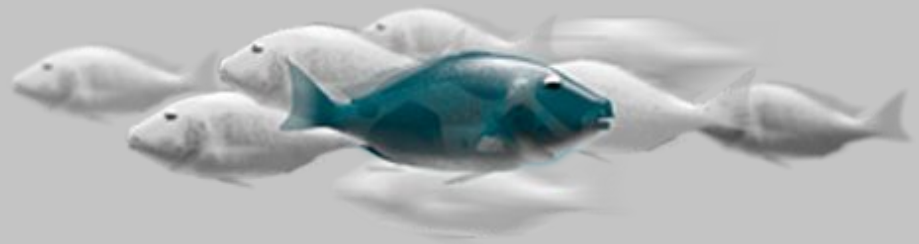
EDEM



Máster en Big Data & Cloud | VII Edición

Pyspark

Ignacio Reyes Vázquez

0. Presentación

EDUCACIÓN

- Administración y Dirección de Empresas, especializado en Business Intelligence (UPV) 2019-2023
- Máster en Data Analytics para la Empresa (EDEM) 2023-2024
- Máster en Project Management (UNIR) 2024-2025

TRAYECTORIA PROFESIONAL

- Data Scientist (Beca de Colaboración UPV) 2022-2023
- Data Scientist/Data Engineer (NTT DATA) 2023-2024
- Data Architect (BBVA TECHNOLOGY) 2024-Act.

https://www.linkedin.com/in/ignacio-reyes-vazquez/

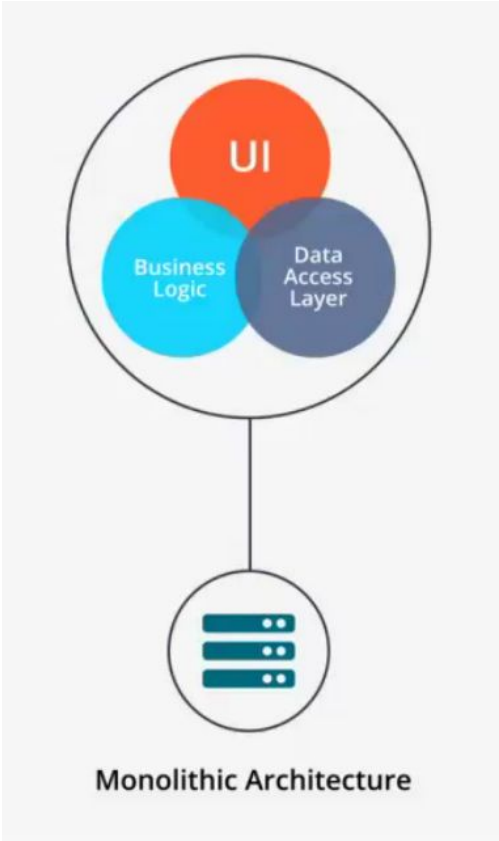
1. Antes de Apache Spark

Arquitecturas monolíticas VS distribuidas

Arquitectura Monolítica:

Todo el código de la aplicación está en un solo bloque. Todas las partes (interfaz, lógica de negocio, base de datos) están juntas.

- Pros: fácil de desarrollar al principio. Contras: difícil de escalar y de actualizar.
- Ejemplo: Una app de notas donde crear, editar, borrar y mostrar notas está todo en un mismo programa.

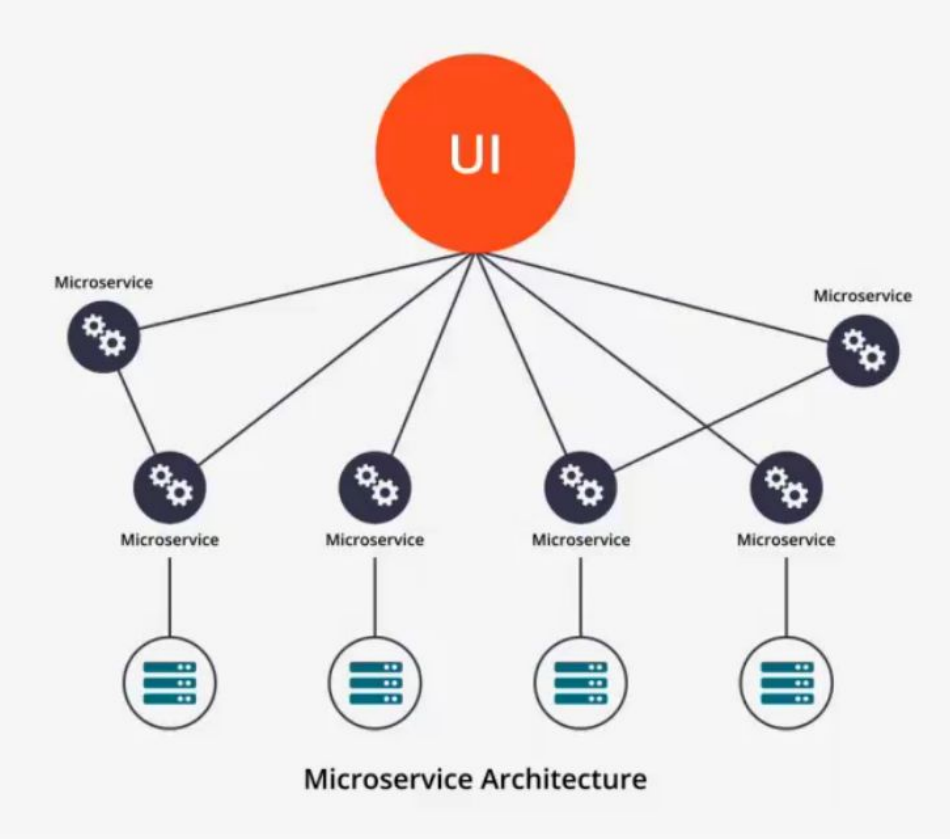


Arquitecturas monolíticas VS distribuidas

Arquitectura Distribuida:

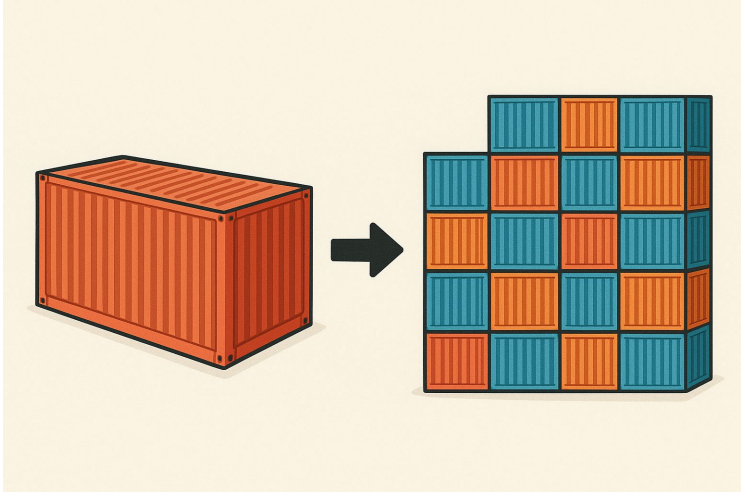
La aplicación se divide en varias partes (servicios) que se comunican entre sí a través de la red. Cada parte puede estar en un servidor distinto.

- Pros: fácil de escalar, más resistente y actualizable por partes. Contras: más compleja de desarrollar y mantener.
- Ejemplo: Una tienda en línea donde catálogo, carrito y pagos funcionan en servidores separados y se comunican entre ellos.



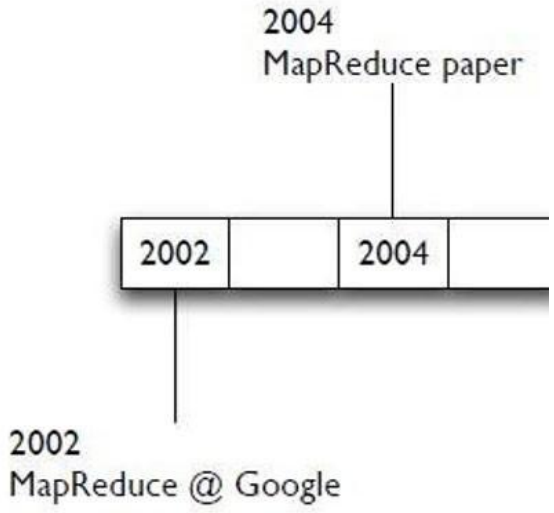
Arquitecturas monolíticas VS distribuidas

- Durante décadas, el objetivo fue crear máquinas más grandes y potentes, pero este enfoque tiene limitaciones.
- A medida que las aplicaciones crecen, características como la facilidad de mantenimiento, la agilidad, la capacidad de prueba y la capacidad de implementación se ven afectadas negativamente. Además, tiene un alto costo y una escalabilidad limitada.
- Estas son las razones por las que una arquitectura monolítica no es un buen enfoque para una aplicación de big data.



Creación de MapReduce

- Google se propuso crear un motor de búsqueda capaz de indexar y buscar todos los datos de Internet a gran velocidad.
- Los sistemas de almacenamiento tradicionales y los métodos de programación convencionales no podían manejar la escala masiva que Google necesitaba.
- Esto llevó a la creación de nuevos enfoques, como el Google File System (GFS), MapReduce (MR) y Bigtable.



Creación de MapReduce

- MapReduce (MR) introdujo un nuevo paradigma de programación paralela. Basado en la programación funcional, permite procesar grandes volúmenes de datos distribuidos en sistemas como GFS y Bigtable.

- En lugar de traer los datos al programa, las aplicaciones MR envían el código de cálculo (funciones de map y reduce) directamente al lugar donde residen los datos. Esto es clave en escenarios de Big Data, donde mover enormes cantidades de información sería ineficiente.

```
words = ["dog", "cat", "dog", "tree", "cat", "dog"]
```

MAP

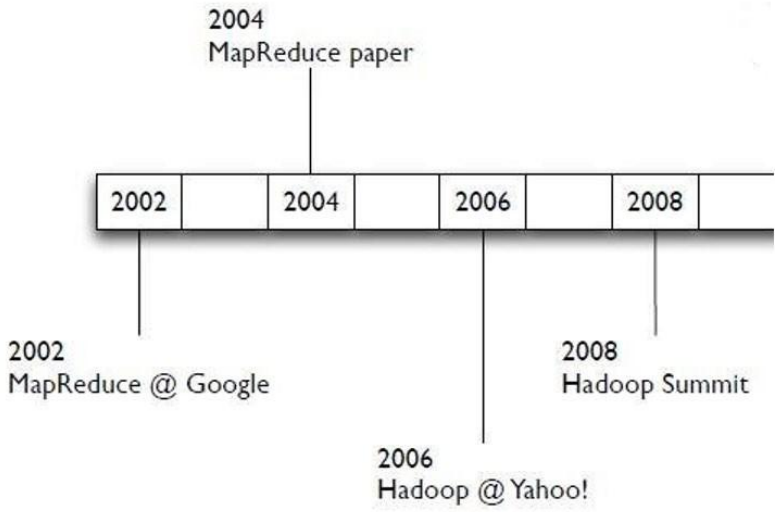
```
[('dog', 1), ('cat', 1), ('dog', 1), ('tree', 1), ('cat', 1), ('dog', 1)]
```

```
[('dog', 3), ('cat', 2), ('tree', 1)]
```

REDUCE

Creación de HDFS

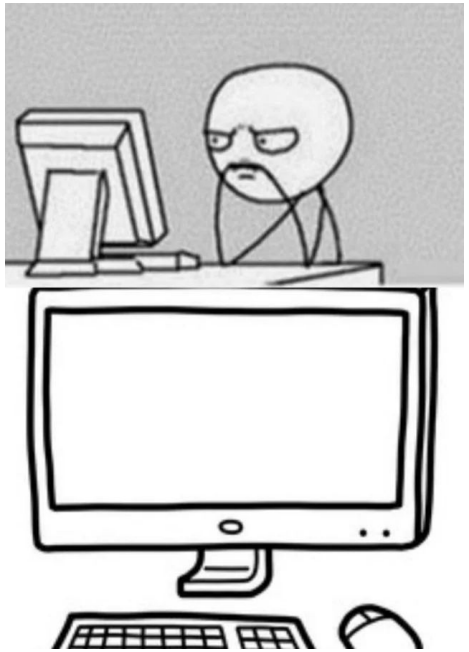
- Inspirado en el Google File System (GFS) para manejar grandes volúmenes de datos.
- Donado a la Apache Software Foundation en abril de 2006.
- Parte del ecosistema Apache Hadoop: incluye HDFS, MapReduce y YARN.
- Inspiró una gran comunidad de contribuyentes de código abierto y dos empresas comerciales basadas en código abierto: Cloudera y Hortonworks.
- Módulos principales: Hadoop y Common.



Uso de MapReduce en HDFS

- Difícil de gestionar y operar; API de MapReduce difícil y engorrosa.
- Resultados intermedios se escriben en disco, causando lentitud en trabajos grandes.
- No soporta de forma nativa otras cargas de trabajo: ML, streaming o consultas SQL.
- Se crearon sistemas a medida (Hive, Storm, Impala, Mahout), aumentando la complejidad operativa.

Y con todo esto… ¿Había alguna forma de hacer que Hadoop y MR fueran más sencillos y rápidos?



Primeros Pasos de Spark

En 2009, investigadores de la Universidad de California en Berkeley que habían trabajado anteriormente en Hadoop MapReduce asumieron este reto con un proyecto al que llamaron Spark.

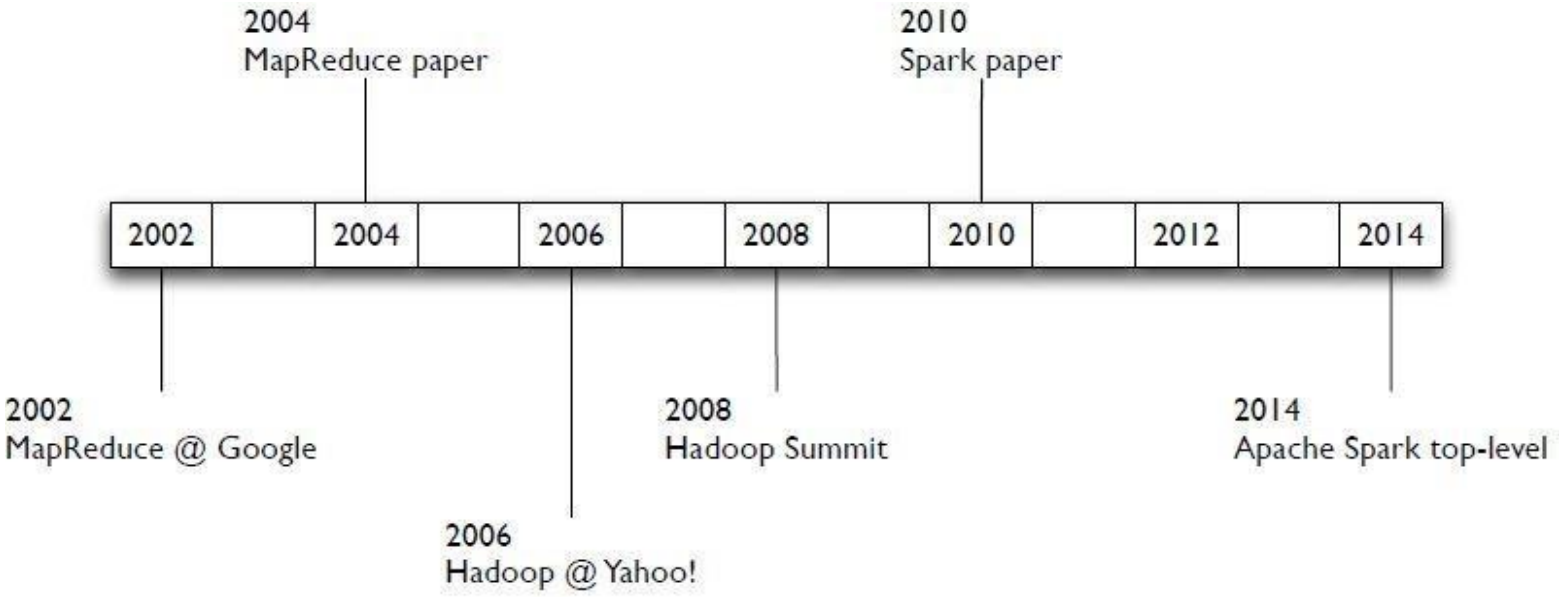
El objetivo era incorporar ideas de Hadoop MapReduce, pero también mejorar el sistema:

1. Hacerlo altamente tolerante a fallos y extremadamente paralelo.
2. Admitir el almacenamiento en memoria para los resultados intermedios de los cálculos de map y reduce.
3. Ofrecer API fáciles y combinables en varios lenguajes como modelo de programación.
4. Y admitir otras cargas de trabajo de forma unificada.

Primeros Pasos de Spark

- En 2013, Spark ya se había generalizado y algunos de sus creadores e investigadores originales donaron el proyecto Spark a la ASF y formaron una empresa llamada Databricks.

- Databricks y la comunidad de desarrolladores de código abierto trabajaron para lanzar Apache Spark 1.0 en mayo de 2014, bajo la dirección de la ASF.



2. Introducción Apache Spark

Qué es Apache Spark

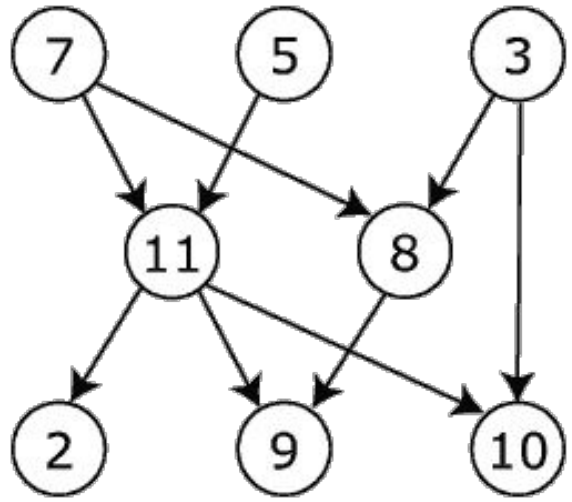
"Apache Spark™ es un motor multilingüe para ejecutar ingeniería de datos, ciencia de datos y aprendizaje automático en máquinas de un solo nodo o clústeres."

- Apache Spark es un motor: es el núcleo o sistema que hace funcionar procesos de datos, igual que un motor hace funcionar un coche.
- Multilingüe: Spark puede ser usado desde varios lenguajes de programación, por ejemplo: Python, Scala, Java o R.
- En máquinas de un solo nodo o clusters: Spark puede funcionar en una sola máquina (un solo nodo), como tu propio ordenador o en un conjunto de máquinas conectadas (un clúster), para trabajar con datos muy grandes de forma distribuida.

Características Clave de Apache Spark

Velocidad:

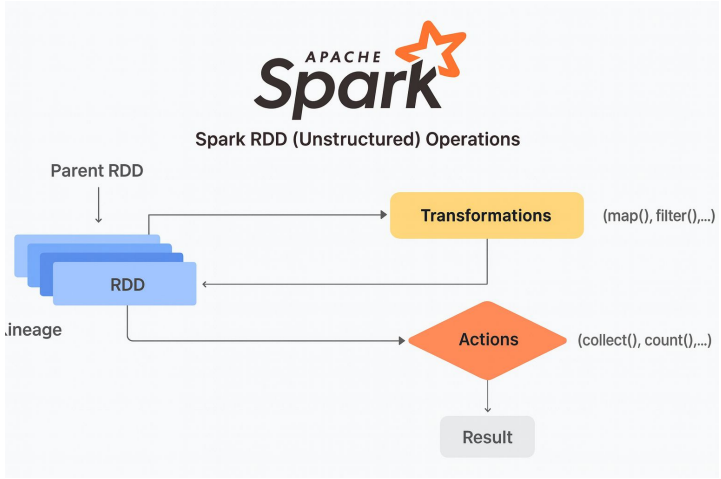
- Aprovecha bien el **hardware moderno**: Utiliza varios núcleos y memoria de forma inteligente, haciendo muchas partes del trabajo al mismo tiempo.
- Divide el trabajo en **tareas paralelas** (DAG: Directed Acyclic Graph): Spark transforma las operaciones en un "mapa" de pasos que se pueden ejecutar en paralelo, acelerando el proceso.
- **Procesa en memoria**: Guarda los resultados intermedios en RAM en lugar de en disco, lo que reduce tiempos de lectura/escritura y acelera muchísimo.
- Optimiza el código con **Tungsten**: Genera código muy eficiente y compacto que el ordenador puede ejecutar más rápido.



Características Clave de Apache Spark

Facilidad de Uso:

- Abstracción fundamental con **RDD**: Spark proporciona una estructura de datos lógica básica llamada RDD (Resilient Distributed Dataset). Todas las demás abstracciones de datos de nivel superior, como DataFrames y Datasets, se construyen sobre esta.
- Operaciones con **transformaciones y acciones**: Spark utiliza un modelo de programación sencillo que permite enfocarse en el contenido del cálculo y mejora la productividad.
- Soporte para **múltiples lenguajes de programación**: Puedes programar en Scala, Java, **Python**, SQL y R, según tus preferencias.



Características Clave de Apache Spark

Modularidad:

- Motor de procesamiento **unificado** para muchos tipos de cargas de trabajo (SQL, ML, streaming, gráficos, etc.).
- No es necesario disponer de motores distintos para cargas de trabajo dispares.
- Se puede escribir una única aplicación Spark que lo haga todo.

Extensibilidad:

- Spark desacopla el almacenamiento y el cálculo → necesidad de un sistema de almacenamiento distribuido.
- **Compatibilidad** con numerosas fuentes: Hadoop, Cassandra, HBase, MongoDB, Hive, RDBMS, etc. Los lectores y escritores DF también se pueden ampliar para leer desde otras fuentes, como Apache Kafka, Kinesis, Azure Storage y Amazon S3.

Casos de Uso de Spark

Tanto si eres ingeniero de datos, científico de datos o ingeniero de aprendizaje automático, Apache Spark te será útil en múltiples escenarios, entre ellos:

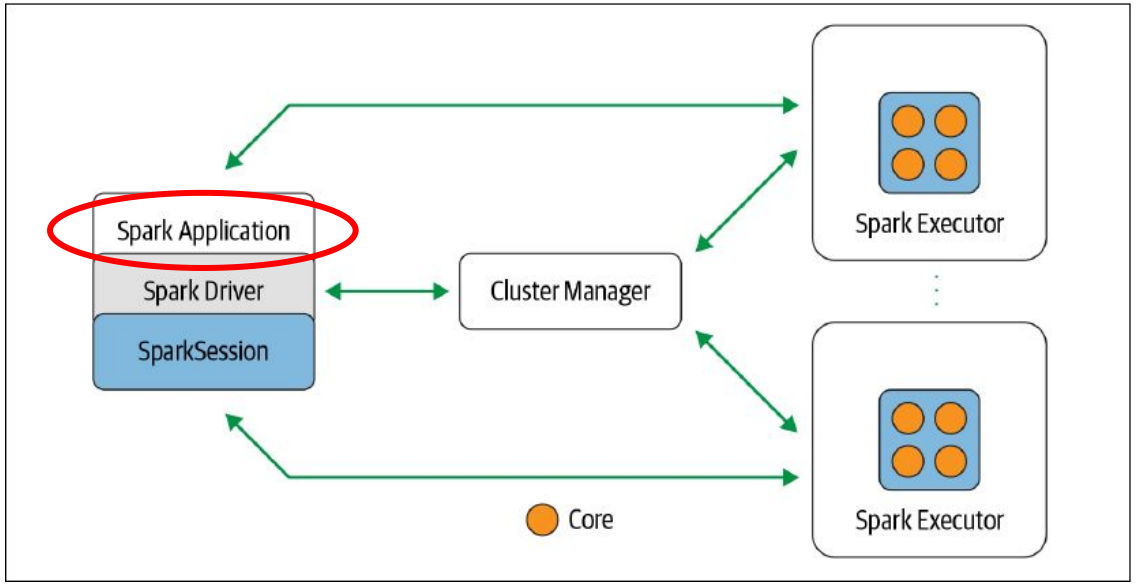
- Procesar en paralelo grandes volúmenes de datos distribuidos en un clúster, aprovechando la computación distribuida para acelerar el rendimiento.
- Realizar consultas ad hoc o interactivas que permiten explorar, analizar y visualizar datos de forma rápida.
- Crear, entrenar y evaluar modelos de aprendizaje automático utilizando la biblioteca integrada MLlib.
- Construir canalizaciones de datos de extremo a extremo, capaces de integrar y transformar datos procedentes de múltiples fuentes.
- Analizar grafos y redes sociales, facilitando el estudio de relaciones y estructuras complejas mediante GraphX.

3. Arquitectura de Apache Spark

Componentes de Apache Spark

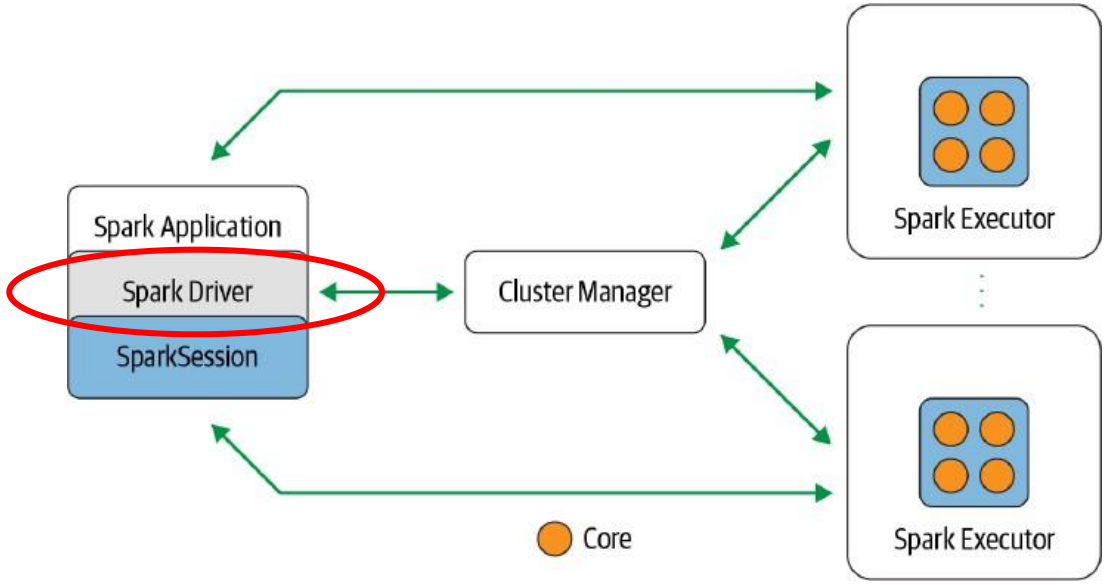
- Como hemos visto, Spark es un motor de procesamiento de datos distribuido, lo que significa que un clúster de máquinas trabaja de manera coordinada para ejecutar las tareas.

- A nivel arquitectónico, una **aplicación Spark** consta de un programa controlador (driver), que se encarga de orquestar y coordinar las operaciones paralelas en todo el clúster.



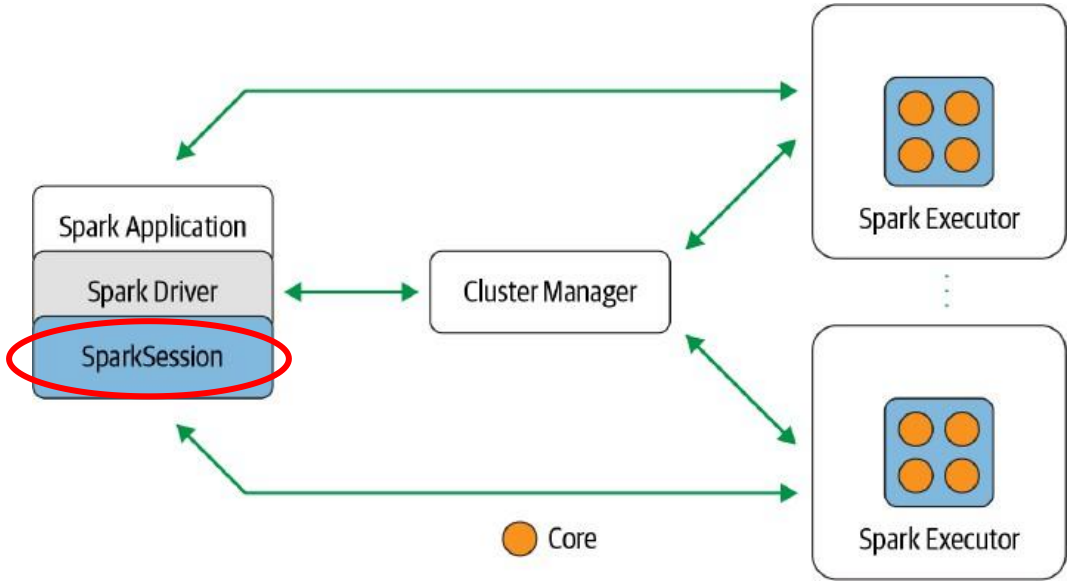
Spark Driver

- Responsable de i**nstanciar una SparkSession**
- Solicita recursos (CPU, memoria, etc.) al gestor del clúster para los ejecutores de Spark (JVMs)
- Transforma todas las **operaciones de Spark en cálculos DAG**, las programa y distribuye su ejecución como tareas entre los ejecutores de Spark



Spark Session

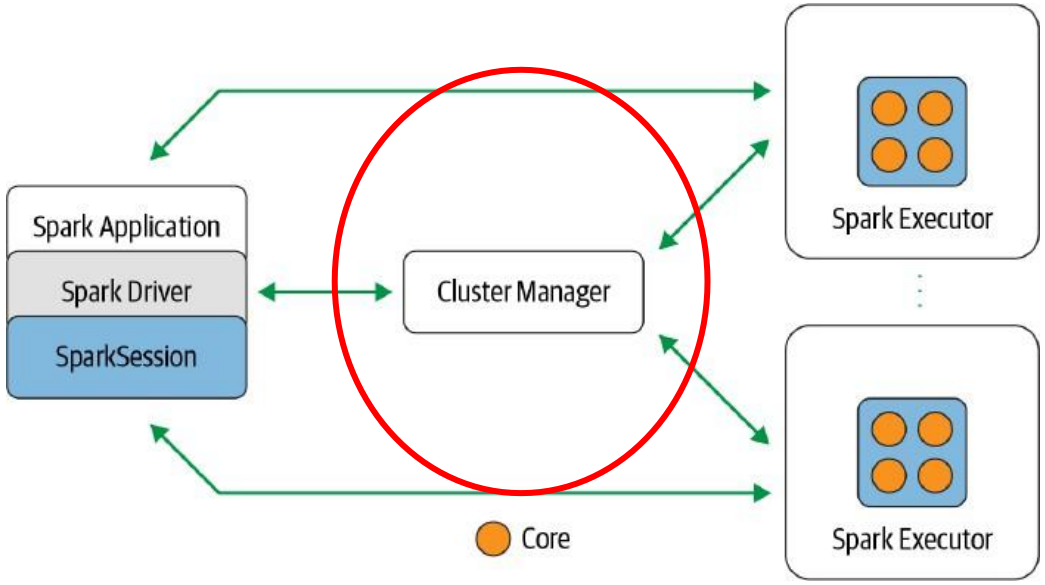
- Punto de **entrada principal** para interactuar con Spark (DF, SQL…)
- Gestiona la configuración de la aplicación y la inicia, al igual que la conexión con el clúster a través del SparkContext.



Cluster Manager

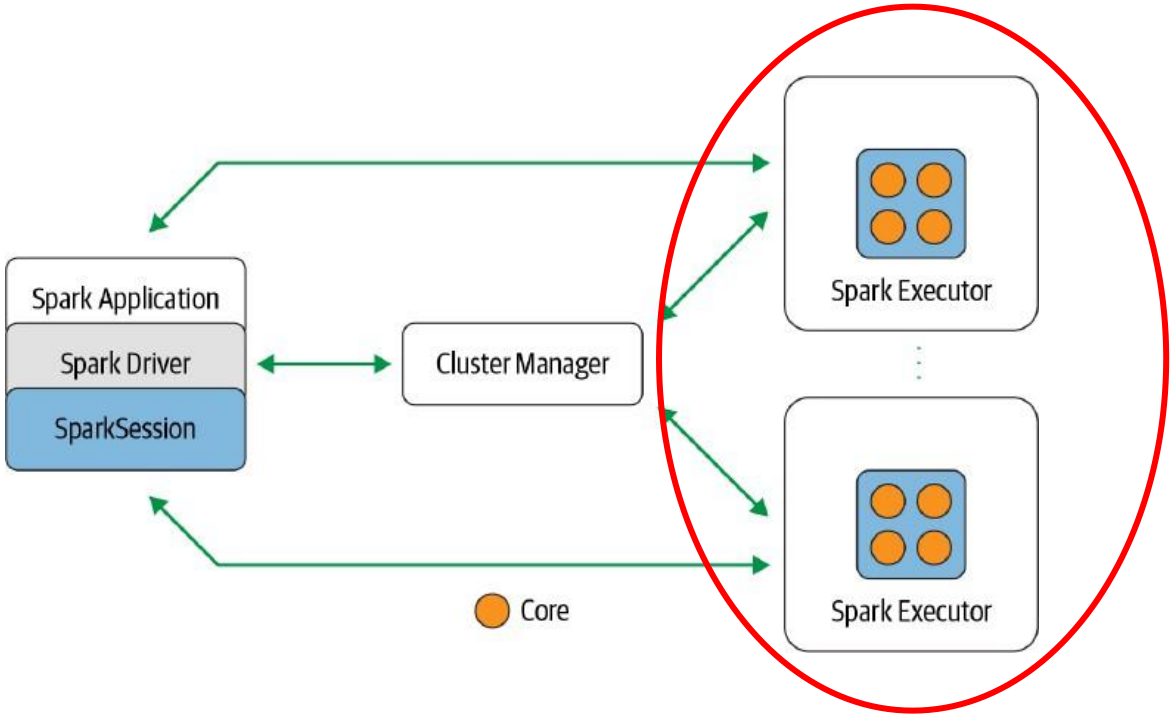
- Administra los **recursos del cluster** (CPU, memoria y nodos) y recibe las solicitudes de las aplicaciones Spark para ejecutar sus jobs.

- Decide cómo distribuir los recursos disponibles, **lanza los ejecutores** en los nodos worker y coordina la ejecución de las tareas.



Spark Executor

- Los ejecutores se ejecutan en los nodos del cluster y son responsables de **ejecutar las tareas** asignadas por el driver, almacenar datos en memoria y devolver los resultados.
- Dependiendo del modo de despliegue, puede haber uno o varios ejecutores por nodo.



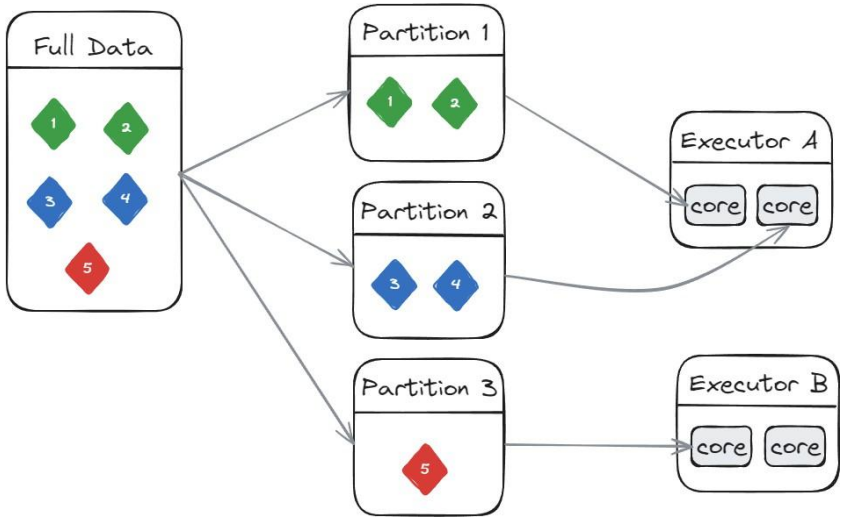
Métodos de Despliegue

Modo	Dónde corre el Driver (el que dirige el trabajo)	Dónde corren los Ejecutores (los que hacen el trabajo)	Quién gestiona los recursos	Cómo entenderlo fácilmente
Local	En tu propio ordenador	En tu propio ordenador (en el mismo proceso)	No hay gestor real	Todo corre "en tu portátil"
Standalone	En cualquier máquina del cluster	En cada nodo del cluster (uno o varios por nodo)	Gestor simple propio de Spark	Un cluster pequeño administrado por Spark
YARN (Client)	En tu ordenador (fuera del cluster)	En contenedores que YARN crea en los nodos	YARN (ResourceManager + ApplicationMaster)	Tú diriges desde fuera y YARN ejecuta dentro
YARN (Cluster)	Dentro del cluster (dentro del ApplicationMaster)	Igual que en YARN Client (contenedores YARN)	YARN	Spark se ejecuta completamente dentro del cluster
Kubernetes	Dentro de un pod de Kubernetes	Cada ejecutor en su propio pod	Kubernetes	Todo corre como contenedores

4. Procesamiento en Paralelo

Datos distribuidos y Particiones

- Los datos reales se almacenan en **particiones**, que son fragmentos del dataset distribuidos en un sistema de almacenamiento como HDFS, S3 u otros sistemas distribuidos.
- Spark intenta aprovechar la **localidad** de los datos, es decir, intenta que cada ejecutor procese las particiones que están almacenadas en el mismo nodo o lo más cerca posible. Esto reduce el uso de la red, evita mover datos innecesariamente y permite un paralelismo más rápido y eficiente.



Jobs, Stages & Tasks

- **Jobs Spark:** El driver toma tu código y lo convierte en uno o varios jobs.
- **Stages Spark:** Cada job se divide en stages según si las operaciones pueden hacerse en paralelo o necesitan esperar datos (puntos donde hay shuffle).
- **Tasks Spark:** Cada stage se ejecuta como muchas tasks. Una task es la unidad mínima de trabajo: se ejecuta en un solo núcleo y procesa una sola partición de datos.