**SPARK**

**What is Big Data?**

Big data refers to extremely large and diverse collections of structured, unstructured, and semi-structured data that continues to grow exponentially over time. These datasets are so huge and complex in volume, velocity, and variety, that traditional data management systems cannot store, process, and analyze them.

The amount and availability of data is growing rapidly, spurred on by digital technology advancements, such as connectivity, mobility, the Internet of Things (IoT), and artificial intelligence (AI). As data continues to expand and proliferate, new big data tools are emerging to help companies collect, process, and analyze data at the speed needed to gain the most value from it.

Big data describes large and diverse datasets that are huge in volume and also rapidly grow in size over time. Big data is used in machine learning, predictive modeling, and other advanced analytics to solve business problems and make informed decisions.

Read on to learn the definition of big data, some of the advantages of big data solutions, common big data challenges, and how Google Cloud is helping organizations [build their data clouds](https://cloud.google.com/data-cloud) to get more value from their data.

[Get started for free](https://console.cloud.google.com/freetrial)

Big data examples

Data can be a company’s most valuable asset. Using big data to reveal insights can help you understand the areas that affect your business—from market conditions and customer purchasing behaviors to your business processes.

Here are some big data examples that are helping transform organizations across every industry:

* Tracking consumer behavior and shopping habits to deliver [hyper-personalized retail product recommendations](https://cloud.google.com/blog/products/ai-machine-learning/ikea-uses-google-cloud-recommendations-ai) tailored to individual customers
* Monitoring payment patterns and analyzing them against historical customer activity to [detect fraud in real time](https://cloud.google.com/blog/products/databases/how-ravelin-scales-fraud-detection-with-bigtable)
* Combining data and information from every stage of an order’s shipment journey with hyperlocal traffic insights to [help fleet operators optimize last-mile delivery](https://cloud.google.com/blog/products/maps-platform/introducing-last-mile-fleet-solution-maximize-what-your-fleet-can-do-start-finish)
* Using AI-powered technologies like [natural language processing to analyze unstructured medical data](https://cloud.google.com/blog/topics/healthcare-life-sciences/natural-language-processing-nlp-healthcare-insights-clinical-research-data-cloud) (such as research reports, clinical notes, and lab results) to gain new insights for improved treatment development and enhanced patient care
* Using image data from cameras and sensors, as well as GPS data, to [detect potholes and improve road maintenance in cities](https://cloud.google.com/blog/products/ai-machine-learning/video-intelligence-machine-learning-improves-pothole-detection)
* Analyzing public datasets of satellite imagery and geospatial datasets to visualize, monitor, measure, and predict [the social and environmental impacts of supply chain operations](https://cloud.google.com/blog/topics/consumer-packaged-goods/sustainable-sourcing-for-consumer-brands-with-google-cloud)

These are just a few ways organizations are using big data to become more data-driven so they can adapt better to the needs and expectations of their customers and the world around them.

The Vs of big data

Big data definitions may vary slightly, but it will always be described in terms of volume, velocity, and variety. These big data characteristics are often referred to as the “3 Vs of big data” and were first defined by Gartner in 2001.

**Volume**

As its name suggests, the most common characteristic associated with big data is its high volume. This describes the enormous amount of data that is available for collection and produced from a variety of sources and devices on a continuous basis.

**Velocity**

Big data velocity refers to the speed at which data is generated. Today, data is often produced in real time or near real time, and therefore, it must also be processed, accessed, and analyzed at the same rate to have any meaningful impact.

**Variety**

Data is heterogeneous, meaning it can come from many different sources and can be structured, unstructured, or semi-structured. More traditional structured data (such as data in spreadsheets or relational databases) is now supplemented by unstructured text, images, audio, video files, or semi-structured formats like sensor data that can’t be organized in a fixed data schema.

In addition to these three original Vs, three others that are often mentioned in relation to harnessing the power of big data: **veracity**,**variability**, and**value**.

* **Veracity**: Big data can be messy, noisy, and error-prone, which makes it difficult to control the quality and accuracy of the data. Large datasets can be unwieldy and confusing, while smaller datasets could present an incomplete picture. The higher the veracity of the data, the more trustworthy it is.
* **Variability:** The meaning of collected data is constantly changing, which can lead to inconsistency over time. These shifts include not only changes in context and interpretation but also data collection methods based on the information that companies want to capture and analyze.
* **Value:**It’s essential to determine the business value of the data you collect. Big data must contain the right data and then be effectively analyzed in order to yield insights that can help drive decision-making.

How does big data work?

The central concept of big data is that the more visibility you have into anything, the more effectively you can gain insights to make better decisions, uncover growth opportunities, and improve your business model.

Making big data work requires three main actions:

* **Integration:**Big data collects terabytes, and sometimes even petabytes, of raw data from many sources that must be received, processed, and transformed into the format that business users and analysts need to start analyzing it.
* **Management:**Big data needs big storage, whether in the cloud, on-premises, or both. Data must also be stored in whatever form required. It also needs to be processed and made available in real time. Increasingly, companies are turning to cloud solutions to take advantage of the unlimited compute and scalability.
* **Analysis:**The final step is analyzing and acting on big data—otherwise, the investment won’t be worth it. Beyond exploring the data itself, it’s also critical to communicate and share insights across the business in a way that everyone can understand. This includes using tools to create data visualizations like charts, graphs, and dashboards.

Big data benefits

Improved decision-making

Big data is the key element to becoming a data-driven organization. When you can manage and analyze your big data, you can discover patterns and unlock insights that improve and drive better operational and strategic decisions.

Increased agility and innovation

Big data allows you to collect and process real-time data points and analyze them to adapt quickly and gain a competitive advantage. These insights can guide and accelerate the planning, production, and launch of new products, features, and updates.

Better customer experiences

Combining and analyzing structured data sources together with unstructured ones provides you with more useful insights for consumer understanding, personalization, and ways to optimize experience to better meet consumer needs and expectations.

Continuous intelligence

Big data allows you to integrate automated, real-time data streaming with advanced data analytics to continuously collect data, find new insights, and discover new opportunities for growth and value.

More efficient operations

Using big data analytics tools and capabilities allows you to process data faster and generate insights that can help you determine areas where you can reduce costs, save time, and increase your overall efficiency.

Improved risk management

Analyzing vast amounts of data helps companies evaluate risk better—making it easier to identify and monitor all potential threats and report insights that lead to more robust control and mitigation strategies.

Challenges of implementing big data analytics

While big data has many advantages, it does present some challenges that organizations must be ready to tackle when collecting, managing, and taking action on such an enormous amount of data.

The most commonly reported big data challenges include:

* **Lack of data talent and skills.**Data scientists, data analysts, and data engineers are in short supply—and are some of the most highly sought after (and highly paid) professionals in the IT industry. Lack of big data skills and experience with advanced data tools is one of the primary barriers to realizing value from big data environments.
* **Speed of data growth.**Big data, by nature, is always rapidly changing and increasing. Without a solid infrastructure in place that can handle your processing, storage, network, and security needs, it can become extremely difficult to manage.
* **Problems with data quality.**Data quality directly impacts the quality of decision-making, data analytics, and planning strategies. Raw data is messy and can be difficult to curate. Having big data doesn’t guarantee results unless the data is accurate, relevant, and properly organized for analysis. This can slow down reporting, but if not addressed, you can end up with misleading results and worthless insights.
* **Compliance violations.**Big data contains a lot of sensitive data and information, making it a tricky task to continuously ensure data processing and storage meet data privacy and regulatory requirements, such as data localization and data residency laws.
* **Integration complexity.**Most companies work with data siloed across various systems and applications across the organization. Integrating disparate data sources and making data accessible for business users is complex, but vital, if you hope to realize any value from your big data.
* **Security concerns.**Big data contains valuable business and customer information, making big data stores high-value targets for attackers. Since these datasets are varied and complex, it can be harder to implement comprehensive strategies and policies to protect them.

How are data-driven businesses performing?

Some organizations remain wary of going all in on big data because of the time, effort, and commitment it requires to leverage it successfully. In particular, businesses struggle to rework established processes and facilitate the cultural change needed to put data at the heart of every decision.

But becoming a data-driven business is worth the work. Recent research shows:

* [58% of companies](https://www.ciodive.com/news/data-driven-companies-revenue-coronavirus-covid19/578159/) that make data-based decisions are more likely to beat revenue targets than those that don't
* Organizations with advanced insights-driven business capabilities are [2.8x more likely](https://www.forrester.com/blogs/move-from-insight-to-impact-data-strategy-insights-2020/) to report double-digit year-over-year growth
* Data-driven organizations generate, on average, [more than 30%](https://www.accenture.com/nl-en/blogs/insights/data-driven-enterprise) growth per year

The enterprises that take steps now and make significant progress toward implementing big data stand to come as winners in the future.

Big data strategies and solutions

Developing a solid data strategy starts with understanding what you want to achieve, identifying specific use cases, and the data you currently have available to use. You will also need to evaluate what additional data might be needed to meet your business goals and the new systems or tools you will need to support those.

Unlike traditional data management solutions, big data technologies and tools are made to help you deal with large and complex datasets to extract value from them. Tools for big data can help with the volume of the data collected, the speed at which that data becomes available to an organization for analysis, and the complexity or varieties of that data.

For example, [data lakes](https://cloud.google.com/learn/what-is-a-data-lake) ingest, process, and store structured, unstructured, and semi-structured data at any scale in its native format. Data lakes act as a foundation to run different types of smart analytics, including visualizations, real-time analytics, and [machine learning](https://cloud.google.com/learn/what-is-machine-learning).

It’s important to keep in mind that when it comes to big data—there is no one-size-fits-all strategy. What works for one company may not be the right approach for your organization’s specific needs.

Here are four key concepts that our Google Cloud customers have taught us about shaping a winning approach to big data:

**Open**

Today, organizations need the freedom to build what they want using the tools and solutions they want. As data sources continue to grow and new technology innovations become available, the reality of big data is one that contains multiple interfaces, open source technology stacks, and clouds. Big data environments will need to be architected to be both open and adaptable to allow for companies to build the solutions and get the data it needs to win.

**Intelligent**

Big data requires data capabilities that will allow them to leverage smart analytics and AI and ML technologies to save time and effort delivering insights that improve business decisions and managing your overall big data infrastructure. For example, you should consider automating processes or enabling self-service analytics so that people can work with data on their own, with minimal support from other teams.

**Flexible**

Big data analytics need to support innovation, not hinder it. This requires building a data foundation that will offer on-demand access to compute and storage resources and unify data so that it can be easily discovered and accessed. It’s also important to be able to choose technologies and solutions that can be easily combined and used in tandem to create the perfect data toolsets that fit the workload and use case.

**Trusted**

For big data to be useful, it must be trusted. That means it’s imperative to build trust into your data—trust that it’s accurate, relevant, and protected. No matter where data comes from, it should be secure by default and your strategy will also need to consider what security capabilities will be necessary to ensure compliance, redundancy, and reliability

LINK - [ <https://cloud.google.com/learn/what-is-big-data> ]

----------------------------------------------------------------------------------------------------

## What is Apache Spark?

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, [machine learning](https://aws.amazon.com/what-is/machine-learning/), and graph processing. You’ll find it used by organizations from any industry, including at FINRA, Yelp, Zillow, DataXu, Urban Institute, and CrowdStrike.

## What is the history of Apache Spark?

Apache Spark started in 2009 as a research project at UC Berkley’s AMPLab, a collaboration involving students, researchers, and faculty, focused on data-intensive application domains. The goal of Spark was to create a new framework, optimized for fast iterative processing like machine learning, and interactive data analysis, while retaining the scalability, and fault tolerance of Hadoop MapReduce. The first paper entitled, “Spark: Cluster Computing with Working Sets” was published in June 2010, and Spark was open sourced under a BSD license. In June, 2013, Spark entered incubation status at the Apache Software Foundation (ASF), and established as an Apache Top-Level Project in February, 2014. Spark can run standalone, on Apache Mesos, or most frequently on Apache Hadoop.

## How does Apache Spark work?

Hadoop MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm. Developers can write massively parallelized operators, without having to worry about work distribution, and fault tolerance. However, a challenge to MapReduce is the sequential multi-step process it takes to run a job. With each step, MapReduce reads data from the cluster, performs operations, and writes the results back to HDFS. Because each step requires a disk read, and write, MapReduce jobs are slower due to the latency of disk I/O.

Spark was created to address the limitations to MapReduce, by doing processing in-memory, reducing the number of steps in a job, and by reusing data across multiple parallel operations. With Spark, only one-step is needed where data is read into memory, operations performed, and the results written back—resulting in a much faster execution. Spark also reuses data by using an in-memory cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset. Data re-use is accomplished through the creation of DataFrames, an abstraction over Resilient Distributed Dataset (RDD), which is a collection of objects that is cached in memory, and reused in multiple Spark operations. This dramatically lowers the latency making Spark multiple times faster than MapReduce, especially when doing machine learning, and interactive analytics.

## Key differences: Apache Spark vs. Apache Hadoop

Outside of the differences in the design of Spark and Hadoop MapReduce, many organizations have found these big data frameworks to be complimentary, using them together to solve a broader business challenge.

Hadoop is an open source framework that has the Hadoop Distributed File System (HDFS) as storage, YARN as a way of managing computing resources used by different applications, and an implementation of the MapReduce programming model as an execution engine. In a typical Hadoop implementation, different execution engines are also deployed such as Spark, Tez, and Presto.

Spark is an open source framework focused on interactive query, machine learning, and real-time workloads. It does not have its own storage system, but runs analytics on other storage systems like HDFS, or other popular stores like [Amazon Redshift](https://aws.amazon.com/redshift/), [Amazon S3](https://aws.amazon.com/s3/), Couchbase, Cassandra, and others. Spark on Hadoop leverages YARN to share a common cluster and dataset as other Hadoop engines, ensuring consistent levels of service, and response.

## What are the benefits of Apache Spark?

There are many benefits of Apache Spark to make it one of the most active projects in the Hadoop ecosystem. These include:

#### **Fast**

Through in-memory caching, and optimized query execution, Spark can run fast analytic queries against data of any size.

#### **Developer friendly**

Apache Spark natively supports Java, Scala, R, and Python, giving you a variety of languages for building your applications. These APIs make it easy for your developers, because they hide the complexity of distributed processing behind simple, high-level operators that dramatically lowers the amount of code required.

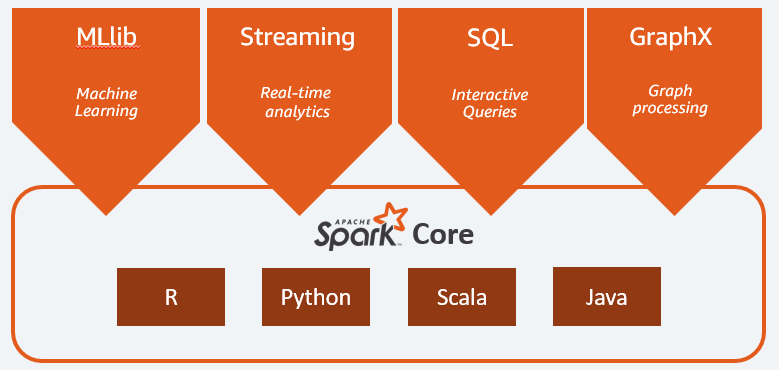
#### **Multiple workloads**

Apache Spark comes with the ability to run multiple workloads, including interactive queries, real-time analytics, machine learning, and graph processing. One application can combine multiple workloads seamlessly.

## What are Apache Spark Workloads?

The Spark framework includes:

* Spark Core as the foundation for the platform
* Spark SQL for interactive queries
* Spark Streaming for real-time analytics
* Spark MLlib for machine learning
* Spark GraphX for graph processing



### **Spark Core**

Spark Core is the foundation of the platform. It is responsible for memory management, fault recovery, scheduling, distributing & monitoring jobs, and interacting with storage systems. Spark Core is exposed through an application programming interface (APIs) built for Java, Scala, Python and R. These APIs hide the complexity of distributed processing behind simple, high-level operators.

### **MLlib**

#### Machine Learning

Spark includes MLlib, a library of algorithms to do machine learning on data at scale. Machine Learning models can be trained by data scientists with R or Python on any Hadoop data source, saved using MLlib, and imported into a Java or Scala-based pipeline. Spark was designed for fast, interactive computation that runs in memory, enabling machine learning to run quickly. The algorithms include the ability to do classification, regression, clustering, collaborative filtering, and pattern mining.

### **Spark Streaming**

#### Real-time

Spark Streaming is a real-time solution that leverages Spark Core’s fast scheduling capability to do streaming analytics. It ingests data in mini-batches, and enables analytics on that data with the same application code written for batch analytics. This improves developer productivity, because they can use the same code for batch processing, and for real-time streaming applications. Spark Streaming supports data from Twitter, Kafka, Flume, HDFS, and ZeroMQ, and many others found from the Spark Packages ecosystem.

### **Spark SQL**

#### Interactive Queries

Spark SQL is a distributed query engine that provides low-latency, interactive queries up to 100x faster than MapReduce. It includes a cost-based optimizer, columnar storage, and code generation for fast queries, while scaling to thousands of nodes. Business analysts can use standard SQL or the Hive Query Language for querying data. Developers can use APIs, available in Scala, Java, Python, and R. It supports various data sources out-of-the-box including JDBC, ODBC, JSON, HDFS, Hive, ORC, and Parquet. Other popular stores—Amazon Redshift, Amazon S3, Couchbase, Cassandra, MongoDB, Salesforce.com, Elasticsearch, and many others can be found from the [Spark Packages](https://spark-packages.org/?q=tags%3A%22Data%20Sources%22) ecosystem.

### **GraphX**

#### Graph Processing

Spark GraphX is a distributed graph processing framework built on top of Spark. GraphX provides ETL, exploratory analysis, and iterative graph computation to enable users to interactively build, and transform a graph data structure at scale. It comes with a highly flexible API, and a selection of distributed Graph algorithms.

## What are the use cases of Apache Spark?

Spark is a general-purpose distributed processing system used for big data workloads. It has been deployed in every type of big data use case to detect patterns, and provide real-time insight. Example use cases include:

### **Financial Services**

Spark is used in banking to predict customer churn, and recommend new financial products. In investment banking, Spark is used to analyze stock prices to predict future trends.

### **Healthcare**

Spark is used to build comprehensive patient care, by making data available to front-line health workers for every patient interaction. Spark can also be used to predict/recommend patient treatment.

### **Manufacturing**

Spark is used to eliminate downtime of internet-connected equipment, by recommending when to do preventive maintenance.

### **Retail**

Spark is used to attract, and keep customers through personalized services and offers.

## How deploying Apache Spark in the cloud works?

Spark is an ideal workload in the cloud, because the cloud provides performance, scalability, reliability, availability, and massive economies of scale. ESG research found 43% of respondents considering cloud as their primary deployment for Spark. The top reasons customers perceived the cloud as an advantage for Spark are faster time to deployment, better availability, more frequent feature/functionality updates, more elasticity, more geographic coverage, and costs linked to actual utilization.

## What are the AWS offerings for Apache Spark?

[Amazon EMR](https://aws.amazon.com/emr/) is the best place to deploy Apache Spark in the cloud, because it combines the integration and testing rigor of commercial Hadoop & Spark distributions with the scale, simplicity, and cost effectiveness of the cloud. It allows you to launch Spark clusters in minutes without needing to do node provisioning, cluster setup, Spark configuration, or cluster tuning. EMR enables you to provision one, hundreds, or thousands of compute instances in minutes. You can use [Auto Scaling](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-automatic-scaling.html) to have EMR automatically scale up your Spark clusters to process data of any size, and back down when your job is complete to avoid paying for unused capacity. You can lower your bill by committing to a set term, and saving up to 75% using [Amazon EC2 Reserved Instances](https://aws.amazon.com/ec2/pricing/reserved-instances/), or running your clusters on spare AWS compute capacity and saving up to 90% using [EC2 Spot](https://aws.amazon.com/ec2/spot/).

LINK - [ https://aws.amazon.com/what-is/apache-spark/ ]

**An Introduction to Apache Spark: Big Data Processing Made Easy**

Reference:[https://pixabay.com/illustrations](https://pixabay.com/illustrations/big-data-abstract-7644530/" \t "_blank)

In today’s data-driven world, organizations grapple with massive amounts of information pouring in from various sources. To extract meaningful insights from this data, a robust and scalable processing framework is essential.

Apache Spark has revolutionized the world of big data processing, providing a fast, scalable, and versatile solution for handling large-scale data analytics tasks. In this article, we will explore the following topics:

* Introduction of Spark
* Before Spark…
* The key concepts and features of Spark
* Core components
* Common use cases of spark

Let’s Start…

**What is Apache Spark?**

Apache Spark is an open-source distributed computing system designed for big data processing and analytics. It provides an interface for programming clusters with implicit data parallelism and fault tolerance. Spark is known for its speed and efficiency, thanks to its in-memory computing capabilities and optimized data processing techniques.

**Before the evolution of the spark...**

Industries were using Hadoop extensively to analyze their data sets. Hadoop is an open-source framework designed to process and store large volumes of data across distributed computing clusters. The reason to use the Hadoop framework is, It is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant, and cost-effective. Here, the main concern was to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.

Remember, **Spark is not a modified version of Hadoop** and also it is not dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark.

Spark can use Hadoop in two ways — one is **storage** and the second is **processing**. Since Spark has its own cluster management computation, it uses Hadoop for storage purposes only.

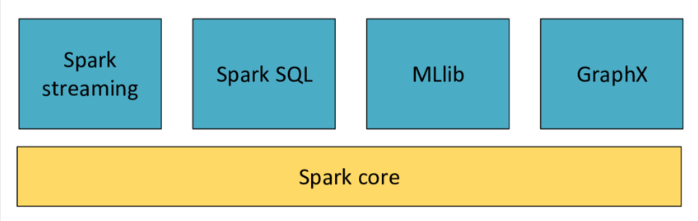
**Features of Apache Spark:**

* **In-Memory Processing(speed)**: Spark leverages in-memory computing to store and process data in memory, resulting in significantly faster data processing compared to disk-based systems like Hadoop MapReduce. By minimizing disk I/O, Spark enables iterative algorithms, interactive data exploration, and real-time analytics.
* **Distributed Computing**: Spark distributes data and computation across multiple nodes in a cluster, enabling parallel processing and efficient resource utilization. It automatically manages task scheduling, data partitioning, and fault tolerance, ensuring scalability and high availability.
* **Broad Language Support**: Spark supports multiple programming languages, including Scala, Java, Python, and R. This allows developers and data scientists to work with Spark using their preferred language and leverage existing code and libraries.
* **Integration with Big Data Ecosystem**: Spark integrates well with various data storage systems and technologies, including Hadoop Distributed File System (HDFS), HBase, Cassandra, and Amazon S3. It can read and write data from different sources and seamlessly interoperate with existing big data tools and frameworks.

**key concepts in Spark:**

* **Resilient Distributed Datasets (RDDs)**: RDDs are the fundamental data structure in Spark. They represent distributed collections of objects that can be processed in parallel. RDDs are fault-tolerant and allow for iterative computations, making them ideal for big data analytics.
* **DataFrames and Datasets**: Spark introduced higher-level abstractions called DataFrames and Datasets, built on top of RDDs. DataFrames provide a structured and optimized way to work with structured data, while Datasets offer a type-safe and object-oriented API. Both DataFrames and Datasets support various data manipulation operations.

**Components of Spark:**



Reference:[https://www.researchgate.net](https://www.researchgate.net/figure/Spark-Components-Including-MLlib-GraphX-Spark-SQL-Spark-Streaming-on-top-of-Spark-Core_fig1_366212873" \t "_blank)

1. **Spark Streaming**: Spark Streaming enables processing and analyzing real-time streaming data. It ingests data in mini-batches and performs parallel processing on the live data stream. Spark Streaming integrates with other Spark components, allowing seamless integration of batch and real-time processing.
2. **Spark SQL**: Spark SQL is a module in Spark that provides a programming interface for querying structured and semi-structured data using SQL, HiveQL, or DataFrame APIs. It enables seamless integration with other Spark components and supports various data sources, including Hive, Parquet, JSON, and JDBC.
3. **Machine Learning Library (MLlib)**: MLlib is Spark’s scalable machine learning library. It offers a wide range of algorithms and utilities for classification, regression, clustering, recommendation systems, and more.
4. **Graph Processing with GraphX**: Spark’s GraphX is a graph processing library that provides an API for graph computation and analysis. It enables efficient graph parallel algorithms and integrates with the rest of the Spark ecosystem, making it easy to combine graph processing with other data processing tasks.

**Common Use Cases of Spark:**

Apache Spark finds applications in various industries and domains. Here are some common use cases where Spark excels:

**1. Fraud Detection**:  
Spark’s real-time processing capabilities and machine learning library (MLlib) make it suitable for fraud detection. By analyzing large volumes of transactional data in real time and applying machine learning algorithms, Spark can identify patterns and anomalies that indicate fraudulent activities, helping businesses prevent financial losses.

**2. Recommendation Systems:**  
Spark’s machine learning capabilities and distributed computing enable efficient recommendation systems. By analyzing user behavior, preferences, and historical data, Spark can generate personalized recommendations for products, movies, music, and more, enhancing user engagement and driving sales.

**3. Predictive Analytics:**  
Spark’s machine learning algorithms and in-memory processing are valuable for predictive analytics. Businesses can leverage Spark to build predictive models that forecast trends, customer behavior, demand patterns, and market trends. This enables proactive decision-making and enhances business strategies.

Other use cases are Real-time Analytics, Large-scale Data Processing:  
Log Analysis, Healthcare Analytics, Social Media Analysis, etc.

**Final Thought:**

In conclusion, Apache Spark has revolutionized the world of big data processing and analytics. Its powerful features, scalability, and versatility make it a go-to framework for organizations dealing with massive volumes of data.

---------------------------------------------------------------------------------

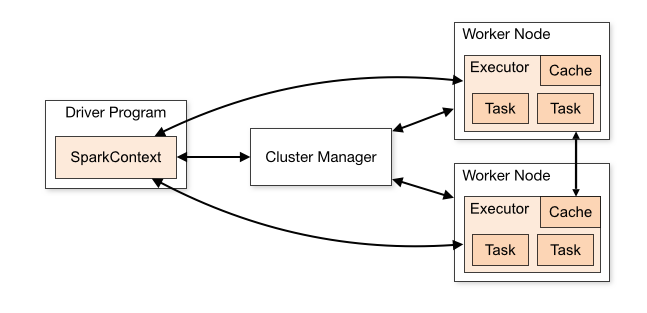
**Spark Architecture: A Deep Dive**

Apache Spark is an open-source distributed computing system designed for big data processing and analytics. Spark is known for its speed and efficiency. If you want more introduction about spark I have already covered that part in [this](https://medium.com/@amitjoshi7/an-introduction-to-apache-spark-big-data-processing-made-easy-d1288153607d) blog. In this article, we will explore the following topics:

* spark Architecture And Applications
* working of spark Architecture
* Abstractions of Apache Spark
* Cluster Manager Types
* Execution Modes

Let's start...

The Apache Spark framework uses a **master-slave architecture** that consists of a driver, which runs as a master node, and many executors that run across as worker nodes in the cluster. Apache Spark can be used for batch processing and real-time processing as well.



Reference:[https://www.analyticsvidhya.com](https://www.analyticsvidhya.com/" \t "_blank)

Before understanding the Spark architecture let's understand the Applications of Spark Architecture which exists in the above diagram:

**The Spark driver**

The driver is the program or process responsible for coordinating the execution of the Spark application. It runs the main function and creates the SparkContext, which connects to the cluster manager.

**The Spark executors**

Executors are worker processes responsible for executing tasks in Spark applications. They are launched on worker nodes and communicate with the driver program and cluster manager. Executors run tasks concurrently and store data in memory or disk for caching and intermediate storage.

**The cluster manager**

The cluster manager is responsible for allocating resources and managing the cluster on which the Spark application runs. Spark supports various cluster managers like Apache Mesos, Hadoop YARN, and standalone cluster manager.

**sparkContext**

SparkContext is the entry point for any Spark functionality. It represents the connection to a Spark cluster and can be used to create RDDs (Resilient Distributed Datasets), accumulators, and broadcast variables. SparkContext also coordinates the execution of tasks.

**Task**

A task is the smallest unit of work in Spark, representing a unit of computation that can be performed on a single partition of data. The driver program divides the Spark job into tasks and assigns them to the executor nodes for execution.

**Working Of Spark Architecture**

When the Driver Program in the Apache Spark architecture executes, it calls the real program of an application and creates a SparkContext. SparkContext contains all of the basic functions. The Spark Driver includes several other components, including a DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager, all of which are responsible for translating user-written code into jobs that are actually executed on the cluster. Spark Driver and SparkContext collectively watch over the job execution within the cluster

The Cluster Manager manages the execution of various jobs in the cluster. Spark Driver works in conjunction with the Cluster Manager to control the execution of various other jobs. The cluster Manager does the task of allocating resources for the job. Once the job has been broken down into smaller jobs, which are then distributed to worker nodes, SparkDriver will control the execution.  
Many worker nodes can be used to process an RDD(Resilient Distributed Dataset) created in SparkContext, and the results can also be cached.

The SparkContext receives task information from the Cluster Manager and enqueues it on worker nodes. The executor is in charge of carrying out these duties. The lifespan of executors is the same as that of the Spark Application. We can increase the number of workers if we want to improve the performance of the system. In this way, we can divide jobs into more coherent parts.

**Two Main Abstractions of Apache Spark**

Apache Spark has a well-defined layer architecture that is designed on two main abstractions:

* **Resilient Distributed Dataset (RDD)**: RDD is an immutable (read-only), fundamental collection of elements or items that can be operated on many devices at the same time (spark parallel processing). Each dataset in an RDD can be divided into logical portions, which are then executed on different nodes of a cluster.
* **Directed Acyclic Graph (DAG)**: DAG is the scheduling layer of the Apache Spark architecture that implements stage-oriented scheduling. Compared to MapReduce which creates a graph in two stages, Map and Reduce, Apache Spark can create DAGs that contain many stages.

**Cluster Manager Types**

The system currently supports several cluster managers:

* **Standalone**— a simple cluster manager included with Spark that makes it easy to set up a cluster.
* **Apache Mesos**— a general cluster manager that can also run Hadoop MapReduce and service applications.
* **Hadoop YARN**— the resource manager in Hadoop 2.
* **Kubernetes**— an open-source system for automating deployment, scaling, and management of containerized applications.

**Execution Modes**

**Cluster mode**

Cluster mode is probably the most common way of running Spark Applications. In cluster mode, a user submits a pre-compiled JAR, Python script, or R script to a cluster manager. The cluster manager then launches the driver process on a worker node inside the cluster, in addition to the executor processes. This means that the cluster manager is responsible for maintaining all Spark Application–related processes.

**Client mode**

Client mode is nearly the same as cluster mode except that the Spark driver remains on the client machine that submitted the application. This means that the client machine is responsible for maintaining the Spark driver process, and the cluster manager maintains the executor processes.

**Local mode**

Local mode is a significant departure from the previous two modes: it runs the entire Spark Application on a single machine. It achieves parallelism through threads on that single machine. This is a common way to learn Spark, test your applications, or experiment iteratively with local development.

**Conclusion**

We learned about the Apache Spark Architecture in order to understand how to build big data applications efficiently. They’re accessible and consist of components, which is very beneficial for cluster computing and big data technology. Spark calculates the desired outcomes in an easy way and is popular for batch processing.

**Components of Apache Spark (EcoSystem)**

**Introduction**

Now since we have some understanding of Spark, let us dive deeper and understand its components. Apache Spark consists of Spark Core Engine, Spark SQL, Spark Streaming, MLlib, GraphX and Spark R. You can use Spark Core Engine along with any of the other five components mentioned above. It is not necessary to use all the Spark components together. Depending on the use case and application, any one or more of these can be used along with Spark Core.

Let us look at each of these components in detail.



1. Spark Core
2. Spark SQL
3. Spark Streaming
4. MLlib(Machine learning library)
5. GraphX
6. Spark R

Now since we have some understanding of Spark let us dive deeper into Spark and understand the components Apache Spark consists of. Apache Spark consists of Spark Core Engine, Spark SQL, Spark Streaming, MLlib, GraphX, and Spark R. You can use Spark Core Engine along with any of the other five components mentioned above. It is not necessary to use all the Spark components together. Depending on the use case and application any one or more of these can be used along with Spark Core.

**Let us look at each of these components in detail.**

**Spark Core:** Spark Core is the heart of the Apache Spark framework. Spark Core provides the execution engine for the Spark platform which is required and used by other components which are built on top of Spark Core as per the requirement. Spark Core provides the in-built memory computing and referencing datasets stored in external storage systems. It is Spark’s core responsibility to perform all the basic I/O functions, scheduling, monitoring, etc. Also, fault recovery and effective memory management are Spark Core’s other important functions.

Spark Core uses a very special data structure called the RDD. Data sharing in distributed processing systems like MapReduce need the data in intermediate steps to be stored and then retrieved from permanent storage like HDFS or S3 which makes it very slow due to the serialization and deserialization of I/O steps. RDDs overcome this as these data structures are in-memory and fault-tolerant and can be shared across different tasks within the same Spark process. The RDDs can be any immutable and partitioned collections and can contain any type of objects; Python, Scala, Java or some user-defined class objects. RDDs can be created either by Transformations of an existing RDD or loading from external sources like HDFS or HBase etc. We will look into RDD and its transformations in-depth in later sections in the tutorial.

**Spark SQL:** Spark SQL is built on top of Shark which was the first interactive SQL on the Hadoop system. Shark was built on top of Hive codebase and achieved performance improvement by swapping out the physical execution engine part of the Hive. But due to the limitations of Hive, Shark was not able to achieve the performance it was supposed to. So the Shark project was stopped and Spark SQL was built with the knowledge of Shark on top of Spark Core Engine to leverage the power of Spark. You can read more about [Shark](https://databricks.com/blog/2014/07/01/shark-spark-sql-hive-on-spark-and-the-future-of-sql-on-spark.html) in the following blog by Reynold Xin, one of the Spark SQL code maintainers.

Spark SQL is named like this because it works with the data in a similar fashion to SQL. In fact it there is a mention that Spark SQL’s aim is to meet SQL 92 standards. But the gist is that it allows developers to write declarative code letting the engine use as much of the data and stored structure (RDDs) as it can to optimize the resultant distributed query behind the scenes. The goal is to allow the user to not have to worry about the distributed nature as much and focus on the business use case. Users can perform extract, transform and load functions on data from a variety of sources in different formats like JSON, Parquet or Hive and then execute ad-hoc queries using Spark SQL.

DataFrame constitutes the main abstraction for Spark SQL. Distributed collection of data ordered into named columns is known as a DataFrame in Spark. In the earlier versions of Spark SQL, DataFrames were referred to as SchemaRDDs. DataFrame API in Spark integrates with the Spark procedural code to render tight integration between procedural and relational processing. DataFrame API evaluates operations in a lazy manner to provide support for relational optimizations and optimize the overall data processing workflow. All relational functionalities in Spark can be encapsulated using the SparkSQL context or HiveContext.

Catalyst, an extensible optimizer is at the core functioning of Spark SQL, which is an optimization framework embedded in Scala to help developers improve their productivity and performance of the queries that they write. Using Catalyst, Spark developers can briefly specify complex relational optimizations and query transformations in a few lines of code by making the best use of Scala’s powerful programming constructs like pattern matching and runtime metaprogramming. Catalyst eases the process of adding optimization rules, data sources and data types for machine learning domains.

**Spark Streaming:** This Spark library is primarily maintained by Tathagat Das and helped by MatieZaharia. As the name suggests this library is for Streaming data. This is a very popular Spark library as it takes Spark’s big data processing power and cranks up the speed. Spark Streaming has the ability to Stream gigabytes per second. This capability of big and fast data has a lot of potentials. Spark Streaming is used for analyzing a continuous stream of data. A common example is processing log data from a website or server.

Spark streaming is not really streaming technically. What it really does is it breaks down the data into individual chunks that it processes together as small RDDs. So it actually does not process data as bytes at a time as it comes in, but it processes data every second or two seconds or some fixed interval of time. So strictly speaking Spark streaming is not real-time but near real-time or micro batching, but it suffices for a vast majority of applications.

Spark streaming can be configured to talk to a variety of data sources. So we can just listen to a port that has a bunch of data being thrown at it, or we can connect to data sources like Amazon Kinesis, Kafka, Flume, etc. There are connectors available to connect Spark to these sources. The good thing about Spark streaming is it is reliable. It has a concept called “checkpointing” to store state to the disk periodically and depending on what kind of data sources or receiver we are using, it can pick up data from the point of failure. It is a very robust mechanism to handle all kinds of failures like disk failure or node failure etc. Spark Streaming has exactly-once message guarantees and helps recover lost work without having to write any extra code or adding additional configurations.

Just like how Spark SQL has the concept of Dataframe/Dataset built on top of RDD, Spark streaming has something called Dstream. This is a collection of RDDs that embodies the entire stream data. The good thing about Dstream is that we can apply most of the built-in functions on RDDs also on the DStream like flatMap, map, etc. Also, the Dstream can be broken into individual RDDs and can be processed one chunk at a time. Spark developers can reuse the same code for stream and batch processing and can also integrate the streaming data with historical data.

**MLlib:** Today many companies focus on building customer-centric data products and services which need machine learning to build predictive insights, recommendations, and personalized results. Data scientists can solve these problems using popular languages like Python and R, but they spend a lot of time in building and supporting infrastructure for these languages. Spark has built-in support for doing machine learning and data science at a massive scale using the clusters. It’s called MLLib which stands for Machine Learning Library.

MLlib is a low-level machine learning library. It can be called from Java, Scala and Python programming languages. It is simple to use, scalable and can be easily integrated with other tools and frameworks. MLlib eases the deployment and development of scalable machine learning pipelines. Machine learning in itself is a subject and it may not be possible to get into details here. But these are some of the important features and capabilities Spark MLLib offers:

* Linear regression, logistic regression
* Support Vector Machines
* Naive Bayes classifier
* K-Means clustering
* Decision trees
* Recommendations using Alternating Least Squares
* Basic statistics
* Chi-squared test, Pearsons or Spearman correlation, min, max, mean, variance
* Feature extraction
* Term Frequency/ Inverse Document Frequency useful for search

 **GraphX:** For graphs and graph-parallel processing Apache Spark provides another API called GraphX. The graph here does not mean charts, lines or bar graphs, but these are graphs in computer sciences like social networks which consist of vertices where each vertex consists of an individual user in the social network and there are many users connected to each other by edges. These edges represent the relationship between the users in the network.

GraphX is useful in giving overall information about the graph network like it can tell how many triangles appear in the graph and apply the PageRank algorithm to it. It can measure things like “connectedness”, degree distribution, average path length and other high-level measures of a graph. It can also join graphs together and transform graphs quickly. It also supports the Pregel API for traversing a graph. Spark GraphX provides Resilient Distributed Graph (RDG- an abstraction of Spark RDD’s). RDG’s API is used by data scientists to perform several graph operations through various computational primitives. Similar to RDDs basic operations like map, filter, property graphs also consist of basic operators.  Those operators take UDFs (user-defined functions) and produce new graphs. Moreover, these are produced with transformed properties and structure.

**Spark R:** R programming language is widely used by Data scientists due to its simplicity and ability to run complex algorithms. But R suffers from a problem that its data processing capacity is limited to a single node. This makes R not usable when processing a huge amount of data. The problem is solved by SparkR which is an R package in Apache Spark. SparkR provides data frame implementation that supports operations like selection, filtering, aggregation, etc. on distributed large datasets. SparkR also has support for distributed machine learning using Spark MLlib.

**Conclusion**

The above components make Apache Spark the best Big data processing engine. All these components are provided out of the box and we can use them separately or together.

LINK - [ <https://www.knowledgehut.com/tutorials/apache-spark-tutorial/apache-spark-components> ]

**PYSPARK**

What is PySpark?

Apache Spark is written in Scala programming language. PySpark has been released in order to support the collaboration of Apache Spark and Python, it actually is a Python API for Spark. In addition, PySpark, helps you interface with Resilient Distributed Datasets (RDDs) in Apache Spark and Python programming language. This has been achieved by taking advantage of the Py4j library.



Py4J is a popular library which is integrated within PySpark and allows python to dynamically interface with JVM objects. PySpark features quite a few libraries for writing efficient programs. Furthermore, there are various external libraries that are also compatible. Here are some of them:

## PySparkSQL

A PySpark library to apply SQL-like analysis on a huge amount of structured or semi-structured data. We can also use SQL queries with PySparkSQL. It can also be connected to [Apache Hive](https://www.databricks.com/glossary/apache-hive). HiveQL can be also be applied. PySparkSQL is a wrapper over the PySpark core. PySparkSQL introduced the DataFrame, a tabular representation of structured data that is similar to that of a table from a relational database management system.

## MLlib

MLlib is a wrapper over the PySpark and it is Spark’s machine learning (ML) library. This library uses the data parallelism technique to store and work with data. The machine-learning API provided by the MLlib library is quite easy to use. MLlib supports many machine-learning algorithms for classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.

## GraphFrames

The GraphFrames is a purpose graph processing library that provides a set of APIs for performing graph analysis efficiently, using the PySpark core and PySparkSQL. It is optimized for fast distributed computing. Advantages of using PySpark: • Python is very easy to learn and implement. • It provides simple and comprehensive API. • With Python, the readability of code, maintenance, and familiarity is far better. • It features various options for data visualization, which is difficult using Scala or Java.

LINK - [ https://www.databricks.com/glossary/pyspark ]