



# Cos'è Apache Spark

*Programmazione*

**Apache Spark è un framework open-source per l'elaborazione distribuita dei dati, progettato per gestire in modo estremamente rapido grandi volumi di informazioni su cluster di macchine.**

**Nasce come evoluzione del modello MapReduce di Hadoop, migliorandone drasticamente le prestazioni grazie all'uso massiccio della memoria RAM: invece di scrivere continuamente su disco tra un'operazione e l'altra, Spark conserva i dati in memoria, riducendo la latenza e rendendo possibile l'analisi interattiva, il machine learning e lo streaming in tempo reale.**

**La sua architettura si basa su un driver che coordina il lavoro e su più executor che processano i dati in parallelo, distribuendo automaticamente il carico e gestendo la tolleranza ai guasti.**



**Dal punto di vista funzionale, Spark fornisce un ecosistema completo per diverse tipologie di workload.**

**Spark SQL permette di interrogare dati strutturati con un linguaggio simile all'SQL; MLlib offre algoritmi di machine learning scalabili; GraphX consente l'elaborazione di grafi complessi; e Spark Streaming (oggi Structured Streaming) consente di processare flussi continui di dati in near real-time.**

**Grazie alla sua flessibilità, al supporto per linguaggi come Python, Scala, Java e R, e alla possibilità di integrazione con sistemi come Hadoop HDFS, S3, Kafka e molti altri, Spark è diventato uno degli strumenti principali nel panorama Big Data per analisi veloci, scalabili e distribuite.**



- **In-memory Computing**

**Spark elabora i dati direttamente in RAM, evitando continui accessi al disco come accade in Hadoop MapReduce.**

**Questo approccio riduce drasticamente la latenza e permette performance anche 100 volte superiori su workload complessi.**

**I dati vengono memorizzati in strutture distribuite (RDD o DataFrame) che possono essere riutilizzate senza ricalcoli inutili.**



- **RDD (Resilient Distributed Dataset) e DataFrame**

**Spark si basa su due modelli di dati:**

- **RDD: collezioni distribuite immutabili, tolleranti ai guasti e processate in parallelo. Sono concettualmente simili a liste distribuite su cluster.**
- **DataFrame e Dataset: livello più alto, ottimizzato grazie a Catalyst (optimizer interno) e a una struttura tabellare simile a SQL.**

**Questo consente sia controllo fine (RDD) sia ottimizzazioni automatiche (DataFrame/Dataset).**



- **Lazy Evaluation**

**Le trasformazioni applicate ai dataset (es. map, filter, select) non vengono eseguite subito.**

**Spark costruisce una pipeline logica delle operazioni e le esegue solo quando necessario, cioè quando viene invocata un'azione (es. collect, count, show).**

**Questo permette di ottimizzare l'esecuzione e ridurre i passaggi non necessari.**



- **DAG Scheduler**

**Spark costruisce un Directed Acyclic Graph (DAG) per descrivere la sequenza di operazioni.**

**Il DAG Scheduler suddivide il lavoro in task paralleli, li raggruppa in stage e li assegna ai nodi del cluster, ottimizzando:**

- parallelismo
- località dei dati
- recupero dai guasti

**È il cuore del sistema di esecuzione distribuita.**



- **Tolleranza ai Guasti**

**Gli RDD sono resilienti: Spark riesce a ricostruire le partizioni perse usando la lineage (il “ciclo di vita” delle operazioni che li hanno generati).**

**Se un nodo del cluster fallisce, Spark ricalcola solo la parte danneggiata, garantendo continuità.**



- Cluster Manager Multipli

**Spark può lavorare con diversi sistemi di gestione del cluster:**

- Standalone (nativo Spark)
- YARN (Hadoop)
- Mesos
- Kubernetes

**Questa compatibilità lo rende facilmente adottabile in contesti diversi.**



- API Multi-Lingua

**Spark supporta ufficialmente:**

- **Scala (linguaggio nativo)**
- **Python (PySpark)**
- **Java**
- **R**

**Questo permette l'integrazione in progetti molto diversi, dal data engineering al machine learning.**



- **Componenti Integrati**

**Spark non è solo un motore di calcolo, ma un ecosistema completo:**

- **Spark SQL: query SQL su DataFrame, gestione tabelle e cataloghi.**
- **MLlib: libreria di machine learning scalabile.**
- **GraphX: analisi e algoritmi su grafi.**
- **Structured Streaming: stream processing in near real-time.**

**Tutto utilizza lo stesso motore, semplificando pipelines dati complesse.**





Caratteristica	Descrizione Tecnica
In-memory Computing	L'elaborazione avviene prevalentemente in RAM, riducendo la latenza rispetto ai sistemi basati su disco ed accelerando analisi iterative o complesse.
RDD e DataFrame	RDD: dataset distribuiti immutabili, tolleranti ai guasti. DataFrame/Dataset: API ottimizzate con struttura tabellare per elaborazioni più efficienti e dichiarative.
Lazy Evaluation	Le trasformazioni vengono accumulate e valutate solo al momento di un'azione, permettendo ottimizzazioni dell'intera pipeline.
DAG Scheduler	Spark costruisce un grafo aciclico (DAG) delle operazioni, suddivide il lavoro in stage e task e li distribuisce sui nodi del cluster.
Tolleranza ai Guasti	Ricostruzione automatica delle partizioni perse tramite lineage degli RDD; ricalcolo solo delle parti coinvolte nel guasto.
Cluster Manager Multipli	Supporto per Standalone, YARN, Mesos e Kubernetes, permettendo l'esecuzione in ambienti diversi con piena portabilità.
API Multi-lingua	Supporto nativo per Scala, Python (PySpark), Java e R, integrabili in flussi ETL, machine learning e applicazioni distribuite.
Componenti Integrati	Moduli come Spark SQL, MLlib, GraphX e Structured Streaming che coprono query, ML, grafi e stream processing usando lo stesso motore.
Scalabilità Orizzontale	Possibilità di aumentare capacità e performance aggiungendo nodi, con bilanciamento automatico del carico e parallelismo efficiente.

## Guida Installazione Apache Spark

- **Installa Java (obbligatorio)**

Vai sul sito Adoptium, scarica e installa una versione di Java 8 o superiore.

Dopo l'installazione, verifica il comando `java -version` per assicurarti che Java sia visibile nel sistema.

- **Installa Python (se vuoi usare PySpark)**

Se intendi lavorare con PySpark, installa Python dal sito ufficiale.

Controlla l'installazione con `python --version`.

- **Scarica Apache Spark**

Vai alla pagina ufficiale di download di Apache Spark.

Scegli una versione stabile e scarica il pacchetto “Pre-built for Apache Hadoop”.

Estrai il contenuto in una cartella del tuo sistema (ad esempio C:\spark su Windows oppure /opt/spark su macOS/Linux).

- **Aggiungi Spark al PATH**

Inserisci le cartelle bin e sbin del tuo Spark nelle variabili d'ambiente PATH, così da poter usare i comandi Spark da terminale.

Su Windows modifica le variabili d'ambiente; su macOS/Linux aggiungi le righe necessarie al tuo file .bashrc o .zshrc.

- **(Solo Windows) Installa winutils.exe**

Spark su Windows richiede winutils.exe per simulare alcune funzioni di Hadoop.

Scaricalo dalla repo dedicata, creane una cartella (ad esempio C:\hadoop\bin\) e aggiungi la variabile d'ambiente HADOOP\_HOME, indicando la cartella principale.

- **Verifica l'installazione**

Apri il terminale e lancia i comandi `spark-shell` per la shell Scala e `pyspark` per PySpark.

Se la console si avvia senza errori, Spark è correttamente installato.

- **Esegui un test minimo**

Avvia `pyspark` e prova a creare una semplice sessione Spark e un DataFrame.

Se il comando `df.show()` restituisce un output, Spark è operativo.



## Spiegazione

- **parallelize crea un RDD distribuendo i dati sul cluster.**
- **map è una trasformazione: Spark costruisce il DAG ma non esegue ancora nulla.**
- **filter è un'altra trasformazione che filtra i valori.**
- **collect è un'azione: obbliga Spark a eseguire davvero tutta la pipeline e restituisce i risultati al driver.**
- **Output finale: solo i quadrati maggiori di 10 → [16, 25].**

```
1.from pyspark.sql import SparkSession
2.
3.# Creazione della sessione Spark
4.spark = SparkSession.builder.appName("Esempio1").getOrCreate()
5.
6.# Creiamo un RDD da una lista
7.numeri = spark.sparkContext.parallelize([1, 2, 3, 4, 5])
8.
9.# Trasformazioni (lazy)
10.quadrati = numeri.map(lambda x: x * x)
11.filtrati = quadrati.filter(lambda x: x > 10)
12.
13.# Azione (trigger dell'esecuzione)
14.risultato = filtrati.collect()
15.
16.print(risultato)
```



## Spiegazione

- **createDataFrame crea un DataFrame distribuito con schema (nome, eta).**
- **filter(df.eta > 25) applica una trasformazione su colonna simile a SQL.**
- **show() è un'azione che stampa le righe filtrate.**

**Output:**

nome	eta
Mario	30
Anna	40

```
1.from pyspark.sql import SparkSession
2.
3.# Sessione Spark
4.spark = SparkSession.builder.appName("Esempio2").getOrCreate()
5.
6.# Creiamo un DataFrame da una lista di tuple
7.dati = [("Mario", 30), ("Luca", 22), ("Anna", 40)]
8.df = spark.createDataFrame(dati, ["nome", "eta"])
9.
10.# Selezioniamo solo le persone sopra i 25 anni
11.adulti = df.filter(df.eta > 25)
12.
13.# Visualizziamo il risultato
14.adulti.show()
```





**Buon Davante a tutti**