What is Apache Spark

- Apache Spark is an open-source, distributed computing framework designed for big data processing and analytics.
- Designed to process large-scale datasets efficiently across multiple machines.
- Provides both batch and real-time (streaming) data processing.
- Runs programs up to 100x faster in memory and 10x faster on disk than traditional Hadoop MapReduce.
- It offers high-level APIs in **Scala, Java, Python (PySpark), R, and SQL**.

Apache Spark Characteristics

Apache Spark stands out as a fast, flexible, and unified data processing framework. Its key characteristics are:

- ✓ Speed
- ✓ Ease of Use
- ✓ Modularity
- ✓ Extensibility
- ✓ Fault Tolerance
- ✓ Scalability

Speed

- In-memory computation makes Spark up to 100x faster than Hadoop MapReduce (which writes intermediate results to disk).
- Uses Directed Acyclic Graph (DAG) execution engine to optimize task scheduling.
- Efficient for iterative algorithms (like ML training) where intermediate results can be reused.

Ease of Use

- APIs available in Scala, Java, Python (PySpark), and R.
- Developers can use SQL queries (Spark SQL) or high-level APIs like DataFrames and Datasets.
- Provides interactive shells (spark-shell, pyspark) for quick exploration.

Modularity

Comes with specialized libraries that integrate seamlessly:

- Spark SQL → for structured data & queries.
- Spark Streaming / Structured Streaming → for real-time data.

- MLlib → for machine learning.
- GraphX → for graph computation.

All libraries run on the **same Spark engine**, making it a unified stack.

Extensibility

- Supports various storage systems: HDFS, Hive, Cassandra, MongoDB, Amazon S3, Azure Data Lake, etc.
- Reads/writes multiple formats: Parquet, ORC, JSON, Avro, CSV.
- Integrates with cluster managers like YARN, Kubernetes, and Mesos.
- Works with data lake formats like Delta Lake, Apache Hudi, and Iceberg.

Fault Tolerance

- Uses RDD (Resilient Distributed Dataset) to recover lost partitions automatically.
- Lineage information allows recomputation of lost data.
- Structured Streaming provides exactly-once guarantees

Scalability

- Scales from a single laptop to thousands of cluster nodes.
- Handles terabytes to petabytes of data.
- Dynamic resource allocation allows flexible cluster usage.

Apache Spark Components as a Unified Stack

Apache Spark provides a unified framework for handling different types of big data workloads (batch, streaming, ML, and graphs) under one system. This avoids the need for multiple tools.

Here are the main components:

Spark Core (Foundation)

- The base engine of Apache Spark.
- Provides essential functionalities like:
  - Memory management
  - Task scheduling
  - Fault tolerance (RDDs & lineage)
  - Interacting with storage systems (HDFS, S3, Cassandra, HBase, etc.)
- Defines the **RDD (Resilient Distributed Dataset)** abstraction.
  - Immutable distributed collection of data.

    - Provides transformations (map, filter, flatMap) and actions (count, collect).
    - Fault-tolerance via lineage tracking (recomputes lost partitions).
- All other Spark libraries are built on top of Spark Core.

Spark SQL (Structured Data Processing)

- Its Provides DataFrames (table-like structure with named columns).
- Provides Datasets (type-safe, available in Scala/Java).
- Includes Catalyst Optimizer for intelligent query optimization.
- Uses Tungsten Execution Engine for efficient memory usage and code generation.
- Supports SQL queries, HiveQL, and integration with BI tools.
- Reads/writes multiple formats: JSON, Parquet, ORC, Avro, JDBC.

Ex:

```
df = spark.read.json("data.json")

df.createOrReplaceTempView("people")

result = spark.sql("SELECT name, age FROM people WHERE age > 30")

result.show()
```

MLlib (Machine Learning Library)

- Spark's scalable machine learning library for big data.
- Provides distributed ML algorithms:
    - Classification (Logistic Regression, Decision Trees).
    - Regression (Linear Regression).
    - Clustering (KMeans, Gaussian Mixture).
    - Collaborative Filtering (ALS for recommendations).
    - Dimensionality Reduction (PCA, SVD).
- Includes utilities for **feature extraction, transformation, selection, and pipeline building**.
- Integrates with Spark SQL and DataFrames.

Example: Building a Logistic Regression model.

```
from pyspark.ml.classification import LogisticRegression
```

```
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")
lr = LogisticRegression(maxIter=10, regParam=0.01)
model = lr.fit(training)
```

GraphX (Graph Processing)

- A library for graph computation and analysis.
- Built on top of RDDs.
- Provides an API for graph-parallel computation.
- Allows transformation of graphs (e.g., subgraph extraction, join operations).
- Includes graph algorithms:
  - PageRank (used by Google Search).
  - Connected Components.
  - Triangle Counting.
  - Shortest Paths.

Why It's Called a Unified Stack

- All components share the same Spark Core engine → no need to switch tools for different workloads.
- Example end-to-end pipeline:
  - Use Structured Streaming to capture live user activity.
  - Store data in Parquet/Delta Lake.
  - Query with Spark SQL.
  - Train ML models with MLlib.
  - Run GraphX for network analysis.
- This makes Spark an all-in-one platform for batch, streaming, machine learning, and graph analytics.

Apache Spark Architecture

Apache Spark follows a master–slave architecture with a driver program (master) and cluster workers (slaves).
It uses a distributed execution engine to process large datasets across multiple nodes in a cluster.

Components of Spark Architecture

Driver Program (Master Node)

- The **entry point** of a Spark application.

- Runs the **main() function** of your program.
- Responsible for:
    - Maintaining **SparkContext** (connection to cluster).
    - Converting user code into a **Directed Acyclic Graph (DAG)**.
    - Splitting the DAG into **stages** and **tasks**.
    - Scheduling tasks on executors.
    - Collecting results back from executors.

Cluster Manager

- Manages resources (CPU, memory) across the cluster.
- Spark supports multiple cluster managers:
    - **Standalone** (built-in Spark manager)
    - **YARN** (Hadoop cluster manager)
    - **Apache Mesos**
    - **Kubernetes**

Worker Nodes

- The machines in the cluster that **run tasks assigned by the driver**.
- Each worker node hosts:

a) Executor

- A process launched for each Spark application.
- Runs tasks and stores data in memory/disk.
- Executors remain alive throughout the application (unlike MapReduce).

b) Task

- The **smallest unit of execution** in Spark.
- One stage is divided into many tasks, distributed across executors.

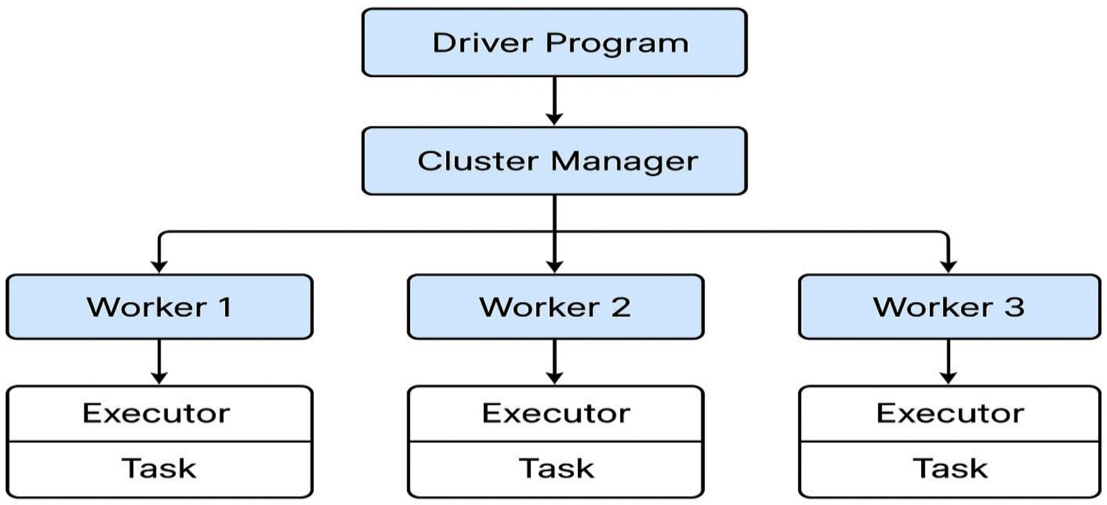
RDD (Resilient Distributed Dataset)

- The fundamental abstraction in Spark.

- Immutable distributed collection of objects, partitioned across nodes.
- Provides **fault tolerance** using **lineage information** (recomputes lost partitions).

How Spark Executes a Job (Workflow)

1. **User Program**: You write Spark code (Scala, Python, Java, R, SQL).
2. **Driver Program**:
    - Initializes SparkContext.
    - Converts user code into a **DAG (Directed Acyclic Graph)** of stages.
3. **DAG Scheduler**:
    - Divides DAG into **stages**.
    - Each stage is split into **tasks** (based on data partitions).
4. **Task Scheduler**:
    - Assigns tasks to available executors on worker nodes.
5. **Cluster Manager**:
    - Provides resources (CPU, memory).
6. **Executors (on Workers)**:
    - Execute the tasks in parallel.
    - Store intermediate results in memory/disk.
7. **Driver**: Collects and combines the results.



Spark Basic Data Types

Spark SQL and DataFrames support the following data types

Numeric types

- **Byte Type:** Represents 1-byte signed integer numbers. The range of numbers is from -128 to 127.
- **Short Type:** Represents 2-byte signed integer numbers. The range of numbers is from -32768 to 32767.
- **Integer Type:** Represents 4-byte signed integer numbers. The range of numbers is from -2147483648 to 2147483647.
- **Long Type:** Represents 8-byte signed integer numbers. The range of numbers is from -9223372036854775808 to 9223372036854775807.
- **Float Type:** Represents 4-byte single-precision floating point numbers.
- **Double Type:** Represents 8-byte double-precision floating point numbers.
- **Decimal Type:** Represents arbitrary-precision signed decimal numbers.

String type

- **String Type:** Represents character string values.
- **Varchar Type(length):** A variant of StringType which has a length limitation. Data writing will fail if the input string exceeds the length limitation. Note: this type can only be used in table schema, not functions/operators.
- **Char Type(length):** A variant of VarcharType(length) which is fixed length. Reading column of type CharType(n) always returns string values of length n. Char type column comparison will pad the short one to the longer length.

Binary Type: Represents byte sequence values.

Boolean Type: Represents boolean values.

Datetime type

Date Type: Represents values comprising values of fields year, month and day, without a time-zone.

Timestamp Type: Timestamp with local time zone(TIMESTAMP_LTZ). It represents values comprising values of fields year, month, day, hour, minute, and second, with the session local time-zone. The timestamp value represents an absolute point in time.

Example:

```
from pyspark.sql.types import *
# Define a schema using Spark data types
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("salary", DoubleType(), True),
    StructField("skills", ArrayType(StringType()), True),
    StructField("properties", MapType(StringType(), StringType()), True),
    StructField("join_date", DateType(), True)
])
```

Spark Core and SQL

Spark Core:

- All the functionalities being provided by Apache Spark are built on the highest of the Spark Core.
- It delivers speed by providing in-memory computation capability. Spark Core is the foundation of parallel and distributed processing of giant dataset.

- It is the main backbone of the essential I/O functionalities and significant in programming and observing the role of the spark cluster.
- It holds all the components related to scheduling, distributing and monitoring jobs on a cluster, Task dispatching, Fault recovery.
- The functionalities of this component are:
    - It contains the basic functionality of spark. (Task scheduling, memory management, fault recovery, interacting with storage systems).
    - Home to API that defines RDDs.

Spark SQL Structured data:

- The Spark SQL component is built above the spark core and used to provide the structured processing on the data.
- It provides standard access to a range of data sources. It includes Hive, JSON, and JDBC.
- It supports querying data either via SQL or via the hive language. This also works to access structured and semi-structured information.
- It also provides powerful, interactive, analytical application across both streaming and historical data.
- Spark SQL could be a new module in the spark that integrates the relative process with the spark with programming API.
- The main functionality of this module is:
    - It is a Spark package for working with structured data.
    - It Supports many sources of data including hive tablets, parquet, json.
    - It allows the developers to intermix SQK with programmatic data manipulation supported by RDDs in python, scala and java.