**SPARK**

Big data refers to extremely large and complex datasets that traditional data processing applications struggle to handle efficiently. It encompasses the volume, variety, and velocity of data, challenging the capabilities of conventional data management tools.

1. Volume
2. Velocity
3. variety

**Hadoop**

1. **MapReduce Model:**

Hadoop primarily uses the MapReduce programming model for processing large data sets in a distributed computing environment. It consists of two main components: Map phase (for processing and converting input data into key-value pairs) and Reduce phase (aggregating the results from the Map phase).

1. **HDFS (Hadoop Distributed File System)**

Hadoop includes HDFS for storing data across multiple machines in a distributed manner. It breaks large files into smaller blocks and distributes them across the cluster.

1. **Batch Processing**

It is suitable for batch processing where data is processed in sequential steps.

1. **Java-Centric**

Initially, Hadoop was predominantly Java-based, but now supports other languages through streaming or frameworks.

**Spark**

1. **In-Memory Processing**

Spark is known for its speed as it performs in-memory processing, keeping data in memory rather than persisting it to disk after each operation. It uses Resilient Distributed Datasets (RDDs) for in-memory computation.

1. **DAG (Directed Acyclic Graph) Execution**

Spark uses DAG for optimization of workflows, enabling parallel processing and efficient task execution.

1. **Multiple Language Support**

Spark provides APIs in multiple languages including Scala, Java, Python, and R, making it more accessible to developers familiar with these languages.

1. **Advanced Analytics**

Spark provides higher-level libraries such as Spark SQL, MLlib (machine learning library), GraphX (graph processing), and Spark Streaming (real-time data processing) for diverse data processing needs.

1. **Compatibility with Hadoop**

Spark can run on top of Hadoop (using YARN) and can directly access HDFS data.

**Differences b/w Hadoop & Spark**

1. Spark generally outperforms Hadoop due to its in-memory computing capabilities.
2. Hadoop primarily uses the MapReduce model, whereas Spark uses a more flexible approach with its RDDs and DAG-based execution.
3. Spark is better suited for real-time processing and iterative workloads compared to Hadoop.
4. Spark provides a more developer-friendly API compared to the lower-level abstraction of Hadoop's MapReduce.

**Kerberos Architecture**

Kerberos is a network authentication protocol designed to provide secure authentication for client-server applications by using secret-key cryptography. Its architecture involves several components working together to authenticate users and services in a network environment.

1)**User:**

The entity seeking access to services within the network.

2)**Key Distribution Center (KDC):**

* Authentication Server (AS): Handles initial authentication requests by issuing ticket-granting tickets (TGTs) upon successful authentication of users.
* Ticket-Granting Server (TGS): Provides service tickets to users with valid TGTs, allowing them access to specific services.

**3)Service Server (SS):** Hosts the services that users want to access.

**Steps**

**1)Authentication (AS Exchange):**

Step 1: The user sends a request to the AS for authentication.

Step 2: The AS verifies the user's credentials and issues a TGT encrypted with a session key known only to the user and the TGS.

2)**Ticket-Granting (TGS Exchange):**

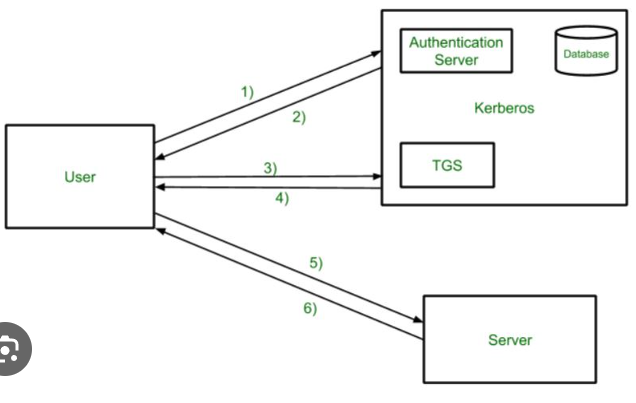
Step 3: When the user wants to access a specific service, they present the TGT to the TGS along with the requested service.

Step 4: The TGS verifies the TGT's authenticity and issues a service ticket encrypted with a session key known only to the user and the SS.

**3)Service Request:**

Step 5: The user presents the service ticket to the SS to access the service.

Step 6: The SS verifies the service ticket's authenticity using its own secret key and grants access to the requested service.



**YARN Architecture**

Apache Hadoop YARN (Yet Another Resource Negotiator) is a resource management layer in the Hadoop ecosystem designed to manage and allocate resources in a cluster. Its architecture consists of several components that work together to manage resources efficiently and execute various data processing workloads:

**1)ResourceManager (RM):**

* ResourceManager Scheduler:

Manages available cluster resources and schedules applications' containers based on resource requirements.

* ResourceManager ApplicationManager:

Manages application lifecycle, negotiates resources, and coordinates with NodeManagers.

**2)NodeManager (NM):**

* NodeManager ContainerManager:

Manages containers, which are the basic execution units on a node. It monitors resource usage and reports to the ResourceManager.

* NodeManager NodeHealthCheckerService:

Monitors the health of the node and reports back to the ResourceManager.

**3)ApplicationMaster (AM):**

* ApplicationMaster per Application:

Each application submitted to YARN has its own ApplicationMaster, responsible for negotiating resources with the ResourceManager and managing task execution within containers.

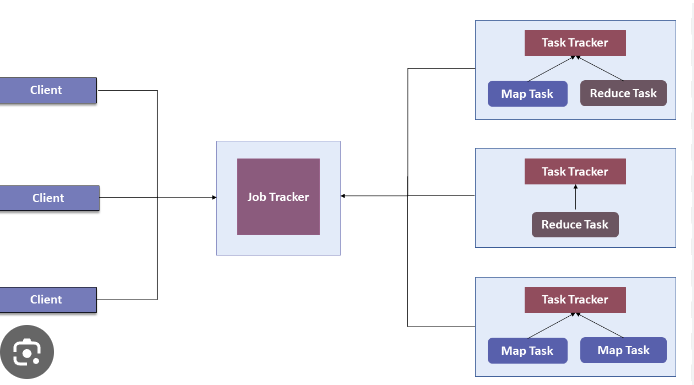
* Distributed Application Logic:

Runs application-specific logic, interacts with ResourceManager for resource allocation, and monitors task execution.

**4)Containers:**

Execution Units:

These are the units where application tasks are executed. Each container has a specific amount of memory and CPU allocated to it.

****

**API’s in Spark**

**1)RDD (Resilient Distributed Dataset)**

* They represent a collection of elements distributed across multiple nodes in a cluster, allowing for parallel processing.
* RDDs are immutable
* RDDs provide a low-level API

**2)Dataframe**

* DataFrames are distributed collections of data organized into named columns, similar to a table in a relational database.
* They provide a more structured and higher-level abstraction than RDDs.

**3)Dataset**

* Datasets maintain compile-time type safety for the data they contain, providing better error checking and code maintenance.
* Datasets offer a rich set of functional programming APIs for manipulating data, similar to RDDs, while also allowing higher-level abstractions like DataFrames.

**Transformation**

Transformations are operations that create a new dataset from an existing one. They are *lazy* operations, meaning they don't compute their results immediately but instead create a lineage of transformations that will be executed only when an action is called.

* map(func): Applies a function to each element in the dataset.
* filter(func): Selects elements based on a given condition.

Types of transformations are

1. Narrow Transformation
2. Wide Transformation

**Action**

Actions are operations that trigger the execution of the Spark job and return results to the driver or write data to external storage. Unlike transformations, actions force the evaluation of the transformations and initiate the computation.

* collect(): Retrieves all elements of the dataset to the driver program.
* count(): Counts the number of elements in the dataset.