



## TP 1 : Hadoop

### Installation :

Nous allons utiliser tout au long de ce TP trois conteneurs représentant respectivement un noeud maître (Namenode) et deux noeuds esclaves (Datanodes).

Après le lancement de Docker, ouvrir la ligne de commande et taper les instructions suivantes :

1. Télécharger l'image docker uploadée sur dockerhub:

```
docker pull liliasfaxi/spark-hadoop:hv-2.7.2
```

2. Créer les trois conteneurs à partir de l'image téléchargée. Pour cela:

- 2.1. Créer un réseau qui permettra de relier les trois conteneurs :

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

- 2.2. Créer et lancer les trois conteneurs (les instructions -p permettent de faire un mapping entre les ports de la machine hôte et ceux du conteneur):

```
docker run -itd --net=hadoop -p 50070:50070 -p 8088:8088 -p 7077:7077 -p 16010:16010 ^
```

```
--name hadoop-master --hostname hadoop-master ^
```

```
liliasfaxi/spark-hadoop:hv-2.7.2
```

```
docker run -itd -p 8040:8042 --net=hadoop ^
```

```
--name hadoop-slave1 --hostname hadoop-slave1 ^
```

```
liliasfaxi/spark-hadoop:hv-2.7.2
```

```
docker run -itd -p 8041:8042 --net=hadoop ^
```

```
--name hadoop-slave2 --hostname hadoop-slave2 ^
```

```
liliasfaxi/spark-hadoop:hv-2.7.2
```

3. Entrer dans le conteneur master pour commencer à l'utiliser :

```
docker exec -it hadoop-master bash
```

Le résultat de cette exécution sera le suivant:

```
root@hadoop-master:~#
```

Vous vous retrouverez dans le shell du namenode, et vous pourrez ainsi manipuler le cluster à votre guise. La première chose à faire, une fois dans le conteneur, est de lancer hadoop et yarn. Un script est fourni pour cela, appelé start-hadoop.sh. Lancer ce script :

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

Le résultat devra ressembler à ce qui suit :

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

Starting namenodes on [hadoop-master]
hadoop-master: Warning: Permanently added 'hadoop-master,172.21.0.2' (ECDSA) to the list of known hosts.
hadoop-master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-hadoop-master.out
hadoop-slave2: Warning: Permanently added 'hadoop-slave2,172.21.0.4' (ECDSA) to the list of known hosts.
hadoop-slave1: Warning: Permanently added 'hadoop-slave1,172.21.0.3' (ECDSA) to the list of known hosts.
hadoop-slave2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-hadoop-slave2.out
hadoop-slave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-hadoop-slave1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-hadoop-master.out

starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanager-hadoop-master.out
hadoop-slave1: Warning: Permanently added 'hadoop-slave1,172.21.0.3' (ECDSA) to the list of known hosts.
hadoop-slave2: Warning: Permanently added 'hadoop-slave2,172.21.0.4' (ECDSA) to the list of known hosts.
hadoop-slave1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop-slave1.out
hadoop-slave2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop-slave2.out
```

## Premiers pas avec Hadoop :

Toutes les commandes interagissant avec le système Hadoop commencent par **hadoop fs**. Ensuite, les options rajoutées sont très largement inspirées des commandes Unix standard.

- Créer un répertoire dans HDFS, appelé input. Pour cela, taper :

```
hadoop fs -mkdir -p input
```

**Remarque :** Si pour une raison ou une autre, vous n'arrivez pas à créer le répertoire input, avec un message d'erreur ressemblant à ceci:

```
ls: `.`: No such file or directory
```

veiller à construire l'arborescence de l'utilisateur principal (root), comme suit:

```
hadoop fs -mkdir -p /user/root
```

- Nous allons utiliser le fichier **purchases.txt**<sup>1</sup> comme entrée pour le traitement MapReduce. Ce fichier se trouve déjà sous le répertoire principal de votre machine master.
- Charger le fichier **purchases** dans le répertoire input que vous avez créé :

<sup>1</sup> <https://github.com/CodeMangler/udacity-hadoop-course/raw/master/Datasets/purchases.txt.gz>

```
hadoop fs -put purchases.txt input
```

- Pour afficher le contenu du répertoire **input**, la commande est :

```
hadoop fs -ls input
```

- Pour afficher les dernières lignes du fichier **purchases** :

```
hadoop fs -tail input/purchases.txt
```

Nous présentons dans le tableau suivant les commandes les plus utilisées pour manipuler les fichiers dans HDFS :

Instruction	Fonctionnalité
hadoop fs -ls	Afficher le contenu du répertoire racine.
hadoop fs -put file.txt	Upload un fichier dans hadoop (à partir du répertoire courant linux).
hadoop fs -get file.txt	Download un fichier à partir de hadoop sur votre disque local.
hadoop fs -tail file.txt	Lire les dernières lignes du fichier.
hadoop fs -cat file.txt	Affiche tout le contenu du fichier.
hadoop fs -mv file.txt newfile.txt	Renommer le fichier.
hadoop fs -rm newfile.txt	Supprimer le fichier.
hadoop fs -mkdir myinput	Créer un repertoire.

### Interfaces web pour Hadoop :

Hadoop offre plusieurs interfaces web pour pouvoir observer le comportement de ses différentes composantes :

- Le port 50070 de la machine maître permet d'afficher les informations de votre **Namenode**.
- Le port 8088 de la machine maître permet d'afficher les informations du resource manager de Yarn et visualiser le comportement des différents jobs.

Une fois votre cluster lancé et prêt à l'emploi, vous pouvez, sur votre navigateur préféré de votre machine hôte, aller à : <http://localhost:50070> . Vous obtiendrez le résultat suivant :

Overview 'hadoop-master:9000' (active)

Started:	Fri Jan 27 19:30:19 UTC 2023
Version:	2.7.2, rUnknown
Compiled:	2016-05-27T18:05Z by root from Unknown
Cluster ID:	CID-b721bea8-93cb-45f0-9023-dff705808b00
Block Pool ID:	BP-195763961-172.17.0.3-1550840521902

Summary

Security is off.

Safemode is off.

5 files and directories, 2 blocks = 7 total filesystem object(s).

Heap Memory used 96.8 MB of 178 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 43.24 MB of 44.06 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.

Vous pouvez également visualiser l'avancement et les résultats de vos Jobs (Map Reduce ou autre) en allant à l'adresse: <http://localhost:8088> :

← → ↺ 🏠

localhost:8088/cluster

☆

📄 ⬇️ 📌 ☰

hadoop

All Applications

Logged in as: dr...

Cluster

About  
Nodes  
Node Labels  
Applications  
NEW  
NEW SAVING  
SUBMITTED  
ACCEPTED  
RUNNING  
FINISHED  
FAILED  
KILLED  
Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Reboot Nodes
0	0	0	0	0	0 B	16 GB	0 B	0	16	0	2	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show: 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
No data available in table											

Showing 0 to 0 of 0 entries

First Previous Next Last

Map Reduce :

Présentation :

Un Job Map-Reduce se compose principalement de deux types de programmes:

- Mappers** : permettent d’extraire les données nécessaires sous forme de clé/valeur, pour pouvoir ensuite les trier selon la clé.
- Reducers** : prennent un ensemble de données triées selon leur clé, et effectuent le traitement nécessaire sur ces données (somme, moyenne, total...).

Wordcount :

Nous allons tester un programme MapReduce grâce à un exemple très simple, le WordCount. Le Wordcount permet de calculer le nombre de mots dans un fichier donné, en décomposant le calcul en deux étapes :

- L'étape de Mapping, qui permet de découper le texte en mots et de délivrer en sortie un flux textuel, où chaque ligne contient le mot trouvé, suivi de la valeur 1 (pour dire que le mot a été trouvé une fois).
- L'étape de Reducing, qui permet de faire la somme des 1 pour chaque mot, pour trouver le nombre total d'occurrences de ce mot dans le texte.

Commençons par créer un projet Maven dans IntelliJ. Définir les valeurs suivantes pour votre projet:

- **GroupId:** `hadoop.mapreduce`
- **ArtifactId:** `wordcount`

Ouvrir le fichier `pom.xml`, et ajouter les dépendances suivantes pour Hadoop, HDFS et Map Reduce:

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>2.7.2</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>2.7.2</version>
  </dependency>
  <dependency>
```

```
<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-common</artifactId>

<version>2.7.2</version>

</dependency>

</dependencies>
```

- Créer un package **tp1** sous le répertoire **src/main/java**.
- Créer la classe **TokenizerMapper**, contenant ce code :

```
package tp1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.StringTokenizer;

public class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

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

    public void map(Object key, Text value, Mapper.Context context
    ) throws IOException, InterruptedException {

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

```
}  
  
}
```

- Créer la classe **IntSumReducer** :

```
package tpl;  
  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
import java.io.IOException;  
  
public class IntSumReducer  
    extends Reducer<Text,IntWritable,Text,IntWritable> {  
  
    private IntWritable result = new IntWritable();  
  
    public void reduce(Text key, Iterable<IntWritable> values,  
        Context context  
    ) throws IOException, InterruptedException {  
        int sum = 0;  
        for (IntWritable val : values) {  
            System.out.println("value: "+val.get());  
            sum += val.get();  
        }  
        System.out.println("--> Sum = "+sum);  
        result.set(sum);  
        context.write(key, result);  
    }  
}
```

```
}  
  
}
```

- Enfin, créer la classe **WordCount** :

```
package tpl;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class WordCount {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        Job job = Job.getInstance(conf, "word count");  
        job.setJarByClass(WordCount.class);  
        job.setMapperClass(TokenizerMapper.class);  
        job.setCombinerClass(IntSumReducer.class);  
        job.setReducerClass(IntSumReducer.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
        System.exit(job.waitForCompletion(true) ? 0 : 1);  
    }  
}
```

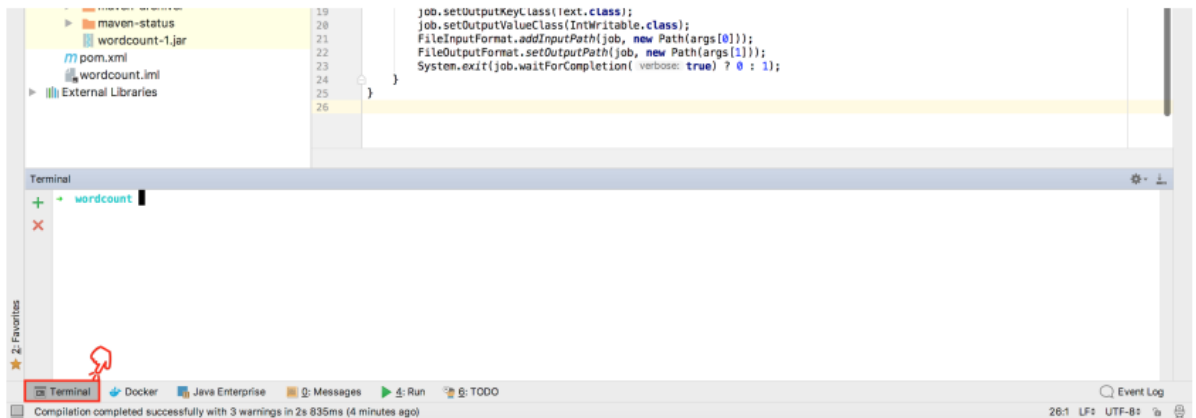


```
}
```

## Lancer Map Reduce sur le cluster :

Dans votre projet IntelliJ :

- Générer le fichier **jar** de l'application :  
Maven -> wordcount -> Lifecycle -> install -> (Right click) Run Maven Build
- Un fichier wordcount-1.0-SNAPSHOT.jar sera créé dans le répertoire **target** du projet. Renommez-le en **wordcount-1.jar**.
- Copier le fichier **jar** créé dans le conteneur **master**. Pour cela:
  - Ouvrir le terminal sur le répertoire du projet. Cela peut être fait avec IntelliJ en ouvrant la vue Terminal située en bas à gauche de la fenêtre principale :



- Taper la commande suivante :

```
docker cp target/wordcount-1.jar hadoop-master:/root/wordcount-1.jar
```

- Revenir au shell du conteneur master et lancer le job map reduce avec cette commande :

```
hadoop jar wordcount-1.jar tp1.WordCount input output
```

Le Job sera lancé sur le fichier purchases.txt que vous aviez préalablement chargé dans le répertoire input de HDFS. Une fois le Job terminé, un répertoire output sera créé. Si tout se passe bien, vous obtiendrez un affichage ressemblant au suivant :

```

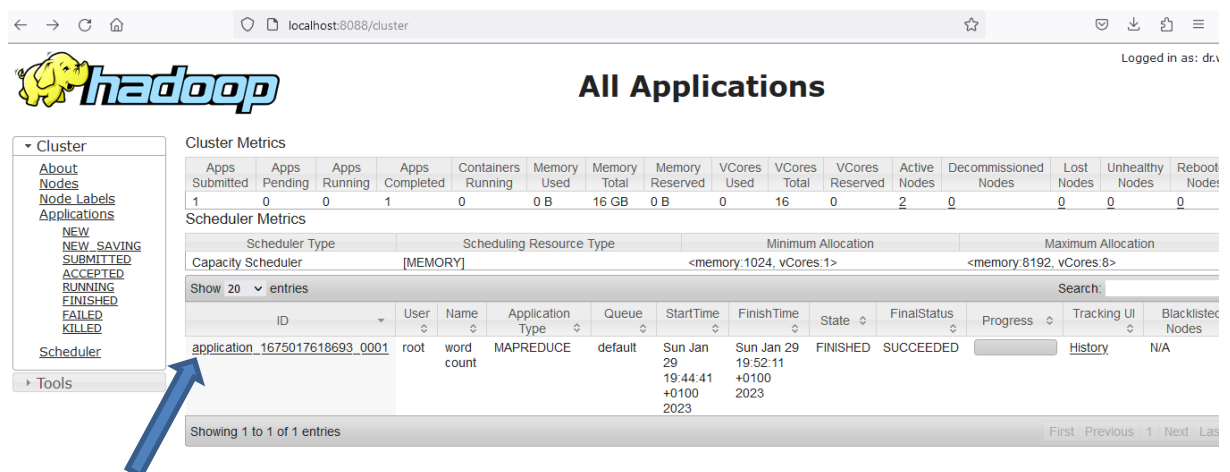
root@hadoop-master:~# hadoop jar wordcount-1.jar tp1.WordCount input output
23/01/29 18:44:37 INFO client.RMProxy: Connecting to ResourceManager at hadoop-master/172.21.0.2:8032
23/01/29 18:44:38 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
23/01/29 18:44:39 INFO input.FileInputFormat: Total input paths to process : 1
23/01/29 18:44:40 INFO mapreduce.JobSubmitter: number of splits:2
23/01/29 18:44:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1675017618693_0001
23/01/29 18:44:41 INFO impl.YarnClientImpl: Submitted application application_1675017618693_0001
23/01/29 18:44:41 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1675017618693_0001/
23/01/29 18:44:41 INFO mapreduce.Job: Running job: job_1675017618693_0001
23/01/29 18:45:02 INFO mapreduce.Job: Job job_1675017618693_0001 running in uber mode : false
23/01/29 18:45:02 INFO mapreduce.Job: map 0% reduce 0%
23/01/29 18:45:19 INFO mapreduce.Job: map 3% reduce 0%
23/01/29 18:45:22 INFO mapreduce.Job: map 17% reduce 0%
23/01/29 18:45:25 INFO mapreduce.Job: map 19% reduce 0%
23/01/29 18:46:44 INFO mapreduce.Job: map 22% reduce 0%
23/01/29 18:46:47 INFO mapreduce.Job: map 33% reduce 0%
23/01/29 18:46:50 INFO mapreduce.Job: map 35% reduce 0%
23/01/29 18:47:53 INFO mapreduce.Job: map 49% reduce 0%
23/01/29 18:47:56 INFO mapreduce.Job: map 50% reduce 0%
23/01/29 18:49:21 INFO mapreduce.Job: map 52% reduce 0%
23/01/29 18:49:26 INFO mapreduce.Job: map 57% reduce 0%
23/01/29 18:49:46 INFO mapreduce.Job: map 74% reduce 0%
23/01/29 18:50:07 INFO mapreduce.Job: map 74% reduce 17%
23/01/29 18:50:30 INFO mapreduce.Job: map 80% reduce 17%
23/01/29 18:51:20 INFO mapreduce.Job: map 83% reduce 17%
23/01/29 18:52:05 INFO mapreduce.Job: map 100% reduce 17%
23/01/29 18:52:07 INFO mapreduce.Job: map 100% reduce 73%
23/01/29 18:52:10 INFO mapreduce.Job: map 100% reduce 100%
23/01/29 18:52:13 INFO mapreduce.Job: Job job_1675017618693_0001 completed successfully
23/01/29 18:52:13 INFO mapreduce.Job: Counters: 50
File System Counters
  FILE: Number of bytes read=7684620
  FILE: Number of bytes written=9325395

```

Vous pouvez afficher le contenu du fichier généré **output/part-r-00000**, avec :

```
hadoop fs -cat output/part-r-00000
```

Il vous est possible de monitorer vos Jobs Map Reduce, en allant à la page: <http://localhost:8088> . Vous trouverez votre Job dans la liste des applications comme suit :



The screenshot shows the Hadoop YARN web interface. On the left is a sidebar with navigation links like 'Cluster', 'About Nodes', 'Node Labels', 'Applications', and 'Tools'. The main area is titled 'All Applications' and displays 'Cluster Metrics' and 'Scheduler Metrics'. Below these is a table of applications. A blue arrow points to the application 'application\_1675017618693\_0001'.

Apps		Apps		Apps		Containers		Memory		Memory		Memory		VCores		VCores		VCores		Active		Decommissioned		Lost		Unhealthy		Reboot	
Submitted	Pending	Running	Completed	Running	Used	Total	Reserved	Used	Total	Reserved	Used	Total	Reserved	Used	Total	Reserved	Used	Total	Reserved	Nodes	Nodes	Nodes	Nodes	Nodes	Nodes	Nodes	Nodes	Nodes	
1	0	0	1	0	0 B	16 GB	0 B	0	16	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation					
Capacity Scheduler		[MEMORY]		<memory:1024, vCores:1>		<memory:8192, vCores:8>					
Show 20 entries											
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1675017618693_0001	root	word count	MAPREDUCE	default	Sun Jan 29 19:44:41 +0100 2023	Sun Jan 29 19:52:11 +0100 2023	FINISHED	SUCCEEDED		History	N/A

Showing 1 to 1 of 1 entries

Il est également possible de voir le comportement des nœuds esclaves, en allant à l'adresse: <http://localhost:8040> pour slave1, et <http://localhost:8041> pour slave2.