



DigitalHouse >
Coding School

DATA SCIENCE

Procesamiento
Distribuido con Apache
Spark

1

Introducción a Big Data

2

Introducción a Apache Spark

3

Spark Core

4

Spark SQL

5

Conclusiones



MUTT DATA

Juan Martín Pampliega - Ingeniero en Informática ITBA
Co Founder Mutt Data - Profesor de Posgrado ITBA

- Comencé a trabajar con herramientas de Big Data en 2010 con Hadoop.
- Trabajé y lideré proyectos en Globant (Google), Despegar, Socialmetrix y Jampp.
- Utilicé distintas herramientas desde DataFlow (Google), Pig, HBase, Hadoop, Hive, Spark, PrestoDB, Airflow, AWS Kinesis, AWS Lambda, Flume, etc.
- Actualmente soy Co Founder en Mutt Data, una empresa enfocada en la consultoría y desarrollo de proyectos de Big Data y Data Science.

Big Data



Todo comienza con eventos en una base

Inicialmente armamos el sistema transaccional en una base:

login, create user, create restaurant, create order, view order, receive order, etc.

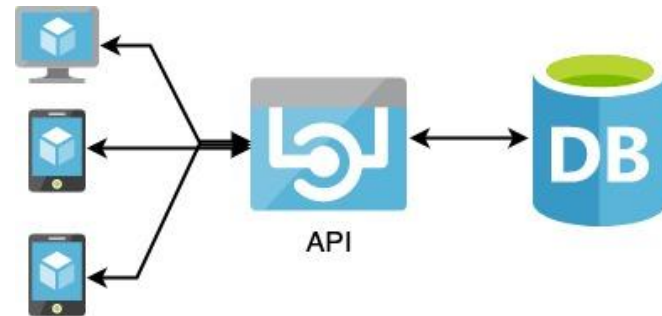
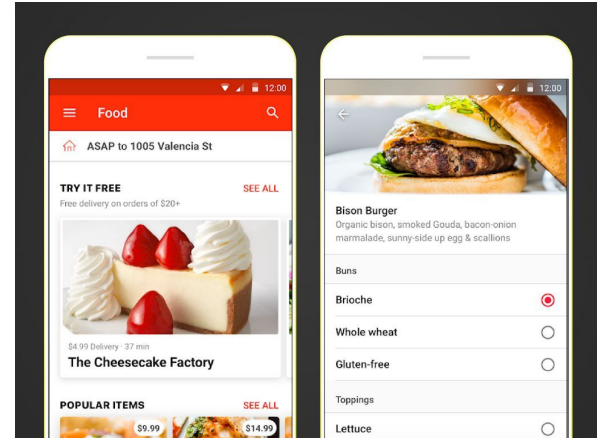
This type of operations mostly involve INSERT/SELECT/UPDATE/DELETE operations over a small group of rows.

OLTP (OnLine Transactional Processing)

Like:

INSERT INTO orders

VALUES ('User1', 'Restaurant1', 'cheeseburger');



Pero los usuarios hacen preguntas

Cuántas órdenes se hicieron ayer?

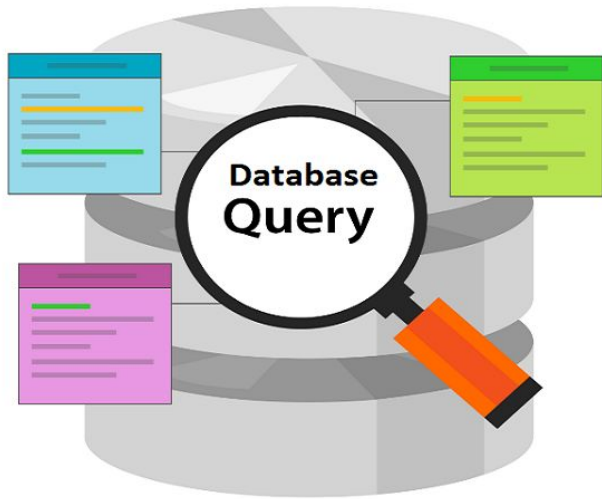
Cuál es el ítem más pedido?

Dónde se hacen la mayoría de mis pedidos?

Cuánta distancia recorre un delivery en promedio por día?

Estas preguntas involucran agregaciones sobre muchas filas (OLAP)

```
SELECT day(date), count(1)
FROM orders
WHERE date = YESTERDAY
GROUP BY day(date)
```



Una base de datos no es suficiente

Dónde están los deliveries en cada momento del día?

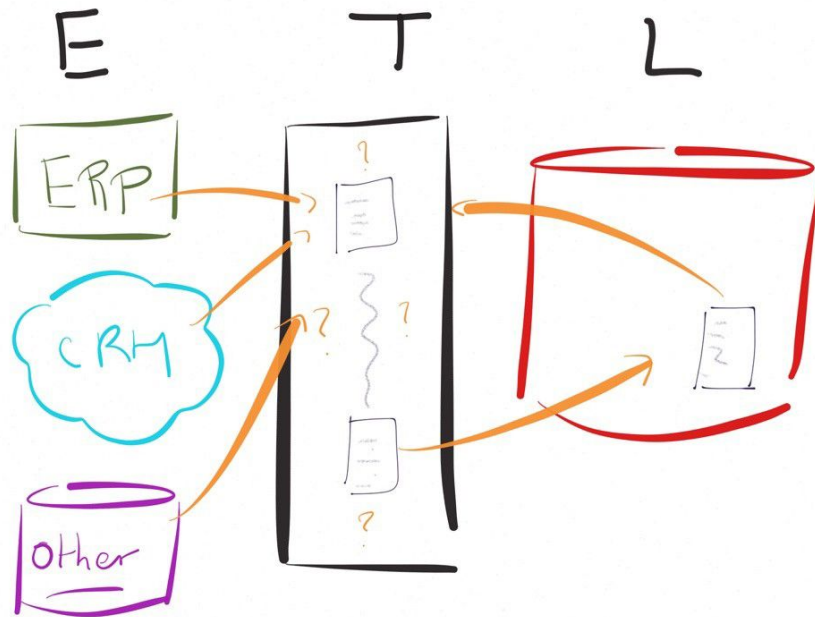
Cuántos usuarios únicos tuvimos cada día?

Cual va a ser el tiempo estimado de delivery de una hamburguesa el sábado a las 20hs?

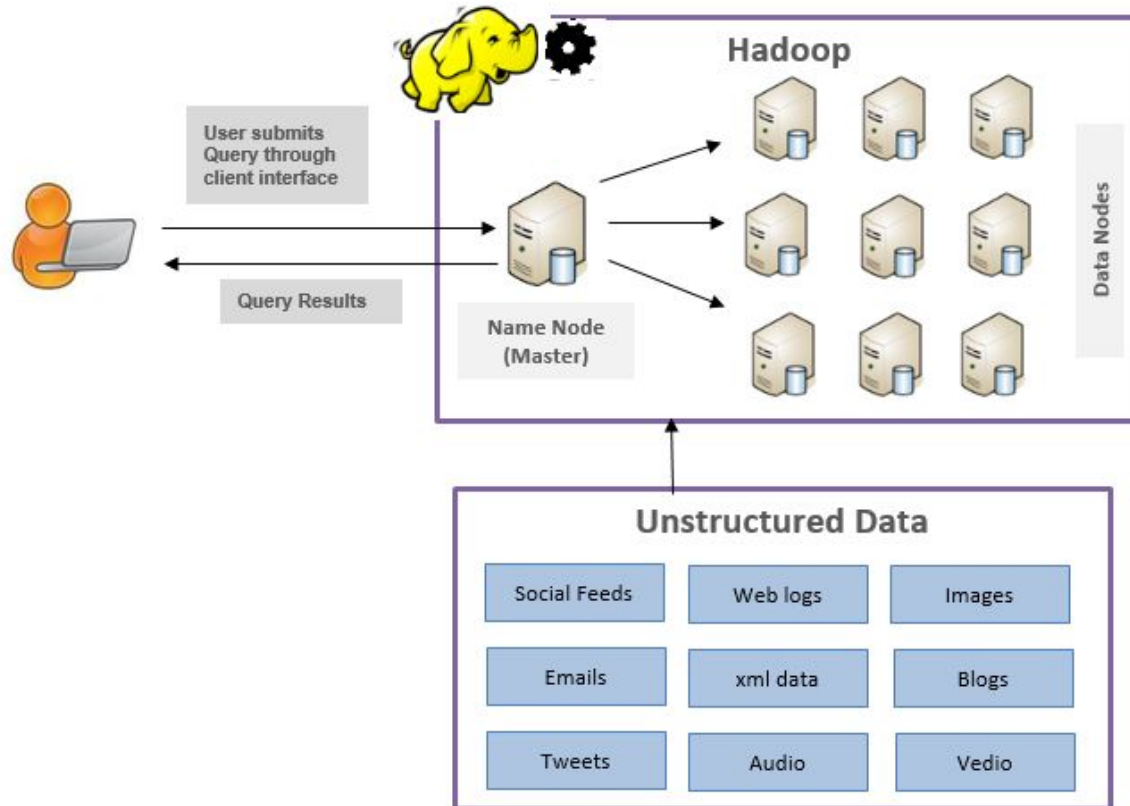
OLAP (OnLine Analytical Processing)

Involucra tener una base de datos o sistema analítico separado de la transaccional.

Los procesos ETLs extraen los datos de los sistemas transaccionales y los llevan a las bases de datos OLAP luego de transformarlos.



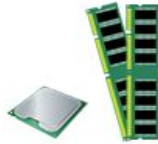
Qué es un cluster?



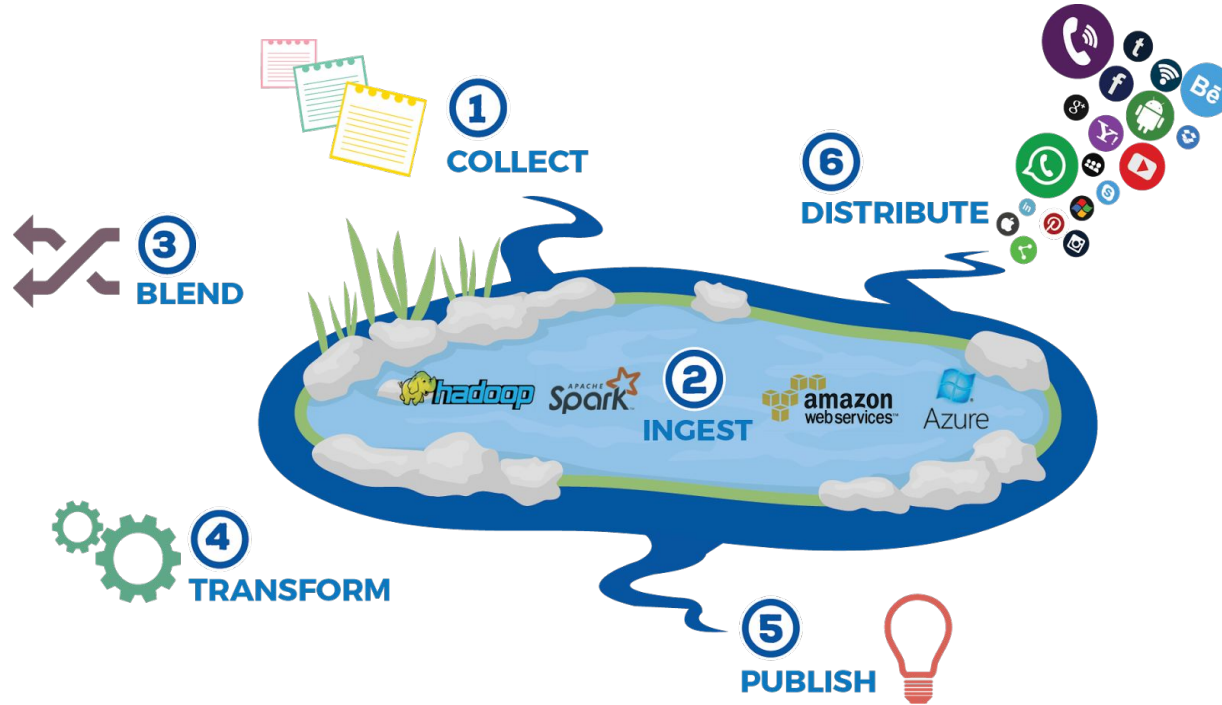
Pets vs Cattle



↑
Vertical Scaling



← Horizontal Scaling →



Apache Spark

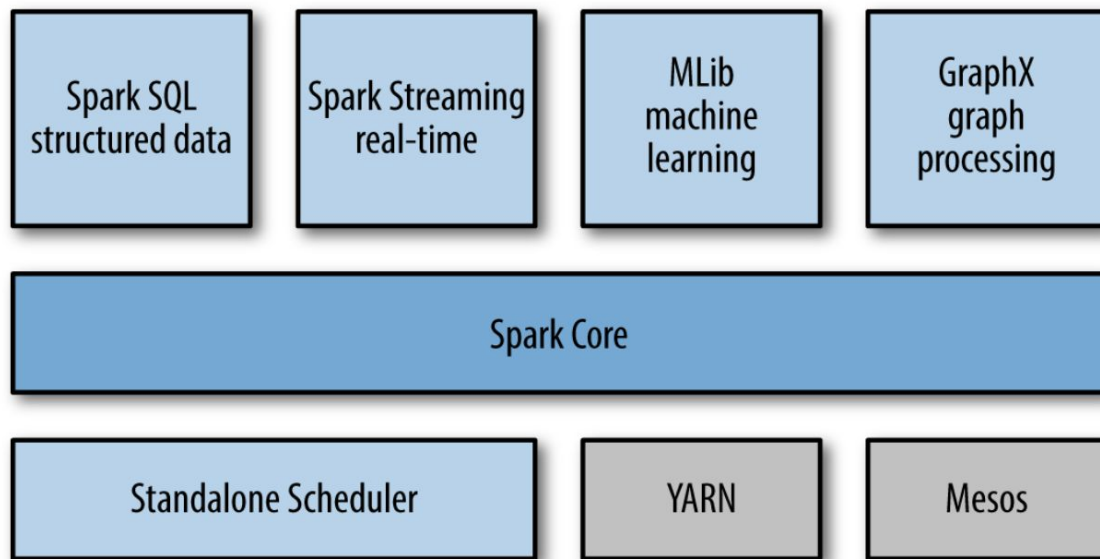


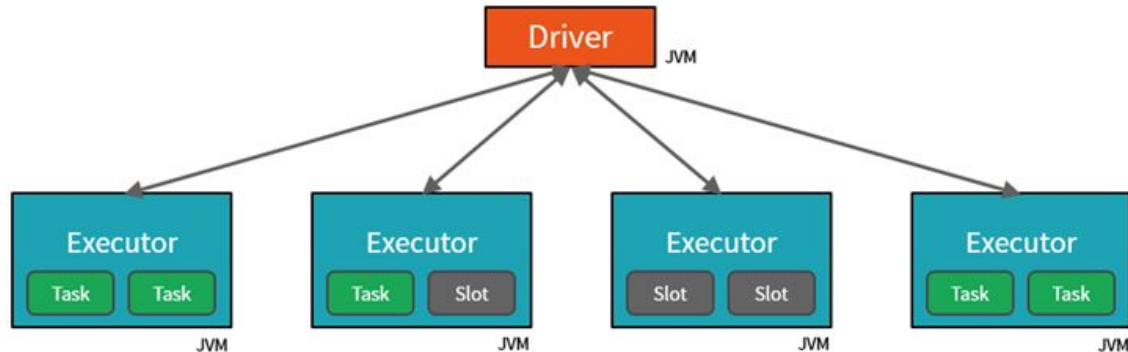
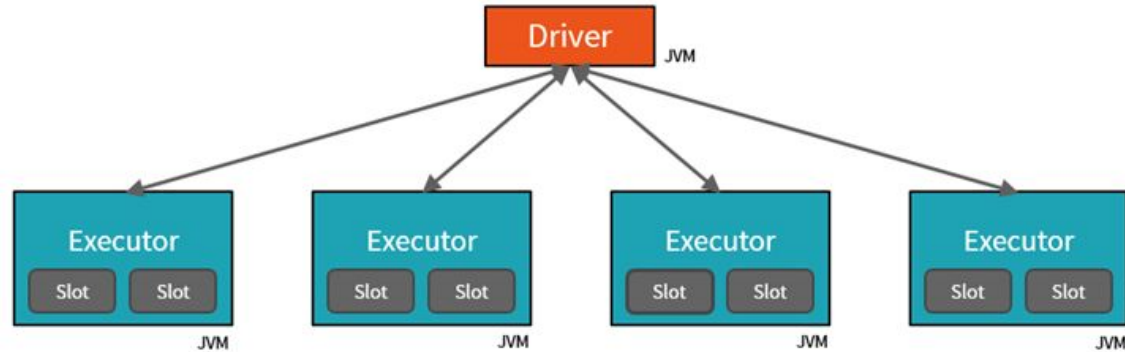
Es un **sistema general de procesamiento de datos a gran escala de forma distribuida.**

Creado en 2009 como un proyecto de research en **UC Berkeley RAD Lab.**

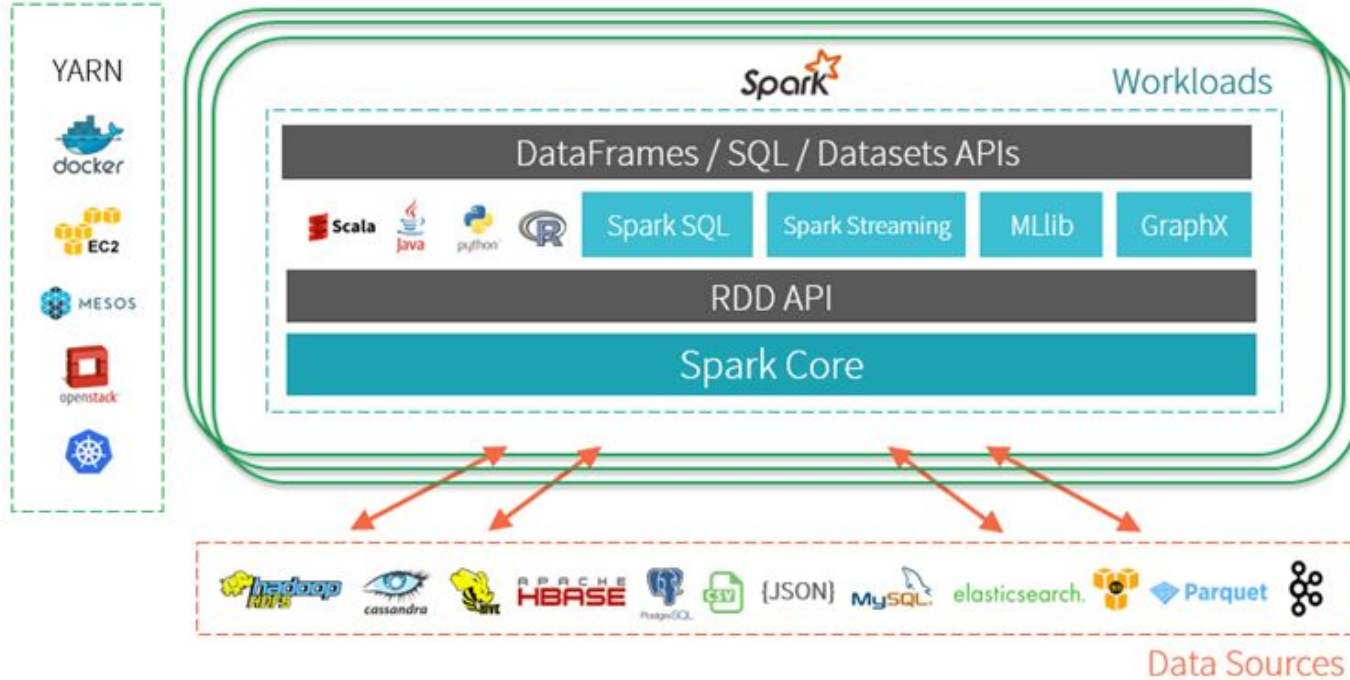
Provee librerías para **Scala, Java, Python y R** para ejecutar este procesamiento de forma distribuido.

Posibilita el procesamiento **batch, interactivo, continuo e iterativo** de forma eficiente.





Environments



Spark Core




```
sc = SparkContext('spark://...', 'MyJob')
```

```
file = sc.textFile('hdfs://...')
```

```
errors = file.filter(lambda line: 'ERROR' in line)
```

```
errors.cache()
```

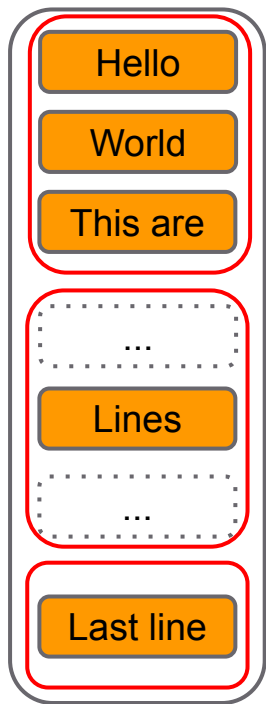
```
errors.count()
```

Resilient distributed
datasets (RDDs)

The diagram consists of an orange rounded rectangle at the top right containing the text 'Resilient distributed datasets (RDDs)'. Two orange arrows originate from the bottom of this box. One arrow points left towards the `textFile` method in the second line of code. The other arrow points down and to the left towards the `count` method in the fifth line of code.


Action

The diagram consists of an orange rounded rectangle containing the text 'Action'. An orange arrow points from the left side of this box towards the `count` method in the fifth line of code.

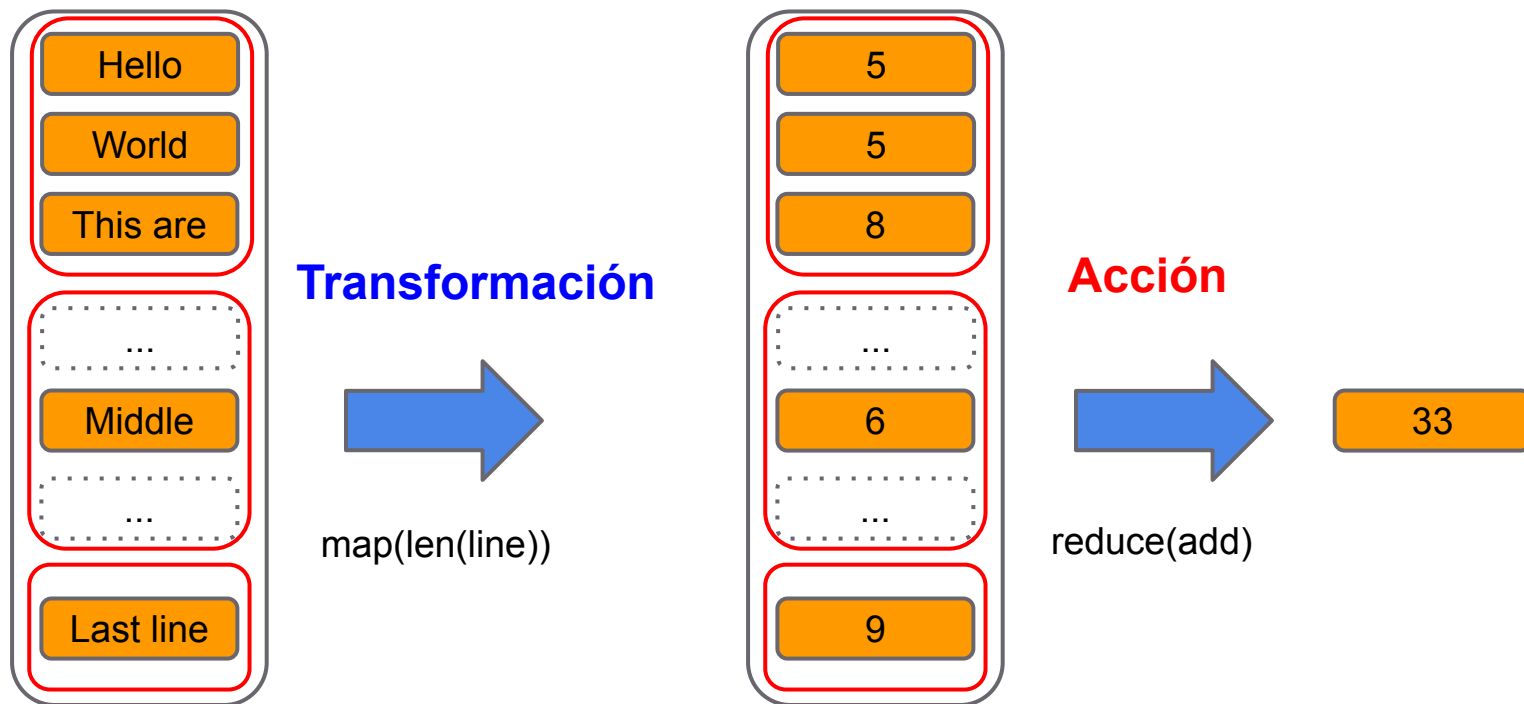


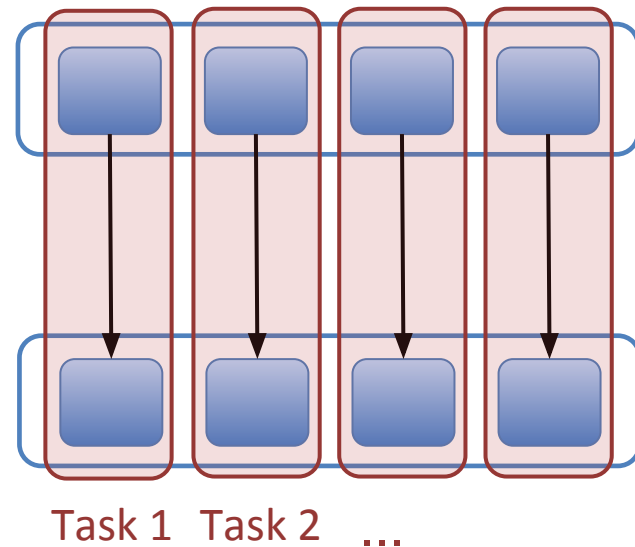
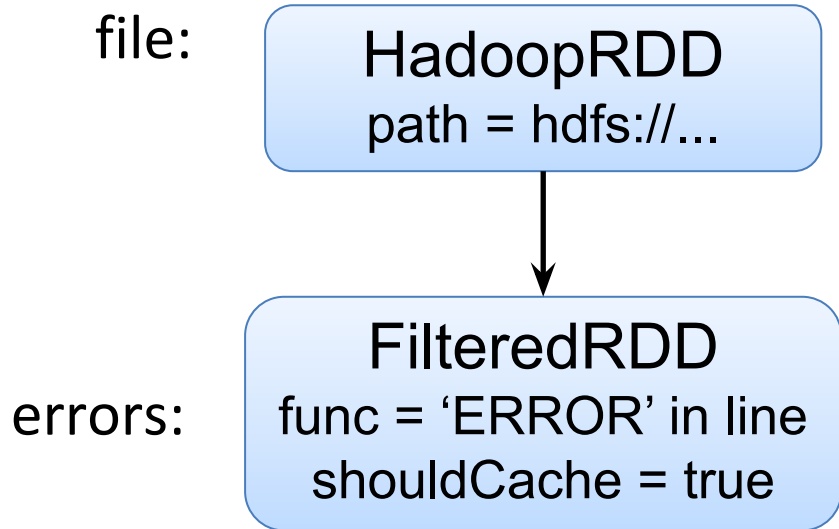
- Colección **inmutable** de objetos.
- **Particionada** y **distribuida**.
- Almacenada en **memoria** (permite spill a disk).
- Las **particiones se re-computan** ante un fallo.
- Hay RDD de distintos tipos y definen:
 - o una serie de particiones
 - o dependencia de una partición de su RDD padre
 - o función para computar una partición

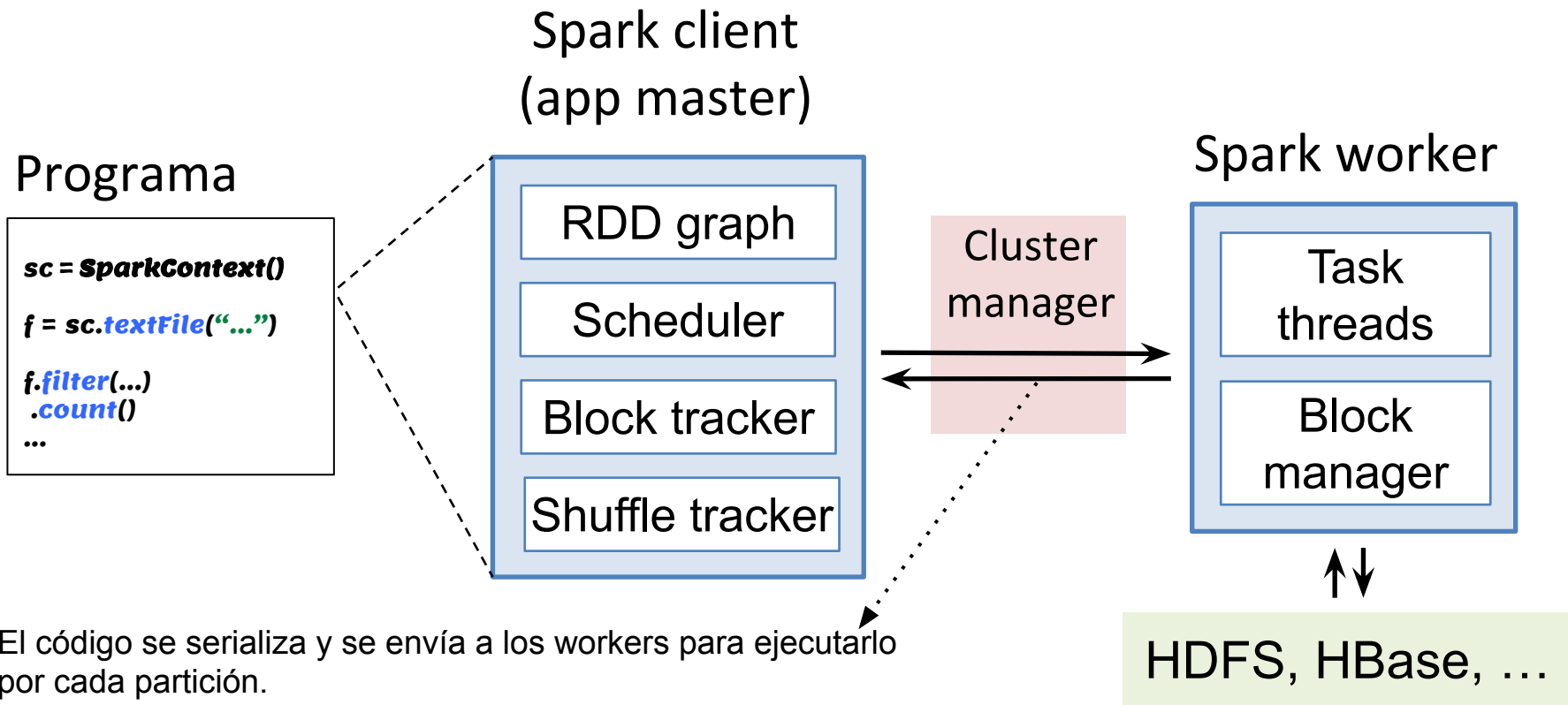
- **Transformaciones:** son operaciones sobre RDDs que se ejecutan de manera lazy y generan otro RDD.
- **Acciones:** son operaciones que se ejecutan en el momento y que combinan la serie de transformaciones acumuladas para obtener un resultado.



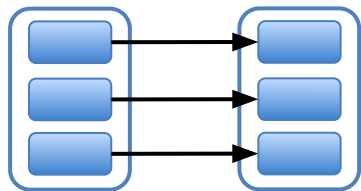
Transformations (lazy)	Actions
select	show
distinct	count
groupBy	collect
sum	save
orderBy	
filter	
limit	



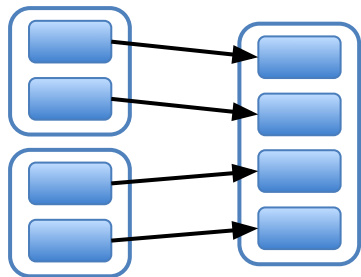




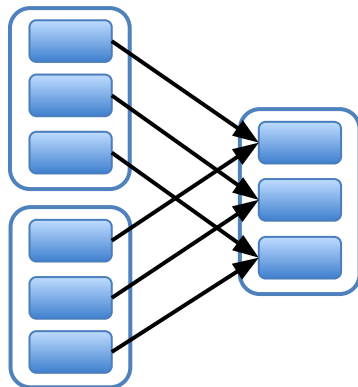
“Narrow” (pipeline-able):



map, filter

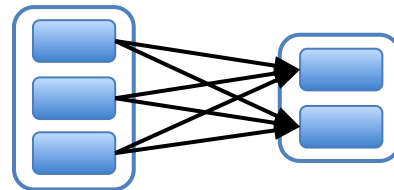


union

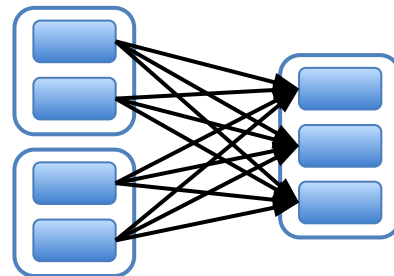


join with inputs
co-partitioned

“Wide” (shuffle):

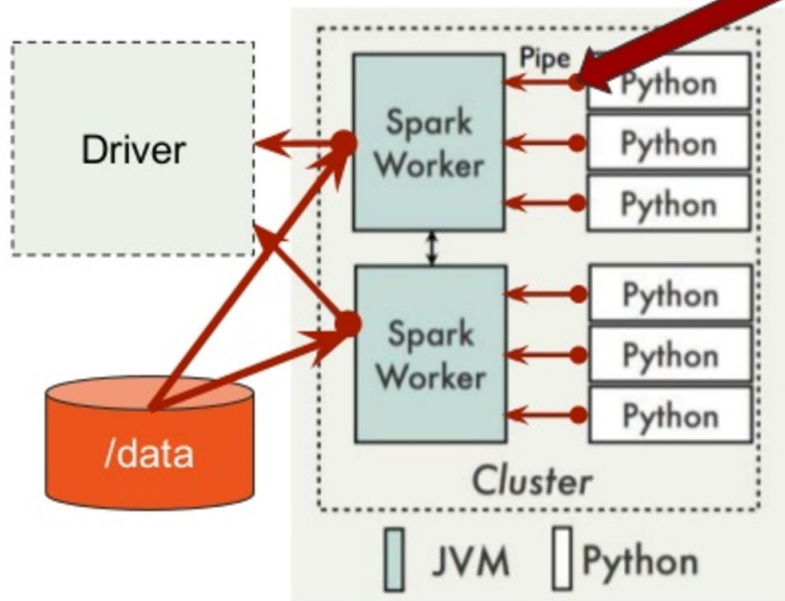


groupByKey



join with inputs not
co-partitioned

```
sc.textFile("/data")  
  .filter(lambda s: "foobar" in s)  
  .count()
```



Java-to-Python
communication
is expensive!

Databricks Workspace



Databricks Platform

A screenshot of the Databricks Workspace interface. On the left is a dark sidebar with navigation icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area shows a Python notebook titled "dh_word_count_quijote". It contains two code cells. Cell 1 imports urllib and requests a gzipped file from GitHub. Cell 2 uses PySpark's RDD API to read the file, process it, and print word frequencies. The interface includes a top toolbar with options like File, View: Code, Permissions, Run All, Clear, Publish, Comments, and Revision history. A command bar at the bottom of the code area shows execution time and user information.

```
dh_word_count_quijote (Python)

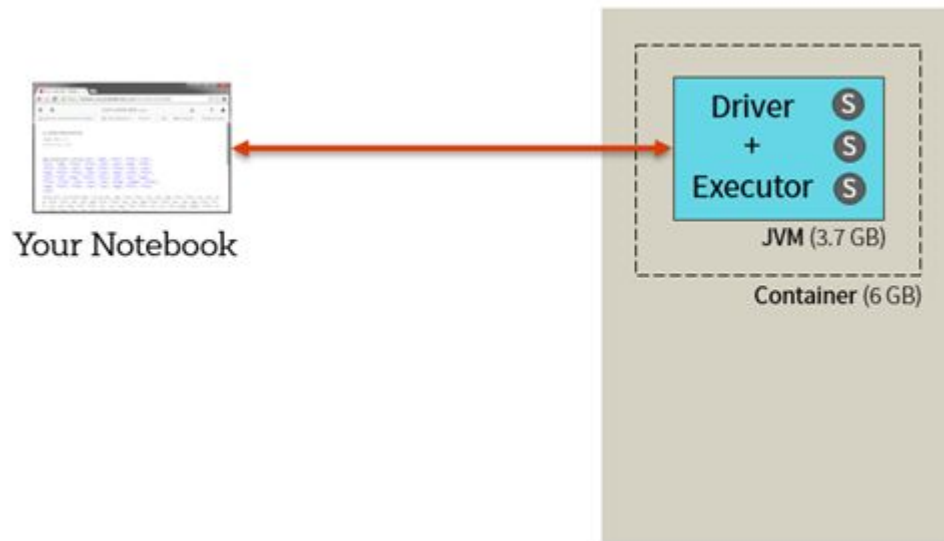
Attached: testclust File View: Code Permissions Run All Clear Publish Comments Revision history

Cmd 1
1 import urllib
2
3 with urllib.request.urlopen('https://github.com/juanpampliega/datasets/raw/master/don-quijote.txt.gz') as response:
4     gzipcontent = response.read()
5
6 with open("/dbfs/tmp/don-quijote.txt.gz", 'wb') as f:
7     f.write(gzipcontent)

Command took 0.93 seconds -- by jpamplie@itba.edu.ar at 8/16/2018, 2:59:04 AM on unknown cluster

Cmd 2
1 from operator import add
2
3 docs = sc.textFile("dbfs:/tmp/don-quijote.txt.gz")
4
5 lower = docs.map(lambda line: line.lower())
6 #print(lower)
7 #print(lower.take(10))
8
9 words = lower.flatMap(lambda line: line.split(' '))
10 #print(words.take(10))
11
12 counts = words.map(lambda word: (word, 1))
13 #print(counts.take(10))
14
15 freq = counts.reduceByKey(add)
16 #print(freq.take(10))
17
18 invFreq = freq.map(lambda t: (t[1],t[0]))
19
```

<https://community.cloud.databricks.com> - Community Edition



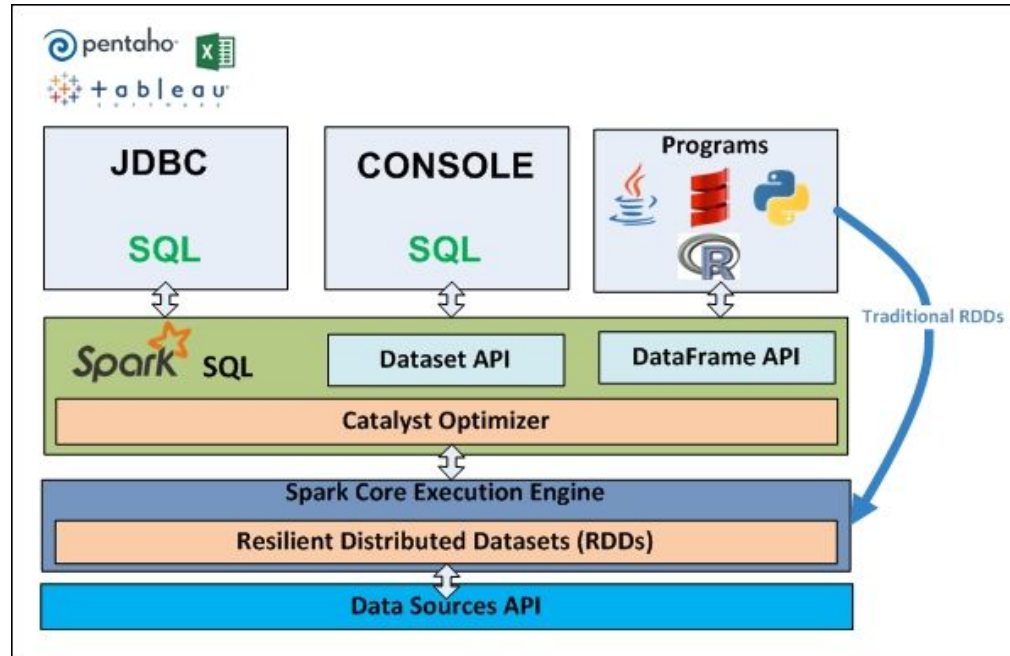
<https://community.cloud.databricks.com> - Community Edition

Práctica Spark Core



Spark SQL





**Módulo de Spark para procesamiento de datos estructurados.
Tiene un objeto de tipo DataFrame (como Pandas) pero distribuido**

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

```
# spark is an existing SparkSession
```

```
df = spark.read.json("examples/src/main/resources/people.json")
```

```
# Displays the content of the DataFrame to stdout
```

```
df.show()
```

```
# +-----+
# | age|   name|
# +-----+
# |null|Michael|
# | 30|   Andy|
# | 19|  Justin|
# +-----+
```

```
# spark, df are from the previous example
# Print the schema in a tree format
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
# +-----+
# |   name|
# +-----+
# |Michael|
# |   Andy|
# |  Justin|
# +-----+

# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-----+
# |   name|(age + 1)|
# +-----+
# |Michael|      null|
# |   Andy|       31|
# |  Justin|       20|
# +-----+

# Select people older than 21
df.filter(df['age'] > 21).show()
# +-----+
# |age|name|
# +-----+
# | 30|Andy|
# +-----+

# Count people by age
df.groupBy("age").count().show()
# +-----+
# | age|count|
# +-----+
# | 19|    1|
# |null|    1|
# | 30|    1|
# +-----+
```

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")
```

```
sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

```
# +-----+
# | age |   name |
# +-----+
# | null | Michael |
# |   30 |    Andy |
# |   19 |   Justin |
# +-----+
```

```
from pyspark.sql import Row
```

```
sc = spark.sparkContext
```

```
# Load a text file and convert each line to a Row.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
```

```
# Infer the schema, and register the DataFrame as a table.
schemaPeople = spark.createDataFrame(people)
schemaPeople.createOrReplaceTempView("people")
```

```
# SQL can be run over DataFrames that have been registered as a table.
teenagers = spark.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

```
# The results of SQL queries are Dataframe objects.
# rdd returns the content as an :class:`pyspark.RDD` of :class:`Row` objects.
teenNames = teenagers.rdd.map(lambda p: "Name: " + p.name).collect()
for name in teenNames:
    print(name)
# Name: Justin
```

```
df = spark.read.load("examples/src/main/resources/users.parquet")  
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

```
df = spark.read.load("examples/src/main/resources/people.json", format="json")  
df.select("name", "age").write.save("namesAndAges.parquet", format="parquet")
```

```
df = spark.read.load("examples/src/main/resources/people.csv",  
                    format="csv", sep=":", inferSchema="true", header="true")
```

```
df = spark.sql("SELECT * FROM parquet.`examples/src/main/resources/users.parquet`")
```


Práctica Spark SQL



CONCLUSIONES



- Spark es el standard de facto para procesamiento distribuido.
- Spark se puede usar en Python a través de la API de PySpark.
- Por default uno utiliza siempre los Dataframes de Spark SQL.
- Al utilizar SparkSQL uno evita el overhead de serialización de código entre la JVM y Python.