

# Arquitecturas Cloud y Big Data



red.es

Centro de  
Referencia Nacional  
en Comercio Electrónico  
y Marketing

CRN  
Digital



"El FSE invierte en tu futuro"  
Fondo Social Europeo


# Índice

1. Introducción a Spark
2. Entorno de trabajo: Databricks Community Edition.
3. Introducción a PySpark: RDDs.
4. RDDs: Transformaciones

# 1. Introducción a Spark



# SPARK: Historia

- En 2009 surge dentro de un proyecto de investigación en Berkeley
- La idea era hacer algo rápido para consultas interactivas.  
De ahí el utilizar datos en memoria en vez de disco.
- En 2010 se hace de código abierto
- En 2013 se transfiere a la fundación Apache.
- Spin-off databricks  databricks
- Actualmente en multitud de entornos y en las 3 grandes CLOUD.



# Que es SPARK

- Apache Spark: motor de código abierto desarrollado específicamente para el procesamiento distribuido (clúster) de datos a gran escala, ya sea utilizando arquitecturas on premise (servidores propios, datacenters) o la nube (CLOUD)
- Spark proporciona almacenamiento en memoria para los cálculos intermedios, lo que lo hace mucho más rápido
- Incorpora APIs varios lenguajes: Scala, Python, SQL, Java, R



# Que es SPARK

- Mayor funcionalidad incorporada y unificada: machine learning (MLlib), SQL para consultas interactivas (Spark SQL), procesamiento tanto en “batch” (por lotes) como “streaming” (en tiempo real).

Spark SQL and  
DataFrames +  
Datasets

Spark Streaming  
(Structured  
Streaming)

Machine Learning  
MLlib

Graph  
Processing  
Graph X

Spark Core and Spark SQL Engine

Scala

SQL

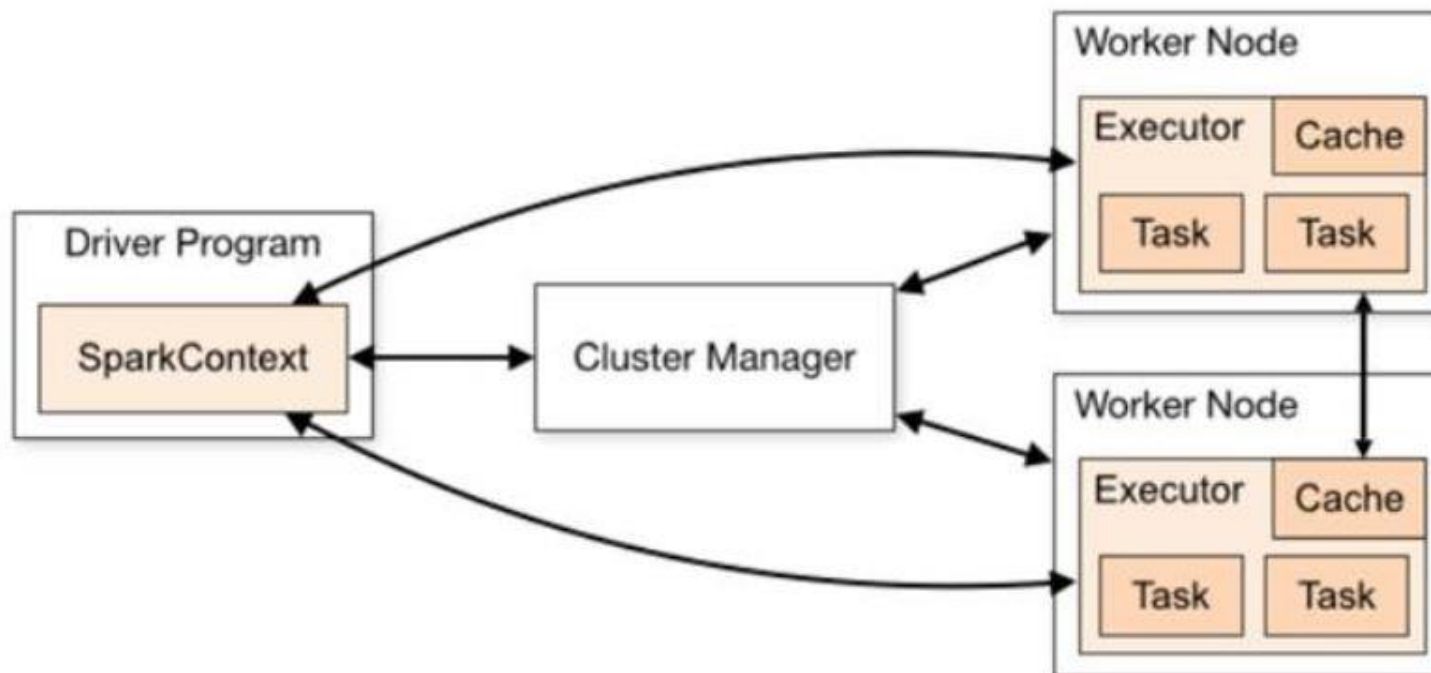
Python

Java

R

# Spark funcionamiento interno

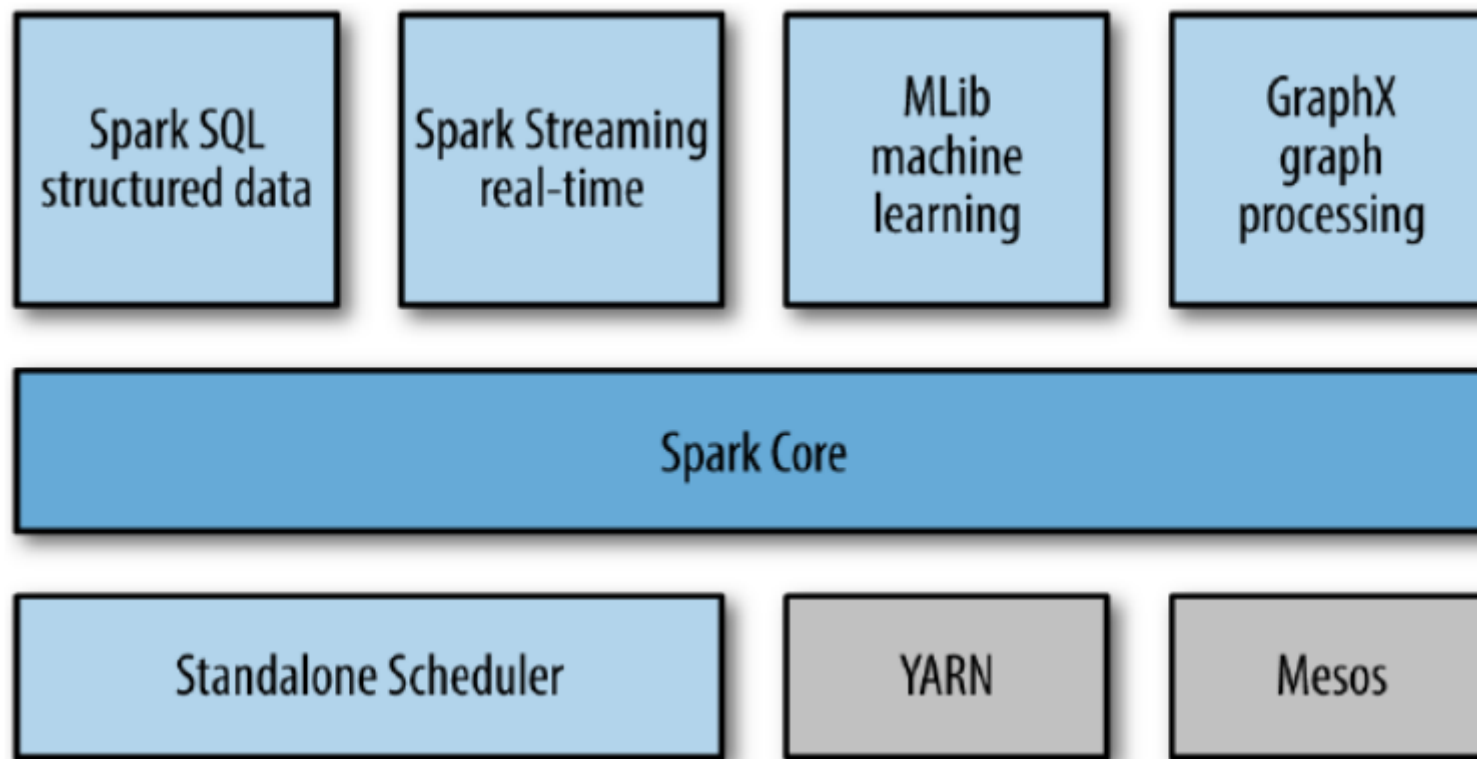
- Existe un nodo “maestro” (DRIVER NODE) que es donde se ejecuta nuestro código (DRIVER PROGRAM), en el que corre el proceso “DRIVER PROCESS” (se le suele llamar también sólo Driver, controlador). El Driver crea la sesión (“SparkSession” y en su caso “SparkContext”), hace petición de los recursos necesarios al clúster.
- El clúster manager (gestor de recursos) maneja y asigna recursos del clúster. Coordina las diferentes etapas del trabajo. Spark admite varios (YARN, Mesos...)
- Los nodos “esclavos” o WORKERS son donde se encuentran repartidos los datos y se llevan a cabo las operaciones con ellos (unidad mínima tareas, Task) en los procesos EXECUTOR.





# Gestores de recursos

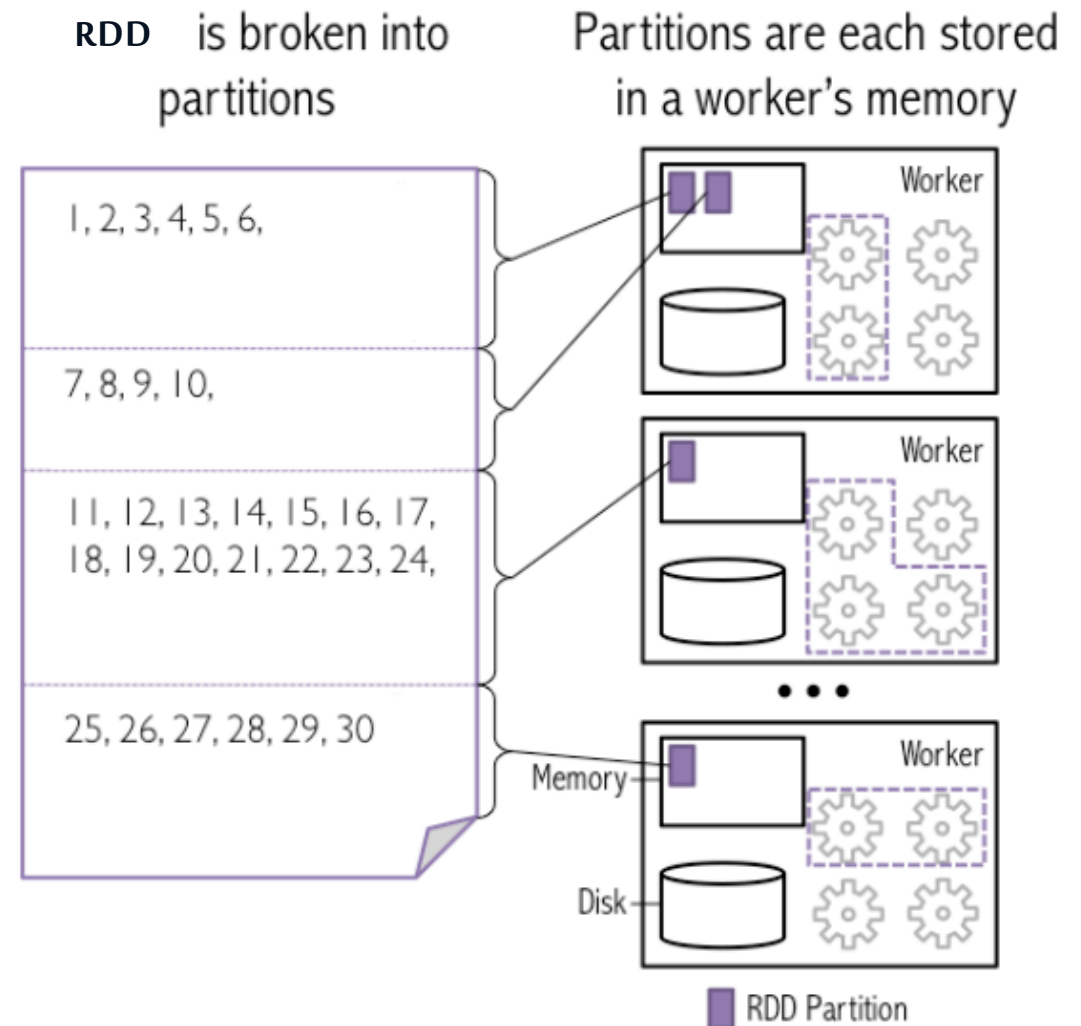
- Spark admite varios gestores de recursos, entre ellos un gestor de recursos autónomo integrado (Standalone Scheduler), Apache Hadoop YARN y Apache Mesos.





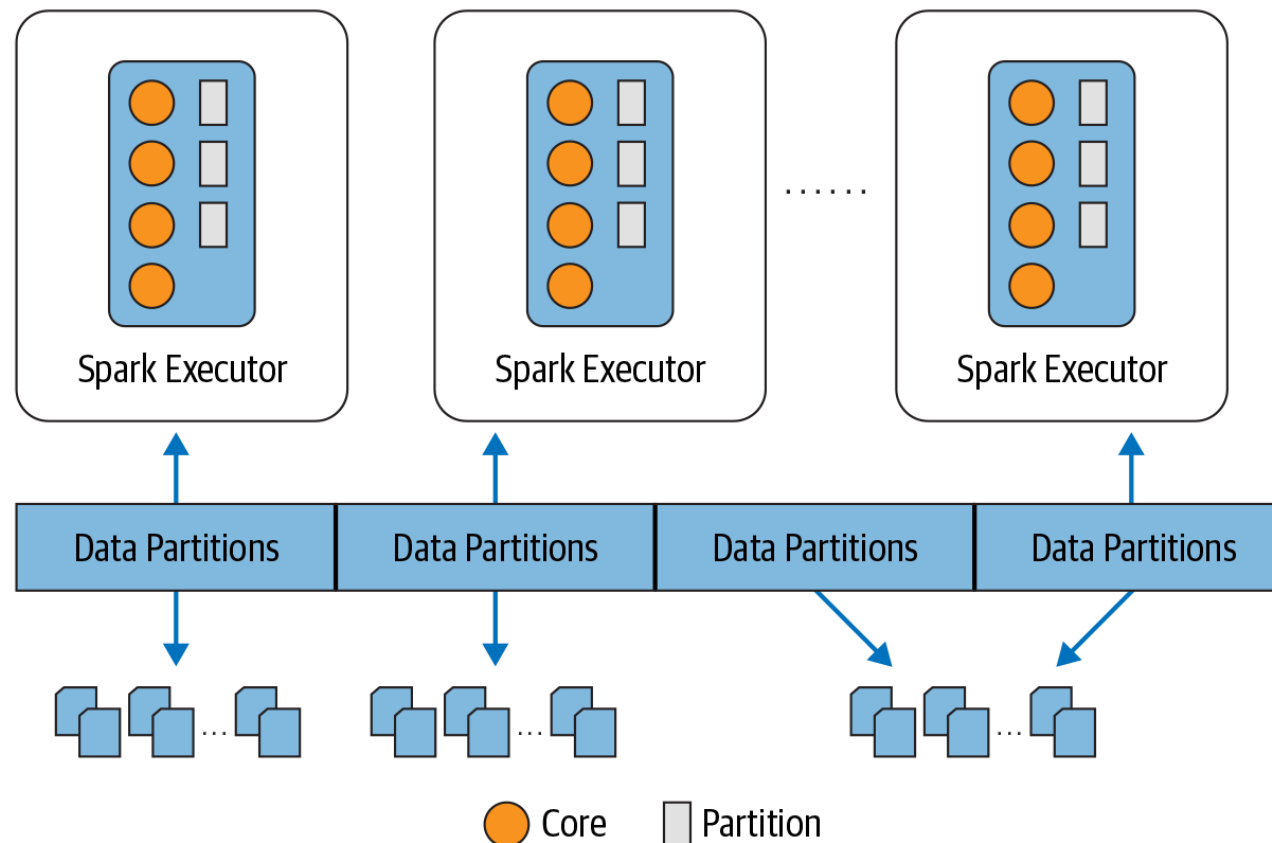
# PARTICIONES DATOS (sistema distribuido)

- Los datos se dividen en particiones que se distribuyen en memoria de distintos nodos (**nodos workers**)
- Aunque esto no siempre es posible, a cada ejecutor de Spark se le adjudica preferentemente una tarea que requiera leer la partición más cercana a él en la red.

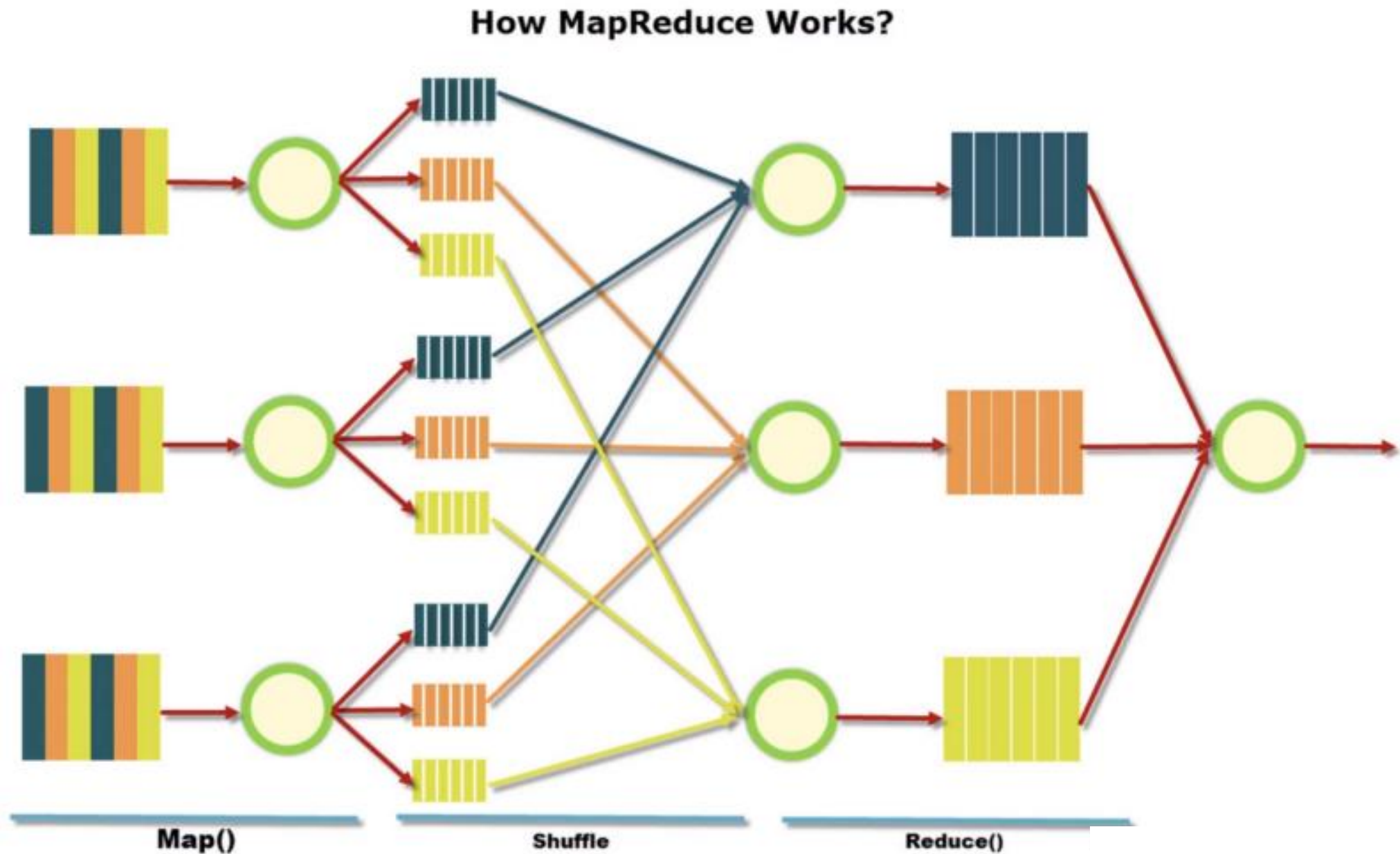


# PARTICIONES DATOS (sistema distribuido)

- Spark divide los datos en particiones y ejecuta las operaciones en las particiones en paralelo
- Los “**executors**” son los **procesos en los nodos workers** encargados de ejecutar dichas operaciones

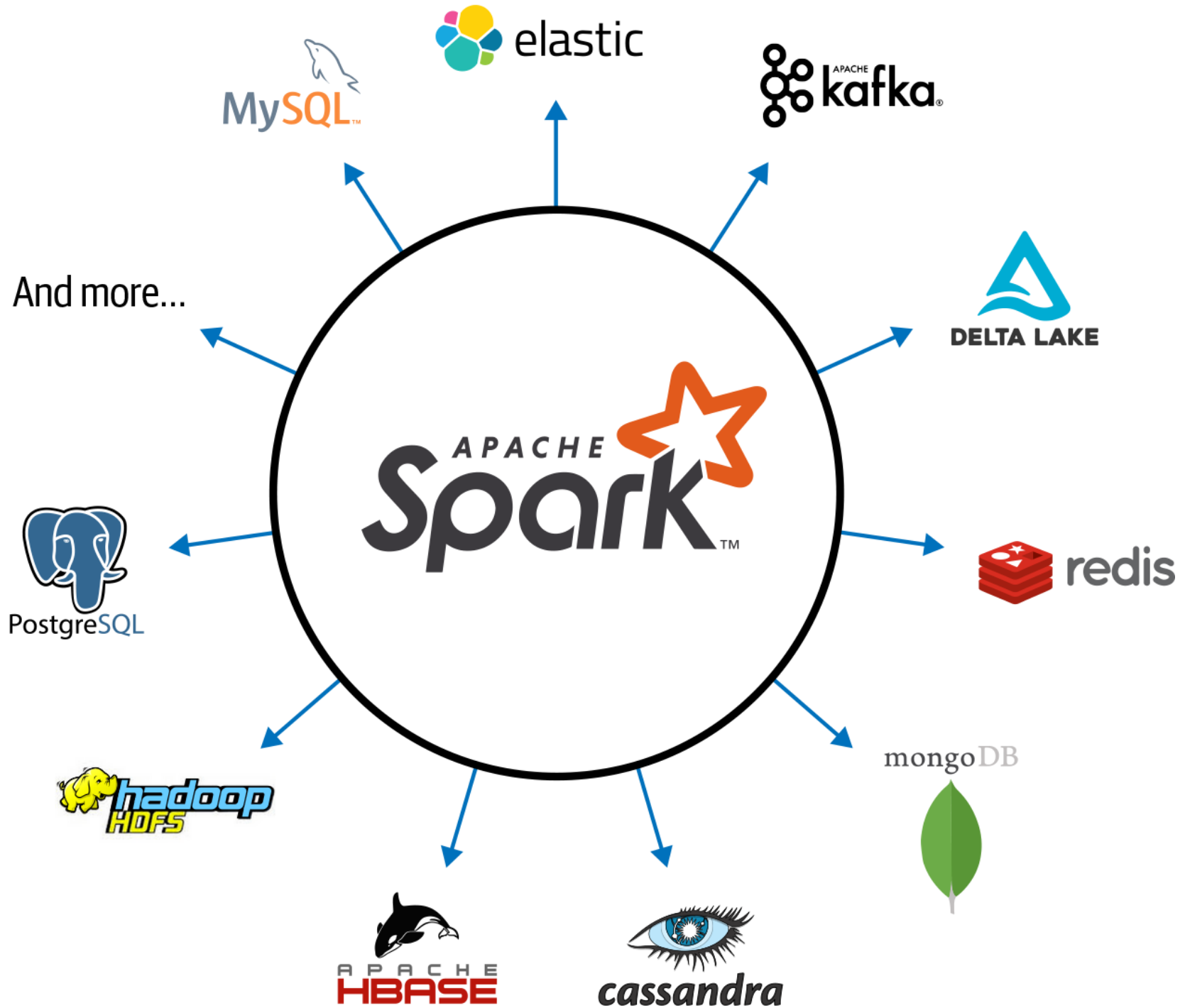


# SPARK: Ejemplo transformación “reduceByKey(...)”



# Distintos orígenes de datos

- **Spark se centra en su motor de cálculo rápido y paralelo más que en el almacenamiento.** A diferencia de Hadoop, Spark desvincula ambos.
- Se puede utilizar Spark para **leer/escribir datos** almacenados en **innumerables fuentes**
- Hadoop: HDFS (“por defecto”), Apache Hbase, Hive.
- RDBMSs (PostgreSQL, MySQL, SQL Server..) y BBDD NoSQL (Cassandra, MongoDB, Redis...)
- Apache Kafka, GCP BigQuery, Azure Storage y Amazon S3
- Comunidad libre de desarrolladores de Spark: **conectores** Spark para una **variedad de fuentes de datos externas.**



## SPARK: Quienes lo usan (ejemplos)

YAHOO!



databricks

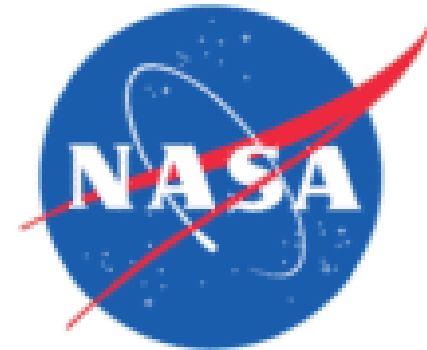


Alibaba.com

NOKIA  
CONNECTING PEOPLE



tripadvisor



ebay

NETFLIX

# 2. Entorno de trabajo: Databricks Community Edition<sup>ⓧ</sup>






# DATABRICKS COMMUNITY EDITION

- Entorno de aprendizaje prácticamente misma interfaz que la comercial (limitación clúster)
- **Totalmente gratuito**, de por vida (no periodo de prueba gratis 15 días...)
- **No te pide datos de la tarjeta de crédito**, no “sorpresas”
- **Databricks** claramente al alza (fundadores de Spark, cloud neutral)
- Enlace: <https://databricks.com/try-databricks>



# CREAR UNA CUENTA

Rellenar formulario: vale cualquier email, nº teléfono opcional.

  
databricks

[Platform](#)  
[Solutions](#)  
[Learn](#)  
[Customers](#)  
[Partners](#)  
[Company](#)




[Try Databricks](#)

[Watch Demos](#)  
[Contact Us](#)  
[Login](#)

RECORDED WITH

## Try Databricks for free







An open and unified data analytics platform for data engineering, data science, machine learning, and analytics. From the original creators of Apache Spark™, Delta lake, MLflow, and Koalas.



### Databricks trial:

- Collaborative environment for data teams to build solutions together.
- Interactive notebooks to use Apache Spark™, SQL, Python, Scala, Delta Lake, MLflow, TensorFlow, Keras, Scikit-learn and more.
- Available as a 14-day full trial in your own cloud, or as a lightweight trial hosted by Databricks.

### Used by:

### Please tell us about yourself

**First Name: \***

**Last Name: \***

**Company \***

**Company Email \***

**Title \***

**Phone Number**

☐ Keep me informed with occasional updates about Databricks and related open source products

By Clicking "Get Started For Free", you agree to the [Privacy Policy](#).

[GET STARTED FOR FREE](#)

## Choose a cloud provider



amazon Web Services

Microsoft Azure

Google Cloud Platform

Get started

By clicking "Get started", you agree to the [Privacy Policy](#) and [Terms of Service](#).

### Don't have a cloud account?

Community Edition is a limited Databricks environment for personal use and training.

[Get started with Community Edition](#)

By clicking "Get started with Community Edition", you agree to the [Privacy Policy](#) and [Community Edition Terms of Service](#).

# CREAR UNA CUENTA

Elegir opción correcta.



# CREAR UNA CUENTA

Comprobar nuestro email, nos llegará un mensaje



## Time to check your email!

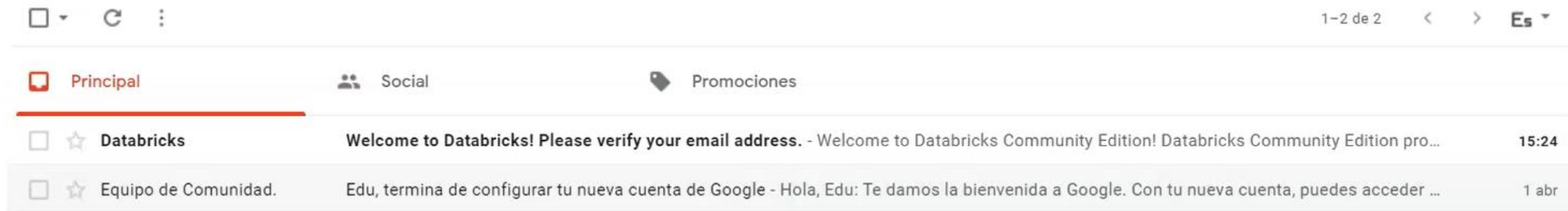
Thank you for signing up. Now it's time to validate your email address.  
Please check the email you provided for next steps.

© Databricks 2022. All rights reserved. Apache, Apache Spark, Spark and the Spark logo are trademarks of the [Apache Software Foundation](#).

[Privacy Policy](#) | [Terms of Use](#)

# CREAR UNA CUENTA

Le damos al primero de los enlaces:



Welcome to Databricks! Please verify your email address.

Recibidos x



**Databricks** <noreply@databricks.com>  
para mí ▾

15:24 (hace 2 minutos)



## Welcome to Databricks Community Edition!

Databricks Community Edition provides you with access to a free micro-cluster as well as a cluster manager and a notebook environment - ideal for developers, data scientists, data engineers and other IT professionals to get started with Spark.

We need you to verify your email address by clicking on [this link](#). You will then be redirected to Databricks Community Edition!


Get started by visiting: <https://community.cloud.databricks.com/login.html?resetpassword&username=edurf.cld%40gmail.com&expiration=-60000&token=801725227c11c256c073dc8f6229cc9cc3e309c5>

If you have any questions, please contact [feedback@databricks.com](mailto:feedback@databricks.com).

- The Databricks Team

# CREAR UNA CUENTA

Escribimos y confirmamos nuestra contraseña



Reset Password

Please enter your new password: \*

Please confirm your new password: \*

Reset password



# CREAR UNA CUENTA

Nos sale la ventana de inicio, ya tenemos nuestra cuenta de DCE.

The screenshot shows the Databricks Data Science & Engineering (DSE) dashboard. At the top, a blue banner indicates the user is using the Databricks Community Edition and provides a link to upgrade. The main content area is titled "Data Science & Engineering" and features four primary action cards: "Notebook" (with a "Create a notebook" link), "Data import" (with a "Browse files" link), "Partner Connect" (listing integrations like Fivetran, dbt, Tableau, and Power BI), and a "Guide: Quickstart tutorial" (with a "Start tutorial" link). Below these cards is a "Recents" section, which is currently empty, displaying a message "There are no recents yet". The bottom of the dashboard is divided into three columns of links: "Documentation" (including "Get started guide", "Best practices", "Data guide", and "More documentation"), "Release notes" (including "Runtime release notes", "Databricks preview releases", "Platform release notes", and "More release notes"), and "Blog posts" (listing several articles with dates and a "More blog posts" link). A left-hand sidebar contains navigation icons for workspace management, search, and other functions. The bottom of the image shows a Windows taskbar with various application icons and a system tray displaying the time as 15:30 on 03/04/2022.

# 3. Introducción a PySpark: RDDs



- **pyspark:** interfaz python a Spark.
- Nos permite ejecutar operaciones en paralelo (cluster) de forma sencilla.
- La gestión de la paralelización es transparente para el programador (si bien a bajo nivel tenemos la opción de variar algunos comportamientos)
- En nuestro código es necesario crear primero una sesión (crear “SparkSession” y además en su caso “SparkContext”). En entorno interactivo (notebook) ya creados, se accede mediante “**spark**” y “**sc**”

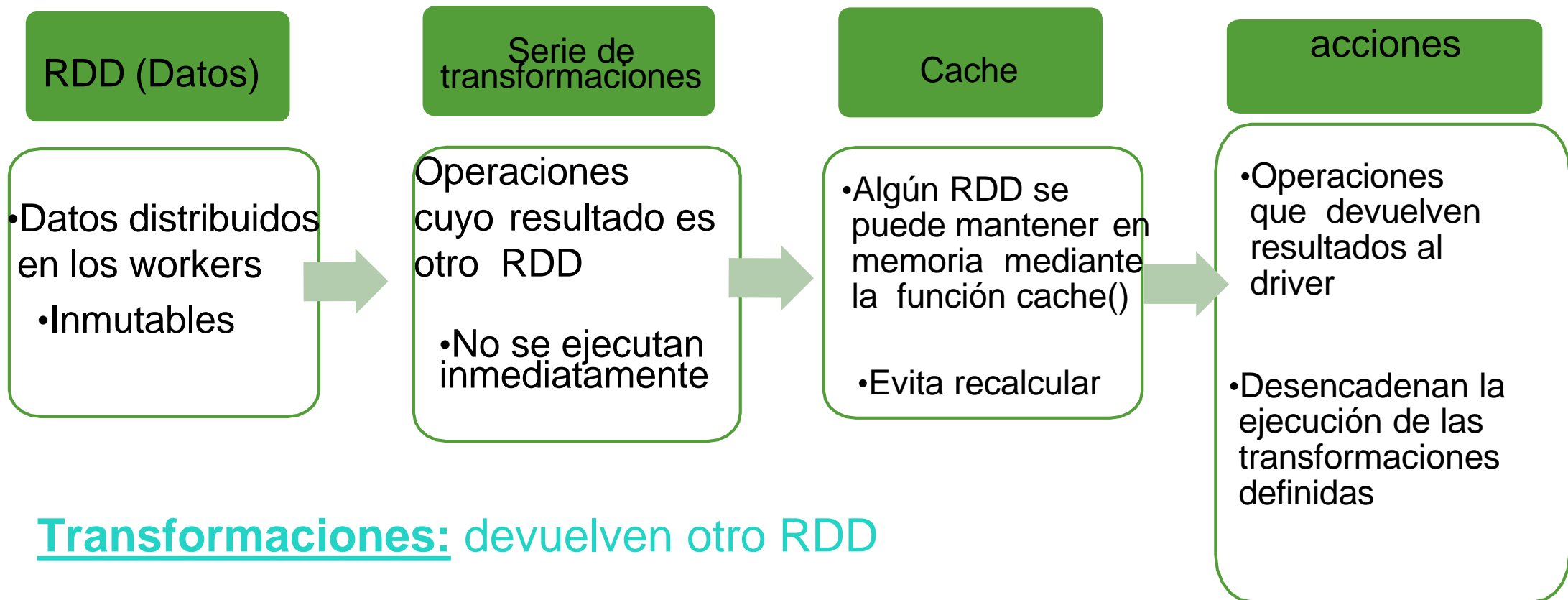
# PYSPARK: RDDs (Resilient Distributed Datasets)

➤ Trabajaremos en primer lugar sobre colecciones de datos denominadas RDD:

- ❑ Es el concepto básico de datos con el que trabaja Spark (estructura de bajo nivel de información).
- ❑ Su contenido está distribuido por el clúster en los nodos workers.
- ❑ Son inmutables. Una vez creados no se pueden modificar.
- ❑ Se pueden transformar para crear nuevos RDDs o realizar acciones sobre ellos pero no modificar.
- ❑ Sobre los RDDs hay dos tipos de operaciones principales:

**TRANSFORMACIONES y ACCIONES**

# SPARK RDDs: Realizar procesamiento, resolver problema



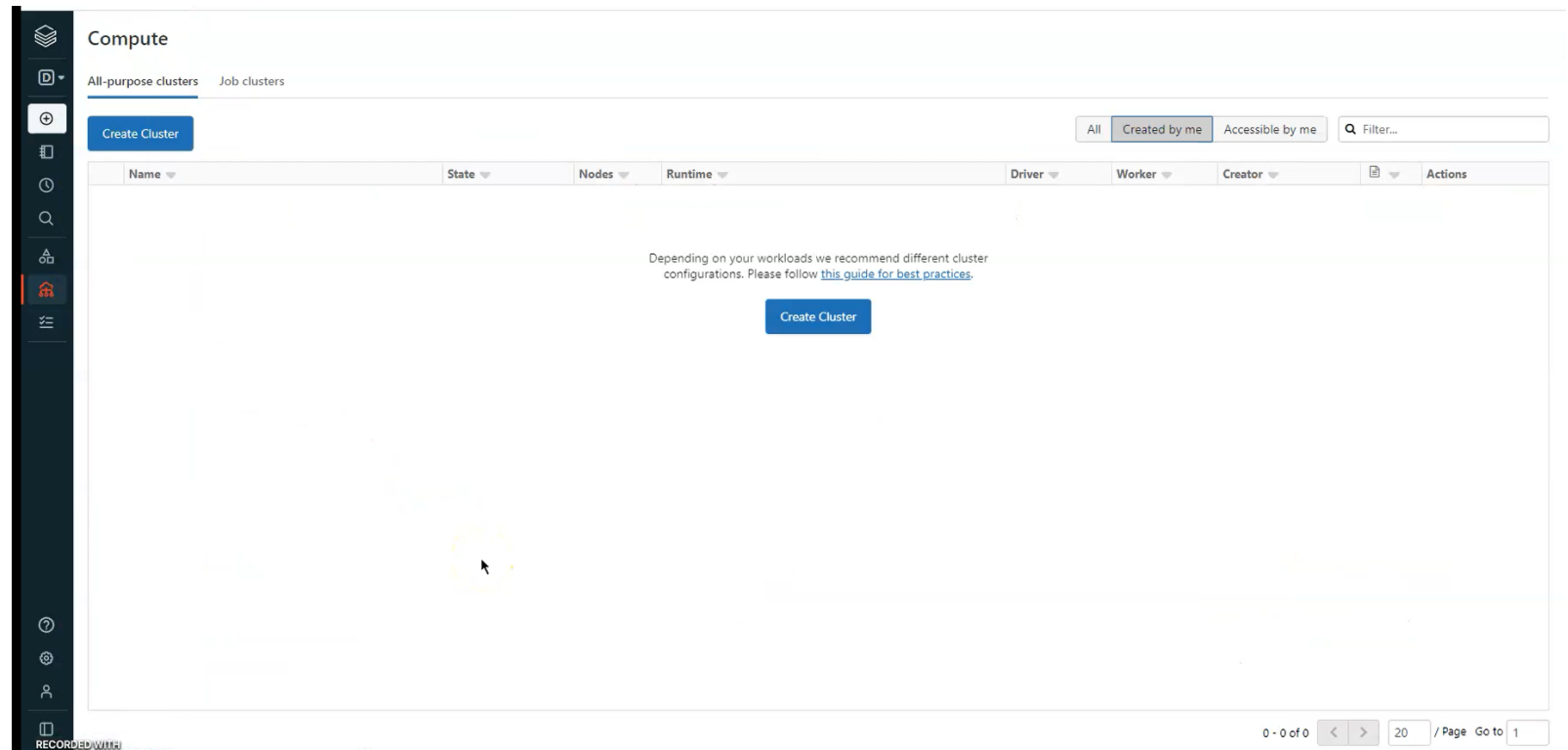
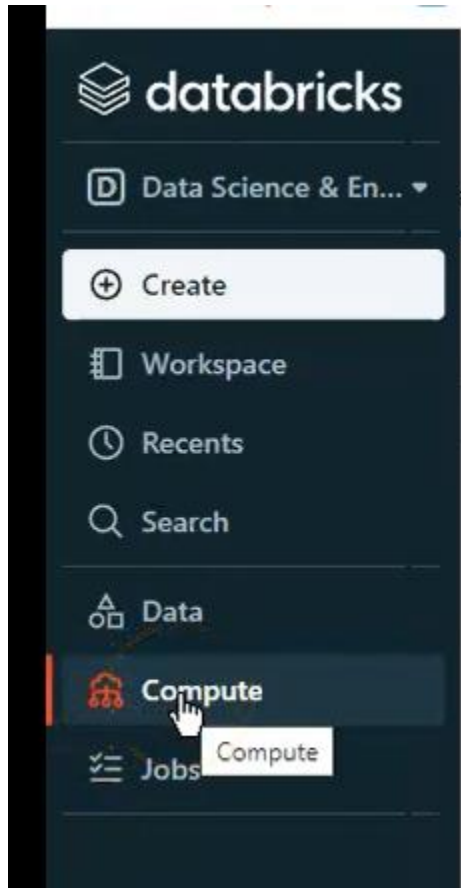
Transformaciones: devuelven otro RDD

Acciones: devuelven un valor

**Proceso para resolver un problema:** sucesión de transformaciones y acciones hasta alcanzar el resultado deseado

# DCE: lo primero es lo primero, CREAR un CLÚSTER

En el **panel de la izquierda** nos vamos a la opción **“Compute”**



# DCE: crear un clúster

Le damos al botón “**Create cluster**”

The screenshot displays the 'Compute' section of a web interface. On the left is a dark sidebar with various icons. The main area has a header 'Compute' and two tabs: 'All-purpose clusters' (selected) and 'Job clusters'. Below the tabs is a blue 'Create Cluster' button. To the right of this button are filter buttons: 'All', 'Created by me' (selected), and 'Accessible by me', followed by a search bar labeled 'Filter...'. Below the filters is a table with columns: Name, State, Nodes, Runtime, Driver, Worker, Creator, and Actions. The table is currently empty, displaying a message: 'Depending on your workloads we recommend different cluster configurations. Please follow [this guide for best practices](#).' A blue 'Create Cluster' button is centered in the table area. A large green arrow points to this button. At the bottom right, there is a pagination bar showing '0 - 0 of 0', navigation arrows, '20 / Page', and 'Go to 1'.

Compute

All-purpose clusters Job clusters

Create Cluster

All Created by me Accessible by me Filter...

Name	State	Nodes	Runtime	Driver	Worker	Creator	Actions
Depending on your workloads we recommend different cluster configurations. Please follow <a href="#">this guide for best practices</a> .							

Create Cluster

0 - 0 of 0 20 / Page Go to 1



# DCE: crear un clúster

Le ponemos nombre (resto por defecto) y “**Create cluster**”

**Create Cluster**

**New Cluster** Cancel Create Cluster **0 Workers:**0 GB Memory, 0 Cores, 0 DBU  
**1 Driver:**15.3 GB Memory, 2 Cores, 1 DBU ⓘ

**Cluster name**

mi\_cluster

**Databricks runtime version** ⓘ

Runtime: 9.1 LTS (Scala 2.12, Spark 3.1.2) | v

**Instance**

Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For more configuration options, please upgrade your Databricks subscription.

**Instances** **Spark**

**Availability zone** ⓘ

auto | v

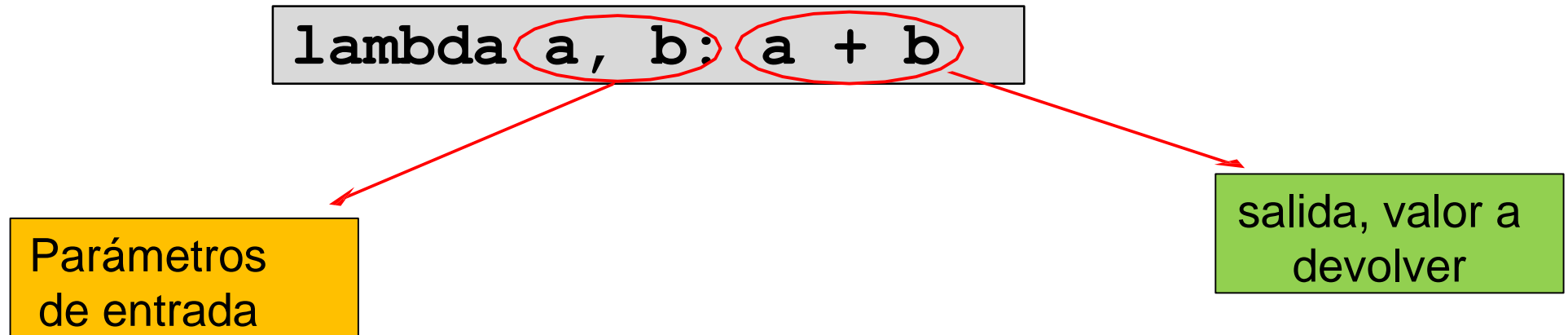
# DCE: crear un clúster

Tarda sobre unos 5 minutos, ya lo tenemos.

The screenshot shows the Databricks Clusters management interface. On the left is a dark sidebar with navigation icons. The main content area has a breadcrumb 'Clusters / mi\_cluster' and a cluster card for 'mi\_cluster' with a green status icon. Above the card are buttons for 'Edit', 'Clone', 'Restart', 'Terminate', and 'Delete'. Below the card is a horizontal menu with tabs: 'Configuration' (selected), 'Notebooks', 'Libraries', 'Event log', 'Spark UI', 'Driver logs', 'Metrics', 'Apps', and 'Spark cluster UI - Master'. The 'Configuration' tab shows the 'Databricks Runtime Version' as '9.1 LTS (includes Apache Spark 3.1.2, Scala 2.12)' and the 'Driver type' as 'Community Optimized' with '15.3 GB Memory, 2 Cores, 1 DBU'. A warning box states: 'Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For [more configuration options](#), please [upgrade your Databricks subscription](#).' Below this is a sub-menu with 'Instances' (selected), 'Spark', 'JDBC/ODBC', and 'Permissions'. The 'Instances' section shows the 'Availability zone' as 'us-west-2b'.

## Funciones lambda (o anónimas) de Python

- Son funciones anónimas. Por ejemplo, para sumar dos números:



- Se usan cuando hay que pasar una función como parámetro (transformaciones, acciones)
- Tienen una única instrucción cuyo valor corresponde al valor devuelto

# Crear un RDD a partir de una lista con sus elementos

- Crea un RDD a partir de una lista Python

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10], 2)
```

Lista de python.  
Puede ser de  
números,  
cadenas...

Número de  
particiones en que  
se dividirá la lista  
(opcional)

# SPARK: ejemplo de Evaluación Perezosa

```
# TODAVÍA NO SE REALIZA NINGUNA ACCIÓN (crear el RDD)

numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10,11])
```

```
# EVALUACIÓN PEREZOSA / DEMORADA

print(numeros.collect())
```



**Collect():** Acción para recuperar el contenido del RDD de los nodos workers, la usaremos para comprobar resultados

- Spark “va apuntando” las operaciones sobre los datos (**transformaciones**), no se ejecutan de forma inmediata.
- No se calcula nada hasta que es necesario (cálculo de un valor, mostrar en dispositivo de salida, **acciones**)

# 4. RDDs: Transformaciones



# RDDs: Transformaciones

- Crean un RDD a partir de otro u otros RDDs
- Evaluación perezosa. No se calculan los resultados inmediatamente.
- Spark apunta la serie de transformaciones que se deben aplicar para ejecutar después.
- Es como una receta

```
lineas.flatMap(...).filter(...).map(...).reduceByKey(...)
```

\* Ojo, notación punto



# RDDs: Transformaciones más comunes

Transformación	Descripción
map(func)	Crea un nuevo RDD a partir de otro aplicando una transformación a cada elemento original
filter(func)	Crea un nuevo RDD a partir de otro manteniendo solo los elementos de la lista original que cumplan una condición
flatMap(func)	Como map pero cada elemento original se puede mapear a 0 o varios elementos de salida
distinct()	Crea un nuevo RDD a partir de otro eliminando duplicados
union(otroRDD)	Une dos RDD en uno
sample()	Obtiene un RDD con una muestra obtenida con reemplazamiento (o sin) a partir de otro RDD.

# Transformación “map()”

- Aplica una transformación a cada elemento del RDD original


```
numeros = sc.parallelize([1,2,3,4,5])  
  
num3 = numeros.map(lambda elemento: 3*elemento)
```

Función que se aplica a cada elemento del rdd números

- **Resultado:** [1,2,3,4,5] → [3,6,9,12,15]
- La función que se pasa a map debe:
  - Recibir un único parámetro, que serán elementos individuales del rdd de partida
  - Devolver el elemento transformado

# Transformación: cuestiones sobre “map()”

- ¿Cuál es el tamaño del rdd de salida?
- El mismo que el tamaño de entrada




```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])  
  
pal_minus = palabras.map(lambda elemento: elemento.lower())  
  
pal_minus.collect()
```

RDD: palabras

```
['HOLA',  
'Que',  
'TAL',  
'Bien']
```

RDD: pal\_minus



```
['hola',  
'que',  
'tal',  
'bien']
```



```
['hola', 'que', 'tal', 'bien']
```

# Transformación “filter()”

- Filtra un RDD manteniendo solo los elementos que cumplan una determinada condición


```
numeros = sc.parallelize([1,2,3,4,5])  
  
pares_rdd = numeros.filter(lambda elemento: elemento%2==0)
```

Función que se aplica a cada elemento para filtrarlo

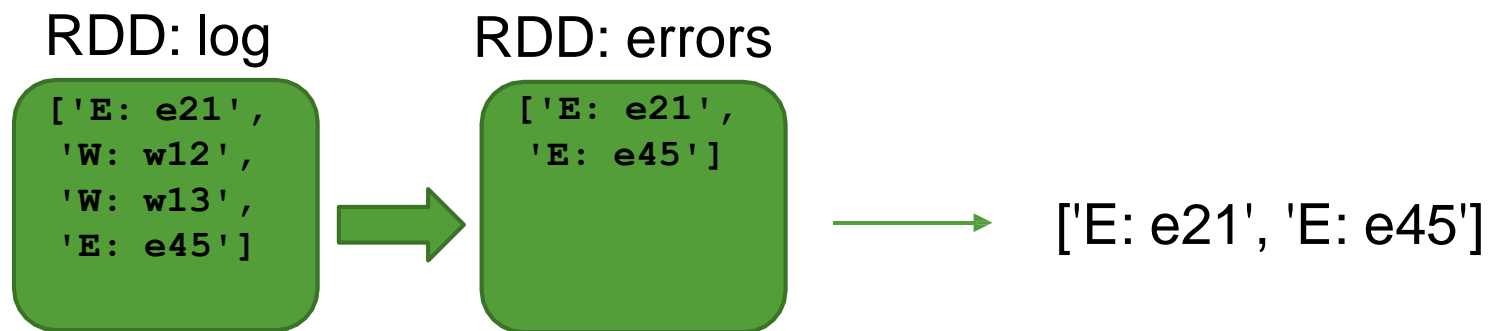
- Resultado: [1,2,3,4,5] → [2,4]
- **La función que se pasa a filter debe:**
  - Recibe un único parámetro, que serán los elementos individuales del rdd de partida
  - Devolver True o False para indicar si el elemento pasa o no el filtro

# Transformación: cuestiones sobre “filter()”

- ¿Cuál es el tamaño del rdd de salida?
  - Menor o igual que el original



```
log = sc.parallelize(['E: e21', 'W: w12', 'W: w13', 'E: e45'])  
  
errors = log.filter(lambda elemento: elemento[0]=='E')  
  
errors.collect()
```



# Transformación “flatMap()”

- Como map pero por cada elemento puede devolver más elementos (lista)
- Devuelve el resultado “plano” (flat), sin diferenciar la salida de cada elemento (corchetes, etc). Útil cuando se utilizan funciones que devuelven listas, tuplas...

```
numeros = sc.parallelize([1,2,3,4,5])
```

```
rdd = numeros.flatMap(lambda elemento : [elemento, 10*elemento])
```

- Resultado → [1, 10, 2, 20, 3, 30, 4, 40, 5, 50]
- **La función que se pasa a flatMap debe:**
  - Recibir un único parámetro, que serán elementos individuales del rdd de partida
  - Devolver una lista de elementos (corchetes)

# Diferencias entre “flatMap()” y “map()”

```
lineas = sc.parallelize(['', 'a', 'a b', 'a b c'])  
  
palabras_map = lineas.map(lambda elemento: elemento.split())  
  
palabras_flat = lineas.flatMap(lambda elemento: elemento.split())
```

La función split() devuelve una lista con las subcadenas separadas por espacio (defecto)

- Con map → `[[], ['a'], ['a', 'b'], ['a', 'b', 'c']]`
- Con flatMap → `['a', 'a', 'b', 'a', 'b', 'c']`
- De aquí viene lo de *flat*, la lista de flatmap se ‘aplana’

# Transformación “distinct()”

- Crea un nuevo RDD eliminando duplicados

```
numeros = sc.parallelize([1,1,2,2,5])  
  
unicos = numeros.distinct()  
unicos.collect()
```

- Resultado: [1,1,2,2,5] → [1, 2, 5]



## Transformación “sample()”

- Devuelve una muestra del RDD de entrada, con o sin reemplazamiento.
- El primer parámetro indica si hay reemplazamiento
- El segundo parámetro indica la proporción (entre 0 y 1) de datos **aproximados** que se seleccionan.

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
```

```
muestra = numeros.sample(False, 0.5)
```

- Resultado -> [1,4,6,9] (aleatorio)
- Cada ejecución da un resultado distinto
- **Es útil cuando hay un número de datos demasiado elevado, en tareas de prueba o depuración.**

# Transformación “union()”

- Une dos RDDs en uno

```
pares = sc.parallelize([2,4,6,8,10])  
impares = sc.parallelize([1,3,5,7,9])  
  
numeros = pares.union(impares)
```

- Resultado: → [2, 4, 6, 8, 10, 1, 3, 5, 7, 9]

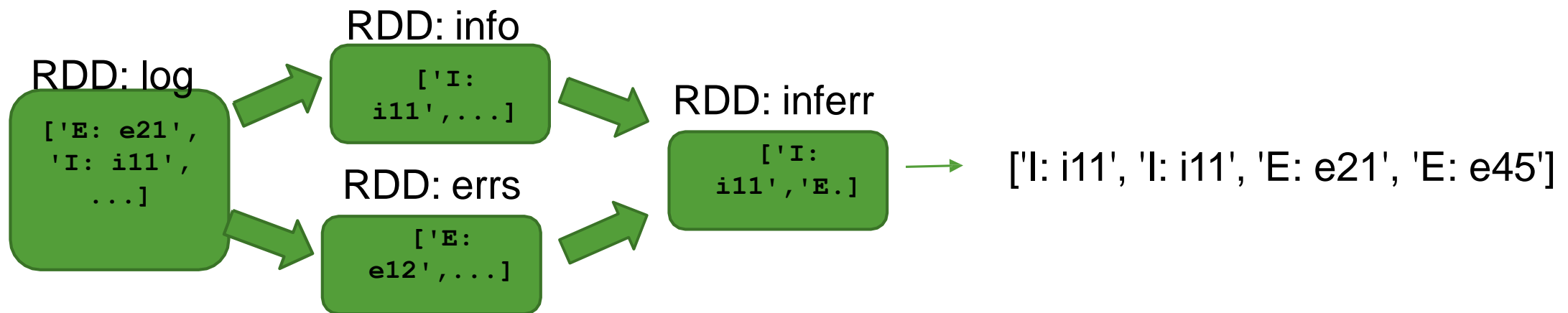
# Transformación “union()”: ejemplo de uso sencillo

```
log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13', 'E: e45'])
```

```
info = log.filter(lambda elemento: elemento[0]=='I')
```

```
error = log.filter(lambda elemento: elemento[0]=='E')
```

```
inferr = info.union(error)
```

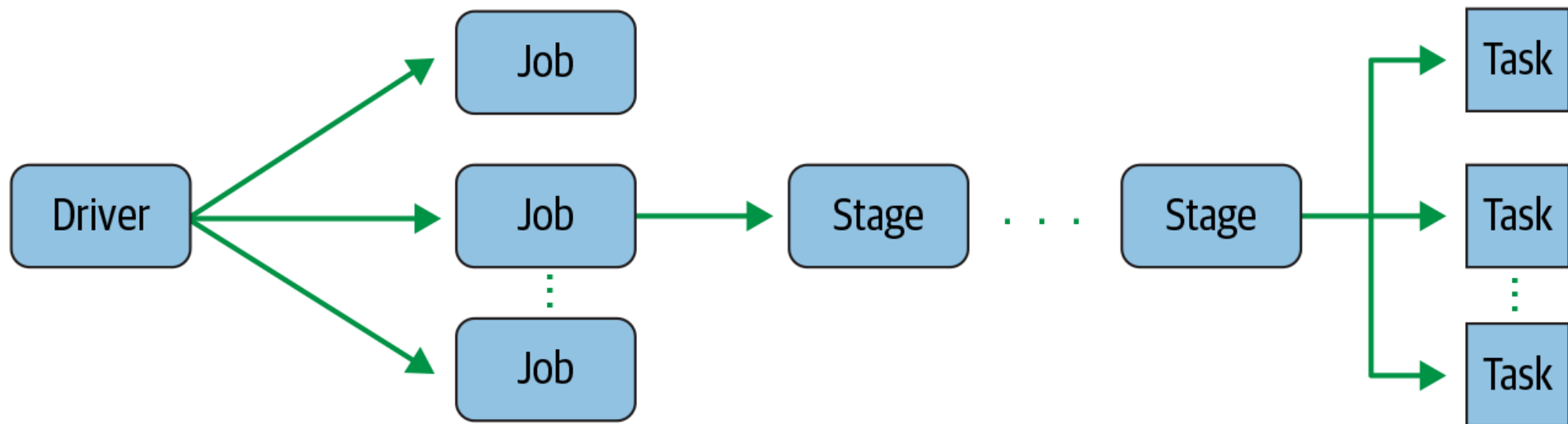


# Concepto de DAG (Direct Acyclic Graph)

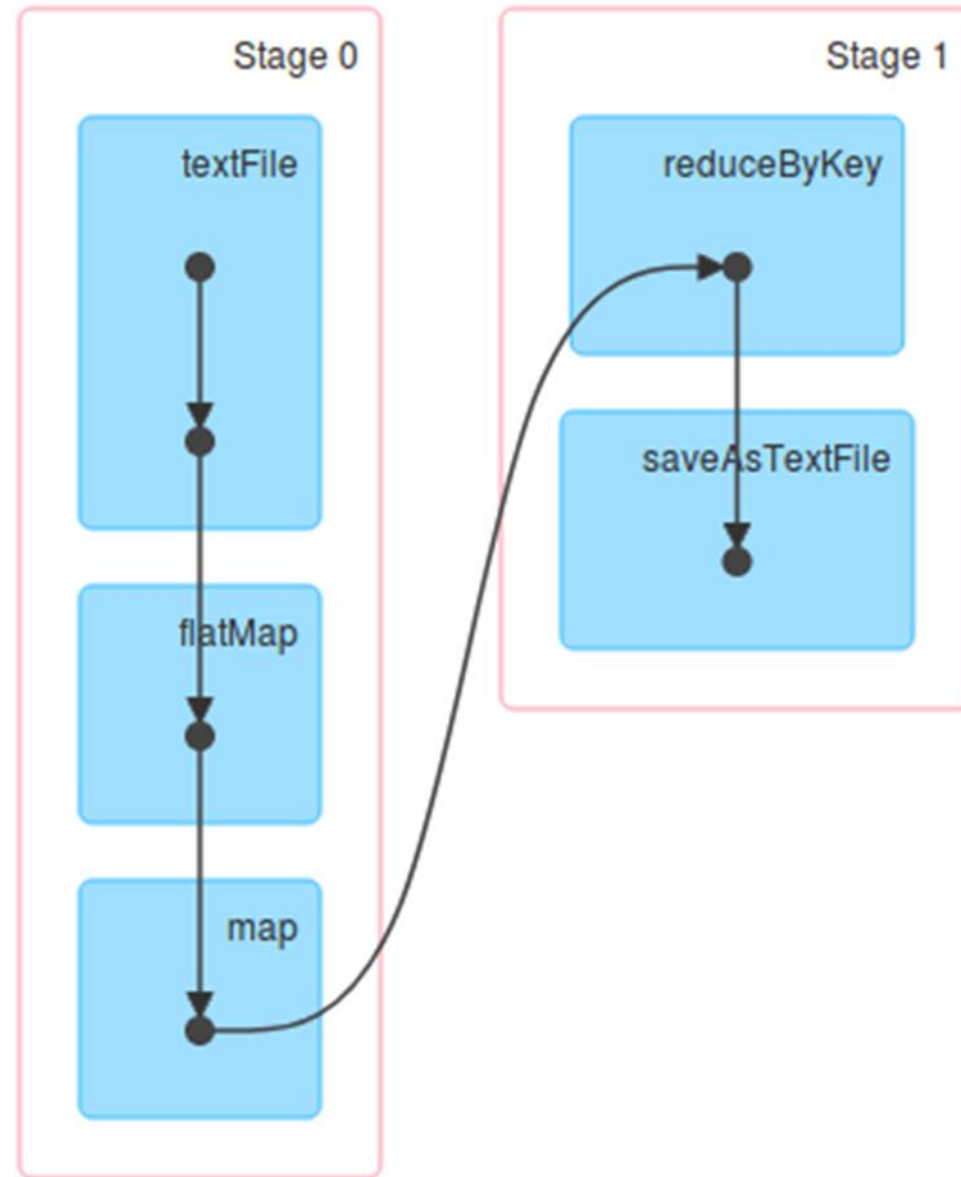
- Spark “construye” o “planifica” sus operaciones (p. ejplo. sobre RDDs o Dataframes) como un grafo acíclico dirigido (DAG).
- Su planificador DAG y el optimizador de consultas construyen un grafo computacional eficiente, que se puede descomponer en tareas que se ejecutan en paralelo entre los workers del clúster.
- Podemos utilizar las APIs para escribir una aplicación Spark (Java, R, Scala, SQL o Python) y Spark la convierte en el mismo DAG.
- El código se ejecuta en los nodos workers del clúster (llevamos el programa a los datos, inversa clásico)

# SPARK: Jobs, Stages, Task

- **Job:** Una computación (operación) paralela que consiste en múltiples tareas que se generan en respuesta a una acción de Spark (por ejemplo, `count()`, `collect()`).
- **Stage:** Cada trabajo (Job) se divide en conjuntos más pequeños de tareas llamadas fases (stages ) que dependen unas de otras.
- **Task:** Unidad más sencilla de trabajo o ejecución que se enviará a un executor de Spark.



# Ejemplo DAG (operaciones clave-valor)





red.es



UNIÓN EUROPEA

*"El FSE invierte en tu futuro"*

**Fondo Social Europeo**

