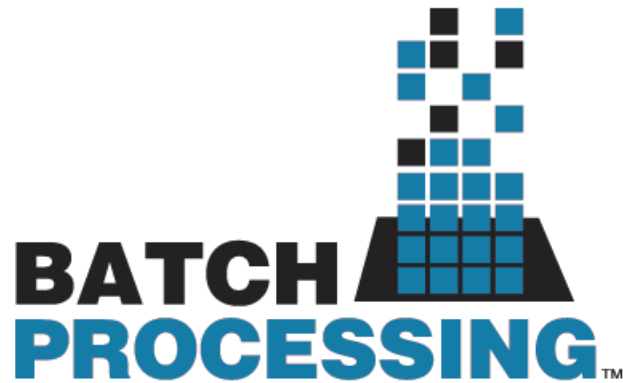


TP1 - Le traitement Batch avec Hadoop HDFS et Map Reduce



Télécharger PDF



Objectifs du TP

Initiation au framework hadoop et au patron MapReduce, utilisation de docker pour lancer un cluster hadoop de 3 noeuds.

Outils et Versions

- [Apache Hadoop](#) Version: 3.3.6.
- [Docker](#) Version *latest*
- [Visual Studio Code](#) Version 1.85.1 (ou tout autre IDE de votre choix)
- [Java](#) Version 1.8.
- Unix-like ou Unix-based Systems (Divers Linux et MacOS)

Hadoop

Présentation

[Apache Hadoop](#) est un framework open-source pour stocker et traiter les données volumineuses sur un cluster. Il est utilisé par un grand nombre de contributeurs et utilisateurs. Il a une licence Apache 2.0.



Hadoop et Docker

Pour déployer le framework Hadoop, nous allons utiliser des conteneurs [Docker](#). L'utilisation des conteneurs va garantir la consistance entre les environnements de développement et permettra de réduire considérablement la complexité de

configuration des machines (dans le cas d'un accès natif) ainsi que la lourdeur d'exécution (si on opte pour l'utilisation d'une machine virtuelle).

Nous avons pour le déploiement des ressources de ce TP suivi les instructions présentées [ici](#).

Installation

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

Vous devez pour cela avoir installé docker sur votre machine, et l'avoir correctement configuré. Ouvrir la ligne de commande, et taper les instructions suivantes:

1. Télécharger l'image docker uploadée sur dockerhub:

```
docker pull liliasfazi/hadoop-cluster:latest
```

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 9870:9870 -p 8088:8088 -p 7077:7077 -p 16010:16010 --name hadoop-master --hostname hadoop-master liliasfazi/hadoop-cluster:latest
```

```
docker run -itd -p 8040:8042 --net=hadoop --name hadoop-worker1 --hostname hadoop-worker1 liliasfazi/hadoop-cluster:latest
```

```
docker run -itd -p 8041:8042 --net=hadoop --name hadoop-worker2 --hostname hadoop-worker2 liliasfazi/hadoop-cluster:latest
```

- 2.3. Vérifier que les trois conteneurs tournent bien en lançant la commande `docker ps`. Un résultat semblable au suivant devra s'afficher:

```
+ ~ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
b0c4f0a988f6   liliasfazi/hadoop-cluster:latest    "sh -c 'service ssh ..." 4 seconds ago  Up 2 seconds  0.0.0.0:8041->8042/tcp
7e97824688a0   liliasfazi/hadoop-cluster:latest    "sh -c 'service ssh ..." 16 seconds ago Up 13 seconds  0.0.0.0:8040->8042/tcp
b2fc11cbe973   liliasfazi/hadoop-cluster:latest    "sh -c 'service ssh ..." 27 seconds ago Up 25 seconds  0.0.0.0:8088->8088/tcp, 0.0.0.0:16010->16010/tcp, 0.0.0.0:50070->50070/tcp
hadoop-master  hadoop-master
```

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' (ED25519) to the list of known hosts.
hadoop-master: WARNING: HADOOP_NAMENODE_OPTS has been replaced by HDFS_NAMENODE_OPTS. Using value of HADOOP_NAMENODE_OPTS.
Starting datanodes
WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_DIR.
hadoop-worker1: Warning: Permanently added 'hadoop-worker1' (ED25519) to the list of known hosts.
hadoop-worker2: Warning: Permanently added 'hadoop-worker2' (ED25519) to the list of known hosts.
hadoop-worker1: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_D
R.
hadoop-worker1: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
hadoop-worker2: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_D
R.
hadoop-worker2: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
Starting secondary namenodes [hadoop-master]
hadoop-master: Warning: Permanently added 'hadoop-master' (ED25519) to the list of known hosts.
hadoop-master: WARNING: HADOOP_SECONDARYNAMENODE_OPTS has been replaced by HDFS_SECONDARYNAMENODE_OPTS. Using value of HADOOP_SECON
DARYNAMENODE_OPTS.

Starting resourcemanager
Starting nodemanagers
hadoop-worker1: Warning: Permanently added 'hadoop-worker1' (ED25519) to the list of known hosts.
hadoop-worker2: Warning: Permanently added 'hadoop-worker2' (ED25519) to the list of known hosts.
```

Premiers pas avec Hadoop

Toutes les commandes interagissant avec le système HDFS commencent par `hdfs dfs`. Ensuite, les options rajoutées sont très largement inspirées des commandes Unix standard.

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

```
hdfs dfs -mkdir -p input
```

En cas d'erreur: *No such file or directory*

Si pour une raison ou une autre, vous n'arrivez pas à créer le répertoire *input*, avec un message ressemblant à ceci: `ls: .: No such file or directory`, veillez à construire l'arborescence de l'utilisateur principal (root), comme suit:

```
hdfs dfs -mkdir -p /user/root
```

- Nous allons utiliser le fichier [purchases.txt](#) 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:

```
hdfs dfs -put purchases.txt input
```

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

```
hdfs dfs -ls input
```

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

```
hdfs dfs -tail input/purchases.txt
```

Le résultat suivant va donc s'afficher:

```

root@hadoop-master:~# hdfs dfs -tail input/purchases.txt
31      17:59  Norfolk Toys    164.34 MasterCard
2012-12-31  17:59  Chula Vista    Music    380.67 Visa
2012-12-31  17:59  Hialeah Toys   115.21 MasterCard
2012-12-31  17:59  Indianapolis    Men's Clothing 158.28 MasterCard
2012-12-31  17:59  Norfolk Garden 414.09 MasterCard
2012-12-31  17:59  Baltimore      DVDs     467.3  Visa
2012-12-31  17:59  Santa Ana      Video Games 144.73 Visa
2012-12-31  17:59  Gilbert Consumer Electronics 354.66 Discover
2012-12-31  17:59  Memphis Sporting Goods 124.79 Amex
2012-12-31  17:59  Chicago Men's Clothing 386.54 MasterCard
2012-12-31  17:59  Birmingham     CDs      118.04 Cash
2012-12-31  17:59  Las Vegas      Health and Beauty 420.46 Amex
2012-12-31  17:59  Wichita Toys   383.9   Cash
2012-12-31  17:59  Tucson Pet Supplies 268.39 MasterCard
2012-12-31  17:59  Glendale       Women's Clothing 68.05 Amex
2012-12-31  17:59  Albuquerque    Toys     345.7  MasterCard
2012-12-31  17:59  Rochester      DVDs     399.57 Amex
2012-12-31  17:59  Greensboro     Baby     277.27 Discover
2012-12-31  17:59  Arlington      Women's Clothing 134.95 MasterCard
2012-12-31  17:59  Corpus Christi DVDs     441.61 Discover

```

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

| Instruction | Fonctionnalité |
|--|--|
| <code>hdfs dfs -ls</code> | Afficher le contenu du répertoire racine |
| <code>hdfs dfs -put file.txt</code> | Upload un fichier dans hadoop (à partir du répertoire courant de votre disque local) |
| <code>hdfs dfs -get file.txt</code> | Download un fichier à partir de hadoop sur votre disque local |
| <code>hdfs dfs -tail file.txt</code> | Lire les dernières lignes du fichier |
| <code>hdfs dfs -cat file.txt</code> | Affiche tout le contenu du fichier |
| <code>hdfs dfs -mv file.txt newfile.txt</code> | Renommer (ou déplacer) le fichier |
| <code>hdfs dfs -rm newfile.txt</code> | Supprimer le fichier |
| <code>hdfs dfs -mkdir myinput</code> | Créer un répertoire |

Interfaces web pour Hadoop

Hadoop offre plusieurs interfaces web pour pouvoir observer le comportement de ses différentes composantes. Il est possible d'afficher ces pages directement sur notre machine hôte, et ce grâce à l'utilisation de l'option `-p` de la commande `docker run`. En effet, cette option permet de publier un port du conteneur sur la machine hôte. Pour pouvoir publier tous les ports exposés, vous pouvez lancer votre conteneur en utilisant l'option `-P`.

En regardant la commande `docker run` utilisée plus haut, vous verrez que deux ports de la machine maître ont été exposés:

- Le port **9870**: qui permet d'afficher les informations de votre namenode.

- Le port **8088**: qui permet d'afficher les informations du resource manager de Yarn et visualiser le comportement des différents jobs.

Une fois votre cluster lancé et hadoop démarré et prêt à l'emploi, vous pouvez, sur votre navigateur préféré de votre machine hôte, aller à : <http://localhost:9870>. Vous obtiendrez le résultat suivant:



Overview 'hadoop-master:9000' (✓active)

| | |
|-----------------------|--|
| Started: | Wed Jan 03 20:57:28 +0100 2024 |
| Version: | 3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c |
| Compiled: | Sun Jun 18 09:22:00 +0100 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1) |
| Cluster ID: | CID-9877bcd0-4e0e-4618-9f63-bc0cb72f991c |
| Block Pool ID: | BP-16431537-127.0.0.1-1704273202935 |

Summary

Security is off.

Safemode is off.

5 files and directories, 2 blocks (2 replicated blocks, 0 erasure coded block groups) = 7 total filesystem object(s).

Heap Memory used 173.91 MB of 310.5 MB Heap Memory. Max Heap Memory is 1.73 GB.

Non Heap Memory used 55.57 MB of 56.73 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

| | |
|------------------------------------|-----------|
| Configured Capacity: | 116.73 GB |
| Configured Remote Capacity: | 0 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>.

The screenshot shows the Hadoop All Applications page. At the top, there's a navigation bar with the Hadoop logo and the title 'All Applications'. Below this, there's a 'Cluster Metrics' table with columns for various application states and resource usage. The table shows 0 apps submitted, 0 pending, 0 running, 0 completed, 0 containers running, 0 B memory used, 16 GB memory total, 0 B memory reserved, 0 VCores used, 16 VCores total, 0 VCores reserved, 2 active nodes, 0 decommissioned nodes, 0 lost nodes, 0 unhealthy nodes, and 0 rebooted nodes. Below the cluster metrics, there's a 'Scheduler Metrics' section showing the scheduler type as 'Capacity Scheduler' and the resource type as '[MEMORY]'. The minimum allocation is '<memory:1024, vCores:1>' and the maximum allocation is '<memory:8192, vCores:32>'. There's also a table for application entries, but it shows 'No data available in table'.

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 clef/valeur, pour pouvoir ensuite les trier selon la clef
- **Reducers** : prennent un ensemble de données triées selon leur clef, 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*, l'équivalent du *HelloWorld* pour les applications de traitement de données. 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 VSCode. **Nous utiliserons dans notre cas JDK 1.8.**

Version de JDK

Ceci n'est pas une suggestion: l'utilisation d'une autre version que 1.8 provoquera des erreurs sans fin. Hadoop est compilé avec cette version de Java, connue pour sa stabilité.

Pour créer un projet Maven dans VSCode:

- Prenez soin d'avoir les extensions *Maven for Java* et *Extension Pack for Java* activées.
- Créer un nouveau répertoire dans lequel vous incluez votre code.
- Faites un clic-droit dans la fenêtre *Explorer* et choisir *Create Maven Project*.
- Choisir *No Archetype*
- Définir les valeurs suivantes pour votre projet:
 - **GroupId:** `hadoop.mapreduce`
 - **ArtifactId:** `wordcount`
 - **Version:** `1`
- Ouvrir le fichier *pom.xml* automatiquement créé, et ajouter les dépendances suivantes pour Hadoop, HDFS et Map Reduce:

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>3.3.6</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>3.3.6</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>3.3.6</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-common -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-common</artifactId>
    <version>3.3.6</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-jobclient -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-jobclient</artifactId>
  </dependency>
</dependencies>
```

```
<version>3.3.6</version>
</dependency>

</dependencies>
```

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

```
package hadoop.mapreduce.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 hadoop.mapreduce.tp1;

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 hadoop.mapreduce.tp1;

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);
    }
}

```

TESTER MAP REDUCE EN LOCAL

Dans votre projet sur VSCode:


- Créer un répertoire *input* sous le répertoire *resources* de votre projet.
- Créer un fichier de test: *file.txt* dans lequel vous insèrerez les deux lignes:

```

Hello Wordcount!
Hello Hadoop!

```

- Nous allons maintenant définir des arguments à la méthode Main: le fichier en entrée sur lequel Map reduce va travailler, et le répertoire en sortie dans lequel le résultat sera stocké. Pour cela:

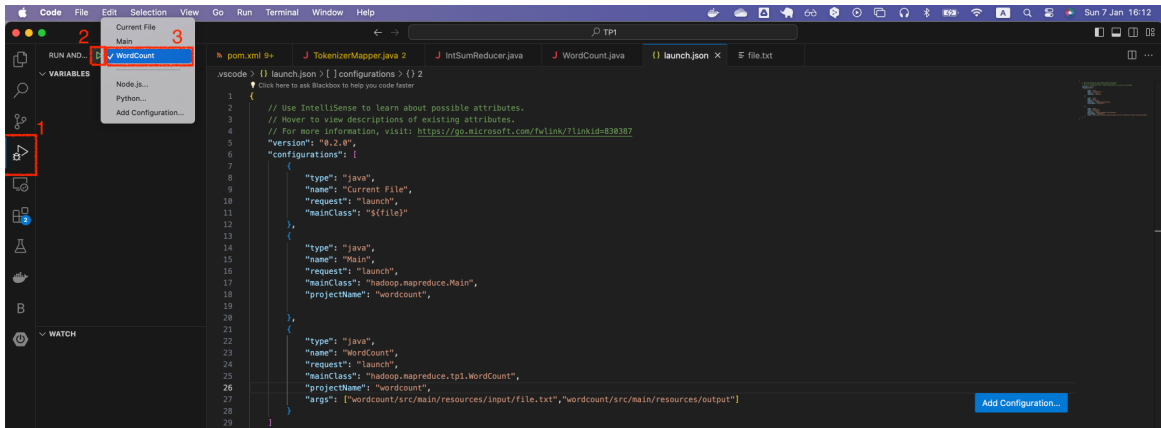
- Ouvrir le fichier *launch.json* de votre projet (Aller à la fenêtre *Run and Debug* , puis cliquer sur *create a launch.json file*).
- Ajouter la ligne suivante dans la configuration **WordCount**, dont la classe principale est *hadoop.mapreduce.tp1.WordCount*:

```
"args": ["wordcount/src/main/resources/input/file.txt", "wordcount/src/main/resources/output"]
```

Arguments

Il est à noter que, dans mon cas, le fichier *launch.json* a été créé sous le répertoire *TP1*, c'est pour cette raison que le chemin des fichiers commence par "wordcount". Si vous créez la configuration directement sous le répertoire *wordcount*, il faudra commencer le chemin par *src*.

- Sélectionner ensuite, dans la liste des configurations du projet, *WordCount* comme configuration par défaut:



- Lancer le programme. Un répertoire *output* sera créé dans le répertoire *resources*, contenant notamment un fichier *part-r-00000*, dont le contenu devrait être le suivant:

```

Hadoop! 1
Hello 2
Wordcount! 1

```

LANCER MAP REDUCE SUR LE CLUSTER

Dans votre projet VSCode:

- Pour pouvoir encapsuler toutes les dépendances du projet dans le fichier JAR à exporter, ajouter le plugin suivant dans le fichier *pom.xml* de votre projet:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.6.0</version> <!-- Use latest version -->
      <configuration>
        <archive>
          <manifest>
            <mainClass>hadoop.mapreduce.tp1.WordCount</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id> <!-- this is used for inheritance merges -->
          <phase>package</phase> <!-- bind to the packaging phase -->
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

- Aller dans l'Explorateur, sous Maven, puis ouvrir le *Lifecycle* du projet wordcount.
- Cliquer sur `package` pour compiler et packager le projet dans un fichier JAR. Un fichier *wordcount-1.0-SNAPSHOT-jar-with-dependencies.jar* sera créé sous le répertoire *target* du projet.
- Copier le fichier jar créé dans le conteneur master. Pour cela:

- Ouvrir le terminal sur le répertoire du projet *wordcount*. Cela peut être fait avec VSCode en allant au menu *Terminal -> New Terminal*.
- Taper la commande suivante:

```
docker cp target/wordcount-1.0-SNAPSHOT-jar-with-dependencies.jar hadoop-master:/root/wordcount.jar
```

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

```
hadoop jar wordcount.jar 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.jar input output
2024-01-07 17:16:01,315 INFO client.DefaultHadoopFallbackProxyProvider: Connecting to ResourceManager at hadoop-master/172.22.0.2:8032
2024-01-07 17:16:01,811 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2024-01-07 17:16:01,835 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1704647694267_0001
2024-01-07 17:16:02,935 INFO mapreduce.Job: input.FileInputFormat: Total input files to process: 1
2024-01-07 17:16:03,140 INFO mapreduce.JobSubmitter: number of splits:2
2024-01-07 17:16:03,463 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1704647694267_0001
2024-01-07 17:16:03,463 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-01-07 17:16:03,719 INFO conf.Configuration: resource-types.xml not found
2024-01-07 17:16:03,719 INFO resource.ResourceUtiliz: Unable to find 'resource-types.xml',
2024-01-07 17:16:04,066 INFO impl.YarnClientImpl: Submitted application application_1704647694267_0001
2024-01-07 17:16:04,122 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1704647694267_0001/
2024-01-07 17:16:04,123 INFO mapreduce.Job: Running job: job_1704647694267_0001
2024-01-07 17:16:04,123 INFO mapreduce.Job: Job job_1704647694267_0001 running in uber mode : false
2024-01-07 17:16:28,857 INFO mapreduce.Job: map 0% reduce 0%
2024-01-07 17:16:56,443 INFO mapreduce.Job: map 7% reduce 0%
2024-01-07 17:16:57,450 INFO mapreduce.Job: map 19% reduce 0%
2024-01-07 17:17:29,737 INFO mapreduce.Job: map 35% reduce 0%
2024-01-07 17:17:44,931 INFO mapreduce.Job: map 44% reduce 0%
2024-01-07 17:17:51,017 INFO mapreduce.Job: map 53% reduce 0%
2024-01-07 17:18:09,123 INFO mapreduce.Job: map 53% reduce 0%
2024-01-07 17:18:15,175 INFO mapreduce.Job: map 57% reduce 0%
2024-01-07 17:18:19,265 INFO mapreduce.Job: map 74% reduce 0%
2024-01-07 17:19:36,537 INFO mapreduce.Job: map 74% reduce 17%
2024-01-07 17:18:45,599 INFO mapreduce.Job: map 80% reduce 17%
2024-01-07 17:19:10,179 INFO mapreduce.Job: map 83% reduce 17%
2024-01-07 17:19:29,462 INFO mapreduce.Job: map 100% reduce 17%
2024-01-07 17:19:31,463 INFO mapreduce.Job: map 100% reduce 67%
2024-01-07 17:19:33,502 INFO mapreduce.Job: map 100% reduce 100%
2024-01-07 17:19:33,519 INFO mapreduce.Job: Job job_1704647694267_0001 completed successfully
2024-01-07 17:19:34,032 INFO mapreduce.Job: Counters: 55

File System Counters
  FILE: Number of bytes read=7694620
  FILE: Number of bytes written=903363
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=21517260
  HDFS: Number of bytes written=499048
  HDFS: Number of read operations=11
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
  HDFS: Number of bytes read erasure-coded=0

Job Counters
  Killed map tasks=1
  Launched map tasks=3
  Launched reduce tasks=1
  Data-local map tasks=3
  Total time spent by all maps in occupied slots (ms)=371193
  Total time spent by all reduces in occupied slots (ms)=71583
  Total time spent by all map tasks (ms)=371193
  Total time spent by all reduce tasks (ms)=71583
  Total vcore-milliseconds taken by all map tasks=371193
  Total vcore-milliseconds taken by all reduce tasks=71583
  Total megabyte-milliseconds taken by all map tasks=380101632
  Total megabyte-milliseconds taken by all reduce tasks=73300992

Map-Reduce Framework
  Map input records=4138476
  Map output records=27932895
  Map output bytes=223244504
  Map output materialized bytes=1299264
  Input split bytes=240
  Combine input records=28488079
  Combine output records=606926
  Reduce input groups=31893
  Reduce shuffle bytes=1289264
  Reduce input records=101742
  Reduce output records=51053
  Spilled Records=708668
  Shuffled Maps =2
  Failed Shuffles=0
  Merged Map outputs=2
  GC time elapsed (ms)=6070
  CPU time spent (ms)=244370
  Physical memory (bytes) snapshot=1347715972
  Virtual memory (bytes) snapshot=7825465344
  Total committed heap usage (bytes)=1117782016
  Peak Map Physical memory (bytes)=526889968
  Peak Map Virtual memory (bytes)=280922688
  Peak Reduce Physical memory (bytes)=38588384
  Peak Reduce Virtual memory (bytes)=2611548160
```

En affichant les dernières lignes du fichier généré *output/part-r-00000*, avec `hdfs dfs -tail output/part-r-00000`, vous obtiendrez l'affichage suivant:

```

Santa 40306
Scottsdale 40173
Seattle 39866
Spokane 40222
Sporting 229932
Springs 40389
St. 80075
Stockton 39996
Supplies 229222
Tampa 40136
Toledo 40139
Toys 229964
Tucson 39870
Tulsa 40247
Vegas 80178
Video 230237
Virginia 40169
Visa 827221
Vista 40080
Washington 40503
Wayne 40439
Wichita 40422
Winston-Salem 40208
Women's 230050
Worth 40336
York 40364
and 229667

```

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 'All Applications' page in the Hadoop interface. It features a sidebar with navigation options like 'Cluster', 'About', 'Nodes', and 'Tools'. The main content area displays a table with columns for ID, User, Name, Application Type, Application Tags, Queue, Application Priority, StartTime, LaunchTime, FinishTime, State, Priority, Running Containers, Allocated CPU V-Cores, Allocated Memory MB, Allocated GPUs, Reserved CPU V-Cores, Reserved Memory MB, Reserved GPUs, % of Queue, % of Cluster, Progress, Tracking UI, and Shutdown Nodes. A single job is listed with ID 'redaction_17064789497_000' and state 'FINISHED'.

Il est également possible de voir le comportement des noeuds workers, en allant à l'adresse: <http://localhost:8041> pour worker1, et <http://localhost:8042> pour worker2. Vous obtiendrez ce qui suit:

The screenshot shows the 'NodeManager information' page in the Hadoop interface. It features a sidebar with navigation options like 'ResourceManager', 'NodeManager', 'Node Information', 'List of Applications', 'List of Containers', and 'Tools'. The main content area displays a table with NodeManager information including:

- Total Vmem allocated for Containers: 16.80 GB
- Vmem enforcement enabled: false
- Total Pmem allocated for Container: 8 GB
- Pmem enforcement enabled: false
- Total VCores allocated for Containers: 8
- Resource types: memory-mb (unit=Mi), vcores
- NodeHealthyStatus: true
- LastNodeHealthTime: Sun Jan 07 17:34:53 GMT 2024
- NodeHealthReport
- NodeManager started on: Sun Jan 07 17:14:51 GMT 2024
- NodeManager Version: 3.3.6 from 1be78238728da9266a4f88195058f08fd012bf9c by ubuntu source checksum d42eb795a5eadb0feb15e44a7f87a9 on 2023-06-16T08:31Z
- Hadoop Version: 3.3.6 from 1be78238728da9266a4f88195058f08fd012bf9c by ubuntu source checksum 5652179ad55176cb287d9c633bb5bbd on 2023-06-16T08:22Z

Application

Écrire un Job Map Reduce permettant, à partir du fichier *purchases* initial, de déterminer le total des ventes par magasin. Il est à noter que la structure du fichier *purchases* est de la forme suivante:

```
date    temps    magasin    produit    cout    paiement
```

Veiller à toujours tester votre code en local avant de lancer un job sur le cluster!

Homework

Vous allez, pour ce cours, réaliser un projet en trinôme ou quadrinôme, qui consiste en la construction d'une architecture Big Data supportant le streaming, le batch processing, et le dashboarding temps réel. Pour la séance prochaine, vous allez commencer par mettre les premières pierres à l'édifice:

- Choisir la source de données sur laquelle vous allez travailler. Je vous invite à consulter les datasets offerts par [Kaggle](#) par exemple, ou chercher une source de streaming tel que Twitter.
- Réfléchir à l'architecture cible. La pipeline devrait intégrer des traitements en batch, des traitements en streaming et une visualisation.