Instalación Hadoop en Ubuntu 16.04

Esta instalación se ha realizado en una máquina Linux Mint 18.3 Sylvia (distribución basada en Ubuntu 16.04 LTS)

Instalar java

```
$ sudo apt update
$ sudo apt install default-jdk
$ java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)
```

Instalar ssh y configurar acceso sin password para el usuario

Vamos a suponer que la cuenta en que se va a usar hadoop es la del usuario “hduser” y que puede ejecutar comandos “su”.

Hadoop require ssh para acceder a los nodos, tanto máquinas remotas como a la máquina local.

```
$ sudo apt install openssh-server
$ cd
$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
...
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Linux Mint 18.3 Sylvia (GNU/Linux 4.13.0-38-generic x86_64)

 * Documentation:  https://www.linuxmint.com
Last login: Mon Apr  9 18:43:58 2018 from 127.0.0.1
$ exit
logout
Connection to localhost closed
```

Instalación de hadoop

Bajamos hadoop de http://apache.uvigo.es y lo instalamos

```
$ wget http://apache.uvigo.es/hadoop/common/hadoop-3.1.0/hadoop-3.1.0.tar.gz
$ tar xvfz hadoop-3.1.0.tar.gz
$ sudo mkdir /usr/local/hadoop
$ sudo mv hadoop-3.1.0/* /usr/local/hadoop/
$ sudo chown -R hduser:hduser /usr/local/hadoop
```

Configuración de hadoop.apache.org

1. Es necesario modificar el fichero .bashrc, pero previamente vemos el camino de la instalación java en el sistema

```
$ update-alternatives --config java
```

```
Sólo hay una alternativa en el grupo de enlaces java (provee /usr/bin/java):
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
Nada que configurar.
```

En este caso, .bashrc debe quedar:

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

```
$ source .bashrc
```

2. Modificar la variable JAVA_HOME en el fichero /usr/local/hadoop/etc/hadoop/hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3. Crear un directorio temporal y modificar el fichero /usr/local/hadoop/etc/hadoop/core-site.xml
Primero creamos el directorio:

```
$ sudo mkdir -p /app/hadoop/tmp
$ sudo chown hduser:hduser /app/hadoop/tmp
```

Modificamos el fichero /usr/local/hadoop/etc/hadoop/core-site.xml en el que la etiqueta <configuration> debe quedar:

```
<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
 </property>

 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
 </property>
</configuration>
```

4. Modificar el fichero /usr/local/hadoop/etc/hadoop/mapred-site.xml

A continuación editamos el fichero /usr/local/hadoop/etc/hadoop/mapred-site.xml, de tal modo que la etiqueta <configuration> debe quedar:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
```

```
  and reduce task.
  </description>
 </property>
</configuration>
```

5. Crear y configurar directorios para almacenar información de namenode y datanode

Creamos y configuramos dos directorios:

```
$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
$ sudo chown -R hduser:hduser /usr/local/hadoop_store
```

En el fichero /usr/local/hadoop/etc/hadoop/hdfs-site.xml la etiqueta <configuration> debe quedar:

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>
```

Formateado del sistema de ficheros hdfs

```
$ hdfs namenode -format
```

Comprobación de funcionamiento de hadoop

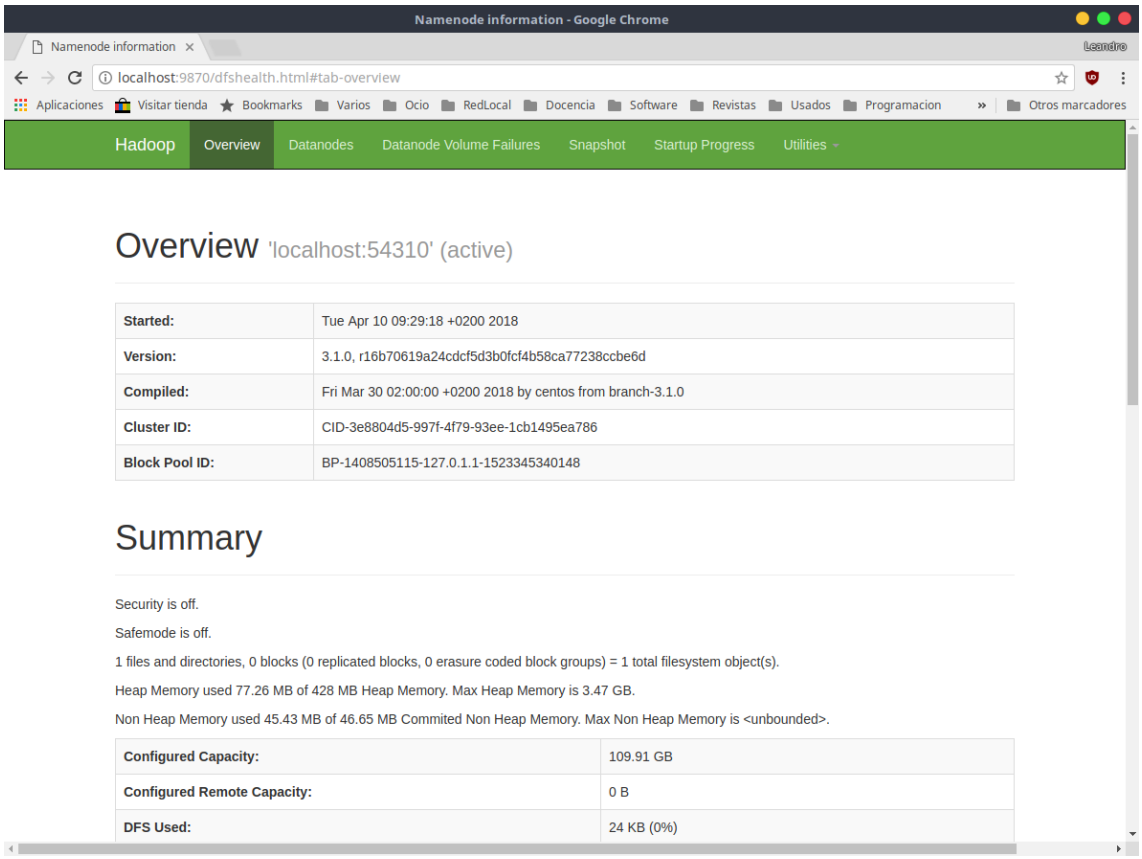
El sistema hadoop se pone en marcha y se para ejecutando los siguientes comandos:

```
$ start-dfs.sh; start-yarn.sh
$ stop-dfs.sh; stop-yarn.sh
```

Se puede comprobar su funcionamiento haciendo uso de jps (herramienta de java para comprobar los procesos de la máquina virtual java):

```
$ start-dfs.sh
$ start-yarn.sh
...
$ jps
10865 ResourceManager
10354 NameNode
11155 NodeManager
10500 DataNode
11270 Jps
```

Además, a través del puerto 9870 podemos acceder a una interfaz gráfica que nos permite obtener información de sistema hadoop:



Ejemplo de ejecución de programa en python en hadoop

Partimos de dos ficheros sencillos de ejemplo. join1_FileA.txt es:

```
able,991
about,11
burger,15
actor,22
```

Y join1_FileB.txt es:

```
Jan-01 able,5
Feb-02 about,3
Mar-03 about,8
Apr-04 able,13
Feb-22 actor,3
Feb-23 burger,5
Mar-08 burger,2
Dec-15 able,100
```

El objetivo es hacer una operación join sobre los ficheros. Para ello, el mapper (fichero join1_mapper.py) será:

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

# --------------------------------------------------------------------------
# Este mapper acepta valores <date word, value> y mueve la fecha con el valor
# No hay ninguna comprobación de errores en la entrada
# --------------------------------------------------------------------------

for line in sys.stdin:
    line       = line.strip()
```

```
    key_value  = line.split(",")
    key_in     = key_value[0].split(" ")
    value_in   = key_value[1]

    if len(key_in)>=2:   # si key_in tiene dos elementos
        date = key_in[0]
        word = key_in[1]
        value_out = date+" "+value_in
        print( '%s\t%s' % (word, value_out) )
    else:
        print( '%s\t%s' % (key_in[0], value_in) )
```

Y el reducer (fichero join1_reducer.py) será:

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

import sys

# ------------------------------------------------------------------------
# Acepta un fichero <palabra, valor> y hace un join de palabras
# Lleva la cuenta de la palabra actual y la anterior, y si la palabra cambia
# hace el join mostrando la palabra y las claves. No hay que hacer comprobación
# de las claves porque hadoop las pone ordenadas
#  No se hace comprobación de errores sobre la entrada
# ------------------------------------------------------------------------

prev_word          = "  "
months             =
['Jan','Feb','Mar','Apr','Jun','Jul','Aug','Sep','Nov','Dec']
dates_to_output    = []
day_cnts_to_output = []
line_cnt           = 0  # contador de líneas

for line in sys.stdin:
    line       = line.strip()
    key_value  = line.split('\t')
    line_cnt   = line_cnt+1

    curr_word  = key_value[0]
    value_in   = key_value[1]

    #-----------------------------------------------------
    # Comprueba que la palabra es nueva
    #-----------------------------------------------------
    if curr_word != prev_word:

        # ----------------------
          # Escribe el resultado de join para nº línea > 1
        # ----------------------
        if line_cnt>1:
          for i in range(len(dates_to_output)):
               print('{0} {1} {2}
{3}'.format(dates_to_output[i],prev_word,day_cnts_to_output[i],curr_word_total_c
nt))
            dates_to_output=[]
            day_cnts_to_output=[]
            prev_word=curr_word

    # ------------------------------------------------------------
    # Procesa la palabra actual
```

```
    # ---------------------------------------------------------

    if (value_in[0:3] in months):
        date_day =value_in.split()
        dates_to_output.append(date_day[0])
        day_cnts_to_output.append(date_day[1])
    else:
        curr_word_total_cnt = value_in

# ---------------------------------------------------------
# Escribe el último resultado del join
# ---------------------------------------------------------
for i in range(len(dates_to_output)):
         print('{0} {1} {2}
{3}'.format(dates_to_output[i],prev_word,day_cnts_to_output[i],curr_word_total_c
nt))
```

Para ejecutar el ejemplo:

```
$ start-dfs.sh
$ hdfs dfs -mkdir /join
$ hdfs dfs -mkdir /join/input
$ hdfs dfs -put join1_File*.txt /join/input
$ hdfs dfs -ls /join/input
Found 2 items
-rw-r--r--   1 hduser supergroup         37 2016-04-26 16:09
/join/input/join1_FileA.txt
-rw-r--r--   1 hduser supergroup        122 2016-04-26 16:09
/join/input/join1_FileB.txt
$ chmod +x join1_*py
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.1.0.jar
-input /join/input -output /join/output -mapper ./join1_mapper.py -reducer
./join1_reducer.py
$ hdfs dfs -ls /join/output
Found 2 items
-rw-r--r--   1 hduser supergroup          0 2016-04-26 16:14
/join/output/_SUCCESS
-rw-r--r--   1 hduser supergroup        157 2016-04-26 16:14 /join/output/part-
00000
$ hdfs dfs -get /join/output/part-00000 ./output.txt
```

Y el fichero output.txt que obtenemos contendrá:

```
Dec-15 able 100 991
Apr-04 able 13 991
Jan-01 able 5 991
Mar-03 about 8 11
Feb-02 about 3 11
Feb-22 actor 3 22
Mar-08 burger 2 15
Feb-23 burger 5 15
```

Instalación Apache Spark

Apache Spark se distribuye en varias versiones. En este apartado se comenta la instalación de la versión de Apache Spark que incluye los binarios hadoop, y que, por tanto, puede ser ejecutada directamente. Además de esta versión, se distribuye también en versión código fuente y precompilada sin incluir ejecutables de hadoop. Estas versiones son útiles en el caso de que deseemos usar Apache Spark en un sistema hadoop en funcionamiento.

Una vez descargado Apache Spark, procedemos a descomprimirlo e instalarlo en el sistema:

```
$ wget http://apache.uvigo.es/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz
$ tar xvfz spark-2.3.0-bin-hadoop2.7.tgz
$ sudo mkdir /usr/local/spark
$ sudo mv spark-2.3.0-bin-hadoop2.7/* /usr/local/spark/
$ sudo chown -R hduser:hduser /usr/local/spark/
```

Añadimos al fichero .bashrc el camino al directorio de instalación:

```
PATH=$PATH:/usr/local/spark/bin
```

Y cargamos de nuevo este fichero:

```
$ source .bashrc
```

Podemos comprobar su funcionamiento ejecutando alguno de los ejemplos proporcionados con spark:

```
$ spark-submit /usr/local/spark/examples/src/main/python/pi.py 10
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/04/26 18:35:01 INFO SparkContext: Running Spark version 1.6.1
16/04/26 18:35:01 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
16/04/26 18:35:02 WARN Utils: Your hostname, XPS13 resolves to a loopback
address: 127.0.1.1; using 192.168.33.157 instead (on interface wlan0)
……
16/04/26 18:35:05 INFO DAGScheduler: Job 0 finished: reduce at
/usr/local/spark/examples/src/main/python/pi.py:39, took 0,919031 s
Pi is roughly 3.150112
……
```

Ejemplo de ejecución de programa en python en spark

Partimos de dos ficheros sencillos de ejemplo. join1_FileA.txt es:

```
able,991
about,11
burger,15
actor,22
```

Y join1_FileB.txt es:

```
Jan-01 able,5
Feb-02 about,3
Mar-03 about,8
Apr-04 able,13
Feb-22 actor,3
Feb-23 burger,5
Mar-08 burger,2
Dec-15 able,100
```

El objetivo es hacer una operación join sobre los ficheros. Para ello, el fichero code.py contendrá dos funciones:

```
def split_fileA(line):
      (word,count) = line.split(',')
      count = int(count)
      return (word,count)

def split_fileB(line):
      (date,rest) = line.split()
      (word,count) = rest.split(',')
      return (word, date + " " + count)
```

Ejecutamos en el terminal en el directorio que contiene los ficheros join1_FileA.txt, join1_FileB.txt y code.py:

```
$ pyspark
……
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Python version 2.7.12 (default, Dec  4 2017 14:50:18)
SparkSession available as 'spark'.
>>> fileA = sc.textFile("join1_FileA.txt")
>>> fileA.collect()
[u'able,991', u'about,11', u'burger,15', u'actor,22']
>>> fileB = sc.textFile("join1_FileB.txt")
>>> fileB.collect()
[u'Jan-01 able,5', u'Feb-02 about,3', u'Mar-03 about,8', u'Apr-04 able,13',
u'Feb-22 actor,3', u'Feb-23 burger,5', u'Mar-08 burger,2', u'Dec-15 able,100']
>>> from code import *
>>> fileA_data = fileA.map(split_fileA)
>>> fileA_data.collect()
[(u'able', 991), (u'about', 11), (u'burger', 15), (u'actor', 22)]
>>> fileB_data = fileB.map(split_fileB)
>>> fileB_data.collect()
[(u'able', u'Jan-01 5'), (u'about', u'Feb-02 3'), (u'about', u'Mar-03 8'),
```

```
(u'able', u'Apr-04 13'), (u'actor', u'Feb-22 3'), (u'burger', u'Feb-23 5'),
(u'burger', u'Mar-08 2'), (u'able', u'Dec-15 100')]
>>> fileB_joined_fileA = fileB_data.join(fileA_data)
>>> out=fileB_joined_fileA.collect()
>>> out
[(u'able', (u'Jan-01 5', 991)), (u'able', (u'Apr-04 13', 991)), (u'able',
(u'Dec-15 100', 991)), (u'burger', (u'Feb-23 5', 15)), (u'burger', (u'Mar-08 2',
15)), (u'about', (u'Feb-02 3', 11)), (u'about', (u'Mar-03 8', 11)), (u'actor',
(u'Feb-22 3', 22))]
```

Y obtenemos los resultados en una lista python.