Cadi-Ayyad University Marrakesh
Faculty of Sciences Semlalia
Computer Science Department

END OF STUDY PROJECT REPORT

To obtain the Bachelor of Fundamental Studies Diploma in:

Mathematical Sciences and Computer Science

Implementation of Kerberos Authentication System for Securing a Hadoop Ecosystem

Authors:

Abderrahmane Graoui

Haytam Ragueb

Supervised by:
Prof. Mariya Ouaissa

Defended on 07/01/2025 before the jury:

Prof. Mariya Ouaissa	Prof. in FS Semlalia
Prof. Soukaina Mjahed	Prof. in FS Semalalia

Academic Year: 2024/2025

Dedications

To our parents who supported and encouraged us throughout our studies.

To our brothers and sisters who shared with us every emotional and loving moment of our lives.

To our supervisors and teachers who guide us throughout our journey.

Acknowledgements

At the end of this project, we would like to express our deepest gratitude to our supervisor, Professor Mariya Ouaissa. She guided us throughout our project with her advice and the many guidelines she provided.

We would also like to thank all our professors for providing us with the necessary background throughout this academic year.

We also thank the members of the jury for agreeing to evaluate our work.

We especially thank our families, who are always our support.

Abstract

With the rapid growth of big data, Apache Hadoop has become a widely adopted framework for distributed data storage and processing. However, its default security model is insufficient for environments that handle sensitive or critical data, as it lacks strong authentication and access control mechanisms. This project aims to enhance the security of the Hadoop ecosystem by integrating the Kerberos authentication protocol, a trusted solution for network authentication based on secret-key cryptography.

The work involves setting up a secure Hadoop environment where user and service identities are authenticated through a centralized Key Distribution Center (KDC). The implementation process covers the installation and configuration of Kerberos, the modification of Hadoop services to support secure authentication, and the resolution of common integration issues. The project demonstrates how Kerberos strengthens Hadoop's security posture by preventing unauthorized access, impersonation, and service abuse.

Ultimately, this implementation provides a practical and scalable approach to securing big data platforms, highlighting the importance of strong identity management in modern distributed systems.

Keywords: Big Data, Authentication, Hadoop, Kerberos.

Table of Contents

List of Figures

List of Tables

General Introduction

In the era of big data, organizations rely increasingly on distributed processing frameworks such as Apache Hadoop to manage and analyze vast volumes of data. While Hadoop offers high scalability and fault tolerance, its default configuration lacks robust security mechanisms, leaving the system vulnerable to unauthorized access and data breaches. As data sensitivity and privacy concerns continue to grow, securing the Hadoop ecosystem has become a critical requirement.

One of the fundamental components of a secure Hadoop environment is a reliable authentication mechanism. Kerberos, a network authentication protocol developed by MIT, provides a strong solution by using secret-key cryptography and trusted third-party verification. Integrating Kerberos with Hadoop ensures that only authenticated users and services can access the cluster, thereby prevent impersonation and enhance the overall security posture of the system.

This report presents the implementation of the Kerberos authentication system within a Hadoop ecosystem. It details the configuration process, challenges encountered, and the resulting improvements in securing user and service identities. The objective is to demonstrate how Kerberos can be effectively deployed to enforce strong authentication in a big data environment, thus contributing to the broader goal of data security and governance.

1. General Context of the Project

1.1. Introduction

In today's data-driven world, Hadoop is widely used for managing and processing large volumes of data across distributed systems. However, its default deployment doesn't enforce secure user verification anyone with access to the system could potentially perform unauthorized actions. To address this, we designed and implemented an authentication layer that ensures only verified users and services can access Hadoop resources. We leveraged Kerberos, a strong, ticket-based authentication protocol, and optionally integrated LDAP for directory-based identity management.

1.2. Presentation of project

This project focuses on enhancing the security of big data infrastructures by implementing a strong authentication mechanism within the Hadoop ecosystem. Hadoop, while powerful in processing large-scale distributed data, lacks secure default configurations, making it susceptible to unauthorized access and identity spoofing. To address this vulnerability, the project proposes the integration of the Kerberos authentication protocol, which ensures that only verified users and services can interact with Hadoop components. Through the configuration and deployment of Kerberos, this work aims to establish a secure communication environment that strengthens the confidentiality and integrity of data operations across the cluster. The project also includes practical implementation steps, performance evaluation, and a discussion on the security benefits achieved through this integration.

1.3. Project issues

In other hand, with this big amount of data using Hadoop, it will appear some security challenges arise related to authentication and access control:

- **Large attack surface & many identities:** Big Data systems process huge volumes of sensitive data across distributed nodes. This requires strong control over both human and machine identities (M2M) to prevent unauthorized access

- **Weak or unmanaged credentials:** Traditional reliance on static passwords, SSH keys, or simple tokens leads to password fatigue, inconsistent key rotation, and mismanagement (keys stored in scripts or spreadsheets).

- **Regulatory & compliance risks:** Weak authentication undermines data confidentiality, integrity, and compliance with regulations (e.g., GDPR, HIPAA, PCI DSS), especially with high-value analytics on personal or financial data.

- **Complexity of distributed identity use:** Authentication systems must handle diverse access flows: users, jobs, services, machines, and automated analytics tools — each requiring reliable identity validation

1.4. Project objective

That will be leading us to made solution for it, step by step using this plan to get our objective:

- **Centralizes identity management:** Manage both human and machine identities via Kerberos, LDAP/AD, or certificate-based schemes.

- **Implements strong, auditable credentials:** Enforce key/keytab rotation, MFA, avoid hard-coded credentials, and eliminate random SSH keys or password files.

- **Supports distributed authentication flows:** Enable secure authentication for users, jobs, services, and inter-node communication (RPC, data transfer) across the cluster.

- **Enhances compliance & auditing:** Maintain logs of authentication events, enforce least privilege principles, and support policy enforcement (RBAC, ABAC), threads which align with regulatory requirements

- **Ensures high availability & scalability:** Operate reliably in a distributed Hadoop or Big Data environment, with automated provisioning

1.5. Project planification

The GANTT chart is a tool used in scheduling and project management to visualize the various tasks that make up a project over time. It is a representation of a connected, valued, and directed graph that graphically displays the project's progress.

The following diagram represents the GANTT chart, which illustrates the management and progress of our project.

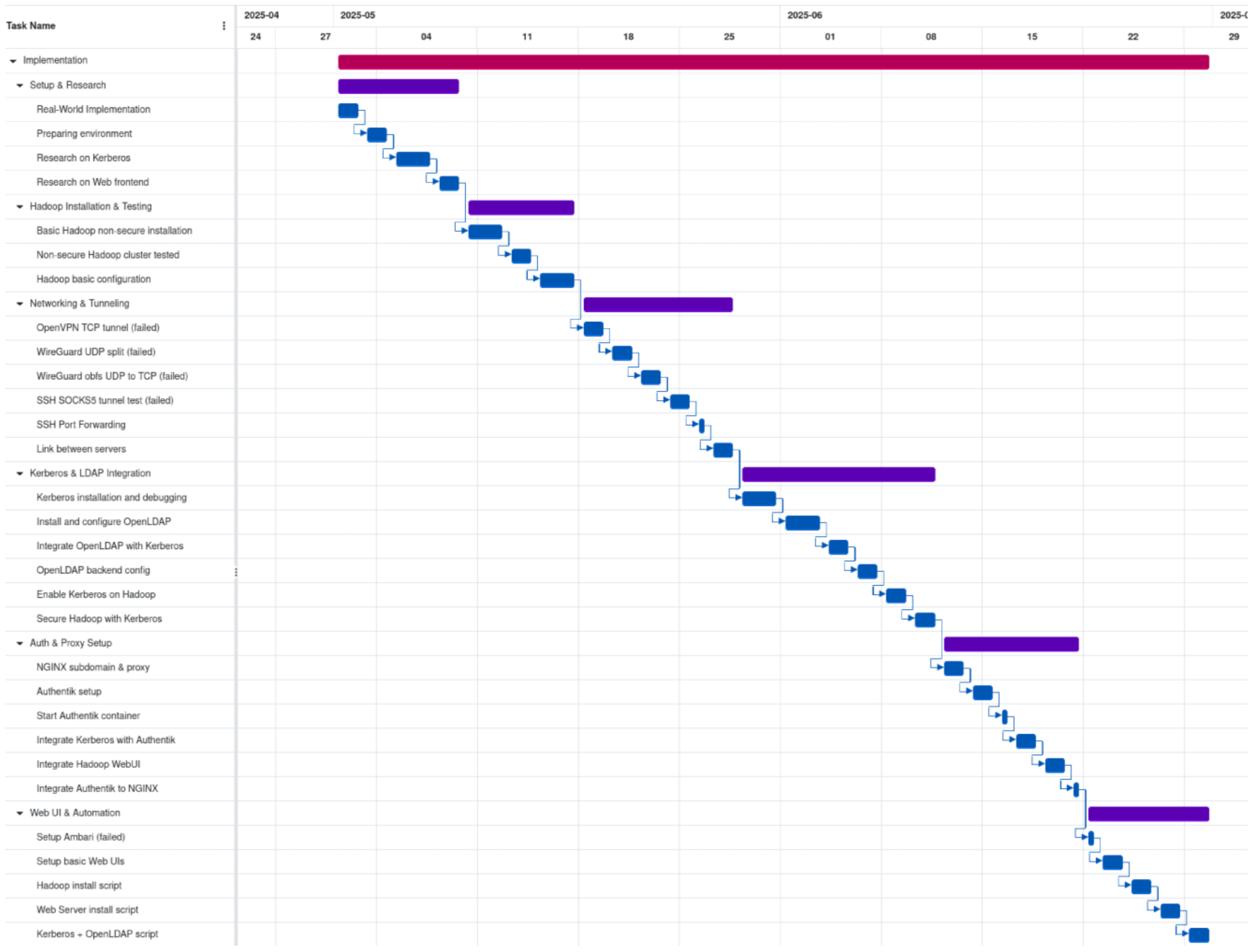


Figure 1. Implementation Gantt Diagram

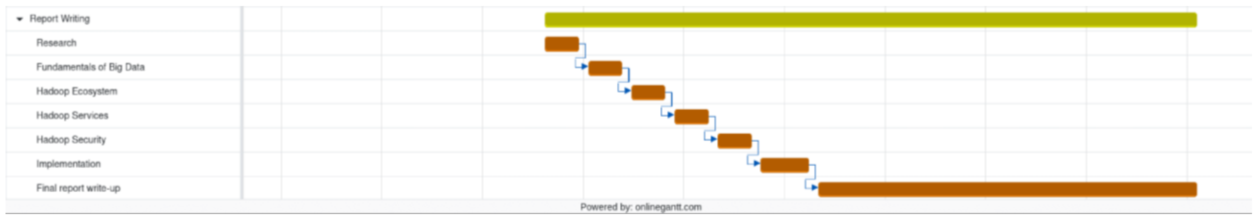


Figure 2. Report Gantt Diagram

1.6. Conclusion

In the summary, by addressing credential management, identity governance, and secure, auditable access paths, this project aims to eliminate weak authentication practices, prevent unauthorized data access, and align Big Data infrastructure with best practices and regulatory standards.

2. Basic Concepts and State of the Art

2.1. Introduction

This chapter provides a comprehensive overview of Big Data by outlining its fundamental concepts, technological ecosystem and major application domains. Furthermore, it highlights the state of the art in Big Data security, surveying existing approaches and open challenges in ensuring data confidentiality, integrity, and availability in large-scale environments.

2.2. Understanding Big Data

With the rapid growth of Internet users, there is an exponential growth in the data being generated. The data is generated from millions of messages we send and communicate via WhatsApp, Facebook, or Twitter, from the trillions of photos taken, and hours of videos getting uploaded in YouTube every single minute. According to a recent survey 2.5 quintillion (2.500.000.000.000.000.000, or 2.5 × 10 18) bytes of data are generated every day. This enormous amount of data generated is referred to as "big data." Big data does not only mean that the data sets are too large, but it is also a blanket term for the data that are too large, complex in nature, which may be structured or unstructured, and arriving at high velocity as well. Of the data available today, 80 percent has been generated in the last few years. The growth of big data is fueled by the fact that more data are generated on every corner of the world that needs to be captured.

Capturing this massive data gives only meager value unless this IT value is transformed into business value. Managing the data and analyzing them have always been beneficial to the organizations; on the other hand, converting these data into valuable business insights has always been the greatest challenge. Data scientists were struggling to find pragmatic techniques to analyze the captured data. The data must be managed at appropriate speed and time to derive valuable insight from it. These data are so complex that it became difficult to process it using traditional database management systems, which triggered the evolution of the big data era. Additionally, there were constraints on the amount of data that traditional databases could handle. With the increase in the size of data either there was a decrease in performance and increase in latency or it was expensive to add additional memory units. All these limitations have been over- come with the evolution of big data technologies that lets us capture, store, process, and analyze the data in a distributed environment. Examples of Big data technologies are Hadoop, a framework for all big data process, Hadoop Distributed File System (HDFS) for distributed cluster storage, and MapReduce for processing

2.3. Evolution of Big Data

The first documentary appearance of big data was in a paper in 1997 by NASA scientists narrating the problems faced in visualizing large data sets, which were a captivating challenge for the data scientists. The data sets were large enough, taxing more memory resources. This problem is termed big data. Big data, the broader concept, was first put forward by a noted consultancy: McKinsey. The three dimensions of big data, namely, volume, velocity, and variety, were defined by analyst Doug Laney. The processing life cycle of big data can be categorized into acquisition, preprocessing, storage and management, privacy and security, analyzing, and visualization. The broader term big data encompasses everything that includes web data, such as click stream data, health data of patients, genomic data from biologic research, and so forth. Figure 2 shows the evolution of big data. The growth of the data over the years is massive. It was just 600 MB in the 1950s but has grown by 2010 up to 100 peta- bytes, which is equal to 100.000.000.000 MB.

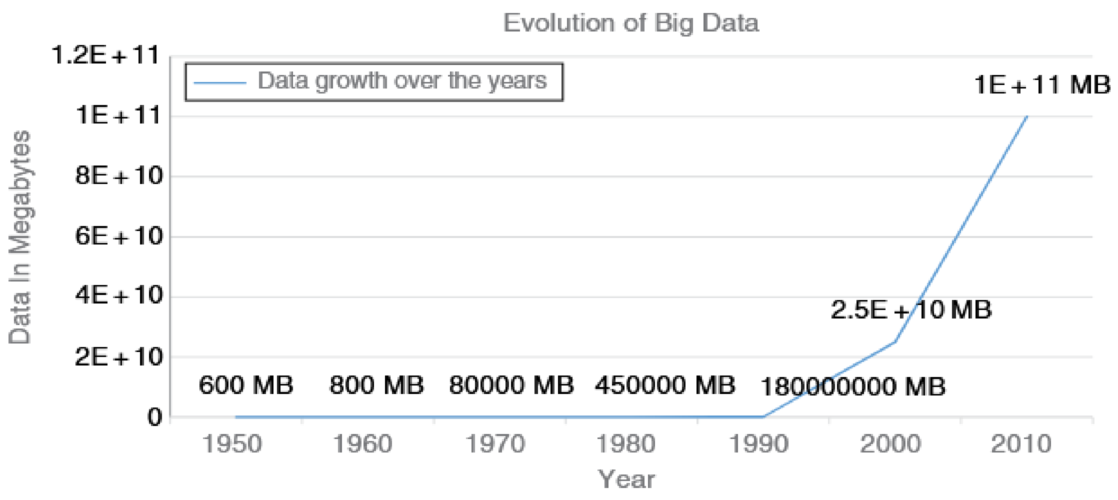


Figure 3. Evolution of Big Data

2.4. Failure of traditional database in handling Big Data

The Relational Database Management Systems (RDBMS) was the most prevalent data storage medium until recently to store the data generated by the organizations. Many vendors provide database systems. These RDBMS were devised to store the data that were beyond the storage capacity of a single computer. The inception of a new technology is always due to limitations in the older technologies and the necessity to overcome them. Below are the limitations of traditional database in handling big data.

- Exponential increase in data volume, which scales in terabytes and petabytes, has turned out to become a challenge to the RDBMS in handling such a massive volume of data.
- To address this issue, the increased the number of processors and added more memory units, which in turn increased the cost.

- Almost 80% of the data fetched were of semi-structured and unstructured for- mat, which RDBMS could not deal with.
- RDBMS could not capture the data coming in at high velocity.

2.5. 3Vs of Big Data

Big data is distinguished by its exceptional characteristics with various dimensions. Figure 3 illustrates various dimensions of big data. The first of its dimensions is the size of the data. Data size grows partially because the cluster storage with commodity hardware has made it cost effective. Commodity hardware is a low cost, low performance, and low specification functional hardware with no distinctive features. This is referred by the term "volume" in big data technology. The second dimension is the variety, which describes its heterogeneity to accept all the data types, be it structured, unstructured, or a mix of both. The third dimension is velocity, which relates to the rate at which the data is generated and being processed to derive the desired value out of the raw unprocessed data.

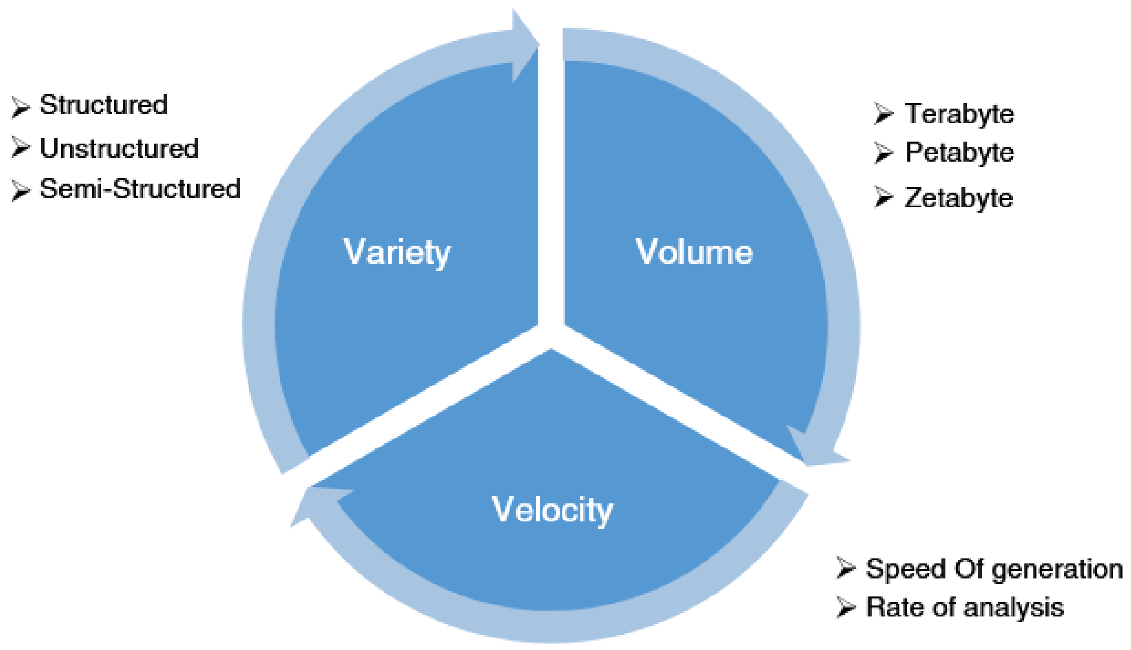


Figure 4. 3V's of Big Data

2.5.1. Volume

Data generated and processed by big data are continuously growing at an ever-increasing pace. Volume grows exponentially since business enterprises are continuously capturing the data to make better and bigger business solutions. Big data volume measures from terabytes to zettabytes (1024 GB = 1 terabyte; 1024 TB = 1 petabyte; 1024 PB = 1 exabyte; 1024 EB = 1 zettabyte; 1024 ZB = 1 yottabyte). Capturing this massive data is cited as an extraordinary opportunity to achieve finer customer service and better business advantage.

This ever-increasing data volume demands highly scalable and reliable storage. The major sources contributing to this tremendous growth in the volume are social media, point of sale (POS) transactions, online banking, GPS sensors, and sensors in vehicles. Facebook generates approximately 500 terabytes of data per day. Every time a link on a website is clicked, an item is purchased online, a video is uploaded in YouTube, data are generated.

2.5.2. **Velocity**

With the dramatic increase in the volume of data, the speed at which the data is generated also surged up. The term "velocity" not only refers to the speed at which data are generated, it also refers to the rate at which data is processed and Volume Velocity Variety Terabyte Petabyte Zettabyte Speed of Generation Rate of analysis Structured Unstructured Semi-Structured

In the big data era, a massive amount of data is generated at high velocity, and sometimes these data arrive so fast that it becomes difficult to capture them, and yet the data needs to be analyzed Figure 4. illustrates the data generated with high velocity in 60 seconds: 3.3 million Facebook posts, 450 thousand tweets, 400 hours of video upload, and 3.1 million Google searches

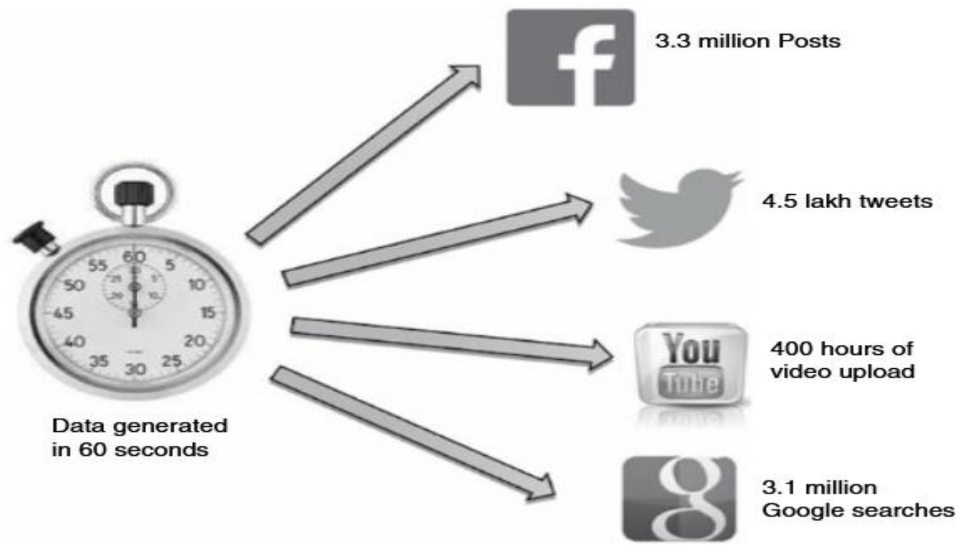

3.3 million Posts
4.5 lakh tweets
400 hours of video upload
3.1 million Google searches
Data generated in 60 seconds

Figure 5. High velocity of data generated in every 60 second

2.5.3. **Variety**

Variety refers to the format of data supported by big data. Data arrives in structured, semi-structured, and unstructured format. Structured data refers to the data processed by traditional database management systems where the data are organized in tables, such as employee details, bank customer details. Semi-structured data is a combination of structured and unstructured data, such as XML. XML data is semi-structured since it does not fit the formal data model (table) associated with traditional database; rather, it contains tags to organize fields within the data. Unstructured data refers to data with no definite structure, such as e-mail messages, photos, and web pages. The data that arrive from

Facebook, Twitter feeds, sensors of vehicles, and black boxes of airplanes are all unstructured, which the traditional database cannot process, and here is when big data comes into the picture. Figure 5 represents the different data types.

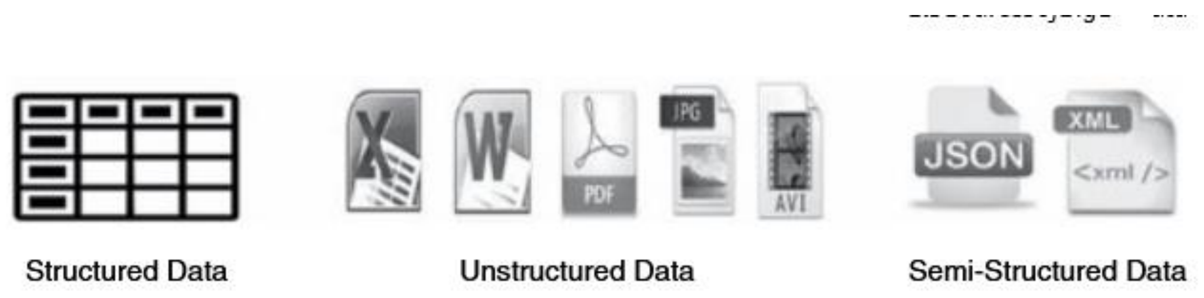


Figure 6. Data variety

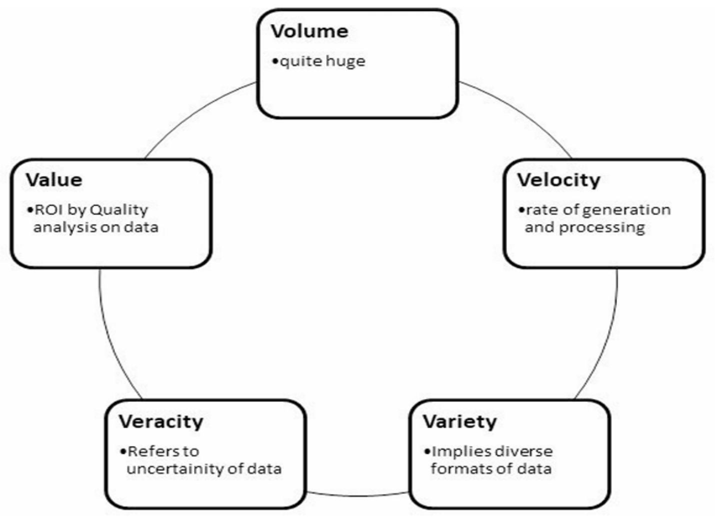

Volume
•quite huge
Velocity
•rate of generation and processing
Variety
•Implies diverse formats of data
Veracity
•Refers to uncertainity of data
Value
•ROI by Quality analysis on data

Figure 6. Big Data 5V's

2.6. The challenges of Big Data

2.6.1. The technical challenges of Big Data

Big Data was originally a primarily technical subject. The priority was to understand why, and especially how, to process massive amounts of data. Today, the technical challenges remain very much present, refocused around three key points that are attracting attention:

- **Ensuring data security:** The growth of internet usage has led to an explosion in the volume of data collected. Advances in networks and connected objects, the widespread adoption of 5G... the reasons are multiple. Consequently, this increase is leading to a rethinking of data security processes. Much of the data collected relates to personal information and privacy... In France, the GDPR has passed this test, leading to a very high level of requirements at the national level. Finding appropriate solutions to prevent information theft is an ongoing battle, to which public opinion is very sensitive.

- **Collect quality data:** For obvious cost and regulatory reasons, data should not be stored without an operational objective. The data collected must be reliable and reusable for self-serving purposes. It facilitates a precise definition of the expectations or behaviors of your marketing targets. Ensuring data quality is a fundamental step in pursuing business objectives. Data analysis is only successful when the information collected is viable and within a strict legislative framework.
- **Optimizing data processing for action:** the goal is to transform data into actionable levers to improve the company's service quality. The challenge is even greater when the information must be processed in real time. Finding technical solutions that ensure speed and accuracy is no easy task. This is a real challenge for internal teams.

2.6.2. The strategic challenges of Big Data

Data collection and storage must be extended to operational decision-making. The human and material investment must be profitable, with a concrete impact on strategic directions.

Data analysis is a valuable decision-making tool. It offers answers to certain problems, bringing quantifiable elements to the table. Choice of target, catchment area, language elements... decisions must be validated by statistics or numerical data. Subjectivity disappears in favor of a choice supported by macro trends. The element of uncertainty is drastically reduced.

Organizational issues are also influenced by the contribution of Big Data. All the data collected supports developments related to the company's digital transformation. By analyzing the figures, processes can be adjusted, lead times shortened, or points in the supply chain to be strengthened can be targeted.

2.6.3. The societal challenges of Big Data

Massive data collection also creates privacy issues. In France, this is now governed by legislation known as the GDPR (General Data Protection Regulation). We often talk about GAFAM to evoke the risks associated with the use of personal data, but all companies must be concerned, even the smallest.

Compliance with GDPR regulations is an obligation that everyone must adhere to. The notion of ethics plays a key role here. If common sense is not enough, regulations come to the fore. The legality of data collection and use is scrutinized. Woe betides companies that are too greedy in their practices, because the legal risk is high. Not to mention the impact on brand image... Data protection is not to be trifled with, especially in Europe.

2.7. Big Data technologies

2.7.1. Airflow

Airflow is a workflow management platform for scheduling and running complex data pipelines in big data systems. It enables data engineers and other users to ensure each task in a workflow is executed in the designated order and has access to the required system resources. Workflows are created in the Python programming language, and Airflow can be used for building machine learning models, transferring data and various other purposes.

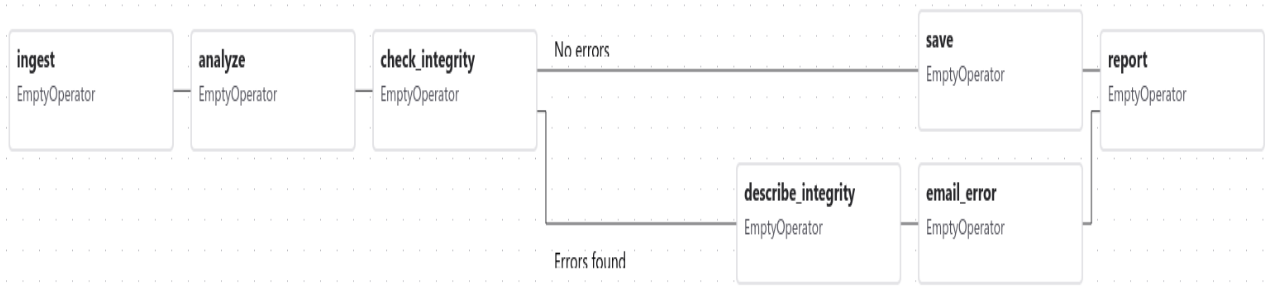


Figure 7. Airflow architecture

2.7.2. Delta Lake

Databricks Inc., a software vendor founded by the creators of the Spark processing engine, developed Delta Lake and then open sourced the Spark-based technology in 2019 through the Linux Foundation. Delta Lake is a table storage layer that can be used to build a data lakehouse architecture combining elements of data lakes and data warehouses for both streaming and batch processing applications.

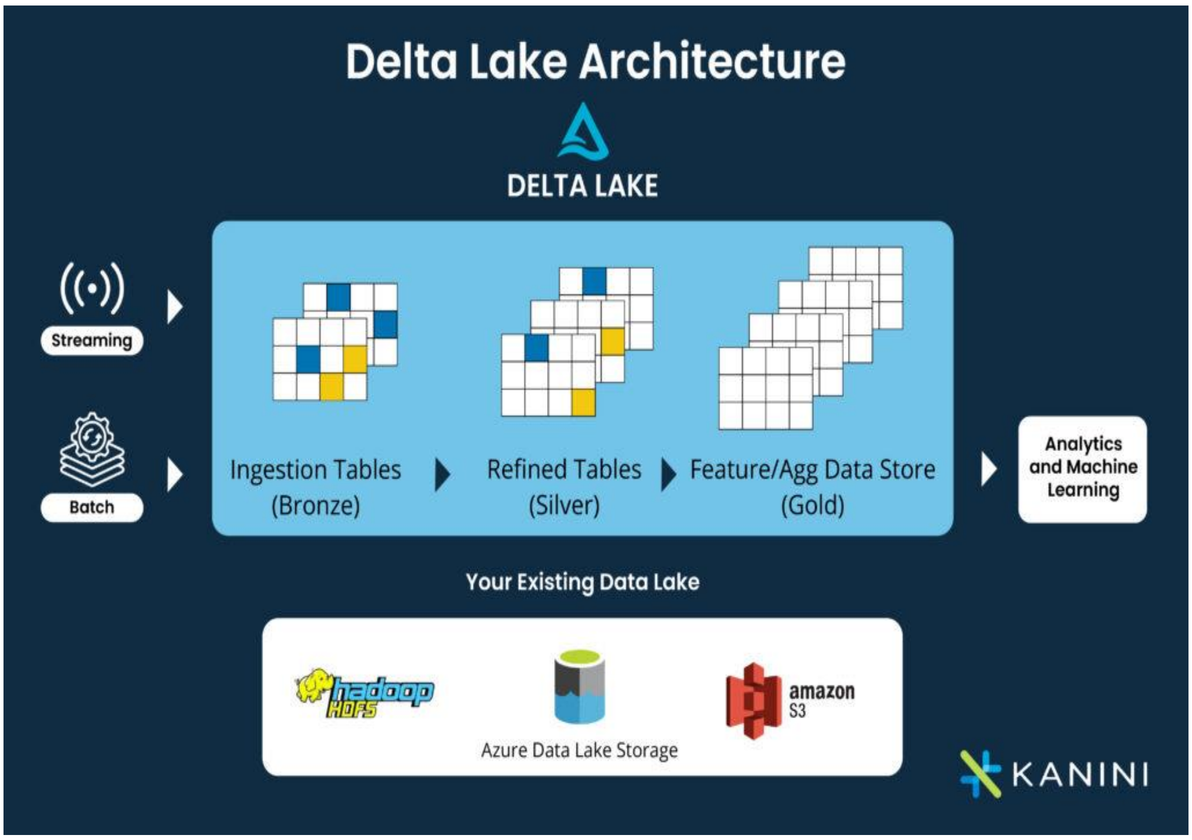


Figure 8. Delta lake architecture

It's designed to sit on top of a data lake and create a single home for structured, semi structured and unstructured data, eliminating data silos that can stymie big data applications. Delta Lake supports ACID transactions that adhere to the principles of atomicity, consistency, isolation and durability. It also includes a liquid clustering capability to optimize how data is stored based on query patterns, as well as the following features:

- The ability to store data in an open Apache Parquet format.
- Uniform Format, or UniForm for short, a function that enables Delta Lake tables to be read in Iceberg and Hudi, two other Parquet-based table formats.
- Compatibility with Spark APIs.

2.7.3. **Drill**

The Apache Drill website describes it as a low latency distributed query engine best suited for workloads that involve large sets of complex data with different types of records and fields. Drill can scale across thousands of cluster nodes and query petabytes of data using SQL and standard connectivity APIs. It can handle a combination of structured, semi structured and nested data, the latter including things such as JSON and Parquet files.

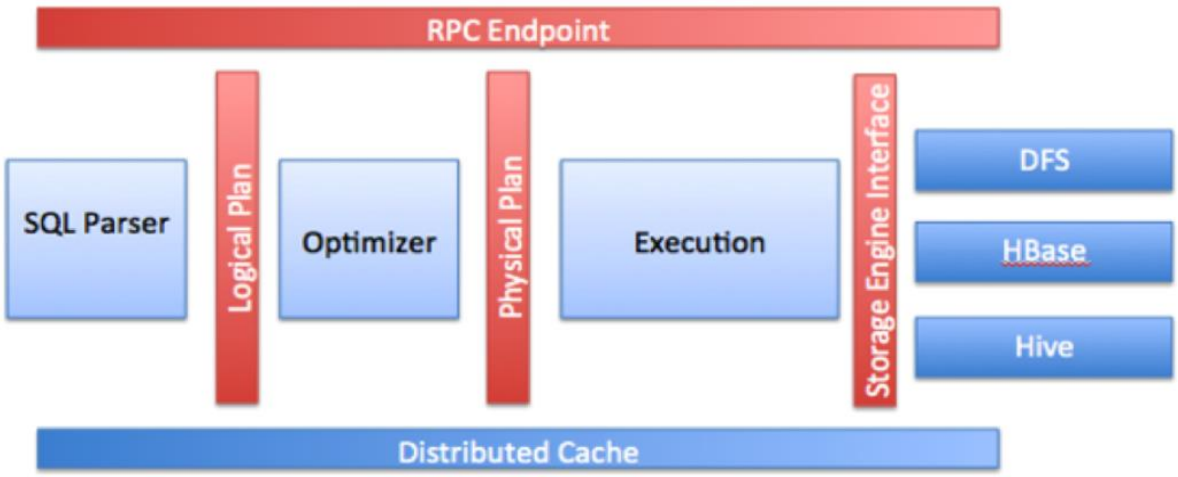


Figure 9. Drill architecture

Drill layers on top of multiple data sources, enabling users to query a wide range of data in different formats. That includes Hadoop sequence files and event logs, NoSQL databases, cloud object storage and various file types. Multiple files can be stored in a directory and queried as if they were a single entity. The software can also do the following:

- Access most relational databases through a plugin.
- Work with commonly used BI tools, such as Tableau and Qlik.
- Run in any distributed cluster environment, although it requires Apache's ZooKeeper software to maintain information about clusters.

2.7.4. Druid

Druid is a real-time analytics database that delivers low latency for queries, high concurrency, multi-tenant capabilities and instant visibility into streaming data. Multiple end users can query the data stored in Druid at the same time with no impact on performance, according to its proponents.

Written in Java and created in 2011, Druid became an Apache technology in 2018. It's generally considered a high-performance alternative to traditional data warehouses that's best suited to event-driven data. Like a data warehouse, it uses column-oriented storage and can load files in batch mode. But it also incorporates features from search systems and time series databases, including the following:

- Native inverted search indexes to speed up searches and data filtering.
- Time-based data partitioning and querying.
- Flexible schemas with native support for semi structured and nested data.

2.7.5. Flink

Another Apache open-source technology, Flink is a stream processing framework for distributed, high-performing and always-available applications. It supports stateful computations over both bounded and unbounded data streams and can be used for batch, graph and iterative processing.

One of the main benefits touted by Flink's proponents is its speed: It can process millions of events in real time for low latency and high throughput. Other potential use cases include data pipelines, and both streaming and batch analytics. Flink, which is designed to run in all common cluster environments, also includes the following features:

- In-memory computations with the ability to access disk storage when needed.
- Three layers of APIs for creating different types of applications.
- A set of libraries for complex event processing, machine learning and other common big data use cases.

2.7.6. Hadoop

A distributed framework for storing data and running applications on clusters of commodity hardware, Hadoop was developed as a pioneering big data technology to help handle the growing volumes of structured, unstructured and semi structured data. First released in 2006, it was almost synonymous with big data early on; it has since been partially eclipsed by other technologies but is still used by many organizations. Hadoop has four primary components:

- The Hadoop Distributed File System (HDFS), which splits data into blocks for storage on the nodes in a cluster, uses replication methods to prevent data loss and manages access to the data.

- YARN, short for Yet Another Resource Negotiator, which schedules jobs to run on cluster nodes and allocates system resources to them.
- Hadoop MapReduce, a built-in batch processing engine that splits up large computations and runs them on different nodes for speed and load balancing.

Hadoop Common, a shared set of utilities and libraries. Initially, Hadoop was limited to running MapReduce batch applications. The addition of YARN in 2013 opened it up to other processing engines and use cases, but the framework is still closely associated with MapReduce. The broader Apache Hadoop ecosystem also includes various big data tools and additional frameworks for processing, managing and analyzing big data.

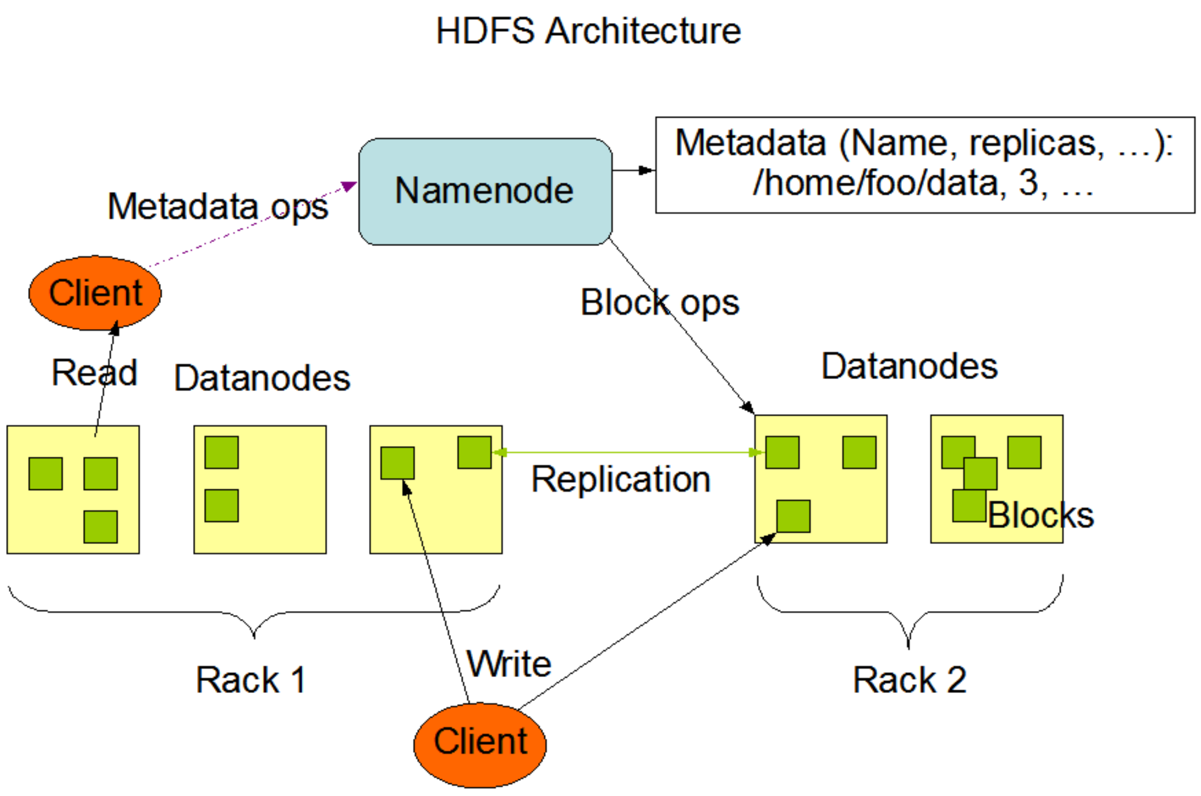


Figure 10. Hadoop architecture

2.7.7. **Hive**

Hive is SQL-based data warehouse infrastructure software for reading, writing and managing large data sets in distributed storage environments. It was created by Facebook but then open sourced to Apache, which continues to develop and maintain the technology.

Hive runs on top of Hadoop and is used to process structured data; more specifically, it's used for data summarization and analysis, as well as for querying large amounts of data. Although it can't be used for online transaction processing or real-time updates, Hive is described by its developers as scalable, fast and flexible. A central metadata repository can be used as a building block for data lakes, and Hive supports ACID, low-latency analytical processing and the ability to read and write Iceberg tables.

Other key features include the following:

- Standard SQL functionality for data querying and analytics.
- A built-in mechanism to help users impose structure on different data formats.
- Access to HDFS files and ones stored in other systems, such as the Apache HBase database.

2.7.8. **HPCC Systems**

HPCC Systems is a big data processing platform developed by LexisNexis Risk Solutions and then open sourced in 2011. While still primarily overseen by the company, it's freely available to download under the Apache 2.0 license. True to its full name -- High-Performance Computing Cluster Systems -- the technology is, at its core, a cluster of computers built from commodity hardware.

A production-ready data lake environment that's designed to enable fast data engineering for analytics applications, HPCC Systems includes three main components:

- Thor, a data refinery engine used to cleanse, merge and transform data for use in queries.
- Roxie, a data delivery engine used to serve up prepared data from the refinery.
- Enterprise Control Language, or ECL, a programming language for developing applications.

In its latest releases, HPCC Systems is a cloud-native platform that can be run in Docker containers on Kubernetes in both the AWS and Microsoft Azure clouds. Deployments of the original bare-metal platform are also still supported, though.

2.7.9. **Hudi**

Hudi (pronounced hoodie) is short for Hadoop Upserts, Deletes and Incrementals. Another open-source technology maintained by Apache, it's used to manage the ingestion and storage of large analytics data sets on Hadoop-compatible file systems, including HDFS and cloud object storage services. Hudi combines an open table format with a software stack built to underpin data lakes and data lakehouses.

First developed by Uber, Hudi is designed to provide efficient and low-latency data ingestion and data preparation capabilities. The technology also supports ACID transactions, multimodal indexing to boost query performance and a time travel feature for analyzing historical data. Moreover, it includes a data management framework that organizations can use to do the following:

- Simplify incremental data processing and data pipeline development.

- Improve data quality in big data systems.
- Manage the lifecycle of data sets.

2.7.10. Iceberg

Iceberg is an open table format used to manage data in data lakes, which it does partly by tracking individual data files in tables rather than by tracking directories. Created by Netflix for use with the company's petabyte-sized tables, Iceberg is now an Apache project. According to the project's website, Iceberg typically "is used in production where a single table can contain tens of petabytes of data.

Designed to improve on the standard layouts that exist within tools such as Hive, Presto, Spark and Trino, the Iceberg table format has functions similar to SQL tables in relational databases. It also accommodates multiple query engines operating on the same data set, including the ones listed above. Other notable features include the following:

- Schema evolution for modifying tables without having to rewrite or migrate data.
- Hidden partitioning of data that avoids the need for users to maintain partitions.
- A time travel capability that supports reproducible queries using the same table snapshot.

2.7.11. Kafka

Kafka is a distributed event streaming platform that, according to Apache, is used by more than 80% of Fortune 100 companies and thousands of other organizations for high-performance data pipelines, streaming analytics, data integration and mission-critical applications. In simpler terms, Kafka is a framework for storing, reading and analyzing streaming data.

The technology decouples data streams and systems, holding the data streams so they can then be used elsewhere. It runs in a distributed environment and uses a high-performance TCP network protocol to communicate with systems and applications. Kafka, which was created by LinkedIn before being passed on to Apache in 2011, is designed to handle petabytes of data and trillions of messages per day. The following are some of the key components in Kafka:

- A set of five core APIs for Java and the Scala programming language.
- Fault tolerance for both servers and clients in Kafka clusters.
- Elastic scalability to up to 1,000 brokers, or storage servers, per cluster.

2.7.12. Kylin

Kylin is a distributed data warehouse and analytics platform for big data. It provides an online analytical processing (OLAP) engine designed to support extremely large data sets. Because Kylin is built on top of other Apache technologies -- including Delta Lake, Parquet and the Gluten compute engine -- it can easily scale to handle those large data loads, according to its backers. The platform supports Hive, Kafka, Iceberg and other external data sources as well as an internal table added in December 2024.

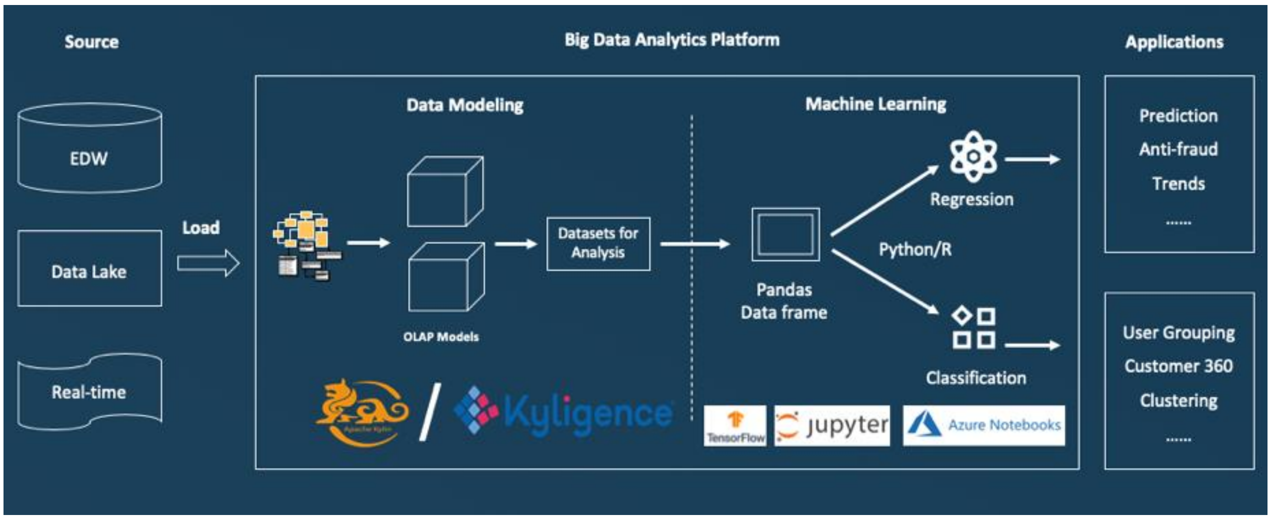


Figure 11. Kylin architecture

How Apache Kylin works with Big Data. In addition, Kylin provides an ANSI SQL interface for multidimensional analysis of big data and integrates with Tableau, Microsoft Power BI and other BI tools. Kylin was initially developed by eBay, which contributed it as an open-source technology in 2014; it became a top-level project within Apache the following year. Other features it offers include the following:

- Pre-calculation of multidimensional OLAP cubes to accelerate analytics.
- Job management and monitoring functions.
- Support for building customized UIs on top of the Kylin core.

2.7.13.Pinot

Pinot is a real-time distributed OLAP data store built to support low latency querying by analytics users. Its design enables horizontal scaling to deliver that low latency even with large data sets and high throughput. To provide the promised performance, Pinot stores data in a columnar format and uses various indexing techniques to filter, aggregate and group data. In addition, configuration changes can be done dynamically without affecting query performance or data availability.

According to Apache, Pinot can handle trillions of records overall while ingesting millions of data events and processing thousands of queries per second. The system has a fault-tolerant architecture with no single point of failure and assumes all stored data is immutable, although it also works with mutable data. Started in 2013 as an internal project at LinkedIn, Pinot was open sourced in 2015 and became an Apache top-level project in 2021. The following features are also part of Pinot:

- Near-real-time data ingestion from streaming sources, plus batch ingestion from HDFS, Spark and cloud storage services.
- A SQL interface for interactive querying and a REST API for programming queries.
- Support for running machine learning algorithms against stored data sets for anomaly detection.

2.7.14.Presto

Formerly known as PrestoDB, this open-source SQL query engine can simultaneously handle fast queries and large data volumes in distributed data sets. Presto is optimized for low-latency interactive querying, and it scales to support analytics applications across multiple petabytes of data in data warehouses, data lakes and other repositories. To further boost performance and reliability, Presto's developers are converting the core execution engine from Java to a C++ version based on Velox, an open-source acceleration library. Presto C++ is available to use but still has limitations that need to be addressed.

Presto's development began at Facebook in 2012. When its creators left the company in 2018, the technology split into two branches: PrestoDB, which was still led by Facebook, and PrestoSQL, which the original developers launched. That continued until December 2020, when PrestoSQL was renamed Trino and PrestoDB reverted to the Presto name. The Presto open-source project is now overseen by the Presto Foundation, which was set up as part of the Linux Foundation in 2019. Presto also includes the following features:

- Connectors to 36 data sources, including Delta Lake, Druid, Hive, Hudi, Iceberg, Pinot and various databases.
- The ability to combine data from multiple sources in a single query.
- A web-based UI and a CLI for querying, plus support for the Apache Superset data exploration tool.

2.7.15.Samza

Samza is a distributed stream processing system that was built by LinkedIn and has been an open-source project managed by Apache since 2013. According to the project website, Samza enables users to build stateful applications for real-time processing of data from Kafka, HDFS and several other sources, with built-in integration to them.

The system can run on top of Hadoop YARN or Kubernetes and offers a standalone deployment option. The Samza site says it can handle "several terabytes" of state data, with low latency and high throughput for fast data analysis. Through a unified API, it can also use the same code written for data streaming jobs to run batch applications. Other features include the following:

- Both high- and low-level APIs for different use cases, plus a declarative SQL interface.
- The ability to run as an embedded library in Java and Scala applications.
- Fault-tolerant features designed to enable rapid recovery from system failures.

2.7.16.**Spark**

Apache Spark is an in-memory data processing and analytics engine that can run on clusters managed by Hadoop YARN, Mesos and Kubernetes or in a standalone mode. It enables large-scale data engineering and analytics for both batch and streaming applications, as well as machine learning and graph processing use cases. That's all supported by the following set of built-in modules and libraries:

- Spark SQL, for optimized processing of structured data via SQL queries.
- Spark Structured Streaming, a stream processing module.
- MLlib, a machine learning library that includes algorithms and related tools.
- GraphX, an API that adds support for graph applications.

Data can be accessed from various sources, including HDFS, flat files and both relational and NoSQL databases. Spark also supports various file formats and offers developers a diverse set of APIs. Spark's performance edge over traditional counterpart MapReduce made it the top choice for batch applications in many big data environments, but it also functions as a general-purpose analytics engine. First developed at the University of California, Berkeley, and now maintained by Apache, it can also process on disk when data sets are too large to fit into the available memory.

2.7.17.**Storm**

Another Apache open-source technology, Storm is a distributed real-time computation system that's designed to reliably process unbounded streams of data. According to the project website, it can be used for applications that include real-time analytics, continuous computation and machine learning on streaming data, as well as extract, transform and load jobs.

Storm clusters are akin to Hadoop ones, but applications continue to run on an ongoing basis unless they're stopped. The system is fault-tolerant and guarantees that data will be processed. In addition, the Apache Storm site says it can be used with any programming language, message queueing system and database.

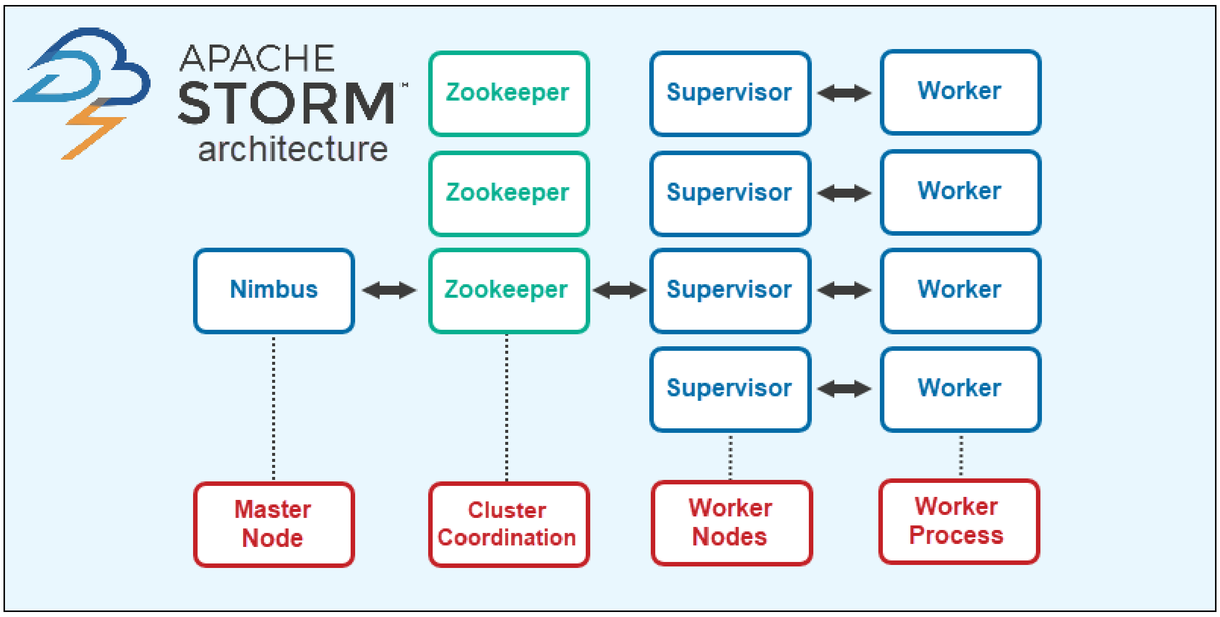


Figure 12. Apache Storm architecture

Storm also includes the following elements:

- A Storm SQL feature that enables SQL queries to be run against streaming data sets.
- Trident and Stream API, two other higher-level interfaces for processing in Storm.
- Use of the Apache ZooKeeper technology to coordinate clusters.

2.7.18. Trino

As mentioned above, Trino is one of the two branches of the Presto query engine. Like the current Presto, it's a distributed SQL engine for use in big data analytics applications. Trino supports low-latency analytics in exabyte-scale data lakes and large data warehouses, according to the Trino Software Foundation. That group, which oversees Trino's development, was originally formed in 2019 as the Presto Software Foundation; its name was also changed as part of the 2020 rebranding of PrestoSQL.

Trino enables users to query data regardless of where it's stored, with support for natively running queries in Hadoop and other data repositories. It includes a CLI and a plugin that lets users run queries in Grafana, an open-source data visualization and dashboard design tool. In addition, Trino works with Tableau, Power BI, Apache Superset, the R programming language, and various other BI and analytics tools.

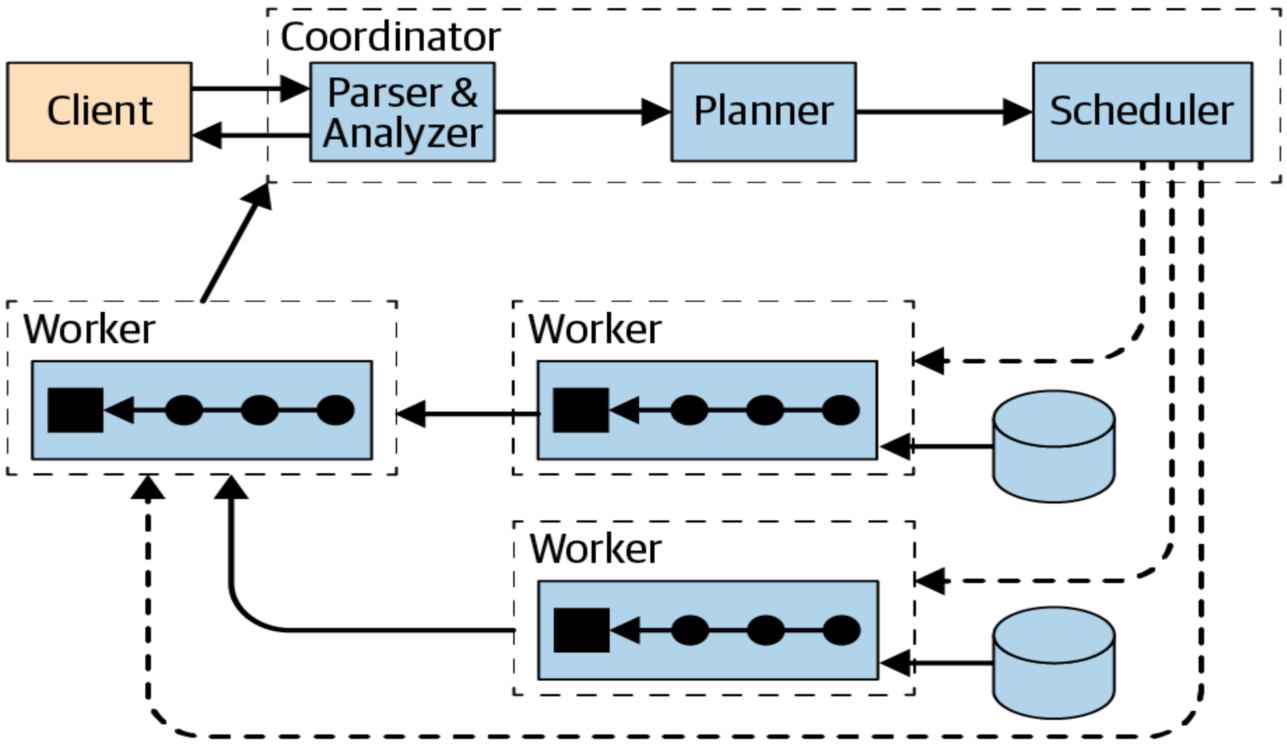


Figure 13. Trino architecture

As with Presto, Trino also is designed for the following:

- Both ad hoc interactive analytics and long-running batch queries.
- Queries that combine data from multiple systems through a federation feature.
- Built-in links to data sources through a set of 38 connectors.

Big data comes with specific terminology that can seem complex to the uninitiated. Here are some key terms explained:

- **Data Lake:** A centralized repository that stores all structured and unstructured data at any scale. Unlike traditional databases, a data lake can contain data in its raw form.
- **ETL (Extract, Transform, Load):** A process used to extract data from various sources, transform it according to specific needs (cleaning, formatting), and load it into a storage system.
- **Machine Learning:** A branch of artificial intelligence that allows systems to learn and improve automatically from experience without being explicitly programmed.
- **Cluster Computing:** Using multiple connected computers (a cluster) to work together as a single system, particularly useful for processing massive volumes of data.
- **MapReduce:** A programming model for processing large amounts of data using computer clusters. It divides a task into smaller subtasks (map) and then combines them to obtain the result (reduce).

2.8. Security in Big Data

Big Data security refers to any measures used to protect data against malicious activity during the storage, processing, and analysis of datasets that are too large and complex to be handled by traditional database applications. Big Data can come in a mix of structured formats (organized into rows and columns containing numbers, dates, etc) or unstructured (social media data, PDF files, emails, images, etc). Estimates show that up to 90 percent of Big Data is unstructured, though.

Big Data’s power is that it often contains hidden insights that can improve business processes, drive innovation, or reveal unknown market trends. Since workloads to analyze this information often combine sensitive customer or proprietary data along with third-party data sources, proper data security is vital. Reputational damage and hefty financial losses are two major consequences of leaks and breaches of Big Data.

There are really three key stages to consider when trying to secure Big Data:

- Securing the transit of data as it moves from source locations for storage or real-time ingestion, usually in the cloud
- Protecting data in the storage layers of a Big Data pipeline (e.g. Hadoop Distributed File System)
- Ensuring the confidentiality of output data such as reports and dashboards that contain intelligence gleaned from running the data through an analytics engine such as Apache Spark

The types of security threats in these environments include improper access controls, Distributed Denial of Service (DDoS) attacks, endpoints generating false or malicious data, or vulnerabilities in libraries, frameworks, and applications used during Big Data workloads.

2.8.1. The Challenges in securing Big Data

There are many challenges particular to Big Data security that emerge due to the architectural and environmental complexities involved. In a Big Data environment, you have an interplay of diverse hardware and technologies across a distributed computing environment. Some examples of challenges are:

- The use of open-source frameworks like Hadoop that weren’t originally designed with security in mind
- The reliance on distributed computing to process these large datasets means there are more systems on which something could go wrong
- Ensuring the validity and authenticity of logs or event data collected from endpoints
- Controlling insider access to data mining tools and monitoring for suspicious behavior
- Difficulty in running standard security audits
- Securing non-relational NoSQL databases

These challenges are additions to, rather than replacements for, the usual challenges involved in securing any type of data.

2.8.2. Authentication

2.8.2.1. Definition

Frameworks, libraries, software utilities, data ingestion, analytics tools, and custom applications — Big Data security starts at the code level. Harmful security errors in code can result in data leakage regardless of whether you've implemented the above well-established security best practices. So, if you're a developer or engineer tasked with working on your organization's Big Data pipeline, you need a solution that scans proprietary, custom, and open-source code rapidly and accurately for exposed API keys, tokens, credentials, and misconfigurations across your environment. By starting with a secure codebase, the challenge of Big Data security becomes a lot less daunting. Spectral provides data loss prevention through automated codebase scanning that covers the entire software development lifecycle. The tool works in minutes and can easily eliminate public blind spots across multiple data sources.

2.8.2.2. Authentication factors

Multifactor Authentication (MFA) methods involve using two or more distinct types of verification to prove your identity. This means you must provide credentials from different categories, like "something you know" (e.g., a password) and "something you have" (e.g., a code from an authenticator app or an SMS code). Examples include using a password combined with a fingerprint scan or a security key. Examples include scenarios like using a password combined with a fingerprint scan.

2.8.2.3. Types of authentication methods

Authentication factors are divided into three main categories: knowledge, possession, and inherence. Understanding and correctly implementing these diverse authentication factors ensures that access to sensitive information or systems remains controlled and secure, minimizing the risk of unauthorized access and data breaches.

2.8.2.3.1. Knowledge Factors

Knowledge factors involve information that the user knows, such as a password or a PIN. These are among the easiest to implement but are also susceptible to compromise if not backed by other factors.

2.8.2.3.2. Possession Factors

Possession factors rely on the user's possession of something, such as a smartphone or a security token. These factors typically work by sending a verification code to the device or using a dedicated app to approve login attempts, which adds a layer of physical security.

2.8.2.3.3. Inherence Factor

Inherence factors often deemed the most secure, involve something the user is, such as fingerprints, voice recognition, or facial recognition. These biometric methods are more difficult for unauthorized users to replicate.

2.8.3. Authorization

In addition to HDFS permissions various Hadoop services also support authorization. This service-level authorization (SLA) is used to control user or group access to various components that may consume resources like memory, CPU, etc. The configuration of these SLAs for various other Hadoop components like YARN, MapReduce, ZooKeeper, Hbase, Hive, etc. is beyond the scope of this blog but is a necessary and pertinent part of the overall Hadoop security model.

Additionally, it has become nigh impossible for Hadoop administrators to enforce a common security policy across all components. There has been a perceived need for a central command-and-control mechanism to implement common security policies across all the Hadoop components in a uniform manner. This has given rise to new Apache projects like Sentry and Ranger. Although they vary in the particulars, both Sentry and Ranger share an aim to provide a central location for managing all security related tasks including role based or attribute-based access control, fine grained authorization, authentication, auditing and data protection. The reader is encouraged to review this topic in detail by accessing the resources at the end.

2.8.4. Audit

Auditing is a mechanism by which a system can keep track of the activities of the users and services interacting within the system. Most Hadoop components can be configured to passively log all activity. It is recommended that key services like HDFS, MapReduce, YARN, Hive, HBase, Sentry or Ranger be configured to enable audit logging. For example, HDFS can be configured to log all general user as well as all service-level authorization activity. Additionally, Ranger and non-Hadoop tools can be used to actively monitor the system and provide alerts for targeted events. Auditing is also essential to achieve and maintain compliance in enterprise systems that contain sensitive data like medical records or financial information.

In a distributed system like Hadoop various components generate audit logs on each server in the cluster. It is recommended that log aggregation tools like syslog be used to collect all relevant information into a central place and a historical view be maintained for a period of time. This can aid in compliance as well as during forensic analysis after a breach is detected.

2.8.4.1. Data encryption

Scalable encryption for data at rest and data in transit is critical to implement across a Big Data pipeline. Scalability is the key point here because you need to encrypt data across analytics toolsets and their output in addition to storage formats like NoSQL. The power of encryption is that even if a threat actor manages to intercept data packets or access sensitive files, a well-implemented encryption process makes the data unreadable.

2.8.4.2. User access control

Getting access control right provides robust protection against a range of Big Data security issues, such as insider threats and excess privileges. Role-based access can help to control access over the many layers of Big Data pipelines. For example, data analysts should have access to analytics tools like R, but they probably shouldn't get access to tools used by Big Data developers, such as ETL software. The principle of least privileges is a good reference point for access control by limiting access to only the tools and data that are strictly necessary to perform a user's tasks.

2.8.4.3. Cloud security monitoring

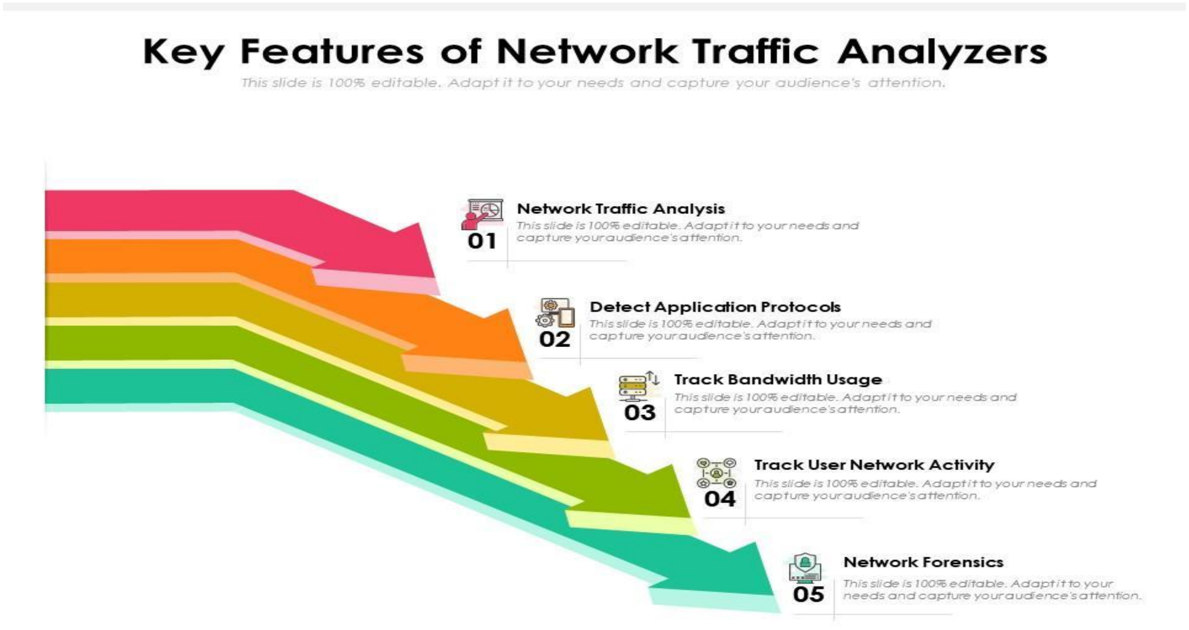
The inherently large storage volumes and processing power needed for Big Data workloads make it practical for most businesses to use cloud computing infrastructure and services for Big Data. But despite the attractiveness of cloud computing, exposed API keys, tokens, and misconfigurations are risks in the cloud worth taking seriously. What if someone leaves a AWS data lake in S3 completely open and accessible to anyone on the Internet? Mitigating these risks becomes easier with an automated scanning tool that works fast to scan public cloud assets for security blind spots.

2.8.4.4. Centralized key management

In a complex Big Data ecosystem, the security of encryption requires a centralized key management approach to ensure effective policy-driven handling of encryption keys. Centralized key management also maintains control over key governance from creation through to key rotation. For businesses running Big Data workloads in the cloud, bring your own key (BYOK) is probably the best option that allows for centralized key management without handing over control of encryption key creation and management to a third-party cloud provider.

2.8.4.5. Network traffic analysis

In a Big Data pipeline, there is constant traffic flow as data gets ingested from many different sources, including streaming data from social media platforms and data from user endpoints. Network traffic analysis provides visibility into network traffic and any potential anomalies such as malicious data from IoT devices or unencrypted communications protocols being used.


Key Features of Network Traffic Analyzers

This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

01 Network Traffic Analysis
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

02 Detect Application Protocols
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

03 Track Bandwidth Usage
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

04 Track User Network Activity
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

05 Network Forensics
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

Figure 14. Key features of network traffic analyzers

2.8.4.6. Insider threat detection

A 2021 report found that 98 percent of organizations feel vulnerable to insider attacks. In the context of Big Data, insider threats pose serious risks to the confidentiality of sensitive company information. A malicious insider with access to analytics reports and dashboards could reveal insights to competitors or even offer their login credentials for sale. A good place to start with insider threat detection is by examining logs for common business applications, such as RDP, VPN, Active Directory, and endpoints. These logs can reveal abnormalities worth investigating, such as unexpected data downloads or abnormal login times.

2.8.4.7. Threat hunting

Threat hunting proactively searches for threats lurking undetected in your network. This process requires the skill set of an experienced cybersecurity analyst to formulate hypotheses about potential threats using intelligence from real-world attacks, threat campaigns, or correlating findings from different security tools. Ironically, Big Data can help improve threat hunting efforts by uncovering hidden insights in large volumes of security data. But to improve Big Data security, threat hunting monitors datasets and infrastructure for artifacts that indicate a compromise of your Big Data environment.

2.8.4.8. Incident investigation

Monitoring Big Data logs and tools for security purposes generates a lot of information which usually ends up in a Security information and event management (SIEM) solution. Given the enormous volumes of data often generated at high velocity in a Big Data environment, SIEM solutions are prone to false positives and analysts get inundated with too many alerts. Ideally, some sort of incident response tool can provide context into security threats that enable faster, more efficient incident investigation.

2.8.4.9. User behavior analytics

User behavior analytics goes a step further than insider threat detection by providing a dedicated toolset to monitor the behavior of users on the systems they interact with. Typically, behavior analytics uses a scoring system to create a baseline of normal user, application, and device behaviors and then alerts you when there are deviations from these baselines. With user behavior analytics you can better detect insider threats and compromised user accounts that threaten the confidentiality, integrity, or availability of the assets within your Big Data environment.

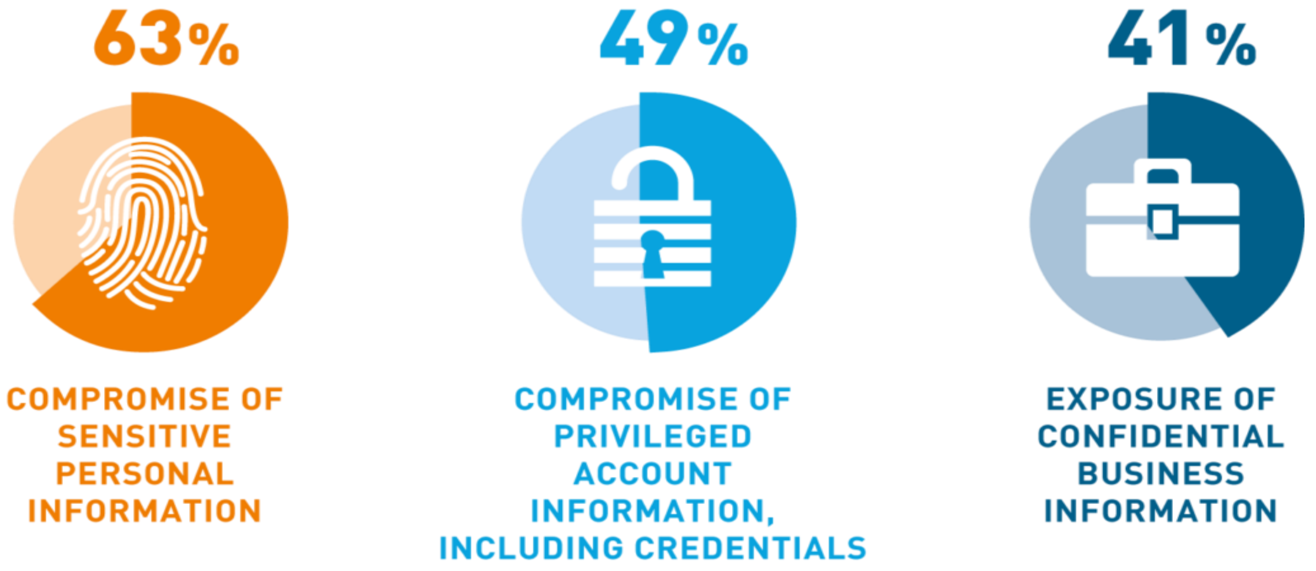


Figure 15. User Behavior Analytics

2.8.4.10. Data exfiltration detection

The prospect of unauthorized data transfers keeps security leaders awake at night particularly if data exfiltration occurs in Big Data pipelines where enormous volumes of potentially sensitive assets can be copied. Detecting data exfiltration requires in-depth monitoring of outbound traffic, IP addresses, and traffic. Preventing exfiltration in the first place comes from tools that find harmful security errors in code and misconfigurations along with data loss prevention and next-gen firewalls. Another important aspect is educating and raising awareness within your organization.

2.9. Conclusion

Big Data is revolutionizing data-driven decision-making across various domains, but it also introduces significant security and privacy challenges. This chapter outlined the key concepts, technologies, and current advancements in Big Data, with a focus on ensuring secure and reliable data management. A clear understanding of these foundations is vital for building scalable and secure Big Data systems.

3. Hadoop Architecture and Components

3.1. Introduction

This chapter provides an in-depth overview of Hadoop's core architecture, including its two main layers: the Hadoop Distributed File System (HDFS) for data storage and the MapReduce programming model for data processing. It also discusses additional components such as YARN (Yet Another Resource Negotiator), which manages cluster resources, and various ecosystem tools, which extend Hadoop's capabilities for data querying, analysis, and real-time access.

3.2. Overview of Hadoop ecosystem

Apache Hadoop, often referred to as Hadoop, is a powerful open-source framework designed to process and store huge data sets by distributing them across clusters of affordable commodity hardware. Its strength lies in its scalability and flexibility, allowing it to work with both structured and unstructured data. In this post, I'll walk you through the core components of Hadoop—large-scale data storage, processing, and management. By the end, you'll have a clear idea of how this foundational technology fits into big data ecosystems.

Hadoop isn't a single tool. It's an ecosystem of multiple modules that work together to manage data storage, processing, and resource coordination.

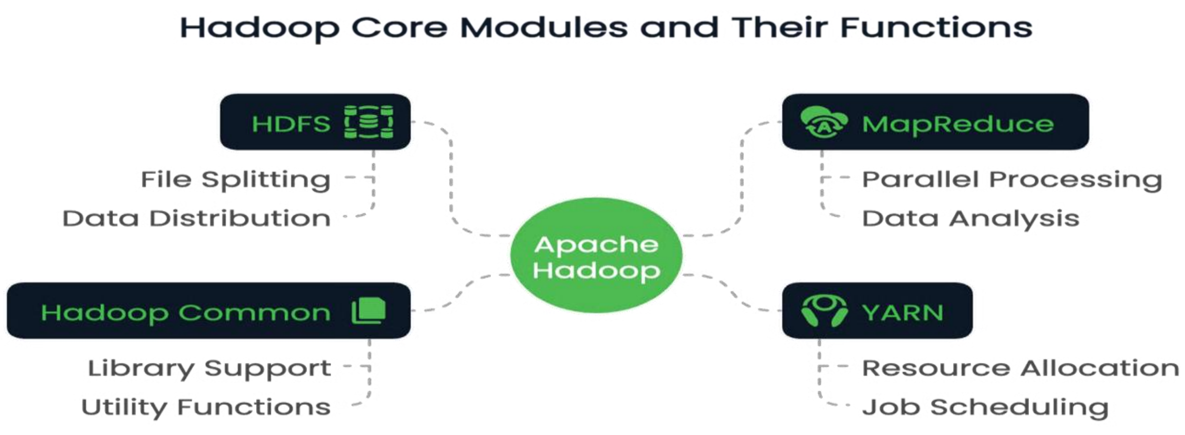


Figure 16. Apache Hadoop architecture

Hadoop consists of four fundamental modules:

- **HDFS (Hadoop Distributed File System):** A distributed storage system that divides files into blocks and distributes them across a cluster.
- **MapReduce:** A programming model for parallel data processing across multiple nodes.
- **YARN (Yet Another Resource Negotiator):** Manages resources and schedules jobs in the cluster.
- **Hadoop Common:** A set of shared libraries and utilities that support other modules.

These modules work together to create a distributed computing environment able of processing petabytes of data. That we will talk about it in the next paragraphs.

3.3. HDFS: Distributed File System

Of the four main components, HDFS is Hadoop's primary storage system. It is designed to reliably store the large amounts of data distributed across a cluster of machines discussed above. Its architecture is designed for this type of access to large data sets and is optimized for fault tolerance, scalability, and data locality.

3.3.1. Architecture and components

The HDFS architecture is based on a master-slave model. At the top is the Name Node, which manages metadata, i.e., the file system tree and information about the location of each file. It does not store the actual data.

Data Nodes are the workhorses. They manage the storage attached to the nodes and respond to client read/write requests. Each Data Node regularly reports to the Name Node with a heartbeat and block reports to ensure consistent progress.

Finally, HDFS includes a secondary Name Node, not to be confused with a failover node. Instead, it periodically checks the Name Node's metadata to reduce startup time and memory overhead.

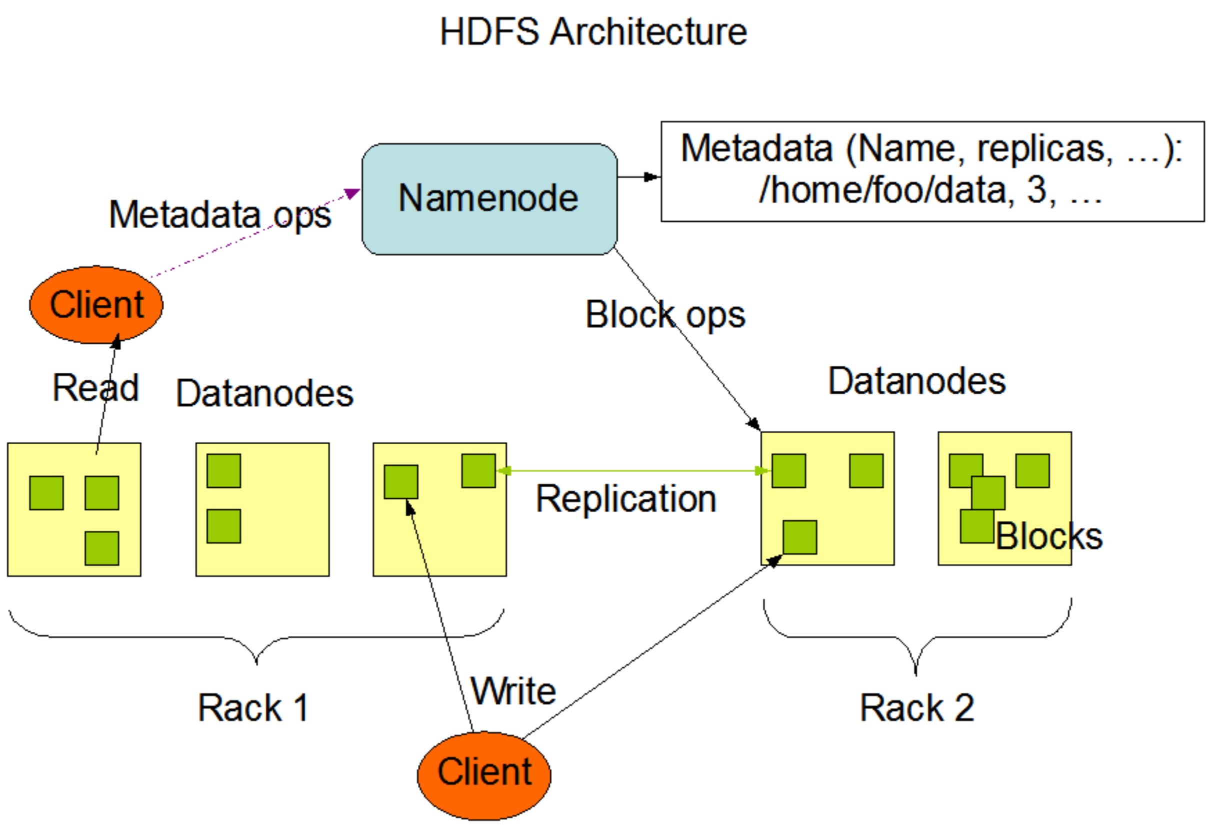


Figure 17. HDFS architecture

- HDFS storage mechanisms:

When a file is written to HDFS, it is divided into fixed size 64MB blocks. Each block is then distributed among different Data Nodes. This distribution strategy promotes parallelism when accessing data and increases fault tolerance.

Data in HDFS isn't stored only once. Each block is replicated (the default is three copies) and distributed across the entire cluster. During a read operation, the system retrieves data from the nearest available replica to maximize throughput and minimize latency. Writes go to one replica first and then propagate to the others, ensuring durability without immediate bottlenecks.

This block-level design also allows HDFS to scale horizontally. New nodes can be added to the cluster, and the system will rebalance the data to utilize the additional capacity.

For a comparison with other big data formats, learn how Apache Parquet optimizes columnar storage for analytical workloads.

3.3.2. Fault tolerance in HDFS

HDFS was designed with failures in mind. In large distributed systems, node failures are expected, not exceptional. HDFS ensures data availability through its replication mechanism. If a Data Node fails, the Name Node detects the failure through missed heartbeats and schedules replication of lost blocks to healthy nodes.

Additionally, the system continuously monitors block health and initiates replication if necessary. The redundant storage strategy, combined with proactive monitoring, ensures that no single point of failure can compromise data integrity or access.

3.4. YARN: Resource Management

YARN stands for Yet Another Resource Negotiator. It is the resource management layer of Hadoop. It was introduced in Hadoop 2.

YARN is designed with the idea of splitting up the functionalities of job scheduling and resource management into separate daemons. The basic idea is to have a global Resource Manager and application Master per application where the application can be a single job or DAG of jobs.

YARN consists of Resource Manager, Node Manager, and per-application Application Master.

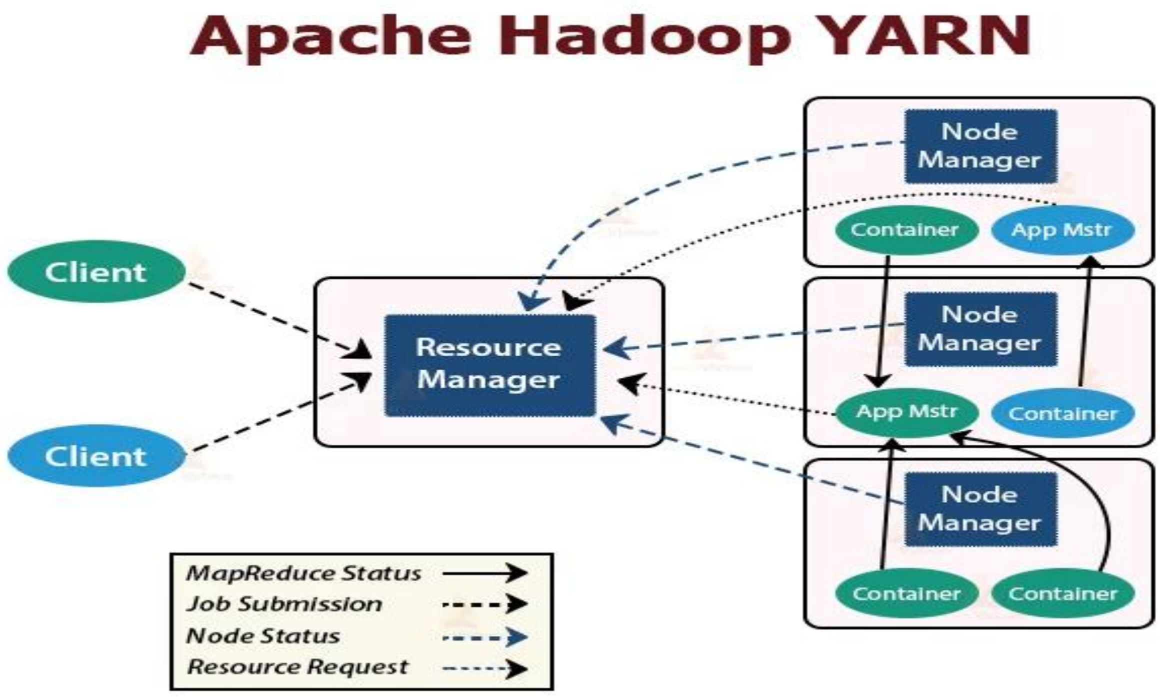


Figure 189. Apache Hadoop YARN

3.4.1. Resource Manager

It arbitrates resources amongst all the applications in the cluster. It has two main components that are Scheduler and the Application Manager.

3.4.1.1. Scheduler

- The Scheduler allocates resources to the various applications running in the cluster, considering the capacities, queues, etc.
- It is a pure Scheduler. It does not monitor or track the status of the application.

- Scheduler does not guarantee the restart of the failed tasks that are failed either due to application failure or hardware failure.
- It performs scheduling based on the resource requirements of the applications.

3.4.1.2. Application Manager

- They are responsible for accepting the job submissions.
- Application Manager negotiates the first container for executing application-specific Application Master.
- They provide service for restarting the Application Master container on failure.
- The per-application Application Master is responsible for negotiating containers from the Scheduler. It tracks and monitors their status and progress.

3.4.2. Node Manager

Node Manager runs on the slave nodes. It is responsible for containers, monitoring the machine resource usage that is CPU, memory, disk, network usage, and reporting the same to the Resource Manager or Scheduler.

3.4.3. Application Master

The per-application Application Master is a framework-specific library. It is responsible for negotiating resources from the Resource Manager. It works with the Node Manager(s) for executing and monitoring the tasks.

3.5. MAPREDUCE: Data processing

3.5.1. MapReduce Framework

Next comes Hadoop's processing engine, MapReduce. It enables distributed computing on large data sets by breaking tasks down into smaller, independent operations that can be executed in parallel.

The model simplifies complex data transformations and is particularly suitable for batch processing in a distributed environment.

3.5.2. MapReduce Overview

MapReduce follows a two-phase programming model: the Map and Reduce model.

During the mapping phase, the input dataset is divided into smaller chunks, and each chunk is processed to produce key-value pairs. These intermediate results are then shuffled and sorted before being passed to the reducing phase, where they are aggregated or transformed into results.

This model is very effective for tasks such as word frequency counting, log filtering, or assembling large-scale datasets.

Developers implement custom logic through functions map()and reduce(), which the framework orchestrates in the cluster.

3.5.3. MapReduce Job Execution Flow

When a MapReduce job is submitted, the input data is first divided into chunks. A Job Tracker (or Resource Manager in YARN-compatible versions) then coordinates the distribution of these chunks among available Task Trackers or Node Managers.

Map tasks run in parallel across nodes, reading input splits and emitting intermediate key-value pairs. These pairs are then automatically shuffled and sorted. Once this phase is complete, reduce tasks begin, extracting grouped data and applying aggregation logic to produce the final result.

The process ensures data locality by assigning tasks close to where the data resides, minimizing network congestion and improving throughput. Task failures are automatically detected and reassigned, contributing to system resilience.

3.5.4. Advantages and limitations of MapReduce

MapReduce offers several clear advantages:

- Excellent parallelism for large-scale batch processing
- Scalability to thousands of nodes
- Strong fault tolerance through task re-execution and checkpoint verification

But it also comes with compromises:

- The model is latency heavy and therefore not ideal for real-time processing.
- Writing custom MapReduce jobs can be cumbersome and difficult to debug.
- It lacks the interactivity and flexibility of newer tools like Apache Spark.

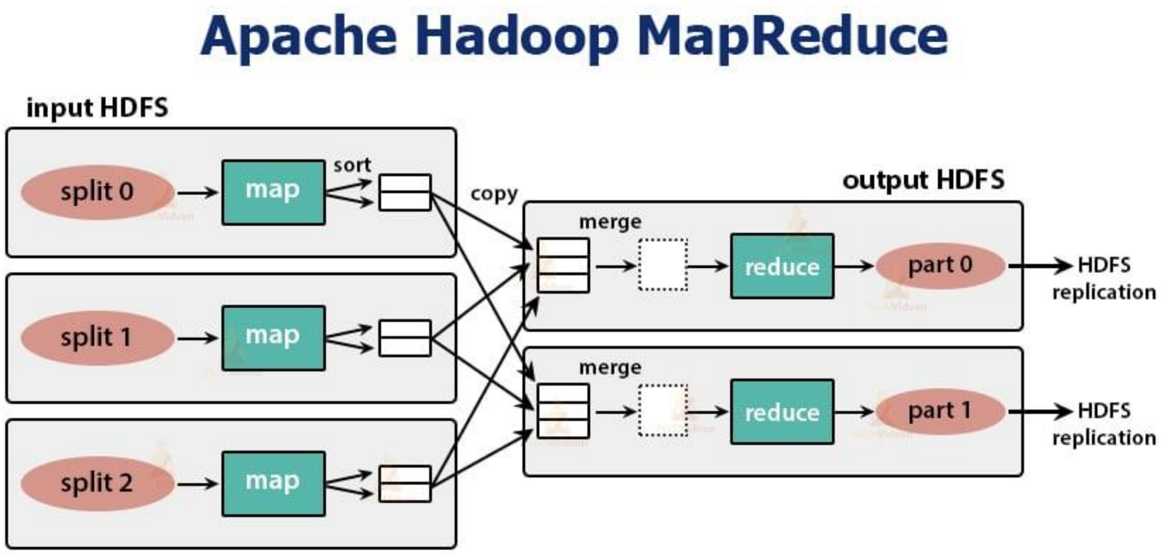


Figure 19. Apache Hadoop MapReduce

The MapReduce framework works on the <key, value> pairs. The MapReduce job is the unit of work the client wants to perform. MapReduce job mainly consists of the input data, the MapReduce program, and the configuration information. Hadoop runs the MapReduce jobs by dividing them into two types of tasks that are map tasks and reduce tasks. The Hadoop YARN scheduled these tasks and are run on the nodes in the cluster.

Due to some unfavorable conditions, if the tasks fail, they will automatically get rescheduled on a different node.

The user defines the map function and the reduce function for performing the MapReduce job.

The input to the map function and output from the reduce function is the key, value pair.

The function of the map tasks is to load, parse, filter, and transform the data. The output of the map task is the input to the reduce task. Reduce task then performs grouping and aggregation on the output of the map task. The MapReduce task is done in two phases:

3.5.5. Phase 1: Map phase

3.5.5.1. Record Reader

Hadoop divides the inputs to the MapReduce job into the fixed-size splits called input splits or splits. The Record Reader transforms these splits into records and parses the data into records, but it does not parse the records itself. Record Reader provides the data to the mapper function in key-value pairs.

3.5.5.2. Map

In the map phase, Hadoop creates one map task which runs a user-defined function called map function for each record in the input split. It generates zero or multiple intermediate key-value pairs as map task output.

The map task writes its output to the local disk. This intermediate output is then processed by the reduce tasks which run a user-defined reduce function to produce the final output. Once the job gets completed, the map output is flushed out.

3.5.5.3. Combine

Input to the single reduce task is the output from all the Mappers that is output from all map tasks. Hadoop allows the user to define a combiner function that runs on the map output.

Combiner groups the data in the map phase before passing it to Reducer. It combines the output of the map function which is then passed as an input to the reduce function.

3.5.5.4. Partition

When there are multiple reducers then the map tasks partition their output, each creating one partition for each reduce task. In each partition, there can be many keys and their associated values but the records for any given key are all in a single partition.

Hadoop allows users to control the partitioning by specifying a user-defined partitioning function. Generally, there is a default Partitioner that buckets the keys using the hash function.

3.5.6. Phase 2: Reduce phase

The various phases in reduce task are as follows:

3.5.6.1. Sort and Shuffle

The Reducer task starts with a shuffle and sort step. The main purpose of this phase is to collect the equivalent keys together. Sort and Shuffle phase downloads the data which is written by the partitioner to the node where Reducer is running.

It sorts each data piece into a large data list. The MapReduce framework performs this sort and shuffles so that we can iterate over it easily in the reduce task.

The sort and shuffling are performed by the framework automatically. The developer through the comparator object can have control over how the keys get sorted and grouped.

3.5.6.2. Reduce

The Reducer which is the user-defined reduce function performs once per key grouping. The reducer filters, aggregates, and combines data in several different ways. Once the reduce task is completed, it gives zero or more key-value pairs to the Output Format. The reduce task output is stored in Hadoop HDFS.

3.5.6.3. Output Format

It takes the reducer output and writes it to the HDFS file by Record Writer. By default, it separates key, value by a tab and each record by a newline character.

3.6. AMBARI: Cluster management and other components

The Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs.

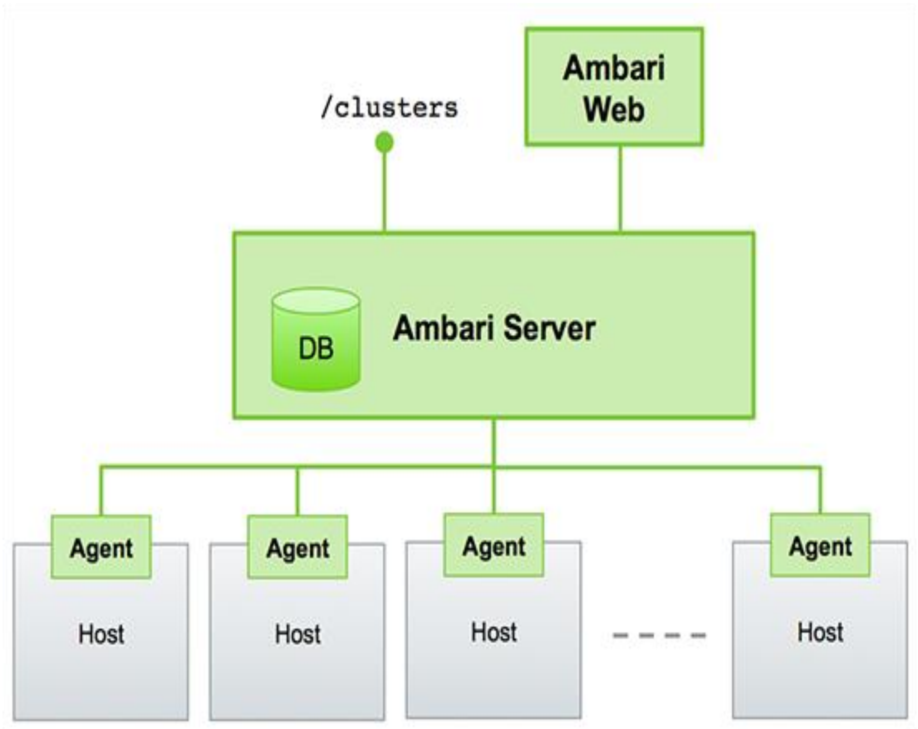


Figure 20. Amabari architecture

Ambari enables System Administrators to:

- Provision a Hadoop cluster

  Ambari provides a step-by-step wizard for installing Hadoop services across any number of hosts.
  Ambari handles configuration of Hadoop services for the cluster.
- Manage a Hadoop cluster

  Ambari provides central management for starting, stopping, and reconfiguring Hadoop services across the entire cluster.
- Monitor a Hadoop cluster

  Ambari provides a dashboard for monitoring health and status of the Hadoop cluster.

  Ambari leverages Ambari Metrics System for metrics collection.

  Ambari leverages Ambari Alert Framework for system alerting and will notify you when your attention is needed (e.g., a node goes down, remaining disk space is low, etc.).

- Cluster lifecycle

The cluster lifecycle in Ambari follows a well-defined path from creation through operation to potential decommissioning.

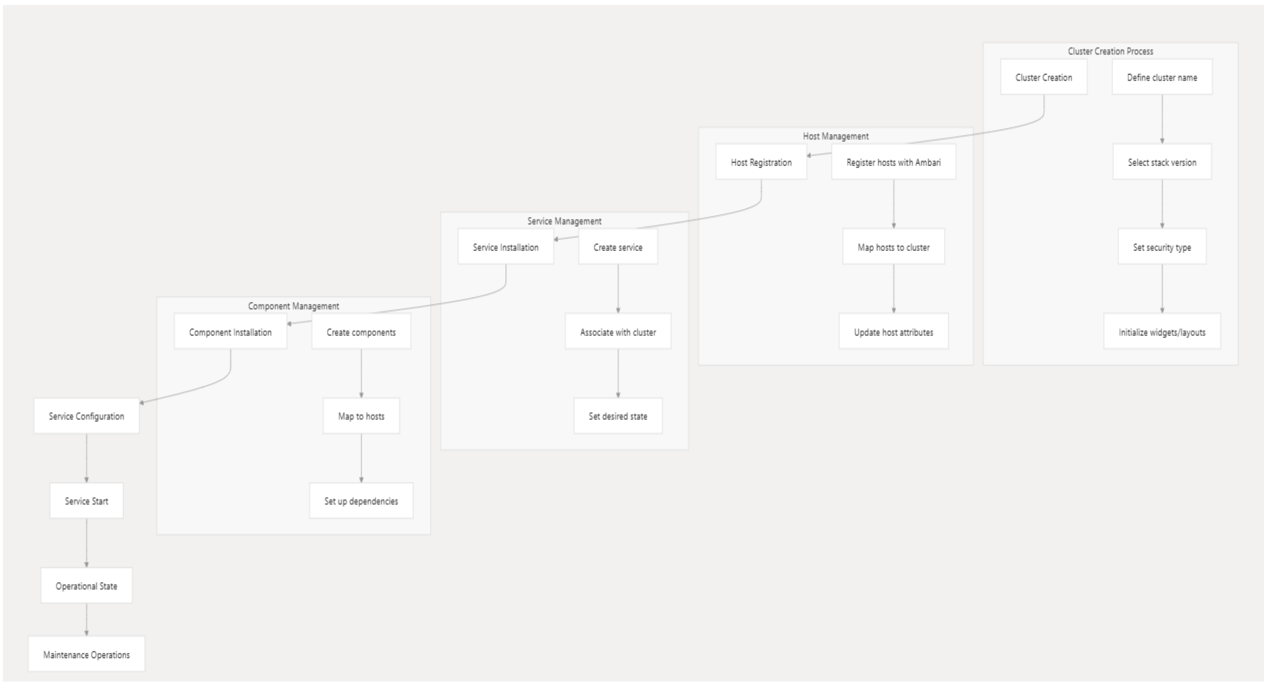


Figure 212. Cluster lifecycle

- Cluster creation and host mapping

  Creating a cluster is a multi-step process that begins with cluster definition and proceeds to host mapping and service installation.

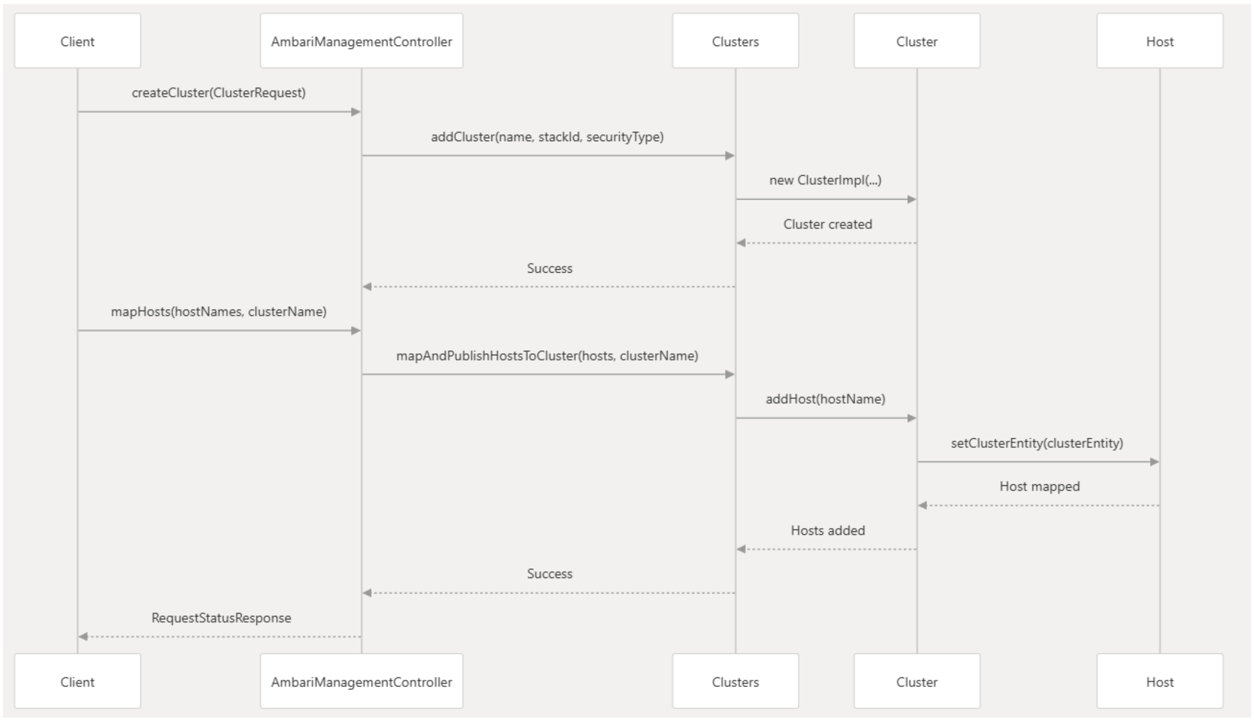

Client
AmbariManagementController
Clusters
Cluster
Host
createCluster(ClusterRequest)
addCluster(name, stackId, securityType)
new ClusterImpl(...)
Cluster created
Success
mapHosts(hostNames, clusterName)
mapAndPublishHostsToCluster(hosts, clusterName)
addHost(hostName)
setClusterEntity(clusterEntity)
Host mapped
Hosts added
Success
RequestStatusResponse

Figure 22. Cluster creation and host mapping

- **Cluster creation code flow:**
  - The client initiates cluster creation through the **AmbariManagementControllerImpl.createCluster()** method
  - This validates the request and calls **clusters.addCluster()** with a name, stack ID, and security type
  - The **ClusterImpl** constructor is called to initialize the cluster
  - Hosts are then mapped to the cluster using **mapAndPublishHostsToCluster()**
  - Host mappings that have persisted in the database are published as events to update the topology
- **Host mapping process**

  Host mapping involves creating relationships between host entities and cluster entities. When a host is mapped to a cluster:
  - An entry is added to the ClusterEntity.hostEntities collection
  - The HostEntity.clusterEntities collection is updated to include the cluster
  - Events are published to update topology information in the system
- **Service installation process**

  Service installation follows these steps:
  - Create a service entity using **ServiceFactory.createNew()**
  - Associate it with the cluster through **cluster.addService()**
  - Create service components using **ServiceComponentFactory.createNew()**
  - Associate components with the service via **service.addServiceComponent()**
  - Install service components on hosts through **createServiceComponentHost()**

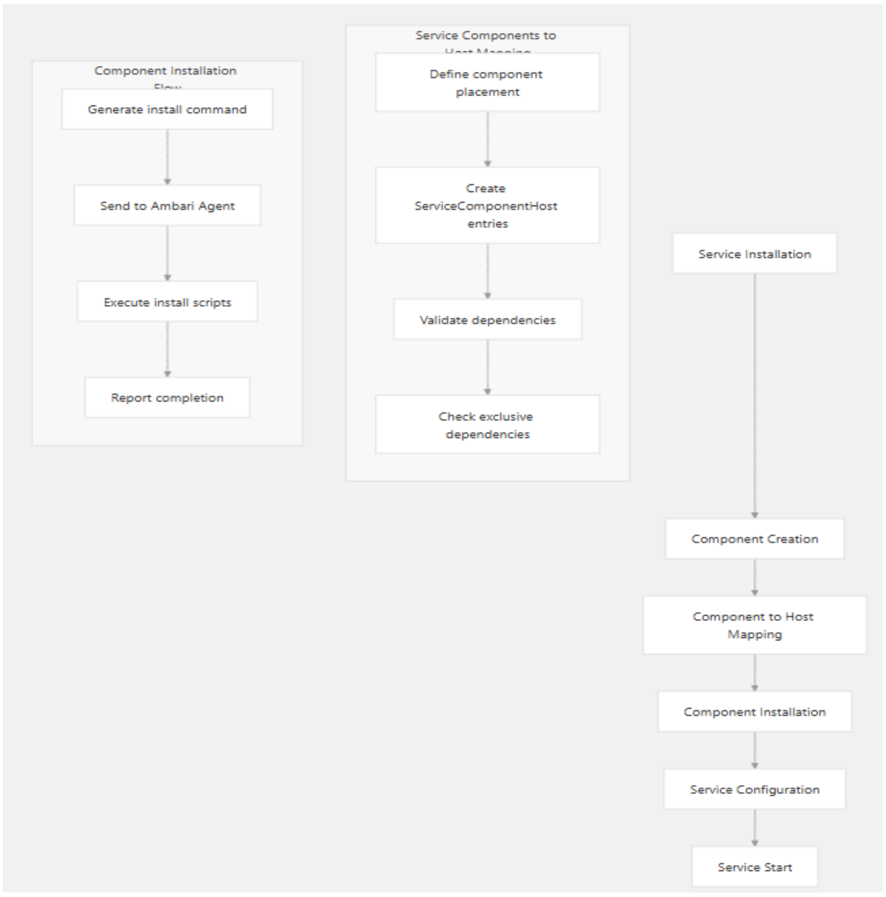


Figure 23. Service installation process

3.7. Securing Hadoop with Kerberos

3.7.1. Kerberos Overview

Kerberos is a network authentication protocol that provides a way for clients and servers to authenticate each other over an insecure network. It was developed by the Massachusetts Institute of Technology (MIT) in the 1980s and has since become a widely adopted authentication protocol in both Unix and Windows environments.

In Kerberos authentication, a client requests a ticket from a Kerberos authentication server (AS) to access a service on a network. The AS issues a ticket-granting ticket (TGT) to the client, which the client can then use to request a service ticket from the ticket-granting server (TGS). The TGS verifies the client's identity and issues a service ticket that the client can use to access the requested service.

The advantage of using Kerberos authentication is that it provides a secure way to authenticate clients and servers over an insecure network. It also provides a way to ensure that only authorized users can access the data and provides a way to audit who has accessed the data and when.

In Hadoop, Kerberos authentication is used to authenticate clients and servers in a Hadoop cluster. It provides a way to ensure that only authorized users can access the data stored in the Hadoop cluster and provides a way to audit who has accessed the data and when.

3.7.2. Kerberos System architecture

In Kerberos authentication, a client requests a ticket from the Kerberos authentication server (AS) to access a service on a network. The AS issues a ticket-granting ticket (TGT) to the client, which the client can then use to request a service ticket from the ticket-granting server (TGS). The TGS verifies the client's identity and issues a service ticket that the client can use to access the requested service. The Kerberos authentication process can be broken down into the following steps:

- **Authentication request:** The client sends an authentication request to the Kerberos authentication server (AS) to request a TGT. The request includes the client's username and a timestamp.
- **TGT issuance:** The AS verifies the client's identity and issues a TGT that is encrypted using the client's password. The TGT includes the client's username, the network address of the client, the network address of the TGS, and a timestamp. The TGT is valid for a specified period of time.
- **Service ticket request:** The client sends a request to the TGS for a service ticket to access a specific service. The request includes the TGT that was issued by the AS, the name of the requested service, and a timestamp.
- **Service ticket issuance:** The TGS verifies the client's identity by decrypting the TGT using the client's password. If the TGT is valid, the TGS issues a service ticket that is encrypted using a secret key shared between the client and the requested service. The service ticket includes the client's username, the network address of the client, the name of the requested service, and a timestamp.
- **Service access:** The client presents the service ticket to the requested service, which decrypts the service ticket using the secret key shared between the client and the service. If the service ticket is valid, the client is granted access to the requested service.

The Kerberos authentication process provides a secure way to authenticate clients and servers over an insecure network. It also provides a way to ensure that only authorized users can access the data and provides a way to audit who has accessed the data and when.

3.7.3. Security Motivation Behind Kerberos in Hadoop

In Hadoop, Kerberos authentication is used to provide secure access control to data stored in the Hadoop Distributed File System (HDFS) and to secure Hadoop services such as MapReduce, YARN, and HBase.

Without Kerberos authentication, anyone who has access to the Hadoop cluster can read, write, and modify data stored in HDFS or launch MapReduce jobs, which can lead to data breaches, unauthorized access, and other security issues.

3.7.4. Benefits of a Kerberos-secured Hadoop cluster

3.7.4.1. Authentication and Authorization

Kerberos authentication provides strong authentication and authorization by verifying the identity of the user or service before granting access to the data or service. This ensures that only authorized users and services can access the data, and that data is protected from unauthorized access.

3.7.4.2. **Secure communication**

Kerberos authentication ensures secure communication between the Hadoop components by encrypting the communication channels. This prevents eavesdropping, data tampering, and other security threats.

3.7.4.3. **Data protection**

Kerberos authentication in Hadoop provides data protection by encrypting the data stored in HDFS using Hadoop Transparent Data Encryption (TDE) and securing the data in transit using Secure Sockets Layer (SSL) encryption.

3.7.4.4. **Auditing and compliance**

Kerberos authentication provides auditing and compliance capabilities by logging all authentication and authorization events in the Hadoop Audit logs. This helps administrators to monitor and track user and service activities and ensure compliance with security policies and regulations.

In summary, Kerberos authentication is an essential security feature in Hadoop that provides secure access control to Hadoop services and data. It ensures strong authentication, secure communication, data protection, and auditing and compliance capabilities.

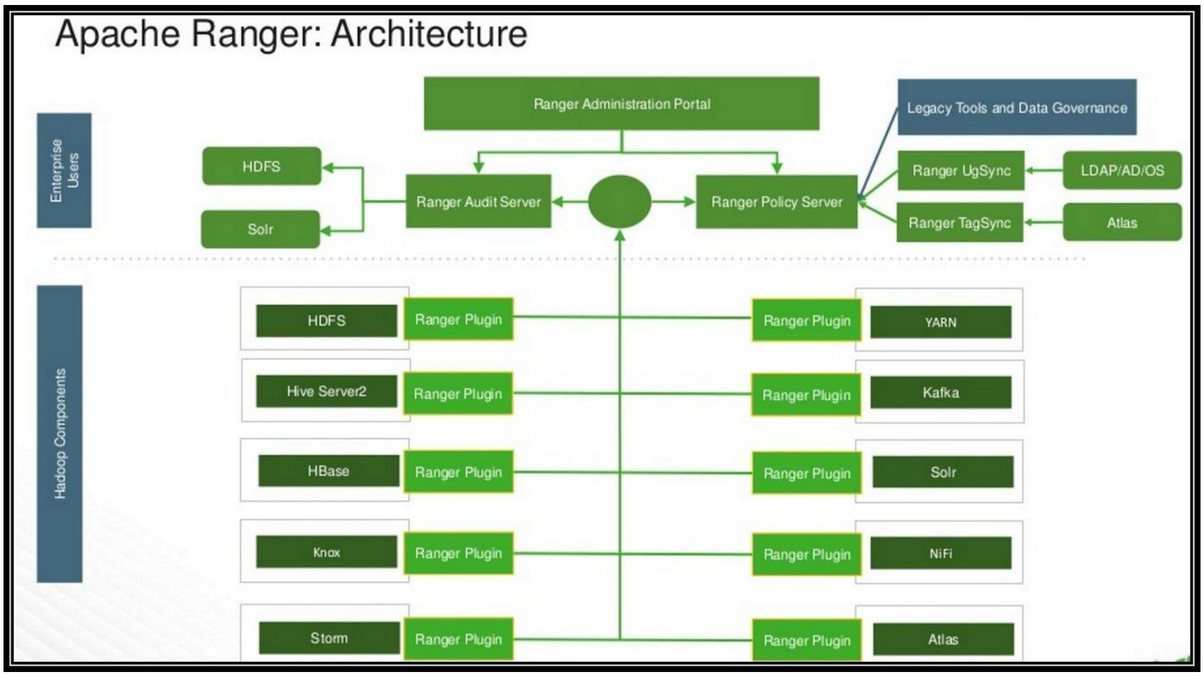


Figure 24. Authorization and auditing with Apache Ranger

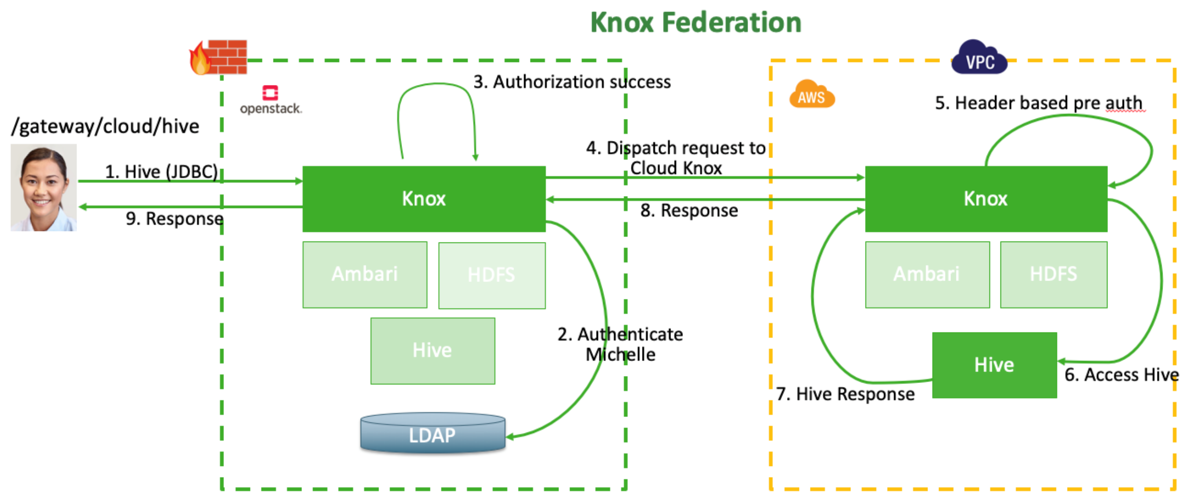


Figure 25. Secure gateway with Apache Knox

3.8. Data governance

Data governance in a Hadoop cluster involves organized management of metadata, access, quality, and compliance. This encompasses several fundamental aspects:

- Cataloging and metadata

Solutions such as Apache Atlas or Ranger allow you to create a centralized catalog of datasets, manage schemas, track data owners, and maintain a complete history of changes (provenance).

- Access control & auditability

Data access is governed by role- and group-based access policies (RBAC). This allows all queries and operations to be recorded (audit logs) to ensure traceability and compliance (e.g., GDPR, HIPAA).

- Data quality and compliance

Governance ensures that only reliable (valid, complete) data is stored or analyzed. It aligns with the strict requirements of regulated industries, particularly finance and healthcare.

3.9. Hadoop data encryption

3.9.1. Background

Encryption can be done at different layers in a traditional data management software/hardware stack. Choosing to encrypt at a given layer comes with different advantages and disadvantages.

- **Application-level encryption:** This is the most secure and most flexible approach. The application has ultimate control over what is encrypted and can precisely reflect the requirements of the user. However, writing applications to do this is hard. This is also not an option for customers of existing applications that do not support encryption.

- **Database-level encryption:** Like application-level encryption in terms of its properties. Most database vendors offer some form of encryption. However, there can be performance issues. One example is that indexes cannot be encrypted.
- **Filesystem-level encryption:** This option offers high performance, application transparency, and is typically easy to deploy. However, it is unable to model some application-level policies. For instance, multi-tenant applications might want to encrypt based on the end user. A database might want different encryption settings for each column stored within a single file.
- **Disk-level encryption:** Easy to deploy and high performance, but also quite inflexible. Only really protects against physical theft.

HDFS-level encryption fits between database-level and filesystem-level encryption in this stack. This has a lot of positive effects. HDFS encryption can provide good performance and existing Hadoop applications can run transparently on encrypted data. HDFS also has more context than traditional filesystems when it comes to making policy decisions.

HDFS-level encryption also prevents attacks at the filesystem-level and below (so-called "OS-level attacks"). The operating system and disk only interact with encrypted bytes, since the data is already encrypted by HDFS.

3.9.2. Use cases

Data encryption is required by several different government, financial, and regulatory entities. For example, the health-care industry has HIPAA regulations, the card payment industry has PCI DSS regulations, and the US government has FISMA regulations. Having transparent encryption built into HDFS makes it easier for organizations to comply with these regulations.

Encryption can also be performed at the application-level, but by integrating it into HDFS, existing applications can operate on encrypted data without changes. This integrated architecture implies stronger encrypted file semantics and better coordination with other HDFS functions.

3.9.3. Ranger Key Management Service (Ranger KMS)

An open-source key management service based on Hadoop's KeyProvider API. For HDFS encryption, the Ranger KMS has three basic responsibilities:

- Provide access to stored encryption zone keys.
- Generate and manage encryption zone keys and create encrypted data keys to be stored in Hadoop.
- Audit all access events in Ranger KMS

3.10. Conclusion

we have studied Hadoop Architecture. The Hadoop follows master-slave topology. The master nodes assign tasks to the slave nodes. The architecture comprises three layers that are HDFS, YARN, and MapReduce. HDFS is the distributed file system in Hadoop for storing big data. MapReduce is the processing framework for processing vast data in the Hadoop cluster in a distributed manner. YARN is responsible for managing the resources amongst applications in the cluster. The HDFS daemon NameNode and YARN daemon ResourceManager run on the master node in the Hadoop cluster. The HDFS daemon DataNode and the YARN NodeManager run on the slave nodes. HDFS and MapReduce framework run on the same set of nodes, which result in very high aggregate bandwidth across the cluster.

4. Securing a Hadoop Ecosystem: an E2E Implementation

4.1. Introduction

To address these vulnerabilities, this chapter presents the implementation of Kerberos, a secure authentication protocol based on symmetric key cryptography and a trusted third-party model. Kerberos is widely used to enforce strong authentication across networked services, and its integration with Hadoop significantly enhances cluster security. This chapter details the step-by-step process of setting up a Kerberos-secured Hadoop environment, including the installation and configuration of the Key Distribution Center (KDC), service principals, and client authentication.

4.2. Work environment

Our implementation utilizes 5 machines, 2 local servers* connected to university network and 3 Virtual Private Servers from Oracle Cloud Infrastructure located in Madrid, Paris and Frankfurt respectfully, all running Canonical Ubuntu 24.04.

Table 1. Environment characteristics

Role	Hardware Configuration	Location
Kerberos + OpenLDAP + Authentik on Docker	2 cores/4 threads, 4GB RAM, 100Mbps bandwidth	Faculty of Sciences Semlalia
Hadoop Namenode + Cloudera Hue on Docker	2 cores/4 threads, 4GB RAM, 1Gbps bandwidth	Faculty of Sciences Semlalia
NGINX + Authentik + Alpine SSH, all on Docker	1 core/2 thread, 1GB RAM + 4GB SWAP, 50Mbps bandwidth**	Oracle Madrid
Hadoop Datanode 1 + SSH	1 core/2 thread, 1GB RAM + 4GB SWAP, 50Mbps bandwidth**	Oracle Paris
Hadoop Datanode 2 + SSH	1 core/2 thread, 1GB RAM + 4GB SWAP, 50Mbps bandwidth**	Oracle Frankfurt

**Machines running Proxmox VE 8.4 hypervisors, services running in a Canonical Ubuntu 24.04 Virtual Machine*
***Oracle Cloud Infrastructure Standard E2.1 Micro shape running Canonical Ubuntu 24.04*

We have made a suite of bash scripts that automate the installation and configuration of these services, they can be found on ***https://github.com/MerGr/HadoopxKerberos***

4.3. Features

❖ **Distributed Minimal 3 Node Hadoop Cluster**

A compact minimal cluster setup comprising of one NameNode + ResourceManager (master) and two DataNodes + NodeManagers (workers) with HDFS and Yarn enabled in Kerberos and HTTPS secure mode

❖ **Kerberos 5 with OpenLDAP Backend**

MIT Kerberos is configured with OpenLDAP as its backend for principal storage.

- The KDC consults LDAP for authentication; OpenLDAP stores both user and service principals in its directory
- Tools like krdb5_ldap_util and Kerberos schemas enable synchronized user data, allowing managed replication and centralized account control.
- Hadoop services obtain Kerberos credentials (TGTs) using keytabs issued from this centralized KDC.

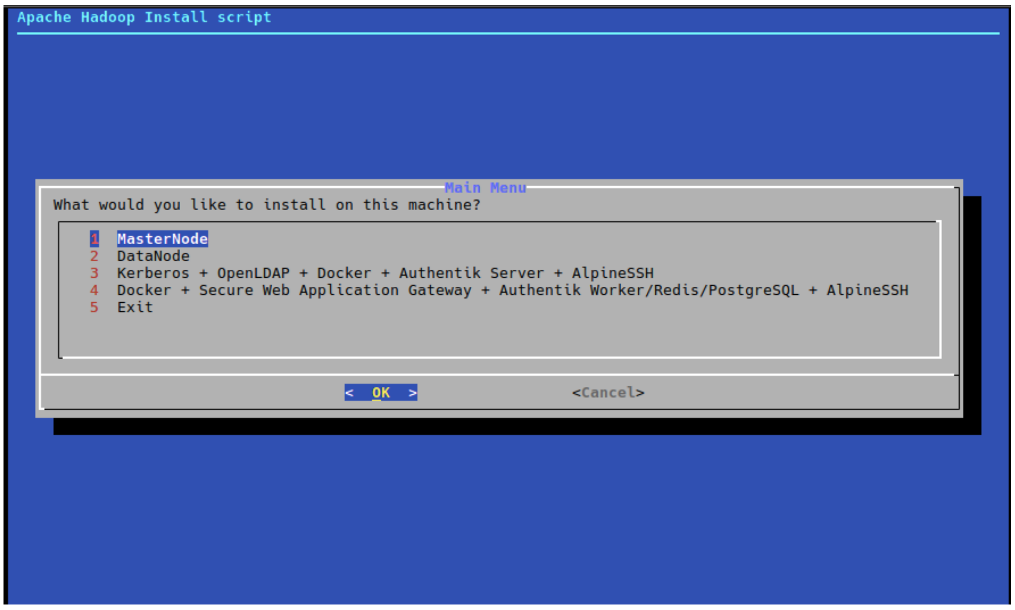
❖ **NGINX Web Proxy + Authentik + SPNEGO + SSL**

The Docker-based SWAG (Secure Web Application Gateway) image serves as a reverse proxy, integrating:

- SSL/TLS via certbot for encryption and fail2ban for protecting against brute-force attacks.
- Authentik acts as an Identity Provider, consuming Kerberos tickets (via SPNEGO) and exposing it through Docker as an OIDC-compatible front-end. It proxies auth to Hadoop web UIs, enabling secure, ticket-based access.

For ease of deployment, we have written multiple BASH automation scripts that can setup a minimal-configuration SSH-**H**adoop-**O**penLDAP-**N**GINX-**A**uthentik-**K**erberos-**D**ocker

5-node setup:



```
Apache Hadoop Install script

                        Main Menu
What would you like to install on this machine?

  1  MasterNode
  2  DataNode
  3  Kerberos + OpenLDAP + Docker + Authentik Server + AlpineSSH
  4  Docker + Secure Web Application Gateway + Authentik Worker/Redis/PostgreSQL + AlpineSSH
  5  Exit

             <  OK  >                  <Cancel>
```

Figure 26. Apache Hadoop Install script

4.4. System Architecture and Usage Guide

This implementation sets up a 5-node system that features:

- 1 Dedicated Namenode
- 2 Dedicated Datanodes
- 1 Kerberos 5 server with OpenLDAP backend
- 1 Web accessible NGINX webserver with Authentik for web-based management and SPNEGO authentication, Authentik can also be used to handle internal Kerberos authentication mechanisms normally handled by OpenSSH + KDC GSSAPI based authentication

4.4.1. Hadoop

⚠ ***Unless otherwise specified, the Hadoop setup must be done for each node of the cluster***

Due to limited hardware resources on the tested machines, each node in the hadoop cluster has only HDFS and YARN enabled.

4.4.1.1. Hadoop UNIX users

As recommended by The Apache Hadoop's official documentation, each service is bound to a unique UNIX user, with specific permissions.

Table 2. Unix user

UNIX user	Purpose
hadoopadmin	Admin user that has full access to the hadoop local install and is what the user SSH into
hdfs	handles HDFS related processes
yarn	handles YARN related processes
hive	handles Hive related processes, is part of HDFS
mapred	handles MapReduce and JobHistory related processes
HTTP	handles HTTPFS related processes, is part of HDFS

All these UNIX users do not have sudo or docker permissions, nor can they access home folders outside of their owner:group. They all share the same SSH public key in their ***~/.ssh/authorized_keys***, trading security for convenience.

hadoopadmin can only use sudo to switch to the other hadoop UNIX users to run commands as them, all the hadoop UNIX users are part of the hadoopadmin group, so in case of permission issues, one should only need to change group permissions.

4.4.1.2. Manual installation of the Hadoop cluster

After we have prepared our environments and our UNIX users, we can start setting up Hadoop and its services.

- ***We will login to a user with sudo access; as some commands require root privileges.***

4.4.1.2.1. Installing dependencies

As of this report's writing, the latest version of Hadoop is ***3.4.1***, which requires Java Runtime 8, SSH and Parallel Remote Shell, provided by the packages **openjdk-8-jre**, **ssh** and **pdsh** respectively.

For a pure CLI installation it is also recommended to install **wget**, **coreutils** and **sed**.

4.4.1.2.2. Preparing directories, binaries & files

- We must create the root directory where Hadoop will be located first; recommending ***/opt/hadoop*** or ***/home/hadoopadmin/hadoop.***
- We download the latest Hadoop tar archive from the Apache Foundation's Hadoop repository and its sha512 hash to verify our downloaded archive. After both files are downloaded to the local machine, we compare hashes with the ***sha512sum.***
- With a successful hash verification, we extract the archive to our root directory, we will note the absolute path as ***HADOOP_HOME***
- We switch to the new Hadoop installation root directory, and we create two folders; ***runtime***, which should contain two subfolders; ***log*** and ***proc,*** and ***data.*** We set both folders' permissions recursively to **770** and we recursively set ***runtime***'s owner and group as ***hadoopadmin:hadoopadmin*** and ***data***'s as ***hdfs:hadoopadmin***.
- In your shell's configuration file, like ***~/.bashrc***, we append these following environmental variables

Table 3. Hadoop Initial Installation environmental variables

Environmental variable	Value
HADOOP_HOME	Location of the Hadoop Installation root directory
JAVA_HOME	Location of the JRE executable
HADOOP_PID_DIR	Points to ${HADOOP_HOME}/runtime/proc
HADOOP_LOG_DIR	Points to ${HADOOP_HOME}/runtime/log
HADOOP_SBIN	Points to ${HADOOP_HOME}/sbin
PATH	Append ${HADOOP_HOME}/bin:${HADOOP_SBIN} to PATH

4.4.1.3. Hadoop Configuration

Hadoop stores its configurations in ***etc/hadoop/*** as XML configuration files, each node can require unique properties and values or share the same property and/or value across all the nodes.

It is also advised to append these variables to ***hadoop-env.sh*** on each node:

```
export HDFS_NAMENODE_USER=hdfs
export HDFS_DATANODE_USER=hdfs
export YARN_RESOURCEMANAGER_USER=yarn
export YARN_NODEMANAGER_USER=yarn
export MAPRED_HISTORYSERVER_USER=mapred
```

4.4.1.3.1. Namenode Configuration

For a secure Kerberized configuration on the namenode, the following properties need to be specified in these XML files:

Table 4. Namenode core-site.xml

Property	**Value**	**Description**
fs.defaultFS	hdfs://localhost:<port>	HDFS Namenode runs on this local machine, default port is 9000
hadoop.security.authentication	kerberos	Authentication type Hadoop should use
hadoop.security.authorization	true	Must be specified to enable kerberized RPC
hadoop.http.authentication.kerberos.principal	HTTP/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the namenode HTTP services
hadoop.http.authentication.kerberos.keytab	/etc/krb5.keytab	The keytab containing the password for the HTTP principal

Table 5. Namenode hdfs-site.xml

Property	**Value**	**Description**
dfs.namenode.name.dir	file://<absolute_path_to _hadoop_data_dir>	The location where the Namenode data is stored
dfs.namenode.kerberos.principal	hdfs/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the namenode HDFS services
dfs.namenode.keytab.file	/etc/krb5.keytab	The keytab containing the password for the HDFS principal
dfs.web.authentication.kerberos.principal	HTTP/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the HTTP services
dfs.web.authentication.kerberos.keytab	/etc/krb5.keytab	The keytab containing the password for the HTTP principal

dfs.http.policy	HTTPS_ONLY	Can be set to HTTP_ONLY, HTTP_AND_HTTPS or HTTPS_ONLY, HTTPS require a valid SSL root CA certificate and a properly configured ssl-server and ssl-client XML files
dfs.namenode.http-address	0.0.0.0:9870	Port and listen address for the HTTP HDFS WebUI
dfs.namenode.https-address	0.0.0.0:9871	Port and listen address for the HTTPS HDFS WebUI
dfs.block.access.token.enable	true	Must be set to enable Secure Mode.

Table 6. Namenode yarn-site.xml

Property	Value	Description
yarn.resourcemanager.hostname	localhost	The binding address for the ResourceManager
yarn.resourcemanager.principal	yarn/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the namenode YARN services
yarn.resourcemanager.keytab	/etc/krb5.keytab	The keytab containing the password for the YARN principal
yarn.webapp.spnego-principal	HTTP/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the YARN HTTP services
yarn.webapp.spnego-keytab-file	/etc/krb5.keytab	The keytab containing the password for the YARN HTTP principal

Table 7. Namenode ssl-client.xml

Property	Value	Description
ssl.server.keystore.location	/path/to/keystore.jks	Absolute path to the SSL CA keystore.jks
ssl.server.keystore.password	Password	The keystore's password
ssl.server.truststore.location	/path/to/truststore.jks	Absolute path to the SS`L CA truststore.jks
ssl.server.truststore.password	password	The truststore's password

Table 8. Namenode ssl-server.xml

Property	Value	Description
ssl.server.keystore.location	/path/to/keystore.jks	Absolute path to the SSL CA keystore.jks
ssl.server.keystore.password	Password	The keystore's password
ssl.server.truststore.location	/path/to/truststore.jks	Absolute path to the SS`L CA truststore.jks
ssl.server.truststore.password	password	The truststore's password

For the CA certs, in the event of an inability to procure a valid SSL certificate, a self-signed certificate can be generated via OpenSSL and used.

4.4.1.3.2. Datanode Configuration

For a secure Kerberized configuration on the datanode, the following properties need to be specified in these XML files:

Table 9. Datanode core-site.xml

Property	Value	Description
fs.default.name	hdfs://namenode_hostname:<port>	Points to the HDFS Namenode, default port is 9000
hadoop.security.authentication	kerberos	Authentication type Hadoop should use
hadoop.security.authorization	true	Must be specified to enable kerberized RPC
hadoop.http.authentication.kerberos.principal	HTTP/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the datanode HTTP services
hadoop.http.authentication.kerberos.keytab	/etc/krb5.keytab	The keytab containing the password for the HTTP principal

Table 10. Datanode hdfs-site.xml

Property	Value	Description
dfs.data.dir	file://<absolute_path_to _hadoop_data_dir>	The location where the Datanode data is stored
dfs.datanode.hostname	Hostname or FQDN pointing to this machine	Unique hostname or FQDN for this node
dfs.datanode.kerberos.principal	hdfs/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the datanode HDFS services
dfs.datanode.keytab.file	/etc/krb5.keytab	The keytab containing the password for the HDFS principal
dfs.web.authentication.kerberos.principal	HTTP/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the HTTP services

dfs.web.authentication.kerberos.keytab	/etc/krb5.keytab	The keytab containing the password for the HTTP principal
dfs.http.policy	HTTPS_ONLY	Can be set to HTTP_ONLY, HTTP_AND_HTTPS or HTTPS_ONLY, HTTPS require a valid SSL root CA certificate and a properly configured ssl-server and ssl-client XML files
dfs.datanode.address	0.0.0.0:9866	Port and listen address for the HDFS service
dfs.datanode.http-address	0.0.0.0:9866	Port and listen address for the HTTP HDFS WebUI
dfs.datanode.https-address	0.0.0.0:9865	Port and listen address for the HTTPS HDFS WebUI
dfs.block.access.token.enable	true	Must be set to enable Secure Mode.
dfs.encrypt.data.transfer	true	Enables data encryption
dfs.data.transfer.protection	authentication	Must be specified to enable SASL authentication

Table 11. Datanode yarn-site.xml

Property	**Value**	**Description**
yarn.resourcemanager.hostname	Hostname or FQDN of the YARN ResourceManager	Points to the YARN ResourceManager, usually running on the Namenode
yarn.nodemanager.principal	yarn/hadoop.domain@COMPANY.REALM	The Kerberos 5 principal used to authenticate with the datanode YARN NodeManager services
yarn.nodemanager.keytab	/etc/krb5.keytab	The keytab containing the password for the YARN NodeManager principal

Table 12. Datanode ssl-client.xml

Property	**Value**	**Description**
ssl.server.keystore.location	/path/to/keystore.jks	Absolute path to the SSL CA keystore.jks
ssl.server.keystore.password	Password	The keystore's password
ssl.server.truststore.location	/path/to/truststore.jks	Absolute path to the SS`L CA truststore.jks
ssl.server.truststore.password	password	The truststore's password

Table 13. Datanode ssl-server.xml

Property	**Value**	**Description**

ssl.server.keystore.location	/path/to/keystore.jks	Absolute path to the SSL CA keystore.jks
ssl.server.keystore.password	Password	The keystore's password
ssl.server.truststore.location	/path/to/truststore.jks	Absolute path to the SS`L CA truststore.jks
ssl.server.truststore.password	password	The truststore's password

Do note that Namenode's ***workers*** file should only contain the Datanodes hostnames, one per line, and the Datanode's file should only contain localhost.

4.4.2. Communication between services via SSH and HTTPS

There are multiple ways to link between machines, ordered from fastest to slowest:

- Directly over LAN using private static IP addresses and hostnames
- Using an OpenVPN/WireGuard-based VPN
- SSH SOCKS5 (-D) or per-port TCP forwarding using the (-L) and (-R) flags

For our specific setup, LAN access is impossible as our machines are in multiple regions.

A WireGuard-based UDP VPN would be the preferred approach, but due to the strict Cadi Ayyad University Network Firewall, UDP traffic is seldom dropped on all ports. And utilizing UDP-to-TCP tunnelling with programs such as ***udp2raw*** was currently unachievable due to routing and packet collision issues for a multi-client configuration.

We ended up using SSH per-port TCP forwarding and assigning unique unprivileged ports for each of our nodes. A working solution but with an expensive speed, performance and complexity costs.

We set up multiple BASH scripts that we can call using SystemD services or a minimal Alpine Linux with OpenSSH pre-installed Docker containers to auto-start the SSH tunnels and restart in case of failure, like a connectivity loss,

On each forwarding rule, we bind the local machine's 0.0.0.0 or localhost address to the service bound on the remote machine's localhost, and to avoid complexity we set the same port for both local and remote.

To access these services, we can just point to our local machine's localhost and the corresponding port.

To ease on the complexity cost, we can offload some of the communication links from SSH to HTTP via our webserver which we will configure later.

These services communicate with each other via NGINX TCP streams on the subdomains, NGINX itself reaches these services via the SSH tunnels running on Docker.

We set up multiple DNS records for these services at our domain's registrar (in our case: Namecheap)

Table 14. Communication between services

Record Type	URL	Pointing to
CNAME	krb.mergrweb.me	OCI Madrid server which forwards via SSH Proxy to the Kerberos server
CNAME	hdfs-nn.mergrweb.me	OCI Paris server which forwards via SSH Proxy to the Namenode server
CNAME	authentik.mergrweb.me	OCI Madrid server which NGINX proxies to Authentik-server Docker container
A	hdfs-dn1.mergrweb.me	OCI Frankfurt server
A	hdfs-dn2.mergrweb.me	OCI Paris server

4.4.3. Kerberos & OpenLDAP

To install MIT Kerberos 5 server with OpenLDAP as the database backend on our dedicated Kerberos/AD machine, we first must make sure we have the required packages:

- For Kerberos: **krb5-kdc krb5-admin-server krb5-kdc-ldap**
- For OpenLDAP: **slapd ldap-utils gunzip schema2ldif**

4.4.3.1. Kerberos Installation

After we have installed the needed packages for Kerberos, we must first set a valid hostname pointing to our loopback interface (127.0.0.1) on ***/etc/hosts***, for example we append this line:

```
127.0.0.1        krb.mergrweb.me
```

Afterwards we create a new realm.

Do note that the realm must be all UPPERCASE; Kerberos is case and time-clock sensitive.

4.4.3.2. OpenLDAP Installation & Integration with MIT Kerberos 5

Before we can create our principals, OpenLDAP must be installed and integrated with Kerberos.

To do so we must first make sure the BASEDN specified in ***/etc/ldap/ldap.conf*** is the same as our default krb realm.

💡*Example*
*If our default realm is **KRB.MERGRWEB.ME**, then our base dn is:*
dc=krb,dc=mergrweb,dc=me

We then extract the Kerberos schema we acquired from the **krb5-kdc-ldap** package and import it with ldap-schema-manager.

Afterwards we create, set passwords and access control rules to the LDAP entries of the administrative entities Kerberos will use to communicate with OpenLDAP; ***kdc-service*** and ***kadmin-service.***

Finally, we edit our ***/etc/krb5.conf*** to point to and use our newly created OpenLDAP database, create the realm and password stash with ***kdb5_ldap_util***, and give read-write permissions by appending */admin@COMPANY.REALM to **/etc/krb5kdc/kadm5.acl**

Now we have a working Kerberos+OpenLDAP setup, and we can create, manage and delete principals for our Hadoop services.

4.4.4. NGINX & Authentik

The webserver runs a Docker image of Linuxserver's SWAG (Secure Web Application Gateway); this NGINX webserver makes it easy to manage services via its built-in proxy configs, integrated certbot for SSL encryption (using LetsEncrypt or ZeroSSL) and fail2ban services.

Authentik, an Identity Provider that can (and while outside the scope of these scripts, it is in the tested implementation), configured to connect to Kerberos as a federated login source provider with sync and SPNEGO support.

Authentik offers a very clean WebUI for managing users, proxy providers (We setup SWAG as a forward auth proxy provider, and Hadoop as an associated application, both associated to the default embedded Auth Outpost), etc...

As of the writing of this report, Authentik's Kerberos Integration is still in the preview stage and is not stable nor production ready.

For easy management, our services run as Docker containers, initialized and configured with Docker Compose YAML configuration files, due to hardware constraints, we split the Authentik frontend and backend between our webserver and the security server respectively, and we initiate a connection between them via SSH per-port proxy forwarding so that the Authentik-Server container can reach its Redis and PostgreSQL containers.

SWAG offers us a simple way to setup subdomains for our services WebUI, protected via Authentik (which SWAG officially supports and integrates into its configs), simply by pointing to our SSH tunnel's internal Docker bridge network (we made a user-defined Docker network called ***'my-net'***) hostname, the port pointing to our WebUI, and the protocol used (***http*** or ***https***, in our case it is https).

At the end, our network topology would look like this:

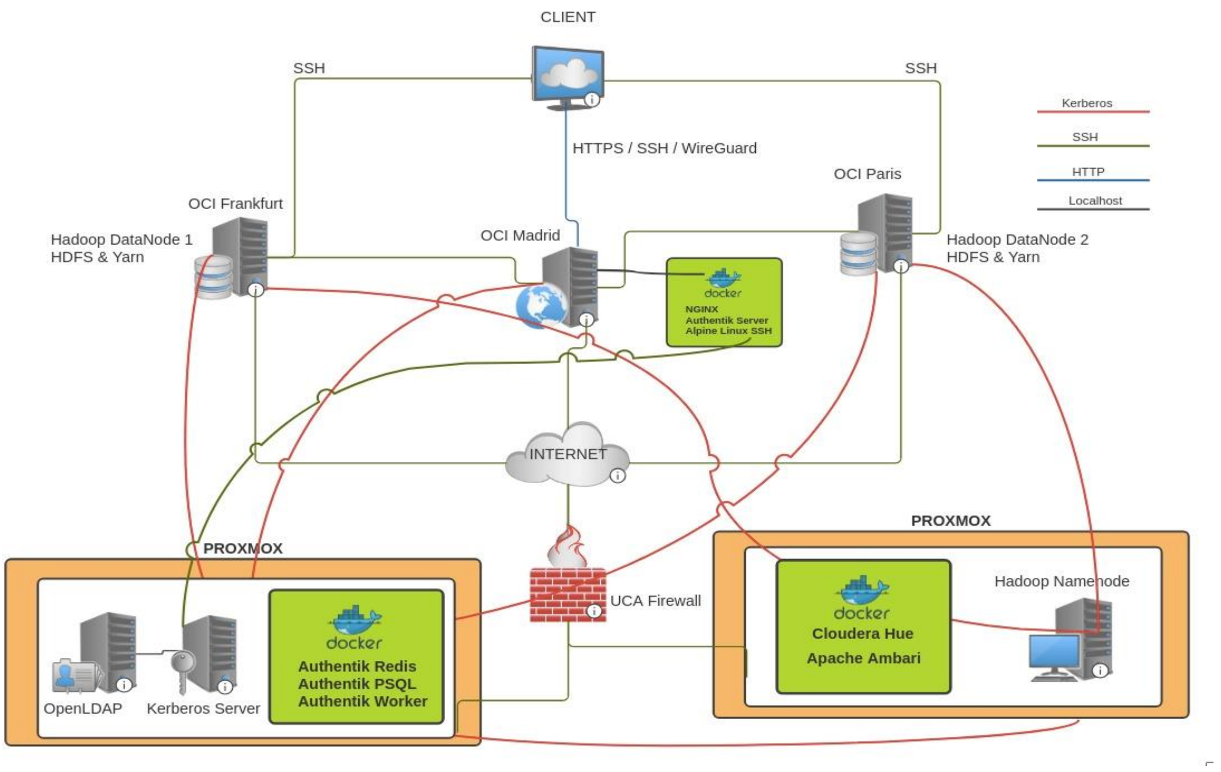


Figure 27. Network topology

Do note that the Kerberos connections are encapsulated via SSH

4.5. Tests and results

We made this special program called ***hdp*** that automates the starting and stopping of the Hadoop cluster

To start the Hadoop cluster, login to the Namenode as hadoopadmin and run this command:

```
hdp start
```

To stop the Hadoop cluster, on the same Namenode as hadoopadmin and run this command:

```
hdp stop
```

hadoopadmin starts/stops the hadoop services by SSH-ing to the nodes, and then starting the daemons as their respective UNIX users via the command ***sudo -u <user> <command>***

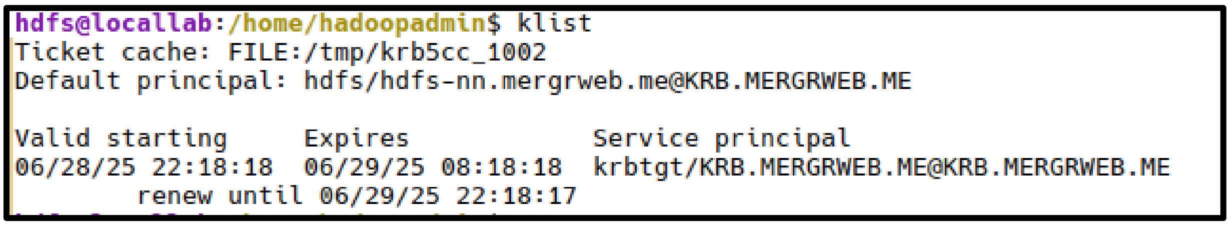
This does means that one must have ~/.ssh/config on every machine in the cluster with aliases pointing to the nodes, including localhost, logging in as user hadoopadmin and their respective authentication method, do note the alias must match what's been set in etc/hadoop/workers file

Whenever a client, be it a person or a UNIX process, attempts to access a Kerberised service. it must first get a valid Ticket Granting Ticket via the command:

```
kinit -kt <keytab file location> <user>/<hostname>@COMPANY.REALM
```

The acquired Ticket Granting Ticket can be listed via the command

```
klist
```

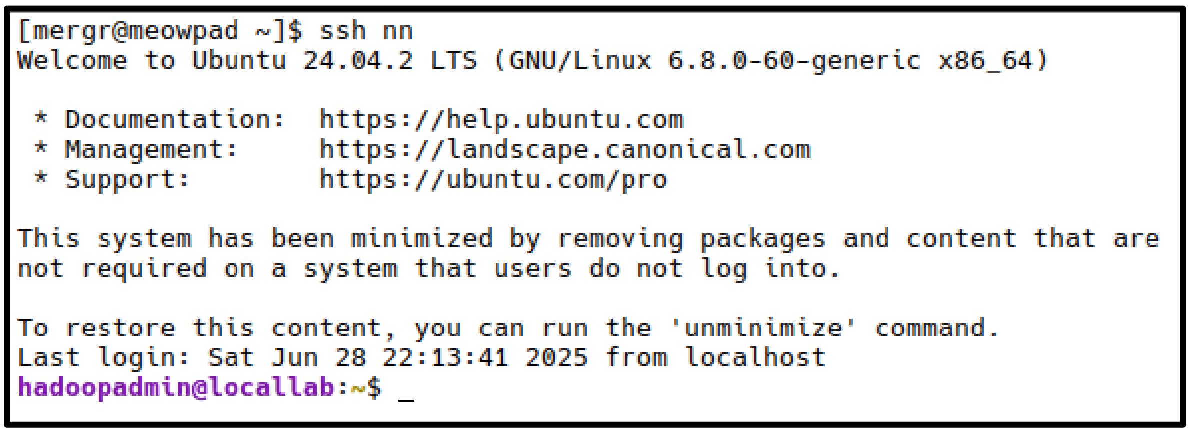


```
hdfs@locallab:/home/hadoopadmin$ klist
Ticket cache: FILE:/tmp/krb5cc_1002
Default principal: hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME

Valid starting     Expires            Service principal
06/28/25 22:18:18  06/29/25 08:18:18  krbtgt/KRB.MERGRWEB.ME@KRB.MERGRWEB.ME
        renew until 06/29/25 22:18:17
```

Of course, we must make sure our client machine has krb5-client installed and configured to point to our KDC server.

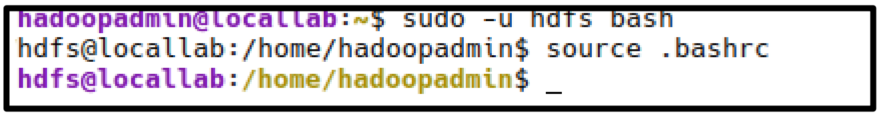
We can now either SSH directly to the kerberized service or via a SPNEGO-enabled web browser, by opening a TCP forwarding port directly to the service's exposed WebUI port bound to the remote's localhost or simply going to https://service.domain.url, which should redirect us to the pre-configured Authentik, which will handle authentication and forwarding of our Kerberos ticket to the service.

Here we SSH into the name node, we already have Hadoop running



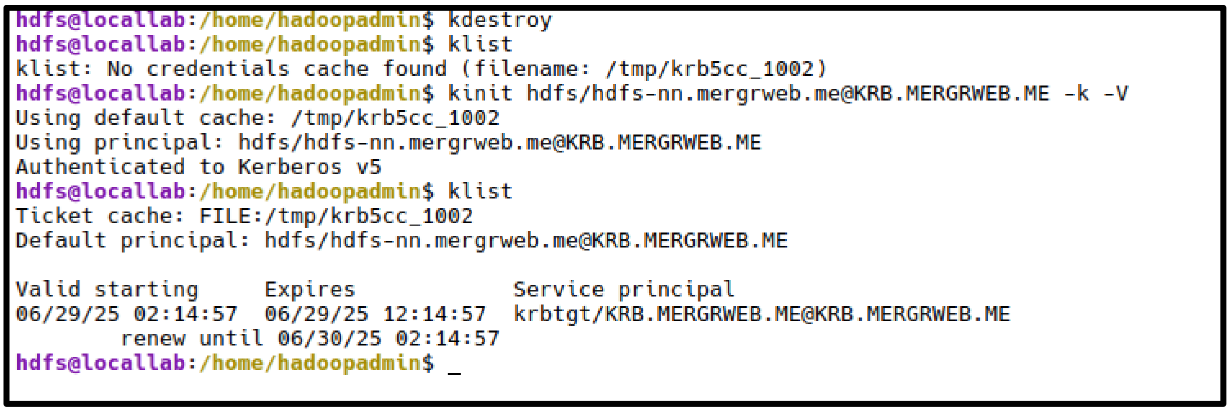
```
[mergr@meowpad ~]$ ssh nn
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-60-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Jun 28 22:13:41 2025 from localhost
hadoopadmin@locallab:~$ _
```

We switch to ***hdfs*** to able to run and have read-write permissions on hdfs-related files



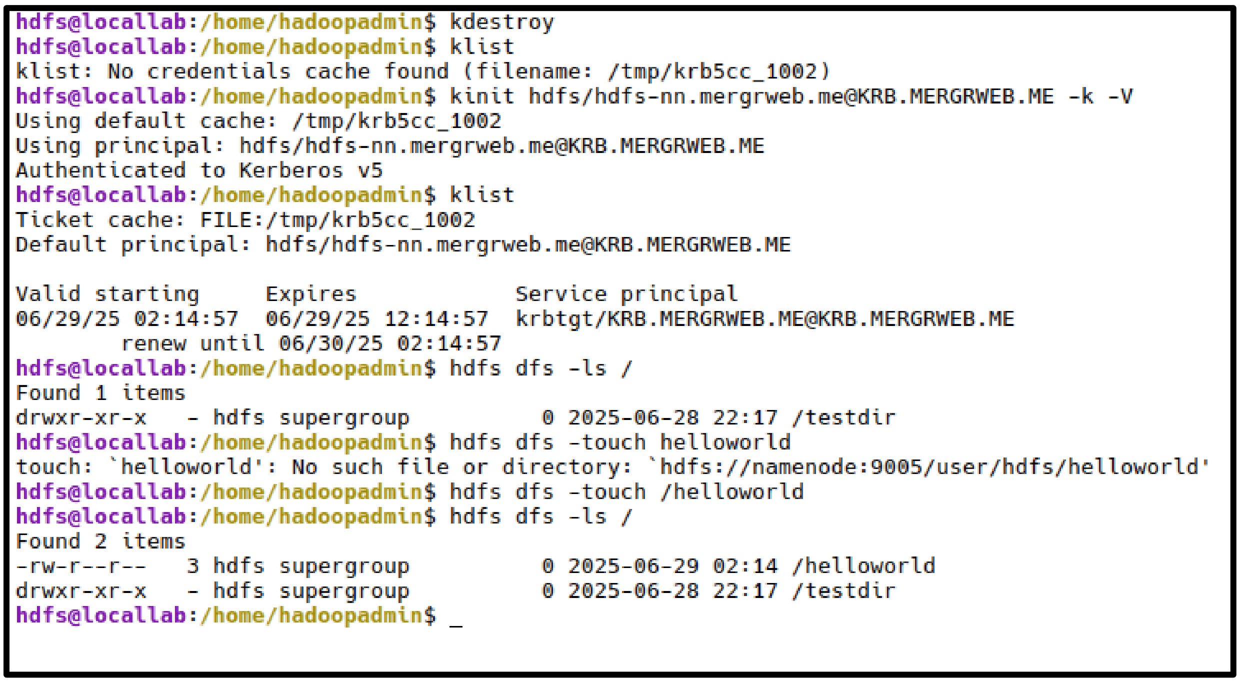
```
hadoopadmin@locallab:~$ sudo -u hdfs bash
hdfs@locallab:/home/hadoopadmin$ source .bashrc
hdfs@locallab:/home/hadoopadmin$ _
```

As a test, we destroy the Kerberos ticket and get a fresh one from the KDC server



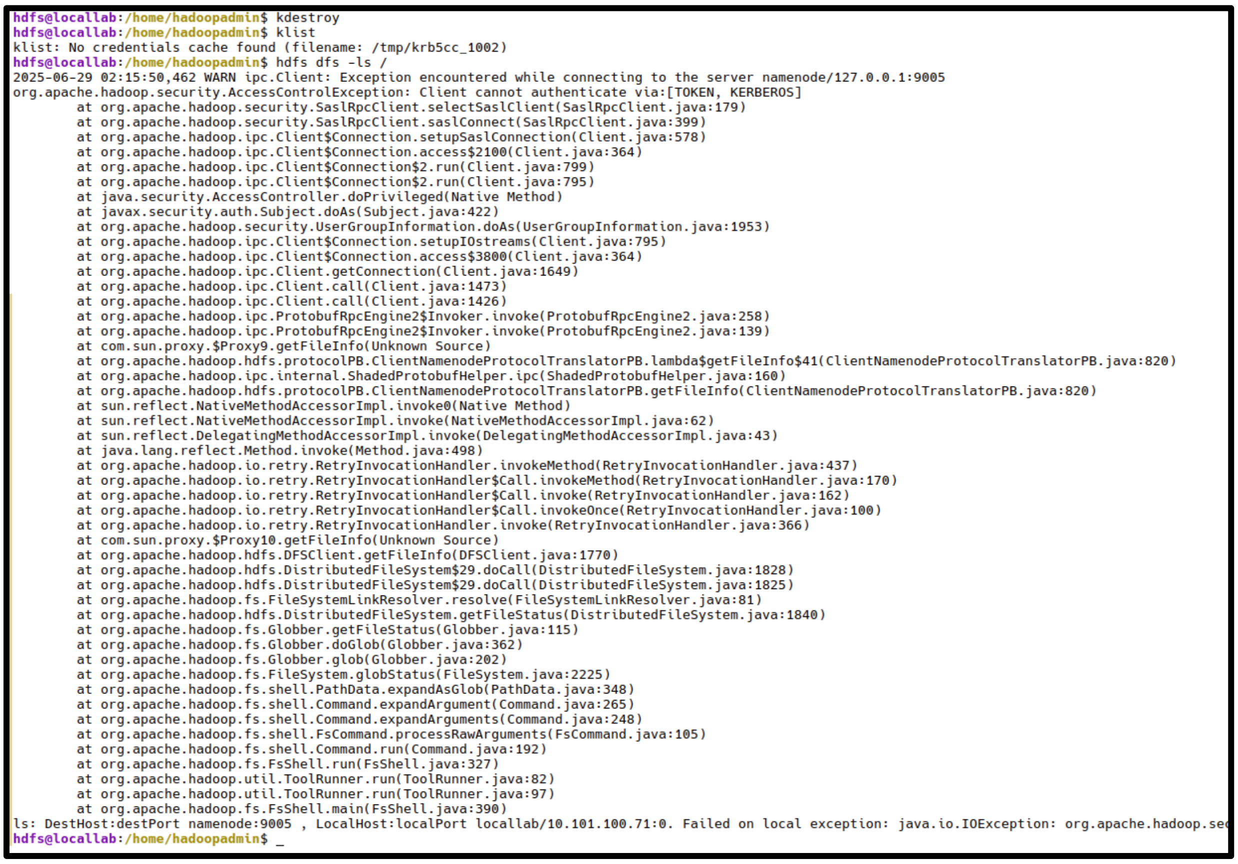
```
hdfs@locallab:/home/hadoopadmin$ kdestroy
hdfs@locallab:/home/hadoopadmin$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1002)
hdfs@locallab:/home/hadoopadmin$ kinit hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME -k -V
Using default cache: /tmp/krb5cc_1002
Using principal: hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME
Authenticated to Kerberos v5
hdfs@locallab:/home/hadoopadmin$ klist
Ticket cache: FILE:/tmp/krb5cc_1002
Default principal: hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME

Valid starting     Expires            Service principal
06/29/25 02:14:57  06/29/25 12:14:57  krbtgt/KRB.MERGRWEB.ME@KRB.MERGRWEB.ME
        renew until 06/30/25 02:14:57
hdfs@locallab:/home/hadoopadmin$ _
```

Here is a successful login and manipulation of the hdfs after successfully authenticating with our ticket, we also note that the created file we just made is of group supergroup. We did not make this group in the Linux shell, but rather, we created and specified it on the OpenLDAP directory.



```
hdfs@locallab:/home/hadoopadmin$ kdestroy
hdfs@locallab:/home/hadoopadmin$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1002)
hdfs@locallab:/home/hadoopadmin$ kinit hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME -k -V
Using default cache: /tmp/krb5cc_1002
Using principal: hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME
Authenticated to Kerberos v5
hdfs@locallab:/home/hadoopadmin$ klist
Ticket cache: FILE:/tmp/krb5cc_1002
Default principal: hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME

Valid starting     Expires            Service principal
06/29/25 02:14:57  06/29/25 12:14:57  krbtgt/KRB.MERGRWEB.ME@KRB.MERGRWEB.ME
        renew until 06/30/25 02:14:57
hdfs@locallab:/home/hadoopadmin$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - hdfs supergroup          0 2025-06-28 22:17 /testdir
hdfs@locallab:/home/hadoopadmin$ hdfs dfs -touch helloworld
touch: `helloworld': No such file or directory: `hdfs://namenode:9005/user/hdfs/helloworld'
hdfs@locallab:/home/hadoopadmin$ hdfs dfs -touch /helloworld
hdfs@locallab:/home/hadoopadmin$ hdfs dfs -ls /
Found 2 items
-rw-r--r--   3 hdfs supergroup          0 2025-06-29 02:14 /helloworld
drwxr-xr-x   - hdfs supergroup          0 2025-06-28 22:17 /testdir
hdfs@locallab:/home/hadoopadmin$ _
```

Here is the case when we do not have a valid Kerberos ticket and attempt to login to Hadoop services:



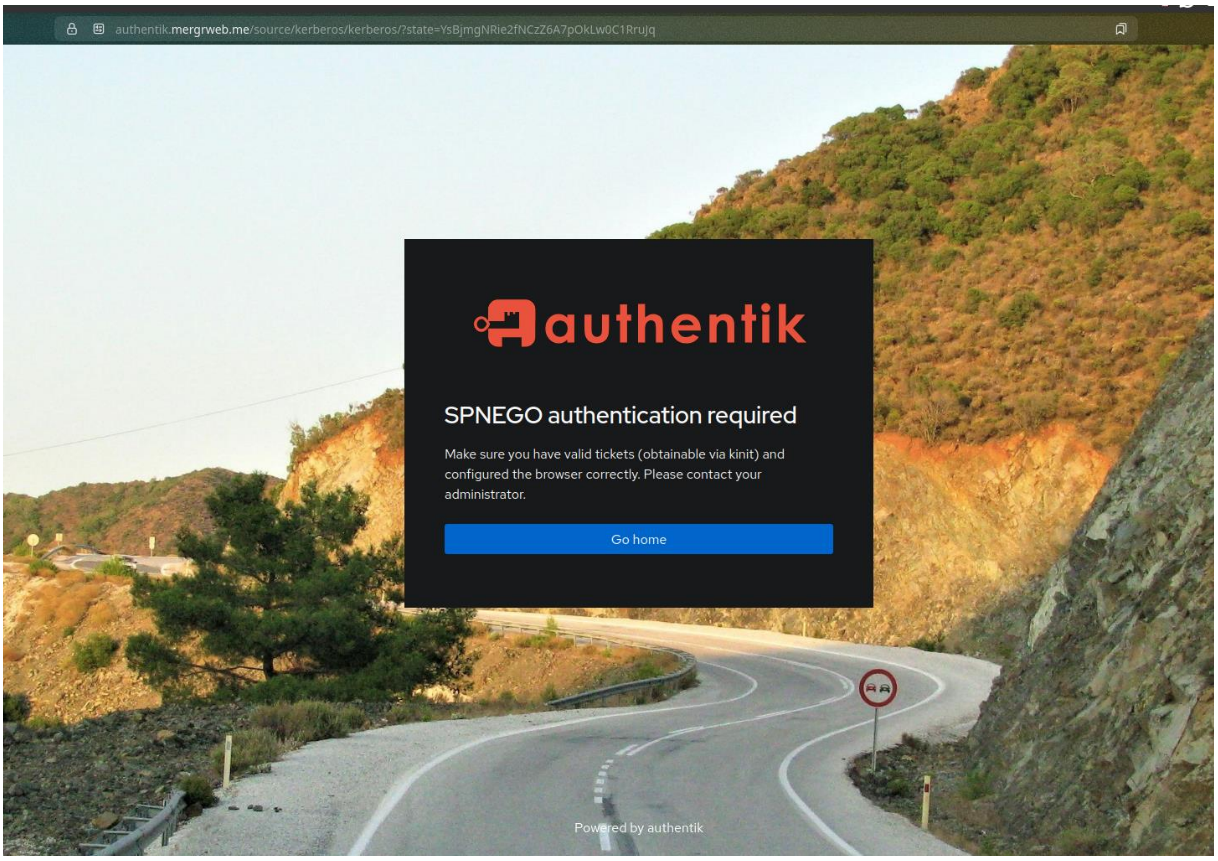
```
hdfs@locallab:/home/hadoopadmin$ kdestroy
hdfs@locallab:/home/hadoopadmin$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1002)
hdfs@locallab:/home/hadoopadmin$ hdfs dfs -ls /
2025-06-29 02:15:50,462 WARN ipc.Client: Exception encountered while connecting to the server namenode/127.0.0.1:9005
org.apache.hadoop.security.AccessControlException: Client cannot authenticate via:[TOKEN, KERBEROS]
        at org.apache.hadoop.security.SaslRpcClient.selectSaslClient(SaslRpcClient.java:179)
        at org.apache.hadoop.security.SaslRpcClient.saslConnect(SaslRpcClient.java:399)
        at org.apache.hadoop.ipc.Client$Connection.setupSaslConnection(Client.java:578)
        at org.apache.hadoop.ipc.Client$Connection.access$2100(Client.java:364)
        at org.apache.hadoop.ipc.Client$Connection$2.run(Client.java:799)
        at org.apache.hadoop.ipc.Client$Connection$2.run(Client.java:795)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1953)
        at org.apache.hadoop.ipc.Client$Connection.setupIOstreams(Client.java:795)
        at org.apache.hadoop.ipc.Client$Connection.access$3800(Client.java:364)
        at org.apache.hadoop.ipc.Client.getConnection(Client.java:1649)
        at org.apache.hadoop.ipc.Client.call(Client.java:1473)
        at org.apache.hadoop.ipc.Client.call(Client.java:1426)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Invoker.invoke(ProtobufRpcEngine2.java:258)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Invoker.invoke(ProtobufRpcEngine2.java:139)
        at com.sun.proxy.$Proxy9.getFileInfo(Unknown Source)
        at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.lambda$getFileInfo$41(ClientNamenodeProtocolTranslatorPB.java:820)
        at org.apache.hadoop.ipc.internal.ShadedProtobufHelper.ipc(ShadedProtobufHelper.java:160)
        at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.getFileInfo(ClientNamenodeProtocolTranslatorPB.java:820)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInvocationHandler.java:437)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeMethod(RetryInvocationHandler.java:170)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invoke(RetryInvocationHandler.java:162)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeOnce(RetryInvocationHandler.java:100)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:366)
        at com.sun.proxy.$Proxy10.getFileInfo(Unknown Source)
        at org.apache.hadoop.hdfs.DFSClient.getFileInfo(DFSClient.java:1770)
        at org.apache.hadoop.hdfs.DistributedFileSystem$29.doCall(DistributedFileSystem.java:1828)
        at org.apache.hadoop.hdfs.DistributedFileSystem$29.doCall(DistributedFileSystem.java:1825)
        at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
        at org.apache.hadoop.hdfs.DistributedFileSystem.getFileStatus(DistributedFileSystem.java:1840)
        at org.apache.hadoop.fs.Globber.getFileStatus(Globber.java:115)
        at org.apache.hadoop.fs.Globber.doGlob(Globber.java:362)
        at org.apache.hadoop.fs.Globber.glob(Globber.java:202)
        at org.apache.hadoop.fs.FileSystem.globStatus(FileSystem.java:2225)
        at org.apache.hadoop.fs.shell.PathData.expandAsGlob(PathData.java:348)
        at org.apache.hadoop.fs.shell.Command.expandArgument(Command.java:265)
        at org.apache.hadoop.fs.shell.Command.expandArguments(Command.java:248)
        at org.apache.hadoop.fs.shell.FsCommand.processRawArguments(FsCommand.java:105)
        at org.apache.hadoop.fs.shell.Command.run(Command.java:192)
        at org.apache.hadoop.fs.FsShell.run(FsShell.java:327)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:82)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:97)
        at org.apache.hadoop.fs.FsShell.main(FsShell.java:390)
ls: DestHost:destPort namenode:9005 , LocalHost:localPort locallab/10.101.100.71:0. Failed on local exception: java.io.IOException: org.apache.hadoop.se
hdfs@locallab:/home/hadoopadmin$ _
```

As the error logs suggest, Hadoop expects a valid Kerberos ticket (in this case a ticket for ***hdfs/hdfs-nn.mergrweb.me@KRB.MERGRWEB.ME***).

Likewise, we can login to our WebUI using a SPNEGO-enabled browser by typing in our NGINX proxied subdomain.

NGINX intercepts our subdomain and redirects us to Authentik appending the service we are attempting to access to the request sent to our Identity Provider, and which, after a proper configuration, validates our TGT with the KDC server, and then forwards it to our Hadoop WebUI for the Kerberos user login.

Naturally, our client machine must have a valid TGT. Otherwise, Authentik will refuse access as shown here:

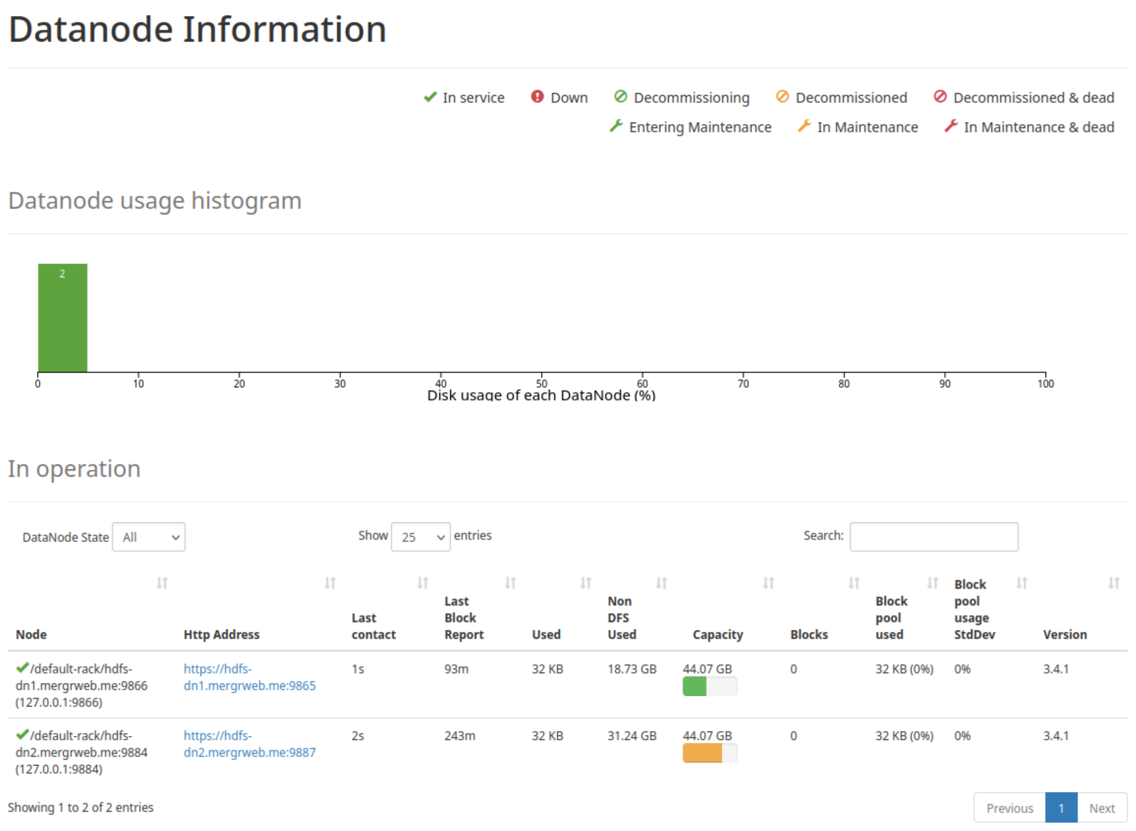

authentik.mergrweb.me/source/kerberos/kerberos/?state=YsBjmgNRie2fNCzZ6A7pOkLw0C1RruJq

SPNEGO authentication required

Make sure you have valid tickets (obtainable via kinit) and configured the browser correctly. Please contact your administrator.

Figure 28. Authentik bad login page

After a successful login, we now have access to (in this case) our Namenode WebUI, where we can manage our HDFS, read logs and monitor our datanodes



Datanode Information

✓ In service ● Down ⊘ Decommissioning ⊘ Decommissioned ⊘ Decommissioned & dead
Entering Maintenance In Maintenance In Maintenance & dead

Datanode usage histogram

In operation

DataNode State All Show 25 entries Search:

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Block pool usage StdDev	Version
✓/default-rack/hdfs-dn1.mergrweb.me:9866 (127.0.0.1:9866)	https://hdfs-dn1.mergrweb.me:9865	1s	93m	32 KB	18.73 GB	44.07 GB	0	32 KB (0%)	0%	3.4.1
✓/default-rack/hdfs-dn2.mergrweb.me:9884 (127.0.0.1:9884)	https://hdfs-dn2.mergrweb.me:9887	2s	243m	32 KB	31.24 GB	44.07 GB	0	32 KB (0%)	0%	3.4.1

Showing 1 to 2 of 2 entries

Figure 29. Namenode WebUI's Datanode Information page

4.6. Future work

While we are proud of what's been accomplished so far, we cannot deny the existence of limitations and possible improvements:

- Switching from a default Hadoop cluster to an Ambari-managed Hadoop cluster.
- SSH explicit per-port TCP forwarding can get bogged down by the lowest link (50Mbps OCI VPS) causing slow network performance and potential connectivity issues with services that expect UDP traffic that the UCA Firewall blocks. Switching to a UDP VPN protocol like Wireguard will increase network performance and massively simplify the network configuration.
- An automatic browser SPNEGO configuration script.

Due to time, hardware constraints and complexity of the network, we could not achieve all what we desired with this project, although the footwork and foundations are there, should these features ever be finalized.

4.7. Conclusion

The successful implementation of Kerberos within the Hadoop ecosystem reinforces the importance of strong authentication in distributed systems. This chapter demonstrated how Kerberos can be integrated into Hadoop to secure interactions between users and services by preventing unauthorized access and identity spoofing. The configuration of the KDC, service principals, and secure communication pathways establishes a trusted environment where sensitive data can be processed with greater assurance.

General Conclusion

The implementation of a Kerberos authentication system within the Hadoop ecosystem addresses a critical need for robust security in big data environments. While Hadoop offers powerful tools for distributed storage and processing, its default configuration lacks adequate mechanisms for user authentication and access control. By integrating Kerberos, this project enhances the security posture of the Hadoop cluster, ensuring that only authorized users and services can interact with sensitive data and system components.

Through the setup and configuration of the Kerberos Key Distribution Center (KDC), the secure modification of Hadoop services, and the resolution of integration challenges, the project demonstrates a practical approach to securing big data infrastructures. The result is a more trustworthy and compliant system capable of operating in environments where data confidentiality and integrity are paramount.

This work highlights the importance of embedding security by design in big data platforms and provides a foundation for future enhancements, such as fine-grained authorization, encryption, and auditing. It contributes to building scalable, secure, and resilient architectures essential for today’s data-driven organizations.

References

- Big Data Hadoop Interview Guide Get answers to the most frequently asked questions in a Hadoop interview (English Edition) (Narayanan, Vishwanathan)
- Big Data, MapReduce, Hadoop, and Spark with Python (Lazy Programmer)
- Expert Hadoop® Administration_ Managing, Tuning, and Securing Spark, YARN, and HDFS - Expert Hadoop Administration Managing, Tuning, and Securing Spark, YARN, and HDFS (Sam R. Alapati)
- https://www.techtarget.com/searchdatamanagement/feature/15-big-data-tools-and-technologies-to-know-about (Article)
- https://eds.fr/2025/01/23/comprendre-le-big-data-enjeux-et-applications-dans-le-monde-moderne _ (Article)
- https://spectralops.io/blog/big-data-security
- https://kyligence.io/blog/use-python-for-data-science-with-apache-kylin/
- https://www.datacamp.com/fr/blog/hadoop-architecture
- https://ambari.apache.org/docs/3.0.0/introduction
- https://deepwiki.com/apache/ambari/5-cluster-management
- https://community.ibm.com/community/user/blogs/archive-user/2016/08/28/ibm-spectrum-scale-security-hadoop-security
- https://hdpweb.o.onslip.net/HDPDocuments/HDP3/HDP-3.0.1/configuring-hdfs-encryption/content/hdfs_encryption_overview.html
- https://documentation.ubuntu.com/server/how-to/openldap/install-openldap/#install-openldap
- https://documentation.ubuntu.com/server/how-to/kerberos/install-a-kerberos-server/
- https://documentation.ubuntu.com/server/how-to/kerberos/kerberos-with-openldap-backend/
- http://techpubs.spinlocksolutions.com/dklar/ldap.html
- http://techpubs.spinlocksolutions.com/dklar/kerberos.html
- https://docs.goauthentik.io/docs/