**Architecture of Apache Spark:**

Apache Spark is an open source cluster computing framework for real-time data processing.

When compared to Hadoop, Spark performance is up to 100 times faster in memory and 10 times faster on disk.

The main feature of Apache Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries, and streaming data

Apache Spark has a well-defined layered architecture where all the spark components and layers are loosely coupled. This architecture is further integrated with various extensions and libraries.

Apache Spark Architecture is based on two main abstractions:

* Resilient Distributed Dataset (RDD)
* Directed Acyclic Graph (DAG)

In master node, we have the driver program, which drives our application. The code you are writing behaves as a driver program or if you are using the interactive shell, the shell acts as the driver program

Inside the driver program, the first thing we do is, we create a Spark Context. Assume that the Spark context is a gateway to all the Spark functionalities. It is similar to our database connection. Any command we execute in our database goes through the database connection. Likewise, anything we do on Spark goes through Spark context.

Now, this Spark context works with the cluster manager to manage various jobs. The driver program & Spark context takes care of the job execution within the cluster. A job is split into multiple tasks which are distributed over the worker node. Anytime an RDD is created in Spark context, it can be distributed across various nodes and can be cached there.

Worker nodes are the slave nodes whose job is to basically execute the tasks. These tasks are then executed on the partitioned RDDs in the worker node and hence returns back the result to the Spark Context.

Spark Context takes the job, breaks the job in tasks and distribute them to the worker nodes. These tasks work on the partitioned RDD, perform operations, collect the results and return to the main Spark Context.

If you increase the number of workers, then you can divide jobs into more partitions and execute them parallelly over multiple systems. It will be a lot faster.

With the increase in the number of workers, memory size will also increase & you can cache the jobs to execute it faster.

Part 2: Using Apache Spark, demonstrate what would you do to the data so that, in the future, you can run efficient analytics on it?

Ans: I use ETL tool to ingest the data from source to data lake and in data lake I partition the data by month and start\_station\_id for future analysis.