Apache Spark

Low-level APIs

RDD (Resilient Distributed Dataset)

What it is?

- There are two sets of low-level APIs:

1. Manipulating distributed data (RDDs)
2. Distributing and manipulating distributed shared variables (broadcast variables and accumulators).

When to Use the Low-Level APIs?

- You should generally use the lower-level APIs in three situations:

1. You need some functionality that you cannot find in the higher-level APIs; for example, if you need very tight control over physical data placement across the cluster.

2. You need to maintain some legacy codebase written using RDDs.

3. You need to do some custom shared variable manipulation.

When to Use the Low-Level APIs?

- **Note:**

It is helpful to *understand* these tools because all Spark workloads compile down to these fundamental primitives. When you're calling a DataFrame transformation, it actually just becomes a set of RDD transformations. This understanding can make your task easier as you begin debugging more and more complex workloads.

How to Use the Low-Level APIs?

- A **SparkContext** is the entry point for low-level API functionality.
- You access it through the SparkSession, which is the tool you use to perform computation across a Spark cluster.

About RDDs

- RDDs were the primary API in the Spark 1.X series and are still available in 2.X, and above but they are not as commonly used.

- In short, an RDD represents an immutable, partitioned collection of records that can be operated on in parallel.

- Unlike DataFrames though, where each record is a structured row containing fields with a known schema, in RDDs the records are just Java, Scala, or Python objects of the programmer's choosing.

About RDDs

- RDDs give you complete control because every record in an RDD is just a Java or Python object.
- You can store anything you want in these objects, in any format you want. This gives a great power.
- **Potential issue** -
- Every manipulation and interaction between values must be defined by hand, meaning that you must “reinvent the wheel” for whatever task you are trying to carry out. Also, optimizations are going to require much more manual work, because Spark does not understand the inner structure of your records as it does with the Structured APIs.

Types of RDDs

- There are lots of subclasses of RDD.
- For the most part, these are internal representations that the DataFrame API uses to create optimized physical execution plans.
- As a user, however, you will likely only be creating two types of RDDs:
- The "generic" RDD type or a **key-value RDD** that provides additional functions, such as aggregating by key.
- For main purposes, these will be the only two types of RDDs that matter.
- Both just represent a collection of objects, but **key-value RDDs** have special operations as well as a concept of custom partitioning by key.

Defining RDD

- Let's formally define RDDs. Internally, each RDD is characterized by five main properties:

1. A list of partitions
2. A function for computing each split
3. A list of dependencies on other RDDs
4. Optionally, a Partitioner for key-value RDDs (e.g., to say that the RDD is hashpartitioned)
5. Optionally, a list of preferred locations on which to compute each split (e.g., block locations for a Hadoop Distributed File System [HDFS] file)

Creating RDDs

- spark.range(10).rdd

- spark.range(10).rdd.toDF()