

# **Capítulo 8**

## ***Big Data 2.0 - Spark***

**Al finalizar el capítulo, el alumno podrá:**

- Describir la tecnología Spark
- Implementar un cluster Spark
- Conocerá los fundamentos del lenguaje Scala y Python para ciencia de datos.

### **Temas**

1. Introducción a Spark
2. Propósitos de Spark
3. Componentes
4. Instalación y configuración
5. Scala y Python

# 1. Introducción a Spark

## Introducción a Spark

Apache Spark es una infraestructura de cluster de código abierto usado con frecuencia para cargas de trabajo de Big Data. 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. 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 Machine Learning.

8 - 4

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.



**Apache Spark** es una framework de computación en cluster open-source. Fue desarrollada originariamente en la Universidad de California, en el AMPLab de Berkeley. El código base del proyecto Spark fue donado más tarde a la Apache Software Foundation que se encarga de su mantenimiento desde entonces. Spark proporciona una interfaz para la programación de clusters completos con Paralelismo de Datos implícito y tolerancia a fallos.

Apache Spark se puede considerar un sistema de computación en cluster de propósito general y orientado a la velocidad. Proporciona APIs en Java, Scala, Python y R. También proporciona un motor optimizado que soporta la ejecución de grafos en general. También soporta un conjunto extenso y rico de herramientas de alto nivel entre las que se incluyen Spark SQL (para el procesamiento de datos estructurados basada en SQL), MLlib para implementar machine learning, GraphX para el procesamiento de grafos y Spark Streaming.

Apache Spark tiene la base de su arquitectura en el llamado RDD o Resilient Distributed DataSet que es un multiset de solo lectura de ítems de datos distribuidos a lo largo de un cluster de máquinas que se mantiene en un entorno tolerante a fallos.<sup>7</sup>

En Spark 1.x, RDD era la API principal pero con el desarrollo de Spark 2.0, se recomienda la utilización de la API DataSet.<sup>8</sup> Aunque la API RDD no se considera descatalogada,<sup>9</sup> la tecnología RDD todavía está presente en la base de la API DataSet.<sup>11</sup>

Spark y sus RDDs fueron desarrollados en 2012 en respuesta a las limitaciones del paradigma de computación en cluster MapReduce que fuerza a la utilización de una estructura lineal dataflow en particular en los programas distribuidos: Los programas basados en MapReduce leen los datos de entrada desde disco, que mapea una función a lo largo de los datos, reduce los resultados del mapa y almacena los resultados de la reducción en disco.

Los RDDs de Spark funcionan como un working set para los programas distribuidos que ofrecen una forma (deliberadamente) restringida de la memoria compartida distribuida.<sup>12</sup>

Spark facilita la implementación de tanto algoritmos iterativos que visitan su dataset muchas veces en un mismo bucle tanto como el análisis de datos interactivo/exploratorio.

La latencia de estas aplicaciones se puede reducir en varios órdenes de magnitud en comparación con la implementación basada en MapReduce

Más allá de los algoritmos de tipo iterativos, están los algoritmos de entrenamiento para sistemas de machine learning, que constituyeron el impulso inicial del desarrollo de Apache Spark.

Apache Spark requiere de un cluster manager y un sistema de almacenamiento distribuido. Para la gestión del cluster, Spark soporta las opciones siguientes:

- Spark Standalone (Cluster Spark Nativo)
- Hadoop YARN
- Apache Mesos

Para el almacenamiento distribuido, Spark presenta interfaces hacia una gran variedad de plataformas:

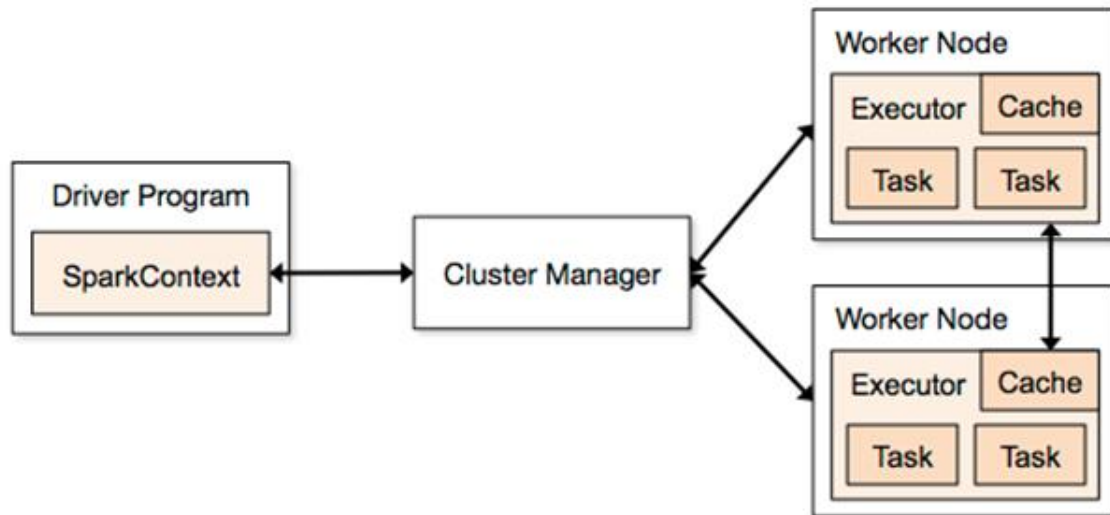
- Hadoop Distributed File System (HDFS)<sup>16</sup>
- MapR File System (MapR-FS)<sup>17</sup>
- Cassandra<sup>18</sup>
- OpenStack Swift
- Amazon S3
- Kudu
- Incluso soporta una solución personalizada.

Spark también soporta un modo local pseudo-distribuido, normalmente utilizado solamente para pruebas o en entornos de desarrollo donde el almacenamiento distribuido no es obligatorio y se puede usar el sistema de archivos local; en un escenario como este, Spark se ejecuta en una única máquina con un *executor* por cada core de CPU.

## Arquitectura Spark

- Las aplicaciones de Spark son ejecutadas independientemente y estas son coordinadas por el objeto Spark `SparkContext` del programa principal (Driver Program).
- `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 crear ejecutores (executors), encargados de ejecutar tareas (tasks) en los nodos del clúster.



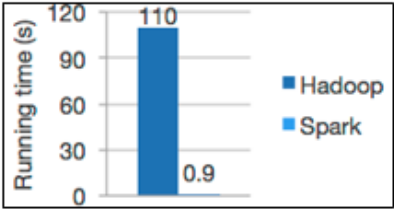
- Cada aplicación posee sus propios ejecutores, los cuales ejecutan tareas en varios subprocesos. Gracias a esto, se consigue aislar las aplicaciones entre sí, tanto en el lado de la programación (cada controlador programa sus propias tareas), como en el lado del ejecutor ( las tareas de las diferentes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

## 2. Propósitos de Spark

### Propósitos de Spark

#### Velocidad

- Ejecute programas hasta 100 veces más rápido que Hadoop MapReduce en la memoria, o 10 veces más rápido en el disco.
- Apache Spark tiene un motor de ejecución DAG avanzado que admite flujo de datos a cíclico y cómputo en memoria.



Framework	Running time (s)
Hadoop	110
Spark	0.9

8 - 7

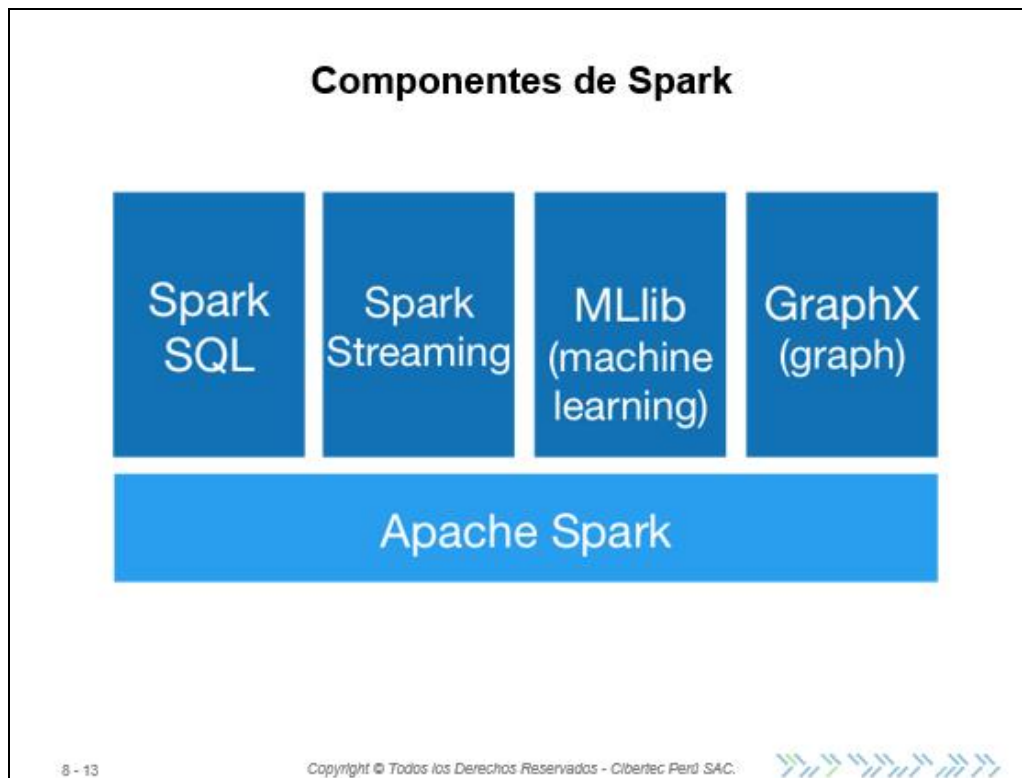
Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

Entre los principales propósitos de Spark tenemos:

- **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.
- **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.

### 3. 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”.

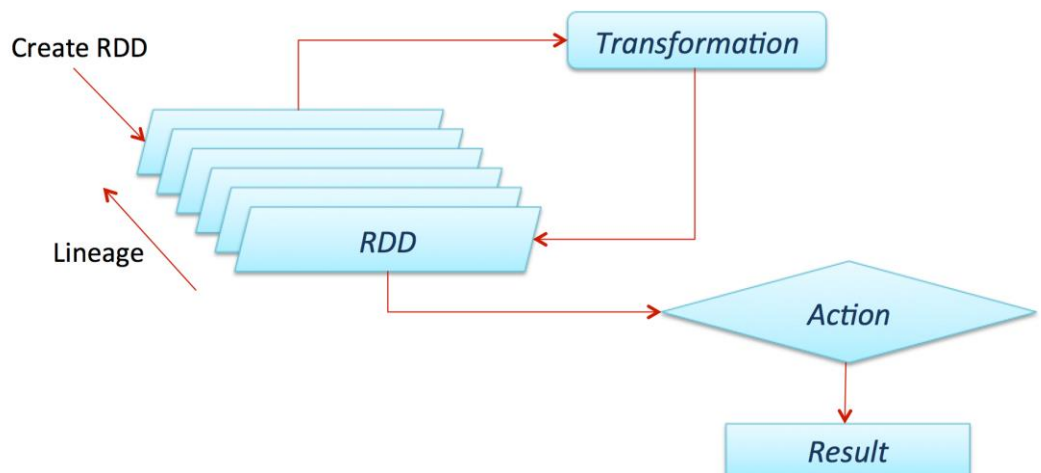
Como se puede observar en la siguiente ilustración se muestra los diferentes componentes de Spark:

#### Apache Spark

- El motor base para el procesamiento en escala y distribuido
- Aunque está construido en Scala, hay APIs para Python, Java y R.
- Se encarga entre otras cosas de:
  - Gestión de la memoria
  - Recuperación ante fallos
  - Planificación, distribución de trabajos en el cluster
  - Monitorizar trabajo
  - Accedes a los sistemas de almacenamiento

## RDD (Resilient Distributed Dataset)

- La principal abstracción de datos en Spark es el RDD (Resilient Distributed Dataset)
- Un RDD representa una colección de items que pueden ser distribuidos en los nodos de computo
- Los APIs disponibles para trabajar con RDDs son Java, Python y Scala
- Operaciones en RDD:
  - Transformaciones: Crea un nuevo RDD aplicando una transformación a un RDD existente
  - Acciones: Retorna los resultados al driver o a un archivo de salida

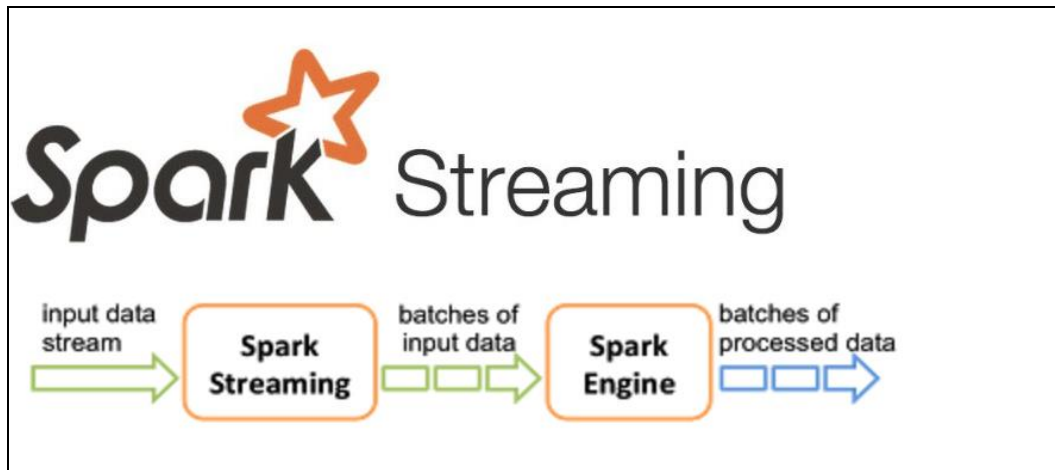


- La principal abstracción de datos en Spark es el RDD (Resilient Distributed Dataset)
- Un RDD representa una colección de items que pueden ser distribuidos en los nodos de computo
- Los APIs disponibles para trabajar con RDDs son Java, Python y Scala
- Operaciones en RDD:
  - Transformaciones: Crea un nuevo RDD aplicando una transformación a un RDD existente
  - Acciones: Retorna los resultados al driver o a un archivo de salida

## Spark Streaming

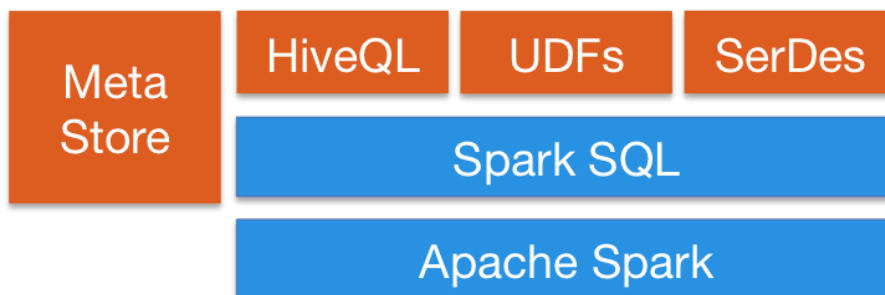
- Se usa para procesar fuentes de datos en tiempo real (streaming data)
- Permite procesar con una alta tolerancia a fallos y un gran rendimiento las fuentes “vivas” de información que le suministremos
- Su unidad fundamental de trabajo es el Dstream (serie de RDDs, que veremos posteriormente)





### Spark SQL

- Permite integrar comandos y componentes relacionales junto con la programación funcional de Spark
- Podemos usar SQL o Hive Query Language
- Permite el acceso a múltiples fuentes de datos
- Permite el acceso por JDBC o ODBC

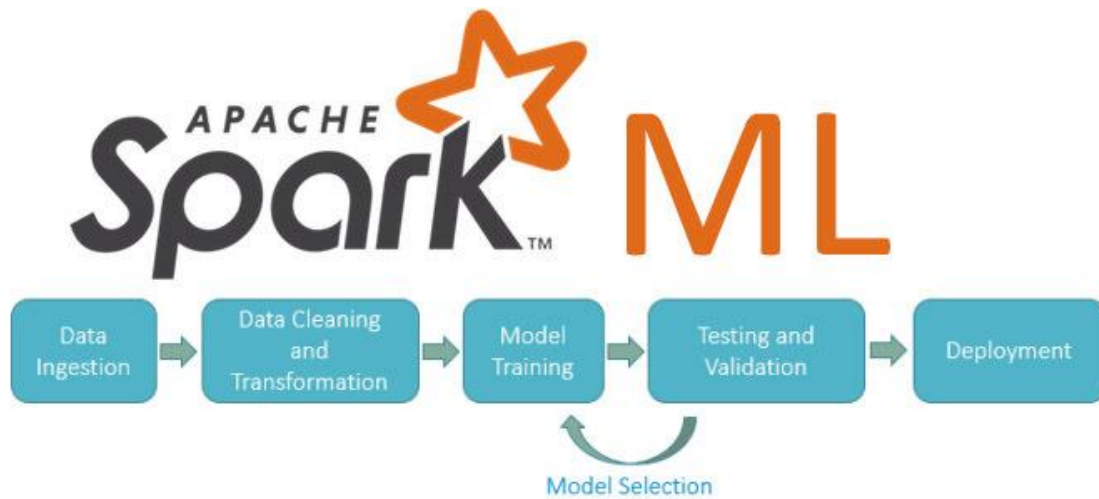


### Spark GraphX

- GraphX es el API para procesamiento paralelo en grafos.
- Spark GraphX implementa Resilient Distributed Graph (RDG- una abstracción de los RDD's).
- RDG's asocia registros con los vértices y bordes de un grafo. Sin embargo, se pueden seguir viendo como colecciones tradicionales de RDD
- Se dispone de una gran cantidad de algoritmos preparados, que permiten agilizar el proceso de construcción de aplicaciones y mejora el rendimiento y velocidad

## Spark MLlib

- Se dispone de una variedad de algoritmos y otros procesos como “data cleaning”
- Por ejemplo, clasificación, clustering, regression, extracción etc...
- Permite su ejecución sobre HDFS, HBase, etc...



## 4. Instalación y Configuración de Spark

### Instalación y configuración Spark

<https://spark.apache.org/downloads.html>



8 - 21 Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

Se dispone de 3 tipos de paquetes:

- Prebuild for Apache Hadoop (Incluye Hadoop)
- Prebuild with user-provided (Sin Hadoop)
- Source (Compilar las fuentes)

El proceso deberá crearse un usuario:

```
# useradd spark
```

Ir a la carpeta /opt y descomprimir:

```
# cd /opt  
# tar -xzf spark-2.3.0-bin-hadoop2.7.tgz
```

Cambiar el nombre de la carpeta

```
# mv spark-2.3.0-bin-hadoop2.7 spark
```

Brindar los permisos al usuario spark

```
# chown -R spark:spark spark
```

Ingresar con el usuario Spark

```
$ su -l spark  
$ cd /opt/spark
```

Ejecutar las variables del entorno

```
export JAVA_HOME=/usr/java/jdk1.8.0_161/jre  
export SPARK_HOME=/opt/spark  
export PATH=$SPARK_HOME/bin:$PATH
```

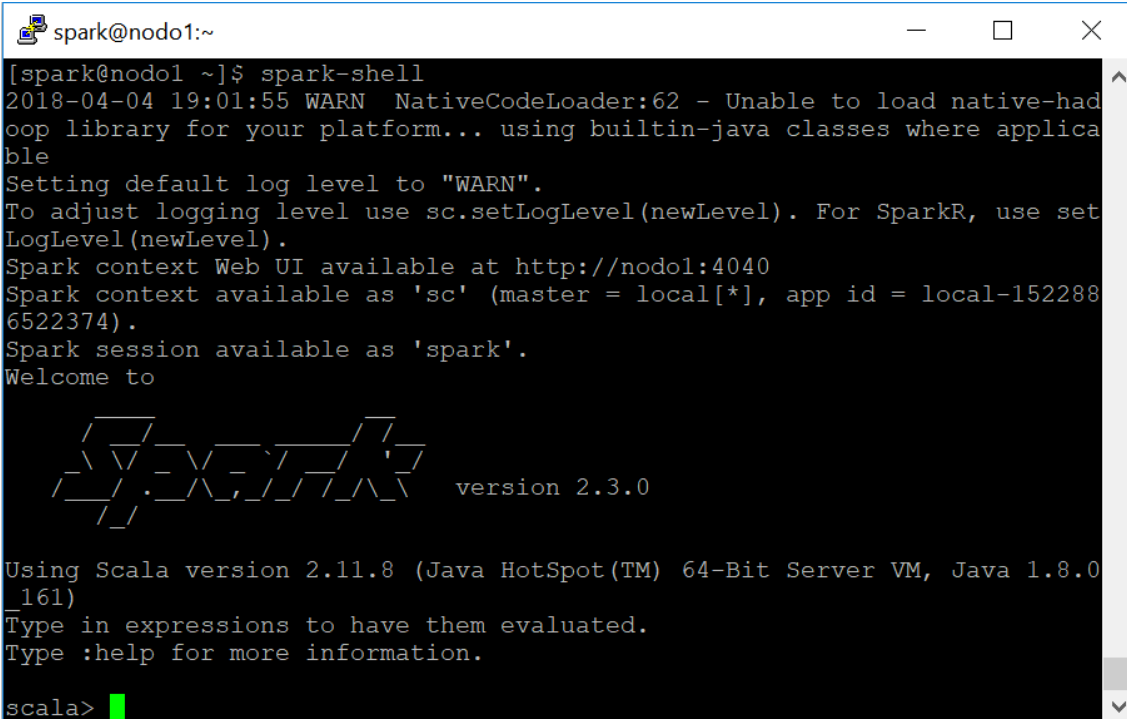
```
$ cat .bashrc  
# .bashrc  
export JAVA_HOME=/usr/java/jdk1.8.0_161/jre  
export SPARK_HOME=/opt/spark  
export PATH=$SPARK_HOME/bin:$PATH
```

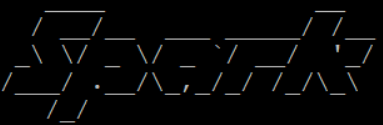
Cargar las variables en el nuevo entorno

```
$ source .bashrc
```

Ejecutar el comando

```
$ spark-shell
```



```
spark@nodo1:~  
[spark@nodo1 ~]$ spark-shell  
2018-04-04 19:01:55 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
Spark context Web UI available at http://nodo1:4040  
Spark context available as 'sc' (master = local[*], app id = local-1522886522374).  
Spark session available as 'spark'.  
Welcome to  
 version 2.3.0  
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_161)  
Type in expressions to have them evaluated.  
Type :help for more information.  
scala>
```

## 5. Scala y Python

### Scala

```
$ spark-shell

scala> val textFile = spark.read.textFile("README.md")

scala> textFile.count() // Number of items in this Dataset

scala> textFile.first() // First item in this Dataset

scala> val linesWithSpark = textFile.filter(line =>
line.contains("Spark"))

scala> textFile.filter(line => line.contains("Spark")).count() // How
many lines contain "Spark"?
```

8 - 24

Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.



### Iniciando con el Shell de Spark

El shell de Spark proporciona una forma simple de aprender la API, así como una poderosa herramienta para analizar datos de forma interactiva. Está disponible en Scala (que se ejecuta en la máquina virtual de Java y, por lo tanto, es una buena forma de utilizar las bibliotecas de Java existentes) o en Python.

Comience ejecutando lo siguiente en el directorio Spark con Scala:

```
./bin/spark-shell
```

La abstracción primaria de Spark es una colección distribuida de elementos llamada Dataset. Los conjuntos de datos se pueden crear desde Hadoop InputFormats (como archivos HDFS) o transformando otros Datasets. Hagamos un nuevo conjunto de datos del texto del archivo README en el directorio fuente de Spark:

```
scala> val textFile = spark.read.textFile("README.md")
textFile: org.apache.spark.sql.Dataset[String] = [value: string]
```

Puede obtener valores del Dataset directamente, llamando a algunas acciones o transformar el Dataset para obtener una nueva. Para obtener más información, lea el documento API.

```
scala> textFile.count() // Number of items in this Dataset
res0: Long = 126 // May be different from yours as README.md will change over
time, similar to other outputs

scala> textFile.first() // First item in this Dataset
res1: String = # Apache Spark
```

Ahora transformaremos este conjunto de datos en uno nuevo. Llamamos filtro para devolver un nuevo conjunto de datos con un subconjunto de los elementos en el archivo.

```
scala> val lineswithSpark = textFile.filter(line => line.contains("Spark"))
lineswithSpark: org.apache.spark.sql.Dataset[String] = [value: string]
```

Podemos encadenar transformaciones y acciones:

```
scala> textFile.filter(line => line.contains("Spark")).count() // How many
lines contain "Spark"?
res3: Long = 15
```

### **Más sobre las operaciones del conjunto de datos**

Las acciones y transformaciones del conjunto de datos se pueden usar para cálculos más complejos. Digamos que queremos encontrar la línea con más palabras:

```
scala> textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b)
a else b)
res4: Long = 15
```

Esto primero asigna una línea a un valor entero, creando un nuevo Dataset. reduce en ese Dataset para encontrar el mayor conteo de palabras. Los argumentos para mapear y reducir son literales de función de Scala (cierres) y pueden usar cualquier función de idioma o biblioteca de Scala / Java. Por ejemplo, podemos llamar fácilmente a funciones declaradas en otro lugar. Usaremos la función Math.max () para hacer que este código sea más fácil de entender:

```
scala> import java.lang.Math
import java.lang.Math

scala> textFile.map(line => line.split(" ").size).reduce((a, b) => Math.max(a,
b))
res5: Int = 15
```

Un patrón de flujo de datos común es MapReduce, popularizado por Hadoop. Spark puede implementar flujos de MapReduce fácilmente:

```
scala> val wordCounts = textFile.flatMap(line => line.split("
")).groupByKey(identity).count()
wordCounts: org.apache.spark.sql.Dataset[(String, Long)] = [value: string,
count(1): bigint]
```

Aquí, llamamos a flatMap para transformar un Dataset de líneas a un Dataset de palabras, y luego combinamos groupByKey y contamos para calcular los recuentos por palabra en el archivo como un Dataset de pares (String, Long). Para recopilar los conteos de palabras en nuestro shell, podemos llamar collect:

```
scala> wordCounts.collect()
res6: Array[(String, Int)] = Array((means,1), (under,2), (this,3),
(Because,1), (Python,2), (agree,1), (cluster.,1), ...)
```

Comience ejecutando lo siguiente en el directorio Spark con Python:

```
./bin/pyspark
```

La abstracción primaria de Spark es una colección distribuida de elementos llamada Dataset. Los conjuntos de datos se pueden crear desde Hadoop InputFormats (como archivos HDFS) o transformando otros Datasets. Debido a la naturaleza dinámica de Python, no necesitamos que el Dataset esté fuertemente tipado en Python. Como resultado, todos los conjuntos de datos en Python son Dataset [Row], y lo llamamos DataFrame para que sea coherente con el concepto de marco de datos en Pandas y R. Hagamos un nuevo DataFrame a partir del texto del archivo README en el directorio fuente de Spark:

```
>>> textFile = spark.read.text("README.md")
```

Puede obtener valores de DataFrame directamente, llamando a algunas acciones o transformar el DataFrame para obtener uno nuevo. Para obtener más información, lea el documento API.

```
>>> textFile.count() # Number of rows in this DataFrame
126

>>> textFile.first() # First row in this DataFrame
Row(value=u'# Apache Spark')
```

Ahora transformaremos este DataFrame en uno nuevo. Llamamos a filtro para devolver un nuevo DataFrame con un subconjunto de las líneas en el archivo.

```
>>> linesWithSpark = textFile.filter(textFile.value.contains("Spark"))
```

Podemos encadenar transformaciones y acciones:

```
>>> textFile.filter(textFile.value.contains("Spark")).count() # How many lines contain "Spark"?
15
```

**Bibliografía:**

“Quick Start”

<http://spark.apache.org/docs/latest/quick-start.html>

Fecha: 01 de diciembre del 2017

“Spark Standalone Mode”

<http://spark.apache.org/docs/latest/spark-standalone.html>

Fecha: 01 de diciembre del 2017

“Spark Overview”

<http://spark.apache.org/docs/latest/index.html>

Fecha: 01 de diciembre del 2017

“Apache Spark”

[https://es.wikipedia.org/wiki/Apache\\_Spark](https://es.wikipedia.org/wiki/Apache_Spark)

Fecha: 01 de diciembre del 2017

“Apache Spark”

[https://es.wikipedia.org/wiki/Apache\\_Spark](https://es.wikipedia.org/wiki/Apache_Spark)

Fecha: 01 de diciembre del 2017

“APACHE SPARK - Big Data”

[http://informatica.gonzalonazareno.org/proyectos/2016-17/Apache\\_spark-alejandro\\_palomino.pdf](http://informatica.gonzalonazareno.org/proyectos/2016-17/Apache_spark-alejandro_palomino.pdf)

Fecha: 01 de diciembre del 2017