1: Télécharger l'image docker

```
docker pull liliasfaxi/hadoop-cluster
docker images
```

```
REPOSITORY                                  TAG         IMAGE ID       CREATED         SIZE
...
liliasfaxi/hadoop-cluster                   latest      b90f134f6fcd   4 months ago    4.92GB
...
```

2: Créer le réseau docker

```
$ docker network create --driver=bridge hadoop
fd6beba0dffdeac7ba1abc39b776f72d59f0ea1b502b44c9ec3bc0f251cad4e7
```

3: Vérifier le réseau

```
$ docker network ls
NETWORK ID     NAME                 DRIVER    SCOPE
12833ae5767d   bridge               bridge    local
fd6beba0dffd   hadoop               bridge    local
d01f19a34fc9   host                 host      local
66b4c10c29b6   none                 null      local
...
```

4: Démarrer le cluster

```
docker run -itd --net=hadoop -p 9870:9870 -p 8088:8088 -p 7077:7077 -p 16010:16010 --name hadoop-master --hostname hadoop-master liliasfaxi/hadoop-cluster:latest
# de77c1cd1d0da90b01424ab026faa20db75cb7b7ff02cc8478581eafe423688e
docker run -itd -p 8040:8042 --net=hadoop --name hadoop-worker1 --hostname hadoop-worker1 liliasfaxi/hadoop-cluster:latest
# 7ea80582c69c45f5c66577e3c94351ae69d7fb164eb4c0f943f63e854fe2bf85
docker run -itd -p 8041:8042 --net=hadoop --name hadoop-worker2 --hostname hadoop-worker2 liliasfaxi/hadoop-cluster:latest
# 46fd8889eecd90d40f95085df646d2e06996a0c96a2484ea1b2f81201fff85e5
```

6: Afficher le contenu de `start-hadoop.sh`

```
# cat start-hadoop.sh
```

```
#!/bin/bash
echo -e "\n"
$HADOOP_HOME/sbin/start-dfs.sh
echo -e "\n"
$HADOOP_HOME/sbin/start-yarn.sh
echo -e "\n"
```

a: Expliquez ce que fait ce script en détail.

Il exécute `start-dfs.sh` puis `start-yarn.sh` et affiche des lignes vides entre les commandes.

b: Pourquoi est-il important de démarrer HDFS avant YARN dans un cluster Hadoop ?

Il faut démarrer le système de fichier avant de pouvoir démarrer le reste des composants.

d: voir les fichiers HDFS : hdfs dfs -ls /

```
# hdfs dfs -ls /
```

7: Créer un répertoire

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

8: Charger le fichier purchases dans le répertoire input *(*de HDFS*)*

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

9: Afficher le contenu du répertoire input

```
# hdfs dfs -ls /input
Found 1 items
-rw-r--r--   2 root supergroup  211312924 2024-12-09 12:45 /input/purchases.txt
```

10: Afficher les dernières lignes du fichier purchases:

```
# 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
```

11

Tester en local:

```
./bin/hadoop jar ../tp2/wordcountjava/target/wordcountjava-1.0-SNAPSHOT.jar org.apache.hadoop.examples.WordCount ../tp2/input/ ../tp2/output/
```

12: Copier le fichier

```
docker cp ../tp2/wordcountjava/target/wordcountjava-1.0-SNAPSHOT.jar hadoop-master:/root/file.jar
```

13: Lancer le job

```
hadoop jar file.jar org.apache.hadoop.examples.WordCount /input output
```

14: Afficher les dernières lignes du résultat

```
# hdfs dfs -ls output/
Found 2 items
-rw-r--r--   2 root supergroup          0 2024-12-09 13:57 output/_SUCCESS
-rw-r--r--   2 root supergroup     499048 2024-12-09 13:57 output/part-r-00000
# hdfs dfs -tail output/part-r-00000

Jose    39898
Kansas  39713
Laredo  40342
Las     80178
Lexington       40343
```

```
Lincoln 40345
Long    39942
Los     40254
Louis   39982
Louisville      40099
Lubbock 39837
Madison 40177
MasterCard      828524
Memphis 40222
Men's   230430
Mesa    40207
Miami   39935
Milwaukee       40316
Minneapolis     39991
Music   230150
Nashville       39854
New     80278
Newark  40577
Norfolk 40231
North   40013
Oakland 39728
Oklahoma        40446
Omaha   40209
Orlando 40195
Orleans 39914
Paso    39882
Paul    40160
Pet     229222
Petersburg      40093
Philadelphia    40748
Phoenix 40333
Pittsburgh      40358
Plano   40170
Portland        40065
Raleigh 40261
Reno    40254
Richmond        39983
Riverside       39963
Rochester       40455
Rouge   40387
Sacramento      40561
Saint   40160
San     200020
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
```

15: Calculer le total des ventes par magasin

Code:

```
// Fonction de mapping modifiée
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    /*
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
    */

    String line = value.toString();
    String[] parts = line.split("\t"); // Séparer la ligne en array des champs
    word.set(parts[2]); // Récupérer le nom du magasin
    context.write(word, one); // Émettre {magasin, 1}
}
```

Résultat *(head)*:

```
Albuquerque 40345
Anaheim 40086
Anchorage   39806
Arlington   40348
Atlanta 40168
Aurora  39808
Austin  40332
Bakersfield 40326
Baltimore   40196
Baton Rouge 40387
Birmingham  40253
Boise   40203
Boston  40338
Buffalo 40053
Chandler    39826
```

16:

1: Montant total des ventes par type de paiement

Code:

```
public static class LineMapper extends Mapper<Object, Text, Text, DoubleWritable> {

    private Text paymentType = new Text(); // clé
    private DoubleWritable salesValue = new DoubleWritable(); // valeur

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] parts = value.toString().split("\t");
        paymentType.set(parts[5]);
        salesValue.set(Float.parseFloat(parts[4]));
        context.write(paymentType, salesValue);
    }
}

public static class DoubleSumReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
    private DoubleWritable result = new DoubleWritable();

    public void reduce(
        Text key, Iterable<DoubleWritable> values, Context context
    ) throws IOException, InterruptedException {
        double sum = 0.0;
        for (DoubleWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Résultat *(head)*:

```
Amex    2.0662836677299362E8
Cash    2.0724507868629E8
Discover        2.0686962146348673E8
MasterCard      2.0701152435478592E8
Visa    2.0670336197324947E8
```

2: Nombre de transactions par jour

Code:

```
public static class LineMapper extends Mapper<Object, Text, Text, IntWritable> {

    private Text date = new Text(); // clé
    private IntWritable one = new IntWritable(); // valeur

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

```
        String[] parts = value.toString().split("\t");
        date.set(parts[0]);
        one.set(1);
        context.write(date, one);
    }
}
```

Résultat *(head)*:

```
2012-01-01      11341
2012-01-02      11312
2012-01-03      11382
2012-01-04      11276
2012-01-05      11399
2012-01-06      11406
2012-01-07      11382
2012-01-08      11497
2012-01-09      11311
2012-01-10      11329
2012-01-11      11401
2012-01-12      11384
2012-01-13      11414
2012-01-14      11348
2012-01-15      11327
2012-01-16      11388
```

3: Montant moyen par magain

Code:

```
public static class LineMapper extends Mapper<Object, Text, Text, DoubleWritable> {

    private Text location = new Text(); // clé
    private DoubleWritable salesValue = new DoubleWritable(); // valeur

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] parts = value.toString().split("\t");
        location.set(parts[2]);
        salesValue.set(Float.parseFloat(parts[4]));
        context.write(location, salesValue);
    }
}

public static class DoubleAverageReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
    private DoubleWritable result = new DoubleWritable();

    public void reduce(
        Text key, Iterable<DoubleWritable> values, Context context
    ) throws IOException, InterruptedException {
        double sum = 0;
        int nb = 0;
        for (DoubleWritable val : values) {
            sum += val.get();
            ++nb;
```

```
        }
        result.set(sum / nb);
        context.write(key, result);
    }
}
```

Résultat *(head)*:

```
Albuquerque     249.06840032768434
Anaheim 250.41678821863653
Anchorage       251.16815955261032
Arlington       248.71424680229293
Atlanta 247.41873144066471
Aurora  251.00047048131736
Austin  248.23819164385816
Bakersfield     247.89857234915644
Baltimore       251.53494911058297
Baton Rouge     248.9049493336216
```

~~4: Produits les plus vendus~~

5: Nombre de ventes par catégorie de produit

Code:

```java
public static class LineMapper extends Mapper<Object, Text, Text, IntWritable> {

    private Text product = new Text(); // clé
    private IntWritable one = new IntWritable(); // valeur

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] parts = value.toString().split("\t");
        product.set(parts[3]);
        one.set(1);
        context.write(product, one);
    }
}
```

Résultat *(head)*:

```
Baby    230293
Books   229787
CDs     230039
Cameras 229320
Children's Clothing     230469
Computers       229059
Consumer Electronics    229761
Crafts  229749
DVDs    230274
Garden  230073
Health and Beauty       229667
Men's Clothing  230430
```

```
Music   230150
Pet Supplies    229222
Sporting Goods  229932
Toys    229964
Video Games     230237
Women's Clothing        230050
```