
BIG DATA

INTRODUCCION A SPARK

EDUARD LARA

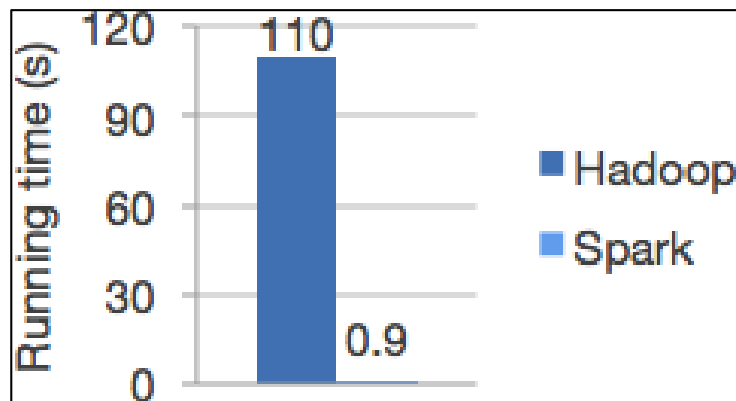


1. INTRODUCCION A SPARK

- ❑ La tecnología Spark está bastante de moda hoy en día.
- ❑ Apache Spark es un entorno que con ciertas similitudes a Hadoop pero también muy distinto
- ❑ Es un entorno de procesamiento distribuido y paralelo que está orientado a trabajar **en memoria** (análisis en tiempo real), mientras Hadoop MapReduce trabaja en disco y procesos batch
- ❑ Por eso es mucho más rápido que Hadoop MapReduce
- ❑ **Permite el análisis rápido** de grandes conjuntos de datos
- ❑ Permite trabajar con distintos entornos como Bases de Datos NoSQL, implementa procesos en Real Time, machine learning o análisis de grafos etc

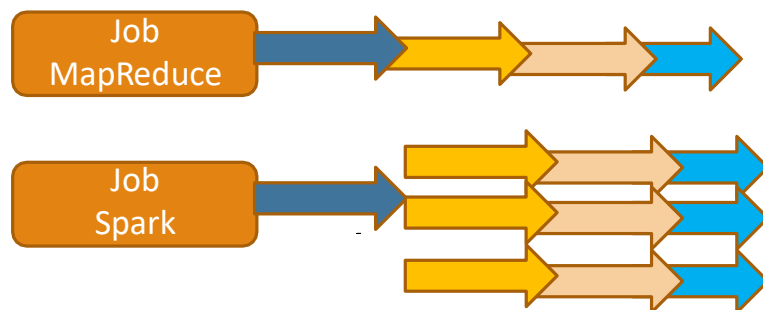
1. INTRODUCCION A SPARK

- ❑ Es un producto totalmente distinto a Hadoop, pero se integran bien y son compatibles
- ❑ Se puede ejecutar Spark por separado o se puede ejecutar con el cluster Hadoop de manera conjunta
- ❑ A continuación se muestra un grafico de la diferencia de velocidad que podemos tener en un momento dado entre Hadoop MapReduce y Spark. Aunque depende mucho del entorno y para que estemos haciendo el trabajo



1. INTRODUCCION A SPARK

- ❑ Al contrario que Hadoop MapReduce que trabaja con procesos tipo Batch, Spark está orientado al trabajo in-memory y en entornos de procesamiento real
- ❑ Por lo tanto mientras MapReduce trabaja de manera secuencial, Spark lo hace en paralelo
- ❑ En el típico Job MapReduce vamos dando distintos pasos hasta conseguir el trabajo, mientras que en un job de tipo Spark el mismo trabajo se paraleliza n veces para poder conseguir un rendimiento mayor



1. INTRODUCCION A SPARK

- ❑ Spark es totalmente compatible con Hadoop
 - ❖ Se puede ejecutar sobre HDFS y otros datasources que vienen con Hadoop
 - ❖ Se integra con MapReduce. Se puede usar en el mismo cluster que MapReduce
 - ❖ Una aplicación Spark se puede lanzar sobre YARN
 - ❖ Se pueden mezclar aplicaciones Spark y MapReduce para batch y Real Time de forma cooperativa
- ❑ Soporta múltiples fuentes de datos
 - ❖ HIVE, JSON
 - ❖ CASSANDRA, CSV
 - ❖ RDBMS

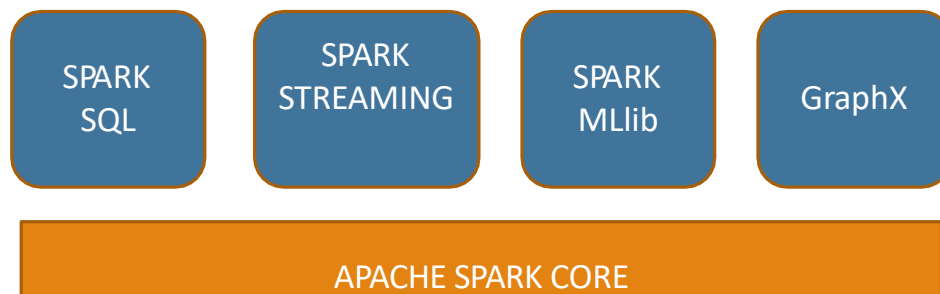
1. INTRODUCCION A SPARK

- ❑ Está construido en Scala, pero se pueden escribir aplicaciones en Java, Python y R.
- ❑ Se recomienda conocer el lenguaje natural (Scala) en el que está hecho el producto (Spark), ya que podemos encontrar características muy interesantes o opciones que no están en los otros lenguajes
- ❑ Es un lenguaje relativamente sencillo y fácil de utilizar, con similitudes con otros lenguajes ya conocidos
- ❑ Dispone de un Shell interactivo, otra diferencia respecto Hadoop, con el que empezar a hacer pruebas

1. INTRODUCCION A SPARK

- ❑ Spark está compuesto por un componente que es el Core con unas librerías específicas y luego asociados tenemos un conjunto de librerías que nos dan los 4 pilares de Spark:

- ❑ SQL
- ❑ Streaming
- ❑ MLlib
- ❑ GraphX



- ❑ A Spark se le pueden añadir mas cosas como a Hadoop, pero estas librerías son los que vienen por defecto

1. INTRODUCCION A SPARK

Spark Core

- ❑ El motor básico que sirve para el procesamiento en paralelo y distribuido. Aunque está construido en Scala, hay APIs para usarlo en Python, Java y R.
- ❑ Se encarga de la base del cluster:
 - ❑ Gestión de la memoria
 - ❑ Recuperación ante fallos
 - ❑ Planificación, distribución de trabajos en el cluster
 - ❑ Monitorizar todos los trabajos. Si se caen intentar relanzarlos en otro entorno
 - ❑ Accedes a los sistemas de almacenamiento
 - ❑ Poner en coordinación el resto de herramientas

1. INTRODUCCION A SPARK

Spark Core

- ❑ Spark Core usa una estructura de datos especial denominada **RDD** (Resilient Distributed Datasets).
- ❑ Representa el almacenamiento de Spark. Permite, en vez de trabajar en disco, realizar procesos de una alta disponibilidad y fault tolerant 'in-memory' de datos
- ❑ Son colecciones de registros inmutables y particionados que pueden ser manejadas en paralelo.
- ❑ Los RDDs pueden contener objetos de cualquier tipo: Python, Scala, Java o personalizados.
- ❑ Los RDD se crean habitualmente transformándolos de otros RDDs existentes, o cargandos los datos de una Fuente externa, como por ejemplo HDFS o HBase.

1. INTRODUCCION A SPARK

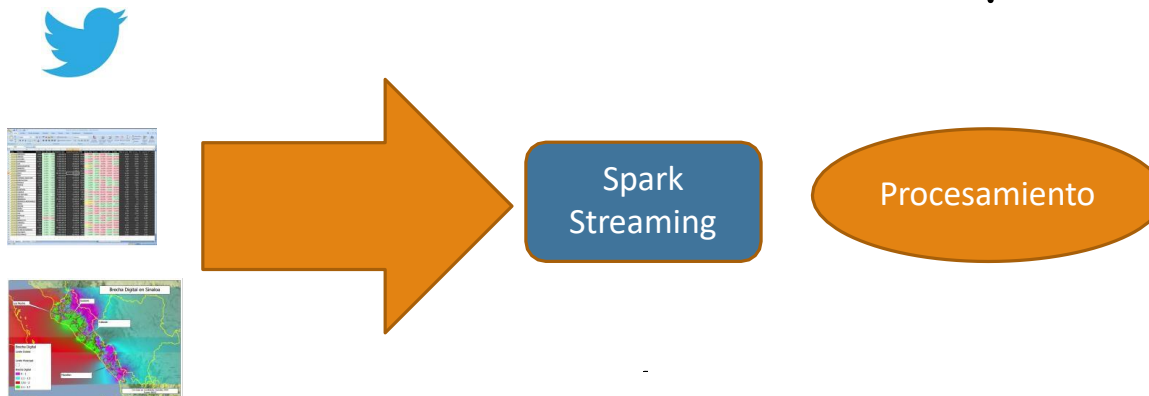
Spark Streaming

- ❑ Una de las 4 librerías que tiene el core de Spark
- ❑ Es uno de los componentes más utilizados dentro de Spark ya que nos permite procesar fuentes de datos en tiempo real como streaming (streaming data)
- ❑ Permite procesar con una alta tolerancia a fallos (trabajamos con RDDs otros componentes adicionales) y un gran rendimiento las fuentes continuas "vivas" de información (streaming) que le suministremos.
- ❑ Esos RDDs de fuentes vivas se puede ir alimentando, transformando y se puede ir realizando operaciones sobre ellos.

1. INTRODUCCION A SPARK

Spark Streaming

- ❑ Su unidad fundamental de trabajo es el Dstream (serie de RDDs)
- ❑ Datos de Twitter, financieros (información de bolsa) o geográficos (información de tiempo) se pueden pasar a Streaming. Spark Streaming puede ir leyendo esta información viva que se le va suministrando continuamente e ir procesándola.



1. INTRODUCCION A SPARK

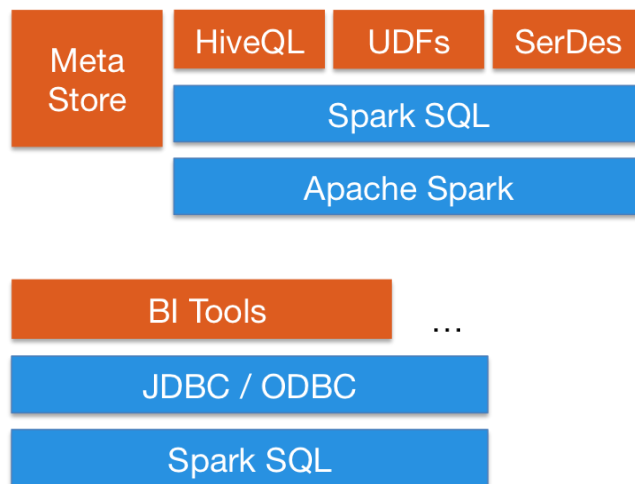
Spark Sql

- ❑ Permite manejar un entorno que integra comandos y componentes relacionales junto con la programación funcional de Spark Scala y el resto
- ❑ Podemos usar tanto SQL como Hive Query Language
- ❑ Permite el acceso a múltiples fuentes de datos
- ❑ Dispone de 4 librería básicas
 - ❑ Data Source y DataFrame
 - ❑ Interprete y Optimizador para generar planes de ejecución y mejorar el acceso
 - ❑ Servicio SQL
 - ❑ Permite el acceso por JDBC o ODBC

1. INTRODUCCION A SPARK

Spark SQL

- ❑ Tenemos Apache Spark, Spark SQL y por encima podemos utilizar distintos componentes para acceder a la información
- ❑ Si tuviéramos herramientas que quisiéramos usar a través de SPARC SQL podemos acceder por dos vías JDBC o ODBC



1. INTRODUCCION A SPARK

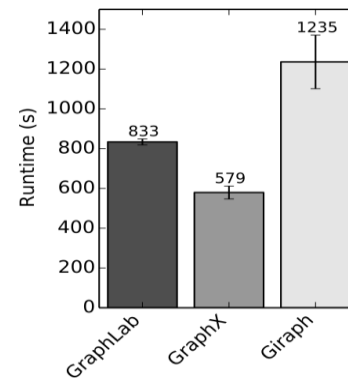
Spark GraphX (tercer componente de Spark)

- ❑ Es un API para trabajar con procesamientos paralelos y distribuidos en grafos. Los grafos es una forma de trabajar con almacenamientos que en vez de guardarlo con las típicas estructuras de columnas o documental lo guarda en forma de grafos
- ❑ GraphX implementa los RDG (Resilient Distributed Graph) una abstracción funcional de los RDD's.
- ❑ Los RDG's asocian registros con los vertices y bordes de un grafo.
- ❑ Como usuario de Spark se pueden seguir viendo o bien grafos o bien como colecciones tradicionales de RDD.

1. INTRODUCCION A SPARK

Spark GraphX (tercer componente de Spark)

- ❑ Dada la complejidad de GraphX, dispone de una gran cantidad de algoritmos y librerías preparadas, que permiten agilizar el proceso de construcción de aplicaciones. Podemos construir aplicaciones con muy poca cantidad de líneas y sobretodo mejorar enormemente el rendimiento y la velocidad
- ❑ Ejemplo de tiempos con otros productos que hay hoy en día en el mercado que son competidores directos de Spark.



1. INTRODUCCION A SPARK

Spark GraphX

- ❑ Ya existen herramientas independientes que permiten trabajar con estos componentes de streaming y de SQL, como el caso de GraphX, con Neo4j
- ❑ La ventaja de utilizar Spark, con respecto a utilizar otras herramientas más independientes, es que lo junta todo en un entorno de cluster con un gran rendimiento y con una alta disponibilidad brutal.

1. INTRODUCCION A SPARK

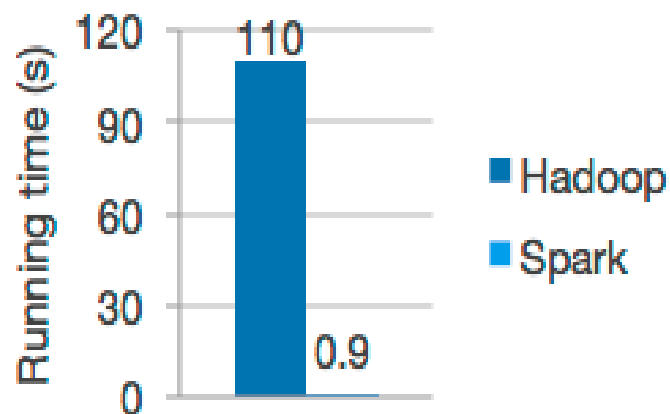
Mlib

- ❑ Utilizar Machine Learning está muy de moda para distintos entornos: financiero, educación, algoritmos de control de seguridad, etc
- ❑ El machine learning es otro de esos componentes que ha llegado para quedarse y que impactará notablemente en la vida de todos nosotros, en numerosos sectores
- ❑ Mlib dispone de una variedad de algoritmos ya preconstruidos que nos permiten trabajar con Machine Learning de manera muy transparente y de otros procesos como data cleaning, clasificación, clustering, transformación, regresión, extracción, etc
- ❑ Permite su ejecución sobre HDFS, HBAs, etc...

1. INTRODUCCION A SPARK

Mlib

- Un pequeño ejemplo con una regresión logística en Hadoop o Spark



Logistic regression in Hadoop and Spark

2. DESCARGA DE SPARK

- ❑ Vamos a descargar e instalar Spark dentro de nuestro entorno Hadoop y ver cómo podemos utilizarlo ahí
- ❑ Hay diferentes tipos de descarga y de instalación. En la pagina de descargas de Spark tenemos 2 opciones
 - ❑ Podemos hacer una instalación con Hadoop pre-empaquetado que nos viene muy bien para empezar a trabajar si no tenemos una infraestructura hadoop
 - ❑ También podemos descargar un binario sin Hadoop, para usarlo con una plataforma hadoop de forma independiente (este seria nuestro caso)
- ❑ Otras dos maneras de poder descargar e instalar Spark es a través de Maven o con PyPi

2. DESCARGA DE SPARK

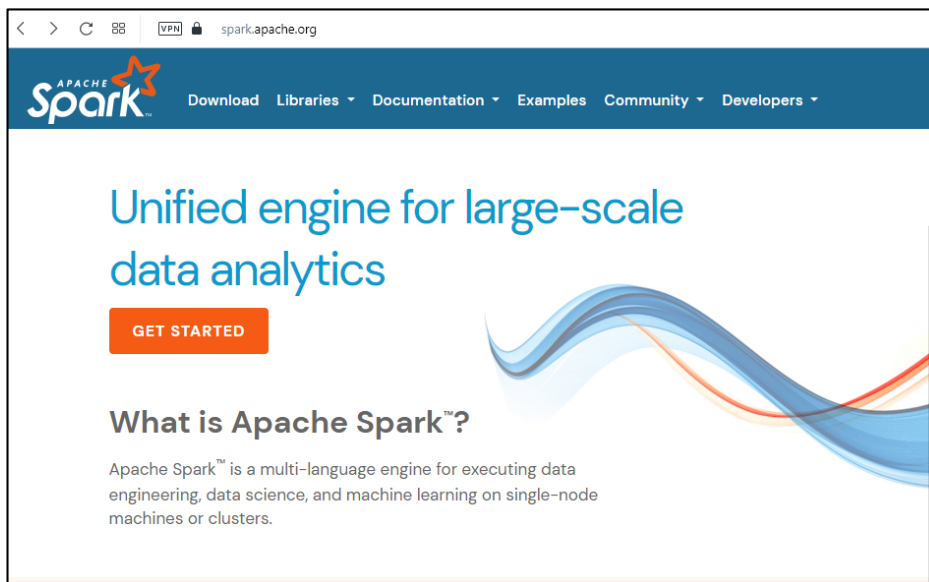
- ❑ Spark se puede usar en Windows, Unix y Mac
- ❑ Podemos ejecutarlo de forma local en una maquina virtual para empezar a probar. En realidad el único requisito necesario para trabajar con Spark sobretodo en forma Standalone es tener Java.
- ❑ Se necesita Java 8 (o superior) y dependiendo de las APIs de los componentes que vayamos a usar, se necesita Python 2.7 (o superior) y R 3.1 (o superior)
- ❑ Para la versión de Spark 3.2, se usa Scala 2.13
- ❑ En Spark podemos usar todos estos lenguajes Java, Python, Scala , R, SQL

2. DESCARGA DE SPARK

- ❑ Podemos instalar nuestro Spark en un entorno de cluster o bien de forma standalone.
- ❑ Despliegues soportados en la versión 2.3
 - ❑ Amazon EC2: nos permite descargarnos unos scripts que nos monta un Cluster EC2 en 5 minutos
 - ❑ Standalone: cluster standalone sin necesidad de un cluster manager. Va bien para las primeras pruebas, montar un pequeño cluster independiente de Hadoop
 - ❑ Mesos para desplegar en un cluster Apache Mesos
 - ❑ Yarn para desplegar en un cluster Hadoop
 - ❑ Kubernetes, despliegue en infraestructura dockers. Plataforma impulsada por Google

2. DESCARGA DE SPARK

Paso 1. Vamos a la página web de Apache Spark
<https://spark.apache.org>. Aquí tenemos una serie de documentación e información sobre Spark, como sus características clave: Batch/streaming Data, SQL Analytics, Machine learning y Data Science at scale



Simple. Fast. Scalable. Unified.

Key features

Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.

SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.

Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling

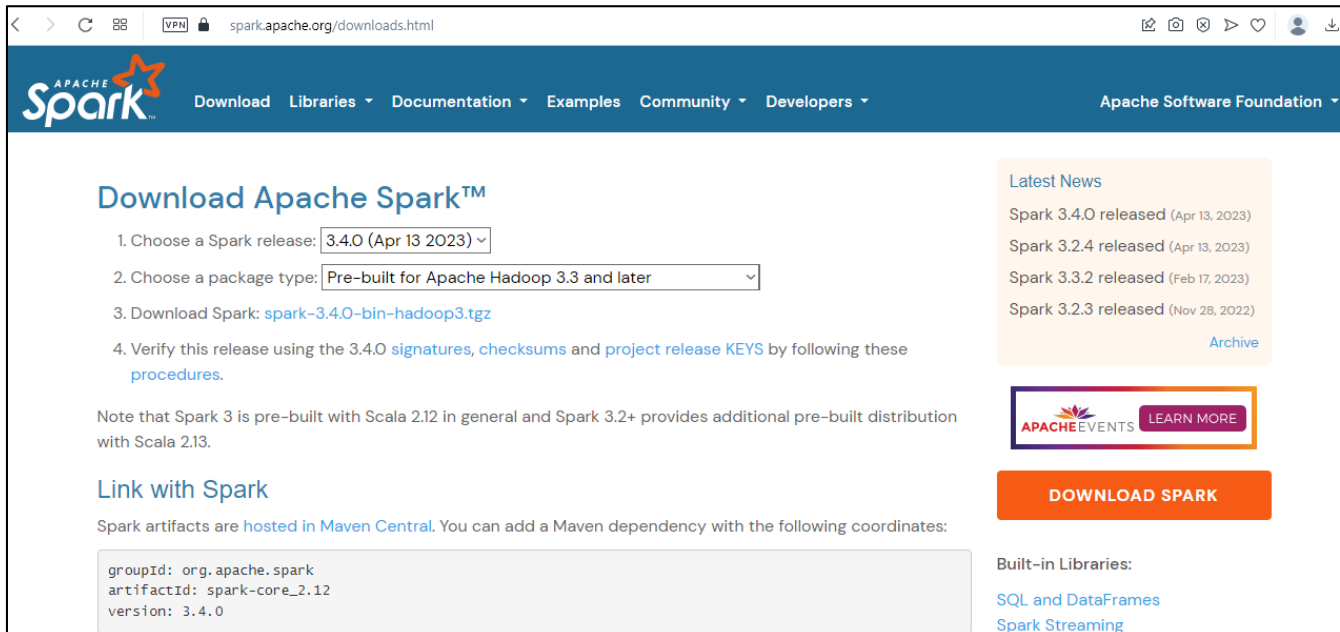
Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

2. DESCARGA DE SPARK

Paso 2. En la parte superior vemos las librerías que podemos utilizar: SQL and Dataframes, Spark Streaming, MLib (machine learninh) y GraphX

Paso 3. Haciendo click en Download vamos a la página de descargas, donde podemos descargarlo o bien a través del enlace tar.gz, o bien con Maven o PyPi o source code



The screenshot shows the Apache Spark download page. The header includes the Apache Spark logo and navigation links: Download, Libraries, Documentation, Examples, Community, and Developers. The main content area is titled "Download Apache Spark™" and contains a step-by-step guide:

- Choose a Spark release: 3.4.0 (Apr 13 2023) (dropdown menu)
- Choose a package type: Pre-built for Apache Hadoop 3.3 and later (dropdown menu)
- Download Spark: [spark-3.4.0-bin-hadoop3.tgz](#)
- Verify this release using the 3.4.0 [signatures](#), [checksums](#) and [project release KEYS](#) by following these [procedures](#).

Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

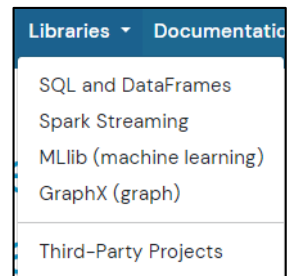
Link with Spark

Spark artifacts are [hosted in Maven Central](#). You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.12
version: 3.4.0
```

On the right side, there is a "Latest News" section with links to Spark 3.4.0, 3.2.4, 3.3.2, and 3.2.3 releases. Below this is an "APACHE EVENTS" banner and a large orange "DOWNLOAD SPARK" button.

At the bottom right, under "Built-in Libraries:", there are links for [SQL and DataFrames](#) and [Spark Streaming](#).

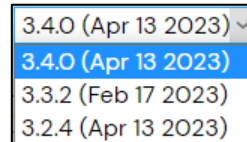


A dropdown menu titled "Libraries" with the following options:

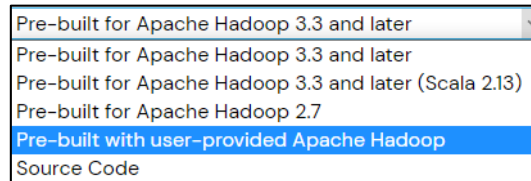
- SQL and DataFrames
- Spark Streaming
- MLlib (machine learning)
- GraphX (graph)
- Third-Party Projects

2. DESCARGA DE SPARK

Paso 4. En la parte superior podemos elegir la versión con la que queremos trabajar: 3.4.0, 3.3.2 y la 3.2.4. Hay poca diferencia entre las versiones según la fecha:



A continuación podemos indicar el tipo de descarga que queremos hacer. Tenemos 4 posibilidades de descarga:



Descargaremos la versión Pre-built with user-provided Apache Hadoop, porque ya tenemos Hadoop. Si pinchamos se cambia el nombre del producto a bajar

3. Download Spark: [spark-3.4.0-bin-without-hadoop.tgz](#)

2. DESCARGA DE SPARK

Paso 5. Como tenemos la versión de hadoop 3.2.4, podría haber problemas con la versión user-provided de Spark 3.4.0, que está más orientada hacia hadoop 3.3.0

```
hadoop@nodo1 (192.168.0.101) - byobu
File Edit View Search Terminal Help
hadoop@nodo1:~$ hadoop version
Hadoop 3.2.4
Source code repository Unknown -r 7e5d9983b388e372fe640f21f048f2f2ae6e9eba
Compiled by ubuntu on 2022-07-12T11:58Z
Compiled with protoc 2.5.0
From source with checksum ee031c16fe785bbb35252c749418712
This command was run using /opt/hadoop/share/hadoop/common/hadoop-common-3.2.4.jar
hadoop@nodo1:~$ cd Downloads/
```

Pre-built for Apache Hadoop 3.3 and later
Pre-built for Apache Hadoop 3.3 and later
Pre-built for Apache Hadoop 3.3 and later (Scala 2.13)
Pre-built for Apache Hadoop 2.7
Pre-built with user-provided Apache Hadoop
Source Code

A la versión de Spark 3.3.2 le pasa lo mismo, por lo tanto a priori la mejor opción consiste en descargar la versión user-provided de Spark 3.2.4 por tema de compatibilidad

Download Apache Spark™

1. Choose a Spark release: 3.2.4 (Apr 13 2023) ▾

2. Choose a package type:

Pre-built for Apache Hadoop 3.2 and later
Pre-built for Apache Hadoop 3.2 and later
Pre-built for Apache Hadoop 3.2 and later (Scala 2.13)
Pre-built for Apache Hadoop 2.7
Pre-built with user-provided Apache Hadoop
Source Code

Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

Download Apache Spark™

1. Choose a Spark release: 3.2.4 (Apr 13 2023) ▾

2. Choose a package type:

Pre-built with user-provided Apache Hadoop

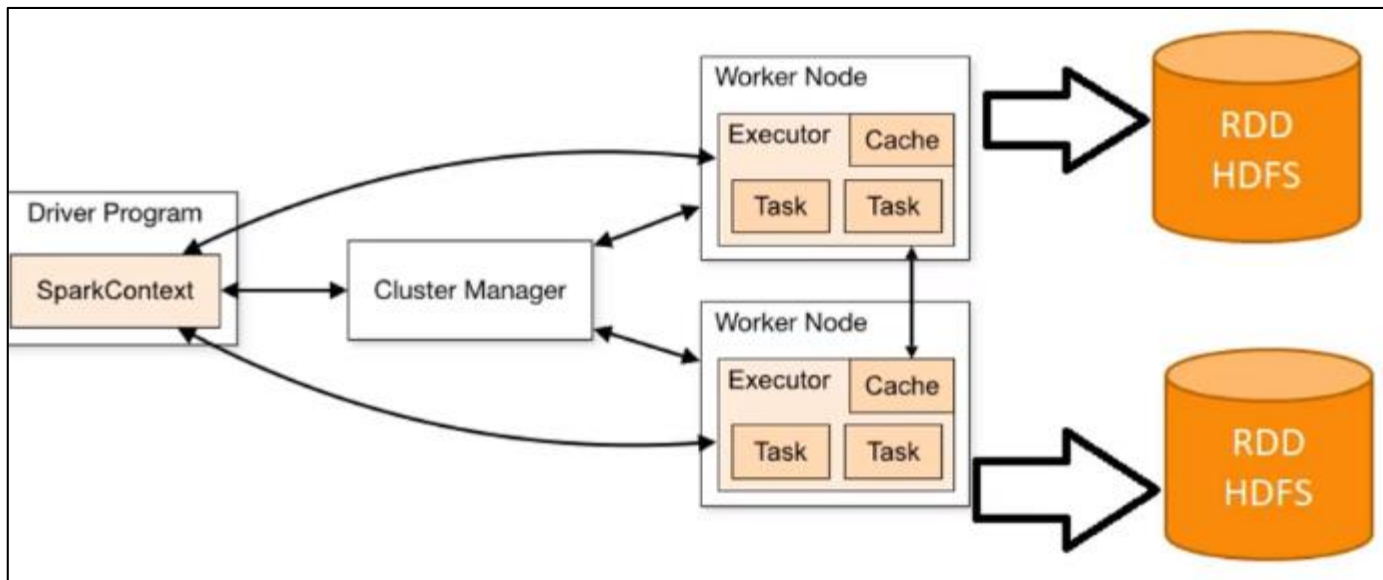
3. Download Spark: [spark-3.2.4-bin-without-hadoop.tgz](#)

4. Verify this release using the 3.2.4 [signatures](#), [checksums](#) and [project release KEYS](#) by following these [procedures](#).

Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

3. ARQUITECTURA DE PROCESOS SPARK

- ❑ Montaremos nuestro primer entorno de Spark dentro de un clúster de tipo standalone independiente es decir del propio Spark. Y posteriormente lo integraremos con nuestro Hadoop
- ❑ Aquí tenemos todos los componentes de Spark a nivel de procesos que funcionan independientemente del tipo de cluster donde vayamos a desplegar nuestro Spark



3. ARQUITECTURA DE PROCESOS SPARK

- ❑ Una aplicación Spark o programa Driver es un conjunto de procesos que se van a lanzar dentro de un cluster y que son coordinados por el componente SparkContext
- ❑ Cada programa tienen que implementar este objeto
- ❑ El SparkContext se conecta al gestor del cluster (Cluster Manager) que puede ser: Mesos, Hadoop o un cluster independiente.
- ❑ El Cluster Manager va a reservar en cada uno de los nodos necesarios del cluster, espacio para un componente denominado Executor
- ❑ El Executor es el proceso donde se va a mandar la aplicación java, Python, Scala para ser ejecutada

3. ARQUITECTURA DE PROCESOS SPARK

- ❑ El Executor va a usar varias tareas que se van a lanzar dentro de cada nodo
- ❑ También se utiliza una parte de memoria caché, ya que Spark es un ejecutor de procesos in-memory
- ❑ Cada worker tendrá executors y cada executor tendrá distintas tareas que realizarán esos procesos que se están lanzando desde la aplicación Java, Python, Scala
- ❑ A la hora de acceder a los datos podremos acceder o bien a HDFS o bien a los RDDs del propio Spark
- ❑ Esta arquitectura de procesos es independiente respecto al cluster donde lo queremos desplegar
- ❑ Mientras Spark pueda lanzar un Executor y poder empezar a coordinar los trabajos las tareas, se puede lanzar sobre Apache Hadoop, Apache Mesos, StandAlone

4. INSTALACION DE SPARK

Paso 1. Descargaremos la versión 3.2.4 de Spark without hadoop por tema de compatibilidad con nuestra versión de hadoop 3.2.4. En principio la instalación es igual como en otros productos como Hive, Hue, etc



Paso 2. Descomprimos el fichero tgz y el directorio resultante lo renombramos a spark

```
hadoop@nodo1:~/Downloads$ tar xvf spark-3.2.4-bin-without-hadoop.tgz
```

```
hadoop@nodo1:~/Downloads$ ls
hadoop-3.2.4.tar.gz  spark-3.2.4-bin-without-hadoop
puertos.csv         spark-3.2.4-bin-without-hadoop.tgz
hadoop@nodo1:~/Downloads$ mv spark-3.2.4-bin-without-hadoop spark
hadoop@nodo1:~/Downloads$ ls
hadoop-3.2.4.tar.gz  puertos.csv  spark  spark-3.2.4-bin-without-hadoop.tgz
hadoop@nodo1:~/Downloads$
```

4. INSTALACION DE SPARK

Paso 3. Movemos el directorio spark al directorio /opt/hadoop como el resto de programas instalados. Dentro del directorio spark podemos ver una serie de directorios interesantes: bin, examples, conf, jars (donde están las librerías), kubernetes, Python, R, etc

```
hadoop@nodol1:~/Downloads$ mv spark /opt/hadoop
hadoop@nodol1:~/Downloads$ ls -l /opt/hadoop
total 220
drwxr-xr-x  2 hadoop hadoop  4096 jul 12  2022 bin
drwxr-xr-x  3 hadoop hadoop  4096 jul 12  2022 etc
drwxrwxr-x 11 hadoop hadoop  4096 mar 28 22:06 hive
drwxr-xr-x  9 hadoop hadoop  4096 mar 29 07:32 hue
drwxr-xr-x  2 hadoop hadoop  4096 jul 12  2022 include
drwxr-xr-x  3 hadoop hadoop  4096 jul 12  2022 lib
drwxr-xr-x  4 hadoop hadoop  4096 jul 12  2022 libexec
-rw-rw-r--  1 hadoop hadoop 150571 jul 12  2022 LICENSE
drwxrwxr-x  3 hadoop hadoop  4096 mar 29 07:30 logs
-rw-rw-r--  1 hadoop hadoop 21932 jul 12  2022 NOTICE
-rw-rw-r--  1 hadoop hadoop  1397 feb 23 19:21 README
drwxr-xr-x  3 hadoop hadoop  4096 jul 12  2022 sbin
drwxr-xr-x  4 hadoop hadoop  4096 jul 12  2022 share
drwxr-xr-x 13 hadoop hadoop  4096 abr  9 20:58 spark
hadoop@nodol1:~/Downloads$
```

```
hadoop@nodol1:/opt/hadoop/spark$ ls -l
total 148
drwxr-xr-x  2 hadoop hadoop  4096 abr  9 20:58 bin
drwxr-xr-x  2 hadoop hadoop  4096 abr  9 20:58 conf
drwxr-xr-x  5 hadoop hadoop  4096 abr  9 20:58 data
drwxr-xr-x  4 hadoop hadoop  4096 abr  9 20:58 examples
drwxr-xr-x  2 hadoop hadoop 12288 abr  9 20:58 jars
drwxr-xr-x  4 hadoop hadoop  4096 abr  9 20:58 kubernetes
-rw-r--r--  1 hadoop hadoop 22878 abr  9 20:58 LICENSE
drwxr-xr-x  2 hadoop hadoop  4096 abr  9 20:58 licenses
-rw-r--r--  1 hadoop hadoop 57677 abr  9 20:58 NOTICE
drwxr-xr-x  7 hadoop hadoop  4096 abr  9 20:58 python
drwxr-xr-x  3 hadoop hadoop  4096 abr  9 20:58 R
-rw-r--r--  1 hadoop hadoop  4512 abr  9 20:58 README.md
-rw-r--r--  1 hadoop hadoop   145 abr  9 20:58 RELEASE
drwxr-xr-x  2 hadoop hadoop  4096 abr  9 20:58 sbin
drwxr-xr-x  2 hadoop hadoop  4096 abr  9 20:58 yarn
hadoop@nodol1:/opt/hadoop/spark$
```

4. INSTALACION DE SPARK

Paso 4. Dentro del directorio bin nos encontramos con varios ficheros ejecutables como spark-shell y pyspark que nos van a permitir acceder a Spark con los lenguajes de programación Scala y Python respectivamente. También hay para R y SQL.

```
hadoop@nodo1:/opt/hadoop/spark/bin$ ls
beeline                pyspark                spark-class.cmd       spark-sql
beeline.cmd            pyspark2.cmd          sparkR                 spark-sql2.cmd
docker-image-tool.sh   pyspark.cmd           sparkR2.cmd           spark-sql.cmd
find-spark-home        run-example            sparkR.cmd            spark-submit
find-spark-home.cmd    run-example.cmd       spark-shell            spark-submit2.cmd
load-spark-env.cmd     spark-class            spark-shell2.cmd      spark-submit.cmd
load-spark-env.sh      spark-class2.cmd      spark-shell.cmd
hadoop@nodo1:/opt/hadoop/spark/bin$
```

4. INSTALACION DE SPARK

Paso 5. Dentro del directorio conf se guardan las templates o plantillas de configuración, las cuales tenemos que modificar para adaptarlas a nuestras necesidades dependiendo del producto. Uno de ellos es el fichero workers para definir los esclavos que se van a utilizar

```
127 hadoop@nodo1:/opt/hadoop/spark/conf$ ls -l
total 36
-rw-r--r-- 1 hadoop hadoop 1105 abr  9 20:58 fairscheduler.xml.template
-rw-r--r-- 1 hadoop hadoop 2471 abr  9 20:58 log4j.properties.template
-rw-r--r-- 1 hadoop hadoop 9141 abr  9 20:58 metrics.properties.template
-rw-r--r-- 1 hadoop hadoop 1292 abr  9 20:58 spark-defaults.conf.template
-rwxr-xr-x 1 hadoop hadoop 4428 abr  9 20:58 spark-env.sh.template
-rw-r--r-- 1 hadoop hadoop  865 abr  9 20:58 workers.template
hadoop@nodo1:/opt/hadoop/spark/conf$
```


4. INSTALACION DE SPARK

Paso 6. El directorio sbin es muy interesante, ya que contiene una serie de scripts (starts, stops, etc) que nos permiten arrancar y parar los distintos servicios de Spark dependiendo del modo en que lo estemos utilizando

```
hadoop@nodo1:/opt/hadoop/spark/sbin$ ls
decommission-slave.sh      start-mesos-dispatcher.sh    stop-master.sh
decommission-worker.sh    start-mesos-shuffle-service.sh stop-mesos-dispatcher.sh
slaves.sh                 start-slave.sh               stop-mesos-shuffle-service.sh
spark-config.sh           start-slaves.sh              stop-slave.sh
spark-daemon.sh           start-thriftserver.sh        stop-slaves.sh
spark-daemons.sh         start-worker.sh              stop-thriftserver.sh
start-all.sh             start-workers.sh             stop-worker.sh
start-history-server.sh   stop-all.sh                 stop-workers.sh
start-master.sh           stop-history-server.sh       workers.sh
hadoop@nodo1:/opt/hadoop/spark/sbin$
```

4. INSTALACION DE SPARK

Paso 7. En /home/hadoop editamos el fichero de arranque .bashrc, para crear la variable de entorno SPARK_DIST_CLASSPATH y agregar en PATH los directorios bin y sbin de Spark, de manera que esto nos permita encontrar los binarios y los scripts de arranque y parada de Spark.

```
GNU nano 6.2                                .bashrc
if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi
fi
[ -r /home/hadoop/.config/byobu/prompt ] && . /home/hadoop/.config/byobu/prompt

export HADOOP_HOME=/opt/hadoop
export HIVE_HOME=/opt/hadoop/hive
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$HIVE_HOME/bin

export PATH=$PATH:/opt/hadoop/spark/bin:/opt/hadoop/spark/sbin
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
```

4. INSTALACION DE SPARK

Paso 8. La variable de entorno SPARK_DIST_CLASSPATH se define mediante \$(hadoop classpath). \$() permite que **hadoop classpath** se vea como un comando, cuya ejecución devuelve todos los paths del entorno hadoop. Eso es precisamente lo que se esta asociando a la variable SPARK_DIST_CLASSPATH

```
hadoop@nodol1:~$ hadoop classpath
/opt/hadoop/etc/hadoop:/opt/hadoop/share/hadoop/common/lib/*:/opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/hdfs:/opt/hadoop/share/hadoop/hdfs/lib/*:/opt/hadoop/share/hadoop/hdfs/*:/opt/hadoop/share/hadoop/mapreduce/lib/*:/opt/hadoop/share/hadoop/mapreduce/*:/opt/hadoop/share/hadoop/yarn:/opt/hadoop/share/hadoop/yarn/lib/*:/opt/hadoop/share/hadoop/yarn/*
hadoop@nodol1:~$
```

Paso 9. Para recargar las nuevas variables de entorno de .bashrc:

```
hadoop@nodol1:~$ source .bashrc
hadoop@nodol1:~$ . ./bashrc
hadoop@nodol1:~$ echo $SPARK_DIST_CLASSPATH
/opt/hadoop/etc/hadoop:/opt/hadoop/share/hadoop/common/lib/*:/opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/hdfs:/opt/hadoop/share/hadoop/hdfs/lib/*:/opt/hadoop/share/hadoop/hdfs/*:/opt/hadoop/share/hadoop/mapreduce/lib/*:/opt/hadoop/share/hadoop/mapreduce/*:/opt/hadoop/share/hadoop/yarn:/opt/hadoop/share/hadoop/yarn/lib/*:/opt/hadoop/share/hadoop/yarn/*
hadoop@nodol1:~$ █
```

5. PROBAR SPARK EN MODO CLIENTE

Paso 1. Comprobaremos el funcionamiento de Spark en modo cliente. Todavía no lo vamos a unir a un clúster ni vamos a arrancarlo en modo standAlone. Sólo veremos un par de herramientas spark-Shell y pyspark, que permiten:

- ❖ probar si el entorno de Spark funciona
- ❖ hacer pruebas con el lenguaje sin necesidad de cluster, trabajando con MV pequeñas para probar los comandos

Spark permite trabajar con varios lenguajes de programación, entre los que destacan Scala y Python. Los programas spark-shell y pyspark permiten probar comandos de scala y Python respectivamente. Ambas se encuentran en /opt/hadoop/spark/bin

```
hadoop@nodo1:/opt/hadoop/spark/bin$ ls
beeline                pyspark                spark-class.cmd       spark-sql
beeline.cmd            pyspark2.cmd           sparkR                 spark-sql2.cmd
docker-image-tool.sh   pyspark.cmd            sparkR2.cmd           spark-sql.cmd
find-spark-home        run-example            sparkR.cmd             spark-submit
find-spark-home.cmd    run-example.cmd        spark-shell            spark-submit2.cmd
load-spark-env.cmd     spark-class            spark-shell2.cmd       spark-submit.cmd
load-spark-env.sh      spark-class2.cmd       spark-shell.cmd
hadoop@nodo1:/opt/hadoop/spark/bin$
```

5. PROBAR SPARK EN MODO CLIENTE

Paso 2. La herramienta spark-shell nos permite programar en Scala. Si ejecutamos **spark-shell** (ya esta puesto en el path) se crea un pequeño contexto local para poder trabajar, el cual llama al entorno de línea de comandos de Scala

De forma transparente crea el componente `sparkContext` que apunta al proceso local iniciado. El programa Driver necesita este componente llamado `sc` para acceder a los recursos de Spark

```
2023-07-18 15:06:05,705 INFO ReplicatorMain: Created Spark session
Spark context Web UI available at http://nod01:4040
Spark context available as 'sc' (master = local[*], app id = local-1681650363147).
Spark session available as 'spark'.
Welcome to

      /_/_/ \_/_/ \_/_/ \_/_/
     /  V  V  V  V  V  V  V
    /___/ . _\ \ , /___/ /___/ \
   /___/          /___/ \_/_/
  /___/

version 3.2.4

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_352)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

5. PROBAR SPARK EN MODO CLIENTE

Paso 3. El primer comando para empezar a probar en scala es por ejemplo el comando **:help** de ayuda en scala. Muchos comandos de Scala a nivel de Shell, comienzan con : (dos puntos)

```
type in expressions to have them evaluated.
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:completions <string>      output completions for the given string
:edit <id>|<line>          edit history
:h? <string>               search the history
:imports [name name ...]  show import history, identifying sources of names
:implicits [-v]            show the implicits in scope
:javap <path|class>        disassemble a file or class name
:line <id>|<line>          place line(s) at the end of history
:load <path>               interpret lines in a file
:paste [-raw] [path]       enter paste mode or paste a file
:power                     enable power user mode
```

5. PROBAR SPARK EN MODO CLIENTE

Paso 4. Vamos a hacer un pequeño programa en Scala que cuente el nº de líneas del fichero de texto README.md que viene con el propio Spark. Creamos una variable en scala llamada fichero, asociada a ese fichero:

```
scala> val fichero=sc.textFile("file:///opt/hadoop/spark/README.md")
2023-04-16 14:03:50,112 INFO memory.MemoryStore: Block broadcast_0 stored as values in memo
ry (estimated size 435.7 KiB, free 365.9 MiB)
2023-04-16 14:03:50,143 INFO memory.MemoryStore: Block broadcast_0_piece0 stored as bytes i
n memory (estimated size 44.2 KiB, free 365.8 MiB)
2023-04-16 14:03:50,145 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory o
n nodol:40363 (size: 44.2 KiB, free: 366.3 MiB)
2023-04-16 14:03:50,148 INFO spark.SparkContext: Created broadcast 0 from textFile at <cons
ole>:23
fichero: org.apache.spark.rdd.RDD[String] = file:///opt/hadoop/spark/README.md MapPartition
sRDD[1] at textFile at <console>:23

scala>
```

Con la clausula file accedemos a un fichero en modo local, no a un fichero de HDFS, ni a una RDD de Spark. Asocia el fichero README.md a la variable Scala fichero.

5. PROBAR SPARK EN MODO CLIENTE

Paso 5. Si ejecutamos la variable fichero, Scala devuelve un texto indicando que es un objeto tipo RDD string que está apuntando al fichero README.md.

```
scala> fichero
res0: org.apache.spark.rdd.RDD[String] = file:///opt/hadoop/spark/README.md MapPartitionsRDD[1] at textFile at <console>:23
```

Paso 6. Si ponemos fichero.count(), este método asociado nos retorna el número de líneas que tiene este fichero

```
2023-04-16 14:17:49,442 INFO scheduler.DAGScheduler: Job 0 finished: count at <console>:24, took 0,373838 s
res1: Long = 109
scala> █
```

Paso 7. Para salir hacemos :q

```
scala> :q
2023-04-16 14:34:31,481 INFO server.AbstractConnector: Stopped Spark@4ea48b2e{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
2023-04-16 14:34:31,483 INFO ui.SparkUI: Stopped Spark web UI at http://nod01:4040
2023-04-16 14:34:31,492 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```


5. PROBAR SPARK EN MODO CLIENTE

Paso 8. Ahora veremos el mismo ejemplo pero hecho con Python o pyspark: Leeremos el fichero y contaremos el nº de líneas que tiene. Ejecutamos pyspark

```
675c30c{/metrics/json,null,AVAILABLE,@Spark}
Welcome to

      /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
     /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
    /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
   /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
  /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
 /_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/
/_/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/ _/_/_\_/

version 3.2.4

Using Python version 3.10.6 (main, Nov 14 2022 16:10:14)
Spark context Web UI available at http://nodol1:4040
Spark context available as 'sc' (master = local[*], app id = local-1681655842933).
SparkSession available as 'spark'.
>>>
>>>
```

Crea un entorno para trabajar dentro de la shell de Python. A diferencia de spark-Shell que llamaba `sc` al `sparkcontext`, aquí a la sesión la llama `spark`

5. PROBAR SPARK EN MODO CLIENTE

Paso 9. Creamos una variable fichero en Python que vamos a asociar al fichero Readme, usando el contexto spark y la función read.text

```
>>> fichero=spark.read.text("file:///opt/hadoop/spark/README.md")
2023-04-16 14:59:16,381 INFO datasources.InMemoryFileIndex: It took 0 ms to list leaf files
for 1 paths.
>>>
```

Paso 10. Si ejecutamos fichero vemos que se trata de un DataFrame de tipo string

```
>>> fichero
DataFrame[value: string]
>>>
```

Paso 11. Contamos el numero de líneas con el método count

```
2023-04-16 15:06:16,934 INFO schedule
ccessorImpl.java:0, took 0,019071 s
109
>>>
```

Paso 12. Para salir ejecutamos quit()

```
>>> quit()
2023-04-17 16:31:30,659 INFO server.AbstractConnector: Stopped Spark@4dafb456{HTTP/1
.1, (http/1.1)}{0.0.0.0:4040}
2023-04-17 16:31:30,662 INFO ui.SparkUI: Stopped Spark web UI at http://nod01:4040
2023-04-17 16:31:30,672 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerM
asterEndpoint stopped!
```

6. INTEGRACION DE SPARK CON HADOOP

Paso 1. Veremos cómo podemos integrar nuestro Spark dentro del cluster de hadoop, para poder empezar a lanzar comandos y acceder a HDFS. Posteriormente veremos como lanzar Jobs o procesos.

Tenemos que editar de nuevo el fichero de entorno .bashrc, para indicar dos variables de entorno: SPARK_HOME (donde se encuentra el home de spark) y HADOOP_CONF_DIR (que apunta al directorio donde están todos los ficheros de configuración de Hadoop, yarn-site.xml, hfs-site.xml, etc)

```
export HADOOP_HOME=/opt/hadoop
export HIVE_HOME=/opt/hadoop/hive
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$HIVE_HOME/bin

export PATH=$PATH:/opt/hadoop/spark/bin:/opt/hadoop/spark/sbin
export SPARK_DIST_CLASSPATH=$(hadoop classpath)

export SPARK_HOME=/opt/hadoop/spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

6. INTEGRACION DE SPARK CON HADOOP

Paso 2. Para recargar el fichero de entorno podemos salir y entrar en la Shell o ejecutar dos comandos diferentes: `source .bashrc` o `./bashrc`

```
hadoop@nodol1:~$ source .bashrc
hadoop@nodol1:~$ . ./bashrc
hadoop@nodol1:~$ echo $HADOOP_CONF_DIR
/opt/hadoop/etc/hadoop
hadoop@nodol1:~$
```

Paso 3. Dentro de HDFS creamos un directorio llamado spark (el nombre no es importante), donde guardaremos los ficheros de datos.

```
hadoop@nodol1:/datos$ hdfs dfs -ls /
hadoop@nodol1:/datos$ hdfs dfs -mkdir /spark
hadoop@nodol1:/datos$
```

Paso 4. Descargamos el fichero `puertos.csv` y lo subimos al directorio HDFS `/spark` de Hadoop. Se trata de información de los puertos marítimos de España (descargas, nº barcos, etc). Leeremos este fichero y haremos algunas operaciones sobre el mismo.

```
hadoop@nodol1:~/Downloads$ hdfs dfs -put puertos.csv /spark
hadoop@nodol1:~/Downloads$ hdfs dfs -ls /spark
Found 1 items
-rw-r--r--    2 hadoop supergroup    112636 2023-04-17 16:38 /spark/puertos.csv
hadoop@nodol1:~/Downloads$
```

6. INTEGRACION DE SPARK CON HADOOP

Paso 5. En una nueva pestaña, abrimos una sesión de spark-shell para conectarnos al entorno de Spark. Desde Scala accederemos al fichero subido anteriormente a HDFS, para trabajar con el

```

      / _ \   / _ \   / _ \   / _ \   / _ \
     / _ \ / _ \ / _ \ / _ \ / _ \
    / _ \| _ \| | _ \| | _ \| | _ \|
   / _ \| |_) | | |_) | | |_) | | |_) |
  / _ \|___|_|_||___|_|_||___|_|_||___|
                                     version 3.2.4

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_352)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

```

Paso 6. Creamos una variable v1 que apunta al fichero puertos.csv de HDFS

```
scala> val v1=sc.textFile("/spark/puertos.csv")
2023-04-17 17:39:25,448 INFO memory.MemoryStore: Block broadcast_0 stored as values
in memory (estimated size 435.7 KiB, free 365.9 MiB)
2023-04-17 17:39:25,486 INFO memory.MemoryStore: Block broadcast_0_piece0 stored as
bytes in memory (estimated size 44.2 KiB, free 365.8 MiB)
2023-04-17 17:39:25,488 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in m
emory on nodol:35587 (size: 44.2 KiB, free: 366.3 MiB)
2023-04-17 17:39:25,492 INFO spark.SparkContext: Created broadcast 0 from textFile a
t <console>:23
v1: org.apache.spark.rdd.RDD[String] = /spark/puertos.csv MapPartitionsRDD[1] at tex
tFile at <console>:23

scala>
```

6. INTEGRACION DE SPARK CON HADOOP

Paso 7. La variable v1 se trata de un RDD de tipo string que apunta puertos.csv

```
scala> v1
res0: org.apache.spark.rdd.RDD[String] = /spark/puertos.csv MapPartitionsRDD[1] at t
extFile at <console>:23

scala> █
```

Paso 8. Contamos el número de líneas que tiene este fichero, 1374 líneas

```
scala> v1.count()
2023-04-17 18:01:03,168 INFO mapred.FileInputFormat: Total input files to process :
1
2023-04-17 18:01:03,244 INFO spark.SparkContext: Starting job: count at <console>:24
2023-04-17 18:01:03,262 INFO scheduler.DAGScheduler: Got job 0 (count at <console>:2
4) with 2 output partitions
```

```
le>:24, took 0,533961 s
res1: Long = 1374

scala> █
```

6. INTEGRACION DE SPARK CON HADOOP

Paso 9. Ahora a partir de este RDD vamos a crear un nuevo RDD, filtrando las líneas en las que aparece la palabra Barcelona. Para eso utilizaremos un comando de Spark denominado Filter

```
scala> val v2=v1.filter(l => l contains "Barcelona")
v2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:23
```

Filter distingue entre mayúsculas y minúsculas. Crea un nuevo RDD formado por todas aquellas líneas que contienen la palabra Barcelona.

Paso 10. Si ejecutamos v2.count() debería indicar que hay 48 líneas con la palabra Barcelona

```
scala> v2.count()
2023-04-17 19:17:18,635 INFO spark.SparkContext: Starting job: count at <console>:24
2023-04-17 19:17:18,642 INFO scheduler.DAGScheduler: Got job 1 (count at <console>:24) with
2 output partitions
```

```
2023-04-17 19:17:18,753 INFO scheduler.DAGScheduler: Job 1 finished: count at <console>:24,
took 0,117975 s
res2: Long = 48
```

```
scala> █
```

6. INTEGRACION DE SPARK CON HADOOP

NOTA. La integración de Spark con un cluster Hadoop es muy sencilla. Configurando solo un par de variables de entorno, desde el cliente spark-shell. ya tenemos acceso al entorno HDFS Hadoop, Spark-shell va muy bien hacer pruebas, pero no es la situación ideal. Lo ideal es poder lanzar trabajos Spark (en Java, Scala o Python) contra el cluster hadoop, para que se puedan utilizar todas las características de ese cluster: nodos, recursos, memoria, etc. Esto se hace con el comando **spark-submit** que veremos a continuación

7. LANZAR PROGRAMA SCALA CONTRA YARN

Paso 1. Para lanzar una aplicación o proceso Spark (en Java, Scala o Python) contra un clúster de tipo Hadoop, Mesos o standalone, tenemos que utilizar el comando **spark-submit**

Lanzaremos uno de los ejemplos que tenemos dentro del propio Spark

```
hadoop@nodol1:~$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name "apli1" /opt/hadoop/spark/examples/jars/spark-examples_2.12-3.2.4.jar 5
```

spark-submit + nombre programa + resto argumentos

- a) **--class org.apache.spark.examples.SparkPi** → Programa en spark (en Java, Scala, Python) que está dentro del fichero de ejemplos incluido en la distribución de Spark. En este caso Sparkpi realiza el calculo del número Pi.
- b) **--master yarn** → Indicamos que vamos a ejecutar una aplicación Spark contra un clúster Hadoop de tipo Yarn

7. LANZAR PROGRAMA SCALA CONTRA YARN

- c) **--deploy-mode cluster** → Indicamos que se ejecute dentro de todos los nodos del clúster. En caso contrario intenta trabajar en modo cliente, de forma local desde la máquina donde se lanzó
- d) **--name "apli1"** → Ponemos un nombre a nuestra aplicación, para que luego podamos reconocerla cuando se ejecuta
- e) **/opt/hadoop/spark/examples/jars/spark-examples_2.12-3.2.4.jar** → Indica el jar donde se encuentra la clase SparkPi
- f) **5** → Cuanto mayor este numero, mas tiempo tarda en calcular, más operaciones hace y más puedo probar mi clase o programa.

7. LANZAR PROGRAMA SCALA CONTRA YARN

Paso 2. Lanzamos la aplicación y observamos los resultados que salen:

- a) Debemos de ver que se va a conectar al ResourceManager del clúster, en este caso por el nodo1, desde donde se ha lanzado

```
hadoop@nodo1:~$ spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --name "aplil" /opt/hadoop/spark/
examples/jars/spark-examples_2.12-3.2.4.jar 5
2023-04-18 08:07:50,611 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes whe
re applicable
2023-04-18 08:07:50,661 INFO client.RMProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032
2023-04-18 08:07:50,884 INFO yarn.Client: Requesting a new application from cluster with 2 NodeManagers
2023-04-18 08:07:51,254 INFO conf.Configuration: resource-types.xml not found
```

- b) A continuación indica que ha sido aceptada (ACCEPTED) y muestra la URL de tracking para hacer el seguimiento de su ejecución.

```
2023-04-18 08:07:53,626 INFO impl.YarnClientImpl: Submitted application application_1681800749719_0002
2023-04-18 08:07:54,629 INFO yarn.Client: Application report for application_1681800749719_0002 (state: ACCEPTED)
2023-04-18 08:07:54,632 INFO yarn.Client:
  client token: N/A
  diagnostics: AM container is launched, waiting for AM container to Register with RM
  ApplicationMaster host: N/A
  ApplicationMaster RPC port: -1
  queue: default
  start time: 1681805273605
  final status: UNDEFINED
  tracking URL: http://nodo1:8088/proxy/application_1681800749719_0002/
  user: hadoop
```

7. LANZAR PROGRAMA SCALA CONTRA YARN

- c) Cuando ya ha pasado la cola y ha sido aceptada completamente se pone en estado RUNNING. Cuando acaba se pone a FINISH

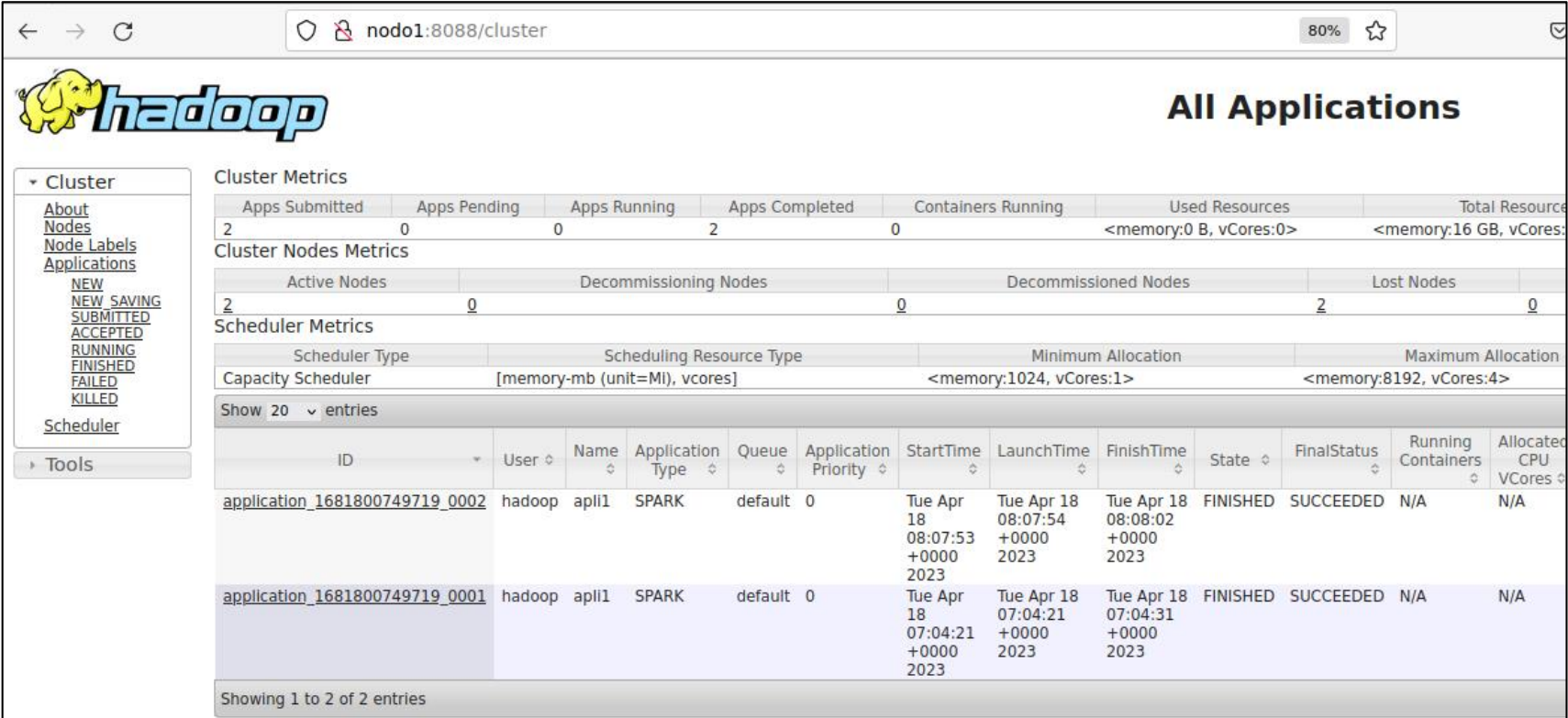
```
2023-04-18 08:07:55,635 INFO yarn.Client: Application report for application_1681800749719_0002 (state: ACCEPTED)
2023-04-18 08:07:56,637 INFO yarn.Client: Application report for application_1681800749719_0002 (state: ACCEPTED)
2023-04-18 08:07:57,640 INFO yarn.Client: Application report for application_1681800749719_0002 (state: RUNNING)
2023-04-18 08:07:57,640 INFO yarn.Client:
    client token: N/A
    diagnostics: N/A
    ApplicationMaster host: nodo2
    ApplicationMaster RPC port: 38245
    queue: default
    start time: 1681805273605
    final status: UNDEFINED
    tracking URL: http://nodo1:8088/proxy/application_1681800749719_0002/
    user: hadoop
2023-04-18 08:07:58,643 INFO yarn.Client: Application report for application_1681800749719_0002 (state: RUNNING)
2023-04-18 08:07:59,646 INFO yarn.Client: Application report for application_1681800749719_0002 (state: RUNNING)
2023-04-18 08:08:00,652 INFO yarn.Client: Application report for application_1681800749719_0002 (state: RUNNING)
2023-04-18 08:08:01,655 INFO yarn.Client: Application report for application_1681800749719_0002 (state: RUNNING)
2023-04-18 08:08:02,658 INFO yarn.Client: Application report for application_1681800749719_0002 (state: FINISHED)
```

- d) Su estado final es SUCCEEDED, se ha ejecutado correctamente .

```
2023-04-18 08:08:02,659 INFO yarn.Client:
    client token: N/A
    diagnostics: N/A
    ApplicationMaster host: nodo2
    ApplicationMaster RPC port: 38245
    queue: default
    start time: 1681805273605
    final status: SUCCEEDED
    tracking URL: http://nodo1:8088/proxy/application_1681800749719_0002/
    user: hadoop
2023-04-18 08:08:02,667 INFO util.ShutdownHookManager: Shutdown hook called
2023-04-18 08:08:02,667 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-8a3afc65-14cf-46d8-836f-5f37076af348
2023-04-18 08:08:02,671 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-78243011-a432-4bd0-b163-1c3567ad13dd
hadoop@nodo1:~$
```

7. LANZAR PROGRAMA SCALA CONTRA YARN

Paso 3. Podemos ir a la pagina web de gestión de Hadoop, nodo1:8088 para ver como se ha ejecutado esta aplicación Spark



The screenshot displays the Hadoop cluster management web interface. The browser address bar shows 'nodo1:8088/cluster'. The page title is 'All Applications'. The sidebar on the left contains a 'Cluster' section with links for 'About', 'Nodes', 'Node Labels', 'Applications', and a status filter menu with options: 'NEW', 'NEW SAVING', 'SUBMITTED', 'ACCEPTED', 'RUNNING', 'FINISHED', 'FAILED', 'KILLED', and 'Scheduler'. Below the sidebar, the 'Cluster Metrics' section shows a summary of application states: Apps Submitted (2), Apps Pending (0), Apps Running (0), Apps Completed (2), Containers Running (0), Used Resources (<memory:0 B, vCores:0>), and Total Resource (<memory:16 GB, vCores:16>). The 'Cluster Nodes Metrics' section shows Active Nodes (2), Decommissioning Nodes (0), Decommissioned Nodes (0), and Lost Nodes (2). The 'Scheduler Metrics' section shows the Scheduler Type (Capacity Scheduler), Scheduling Resource Type ([memory-mb (unit=Mi), vcores]), Minimum Allocation (<memory:1024, vCores:1>), and Maximum Allocation (<memory:8192, vCores:4>). The 'Show 20 entries' section displays a table of application details.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores
application_1681800749719_0002	hadoop	apl11	SPARK	default	0	Tue Apr 18 08:07:53 +0000 2023	Tue Apr 18 08:07:54 +0000 2023	Tue Apr 18 08:08:02 +0000 2023	FINISHED	SUCCEEDED	N/A	N/A
application_1681800749719_0001	hadoop	apl11	SPARK	default	0	Tue Apr 18 07:04:21 +0000 2023	Tue Apr 18 07:04:21 +0000 2023	Tue Apr 18 07:04:31 +0000 2023	FINISHED	SUCCEEDED	N/A	N/A

Showing 1 to 2 of 2 entries

7. LANZAR PROGRAMA SCALA CONTRA YARN

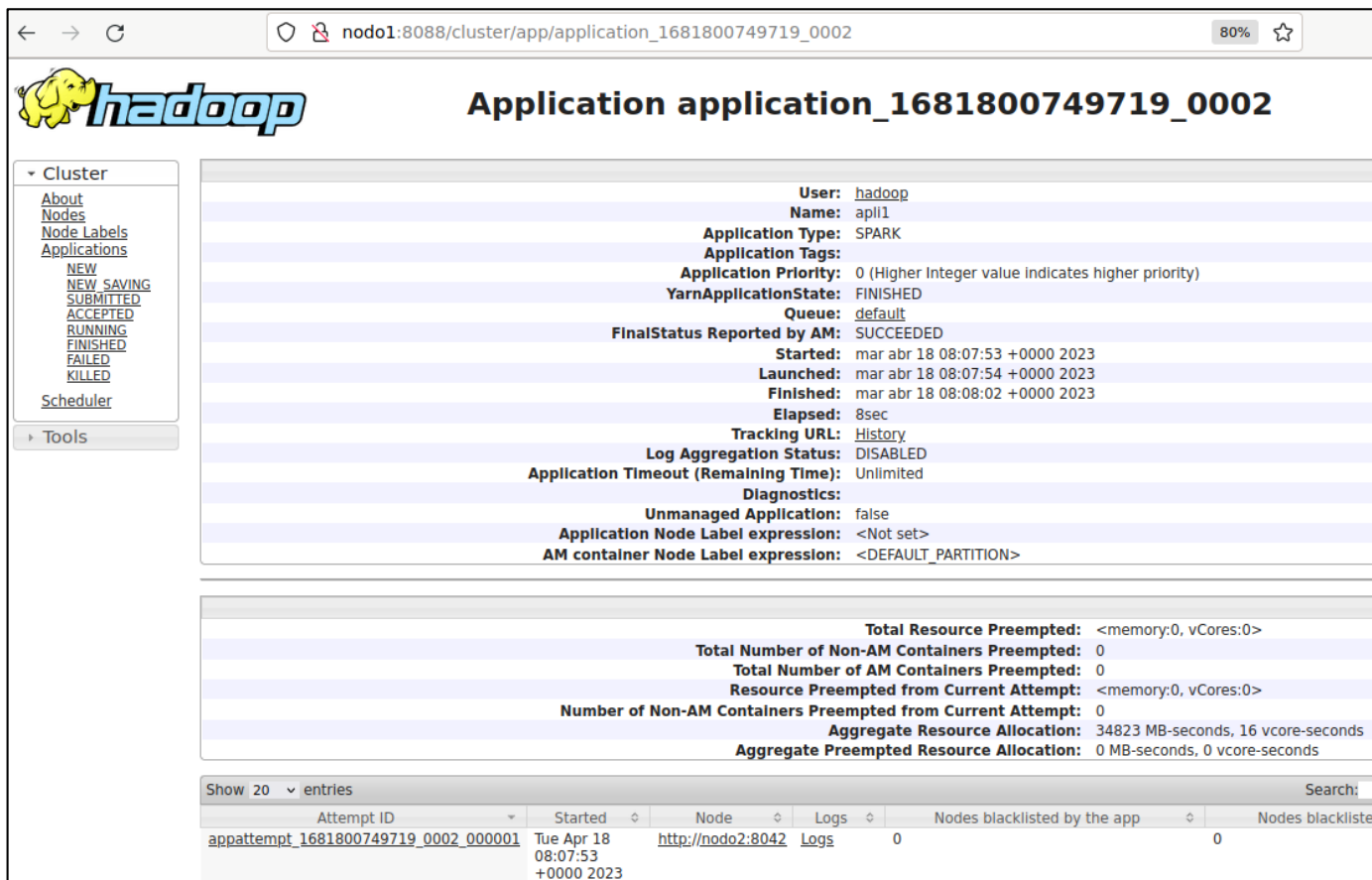
Paso 4. Vemos que el usuario con el que se ha lanzado es **hadoop**, el nombre que le hemos puesto a la aplicación es **apli1**, el tipo de aplicación es **Spark** y que ha terminado de manera satisfactoria

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers
application_1681800749719_0002	hadoop	apli1	SPARK	default	0	Tue Apr 18 08:07:53 +0000 2023	Tue Apr 18 08:07:54 +0000 2023	Tue Apr 18 08:08:02 +0000 2023	FINISHED	SUCCEEDED	N/A

Vemos que una vez que se lanza una aplicación Spark dentro del cluster Hadoop, se aprovechan todas las características de este entorno y podemos ver su ejecución a través de esta página web.

7. LANZAR PROGRAMA SCALA CONTRA YARN

Paso 5. Si hacemos click en ApplicationID, vemos diferentes informaciones: recursos que ha utilizado la aplicación, lo que ha tardado y más abajo el nodo2 donde se ha realizado la ejecución.



The screenshot displays the Hadoop YARN web interface. The browser address bar shows the URL: `nodo1:8088/cluster/app/application_1681800749719_0002`. The page title is "Application application_1681800749719_0002".

Left Sidebar:

- Cluster
 - About
 - Nodes
 - Node Labels
 - Applications
 - NEW
 - NEW SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
 - Scheduler
- Tools

Main Content Area:

User: `hadoop`
Name: `apl11`
Application Type: `SPARK`
Application Tags:
Application Priority: `0` (Higher Integer value Indicates higher priority)
YarnApplicationState: `FINISHED`
Queue: `default`
FinalStatus Reported by AM: `SUCCEEDED`
Started: `mar abr 18 08:07:53 +0000 2023`
Launched: `mar abr 18 08:07:54 +0000 2023`
Finished: `mar abr 18 08:08:02 +0000 2023`
Elapsed: `8sec`
Tracking URL: `History`
Log Aggregation Status: `DISABLED`
Application Timeout (Remaining Time): `Unlimited`
Diagnostics:
Unmanaged Application: `false`
Application Node Label expression: `<Not set>`
AM container Node Label expression: `<DEFAULT_PARTITION>`

Resource Usage:

- Total Resource Preempted:** `<memory:0, vCores:0>`
- Total Number of Non-AM Containers Preempted:** `0`
- Total Number of AM Containers Preempted:** `0`
- Resource Preempted from Current Attempt:** `<memory:0, vCores:0>`
- Number of Non-AM Containers Preempted from Current Attempt:** `0`
- Aggregate Resource Allocation:** `34823 MB-seconds, 16 vcore-seconds`
- Aggregate Preempted Resource Allocation:** `0 MB-seconds, 0 vcore-seconds`


Table:

Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted
appattempt_1681800749719_0002_000001	Tue Apr 18 08:07:53 +0000 2023	http://nodo2:8042	Logs	0	0

7. LANZAR PROGRAMA SCALA CONTRA YARN

Paso 6. Para comprobar lo que ha hecho la aplicación, ya que no ha sacado nada por pantalla, vamos a los logs del nodo2. Dentro veremos todo lo que ha generado este contenedor con esta aplicación.

En el enlace stdout (salida estándar) podemos ver el texto que produce la salida de esta aplicación sobre el numero pi



hadoop Logs for container_1

▼ ResourceManager

RM Home

Local Logs:

▼ NodeManager

Tools

[directory.info](#) : Total file length is 26145 bytes.

[launch_container.sh](#) : Total file length is 5912 bytes.

[prelaunch.err](#) : Total file length is 0 bytes.

[prelaunch.out](#) : Total file length is 100 bytes.

[stderr](#) : Total file length is 24165 bytes.

[stdout](#) : Total file length is 32 bytes.



hadoop Logs for

▼ ResourceManager

RM Home

Pi is roughly 3.145542291084582

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 1. Vamos a lanzar una aplicación Spark en Python (incluyendo el código fuente) contra nuestro clúster Hadoop, usando el comando spark-submit. Descargamos dos ficheros:

- ContarPalabras.py → código en Python que cuenta las palabras
- quijote.txt → archivo con el contenido del quijote

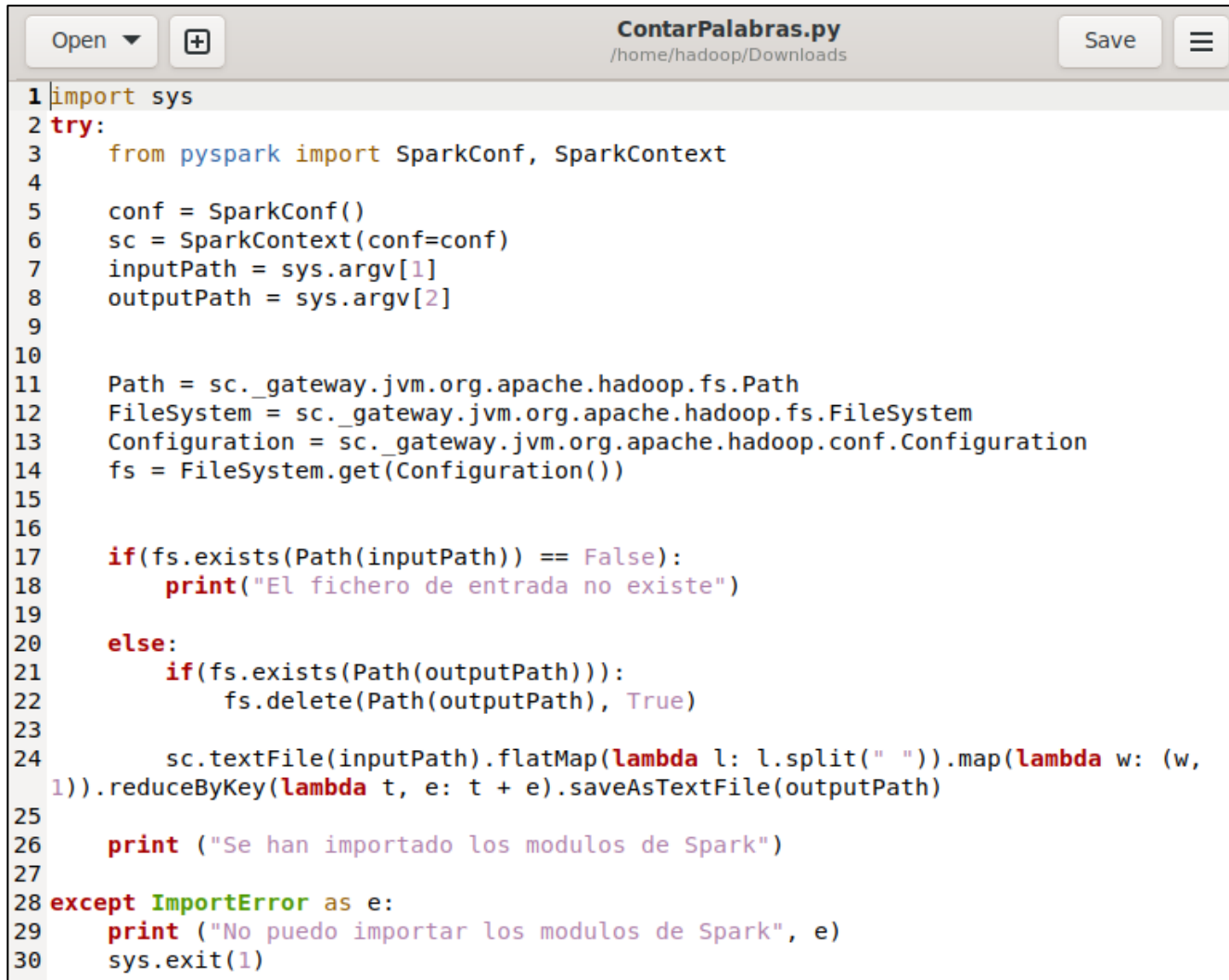
```
hadoop@nodol1:~/Downloads$ ls -l
total 687836
-rw-rw-r-- 1 hadoop hadoop      932 abr 18 10:17 ContarPalabras.py
-rw-rw-r-- 1 hadoop hadoop 492368219 feb 23 19:10 hadoop-3.2.4.tar.gz
-rw-rw-r-- 1 hadoop hadoop    112636 abr 16 21:22 puertos.csv
-rw-rw-r-- 1 hadoop hadoop    2198927 abr 18 11:31 quijote.txt
-rw-rw-r-- 1 hadoop hadoop 209645343 abr 17 13:20 spark-3.2.4-bin-without-hadoop.tgz
hadoop@nodol1:~/Downloads$
```

Paso 2. Subimos el fichero quijote.txt al directorio HDFS /practicass

```
hadoop@nodol1:~/Downloads$ hdfs dfs -mkdir /practicass
hadoop@nodol1:~/Downloads$ hdfs dfs -put quijote.txt /practicass
hadoop@nodol1:~/Downloads$ hdfs dfs -ls /practicass
Found 1 items
-rw-r--r--  2 hadoop supergroup    2198927 2023-04-18 11:35 /practicass/quijote.txt
hadoop@nodol1:~/Downloads$
```

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 3. Comentamos un poco el código en python

A screenshot of a code editor window titled 'ContarPalabras.py' with the file path '/home/hadoop/Downloads'. The editor contains a Python script that uses PySpark to count words in a file. The script includes imports for sys, SparkConf, and SparkContext, followed by the creation of a SparkContext and the loading of Hadoop filesystem classes. It then checks for the existence of input and output paths, deletes the output path if it exists, and finally runs a word count operation using flatMap, map, and reduceByKey. The script ends with an exception handler for ImportError and a sys.exit(1) call.

```
1 import sys
2 try:
3     from pyspark import SparkConf, SparkContext
4
5     conf = SparkConf()
6     sc = SparkContext(conf=conf)
7     inputPath = sys.argv[1]
8     outputPath = sys.argv[2]
9
10
11     Path = sc._gateway.jvm.org.apache.hadoop.fs.Path
12     FileSystem = sc._gateway.jvm.org.apache.hadoop.fs.FileSystem
13     Configuration = sc._gateway.jvm.org.apache.hadoop.conf.Configuration
14     fs = FileSystem.get(Configuration())
15
16
17     if(fs.exists(Path(inputPath)) == False):
18         print("El fichero de entrada no existe")
19
20     else:
21         if(fs.exists(Path(outputPath))):
22             fs.delete(Path(outputPath), True)
23
24         sc.textFile(inputPath).flatMap(lambda l: l.split(" ")).map(lambda w: (w,
25 1)).reduceByKey(lambda t, e: t + e).saveAsTextFile(outputPath)
26
27     print ("Se han importado los modulos de Spark")
28 except ImportError as e:
29     print ("No puedo importar los modulos de Spark", e)
30     sys.exit(1)
```

8. LANZAR PROGRAMA PYTHON CONTRA YARN

- ❖ `import sys` → importamos librerías de sistema de Python
- ❖ `import SparkConf` y `SparkContext` → Importamos las librerías que nos permiten conectar con Spark y definir su contexto `sc`
- ❖ Se definen dos variables de cadena que van a recoger los parámetros de entrada de la línea de comandos: el fichero que se quieren contar palabras y el directorio donde dejar los resultados
- ❖ Se definen una serie de variables que permiten acceder al sistema de ficheros HDFS (`Path`, `FileSystem`)
- ❖ Si el fichero que he puesto en la entrada al llamar el programa no existe, da un error. Si existe sigue hacia adelante.
- ❖ Con `textFile` se lee el fichero de entrada, y se le aplica una serie de procesados similar al wordcount de Java: `flatMap`, `map` y `reduceByKey` dividen el contenido del fichero en palabras usando los espacios en blanco y luego va contando las ocurrencias. El resultado lo deja en el directorio de salida.

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 4. Lanzaremos este programa en pyspark contra el clúster Hadoop, usando el comando spark-submit.

```
hadoop@nodol1:~/Downloads$ spark-submit --master yarn --deploy-mode cluster --name "ContarPalabras" ContarPalabras.py /practicass/quijote.txt /salida_spark_wc
```

`--master [yarn/local]` → permite indicar contra qué tipo de infraestructura vamos a lanzar nuestro programa Spark, contra un cluster de tipo Yarn Hadoop o en modo local (ejecución en modo cliente parecido a lo que hemos hecho con spark-Shell y/o pyspark

`--deploy-mode [cluster/client]` → Indica que use todo el clúster. Por defecto es modo client, que quiere decir que se ejecuta todo en local.

`--name "ContarPalabras"` → Nombre del programa para su localización

`ContarPalabras.py` → el programa en Python que cuenta palabras

`/practicass /quijote.txt` → Fichero del que tiene que contar palabras

`/salida_spark_wc` → Directorio HDFS donde dejar los resultados. No debe de existir porque sino da un error.

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 5. Una vez lanzado es importante comprobar que efectivamente se ha conectado al ResourceManager del nodo1 y al cluster yarn. También indica que se trata de un clúster con 2 nodemanagers.

```
hadoop@nodo1:~/Downloads$ spark-submit --master yarn --deploy-mode cluster --name "ContarPalabras" ContarPalabras.py  
/practicass/quijote.txt /salida_spark_wc  
2023-04-18 11:45:02,753 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using b  
uilt-in java classes where applicable  
2023-04-18 11:45:02,810 INFO client.RMPProxy: Connecting to ResourceManager at nodo1/192.168.0.101:8032  
2023-04-18 11:45:03,028 INFO yarn.Client: Requesting a new application from cluster with 2 NodeManagers  
2023-04-18 11:45:03,365 INFO conf.Configuration: resource-types.xml not found  
2023-04-18 11:45:03,366 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
```

Sube ciertos recursos como pyspark.zip que son necesarios y acaba indicando que la tarea ha sido aceptada

```
2023-04-18 11:45:05,278 INFO yarn.Client: Uploading resource file:/opt/hadoop/spark/python/lib/pyspark.zip -> hdfs://  
nodo1:9000/user/hadoop/.sparkStaging/application_1681817673583_0002/pyspark.zip  
2023-04-18 11:45:05,313 INFO yarn.Client: Uploading resource file:/opt/hadoop/spark/python/lib/py4j-0.10.9.5-src.zip  
-> hdfs://nodo1:9000/user/hadoop/.sparkStaging/application_1681817673583_0002/py4j-0.10.9.5-src.zip  
2023-04-18 11:45:05,409 INFO yarn.Client: Uploading resource file:/tmp/spark-cc109c59-c5be-4223-8c6f-61cfb3d22e6d/_sp  
ark_conf_8067102140360714102.zip -> hdfs://nodo1:9000/user/hadoop/.sparkStaging/application_1681817673583_0002/_sp  
2023-04-18 11:45:06,554 INFO yarn.Client: Application report for application_1681817673583_0002 (state: ACCEPTED)
```

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 6. Una vez que esta en la cola y ya tiene sitio para su ejecución, pasa a estado RUNNING. Empezara a llamar a los nodos que necesite, en función del número del código indicado y del tamaño del fichero. Al final indicara FINISHED y SUCCEEDED si ha sido exitoso

```
2023-04-18 11:45:09,608 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:09,608 INFO yarn.Client:
    client token: N/A
    diagnostics: N/A
    ApplicationMaster host: nodo2
    ApplicationMaster RPC port: 41315
    queue: default
    start time: 1681818305526
    final status: UNDEFINED
    tracking URL: http://nodo1:8088/proxy/application_1681817673583_0002/
    user: hadoop
2023-04-18 11:45:10,611 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:11,614 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:12,619 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:13,621 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:14,634 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:15,637 INFO yarn.Client: Application report for application_1681817673583_0002 (state: RUNNING)
2023-04-18 11:45:16,642 INFO yarn.Client: Application report for application_1681817673583_0002 (state: FINISHED)
2023-04-18 11:45:16,642 INFO yarn.Client:
    client token: N/A
    diagnostics: N/A
    ApplicationMaster host: nodo2
    ApplicationMaster RPC port: 41315
    queue: default
    start time: 1681818305526
    final status: SUCCEEDED
    tracking URL: http://nodo1:8088/proxy/application_1681817673583_0002/
    user: hadoop
```

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 7. De la misma forma que todo programa lanzado sobre Yarn o MapReduce, cuando lanzamos un programa Spark a través de Hadoop, también podemos verlo a través de la pagina web de gestión del clúster → nodo1:8088

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers
application_1681817673583_0002	hadoop	ContarPalabras	SPARK	default	0	Tue Apr 18 11:45:05 +0000 2023	Tue Apr 18 11:45:05 +0000 2023	Tue Apr 18 11:45:15 +0000 2023	FINISHED	SUCCEEDED	N/A

De nuestra aplicación lanzada observamos:

- Application ID
- el usuario que la ha lanzado
- el nombre que le hemos puesto
- al momento de lanzarlo
- Aplicación tipo spark

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 8. Si hacemos click en ApplicationID vemos los parámetros como una aplicación MapReduce normal: Observamos que se ha lanzado por el nodo2, acceso a los logs, etc

The screenshot displays the Hadoop YARN web interface for an application named 'ContarPalabras'. The interface is divided into several sections:

- Application Information:** A table-like view showing key properties of the application.
- Resource Information:** A table-like view showing resource allocation and preemption details.
- Attempt List:** A table showing the details of the application's attempts.

Application Information:

User:	hadoop
Name:	ContarPalabras
Application Type:	SPARK
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	mar abr 18 11:45:05 +0000 2023
Launched:	mar abr 18 11:45:05 +0000 2023
Finished:	mar abr 18 11:45:15 +0000 2023
Elapsed:	10sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Resource Information:

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	57481 MB-seconds, 27 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Attempt List:

Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by
appattempt_1681817673583_0002_000001	Tue Apr 18 11:45:05 +0000 2023	http://nodo2:8042	Logs	0	0

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 9. Si accedemos a los logs, en stdout que es la salida estándar del sistema, no hay mucha información. En cambio en stderr si que se indican los pasos concretos que se han realizado con esta aplicación Python. Vemos que todo ha funcionado correctamente y que ha grabado la información en el sitio adecuado.






Showing 4096 bytes. Click [here](#) for full log

```
TrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 192.168.0.102:41898
2023-04-18 11:44:46,366 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 1.0 (TID 3) in 382 ms on nodo2 (executo
2023-04-18 11:44:46,368 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 2) in 385 ms on nodo3 (executo
2023-04-18 11:44:46,370 INFO scheduler.DAGScheduler: ResultStage 1 (runJob at SparkHadoopWriter.scala:83) finished in 0,4
2023-04-18 11:44:46,372 INFO scheduler.DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks.
2023-04-18 11:44:46,372 INFO cluster.YarnClusterScheduler: Removed TaskSet 1.0, whose tasks have all completed, from pool
2023-04-18 11:44:46,372 INFO cluster.YarnClusterScheduler: Killing all running tasks in stage 1: Stage finished
2023-04-18 11:44:46,375 INFO scheduler.DAGScheduler: Job 0 finished: runJob at SparkHadoopWriter.scala:83, took 2,619539
2023-04-18 11:44:46,376 INFO io.SparkHadoopWriter: Start to commit write Job job_202304181144438538037312073573274_0008.
2023-04-18 11:44:46,479 INFO storage.BlockManagerInfo: Removed broadcast_1_piece0 on nodo2:39279 in memory (size: 7.3 KiB
2023-04-18 11:44:46,481 INFO storage.BlockManagerInfo: Removed broadcast_1_piece0 on nodo3:34527 in memory (size: 7.3 KiB
2023-04-18 11:44:46,491 INFO storage.BlockManagerInfo: Removed broadcast_1_piece0 on nodo2:46739 in memory (size: 7.3 KiB
2023-04-18 11:44:46,500 INFO storage.BlockManagerInfo: Removed broadcast_2_piece0 on nodo2:46739 in memory (size: 55.4 Ki
2023-04-18 11:44:46,506 INFO storage.BlockManagerInfo: Removed broadcast_2_piece0 on nodo3:34527 in memory (size: 55.4 Ki
2023-04-18 11:44:46,507 INFO storage.BlockManagerInfo: Removed broadcast_2_piece0 on nodo2:39279 in memory (size: 55.4 Ki
2023-04-18 11:44:46,508 INFO io.SparkHadoopWriter: Write Job job_202304181144438538037312073573274_0008 committed. Elapse
2023-04-18 11:44:46,521 INFO yarn.ApplicationMaster: Final app status: SUCCEEDED, exitCode: 0
2023-04-18 11:44:46,530 INFO spark.SparkContext: Invoking stop() from shutdown hook
2023-04-18 11:44:46,540 INFO server.AbstractConnector: Stopped Spark@26f3f30e{HTTP/1.1, {http/1.1}}{0.0.0.0:0}
2023-04-18 11:44:46,545 INFO ui.SparkUI: Stopped Spark web UI at http://nodo2:32931
2023-04-18 11:44:46,551 INFO cluster.YarnClusterSchedulerBackend: Shutting down all executors
2023-04-18 11:44:46,551 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
2023-04-18 11:44:46,563 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2023-04-18 11:44:46,593 INFO memory.MemoryStore: MemoryStore cleared
2023-04-18 11:44:46,594 INFO storage.BlockManager: BlockManager stopped
2023-04-18 11:44:46,597 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2023-04-18 11:44:46,598 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator s
2023-04-18 11:44:46,627 INFO spark.SparkContext: Successfully stopped SparkContext
2023-04-18 11:44:46,627 INFO yarn.ApplicationMaster: Unregistering ApplicationMaster with SUCCEEDED
2023-04-18 11:44:46,634 INFO impl.AMRMClientImpl: Waiting for application to be successfully unregistered.
2023-04-18 11:44:46,737 INFO yarn.ApplicationMaster: Deleting staging directory hdfs://nodo1:9000/user/hadoop/.sparkStagi
2023-04-18 11:44:46,744 INFO util.ShutdownHookManager: Shutdown hook called
2023-04-18 11:44:46,744 INFO util.ShutdownHookManager: Deleting directory /tmp/hadoop-hadoop/nm-local-dir/usercache/hadoo
2023-04-18 11:44:46,747 INFO util.ShutdownHookManager: Deleting directory /tmp/hadoop-hadoop/nm-local-dir/usercache/hadoo
```






8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 10. Si vamos a la web HDFS → nodo1:9870, en Utilities/Browse the file System, vemos que se ha generado el directorio /salida_spark_wc y dentro tenemos el típico el _SUCCESS y los part con la información que haya podido generar, en este caso, en 2 nodos

Browse Directory

/ Go!   

Show 25 entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 18 11:35	0	0 B	practicas	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 18 11:45	0	0 B	salida_spark_wc	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 17 16:38	0	0 B	spark	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 18 07:04	0	0 B	user	

Showing 1 to 4 of 4 entries Previous 1 Next

File information - part-00000




Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0





Block ID: 1073741860
Block Pool ID: BP-627469084-192.168.0.101-1677877450981
Generation Stamp: 1036
Size: 325770
Availability:
• nodo3
• nodo2

Close

Browse Directory

/salida_spark_wc Go!   

Show 25 entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	0 B	Apr 18 11:45	2	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	318.13 KB	Apr 18 11:45	2	128 MB	part-00000	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	315.85 KB	Apr 18 11:45	2	128 MB	part-00001	

Showing 1 to 3 of 3 entries Previous 1 Next

File information - part-00001

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073741859
Block Pool ID: BP-627469084-192.168.0.101-1677877450981
Generation Stamp: 1035
Size: 323432
Availability:
• nodo2
• nodo3

Close

8. LANZAR PROGRAMA PYTHON CONTRA YARN

Paso 11. Si vamos al sistema de ficheros HDFS desde la línea de comandos, podemos listar el directorio /salida_spark_wc y hacer un cat de los ficheros part-00000 y part-00001 (o un get para cogerlos)

```
hadoop@nodol1:~/Downloads$ hdfs dfs -ls /salida_spark_wc
Found 3 items
-rw-r--r--    2 hadoop supergroup          0 2023-04-18 13:02 /salida_spark_wc/_SUCCESS
-rw-r--r--    2 hadoop supergroup 325770 2023-04-18 13:02 /salida_spark_wc/part-00000
-rw-r--r--    2 hadoop supergroup 323432 2023-04-18 13:02 /salida_spark_wc/part-00001
hadoop@nodol1:~/Downloads$ hdfs dfs -cat /salida_spark_wc/part-00000
```

```
('malla', 1)
('arman;', 1)
('descabezar,', 1)
('batel', 1)
('jarcia', 1)
('incontrastable', 1)
('pergaminos,', 1)
('teórica', 1)
('vivieron', 1)
('resplandecieron', 1)
('Gaula?;', 1)
('manual', 1)
('Blanco?;', 1)
('Grecia?;', 1)
('Belianís?;', 1)
('Perión', 1)
('Esplandián?;', 1)
('Tracia?;', 1)
('Roldán?;', 1)
('Ferrara', 1)
('Cosmografía?', 1)
('Déstos', 1)
('arbitrio', 1)
('lloviere', 1)
('roe', 1)
('sustentándola', 1)
hadoop@nodol1:~/Downloads$
```

Tendrían que salir cada una de las palabras junto con el número de veces que se repiten. Los ficheros part-0000 y part-00001 son similares

9. SPARK EN MODO STANDALONE

Paso 1. El funcionamiento de Spark en modo StandAlone es independiente de un cluster hadoop. El paquete de instalación ya lleva una pre-instalación de hadoop y las librerías necesarias para crear un cluster standalone. Va muy bien para entornos de pruebas, desarrollos de aplicaciones y testing.

En la pagina de descargas de hadoop tenemos diferentes versiones:

- ❖ La versión Pre-built with user-provided Apache Hadoop, es la versión con la que hemos estado trabajando, que no lleva hadoop preinstalado, porque lo enlazar con nuestro entorno Hadoop
- ❖ Las versiones Pre-built for Apache Hadoop, que son las versiones para instalar y trabajar con Apache Spark en modo StandAlone,



The screenshot shows the 'Download Apache Spark™' page. It has two main steps: '1. Choose a Spark release:' and '2. Choose a package type:'. Step 1 has a dropdown menu with '3.2.4 (Apr 13 2023)' selected. Step 2 has a dropdown menu with several options. The option 'Pre-built with user-provided Apache Hadoop' is highlighted in blue.

Step	Option
1. Choose a Spark release:	3.2.4 (Apr 13 2023) ▼
2. Choose a package type:	Pre-built for Apache Hadoop 3.2 and later ▼
3.	Pre-built for Apache Hadoop 3.2 and later
3.	Pre-built for Apache Hadoop 3.2 and later (Scala 2.13)
4.	Pre-built for Apache Hadoop 2.7
	Pre-built with user-provided Apache Hadoop
	Source Code

9. SPARK EN MODO STANDALONE

Paso 2. Hay tres versiones que vienen con un paquete Hadoop preinstalado. No es un Hadoop completo, pero viene con las librerías de Hadoop y todo lo necesario para poder montar un entorno cluster standalone, para poder empezar a hacer pruebas lo mas rápidamente posible, independiente de un cluster hadoop convencional.

Nos descargamos la versión 3.2 que lleva hadoop empotrado:

Download Apache Spark™

1. Choose a Spark release:
2. Choose a package type:
3. Download Spark: [spark-3.2.4-bin-hadoop3.2.tgz](#)

```
hadoop@nodo1:~/Downloads$ ls
ContarPalabras.py  puertos.csv  spark-3.2.4-bin-hadoop3.2.tgz
hadoop-3.2.4.tar.gz  quijote.txt  spark-3.2.4-bin-without-hadoop.tgz
hadoop@nodo1:~/Downloads$
```

9. SPARK EN MODO STANDALONE

Paso 3. Instalaremos la versión standalone de Spark en cualquier sitio distinto a /opt/hadoop, ya que este es el directorio de los productos asociados al cluster Hadoop. Tampoco le indicaremos ninguna configuración de Hadoop, porque entonces intentaría ir a nuestro clúster y lo que queremos es que funcione de forma independiente (standalone)

Lo descomprimos en Downloads y después movemos el contenido a /home/hadoop/spark_standalone

```
1 hadoop@nodol1:~/Downloads$ tar xvf spark-3.2.4-bin-hadoop3.2.tgz
spark-3.2.4-bin-hadoop3.2/
spark-3.2.4-bin-hadoop3.2/R/
spark-3.2.4-bin-hadoop3.2/R/lib/
```

```
hadoop@nodol1:~/Downloads$ ls
ContarPalabras.py      spark-3.2.4-bin-hadoop3.2
hadoop-3.2.4.tar.gz    spark-3.2.4-bin-hadoop3.2.tgz
puertos.csv            spark-3.2.4-bin-without-hadoop.tgz
quijote.txt
hadoop@nodol1:~/Downloads$ mv spark-3.2.4-bin-hadoop3.2 spark_standalone
hadoop@nodol1:~/Downloads$ mv spark_standalone/ ..
hadoop@nodol1:~/Downloads$ cd ..
hadoop@nodol1:~$ ls
Desktop  Downloads  Music      practicas  Public  spark_standalone  Videos
Documents  fl.txt    Pictures  prueba.txt  snap    Templates
hadoop@nodol1:~$
```

9. SPARK EN MODO STANDALONE

Paso 4. Usaremos el usuario root u otro usuario para arrancar y trabajar en modo standalone. El usuario hadoop tiene todas las variables de configuración y de entorno apuntando a nuestro cluster Hadoop. Si ejecutamos las comandas de Spark standalone con este usuario, intentaría trabajar con nuestro cluster hadoop y eso no lo queremos. En cambio el usuario root no tiene ninguna configuración del cluster hadoop existente.

```
hadoop@nodol:~/spark_standalone/sbin$ sudo -s  
[sudo] password for hadoop:  
root@nodol:/home/hadoop/spark_standalone/sbin#
```

```
root@nodol:/home/hadoop/spark_standalone/sbin# exit  
exit  
root@nodol:/home/hadoop/spark_standalone/sbin#
```

Paso 5. Se recomienda parar el clister Hadoop para asegurarnos mas aun si cabe que esta trabajando en modo standalone

stop-yarn.sh y **stop-dfs.sh**

9. SPARK EN MODO STANDALONE

Paso 6. Para arrancar y trabajar con Spark en modo cluster standalone, tenemos que arrancar maestro y esclavos como pasa con un cluster Hadoop, donde tenemos varios nodos Master y esclavos. En el directorio sbin de spark tenemos una serie de scripts que nos permiten arrancar un cluster standAlone. En concreto usaremos:

- ❖ start-master.sh → para arrancar el maestro
- ❖ start-slaves.sh → para arrancar el esclavo

```
hadoop@nodol:~/spark_standalone/sbin$ ls
decommission-slave.sh      start-worker.sh
decommission-worker.sh    start-workers.sh
slaves.sh                 stop-all.sh
spark-config.sh           stop-history-server.sh
spark-daemon.sh           stop-master.sh
spark-daemons.sh         stop-mesos-dispatcher.sh
start-all.sh             stop-mesos-shuffle-service.sh
start-history-server.sh   stop-slave.sh
start-master.sh           stop-slaves.sh
start-mesos-dispatcher.sh stop-thriftserver.sh
start-mesos-shuffle-service.sh stop-worker.sh
start-slave.sh            stop-workers.sh
start-slaves.sh           workers.sh
start-thriftserver.sh
hadoop@nodol:~/spark_standalone/sbin$
```


9. SPARK EN MODO STANDALONE

Paso 7. Arrancaremos primero el maestro y comprobaremos que está arriba. Dentro del usuario root, ejecutamos **./start-master.sh** dentro del directorio spark_standalone/sbin

```
root@nod01:/home/hadoop/spark_standalone/sbin# ./start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/spark_standalone/logs/spark-root-org.apache.spark.deploy.master.Master-1-nod01.out
root@nod01:/home/hadoop/spark_standalone/sbin#
```

Nos indica el directorio de log, en caso de que haya problemas con la arrancada

Paso 8. Para parar el maestro podemos hacer **./stop-master.sh**. Lo volvemos a arrancar

```
root@nod01:/home/hadoop/spark_standalone/sbin# ./stop-master.sh
stopping org.apache.spark.deploy.master.Master
root@nod01:/home/hadoop/spark_standalone/sbin# ./start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/spark_standalone/logs/spark-root-org.apache.spark.deploy.master.Master-1-nod01.out
root@nod01:/home/hadoop/spark_standalone/sbin#
```

9. SPARK EN MODO STANDALONE

Paso 9. Para comprobar que el master standalone esta arrancando, ponemos en un navegador web localhost:8080. Aparece un entorno web de gestión de Spark en modo standalone. Indica que:

- ❖ El Spark Master esta escuchando en el puerto 7077
- ❖ El numero de Alive Workers o esclavos tenemos funcionando
- ❖ Cuántas aplicaciones se están ejecutando o han terminado.

← → ↻ localhost:8080

Spark 3.2.4 **Spark Master at spark://nodo1:7077**

URL: spark://nodo1:7077
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

- Workers (0)

Worker Id	Address	State	Cores
-----------	---------	-------	-------

- Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor
----------------	------	-------	---------------------	------------------------

- Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor
----------------	------	-------	---------------------	------------------------

9. SPARK EN MODO STANDALONE

Paso 10. Si hacemos `ps -ef | grep spark`, vemos que efectivamente ha arrancado el maestro y que aquí en esta página web lo puedo ver.

```
root@nodol:/home/hadoop/spark_standalone/sbin# ps -ef | grep spark
root      291683      1257    0 abr18 ?                00:00:16 /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -cp /home/hadoop/s
park_standalone/conf/./home/hadoop/spark_standalone/jars/* -Xmx1g org.apache.spark.deploy.master.Master --host nodol
--port 7077 --webui-port 8080
root      709572    293436    0 03:59 pts/0      00:00:00 grep --color=auto spark
root@nodol:/home/hadoop/spark_standalone/sbin#
```

9. SPARK EN MODO STANDALONE

Paso 11. Para arrancar el esclavo le tenemos que indicar en que nodo y por qué puerto está escuchando el maestro. En la misma maquina tendremos un proceso maestro y esclavo standalone

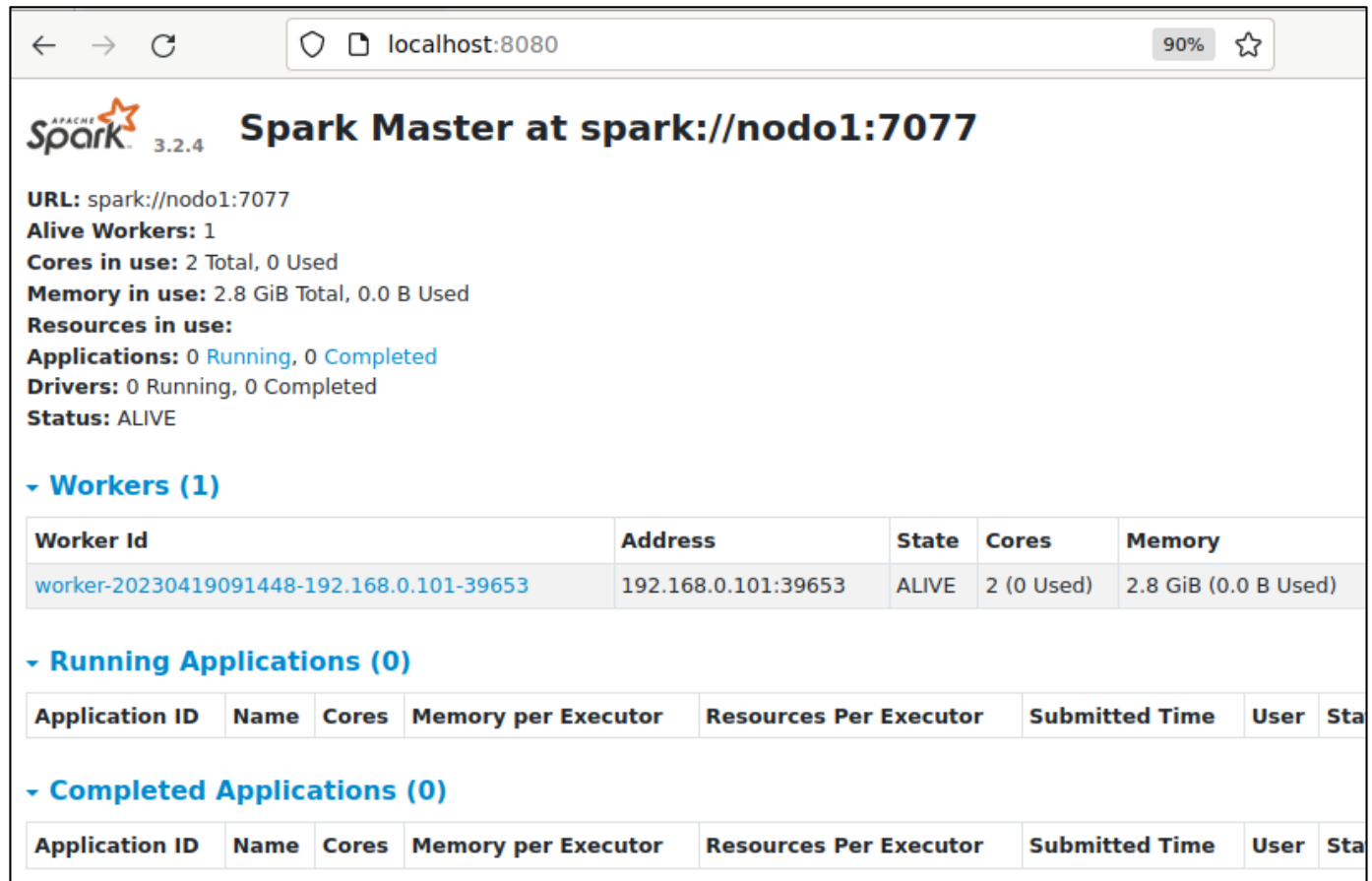
Si queremos establecer otro esclavo en otro nodo, previamente se deben de copiar los binarios de Spark allí, y en el comando de arrancada indicarle donde esta el maestro y su puerto, en este caso **nodo1:7077**

```
root@nod01:/home/hadoop/spark_standalone/sbin# ./start-worker.sh nodo1:7077
starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/spark_standalone/logs
/spark-root-org.apache.spark.deploy.worker.Worker-1-nod01.out
root@nod01:/home/hadoop/spark_standalone/sbin#
```


Si todo va bien nos indica el fichero de log por donde podríamos buscar si ha habido problemas

9. SPARK EN MODO STANDALONE

Paso 12. Vamos a la pagina de administración Spark standalone localhost:8080, refrescamos y nos debería salir el worker que hemos arrancado. Este esclavo es el que va a recibir las peticiones de los clientes



← → ↻ localhost:8080 90% ☆

 3.2.4 **Spark Master at spark://nodo1:7077**

URL: spark://nodo1:7077
Alive Workers: 1
Cores in use: 2 Total, 0 Used
Memory in use: 2.8 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

▾ **Workers (1)**

Worker Id	Address	State	Cores	Memory
worker-20230419091448-192.168.0.101-39653	192.168.0.101:39653	ALIVE	2 (0 Used)	2.8 GiB (0.0 B Used)

▾ **Running Applications (0)**

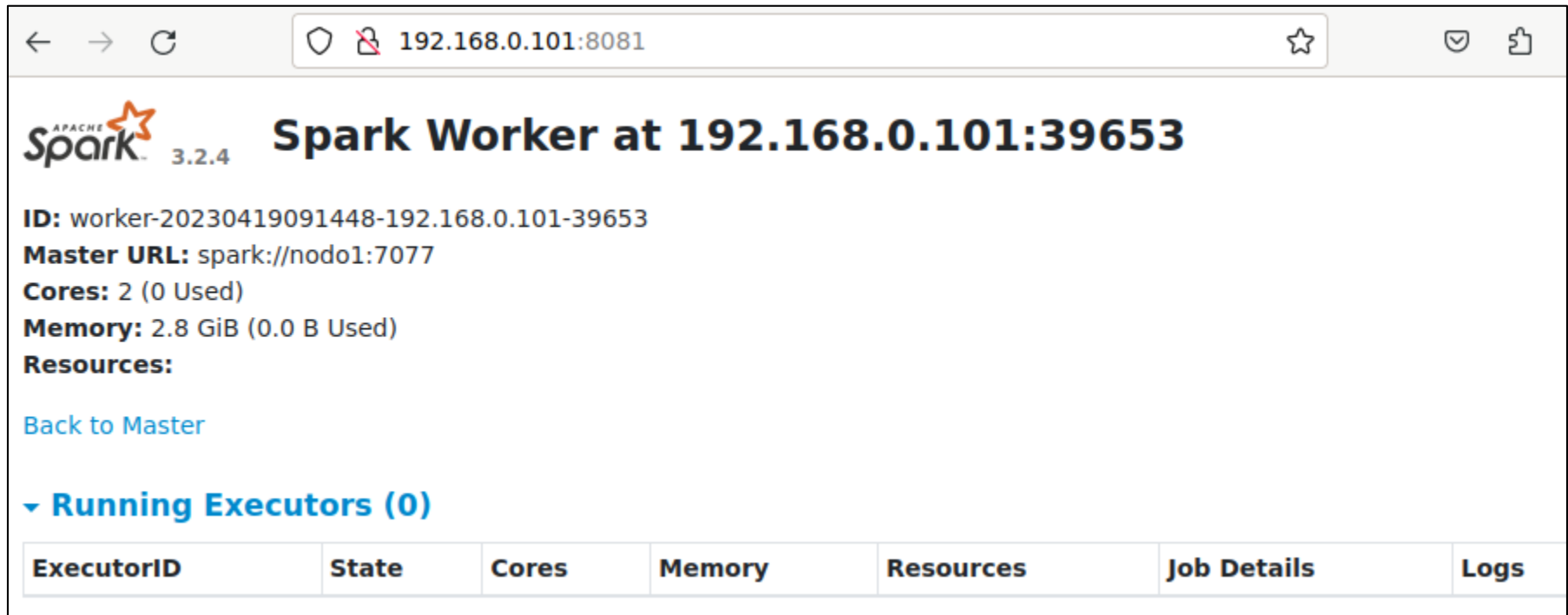
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	Sta
----------------	------	-------	---------------------	------------------------	----------------	------	-----

▾ **Completed Applications (0)**


Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	Sta
----------------	------	-------	---------------------	------------------------	----------------	------	-----

9. SPARK EN MODO STANDALONE

Paso 13. Si hacemos click en el worker_id nos sale una pantalla de este nodo donde se listan las tareas que ha ido realizando



← → ↻ 192.168.0.101:8081 ☆

 **Spark** 3.2.4 **Spark Worker at 192.168.0.101:39653**

ID: worker-20230419091448-192.168.0.101-39653
Master URL: spark://nodo1:7077
Cores: 2 (0 Used)
Memory: 2.8 GiB (0.0 B Used)
Resources:

[Back to Master](#)

▼ **Running Executors (0)**

ExecutorID	State	Cores	Memory	Resources	Job Details	Logs
------------	-------	-------	--------	-----------	-------------	------

Ya tenemos un nodo worker arrancado y podríamos ir arrancando otros nodos esclavos en otros en otras máquinas físicas para hacer el cluster standalone mas grande

9. SPARK EN MODO STANDALONE

Paso 14. Vamos a hacer un pequeño ejemplo de ejecución, lanzando un programa contra el cluster standalone de Spark formado por un maestro y un esclavo definidos anteriormente.

Abrimos spark-shell en el directorio /spark_standalone/bin/. Vamos a realizar el típico programa en Scala, que lea un fichero y cuente el numero de líneas que tiene

[illegible]

9. SPARK EN MODO STANDALONE

Paso 15. Creamos una variable fichero usando el contexto sc y la función `textFile`. Leeremos el fichero `README.md`, que se encuentra en la base del directorio `spark`. Esto nos va a crear una RDD de tipo `string`. Llamamos a la función `count` y nos muestra el nº de líneas del fichero: 109

```
scala> val fichero=sc.textFile("../README.md")
fichero: org.apache.spark.rdd.RDD[String] = ../README.md MapPartitionsRDD[3] at textFile at <console>:23

scala> fichero.count()
res1: Long = 109

scala> █
```

NOTA: La primera vez que hicimos esto con Spark en modo Hadoop, daba error porque decía que no estaba el fichero en el sistema HDFS. Le teníamos que poner "file" para que lo cogiera del sistema de ficheros local. Aquí no es necesario porque él sabe que no está trabajando con Hadoop y por lo tanto por defecto va a buscar el fichero dentro del directorio `spark_standalone`

9. SPARK EN MODO STANDALONE

Paso 16. Con la función `first()` de Scala nos indica cuál es la primera línea de ese fichero

```
scala> fichero.first()
res2: String = # Apache Spark
scala>
```

Paso 17. Si quiero saber todas las líneas del fichero `README.md` que contienen la palabra `Spark`, usaremos la función `filter`.

```
scala> val fichero1=fichero.filter(l => l contains("Spark"))
fichero1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:23
.
scala> fichero1.count()
res6: Long = 19
scala>
```

La función `filter` crea un segundo RDD a partir del primero, formado por las líneas que contienen "Spark". Sólo salen sólo 19 líneas

Paso 18. Para salir de Scala, ejecutamos `:q`

```
scala> :q
root@nod01:/home/hadoop/spark_standalone/bin#
```

9. SPARK EN MODO STANDALONE

Pasos para establecer mas esclavos en modo standalone

a) Primero vamos al directorio llamado conf, donde hay un fichero llamado workers.template. Lo copiamos a workers. Y aquí dentro pondríamos los nodos que queremos que tengan esclavos

```
root@nodol:/home/hadoop/spark_standalone/conf# ls -l
total 36
-rw-r--r-- 1 hadoop hadoop 1105 abr  9 21:17 fairscheduler.xml.template
-rw-r--r-- 1 hadoop hadoop 2471 abr  9 21:17 log4j.properties.template
-rw-r--r-- 1 hadoop hadoop 9141 abr  9 21:17 metrics.properties.template
-rw-r--r-- 1 hadoop hadoop 1292 abr  9 21:17 spark-defaults.conf.template
-rwxr-xr-x 1 hadoop hadoop 4428 abr  9 21:17 spark-env.sh.template
-rw-r--r-- 1 hadoop hadoop  865 abr  9 21:17 workers.template
root@nodol:/home/hadoop/spark_standalone/conf#
```

```
root@nodol:/home/hadoop/spark_standalone/conf# cp workers.template workers
root@nodol:/home/hadoop/spark_standalone/conf#
```

```
GNU nano 6.2 workers *
#
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# A Spark Worker will be started on each of the machines listed below.
nodol
nodol2
nodol3
```

9. SPARK EN MODO STANDALONE

- b) Copiamos el software Spark en modo standalone en el resto de los nodos.
- c) Dentro del directorio sbin existe el script start-workers.sh que va arrancando todos los nodos definidos en conf/workers

```
root@nodol:/home/hadoop/spark_standalone/sbin# ls
decommission-slave.sh      start-mesos-dispatcher.sh  stop-master.sh
decommission-worker.sh    start-mesos-shuffle-service.sh  stop-mesos-dispatcher.sh
slaves.sh                 start-slave.sh              stop-mesos-shuffle-service.sh
spark-config.sh           start-slaves.sh             stop-slave.sh
spark-daemon.sh           start-thriftserver.sh       stop-slaves.sh
spark-daemons.sh         start-worker.sh             stop-thriftserver.sh
start-all.sh             start-workers.sh            stop-worker.sh
start-history-server.sh   stop-all.sh               stop-workers.sh
start-master.sh           stop-history-server.sh      workers.sh
root@nodol:/home/hadoop/spark_standalone/sbin#
```