Spark is a distributed processing engine, but it does not have its own distributed storage. It runs on top of out of the box cluster resource manager and distributed storage.

Spark core has two parts to it:

- Core APIs: The Unstructured APIs(RDDs), Structured APIs(DataFrames, Datasets). Available in Scala, Python, Java, and R.
- Compute Engine: Memory Management, Task Scheduling, Fault Recovery, Interacting with Cluster Manager.

Outside the Core APIs Spark provides:

- Spark SQL: Interact with structured data through SQL like queries.
- Streaming: Consume and Process a continuous stream of data.
- MLlib: Machine Learning Library. However, I wouldn't recommend training deep learning models here.
- GraphX: Typical Graph Processing Algorithm.

All the above four directly depend upon spark core APIs for distributed computing.

Advantages of Spark

- Spark provides a unified platform for batch processing, structured data handling, streaming, and much more.
- Compared with map-reduce of Hadoop, the spark code is much easy to write and use.
- The most important feature of Spark, it abstracts the parallel programming aspect. Spark core abstracts the complexities of distributed storage, computation, and parallel programming.

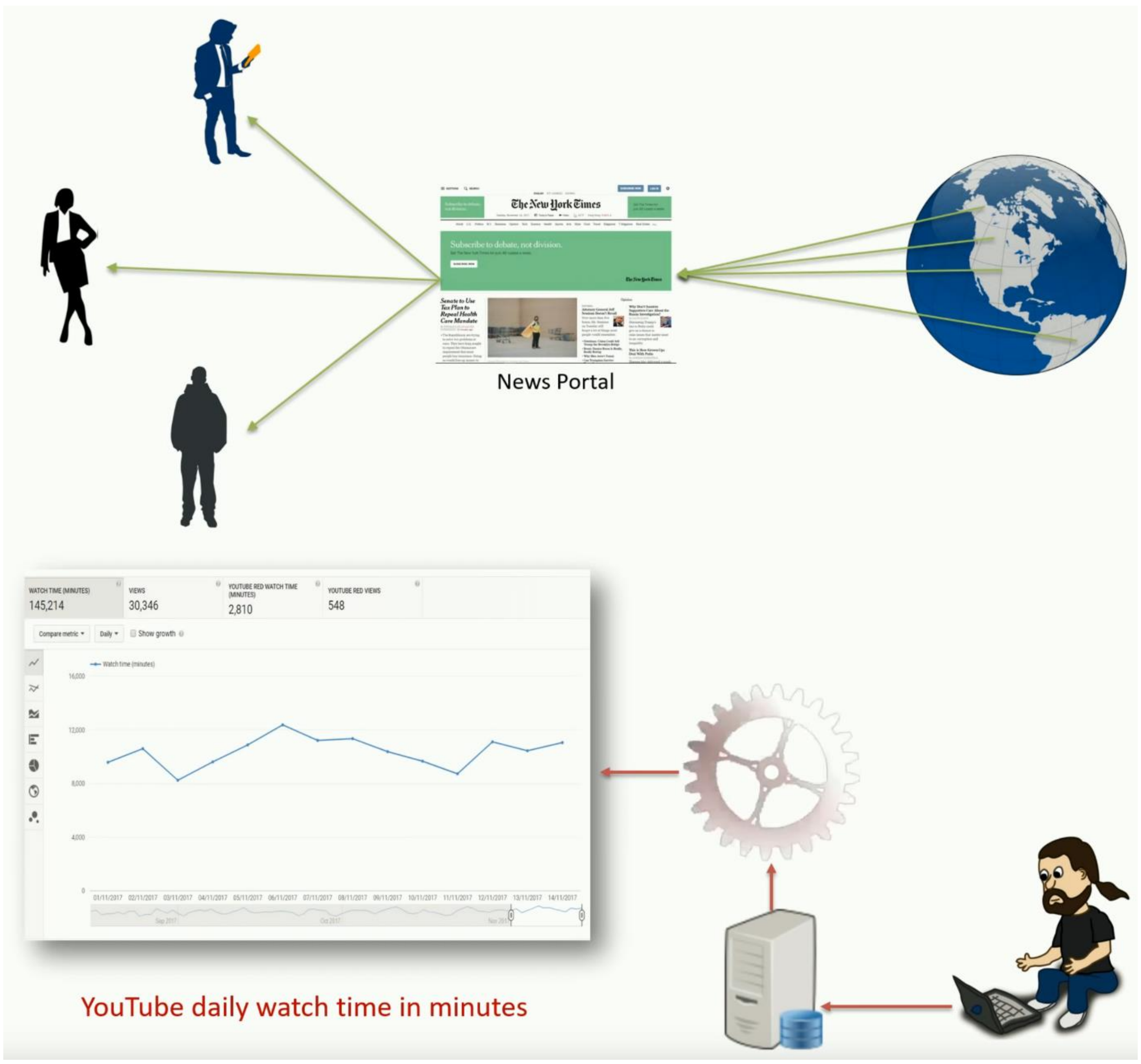
One of the primary use cases of Apache Spark is large scale data processing. We create programs and execute them on spark clusters.

Executions of Program on a Cluster

There are primarily two methods to execute programs on spark cluster:

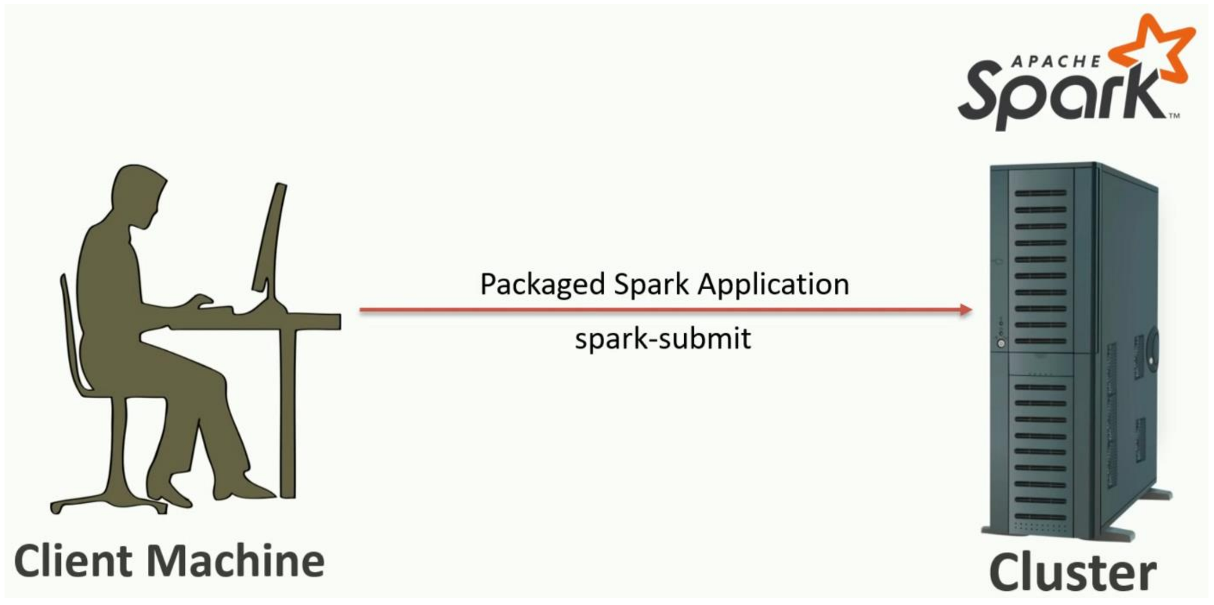
1. Interactive clients like *spark-shell*, *py-spark*, notebooks etc.
2. Submit a job.

Most of the development process happens on interactive clients, but when we have to put our application in production, we use Submit a job approach.

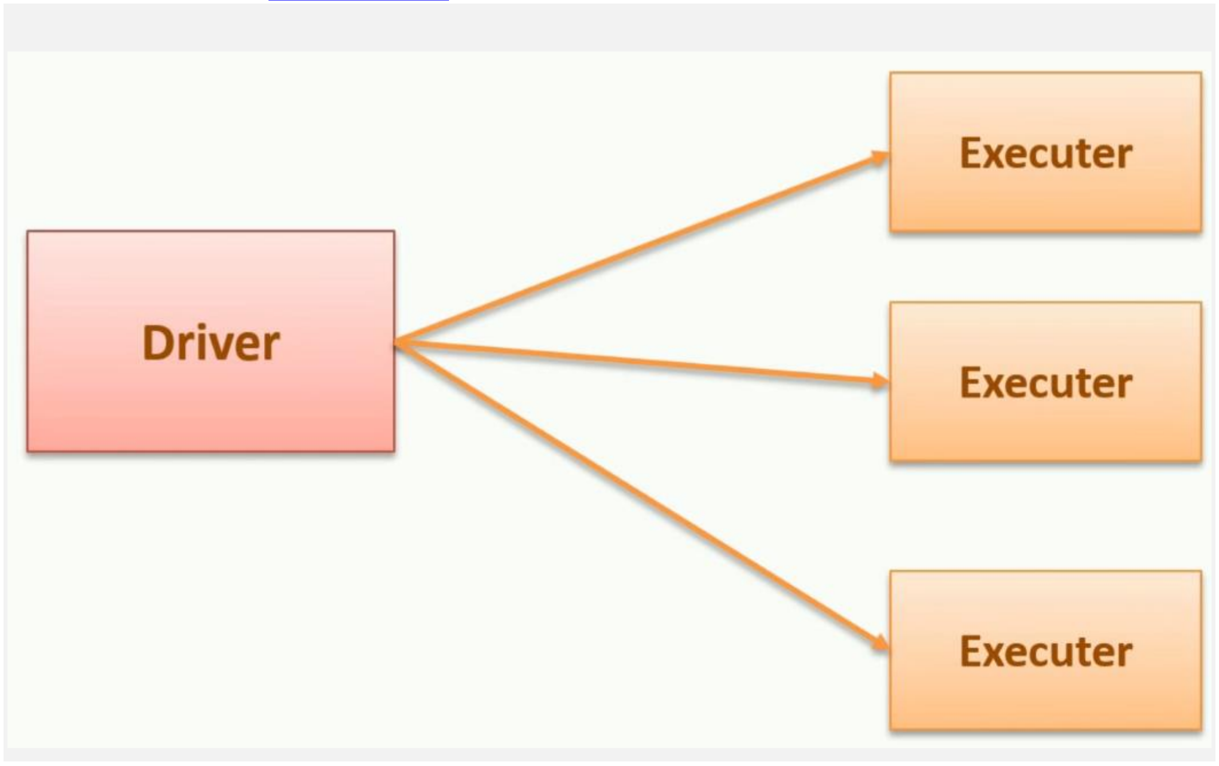


News Feed — Streaming Job. Daily YouTube analytics — Batch Job

For both long-running streaming job or a periodic batch job, we package our application and submit it to Spark cluster for execution.



Spark, a distributed processing engine, follows the master-slave architecture. In spark terminology, the master is the *driver*, and slaves are the *executors*.



Driver is responsible for:

1. Analyzing
2. Distributing.
3. Monitoring.
4. Scheduling.
5. Maintaining all the necessary information during the life time of the spark process.

Executors are only responsible for executing the part of the code assigned to them by the driver and reporting the status back to the driver.

Each spark process would have a separate driver and exclusive executors.

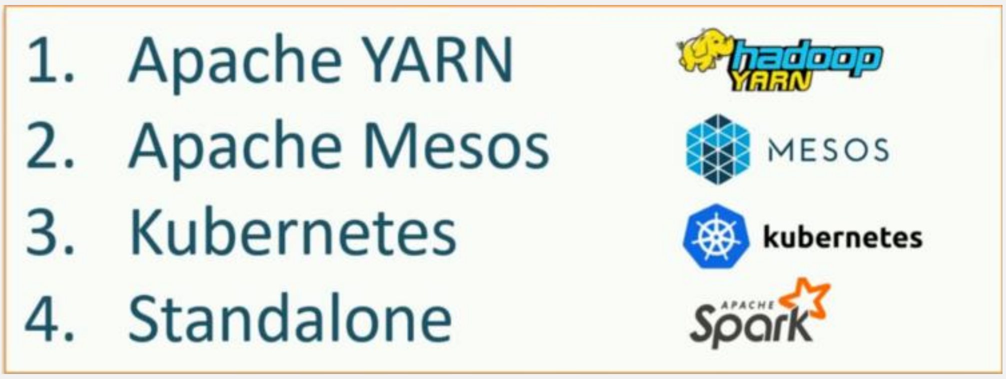
Modes of execution

1. **Client Mode:** The driver is the local VM, where you submit your application. By default, spark submits all applications in client mode. Since the driver is the master node in the entire spark process, in production set up, it is not advisable. For debugging, it makes more sense for using client mode.

2. **Cluster Mode:** The driver is one of the executors in the cluster. In the spark-submit, you can pass the argument as follows:

```
--deploy-mode cluster
```

Cluster Resource Manager



1. Apache YARN
2. Apache Mesos
3. Kubernetes
4. Standalone

hadoop YARN
MESOS
kubernetes
APACHE Spark

Yarn and Mesos are the commonly used cluster manager.

Kubernetes is a general purpose container orchestrator .

Note : Spark on Kubernetes is not production ready.

Yarn being most popular resource manager for spark, let us see the inner working of it:

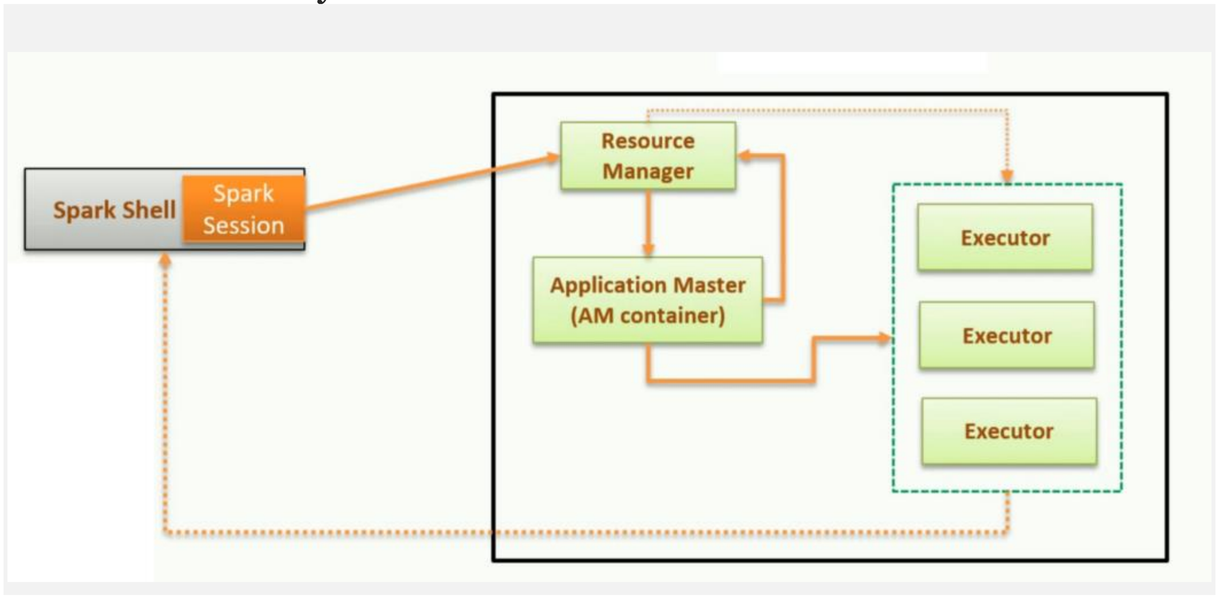
In a client mode application the driver is our local VM, for starting a spark application:

Step 1: As soon as the driver starts a spark session request goes to Yarn to create a yarn application.

Step 2: Yarn Resource Manager creates an Application Master. For client mode, AM acts as an executor launcher.

Step 3: AM would reach out to Yarn Resource manger to request for further containers.

Step 4: Resource Manager would allocate new containers, and AM would start executors in each container. After that, executors directly communicates with the driver.



Note: In the cluster mode the driver starts in the AM.

Executor and Memory Tuning

Hardware — 6 Nodes, and Each node 16 cores, 64 GB RAM

Let us start with the number of cores. The number of cores represents concurrent task an executor can run. Research has shown that any application having more than 5 concurrent task leads to a bad show. Hence, I would suggest sticking with 5.

1 Core an 1 GB RAM is needed for OS and Hadoop daemons. Hence we are left with 63 GB Ram and 15 Cores.

For 15 Cores, we can have 3 executors per node. That gives us 18 executors in total. AM Container requires 1 executor. Hence we can get 17 executors.

Coming to memory, we get 63/3 = 21 GB per executor. However, small overhead needs to be accounted for while calculating the full memory request.

```
Formula for that over head = max(384, .07 *
spark.executor.memory)Calculating that overhead = .07 * 21 =
1.47
```

Hence the memory comes down to approximately 19 GB.

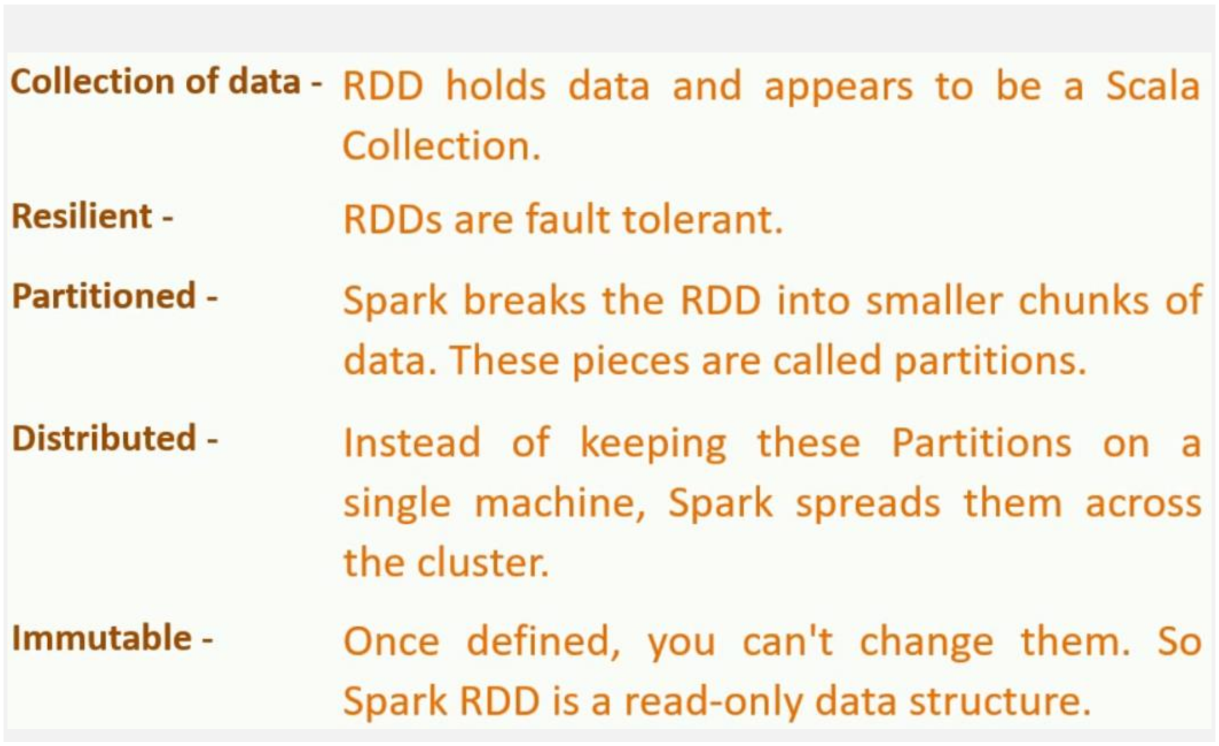
Hence the system comes to :

```
--num-executors 17 --executor-memory 19G --executor-cores 5
```

Spark Core

Now we look at some core APIs spark provides. Spark needs a data structure to hold the data. We have three alternatives RDD, DataFrame, and Dataset. Since Spark 2.0 it is recommended to use only Dataset and DataFrame. These two internally compile to RDD itself.

These three are resilient, distributed, partitioned and immutable collection of data.



Collection of data -	RDD holds data and appears to be a Scala Collection.
Resilient -	RDDs are fault tolerant.
Partitioned -	Spark breaks the RDD into smaller chunks of data. These pieces are called partitions.
Distributed -	Instead of keeping these Partitions on a single machine, Spark spreads them across the cluster.
Immutable -	Once defined, you can't change them. So Spark RDD is a read-only data structure.

Task: The smallest unit of work in Spark and is performed by an executor.

Dataset offers two type of actions:

- **Transformations:** Creates new Dataset from the existing one. It is lazy, and data remains distributed.
- **Action:** Action returns data to the driver, non-distributed in nature. Action on a Dataset triggers a job.

Shuffle and Sort: Re-partitioning of a dataset for performing actions on it. It is an abstraction in spark, and we do not need to write code for it. This activity requires a new stage.