

Actividad 1. (0.5 puntos)

1.- INTRODUCCIÓN/JUSTIFICACIÓN DEL PROYECTO

Spark es un motor de procesamiento de datos distribuido y de código abierto diseñado para trabajar con grandes volúmenes de datos de manera eficiente y escalable. Fue desarrollado en la Universidad de California, Berkeley en el año 2009 y desde entonces ha ganado popularidad como una herramienta de procesamiento de datos para empresas de todos los tamaños.

La principal ventaja de Spark es su capacidad para procesar datos en memoria, lo que significa que puede manejar grandes cantidades de información mucho más rápido que otras herramientas de procesamiento de datos. Además, Spark es altamente escalable y puede manejar tareas de procesamiento de datos en clústeres de cientos o incluso miles de nodos.

Spark también ofrece una variedad de bibliotecas y herramientas para procesamiento de datos, incluyendo Spark SQL para consultas SQL en datos estructurados, Spark Streaming para procesamiento de datos en tiempo real, y MLlib para aprendizaje automático. En general, Spark es una herramienta poderosa y flexible para el procesamiento de datos que se ha convertido en una parte integral del ecosistema de big data.

Spark se basa en un motor computacional, lo que significa que se encarga de la programación, distribución y supervisión de la aplicación. Cada tarea se realiza en varios equipos de trabajo llamados clúster de computación. Un clúster informático hace referencia a la división de tareas. Una máquina realiza una tarea, mientras que las otras contribuyen a la salida final a través de una tarea diferente. Al final, todas las tareas se agregan para producir una salida. Hay varias razones por las que he decidido usar spark, están son unas de ellas:

- Mayor velocidad: Spark está diseñado para procesar datos en memoria, lo que lo hace mucho más rápido que los sistemas que utilizan discos duros para el almacenamiento. Esto permite que Spark procese grandes volúmenes de datos de manera más rápida que otros sistemas.
- Mayor escalabilidad: Spark está diseñado para funcionar en clústeres de computadoras, lo que lo hace altamente escalable. A medida que se agregan más nodos al clúster, Spark puede procesar más datos y aumentar la velocidad de procesamiento.
- Variedad de herramientas: Spark ofrece una variedad de bibliotecas y herramientas para procesamiento de datos, incluyendo Spark SQL para consultas SQL en datos estructurados, Spark Streaming para procesamiento de datos en tiempo real, y MLlib para aprendizaje automático. Esto hace que Spark sea una herramienta muy flexible para el procesamiento de datos.
- Fácil de usar: Spark se puede utilizar con varios lenguajes de programación,

incluyendo Scala, Java, Python y R, lo que lo hace más accesible para desarrolladores con diferentes antecedentes. Además, Spark tiene una amplia documentación y una comunidad activa, lo que facilita su aprendizaje y uso.

- Integración con otros sistemas: Spark se integra bien con otros sistemas de big data, como Hadoop y Cassandra. Esto significa que puede utilizarse junto con estas herramientas para aprovechar al máximo sus capacidades y ampliar las capacidades de procesamiento de datos.
- En resumen, Spark es una herramienta poderosa y flexible para el procesamiento de datos que ofrece una mayor velocidad, escalabilidad, variedad de herramientas, facilidad de uso e integración con otros sistemas de big data. Estas características hacen que Spark sea una excelente opción para empresas y organizaciones que necesitan procesar grandes volúmenes de datos de manera eficiente y escalable.

Actividad 2. (1 punto)

2.- INSTALACIÓN

Lo primero que haremos será ir a la página oficial de spark, y descargaremos la versión asociada a nuestra versión de hadoop, como tenemos la versión 3.3.4, elegiremos la de 3.3 or later

The screenshot shows a Firefox browser window titled "nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The address bar shows the URL <https://spark.apache.org/downloads.html>. The Apache Spark logo is at the top left. Below it, the main navigation menu includes "Download", "Libraries", "Documentation", "Examples", "Community", and "Developers". The "Download" section is active. A dropdown menu for "Choose a package type" is open, showing "Pre-built for Apache Hadoop 3.3 and later" as the selected option. Other options listed are "Pre-built for Apache Hadoop 3.3 and later (Scala 2.13)", "Pre-built for Apache Hadoop 2.7", and "Pre-built with user-provided Apache Hadoop Source Code". The text "Note that Spark 3 is pre-built" is visible below the dropdown. At the bottom, there is a note about Maven Central dependencies.

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Firefox

File Edit View History Bookmarks Tools Help

How To Install Apache Spark Spark Select The First Row Downloads | Apache Spark +

← → ⌂ ⌂ https://spark.apache.org/downloads.html

Centos Wiki Documentation Forums

APACHE Spark™ Download Libraries Documentation Examples Community

Download Apache Spark™

1. Choose a Spark release: 3.3.2 (Feb 17 2023) ▾
2. Choose a package type: Pre-built for Apache Hadoop 3.3 and later
3. Download Spark: [spark-3.3.2-bin-hadoop3.tgz](https://spark.apache.org/downloads.html)
4. Verify this release using the 3.3.2 signatures, checksums and project

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Firefox

File Edit View Bookmarks Tools Help

How To Install Apache Spark Spark Select The First Row Apache Downloads +

← → ⌂ ⌂ https://www.apache.org/dyn/closer.lua/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz

Centos Wiki Documentation Forums

News About Make a Donation The Apache Software Foundation

THE APACHE SOFTWARE FOUNDATION ESTABLISHED 1999

COMMUNITY-LED DEVELOPMENT

Projects People Community

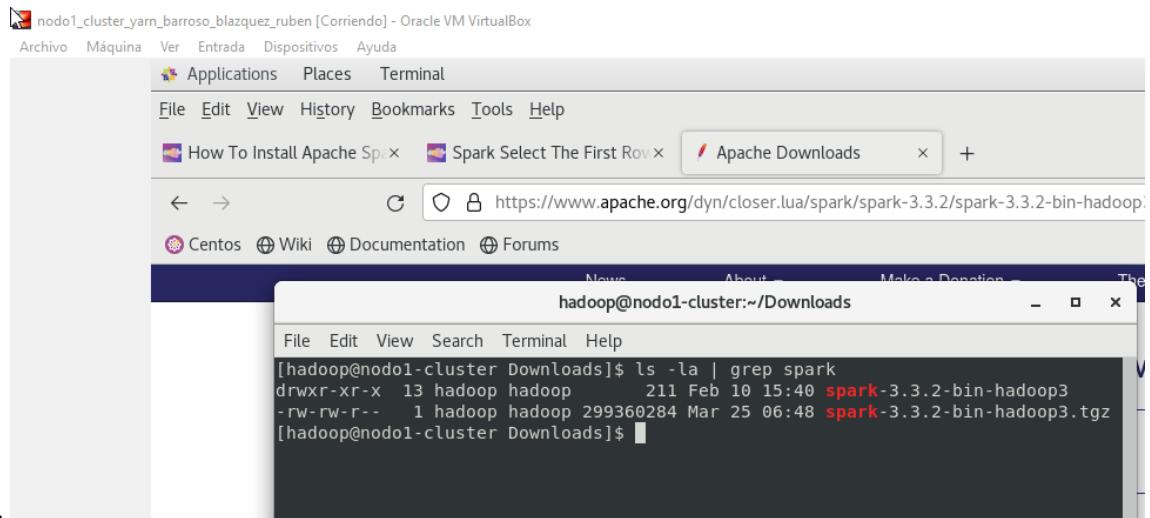
We suggest the following site for your download:
<https://dlcdn.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz>

Alternate download locations are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature (.asc file) c

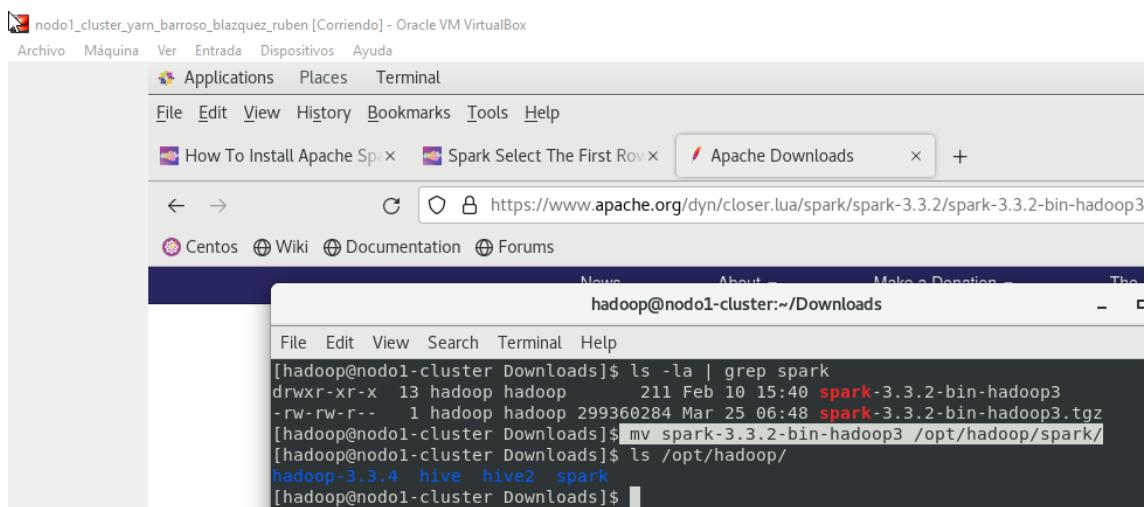
HTTP

Ahora nos vamos a descargas y descomprimimos la carpeta de spark con tar -xvf spark...

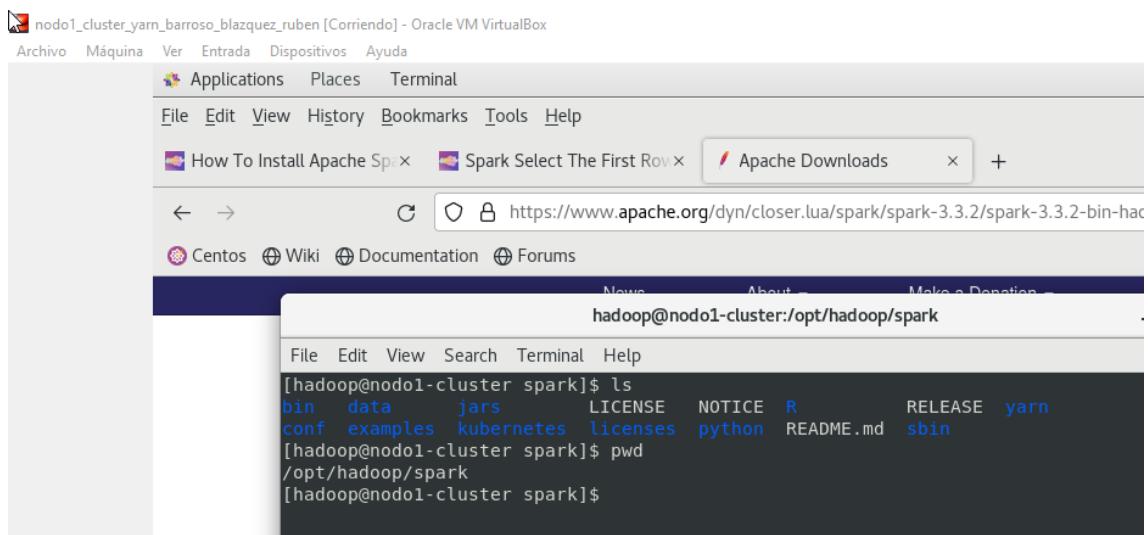


```
[hadoop@nodo1-cluster:~/Downloads]$ ls -la | grep spark
drwxr-xr-x 13 hadoop hadoop 211 Feb 10 15:40 spark-3.3.2-bin-hadoop3
-rw-rw-r-- 1 hadoop hadoop 299360284 Mar 25 06:48 spark-3.3.2-bin-hadoop3.tgz
```

Y la movemos a una carpeta spark , dentro de nuestro opt de hadoop

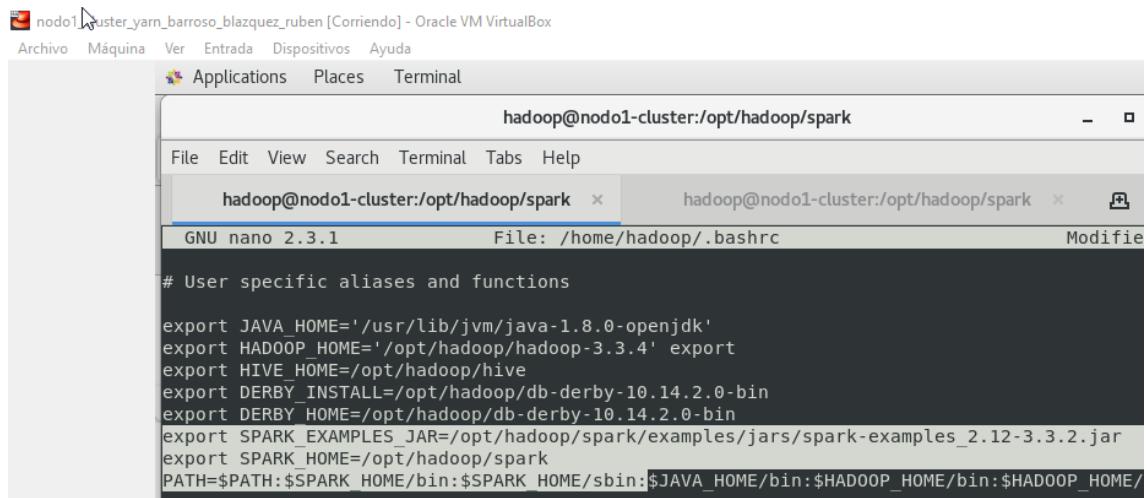


```
[hadoop@nodo1-cluster:~/Downloads]$ mv spark-3.3.2-bin-hadoop3 /opt/hadoop/spark/
[hadoop@nodo1-cluster:~/Downloads]$ ls /opt/hadoop/
hadoop-3.3.4  hive  hive2  spark
[hadoop@nodo1-cluster:~/Downloads]$
```



```
[hadoop@nodo1-cluster:/opt/hadoop/spark]$ ls
bin  data  jars  LICENSE  NOTICE  R  RELEASE  yarn
conf  examples  kubernetes  licenses  python  README.md  sbin
[hadoop@nodo1-cluster:spark]$ pwd
/opt/hadoop/spark
[hadoop@nodo1-cluster:spark]$
```

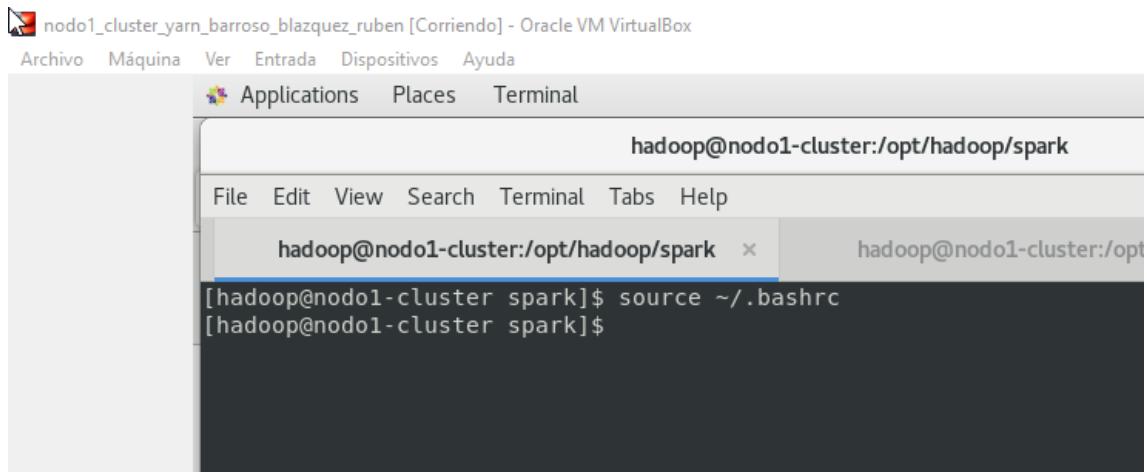
Ahora lo que haremos será establecer en el **.bashrc** los binarios de **spark** para poder acceder a ellos desde cualquier parte:



The screenshot shows a terminal window titled "nodo1_Cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The window has two tabs: "hadoop@nodo1-cluster:/opt/hadoop/spark" and "hadoop@nodo1-cluster:/opt/hadoop/.bashrc". The right tab is active, showing the contents of the .bashrc file. The file contains environment variable assignments for Java, Hadoop, Hive, Derby, and Spark, along with a PATH definition.

```
hadoop@nodo1-cluster:/opt/hadoop/.bashrc
File: /home/hadoop/.bashrc
GNU nano 2.3.1
# User specific aliases and functions
export JAVA_HOME='/usr/lib/jvm/java-1.8.0-openjdk'
export HADOOP_HOME='/opt/hadoop/hadoop-3.3.4' export
export HIVE_HOME=/opt/hadoop/hive
export DERBY_INSTALL=/opt/hadoop/db-derby-10.14.2.0-bin
export DERBY_HOME=/opt/hadoop/db-derby-10.14.2.0-bin
export SPARK_EXAMPLES_JAR=/opt/hadoop/spark/examples/jars/spark-examples_2.12-3.3.2.jar
export SPARK_HOME=/opt/hadoop/spark
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/
```

Y hacemos un source para confirmar los cambios en la terminal



The screenshot shows a terminal window titled "nodo1_Cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The window has two tabs: "hadoop@nodo1-cluster:/opt/hadoop/spark" and "hadoop@nodo1-cluster:/opt/hadoop/.bashrc". The left tab is active, showing the command "source ~/.bashrc" being run in the terminal.

```
[hadoop@nodo1-cluster spark]$ source ~/.bashrc
[hadoop@nodo1-cluster spark]$
```

Una vez hecho esto , podemos lanzar la Shell de spark con el comando **spark-shell**

Actividad 3. (1 punto)

3.- CONFIGURACIÓN

Lo primero que haremos será copiar las template de configuración sin el .template mediante la función **cp**



The screenshot shows a Linux desktop interface with a terminal window open. The terminal window title is "hadoop@nodo1-cluster:/opt/hadoop/spark/conf". The terminal content shows the user navigating to the configuration directory and listing files. The file "spark-env.sh.template" is highlighted in green.

```
[hadoop@nodo1-cluster spark]$ cd conf/
[hadoop@nodo1-cluster conf]$ ls
fairscheduler.xml.template    metrics.properties.template   spark-env.sh.template
log4j2.properties.template    spark-defaults.conf.template workers.template
[hadoop@nodo1-cluster conf]$
```

A screenshot of a terminal window titled "hadoop@nodo1-cluster:/opt/hadoop/spark/conf". The window has a standard Linux desktop interface with a menu bar at the top. The title bar shows the current directory as "/opt/hadoop/spark/conf". The menu bar includes "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". Below the menu bar, there are two tabs visible: "hadoop@nodo1-cluster:/opt/hadoop/spark/co..." and "hadoop@nodo1-cluster:/opt/hadoop/spark...". The main area of the terminal shows the command "[hadoop@nodo1-cluster conf]\$ cp fairscheduler.xml.template fairscheduler.xml" entered by the user.

```

[hadoop@nodo1-cluster conf]$ ls -la
total 68
drwxr-xr-x  2 hadoop hadoop 311 Mar 25 07:49 .
drwxr-xr-x 13 hadoop hadoop 211 Feb 10 15:40 ..
-rw-r--r--  1 hadoop hadoop 1105 Mar 25 07:48 fairscheduler.xml
-rw-r--r--  1 hadoop hadoop 1105 Feb 10 15:40 fairscheduler.xml.template
-rw-r--r--  1 hadoop hadoop 3350 Feb 10 15:40 log4j2.properties.template
-rw-r--r--  1 hadoop hadoop 9141 Mar 25 07:49 metrics.properties
-rw-r--r--  1 hadoop hadoop 9141 Feb 10 15:40 metrics.properties.template
-rw-r--r--  1 hadoop hadoop 1292 Mar 25 07:49 spark-defaults.conf
-rw-r--r--  1 hadoop hadoop 1292 Feb 10 15:40 spark-defaults.conf.template
-rw-r--r--  1 hadoop hadoop 4506 Mar 25 07:49 spark-env.sh
-rw-r--r--  1 hadoop hadoop 4506 Feb 10 15:40 spark-env.sh.template
-rw-r--r--  1 hadoop hadoop 865 Mar 25 07:49 workers
-rw-r--r--  1 hadoop hadoop 865 Feb 10 15:40 workers.template
[hadoop@nodo1-cluster conf]$

```

Una vez copiados los fichero sin template lo que haremos será configurar el fichero **spark_default.conf** y estableceremos el **spark.master** a **yarn**, y metemos la configuración de memoria de spark en los campos **spark.yarn.memory**, **spark.executor.memory**, y **spark.driver.memory**

```

[hadoop@nodo1-cluster:/opt/hadoop/spark/conf]$ nano spark-defaults.conf
GNU nano 2.3.1          File: spark-defaults.conf          Modified

# 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.
#
# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

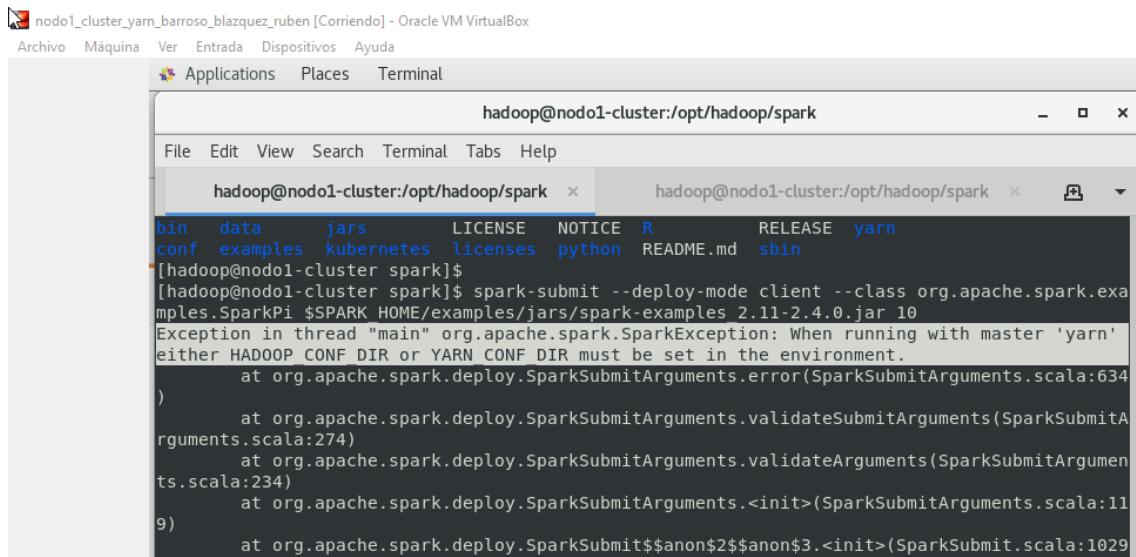
# Example:
spark.master          yarn
spark.yarn.am.memory  512m
spark.executor.memory 512m
spark.driver.memory   512m
# spark.eventLog.enabled    true

```

Una vez configurado podremos lanzar un ejemplo que viene por defecto en spark con el siguiente comando:

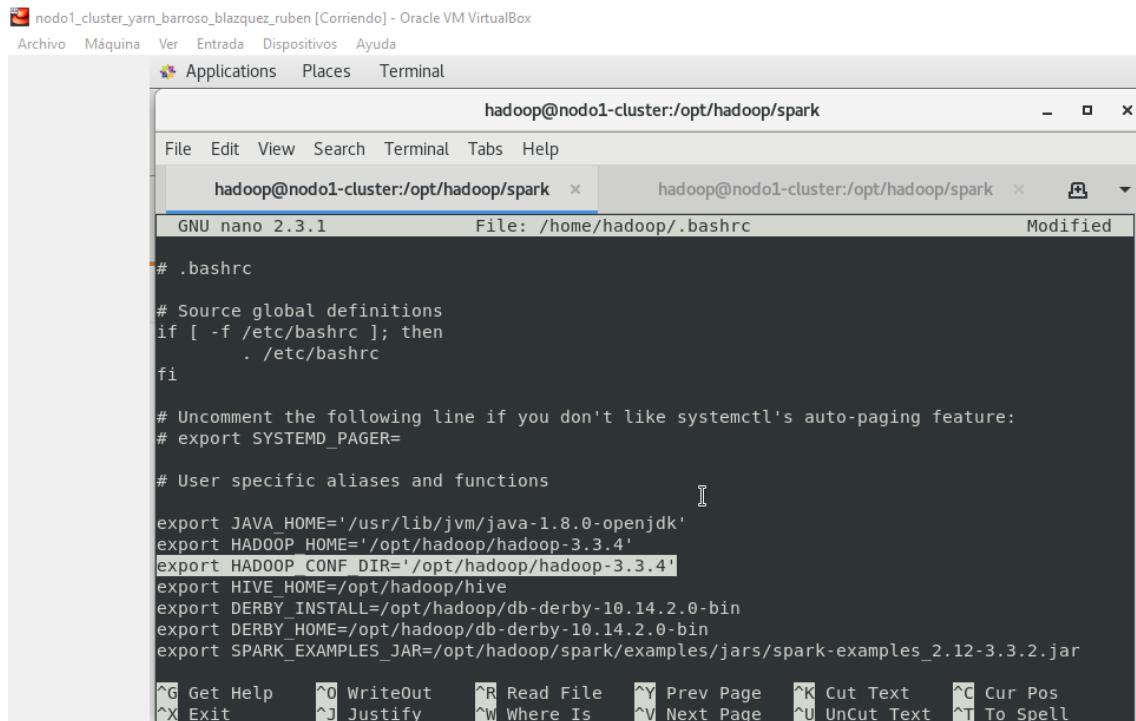
```
spark-submit --deploy-mode client --class org.apache.spark.examples.SparkPi $SPARK_EXAMPLES_JAR 10
```

Seguramente nos de este error



```
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark x hadoop@nodo1-cluster:/opt/hadoop/spark x
bin  data  jars  LICENSE  NOTICE  R  RELEASE  yarn
conf  examples  kubernetes  licenses  python  README.md  sbin
[hadoop@nodo1-cluster spark]$
[hadoop@nodo1-cluster spark]$ spark-submit --deploy-mode client --class org.apache.spark.examples.SparkPi $SPARK_HOME/examples/jars/spark-examples_2.11-2.4.0.jar 10
Exception in thread "main" org.apache.spark.SparkException: When running with master 'yarn' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
        at org.apache.spark.deploy.SparkSubmitArguments.error(SparkSubmitArguments.scala:634)
        at org.apache.spark.deploy.SparkSubmitArguments.validateSubmitArguments(SparkSubmitArguments.scala:274)
        at org.apache.spark.deploy.SparkSubmitArguments.validateArguments(SparkSubmitArguments.scala:234)
        at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.scala:119)
        at org.apache.spark.deploy.SparkSubmit$$anon$2$anon$3.<init>(SparkSubmit.scala:1029)
```

Para ello tenemos que definir en el bashrc además del **Hadoop_Home** el **HADOOP_CONF_DIR**, que es la misma ruta que tengamos el **hadoop_home**



```
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark x hadoop@nodo1-cluster:/opt/hadoop/spark x
GNU nano 2.3.1          File: /home/hadoop/.bashrc          Modified
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
export JAVA_HOME='/usr/lib/jvm/java-1.8.0-openjdk'
export HADOOP_HOME='/opt/hadoop/hadoop-3.3.4'
export HADOOP_CONF_DIR='/opt/hadoop/hadoop-3.3.4'
export HIVE_HOME=/opt/hadoop/hive
export DERBY_INSTALL=/opt/hadoop/db-derby-10.14.2.0-bin
export DERBY_HOME=/opt/hadoop/db-derby-10.14.2.0-bin
export SPARK_EXAMPLES_JAR=/opt/hadoop/spark/examples/jars/spark-examples_2.12-3.3.2.jar
```

Una vez configurado dicho parámetro, lanzamos el comando del jar de prueba

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Terminal

hadoop@nodo1-cluster:/opt/hadoop/spark

```
hadoop@nodo1-cluster:~/opt/hadoop/spark$ spark-submit --deploy-mode client --class org.apache.spark.examples.SparkPi $SPARK EXAMPLES JAR 10
23/03/25 08:05:48 INFO SparkContext: Running Spark version 3.3.2
23/03/25 08:05:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
23/03/25 08:05:48 INFO ResourceUtils: =====
=====
23/03/25 08:05:48 INFO ResourceUtils: No custom resources configured for spark.driver.
23/03/25 08:05:48 INFO ResourceUtils: =====
=====
23/03/25 08:05:48 INFO SparkContext: Submitted application: Spark Pi
23/03/25 08:05:48 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 512, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
23/03/25 08:05:48 INFO ResourceProfile: Limiting resource is cpus at 1 tasks per executor
23/03/25 08:05:48 INFO ResourceManager: Added ResourceProfile id: 0
23/03/25 08:05:49 INFO SecurityManager: Changing view acls to: hadoop
23/03/25 08:05:49 INFO SecurityManager: Changing modify acls to: hadoop
23/03/25 08:05:49 INFO SecurityManager: Changing view acls groups to:
23/03/25 08:05:49 INFO SecurityManager: Changing modify acls groups to:
23/03/25 08:05:49 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop); groups with view permissions: Set(); users with modify permissions: Set(hadoop); groups with modify permissions: Set()
23/03/25 08:05:49 INFO Utils: Successfully started service 'sparkDriver' on port 40773.
23/03/25 08:05:49 INFO SparkEnv: Registering MapOutputTracker
23/03/25 08:05:49 INFO SparkEnv: Registering BlockManagerMaster
23/03/25 08:05:49 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
23/03/25 08:05:49 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
23/03/25 08:05:49 INFO SecurityManager: Changing modify acls groups to:
23/03/25 08:05:49 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop); groups with view permissions: Set(); users with modify permissions: Set(hadoop); groups with modify permissions: Set()
23/03/25 08:05:49 INFO Utils: Successfully started service 'sparkDriver' on port 40773.
23/03/25 08:05:49 INFO SparkEnv: Registering MapOutputTracker
23/03/25 08:05:49 INFO SparkEnv: Registering BlockManagerMaster
23/03/25 08:05:49 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTop
```

```
[hadoop@nodo1-cluster jars]$ jps
10900 NameNode
22680 Jps
12714 RunJar
11148 SecondaryNameNode
22396 SparkSubmit
11389 ResourceManager
[hadoop@nodo1-cluster jars]$
```

y podemos ver como en nuestro servicio de hadoop yarn tenemos el job de spark

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime
application_1679738497453_0002	hadoop	Spark Pi	SPARK		default	0	Sat Mar 25 08:06:09 -0400 2023	N/A	N/A

Ahora vamos a configurar el **HistoryServer de spark**.

Lo primero que haremos será modificar el fichero **spark_defaults.conf** y añadiremos las siguientes líneas:

spark.eventLog.enabled true

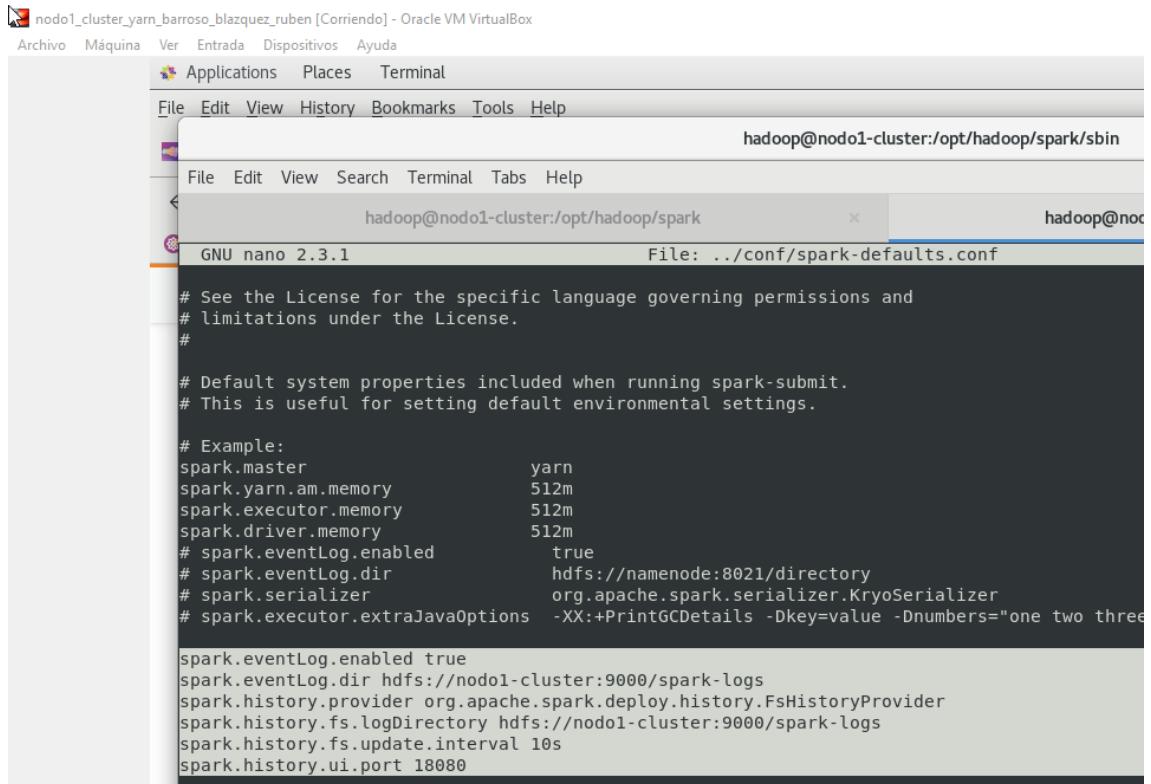
```
spark.eventLog.dir hdfs://namenode:9000/spark-logs
```

```
spark.history.provider org.apache.spark.deploy.history.FsHistoryProvider
```

```
spark.history.fs.logDirectory hdfs://namenode:9000/spark-logs
```

```
spark.history.fs.update.interval 10s
```

```
spark.history.ui.port 18080
```



```
hadoop@nodo1-cluster:/opt/hadoop/spark/sbin
```

```
File Edit View History Bookmarks Tools Help
```

```
hadoop@nodo1-cluster:/opt/hadoop/spark
```

```
hadoop@nodo1-cluster:~
```

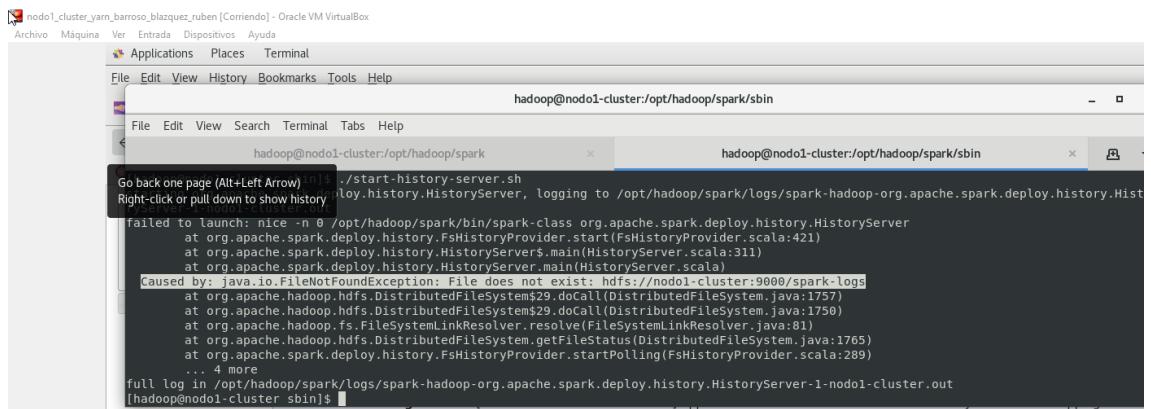
```
GNU nano 2.3.1 File: ../conf/spark-defaults.conf
```

```
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

# Example:
spark.master yarn
spark.yarn.am.memory 512m
spark.executor.memory 512m
spark.driver.memory 512m
# spark.eventLog.enabled true
# spark.eventLog.dir hdfs://namenode:8021/directory
# spark.serializer org.apache.spark.serializer.KryoSerializer
# spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"

spark.eventLog.enabled true
spark.eventLog.dir hdfs://nodo1-cluster:9000/spark-logs
spark.history.provider org.apache.spark.deploy.history.FsHistoryProvider
spark.history.fs.logDirectory hdfs://nodo1-cluster:9000/spark-logs
spark.history.fs.update.interval 10s
spark.history.ui.port 18080
```

Y lanzamos el history server con el siguiente binario:



```
hadoop@nodo1-cluster:/opt/hadoop/spark/sbin
```

```
File Edit View History Bookmarks Tools Help
```

```
hadoop@nodo1-cluster:/opt/hadoop/spark
```

```
hadoop@nodo1-cluster:~
```

```
./start-history-server.sh
```

```
Go back one page (Alt+Left Arrow) [1/1] Right-click or pull down to show history.
```

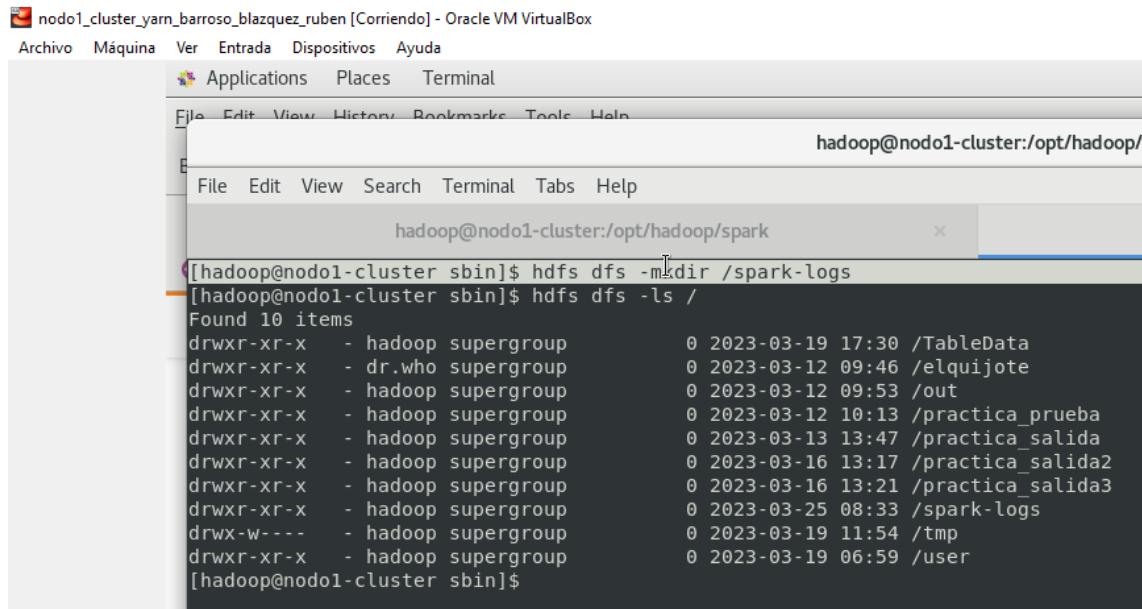
```
Failed to launch: nice -n 0 /opt/hadoop/spark/bin/spark-class org.apache.spark.deploy.history.HistoryServer
at org.apache.spark.deploy.history.FsHistoryProvider.start(FsHistoryProvider.scala:421)
at org.apache.spark.deploy.history.HistoryServers$.main(HistoryServer.scala:311)
at org.apache.spark.deploy.history.HistoryServer.main(HistoryServer.scala)
```

```
Caused by: java.io.FileNotFoundException: File does not exist: hdfs://nodo1-cluster:9000/spark-logs
at org.apache.hadoop.hdfs.DistributedFileSystem$29.doCall(DistributedFileSystem.java:1757)
at org.apache.hadoop.hdfs.DistributedFileSystem$29.doCall(DistributedFileSystem.java:1750)
at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
at org.apache.hadoop.hdfs.DistributedFileSystem.getFileStatus(DistributedFileSystem.java:1765)
at org.apache.spark.deploy.history.FsHistoryProvider.startPolling(FsHistoryProvider.scala:289)
... 4 more
```

```
full log in /opt/hadoop/spark/logs/spark-hadoop-org.apache.spark.deploy.history.HistoryServer-1-nodo1-cluster.out
```

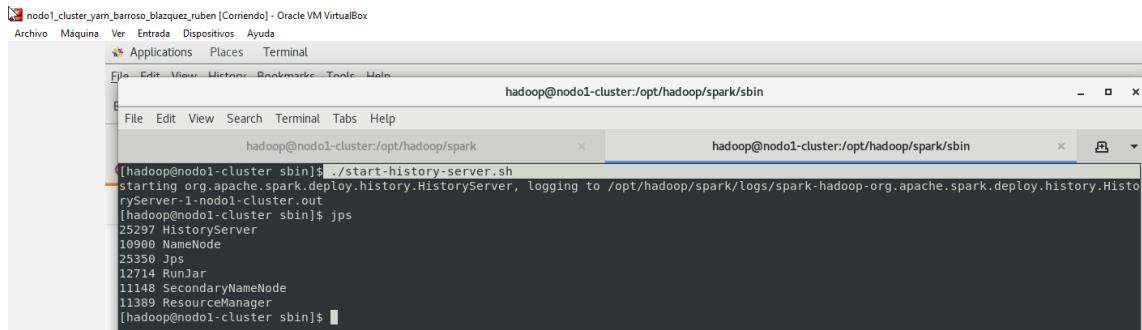
```
[hadoop@nodo1-cluster sbin]$
```

Como podemos observar nos da un error de que no encuentra spark-logs, para corregir este error debemos ir a nuestro hdfs y crear la carpeta **spark-logs**



```
[hadoop@nodo1-cluster sbin]$ hdfs dfs -mkdir /spark-logs
[hadoop@nodo1-cluster sbin]$ hdfs dfs -ls /
Found 10 items
drwxr-xr-x  - hadoop supergroup          0 2023-03-19 17:30 /TableData
drwxr-xr-x  - dr.who supergroup          0 2023-03-12 09:46 /elquijote
drwxr-xr-x  - hadoop supergroup          0 2023-03-12 09:53 /out
drwxr-xr-x  - hadoop supergroup          0 2023-03-12 10:13 /practica_prueba
drwxr-xr-x  - hadoop supergroup          0 2023-03-13 13:47 /practica_salida
drwxr-xr-x  - hadoop supergroup          0 2023-03-16 13:17 /practica_salida2
drwxr-xr-x  - hadoop supergroup          0 2023-03-16 13:21 /practica_salida3
drwxr-xr-x  - hadoop supergroup          0 2023-03-25 08:33 /spark-logs
drwxr-w---  - hadoop supergroup          0 2023-03-19 11:54 /tmp
drwxr-xr-x  - hadoop supergroup          0 2023-03-19 06:59 /user
[hadoop@nodo1-cluster sbin]$
```

Y volvemos a lanzar el history server, ahora ya vemos con jps como se ha lanzado correctamente



```
[hadoop@nodo1-cluster sbin]$ ./start-history-server.sh
starting org.apache.spark.deploy.history.HistoryServer, logging to /opt/hadoop/spark/logs/spark-hadoop-org.apache.spark.deploy.history.HistoryServer-1-nodo1-cluster.out
[hadoop@nodo1-cluster sbin]$ jps
25297 HistoryServer
10980 NameNode
25350 Jps
12714 RunJar
11148 SecondaryNameNode
11389 ResourceManager
[hadoop@nodo1-cluster sbin]$
```

Y podemos ir a la interfaz gráfica que nos levanta dicho binario

The screenshot shows a Firefox browser window with the following details:

- Title Bar:** nodo1_cluster_yern_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
- Menu Bar:** Archivo, Máquina, Ver, Entrada, Dispositivos, Ayuda, Applications, Places, Firefox
- Toolbar:** File, Edit, View, History, Bookmarks, Tools, Help
- Address Bar:** nodo1-cluster:18080
- Content Area:**
 - Section:** History Server
 - Text:** Event log directory: hdfs://nodo1-cluster:9000/spark-logs
 - Text:** Last updated: 2023-03-25 10:01:28
 - Text:** Client local time zone: America/New_York
 - Table:** Shows a single completed application entry.

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.3.2	application_1679749067710_0001	Spark Pi	2023-03-25 09:59:54	2023-03-25 10:00:18	24 s	hadoop	2023-03-25 10:00:18	Download

Showing 1 to 1 of 1 entries
[Show incomplete applications](#)

Se puede observar la última ejecución realizada que es el ejemplo que hemos hecho anteriormente

Pudiendo ver la información de dicho job clickando en el APP ID

The screenshot shows a Firefox browser window with the following details:

- Title Bar:** nodo1_cluster_yern_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
- Menu Bar:** Archivo, Máquina, Ver, Entrada, Dispositivos, Ayuda, Applications, Places, Firefox
- Toolbar:** File, Edit, View, History, Bookmarks, Tools, Help
- Address Bar:** nodo1-cluster:18080/history/application_1679749067710_0001/jobs/
- Content Area:**
 - Section:** Spark Pi - Spark Job
 - Sub-section:** Spark Jobs (1)
 - Text:** User: hadoop
Total Uptime: 24 s
Scheduling Mode: FIFO
Completed Jobs: 1
 - Links:** Event Timeline, Completed Jobs (1)
 - Table:** Shows a completed job entry.
 - Page Navigation:** Page: 1 / 1 Pages. Jump to 1, Show 100 items in a page, Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	reduce at SparkPi.scala:38 reduce at SparkPi.scala:38	2023/03/25 10:00:17	1 s	1/1	10/10

Page: 1 / 1 Pages. Jump to 1, Show 100 items in a page, Go

Podríamos ver también el flujo de grafos que han seguido los datos en nuestro proceso mediante la pestaña Stage

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Firefox

File Edit View Bookmarks Tools Help

How To Install Apache Spark Select The Fi Browsing HDFS Apache Spark Tutorial Spark Step-by-Step History Server Spark Pi - Details

Centos Wiki Documentation Forums

Spark 3.3.2 Jobs Stages Storage Environment Executors

Details for Stage 0 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 0.4 s
Locality Level Summary: Process local: 10
Associated Job IDs: 0

DAG Visualization

```

graph TD
    subgraph Stage0 [Stage 0]
        direction TB
        A[parallelize] --> B["ParallelCollectionRDD [0]  
parallelize at SparkPi.scala:34"]
        B --> C["map"]
        C --> D["MapPartitionsRDD [1]  
map at SparkPi.scala:34"]
    end

```

Show Additional Metrics Event Timeline

Summary Metrics for 10 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile
Duration	6.0 ms	10.0 ms	17.0 ms	33.0 ms
GC Time	0.0 ms	0.0 ms	0.0 ms	0.0 ms

Aggregated Metrics by Executor

También podemos ver las variables almacenadas en memoria

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Sat 17:36 • 44

Applications Places Firefox

File Edit View Bookmarks Tools Help

How To Install Apache Spark Select The Fi Browsing HDFS Apache Spark Tutorial Spark Step-by-Step History Server Spark Pi - Storage Data Types - Spark (sin asunto) - ruben

Centos Wiki Documentation Forums

Spark 3.3.2 Jobs Stages Storage Environment Executors

Storage

En nuestro caso no tenemos ninguna

Variables de entorno:

Environment

Runtime Information

Name	Value
Java Home	/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.362.b08-1.el7_9.x86_64/jre
Java Version	1.8.0_362 (Red Hat, Inc.)
Scala Version	version 2.12.15

Spark Properties

Name	Value
spark.app.id	application_1679749067710_0001
spark.app.initialJarUrls	spark://nodo1-cluster:34661/jars/spark-examples_2_12-3.3.2.jar
spark.app.name	Spark Pi
spark.app.startTime	1679752794583
spark.app.submitTime	1679752794535
spark.driver.appUAddress	http://nodo1-cluster:4040
spark.driver.extraJavaOptions	-XX:+IgnoreUnrecognizedVMOptions -add-opens=java.base/java.lang:ALL-UNNAMED -add-opens=java.base/java.lang.invoke:ALL-UNNAMED -add-opens=java.base/java.lang.reflect:ALL-UNNAMED -add-opens=java.base/java.io:ALL-UNNAMED -add-opens=java.base/java.net:ALL-UNNAMED -add-opens=java.base/java.util.concurrent.atomic:ALL-UNNAMED -add-opens=java.base/sun.nio.cs:ALL-UNNAMED -add-opens=java.base/sun.security.action:ALL-UNNAMED -add-opens=java.base/sun.util.calendar:ALL-UNNAMED -add-opens=java.security.gss/sun.security.krb5:ALL-UNNAMED

y la información de los ejecutadores

node1_cluster_yarn_barneo_blanquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Firefox

Sat 17:38 •

File Edit View History Bookmark Tools Help

How To Install Apache Spark Select The F x Browsing HDFS x Apache Spark Tuto x Spark Step-by-Step x History Server x Spark Pi - Executors x Data Types - Spark x M (sin asunto) - ruben x

← → C nodo1-cluster:18080/history/application_1679749067710_0001/executors/

Centos Wiki Documentation Forums

 3.3.2 Jobs Stages Storage Environment Executors

Spark Pi application U

Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(3)	0	0.0 B / 279.9 MiB	0.0 B	2	0	0	10	10	26 s (25.0 ms)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	0	0.0 B / 279.9 MiB	0.0 B	2	0	0	10	10	26 s (25.0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	nodo1-cluster:36309	Active	0	0.0 B / 93.3 MiB	0.0 B	0	0	0	0	0	24 s (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	nodo1-cluster:39700	Active	0	0.0 B / 93.3 MiB	0.0 B	1	0	0	6	6	1 s (10.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
2	nodo1-cluster:32905	Active	0	0.0 B / 93.3 MiB	0.0 B	1	0	0	4	4	1 s (15.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr

Showing 1 to 3 of 3 entries Previous Next

Miscellaneous Process

Show 20 entries Search:

Process ID	Address	Status	Cores	Loss
------------	---------	--------	-------	------

Actividad 4. (1 punto)

4.- PRUEBA DE FUNCIONAMIENTO BÁSICO

Para explicar el funcionamiento básico de spark, veremos spark SQL, esta sería la estructura principal que usaría:



Lo primero que haremos será leer un fichero CSV

Para ello lo primero que haremos será abrir la consola de spark con **spark-shell**:

Para leer un fichero hay 2 maneras, con esquema o sin esquema, primero lo haremos sin especificar ninguno, usando el siguiente código de Scala, cabe destacar que como nuestro sistema está conectado a hadoop hdfs el fichero al que querramos acceder debe estar ahí guardado, en nuestro caso está en /spark_files

Browse Directory

/spark_files

Show 25 entries

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	dr.who	supergroup	1.22 MB	Mar 25 11:25	2	128 MB	housing.csv

Showing 1 to 1 of 1 entries

Go!

Search:

Previous **1** Next

```
val df = spark.read_format('csv').option('header',true).load('/spark_files/housing.csv')
```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark/examples/jars      hadoop@nodo1-cluster:/opt/hadoop/spark
scala> val df= spark.read.format("csv").option("header", "true").load("/spark files/housing.csv")
df: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 8 more fields]
scala>

```

Y podemos ver el contenido con **df.show**

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark/examples/jars      hadoop@nodo1-cluster:/opt/hadoop/spark
scala> val df= spark.read.format("csv").option("header", "true").load("/spark files/housing.csv")
df: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 8 more fields]
scala> df.show
+-----+-----+-----+-----+-----+-----+-----+-----+
|latitud|longitud|precio_casa|media_casa|años_casa|precio_medio|distancia_c|distancia_d|distancia_mar|distancia_e|
+-----+-----+-----+-----+-----+-----+-----+-----+
|-122.23| 37.88| 41.0| 880.0| 129.0| 322.0| 126.0| 8.3252| 452600.0| NEAR BAY|
|-122.22| 37.86| 21.0| 7099.0| 1106.0| 2401.0| 1138.0| 8.3014| 358500.0| NEAR BAY|
|-122.24| 37.85| 52.0| 1467.0| 190.0| 496.0| 177.0| 7.2574| 352100.0| NEAR BAY|
|-122.25| 37.85| 52.0| 1274.0| 235.0| 558.0| 219.0| 5.6431| 341300.0| NEAR BAY|
|-122.25| 37.85| 52.0| 1627.0| 286.0| 565.0| 259.0| 3.8462| 342200.0| NEAR BAY|
|-122.25| 37.85| 52.0| 919.0| 213.0| 413.0| 193.0| 4.0368| 269700.0| NEAR BAY|
|-122.25| 37.84| 52.0| 2535.0| 489.0| 1094.0| 514.0| 3.6591| 299200.0| NEAR BAY|
|-122.25| 37.84| 52.0| 3104.0| 687.0| 1157.0| 647.0| 3.12| 241400.0| NEAR BAY|
|-122.26| 37.84| 42.0| 2555.0| 665.0| 1206.0| 595.0| 2.0804| 226700.0| NEAR BAY|
|-122.25| 37.84| 52.0| 3549.0| 707.0| 1551.0| 714.0| 3.6912| 261100.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2282.0| 434.0| 910.0| 402.0| 3.2031| 281500.0| NEAR BAY|
|-122.26| 37.85| 52.0| 3503.0| 752.0| 1504.0| 734.0| 3.2705| 241800.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2491.0| 474.0| 1098.0| 468.0| 3.075| 213500.0| NEAR BAY|
|-122.26| 37.84| 52.0| 696.0| 191.0| 345.0| 174.0| 2.6736| 191300.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2643.0| 626.0| 1212.0| 620.0| 1.9167| 159200.0| NEAR BAY|
|-122.26| 37.85| 50.0| 1120.0| 283.0| 697.0| 264.0| 2.125| 140000.0| NEAR BAY|

```

Ahora cargaremos el mismo dataset, pero con una estructura predefinida:

Lo primero que tenemos que hacer es importar la librería de tipos de datos, para ello lanzaremos el siguiente comando en la consola de spark

Import org.apache.spark.sql.types._

Una vez importado, creamos nuestra estructura

```

val esquemaPersonalizado =
StructType(Array(StructField("latitud",DoubleType),StructField("longitud",
DoubleType),StructField("precio_casa",DoubleType),StructField("media_casa",
DoubleType),StructField("años_casa", DoubleType) ,StructField("precio_medio",
DoubleType) ,StructField("distancia_c", DoubleType) ,StructField("distancia_d",
DoubleType) ,StructField("distancia_mar", DoubleType) ,StructField("distancia_e",
DoubleType)))

```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark
hadoop@nodo1-cluster:/opt/hadoop/spark

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> val esquemaPersonalizado = StructType(Array(StructField("latitud",DoubleType),StructField("longitud", DoubleType),StructField("precio_casa",DoubleType),StructField("media_casa"
, DoubleType),StructField("ahos_casa", DoubleType) ,StructField("precio_medio", DoubleType) ,StructField("distancia_c", DoubleType) ,StructField("distancia_d", DoubleType) ,StructField(
"distancia_e", DoubleType)))
esquemaPersonalizado: org.apache.spark.sql.types.StructType = StructType(StructField(latitud,DoubleType,true),StructField(longitud,DoubleType,true),StructField(precio_casa,DoubleType,
true),StructField(media_casa,DoubleType,true),StructField(ahos_casa,DoubleType,true),StructField(precio_medio,DoubleType,true),StructField(distancia_c,DoubleType,true),StructField(dis-
tancia_d,DoubleType,true),StructField(distancia_mar,DoubleType,true),StructField(distancia_e,DoubleType,true))

scala>

```

y dicha estructura se la pasamos a la lectura del fichero realizada anteriormente

```

val df =
spark.read_format('csv').schema(esquemaPersonalizado).option('header',true).load('/
spark_files/housing.csv')

```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:/opt/hadoop/spark
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:/opt/hadoop/spark/examples/jars
hadoop@nodo1-cluster:/opt/hadoop/spark

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> val esquemaPersonalizado = StructType(Array(StructField("latitud",DoubleType),StructField("longitud", DoubleType),StructField("precio_casa",DoubleType),
DoubleType),StructField("ahos_casa", DoubleType) ,StructField("precio_medio", DoubleType) ,StructField("distancia_c", DoubleType) ,StructField("distancia_d", DoubleType) ,
StructField("distancia_e", DoubleType)))
esquemaPersonalizado: org.apache.spark.sql.types.StructType = StructType(StructField(latitud,DoubleType,true),StructField(longitud,DoubleType,true),StructField(precio_casa,DoubleType,
true),StructField(media_casa,DoubleType,true),StructField(ahos_casa,DoubleType,true),StructField(precio_medio,DoubleType,true),StructField(distancia_c,DoubleType,true),StructField(dis-
tancia_d,DoubleType,true),StructField(distancia_mar,DoubleType,true),StructField(distancia_e,DoubleType,true))

scala> val df = spark.read.format("csv").schema(esquemaPersonalizado).option("header", "true").load("/spark_files/housing.csv")
df_with_schema: org.apache.spark.sql.DataFrame = [latitud: double, longitud: double ... 8 more fields]

scala>

```

Como podemos observar ahora si le pone los tipos de datos que queremos al dataset obtenido del csv, en el primer ejemplo los ponía todos como strings

Una vez realizadas una operación de lectura, vamos a ver una de escritura, esta no cambia prácticamente de la de lectura, solo hay que cambiar el read por un write:

```
df_with_schema.write.format("json").save("/spark_files/housingJSON ")
```

cabe destacar que esto crea una carpeta y dentro de esa carpeta estará nuestro json

```

hadoop@nodo1-cluster:/opt/hadoop/spark/examples/jars
scala> df.withSchema.write.format("json").save("/spark_files/housing.json")
scala>

```

Y si vemos la carpeta en el hdfs...

Name	Type	Size	Last Modified
housing.csv	File	1.22 MB	Mar 25 11:28
housingDirectory	Directory	0 B	Mar 25 11:56
housingSQL	File	0 B	Mar 25 11:56

Name	Type	Size	Last Modified
part-00000-60af71cc-db87-4528-a6d3-bb73c0be74d9-c000.json	File	3368106	Mar 25 11:56

Block Information

- Block ID: 1073741854
- Block Pool ID: BP-1716972341-192.168.2.101-167862866695
- Generation Stamp: 1030
- Size: 3368106

Availability:

- nodo2-cluster
- nodo3-cluster

File contents

```

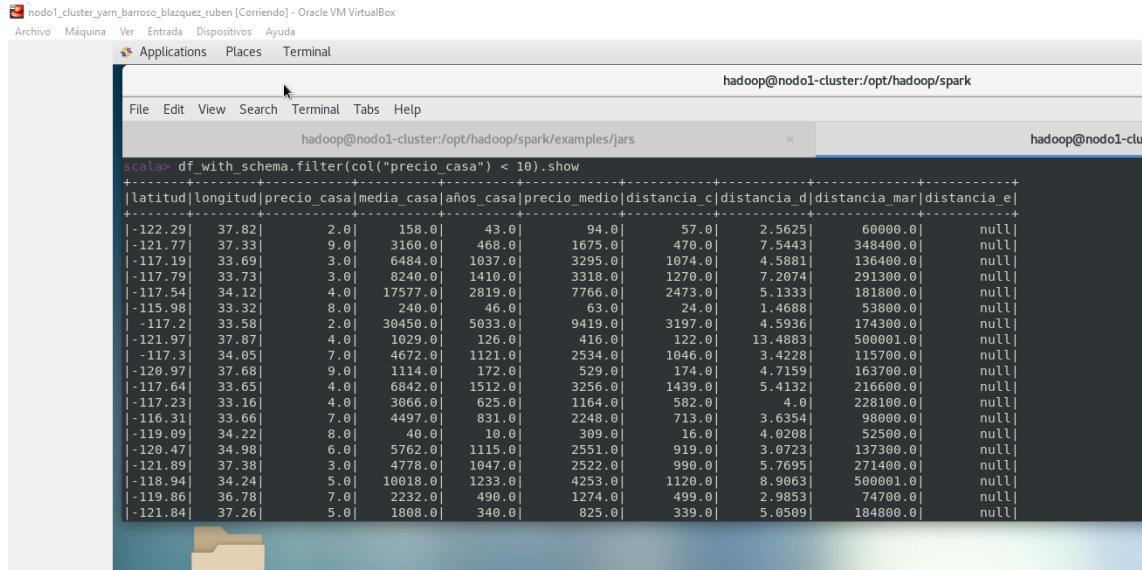
{"latitud":122.23,"longitud":37.88,"precio_casa":41.0,"media_casa":880.0,"años_casa":129.0,"precio_medio":322.0,"distancia_c":126.0,"distancia_d":8.3252,"distancia_mar":4526.0}
{"latitud":122.22,"longitud":37.86,"precio_casa":21.0,"media_casa":7099.0,"años_casa":1106.0,"precio_medio":2401.0,"distancia_c":1138.0,"distancia_d":8.3014,"distancia_mar":358500.0}

```

Filtrado de información:

Para filtrar información podemos usar el método filter sobre nuestro dataframe

```
df_with_schema.filter(col("precio_casa") <= 10)
```



The screenshot shows a terminal window titled "nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The command executed is:

```
scala> df_with_schema.filter(col("precio_casa") < 10).show
```

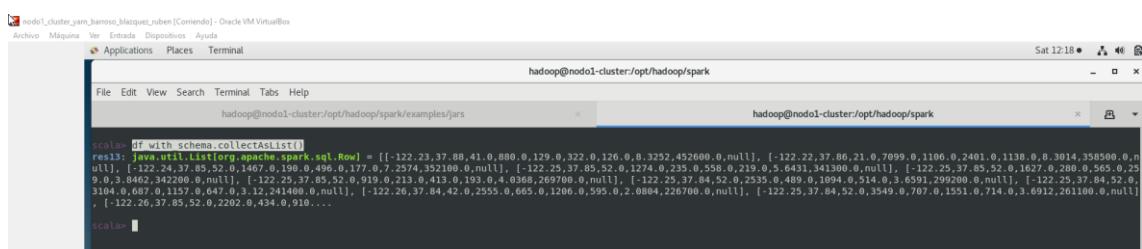
The output displays a filtered DataFrame with the following schema and data:

latitud	longitud	precio_casa	media_casa	años_casa	precio_medio	distancia_c	distancia_d	distancia_mar	distancia_e
-122.29	37.82	2.0	158.0	43.0	94.0	57.0	2.5625	60000.0	null
-121.77	37.33	9.0	3160.0	468.0	1675.0	470.0	7.5443	348400.0	null
-117.19	33.69	3.0	6484.0	1037.0	3295.0	1074.0	4.5881	136400.0	null
-117.79	33.73	3.0	8240.0	1410.0	3318.0	1270.0	7.2074	291300.0	null
-117.54	34.12	4.0	17577.0	2819.0	7766.0	2473.0	5.1333	181800.0	null
-115.98	33.32	8.0	240.0	46.0	63.0	24.0	1.4688	53800.0	null
-117.2	33.58	2.0	30450.0	5033.0	9419.0	3197.0	4.5936	174300.0	null
-121.97	37.87	4.0	1029.0	126.0	416.0	122.0	13.4883	500001.0	null
-117.3	34.05	7.0	4672.0	1121.0	2534.0	1046.0	3.4228	115700.0	null
-120.97	37.68	9.0	1114.0	172.0	529.0	174.0	4.7159	163700.0	null
-117.64	33.65	4.0	6842.0	1512.0	3256.0	1439.0	5.4132	216600.0	null
-117.23	33.16	4.0	3066.0	625.0	1164.0	582.0	4.0	228100.0	null
-116.31	33.66	7.0	4497.0	831.0	2248.0	713.0	3.6354	98000.0	null
-119.09	34.22	8.0	40.0	10.0	389.0	16.0	4.0208	52500.0	null
-120.47	34.98	6.0	5762.0	1115.0	2551.0	919.0	3.0723	137300.0	null
-121.89	37.38	3.0	4778.0	1047.0	2522.0	990.0	5.7695	271400.0	null
-118.94	34.24	5.0	10018.0	1233.0	4253.0	1120.0	8.9063	500001.0	null
-119.86	36.78	7.0	2232.0	490.0	1274.0	499.0	2.9853	74700.0	null
-121.84	37.26	5.0	1808.0	340.0	825.0	339.0	5.0509	184800.0	null

Hay que tener en cuenta que cuando hacemos el filtro esto devuelve un dataframe que para mostrar su información debemos hacer el .show

Transformar Dataframe en Lista

método .collectAsList()



The screenshot shows a terminal window titled "nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The command executed is:

```
scala> df with schema.collectAsList()
```

The output displays the collected data as a list of Row objects:

```
res13: java.util.List[org.apache.spark.sql.Row] = [[-122.23,37.88,41.0,880.0,129.0,322.0,126.0,8.3252,452600.0,null], [-122.22,37.86,21.0,7099.0,1106.0,2401.0,1138.0,8.3014,358500.0,null], [-122.24,37.85,52.0,1467.0,198.0,496.0,177.0,7.2574,352100.0,null], [-122.25,37.85,52.0,1274.0,235.0,558.0,219.0,5.0431,341300.0,null], [-122.25,37.85,52.0,1627.0,6.280.0,565.0,0.259.0,3.8462,342200.0,null], [-122.25,37.85,52.0,919.0,213.0,413.0,193.0,4.0368,269700.0,null], [-122.25,37.84,52.0,2353.0,489.0,1094.0,514.0,3.6591,299200.0,null], [-122.25,37.84,52.0,3194.0,687.0,1157.0,12241400.0,null], [-122.26,37.84,42.0,2555.0,665.0,1266.0,595.0,2.0804,226700.0,null], [-122.25,37.84,52.0,3549.0,707.0,1551.0,714.0,3.6912,261100.0,null], [-122.26,37.85,52.0,2292.0,434.0,910.0,...]]
```

Selección de información

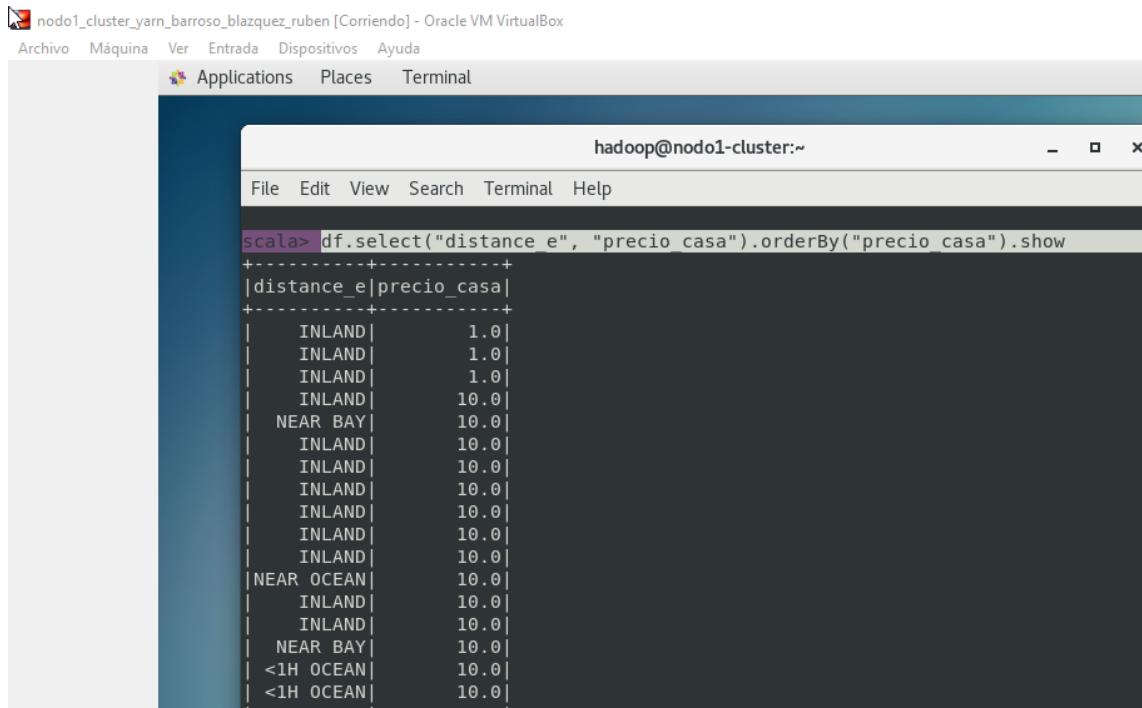
Para mapear información lo primero que deberemos tener es una secuencia de valores, para ello lo que haremos será obtener una columna de nuestro dataframe con el comando:

```
val sequence = df.select("distance_e")
```

Ordenación de un dataframe

Para ordenar un dataframe se puede usar la función **orderBy**, si quieres ordenar de

manera descendente, se puede usar la función **desc(nombre_column)** dentro del order by



The screenshot shows a terminal window titled "hadoop@nodo1-cluster:~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. Below the menu is a command-line interface where the user has run the following Scala code:

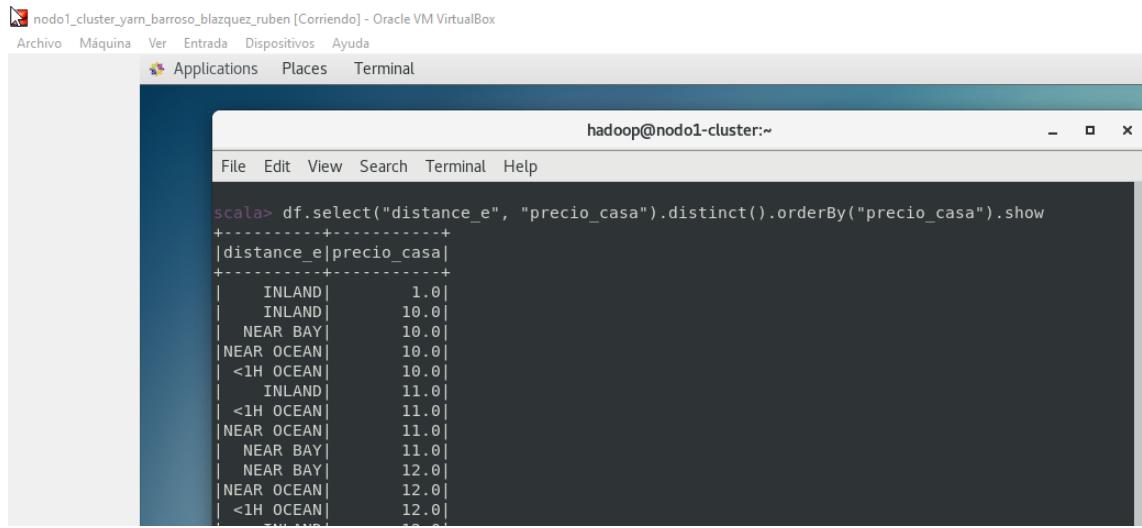
```
scala> df.select("distance_e", "precio_casa").orderBy("precio_casa").show
```

The output displays a table with two columns: "distance_e" and "precio_casa". The data shows multiple rows for each distance category, all with a price of 10.0.

distance_e	precio_casa
INLAND	1.0
INLAND	1.0
INLAND	1.0
INLAND	10.0
NEAR BAY	10.0
INLAND	10.0
NEAR OCEAN	10.0
INLAND	10.0
INLAND	10.0
NEAR BAY	10.0
<1H OCEAN	10.0
<1H OCEAN	10.0

Eliminación de duplicados

Para eliminar las columnas repetidas podemos usar el comando **distinct()**



The screenshot shows a terminal window titled "hadoop@nodo1-cluster:~". The window has a menu bar with File, Edit, View, Search, Terminal, and Help. Below the menu is a command-line interface where the user has run the following Scala code:

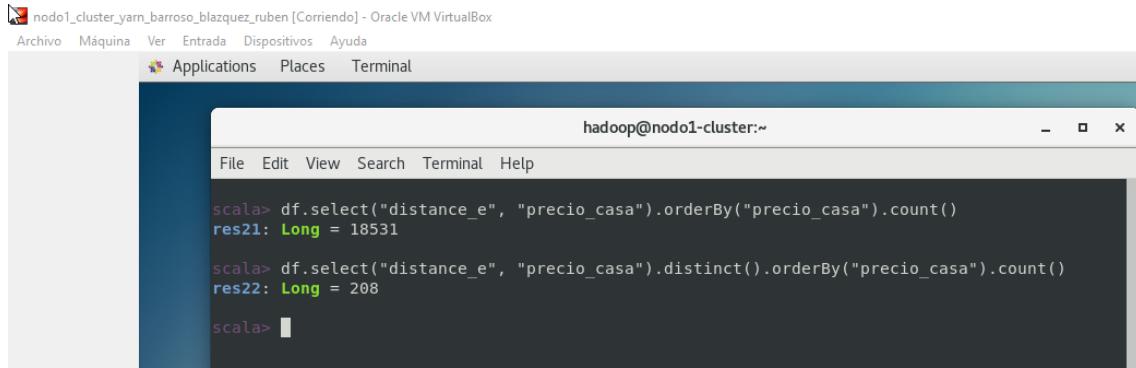
```
scala> df.select("distance_e", "precio_casa").distinct().orderBy("precio_casa").show
```

The output displays a table with two columns: "distance_e" and "precio_casa". The data shows unique rows for each distance category, with prices ranging from 1.0 to 12.0.

distance_e	precio_casa
INLAND	1.0
INLAND	10.0
NEAR BAY	10.0
NEAR OCEAN	10.0
<1H OCEAN	10.0
INLAND	11.0
<1H OCEAN	11.0
NEAR OCEAN	11.0
NEAR BAY	11.0
NEAR BAY	12.0
NEAR OCEAN	12.0
<1H OCEAN	12.0
INLAND	12.0

Contar los elementos obtenidos

Para contar los elementos usamos la función **count()**



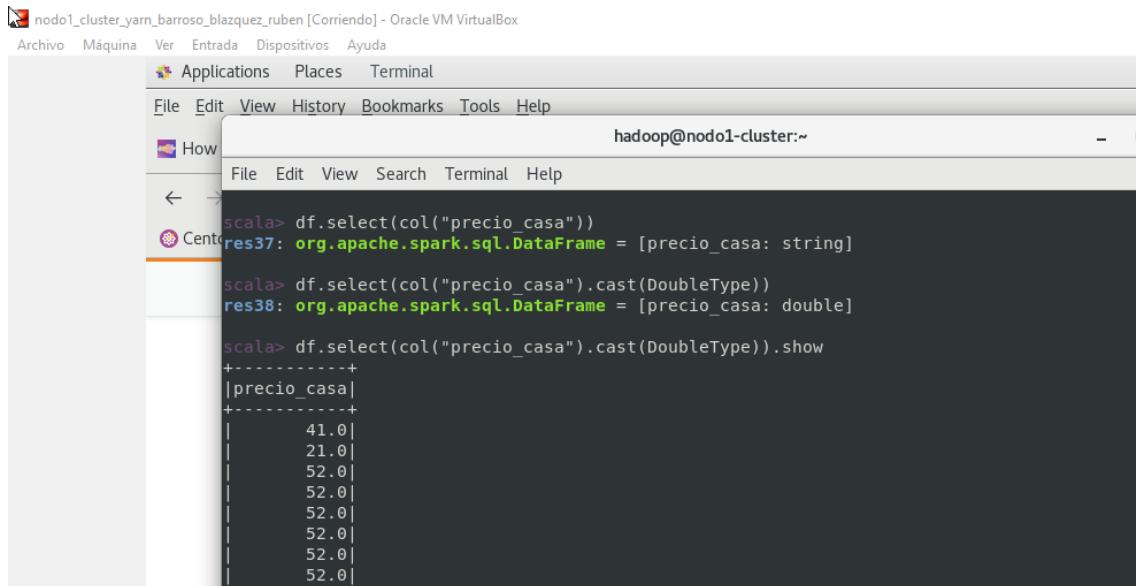
```
hadoop@nodo1-cluster:~$ df.select("distance_e", "precio_casa").orderBy("precio_casa").count()
res21: Long = 18531

hadoop@nodo1-cluster:~$ df.select("distance_e", "precio_casa").distinct().orderBy("precio_casa").count()
res22: Long = 208
```

Aquí podemos ver que distintos elementos solo hay 208

Conversión de datos

Se pueden convertir columnas de un DataFrame de un tipo de datos a otro utilizando los métodos **cast** y **to**. Por ejemplo, para convertir una columna llamada "precio_casa" de tipo String a tipo Double en un DataFrame llamado "df", puede hacer lo siguiente:



```
hadoop@nodo1-cluster:~$ df.select(col("precio_casa"))
res37: org.apache.spark.sql.DataFrame = [precio_casa: string]

hadoop@nodo1-cluster:~$ df.select(col("precio_casa").cast(DoubleType))
res38: org.apache.spark.sql.DataFrame = [precio_casa: double]

hadoop@nodo1-cluster:~$ df.select(col("precio_casa").cast(DoubleType)).show
+-----+
|precio_casa|
+-----+
|      41.0|
|      21.0|
|      52.0|
|      52.0|
|      52.0|
|      52.0|
|      52.0|
|      52.0|
```

Aquí se puede ver como al principio el tipo de dato es string, y luego se hace un casteo a double

Operaciones sobre cadenas

Se pueden realizar operaciones de cadena en columnas de un **DataFrame** utilizando los métodos **split**, **substring**, **concat** y otros. Por ejemplo, para dividir una columna llamada

"**nombre_completo**" en dos columnas separadas de "**nombre**" y "**apellido**" en un **DataFrame** llamado "**personas**", en nuestro caso tenemos una columna que es distancia del mar, así que cogeremos los 2 primeros letras con substring:

```

nodo1_cluster_yarn_barroso_bla... [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
File Edit View Bookmarks Tools Help
hadoop@nodo1-cluster:~ %
scala> df.withColumn("distance_e_reduced", substring($"distance_e", 0, 3))
res40: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 9 more fields]

scala> df.withColumn("distance_e_reduced", substring($"distance_e", 0, 3)).show
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|latitud|longitud|precio_casa|media_casa|años_casa|precio_medio|distancia_c|distancia_d|distancia_mar|distance_e|distance_e_reduced|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|-122.23| 37.88| 41.0| 880.0| 129.0| 322.0| 126.0| 8.3252| 452600.0| NEAR BAY| NEA|
|-122.22| 37.86| 21.0| 7099.0| 1106.0| 2401.0| 1138.0| 8.3014| 358500.0| NEAR BAY| NEA|
|-122.24| 37.85| 52.0| 1467.0| 190.0| 496.0| 177.0| 7.2574| 352100.0| NEAR BAY| NEA|
|-122.25| 37.85| 52.0| 1274.0| 235.0| 558.0| 219.0| 5.6431| 341300.0| NEAR BAY| NEA|
|-122.25| 37.85| 52.0| 1627.0| 280.0| 565.0| 259.0| 3.8462| 342200.0| NEAR BAY| NEA|
|-122.25| 37.85| 52.0| 919.0| 213.0| 413.0| 193.0| 4.0368| 269700.0| NEAR BAY| NEA|
|-122.25| 37.84| 52.0| 2535.0| 489.0| 1094.0| 514.0| 3.6591| 299200.0| NEAR BAY| NEA|
|-122.25| 37.84| 52.0| 3104.0| 687.0| 1157.0| 647.0| 3.12| 241400.0| NEAR BAY| NEA|
|-122.26| 37.84| 42.0| 2555.0| 665.0| 1206.0| 595.0| 2.0804| 226700.0| NEAR BAY| NEA|
|-122.25| 37.84| 52.0| 3549.0| 707.0| 1551.0| 714.0| 3.6912| 261100.0| NEAR BAY| NEA|
|-122.26| 37.85| 52.0| 2202.0| 434.0| 910.0| 402.0| 3.2031| 281500.0| NEAR BAY| NEA|
|-122.26| 37.85| 52.0| 2491.0| 474.0| 1098.0| 468.0| 3.075| 213500.0| NEAR BAY| NEA|
|-122.26| 37.84| 52.0| 696.0| 191.0| 345.0| 174.0| 2.6736| 191300.0| NEAR BAY| NEA|
|-122.26| 37.85| 52.0| 2643.0| 626.0| 1212.0| 620.0| 1.9167| 159200.0| NEAR BAY| NEAR|
|-122.26| 37.85| 50.0| 1120.0| 283.0| 697.0| 264.0| 2.125| 140000.0| NEAR BAY| NEAR|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

```

nodo1_cluster_yarn_barroso_bla... [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
File Edit View Bookmarks Tools Help
hadoop@nodo1-cluster:~ %
scala> df.withColumn("distance_e_split", split($"distance_e", " ")(0))
res51: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 9 more fields]

scala> df.withColumn("distance_e_split", split($"distance_e", " ")(0)).show
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|latitud|longitud|precio_casa|media_casa|años_casa|precio_medio|distancia_c|distancia_d|distancia_mar|distance_e|distance_e_split|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|-122.23| 37.88| 41.0| 880.0| 129.0| 322.0| 126.0| 8.3252| 452600.0| NEAR BAY| NEAR|
|-122.22| 37.86| 21.0| 7099.0| 1106.0| 2401.0| 1138.0| 8.3014| 358500.0| NEAR BAY| NEAR|
|-122.24| 37.85| 52.0| 1467.0| 190.0| 496.0| 177.0| 7.2574| 352100.0| NEAR BAY| NEAR|
|-122.25| 37.85| 52.0| 1274.0| 235.0| 558.0| 219.0| 5.6431| 341300.0| NEAR BAY| NEAR|
|-122.25| 37.85| 52.0| 1627.0| 280.0| 565.0| 259.0| 3.8462| 342200.0| NEAR BAY| NEAR|
|-122.25| 37.85| 52.0| 919.0| 213.0| 413.0| 193.0| 4.0368| 269700.0| NEAR BAY| NEAR|
|-122.25| 37.84| 52.0| 2535.0| 489.0| 1094.0| 514.0| 3.6591| 299200.0| NEAR BAY| NEAR|
|-122.25| 37.84| 52.0| 3104.0| 687.0| 1157.0| 647.0| 3.12| 241400.0| NEAR BAY| NEAR|
|-122.26| 37.84| 42.0| 2555.0| 665.0| 1206.0| 595.0| 2.0804| 226700.0| NEAR BAY| NEAR|
|-122.25| 37.84| 52.0| 3549.0| 707.0| 1551.0| 714.0| 3.6912| 261100.0| NEAR BAY| NEAR|
|-122.26| 37.85| 52.0| 2202.0| 434.0| 910.0| 402.0| 3.2031| 281500.0| NEAR BAY| NEAR|
|-122.26| 37.85| 52.0| 2491.0| 474.0| 1098.0| 468.0| 3.075| 213500.0| NEAR BAY| NEAR|
|-122.26| 37.84| 52.0| 696.0| 191.0| 345.0| 174.0| 2.6736| 191300.0| NEAR BAY| NEAR|
|-122.26| 37.85| 52.0| 2643.0| 626.0| 1212.0| 620.0| 1.9167| 159200.0| NEAR BAY| NEAR|
|-122.26| 37.85| 50.0| 1120.0| 283.0| 697.0| 264.0| 2.125| 140000.0| NEAR BAY| NEAR|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Actividad 5. (1,20 puntos)

5.- Consultas Complejas y Tratamiento/Limpieza de datos con Spark SQL

Uniones entre dataframes

```

nodo1_cluster_yarn_barroso_bla... [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
File Edit View Bookmarks Tools Help
hadoop@nodo1-cluster:~ %
scala> df.count()
res55: Long = 18531

scala> df_with_schema.count()
res56: Long = 18531

scala> val resultUnion = df.union(df_with_schema)
resultUnion: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [latitud: string, longitud: string ... 8 more fields]

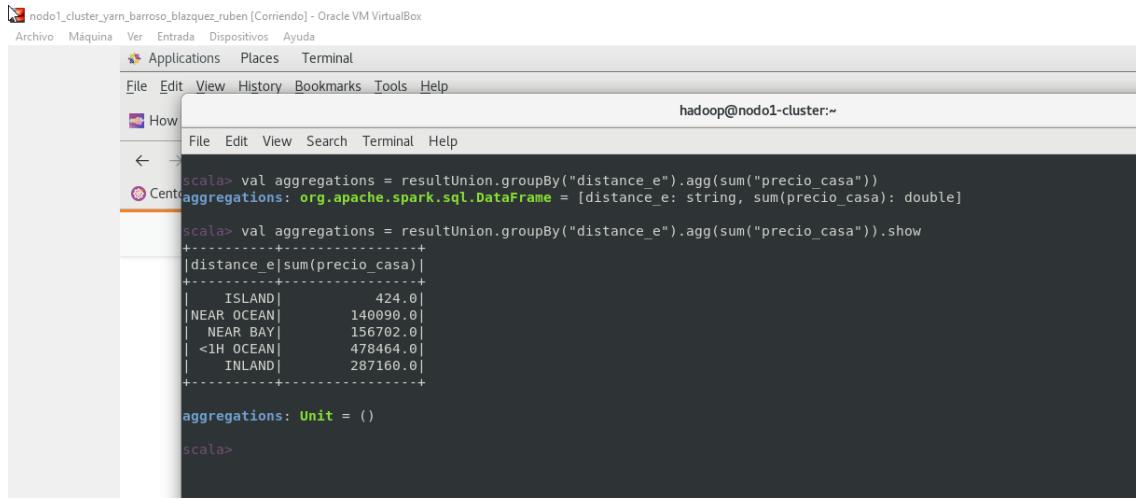
scala> resultUnion.count()
res57: Long = 37062

scala> resultUnion.show
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|latitud|longitud|precio_casa|media_casa|años_casa|precio_medio|distancia_c|distancia_d|distancia_mar|distance_e|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|-122.23| 37.88| 41.0| 880.0| 129.0| 322.0| 126.0| 8.3252| 452600.0| NEAR BAY|
|-122.22| 37.86| 21.0| 7099.0| 1106.0| 2401.0| 1138.0| 8.3014| 358500.0| NEAR BAY|
|-122.24| 37.85| 52.0| 1467.0| 190.0| 496.0| 177.0| 7.2574| 352100.0| NEAR BAY|
|-122.25| 37.85| 52.0| 1274.0| 235.0| 558.0| 219.0| 5.6431| 341300.0| NEAR BAY|
|-122.25| 37.85| 52.0| 1627.0| 280.0| 565.0| 259.0| 3.8462| 342200.0| NEAR BAY|
|-122.25| 37.85| 52.0| 919.0| 213.0| 413.0| 193.0| 4.0368| 269700.0| NEAR BAY|
|-122.25| 37.84| 52.0| 2535.0| 489.0| 1094.0| 514.0| 3.6591| 299200.0| NEAR BAY|
|-122.25| 37.84| 52.0| 3104.0| 687.0| 1157.0| 647.0| 3.12| 241400.0| NEAR BAY|
|-122.26| 37.84| 42.0| 2555.0| 665.0| 1206.0| 595.0| 2.0804| 226700.0| NEAR BAY|
|-122.25| 37.84| 52.0| 3549.0| 707.0| 1551.0| 714.0| 3.6912| 261100.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2202.0| 434.0| 910.0| 402.0| 3.2031| 281500.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2491.0| 474.0| 1098.0| 468.0| 3.075| 213500.0| NEAR BAY|
|-122.26| 37.84| 52.0| 696.0| 191.0| 345.0| 174.0| 2.6736| 191300.0| NEAR BAY|
|-122.26| 37.85| 52.0| 2643.0| 626.0| 1212.0| 620.0| 1.9167| 159200.0| NEAR BAY|
|-122.26| 37.85| 50.0| 1120.0| 283.0| 697.0| 264.0| 2.125| 140000.0| NEAR BAY|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Agrupaciones y Agregaciones

`df.groupBy(columnsToGroup).agg(columnsWithAggregateFunction)`

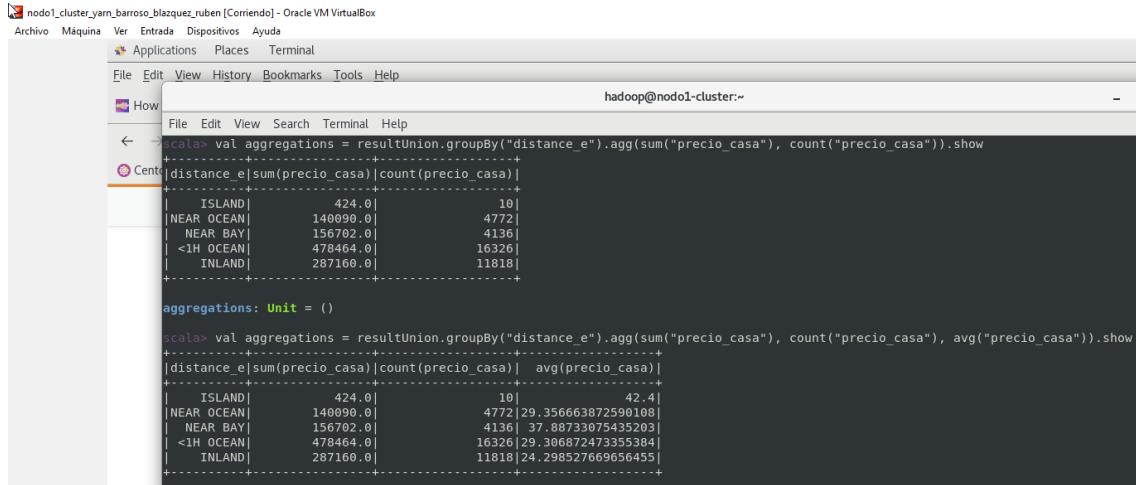


```
scala> val aggregations = resultUnion.groupBy("distance_e").agg(sum("precio_casa"))
aggregations: org.apache.spark.sql.DataFrame = [distance_e: string, sum(precio_casa): double]

scala> val aggregations = resultUnion.groupBy("distance_e").agg(sum("precio_casa")).show
+-----+-----+
|distance_e|sum(precio_casa)|
+-----+-----+
| ISLAND | 424.0 |
| NEAR OCEAN| 140090.0 |
| NEAR BAY| 156702.0 |
| <1H OCEAN| 478464.0 |
| INLAND | 287160.0 |
+-----+-----+

aggregations: Unit = ()
```

Podemos poner una agregación o varias , en este caso hemos puesto la suma, pero podríamos poner también la media, el conteo etc...



```
scala> val aggregations = resultUnion.groupBy("distance_e").agg(sum("precio_casa"), count("precio_casa")).show
+-----+-----+-----+
|distance_e|sum(precio_casa)|count(precio_casa)|
+-----+-----+-----+
| ISLAND | 424.0 | 10 |
| NEAR OCEAN| 140090.0 | 4772 |
| NEAR BAY| 156702.0 | 4136 |
| <1H OCEAN| 478464.0 | 16326 |
| INLAND | 287160.0 | 11818 |
+-----+-----+-----+

aggregations: Unit = ()

scala> val aggregations = resultUnion.groupBy("distance_e").agg(sum("precio_casa"), count("precio_casa"), avg("precio_casa")).show
+-----+-----+-----+-----+
|distance_e|sum(precio_casa)|count(precio_casa)| avg(precio_casa)|
+-----+-----+-----+-----+
| ISLAND | 424.0 | 19 | 42.4 |
| NEAR OCEAN| 140090.0 | 4772 | 29.356663872590108 |
| NEAR BAY| 156702.0 | 4136 | 37.88733075435203 |
| <1H OCEAN| 478464.0 | 16326 | 29.306872473355384 |
| INLAND | 287160.0 | 11818 | 24.298527669656455 |
+-----+-----+-----+-----+
```

Joins

Al igual que en **SQL**, con **spark SQL** puedes hacer también **joins**, en este caso vamos a unir 2 dataframes, uno de **esperanza de vida**

nodo1_cluster_yarn_barroso_blazquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Terminal

File Edit View Search Terminal Help

```
scala> lifesat.show
+-----+-----+
|   Country|GDP per capita (USD)|Life satisfaction|
+-----+-----+
| South Africa| 11466.189671582|        4.7|
| Colombia| 13441.4929522564|        6.3|
| Brazil| 14063.9825053838|        6.4|
| Mexico| 17887.7507356971|        6.5|
| Chile| 23324.5247506509|        6.5|
| Russia| 26456.3879381321|        5.8|
| Greece| 27287.0834009302|        5.4|
| Turkey| 28384.9877846263|        5.5|
| Latvia| 29932.4939100562|        5.9|
| Hungary| 31007.7684065437|        5.6|
| Portugal| 32181.1545372343|        5.4|
| Poland| 32238.157259275|        6.1|
| Estonia| 35638.4213511812|        5.7|
| Spain| 36215.4475907307|        6.3|
| Slovenia| 36547.7389559849|        5.9|
| Lithuania| 36732.034744031|        5.9|
| Israel| 38341.3075704083|        7.2|
| Italy| 38992.1483807498|        6.0|
| United Kingdom| 41627.129269425|        6.8|
```

Y un dataframe de gdp per capita

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:~>

File Edit View Search Terminal Help

scala> gdp_per_capita.show
+-----+-----+-----+
| Entity|Code|Year|GDP per capita, PPP (constant 2017 international $)|
+-----+-----+-----+
| Afghanistan| AFG|2002| 1189.78466765718|
| Afghanistan| AFG|2003| 1235.81006329565|
| Afghanistan| AFG|2004| 1200.27801321734|
| Afghanistan| AFG|2005| 1286.79365893927|
| Afghanistan| AFG|2006| 1315.78911741835|
| Afghanistan| AFG|2007| 1460.8257513794|
| Afghanistan| AFG|2008| 1484.11446132539|
| Afghanistan| AFG|2009| 1758.90447663765|
| Afghanistan| AFG|2010| 1957.02906990812|
| Afghanistan| AFG|2011| 1904.55992565594|
| Afghanistan| AFG|2012| 2075.49161435331|
| Afghanistan| AFG|2013| 2116.46525771251|
| Afghanistan| AFG|2014| 2102.38460375974|
| Afghanistan| AFG|2015| 2068.26590413364|
| Afghanistan| AFG|2016| 2057.06797755329|
| Afghanistan| AFG|2017| 2058.4002210698|
| Afghanistan| AFG|2018| 2033.80438893717|
| Afghanistan| AFG|2019| 2065.03623501599|
| Afghanistan| AFG|2020| 1978.96157872183|

```

Como se puede observar se relacionan por el nombre del país, asique podemos hacer un **join** de los 2 dataframes con el siguiente comando:

```
lifesat.join(gdp_per_capita, lifesat("Country") === gdp_per_capita("Entity"),
"inner")
```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:~>

File Edit View Search Terminal Help

scala> val df_join = lifesat.join(gdp_per_capita, lifesat("Country") === gdp_per_capita("Entity"), "inner")
df_join: org.apache.spark.sql.DataFrame = [Country: string, GDP per capita (USD): string ... 5 more fields]

scala> df_join.show
+-----+-----+-----+-----+-----+-----+
| Country|GDP per capita (USD)|Life satisfaction| Entity|Code|Year|GDP per capita, PPP (constant 2017 international $)|
+-----+-----+-----+-----+-----+-----+
|Australia| 48697.8370282475| 7.3|Australia| AUS|1990| 31058.2596709541|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1991| 30542.9197143586|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1992| 30298.9290163388|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1993| 31213.0798529168|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1994| 32114.830840244|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1995| 32946.9006734039|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1996| 33778.0635392223|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1997| 34727.1997827083|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1998| 35940.0171358725|
|Australia| 48697.8370282475| 7.3|Australia| AUS|1999| 37334.4752293048|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2000| 38343.002147534|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2001| 38559.9041923683|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2002| 39616.3360972665|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2003| 40298.8860528145|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2004| 41449.8419535385|
|Australia| 48697.8370282475| 7.3|Australia| AUS|2005| 42217.136349705|

```

Utilización de la función explode en un DataFrame: Puede utilizar la función **explode** para dividir una columna que sea una **lista** en varias filas en un **DataFrame**. Por ejemplo,

para dividir una columna de **matriz** llamada "**personas_interesadas**" y contar el número de **casa**s por las cuales una persona pueda estar interesada, puede hacer lo siguiente

```
val resultado = df.select(explode($"personas_interesadas")
  ).alias("personas")).groupBy("personas").count()
```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:~>

scala> val df_with_persons = df.withColumn("personas_interesadas", lit(Array("Ruben","Pablo","Diego","Juanma")))
df_with_persons: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 9 more fields]

scala> val resultado = df_with_persons.select(explode($"personas_interesadas").alias("personas")).groupBy("personas").count()
resultado: org.apache.spark.sql.DataFrame = [personas: string, count: bigint]

scala> resultado.show
+-----+
|personas|count|
+-----+
| Ruben|18531|
| Diego|18531|
| Pablo|18531|
| Juanma|18531|
+-----+

```

Tratamiento/Limpieza de datos

Con **spark SQL** se puede realizar una buena limpieza de los datos, para ello tenemos diferentes funciones

Eliminación de Valores NULOS, Puedes utilizar la función **na.fill** del DataFrame para reemplazar valores nulos en una columna. Por ejemplo, para reemplazar los valores nulos en la columna "precio_casa" con el valor 0

```
val dfClean = df.na.fill(Map("precio_casa" -> 0))
```

```

nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:~

File Edit View Search Terminal Help

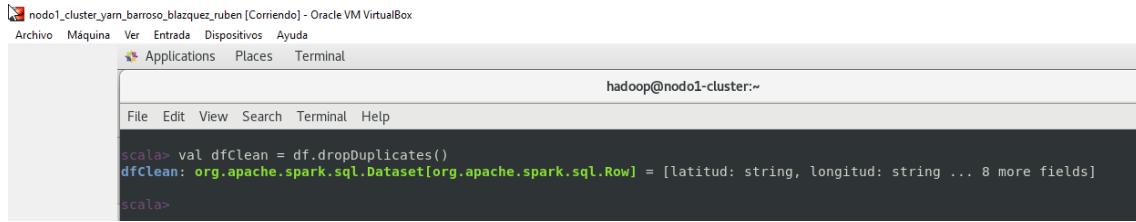
scala> val dfClean = df.na.fill(Map("precio_casa" -> 0))
dfClean: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 8 more fields]

scala> dfClean.show
+-----+
|latitud|longitud|precio_casa|media_casa|años_casa|precio_medio|distancia_c|distancia_d|distancia_m|distance_e|
+-----+
|-122.23| 37.88|    41.0|   880.0|   129.0|    322.0|   126.0|    8.3252| 452600.0| NEAR BAY|
|-122.22| 37.86|    21.0|  7099.0|  1106.0|   2401.0|  1138.0|    8.3014| 358500.0| NEAR BAY|
|-122.24| 37.85|    52.0|  1467.0|   190.0|    496.0|   177.0|    7.2574| 352100.0| NEAR BAY|
|-122.25| 37.85|    52.0|  1274.0|   235.0|    558.0|   219.0|    5.6431| 341300.0| NEAR BAY|
|-122.25| 37.85|    52.0|  1627.0|   280.0|    565.0|   259.0|    3.8462| 342200.0| NEAR BAY|
|-122.25| 37.85|    52.0|    919.0|   213.0|    413.0|   193.0|    4.0368| 269700.0| NEAR BAY|
|-122.25| 37.84|    52.0|  2535.0|   489.0|   1094.0|   514.0|    3.6591| 299200.0| NEAR BAY|
|-122.25| 37.84|    52.0|  3104.0|   687.0|   1157.0|   647.0|     3.12| 241400.0| NEAR BAY|
|-122.26| 37.84|    42.0|  2555.0|   665.0|   1206.0|   595.0|    2.0804| 226700.0| NEAR BAY|
|-122.25| 37.84|    52.0|  3549.0|   707.0|   1551.0|   714.0|    3.6912| 261100.0| NEAR BAY|
|-122.26| 37.85|    52.0|  2202.0|   434.0|    910.0|   402.0|    3.2031| 281500.0| NEAR BAY|
|-122.26| 37.85|    52.0|  3503.0|   752.0|   1504.0|   734.0|    3.2705| 241800.0| NEAR BAY|
|-122.26| 37.85|    52.0|  2491.0|   474.0|   1098.0|   468.0|    3.075| 213500.0| NEAR BAY|
|-122.26| 37.84|    52.0|   696.0|   191.0|    345.0|   174.0|    2.6736| 191300.0| NEAR BAY|
|-122.26| 37.85|    52.0|  2643.0|   626.0|   1212.0|   620.0|    1.9167| 159200.0| NEAR BAY|
|-122.26| 37.85|    50.0|  1120.0|   283.0|    697.0|   264.0|     2.125| 140000.0| NEAR BAY|

```

Eliminación de Valores Duplicados:

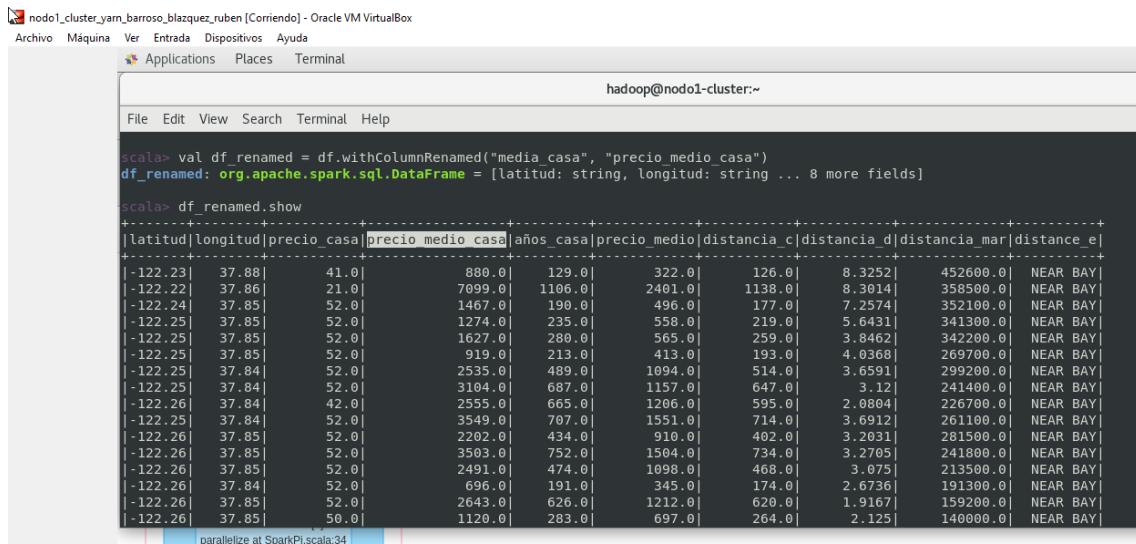
```
val dfClean = df.dropDuplicates()
```



The screenshot shows a terminal window titled "nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The window has tabs for "Archivo", "Máquina", "Ver", "Entrada", "Dispositivos", and "Ayuda". Below the tabs are "Applications", "Places", and "Terminal". The terminal prompt is "hadoop@nodo1-cluster:~". The Scala code `val dfClean = df.dropDuplicates()` is entered, followed by the output: `dfClean: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [latitud: string, longitud: string ... 8 more fields]`.

Renombrado de Columnas:

```
val df_renamed = df.withColumnRenamed("media_casa", "precio_medio_casa")
```



The screenshot shows a terminal window titled "nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox". The window has tabs for "Archivo", "Máquina", "Ver", "Entrada", "Dispositivos", and "Ayuda". Below the tabs are "Applications", "Places", and "Terminal". The terminal prompt is "hadoop@nodo1-cluster:~". The Scala code `val df_renamed = df.withColumnRenamed("media_casa", "precio medio casa")` is entered, followed by the output: `df_renamed: org.apache.spark.sql.DataFrame = [latitud: string, longitud: string ... 8 more fields]`. Then, `df_renamed.show` is run, displaying the DataFrame schema and some sample data. The schema includes columns: latitud, longitud, precio_casa, precio_medio_casa, años_casa, precio_medio, distancia_c, distancia_d, distancia_mar, and distance_e. The data shows various coordinates and values.

Eliminación de Columnas

```
val df_with_drop = df.drop("distancia_c", "distancia_d")
```

Actividad 6. (1,20 puntos)

6.- Utilización de Spark Streaming

¿Qué es Spark Streaming?

Spark Streaming es un módulo de procesamiento de datos en tiempo real en Spark que permite procesar y analizar datos en tiempo real.

El módulo utiliza la capacidad de procesamiento de lotes de Spark para procesar datos de streaming en pequeños lotes (batch) con una latencia muy baja, lo que lo hace ideal para aplicaciones de análisis de datos en tiempo real. Permite la integración con una amplia variedad de fuentes de datos en streaming, incluyendo Kafka, Flume, Twitter y muchas más. Los datos se reciben en tiempo real y se procesan en pequeños lotes, lo que permite el análisis de datos en tiempo real y la generación de informes en tiempo real.

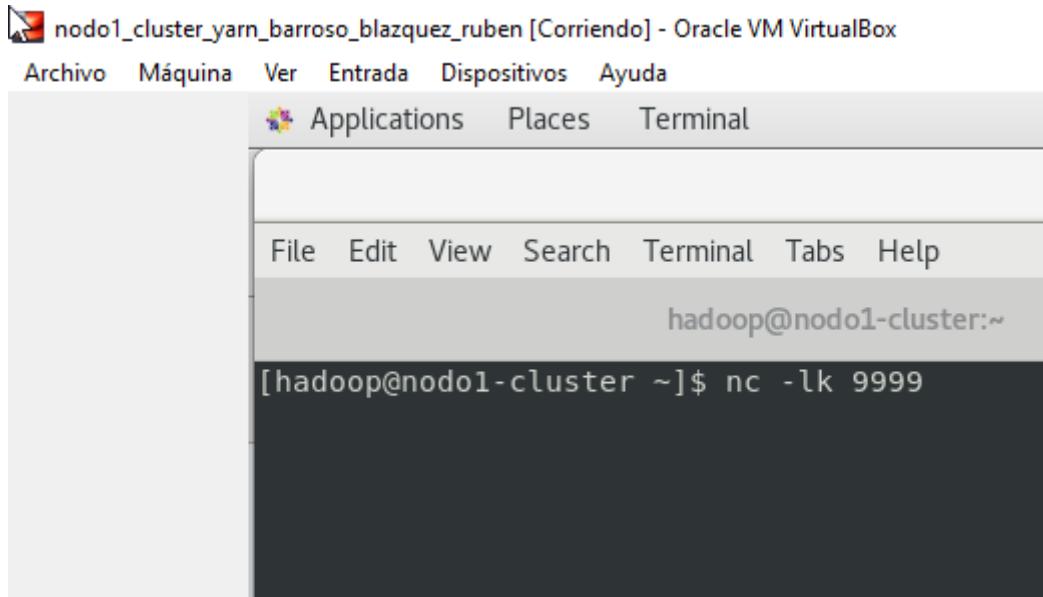
Para probar como funciona spark streaming lo primero que tenemos que hacer es importar estos 2 paquetes:

Import org.apache.spark.sql.Row

Import spark.implicits._

Y ahora lo que haremos será poner a escuchar en un socket a spark, para ello lo primero que haremos será crear un servidor con **netcat**, al cual spark estará escuchando, para ello ejecutamos en otra terminal el siguiente comando,

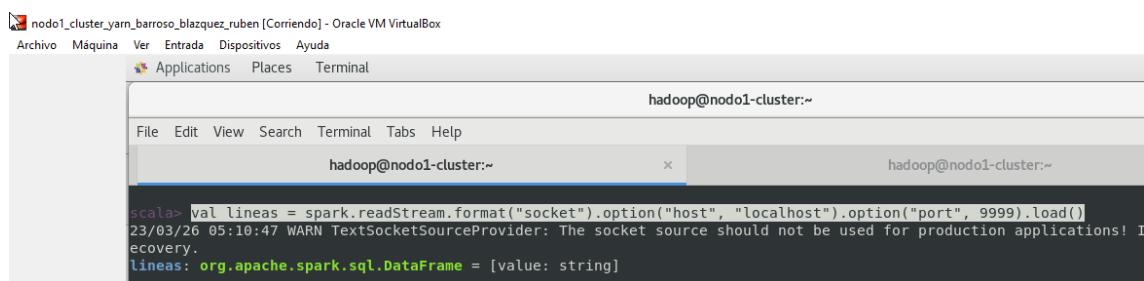
```
nc -lk 9999
```



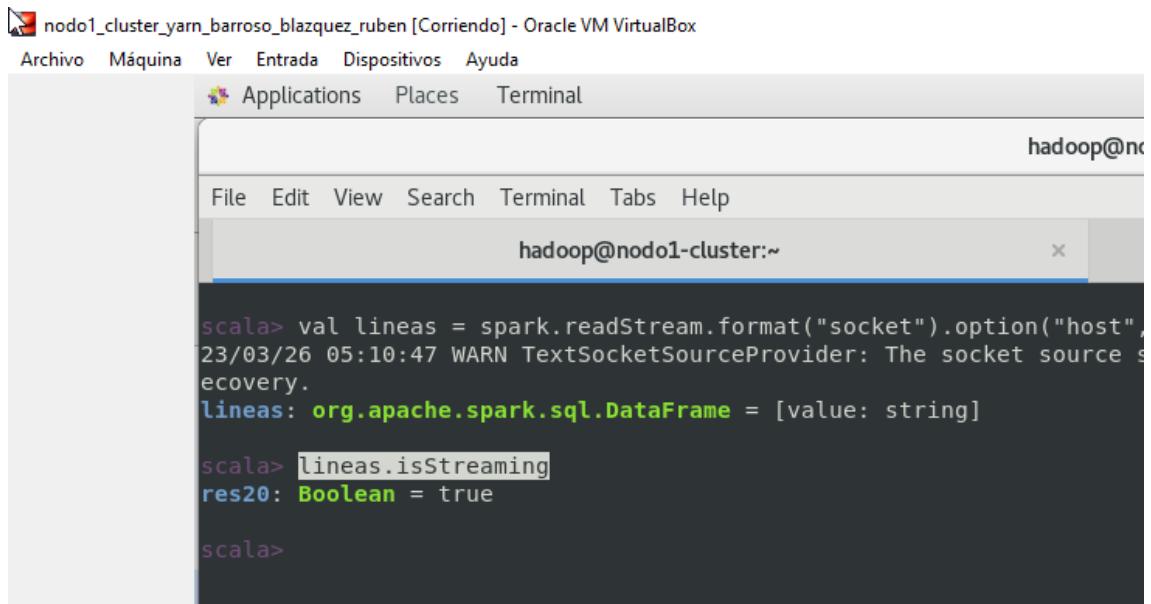
y dejamos el proceso corriendo, ahora cada palabra que escribamos en esa terminal, la recibirá spark.

Ahora, para que spark esté escuchando lo que tenemos que hacer es definir un socket, para ello crearemos la siguiente variable

```
val lineas = spark.readStream().format("socket").option("host", "localhost").option("port", 9999).load()
```



¿Cómo comprobamos si realmente está escuchando? Podemos verlo con el parámetro **isStreaming**, del socket que acabamos de crear



```
scala> val lineas = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).option("recovery", "memory").start()
23/03/26 05:10:47 WARN TextSocketSourceProvider: The socket source should not be used with memory recovery.
lineas: org.apache.spark.sql.DataFrame = [value: string]

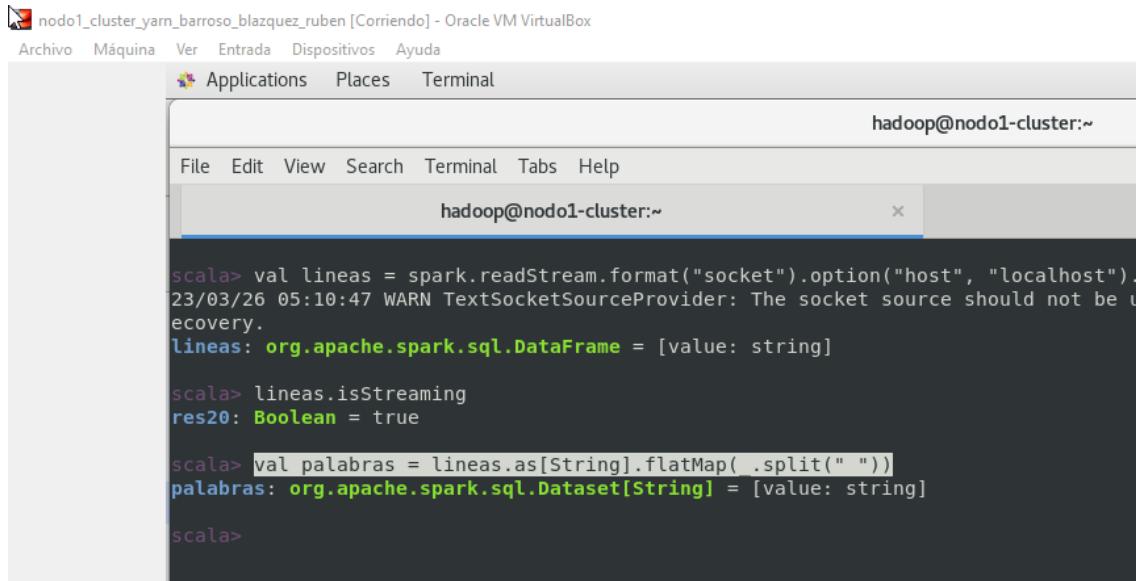
scala> lineas.isStreaming
res20: Boolean = true

scala>
```

Se puede observar que sí está escuchando

La siguiente pregunta es, **¿Cómo obtenemos las palabras escritas que llegan al streaming?** Bueno, pues lo primero que tenemos que hacer es obtenerlas de el socket líneas creado,y mapearlas para hacerlas un Split por espacios , ya que el socket nos devolverá un string entero por ello usaremos la siguiente consulta:

Val palabras = líneas.as[String].flatMap(_.split(" "))



```
scala> val lineas = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).option("recovery", "memory").start()
23/03/26 05:10:47 WARN TextSocketSourceProvider: The socket source should not be used with memory recovery.
lineas: org.apache.spark.sql.DataFrame = [value: string]

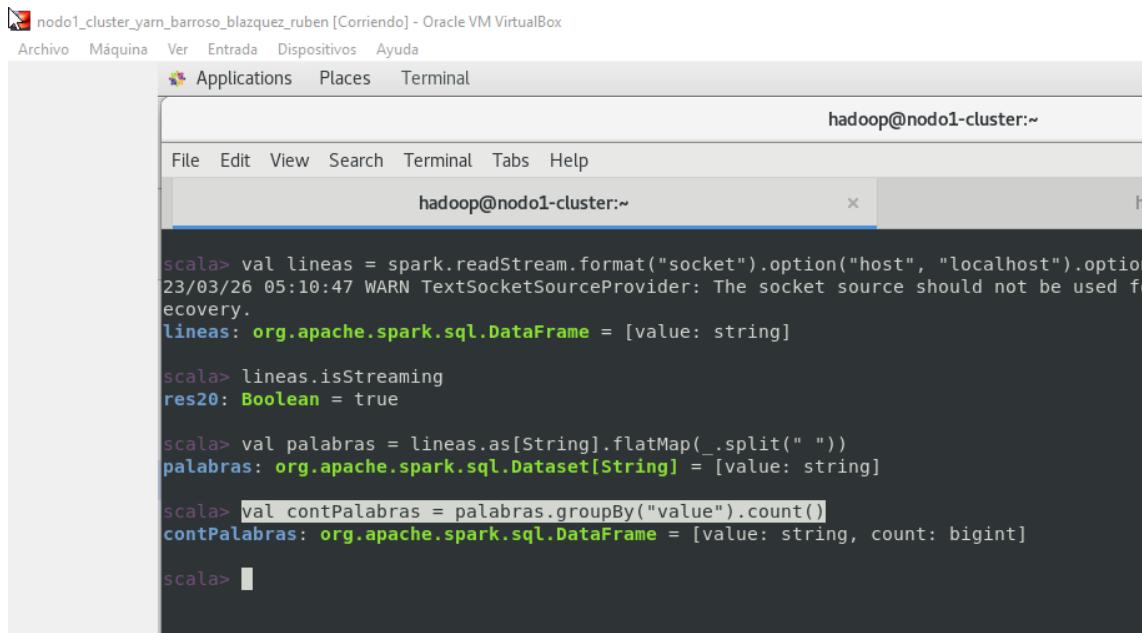
scala> lineas.isStreaming
res20: Boolean = true

scala> val palabras = lineas.as[String].flatMap(_.split(" "))
palabras: org.apache.spark.sql.Dataset[String] = [value: string]

scala>
```

Ahora contaremos las palabras separadas agrupándolas por el value:

Val contPalabras = palabras.groupBy("value").count()



```
nodo1_cluster_yarn_barroso_blaquez_ruben [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Applications Places Terminal
hadoop@nodo1-cluster:~
```

```
File Edit View Search Terminal Tabs Help
hadoop@nodo1-cluster:~
```

```
scala> val lineas = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).load()
23/03/26 05:10:47 WARN TextSocketSourceProvider: The socket source should not be used for recovery.
lineas: org.apache.spark.sql.DataFrame = [value: string]

scala> lineas.isStreaming
res20: Boolean = true

scala> val palabras = lineas.as[String].flatMap(_.split(" "))
palabras: org.apache.spark.sql.Dataset[String] = [value: string]

scala> val contPalabras = palabras.groupBy("value").count()
contPalabras: org.apache.spark.sql.DataFrame = [value: string, count: bigint]

scala> 
```

Y por último escribiremos en la consola de scala el resultado de estas consultas en la consola mediante un **Stream**, que lo que hará será escribir en la consola cada vez que detecte cambios spark o reciba nuevos valores. Esto se puede hacer con la siguiente consulta;

```
Val query =
contPalabras.writeStream.format("console").outputMode("update").start()
```

Y le decimos que espere 60000 milisegundos para que termine de escuchar para escribir información con el comando:

```
Query.awaitTermination(60000)
```

Ahora escribimos en nuestro netcat

```
[hadoop@nodo1-cluster ~]$ nc -lk 9999

buenas
esto
es
una
prueba
para
la
practica
spark
```

Y vamos viendo como por grupos va spark recibiendo la información

```
val contPalabras = palabras.groupBy("value").count()
contPalabras: org.apache.spark.sql.DataFrame = [value: string, count: bigint]

val query = contPalabras.writeStream.format("console").outputMode("update").start()
23/03/26 05:23:03 WARN ResolveWriteToStream: Temporary checkpoint location created which is del
/tmp/temporary-e7c47b87-4e4a-463b-alba-12ba60f0d3bb. If it's required to delete it under any c
ing.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folde
23/03/26 05:23:03 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in str
sabled.

query: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution.streaming

Batch: 0
-----
+----+---+
|value|count|
+----+---+
-----
```

```
Batch: 1
-----
```

nodo1_cluster_yarn_barroso_blazquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Terminal

hadoop@nodo1-cluster:~

```
File Edit View Search Terminal Tabs Help
```

hadoop@nodo1-cluster:~

```
Batch: 2
-----
+-----+
| value|count|
+-----+
|buenas|    1|
+-----+-----+
```

```
Batch: 3
-----
+-----+
| value|count|
+-----+
|  una|    1|
|prueba|    1|
|  para|    1|
|    es|    1|
|  esto|    1|
+-----+-----+
```

nodo1_cluster_yarn_barroso_blazquez_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Applications Places Terminal

hadoop@nodo1-cluster:~

```
File Edit View Search Terminal Tabs Help
```

hadoop@nodo1-cluster:~

```
+-----+
| value|count|
+-----+
|  una|    1|
|prueba|    1|
|  para|    1|
|    es|    1|
|  esto|    1|
+-----+-----+
```

```
Batch: 4
-----
+-----+
|    value|count|
+-----+
|  spark|    1|
|practica|    1|
|    la|    1|
+-----+-----+
```

Y Si volvemos a escribir spark en el netcat...

```
spark  
spark spark spark spark
```

Veremos como está recibiendo la información

```
+-----+  
[Stage 31:=====]> (106 + 2) / 200  
Etiquetas para Rubén ▾
```

Y cuando la recibe veremos como ahora ha contado que spark ha salido 5 veces

```
nodo1_cluster_yarn_barroso_blazquez_ruben [Corriendo] - Oracle VM VirtualBox  
Archivo Máquina Ver Entrada Dispositivos Ayuda  
Applications Places Terminal  
File Edit View Search Terminal Tabs Help  
hadoop@nodo1-cluster:~  
+-----+-----+  
--  
Batch: 4  
+-----+-----+  
| value|count|  
+-----+-----+  
| spark|    1|  
|practica|    1|  
|     la|    1|  
+-----+-----+  
--  
Batch: 5  
+-----+-----+  
|value|count|  
+-----+-----+  
|spark|    5|  
+-----+-----+  
Etiquetas para Rubén ▾
```

Cabe destacar que cuando abrimos un socket con spark streaming, este crea una aplicación en la interfaz gráfica de spark, y podemos ver el grafo de ejecución, los Jobs completados, la información recibida etc...

Spark Jobs (2)

User: hadoop
Total Uptime:
Scheduling Mode: FIFO
Completed Jobs: 26

Event Timeline
Completed Jobs (26)

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded
25 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 6 start at <console>:32	2023/03/26 05:27:04	9 s	2/2	202/202
24 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:26:19	9 s	2/2	202/202
23 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:23:48	13 s	2/2	202/202
22 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:23:40	8 s	2/2	202/202
21 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:23:31	9 s	2/2	202/202
20 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:23:22	9 s	2/2	202/202
19 (61572830-2b2c-465e-8869-deede4cefa63)	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5 start at <console>:32	2023/03/26 05:23:04	17 s	2/2	202/202
18	show at <console>:27 show at <console>:27	2023/03/25 18:02:50	0.7 s	1/1 (1 skipped)	1/1 (1 skipped)

Además en la parte de Sql / Dataframe puedes observar la información que hemos recibido en la consola

SQL / DataFrame

Completed Queries: 37

Completed Queries (37)

ID ▾	Description	Submitted	Duration	Job ID
36	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 6	2023/03/26 05:27:13	4 ms	
35	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 6	2023/03/26 05:27:04	9 s	[25]
34	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 6	2023/03/26 05:27:04	9 s	
33	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5	2023/03/26 05:26:28	4 ms	
32	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5	2023/03/26 05:26:19	9 s	[24]
31	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 5	2023/03/26 05:26:19	9 s	
30	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 4	2023/03/26 05:24:01	26 ms	
29	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 4	2023/03/26 05:23:48	13 s	[23]
28	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 4	2023/03/26 05:23:48	13 s	
27	id = 6148424-bce0-44e6-8c21-ea44c28b5a41 runid = 61572830-2b2c-465e-8869-deede4cefa63 batch = 3	2023/03/26 05:23:48	4 ms	

Y clickando en una descripción..

Details for Query 30

Submitted Time: 2023/03/26 05:24:01
Duration: 26 ms

Show the Stage ID and Task ID that corresponds to the max metric

LocalTableScan

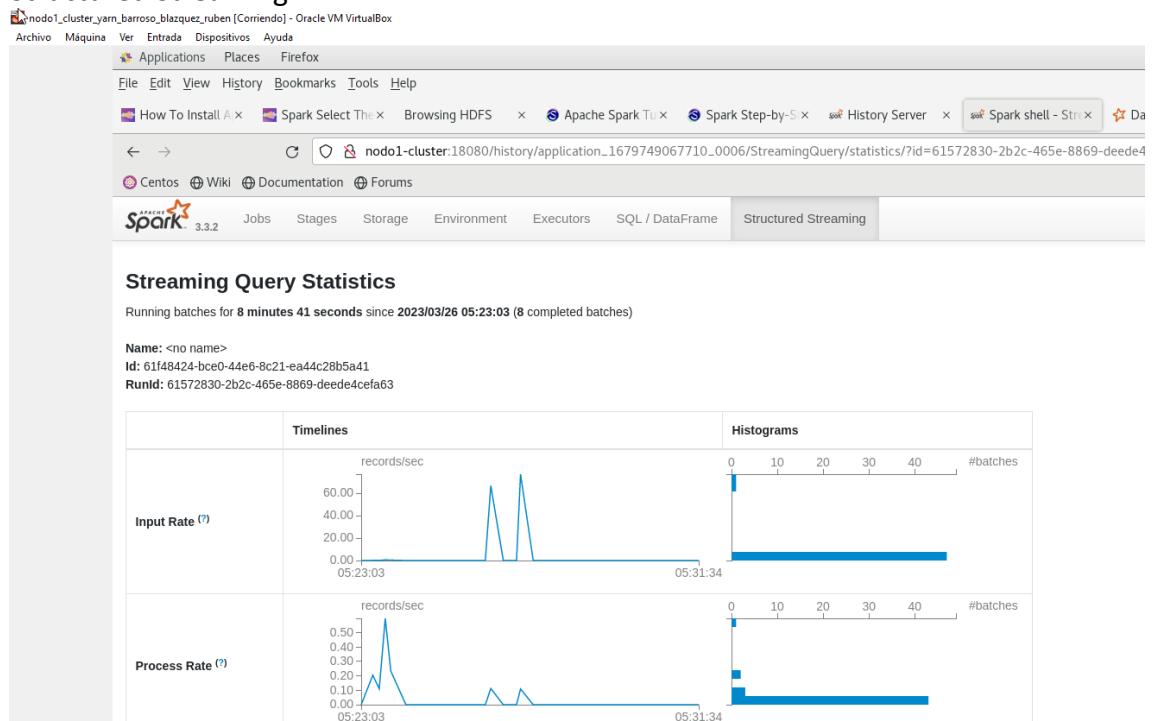
▼ Details

```
== Physical Plan ==
LocalTableScan (1)

(1) LocalTableScan
Output [2]: [value#832, count#833]
Arguments: [value#832, count#833]
```

► SQL / DataFrame Properties

También puedes ver el número de ingestión de datos por tiempo en la pestaña de Structured Streaming



Actividad 7. (1,20 puntos)

7.- Spark con Python (PySpark)

¿Qué es Pyspark?

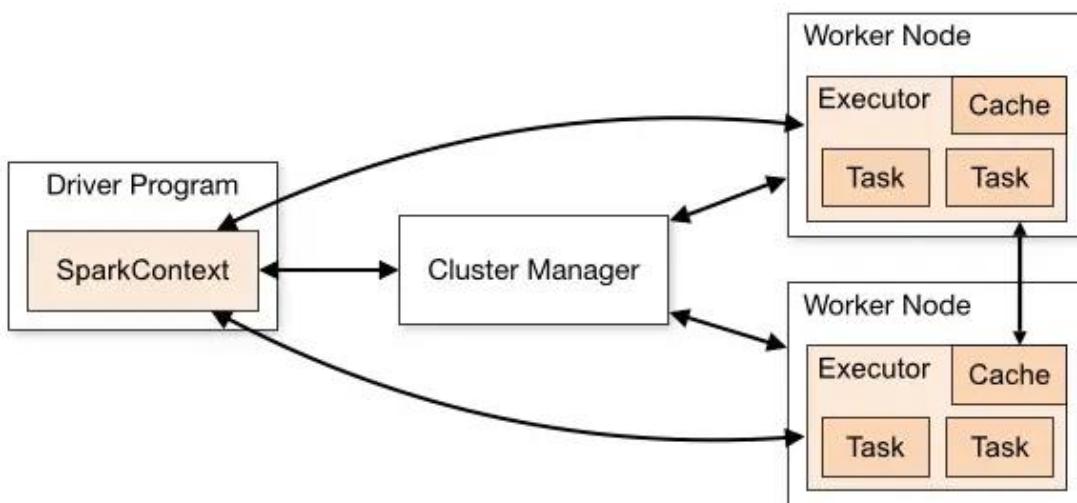
Spark es el nombre del motor para realizar la computación en clúster, mientras que PySpark es la biblioteca de Python para usar Spark.

Ventajas de PySpark

1. PySpark es un motor de procesamiento distribuido, en memoria y de uso general que le permite procesar datos de manera eficiente y distribuida.
2. Las aplicaciones que se ejecutan en PySpark son 100 veces más rápidas que los sistemas tradicionales.
3. Obtendrá grandes beneficios al usar PySpark para canalizaciones de ingesta de datos.
4. Con PySpark podemos procesar datos de Hadoop HDFS, AWS S3 y muchos sistemas de archivos.
5. PySpark también se usa para procesar datos en tiempo real usando Streaming y Kafka.
6. Con la transmisión de PySpark, también puede transmitir archivos desde el sistema de archivos y también desde el socket.
7. PySpark tiene bibliotecas gráficas y de aprendizaje automático de forma nativa.

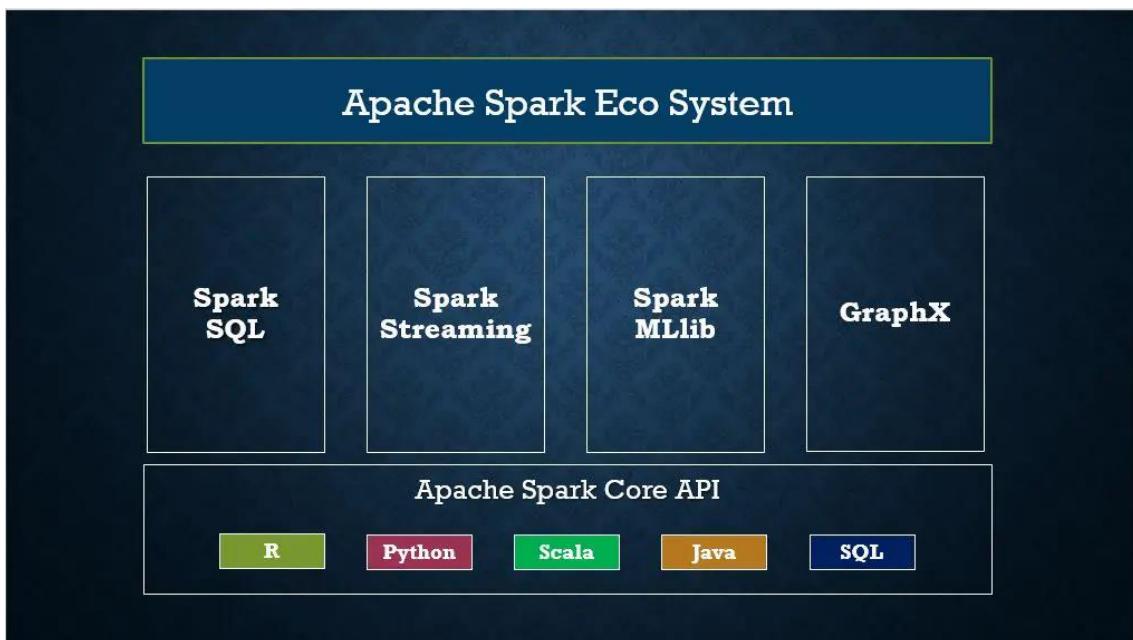
Arquitectura PySpark

Apache Spark funciona en una arquitectura maestro-esclavo donde el maestro se llama "Conductor" y los esclavos se llaman "Trabajadores". Cuando ejecuta una aplicación Spark, Spark Driver crea un contexto que es un punto de entrada a su aplicación, y todas las operaciones (transformaciones y acciones) se ejecutan en los nodos trabajadores y los recursos son administrados por el Administrador de clústeres



Módulos PySpark

- RDD (`pyspark.RDD`)
- PySpark DataFrame y SQL (`pyspark.sql`)
- PySpark Streaming (`pyspark.streaming`)
- PySpark MLlib (`pyspark.ml` , `pyspark.mllib`)
- PySpark GraphFrames (`GraphFrames`)
- PySpark Resource (`pyspark.resource`) Es nuevo en PySpark 3.0



Configuración y Ejemplo básico con PySpark

Para iniciar **Spark** dentro de **Python** lo primero que tenemos que hacer es crear una sesión de **Spark-Shell** mediante Python, para ello creamos un objeto **SparkSession**.

Una vez creada la sesión, deberemos acceder al objeto de **SparkContext** que nos proporciona nuestro objeto de **SparkSession** y es el motor interno que permite las conexiones con los clústeres y la creación de objetos para la paralelización. Si desea ejecutar una operación, necesita un **SparkContext**.

Para ello creamos tendríamos que hacer lo siguiente:

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
PySpark > main.py
Project main.py
1 from pyspark.sql import SparkSession
2
3 if __name__ == '__main__':
4     spark = SparkSession.builder.master("local[*]").appName("BDDPractice").getOrCreate()
5
6     sc = spark.sparkContext
```

Creación de RDD

Ahora que **SparkContext** está listo, puede crear una colección de datos denominada **RDD, Resilient Distributed Dataset**. El cálculo en un **RDD** se paraleliza automáticamente en todo el clúster.

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
PySpark > main.py
Project main.py
1 from pyspark.sql import SparkSession
2
3 if __name__ == '__main__':
4     spark = SparkSession.builder.master("local[*]").appName("BDDPractice").getOrCreate()
5
6     sc = spark.sparkContext
7
8     nums = [1, 2, 3, 4]
9     nums = sc.parallelize(nums)
10
11     print('First Number: ', nums.take(1))
12
13 if __name__ == '__main__':
14     Python Console <...>
15     First Number: [1]
```

¿Qué es un RDD?, Es una estructura de datos fundamental en Apache Spark, que permite el procesamiento de datos distribuidos y paralelos en un clúster. En términos simples, es una colección de datos que se puede dividir en particiones y distribuir a través de múltiples nodos en un clúster para procesarlos de manera eficiente.

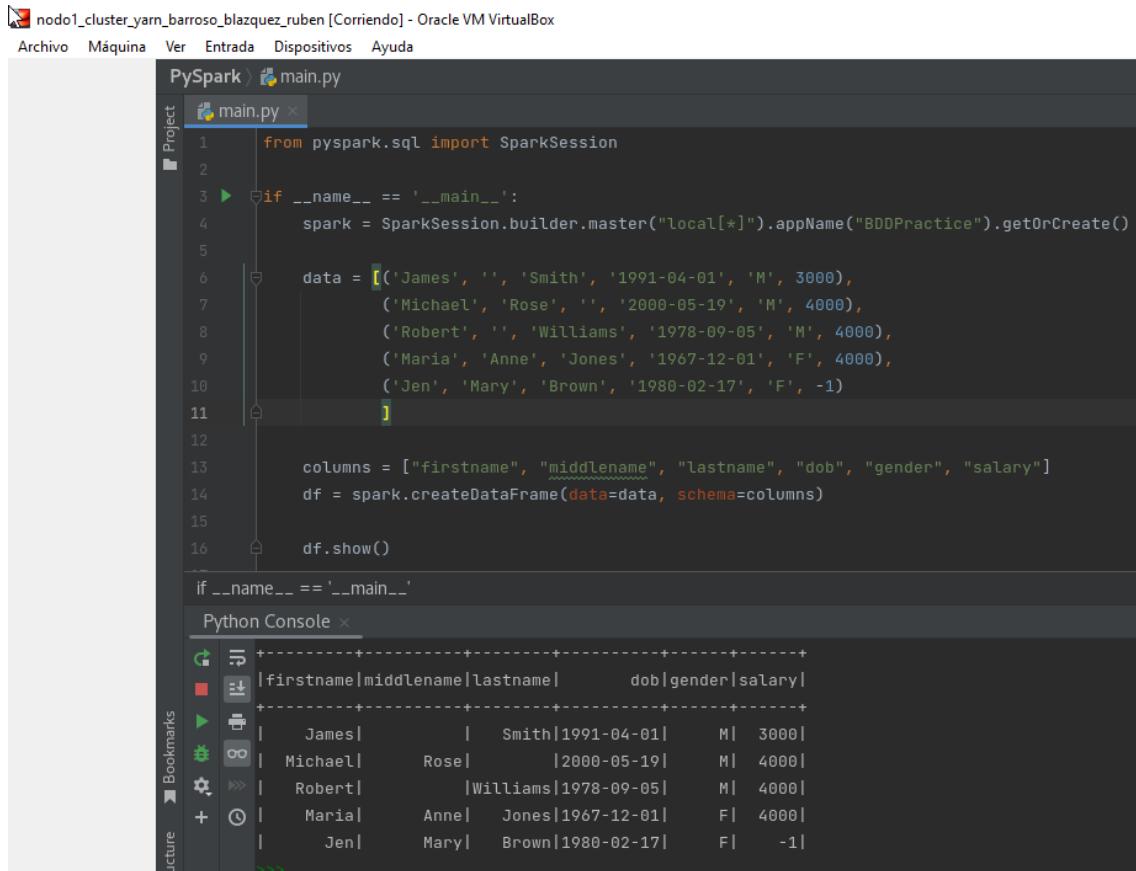
Además, los **RDDs** en **Spark** ofrecen tolerancia a fallos, lo que significa que si algún nodo del clúster falla, los datos se pueden recuperar automáticamente y el proceso de procesamiento se puede continuar sin pérdida de datos. Esto los hace especialmente útiles para el procesamiento de grandes volúmenes de datos en entornos de alta

disponibilidad.

PySpark DataFrame y SQL

¿Es PySpark más rápido que los pandas? Debido a la ejecución paralela en todos los núcleos en varias máquinas, **PySpark** ejecuta operaciones más rápido que pandas. En otras palabras, pandas **DataFrames** ejecuta operaciones en un solo nodo, mientras que **PySpark** se ejecuta en varias máquinas.

Para crear un **dataframe** con **pyspark** podemos usar la función `createDataframe`, en este caso no es necesario tener un contexto como para los **RDD** sino que con el objeto **SparkSession** se crean los **dataframes**.



The screenshot shows the PyCharm IDE interface. The top menu bar includes 'Archivo', 'Máquina', 'Ver', 'Entrada', 'Dispositivos', and 'Ayuda'. The title bar says 'nodo1_cluster_yarn_barroso_blezquez_ruben [Corriendo] - Oracle VM VirtualBox'. The 'Project' tool window on the left shows a file named 'main.py'. The code in 'main.py' is as follows:

```
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession.builder.master("local[*]").appName("BDDPractice").getOrCreate()
    data = [('James', '', 'Smith', '1991-04-01', 'M', 3000),
            ('Michael', 'Rose', '', '2000-05-19', 'M', 4000),
            ('Robert', '', 'Williams', '1978-09-05', 'M', 4000),
            ('Maria', 'Anne', 'Jones', '1967-12-01', 'F', 4000),
            ('Jen', 'Mary', 'Brown', '1980-02-17', 'F', -1)]
    columns = ["firstname", "middlename", "lastname", "dob", "gender", "salary"]
    df = spark.createDataFrame(data=data, schema=columns)
    df.show()
if __name__ == '__main__':
```

The 'Python Console' tool window at the bottom displays the output of the `df.show()` command:

firstname	middlename	lastname	dob	gender	salary
James		Smith	1991-04-01	M	3000
Michael	Rose		2000-05-19	M	4000
Robert		Williams	1978-09-05	M	4000
Maria	Anne	Jones	1967-12-01	F	4000
Jen	Mary	Brown	1980-02-17	F	-1

Al igual que hicimos en **SparkSQL**, aquí también podemos leer ficheros de **hdfs** con

```
df = spark.read.csv("/spark_files/housing.csv")
df.printSchema()
```

nodo1_cluster_yarn_barroso_blanco_ruben [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

PySpark > main.py

Project main.py

```
1  from pyspark.sql import SparkSession
2
3  if __name__ == '__main__':
4      spark = SparkSession.builder.master("local[*]").appName("BDDPractice").getOrCreate()
5
6      housing = spark.read.csv('/spark_files/housing.csv', header=True)
7
8      housing.show()
```

Python Console

latitud	longitud	precio_casa	media_casa	años_casa	precio_medio	distancia_c	distancia_d	distancia_mar	distance_e
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.12	241400.0	NEAR BAY
-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY
-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY

Bookmarks

Structure

Project

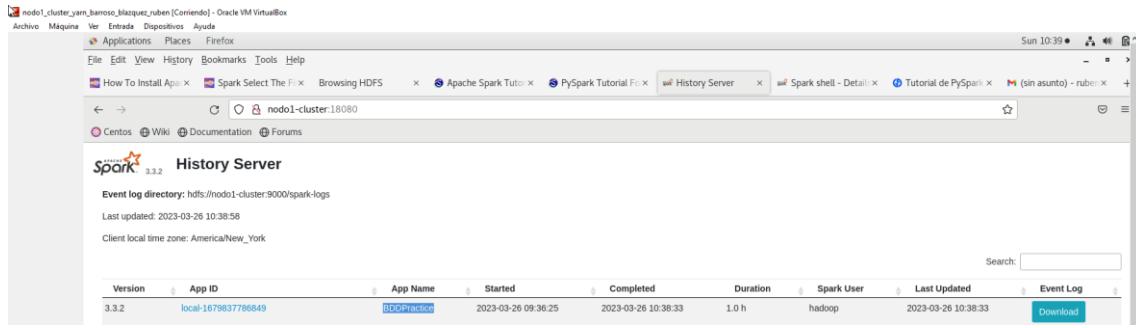
PySpark Streaming

Funciona igual que en `sparkSql`, lo único a tener en cuenta es que el método `readStream` que usamos anteriormente, en `PySpark` es un método del objeto `SparkSession`



Spart UI

Por último destacar el hecho de que cuando cerramos la sesión de **pySpark**, se crea la aplicación en nuestro SPARK UI, para poder ver todo lo que se ha realizado en dicha sesión



Actividad 8. (1,20 puntos)

8.- ML LIB Clasificador WIth PySpark

```
from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.classification import DecisionTreeClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

Crear una sesión de Spark

```
spark = SparkSession.builder.appName("IrisClassification").getOrCreate()
```

Cargar el conjunto de datos de iris

```
data = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("iris.csv")
```

Crear el vector de características

```
assembler = VectorAssembler(inputCols=["sepal_length", "sepal_width", "petal_length", "petal_width"], outputCol="features")

data = assembler.transform(data)
```

```
# Dividir el conjunto de datos en conjuntos de entrenamiento y prueba  
(trainingData, testData) = data.randomSplit([0.7, 0.3], seed=100)
```

```
# Crear un clasificador de árbol de decisión  
dt = DecisionTreeClassifier(labelCol="species", featuresCol="features")
```

```
# Entrenar el modelo  
model = dt.fit(trainingData)
```

```
# Realizar predicciones sobre los datos de prueba  
predictions = model.transform(testData)
```

```
# Evaluar la precisión del modelo  
evaluator = MulticlassClassificationEvaluator(labelCol="species",  
predictionCol="prediction", metricName="accuracy")  
  
accuracy = evaluator.evaluate(predictions)  
  
print("La precisión del modelo es:", accuracy)
```

```
# Cerrar la sesión de Spark  
spark.stop()
```

Segundo Ejemplo:

```
from pyspark import SparkContext  
  
from pyspark.sql import SparkSession
```

```
from pyspark.ml.feature import StringIndexer, VectorAssembler  
from pyspark.ml.classification import RandomForestClassifier  
  
  
sc = SparkContext("local", "Random Forest Example")  
spark = SparkSession(sc)  
  
  
data = spark.read.csv("iris.csv", header=True, inferSchema=True)
```

Luego, utilizamos la clase StringIndexer de MLlib para convertir la columna "especie" de etiquetas de texto a etiquetas numérica

```
indexer = StringIndexer(inputCol="especie", outputCol="label")  
  
data = indexer.fit(data).transform(data)
```

Después, utilizamos la clase VectorAssembler de MLlib para combinar las columnas de características (largo del sépalo, ancho del sépalo, largo del pétalo, ancho del pétalo) en una sola columna de características:

```
assembler = VectorAssembler(inputCols=["largo_sepalo", "ancho_sepalo",  
"largo_petalo", "ancho_petalo"], outputCol="features")  
  
data = assembler.transform(data)  
  
(trainingData, testData) = data.randomSplit([0.7, 0.3])  
  
rf = RandomForestClassifier(numTrees=10, featuresCol="features", labelCol="label")  
  
model = rf.fit(trainingData)  
  
predictions = model.transform(testData)  
  
accuracy = predictions.filter(predictions.label == predictions.prediction).count() /  
float(testData.count())  
  
print("Precisión en datos de prueba: %f" % accuracy)
```

Actividad 9. (1,20 puntos)

9.- ML LIB Regresión WIth PySpark

Predicción precio de una casa

```
from pyspark import SparkContext  
  
from pyspark.sql import SparkSession  
  
from pyspark.ml.feature import VectorAssembler  
  
from pyspark.ml.regression import LinearRegression
```

```
sc = SparkContext("local", "Linear Regression Example")
```

```
spark = SparkSession(sc)
```

```
data = spark.read.csv("datos.csv", header=True, inferSchema=True)
```

utilizamos la clase VectorAssembler de MLLib para combinar las columnas de tamaño y antigüedad en una sola columna de características:

```
assembler = VectorAssembler(inputCols=["tamaño", "antiguedad"],  
outputCol="features")  
  
data = assembler.transform(data)  
  
(trainingData, testData) = data.randomSplit([0.7, 0.3])
```

```
lr = LinearRegression(featuresCol="features", labelCol="precio")  
  
model = lr.fit(trainingData)
```

```
predictions = model.transform(testData)
```

```
r2 = model.summary.r2  
print("Coeficiente de determinación (R2) en datos de prueba: %f" % r2)
```

Actividad 10. (0.5 puntos)

10.- OTRAS Y CONCLUSIONES

Ha sido un proyecto interesante en el cual se han aprendido las bases de spark, tanto a nivel gráfico como a nivel de procesamiento de datos, ya que en el curso solo hemos aprendido pandas, y con spark te das cuenta que pandas no es lo mejor a la hora de tratar dataframes, sino que hay soluciones aceptables o mejores como es el caso de spark, que al usar el paralelismo y los RDD es capaz de procesar estos datos de manera más rápida.

Un problema que he tenido con este proyecto ha sido que al instalar spark, cuando intentaba hacer un start-all de los nodos del cluster de hadoop este levantaba solo spark, y no la información de hadoop, pero se ha podido solventar ya que hadoop tiene sus propios binarios para lanzar de manera manual los datanodes y namenodes sin necesidad del start-all.sh