

# **Procesamiento de Datos en Spark**

Curso académico 2017/18

# Presentación

- "Data Engineer" en Audiense.com
- 12 años de experiencia
  - Information Retrieval / motores búsqueda
  - NLP & Analítica Redes Sociales
  - Big Data & Cloud

[fjavieralba@gmail.com](mailto:fjavieralba@gmail.com)



# Planificación

- **Bloque 1**
  - Apache Spark: Introducción
  - Diferencias con Hadoop / MapReduce
  - Operaciones básicas de procesado de datos
- **Bloque 2**
  - DataFrames
  - SparkSQL
  - Lectura y escritura de ficheros
  - Integración con otras tecnologías Big Data
  - Gestión de Memoria

# Planificación

- **Bloque 3**
  - RDDs
  - Integración con DataFrames
- **Bloque 4**
  - GraphX & Graphframes
  - Spark Streaming
  - MLlib

# Evaluación

- 100% de la nota: PECs (Pruebas Evaluación Continua)
- 3 tareas diferentes, basadas en contenidos y ejercicios de clase

# Bibliografía

- Spark Programming Guides:
  - <https://spark.apache.org/docs/latest/#where-to-go-from-here>
- Learning Pyspark:
  - <http://proquest.safaribooksonline.com/9781786463708>
- Spark: The Definitive Guide

# Máquina Virtual

- Instrucciones de descarga e instalación en el Aula Virtual
- Credenciales:
  - Usuario: spark
  - Password: spark



**¿Qué es Apache Spark?**

# Introducción

- **Motor de Computación Unificado**
- Conjunto de **Librerías** (batch, streaming, SQL, ML...)
- **Procesamiento paralelo** en clústeres de servidores
- Soporta varios lenguajes de programación (Python, Java, Scala, R)
- Puede ejecutarse desde en un portátil hasta en un clúster de miles de servidores

Structured  
Streaming

Advanced  
Analytics

Libraries &  
Ecosystem

Structured APIs

Datasets

DataFrames

SQL

Low level APIs

RDDs

Distributed Variables



# Filosofía

# Motor de Computación

- No almacenamiento
- Centrado en **procesamiento de los datos**
- Integración con sistemas de almacenamiento
  - Cloud FS (S3, Azure)
  - HDFS
  - NoSQL (Cassandra, MongoDB)
  - Search (Elasticsearch, SolR)
  - Streams (Kafka, Kinesis)

# Unificado

- Tradicionalmente, ha sido necesario utilizar diferentes tecnologías para el procesamiento paralelo de datos:
  - Extracción: Flume, Sqoop, Logstash, Fluentd...
  - Transformación: Python, tecnologías propietarias
  - Batch Analytics: DWH (Teradata, Redshift), Hive, Pig, Hadoop MR
  - Realtime Analytics: Storm, CEP
- Spark propone una plataforma unificada para escribir aplicaciones Big Data

# Unificado

- Tiene herramientas para:
  - Carga de datos e integración con sistemas externos
  - Transformaciones y procesamiento desde diferentes lenguajes
  - Consultas SQL
  - Machine Learning
  - Procesamiento de “streams”
- Traducidas al mismo motor de procesamiento subyacente => optimización automática
- Podemos “mezclar” todas estas herramientas en una sola app

# Librerías

- Diferentes librerías construídas sobre “Spark core”
- Son la parte más activa del proyecto Open Source:
  - SparkSQL (SQL y datos estructurados)
  - MLlib (Machine Learning)
  - SparkStreaming (Procesamiento de “Streams”)
  - GraphX (Analíticas sobre grafos)
- Muchas más librerías opensource de terceros ([spark-packages.org](http://spark-packages.org))

# Historia



# Historia

- 2009: Proyecto de investigación en UC Berkeley (AMPLab)
- 2010: Publicación del paper (*Matei Zaharia, Mosharaf Chowdhury, Michael Franklin, Scott Shenker, Ion Stoica*)
- Hadoop Mapreduce era el estándar para el procesamiento en paralelo en clústers de cientos de nodos
- Hadoop permitió crear innovadoras aplicaciones Big Data, pero era ineficiente para problemas iterativos (ML, Grafos)
- Spark basó su API en la programación funcional e implementó el motor de procesamiento para que soportara el almacenamiento de datos en memoria para poder hacer iteraciones (varias pasadas) sobre los datos más eficientemente

# Historia

- Inicialmente sólo se soportaban aplicaciones batch, pero pronto se vio que era posible resolver consultas "ad hoc" y realizar tareas interactivas de ML: REPL Scala
- 2011: Shark (primer motor SQL de Spark)
- Se empiezan a añadir diferentes librerías construidas sobre Spark Core. Primeras versiones de MLlib, SparkStreaming y GraphX.
- 2013: Amplia difusión. 30+ organizaciones contribuyen al código. AMPLab cede Spark a Apache Software Foundation y lanzan la empresa Databricks
- 2014: Spark 1.0 (Spark SQL, datos estructurados)
- 2016: Spark 2.0 (DataFrames, StructuredStreaming, ML Pipelines...)

# Diferencias con Hadoop / MR

- No hay capa de **almacenamiento** en Spark
- No se basa en MapReduce, sino que tiene su propio motor de procesamiento distribuído.
- Es capaz de mantener grandes cantidades de datos **en memoria**, entre diferentes “jobs”. Esto le permite tener un rendimiento mucho mayor, ya que en Hadoop/MapReduce, los datos siempre se leen/escriben a disco.
- Dos tipos de aplicaciones que aprovechan esto son:
  - Algoritmos Iterativos (ML, grafos)
  - Análisis interactivo (consultas ad-hoc, exploración de datos)

# Diferencias con Hadoop / MR

- En vez de basar todo el procesamiento en fases Map y Reduce, el motor de Spark construye un grafo acíclico dirigido (DAG) de operadores de procesamiento
- Dicho grafo de procesamiento es mucho más general que sólo las fases Map y Reduce, y esto hace que un procesamiento mucho más complejo pueda ser expresado en un solo “job” Spark (En MapReduce serían necesarios muchos “jobs” para realizar el mismo procesamiento)
- Proporciona una consola interactiva, muy útil para pruebas y exploración de conjuntos de datos

# Pyspark

# pyspark

- Ejercicio de Clase 1 (EC1):
  - toma de contacto con la shell pyspark
  - Crear SparkSession
  - Leer datos de un fichero JSON
  - Operaciones básicas
  - Interacción básica con spark SQL

# pyspark

spark@spark-vm:~\$ pyspark

```
Welcome to  
Python 2.7.13 |Anaconda 2.5.0 (x86_64)| (default, Feb  7 2018 13:48:35)  
[GCC 4.2.1 Compatible Clang 3.6.2 (tags/RELEASE_36-branch)]  
ipython version 2.2.1  
  
Using Python version 3.6.4 (default, Feb  7 2018 13:48:35)  
SparkSession available as 'spark'.  
>>> from pyspark.sql import SparkSession  
>>> spark = SparkSession.builder.appName("spark intro").getOrCreate()  
>>>  
```

# pyspark

```
>>> df = spark.read.json("uah-spark/bloque1/ec1/people.json")
>>> df.show()
+---+---+
| age| name|
+---+---+
| null|Michael|
| 30| Andy|
| 19| Justin|
+---+---+
>>> ■
```

# pyspark

```
>>> df.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

# pyspark

```
>>> df.select("name").show()
+-----+
|    name|
+-----+
|Michael|
|   Andy|
| Justin|
+-----+
>>> █
```

# pyspark

```
>>> df.count()  
3
```

# pyspark

```
>>> df.filter(df['age'] > 21).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

# pyspark

```
>>> df.groupBy("age").count().show()
+----+-----+
| age|count|
+----+-----+
|  19|     1|
| null|     1|
|  30|     1|
+----+-----+
```

# pyspark

```
>>> df.createOrReplaceTempView("people")
```

```
>>> spark.sql("show tables").show()
+-----+-----+-----+
|database|tableName|isTemporary|
+-----+-----+-----+
|        |    people|      true|
+-----+-----+-----+
```

# pyspark

```
>>> sqlDF = spark.sql("SELECT * from people")
>>> sqlDF.show()
+---+----+
| age| name|
+---+----+
| null|Michael|
| 30 | Andy |
| 19 | Justin|
+---+----+
```

# pyspark

```
>>> sqlDF2 = spark.read.table("people")
>>> sqlDF2.show()
+---+----+
| age| name|
+---+----+
| null|Michael|
| 30| Andy|
| 19| Justin|
+---+----+
```

# pyspark

- Ejercicio de Clase 2 (EC2):
  - crear DataFrame desde datos en memoria
  - Algunas “queries” sencillas usando la api de DataFrames
  - Realizar las mismas queries utilizando spark SQL
  - Primera toma de contacto con consola spark-sql

# pyspark

spark@spark-vm:~\$ pyspark

# pyspark

```
>>>  
>>> data = [(123, 'juan', 1000), (456, 'ana', 1200), (789, 'pedro', 1700)]  
>>>
```

```
>>> df1 = spark.createDataFrame(data)  
>>>
```

# pyspark

```
>>> df1
DataFrame[_1: bigint, _2: string, _3: bigint]
```

```
>>> df1.show()
+---+---+---+
| _1| _2| _3|
+---+---+---+
|123| juan|1000|
|456| ana |1200|
|789| pedro|1700|
+---+---+---+
```

# pyspark

```
>>> df2 = spark.createDataFrame(data).toDF("id", "nombre", "sueldo")
>>>
>>> df2.show()
+---+-----+-----+
| id|nombre|sueldo|
+---+-----+-----+
|123| juan| 1000|
|456| ana| 1200|
|789| pedro| 1700|
+---+-----+-----+
```

# pyspark

```
>>> df2.filter("sueldo < 1200").show()
```

# pyspark

```
>>> df2.filter("sueldo < 1200").show()
+---+-----+-----+
| id|nombre|sueldo|
+---+-----+-----+
|123| juan| 1000|
+---+-----+-----+
```

# pyspark

```
>>> df2.filter("sueldo > 1000").show()
+---+-----+-----+
| id|nombre|sueldo|
+---+-----+-----+
|456|  ana|  1200|
|789| pedro|  1700|
+---+-----+-----+
```

# pyspark

```
>>> df2.count()  
3
```

```
>>> df2.where("sueldo > 1000").show()  
+---+-----+-----+  
| id|nombre|sueldo|  
+---+-----+-----+  
| 456|   ana|   1200|  
| 789| pedro|   1700|  
+---+-----+-----+
```

# pyspark

```
>>> from pyspark.sql.functions import *
>>>
>>> df2.select(avg("sueldo")).show()
+-----+
|avg(sueldo)|
+-----+
|      1300.0|
+-----+
```

# pyspark

```
>>> df2.createOrReplaceTempView("empleados_tmp")  
>>>
```

# pyspark

```
>>> spark.sql("show tables").show()
+-----+-----+-----+
|database|  tableName|isTemporary|
+-----+-----+-----+
|          |empleados_tmp|      true|
+-----+-----+-----+
```

# pyspark

```
>>> spark.sql("select * from empleados_tmp where sueldo > 1000").show()
+---+-----+-----+
| id|nombre|sueldo|
+---+-----+-----+
|456|  ana|  1200|
|789| pedro|  1700|
+---+-----+-----+
```

# pyspark

```
>>> df_resultado = spark.sql("select * from empleados_tmp where sueldo > 1000")
>>> df_resultado.show()
+---+-----+-----+
| id|nombre|sueldo|
+---+-----+-----+
|456|   ana|   1200|
|789| pedro|   1700|
+---+-----+-----+
```

# pyspark

```
>>> spark.sql("select nombre from empleados_tmp where sueldo < 1200").show()
+-----+
|nombre|
+-----+
|  juan|
+-----+
```

# pyspark

```
>>> spark.sql("select avg(sueldo) from empleados_tmp").show()  
+-----+  
|avg(sueldo)|  
+-----+  
|      1300.0|  
+-----+
```

# pyspark

```
>>> df2.write.saveAsTable("empleados")
>>>
>>> spark.sql("show tables").show()
+-----+-----+-----+
|database|  tableName|isTemporary|
+-----+-----+-----+
| default|    empleados|      false|
|          |empleados_tmp|       true|
+-----+-----+-----+
```

# spark-sql

```
spark@spark-vm:~$ spark-sql
```

# spark-sql

```
spark-sql> show tables;
18/02/12 18:49:10 INFO SparkSqlParser: Parsing command: show tables
18/02/12 18:49:13 INFO HiveMetaStore: 0: get_database: default
18/02/12 18:49:13 INFO audit: ugi=spark ip=unknown-ip-addr      cmd=get_database
: default
18/02/12 18:49:13 INFO HiveMetaStore: 0: get_database: default
18/02/12 18:49:13 INFO audit: ugi=spark ip=unknown-ip-addr      cmd=get_database
: default
18/02/12 18:49:13 INFO HiveMetaStore: 0: get_tables: db=default pat=*
18/02/12 18:49:13 INFO audit: ugi=spark ip=unknown-ip-addr      cmd=get_tables:
db=default pat=*
18/02/12 18:49:14 INFO CodeGenerator: Code generated in 432.527895 ms
default empleados      false
Time taken: 3.467 seconds, Fetched 1 row(s)
18/02/12 18:49:14 INFO CliDriver: Time taken: 3.467 seconds, Fetched 1 row(s)
spark-sql> █
```

# spark-sql

```
spark-sql> select * from empleados where sueldo > 1000;
```

```
:376) finished in 1.039 s
18/02/12 18:49:57 INFO DAGScheduler: Job 0 finished: processCmd at CliDriver.jav
a:376, took 1.601075 s
456      ana     1200
789      pedro   1700
Total: 2 rows
```

# spark-sql

```
spark-sql> select avg(sueldo) from empleados;
```

```
a:376, took 0.758058 s  
1300.0
```

# Arquitectura de Spark (intro)

# Arquitectura

- Objetivo: coordinar un conjunto de máquinas para que puedan realizar conjuntamente tareas de procesamiento de datos en paralelo
- Cluster Manager: software que gestiona y asigna los recursos de un clúster de máquinas (Spark standalone, YARN, MESOS)
- Enviaremos una aplicación Spark al cluster mánager, y éste se encargará de asignar los recursos disponibles en cada servidor para la ejecución de cada tarea.

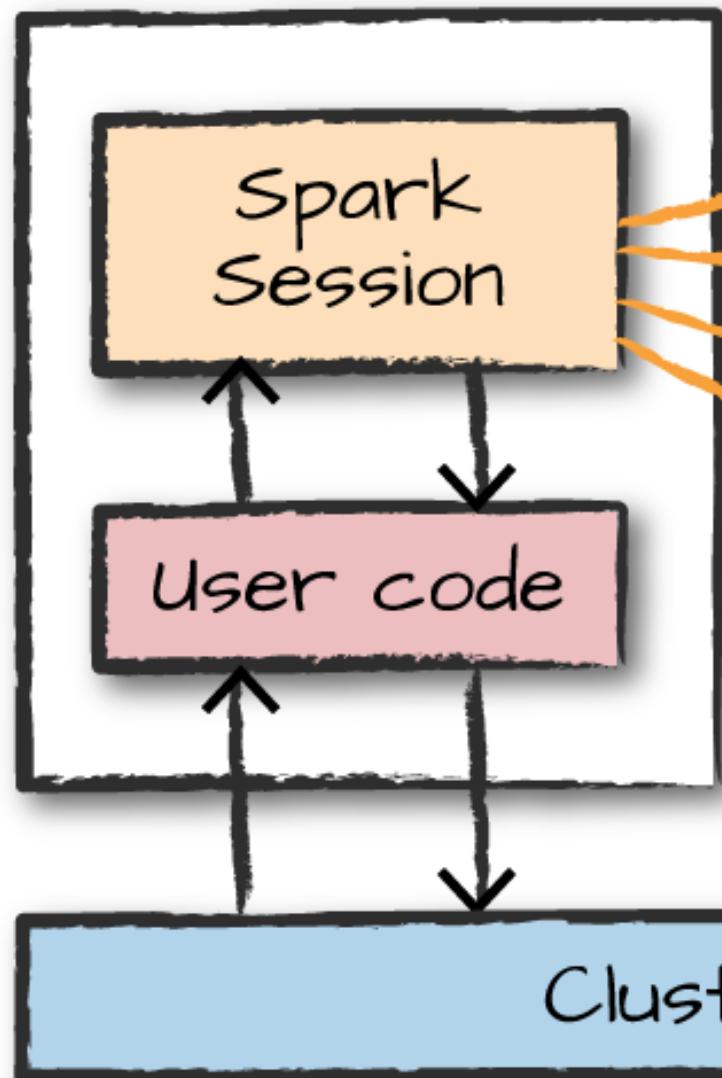
# Arquitectura

- Aplicación Spark:
  - 1 proceso **driver**
  - N procesos **executors**
- **Driver:**
  - Ejecuta nuestra función main()
  - Mantiene información sobre la aplicación Spark
  - Se encarga de responder a los parámetros de entrada
  - Analiza, distribuye y programa en el tiempo el trabajo de cada executor

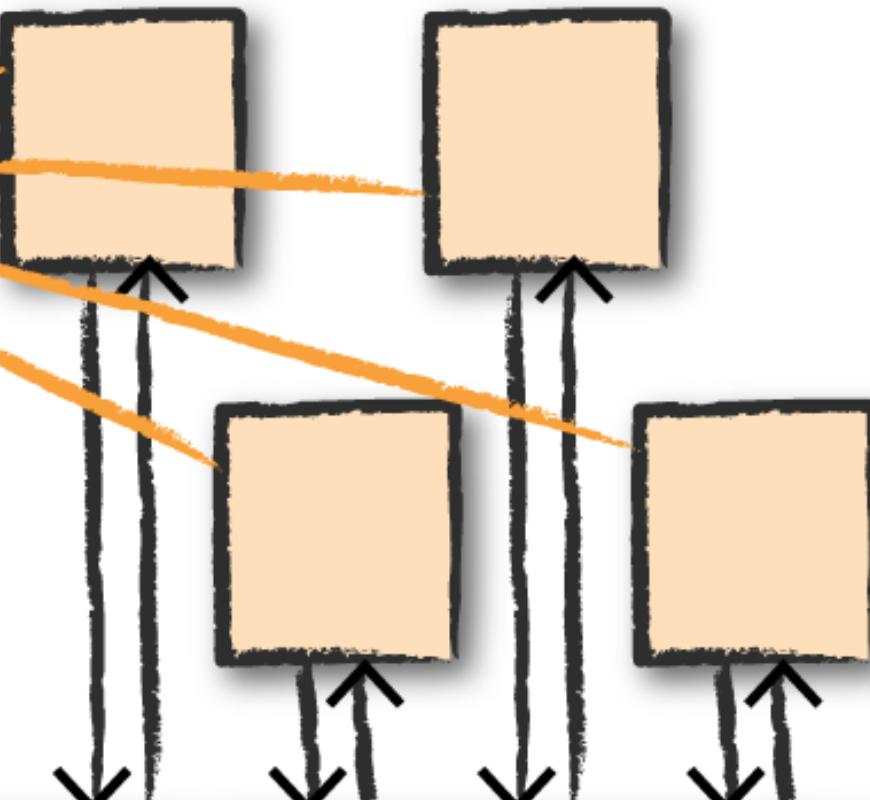
# Arquitectura

- **Executors:**
  - Ejecutan el trabajo que les asigna el driver
  - Sólo se preocupan de:
    - ejecutar el código que les ha sido asignado
    - reportar al driver el estado de la computación en este nodo particular

## Driver Process

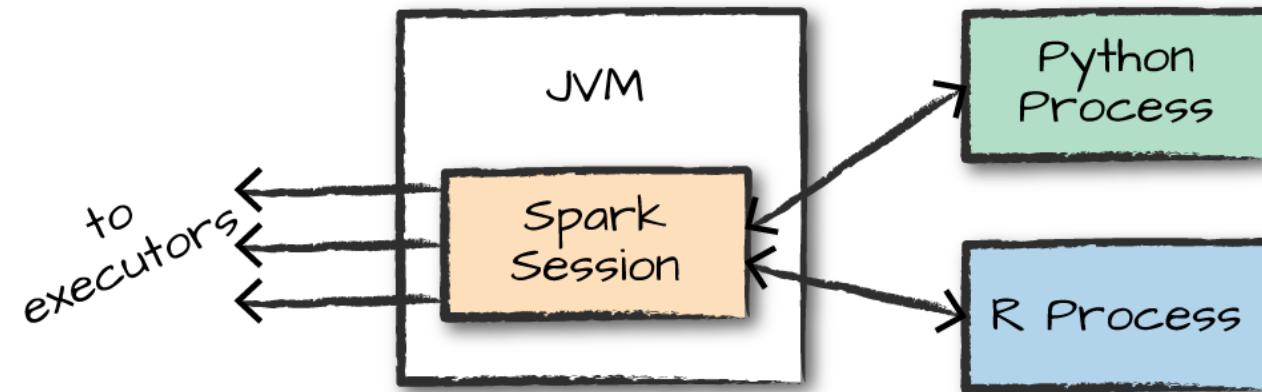


## Executors



# Spark Session

- Objeto disponible para el usuario que escribe la aplicación
- Es el punto de entrada para la ejecución de código Spark.
- Si estamos usando un lenguaje no-JVM (Python, R), sparksession se encarga de traducir nuestro código a código Spark (Scala – JVM), que se ejecuta en los executors

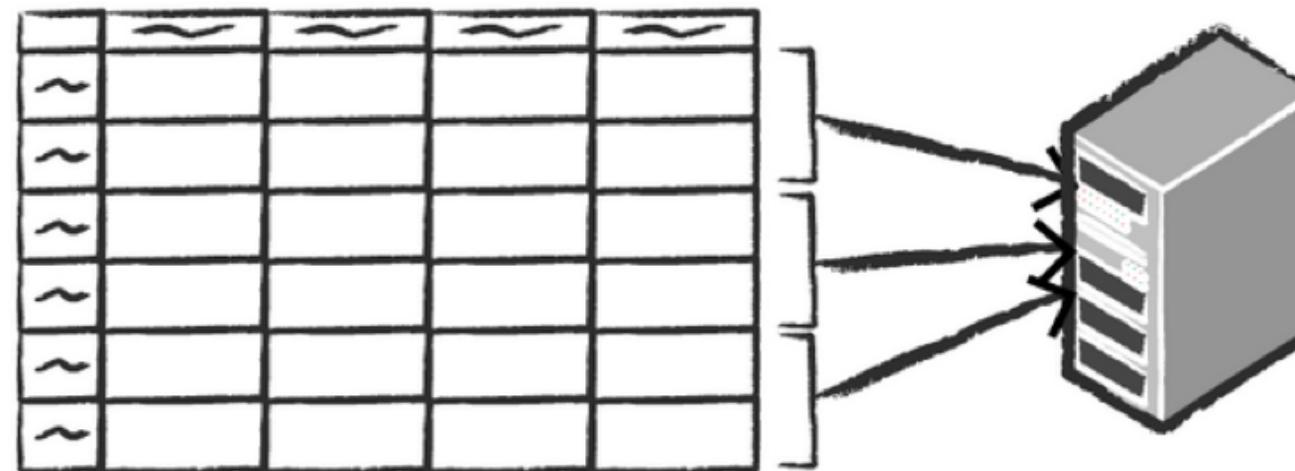


The background of the image consists of a grid of alternating white and black rectangular blocks. Within these white blocks are several smaller, square-shaped pools of water, each containing a grid of blue and white mosaic tiles.

# DataFrames

# DataFrame

- La estructura de datos más común en Spark
- Representa una tabla de datos, organizados en filas y columnas
- La diferencia con una tabla “normal” es que un DataFrame puede estar distribuído entre muchos servidores
- Podemos tener un DF más grande que la memoria de un sólo servidor, y podemos procesar sus datos en paralelo



# Particiones

- Spark distribuye un Dataframe en “trozos” (conjuntos de filas) que se denominan particiones
- Cada executor trabaja en una ó N particiones, y así se puede paralelizar el procesamiento
- Una partición es un conjunto de filas que se almacenan en un nodo (servidor) del clúster.
- El número de particiones y el número de executors, determinan el paralelismo que es posible aplicar a un procesamiento

# Transformaciones

- Spark se basa en paradigmas de programación funcional
- Una de las bases de la programación funcional es la **inmutabilidad** de las estructuras de datos
- Las estructuras de datos (DataFrames) en Spark son inmutables => no pueden modificarse una vez creados
- Cuando queremos “cambiar” un DataFrame, le indicamos a Spark qué transformación queremos aplicar, y realmente lo que ocurre es que el resultado será otro nuevo DataFrame

# Transformaciones

- Si aplicamos una transformación a un DataFrame en Spark, no veremos ninguna salida
- Esto es porque Spark no realiza ningún procesamiento hasta que se invoque una acción

```
>>> myRange = spark.range(1000).toDF("number")
>>>
>>> divisBy2 = myRange.where("number % 2 = 0")
```

# Transformaciones: 1 a 1

- Cada partición de entrada contribuye sólo a una partición de salida

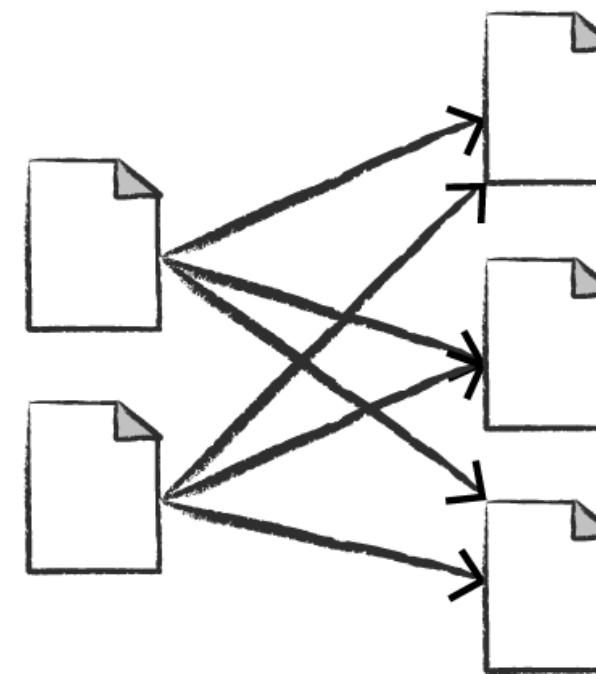
Narrow transformations  
1 to 1



# Transformaciones: 1 a N (Shuffle)

- Cada partición de entrada puede contribuir a muchas particiones de salida

Wide transformations  
(shuffles) 1 to N



# Evaluación Perezosa

- Spark espera hasta “el último momento” para ejecutar el grafo de instrucciones (DAG)
- Las transformaciones que vamos indicando, no se aplican inmediatamente, sino que se va construyendo un plan de todas las transformaciones que se quieren aplicar a un DataFrame de entrada
- Spark compila este plan lógico de transformaciones, y lo traduce a un plan físico, que optimizará lo máximo posible para ejecutarlo en los ejecutores del clúster.
- Esto permite a Spark aplicar varias estrategias de optimización (por ejemplo, “Predicate Pushdown” de filtros “where”)

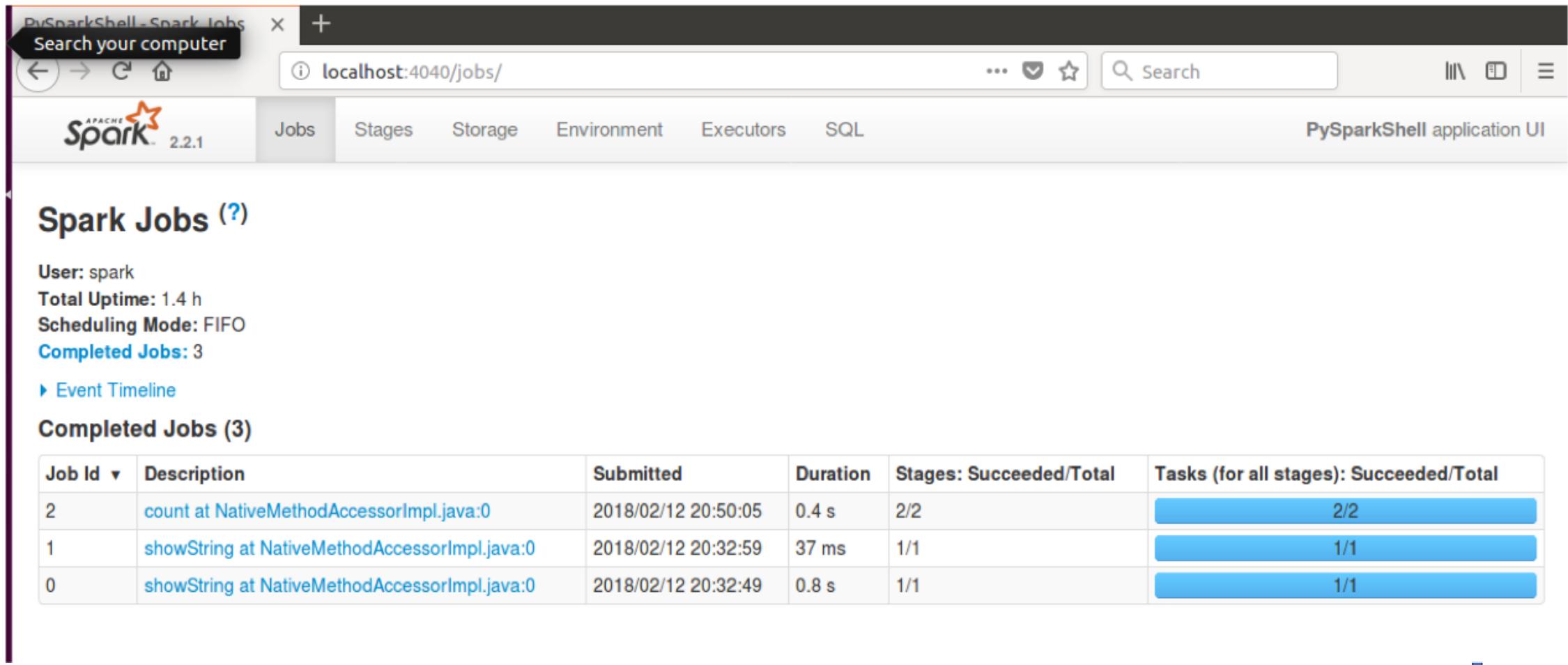
# Acciones

- Las acciones son lo que “dispara” la ejecución en Spark.
- Hasta que no se invoca una acción, no se aplican las transformaciones que hayamos definido sobre un DataFrame

```
>>> divisBy2.count()  
500
```

- 3 tipos de acciones
  - Ver datos en consola (show)
  - Recolectar datos a objetos nativos de Python
  - Escribir en fuentes de datos externas

# Interfaz Web



The screenshot shows the PySparkShell - Spark Jobs application UI running on localhost:4040/jobs/. The interface includes a header bar with tabs for Jobs, Stages, Storage, Environment, Executors, and SQL. The main content area displays information about the user (spark), total uptime (1.4 h), scheduling mode (FIFO), and completed jobs (3). It also includes a link to the event timeline. Below this, a table lists three completed jobs with details such as job ID, description, submission time, duration, stages, and tasks.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at NativeMethodAccessorImpl.java:0	2018/02/12 20:50:05	0.4 s	2/2	2/2
1	showString at NativeMethodAccessorImpl.java:0	2018/02/12 20:32:59	37 ms	1/1	1/1
0	showString at NativeMethodAccessorImpl.java:0	2018/02/12 20:32:49	0.8 s	1/1	1/1

# Jupyter Notebooks

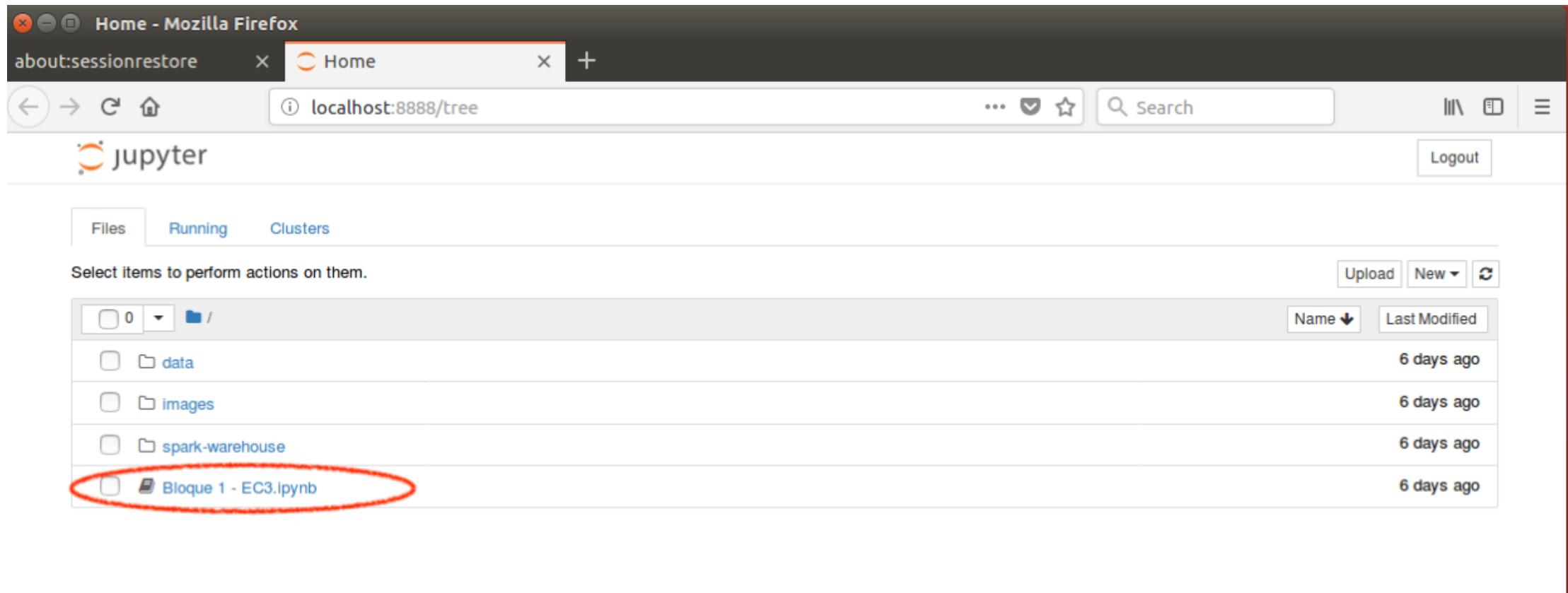


# Jupyter Notebooks – EC3

- Es posible utilizar pyspark desde notebooks de Jupyter
- Vamos a realizar un ejercicio para resumir lo visto hasta ahora, utilizando jupyter notebook

```
spark@spark-vm:~/jupyter$ workon jupyter
(jupyter) spark@spark-vm:~/jupyter$ jupyter notebook
```

# Jupyter Notebooks – EC3



Home - Mozilla Firefox

about:sessionrestore × Home × +

localhost:8888/tree

jupyter Logout

Files Running Clusters

Select items to perform actions on them.

Upload New ↗

	Name	Last Modified
<input type="checkbox"/>	data	6 days ago
<input type="checkbox"/>	images	6 days ago
<input type="checkbox"/>	spark-warehouse	6 days ago
<input type="checkbox"/>	Bloque 1 - EC3.ipynb	6 days ago

# spark-submit



# spark-submit

- Herramienta de línea de comandos, incluída en Spark para poder ejecutar nuestros Jobs Spark en producción
- Permite enviar nuestro código de aplicación Spark a un clúster, y ejecutarlo allí
- Funciona con todos los “cluster managers” (standalone, YARN, Mesos)
- Soporta el lanzamiento de aplicaciones Spark escritas en cualquier lenguaje soportado (Scala, Java, Python, R)

# spark-submit

- Código Scala:

```
spark@spark-vm:~$ spark-submit --class org.apache.spark.examples.SparkPi -master local $SPARK_HOME/examples/jars/spark-examples_2.11-2.2.1.jar 10
```

```
[... completed, ... pool]
18/02/19 22:44:18 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38,
took 4.864905 s
Pi is roughly 3.1404191404191404
18/02/19 22:44:19 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
```

# spark-submit

- Código Python:

```
spark@spark-vm:~$ spark-submit --master local $SPARK_HOME/examples/src/main/python/pi.py 10
```

```
18/02/19 22:47:53 INFO DAGScheduler: Job 0 finished: reduce at /home/spark/spark-  
2.2.1-bin-hadoop2.7/examples/src/main/python/pi.py:43, took 4.820503 s  
Pi is roughly 3.155800  
18/02/19 22:47:53 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
```