## I am the type of person, if you ask me a question and i don't know the answer,

I am gonna tell you i don't know.I know how to find the answer and i will find the answer.

## The interview might not going very well, But i can assure you i have the potential to get the job done.

I was responsible for sqooping data from rdbms to hive, sometimes to hdfs.

I have solid understanding language like java, python, javascript, scala as well

I am pretty good at sql queries

Pretty good knowledge of spark scala and pyspark as well

**Toyota data:**  
Review Data analytic

## Toyota Uses Big Data to Guard Against Accelerator-Brake Mix-Up:

The system is a response to an increasingly common cause of traffic accident in aging Japan where the driver, often elderly, mistakes the accelerator for the brake.

Some 15% of fatal accidents on Japanese roads in 2018 were caused by drivers who were 75 years or older, showed a report from the government, which actively encourages elderly drivers to give up their licenses.

Toyota's announcement comes as automakers globally invest heavily in so-called active safety features as they work to develop fully autonomous cars.

## Behavior Prediction

"This opens up a new opportunity for us to communicate with millions of existing Toyota customers to improve sales and retention for their brand," said Andrew Gillman, vice president of sales and marketing for automotiveMastermind.

Our behavior prediction technology shows dealers what customers they should be contacting and marketing to in order to increase sales in the fastest, most efficient way. We have seen our dealer partners experience significant increases in sales as well as growth in customer retention immediately upon implementing our technology."

MapReduce

# **How it Works - Hadoop MapReduce Tutorial**

## **What is MapReduce?**

**MAPREDUCE** is a software framework and programming model used for processing huge amounts of data. **MapReduce** program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

## **How MapReduce Works? Complete Process**

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

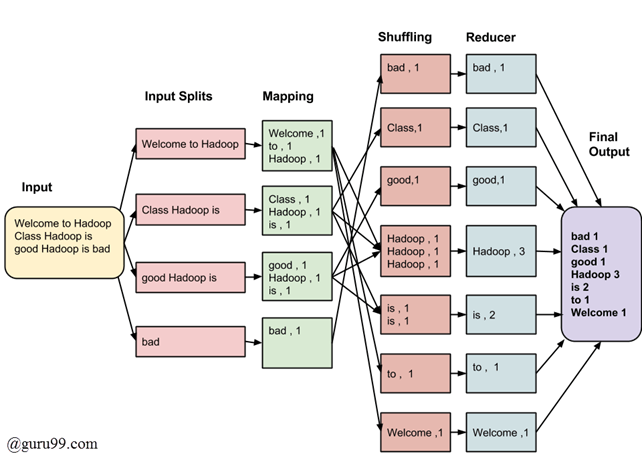
Let's understand this with an example –

Consider you have following input data for your Map Reduce Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad



MapReduce Architecture

The final output of the MapReduce task is

|  |  |
| --- | --- |
| bad | 1 |
| Class | 1 |
| good | 1 |
| Hadoop | 3 |
| is | 2 |
| to | 1 |
| Welcome | 1 |

The data goes through the following phases

**Input Splits:**

An input to a MapReduce job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map

**Mapping**

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

**Shuffling**

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

**Reducing**

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

## **MapReduce Architecture explained in detail**

* One map task is created for each split which then executes map function for each record in the split.
* It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel.
* However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
* For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default).
* Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
* Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
* Map output is intermediate output which is processed by reduce tasks to produce the final output.
* Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
* In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
* Reduce task doesn't work on the concept of data locality. An output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
* On this machine, the output is merged and then passed to the user-defined reduce function.
* Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

## **How MapReduce Organizes Work?**

Hadoop divides the job into tasks. There are two types of tasks:

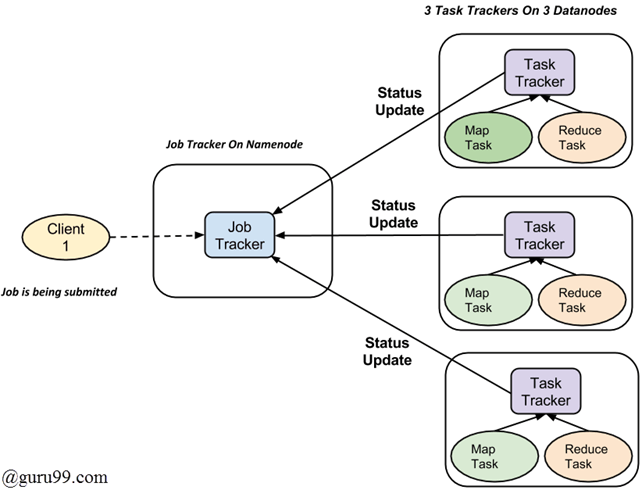
1. **Map tasks** (Splits & Mapping)
2. **Reduce tasks** (Shuffling, Reducing)

as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

1. **Jobtracker**: Acts like a **master** (responsible for complete execution of submitted job)
2. **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job

For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.



* A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
* It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
* Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
* Task tracker's responsibility is to send the progress report to the job tracker.
* In addition, task tracker periodically sends **'heartbeat'** signal to the Jobtracker so as to notify him of the current state of the system.
* Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Hive commands behind the scene

* Every time you run a Hive commands it turns into a MapReduce Job. You don't have to know what is happening behind the scene but if you do it helps to optimize your query.
* When the data stored in Hdfs for Hive its not in tabular format, it's just a text file with comma separated or other delimiter.
* The query first goes to the mapper function and then based on the where conditions it searches each record to satisfy the where clause.
* After that the mapper select the specific columns you have provided in your select statement.
* As we know that MapReduce Works as key value pairs, so the key here is each row(number provided by the mapper)And the value is each individual column value that we provided.

Hive Partitioning.

Pros:

* It distributes execution load horizontally.
* It improves query performance.
* If you want to logically organize your data into directories.
* In partition faster execution of queries with the low volume of data takes place.
* For example, search population from Vatican City returns very fast instead of searching entire world population
* Use partition to those column which occurs in where clause mostly.   
    
  Cons:
* There is the possibility of too many small partition creations- too many directories.
* Partition is effective for low volume data. But there some queries like group by on high volume of data take a long time to execute. For example, grouping population of China will take a long time as compared to a grouping of the population in Vatican City.

Hive Bucketing

Pros:

* It provides faster query responses like portioning.
* In bucketing due to equal volumes of data in each partition, joins at Map side will be quicker.

Cons:

* We can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually by programmers.

# 

Sqoop import behind the scene

1. When we run the sqoop import command it creates java classes and those java are used to import data from mysql to hdfs.

2. It also uses parallel processing, it's not just one thread running to process the import, there are by default 4 threads are running.

3. When we use the sqoop command there is a mapreduce job running behind the scene.

4. In the line where it says,"Beginning code generation". This is where java classes are generated.

5. The line that says “Executing SQL statement: select t.\* from 'tablename' as t limit 1”, This is where it collects the metadata of the table from line one that is gonna be used in java classes.

6. After creating the metadata it generates a jar file.

7. And then it started the import job.

Here are the 5 Vs of big data:

* **Volume** refers to the vast amount of data generated every second. Just think of all the emails, Twitter messages, photos, video clips and sensor data that we produce and share every second. We are not talking terabytes, but zettabytes or brontobytes of data. On Facebook alone we send 10 billion messages per day, click the like button 4.5 billion times and upload 350 million new pictures each and every day. If we take all the data generated in the world between the beginning of time and the year 2000, it is the same amount we now generate every minute! This increasingly makes data sets too large to store and analyze using traditional database technology. With big data technology we can now store and use these data sets with the help of distributed systems, where parts of the data is stored in different locations, connected by networks and brought together by software.
* **Velocity** refers to the speed at which new data is generated and the speed at which data moves around. Just think of social media messages going viral in minutes, the speed at which credit card transactions are checked for fraudulent activities or the milliseconds it takes trading systems to analyze social media networks to pick up signals that trigger decisions to buy or sell shares. Big data technology now allows us to analyze the data while it is being generated without ever putting it into databases.
* **Variety** refers to the different types of data we can now use. In the past we focused on structured data that neatly fits into tables or relational databases such as financial data (for example, sales by product or region). In fact, 80 percent of the world’s data is now unstructured and therefore can’t easily be put into tables or relational databases—think of photos, video sequences or social media updates. With big data technology we can now harness differed types of data including messages, social media conversations, photos, sensor data, video or voice recordings and bring them together with more traditional, structured data.
* **Veracity** refers to the messiness or trustworthiness of the data. With many forms of big data, quality and accuracy are less controllable, for example Twitter posts with hashtags, abbreviations, typos and colloquial speech. Big data and analytics technology now allows us to work with these types of data. The volumes often make up for the lack of quality or accuracy.

But all the volumes of fast-moving data of different variety and veracity have to be turned into value! This is why value is the one V of big data that matters the most.

**Value** refers to our ability turn our data into value. It is important that businesses make a case for any attempt to collect and leverage big data. It is easy to fall into the buzz trap and embark on big data initiatives without a clear understanding of the business value it will bring.

Spark Performance Optimization

1. Right configuration

2. Like right no of executors and executor cores

3. Right memory configuration

4. Enabling spark shuffle service

5. Avoid unnecessary Calculations

6. Avoid data Shuffling

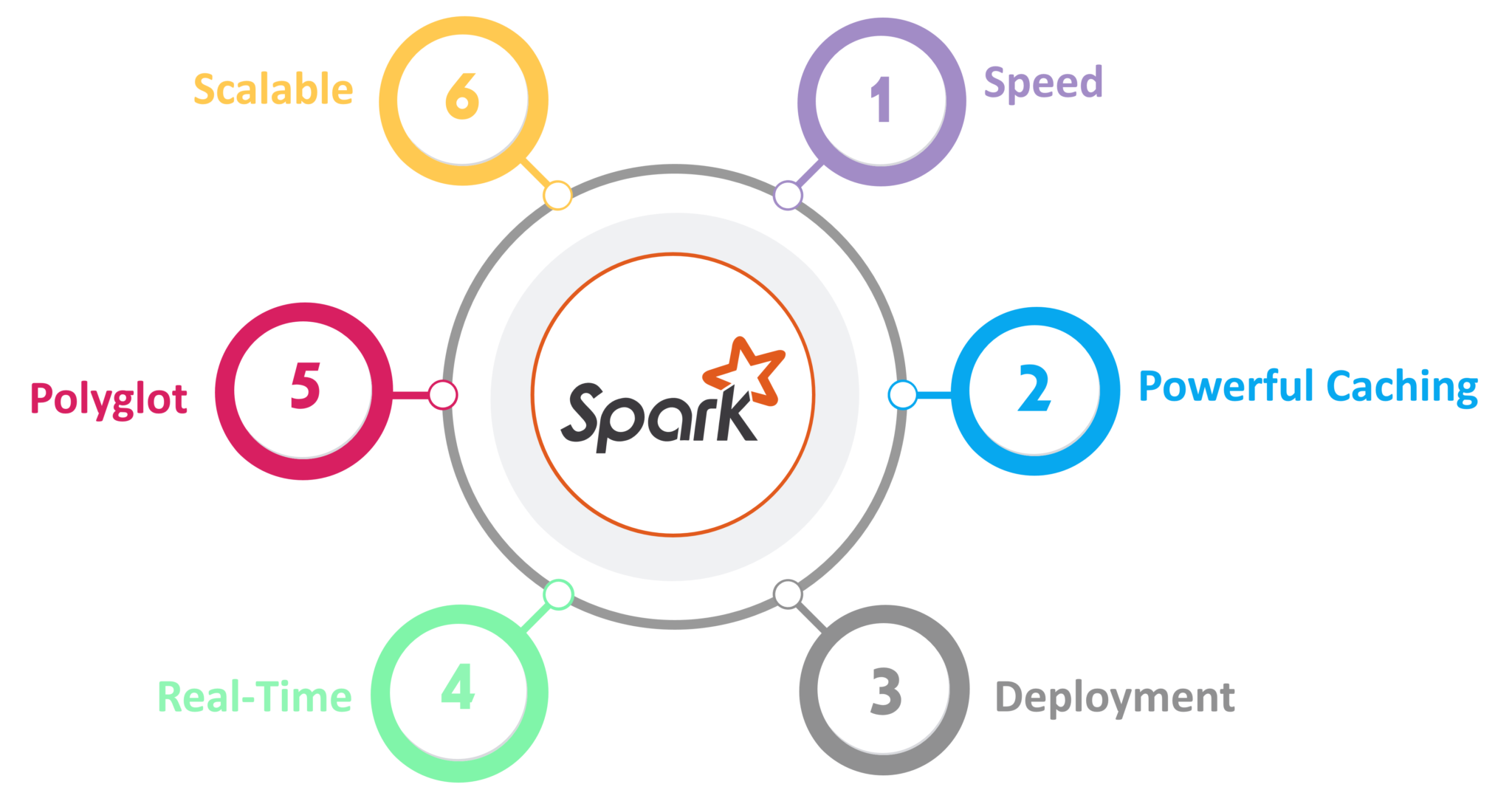
Spark Architecture

Apache Spark is an open-source cluster computing framework which is setting the world of Big Data on fire. According to [*Spark Certified Experts*](https://www.edureka.co/apache-spark-scala-training), Sparks performance is up to 100 times faster in memory and 10 times faster on disk when compared to Hadoop. In this blog, I will give you a brief insight on Spark Architecture and the fundamentals that underlie Spark Architecture.

## Spark & its Features

Apache Spark is an open source cluster computing framework for real-time data processing. The main feature of Apache Spark is its *in-memory cluster computing* that increases the processing speed of an application. Spark provides an interface for programming entire clusters with implicit *data parallelism and fault tolerance*. It is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries, and streaming.

### Features of Apache Spark:

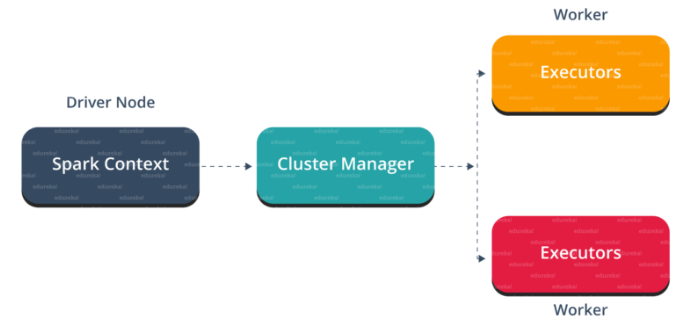
 **Fig: Features of Spark**

1. Speed  
   Spark runs up to 100 times faster than Hadoop MapReduce for large-scale data processing. It is also able to achieve this speed through controlled partitioning.
2. Powerful Caching  
   Simple programming layer provides powerful caching and disk persistence capabilities.
3. Deployment  
   It can be deployed through *Mesos, Hadoop via YARN, or Spark’s own cluster manager.*
4. Real-Time  
   It offers Real-time computation & low latency because of *in-memory computation.*
5. Polyglot  
   Spark provides high-level APIs in Java, Scala, Python, and R. Spark code can be written in any of these four languages. It also provides a shell in Scala and Python.

## Spark Architecture Overview

Apache Spark has a well-defined layered architecture where all the spark components and layers are loosely coupled. This architecture is further integrated with various extensions and libraries. Apache Spark Architecture is based on two main abstractions:

* *Resilient Distributed Dataset (RDD)*
* *Directed Acyclic Graph (DAG)*

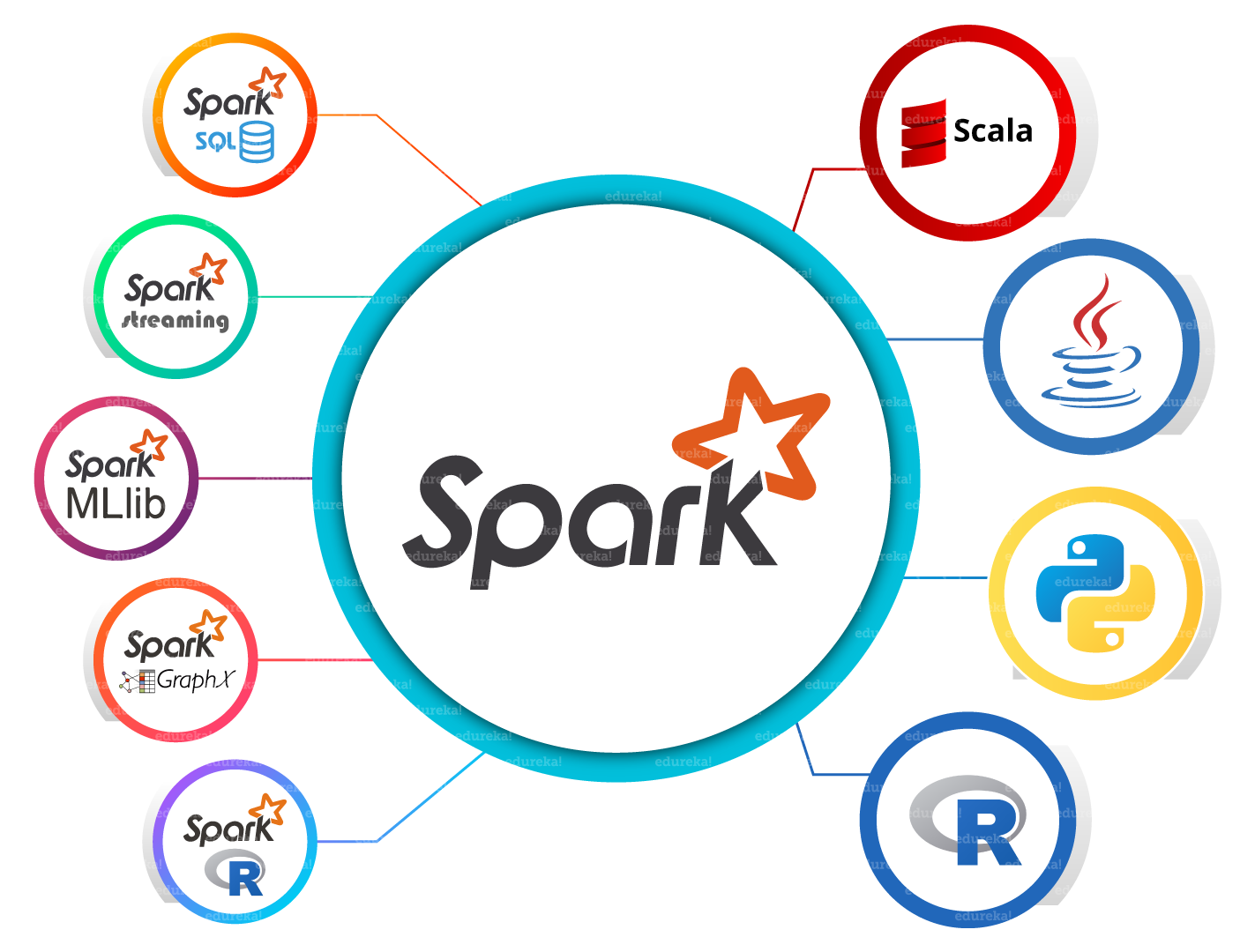
** Fig: Spark Architecture

But before diving any deeper into the Spark architecture, let me explain few fundamental concepts of Spark like Spark Eco-system and RDD. This will help you in gaining better insights.

Let me first explain what is Spark Eco-System.

## Spark Eco-System

As you can see from the below image, the spark ecosystem is composed of various components like Spark SQL, Spark Streaming, MLlib, GraphX, and the Core API component.

 Fig: Spark Eco-System

1. Spark Core  
   Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems.
2. Spark Streaming  
   Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams.
3. Spark SQL  
   Spark SQL is a new module in Spark which integrates relational processing with Spark’s functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.
4. GraphX  
   GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph (a directed multigraph with properties attached to each vertex and edge).
5. MLlib (Machine Learning)  
   MLlib stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.
6. *SparkR*It is an R package that provides a distributed data frame implementation. It also supports operations like selection, filtering, aggregation but on large data-sets.

As you can see, Spark comes packed with high-level libraries, including support for R, SQL, Python, Scala, Java etc. These standard libraries increase the seamless integrations in a complex workflow. Over this, it also allows various sets of services to integrate with it like MLlib, GraphX, SQL + Data Frames, Streaming services etc. to increase its capabilities.

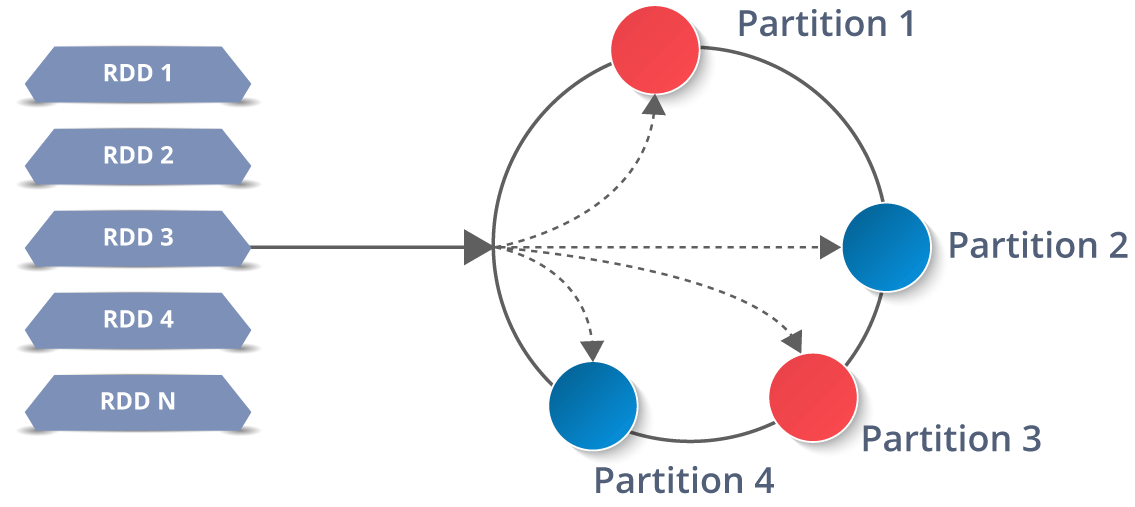
Now, let’s discuss the fundamental Data Structure of Spark, i.e. RDD.

#### Subscribe to our YouTube channel to get new updates...

## Resilient Distributed Dataset(RDD)

RDDs are the building blocks of any Spark application. RDDs Stands for:

* *Resilient:* Fault tolerant and is capable of rebuilding data on failure
* *Distributed:* Distributed data among the multiple nodes in a cluster
* *Dataset:* Collection of partitioned data with values



It is a layer of abstracted data over the distributed collection. It is immutable in nature and follows [*lazy transformations*](https://www.edureka.co/blog/spark-tutorial/#Spark_Features).

Now you might be wondering about its working. Well, the data in an RDD is split into chunks based on a key. RDDs are highly resilient, i.e, they are able to recover quickly from any issues as the same data chunks are replicated across multiple executor nodes. Thus, even if one executor node fails, another will still process the data. This allows you to perform your functional calculations against your dataset very quickly by harnessing the power of multiple nodes.

Moreover, once you create an RDD it becomes *immutable*. By immutable I mean, an object whose state cannot be modified after it is created, but they can surely be transformed.

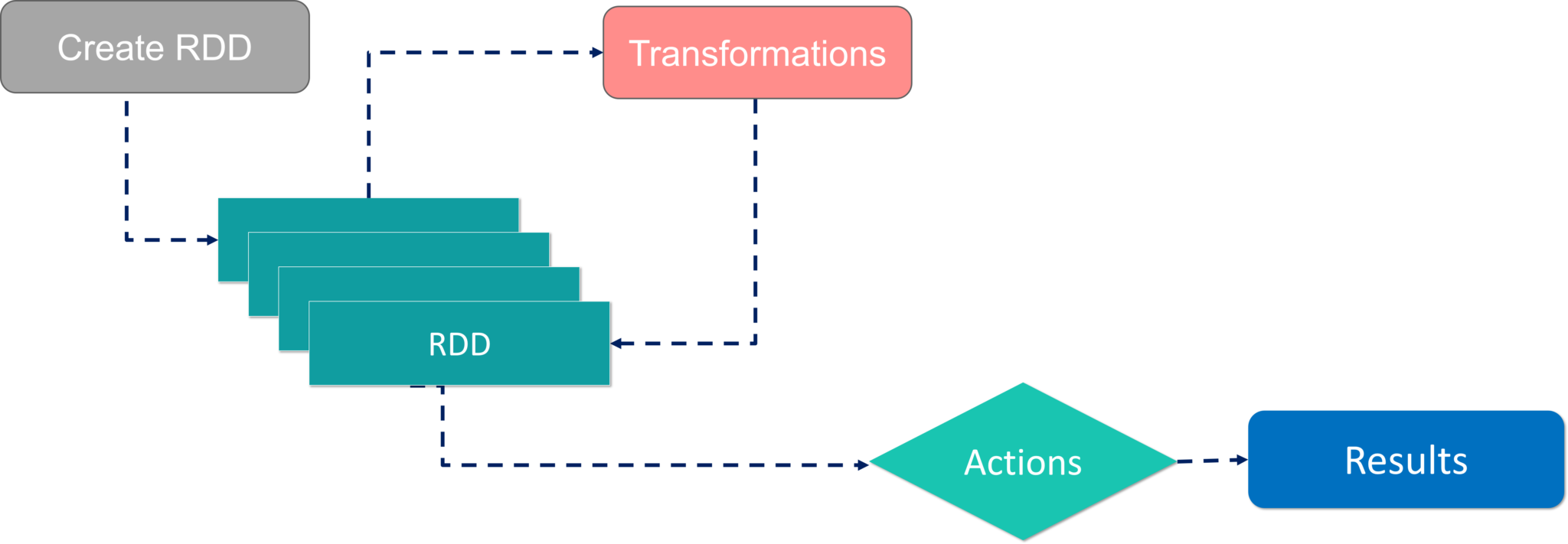


### **[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-certification-training)**

* *[Instructor-led Sessions](https://www.edureka.co/apache-spark-scala-certification-training)*
* *[Real-life Case Studies](https://www.edureka.co/apache-spark-scala-certification-training)*
* *[Assessments](https://www.edureka.co/apache-spark-scala-certification-training)*
* *[Lifetime Access](https://www.edureka.co/apache-spark-scala-certification-training)*

[Explore Curriculum](https://www.edureka.co/apache-spark-scala-certification-training)

Talking about the distributed environment, each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. Due to this, you can perform transformations or actions on the complete data parallelly. Also, you don’t have to worry about the distribution, because Spark takes care of that.

 Workflow of RDD

There are two ways to create RDDs − parallelizing an existing collection in your driver program, or by referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, etc.

With RDDs, you can perform two types of operations:

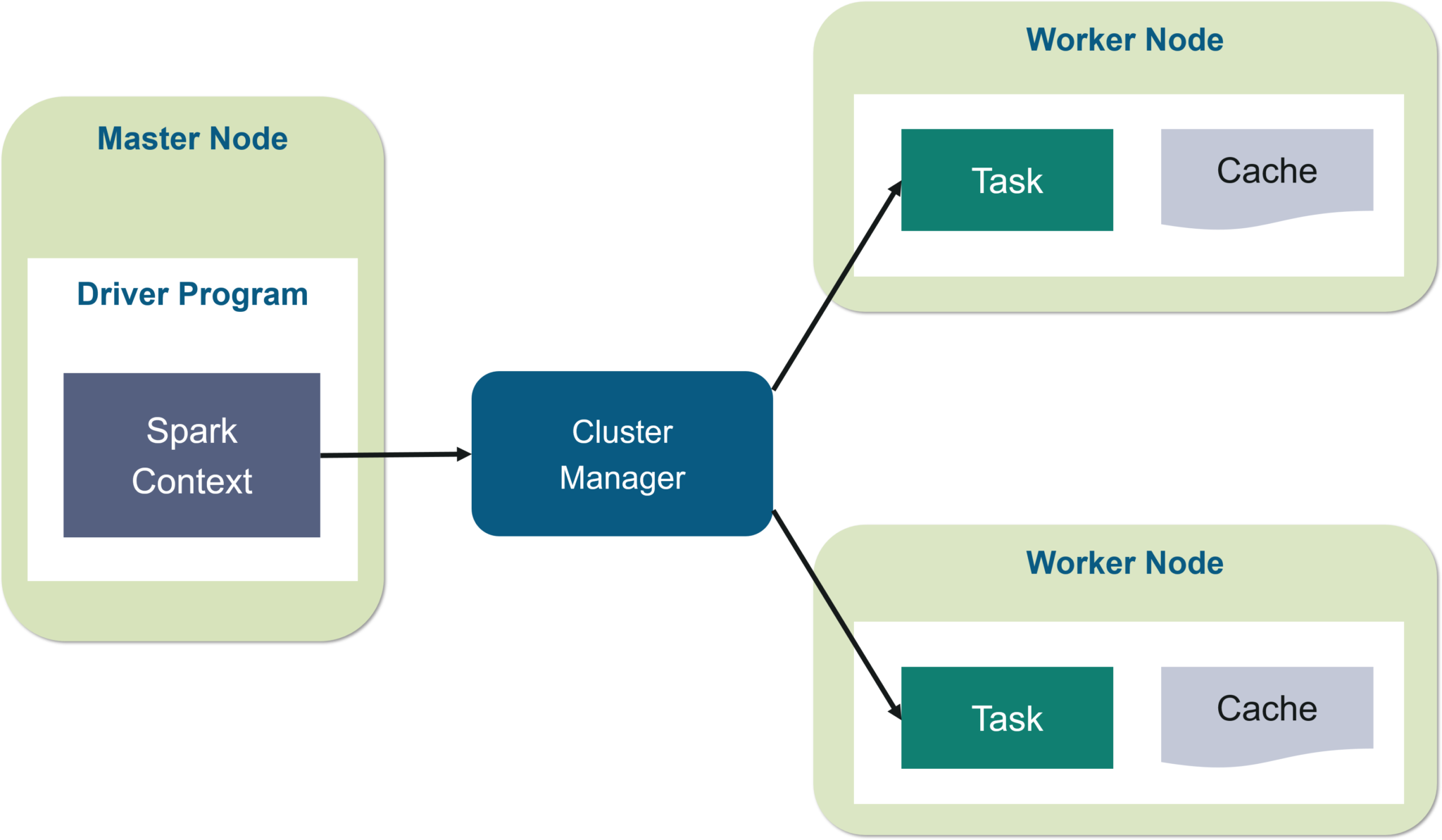
1. Transformations: They are the operations that are applied to create a new RDD.
2. Actions: They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

I hope you got a thorough understanding of RDD concepts. Now let’s move further and see the working of Spark Architecture.

## Working of Spark Architecture

As you have already seen the basic architectural overview of Apache Spark, now let’s dive deeper into its working.

In your master node, you have the *driver program*, which drives your application. The code you are writing behaves as a driver program or if you are using the interactive shell, the shell acts as the driver program.

 Fig: Spark Architecture

Inside the driver program, the first thing you do is, you *create* a *Spark Context.* Assume that the Spark context is a gateway to all the Spark functionalities. It is similar to your database connection. Any command you execute in your database goes through the database connection. Likewise, anything you do on Spark goes through Spark context.

Now, this Spark context works with the *cluster manager* to manage various jobs. The driver program & Spark context takes care of the job execution within the cluster. A job is split into multiple tasks which are distributed over the worker node. Anytime an RDD is created in Spark context, it can be distributed across various nodes and can be cached there.

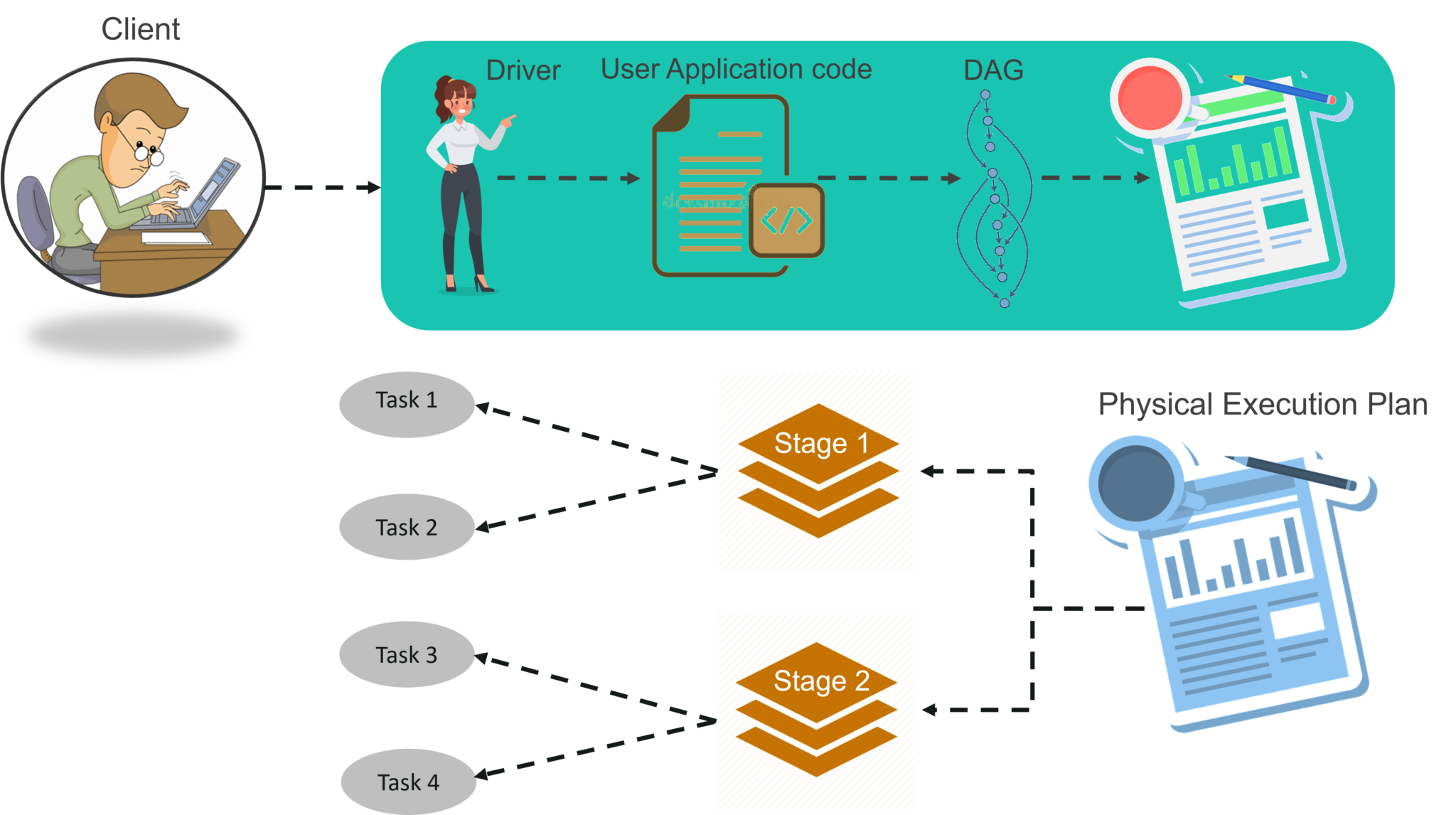
W*orker nodes* are the slave nodes whose job is to basically execute the tasks. These tasks are then executed on the partitioned RDDs in the worker node and hence returns back the result to the Spark Context.

Spark Context takes the job, breaks the job in tasks and distribute them to the worker nodes. These tasks work on the partitioned RDD, perform operations, collect the results and return to the main Spark Context.

If you increase the number of workers, then you can divide jobs into more partitions and execute them parallelly over multiple systems. It will be a lot faster.

With the increase in the number of workers, memory size will also increase & you can cache the jobs to execute it faster.

To know about the workflow of Spark Architecture, you can have a look at the infographic below:

 Fig: Spark Architecture Infographic

STEP 1: The client submits spark user application code. When an application code is submitted, the driver implicitly converts user code that contains transformations and actions into a logically *directed acyclic graph* called *DAG.* At this stage, it also performs optimizations such as pipelining transformations.

### **Big Data Training**

**[BIG DATA HADOOP CERTIFICATION TRAINING](https://www.edureka.co/big-data-hadoop-training-certification)**

### **[Big Data Hadoop Certification Training](https://www.edureka.co/big-data-hadoop-training-certification)**

*[Reviews](https://www.edureka.co/big-data-hadoop-training-certification)*

[5(155709)](https://www.edureka.co/big-data-hadoop-training-certification)

**[PYTHON SPARK CERTIFICATION TRAINING USING PYSPARK](https://www.edureka.co/pyspark-certification-training)**

### **[Python Spark Certification Training using PySpark](https://www.edureka.co/pyspark-certification-training)**

*[Reviews](https://www.edureka.co/pyspark-certification-training)*

[5(4541)](https://www.edureka.co/pyspark-certification-training)

**[APACHE SPARK AND SCALA CERTIFICATION TRAINING](https://www.edureka.co/apache-spark-scala-certification-training)**

### **[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-certification-training)**

*[Reviews](https://www.edureka.co/apache-spark-scala-certification-training)*

[5(25935)](https://www.edureka.co/apache-spark-scala-certification-training)

**[SPLUNK TRAINING & CERTIFICATION- POWER USER & ADMIN](https://www.edureka.co/splunk-certification-training)**

### **[Splunk Training & Certification- Power User & Admin](https://www.edureka.co/splunk-certification-training)**

*[Reviews](https://www.edureka.co/splunk-certification-training)*

[5(7118)](https://www.edureka.co/splunk-certification-training)

**[APACHE KAFKA CERTIFICATION TRAINING](https://www.edureka.co/kafka-certification-training)**

### **[Apache Kafka Certification Training](https://www.edureka.co/kafka-certification-training)**

*[Reviews](https://www.edureka.co/kafka-certification-training)*

[5(5846)](https://www.edureka.co/kafka-certification-training)

**[HADOOP ADMINISTRATION CERTIFICATION TRAINING](https://www.edureka.co/hadoop-administration-training-certification)**

### **[Hadoop Administration Certification Training](https://www.edureka.co/hadoop-administration-training-certification)**

*[Reviews](https://www.edureka.co/hadoop-administration-training-certification)*

[5(24766)](https://www.edureka.co/hadoop-administration-training-certification)

**[ELK STACK TRAINING & CERTIFICATION](https://www.edureka.co/elk-stack-training-sp)**

### **[ELK Stack Training & Certification](https://www.edureka.co/elk-stack-training-sp)**

*[Reviews](https://www.edureka.co/elk-stack-training-sp)*

[5(1158)](https://www.edureka.co/elk-stack-training-sp)

**[COMPREHENSIVE HIVE CERTIFICATION TRAINING](https://www.edureka.co/comprehensive-hive)**

### **[Comprehensive Hive Certification Training](https://www.edureka.co/comprehensive-hive)**

*[Reviews](https://www.edureka.co/comprehensive-hive)*

[5(2132)](https://www.edureka.co/comprehensive-hive)

**[APACHE STORM CERTIFICATION TRAINING](https://www.edureka.co/apache-storm-certification-training)**

### **[Apache Storm Certification Training](https://www.edureka.co/apache-storm-certification-training)**

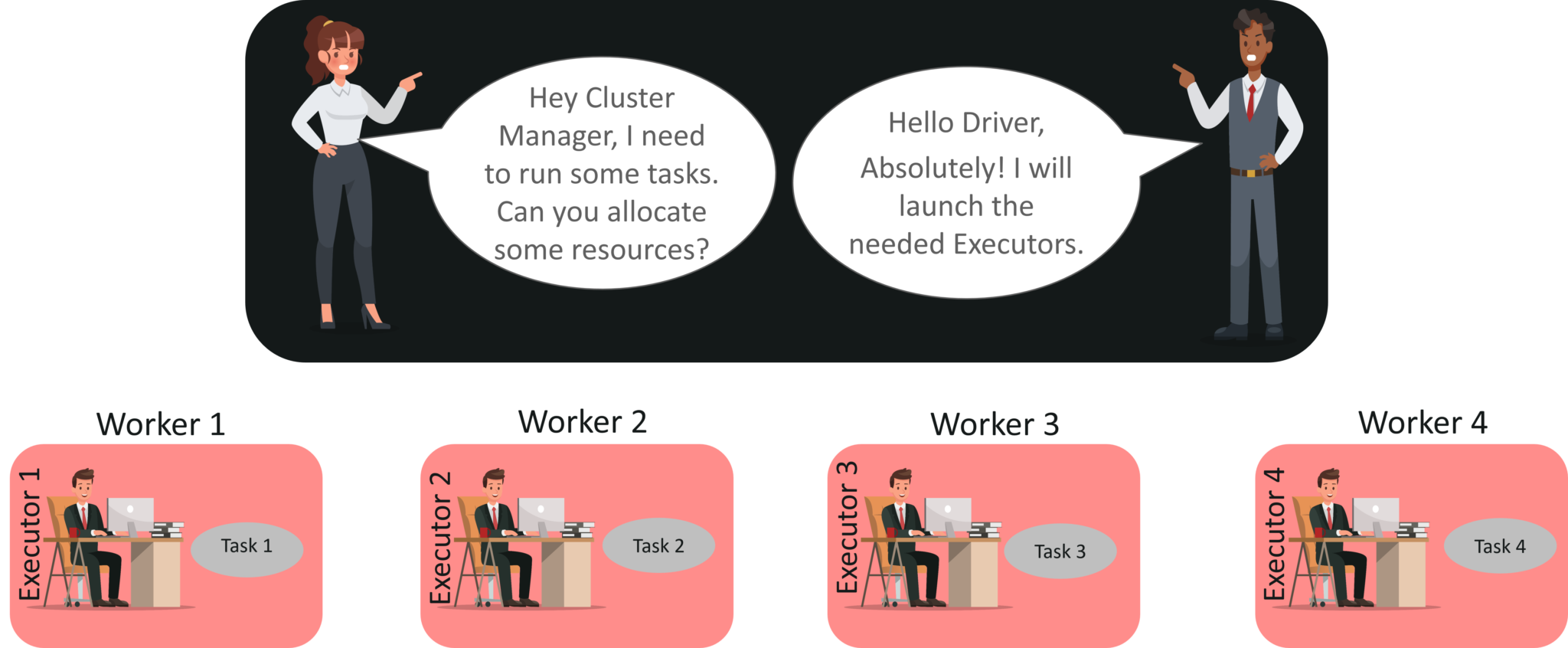
*[Reviews](https://www.edureka.co/apache-storm-certification-training)*

[5(5514)](https://www.edureka.co/apache-storm-certification-training)

Next

STEP 2: After that, it converts the logical graph called DAG into physical execution plan with many stages. After converting into a physical execution plan, it creates physical execution units called tasks under each stage. Then the tasks are bundled and sent to the cluster.

STEP 3: Now the driver talks to the cluster manager and negotiates the resources. Cluster manager launches executors in worker nodes on behalf of the driver. At this point, the driver will send the tasks to the executors based on data placement. When executors start, they register themselves with drivers. So, the driver will have a complete view of executors that are executing the task.

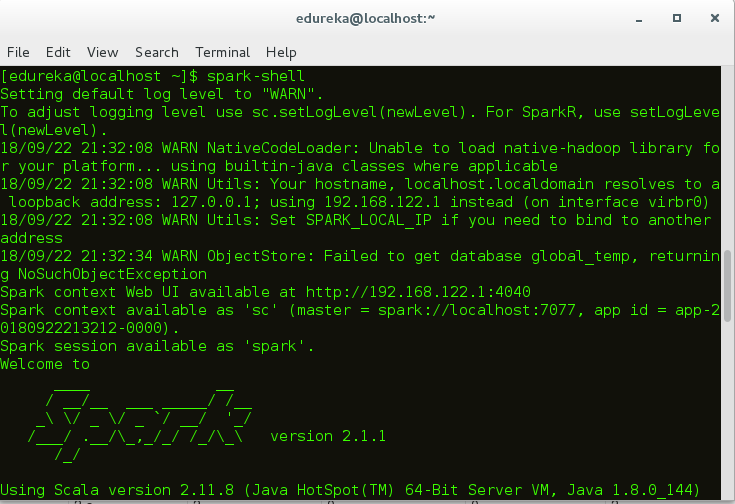


STEP 4: During the course of execution of tasks, driver program will monitor the set of executors that runs. Driver node also schedules future tasks based on data placement.

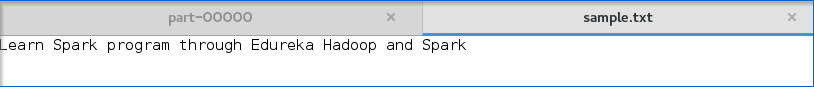
This was all about Spark Architecture. Now, let’s get a hand’s on the working of a Spark shell.

## Example using Scala in Spark shell

At first, let’s start the Spark shell by assuming that Hadoop and Spark daemons are up and running. *Web UI* port for Spark is *localhost:4040.*

** Fig: Spark-shell

Once you have started the Spark shell, now let’s see how to execute a word count example:

1. In this case, I have created a simple text file and stored it in the hdfs directory. You can also use other large data files as well.  
    Fig: Input text file
2. Once the spark shell has started, let’s create an RDD. For this, you have to specify the input file path and apply the transformation flatMap(). Below code illustrates the same:

|  |  |
| --- | --- |
| 1 | scala> var map = sc.textFile("hdfs://localhost:9000/Example/sample.txt").flatMap(line => line.split(" ")).map(word => (word,1)); |

3. On executing this code, an RDD will be created as shown in the figure.

 Fig: RDD creation

4. After that, you need to apply the action reduceByKey() to the created RDD.

|  |  |
| --- | --- |
| 1 | scala> var counts = map.reduceByKey(\_+\_); |

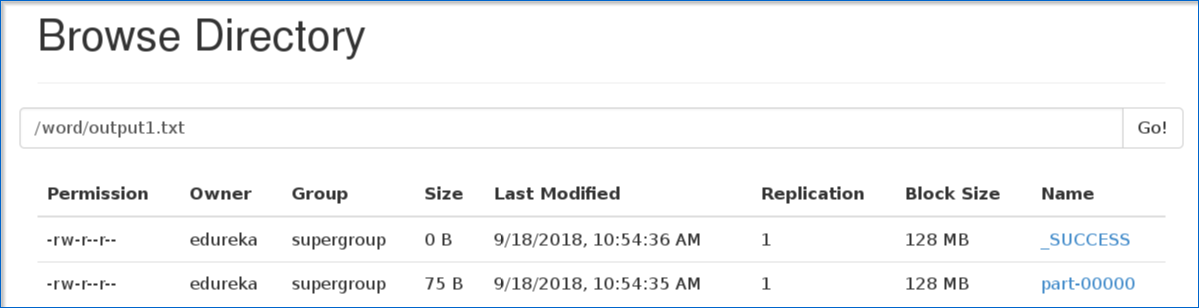
After applying action, execution starts as shown below.

 Fig: Spark execution in the shell

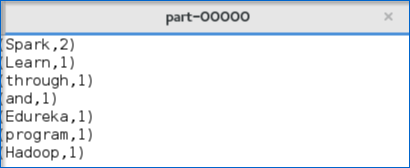
5. Next step is to save the output in a text file and specify the path to store the output.

 Fig: Specifying the Output path

6. After specifying the output path, go to the *hdfs web browser localhost:50040.* Here you can see the output text in the ‘part’ file as shown below.

 Fig: Output part file

7. Below figure shows the output text present in the ‘part’ file.

 Fig: Output text

I hope that you have understood how to create a Spark Application and arrive at the output.

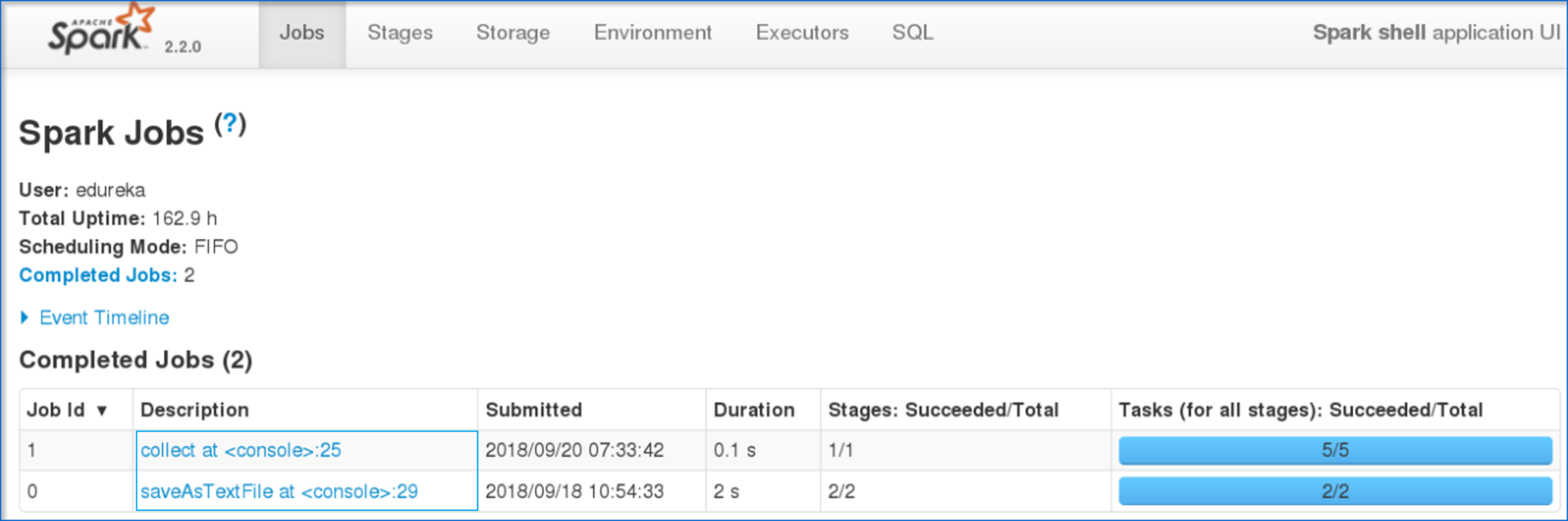


### **[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-certification-training)**

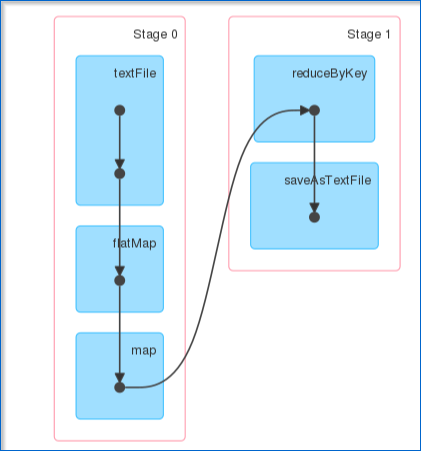
[Weekday / Weekend Batches](https://www.edureka.co/apache-spark-scala-certification-training)

[See Batch Details](https://www.edureka.co/apache-spark-scala-certification-training)

Now, let me take you through the web UI of Spark to understand the DAG visualizations and partitions of the executed task.

 Fig: Spark Web User Interface

* On clicking the task that you have submitted, you can view the Directed Acyclic Graph (DAG) of the completed job.

 Fig: DAG Visualization

* Also, you can view the summary metrics of the executed task like – time taken to execute the task, job ID, completed stages, host IP Address etc.

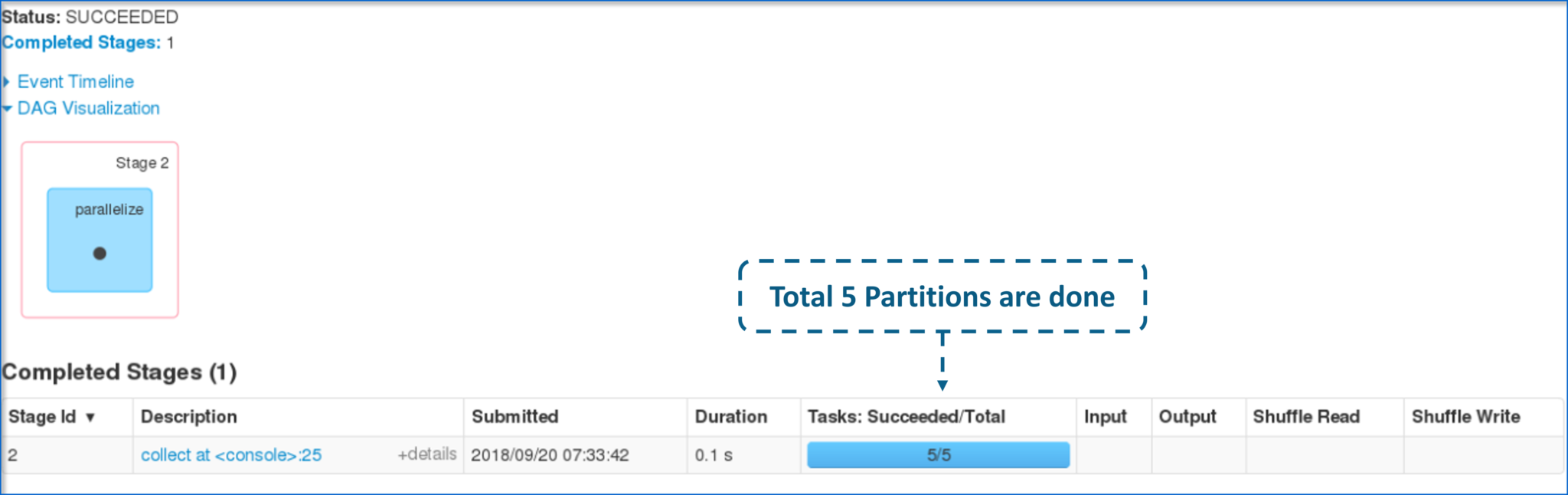
Now, let’s understand about partitions and parallelism in RDDs.

* A *partition* is a *logical* *chunk* of a *large* *distributed* *data* *set.*
* By default, Spark tries to *read* *data* *into* *an* *RDD* from the *nodes* that are *close* *to* *it.*

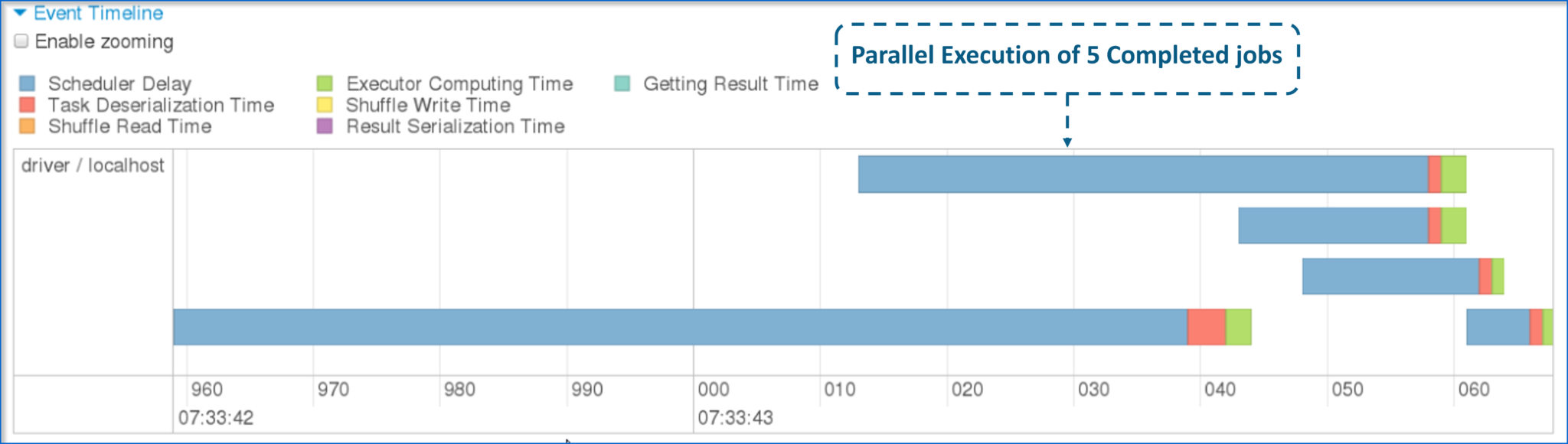
Now, let’s see how to execute a parallel task in the shell.

 Fig: Parallel Execution of a task

* Below figure shows the total number of partitions on the created RDD.

 Fig: Partitions of the completed task

* Now, let me show you how parallel execution of 5 different tasks appears.

 Fig: Parallelism of the 5 completed tasks

This brings us to the end of the blog on Apache Spark Architecture. I hope this blog was informative and added value to your knowledge.

*If you wish to learn Spark and build a career in domain of Spark to perform large-scale Data Processing using RDD, Spark Streaming, SparkSQL, MLlib, GraphX and Scala with Real Life use-cases, check out our interactive, live-online* [*Apache Spark Certification Training*](https://www.edureka.co/apache-spark-scala-training) *here, that comes with 24\*7 support to guide you throughout your learning period.*

*Got a question for us? Please mention it in the comments section of “Spark Architecture” article and we wil*

What is a data pipeline?

A data pipeline is a software that consolidates data from multiple sources and makes it available to be used strategically.

The data pipeline architecture consists of several layers:-

1) Data Ingestion

2) Data Collector

3) Data Processing

4) Data Storage

5) Data Query

6) Data Visualization

Let’s get into details of each layer & understand how we can build a real-time data pipeline.

##### 1) Data Ingestion

Data ingestion is the first step in [building a data pipeline](https://www.perfomatix.com/blog/kafka-used-building-real-time-data-analytics/). Data Ingestion helps you to bring data into the pipeline. It means taking unstructured data from where it is originated into a data processing system where it can be stored & analyzed for making data-driven business decisions.

Data Ingestion process to be effective needs, to begin with prioritizing data sources, validating individual files & routing data streams to the correct destination. It should be well designed to handle and upgrade the new data sources, technology and applications. It should also allow rapid consumption of data.

You can use Open source Data Ingestion Tools like Apache Flume. Apache Flume is a reliable distributed service for efficiently collecting, aggregating, and moving large amounts of log data.

Its functions are –

1. Stream the Data — Ingest streaming data from multiple sources into Hadoop for storage and analysis.
2. Insulate the System — Buffer storage platform from transitory spikes, when the rate of incoming data surpasses the rate at which data is written to the destination.
3. Scale Horizontally — Ingest new data streams & additional volume as needed.

You can also use Apache NiFi or elastic Logstash. All of them can ingest Data of all Shapes, Sizes, and Sources.

##### 2) Data Collector

In Data collector layer, the focus is on the transportation of data from ingestion layer to rest of data pipeline. We use a messaging system called Apache Kafka to act as a mediator between all the programs that can send and receive messages.

Apache Kafka can process streams of data in real-time and store streams of data safely in a distributed replicated cluster.

Kafka works along with Apache Storm, Apache HBase and Apache Spark for real-time analysis and rendering of streaming data.

There are four components involved in moving the data in and out of Apache Kafka –

1. Topics — Topic is a user-defined category to which messages are published.
2. Producers — Producers report messages to one or more topics
3. Consumers — Consumers subscribe to topics & process the reported messages.
4. Brokers — Brokers manage the persistence & replication of message data.

##### 3) Data Processing

In this layer, the main focus is to process the collected data from the previous layer. Layer helps to route the data to a different destination, classify the data flow, and it’s the first point where the analytics takes place.

You can use Apache Spark for the real-time data processing as it is a fast, in-memory data processing engine. It can run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells. Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

It’s elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets.

You can use other platforms like Apache storm, Apache Flink depending on your particular use case.

##### 4) Data Storage

This layer ensures to keep data in the right place based on usage. A Relational Database is a place you may have stored our data over the years, but with the new big data enterprise applications, you should no longer assume that your persistence should be relational.

You need different databases to handle the different variety of data, but using different databases creates overhead issues.

You can use Polyglot persistence to use multiple databases to power a single application. Advantages of Polygon Persistence are faster response times, it helps your data to scale well and gives you a rich experience.

Tools used for data storage can HDFS, GFS, Amazon S3.

##### 5) Data Query

This layer is where strong analytic processing takes place. Analytics Query Tools available are Apache Hive, Spark SQL, Amazon Redshift, Presto.

Apache Hive is data warehouse built on top of Apache Hadoop for providing data summarization, ad-hoc query, and analysis of large datasets. Data analysts use Hive to query, summarize, explore and analyze the data, then turn it into actionable business insight.

Apache Hive helps to project structure onto the data in Hadoop and to query that data using a SQL.

You can also use Spark SQL for data query. Spark SQL is a Spark module for structured data processing.

Presto is an open-source distributed SQL query engine used to run interactive analytic queries against data sources of all sizes.

##### 6) Data Visualization

This layer focus on Big Data Visualization. You need something that grabs people’s attention, pull them in & make your findings well-understood. Data Visualization layer provides full Business Infographics.

Based on your business requirements, you can create Custom dashboards, Real-Time Dashboards using data visualization tools in the market.

Tableau is one of the best data visualization tool available in the market today with a Drag and Drop functionality. Tableau allows the users to design Charts, Maps, Tabular, Matrix reports, Stories and Dashboards without having any technical knowledge. It helps you to quickly analyze, visualize and share information whether it’s structured or unstructured, petabytes or terabytes has millions or billions of rows, you can turn big data into big ideas.

You can use Kibana dashboard. A Kibana dashboard displays a collection of pre-saved visualizations. You can arrange and resize the visualizations as need and save dashboards, and they can be reloaded and shared.

You can use Intelligent agents,Angular.js,React.js & Recommender systems as well for Data Visualization.

##### Summary

Building a data pipeline is a long & tedious process, and you require lots of technical expertise & experience to create one layer by layer.