**INTRODUCTION TO APACHE SPARK ARCHITECTURE**

[**https://www.educba.com/apache-spark-architecture/?source=leftnav**](https://www.educba.com/apache-spark-architecture/?source=leftnav)

Apache Spark **Architecture is an open-source framework**-based component that are **used to process a large amount of unstructured, semi-structured and structured data for analytics**. Spark Architecture is considered as an **alternative to Hadoop and map-reduce architecture** **for big data processing**. Spark architecture **associated** with Resilient Distributed Datasets (**RDD**) and Directed Acyclic Graph (**DAG)** **for data storage and processing**. Also, it has **four components** that are part of the architecture such as **spark driver, Executors, Cluster managers, Worker Nodes**. Spark **uses the Dataset and data frames as the primary data storage** component that **helps to optimize the Spark process and the big data computation**.

Spark helps in resolving high computational tasks.

**APACHE SPARK ARCHITECTURE**

The Architecture of Apache **spark has loosely coupled components**. Spark **consider the master/worker process in the architecture and all the task works on the top** of the **Hadoop distributed file system**. Apache **spark** makes **use of Hadoop for data processing and data storage processes**. They are **considered to be in-memory data processing engine and makes their applications run on Hadoop clusters faster than a memory**. Having in-memory processing **prevents the failure of disk I/O**. Spark **allows the heterogeneous job to work** with the same data. Spark **divides its data into partitions, the size of the split partitions depends on the given data source**.

the two main **implementations** of Apache Spark Architecture:

1. Resilient Distributed Datasets (**RDD**)

It is responsible for **providing API for controlling caching and partitioning**. It’s an important **toolset for data computation**. It helps in recomputing elements in case of failures and is considered to be immutable data and acts as an interface. **Transformations and actions** are the two operations done by RDD.

2. Directed Acyclic Graph (**DAG**)

It **forms a sequence connection from one node to another**. The **driver converts the program into DAG for each** job. The Apache Spark Eco-system has various components like API core, Spark SQL, Streaming and real-time processing, MLIB, and Graph X. Some terminologies that to be learned here is Spark shell which helps in reading large volumes of data, Spark context -cancel, run a job, task ( a work), job( computation)

Components of Apache Spark Architecture

The Four main components of Spark are given below and it is necessary to understand them for the complete framework.

* **Spark Driver**
* **Executors**
* **Cluster manager**
* **Worker Nodes**

The following diagram shows the Architecture and Components of spark:

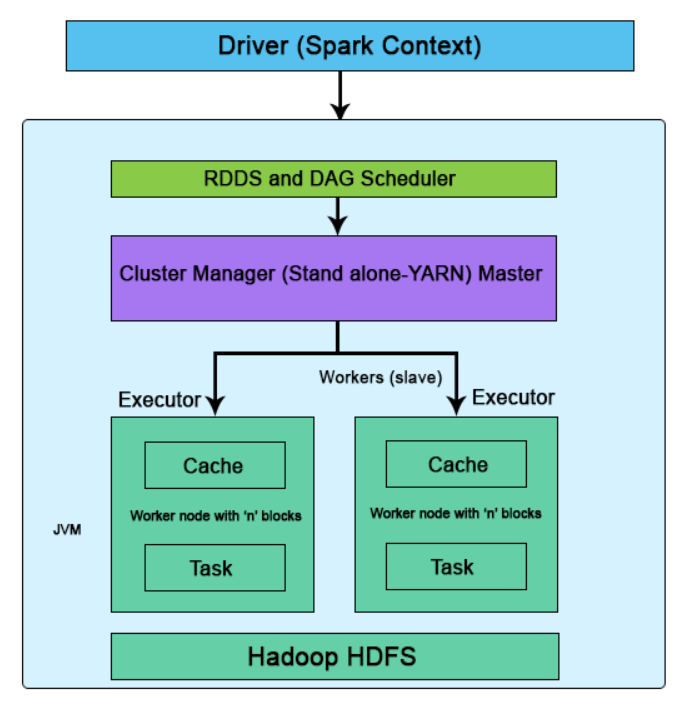


Figure 1 : Standalone mode of Apache Spark Architecture

The execution flow begins as follows:

1. Spark Driver

The driver’s **responsibility is to coordinate the tasks and the workers for management**. It’s an Application JVM process and is considered a master node. A **driver splits** **the spark into tasks and schedules to execute on executors in the clusters**. the driver programs invoke the main application and create a spark context (acts as a gateway) collectively monitor the job working within the given cluster and connect to a Spark cluster All the functionalities and the commands are done through the spark context.

Spark context is an entry for each session. **Spark driver has more components to execute jobs in the clusters. Spark clusters get connected to different types of cluster managers and simultaneously context acquires worker nodes to execute and store data**. In the cluster, when we execute the process their job is subdivided into stages with gain stages into scheduled tasks.

2. Executor

It is **responsible for the execution of a job and stores data in a cache**. At the very **initial stage, executors register with the drivers**. This **executor has a number of time slots to run the application concurrently**. Executors **perform read/ write process on external sources**. The executor runs the job when it has loaded data and they are been removed in the idle mode. The executor is enabled by dynamic allocation and they are constantly included and excluded depending on the duration. During the execution of the tasks, the **executors are monitored by a driver program**. Executors execute users’ task in java process.

3. Cluster Manager

It **helps in managing the clusters which have one master and number of slaves**. There are **two types of cluster managers like YARN and standalone** both these are **managed by Resource Manager and Node**. cluster work on Stand-alone requires Spark Master and worker node as their roles. **The responsibility of the cluster manager is to allocate resources and to execute the task**.

4. Worker Nodes

They are the **slave nodes**; the main **responsibility** is to **execute the tasks and the output of them** is **returned back to the spark context**. They **communicate** with **the master node about the availability of the resources**. Spark context executes it and issues to the worker nodes**. Each worker nodes are been assigned one spark worker for monitoring**. They make the computation very simply by increasing the worker nodes (1 to n no of workers) so that all the tasks are performed parallel by dividing the job into partitions on multiple systems. The other **element task is considered to be a unit of work and assigned to one executor, for each partition spark runs one task**.

**INTRODUCTION TO SPARK EXECUTOR**

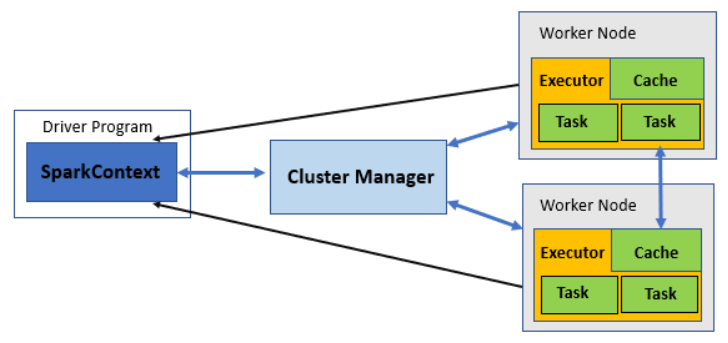
<https://www.educba.com/spark-executor/>

The spark executors in apache spark can **enhance the performance** of the system.

There is a **distributing agent** called spark executor which **is responsible for executing the given tasks**. Executors in Spark are **the worker nodes that help in running individual tasks by being in charge of a given spark job**. These are launched at the beginning of Spark applications, and as soon as the task is run, results are immediately sent to the driver. In-**memory the storage provided by executors for Spark RDD** and **are cached by programs by the user with block manager**. Run the application for a complete lifespan, by which executor’s static allocation is inferred.

HOW APACHE SPARK EXECUTOR WORKS?

The **executor starts over an application** every time it gets an event. **Spark on YARN is used while using this executor and this is generally not compatible with Mesos**. Using **Spark executor can be done in any way like in start running applications of Sparkafter MapR FS, Hadoop FS, or Amazon S# destination close files**. **Each time the Hadoop FS destination closes a file, the spark application each time, can convert Arvo files into Parquet**. **In an external system, the Spark application is started**. It does **not wait for the monitoring application nor complete**. Additional **processing is done after submitting the application successfully**.



Client or cluster mode can be used to run an application by it. But Client mode is used only when the use of resources is not a deal to be considered. Make sure you perform the task prerequisite before using the Spark executor. The number of worker nodes has to be specified before configuring the executor. Enabling dynamic memory allocation can also be an option by specifying the maximum and a minimum number of nodes needed within the range. Also, by specifying the minimum amount of memory required, used by executor and application driver one can pass additional cluster manager properties to it.

Specify the custom Java home and Spark directories and a proxy user of Hadoop and also the Kerberos credentials. Language can be specified which is used to write applications and then the properties as per specific language defined. Executors can generate the events for a variant event stream which can also be configured.

Conditions in creating a Spark Executor -

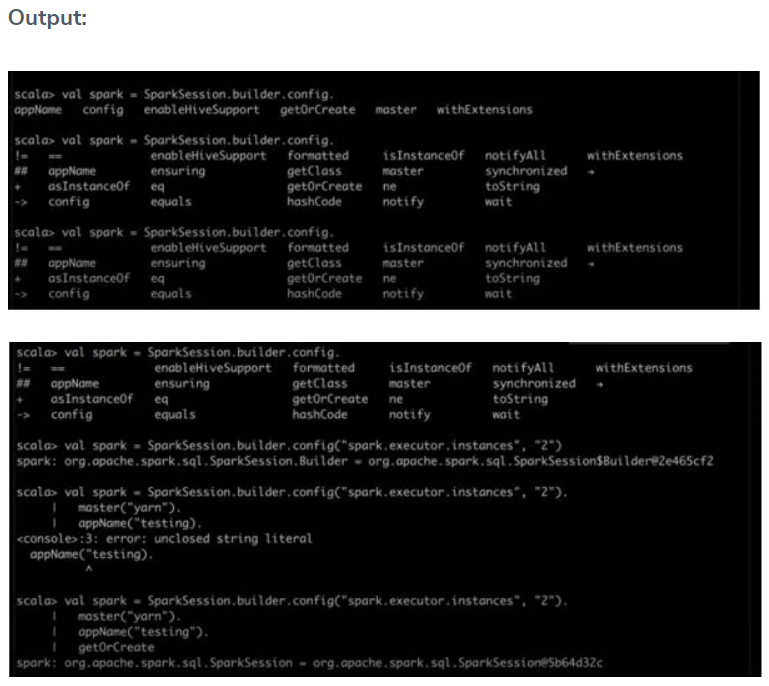
When the message – RegisteredExecutor is received, by coarseGrainedExecutorBackend. And this happens only in YARN.

When the registration of MesosExecutorBackend in Mesos in Spark.

For local mode, a local endpoint will be created.

Code –

at org.apache.spark. SparkContext$Sanonfun$assertNoOtherContextIsRunning$2.appl Que  
at org.apache.spark.SparkContext anonfun$assertNoOtherContextIsRunning$2.apply(SparkContext.scala:245  
at scala.Option.foreach  
at org.apache.spark.SparkContext.assertNoOtherContext IsRunning  
at org.apache.spark.SparkContext.mark Partially Constructed  
at org.apache.spark. SparkContext.<init>  
scala  
sc.stop  
val conf = new SparkConf.  
setAppName("testing").  
setMaster("yarn").  
set("spark.executor.instances", "2") conf: org.apache.spark.SparkConf - org.apache.spark. SparkConfe69d7b1f9  
val sc = new SparkContext(conf) sc: org.apache.spark. SparkContext - org.apache.spark. SparkContext e31b4e1  
sc.stop  
scala import org.apache.spark.sql.SparkSession SparkSession SparkSession Extensions  
import org.apache.spark.sql.SparkSession import org.apache.spark.sql.SparkSession  
scala val spark Spark SparkConf SparkContext SparkSession  
scala val spark SparkSession.  
Builder clearActiveSession getactiveSession builder clear Default Session getDefault Session  
spark = SparkSession.builder.  
appName config enableHiveSupport getorCreate  
val spark  
SparkSession.builder. // hit tab  
val spark = SparkSession.builder.config("spark.executor.instances", "2".  
master("yarn").  
appName("testing).  
val spark = SparkSession.builder.config("spark.executor.instances", "2").  
master("yarn").  
appName("testing").  
getOrCreate



Advantages and uses to be noted about Apache Executor in Spark

1. Every application has its own process of execution while the application is running in a task of multiple threads.

2. To the underlying cluster manager, the spark executor is agnostic. meaning as long as the process is done, communication with each other is done.

3. Acceptance of incoming connections from all the other executors.

4. The executor should run closer to the worker nodes because the driver schedules tasks on the cluster. These can be preferably run in the same local area network.

**DRIVER**

Driver is a Java process. This is the process where the main() method of our Scala, Java, Python program runs. It **executes the user code and creates a SparkSession or SparkContext and the SparkSession** is **responsible to create DataFrame, DataSet, RDD, execute SQL, perform Transformation & Action**, etc.

Responsibility of DRIVER

1. The main() method of our program runs in the Driver process. It **creates Spark Session** or SparkContext.
2. **Conversion of the user code into Task** (transformation and action). It looks at the user code and **determines are the possible Tasks, i.e. the number of tasks to be performed is decided by the Driver**.
3. Helps **to create the Lineage, Logical Plan and Physical Plan**.
4. Once the **Physical Plan is generated, the Driver schedules the execution of the tasks by coordinating with the Cluster Manager**.
5. **Coordinates with all the Executors for the execution of Tasks**. It **looks at the current set of Executors and schedules our tasks**.
6. Keeps track of the data (in the form of metadata) which was cached (persisted) in Executor’s (worker’s) memory.

**INTRODUCTION TO SPARK DATASET**

<https://www.educba.com/spark-dataset/?source=leftnav>

Dataset is the best of **both RDD and Dataframe**. RDD **provides compile-time type safety**, but there is an **absence of automatic optimization**. **Dataframe provides automatic optimization**, but **it lacks compile-time type safety.** **Dataset provides both compile-time type safety as well as automatic optimization**. Hence, the dataset is the best choice for Spark developers using Java or Scala.

Spark **Dataset** is one of the **basic data structures by SparkSQL**. It **helps in storing the intermediate data for spark data processing**. Spark **dataset with row type is very similar to Data frames that work as a tabular form on the Resilient distributed dataset (RDD)**. The Datasets in Spark are known for their **specific features** such **as type-safety, immutability, schemas, performance optimization, lazy evaluation, Serialization, and Garbage Collection**. The Datasets are supported through Scala and Java programming APIs. Spark’s dataset **supports both** **compile-time safety and optimizations**, making it a preferred choice for implementation in the spark framework.

WHY DO WE NEED SPARK DATASET?

RDD is the **core of Spark**. Inspired by SQL and to make things easier, Dataframe was created on top of RDD. Dataframe is equivalent to a table in a relational database or a DataFrame in Python.

RDD provides compile-time type safety, but there is an absence of automatic optimization in RDD.

Dataframe provides automatic optimization, but it lacks compile-time type safety.

Dataset is added as an extension of the Dataframe. Dataset combines both RDD features (i.e. compile-time type safety) and Dataframe (i.e. Spark SQL automatic optimization).

[RDD(Spark 1.0)] -> [Dataframe(Spark1.3)] -> [Dataset(Spark1.6)]

As Dataset has compile-time safety, it is only supported in a compiled language(Java & Scala ) but not in an interpreted language(R & Python). But Spark Dataframe API is available in all four languages( Java, Scala, Python & R ) supported by Spark.

HOW TO CREATE A SPARK DATASET?

<https://www.educba.com/spark-dataset/?source=leftnav>

There are multiple ways of creating a Dataset based on the use cases.

**Features** of Spark Dataset -

1. **Type Safety**: Dataset provides compile-time type safety. It means that the application’s syntax and analysis errors will be checked at compile time before it runs.

2. **Immutability**: Dataset is also immutable like RDD and Dataframe. It means we can not change the created Dataset. Every time a new dataset is created when any transformation is applied to the dataset.

3. **Schema**: Dataset is an in-memory tabular structure that has rows and named columns.

4. **Performance and Optimization**: Like Dataframe, the Dataset also uses Catalyst Optimization to generate an optimized logical and physical query plan.

5. **Programming language**: The dataset api is only present in Java and Scala, which are compiled languages but not in Python, which is an interpreted language.

6. **Lazy Evaluation**: Like RDD and Dataframe, the Dataset also performs the lazy evaluation. It means the computation happens only when action is performed. Spark makes only plans during the transformation phase.

7. **Serialization and Garbage Collection**: The spark dataset does not use standard serializers (Kryo or Java serialization). Instead, it uses Tungsten’s fast in-memory encoders, which understand the internal structure of the data and can efficiently transform objects into internal binary storage. It uses off-heap data serialization using a Tungsten encoder, and hence there is no need for garbage collection.

**INTRODUCTION TO SPARK DATAFRAME**

A spark data frame can be said to be a **distributed data collection organized into named columns and is also used to provide operations such as filtering, computation of aggregations, grouping, and can be used with Spark SQL**. **Data frames can be created by using structured data files, existing RDDs, external databases, and Hive tables**. It is basically termed and known as an abstraction layer which is built on top of RDD and is also followed by the dataset API, which was introduced in later versions of Spark (2.0 +). Moreover, the datasets were not introduced in Pyspark but only in Scala with Spark, but this was not the case in the case of Dataframes. **Data frames, popularly known as DFs, are logical columnar formats that make working with RDDs easier and more convenient**, also making use of the same functions as RDDs in the same way. If you talk more on the conceptual level**, it is equivalent to the relational tables along with good optimization features and techniques**.

HOW TO CREATE A DATAFRAME?

A Data Frame is generally created by any one of the mentioned methods. It can be **created by making use of Hive tables, external databases, structured data files or even in the case of existing RDDs**. These all ways can create these named columns known as Dataframes used for the processing in Apache Spark. By making use of SQLContext or SparkSession, applications can be used to create Dataframes.

SPARK DATAFRAMES OPERATIONS

In Spark, a data frame is the distribution and collection of an organized form of data into named columns which is equivalent to a relational database or a schema or a data frame in a language such as R or python but along with a richer level of optimizations to be used. It is **used to provide a specific domain kind of language that could be used for structured data manipulation**.

The below mentioned are some basic Operations of Structured Data Processing by making use of Dataframes.

<https://www.educba.com/spark-dataframe/?source=leftnav>

Advantages of Spark DataFrame

The **data frame is the Data’s distributed collection**, and therefore the data is organized in named column fashion.

They are **more or less similar to the table in the case of relational databases** and **have a rich set of optimization**.

Dataframes are **used to empower the queries written in SQL** and also **the data frame API**.

It can be **used to process both structured as well as unstructured** kinds of **data**.

The use of a catalyst optimizer makes **optimization easy and effective**.

The libraries are present in many languages such as Python, Scala, Java, and R.

This is used to provide **strong compatibility with Hive** and is **used to run unmodified Hive queries on the already present hive warehouse**.

It can **scale** very well, right **from a few kbs on the personal system to many petabytes on the large clusters**.

It is used to **provide an easy level of integration with other big data technologies and frameworks**.

The **abstraction which they provide to RDDs is efficient** and makes processing faster.

**INTRODUCTION TO RDD IN SPARK**

An RDD, which stands **for Resilient Distributed Dataset**, is one of the most important concepts in Spark. It is **a read-only collection of records** which is **partitioned and distributed across the nodes in a cluster**. It **can be transformed into some other RDD through operations**, and **once an RDD is created, it cannot be changed**; **rather, a new RDD will be created**.

One **important feature through which Spark overcame the limitations of Hadoop is via RDD** because rather than replicating the data, Resilient Distributed Datasets (RDD) **maintains the data across the nodes in a cluster and will recover back the data with the help of a lineage graph**. In Hadoop, **the data was redundantly stored among the machines, which provided the property of fault tolerance**. Thus **an RDD is a fundamental abstraction provided by Spark for distributed data and computation**.

The different ways of **creating an RDD** are

* **Loading an external data set**
* **Passing the data through Parallelize method**
* **By transforming an existing RDD**

Let’s discuss each of them in detail, but before that, we need to set up a spark-shell which is the driver program of spark. In this article, we have included the lines of code in scala. RDD’s can have any type of Python, Java or Scala objects, including user-defined classes. Thus below are the steps to be followed to launch spark-shell.

Launching Spark-Shell –

<https://www.educba.com/rdd-in-spark/?source=leftnav>