

APACHE SPARK - Big Data



ÍNDICE

1. Introducción.....	4
1.1 Introducción a Apache Spark.....	4
1.2 Evolución de Apache Spark.....	4
1.3 ¿Por Qué Spark?.....	5
1.4 Componentes de Spark.....	6
2. Arquitectura de un clúster de Apache Spark.....	7
3. API de Spark.....	8
3.1 Tipos de transformaciones de un RDD.....	9
3.2 Transformaciones de un RDD.....	10
3.3 Acciones de un RDD.....	13
3.4 Persistencia de los RDD.....	14
4. Spark VS Hadoop.....	16
4.1 Rendimiento.....	16
4.2 Usabilidad.....	16
4.3 Costes.....	17
4.4 Compatibilidad	17
4.5 Procesamiento de datos.....	17
4.6 Tolerancia a fallos.....	17
4.7 Seguridad.....	18
4.8 Conclusiones.....	18
5. Topología de red.....	19
6. Características y requisitos.....	20
7. Instalación y configuración de Spark en Debian.....	21
7.1 Instalación de Java.....	21
7.2 Instalación de Scala.....	22
7.3 Instalación y configuración de OpenSSH.....	22
7.4 Instalación y configuración de Apache Spark.....	23
7.5 Creación de los servidores esclavos.....	24
7.6 Configuración de OpenSSH en master.....	26
7.7 Configuración de Apache Spark en el servidor master.....	27
8. Instalación y configuración de Spark en Windows 10.....	31
8.1 Instalación de Java SE Development Kit 8.....	31
8.2 Instalación de Scala.....	33
8.3 Instalación de Spark.....	35
8.4 Instalación de Winutils y establecimiento de variables.....	36
9. Conexión de MySQL con Spark.....	42
9.1 Ejecución de algunas consultas al dataframe.....	44
9.2 Creación de un DataFrame basado en el contenido de un fichero JSON.....	45
10. Streaming de Tweet's.....	46
11. Zeppelin + Spark.....	48
11.1 ¿Qué es Zeppelin?.....	48
11.2 Instalación y configuración de Zeppelin.....	48
11.3 Ejemplos.....	50

12. Problemas encontrados.....	52
12.1: cat: /release: No such file or directory.....	52
12.2 HOSTNAME: HOSTNAME: Name or service not known.....	52
12.3 The root scratch dir: /tmp/hive on HDFS should be writable.....	52
12.4 java.lang.NoClassDef FoundError: org/apache/spark/Logging.....	53
12.5 warn "replicated to only 0 peer(s) instead of 1 peers".....	53
13. Bibliografía.....	54

1. INTRODUCCIÓN

1.1 Introducción a Apache Spark

Hoy en día se genera gran cantidad de datos en campos como la industria y la ciencia, por ello, es necesario herramientas como Apache Spark para trabajar con estos datos.

Por otra parte, algunas industrias están utilizando Hadoop para almacenar, procesar y analizar grandes volúmenes de datos. Hadoop se basa en el modelo de programación *MapReduce* y permite una solución de computación que es escalable, tolerante a fallos, flexible y rentable. La principal preocupación que presenta Hadoop es mantener la velocidad de espera entre las consultas y el tiempo para ejecutar el programa en el procesamiento de grandes conjuntos de datos.

Posteriormente salió a la luz Apache Spark introducido por la empresa *Apache Software Foundation* para acelerar el proceso de software de cálculo computacional Hadoop. Aunque es importante mencionar que Apache Spark depende de Hadoop, ya que lo utiliza para propósitos de almacenamiento.

Apache Spark es una infraestructura informática de clúster de código abierto usado con frecuencia para cargas de trabajo de Big Data¹. Además ofrece un desempeño rápido, ya que el almacenamiento de datos se gestiona en memoria, lo que mejora el desempeño de cargas de trabajo interactivas sin costos de E/S (periféricos de entrada/salida). Por otro lado, Apache Spark es compatible con las bases de datos de gráficos, el análisis de transmisiones, el procesamiento general por lotes, las consultas ad-hoc y el aprendizaje automático.

Empresas como Alibaba Taobao y Tencent, ya están utilizando Apache Spark como gestor de datos. La empresa Tencent posee actualmente 800 millones de usuarios activos, generando un total de 700 TB de datos procesados al día en un clúster de más de 8000 nodos de computación.

1.2 Evolución de Apache Spark

Spark fue un subproyecto de Hadoop desarrollado en 2009 por Matei Zaharia en la Universidad de Berkeley AMPLab, que posteriormente fue de código abierto bajo una licencia BSD.

Posteriormente, en 2013, el proyecto Spark fue donada a la empresa *Apache Software Foundation*, y actualmente se ha convertido en un proyecto de alto nivel.

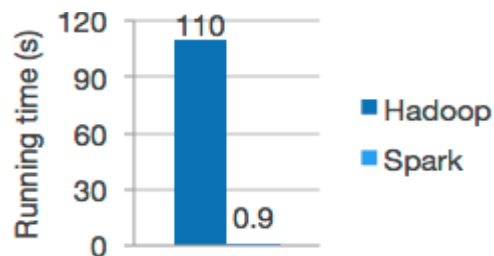
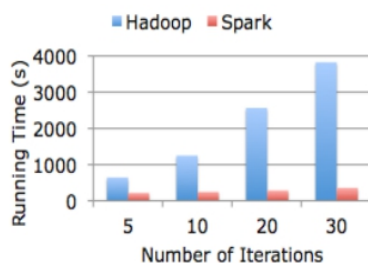
¹ *Big Data*: Conjunto de herramientas utilizadas para gestionar, manipular y analizar grandes volúmenes de datos que no pueden ser gestionados por herramientas informáticas tradicionales como una base de datos.

1.3 ¿Por qué utilizar Spark?

Alguna de las razones por las que se debería usar Apache Spark son:

- **Velocidad:** Apache Spark es capaz de ejecutar hasta 100 veces más rápido aplicaciones ejecutadas en memoria y 10 veces más rápido cuando se ejecuta en HDD. Esto se debe principalmente a la reducción de número de operaciones de lectura / escritura en el disco y al nuevo almacenamiento de datos de procesamiento intermedio en memoria. Gracias a esta mejora en la velocidad, Spark ofrece una experimentación más veloz, mayor interactividad y mayor productividad para los analistas.

En la siguiente ilustraciones se pueden observar como Apache Spark posee una mayor velocidad de procesamiento en comparación a Hadoop MapReduce:

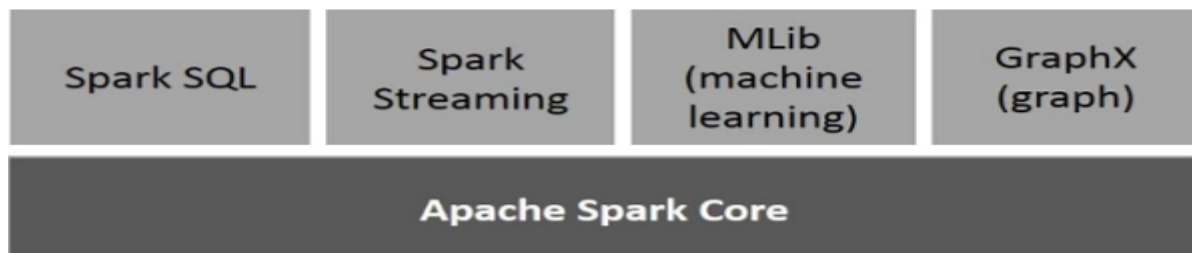


- **Potencia:** Apache Spark nos permite realizar más operaciones que Hadoop MapReduce: integración con lenguaje R (Spark R), procesamiento de streaming, cálculo de grafos (GraphX), machine learning (MLlib), y análisis interactivos. Gracias a esta mejora en la potencia, se podrá desplegar nuevos proyectos de Big Data con menos presupuesto y con soluciones más completas.
- **Fácil uso:** Uno de los principales problemas que poseía Hadoop, es que requería de usuarios con niveles avanzados de MapReduce o programación avanzada en Java. Este inconveniente desaparece con la llegada de Spark, ya que gracias a la API nos permite programar en R, Python, Scala e incluso en Java. Además nos permite programar interactivamente en Python y Scala desde la consola directamente.
- **Entiende SQL:** Gracias al módulo Spark SQL se permite la consulta de datos estructurados y semi-estructurados utilizando lenguaje SQL o gracias a la API, la cual puede ser utilizada con Java, Scala, Python o R.
- **Excelente comunidad:** Apache Spark presenta una comunidad cada vez más activa, en la que los desarrolladores mejoran las características de la plataforma, y ayudan al resto de programadores a implementar soluciones o resolver problemas.
- **Procesar y analizar datos que con las tecnologías actuales era imposible.**
- **Escalabilidad:** Spark nos da la posibilidad de ir incrementando nuestro clúster a medida que vamos necesitando más recursos.

1.4 Componentes de Spark

Spark ha tenido un gran reconocimiento en el mundo del Big Data debido a su rendimiento computacional y a su amplia variedad de librerías. Por ello, una de las características más reconocibles de Spark es ser una plataforma de plataformas, es decir, un “todo en uno”.

Como se puede observar en la siguiente ilustración se muestra los diferentes componentes de Spark:



- **Spark Core:** Es el “corazón” de Spark, responsable de gestionar las funciones como la programación de las tareas.
- **Spark SQL:** Es un módulo ubicado en la parte superior del núcleo de Spark, que introduce una nueva idea de datos llamada *SchemaRDD*, el cual proporciona soporte para datos estructurados y semi-estructurados. Gracias a este componente, se permite combinar consultas SQL con programas de Spark y además, se permite la consulta de datos estructurados utilizando lenguaje SQL o con la API que nos ofrece Apache Spark.
- **Spark Streaming:** Spark es capaz de realizar análisis de streaming sin problemas, gracias a la gran velocidad de programación de su núcleo. Además, gracias a la API se permite crear aplicaciones escalables e intolerantes a fallos de streaming. Otra ventaja que posee Spark, es que es capaz de procesar grandes datos en tiempo real, mientras que MapReduce, solamente es capaz de gestionar datos en lotes. Gracias a esta ventaja, los datos son analizados conforme entran, sin tiempo de latencia y a través de un proceso de gestión en continuo tránsito.
- **MLib:** Es un framework de aprendizaje ubicado en la parte superior del núcleo de Spark, que tiene como finalidad hacer práctico, escalable y fácil el “machine learning”. Este framework posee un conjunto de algoritmos y utilidades, como por ejemplo: clasificación, regresión, clustering, filtrado colaborativo y reducción de dimensionalidad. Gracias a la arquitectura de Spark basada en memoria distribuida, se ha conseguido una velocidad de hasta nueve veces superior a la implementación basada en disco de Apache Mahout² y escalas mejor que Vowpal Wabbit³.
- **GraphX:** Es un entorno de procesamiento gráfico ubicado en la parte superior de Spark, el cual proporciona una API para gráficos y cálculo gráfico en paralelo.

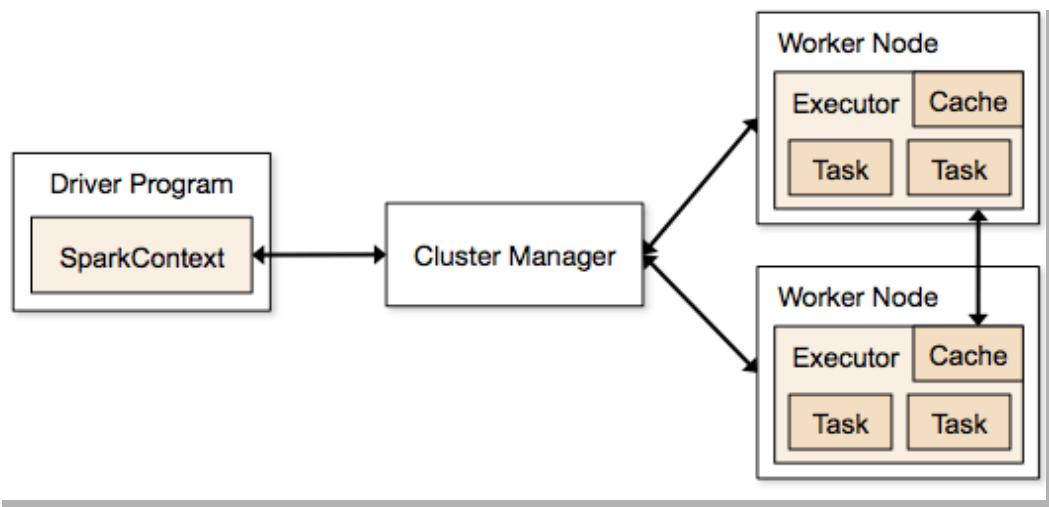
² *Apache Mahout* es un proyecto de código abierto que se utiliza principalmente para crear algoritmos escalables de aprendizaje automático.

³ *Vowpal Wabbit* es un proyecto iniciado en Yahoo! Research y que continúa en Microsoft Research para diseñar un algoritmo de aprendizaje rápido, escalable y útil.

2. ARQUITECTURA DE UN CLÚSTER DE APACHE SPARK

Hay varios datos útiles que destacar sobre esta arquitectura:

1. Las aplicaciones de Spark son ejecutadas independientemente y estas son coordinadas por el objeto Spark SparkContext del programa principal (*Driver Program*⁴).
2. SparkContext es capaz de conectarse a gestores de clúster (*Cluster Manager*), los cuales se encargan de asignar recursos en el sistema. Hay varios tipos de gestores de clúster:
 - [Standalone](#): sencillo gestor de clústeres, incluido con Spark, que facilita la creación de un clúster.
 - [Apache Mesos](#): es un gestor de clústeres un poco más avanzado que el anterior, que puede ejecutar Hadoop, MapReduce y aplicaciones de servicio.
 - [Hadoop YARN](#): es el gestor de recursos en Hadoop 2.
3. Una vez conectados, Spark puede encargarse de que se creen ejecutores (*executors*), encargados de ejecutar tareas (tasks) en los nodos del clúster.



Cada aplicación posee sus propios ejecutores, los cuales ejecutan tareas en varios subprocesos. Gracias a esto, se consigue aislar las aplicaciones entre sí, tanto en el lado de la programación (cada controlador programa sus propias tareas), como en el lado del ejecutor (las tareas de las diferentes aplicaciones se ejecutan en distintas JVM⁵s). Sin embargo, esto significa que los datos no se pueden compartir entre diferentes aplicaciones Spark

4 Driver Program: proceso que ejecuta la función "main ()" y crea el SparkContext.

5 JVM: máquina virtual Java de proceso nativo.

3. API DE SPARK

Antes de empezar con la configuración e instalación de Spark, es conveniente conocer algunos conceptos básicos de la API⁶:

- **SparkContext**: es la parte principal de la API de Apache Spark, donde realizaremos todas las operaciones. Gracias a SparkContext se facilita la conexión de la aplicación con un clúster Spark.
- **Resilient Distributed Dataset (RDD)**: son grupos de datos de solo lectura generados tras hacer acciones en los datos originales. Estos RDDs permiten cargar gran cantidad de datos en memoria para un veloz procesamiento y además pueden dividirse para ser tratado de forma paralela. Por lo que, los programadores son capaces de realizar operaciones en grandes cantidades de datos de forma rápida y tolerante a fallos. Además, la API de Spark nos permite la conexión con repositorios de datos, como Hadoop, Cassandra, y creación de los RDD's.

Existen varias formas de generar RDD's:

- ➔ Obtener datos de un *fichero*.
- ➔ Obtener datos almacenados en *memoria*.
- ➔ Obtener datos de otro *RDD*.

Por otra parte, las operaciones que se pueden realizar sobre un RDD son:

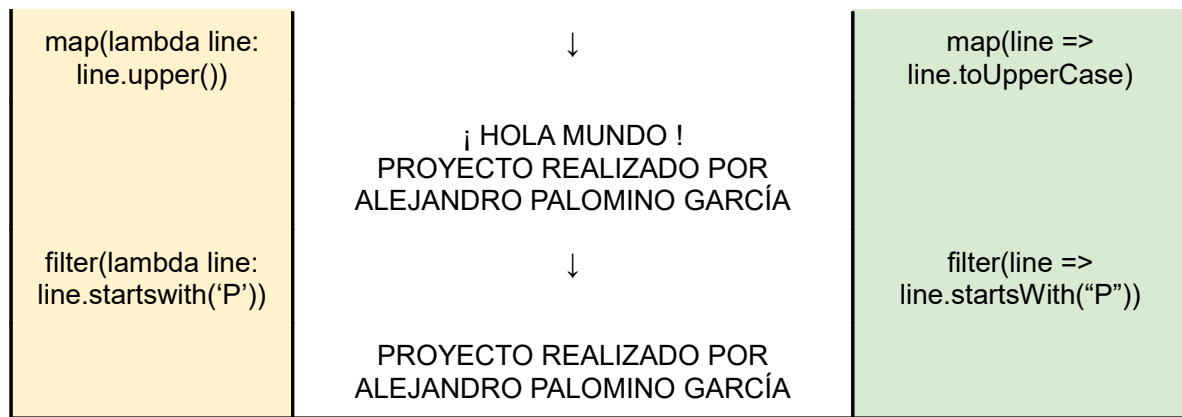
- ➔ **Acciones**: devuelven un valor a la aplicación. Algunos ejemplos sobre acciones sobre RDD son: contar los elementos que posee el mismo (*count()*), guardar su contenido en un fichero (*saveAsTextFile(fichero)*), devolver un array⁷ de los primeros *n* elementos del RDD (*take(n)*) o devolver un array con todos los elementos de un RDD (*collect()*).
- ➔ **Transformaciones**: consiste en obtener un nuevo RDD tras transformar el original. Algunos ejemplos sobre transformaciones sobre RDD son: crear un nuevo RDD en base a una función (*map(función)* / *filter(función)*).

Un ejemplo de transformación de un RDD sería:

Python	Frase	Scala
	¡ hola mundo ! Proyecto realizado por Alejandro Palomino García	

⁶ **API**: es un conjunto de comandos, funciones y protocolos informáticos que facilitan y permiten a los desarrolladores la creación de determinados programas para ciertos sistemas operativos

⁷ **Array**: conjunto de variables de una misma clase.



NOTA: El procesamiento de datos del ejemplo anterior no se llevará a cabo hasta que se realice una acción (por ejemplo la acción de tipo `count()`).

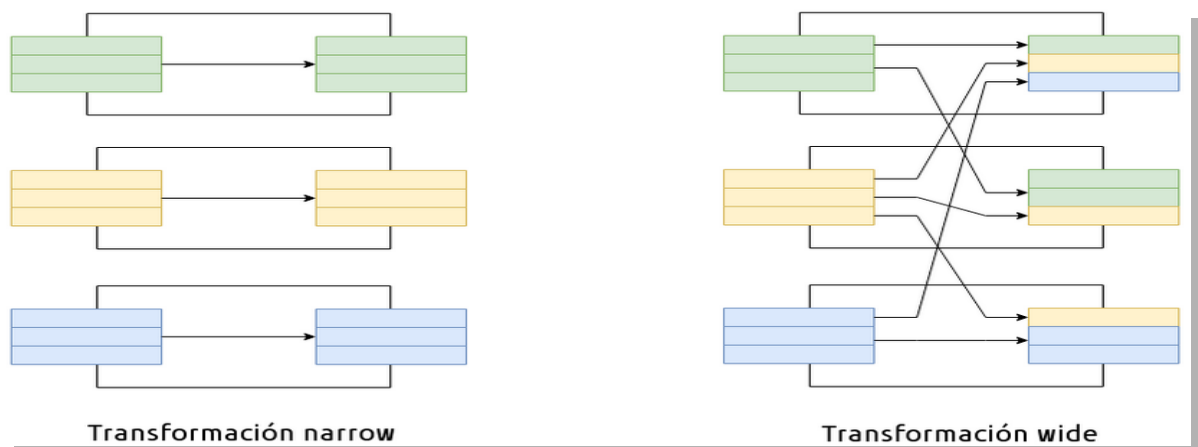
¡ RECOMENDACIÓN !

Para una mayor eficiencia es recomendable almacenar RDD's en memoria para un rápido acceso a los mismos. Para hacer el almacenamiento en memoria se puede utilizar la función `"cache()"`.

3.1 Tipos de transformaciones de un RDD

Existen dos tipos de operaciones de transformación, ya que posiblemente los datos a procesar se encuentren en diferentes RDD's:

- **Narrow:** utilizado cuando los datos a tratar están ubicados en la misma partición del RDD y no hace falta mezclar dichos datos. Algunos ejemplos de funciones para este tipo son: `filter()`, `sample()`, `map()` o `flatMap()`.
- **Wide:** utilizado cuando los datos a tratar están ubicados en diferentes particiones de un RDD y es necesario mezclar dichas particiones. Algunos ejemplos de funciones para este tipo son: `groupByKey()` o `reduceByKey()`.



3.2 Transformaciones de un RDD

Las transformaciones de Spark utilizan un mecanismo de “evaluación perezosa”, es decir, las transformaciones en un RDD no se ejecutan hasta que se realiza alguna acción sobre el mismo. Las transformaciones que se puede realizar sobre un RDD son:

Transformación	Definición	Ejemplo
map(func)	Devuelve un nuevo RDD tras pasar cada elemento del RDD original a través de una función.	<pre>val v1 = sc.parallelize(List(2, 4, 8)) val v2 = v1.map(_ * 2) v2.collect res0: Array[Int] = Array(4, 8, 16)</pre>
filter(func)	Realiza un filtrado de los elementos del RDD original para devolver un nuevo RDD con los datos filtrados.	<pre>val v1 = sc.parallelize(List("ABC", "BCD", "DEF")) val v2 = v1.filter(_.contains("A")) v2.collect res0: Array[String] = Array(ABC)</pre>
flatMap(func)	Parecido a la operación map, pero la función devuelve una secuencia de valores.	<pre>val x = sc.parallelize(List("Ejemplo proyecto Alejandro", "Hola mundo"), 2) val y = x.map(x => x.split(" ")) // split(" ") returns an array of words y.collect res0: Array[Array[String]] = Array(Array(Ejemplo, proyecto, Alejandro), Array(Hola, mundo)) val y = x.flatMap(x => x.split(" ")) y.collect res1: Array[String] = Array(Ejemplo, proyecto, Alejandro, Hola, mundo)</pre>
mapPartitions (func)	Similar a la operación map, pero se ejecuta por separado en cada partición del RDD.	<pre>val a = sc.parallelize(1 to 9, 3) def myfunc[T](iter: Iterator[T]) : Iterator[(T, T)] = { var res = List[(T, T)]() var pre = iter.next while (iter.hasNext) {val cur = iter.next; res ::= (pre, cur) pre = cur;} res.iterator} a.mapPartitions(myfunc).collect res0: Array[(Int, Int)] = Array((2,3), (1,2), (5,6), (4,5), (8,9), (7,8))</pre>
sample(withReplacement, fraction, seed)	Muestra una fracción de los datos, con o sin replazo, utilizando una semilla que genera número aleatorios.	<pre>val randRDD = sc.parallelize(List((7,"cat"), (6, "mouse"),(7, "cup"), (6, "book"), (7, "tv"), (6, "screen"), (7, "heater"))) val sampleMap = List((7, 0.4), (6, 0.6)).toMap randRDD.sampleByKey(false, sampleMap,42).collect res0: Array[(Int, String)] = Array((6,book), (7,tv), (7,heater))</pre>
union(otherDataset)	Devuelve un nuevo RDD con la unión de los elementos de los RDDs seleccionados.	<pre>val a = sc.parallelize(1 to 3, 1) val b = sc.parallelize(5 to 7, 1) a.union(b).collect() res0: Array[Int] = Array(1, 2, 3, 5, 6, 7)</pre>
intersection(otherDataset)	Devuelve los elementos de los RDDs que son iguales.	<pre>val x = sc.parallelize(1 to 20) val y = sc.parallelize(10 to 30) val z = x.intersection(y) z.collect</pre>

		<code>res0: Array[Int] = Array(16, 14, 12, 18, 20, 10, 13, 19, 15, 11, 17)</code>
distinct ([numTasks])	Devuelve los elementos de los RDDs que son distintos.	<pre>val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2) c.distinct.collect res0: Array[String] = Array(Dog, Gnu, Cat, Rat)</pre>
groupByKey ([numTasks])	Similar al <i>grupoBy</i> , realiza el agrupamiento por clave de un conjunto de datos, pero en lugar de suministrar una función, el componente clave de cada par se presentará automáticamente al particionador.	<pre>val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2) val b = a.keyBy(_.length) b.groupByKey.collect res0: Array[(Int, Seq[String])] = Array((4,ArrayBuffer(lion)), (6,ArrayBuffer(spider)), (3,ArrayBuffer(dog, cat)), (5,ArrayBuffer(tiger, eagle)))</pre>
reduceByKey (func, [numTasks])	Devuelve un conjunto de datos de pares (K, V) donde los valores de cada clave son agregados usando la función de reducción dada.	<pre>val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2) val b = a.map(x => (x.length, x)) b.reduceByKey(_ + _).collect res0: Array[(Int, String)] = Array((3,dogcatowlgnuant))</pre>
aggregateByKey (zeroValue)(seqOp, combOp, [numTasks])	Devuelve un conjunto de datos de pares (K, U) donde los valores de cada clave se agregan utilizando las funciones combinadas dadas y un valor por defecto de: "cero".	<pre>val nombres = sc.parallelize(List(("David", 6), ("Abby", 4), ("David", 5), ("Abby", 5))) nombres.aggregateByKey(0)((k,v) => v.toInt+k, (v,k) => k+v).collect res0: Array[(String, Int)] = Array((Abby,9), (David,11))</pre>
sortByKey ([ascending], [numTasks])	Esta función ordena los datos del RDD de entrada y los almacena en un nuevo RDD.	<pre>val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2) val b = sc.parallelize(1 to a.count.toInt, 2) val c = a.zip(b) c.sortByKey(true).collect res0: Array[(String, Int)] = Array((ant,5), (cat,2), (dog,1), (gnu,4), (owl,3)) c.sortByKey(false).collect res1: Array[(String, Int)] = Array((owl,3), (gnu,4), (dog,1), (cat,2), (ant,5))</pre>
join (otherDataset, [numTasks])	Realiza una unión interna utilizando dos RDD de valor clave. Cuando se introduce conjuntos de datos de tipo (K, V) y (K, W), se devuelve un conjunto de datos de (K, (V, W)) para cada clave.	<pre>val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3) val b = a.keyBy(_.length) val c = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3) val d = c.keyBy(_.length) b.join(d).collect res0: Array[(Int, (String, String))] = Array((6, (salmon,salmon)), (6,(salmon,rabbit)), (6, (salmon,turkey)), (6,(salmon,salmon)), (6, (salmon,rabbit)), (6,(salmon,turkey)), (3,(dog,dog)), (3,(dog,cat)), (3,(dog,gnu)), (3,(dog,bee)), (3, (rat,dog)), (3,(rat,cat)), (3,(rat,gnu)), (3,(rat,bee)))</pre>
cogroup (otherDataset, [numTasks])	Un conjunto muy potente de funciones que permiten agrupar hasta tres valores	<pre>val a = sc.parallelize(List(1, 2, 1, 3), 1) val b = a.map(_,"b") val c = a.map(_,"c")</pre>

	claves de RDDs utilizando sus claves.	<pre>val d = a.map(_,"d") b.cogroup(c, d).collect res0: Array[(Int, (Iterable[String], Iterable[String], Iterable[String]))] = Array((2,(ArrayBuffer(b),ArrayBuffer(c),ArrayBuffer(d))), (3,(ArrayBuffer(b),ArrayBuffer(c),ArrayBuffer(d))), (1,(ArrayBuffer(b, b),ArrayBuffer(c, c),ArrayBuffer(d, d))))</pre>
cartesian (otherDatas et)	Calcula el producto cartesiano entre dos RDD ,es decir cada elemento del primer RDD se une a cada elemento del segundo RDD, y los devuelve como un nuevo RDD.	<pre>val x = sc.parallelize(List(1,2,3,4,5)) val y = sc.parallelize(List(6,7,8,9,10)) x.cartesian(y).collect res0: Array[(Int, Int)] = Array((1,6), (1,7), (1,8), (1,9), (1,10), (2,6), (2,7), (2,8), (2,9), (2,10), (3,6), (3,7), (3,8), (3,9), (3,10), (4,6), (5,6), (4,7), (5,7), (4,8), (5,8), (4,9), (4,10), (5,9), (5,10))</pre>
pipe (command, [envVars])	Toma los datos RDD de cada partición y los envía a través de stdin a un shell-command	<pre>val a = sc.parallelize(1 to 9, 3) a.pipe("head -n 1").collect res0: Array[String] = Array(1, 4, 7)</pre>
coalesce (numPartitions)	Disminuye el número de particiones en el RDD al número especificado (numPartitions)	<pre>val y = sc.parallelize(1 to 10, 10) val z = y.coalesce(2, false) z.partitions.length res0: Int = 2</pre>
repartition (numPartitions)	Reorganiza aleatoriamente los datos en el RDD para crear más o menos particiones.	<pre>val x = (1 to 12).toList val numbersDf = x.toDF("number") numbersDf.rdd.partitions.size res0: Int = 4 Partition 00000: 1, 2, 3 Partition 00001: 4, 5, 6 Partition 00002: 7, 8, 9 Partition 00003: 10, 11, 12 val numbersDfR = numbersDf.repartition(2) Partition A: 1, 3, 4, 6, 7, 9, 10, 12 Partition B: 2, 5, 8, 11</pre>
repartitionAndSortWithinPartitions (partitioner)	Reparte el RDD de acuerdo con el particionador dado y, dentro de cada partición resultante, clasifica los registros por sus claves. Como se puede observar en el ejemplo pedimos que los datos sean organizado en dos particiones: A y C como una particion y, B y D como otra.	<pre>>> pairs = sc.parallelize([["a",1], ["b",2], ["c",3], ["d",3]]) >> pairs.collect() # Output [['a', 1], ['b', 2], ['c', 3], ['d', 3]] >>pairs.repartitionAndSortWithinPartitions(2).glom().collect() # Output [[('a', 1), ('c', 3)], [('b', 2), ('d', 3)]] // Reorganización basado en cierta condición. >>pairs.repartitionAndSortWithinPartitions(2,partitionFunc=lambda x: x == 'a').glom().collect() # Output [[('b', 2), ('c', 3), ('d', 3)], [('a', 1)]]</pre>

NOTA: Todos los ejemplos están programados en Scala, excepto el último que está en Python.

3.3 Acciones de un RDD

Las acciones que se puede realizar sobre un RDD son:

Transformación	Definición	Ejemplo
reduce(func)	Agrega los elementos del dataset usando una función. Esta función debe ser conmutativa y asociativa para que pueda calcularse correctamente en paralelo.	<pre>val a = sc.parallelize(1 to 100, 3) a.reduce(_ + _) res0: Int = 5050</pre>
collect()	Convierte un RDD en un array ⁸ y lo muestra por pantalla.	<pre>val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"), 2) c.collect res0: Array[String] = Array(Gnu, Cat, Rat, Dog, Gnu, Rat)</pre>
count()	Devuelve el número de elementos del dataset.	<pre>val a = sc.parallelize(1 to 4) a.count res0: Long = 4</pre>
first()	Devuelve el primer elemento del conjunto de datos	<pre>val c = sc.parallelize(List("Gnu", "Cat", "Rat", "Dog"), 2) c.first res0: String = Gnu</pre>
take(n)	Devuelve un array con los primeros <i>n</i> elementos del dataset.	<pre>val b = sc.parallelize(List("dog", "cat", "ape", "salmon", "gnu"), 2) b.take(2) res0: Array[String] = Array(dog, cat)</pre>
takeSample(withReplacement, num, [seed])	Devuelve un array con una muestra aleatoria de elementos numéricos del dataset, con o sin sustitución, con la opción de especificar opcionalmente una semilla de generador de números aleatorios.	<pre>val x = sc.parallelize(1 to 200, 3) x.takeSample(true, 20, 1) res0: Array[Int] = Array(74, 164, 160, 41, 123, 27, 134, 5, 22, 185, 129, 107, 140, 191, 187, 26, 55, 186, 181, 60)</pre>
takeOrdered(n, [ordering])	Devuelve los primeros <i>n</i> elementos del RDD usando su orden original o un comparador personalizado.	<pre>val b = sc.parallelize(List("dog", "cat", "ape", "salmon", "gnu"), 2) b.takeOrdered(2) res0: Array[String] = Array(ape, cat)</pre>
saveAsTextFile(path)	Guarda el RDD como un archivos de texto.	<pre>val a = sc.parallelize(1 to 10000, 3) a.saveAsTextFile("/home/usuario/datos") root@master:/home/usuario/datos # ls part-00000 part-00001 part-00002 _SUCCESS // Como se puede observar se han creado las 3 particiones, las cuales hemos especificado.</pre>

⁸ Array: medio de guardar un conjunto de elementos de la misma clase.

saveAsSequenceFile(path)	Guarda el RDD como un archivo de secuencia Hadoop.	<pre> val v = sc.parallelize(Array(("owl",3), ("gnu",4), ("dog",1), ("cat",2), ("ant",5)), 2) v.saveAsSequenceFile("/home/usuario/seq_datos") root@master:/home/usuario/seq_datos# ls -a . .. part-00000 .part-00000.crc part-00001 .part-00001.crc _SUCCESS _SUCCESS.crc </pre>
saveAsObjectFile(path) (Java and Scala)	Guarda los elementos del conjunto de datos en un formato simple utilizando la serialización de Java	<pre> val x = sc.parallelize(Array(("owl",3), ("gnu",4), ("dog",1), ("cat",2), ("ant",5)), 2) x.saveAsObjectFile("/home/usuario/objFile") root@master:/home/usuario/objFile# ls -a . .. part-00000 .part-00000.crc part-00001 .part-00001.crc _SUCCESS _SUCCESS.crc </pre>
countByKey()	Sólo disponible en RDD de tipo (K, V). Devuelve un hashmap de pares (K, Int) con el recuento de cada clave.	<pre> val c = sc.parallelize(List((3, "Gnu"), (3, "Yak"), (5, "Mouse"), (3, "Dog")), 2) c.countByKey res3: scala.collection.Map[Int,Long] = Map(3 -> 3, 5 -> 1) </pre>
foreach(func)	Ejecute una función func en cada elemento del dataset.	<pre> val c = sc.parallelize(List("cat", "dog", "tiger", "lion", "gnu", "crocodile", "ant", "whale", "dolphin", "spider"), 3) c.foreach(x => println(x + "s are beautiful")) cats are beautiful dogs are beautiful tigers are beautiful ants are beautiful whales are beautiful dolphins are beautiful spiders are beautiful lions are beautiful gnus are beautiful crocodiles are beautiful </pre>

NOTA: Todos los ejemplos están programados en Scala.

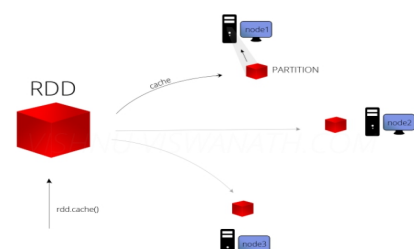
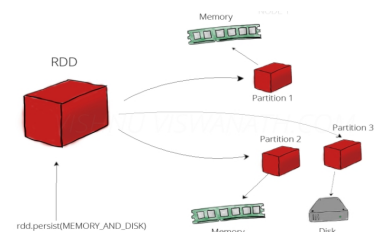
3.4 Persistencia de los RDD

La persistencia o cacheo de un dataset en memoria, es una de las características más importantes de Apache Spark, para poder realizar operaciones. Cuando se persiste un RDD, cada nodo almacena todas las particiones que posee en memoria para poder reutilizarlas al ejecutar otras acciones en dicho dataset. Gracias a esto, las futuras acciones se ejecutarán mucho más rápido (normalmente hasta 10 veces más veloz).

Para convertir un RDD en persistente, se debe usar los métodos *persist()* o *cache()*. Gracias a la caché Spark es tolerante a fallo, es decir, si se pierde alguna partición de un RDD, automáticamente se recalcula utilizando las transformaciones que lo crearon originalmente.

Por otra parte, mientras que el método *cache()* almacena los RDD en el nivel de almacenamiento por defecto (*storageLevel.MEMORY_ONLY*), el método *persist()* nos permite seleccionar el nivel de almacenamiento mediante un parámetro (*persist(StorageLevel.NIVEL)*).

Los niveles de almacenamientos que se pueden establecer como parámetro son:

Nivel de almacenamiento	Definición
MEMORY_ONLY	<p>Almacena los RDD en la memoria principal. Este es el nivel de almacenamiento que utiliza el método <i>cache()</i></p> 
MEMORY_AND_DISK	<p>Almacena los RDD en la memoria principal, pero si hubiese RDDs que no cupieran se guardarían en disco.</p> 
MEMORY_ONLY_SER (Java y Scala)	Almacena los RDD como objetos Java serializados.
MEMORY_AND_DISK_SER (Java y Scala)	Almacena los RDD como objetos Java serializados, pero si hubiese RDDs que no cupieran se guardarían en disco.
DISK_ONLY	Almacena los RDD en disco.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Similares a los niveles de almacenamiento mencionados anteriormente, pero duplicando las particiones en dos nodos.

NOTA: Para evitar almacenar datos innecesario, se puede llamar al método *unpersist()* con el que eliminaremos dichos datos en memoria y disco. Por otra parte, para saber si un RDD se encuentra cacheado podemos usar la propiedad booleana: *is_cached*.

4. SPARK VS HADOOP

En este apartado vamos a resolver la famosa pregunta: Spark y Hadoop son ¿Competencia o complemento?.

No es fácil hacer una comparación entre Hadoop y Spark, ya que ambos realizan cosas similares aunque poseen algunas áreas donde sus funcionalidades no se superponen. Por ejemplo, Spark no posee un sistema de archivos y, por ello, debe apoyarse en el sistema de archivos de Hadoop (HDFS⁹).

Vamos a comparar ambos frameworks:

4.1 Rendimiento

Como se ha mencionado anteriormente, Spark procesa los datos en memoria mientras que Hadoop en disco. Por lo que, no es fácil determinar quién consigue un mayor desempeño, ya que ambos procesan los datos de forma diferente.

En lo que respecta a Spark:

- Al trabajar en memoria se acelerará todos los procesos.
- Hace falta más memoria para el almacenamiento de datos.
- Su rendimiento puede verse afectado a causa de utilizar aplicaciones pesadas.

En el caso de Hadoop:

- Al trabajar en disco la velocidad de procesos resultará más lenta.
- Los recursos de almacenamiento son inferiores en comparación a Spark.
- Hadoop elimina los datos cuando no son necesarios, por lo que no se produce apenas pérdidas de rendimiento para aplicaciones pesadas.

4.2 Usabilidad

En este caso Apache Spark supera a Hadoop, ya que es conocido por su facilidad de uso gracias a su APIs para Scala, Python, Java y Spark SQL.

Además Spark posee un modo interactivo en el que se pueden recibir respuestas directas. Por otra parte, es cierto que MapReduce posee complementos como Pig y Hive que lo hacen un poco más sencillo de usar.

⁹ *HDFS*: Hadoop's Distributed File System

4.3 Costes

Ambos productos son de código abierto, pero aún así se necesita gastar dinero en hardware y personal. Spark y Hadoop han sido diseñado para poder funcionar correctamente en hardware básico como servidores de bajo costo.

Por otra parte, Spark necesita más memoria RAM ya que se necesita tanta memoria como cantidad de datos se necesiten procesar para poder tener un rendimiento óptimo. Por lo que si se necesitan procesar gran cantidad de datos, Hadoop sería la opción más barata.

Haciendo referencia a los benchmarks¹⁰, Spark es más rentable que Hadoop debido a que con menos hardware se pueden realizar las mismas tareas de forma más veloz. De hecho, Spark a conseguido procesar 100 TB de datos, tres veces más rápido que MapReduce con una décima parte de la cantidad de máquinas requeridas. Gracias a esta hazaña, Spark consiguió ganar la edición de [*2014 Daytona GraySort Benchmark*](#).

4.4 Compatibilidad

Apache Spark y MapReduce son compatibles entre sí. Además Spark posee todas las compatibilidades de MapReduce para fuentes de datos, formatos de archivos o incluso para herramientas de BI (Business Intelligence), a través de JDBC (Java Database Connectivity) y ODBC (Open Database Connectivity) .

4.5 Procesamiento de datos

Apache Spark es capaz de realizar más operaciones a parte del procesamiento de datos: es capaz de procesar gráficos, utilizar las bibliotecas de aprendizaje de máquinas existentes e incluso procesar datos en tiempo real y por lotes. Mientras que podemos tener todas estas características en una sola plataforma, Hadoop MapReduce necesita otras plataformas como Storm o Impala para procesamiento en streaming, Giraph para procesamiento de gráficos y Apache Mahout para el aprendizaje de máquinas. Por otro lado, Hadoop MapReduce es ideal para el procesamiento de datos por lotes.

4.6 Tolerancia a fallos

Debido a que MapReduce se basa en el procesamiento de datos en disco, si se bloquea un proceso en medio de su ejecución se podría continuar donde se dejó. Mientras que en Apache Spark se tendría que comenzar a procesar desde el principio.

Por lo que, Hadoop MapReduce es un poco más tolerable a fallos que Spark.

¹⁰ Técnica utilizada para medir el rendimiento de un sistema o componente del mismo.

4.7 Seguridad

En este caso Hadoop vence a Spark, ya que:

- Proporciona todos los beneficios obtenidos en los proyectos de seguridad de Hadoop.
- Hadoop posee el servicio *level Authorization* , con el que se asegura que los clientes tengan los permisos adecuados.
- Hadoop soporta autenticación Kerberos¹¹.
- Posee Hadoop YARN

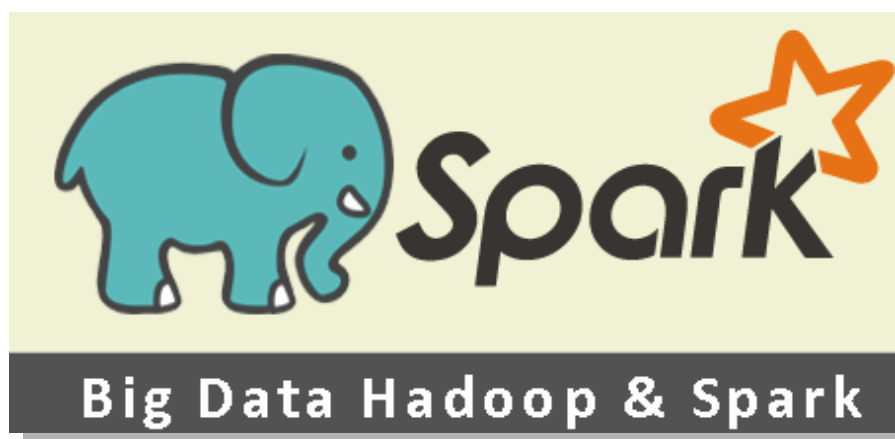
Por otro lado, Spark necesita Hadoop YARN para obtener beneficios de seguridad, como el uso de autenticación kerberos, y ejecutarse en HDFS para acceder a permisos de nivel de archivo.

En pocas palabras, la seguridad en Spark todavía está en su infancia, mientras que Hadoop MapReduce tiene más características y proyectos de seguridad.

4.8 Conclusiones

Como se ha podido observar cada uno domina al otro en distintas áreas. Por ejemplo Hadoop ofrece elementos que no posee Spark, como sistemas de archivos distribuidos. Por otra parte, Spark ofrece procesamiento en memoria en tiempo real para datos que lo requieran.

En conclusión, el escenario ideal de Big Data es el de Hadoop y Spark trabajando en un mismo equipo.



¹¹ Es un protocolo de autenticación que permite a dos ordenadores en una red insegura demostrar su identidad mutuamente de forma segura.

5. TOPOLOGÍA DE RED

Para la creación del clúster¹² de servidores con Apache Spark, se han utilizado cinco máquinas virtuales. Una de las cinco máquinas será el servidor principal (Spark-Master) y el resto servidores esclavos (Spark-WorkerN). Las máquinas utilizadas son:

Hostname	IP (10.0.0.0/24)	Función
master	10.0.0.1	Servidor principal
worker1	10.0.0.2	Servidor esclavo
worker2	10.0.0.3	Servidor esclavo
worker3	10.0.0.4	Servidor esclavo

Para configurar la red de la máquinas editamos el fichero “*/etc/network/interfaces*”:

```
# Configuración para la red interna
auto eth0
iface eth0 inet static
    address 10.0.0.X
    netmask 255.255.255.0
```

¹² Un *clúster* consiste en la unión de varios servidor. los cuales trabajarán de forma paralela como si de un solo servidor se tratase.

6. CARACTERÍSTICAS Y REQUISITOS

Las máquinas virtuales utilizadas tendrán Debian 8.6 (Jessie) como sistema operativo y estarán virtualizadas con VirtualBox. Las principales características de Hardware son:

----- MASTER -----

- **Nº de cores:** 2
- **Memoria RAM:** 2048 Mb
- **Discos duros:** Disco duro virtual reservado dinámicamente
- **Tarjeta de red:** Dos adaptadores de red (Un adaptador puente a wlan0 y un adaptador para la red interna).

----- WORKERS -----

- **Nº de cores:** 1
- **Memoria RAM:** 1024 Mb
- **Discos duros:** Disco duro virtual reservado dinámicamente
- **Tarjeta de red:** Dos adaptadores de red (Un adaptador puente a wlan0 y un adaptador para la red interna).

Para el funcionamiento correcto del clúster de servidores, Apache Spark necesita la siguiente lista de software instalada:

- **Java:** vamos a instalar la versión 8.
- **Scala:** vamos a instalar la versión 2.11.8
- **OpenSSH:** Para permitir el acceso remoto por SSH entre las máquinas.
- **Apache Spark:** Spark pre-compilado para Hadoop.

7. INSTALACIÓN Y CONFIGURACIÓN DE SPARK EN DEBIAN

Vamos a instalar una serie de herramientas necesarias en todos los servidores a no ser que se indique lo contrario.

7.1 Instalación de Java

Añadimos los repositorios correspondientes:

```
root@master:/# echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" > /etc/apt/sources.list.d/webupd8team-java.list
root@master:/# echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" >> /etc/apt/sources.list.d/webupd8team-java.list
```

Importamos la siguiente clave para poder instalar JAVA 8:

```
root@master:/# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
```

Actualizamos los repositorios:

```
root@master:/# apt-get update
```

Posteriormente instalamos JAVA 8 gracias a la herramienta *apt-get*:

```
root@master:/# apt-get install oracle-java8-installer
```

Comprobamos la versión de nuestro JAVA:

```
root@master:/# java -version

java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

7.2 Instalación de Scala

Descargamos la última versión del paquete de instalación de scala, en nuestro caso [scala-2.13.0-M1.deb](#):

```
root@master:/# wget https://downloads.lightbend.com/scala/2.13.0-M1/scala-2.13.0-M1.deb
```

Posteriormente, con la herramienta dpkg, instalamos el archivo descargado anteriormente:

```
root@master:/# dpkg -i scala-2.13.0-M1.deb
```

Comprobamos que se ha instalado correctamente scala y que se ejecuta su shell interactiva correctamente:

```
root@master:/# scala -version
```

```
Scala code runner version 2.13.0-M1 -- Copyright 2002-2017, LAMP/EPFL and Lightbend, Inc.
```

```
root@master:/# scala
```

```
root@master:/home/usuario# scala
Welcome to Scala 2.13.0-M1 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121).
Type in expressions for evaluation. Or try :help.

scala> object HelloWorld {
  |   def main(args: Array[String]) {
  |     println("Hello, world!")
  |   }
  | }
defined object HelloWorld

scala> HelloWorld.main(null)
Hello, world!

scala> :q
```

7.3 Instalación y configuración de OpenSSH

Apache Spark necesita acceso remoto sin pedir contraseña, para ello vamos a hacer uso de la herramienta SSH. En la máquina *master* instalamos el paquete openssh-server:

```
root@master:/# apt-get install openssh-server
```

En las máquinas workers permitimos el acceso remoto al usuario root:

```
root@master:/# nano /etc/ssh/sshd_config
PermitRootLogin yes
```

Posteriormente reiniciamos el servicio SSH:

```
root@master:/# service ssh restart
```

7.4 Instalación y configuración de Apache Spark

Descargamos e instalamos la última versión de Apache Spark de [ESTE](http://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-hadoop2.7.tgz) enlace:

```
root@master:/# wget http://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-hadoop2.7.tgz
```

Creamos el directorio `/opt/spark`:

```
root@master:/# mkdir /opt/spark
```

Descomprimos el fichero descargado anteriormente:

```
root@master:/# tar -xzf spark-2.1.0-bin-hadoop2.7.tgz
```

Copiamos los ficheros de la carpeta descomprimida anteriormente a la nueva ubicación:

```
root@master:/# mv spark-2.1.0-bin-hadoop2.7/* /opt/spark/
```

Añadimos una nueva entrada en el fichero `.bashrc`:

```
root@master:/# nano ~/.bashrc

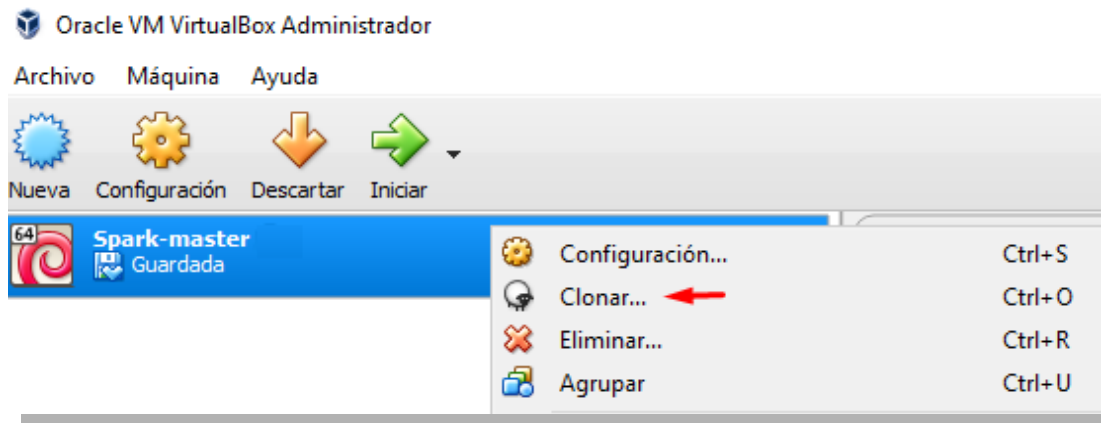
export SPARK_HOME=/opt/spark/
export PATH="/opt/spark/bin:/opt/spark/sbin:.$PATH"
```

Cargamos las variables definidas anteriormente en el fichero `.bashrc`:

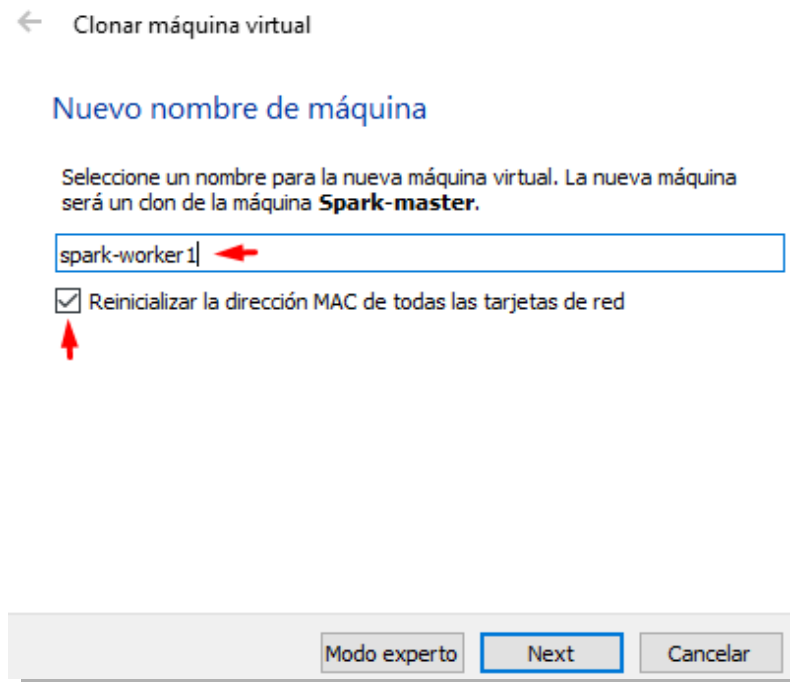
```
root@master:/# source ~/.bashrc
```

7.5. Creación de los servidores esclavos

Para no tener que realizar todas las acciones anteriores, se va a clonar el servidor master. Para ello, damos click derecho en la máquina *Spark-master* y pinchamos en el apartado *Clonar...*:

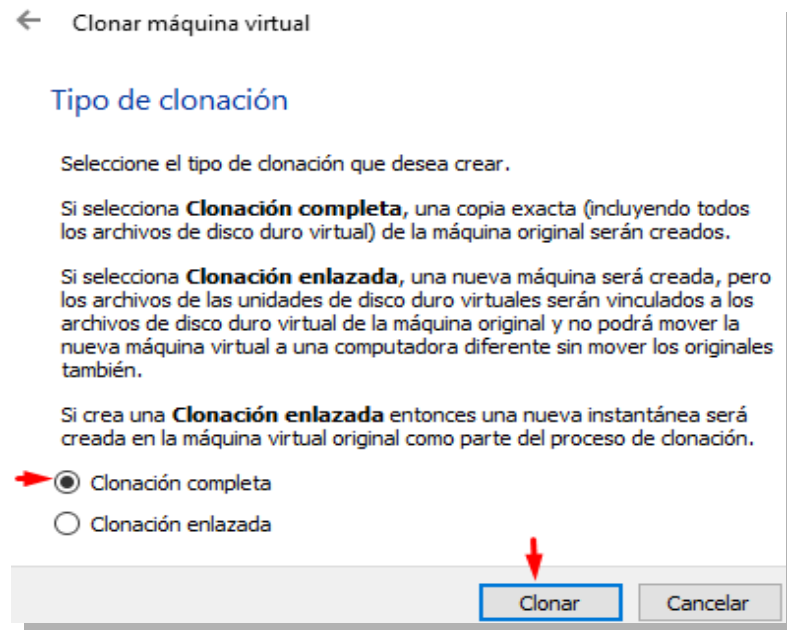


Posteriormente le asignamos un nombre a la máquina, en nuestro caso *spark-worker1*.



Muy Importante !!! : Recuerda reiniciar la MAC de las máquinas clonadas como se observa en la anterior ilustración, para no tener conflictos con las configuración de red.

Seleccionamos el tipo de clonación *completa*, para realizar una copia exacta de las máquinas:



Posteriormente accedemos a la nueva máquina clonada y modificamos el fichero `/etc/hostname` para cambiar el nombre del equipo:

```
root@master:/# nano /etc/hostname  
[NUEVO_HOSTNAME]
```

También se debe de alterar el fichero `/etc/hosts` y realizar el siguiente cambio:

```
root@[NUEVO_HOSTNAME]:/# nano /etc/hosts  
127.0.1.1 [NUEVO_HOSTNAME]
```

Por último vamos a cambiar la IP interna de la máquina clonada, para ello editamos el fichero `/etc/network/interfaces`:

```
root@[NUEVO_HOSTNAME]:/# nano /etc/network/interfaces  
  
# Configuración para la red interna  
auto eth0  
iface eth0 inet static  
    address [IP_MAQUINA]  
    netmask 255.255.255.0
```

Por último reiniciamos la máquina para que se realicen todos los cambios ejecutados anteriormente.

7.6. Configuración de OpenSSH en master

Posteriormente generamos una clave RSA para permitir el acceso remoto de los servidores *workers*:

```
root@master:/# ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
0c:98:6e:f6:1a:32:0e:d4:c9:b6:f8:4a:ea:58:53:e6 root@master
The key's randomart image is:
+---[RSA 2048]-----+
|
|  o
|  o.
| o.. o
| . =* S
| . o*..
|.o=.E.
|+++.+ o
|+oo..
+-----+
```

Copiamos la clave RSA generada anteriormente en los servidores *workers*.

```
root@master:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.2
root@master:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.3
root@master:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.4
```

Adjunto la salida de la última instrucción ejecutada:

```
root@master:/# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.4

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are
already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install
the new keys
root@10.0.0.4's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'root@10.0.0.4'"
and check to make sure that only the key(s) you wanted were added.
```

7.7. Configuración de Apache Spark en el servidor master

Vamos a indicar quién es el nodo master y quiénes son los esclavos. Para ello nos dirigimos a la carpeta *conf* y renombramos el fichero denominado *slaves.template* a *slaves*. Este fichero contiene las direcciones IP de los nodos esclavos:

```
root@master:/opt/spark/conf# cp slaves.template slaves
root@master:/opt/spark/conf# nano slaves

# A Spark Worker will be started on each of the machines listed below.
# localhost
10.0.0.2
10.0.0.3
10.0.0.4
```

Posteriormente en la carpeta *conf*, renombramos el fichero denominado *spark-env.sh.template* a *spark-env.sh*:

```
root@master:/opt/spark/conf# cp spark-env.sh.template spark-env.sh
root@master:/opt/spark/conf# nano spark-env.sh

export SPARK_MASTER_HOST=10.0.0.1 # En mi caso: 10.0.0.1.
export SPARK_WORKER_CORES=1 # Número de cores que se ejecutan en el servidor.
export SPARK_WORKER_INSTANCES=1 # Número de procesos worker por cada nodo.
```

Para comprobamos que Apache Spark funciona correctamente, ejecutamos el siguiente comando para calcular el número Pi con Spark:

```
root@master:/opt/spark/sbin# run-example SparkPi

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
17/04/14 19:40:26 INFO SparkContext: Running Spark version 2.1.0
17/04/14 19:40:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
17/04/14 19:40:27 WARN Utils: Your hostname, master resolves to a loopback address:
127.0.1.1; using 192.168.1.136 instead (on interface eth0)

[...]

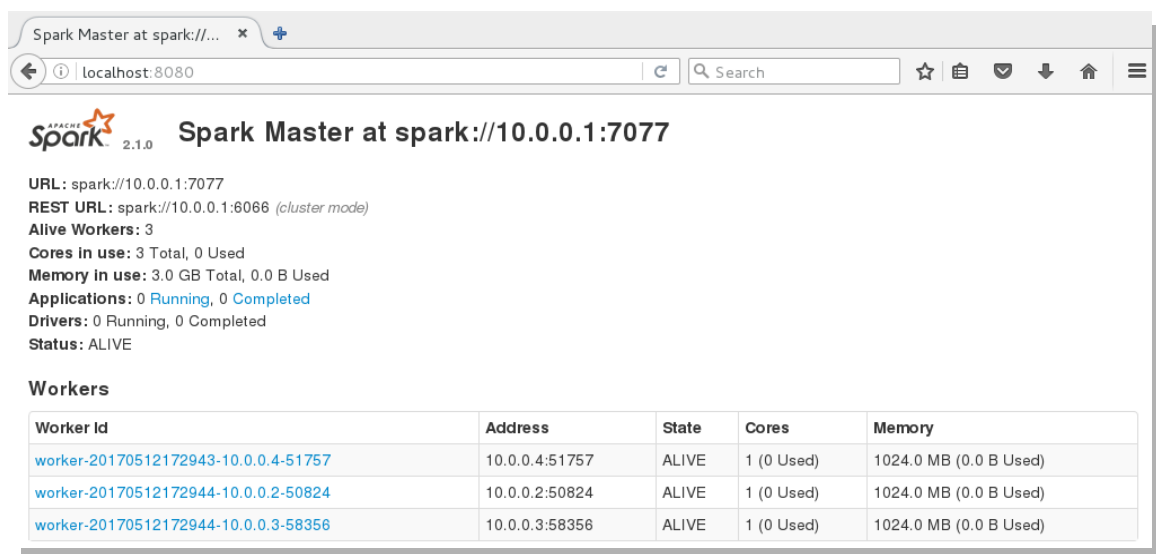
Pi is roughly 3.147915739578698
```

Para comprobar que el clúster funciona correctamente, nos dirigimos a la ubicación `/opt/spark/sbin` y ejecutamos el script denominado `start-all.sh`:

```
root@master:/opt/spark/sbin# start-all.sh
```

```
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-root-
org.apache.spark.deploy.master.Master-1-master.out
10.0.0.3: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-
org.apache.spark.deploy.worker.Worker-1-worker2.out
10.0.0.2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-
org.apache.spark.deploy.worker.Worker-1-worker1.out
10.0.0.4: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-
org.apache.spark.deploy.worker.Worker-1-worker3.out
```

Posteriormente accedemos al panel web:



Spark Master at spark://10.0.0.1:7077

URL: spark://10.0.0.1:7077
REST URL: spark://10.0.0.1:6066 (cluster mode)
Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170512172943-10.0.0.4-51757	10.0.0.4:51757	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20170512172944-10.0.0.2-50824	10.0.0.2:50824	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20170512172944-10.0.0.3-58356	10.0.0.3:58356	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Probamos la ejecución de la función `SparkPi` en el clúster:

```
root@master:/# MASTER=spark://10.0.0.1:7077 run-example SparkPi
```

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
17/04/24 17:26:13 INFO SparkContext: Running Spark version 2.1.0
17/04/24 17:26:14 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
where applicable
17/04/24 17:26:14 WARN SparkConf:
SPARK_WORKER_INSTANCES was detected (set to '1').
This is deprecated in Spark 1.0+.
```

Please instead use:

- `./spark-submit` with `--num-executors` to specify the number of executors
- Or set `SPARK_EXECUTOR_INSTANCES`
- `spark.executor.instances` to configure the number of instances in the spark config.

```
17/04/24 17:26:14 WARN Utils: Your hostname, master resolves to a loopback address: 127.0.1.1; using 10.0.0.1 instead (on
interface eth0)
```

```
17/04/24 17:26:14 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
```

17/04/24 17:26:14 INFO SecurityManager: Changing view acls to: root
17/04/24 17:26:14 INFO SecurityManager: Changing modify acls to: root
17/04/24 17:26:14 INFO SecurityManager: Changing view acls groups to:
17/04/24 17:26:14 INFO SecurityManager: Changing modify acls groups to:
17/04/24 17:26:14 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups with view permissions: Set(); users with modify permissions: Set(root); groups with modify permissions: Set()
17/04/24 17:26:14 INFO Utils: Successfully started service 'sparkDriver' on port 59652.
17/04/24 17:26:14 INFO SparkEnv: Registering MapOutputTracker
17/04/24 17:26:14 INFO SparkEnv: Registering BlockManagerMaster
17/04/24 17:26:14 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
17/04/24 17:26:14 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
17/04/24 17:26:14 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-4f2fbaad-dbeb-4193-b3a0-6a12b0106aa2
17/04/24 17:26:14 INFO MemoryStore: MemoryStore started with capacity 413.9 MB
17/04/24 17:26:15 INFO SparkEnv: Registering OutputCommitCoordinator
17/04/24 17:26:15 INFO Utils: Successfully started service 'SparkUI' on port 4040.
17/04/24 17:26:15 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://10.0.0.1:4040
17/04/24 17:26:15 INFO SparkContext: Added JAR file:/opt/spark/examples/jars/scopt_2.11-3.3.0.jar at spark://10.0.0.1:59652/jars/scopt_2.11-3.3.0.jar with timestamp 1493047575473
17/04/24 17:26:15 INFO SparkContext: Added JAR file:/opt/spark/examples/jars/spark-examples_2.11-2.1.0.jar at spark://10.0.0.1:59652/jars/spark-examples_2.11-2.1.0.jar with timestamp 1493047575474
17/04/24 17:26:15 INFO StandaloneAppClient\$ClientEndpoint: Connecting to master spark://10.0.0.1:7077...
17/04/24 17:26:15 INFO TransportClientFactory: Successfully created connection to /10.0.0.1:7077 after 36 ms (0 ms spent in bootstraps)
17/04/24 17:26:15 INFO StandaloneSchedulerBackend: Connected to Spark cluster with app ID app-20170424172615-0000
17/04/24 17:26:15 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 38322.
17/04/24 17:26:15 INFO NettyBlockTransferService: Server created on 10.0.0.1:38322
17/04/24 17:26:15 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
17/04/24 17:26:15 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 10.0.0.1, 38322, None)
17/04/24 17:26:15 INFO BlockManagerMasterEndpoint: Registering block manager 10.0.0.1:38322 with 413.9 MB RAM, BlockManagerId(driver, 10.0.0.1, 38322, None)
17/04/24 17:26:15 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 10.0.0.1, 38322, None)
17/04/24 17:26:15 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 10.0.0.1, 38322, None)
17/04/24 17:26:15 INFO StandaloneAppClient\$ClientEndpoint: Executor added: app-20170424172615-0000/0 on worker-20170424164646-10.0.0.3-49533 (10.0.0.3:49533) with 1 cores
17/04/24 17:26:16 INFO StandaloneSchedulerBackend: Granted executor ID app-20170424172615-0000/0 on hostPort 10.0.0.3:49533 with 1 cores, 1024.0 MB RAM
17/04/24 17:26:16 INFO StandaloneAppClient\$ClientEndpoint: Executor added: app-20170424172615-0000/1 on worker-20170424164714-10.0.0.2-48513 (10.0.0.2:48513) with 1 cores
17/04/24 17:26:16 INFO StandaloneSchedulerBackend: Granted executor ID app-20170424172615-0000/1 on hostPort 10.0.0.2:48513 with 1 cores, 1024.0 MB RAM
17/04/24 17:26:16 INFO StandaloneAppClient\$ClientEndpoint: Executor added: app-20170424172615-0000/2 on worker-20170424164655-10.0.0.4-54490 (10.0.0.4:54490) with 1 cores
17/04/24 17:26:16 INFO StandaloneSchedulerBackend: Granted executor ID app-20170424172615-0000/2 on hostPort 10.0.0.4:54490 with 1 cores, 1024.0 MB RAM
17/04/24 17:26:16 INFO StandaloneAppClient\$ClientEndpoint: Executor updated: app-20170424172615-0000/0 is now **RUNNING**
17/04/24 17:26:16 INFO StandaloneAppClient\$ClientEndpoint: Executor updated: app-20170424172615-0000/1 is now **RUNNING**
17/04/24 17:26:16 INFO StandaloneAppClient\$ClientEndpoint: Executor updated: app-20170424172615-0000/2 is now **RUNNING**
17/04/24 17:26:16 INFO StandaloneSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
17/04/24 17:26:16 INFO SharedState: Warehouse path is 'file:/home/usuario/spark-warehouse'.
17/04/24 17:26:17 INFO SparkContext: Starting job: reduce at SparkPi.scala:38
17/04/24 17:26:17 INFO DAGScheduler: Got job 0 (reduce at SparkPi.scala:38) with 2 output partitions
17/04/24 17:26:17 INFO DAGScheduler: Final stage: ResultStage 0 (reduce at SparkPi.scala:38)
17/04/24 17:26:17 INFO DAGScheduler: Parents of final stage: List()
17/04/24 17:26:17 INFO DAGScheduler: Missing parents: List()
17/04/24 17:26:17 INFO DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:34), which has no missing parents
17/04/24 17:26:18 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 1832.0 B, free 413.9 MB)
17/04/24 17:26:18 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 10.0.0.1:38322 (size: 1172.0 B, free: 413.9 MB)
17/04/24 17:26:18 INFO TaskSchedulerImpl: Adding task set 0.0 with 2 tasks

```

17/04/24 17:26:19 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(null)
(10.0.0.2:58323) with ID 1
17/04/24 17:26:19 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, 10.0.0.2, executor 1, partition 0,
PROCESS_LOCAL, 6082 bytes)
17/04/24 17:26:19 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(null)
(10.0.0.4:38009) with ID 2
17/04/24 17:26:19 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 10.0.0.4, executor 2, partition 1,
PROCESS_LOCAL, 6082 bytes)
17/04/24 17:26:19 INFO BlockManagerMasterEndpoint: Registering block manager 10.0.0.2:48312 with 413.9 MB RAM,
BlockManagerId(1, 10.0.0.2, 48312, None)
17/04/24 17:26:19 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(null)
(10.0.0.3:42850) with ID 0
17/04/24 17:26:19 INFO BlockManagerMasterEndpoint: Registering block manager 10.0.0.4:42877 with 413.9 MB RAM,
BlockManagerId(2, 10.0.0.4, 42877, None)
17/04/24 17:26:20 INFO BlockManagerMasterEndpoint: Registering block manager 10.0.0.3:35754 with 413.9 MB RAM,
BlockManagerId(0, 10.0.0.3, 35754, None)
17/04/24 17:26:20 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 10.0.0.2:48312 (size: 1172.0 B, free: 413.9
MB)
17/04/24 17:26:20 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on 10.0.0.4:42877 (size: 1172.0 B, free: 413.9
MB)
17/04/24 17:26:20 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1229 ms on 10.0.0.2 (executor 1) (1/2)
17/04/24 17:26:21 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 2,977 s
17/04/24 17:26:21 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 1439 ms on 10.0.0.4 (executor 2) (2/2)
17/04/24 17:26:21 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
17/04/24 17:26:21 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 3,660659 s
Pi is roughly 3.1399756998784993


```

[...]

Como se puede observar en la siguiente ilustración se ha completado correctamente la aplicación:

Spark Master at spark://...

localhost:8080



Spark Master at spark://10.0.0.1:7077

URL: spark://10.0.0.1:7077

REST URL: spark://10.0.0.1:8086 (cluster mode)

Alive Workers: 3

Cores in use: 3 Total, 0 Used

Memory in use: 3.0 GB Total, 0.0 B Used

Applications: 0 Running, 1 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170512172943-10.0.0.4-51757	10.0.0.4:51757	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20170512172944-10.0.0.2-50824	10.0.0.2:50824	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20170512172944-10.0.0.3-58356	10.0.0.3:58356	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20170512173232-0000	Spark Pi	3	1024.0 MB	2017/05/12 17:32:32	root	FINISHED	7 s

8. INSTALACIÓN Y CONFIGURACIÓN DE SPARK EN WINDOWS 10

8.1 Instalación de Java SE Development Kit 8

Descargamos JAVA JDK desde la [página oficial](#):

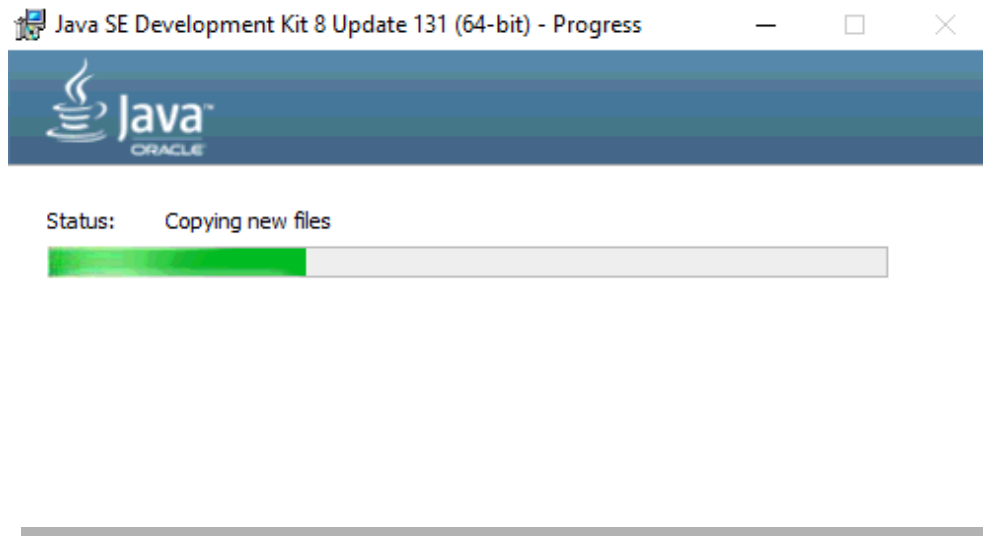
The screenshot shows the Oracle Technology Network website for Java SE Downloads. The main heading is "Java SE Development Kit 8 Downloads". Below it, there is a table of download links for various operating systems and architectures. A red arrow points to the "JDK 8u131 checksum" link. Another red arrow points to the "JDK 8u131-windows-x64.exe" download link in the table.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

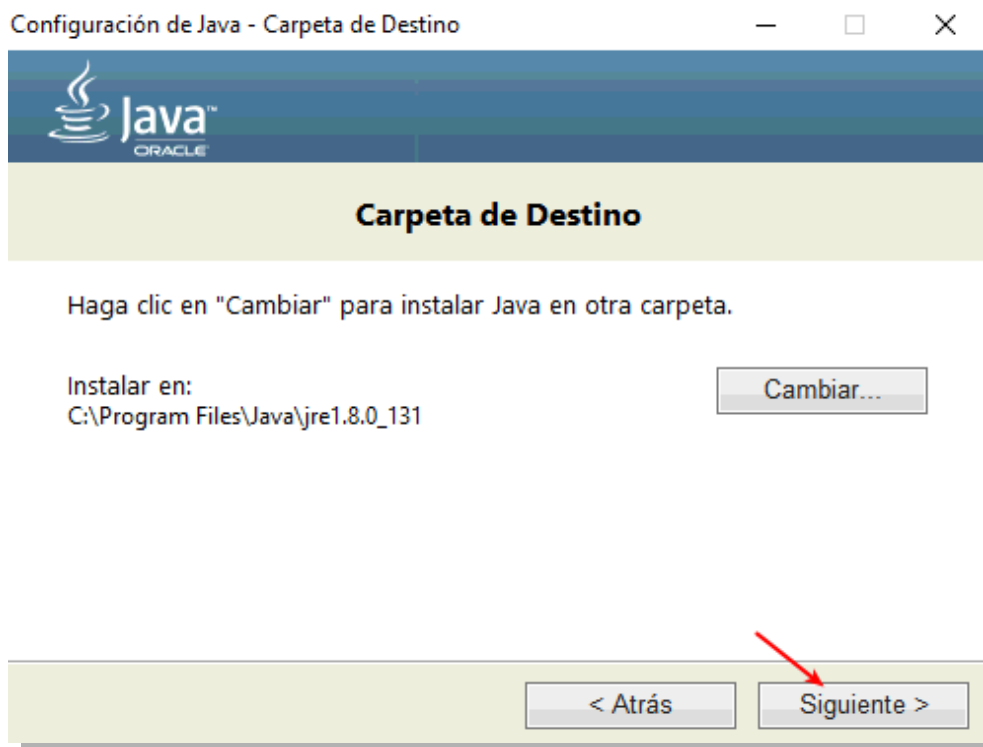
Posteriormente seleccionamos la casilla *next* >:

The screenshot shows the "Java SE Development Kit 8 Update 131 (64-bit) - Custom Setup" window. It displays a list of optional features to install: "Development Tools", "Source Code", and "Public JRE". The "Development Tools" feature is selected. Below the list, the installation path is shown as "C:\Program Files\Java\jdk1.8.0_131\". At the bottom, there are three buttons: "< Back", "Next >", and "Cancel". A red arrow points to the "Next >" button.

Como se puede observar en la siguiente ilustración la instalación está en proceso:



Más tarde, aparecerá otra ventana damos click en el apartado *siguiente*:

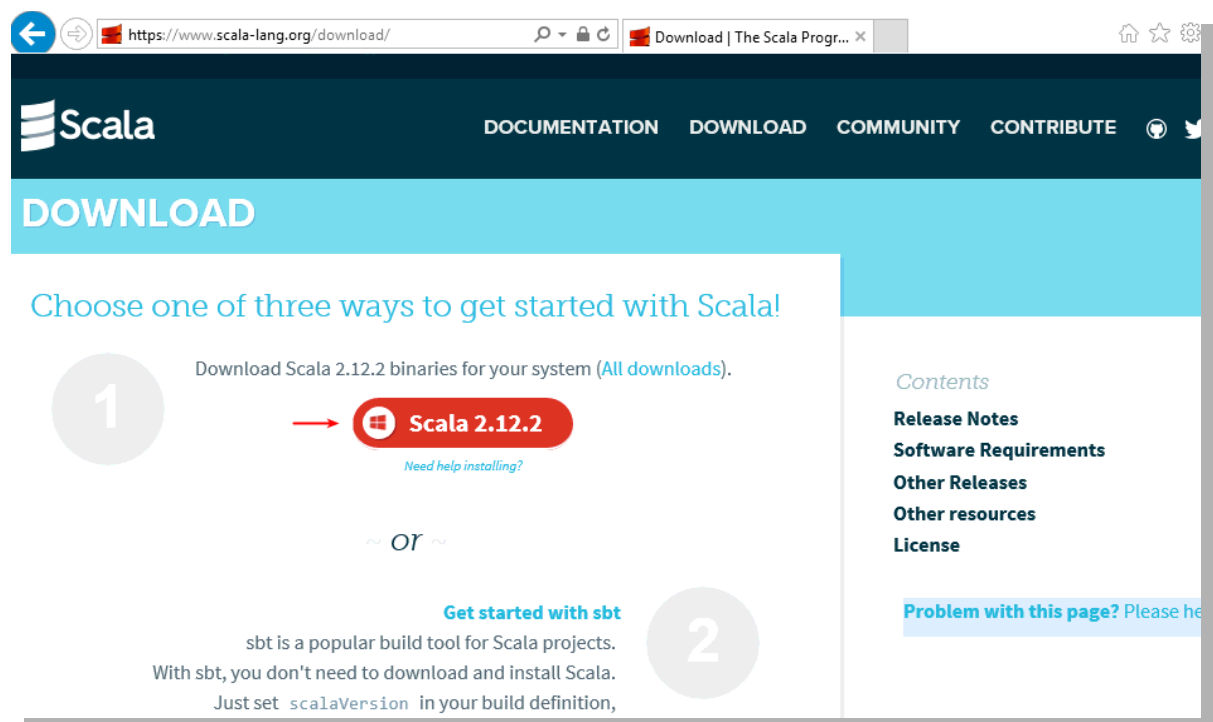


Una vez se haya instalado todo correctamente seleccionamos la casilla *close*:

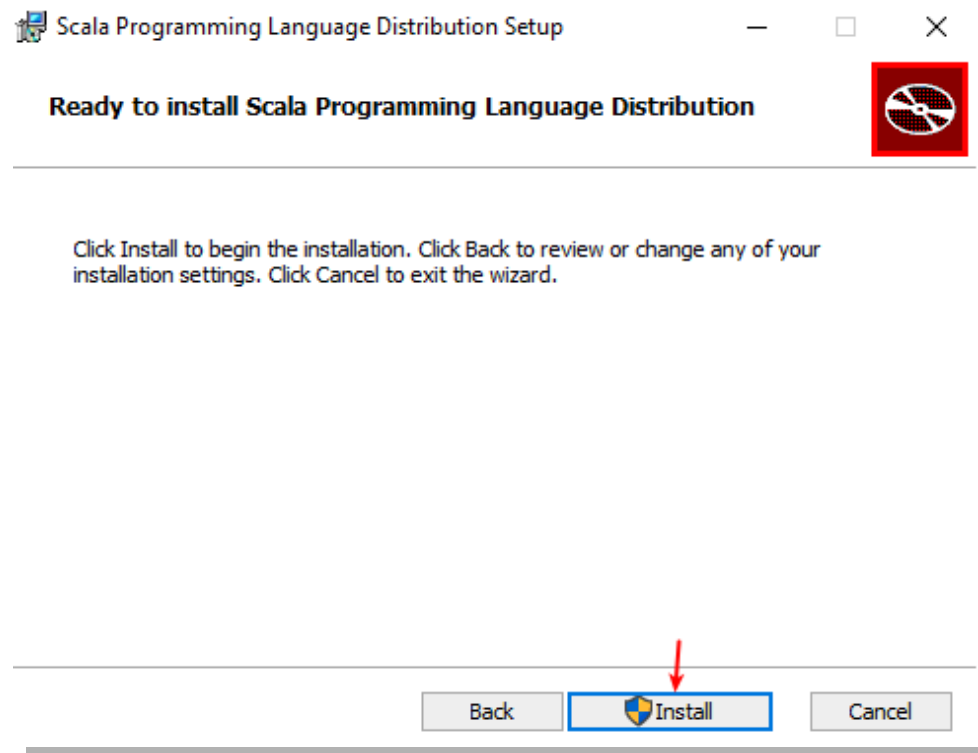


8.2 Instalación de Scala

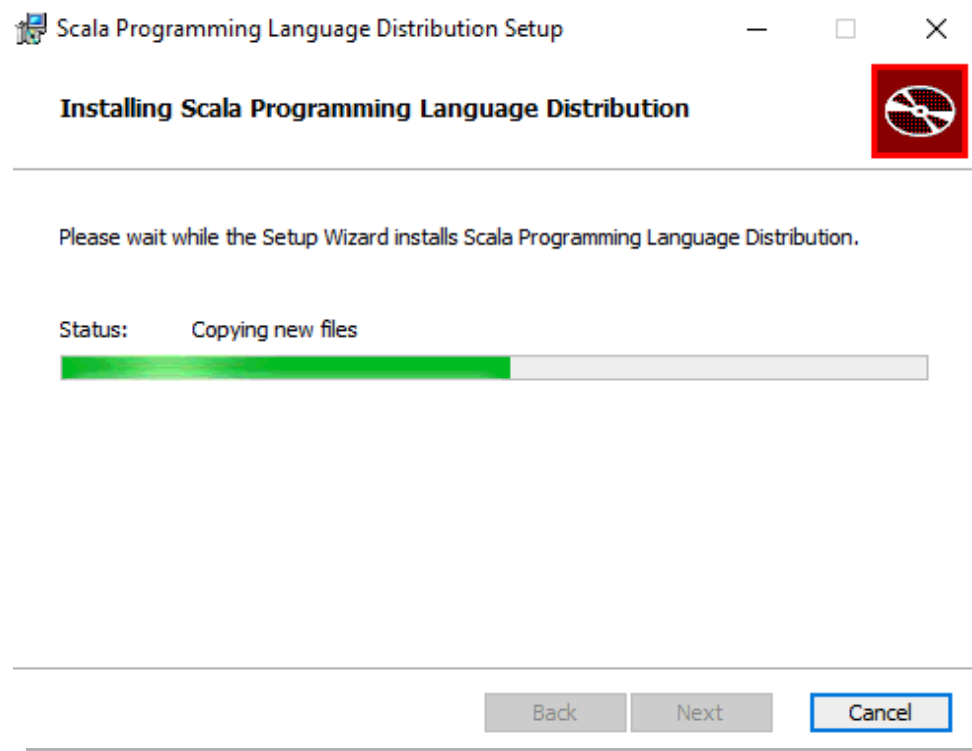
Descargamos Scala desde su [página oficial](https://www.scala-lang.org/download/):



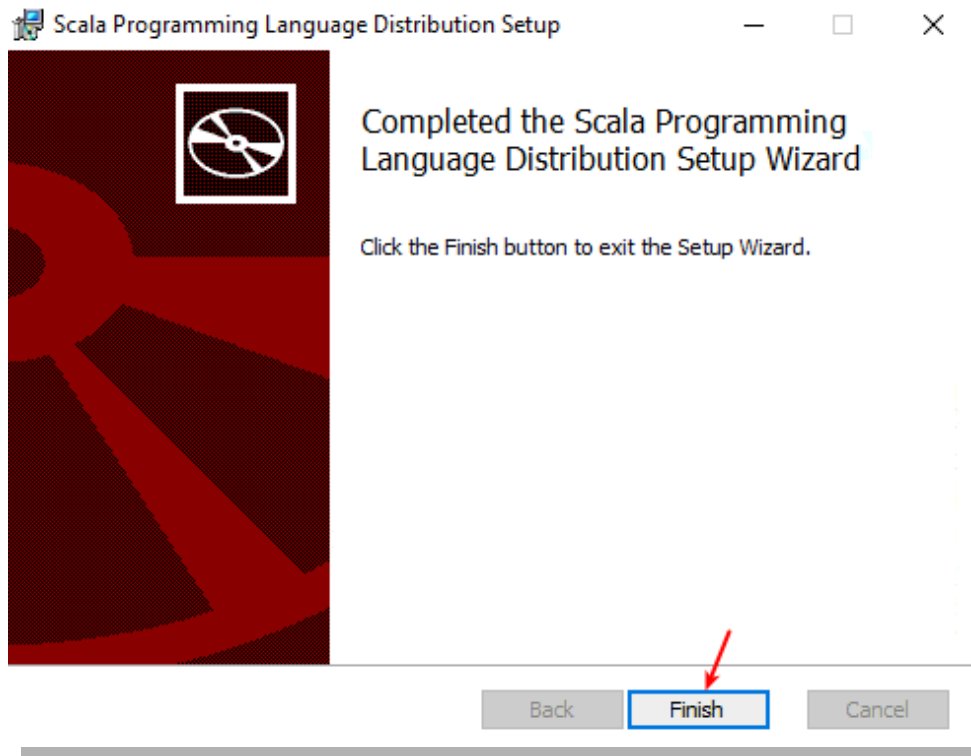
Posteriormente seleccionamos el apartado *Install*, el cual nos pedirá permisos de administrador:



Como se puede observar en la siguiente ilustración la instalación está en proceso:



Una vez se haya instalado Scala damos click en el apartado *Finish*:

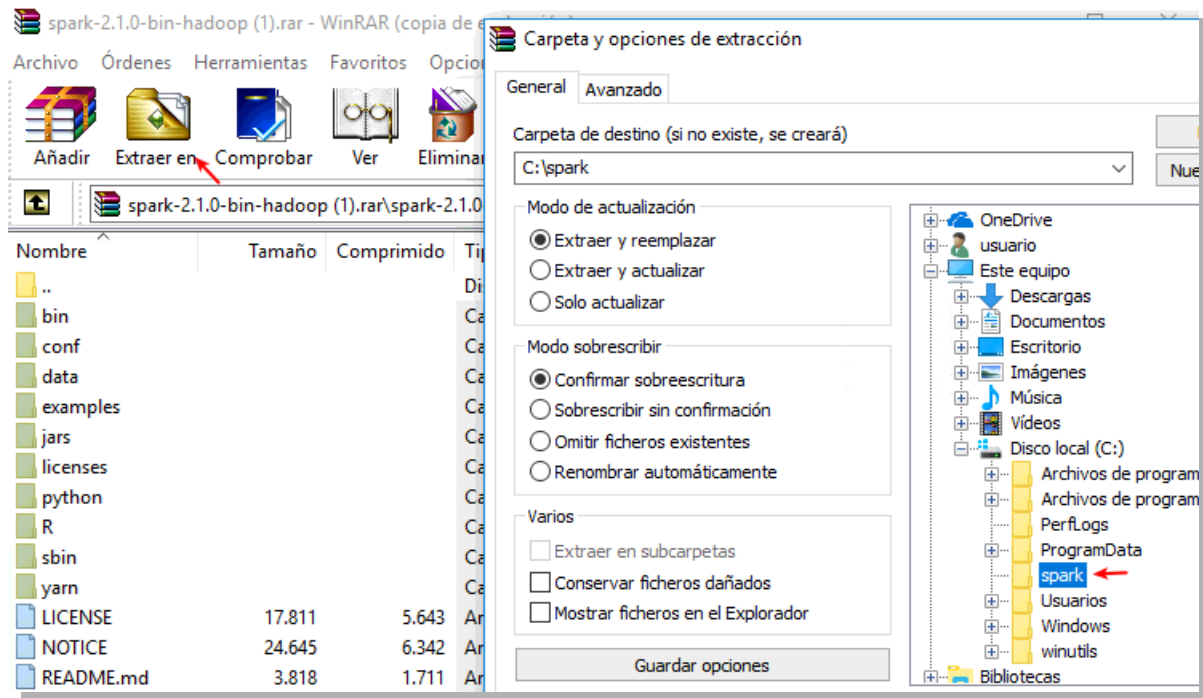


8.3 Instalación de Spark

Descargamos Apache Spark desde su [página oficial](https://spark.apache.org/downloads.html):

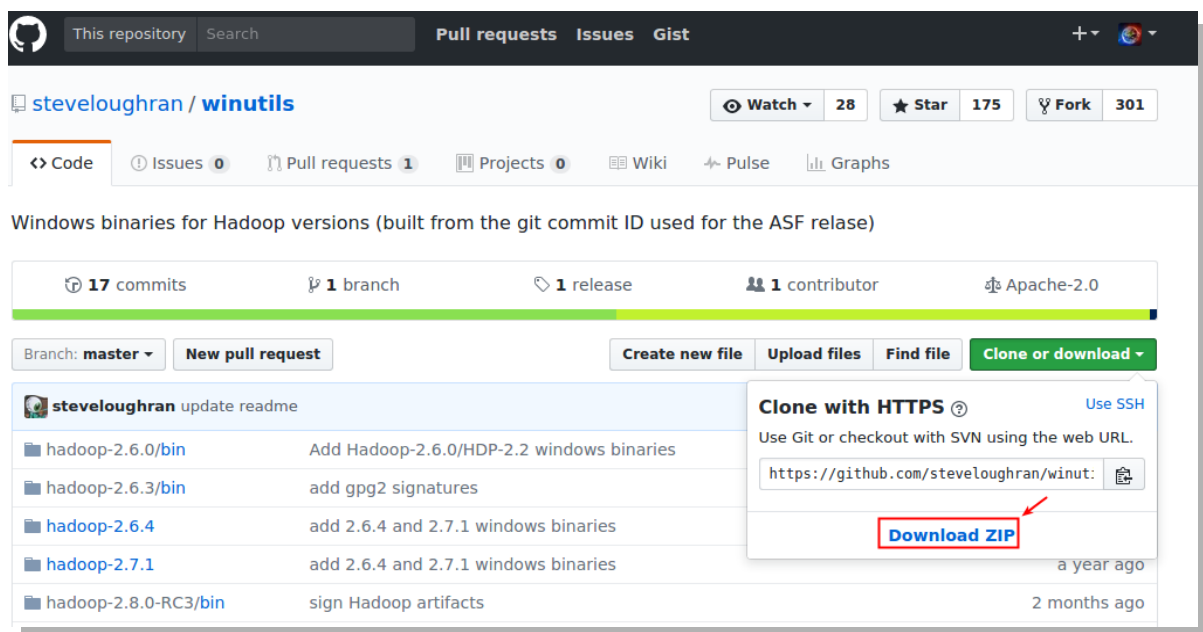


Posteriormente vamos a descomprimir el contenido de la carpeta *spark-2.1.0-bin-hadoop2.7* del fichero descargado en una carpeta previamente creada. En mi caso, el contenido se descomprime en una carpeta llamada *spark* ubicada en la raíz del disco C: :

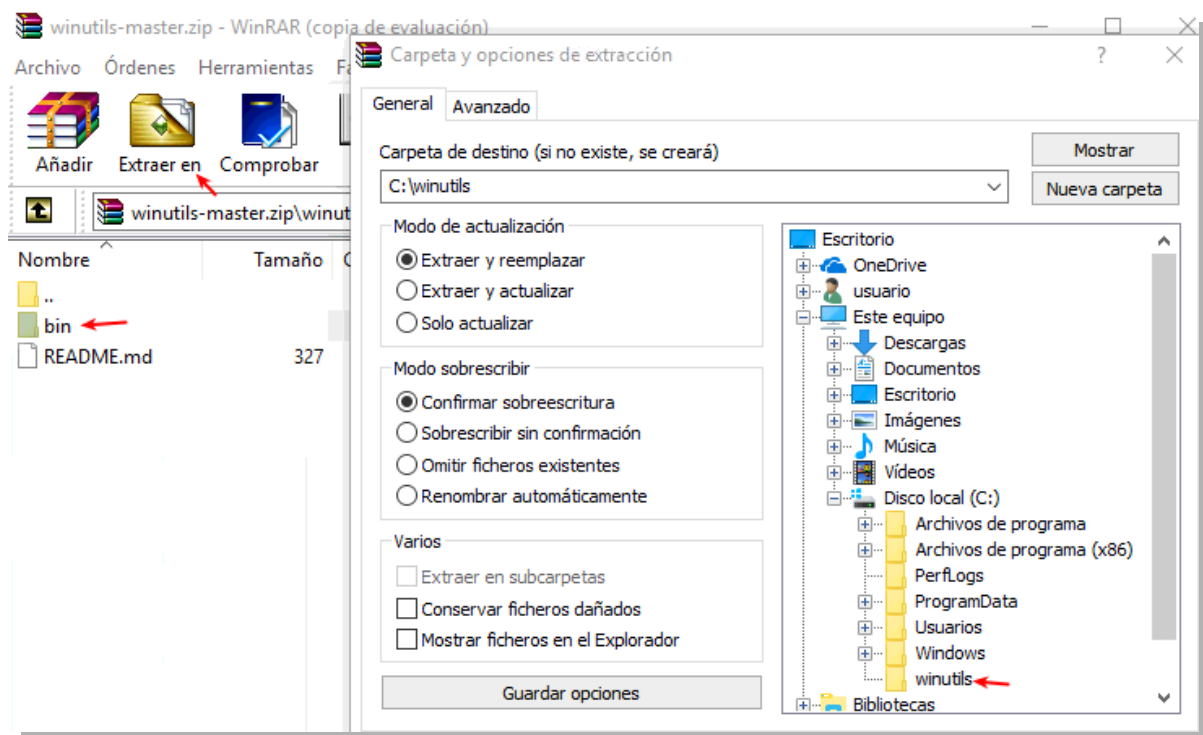


8.4 Instalación de Winutils y establecimiento de variables

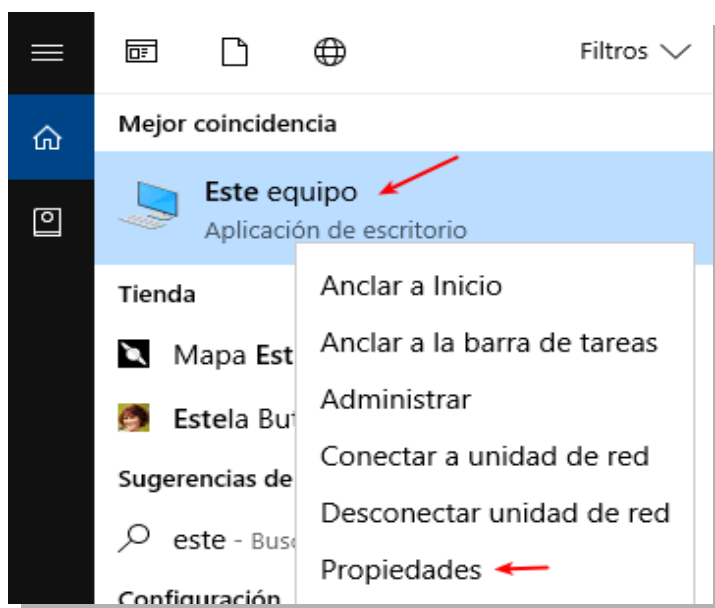
Para evitar el error: *Failed to locate the winutils binary in the hadoop binary path*, al ejecutar spark-shell es necesario la instalación de *Winutils*. Para ello vamos a necesitar descargar [ESTE](#) repositorio alojado en Github:



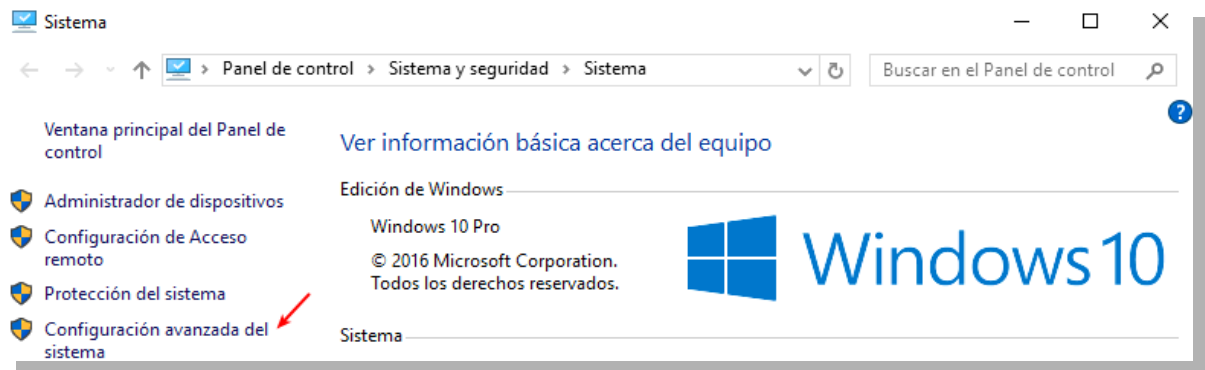
Posteriormente descomprimos la carpeta *bin* de la carpeta *hadoop-2.7.1* ubicado en el fichero descargado anteriormente, en una carpeta previamente creada en la raíz de nuestro disco duro, en nuestro caso la carpeta *Winutils*:



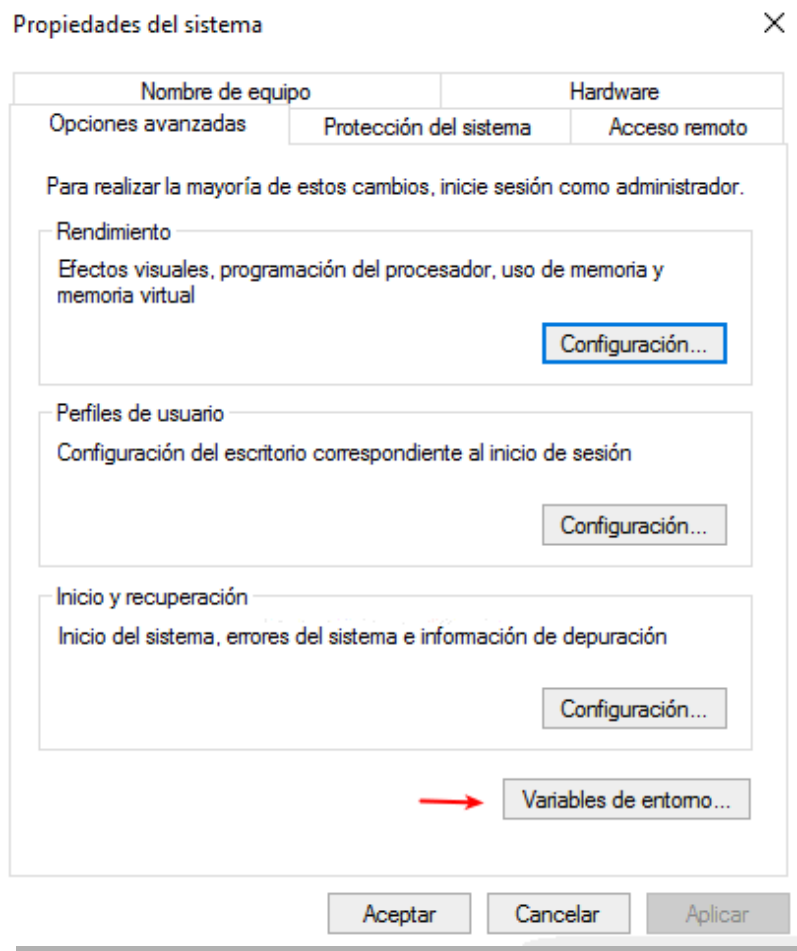
Más tarde, debemos establecer las variables de entorno. Para ello hacemos click derecho en *Este equipo* y seleccionamos el apartado *Propiedades*:



En la siguiente ventana que aparece, debemos seleccionar la opción *Configuración avanzada del sistema*:



Posteriormente seleccionamos la casilla: *Variables de entorno...*:



Añadimos nuevas variables de sistema seleccionando el apartado *Nueva...*:

The image shows two screenshots from a Windows operating system. The top screenshot is the 'Variables del sistema' (System Variables) dialog box. It contains a table with system variables and their values. A red arrow points to the 'Nueva...' (New...) button. The bottom screenshot is the 'Nueva variable del sistema' (New System Variable) dialog box. It has two text input fields: 'Nombre de la variable:' (Variable name) containing 'VARIABLE' and 'Valor de la variable:' (Variable value) containing 'DIRECTORIO'. Below these are two buttons: 'Examinar directorio...' (Browse for folder...) and 'Examinar archivo...' (Browse for file...). A red arrow points to the 'Aceptar' (OK) button.

Variable	Valor
ComSpec	C:\Windows\system32\cmd.exe
NUMBER_OF_PROCESSORS	1
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Wi...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Familv 6 Model 58 Stepping 9. GenuineIntel

Nueva variable del sistema

Nombre de la variable: VARIABLE

Valor de la variable: DIRECTORIO

Examinar directorio... Examinar archivo... Aceptar Cancelar

En mi caso, he asignado los nombres y valores de variables que aparecen en la siguiente tabla:

NOMBRE_VARIABLE	VALOR_DIRECTORIO
SCALA_HOME	C:\Program Files (x86)\scala\bin
SPARK_HOME	C:\spark
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_131
HADOOP_HOME	C:\winutils

Por lo que, las variables que se han establecido en mi sistema han sido:

Variables del sistema

Variable	Valor
ComSpec	C:\Windows\system32\cmd.exe
HADOOP_HOME	C:\winutils
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_131
NUMBER_OF_PROCESSORS	1
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Wi...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

Variables del sistema

Variable	Valor
SCALA_HOME	C:\Program Files (x86)\scala
SPARK_HOME	C:\spark
TEMP	C:\Windows\TEMP
TMP	C:\Windows\TEMP
USERNAME	SYSTEM
windir	C:\Windows

Posteriormente se debe editar la variable de entorno: *path*. Para ello seleccionamos la casilla *Editar...*:

Variables de entorno

Variables de usuario para usuario

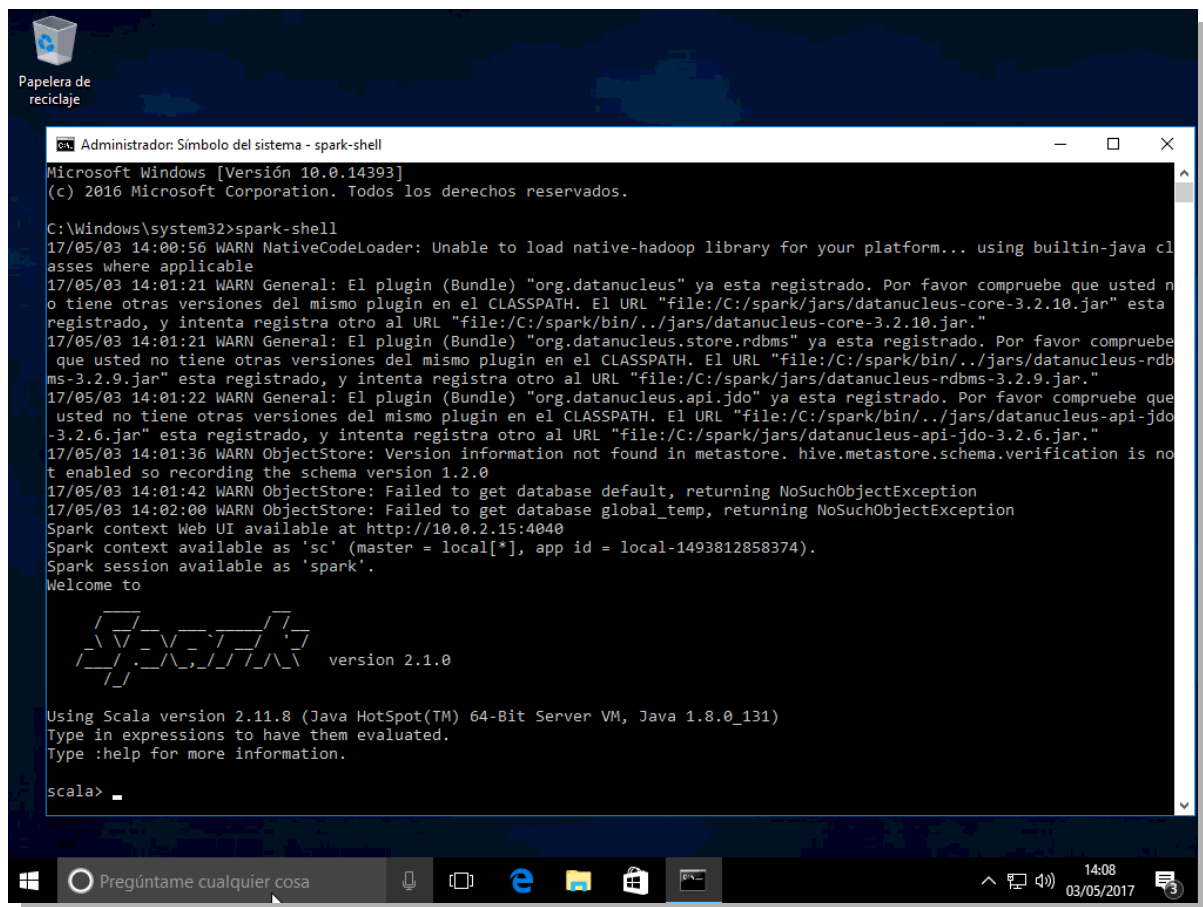
Variable	Valor
OneDrive	C:\Users\usuario\OneDrive
Path	%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Nueva... Editar... Eliminar

Añadimos dos nuevos valores a la variable, quedando de la siguiente forma:

[...];%SPARK_HOME%\bin;%JAVA_HOME%\bin

Posteriormente abrimos la consola de comandos como administrador y ejecutamos el comando *spark-shell*:



```
Administrador: Símbolo del sistema - spark-shell
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>spark-shell
17/05/03 14:00:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/05/03 14:01:21 WARN General: El plugin (Bundle) "org.datanucleus" ya esta registrado. Por favor compruebe que usted no tiene otras versiones del mismo plugin en el CLASSPATH. El URL "file:/C:/spark/jars/datanucleus-core-3.2.10.jar" esta registrado, y intenta registra otro al URL "file:/C:/spark/bin/./jars/datanucleus-core-3.2.10.jar."
17/05/03 14:01:21 WARN General: El plugin (Bundle) "org.datanucleus.store.rdbms" ya esta registrado. Por favor compruebe que usted no tiene otras versiones del mismo plugin en el CLASSPATH. El URL "file:/C:/spark/bin/./jars/datanucleus-rdbms-3.2.9.jar" esta registrado, y intenta registra otro al URL "file:/C:/spark/jars/datanucleus-rdbms-3.2.9.jar."
17/05/03 14:01:22 WARN General: El plugin (Bundle) "org.datanucleus.api.jdo" ya esta registrado. Por favor compruebe que usted no tiene otras versiones del mismo plugin en el CLASSPATH. El URL "file:/C:/spark/bin/./jars/datanucleus-api-jdo-3.2.6.jar" esta registrado, y intenta registra otro al URL "file:/C:/spark/jars/datanucleus-api-jdo-3.2.6.jar."
17/05/03 14:01:36 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.validation is not enabled so recording the schema version 1.2.0
17/05/03 14:01:42 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/05/03 14:02:00 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1493812858374).
Spark session available as 'spark'.
Welcome to

  ____  __
 / ___/ /_  __
/ /   / __/ /_
/ /___/ __/ /_
/_/___/_/ /_

version 2.1.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_131)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

¡ RECOMENDACIÓN !

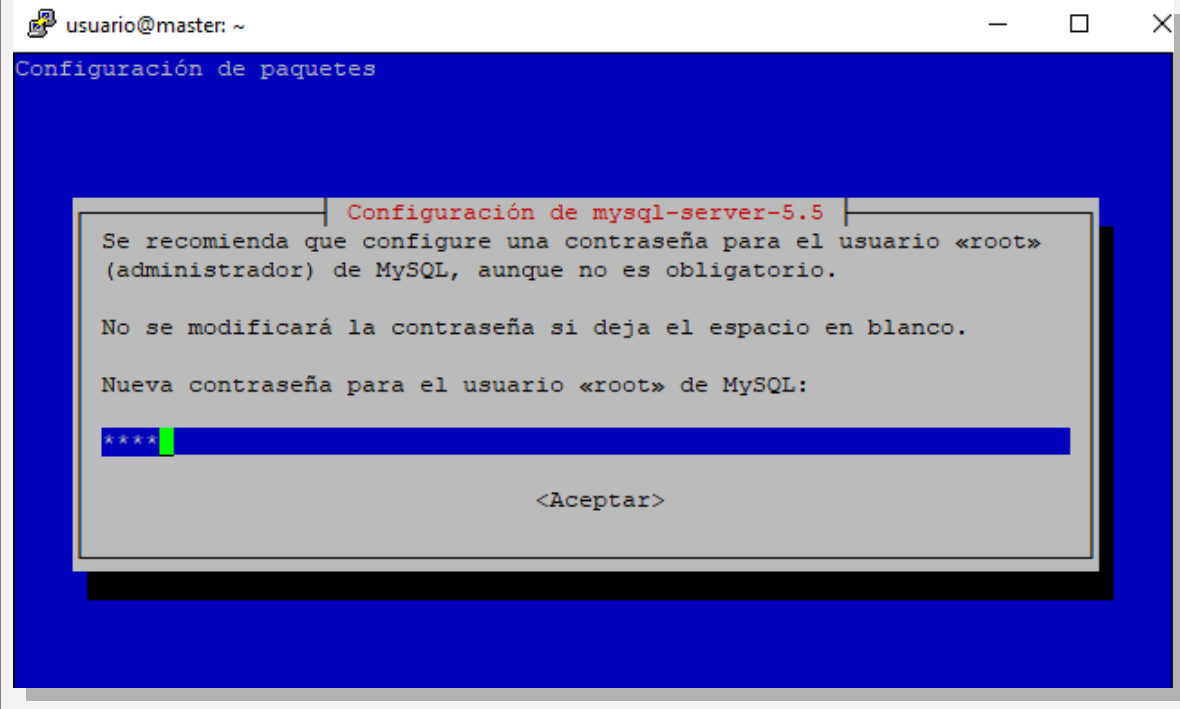
Para evitar la salida de gran cantidad de mensaje a la hora de ejecutar Spark, renombramos el fichero *log4j.properties.template* a *log4j.properties*, ubicado en *C:\spark-2.1.0-bin-hadoop2.7\config*.

Posteriormente se debe cambiar la primera entrada de configuración de *log4j.rootCategory=INFO* a *log4j.rootCategory=WARN*.

9. CONEXIÓN DE MYSQL CON SPARK

Apache Spark puede realizar consultas a la base de datos MySQL incluso 10 veces más rápido. Por lo que, vamos a realizar una conexión entre Apache Spark y MySQL mediante *spark-shell*. Para ello, se debe instalar MySQL:

```
root@master:/# apt-get install mysql-server
```



Accedemos a MySQL. para crear y usar una nueva base de datos:

```
root@master:/# mysql -u root -p
```

```
mysql> CREATE DATABASE db_spark;  
mysql> USE db_spark;
```

Posteriormente creamos una tabla de ejemplo:

```
mysql> CREATE TABLE `clientes` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(50) DEFAULT NULL,  
  `apellidos` varchar(50) DEFAULT NULL,  
  `género` char(1) DEFAULT NULL,  
  `edad` tinyint(4) DEFAULT NULL,  
  PRIMARY KEY (`ID`)  
);
```

Insertamos algunos datos a la tabla creada anteriormente:

```
mysql> INSERT INTO clientes (nombre, apellidos, género, edad)
VALUES ('Alejandro', 'Palomino García', 'H', 20);
mysql> INSERT INTO clientes (nombre, apellidos, género, edad)
VALUES ('Laura', 'López Sánchez', 'M', 51);
mysql> INSERT INTO clientes (nombre, apellidos, género, edad)
VALUES ('Daniel', 'Ruíz Domingo', 'H', 68);
mysql> INSERT INTO clientes (nombre, apellidos, género, edad)
VALUES ('Ignacio', 'Robles Martínez', 'H', 15);
mysql> INSERT INTO clientes (nombre, apellidos, género, edad)
VALUES ('Sonia', 'Paéz García', 'M', 41);
```

Más tarde, se debe descargar el driver *MySQL JDBC* para poder realizar la conexión:

```
root@master:/# wget http://central.maven.org/maven2/mysql/mysql-connector-
java/5.1.26/mysql-connector-java-5.1.26.jar
```

Iniciamos spark-shell con el parámetro *--jars* para poder utilizar el driver *mysql-connector-java-5.1.26.jar*:

```
root@master:/# MASTER="spark://10.0.0.1:7077" spark-shell --jars mysql-connector-
java-5.1.26.jar
```

Posteriormente creamos una variable donde se almacenará la información de la tabla *clientes*:

```
scala> val dataframe_mysql = spark.sqlContext.read.format("jdbc").option("url",
"jdbc:mysql://localhost/db_spark").option("driver",
"com.mysql.jdbc.Driver").option("dbtable", "clientes").option("user",
"root").option("password", "root").load()
```

Para mostrar el contenido del dataframe¹³ en tiempo real, se debe ejecutar el comando:

```
scala> dataframe_mysql.show
```

```
scala> dataframe_mysql.show
+----+-----+-----+-----+
| id | nombre | apellidos | género | edad |
+----+-----+-----+-----+
| 1 | Alejandro | Palomino García | H | 20 |
| 2 | Laura | López Sánchez | M | 51 |
| 3 | Daniel | Ruiz Domingo | H | 68 |
| 4 | Ignacio | Robles Martínez | H | 15 |
| 5 | Sonia | Paéz García | M | 41 |
+----+-----+-----+-----+
```

¹³*DataFrame*: conjunto de datos organizado en columnas. Es equivalente a una tabla en una base de datos relacional.

9.1 Ejecución de algunas consultas al dataframe

Algunas consultas que se le pueden realizar a nuestro dataframe son:

- Mostrar datos de la columna "nombre":

```
scala> dataframe_mysql.select("nombre").show()
+-----+
| nombre |
+-----+
| Alejandro |
| Laura |
| Daniel |
| Ignacio |
| Sonia |
+-----+
```

- Contar y agrupar por género:

```
scala> dataframe_mysql.groupBy("género").count().show()

+-----+-----+
| género | count |
+-----+-----+
| M | 2 |
| H | 3 |
+-----+-----+
```

- Mostrar personas con más de veinte años:

```
scala> dataframe_mysql.filter($"edad" > 20).show()
+---+-----+-----+-----+-----+
| id | nombre | apellidos | género | edad |
+---+-----+-----+-----+-----+
| 2 | Laura | López Sánchez | M | 51 |
| 3 | Daniel | Ruíz Domingo | H | 68 |
| 5 | Sonia | Paéz García | M | 41 |
+---+-----+-----+-----+-----+
```

9.2 Creación de un DataFrame basado en el contenido de un fichero JSON

Como se puede observar, tenemos un fichero JSON con el siguiente contenido:

/home/usuario/correos.json

```
{ "nombre": "Rodrigo", "apellidos": "Vergara García", "email": "rodverga@gmail.com" }  
{ "nombre": "Sandra", "apellidos": "Bernal Romero", "email": "sabero@gmail.com" }  
{ "nombre": "Alberto", "apellidos": "Vaquero Sánchez", "email": "alvasan@gmail.com" }
```

Para crear un DataFrame basado en el fichero anterior vamos a ejecutar el siguiente comando:

```
scala> val correosDF = spark.read.json("/home/usuario/correos.json")
```

Posteriormente mostramos el contenido de nuestro dataframe basado en el fichero correos.json:

```
scala> correosDF.show()  
+-----+-----+-----+  
|  apellidos  |      email      | nombre |  
+-----+-----+-----+  
| Vergara García | rodverga@gmail.com | Rodrigo |  
| Bernal Romero | sabero@gmail.com | Sandra |  
| Vaquero Sánchez | alvasan@gmail.com | Alberto |  
+-----+-----+-----+
```

Por otra parte, los DataFrames se puede convertir en un “dataset” asignándole una clase:

```
scala> case class Correos(nombre: String, email: String, apellidos:String)  
scala> val correosDS = spark.read.json("/home/usuario/correos.json").as[Correos]  
scala> correosDS.show()  
  
+-----+-----+-----+  
|  apellidos  |      email      | nombre |  
+-----+-----+-----+  
| Vergara García | rodverga@gmail.com | Rodrigo |  
| Bernal Romero | sabero@gmail.com | Sandra |  
| Vaquero Sánchez | alvasan@gmail.com | Alberto |  
+-----+-----+-----+
```

10. STREAMING DE TWEET'S

El core de Spark posee una librería denominada “Spark Streaming” para el manejo en directo de datos. Vamos a ilustrar un ejemplo con Twitter para hacer uso de ésta librería:

Para empezar, iniciamos spark-shell y agregamos varios ficheros con extensión .JAR:

```
# MASTER="spark://10.0.0.1:7077" spark-shell --jars twitter4j-core-4.0.6.jar,spark-streaming-twitter\_2.11-1.6.3.jar,commons-dbutils-1.6.jar,spark-core\_2.11-1.5.2.jar,twitter4j-stream-4.0.6.jar
```

Posteriormente ejecutamos los siguientes comandos:

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.twitter._
import org.apache.spark.streaming.StreamingContext._
import twitter4j.auth.Authorization
import twitter4j.Status
import twitter4j.auth.AuthorizationFactory
import twitter4j.conf.ConfigurationBuilder
import org.apache.spark.streaming.api.java.JavaStreamingContext

import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext
import org.apache.spark.mllib.feature.HashingTF
import org.apache.spark.mllib.linalg.Vector
import org.apache.spark.SparkConf
import org.apache.spark.api.java.JavaSparkContext
import org.apache.spark.api.java.function.Function
import org.apache.spark.streaming.Duration
import org.apache.spark.streaming.api.java.JavaDStream
import org.apache.spark.streaming.api.java.JavaReceiverInputDStream

val consumerKey = "[TU_CONSUMERKEY_AQUÍ]"
val consumerSecret = "[TU_CONSUMERSECRET_AQUÍ]"
val accessToken = "[TU_ACCESSTOKEN_AQUÍ]"
val accessTokenSecret = "[TU_ACCESSTOKENSECRET_AQUÍ]"
val url = "https://stream.twitter.com/1.1/statuses/filter.json"

val documents: RDD[Seq[String]] = spark.sparkContext.textFile("").map(_._split(" ")).toSeq

// Twitter Streaming
```

```

val ssc = new JavaStreamingContext(spark.sparkContext,Seconds(2))

val conf = new ConfigurationBuilder()
conf.setOAuthAccessToken(accessToken)
conf.setOAuthAccessTokenSecret(accessTokenSecret)
conf.setOAuthConsumerKey(consumerKey)
conf.setOAuthConsumerSecret(consumerSecret)
conf.setStreamBaseURL(url)
conf.setSiteStreamBaseURL(url)

val filter = Array("Betis", "ApacheSpark")

val auth = AuthorizationFactory.getInstance(conf.build())
val tweets : JavaReceiverInputDStream[twitter4j.Status] = TwitterUtils.createStream(ssc,
auth, filter)

val statuses = tweets.dstream.map(status => status.getText)
statuses.print()
ssc.start()

```

NOTA: En mi caso, voy a filtrar por los términos: “Betis” y “ApacheSpark”

Como se puede observar en la siguiente ilustración todos los Tweets y ReTweets que se están haciendo en directo y contengan una de ambas palabras, aparecerán por pantalla:

```

usuario@master ~
-----
Time: 1494514454000 ms
-----
Time: 1494514456000 ms
-----
Time: 1494514458000 ms
RT @betis_bet: Portada ABC (1992) "El Betis ofrece 900 millones de pesetas por Maradona". https://t.co/WgRBb1Sz9
-----
Time: 1494514460000 ms
-----
Time: 1494514462000 ms
RT @juanbustos5: Lorenzo Serra Ferrer nuevo vicepresidente deportivo del Betis Muy buena elección. Profesional, trabajador y necesario par...
-----
Time: 1494514464000 ms
-----
Time: 1494514466000 ms
-----
Time: 1494514468000 ms
-----
Time: 1494514470000 ms
-----
Time: 1494514472000 ms
Tweet de ejemplo sobre #ApacheSpark. By: Alejandro P.
RT @tarugobetico: Ojo!,Nolito no juega desde diciembre en el City,el jugador quiere salir,habrá que estar atentos...el Betis lo sigue desde...
RT @juanantgonzalez: Serra dijo un día "ahora el Betis será lo que quiera su presidente" Ahora sin duda el Betis será lo que Serra quiera
-----
Time: 1494514474000 ms
-----
RT @betis_bet: Portada Don Balón (2005) Campeón de la copa de rey. "MUSHO BETIS". https://t.co/g0N00E9dbu

```

11. ZEPPELIN + SPARK

11.1 ¿Qué es Zeppelin?

Apache Zeppelin permite trabajar sobre una interfaz web como si de una shell se tratase, también conocido como *web notebook*. Esta herramienta se centra en el análisis de datos interactivos mediante lenguajes y tecnologías como Shell, Spark, SparkSQL, Elasticsearch, etc...

11.2 Instalación y configuración de Zeppelin

Lo primero que debemos de hacer es ejecutar el siguiente comando para descargar el paquete comprimido *zeppelin-0.7.1-bin-all.tgz*:

```
root@master:/# wget http://apache.rediris.es/zeppelin/zeppelin-0.7.1/zeppelin-0.7.1-bin-all.tgz
```

Posteriormente descomprimos el fichero en la ubicación */opt/*:

```
root@master:/# tar zxvf zeppelin-0.7.1-bin-all.tgz -C /opt/
```

Añadimos una nueva entrada en el fichero *.bashrc*:

```
root@master:/# nano ~/.bashrc

export ZEPPELIN_HOME=/opt/zeppelin-0.7.1-bin-all/
export PATH="/opt/zeppelin-0.7.1-bin-all/bin/:$PATH"
```

Cargamos las variables definidas anteriormente en el fichero *.bashrc*:

```
root@master:/# source ~/.bashrc
```

Podemos comprobar que se ha definido correctamente la variable ejecutando el siguiente comando:

```
root@master:/home/usuario# zeppelin-daemon.sh --version
0.7.1
```


Posteriormente vamos a establecer *la URL del master*. Para ello nos dirigimos a la carpeta *conf* y renombramos el fichero denominado *zeppelin-env.sh.template* a *zeppelin-env.sh*:

```
root@master:/opt/zeppelin-0.7.1-bin-all/conf# cp zeppelin-env.sh.template zeppelin-env.sh
```

Una vez hayamos renombrado el fichero, editamos y añadimos el siguiente contenido:

```
root@master:/opt/zeppelin-0.7.1-bin-all/conf# nano zeppelin-env.sh
export MASTER=spark://10.0.0.1:7077
```

También debemos cambiar el puerto por defecto (*8080*), ya que también es usado por Spark. Para ello nos dirigimos a la carpeta *conf* y renombramos el fichero denominado *zeppelin-site.xml.template* a *zeppelin-site.xml*:

```
root@master:/opt/zeppelin-0.7.1-bin-all/conf# cp zeppelin-site.xml.template zeppelin-site.xml
```

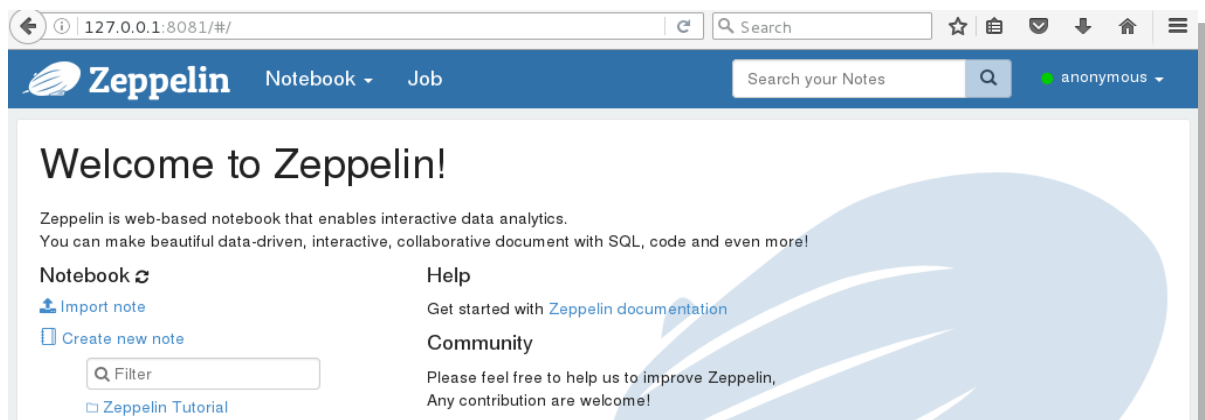
Más tarde editamos el nuevo fichero renombrado, cambiando el puerto *8080* por otro distinto. En mi caso he puesto el puerto *8081*:

```
root@master:/opt/zeppelin-0.7.1-bin-all/conf# nano zeppelin-site.xml
<property>
  <name>zeppelin.server.port</name>
  <value>8081</value>
  <description>Server port.</description>
</property>
```

Más tarde, iniciamos Zeppelin gracias al siguiente comando:

```
root@master:/home/usuario# zeppelin-daemon.sh start
Zeppelin start          [ OK ]
```

Si accedemos a *127.0.0.1:8081/* desde el navegador web, podemos observar la página principal de Zeppelin:



11.3 Ejemplos

Vamos a ejecutar un par de ejemplos en Zeppelin utilizando nuestro clúster de Spark como intérprete. Para ello, creamos una nueva nota con el siguiente contenido:

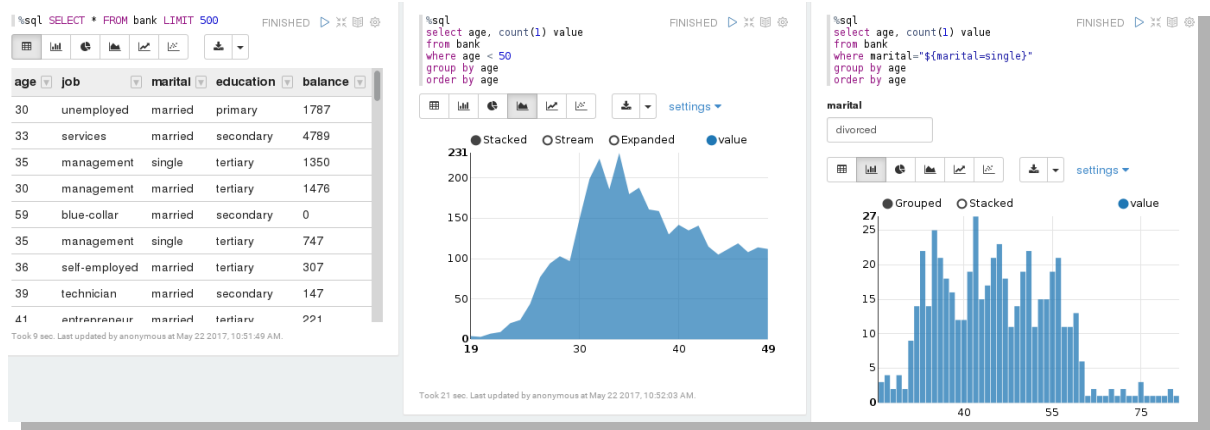
```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

val bankText = sc.parallelize(
  IOUtils.toString(
    new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv"),
    Charset.forName("utf8")).split("\n"))

case class Bank(age: Integer, job: String, marital: String, education: String, balance: Integer)

val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
  s => Bank(s(0).toInt,
    s(1).replaceAll("\"", ""),
    s(2).replaceAll("\"", ""),
    s(3).replaceAll("\"", ""),
    s(5).replaceAll("\"", "").toInt
  )
).toDF()
bank.registerTempTable("bank")
```

Una vez creada la tabla “bank”, vamos a realizar algunas consultas de prueba:



También podemos utilizar *PySpark* como se muestra en el siguiente ejemplo:

```
%pyspark
import StringIO
import matplotlib
matplotlib.use('Agg')
def show(p):
  img = StringIO.StringIO()
```

```
p.savefig(img, format='svg')
img.seek(0)
print '%html ' + img.buf
show(plt)
```

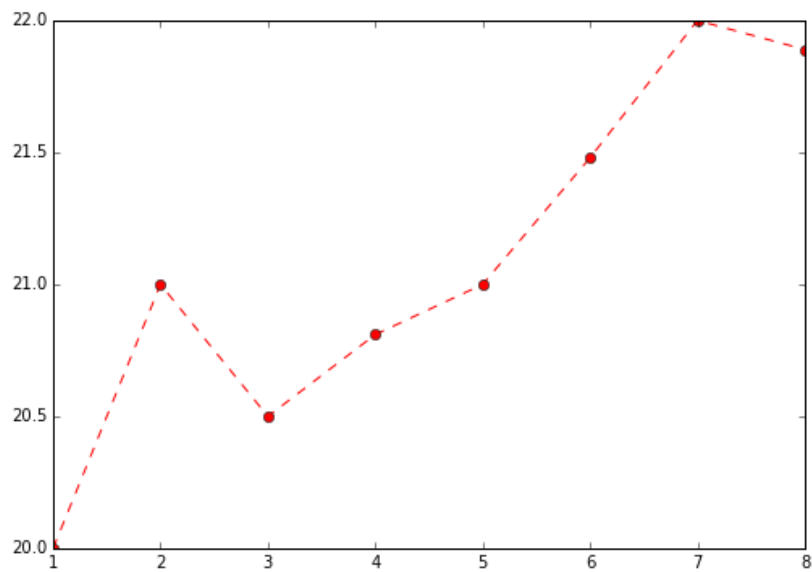
Posteriormente damos valor a las variables X e Y para generar el gráfico:

```
%pyspark
import matplotlib.pyplot as plt

# Test data
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [20, 21, 20.5, 20.81, 21.0, 21.48, 22.0, 21.89]

# Plot
plt.plot(x, y, linestyle='dashed', marker='o', color='red')
```

```
[<matplotlib.lines.Line2D object at 0x7fc501203750>]
```



NOTA: para ejecutar el último ejemplo debemos de instalar la librería “matplotlib”. Podemos hacerlo con apt-get install python-matplotlib

Adjunto enlace de un video subido a Youtube, donde ejecuto los ejemplos expuestos anteriormente:

<https://youtu.be/nuJBH9GfELw>

12. PROBLEMAS ENCONTRADOS

12.1: cat: /release: No such file or directory

Si aparece el error “*cat: /release: No such file or directory*” al iniciar Scala, se debe a que se tiene que establecer la variable JAVA_HOME:

```
root@master:/# nano /etc/environment  
  
JAVA_HOME=/usr/lib/jvm/java-8-oracle/
```

Recargamos la variable establecida anteriormente:

```
root@master:/# source /etc/environment
```

12.2 HOSTNAME: HOSTNAME: Name or service not known

Si aparece el error “*java.net.UnknownHostException: HOSTNAME: HOSTNAME: Name or service not known*” al iniciar el clúster, se debe modificar el fichero “*/etc/hosts*” y comprobar que se encuentra la línea “*127.0.1.1 hostname*”, donde la variable “hostname” es el nombre de tu máquina:

```
root@master:/# nano /etc/hosts  
  
127.0.1.1 hostname
```

12.3 The root scratch dir: /tmp/hive on HDFS should be writable

El error “*The root scratch dir: /tmp/hive on HDFS should be writable*” me apareció a la hora de ejecutar el comando *spark-shell* en *Windows 10*. La solución es tan sencilla como cambiar los permisos a la carpeta *\tmp\hive*:

```
C:\winutils\bin\winutils.exe chmod 777 C:\tmp\hive
```

12.4 Java.lang.NoClassDef FoundError: org/apache/spark/Logging

Mientras ejecutaba los comandos para hacer el ejemplo de streaming de tweets me apareció el siguiente error *Exception in thread "main" java.lang.NoClassDef FoundError: org/apache/spark/Logging*. La solución fue tan sencilla como descargar el siguiente fichero con extensión .jar del siguiente enlace: [spark-core_2.11-1.5.2.jar](#) y añadirlo al ejecutar la spark-shell.

12.5 warn "replicated to only 0 peer(s) instead of 1 peers"

Tras ejecutar los comandos para hacer streaming de tweets en *spark-shell* me apareció la siguiente advertencia: *warn "replicated to only 0 peer(s) instead of 1 peers"*.

El problema es que no estaba ejecutando la spark-shell en modo clúster. Estaba ejecutando la siguiente sentencia:

```
# spark-shell --jars twitter4j-core-4.0.6.jar,spark-streaming-twitter_2.11-1.6.3.jar,commons-dbutils-1.6.jar,spark-core_2.11-1.5.2.jar,twitter4j-stream-4.0.6.jar
```

La solución fue añadir MASTER="spark://10.0.0.1:7077" al principio del comando:

```
# MASTER="spark://10.0.0.1:7077" spark-shell --jars twitter4j-core-4.0.6.jar,spark-streaming-twitter_2.11-1.6.3.jar,commons-dbutils-1.6.jar,spark-core_2.11-1.5.2.jar,twitter4j-stream-4.0.6.jar
```

13. BIBLIOGRAFÍA

<http://spark.apache.org/>
<https://geekytheory.com/apache-spark-que-es-y-como-funciona/>
<http://spark.apache.org/docs/latest/>
<https://www.ibm.com/analytics/es/es/technology/spark.html>
<https://aws.amazon.com/es/emr/details/spark/>
http://www.w3ii.com/es/apache_spark/apache_spark_introduction.html
<https://aspgems.com/blog/apache-spark/7-razones-por-las-que-deberias-usar-apache-spark>
<http://josededeveloper.com/2015/06/12/primer-ejemplo-con-apache-spark/>
https://en.wikipedia.org/wiki/Apache_Spark
<https://www.cloudera.com/documentation/enterprise/5-6-x/PDF/cloudera-spark.pdf>
<https://bbvaopen4u.com/es/actualidad/apache-spark-las-ventajas-de-usar-al-nuevo-rey-de-big-data>
<https://aspgems.com/blog/big-data/por-que-usamos-apache-spark-para-nuestros-proyectos-de-big-data>
https://www.tutorialspoint.com/apache_spark/apache_spark_installation.htm
<http://www.baoss.es/apache-spark/>
<http://mahout.apache.org/>
<http://hunch.net/~vw/>
<http://arjon.es/2015/08/23/vagrant-spark-zeppelin-a-toolbox-to-the-data-analyst/>
<https://geekytheory.com/como-crear-un-cluster-de-servidores-con-apache-spark>
<https://www.santoshsrinivas.com/installing-apache-spark-on-ubuntu-16-04/>
<https://www.scala-lang.org/old/node/166.html>
<https://www.adictosaltrabajo.com/tutoriales/introduccion-a-apache-spark-batch-y-streaming/>
<http://paxcel.net/blog/how-to-setup-apache-spark-standalone-cluster-on-multiple-machine/>
<http://www.franciscojavierpulido.com/2014/05/spark-iii-como-crear-y-configurar-un.html>
<http://spark.apache.org/docs/latest/cluster-overview.html>
<https://www.youtube.com/watch?v=omlwDosMGVk>
<http://formacionhadoop.com/aulavirtual/pluginfile.php/115/course/summary/Apache%20Spark%20Cap%C3%ADtulo%203%20-%20Spark%20Conceptos%20B%C3%A1sicos.pdf>
<http://damarcant.blogspot.com.es/2016/02/big-data-con-apache-spark-i-introduccion.html>
<http://formacionhadoop.com/aulavirtual/pluginfile.php/115/course/summary/Apache%20Spark%20Cap%C3%ADtulo%202%20-%20Introducci%C3%B3n%20a%20Apache%20Spark.pdf>
<http://blog.powerdata.es/el-valor-de-la-gestion-de-datos/spark-vs-hadoop-quien-saldrá-vencedor>
<https://www.xplenty.com/blog/2014/11/apache-spark-vs-hadoop-mapreduce/>
<http://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html>
<http://www.datamation.com.ar/big-data-hadoop-y-spark-competencia-o-complemento-7509>
<https://www.youtube.com/watch?v=fn3WeMZZcCk>
<https://databricks.com/spark/about>
http://www.bigdata-toronto.com/2016/assets/getting_started_with_apache_spark.pdf
<https://wp-alvaromonsalve.rhcloud.com/2015/02/25/apache-spark-operaciones-basicas-ii-transformaciones/>
<https://spark.apache.org/docs/latest/programming-guide.html>
http://www.w3ii.com/es/apache_spark/apache_spark_quick_guide.html
<http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html#distinct>
<https://books.google.es/books?id=JqunDAAQBAJ&pg=PT54&lpg=PT54#v=onepage&q&f=false>
<http://www.franciscojavierpulido.com/2014/05/spark-iv-desarrollando-programas-en.html>
<https://scalerablog.wordpress.com/2016/03/30/spark-operaciones-basicas-con-rdds/>
http://reader.digitalbooks.pro/content/preview/books/41061/book/OEBPS/Text/capitulo_3.html