count (currently a string) to int

    try:

        count = int(count)

    except ValueError:

        # count was not a number, so silently ignore/discard this line

        continue

    # this IF-switch only works because Hadoop sorts map output by key (here: word) before it is passed to the reducer

    if current\_word == word:

        current\_count += count

    else:

        if current\_word:

            # write result to STDOUT

            print '%s\t%s' % (current\_word, current\_count)

        current\_count = count

        current\_word = word

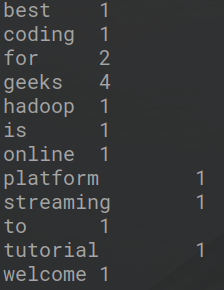
# do not forget to output the last word if needed!

if current\_word == word:

    print '%s\t%s' % (current\_word, current\_count)

Now let’s check the reducer code reducer.py with mapper.py en local (sur linux)

cat data.txt | python mapper.py | sort -k1,1 | python reducer.py



We can see that the reducer is also working fine in local system.

**Step 4:** create a directory **word\_count\_in\_python** in the HDFS and store our **file data.txt.**

hdfs dfs -mkdir /word\_count\_in\_python

Copy **data.txt**to this folder in our HDFS with help of [**copyFromLocal**](https://www.geeksforgeeks.org/hadoop-copyfromlocal-command/) command.

hdfs dfs -copyFromLocal documents/data.txt /word\_count\_in\_python

hdfs dfs -ls / # list down content of the root directory

hdfs dfs -ls word\_count\_in\_python # list content of word\_count\_in\_python directory

cd Documents/

chmod 777 mapper.py reducer.py # changing the permission to read, write, execute for user, group and others

**Step 5:  Final Step (téléchargement de la bibliothèque hadoop-streaming.jar** )

Now download the latest **hadoop-streaming jar.**

Then place, this Hadoop-streaming jar file to a place from you can easily access it. (***for example*** in ***Documents, with***  **mapper.py** and **reducer.py** files.

**Basic options that we can use with hadoop-streaming.jar**

| **Option** | **Description** |
| --- | --- |
| -mapper | The command to be run as the mapper |
| -reducer | The command to be run as the reducer |
| -input | The DFS input path for the Map step |
| -output | The DFS output directory for the Reduce step |

**Running**

hadoop jar /home/dikshant/Documents/hadoop-streaming-2.7.3.jar

-input word\_count\_in\_python/word\_count\_data.txt \

-output word\_count\_in\_python/output \

-mapper Documents/mapper.py \

-reducer Documents/reducer.py

hadoop jar /home/cloudera/dossier1/TpPython/hadoop-streaming-3.4.1.jar -files /home/cloudera/dossier1/TpPython/mapper.py,/home/cloudera/dossier1/TpPython/reducer.py -input word\_count\_in\_python/data.txt -output word\_count\_in\_python/output113 -mapper "python mapper.py" -reducer "python reducer.py"

let’s check our output in output file at location ***/word\_count\_in\_python/output/part-00000***.

hdfs dfs -cat /word\_count\_in\_python/output/part-00000

hdfs dfs -cat word\_count\_in\_python/output113/part-00000

