What is Apache Spark?

- Apache Spark, an open-source cluster computing framework, is developed in 2009 at UC Berkeley AMP Lab.

- Apache Spark is a unified analytics engine for large-scale distributed data processing and machine learning.

- Spark provides an unified engine with a stack of libraries that allow for complex analytics, including batch, streaming data.

- Spark is one of the largest OSS(Open Source Software) communities in big data analytics, and its applications range from business, finance, healthcare, and other scientific computing.

Why Apache Spark?

Here are four main reasons from the official Apache Spark™ website that should convince you to use Spark:

- **Speed**
  - Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
  - Utilizes an advanced DAG (Directed Acyclic Graph) execution engine supporting acyclic data flow and in-memory computing.
- **Ease of Use**
  - Write applications quickly in Java, Scala, Python, or R.
  - Spark offers over 80 high-level operators that make it easy to build parallel apps.
  - You can use it interactively from the Scala, Python, and R shells.
- **Generality**
  - Combine SQL, streaming, and complex analytics.
  - Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.
  - Seamlessly combine these libraries in the same application.
- **Runs Everywhere**
  - Spark runs on Hadoop, Mesos, standalone, or in the cloud.
  - It can access diverse data sources including HDFS, Cassandra, Mysql, and S3.
  - Spark can run on a single machine or in a cluster of computers.

Speed
Ease of Use
APACHE Spark
Generality
Runs Everywhere

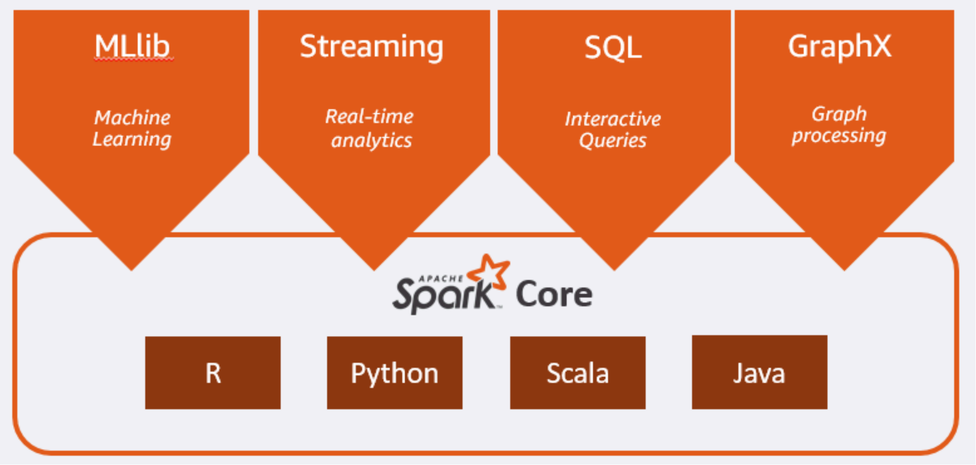
Powerful Use Cases of Apache Spark in Action

Apache Spark is a powerful big data processing framework that excels in various scenarios. Here are some compelling examples of how industry leaders leverage Spark's capabilities:

- **Recommendation Engines:**
  - **Netflix:** Spark personalizes recommendations for millions of users by analyzing vast amounts of viewing data.
  - **Pinterest:** Spark analyzes user behavior in real-time, suggesting relevant content as users explore the platform.
- **Real-time Analytics:**
  - **Uber:** processes massive datasets from user trips, enabling efficient data transformation and analysis for informed decision-making.
  - **Twitter:** Spark analyzes real-time data streams, allowing for sentiment analysis, trend identification, and personalized recommendations.
  - **eBay:** Spark empowers real-time analytics for fraud detection and personalized recommendations, enhancing user experience and driving sales.
- **Large-Scale Data Processing:**
  - **Alibaba:** Spark tackles petabytes of data generated across its e-commerce platforms, enabling large-scale data analysis tasks.
  - **Broad Institute (MIT & Harvard):** Spark streamlines genetic data analysis, facilitating crucial scientific advancements.
- **...** source,source

The Apache Spark Ecosystem

Spark consists of several libraries that can be used together in the same application:



- **Spark Core** is a fundamental component of the platform, serving various crucial functions including:
  - **Memory Management:** Efficiently allocates memory across the cluster for data and computations.
  - **Fault Recovery:** Automatically recovers from failures by restarting tasks on different nodes.
  - **Task Scheduling:** Optimizes task execution across the cluster for best performance.
  - **Distributed Processing:** Distributes data and computations across the cluster for parallel processing.
  - **Job Monitoring:** Monitors the progress and health of running jobs.
  - **Storage System Interaction:** Reads and writes data from various storage systems like HDFS, S3, etc.

- **Spark MLlib** used to enable machine learning in a distributed manner with its algorithms and utilities.
- **Spark Streaming** Used to stream live data into analytics applications.
- **Spark SQL** Used to run SQL queries on your data, both static and streaming data. It supports multiple data sources and data types(DataFrames, Datasets).
- **Spark GraphX** Graph computation framework which includes APIs for creating and manipulating graphs.

Apache Spark Architecture Overview

Two Main Abstractions of Apache Spark : **RDD** & **DAG**

Resilient Distributed Datasets (RDDs):

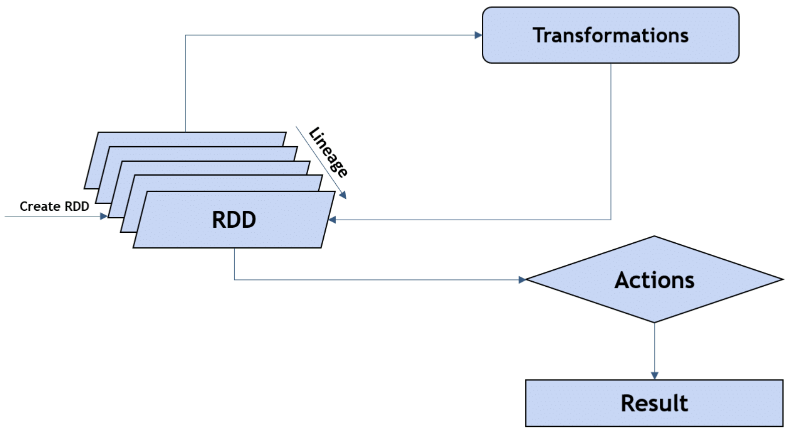
- **Resilient:** RDDs automatically recover from failures.
- **Distributed:** RDDs are fragmented and distributed across a cluster for parallel processing.
- **Dataset:** RDDs represent collections of elements.

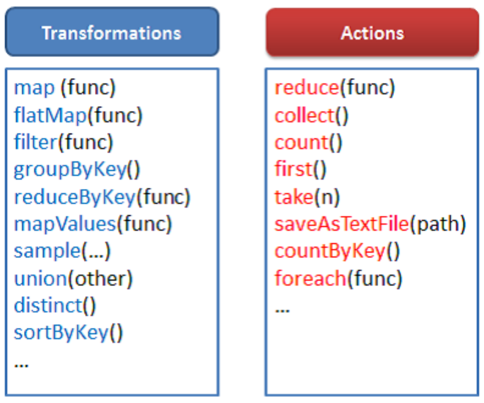
Key Characteristics of RDDs:

- **Fault Tolerance:** RDDs are fault-tolerant, meaning they can recover from failures. In case of a node failure, Spark can rebuild the lost data using lineage tracking, which records the steps used to create the RDD.
- **Immutability:** RDDs are immutable, meaning they cannot be modified after creation. This immutability simplifies error handling and optimizations.
- **Data Partitions**: Each RDD is split into partitions, which are smaller chunks of data. Partitions can be processed independently on different nodes in the cluster, further boosting parallelism.
- **Distributed Data:** RDDs are distributed across multiple nodes in a cluster. This allows Spark to process large datasets efficiently by parallelizing operations across nodes.

RDDs support two primary operations:

- Transformations :
  - Operations like map, filter, join.
  - Lazy operations to build RDDs from others.
- Actions :
  - Operations like count, collect, save.
  - Return results or write to storage.



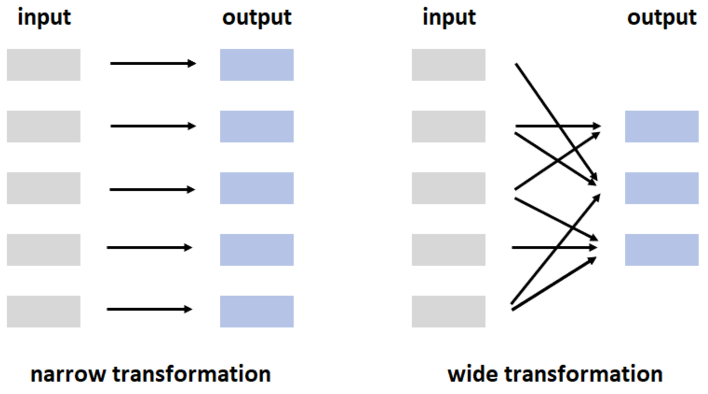


Transformations	Actions

map (func)
flatMap(func)
filter(func)
groupByKey()
reduceByKey(func)
mapValues(func)
sample(...)
union(other)
distinct()
sortByKey()
...
	reduce(func)
	collect()
	count()
	first()
	take(n)
	saveAsTextFile(path)
	countByKey()
	foreach(func)
	...

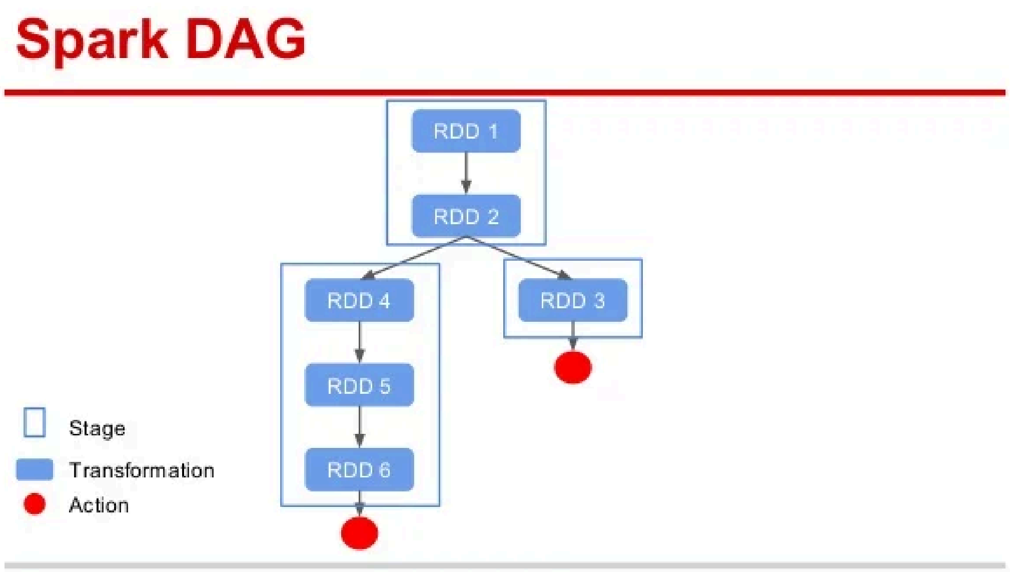
Types of transformations

- Narrow:
  - Operates on single partition (filter, map, withColumn)
- Wide:
  - May require shuffling data across partitions (join, groupBy, repartition)

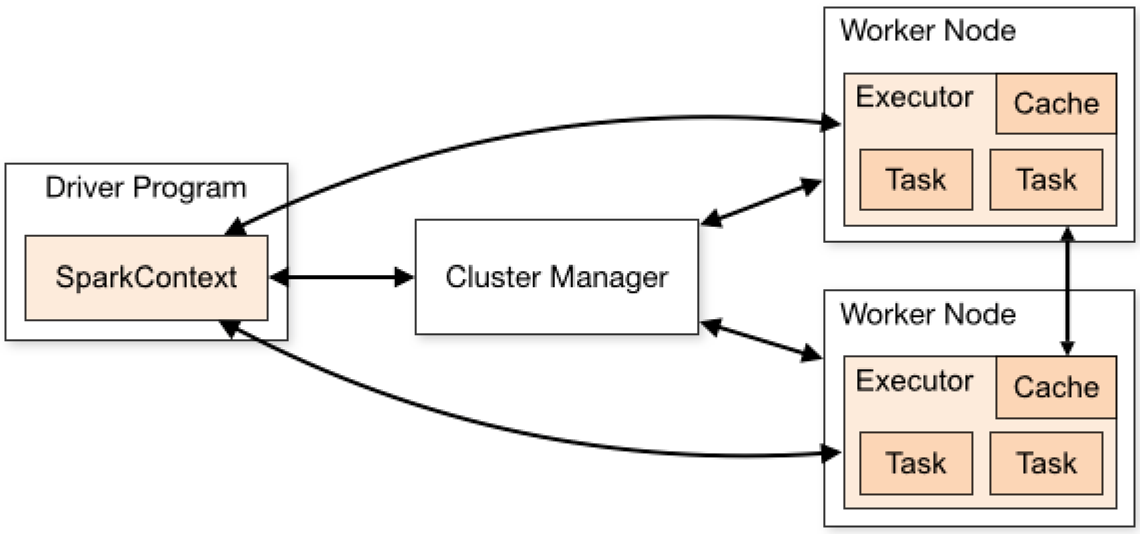


Directed Acyclic Graph (DAG):

- **Workflow Representation:**
  - Spark utilizes DAGs to represent the sequence of transformations and actions within a Spark job.
  - The Driver program translates user code into a DAG, outlining the sequence of transformations and actions to be executed.
- **Nodes and Edges:**
  - A DAG is a directed acyclic graph, meaning data flows in a single direction with no loops.
  - Nodes in the DAG represent RDD operations (transformations and actions), while edges represent the flow of data between them.



Spark Architecture and Execution Flow



Apache Spark utilizes a master-slave architecture for distributed data processing. Here's a breakdown of the key components and execution flow:

Components:

- **Driver Program:** The master node that coordinates the Spark application. It reads the application code, creates a SparkContext, and submits tasks to the cluster manager.
- **SparkContext (or SparkSession):** The entry point for Spark functionality in the driver program. It provides methods for creating RDDs, scheduling tasks, and interacting with the cluster manager.
- **Cluster Manager:** Manages the cluster resources (CPU, memory, network resources) and allocates them to Spark applications on Worker Nodes. Examples include YARN, Mesos, or standalone mode.
- **Worker Nodes:** Slave nodes in the cluster that execute tasks submitted by the driver program.
- **Executors:** Processes running on worker nodes responsible for executing Spark tasks. Each worker node can have multiple executors.

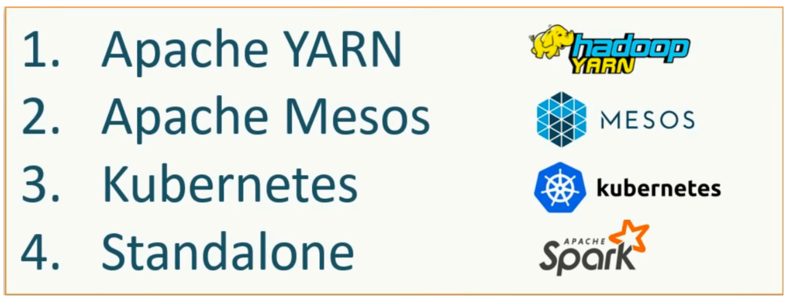
Execution Flow:

1. **Driver Program:** The application starts with the driver program, which defines the Spark job and creates a SparkContext (or SparkSession in newer versions).
2. **RDD Creation:** The driver program uses the SparkContext to create RDDs from various data sources (e.g., HDFS, files, databases).
3. **DAG Generation:** Spark translates the application logic into a DAG, where each node represents a transformation on an RDD and edges represent dependencies between them.
4. **Task Scheduling:** The DAG Scheduler breaks down the DAG into smaller, independent tasks that can be executed on worker nodes.
5. **Task Execution:** The Task Scheduler submits tasks to the cluster manager, which allocates resources on worker nodes and launches executors to execute the tasks.
6. **In-Memory Computation:** Workers execute the tasks on the allocated executors, leveraging in-memory computation for faster processing.
7. **Results Collection:** Task results are sent back to the driver program.
8. **DAG Completion:** Once all tasks are complete, the DAG execution is finished.

Distributed Deployments

For serious use of Spark, you will want to run it on a distributed cluster. There are a few main options for distributed deployment:

- Standalone — Spark’s own simple standalone cluster manager. Great for testing purposes.
- YARN — Run Spark on top of Hadoop NextGen (YARN) which can run distributed workloads on top of Hadoop clusters.
- Mesos — General cluster manager that can also run Hadoop MapReduce and Spark applications.
- Kubernetes — Open-source system for automating deployment, scaling, and management of containerized applications like Spark.



1. Apache YARN
2. Apache Mesos
3. Kubernetes
4. Standalone

hadoop YARN
MESOS
kubernetes
APACHE Spark

Spark Execution Modes :

There are 3 types of execution modes :

1. Local Mode :
   - Everything runs on a single machine - the Spark driver and executors all reside locally.
   - Ideal for development and testing due to its simplicity.
   - Not suitable for large-scale data processing due to resource limitations of a single machine.
2. Client Mode :
   - Spark driver runs on the machine submitting the application (client).
   - Executors run on worker nodes in a separate cluster.
   - Offers more resources than local mode but can introduce latency due to network communication between driver and executors.
3. Cluster Mode :
   - Both the Spark driver and executors run on nodes within the cluster managed by a resource manager (YARN, Mesos, etc.).
   - Provides the best performance and scalability for large-scale data processing.
   - Most common mode for production deployments.

Motivation for this training

I'm interested in learning PySpark because:

1. Spark is considered one of the most powerful tools for handling Big Data.

2. Learning Spark has been challenging for me. I've struggled to find comprehensive examples that cover the entire process in one file.

3. Good sources are expensive.

What is PySpark?

PySpark is an interface for Apache Spark in Python. With PySpark, you can write Python and SQL-like commands to manipulate and analyze data in a distributed processing environment.

What is PySpark used for?

Most data scientists and analysts are familiar with Python and use it to implement machine learning workflows. PySpark allows them to work with a familiar language on large-scale distributed datasets.

Why PySpark?

- PySpark is highly popular and widely used due to its simplicity and accessibility.
- It contains extensive libraries and resources for machine learning and deep learning tasks.
- To work with large-scale and real-time datasets, knowledge of Python and R frameworks alone will not suffice.

How to Install PySpark

1. **Run the Docker Container with PySpark:**

   Run the Docker container with the specified port mapping:

   ```
   docker run -p 8888:8888 jupyter/all-spark-notebook:spark-3.2.1
   ```

2. **Access Jupyter Notebook:**

   Once the container is running, navigate to the following link in your browser: **http://localhost:8888**

3. **Retrieve Jupyter Token:**

   Inside the container's terminal, execute the following command to list Jupyter server information:

   ```
   jupyter server list
   ```

   Copy the token from the output, which will be in the format:

http://<container_id>:8888/?token=<token_value> :: /home/jovyan

Extract the token value, for example:

ead98aad738c241c31bcfdd762bfe37fb9b95b14dfd978ba

4. **Start Jupyter Notebook:**

   Paste the token value into the password field

5. **Test Spark:**

   go to the terminal and run the following command:

   ```
   spark-shell
   ```

6. **Now, create a new notebook and let's start.**