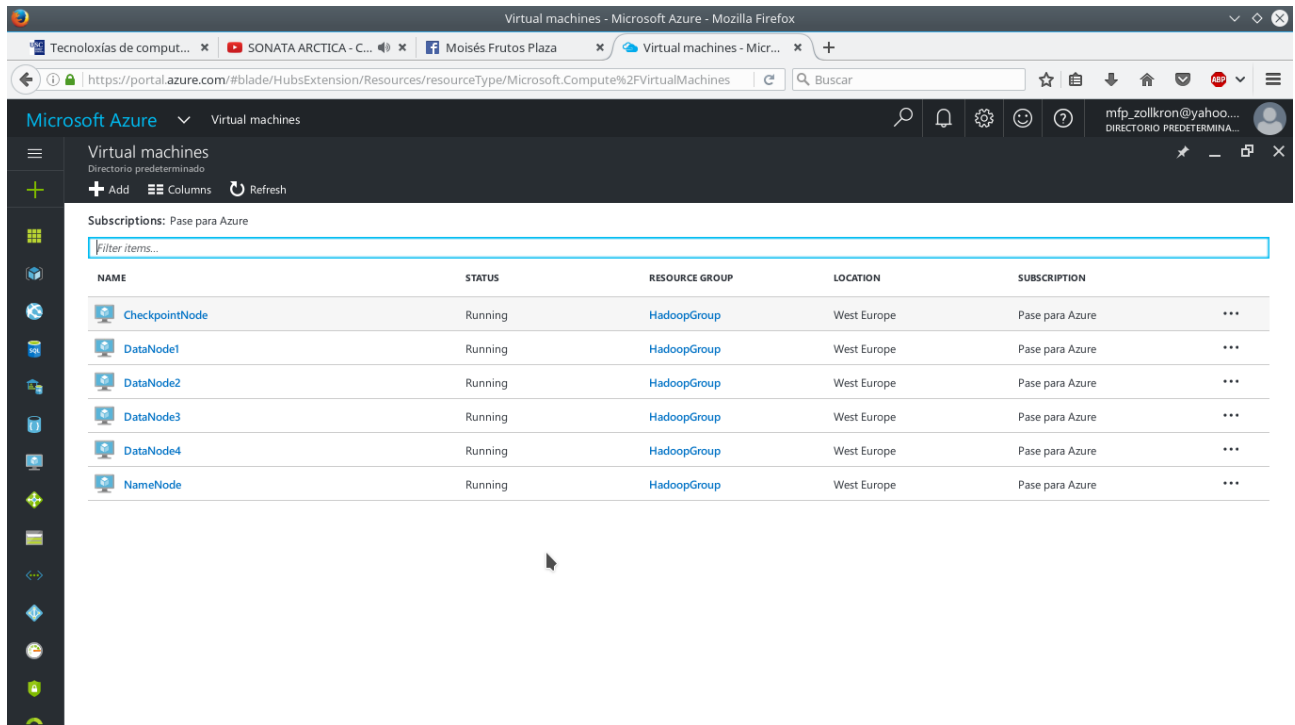


TECNOLOGÍAS DE COMPUTACIÓN DE DATOS MASIVOS



Virtual machines - Microsoft Azure - Mozilla Firefox

https://portal.azure.com/#blade/HubsExtension/Resources/resourceType/Microsoft.Compute%2FVirtualMachines

Microsoft Azure Virtual machines

Subscriptions: Pase para Azure

NAME	STATUS	RESOURCE GROUP	LOCATION	SUBSCRIPTION
CheckpointNode	Running	HadoopGroup	West Europe	Pase para Azure
DataNode1	Running	HadoopGroup	West Europe	Pase para Azure
DataNode2	Running	HadoopGroup	West Europe	Pase para Azure
DataNode3	Running	HadoopGroup	West Europe	Pase para Azure
DataNode4	Running	HadoopGroup	West Europe	Pase para Azure
NameNode	Running	HadoopGroup	West Europe	Pase para Azure

PRÁCTICA 1 – MONTAJE DE CLÚSTER HADOOP EN AZURE

MOISÉS FRUTOS PLAZA 48488132-S
moises.frutos@um.es
MÁSTER EN BIG DATA

ENTREGADO EL 13 DE NOVIEMBRE DE 2016

ÍNDICE

Índice de contenido

1. INTRODUCCIÓN.....	3
2. DARNOS DE ALTA EN LA PLATAFORMA MICROSOFT AZURE.....	4
3. PREPARAR UNA IMAGEN BASE DE MAQUINA HADOOP EN EL CLÚSTER.....	4
3.1. CREACIÓN DE UN USUARIO HDMaster PARA EL MANEJO DE HADOOP.....	5
3.2. CONFIGURACIÓN DE LOS DEMONIOS HADOOP.....	7
3.3. CONFIGURACIÓN DEL ENTORNO.....	11
3.4. OBTENCIÓN DE LA IMAGEN BASE.....	11
3.5. CREACIÓN DEL GRUPO HADOOP DENTRO DEL CLÚSTER.....	12
3.6. CREACIÓN DE LA RED VIRTUAL DENTRO DEL CLÚSTER.....	12
3.7. CREACIÓN DEL NAMENODE / RESOURCEMANAGER.....	12
3.8. CREACIÓN DEL CHECKPOINTNODE / JOBHISTORY.....	14
3.9. CREACIÓN DE LOS DATANODES.....	14
3.10. ADICIÓN DE LAS IPS PÚBLICAS AL DNS LOCAL.....	15
3.11. ELIMINACIÓN DE LA IMAGEN BASE DE MÁQUINA HADOOP.....	16
4. CÓMO ENCENDER / APAGAR MÁQUINAS VIRTUALES EN AZURE.....	17
5. PREPARACIÓN DEL SISTEMA HDFS.....	17
5.1. ESPECIFICACIÓN DE DATANODES / NODEMANAGERS.....	17
5.2. INICIO DE LOS DEMONIOS EN EL CLÚSTER.....	18
5.3. PARADA DE LOS DEMONIOS EN EL CLÚSTER.....	20
5.4. PRUEBA DE EJEMPLO.....	21
5.5. CREACIÓN DE LOS USUARIOS EN HDFS.....	21
5.6. PRUEBA COMO USUARIO NO PRIVILEGIADO (NO HDMaster).....	21
6. INSTALAR HADOOP EN LOCAL.....	22
7. EJECUCIÓN DE WORDCOUNT.....	23
8. EJECUCIÓN DE BENCHMARKS.....	26
9. AÑADIR Y QUITAR DATANODES / JOBTRACKERS.....	28
10. RACK AWARENESS.....	31

1. INTRODUCCIÓN

Esta práctica tiene como propósito desplegar un clúster Hadoop en la plataforma de Microsoft Azure, para ello se siguen una serie de pasos:

- Darnos de alta en la plataforma Microsoft Azure usando una cuenta Microsoft
- Preparar una imagen base de máquina Hadoop en el clúster
 - Crear un usuario hdmaster para el manejo Hadoop dentro del clúster en la imagen base
 - Configuración de los demonios Hadoop
 - Configuración del entorno
 - Obtenemos la imagen base
 - Creamos grupo de Hadoop para el Clúster
 - Creamos Red Virtual
 - Creamos el NameNode/Resource Manager
 - Creamos el CheckPointNode/JobHistory
 - Creamos los DataNodes
 - Añadimos las IPs públicas al DNS Local
 - Eliminar la máquina virtual que nos ha servido de imagen base Hadoop
- Cómo Encender/Apagar Máquinas Virtuales en Azure
- Preparación del HDFS
 - Especificar los Datanodes/Nodemangers
 - Inicio de los Demonios Hadoop en el clúster
 - Parada de los Demonios Hadoop en el clúster
 - Prueba de ejemplo
 - Creación de los usuarios en HDFS
 - Prueba como usuario No Privilegiado (No hdmaster)
- Instalar Hadoop en local
- Ejecución de WordCount
- Ejecuciones de Benchmarks
- Añadir y Retirar Datanodes/Jobtrackers
- Rack Awareness

A continuación se irá viendo, apartado por apartado, el despliegue del clúster en Azure y todas las pruebas realizadas.

2. DARNOS DE ALTA EN LA PLATAFORMA MICROSOFT AZURE

Para darnos de alta en plataforma de Azure tenemos que seguir los siguientes pasos:

1. Si no tenemos cuenta Microsoft podemos crearnos una aquí: <https://www.microsoft.com/es-ES/account/default.aspx>
2. Después nos tenemos que dirigir aquí y darnos de alta en la plataforma Azure con nuestra cuenta Microsoft: <http://portal.azure.com/>
3. Tras ello seguir los pasos que indique la plataforma.
4. Una vez dentro de la plataforma Azure debemos de bajarnos el cliente Azure CLI para poder ejecutar acciones de Azure desde la línea de comandos. Lo podemos bajar aquí: <https://azure.microsoft.com/es-es/documentation/articles/xplat-cli-install/>
5. Una vez tengamos todo listo podremos logearnos dentro de la plataforma Azure con `azure login`
6. Para ver las regiones en las que podemos lanzar las máquinas, tenemos que ejecutar `azure location list`
7. Nos registramos para poder usar computación, almacenamiento y red (si diera error, ejecutamos el mismo comando otra vez):

```
1. azure provider register Microsoft.Compute
2. azure provider register Microsoft.Storage
3. azure provider register Microsoft.Network
```

Tras esto ya tenemos la plataforma de Azure lista para preparar nuestra primera máquina virtual que nos va a servir de base para montar el clúster Hadoop.

3. PREPARAR UNA IMAGEN BASE DE MAQUINA HADOOP EN EL CLÚSTER

Para preparar la imagen base tenemos que hacer lo siguiente:

1. Creamos un Resource Group para la instalación base, en la localización westeurope:

```
azure group create BaseInst -l westeurope
```

2. Creamos una máquina virtual en ese grupo, basada en Debian, en la que instalaremos Hadoop:

```
azure vm quick-create -g BaseInst -n HadoopBase -y Linux
-Q Debian -u poned_un_nombre -p poned_una_contraseña -z
Standard_D1_v2 -l westeurope
```

- Después de -u ponemos el nombre que queramos para el administrador de la máquina; después de -p una contraseña que tiene que tener al menos 8 caracteres, conteniendo al menos una minúscula, una mayúscula, un número y un signo de entre !@#\$%^&+=

- La creación puede llevarle varios minutos, cuando acabe muestra información sobre la máquina creada, entre otras cosas su IP pública
 - Nos conectamos por ssh con la máquina creada, usando el usuario y contraseña
3. Una vez dentro de la MV de Azure, instalamos Java (OpenJDK v8) y otras librerías que usa Hadoop:

```
$ sudo su
# apt-get update
# apt-get install openjdk-8-jre libssl-dev
```

4. Descargamos la última versión estable de Hadoop en /opt/yarn (en el momento de escribir este documento es la 2.7.3), crea un enlace simbólico y define la variable HADOOP_PREFIX:

```
# mkdir /opt/yarn
# cd /opt/yarn
# wget
http://mirrors.gigenet.com/apache/hadoop/common/hadoop-
2.7.3/hadoop-2.7.3.tar.gz
# tar xvzf hadoop-2.7.3.tar.gz
# rm hadoop-2.7.3.tar.gz
# ln -s hadoop-2.7.3 hadoop
# export HADOOP_PREFIX=/opt/yarn/hadoop
```

3.1. CREACIÓN DE UN USUARIO HDMaster PARA EL MANEJO DE HADOOP

5. Creamos un grupo hadoop y un usuario hdmaster para ejecutar los diferentes demonios (HDFS y YARN). Cambiamos también el propietario del directorio de hadoop:

```
# groupadd -r hadoop
# useradd -r -g hadoop -d /opt/yarn -s /bin/bash
hdmaster
# chown -R hdmaster:hadoop /opt/yarn
```

6. Creamos los directorios para los datos de HDFS (NameNode, DataNodes y Checkpoint node) y hacemos que sean propiedad del usuario hdmaster. En un sistema real, estos directorios deberían estar en particiones separadas con suficiente espacio libre:

```
# mkdir -p /var/data/hadoop/hdfs/nn
# mkdir -p /var/data/hadoop/hdfs/cpn
# mkdir -p /var/data/hadoop/hdfs/dn
# chown -R hdmaster:hadoop /var/data/hadoop/hdfs
```

7. Creamos directorios para los ficheros de log y haz que sean propiedad del usuario hdmaster:

```
# mkdir -p /var/log/hadoop/yarn
# mkdir -p /var/log/hadoop/hdfs
# mkdir -p /var/log/hadoop/mapred
# chown -R hdmaster:hadoop /var/log/hadoop
```

8. Modificamos el fichero /etc/hosts para indicar el nombre/ip de las máquinas que harán el papel de Namenode/ResourceManager (10.0.0.10) y de CheckPoint Node/JobHistory server (10.0.0.5), añadiendo las siguientes líneas (Nota: añadimos un datanode más que nos hará falta más adelante):

```
10.0.0.4      namenode resourcemanager
10.0.0.5      checkpointnode jobhistoryserver
10.0.0.6      datanode1
10.0.0.7      datanode2
10.0.0.8      datanode3
10.0.0.9      datanode4
10.0.0.10     datanode5
```

9. Modificamos el fichero /etc/ssh/ssh_config y ponemos el parámetro *StrictHostKeyChecking* a *no*.

10. Como usuario hdmaster (su - hdmaster) copiamos los ficheros de /etc/skel (cp /etc/skel/* ~) y añadimos a ~/.bashrc las líneas:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_PREFIX=/opt/yarn/hadoop
export PATH=$PATH:$HADOOP_PREFIX/bin
```

11. Reiniciamos sesión para que se cargue el ~/.bashrc con las nuevas variables de entorno.

12. Comprobamos que Hadoop funciona con:

```
$ hadoop version
```

13. El usuario hdmaster debe poder conectarse entre los nodos del clúster por ssh sin password. Ejecutamos lo siguiente como usuario hdmaster:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 644 ~/.ssh/authorized_keys
```

14. Probamos que podemos hacer *ssh localhost* y ya no nos pide la contraseña.

3.2. CONFIGURACIÓN DE LOS DEMONIOS HADOOP

15. Como usuario hdmaster, cambiamos los siguientes ficheros en la ruta `$HADOOP_PREFIX/etc/hadoop/`:

core-site.xml: configuración general de Hadoop

```
<configuration>
  <property>
    <!-- nombre del Namenode -->
    <name>fs.defaultFS</name>
    <value>hdfs://namenode:9000/</value>
    <final>true</final>
  </property>
  <property>
    <!-- Almacenamiento temporal (debe tener suficiente espacio) -->
    <name>hadoop.tmp.dir</name>
    <value>/var/tmp</value>
    <final>true</final>
  </property>
  <property>
    <!-- Usuario por defecto para el interfaz web -->
    <name>hadoop.http.staticuser.user</name>
    <value>hdfs</value>
    <final>true</final>
  </property>
</configuration>
```

hdfs-site.xml: configuración de HDFS

```
<configuration>
  <property>
    <!-- Factor de replicacion de los bloques -->
    <name>dfs.replication</name>
    <value>3</value>
    <final>true</final>
  </property>
  <property>
    <!-- Tamano del bloque (por defecto 128m) -->
    <name>dfs.blocksize</name>
    <value>64m</value>
    <final>true</final>
  </property>
  <property>
    <!-- Lista (separada por comas) de directorios donde el namenode guarda los metadatos.
    En un sistema real debería incluir por lo menos dos directorios:
    uno en el disco local del namenode y otro remoto montado por NFS -->
    <name>dfs.namenode.name.dir</name>
```

```

    <value>file:///var/data/hadoop/hdfs/nn</value>
    <final>true</final>
  </property>
</property>
<property>
  <!-- Lista (separada por comas) de directorios donde el checkpoint node guarda los
checkpoints.
      Igual que el el dfs.namenode.name.dir, deberían indicarse un directorio local y
uno remoto -->
    <name>fs.checkpoint.dir</name>
    <value>file:///var/data/hadoop/hdfs/cpn</value>
    <final>true</final>
  </property>
</property>
  <!-- Lista (separada por comas) de directorios donde el checkpoint node guarda los
edits temporales -->
    <name>fs.checkpoint.edits.dir</name>
    <value>file:///var/data/hadoop/hdfs/cpn</value>
    <final>true</final>
  </property>
  <!-- Lista (separada por comas) de directorios donde los datanodes guardan los
datos:
      por rendimiento, si los nodos tiene varios discos es conveniente
      especificar un directorio en cada uno de los discos locales -->
    <name>dfs.datanode.data.dir</name>
    <value>file:///var/data/hadoop/hdfs/dn</value>
    <final>true</final>
  </property>
</property>
  <!-- Dirección y puerto del interfaz web del namenode -->
    <name>dfs.namenode.http-address</name>
    <value>namenode:50070</value>
    <final>true</final>
  </property>
</property>
  <!-- Dirección y puerto del interfaz web del checkpoint node (aka secondary
namenode) -->
    <name>dfs.namenode.secondary.http-address</name>
    <value>checkpointnode:50090</value>
    <final>true</final>
  </property>
</configuration>

```

yarn-site.xml: configuración de YARN

```

<configuration>
  <property>
    <!-- El ResourceManager -->
    <name>yarn.resourcemanager.hostname</name>

```



```

    <value>resourcemanager</value>
    <final>true</final>
  </property>
  <property>
    <!-- Indica a los NodeManagers que tienen que implementar el servicio de barajado
mapreduce -->
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
    <final>true</final>
  </property>
  <property>
    <!-- Clase que implementa el servicio de barajado mapreduce -->
    <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    <final>true</final>
  </property>
  <property>
    <!-- Numero de cores del NodeManager (por defecto: 8) -->
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>1</value>
    <final>true</final>
  </property>
  <property>
    <!-- MB de RAM del NodeManager para los containers (por defecto: 8192) -->
    <!-- debe ser menor que la RAM física, para que funcionen otros servicios -->
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>3072</value>
    <final>true</final>
  </property>
  <property>
    <!-- Ratio memoria virtual/memoria fisica (por defecto: 2.1) -->
    <name>yarn.nodemanager.vmem-pmem-ratio</name>
    <value>2</value>
    <final>true</final>
  </property>
  <property>
    <!-- Numero maximo de cores por container (por defecto: 32) -->
    <name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>1</value>
    <final>true</final>
  </property>
  <property>
    <!-- Memoria minima (MB) permitida por container (por defecto: 1024) -->
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>1024</value>
    <final>true</final>
  </property>
  <property>
    <!-- Memoria máxima (MB) permitida por container (por defecto: 8192) -->
    <name>yarn.scheduler.maximum-allocation-mb</name>

```

```
<value>3072</value>
<final>true</final>
</property>
</configuration>
```

mapred-site.xml: configuración de MapReduce (copiamos primero el fichero mapred-site.xml.template a mapred-site.xml)

```
<configuration>
  <property>
    <!-- Framework que realiza el MapReduce -->
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
    <final>true</final>
  </property>
  <property>
    <!-- Memoria maxima (MB) por map (por defecto: 1536) -->
    <name>mapreduce.map.memory.mb</name>
    <value>1536</value>
    <final>true</final>
  </property>
  <property>
    <!-- Memoria maxima (MB) por reduce (por defecto: 3072)-->
    <name>mapreduce.reduce.memory.mb</name>
    <value>3072</value>
    <final>true</final>
  </property>
  <property>
    <!-- Heap maximo para las JVM de los maps (por defecto: -Xmx1024M)-->
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx1024M</value>
    <final>true</final>
  </property>
  <property>
    <!-- Heap maxima para las JVM de los reduces (por defecto: -Xmx2560M)-->
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx2048M</value>
    <final>true</final>
  </property>
  <property>
    <!-- El JobHistory Server -->
    <name>mapreduce.jobhistory.address</name>
    <value>jobhistoryserver:10020</value>
    <final>true</final>
  </property>
  <property>
    <!-- Interfaz web del JobHistory Server -->
    <name>mapreduce.jobhistory.webapp.address</name>
```

```
<value>jobhistoryserver:19888</value>
<final>true</final>
</property>
</configuration>
```

3.3. CONFIGURACIÓN DEL ENTORNO

16. Otros ficheros de configuración son los ficheros \$HADOOP_PREFIX/etc/hadoop/*-env.sh. Modificamos los siguientes:

hadoop-env.sh:

```
JAVA_HOME: definidlo como /usr/lib/jvm/java-8-openjdk-amd64
HADOOP_LOG_DIR: directorio donde se guardan los logs de hdfs. Definidlo como
/var/log/hadoop/hdfs
```

yarn-env.sh:

```
JAVA_HOME: definidlo como /usr/lib/jvm/java-8-openjdk-amd64
YARN_LOG_DIR: directorio donde se guardan los logs de YARN. Definidlo como
/var/log/hadoop/yarn
```

mapred-env.sh:

```
JAVA_HOME: definidlo como /usr/lib/jvm/java-8-openjdk-amd64
HADOOP_MAPRED_LOG_DIR: directorio donde se guardan los logs de MapReduce.
Definidlo como /var/log/hadoop/mapred
```

3.4. OBTENCIÓN DE LA IMAGEN BASE

17. Preparamos la máquina para poder clonarla:

- Dentro de la máquina de Azure, ejecuta como administrador:

```
waagent -deprovision+user
```

- Nos desconectamos de la máquina de Azure, ejecutando logout (o CTRL-D) las veces que sean necesarias
- Paramos y liberamos los recursos de la máquina de Azure ejecutando:

```
azure vm deallocate -g BaseInst -n HadoopBase
```

- Generalizamos esta máquina para poder lanzar otras iguales a ella:

```
azure vm generalize -g BaseInst -n HadoopBase
```

- Capturamos la imagen y una plantilla local para lanzar las nuevas máquinas:

```
azure vm capture -g BaseInst -n HadoopBase -p TCDM1617  
-t imagenbase-template.json
```

3.5. CREACIÓN DEL GRUPO HADOOP DENTRO DEL CLÚSTER

18. Creamos un nuevo grupo para las instancias del clúster:

```
azure group create -n HadoopGroup -l westeurope
```

3.6. CREACIÓN DE LA RED VIRTUAL DENTRO DEL CLÚSTER

19. Creamos una VNet que conectará las máquinas del clúster y una subnet dentro de la misma:

```
azure network vnet create -a 10.0.0.0/8 -g HadoopGroup  
-n HadoopVnet -l westeurope  
azure network vnet subnet create -a 10.0.0.0/24 -g  
HadoopGroup -e HadoopVnet -n HadoopSubnet
```

3.7. CREACIÓN DEL NAMENODE / RESOURCEMANAGER

Primero antes de nada, a partir de la plantilla JSON *imagenbase-template.json*, creamos las plantillas adaptadas para los tres tipos de máquinas:

1. Para el NameNode/Resource manager, copiamos *imagenbase-template.json* en *namenode-template.json* y modificamos en este fichero los siguientes "parameters":
 - En "vmName": anádele "defaultValue": "NameNode"
 - En "vmSize", cambiamos "defaultValue" a "Standard_D2_v2" (para tener una máquina virtual más potente para el NameNode)
 - En "adminUsername" le añadimos un "defaultValue" con un nombre de usuario (con el que nos queramos conectar a la máquina),
 - En "adminPassword" le añadimos un "defaultValue" con una contraseña

- La contraseña tiene que tener las mismas características que la que usamos cuando creamos la imagen base
 - En "resources":"storageProfile":"vhd":"uri", cambiamos el nombre del .vhd, añadiéndole NameNode- al principio (antes de osDisk)
2. Para el CheckpointNode, copiamos *imagenbase-template.json* en *checkpointnode-template.json* y modificamos en este fichero los siguientes parámetros:
- En "vmName": anádele "defaultValue": "CheckPointNode"
 - Añade los mismos valores de antes a "adminUsername" y "adminPassword"
 - En "resources":"storageProfile":"vhd":"uri", cambia el nombre del .vhd, añadiéndole CheckPointNode- al principio (antes de osDisk)
3. Para los DataNodes del 1 al 4, copia imagenbase-template.json en los 4 ficheros datanodeN-template.json, con N=1,2,3,4, y modifica en estos ficheros los siguientes parámetros:
- En "vmName": anádele "defaultValue": "DataNodeN", con N=1,2,3,4
 - Añade los mismos valores de antes a "adminUsername" y "adminPassword"
 - En "resources":"storageProfile":"vhd":"uri", cambia el nombre del .vhd, añadiéndole DataNodeN- al principio (antes de osDisk), con N=1,2,3,4

Ahora pasamos a la creación del NameNode propiamente dicho:

20. Creamos una IP pública para el NameNode

```
azure network public-ip create -g HadoopGroup -n
NameNodeIP --allocation-method Static -l westeurope
(Apuntamos el valor de la IP pública)
```

21. Creamos una interfaz de red para conectarla a esa IP, y le damos la IP privada 10.0.0.4

```
azure network nic create -a 10.0.0.4 -g HadoopGroup -n
NameNodeNIC -m HadoopVnet -k HadoopSubnet -p NameNodeIP -l
westeurope
```

22. Modificamos la plantilla *namenode-template.json* y añade a "parameters":"networkInterfaceId" un campo "defaultValue": con valor igual al ID del NIC creado (el que empieza por /subscriptions)

23. Desplegamos la máquina usando la plantilla json:

```
azure group deployment create -g HadoopGroup -n NameNode  
-f ./namenode-template.json
```

3.8. CREACIÓN DEL CHECKPOINTNODE / JOBHISTORY

24. Creamos una IP pública para el CheckPointNode

```
azure network public-ip create -g HadoopGroup -n  
CheckPointNodeIP --allocation-method Static -l westeurope  
(Apuntamos el valor de la IP pública)
```

25. Creamos una interfaz de red para conectarla a esa IP, y le damos la IP privada 10.0.0.5

```
azure network nic create -a 10.0.0.5 -g HadoopGroup -n  
CheckPointNodeNIC -m HadoopVnet -k HadoopSubnet -p  
CheckPointNodeIP -l westeurope
```

26. Modificamos la plantilla *checkpointnode-template.json* y añade a "parameters": "networkInterfaceId" un campo "defaultValue": con valor igual al ID del NIC creado (el que empieza por */subscriptions*)

27. Desplegamos la máquina usando la plantilla json:

```
azure group deployment create -g HadoopGroup -n  
CheckPointNode -f ./checkpointnode-template.json
```

3.9. CREACIÓN DE LOS DATANODES

28. El siguiente script Python se automatiza la creación de los 4 datanodes. Lo ejecutamos y apuntamos las IPs públicas de cada máquina (las IPs privadas deberían ser 10.0.0.6-10.0.0.9):

datanodes.py

```
#!/usr/bin/env python2  
# coding: utf-8  
  
from __future__ import with_statement, print_function  
import json  
from subprocess import call, Popen, PIPE
```

```

from collections import OrderedDict

privateip_start = 6
for n in range(1,5):
    # Crea IP pública
    sn = str(n)
    command = "azure network public-ip create HadoopGroup DataNode"+sn+"IP --allocation-
method Static -l westeurope"
    call(command, shell=True)
    # Crea el NIC
    command = "azure network nic create HadoopGroup DataNode"+sn+"NIC -k HadoopSubnet -m
HadoopVnet -p DataNode"+str(sn)+"IP -a 10.0.0."+str(privateip_start)+" -l westeurope"
    call(command, shell=True)
    privateip_start = privateip_start + 1
    # Obtiene la info del NIC
    command = "azure network nic show HadoopGroup DataNode"+sn+"NIC"
    outinfo = Popen(command, shell=True, stdout=PIPE)
    nicinfo = outinfo.stdout.readlines()
    nicid = None
    for line in nicinfo:
        sline = str(line)
        if "Id" in sline:
            nicid = sline.split(':')[2].strip()
            break
    # Modifica la template
    filename = "datanode"+sn+"-template.json"
    with open(filename, "r+") as jsonfile:
        jsondata = json.load(jsonfile, object_pairs_hook=OrderedDict)

    jsondata["parameters"][['networkInterfaceId']]['defaultValue'] = nicid
    with open(filename, "w+") as jsonfile:
        jsonfile.write(json.dumps(jsondata))

    # Realiza el despliegue
    command = "azure group deployment create HadoopGroup DataNode"+sn+" -f ./"+filename
    call(command, shell=True)

```

3.10. ADICIÓN DE LAS IPS PÚBLICAS AL DNS LOCAL

29. Para facilitar el acceso a la interfaz web de Hadoop, en tu PC (o en la máquina virtual) modifica el DNS local (fichero `/etc/hosts`) para añadir las IPs públicas de las máquinas del cluster

/etc/hosts

IP-pública-namenode	namenode resourcemanager
---------------------	--------------------------

IP-pública-checkpointnode	checkpointnode jobhistoryserver
IP-pública-datanode1	datanode1
IP-pública-datanode2	datanode2
IP-pública-datanode3	datanode3
IP-pública-datanode4	datanode4

3.11. ELIMINACIÓN DE LA IMAGEN BASE DE MÁQUINA HADOOP

30. Para reducir el consumo de recursos es conveniente que eliminemos la máquina virtual HadoopBase:

```
azure vm delete -g BaseInst -n HadoopBase
```


4. CÓMO ENCENDER / APAGAR MÁQUINAS VIRTUALES EN AZURE

31. Para encender y apagar las máquinas virtuales en Azure nos podemos crear dos sencillos scripts:

encenderMaquinas.sh

```
#!/bin/bash
for m in
{NameNode,CheckPointNode,DataNode1,DataNode2,DataNode3,DataNode4,DataNode
5};
do azure vm start -g HadoopGroup -n $m;
done
```

apagarMaquinas.sh

```
#!/bin/bash
for m in
{DataNode5,DataNode4,DataNode3,DataNode2,DataNode1,CheckPointNode,NameNod
e};
do azure vm deallocate -g HadoopGroup -n $m;
done
```

5. PREPARACIÓN DEL SISTEMA HDFS

32. Nos conectamos por ssh al Namenode, nos convertimos en el usuario hdmaster (sudo su – hdmaster), y como usuario hdmaster, iniciamos el HDFS ejecutando, en el Namenode:

```
$ hdfs namenode -format
Al finalizar debería indicar: "INFO common.Storage:
Storage directory /var/data/hadoop/hdfs/nn has been successfully
formatted."
```

5.1. ESPECIFICACIÓN DE DATANODES / NODEMANAGERS

33. En el fichero `$HADOOP_PREFIX/etc/hadoop/slaves` borramos `localhost` y ponemos las IPs internas de los cuatro DataNodes/NodeManagers (una IP por línea).

5.2. INICIO DE LOS DEMONIOS EN EL CLÚSTER

34. En el NameNode/ResourceManager, como usuario hdmaster, iniciamos los demonios del HDFS y YARN ejecutando:

```
$ $HADOOP_PREFIX/sbin/hadoop-daemon.sh start namenode
$ $HADOOP_PREFIX/sbin/yarn-daemon.sh start
resourcemanager
```

35. Comprobamos los ficheros de log en el directorio /var/log/hadoop del NameNode, para comprobar si hubo errores o no al iniciar.

36. También desde el NameNode, iniciamos el demonio del HDFS y YARN en los DataNodes:

```
$ $HADOOP_PREFIX/sbin/hadoop-daemons.sh start datanode
$ $HADOOP_PREFIX/sbin/yarn-daemons.sh start nodemanager
```

37. Nos conectamos por ssh al checkpointnode y lanzamos los demonios correspondientes al CheckPoint node (también conocido como Secondary NN) y al JobHistory server:

```
$ $HADOOP_PREFIX/sbin/hadoop-daemon.sh start
secondarynamenode
$ $HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh start
historyserver
```

38. Abrimos un navegador en nuestro PC y comprobamos las siguientes páginas para ver que todo funciona correctamente:

- <http://namenode:50070> interfaz web del HDFS

Namenode information - Mozilla Firefox

Tecnologías de comput... x Namenode information x WhatsApp x +

namenode:50070/dfshealth.html#tab-datanode-information Namenode information

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
datanode3:50010 (10.0.0.8:50010)	2	In Service	29.4 GB	1.53 GB	3.04 GB	24.83 GB	46	1.53 GB (5.22%)	0	2.7.3
datanode1:50010 (10.0.0.6:50010)	0	In Service	29.4 GB	1.98 GB	3.04 GB	24.38 GB	53	1.98 GB (6.73%)	0	2.7.3
datanode4:50010 (10.0.0.9:50010)	1	In Service	29.4 GB	1.56 GB	3.04 GB	24.8 GB	47	1.56 GB (5.32%)	0	2.7.3
datanode2:50010 (10.0.0.7:50010)	1	In Service	29.4 GB	1.51 GB	3.04 GB	24.85 GB	49	1.51 GB (5.14%)	0	2.7.3

Decommissioning


Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
------	--------------	-------------------------	------------------------------	--------------------------------------------------------

- <http://namenode:8088> interfaz web de YARN

Nodes of the cluster - Mozilla Firefox

Tecnologías de comput... x Nodes of the cluster x Namenode information x WhatsApp x +

namenode:8088/cluster/nodes



Nodes of the cluster

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
0	0	0	0	0	0 B	12 GB	0 B	0	4	0	4	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:3072, vCores:1>

Show 20 entries

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used
/default-rack		RUNNING	datanode4:44318	datanode4:8042	sáb nov 12 17:15:42 +0000 2016		0	0 B	3 GB	0
/default-rack		RUNNING	datanode3:52772	datanode3:8042	sáb nov 12 17:15:43 +0000 2016		0	0 B	3 GB	0
/default-rack		RUNNING	datanode2:35505	datanode2:8042	sáb nov 12 17:15:41 +0000 2016		0	0 B	3 GB	0
/default-rack		RUNNING	datanode1:40050	datanode1:8042	sáb nov 12 17:15:41 +0000 2016		0	0 B	3 GB	0

Showing 1 to 4 of 4 entries

- <http://checkpointnode:50090> interfaz web del CheckPoint node

SecondaryNameNode information - Mozilla Firefox

checkpointnode:50090/status.html

Overview

Version	2.7.3
Compiled	2016-08-18T01:41Z by root from branch-2.7.3
NameNode Address	namenode:9000
Started	12/11/2016 18:12:25
Last Checkpoint	Never
Checkpoint Period	3600 seconds
Checkpoint Transactions	1000000

Checkpoint Image URI

- file:///var/tmp/dfs/namesecondary

Checkpoint Editlog URI

- file:///var/tmp/dfs/namesecondary

- <http://checkpointnode:19888/> interfaz web del JobHistory server

JobHistory - Mozilla Firefox

checkpointnode:19888/jobhistory

Logged in as: hdfs

JobHistory

Retired Jobs

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016.11.06 23:02:21 UTC	2016.11.06 23:02:28 UTC	2016.11.06 23:02:51 UTC	job_1478471652556_0007	TeraValidate	hdmaster	default	SUCCEEDED	1	1	1	1
2016.11.06 22:59:48 UTC	2016.11.06 22:59:54 UTC	2016.11.06 23:01:59 UTC	job_1478471652556_0006	TeraSort	hdmaster	default	SUCCEEDED	16	16	1	1
2016.11.06 22:57:56 UTC	2016.11.06 22:58:03 UTC	2016.11.06 22:58:25 UTC	job_1478471652556_0004	TeraGen	hdmaster	default	SUCCEEDED	2	2	0	0
2016.11.06 22:56:31 UTC	2016.11.06 22:56:37 UTC	2016.11.06 22:56:59 UTC	job_1478471652556_0003	hadoop-mapreduce-client-jobclient-2.7.3-tests.jar	hdmaster	default	SUCCEEDED	10	10	1	1
2016.11.06 22:53:19 UTC	2016.11.06 22:53:25 UTC	2016.11.06 22:54:03 UTC	job_1478471652556_0002	hadoop-mapreduce-client-jobclient-2.7.3-tests.jar	hdmaster	default	SUCCEEDED	10	10	1	1
2016.11.06 22:51:23 UTC	2016.11.06 22:51:33 UTC	2016.11.06 22:52:22 UTC	job_1478471652556_0001	hadoop-mapreduce-client-jobclient-	hdmaster	default	SUCCEEDED	10	10	1	1

5.3. PARADA DE LOS DEMONIOS EN EL CLÚSTER

39. El proceso de parar los demonios es el inverso del seguido para iniciarlos, cambiando start por stop. (No hay que pararlos de momento, a menos que tengamos que apagar las máquinas. Para evitar problemas, es recomendable que siempre que detengamos las máquinas detengamos los demonios antes). Para ello podemos usar el siguiente shell script:

pararDemonios.sh (En el NameNode)

```
#!/bin/bash
$HADOOP_PREFIX/sbin/yarn-daemons.sh stop nodemanager
$HADOOP_PREFIX/sbin/hadoop-daemons.sh stop datanode
$HADOOP_PREFIX/sbin/yarn-daemon.sh stop
$HADOOP_PREFIX/sbin/hadoop-daemon.sh stop namenode
```

pararDemonios.sh (En el CheckPointNode)

```
#!/bin/bash
$HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh stop historyserver
$HADOOP_PREFIX/sbin/hadoop-daemon.sh start secondarynamenode
```

5.4. PRUEBA DE EJEMPLO

40. Como test de nuestra instalación, podemos ejecutar un ejemplo de MapReduce. Para ello, desde el NameNode, como usuario hdmaster, ejecutamos lo siguiente:

```
$ export
YARN_EXAMPLES=$HADOOP_PREFIX/share/hadoop/mapreduce
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-
2.7.*.jar pi 16 1000
```

41. Mientras se ejecuta, podemos comprobar en el interfaz web del YARN la evolución. Al terminar, podemos comprobar también que se ha guardado información de la ejecución en el JobHistory server.

5.5. CREACIÓN DE LOS USUARIOS EN HDFS

42. En el Namenode, como usuario hdmaster, creamos un directorio en HDFS (dentro de /user) para el usuario que vaya a ejecutar las tareas MapReduce, que será el usuario que creamos para acceder a las máquinas de azure:

```
$ hdfs dfs -mkdir -p /user/ponemos_un_nombre
$ hdfs dfs -chown ponemos_un_nombre
/user/ponemos_un_nombre
$ hdfs dfs -ls /user
```

43. Debemos dar los permisos adecuados en `/tmp` del HDFS para que el usuario pueda lanzar aplicaciones MapReduce:

```
$ hdfs dfs -chmod 1777 /tmp
$ hdfs dfs -chmod 1777 /tmp/hadoop-yarn
$ hdfs dfs -chmod 1777 /tmp/hadoop-yarn/staging
$ hdfs dfs -chmod 1777 /tmp/hadoop-yarn/staging/history
```

5.6. PRUEBA COMO USUARIO NO PRIVILEGIADO (NO HDMaster)

44. En el Namenode, como usuario normal (no hdmaster), añadimos al `.bashrc` las siguientes líneas:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_PREFIX=/opt/yarn/hadoop
export PATH=$PATH:$HADOOP_PREFIX/bin
```

45. Reiniciamos sesión y comprobamos que podemos acceder al HDFS ejecutando:

```
$ hdfs dfs -ls
(Este comando no debería dar ninguna salida (ya que el
usuario no tiene ningún fichero), ni ningún mensaje de error)
```

6. INSTALAR HADOOP EN LOCAL

46. En nuestro PC descargamos la misma versión de Hadoop y la descomprimos en algún directorio. Para mayor comodidad, podemos crear un enlace:

```
$ ln -s hadoop-2.7.3 hadoop
```

47. Definid la variable `HADOOP_PREFIX` al directorio donde lo hayáis descargado, por ejemplo:

```
$ export HADOOP_PREFIX=$HOME/hadoop
```

48. Copiamos los ficheros de configuración de Hadoop desde el NameNode a nuestra máquina:

```
$ scp usuario@ip-externa-del-  
namenode:/opt/yarn/hadoop/etc/hadoop/*site.xml  
$HADOOP_PREFIX/etc/hadoop/
```

49. Para más comodidad, podemos poner en el path el directorio bin de Hadoop (también lo podemos poner en el *.bashrc* para que quede):

```
$ export PATH=$PATH:$HADOOP_PREFIX/bin
```

50. Comprobamos que funciona, por ejemplo con el comando:

```
$ hdfs dfs -ls /user
```

7. EJECUCIÓN DE WORDCOUNT

51. Creamos el proyecto WordCount en Eclipse y compilamos el jar.

52. Podemos hacer una primera ejecución en local en el propio Eclipse poniendo en el *Run Configuration* los siguientes argumentos en la máquina virtual:

```
-Dlog4j.configuration =  
file:///Path_completo_instalación_hadoop/etc/hadoop/log4j.properties
```

53. Para ejecutarlo en el clúster:

- Copiamos mediante *scp* al NameNode el fichero *target/wordcount-0.0.1-SNAPSHOT.jar*
- Nos conectamos por *ssh* al NameNode y, como usuario no privilegiado, ejecutamos:

```
$ yarn jar wordcount-0.0.1-SNAPSHOT.jar libros salidawc
```

Salida de la Ejecución:

```
16/11/12 17:48:06 INFO input.FileInputFormat: Total input paths to process : 16
16/11/12 17:48:07 INFO mapreduce.JobSubmitter: number of splits:20
16/11/12 17:48:07 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1478970693939_0001
16/11/12 17:48:07 INFO impl.YarnClientImpl: Submitted application application_1478970693939_0001
16/11/12 17:48:07 INFO mapreduce.Job: The url to track the job: http://resourcemanager:8088/proxy/application_1478970693939_0001/
16/11/12 17:48:07 INFO mapreduce.Job: Running job: job_1478970693939_0001
16/11/12 17:48:19 INFO mapreduce.Job: Job job_1478970693939_0001 running in uber mode : false
16/11/12 17:48:19 INFO mapreduce.Job: map 0% reduce 0%
16/11/12 17:48:33 INFO mapreduce.Job: map 1% reduce 0%
16/11/12 17:48:39 INFO mapreduce.Job: map 2% reduce 0%
16/11/12 17:48:45 INFO mapreduce.Job: map 3% reduce 0%
16/11/12 17:48:53 INFO mapreduce.Job: map 4% reduce 0%
16/11/12 17:48:57 INFO mapreduce.Job: map 5% reduce 0%
16/11/12 17:49:05 INFO mapreduce.Job: map 6% reduce 0%
16/11/12 17:49:12 INFO mapreduce.Job: map 7% reduce 0%
16/11/12 17:49:19 INFO mapreduce.Job: map 8% reduce 0%
16/11/12 17:49:24 INFO mapreduce.Job: map 9% reduce 0%
16/11/12 17:49:28 INFO mapreduce.Job: map 11% reduce 0%
16/11/12 17:49:37 INFO mapreduce.Job: map 13% reduce 0%
16/11/12 17:49:38 INFO mapreduce.Job: map 14% reduce 0%
16/11/12 17:49:43 INFO mapreduce.Job: map 19% reduce 0%
16/11/12 17:49:46 INFO mapreduce.Job: map 21% reduce 0%
16/11/12 17:49:49 INFO mapreduce.Job: map 26% reduce 0%
16/11/12 17:49:55 INFO mapreduce.Job: map 31% reduce 0%
16/11/12 17:49:58 INFO mapreduce.Job: map 32% reduce 0%
16/11/12 17:50:00 INFO mapreduce.Job: map 37% reduce 0%
16/11/12 17:50:07 INFO mapreduce.Job: map 43% reduce 0%
16/11/12 17:50:11 INFO mapreduce.Job: map 48% reduce 0%
16/11/12 17:50:16 INFO mapreduce.Job: map 53% reduce 0%
16/11/12 17:50:19 INFO mapreduce.Job: map 54% reduce 0%
16/11/12 17:50:22 INFO mapreduce.Job: map 59% reduce 0%
16/11/12 17:50:26 INFO mapreduce.Job: map 64% reduce 0%
16/11/12 17:50:27 INFO mapreduce.Job: map 65% reduce 0%
16/11/12 17:50:31 INFO mapreduce.Job: map 70% reduce 0%
16/11/12 17:50:34 INFO mapreduce.Job: map 72% reduce 0%
16/11/12 17:50:36 INFO mapreduce.Job: map 77% reduce 0%
16/11/12 17:50:39 INFO mapreduce.Job: map 82% reduce 0%
16/11/12 17:50:40 INFO mapreduce.Job: map 87% reduce 0%
16/11/12 17:50:44 INFO mapreduce.Job: map 92% reduce 0%
16/11/12 17:50:45 INFO mapreduce.Job: map 97% reduce 0%
16/11/12 17:50:46 INFO mapreduce.Job: map 98% reduce 0%
16/11/12 17:50:54 INFO mapreduce.Job: map 98% reduce 32%
16/11/12 17:51:08 INFO mapreduce.Job: map 100% reduce 32%
16/11/12 17:51:11 INFO mapreduce.Job: map 100% reduce 100%
16/11/12 17:51:12 INFO mapreduce.Job: Job job_1478970693939_0001 completed successfully
16/11/12 17:51:12 INFO mapreduce.Job: Counters: 51
  File System Counters
    FILE: Number of bytes read=56822110
    FILE: Number of bytes written=68137160
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=337499621
    HDFS: Number of bytes written=2017550
    HDFS: Number of read operations=63
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Killed map tasks=1
    Launched map tasks=21
    Launched reduce tasks=1
    Data-local map tasks=20
    Rack-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=932218
    Total time spent by all reduces in occupied slots (ms)=79704
    Total time spent by all map tasks (ms)=466109
    Total time spent by all reduce tasks (ms)=26568
    Total vcore-milliseconds taken by all map tasks=466109
    Total vcore-milliseconds taken by all reduce tasks=26568
    Total megabyte-milliseconds taken by all map tasks=715943424
    Total megabyte-milliseconds taken by all reduce tasks=81616896
  Map-Reduce Framework
    Map input records=455724
    Map output records=128645597
    Map output bytes=1473559853
    Map output materialized bytes=8818048
    Input split bytes=2404
    Combine input records=131819927
    Combine output records=3760577
    Reduce input groups=158236
    Reduce shuffle bytes=8818048
    Reduce input records=586247
    Reduce output records=158236
    Spilled Records=4346824
    Shuffled Maps =20
    Failed Shuffles=0
    Merged Map outputs=20
    GC time elapsed (ms)=3959
    CPU time spent (ms)=360940
    Physical memory (bytes) snapshot=4574191616
    Virtual memory (bytes) snapshot=58723725312
    Total committed heap usage (bytes)=3249160192
  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=337497217
  File Output Format Counters
    Bytes Written=2017550
moises@NameNode:~$
```


- En este caso, se usan todos los ficheros del directorio y los datos se leen y se guardan en HDFS (directorio /user/ec2-user)
- Comprobamos el proceso de ejecución mediante en interfaz web de YARN (namenode:8088)
- Descargamos el fichero de salida a tu PC a través del interfaz web de HDFS (namenode:50070, menu Utilities -> Browse filesystem)

salidaWordCount.out (Fichero descargado y renombrado)

```

a      45639
aa     1038
aachen 965
aaliyah 2063
aardvark 2056
aardvarks 992
aaron 1005
ab     1019
aba    8
abaci 1032
aback 1049
abacus 2031
abacuses 989
abad 70
abade 2
abadejo 4
abades 11
abadesa 33
abadesas 1
abadía 7
abaft 990
abaja 1
abajada 1
abajan 1
abajar 7
abajarse 3
abajase 1
abaje 1
abajen 1
abajes1
abajito1
abajo 409
abajó 2
abalanzaba 1
abalanzaban 1
abalanzado 1
abalanzaron 2
abalanzarse 1

```

abalanzándose	1
abalanzáronse	1
abalanzó	2
aballa	1
aballar	3
aballemos	1
abalone	2081
abalones	993
abalorio	3

(...) //Corto el fichero aquí por ser demasiado largo

8. EJECUCIÓN DE BENCHMARKS

54. La instalación de Hadoop incluye un conjunto de programas de test y ejemplos. Para ver la lista de disponibles, ejecutar, en el Namenode como usuario hdmaster:

```
$ export
YARN_EXAMPLES=$HADOOP_PREFIX/share/hadoop/mapreduce
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-
jobclient-*tests.jar
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-
*.jar
```

55. Benchmarking de HDFS con TestDFSIO:

- Escribir 10 ficheros de 100 MB cada uno

```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-
jobclient-*tests.jar TestDFSIO -write -nrFiles 10 -fileSize 100
```

TestDFSIO_Write_results.log

```
----- TestDFSIO ----- : write
    Date & time: Sun Nov 06 22:54:07 UTC 2016
    Number of files: 10
Total MBytes processed: 1000.0
    Throughput mb/sec: 35.08771929824562
Average IO rate mb/sec: 38.19384002685547
    IO rate std deviation: 9.969821064083751
    Test exec time sec: 48.941
```

- Leer 10 ficheros de 100 MB cada uno

```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-
jobclient-*tests.jar TestDFSIO -read -nrFiles 10 -fileSize 100
```

TestDFSIO_Read_results.log

```
----- TestDFSIO ----- : read
      Date & time: Sun Nov 06 22:57:01 UTC 2016
      Number of files: 10
Total MBytes processed: 1000.0
  Throughput mb/sec: 219.63540522732265
Average IO rate mb/sec: 305.41790771484375
IO rate std deviation: 142.13839566453248
Test exec time sec: 31.689
```

- Limpiar los ficheros creados en el test

```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-client-
jobclient-*tests.jar TestDFSIO -clean
```

56. Benchmarking el MapReduce con un TeraSort:

- En el NameNode, como usuario hdmaster, ejecutamos el siguiente comando, que genera un fichero con 10 millones de filas de 100 bytes cada una (~1GB), en el directorio terasortin

```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-
*.jar teragen 10000000 terasortin
```

- Después ejecutamos un sort sobre los datos creados, salida a terasortout:

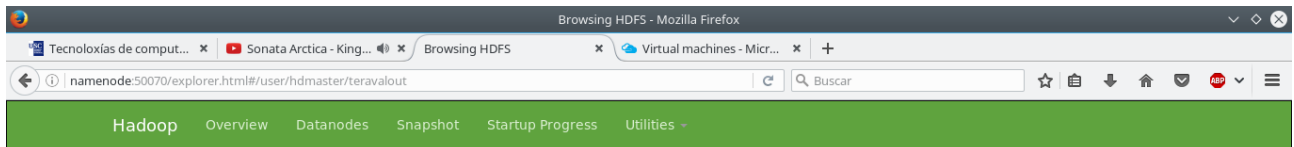
```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-
*.jar terasort terasortin terasortout
```

- Verificamos que la ordenación fue correcta correctamente:

```
$ yarn jar $YARN_EXAMPLES/hadoop-mapreduce-examples-
```

*.jar teravalidate terasortout teravalout

- Revisamos la salida en el directorio teravalout para ver que no dio errores.



9. AÑADIR Y QUITAR DATANODES / JOBTRACKERS

57. Aunque no es estrictamente necesario para añadir o retirar nodos del cluster, es conveniente tener una lista en la que podamos indicar los nodos que se pueden añadir o retirar del cluster. Para ello, hacemos lo siguiente en el NameNode (como usuario hdmaster):

58. Paramos los demonios.

59. Creamos cuatro ficheros:

```
$touch $HADOOP_PREFIX/etc/hadoop/dfs.include  
$touch $HADOOP_PREFIX/etc/hadoop/dfs.exclude  
$touch $HADOOP_PREFIX/etc/hadoop/yarn.include  
$touch $HADOOP_PREFIX/etc/hadoop/yarn.exclude  
(inicialmente vacíos)
```

60. En los ficheros *dfs.include* y *yarn.include*, ponemos los nombres de todos los DataNodes/NodeManagers que queramos que estén en el clúster. Dejamos los ficheros *dfs.exclude* y *yarn.exclude* vacíos.

61. En el fichero de configuración *hdfs-site.xml*, añadimos dos propiedades:

1. **dfs.hosts:** nombre de un fichero con lista de hosts que pueden actuar como DataNodes; si el fichero está vacío, cualquier nodo está permitido. Darle como valor, el path al fichero *dfs.include*
2. **dfs.hosts.exclude:** nombre de un fichero con lista de hosts que no pueden actuar como DataNodes; si el fichero está vacío, ninguno está excluido. Darle como valor, el path al fichero *dfs.exclude*

62. En el fichero *yarn-site.xml*, añadid dos propiedades:

1. **yarn.resourcemanager.nodes.include-path:** nombre de un fichero con lista de hosts que pueden actuar como NodeManagers; si el fichero está vacío, cualquier nodo está permitido. Darle como valor, el path al fichero *yarn.include*
2. **yarn.resourcemanager.nodes.exclude-path:** nombre de un fichero con lista de hosts que no pueden actuar como NodeManagers; si el fichero está vacío, ninguno está excluido. Darle como valor, el path al fichero *yarn.exclude*

63. Reiniciamos los demonios.

64. Añadimos ahora un nuevo datanode/nodemanager al clúster.

65. En el Namenode, añadimos el nombre de nuevo nodo (datanode5) en los ficheros *dfs-include* y *yarn-include*

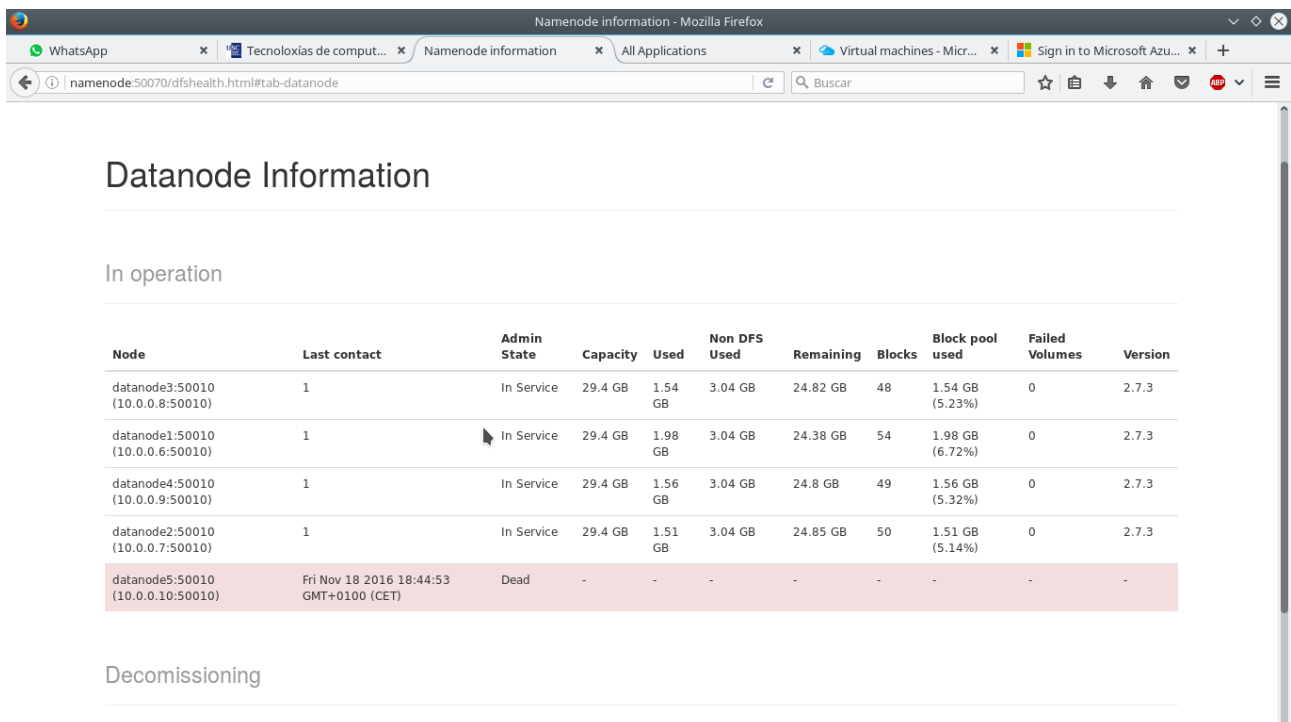
66. Actualizamos el NameNode con los nuevos DataNodes ejecutando:

```
$ hdfs dfsadmin -refreshNodes
```

67. Actualizamos el JobTracker con los nuevos TaskTrackers ejecutando:

```
$ yarn rmadmin -refreshNodes
```

68. Comprobamos en el interfaz web del NameNode, que este nuevo nodo aparece como Dead



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
datanode3:50010 (10.0.0.8:50010)	1	In Service	29.4 GB	1.54 GB	3.04 GB	24.82 GB	48	1.54 GB (5.23%)	0	2.7.3
datanode1:50010 (10.0.0.6:50010)	1	In Service	29.4 GB	1.98 GB	3.04 GB	24.38 GB	54	1.98 GB (6.72%)	0	2.7.3
datanode4:50010 (10.0.0.9:50010)	1	In Service	29.4 GB	1.56 GB	3.04 GB	24.8 GB	49	1.56 GB (5.32%)	0	2.7.3
datanode2:50010 (10.0.0.7:50010)	1	In Service	29.4 GB	1.51 GB	3.04 GB	24.85 GB	50	1.51 GB (5.14%)	0	2.7.3
datanode5:50010 (10.0.0.10:50010)	Fri Nov 18 2016 18:44:53 GMT+0100 (CET)	Dead	-	-	-	-	-	-	-	-

Decommissioning

69. Iniciamos un nuevo nodo como el resto de nodos del clúster (con lo que tendrá ya Hadoop instalado), con la IP interna 10.0.0.10

70. Nos conectamos por ssh al nuevo nodo e iniciamos manualmente los demonios del datanode/jobtracker en el nuevo nodo (como usuario hdmaster):

```
$ $HADOOP_PREFIX/sbin/hadoop-daemon.sh start datanode
$ $HADOOP_PREFIX/sbin/yarn-daemon.sh start nodemanager
```

71. El nodo contacta automáticamente con el Namenode y se unirá al cluster

72. Usa (en el NameNode) los comandos *hdfs dfsadmin -report* y *yarn node -list* para comprobar que el nuevo nodo se ha añadido. Puedes comprobarlo también el interfaz web del NameNode y de YARN.

The screenshot shows the 'Datanode Information' page in the Hadoop NameNode web interface. It displays a table of nodes in operation and a section for decommissioning.

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
datanode5:50010 (10.0.0.10:50010)	2	In Service	29.4 GB	32 KB	3.04 GB	26.36 GB	0	32 KB (0%)	0	2.7.3
datanode3:50010 (10.0.0.8:50010)	1	In Service	29.4 GB	1.54 GB	3.04 GB	24.82 GB	48	1.54 GB (5.23%)	0	2.7.3
datanode1:50010 (10.0.0.6:50010)	2	In Service	29.4 GB	1.98 GB	3.04 GB	24.38 GB	54	1.98 GB (6.72%)	0	2.7.3
datanode4:50010 (10.0.0.9:50010)	2	In Service	29.4 GB	1.56 GB	3.04 GB	24.8 GB	49	1.56 GB (5.32%)	0	2.7.3
datanode2:50010 (10.0.0.7:50010)	2	In Service	29.4 GB	1.51 GB	3.04 GB	24.85 GB	50	1.51 GB (5.14%)	0	2.7.3

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
------	--------------	-------------------------	------------------------------	--------------------------------------------------------

73. Añade la IP interna del nodo al fichero slaves en el Namenode para que se inicien/paren los demonios cuando usemos *hadoop-daemons* y/o *yarn-daemons*

74. El nuevo nodo, inicialmente está vacío (no tiene datos de HDFS), con lo que el clúster estará desbalanceado. Se puede forzar el balanceo ejecutando, en el NameNode:

```
$ hdfs dfs balancer
```

75. En nuestro caso, al tener muy pocos datos, el HDFS considera que no es necesario balancear. Para más información, ver <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html#balancer>

76. Retiramos ahora un DataNode

77. En principio, el apagado de un DataNode no debería afectar al clúster. Sin embargo, si queremos hacer un apagado programado de un DataNode es preferible advertir al NameNode previamente. Utiliza los siguientes pasos para eliminar, por ejemplo, el Datanode5.

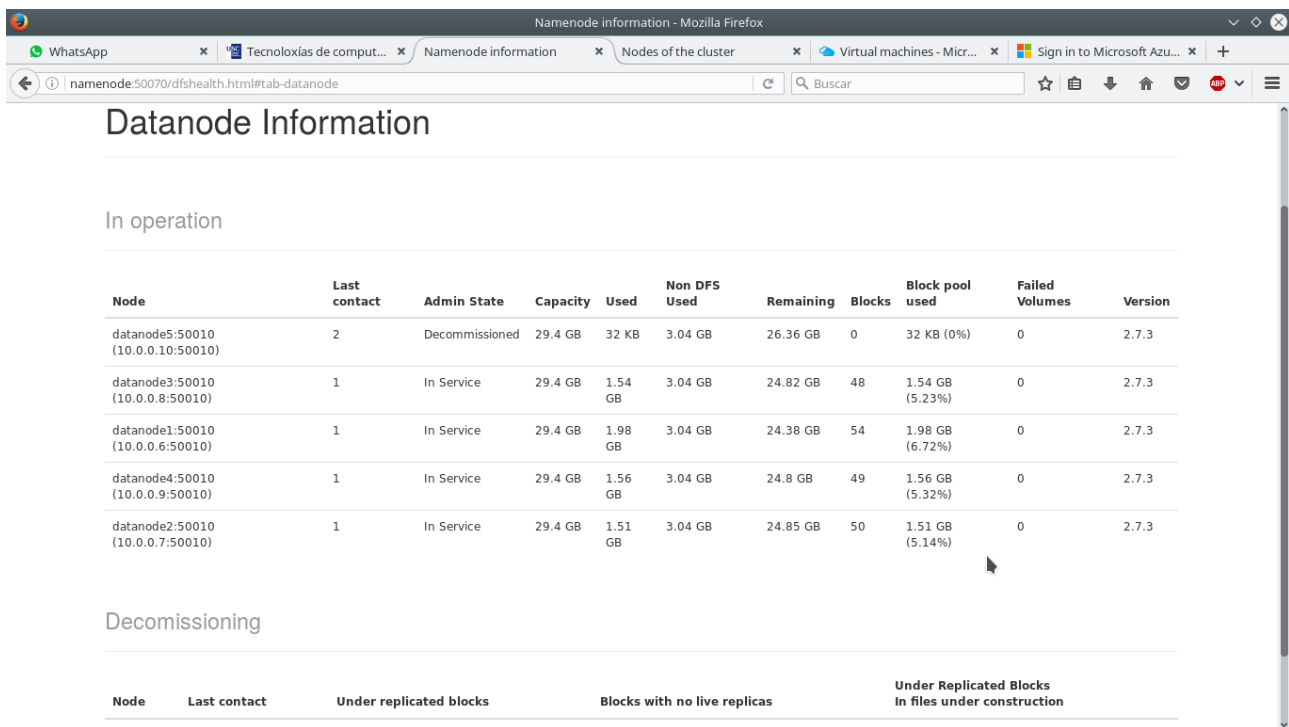
78. Pasos:

1. Ponemos el nombre del nodo o nodos que queremos retirar en el fichero *dfs-*

exclude y *yarn-exclude* y ejecutar

```
$ hdfs dfsadmin -refreshNodes
$ yarn rmadmin -refreshNodes
```

79. Comprobamos que al cabo de un rato, en el interfaz web o mediante los comandos los comandos *hdfs dfsadmin -report* y *yarn node -list*, que el/los nodo(s) excluido(s) aparece(n) que está(n) Decomissioned en HDFS y YARN



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
datanode5:50010 (10.0.0.10:50010)	2	Decommissioned	29.4 GB	32 KB	3.04 GB	26.36 GB	0	32 KB (0%)	0	2.7.3
datanode3:50010 (10.0.0.8:50010)	1	In Service	29.4 GB	1.54 GB	3.04 GB	24.82 GB	48	1.54 GB (5.23%)	0	2.7.3
datanode1:50010 (10.0.0.6:50010)	1	In Service	29.4 GB	1.98 GB	3.04 GB	24.38 GB	54	1.98 GB (6.72%)	0	2.7.3
datanode4:50010 (10.0.0.9:50010)	1	In Service	29.4 GB	1.56 GB	3.04 GB	24.8 GB	49	1.56 GB (5.32%)	0	2.7.3
datanode2:50010 (10.0.0.7:50010)	1	In Service	29.4 GB	1.51 GB	3.04 GB	24.85 GB	50	1.51 GB (5.14%)	0	2.7.3

Decommissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
------	--------------	-------------------------	------------------------------	-----------------------------------------------------

80. Ya podríamos parar los demonios en el nodo decomisionado y apagarlo (podemos borrar la MV DataNode5). Si no queremos volver a incluirla en el clúster:

81. Eliminamos el/los nodo(s) de los ficheros include y exclude y ejecutar otra vez

```
$ hdfs dfsadmin -refreshNodes
$ yarn rmadmin -refreshNodes
```

82. Eliminamos el/los nodo(s) del fichero slaves

10. RACK AWARENESS

83. Apagamos los demonios.

84. Creamos un fichero `$HADOOP_PREFIX/etc/hadoop/topology.data` que tenga en cada línea la IP de uno de los Nodemanagers y el rack donde está, como en este ejemplo (cambiando las IPs por las tuyas)

```
10.0.0.6      /rack1
10.0.0.7      /rack1
10.0.0.8      /rack2
10.0.0.9      /rack2
```

85. Creamos un script de bash `$HADOOP_PREFIX/etc/hadoop/topology.script` como el siguiente (fuente: http://wiki.apache.org/hadoop/topology_rack_awareness_scripts). Le Damos permisos de ejecución (`chmod +x topology.script`).

topology.script

```
#!/bin/bash

HADOOP_CONF=/opt/yarn/hadoop/etc/hadoop
while [ $# -gt 0 ] ; do
    nodeArg=$1
    exec< ${HADOOP_CONF}/topology.data
    result=""
    while read line ; do
        ar=( $line )
        if [ "${ar[0]}" = "$nodeArg" ] ; then
            result="${ar[1]}"
        fi
    done
    shift
    if [ -z "$result" ] ; then
        echo -n "/default-rack "
    else
        echo -n "$result "
    fi
done
```

86. Definimos en el fichero `core-site.xml` la propiedad `net.topology.script.file.name` y le damos como valor el path completo al script.

87. Iniciamos los demonios y comprobamos que se han identificado los racks ejecutando:

```
$ hdfs dfsadmin -printTopology
```

```
hdmaster@NameNode:~$ hdfs dfsadmin -printTopology
Rack: /rack1
  10.0.0.6:50010 (datanode1)
  10.0.0.7:50010 (datanode2)
Rack: /rack2
  10.0.0.8:50010 (datanode3)
  10.0.0.9:50010 (datanode4)
hdmaster@NameNode:~$ hdfs dfsadmin -report
Configured Capacity: 126286282752 (117.61 GB)
Present Capacity: 113220235264 (105.44 GB)
DFS Remaining: 106147045376 (98.86 GB)
DFS Used: 7073189888 (6.59 GB)
DFS Used%: 6.25%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-----
Live datanodes (4):
Name: 10.0.0.6:50010 (datanode1)
Hostname: datanode1
Rack: /rack1
Decommission Status : Normal
Configured Capacity: 31571570688 (29.40 GB)
DFS Used: 2121170944 (1.98 GB)
Non DFS Used: 3268083712 (3.04 GB)
DFS Remaining: 26182316032 (24.38 GB)
DFS Used%: 6.72%
DFS Remaining%: 82.93%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
```