Introduction to Apache Spark (PySpark)

Apache Spark

- Apache Spark is an open-source unified analytics engine for large-scale data processing.
- Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs:
  - MapReduce programs read input data from disk
  - Map a function across the data
  - Reduce the results of the map
  - Store reduction results on disk.
- Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.
- Apache Spark requires a **cluster manager** (YARN / Mesos / Kubernetes) and a **distributed storage system** (HDFS / Mapr-FS / S3 / Cassandra).
- Spark also supports a **pseudo-distributed local mode**, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.

Why Spark?

Speed	Allows applications in Hadoop to run: ➢ 100 times faster in memory ➢ 10 times faster on disk
Ease of Use	Allows to quickly write applications in Java, Scala and Python
Advanced Analytics	Supports SQL queries, streaming data and advanced analytics

Spark processed 100 TB data in 23 Mins!!!

Gray sort competition: Winner Spark-based (previously MR)

	Hadoop MR Record	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark-based System 3x faster with 1/10 # of nodes

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)

http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Spark Modules

Spark Core

Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides an RDD (Resilient Distributed Dataset) and in-memory computing capabilities.

Spark SQL and DataFrame

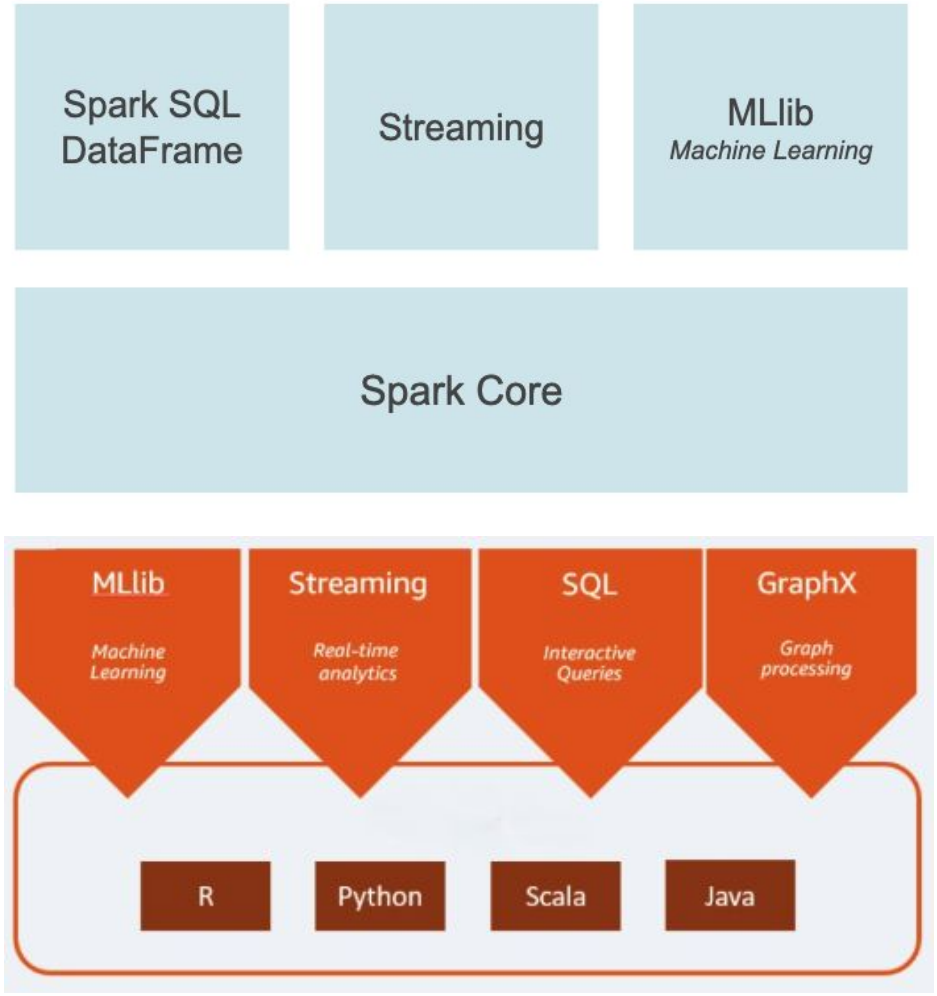
Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrame and can also act as distributed SQL query engine.

Streaming

Running on top of Spark, the streaming feature in Apache Spark enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics.

MLlib

Built on top of Spark, MLlib is a scalable machine learning library that provides a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

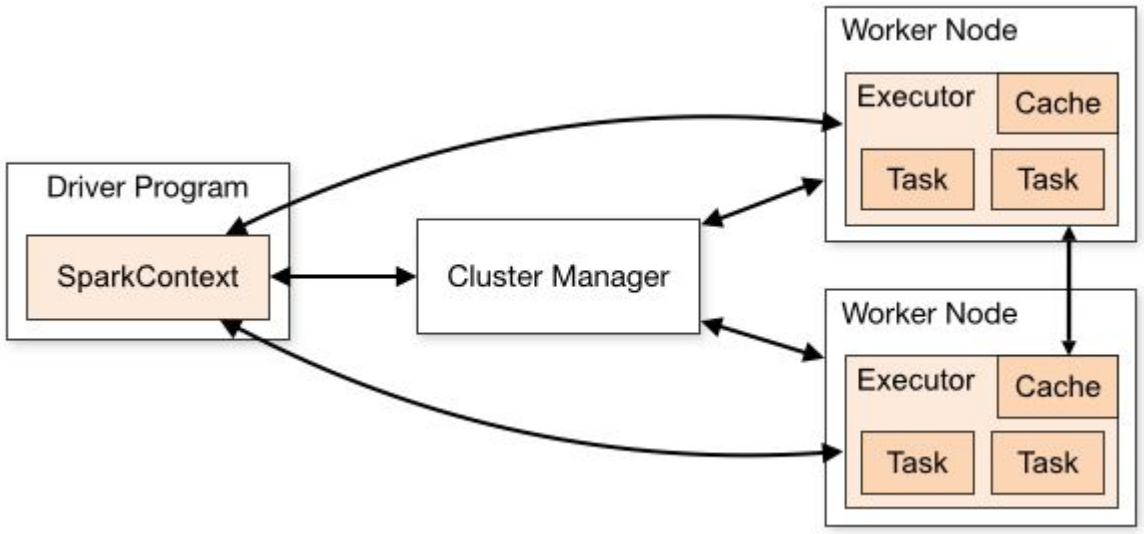


Spark Core

Spark Core is the base engine for large-scale parallel and distributed data processing. It is responsible for:

- Memory management and fault recovery
- Scheduling, distributing and monitoring jobs on a cluster
- Interacting with storage systems

Spark introduces the concept of an **RDD (Resilient Distributed Dataset),** an immutable fault-tolerant, distrib ı in parallel.



http://spark.apache.org/docs/latest/cluster-overview.html

Spark API

History of Spark APIs

Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

Fast/efficient internal representations

Internally rows, externally JVM objects

Almost the "Best of both worlds": **type safe + fast**

But slower than DF
Not as good for interactive analysis, especially Python

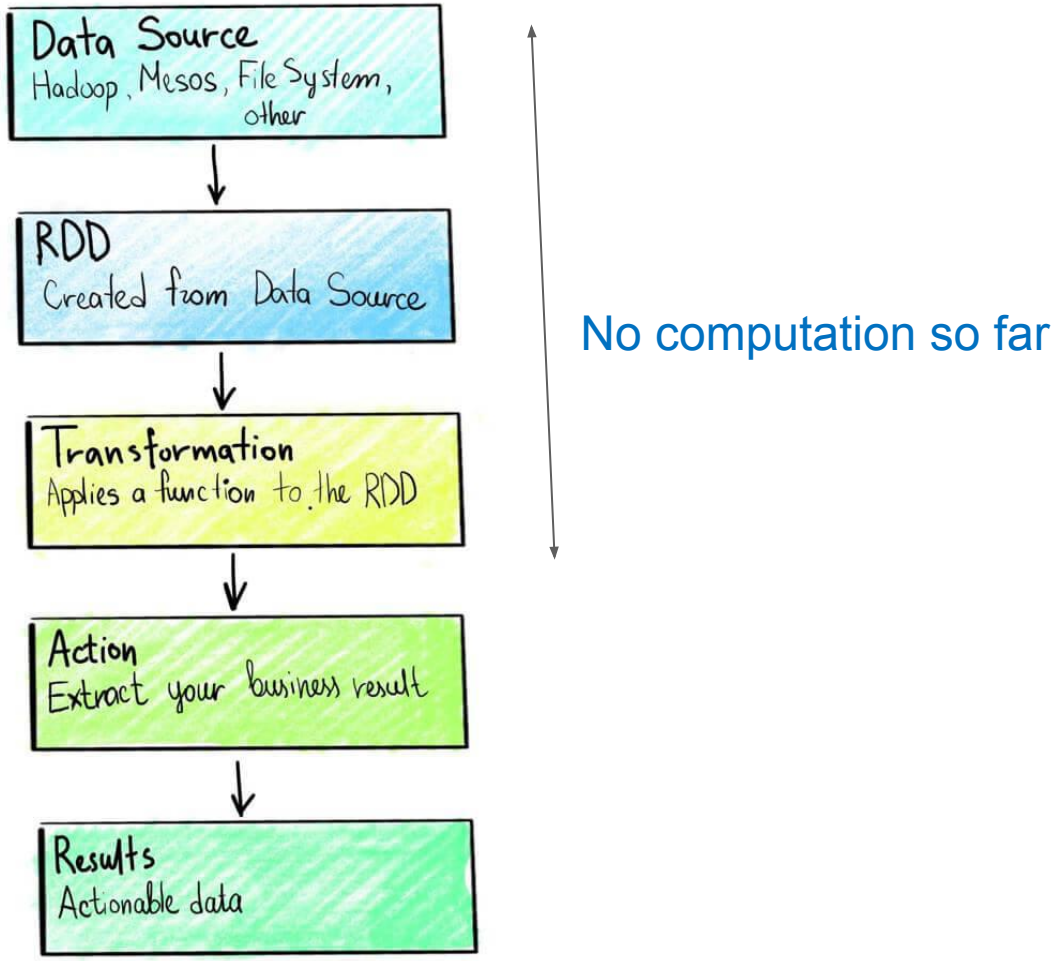
databricks

RDD - Operations

An RDD can contain any type of object and is created by loading an external dataset or distributing a collection from the driver program. RDDs support two types of operations:

- **Transformations** are operations (such as map, filter, join, union, and so on) that are performed on an RDD and which yield a new RDD containing the result.

- **Actions** are operations (such as reduce, count, first, and so on) that return a value after **running a computation** on an RDD.

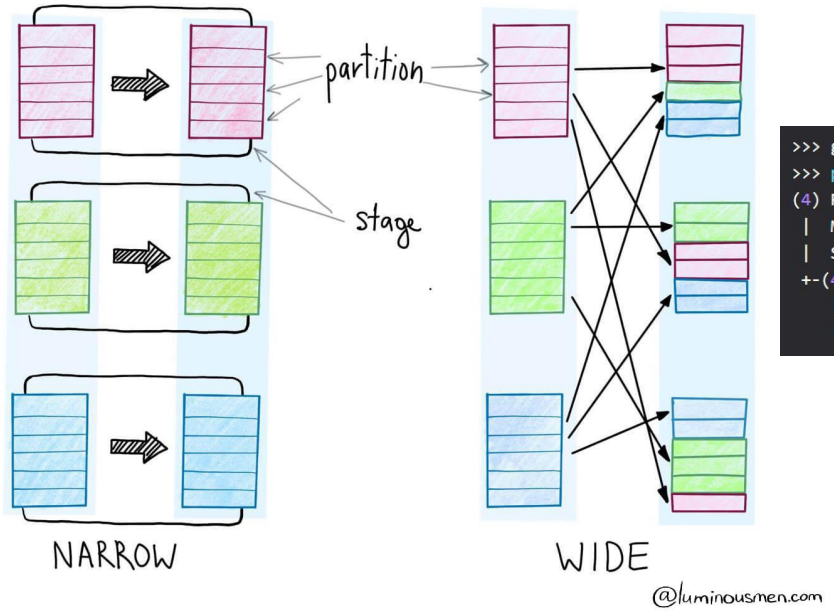


Transformations

- The result of applying this operation to RDD is new RDD. As a rule, these are operations that somehow transform the elements of the given data.

- Transformations by their nature are lazy, i.e. when we call some operation on RDD, it is not executed immediately. Spark keeps a record of which operation is called (via DAG, we will talk about it later).

```
>>> filteredRDD = rdd.filter(lambda x: x > 10)
>>> print(filteredRDD.toDebugString())  # to see the execution graph; only one stage is cre
(4) PythonRDD[1] at RDD at PythonRDD.scala:53 []
 |  ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:195 []
>>> filteredRDD.collect()
[11, 12, 13, 14, 15, 16, 17, 18, 19]
```




```
>>> groupedRDD = filteredRDD.groupBy(lambda x: x % 2)  # group data based on mod
>>> print(groupedRDD.toDebugString())  # two separate stages are created, because
(4) PythonRDD[6] at RDD at PythonRDD.scala:53 []
 |  MapPartitionsRDD[5] at mapPartitions at PythonRDD.scala:133 []
 |  ShuffledRDD[4] at partitionBy at NativeMethodAccessorImpl.java:0 []
 +-(4) PairwiseRDD[3] at groupBy at <ipython-input-5-a92aa13dcb83>:1 []
    |  PythonRDD[2] at groupBy at <ipython-input-5-a92aa13dcb83>:1 []
    |  ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:195 []
```

Actions

- Actions are applied when it is necessary to materialize the result — save the data to disk, write the data to a database or output a part of the data to the console. The collect operation that we have used so far is also an action — it collects data.

- The actions are not lazy — they will actually trigger the data processing. Actions are RDD operations that produce values that are not RDD.

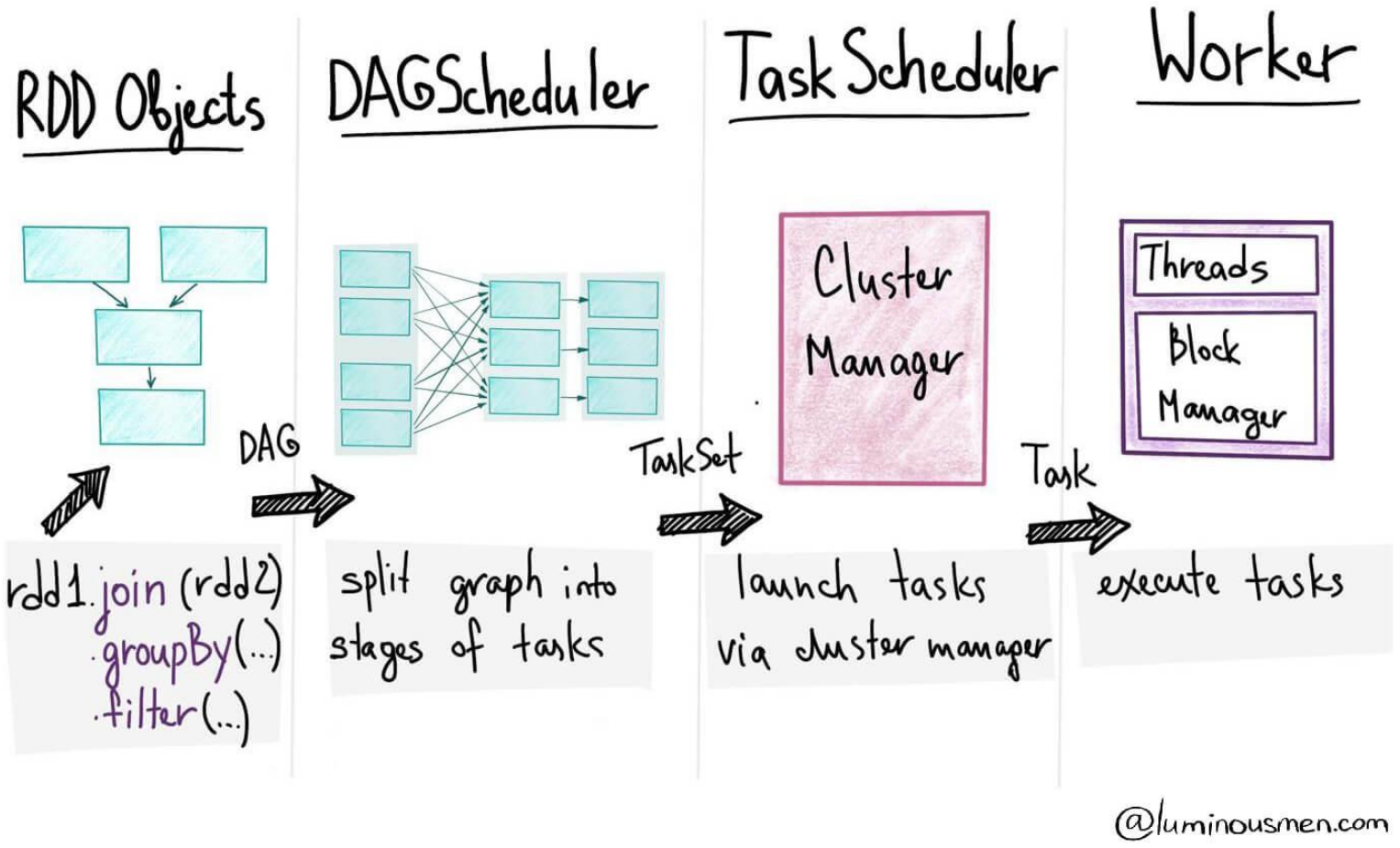
```
>> filteredRDD.reduce(lambda a, b: a + b)
135
```

DAG

Spark defines tasks that can be computed in parallel with the partitioned data on the cluster.

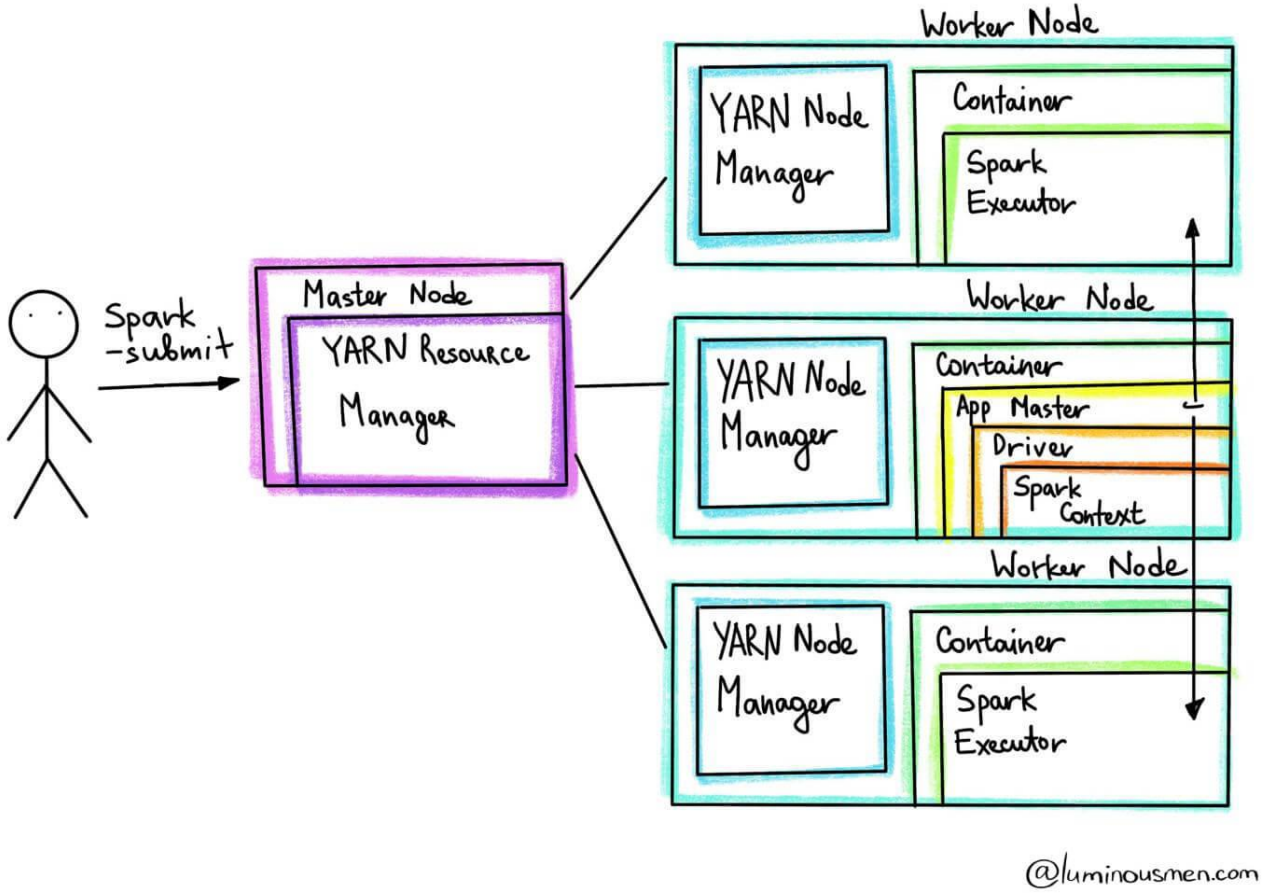
With these defined tasks, Spark builds a logical flow of operations that can be represented as a directional and acyclic graph, also known as **DAG** (Directed Acyclic Graph), where the node represents an RDD partition and the edge represents a data transformation.

Spark builds the execution plan implicitly from the application provided by Spark.



Anatomy of Spark Application

- Driver
- Application Master
- Spark Context
- Resource Manager (Cluster Manager)
- Executors



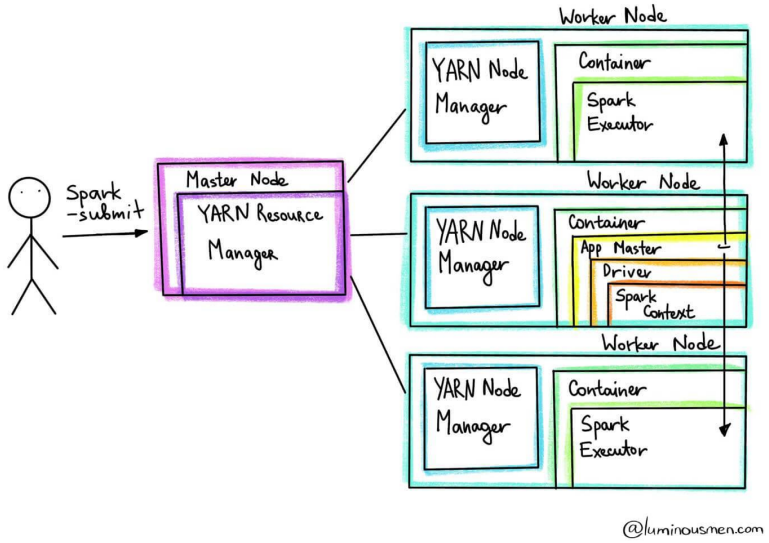
Anatomy of Spark Application

- The **Driver** (aka driver program) is responsible for converting a user application to smaller execution units called **tasks** and then schedules them to run with a cluster manager on executors. The driver is also responsible for executing the Spark application and returning the status/results to the user.

- **Resource Manager** in a distributed Spark application is a process that controls, governs, and reserves computing resources in the form of containers on the cluster.

- **Application Master** is for negotiating resources with Resource Manager(s) and work with Node Manager(s) to perform and monitor application tasks.

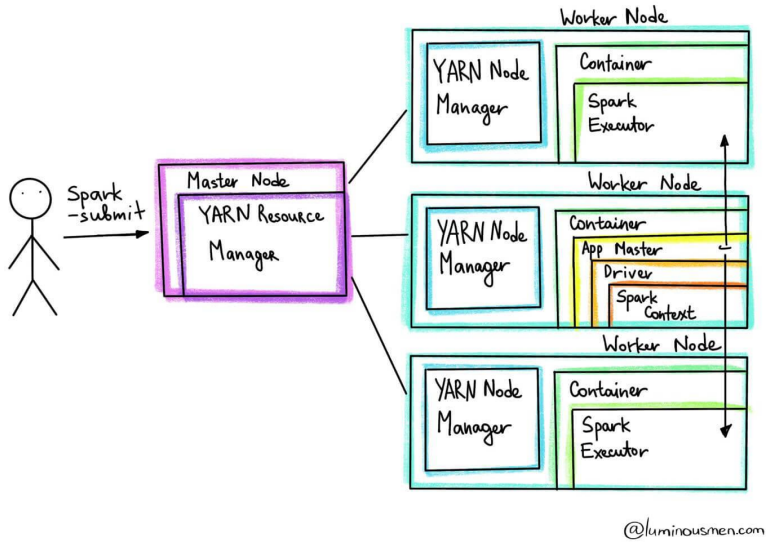
The Driver informs the Application Master of the executor's needs for the application, and the Application Master negotiates the resources with the Resource Manager to host these executors,



Anatomy of Spark Application

- **Spark Context** is the main entry point into Spark functionality. Spark Context is created by Spark Driver for each Spark application when it is first submitted by the user. It exists throughout the lifetime of the Spark application. Spark Context stops working after the Spark application is finished. For each JVM only one Spark Context can be active. You must stop() activate Spark Context before creating a new one.

- **Executors** are the processes at the worker's nodes, whose job is to complete the assigned tasks. These tasks are executed on the worker nodes and then return the result to the Spark Driver.



Hands-On Tutorials

- Installation: https://spark.apache.org/docs/latest/api/python/getting_started/install.html
- Notebook Setup on Linux : https://opensource.com/article/18/11/pyspark-jupyter-notebook
    - export SPARK_HOME=/home/ubuntu/miniconda3/envs/spark/lib/python3.7/site-packages/pyspark
    - export PYSPARK_DRIVER_PYTHON=jupyter
    - export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
- Hands-On Example on Spark Shell: https://github.com/AlmaBetter-School/pyspark-examples
- Other Examples: https://github.com/apache/spark/tree/branch-2.4/examples/src/main/python

Concepts Review

Term	Meaning
Application	User program built on Spark. Consists of a driver program and executors on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

References

- https://luminousmen.com/post/spark-core-concepts-explained
- https://luminousmen.com/post/spark-anatomy-of-spark-application