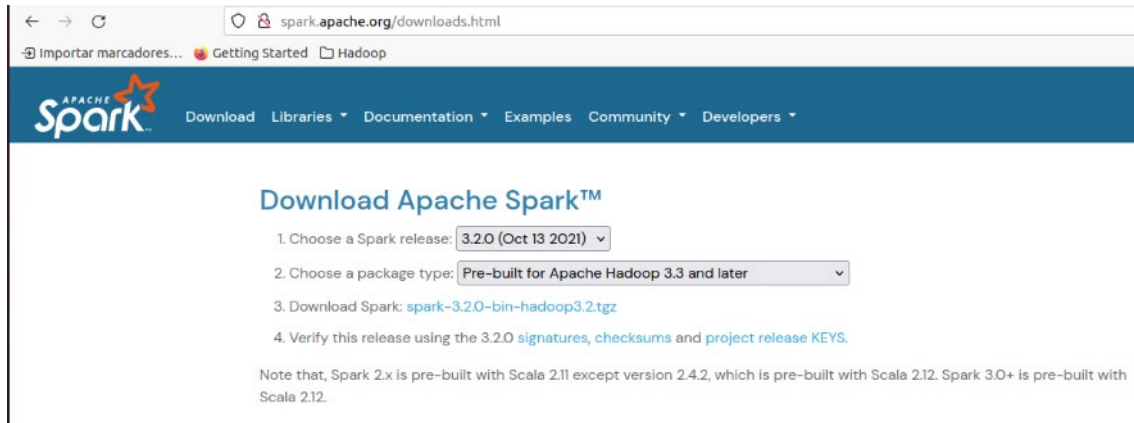


## 8. SPARK Configuración y primeros pasos

### 8.1. Instalar SPARK

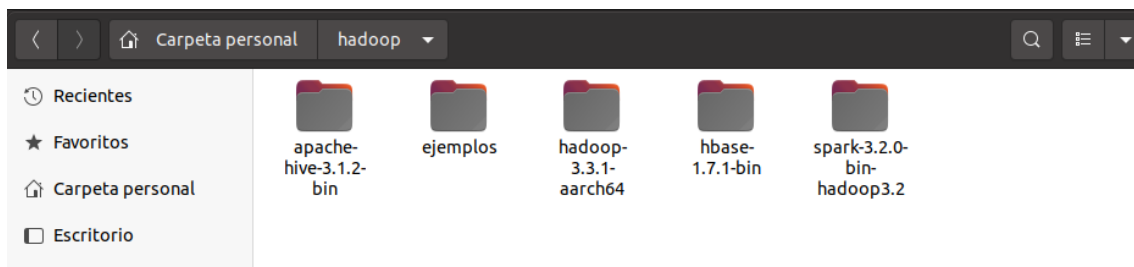
#### 8.1.1. Descargar SPARK

Descargar la versión 3.3.1 “Pre-Build for Apache Hadoop 3.3.and later” desde <http://spark.apache.org/downloads.html>



#### 8.1.2. Descomprimir y configurar

Guardar el archivo “spark-3.3.1-bin-hadoop3.tgz” y descomprimirlo en la carpeta del curso



Agregar a la variable PATH la ruta y crear la variable SPARK\_HOME

```
$ sudo gedit ~/.profile
```

Agregar las rutas en función de los directorios donde se hayan descomprimido los archivos

```
PATH="/home/hadoop/hadoop/spark-3.2.0-bin-hadoop3.2/bin:$PATH"
export SPARK_HOME="/home/hadoop/hadoop/spark-3.2.0-bin-hadoop3.2"
```

El resultado del fichero sería similar a:

```

28
29 #agrego las variables para hadoop y modifico el path
30 PATH="/home/hadoop/hadoop/hadoop-3.3.1-aarch64/hadoop-3.3.1/bin:$PATH"
31 PATH="/home/hadoop/hadoop/hadoop-3.3.1-aarch64/hadoop-3.3.1/sbin:$PATH"
32 PATH="/home/hadoop/hadoop/apache-hive-3.1.2-bin/bin:$PATH"
33 PATH="/home/hadoop/hadoop/hbase-1.7.1-bin/hbase-1.7.1/bin:$PATH"
34 PATH="/home/hadoop/hadoop/spark-3.2.0-bin-hadoop3.2/bin:$PATH"
35
36
37 export HADOOP_HOME="/home/hadoop/hadoop/hadoop-3.3.1-aarch64/hadoop-3.3.1"
38 export HADOOP_MAPRED_HOME=$HADOOP_HOME
39 export HADOOP_COMMON_HOME=$HADOOP_HOME
40 export HADOOP_HDFS_HOME=$HADOOP_HOME
41 export YARN_HOME=$HADOOP_HOME
42 export HIVE_HOME="/home/hadoop/hadoop/apache-hive-3.1.2-bin"
43 export HBASE_HOME="/home/hadoop/hadoop/hbase-1.7.1-bin/hbase-1.7.1"
44 export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
45 export SPARK_HOME="/home/hadoop/hadoop/spark-3.2.0-bin-hadoop3.2"
46
47

```

Reiniciar la sesión para asegurar que carga correctamente las variables

## 8.2. Funcionamiento básico

### 8.2.1. Inicio de Spark Shell

Iniciar el Shell de Spark con el comando spark-shell

```
$ spark-shell
```

```

Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1635348736768
).
Spark session available as 'spark'.
Welcome to

  ____              _
 / ___|  _ \   ___| | | |
 \___ \ |_) | / __| |_| |
  ___) ||  __/| | | | | |
 |____||_| \_| \___|_____|
                               version 3.2.0

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_292)
Type in expressions to have them evaluated.
Type :help for more information.

```

Inicia un contexto “sc” para ejecutar procesos. En la imagen indica que es la máquina <http://10.0.2.15:4040>. Accediendo a esta dirección desde el navegador vemos los servicios de spark Shell

The screenshot shows the Spark shell interface with the 'Environment' tab selected. It displays two sections: 'Runtime Information' and 'Spark Properties'.

Name	Value
Java Home	/usr/lib/jvm/java-8-openjdk-amd64/jre
Java Version	1.8.0_292 (Private Build)
Scala Version	version 2.12.15

Name	Value
spark.app.id	local-1635348736768
spark.app.name	Spark shell
spark.app.startTime	1635348735236

### 8.3. Comandos scala

Breve explicación de los comandos de scala utilizados en el ejemplo

Palabra Reservada	Descripción
VAL	Dar un nombre al resultado de una expresión
Sc.textFile	Carga un documento
.map	Crea elementos a partir de un separador
.filter	Filtra en función de unos criterios
!=	No es igual
&&	Operador lógico AND
.ReduceByKey	Agrupar por el campo clave
Math.max	Utiliza la librería matemática Math, para obtener el valor máximo
.foreach	Recorre todos los elementos de la lista
Println	Imprime por pantalla cada elemento en una línea

### 8.4. Ejemplo cálculo de temperatura con scala

Calcular la temperatura máxima desde un registro de temperaturas

Cargamos el archivo en una variable rdd de spark

```
scala> val lines =
sc.textFile("/home/hadoop/hadoop/ejemplos/spark/sample.txt")
```

Dividimos el archivo en líneas

```
scala> val records=lines.map(_.split("\t"))
```

Retiramos registros incorrectos filtrando las filas

```
scala> val filtered=records.filter(rec=>(rec(1) != "9999" &&
rec(2).matches("[01459]")))
```

Convertimos cada registro en una tupla clave-valor

```
scala> val tuplas=filtered.map(rec=>(rec(0).toInt, rec(1).toInt))
```

Agrupamos las tuplas y obtenemos el máximo para cada caso con `reduceByKey` y `Math.max`

```
scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.max(a,b))
```

Escribimos el resultado por pantalla con `println`

```
scala> maxTemps.foreach(println(_))
```

Muestra dos resultados por pantalla

```
scala> val lines = sc.textFile("/home/hadoop/hadoop/ejemplos/spark/sample.txt")
lines: org.apache.spark.rdd.RDD[String] = /home/hadoop/hadoop/ejemplos/spark/sample.txt MapPartitionsR

scala> val records=lines.map(_.split("\t"))
records: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[7] at map at <console>:23

scala> val filtered=records.filter(rec=>(rec(1) != "9999" && rec(2).matches("[01459]")))
filtered: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[8] at filter at <console>:23

scala> val tuplas=filtered.map(rec=>(rec(0).toInt, rec(1).toInt))
tuplas: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[9] at map at <console>:23

scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.max(a,b))
maxTemps: org.apache.spark.rdd.RDD[(Int, Int)] = ShuffledRDD[10] at reduceByKey at <console>:23

scala> maxTemps.foreach(println(_))
(1949,111)===== (1 + 1) / 2]
(1950,22)
scala>
```

También podemos crear una carpeta con el resultado con el comando `saveAsTextFile`. Procesa el RDD y lo guarda en la carpeta indicada

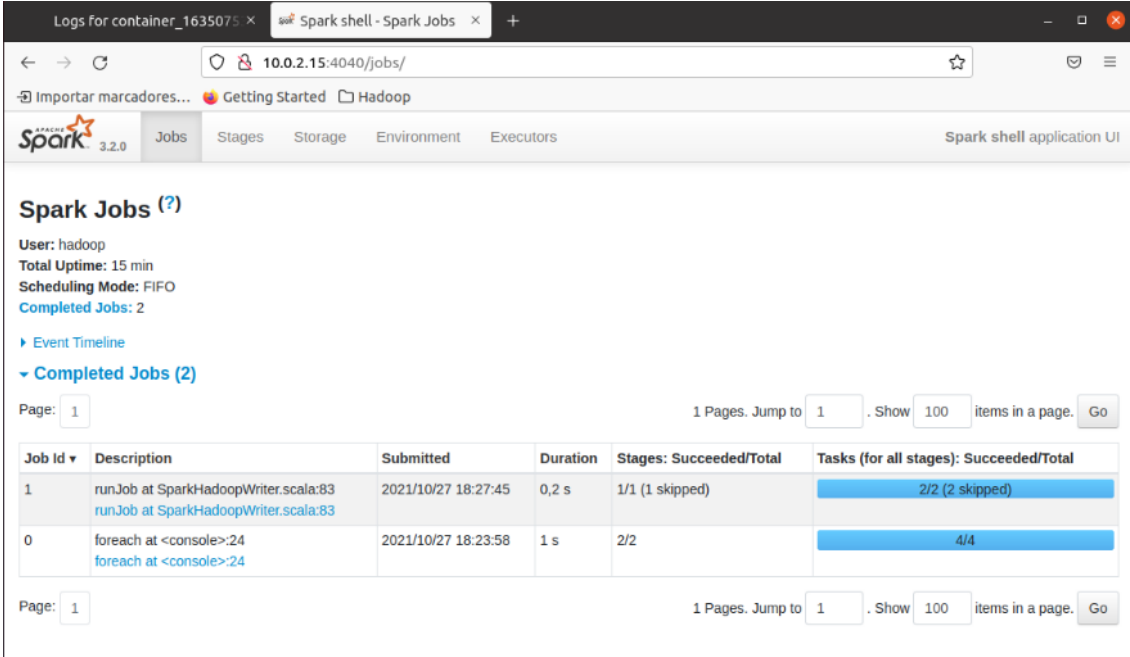
```
scala> maxTemps.saveAsTextFile("output")
```

Verificar desde shell de Ubuntu

```
$ cat output/*
```

```
hadoop@hadoop2:~$ cat output/*
(1950,22)
(1949,111)
```

Estos procesos los podemos consultar desde el UI de spark-shell



The screenshot shows the Spark Jobs UI for a container named 'container\_163507'. The browser address bar shows '10.0.2.15:4040/jobs/'. The UI includes tabs for 'Jobs', 'Stages', 'Storage', 'Environment', and 'Executors'. The 'Jobs' tab is active, displaying 'Spark Jobs (?)' with user 'hadoop', total uptime of 15 min, and scheduling mode of FIFO. It shows 2 completed jobs. Below this is a table of completed jobs:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	runJob at SparkHadoopWriter.scala:83 runJob at SparkHadoopWriter.scala:83	2021/10/27 18:27:45	0,2 s	1/1 (1 skipped)	2/2 (2 skipped)
0	foreach at <console>:24 foreach at <console>:24	2021/10/27 18:23:58	1 s	2/2	4/4

### 8.5. Ejemplo Cálculo de temperatura con Python

El mismo ejemplo anterior utilizando Python y expresiones lambda.

Crear un archivo con el código siguiente y guardarlo con el nombre MaxTemp.py

```
from pyspark import SparkContext
import re, sys

sc = SparkContext("local", "Max Temperature")
sc.textFile(sys.argv[1]) \
    .map(lambda s: s.split("\t")) \
    .filter(lambda rec: (rec[1] != "9999" and re.match("[01459]", rec[2]))) \
    .map(lambda rec: (int(rec[0]), int(rec[1]))) \
    .reduceByKey(max) \

    .saveAsTextFile(sys.argv[2])
```

Ejecutar el programa desde el Shell de ubuntu con el comando spark-submit. Suponiendo que el archivo MaxTemp.py los datos sample.txt están en la misma ruta.

```
$ spark-submit --master local MaxTemp.py sample.txt output
```

```
hadoop@hadoop2:~/hadoop/ejemplos/spark$ spark-submit --master local MaxTemp.py sample.txt output
21/10/27 18:55:41 WARN Utils: Your hostname, hadoop2 resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
21/10/27 18:55:41 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/10/27 18:55:41 INFO SparkContext: Running Spark version 3.2.0
21/10/27 18:55:41 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/10/27 18:55:42 INFO ResourceUtils: =====
21/10/27 18:55:42 INFO ResourceUtils: No custom resources configured for spark.driver.
```

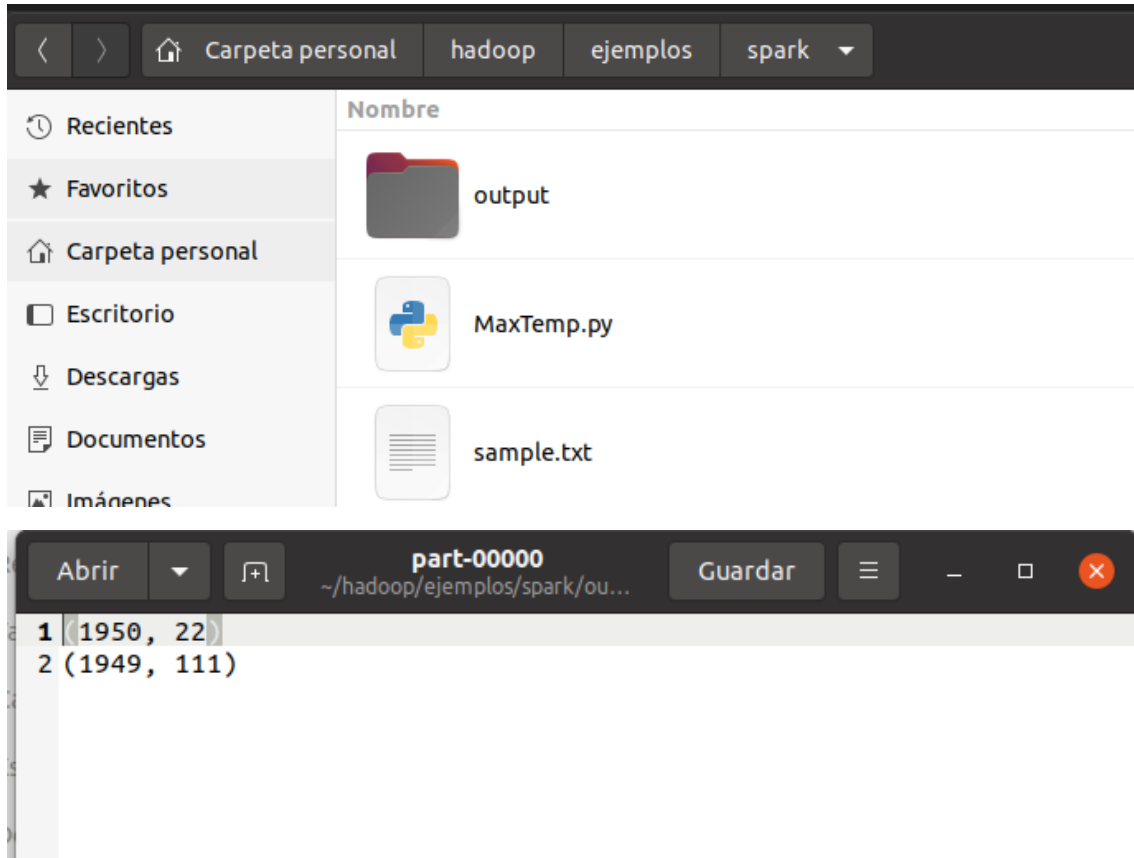
....

```

21/10/27 18:55:46 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
21/10/27 18:55:46 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/10/27 18:55:46 INFO MemoryStore: MemoryStore cleared
21/10/27 18:55:46 INFO BlockManager: BlockManager stopped
21/10/27 18:55:46 INFO BlockManagerMaster: BlockManagerMaster stopped
21/10/27 18:55:46 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/10/27 18:55:46 INFO SparkContext: Successfully stopped SparkContext
21/10/27 18:55:46 INFO ShutdownHookManager: Shutdown hook called
21/10/27 18:55:46 INFO ShutdownHookManager: Deleting directory /tmp/spark-8da475fb-f38b-4677-b5b5-fa1de91d2275
21/10/27 18:55:46 INFO ShutdownHookManager: Deleting directory /tmp/spark-dfe08441-3cb4-4110-a21b-2a6ad111f481
21/10/27 18:55:46 INFO ShutdownHookManager: Deleting directory /tmp/spark-8da475fb-f38b-4677-b5b5-fa1de91d2275/pyspark-370bddde-2
4cd-4f50-baf3-162135622558
hadoop@hadoop2:~/hadoop/ejemplos/spark$

```

El resultado está en la carpeta output



### 8.6. Ejemplo de optimización utilizando cache

Basándonos en el ejemplo 8.3 vamos a indicar que guarde las tuplas en la caché, que es lo mismo que indicarle que guarde el RDD en memoria. Este paso optimiza el rendimiento en el caso de múltiples consultas.

Cargamos el archivo en una variable rdd de spark

```
scala> val lines =
sc.textFile("/home/hadoop/hadoop/ejemplos/spark/sample.txt")
```

Dividimos el archivo en líneas

```
scala> val records=lines.map(_.split("\t"))
```

Retiramos registros incorrectos filtrando las filas

```
scala> val filtered=records.filter(rec=>(rec(1) != "9999" &&
rec(2).matches("[01459]")))
```

Convertimos cada registro en una tupla clave-valor

```
scala> val tuplas=filtered.map(rec=>(rec(0).toInt, rec(1).toInt))
```

Almacenamos las tuplas en la caché

```
scala> tuplas.cache()
```

Agrupamos las tuplas y obtenemos el máximo para cada caso con `reduceByKey` y `Math.max`. En la misma instrucción lo mostramos por pantalla

```
scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.max(a,b)).println(_)
```

Repetimos el proceso pero con la instrucción “min”

```
scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.min(a,b)).println(_)
```

```
scala> val lines = sc.textFile("/home/hadoop/hadoop/ejemplos/spark/sample.txt")
lines: org.apache.spark.rdd.RDD[String] = /home/hadoop/hadoop/ejemplos/spark/sample.txt MapPartitionsRDD[2] at textFile at <console>:23

scala> val records=lines.map(_.split("\t"))
records: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:23

scala> val filtered=records.filter(rec=>(rec(1) != "9999" && rec(2).matches("[01459]")))
filtered: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at filter at <console>:23

scala> val tuplas=filtered.map(rec=>(rec(0).toInt, rec(1).toInt))
tuplas: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[4] at map at <console>:23

scala> tuplas.cache()
res0: tuplas.type = MapPartitionsRDD[4] at map at <console>:23

scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.max(a,b)).foreach(println(_))
(1950,22)
(1949,111)
maxTemps: Unit = ()

scala> val maxTemps=tuplas.reduceByKey((a,b)=>Math.min(a,b)).foreach(println(_))
(1950,-11)
(1949,78)
maxTemps: Unit = ()

scala>
```

## 8.7. Otros ejemplos

### 8.7.1. Ejecutar `mnmcounr.py` sobre el fichero `mnmcounr_dataset.csv`

Este ejemplo utiliza `DataFrame API` para facilitar la programación.

### 8.7.2. Ejecutar `Ejemplo_Eschema.py`

Este ejemplo aplica un esquema a un conjunto de datos integrados en el código.

### 8.7.3. Ejecutar FireCalls.py sobre el fichero sf-fire-calls.csv

Este ejemplo aplica un esquema a un conjunto de datos y realiza múltiples operaciones de transformación, filtrado, agrupamiento y cálculo.