

Credit: <https://www.apache.org/>

**Apache Hadoop and Apache Spark**

**Apache Hadoop and Apache Spark**

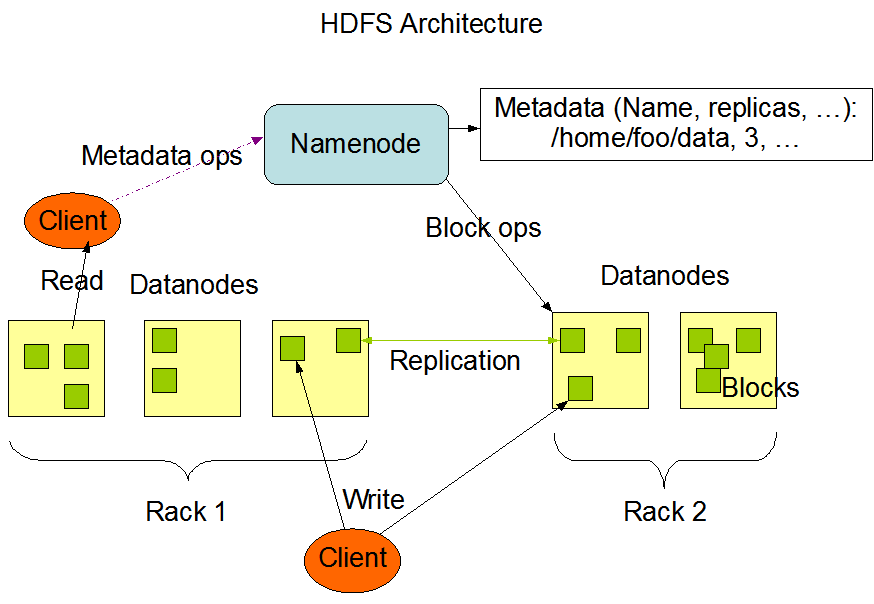


Apache Hadoop and Apache Spark

**I. Storage**

**1. Apache Hadoop Distributed File System (HDFS)**

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.



HDFS Architecture

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master ResourceManager, one worker NodeManager per cluster-node, and MRAppMaster per application (see YARN Architecture Guide).

Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration.

The Hadoop job client then submits the job (jar/executable etc.) and configuration to the ResourceManager which then assumes the responsibility of distributing the software/configuration to the workers, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

**2. Alluxio**



Alluxio

Alluxio is a data orchestration technology for analytics and AI for the cloud. It bridges the gap between data driven applications and storage systems, bringing data from the storage tier closer to the data driven applications and makes it easily accessible enabling applications to connect to numerous storage systems through a common interface. Alluxio’s memory-first tiered architecture enables data access at speeds orders of magnitude faster than existing solutions.

Alluxio is an abstraction layer between compute and distributed/cloud storage systems including HDFS providing a unified file system namespace.

With Alluxio, you also get workload acceleration – hot data is cached in memory, which greatly accelerates the performance of MapReduce jobs that are accessing this data. In addition, Alluxio delivers predictable performance, allowing the system to guarantee a certain quality of service.

Alluxio also makes it easy to offload your MapR/HDFS compute to any object store, cloud or on-prem, and run all of your existing job as-is on Alluxio + the object store of your choosing.

In the data ecosystem, Alluxio lies between data driven applications, such as Apache Spark, Presto, Tensorflow, Apache HBase, Apache Hive, or Apache Flink, and various persistent storage systems.

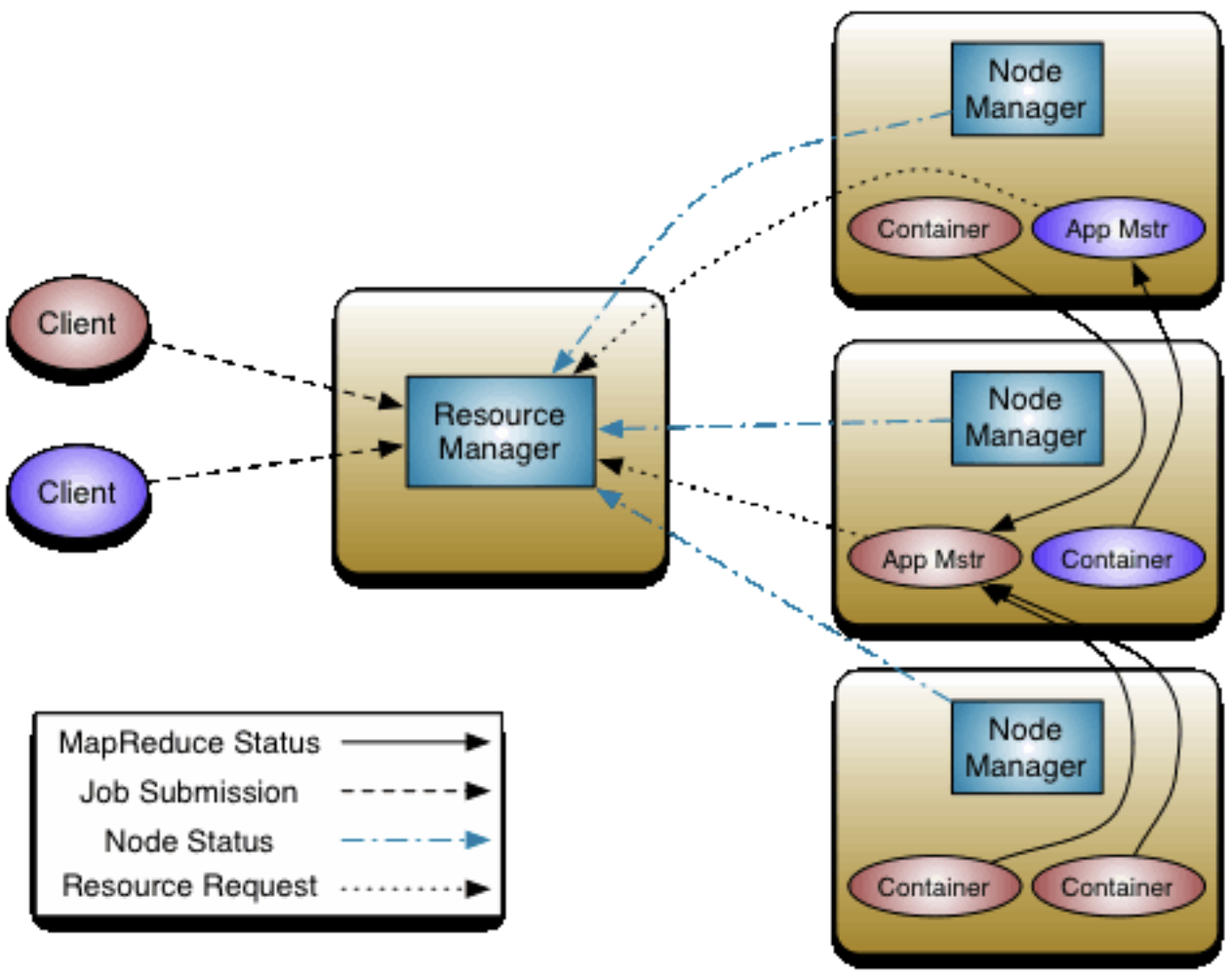
**II. Resource Management**

**3. Apache Hadoop Yet Another Resource Negotiator (YARN)**

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job or a DAG of jobs.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.



Apache Hadoop Yet Another Resource Negotiator (YARN)

The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource Container which incorporates elements such as memory, cpu, disk, network etc.

The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

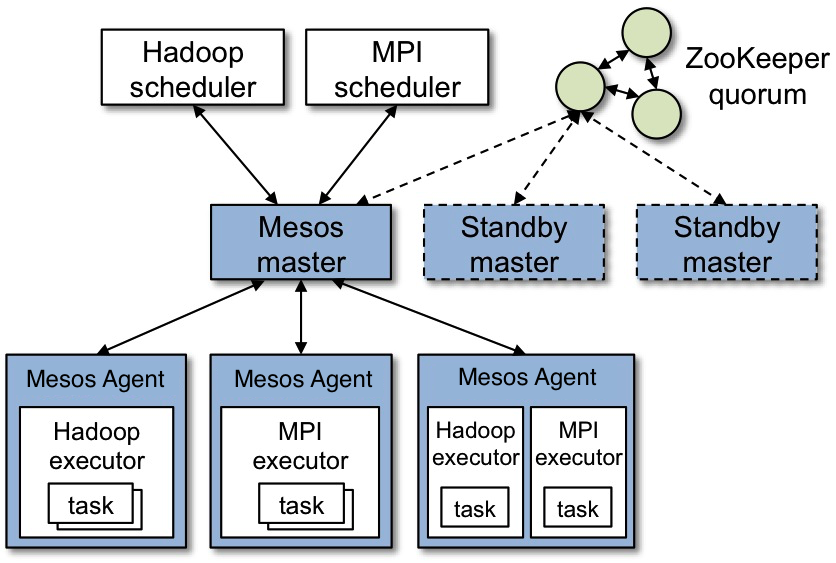
**4. Apache Mesos**



Apache Mesos

Apache Mesos is a distributed systems kernel.

Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Kafka, Elasticsearch) with API’s for resource management and scheduling across entire datacenter and cloud environments.



Mesos

Apache Mesos abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to be built and run effectively.

**III. In-Memory Processing**

**5. Apache Spark**

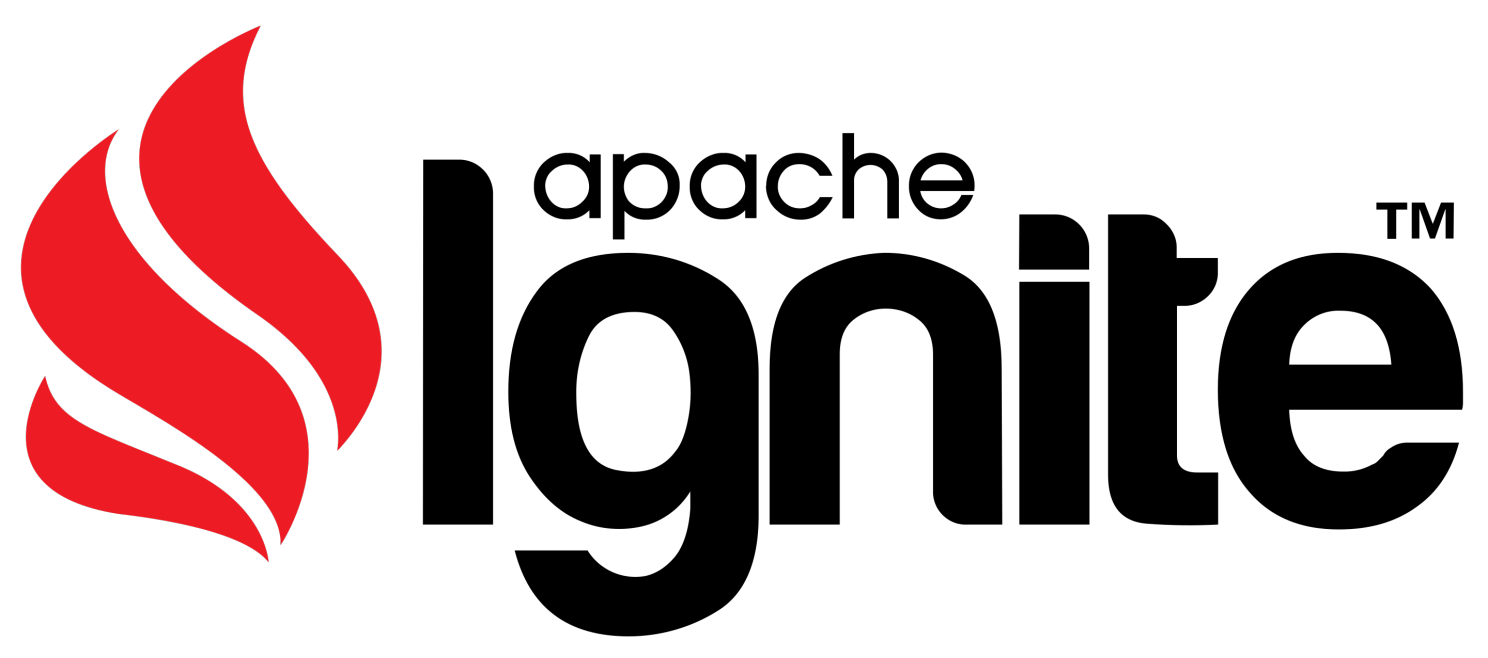
Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.



**Spark and Apache Hadoop**

Spark is a general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning.

**6. Apache Ignite**



Apache Ignite as a distributed in-memory database scales horizontally across memory and disk without compromise.

Apache Ignite works with memory, disk, and Intel Optane as active storage tiers. This multi-tier architecture combines the advantages of in-memory computing with disk durability and strong consistency, all in one system.

**IV. Stream Processing**

**7. Apache Kafka**



Apache Kafka is a distributed event streaming platform for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premise as well as cloud environments.

**8. Apache Storm**



Apache Storm is a distributed real-time computation system. Apache Storm enables you to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing.

Apache Storm has many use cases: real-time analytics, online machine learning, continuous computation, distributed RPC, and ETL.

Apache Storm integrates with the queueing and database technologies. An Apache Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.

**9. Apache Flink**



Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.

**V. SQL Over Hadoop**

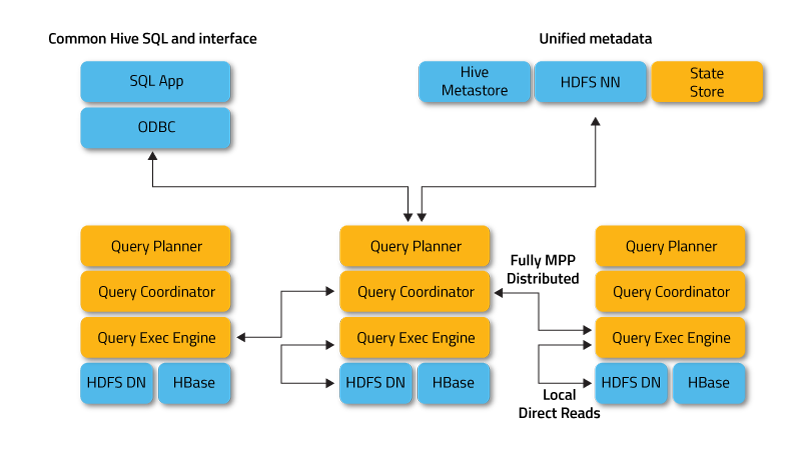
**10. Apache Impala**



Apache Impala

Impala provides interactive SQL queries directly on your Apache Hadoop data stored in HDFS, HBase, or the Amazon Simple Storage Service (S3). In addition to using the same unified storage platform, Impala also uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Impala query UI in Hue) as Apache Hive. This provides a familiar and unified platform for real-time or batch-oriented queries.

Impala is an addition to tools available for querying big data. Impala does not replace the batch processing frameworks built on MapReduce such as Hive. Hive and other frameworks built on MapReduce are best suited for long running batch jobs, such as those involving batch processing of Extract, Transform, and Load (ETL) type jobs.



Impala Architecture

Impala raises the bar for SQL query performance on Apache Hadoop while retaining a familiar user experience. With Impala, you can query data, whether stored in HDFS or Apache HBase – including SELECT, JOIN, and aggregate functions – in real time.

**11. Apache Drill**



Apache Drill

Drill is an SQL query engine for Big Data exploration. Drill is designed from the ground up to support high-performance analysis on the semi-structured and rapidly evolving data coming from modern Big Data applications, while still providing the familiarity and ecosystem of ANSI SQL, the industry-standard query language. Drill provides plug-and-play integration with existing Apache Hive and Apache HBase deployments.

Apache Drill is a low latency distributed query engine for large-scale datasets, including structured and semi-structured/nested data.

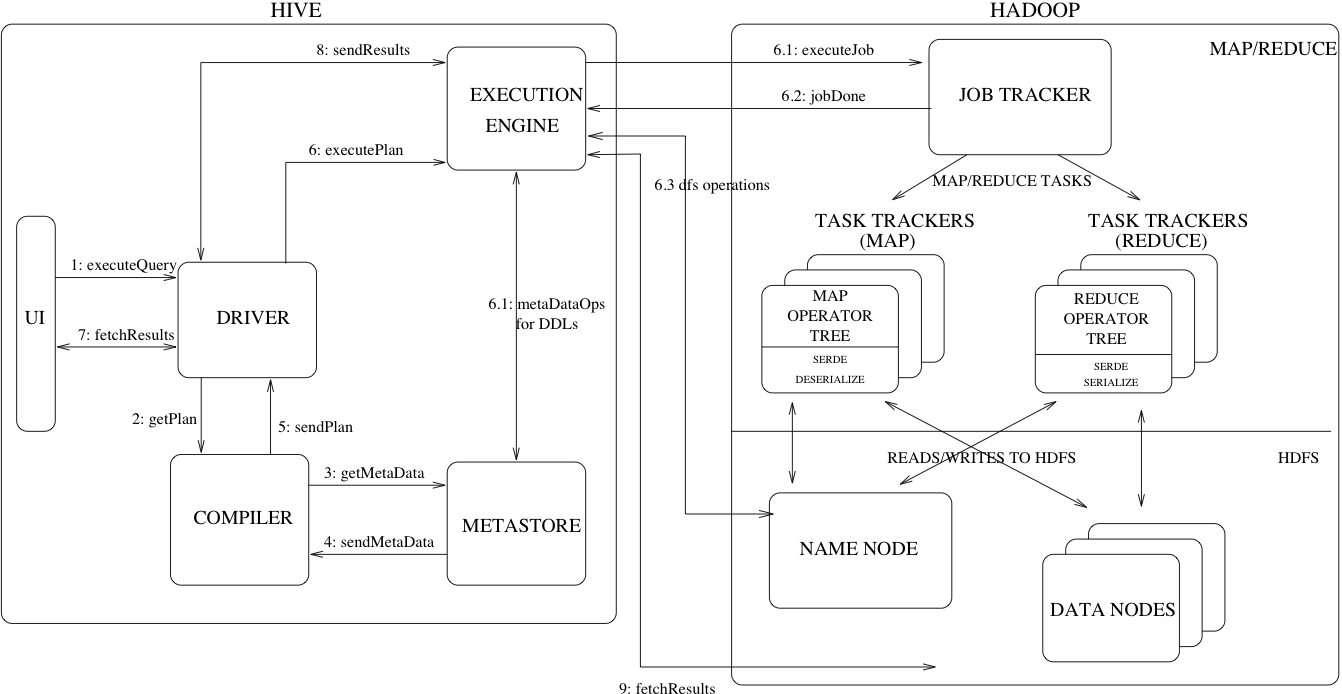
**12. Apache Hive**



Apache Hive

Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale. Hive Metastore(HMS) provides a central repository of metadata that can easily be analyzed to make informed, data driven decisions, and therefore it is a critical component of many data lake architectures. Hive is built on top of Apache Hadoop and supports storage on S3, ADLS, GS, etc. Hive allows users to read, write, and manage petabytes of data using SQL.

Hive is a data warehousing infrastructure based on Apache Hadoop. Hadoop provides massive scale out and fault tolerance capabilities for data storage and processing on commodity hardware.



Hive and Hadoop

Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data. It provides SQL which enables users to do ad-hoc querying, summarization and data analysis easily. At the same time, Hive's SQL gives users multiple places to integrate their own functionality to do custom analysis, such as User Defined Functions (UDFs).

**VI. NoSQL Database**

**13. Apache HBase**



Apache HBase

Apache HBase is the Hadoop database, a distributed, scalable, big data store.

Apache HBase is a distributed, versioned, non-relational database modeled after Google's Bigtable: A Distributed Storage System for Structured Data. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database isn’t an RDBMS which supports SQL as its primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages.

**HBase and Hadoop/HDFS**

HDFS is a distributed file system that is well suited for the storage of large files. However, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups.

**VII. Search Engine**

**14. Apache Solr**



Apache Solr

Apache Solr is a search server built on top of Apache Lucene, an open source, Java- based, information retrieval library. Solr is designed to drive powerful document retrieval or analytical applications involving unstructured data, semi-structured data or a mix of unstructured and structured data. It also has secondary support for limited relational, graph, statistical, data analysis or storage related use cases.

**15. Apache Lucene**



Apache Lucene

Apache Lucene is a high-performance, full-featured search engine library written entirely in Java. It is a technology suitable for nearly any application that requires structured search, full-text search, faceting, nearest-neighbor search across high-dimensionality vectors, spell correction or query suggestions.

Lucene Core is a Java library providing powerful indexing and search features, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities. The PyLucene sub project provides Python bindings for Lucene Core.

Although Lucene provides the ability to create your own queries through its API, it also provides a rich query language through the Query Parser, a lexer which interprets a string into a Lucene Query using JavaCC.

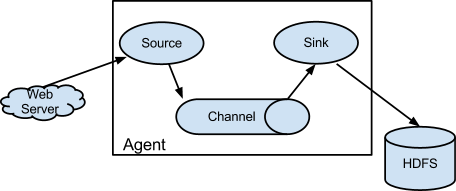
**VIII. Data Piping**

**16. Apache Flume**



Apache Flume

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.



Apache Flume Data Flow Model

An Event is a unit of data that flows through a Flume agent. The Event flows from Source to Channel to Sink, and is represented by an implementation of the Event interface. An Event carries a payload (byte array) that is accompanied by an optional set of headers (string attributes).

A Flume agent is a process (JVM) that hosts the components that allow Events to flow from an external source to a external destination.

A Source consumes Events having a specific format, and those Events are delivered to the Source by an external source like a web server.

The Channel is a passive store that holds the Event until that Event is consumed by a Sink.

One type of Channel available in Flume is the FileChannel which uses the local filesystem as its backing store.

A Sink is responsible for removing an Event from the Channel and putting it into an external repository like HDFS (in the case of an HDFSEventSink) or forwarding it to the Source at the next hop of the flow. The Source and Sink within the given agent run asynchronously with the Events staged in the Channel.

**17. Apache NiFi**



Apache NiFi

Apache NiFi is a dataflow system based on the concepts of flow-based programming. It supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.

NiFi has a web-based user interface for design, control, feedback, and monitoring of dataflows. It is highly configurable along several dimensions of quality of service, such as loss-tolerant versus guaranteed delivery, low latency versus high throughput, and priority-based queuing. NiFi provides fine-grained data provenance for all data received, forked, joined cloned, modified, sent, and ultimately dropped upon reaching its configured end-state.

**IX. Machine Learning**

**18. Apache Spark MLlib**

MLlib is Apache Spark's scalable machine learning library.

MLlib fits into Spark's APIs and interoperates with NumPy in Python and R libraries. You can use any Hadoop data source (e.g. HDFS, HBase, or local files) to plug into Hadoop workflows.

At a high level, MLlib provides tools such as:

• ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering

• Featurization: feature extraction, transformation, dimensionality reduction, and selection

• Pipelines: tools for constructing, evaluating, and tuning ML Pipelines

• Persistence: saving and load algorithms, models, and Pipelines

• Utilities: linear algebra, statistics, data handling, etc.

**19. Apache Mahout**



Apache Mahout

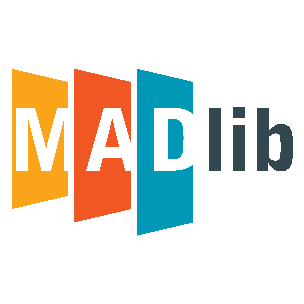
Apache Mahout is a scalable and versatile machine learning library designed for distributed data processing. It offers a comprehensive set of algorithms for various tasks, including classification, clustering, recommendation, and pattern mining. Built on top of the Apache Hadoop ecosystem, Mahout leverages MapReduce and Spark to enable data processing on large-scale datasets.

Apache Mahout is a distributed linear algebra framework and mathematically expressive Scala DSL designed to let mathematicians, statisticians, and data scientists implement their own algorithms. Apache Spark is the recommended out-of-the-box distributed back-end, or can be extended to other distributed backends.

Mahout has support for multiple distributed backends (including Apache Spark).

Mahout has modular native solvers for CPU/GPU/CUDA acceleration.

**20. Apache MADlib**



Apache MADlib

Apache MADlib is an open-source library for scalable in-database analytics. It provides data-parallel implementations of mathematical, statistical, graph and machine learning methods for structured and unstructured data.

**X. Scripting**

**21. Apache Pig**



Apache Pig

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin.

**XI. Meta Data Management**

**22. Apache Atlas**



Apache Atlas

Atlas is a scalable and extensible set of core foundational governance services – enabling enterprises to effectively and efficiently meet their compliance requirements within Hadoop and allows integration with the whole enterprise data ecosystem.

Apache Atlas provides open metadata management and governance capabilities for organizations to build a catalog of their data assets, classify and govern these assets and provide collaboration capabilities around these data assets for data scientists, analysts and the data governance team.

**XII. Coordinate & Management**

**23. Apache Ambari**



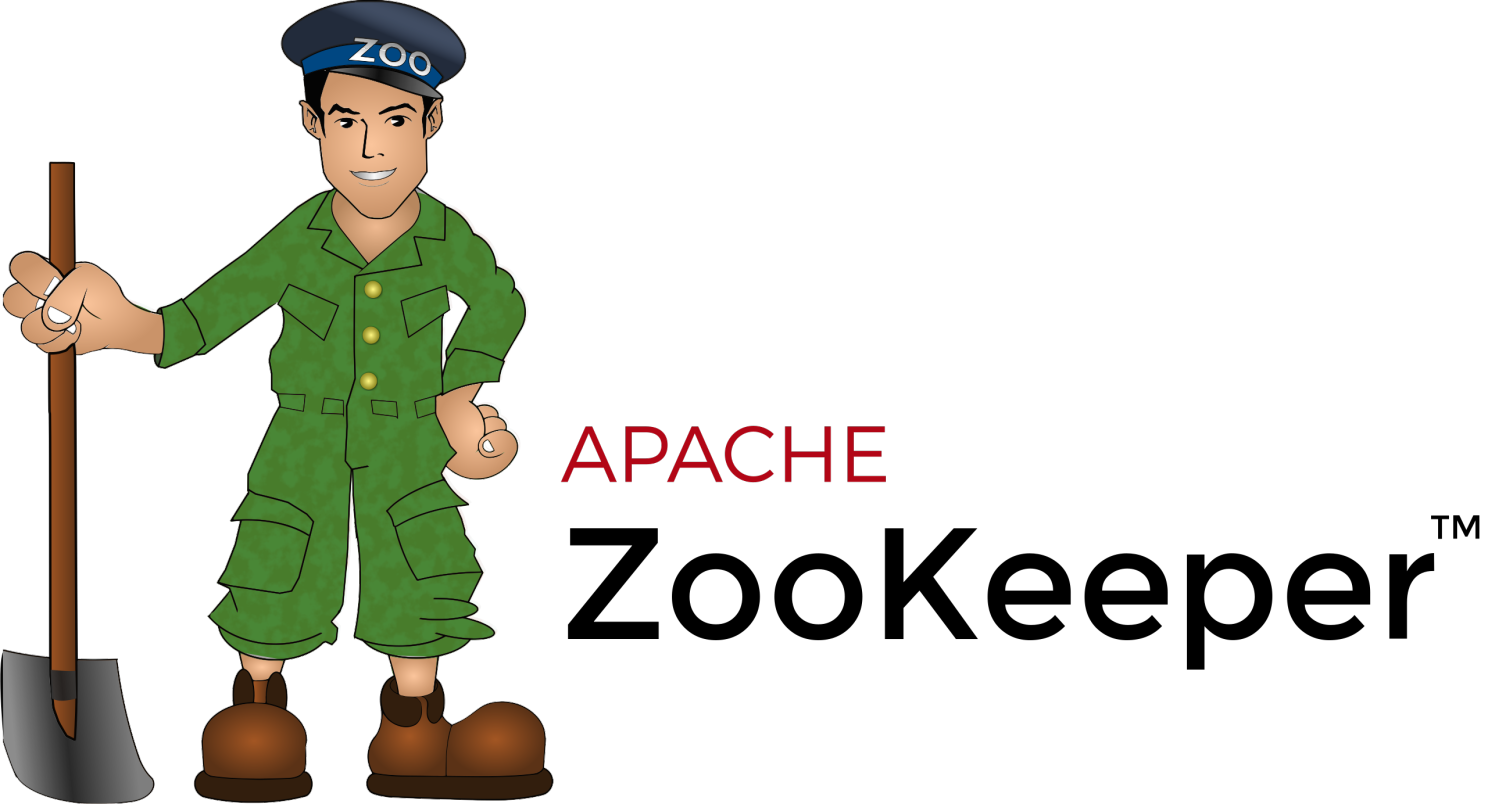
Apache Ambari

Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides a Hadoop management web UI backed by its RESTful APIs.

Ambari enables system administrators, provision, manage, and monitor a Hadoop Cluster.

Ambari enables application developers and system integrators to integrate Hadoop provisioning, management, and monitoring capabilities to their own applications with the Ambari REST APIs.

**24. Apache ZooKeeper**



Apache ZooKeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.

Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

**XIII. Security**

**25. Apache Ranger**



Apache Ranger

Apache Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform/ecosystem.

With the advent of Apache YARN, the Hadoop platform can now support a true data lake architecture. Enterprises can potentially run multiple workloads, in a multi-tenant environment. Data security within Hadoop needs to evolve to support multiple use cases for data access, while also providing a framework for central administration of security policies and monitoring of user access.

Apache Ranger supports fine grained authorization and auditing for Apache projects

• Apache Hadoop

• Apache Hive

• Apache HBase

• Apache Storm

• Apache Knox

• Apache Solr

• Apache Kafka

• YARN

**Ranger and Hadoop**

Apache Ranger at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

**XIV. Scheduler**

**26. Apache Airflow**



Apache Airflow is a platform for developing, scheduling, and monitoring batch-oriented workflows.

Airflow’s extensible Python framework enables you to build workflows connecting with virtually any technology. A web interface helps manage the state of your workflows. Airflow is deployable in many ways, varying from a single process on your laptop to a distributed setup to support even the biggest workflows.

**27. Apache Oozie**



Apache Oozie

Oozie is a workflow scheduler system to manage Apache Hadoop jobs.

Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.

Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.

Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, and Distcp) as well as system specific jobs (such as Java programs and shell scripts).

Oozie is a scalable, reliable and extensible system.

**XV. Data Format**

**28. Apache Parquet**



Apache Parquet

Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.

**29. Apache Avro**



Apache Avro

Apache Avro is a serialization format for record data and streaming data pipelines.

Avro is a data serialization system that provides:

• Rich data structures.

• A compact, fast, binary data format.

• A container file, to store persistent data.

• Remote procedure call (RPC).

• Simple integration with dynamic languages. Code generation is not required to read or write data files nor to use or implement RPC protocols. Code generation as an optional optimization, only worth implementing for statically typed languages.

**30. Apache ORC**



Apache ORC

Apache ORC is the smallest, fastest columnar storage for Hadoop workloads.

• Includes support for ACID transactions and snapshot isolation.

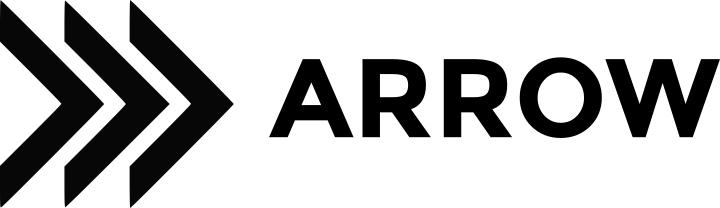
• Built-in Indexes including minimum, maximum, and bloom filters for each column.

• Supports all of Hive's complex types including the compound types: structs, lists, maps, and unions.

ORC is a self-describing type-aware columnar file format designed for Hadoop workloads. It is optimized for large streaming reads, but with integrated support for finding required rows quickly.

Storing data in a columnar format lets the reader read, decompress, and process only the values that are required for the current query. Because ORC files are type-aware, the writer chooses the most appropriate encoding for the type and builds an internal index as the file is written.

**31. Apache Arrow**



Apache Arrow

Apache Arrow is a software development platform for building high performance applications that process and transport large data sets. It is designed to both improve the performance of analytical algorithms and the efficiency of moving data from one system (or programming language to another).

A critical component of Apache Arrow is its in-memory columnar format, a standardized, language-agnostic specification for representing structured, table-like datasets in-memory. This data format has a rich data type system (included nested and user-defined data types) designed to support the needs of analytic database systems, data frame libraries, and more.

**32. Apache Thrift**



Apache Thrift

The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between different programming languages.

Apache Thrift allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages. Instead of writing a load of boilerplate code to serialize and transport your objects and invoke remote methods, you can get right down to business.

**Apache Knox, Apache Tez, Apache Phoenix, Apache Zeppelin**

**33. Apache Knox**



Apache Knox

The Apache Knox Gateway is an Application Gateway for interacting with the REST APIs and UIs of Apache Hadoop deployments.

The Knox Gateway provides a single access point for all REST and HTTP interactions with Apache Hadoop clusters.

Knox delivers three groups of user facing services:

• Proxying Services

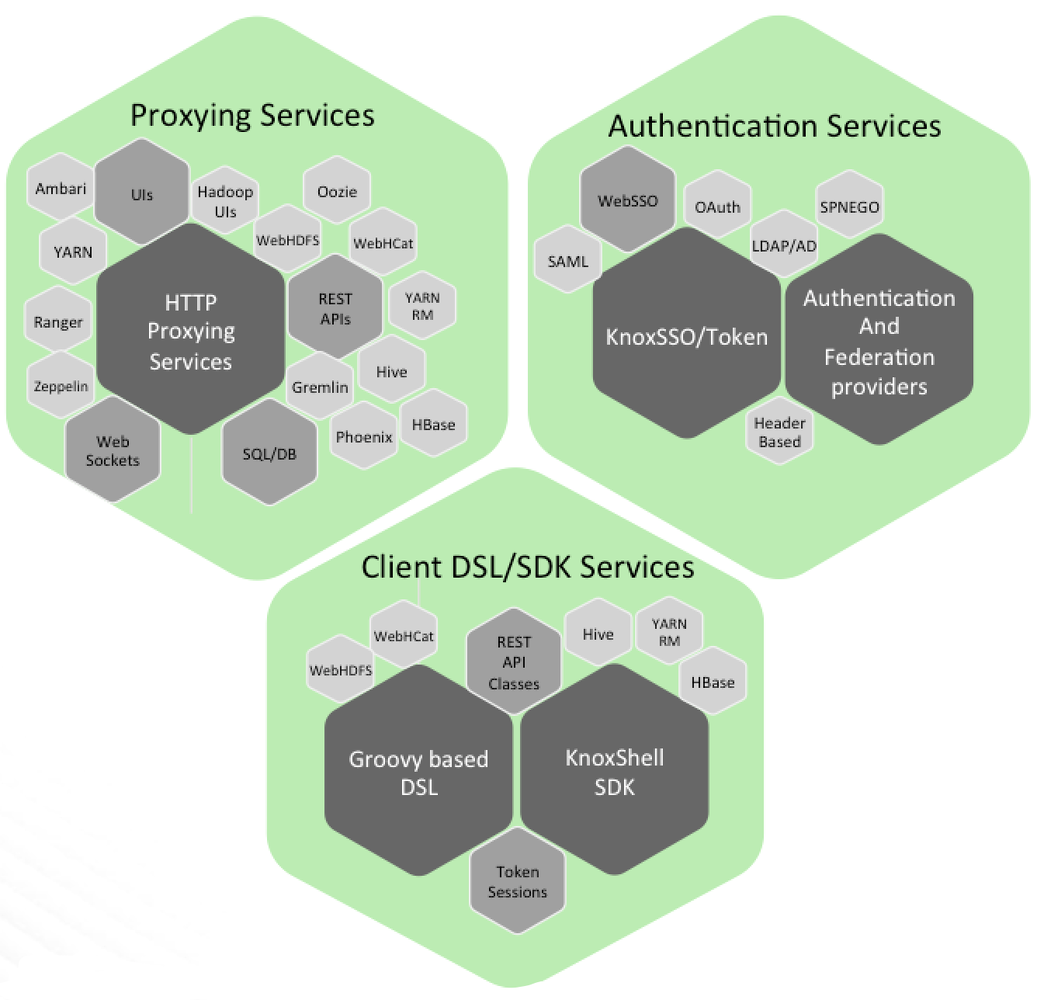
Primary goals of the Apache Knox project is to provide access to Apache Hadoop via proxying of HTTP resources.

• Authentication Services

Authentication for REST API access as well as WebSSO flow for UIs. LDAP/AD, Header based PreAuth, Kerberos, SAML, OAuth are all available options.

• Client Services

Client development can be done with scripting through DSL or using the Knox Shell classes directly as SDK.



Apache Knox Proxying Services, Authentication Services, Client Services

The Knox API Gateway is designed as a reverse proxy with consideration for pluggability in the areas of policy enforcement, through providers and the backend services for which it proxies requests.

**34. Apache Tez**

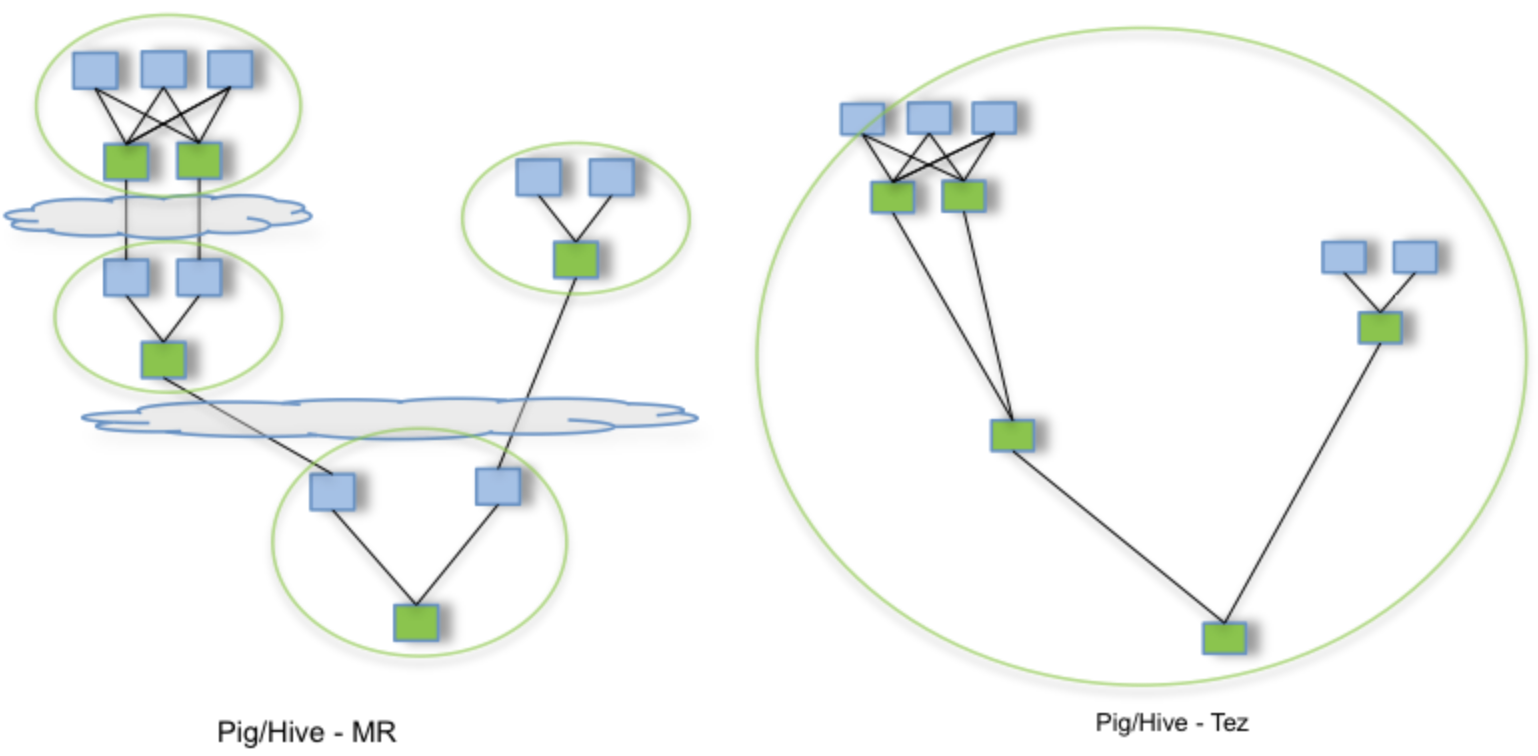


Apache Tez

The Apache TEZ project is aimed at building an application framework which allows for a complex directed-acyclic-graph of tasks for processing data. It is currently built atop Apache Hadoop YARN.

Apache Tez is an extensible framework for building high performance batch and interactive data processing applications, coordinated by YARN in Apache Hadoop. Tez improves the MapReduce paradigm by dramatically improving its speed, while maintaining MapReduce’s ability to scale to petabytes of data. Important Hadoop ecosystem projects like Apache Hive and Apache Pig use Apache Tez.

By allowing projects like Apache Hive and Apache Pig to run a complex DAG of tasks, Tez can be used to process data, that earlier took multiple MR jobs, now in a single Tez job.



Multiple Pig/Hive-MR jobs vis-à-vis Single Pig/Hive-Tez job

**35. Apache Phoenix**



Apache Phoenix

Apache Phoenix enables OLTP and operational analytics in Hadoop for low latency applications.

Apache Phoenix combines the best of both worlds:

• the power of standard SQL and JDBC APIs with full ACID transaction capabilities and

• the flexibility of late-bound, schema-on-read capabilities from the NoSQL world by leveraging HBase as its backing store

Apache Phoenix is fully integrated with other Hadoop products such as Spark, Hive, Pig, Flume, and Map Reduce.

**36. Apache Zeppelin**



Apache Zeppelin

Apache Zeppelin is a web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala, Python, and R.

Apache Zeppelin is Multi-purpose Notebook

• Data Ingestion

• Data Discovery

• Data Transformation

• Data Analytics

• Data Visualization & Collaboration

Apache Zeppelin with Spark integration provides

• Automatic SparkContext and SQLContext injection

• Runtime jar dependency loading from local filesystem or maven repository

• Canceling job and displaying its progress



Apache

**Apache MapReduce**

MapReduce is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. As the processing component, MapReduce is the heart of Apache Hadoop.

The term "MapReduce" refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

MapReduce programming offers several benefits to help you gain valuable insights from your big data:

• Scalability: Businesses can process petabytes of data stored in the Hadoop Distributed File System (HDFS).

• Flexibility: Hadoop enables easier access to multiple sources of data and multiple types of data.

• Speed: With parallel processing and minimal data movement, Hadoop offers fast processing of massive amounts of data.

An example of MapReduce

Here is a very simple example of MapReduce. No matter the amount of data you need to analyze, the key principles remain the same.

Assume you have five files and each file contains two columns. These columns represent a key and a value in Hadoop terms that represent a city and the corresponding temperature recorded in that city for the various measurement days.

The city is the key and the temperature is the value. For example, Toronto, 20. Out of all the data we have collected, you want to find the maximum temperature for each city across the data files. It's important to note that each file might have the same city represented multiple times.

Using the MapReduce framework, you can break this down into five map tasks, where each mapper works on one of the five files. The mapper task goes through the data and returns the maximum temperature for each city.

For example, the results produced from one mapper task for the data provided would look like this: (Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33).

Assume the other four mapper tasks (working on the other four files not shown here) produced these intermediate results:

(Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37) (Toronto, 32) (Whitby, 20) (New York, 33) (Rome, 38) (Toronto, 22) (Whitby, 19) (New York, 20) (Rome, 31) (Toronto, 31) (Whitby, 22) (New York, 19) (Rome, 30).

All five of these output streams would be fed into the reduce tasks, which combine the input results and output a single value for each city, producing a final result set as follows: (Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38).

As an analogy, you can think of map and reduce tasks as the way a census was conducted in Roman times. In this method, the census bureau would dispatch its people to each city in the empire. Each census taker in each city would be tasked to count the number of people in that city and then return their results to the capital city. There, the results from each city would be reduced to a single count (the sum of all cities) to determine the overall population of the empire.

This process involves mapping of people to cities, in parallel, and then combining the results through reduction. It is much more efficient than sending a single person to count every person in the empire in a serial fashion.

**Hadoop vis-à-vis Spark**

Apache Hadoop and Apache Spark are two open-source frameworks you can use to manage and process large volumes of data for analytics. Organizations must process data at scale and speed to gain real-time insights for business intelligence. Apache Hadoop allows you to cluster multiple computers to analyze massive datasets in parallel more quickly. Apache Spark uses in-memory caching and optimized query execution for fast analytic queries against data of any size. Spark is a more advanced technology than Hadoop, as Spark uses artificial intelligence and machine learning (AI/ML) in data processing. However, many companies use Spark and Hadoop together to meet their data analytics goals.

**Similarities Between Hadoop and Spark**

**Distributed big data processing**

Big data is collected frequently, continuously, and at scale in various formats.

To store, manage, and process big data, Apache Hadoop separates datasets into smaller subsets or partitions. It then stores the partitions over a distributed network of servers. Likewise, Apache Spark processes and analyzes big data over distributed nodes to provide business insights.

Depending on the use cases, you might need to integrate both Hadoop and Spark with different software for optimum functionality.

**Fault tolerance**

Apache Hadoop continues to run even if one or several data processing nodes fail. It makes multiple copies of the same data block and stores them across several nodes. When a node fails, Hadoop retrieves the information from another node and prepares it for data processing.

Meanwhile, Apache Spark relies on a special data processing technology called Resilient Distributed Dataset (RDD). With RDD, Apache Spark remembers how it retrieves specific information from storage and can reconstruct the data if the underlying storage fails.

**Key Components of Hadoop and Spark**

Hadoop Components

• Hadoop Distributed File System (HDFS) is a special file system that stores large datasets across multiple computers. These computers are called Hadoop clusters.

• Yet Another Resource Negotiator (YARN) schedules tasks and allocates resources to applications running on Hadoop.

• Hadoop MapReduce allows programs to break large data processing tasks into smaller ones and runs them in parallel on multiple servers.

• Hadoop Common, or Hadoop Core, provides the necessary software libraries for other Hadoop components.

Spark Components

• Spark Core coordinates the basic functions of Apache Spark. These functions include memory management, data storage, task scheduling, and data processing.

• Spark SQL allows you to process data in Spark's distributed storage.

• Spark Streaming and Structured Streaming allow Spark to stream data efficiently in real time by separating data into tiny continuous blocks.

• Machine Learning Library (MLlib) provides several machine learning algorithms that you can apply to big data.

• GraphX allows you to visualize and analyze data with graphs.

**Key Differences Between Hadoop and Spark**

Both Hadoop and Spark allow you to process big data in different ways.

Apache Hadoop was created to delegate data processing to several servers instead of running the workload on a single machine.

Meanwhile, Apache Spark is a newer data processing system that overcomes key limitations of Hadoop. Despite its ability to process large datasets, Hadoop only does so in batches and with substantial delay.

Architecture

Hadoop has a native file system called Hadoop Distributed File System (HDFS). HDFS lets Hadoop divide large data blocks into multiple smaller uniform ones. Then, it stores the small data blocks in server groups.

Meanwhile, Apache Spark does not have its own native file system. Many organizations run Spark on Hadoop’s file system to store, manage, and retrieve data.

Performance

Hadoop can process large datasets in batches but may be slower. To process data, Hadoop reads the information from external storage and then analyzes and inputs the data to software algorithms.

For each data processing step, Hadoop writes the data back to the external storage, which increases latency. Hence, it is unsuitable for real-time processing tasks but ideal for workloads with tolerable time delays. For example, Hadoop is suited to analyze monthly sales records. But it may not be the best choice to determine real-time brand sentiment from social media feeds.

Apache Spark, on the other hand, is designed to process enormous amounts of data in real time.

Instead of accessing data from external storage, Spark copies the data to RAM before processing it. It only writes the data back to external storage after completing a specific task. Writing and reading from RAM are exponentially faster than doing the same with an external drive. Moreover, Spark reuses the retrieved data for numerous operations.

Therefore, Spark performs better than Hadoop in varying degrees for both simple and complex data processing.

Machine learning

Apache Spark provides a machine learning library called MLlib. Data scientists use MLlib to run regression analysis, classification, and other machine learning tasks. You can also train machine learning models with unstructured and structured data and deploy them for business applications.

In contrast, Apache Hadoop does not have built-in machine learning libraries. Instead, you can integrate Spark with other software like Apache Mahout to build machine learning systems. The choice of software depends on the specific requirements of the workload. You can consider things like the size and complexity of data, the type of machine learning models you want to use, and the performance and scalability requirements of your application.

Security

Apache Hadoop is designed with robust security features to safeguard data. For example, Hadoop uses encryption and access control to prevent unauthorized parties from accessing and manipulating data storage.

Apache Spark, however, has limited security protections on its own. According to Apache Software Foundation, you must enable Spark’s security feature and ensure that the environment it runs on is secure.

Spark supports several deployment types, with some more secure than others. For example, deploying Spark on Hadoop improves overall security due to Hadoop's encrypted distributed storage.

Scalability

It takes less effort to scale with Hadoop than Spark. If you need more processing power, you can add additional nodes or computers on Hadoop at a reasonable cost.

In contrast, scaling the Spark deployments typically requires investing in more RAM. Costs can add up quickly for on-premises infrastructure.

Cost

Apache Hadoop is more affordable to set up and run because it uses hard disks for storing and processing data. You can set up Hadoop on standard or low-end computers.

Meanwhile, it costs more to process big data with Spark as it uses RAM for in-memory processing. RAM is generally more expensive than a hard disk with equal storage size.

**Usage of Hadoop and Spark**

Apache Spark was introduced to overcome the limitations of Hadoop’s external storage-access architecture. Apache Spark replaces Hadoop’s original data analytics library, MapReduce, with faster machine learning processing capabilities.

However, Spark is not mutually exclusive with Hadoop. While Apache Spark can run as an independent framework, many organizations use both Hadoop and Spark for big data analytics.

Depending on specific business requirements, you can use Hadoop, Spark, or both for data processing. Here are some things you might consider in your decision.

Cost-effective scaling

Apache Hadoop is the better option for building and scaling a cost-effective data processing pipeline. Adding more computers to an existing Hadoop cluster can increase Hadoop’s processing capacity. This is more affordable than purchasing additional RAM to scale the Apache Spark framework.

Batch processing

Batch processing refers to when you process large numbers of data without being confined to a stipulated timeline. When batch processing is preferred, organizations use Apache Hadoop because it supports parallel processing across multiple nodes. For example, you can use Hadoop to generate non-time-sensitive inventory reports from tens of thousands of records.

Real-time analytics

Use Apache Spark if you’re dealing with fast-moving data. A data stream is information or data transmitted continuously by software. Apache Spark can process live data streams and provide insightful analysis in real time. For example, financial institutions use Apache Spark to detect fraud in ongoing transactions and alert bank officers.

Machine learning capability

Machine learning involves training software functions or models with large numbers of datasets. Apache Spark is more suitable for such tasks because of its built-in machine learning library. This means Spark can train machine learning models in real time without additional integrations.

Security, speed, and interactive analytics

You can use Hadoop and Spark to benefit from the strengths of both frameworks. Hadoop provides secure and affordable distributed processing. If you run Spark on Hadoop, you can shift time-sensitive workloads, such as graph analytics tasks, to Spark’s in-memory data processors. You get performance and secure external storage processing for your analytics.

**Summary of Differences Between Hadoop and Spark**

Summary of Differences Between Hadoop and Spark

• Architecture

Hadoop stores and processes data on external storage.

Spark stores and process data on internal memory.

• Performance

Hadoop processes data in batches.

Spark processes data in real time.

• Cost

Hadoop is affordable.

Spark is comparatively more expensive.

• Scalability

Hadoop is easily scalable by adding more nodes.

Spark is comparatively more challenging.

• Machine learning

Hadoop integrates with external libraries to provide machine learning capabilities.

Spark has built-in machine learning libraries.

• Security

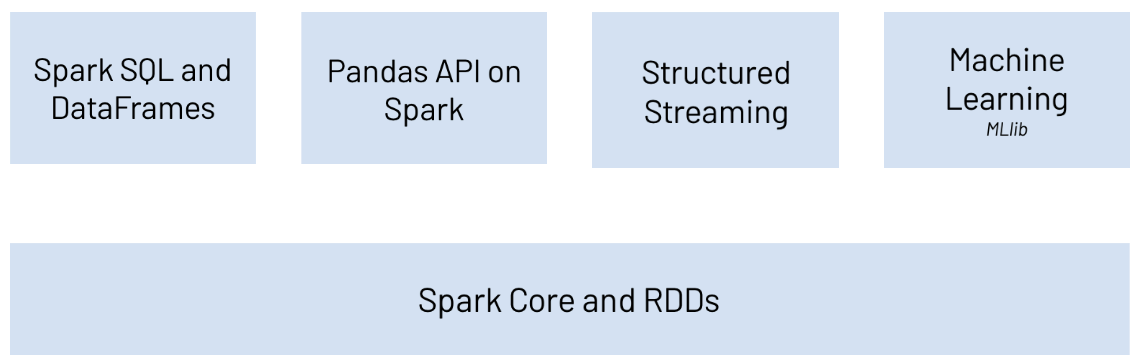
Hadoop has strong security features, storage encryption, and access control.

Spark has basic security. IT relies on you setting up a secure operating environment for the Spark deployment.

**PySpark**

PySpark is the Python API for Apache Spark. It enables you to perform real-time, large-scale data processing in a distributed environment using Python.

PySpark supports all of Spark’s features such as Spark SQL, DataFrames, Structured Streaming, Machine Learning (MLlib) and Spark Core.



PySpark

Spark SQL and DataFrames

Spark SQL is Apache Spark’s module for working with structured data. It allows you to seamlessly mix SQL queries with Spark programs. With PySpark DataFrames you can efficiently read, write, transform, and analyze data using Python and SQL.

Pandas API on Spark

Pandas API on Spark allows you to scale your pandas workload to any size by running it distributed across multiple nodes.

Structured Streaming

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.

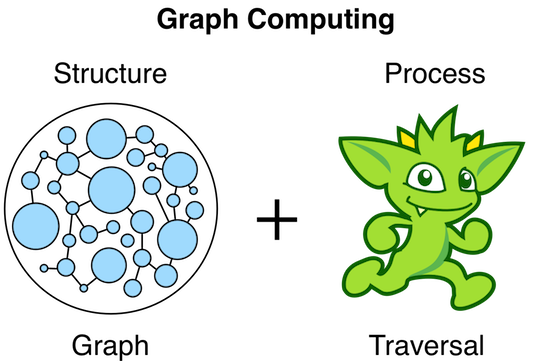
Machine Learning (MLlib)

Built on top of Spark, MLlib is a scalable machine learning library that provides a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

Spark Core and RDDs

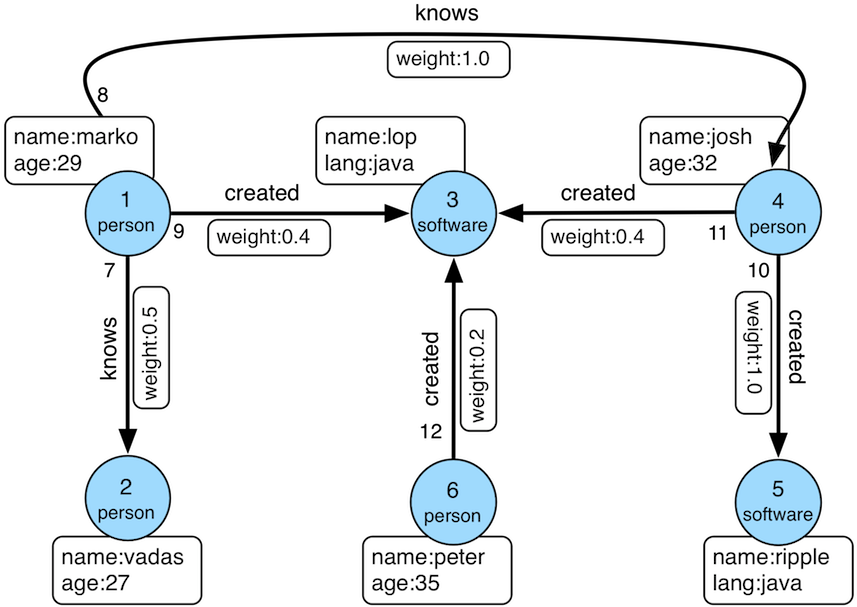
Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides RDDs (Resilient Distributed Datasets) and in-memory computing capabilities.

**Graph Computing**



Graph Computing

A graph is a data structure composed of vertices (nodes, dots) and edges (arcs, lines). When modeling a graph in a computer and applying it to modern data sets and practices, the generic mathematically-oriented, binary graph is extended to support both labels and key/value properties. This structure is known as a property graph. More formally, it is a directed, binary, attributed multi-graph.



Property Graph

Similar to computing in general, graph computing makes a distinction between structure (graph) and process (traversal). The structure of the graph is the data model defined by a vertex/edge/property topology. The process of the graph is the means by which the structure is analyzed. The typical form of graph processing is called a traversal.

**Graph Structure**

A graph’s structure is the topology formed by the explicit references between its vertices, edges, and properties. A vertex has incident edges. A vertex is adjacent to another vertex if they share an incident edge. A property is attached to an element and an element has a set of properties. A property is a key/value pair, where the key is always a character String.

**Graph Process**

The primary way in which graphs are processed are via graph traversals. A traversal is an algorithmic walk across the elements of a graph according to the referential structure explicit within the graph data structure. For example: "What software does vertex 1’s friends work on?"

This English-statement can be represented in the following algorithmic/traversal fashion:

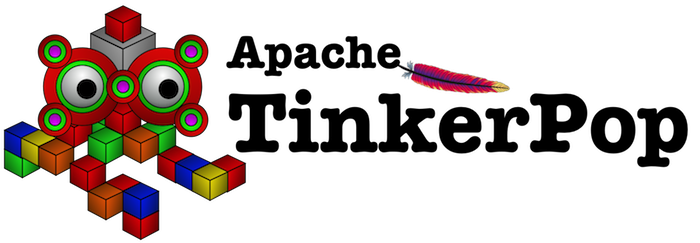
1. Start at vertex 1.

2. Walk the incident known-edges to the respective adjacent friend vertices of 1.

3. Move from those friend-vertices to software-vertices via created-edges.

4. Finally, select the name-property value of the current software-vertices.

**Apache TinkerPop**

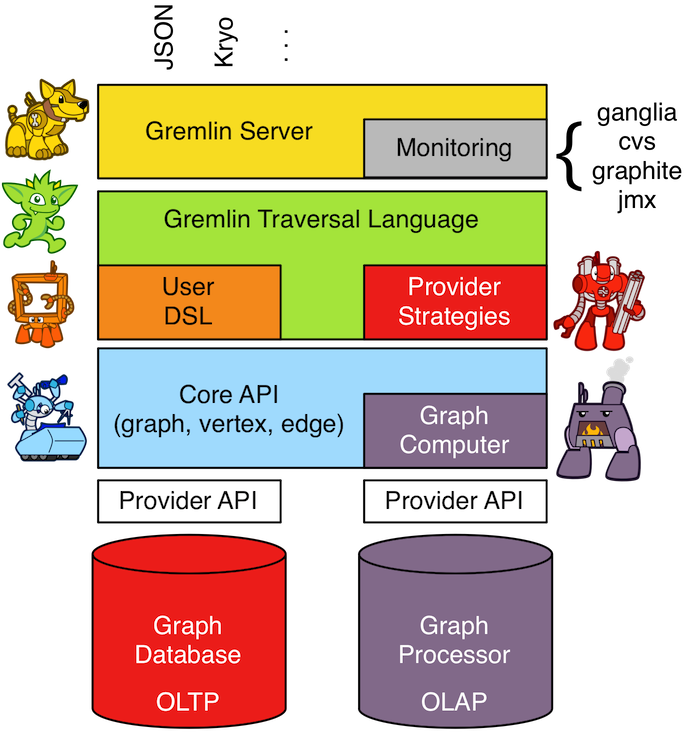


Apache TinkerPop

Apache TinkerPop is a graph computing framework for both graph databases Online Transactional Processing (OLTP) and graph analytic systems Online Analytical Processing (OLAP).

TinkerPop’s role in graph computing is to provide the appropriate interfaces for graph providers and users to interact with graphs over their structure and process. When a graph system implements the TinkerPop structure and process APIs, their technology is considered TinkerPop-enabled and becomes nearly indistinguishable from any other TinkerPop-enabled graph system save for their respective time and space complexity.

TinkerPop is an abstraction layer over different graph databases and different graph processors.



TinkerPop Abstraction Layer

**Gremlin Query Language**



Gremlin

Gremlin is the graph traversal language of Apache TinkerPop. Gremlin is a functional, data-flow language that enables users to succinctly express complex traversals on (or queries of) their application's property graph.

**OLTP and OLAP Traversals**

TinkerPop-enabled graph systems execute Gremlin traversals, but also, every Gremlin traversal can be evaluated as either a real-time database query or as a batch analytics query. The former is known as an online transactional process (OLTP) and the latter as an online analytics process (OLAP). This universality is made possible by the Gremlin traversal machine. This distributed, graph-based virtual machine understands how to coordinate the execution of a multi-machine graph traversal. The benefit is that the user does not need to learn both a database query language and a domain-specific BigData analytics language (e.g. Spark DSL, MapReduce, etc.). Gremlin is all that is required to build a graph-based application because the Gremlin traversal machine will handle the rest.

**Imperative and Declarative Traversals**

A Gremlin traversal can be written in either an imperative (procedural) manner, a declarative (descriptive) manner, or in a hybrid manner containing both imperative and declarative aspects.

An imperative Gremlin traversal tells the traversers how to proceed at each step in the traversal.

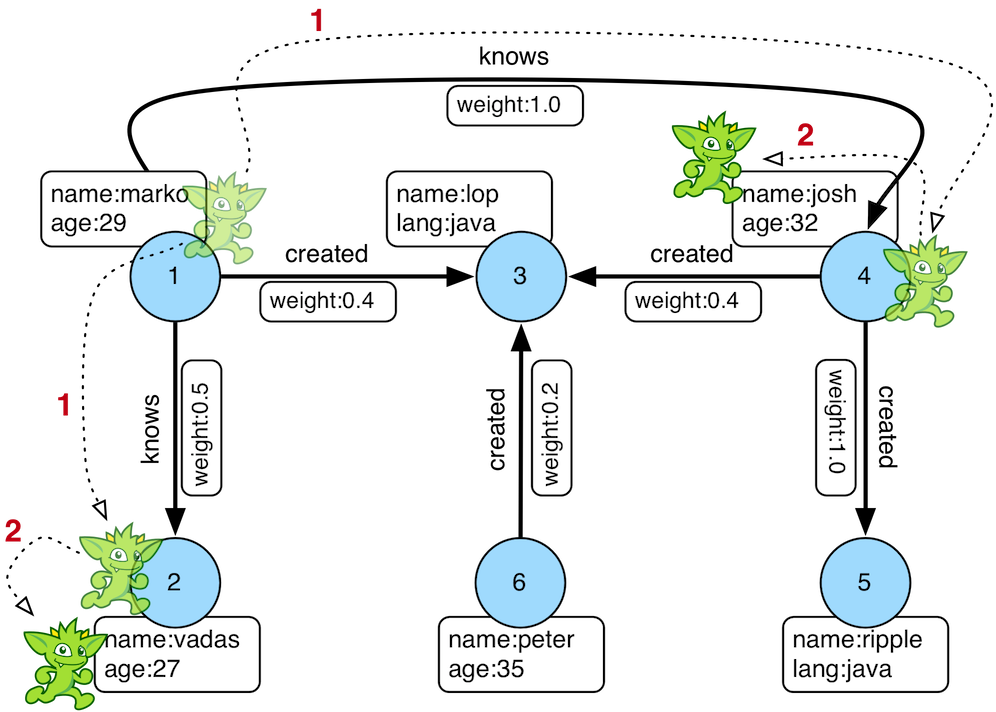
A declarative Gremlin traversal does not tell the traversers the order in which to execute their walk, but instead, allows each traverser to select a pattern to execute from a collection of (potentially nested) patterns.

Given the TinkerPop graph, the following algorithmic/traversal will return the names of all the people that the marko-vertex knows.

1. Open the toy graph and reference it by the variable graph.

2. Create a graph traversal source from the graph using the standard, OLTP traversal engine. This object should be created once and then re-used.

3. Spawn a traversal off the traversal source that determines the names of the people that the marko-vertex know.



The Name of The People That Marko Knows

Or, if the marko-vertex is already realized with a direct reference pointer (i.e. a variable), then the traversal can be spawned off that vertex.

**Credit: https://www.apache.org/**