**Top Apache Spark Interview Questions You Should Prepare In 2019**

2018 has been the year of Big Data – the year when big data and analytics made tremendous progress through innovative technologies, data-driven decision making and outcome-centric analytics. Worldwide revenues for big data and business analytics (BDA) will grow from $130.1 billion in 2016 to more than $203 billion in 2020 (source IDC). Prepare with these top **Apache Spark Interview Questions** to get an edge in the burgeoning Big Data market where global and local enterprises, big or small, are looking for a quality [***Big Data and Hadoop experts***](https://www.edureka.co/apache-spark-scala-training). Before moving ahead, you can check out [***this skill report***](https://www.edureka.co/skill-report)which talks about the top technical skills to master in 2019.

As a big data professional, it is essential to know the right buzzwords, learn the right technologies and prepare the right answers to commonly asked Spark interview questions. With questions and answers around *Spark Core*, *Spark Streaming*, *Spark SQL*, *GraphX*, *MLlib* among others, this blog is your gateway to your next Spark job.

**Apache Spark Interview Questions And Answers**

**1. Compare Hadoop and Spark.**

We will compare Hadoop MapReduce and Spark based on the following aspects:

|  |  |  |
| --- | --- | --- |
| **Apache Spark vs. Hadoop** | | |
| **Feature Criteria** | **Apache Spark** | **Hadoop** |
| **Speed** | 100 times faster than Hadoop | Decent speed |
| **Processing** | Real-time & Batch processing | Batch processing only |
| **Difficulty** | Easy because of high level modules | Tough to learn |
| **Recovery** | Allows recovery of partitions | Fault-tolerant |
| **Interactivity** | Has interactive modes | No interactive mode except Pig & Hive |

**Table:** *Apache Spark versus Hadoop*

*Let us understand the same using an interesting analogy.*

*“Single cook cooking an entree is regular computing. Hadoop is multiple cooks cooking an entree into pieces and letting each cook her piece.  
Each cook has a separate stove and a food shelf. The first cook cooks the meat, the second cook cooks the sauce. This phase is called “Map”. A the end the main cook assembles the complete entree. This is called “Reduce”. For Hadoop, the cooks are not allowed to keep things on the stove between operations. Each time you make a particular operation, the cook puts results on the shelf. This slows things down.  
For Spark, the cooks are allowed to keep things on the stove between operations. This speeds things up. Finally, for Hadoop the recipes are written in a language which is illogical and hard to understand. For Spark, the recipes are nicely written.” – Stan Kladko, Galactic Exchange.io*

**2. What is Apache Spark?**

* [***Apache Spark***](https://www.edureka.co/blog/spark-tutorial/) is an open-source cluster computing framework for real-time processing.
* It has a thriving open-source community and is the most active Apache project at the moment.
* Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

Spark is of the most successful projects in the Apache Software Foundation. Spark has clearly evolved as the market leader for Big Data processing. Many organizations run Spark on clusters with thousands of nodes. Today, Spark is being adopted by major players like Amazon, eBay, and Yahoo!

[**GET STARTED WITH SPARK**](https://www.edureka.co/apache-spark-scala-training)

**3. Explain the key features of Apache Spark.**

The following are the key features of Apache Spark:

1. **Polyglot**
2. **Speed**
3. **Multiple Format Support**
4. **Lazy Evaluation**
5. **Real Time Computation**
6. **Hadoop Integration**
7. **Machine Learning**

Let us look at these features in detail:

1. **Polyglot**: Spark provides high-level APIs in Java, Scala, Python and R. Spark code can be written in any of these four languages. It provides a shell in Scala and Python. The Scala shell can be accessed through **./bin/spark-shell** and Python shell through **./bin/pyspark** from the installed directory.
2. **Speed**: Spark runs upto 100 times faster than Hadoop MapReduce for large-scale data processing. Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.
3. **Multiple Formats**: Spark supports multiple data sources such as Parquet, JSON, Hive and Cassandra. The Data Sources API provides a pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.
4. **Lazy Evaluation**: Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For transformations, Spark adds them to a DAG of computation and only when the driver requests some data, does this DAG actually gets executed.
5. **Real Time Computation**: Spark’s computation is real-time and has less latency because of its in-memory computation. Spark is designed for massive scalability and the Spark team has documented users of the system running production clusters with thousands of nodes and supports several computational models.
6. **Hadoop Integration**: Apache Spark provides smooth compatibility with Hadoop. This is a great boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.
7. **Machine Learning**: Spark’s MLlib is the machine learning component which is handy when it comes to big data processing. It eradicates the need to use multiple tools, one for processing and one for machine learning. Spark provides data engineers and data scientists with a powerful, unified engine that is both fast and easy to use.

**4. What are the languages supported by Apache Spark and which is the most popular one?**

Apache Spark supports the following four languages: Scala, Java, Python and R. Among these languages, Scala and Python have interactive shells for Spark. The Scala shell can be accessed through **./bin/spark-shell**and the Python shell through **./bin/pyspark**. Scala is the most used among them because Spark is written in Scala and it is the most popularly used for Spark.

**5. What are benefits of Spark over MapReduce?**

Spark has the following benefits over MapReduce:

1. Due to the availability of in-memory processing, Spark implements the processing around 10 to 100 times faster than Hadoop MapReduce whereas MapReduce makes use of persistence storage for any of the data processing tasks.
2. Unlike Hadoop, Spark provides inbuilt libraries to perform multiple tasks from the same core like batch processing, Steaming, Machine learning, Interactive SQL queries. However, Hadoop only supports batch processing.
3. Hadoop is highly disk-dependent whereas Spark promotes caching and in-memory data storage.
4. Spark is capable of performing computations multiple times on the same dataset. This is called iterative computation while there is no iterative computing implemented by Hadoop.

**6. What is YARN?**

Similar to Hadoop, YARN is one of the key features in Spark, providing a central and resource management platform to deliver scalable operations across the cluster. YARN is a distributed container manager, like Mesos for example, whereas Spark is a data processing tool. Spark can run on YARN, the same way Hadoop Map Reduce can run on YARN. Running Spark on YARN necessitates a binary distribution of Spark as built on YARN support.

**7. Do you need to install Spark on all nodes of YARN cluster?**

No, because Spark runs on top of YARN. Spark runs independently from its installation. Spark has some options to use YARN when dispatching jobs to the cluster, rather than its own built-in manager, or Mesos. Further, there are some configurations to run YARN. They include *master*, *deploy-mode*, *driver-memory*, *executor-memory*, *executor-cores*, and *queue*.

**8. Is there any benefit of learning MapReduce if Spark is better than MapReduce?**

Yes, MapReduce is a paradigm used by many big data tools including Spark as well. It is extremely relevant to use MapReduce when the data grows bigger and bigger. Most tools like Pig and Hive convert their queries into MapReduce phases to optimize them better.

**9. Explain the concept of Resilient Distributed Dataset (RDD).**

RDD stands for Resilient Distribution Datasets. An RDD is a fault-tolerant collection of operational elements that run in parallel. The partitioned data in RDD is immutable and distributed in nature. There are primarily two types of RDD:

1. Parallelized Collections: Here, the existing RDDs running parallel with one another.
2. Hadoop Datasets: They perform functions on each file record in HDFS or other storage systems.

RDDs are basically parts of data that are stored in the memory distributed across many nodes. RDDs are lazily evaluated in Spark. This lazy evaluation is what contributes to Spark’s speed.

**10. How do we create RDDs in Spark?**

Spark provides two methods to create RDD:

1. By parallelizing a collection in your Driver program.

2. This makes use of SparkContext’s ‘parallelize’

|  |  |
| --- | --- |
| 1  2  3 | method val DataArray = Array(2,4,6,8,10)    val DataRDD = sc.parallelize(DataArray) |

3. By loading an external dataset from external storage like HDFS, HBase, shared file system.

**11. What is Executor Memory in a Spark application?**

Every spark application has same fixed heap size and fixed number of cores for a spark executor. The heap size is what referred to as the Spark executor memory which is controlled with the spark.executor.memory property of the ***–executor-memory*** flag. Every spark application will have one executor on each worker node. The executor memory is basically a measure on how much memory of the worker node will the application utilize.

**12. Define Partitions in Apache Spark.**

As the name suggests, partition is a smaller and logical division of data similar to ‘split’ in MapReduce. It is a logical chunk of a large distributed data set. Partitioning is the process to derive logical units of data to speed up the processing process. Spark manages data using partitions that help parallelize distributed data processing with minimal network traffic for sending data between executors. By default, Spark tries to read data into an RDD from the nodes that are close to it. Since Spark usually accesses distributed partitioned data, to optimize transformation operations it creates partitions to hold the data chunks. Everything in Spark is a partitioned RDD.

**13. What operations does RDD support?**

RDD (Resilient Distributed Dataset) is main logical data unit in Spark. An RDD has distributed a collection of objects. Distributed means, each RDD is divided into multiple partitions. Each of these partitions can reside in memory or stored on the disk of different machines in a cluster. RDDs are immutable (Read Only) data structure. You can’t change original RDD, but you can always transform it into different RDD with all changes you want.

RDDs support two types of operations: transformations and actions.

*Transformations*: Transformations create new RDD from existing RDD like map, reduceByKey and filter we just saw. Transformations are executed on demand. That means they are computed lazily.

*Actions*: Actions return final results of RDD computations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.

**[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-training" \t "_blank)**[Watch The Course Preview](https://www.edureka.co/apache-spark-scala-training" \t "_blank)

**14. What do you understand by Transformations in Spark?**

Transformations are functions applied on RDD, resulting into another RDD. It does not execute until an action occurs. map() and filter() are examples of transformations, where the former applies the function passed to it on each element of RDD and results into another RDD. The filter() creates a new RDD by selecting elements from current RDD that pass function argument.

|  |  |
| --- | --- |
| 1  2  3 | val rawData=sc.textFile("path to/movies.txt")    val moviesData=rawData.map(x=&gt;x.split("\t")) |

As we can see here, *rawData* RDD is transformed into *moviesData* RDD. Transformations are lazily evaluated.

**15. Define Actions in Spark.**

An action helps in bringing back the data from RDD to the local machine. An action’s execution is the result of all previously created transformations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.

*reduce()* is an action that implements the function passed again and again until one value if left. *take()*action takes all the values from RDD to a local node.

|  |  |
| --- | --- |
| 1 | moviesData.saveAsTextFile(“MoviesData.txt”) |

As we can see here, *moviesData*RDD is saved into a text file called *MoviesData.txt*.

**16. Define functions of SparkCore.**

*Spark Core* is the base engine for large-scale parallel and distributed data processing. The core is the distributed execution engine and the Java, Scala, and Python APIs offer a platform for distributed ETL application development. SparkCore performs various important functions like memory management, monitoring jobs, fault-tolerance, job scheduling and interaction with storage systems. Further, additional libraries, built atop the core allow diverse workloads for streaming, SQL, and machine learning. It is responsible for:

1. Memory management and fault recovery
2. Scheduling, distributing and monitoring jobs on a cluster
3. Interacting with storage systems

**17. What do you understand by Pair RDD?**

Apache defines PairRDD functions class as

|  |  |
| --- | --- |
| 1 | class PairRDDFunctions[K, V] extends Logging with HadoopMapReduceUtil with Serializable |

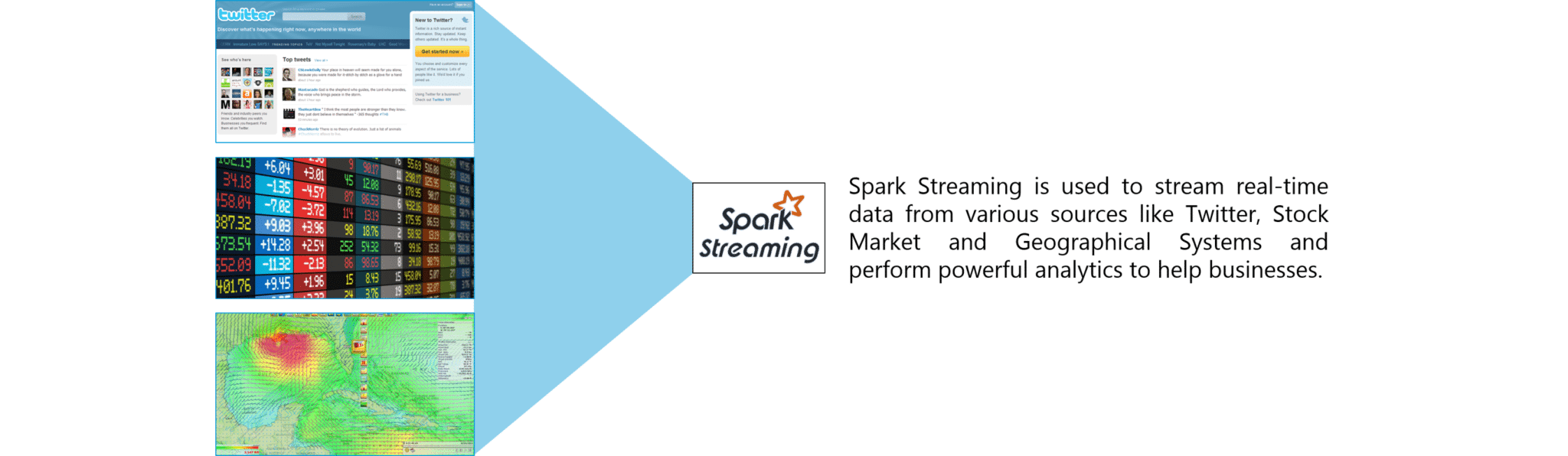
Special operations can be performed on RDDs in Spark using key/value pairs and such RDDs are referred to as Pair RDDs. Pair RDDs allow users to access each key in parallel. They have a *reduceByKey()* method that collects data based on each key and a *join()* method that combines different RDDs together, based on the elements having the same key.

**18. Name the components of Spark Ecosystem.**

1. **Spark Core**: Base engine for large-scale parallel and distributed data processing
2. **Spark Streaming**: Used for processing real-time streaming data
3. **Spark SQL**: Integrates relational processing with Spark’s functional programming API
4. **GraphX**: Graphs and graph-parallel computation
5. **MLlib**: Performs machine learning in Apache Spark

**19. How is Streaming implemented in Spark? Explain with examples.**

*Spark Streaming* is used for processing 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. The fundamental stream unit is DStream which is basically a series of RDDs (Resilient Distributed Datasets) to process the real-time data. The data from different sources like Flume, HDFS is streamed and finally processed to file systems, live dashboards and databases. It is similar to batch processing as the input data is divided into streams like batches.

****

**Figure:** *Spark Interview Questions – Spark Streaming*

**20. Is there an API for implementing graphs in Spark?**

*GraphX* is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph.

The property graph is a directed multi-graph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. Here, the parallel edges allow multiple relationships between the same vertices. 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.

To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

**21. What is PageRank in GraphX?**

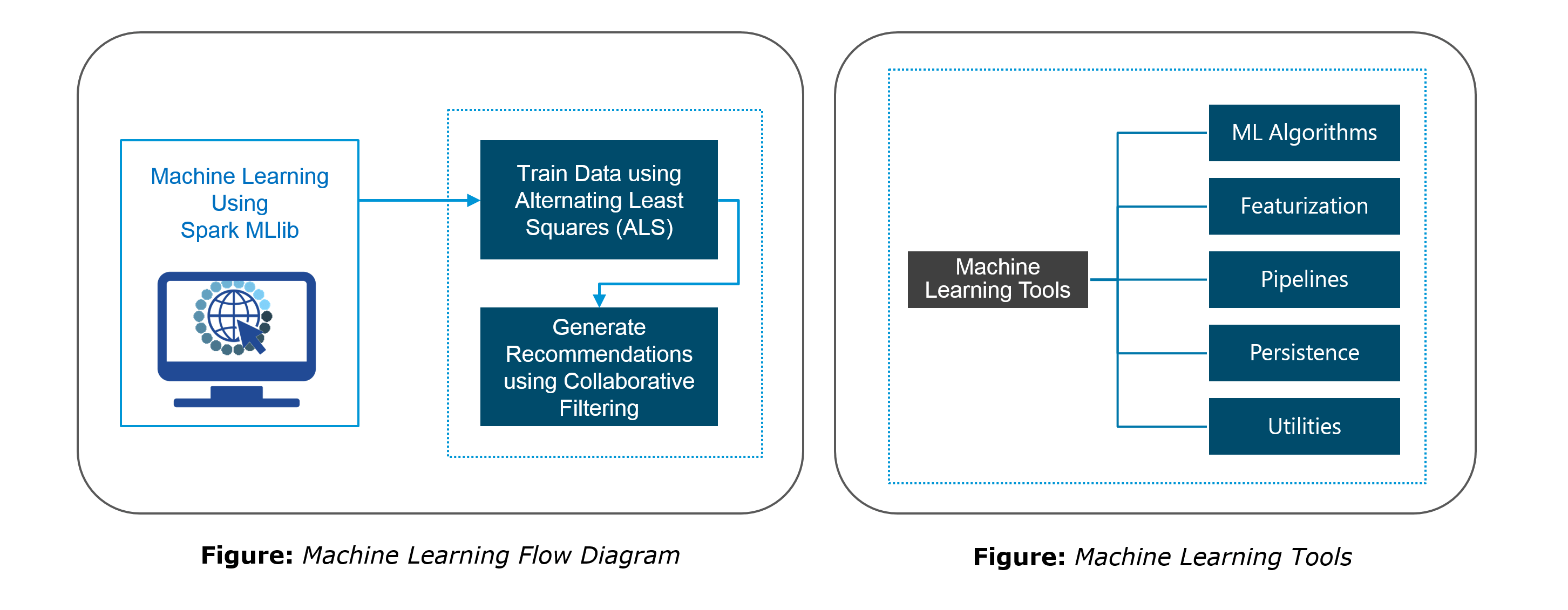
PageRank measures the importance of each vertex in a graph, assuming an edge from *u* to *v* represents an endorsement of *v*’s importance by *u*. For example, if a Twitter user is followed by many others, the user will be ranked highly.

GraphX comes with static and dynamic implementations of PageRank as methods on the PageRank Object. Static PageRank runs for a fixed number of iterations, while dynamic PageRank runs until the ranks converge (i.e., stop changing by more than a specified tolerance). GraphOps allows calling these algorithms directly as methods on Graph.

[**LEARN SPARK FROM EXPERTS**](https://www.edureka.co/apache-spark-scala-training)

**22. How is machine learning implemented in Spark?**

MLlib is scalable machine learning library provided by Spark. It aims at making machine learning easy and scalable with common learning algorithms and use cases like clustering, regression filtering, dimensional reduction, and alike.

****

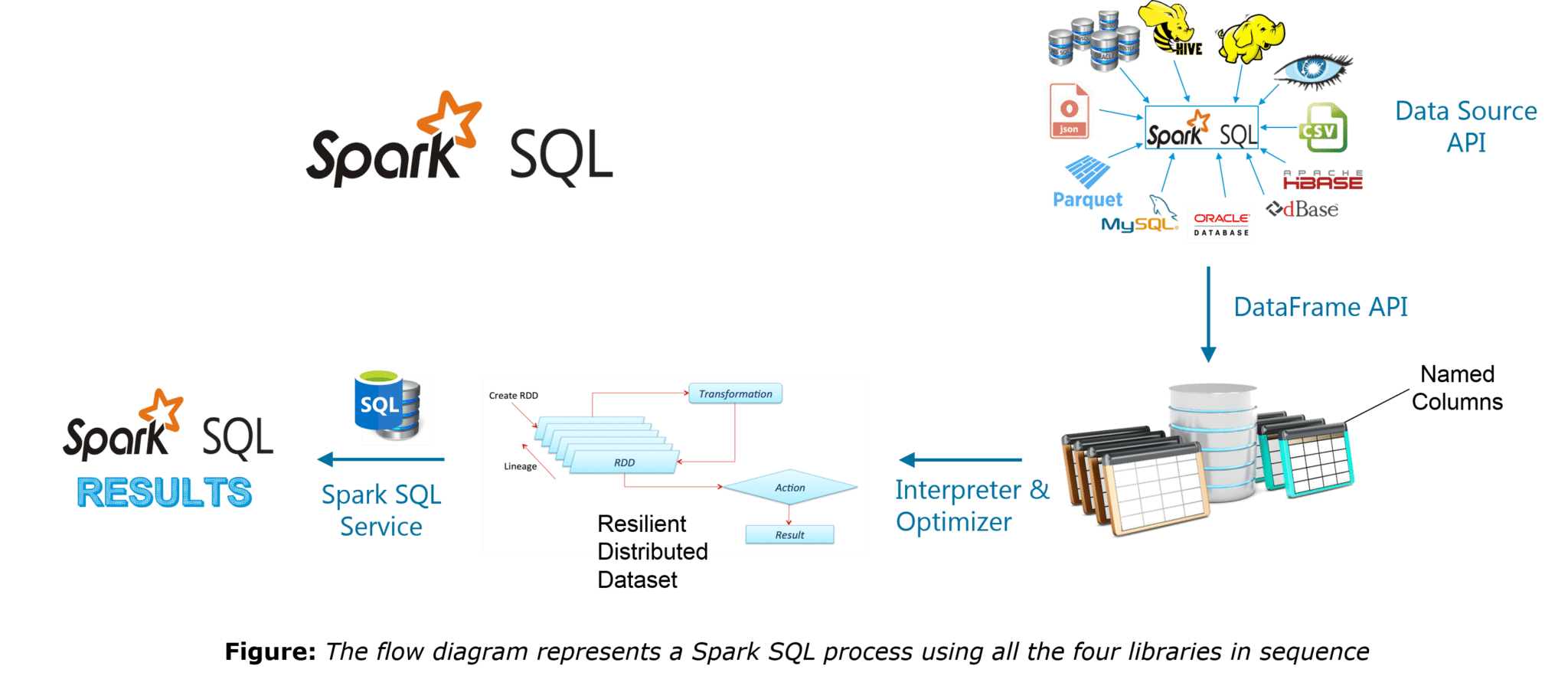
**23. Is there a module to implement SQL in Spark? How does it work?**

*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.

Spark SQL integrates relational processing with Spark’s functional programming. Further, it provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool.

The following are the four libraries of Spark SQL.

1. Data Source API
2. DataFrame API
3. Interpreter & Optimizer
4. SQL Service

**24. What is a Parquet file?**

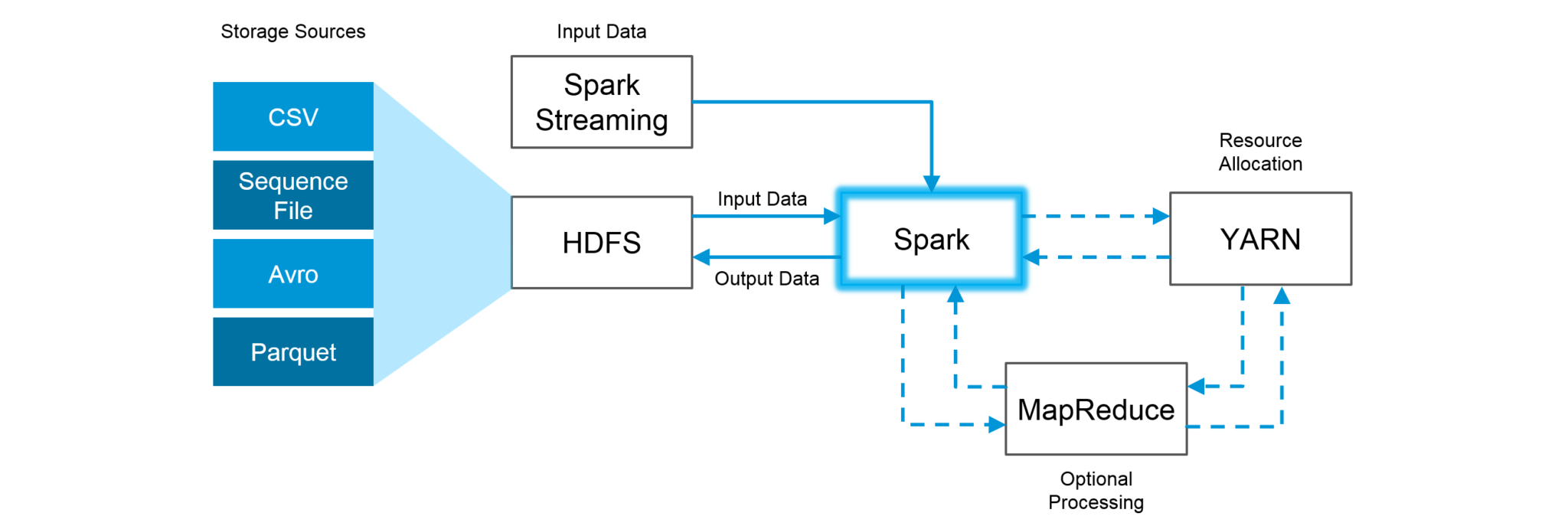
Parquet is a columnar format file supported by many other data processing systems. Spark SQL performs both read and write operations with Parquet file and consider it be one of the best big data analytics formats so far.

Parquet is a columnar format, supported by many data processing systems. The advantages of having a columnar storage are as follows:

1. Columnar storage limits IO operations.
2. It can fetch specific columns that you need to access.
3. Columnar storage consumes less space.
4. It gives better-summarized data and follows type-specific encoding.

**25. How can Apache Spark be used alongside Hadoop?**

The best part of Apache Spark is its compatibility with Hadoop. As a result, this makes for a very powerful combination of technologies. Here, we will be looking at how Spark can benefit from the best of Hadoop. Using Spark and Hadoop together helps us to leverage Spark’s processing to utilize the best of Hadoop’s HDFS and YARN.

**Figure:***Using Spark and Hadoop*

Hadoop components can be used alongside Spark in the following ways:

1. **HDFS**: Spark can run on top of HDFS to leverage the distributed replicated storage.
2. **MapReduce**: Spark can be used along with MapReduce in the same Hadoop cluster or separately as a processing framework.
3. **YARN**: Spark applications can also be run on YARN (Hadoop NextGen).
4. **Batch & Real Time Processing**: MapReduce and Spark are used together where MapReduce is used for batch processing and Spark for real-time processing.

**26. What is RDD Lineage?**

Spark does not support data replication in the memory and thus, if any data is lost, it is rebuild using RDD lineage. RDD lineage is a process that reconstructs lost data partitions. The best is that RDD always remembers how to build from other datasets.

**27. What is Spark Driver?**

Spark Driver is the program that runs on the master node of the machine and declares transformations and actions on data RDDs. In simple terms, a driver in Spark creates SparkContext, connected to a given Spark Master.  
The driver also delivers the RDD graphs to Master, where the standalone cluster manager runs.

**[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-training" \t "_blank)**[Watch The Course Preview](https://www.edureka.co/apache-spark-scala-training" \t "_blank)

**28. What file systems does Spark support?**

The following three file systems are supported by Spark:

1. Hadoop Distributed File System (HDFS).
2. Local File system.
3. Amazon S3

**29. List the functions of Spark SQL.**

Spark SQL is capable of:

1. Loading data from a variety of structured sources.
2. Querying data using SQL statements, both inside a Spark program and from external tools that connect to Spark SQL through standard database connectors (JDBC/ODBC). For instance, using business intelligence tools like Tableau.
3. Providing rich integration between SQL and regular Python/Java/Scala code, including the ability to join RDDs and SQL tables, expose custom functions in SQL, and more.

**30. What is Spark Executor?**

When SparkContext connects to a cluster manager, it acquires an Executor on nodes in the cluster. Executors are Spark processes that run computations and store the data on the worker node. The final tasks by SparkContext are transferred to executors for their execution.

**31. Name types of Cluster Managers in Spark.**

The Spark framework supports three major types of Cluster Managers:

1. **Standalone**: A basic manager to set up a cluster.
2. **Apache Mesos**: Generalized/commonly-used cluster manager, also runs Hadoop MapReduce and other applications.
3. **YARN**: Responsible for resource management in Hadoop.

**32. What do you understand by worker node?**

Worker node refers to any node that can run the application code in a cluster. The driver program must listen for and accept incoming connections from its executors and must be network addressable from the worker nodes.

Worker node is basically the slave node. Master node assigns work and worker node actually performs the assigned tasks. Worker nodes process the data stored on the node and report the resources to the master. Based on the resource availability, the master schedule tasks.

**33. Illustrate some demerits of using Spark.**

The following are some of the demerits of using Apache Spark:

1. Since Spark utilizes more storage space compared to Hadoop and MapReduce, there may arise certain problems.
2. Developers need to be careful while running their applications in Spark.
3. Instead of running everything on a single node, the work must be distributed over multiple clusters.
4. Spark’s “in-memory” capability can become a bottleneck when it comes to cost-efficient processing of big data.
5. Spark consumes a huge amount of data when compared to Hadoop.

**34. List some use cases where Spark outperforms Hadoop in processing.**

1. **Sensor Data Processing**: Apache Spark’s “In-memory” computing works best here, as data is retrieved and combined from different sources.
2. **Real Time Processing**: Spark is preferred over Hadoop for real-time querying of data. e.g. *Stock Market Analysis*, *Banking*, *Healthcare*, *Telecommunications*, etc.
3. **Stream Processing**: For processing logs and detecting frauds in live streams for alerts, Apache Spark is the best solution.
4. **Big Data Processing**:Spark runs upto 100 times faster than Hadoop when it comes to processing medium and large-sized datasets.

**35. What is a Sparse Vector?**

A sparse vector has two parallel arrays; one for indices and the other for values. These vectors are used for storing non-zero entries to save space.

|  |  |
| --- | --- |
| 1 | Vectors.sparse(7,Array(0,1,2,3,4,5,6),Array(1650d,50000d,800d,3.0,3.0,2009,95054)) |

The above sparse vector can be used instead of dense vectors.

|  |  |
| --- | --- |
| 1 | val myHouse = Vectors.dense(4450d,2600000d,4000d,4.0,4.0,1978.0,95070d,1.0,1.0,1.0,0.0) |

**36. Can you use Spark to access and analyze data stored in Cassandra databases?**

Yes, it is possible if you use Spark Cassandra Connector.To connect Spark to a Cassandra cluster, a Cassandra Connector will need to be added to the Spark project. In the setup, a Spark executor will talk to a local Cassandra node and will only query for local data. It makes queries faster by reducing the usage of the network to send data between Spark executors (to process data) and Cassandra nodes (where data lives).

**37. Is it possible to run Apache Spark on Apache Mesos?**

Yes, Apache Spark can be run on the hardware clusters managed by Mesos. In a standalone cluster deployment, the cluster manager in the below diagram is a Spark master instance. When using Mesos, the Mesos master replaces the Spark master as the cluster manager. Mesos determines what machines handle what tasks. Because it takes into account other frameworks when scheduling these many short-lived tasks, multiple frameworks can coexist on the same cluster without resorting to a static partitioning of resources.

**38. How can Spark be connected to Apache Mesos?**

To connect Spark with Mesos:

1. Configure the spark driver program to connect to Mesos.
2. Spark binary package should be in a location accessible by Mesos.
3. Install Apache Spark in the same location as that of Apache Mesos and configure the property ‘spark.mesos.executor.home’ to point to the location where it is installed.

**39. How can you minimize data transfers when working with Spark?**

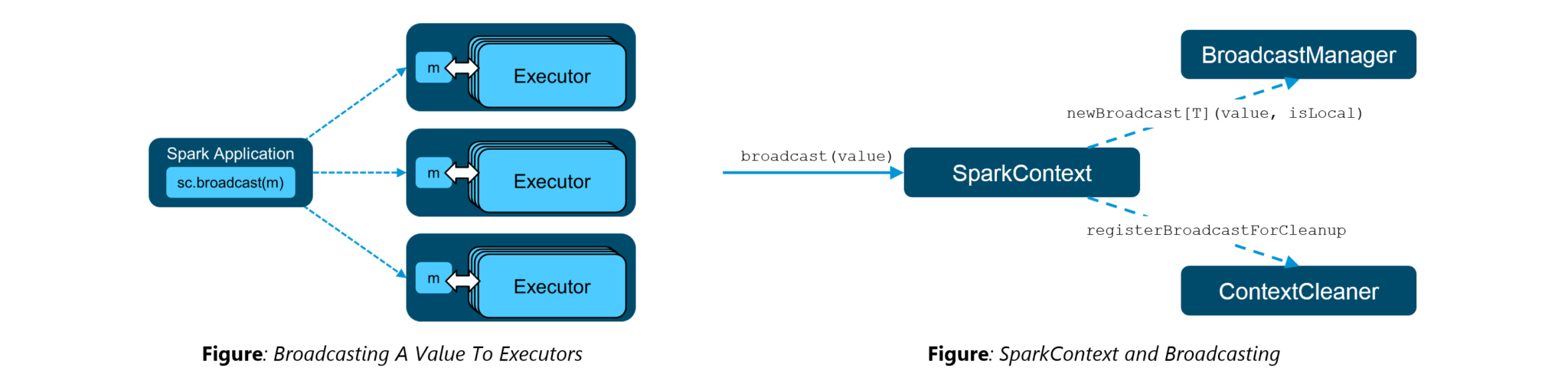
Minimizing data transfers and avoiding shuffling helps write spark programs that run in a fast and reliable manner. The various ways in which data transfers can be minimized when working with Apache Spark are:

1. Using Broadcast Variable- Broadcast variable enhances the efficiency of joins between small and large RDDs.
2. Using Accumulators – Accumulators help update the values of variables in parallel while executing.

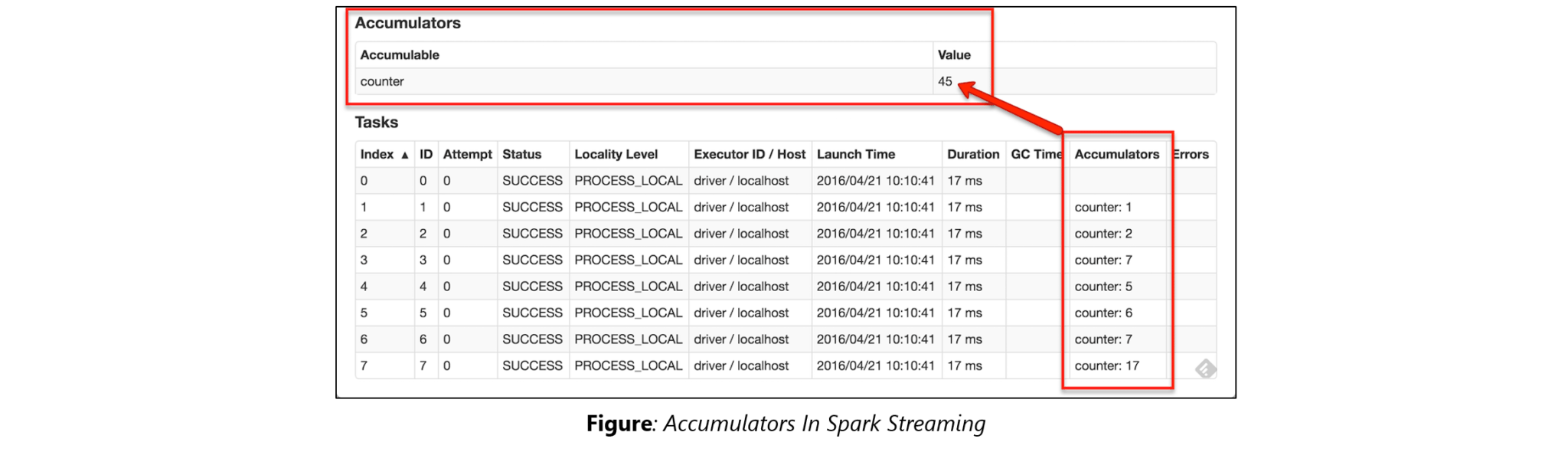
The most common way is to avoid operations ByKey, repartition or any other operations which trigger shuffles.

**40. What are broadcast variables?**

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used to give every node a copy of a large input dataset in an efficient manner. Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

**41. Explain accumulators in Apache Spark.**

Accumulators are variables that are only added through an associative and commutative operation. They are used to implement counters or sums. Tracking accumulators in the UI can be useful for understanding the progress of running stages. Spark natively supports numeric accumulators. We can create named or unnamed accumulators.

****

**42. Why is there a need for broadcast variables when working with Apache Spark?**

Broadcast variables are read only variables, present in-memory cache on every machine. When working with Spark, usage of broadcast variables eliminates the necessity to ship copies of a variable for every task, so data can be processed faster. Broadcast variables help in storing a lookup table inside the memory which enhances the retrieval efficiency when compared to an RDD *lookup()*.

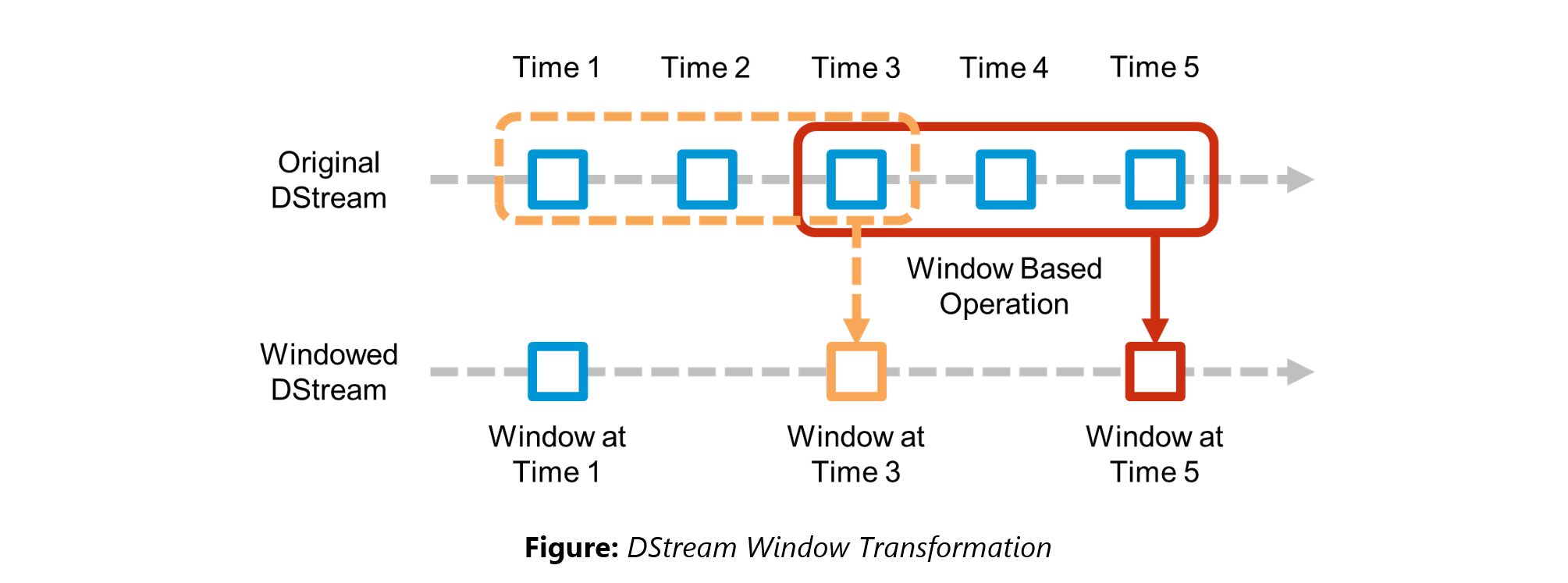
[**LEARN SPARK FROM EXPERTS**](https://www.edureka.co/apache-spark-scala-training)

**43. How can you trigger automatic clean-ups in Spark to handle accumulated metadata?**

You can trigger the clean-ups by setting the parameter ‘*spark.cleaner.ttl*’ or by dividing the long running jobs into different batches and writing the intermediary results to the disk.

**44. What is the significance of Sliding Window operation?**

Sliding Window controls transmission of data packets between various computer networks. Spark Streaming library provides windowed computations where the transformations on RDDs are applied over a sliding window of data. Whenever the window slides, the RDDs that fall within the particular window are combined and operated upon to produce new RDDs of the windowed DStream.

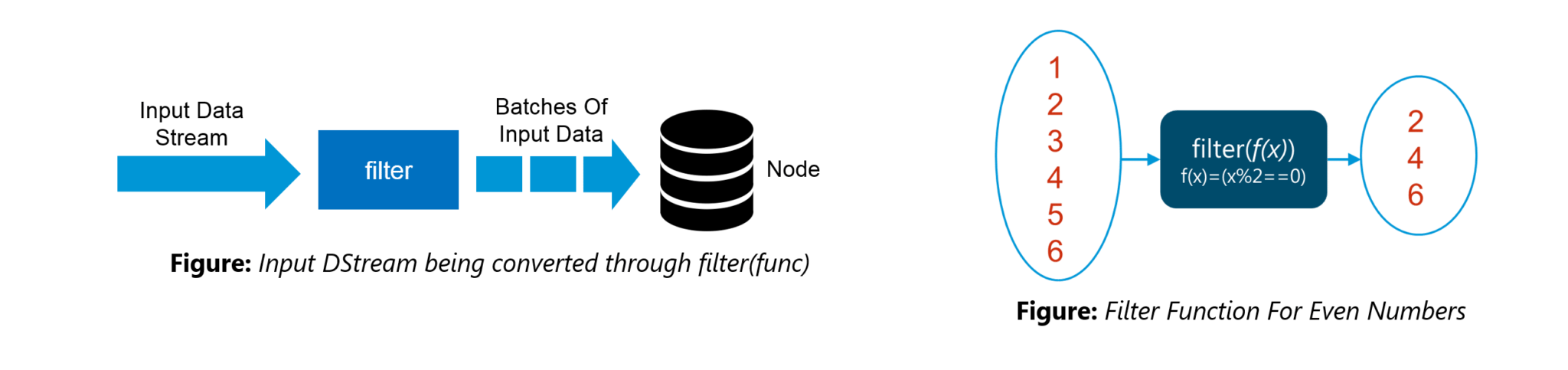
**45. What is a DStream in Apache Spark?**

***Discretized Stream***(DStream) is the basic abstraction provided by Spark Streaming. It is a continuous stream of data. It is received from a data source or from a processed data stream generated by transforming the input stream. Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval. Any operation applied on a DStream translates to operations on the underlying RDDs.

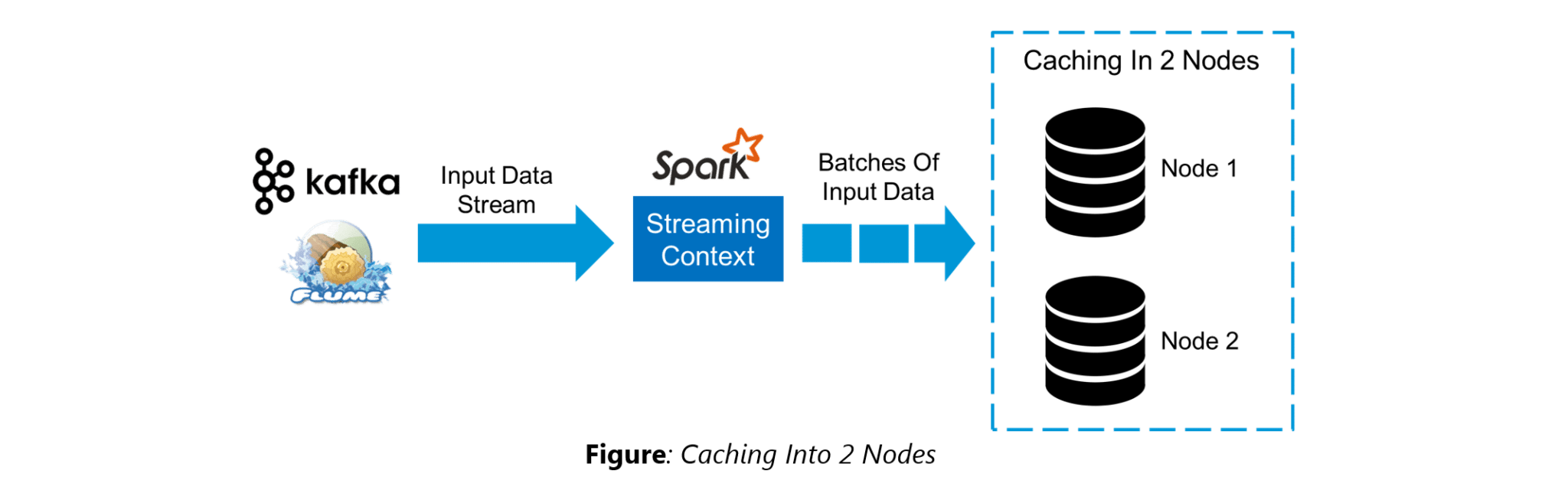
DStreams can be created from various sources like Apache Kafka, HDFS, and Apache Flume. DStreams have two operations:

1. Transformations that produce a new DStream.
2. Output operations that write data to an external system.

There are many DStream transformations possible in Spark Streaming. Let us look at **filter(*func*)**. filter(*func*) returns a new DStream by selecting only the records of the source DStream on which *func* returns true.

**46. Explain Caching in Spark Streaming.**

DStreams allow developers to cache/ persist the stream’s data in memory. This is useful if the data in the DStream will be computed multiple times. This can be done using the persist() method on a DStream. For input streams that receive data over the network (such as Kafka, Flume, Sockets, etc.), the default persistence level is set to replicate the data to two nodes for fault-tolerance.

****

**47. When running Spark applications, is it necessary to install Spark on all the nodes of YARN cluster?**

Spark need not be installed when running a job under YARN or Mesos because Spark can execute on top of YARN or Mesos clusters without affecting any change to the cluster.

**48. What are the various data sources available in Spark SQL?**

Parquet file, JSON datasets and Hive tables are the data sources available in Spark SQL.

**49. What are the various levels of persistence in Apache Spark?**

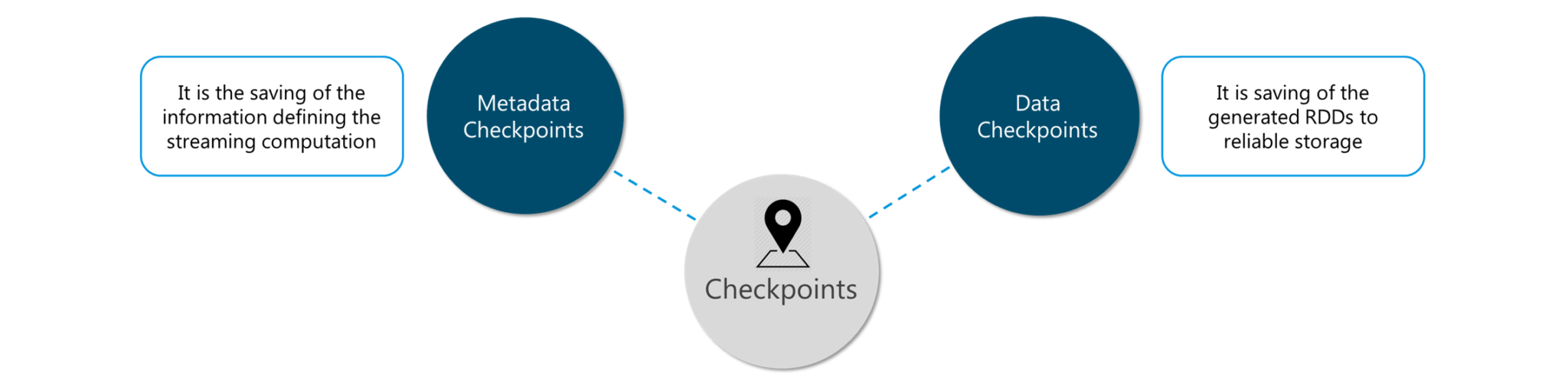
Apache Spark automatically persists the intermediary data from various shuffle operations, however, it is often suggested that users call persist () method on the RDD in case they plan to reuse it. Spark has various persistence levels to store the RDDs on disk or in memory or as a combination of both with different replication levels.

The various storage/persistence levels in Spark are:

1. MEMORY\_ONLY: Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they’re needed. This is the default level.
2. MEMORY\_AND\_DISK: Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don’t fit on disk, and read them from there when they’re needed.
3. MEMORY\_ONLY\_SER: Store RDD as *serialized* Java objects (one byte array per partition).
4. MEMORY\_AND\_DISK\_SER: Similar to MEMORY\_ONLY\_SER, but spill partitions that don’t fit in memory to disk instead of recomputing them on the fly each time they’re needed.
5. DISK\_ONLY: Store the RDD partitions only on disk.
6. OFF\_HEAP: Similar to MEMORY\_ONLY\_SER, but store the data in off-heap memory.

**50. Does Apache Spark provide checkpoints?**

*Checkpoints* are similar to checkpoints in gaming. They make it run 24/7 and make it resilient to failures unrelated to the application logic.

**Figure:** *Spark Interview Questions – Checkpoints*

Lineage graphs are always useful to recover RDDs from a failure but this is generally time-consuming if the RDDs have long lineage chains. Spark has an API for checkpointing i.e. a REPLICATE flag to persist. However, the decision on which data to checkpoint – is decided by the user. Checkpoints are useful when the lineage graphs are long and have wide dependencies.

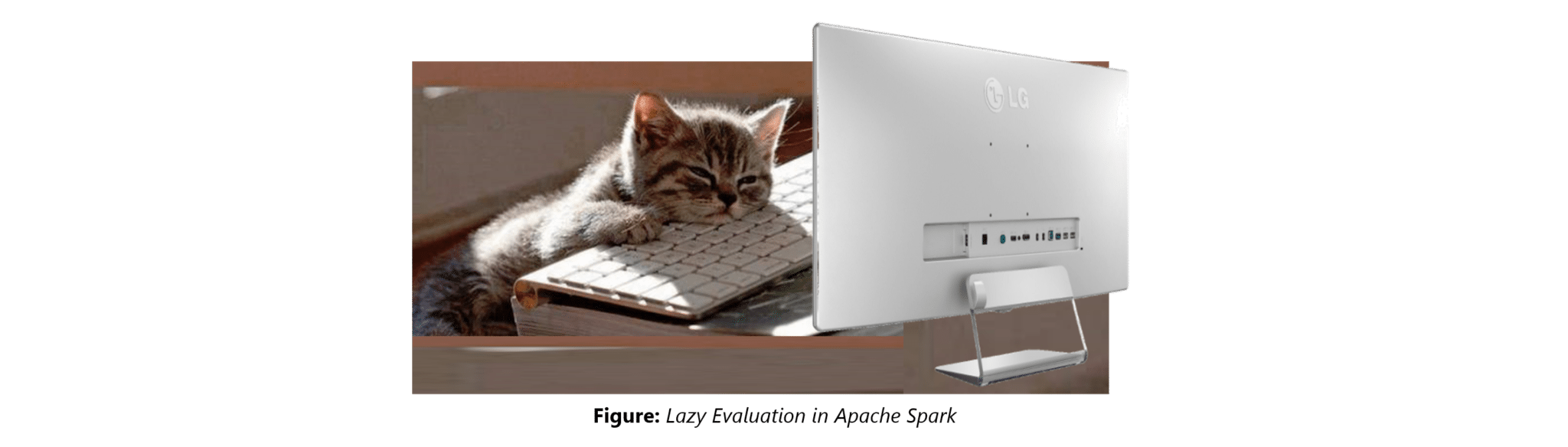
**51. How Spark uses Akka?**

Spark uses Akka basically for scheduling. All the workers request for a task to master after registering. The master just assigns the task. Here Spark uses Akka for messaging between the workers and masters.

**[Apache Spark and Scala Certification Training](https://www.edureka.co/apache-spark-scala-training" \t "_blank)**[Watch The Course Preview](https://www.edureka.co/apache-spark-scala-training" \t "_blank)

**52. What do you understand by Lazy Evaluation?**

Spark is intellectual in the manner in which it operates on data. When you tell Spark to operate on a given dataset, it heeds the instructions and makes a note of it, so that it does not forget – but it does nothing, unless asked for the final result. When a transformation like map*()* is called on an RDD, the operation is not performed immediately. Transformations in Spark are not evaluated till you perform an action. This helps optimize the overall data processing workflow.

**53. What do you understand by SchemaRDD in Apache Spark RDD?**

*SchemaRDD* is an RDD that consists of row objects (wrappers around the basic string or integer arrays) with schema information about the type of data in each column.

SchemaRDD was designed as an attempt to make life easier for developers in their daily routines of code debugging and unit testing on SparkSQL core module. The idea can boil down to describing the data structures inside RDD using a formal description similar to the relational database schema. On top of all basic functions provided by common RDD APIs, SchemaRDD also provides some straightforward relational query interface functions that are realized through SparkSQL.

Now, it is officially renamed to *DataFrame API* on Spark’s latest trunk.

**54. How is Spark SQL different from HQL and SQL?**

Spark SQL is a special component on the Spark Core engine that supports SQL and Hive Query Language without changing any syntax. It is possible to join SQL table and HQL table to Spark SQL.

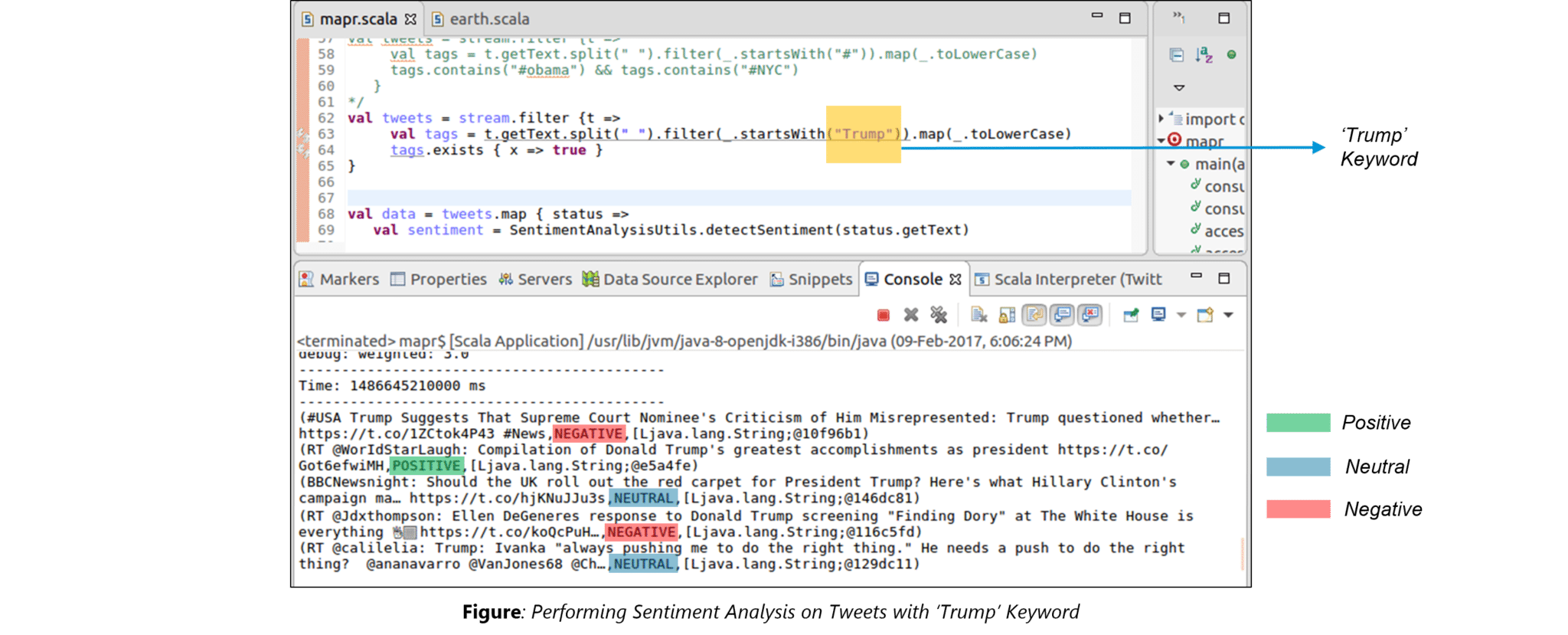
**55. Explain a scenario where you will be using Spark Streaming.**

When it comes to Spark Streaming, the data is streamed in real-time onto our Spark program.

Twitter Sentiment Analysis is a real-life use case of Spark Streaming. Trending Topics can be used to create campaigns and attract a larger audience. It helps in crisis management, service adjusting and target marketing.

Sentiment refers to the emotion behind a social media mention online. Sentiment Analysis is categorizing the tweets related to a particular topic and performing data mining using Sentiment Automation Analytics Tools.

Spark Streaming can be used to gather live tweets from around the world into the Spark program. This stream can be filtered using Spark SQL and then we can filter tweets based on the sentiment. The filtering logic will be implemented using MLlib where we can learn from the emotions of the public and change our filtering scale accordingly.



The above figure displays the sentiments for the tweets containing the word *‘Trump’*.

**DataFlair**

So, here is the Spark Interview Questions list which contains all types of interview Questions asked in Spark interview.

**Que 1. What is Apache Spark?**  
[Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/) is an open source big data framework. It has an expressive APIs to allow big data professionals to efficiently execute streaming as well as the batch. It provides faster and more general data processing platform engine. It is basically designed for fast computation. It was developed at UC Berkeley in 2009. Spark is an Apache project which is also known as “lighting fast cluster computing“. It distributes data in file system across the cluster, and process that data in parallel. It covers a wide range of workloads like batch applications, iterative algorithms, interactive queries and streaming. It lets you write an application in Java, Python or Scala.

It was developed to overcome the limitations of [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) cluster computing paradigm. Spark keeps things in memory whereas map reduce keep shuffling things in and out of disk. It allows to cache data in memory which is beneficial in iterative algorithm those used in machine learning.  
Spark is easier to develop as it knows how to operate on data. It supports SQL queries, streaming data as well as graph data processing. Spark doesn’t need Hadoop to run, it can run on its own using other storages like Cassandra, S3 from which spark can read and write. In terms of speed spark run programs up to 100x faster in memory or 10x faster on disk than Map Reduce.

**Que 2. Why Apache Spark?**  
Basically, we had so many general purpose cluster computing tools. For example [Hadoop MapReduce](https://data-flair.training/blogs/hadoop-mapreduce-tutorial/), Apache Storm, Apache Impala, Apache Storm, Apache Giraph and many more. But each one has some limitations in their functionality as well. Such as:

1. Hadoop MapReduce can only allow for batch processing.  
2. If we talk about stream processing only Apache Storm / S4 can perform.  
3. Again for interactive processing, we need Apache Impala / Apache Tez.  
4. While we need to perform graph processing, we opt for Neo4j / Apache Giraph.

Therefore, No single engine can perform all the tasks together. hence there was a big demand for a powerful engine that can process the data in real-time (streaming) as well as in batch mode  
Also, which can respond to sub-second and perform [in-memory processing](https://data-flair.training/blogs/apache-spark-in-memory-computing/)  
.

In this way, Apache Spark comes in picture. It is a powerful open-source engine that offers interactive processing, real-time [stream processing](https://data-flair.training/blogs/apache-spark-streaming-tutorial/), graph processing, in-memory processing as well as batch processing. Even with very fast speed, ease of use and also standard interface at the same time.

**Que 3. What are the components of Apache Spark Ecosystem?**  
**Apache spark consists of following components**  
1.Spark Core  
2.Spark SQL  
3.Spark Streaming  
4.MLlib  
5.GraphX

**Spark Core:** Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines [resilient distributed datasets (RDDs)](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/), which are Spark’s main programming abstraction.It also provides many APIs for building and manipulating these RDDS.

Spark SQL: Spark SQL provides an interface to work with structured data.It allows querying in SQL as well as Apache [Hive](https://data-flair.training/blogs/apache-hive-tutorial/)variant of SQL(HQL).It supports many sources.

Spark Streaming: It is spark component that enables processing of live streams of data.

MLlib: Spark comes with common machine learning package called MLlib

GraphX: GraphX is a library for manipulating graphs (e.g., a social network’s friend graph)and performing graph-parallel computations.

**Que 4. What is Spark Core?**  
Spark Core is the fundamental unit of the whole Spark project. It provides all sort of functionalities like task dispatching, scheduling, and input-output operations etc.Spark makes use of Special data structure known as [RDD (Resilient Distributed Dataset)](http://data-flair.training/blogs/rdd-in-apache-spark/). It is the home for API that defines and manipulate the RDDs. Spark Core is distributed execution engine with all the functionality attached on its top. For example, MLlib, [SparkSQL](http://data-flair.training/blogs/spark-sql-tutorial/), GraphX, [Spark Streaming](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/). Thus, allows diverse workload on single platform. All the basic functionality of Apache Spark Like [in-memory computation](http://data-flair.training/blogs/apache-spark-in-memory-computing/)**,**[fault tolerance](http://data-flair.training/blogs/apache-spark-streaming-fault-tolerance/), memory management, monitoring, task scheduling is provided by Spark Core.  
Apart from this Spark also provides the basic connectivity with the data sources. For example, [HBase](http://data-flair.training/blogs/category/hbase/), Amazon S3, [HDFS](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/)etc.  
**Que 5. Which all languages Apache Spark supports?**  
**A**pache Spark is written in Scala language. Spark provides an API in Scala, Python, and Java in order to interact with Spark. It also provides APIs in R language.  
**Que 6. How is Apache Spark better than Hadoop?**  
[Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/) is lightening fast cluster computing tool. It is up to 100 times faster than [Hadoop MapReduce](https://data-flair.training/blogs/hadoop-mapreduce-tutorial/) due to its very fast in-memory data analytics processing power.  
Apache Spark is a [Big Data](https://data-flair.training/blogs/what-is-big-data/) Framework. Apache Spark is a general purpose data processing engine and is generally used on top of HDFS. Apache Spark is suitable for the variety of data processing requirements ranging from Batch Processing to Data Streaming.

[Hadoop](https://data-flair.training/blogs/hadoop-tutorial-for-beginners/) is an open source framework which processes data stored in [HDFS](https://data-flair.training/blogs/hadoop-hdfs-tutorial/). Hadoop can process structured, unstructured or semi-structured data. Hadoop MapReduce can process the data only in Batch mode.

Apache Spark surpasses Hadoop in many cases such as  
1. Processing the data in memory which is not possible in Hadoop  
2. Processing the data that is in batch, iterative, interactive & [streaming](https://data-flair.training/blogs/apache-spark-streaming-tutorial/) i.e. Real Time mode. Whereas Hadoop processes only in batch mode.  
3. Spark is faster because it reduces the number of disk read-write operations due to its virtue of storing intermediate data in memory. Whereas in Hadoop MapReduce intermediate output which is output of Map() is always written on local hard disk  
4. Apache Spark is easy to program as it has hundreds of high-level operators with [RDD (Resilient Distributed Dataset)](https://data-flair.training/blogs/apache-spark-rdd-tutorial/)  
5. Apache Spark code is compact due compared to Hadoop MapReduce. Use of Scala makes it very short, reduces programming efforts. Also, Spark provides rich APIs in various languages such as Java, [Scala](https://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/), Python, and [R](https://data-flair.training/blogs/r-programming-tutorial/).  
6. Spark & Hadoop are both highly [fault-tolerant](https://data-flair.training/blogs/fault-tolerance-in-apache-spark/).  
7. Spark application running in Hadoop clusters is up to 10 times faster on disk than Hadoop MapReduce.

**Que 7. What are the different methods to run Spark over Apache Hadoop?**  
Instead of [MapReduce](https://data-flair.training/blogs/hadoop-mapreduce-tutorial/) we can use [spark](https://data-flair.training/blogs/apache-spark-for-beginners/) on top of [Hadoop ecosystem](https://data-flair.training/blogs/hadoop-ecosystem-components/)  
-spark with [HDFS](https://data-flair.training/blogs/hadoop-hdfs-tutorial/)  
you can read and write data in HDFS  
-spark with [Hive](https://data-flair.training/blogs/apache-hive-tutorial/)  
you can read and analyse and write back to the hive  
**Que 8. What is SparkContext in Apache Spark?**  
A SparkContext is a client of Spark’s execution environment and it acts as the master of the [Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/)application. SparkContext sets up internal services and establishes a connection to a Spark execution environment. You can [create RDDs](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/), accumulators and broadcast variables, access Spark services and run jobs (until SparkContext stops) after the creation of SparkContext. Only one SparkContext may be active per JVM. You must stop() the active SparkContext before creating a new one.

In Spark shell, a special interpreter-aware SparkContext is already created for the user, in the variable called sc.

The first step of any Spark driver application is to create a SparkContext. The SparkContext allows the Spark driver application to access the cluster through a resource manager. The resource manager can be [YARN](http://data-flair.training/blogs/category/yarn/), or [Spark’s Cluster Manager](http://data-flair.training/blogs/apache-spark-cluster-managers-tutorial/).

**Few functionalities which SparkContext offers are:**  
1. We can get the current status of a Spark application like configuration, app name.  
2. We can set Configuration like master URL, default logging level.  
3. One can create Distributed Entities like [RDDs.](http://data-flair.training/blogs/rdd-in-apache-spark/)

**Que 9. What is SparkSession in Apache Spark?**  
Starting from [Apache Spark](https://data-flair.training/forums/topic/what-is-sparksession-in-apache-spark) 2.0, Spark Session is the new entry point for Spark applications.

Prior to 2.0, [SparkContext](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/) was the entry point for spark jobs. [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) was one of the main APIs then, and it was created and manipulated using Spark Context. For every other APIs, different contexts were required – For SQL, SQL Context was required; For [Streaming](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/), Streaming Context was required; For [Hive](http://data-flair.training/blogs/category/hive/), Hive Context was required.

But from 2.0, RDD along with DataSet and its subset [DataFrame](http://data-flair.training/blogs/apache-spark-dataframe-tutorial/) APIs are becoming the standard APIs and are a basic unit of data abstraction in Spark. All of the user defined code will be written and evaluated against the DataSet and DataFrame APIs as well as RDD.

So, there is a need for a new entry point build for handling these new APIs, which is why Spark Session has been introduced. Spark Session also includes all the APIs available in different contexts – Spark Context, SQL Context, Streaming Context, Hive Context.

**Que 10. SparkSession vs SparkContext in Apache Spark.**  
[Spark Context:](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/)  
Prior to [Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/) 2.0.0 sparkContext was used as a channel to access all spark functionality.  
The spark driver program uses spark context to connect to the cluster through a resource manager ([YARN](http://data-flair.training/blogs/category/yarn/) orMesos..).  
sparkConf is required to create the spark context object, which stores configuration parameter like appName (to identify your spark driver), application, number of core and memory size of executor running on worker node.

In order to use APIs of [SQL](http://data-flair.training/blogs/spark-sql-tutorial/)**,**[HIVE](http://data-flair.training/blogs/category/hive/)**, and**[Streaming](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/), separate contexts need to be created.

**Example:**  
creating sparkConf :

val conf = new SparkConf().setAppName(“RetailDataAnalysis”).setMaster(“spark://master:7077”).set(“spark.executor.memory”, “2g”)

creation of sparkContext:

val sc = new SparkContext(conf)

**Spark Session:**

SPARK 2.0.0 onwards, SparkSession provides a single point of entry to interact with underlying Spark functionality and  
allows programming Spark with [DataFrame](http://data-flair.training/blogs/apache-spark-dataframe-tutorial/) and Dataset APIs. All the functionality available with sparkContext are also available in sparkSession.

In order to use APIs of SQL, HIVE, and Streaming, no need to create separate contexts as sparkSession includes all the APIs.

Once the SparkSession is instantiated, we can configure Spark’s run-time config properties.

**Example:**

Creating Spark session:  
val spark = SparkSession  
.builder  
.appName(“WorldBankIndex”)  
.getOrCreate()

Configuring properties:  
spark.conf.set(“spark.sql.shuffle.partitions”, 6)  
spark.conf.set(“spark.executor.memory”, “2g”)

Spark 2.0.0 onwards, it is better to use sparkSession as it provides access to all the spark Functionalities that sparkContext does. Also, it provides APIs to work on DataFrames and Datasets.

**Que 11. What are the abstractions of Apache Spark?**  
There are several abstractions of [Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/):

**1. RDD:**  
An RDD refers to Resilient Distributed Datasets. RDDs are Read-only partition collection of records. It is Spark’s core abstraction and also a fundamental data structure of Spark. It offers to conduct [in-memory computations](https://data-flair.training/blogs/apache-spark-in-memory-computing/) on large clusters. Even in a [fault-tolerant](https://data-flair.training/blogs/fault-tolerance-in-apache-spark/) manner. For more detailed insights on RDD.follow link: [Spark RDD – Introduction, Features & Operations of RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/)

**2. DataFrames:**  
It is a Dataset organized into named columns. DataFrames are equivalent to the table in a relational database or data frame in [R](https://data-flair.training/blogs/r-programming-tutorial/) /Python. In other words, we can say it is a relational table with good optimization technique. It is an immutable distributed collection of data. Allowing higher-level abstraction, it allows developers to impose a structure onto a distributed collection of data,. For more detailed insights on DataFrames. refer link:[Spark SQL DataFrame Tutorial – An Introduction to DataFrame](https://data-flair.training/blogs/apache-spark-sql-dataframe-tutorial/)

**3. Spark Streaming:**  
It is a Spark’s core extension, which allows Real-time stream processing From several sources. For example Flume and Kafka. To offer a unified, continuous DataFrame abstraction that can be used for interactive and batch queries these two sources work together. It offers scalable, high-throughput and fault-tolerant processing. For more detailed insights on Spark Streaming. refer link: [Spark Streaming Tutorial for Beginners](https://data-flair.training/blogs/apache-spark-streaming-tutorial/)

**4. GraphX**  
It is one more example of specialized data abstraction. It enables developers to analyze social networks. Also, other graphs alongside Excel-like two-dimensional data. For more detailed insights on GaphX

**Que 12. How can we create RDD in Apache Spark?**  
**Resilient Distributed Datasets (RDD)** is spark’s core abstraction which is a [resilient distributed dataset](https://data-flair.training/blogs/apache-spark-rdd-tutorial/).  
It is an immutable (read-only) distributed collection of objects.  
Each [dataset](https://data-flair.training/blogs/apache-spark-dataset-tutorial/) in RDD is divided into logical partitions,  
which may be computed on different nodes of the cluster.  
Including user-defined classes, RDDs may contain any type of Python, Java, or [Scala](https://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/) objects.

In 3 ways we can create RDD in [Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/):  
1. Through distributing collection of objects  
2. By loading an external dataset  
3. From existing Apache Spark RDDs

1. **Using parallelized collection**

RDDs are generally created by parallelizing an existing collection  
i.e. by taking an existing collection in the program and passing  
it to [SparkContext’s](https://data-flair.training/blogs/learn-apache-spark-sparkcontext/) parallelize() method.

scala > val data = Array(1,2,3,4,5)  
scala > val dataRDD = sc.parallelize (data)  
scala > dataRDD.count

2. **External Datasets**

In Spark, a distributed dataset can be formed from any data source supported by Hadoop.

val dataRDD = spark.read.textFile(“F:/BigData/DataFlair/Spark/Posts.xml”).rdd

3. **Creating RDD from existing RDD**

Transformation is the way to create an RDD from already existing RDD.

Transformation acts as a function that intakes an RDD and produces another resultant RDD.  
The input RDD does not get changed,  
Some of the [operations](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/) applied on RDD are: filter, Map, FlatMap

val dataRDD = spark.read.textFile(“F:/Mritunjay/BigData/DataFlair/Spark/Posts.xml”).rdd

val resultRDD = data.filter{line => {line.trim().startsWith(“<row”)}  
}

**Que 13. Why is Spark RDD immutable?**  
Following are the reasons:  
– Immutable data is always safe to share across multiple processes as well as multiple threads.  
– Since [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) is immutable we can recreate the RDD any time. (From lineage graph).  
– If the computation is time-consuming, in that we can cache the RDD which result in performance improvement.

**Please add more points if I am missing something**  
RDDs are also fault-tolerant and evaluate lazily

**Que 14. Explain the term paired RDD in Apache Spark**  
**Introduction**  
Paired RDD is a distributed collection of data with the key-value pair. It is a subset of [Resilient Distributed Dataset](https://data-flair.training/blogs/apache-spark-rdd-tutorial/). So it has all the [feature of RDD](https://data-flair.training/blogs/apache-spark-rdd-features/) and some new feature for the key-value pair. There are many [transformation operations](https://data-flair.training/blogs/spark-rdd-operations-transformations-actions/) available for Paired RDD. These operations on Paired RDD are very useful to solve many use cases that require sorting, grouping, reducing some value/function.  
Commonly used operations on paired RDD are: groupByKey() reduceByKey() countByKey() join() etc  
Creation of Paired RDD:  
val pRDD:[(String),(Int)]=sc.textFile(“path\_of\_your\_file”)  
.flatMap(line => line.split(” “))  
.map{word=>(word,word.length)}  
Also using subString method(if we have a file with id and some value, we can create paired rdd with id as key and value as other details)

val pRDD2[(Int),(String)]=sc.textFile(“path\_of\_your\_file”)  
.keyBy(line=>line.subString(1,5).trim().toInt)  
.mapValues(line=>line.subString(10,30).trim())

**Que 15. How is RDD in Spark different from Distributed Storage Management?**  
Some of the differences between an [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) and Distributed Storage are as follows:

Resilient Distributed Dataset (RDD) is the primary abstraction of data for [Apache Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/)framework.  
Distributed Storage is simply a file system which works on multiple nodes.

RDDs store data [in-memory](http://data-flair.training/blogs/apache-spark-in-memory-computing/) (unless explicitly cached).  
Distributed Storage stores data in persistent storage.

RDDs can re-compute itself in the case of failure or data loss.  
If data is lost from the Distributed Storage system it is gone forever (unless there is an internal replication system).

**Que 16. Explain transformation and action in RDD in Apache Spark.**  
Transformations are [operations on RDD](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) that create one or more new [RDDs](http://data-flair.training/blogs/rdd-in-apache-spark/). E.g. map, filter, reduceByKey etc. In other words, transformations are functions that take an RDD as the input and produce one or more RDDs as the output. There is no change in the input RDD, but it always produces one or more new RDDs by applying the computations they represent.Transformations are lazy, i.e. are not executed immediately. Only after calling an action are transformations executed.

Actions are RDD operations that produce non-RDD values. In other words, an RDD operation that returns a value of any type but an RDD is an action. They trigger execution of RDD transformations to return values. Simply put, an action evaluates the [RDD lineage graph](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/). E.g. collect, reduce, count, foreach etc.

**Que 17. What are the types of Apache Spark transformation?**  
To understand the types of Transformation better, Let’s begin with the brief introduction of Transformation in[Apache Spark](https://data-flair.training/blogs/apache-spark-for-beginners/).

**Transformation in Spark**  
Spark Transformation is a function that produces new [RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/)from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. As RDDs are immutable in nature, so input RDDs, cannot be changed.  
An RDD lineage, built by Applying transformation built with the entire parent RDDs of the final RDD(s). In other words, it is also known as RDD operator graph or RDD dependency graph. It is a logical execution plan i.e., it is [Directed Acyclic Graph (DAG)](https://data-flair.training/blogs/dag-in-apache-spark/) of the entire parent RDDs of RDD.

Transformations are lazy in nature i.e., they get execute when we call an action. They are not executed immediately. Two most basic type of transformations is a map(), filter().

Resultant RDD is always dissimilar from its parent RDD. It can be smaller (e.g. filter, count, distinct, sample), bigger (e.g. flatMap(), union(), Cartesian()) or the same size (e.g. map).

Now, let’s focus on the question, there are fundamentally two types of transformations:

**1. Narrow transformation –**  
While talking about Narrow transformation, all the elements which are required to compute the records in single partition reside in the single partition of parent RDD. To calculate the result, a limited subset of partition is used. This Transformation are the result of map(), filter().

**2. Wide Transformations –**  
Wide transformation means all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. Partitions may reside in many different partitions of parent RDD. This Transformation is a result of groupbyKey() and reducebyKey().

For more detailed insights of Transformations in Spark.

* **Que 18. Explain the RDD properties.**  
  RDD (Resilient Distributed Dataset) is a basic abstraction in [Apache Spark](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/).
* RDD is an immutable, partitioned collection of elements on the cluster which can be operated in parallel.
* **Each RDD is characterized by five main properties :**
* *Below operations are lineage operations.*1. List or Set of partitions.  
  2. List of dependencies on other (parent) RDD  
  3. A function to compute each partition

*Below operations are used for optimization during execution.*

4. Optional preferred location **[i.e. block location of an HDFS file] [it’s about data locality]**  
5. Optional partitioned info **[i.e. Hash-Partition for Key/Value pair –> When data shuffled how data will be traveled]**

Examples :  
**#HadoopRDD :**

* HadoopRDD provides core functionality for reading data stored in Hadoop ([HDFS](http://data-flair.training/blogs/hdfs-data-read-operation/), [HBase](http://data-flair.training/blogs/category/hbase/), Amazon S3..) using the older [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/)API (org.apache.hadoop.mapred)
* Properties of HadoopRDD :

*1. List or Set of partitions: One per HDFS block  
2. List of dependencies on parent RDD: None  
3. A function to compute each partition: read respective HDFS block  
4. Optional Preferred location: HDFS block location  
5. Optional partitioned info: None*

**#FilteredRDD :**

* Properties of FilteredRDD:

*1. List or Set of partitions: No. of partitions same as parent RDD  
2. List of dependencies on parent RDD: ‘one-to-one’ as parent (same as parent)  
3. A function to compute each partition: compute parent and then filter it  
4. Optional Preferred location: None (Ask Parent)  
5. Optional partitioned info: None*

**Que 19. What is lineage graph in Apache Spark?**  
When we apply a different [transformation on RDD](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) it [creates RDD](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/) Linage graph. It is a new [RDD](http://data-flair.training/blogs/rdd-in-apache-spark/) from already existing RDDs. It is the dependencies graph between the existing and the new RDD formed. the need of RDD lineage graph arrives when we want to compute new RDD or if we want to recover the lost data from the lost persisted RDD.

* Adding few more points on lineage graph:  
  You can check lineage between two RDDs using rdd0.toDebugString. This gives back you the lineage graph from current rdd to all the previous dependencies of RDDs. See below. Whenever you see “+-” symbol from the toDebugString output, it means there will be next stage from the next operation onwards. This is indicates to identify that how many stage are created.

scala> val rdd0 = sc.parallelize(List(“Ashok Vengala”,”Ashok Vengala”,”DataFlair”))  
rdd0: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[10] at parallelize at <console>:31

scala> val count = rdd0.flatMap(rec => rec.split(” “)).map(word => (word,1)).reduceByKey(\_+\_)  
count: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[13] at reduceByKey at <console>:33

scala> count.toDebugString  
res24: String =  
(2) ShuffledRDD[13] at reduceByKey at <console>:33 []  
+-(2) MapPartitionsRDD[12] at map at <console>:33 []  
| MapPartitionsRDD[11] at flatMap at <console>:33 []  
| ParallelCollectionRDD[10] at parallelize at <console>:31 []

From down to up (i.e, last three rows): These will be performed in stage-0. And the first row(ShuffledRDD): this will operation will be performed in stage-1.

In toDebugString output, we are seeing something like ParallelCollectionRDD, MapPartitionsRDD and ShuffleRDD. These are all implementation of RDD abstract class.

**Que 20.  Explain the terms  Spark Partitions and Partitioners.**  
**PARTITIONS :**

<li style=”list-style-type: none”>

* Partitions also known as ‘Split’ in HDFS, is a logical chunk of data set which may be in the range of Petabyte, Terabytes and distributed across the cluster.
* By Default, Spark creates one Partition for each block of the file (For HDFS)
* Default block size for HDFS block is 64 MB (Hadoop Version 1) / 128 MB (Hadoop Version 2) so as the split size.
* However, one can explicitly specify the number of partitions to be created.
* Partitions are basically used to speed up the data processing.

**PARTITIONER :**

* An object that defines how the elements in a key-value pair RDD are partitioned by key. Maps each key to a partition ID, from 0 to (number of partitions – 1)
* Partitioner captures the data distribution at the output. A scheduler can optimize future operation based on the type of partitioner. (i.e. if we perform any operation say transformation or action which require shuffling across nodes in that we may need the partitioner. Please refer reduceByKey() transformation in the forum)
* Basically there are three types of partitioners in Spark:

(1) Hash-Partitioner (2) Range-Partitioner (3) One can make its *Custom Partitioner*

**Property Name :**spark.default.parallelism  
**Default Value:**For distributed shuffle operations like reduceByKey and join, the largest number of partitions in a parent RDD. For operations like parallelize with no parent RDDs, it depends on the cluster manager:  
•Local mode: number of cores on the local machine  
•Mesos fine-grained mode: 8  
•Others: total number of cores on all executor nodes or 2, whichever is larger  
**Meaning :**Default number of partitions in RDDs returned by transformations like join.

Ans. Partition in Spark is similar to split in HDFS. A partition in Spark is a logical division of data stored on a node in the cluster. They are the basic units of parallelism in Apache Spark. RDDs are a collection of partitions.When some actions are executed, a task is launched per partition.

By default, partitions are automatically created by the framework. However, the number of partitions in Spark are configurable to suit the needs.  
For the number of partitions, if spark.default.parallelism is set, then we should use the value from SparkContext defaultParallelism, othewrwise we suould use the max number of upstream partitions. Unless spark.default.parallelism is set, the number of partitions will be the same as that of the largest upstream RDD, as this would least likely cause an out-of-memory errors.

A partitioner is an object that defines how the elements in a key-value pair RDD are partitioned by key, maps each key to a partition ID from 0 to numPartitions – 1. It captures the data distribution at the output. With the help of partitioner, the scheduler can optimize the future operations. The contract of partitioner ensures that records for a given key have to reside on a single partition.  
We should choose a partitioner to use for a cogroup-like operations. If any of the RDDs already has a partitioner, we should choose that one. Otherwise, we use a default HashPartitioner.

There are three types of partitioners in Spark :  
a) Hash Partitioner b) Range Partitioner c) Custom Partitioner  
Hash – Partitioner : Hash- partitioning attempts to spread the data evenly across various partitions based on the key.  
Range – Partitioner : In Range- Partitioning method , tuples having keys with same range will appear on the same machine.

RDDs can be created with specific partitioning in two ways :  
i) Providing explicit partitioner by calling partitionBy method on an RDD  
ii) Applying transformations that return RDDs with specific partitioners .

* **Que 21. By Default, how many partitions are created in RDD in Apache Spark?**  
  By Default, Spark creates one Partition for each block of the file (For HDFS)
* Default block size for HDFS block is 64 MB (Hadoop Version 1) / 128 MB (Hadoop Version 2).
* However, one can explicitly specify the number of partitions to be created.

**Example1:**

<li style=”list-style-type: none”>

* No Partition is not specified

val rdd1 = sc.textFile("/home/hdadmin/wc-data.txt")

**Example2:**

<li style=”list-style-type: none”>

* Following code create the RDD of 10 partitions, since we specify the no. of partitions.

val rdd1 = sc.textFile("/home/hdadmin/wc-data.txt", 10)

One can query about the number of partitions in following way :

rdd1.partitions.length

**OR**

rdd1.getNumPartitions

* Best case Scenario is that we should make RDD in following way:  
   **numbers of cores in Cluster = no. of partitions**

val rdd1 = sc.textFile(“/home/hdadmin/wc-data.txt”)

Consider the size of wc-data.txt is of 1280 MB and Default block size is 128 MB. So there will be 10 blocks created and 10 default partitions(1 per block).

For a better performance, we can increase the number of partitions on each block. Below code will create 20 partitions on 10 blocks(2 partitions/block). Performance will be improved but need to make sure that each cluster is running on 2 cores minimum.

val rdd1 = sc.textFile(“/home/hdadmin/wc-data.txt”, 20)

**Que 22. What is Spark DataFrames?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-dataframes-in-spark)  
**Que 23. What are benefits of DataFrame in Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-benefits-of-dataframe-in-spark#post-1611)  
**Que 24. What is Spark Dataset?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-dataset-in-spark)  
**Que 25. What are the advantages of datasets in spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-advantages-of-datasets-in-spark)  
**Que 26. What is Directed Acyclic Graph in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-directed-acyclic-graph-in-apache-spark)  
**Que 27. What is the need for Spark DAG?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-the-need-for-dag-in-the-spark#post-2226)  
**Que 28.What is the difference between DAG and Lineage?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-the-difference-between-dag-and-lineage)  
**Que 29. What is the difference between Caching and Persistence in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-difference-between-caching-and-persistence#post-45)  
**Que 30. What are the limitations of Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-limitations-of-apache-spark)  
**Que 31. Different Running Modes of Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/different-running-modes-of-apache-spark)  
**Que 32. What are the different ways of representing data in Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-different-ways-of-representing-data-in-spark)  
**Que 33. What is write ahead log(journaling) in Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-write-ahead-logjournaling-in-spark)  
**Que 34. Explain catalyst query optimizer in Apache Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-catalyst-query-optimizer-in-apache-spark)  
**Que 35. What are shared variables in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-shared-variables-in-apache-spark)  
**Que 36. How does Apache Spark handles accumulated Metadata?**  
[**View Answer**](https://data-flair.training/forums/topic/how-does-apache-spark-handles-accumulated-meta-data)  
**Que 37. What is Apache Spark Machine learning library?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-apache-spark-machine-learning-library)  
**Que 38. List commonly used Machine Learning Algorithm.**  
[**View Answer**](https://data-flair.training/forums/topic/list-commonly-used-machine-learning-algorithm)  
**Que 39. What is the difference between DSM and RDD?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-the-difference-between-dsm-and-rdd-1)  
**Que 40. List the advantage of Parquet file in Apache Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/list-the-advantage-of-parquet-file-in-apache-spark)  
**Que 41. What is lazy evaluation in Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-lazy-evaluation-in-spark)  
**Que 42. What are the benefits of Spark lazy evaluation?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-benefits-of-lazy-evaluation-in-spark)  
**Que 43. How much faster is Apache spark than Hadoop?**  
[**View Answer**](https://data-flair.training/forums/topic/how-much-faster-is-apache-spark-than-hadoop)  
**Que 44. What are the ways to launch Apache Spark over YARN?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-ways-to-launch-apache-spark-over-yarn)  
**Que 45. Explain various cluster manager in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/explain-various-cluster-manager-in-apache-spark)  
**Que 46. What is Speculative Execution in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/speculative-execution-in-apache-spark)  
**Que 47. How can data transfer be minimized when working with Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/how-can-data-transfer-be-minimized-when-working-with-apache-spark)  
**Que 48. What are the cases where Apache Spark surpasses Hadoop?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-cases-where-apache-spark-surpasses-hadoop)  
**Que 49. What is action, how it process data in apache spark**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-action-how-it-process-data-in-spark)  
**Que 50. How is fault tolerance achieved in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/how-is-fault-tolerance-achieved-in-apache-spark)  
**Que 51. What is the role of Spark Driver in spark applications?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-the-role-of-spark-driver-in-spark-applications)  
**Que 52. What is worker node in Apache Spark cluster?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-worker-node-in-apache-spark-cluster)  
**Que 53. Why is Transformation lazy in Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/why-transformation-is-lazy-in-spark)  
**Que 54. Can I run Apache Spark without Hadoop?**  
[**View Answer**](https://data-flair.training/forums/topic/can-i-run-apache-spark-without-hadoop)  
**Que 55. Explain Accumulator in Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-accumulator-in-detail)  
**Que 56.  What is the role of Driver program in Spark Application?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-role-of-driver-program-in-spark-application)  
**Que 57. How to identify that given operation is Transformation/Action in your program?**  
[**View Answer**](https://data-flair.training/forums/topic/how-to-identify-that-given-operation-is-transformationaction-in-your-program)  
**Que 58. Name the two types of shared variable available in Apache Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/name-the-two-types-of-shared-variable-available-in-apache-spark)  
**Que 59. What are the common faults of the developer while using Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-common-faults-of-developer-while-using-apache-spark)  
**Que 60. By Default, how many partitions are created in RDD in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/by-default-how-many-partitions-are-created)  
**Que 61. Why we need compression and what are the different compression format supported?**  
[**View Answer**](https://data-flair.training/forums/topic/why-we-need-compression-and-what-the-different-compression-format-supported)  
**Que 62. Explain the filter transformation.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-filter-transformation)  
**Que 63. How to start and stop spark in interactive shell?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-the-command-to-start-and-stop-the-spark-in-interactive-shell)  
**Que 64. Explain sortByKey() operation.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-sortbykey-operation)  
**Que 65. Explain distnct(),union(),intersection() and substract() transformation in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-distnct-union-intersection-and-substract-transformation)  
**Que 66.Explain foreach() operation in apache spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-foreach-operation)  
**Que 67.groupByKey vs reduceByKey in Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/groupbykey-vs-reducebykey-in-apache-spark)  
**Que 68. Explain mapPartitions() and mapPartitionsWithIndex()**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-mappartitions-and-mappartitionswithindex)  
**Que 69. What is Map in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-map-in-apache-spark)  
**Que 70. What is FlatMap in Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-flatmap-in-apache-spark)  
**Que 71.Explain fold() operation in Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-fold-operation)  
**Que 72. Explain API createOrReplaceTempView()**  
[**View Answer**](https://data-flair.training/forums/topic/explain-api-createorreplacetempview)  
**Que 73. Explain values() operation in Apache Spark.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-values-operation)  
**Que 74. Explain keys() operation in Apache spark.**  
[**View Answer**](https://data-flair.training/forums/topic/explain-keys-operation)  
**Que 75. Explain textFile Vs wholeTextFile in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/textfile-vs-wholetextfile-in-spark)  
**Que 76. Explain cogroup() operation in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-cogroup-operation)  
**Que 77. Explain pipe() operation in Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-pipe-operation)  
**Que 78. Explain Spark coalesce() operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-coalesce-operation)  
**Que 79.Explain the repartition() operation in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-repartition-operation)  
**Que 80. Explain fullOuterJoin() operation in Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-fullouterjoin-operation)  
**Que 81. Expain Spark leftOuterJoin() and rightOuterJoin() operation**  
[**View Answer**](https://data-flair.training/forums/topic/expain-leftouterjoin-and-rightouterjoin-operation)  
**Que 82. Explain Spark join() operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-join-operation)  
**Que 83. Explain the top() and takeOrdered() operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-top-and-takeordered-operation)  
**Que 84. Explain first() operation in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-first-operation)  
**Que 85. Explain sum(), max(), min() operation in Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-sum-max-min-operation)  
**Que 86. Explain countByValue() operation in Apache Spark RDD**  
[**View Answer**](https://data-flair.training/forums/topic/explain-countbyvalue-operation)  
**Que 87. Explain the lookup() operation in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-lookup-operation)  
**Que 88. Explain Spark countByKey() operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-countbykey-operation)  
**Que 89. Explain Spark saveAsTextFile() operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-saveastextfile-operation)  
**Que 90. Explain reduceByKey() Spark operation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-reducebykey-operation)  
**Que 91. Explain the operation reduce() in Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-operation-reduce)  
**Que 92.Explain the action count() in Spark RDD**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-action-count)  
**Que 93. Explain Spark map() transformation**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-map-transformation)  
**Que 94. Explain the flatMap() transformation in Apache Spark**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-filatmap-transformation)  
**Que 95. What are the limitations of Apache Spark?**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-limitations-of-apache-spark)  
**Que 96. What is Spark SQL?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-spark-sql)  
**Que 97. Explain Spark SQL caching and uncaching**  
[**View Answer**](https://data-flair.training/forums/topic/spark-sql-caching-and-uncaching)  
**Que 98. Explain Spark streaming**  
[**View Answer**](https://data-flair.training/forums/topic/explain-spark-streaming)  
**Que 99. What is DStream in Apache Spark Streaming?**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-dstream-in-apache-spark-streaming)  
**Que 100. Explain different transformations in DStream in Apache Spark Streaming**  
[**View Answer**](https://data-flair.training/forums/topic/explain-different-transformations-in-dstream-in-apache-spark-streaming)  
**Que 101. What is Starvation scenario in spark streaming**  
[**View Answer**](https://data-flair.training/forums/topic/what-is-starvation-scenario-in-spark-streaming)  
**Que 102.Explain the level of parallelism in spark streaming**  
[**View Answer**](https://data-flair.training/forums/topic/explain-the-level-of-parallelism)  
**Que 103. What are the different input sources for Spark Streaming**  
[**View Answer**](https://data-flair.training/forums/topic/what-are-the-different-input-sources-for-spark-streaming)  
**Que 104. Explain Spark Streaming with Socket**  
[**View Answer**](https://data-flair.training/forums/topic/spark-streaming-with-socket)  
**Que 105.  Define the roles of the file system in any framework?**  
[**View Answer**](http://data-flair.training/forums/topic/what-is-the-role-of-file-system)  
**Que 106. How do you parse data in XML? Which kind of class do you use with Java to parse data?**  
[**View Answer**](http://data-flair.training/forums/topic/how-do-you-parse-data-in-xml-which-kind-of-class-you-use-with-java-to-pass-data)  
**Que 107. What is PageRank in Spark?**  
[**View Answer**](https://data-flair.training/forums/)  
**Que 108. What are the roles and responsibilities of worker nodes in the Apache Spark cluster? Is Worker Node in Spark is same as Slave Node?**  
[**View Answer**](http://data-flair.training/forums/topic/what-is-worker-node-in-apache-spark-cluster)  
**Que 109. How to split single HDFS block into partitions RDD?**  
[**View Answer**](http://data-flair.training/forums/topic/rdd-partition)  
**Que 110. On what all basis can you differentiate RDD, DataFrame, and DataSet?**  
[**View Answer**](http://data-flair.training/forums/topic/rdds-vs-dataframes)  
So, this was all on Apache spark interview Questions. hope you like the Apache spark interview Questions and Answers explained to it.