

Práctica 1: Instalación de cluster de Hadoop

El contenido de este documento está disponible en https://github.com/Loksly/tcdm_2016

Introducción

Este repositorio tiene por finalidad la de documentar el desarrollo de la primera práctica de la asignatura de TCDM.

Pasos llevados a cabo:

Parte obligatoria:

- ☒ 1. Instalar manualmente un cluster Hadoop v2 con máquinas virtuales en Azure
- ☒ 2. Ejecutar el programa WordCount en Java en el cluster
- ☒ 3. Ejecutar benchmarks para HDFS y MapReduce en el cluster
- ☒ 4. Añadir y retirar nodos del cluster Hadoop
- ☒ 5. Hacer que el cluster sea rack-aware

Parte opcional:

- ☐ 1. Despliegue de un cluster usando Apache Ambari
- ☒ 2. Ejecución del WordCount en un cluster Azure HDInsight

1. Instalar manualmente un cluster Hadoop v2 con máquinas virtuales en Azure:

Teniendo *nodejs* y *npm* instalado, instalar suite [azure](#):

```
npm install -g azure
azure login
```

Después se deben seguir los pasos para obtener la autenticación que permite vincular la suite con el usuario.

Una vez concluido ese proceso, a continuación, se deben indicar que se va a hacer uso de los servicios Compute, Storage y Network, mediante los comandos:

```
azure provider register Microsoft.Compute
azure provider register Microsoft.Storage
azure provider register Microsoft.Network
```

Crear un *Resource Group* que se ejecutará en la zona del Oeste de Europa:

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

Ahora vamos a crear una plantilla de máquina virtual que luego replicaremos varias veces para el despliegue de todos los nodos necesarios, para ello es necesario elegir un nombre de usuario y una contraseña segura (que tiene que tener al menos 8 caracteres, conteniendo al menos una minúscula, una mayúscula, un número y un signo de entre !@#\$%^&+=). Para el tipo de máquina virtual vamos a elegir *Standard_D1_v2* pero es posible ver más alternativas en la web de [Azure](#)

```
nombrequesuario=el_que_se_quiera
pass=passwordseguro
tipovm=Standard_D1_v2
azure vm quick-create -g BaseInst -n HadoopBase -y Linux -Q Debian -u $nombrequesuario -p $pass -z Standard_D1_v2 -l w
```

Al terminar se muestra un informe que incluye la IP de la nueva máquina virtual (campo *Network Profile/Network Interfaces/Network Interface #1/Location/Public IP Address*), aunque también es posible acceder al mismo desde [el panel de](#)

[control de Azure](#). Lo siguiente será descargarse el script que realiza la instalación de *Hadoop*, para ello realiza un ssh a la máquina recién creada y utiliza la contraseña seleccionada.

```
wget https://github.com/Loksly/tcdm_2016/raw/master/scripts/hadoop_install.sh
sudo bash
bash hadoop_install.sh

su - hdmaster
wget https://github.com/Loksly/tcdm_2016/raw/master/scripts/hdmaster_config.sh
bash hdmaster_config.sh
hadoop version #debe mostrar la versión de Hadoop, si no es así repasar este último paso.
```

Ahora hay que configurar hadoop como usuario *hdmaster*:

```
wget https://github.com/Loksly/tcdm_2016/raw/master/scripts/hadoop_config.sh
bash hadoop_config.sh
```

Otros ficheros de configuración son los ficheros `$HADOOP_PREFIX/etc/hadoop/*-env.sh`. Hay que modificar 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`

Crear las VMs

Obtener la imagen lista para la replicación:

Dentro de la máquina de Azure, ejecuta como administrador:

```
waagent -deprovision+user
```

Desconectate de la máquina de Azure, ejecutando `logout` (o `CTRL-D`) las veces que sean necesarias

```
Deallocate la máquina de Azure ejecutando:
azure vm deallocate -g BaseInst -n HadoopBase
```

Generaliza esta máquina para poder lanzar otras iguales a ella

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

Captura la imagen y una plantilla local para lanzar las nuevas máquinas

```
azure vm capture -g BaseInst -n HadoopBase -p TCDM1617 -t imagenbase-template.json
azure group create -n HadoopGroup -l westeurope
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
```

Para registrar una nueva máquina virtual con su interfaz de red es necesario ejecutar estos pasos:

```
azure network public-ip create -g HadoopGroup -n NameNodeIP --allocation-method Static -l westeurope
azure network nic create -a 10.0.0.4 -g HadoopGroup -n NameNodeNIC -m HadoopVnet -k HadoopSubnet -p NameNodeIP -l wes
```

Es necesario clonar el fichero *imagenbase-template.json* y modificar los valores de los campos: *vmName*, *adminUsername* y *adminPassword*, así como el campo *networkInterfaceId* con el obtenido con el comando anterior, asignarle un nombre y luego

desplegar la VM así:

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

Hay que repetir este proceso para el checkpoint node modificando en donde sea necesario.

Para desplegar los datanodes se puede usar el script disponible en:

https://github.com/Loksly/tcdm_2016/blob/master/scripts/deploynodes.py

Una vez se ha concluido el proceso y para reducir el consumo de recursos es conveniente eliminar la máquina virtual HadoopBase:

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

Arrancar las VMs en cualquier momento o pararlas:

En el ordenador local están disponibles dos scripts:

- Para arrancar: https://github.com/Loksly/tcdm_2016/raw/master/scripts/wakeup.sh
- Para parar: https://github.com/Loksly/tcdm_2016/raw/master/scripts/sleeps.sh

Nota: es posible que muestre un error relacionado con el DataNode5, si aún no se ha creado, se creará posteriormente.

Arrancar Yarn y los data nodes.

En el equipo NameNode, nos conectamos con un `ssh usuario@namenode`, y después cambiamos de usuario con:

```
sudo su - hdmaster
cd
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/namenode-run.sh"
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/namenode-stop.sh"
bash namenode-run.sh
#para pararlo: bash namenode-stop.sh
```

Y ahora con el equipo checkpointnode, nos conectamos con un `ssh usuario@checkpointnode`, y después cambiamos de usuario con:

```
sudo su - hdmaster
cd
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/checkpointnode-run.sh"
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/checkpointnode-stop.sh"
bash checkpointnode-run.sh
#para pararlo: bash checkpointnode-stop.sh
```

NOMBRE	ESTADO	GRUPO DE RECURSOS	UBICACIÓN	SUSCRIPCIÓN	DIRECCIÓN IP PÚBLICA	ID. DE SUSCRIPCIÓN
CheckPointNode	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.10	7b7c1b11-1d40-4681-b8d1-111111111111
DataNode1	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.11	7b7c1b11-1d40-4681-b8d1-111111111111
DataNode2	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.12	7b7c1b11-1d40-4681-b8d1-111111111111
DataNode3	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.13	7b7c1b11-1d40-4681-b8d1-111111111111
DataNode4	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.14	7b7c1b11-1d40-4681-b8d1-111111111111
NameNode	En ejecución	HadoopGroup	Oeste de Europa	Pase para Azure	13.92.210.15	7b7c1b11-1d40-4681-b8d1-111111111111

Las capturas de las interfaces web del HDFS, YARN, CheckPoint Node y JobHistory están disponibles en:

- [HDFS](#)
- [YARN](#)
- [JobHistory](#)
- [CheckpointNode](#)

Para ejecutar la aplicación, es necesario descargar el fichero de Mega, pero en vez de descargarlo en local y luego subirlo, es más fácil instalar curl (mediante un `sudo apt-get install curl -y`) y luego ejecutar el comando, como usuario normal (loksly en este caso):

```
wget -O mega.sh "http://pastebin.com/raw/JNZ0VUpi"
bash mega.sh 'https://mega.nz/#!T41hjDqI!7qJXHdkffuZrrbZIGYfaeiwMnI53PEnl0-Wo_qgTbt4' libros.tar.gz
tar xvf AERqiL7XHKcX3EXBwMTmXgTmMPZX
hdfs dfs -put libros .
wget "https://github.com/Loksly/tcdm_2016/blob/master/wordcount-0.0.1-SNAPSHOT.jar"
yarn jar wordcount-0.0.1-SNAPSHOT.jar libros salidawc
```

Una vez terminado el fichero de salida estará disponible en el directorio salidawc de HDF, con nombre `part-r-00000`, del que podemos ver aquí un pequeño extracto del mismo:

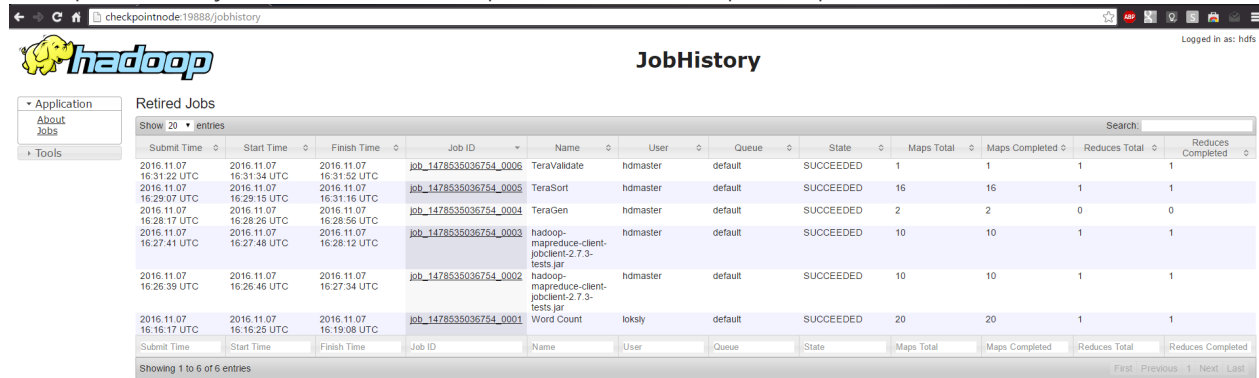
```
a 45639
aa 1038
aachen 965
aaliyah 2063
aardvark 2056
aardvarks 992
aaron 1005
ab 1019
aba 8
abaci 1032
aback 1049
abacus 2031
abacuses 989
abad 70
abade 2
abadejo 4
abades 11
abadesa 33
```

Ejecutar los benchmarks

Para ejecutar los benchmarks tan sólo es necesario, como usuario hdmaster:

```
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/run_benchmarks.bash" #descargar el script
bash run_benchmarks.bash
```

Como prueba de la ejecución de los benchmark podemos utilizar esta captura de pantalla:



Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016.11.07 16:31:22 UTC	2016.11.07 16:31:34 UTC	2016.11.07 16:31:52 UTC	job_1478535036754_0006	TeraValidate	hdmaster	default	SUCCEEDED	1	1	1	1
2016.11.07 15:29:07 UTC	2016.11.07 16:23:15 UTC	2016.11.07 16:31:16 UTC	job_1478535036754_0005	TeraSort	hdmaster	default	SUCCEEDED	16	16	1	1
2016.11.07 16:28:17 UTC	2016.11.07 16:28:26 UTC	2016.11.07 16:28:56 UTC	job_1478535036754_0004	TeraGen	hdmaster	default	SUCCEEDED	2	2	0	0
2016.11.07 16:27:41 UTC	2016.11.07 16:27:48 UTC	2016.11.07 16:28:12 UTC	job_1478535036754_0003	hadoop-mapreduce-client-jobclient-2.7.3-tests.jar	hdmaster	default	SUCCEEDED	10	10	1	1
2016.11.07 16:26:39 UTC	2016.11.07 16:26:46 UTC	2016.11.07 16:27:34 UTC	job_1478535036754_0002	hadoop-mapreduce-client-jobclient-2.7.3-tests.jar	hdmaster	default	SUCCEEDED	10	10	1	1
2016.11.07 16:16:17 UTC	2016.11.07 16:16:25 UTC	2016.11.07 16:19:08 UTC	job_1478535036754_0001	Word Count	loksly	default	SUCCEEDED	20	20	1	1

Añadir y retirar datanodos/jobtrackers

En este caso vamos a dar de alta un nuevo nodo, para ello tendremos que registrar otra interfaz de red con IP interna 10.0.0.10 y otra máquina virtual basada en la plantilla, modificando el archivo json correspondiente para reflejar la nueva interfaz, y poniendo como nombre DataNode5.

1. Paramos los demonios con el script namenode-stop.sh, y checkpointnode-stop.sh (en el checkpoint node)
2. En el namenode creamos cuatro ficheros: \$HADOOP_PREFIX/etc/hadoop/dfs.include, \$HADOOP_PREFIX/etc/hadoop/dfs.exclude, \$HADOOP_PREFIX/etc/hadoop/yarn.include y \$HADOOP_PREFIX/etc/hadoop/yarn.exclude (inicialmente vacíos).
3. En los ficheros dfs.include y yarn.include, ponemos los nombres de todos los DataNodes/NodeManagers que querramos que estén en el cluster.
4. En el fichero de configuración *hdfs-site.xml*, añadimos dos propiedades:

```
<property>
  <name>dfs.hosts</name>
  <value>/opt/yarn/hadoop/etc/hadoop/dfs.include</value>
  <final>true</final>
</property>
<property>
  <name>dfs.hosts.exclude</name>
  <value>/opt/yarn/hadoop/etc/hadoop/dfs.exclude</value>
  <final>true</final>
</property>
```

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

```
<property>
  <name>yarn.resourcemanager.nodes.include-path</name>
  <value>/opt/yarn/hadoop/etc/hadoop/yarn.include</value>
  <final>true</final>
</property>
<property>
  <name>yarn.resourcemanager.nodes.exclude-path</name>
  <value>/opt/yarn/hadoop/etc/hadoop/yarn.exclude</value>
  <final>true</final>
</property>
```

Añadimos en el fichero slaves la nueva ip, en una nueva línea:

```
echo "10.0.0.10" >> /opt/yarn/hadoop/etc/hadoop/slaves
```

Reiniciamos los demonios:

```
bash namenode-start.sh
```

Nos vamos al checknode y arrancamos también:

```
bash checkpointnode-start.sh
```

Ahora de vuelta a namenode:

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

namenode:50070/dfshealth.html#tab-datanode

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities -

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)	1	In Service	29.4 GB	24 KB	3.04 GB	26.36 GB	0	24 KB (0%)	0	2.7.3
datanode3.50010 (10.0.0.8:50010)	0	In Service	29.4 GB	2.22 GB	3.04 GB	24.15 GB	62	2.22 GB (7.53%)	0	2.7.3
datanode1.50010 (10.0.0.6:50010)	2	In Service	29.4 GB	2.71 GB	3.04 GB	23.66 GB	78	2.71 GB (9.21%)	0	2.7.3
datanode4.50010 (10.0.0.9:50010)	0	In Service	29.4 GB	2.2 GB	3.04 GB	24.16 GB	71	2.2 GB (7.49%)	0	2.7.3
datanode2.50010 (10.0.0.7:50010)	0	In Service	29.4 GB	2.42 GB	3.04 GB	23.95 GB	80	2.42 GB (8.22%)	0	2.7.3

Decommissioning

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

Hadoop, 2016.

Ahora vamos a proceder a quitar un nodo, por ejemplo el nodo datanode3, 10.0.0.8, para ello lo ponemos en los ficheros .exclude que acabamos de crear e informamos a los demonios del cambio mediante:

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

namenode:50070/dfshealth.html#tab-datanode

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities -

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	In Service	29.4 GB	24 KB	3.04 GB	26.36 GB	0	24 KB (0%)	0	2.7.3
datanode3.50010 (10.0.0.8:50010)	0	Decommission In Progress	29.4 GB	2.22 GB	3.04 GB	24.15 GB	62	2.22 GB (7.53%)	0	2.7.3
datanode1.50010 (10.0.0.6:50010)	0	In Service	29.4 GB	2.71 GB	3.04 GB	23.66 GB	78	2.71 GB (9.21%)	0	2.7.3
datanode4.50010 (10.0.0.9:50010)	0	In Service	29.4 GB	2.2 GB	3.04 GB	24.16 GB	71	2.2 GB (7.49%)	0	2.7.3
datanode2.50010 (10.0.0.7:50010)	0	In Service	29.4 GB	2.42 GB	3.04 GB	23.95 GB	80	2.42 GB (8.22%)	0	2.7.3

Decommissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
datanode3.50010 (10.0.0.8:50010)		0	0	0

Hadoop, 2016.

Ahora podríamos quitarlo del fichero slaves (aunque deberíamos apagar previamente sus demonios para que se apague correctamente).

Rack Awareness

Para llevar a cabo esta funcionalidad hay que crear un fichero `/opt/yarn/hadoop/etc/hadoop/topology.data` con este contenido:

```
10.0.0.6      /rack1
10.0.0.7      /rack1
10.0.0.9      /rack2
10.0.0.10     /rack2
datanode1     /rack1
datanode2     /rack1
datanode4     /rack2
datanode5     /rack2
```

Descargar el fichero `topology.script` y ponerlo como ejecutable

```
cd /opt/yarn/hadoop/etc/hadoop/
wget "https://github.com/Loksly/tcdm_2016/blob/master/scripts/topology.script"
chmod +x topology.script
```

y modificar el fichero `core.xml`, para añadirle esta propiedad:

```
<property>
  <name>net.topology.script.file.name</name>
  <value>/opt/yarn/hadoop/etc/hadoop/topology.script</value>
  <final>true</final>
</property>
```

Al hacerlo reiniciamos los demonios con el script `stop.sh` y el script `start.sh`, y podemos ver el resultado ejecutando:

```
hdfs dfsadmin -printTopology
```

```
hdmaster@NameNode:/opt/yarn$ bash start.sh
Starting namenode, logging to /var/log/hadoop/hdfs/hadoop-hdmaster-namenode-NameNode.out
Starting resourcemanager, logging to /var/log/hadoop/yarn/yarn-hdmaster-resourcemanager-NameNode.out
10.0.0.7: starting datanode, logging to /var/log/hadoop/hdfs/hadoop-hdmaster-datanode-DataNode2.out
10.0.0.9: starting datanode, logging to /var/log/hadoop/hdfs/hadoop-hdmaster-datanode-DataNode4.out
10.0.0.6: starting datanode, logging to /var/log/hadoop/hdfs/hadoop-hdmaster-datanode-DataNode1.out
10.0.0.10: starting datanode, logging to /var/log/hadoop/hdfs/hadoop-hdmaster-datanode-DataNode5.out
10.0.0.9: starting nodemanager, logging to /var/log/hadoop/yarn/yarn-hdmaster-nodemanager-DataNode4.out
10.0.0.7: starting nodemanager, logging to /var/log/hadoop/yarn/yarn-hdmaster-nodemanager-DataNode2.out
10.0.0.10: starting nodemanager, logging to /var/log/hadoop/yarn/yarn-hdmaster-nodemanager-DataNode5.out
10.0.0.6: starting nodemanager, logging to /var/log/hadoop/yarn/yarn-hdmaster-nodemanager-DataNode1.out
hdmaster@NameNode:/opt/yarn$ hdfs dfsadmin -printTopology
Rack: /rack1
  10.0.0.6:50010 (datanode1)
  10.0.0.7:50010 (datanode2)
Rack: /rack2
  10.0.0.10:50010 (datanode5)
  10.0.0.9:50010 (datanode4)
hdmaster@NameNode:/opt/yarn$
```

```
hdmaster@NameNode:~$ hdfs dfsadmin -report
Safe mode is ON
Configured Capacity: 126286282752 (117.61 GB)
Present Capacity: 113223499776 (105.45 GB)
DFS Remaining: 102980100096 (95.91 GB)
DFS Used: 10243399680 (9.54 GB)
DFS Used%: 9.05%
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: 3210465280 (2.99 GB)
Non DFS Used: 3265912832 (3.04 GB)
DFS Remaining: 25095192576 (23.37 GB)
DFS Used%: 10.17%
DFS Remaining%: 79.49%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 13 22:03:31 UTC 2016

Name: 10.0.0.7:50010 (datanode2)
Hostname: datanode2
Rack: /rack1
Decommission Status : Normal
Configured Capacity: 31571570688 (29.40 GB)
DFS Used: 3038068736 (2.83 GB)
Non DFS Used: 3267444736 (3.04 GB)
DFS Remaining: 25266057216 (23.53 GB)
DFS Used%: 9.62%
DFS Remaining%: 80.03%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 13 22:03:31 UTC 2016

Name: 10.0.0.10:50010 (datanode5)
Hostname: datanode5
Rack: /rack2
Decommission Status : Normal
Configured Capacity: 31571570688 (29.40 GB)
DFS Used: 1049755648 (1001.13 MB)
Non DFS Used: 3263705088 (3.04 GB)
DFS Remaining: 27258109952 (25.39 GB)
DFS Used%: 3.33%
DFS Remaining%: 86.34%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 13 22:03:30 UTC 2016

Name: 10.0.0.9:50010 (datanode4)
Hostname: datanode4
Rack: /rack2
Decommission Status : Normal
Configured Capacity: 31571570688 (29.40 GB)
DFS Used: 2945110016 (2.74 GB)
Non DFS Used: 3265720320 (3.04 GB)
DFS Remaining: 25360740352 (23.62 GB)
DFS Used%: 9.33%
DFS Remaining%: 80.33%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 13 22:03:30 UTC 2016

Parte opcional

Ejecución del WordCount en un cluster Azure HDInsight

Para ello es necesario hacer login en el [portal de Azure](#) y añadir un nuevo HDInsight y un nuevo almacenamiento.

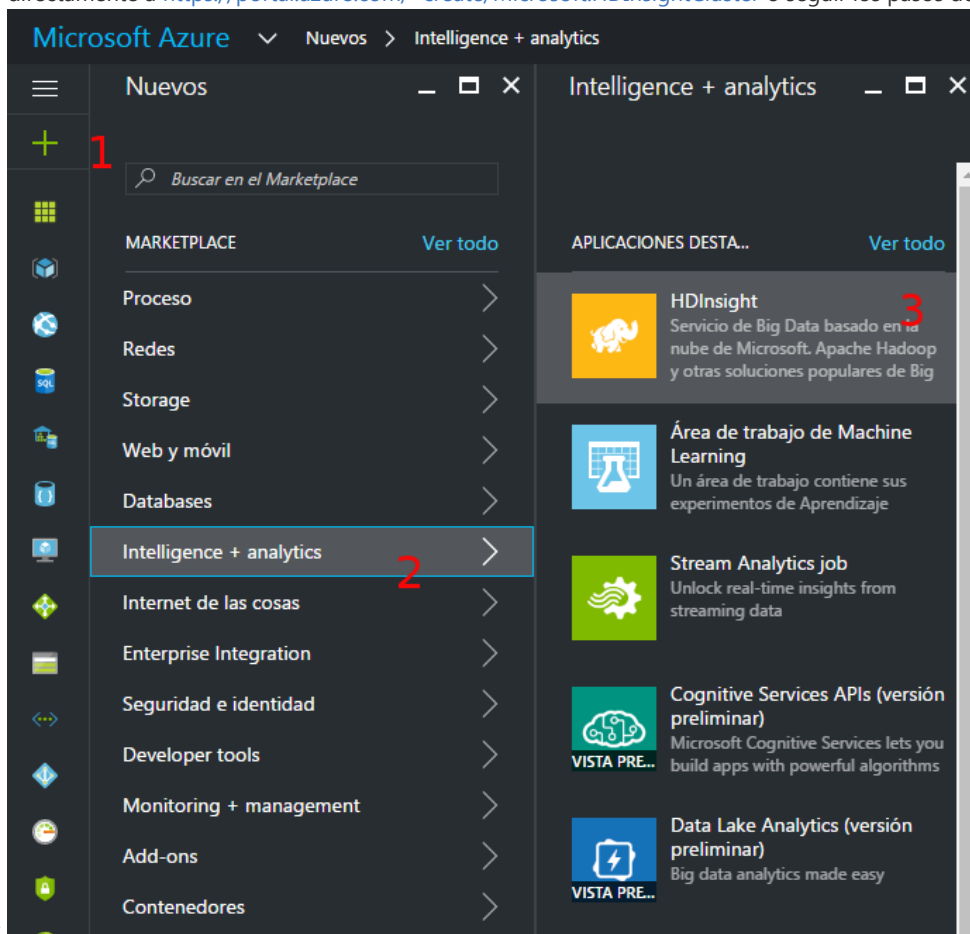
Crear un nuevo espacio de almacenamiento:

Versión por asistente:

Para ello es necesario acceder a <https://portal.azure.com/#create/Microsoft.StorageAccount-ARM> y cumplimentar con los siguientes datos:

- Nombre: lokslytest
- Tipo de cuenta: Uso general
- Rendimiento: Estándar
- Cifrado del lado del servidor: Deshabilitado
- Grupo de recursos: almacenamiento
- Ubicación oeste.

Se puede acceder directamente a <https://portal.azure.com/#create/Microsoft.HDInsightCluster> o seguir los pasos de la



siguiente imagen:

Después cumplimentar con los siguientes datos (son modificables/personalizables, pueden elegirse otros):

- Nombre del cluster: lokslytest
- Configuración del cluster:
 - Tipo de cluster: Hadoop
 - Sistema operativo: Linux
 - Versión: 2.7.1
 - Nivel de cluster: Estándar
- Credenciales:
 - Introducir contraseña de administrador (*recordar este dato para conectarnos al servidor ambari*)
 - Introducir nombre de usuario SSH (*recordar este dato para enviar el trabajo, en este ejemplo loksly*)
 - Introducir contraseña SSH (*recordar este dato para enviar el trabajo*)
- Origen de datos:
 - Método de selección: Desde todas las suscripciones

- Seleccionar existentes: lokslytest
- Ubicación (la que ponga por defecto, que será la misma que la del almacenamiento).
- Precios:
 - Número de nodos de trabajador: 2
 - Tamaño del nodo Trabajador: D3 v2 (2 nodos, 8 núcleos)
 - Tamaño del nodo Encabezado: D3 v2 (2 nodos, 8 núcleos)
- Grupo de recursos: insighttest

El proceso dura unos cuantos minutos.

Mientras tanto podemos enviar los ficheros de prueba al almacenamiento.

Para ello debemos seguir los siguientes pasos, ejecutados en la máquina local:

1. El usuario ssh debe almacenarse en la variable \$USERSSH,

```
USERSSH=loksly
```

1. obtener el listado de almacenamientos asociados a nuestro usuario en Azure.

```
azure storage account list
```

1. Seleccionamos la cuenta con la que queremos trabajar y almacenamos su valor en una variable:

```
storageaccount=CUENTA_DE_ENTRE_EL_LISTADO_PREVIO
```

1. Obtener la contraseña de acceso a ese almacenamiento

```
azure storage account keys list $storageaccount
```

1. Ahora almacenaremos la key en la variable con:

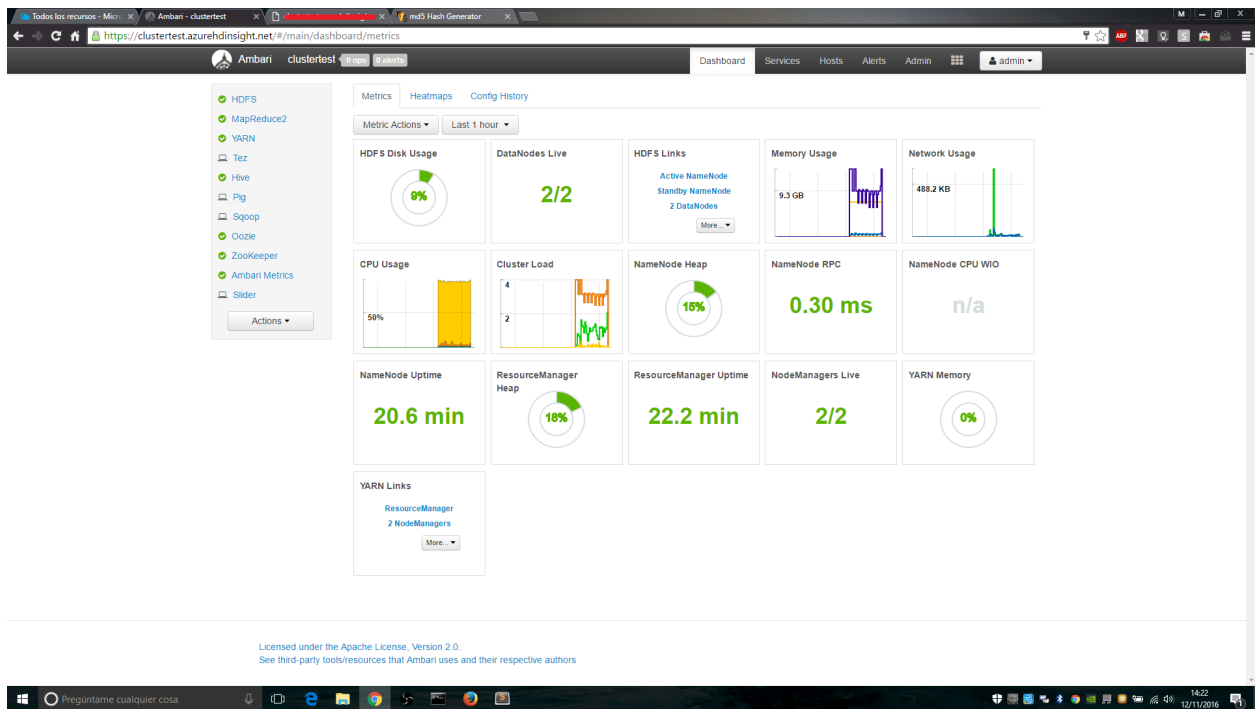
```
key=#####
```

1. Supuesto que estamos en el directorio previo al de los libros, procedemos a subir los libros con estos comandos, así:

```
files=`ls libros`
cd libros
for file in $files
do
    azure storage blob upload -a $storageaccount -k $key $file $insight user/$USERSSH/libros/$file
done
```

Una vez hemos enviado los ficheros al sistema de almacenamiento y hemos comprobado que ha terminado de crearse el cluster creado podemos proceder a la ejecución del programa.

Primero podemos ver el ambari que se ha creado accediendo a la dirección https que nos muestra en el panel de control, en mi caso <https://clustertest.azurehdinsight.net>, como usuario ponemos admin y como contraseña la del apartado anterior.



Ahora vamos a ejecutar el programa, para ello hacemos ssh al servidor, en mi caso loksly@#####-ssh.azurehdinsight.net y seguimos estos pasos:

```
loksly@hn0-cluste:~$ wget https://github.com/Loksly/tcdm_2016/archive/master.zip
--2016-11-12 13:36:16-- https://github.com/Loksly/tcdm_2016/archive/master.zip
Resolving github.com (github.com)... 192.30.253.112, 192.30.253.113
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/Loksly/tcdm_2016/zip/master [following]
--2016-11-12 13:36:16-- https://codeload.github.com/Loksly/tcdm_2016/zip/master
Resolving codeload.github.com (codeload.github.com)... 192.30.253.120, 192.30.253.121
Connecting to codeload.github.com (codeload.github.com)|192.30.253.120|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'master.zip'
```

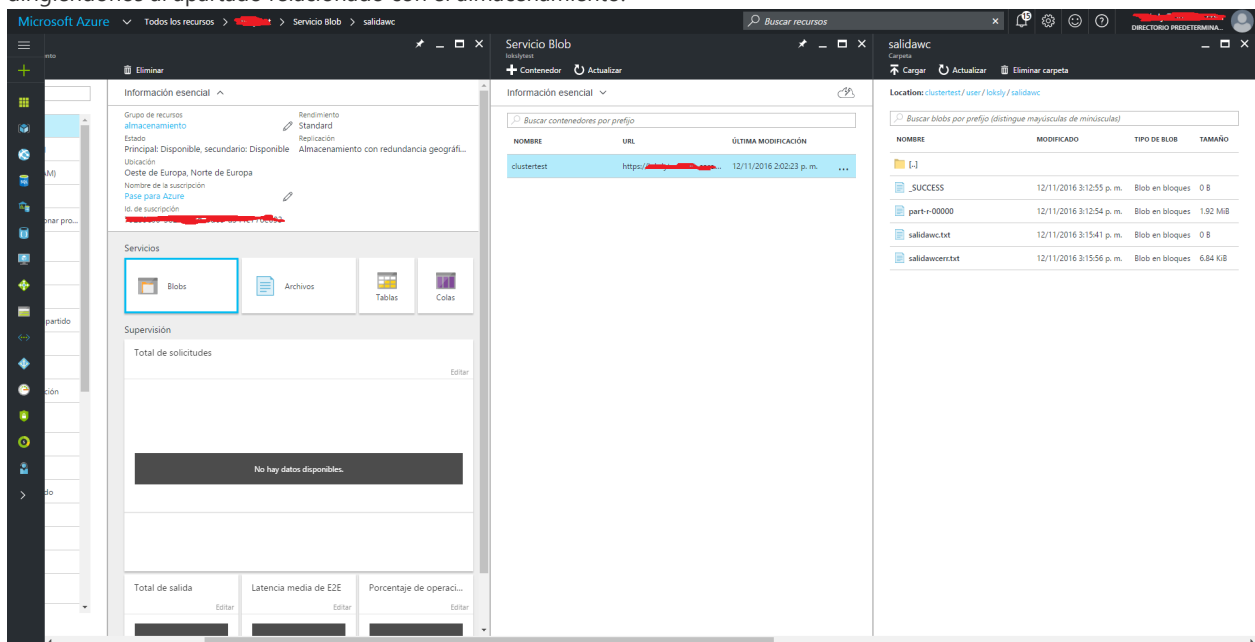
[<=>]

```
2016-11-12 13:36:17 (68.6 MB/s) - 'master.zip' saved [5769]
```

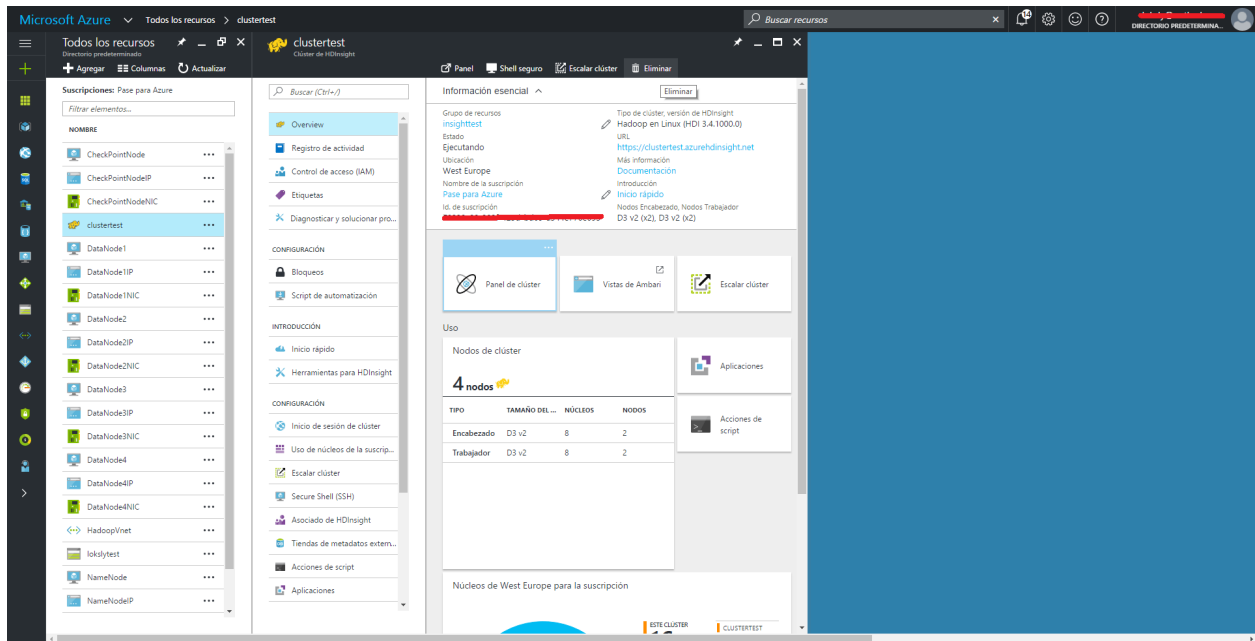
```
loksly@hn0-cluste:~$ unzip master.zip
Archive: master.zip
023b9d83d9ba0d1f02a96b70d624328a8731a94e
  creating: tcdm_2016-master/
 extracting: tcdm_2016-master/README.md
 inflating: tcdm_2016-master/wordcount-0.0.1-SNAPSHOT.jar
loksly@hn0-cluste:~$ cd tcdm_2016-master/
loksly@hn0-cluste:~/tcdm_2016-master$ hdfs dfs -ls libros
Found 16 items
-rwxrwxrwx 1 441804 2016-11-12 13:47 libros/pg14329.txt.gz
-rwxrwxrwx 1 264123 2016-11-12 13:47 libros/pg1619.txt.gz
-rwxrwxrwx 1 455129 2016-11-12 13:47 libros/pg16625.txt.gz
-rwxrwxrwx 1 939502 2016-11-12 13:48 libros/pg17013.txt.gz
-rwxrwxrwx 1 737367 2016-11-12 13:48 libros/pg17073.txt.gz
-rwxrwxrwx 1 219304 2016-11-12 13:48 libros/pg18005.txt.gz
-rwxrwxrwx 1 813698 2016-11-12 13:48 libros/pg2000.txt.gz
-rwxrwxrwx 1 328494 2016-11-12 13:48 libros/pg24536.txt.gz
-rwxrwxrwx 1 504188 2016-11-12 13:48 libros/pg25640.txt.gz
-rwxrwxrwx 1 38194 2016-11-12 13:48 libros/pg25807.txt.gz
-rwxrwxrwx 1 103986 2016-11-12 13:48 libros/pg32315.txt.gz
-rwxrwxrwx 1 125693 2016-11-12 13:48 libros/pg5201.txt.gz
-rwxrwxrwx 1 82099 2016-11-12 13:48 libros/pg7109.txt.gz
-rwxrwxrwx 1 99685 2016-11-12 13:48 libros/pg8870.txt.gz
-rwxrwxrwx 1 85187 2016-11-12 13:48 libros/pg9980.txt.gz
-rwxrwxrwx 1 330326458 2016-11-12 14:04 libros/random_words.txt.bz2
loksly@hn0-cluste:~/tcdm_2016-master$ hdfs dfs -ls libros
loksly@hn0-cluste:~/tcdm_2016-master$ hdfs dfs -put salidawc.txt salidawc
loksly@hn0-cluste:~/tcdm_2016-master$ hdfs dfs -put salidawcerr.txt salidawc
```

```
at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
16/11/12 13:44:20 INFO client.ConfigurationLoaderProxyProvider: Falling over to rm2
16/11/12 13:44:21 INFO mapreduce.JobSubmitter: number of splits:1
16/11/12 13:44:21 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1478955625320_0003
16/11/12 13:44:22 INFO mapreduce.Job: The url to track the job: http://hml-cluster-0b12savyb3yelhbkja5bmbvc.ax.internal.cloudapp.net:8088/proxy/application_1478955625320_0003/
16/11/12 13:44:22 INFO mapreduce.Job: Running job: job_1478955625320_0003
16/11/12 13:44:38 INFO mapreduce.Job: map 0% reduce 0%
16/11/12 13:44:55 INFO mapreduce.Job: map 100% reduce 0%
16/11/12 13:45:01 INFO mapreduce.Job: map 100% reduce 100%
16/11/12 13:45:03 INFO mapreduce.Job: Job job_1478955625320_0003 completed successfully
16/11/12 13:45:03 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=272094
FILE: Number of bytes written=36055
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
WASB: Number of bytes read=441954
WASB: Number of bytes written=202981
WASB: Number of read operations=0
WASB: Number of large read operations=0
WASB: Number of write operations=0
Job Counters
Launched map tasks=1
Launched reduce tasks=1
Task-local map tasks=1
Total time spent by all maps in occupied slots (ms)=15069
Total time spent by all reduces in occupied slots (ms)=3249
Total time spent by all map tasks (ms)=15069
Total time spent by all reduce tasks (ms)=3249
Total vcore-seconds taken by all map tasks=15069
Total vcore-seconds taken by all reduce tasks=3249
Total megabyte-seconds taken by all map tasks=2345964
Total megabyte-seconds taken by all reduce tasks=4990464
Map-Reduce Framework
Map input records=19069
Map output records=181486
Map output bytes=1803549
Map output materialized bytes=272094
Input split bytes=150
Combine input records=181486
Combine output records=17854
Reduce input groups=17854
Reduce shuffle bytes=272094
Reduce input records=17854
Reduce output records=17854
Spilled Records=35708
Shuffled Maps=1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=33
CPU time spent (ms)=4000
Physical memory (bytes) snapshot=1090195456
Virtual memory (bytes) snapshot=230226880
Total committed heap usage (bytes)=2119172096
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=441804
File Output Format Counters
Bytes Written=202981
tokslyhml-cluster-0b12savyb3yelhbkja5bmbvc> yarn jar wordcount-0.0.1-SNAPSHOT-jar-libs solidawc > solidawc.txt 2>solidawcerr.txt
```

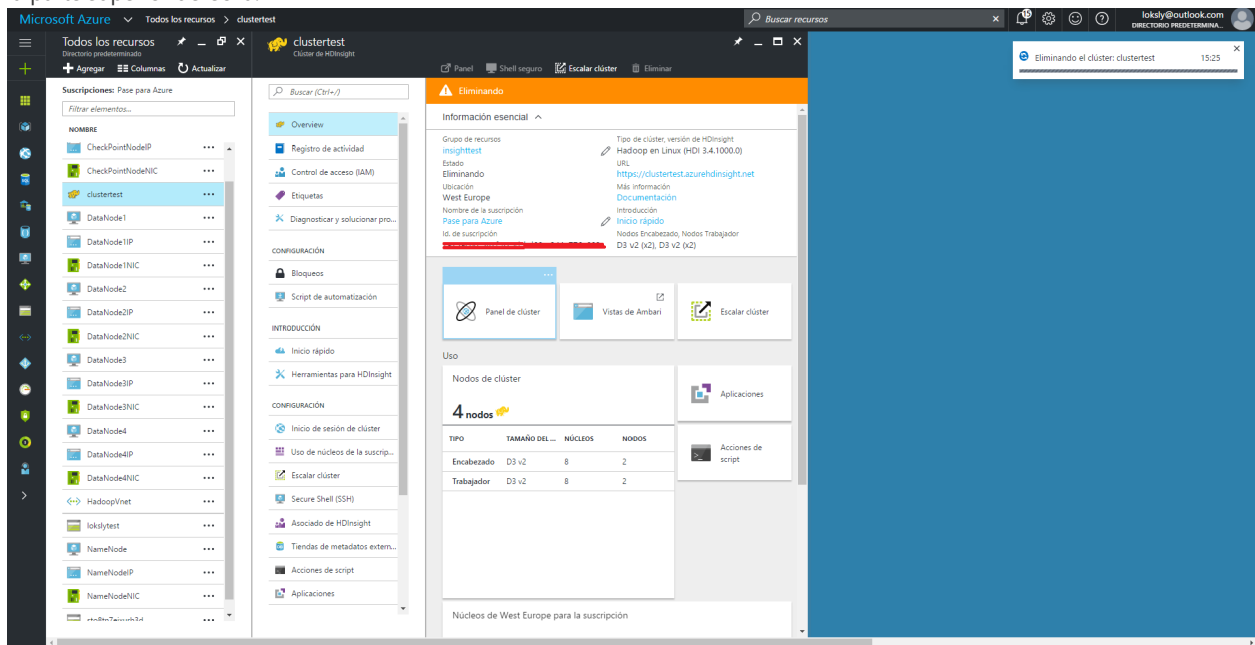
Desde el panel de control de azure podemos acceder al resultado de la ejecución, tal y como muestra la imagen, dirigiéndonos al apartado relacionado con el almacenamiento.



Una vez ha concluido la ejecución podemos proceder a la eliminación del cluster de Insight, de esta forma no consumirá recursos y el resultado de su procesamiento ha quedado almacenado fuera del mismo. Para ello podemos irnos al apartado del cluster (en nuestro caso llamado clusterest), y pulsar sobre el botón de *eliminar* situado en la parte superior derecha.



Este proceso dura un rato, pero podemos ver que sigue procediendo a su eliminación por el mensaje de progreso situado en la parte superior derecha.



Copyright information

Copyright disclaimer: There are parts of this code that belongs to it's original author.

http://persoal.citius.usc.es/tf.pena/TCDM/P1/instalacin_manual_de_un_cluster.html