

TP1 - Hadoop et Map Reduce

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: 2.7.2.
- [Docker](#) Version 17.09.1
- [IntelliJ IDEA](#) Version Ultimate 2016.1 (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 ce TP trois conteneurs représentant respectivement un noeud maître (Namenode) et deux noeuds esclaves (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 liliastaxi/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 \
liliastaxi/spark-hadoop:hv-2.7.2

docker run -itd -p 8040:8042 --net=hadoop \
--name hadoop-slave1 --hostname hadoop-slave1 \
liliastaxi/spark-hadoop:hv-2.7.2

docker run -itd -p 8041:8042 --net=hadoop \
--name hadoop-slave2 --hostname hadoop-slave2 \
liliastaxi/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,172.22.0.2' (ECDSA) to the list of known hosts.
hadoop-master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-hadoop-master.out
hadoop-slave1: Warning: Permanently added 'hadoop-slave1,172.22.0.3' (ECDSA) to the list of known hosts.
hadoop-slave2: Warning: Permanently added 'hadoop-slave2,172.22.0.4' (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-slave2: Warning: Permanently added 'hadoop-slave2,172.22.0.4' (ECDSA) to the list of known hosts.
hadoop-slave1: Warning: Permanently added 'hadoop-slave1,172.22.0.3' (ECDSA) to the list of known hosts.
hadoop-slave2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop-slave2.out
hadoop-slave1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop-slave1.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
```

Erreur

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`, 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](#) 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:

```
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
```

Le résultat suivant va donc s'afficher:

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

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

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

Instruction	Fonctionnalité
<code>hadoop fs -rm newfile.txt</code>	Supprimer le fichier
<code>hadoop fs -mkdir myinput</code>	Créer un répertoire
<code>hadoop fs -cat file.txt \ less</code>	Lire le fichier page par page

Interfaces web pour Hadoop

Hadoop offre plusieurs interfaces web pour pouvoir observer le comportement de ses différentes composantes. Vous pouvez afficher ces pages en local sur votre machine grâce à 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 le contenu du fichier `start-container.sh` fourni dans le projet, vous verrez que deux ports de la machine maître ont été exposés:

- Le port **50070**: 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 prêt à l'emploi, vous pouvez, sur votre navigateur préféré de votre machine hôte, aller à : `http://localhost:50070`. Vous obtiendrez le résultat suivant:

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Overview 'hadoop-master:9000' (active)

Started:	Fri Jan 26 11:47:09 UTC 2018
Version:	2.7.2, rUnknown
Compiled:	2016-05-27T18:05Z by root from Unknown
Cluster ID:	CID-3c662456-d44e-4301-bc39-28e479c4dc88
Block Pool ID:	BP-431089505-172.17.0.2-1465730089024

Summary

Security is off.
Safemode is off.
20 files and directories, 8 blocks = 28 total filesystem object(s).
Heap Memory used 85.33 MB of 165.5 MB Heap Memory. Max Heap Memory is 889 MB.
Non Heap Memory used 37.42 MB of 38.44 MB Committed Non Heap Memory. Max Non Heap Memory is 214 MB.

Configured Capacity:	125.49 GB
DFS Used:	406.85 MB (0.32%)
Non DFS Used:	34.1 GB

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

*** 🔖

🔍 Rechercher

⬇️ 📄 🔄 🌐 ☰



All Applications

Logged in as: dr.who

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	Rebooted 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:32>	

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 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 IntelliJ IDEA. Nous utiliserons dans notre cas JDK 1.8.

- Définir les valeurs suivantes pour votre projet:
 - **GroupId**: `hadoop.mapreduce`
 - **ArtifactId**: `wordcount`
 - **Version**: `1`
- 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 *tn.insat.tp1* sous le répertoire *src/main/java*
- Créer la classe *TokenizerMapper*, contenant ce code:

```
package tn.insat.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 tn.insat.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 tn.insat.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 IntelliJ:

- 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!

```

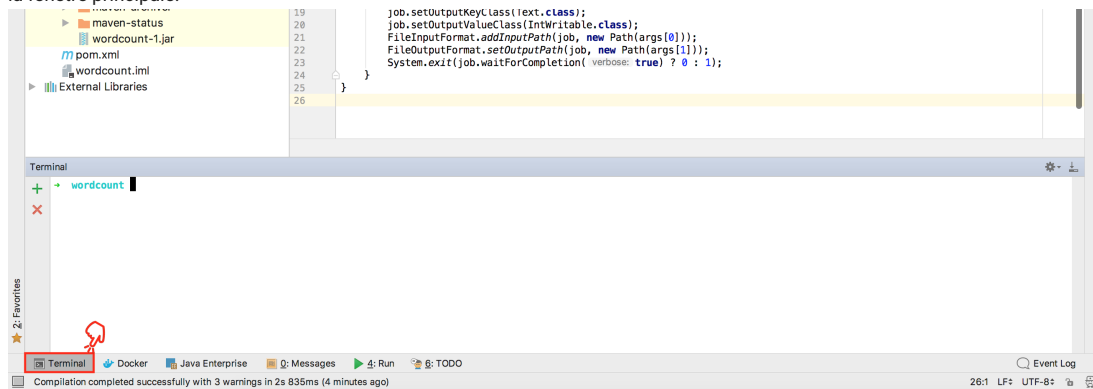
- Créer une configuration de type *Application* (*Run->Edit Configurations...->+>Application*).
- Définir comme **Main Class**: `tn.insat.tp1.WordCount`, et comme **Program Arguments**: `src/main/resources/input/file.txt`
`src/main/resources/output`
- Lancer le programme. Un répertoire *output* sera 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 IntelliJ:

- Créer une configuration Maven avec la ligne de commande: `package install`
- Lancer la configuration. Un fichier *wordcount-1.jar* sera créé dans le répertoire *target* du projet.
- 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 tn.insat.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 tn.insat.tp1.WordCount input output
18/01/27 10:58:13 INFO client.RMProxy: Connecting to ResourceManager at hadoop-master/172.22.0.2:8032
18/01/27 10:58:14 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute y
our application with ToolRunner to remedy this.
18/01/27 10:58:14 INFO input.FileInputFormat: Total input paths to process : 1
18/01/27 10:58:14 INFO mapreduce.JobSubmitter: number of splits:1
18/01/27 10:58:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1517050438813_0001
18/01/27 10:58:15 INFO impl.YarnClientImpl: Submitted application application_1517050438813_0001
18/01/27 10:58:16 INFO mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_1517050438813_0001/
18/01/27 10:58:16 INFO mapreduce.Job: Running job: job_1517050438813_0001
18/01/27 10:58:29 INFO mapreduce.Job: Job job_1517050438813_0001 running in uber mode : false
18/01/27 10:58:29 INFO mapreduce.Job: map 0% reduce 0%
18/01/27 10:58:47 INFO mapreduce.Job: map 42% reduce 0%
18/01/27 10:59:11 INFO mapreduce.Job: map 67% reduce 0%
18/01/27 10:59:28 INFO mapreduce.Job: map 100% reduce 0%
18/01/27 10:59:37 INFO mapreduce.Job: map 100% reduce 100%
18/01/27 10:59:37 INFO mapreduce.Job: Job job_1517050438813_0001 completed successfully
18/01/27 10:59:38 INFO mapreduce.Job: Counters: 49
File System Counters
  FILE: Number of bytes read=2568324
  FILE: Number of bytes written=4086870
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=44337931
  HDFS: Number of bytes written=491081
  HDFS: Number of read operations=6
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
  Launched map tasks=1
  Launched reduce tasks=1
  Data-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=57326
  Total time spent by all reduces in occupied slots (ms)=6032
  Total time spent by all map tasks (ms)=57326
  Total time spent by all reduce tasks (ms)=6032
  Total vcore-milliseconds taken by all map tasks=57326
  Total vcore-milliseconds taken by all reduce tasks=6032
  Total megabyte-milliseconds taken by all map tasks=58701824
  Total megabyte-milliseconds taken by all reduce tasks=6176768
Map-Reduce Framework
  Map input records=868279
  Map output records=5872188
  Map output bytes=67826564
  Map output materialized bytes=1284159
  Input split bytes=120
  Combine input records=5072188
  Combine output records=101438
  Reduce input groups=50766
  Reduce shuffle bytes=1284159
  Reduce input records=101438
  Reduce output records=50766
  Spilled Records=304314
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=225
  CPU time spent (ms)=51590
  Physical memory (bytes) snapshot=549974016
  Virtual memory (bytes) snapshot=1775112192
  Total committed heap usage (bytes)=292028416
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=44337811
File Output Format Counters
  Bytes Written=491081

```

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

Petersburg	8430
Philadelphia	8471
Phoenix	8431
Pittsburgh	8470
Plano	8323
Portland	8367
Raleigh	8345
Reno	8334
Richmond	8388
Riverside	8338
Rochester	8440
Rouge	8396
Sacramento	8597
Saint	8494
San	42110
Santa	8416
Scottsdale	8443
Seattle	8339
Spokane	8356
Sporting	48207
Springs	8534
St.	16881
Stockton	8289
Supplies	48265
Tampa	8400
Toledo	8314
Toys	48463
Tucson	8546
Tulsa	8444
Vegas	16957
Video	48439
Virginia	8465
Visa	174018
Vista	8510
Washington	8477
Wayne	8527
Wichita	8547
Winston-Salem	8459
Women's	48252
Worth	8462
York	8529
and	48408

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:



All Applications

Logged in as: dr.who

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	Rebooted Nodes
1	0	0	1	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
application_1517050438813_0001	root	word count	MAPREDUCE	default	Sat Jan 27 11:58:15 +0100 2018	Sat Jan 27 11:59:36 +0100 2018	FINISHED	SUCCEEDED		History	N/A

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

Il est également possible de voir le comportement des noeuds esclaves, en allant à l'adresse: <http://localhost:8041> pour *slave1*, et <http://localhost:8042> pour *slave2*. Vous obtiendrez ce qui suit:



NodeManager information

Logged in as: dr.who

ResourceManager
NodeManager
Node Information
List of Applications
List of Containers
Tools

Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	true
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total VCores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Sat Jan 27 11:14:00 UTC 2018
NodeHealthReport	
Node Manager Version:	2.7.2 from Unknown by root source checksum c63f7cc71b8f63249e35126f07492d on 2016-05-27T18:16Z
Hadoop Version:	2.7.2 from Unknown by root source checksum d0fda26633fa762bfb67ec759be689c on 2016-05-27T18:05Z

Application

Écrire un Job Map Reduce permettant, à partir du fichier purchases initial, de déterminer le total des ventes par magasin. 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.