✨ Member-only story

# Important Spark Topics for Data Engineer

Important Spark operations and Transformations for Data Engineers.

Vishal Barvaliya · Follow

4 min read · Jan 14



In this blog we will break down all the **Important Spark Topics** For data engineer to crack any data engineering interview and work in any spark related data engineering projects!

Image Source

## 1. Spark Core Concepts - RDD Operations:

### Transformation Operations:

1. `**map**`: Applies a function to each element of the RDD.
2. `**filter**`: Selects elements that satisfy a given condition.
3. `**flatMap**`: Similar to `map`, but each input item can be mapped to zero or more output items.
4. `**distinct**`: Removes duplicate elements from the RDD.
5. `**union**`: Combines two RDDs into one.
6. `**groupByKey**`: Groups elements by key in a key-value pair RDD.
7. `**reduceByKey**`: Combines values with the same key using a specified reduce function.
8. `**sortByKey**`: Sorts elements based on keys in a key-value pair RDD.
9. `**mapPartitions**`: Applies a function to each partition of the RDD.
10. `**coalesce**`: Reduces the number of partitions in the RDD.

### Action Operations:

1. `**count**`: Returns the number of elements in the RDD.
2. `**collect**`: Retrieves all elements of the RDD as an array.
3. `**reduce**`: Aggregates the elements of the RDD using a specified function.
4. `**first**`: Retrieves the first element of the RDD.
5. `**take**`: Retrieves the first N elements of the RDD.
6. `**foreach**`: Applies a function to each element of the RDD.

7. `**countByValue**`: Counts the occurrences of each unique element in the RDD.
8. `**saveAsTextFile**`: Saves the RDD as a text file.
9. `**saveAsSequenceFile**`: Saves the RDD as a Hadoop SequenceFile.
10. `**foreachPartition**`: Applies a function to each partition of the RDD.

. . .

## 2. DataFrames and Spark SQL Operations:

#### DataFrame Transformation Operations:

1. `**select**`: Selects specific columns from a DataFrame.
2. `**filter**`: Filters rows based on a specified condition.
3. `**groupBy**`: Groups the DataFrame by specified columns.
4. `**join**`: Joins two DataFrames based on a common column.
5. `**orderBy**`: Sorts the DataFrame based on one or more columns.
6. `**distinct**`: Removes duplicate rows from the DataFrame.
7. `**withColumn**`: Adds a new column or replaces an existing one.
8. `**drop**`: Drops specified columns from the DataFrame.
9. `**agg**`: Performs aggregation operations like sum, count, etc.
10. `**na**` (handling missing data): Provides methods for handling missing data, such as `drop` and `fill`.
11. `**cache**`: Caches the DataFrame in memory for faster access.

#### DataFrame Action Operations:

1. `**show**`: Displays the first N rows of the DataFrame.
2. `**count**`: Returns the number of rows in the DataFrame.
3. `**collect**`: Retrieves all rows of the DataFrame as an array.
4. `**head**`: Returns the first N rows as a list.
5. `**first**`: Returns the first row of the DataFrame.
6. `**foreach**`: Applies a function to each row of the DataFrame.
7. `**write**`: Writes the DataFrame to an external storage system.
8. `**writeStream**`: Writes the DataFrame as a streaming job.
9. `**unpersist**`: Removes the DataFrame from the cache.

. . .

## 3. Spark Streaming Operations:

#### Streaming Operations:

1. `**updateStateByKey**`: Updates the state of the streaming application based on key-value pairs.
2. `**window**`: Performs operations over a sliding window of data.
3. `**reduceByKeyAndWindow**`: Combines values with the same key within a specified window using a reduce function.
4. `**countByWindow**`: Counts the number of elements in each window.
5. `**foreachRDD**`: Applies a function to each RDD in the DStream.
6. `**transform**`: Applies a transformation on the DStream.
7. `**union**`: Combines two DStreams.

8. `**dstream**` transformations (e.g., `**map**`, `**filter**`): Applies transformations on each batch of the DStream.

. . .

## 4. Structured Streaming Operations:

#### Structured Streaming Operations:

1. `**writeStream**`: Writes the output of a streaming query to an external storage system.
2. `**outputMode**`: Specifies the output mode of the streaming query (e.g., `append`, `complete`, `update`).
3. `**trigger**`: Defines the trigger interval for the streaming query.
4. `**format**`: Specifies the output format for the streaming query (e.g., `parquet`, `json`).
5. `**option**`: Sets configuration options for the output format.
6. **start**: Initiates the execution of the streaming query.
7. **awaitTermination**: Waits for the termination of the streaming query.
8. **isActive**: Checks if the streaming query is actively processing data.
9. **recentProgress**: Retrieves information about recent progress of the streaming query.
10. **stop**: Stops the streaming query.

. . .

## 5. Spark Performance Optimization Operations:

#### Optimization Techniques:

1. **Proper partitioning strategies:** Ensures data is distributed optimally across partitions for parallel processing.
2. **Efficient caching and persistence:** Stores intermediate results in memory to avoid recomputation.
3. **Effective use of broadcast variables:** Distributes read-only variables efficiently to worker nodes.
4. **Accumulators for distributed computations:** Performs efficient aggregations across distributed data.

. . .

## 6. Cluster Management Operations:

#### Integration with Cluster Managers:

1. **Apache Hadoop YARN integration:** Utilizes YARN for resource management and job scheduling.
2. **Apache Mesos integration:** Integrates with Mesos, an open-source cluster manager, for resource sharing.
3. **Configuring cluster resources for Spark applications:** Adjusts resource allocation for optimal Spark application performance.

. . .

## 7. Integration with Big Data Ecosystem Operations:

#### Hadoop and Hive Integration:

1. **Reading and writing data to/from HDFS:** Interacts with the Hadoop Distributed File System for data storage and retrieval.

2. **Interacting with Hive tables in Spark:** Accesses and manipulates data stored in Hive tables using Spark.

#### Kafka Integration:

1. **Reading and writing data to/from Kafka:** Connects Spark applications to Apache Kafka for real-time data streaming and processing.

. . .

## 8. Deployment and Scaling Operations:

#### Deployment Strategies:

1. **Local deployment:** Runs Spark applications on a single machine for development and testing.
2. **Cluster deployment:** Deploys Spark applications on a cluster of machines for distributed processing.
3. **Dynamic resource allocation configuration:** Adapts resource allocation based on application needs, which improves efficiency.

. . .

I hope you like this blog, if do then consider claps and follow for more such Blogs on data engineering.

*Happy Reading!!!*

> **Best of luck with your journey!!!**
>
> **Follow for more such content on Data Engineering and Data Science.**

**Resources used to write this blog:**