# Introduction to Apache Spark

ddas.tech/introduction-to-apache-spark/

September 23, 2022

*Created By: Debasis Das (Sep – 2022)*

## Table of Contents

## What is Apache Spark?

Apache Spark is a multi-language engine for executing data engineering, data science and machine Learning on a single-node machine or clusters

Spark abstracts the execution of code on multiple machines on a cluster

- Spark is an open source big data framework
- Data Processing Platform engine
- It distributes data in file system across the cluster and processes the data in parallel
- Advanced Analytics engine which can process real-time data
- An in-memory processing framework which is efficient and is much faster in comparison to Map-Reduce.

## Apache Spark Ecosystem

Apache Spark is a distributed data processing framework

- Spark Ecosystem consists of 2 layers

- **Layer 1** consists of 4 libraries such as
    - Spark SQL and DataFrames (Allows us to use SQL Queries to process the data)
    - Spark Streaming – (Allows us to process continuous stream of data)
    - MLLib (Machine Learning)
    - GraphX (graph)
- **Layer 2** Consists of Spark Core which consists of a distribute computing engine also called as Spark Engine and Spark Core APIs available in Scala, Java, Python and R

Sparks requires a cluster of hardware to do distributed data processing however Spark itself does not manage the Clusters. Cluster manager such as Hadoop YARN, Kubernetes and Mesos can be used for managing the resources.

Spark only does distributed data processing and itself does not store any data. The data storage can be in HDFS, Amazon S3 , Google Cloud Storage etc

**Spark Compute Engine** is responsible for

- Breaking the data processing into smaller tasks,
- Scheduling the tasks to be run on the clusters for parallel execution.
- Managing and monitoring the tasks.
- For interacting with the storage manager and cluster manager
- Provides fault tolerance for the spark jobs

**Spark Core APIs** – Programming interface that provides the core apis in Scala, Java, Python and R.

## Installing Apache Spark on a Mac

**- Install Java 8 or Java11 on Mac**
using **brew install java11**

**- Verify Java Version using java -version**

```
(base) Debasiss-MacBook-Pro:~ debasisdas$ java -version
openjdk version "11.0.15" 2022-04-19
OpenJDK Runtime Environment Homebrew (build 11.0.15+0)
OpenJDK 64-Bit Server VM Homebrew (build 11.0.15+0, mixed mode)
```

**- Download Apache Spark** by downloading from https://spark.apache.org
https://www.apache.org/dyn/closer.lua/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz

- After downloading untar it using the **tar -zxvf spark-3.3.0-bin-hadoop3.tar**

- **Set the SPARK_HOME** path to the folder which contains the **spark-3.3.0-bin-hadoop3** folder

- **Set the PATH** using **export PATH=$PATH:$SPARK_HOME/bin**

- Check the Spark Installation using **spark-shell** command

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.15)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- Quit the spark shell using **:q**

- **Check for pyspark shell** using **pyspark** command

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Python version 3.8.5 (default, Sep  4 2020 02:22:02)
Spark context Web UI available at http://192.168.1.5:4040
Spark context available as 'sc' (master = local[*], app id = local-1655607419321).
SparkSession available as 'spark'.
```

- Check the spark version

```
>>> spark.version
'3.3.0'

Control d to exit from the pyspark shell
```

## Spark Session Config

The Spark Session Configs can be set using one of the following options

- Environment Variables
- spark-defaults.conf
- command line options
- SparkConf from application code

Spark properties control most application parameters and can be set using a SparkConf object. The **SparkConf** object created can then be passed to SparkContext.

SparkConf from application code should mostly be used for **Run-time behavio**r such as **task.maxFailures**

Environment variables can be used to set per machine settings such as the IP address.

Deployment / Resources configuration such as **driver.memory** or number of **executor.instances** can be set as **command line** options

When a Spark-submit is called it merges the configuration from environment variables, spark-defaults.conf and command line options. to create a set of initial configs for our application. On top of this config developers can add/update config using SparkConf from the application code. The combination of all these configuration becomes the SessionConfig which is the Final Spark Config for your applications

**Precedence** SparkConf (Application Code ) **>** Command Line **>** spark-defaults.conf **>** Environment Variables

## Spark Context

- Allows spark application to access spark cluster with the help of a Resource Manager such as Yarn, Mesos or Kubernetes.
- Once a spark context is created it can be used to create RDDs, ingress Spark Service, Run Jobs etc.
- Only one SparkContext may be active per JVM

- Functions of Spark Context in a Spark Application are as follows
    - Get the current status of the application
    - Set configurations
    - Cancel a Job
    - Access services
    - Closure cleaning in Spark
    - Programmable dynamic allocation
    - Register Spark Listener
    - Cancel a stage
    - Access persistent RDD.

## Apache Spark RDD

- RDD stands for Resilient Distributed DataSet.
- Is the fundamental data structure of Apache Spark.
- Immutable collection of objects which computes on different node of a cluster
- Fault tolerant
- Spark RDD can be cached and manually partitioned
- The RDD can be persisted in memory.

## Difference between RDD, DataFrame and DataSet

- **RDD** – RDD stands for Resilient Distributed datasets.
    Resilient – a RDD partition can be recreated and processed at anywhere in the cluster.
    - Is a fundamental data structure of Spark
    - Read only partition collection of records
    - Is Fault tolerant (Stores information on how they are created). In case there is a fault the driver will notice the fault and will assign the RDD partition to available executor cores. The new executor core will reload the RDD partition and will start the processing.
    - Allows developers to perform in-memory computations on large clusters.
    - Are language native object
    - Does not have row column structure
    - RDDs are internally broken down into partitions
- **Spark DataFrame**
    - Unlike RDD, data is organized as named columns
    - Immutable distributed collection of data

- **Spark DataSet APIs**
    - Extension to Spark Dataframe API which provides type-safe, object oriented programming interface
    - Takes advantage of Spark Catalyst optimizer by exposing expressions and data fields to a query planner
    - works on a JVM based language such as Java or Scala

## Lazy Evaluation in Spark?

- All transformations in Apache Spark is Lazy. i.e they do not compute the results right away.
- It remembers the transformations applied to some base data set.
- Spark computes transformations when an action requires a result for the driver program.

## Narrow Transformations vs Wide Transformations

- **Narrow Transformations** – Result of Map, Filter such that data is from a single partition only. eg of Narrow Transformations are
    - Map
    - FlatMap
    - MapPartition
    - Filter
    - Sample
    - Union
- **Wide Transformations** – Result of groupByKey() and reduceByKey() like functions. The data required to compute the record in a single partition may live in many partitions of the parent RDD.
- Eg of Wide Transformations are
    - Intersection
    - Distinct
    - ReduceByKey
    - GroupByKey
    - Join
    - Cartesian
    - Repartition
    - Coalesce

## Catalyst Optimizer

- Is at the core of the Spark SQL
- Builds an extensible query optimizer.
- Has libraries specific to relational query processing (expressions, logical query plans)

- Handles different phase of query optimization such as analysis, logical optimization, physical planning and code generation to compile parts of queries to java byte code.

## Spark Data Sources

- Spark Data Sources can be external or internal data sources
- **External Data Sources** Include
  - JDBC Data Sources such as Oracle, SQLServer, PostgreSQL
  - No SQL Data Systems such as MongoDB
  - Cloud Data Warehouses such as Snowflake, Redshift
  - Stream Integrators (Kafka, Kinesis)
- Spark Datasource APIs can be used to directly connect to these external data sources
- Data Integration tools can also be used to get the data from the external data sources to the Distributed Data Storage
- **Internal Data Source** are the distributed data storage such as HDFS, Amazon S3, Azure Blob, Google Cloud etc
- Commonly used file formats for the internal storage are CSV, JSON, Parquet, AVRO, Plain Text etc

## Different Read Mode options for Spark?

- PERMISSIVE
- DROPMALFORMED (drops the Mal formed record)
- FAILFAST (Raises an exception and terminates on encountering a malformed record)

## References

- https://data-flair.training
- https://databricks.com