Hadoop Lab

- **You need only answer the 2 questions marked as such.**
- **Answers must be provided by completing the file:** `answer-namelist-hadoop-2022.txt`**. (*namelist* lists the surname of students in the group. As in: answer-groz-maniu-hadoop-2022.txt)**
- **The file must be returned on ecampus.**
- **The other instructions are just meant to help you understand what's going on.**

1 Environment

We will run a toy Hadoop application on a single node. A better option would have been to use the docker image from `bde2020` and deploy a cluster of containers with docker-compose. But this goes beyond the scope of this project and the machines at university do not support docker-compose. Therefore we install a lighter image `sequenceiq/hadoop-docker:2.7.0` which runs directly hadoop within a single container. This image is rather outdated, rather limited, and only features Python 2.6.
As indicated in the image's webpage, it should be launched as follows :

```
docker run -it sequenceiq/hadoop-docker:2.7.0 /etc/bootstrap.sh -bash
```

1.1 Setting up Hadoop in pseudo-dostributed mode

In this mode we use one Java process per Hadoop daemon (unline *standalone* mode), but we do not have a real cluster.

You can observe in the log messages that print after launching the process that the container runs a namenode, a secundary namenode, and a datanode. But those 3 nodes are located on the same container.

You can check with : `bin/hdfs dfsadmin -report` and `bin/hdfs getconf -namenodes`

1. The Hadoop executables are located in directory `$HADOOP_PREFIX`. Check where this directory is located and **make it the current directory**.
2. With a quick look at configuration files : `etc/hadoop/core-site.xml` and `etc/hadoop/hdfs-site.xml`
   - check that we are using HDFS, and not the local filesystem as in standalone mode.
   - check the default replication factor.
3. Then check which Java processes are running with `jps`. You should see : Jps, DataNode, NameNode, SecondaryNameNode, NodeManager, ResourceManager.

1.2 Managing files with HDFS

1. Move the text file `words.txt` from your computer to the container. `docker cp words.txt myhadoop:/`.
2. Create a (distributed) directory in hdfs `bin/hadoop fs -mkdir /rep`[1].
3. Copy the file from the local file system to hdfs : `bin/hadoop fs -put /words.txt /rep/words.txt`.

   You can then check with `hadoop fs -ls /rep` and `hadoop fs -tail /rep` (file is a bit large for `cat`)

[1] Here and below, you may alternatively use `bin/hdfs dfs` instead of `bin/hadoop fs` since these are instructions dealing with hdfs and not an arbitrary file system.

2 Hadoop Streaming: running a MapReduce job defined with executables

https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html

Our purpose in this section is to execute a mapreduce job defined with arbitrary executables. First with shell scripts, then with two python scripts; `mapper.py` and `reducer.py`.

4. Let's first run the Hadoop Streaming utility with a Java class: `org.apache.hadoop.mapred.lib.IdentityMapper` and a bash scripts: `/usr/bin/wc`.

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.7.0.jar \
  -input /rep \
  -output outputdir0 \
  -inputformat org.apache.hadoop.mapred.KeyValueTextInputFormat \
  -mapper org.apache.hadoop.mapred.lib.IdentityMapper \
  -reducer /usr/bin/wc
```

> The output directory will be created by the Job. It should not pre-exist.

If this fails by getting stuck with a message like "Running job: job_xxxx" and nothing happens for more than 1 minute, I just don't know how to fix that (it's a rather old image, people nowadays would rather use the bde2020 image, but it requires docker compose). In that case just skip this one question (you may indicate in your answer that your job got stuck in this way), but you can and should still answer the following questions as you can simulate mapreduce and its shuffle step using a unix pipe, as we discuss next.

Question 1: (not graded) What is the result of this job, and how did you observe it ? If you were stuck due to the issue mentioned above, just indicate so and don't answer this question.

5.

> It's a good idea to check locally your pipeline before running it on map reduce, with the instruction below. A priori, if you restricted yourself to the instructions above, you probably cannot run the pipeline, neither on the namenode (no Python) nor on nodemanager (no files), but you can easily run the pipeline on your own computer outside of docker, or use any a container with both python and the scripts (adapting one of the above or creating another one).
> `echo 'we know what we are, but not what we may be' \`
> ` | ./mapper.py | sort -t 1 | ./reducer.py`
> Of course, use `cat` instead of `echo` when the input is a file.
> When testing with a unix pipe, it is better to use the reduced file miniword, which only contains words starting with an "a", that can be obtained using `sed -Ei.old '/.* [b-zB-Z]/d' miniwords.txt`

If your job got stuck as described in the previous question, just run this job locally with a unix pipe (on the reduced dataset), as discussed above. Otherwise, put the Python scripts `mapper.py` and `reducer.py` on the namenode (in our case, that just means on the container). Make sure they are executable (otherwise : `chmod a+x mapper.py`...)

Then execute a MapReduce job that uses these mapper and reducer, adapting the instructions below if needed :

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.7.0.jar
  -input /rep\
  -output outputdir1 \
  -inputformat org.apache.hadoop.mapred.KeyValueTextInputFormat \
  -mapper /mapper.py \
  -reducer /reducer.py
```

3 Write your own MapReduce job

The file `words.txt` provided with these instructions contains words from multiple langage[2]s. Each line is a pair: `language word` as illustrated below. Neither word nor language contain any space:

[2] This file was obtained by joining files from https://github.com/lorenbrichter/Words/tree/master/Words

```
deutsch Aachen
français abaque
english aardvark
...
```

Question 2: Write a Job that processes the file `words.txt` to compute the words that appear at least once in both french and english (each word should of course appear once only in the result). This means you are computing the intersection of the two languages. Your program should write the result as a list of pairs (w,0) where w is a word that appears in both french and english.

- Your Job should be efficient even with much larger files (be reasonably clever about your map and reduce. Don't just purposedly send the whole data to one single reducer).
- Indication: you will have to deal with one non-ascii caracter. Since by default you will probably be using python2, the header of your python file should probably be :

  ```
  #!/usr/bin/env python
  # -*- coding: utf-8 -*-
  ```

- Ideally you should not assume that each word appears only once in each language. Your program should able to accept a word x that occurs twice with "français" and once with "english", for instance.