Introduzione Spark

Spark

Che cos’è?

Capiamone le basi

- Apache Spark è un motore di analisi open-source utilizzato per carichi di lavoro Big Data.

- È in grado di gestire l'analisi e l'elaborazione dei dati sia in batch sia in tempo reale.

    - **Batch Processing** (elaborazione in batch): In questa modalità, i dati vengono raccolti, processati e analizzati in lotti o gruppi.

    - **Real-time Processing** (elaborazione in tempo reale): In questa modalità, i dati vengono elaborati man mano che arrivano, senza dover aspettare che si accumulino in lotti.

Spark

Che cos’è?

Capiamone le basi e perché è nato

- Apache Spark è nato nel 2009 come progetto di ricerca presso l'Università di Berkeley, in California con l'obiettivo di trovare un modo per velocizzare l'elaborazione dei dati nei sistemi Hadoop.

Cos’è Hadoop?

- Apache **Hadoop** è un framework software e un motore di elaborazione parallela open-source basato su Java. Consente di suddividere le attività di analisi dei Big Data in attività più piccole che possono essere eseguite in parallelo utilizzando un algoritmo (come l'algoritmo **MapReduce**) e distribuendole su un cluster Hadoop.

Spark

Che cos’è?

MapReduce

- **Esigenza**: centinaia di computazioni al giorno che processano grandi quantità di dati, come documenti, logs o pagine web per ottenere dati strutturati.

- Una computazione può essere vista come una **sequenza finita di round**,ognuno dei quali è composto da una funzione di map e una di reduce

- Durante la computazione di ogni round:

    - La funzione di **Map** prende in input una coppia chiave/valore e restituisce una coppia di valori intermedi chiave/valore.

    - La funzione **Shuffle** raggruppa tutti i valori intermedi con stessa chiave e la passa in input alla funzione di Reduce.

    - La funzione di **Reduce** prende in input una coppia di chiave intermedia/lista di valori, fa operazioni su questi valori e restituisce un insieme di valori.

Input →
files con un documento per record

"document1", "to be or not to be"

funzione Map →
<$nome_{doc}$, $contenuto_{doc}$>
<parola, 1>

input:
e output:

funzione Reduce →
<parola, lista di 1>
output: <parola, $occorrenze_{\#}$>

input:
e

Input →

files con un documento per record

“document1”, “to be or not to be”

↓

“to”, “1”
“be”, “1”
“or”, “1”
...

funzione Map → input:
<*nome*$_{doc}$, *contenuto*$_{doc}$> e output:
<*parola*, *1*>

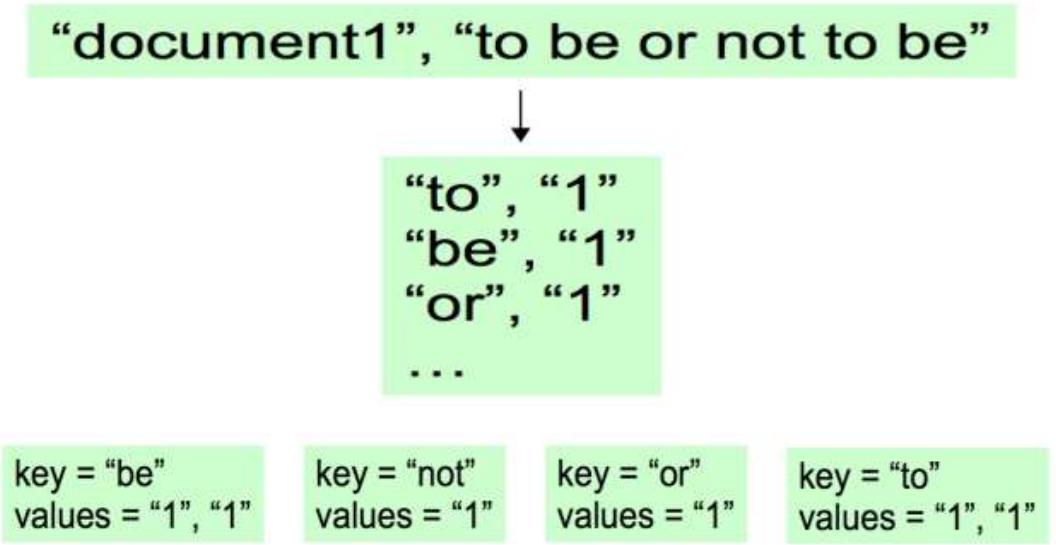
funzione Reduce → input:
<*parola, lista di 1*> *e*
output: <*parola, occorrenze*$_{\#}$>

Input →

files con un documento per record

funzione Map → input:
<nome$_{doc}$, contenuto$_{doc}$> e output:
<parola, 1>

funzione Reduce → input:
<parola, lista di 1> *e*
output: <parola, occorrenze$_{\#}$>



Input →

files con un documento per record

funzione Map → input:

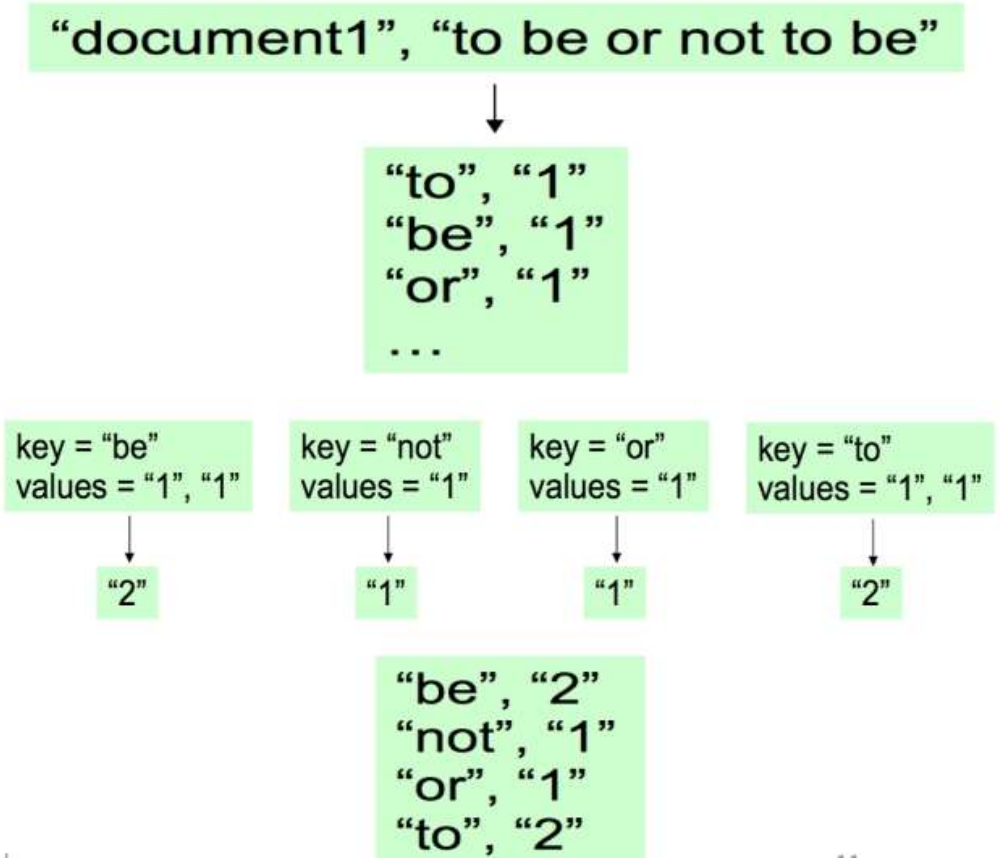
<$nome_{doc}$, $contenuto_{doc}$> e output:

<parola, 1>

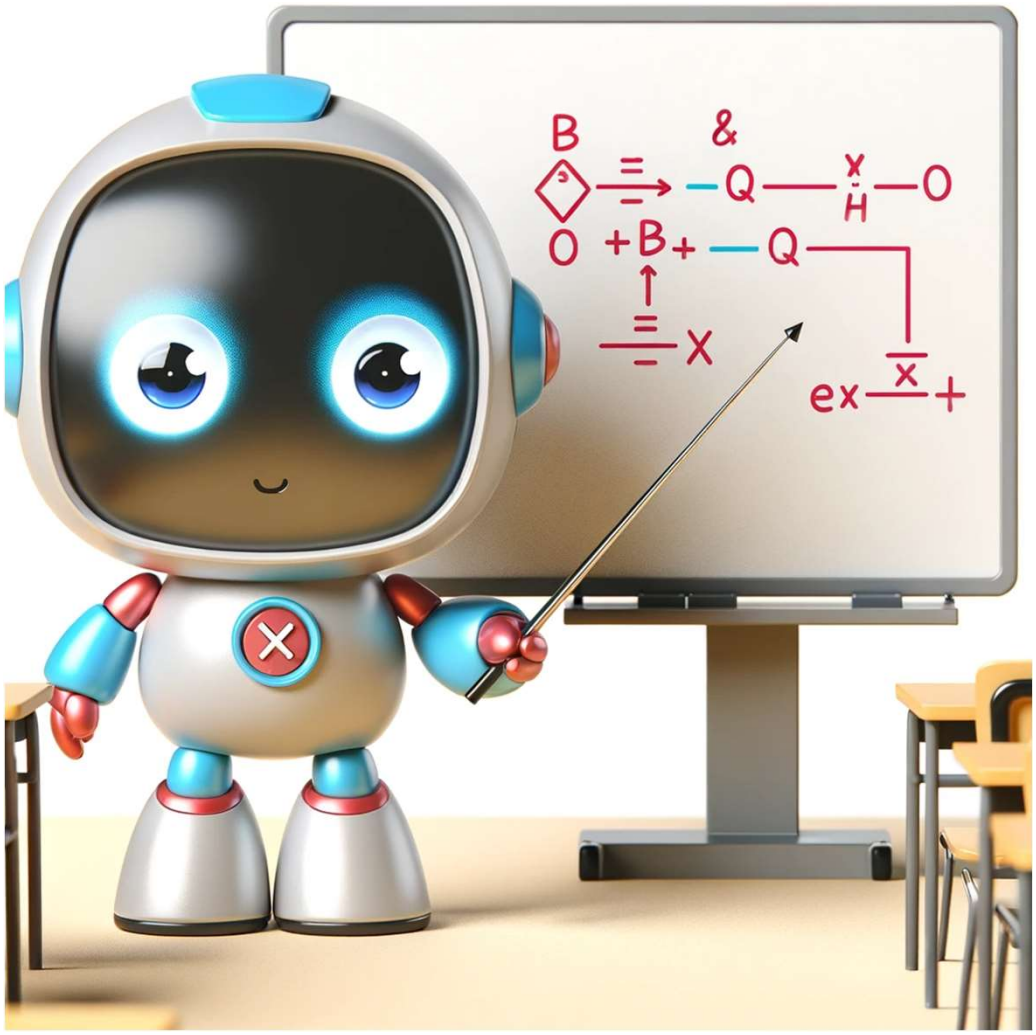
funzione Reduce → input:

<parola, lista di 1> e

output: <parola, $occorrenze_{\#}$>



Esempio



Spark

Che cos’è?

Capiamone le basi

- Questo approccio trova applicazione in una vasta gamma di scenari e settori, ecco alcuni esempi MapReduce può essere utilizzato:

  - **Conteggio delle parole in grandi collezioni di testi.**
  - **Analisi dei dati di log.**
  - **Analisi dei dati scientifici.**
  - **Elaborazione di dati di sensori e IoT.**

Spark

Che cos’è?

Capiamone le basi

- Sebbene Hadoop fosse inizialmente una soluzione innovativa per l'elaborazione di grandi quantità di dati, aveva alcuni problemi chiave che Apache Spark mirava a risolvere:
    - **Velocità**
    - **Facilità di sviluppo**
    - **Diversità di Workload**
    - **Efficienza del cluster**

Un motore informatico
unificato e un insieme
di librerie per big data
i suoi componenti chiave.

Spark

Che cos’è?

La filosofia dietro Spark

- **Unificato**: Spark è progettato per supportare un'ampia gamma di attività di analisi dei dati, che vanno dal semplice caricamento dei dati e query SQL all'apprendimento automatico e al calcolo in streaming, sullo stesso motore di elaborazione e con un insieme coerente di API.

- Prima di Spark, nessun sistema open source ha cercato di fornire questo tipo di motore unificato per l'elaborazione parallela dei dati, il che significa che gli utenti dovevano mettere insieme un motore unificato partendo da diversi sistemi e diverse API.

Un **motore informatico** unificato e un insieme di librerie per big data i suoi componenti chiave.

Spark

Che cos’è?

La filosofia dietro Spark

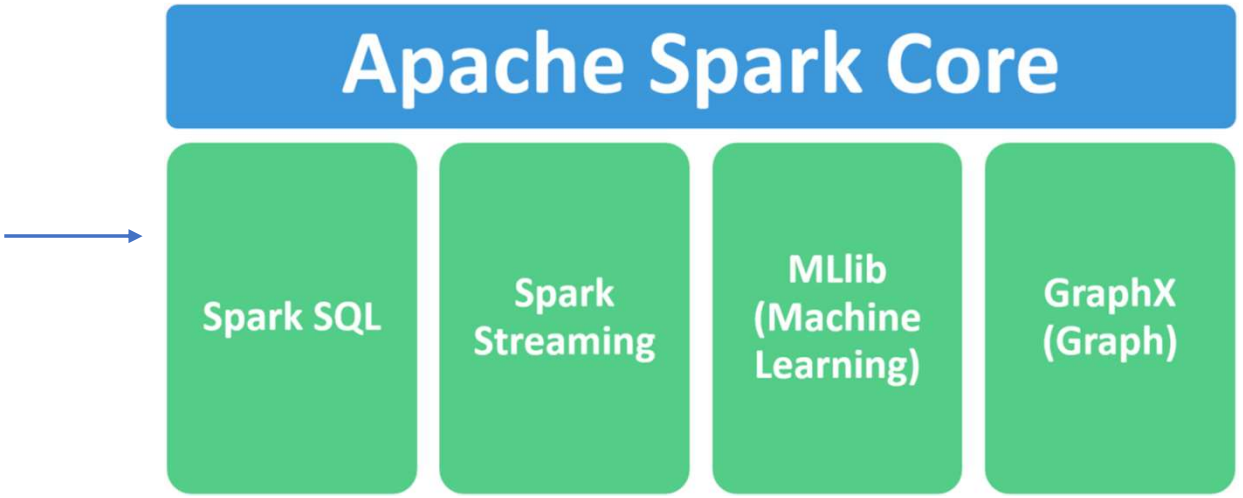
- **Motore di calcolo**: mentre Spark punta all'unificazione, Spark stesso limita attentamente il suo ambito ad essere un motore di calcolo.

- Spark può essere utilizzato con un'ampia varietà di storage persistente sistemi, inclusi sistemi di archiviazione cloud come Azure Storage e Amazon S3, file system distribuiti come Apache Hadoop, archivi di valori-chiave come Apache Cassandra e bus di messaggi come Apache Kafka.

Spark

Che cos’è?

Un motore informatico unificato e un insieme di **librerie** per big data i suoi componenti chiave.



Spark

Che cos’è?

Capiamone le basi

- Apache Spark si basa su Hadoop **MapReduce** e ne estende il modello per utilizzarlo in modo efficiente durante altri tipi di operazioni, incluse query interattive ed elaborazione di flussi di dati.

- Spark offre binding nativi per i linguaggi di programmazione Java, Scala, Python e R. Inoltre, include diverse librerie per supportare la creazione di applicazioni per il machine learning [MLlib], l'elaborazione di dati in streaming [streaming Spark] e l'elaborazione di grafici [GraphX].

Spark

Quali sono i vantaggi di Apache Spark?

Architettura Spark

Spark

Struttura basica di spark

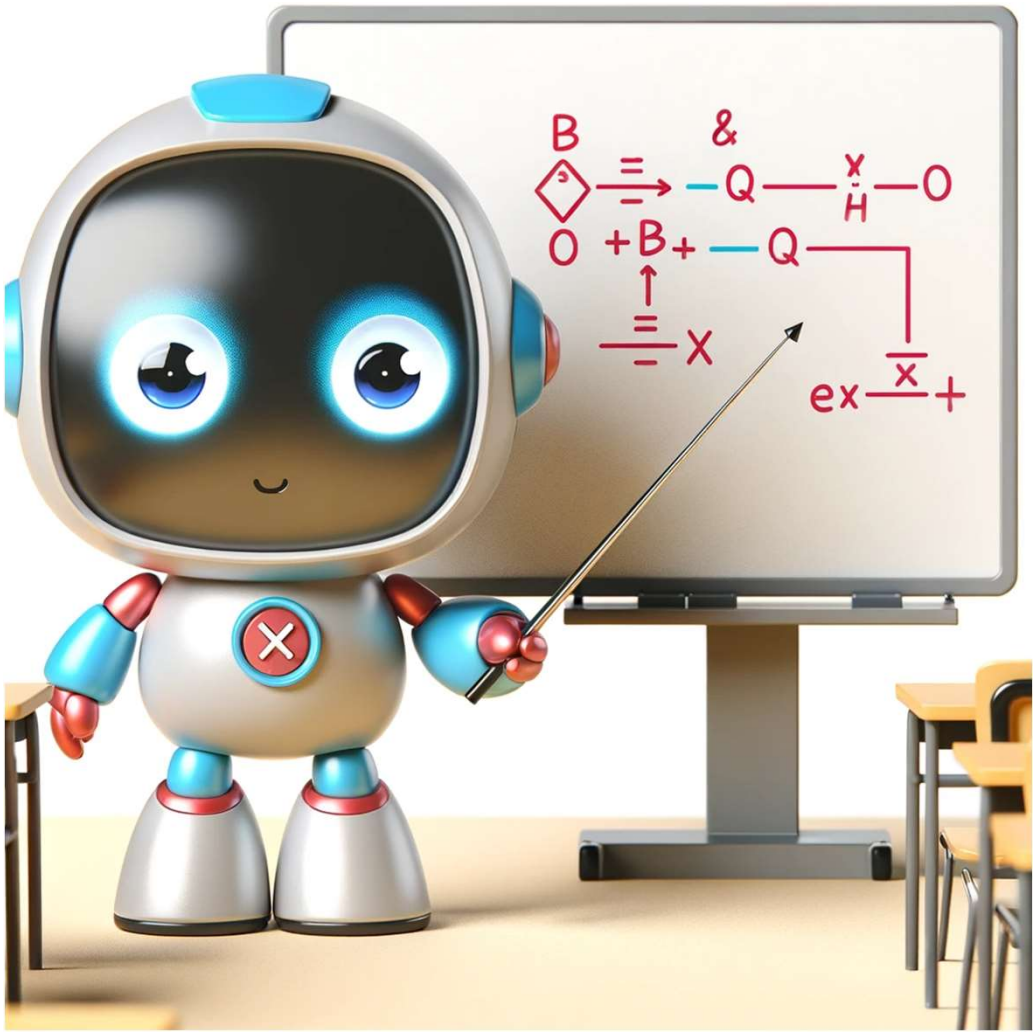
Entriamo nel dettaglio

- Un **cluster**, ossia un gruppo di macchine, mette insieme le risorse di molte macchine, permettendo di sfruttare tutte un insieme di risorse come se fossero una sola.

- Tuttavia, da solo, un gruppo di macchine non è sufficientemente potente; è necessaria una struttura per coordinarle e farle lavorare insieme in modo efficiente.

- **Spark** è uno strumento progettato appositamente per questo scopo: gestire e coordinare l'esecuzione di attività sui dati attraverso un gruppo di computer.

Esempio



Spark

Struttura basica di spark

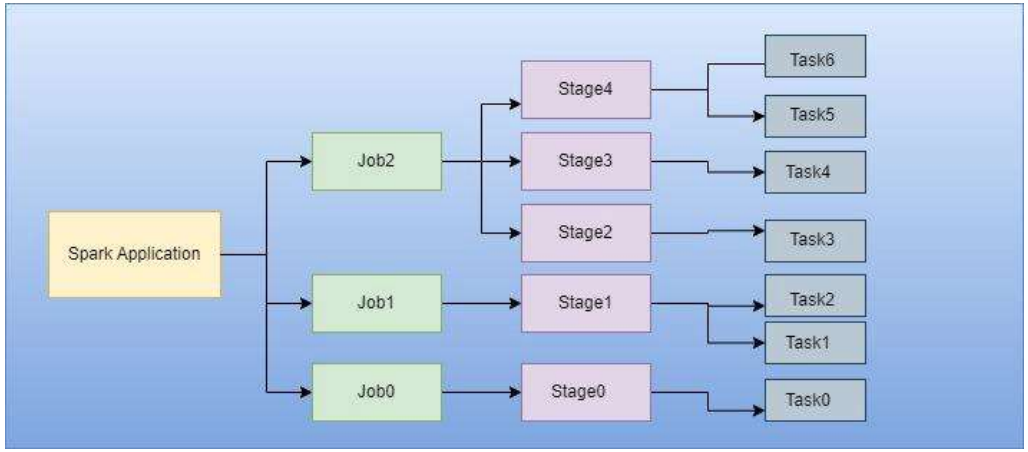
Azioni che triggerano la creazione di un job

- Nella nostra applicazione precedente, abbiamo eseguito 3 lavori Spark (0, 1, 2).
  - Job 0: lettura del file CSV.
  - Job 1: Inferire lo schema dal file.
  - Job 2: Verifica del conteggio.
- Quindi, se guardiamo lo screenshot sulla sinistra, mostra chiaramente i risultati dei 3 lavori Spark relativi alle 3 azioni.

Spark

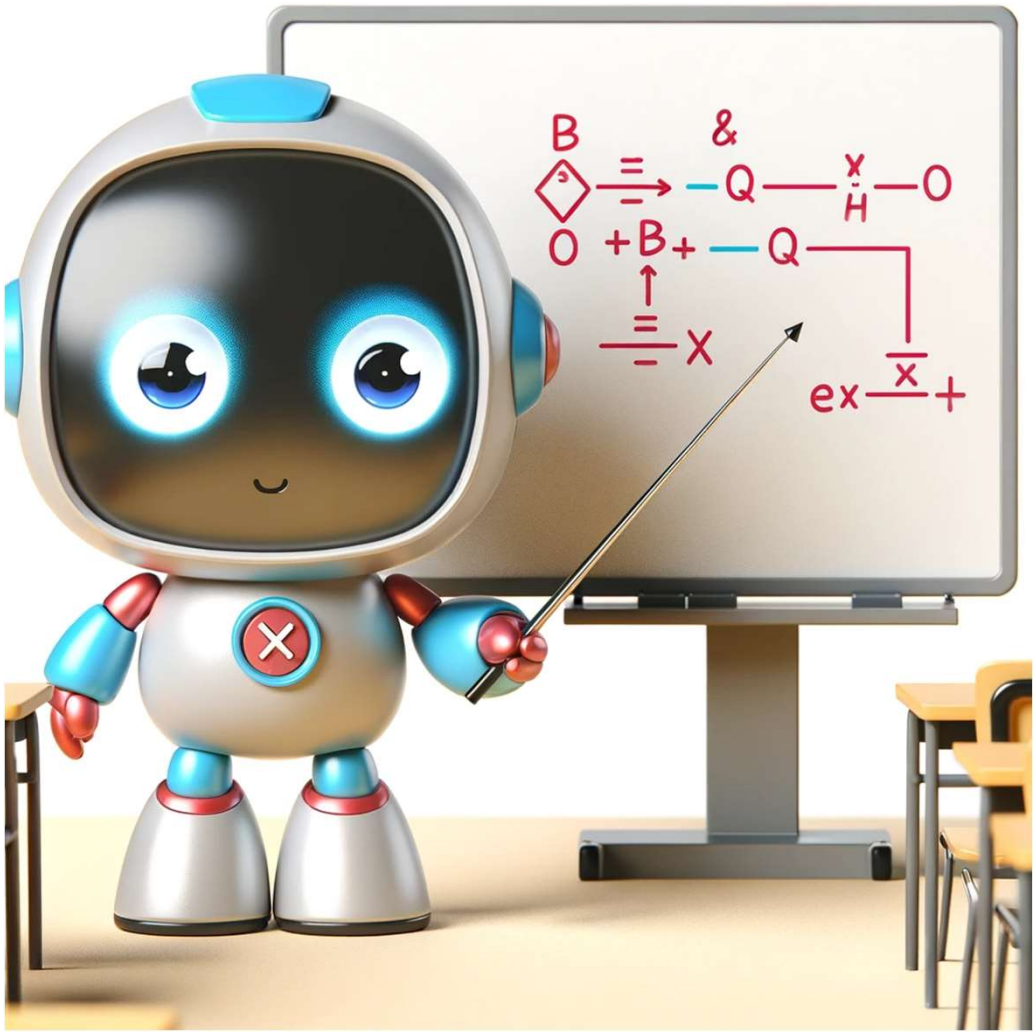
Struttura basica di spark

Cos’è uno Stage?



- Nel contesto di Apache Spark, uno **Stage** è un'unità di parallelismo in un lavoro Spark. Rappresenta un insieme di compiti che possono essere eseguiti insieme come parte di un singolo lavoro.
- L'applicazione Spark è suddivisa in diversi Job per ogni azione che verrà eseguita e i Job sono suddivisi in Stage, gli Stage infine sono suddivisi in task.
- Uno Stage è una collezione di compiti che condividono le stesse dipendenze, il che significa che devono scambiarsi dati l'uno con l'altro durante l'esecuzione.

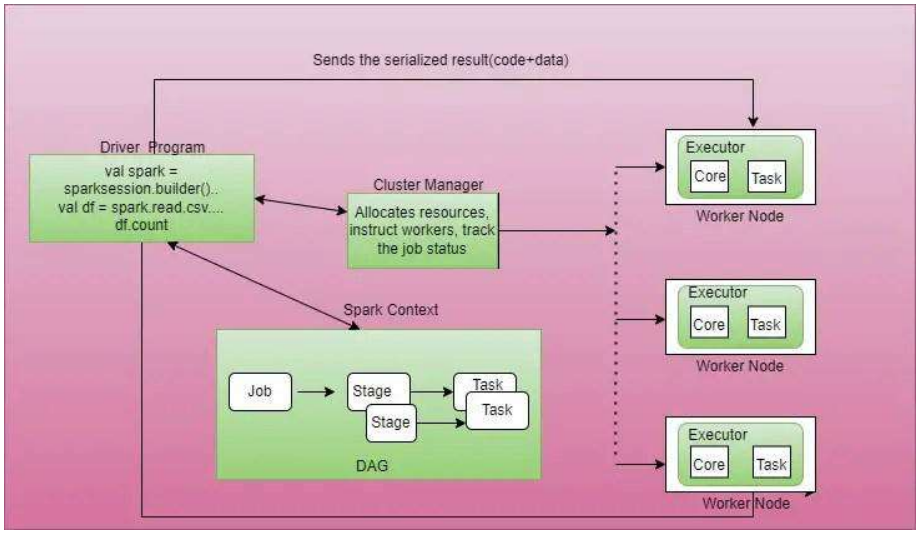
Esempio



Spark

Struttura basica di spark

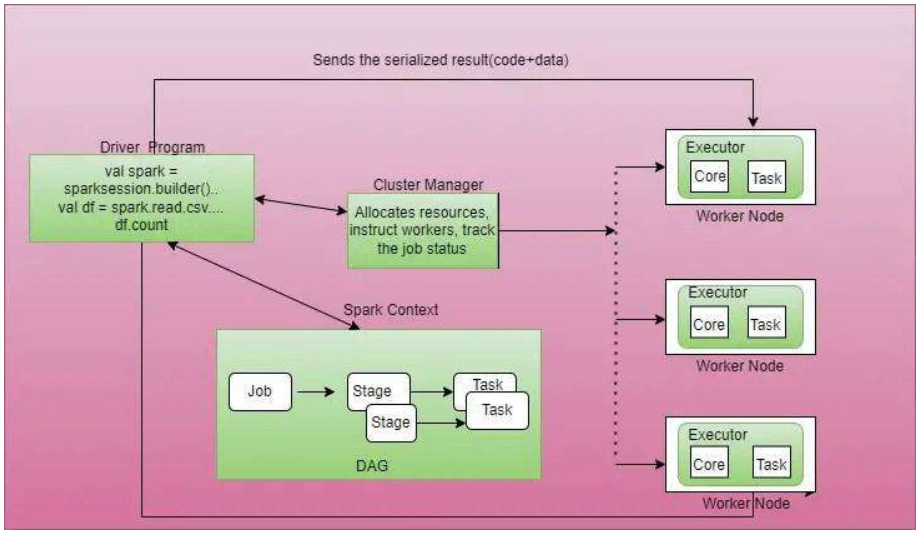
Spark Workers



- L'Executor di Spark è un processo che viene eseguito su un nodo **worker** in un cluster Spark ed è responsabile dell'esecuzione dei compiti assegnati ad esso dal programma driver di Spark.

- Punti chiave dell'Executor di Spark:

    - Esegue compiti sui nodi worker come indicato dal Driver.
    - Più Executor vengono eseguiti contemporaneamente in un'applicazione Spark.
    - Creato quando viene creato un SparkContext e viene eseguito fino alla terminazione dell'applicazione.
    - Comunica con il Driver per l'assegnazione dei compiti e segnala lo stato dei compiti.



Spark

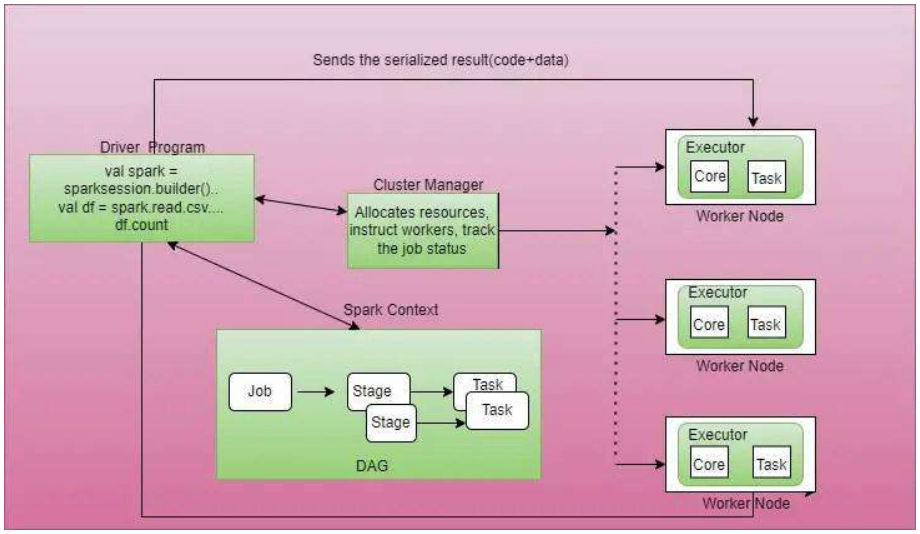
Struttura basica di spark

Spark Workers

- Gli Executor sono i pilastri di un'applicazione Spark, poiché eseguono le effettive computazioni sui dati.
- Quando un programma driver Spark invia un compito a un cluster, questo viene diviso in unità di lavoro più piccole chiamate ‘’**job**".
- Ogni Executor viene allocato con una certa quantità di risorse di memoria e CPU quando viene avviato e utilizza questa memoria per memorizzare i dati in memoria per un accesso più veloce durante le computazioni.
- Gli Executor gestiscono anche i dati memorizzati nella cache e sul disco e gestiscono le operazioni di shuffle (quando i dati devono essere scambiati tra i nodi).

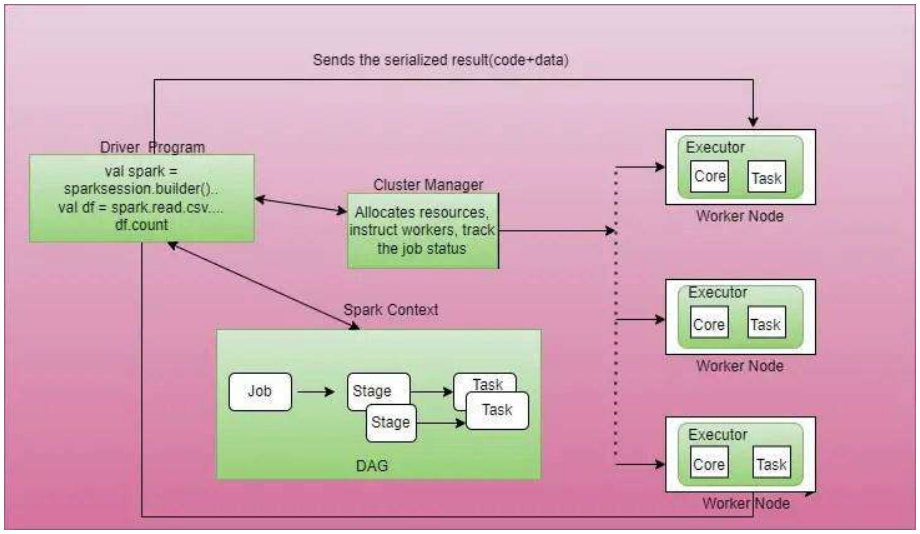
Spark

Struttura basica di spark



Tipi di Esecutori

- In Apache Spark, ci sono diversi tipi di Executor che possono essere utilizzati in base alle esigenze dell'applicazione. Questi sono:
    - **Default Executor.**
    - **Coarse-Grained Executor.**
    - **Fine-Grained Executor.**
    - **External Executors.**
- Ogni tipo di Executor ha vantaggi e svantaggi propri e la scelta dell'Executor dipende dalle esigenze dell'applicazione. Ad esempio, se l'applicazione ha un grande set di dati da elaborare, potrebbe essere più adatto un Coarse-Grained Executor, mentre se l'applicazione ha molti piccoli compiti, potrebbe essere più appropriato un Fine-Grained Executor.



Spark

Struttura basica di spark

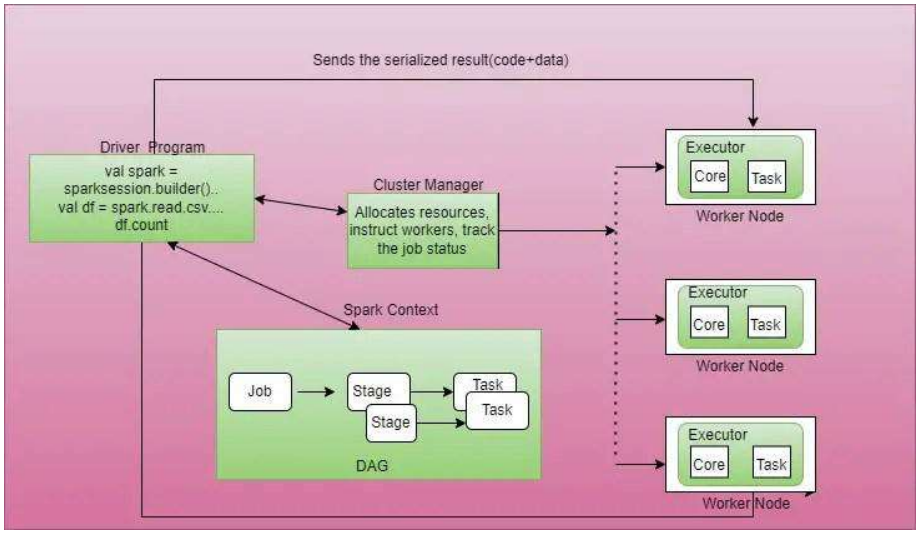
Parametri di configurazione

- Apache Spark fornisce una serie di opzioni di configurazione per gli Executor che possono essere utilizzate per ottimizzare le loro prestazioni e l'utilizzo delle risorse. Ecco alcune delle principali opzioni di configurazione:
    - **Memoria dell'Executor.**
    - **Core dell'Executor.**
    - **Numero di Executor.**
    - **Memoria di shuffle.**
- Queste sono solo alcune delle opzioni di configurazione disponibili per gli Executor di Spark. Ottimizzando queste impostazioni è possibile migliorare significativamente le prestazioni e l'utilizzo delle risorse delle applicazioni Spark.

Spark

Struttura basica di spark

Performance degli Esecutori di Spark



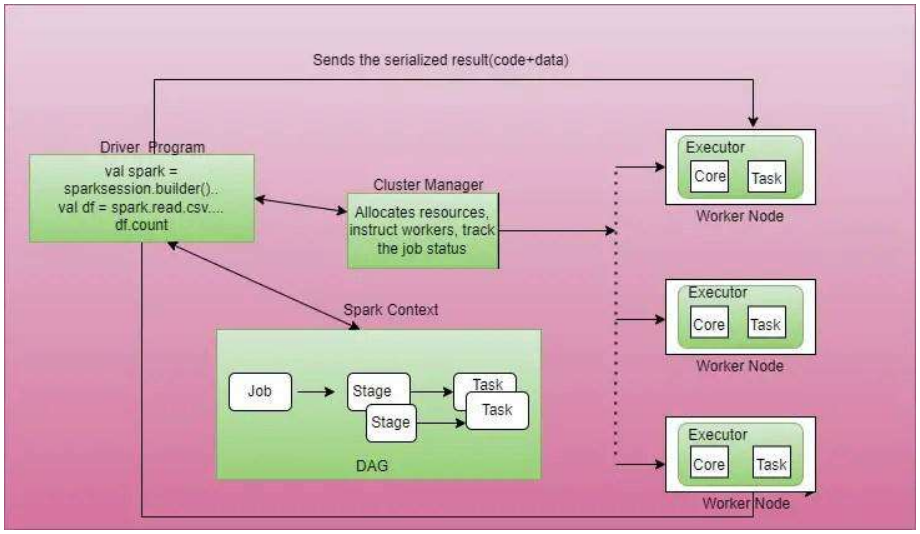
- Le prestazioni degli Executor di Spark possono avere un impatto significativo sulle prestazioni complessive di un'applicazione Spark. Ecco alcuni fattori che possono influenzare le prestazioni degli Executor di Spark:

  - Memoria.
  - CPU.
  - Rete.
  - Distribuzione dei dati.
  - Granularità dei compiti.

Spark

Struttura basica di spark

Usi degli esecutori

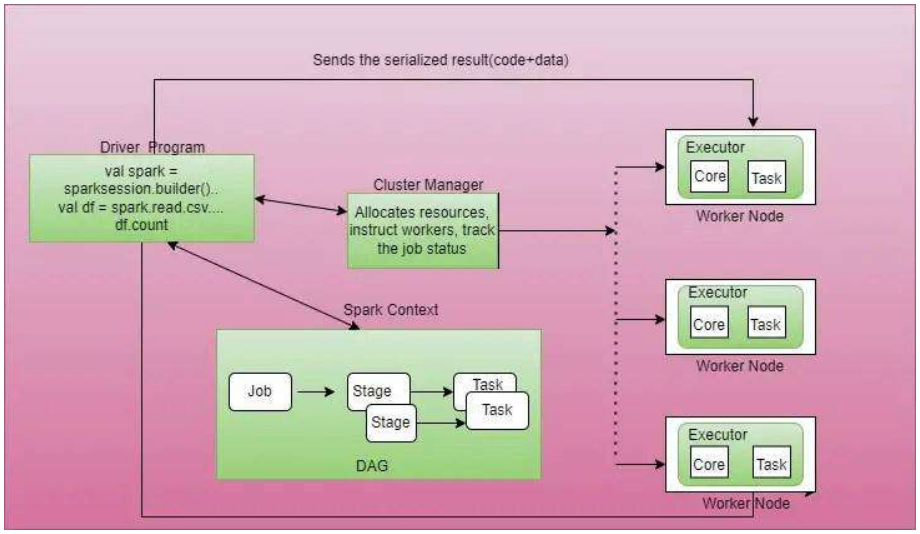


- Gli Executor di Spark sono i mattoni fondamentali di Apache Spark e svolgono un ruolo critico nel processare i dati in modo distribuito. Ecco alcuni casi d'uso comuni in cui gli Executor di Spark sono utilizzati:
    - Elaborazione dei dati.
    - Machine Learning.
    - Dati in streaming.
    - Elaborazione di grafi.
    - Analisi interattiva.
    - Elaborazione a batch.

Spark

Struttura basica di spark

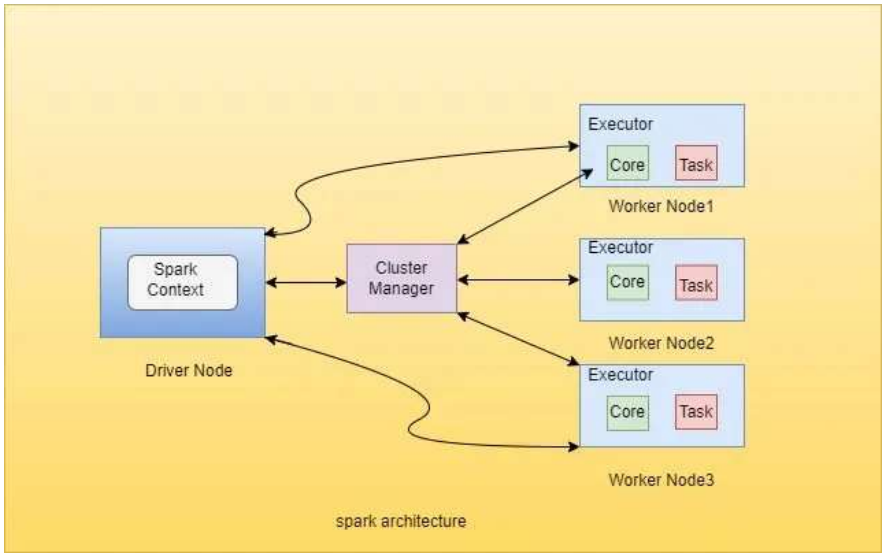
Come setto i parametri degli esecutori?



- Spark Execution Memory: La quantità di memoria allocata a un executor è determinata dal parametro di configurazione spark.executor.memory, che specifica la quantità di memoria da allocare per ogni executor. **Questo parametro viene impostato nel file di configurazione di Spark o tramite l'oggetto SparkConf nel codice dell'applicazione.**

- Il valore di spark.executor.memory può essere impostato in diversi modi, ad esempio:

    - Valore fisso.
    - Assegnazione dinamica.



Spark

Struttura basica di spark

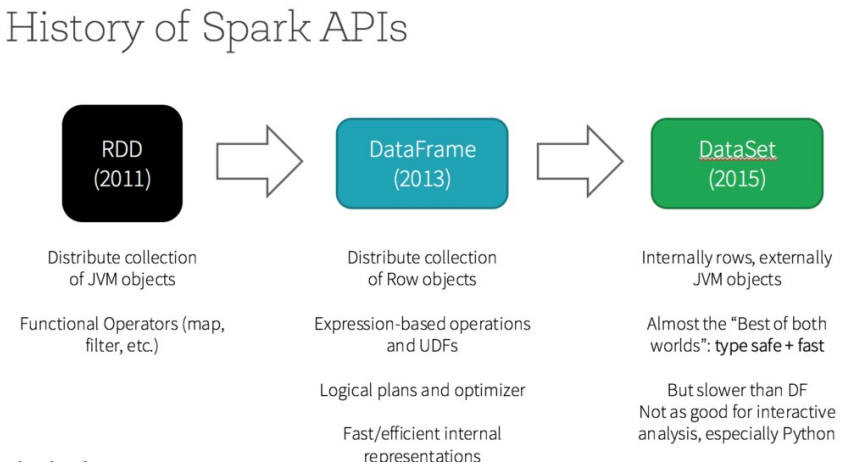
Spark Driver

- Il driver di Spark in Apache Spark è il componente principale che gestisce l'esecuzione complessiva di un'applicazione Spark.

- Nell'architettura master-slave di Spark, il driver è il nodo principale che coordina le attività e interagisce con il Cluster Manager per distribuire i compiti ai nodi worker. Alcuni punti chiave del driver di Spark includono:

    - Gestisce l'esecuzione complessiva di un'applicazione Spark.
    - C'è un solo driver per ogni applicazione Spark.
    - Responsabile della coordinazione dei compiti, della pianificazione e dell'interazione con il Cluster Manager.
    - Inizializza SparkContext, che rappresenta una connessione a un cluster Spark.
    - Monitora il progresso dell'esecuzione e garantisce la tolleranza ai guasti.

RDD



RDD

Cosa sono?

Introduzione RDD

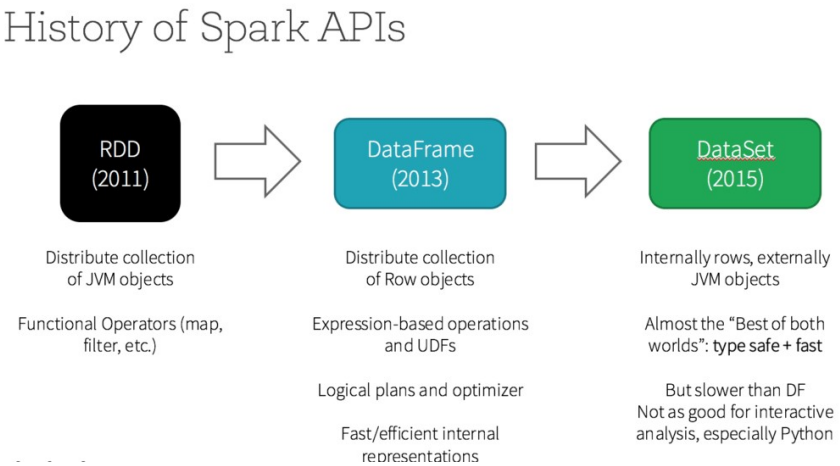
- **RDD** (Resilient Distributed Dataset) è una struttura dati fondamentale di Spark ed è l'astrazione primaria dei dati in Apache Spark e Spark Core.
- Gli RDD sono collezioni distribuite di oggetti **immutabili** con tolleranza agli errori, il che implica che una volta creato un RDD, **non è possibile modificarlo**.
- Ogni set di dati contenuto in un RDD è suddiviso in partizioni logiche, che possono essere elaborate su diversi nodi del cluster.

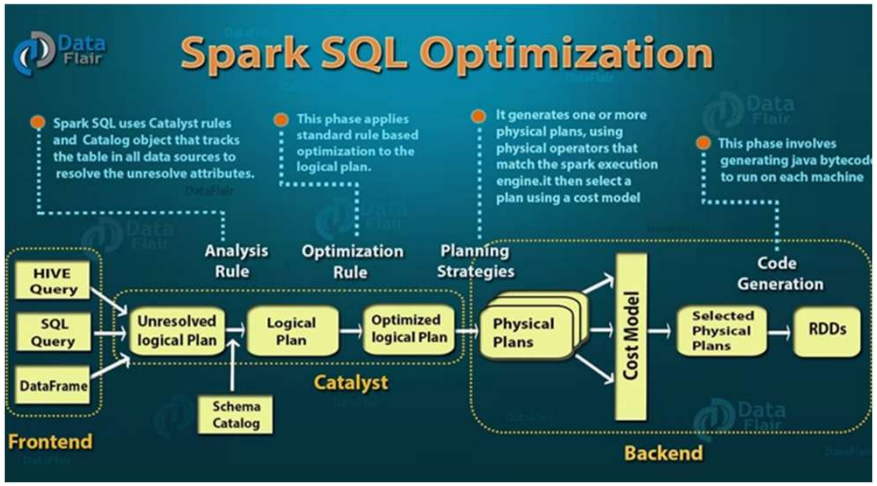
RDD

Cosa sono?

Vantaggi RDD

- Vantaggi nell'utilizzo degli RDD:
  - **Resilienza**
  - **Parallelismo**
  - **Tolleranza agli errori**
  - **Supporto a molteplici linguaggi**
  - **In memory computation**
  - **Lazy Evaluation**





RDD

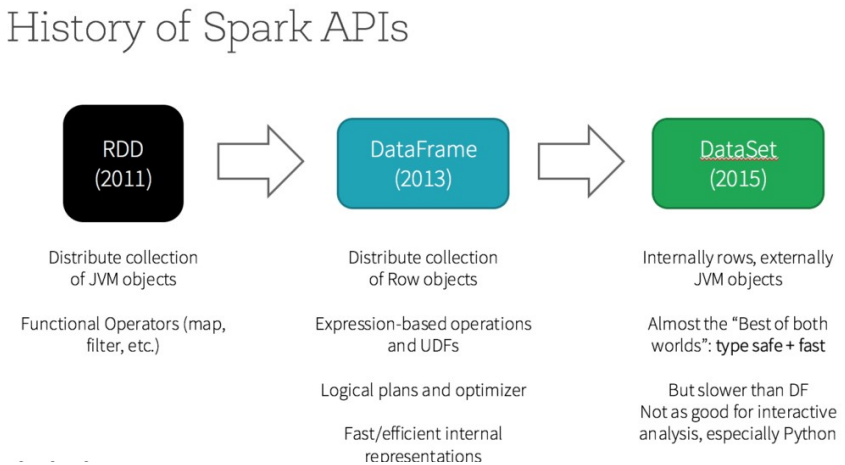
Cosa sono?

Limitazioni RDD

- **Nessun motore di ottimizzazione degli input:** L'RDD non prevede l'ottimizzazione automatica. Non può utilizzare ottimizzatori avanzati Spark come **catalyst optimizer** e il motore di esecuzione di **Tungsten**.

- **Nessun controllo sul tipo a run-time**: Gli RDD non dispongono di tipizzazione statica né di sicurezza del tipo in fase di esecuzione. Questo significa che non consentono di controllare gli errori in fase di esecuzione.

- **Limitazione delle prestazioni e sovraccarico della serializzazione e della garbage collection:** Poiché gli RDD sono oggetti JVM in memoria, ciò comporta il sovraccarico della Garbage Collection e inoltre la serializzazione Java diventa costosa quando i dati aumentano.

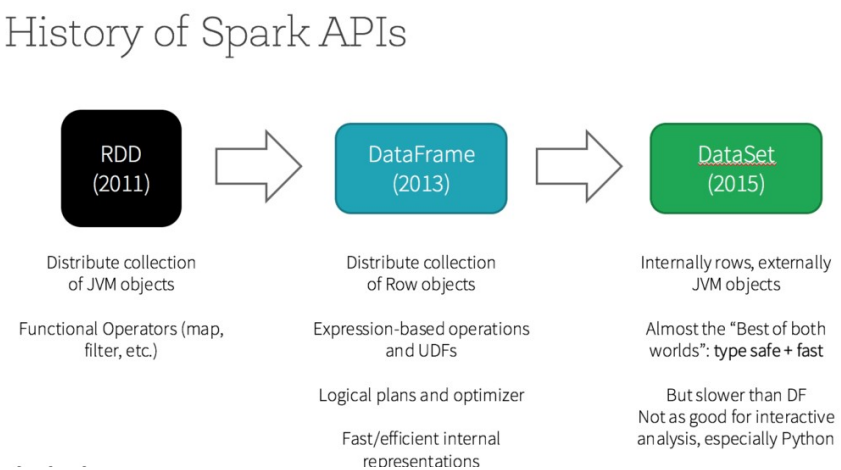


RDD

Cosa sono?

Limitazioni RDD

- **Gestione dei dati strutturati:** RDD non fornisce la visualizzazione dello schema dei dati. Non prevede la gestione dei dati strutturati.

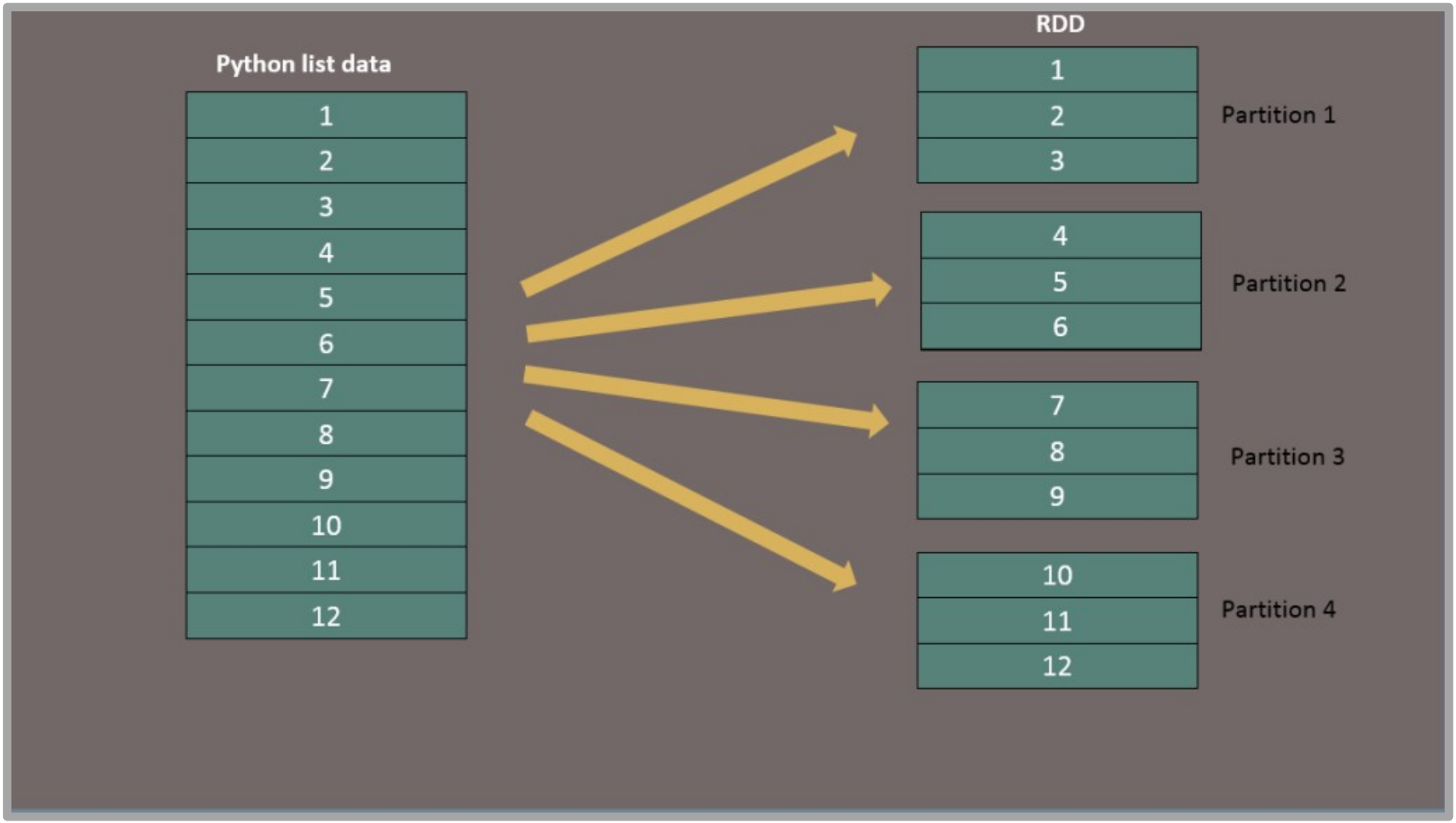


RDD

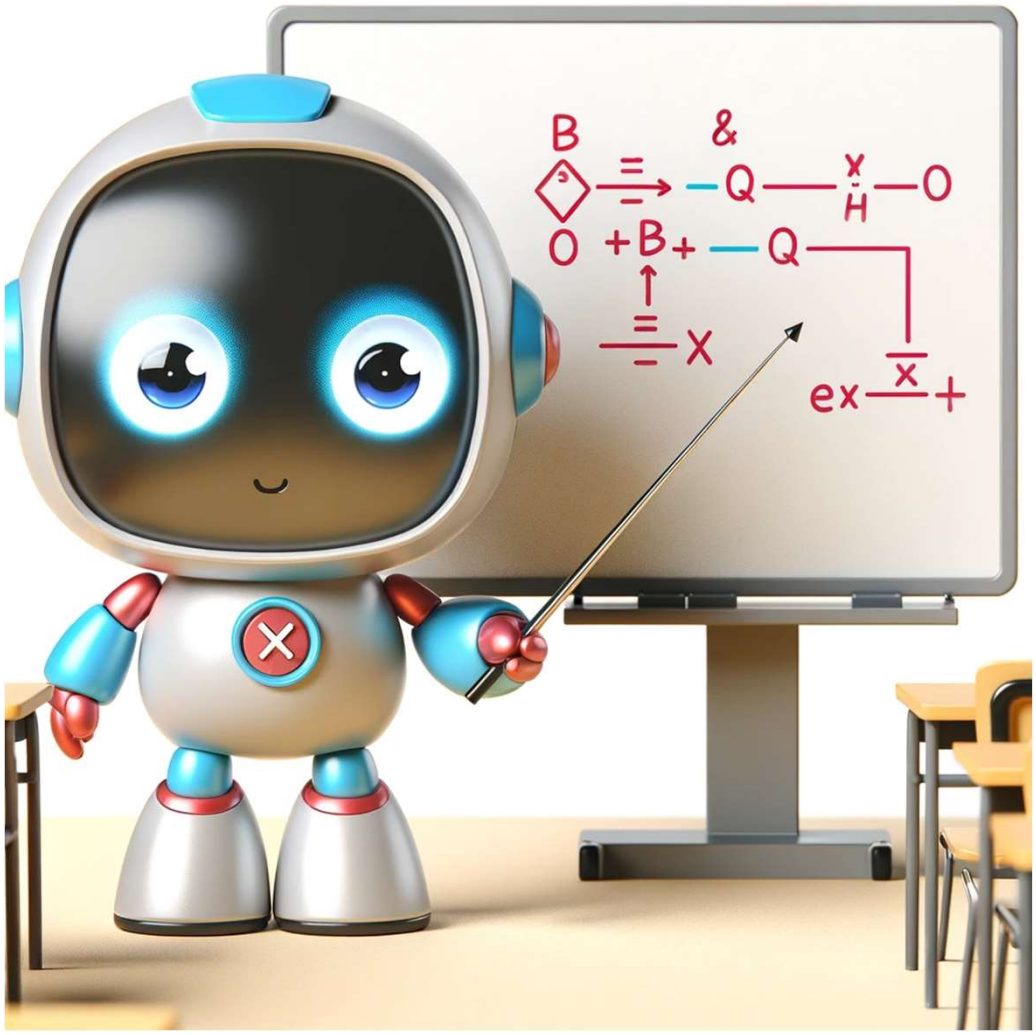
Iniziamo!

Creazione RDD

- Gli **RDD** vengono creati principalmente in due modi diversi, in primo luogo parallelizzando una raccolta esistente e in secondo luogo facendo riferimento a un set di dati in un sistema di archiviazione esterno (HDFS, S3 e molti altri).


Python list data
1
2
3
4
5
6
7
8
9
10
11
12
RDD
1
2
3
Partition 1
4
5
6
Partition 2
7
8
9
Partition 3
10
11
12
Partition 4

Esempio



RDD

Manipolazione RDD

Azioni

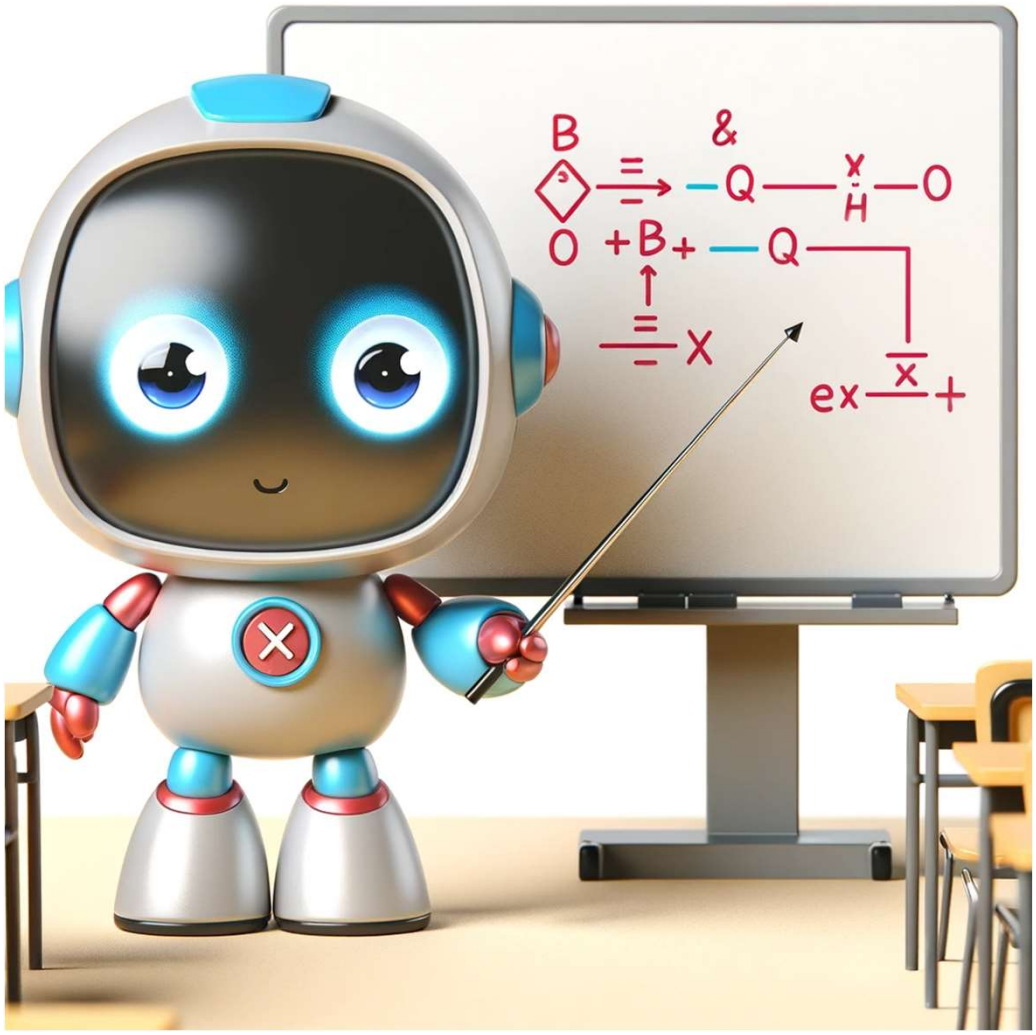
- Mentre le traformazioni dichiarano gli RDD, le azioni producono valori che tornano al programma driver di Spark o che vengono memorizzati su file. A causa della lazy evaluation, i calcoli vengono effettuati nel cluster solo quando si effettua un'azione.

- Le azioni più comuni che si utilizzano sono probabilmente **reduce**, **collect** e **take**.

- Altre azioni:

    - **First**
    - **Count**

Esempio



DataFrames

DataFrames

Introduzione DataFrames

Cosa sono?

- Un DataFrame è la struttura dati più comune nell'ambito delle API strutturate e rappresenta semplicemente una tabella di dati con righe e colonne. L'elenco delle colonne e i tipi di dati in tali colonne costituiscono lo schema.

- Il concetto di DataFrame non è esclusivo di Spark. R e Python entrambi hanno concetti simili. Tuttavia, i DataFrames in Python e R (con alcune eccezioni) esistono su una singola macchina, invece di essere distribuiti su più macchine.

- Questa limitazione vincola le operazioni che è possibile eseguire su un DataFrame in Python o R alle risorse disponibili sulla macchina specifica. Tuttavia, grazie alle interfacce linguistiche fornite da Spark per Python e R, è relativamente semplice convertire i DataFrames di Pandas (Python) in DataFrames di Spark e i DataFrames di R in DataFrames di Spark (in R).

DataFrames

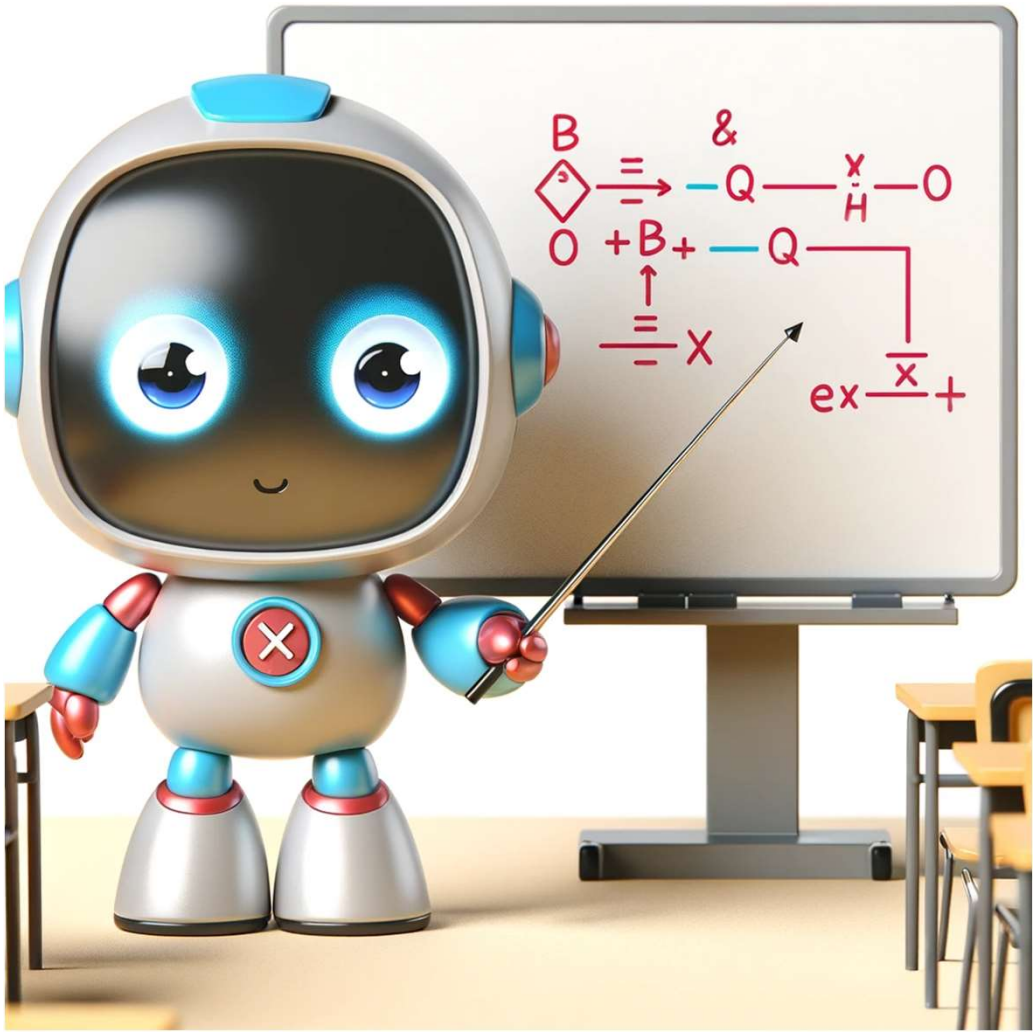
Introduzione DataFrames

Partizioni

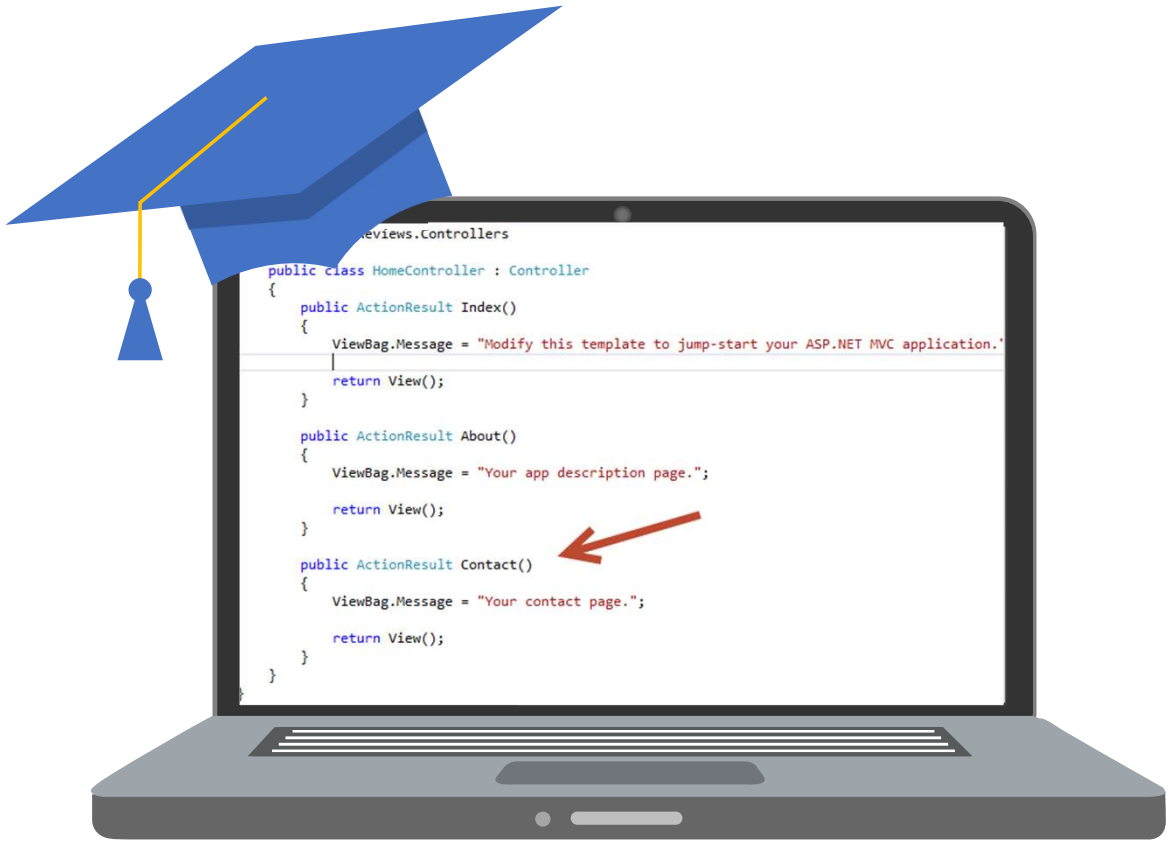
- Per consentire a ciascun esecutore di lavorare in parallelo, Spark suddivide i dati in blocchi chiamati partizioni. Una partizione è una collezione di righe che risiede su una macchina fisica nel nostro cluster. Le partizioni di un DataFrame rappresentano come i dati vengono distribuiti fisicamente nel cluster di macchine durante l'esecuzione.

- Una cosa importante da notare è che, con i DataFrames, non manipoliamo (per la maggior parte) manualmente le partizioni su base individuale. Invece, specifichiamo semplicemente trasformazioni di dati di alto livello sulle partizioni fisiche, e Spark determina come effettivamente eseguire questo lavoro sul cluster.

Esempio



Esercizio



Esercizio Dataframes

Testo:

“Facendo riferimento al file DF_Esercizio No_Sol eseguire tutti i punti”

NB: Usate la Documentazione di Spark per la sezione Dataframes: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.html

Console Applications

Exception