
BIG DATA

**LENGUAJES DE SPARK
PYSPARK Y SCALA**

EDUARD LARA

1. INTRODUCCION

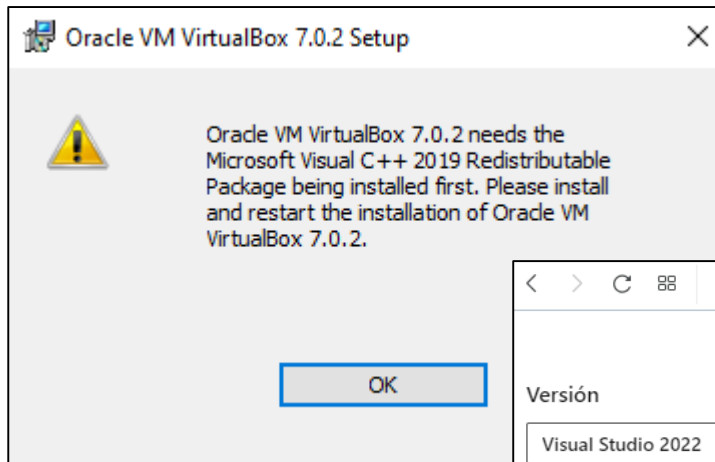
- ❖ Ante la gran explosión de datos que ha supuesto el BigData, los analistas necesitan herramientas que faciliten el estudio de esos datos en sistemas paralelos
- ❖ Apache Spark se ha situado rápidamente como la herramienta más popular para análisis de datos ya que ofrece facilidad de uso, velocidad y generalidad.
- ❖ Spark permite combinar en un único motor de ejecución y un entorno de desarrollo simple, diferentes tipos de computación:
 - ❖ el procesamiento de textos en paralelo
 - ❖ al aprendizaje automático
 - ❖ búsquedas SQL
 - ❖ el procesamiento de flujos en tiempo real
- ❖ Estas características hacen que SPARK sea un punto de inicio excelente para los que quieran introducir en el mundo de Big Data.

1. INTRODUCCION

- ❖ Para hacer las pruebas y examinar los lenguajes de programación de Spark, usaremos una máquina virtual de Virtual Box desplegada mediante el software vagrant
- ❖ Esa máquina virtual dispone ya de una instalación de Apache Spark y también de Apache Zeppelin
- ❖ Apache Zeppelin es una aplicación que nos va a permitir escribir y probar directamente los programas en SPARK usando un simple navegador web.
- ❖ El software necesario está disponible tanto para Microsoft Windows como para Mac OS X y Linux. Pero en este caso haremos el despliegue de la maquina virtual solo en Microsoft Windows.

2. INSTALACION VIRTUAL BOX

Paso 1. Para instalar Virtual Box en Windows, se debe de tener instalado previamente Microsoft Visual C++ 2019 Redistributable. El fichero debe de tener un nombre similar a vc_redist_x64.exe



learn.microsoft.com/es-es/cpp/windows/latest-supported-vc-redist

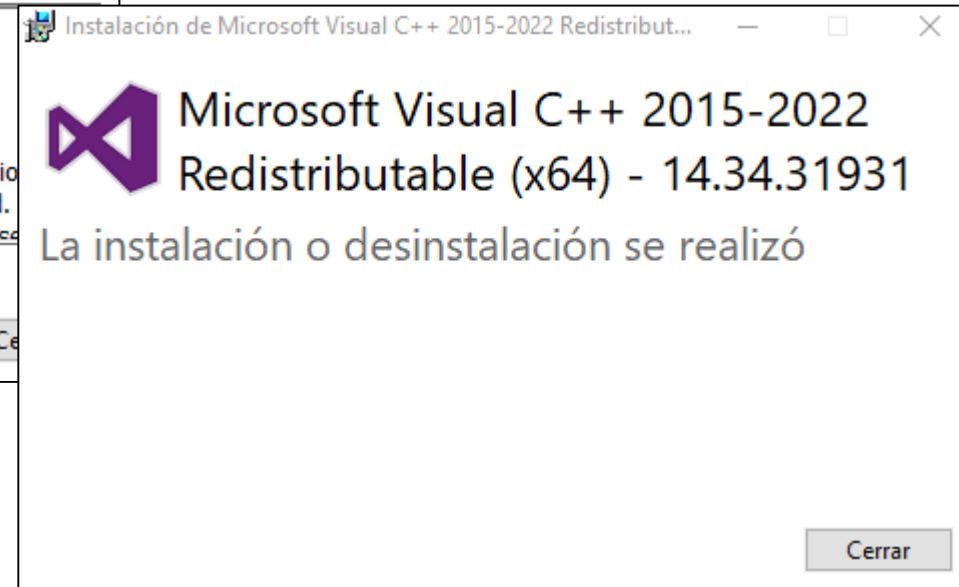
Visual Studio 2015, 2017, 2019 y 2022

En esta tabla se enumeran los paquetes compatibles más recientes de Microsoft Visual C++ Redistributable en inglés (en-US) para Visual Studio 2015, 2017, 2019 y 2022. La versión compatible más reciente incluye las características de C++ implementadas más recientes, la seguridad, la confiabilidad y las mejoras de rendimiento. También incluye las últimas actualizaciones de conformidad del lenguaje C++ estándar y los estándares de biblioteca. Se recomienda instalar esta versión para todas las aplicaciones creadas con Visual Studio 2015, 2017, 2019 o 2022.

Architecture	Vínculo	Notas
ARM64	https://aka.ms/vs/17/release/vc_redist.arm64.exe	Permalink para obtener la versión de ARM64 compatible más reciente
X86	https://aka.ms/vs/17/release/vc_redist.x86.exe	Permalink para obtener la versión x86 compatible más reciente

2. INSTALACION VIRTUAL BOX

Paso 2. Instalamos esta librería. Microsoft Visual C++ Redistributable es una serie de archivos que se deben instalar en el sistema para que se puedan usar ciertos programas programados con Visual C++, entre ellos el Virtual Box



2. INSTALACION VIRTUAL BOX

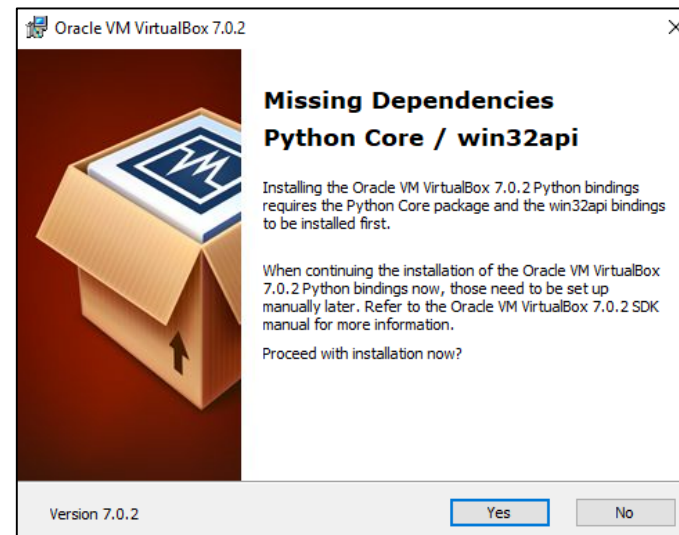
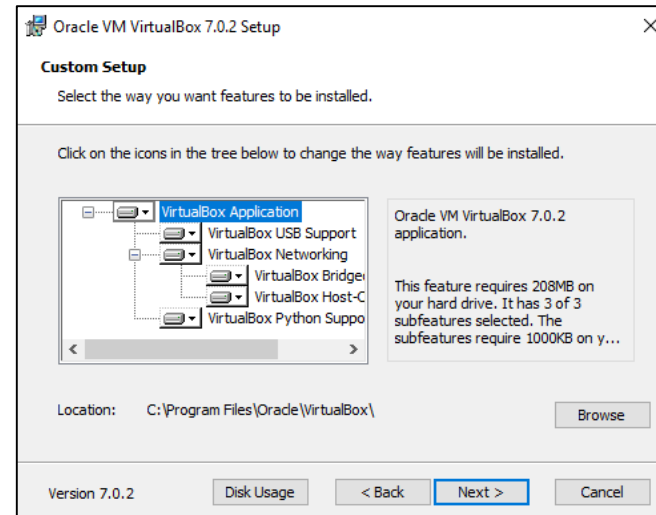
Paso 3. Para descargar virtual box, vamos a la página <https://www.virtualbox.org/wiki/Downloads> (la versión actual es la 7)



The screenshot shows the VirtualBox Downloads page in a web browser. The browser's address bar displays www.virtualbox.org/wiki/Downloads. The page features the VirtualBox logo and a navigation menu on the left with links to About, Screenshots, Downloads, Documentation, End-user docs, Technical docs, Contribute, and Community. The main content area is titled "Download VirtualBox" and includes a sub-header "VirtualBox binaries". Below this, it states: "Here you will find links to VirtualBox binaries and its source code." and "By downloading, you agree to the terms and conditions of the respective license." It also mentions: "If you're looking for the latest VirtualBox 6.1 packages, see [VirtualBox 6.1 builds](#). Version 6.1 will remain supported until December 2023." The section "VirtualBox 7.0.4 platform packages" lists several download links: [Windows hosts](#), [macOS / Intel hosts](#), [Developer preview for macOS / Arm64 \(M1/M2\) hosts](#), [Linux distributions](#), [Solaris hosts](#), and [Solaris 11 IPS hosts](#). At the bottom of the page, it says: "The binaries are released under the terms of the GPL version 3." and "See the [changelog](#) for what has changed." A download completion notification is visible in the bottom right corner, showing a VirtualBox logo icon, the filename "VirtualBox-7.0.2-154219-Win.exe", and the status "Download complete".

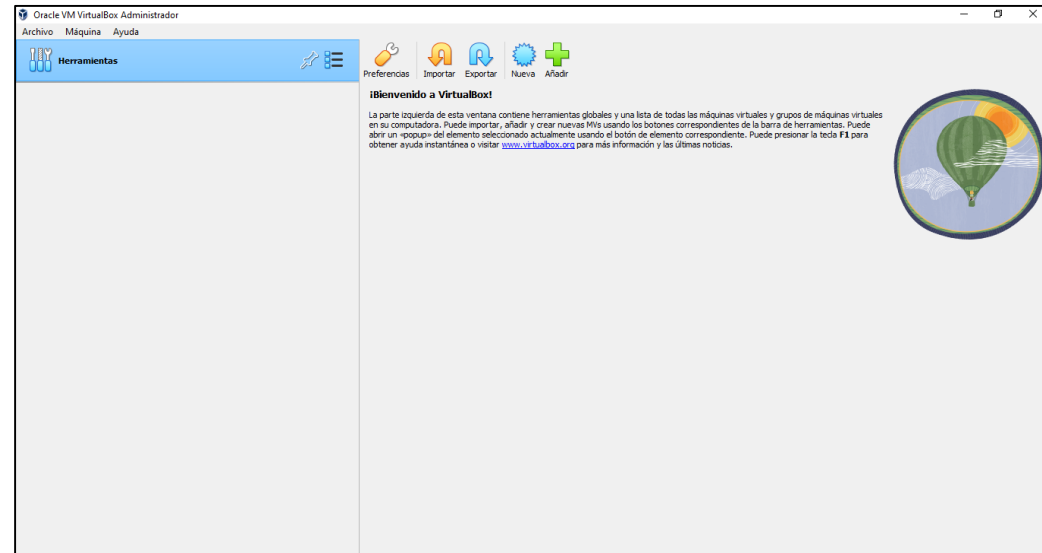
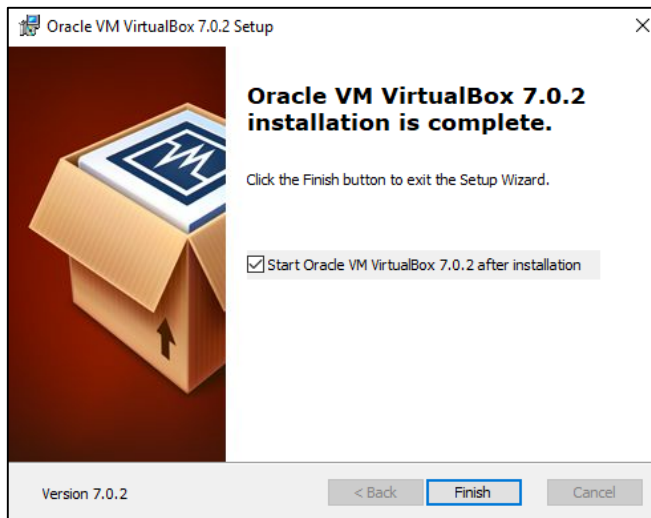
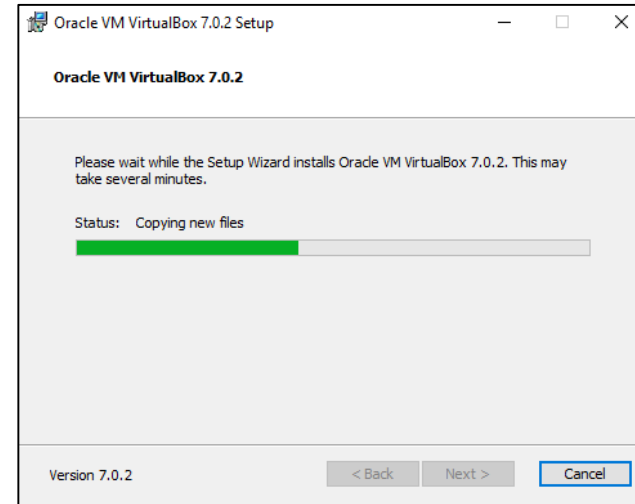
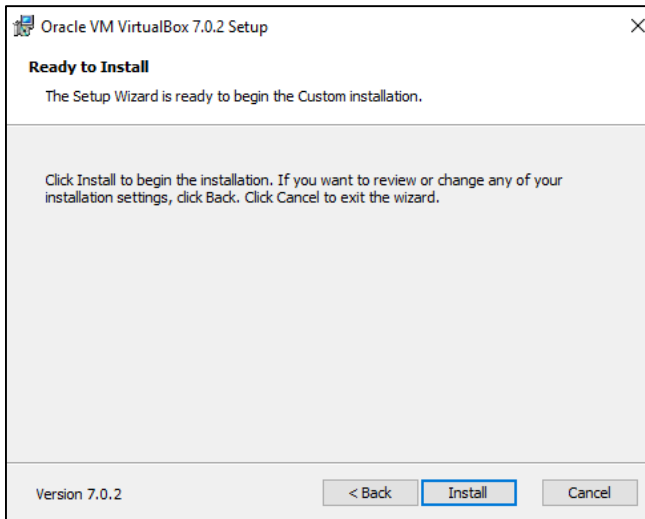
2. INSTALACION VIRTUAL BOX

Paso 4. Iniciamos la instalación de virtual box:



2. INSTALACION VIRTUAL BOX

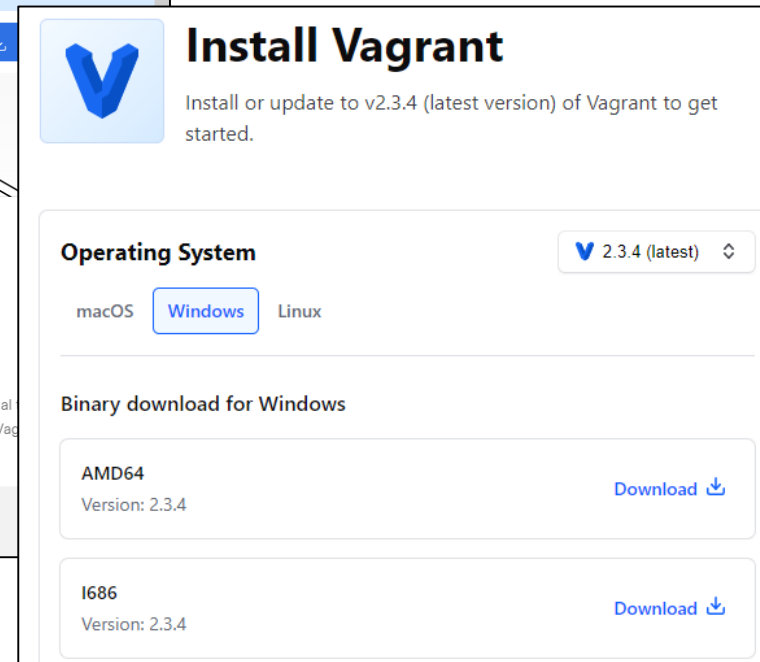
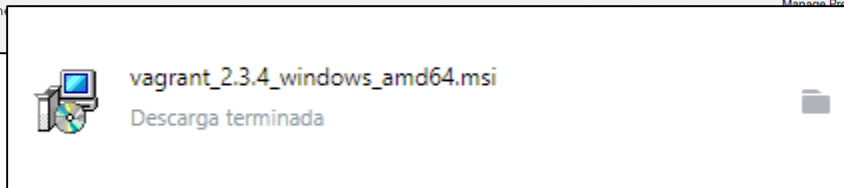
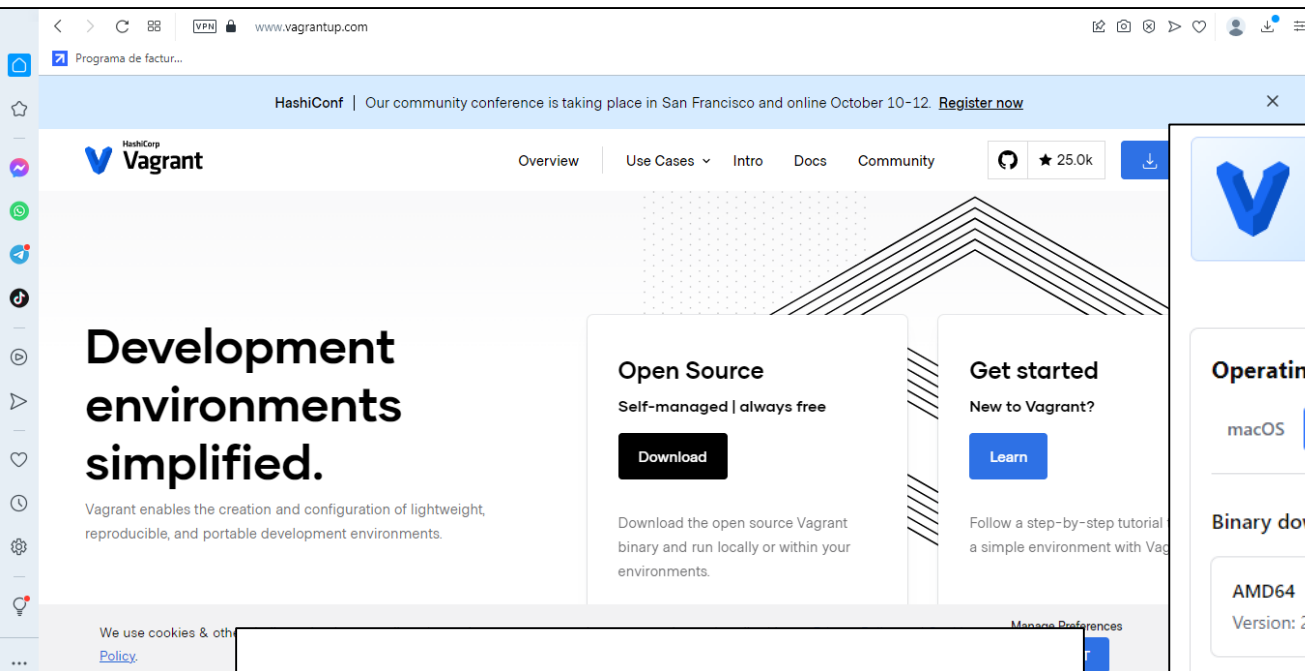
Paso 5. Finalizamos la instalación e iniciamos el programa



3. INSTALACION VAGRANT

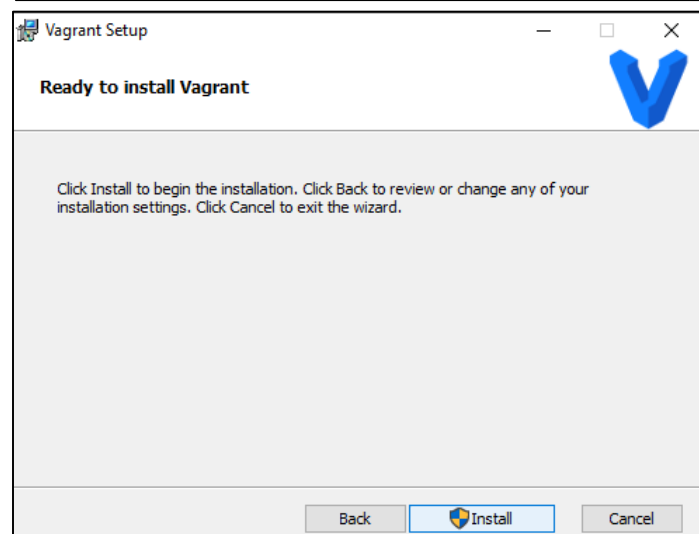
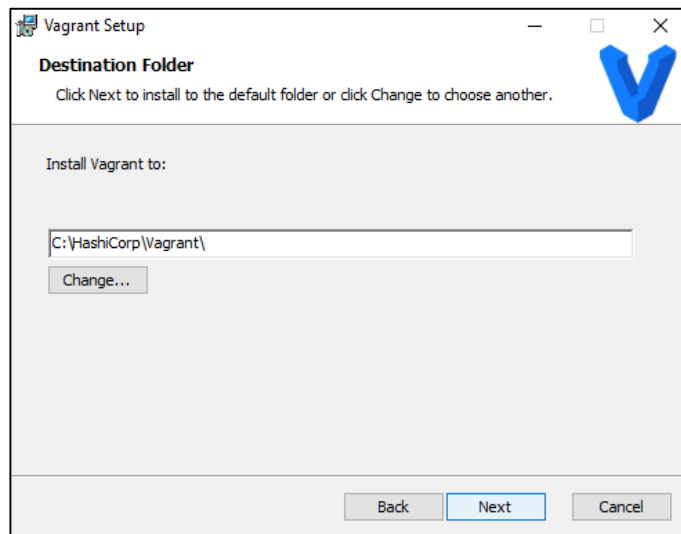
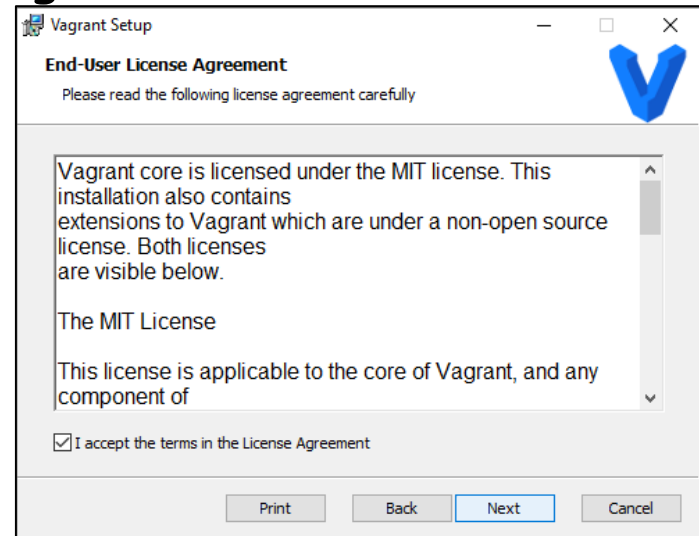
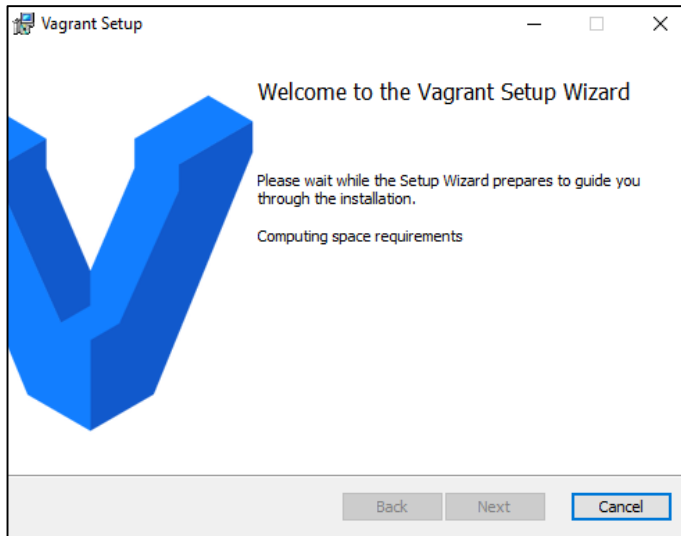
Paso 1. Una vez instalado el gestor de máquinas virtuales Virtual Box, usaremos Vagrant para desplegar estas máquinas virtuales.

Vamos a la pagina de vagrant: www.vagrantup.com, y descargamos la ultima versión para Windows:



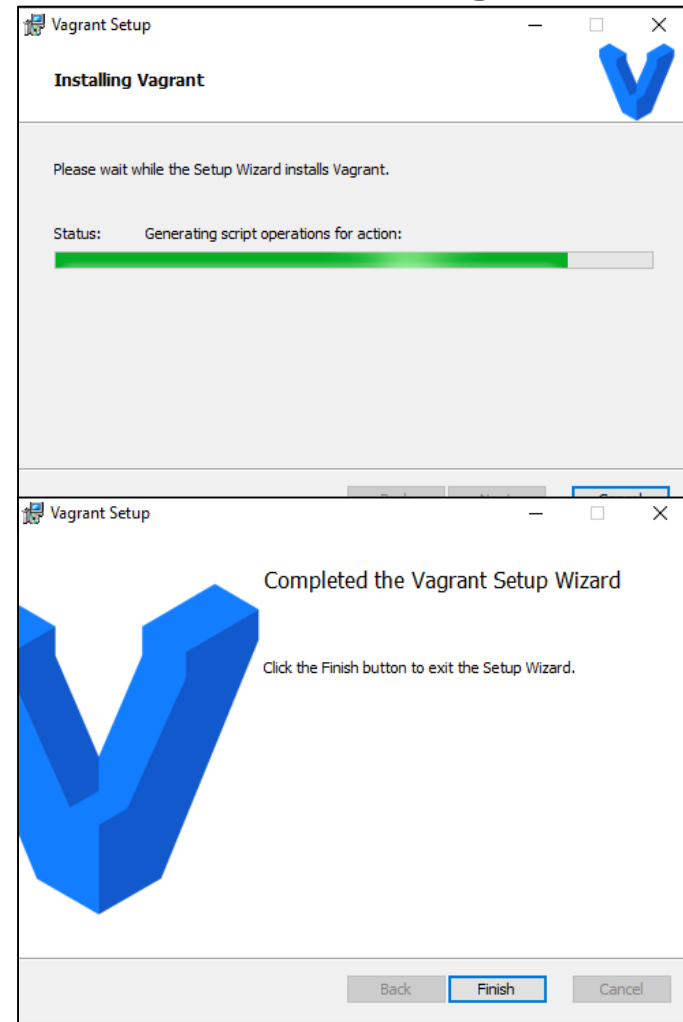
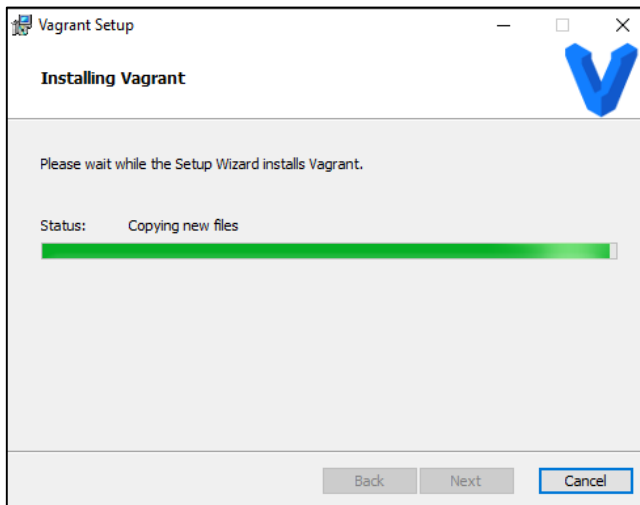
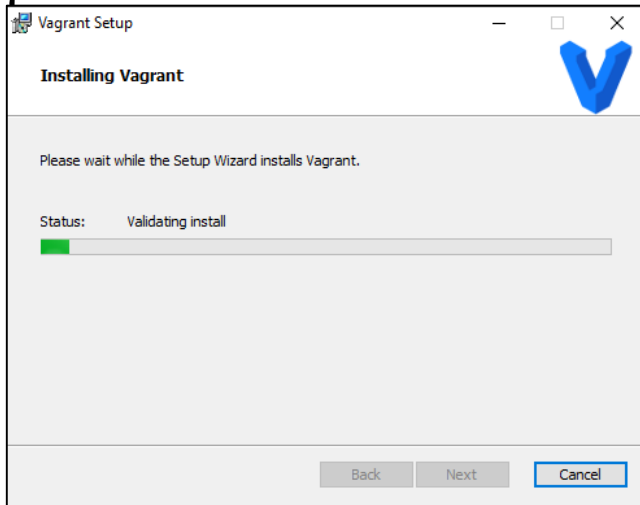
3. INSTALACION VAGRANT

Paso 2. Iniciamos la instalación de Vagrant



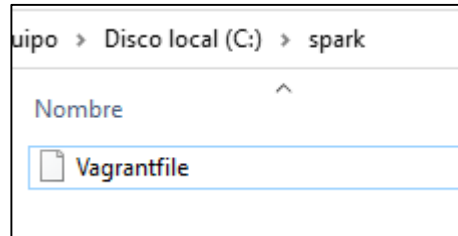
3. INSTALACION VAGRANT

Paso 3. Una vez terminada la instalación, Vagrant nos recomienda que reiniciamos el sistema para que se active la configuración



4. INSTALACION MAQUINA VIRTUAL

Paso 1. Descargamos el fichero vagrantfile que tiene la descripción de la máquina virtual que contiene Apache Zeppelin con todos los contenidos de Spark. Lo ponemos en un directorio raíz de uno de los discos, con un nombre sencillo:



```
Vagrantfile: Bloc de notas
Archivo Edición Formato Ver Ayuda
Vagrant.configure(2) do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  config.vm.box = "tfpena/cursospark"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 8080, host: 8080, host_ip: "127.0.0.1"
  config.vm.network "forwarded_port", guest: 8081, host: 8081, host_ip: "127.0.0.1"
  config.vm.network "forwarded_port", guest: 8085, host: 8085, host_ip: "127.0.0.1"
  config.vm.network "forwarded_port", guest: 4040, host: 4040, host_ip: "127.0.0.1"
  config.vm.network "forwarded_port", guest: 4000, host: 4000, host_ip: "127.0.0.1"
```

```
config.vm.provider "virtualbox" do |vb|
  vb.memory = "4096"
  vb.cpus = "4"
  vb.name = "cursospark"
end

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.
# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end

# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
# documentation for more information about their specific syntax and use.
# config.vm.provision "shell", inline: <<-SHELL
#   sudo apt-get update
#   sudo apt-get install -y apache2
# SHELL
end
```

4. INSTALACION MAQUINA VIRTUAL

Paso 2. Vagrantfile es un fichero de texto normal que se puede abrir utilizando el bloc de notas. Nos indica la configuración de la máquina virtual:

- ❖ el nombre de la máquina
- ❖ los puertos que tiene abiertos (sólo son visibles desde la máquina local con lo cual no suponen ningún problema de seguridad)
- ❖ la cantidad de memoria en megabytes y de Cores o de CPU que se reservan para la máquina virtual.

En este caso, si nuestro ordenador tiene 8 gigas de memoria RAM, reservamos 4 gigas para la MV. Ese valor se puede cambiar y lo mismo pasa con el número de CPUs.

Una vez modificados estos valores, para adaptarlos a las características de nuestra máquina, salvamos el fichero

4. INSTALACION MAQUINA VIRTUAL

Paso 3. A continuación vamos al símbolo del sistema, y desde aquí accedemos al directorio spark. Comprobamos que el fichero está aquí.

```
C:\>cd spark
C:\spark>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 5430-AFBF

Directorio de C:\spark

02/04/2023  23:23    <DIR>        .
02/04/2023  23:23    <DIR>        ..
02/04/2023  09:32                3.648 Vagrantfile
                1 archivos          3.648 bytes
                2 dirs  42.262.417.408 bytes libres

C:\spark>_
```

4. INSTALACION MAQUINA VIRTUAL

Paso 4. Definimos la variable de entorno **VAGRANT_HOME** de manera que apunte al directorio c:\spark.

Luego ejecutamos el comando **vagrant up**. Si todo va bien este comando nos descarga, instala e inicializa la máquina virtual.

```
C:\spark>set VAGRANT_HOME=c:\spark

C:\spark>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'tfpena/cursospark' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'tfpena/cursospark'
    default: URL: https://vagrantcloud.com/tfpena/cursospark
==> default: Adding box 'tfpena/cursospark' (v1.0.0) for provider: virtualbox
    default: Downloading: https://vagrantcloud.com/tfpena/boxes/cursospark/versions/1.0.0/providers/virtualbox.box
    default:
==> default: Successfully added box 'tfpena/cursospark' (v1.0.0) for 'virtualbox'!
==> default: Importing base box 'tfpena/cursospark'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'tfpena/cursospark' version '1.0.0' is up to date...
==> default: Setting the name of the VM: cursospark
Vagrant is currently configured to create VirtualBox synced folders with
the `SharedFoldersEnableSymlinksCreate` option enabled. If the Vagrant
guest is not trusted, you may want to disable this option. For more
information on this option, please refer to the VirtualBox manual:

    https://www.virtualbox.org/manual/ch04.html#sharedfolders

This option can be disabled globally with an environment variable:

    VAGRANT_DISABLE_VBOXSYMLINKCREATE=1

or on a per folder basis within the Vagrantfile:

    config.vm.synced_folder '/host/path', '/guest/path', SharedFoldersEnableSymlinksCreate: false
==> default: Clearing any previously set network interfaces...
```

4. INSTALACION MAQUINA VIRTUAL

Paso 5. La primera vez que arrancamos la maquina tarda bastante, ya que se instala localmente. Finalmente nos indica que la máquina está levantada y preparada

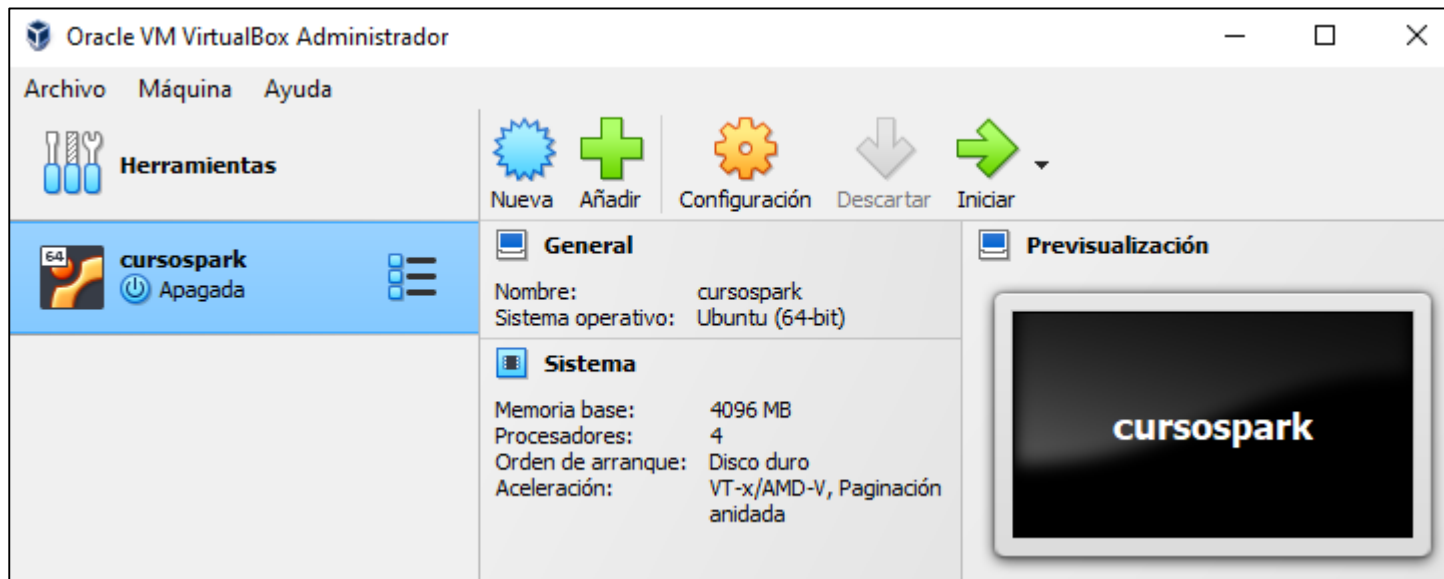
```
config.vm.synced_folder '/host/path', '/guest/path', SharedFoldersEnableSymlinksCreate: false
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 8080 (guest) => 8080 (host) (adapter 1)
    default: 8081 (guest) => 8081 (host) (adapter 1)
    default: 8085 (guest) => 8085 (host) (adapter 1)
    default: 4040 (guest) => 4040 (host) (adapter 1)
    default: 4000 (guest) => 4000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.3.36
    default: VirtualBox Version: 7.0
==> default: Mounting shared folders...
    default: /vagrant => C:/spark

C:\spark>
```


4. INSTALACION MAQUINA VIRTUAL

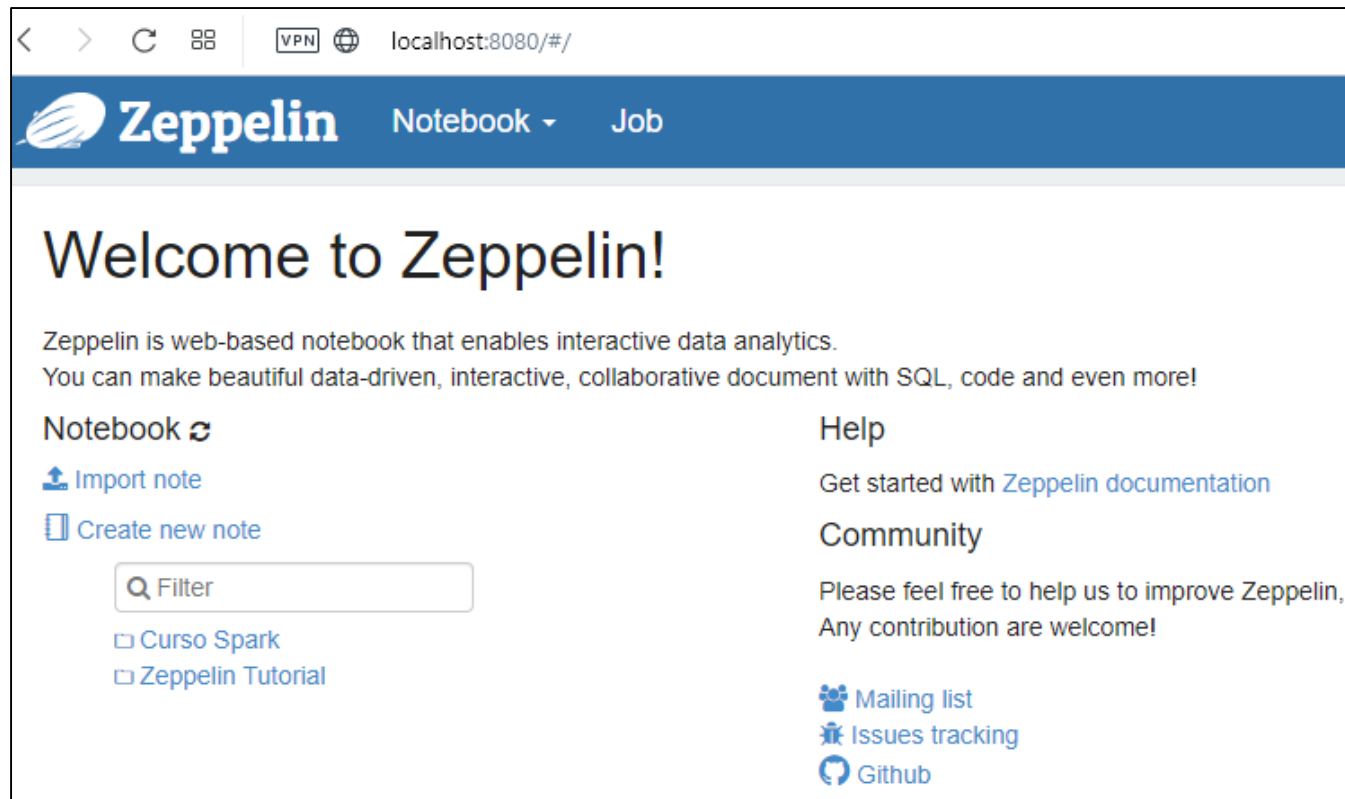
Paso 6. Con esto queda finalizada la instalación de nuestra máquina virtual y de las herramientas necesarias para visualizar el curso.

Si vamos a Virtual Box veremos que se ha creado la maquina virtual que esta gestionada mediante vagrant.



5. APACHE ZEPPELIN

Paso 1. Una vez iniciados los servicios, abrimos en un navegador la pagina localhost:8080. Aquí es donde esta escuchando la aplicación Apache Zepelín que contiene los notebooks del curso. Arranca en la pagina de bienvenida



5. APACHE ZEPPELIN

Paso 2. En los notebooks de tenemos:

- Un link al Curso Spark están los contenidos de Spark y de los lenguajes de programación
- Un link al tutorial de Apache Zeppelin donde nos presentan algunas características básicas de esta herramienta

Curso Spark

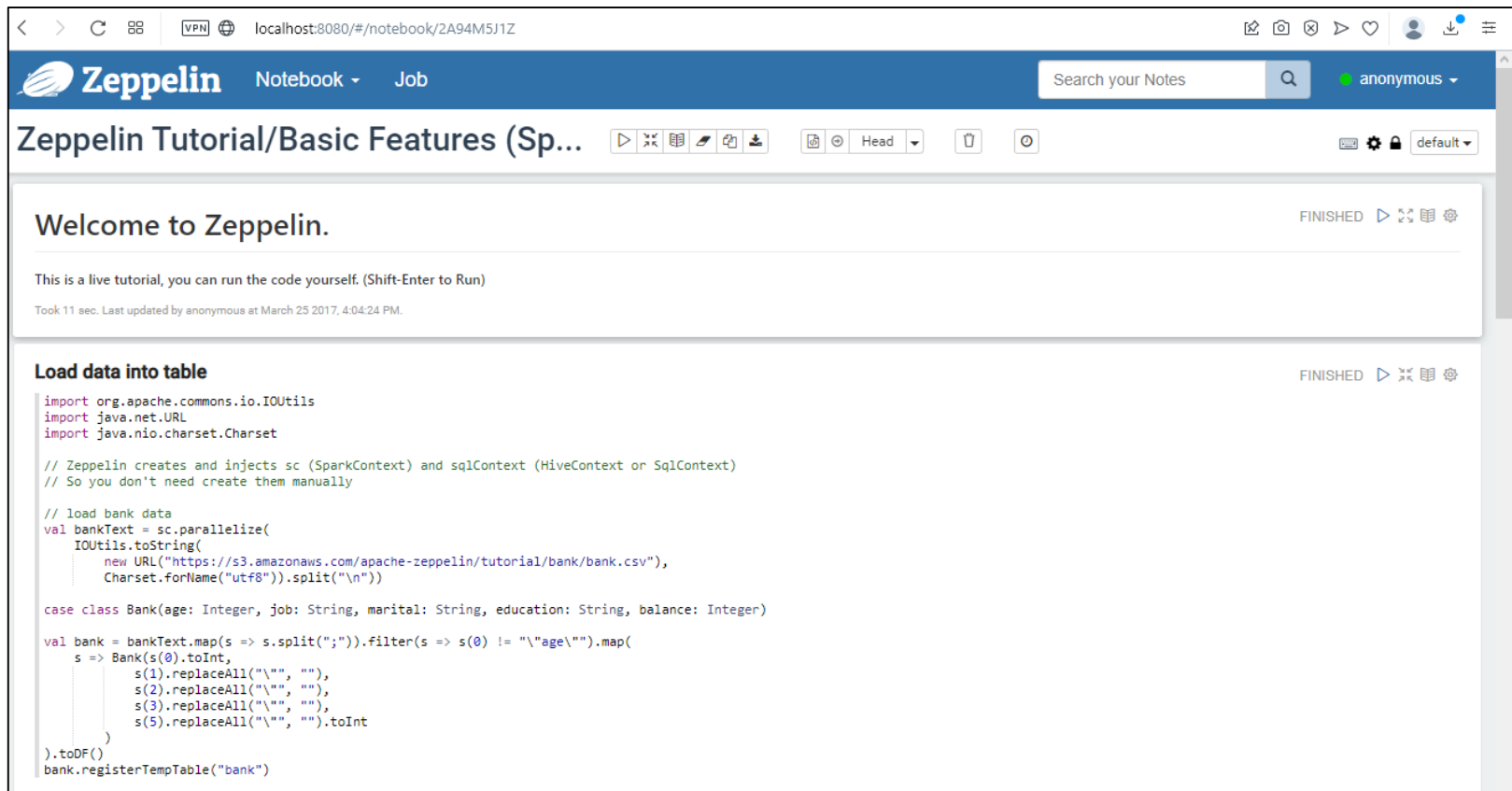
- ▢ 01 - Introducción a Apache Spark
- ▢ 02 - Introducción a los RDDs
- ▢ 03 - Principales transformaciones y acciones
- ▢ 04 - RDDs con pares clave-valor
- ▢ 05 - RDDs numéricos
- ▢ 06 - Persistencia y particionado
- ▢ 07 - Lectura y escritura de ficheros
- ▢ 08 - Ejecución de scripts
- ▢ 09 - Aspectos avanzados
- ▢ 10 - Spark SQL
- ▢ 11 - Spark Streaming
- ▢ 12 - Spark ML
- ▢ 13 - Spark GraphX

Zeppelin Tutorial

- ▢ Basic Features (Spark)
- ▢ Matplotlib (Python • PySpark)
- ▢ R (SparkR)
- ▢ Using Flink for batch processing
- ▢ Using Mahout
- ▢ Using Pig for querying data

5. APACHE ZEPPELIN

Paso 3. Si vamos a características básicas del Tutorial de Zeppelin, nos muestra un ejemplo de cómo desde Zeppelin se escribe un programa escrito en Scala, con funciones de Spark, que se puede ejecutar directamente desde el navegador.



The screenshot displays the Apache Zeppelin Notebook web interface. The browser address bar shows 'localhost:8080/#/notebook/2A94M5J1Z'. The Zeppelin header includes the logo, 'Notebook' and 'Job' tabs, a search bar, and a user profile dropdown set to 'anonymous'. The notebook title is 'Zeppelin Tutorial/Basic Features (Sp...'. The main content area shows a 'Welcome to Zeppelin.' message with a 'FINISHED' status and a 'Run' button. Below this, a code block titled 'Load data into table' is shown, also with a 'FINISHED' status and a 'Run' button. The code is written in Scala and uses Spark and IOUtils to load data from a CSV file and register it as a temporary table named 'bank'.

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

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually

// load bank data
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)

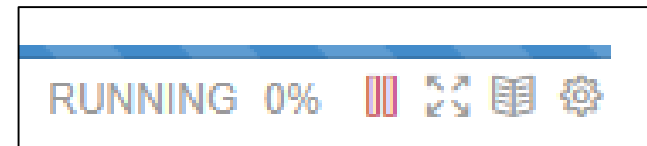
val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
  s => Bank(s(0).toInt,
    s(1).replaceAll("\"", ""),
    s(2).replaceAll("\"", ""),
    s(3).replaceAll("\"", ""),
    s(5).replaceAll("\"", "").toInt)
).toDF()
bank.registerTempTable("bank")
```

5. APACHE ZEPPELIN

Paso 4. La parte superior de texto está escrito utilizando Markdown. Para ello en cada una de esas casillas tenemos que poner simplemente %md al principio y luego utilizamos la sintaxis propia de Markdown.

```
%md
## Welcome to Zeppelin.
##### This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)
```

Una vez tengamos todo escrito le damos a Mayúsculas+Enter o a la flecha de Play para ejecutar el Markdown.



La primera vez que lo ejecutemos va a tardar un cierto rato porque necesita cargar todos los motores de Markdown, de Scala y de Spark para poder ejecutar

5. APACHE ZEPPELIN

Paso 5. En la parte inferior tenemos otra tabla, con un título y un programa escrito en Scala. Es un programa que accede a una url donde se descarga información de datos bancarios. Esos datos se paralelizan, se distribuyen entre lo que serían los nodos de nuestro cluster de Bigdata y se crea una tabla con esos datos para luego analizar la tabla utilizando SQL

Load data into table

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

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually

// load bank data
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")

import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:32
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, balance: int]
```

Took 8 sec. Last updated by anonymous at March 30 2017, 6:43:33 PM. (outdated)

5. APACHE ZEPPELIN

Paso 6. Le damos a ejecutar, nos va a tardar un cierto tiempo. La primera ejecución ya sea en Markdown, Scala o Python va a tardar bastante tiempo porque Zeppelin tiene que arrancar en la máquina virtual todos los motores para procesar estos lenguajes.

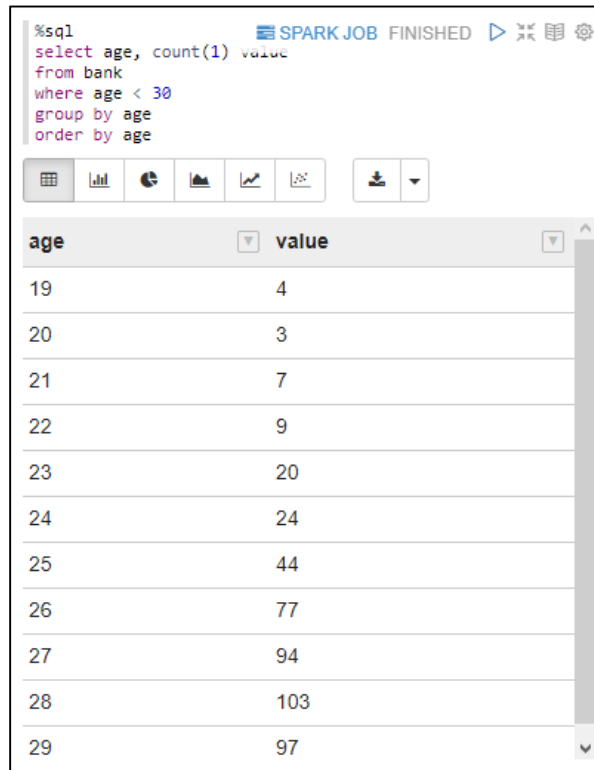
```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:32
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string, marital: string, education: string, balance: int]
```

Nos indica los pasos que ha llevado a cabo:

- ❖ ha creado una colección de datos paralela
- ❖ ha creado una tabla que denomina banca.
- ❖ Esta tabla se puede procesar después con SQL.

5. APACHE ZEPPELIN

Paso 7. En el primer ejemplo se realiza una query donde se selecciona la edad y el número de elementos de la tabla que tengan menos de 30 años. Si le damos a ejecutar tarda un rato porque la primera vez que ejecuta una aplicación SQL tiene que levantar el motor de ejecución. Una vez el trabajo ha terminado nos muestra la salida en esa tabla.



The screenshot shows the Apache Zeppelin SQL console interface. At the top, the status 'SPARK JOB FINISHED' is displayed. Below it, the SQL query is entered in a text area. The query is as follows:

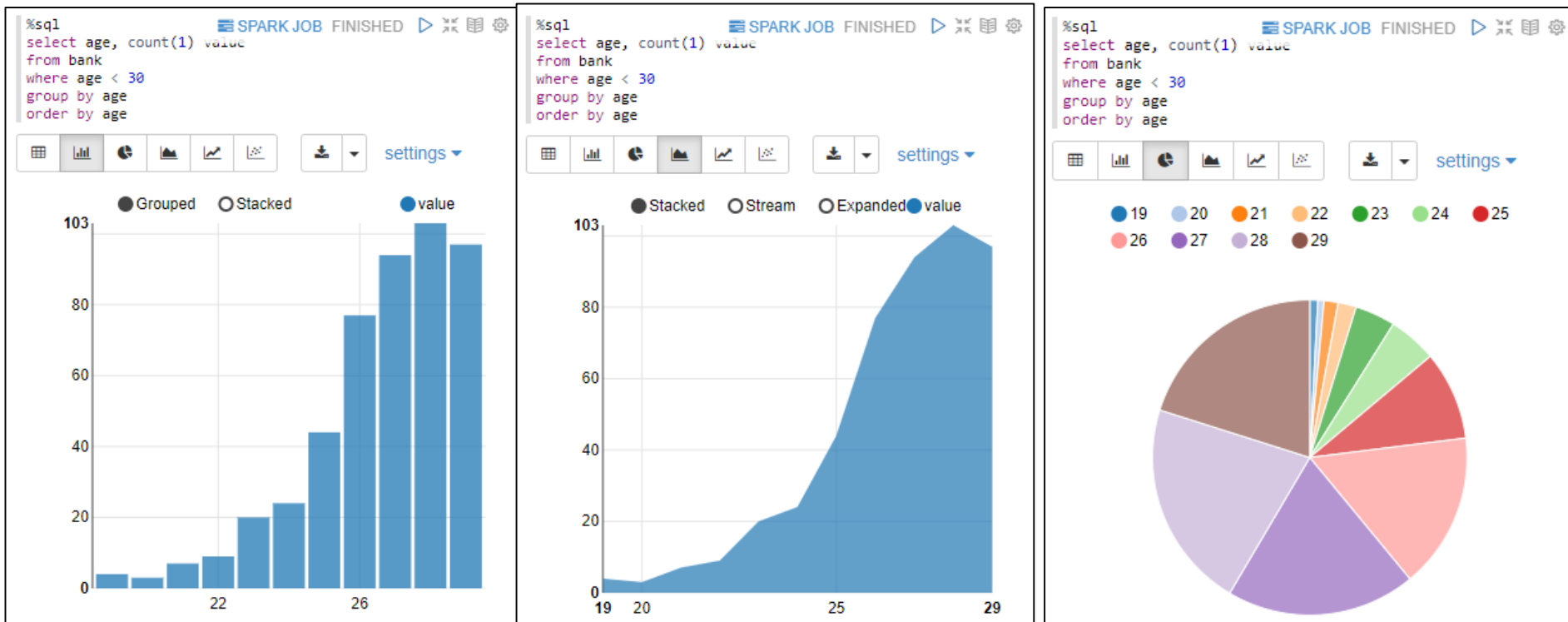
```
%sql
select age, count(1) value
from bank
where age < 30
group by age
order by age
```

Below the query, there is a toolbar with icons for table, bar chart, pie chart, line chart, and area chart. The table view is selected, and the results are displayed in a table with two columns: 'age' and 'value'.

age	value
19	4
20	3
21	7
22	9
23	20
24	24
25	44
26	77
27	94
28	103
29	97

5. APACHE ZEPPELIN

Paso 8. El propio zeppelin nos permite mostrar esa tabla de diferentes formas simplemente marcando botones. Podemos ver los mismos valores expresados con diferentes formatos



5. APACHE ZEPPELIN

Paso 9. Podemos probar otras opciones de este tutorial de Zeppelin. Por ejemplos el uso de Python y Matplotlib para crear una gráfica simple que se ve directamente en nuestro navegador.

+ Create new note

Q Filter

Curso Spark

Zeppelin Tutorial

Basic Features (Spark)

Matplotlib (Python • PySpark)

R (SparkR)

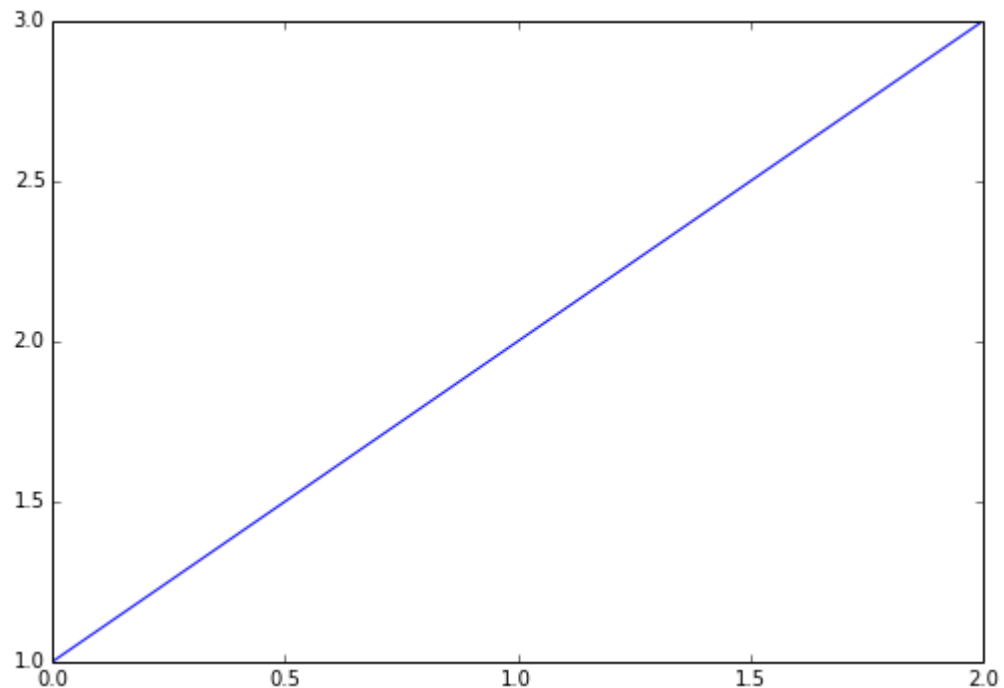
Using Flink for batch processing

Using Mahout

Using Pig for querying data

```
%python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3])
```

FINISHED ▶ ⌵ 📖 ⚙



5. APACHE ZEPPELIN

Paso 10. Podemos cerrar el navegador. Si queremos parar nuestra máquina virtual volvemos al símbolo del sistema y desde allí ejecutamos la orden **vagrant halt**: la máquina virtual no se destruye simplemente se para y podemos volver a levantarla de nuevo con un **vagrant up**

```
C:\spark>vagrant halt
==> default: Attempting graceful shutdown of VM...
C:\spark>
```

En el caso de que queramos destruir y limpiar totalmente la MV basta con ejecutar el comando **vagrant destroy** y perderemos la máquina virtual que hemos instalado.

Si queremos hacer limpieza completa basta con borrar todos los ficheros y directorios que se han instalado en la carpeta c:\spark

6. INTRODUCCION A APACHE SPARK

Arrancada Apache Zeppelin

Vamos a ver los conceptos clave de Apache Spark siguiendo el curso dentro de Apache Zeppelin. Iniciamos la máquina virtual, ejecutando el comando **vagrant up** en el directorio `c:\spark`, y abrimos el navegador en la url `localhost:8080`. Nos dirigimos a la pestaña Curso Spark/01 - Introducción a Apache Spark.

```
C:\spark>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
C:\spark>_
```

The screenshot displays the Apache Zeppelin web interface in a browser window. The address bar shows `localhost:8080/#/`. The interface has a blue header with the Zeppelin logo and navigation tabs for 'Notebook' and 'Job'. The main content area shows a notebook titled 'Curso Spark/01 - Introducción a Apache Spark'. The notebook content includes a title 'Introducción a Apache Spark' and two sections: 'Plataforma de computación cluster rápida' and 'Historia'. The 'Plataforma de computación cluster rápida' section lists features like extending MapReduce, interactive queries, streaming processing, memory support, and performance improvements. The 'Historia' section lists key milestones from 2009 to 2014, including its origin at UC Berkeley, becoming open source, and being transferred to the Apache Software Foundation. The left sidebar contains a 'Welcome to Zeppelin!' message, a description of Zeppelin as a web-based notebook, and links to 'Import note', 'Create new note', and a search filter. The right sidebar has a 'Help' section with links to 'Get started with Zeppelin doc', 'Community', and 'Please feel free to help us to Any contribution are welcome', along with links to 'Mailing list', 'Issues tracking', and 'Github'.

Zeppelin Notebook Job

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more.

Notebook

- Import note
- Create new note

Filter

- Curso Spark
- Zeppelin Tutorial

Help

Get started with Zeppelin doc

Community

Please feel free to help us to Any contribution are welcome

Mailing list

Issues tracking

Github

Curso Spark/01 - Introducción a Apache Spark

Introducción a Apache Spark

Plataforma de computación cluster rápida

- Extiende modelo MapReduce soportando de manera eficiente otros tipos de computación
 - queries interactivas
 - procesado streaming
- Soporta computaciones en memoria
- Mejora a MapReduce para aplicaciones complejas (10-20x más rápido)

Propósito general

- Modos de funcionamiento batch, interactivo o streaming
- Reduce el número de herramientas a emplear y mantener

Historia

- Iniciado en el 2009 en el UC Berkeley RAD Lab (AMPLab)
 - Motivado por la ineficiencia de MapReduce para trabajos iterativos e interactivos
- Mayores contribuidores: Databricks, Yahoo! e Intel
- Declarado open source en marzo del 2010
- Transferido a la Apache Software Foundation en junio de 2013, TLP en febrero de 2014
- Uno de los proyectos Big Data más activos
- Versión 1.0 lanzada en mayo de 2014

6. INTRODUCCION A APACHE SPARK

Plataforma de computación clúster rápida

- ❖ Apache Spark se puede definir como una plataforma de computación clúster rápida, pensada para ejecutar programas paralelos en clúster con miles de máquinas
- ❖ Extiende el modelo MapReduce introducido por Google y llevado al mundo OpenSource en el proyecto Apache Hadoop.
- ❖ MapReduce solamente soporta la computación en Batch: Un montón de datos almacenados en los nodos de nuestro clúster, con un proceso por nodo que puede llevar minutos, horas o incluso días.
- ❖ Apache Spark soporta de manera eficiente otros tipos de computación:
 - ❖ Queries interactivas usando el lenguaje SQL
 - ❖ Procesado streaming

6. INTRODUCCION A APACHE SPARK

- ❖ Spark soporta computaciones en memoria: procesa los datos a medida que se reciben sin necesidad de almacenarlos en el disco
- ❖ Esta es una de las mejoras que introduce Spark con respecto de MapReduce. Spark reduce el uso del disco que en el caso de Apache Hadoop es bastante elevado. En general se ha verificado que Apache Spark mejora MapReduce en aplicaciones complejas yendo entre 10 y 20 veces más rápido
- ❖ En resumen Spark es una plataforma de propósito general ya que incorpora diferentes modos de funcionamiento:
 - el modo tradicional en batch
 - el modo interactivo
 - el modo en streaming
- ❖ Tiene la ventaja de que reduce el número de herramientas a emplear y mantener ya que proporciona diferentes modos de funcionamiento con una única herramienta.

6. INTRODUCCION A APACHE SPARK

Historia

- ❖ El proyecto de Apache Spark se inició en Estados Unidos en el UC Berkeley RAD Lab (AMPLab) en el año 2009
- ❖ La motivación fue la ineficiencia observada en el modelo MapReduce y en Apache Hadoop, para trabajos iterativos (que utilizaban una y otra vez los mismos datos) e interactivos (que necesitaban responder rápidamente a consultas del usuario)
- ❖ Los mayores contribuidores en esta primera etapa fueron las empresas Databricks, Yahoo e Intel.
- ❖ En marzo de 2010 se declaró open source y en junio de 2013 se transfirió a la Apache Software Foundation
- ❖ Se convirtió en un TLP proyecto de primer nivel de Apache en febrero de 2014. Versión 1.0 fue lanzada en mayo de 2014
- ❖ Es uno de los proyecto Big Data más activos. Hay un montón de gente que colabora en la mejora de Apache Spark y se están lanzando continuamente nuevas versiones.

6. INTRODUCCION A APACHE SPARK

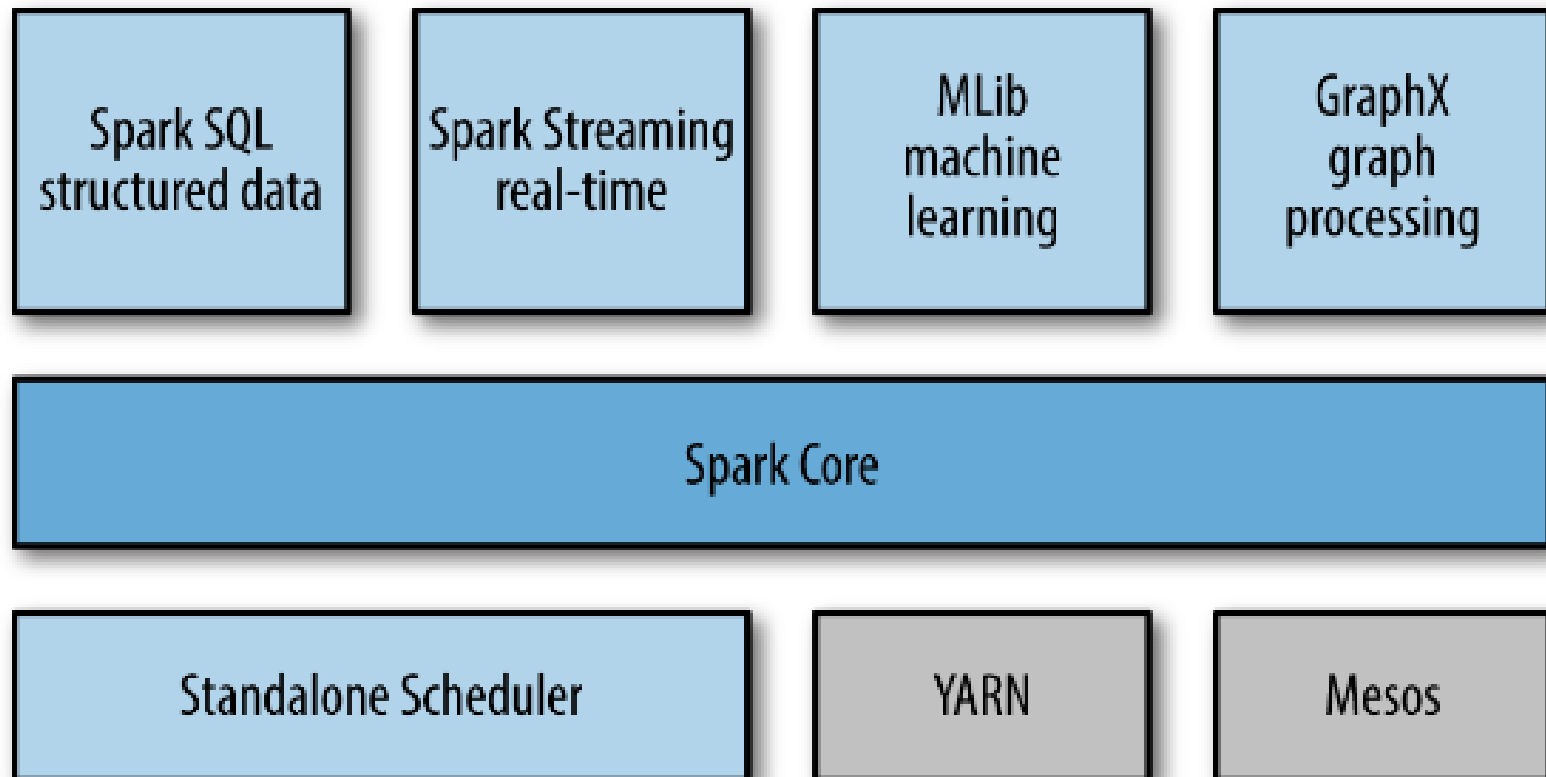
Resumen características de Spark

- ❖ Soporta una gran variedad de cargas de trabajo diferentes:
 - batch típica de MapReduce
 - consultas interactivas
 - procesamiento de datos a medida que llegan
 - procesamiento de datos en streaming
- ❖ Incorpora una librería de machine learning o de aprendizaje automático y una librería para procesamiento de grafos.
- ❖ Aunque Spark se ha programado en Scala, incorpora APIs para los lenguajes Scala, Java, Python, SQL y R (lenguaje estadístico).
- ❖ Además proporciona shells interactivos en Scala y Python que nos permiten desarrollar códigos de forma muy rápida y muy cómoda
- ❖ Se integra fácilmente con otras soluciones Big Data como es HDFS Hadoop, Cassandra una base de datos no SQL, etc

6. INTRODUCCION A APACHE SPARK

La pila Spark

- ❖ En la siguiente imagen vemos el diagrama de bloques básicos de Spark



6. INTRODUCCION A APACHE SPARK

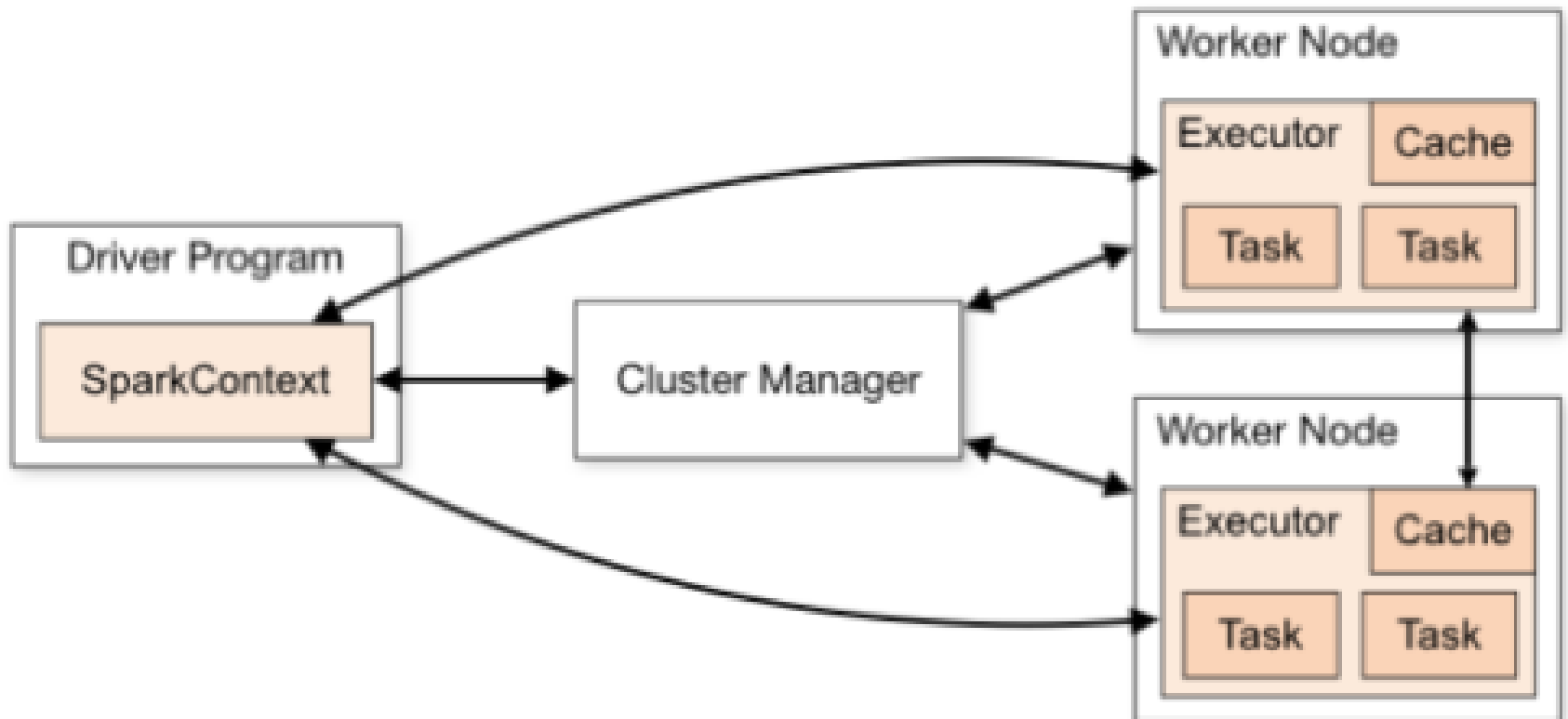
La pila Spark

- ❖ Por debajo tenemos el clúster al que Spark se conecta. Lo puede hacer a través de:
 - ❖ su propio planificador Standalone (incorporado en e propio Spark)
 - ❖ También trabaja con Yarn, el planificador de tareas que viene con Apache Hadoop
 - ❖ Apache Mesos
- ❖ El core de SPARK nos permite programarlo directamente o bien a través de librerías de alto nivel como son:
 - ❖ Spark SQL para consultas interactivas,
 - ❖ Spark Streaming para procesos en tiempo real
 - ❖ la librería de aprendizaje automático MLlib
 - ❖ la librería para el procesamiento de grafos GraphX.

6. INTRODUCCION A APACHE SPARK

Conceptos clave de Spark

- ❖ Los conceptos clave de SPARC se pueden ver en esta gráfica.



6. INTRODUCCION A APACHE SPARK

Programa Driver

- ❖ Programa escrito por el programador, que se ejecuta o bien en un nodo del cluster o bien en la propia máquina del programador
- ❖ El programador dentro de ese programa Driver tiene que crear un objeto de tipo `SparkContext`
- ❖ Este objeto va recibiendo las distintas instrucciones que le pasa el programador y va convirtiendo el programa en un conjunto de tareas definidas mediante un Grafo Dirigido Acíclico (DAG) de operaciones.
- ❖ Este Grafo Dirigido Acíclico se convierte después en un plan de ejecución físico y llegados determinados puntos del programa se lanza su ejecución a los nodos del cluster.
- ❖ El objeto `SparkContext` es el que se conecta con el Cluster Manager que puede ser Yarn, Mesos o el gestor que incorpora Apache Spark

6. INTRODUCCION A APACHE SPARK

SparkContext

- ❖ El SparkContext se encarga de planificar las tareas en los ejecutores e ir enviándolas para que las vayan ejecutando.
- ❖ Realiza la conexión con el clúster y permite construir objetos de tipo RDDs a partir de ficheros, listas u otros objetos del usuario
- ❖ En el shell de Spark, SparkContext esta definido automáticamente. Ya hay una variable denominada sc, que nos define el SparkContext

SparkContext en Python

```
from pyspark import SparkContext  
sc = SparkContext(master="local", appName="Mi app")
```

- ❖ Importamos la librería Sparkcontext y a partir de esa librería creamos nuestro objeto SparkContext especificando el tipo de gestor de clase que vamos a utilizar. Si master="local" utilizamos únicamente la máquina local para ejecutar todas las operaciones.
- ❖ Opcionalmente podemos darle un nombre a nuestra aplicación

6. INTRODUCCION A APACHE SPARK

SparkContext en Scala

- ❖ Si programamos en Scala sería algo bastante parecido simplemente cambian las importaciones.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
val conf = new SparkConf().setMaster("local").setAppName("My App")
val sc = new SparkContext(conf)
```

- ❖ Definimos un objeto conf a partir de la clase SparkConf y especificamos que se ejecute en la máquina local y el nombre a nuestra aplicación
- ❖ A partir de esta configuración definimos el SparkContext.
- ❖ Una vez creado sparkContext va lanzando distintas tareas a los ejecutores

6. INTRODUCCION A APACHE SPARK

Ejecutores

- ❖ Dentro de cada uno de esos nodos worker se está ejecutando un proceso en background llamado el Ejecutor que el que se encarga de lanzar tareas a medida que el SparkContext lo va pidiendo
- ❖ Ejecutan las tareas individuales y devuelven los resultados al Driver
- ❖ Cada ejecutor mantiene una caché en memoria, donde se almacenan los datos que utilizan las distintas tareas. En el caso de que la memoria se acabe, puede volcar parte de los datos al disco duro.

Cluster Manager

- ❖ El gestor de clúster se encarga de recibir las peticiones del SparkContext y de lanzar las tareas en los ejecutores.
- ❖ Crea diferentes tareas en los nodos worker del cluster
- ❖ Se trata de un componente enchufable en SPARK, en el sentido de que puede ser intercambiado y que Spark funciona con diferentes clusters managers: Yarn gestor por defecto de Apache Hadoop, Apache Mesos o Spark standalone, el gestor de clusters de Spark.

7. INTRODUCCION A LOS RDDs

- ❖ Accedemos al segundo enlace del curso Spark 02 - Introducción a los RDDs
- ❖ Trata sobre las estructuras de datos básicos con los que trabaja Apache Spark, denominados RDDs.
- ❖ Veremos cómo crearlos y cómo realizar operaciones simples sobre ellos.

The screenshot shows a Zeppelin Notebook interface. The browser address bar indicates the URL is `localhost:8080/#/notebook/2CBX5N42D`. The notebook title is `Curso Spark/02 - Introducción a los ...`. The content of the notebook is as follows:

RDD: Resilient Distributed Datasets

- Colección inmutable y distribuida de elementos que pueden manipularse en paralelo
- Un programa Spark opera sobre RDDs:
 - Creación de RDDs
 - Transformación de RDDs (map, filter, etc.)
 - Realización de acciones sobre RDDs para obtener resultados
- Spark automáticamente distribuye los datos y paraleliza las operaciones

Took 0 sec. Last updated by anonymous at June 09 2017, 8:46:55 PM. (outdated)

Creación de RDDs

Dos formas:

- Paralelizando una colección en el programa driver

Took 0 sec. Last updated by anonymous at June 09 2017, 8:47:02 PM. (outdated)

At the bottom, there are two code blocks, both marked as 'FINISHED':

```
%pyspark
# Ejemplo en PySpark
```

```
%spark
// Ejemplo en Scala
```


7. INTRODUCCION A LOS RDDs

RDD: Resilient Distributed Dataset

- ❖ Colección inmutable y distribuida de elementos que pueden manipularse en paralelo
- ❖ Son inmutables, una vez construido no se pueden modificar. Lo que podemos hacer es a partir de un RDD, obtener nuevos RDDs
- ❖ Una vez creado un RDD, Apache Spark automáticamente distribuye los datos que lo forman entre los nodos del clúster y ejecuta en paralelo todas las operaciones que se hacen sobre él
- ❖ Un programa Spark opera sobre un RDD:
 - Creándolo
 - Transformándolo. Transforma un RDD en otro de RDD mediante operaciones de mapeo (map), de filtrado (filter)
 - Realizando acciones sobre RDDs que producen resultados que podemos visualizar o almacenar


7. INTRODUCCION A LOS RDDs

Creación de RDDs

- ❖ Hay dos formas de crear RDDs:
 1. Creando una colección en el programa Driver y paralelizándola.
 2. Usando ficheros de entrada
- ❖ Lo veremos usando Python y Spark.

7. INTRODUCCION A LOS RDDs

a) Creando RDDs mediante colecciones usando pyspark



The screenshot shows a Jupyter Notebook interface. At the top right, it says 'RUNNING 0%' with a red progress bar and icons for expand, view, and settings. The code cell contains the following text: `%pyspark` on the first line, `# Ejemplo en PySpark` on the second line, and `rdd1 = sc.parallelize([1,2,3])` on the third line. Below the code cell, a blue progress bar is shown, and the text 'Started a few seconds ago.' is displayed.

```
%pyspark
# Ejemplo en PySpark
rdd1 = sc.parallelize([1,2,3])
```

Started a few seconds ago.

- ❖ Mediante `%pyspark` inicializamos para escribir en código pyspark.
- ❖ `sc` es el `SparkContext` creado automáticamente en el shell de spark dentro de los notebooks de Apache Zeppelin
- ❖ `rdd1` lo creamos paralelizando una lista de valores `[1,2, 3]`
- ❖ La primera vez que lo ejecutamos tarda más tiempo.
- ❖ Finalmente Apache Spark ha creado un RDD con esos datos y los ha distribuido. Pero no muestra nada porque no se ha programado

7. INTRODUCCION A LOS RDDs

- ❖ Ahora paralizaremos un array de numpy

```
%pyspark
# Ejemplo en PySpark
rdd1 = sc.parallelize([1,2,3])

import numpy as np
rdd2=sc.parallelize(np.array(range(100)))
print(rdd2.collect())
```

SPARK JOB FINISHED

- ❖ Con range(100) crea una lista de 100 números del 0 al 99. A partir de la lista crea un array, lo paraleliza y se crea el RDD
- ❖ El método collect() recupera los datos del RDD en forma de lista y se lo pasa al programa Driver que los imprime

```
%pyspark
# Ejemplo en PySpark
rdd1 = sc.parallelize([1,2,3])

import numpy as np
rdd2=sc.parallelize(np.array(range(100)))
print(rdd2.collect())
```

SPARK JOB FINISHED

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99]
```

7. INTRODUCCION A LOS RDDs

- ❖ El metodo `glom()` permite convertir un RDD en un nuevo RDD, que contiene la lista de cada una de las particiones que ha creado Apache Spark con nuestro RDD. La salida es una lista en forma de 4 listas, puesto que ha creado 4 particiones.
- ❖ El nº de particiones va a depender por defecto del nº total de cores de que disponga la CPU de nuestra MV, en este caso 4. También se puede modificar manualmente

```
%pyspark
# Ejemplo en PySpark
rdd1 = sc.parallelize([1,2,3])

import numpy as np
rdd2=sc.parallelize(np.array(range(100)))
print(rdd2.glom().collect())
```

SPARK JOB FINISHED

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24], [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4
2, 43, 44, 45, 46, 47, 48, 49], [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 6
2, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74], [75, 76, 77, 78, 79, 80, 81, 8
2, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]
```

7. INTRODUCCION A LOS RDDs

b) Creando RDDs mediante colecciones usando Scala

- ❖ Programaremos Apache Spark en Scala. Es muy parecido a Python

```
%spark
// Ejemplo en Scala
val rdd3 = sc.parallelize(List(1, 2, 3))
```

RUNNING 0%

- ❖ Con `sc SparkContext` creamos un nuevo RDD paralelizado con una lista [1,2,3].
- ❖ La primera vez tarda un poco en ejecutarse, pero no muestra nada

```
%spark
// Ejemplo en Scala
val rdd3 = sc.parallelize(List(1, 2, 3))
```

FINISHED

```
rdd3: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:30
```

7. INTRODUCCION A LOS RDDs

❖ También podemos paralelizar un rango del 1 al 100 en Scala

```
%spark SPARK JOB FINISHED  
// Ejemplo en Scala  
val rdd3 = sc.parallelize(List(1, 2, 3))  
  
val rdd4 = sc.parallelize(1 to 100)  
  
val lista = rdd4.glom().collect()  
  
rdd3: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[11] at parallelize at <console>:30  
rdd4: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:29  
lista: Array[Array[Int]] = Array(Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25), Array(26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50), Array(51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75), Array(76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100))
```

RDD4 es un array de enteros que se ha paralelizado correctamente. A partir de él se ha creado una lista cuya impresión aparece como un array de 4 subarrays que Apache Spark ha paralelizado y distribuido desde el array original de 1 hasta el 100

7. INTRODUCCION A LOS RDDs

c) Creando RDDs mediante ficheros usando pyspark

La forma de crear y paralelizar RDDs es cómoda para hacer pruebas pero en la práctica es mas común obtener los datos a partir de ficheros.

```
%pyspark
# Ejemplo en PySpark (fichero ../datos/quijote.txt)
quijote = sc.textFile("../datos/quijote.txt")
print(quijote.take(1000))
```

[u'\uffffThe Project Gutenberg EBook of Don Quijote, by Miguel de Cervantes Saavedra', u'', u'This eBook is for the use of anyone anywhere at no cost and with', u'almost no restrictions whatsoever']

- ❖ Existe un fichero almacenado en la maquina virtual, que contiene el texto de El Quijote. Creamos un RDD a partir de ese fichero mediante la función `textFile`: Lee todas las líneas del fichero y las paraleliza, es decir las distribuye automáticamente entre los distintos nodos de nuestro clúster
- ❖ Podemos ver parte de ese fichero con el método `take(1000)` que toma las mil primeras líneas del fichero que acabamos de leer

7. INTRODUCCION A LOS RDDs

d) Creando RDDs mediante ficheros usando Scala

Es Scala es muy parecido.

```
%spark FINISHED ▶ ✖ 📖 ⚙️
// Ejemplo en Scala (fichero ../datos/quijote.txt)
val quijote = sc.textFile("../datos/quijote.txt")
quijote.take(1000).foreach(println)
```

Paralelizamos las líneas de El quijote, y para cada una de las mil primeras líneas le indicamos que las imprima

7. INTRODUCCION A LOS RDDs

Particiones

Spark divide los RDDs en diferentes particiones, siguiendo diferentes criterios:

- ❖ Si Spark esta en una única maquina virtual, el nº de particiones que nos crea por defecto viene definido por el nº de núcleos o de cores de computación que tenga nuestra maquina virtual.
- ❖ Si se trata de un cluster hadoop, el nº de particiones por defecto es función del tamaño del cluster o del número de bloques del fichero (p.e. bloques HDFS). Por ejemplo si tenemos nuestro fichero almacenado en el sistema de ficheros de Hadoop, ese fichero va a estar distribuido en un conjunto de bloques y el nº de particiones que elige Spark de forma automática viene dado en función del nº de bloques del fichero
- ❖ El nº de particiones también se puede especificar manualmente en el momento de la creación de un RDD

7. INTRODUCCION A LOS RDDs

e) Creando RDDs mediante colecciones usando pyspark, con n° particiones

```
%pyspark
rdd = sc.parallelize([1,2,3,4], 2)
print(rdd.glom().collect())
print(rdd.getNumPartitions())

[[1, 2], [3, 4]]
2
```

SPARK JOB FINISHED

- ❖ Creamos un rdd con paralelize y la lista [1,2,3,4], pero ahora usamos un argumento opcional el n° de particiones, en este caso 2
- ❖ Podemos ver que ha repartido esta lista pequeña en dos sublistas: la primera contiene los dos primeros elementos y la segunda contiene los dos segundos elementos.
- ❖ También podemos ver el número de particiones que es 2

7. INTRODUCCION A LOS RDDs

f) Creando RDDs mediante colecciones usando Scala, con n° particiones

En Scala es exactamente igual

```
%spark SPARK JOB FINISHED  
val rdd = sc.parallelize(List(1,2,3,4), 2)  
println(rdd.partitions.size)  
val parti = rdd.glom().collect()  
  
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[22] at  
parallelize at <console>:29  
2  
parti: Array[Array[Int]] = Array(Array(1, 2), Array(3, 4))
```

Obtenemos el número de particiones: 2

Me indica las particiones que ha creado

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

- ❖ Las operaciones básicas que se pueden hacer sobre los RDDs son de dos tipos:

Transformaciones

- ❖ Las transformaciones convierten un RDD en otro RDD aplicando operaciones de mapeo, filtrado, etc.
- ❖ Podemos modificar un RDD de partida para obtener otro u otros con los datos interesados.

Acciones

- ❖ Las acciones permiten realizar operaciones que obtienen resultados a partir de una RDD.
- ❖ Los resultados pueden visualizarse por pantalla o almacenarse en disco.

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Transformaciones

- ❖ Son operaciones que trabajan sobre RDDs que devuelven un nuevo RDD
- ❖ Se computan de forma "perezosa" (*lazy*). Las transformaciones sobre un RDD no se hacen en el momento en que se llaman sino que se van acumulando hasta que se realiza una determinada acción que requiere obtener un resultado concreto. Entonces se hacen todas las transformaciones definidas sobre las diferentes RDDs.
- ❖ En general las transformaciones van a ejecutar una función (anónima o una función Lambda en Python) sobre cada uno de los elementos del RDD de origen.
- ❖ Eso hace que se haga operaciones totalmente paralelas e independientes, en cada uno de los nodos del clúster, sobre cada una de las particiones que tiene el RDD

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Transformación en pyspark

- ❖ Transformación sobre el RDD Quijote definido antes en pyspark

```
%pyspark
quijote = sc.textFile("../datos/quijote.txt")
quijs = quijote.filter(lambda l: "Quijote" in l)
sanchs = quijote.filter(lambda l: "Sancho" in l)
quijssancs = quijs.intersection(sanchs)
```

- ❖ Obtenemos un nuevo RDD **quijs** a partir del RDD **quijote** usando la transformación filter. Definimos una función lambda en Python, para cada uno de las líneas del fichero, devuelve las líneas que incluyen la palabra Quijote, obteniendo un nuevo RDD
- ❖ Obtenemos un nuevo RDD **sanchs** con las líneas que contengan la palabra Sancho. Y una nueva transformación que sea tener una intersección entre ellas.
- ❖ La ejecución es instantánea porque se comportan de forma perezosa todas las transformaciones. Las 3 operaciones se guardan en el DAG que crea Spark y se ejecutan más tarde cuando llamemos a una acción que necesite obtener datos a partir de estos RDDs

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Transformación en Scala

- ❖ En Spark - Scala se hace de forma similar a la Python:

```
%spark
val quijote = sc.textFile("../datos/quijote.txt")
val quijs = quijote.filter(l => l contains "Quijote")
val sanchs = quijote.filter(l => l contains "Sancho")
val quijsanacs = quijs.intersection(sanchs)
```

- ❖ Crea un RDD con la función filter, usando una función anónima de Scala, de manera que para cada línea devuelve solamente aquellas líneas que contienen la palabra Quijote
- ❖ Después crea otro RDD similar pero con aquellas líneas que contengan la palabra Sancho.
- ❖ La operación intersección es igual que antes
- ❖ La ejecución es rápida porque en realidad no se ha hecho ninguna operación. Las operaciones se hacen en el momento que ejecutemos una acción.

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Acciones sobre RDDs

Las acciones obtienen los datos de salida a partir de los RDDs

- ❖ Devuelven valores al Driver o almacenando valores en un fichero
- ❖ Fuerzan a que se realicen todas las transformaciones pendientes

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Acciones en python

- ❖ Por ejemplo, queremos saber cuántas líneas contienen simultáneamente las palabras Quijote y Sancho. Hacemos una acción count sobre el RDD y lo imprimimos.

```
%pyspark
nqs = quijsancs.count()
print("Líneas con Quijote y Sancho {}".format(nqs))
```

Líneas con Quijote y Sancho 350

- ❖ En este caso la ejecución tarda un poco más, porque ahora sí se ha realizado la operación count y se han tenido que crear los RDDs anteriores (quijote, quijs, sancs y quijsancs)
- ❖ A partir del RDD intersección quijsancs obtenemos el número de líneas que incluyen las palabras Quijote y Sancho, que son 350

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

- ❖ Si queremos imprimir 10 líneas al azar sin repetición (para ello ponemos el false) que contengan las dos palabras Sancho y Quijote:

```
%pyspark
nqs = quijsancs.count()
print("Líneas con Quijote y Sancho {}".format(nqs))
for l in quijsancs.takeSample(False,10):
    print(l)
```

SPARK JOBS FINISHED

- ❖ Tenemos que tener en cuenta que takeSample es una acción, y cada vez que se ejecuta implica que se recalculen todos los RDDs necesarios para obtener el resultado.
- ❖ En este caso el RDD quijsancs necesita que se obtengan de nuevo los RDDs Quijote y Sancho. Para cada una de esas acciones se volverían a acceder al fichero se volvería hacer las operaciones de filtrado y se volvería a hacer la operación de intersección

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Cacheado de los RDDs

- ❖ Para evitar que se hagan esas operaciones es conveniente que aquellos RDDs sobre los que vayamos a realizar diferentes acciones se cachean en memoria.

```
%pyspark
quijote = sc.textFile("../datos/quijote.txt")
quijs = quijote.filter(lambda l: "Quijote" in l)
sanchs = quijote.filter(lambda l: "Sancho" in l)
quijssancs = quijs.intersection(sanchs)
quijssancs.cache()
```

PythonRDD[91] at RDD at PythonRDD.scala:43

- ❖ Se fuerza a su almacenamiento en memoria y eso se hace poniendo el nombre del RDD y el método `cache` detrás de él.
- ❖ Cuando hacemos la primera operación de contado se guarda en memoria y la segunda operación de `takesample` ya se lee directamente de memoria sin tener que volver a recalcular de nuevo

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Acciones en Scala

- ❖ En Scala es bastante parecido

```
%spark
val nqs = quijsancs.count()
println("Líneas con Quijote y Sancho:"+nqs)

nqs: Long = 350
Líneas con Quijote y Sancho:350
```

- ❖ Si queremos ver 10 líneas sin repetición, que incluyen ambos términos Quijote y Sancho.

```
%spark
val nqs = quijsancs.count()
println("Líneas con Quijote y Sancho:"+nqs)

quijsancs.takeSample(false,10).foreach(println)

nqs: Long = 350
Líneas con Quijote y Sancho:350
-Tú me harás desesperar, Sancho -dijo don Quijote-. Ven acá, her
eje: ¿no te
```

8. INTRODUCCION A LAS TRANSFORMACIONES Y ACCIONES

Acciones en Scala

- ❖ Si queremos imprimir todas las líneas que contengan los términos Quijote y Sancho, simplemente cambiamos takeSample por collect y foreach para cada una de las líneas.

```
%spark
val nqs = quijsancs.count()
println("Líneas con Quijote y Sancho:"+nqs)

quijsancs.takeSample(false,10).foreach(println)
quijsancs.collect().foreach(println)
```

SPARK JOBS FINISHED

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

- ❖ A continuación veremos las principales transformaciones y acciones en Spark. Iniciamos la MV y dentro del Apache Zeppelin, vamos a Curso Spark/ 03 - Principales transformaciones y acciones
- ❖ Existen diferentes tipos de transformaciones que se pueden hacer sobre los RDDs
 - ❖ Transformaciones elemento a elemento que generan un nuevo RDD a partir de uno dado modificando los elementos del mismo.
 - ❖ Transformaciones sobre dos RDDs. Operaciones tipo conjunto sobre dos RDDs

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función Filter en pyspark

- ❖ Es una de las transformación mas importantes. Filtra los elementos de un RDD para obtener un nuevo RDD que contiene los elementos del original que cumplen con una cierta condición.

```
%pyspark
# Obtener los valores positivos de un rango de números
from __future__ import print_function
from test_helper import Test
rdd = sc.parallelize(xrange(-5,5))          # Rango [-5, 5)
filtered_rdd = rdd.filter(lambda x: x >= 0)  # Devuelve los positivos

Test.assertEquals(filtered_rdd.collect(), [0, 1, 2, 3, 4])

1 test passed.
```

- ❖ En este ejemplo, queremos obtener los valores positivos de un rango entre -5 y 4. La operación de filtrado, con una función lambda donde para cada elemento del RDD devuelve solamente los que sean mayor que 0, genera un nuevo RDD llamado filtered_rdd
- ❖ Con assertEquals comprobamos si el resultado es correcto, comparando que la lista obtenida debe ser igual a la lista [0,1,2,3,4]

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función Filter en Scala

- ❖ En Scala es muy parecido, cambia la definición del rango [-5, 5]

```
%spark
// Obtener los valores positivos de un rango de números
val rdd = sc.parallelize(-5 to 5)      // Rango [-5, 5]
val filtered_rdd = rdd.filter(x => x>=0) // Devuelve los positivos

assert(filtered_rdd.collect().sameElements(Array(0, 1, 2, 3, 4, 5)))

rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[27] at parallelize at <console>:30
filtered_rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[28] at filter at <console>:31
```

- ❖ La función `assert` compara los elementos del RDD filtrado con los del array [0, 1, 2, 3, 4, 5]. Si no dice nada al ejecutarlo es correcto

```
%spark
// Obtener los valores positivos de un rango de números
val rdd = sc.parallelize(-5 to 5)      // Rango [-5, 5]
val filtered_rdd = rdd.filter(x => x>=0) // Devuelve los positivos

assert(filtered_rdd.collect().sameElements(Array(0, 1, 2, 3, 4, 5,6)))

rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[29] at parallelize at <console>:30
filtered_rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[30] at filter at <console>:31
java.lang.AssertionError: assertion failed
    at scala.Predef$.assert(Predef.scala:165)
```

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función Map en Pyspark

- ❖ Map aplica una función a cada uno de los elementos de un RDD.

```
%pyspark
# Ejemplo en PySpark
# Añade 1 a cada elemento del RDD
# Para cada elemento, obtiene una tupla (x, x**2)
def add1(x):
    return(x+1)

squared_rdd = (filtered_rdd
               .map(add1)                # Añade 1 a cada elemento del RDD
               .map(lambda x: (x, x*x))) # Para cada elemento, obtén una tupla (x, x**2)

Test.assertEquals(squared_rdd.collect(), [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)])
```

- ❖ Definimos la función `add1` que coge un elemento y devuelve el elemento más uno. Aplicamos esta función al RDD `filtered_rdd` mediante un mapeo. Aplicamos otro mapeo adicional en donde a cada elemento `x` del RDD generado por el primer Map le aplicamos una función Lambda que nos devuelve una tupla de tipo `(x,x*x)`
- ❖ Si `filtered_rdd=[0, 1, 2, 3, 4]`, el primer mapeo devuelve `[1,2,3,4,5]` y el segundo mapeo `[(1,1),(2,4),(3,9),(4,16),(5,25)]` que es el valor correcto de `squared_rdd`

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Funcion Map en Scala

- ❖ En Scala solo cambia la definición de la función.

```
%spark
/* Añade 1 a cada elemento del RDD
   Para cada elemento, obtén una tupla (x, x**2) */
def add1(x: Int): Int = return(x+1)

val squared_rdd = (filtered_rdd
                  .map(add1)
                  .map(x => (x, x*x)))

assert(squared_rdd.collect().sameElements(Array((1,1), (2,4), (3,9), (4,16), (5,25), (6,36))))

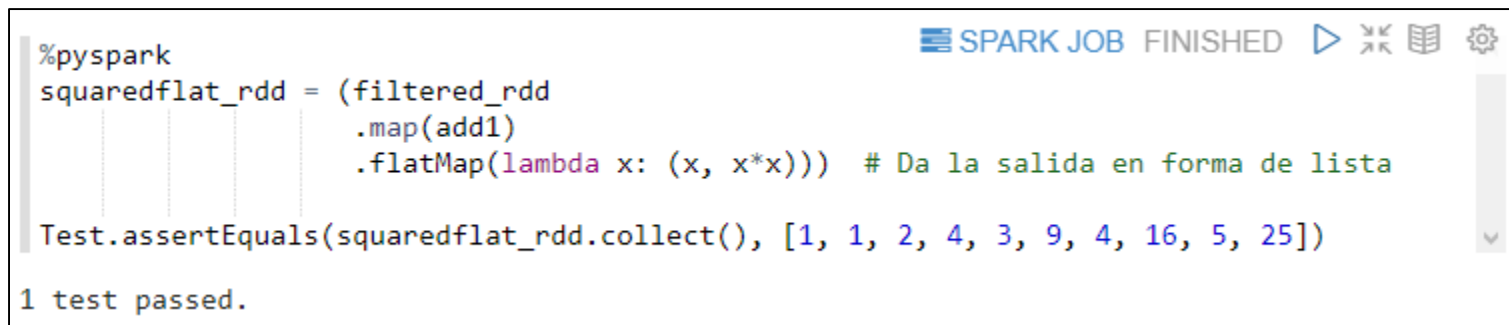
add1: (x: Int)Int
squared_rdd: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[34] at map at <console>:37
```

- ❖ En este caso la función add1 la definimos como un parámetro entero X que devuelve un entero X+1.
- ❖ Al RDD filtrado le mapeamos la función add1, y una función lambda que para cada elemento X generado por el mapa anterior devuelve la tupla (X, x*X)
- ❖ Como aquí el RDD filtrado era [0,1,2,3,4,5], el 1º mapeado resulta [1,2,3,4,5,6] y el segundo [(1,1),(2,4),(3,9),(4,16),(5,25),(6,36)]

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Funcion flatMap en Python

- ❖ Es una variante del Map que funciona igual que el MAP pero la salida la da en forma aplanada: si está formada por una lista de lista o arrays de arrays lo convierte en una lista simple.



The screenshot shows a Jupyter Notebook interface with a PySpark kernel. The code defines a new RDD, `squaredflat_rdd`, by applying `.map(add1)` followed by `.flatMap(lambda x: (x, x*x))` to a `filtered_rdd`. A comment indicates this produces a flat list. A test assertion follows, and the output shows the test passed.

```
%pyspark
squaredflat_rdd = (filtered_rdd
                  .map(add1)
                  .flatMap(lambda x: (x, x*x))) # Da la salida en forma de lista

Test.assertEquals(squaredflat_rdd.collect(), [1, 1, 2, 4, 3, 9, 4, 16, 5, 25])

1 test passed.
```

SPARK JOB FINISHED

- ❖ El mismo código anterior en Python, cambiando el segundo map por `flatMap`. La salida en vez de ser una lista de tuplas, resulta una lista simple

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función flatMap en Scala

- ❖ Igual podemos hacer en scala substituyendo el map por el flatMap. De nuevo la salida nos lo da como un array simple

```
%spark FINISHED ▶ ⌵ 📖 ⚙  
  
val squaredflat_rdd = (filtered_rdd  
    .map(add1)  
    .flatMap(x => List(x, x*x))) // Da la salida en forma de lista  
assert(squaredflat_rdd.collect().sameElements(Array(1, 1, 2, 4, 3, 9, 4, 16, 5, 25, 6, 36)))
```

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función sample

- ❖ **sample(withReplacement, fraction, seed=None)**
devuelve un RDD que contiene una muestra/subconjunto de los elementos del RDD original
- ❖ En la llamada sample tenemos que especificar tres parámetros:
 - ❖ withReplacement - si True, cada elemento puede aparecer varias veces en la muestra, si false solo puede aparecer 1 vez
 - ❖ fraction - Nos permite especificar el tamaño esperado de la muestra como una fracción del tamaño del RDD original. Depende de lo que hayamos especificado en witReplacement.
 - ❖ sin reemplazo (false): fraction nos indica probabilidad de seleccionar un elemento, su valor debe estar entre [0, 1]
 - ❖ con reemplazo (true): fraction indica número esperado de veces que se escoge un elemento, su valor debe ser ≥ 0
 - ❖ seed - semilla para el generador de números aleatorios

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función sample en pyspark

```
%pyspark
srdd1 = squaredflat_rdd.sample(False, 0.5)
srdd2 = squaredflat_rdd.sample(True, 2)
srdd3 = squaredflat_rdd.sample(False, 0.8, 14)
print('s1={0}\ns2={1}\ns3={2}'.format(srdd1.collect(), srdd2.collect(),
    srdd3.collect()))

s1=[1, 3, 4, 5]
s2=[1, 1, 1, 1, 1, 1, 2, 4, 4, 3, 3, 3, 9, 4, 16, 16, 25, 25, 25]
s3=[1, 1, 2, 4, 3, 9, 4, 16, 25]
```

- ❖ Partimos del RDD squaredflat [1, 1, 2, 4, 3, 9, 4, 16, 5, 25]
- ❖ 1º ejemplo: Tomamos muestra de esa lista sin reemplazo y con una probabilidad de escoger cada uno de sus elementos de 0.5. Como depende de cómo se generan los números aleatorios, nos devuelve una lista de 4 elementos [1, 3, 4, 5]. La mitad probabilidad de escoger un elemento o no

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función sample en pyspark

- ❖ 2º ejemplo: Tomamos muestra de esa lista con reemplazo, donde el nº esperado de veces que se escoge cada uno de los elementos es 2. Hay elementos repetidos, el 3 lo coge 3 veces, el 16 lo coge 2 veces, hay algún elemento que lo coge más de 2 veces y el nº de elementos debería de ser el doble del nº de elementos de la lista original.
- ❖ 3º ejemplo: Es igual al primero, sin reemplazo pero con una probabilidad de escoger cada uno de los elementos de la lista de 0.8. No deberían de aparecer elementos repetidos, aparecen repetidos el 1 y el 4 porque ya están repetidos en el RDD original y el nº de elementos que se toman es mayor que en el primer caso ya que la probabilidad que hemos puesto es más alta.

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función sample en Scala

❖ En Scala el resultado será parecido.

```
%spark
val s1 = squaredflat_rdd.sample(false, 0.5).collect()
val s2 = squaredflat_rdd.sample(true, 2).collect()
val s3 = squaredflat_rdd.sample(false, 0.8).collect()
```

SPARK JOBS FINISHED

```
s1: Array[Int] = Array(1, 4, 3, 4, 6, 36)
```

```
s2: Array[Int] = Array(1, 1, 1, 1, 1, 4, 4, 4, 3, 16, 16, 5, 5, 5, 5, 5, 25, 25, 25, 36, 36)
```

```
s3: Array[Int] = Array(1, 1, 4, 3, 9, 4, 16, 5, 25, 36)
```

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función distinct

- ❖ La función distinct, a partir de un RDD original devuelve un nuevo RDD eliminando los elementos que estén duplicados
- ❖ El orden de la salida no está definido, es decir el orden en el que aparecen los elementos en el nuevo RDD puede ser cualquiera

Función distinct en Python

- ❖ Si partimos del squaredflat, tiene repetidos los elementos 1 y 4, nos quedamos solamente con todos aquellos elementos que sean distintos del original. El orden que devuelve puede ser cualquiera.

```
%pyspark
distinct_rdd = squaredflat_rdd.distinct()
print(distinct_rdd.collect())
```

[16, 4, 1, 25, 5, 9, 2, 3]

SPARK JOB FINISHED

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función distinct en Scala

- ❖ En Scala también llamamos al método distinct. Lo recogemos y llamamos a la función mkstring para que lo convierta en un string y permita imprimirlos.
- ❖ El orden que nos devuelve es diferente del caso de Python

```
%spark
val distinct_rdd = squaredflat_rdd.distinct()
println(distinct_rdd.collect().mkString(" "))

distinct_rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[55] at distinct at <console>:37
4 16 36 25 1 9 5 6 2 3
```

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función groupBy

- ❖ La función groupby nos devuelve un RDD en donde los datos se agrupan en formato clave valor y la clave se obtiene mediante una función que especifica el usuario.

Función groupBy en python

```
%pyspark
grouped_rdd = distinct_rdd.groupBy(lambda x: x%3)
print(grouped_rdd.collect())
print([(x,sorted(y)) for (x,y) in grouped_rdd.collect()])

[(0, <pyspark.resultiterable.ResultIterable object at 0x7fcea977290>), (1, <pyspark.resultiterable.ResultIterable object at 0x7fcea9b4910>), (2, <pyspark.resultiterable.ResultIterable object at 0x7fcea9b4450>)]
[(0, [3, 9]), (1, [1, 4, 16, 25]), (2, [2, 5])]
```

- ❖ Partimos del distinct_RDD anterior, y los agruparemos en pares clave-valor donde la clave se obtiene con la función lambda para cada x nos devuelve su módulo 3. Por tanto hay 3 claves definidas 0, 1 y 2

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función groupBy en python

- ❖ Para cada una de estas claves, el valor es un iterable de pyspark que lo podemos visualizar mediante el ultimo print
- ❖ Imprimimos una nueva lista con el par $\langle x, \text{sorted}(y) \rangle$ obtenida a partir de los valores x e y del RDD `grouped_rdd.collect()`
- ❖ Obtenemos 3 y 9 cuyo modulo 3 es igual a 0, los valores 1 4 16 y 25 cuyo modulo 3 es igual a 1 y los valores 2 y 5 cuyo módulo 3 es igual a 2.
- ❖ Mediante groupBy agrupamos y obtenemos a partir del RDD original, un RDD en donde cada uno de estos elementos es una tupla con una clave obtenida mediante la función modulo 3 y los valores son una lista de todos los elementos de la RDD original que están asociados a esa clave

9. PRINCIPALES TRANSFORMACIONES Y ACCIONES

Función groupBy en Scala

- ❖ Podemos hacer lo mismo en Scala. Al `distinct_rdd` le aplico la función `groupBy` al RDD en función de módulo 3
- ❖ Luego se hace un `foreach` donde para cada clave 0, 1 y 2, se imprimen los elementos asociados dentro de un paréntesis
 - ❖ Clave 0 con los valores 36 9 6 y 3
 - ❖ Clave 1 con los valores 4 16 25 1
 - ❖ Clave 2 con los valores 5 y 2.

```
%spark
val grouped_rdd = distinct_rdd.groupBy(x => x%3)
val grouped = grouped_rdd.collect()
grouped.foreach(f => println(f._1+" (" +f._2.mkString(" ")+" ")

grouped_rdd: org.apache.spark.rdd.RDD[(Int, Iterable[Int])] = ShuffledRDD[75] at groupBy at <console>:39
grouped: Array[(Int, Iterable[Int])] = Array((0,CompactBuffer(36, 9, 6, 3)), (1,CompactBuffer(4, 16, 25, 1)), (2,CompactBuffer(5, 2)))
0 (36 9 6 3)
1 (4 16 25 1)
2 (5 2)
```

SPARK JOB FINISHED

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

- ❖ Veremos un conjunto pequeño de transformaciones que utilizan dos RDDs de partida
- ❖ Hacen operaciones de tipo conjunto como por ejemplo uniones, intersecciones y diferencias entre los datos de dos RDDs, generando un tercer RDD con los datos de partida

Función unión en pyspark

- ❖ Devuelve un RDD con los datos de los dos de partida.

```
%pyspark
rdda = sc.parallelize(['a', 'b', 'c'])
rddb = sc.parallelize(['c', 'd', 'e'])
rddu = rdda.union(rddb)
Test.assertEquals(rddu.collect(),['a', 'b', 'c', 'c', 'd', 'e'])

1 test passed.
```

SPARK JOB FINISHED

- ❖ Tenemos dos redes paralelizando dos listas sencillas {a,b,c} {c,d,e}
- ❖ hacemos la unión y simplemente comprobamos RRDu tiene que dar igual a esta lista {a,b,c,c,d,e}
- ❖ El test muestra que la aserción es correcta

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

Función unión en Scala

- ❖ En Scala es exactamente igual

```
%spark
val rdda = sc.parallelize(List('a', 'b', 'c'))
val rddb = sc.parallelize(List('c', 'd', 'e'))
val rddu = rdda.union(rddb)
assert(rddu.collect().sameElements(Array('a', 'b', 'c', 'c', 'd', 'e')))
```

rdda: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[15] at parallelize at <console>:29
rddb: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[16] at parallelize at <console>:29
rddu: org.apache.spark.rdd.RDD[Char] = UnionRDD[17] at union at <console>:33

- ❖ Generamos dos RDDs a partir de dos listas
- ❖ Hacemos la unión y comprobamos que los elementos del RDD Unión son los mismos que los del array {a, b, c, c, d, e}
- ❖ No nos da ningún error, eso quiere decir que la aserción fue correcta.

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

Función Intersección en pyspark

- ❖ Igual que tenemos el método para obtener la unión de dos RDDs también tenemos la intersección de dos RDDs que devuelve un RDD con los datos comunes a ambos

```
%pyspark
rdda = sc.parallelize(['a', 'b', 'c'])
rddb = sc.parallelize(['c', 'd', 'e'])
rddi = rdda.intersection(rddb)
Test.assertEquals(rddi.collect(),['c'])

1 test passed.
```

SPARK JOB FINISHED

- ❖ En Python nos devuelve un RDD que sólo contiene el elemento c

Funcion Intserseccion en Scala

```
%spark
val rdda = sc.parallelize(List('a', 'b', 'c'))
val rddb = sc.parallelize(List('c', 'd', 'e'))
val rddi = rdda.intersection(rddb)
assert(rddi.collect().sameElements(Array('c')))
```

```
rdda: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[33] at parallelize at <console>:29
rddb: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[34] at parallelize at <console>:29
rddi: org.apache.spark.rdd.RDD[Char] = MapPartitionsRDD[40] at intersection at <console>:33
```

SPARK JOB FINISHED

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

Función subtract en pyspark

- ❖ La transformación subtract devuelve un RDD con los datos del primer RDD menos los datos del segundo RDD

```
%pyspark
rdda = sc.parallelize(['a', 'b', 'c'])
rddb = sc.parallelize(['c', 'd', 'e'])
rdds = rdda.subtract(rddb)
Test.assertEquals(rdds.collect(), ['a', 'b'])

1 test passed.
```

- ❖ Si restamos los datos del primero {a,b,c} menos los datos del segundo {c,d,e} nos debería devolver un RDD con los datos {a,b}

Funcion Subtract en Scala

```
%spark
val rdda = sc.parallelize(List('a', 'b', 'c'))
val rddb = sc.parallelize(List('c', 'd', 'e'))
val rdds = rdda.subtract(rddb)
assert(rdds.collect().sameElements(Array('a', 'b')))
```

rdda: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[59] at parallelize at <console>:29
rddb: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[60] at parallelize at <console>:29
rdds: org.apache.spark.rdd.RDD[Char] = MapPartitionsRDD[64] at subtract at <console>:33

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

Función cartesiana

- ❖ La función cartesian devuelve un RDD con el producto cartesiano de ambos RDDs (operación muy costosa)
- ❖ Al hacer el producto cartesiano de RDDa y de RDDb, devuelve un RDD donde cada elemento de A se combina con cada uno de los elementos B [(a,c),(a,d),(a,e),(b,c),(b,d),(b,e),(c,c), (c,d), (c,e)]

```
%pyspark
rddc = rdda.cartesian(rddb)
Test.assertEquals(rddc.collect(),
    [('a','c'),('a','d'),('a','e'),('b','c'),('b','d'),('b','e'),('c','c'),
     ('c','d'),('c','e')])

1 test passed.
```

En pyspark

```
%spark
val rddc = rdda.cartesian(rddb)
assert(rddc.collect().sameElements(
    Array(('a','c'),('a','d'),('a','e'),('b','c'),('b','d'),('b','e'),
          ('c','c'),('c','d'),('c','e'))))

rddc: org.apache.spark.rdd.RDD[(Char, Char)] = CartesianRDD[66] at cartesian at <console>:33
```

En Scala

10. TRANSFORMACIONES SOBRE DOS RDDS SIMPLES

Función cartesiana

- ❖ Esta operación es muy costosa ya que si pensamos en RDDs reales distribuidos a lo largo de todo el clúster, estamos haciendo que cada uno de los elementos del RDDa se tiene que copiar en donde están todos los elementos del RDDb
- ❖ Es una operación que supone mucho tiempo en cuanto a comunicaciones, por lo que es recomendable no utilizar este tipo de método
- ❖ Siempre que sea posible es mejor utilizar métodos de tipo reducción, donde las comunicaciones son más pequeñas: primero se hace una reducción a nivel de cada partición internamente en cada uno de los nodos de nuestro cluster y las comunicaciones son solamente de los elementos ya previamente reducidos en vez de ser de todos los elementos del original

11. EJERCICIO WORDCOUNT

- ❖ Realizar en pyspark el programa wordcount, que cuente el número de veces que aparece cada palabra en el texto del Quijote.