IT Academy by KIBERNUM

Sesión: Apache Spark

por Kibernum Capacitación S.A.

APACHE SPARK: QUÉ ES SPARK Y POR QUÉ SE NECESITA

Spark permite ejecutar trabajos de procesamiento en memoria (in-memory), lo que lo hace considerablemente más rápido que sistemas como Hadoop MapReduce, que dependen del acceso constante al disco.

Spark se destaca por ofrecer una plataforma unificada que permite realizar distintos tipos de procesamiento de datos:

- Procesamiento batch (por lotes)
- Procesamiento en tiempo real (streaming)
- Análisis con SQL
- Machine Learning
- Procesamiento de grafos

¿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

Spark Core es el módulo principal del sistema. Maneja:

- Tareas de ejecución distribuida
- Gestión de memoria
- Control de errores
- Planificación de trabajos
- Todos los otros módulos de Spark (SQL, Streaming, MLlib, etc.) se construyen sobre Spark Core.

Su función principal es distribuir las tareas de un programa entre los distintos nodos de un clúster, y coordinar su ejecución paralela.

RDD (Resilient Distributed Dataset)

- Un RDD es la unidad fundamental de datos en Spark. Representa una colección inmutable de objetos, que puede ser distribuida entre varios nodos.
- Características clave de los RDD:
- Tolerancia a fallos: Si un nodo falla, Spark puede reconstruir el RDD usando su historial (lineage).

Evaluación perezosa (lazy evaluation)

- Las operaciones no se ejecutan hasta que no se requiere un resultado final.
- Inmutabilidad: Una vez creado, un RDD no se modifica; las transformaciones generan nuevos RDDs.

```
#Ejemplo simple (en PySpark):

rdd = sc.parallelize([1, 2, 3, 4])
rdd_squared = rdd.map(lambda x: x * x)
print(rdd_squared.collect())  # Resultado: [1, 4, 9, 16]

```

DAG (Directed Acyclic Graph)

Spark construye internamente un DAG para representar las operaciones a realizar sobre los datos. Cada nodo del DAG es una transformación (como .map, .filter) y las flechas representan la dependencia entre ellas.

Esto permite:

- Optimizar el plan de ejecución antes de correr el trabajo.
- Dividir el trabajo en stages (etapas) eficientes.
- Recuperar datos en caso de fallos.
- Lazy Evaluation (Evaluación Perezosa)
- En Spark, las transformaciones como .map() o .filter() no se ejecutan inmediatamente. Solo cuando se llama a una acción como .collect() o .count(), Spark evalúa todas las transformaciones en conjunto, optimizando el camino más eficiente.

PROBLEMAS QUE RESUELVE SPARK

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

Procesamiento lento por acceso a disco (Hadoop MapReduce)

Ineficiencia en algoritmos iterativos

Spark Streaming y más recientemente Structured Streaming, Spark permite el procesamiento en tiempo

Ecosistema fragmentado

Todo bajo una misma API y entorno de ejecución.

COMPARATIVA CON OTROS SISTEMAS DE PROCESAMIENTO

A medida que las necesidades de análisis de datos masivos han crecido, han surgido múltiples herramientas para abordar los desafíos del Big Data. Entre ellas, destacan Apache Hadoop MapReduce, Apache Flink y Apache Spark, cada una con sus propias fortalezas y limitaciones.

Esta sección presenta una comparación estructurada entre Apache Spark y otros sistemas populares, especialmente en términos de rendimiento, capacidades, y casos de uso.

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
Soporte para SQL	Hive (proyecto separado)	Spark SQL
Reusabilidad de datos	No	Sí, mediante RDDs/DataFrames
Manejo de errores	Alta tolerancia a fallos	Alta tolerancia a fallos
Ecosistema	Herramientas separadas	Plataforma unificada

CUÁNDO CONVIENE USAR SPARK

Casos ideales para usar Spark

- Cuando se necesita procesamiento rápido de grandes volúmenes de datos
- Spark fue diseñado para procesar datos a gran escala de forma eficiente y rápida, especialmente cuando se utilizan estructuras en memoria (RAM) como RDDs y DataFrames.
- Cuando se requiere una plataforma unificada para múltiples tipos de procesamiento
- Spark permite combinar diferentes tipos de análisis de datos dentro de una sola aplicación.
- Cuando se necesita análisis en tiempo real o casi real
- Con Structured Streaming, Spark puede procesar flujos de datos con baja latencia y responder en segundos o menos.
- Cuando se cuenta con un clúster distribuido o recursos en la nube
- Spark está optimizado para ejecutarse en clústeres distribuidos y puede integrarse fácilmente con servicios en la nube.

Cuándo NO conviene usar Spark

A pesar de sus ventajas, Spark puede no ser ideal en ciertos contextos:

- Procesos muy pequeños o simples: si solo necesitas procesar unos pocos MB de datos, Spark puede ser un "martillo para un clavo pequeño", con sobrecarga innecesaria.
- Cuando el entorno tiene recursos limitados de memoria RAM: Spark usa intensivamente la memoria, y puede tener problemas de rendimiento o errores si los nodos no están bien configurados.
- Si se necesita baja latencia de milisegundos reales: para casos de streaming ultra-rápido, Apache Flink puede ser más adecuado.

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.

API de Spark SQL

Permite la ejecución de consultas sql.

API de Spark Streaming

Permite el procesamiento en tiempo real.

API de MLlib

Permite ejecutar algoritmos de machine learning.

A continuación, se presentan algunos ejercicios de 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]
!pip install pyspark

# ✪ Iniciar una sesión de Spark
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Ejemplo PySpark") \
    .getOrCreate()
```

```
[2]
# 📊 Crear un DataFrame desde una lista de Python
datos = [
    ("Juan", 28),
    ("María", 35),
    ("Pedro", 40),
    ("Lucía", 22)
]

df = spark.createDataFrame(datos, ["Nombre", "Edad"])
df.show()
```

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

```
[3]
# 🧪 Filtrar y transformar datos
df_mayores = df.filter(df["Edad"] > 30)
df_mayores.show()

df_con_meses = df.withColumn("Edad_en_meses", df["Edad"] * 12)
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|
+------+----+-------------+
```

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

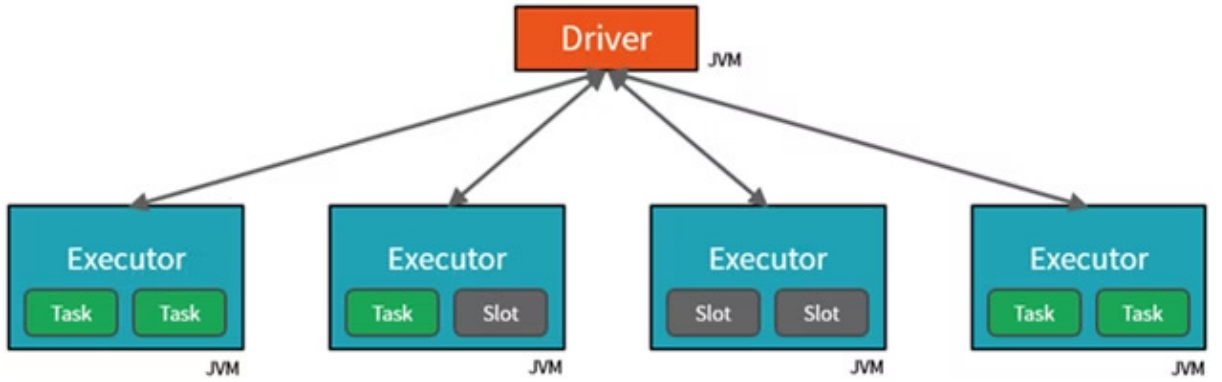
```
+------+----+
|Nombre|Edad|
+------+----+
|  Juan|  28|
| María|  35|
+------+----+
```

```
# 🧹 Cerrar la sesión de Spark
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.



Esta arquitectura permite que Spark distribuya eficientemente el procesamiento de datos a través de múltiples nodos, aprovechando al máximo los recursos disponibles y proporcionando tolerancia a fallos.

PRINCIPALES COMPONENTES DE SPARK (DRIVER, EXECUTOR, CLUSTER MANAGER)

Driver Program

¿Qué hace el Driver Program?

- Crea el DAG (Directed Acyclic Graph): El Driver convierte las transformaciones y acciones del programa Spark en un DAG, una representación de los pasos que deben ejecutarse en paralelo. Este DAG se optimiza antes de su ejecución para reducir la latencia y maximizar la eficiencia.
- Envía tareas a los Executors: Una vez optimizado el DAG, el Driver divide el trabajo en stages (etapas) y tasks (tareas), que son enviadas a los executors para su ejecución.
- Monitorea y gestiona tareas: El Driver recopila información sobre el estado de las tareas ejecutadas por los executors y gestiona posibles fallos.

Executors

¿Qué hacen los Executors?

- Ejecutan tareas distribuidas: Cada executor recibe una parte del trabajo (tasks) y las ejecuta de manera distribuida.
- Almacenan datos en memoria: Los datos intermedios se almacenan en memoria para mejorar la eficiencia en tareas iterativas.
- Reportan progreso al Driver: Los executors informan el estado de sus tareas y los resultados obtenidos al Driver.

Tipos de Tareas Ejecutadas por los Executors:

- Transformaciones: Aplican funciones como map(), filter(), reduce(), etc.
- Acciones: Recopilan resultados, como collect() o count().

Cluster Manager

¿Qué hace el Cluster Manager?

- Asigna recursos: Determina cuántos executors se necesitan para ejecutar las tareas y asigna la memoria y CPU requeridas.
- Gestiona nodos: Monitorea los nodos del clúster y reasigna tareas si un nodo falla o si se necesita escalar.
- Comunica información al Driver: Mantiene informado al Driver sobre el estado de los ejecutores y el progreso de las tareas.

El Driver Program es el proceso maestro que inicia, coordina y supervisa la ejecución de una aplicación Spark. Es el punto de entrada principal de una aplicación Spark y controla el flujo del programa.

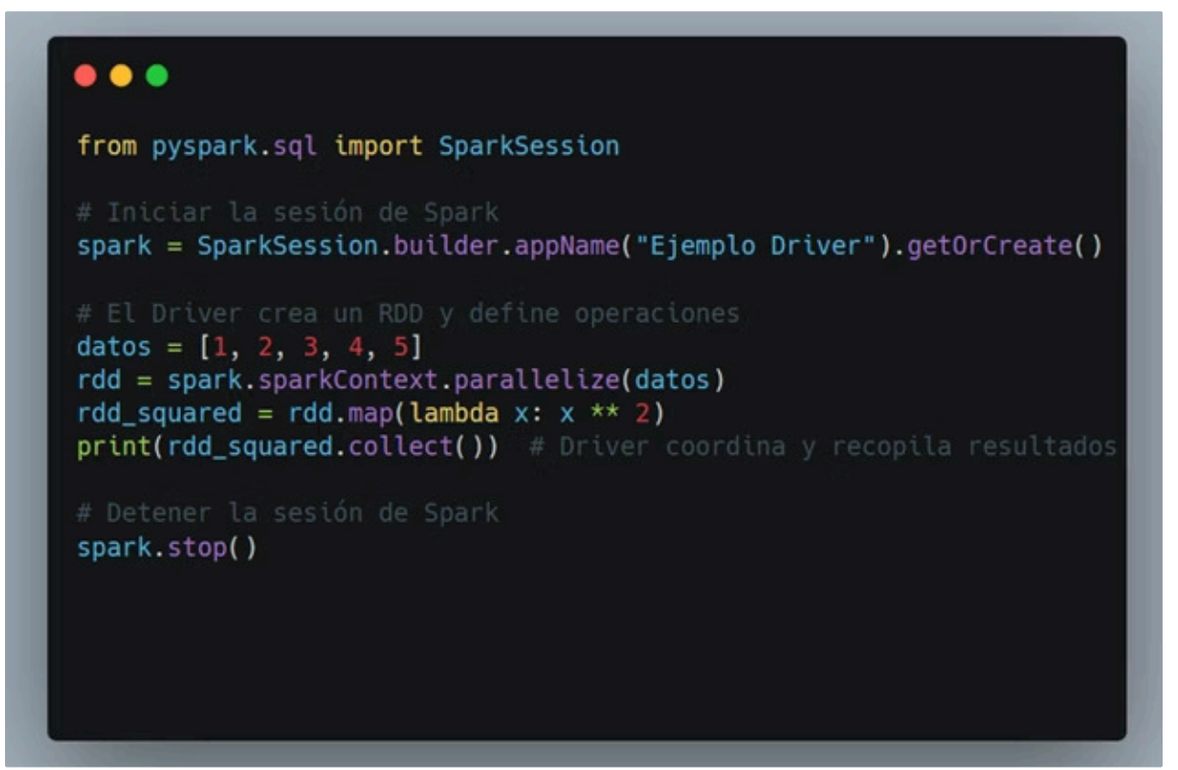
¿Qué hace el Driver Program?

Crea el DAG (Directed Acyclic Graph): El Driver convierte las transformaciones y acciones del programa Spark en un **DAG**, una representación de los pasos que deben ejecutarse en paralelo. Este DAG se optimiza antes de su ejecución para reducir la latencia y maximizar la eficiencia.

Envía tareas a los Executors: Una vez optimizado el DAG, el Driver divide el trabajo en stages (etapas) y tasks (tareas), que son enviadas a los executors para su ejecución.

Monitorea y gestiona tareas: El Driver recopila información sobre el estado de las tareas ejecutadas por los executors y gestiona posibles fallos.

Ejemplo en PySpark



```python
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()
```

El **Cluster Manager** es el componente encargado de **asignar recursos y gestionar los nodos** del clúster. Es responsable de coordinar la distribución de tareas entre los **executors** y supervisar su ejecución.

¿Qué hace el Cluster Manager?

Asigna recursos: Determina cuántos executors se necesitan para ejecutar las tareas y asigna la memoria y CPU requeridas.

Gestiona nodos: Monitorea los nodos del clúster y reasigna tareas si un nodo falla o si se necesita escalar.

Comunica información al Driver: Mantiene informado al Driver sobre el estado de los ejecutores y el progreso de las tareas.

Los **Executors** son procesos que **ejecutan tareas** asignadas por el Driver y devuelven los resultados. Corren en los nodos del clúster y manejan el procesamiento de datos de forma distribuida.

¿Qué hacen los Executors?

Ejecutan tareas distribuidas: Cada executor recibe una parte del trabajo (tasks) y las ejecuta de manera distribuida.

Almacenan datos en memoria: Los datos intermedios se almacenan en memoria para mejorar la eficiencia en tareas iterativas.

Reportan progreso al Driver: Los executors informan el estado de sus tareas y los resultados obtenidos al Driver.

Tipos de Tareas Ejecutadas por los Executors:

Transformaciones: Aplican funciones como map(), filter(), reduce(), etc.

Acciones: Recopilan resultados, como collect() o count().

Ejemplo en PySpark:

```
# 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 E IMPLEMENTACIONES DE SPARK

El modo de funcionamiento en cluster permite que una aplicación Spark ejecute tareas en nodos distribuidos de un clúster, 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.

El modo cluster es ideal cuando:

- Grandes volúmenes de datos: Los datos son demasiado grandes para procesarlos en una sola máquina.
- Procesamiento distribuido: Se requiere paralelismo para tareas complejas.
- Entornos de producción: Aplicaciones en producción que necesitan alta disponibilidad, escalabilidad y tolerancia a fallos.

Principales formas de implementar Spark

- Modo Local (Local Mode): En el modo local, Spark se ejecuta en una sola máquina.
- Modo Cluster (Cluster Mode): Spark se ejecuta en un clúster distribuido.
- Modo Cloud (Cloud-Native): permite ejecutar Spark en plataformas de nube.

Modo de Ejecución	Características	Uso recomendado
Local Mode	Se ejecuta en una sola máquina.	Desarrollo y pruebas locales.
Standalone Cluster	Administración nativa de Spark.	Clústeres pequeños o de prueba.
YARN Cluster	Integración con ecosistema Hadoop.	Entornos empresariales Hadoop.
Kubernetes Cluster	Orquestación de contenedores.	Entornos Cloud-native o híbridos.

El modo de funcionamiento en cluster de Apache Spark permite ejecutar tareas de procesamiento de datos de manera distribuida, aprovechando la potencia de múltiples nodos. Con opciones flexibles como Standalone, YARN y Kubernetes, Spark puede adaptarse a diferentes entornos y requisitos, proporcionando escalabilidad, rendimiento y tolerancia a fallos.

IMPLEMENTACIONES DE SPARK

La implementación de Apache Spark consiste en configurar y ejecutar aplicaciones Spark en diferentes entornos o plataformas, dependiendo de los requisitos del sistema y del volumen de datos. Estas implementaciones pueden ejecutarse de forma local, en clusters dedicados o en entornos cloud-native.

Principales formas de implementar Spark

- Modo Local (Local Mode), En el modo local, Spark se ejecuta en una sola máquina.
- Modo Cluster (Cluster Mode), Spark se ejecuta en un clúster distribuido.
- Modo Cloud (Cloud-Native), permite ejecutar Spark en plataformas de nube.

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 gestionadoy 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 Y ACTIVIDAD PRÁCTICA GUIADA

PySpark

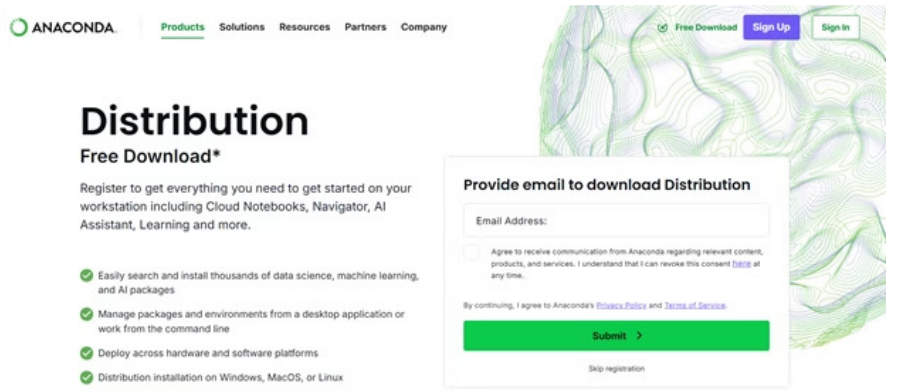
PySpark es la API de Apache Spark para Python que permite ejecutar tareas de procesamiento de datos distribuidos en clústeres Spark desde scripts escritos en Python. PySpark ofrece una interfaz de alto nivel para interactuar con el motor de Apache Spark, lo que permite a los desarrolladores trabajar 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.

Instalación de Anaconda

https://www.anaconda.com/download/



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:

Instalación de PySpark