**Spark**

Spark is an open-source, distributed computing system that is designed for big data processing and analytics.

Spark is known for its speed. It performs in-memory computations, which makes it much faster than traditional disk-based systems like Hadoop MapReduce.

It offers APIs in multiple programming languages like Scala, Java, Python, and R, making it accessible to a broader audience. This flexibility allows users to write applications in their preferred language.

Spark provides various libraries and tools that extend its functionality. For instance, Spark SQL for working with structured data using SQL queries, Spark Streaming for real-time processing, MLlib for machine learning tasks, and GraphX for graph processing.

It maintains fault tolerance through its resilient distributed dataset (RDD) abstraction. It tracks the lineage of operations, enabling the reconstruction of lost data partitions due to failures.

Spark can easily handle large amounts of data by distributing computations across a cluster of machines.

**Components of Spark**

**Spark Core:** It provides the basic functionality of Spark, including task scheduling, memory management, fault recovery, and interacting with storage systems. Core also implements the resilient distributed dataset (RDD), the fundamental data structure in Spark.

**Spark SQL:** This module allows SQL-like querying capabilities on structured data within Spark. It enables users to run SQL queries alongside Spark programs, making it easier to integrate SQL queries with complex analytics.

**Spark Streaming:** It enables real-time processing of streaming data. Spark Streaming ingests live data streams and divides them into micro-batches, which are then processed using Spark's computational engine.

**MLlib (Machine Learning Library):** MLlib provides a rich set of machine learning algorithms and utilities for tasks such as classification, regression, clustering, collaborative filtering, and more. It's designed to be scalable and work efficiently with large-scale datasets.

**GraphX:** This component is used for graph processing and analytics. GraphX provides an API for expressing graph computation and includes algorithms for graph analytics and processing.

**Spark Architecture**

The Apache Spark framework uses a master-slave architecture that consists of a driver, which runs as a master node, and many executors that run across as worker nodes in the cluster.

**The Driver Node:** It houses the user-defined main program, SparkContext, and coordinates job scheduling, task distribution, monitoring, and resource management. Responsible for breaking down the application into jobs and stages, it orchestrates task execution across worker nodes, ensuring successful execution and handling failures,

**The Worker Node:** They are the computational units within a Spark cluster responsible for executing tasks and storing data. These nodes host the Executors, which perform computations for the Spark application. Worker nodes contain multiple Executors, and each Executor manages a certain amount of memory and CPU cores allocated by the cluster manager. They handle the execution of individual tasks, process data in memory or on disk, and communicate with the Driver Node. Worker nodes play a crucial role in parallel processing, as they distribute and execute tasks across the cluster.

**The Cluster Manager:** The cluster manager is responsible for resource allocation and management within a Spark cluster. It acts as an intermediary between the Spark application and the underlying cluster infrastructure, overseeing the allocation of resources (such as CPU, memory, and storage) across worker nodes. They monitor node availability, handle failures, and ensure optimal resource utilization, playing an important role in the deployment and execution of Spark applications across the cluster infrastructure.

**SparkContext** is the heart of spark application. It manages the execution of applications, including dividing the task into stages, optimizing execution plans and interacting with cluster managers.

**RDD**s are the core abstraction in spark. They are an immutable distributed collection of elements of data, partitioned across nodes in clusters that can be operated in parallel with a low-level API that offers transformations and actions.

**Transformation**

Transformations are operations that create a new dataset (or RDD) by applying a function to each element of the original dataset in a distributed manner. Transformations are lazy, they don't compute results immediately; instead, they build a lineage of transformations that describe how to compute the final dataset.

**Narrow Transformation** are operations on distributed dataset that are applied within a single partition without the need of shuffling or data exchange between partitions. These transformations operate independently on individual partitions of the dataset and do not require data movement across clusters.

**Wide Transformations** are operations on distributed datasets that involve data shuffling or exchange between partitions. These transformations depend on data from multiple partitions requiring data movement and coordination across the clusters

**Action**

Actions are operations that initiate the execution of the Directed Acyclic Graph (DAG) of transformations and produce a result by triggering the actual computation on the distributed datasets. They are the point at which the execution of transformations are triggered. When action is called, spark evaluated the series of transformations defined on the dataset and comptes the final result

**Directed Acyclic Graph (DAG)**

DAG is the logical execution plan of a spark to represent and optimize the flow of operations in a data processing job on a distributed dataset. When a sequence of transformations is applied to a dataset, spark creates a DAG representing these transformations. Each transformation creates a new RDD or dataset and the resulting dataset is linked to its parent dataset in the DAG. It captures the dependencies between different stages of computations. It is directed because it shows the flow of data and operations from the initial dataset through the various transformations to the final output. It is acyclic because it does not contain loops