

Cluster Spark con VM ROCKY
40% Parcial Procesamiento de datos



Isaías Acosta Herrera

Juan Felipe Gonzalez

Diego Alejandro Jara

Tomas Felipe Guerra

Pontificia Universidad Javeriana de Colombia

Procesamiento de Datos

Jhon Corredor

Bogotá, Colombia

8 de Septiembre de 2025

INTRODUCCIÓN:

En este taller nos enfocamos en aprender cómo se configura un clúster de Apache Spark desde cero en un entorno Linux (Rocky). La idea principal era pasar de tener máquinas virtuales independientes a lograr que trabajen de forma coordinada bajo un esquema maestro-trabajadores, lo cual es la base de los sistemas distribuidos modernos.

Con este taller se buscaba adquirir una visión mucho más clara de cómo se gestiona la infraestructura en proyectos de procesamiento de datos, y cómo una buena configuración inicial es la clave para que Spark pueda desplegarse y correr trabajos en paralelo sin problemas.

El proceso fue muy práctico porque no solo instalamos el software necesario, sino que también entendimos por qué se necesitan ciertos pasos, como la comunicación sin contraseña entre nodos, la creación de un sistema de archivos compartido y la configuración de variables de entorno que permiten que Spark funcione correctamente en todos los equipos.

OBJETIVO:

El objetivo principal fue configurar y dejar en funcionamiento un clúster de Spark en modo standalone. Esto implicó:

- Garantizar que el nodo maestro pudiera comunicarse con los nodos trabajadores sin restricciones.
- Crear un sistema de archivos compartido que asegurara que todos los nodos tuvieran acceso al mismo Spark instalado.
- Instalar las dependencias necesarias (como Java y Python).
- Definir las variables de entorno que permiten que Spark reconozca su ubicación y pueda ejecutarse correctamente.
- Levantar los servicios y validar la correcta operación del clúster desde la línea de comandos y también desde la interfaz web.

En últimas, se trató de replicar un escenario real de trabajo con Spark, entendiendo cómo se organiza la infraestructura detrás de un sistema que se usa a diario en Big Data.

METODOLOGÍA:

La metodología utilizada siguió una estructura clara , con pasos muy definidos que fuimos ejecutando uno a uno como se evidencia a continuación:

3.1 Comunicación entre nodos

Primero establecemos la conexión sin contraseña entre el maestro y los trabajadores usando ssh-keygen y ssh-copy-id. Esto es esencial porque un clúster debe poder ejecutar tareas en diferentes nodos sin que el usuario esté ingresando credenciales cada vez. Después, editamos el archivo `/etc/hosts` para registrar las IPs de los nodos con nombres más fáciles de recordar. Esto hace más simple identificar quién es el maestro y quiénes son los trabajadores dentro del sistema.

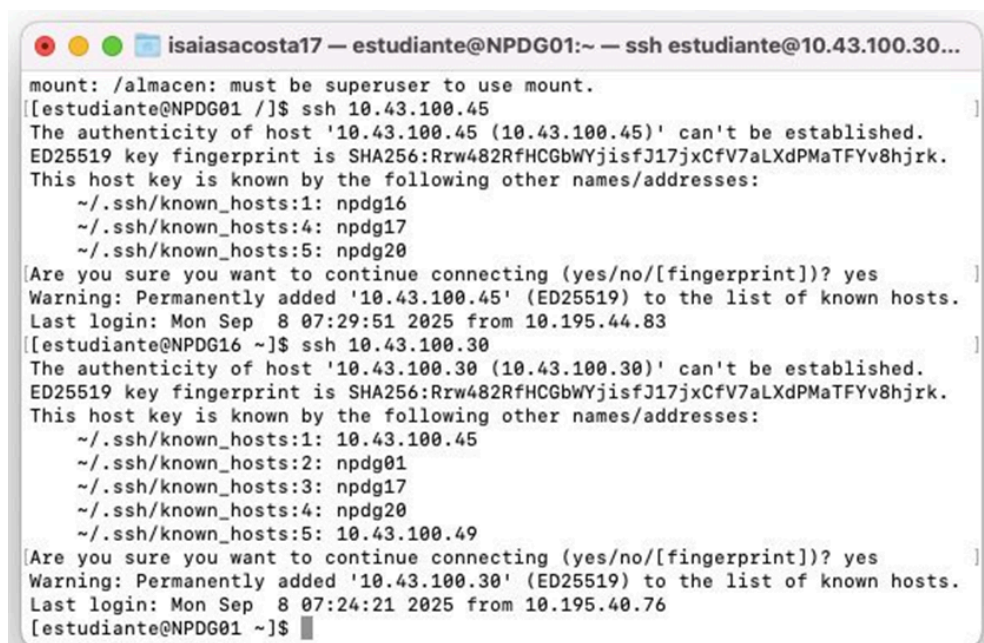
- Se le agregó al archivo `/etc/hosts` de todos los workers y el master:

La IP y el nombre respectivo de cada nodo, para que se puedan acceder entre sí con el nombre del nodo sin necesidad de la IP.

```
GNU nano 5.6.1 /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

10.43.100.30 NPDG01
10.43.100.46 NPDG17
10.43.100.49 NPDG20
10.43.100.45 NPDG16
```

- Comprobación de la conexión entre clústeres



```
isaiasacosta17 — estudiante@NPDG01:~ — ssh estudiante@10.43.100.30...
mount: /almacen: must be superuser to use mount.
[estudiante@NPDG01 /]$ ssh 10.43.100.45
The authenticity of host '10.43.100.45 (10.43.100.45)' can't be established.
ED25519 key fingerprint is SHA256:Rrw482RfHCGbWYjisfJ17jxCfV7aLXdPMaTFYv8hjrK.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: npdg16
  ~/.ssh/known_hosts:4: npdg17
  ~/.ssh/known_hosts:5: npdg20
[Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.43.100.45' (ED25519) to the list of known hosts.
Last login: Mon Sep  8 07:29:51 2025 from 10.195.44.83
[estudiante@NPDG16 ~]$ ssh 10.43.100.30
The authenticity of host '10.43.100.30 (10.43.100.30)' can't be established.
ED25519 key fingerprint is SHA256:Rrw482RfHCGbWYjisfJ17jxCfV7aLXdPMaTFYv8hjrK.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: 10.43.100.45
  ~/.ssh/known_hosts:2: npdg01
  ~/.ssh/known_hosts:3: npdg17
  ~/.ssh/known_hosts:4: npdg20
  ~/.ssh/known_hosts:5: 10.43.100.49
[Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.43.100.30' (ED25519) to the list of known hosts.
Last login: Mon Sep  8 07:24:21 2025 from 10.195.40.76
[estudiante@NPDG01 ~]$
```

3.2 Sistema de archivos compartido

Luego creamos una carpeta llamada /almacen en el maestro y la configuramos con NFS (Network File System) para que los trabajadores pudieran acceder a ella. Con esto, todos los nodos comparten el mismo contenido y evitan duplicaciones innecesarias. Después, desde cada trabajador montamos la carpeta del maestro y verificamos que el montaje estaba correcto usando df -h. Esta parte fue clave porque sobre esa carpeta se instaló Spark.

- Para compartir el servidor NFS se ponen las direcciones IP de los workers en el archivo /etc/exports compartiendo la carpeta almacén.

```
GNU nano 5.6.1 /etc/exports
/almacen 10.43.100.46(rw,no_root_squash,subtree_check,fsid=0)
/almacen 10.43.100.49(rw,no_root_squash,subtree_check,fsid=0)
/almacen 10.43.100.30(rw,no_root_squash,subtree_check,fsid=0)
```

- Comprobación del funcionamiento correcto del servidor NFS en el master

```
estudiante@NPDG16 /]$ sudo systemctl status nfs-server
nfs-server.service - NFS server and services
Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; preset: disabled)
Drop-In: /run/systemd/generator/nfs-server.service.d
└─order-with-mounts.conf
Active: active (exited) since Mon 2025-09-08 08:18:36 -05; 8min ago
Docs: man:rpc.nfsd(8)
      man:exportfs(8)
Process: 787084 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
Process: 787087 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
Process: 787118 ExecStart=/bin/sh -c if systemctl -q is-active gssproxy; then systemctl reload gssproxy ; fi (code=exited, status=0/SUCCESS)
Main PID: 787118 (code=exited, status=0/SUCCESS)
CPU: 41ms

Sep 08 08:18:36 NPDG16 systemd[1]: Starting NFS server and services...
Sep 08 08:18:36 NPDG16 systemd[1]: Finished NFS server and services.
```

- Creación de la carpeta almacén en los workers

```
[[estudiante@NPDG01 ~]$ cd /
[[estudiante@NPDG01 /]$ sudo mkdir -p almacen
[[sudo] password for estudiante:
```

- Se instala NFS en cada worker.

```

[estudiante@NPDG01 ~]$ sudo dnf install -y nfs-utils
[sudo] password for estudiante:
Last metadata expiration check: 2:54:54 ago on Mon 08 Sep 2025 05:35:13 AM -05.
Dependencies resolved.
=====
Package                                                    Architecture
=====
Installing:
nfs-utils                                                    x86_64
Installing dependencies:
gssproxy                                                    x86_64
libev                                                        x86_64
libnfsidmap                                                  x86_64
libverto-libev                                              x86_64
python3-pyyaml                                              x86_64
quota                                                       x86_64
quota-nls                                                   noarch
rpcbind                                                     x86_64
sssd-nfs-idmap                                              x86_64
=====
Transaction Summary
=====
Install 10 Packages

```

- Se monta en el directorio de los workers la carpeta almacén del máster. Esto se realiza con `sudo mount 10.43.100.45:/almacen /almacen` donde 10.43.100.45 es el nodo master.

```

[estudiante@NPDG01 ~]$ sudo mount 10.43.100.45:/almacen /almacen
Created symlink /run/systemd/system/remote-fs.target.wants/rpc-statd.service → /usr/lib/systemd/system/rpc-statd.service.

```

```

[estudiante@NPDG01 ~]$ df -h

```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	4.0M	0	4.0M	0%	/dev
tmpfs	7.7G	0	7.7G	0%	/dev/shm
tmpfs	3.1G	21M	3.1G	1%	/run
efivarfs	256K	43K	209K	18%	/sys/firmware/efi/efivars
/dev/mapper/rl_plantillarocky9-root	24G	4.1G	20G	18%	/
/dev/sda2	960M	416M	545M	44%	/boot
/dev/sda1	1022M	7.1M	1015M	1%	/boot/efi
/dev/mapper/rl_plantillarocky9-var	15G	404M	15G	3%	/var
/dev/mapper/rl_plantillarocky9-home	15G	185M	15G	2%	/home
tmpfs	1.6G	4.0K	1.6G	1%	/run/user/1001
10.43.100.45:/almacen	24G	4.1G	20G	18%	/almacen

3.3 Instalación y despliegue de Spark

Una vez asegurada la comunicación y el almacenamiento compartido, copiamos el paquete de Spark (`spark-3.5.2-bin-hadoop3.tgz`) al directorio `/almacen`. Luego lo descomprimos y renombramos la carpeta a “Spark” para mayor claridad. Además, instalamos Java 8 en todos los nodos, ya que Spark lo requiere como dependencia. Finalmente, configuramos las variables de entorno en `.bashrc` para que todos los comandos de Spark pudieran ejecutarse sin problemas y quedarán disponibles desde cualquier sesión.

- Copia de Spark desde el servidor de un compañero de clase a la carpeta `almacen`.

```
[estudiante@NPDG16 almacén]$ scp estudiante@10.43.100.44:/almacen/*.tgz .
The authenticity of host '10.43.100.44 (10.43.100.44)' can't be established.
ED25519 key fingerprint is SHA256:Rrw482RfHCGbWYjisfJ17jxCfV7aLXdPMaTFYv8Hjrk.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: 10.43.100.45
  ~/.ssh/known_hosts:2: npdg01
  ~/.ssh/known_hosts:3: npdg17
  ~/.ssh/known_hosts:4: npdg20
  ~/.ssh/known_hosts:5: 10.43.100.49
  ~/.ssh/known_hosts:6: 10.43.100.30
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.43.100.44' (ED25519) to the list of known hosts.
estudiante@10.43.100.44's password:
spark-3.5.6-bin-hadoop3.tgz                                100% 382MB 438.7MB/s   00:00
[estudiante@NPDG16 almacén]$ ll
total 391532
-rw-r--r--. 1 estudiante estudiante      44 Sep  8 08:31 hola
-rw-r--r--. 1 estudiante estudiante 400923510 Sep  8 08:35 spark-3.5.6-bin-hadoop3.tgz
```

- Descomprimir archivo Spark

```
[estudiante@NPDG16 almacén]$ tar -zxvf spark-3.5.6-bin-hadoop3.tgz
```

Después de realizar la descompresión del archivo de instalación con tar, podemos verificar que efectivamente se encuentre la versión descomprimida y con el comando “mv spark-3.5.6-bin-hadoop3 Spark” podemos cambiar su nombre para que sea más sencillo acceder más adelante.

```
[estudiante@NPDG16 almacén]$ ll
total 391532
drwxr-sr-x. 13 estudiante estudiante    4096 May 23 01:49 spark-3.5.6-bin-hadoop3
-rw-r--r--.  1 estudiante estudiante 400923510 Sep  8 08:35 spark-3.5.6-bin-hadoop3.tgz
[estudiante@NPDG16 almacén]$ mv spark-3.5.6-bin-hadoop3 Spark
[estudiante@NPDG16 almacén]$ ll
```

- Instalamos Java en el master y los workers

```
[[estudiante@NPDG01 ~]$ sudo dnf install java-1.8.0-openjdk java-1.8.0-openjdk-devel -y
[[sudo] password for estudiante:
Last metadata expiration check: 3:10:28 ago on Mon 08 Sep 2025 05:35:13 AM -05.
Dependencies resolved.

=====
Package                                     Architecture
=====
Installing:
  java-1.8.0-openjdk                        x86_64
  java-1.8.0-openjdk-devel                  x86_64
Installing dependencies:
  ttmkfdir                                  x86_64
  xorg-x11-fonts-Type1                      noarch

Transaction Summary
=====
Install  4 Packages

Total download size: 10 M

[[estudiante@NPDG01 ~]$ java -version
openjdk version "1.8.0_462"
OpenJDK Runtime Environment (build 1.8.0_462-b08)
OpenJDK 64-Bit Server VM (build 25.462-b08, mixed mode)
```

- Se exporta en todos los nodos la ubicación de SPARK y Python.


```
[[estudiante@NPDG01 ~]$ env | grep -i SPARK
[[estudiante@NPDG01 ~]$ cd
[[estudiante@NPDG01 ~]$ nano .bashrc
```

```
# export SYSTEMD_PAGER=

export SPARK_HOME="/almacen/Spark"
export SPARK_CONF_DIR="$SPARK_HOME/conf"
export PATH="$SPARK_HOME/bin":$PATH
export PYSPARK_PYTHON="/usr/local/bin/python3"
export PYSPARK_DRIVER_PYTHON="/usr/local/bin/python3"
```

3.4 Configuración específica de Spark y trabajo sobre los archivos spark-env.sh y workers

En el primero, definimos la IP del maestro, lo que asegura que los servicios sepan a dónde conectarse. En el segundo, listamos las IPs de los trabajadores para que el maestro pueda reconocerlos al momento de levanta

Entramos al directorio de configuración de Spark el clúster.

- Se ha duplicado el archivo de configuración de Spark “spark-env.sh” desde el template y están listos los ficheros que controlan la configuración de Spark master y los workers.

```
[[estudiante@NPDG01 conf]$ ll
total 44
-rw-r--r--. 1 estudiante estudiante 1105 May 23 01:49 fairscheduler.xml.template
-rw-r--r--. 1 estudiante estudiante 3350 May 23 01:49 log4j2.properties.template
-rw-r--r--. 1 estudiante estudiante 9141 May 23 01:49 metrics.properties.template
-rw-r--r--. 1 estudiante estudiante 1292 May 23 01:49 spark-defaults.conf.template
-rwxr-xr-x. 1 estudiante estudiante 4694 Sep  8 08:56 spark-env.sh
-rwxr-xr-x. 1 estudiante estudiante 4694 May 23 01:49 spark-env.sh.template
-rw-r--r--. 1 estudiante estudiante 865 May 23 01:49 workers.template
[[estudiante@NPDG01 conf]$ nano spark-env.sh
[[estudiante@NPDG01 conf]$ cp workers.template workers
[[estudiante@NPDG01 conf]$ nano workers
[[estudiante@NPDG01 conf]$ sudo yum -y install java-1.8.0-openjdk-devel
[[sudo] password for estudiante:
Last metadata expiration check: 3:27:23 ago on Mon 08 Sep 2025 05:35:13 AM -05.
Package java-1.8.0-openjdk-devel-1:1.8.0.462.b08-3.el9.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[[estudiante@NPDG01 conf]$
```

3.5 Levantamiento de servicios y validación

Con toda la configuración hecha, instalamos la herramienta jps para poder listar los procesos Java en ejecución. Finalmente, ejecutamos el script start-all.sh que levanta tanto el maestro como los trabajadores. La validación se hizo en dos partes: con jps, revisando los procesos activos, y con la interfaz web de Spark (<http://ip-maestro:8080>), donde confirmamos visualmente que el clúster estaba arriba y funcionando.

- Levantamiento del servicio spark junto a pruebas de funcionalidad.

```
[estudiante@NPDG16 Spark]$ cd /almacen/Spark/sbin
[estudiante@NPDG16 sbin]$ ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /almacen/Spark/logs/spark-estudiante-org.apache.spark.deploy.master.Master-1-NPDG16.out
10.43.100.49: starting org.apache.spark.deploy.worker.Worker, logging to /almacen/Spark/logs/spark-estudiante-org.apache.spark.deploy.worker.Worker-1-NPDG20.out
10.43.100.45: starting org.apache.spark.deploy.worker.Worker, logging to /almacen/Spark/logs/spark-estudiante-org.apache.spark.deploy.worker.Worker-1-NPDG16.out
10.43.100.46: starting org.apache.spark.deploy.worker.Worker, logging to /almacen/Spark/logs/spark-estudiante-org.apache.spark.deploy.worker.Worker-1-NPDG17.out
10.43.100.39: starting org.apache.spark.deploy.worker.Worker, logging to /almacen/Spark/logs/spark-estudiante-org.apache.spark.deploy.
```

Accedemos entonces al directorio desde donde podremos levantar nuestro cluster ya construido y allí ejecutamos el comando “./start-all.sh”. Una vez lo ponemos en ejecución vemos cómo se inician cada uno de los nodos (master y workers) y así podemos confirmar que el servicio del clúster está arriba.

```
[estudiante@NPDG16 sbin]$ jps
884988 Jps
883320 Master
883843 Worker
```

Accedemos a la ip creada del cluster donde podemos ver ya la información completa del cluster como sus cores.

Spark Master at spark://10.43.100.45:7077

URL: spark://10.43.100.45:7077
 Alive Workers: 4
 Cores in use: 16 Total, 0 Used
 Memory in use: 57.3 GiB Total, 0.0 B Used
 Resources in use:
 Applications: 0 Running, 0 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers (4)

Worker Id	Address	State	Cores	Memory	Resources
worker-20250908205742-10.43.100.45-37289	10.43.100.45:37289	ALIVE	4 (0 Used)	14.3 GiB (0.0 B Used)	
worker-20250908205744-10.43.100.46-40391	10.43.100.46:40391	ALIVE	4 (0 Used)	14.3 GiB (0.0 B Used)	
worker-20250908205801-10.43.100.49-36161	10.43.100.49:36161	ALIVE	4 (0 Used)	14.3 GiB (0.0 B Used)	
worker-20250908205810-10.43.100.30-34279	10.43.100.30:34279	ALIVE	4 (0 Used)	14.3 GiB (0.0 B Used)	

Running Applications (0)

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

Completed Applications (0)

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

PRUEBAS:

Para ejecutar algunas pruebas sobre nuestro cluster de Spark, acudimos a los ejemplos incluidos dentro de los recursos de instalación. Lo primero que debemos hacer entonces es acceder al directorio donde se encuentra el archivo que contiene todos los ejemplos:

```
[estudiante@NPDG16 ~]$ cd /almacen/Spark/examples
[estudiante@NPDG16 examples]$ ls
jars  src
```


Una vez accedemos, se ve que por defecto hay dos directorios más. Donde se encuentran los ejemplos es en el directorio “jars” como se ve a continuación:

```
[estudiante@NPDG16 examples]$ cd jars
[estudiante@NPDG16 jars]$ ls
scopt_2.12-3.7.1.jar  spark-examples_2.12-3.5.6.jar
```

Luego, si queremos ver la lista de ejemplos disponibles que se encuentran en el archivo “spark-examples_2.12-3.5.6.jar” podemos ejecutar el siguiente comando:

```
[estudiante@NPDG16 jars]$ jar tf /almacen/Spark/examples/jars/spark-examples_2.12-3.5.6.jar | grep org/apache/spark/examples
org/apache/spark/examples/
org/apache/spark/examples/SparkHdfsLR$DataPoint.class
org/apache/spark/examples/SparkLR$DataPoint$.class
org/apache/spark/examples/JavaHdfsLR$ComputeGradient.class
org/apache/spark/examples/mllib/
org/apache/spark/examples/mllib/StreamingTestExample.class
org/apache/spark/examples/mllib/JavaSVDExample.class
org/apache/spark/examples/mllib/HypothesisTestingExample.class
org/apache/spark/examples/mllib/PowerIterationClusteringExample$Params$typecreator1$.class
org/apache/spark/examples/mllib/JavaLBFGSExample.class
org/apache/spark/examples/mllib/PCAOnSourceVectorExample.class
org/apache/spark/examples/mllib/TallSkinnyPCA.class
org/apache/spark/examples/mllib/TallSkinnySVD.class
org/apache/spark/examples/mllib/IsotonicRegressionExample$.class
org/apache/spark/examples/mllib/PowerIterationClusteringExample$.class
```

Como se puede ver en la imagen anterior, una vez ejecutamos el comando ubicado en la parte superior, encontraremos un listado de ejemplos ejecutables. De todas las posibilidades, decidimos escoger dos ejemplos sencillos que probaran el funcionamiento del clúster, por lo que, para efectos prácticos, seleccionamos JavaWordCount y SparkPi, los cuáles se explican a continuación.

1era Prueba: JavaWordCount

Descripción: Archivo de ejecución cuyo objetivo es contar la cantidad de palabras de un archivo de texto que recibe como entrada.

Ejecución

Para ejecutar este archivo de prueba se hizo uso del siguiente comando:

```
[estudiante@NPDG16 jars]$ /almacen/Spark/bin/spark-submit \
--class org.apache.spark.examples.JavaWordCount \
/almacen/Spark/examples/jars/spark-examples_2.12-3.5.6.jar \
/almacen/Spark/README.md
25/09/08 20:38:17 INFO SparkContext: Running Spark version 3.5.6
25/09/08 20:38:17 INFO SparkContext: OS info Linux, 5.14.0-570.26.1.el9_6.x86_64, amd64
25/09/08 20:38:17 INFO SparkContext: Java version 1.8.0_462
```

Ya que para ejecutar esta prueba se requiere un archivo como entrada, utilizamos el README que viene incluido con spark, e hicimos que el script se ejecutara sobre este.

Luego del comando, podemos ver como en los logs empieza a correr Spark y se inician todos los recursos necesarios. Al final de la ejecución podemos ver el resultado, el cuál debería ser como el siguiente:

```
25/09/08 20:38:26 INFO DAGScheduler: Job 0 finished: collect at JavaWordCount.java:53, took 1.647797 s
package: 1
For: 3
Programs: 1
processing.: 2
Because: 1
The: 1
cluster.: 1
its: 1
[run: 1
APIs: 1
have: 1
Try: 1
computation: 1
through: 1
several: 1
This: 2
graph: 1
Hive: 2
storage: 1
["Specifying: 1
To: 2
"yarn": 1
Once: 1
["Useful: 1
```

Como se puede ver, una vez termina de ejecutarse el job aparece cada una de las palabras del archivo README incluido en Spark y la cantidad de repeticiones de las mismas, por lo que se puede afirmar que la prueba fue ejecutada con éxito.

2da Prueba: SparkPi

Descripción: Archivo de ejecución en lenguaje Scala cuyo objetivo es calcular un número dado de los dígitos del número Pi.

Ejecución

Para ejecutar este archivo de prueba se hizo uso del siguiente comando:

```
[estudiante@NPDG16 jars]$ /almacen/Spark/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
/almacen/Spark/examples/jars/spark-examples_2.12-3.5.6.jar 20|
```

Para ejecutar esta prueba, el único argumento de entrada necesario es el número de dígitos que queremos que Spark calcule. Para este caso, se hizo la ejecución con 20 dígitos, lo que arrojó el siguiente resultado:

```
25/09/08 20:49:14 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.393679 s
Pi is roughly 3.140097570048785
```

Como se puede ver en la imagen anterior, una vez termina de ejecutarse el job, el resultado que obtenemos es la frase “Pi is roughly” acompañada del número Pi con los dígitos que dimos como argumento de entrada en el comando de ejecución, por lo que se puede afirmar que la prueba fue ejecutada con éxito.

RESULTADOS:

Al finalizar, logramos tener un clúster de Spark funcional, con comunicación activa entre maestro y trabajadores, un sistema de archivos compartido correctamente montado y Spark desplegado en todos los nodos. Los resultados se pudieron comprobar en la práctica con la visualización de procesos en la terminal y con la interfaz gráfica de Spark, que mostró al maestro y los nodos conectados.

Un resultado importante fue darnos cuenta de cómo cada paso estaba encadenado con el siguiente: sin comunicación SSH no había coordinación, sin NFS no podíamos compartir Spark, sin variables de entorno Spark no funcionaba, y sin la configuración de `spark-env.sh` y `workers` el clúster no se levantaba. Esto nos permitió entender no solo cómo hacerlo, sino por qué hacerlo de esa manera.

CONCLUSIONES:

- El taller fue fundamental para entender cómo se levanta un clúster desde cero y cuáles son los elementos clave para que funcione.
- Nos dimos cuenta de que Spark depende de una buena organización inicial, sobre todo en la comunicación y el sistema de archivos compartido.
- Validamos la importancia de las variables de entorno y los archivos de configuración como base para que Spark pueda desplegarse.
- La práctica de levantar servicios y revisar su estado con herramientas como `jps` y la interfaz web nos da una visión muy cercana a lo que se hace en entornos de producción.
- En general, logramos cumplir con todo lo solicitado y quedamos con un aprendizaje sólido sobre cómo montar y validar un clúster de Spark.

BIBLIOGRAFÍA:

PySpark Overview — PySpark 4.0.1 documentation. Apache Spark.
<https://spark.apache.org/docs/latest/api/python/index.html>.