APACHE SPARK ARCHITECTURE INTRODUCTION

By QuickTechie.com

About Databricks PySpark Certification Syllabus

If you check the syllabus it is mentioned by just simple topic on the certification web page. However, if you pick each individual point then syllabus is quite huge and you should have a good amount of fundamental concepts cleared and well-practiced all the questions given here. In total there are 9 sections which are being evaluated in the exam and each section could have their own topics. Which makes entire syllabus to cover 40+ topics. That is considered a quite a huge syllabus compares to any other certifications. The advantage of this, once you learn all the concepts mentioned in the certification, you would become expert in PySpark framework using Spark 3.0 . We will go through each topic mentioned in the syllabus one by one and try clear all your concepts. We would divide entire syllabus into different chapters correspond to each section.

Describe basic Spark architecture and define terminology such as "driver" and "executor".

Driver

In your Spark application you would be having one component that is known as Driver, driver is a program using which you create a SparkContext object which is connected to the Spark master which can be a Local, Standalone or YARN etc. Driver program can an independently located or can be placed on the same node where master exists. Driver must be accessible from all the worker nodes as well over the network. You would be having some code written your Spark application as below

```
# IP address of the master or you can provide as #yarn in case of Hadoop
    conf = SparkConf()
            .setMaster("URL for the Master")
            .setAppName("HESparkApp")

    sc = spark.SparkContext(conf)
```

Driver has the following responsibilities

- Driver code will create the SparkContext (Entry point to the Spark Cluster) and declares all the operations like transformations and actions and create DAG (Direct Acyclic Graph)
- Next, driver would submit the serialized RDD graph to the master. And then its master responsibility to divide entire DAG into smaller tasks and submit them to workers for further executions.
- Worker are nodes in the cluster where all the divided tasks would be executed in parallel on the RDD partitions.

In case of Hadoop master can be “yarn” (Yet another resource negotiator). You would be writing your application main method (this is a starting point of your application). So if you think in terms of Java then this is a Class where you would be writing your public static void main(String args[]) method. It depends on what mode you are using

- **Cluster mode**: In case of cluster mode driver would run on one of the machine/nodes which is part of the cluster itself. For example, in case of YARN, driver runs inside an application master process which is managed by YARN on the cluster and client would go away once the application initiated.
- **Client mode**: Driver runs in the client process (part of the same process which initiate your application)

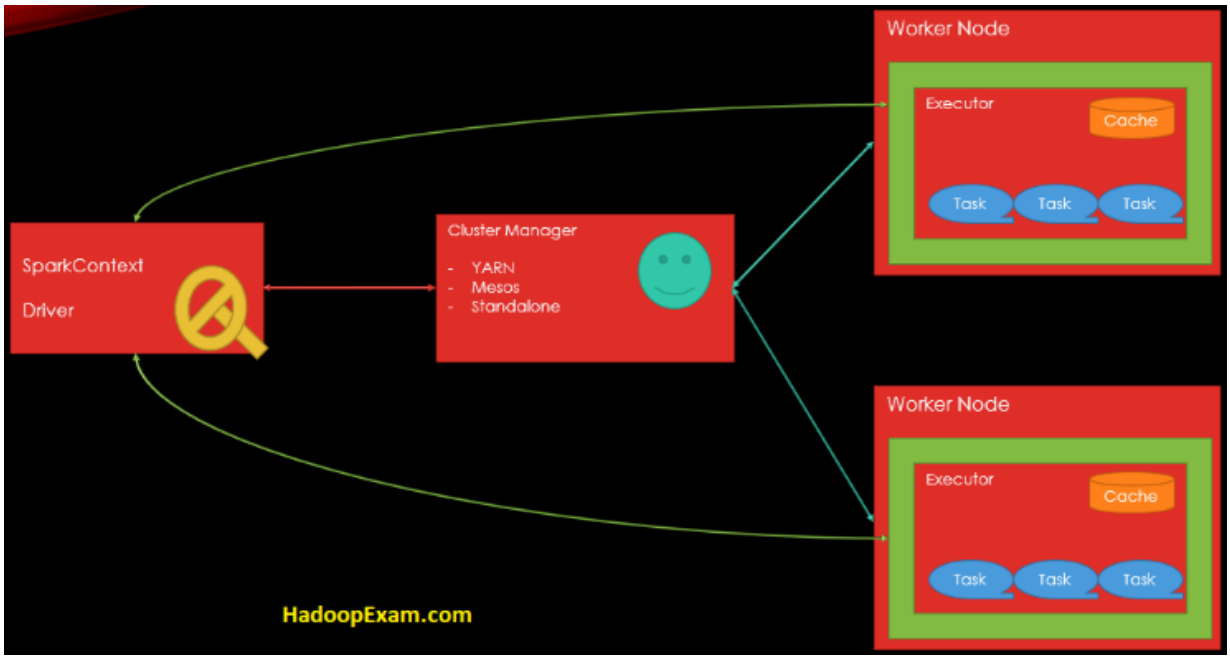
Command line example for submitting application using cluster and client mode are below.

```
spark-submit
  --master yarn \
  --deploy-mode cluster \
  --driver-memory 12g \
  --executor-memory 2g \
  --executor-cores 1 \
  --queue nameofthequeue \
  hePySparkApp.py
```

Similarly, for the client mode, you can start spark shell as below.

```
spark-shell --master yarn --deploy-mode client
```

So, we can say that SparkContext is the coordinator for the submitted application (which runs under the Driver process).



As you can see in above image, once the resources are acquired for running the application tasks, then SparkContext would submit the tasks on the executor. One Spark application is equivalent to having one SparkContext object.

Remember:

- Driver program keep listening for the incoming connections for the executors until the application runs. Driver must be always available on the network.
- You should always run the Drive process near the worker nodes, if possible.
- If you wanted to explicitly assign the memory for driver process in your application then you have to use it like this "spark-submit –driver-memory 4g"
- Your entire application state is maintained in the Driver process.
- Driver process connects with the Cluster manager to get the resources from the Cluster, once it gets the physical resources from cluster. Same would be used to launch the executors.
- Driver also stores the metadata about the currently running application and same application you can see in the web UI.

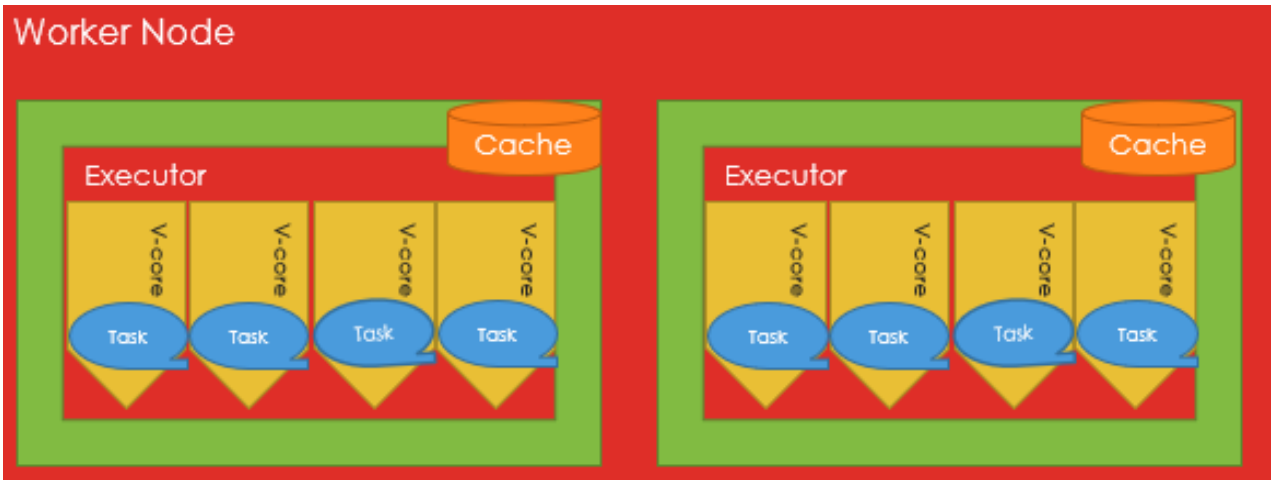
Executor

- Each application has its own executor processes and remain up only until your application and tasks runs. And make sure your application runs isolated and does not interact with any other application already submitted. Tasks created for each application runs in a different JVM.

- Data between the running application cannot be shared directly.
- If you want to share the data between the running application then first that data needs to be stored on the external storage and then only application can share the data.
- All the tasks which needs to be executed on the executor are assigned by the driver.
- Executor would run the tasks and report back their state and result to the driver.
- Executor tries to store applications data in memory, if there is not enough memory then it will store the same on the Disk.

Explain how parallelization allows Spark to improve speed and scalability of an application.

In Spark Core or Slots represent number of threads available to each executor process. As you can see in the below image



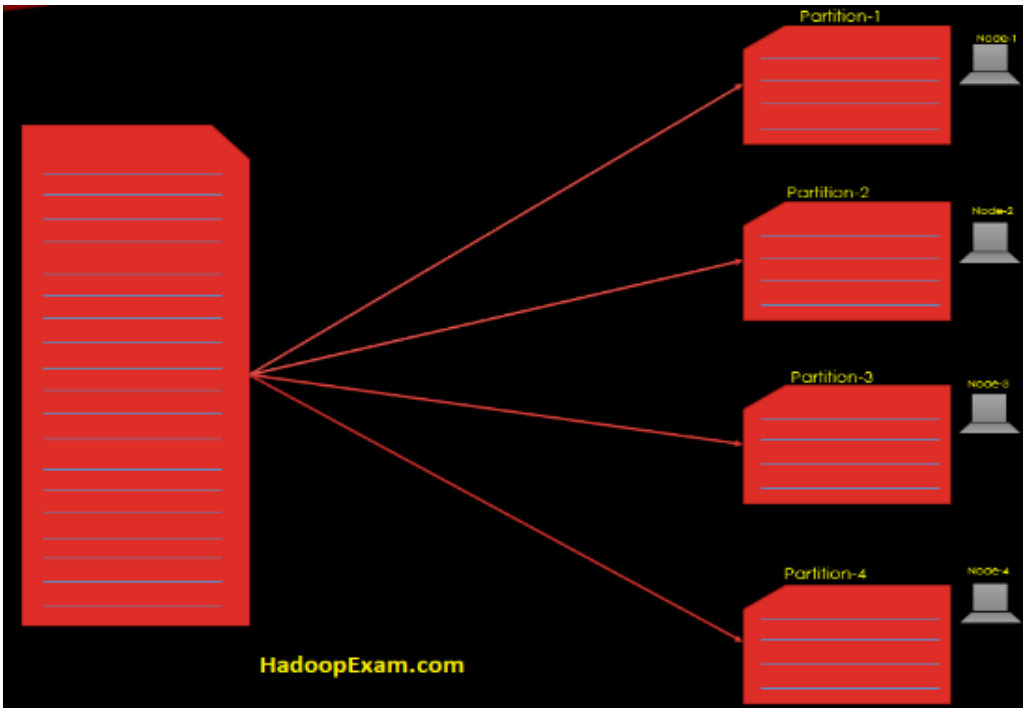
There are 4 cores available to each executor on the worker node. Hence, there are total 8 cores on this given worker node which can execute at the max 8 tasks in parallel. Each individual core at the max can have single thread running. You can use these terms core, thread and slot interchangeably, only for Spark.

Partitions

Let's understand things conceptually at first. If you have huge volume of data which may not fit on a single machine then you split them in multiple parts. For example, you have a Data size of 5TB and your computer has the capacity of 1 TB then you need to divide them into 5 parts (1 TB for each machine). Also, you need to sort this 5 TB data, then

you would be sorting independently on each machine in parallel. This each one TB data you can consider as a partition, so we can say there are in total 5 partitions.

In case of Apache Spark size is not the reason to partition the data but rather you want to parallelize the work, so that your application runs fast enough. Even Spark keeps the data in the memory of each node, similar thing is depicted in the below block diagram. Where each node has 1 partition from the huge data file. And can be worked upon parallelly.



In Spark cluster

- As a developer you can decide how many partitions should be created and also configurable.
- If number of partitions is very few then concurrent computations is also affected and you would have higher latency for your job. And cluster resources not properly utilized.
- Number of partitions you should decide based on the number of cores in your entire cluster. Because at a time a single can work on only one partitions.
- Suppose your cluster has 100 cores then at the max you could have 100 tasks running concurrently in the cluster. So, if you have more than 100 let's say 150 partitions then 50 partitions have to wait for the cores to get free.
- A single RDD partition cannot span more than one node.
- All the tuples in the same partitions are guaranteed to be on the same machine.

Partitions Strategy: There are two partition strategy which are popular

- Hash partitioning
- Range Partitioning

Which partition strategy is best fit decided based on the following factor?

- Number of cores
- Size of the file

Spark Execution

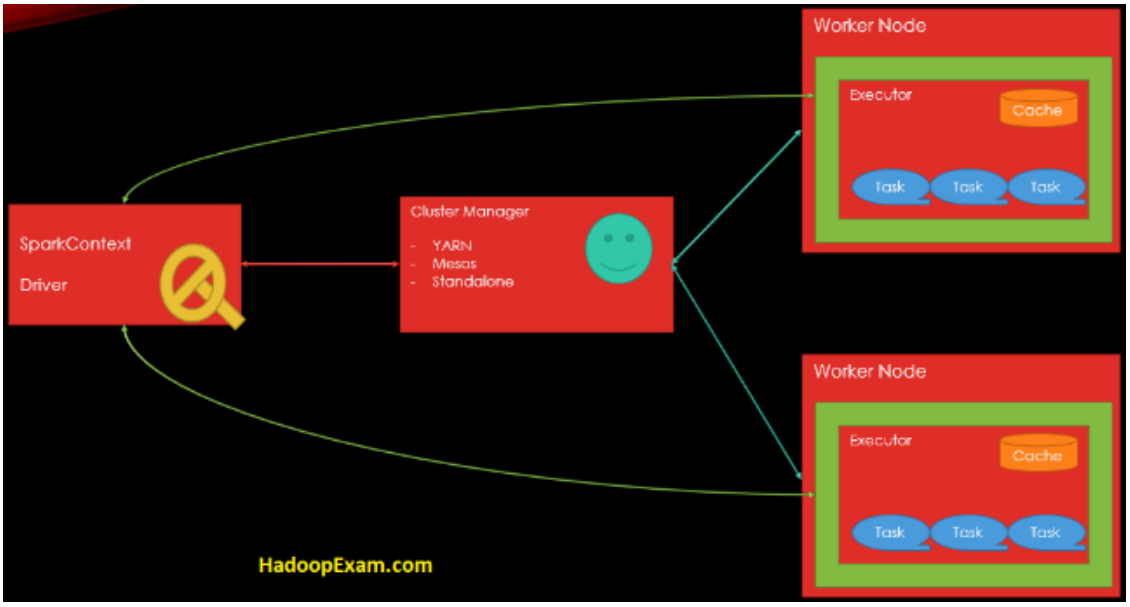
Spark's execution model and the breakdown between the different elements

- Jobs
- Tasks
- Stages

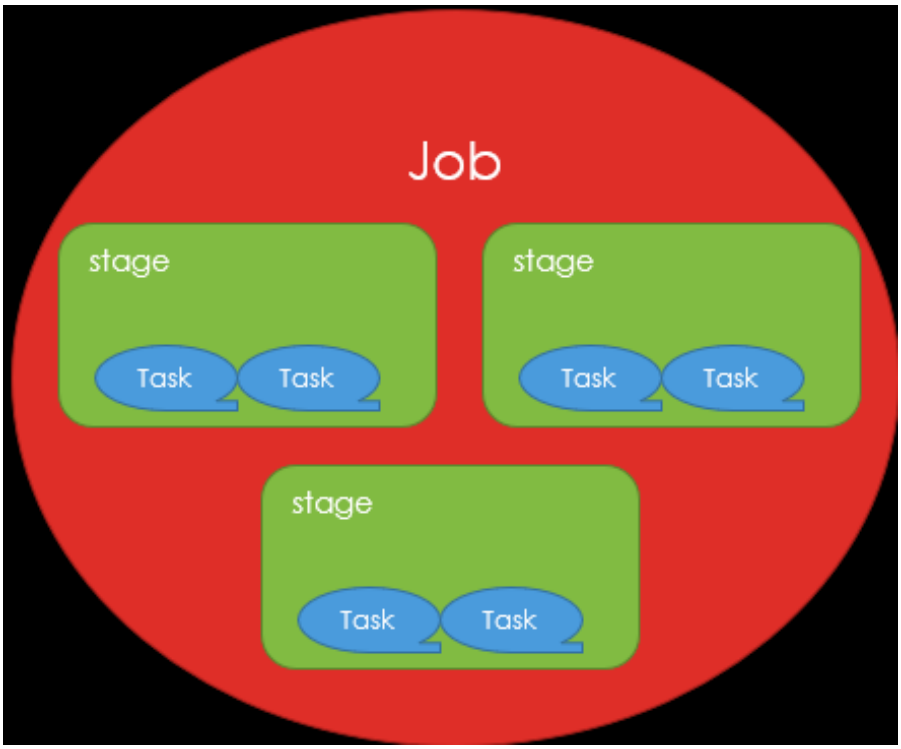
Let's learn few terminologies related to the Spark Architectural components.

- **Application**: Your entire program you write using Spark either in Python, Scala, Java or R. Would be called an application (single application). Your one application would have one SparkContext (means one Driver process) and many executors in the cluster. Below diagram represent one single Spark application.

  You would be bundling your entire application is one JAR file (in case of Scala) or create a Python file. While bundling JAR's keep in mind you should not include Hadoop and Spark libraries, because those would be already bundled by the runtime env.



- **Driver**: This is where your main () method would be executed and SparkContext would be created. You can consider Driver as a master for your Spark Application. It also hosts the Web UI for your application. Similarly, tasks schedulers reside on the Driver which schedule tasks that needs to be executed on executors. If your Spark Application got crashed then your Spark application would also be killed.
- **Cluster Manager:** This is the component which is responsible for managing the resources in the entire cluster example YARN, Mesos or Spark Standalone cluster manager. This is used to make sure that no application starves because lack of cluster resource availability.
- **Deployment Modes:** This is not a component but the strategy where your Driver program should run, in case of cluster mode driver process would run on one of the nodes in the cluster and in case of client mode driver process would run outside the cluster.
- **Worker Nodes:** These are the nodes in your cluster which would be running executor process for your application and finally task would be executed on the executor process.
- **Executor**: This is a process which is launched on the Worker Nodes and runs the tasks for your submitted applications and other than this it also keeps the data in memory or disk storage. Each application has its own executors.



- **Task:** This is the smallest unit of work from your application which would be send to executor specific to an application. Individual tasks run the computation on the a single RDD partitions, as we have multiple partitions of an RDD in a cluster. So various same tasks run on the different partitions of the RDD in the cluster.

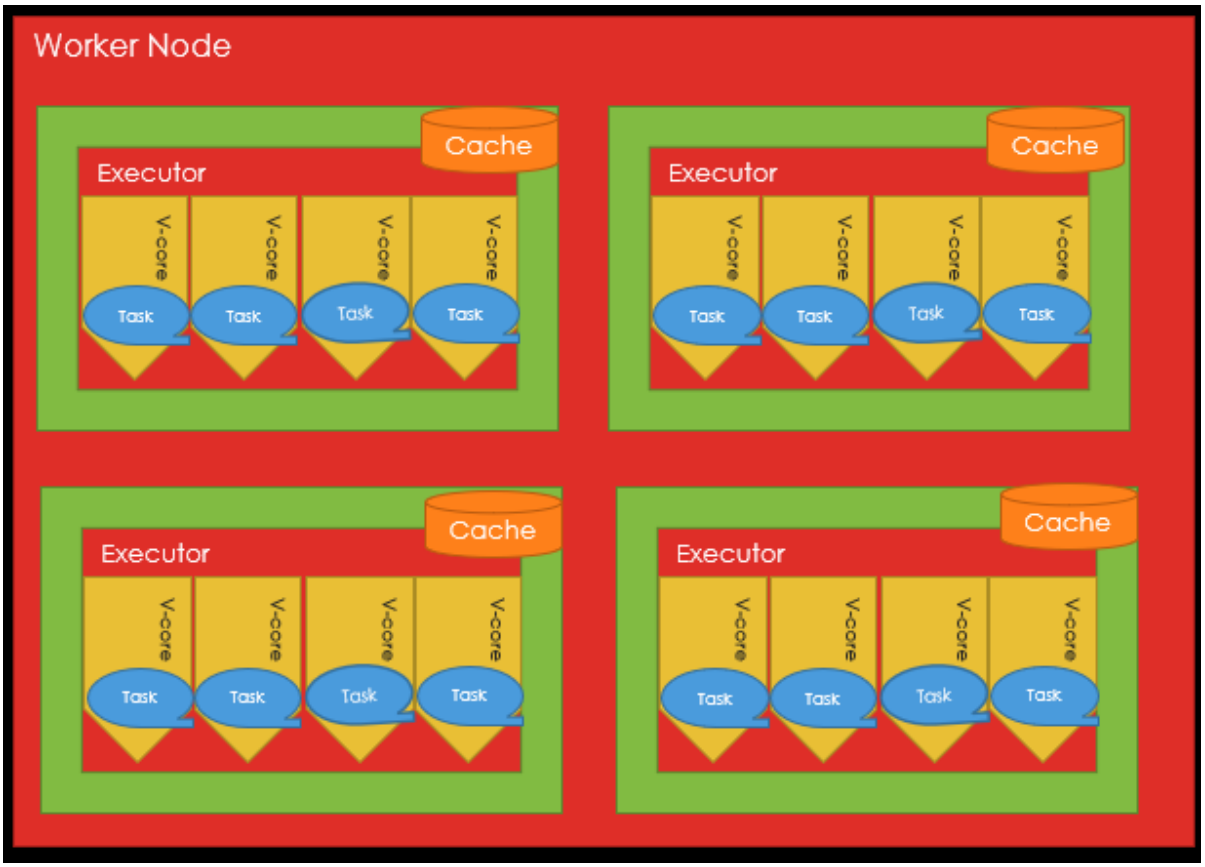
  A task would always be part of the single stage and work on the single partitions only. Before initiating new stage all the tasks in a stage should be completed. You can also say that task is a command which would be sent from the driver to the executor by serializing the Function object. And executor de-sterilizes this and execute it.

- **Job**: A Job can have multiple tasks (usually it has) which gets Spawned when Spark action is initiated (e.g. collect, count, show etc.). This is very important to remember that Job would be created only when action is executed. Whenever a job is executed, an execution plan is created according to the lineage graph.
  - So, whenever you think about the job, first think how many actions are there in your Spark application.
  - It may be running jobs concurrently.

- **Stages**: A job is sliced into stages, and a stage is a parallel task (one task per partitions), Spark job is sliced into the stages, stage can run on the partitions of the single RDD. For each shuffle new stage would be created. Shuffle introduce a

barrier where stages/tasks has to wait previous stage to be finished. It means in a single stage you would not have shuffle. Again, we can say that stage is a collection of tasks, same process runs against different subset of data which is represented as partition. In each stage number of tasks= number of partitions in that stage.

Each stage can be executed on multiple executors and single executor can run multiple v-cores. Each v-core can execute exactly one tasks at a time.



Let's see the below Spark example conceptually to understand more all these terms

- **Step-1**: You would be loading two data files (HECourse.csv and HELearner.csv) as two separate DataFrames
- **Step-2**: Independently replace empty values with Nan in both the DataFrame
- **Step-3**: Join both the DataFrame using the common column.
- **Step-4**: Apply another map function on the data and filter all the Learners where is less than 5000
- **Step-5**: Finally save the DataFrame as Parquet file in HDFS

In above case lets go step by step to understand the stages

- In Step-1 Each file loaded independently, hence there would be two stages created.
- Next stage would be created when shuffle is introduced and that is when join is initialized.
- And all other follow-up operations can also be part of the same stage, because they would be done sequentially and there are no benefits of creating additional stages. So, Saving the data would be part of the same stage. Hence, there would be in total 3 stages.
- How would you calculate total number of tasks in this case?
  Do the sum for all the stages (individual stage X Number of partitions in that stage)