Práctica 1

Objetivo de la práctica y entrega de documentación

En esta práctica utilizaremos contenedores Docker para desplegar un cluster Hadoop.

Para poder usar docker y completar la práctica, vuestro PC debe cumplir ciertos requisitos:

1. 8 GB RAM (mínimo recomendado)
2. Preferiblemente, sistema operativo Linux
3. Si usáis una MV con Linux, la MV debería tener 2 cores y 6-8 GB de RAM y disponer de aceleración por hardware
4. En caso de tener Windows, versión 10 o superior con WSL2 ó Hyper-V activado

Instalación de Docker:

- Linux: mira en https://docs.docker.com/engine/install/ cómo instalar Docker-engine para tu distribución.
- Windows: descarga el instalador desde https://docs.docker.com/desktop/windows/install/

Actividad guiada

1. Instalar manualmente un cluster Hadoop con contenedores Docker
2. Ejecutar una aplicación Java simple de demostración

Tareas a realizar

1. Añadir un nodo de Backup y un TimelineServer
2. Añadir y retirar nodos Datanodes/Nodemanagers
3. Hacer que el cluster sea *rack-aware*

Entrega obligatoria

- Generar una memoria con las capturas de pantalla especificadas, en el que se demuestre que se han realizado y entendido las tareas propuestas. Este documento tiene que incluir capturas de pantalla en las que, como mínimo, se muestre lo siguiente:
    a. Capturas de pantalla que demuestren el funcionamiento del nodo de Backup y del TimeLineServer.
    b. Capturas de pantalla en las que se vea un nuevo nodo añadido y uno retirado (*decomisionado*).
    c. Captura de pantalla en la que se vea los nodos separados por rack.
- Más detalles se muestran al final de la descripción de cada una de las tareas propuestas.

IMPORTANTE: El documento no puede consistir en una secuencia de imágenes. Es necesario describir cómo se han ejecutado cada una de las tareas propuestas.

Practica: Instalación y despliegue de un cluster Hadoop 3

Objetivo

Instalar un cluster Hadoop usando contenedores Docker:

- Un cluster de 5 máquinas: 1 NameNode/ResourceManager y 4 DataNodes/NodeManagers

Apartado 1: Creación de imágenes Docker para los diferentes servicios

Vamos a crear contenedores específicos para los servicios de NameNode/ResourceManager y DataNode/NodeManager. Antes de nada, crea una red en la que iniciaremos los contenedores Docker:

```
docker network create hadoop-cluster
```

1.1 Servicio NameNode/ResourceManager

Inicia un contenedor para esos servicios ejecutando:

```
docker container run -ti --name namenode --network=hadoop-cluster --
hostname namenode --net-alias resourcemanager --expose=8000-10000 -p
9870:9870 -p 8088:8088 dsevilla/hadoop-base /bin/bash
```

El contenedor se identifica con ambos nombres `namenode` y `resourcemanager` y es accesible desde el host a través de los puertos 9870 (NameNode) y 8088 (ResourceManager).

1.1.1 Creación de directorios para los datos del **NameNode**

Debemos especificar el directorio o directorios en el que el NameNode guardará la metainformación de HDFS. En un sistema real se deben usar por lo menos dos directorios: uno en el disco local del NameNode (preferible disponer de varios discos configurados en RAID) y otro remoto (por ejemplo, montado por NFS).

En nuestro caso, crearemos un único directorio y haremos que sea propiedad del usuario `hdadmin`, que será el que ejecute los demonios del NameNode y del ResourceManager, ejecutando en el docker, como root, los comandos:

```
mkdir -p /var/data/hdfs/namenode
chown hdadmin:hadoop /var/data/hdfs/namenode
```

1.1.2 Configuración los demonios NameNode/ResourceManager

Todos los demonios de Hadoop se configuran, principalmente, mediante cuatro ficheros, localizados en `$HADOOP_HOME/etc/hadoop/`, en los que se pueden indicar un gran número de propiedades (véase http://hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-common/ClusterSetup.html para más información):

- `core-site.xml`: configuración principal, valores por defecto en http://hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-common/core-default.xml
- `hdfs-site.xml`: configuración del HDFS, valores por defecto en http://hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml
- `yarn-site.xml`: configuración del YARN, valores por defecto en http://hadoop.apache.org/docs/stable3/hadoop-yarn/hadoop-yarn-common/yarn-default.xml
- `mapred-site.xml`: configuración del MapReduce, valores por defecto en http://hadoop.apache.org/docs/stable3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml

Para nuestro NameNode/ResourceManager, como usuario hdadmin (`su - hdadmin`), cambia los siguientes ficheros en `$HADOOP_HOME/etc/hadoop/`:

- ***core-site.xml***: configuración general de Hadoop

```
<configuration>

  <property>
    <!-- Nombre del filesystem por defecto -->
    <!-- Como queremos usar HDFS tenemos que indicarlo con hdfs:// y
el servidor y puerto en el que corre el NameNode -->
    <name>fs.defaultFS</name>
    <value>hdfs://namenode:9000/</value>
    <final>true</final>
  </property>

  <property>
    <!-- Directorio para almacenamiento temporal (debe tener
suficiente espacio) -->
    <name>hadoop.tmp.dir</name>
    <value>/var/tmp/hadoop-${user.name}</value>
    <final>true</final>
  </property>

</configuration>
```

- ***hdfs-site.xml***: configuración del demonio NameNode (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. -->
    <name>dfs.namenode.name.dir</name>
    <value>file:///var/data/hdfs/namenode</value>
    <final>true</final>
  </property>

  <property>
    <!-- Dirección y puerto del interfaz web del namenode -->
    <name>dfs.namenode.http-address</name>
    <value>namenode:9870</value>
    <final>true</final>
  </property>

</configuration>
```

- ***yarn-site.xml***: configuración del demonio ResourceManager (YARN)

```
<configuration>

  <property>
    <!-- Nombre del equipo que ejecuta el demonio ResourceManager -->
    <name>yarn.resourcemanager.hostname</name>
    <value>resourcemanager</value>
    <final>true</final>
  </property>

  <property>
    <!-- Agrega los logs de todos los contenedores y aplicaciones -->
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
    <final>true</final>
  </property>

  <property>
    <!-- Directorio donde se agregarán los logs en HDFS -->
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/tmp/logs</value>
    <final>true</final>
  </property>
```

```
  <property>
    <!-- Número máximo de vcores que un ApplicationMaster puede pedir
al RM (por defecto: 4) -->
    <!-- Peticiones mayores lanzan una InvalidResourceRequestException
-->
    <name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>1</value>
    <final>true</final>
  </property>

  <property>
    <!-- Memoria minima (MB) que un ApplicationMaster puede solicitar
al RM (por defecto: 1024) -->
    <!-- La memoria asignada a un contenedor será múltiplo de esta
cantidad -->
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>128</value>
    <final>true</final>
  </property>

  <property>
    <!-- Memoria maxima (MB) que un ApplicationMaster puede solicitar
al RM (por defecto: 8192 MB) -->
    <!-- Peticiones mayores lanzan una InvalidResourceRequestException
-->
    <!-- Puedes aumentar o reducir este valor en funcion de la memoria
de la que dispongas -->
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>4096</value>
    <final>true</final>
  </property>

</configuration>
```

- ***mapred-site.xml***: configuración del framework MapReduce de Hadoop

```
<configuration>

  <property>
    <!-- Framework que realiza el MapReduce -->
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
    <final>true</final>
  </property>

  <!-- Configuracion del ApplicationMaster (AM) del MR -->

  <property>
    <!-- Localizacion del software MR para el AM -->
```

```
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>

  <property>
    <!-- Numero maximo de cores para el ApplicationMaster (por
defecto: 1) -->
    <name>yarn.app.mapreduce.am.resource.cpu-vcores</name>
    <value>1</value>
    <final>true</final>
  </property>

  <property>
    <!-- Memoria que necesita el ApplicationMaster del MR (por
defecto: 1536) -->
    <name>yarn.app.mapreduce.am.resource.mb</name>
    <value>1536</value>
    <final>true</final>
  </property>


  <!-- Configuracion de los maps y reduces del MR -->

  <property>
    <!-- Ratio del tamaño del heap al tamaño del contenedor para las
JVM (por defecto: 0.8)-->
    <name>mapreduce.job.heap.memory-mb.ratio</name>
    <value>0.8</value>
    <final>true</final>
  </property>

  <!-- Maps -->
  <property>
    <!-- Localizacion del software MR para los maps -->
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>

  <property>
    <!-- Numero maximo de cores para cada tarea map (por defecto: 1)
-->
    <name>mapreduce.map.cpu.vcores</name>
    <value>1</value>
    <final>true</final>
  </property>

  <property>
    <!-- Opciones para las JVM de los maps -->
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx3072M</value> <!-- "-Xmx" define el tamaño máximo de la
```

```
pila de Java -->
    <final>true</final>
  </property>

  <property>
    <!-- Memoria maxima (MB) por map (si -1 se optiene a partir de
mapreduce.map.java.opts y mapreduce.job.heap.memory-mb.ratio) -->
    <name>mapreduce.map.memory.mb</name>
    <value>-1</value>
    <final>true</final>
  </property>

  <!-- Reduces -->
  <property>
    <!-- Localizacion del software MR para los reducers -->
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
  </property>

  <property>
    <!-- Numero maximo de cores para cada tarea reduce (por defecto:
1) -->
    <name>mapreduce.reduce.cpu.vcores</name>
    <value>1</value>
    <final>true</final>
  </property>

  <property>
    <!-- Opciones para las JVM de los reduces -->
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx3072M</value> <!-- Xmx define el tamaño máximo de la
pila de Java -->
    <final>true</final>
  </property>

  <property>
    <!-- Memoria maxima (MB) por reduce (si -1 se optiene a partir de
mapreduce.map.java.opts y mapreduce.job.heap.memory-mb.ratio) -->
    <name>mapreduce.reduce.memory.mb</name>
    <value>-1</value>
    <final>true</final>
  </property>

</configuration>
```

1.1.3 Inicializa el HDFS

Es necesario inicializar el sistema HDFS ejecutando, como usuario hdadmin:

```
hdfs namenode -format
```

Al finalizar el proceso de inicialización, si todo fue bien debería aparecer, entre otros mensajes, lo siguiente:

```
Storage directory /var/data/hdfs/namenode has been successfully
formatted.
```

Revisa ese directorio para comprobar qué se ha creado. Comprueba también que se ha creado un directorio para los logs en `$HADOOP_HOME/logs`.

1.1.4 Inicio de los demonios

1. Inicia el demonio NodeManager ejecutando (como usuario `hdadmin`):

```
hdfs --daemon start namenode
```

Mira los ficheros creados en el directorio de logs (`$HADOOP_HOME/logs`) para comprobar que todo ha ido bien y que no aparecen errores. Ejecuta el comando `jps` para ver que la JVM está corriendo.

1. Inicia el demonio ResourceManager ejecutando (como usuario `hdadmin`):

```
yarn --daemon start resourcemanager
```

Mira de nuevo los ficheros creados en el directorio de logs (`$HADOOP_HOME/logs`) para comprobar que todo ha ido bien y que no aparecen errores. Ejecuta el comando `jps` para ver que la JVM está corriendo.

1.1.5 Acceso a los interfaces web de los demonios

Los diferentes servicios de Hadoop ofrecen una interfaz web de control. Abre un navegador y comprueba los siguientes enlaces:

- http://localhost:9870 interfaz web del HDFS.
- http://localhost:8088 interfaz web de YARN.

1.1.6 Parar los demonios

Para detener los demonios, ejecutamos las instrucciones anteriores en cambiando start por stop:

```
yarn --daemon stop resourcemanager
hdfs --daemon stop namenode
```

1.1.7 Automatización del inicio

Para que podamos parar y reiniciar los contenedores y que los demonios sigan funcionando, vamos a salvar el contenedor como imagen, pero antes vamos a preparar un script para iniciar los demonios de forma automática cuando iniciemos el docker. Para ello, crea, **como root**, el fichero `/inicio.sh` en el directorio raíz, con el siguiente contenido:

```
#! /bin/sh
export JAVA_HOME=/usr/lib/jvm/default-java
```

```
export HADOOP_HOME=/opt/bd/hadoop

# Inicio el NameNode y el ResourceManager
su - hdadmin -c "$HADOOP_HOME/bin/hdfs --daemon start namenode"
su - hdadmin -c "$HADOOP_HOME/bin/yarn --daemon start resourcemanager"

# Lazo para mantener activo el contenedor
while true; do sleep 10000; done
```

Además, dale permisos de ejecución al fichero:

```
chmod +x /inicio.sh
```

1.1.8 Creación de la imagen Docker para el DataNode/NodeManager

Sal del contenedor y crea una imagen a partir del mismo, haciendo:

```
docker container commit namenode namenode-image
```

Comprueba que la imagen se han creado (ejecuta `docker images`) y si todo está bien, borra el contenedor:

```
docker container rm namenode
```

1.2 Servicio DataNode/NodeManager

El proceso es muy similar al del caso anterior. Empieza iniciando un contenedor para esos servicios ejecutando:

```
docker container run -ti --name datanode --network=hadoop-cluster --
hostname datanode --expose=8000-10000 --expose=50000-50200
dsevilla/hadoop-base /bin/bash
```

El contenedor se identifica en la red con el nombre datanode y no publica ningún puerto.

1.2.1 Creación de directorios para los datos de los DataNodes

Debemos especificar el directorio o directorios en los que los DataNodes guardarán los bloques HDFS. Lo ideal es diponer de un nodo con varios discos y crear un directorio en cada uno de los discos locales para mejorar el rendimiento en los accesos. No es conveniente usar RAID, ya que HDFS garantiza la redundancia.

En nuestro caso, crearemos un único directorio, ejecutando en el docker, como root, los comandos:

```
mkdir -p /var/data/hdfs/datanode
chown hdadmin:hadoop /var/data/hdfs/datanode
```

1.2.2 Configuración los demonios DataNode/NodeManager

Para nuestro DataNode/NodeManager, como usuario hdadmin (`su - hdadmin`), cambia los siguientes ficheros en `$HADOOP_HOME/etc/hadoop/`:

- ***core-site.xml***: configuración general de Hadoop, igual que en el NameNode aquí
- ***hdfs-site.xml***: configuración del demonio DataNode (HDFS)

```
<configuration>

    <property>
      <!-- Lista (separada por comas) de directorios donde los
DataNodes guardan los bloques HDFS -->
      <name>dfs.datanode.data.dir</name>
      <value>file:///var/data/hdfs/datanode</value>
      <final>true</final>
    </property>

</configuration>
```

- ***yarn-site.xml***: configuración del demonio NodeManager (YARN)

```
<configuration>

  <property>
    <!-- Nombre del equipo que ejecuta el demonio ResourceManager -->
    <name>yarn.resourcemanager.hostname</name>
    <value>resourcemanager</value>
    <final>true</final>
  </property>

  <property>
    <!-- Agrega los logs de todos los contenedores y aplicaciones -->
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
    <final>true</final>
  </property>

  <property>
    <!-- Directorio donde se agregarán los logs en HDFS -->
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/tmp/logs</value>
    <final>true</final>
  </property>

  <property>
    <!-- Activa la auto-deteccion de las capacidades de los nodos
(memoria y CPU) -->
    <name>yarn.nodemanager.resource.detect-hardware-
capabilities</name>
```

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

  <property>
    <!-- Número de vcores  que pueden alocarse para contenedores -->
    <!-- Si vale -1, se detecta automáticamente (si la auto-deteccion
está activada) -->
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>-1</value>
    <final>true</final>
  </property>

  <property>
    <!-- MB de RAM física que puede ser reservada para los containers
(por defecto: 8192) -->
    <!-- debe ser menor que la RAM fisica, para que funcionen otros
servicios -->
    <!-- Si vale -1, se detecta automáticamente (si la auto-deteccion
está activada) -->
    <!-- Se puede cambiar por un valor fijo, p.e. 3072 (3 GB) -->
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>4096</value>
    <final>true</final>
  </property>

  <property>
    <!-- Deshabilita el chequeo de los límites de uso de la memoria
virtual -->
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</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>

</configuration>
```

- ***mapred-site.xml***: configuración del framework MapReduce de Hadoop, igual que en el Resourcemanager.

1.2.3 Inicio de los demonios

1. Inicia el demonio NodeManager ejecutando (como usuario `hdadmin`):

```
hdfs --daemon start datanode
```

Mira los ficheros creados en el directorio de logs (`$HADOOP_HOME/logs`) para comprobar que no hay errores (debería indicar que no puede conectarse al namenode). Ejecuta el comando `jps` para ver que la JVM está corriendo.

1. Inicia el demonio NodeManager ejecutando (como usuario `hdadmin`):

```
yarn --daemon start nodemanager
```

Mira de nuevo los ficheros creados en el directorio de logs (`$HADOOP_HOME/logs`) para comprobar que no hay errores. Ejecuta el comando `jps` para ver que la JVM está corriendo.

1.2.4 Parar los demonios

Para detener los demonios, ejecutamos las instrucciones anteriores en cambiando start por stop:

```
yarn --daemon stop nodemanager
hdfs --daemon stop datanode
```

1.2.5 Automatización del inicio

Para que los demonios se inicien al lanzar el Docker, de forma similar a como lo hicimos con el NameNode/ResourceManager crea, **como root**, el fichero `/inicio.sh` en el directorio raíz, con el siguiente contenido:

```
#! /bin/sh
export JAVA_HOME=/usr/lib/jvm/default-java
export HADOOP_HOME=/opt/bd/hadoop

# Inicio el DataNode y el NodeManager
su - hdadmin -c "$HADOOP_HOME/bin/hdfs --daemon start datanode"
su - hdadmin -c "$HADOOP_HOME/bin/yarn --daemon start nodemanager"

# Lazo para mantener activo el contenedor
while true; do sleep 10000; done
```

Recuerda darle permisos de ejecución al fichero:

```
chmod +x /inicio.sh
```

1.2.6 Creación de la imagen Docker para el DataNode/NodeManager

Sal del contenedor (`exit`) y crea una imagen a partir del mismo, haciendo:

```
docker container commit datanode datanode-image
```

Comprueba que la imagen se ha creado (ejecuta `docker images`) y si todo está bien, borra el contenedor:

```
docker container rm datanode
```

Apartado 2: Inicio del cluster Hadoop con contenedores Docker

Vamos ya a iniciar nuestro cluster, formado por 1 NameNode/ResourceManager y 4 DataNodes/NodeManagers (en caso de disponer de poca RAM libre, puedes iniciar solo 3 de estos).

1. Inicia el NameNode/ResourceManager (puedes ajustar las especificaciones de CPU y memoria en función del número de cores y RAM del equipo)

```
docker container run -d --name namenode --network=hadoop-cluster --
hostname namenode --net-alias resourcemanager --cpus=1 --memory=3072m
--expose=8000-10000 -p 9870:9870 -p 8088:8088 -p 8888:8888 -p
4040:4040 namenode-image /inicio.sh
```

1. Inicia los 4 DataNodes/NodeManagers (puedes ajustar las especificaciones de cpu y memoria en función del número de cores y RAM del equipo)

```
for i in {1..4}; do docker container run -d --name datanode$i --
network=hadoop-cluster --hostname datanode$i --cpus=1 --memory=3072m
--expose=8000-10000 --expose=50000-50200 datanode-image /inicio.sh;
done
```

1. Comprueba con `docker container ps` que están ejecutándose los 5 contenedores.
2. Conéctate al NameNode ejecutando:

```
docker container exec -ti namenode /bin/bash
```

y una vez dentro de contenedor, como usuario hdadmin prueba que todo va bien ejecutando:

```
hdfs dfsadmin -report
yarn node -list
```

1. Adicionalmente, comprueba en el interfaz web del NameNode y del NodeManager para ver si es correcto

A partir de ahora, para detener los contenedores, basta hacer `docker container stop` y el nombre de cada uno y para volverlos a iniciar, simplemente `docker container start` y el nombre:

```
docker container stop namenode datanode{1..4}
docker container start namenode datanode{1..4}
```

Por seguridad, para los contenedores antes de apagar el PC o la máquina virtual

Apartado 3: Creación de directorios en HDFS y copia de datos

3.1 Creación de los directorios de los usuarios en HDFS

Dentro del Namenode, como usuario `hdadmin`, crea directorios en HDFS (dentro de `/user`) para el usuario `hdadmin` y para el usuario local `luser` (que es el que va a ejecutar las tareas MapReduce):

```
hdfs dfs -mkdir -p /user/hdadmin
hdfs dfs -mkdir -p /user/luser
hdfs dfs -chown luser /user/luser
hdfs dfs -ls /user
```

Debemos crear también un directorio `/tmp` y darle los permisos adecuados

```
hdfs dfs -mkdir -p /tmp/hadoop-yarn/staging
hdfs dfs -mkdir -p /tmp/logs
hdfs dfs -chmod -R 1777 /tmp
hdfs dfs -chmod -R 1777 /tmp/logs
```

3.2 Copia de ficheros de usuario

1. En el Namenode, conviertete en usuario no privilegiado (haciendo, como root, `su - luser`)

2. Comprueba que puedes acceder al HDFS ejecutando:

```
hdfs dfs -ls
```

(Este comando no debería dar ninguna salida, ya que no tenemos nada en ese directorio)

1. Descarga de https://umubox.um.es/index.php/s/TDBPLG59W3BrOaD/download unos ficheros de ejemplo que usaremos en esta practica y cópialos al contenedor del NameNode usando:

```
docker container cp ~/Downloads/libros.tar namenode:/tmp
```

1. En el NameNode "destarea" el fichero y copia los datos a HDFS

```
cd /tmp; tar xvf libros.tar
hdfs dfs -put libros .
hdfs dfs -ls libros
```

Una vez copiados borra el directorio libros del disco local (`rm -rf /tmp/libros`) y, como root, el fichero `libros.tar`

1. Mira en el interfaz web del HDFS donde se encuentran los bloques correspondientes al fichero **random_words.txt.bz2**. Comprueba que cada bloque tiene 3 réplicas.
   - Abre el navegador y conéctate al interfaz web del NameNode
   - Ve al menu "Utilities" -> "Browse the filesystem"

Apartado 4: Prueba de aplicaciones MapReduce simples

4.1 Aplicación MapReduce para el cálculo de Pi

En el NameNode, **como usuario `hdadmin`**, ejecuta el siguiente ejemplo de aplicación MapReduce:

```
export MAPRED_EXAMPLES=$HADOOP_HOME/share/hadoop/mapreduce
yarn jar $MAPRED_EXAMPLES/hadoop-mapreduce-examples-*.jar pi 16 1000
```

Comprueba en el interfaz web de Yarn la ejecución de esta tarea.

5.2 Aplicación WordCount

Descarga el código Java del wordcount de aquí.

1. Cópialo en el NameNode

```
docker cp ~/Downloads/wordcount.tgz namenode:/home/luser
```

1. En el NameNode, como usuario `luser` descomprímelo y compílalo usando maven

```
cd; tar xvzf wordcount.tgz
cd wordcount
mvn package
```

1. Ejecútalo con el comando Yarn:

```
yarn jar target/wordcount*.jar libros/p* wordcount-out
```

Comprueba en el interfaz web de Yarn la ejecución de esta tarea.

1. Trae los ficheros de salida del HDFS al disco local del NameNode:

```
hdfs dfs -get wordcount-out
```

1. Comprueba los ficheros de salida

Tareas a realizar

Tarea 1: Añadir al cluster un servidor de Backup y un TimeLineServer

1. Servidor de Backup

El servidor de backup realiza una tarea doble:

1. Mantiene una copia de seguridad permanentemente actualizada de los metadatos del NameNode
2. Realiza tareas de Checkpoint sobre estos metadatos

Más información sobre este servicio en https://hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Backup_Node

Importante: Antes de iniciar el servicio de backup, inicia el clúster, ve al NameNode y obtén una captura de pantalla en la que se vean los ficheros del directorio de metadatos del NameNode dentro de current (`/var/data/hdfs/namenode/current`) e inclúyela en la memoria.

Para añadir el servidor de backup, tenéis que seguir los siguientes pasos (con el cluster funcionando):

1. Inicia un nuevo Docker a partir de la imagen hadoop-base de la siguiente forma:

```
docker container run -ti --name backupnode --network=hadoop-cluster --hostname backupnode --cpus=1 --memory=3072m --expose=50100 -p 50105:50105 dsevilla/hadoop-base /bin/bash
```

1. Crea un directorio donde se guardarán los backups. Haz que el propietario de ese directorio sea hdadmin y crea dentro del mismo la carpeta `dfs/name`

2. Como usuario hdadmin, añade al fichero `core-site.xml` las siguentes propiedades

a) `fs.defaultFS`: Nombre del filesystem por defecto. Dale el valor `hdfs://namenode:9000/`.

b) `hadoop.tmp.dir`: Indica el directorio donde se guardarán las copias de seguridad. Dale el valor del directorio que has creado (sin incluir `dfs/name`).

```
<configuration>

  <property>
    <!-- Nombre del filesystem por defecto -->
    <!-- Como queremos usar HDFS tenemos que indicarlo con hdfs:// y el servidor y puerto en el que corre el NameNode -->
    <name>fs.defaultFS</name>
    <value>hdfs://namenode:9000/</value>
    <final>true</final>
  </property>

  <property>
    <!-- Directorio para almacenamiento temporal (debe tener suficiente espacio) -->
    <name>hadoop.tmp.dir</name>
    <value>/var/data/hdfs/backupnode</value>
    <final>true</final>
  </property>

</configuration>
```

1. Como usuario `hdadmin`, añade al fichero `hdfs-site.xml` las siguentes propiedades

a) `dfs.namenode.backup.address`: Dirección y puerto del nodo de backup. Dale el valor `backupnode:50100`.

b) `dfs.namenode.backup.http-address`: Dirección y puerto del servicio web del nodo de backup. Dale el valor `backupnode:50105`.

```
<configuration>

 <property>
    <name>dfs.namenode.backup.address</name>
    <value>backupnode:50100</value>
    <final>true</final>
  </property>

  <property>
    <name>dfs.namenode.backup.http-address</name>
    <value>backupnode:50105</value>
    <final>true</final>
  </property>

</configuration>
```

1. Inicia el servidor de backup ejecutando:

```
hdfs namenode -backup
```

1. Analiza el directorio de backup para ver lo que se ha creado. Compáralo con el directorio con los metadatos del NameNode
2. Mira en los mensajes del servicio de backup información que indique que se ha realizado un checkpoint

Nota: Una vez obtenidos los datos para el ejercicio, puedes parar el servicio de backup. Si quieres poder reiniciarlo de forma facil, sal del contenedor, guarda el Docker como una imagen e inícialo haciendo:

```
docker container run -d --name backupnode --network=hadoop-cluster --
hostname backupnode --cpus=1 --memory=3072m --expose=50100 -p
50105:50105 backupnode-image su - hdadmin -c
"JAVA_HOME=/usr/lib/jvm/default-java /opt/bd/hadoop/bin/hdfs namenode
-backup"
```

Y para comprobar que se está ejecutando correctamente el servicio de backup, haz:

```
docker container logs backupnode
```

Información para la documentación en la memoria

Incluir en la memoria:

1. Captura de pantalla en la que se vean los mensajes que genera el servicio de backup, destacando aquellos en los que se vea como se hace el checkpoint
2. Captura de pantalla en la que se compare el contenido del directorio del backup con el directorio con los metadatos de NameNode, antes y una vez que el servicio de backup se ha completado
3. Captura de pantalla del interfaz web del nodo de backup

2. TimeLineServer

El servidor de *línea temporal* de YARN mantiene un histórico y proporciona métricas de las aplicaciones ejecutadas mediante YARN (es similar a la funcionalidad del Job History Server porporcionado por MapReduce).

Proporciona tanto información genérica acerca de aplicaciones completadas (contenedores en los que se ejecutó la aplicación, intentos de ejecución, el nombre del usuario, de la cola, etc.) como información específica del framework concreto de la aplicación (por ejemplo, el framework MapReduce puede publicar información sobre el número de maps y reduces, u otros contadores). La información es accesible a través de un interfaz web o vía una API REST.

El Timeline Server se ejecuta como un demonio standalone que puede correr en un nodo del cluster o colocarse con el ResourceManager. Más información sobre el servicio en https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/TimelineServer.html.

Para añadir el TimeLineServer, tenéis que seguir los siguientes pasos (con el cluster funcionando):

1. Ve al NameNode/ResourceManager y detén el servicio ResourceManager
2. En este sistema, edita el fichero `yarn-site.xml` y añade las siguentes propiedades:

a) `yarn.timeline-service.hostname`: Nombre del equipo que ejecutará el demonio de línea de tiempo por defecto. Dale el valor `timelineserver` (llamaremos de esta forma al Docker que ejecutará el servicio).

b) `yarn.timeline-service.enabled`: Indica que si el servicio de linea de tiempo está activo o no. Dale el valor `true`.

c) `yarn.system-metrics-publisher.enabled`: Le indica al ResourceManager que publique las metricas de YARN en el timeline server. Dale el valor `true`.

```
  <property>
    <name>yarn.timeline-service.hostname</name>
    <value>timelineserver</value>
    <final>true</final>
  </property>

  <property>
```

```
    <name>yarn.timeline-service.enabled</name>
    <value>true</value>
    <final>true</final>
  </property>

  <property>
    <name>yarn.system-metrics-publisher.enabled</name>
    <value>true</value>
    <final>true</final>
  </property>
```

1. Reinicia el servicio ResourceManager
2. Inicia un nuevo Docker a partir de la imagen hadoop-base de la siguiente forma:

```
docker container run -ti --name timelineserver --network=hadoop-
cluster --hostname timelineserver --cpus=1 --memory=3072m --
expose=10200 -p 8188:8188 dsevilla/hadoop-base /bin/bash
```

1. En este nuevo Docker, levanta el servicio timelineserver ejecutando:

```
yarn --daemon start timelineserver
```

1. Vuelve al NameNode/ResourceManager y ejecuta una aplicación con yarn (la de el cálculo de pi o el wordcount).
2. Comprueba en el servidor web del TimeLineServer (http://localhost:8188) que se recoge la información de la ejecución

Información para la memoria

Incluir en la memoria:

1. Captura de pantalla del interfaz web del TimeLineServer en la que se vea que se ha recogido la información de la ejecución de una o más tareas

Tarea 2: Añadir un nuevo DataNode/NodeManager

1. Creación de ficheros de nodos incluidos y excluidos

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, haced lo siguiente en el NameNode (como usuario `hdadmin`):

Para los demonios del NameNode y del ResourceManager.

Crea cuatro ficheros: `${HADOOP_HOME}/etc/hadoop/dfs.include`, `${HADOOP_HOME}/etc/hadoop/dfs.exclude`, `${HADOOP_HOME}/etc/hadoop/yarn.include` y `${HADOOP_HOME}/etc/hadoop/yarn.exclude` (inicialmente vacíos).

En los ficheros `dfs.include` y `yarn.include`, poned los nombres de todos los DataNodes/NodeManagers que querramos que estén en el cluster (datanode1, datanode2, datanode3 y datanode4, un nombre por línea). Deja los ficheros `dfs.exclude` y `yarn.exclude` vacíos.

En el fichero de configuración `hdfs-site.xml`, añade dos propiedades:

- `dfs.hosts`: nombre de un fichero con lista de hosts que pueden actuar como DataNodes; si el fichero está vacío, cualquier nodo están permitido. Dale como valor, el path completo al fichero `dfs.include`.

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

- `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. Dale como valor, el path completo al fichero `dfs.exclude`.

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

En el fichero `yarn-site.xml`, añade dos propiedades:

- `yarn.resourcemanager.nodes.include-path`: nombre de un fichero con la lista de hosts que pueden actuar como NodeManagers; si el fichero está vacío, cualquier nodo están permitido. Dale como valor, el path completo al fichero `yarn.include`.

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

- `yarn.resourcemanager.nodes.exclude-path`: nombre de un fichero con la lista de hosts que no pueden actuar como NodeManagers; si el fichero está vacío, ninguno está excluido. Dale como valor, el path completo al fichero `yarn.exclude`.

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

Reinicia los demonios del NameNode y del ResourceManager.

Comprueba en los ficheros de log que se han incluido al HDFS y al YARN los nodos `datanode{1,2,3,4}`.

2. Añadir un datanode/nodemanager

Vamos a añadir un nuevo DataNode/NodeManager al cluster:

1. En el NameNode, añade el nombre de un nuevo nodo (datanode5) en el fichero `yarn.include` (no lo añadas, de momento, en el `dfs.include`).
2. Actualiza el ResourceManager con el nuevo NodeManager ejecutando:

```
yarn rmadmin -refreshNodes
```

1. Iniciar un nuevo contenedor para hacer de DataNode/NodeManager:

```
docker container run -d --name datanode5 --network=hadoop-cluster --
hostname datanode5 --cpus=1 --memory=3072m --expose=8000-10000 --
expose=50000-50200 datanode-image /inicio.sh
```

1. Comprueba usando (en el NameNode como usuario hdadmin) los comandos `hdfs dfsadmin -report` y `yarn node -list` que el nuevo contenedor se ha añadido al YARN pero no al HDFS
2. Añade ahora el nombre del nuevo nodo al ficheros `dfs.include`
3. Actualiza el NameNode con el nuevo DataNode ejecutando:

```
hdfs dfsadmin -refreshNodes
```

1. Comprueba de nuevo que ahora el contenedor sí está incluido en el HDFS. Puedes comprobarlo también el interfaz web del NameNode y de YARN.

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

```
hdfs balancer
```

Para más información,
véase https://hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html#balancer

Información para la memoria

Incluir en la memoria capturas de pantalla en las que se vean:

1. Las líneas de los ficheros de log del namenode y del resourcemanager que muestran que se han incluido los nodos indicados en los ficheros include (punÇto 4 del apartado 1 **Creación de ficheros de nodos incluidos y excluidos**)

2. Los pasos indicados para añadir un nuevo datanode/nodemanager, con las salidas de los comandos `hdfs dfsadmin -report` y `yarn node -list`. Se debe visualizar que el nodo inicialmente se añade a YARN y no a HDFS (paso 4 del apartado 2) y luego que se añade a ambos servicios.
3. Salida de la ejecución del balanceador de carga. Indica también cuántos datos se han movido y cuántos bloques tiene el datanode5

Tarea 3: Retirar un DataNode/NodeManager

En principio, el apagado de un DataNode/NodeManager puede hacerse directamente y no afecta al cluster. Sin embargo, si queremos hacer un apagado programado de un DataNode/nodeManager es preferible advertir al NameNode previamente.

Sigue los siguiente pasos para eliminar, por ejemplo, el datanode4.

1. Pon el nombre del nodo o nodos que queremos retirar en los fichero `dfs.exclude` y `yarn.exclude` y ejecutar

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

1. Comprueba que al cabo de un rato, usando el interfaz web y 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

Ya podríamos parar los demonios en el nodo decomisionado y parar el contenedor asociado. Si no queremos volver a incluirlo en el cluster:

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

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

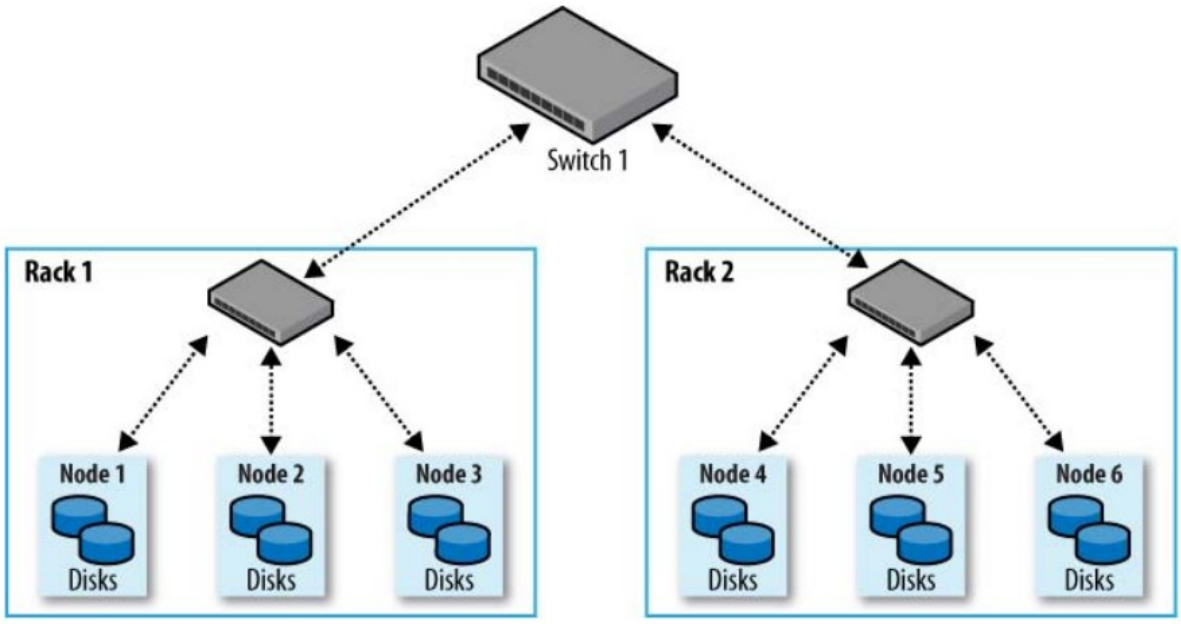
Información para la memoria

Añade a la memoria una captura de pantalla en las que se vea el interfaz web del HDFS mostrando que el datanode4 está decomisionado, y del interfaz web del YARN mostrando que el datanode4 ya no está entre los nodos disponibles.

Tarea 4: Rack awareness

Para obtener el máximo rendimiento, es importante configurar Hadoop para para que conozca la topología de nuestra red. Por defecto, Hadoop considera que todos los DataNodes/NodeManagers son iguales y están situados en un único rack, que se identifica como `/default-rack`.

Para clusters multirack, debemos indicar a Hadoop en que rack está cada nodo, para mejorar la eficiencia y la fiabilidad.



En la imagen, se muestra una arquitectura típica en 2 niveles de un cluster Hadoop. Esta topología puede describirse en forma de árbol, como /switch1/rack1 y /switch1/rack2, o, simplificando /rack1 y /rack2. Para indicarle esta topología a Hadoop, es necesario utilizar un script que mapee los nombres de los nodos al rack en el que se encuentran.

En nuestro caso, vamos a suponer que tenemos 2 racks (rack1 y rack2) y que tenemos dos nodos en cada rack. Haced lo siguiente en el NameNode (como usuario hdadmin):

1. Ejecuta el comando `hdfs dfsadmin -printTopology` para ver como es la topología actual. Apunta las IPs (sin los puertos) de los datanodos.
2. Apaga los demonios NameNode
3. Crea un fichero `$HADOOP_HOME/etc/hadoop/topology.data` que tenga en cada linea la IP de uno de los DataNodes y el rack donde está, como en este ejemplo (cambiando las IPs por las tuyas):

```
IPdatanode1     /rack1
IPdatanode2     /rack1
IPdatanode3     /rack2
IPdatanode5     /rack2
```

1. Crea un script de bash `$HADOOP_HOME/etc/hadoop/topology.script` como el siguiente (fuente: http://wiki.apache.org/hadoop/topology_rack_awareness_scripts). Dale permisos de ejecución (`chmod +x topology.script`).

```
#! /bin/bash

HADOOP_CONF=$HADOOP_HOME/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
```

1. Define en el fichero `core-site.xml` la propiedad `net.topology.script.file.name` y darle como valor el path completo al script

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

1. Inicia los demonios y comprueba que se han identificado los racks ejecutando `hdfs dfsadmin -printTopology`

Información para la memoria

Añade a la memoria una captura de pantalla en la que se vea ta salida del comando `hdfs dfsadmin -printTopology` mostrando la distribución por racks.