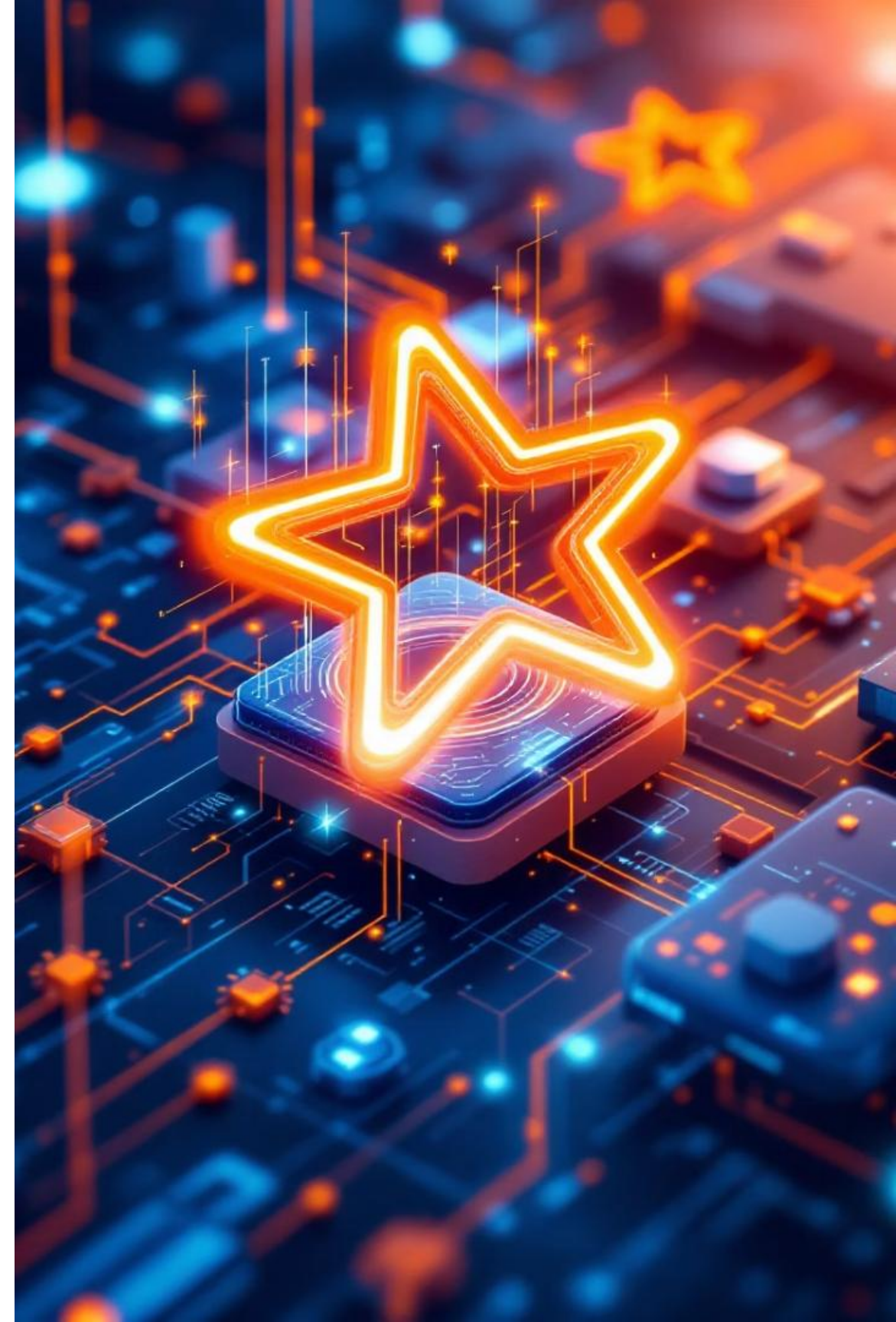


Apache Spark: Procesamiento Distribuido de Datos

Apache Spark es un motor de procesamiento de datos distribuido de código abierto, diseñado para realizar análisis de grandes volúmenes de datos de forma rápida, escalable y versátil. Desarrollado originalmente en el AMPLab de la Universidad de Berkeley, se convirtió en un proyecto de la Apache Software Foundation en 2014.

Esta presentación explorará las características fundamentales de Spark, su arquitectura, implementaciones y ventajas frente a otros sistemas de procesamiento de datos. Veremos cómo Spark permite ejecutar trabajos en memoria, haciéndolo considerablemente más rápido que sistemas como Hadoop MapReduce.

 **por Kibernetum Capacitación S.A.**



Preguntas de Activación de Contenidos

1. ¿Por qué Apache Spark es más rápido que Hadoop MapReduce para procesar grandes volúmenes de datos?
2. ¿Qué ventajas ofrece Apache Spark al proporcionar una plataforma unificada para múltiples tipos de procesamiento?
3. ¿Qué problemas específicos resuelve Apache Spark en comparación con tecnologías tradicionales como Hadoop MapReduce?



¿Qué es Apache Spark y por qué se necesita?



Procesamiento en memoria

Spark ejecuta trabajos de procesamiento in-memory, haciéndolo hasta 100 veces más rápido que sistemas como Hadoop MapReduce, que dependen del acceso constante al disco.



Plataforma unificada

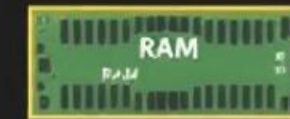
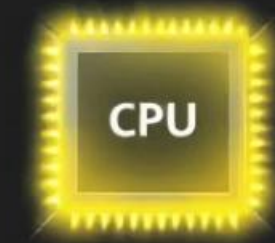
Permite realizar distintos tipos de procesamiento: batch, streaming en tiempo real, análisis con SQL, machine learning y procesamiento de grafos.



APIs en varios lenguajes

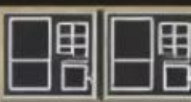
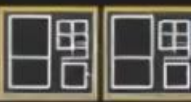
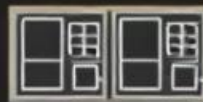
Ofrece APIs en Scala, Python, Java y R, con una estructura coherente para desarrollar pipelines complejos de datos.

In-memory computing vs. disk-based computing



Disk-based

directly directly directed with all spinning data drives



CPU lower spinning disk drives



Apache Spark

¿Por qué se necesita Apache Spark?

A medida que el volumen, velocidad y variedad de los datos (las 3V de Big Data) han crecido exponencialmente, las tecnologías tradicionales de análisis y almacenamiento empezaron a mostrar limitaciones importantes. En ese contexto, Spark surge como una respuesta a múltiples desafíos del procesamiento de datos a gran escala.

Limitaciones de tecnologías anteriores como Hadoop

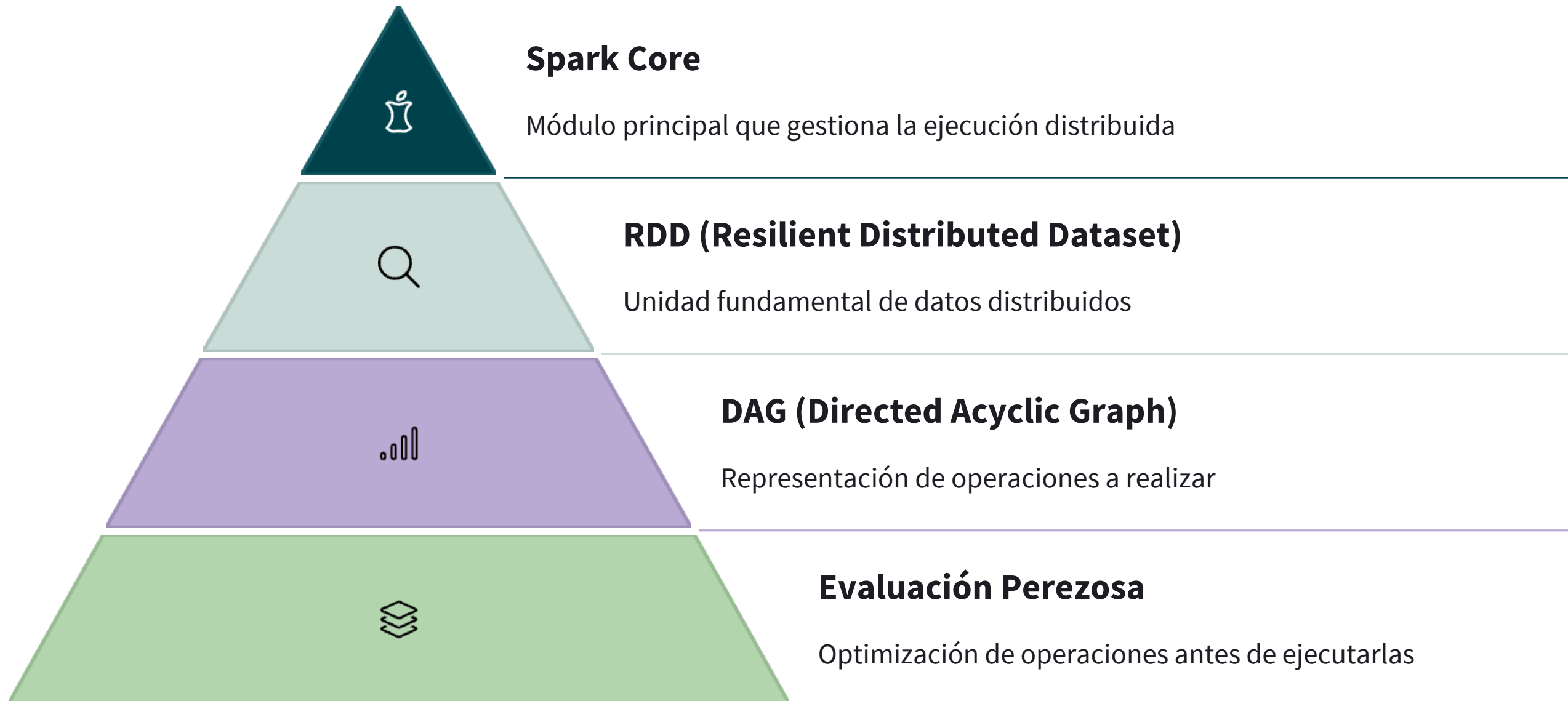
MapReduce:

- Procesamiento lento: Hadoop MapReduce guarda resultados intermedios en disco entre etapas, lo que genera alta latencia.
- Falta de integración: Cada tipo de procesamiento (batch, streaming, ML, SQL) debía hacerse con herramientas distintas, lo que dificultaba el mantenimiento y la interoperabilidad.
- Ineficiencia en algoritmos iterativos: MapReduce no es adecuado para tareas que requieren múltiples pasos iterativos, como el entrenamiento de modelos de aprendizaje automático.

Ventajas de Spark:

- Velocidad: Gracias al procesamiento en memoria, Spark puede ser hasta 100 veces más rápido que MapReduce para ciertas tareas.
- Versatilidad: Permite trabajar con distintos tipos de datos y procesos desde una misma plataforma.
- Simplicidad: Ofrece APIs en varios lenguajes (Scala, Python, Java, R) y una estructura coherente para desarrollar pipelines complejos de datos.
- Ecosistema poderoso: Spark incluye bibliotecas integradas como Spark SQL para consultas estructuradas, Spark Streaming para datos en tiempo real, MLlib para machine learning, GraphX para grafos

Conceptos Básicos de Spark



Problemas que Resuelve Spark



Procesamiento lento por acceso a disco

Spark supera las limitaciones de Hadoop MapReduce



Ineficiencia en algoritmos iterativos

Optimiza tareas que requieren múltiples pasos



Necesidad de procesamiento en tiempo real

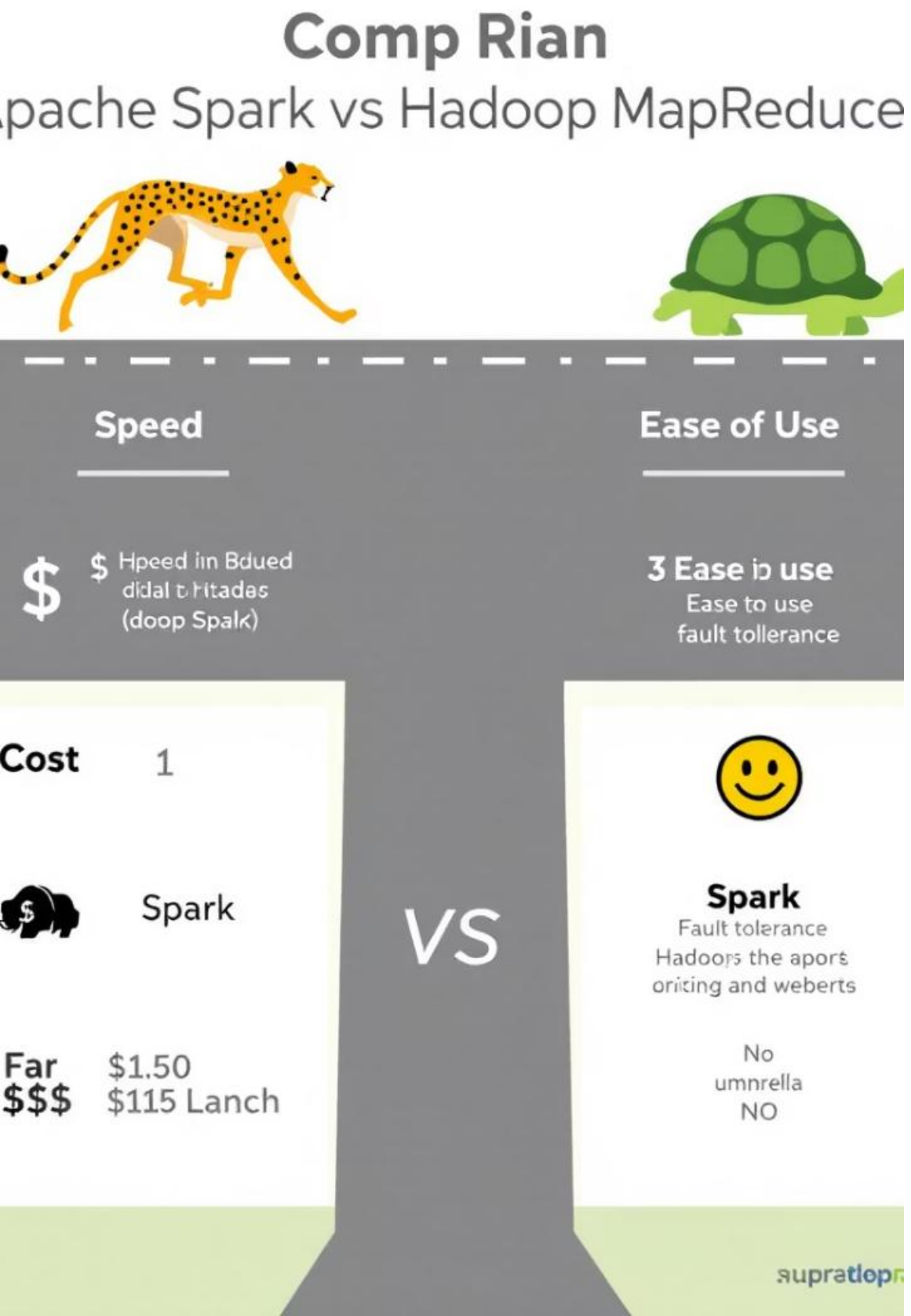
Spark Streaming permite análisis de datos en tiempo real



Ecosistema fragmentado

Unifica diferentes tipos de procesamiento bajo una misma API

Apache Spark nació como solución directa a varias limitaciones críticas encontradas en sistemas tradicionales de procesamiento de datos, especialmente en entornos que demandan velocidad, flexibilidad y análisis en tiempo real.



Comparativa con Otros Sistemas de Procesamiento

Característica	Hadoop MapReduce	Apache Spark
Modelo de ejecución	Procesamiento por lotes (batch)	Procesamiento en memoria y por lotes
Velocidad	Lento, acceso constante a disco	Rápido, mantiene datos en memoria
Facilidad de uso	API de bajo nivel, verbosa	API de alto nivel, expresiva y moderna
Procesamiento en tiempo real	No compatible	Compatible (con Spark Streaming)
Machine Learning	Limitado, con Mahout	Integrado con MLlib

¿Cuándo Conviene Usar Spark?

Casos ideales para Spark

- Procesamiento rápido de grandes volúmenes de datos
- Necesidad de plataforma unificada para múltiples tipos de análisis
- Análisis en tiempo real o casi real
- Disponibilidad de clúster distribuido o recursos en la nube

Casos donde NO conviene usar Spark

- Procesos muy pequeños o simples (pocos MB de datos)
- Entornos con recursos limitados de memoria RAM
- Necesidad de latencia de milisegundos reales (considerar Apache Flink)

API de Spark

La API de Apache Spark es el conjunto de herramientas y funciones que permite a los desarrolladores interactuar con Spark para procesar datos de manera distribuida. Spark tiene varias APIs en diferentes lenguajes de programación, como Scala, Java, Python (PySpark) y R. Algunos de los principales módulos son:

Spark SQL

Permite la ejecución de consultas SQL sobre datos estructurados

GraphX

Procesamiento y análisis de grafos a gran escala



Spark Streaming

Facilita el procesamiento de datos en tiempo real

MLlib

Biblioteca para ejecutar algoritmos de machine learning distribuidos

Ejemplos Prácticos

Requisitos: Ingresar a: <https://colab.research.google.com>

▼ Introducción a PySpark en Jupyter Notebook

Este notebook contiene un ejemplo práctico para conocer la API de PySpark.

```
[1] 1 !pip install pyspark
      2
      3 # ⚙️ Iniciar una sesión de Spark
      4 from pyspark.sql import SparkSession
      5
      6 spark = SparkSession.builder \
      7     .appName("Ejemplo PySpark") \
      8     .getOrCreate()
```

Ejemplos Prácticos

```
1 # 🇵🇹 Crear un DataFrame desde una lista de Python
2 datos = [
3     ("Juan", 28),
4     ("María", 35),
5     ("Pedro", 40),
6     ("Lucía", 22)
7 ]
8
9 df = spark.createDataFrame(datos, ["Nombre", "Edad"])
10 df.show()
```

```
⇒ +-----+-----+
  |Nombre|Edad|
  +-----+-----+
  |  Juan|  28|
  |María|  35|
  |Pedro|  40|
  |Lucía|  22|
  +-----+-----+
```

Ejemplos Prácticos

```
[3] 1 # 🖋 Filtrar y transformar datos
    2 df_mayores = df.filter(df["Edad"] > 30)
    3 df_mayores.show()
    4
    5 df_con_meses = df.withColumn("Edad_en_meses", df["Edad"] * 12)
    6 df_con_meses.show()
```



```
+-----+-----+
|Nombre|Edad|
+-----+-----+
| María| 35|
| Pedro| 40|
+-----+-----+

+-----+-----+-----+
|Nombre|Edad|Edad_en_meses|
+-----+-----+-----+
| Juan| 28|          336|
| María| 35|          420|
| Pedro| 40|          480|
| Lucía| 22|          264|
+-----+-----+-----+
```


Ejemplos Prácticos

```
[4] 1 # 📄 Usar SQL con Spark
    2 df.createOrReplaceTempView("personas")
    3 resultado_sql = spark.sql("SELECT Nombre, Edad FROM personas WHERE Edad BETWEEN 25 AND 35")
    4 resultado_sql.show()
```

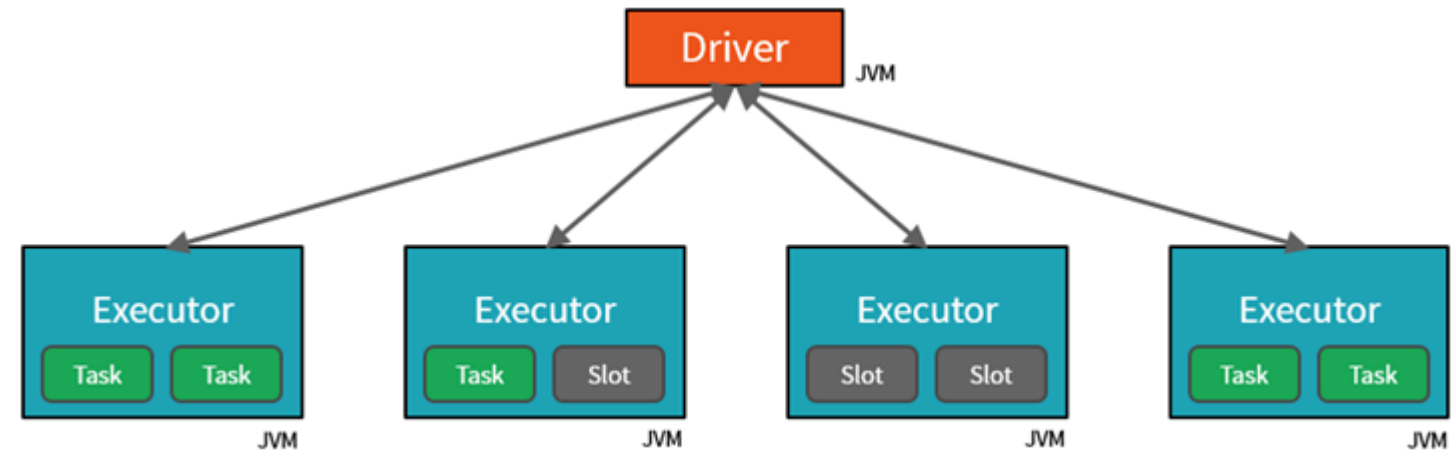
```
⇒ +-----+-----+
   |Nombre|Edad|
   +-----+-----+
   | Juan | 28 |
   | María| 35 |
   +-----+-----+
```

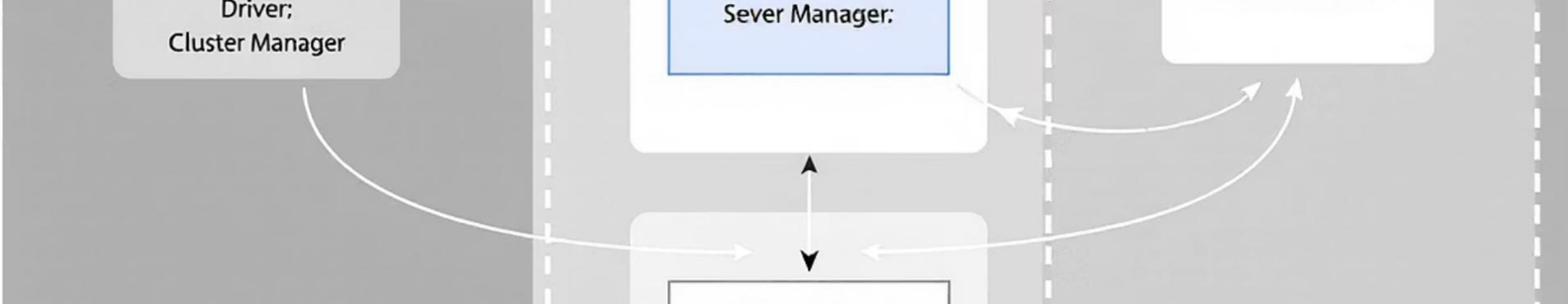
```
▶ 1 # 🛠 Cerrar la sesión de Spark
   2 spark.stop()
```

https://colab.research.google.com/drive/1AE84zdNrmdv9KxO4_yWSNxkQHROqGTmL?usp=sharing

Arquitectura General de Spark

La arquitectura de Apache Spark es un modelo distribuido de procesamiento de datos que permite ejecutar tareas de análisis en paralelo en un clúster de computadoras. Se basa en un modelo donde un Driver Program controla y coordina el flujo de ejecución, mientras que múltiples Workers (nodos) procesan las tareas asignadas.





Componentes Principales de Spark

Driver Program

- Crea el DAG (Directed Acyclic Graph)
- Envía tareas a los Executors
- Monitorea y gestiona tareas

Cluster Manager

- Asigna recursos
- Gestiona nodos
- Comunica información al Driver

Executors

- Ejecutan tareas distribuidas
- Almacenan datos en memoria
- Reportan progreso al Driver

Ejemplo en PySpark de Driver Program



```
from pyspark.sql import SparkSession

# Iniciar la sesión de Spark
spark = SparkSession.builder.appName("Ejemplo Driver").getOrCreate()

# El Driver crea un RDD y define operaciones
datos = [1, 2, 3, 4, 5]
rdd = spark.sparkContext.parallelize(datos)
rdd_squared = rdd.map(lambda x: x ** 2)
print(rdd_squared.collect()) # Driver coordina y recopila resultados

# Detener la sesión de Spark
spark.stop()
```


Ejemplo en PySpark de Executors



```
# El Driver asigna tareas a los Executors
datos = [1, 2, 3, 4, 5]
rdd = spark.sparkContext.parallelize(datos)

# Los Executors ejecutan la transformación map()
rdd_cuadrado = rdd.map(lambda x: x ** 2)

# Los Executors devuelven los resultados al Driver
print(rdd_cuadrado.collect())
```

Modo de Funcionamiento en Cluster

4

Modos de Ejecución

Local, Standalone, YARN y Kubernetes

100x

Más Rápido

Que Hadoop MapReduce en ciertas tareas

3

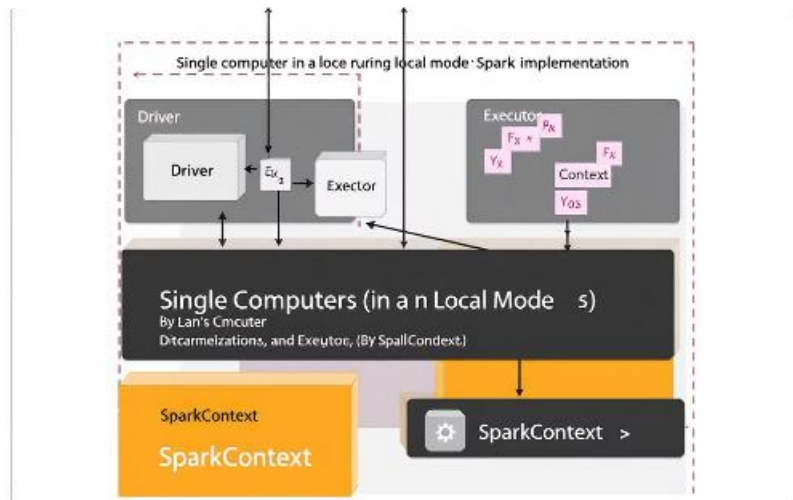
Componentes Clave

Driver, Cluster Manager y Executors

El modo de funcionamiento en cluster permite que una aplicación Spark ejecute tareas en nodos distribuidos, donde los datos y el procesamiento se dividen entre múltiples máquinas. Este enfoque es clave para procesar grandes volúmenes de datos de manera eficiente y escalable.

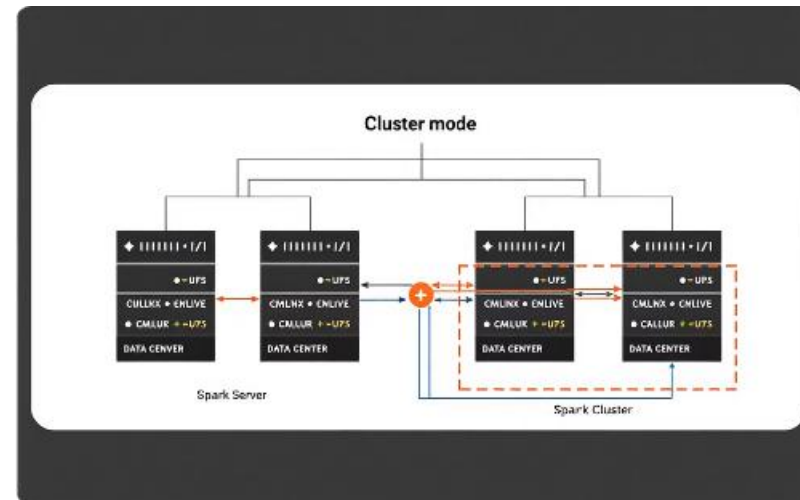


Implementaciones de Spark



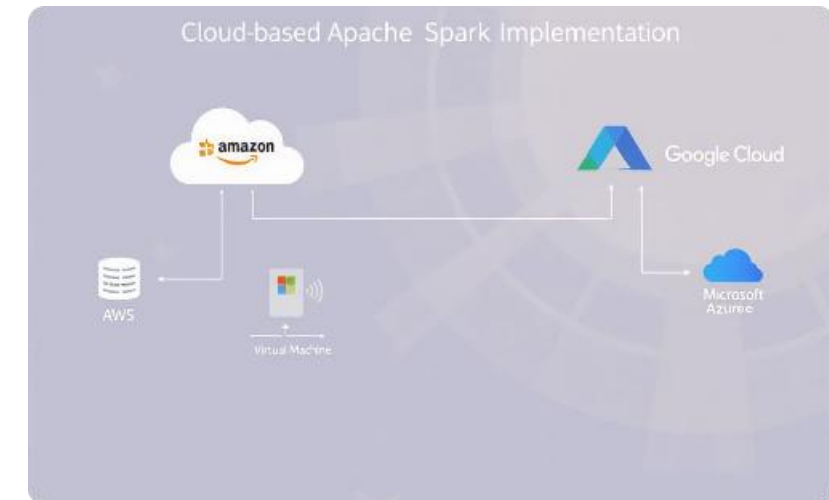
Modo Local

Spark se ejecuta en una sola máquina, ideal para desarrollo y pruebas. Todos los componentes (Driver, Executors) operan en el mismo sistema, facilitando la depuración y el aprendizaje inicial.



Modo Cluster

Spark se ejecuta en un clúster distribuido, con el Driver y los Executors en diferentes nodos. Permite procesar grandes volúmenes de datos aprovechando recursos de múltiples máquinas.



Modo Cloud

Permite ejecutar Spark en plataformas de nube como Amazon EMR, Google Cloud Dataproc, Azure HDInsight o Databricks, aprovechando la elasticidad y escalabilidad de la infraestructura cloud.

Spark en la Nube

Apache Spark en la nube consiste en implementar y ejecutar aplicaciones Spark en plataformas de **cloud computing**, aprovechando la elasticidad, escalabilidad y alta disponibilidad que ofrecen los proveedores de servicios en la nube.

Este enfoque permite a las organizaciones procesar grandes volúmenes de datos sin preocuparse por la infraestructura física, ya que los servicios gestionados en la nube se encargan de:

- Aprovisionar clústeres Spark.
- Gestionar los nodos de trabajo (workers).
- Asignar recursos de forma dinámica según la carga de trabajo.

Opciones en la Nube:

- Amazon EMR (Elastic MapReduce)**: Ejecutar Spark en AWS.
- Google Cloud Dataproc**: Ofrece Spark como un servicio gestionado en Google Cloud.
- Azure HDInsight**: Servicio de clúster en la nube de Azure.
- Databricks**: Plataforma basada en Apache Spark.

Ventajas principales:

- Escalabilidad automática**: Ajuste dinámico de recursos según el volumen de datos.
- Alta disponibilidad**: Clústeres distribuidos con replicación automática.
- Costos optimizados**: Paga solo por los recursos utilizados.
- Integración sencilla**: Con almacenamiento distribuido como S3, GCS o Azure Blob

PySpark



PySpark es la API de Apache Spark para Python que permite ejecutar tareas de procesamiento de datos distribuidos en clústeres Spark desde scripts escritos en Python. PySpark ofrece una interfaz de alto nivel para interactuar con el motor de Apache Spark, lo que permite a los desarrolladores trabajar con RDDs (Resilient Distributed Datasets), DataFrames, SQL, Machine Learning y Streaming de forma sencilla.

¿Por qué usar PySpark?

- **Procesamiento distribuido:** PySpark permite procesar grandes volúmenes de datos en paralelo en múltiples nodos de un clúster.
- **APIs de alto nivel:** Ofrece una sintaxis sencilla para transformar, manipular y analizar datos de manera eficiente.
- **Integración con Machine Learning:** PySpark integra MLlib, lo que facilita la creación de modelos de Machine Learning a gran escala.
- **Compatibilidad con SQL:** Permite ejecutar consultas SQL sobre grandes volúmenes de datos almacenados en formato distribuido.
- **Streaming en tiempo real:** Procesa datos en tiempo real mediante Spark Streaming o Structured Streaming.

Actividad Práctica Guiada

Instalación de Anaconda <https://www.anaconda.com/download/>



[Products](#)

[Solutions](#)

[Resources](#)

[Partners](#)

[Company](#)

[Free Download](#)

[Sign Up](#)

[Sign In](#)

Distribution

Free Download*

Register to get everything you need to get started on your workstation including Cloud Notebooks, Navigator, AI Assistant, Learning and more.

- ✓ Easily search and install thousands of data science, machine learning, and AI packages
- ✓ Manage packages and environments from a desktop application or work from the command line
- ✓ Deploy across hardware and software platforms
- ✓ Distribution installation on Windows, MacOS, or Linux

Provide email to download Distribution

Email Address:

☐

Agree to receive communication from Anaconda regarding relevant content, products, and services. I understand that I can revoke this consent [here](#) at any time.

By continuing, I agree to Anaconda's [Privacy Policy](#) and [Terms of Service](#).

[Submit](#) >

[Skip registration](#)

Actividad Práctica Guiada

Si instalas Anaconda (versión completa)

- **Anaconda** incluye una gran cantidad de herramientas, paquetes y entornos preconfigurados para ciencia de datos, aprendizaje automático e ingeniería de software.

- **Principales productos que se instalan con Anaconda:**

- **Conda** → Sistema de gestión de paquetes y entornos virtuales.

- **Python** → Interprete de Python (generalmente la última versión estable).

- **Jupyter Notebook y JupyterLab** → Entorno interactivo para ejecutar código Python.

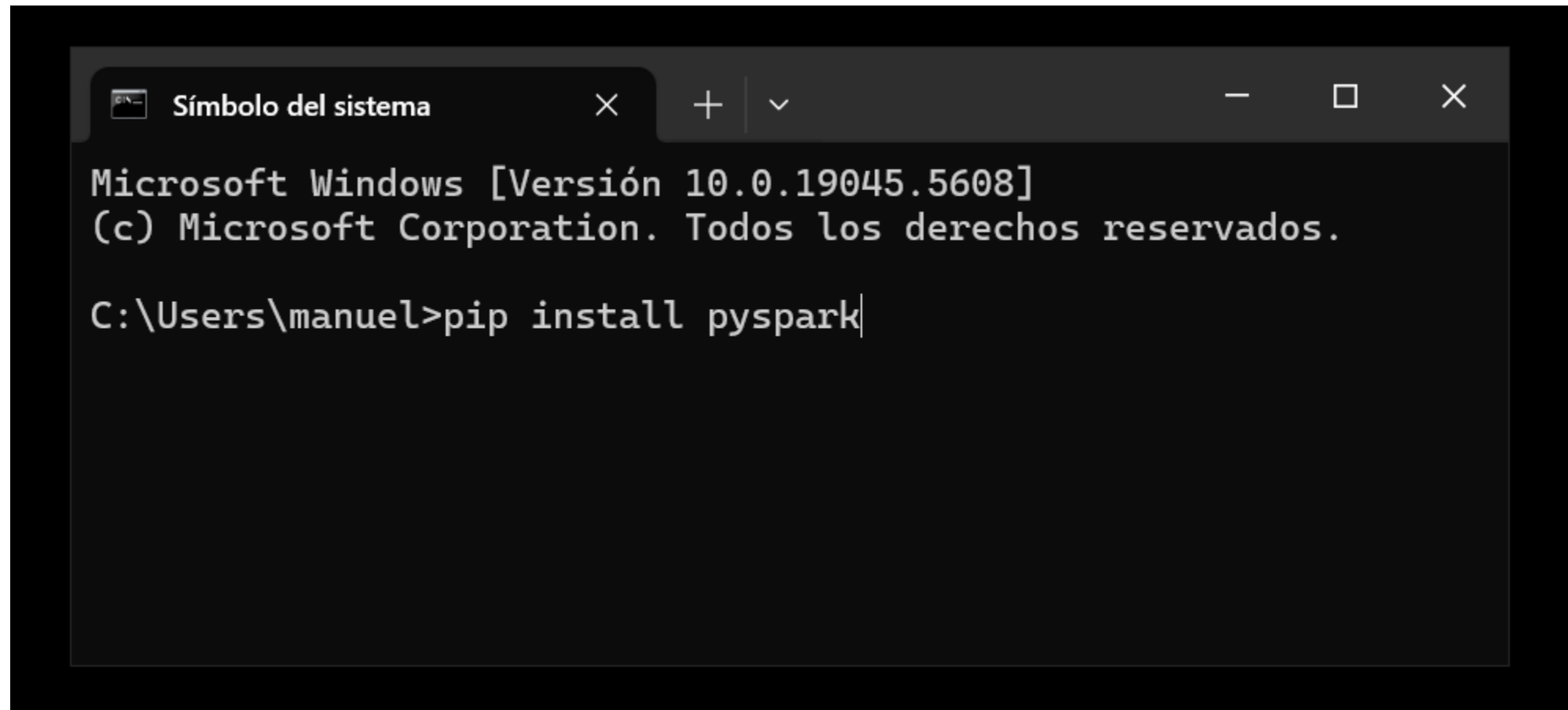
- **Spyder** → IDE para ciencia de datos en Python.

- **R** (Opcional) → Lenguaje de programación para estadística y ciencia de datos.

- **Paquetes Científicos y de IA/ML**

Actividad Práctica Guiada

Instalación de PySpark



A screenshot of a Windows Command Prompt window. The title bar at the top reads "Símbolo del sistema" and includes standard window controls (minimize, maximize, close). The main area of the window displays the following text:

```
Microsoft Windows [Versión 10.0.19045.5608]  
(c) Microsoft Corporation. Todos los derechos reservados.  
  
C:\Users\manuel>pip install pyspark|
```


Enlaces de Interés

PySpark on Databricks

<https://docs.databricks.com/aws/en/pyspark/>

Apache Spark en Amazon

<https://aws.amazon.com/es/emr/features/spark/>

Desafío: Características y Funciones de Apache Spark

Objetivo:

- Reconocer las características principales de Apache Spark.
- Comprender cómo cada función contribuye al procesamiento eficiente de datos distribuidos.
- Sintetizar información clave en una tabla para su análisis comparativo.

Instrucciones:

- Completa la siguiente tabla identificando las **funciones o características principales de Apache Spark**.
- En la primera columna, se encuentra el nombre de la característica.
- En la segunda columna, describe en qué consiste y cómo contribuye al procesamiento de datos distribuidos.

Función/Característica	Descripción
Procesamiento en Memoria	
Modelo de DAG (Directed Acyclic Graph)	
Compatibilidad con Lenguajes	
RDDs (Resilient Distributed Datasets)	
DataFrames y Datasets	
Spark SQL	
Spark Streaming	
MLlib (Machine Learning Library)	
GraphX	
Alta Tolerancia a Fallos	
Escalabilidad Horizontal	
Modo Cluster y Cloud	

Reflexión Final

1. ¿Cuál es el papel del Driver Program en la arquitectura de Apache Spark y por qué es clave para el procesamiento distribuido?
2. ¿En qué situaciones es más adecuado implementar Apache Spark en la nube y qué ventajas ofrece este enfoque?
3. ¿Qué ventajas ofrece PySpark en comparación con bibliotecas tradicionales como Pandas para el análisis de grandes volúmenes de datos?

