# CLúSTER SPARK CON HADOOP MANUALMENTE

## CREAR MÁQUINA VIRTUAL

Vamos a crear una máquina virtual en virtual box con las configuraciones e instalaciones básicas y después la clonaremos:

* Nombre: master (las otras tres máquinas se llamarán nodo1 y nodo2)
* RAM: 8GB (si es posible)
* Procesadores: 2
* Disco duro: 100GB
* Interfaz de red: Red NAT ("BigData"): 192.0.2.5 (la ip de las otras dos máquinas será 192.0.2.10, 192.0.2.20)
* Sistema operativo: Ubuntu 24
* Usuario / password: iabd / Password0

### FICHERO /etc/hosts

Modificamos el fichero /etc/hosts para que cuando clonemos la máquina reconozca los tres sistemas por el nombre:

***sudo nano /etc/hosts***

***/etc/hosts***

192.0.2.5 master

192.0.2.10 nodo1

192.0.2.20 nodo2

### SSH

Instalamos ssh para poder conectarnos al equipo vía ssh

***sudo apt-get update***

***sudo apt-get install openssh-server***

### JAVA

Instalamos java con los siguientes comandos:

***sudo apt-get install default-jdk***

Con el siguiente comando comprobamos cual es la versión que tenemos:

***java -version***

Ahora necesitamos saber cuál es la ruta donde se ha instalado Java

***dirname $(dirname $(readlink -f $(which java)))***

### HADOOP

Descargamos Hadoop (<https://hadoop.apache.org/releases.html>) y lo instalamos Hadoop en /opt

***cd /opt***

***sudo tar -zxvf /home/iabd/Downloads/hadoop-X.Y.Z.tar.gz***

***sudo mv hadoop-X.Y.Z hadoop***

***sudo chown -R iabd:iabd hadoop***

***sudo mkdir -p /datos/{namenode,datanode}***

***sudo chown -R iabd:iabd /datos***

***nano /opt/hadoop/etc/hadoop/core-site.xml***

**/opt/hadoop/etc/hadoop/core-site.xml**

*<property>*

*<name>fs.default.name</name>*

*<value>hdfs://master:9000</value>*

*</property>*

***nano /opt/hadoop/etc/hadoop/hdfs-site.xml***

**/opt/hadoop/etc/hadoop/hdfs-site.xml**

*<property>*

*<name>dfs.namenode.name.dir</name>*

*<value>/datos/namenode</value>*

*</property>*

*<property>*

*<name>dfs.datanode.data.dir</name>*

*<value>/datos/datanode</value>*

*</property>*

*<property>*

*<name>dfs.replication</name>*

*<value>2</value>*

*</property>*

*<property>*

*<name>dfs.webhdfs.enabled</name>*

*<value>true</value>*

*</property>*

*<property>*

*<name>dfs.namenode.http-address</name>*

*<value>master:9870</value>*

*</property>*

***nano /opt/hadoop/etc/hadoop/hadoop-env.sh***

***/opt/hadoop/etc/hadoop/hadoop-env.sh***

export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

### YARN

***nano /opt/hadoop/etc/hadoop/mapred-site.xml***

**/opt/hadoop/etc/hadoop/mapred-site.xml**

*<property>*

*<name>mapreduce.framework.name</name>*

*<value>yarn</value>*

*</property>*

*<property>*

*<name>yarn.app.mapreduce.am.env</name>*

*<value>HADOOP\_MAPRED\_HOME=$HADOOP\_HOME</value>*

*</property>*

*<property>*

*<name>mapreduce.map.env</name>*

*<value>HADOOP\_MAPRED\_HOME=$HADOOP\_HOME</value>*

*</property>*

*<property>*

*<name>mapreduce.reduce.env</name>*

*<value>HADOOP\_MAPRED\_HOME=$HADOOP\_HOME</value>*

*</property>*

***nano /opt/hadoop/etc/hadoop/yarn-site.xml***

**/opt/hadoop/etc/hadoop/yarn-site.xml**

*<property>*

*<name>yarn.resourcemanager.hostname</name>*

*<value>master</value>*

*</property>*

*<property>*

*<name>yarn.nodemanager.aux-services</name>*

*<value>mapreduce\_shuffle</value>*

*</property>*

***nano /opt/hadoop/etc/hadoop/workers***

**/opt/hadoop/etc/hadoop/workers**

*nodo1*

*nodo2*

### SPARK

Descargamos Spark (<https://spark.apache.org/downloads.html>) y lo instalamos en /opt

***cd /opt***

***sudo tar -zxvf /home/iabd/Downloads/spark-3.5.0-bin-hadoop3.tgz***

***sudo mv spark-X.Y.Z-biin-hadoopW spark***

***sudo chown -R iabd:iabd spark***

***cd /opt***

***sudo tar -zxvf /home/iabd/Descargas/spark-3.5.4-bin-hadoop3.tgz***

***sudo mv spark-3.5.4-bin-hadoop3 spark***

***sudo chown -R iabd:iabd spark***

### VARIABLES DE ENTORNO

***cd /home/iabd***

***nano /home/iabd/.bashrc***

***/home/iabd/.bashrc***

export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

export HADOOP\_HOME=/opt/hadoop

export HADOOP\_INSTALL=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export HADOOP\_YARN\_HOME=$HADOOP\_HOME

export SPARK\_HOME=/opt/spark

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin:$SPARK\_HOME/bin:$SPARK\_HOME/sbin

export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib/native"

***source ./.bashrc***

## CLONAR MÁQUINA

Vamos a clonar la máquina creada dos veces (recordar cambiar la MAC) y después hay que hacer algunos cambios en las máquinas clonadas:

* Cambiar el hostname: nodo1 y nodo2
* Cambiar la IP:
  + nodo1 a 192.0.2.10
  + nodo2 a 192.0.2.20

## DISTRIBUIR PARES DE CLAVES DE AUTENTICACIÓN PARA EL USUARIO DE HADOOP

En todos los equipos generamos la clave ssh (para el usuario iabd que es con el que hemos iniciado sesión) con el siguiente comando:

***cd /home/iabd***

***ssh-keygen -t rsa***

Al generar la clave dejamos el campo de la contraseña en blanco.

master

***cat /home/iabd/.ssh/id\_rsa.pub >> /home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@nodo1:/home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@nodo2:/home/iabd/.ssh/authorized\_keys***

nodo1

***cat /home/iabd/.ssh/id\_rsa.pub >> /home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@master:/home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@nodo2:/home/iabd/.ssh/authorized\_keys***

nodo2

***cat /home/iabd/.ssh/id\_rsa.pub >> /home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@master:/home/iabd/.ssh/authorized\_keys***

***scp /home/iabd/.ssh/authorized\_keys iabd@nodo1:/home/iabd/.ssh/authorized\_keys***

## CONFIGURACIÓN MASTER

### HADOOP

***hdfs namenode -format***

### SPARK

Crear los ficheros de configuración:

***cd /opt/spark***

***cp conf/spark-env.sh.template conf/spark-env.sh***

***cp conf/workers.template conf/workers***

Editar spark-env.sh para establecer algunas variables de entorno (añadir las siguientes líneas al final del fichero:

***nano /opt/spark/conf/spark-env.sh***

**/opt/spark/conf/spark-env.sh**

*export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64*

*export SPARK\_MASTER\_HOST='master'*

Configurar los workers:

***nano /opt/spark/conf/workers***

**/opt/spark/conf/workers**

*nodo1*

*nodo2*

## PUESTA EN MARCHA

### SPARK

Iniciar el clúster spark (desde el master):

***$SPARK\_HOME/sbin/start-master.sh***

Iniciar los workers (desde el master):

***$SPARK\_HOME/sbin/start-workers.sh spark://master:7077***

Vamos a acceder a la interfaz web para ver el estado del clúster

***http://master:8080***

Tabla

Descripción generada automáticamente

### HADOOP

Iniciar el clúster hadoop (desde el master):

***$HADOOP\_HOME/sbin/start-all.sh***

### EJEMPLO 1

Creamos un archivo example1.py en la máquina master con el siguiente código:

***cd***

***nano example1.py***

**/home/iabd/example1.py**

*from pyspark import SparkContext*

*sc = SparkContext("spark://master:7077","Example")*

*data = sc.parallelize([1, 2, 3, 4, 5])*

*result = data.reduce(lambda a, b: a + b)*

*print(result)*

*sc.stop()*

Ejecutamos el programa:

***spark-submit --master spark://master:7077 example1.py***

### EJEMPLO 2

Subimos un archivo a hdfs:

***hdfs dfs -put /home/iabd/el\_quijote.txt /***

Creamos un archivo example2.py en la máquina master con el siguiente código:

***cd***

***nano example2.py***

**/home/iabd/example2.py**

*from pyspark.sql import SparkSession*

*# Inicializar SparkSession*

*spark = SparkSession.builder.appName("QuijoteCount").getOrCreate()*

*# Leer el archivo desde HDFS*

*quijoteTxt = spark.read.text("hdfs://master:9000/el\_quijote.txt")*

*# Contar el número de líneas en el archivo*

*count = quijoteTxt.count()*

*count\_message = "El número total de líneas es: " + str(count)*

*# Obtener la primera línea del archivo*

*first\_line = quijoteTxt.first()*

*first\_line\_message = "La primera línea es: " + str(first\_line['value'])*

*# Filtrar las líneas que contienen la palabra "Quijote"*

*lineasConQuijote = quijoteTxt.filter(quijoteTxt.value.contains("Quijote"))*

*# Contar el número de líneas que contienen la palabra "Quijote"*

*quijote\_count = lineasConQuijote.count()*

*quijote\_count\_message = "El número de líneas que contienen la palabra 'Quijote' es: " + str(quijote\_count)*

*# Agregar todos los mensajes a una lista*

*messages = [count\_message, first\_line\_message, quijote\_count\_message]*

*# Crear un RDD con los mensajes*

*rdd = spark.sparkContext.parallelize(messages)*

*# Definir la ruta de salida en HDFS*

*output\_path = "hdfs://master:9000/salida"*

*# Guardar los mensajes en HDFS*

*rdd.saveAsTextFile(output\_path)*

*# Cerrar la sesión de Spark*

*spark.stop()*

*Ejecutamos el programa:*

***spark-submit --master spark://master:7077 example2.py***

Queda comprobar que ha creado el directorio en hdfs y dentro hay ficheros con el resultado

***hdfs dfs -ls /salida***

***hdfs dfs -cat /salida/part-00000***

***hdfs dfs -cat /salida/part-00001***

# CLúSTER SPARK CON HADOOP CON DOCKER

Instalamos docker-compose

***sudo apt-get install docker-compose***

El archivo docker-compose.yml que necesitamos para crear los tres contenedores puede ser como se indica a continuación.

***nano docker-compose.yml***

***docker-compose.yml***

*services:*

*namenode: # Servicio para el NameNode de Hadoop*

*image: bde2020/hadoop-namenode:latest # Imagen Docker del NameNode*

*container\_name: namenode # Nombre del contenedor*

*volumes: # Monta volúmenes para persistencia de datos*

*- namenode\_data:/hadoop/dfs/name*

*environment: # Variables de entorno para la configuración de Hadoop*

*- CLUSTER\_NAME=test # Nombre del clúster de Hadoop*

*env\_file: # Archivo de variables de entorno*

*- ./hadoop.env # Archivo externo con configuraciones de Hadoop*

*ports: # Puertos que se exponen*

*- "9870:9870" # Puerto para el UI del NameNode de Hadoop*

*- "9000:9000" # Puerto para las operaciones de HDFS*

*networks:*

*- spark\_network # Red a la que pertenece este servicio*

*restart: always*

*datanode: # Servicio para los DataNodes de Hadoop*

*image: bde2020/hadoop-datanode:latest # Imagen Docker del DataNode*

*container\_name: datanode # Nombre del contenedor*

*volumes: # Monta volúmenes para persistencia de datos*

*- datanode\_data:/hadoop/dfs/data*

*environment: # Variables de entorno para la configuración de Hadoop*

*- SERVICE\_PRECONDITION=namenode:9870 # Condición para esperar que el NameNode esté listo*

*env\_file: # Archivo de variables de entorno*

*- ./hadoop.env # Archivo externo con configuraciones de Hadoop*

*networks:*

*- spark\_network # Red a la que pertenece este servicio*

*restart: always*

*spark-master: # Servicio para el Spark Master*

*image: bde2020/spark-master:3.0.1-hadoop3.2 # Imagen Docker del Spark Master*

*container\_name: spark-master # Nombre del contenedor*

*ports: # Puertos que se exponen*

*- "8080:8080" # Puerto para el UI del Spark Master*

*- "7077:7077" # Puerto para la conexión de los Workers de Spark*

*environment: # Variables de entorno para la configuración de Spark*

*- "SPARK\_LOCAL\_IP=spark-master" # Configuración de IP local para Spark*

*networks:*

*- spark\_network # Red a la que pertenece este servicio*

*restart: always*

*spark-worker1: # Servicio para los Spark Workers*

*image: bde2020/spark-worker:3.0.1-hadoop3.2 # Imagen Docker del Spark Worker*

*container\_name: spark-worker1 # Nombre del contenedor*

*environment: # Variables de entorno para la configuración de Spark*

*- "SPARK\_MASTER=spark://spark-master:7077" # URL del Spark Master*

*networks:*

*- spark\_network # Red a la que pertenece este servicio*

*depends\_on:*

*- spark-master # Dependencia para esperar que el Spark Master esté listo*

*restart: always*

*spark-worker2: # Servicio para los Spark Workers*

*image: bde2020/spark-worker:3.0.1-hadoop3.2 # Imagen Docker del Spark Worker*

*container\_name: spark-worker2 # Nombre del contenedor*

*environment: # Variables de entorno para la configuración de Spark*

*- "SPARK\_MASTER=spark://spark-master:7077" # URL del Spark Master*

*networks:*

*- spark\_network # Red a la que pertenece este servicio*

*depends\_on:*

*- spark-master # Dependencia para esperar que el Spark Master esté listo*

*restart: always*

*volumes: # Definición de volúmenes para la persistencia de datos*

*namenode\_data:*

*datanode\_data:*

*networks: # Definición de redes personalizadas*

*spark\_network: # Red para la comunicación entre los servicios de Hadoop y Spark*

*driver: bridge*

En el fichero yml se indica que necesitamos el fichero hadoop.env con las configuraciones de hadoop:

***nano hadoop.env***

***hadoop.env***

*CORE\_CONF\_fs\_defaultFS=hdfs://namenode:9000*

Ahora ya podemos iniciar nuestro clúster con el siguiente comando:

***sudo docker-compose up -d***

Esto iniciará los servicios definidos en el archivo docker-compose.yml.

Para acceder a Spark UI: http://localhost:8080

Para Hadoop NameNode UI: http://localhost:9870

### EJEMPLO 1

Creamos un archivo example1.py en la máquina master con el siguiente código:

***cd***

***nano example1.py***

**/home/iabd/example1.py**

*from pyspark import SparkContext*

*sc = SparkContext(" spark://spark-master:7077", "Example")*

*data = sc.parallelize([1, 2, 3, 4, 5])*

*result = data.reduce(lambda a, b: a + b)*

*print(result)*

*sc.stop()*

Copiamos el script al contenedor de spark-master:

***sudo docker cp example1.py spark-master:/example1.py***

Accedemos al contenedor spark-master:

***sudo docker exec -it spark-master bash***

Ejecutamos el script con spark-submit:

***/spark/bin/spark-submit --master local[\*] /example1.py***

### EJEMPLO 2

Subimos un archivo a hdfs (primero hay que pasarlo al contenedor donde tenemos el namenode):

***sudo docker cp el\_quijote.txt namenode:/el\_quijote.txt***

***sudo docker exec -it namenode bash***

***hdfs dfs -put /el\_quijote.txt /el\_quijote.txt***

Creamos un archivo example2.py en la máquina virtual con el siguiente código:

***exit***

***cd***

***nano example2.py***

**/home/iabd/example2.py**

*# -\*- coding: utf-8 -\*-*

*from pyspark.sql import SparkSession*

*# Inicializar SparkSession*

*spark = SparkSession.builder.appName("QuijoteCount").getOrCreate()*

*# Leer el archivo desde HDFS*

*quijoteTxt = spark.read.text("hdfs://namenode:9000/el\_quijote.txt")*

*# Contar el número de líneas en el archivo*

*count = quijoteTxt.count()*

*count\_message = "El número total de líneas es: " + str(count)*

*# Obtener la primera línea del archivo*

*first\_line = quijoteTxt.first()*

*first\_line\_message = "La primera línea es: " + str(first\_line['value'])*

*# Filtrar las líneas que contienen la palabra "Quijote"*

*lineasConQuijote = quijoteTxt.filter(quijoteTxt.value.contains("Quijote"))*

*# Contar el número de líneas que contienen la palabra "Quijote"*

*quijote\_count = lineasConQuijote.count()*

*quijote\_count\_message = "El número de líneas que contienen la palabra 'Quijote' es: " + str(quijote\_count)*

*# Agregar todos los mensajes a una lista*

*messages = [count\_message, first\_line\_message, quijote\_count\_message]*

*# Crear un RDD con los mensajes*

*rdd = spark.sparkContext.parallelize(messages)*

*# Definir la ruta de salida en HDFS*

*output\_path = "hdfs://namenode:9000/salida"*

*# Guardar los mensajes en HDFS*

*rdd.saveAsTextFile(output\_path)*

*# Cerrar la sesión de Spark*

*spark.stop()*

Copiamos el script al contenedor de spark-master:

***sudo docker cp example2.py spark-master:/example2.py***

Accedemos al contenedor spark-master:

***sudo docker exec -it spark-master bash***

Ejecutamos el script con spark-submit:

***/spark/bin/spark-submit --master local[\*] /example2.py***

Queda comprobar que ha creado el directorio en hdfs y dentro hay ficheros con el resultado

***sudo docker exec -it namenode bash***

***hdfs dfs -ls /salida***

***hdfs dfs -cat /salida/part-00000***

***hdfs dfs -cat /salida/part-00001***

# CLúSTER SPARK CON S3 CON AMAZON AWS

**Paso 1**: Crear un Clúster de EMR

* Nombre: ClusterSpark
* Paquete de aplicaciones: Spark 3.5.0
* Grupo de instancias – Tarea 1 de 1 – Eliminar grupo de instancias
* Opción de aprovisionamiento y escaldo de clúster – Establecer el tamaño del clúster manualmente: 2 instancias centrales
* Terminación del clúster: Terminar el clúster manualmente
* Configuración de seguridad y par de claves de EC2 - Par de claves de Amazon EC2 para el protocolo SSH al clúster: vockey
* Rol de servicio de Amazon EMR - Elegir un rol de servicio existente - Rol de servicio: EMR\_DefaultRole
* Perfil de instancia de EC2 para Amazon EMR - Elegir un perfil de instancia existente - Perfil de instancia: EMR\_EC2\_DefaultRole
* Crear clúster

**Paso 2**: Modificar el grupo de seguridad del nodo principal para abrir el puerto SSH

### EJEMPLO 1

Conectarse al nodo principal del clúster EMR a través de SSH

Creamos un archivo example1.py en el nodo principal con el siguiente código:

***nano example1.py***

**example1.py**

*from pyspark import SparkContext*

*sc = SparkContext(appName="Example")*

*data = sc.parallelize([1, 2, 3, 4, 5])*

*result = data.reduce(lambda a, b: a + b)*

*print(result)*

*sc.stop()*

Ejecutar el script con spark-submit:

***spark-submit --master local[\*] example1.py***

### EJEMPLO 2

Conectarse al nodo principal del clúster EMR a través de SSH

Creamos un archivo example2.py en el nodo principal con el siguiente código:

***nano example2.py***

**example2.py**

*from pyspark.sql import SparkSession*

*# Inicializar SparkSession*

*spark = SparkSession.builder.appName("QuijoteCount").getOrCreate()*

*# Leer el archivo desde S3*

*quijoteTxt = spark.read.text("s3://aws-logs-529076800613-us-east-1/elasticmapreduce/j-32GDW9M2TLJDG/el\_quijote.txt")*

*# Contar el número de líneas en el archivo*

*count = quijoteTxt.count()*

*count\_message = "El número total de líneas es: " + str(count)*

*# Obtener la primera línea del archivo*

*first\_line = quijoteTxt.first()*

*first\_line\_message = "La primera línea es: " + str(first\_line['value'])*

*# Filtrar las líneas que contienen la palabra "Quijote"*

*lineasConQuijote = quijoteTxt.filter(quijoteTxt.value.contains("Quijote"))*

*# Contar el número de líneas que contienen la palabra "Quijote"*

*quijote\_count = lineasConQuijote.count()*

*quijote\_count\_message = "El número de líneas que contienen la palabra 'Quijote' es: " + str(quijote\_count)*

*# Agregar todos los mensajes a una lista*

*messages = [count\_message, first\_line\_message, quijote\_count\_message]*

*# Crear un RDD con los mensajes*

*rdd = spark.sparkContext.parallelize(messages)*

*# Definir la ruta de salida en HDFS*

*output\_path = "s3://aws-logs-529076800613-us-east-1/elasticmapreduce/j-32GDW9M2TLJDG/salida"*

*# Guardar los mensajes en HDFS*

*rdd.saveAsTextFile(output\_path)*

*# Cerrar la sesión de Spark*

*spark.stop()*

En S3 descargar los archivos generados para ver su contenido