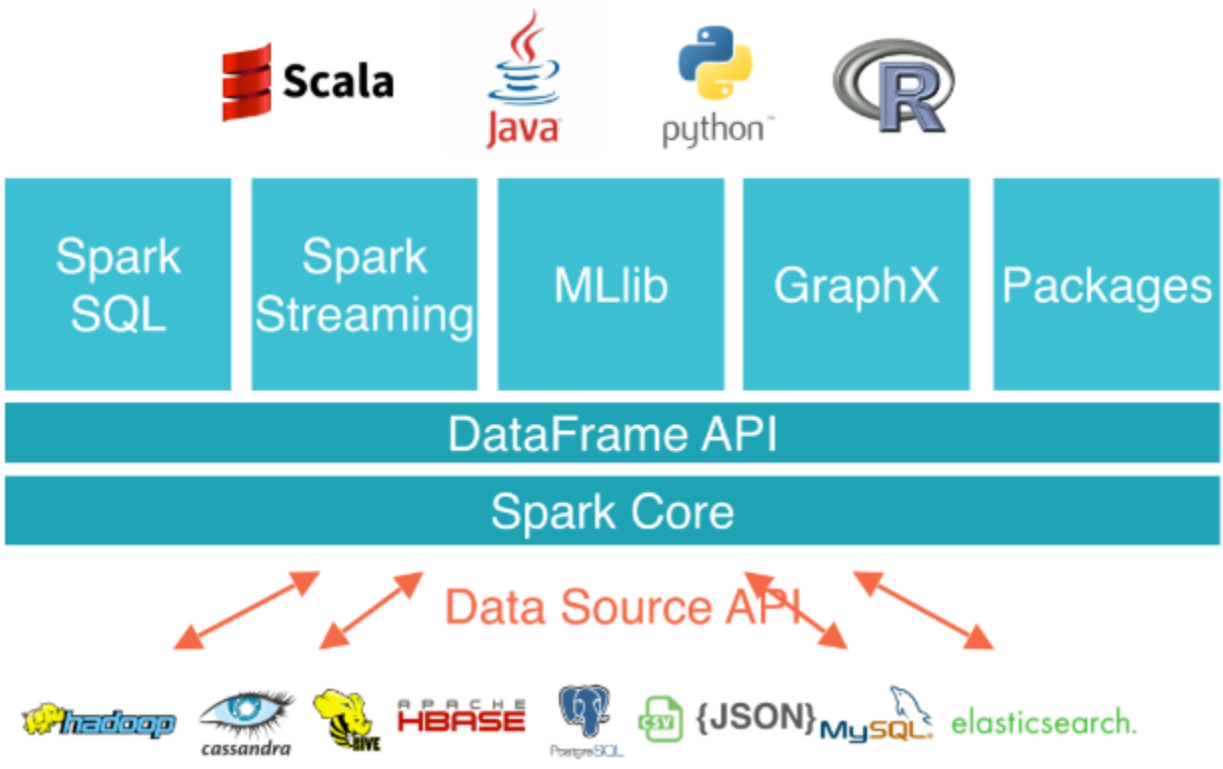
Apache Spark

Apache Spark is an open-source distributed general-purpose cluster-computing framework.

- Spark was initially started by Matei Zaharia at UC Berkeley's AMPLab in 2009, and open sourced in 2010 under a BSD license.
- In 2013, the project was donated to the Apache Software Foundation and switched its license to Apache 2.0. In February 2014, Spark became a Top-Level Apache Project.
- In November 2014, Spark founder M. Zaharia's company Databricks set a new world record in large scale sorting using Spark.
- Spark had in excess of 1000 contributors in 2015, making it one of the most active projects in the Apache Software Foundation and one of the most active open source big data projects.
- Given the popularity of the platform by 2014, paid programs like General Assembly and free fellowships like The Data Incubator have started offering customized training courses

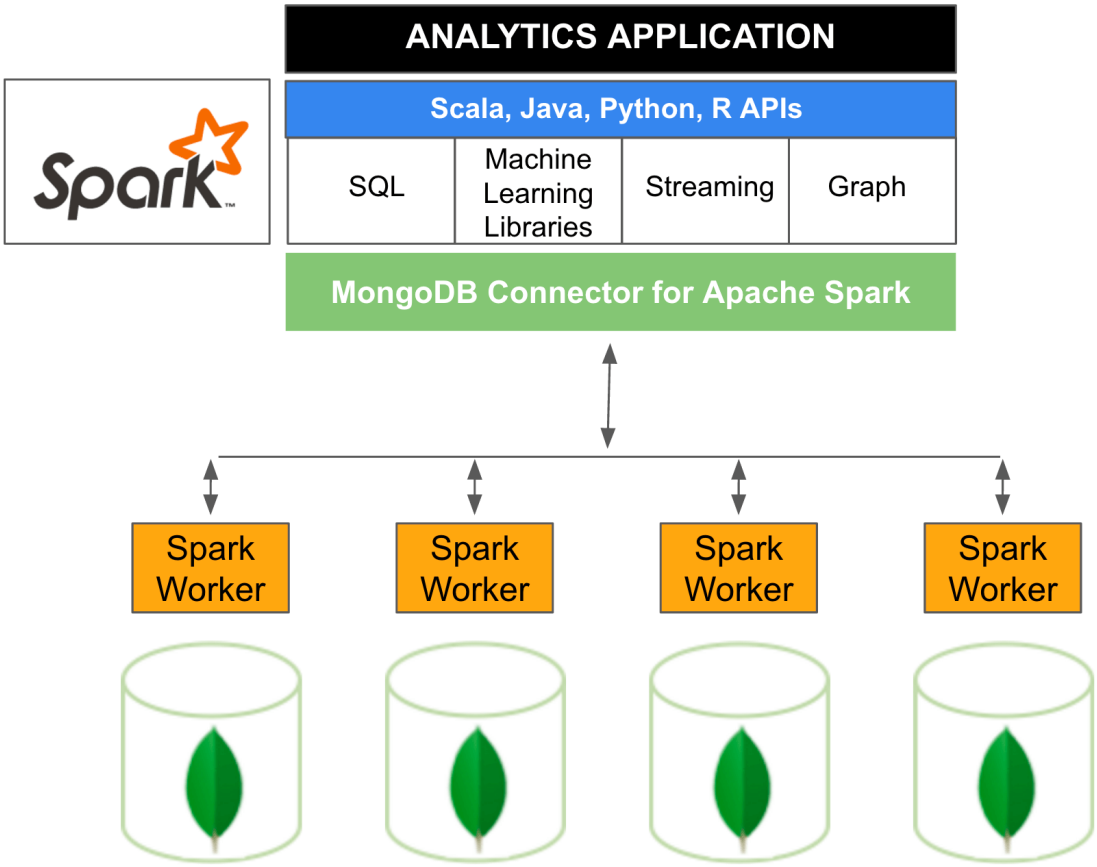
Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Below architecture shows how Apache Spark is composed/interact with other components.



- Apache Spark has as its architectural foundation the RDD(Resilient Distributed Dataset), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.
- The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API.
- Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface (for Java, Python, Scala, and R) centered on the RDD abstraction (the Java API is available for other JVM languages, but is also usable for some other non-JVM languages that can connect to the JVM, such as Julia).

- Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture. However, this convenience comes with the penalty of latency equal to the mini-batch duration. Other streaming data engines that process event by event rather than in mini-batches include Storm and the streaming component of Flink. Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.

Below diagram is a reference Apache Spark architecture based on MongoDB. The MongoDB Connector for Apache Spark exposes all of Spark's libraries, including Scala, Java, Python and R. MongoDB data is materialized as DataFrames and Datasets for analysis with machine learning, graph, streaming, and SQL APIs.



When you see above diagram, it looks like the similar architecture with MapReduce - below table shows its difference:

Item	MapReduce	Apache Spark
Data Processing	batch processing	batch processing + real-time data processing
Processing Speed	slower than Apache Spark, because of I/O disk latency	100x faster in memory and 10x faster while running on disk
Category	Data Processing Engine	Data Processing Engine
Costs	less costlier comparing Apache Spark	more Costlier because of large amount of RAM
Scalability	both are scalable limited to 1000 nodes in single cluster	both are scalable limited to 1000 nodes in single cluster
Machine Learning	more compatible with Apache Mahout while integrating with Machine Learning	have inbuilt API's to Machine Learning
Compatibility	Majorly compatible with all the data sources and file formats	Apache Spark can integrate with all data sources and file formats supported by Hadoop cluster
Security	more secured compared to Apache Spark	security feature in Apache Spark is more evolving and getting matured

Scheduler	dependent on external scheduler	have own scheduler
Fault Tolerance	Use replication for fault tolerance	using RDD and other data storage models for fault tolerance
Ease of Use	bit complex comparing Apache Spark because of Java APIs	Easier to use because of Rich APIs
Duplicate Elimination	not supported	Apache Spark process every records exactly once hence eliminates duplication
Language Support	primary language is Java but languages like C, C++, ruby, Python, Perl, Groovy is also supported	supports Java, Scalar, Python and R
Latency	very high latency	much faster comparing MapReduce framework
Complexity	hard to write and debug codes	easy to write and debug
Apache Community	open source framework for processing data	open source framework for processing data at higher speed
Coding	more lines of code	lesser lines of code
Interactive Mode	not interactive	interactive
Infrastructure	commodity hardware's	mid to high level hardware's
SQL	supports through Hive Query Language	supports through Spark SQL

Key difference between MapReduce vs Apache Spark

- MapReduce is strictly disk-based while Apache Spark uses memory and can use a disk for processing.
- MapReduce and Apache Spark both have similar compatibility in terms of data types and data sources.
- The primary difference between MapReduce and Spark is that MapReduce uses persistent storage and Spark uses Resilient Distributed Datasets.
- Hadoop MapReduce is meant for data that does not fit in the memory whereas Apache Spark has a better performance for the data that fits in the memory, particularly on dedicated clusters.
- Hadoop MapReduce can be an economical option because of Hadoop as a service and Apache Spark is more cost effective because of high availability memory
- Apache Spark and Hadoop MapReduce both are failure tolerant but comparatively Hadoop MapReduce is more failure tolerant than Spark.
- Hadoop MapReduce requires core java programming skills while Programming in Apache Spark is easier as it has an interactive mode.
- Spark is able to execute batch-processing jobs between 10 to 100 times faster than the MapReduce Although both the tools are used for processing Big Data.

When to use MapReduce

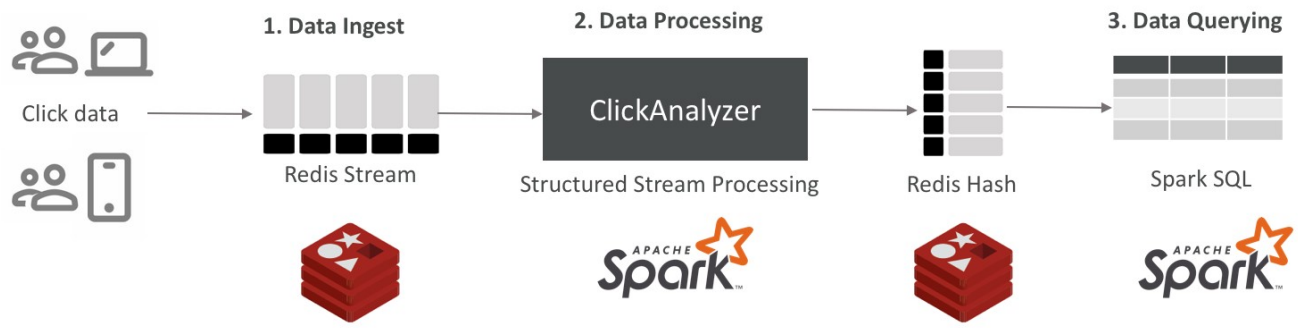
- Linear Processing of large Dataset
- No intermediate Solution required

When to use Apache Spark:

- Fast and interactive data processing
- Joining Datasets
- Graph processing
- Iterative jobs
- Real-time processing
- Machine Learning

Accelerating Apache Spark with in-memory DB (Redis)

Even though Apache Spark has better performance then MapReduce, but you may thirsty in enhancing processing performance - in-memory DB like Redis will help you to have better performance. Below architecture shows how to combine Redis with Apache Spark:


1. Data Ingest
2. Data Processing
3. Data Querying
Click data
Redis Stream
ClickAnalyzer
Structured Stream Processing
Redis Hash
Spark SQL