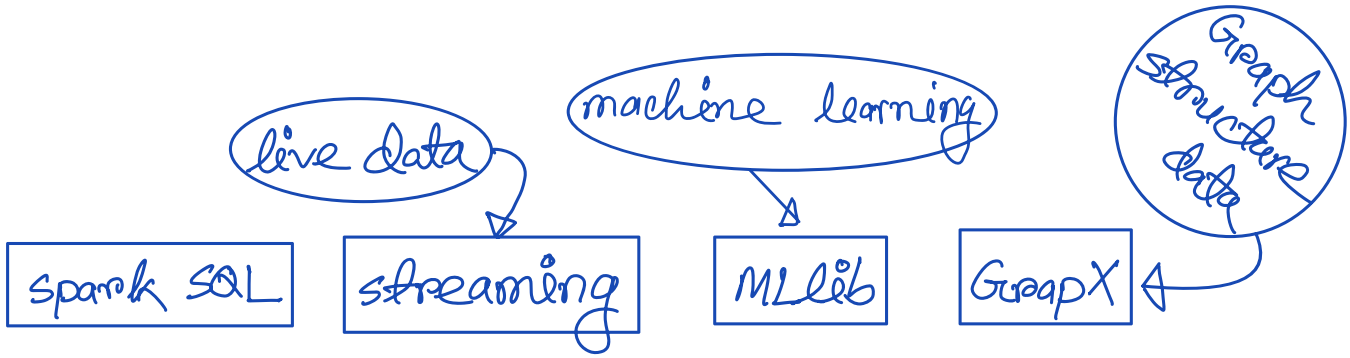
Course: Apache PySpark by Example

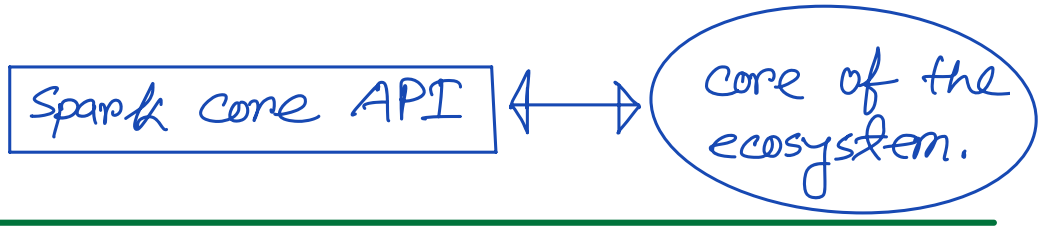
Practice environment: Google Colaboratory.
Spark 2.3

Why Pyspark:

1. Speed.
2. Easy to development.
3. Couple of language APIs.
4. Own eco-system.

Apache Spark Eco-system:





---

standalone

Responsibilities of Spark Core API:

(i) Task scheduling.

(ii) Memory management.

(iii) Fault recovery.

(iv) Interacting with storage system.

Responsibilities of Spark SQL and dataframe:

(i) Allow dataframe programming abstraction to execute query and make visualization.

(ii) Spark SQL act as a distribute SQL query engine to intermix SQL's queries with the programmatic data manipulation to solve complex analytics with SQL.

Responsibilities of Spark streaming:

(i) Process real time data.

(ii) Analyze streaming and historical data.

(iii) Use similar code for batch data and real-time data.

Responsibilities of MLlib:

(i) Provide scalable machine learning algorithm.

(ii) Works in memory

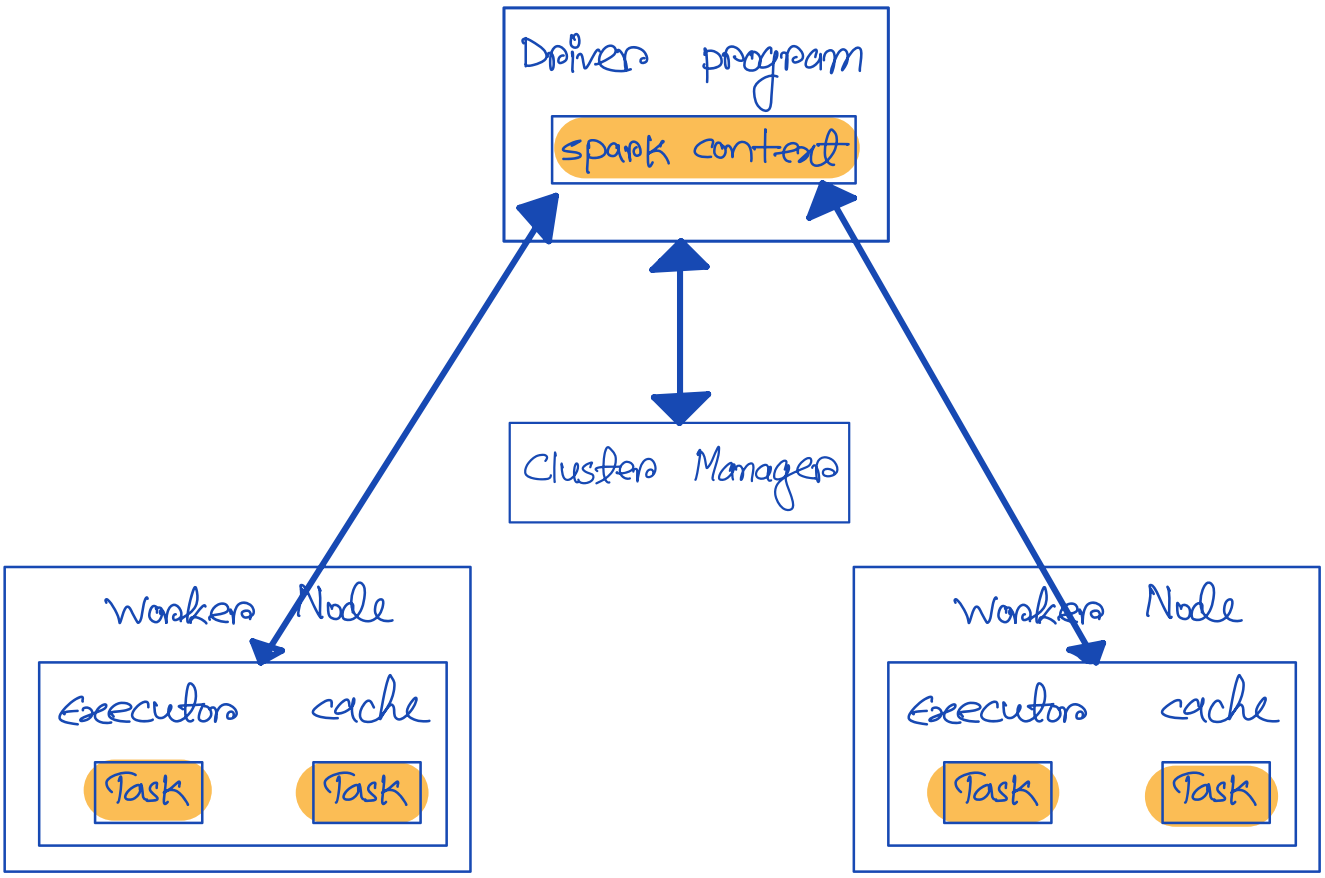
Responsibilities of GraphX:

(i) Enable working with graph-structure data.

(ii) Introduce a new graph abstraction, the directed multigraph.

(iii) Useful for visualize graph like social network.

Spark Components:



- Different types of cluster managers:

  (I) standalone. (II) Apache Mesos.

  (III) Hadoop YARN (IV) Kubernets.

- Spark workflow:

  Create spark session.

  → in background Pyspark uses Py4J to launch a JVM and creates a JAVA spark context.

Partitions: As Spark is a distributed system, we want the workers to work in parallel, and that's why Sparks need to break the data into chunks or partitions.

Actions: Three types of actions perform in spark.

(i) View data (.show()): fetch data to console.

(ii) collect data (.collect()): collect data for drivers.

(iii) write data (.write.format()): to write output sources.

Basic commands:

`!wget url...` : to download the file from remote source.

`!ls` : to see the file stored in the directory.

`!mv` : for renaming files.

This course focus on two major APIs:

① Dataframe API high-level API	② Resilient Distributed Dataset (RDDs) Low-level API

`df.limit(n)`: returns a new dataframe whereas

`df.head(n)`: returns an array

`df.printSchema()`: returns the schema of the dataset.

`df.column1` or `df['column1']` : to access a column.

`df.select('column1','column2').show(3)`

`df.withColumnRenamed('Existing Column', 'New column')`

`df.filter(col('colname') > 1)`
→ conditions

`df.select('column_name').distinct().show()` : shows all district elements from column_name.

`df.orderBy(col('column'))`

`df1.union(df2)` : concate two dataframe with same number of column and schema.

`df.count()` : return the number of rows.

Join Operation :

```
df.join(df2, df.column == df1.column, how = 'inner').
```

`df.cache()` command for lazy calculations.