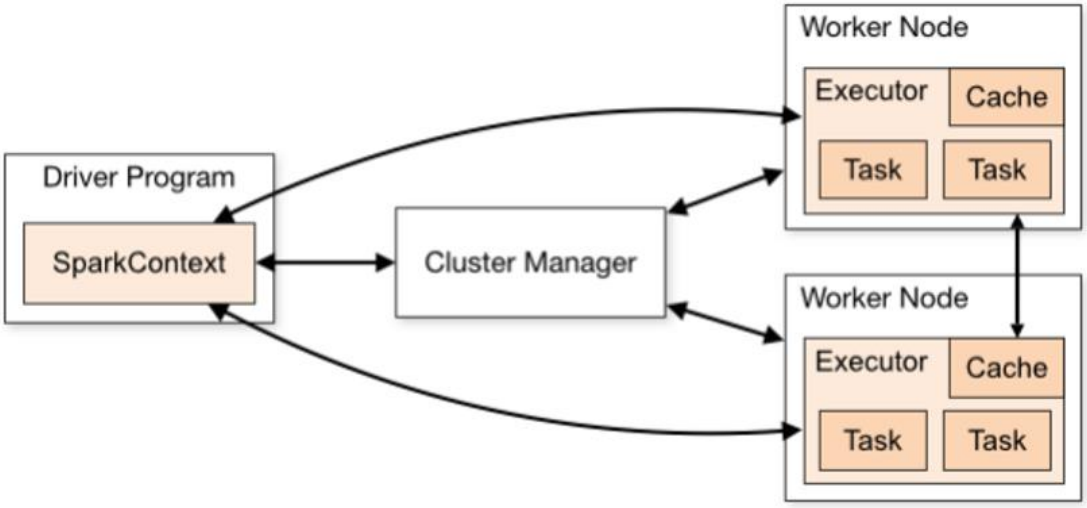
SPARK ARCHITECTURE

Apache Spark is a distributed computing system designed for fast and flexible large-scale parallel data processing. It has a master-slave architecture.



To understand the architecture of Apache Spark, we need to understand the following components:

1. **Driver Program -** It is the central coordinator that manages the execution of a Spark application.
    - **Initiates SparkContext/SparkSession:** Driver program is responsible for starting the SparkContext/SparkSession which acts as the entry point to the Spark Application.
    - It runs on the **Master Node**.
    - **Plans and schedules tasks:** It transforms the user code into tasks, creates an execution plan and distributes tasks across the worker nodes.
    - **Creates Execution Plan:** It creates the execution plan, and it is responsible for distributing tasks across the worker nodes.
    - **Collects and reports metrics**: It gathers information about the progress and health of the application.

2. **SparkContext -** This is the main entry point for Spark functionality. It will connect the Spark application with the cluster where it will be executed.
    - **Entry Point:** SparkContext is the original entry point for Spark functionality.
    - **RDDs Focused:** It is mainly used to create and manipulate RDDs (Resilient Distributed Datasets).
    - **Independent Object:** It operates independently and requires separate contexts for SQL, streaming, etc.

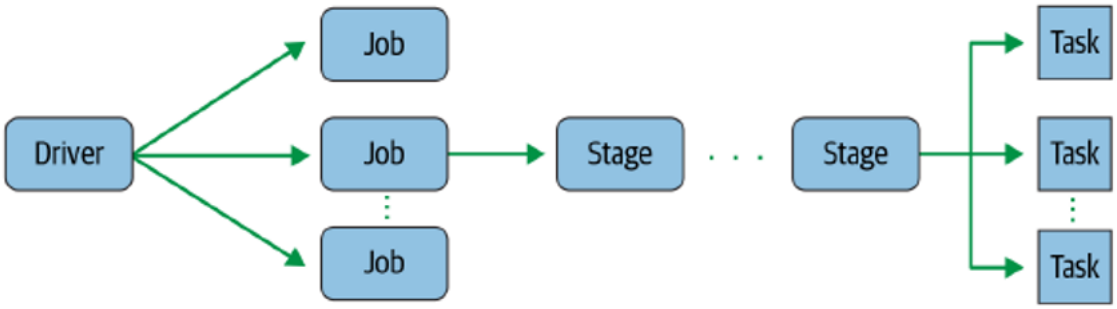
3. **SparkSession -** Introduced in Spark 2.0, it is a unified entry point for reading data, processing data, and interacting with different Spark features.
    - **Unified Entry Point:** SparkSession is a unified entry point for all the Spark functionalities, including working with RDDs, DataFrames, and Datasets.
    - **DataFrames and Datasets Focused:** Provides APIs to create and manipulate DataFrames and Datasets, which are high-level abstractions over RDDs (Resilient Distributed Datasets). Supports a wide range of operations on DataFrames and Datasets, including transformations (like select, filter, groupBy) and actions (like collect, show, count).
    - **Encapsulates SparkContext:** Internally, it encapsulates a SparkContext, SQLContext, and HiveContext, providing a simpler and more intuitive interface.
    - **Additional Functionalities:** It includes methods for reading data from various sources (like Hive, HDFS, JDBC, etc.), configuration management, and catalog operations (metadata management).

4. **Cluster Manager -** The Cluster Manager is responsible for resource management and job scheduling. It allocates resources across the cluster and manages the execution of tasks. Spark supports different cluster managers:
    - **Standalone**: A simple cluster manager included with Spark.
    - **Apache Mesos**: A general cluster manager that can run Hadoop and other applications.
    - **Hadoop YARN(Yet Another Resource Negotiator)**: The resource manager for Hadoop.
    - **Kubernetes**: An open-source system for automating deployment, scaling, and managing containerized applications.

5. **Executors -** Executors are the worker nodes that run individual tasks within a Spark job. They perform the following roles:
    - **Execute code assigned by the driver**: Executors run the tasks that the driver program assigns to them.
    - **Store data**: Executors store data for the duration of the job, which can be used in future tasks.
    - **Report status and results**: Executors send status updates and results back to the driver.

6. **Transformations -** Transformations are operations that create a new dataset from an existing one. They are lazy, meaning they do not immediately execute when called. Instead, they define a logical execution plan (DAG) that Spark builds up internally. Transformations are only executed when an action is called, triggering the computation.
    - **Lazy Evaluation:** Transformations are not executed immediately. Spark records the transformation and builds up a logical execution plan.
    - **New Dataset:** Each transformation creates a new dataset from the existing one.
    - **Types:** There are two types of transformations
        i. **Narrow Transformations:** The data required to compute the records in a single partition resides in a single partition of the parent RDD (e.g., map, filter).
        ii. **Wide Transformations:** The data required to compute the records in a single partition may reside in multiple partitions of the parent RDD, causing a shuffle (e.g., reduceByKey, groupByKey)

7. **Action -** Actions are operations that trigger the execution of the transformations to produce a result. They cause the execution of the previously defined transformations and return a value or write data to an external storage system. Example – collect(), count(), take()
    - **Trigger Execution:** Actions force the execution of the DAG built up by transformations.
    - **Result Generation:** Actions return a result to the driver program or write data to external storage.
    - **Execution Plan:** When an action is called, Spark's DAG scheduler submits a job, which is divided into stages and tasks for execution.



8. **Job -** A job in Spark is the highest-level unit of computation. It is triggered by an action (e.g., count, collect, saveAsTextFile) on an RDD, DataFrame, or Dataset. A job represents a complete computation, which can consist of multiple stages and tasks.
    - **Triggering:** Jobs are triggered by actions. Transformations (like map, filter, reduceByKey) are lazy and only define the computation DAG (Directed Acyclic Graph), but do not execute it until an action is called.
    - **Execution:** Each job consists of multiple stages, which are further divided into tasks.
    - **Lifecycle:** When an action is invoked, Spark creates a logical execution plan, which is optimized and transformed into a physical execution plan. This plan is then divided into stages and tasks, and the job is submitted for execution.

9. **Stages** - A **stage** in Spark is a set of tasks that can be executed in parallel. Stages are determined based on the shuffle boundaries. Each stage corresponds to a group of tasks that have the same set of dependencies and can be executed without requiring a shuffle of the data.
    - **Division of Jobs:** Jobs are divided into stages by Spark based on shuffle operations. Each shuffle operation results in a new stage.
    - **Dependencies:** Stages are created based on narrow and wide dependencies. Narrow dependencies (like map, filter) allow data to be pipelined, while wide dependencies (like groupByKey, reduceByKey) require shuffling of data, creating a boundary between stages.
    - **Execution Order:** Stages are executed sequentially. A stage will not start until its parent stages have completed.
    - **Task Granularity:** Each stage consists of multiple tasks, where each task processes a partition of the data.

10. **Tasks** - Tasks are the smallest units of execution in Spark. Each stage is divided into multiple tasks, with each task processing a partition of the data. Tasks are distributed across the cluster and executed in parallel on different nodes.

11. **Shuffle** - Shuffle refers to the process of redistributing data across the cluster, which is often necessary when performing wide dependency transformations such as groupByKey, reduceByKey, and join. Shuffling involves:

    - **Repartitioning data**: Data is reorganized across different nodes.
    - **Sorting**: Data within each partition is sorted.
    - **Data transfer**: Intermediate data is transferred between executors to achieve the required data distribution.

12. **Result** – The final output of the Spark job.