Introducción a Spark

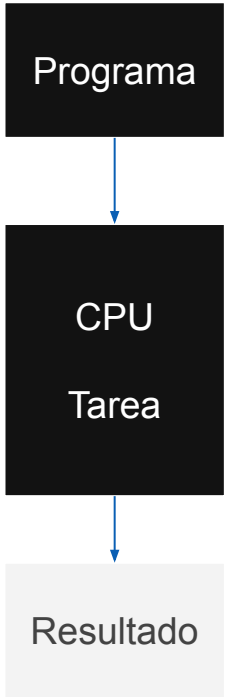
Introducción a Spark

01

Introducción

Procesamiento en local

El **procesamiento local es el procesamiento clásico. Con sus limitaciones de computación y almacenamiento. R, Python, ...**



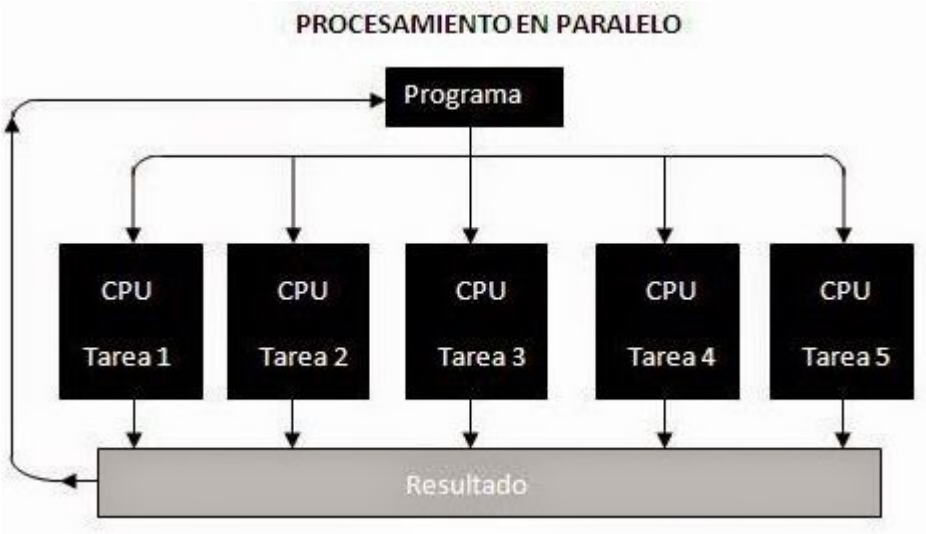
¿Qué es un *cluster*?

Un **cluster** es un grupo de múltiples ordenadores (o máquinas virtuales) unidos mediantes una red de alta velocidad tal que el conjunto es visto como un único ordenador. Un buen cluster se caracteriza por tener:

- Alto rendimiento
- Alta disponibilidad
- Equilibrio de carga
- Escalabilidad

¿Qué es el procesamiento en paralelo?

En el **procesamiento en paralelo** un problema grande, en caso que sea posible, se divide en otros más pequeños que luego se ejecutan y resuelven simultáneamente en distintos nodos.



¿Qué es un nodo?

Un ***nodo*** es cada uno de los ordenadores (o máquinas virtuales) que forman parte de un cluster. Hay distintos tipos de nodos, entre los que se hallan:

- *master:* supervisa el almacenamiento de los datos y de que los cómputos de los datos se lleven a cabo en paralelo.
- *worker:* almacena los datos y lanza cálculos. Recibe instrucciones del nodo *master*.

¿Qué es MapReduce?

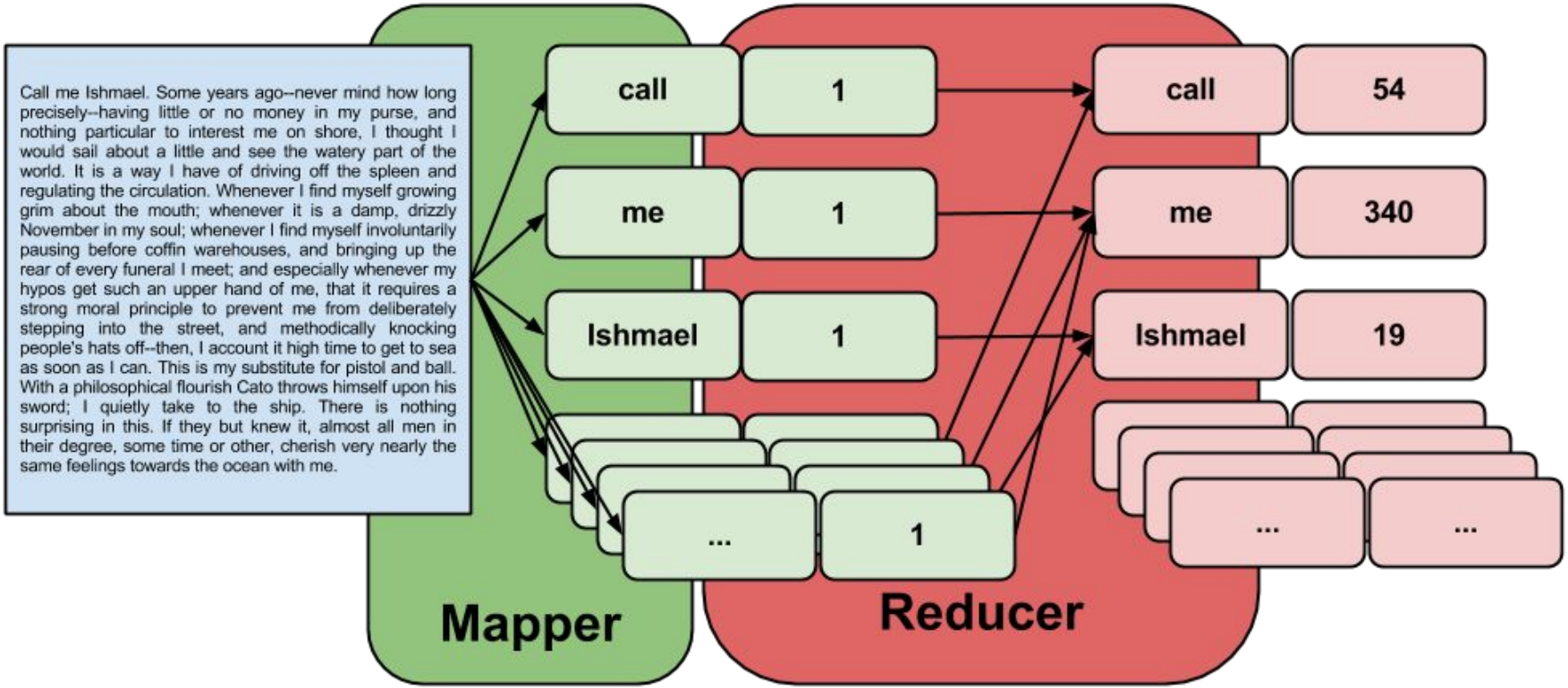
MapReduce procesa conjuntos de datos en paralelo y de forma distribuida. Consta de dos etapas principales:

- *map*: se toma un conjunto de datos y se convierte en otro donde cada elemento se convierte en una tupla (clave/valor).

- *reduce*: se toma el resultado del *map* y se combinan las tuplas en conjuntos más pequeños de tuplas.

¿Qué es MapReduce?

Ejemplo



¿Qué es MapReduce?

02

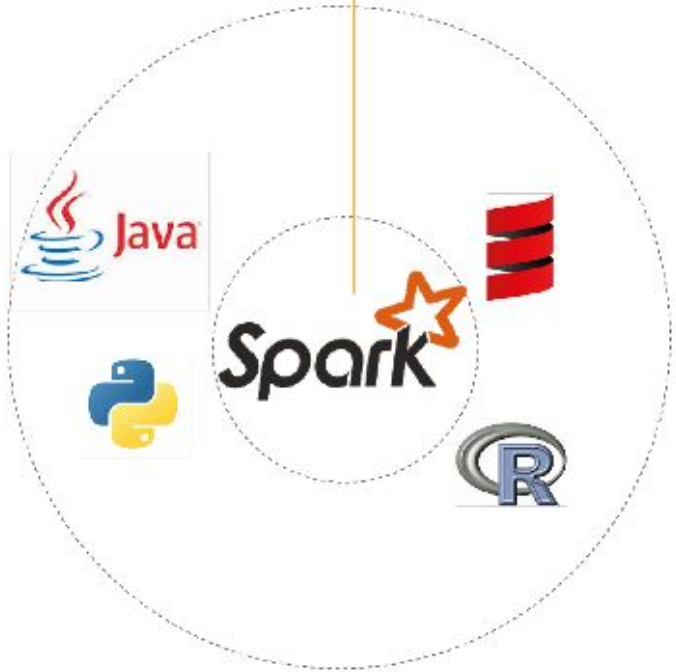
Contexto

¿Qué es Spark?

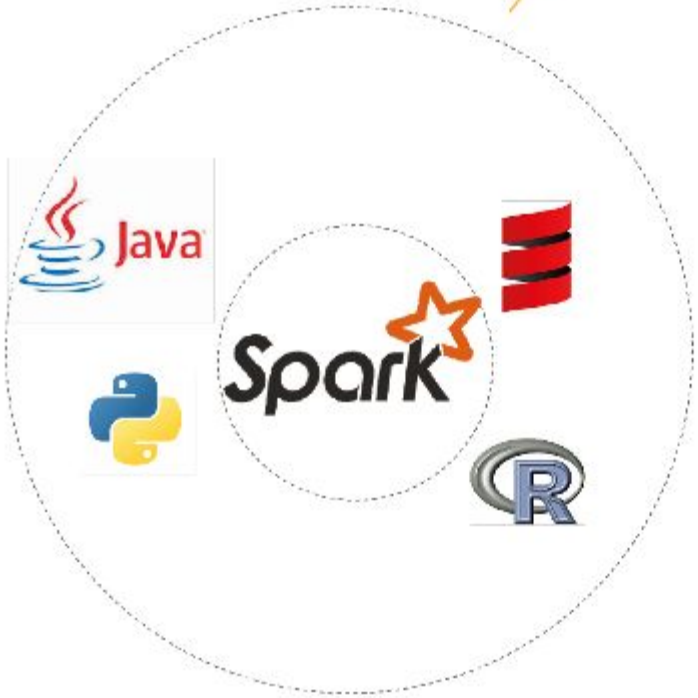
Spark es un framework de propósito general para el procesamiento de datos en un cluster (en paralelo).

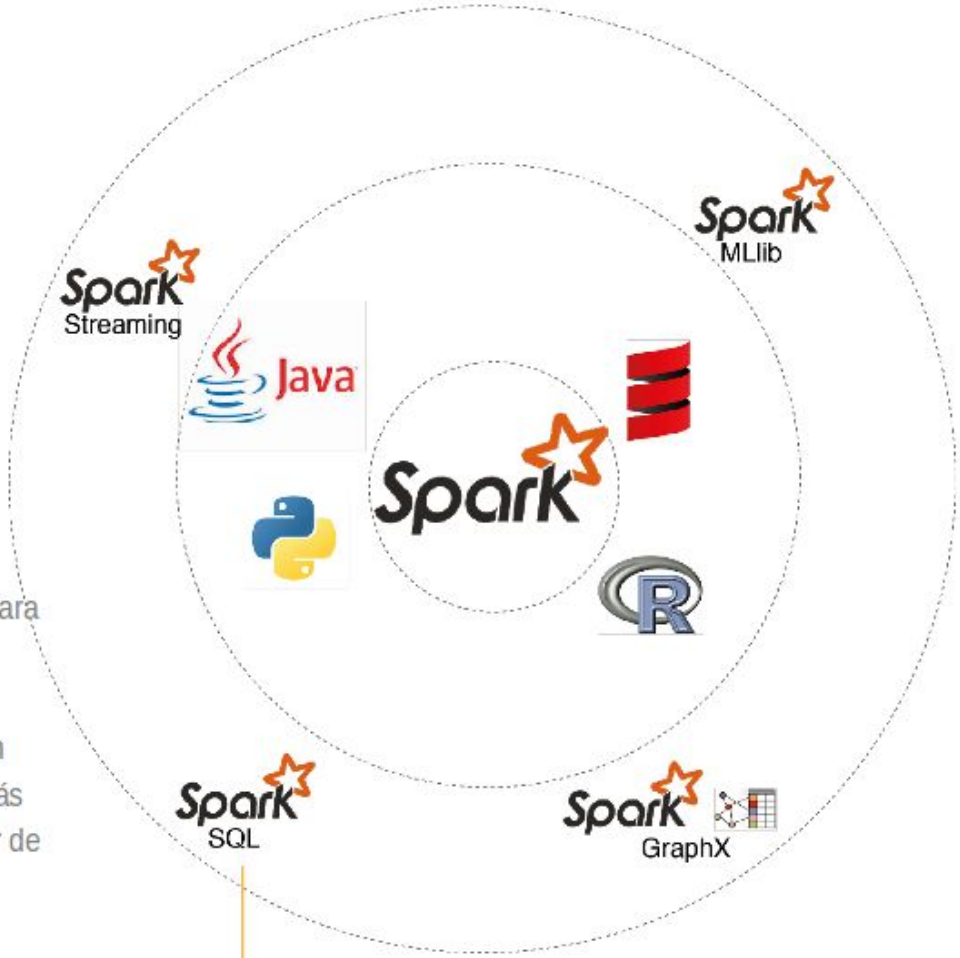
Spark Core

contiene las funcionalidades básicas de Spark. Esto incluye tareas de planificación, manejo de memoria, recuperación de fallos, interacción con sistemas de almacenamiento, entre otras.



Spark posee APIs: Scala, Java, Python (PySpark), y R





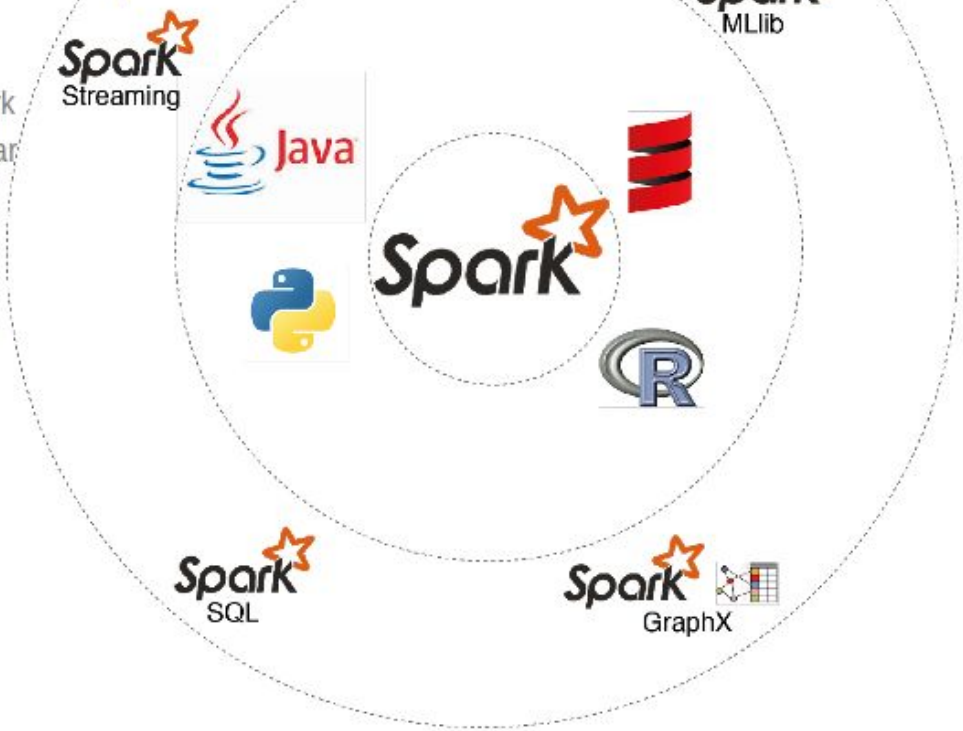
Spark SQL

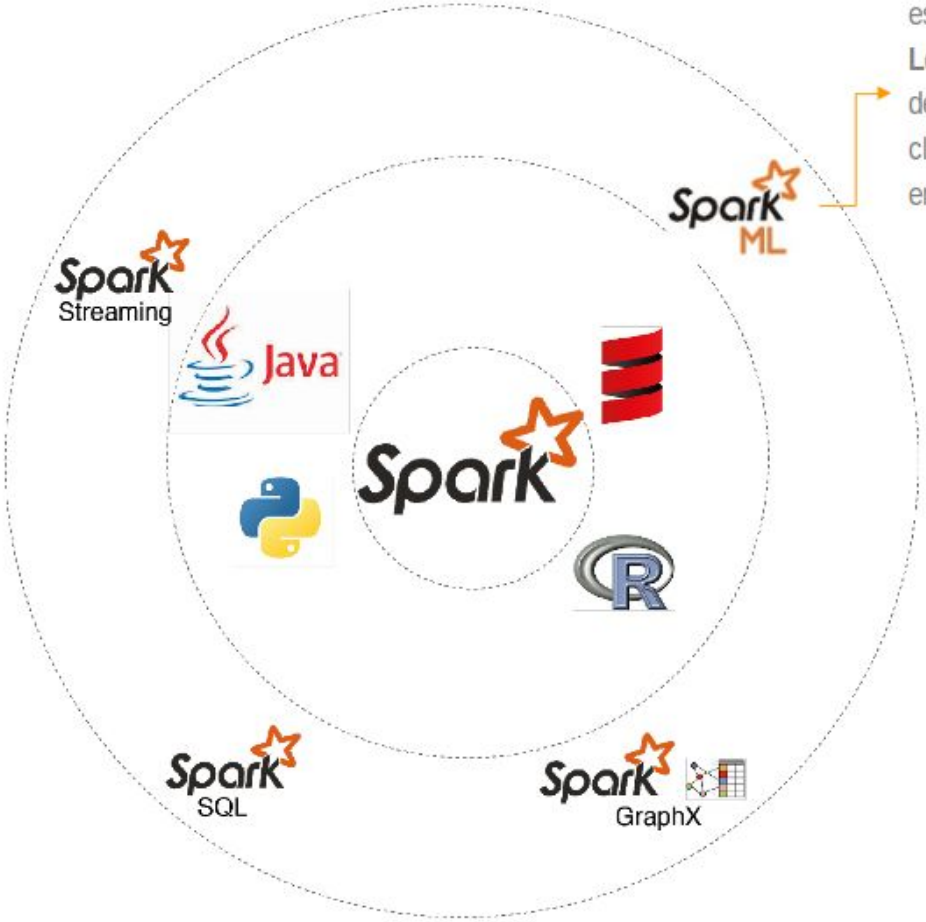
es una extensión de Spark para el procesamiento de datos estructurados. Provee una abstracción de programación llamada **DataFrames**, además puede actuar como un motor de consultas distribuidas SQL.

Spark Streaming

es una extensión del core Spark API. Permite procesar y analizar flujos de datos en tiempo real. Puede leer datos :

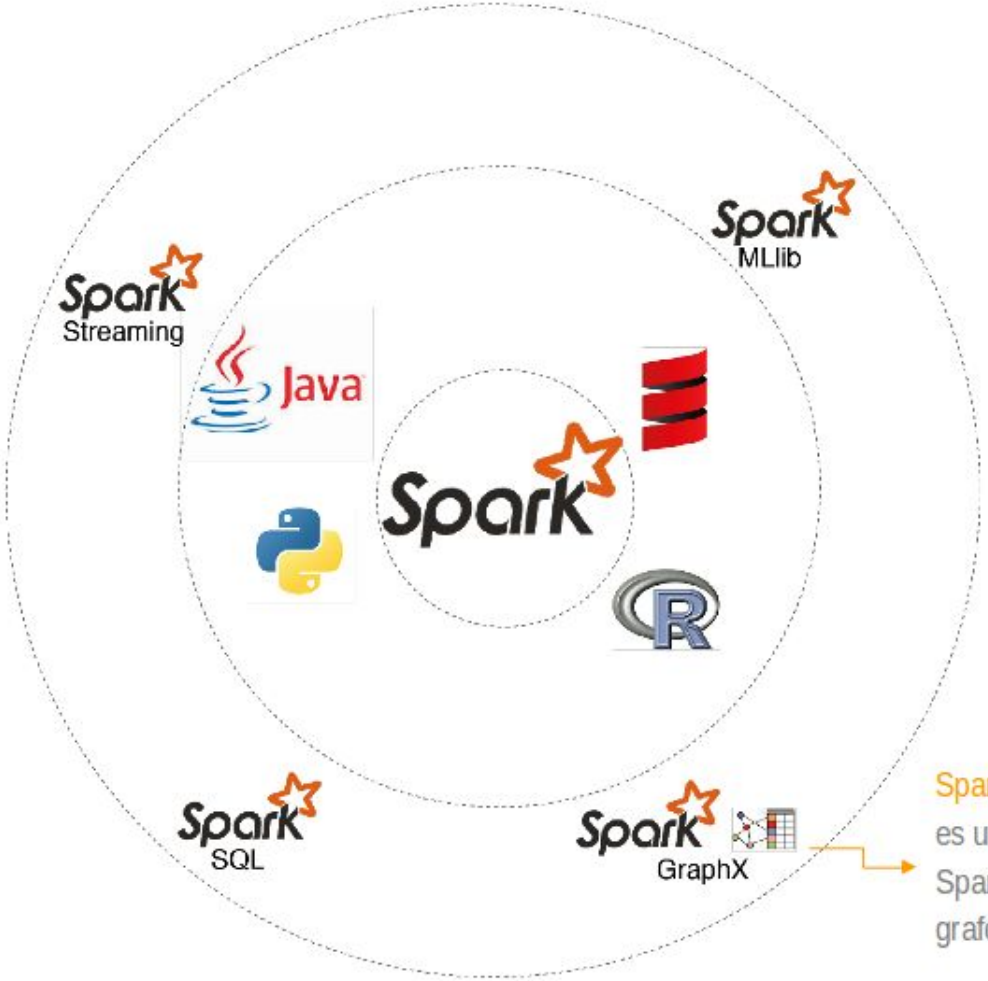
HDFS, Flume, Kafka, Twitter y Amazon Kinesis





Spark ML

es la biblioteca de **Machine Learning**, incluye algoritmos de clasificación, regresión, clustering, recomendación, entre otras utilidades.



Spark GraphX /GraphFrames es una extensión de la API de Spark para la computación de grafos en paralelo.

Componentes principales

Se orienta en torno a 3 elementos

- Estructura de datos
  - RDD, DataFrame o Dataset
- Ejecución Lazy (Perezosa)
  - Se secuencia el trabajo en transformaciones, que se ejecutan cuando se indica una acción
- DAG (Directed Acyclic Graph)
  - La secuencia de transformaciones componen jobs definidos en DAGs. Nuevos componentes de Spark se encargan de optimizar las operaciones internas.

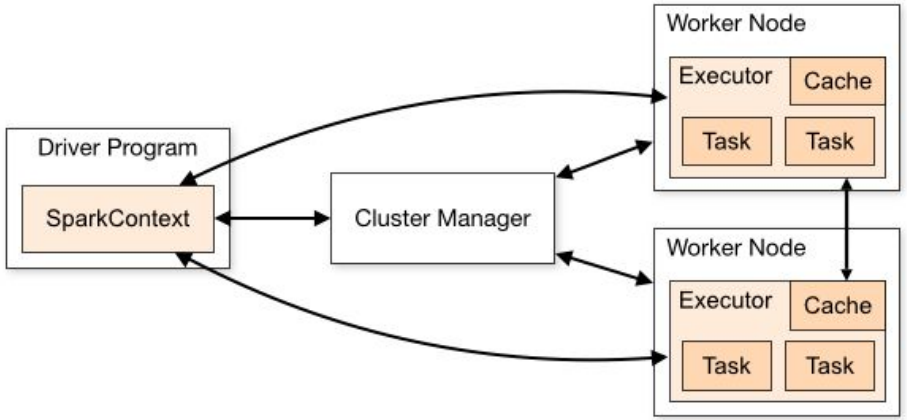
03

Arquitectura

Arquitectura *cluster* Spark

Spark Cluster

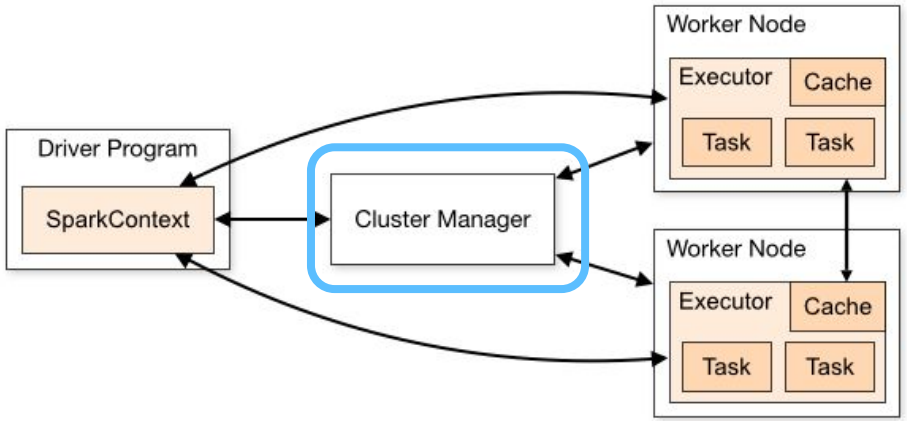
Spark Cluster es un conjunto de máquinas o nodos donde se encuentra instalado Spark. Estas máquinas incluyen a los Spark Workers, un Spark Master, y por lo menos un Spark Driver.



Arquitectura *cluster* Spark

Spark Master / Cluster Manager

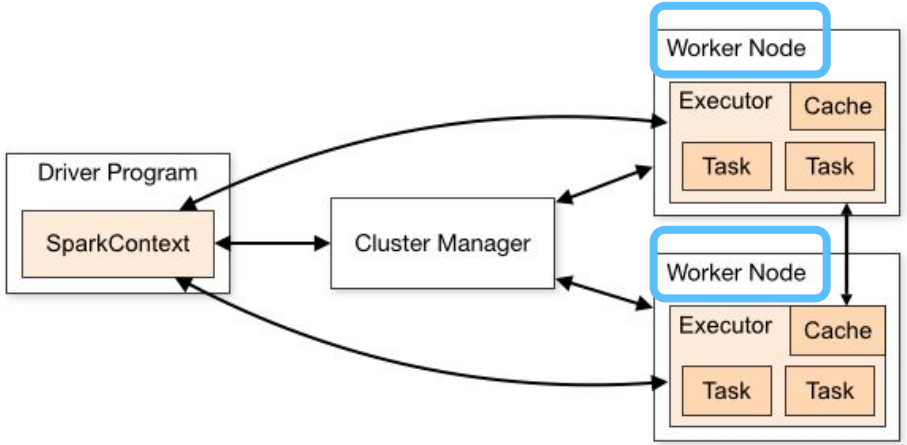
Spark Master / Cluster Manager, dependiendo del modo de despliegue, actúa como un gestor de recursos encargado de decidir cuando y cuantos executors son lanzados en los workers del cluster.



Arquitectura *cluster* Spark

Spark Worker

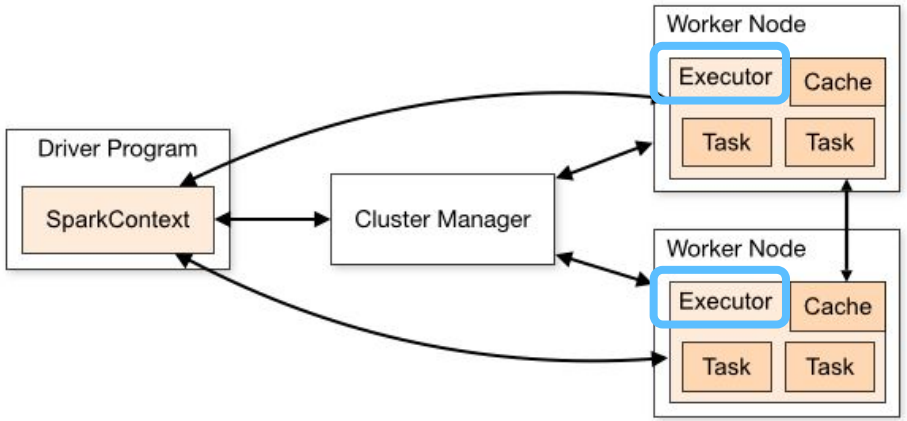
La función principal de un **Spark Worker** es instanciar y lanzar los executors en nombre del Spark Context.



Arquitectura *cluster* Spark

Spark Executor

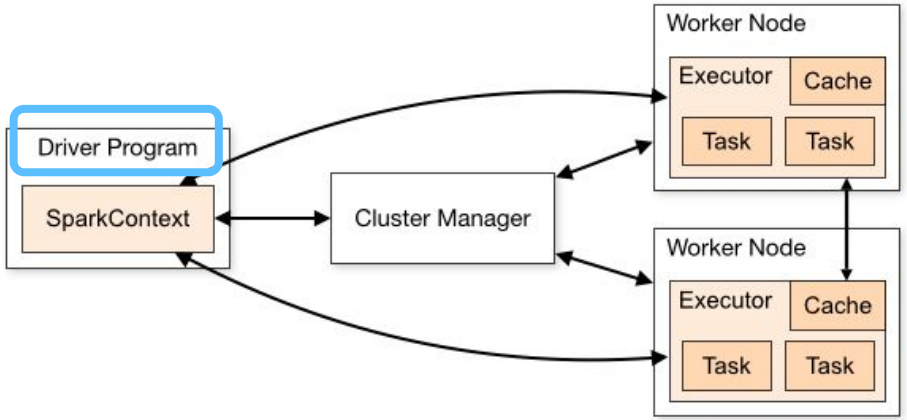
Un **Spark Executor** está encargado de ejecutar tareas asignadas por el Spark Context y almacenar particiones de datos en memoria.



Arquitectura *cluster* Spark

Spark Driver

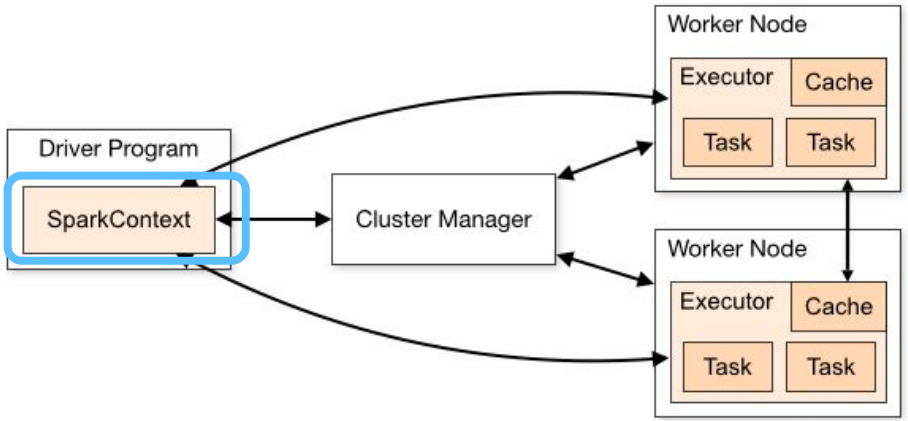
Al recibir del Spark Master la información sobre los workers, el **Spark Driver** distribuye las tareas a los executors de cada worker. El driver también recibe desde los executors el resultado de las tareas.



Arquitectura *cluster* Spark

SparkContext

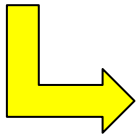
Para usar Spark y sus APIs es necesario instanciar un **SparkContext**.
El SparkContext representa la conexión al cluster, diciéndole a Spark cómo y dónde acceder a él.
El Spark Driver lo utiliza para comunicarse con el Cluster Manager y enviar jobs a los workers. Además permite configurar los parámetros de Spark.



Arquitectura *cluster* Spark

SparkContext

Siempre se trabaja con el **mismo SparkContext en un cluster** a menos que explícitamente se indique lo contrario.



En principio sólo puede haber un SparkContext por JVM. El SparkContext que se inicializa casará con el JVM del Spark Driver. Éste necesita un JVM para poder entender los comandos de Spark mediante Java.

Arquitectura *cluster* Spark

SparkSession

SparkSession representa un punto de entrada único a todas las funcionalidades de Spark e incluye las variables de contexto necesarias para manipular datos.

Al inicializar una **SparkSession** también se está inicializando a la par un **SparkContext.**

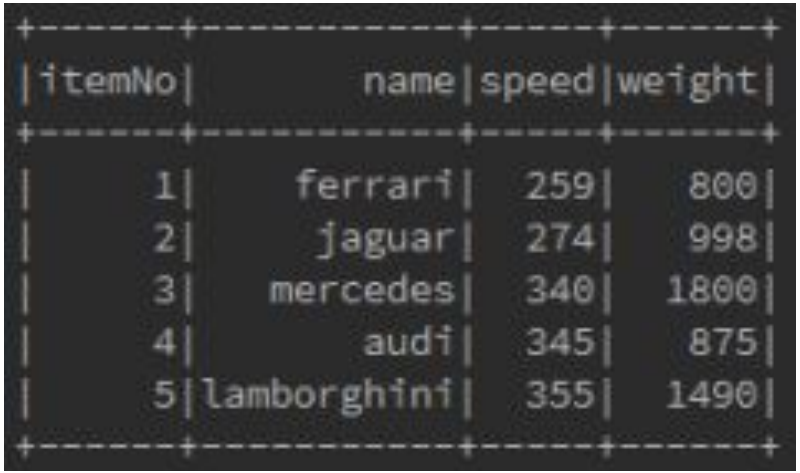
04

Estructuras de datos

Estructuras de datos

DataFrame

Un **DataFrame** es una colección **inmutable** y distribuida de datos organizados en columnas de forma tabular que permite una abstracción de los datos en alto nivel.



itemNo	name	speed	weight
1	ferrari	259	800
2	jaguar	274	998
3	mercedes	340	1800
4	audi	345	875
5	lamborghini	355	1490

DataFrame

Características

Las características principales de los DataFrames son las siguientes:

- Conjunto de filas (***Rows***) con un esquema (***Schema***)
- Colección distribuida de datos organizados en filas y columnas nombradas.
- Conceptualmente equivalente a una tabla en una base de datos relacional, o a un *dataframe* de R o Python, pero con optimizaciones avanzadas para soportar aplicaciones Big Data.
- Pueden construirse a partir de distintas fuentes: ficheros de datos estructurados, tablas de Hive, bases de datos externas, o ***RDDs***.
- DataFrame API disponible en Scala, Java, R y ***Python (PySpark)***.

DataFrame

Limitaciones

Las limitaciones principales de los DataFrames son las siguientes:

- Posee límites en manipular datos **cuando la estructura no es conocida** (compile-time type safety)
  e.g. si se intenta acceder a una columna que no existe, no se detectará el error hasta la ejecución del programa.

Estructuras de datos

RDD

Un **RDD**, *Resilient Distributed Dataset,* es la unidad fundamental de datos en Spark. Consiste en una colección inmutable y distribuida de objetos que pueden ser almacenados en memoria o disco.

05

Patrones de diseño

Patrones de diseño

Lazy Evaluation

La ***lazy evaluation*** es una estrategia de evaluación que retrasa el cálculo de una expresión hasta que su valor sea necesario. También evita repetir la evaluación en caso de no sea necesaria en posteriores ocasiones.

La API de Spark core API divide sus funcionalidades entre **transformaciones** y **acciones**. Hasta que no se lanza una acción, Spark no evalúa el programa y se construye un grafo acíclico dirigido (DAG) con todas las transformaciones.

Lazy Evaluation

Transformaciones vs Acciones

Transformaciones	Acciones
● Definen la lógica para el procesamiento de los datos ● Se utilizan para: - *mapear* - agregar - hacer *joins* - ordenar - hacer operaciones de conjuntos - controlar las particiones	● Ejecutan el DAG creado por transformaciones. ● Se utilizan para: - convertir a tipo *array* - vista previa de datos - guardar

Lazy Evaluation

Tipos de transformaciones

map	filter	flatMap	mapPartitions	mapPartitionswithIndex
sample	union	intersection	distinct	aggregateByKey
groupByKey	reduceByKey	sortByKey	join	repartitionAndSortWithinPartitions
repartition	coalesce	pipe	cartesian	cogroup

Lazy Evaluation

Tipos de acciones

reduce	collect	count	first
take	takeSample	takeOrdered	saveAsTextFile
saveAsSequenceFile	saveAsObjectFile	countByKey	foreach

Lazy Evaluation

Ventajas

- **Aumento de la manejabilidad**: se puede organizar el programa en operaciones más pequeñas. Se reduce el número de iteraciones sobre los datos mediante la agrupación de operaciones.
- **Ahorro en computación e incremento en velocidad**: ahorra cálculos puesto que sólo los valores necesarios se computan. Ahorra también el viaje entre el driver y el cluster, lo que acelera el proceso.
- **Reducción de complejidad**: las complejidades principales de una operación son tiempo y espacio. Al no ejecutar cada operación, se reduce el tiempo. Permite trabajar con una estructura de datos infinita, puesto que las acciones se llevan a cabo sólo cuando se necesitan los datos.
- **Optimización**: la proporciona al reducir el número de *queries*.

NOTEBOOK

Conceptos básicos de RDDs

Introducción a Spark

RDDs

Thanks