TDP Documentation

2.2

English

© Copyright Tecnisys 2026. All rights reserved.

Table of Contents

PART I - INTRODUCTION

Overview

Concepts

Apache Airflow

Apache Ambari

Apache Atlas

Delta Lake

Apache Druid

Apache Hadoop

Apache HDFS

Apache MapReduce

Apache YARN

Apache Flink

Great Expectations

Apache HBase

Apache Hive

Apache Iceberg

Apache Kafka

Kerberos

Apache Knox

Apache Livy

Apache NiFi

Apache Ozone

Apache Ranger

Apache Ranger KMS

Apache Solr

Apache Spark

Apache Sqoop

Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

Trino

Apache Zeppelin

Apache Zookeeper

PART II - QUICK START

TDP Sandbox

PART III - INSTALLATION

Minimum Requirements

Support Tools

Environment Preparation

Installation Packages

Apache Ambari Installation

Cluster Creation and Component Installation

PART IV - UPDATE

Update Flowchart

Update Prerequisites

Apache Ambari Update

New Version Registration

Components Update

PART V - COMPONENTS

Version Matrix

PART VI - TECHNICAL NOTES

Version Highlights

Added Components

Removed Components

Fixes

Fix 202408001

Fix 202408002

© Copyright Tecnisys 2026. All rights reserved.

More Information about each TDP Platform component

PART VII - ROADMAP

Roadmap

PART VIII - OTHER INFORMATION

Support

Legal Information

© Copyright Tecnisys 2026. All rights reserved.

PART I - INTRODUCTION

© Copyright Tecnisys 2026. All rights reserved.

Overview

Discover what TDP is and how it can help your organization develop data-driven business models.

What is TDP?

The Tecnisys Data Platform (TDP) is Tecnisys' Data Platform for data analysis and governance. Our goal is to assist organizations in developing data-driven business models.

TDP consists only of open-source projects, does not require a license or active subscription for full operation, and offers the main open-source components of a Modern Data Stack, such as Apache Spark, Apache Iceberg, Apache Kafka, Apache Flink, Apache Druid, Apache NiFi, Apache Airflow, Apache Superset, and many others.

Figure 1 - Key Components

With TDP, it is possible to move from ingestion to data analysis, creating integrated pipelines and continuous data flows for different types of workloads, whether streaming

© Copyright Tecnisys 2026. All rights reserved.

or batch. The platform allows for quick exploration and analysis of large data volumes, training machine learning models, presenting dynamic graphs, and generating strategic knowledge for decision-making. All this with end-to-end data security, auditing, and governance.

Visit our website and sign up to download this amazing Data Platform for free.

Why use TDP?

TDP enables the implementation of a complete and integrated Data Environment in just a few guided steps. Additionally, the platform features a full graphical interface for centralized management of all machines and cluster services.

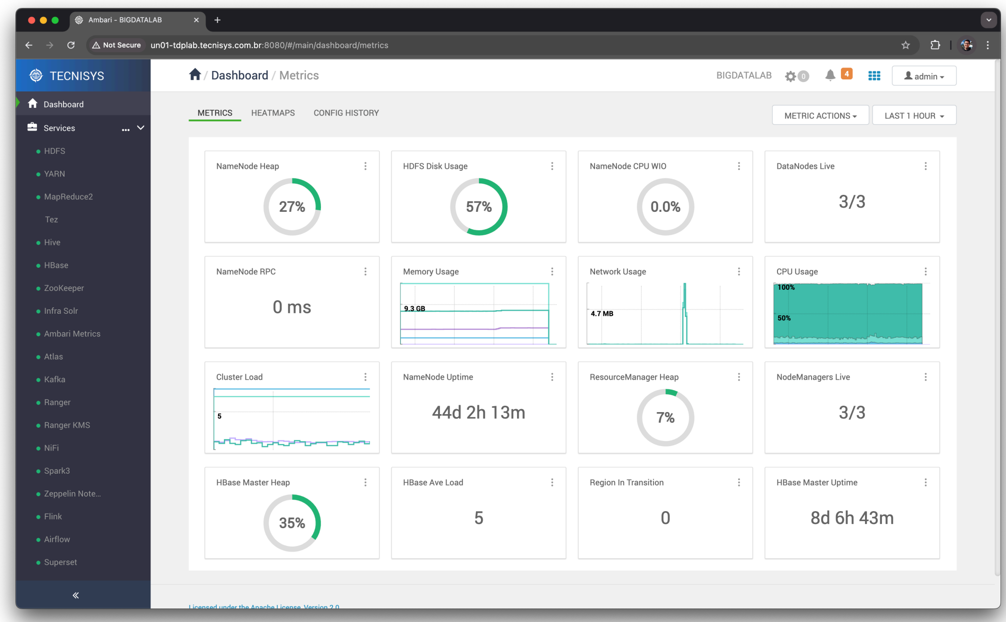


Figure 2 - Graphical interface for managing machines and services

Among the solutions provided by the TDP Platform, the following stand out:

- Centralized administration
- Distributed storage and processing
- Decentralized data analysis and machine learning
- Data virtualization
- Automated data pipelines
- Flexible data flows
- Collaborative development
- Text search and indexing

© Copyright Tecnisys 2026. All rights reserved.

- Data bus
- Advanced security
- Data governance
- Dynamic data visualization

TDP is a constantly evolving Brazilian Data Platform, developed by a team of software engineers, data engineers, and specialists attentive to organizational needs.

© Copyright Tecnisys 2026. All rights reserved.

CONCEPTS

© Copyright Tecnisys 2026. All rights reserved.

Apache Airflow

Pipeline Orchestration

A data pipeline consists of a set of tasks or actions that need to be executed in a certain order or logical sequence (Workflow) to achieve a desired result.

Data pipelines can be represented as DAGs - _Directed Acyclic Graph - which consist of blocks that follow a sequence, as the previous one is executed, allowing forking at various points, without the possibility of returning to the starting point. In DAGs, it's possible to define operators (which become tasks) and relationships between operators programmatically.

Although many Pipeline Orchestrators have been developed over the years to execute tasks (DAGs), Airflow has several important features that make it particularly suitable for implementing efficient and batch-oriented data pipelines.

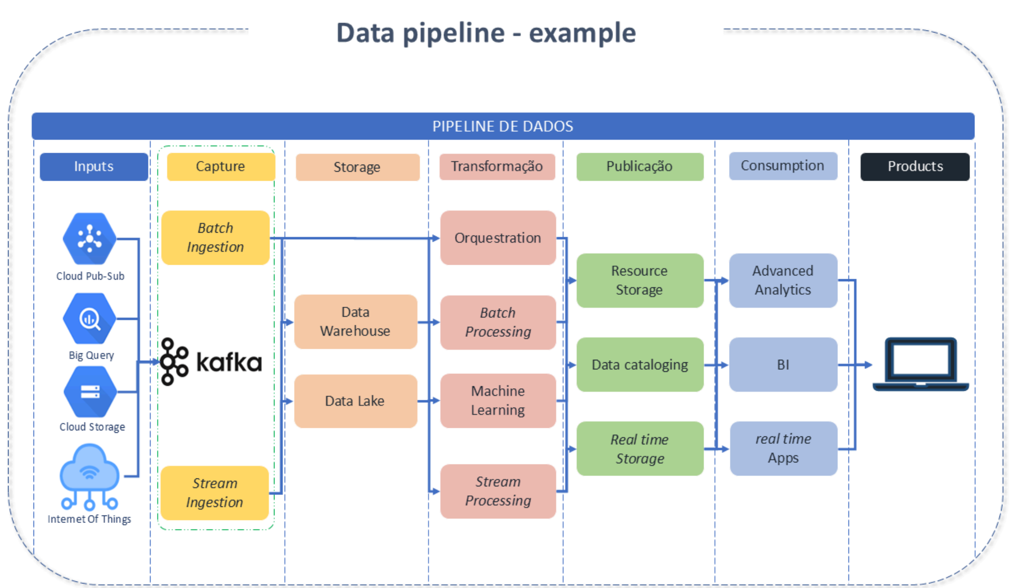


Figure 1 - Data Pipeline

© Copyright Tecnisys 2026. All rights reserved.

Apache Airflow Features:

Apache Airflow was created at Airbnb in 2014 as a solution to manage their complex workflows. From the beginning, the project has been open-source, becoming an Apache incubator project in 2017 and a top-level Apache project in 2019.

It is a feature-rich tool and has a set of fundamental resources for a Big Data solution:

- **Versatility:** The ability to implement pipelines using Python code allows the creation of complex pipelines with anything compatible with Python.
- **Easy Integration:** Airflow's Python base makes it easy to extend and add integrations with many different systems. The Airflow community has already developed a rich collection of extensions for different databases, cloud services, etc.
- **Rich programming semantics:** It allows the execution of pipelines at regular intervals and the creation of efficient pipelines using incremental processing to avoid costly recalculations of existing results.
- **Backfilling:** Allows easy reprocessing of historical data and recalculation of any derived datasets after code changes.
- **Rich Web Interface:** Airflow's rich web interface provides easy visualization for monitoring pipeline execution results and debugging any failures.
- **Open Source:** Ensures job creation without any vendor dependency.

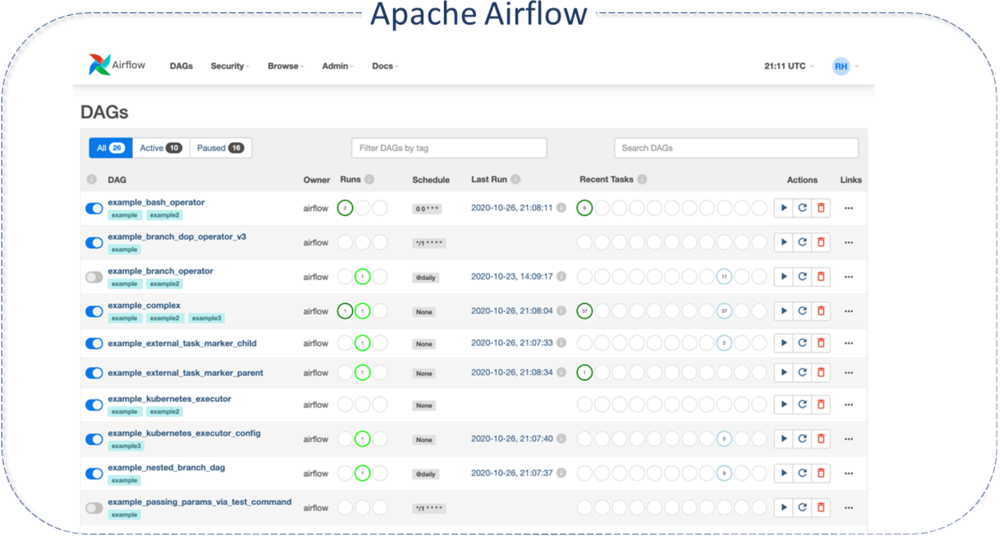


Figure 2 - Apache Airflow Interface

Apache Airflow Architecture

Airflow is organized from the following main components:

© Copyright Tecnisys 2026. All rights reserved.

- **DAGs - _Directed Acyclic Graph: The DAG is the main concept in Airflow. It represents a workflow (a collection of individual tasks , organized with their respective dependencies and data flows - the tasks themselves describe what will be done, such as data fetching, analysis, triggering another app, etc.).

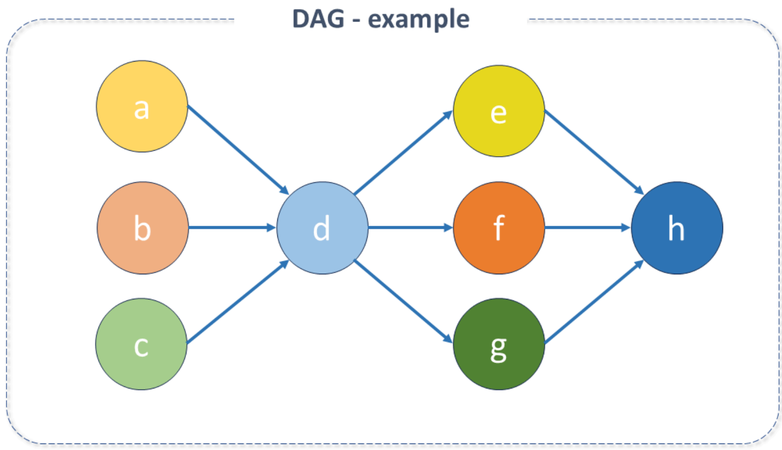


Figure 3 - Example of the structure of a DAG

The DAG doesn't care about what happens inside a task, but rather how to execute it (the order of execution, how many times to repeat them, if there is a timeout, etc.). The DAG structure is composed of the declaration of dependencies between tasks.

Besides that, DAG files contain some additional metadata about the DAG, informing Airflow how and when it should be executed. This programmatic approach offers a lot of flexibility to create DAGs and allows customization in the way pipelines are created.

DAGs do not require scheduling, but it is very common for them to be defined, which is done by the Scheduler A DAG is executed in two ways: triggered manually or through an API or on a defined schedule, as part of the DAG.

Its most important parameters are:

- **dag_id:** DAG identifier
- **start date:***timestamp* from which the Scheduler will schedule the DAG.
- **schedule interval**: DAG execution interval.
- **default args**: Dictionary with defaults that should be passed to the tasks.

© Copyright Tecnisys 2026. All rights reserved.

- **Scheduler**: The Scheduler handles the triggering of scheduled workflows and the submission of tasks to the Executor.
- **Webserver**: Presents a user interface that allows for the visualization of DAGs and tasks, their inspection, triggering, debugging, and analysis of their results.
- **Executor**: A process that handles tasks in execution. In a standard installation, it runs within the Scheduler. However, most production-suitable Executors send task execution to Workers.

  Most Executors will introduce other components to communicate with their Workers (such as the task queue). However, we can understand the Executor and its Workers as a single logical component handling task execution.

  The Executor is configurable, and depending on the requirements, you can choose from a few options:

  - **Local**:
    - **Sequential Executor**: This is the default executor. It will execute one task instance at a time.
    - **Local Executor**: Executes tasks by generating processes in a controlled manner in different modes. Since the BaseExecutor has the option to receive a parallelism parameter to limit the number of generated processes, when this parameter is 0, the number of processes that the Local Executor can generate becomes unlimited.
  - **Remote**:
    - **Dask Executor**: Allows the execution of Airflow tasks in a Dask Distributed Cluster. Dask Clusters can run on a single machine or remote networks.
    - **Celery Executor**: It is one way to scale the number of workers.
    - **Kubernetes Executor**: Introduced in Apache Airflow 1.10.0, it allows Airflow to be scaled very easily as tasks are executed in Kubernetes.
    - **Celery Kubernetes Executor**: Allows simultaneous execution of Celery Executor and Kubernetes Executor. It inherits the scalability of the Celery

© Copyright Tecnisys 2026. All rights reserved.

Executor to handle high loads during peak hours and runtime isolation of the KubernetesExecutor.

- **Workers**: These are separate processes that execute scheduled tasks.
- **DAG files**: Store the files that are read by the Scheduler and Executor (and by the Workers that the Executor has).
- **Metadata Database**: SQL database used by the Scheduler, Executor, and Webserver to store metadata about the data pipelines being executed.

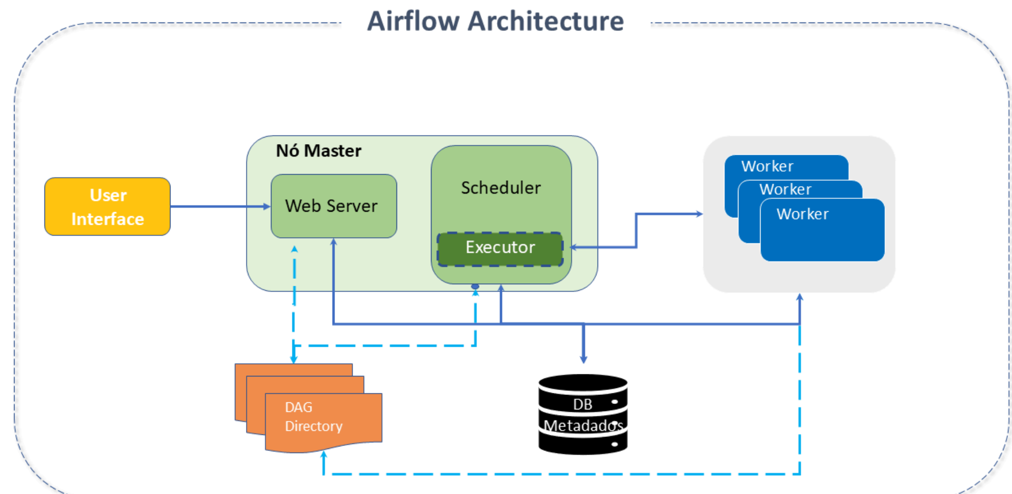


Figure 4 - Airflow Arquiteture

Need for a Shared Directory for DAGs in Airflow

In a distributed Apache Airflow environment, especially when using the Celery Executor, Kubernetes Executor, or Dask Executor, it is essential to ensure that all Workers have consistent access to the same DAG files.

DAGs are Python scripts that define Airflow task workflows.

When there are multiple Workers executing tasks in parallel, all of them must be able to read the same set of DAGs.

By default, the Scheduler and the Webserver read the files directly from the directory: `/usr/tdp/current/airflow/dags`.

© Copyright Tecnisys 2026. All rights reserved.

However, if this directory is not shared among the nodes, each Worker will only see its local DAGs — resulting in execution failures or DAGs not appearing in the Web UI.

Shared Directory via NFS

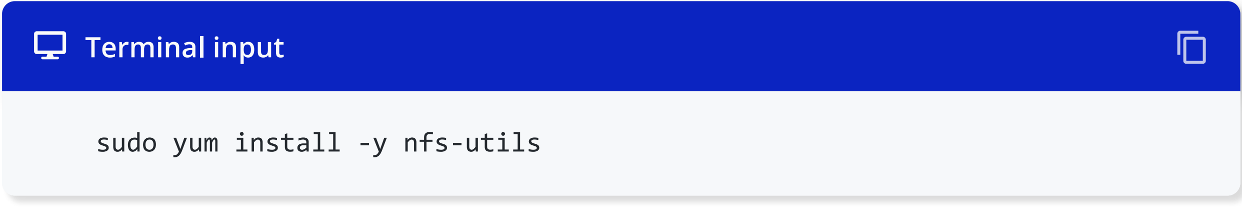
The simplest and most TDP-compatible (Tecnisys Data Platform) approach is to mount an NFS (Network File System) exporting the directory `/usr/tdp/current/airflow/dags` from the primary node (usually where the Airflow Scheduler is installed) and mounting it on the other hosts running the Airflow Worker.

This way, all Airflow components — Webserver, Scheduler, and Workers — access exactly the same DAG repository.

Configuration Example

1. Install the NFS server and export the directory:

   1.1 Install NFS server – On the NFS server (example: tdp-mn01):



Terminal input

```
sudo yum install -y nfs-utils
```

   1.2 Create the DAGs directory if it does not exist:

Terminal input

```
sudo mkdir -p /usr/tdp/current/airflow/dags
```

   1.3 Set appropriate permissions:

Terminal input

```
sudo chown -R airflow:airflow /usr/tdp/current/airflow/dags
sudo chmod 755 /usr/tdp/current/airflow/dags
```

2. Edit the `/etc/exports` file and add the line below (adjust the network according to your environment):

© Copyright Tecnisys 2026. All rights reserved.

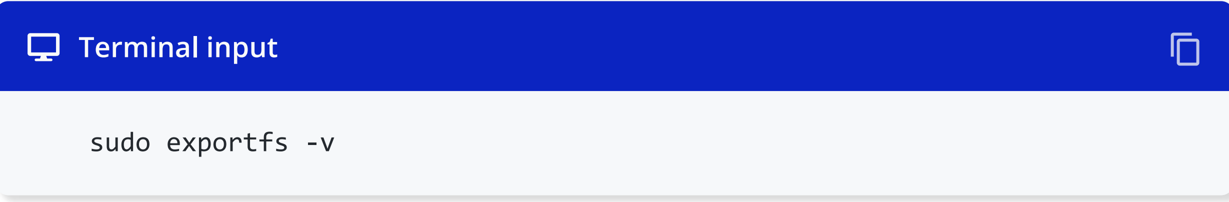
2.1 File edit:

```
/usr/tdp/current/airflow/dags  10.0.0.0/24(rw,sync,no_subtree_check)
```

2.2 Export and enable the service:

```
sudo exportfs -arv
sudo systemctl enable nfs-server
sudo systemctl start nfs-server
```

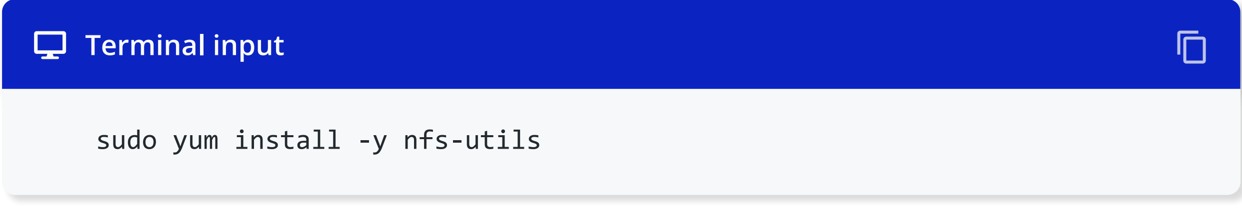
2.3 Verify that the share is active:



```
sudo exportfs -v
```

3. On the Airflow Worker nodes (example: tdp-wn01, tdp-wn02):

3.1 Install the NFS client and mount the shared directory:



```
sudo yum install -y nfs-utils
```

3.2 Create the destination directory (if it does not exist):

```
sudo mkdir -p /usr/tdp/current/airflow/dags
```

3.3 Mount the NFS directory:

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
    sudo mount -t nfs tdp-mn01:/usr/tdp/current/airflow/dags
/usr/tdp/current/airflow/dags
```

3.4 To ensure persistence after reboot, add the entry to `/etc/fstab`:

Terminal input

```
    tdp-mn01:/usr/tdp/current/airflow/dags /usr/tdp/current/airflow/dags nfs
defaults 0 0
```

Apache Airflow Resources

- Pooling: Pooling is an additional feature that provides resource management mechanisms. Pools limit the execution of parallelism on the resource when many processes demand it at the same time, avoiding overload.

- Queuing: All tasks go to the default queue. It is possible to define queues and workers to consume tasks from one or more queues. Queues are especially useful when some tasks need to be executed in a specific environment or resource.

- Plugins: Airflow has a simple plugin manager that can integrate external resources into its core by simply "dropping" files into the $Airflow_home/plugins folder. Python modules in the plugins folder are imported, and web macros and views are integrated into Airflow's main collections and become available for use.

Important Concepts:

- Dag Runs: Every time a DAG is executed, a new instance of it is created, called a Dag-Run by Airflow. DAG runs can be executed in parallel for the same DAG, and each one has a defined data interval that identifies the data period and which tasks it should operate on.

- Tasks: Although from the user's point of view, tasks and operators are equivalent, in Airflow, there are Tasks components that manage the operating state of the operators (define a unit of work within a DAG through the operators). They are represented as a node in the DAG.

© Copyright Tecnisys 2026. All rights reserved.

There are three common types of tasks:

- **Operators**: Conceptually, they are a template for predefined tasks that can be declared within the DAG. They are the components specialized in executing a single, specific task within the workflow. It is a step in the workflow and is usually (not always) atomic, not carrying information from previous operators. In this way, it has autonomy. As they are the ones that actually execute the tasks, both terms are often used interchangeably. Airflow has an extensive set of Operators available, some integrated into the core or pre-installed. The most popular are:

    - **PythonOperator**: Calls a Python function.

    - **EmailOperator**: Sends an email.

    - **BashOperator**: Executes a Bash script, command, or set of commands.

- **Sensors**: A special subclass of Operators that "waits" for an external event.

- **Taskflow-decorated @task**: A custom Python function packaged as a task. It makes creating DAGs much easier for those who use simple Python code instead of Operators to write DAGs.

> NOTE
>
> - If two operators need to share information, they can be combined into a single operator. If this is not possible, there is a cross-communication feature called *xcom*.
> - Operators do not need to be assigned to DAGs immediately (but once assigned, they cannot be transferred or unassigned).

:::

- **Task instances**: Represent the state of a task -> at what stage of the lifecycle the task is in. It instantiates a task - which has been assigned to a DAG and has a state associated with a specific execution. The possible states are:

    - **None**: The task has not been added to the execution queue (scheduled) because its dependencies have not yet been met.

© Copyright Tecnisys 2026. All rights reserved.

- **Scheduled**: The dependencies have been met, and the task can be executed.
- **Queued**: The task has been linked to a worker by the executor and is waiting for availability to be executed.
- **Running**: The task is running.
- **Success**: The task was executed without errors.
- **Failed**: The task encountered errors during execution and failed.
- **Skipped**: The task was bypassed due to some mechanism.
- **Upstream failed**: The dependencies failed, and therefore the task was not executed.
- **Up for retry**: The task failed, but there are defined mechanisms for retries, and therefore it can be rescheduled.
- **Sensing**: The task is a smart sensor.
- **Removed**: The task has been removed from the DAG since its last execution.

Best Practices for Apache Airflow

The Apache community provides a series of Best practices recomendations, some of which we summarize below:

- **Writing a DAG**: Although creating a new DAG in Airflow is a fairly simple task, the community warns of a number of cautions necessary to ensure that its execution does not produce unexpected results.
- **Generating Dynamic DAGs**: Sometimes writing a DAG manually is not practical. The dinamic DAG generation feature can be useful on these occasions.
- **Triggering DAGs after Changes**: The system needs enough time to process changed files. It is recommended that you avoid triggering DAGs immediately after their modification or the modification of accompanying files.
- **Reducing DAG Complexity**: Although Airflow is good at handling large volumes of DAGs with many tasks and dependencies, very complex and large numbers of DAGs can affect the Scheduler's performance. It is recommended to simplify and optimize them whenever possible.
- **Testing a DAG**: DAGs should be treated as production-level code and have numerous associated tests to ensure they produce the desired results. It is possible to write a wide variety of tests for a DAG.
- **Metadata DB Maintenance**: Over time, the Metadata database increases its storage coverage. Airflow's CLI allows you to clean up old data with the command airflow db clean.

© Copyright Tecnisys 2026. All rights reserved.

- **Handling Conflicting/Complex Python Dependencies**: Airflow has many Python dependencies that sometimes conflict with what the code wants. The community offers some strategies that can be employed to mitigate the risks.

Apache Airflow Project Details

Airflow is written in Python. Its workflows are created using Python scripts. It was designed under the principle of "configuration as code." Although there are other tools that adopt this principle, the use of Python is what allows developers to import libraries and classes.

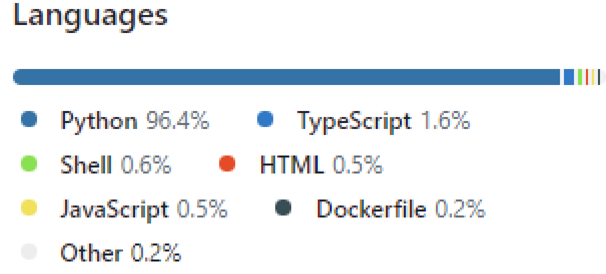


Figure 5 - Airflow languages

TDP Kubernetes

This component is also available in the **TDP Kubernetes** edition since version 3.0.

The current version is **3.0.2**, distributed via Helm Chart `tdp-airflow` v3.0.1.

For configuration details, see the TDP Kubernetes documentation.

Source(s):

- Airflow.apache.org.
- cwiki.apache.org.
- github.com/apache/airflow
- Airflow-fundaments (comunidade).

© Copyright Tecnisys 2026. All rights reserved.

Apache Ambari

Cluster Management

Developed to simplify the deployment and administration of Big Data Clusters, the Apache Ambari enables automated deployment, management of services and hosts, system monitoring of the environment, configuration versioning, and much more.

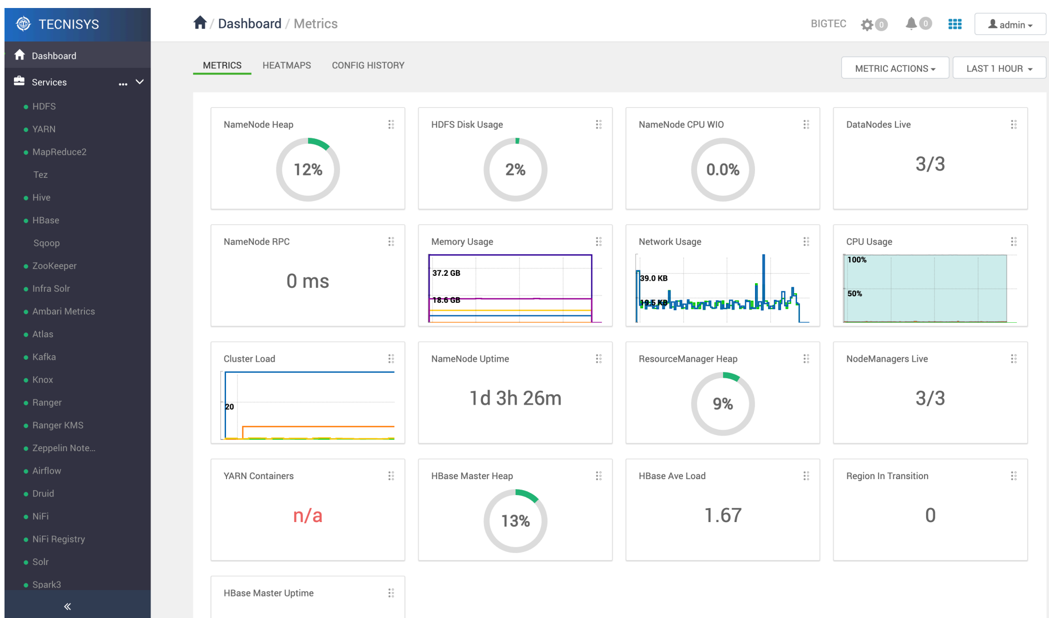


Figure 1 - Ambari example

Its consistent architecture, robust REST API, and intuitive and interactive web interface provide all the necessary resources for centralized cluster administration.

Currently, the Apache Ambari project is thriving thanks to an active and renewed community. You can follow this constant evolution in its Github repository.

Objectives

© Copyright Tecnisys 2026. All rights reserved.

Among its main objectives, conceived at the beginning of the project, the following stand out:

- **Independent Platform :** The system must architecturally support any hardware and operating system. Components that are inherently platform-dependent must be pluggable with well-defined interfaces.
- **Pluggable Components:** The architecture should not assume specific tools and technologies. Any specific tools and technologies should be encapsulated by pluggable components. The architecture should be easily extensible. The goal of pluggability does not cover the standardization of protocols between components or interfaces to operate with third-party implementations.
- **Version and Update Management:** Components running on multiple hosts must support different protocol versions to support independent updates. Updating any Ambari component should not affect the Cluster state.
- **Extensibility:** The architecture must support the addition of new services, components, and APIs. Extensibility also implies ease in modifying any configuration or provisioning steps for Service Platforms. Additionally, the ability to support different Service Platforms needs to be considered.
- **Failure Recovery:** The system must be able to recover from any component failure to a consistent state. The system should attempt to complete pending operations after recovery. If certain errors are unrecoverable, the failure should still keep the system in a consistent state.
- **Security:** Security implies 1) authentication and authorization based on user profiles (API and web interface), 2) installation, management, and monitoring of the Service Platform protected via Kerberos, and 3) authentication and encryption of network communication between Ambari components.
- **Error Tracking:** The architecture should strive to simplify the fault tracking process. Failures should be propagated to the user with the details and indicators necessary for analysis.
- **Intermediate Feedback and Near Real-Time Operations:** For operations that take a while to complete, the system needs to be able to provide feedback to the user with intermediate progress regarding the current execution of tasks, percentage of operation completed, and reference to the operation log, in a timely manner (almost in real-time).

Terminology

© Copyright Tecnisys 2026. All rights reserved.

Following are the meanings of the technical terms of the Apache Ambari project used in this documentation:

- Service: Service refers to the Services Platform services, such as HDFS, YARN, Spark, Kafka, among others. A service can have multiple components (e.g. HDFS has NameNode, DataNode, etc.) or be just a client library, with no daemon service running continuously on the cluster.

- Component: A service consists of one or more components. For example, HDFS has 3 components: NameNode, Secondary NameNode and DataNode. Components may be optional. A component can span multiple nodes (for example, instances of the DataNode component on multiple nodes).

- Node (Node or Host): Does not refer to a machine (physical or virtual) in the Cluster. Node and host are used interchangeably in this documentation.

- Operation: An operation refers to a set of changes or actions performed on a cluster to satisfy a user request or to obtain a safe state change in the Cluster. For example, starting a service or running a smoke test are operations. If a user request to add a new service to the Cluster includes the execution of a smoke test as well, the entire set of actions to fulfill the user request will make up an Operation. An operation can consist of multiple ordered “actions”.

- Task: Task is the unit of work sent to execute on a node. A task is the work that the node has to perform as part of an action. For example, an “action” may consist of installing a DataNode on node N1 and installing a DataNode and a Secondary NameNode on node N2. In this case, a “task” for N1 will be to install a DataNode and the “tasks” for N2 will be to install a DataNode and a Secondary Namenode.

- Stage: A stage refers to a set of tasks performed to complete an operation and are independent of each other. All tasks in the same stage can be performed on different nodes in parallel.

- Action: An "action" consists of a task or tasks on a machine or group of machines. Each action is tracked by an ID and nodes report its status at least at the action granularity. An action can be considered a step in execution. In this documentation, a stage and an action have a one-to-one correspondence unless otherwise specified. An action ID will be a bijection from request-id to stage-id.

© Copyright Tecnisys 2026. All rights reserved.

- Stage Plan: An operation typically consists of multiple tasks on multiple machines and they often have dependencies that require them to be executed in a specific order. Some tasks must be completed before others can be scheduled. Therefore, the tasks required for an operation can be divided into several steps, in which each step must be completed before the next stage, but all tasks in the same stage can be scheduled in parallel on different nodes.

- Manifest: The manifest refers to the definition of a task that is sent to a node for execution. The manifest must completely define the task and must be serializable. The manifest can also be persisted to disk for recovery or logging.

- Role: A role maps a component (e.g. NameNode) or an action (e.g. HDFS rebalancing, HBase smoke test, etc.).

Architecture

The Ambari architecture involves:

- **Ambari Web:** A client-side application responsible for the web interface that displays information and performs operations on the Cluster through the REST API provided by Ambari Server.
- **Ambari Server:** The Master component responsible for Cluster control. It consists of several entry points available for different needs, such as:
    - Daemon Management: Entry point for starting, stopping, resetting, and restarting the Ambari Server daemon.
    - Software Update: Entry point for upgrading Ambari Server after installation.
    - Software Configuration: Entry point for preliminary Ambari setup.
    - Authentication and Authorization Configuration: Entry point for setting up the identity management, authentication, and authorization mechanism. Available options: LDAP (Lightweight Directory Access Protocol), PAM (Pluggable Authentication Module), and Kerberos.
    - Backup and Restore: Entry point for performing backup (snapshot) and restore of the current installation, except for the database.
- **Ambari Agent:** Slave component responsible for executing actions and sending information and metrics from a given host.
- **Database:** Persistence area for the state, metrics, and metadata of the Cluster infrastructure (services and hosts). Ambari supports several Relational Database Management Systems (RDBMS), such as PostgreSQL (default), MySQL, MariaDB,

© Copyright Tecnisys 2026. All rights reserved.

Oracle, and Microsoft SQL Server, the choice of which is made during the first setup of Ambari Server.

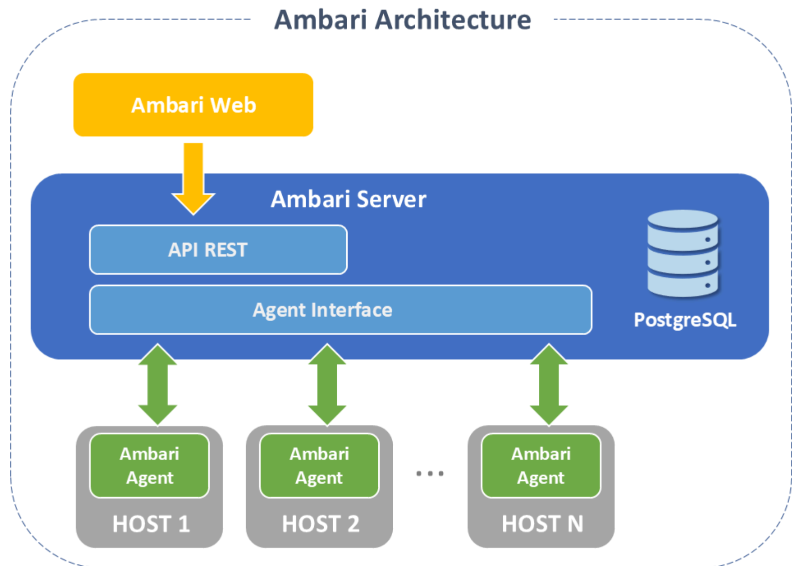


Figure 2 - Ambari Architecture

Languages

Ambari is predominantly developed in Java, JavaScript, and Python.

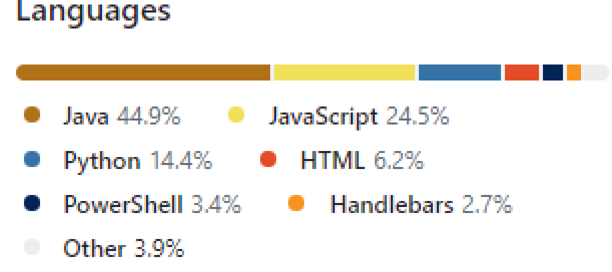


Figure 3 - Apache Ambari Main Languages

Sources:

- https://ambari.apache.org
- https://cwiki.apache.org/confluence/display/ambari
- https://github.com/apache/ambari

© Copyright Tecnisys 2026. All rights reserved.

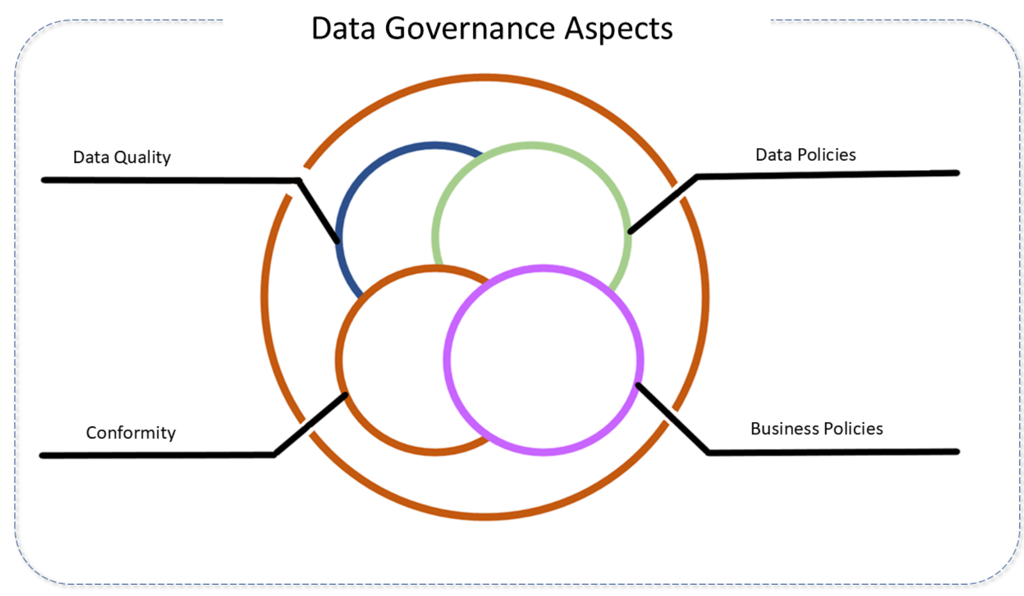
Apache Atlas

Data Governance

In response to the intense search for value and speed in decision-making by modern organizations, the market increasingly offers technologies, environments, and components to be consumed and explored with "real-time," "near-real-time," "streaming," and other strategies that allow data transformation and access to information in the shortest possible time.

Although this analytical evolution adds high capacity to digital transformations and optimizes technology parks, it demands from organizations a structured "foundation" and high maturity in maintaining processes and assets.

This is where Data Governance comes in, a fundamental process that ensures alignment with the compliance of organizations, improving the quality of insights, assisting in the adoption of regulatory compliance, reducing costs with centralized policies and systems, growing controlled and organized data, protecting them from any setbacks, within a cultural environment favorable to innovation and permeating the entire technology pipeline through which data will navigate.



© Copyright Tecnisys 2026. All rights reserved.

Figure 1 - Data Governance aspects

Apache Atlas Features

Apache Atlas is a scalable set of core governance services provided for Hadoop, aiming to assist organizations in meeting their compliance requirements. For this, it uses prescriptive and forensic models enriched by business taxonomic metadata.

Apache Atlas allows organizations to build a catalog of their assets, classifying, managing, and providing collaboration capabilities for their use by data scientists and governance teams.

The tool was designed to exchange metadata with other tools and processes inside and outside the Hadoop stack, thus allowing platform-independent governance controls to effectively meet compliance requirements.

These services include:

- **Search and prescriptive lineage:** facilitating predefined and ad hoc exploration of data and metadata and maintaining a history of data sources and how specific data were generated.
- **Metadata-driven data access control.**
- **Flexible modeling of business and operational data.**
- **Data classification:** assisting in understanding the nature of data and its classification based on external and internal sources.
- **Metadata exchange with other metadata tools.**

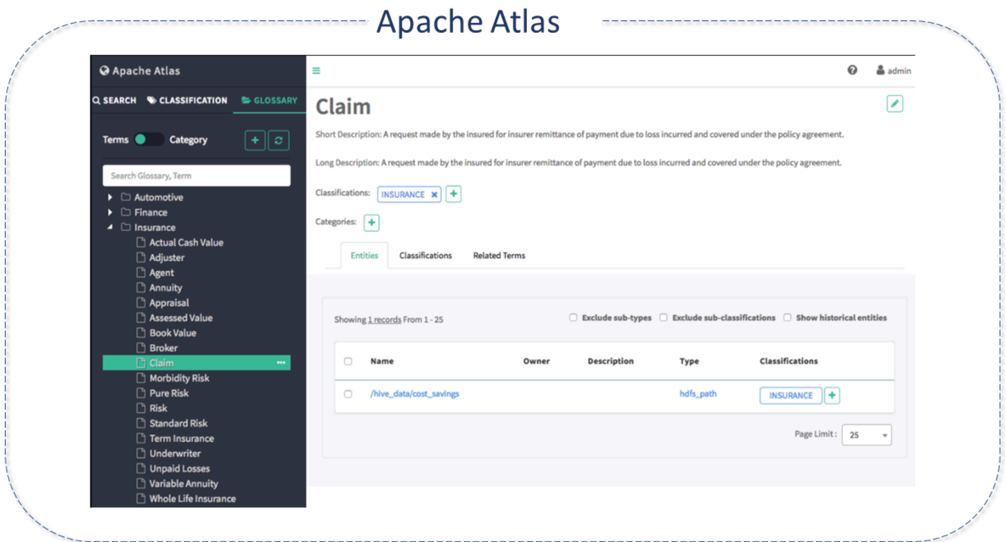


Figure 2 - Apache Atlas Interface

Apache Atlas Architecture

The architecture of Apache Atlas involves the following components:

Core

- **Type System:** Allows the definition of a model for metadata objects that you want to manage.
    - The model is composed of definitions called types.
    - The instances of the types are called entities and represent the metadata objects that will be managed.
    - All metadata objects managed by Atlas in the out-of-the-box (ready-to-use) model are modeled using types and represented as entities.
    - The generic nature of modeling in Atlas allows integrators and data administrators to define technical and business metadata and relate them through Atlas features.
- **Graph Engine:** Internally, Atlas persists the metadata objects it manages using a Graph model. This approach provides great flexibility and enables efficient manipulation of relationships between metadata objects.
    - The Graph engine is responsible for the translation between types and entities of the Type System and for the underlying graph persistence model.
    - The Graph engine also creates indexes for metadata objects so that searches can be performed efficiently.
    - Atlas uses JanusGraph to store metadata objects.
- **Ingestion/Export:** The ingest component enables the addition of metadata in Atlas, and the export component makes detected changes in metadata available to be generated as events and consumed by Consumers in response to changes in metadata in real-time.

Integration

There are two methods to manage metadata in Atlas:

- **API:** All Atlas functionalities are made available to the end-user via REST API, which enables the creation, modification, and deletion of types and entities. This is the

© Copyright Tecnisys 2026. All rights reserved.

main mechanism for querying and discovering the types and entities managed by Atlas.

- **Messaging:** In addition to APIs, there is the possibility of integration with Atlas using the messaging interface.

    - This interface is very useful for communication between metadata objects for Atlas and to consume metadata change events from Atlas. It is particularly useful when you want to use a more flexible integration to achieve more scalability, reliability, etc.
    - Atlas uses Kafka as a notification server for communication between hooks and downstream consumers of metadata notification events. Events are written by hooks and Atlas to different Kafka topics.
- **Metadata Sources**

    Atlas supports integration with several metadata sources, ready to use. Integration implies two things:

    - There are metadata models that Atlas natively defines to represent objects of these components.
    - There are components provided by Atlas to ingest metadata objects from these components (real-time or batch).

    Currently, it supports the ingestion and management of metadata from the following sources:

    - HBase
    - Hive
    - Sqoop *(deprecated — retired by the Apache Software Foundation)*
    - Storm
    - Kafka
    - Falcon

Applications

The metadata managed by Apache Atlas is consumed by a variety of applications.

- **Admin UI:** This web-based component enables administrators and data scientists to discover and "annotate" metadata.

© Copyright Tecnisys 2026. All rights reserved.

- It is a search interface with an SQL-like query language that can be used to query the metadata types and objects managed by Atlas.
- The Interface uses the Atlas REST API to create its functionality.

- **Ranger Tag-Based Policies:** Ranger is an advanced security management solution for the Hadoop ecosystem with broad integration with a variety of components.

- By integrating with Atlas, it allows security administrators to define metadata-driven policies, aiming at effective governance.

- It is a Consumer of metadata change events notified by Atlas.

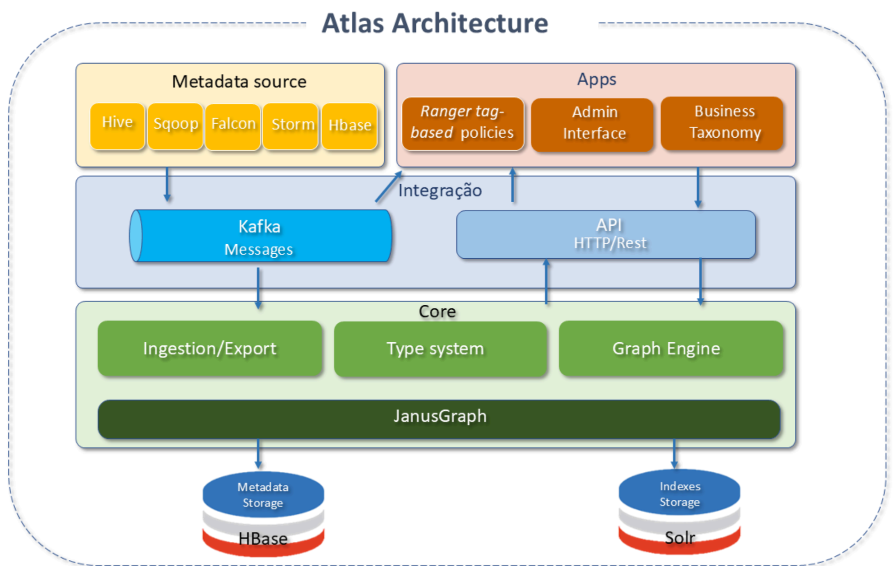


Figure 3 - Arquitetura Apache Atlas

Apache Atlas Resources

- **Knowledge Base:** Leverages existing metadata. Supports metadata exchange with other components, third-party applications, or governance tools.

- **Centralized Auditing:** Indexed and "searchable" storage from a historical repository of all governance events, including access, grants, denials, operational events related to data provenance, and metrics.

- **Policy Engine:** Runtime compliance policy based on data classification schemes, attributes, and roles.

- **RESTful Interface (compliant with REST criteria):** Support for third-party applications through REST APIs.

© Copyright Tecnisys 2026. All rights reserved.

- **Lineage:**
  - Intuitive UI to visualize data lineages as they move through processes.
  - REST APIs to work with types and instances.
- **Integration with Ranger:**: Integration with Apache Ranger enabling data authorization/masking on data access, based on classifications associated with entities. It can be used to implement dynamic classification and role-based security policies.
- **High Availability and Fault Tolerance:** Apache Atlas uses and interacts with a variety of systems to provide metadata management and data lineage for data administrators. These dependencies must be configured appropriately to achieve a high degree of availability. The community describes this support in detail here.
- **Classification Propagation**:: Apache Atlas enables classifications for entities to be automatically associated with other related entities. This is very useful when dealing with scenarios where a dataset derives its data from other datasets.
- **Business Metadata**:The Atlas type system allows the definition of a model and the creation of entities for metadata objects that you want to manage. The model captures technical attributes such as name, description, creation date, number of replicas, etc. This is very useful for expanding technical attributes with additional attributes from capturing business details to assist in organizing, searching, and managing metadata entities.
- **Security:** The following security resources are available to help secure the Platform:
  - Support for one-way SSL (server authentication) and two-way SSL (client and server authentication).
  - Service Authentication: The Platform, upon startup, is associated with an authenticated identity. By default, in an unsecured environment, this identity is the same as the user authenticated in the OS to start the server. However, in a secure cluster that uses Kerberos, it is recommended to configure a Keytab and Principal so that the Platform is authenticated in the KDC. With this, the service will interact with other secure cluster services.
  - SPNEGO-based Authentication: HTTP access to the Atlas Platform can be protected by enabling SPNEGO support on the platform. There are currently

© Copyright Tecnisys 2026. All rights reserved.

two supported mechanisms:
    - simple: authentication is performed through a provided username.
    - Kerberos: the client's authenticated KDC identity is leveraged to authenticate to the server.
- **Repair Index:** In rare cases, during entity creation, the corresponding indexes are not created in Solr. As Atlas relies heavily on Solr this would result in the entity not being returned by a search (advanced search is not affected). The Atlas Index Repair Utility for JanusGraph allows the restoration of all indexes.

- **Atlas Server Entity Type:** The AtlasServer entity type is a special type of entity:

    - Created during Export or Import operations.
    - Has special property pages that show detailed audits for export import operations.
    - Entities are linked to it using the new option within the SoftReference.
- **Replicated Attributes**: Through the attributes of the Referenciável entity type replicatedFrom and replicatedTo, it is possible to know how entities arrived at the Apache Atlas instance, whether they were created by hook ingestion or imported from another instance.

- **SoftReference**: The attribute persistence strategy is determined based on its type. The isSoftReference attribute option set to true causes a non-primitive attribute type to be treated as a primitive attribute.

- **Ranger-Atlas Access Policies**:

    - **Classification-based access controls:** A data entity can be marked as metadata related to compliance or a specific taxonomy and used to assign permissions to a user or group.
    - **Data expiration-based access policy:** Expiration dates to automatically deny access to tagged data after the specified date.
    - **Location-specific access policies:** Access to a user while in a particular location but not in another, even though it is the same user.
    - **Prohibition of dataset combinations:** Restriction based on datasets (for example, to prevent them from being combined).

Apache Atlas Project Details

© Copyright Tecnisys 2026. All rights reserved.

Apache Atlas was developed predominantly in Java. It has Dashboard components in JavaScript.

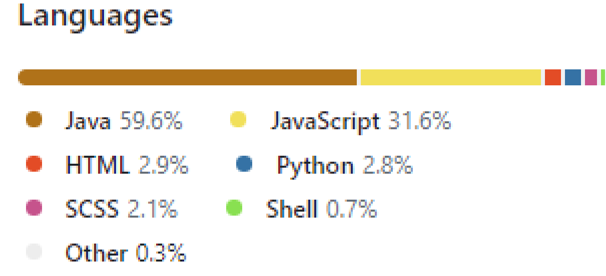


Figure 4 - Atlas Languages

Fonte(s):

- Atlas.apache.org
- GitHub - Apache Atlas
- Guia do Usuário técnico - Apache Atlas

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake

Storage Structure

In modern data architectures, the **storage layer** is the foundation that enables storing large volumes of data in a scalable and cost-effective way. It uses systems like Amazon S3, Azure Data Lake Storage, or HDFS to store files in formats such as Parquet, ORC, or Avro. However, this layer does not enforce structure, version control, schema validation, or transactions — it only stores data.

On top of this foundation, the concept of the **Data Lake** emerged: an architecture designed to store raw data at scale, coming from multiple sources in different formats. The promise of data lakes was to centralize everything in one place — making future analyses easier, integrating varied data, and reducing costs. This made them popular in data science projects and massive ingestion pipelines.

Why Data Lakes Became Popular

Even with technical limitations, data lakes became popular for their flexibility and cost savings. They allow:

- **Storing data in its original format**, without the need for prior modeling.
- **Collecting data in batch and real-time**, with virtually unlimited scalability.
- **Serving multiple user profiles and tools**, supporting ad hoc analysis, machine learning, reports, and dashboards.
- **Consolidating and democratizing access to data**, breaking down legacy system silos.

© Copyright Tecnisys 2026. All rights reserved.

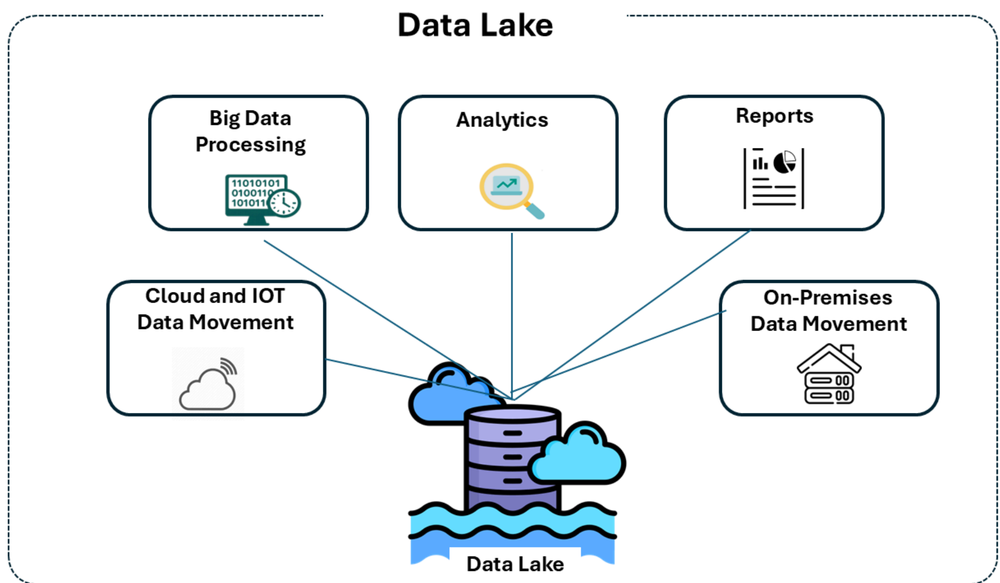


Figure 1 - Data Lake

Limitations of Data Lakes

Despite the benefits, traditional data lakes do not offer the necessary controls for reliable analytical environments:

- **No ACID transactions**: multiple writes can cause inconsistency or corruption.
- **No version control of data**: it's not possible to query or restore a previous table state.
- **No schema consistency validation**: structural errors may go unnoticed, compromising analyses.
- **No performance optimization for queries**: small and disorganized files affect performance.

These limitations prevent data lakes from being a reliable base for critical analytical applications. To overcome these issues, a new layer was created — the **Delta Lake**.

What is Delta Lake

Delta Lake is a transactional layer that operates on top of an existing data lake. It does not replace the data lake — it complements it. By adding ACID transactions, version control, schema management, and integration with tools like Apache Spark, Delta Lake turns raw data lakes into a more reliable, robust, and auditable platform — known as a **Lakehouse**[1].

Delta Lake Features

© Copyright Tecnisys 2026. All rights reserved.

Delta Lake introduces essential functionalities that fix the deficiencies of traditional data lakes and enable a reliable and scalable data architecture. Key features include:

- **ACID Transactions**: ensure atomicity, consistency, isolation, and durability even in distributed environments. Reads and writes are safe and predictable.
- **Version Control and Time Travel**: all data changes are automatically versioned, enabling queries on any past state, rollback of changes, and historical audits.
- **Schema Management and Validation**: prevents ingestion of data with incorrect structures. Schema can evolve in a controlled way without compromising table integrity.
- **Unification of Batch and Streaming**: the same Delta table can receive batch and streaming data, offering consistency across ingestion modes.
- **Optimized Query Performance**: includes techniques like small file compaction, column ordering (Z-order), and *data skipping* to dramatically speed up analytical queries.
- **Spark Ecosystem Integration**: allows SQL operations, structured APIs, and real-time processing directly on Delta tables.

Delta Lake Architecture

Delta Lake's architecture builds on top of traditional data lakes, introducing a transactional layer that ensures reliability and performance. This layer consists of three key components:

- **Storage Layer**: Uses the same repository as the data lake, such as Amazon S3, Azure Data Lake Storage, or HDFS, where data is stored in Parquet format. This physical layer continues to provide scalability and cost-effectiveness.
- **Delta Log**: A transaction log that records all table changes. This log enables version control, ensuring every change is tracked and enabling time travel and auditing.
- **Delta Tables**: Logical abstractions that unify the stored data and the Delta Log. Through this integration, Delta tables provide ACID transactions, schema management and evolution, and unified batch and streaming ingestion.

© Copyright Tecnisys 2026. All rights reserved.

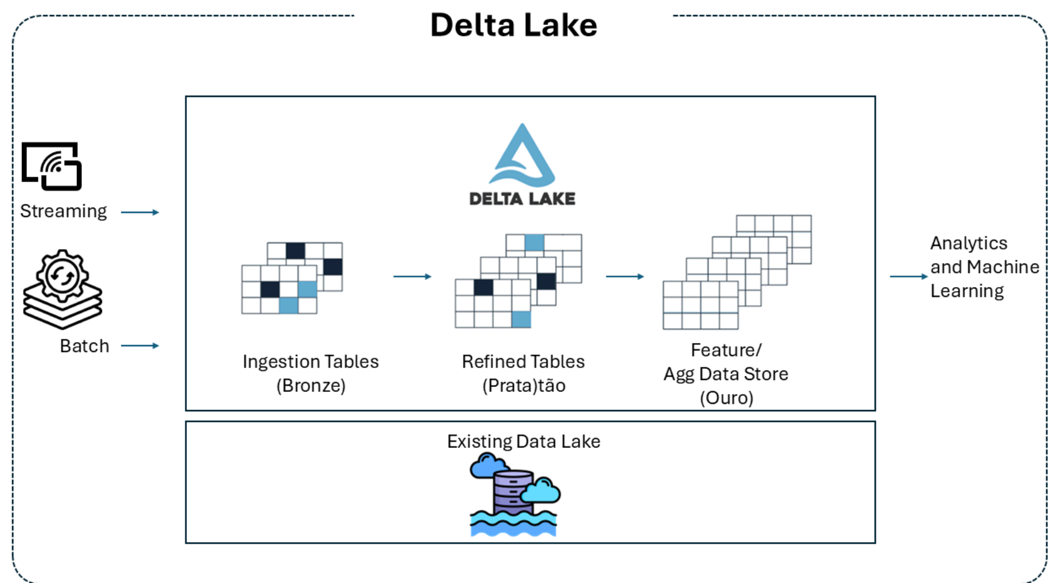


Figure 2 - Delta Lake Architecture

How Delta Lake Works

Delta Lake continuously operates on physical storage, managing data changes and ensuring secure and trackable transactions. Its operation is organized into three main areas:

Data Ingestion

Delta Lake supports multiple ingestion modes:

- **Streaming**: Integrates with Apache Spark Structured Streaming for continuous ingestion, enabling real-time updates with exactly-once semantics.

- **Batch**: Supports large-scale batch ingestion with operations like `MERGE`, `UPDATE`, `DELETE`, and `INSERT`, without compromising table integrity.

Query Processing

Delta tables can be queried using:

- **SQL**: Interactive or analytical queries directly on versioned data.
- **High-level APIs**: Compatible with Apache Spark APIs (DataFrame, SQL, Structured Streaming).
- **Time Travel**: By navigating the Delta Log, it’s possible to query the state of data at any point in time.

© Copyright Tecnisys 2026. All rights reserved.

Data Management

Delta Lake offers mechanisms for continuous optimization:

- **Small File Compaction**: Operations like `OPTIMIZE` reorganize data to improve query performance.
- **Z-order Sorting**: Organizes data in high-selectivity columns, reducing read costs.
- **Obsolete Data Cleanup**: With `VACUUM`, unreferenced files are safely removed, freeing up space and preserving performance.

These features work in an integrated way, maintaining data integrity even under high concurrency or continuous ingestion scenarios.

Delta Lake Capabilities

- **ACID Transactions in Spark**: serializable isolation levels ensure readers never see inconsistent data.
- **Scalable Metadata Handling**: leverages Spark's distributed processing to manage metadata for petabyte-scale tables with billions of files.
- **Unified Streaming and Batch**: one Delta table is both a batch table and a streaming source and sink. Streaming ingestion, batch backfill, and interactive queries work seamlessly.
- **Schema Enforcement**: automatically handles schema variations to prevent invalid record insertion during ingestion.
- **Time Travel**: data versioning allows rollback, full historical audit trails, and reproducible machine learning experiments.
- **Upserts and Deletes**: supports merge, update, and delete operations for complex use cases like change data capture, slowly changing dimensions (SCD), and streaming upserts.
- **Connector Ecosystem**: connectors are available to read and write Delta tables from various engines such as Apache Spark, Flink, Hive, Trino, AWS Athena, etc.

When to Use Delta Lake

Delta Lake is ideal for scenarios requiring:

- **Data in various formats** coming from multiple sources;
- **Use of the data in many different downstream tasks**, such as analytics, data science, machine learning, etc.;

© Copyright Tecnisys 2026. All rights reserved.

- **Flexibility** to run many different types of queries without having to ask questions in advance;
- **Real-time Data Processing**: real-time ingestion and analysis with consistency guarantees.
- **Management of Large Data Volumes**: supports petabytes of data with optimized performance.
- **Audit and Versioning Needs**: access to previous data versions for auditing and recovery.

When Delta Lake May Not Apply

Delta Lake may not be the best choice for:

- **High-concurrency OLTP Systems**: applications requiring low-latency and high-concurrency transactions may benefit more from traditional relational databases.
- **Complex Constraint Requirements**: such as foreign keys and triggers, which are not natively supported by Delta Lake.

Interaction with Apache Spark

Delta Lake is fully compatible with Apache Spark APIs and was designed for complete integration with Structured Streaming, allowing unified data operations across batch and streaming, and supporting scalable incremental processing. It enables:

- **Real-time Data Ingestion and Processing**: using Spark Structured Streaming.
- **SQL Queries and DML Operations**: with support for ACID transactions.
- **Performance Optimizations**: leveraging Spark's distributed processing capabilities.

Delta Lake X Traditional RDBMS

- **Storage**: Delta Lake uses Parquet files in distributed file systems, while RDBMSs use block storage.
- **Transactions**: Delta Lake provides ACID transactions in distributed environments, whereas RDBMSs handle them in centralized systems.
- **Schema Evolution**: Delta Lake allows flexible schema evolution, while RDBMSs require more rigid structural changes.

Best Practices for Delta Lake

© Copyright Tecnisys 2026. All rights reserved.

- **Small File Compaction**: use the OPTIMIZE command to improve query performance.
- **Z-order Sorting**: apply Z-ordering on frequently filtered columns to accelerate queries.
- **Version Management**: use time travel for audits and data recovery.

Delta Lake Project Details

- **Programming Language**: developed in Scala and Java.
- **License**: open source under Apache 2.0 license.
- **Integration**: compatible with Apache Spark, Flink, Presto, Trino, among others.

TDP Kubernetes

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **4.0.0**, distributed via Helm Chart `tdp-deltalake` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

References

- Delta Lake Documentation
- GitHub - Delta Lake

Footnotes

1. A lakehouse architecture combines the advantages of data lakes and data warehouses in a single platform. It uses the low-cost, scalable storage of a Data Lake, provides ACID transactions, versioning, schema enforcement, and catalogs like a Data Warehouse, supports batch and streaming processing in the same place, and enables SQL and machine learning on the same data. ↩

© Copyright Tecnisys 2026. All rights reserved.

Apache Druid

Real-Time Analytical Database

The ability to store, process, and retrieve data at very high speeds is one of the main requirements met by a Real-Time Analytical Database.

In traditional DBMSs, deriving value from data is a difficult task because the data needs to be moved, transformed, and loaded into a Data Warehouse or Data Lake before consumption. This takes time, sometimes hours, often days.

Streaming mechanisms, like Kafka, help, but only to a certain extent, as the fundamental need is a database with the power to process large amounts of data in real time.

Real-time analytical databases optimize resources to allow for heavy workloads. They feature lightweight ingestion protocols and efficient disk storage structures to allow very fast ingestions. Their architecture uses massively parallel processing with a high degree of concurrency, avoiding high infrastructure costs.

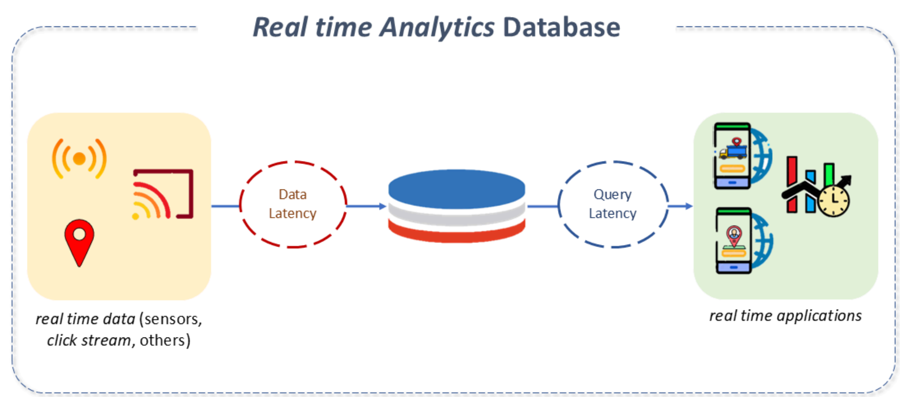


Figure 1 - Data Governance Aspects

Apache Druid Features

© Copyright Tecnisys 2026. All rights reserved.

Apache Druid is a real-time analytical database management system designed for fast, fragmented analyses (OLAP queries) on large datasets. It empowers use cases where real-time ingestion, fast query performance, and high productivity are important.

Its origin dates back to 2011 when a tech company's data team decided to create their own database after trying several market alternatives to solve a real-time data aggregation and query problem from the Internet to analyze digital ad auctions. The first version of Druid scanned, filtered, and aggregated a billion rows in 950 milliseconds.

Druid became open source after a few years and an Apache Software Foundation top-level project in 2016.

In 2023, more than 1400 organizations use Druid, and the tool has over 10,000 active developers in its community.

Among its main features, we highlight:

- **Scalability and Flexibility**: Its elastic and distributed architecture allows the creation of any application at any scale.
- **Efficiency and Integration**: Druid's motto is "do it only if necessary," minimizing the Cluster's work:
  - Does not load data from disk to memory (or vice versa) when not needed.
  - Does not decode data if it can operate directly on encoded data.
  - Does not read the entire dataset if it can read a smaller index.
  - Does not start new processes for each query if it can use a long-running process.
  - Does not send unnecessary data between processes or servers.
  - The Druid query engine and storage format are fully integrated and designed together to minimize the work done by data servers.
  - Generally used as a backend database for GUI ("Graphical User Interface") analytical applications or for highly concurrent APIs demanding fast aggregations.
- **Resilience and Durability**: Druid is "self-healing," "self-balancing," and fault-tolerant. It is designed to operate continuously without downtime, even during configuration changes and software updates, preventing data loss, even in case of major system failures. Its Cluster rebalances itself automatically in the background

© Copyright Tecnisys 2026. All rights reserved.

without downtime. When a server fails, the system "absorbs" the failure and continues operating.

- **High Performance**: Druid's combine features to allow high performance in high concurrency, avoiding unnecessary work.

- **High Concurrency**: This was one of the original goals of the Druid project. Many Clusters support hundreds of thousands of queries per second. The key to this is the unique relationship between storage and computing resources. Data is stored in segments, which are checked in parallel by scatter/gather queries.

- **High-Speed Data Ingestion**: For streaming ingestion, middle managers and indexers are enabled to respond to queries in real-time. All tables are always fully indexed, making index creation unnecessary.

- Druid works best with event-oriented data. Common application areas for Druid include:

    - Clickstream analysis, including web and mobile analytics.
    - Network telemetry analysis, including network performance monitoring.
    - Server metrics storage.
    - Supply chain analysis, including manufacturing metrics.
    - Application performance metrics.
    - Digital marketing/advertising analysis.
    - Business intelligence (OLAP).

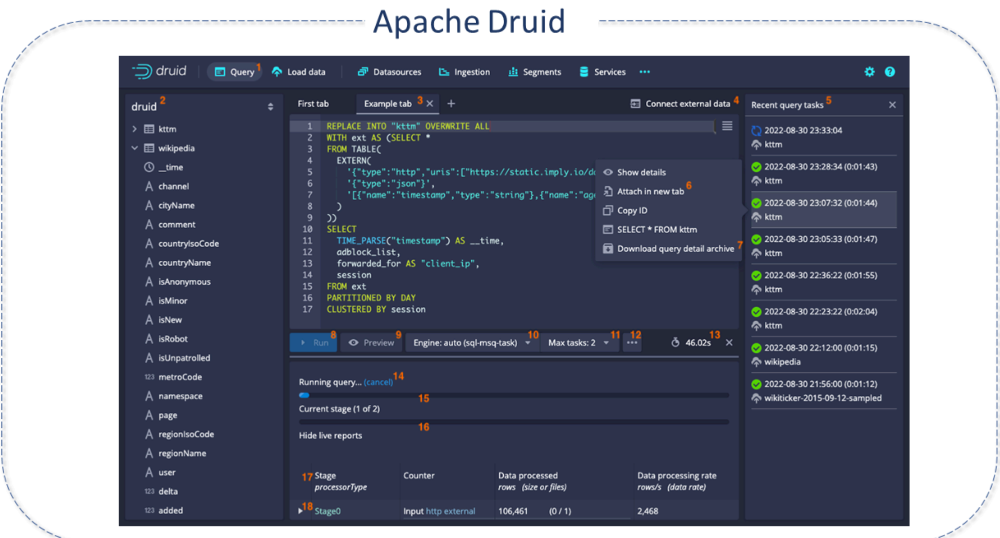


Figure 2 - Interface Apache Druid

© Copyright Tecnisys 2026. All rights reserved.

Apache Druid Architecture

Apache Druid has a distributed architecture designed to be cloud-friendly and easy to operate. Services can be configured and scaled independently, ensuring maximum flexibility in Cluster operations.

Druid combines ideas from data warehouses, time-series databases, and log search systems.

Some of its main components include:

Druid Services (Process Types)

- Coordinator: Manages data availability in the Cluster.
- Overlord: Controls the assignment of workloads for data ingestion.
- Broker: Handles queries from external clients.
- Router Services (optional): Route requests to Brokers, Coordinators, and Overlords.
- Historical Services: Store queryable data.
- The MiddleManager Services: Ingest data for most ingestion methods.
- Indexer (optional and experimental): Acts as an alternative to "MiddleManager + Peon". Instead of forking a separate JVM process per task, the indexer runs tasks as separate threads within a single JVM process.

> NOTE
>
> Services can be viewed in the Services Tab in the web console.

Servers

Druid servers can be deployed in any desired way. To facilitate deployment, it is suggested to organize them into three types:

- Master: Runs Coordinator and Overlord processes, manages data availability and ingestion.

© Copyright Tecnisys 2026. All rights reserved.

- Query: Runs Broker and optional Router processes and handles queries from external clients.
- Data: Runs Historical and MiddleManager processes, performs ingestion workloads, and stores all queryable data.

External Dependencies:

In addition to its built-in process types, Druid also has three external dependencies that leverage existing infrastructure when present:

- Deep Storage: As part of ingestion, Druid securely stores a copy of the data segment in Deep Storage, creating a continuous and automated additional copy of the data in the cloud or HDFS. It makes the segment immediately available for queries and creates a replica of each data segment. It is always possible to recover data from Deep Storage, even in the unlikely event that all servers fail.
- Metadata Storage: Contains various shared system metadata, such as segment usage information and task information. In a Cluster deployment, it is usually a traditional RDBMS like PostgreSQL. In a single-server deployment, it can be a local Apache Derby database.
- Zookeeper: Used for active Cluster management, coordination, and leader election.

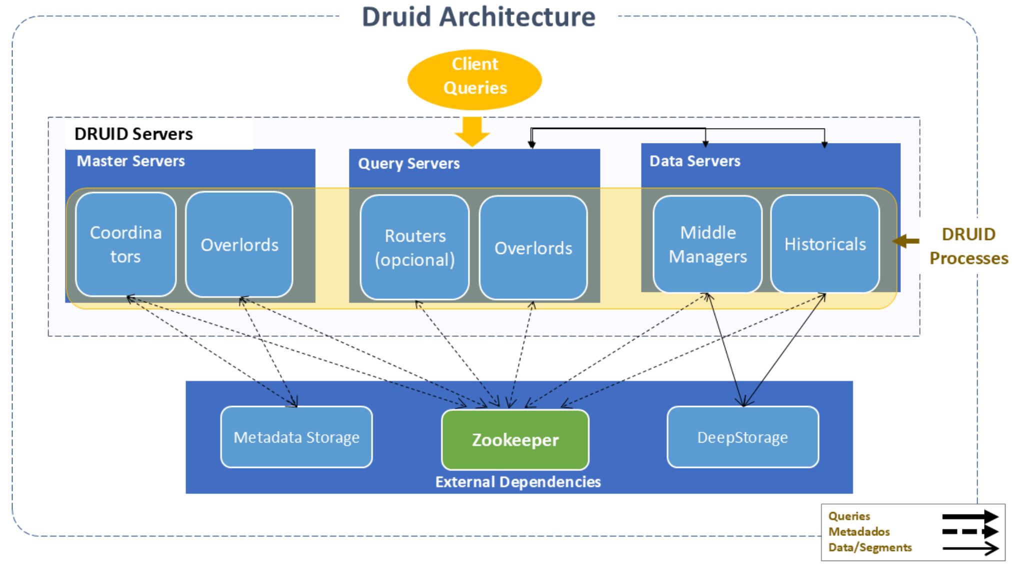


Image 3 -Druid Architecture

How Apache Druid Works

Storage Design

© Copyright Tecnisys 2026. All rights reserved.

- **Datasources and Segments:**: Druid data is stored in datasources, which are similar to tables in a traditional RDBMS. Each datasource is partitioned by time and, optionally, by other attributes. Each time interval is called a chunk – for example, a single day, if partitioned by day. Within a chunk, data is partitioned into one or more segments. Each segment is a single file. Once segments are organized into time chunks, it is sometimes appropriate to think of segments as living on a timeline.

  A datasource can have from a few segments to hundreds of thousands or millions of segments. Each segment is created by a MiddleManager as mutable and uncommitted. Data can be queried as soon as it is added to an uncommitted segment. The segment building process speeds up later queries, producing a compact and indexed data file:

  - Converted to columnar format.
  - Indexed with bitmap indexes.
  - Compressed with type-aware compression for all columns.

  Periodically, segments are committed and published in deep storage, becoming immutable, and passed from MiddleManagers to Historical processes. An entry about the segment is also written in Metadata Storage. This entry is a self-descriptive bit of metadata about the segment, including things like the segment schema, its size, and location in deep storage, informing Coordinators of what data is available in the Cluster.

- **Indexing and Handoff:**: Indexing is the process by which new segments are created. Handoff is the process by which they are published and begin to be served by Historical processes.

- **Segment Identifiers:**:

  Every segment has a 4-part identifier with the following components:

  - Datasource name
  - Time interval (for the time chunk containing the segment), which corresponds to the segmentGranularity specified at ingestion time.
  - Version number (usually an ISO8601_timestamp_corresponding to when the set of segments was started). This serves multiversion concurrency control.
  - Partition number (an integer unique within the datasource+interval+version, not necessarily contiguous).

© Copyright Tecnisys 2026. All rights reserved.

- Segment Lifecycle:

  Every segment has a lifecycle involving:

  - Metadata storage: Segment metadata is stored in Metadata Storage as soon as the segment finishes being built (publication).
  - Deep storage: Segment data files are uploaded to deep storage as soon as the segment finishes being built, before publication.
  - Available for querying on a Druid data server, such as a real-time task or a Historical process.

Ingestion:

Loading data into Druid is referred to as ingestion or indexing. When data is ingested into Druid, it reads the data from the source system and stores it in data files called segments. Generally, segments contain a few million rows each.

For most ingestion methods, a MiddleManager or Indexer process will load the data. The only exception is Hadoop-based ingestion, which uses a MapReduce job on Yarn.

During ingestion, Druid creates segments and stores them in deep storage. Historical nodes load the segments into memory to answer queries. For streaming ingestion, MiddleManagers and Indexers can respond to queries in real time as data is being loaded.

Common ingestion methods include:

- **Streaming**: Two options are Kafka and Kinesis.

  - **Kafka**:: When the Kafka indexing service is enabled, supervisors can be configured on the Overlord to manage the creation and lifecycle of Kafka indexing tasks. Kafka indexing tasks read events using Kafka's partition and offset mechanism to ensure exactly-once ingestion. The supervisor monitors the state of indexing tasks to:

    - Coordinate handoffs
    - Manage failures
    - Ensure scalability and replication requirements are met.
  - **Kinesis**:: When the Kinesis indexing service is enabled, supervisors can be configured on the Overlord to manage the creation and lifecycle of Kinesis

© Copyright Tecnisys 2026. All rights reserved.

indexing tasks. These tasks read events using Kinesis's shard and sequence number mechanism to ensure exactly-once ingestion. The supervisor monitors the state of indexing tasks to:

- Coordinate handoffs
- Manage failures
- Ensure scalability and replication requirements are met.

- **Batch**: Three options are Native batch, SQL, or Hadoop-based.
    - **Native Batch Ingestion**::

      Apache Druid supports two types of native batch indexing:

        - Parallel indexing: Allows multiple indexing tasks to run concurrently.
        - Simple indexing: Runs a single task at a time.
    - **Hadoop-based Batch Ingestion**::

      Hadoop-based Batch Ingestion is supported via a task that can be submitted to a running Overlord instance. For comparisons of ingestion types (Hadoop-based, native, and native batch), refer to the community site here.

    - **SQL-based Batch Ingestion**::

      Druid supports SQL-based batch ingestion using the druid-multi-stage-query extension, which adds a multi-stage query engine for SQL, enabling SQL Insert and Replace operations as batch tasks. As an experimental feature, the Select operation can also be used.

Query Processing:

Queries are distributed across the Druid cluster and managed by a Broker. Queries first enter the Broker, which identifies the segments with data that may belong to the query. The segment list is always broken down by time and other attributes, depending on how the datasource is partitioned. The Broker identifies which Historicals and MiddleManagers are serving the segments and distributes a rewritten subquery to each of these processes. These processes execute each subquery and return the results to the Broker, which merges them to obtain the final response, returning it to the original caller.

Data Management:

© Copyright Tecnisys 2026. All rights reserved.

Data management operations involving replacing or deleting segments include:

- Updates
- Deletions
- Schema Changes: For new and existing data.
- Compaction and Automatic Compaction.

SQL Queries:

Queries can be made via Druid SQL (Druid translates SQL queries into its native query language) or native Druid SQL.

The SQL plan happens at the Broker. To configure the SQL plan and JDBC query, the Broker's runtime properties must be configured.

Key Resources of Apache Druid:

- **Columnar Storage Format**: Druid uses column-oriented storage, meaning it loads only the columns needed for a particular query. This improves query speed. Additionally, to support fast scans and aggregations, Druid optimizes the storage of each column according to its data type.

- **Optimized Data Format**: Ingested data is automatically columnarized, time-indexed, dictionary-encoded, bitmap-indexed, and type-aware compressed.

- **Real-Time or Batch Ingestion**: Druid can ingest data in real-time or batches. Ingested data is immediately available for querying.

- **Indexes for Fast Filtering**: Druid uses compressed bitmap indexes, such as roaring or CONCISE, for fast filtering and searching across multiple columns.

- **Time-Based Partitioning**: Druid partitions data by time first and can optionally implement additional partitioning based on other fields. Time-based queries access only the partitions corresponding to the query's time range, significantly improving performance.

- **Approximation Algorithms**: Druid includes algorithms for approximate count-distinct, approximate ranking, and approximate histogram and quantile calculations. These algorithms offer limited memory usage and are generally much

© Copyright Tecnisys 2026. All rights reserved.

faster than exact calculations. For situations where precision is more important than speed, Druid also offers exact count and ranking.

- **Automatic Roll-Up at Ingestion Time**: Druid optionally supports summarizing data at ingestion time. This pre-aggregation can significantly reduce storage costs and improve performance.

- **Interactive Query Engine**: Uses scatter/gather for high-speed queries with pre-loaded data in memory or local storage to avoid data movement and network latency.

- **Tiers and QoS**: Configurable tiers with quality of service allow optimal cost-performance for mixed workloads, ensuring priority and avoiding resource contention.

- **Stream Ingestion**: Connector-free integration with streaming platforms allows "query on arrival," high scalability, low latency, and guaranteed consistency.

- **Uninterrupted Reliability**: Automatic data services, including continuous backup, automated recovery, and multi-node replication, ensure high availability and durability.

When to Use Apache Druid

Druid is used by many organizations of various sizes for different use cases. It is a good choice for the following situations:

- High insertion rates and few updates.
- Predominance of aggregation queries and reports, such as group by.
- Query latencies ranging from 100ms to a few seconds.
- Data with time components (Druid includes optimizations and design options specifically related to time).
- Even when there are multiple tables, each query hits only one large distributed table. Queries may potentially hit more than one smaller lookup table.
- Data columns with high cardinality (e.g., URLs, user IDs) and the need for fast counting and ranking.

When Not to Use Apache Druid

- Low-latency updates to existing records using a primary key. Druid supports streaming inserts but not streaming updates. Updates can be performed using

© Copyright Tecnisys 2026. All rights reserved.

background batch jobs.
- Creating an offline reporting system where query latency is not very important.
- Needing large joins, meaning joining one large fact table to another large fact table.

Apache Druid Integration with Spark

Druid and Spark are complementary solutions since Druid can be used to accelerate OLAP queries in Spark.

Spark is a general cluster computing framework initially designed around the concept of Resilient Distributed Datasets (RDDs). RDDs allow data reuse by keeping intermediate results in memory and enable Spark to provide fast calculations for iterative algorithms. This is especially good for certain workflows, like machine learning (where the same operation may be applied repeatedly until a result converges).

Spark's generality makes it well-suited as an engine for processing (cleaning or transforming) data. While it provides the capability to query data via Spark SQL, like Hadoop, its query latencies are not specifically targeted for subsecond iterative responses.

Druid focuses on extremely low-latency queries and is ideal for powering applications used by thousands of users, where each query needs to return quickly enough for users to interactively explore the data.

Druid fully indexes all data and can act as an intermediary layer between Spark and an application.

A typical setup is to process data in Spark and load the processed data into Druid for faster access. For more details on Druid and Spark interaction, refer to the documentation here.

Major Differences Between Apache Druid and Traditional RDBMS

- **Schemas**: Druid stores data in datasources, similar to tables in traditional RDBMS. Its data models share similarities with relational and time-series data models.

    - Its schemas must always include a primary timestamp. This field is used for partitioning and sorting data. It is also used to quickly identify and retrieve data within the query time range and for data management, such as dropping time chunks, overwriting time chunks, and time-based retention rules.

© Copyright Tecnisys 2026. All rights reserved.

- **Dimensions**: Dimensions are columns that Druid stores as-is. They can be used for any purpose, such as grouping, filtering, or aggregating dimensions at query time. With rollup disabled, Druid treats the set of dimensions as a set of columns to be ingested. Dimensions behave exactly like in any database without a rollup feature.

- **Metrics**: Metrics are columns that Druid stores in an aggregated form. They are most useful with rollup enabled. With a specified metric, an aggregation function can be applied to each row during ingestion.

Best Practices for Apache Druid

- Rollup:: Druid can roll up data as it ingests to minimize the volume of raw data that needs to be stored. This is a form of summarization or pre-aggregation.

- Partitioning and Sorting:: Proper partitioning and sorting can have a substantial impact on performance.

- Sketches for High Cardinality Columns: for high cardinality columns: When dealing with high cardinality columns, like user IDs or other unique identifiers, consider using sketches for approximate analysis before operating with actual values.

  - When using a sketch, Druid does not store the raw original data but a sketch of it that can be used for later calculations at query time.
  - Popular use cases include distinct count and quantile calculations.
  - Sketches serve two purposes: improving rollup and reducing memory consumption during queries.
- Strings vs Numeric Dimensions:: To ingest a column as a numeric dimension, specify the column type in the dimensions section. If the type is omitted, Druid will ingest the column as a default string type.

- Segments:: For optimal performance under heavy query loads, segment file size should be within the recommended range of 300 to 700 MB. If larger, consider adjusting the segment time granularity or partitioning your data, and/or adjusting the targetRowsPerSegment in partitionsSpec. A good starting point for this parameter is 5 million rows.

Apache Druid Project Details

Apache Druid was developed in Java.

© Copyright Tecnisys 2026. All rights reserved.

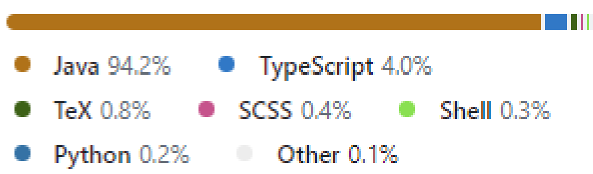


Figure 4 - Druid languages

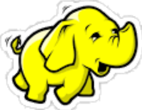
Sources:

- Apache Druid
- Apache Druid Technology
- Apache Druid GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Hadoop

Distributed processing of large volumes of data



Apache Hadoop is an open-source implementation of the MapReduce paradigm, which defines an architecture for distributed and parallel processing of large volumes of data, so that they can be "executed" on multiple servers, using simple programming models.

> **NOTE**
>
> The MapReduce paradigm expands single servers to many machines, offering local computing and storage. The reason for its scalability is the distributed nature of its solution - a complex initial task is divided into several smaller ones, and then executed on different machines. Subsequently, the result is combined to generate the solution of the initial task.

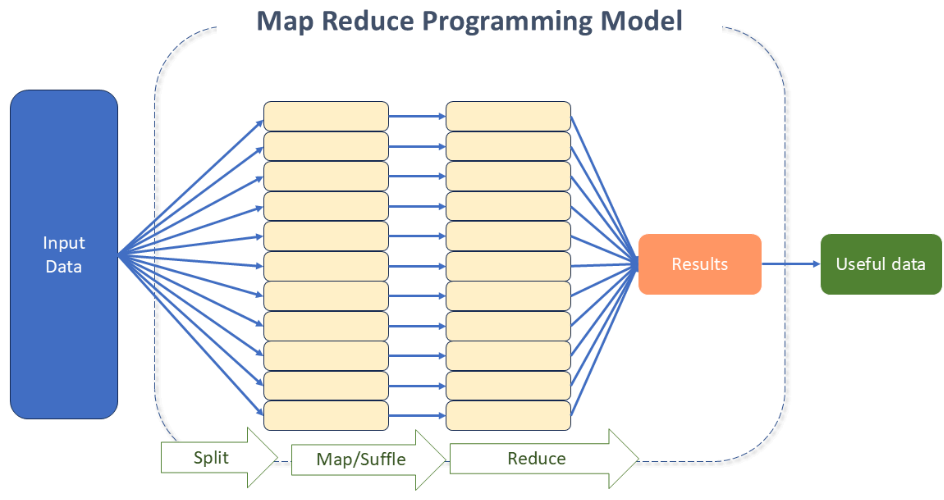


Figure 1 - MapReduce Paradigm

Apache Hadoop emerged in 2006, from a web search solution called Nutch Distributed File System (NDFS) which, according to its creators, Doug Cutting and Mike Cafarella,

© Copyright Tecnisys 2026. All rights reserved.

had its genesis in the document Google File System, published in October 2003. NDFS evolved into what we now know as Hadoop Distributed File System (HDFS).

Apache Hadoop, as well as its set of linked solutions, is one of the most popular and high-performance implementations currently used to solve challenges related to the collection, storage, and analysis of data, whether structured or not, static or real-time.

Apache Hadoop Benefits

- **Scalability:** Hadoop achieves scalability relatively simply through minor modifications to configuration files. The effort to prepare an environment with a thousand machines is not much greater than with 10. The limitations are due to the resources, disk space, and processing capacity available.
- **Simplicity:** Hadoop manages issues related to parallel computing, such as fault tolerance, scheduling, and load balancing. Its operations boil down to mapping (Map) and joining (Reduce), which keeps it focused on abstraction and task processing in the MapReduce programming model.
- **Low Cost/Economy:**
    - As free software, it does not require the purchase of licenses or the hiring of specialized personnel.
    - It can process data on conventional machines and networks.
    - Allows applications to run "in the cloud", without the need to deploy your own cluster of machines.
- **Open Source:** It has an active community that involves many companies and independent programmers sharing knowledge, promoting improvements, and collaborating with its quality. This collaborative work promotes timely updates and fixes with higher quality, as individual production is evaluated by everyone.
- **High Availability:** Hadoop does not depend on hardware to deliver high availability. Its libraries were built to detect and handle failures at the application layer, thus providing high availability in a computer cluster.

Apache Hadoop Architecture

The key components of Hadoop are the MapReduce and the HDFS. With its evolution, new subprojects were incorporated, aiming to solve specific problems.

Main Components of Apache Hadoop

- **Hadoop Common:** It is the "pillar" of Hadoop: where a predefined set of utilities and libraries that can be used by other modules within the Hadoop Ecosystem (Java

© Copyright Tecnisys 2026. All rights reserved.

Archive (JAR) files and scripts) is stored.

- **HDFS:** It is a scalable, distributed file system designed to store very large files and provide high-throughput access. It provides high-throughput access to application data.
  - The idea behind its solution is to ensure scalability and reliability for parallel processing, using the concept of replicas and blocks (with a fixed size of 64MB, in general, stored on more than one server).
- **Hadoop YARN:** Responsible for managing and scheduling the computational resources of the Cluster.
- **Hadoop MapReduce:** It is a software framework created to process large amounts of data. An excellent solution for parallel data processing.
  - Each task is specified in terms of mapping and reducing functions. Both tasks run in parallel, on the Cluster, while the necessary storage system is provided by HDFS.

Subprojects of Apache Hadoop

The need for new alternatives to promote faster and more efficient processing than traditional DBMSs brought large companies closer to the Platform, which, because it is an open-source project, received great encouragement.

With the emergence of new needs, new tools were created to run on top of Hadoop. As they consolidated, they became part of it. It was these contributions that collaborated with the rapid evolution of Apache Hadoop.

Below are the main tools of the Hadoop Ecosystem, which make up the Tecnisys Data Platform (TDP):

© Copyright Tecnisys 2026. All rights reserved.

Figure 2 - Hadoop Ecosystem

Evolution of Apache Hadoop

▸ See the Timeline - with the evolution of Apache Hadoop

Configuration and Use

- Configuring a Single Node Cluster
- Configuring a Cluster
- Starting and Stopping a Cluster
- Using the File System shell
- Knowing the commands and subprojects of Hadoop

Compatibility Guidelines

- Hadoop.apache.org -> Specifications for the Hadoop developer community

© Copyright Tecnisys 2026. All rights reserved.

- Hadoop.apache.org -> Specifications for System Administrators

The Future of Apache Hadoop

In April 2021, the Apache Software Foundation announced the retirement of 13 projects related to BigData, ten of which were part of the Hadoop Ecosystem (Eagle, Sentry, Tajo, etc.).

This action raised questions about the future of Apache Hadoop. However, the Platform has a significant number of large users who continue to use it.

According to Google Trends, Apache Hadoop reached its peak in popularity between 2014 and 2017 and, as in any technological cycle, its popularity may indeed gradually decline in favor of other emerging tools. However, the technology continues to progress, with a focus on the continuous evolution of HDFS, YARN, and MapReduce. Its latest version was released in March 2023.

Apache Hadoop Project Details

Apache Hadoop was developed predominantly in Java.

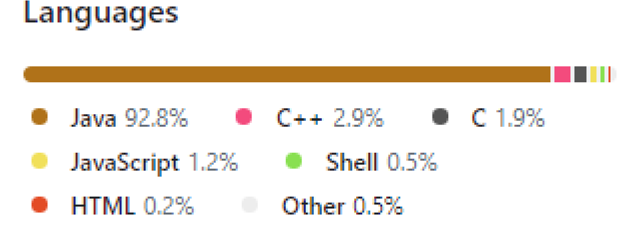


Figure 3 - Hadoop Languages

Source(s):

- https://hadoop.apache.org/
- https://cwiki.apache.org/confluence/display/hadoop
- https://github.com/apache/hadoop

© Copyright Tecnisys 2026. All rights reserved.

Apache HDFS

Distributed File System

A distributed file system (DFS) is a file system that enables client access to data stored on multiple servers, through a computer network, as if accessing local storage.

Files are distributed across multiple storage servers and in multiple locations, which allows for the sharing of data and resources.

DFS groups several storage nodes and logically distributes data sets on them, each with its own configuration. Data may reside on various types of devices, such as solid-state drives and hard disks.

Data is replicated, which facilitates redundancy to achieve availability.

To expand the infrastructure, the organization needs only to add more nodes to the system.

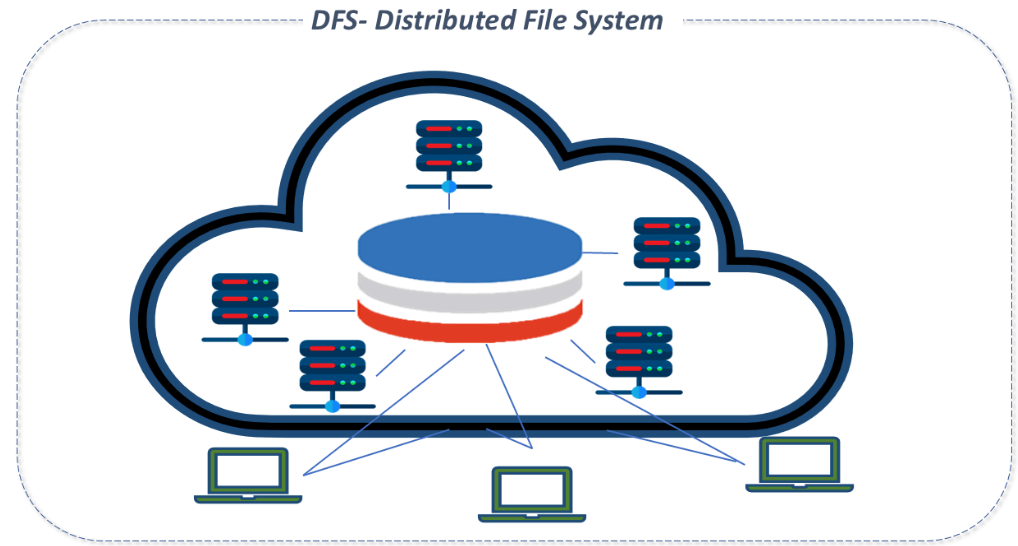


Figure 1 - DFS - Distributed File System

Apache HDFS Features

© Copyright Tecnisys 2026. All rights reserved.

HDFS (Hadoop Distributed File System) is the primary storage system used by Hadoop.

It is an extremely reliable distributed file system, designed to run on commodity hardware that works with high data transfer rates between nodes.

HDFS was originally built as infrastructure for the Apache Web Nutch search engine project and today is part of the core of the Apache Hadoop project. It has many similarities with existing distributed file systems. However, its differences are significant:

- it is highly fault-tolerant
- it is designed for low-cost hardware
- it offers high throughput for data access
- it is suitable for handling large data sets.

We highlight below some of its main features:

- **Fault tolerance**:
  An instance of HDFS can consist of thousands of servers storing part of the data of a file system, which involves a large number of components. This increases the likelihood of failures because, at some point, one or more components may become non-functional. For this reason, fault tolerance is the primary premise of HDFS. The detection of failures and quick, automatic recovery is its central architectural goal.

- **Streaming data access**:
  The data access pattern is streaming data, i.e., data generated in real time and in continuous flow.

- **Support for large data sets**:
  A common file in HDFS can easily reach terabytes in size, as applications running on HDFS handle large data sets. HDFS has been tuned to provide high aggregate bandwidth, scale to hundreds of nodes in a single cluster, and support tens of millions of files in a single instance.

- **Simple Coherence Model**:
  HDFS was designed based on the write-once-read-many principle. Once data is written, there is no support for updates at an arbitrary point or changes except for appends and truncates. This premise simplifies data coherence issues and ensures

© Copyright Tecnisys 2026. All rights reserved.

high throughput in data access. A MapReduce application or a web crawler fits perfectly into this model.

- **Interfaces that bring applications closer to data**:
  Computing requested by an application is always more efficient if executed close to the data it operates on. Especially when the dataset size is large, it minimizes network congestion and increases the overall system throughput. HDFS provides interfaces for applications to be close to the data.

- **Portability across heterogeneous hardware and software platforms**:
  HDFS was designed to be easily "portable" from one platform to another. This facilitates its widespread adoption as a platform for a large set of applications.

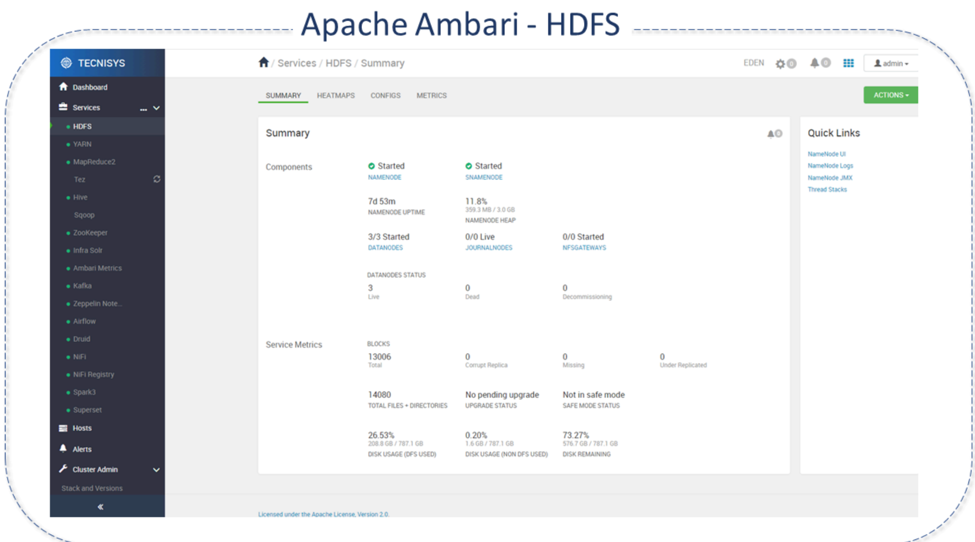


Figure 2 - Ambari/HDFS Interface

Architecture of Apache HDFS

The architecture of HDFS is a master/slave type, in which a single master (the Namenode) controls the operation of the others, the slaves (DataNodes).

HDFS divides a file into blocks, which are stored as independent units, facilitating the processes of distribution, replication, recovery, and processing.

- Namenode:
  The Namenode is the central piece of the HDFS file system. It is responsible for mapping the blocks to the DataNodes, determining where they and their replicas

© Copyright Tecnisys 2026. All rights reserved.

should be stored, and registering to which file each block belongs. The Namenode operates through the files:

- *fsimage:* which stores which blocks belong to each file
- *edit log:* which stores write operations on the system.

> **NOTE**
>
> When the system is initialized, the Namenode loads all the information from the fsimage into its memory, registering the current state of the system. It then records modifications through the edit log to update the system.

- Client applications interact with the Namenode whenever they want to locate or work with a file (copy, move, delete), and the Namenode responds to the requests by returning a list of servers where the data resides.

- When a client executes a write operation, it is recorded in the edit log, and then the filesystem namespace is modified (where the file information is stored). The filesystem namespace is stored locally.

- The Namenode performs operations on the filesystem namespace, such as opening, closing, renaming files, and directories. Additionally, any changes to the namespace are recorded by the Namenode.

- An HDFS cluster consists of a single Namenode, which greatly simplifies the system architecture.

- The community has developed several guidelines that are noteworthy regarding the Namenode:

  - Usage Recommendations for the Namenode
  - Namenode Does Not Start
  - ConnectionRefused

In the new HA (High Availability) architecture, the cluster consists of Namenodes in three distinct states: active, standby, and observer.

- **Datanodes**:
  The DataNodes are responsible for responding to read and write requests from the

© Copyright Tecnisys 2026. All rights reserved.

file system's clients and for storing the blocks.

- The DataNodes are spread throughout the cluster, usually one per node:
    - They are responsible for creating, deleting, and replicating blocks, under instructions from the Namenode.
    - Blocks are usually read from the disk, however, frequently accessed blocks may be cached within the DataNode. Thus, scheduling managers can run tasks on nodes where the block is cached, improving performance.

> **NOTE**
>
> A heartbeat indicates that the DataNode is operational. A block report indicates which blocks are stored on the DataNode. These data allow the Namenode to know in advance which DataNodes are available for storage and for reading, thus avoiding directing the client to DataNodes that are not operational.

Namenodes and Datanodes are software components designed to run on common machines. These machines generally run a GNU/Linux operating system.

- A typical deployment has one dedicated machine that runs only the NameNode software.
- Each of the other machines in the cluster runs an instance of the DataNode software.
- The architecture does not prevent multiple DataNodes from running on the same machine, but in a real deployment, this case is very rare.

© Copyright Tecnisys 2026. All rights reserved.

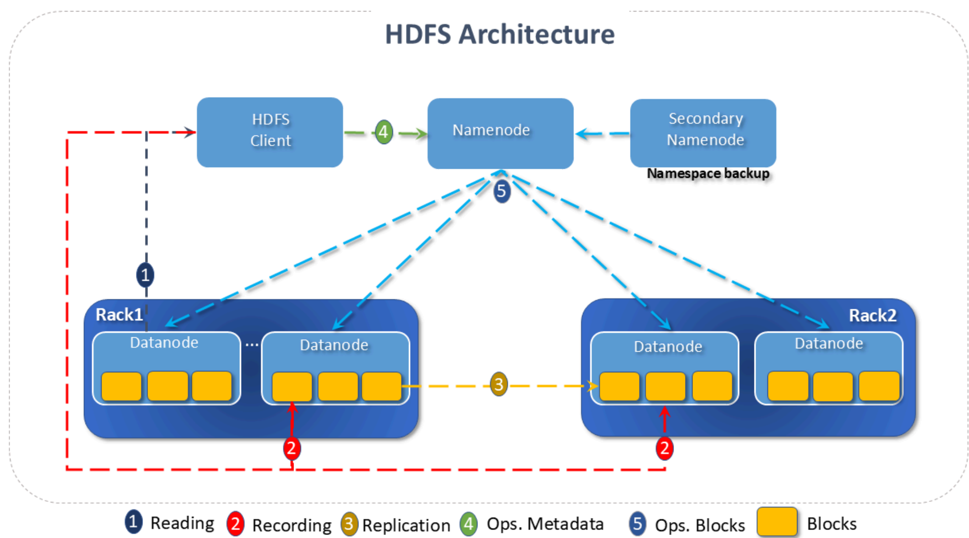


Figure 3 - HDFS Architecture

Apache HDFS Resources

- **File System Namespace:** HDFS supports the traditional hierarchical organization of files. A user or an application can create directories and store files within these directories.

>
>
> The file system namespace hierarchy is similar to most other existing file systems: one can create and delete files, move a file between directories, or rename it.

- **Userquotas (User Quotas Support)** Allows the administrator to set quotas for the number of names and the volume of space used for individual directories. Name quotas and space quotas operate independently, but their administration and implementation are parallel.

- **Access Permissions (Access Permissions Support)** Implements a permissions model for files and directories that shares much with the POSIX model.

- **Data Replication:** HDFS is designed to store large files across machines in a large cluster. Each file is stored as a sequence of blocks that are replicated to ensure fault tolerance. Block size and replication factor are configurable per file. All blocks of a file, except the last one, are the same size, and users can start a new block without

© Copyright Tecnisys 2026. All rights reserved.

filling the last one to the configured size, after support for variable-length block was added for append and hsync. An application can specify the number of replicas of a file. This number is called the replication factor and can be set at creation and modified later. The replication factor is stored by the NameNode, which is responsible for making all decisions about data replication. By default, the number of replicas is 3 (three), and the blocks are placed so that the first is on a different rack from the others, and the second is on the same rack as the third but on separate nodes. This way, the system can handle two types of failures: DataNode failures and rack failures. Additionally, this policy improves write operations by reducing bandwidth, considering that between distinct racks, it is lower than within the same rack since the data is in two distinct racks, not three, which would be the simplest policy. HDFS tries to satisfy requests with a replica closest to the client.

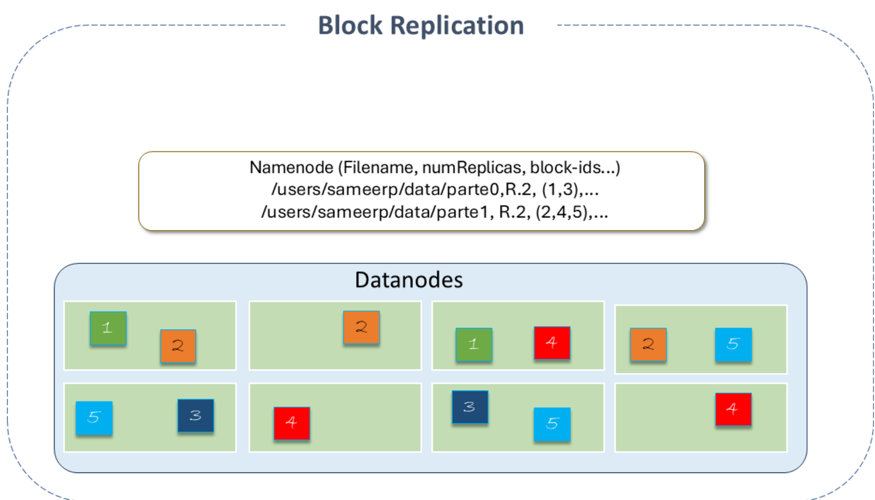


Figure 1 - Block Replication

- **Replica Placement:** The placement of replicas is critical to ensuring HDFS reliability and performance. Optimizing their placement distinguishes HDFS from most other distributed file systems. It is a feature that requires much tuning and experience. The implementation of the replica placement policy is a first effort in this direction.
  After support for Storage Types and Storage Policies was added to HDFS, the NameNode takes into account the policy for replica placement, besides the rack awareness.
  Details on this can be found in "Replicas Placement: The First Baby Steps" on the community site.

© Copyright Tecnisys 2026. All rights reserved.

- **Replica Selection:**
  To minimize global bandwidth consumption and latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there is one on the same rack as the reader node, this replica will be preferred to satisfy the request. If the HDFS cluster spans multiple datacenters, the replica in the local datacenter will be preferred over remote ones.

- **Block Placement Policies:** When the replication factor is 3 (three), the HDFS placement policy is to place one replica on the local machine, if the writer is on a DataNode, otherwise, on a random DataNode in the same rack as the writer, another replica on a node in a different remote rack, and the last on a different node in the same remote rack.
  When the replication factor is greater than 3, the placement of the additional replicas is determined randomly, keeping the number of replicas per rack below the upper limit (calculated as: *replicas-1/racks+2*). Additionally, HDFS supports **four distinct pluggable block placement policies**.
  Users can choose the policy based on their infrastructure and use case. By default, HDFS supports BlockPlacementPolicyDefault.

- **Safemode:** Upon startup, the NameNode enters an initial state called safemode. Block replication does not occur in this situation. The NameNode receives Heartbeat and Blockreport messages (with the list of data blocks a DataNode is hosting) from the DataNodes.
  Each block has a specified minimum number of replicas. A block is considered safely replicated when the minimum number of replicas for the data block is verified with the NameNode. After a configurable percentage of safely replicated data blocks check in with the NameNode (plus an additional 30 seconds), the NameNode exits safemode and then determines the list of data blocks (if any) that still have fewer than the specified number of replicas. The NameNode then replicates these blocks to other DataNodes.

- **HDFS Federation:** HDFS Federation adds support for multiple NameNodes/NameSpaces to HDFS.

- **Viewfs:** ViewFs (View File System) provides a way to manage multiple Namespaces (or Namespace volumes). It is particularly useful for clusters with multiple NameNodes and also multiple NameSpaces in HDFS Federation. It is analogous to client-side mount tables in Unix-Linux systems. It can be used to create personalized views of Namespaces and also common views across the cluster.

© Copyright Tecnisys 2026. All rights reserved.

- **Snapshots:** Snapshots are read-only "point-in-time" copies of the file system. Common use cases for snapshots are data backup, protection against user errors, and disaster recovery.

- **Edits Viewer:** Allows analysis of the edits log file. It operates on files only and does not require the Hadoop cluster to be running.

- **ImageViewer:** A tool that dumps the content of HDFS fsimage files into a "readable" format and provides a read-only WebHDFS API for offline analysis and examination of a Hadoop cluster's Namespace.

- **Centralized Cache Management:** Allows specifying paths to be cached by HDFS.

- **NFS Gateway:** Allows HDFS to be mounted as part of the client's local file system. It supports NFSv3.

- **Transparent Encryption:** HDFS implements end-to-end transparent encryption. Once configured, data read from and written to special HDFS directories are encrypted and decrypted transparently without requiring changes to the user's application code.

- **Multihomed Networks:** HDFS supports multihomed networks, where cluster nodes are connected with more than one network interface. There are several reasons to use it, such as security, performance, and fault tolerance.

- **Memory Storage Support:** HDFS supports writing to off-heap memory (managed by the developer) managed by DataNodes.

- **Synthetic Load Generator:** A testing tool that allows verifying the NameNode's behavior under different client loads.

- **Disk Balancer:** A command-line tool that distributes data evenly across all DataNode disks. It is different from the Balancer, which takes care of data balancing in the cluster. The Disk Balancer operates by creating and executing a plan on the DataNode. It is enabled by default in a cluster.

- **DataNode Administration Guide:** A tool that allows operations such as:

    - DataNode maintenance mode, created to allow repairs/maintenance.
    - Decommissioning DataNodes.

© Copyright Tecnisys 2026. All rights reserved.

- Recommissioning.

- **Router Federation:** Adds a software layer to "federate" the Namespaces, allowing users to access any sub-cluster transparently.

- **Provided Storage:** Allows data stored outside HDFS to be mapped and addressed from HDFS.

- **Observer NameNode:** In an HDFS cluster enabled for high availability (HA), there is a single Active NameNode and one or more Standby NameNodes. The Active NameNode is responsible for serving all client requests, while the Standby NameNode only keeps up-to-date information about the Namespace and block locations, receiving reports from all DataNodes. The Observer NameNode feature addresses these functions via a new NameNode called "Observer NameNode." Like the Standby NameNode, the Observer NameNode stays updated on the Namespace and block location information. But additionally, it has the ability to produce consistent reads like the Active NameNode. Since read requests constitute a large part of a typical environment, this helps balance the NameNode traffic load and improve overall throughput.

Important Notes

- Although HDFS follows the Filesystem naming convention, some paths and names (e.g., /.reserved and .snapshot) are reserved. Features like transparent encryption and snapshot use reserved paths.

- Currently, HDFS does not support hard links (the link acting as a pointer to a file or directory inode) or soft links (or symbolic link - acting as a pointer or reference to the file name). However, its architecture does not prevent the implementation of these features.

Details of Project Apache HDFS

HDFS was built in the Java language, highly portable, which makes it implementable on a wide variety of machines. Any machine that supports Java can run the NameNode or DataNode software.

Sources: Hadoop Documentation

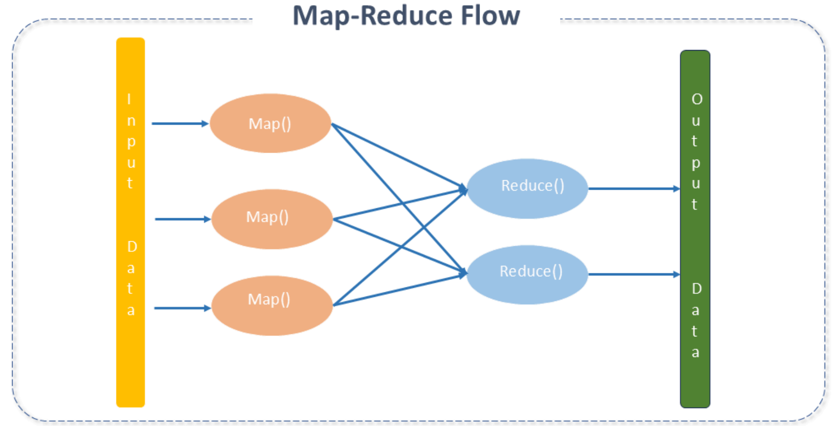
© Copyright Tecnisys 2026. All rights reserved.

Apache MapReduce

Apache MapReduce is a framework developed to write applications that process large amounts of data across large clusters of common hardware. It ensures reliability, parallelism, fault tolerance, and facilitates data locality (the ability for a program to run where the data is stored).

MapReduce was developed from a technology disclosed by Google, created by Jeffrey Dean and Sanjay Ghemawat, to optimize the indexing and cataloging of web data.

Despite Hadoop's evolution from its initial version, the high-level flow of the MapReduce processor has remained constant.



MapReduce Philosophy

Apache MapReduce Architecture

MapReduce organizes processing into two main processes — Map and Reduce — with several smaller tasks integrated into the process flow. A typical MapReduce job divides the input dataset into independent parts processed in parallel by Map tasks. The results are then sorted and passed to the Reduce tasks.

© Copyright Tecnisys 2026. All rights reserved.

Input and output of the job are usually stored in a file system. Typically, the compute and storage nodes are the same, allowing for efficient scheduling of tasks at the nodes where the data is stored, resulting in higher cluster bandwidth.

The MapReduce framework includes a single Resource Manager master, one NodeManager worker per cluster node, and one MRAppMaster per application.

Applications specify input and output locations and provide map and reduce functions through appropriate interfaces or abstract classes. These and other job parameters comprise the job configuration.

The client submits the job (jar/executable, etc.) and configuration to the ResourceManager, which manages distribution to workers, task scheduling, and monitoring, providing status and diagnostics.

Inputs and Outputs

MapReduce operates exclusively on key-value pairs, treating the input as a set of pairs and producing a set of pairs as output. Key and value classes must be serializable, implementing the WritableComparable interface to facilitate sorting.

Process Components

- **InputFileFormat**: The MapReduce process starts by reading the file stored in the HDFS, which can be of any type, and its processing is controlled by the *Inputformat*.

- **RecordReader and *Input Split***: The file is divided into "parts" known as *input splits*. Their sizes are controlled by the *mapred.max.split.size* and *mapred.min.split.size* parameters. By default, the size of the *input split* is the same as the block size and cannot be changed, except in very specific cases. For non-splittable format files like .gzip, the *input split* will be equal to the file size.

  The *RecordReader* function is responsible for reading data from the *input split* stored in HDFS. The default format is *TextInputFileFormat* and the RecordReader delimiter is /n, meaning only one line will be treated as a record by the RecordReader. Sometimes the behavior of the RecordReader can be customized by creating a custom RecordReader.

- **Mapper**: The *Mapper* class is responsible for processing the *input split*. The *RecordReader* function passes each read record to the *Map* function of the *Mapper*.

© Copyright Tecnisys 2026. All rights reserved.

The *Mapper* contains the *setup* and *cleanup* methods.

- The *setup* is executed before the *Mapper* processing, so any initialization operations must be performed within it.
- The *cleanup* method is executed once all records in the *input split* are processed, so any cleanup operations must be performed within it.

The *Mapper* processes the records and emits the output using the *object context*, which enables the Mapper and Reducer to interact with other Hadoop systems, allowing communication between *Mapper*, *Combiner*, and *Reducer*.

- The output pairs do not need to be of the same type as the input pairs.
- Applications can use the *counter* to report their statistics.
- The number of maps is usually determined by the total input size (total number of output file blocks).

- **Partitioner:** The *Partitioner* assigns a partition number to the record emitted by the *Mapper* so that records with the same key always get the same partition number, ensuring that records with the same key always go to the same *Reducer*.

- ***Shuffling* and *Sorting*:** The process of transferring data from the *Mapper* to the *Reducer* is known as *shuffling*. The *Reducer* starts *threads* to read data from the *Mapper* machine and reads all partitions belonging to it for processing using the HTTP protocol.

- **Reducer:** The number of *Reducers* that Hadoop can have depends on the number of Map outputs and several other parameters, and this can be controlled.
  The *Reducer* contains *reduce()* which is executed for each unique key emitted by the *Mapper*.

  - The total number of reduces is usually calculated as:

```
(0.95 or 1.75) * (number of nodes * max number of containers per node)
```

NOTE

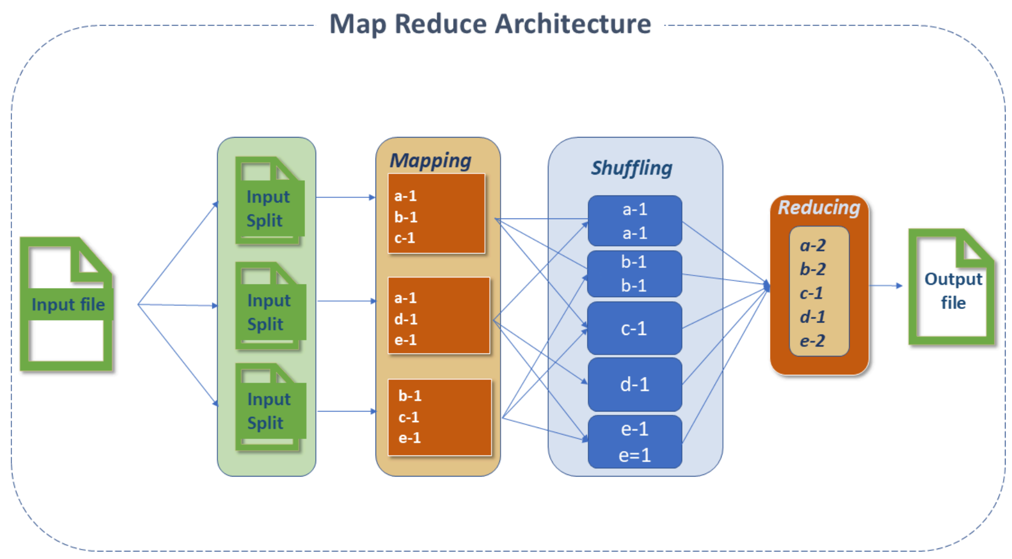
With 0.95, all reduces can start immediately and begin transferring map outputs as they finish. With 1.75, the faster nodes will finish their "first

© Copyright Tecnisys 2026. All rights reserved.

> round" of reduces and launch a second wave, doing a better job of load balancing.

- **Combiner**: Also called *mini reducer* or *localized reducer*, it is a process executed on the *Mapper* machines that takes the intermediate key emitted by the *Mapper* and applies the user-defined *Combiner* reduce function on the same machine.
  For each *Mapper* there is a *Combiner* available on the *Mapper* machines. The *Combiner* significantly reduces the volume of *data shuffling* between Mapper and *Reducer* and thus helps in performance improvement. However, there is no guarantee they will be executed.

- **Output format:** Translates the final *key/value* pair from the *Reduce* function and writes it to a file in HDFS via the *record writer*. By default, the output key values are separated by *tab* and the records are separated by a *newline* character. This can be modified by the user.



Apache MapReduce Architecture

Apache MapReduce Patterns

There are template solutions developed by people who solved specific problems and that can be reused:

- "Summarization" patterns:
  - Word Count.
  - Minimum and maximum

© Copyright Tecnisys 2026. All rights reserved.

- "Filtering" patterns:

  - Reduce "top-k" algorithm: A popular MapReduce algorithm where the mappers are responsible for emitting the top-k records at their level, and then the reducer filters the top-k records from all received from the mappers.
- "Join" patterns:

  - Reduce side join: Process where the "join" operation is performed in the "reducer" phase. Mapper reads the input data which is combined based on a common column or join key.

  - Composite Join:

    - Sorting and partitioning

Best Practices for Apache MapReduce

- **Hardware Configuration:**

  - Hardware configuration is very important for performance. A system with more memory will always perform better.
  - Bandwidth is also critical since MapReduce jobs may require data *shuffling* from one machine to another.
- **Operating System Tuning:**

  - *Transparent Huge Pages (THP):* Machines used in Hadoop should have THP disabled. THP does not work well in a Hadoop cluster and causes high CPU costs. It is recommended to disable it on each job node.

  - *Avoid unnecessary memory swapping:* In Hadoop, swapping can affect job performance and should be avoided unless absolutely necessary. The *swappiness* setting can be set to 0 (zero).

  - *CPU Configuration:* In most Operating Systems, the CPU is configured to save power and is not optimized for systems like Hadoop. By default, the *scaling governor* is set to power-saving mode and needs to be changed with the following command:

© Copyright Tecnisys 2026. All rights reserved.

```
cpufreq-set -r -g
```

- **Network Adjustments:** *Shuffling* consumes significant Hadoop time as it requires frequent master and worker connections. The *net.core.somax.conn* should be set to a higher value, which can be done by adding or editing `/etc/sysctl.conf` with the `net.core.somaxconn=1024`. entry.

- **File System Choice:**

    - The Linux distribution comes with a default file system designed for heavy I/O loads, which can significantly impact Hadoop performance. The latest Linux distribution comes with `EXT4` as the default file system, which performs better than `EXT3`.
    - The file system records the last access time for each read operation on the file, causing a disk write for each read. This setting can be disabled by adding a `noatime` attribute to the `file system mount` option. Some use cases have observed a performance improvement of over 20% with `noatime`.
- **Optimizations:**

    - *Combiner:* Shuffling data over the network can be expensive as transferring more data always takes more processing time.
      Reduce cannot be used in all use cases, but in most cases, we can use the *Combiner*, which reduces the size of the data transferred over the network during shuffling as it acts as a *mini reduce* and is executed on the *Mapper* machines.

    - *Map output compression:* The *Mapper* processes the output and stores it on the local disk.
      When generating a large amount of output, this intermediate result can be compressed with the *LZO* function, reducing disk I/O during shuffling.
      This is done by setting Isto é feito definindo o `mapred.compress.mapoutput` to `true`.

    - *Record filtering:* Filtering records on the *Mapper* side results in fewer data written to the local disk and faster subsequent stages (with less data to operate on).

© Copyright Tecnisys 2026. All rights reserved.

- *Avoiding many small files:* Very small files can take a long time to execute. HDFS stores these files as a separate block, which can overload file processing with the initialization of many *Mappers*.
  It is good practice to compact small files into a single large file and then run MapReduce on it.
  In some cases, this can result in a 100% performance optimization.

- *Avoiding non-splittable file formats:* Non-splittable formats (e.g., gzip) are processed all at once.
  If they are very small files, it will take a lot of time because for each file, a Mapper will be started.
  The best practice is to use a splittable format like Text, AVRO, ORC, etc.

- **Runtime Configurations**

  - *Java Memory: Map* and *Reduce* are JVM processes that, therefore, use JVM memory for execution.
    The memory size can be adjusted in the `mapred.child.java.opts` property.

  - *Map spill memory:* The output records of the *Mapper* are stored in a circular *buffer* whose default size is 100MB.
    When the output exceeds 70% of this size, the data will be written to disk.
    To increase the *buffer* memory, use the `io.sort.mb` property.

  - *Map adjustment:* The number of *Mappers* is controlled by the *mapred.min.split.size*.
    If there are many tasks running one after another, it is ideal to set `mapred.job.reuse.jvm.num.tasks to -1`.

WARNING

This should be used very carefully as in the case of long-running tasks, JVM overhead will not increase performance, but rather the opposite.

- **File System Optimization:**

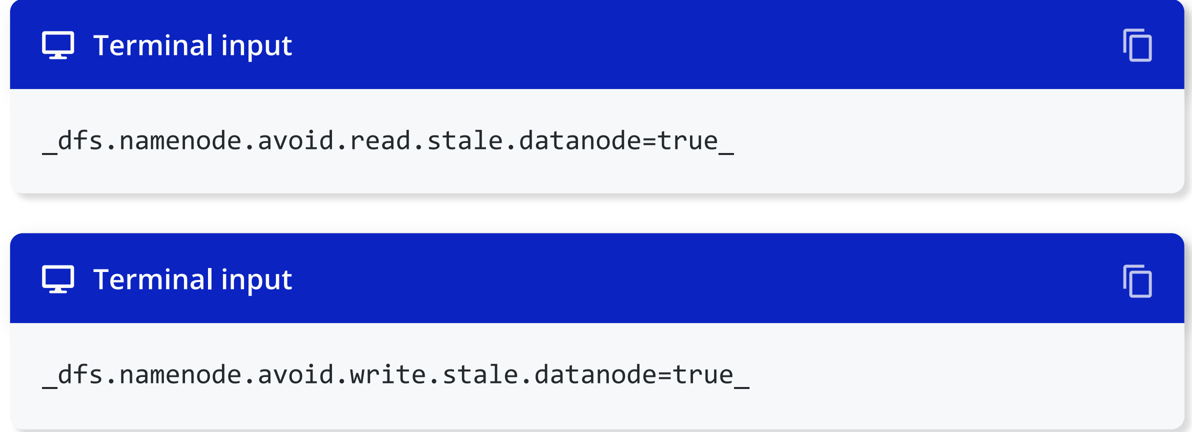
  - *Mount option:* There are some efficient mount options for Hadoop clusters, such as the *noatime* configured for `Ext4` and `XFS`.

© Copyright Tecnisys 2026. All rights reserved.

- *HDFS block size:* Block size is important in NameNode performance and job execution.
  The NameNode maintains metadata for each block it stores in the DataNode and therefore occupies a lot of memory with block sizes much smaller than recommended.
  The recommended value for *dfs.blocksize* should be between `134,217,728` and `1,073,741,824`.

- *Short circuit read:* The HDFS read operation goes through the DataNode, which, after the client's request, sends the file data over the TCP *socket* to the client.
  In short circuit read, the client reads the file directly, thus bypassing the DataNode.
  This only happens if the client is on the same node as the data.

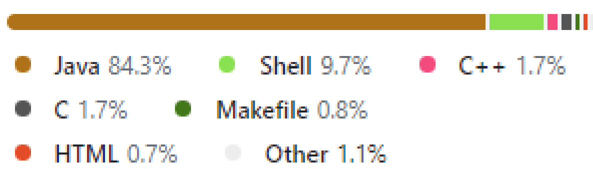
- *Stale Datanode:* The DataNode sends a *heartbeat* to the NameNode at regular intervals to indicate it is active.
  To avoid sending read and write requests to inactive DataNodes, adding the following properties in *hdfs-site.xml* is efficient:



Apache MapReduce Project Details

Although Hadoop is implemented in Java, MapReduce applications can use any language that supports Hadoop Streaming or Hadoop Pipes.

© Copyright Tecnisys 2026. All rights reserved.



MapReduce Languages

Sources:

- Hadoop.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache YARN

Resource Management

Big Data Processing is a challenging task and is not feasible in a centralized system. Distributed computing solutions need to be adopted to enable the parallel processing required by the technology.

In this environment, multiple tenants with different demands can share computing resources such as data, storage, network, memory, and CPU, making resource management a critical task in this technology.

Big Data users can request multiple processing jobs, each with different requirements.

The Resource Manager acts as a "scheduler," which schedules and prioritizes requests according to the requirements. It essentially acts as a referee, managing and allocating available resources.

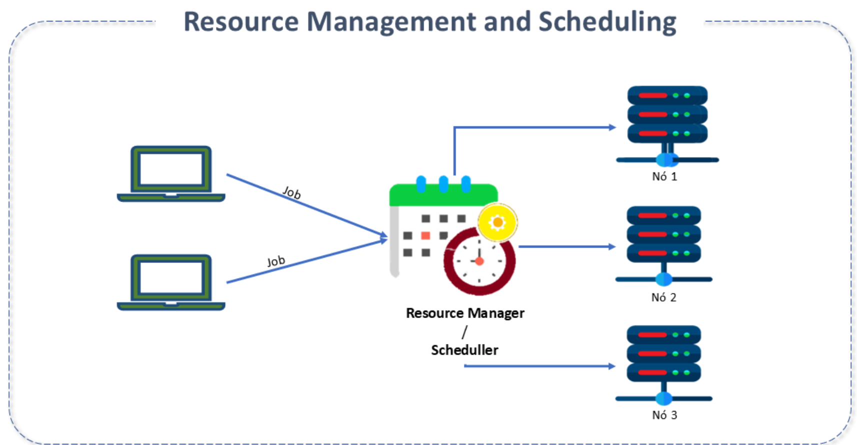
In the early versions of Hadoop, there was no Resource Manager. Hadoop consisted of two main parts:

- **Storage:** Handled by HDFS
- **Processing:** Handled by MapReduce. The only job that could be executed and submitted to Hadoop until version 2.

  The tasks of resource allocation and scheduling were handled by the MapReduce JobTracker service.

With the evolution of solutions where real-time and near-real-time processing became predominant, it became essential to have an application executor and a resource manager to schedule and execute all types of applications, including MapReduce, in real time.

© Copyright Tecnisys 2026. All rights reserved.



Resource Management and Scheduling

Apache YARN Features

The Yet Another Resource Negotiator (YARN) was introduced in Apache Hadoop 2.0 to meet these needs and also solve scalability and management capacity issues that existed in the previous version of Hadoop. It is responsible for helping manage resources among the Clusters, taking on the responsibility for scheduling and resource allocation for the Hadoop System, tasks previously performed by the MapReduce JobTracker.

Today, Apache YARN has gained popularity due to the advantages it offers in scalability and flexibility, as well as its versatility and low cost, as it can be used on common hardware. It is successfully implemented at eBay, Facebook, Spotify, Xing, Yahoo, etc.

Among its main features, we highlight:

- **Multitenancy:** A comprehensive set of limits is provided to prevent a single application, user, or queue from monopolizing the queue or Cluster resources, thus avoiding Cluster overload.

- **Docker for YARN:** The Linux Container Executor (LCE) allows the YARN NodeManager to start YARN containers for execution directly on the host machine or within Docker containers.

  Docker containers provide a custom execution environment in which the application code runs, isolated from the NodeManager's execution environment

© Copyright Tecnisys 2026. All rights reserved.

and other applications.

Docker for YARN provides consistency (every YARN container will have the same software environment) and isolation (no interference with what is installed on the physical machine).

- **Scalability:** Apache YARN's scalability is known for scaling to thousands of nodes. It is determined by the Resource Manager and is proportional to the number of nodes, active applications, active containers, and heartbeat frequency (of nodes and applications).

- **High availability for components:** Fault tolerance is a fundamental principle of Apache YARN. This responsibility is delegated to the Resource Manager (NodeManager and Application Master failures) and ApplicationMaster (container failures).

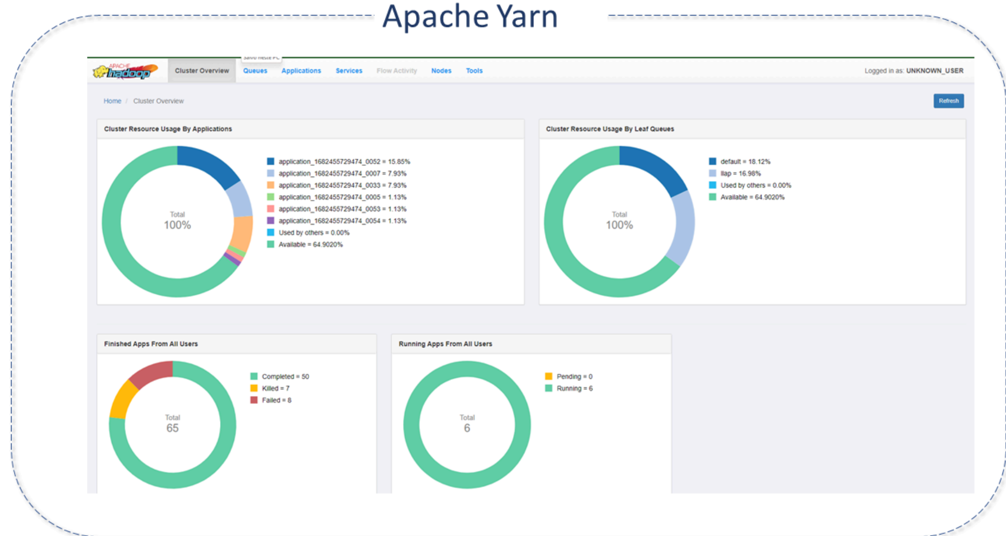
- **Flexible Resource Model:** In Apache YARN, a resource request is defined in terms of memory, CPU, locality, etc., resulting in a generic definition. The capacity of NodeManager and Worker nodes is calculated based on the installed memory and CPU cores.

- **Multiple data processing algorithms:** Apache YARN was developed to run on a wide variety of data processing algorithms. It is a framework for generic resource management and allows the execution of various algorithms over data.

- **Log aggregation and resource localization:** To manage user logs, Apache YARN uses log aggregation. Once the application is completed, the NodeManager service aggregates the user logs related to an application and writes the aggregated logs into a single log file in HDFS.

- **Efficient and reliable resources:** Apache YARN provides a generic resource management framework with support for data analysis through various data processing algorithms.

© Copyright Tecnisys 2026. All rights reserved.



YARN Interface

Apache YARN Architecture

Apache YARN consists of three components:

- **Resource Manager:** It is the master node, responsible for resource management in the Cluster. It is the "ultimate authority" that arbitrates resources among all applications in the system.

  There is one Resource Manager per Cluster, and it "knows" the location and resources of all slaves, which includes information such as GPU, CPU, and memory needed to run applications. The Resource Manager acts as a proxy between the client and all other nodes.

  It has two main components:

  - **Scheduler:** Responsible for resource allocation for the various running applications.

    The Scheduler does not monitor or track the application status and does not provide guarantees about restarting tasks affected by application or hardware failure. Its function is performed based on the resource requirements of the applications and the abstract notion of a resource container, which incorporates elements such as memory, CPU, disk, network, etc.

© Copyright Tecnisys 2026. All rights reserved.

> **NOTE**
>
> Containers are portions of resources (CPU, memory, etc.) of a Cluster machine that can be reserved for running an application, whether this application is Hadoop Map Reduce or not.

Starting from Hadoop version 2.0, the coordination of a Job does not occur on the Master node but in a container instantiated on a worker machine, which usually has two components (it may have three, in the case of the application management instance): Node Manager, Application Master, and DataNode.

The Scheduler has a "pluggable" policy responsible for partitioning the Cluster resources among various queues, applications, etc.

The Scheduler receives requests from the Application Masters for resources and performs its scheduling function. The scheduling strategy is configurable and can be chosen based on the needs of each application.

There are, by default, three schedulers in Apache YARN:

- **Scheduler FIFO:** Uses the simple first-in, first-out strategy. Memory is allocated based on the request time sequence, with the first application in the queue receiving the necessary memory, then the second, and so on. If memory is not available, the applications will wait for availability. In this option, Apache YARN creates a request queue, adding applications to it and starting them one by one.

- Capacity Scheduler: Ensures that the user gets the minimum amount of resources. Helps economically share the Cluster's resources among different users. In other words, the Cluster's resources are shared among various user groups. It works with the concept of queues. A Cluster is divided into partitions (queues), and each queue receives a percentage of resources.

- Fair Scheduler: All applications get almost identical amounts of available resources. When the first application is sent to Apache YARN, it will allocate all available resources to it. If a new application is sent, Apache YARN starts allocating resources to it until both have almost the same amount. Unlike

© Copyright Tecnisys 2026. All rights reserved.

the previous ones, the Fair Scheduler prevents applications from running out of resources and ensures that all applications in the queue get the necessary memory for execution.

  - **Application Manager:** The main task of the Application Manager is to accept job submissions, negotiate the first container for executing the specific Application Master, and provide the service to restart the ApplicationMaster container when it fails.
- NodeManager: Responsible for starting and monitoring job containers. It is the machine-level framework responsible for the containers, monitoring their resource usage and reporting to the ResourceManager/Scheduler.

  - **Containers:** Hadoop 2.0 improved its parallel processing with the addition of containers, which are an abstract concept that supports multitenancy on a data node.

    This was how Hadoop found to define memory, CPU, network requirements: dividing the resources on the data server into containers. Thus, the data server can host multiple jobs, hosting multiple containers.

    The Resource Manager is responsible for scheduling resources by allocating containers.

    This is done based on an algorithm, from the input provided by the client, the Cluster's capacity, and the resource queues and prioritizations in the Cluster.

    A general rule is to start the container on the same node as the data required by the job to facilitate its location.

- **Application Master:** It is a framework-specific library responsible for negotiating resources from the ResourceManager and working with the NodeManager(s) to run and monitor jobs. Upon receiving an application, the ResourceManager will start the ApplicationMaster in an allocated container, which will communicate with the YARN Cluster to handle the application's execution. The main tasks of the ApplicationMaster are:

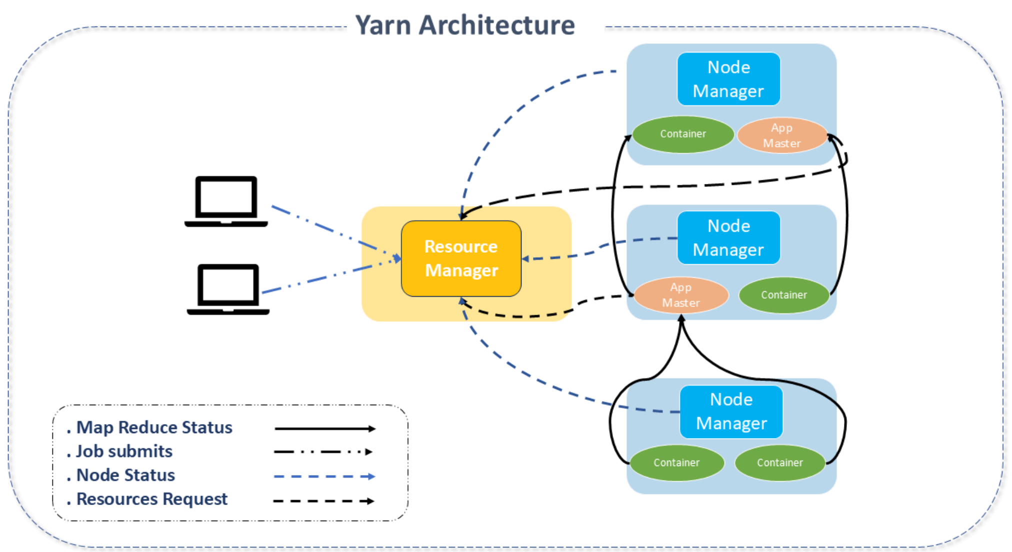
  - Communicate with the ResourceManager to negotiate and allocate resources for future containers.
  - After container allocation, communicate with NodeManagers to start the application's containers.

© Copyright Tecnisys 2026. All rights reserved.

NOTE

The Resource Manager and the Node Manager form the data computing framework.

The main idea of Apache YARN is to separate resource management and job scheduling/monitoring functionalities into separate daemons, thus existing a global ResourceManager (RM) and an ApplicationMaster (AM) per application. An application is a single job or a Directed Acyclic Graph (DAG) of jobs.



YARN Architecture

NOTE

With the implementation of Apache YARN in Hadoop version 2.0, MapReduce maintains compatibility with the previous stable version (Hadoop version 1.x). All MapReduce tasks continue to be executed in Apache YARN.

Apache YARN Supported Features

- Extensible Resource Model: Apache YARN supports an extensible resource model. By default, it tracks CPU and memory for all nodes, applications, and queues, but the resource definition can be extended to include "accountable" resources (resources consumed while the container is running but released later, such as memory and CPU), which means that the resource definition can be extended from

© Copyright Tecnisys 2026. All rights reserved.

its default values (such as CPU and memory) to any type of resource that can be consumed when the task "runs" in the container. Additionally, YARN supports the use of "resource profiles," allowing a user to specify multiple resource requests through a single profile, similar to Amazon Web Services Elastic Compute Cluster instance types. It is an easy way to request resources from a single profile and a means for administrators to regulate consumption.

- Resource Reservation: Apache YARN supports resource reservation through the ReservationSystem, a component that allows the user to specify a resource profile over time and temporal constraints (e.g., deadlines) and reserve resources to ensure predictable execution of important jobs. The ReservationSystem tracks extra resources, performs admission control for reservations, and dynamically instructs the underlying Scheduler to ensure the reservation is met.

- Federation: Apache YARN supports Federation through the YARN Federation feature, which allows transparently connecting multiple YARN (sub)Clusters and making them appear as a single massive Cluster. It can be used to achieve greater scale and/or allow multiple independent Clusters to be used together for very large jobs or for tenants who have capacity across all of them. The approach is to divide a large Cluster into smaller units called sub-clusters. Each unit has its own YARN Resource Manager and compute-node (Node_manager).

- REST APIs: Apache YARN offers RESTful APIs to allow client applications to access different metric data such as Cluster metrics, schedulers, nodes, application state, priorities, resource managers, etc. It also provides information and statistics about the NameNode instance and statistics about applications and containers. These services can be used by remote monitoring applications. Currently, the following components support RESTful information:

    - Resource Manager
    - Application Master
    - History Server
    - Node Manager.
- High Availability: A failure in the resource manager will cause a failure in Apache YARN, and therefore, it is important to implement high availability for the Resource Manager. The High Availability (HA) feature adds redundancy to remove single points of failure.

© Copyright Tecnisys 2026. All rights reserved.

- **High Availability Resource Manager Architecture:** The high availability of the Resource Manager works in an Active/Standby architecture. The Standby Resource Manager takes over when it receives the signal from ZooKeeper.

- **High Availability Resource Manager Features:**

- **Resource Manager state storage:** In case of failure, the Standby Resource Manager will reload the storage state and start from the last execution point. Cluster information will be rebuilt when the NodeManager sends a heartbeat to the new Resource Manager.

- Resource Manager restart and failover: The Resource Manager loads the internal application state from the Resource Manager state storage. The Resource Manager's Scheduler rebuilds its cluster information state when the NodeManager sends a heartbeat. The checkpoint process avoids restarting already completed tasks.

- **Failover and fencing:** In high availability YARN Clusters, there may be two or more Resource Managers in active/Standby mode. A split brain can occur when two Resource Managers assume themselves as active. If this happens, both will control Cluster resources and handle client requests. Failover fencing allows the active Resource Manager to restrict the operation of others. The previously discussed state storage provides the ZooKeeper-based ZKResourceManager StateStore, which allows only a single Resource Manager to write at a time.

- **Leader Elector:** Based on the ZooKeeper ActiveStandbyElector, it is used to elect a new active Resource Manager and implement fencing internally. When a Resource Manager becomes inactive, a new Resource Manager is elected by the ActiveStandbyElector and will take over. If automatic failover is not enabled, the administrator must manually transition the active RM to standby mode and vice versa.

- Nodelabels: It is a marker for each machine, so that machines with the same label name can be used in specific jobs.

  Nodes with more powerful processing resources can be labeled with the same name, and then jobs requiring more machine power can use the same label during submission. Each node can have only one label assigned to it, meaning the Cluster will have a disjoint set of nodes. We can say that a cluster is partitioned based on node labels.

© Copyright Tecnisys 2026. All rights reserved.

Apache YARN also provides features to define queue-level configuration, which defines how much of a partition or queue can be used. There are currently two types of node labels available:

  - **Exclusive:** ensures that it is the only queue allowed to access the node label. The application submitted by the queue with an exclusive label will have exclusive access to the partition so that no other queue can obtain resources.

  - **Non-exclusive:** allows sharing of idle resources with other applications. Queues receive node labels, and applications submitted to these queues will have priority over the respective node labels. If there is no application or job in a queue for these labels, resources will be shared among other non-exclusive node labels. If the queue with the node label submits an application or job during processing, resources will be taken from running tasks and assigned to the associated queues based on priority.

- Node Attributes: They are a way to describe attributes of a node without guaranteeing resources. They can be used by applications to select the correct nodes for their container based on the expression of several such attributes.

- Proxy Web Application: Its main purpose is to reduce the possibility of web-based attacks through Apache YARN.

- Timeline Server and Timeline Server 2: The Timeline Server is responsible for storing and retrieving current and historical application information. It basically has two responsibilities:

  - Persistence of application-specific information:
  - Persistent generic information about completed applications.

  Timeline Server 2 is the next major iteration of Timeline Server, created to address scalability issues and improve usability over Timeline Server 1.

Apache YARN Project Details

Apache YARN was developed in Javascript, Shell, PHP, TypeScript, and HTML.

© Copyright Tecnisys 2026. All rights reserved.

Top languages

JavaScript Shell PHP TypeScript

HTML

YARN Languages

Source(s): Hadoop.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Flink

Distributed Computing Engines

The concept of distributed computing is not new and has become even more well-known with the "explosion" of data in recent years and the growing need to process large flows of information with predictive capabilities.

Among its main benefits, we can mention the ability to add more processing nodes (scalability), in proportion to the growth of data, without compromising the performance or availability of systems, and the ability to process in real-time - the distribution of work across several nodes streamlines and provides efficiency to decision-making.

In the Big Data era, several programming models and frameworks emerged for executing batch jobs in an optimized and distributed way, such as Map Reduce and others, with the ability to process data on a large scale, in parallel, and with fault tolerance, which are very useful for ETL (Extract, Transform, Load), like Apache Hadoop, Apache Spark, Apache Beam, and Apache Flink.

Another way of processing data arising from the Big Data era is streaming, a type of data processing "engine" designed to handle infinite datasets (data arises infinitely and there is no guarantee of the order of arrival). Among the main distributed processing frameworks/engines coded for streaming, we can mention Apache Flink, Apache Storm, Apache Flume, and Apache Samza. These mechanisms receive messages by reading a messaging system (source) - such as Apache Kafka - process them in real-time, filtering those that contain the desired data, and sending them to an output.

Apache Flink

Apache Flink is a framework and distributed processing engine for stateful computation (functions that assume a state and return a new state) of unlimited and limited data

© Copyright Tecnisys 2026. All rights reserved.

streams. It was created by the research team at Berlin Technical University, led by Stephan Ewen. It started as an academic project called Stratosphere, around 2009. In 2014, the project was incorporated into the Apache Software Foundation, becoming an incubation project. It was later promoted to a top-level Apache project. Since then, Flink has evolved significantly, becoming one of the leading platforms for real-time data stream processing.

Apache Flink Features

Apache Flink supports multiple notions of time for state-based stream processing.

Among its main features, we highlight:

- **State processing:** Manages complex states, even in large-scale operations.
- **Correctness and consistency:** Offers data accuracy guarantees (exactly-once).
- **Event-time processing:** Allows handling real-time data, considering the time when events occur.
- **Layered APIs:** Modular and flexible approach to building applications. It provides several APIs at different levels of abstraction, including SQL, DataStream, and DataSet:
    - **DataStream API:** Used for real-time data stream processing applications.
    - **DataSet API:** Aimed at batch data processing.
    - **SQL and Table API:** Allows users to write queries in SQL or a similar table language for data manipulation.

  These APIs allow developers to choose the level of abstraction that best suits their needs.

Apache Flink Architecture

The architecture of Apache Flink is composed of several components:

- **JobManager:** Coordinates the distribution of tasks and resource management. It also handles checkpointing to ensure fault tolerance.
- **TaskManager:** Executes the tasks (or jobs) effectively. Each TaskManager can execute multiple tasks simultaneously.

© Copyright Tecnisys 2026. All rights reserved.

- **Dispatcher:** Offers a REST interface to submit Flink applications and start a new JobMaster for each job, in addition to running the Flink WebUI.
- **Flink Client:** Prepares and sends a data stream to the JobManager. It can disconnect or remain connected to receive progress reports.
- **Resource Manager:** Responsible for resource allocation and provisioning in the Flink cluster, managing task slots.
- **JobMaster:** Manages the execution of a single JobGraph.
- **Sources and Sinks:** Connect with external storage systems for data input and output.

This architecture allows for distributed and scalable processing of large volumes of data. For more detailed information, visit the official Apache Flink documentation at Apache Flink Architecture.

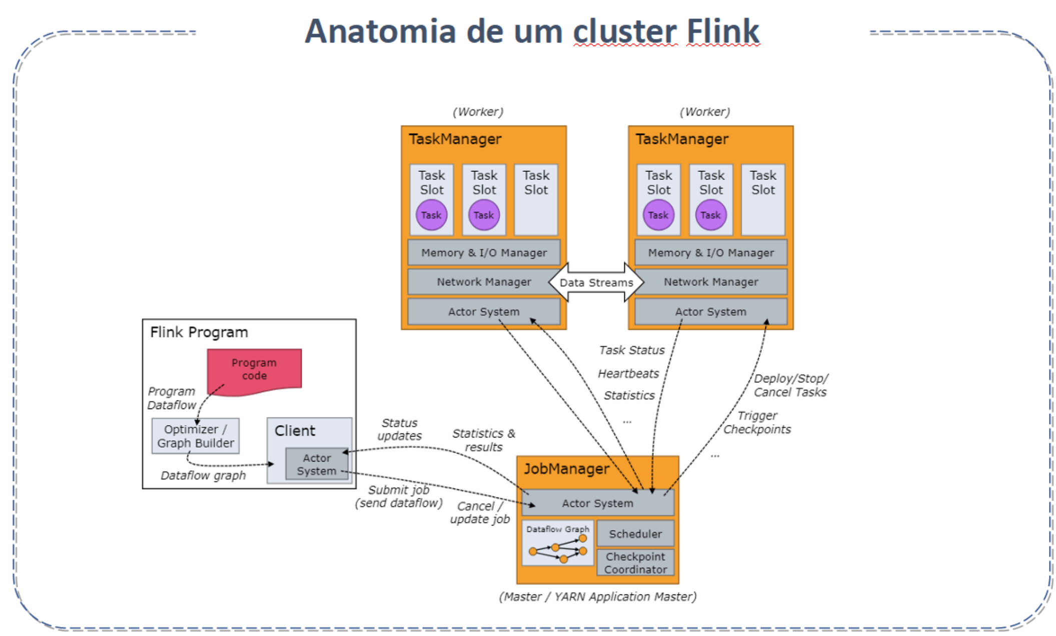


Figure 1 - Anatomy of a Flink cluster

Apache Flink Resources

- **Stream and batch analysis:** Supports both real-time and batch processing.
- **Event-driven applications:** Ideal for systems that react to events in real-time.
- **High performance:** Designed to be fast, with low latency and high throughput.

When to use Apache Flink

- Apache Flink is ideal for applications that require real-time data stream processing and event analysis.

© Copyright Tecnisys 2026. All rights reserved.

- It is recommended for scenarios with fault tolerance requirements and data consistency guarantees.
- It is suitable for systems that require large-scale data processing with low latency.

When Apache Flink is not applicable

- Apache Flink is not the best option for simple data manipulation projects where lighter solutions, such as scripts or traditional ETL tools, would suffice.
- It can be excessive for applications that do not require advanced data stream processing or real-time analysis capabilities.
- It is less efficient for tasks where batch processing, without the need for real-time processing, is sufficient.

Main differences between Apache Spark and Apache Flink

- **Processing Model:** One of the main differences between the two tools is the processing model. Spark uses the RDD (Resilient Distributed Datasets) data processing model - a distributed and immutable collection of objects. Apache Flink uses the stream-based data processing model - a collection of events processed in real-time as they arrive.
- **Speed:** Apache Flink is generally faster than Apache Spark for real-time data processing, because its processing model eliminates the need to transform data into RDDs before processing it.
- **Fault tolerance:** Both tools are fault-tolerant, but Spark uses a disk-based checkpointing mechanism to maintain data integrity, and Flink uses a memory-based approach, storing data in cache, which allows faster recovery in case of failures.
- **Support for complex events:** Apache Flink offers native support for complex event processing, which allows the analysis of patterns and correlations in real-time data streams, unlike Spark, which requires additional libraries and configuration adjustments.

In short, Apache Spark is often used for batch data processing and applications that do not require very low latency, such as log analysis and historical data processing. Apache Flink is often used for real-time data processing, such as fraud analysis and IoT monitoring.

Main differences between Apache Storm and Apache Flink

The comparison between Apache Flink and Apache Storm involves several aspects:

© Copyright Tecnisys 2026. All rights reserved.

- **Use:** Flink is more for unified batch and stream processing, while Storm is focused on real-time stream processing.
- **Data Processing:** Apache Flink offers batch and stream data processing; Storm is great for stream processing.
- **Data Transformation:** Apache Flink has rich options for data transformation; Storm is more limited.
- **Machine Learning Support:** Apache Flink supports Machine Learning, while Apache Storm does not.
- **Query Language:** Apache Flink uses a language similar to SQL; Storm does not have a native query language.
- **Deployment Model:** Both offer different deployment options.
- **Integration with Other Services:** Both integrate well with other services.
- **Scalability:** Apache Storm excels in high scalability.
- **Performance and Reliability:** Both have good performance and are reliable.

Best Practices for Apache Flink

- **Efficient State Management:** Structure and manage states efficiently to optimize performance and ensure data consistency.
- **Performance Optimization:** Monitor performance metrics, fine-tune settings, and use data partitioning strategies to maximize efficiency.
- **Failure Recovery Strategies:** Implement robust checkpointing and recovery strategies to ensure fault tolerance.
- **Scalability and Load Balancing:** Develop applications with scalability in mind, ensuring effective load balancing between cluster nodes.
- **Testing and Validation:** Perform extensive testing to ensure the reliability and accuracy of Flink applications in different scenarios.
- **Updates and Maintenance:** Stay up-to-date with the latest Flink versions and best practices to leverage improvements and fixes.

Apache Flink Project Details

Apache Flink is predominantly used with the Java programming language. It also offers support for Scala, as its Scala API integrates seamlessly with the Java API. Additionally, for data query operations, Flink supports SQL through its Table and SQL API. This flexibility allows developers to choose the language that best suits their needs and preferences.

Sources:

© Copyright Tecnisys 2026. All rights reserved.

- Apache Flink

© Copyright Tecnisys 2026. All rights reserved.

Great Expectations

Data Validation and Quality Assurance

This feature is only available starting from version 2.3.

In the context of Big Data and data pipelines, data validation plays an essential role.

It is a carefully planned process to ensure that the data collected or processed meets previously defined standards, criteria and formats.

Before moving on to analytics, reporting, or machine learning models, data goes through a rigorous "fine-tooth comb," ensuring its quality and consistency. This validation is a crucial checkpoint, as incorrect data can lead to disastrous decisions, misguided analysis, and, in the case of regulated organizations, even legal implications.

The importance of this process goes beyond simply identifying errors. It is the foundation that ensures organizational trust in your data.

Relying on data that hasn't been verified—simple errors like missing values or inconsistencies between tables can compromise entire results. In addition, automating the validation process reduces manual intervention, making it more efficient and preventing bottlenecks in complex pipelines. When done correctly, it also ensures compliance with regulatory requirements and internal policies, indispensable in industries such as healthcare, finance, and technology.

© Copyright Tecnisys 2026. All rights reserved.

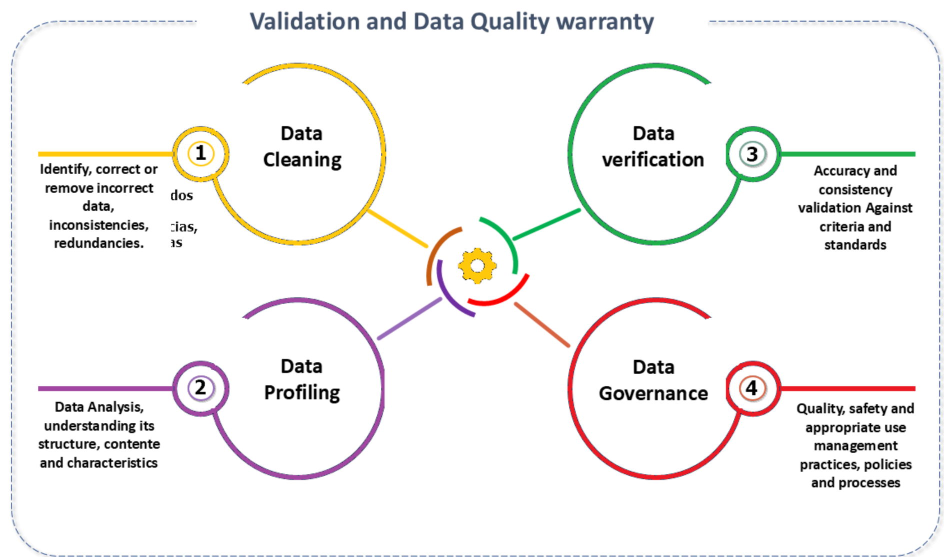


Figure 1 - Quality Assurance

Features of Great Expectations

Great Expectations (GX) is an open-source tool designed to make the entire process of validating and ensuring data quality more accessible, automated, and efficient. It solves challenges faced by data science and engineering teams when dealing with complex pipelines, such as a lack of visibility into data quality and difficulties integrating continuous checks into workflows.

At the heart of Great Expectations is the concept of "expectations." These expectations are essentially rules that define how data should behave. They can be as simple as "this column must not contain null values," or as sophisticated as "the values in this column must be in a range between 0 and 100, with an expected average of 50."

The flexibility to define specific rules makes GE a powerful tool to meet diverse data validation needs.

One of the most outstanding features of the tool is its ability to generate interactive reports, known as Data Docs. These reports not only document the quality of the data, but also allow for a visual analysis of the validations performed, facilitating communication between technical and non-technical teams. They are ideal for promoting transparency in organizations where data quality is critical.

Great Expectations Architecture

© Copyright Tecnisys 2026. All rights reserved.

The architecture of Great Expectations is designed to be modular and adaptable. It is composed of three main elements:

- **Expectations:** Represent the rules that the data must meet. For example, "the values in the 'age' column must be between 0 and 120."
- **Validation Results**: Indicate whether or not data meets expectations, providing detailed compliance metrics.
- **Data Docs**: These are automatically generated interactive reports that provide a clear and accessible view of the quality of the validated data.

These components work together to create a workflow that begins with setting expectations, goes through data validation, and culminates in generating reports that document and monitor data quality over time.

Figure 2 - Great Expectations - Data Validation

Figure 2 - Great Expectations - Data Validation

Great Expectations Philosophy

The philosophy of Great Expectations is simple and transformative: to enable organizations to define and validate what they expect from their data in a structured and reusable way.

It formalizes these expectations in a configurable format, often represented in JSON, allowing for easy maintenance and integration with other tools.

With a catalog of more than 70 types of predefined expectations, GE covers most data validation needs.

In addition, it is possible to create personalized expectations for specific cases, ensuring flexibility and scalability.

© Copyright Tecnisys 2026. All rights reserved.

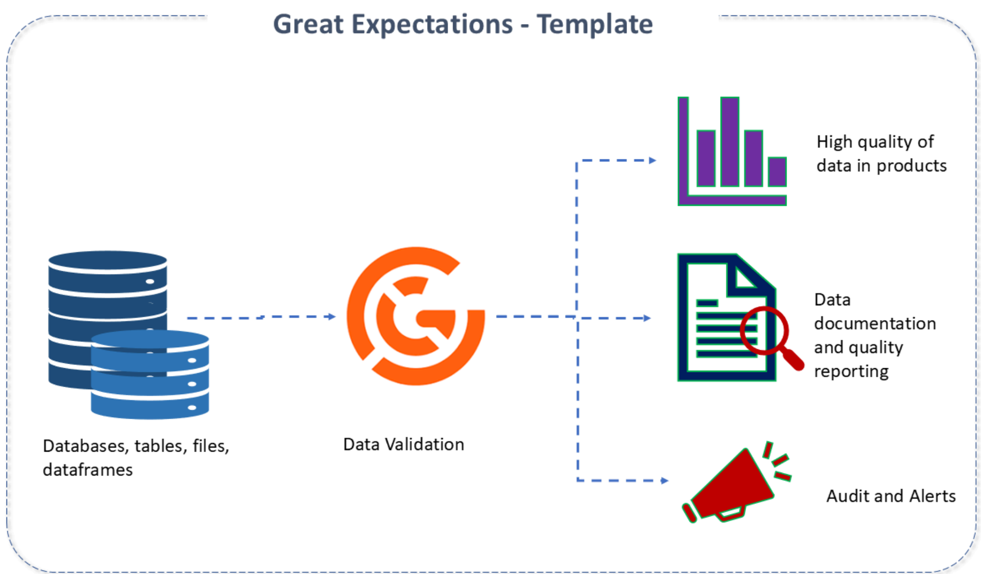


Figure 3 - Great Expectations Model

Great Expectations Features

Great Expectations offers a number of features that make it stand out as a robust solution for data validation:

- **Broad Data Source Support*: Supports CSV files, Parquet, relational databases, and data lakes.
- **Local or Cloud Execution*: Flexibility to run in different environments.
- **Simple and intuitive setup*: User-friendly interfaces to set and manage expectations.
- **Active Community*: Constant updates and support from the community.

Great Expectations is more than a validation tool; It is a milestone in the evolution of data quality assurance, offering clear solutions to complex challenges.

Details of Project Great Expectations

Great Expectations is built on Python, taking advantage of the language's popularity and versatility to make it easier to integrate into data pipelines. Python is widely used by data scientists and engineers, making GE accessible and easy to adopt.

© Copyright Tecnisys 2026. All rights reserved.

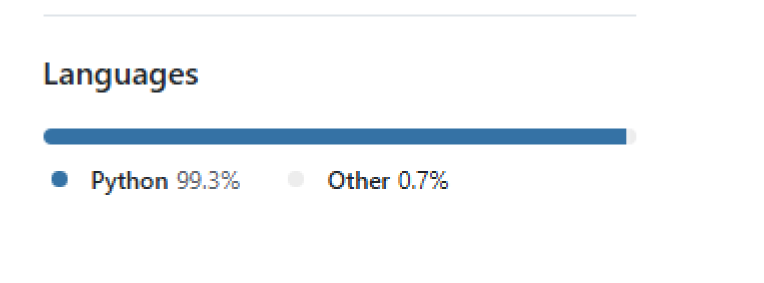


Figure 4 - Great Expectations Language

Sources:

- Great Expectations website

© Copyright Tecnisys 2026. All rights reserved.

Apache HBase

NoSQL database

The NoSQL ("no SQL" or "not only SQL") model represents a non-relational databases (or not only relational, according to more recent definitions). It is a class of database that provides mechanisms for storing and retrieving data modeled in ways other than the tabular relationships used by relational databases.

These databases have been around since the 1960s but gained popularity in the 2000s, triggered by the needs of companies like Facebook, Google, and Amazon.com. They are increasingly used in Big Data and real-time web applications.

NoSQL databases can support query languages similar to SQL.

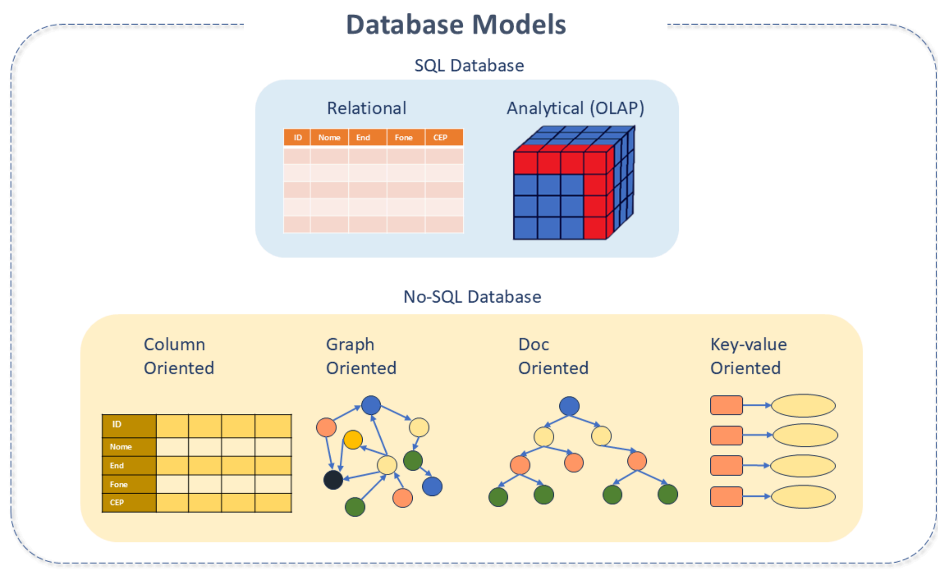


Figure 1 - Database Models

Apache HBase Features

HBase is an open-source, non-relational ("NoSQL"), column-orienteddatabase that runs on top of the HDFS, created to provide efficient random access and real-time read/write

© Copyright Tecnisys 2026. All rights reserved.

on large distributed datasets.

It was created by Powerset to host very large tables and was later absorbed by the Apache Foundation as part of the Hadoop Project, becoming a great option for combining and storing multi-structured and sparse data.

HBase is natively integrated with Hadoop and works seamlessly alongside other data access "engines" through YARN. It has many features that support linear and modular scalability.

Its clusters expand by adding RegionServers that are hosted on "commodity class" servers (common hardware). If a cluster expands from 10 to 20 RegionServers, for example, it doubles its storage and processing capacity.

Its main features are:

- **A Consistent Reads and Writes:** HBase is a consistent DataStore, which makes it very suitable for tasks like high speed counter aggregation.
- **Automatic Sharding:** HBase tables are distributed in the cluster via Regions, and these regions are automatically split and redistributed as data grows.
- **Automatic RegionServer Failover:** Incoming data "resides" in regions made available by RegionServers. If a RegionServer fails, the master server will designate another to take over the regions that were handled by the failed RegionServer.
- **Hadoop/HDFS Integration:** HBase supports HDFS "out of the box" as its distributed file system.
- **MapReduce Support:** HBase supports heavily parallelized processing via MapReduce, acting as both a source and a sink.
- **Java Client API:** HBase supports an "easy to use" Java API for programmatic access.
- **Thrift/REST API:** HBase supports Thrift and REST for non-Java front-ends.
- **Block-cache and Bloom Filters:** HBase supports high volume query optimization.
- **Operational Management:** HBase provides integrated web pages for operational insight and JMX metrics.
- **High Availability:** HBase provides high availability with:
    - Highly available cluster topology information through production deployments with multiple HMaster and Zookeeper instances.
    - Data distribution across multiple nodes ensuring that the loss of a single node only affects the data stored on that node.
    - HBase HA allows data storage, ensuring that the loss of a single node does not result in the loss of data availability.

© Copyright Tecnisys 2026. All rights reserved.

- The HFile format stores data directly in HDFS and can be read or written by Apache Hive, Apache Pig, MapReduce, and Apache Tez, enabling deep analysis in HBase without data movement.

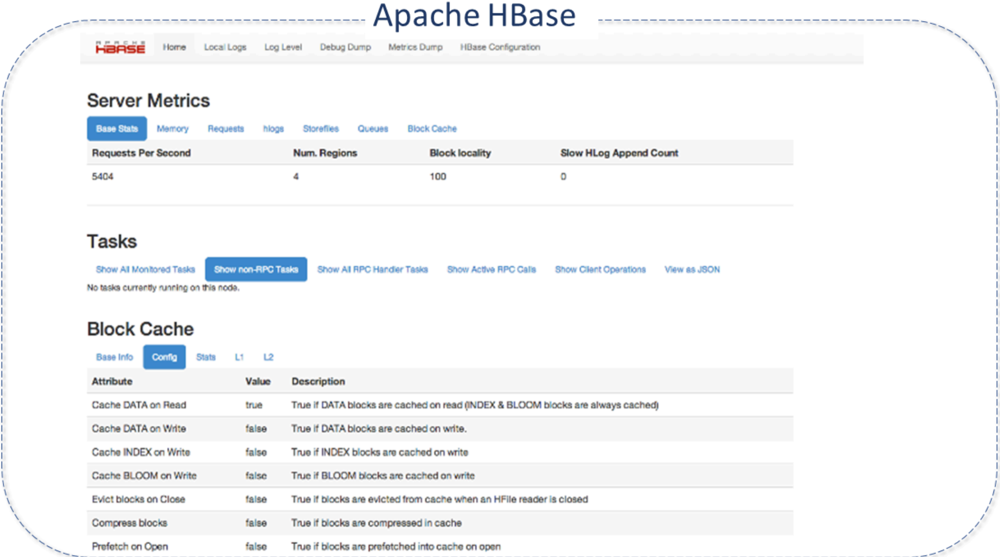


Figure 2 - HBase Interface

Apache HBase Architecture

The HBase architecture also follows the master/slave model and is made up of three main components:

- **HMaster(Master Server):** Is a process responsible for assigning regions to RegionServers in the Hadoop cluster for load balancing. It is responsible for monitoring all Region Server instances in the HBase cluster and acts as an interface for all metadata changes.
  The master typically runs on a dedicated server within the Hadoop cluster. There are other masters in standby available for replacement in case of unavailability, and Zookeeper takes care of this.

> **NOTE**
>
> It is possible for the master to be idle for some time, and it is possible for the Hadoop cluster to work without it during this period, but it is always good to reactivate the master as soon as possible because it drives some critical functionalities such as:

© Copyright Tecnisys 2026. All rights reserved.

> - Management and monitoring of the Hadoop Cluster;
> - Failover control
> - Assignment of regions to RegionServers, with the support of Zookeeper.
> - Load balancing of regions between RegionServers.
> - Schema changes and other metadata operations.
> - DDL operations, such as creating and deleting tables.

- **Regionservers:** maintain the multiple regions designated by the HMaster. Several regions are combined into a single RegionServer.
  They are responsible for client communication and for performing all data-related operations. Any read or write request to the regions is handled by the RegionServer.
  The RegionServer is implemented by the HRegionServer and runs on every data node in the HBase cluster. It is composed of the following components:

  - *Write-Ahead Log (WAL):* The WAL records all HBase data changes. In a normal situation, it is not needed, as data changes are moved from the MemStore to the StoreFiles. However, if a RegionServer crashes or becomes unavailable before the MemStore is flushed, the WAL ensures that changes can be replayed. If the write to the WAL fails, the entire operation fails.The WAL is appended to each RegionServer. Usually, there is only a single WAL instance per RegionServer. The exception is the RegionServer that loads the HBase:meta table. The table has its own dedicated WAL.
  - *Block Cache::* The Block Cache resides in the RegionServer and is responsible for storing frequently accessed data in memory, helping to increase performance.
    The Block Cache follows the least-recently -used (LRU) concept, where least-recently -used records will be removed.
  - *MemStore:* It is a write cache. A temporary memory for all incoming data. Responsible for storing data not yet written to disk. There are multiple MemStores in the HBase cluster.
  - *HFile:* These are the files or cells in which the data is stored. When the MemStore is full, it writes data to the HFile on disk.
- *Regions:* These are the basic element of table availability and distribution.
  They are composed of a Column Family. HBase tables are divided by Rowkey ranges into regions.
  All rows between region start key and region end key will be available in the region.

© Copyright Tecnisys 2026. All rights reserved.

Each region is assigned to a RegionServer, which manages the read/write requests for that region.
As regions are basic blocks of scalability in HBase, a low number of regions per RegionServer is recommended to ensure high performance.
The HBase architecture uses an automatic splitting process to maintain the data. In this process, whenever an HBase table gets too long, it is distributed throughout the system with the help of the HMaster.

- *Zookeeper:* Zookeeper performs distributed coordination, providing regions to the RegionServers and also recovering regions in case of RegionServer failure, loading them onto other RegionServers. If a client wants to share or swap with regions, they need to contact Zookeeper first.
  The HMaster service and all RegionServers are registered with the ZooKeeper service.
  The client accesses ZooKeeper to connect with RegionServers and HMaster. In case of failure of a node in the HBase cluster, the Zookeeper ZKquorum will trigger an error message and start the repair failed nodes.
  The ZooKeeper service maintains and tracks all RegionServers in the HBase cluster and collects information such as the number of RegionServers, assigned datanodes, etc. It is also responsible for establishing client communication with RegionServers, maintaining control of server failures and network partitions, configuration information, etc.

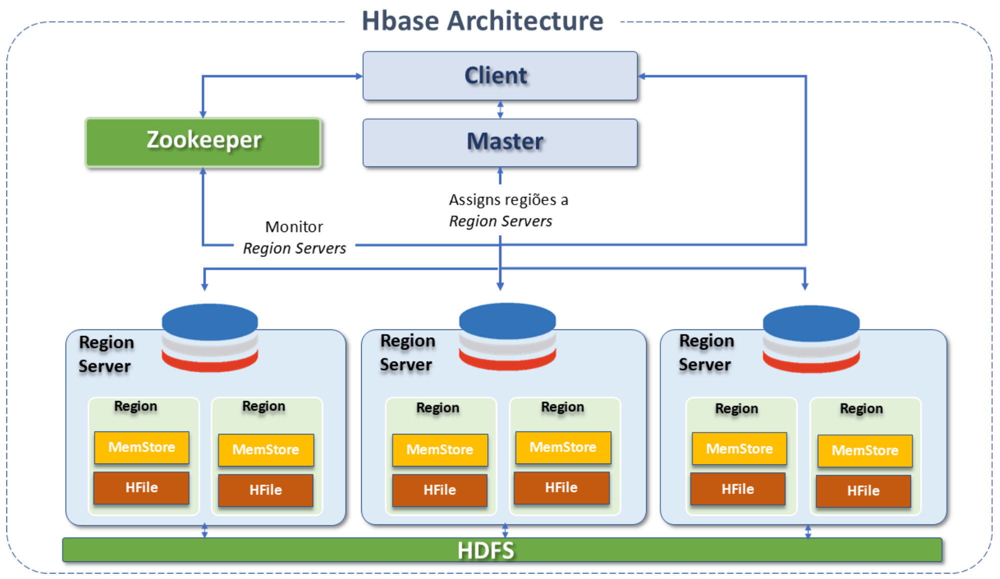


Figure 3 - HBase Architecture

How Apache HBase Works

© Copyright Tecnisys 2026. All rights reserved.

HBase scales linearly, requiring all tables to have a primary key. The key space is divided into sequential blocks which are then assigned to a region. RegionServers have one or more regions, so the load is distributed evenly across the cluster. If the keys within a region are frequently accessed, HBase can subdivide the region so that manual data slicing is not necessary.

The Zookeeper and HMaster servers provide cluster topology information to clients. The clients connect to these and download:

- the list of RegionServers,
- the regions contained in those RegionServers, and
- the key ranges hosted by the regions.

Since clients know exactly where each piece of information is in HBase, they can contact the RegionServer directly without the need for a "central coordinator."

Regionserver includes a memstore to cache frequently accessed rows in memory.

HBase Data Model

In HBase, data is stored in tables, which have rows and columns, and its structure can be understood as a multidimensional map.

Its main components are:

- **Tables:**

  As the HBase architecture is column-oriented, data is stored in tables that are in tabular format. These tables are declared in advance at the time of "schema" definition. A table consists of multiple rows.

- **Rowkey:** A row in HBase corresponds to a Rowkey and one or more columns with values associated with it. The rows are sorted alphabetically by the Rowkey as they are stored. The purpose of the Rowkey is to store data so that related rows are close to each other.

- **Columns:** A column in HBase consists of a column-family (column-family) and a column qualifier delimited by a ":" (colon) character. They are the different attributes of the dataset. Each Rowkey can have an unlimited number of columns.

© Copyright Tecnisys 2026. All rights reserved.

    - The columns are not defined beforehand in the structure, as in the Relational Database.
    - It is not mandatory that all columns are present in all rows.
    - Columns do not store null values.
    - All columns in a Column-Family are stored together in one file (HFile) along with the Rowkey.
- **Column Family:** These are the grouping of multiple columns. A simple read request to a column-family gives access to all columns in that family, making it quick and easy to read the data. A table must have at least one Column-Family, which is created with the table.

- **Column Qualifiers (Column Qualifiers):** These are like column headings or attribute names in a normal table. A column qualifier is added to a column-family to provide the index for a particular piece of data.

- **Cell**
  It is a combination of row, column-family, and a column qualifier and contains a value and a timestamp, which represents the version of the value.

- **Timestamp:** Data is stored in the HBase data model with a Timestamp. The_timestamp_is written next to each value and is the identifier for a particular version of the value. By default, the *timestamp* represents the time on the RegionServer when the data was written, but this can be configured when putting the data into the cell.

Namespace

It consists of a logical grouping of tables analogous to a Database in RDBMS systems. This abstraction lays the groundwork for multi-tenancy resources:

- **Quota Management:** Restricting the amount of resources (regions, tables) that a namespace can consume.

- **Security Administration:** Providing another level of security management for "tenants."

- **Groups of RegionServers:** A table/namespace can be fixed in a subset of RegionServers, ensuring a certain level of isolation.

© Copyright Tecnisys 2026. All rights reserved.

- **Namespace Security Management:** A Namespace/table can be pinned to a subset of RegionServers, ensuring isolation.

Data Model Operations

Like any database, HBase supports a set of basic operations referred to as CRUD (Create, Read, Update, and Delete).

Versions

A tuple (row, column, version) specifies a cell. There can be an unlimited number of cells where the row and columns are the same but the cell address differs in its version dimension.

Joins

HBase does not support joins in the same way as RDBMS. HBase reads are Get and Scan. To enable the join, there are two strategies:

- Denormalize the data when writing to Hbase.
- Have lookup tables and do the join between the HBase tables and an application or MapReduce code.

The best strategy depends on what you want to do.

HBase and Schema Design

Designing a Schema for Bigtable is different from designing a schema for a relational database.

In Bigtable, a schema is a blueprint or model of a table, including the structure of the following table components:

- Rowkeys
- Column families (including garbage collection policies)
- Columns

The main concepts that apply to schema design in Bigtable are:

- Bigtable is a key/value store, not a relational store. It is not compatible with "merging" and transactions are only compatible with a single row.

© Copyright Tecnisys 2026. All rights reserved.

- Each table has only one index: the Rowkey. There are no secondary indexes. Each Rowkey needs to be unique.
- Rows are sorted lexicographically by Rowkey, from the string that has fewer bytes to the one with more.
- Column families are not stored in any specific order.
- Columns are grouped by group and sorted lexicographically within that group.
- The intersection of a row and column can contain multiple cells with date/time timestamp. Each cell contains a unique and timestamped version of the data for that row and column.
- All operations are atomic at the row level. An operation affects an entire row or no rows.
- Ideally, reads and writes should be evenly distributed across the row space of a table.
- Bigtable tables are sparse. A column does not take up space in a row that does not use the column.

The Hbase community indicates, to address this issue, the website Google Cloud Bigtable Schema Design for more details.

Apache HBase Resources

- **Security in Apache HBase**

    - **Web User Interface Security:** HBase provides mechanisms to secure various components and aspects in its relationship with the Hadoop infrastructure, as well as clients and resources external to Hadoop.

    - **Security of client access to Apache HBase:** Recent versions of Apache HBase support optional SASL authentication of clients. The community recommends reading the article Understanding User Authentication and Authorization in Apache HBase, for more details.

    - **Transport Level Security (TLS) in HBase RPC communication:** As of version 2.6.0, HBase supports TLS encryption in client-server and Master-RegionServer communication. TLS is a standard cryptographic protocol designed to provide communication security over computer networks.

    - **Securing access to HDFS and Zookeeper:** HBase requires secure Zookeeper and HDFS to prevent access or modification of metadata and data. HBase uses

© Copyright Tecnisys 2026. All rights reserved.

HDFS to keep its data files, write-ahead logs (WALs) and other data. And it uses Zookeeper to store some metadata for operations (master address, table locks, recoveries, etc.)

  - **Securing access to your client data:** The security of client data should also be considered. Some strategies are offered for this:

    - **Role-based Access Control (RBAC):** Where users and groups can read and write to a specific resource or execute a coprocessor endpoint, using the role paradigm.

    - **Visibility labels** that enable cell labeling and access control to labeled cells, aiming to restrict who can read and write to a certain subset of data.

    - **Transparent encryption of data at rest** in the underlying file system, both HFiles and Wal, protecting data at rest from an invasion.

- **In-memory compaction:** In-memory compaction (Aka Acordion) is a new feature of HBase - 2.0.0. It was first introduced in the blog post Accordion: HBase breathes with in-memory compaction.

- **A RegionServer Offheap Read-Write Path:** To help reduce P99/P999 RPC latencies, Hbase 2.x made the read and write path use an offheap buffer pool. Cells are allocated in offheap memory, out of reach of the JVM garbage collector (GC), reducing GC pressure.

- **Backup and Restore:** HBase Backup and Restore provides the ability to create full and incremental backups of tables in the HBase cluster.

- **Synchronous Replication:** Replication in HBase is asynchronous. Therefore, if the Cluster Master fails, the slave Cluster may not have the most current data. To ensure consistency, the user will not be able to switch to the slave Cluster.

- **APIs:** In addition to the native APIs, HBase supports external APIs.

- **Coprocessors:** (Co-processadores) The Coprocessors framework provides mechanisms to execute custom code directly on the RegionServers that manage the data.

When to Use Apache HBase

Apache HBase is not suitable for all situations.

© Copyright Tecnisys 2026. All rights reserved.

- It is a good candidate only for situations with a large volume of data. For small and medium volumes of data, traditional RDBMS may be better, as their data can be concentrated on one or two nodes.

- It does not have as many extra resources as RDBMS (typed columns, secondary indexes, transactions, advanced query languages, etc.).

- It requires adequate hardware structure. Even HDFS does not work well with less than 5 DataNodes and 1 NameNode.

Difference between Apache HBase and HDFS

HDFS is a distributed file system suitable for storing large files. However, it is not a general-purpose file system and does not provide fast lookups of individual records in files.

HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables.

HBase internally arranges its data in indexed HDFS StoreFiles for high-speed lookups.

Best Practices for Apache HBase

- **Hotspotting:** As HBase rows are lexicographically sorted by row key (a process that optimizes scans, allowing related rows to be stored close to each other), when poorly designed, row keys can be a source of hotspotting (when a large amount of client traffic is directed to one node or just a few nodes, overloading a single (or few) machines). It is recommended to design data access patterns so that the Cluster is used fully and evenly.

  To avoid hotspots on writes, it is important to design row keys so that they are in the same region when they really need to be, but overall, the data is written to multiple regions in the Cluster.

  The community suggests some techniques to avoid hotspotting, which can be seen here.

- **Selecting Number of Regions:**
  When creating tables in HBase, you can explicitly define the number of regions for the table or HBase will calculate it based on an algorithm. It is good practice to

© Copyright Tecnisys 2026. All rights reserved.

define the number of regions for the table and this also helps in improving performance.

- **Choosing the number of column families:**
  Almost always, we have one Rowkey and one column-family, but sometimes we can have more than one column-family. It is good practice to keep the number of column-families as low as possible and never exceed 10.

- **Balanced Cluster:** An HBase cluster is considered "balanced" when most of the RegionServers have an equal number of regions. You may want to enable the balancer, which will balance the Clusters every 5 minutes.

- **Avoid running other jobs on the HBase Cluster:** The integration frameworks used to configure HBase come with other tools like Hive, Spark, etc. However, HBase consumes CPU and memory intensively and sporadically performs large sequential I/O access. Therefore, running other jobs will result in a significant decrease in performance.

- **Avoid "major" compaction and split:** HBase has two forms of compaction: the "minor" compaction, which combines a configurable number of small Hfiles into a larger one, and the "major" compaction, when all store files are read for a region and written to a single file. This takes time and prevents writes to the region until it finishes. It is good practice to avoid "major" compaction.

- **Rules of thumb for table schemas and RegionServer sizing:** There are different datasets with different access patterns and service level expectations. However, the community presents some general rules that can be seen here.

In addition to the above, the HBase Community provides a series of recommendations for tuning HBase performance, which can be read here.

Apache HBase Project Details

HBase was modeled after Google BigTable and written in Java.

© Copyright Tecnisys 2026. All rights reserved.

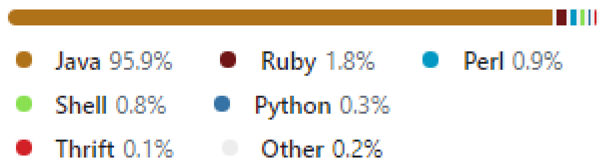


Figure 4 - HBase Languages

Sources:

- https://hbase.apache.org/
- https://hbase.apache.org/book.html
- https://github.com/apache/hbase
- https://cloud.google.com/bigtable/docs/schema-design
- https://blogs.apache.org/hbase/entry/accordion-hbase-breathes-with-in

© Copyright Tecnisys 2026. All rights reserved.

Apache Hive

Analytics

A database is a set of data belonging to the same context, systematically stored within a structure built to support them. In this structure, the necessary business rules coexist to achieve specific goals.

An analytical database is a type of database created to store, manage, and consume data, designed for specific solutions in Business Analysis, Big Data, and BI.

Unlike a typical database, which stores data by transactions or process, an analytical database stores historical data in business metrics. It is based on multidimensional models that use relationships with data to generate multidimensional matrices called "cubes". These models can be directly queried by combining their dimensions, avoiding complex queries that would be performed in conventional databases.

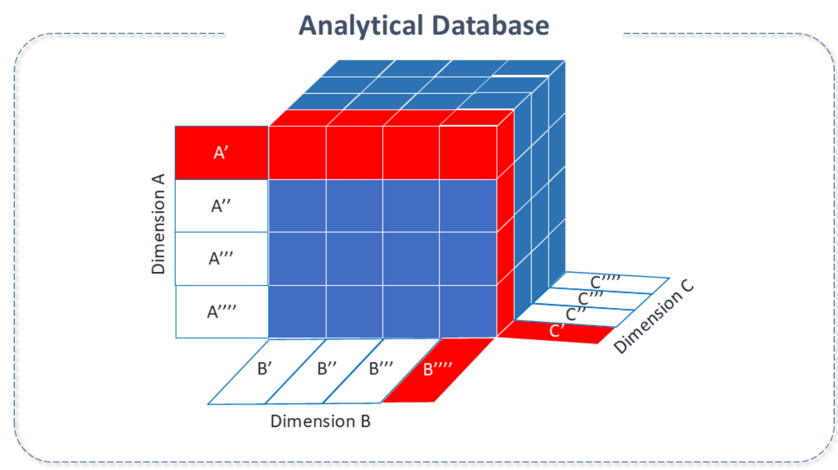


Figure 1 - Analytical Database

© Copyright Tecnisys 2026. All rights reserved.

Apache Hive Features

Apache Hive is an analytical database (Datawarehouse) based on APACHE Hadoop, designed to facilitate data-warehouse users who are proficient in SQL queries but find it challenging to adopt Java or other languages.

It provides a SQL-like interface between the user and the distributed file system of Hadoop-HDFS.

It was designed to facilitate summarization, analysis, querying, reading, writing, and handling large datasets.

With it, it has become possible to define tables with data stored in HDFS and then run queries for transformation or report generation.

- Apache Hive offers standard SQL functionalities, including many features from SQL 2003, SQL 2011, SQL 2016, and later, for analysis.
- Apache Hive can also be extended through "user-defined functions (UDFs), user-defined aggregate functions (HDAFs) and user-defined table functions (UDTFs)".
- There isn't a single "Hive format." Hive includes built-in connectors for:
  - Text files with comma and tab separations (CSV/TSV)
  - Apache Parquet (a columnar storage format available for any project in the Hadoop ecosystem)
  - Apache ORC (a self-describing type-aware columnar file format designed for Hadoop workloads)
  - other formats.
- Users can extend Hive with connectors for other formats from the following profiles:

© Copyright Tecnisys 2026. All rights reserved.

File Formats

	TEXTFILE	SEQUENCE	RCFILE
Data Type	Text Only	Text/Binary	Text/Binary
Internal Storage Order	Based on lines	Based on lines	Based on columns
Compression	File-based	Block-based	Block-based
Splittable	YES	YES	YES
Splittable after Compression	NO	YES	YES

Figure 2 - HIVE File Formats

- The query and storage operations are similar to traditional DBMS. However, there are many differences in the structure and functioning of Hive.
- Apache Hive is not designed for online transaction processing (OLTP). It is best used for traditional Data Warehousing tasks.
- It was designed to maximize scalability, performance, extensibility, fault tolerance, and flexible coupling, with its input formats.
- The simplicity of the Hive query language (HQL) has helped Hive gain popularity in the Hadoop community, being used in many projects worldwide. HQL provides `explain` and `analyze` commands that can be used to check and identify the performance of queries. Additionally, Hive logs contain detailed information for performance investigation and troubleshooting.

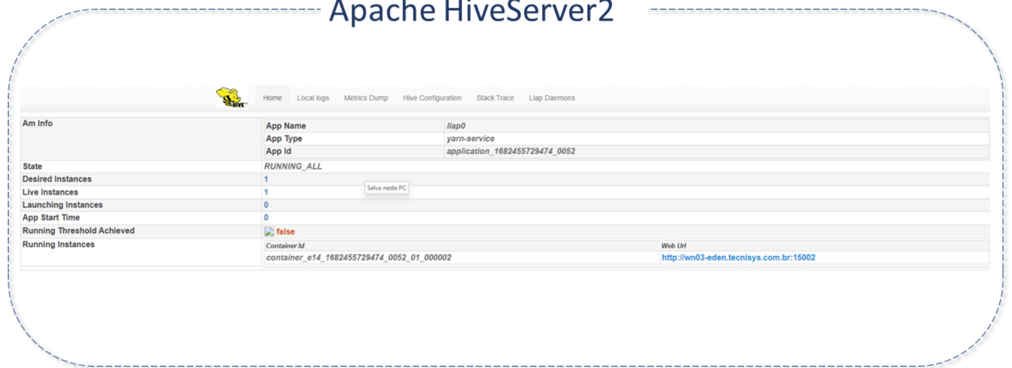


Figure 3 - HiveServer2 Interface

© Copyright Tecnisys 2026. All rights reserved.

Architecture of Apache Hive

Hive consists of multiple components, with the main ones being:

- **Hive Client**: Hive provides different drivers for communication with various types of applications.
  - For Thrift-based applications, it provides the Thrift client.
  - For Java applications, it provides JDBC Drivers.
  - For other types of applications, it provides ODBC drivers.
- **Metastore server**: Stores metadata for each table, such as schema and location, including partition metadata, which helps the driver track the progress of various data sets distributed across the Cluster.
  - Maintains details about tables, partitions, schemas, columns, etc. The data are stored in a traditional RDBMS format.
  - The metadata helps the "driver" to track the data.
  - A backup server regularly replicates the data that can be recovered in case of data loss.
  - Provides a Thrift Service interface for access to information and metadata.
- **Driver**: Responsible for receiving the Hive queries submitted by clients.
  - Initiates the execution of the statement by creating sessions and monitors the lifecycle and progress of the execution.
  - Stores the necessary metadata generated during the execution of a HIVEql statement.

  The driver also serves as a collection point for data or query results obtained after the reduce operation. It has four components:

  - **Parser**: Responsible for checking syntax errors in queries. It is the first step in query execution and returns an error to the client via the Driver if irregularities are found.
  - **Planner**: Successfully parsed queries are forwarded to the planner that generates the execution plans using tables and other metadata information from the metastore.
  - **Optimizer**: Responsible for analyzing the plan and generating a new optimized DAG plan. Optimization can be done on joins, reducing shuffling data, etc., aiming for performance optimization.

© Copyright Tecnisys 2026. All rights reserved.

- **Executor**: Once the parser, planner, and optimizer have completed their tasks, the executor will start executing the job in the order of dependencies. The optimized plan is communicated to each task using a file. The executor takes care of the lifecycle of tasks and monitors their execution.

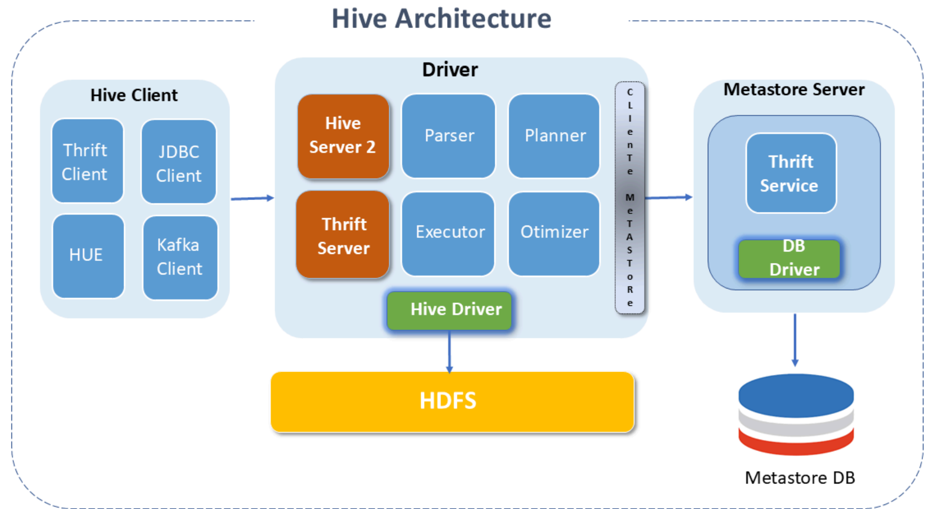


Figure 4 - Architecture of Apache Hive

Interaction of Hive with Hadoop

In general, the interaction between Hive and Apache Hadoop occurs as follows:

"1 " - The UI calls the execution interface of the Driver.
"2 " - The **Driver** creates a session identifier for the query and sends it to the compiler to generate an execution plan.
"3 e 4 " - The **compiler** obtains the necessary metadata from the **metastore**, which are used to check the type of expressions in the query tree and to remove partitions based on the query predicates (when a boolean result is expected).

- The Plan generated by the **compiler** is a DAG of stages, with each stage being a map or reduce job, a metadata operation, or an HDFSoperation.
- **Map/reduce stages**: The plan includes map operator trees (operator trees executed on mappers) and a reduce operator tree (for operations using reducers).

"6 " - The **execution engine** submits these stages to the appropriate components (steps 6, 6.1, 6.2, 6.3).

© Copyright Tecnisys 2026. All rights reserved.

- In each **map** or **reduce**, the deserializer associated with the table or intermediate outputs is used (to read the lines from HDFS files passed through the associated operator tree).

The **output** is written to a temporary HDFS file through the serializer (which happens in the mapper if the operation does not require reduce).

Temporary files are used to provide data for subsequent map/reduce stages of the plan.

For DML operations, the final temporary file is moved to the table's location.

The following figure illustrates how a common query "flows" through the system:

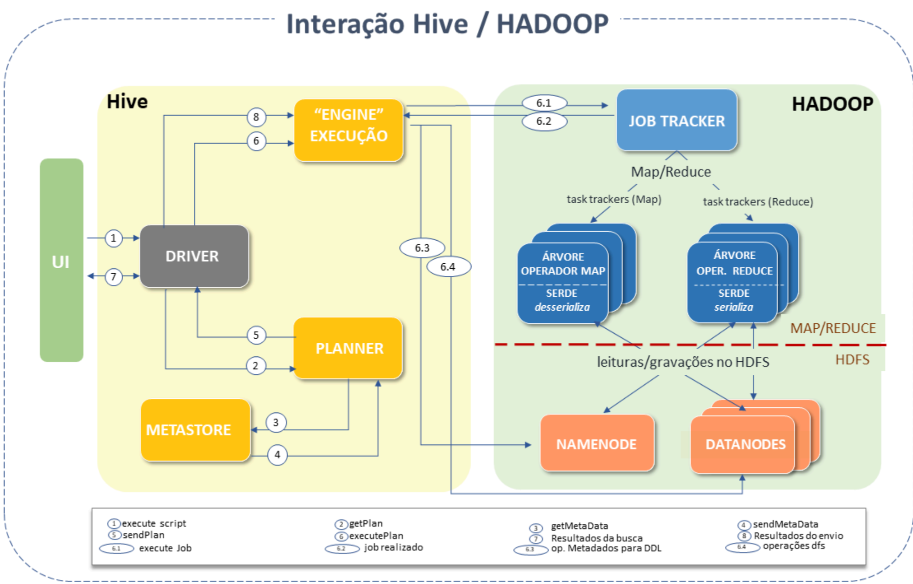


Figure 5 - Interaction of Hive with Hadoop

How Apache Hive Works

- HCatalog: A table and storage management layer for Hadoop that enables tools like Hive and MapReduce to read data from tables.

  It is built "on top" of the Hive Metastore service and thus supports file formats for which Hive SerDe (serialization and deserialization) can be done.

  HCatalog enables viewing data as relational tables, without the need to worry about the location or format of the data.

© Copyright Tecnisys 2026. All rights reserved.

Supports text file, ORC file, sequence file, and RCFile formats, and SerDe can be written for formats like AVRO (an open-source data format that enables grouping serialized data with the schema in the same file).

HCatalog tables are "immutable," which means that data in the table and partition cannot be appended. In the case of partitioned tables, data can only be appended to a new partition, not affecting the old partition.

Interfaces for MapReduce applications are HCatInputFormat and HCatOutputFormat, both are Hadoop-compatible Input and Output Formats for Hive. HCatalog is evolving new interfaces to interact with other components of the Hadoop ecosystem.

- **WebHCat** provides a service for executing Hadoop MapReduce (or YARN), Pig, and Hive tasks. It also allows the execution of Hive metadata operations using an HTTP (REST-style) interface.

Data Organization

By default, Hive stores metadata in an embedded Apache Derby database. Other client-server databases can optionally be used, such as Postgree.

The first four file formats supported by Hive were just plain text, sequential files, ORC, and RCFile formats. Apache Parquet can be read via a plug-in in versions later than 0.1, originally starting in 0.13. Parquet's compressed data representation reduces the amount of data Hive has to traverse, consequently speeding up the execution of queries.

Additional plugins support querying the Bitcoin Blockchain.

Data Units

Hive is organized into:

- **Databases**: Namespaces exist to avoid name conflicts in tables, views, partitions, columns, and all else. Databases can also be used to enforce security for a user or group of users.

- **Tables**: Homogeneous units of data that have the same schema.

© Copyright Tecnisys 2026. All rights reserved.

- **Partitions**: Each table can have one or more partition keys that determine how the data is stored. Partitions, in addition to being storage units, also enable the user to efficiently identify rows that satisfy a specific criterion. Each unique value of a partition key defines a partition of the table.

- **Buckets or Clusters**: Data in each partition can be divided into "buckets" based on the hash function value of some table column. For example, a table can be "broken" by userid, which is one of the table's columns.

Apache Hive Resources

Built on Apache Hadoop, Hive provides the following resources:

- **SQL Access**, facilitating data warehousing tasks such as extraction/transformation/loading (ETL), report generation, and data analysis.

- **Support for concurrency and authentication of multiple clients with Hive-Server 2 (HS2)**, designed to provide better support for open API clients such as JDBC and ODBC.

- **Full support for ACID** properties for ORC tables and insertion to all other formats**.

- **Support for initialization and incremental replication** for data backup and recovery.

- **Central repository of metadata** for Hive tables and partitions in a relational DBMS with access to this information via the MetaStore service API.

- **Mechanism to establish structure in a variety of data formats**.

- **Cost-based query optimizer (CBO)**.

- **Access to files stored directly in Apache HDFS or others, such as Apache Hbase**.

- **Query execution via Apache Tez, Apache Spark, or Mapreduce**.

- **Procedural language with HPL-SQL**.

- **Sub-second query retrieval via Hive LLAP and Apache YARN**.

© Copyright Tecnisys 2026. All rights reserved.

- Beeline-Command Line Shell HiveServer2 supports Beeline Shell, a JDBC-based SQL CLI client that operates both in embedded mode (with an integrated Hive - similar to the Hive CLI) and remotely (connecting to a separate HS2 process via Thrift).

Main Differences Between Hive and Traditional DBMS

In traditional DBMS, a schema is applied to a table when data is loaded into it. This allows the DBMS to check if the inserted data follows the table's representation as specified in its definition. This design is called "schema on write".

Hive does not check data against the table schema on write. It performs checks at read time. This model is called "schema on read". Both approaches have their advantages and disadvantages. Checking during load adds overhead, burdening data load time, but ensures data is not corrupted. Early detection ensures early handling of exceptions. As tables are forced to match the schema during/after data load, they perform better at query time.

Hive, on the other hand, can load data dynamically without schema verification, ensuring a quick initial load but slower performance at query time.

Transactions are fundamental operations in traditional DBMS.

Like any typical RDBMS, Hive supports the four properties of transactions (ACID): Atomicity, Consistency, Isolation, Durability. Transactions were included in Hive 0.13, but only limited to partition level.

Version 0.14 of Hive was fully added to support complete ACID properties. From this version, different line-level transactions such as insert, delete, update are possible. Enabling these commands requires setting appropriate values for configuration properties such as hive.support.concurrency, hive.enforce.bucketing, hive.exec.dynamic.partition.mode

Apache Hive Security

Version 0.7.0 of Hive added integration with Hadoop security, which in turn began using Kerberos authorization support to provide security. Kerberos allows mutual authentication between client and server. In this system, the client's request for a ticket is passed along with the request.

© Copyright Tecnisys 2026. All rights reserved.

Permissions for newly created files in Hive are dictated by HDFS. The distributed file system's authorization model of Hadoop uses three entities: user, group, and others with three permissions: read, write, execute. Default permissions for newly created files can be configured by changing the unmask value for the Hive configuration variable hive.files.umask.value.

Best Practices for Apache Hive

- Tables Partitioning: Apache Hive addresses the inefficiency of executing MapReduce jobs with large tables by offering an automatic partitioning scheme at the time of table creation.

    - In this method, all table data are divided into multiple partitions, each corresponding to specific value(s) of the partition column(s), which are maintained in HDFS records (as sub-directories).
    - A query with partition filtering will only load data from specific partitions (subdirectories), bringing more speed to the process. The choice of partition key is an important factor, always should be a low "cardinal" attribute to avoid overloads.

    NOTE

    Commonly used attributes as partition keys are:

    - partitions by date and time: date-time, year-month-day (even hours)
    - partitions by location: country, territory, city, state.
    - partitions by business logic: department, customers, sales region, applications.

- **Denormalization**: Normalization is a standard process used to model data tables dealing with redundancies and other anomalies. Joins are expensive and complicated operations and generate performance issues. It's a good idea to avoid highly normalized table structures, as they require joining queries.

- **Map/Reduce output compression**: Compression significantly reduces the volume of intermediate data and minimizes the amount of data transfer between Mapper and Reducer.

© Copyright Tecnisys 2026. All rights reserved.

- Compression can be applied to the output of Mapper and Reducer individually. Important to remember that gzip compressed files cannot be split. It means that it should be applied cautiously.
- The file size should not be larger than a few hundred Mb, at the risk of producing unbalanced work.
- Compression codec options can be snappy, izo, bzip, etc.

- Map Joins: Map Joins are efficient if the table "on the other side of the join" is small enough to reside in memory.

    - With the parameter hive.auto.convert.join=true, Hive tries the map join automatically. When using this parameter, it is appropriate to make sure that auto-convert is enabled in the Hive environment.
    - It is essential, still, the parameter hive.enforce.bucketing = true. This setting will cause Hive to reduce scan cycles to find a specific key because bucketing will ensure that the key is in a specific bucket.With the parameter hive.auto.convert.join=true, Hive automatically attempts to use map join. When using this parameter, it is advisable to ensure that auto-convert is enabled in the Hive environment. Additionally, the parameter hive.enforce.bucketing=true is essential. This configuration will reduce the scan cycles required to find a specific key because bucketing ensures that the key is in a specific bucket.

- **Selection of input format**: Input formats play a critical role in Hive's performance. The text type, for example, is not suitable for a system with high data volume because these readable types of formats take up a lot of space and bring overhead to analysis.

    - To address this, Hive introduces columnar input formats like RCFile, ORC, etc. These formats allow reducing read operations in analytical queries, enabling access to each column individually.
    - Other binary formats like Avro, sequence files, Thrift, and ProtoBuf can also help.

- **Parallel execution**: Hadoop can execute MapReduce jobs in parallel, and many queries run in Hive utilize this parallelism automatically.

    - However, single and complex Hive queries are often converted into multiple MapReduce jobs, which, by default, are executed sequentially.

    - Some of the MapReduce stages of a query are not interdependent and can be run in parallel, leveraging "the available" capacity of the Cluster, improving its

© Copyright Tecnisys 2026. All rights reserved.

utilization, and reducing the overall execution time.
  - The Hive configuration to change this behavior is to toggle the flag

    ´´´shell Set hive.exec.parallel=true ´´´
- **Vectorization**: Vectorized query execution improves the performance of Hive.
  - It is a technique that works with processing data in batches, rather than one line at a time, resulting in efficient CPU usage.
  - To enable vectorization, the appropriate parameter is Set hive.vectorized.execution.enabled=true.
- **Unit Testing**: Unit testing determines if the smallest testable part of the code functions. Detecting problems early is very important.
  - Hive allows testing units like UDFs, SerDes, Streaming Scripts, Hive Queries, etc., through quick tests in local mode.
  - Several tools are available that assist in testing Hive queries, such as HiveRunner, Hive_test, and Beetest.
- **Sampling**: Sampling allows a subset of data to be analyzed without the need to analyze the entire set.
  - With a representative sample, a query can return significant results much more quickly, consuming fewer resources.
  - Hive offers the TABLESAMPLE, which allows tables to be sampled, at various levels of granularity - returning subsets or HDFS blocks or just the first n records of each input split.
  - A UDF can be implemented to filter the records according to a sampling algorithm.

Details of Project Apache Hive

Apache Hive was predominantly developed in Java. HiveQL is Hive's SQL-based language. It "applies" SQL syntax in creating tables, loading data, and querying. It also allows the incorporation of custom MapReduce scripts. These scripts can be written in any language using a simple streaming interface (reading lines on standard input and writing on standard output).

© Copyright Tecnisys 2026. All rights reserved.

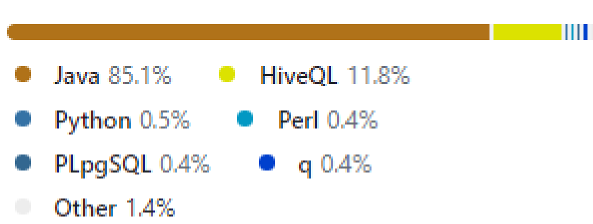


Figure 6 - Languages of Hive

TDP Kubernetes

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **4.0.0**, distributed via Helm Chart `tdp-hive-metastore` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources

- Apache Hive - cwiki
- Apache Hive.org
- GitHub - Apache Hive

© Copyright Tecnisys 2026. All rights reserved.

Apache Iceberg

Table Formats

Table Formats are table structures used to organize and manage complex data, allowing it to be viewed and interacted with as a single, understandable "table" accessible to multiple users and tools simultaneously.

Table Formats add a table-like abstraction layer over native file formats, serving as a metadata layer and providing the primitives needed for computing engines to interact with stored data more efficiently, ensuring enhanced ACID compliance, the ability to log transactional data efficiently, scalability, and the ability to update or delete records.

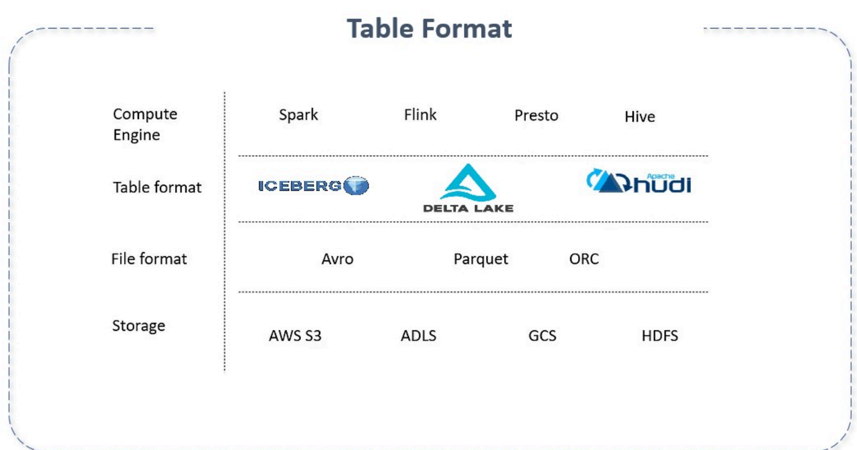


Table Formats add an abstraction layer to data files

Table formats evolved in response to a critical need: to combine the data management advantages of warehouses (OLAP databases) with the greater scalability and cost-effectiveness of data lakes. In essence, they combine the versatility of data lakes for

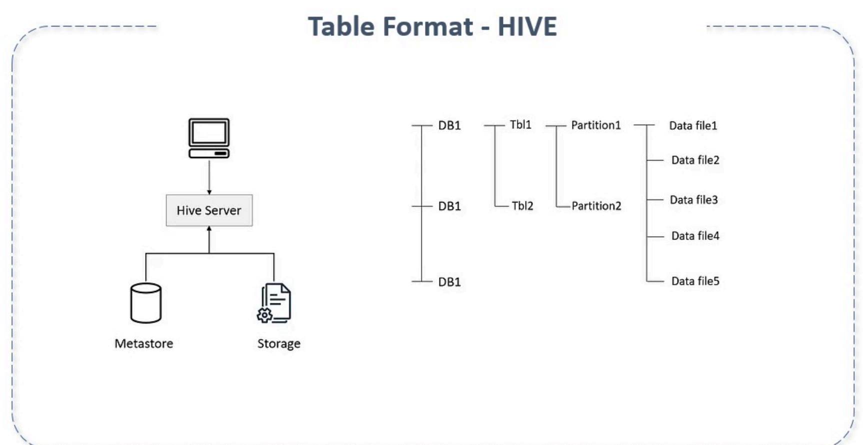
© Copyright Tecnisys 2026. All rights reserved.

handling raw and semi-structured data with the capability to process transactional workloads.

In simple terms, table formats are a way to organize data files. They attempt to bring database features to the data lake. Their primary goal is to provide the table abstraction to people and tools, allowing them to efficiently interact with their underlying data.

Although it wasn't the name used at the time, table formats have existed since System R, Multics and Oracle first implemented Edgar Codd's relational model. When we talk about a group of files being a collection, we're talking about table formats. When a file is viewed as a table, the directory is viewed as a table, which facilitates tracking it and its files. This is done with the help of catalog systems. Under it, files are written in versions, and the latest ones corresponding to a table are tracked with the help of the metastore. The metastore is accessed to know the current state of the table and which files are related.

Apache Hive is one of the oldest table formats used. However, since Hive was written in the pre-cloud era, it didn't anticipate object storage, impacting its performance as Hive's metadata tables grow rapidly. To address these issues, new table formats were created.



Hive Table Format

Apache Iceberg

Apache Iceberg is an open-source table format for large analytic datasets, with a specification that ensures compatibility between languages and implementations. It

© Copyright Tecnisys 2026. All rights reserved.

adds tables to computing engines like Spark, Trino, Flink, Hive, etc., using a high-performance table format that works just like a SQL table.

Apache Iceberg allows cohesive real-time access to historical data, ensuring data integrity and consistency. Its main innovation lies in its ability to support read, write, update, and delete operations on data without rewriting entire datasets.

Apache Iceberg quickly gained adoption among large companies like Apple, Netflix, Amazon, etc. Its completely open nature, engaging multiple companies and the community, makes it ideal for data lake architecture.

Apache Iceberg Features

Apache Iceberg was developed in 2017 by Netflix, donated to the Apache Software Foundation in November 2018. It became a top-level project in 2020. It is used by numerous companies including Airbnb, Apple, LinkedIn, Adobe, etc. It was specifically created to address problems and challenges related to file-formatted tables in data lakes, such as data and schema evolution and concurrent writes, consistently, in parallel. It introduces new features that allow multiple applications (like Dremio) to work together on the same data transactionally, consistently, and define additional information about the state of datasets as they evolve and change over time.

Apache Iceberg adds tables to computing engines like Spark, Trino, Flink and Hive using a high-performance table format that works like a SQL table.

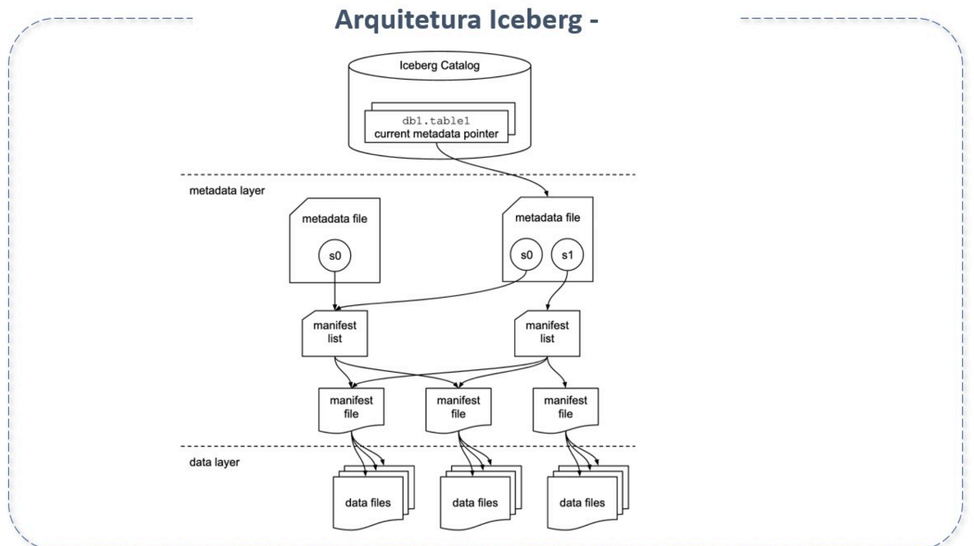
Its main features include:

- **Schema Evolution:** Supports adding, deleting, updating, or renaming without side effects;
- **Hidden Partitionin:** Prevents user errors from causing incorrect results silently or extremely slow queries;
- **Partition Layout Evolution**: Allows updating the layout of a table as data volume or query patterns change.

Architecture of Apache Iceberg

In practice, Apache Iceberg is a table format specification and a set of APIs and libraries that enable engines to interact with tables following this specification.

© Copyright Tecnisys 2026. All rights reserved.



Architecture of an Iceberg table

Its main components are:

- **Iceberg Catalog:** The catalog is a central repository where the reference to each table's metadata file is stored. Its main goal is to support atomic operations to update pointers. For this, it keeps the current location of the metadata pointer (within the catalog there is a reference or pointer to the current metadata file of each table. The pointer value is the location of the metadata file).

- **Metadata Layer:** Metadata is tracked in three files: metadata files, manifest lists, and manifest files.

  - Metadata files: .json file containing information about the table's metadata at a given point in time. It stores the state of the tables. It includes details about the table schema, partition information, current snapshot ID (current-snapshot_id), path to the manifest list, etc. When the state of the tables changes, a new metadata file is created and replaces the old one with an atomic swap. It has the following subsections:

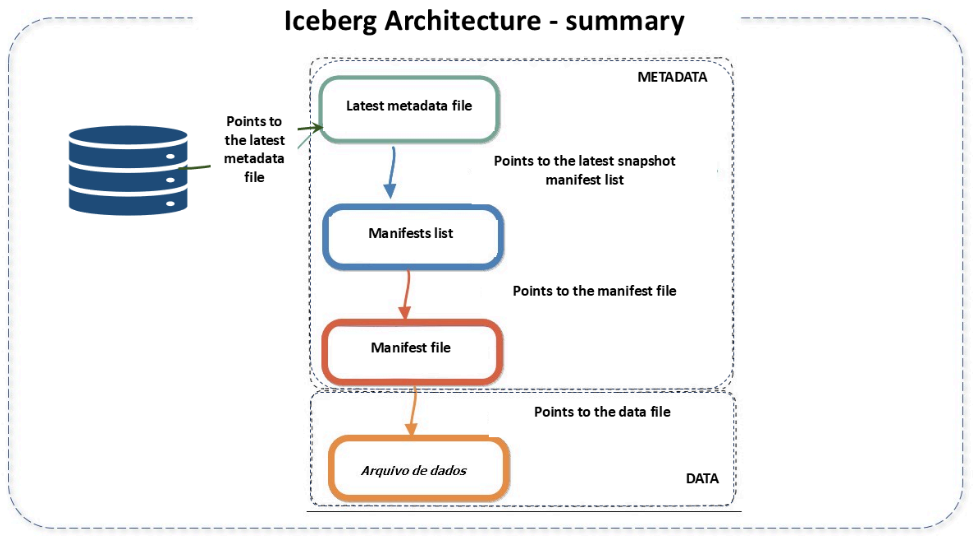
    - Snapshot: : A complete list of all files in the snapshot table. It represents the state of a table at a given moment and is used to access the complete set of data files in the table. It includes information about the table schema, partition specifications, and the location of the manifest list;
    - Schemas: All table schemas altered are tracked by the "schema matrix";
    - Partition Specifications: tracks information about the partition;

© Copyright Tecnisys 2026. All rights reserved.

    - Sort Orders.
  - Manifest List (manifest list): Avro file that contains all manifests and their metrics. They store metadata about the manifests that make up a snapshot, including partition statistics and data file counts. These statistics are used to avoid reading unnecessary manifests for the operation. Acts as a link between the manifest and the snapshot.

  - Manifest Files: Keep track of all data files, along with details and statistics like their formats, location, and metrics.

3. **Data Layer**
  - Data File: The actual physical file. Where the data actually resides.



Iceberg Summary

Apache Iceberg was built for huge tables that can be read without a distributed SQL engine. Each table can contain tens of petabytes of data.

Apache Iceberg Resources

- **User Experience Resources:**

  Apache Iceberg improves the user experience in that:

  - With schema evolution support, it supports additions, deletions, updates, renaming, and reordering of columns in a table without needing to rewrite the table. This makes schema evolution side-effect-free, assigning a unique ID to

© Copyright Tecnisys 2026. All rights reserved.

each newly created column and automatically adding it to the column metadata. It also ensures the uniqueness of each column;
  - With hidden partitioning, it ensures users no longer need to know the structural layout of files before running queries. This avoids user errors causing silently incorrect results or extremely slow queries. Iceberg evolves the table's schema and partition as data scales;

  - With time travel the version control feature, Iceberg ensures that any data changes are saved for future reference, whether adding, deleting, or updating data. Thus, any issue with data versioning can be easily reverted to an older, stable version, ensuring data is not lost and can be compared over time.
- **Reliability**:

  - Snapshot isolation ensures that any read of the dataset sees a consistent "snapshot." In essence, it reads the last confirmed value present at the time of reading. This avoids conflicts. After commits, the record is updated, and a new snapshot is created, reflecting the most recent update;

  - With atomic commits, it ensures that data remains consistent across all queries. To avoid partial changes, any update must be completed on the dataset, or no changes will be saved. This ensures that only correct data is returned, preventing users from seeing incomplete or inconsistent data;

  - Reads are reliable because each transaction (update, addition, deletion) creates a new snapshot. Thus, readers can use the various latest versions of each update to create a reliable query for the table;

  - Operations are at the file level, unlike traditional catalogs that track records by position or name, which requires reading directories and partitions before updating a single record. In Apache Iceberg, it is possible to directly target a single record and any record update without any folder change. This is because the records are stored in their metadata.

- **Reliability**:

  - Each file that belongs to a table has metadata stored for any transaction that occurred, along with extra statistics. This allows users to locate only the files that correspond to the query they want at the time;

© Copyright Tecnisys 2026. All rights reserved.

  - Iceberg uses two levels of metadata to track files in a snapshot - manifest files and the manifest list. The first level contains the data files, along with their partition data and column-level statistics. The second stores the list of manifest snapshots with a range of values for each partition;

  - For fast scan planning, Iceberg uses minimum and maximum values from the partition in the manifest list to filter the manifests. Subsequently, it reads all the returned manifests to get the data file. Thus, it is possible to plan without reading all the manifest files, using the manifest list to restrict the number of manifest files needed for reading.

  With this, it is possible to achieve efficient and economical queries on files.

Best Practices for Using Apache Iceberg

Apache Iceberg is a powerful tool for managing large analytical datasets. To maximize its efficiency and effectiveness, here are some best practices:

- **Data Structure and Schemas**: Keep data schemas simple and evolutionary. Utilize Iceberg's schema evolution functionality to maintain compatibility and minimize disruptions;

- **Efficient Partitioning**: Use logical partitioning to organize data in a way that optimizes the most common queries. Iceberg's hidden partitioning can help avoid performance issues;

- **Metadata Management**: Keep metadata clean and up-to-date. This includes removing old snapshots and unused manifest files to avoid overloading the catalog;

- **Testing and Validation**: Implement rigorous testing when evolving schemas or modifying tables to ensure data integrity;

- **Monitoring and Optimization**: Regularly monitor query performance and optimize tables as needed. This can include adjusting partitioning or modifying indexes;

- **Documentation**: Maintain clear documentation of table structures, schemas, and any associated business logic to facilitate maintenance and collaboration;

- **Security and Access Control**: Implement appropriate access controls and security practices to protect data;

© Copyright Tecnisys 2026. All rights reserved.

- **Resource Usage and Cost**: Be aware of resource usage and associated costs, especially in cloud environments. Optimize storage and compute usage to maintain cost efficiency;

- **Updates and Compatibility**: Stay updated with the latest versions of Apache Iceberg to take advantage of improvements and security fixes.

Using Apache Iceberg according to these practices can significantly improve data management and operational efficiency.

When to Use Apache Iceberg

- **Large Datasets**: When dealing with petabytes of data in huge tables;

- **Modern Data Lakes**: For modern data lake architectures that require robust data and schema management;

- **Scalability and Reliability**: When scalability and reliability are critical for data operations and analysis;

- **ACID Transactions in Data Lakes**: To support ACID transactions in data lakes, ensuring data integrity;

- **Schema Evolution**: When there is a need to evolve data schemas without interruption or data loss;

- **Efficient Updates and Deletions**: If you need to perform efficient updates and deletions in large datasets;

- **Working with Multiple Data Formats**: If working with various data formats and needing a uniform abstraction layer;

- **Concurrent Reads and Writes**: When needing support for concurrent reads and writes without locking;

- **Compliance and Data Governance**: To meet strict compliance and data governance requirements;

- **Integration with Analytics Platforms**: If you want to integrate with analytics platforms like Spark, Trino, Flink, and others;

© Copyright Tecnisys 2026. All rights reserved.

- **Snapshot and Data Traveling**: When snapshot and data traveling functionality is needed for auditing or rollback;
- **Improving Read Performance**: To improve read performance through indexing and file optimizations.

These items highlight ideal scenarios for implementing Apache Iceberg, maximizing its features for efficient data lake management.

When Not to Use Apache Iceberg

Apache Iceberg is not the right tool for:

- **Small Data Volumes**: If the data universe is small and does not require a data lake, Apache Iceberg will not bring benefits.
- **Real-Time Ingestion**: Apache Iceberg does not support real-time data ingestion as it uses batch processing.
- **Centralized Structure**: If the goal is not to use a distributed computing structure, Apache Iceberg is not ideal, as it is designed to use a distributed computing structure to process data.

Development Project Details

Apache Iceberg is primarily developed in Java. Therefore, a good understanding of Java is essential to effectively contribute to the project or customize it. This also implies the need to maintain standard Java coding practices, such as efficient exception handling and using common Java libraries for I/O operations and data manipulation.

TDP Kubernetes

> AVAILABLE IN TDP KUBERNETES
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **1.4.2**, distributed via Helm Chart `tdp-iceberg` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

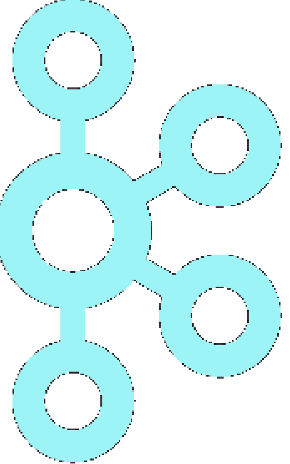
© Copyright Tecnisys 2026. All rights reserved.

Sources

- Iceberg.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Kafka



Data Streaming Platform

Traditional batch processing methods require that data be collected in batches before it can be processed, stored, or analysed.

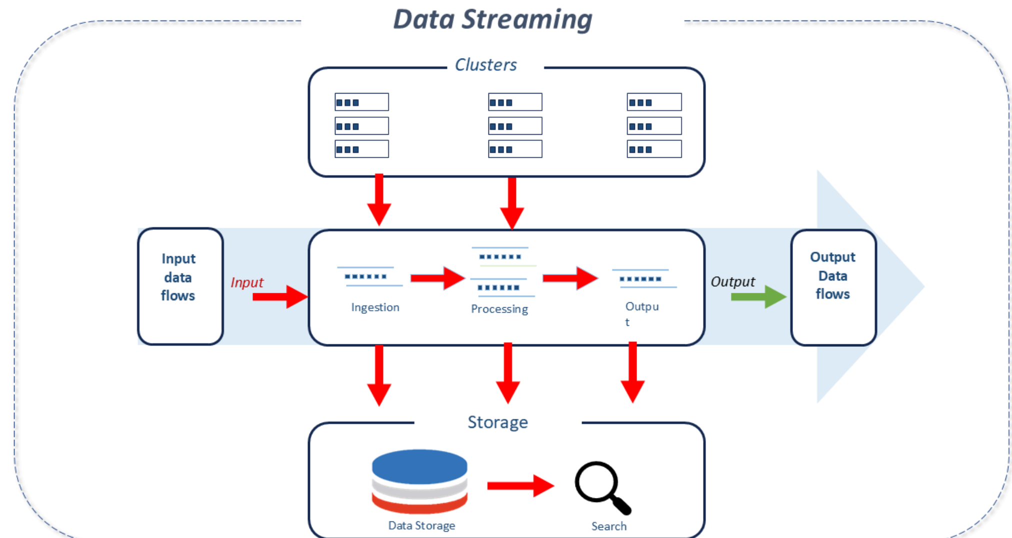
In contrast, data streaming involves the continuous flow of data, which can be processed as soon as it arrives. This allows for dynamic, contextual, and real-time decision making, thereby enabling businesses to harness the full value and potential of their data and applications.

Data streaming captures time-ordered data elements from various event sources such as databases, sensors, mobile devices, cloud services, and software applications.

The captured streams can be processed in real time or retrospectively and routed to different technologies as needed. They can be stored for later retrieval, manipulation, or processing.

Data streaming involves technologies that have emerged in recent years. A real-time data analytics system, for instance, is designed for high data generation, based on events that quickly spread across the network.

© Copyright Tecnisys 2026. All rights reserved.



Data Streaming

Features of Apache Kafka

Kafka is the most commonly used event streaming platform. It is employed for collecting, processing, storing, and integrating data at scale. Kafka supports a variety of use cases including distributed logging, stream processing, data integration, and message publishing/subscribing.

Originally developed as a solution for a LinkedIn software that collected user activity data and used it to display on a web portal, Kafka was built as a fault-tolerant, distributed system that combines three key capabilities:

- Publishing (writing) and subscribing (reading) of event streams, including continuous data import/export from other systems.
- Durable and reliable storage of event streams for as long as necessary.
- Real-time or retrospective event stream processing.

Kafka was built with the following principles in mind:

- Low coupling between producers and consumers.
- Data persistence to support various consumption scenarios and failure handling.
- Maximum end-to-end throughput with low-latency components.
- Handling of diverse data formats and types using binary data formats.

NOTE

© Copyright Tecnisys 2026. All rights reserved.

> Kafka is commonly utilized in stream processing architectures. With its reliable message delivery semantics, it assists in managing high rates of event consumption. Additionally, it provides message replay features for various types of consumers.

Architecture of Apache Kafka

- **Communication System:** Kafka is a distributed system consisting of servers and clients communicating over a high-performance TCP network protocol.
  It can be deployed on bare-metal hardware, virtual machines, containers, and cloud environments.
  Kafka clusters are highly scalable and fault-tolerant. Its communication model uses a Write-Ahead Log (WAL) system where every message is logged before being available to consumer applications so that subscribers and consumers can access it when needed.
  Kafka simplifies system-to-system communication by acting as a centralized communication hub. The implemented publish-subscribe pattern ensures that streams flow in one direction, unlike the bidirectional communication in client-server models.
  Designed as a simple and lightweight client library, it can be easily incorporated into any Java application and integrated with any package.
  It has no external dependencies in the messaging layer. The messaging layer partitions the data for storage and transport.
  This partitioning enables locality, scalability, high performance, and fault tolerance.

- **Key Components of Kafka:**

  - **Clients:** Enable the writing of microservices and distributed applications that read, write, and process event streams in parallel, at scale, and fault-tolerantly. Kafka includes several built-in clients, supplemented by many others provided by the community.
    Clients are available for Java and Scala, including the high-level Kafka Streams library, and for languages like Go, Python, C/C++, and more, as well as REST APIs.

  - **Servers:** Kafka runs as a cluster of one or more servers that can span multiple datacenters or cloud regions. Some of these servers, called brokers, form the storage layer.

© Copyright Tecnisys 2026. All rights reserved.

    - **Brokers:** A typical Kafka cluster consists of multiple brokers. This facilitates load balancing for read and write operations across the cluster.
      Each broker is stateless, with Zookeeper managing its state.
- **Topics:** Organize events similarly to database tables by grouping related data, albeit without enforcing a specific schema.
  They store messages as raw bytes, providing flexibility to handle homogeneous or heterogeneous data.
  Topics are divided into partitions, and physically, each topic is distributed across different brokers, which host one or more partitions for each topic.

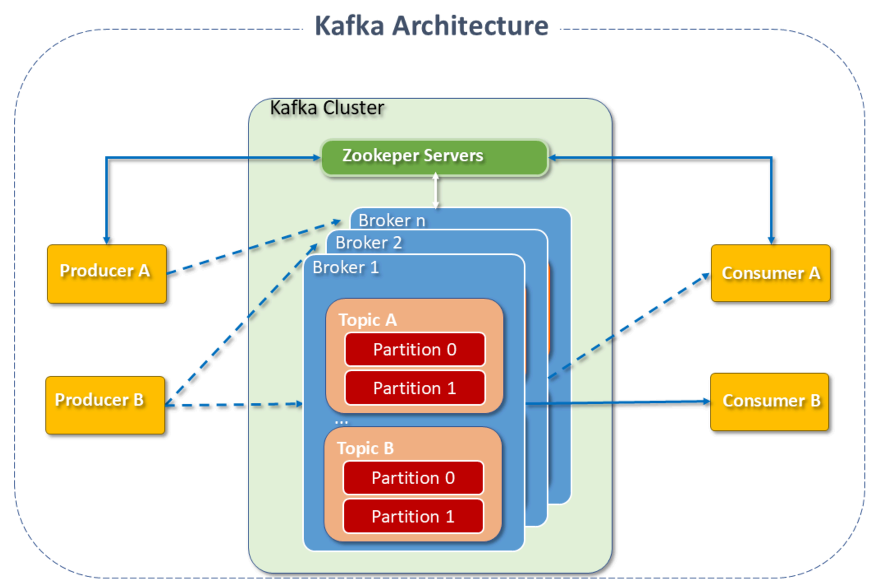
- **Partitions:** Store messages in the order they arrive. Events with the same key (ID) are written to the same partition.
  The number of partitions for a topic is configurable, as is their size.
  Having more partitions generally translates to more parallelism and higher throughput.
  Kafka pipelines should have a uniform number of partitions per broker and all topics on each machine.
  In each partition, one of the brokers acts as the leader, and one or more act as followers.
  Leaders handle read and write requests for their partitions, while followers replicate the leader.
  Followers do not interfere in the leader's operations but act as backups, ready to replace the leader in case of a failure.
  Each Kafka Cluster can simultaneously be a leader for some topic partition or a follower in others. Thus, the load on any server is equally balanced.
  The leader election is conducted with the help of Zookeeper, which manages and coordinates the brokers and consumers.
  Zookeeper keeps track of any additions or failures of brokers in the Cluster, notifying their state to the producers or consumers of the Kafka queues.
  It also assists producers and consumers in coordinating the active brokers, recording which are the leaders for each topic partition and passing this information on to producers and consumers.

- Producers are the applications responsible for sending data to the topic partition for which they are producing data.
  The producer does not write data to the partition. It only creates write requests for messages and sends them to the lead broker.

© Copyright Tecnisys 2026. All rights reserved.

Depending on the configuration, the producer waits for a confirmation of messages.

- **Consumers** are applications or processes that subscribe to (read and process) events.
  They fetch messages from log files belonging to a topic partition. They are the ones who distribute the work across multiple processes.



Apache Kafka Architecture

Kafka UI

Kafka UI is a versatile, fast, and lightweight web interface for managing and monitoring Kafka clusters.

It is an open-source tool that assists in observing data flows, troubleshooting, managing, and analyzing performance.

Its dashboard enables tracking of key metrics for Brokers, Topics, Partitions, and Event Production and Consumption.

Key Features of Kafka UI

- **Message exploration** - navigate through topic messages using Avro, Protobuf, JSON, or plain text encoding.

- **Consumer group visualization** - view per-partition parked offsets and delays.

© Copyright Tecnisys 2026. All rights reserved.

- **Setup assistant** - configure Kafka clusters through a web interface (UI).
- **Multi-cluster management** - monitor and manage all clusters in one place.
- **Performance monitoring with a metrics dashboard** - track and display key Kafka metrics.
- **Visualization of Kafka Brokers** - view topic and partition assignments and controller status.
- **Kafka topics visualization** - view counters, replication status, and custom configurations.
- **Dynamic topic configuration** - create and configure new topics with dynamic settings.
- **Role-based access control** - manage permissions to access the UI with granular precision.
- **Data masking** - obscure sensitive data in topic messages.

Resources of Apache Kafka

- **Open-source tools for large ecosystems**:

  A wide range of community-driven tools.

- Kafka APIs: In addition to command-line tools, Apache Kafka provides five APIs for Java and Scala for management and administration tasks:
  - Admin API Manage and inspect topics, brokers, etc.
  - Producer-API Publish (write) streams of events to one or more topics.
  - Consumer-API: Subscribe (read) one or more topics and process the stream of events produced.
  - Kafka Streams API: Once data is stored as an event in Kafka, it can be processed with the Kafka Streams client library for Java/Scala, which enables the implementation of real-time, mission-critical applications and microservices

© Copyright Tecnisys 2026. All rights reserved.

where the inputs and/or outputs are stored in Kafka topics.
Kafka Streams combines the simplicity of writing and deploying Java and Scala client-side applications with the benefits of server-side cluster technology, making these applications highly scalable, elastic, fault-tolerant, and distributed.
    - Kafka Connect API: Build and run data import/export connectors that consume or produce event streams.
- Operational Monitoring: Kafka is often used for Operational Monitoring, involving statistics from distributed applications for production of centralized sources of operational data.

- Log Aggregation Solution Kafka can be used as a Log Aggregation Solution, collecting physical log files from servers and placing them in a centralized location (such as HDFS) for processing.
Compared to log-centered systems like Flume and Scribe, it offers similar performance with stronger durability guarantees due to replication and lower end-to-end latency.

- Event Sourcing Kafka can be used in Event Sourcing, storing changes to the application state as a sequence of events that can not only be queried but also used to reconstruct past states and retroactively change them.

- Log Compaction Kafka can serve as an external commit-log for distributed systems. The log helps replicate data between nodes and acts as a resynchronization mechanism for failed nodes. The Kafka Log Compaction feature supports this use. In this respect, Kafka is similar to the Apache Bookkeeper project.

- Replication Apache Kafka replicates the log for each topic partition across a configurable number of servers (this replication factor can be set on a per-topic basis).
This helps in automatic failover for these replicas when a server in the cluster fails, so messages remain available in the presence of failures.

- Quotas The Kafka cluster has the capability to "enforce" quotas on requests to control the resources of the broker used by the client.
Two types of client quota can be applied to each client group that shares a quota:

    - Network bandwidth quotas: define the limits of the byte rate (from 0.9).

© Copyright Tecnisys 2026. All rights reserved.

- Request rate quotas: define the limits of CPU utilization as a percentage of network and I/O threads (from 0.11).

Best Practices for Apache Kafka

- **Data Validation:** During the recording of a producer system, it is essential to conduct validation tests on the data that will be recorded in the cluster (non-null values for key fields, for example).

- **Exceptions:** During the recording of a Producer or Consumer, it is important that exception classes be defined and the actions to be taken in accordance with business requirements are established.
  This helps not only in debugging but also mitigates risks (alerts for defined situations, for example).

- **Number of retries:** Generally, there are two types of error in a producer application: errors that are "solvable" with a new attempt (such as network timeouts and "leader not available") and errors that need to be handled by the producer.
  Configuring the number of retries helps mitigate risks related to message loss due to Kafka cluster or network errors.

- **Number of bootstrap URLs:** It is important to have more than one broker listed in the bootstrap broker configuration in the producer program.
  This assists producers in adjusting when there are failures due to unavailability of a broker.
  Producers try to use all the brokers listed until they find one with which they can connect.
  Ideally, all the brokers in the Kafka cluster should be listed to accommodate all connection failures.
  However, in the case of very large clusters, a smaller number that can significantly represent the brokers in the cluster may be chosen.
  Note that the number of retries can affect end-to-end latency and cause duplication of messages in the Kafka queue.

- **New partitions in existing topics:** New partitions in existing topics should be avoided when using key-based partitioning for message distribution.
  Adding new partitions can change the hash calculated for each key because it considers the number of partitions as one of its inputs.

© Copyright Tecnisys 2026. All rights reserved.

This would result in different partitions for the same key.
- **Rebalancing:** Whenever a new consumer joins consumer groups or an old one becomes inactive, a rebalance of partitions in the cluster is triggered.
  Whenever a consumer is losing ownership of its partition, it is important to commit the offsets of the last event received from Kafka.

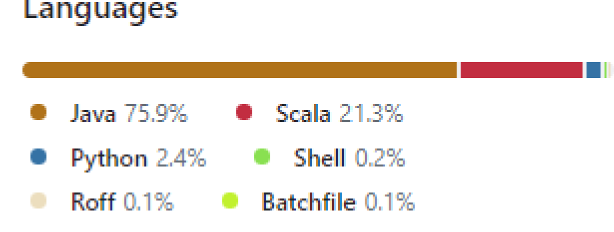
- **Commit offsets at the right time**: In the case of commit offset for messages, it is necessary to do so at the right time. A batch processing application takes more time to complete the processing.
  It is not a rule, but if the processing lasts more than a minute, it is reasonable to perform the commit the offset at regular intervals to avoid duplicate data processing in case of application failure.
  For more critical applications where this duplication may cause financial problems, the time to commit offset should be as short as possible if throughput is not an important factor.

Other recommendations

The blog.kafka.br provides a series of discussions and recommendations that are worth knowing.

Details of Project Apache Kafka

Kafka was written in the programming languages Java and Scala.



Kafka Languages

TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.

> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **4.1.0**, distributed via Helm Chart `tdp-kafka` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources:

- kafka.apache.org
- Central concepts of Kafka - kafka.apache.org
- blog.kafkabr
- cwiki.apache.org/confluence/display/kafka
- github.com/apache/kafka

© Copyright Tecnisys 2026. All rights reserved.

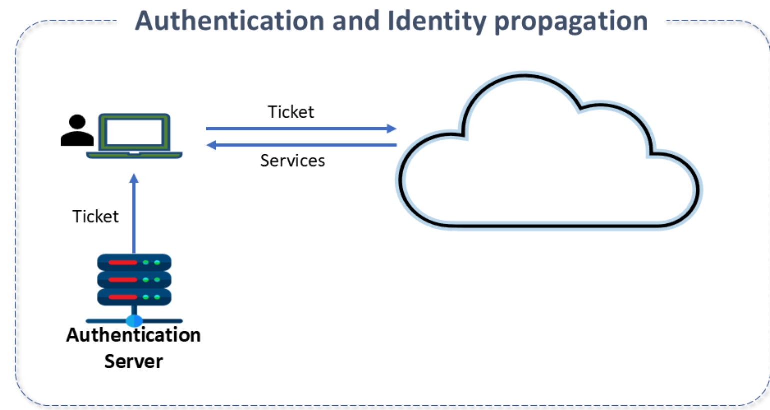
Kerberos

Authentication and Identity Propagation

Authentication can be categorized into two types:

- **Service Authentication**: Verifies the identity among different service components like HDFS, YARN, MapReduce, etc.
- **User Authentication**: A process enabling a device to verify the identity of a user/client connecting to a network resource. Without user authentication, the service merely trusts the identity information provided by the client.

  In most scenarios, a *password* is used as proof of identity. However, it is necessary to prevent the interception or "eavesdropping" of this *password* and to provide a means of user authentication such that whenever a user requests a service, they must prove their identity.



Authentication and Identity Propagation

Features of Kerberos

© Copyright Tecnisys 2026. All rights reserved.

Kerberos is the result of an effort by the MIT - Massachusetts Institute of Technology, known as "Project Athena," started in 1983. It's an open-source computer network authentication protocol that provides Single Sign-On (SSO) based on a trustworthy third party service (users and services rely on a third party: the Kerberos server). It has been adopted by the Hadoop team as a component for authentication to access the Hadoop Cluster. Among its main features, we highlight:

- **Reliability**:
    - Controlled access services are only available if Kerberos is also available. It uses a distributed architecture of servers, with systems enabled to back up others.
    - It is a mutual authentication system ensuring not just that the user is who they claim to be, but also that the services the user is accessing are those expected. Both users and the server will always be assured that the counterpart they are interacting with is authentic.
- **Scalability**: Passwords or secret keys are known only by the Key Distribution Center (KDC) and the Kerberos principals (unique identities such as users or services), making the system scalable to authenticate a large number of entities. Each entity only needs to know its own secret key and register it in advance with the KDC.
- **Security**: The user's password is never transmitted over the network. Kerberos uses temporary tickets (such as the TGT and Service Ticket), which are issued by the authentication server (KDC) and have a limited lifetime.
- **Transparency**: The user is unaware of the authentication process itself, except for the request for a password.
- **Simplicity**: Utilizes the SSO (Single Sign-On) system, where a single ticket can be used by all services until the validity expires. Simplifies user management: Creating, deleting, updating users in Kerberos is very simple.
- **Speed**, as it uses Symmetric Key Operations, which are always faster in SSL authentication operations, which is based on public-private keys.
- **Adaptability**, as it integrates easily with enterprise identity providers such as Active Directory, FreeIPA, or LDAP-based systems.

Architecture of Kerberos

© Copyright Tecnisys 2026. All rights reserved.

Kerberos works in a client-server model. Its basic operation consists of:

- **A ticket**, which is a type of certificate, securely informing the identity of the user to whom access was originally granted.
- **An authenticator**, which is a credential generated by the client with information that will be compared with the ticket, thereby ensuring that the client presenting the ticket is the same for whom the ticket was granted.
- **A key distribution center**, which provides valid temporary tickets to the user to access an application, which will be ratified by the authenticator. The application examines the ticket and the authenticator for validity and grants access if they are valid.

To authenticate and verify the identity of consumers, Kerberos uses symmetric key encryption (the same key is used to encrypt and decrypt data) and a central component called the Key Distribution Center (KDC), which maintains a database of all secret keys. This process involves three main components:

- **Authentication Server (AS)** – performs the initial authentication of the user and issues the Ticket Granting Ticket (TGT).
- **Ticket Granting Server (TGS)** – issues service-specific tickets based on the TGT, eliminating the need to reuse the user's password.
- **Kerberos Database** – stores the secret keys and identities of all users and services authorized in the system.

The guiding idea of the solution is the existence of a server capable of delivering tickets to the user to access services. These tickets remain valid for a certain period. The service does not need to query the KDC to validate the ticket, as it can validate it locally using its own shared key with the KDC:

- The client requests a ticket from the Kerberos server.

- The client submits the ticket to the desired service and is authenticated.

NOTE

There is no possibility of falsifying an identity or forging/reusing a ticket.

Service Authentication

© Copyright Tecnisys 2026. All rights reserved.

Services authenticate with Kerberos when they are initiated, through the Kerberos Principal (a unique identity that can receive tickets for authentication) and the keytab (which contains the authentication credentials of cluster resources).

The Kerberos Principal authenticates the service through the key in the keytab.

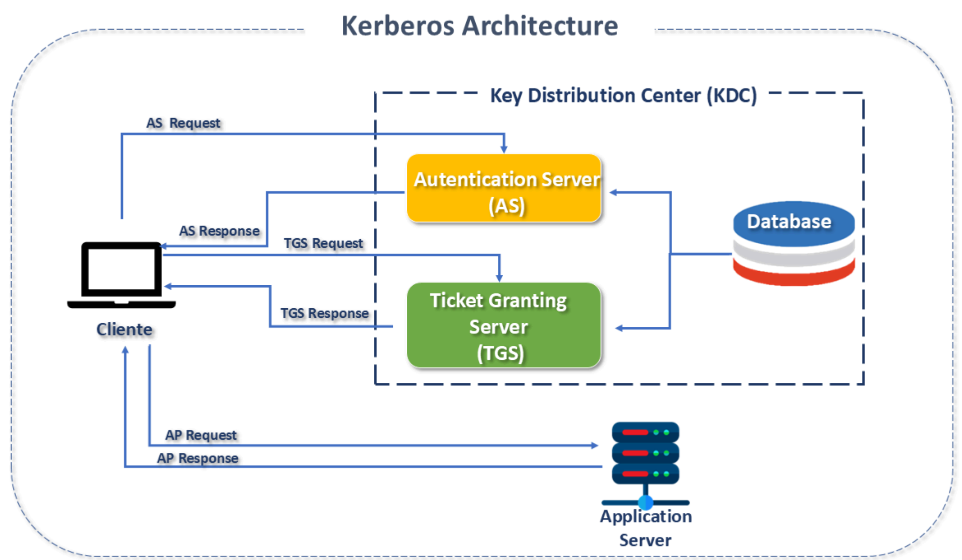
After authentication, the KDC issues the ticket, which will be inserted into the private credentials set. The service can then serve the client.

The process of user authentication occurs, briefly, as follows:

- The user authenticates using their Kerberos Principal.

> NOTE
>
> Initially, the user must log in to the client machine that is enabled to communicate with the Hadoop Cluster.

- The user executes the `kinit` command with the Kerberos Principal and the password.
- `Kinit` will authenticate the user at the KDC, obtaining the resulting ticket and placing it in the ticket cache on the filesystem.



Kerberos Architecture

Best Practices for Kerberos

© Copyright Tecnisys 2026. All rights reserved.

- Encryption configurations in Kerberos are usually set to a variety of types, including "weak" choices such as DES by default. It is recommended to remove weak types to ensure the best possible security.

- When using AES-256, the Java Cryptographic Extensions need to be installed on all cluster nodes to allow encryption types of "unlimited strength". It is important to note that some countries prohibit the use of these types of encryption.

- In environments where Active Directory (AD) users need to access Hadoop Services, it is recommended to establish unidirectional trust between Hadoop Kerberos and the AD Domain.

- As Kerberos is a time-sensitive protocol, all hosts in the domain must be synchronized by time, for example, using the Network Time Protocol (NTP). If the local system time of a client differs from that of the KDC by just 5 minutes (the default), the client will not be able to authenticate.

Details of Project Kerberos

Kerberos was developed in C.

Sources:

- MIT Portal
- Wikipedia

© Copyright Tecnisys 2026. All rights reserved.

Apache Knox

Perimeter Security

Perimeter security refers to the natural or "constructed" barriers to keep intruders out or captives within the boundaries of our solutions.

It assists in protecting the cluster resources and provides a single access point for all REST and HTTP interactions, simplifying client interaction.

Among its main benefits, we can mention:

- "Hides" specific URLs/ports, acting as a proxy.
- Simplifies authentication of multiple services and UIs.
- Enables SSL termination at the perimeter.
- Facilitates the management of endpoints.
- Provides detailed access logs.

Typical perimeter security includes technologies like proxies, firewalls, intrusion detection systems (IDS), and virtual private network (VPN) servers. A combination of all these provides reinforced security.

© Copyright Tecnisys 2026. All rights reserved.

Knox Features

Apache Knox acts as a kind of application proxy, within the perimeter layer, where it receives requests intended for another server and acts on behalf of the client to obtain the requested resource.

It is a system created to extend and simplify the reach of Apache Hadoop services to users outside the cluster, without compromising the security of the ecosystem. Designed as a reverse proxy (a server that resides in front of one or more web servers, intercepting client requests with the aim of enhancing security, reliability, and performance).

Apache Knox integrates with identity management and SSO (single sign-on) systems and allows the identity of these systems to be used for access to Hadoop clusters.

Knox delivers three groups of user-oriented services:

- **Proxy Services:** Through HTTP resource proxying.
- **Authentication Services:** Authenticating access to the REST API, as well as WebSSO flow for UIs. LDAP/AD, Header-based PreAuth, Kerberos, SAML, and OAuth are options.
- **Client Services:** Using scripts via DSL or Knox Shell classes directly as SDK. The interactive script environment KnoxShell combines the interactive Groovy Shell with the Knox Shell SDK Classes for interaction with the data of the deployed Hadoop cluster.

Knox Benefits

The main advantages of the Knox Gateway are:

- **Simplified Access:** Knox extends the REST/HTTP services of Hadoop by encapsulating Kerberos within the cluster.
- **Increased Security:** Exposes the REST/HTTP services of Hadoop without revealing network details, providing SSL/TLS out-of-the-box.

Knox will delegate external client requests to the corresponding Hadoop services and, before delegating, provides all security services configured in the cluster:

© Copyright Tecnisys 2026. All rights reserved.

- **Centralized Control:** Enforces REST API security "centrally," routing requests to multiple Hadoop clusters.
- **Corporate Integration:** Supports LDAP, Active Directory, SSO, SAML, and other authentication systems.
- **Demonstration LDAP:** Available by default in Apache Knox.
- **Audit Logging.**

Apache Knox Architecture

Apache Knox acts as a single point of contact for Apache Hadoop services in the cluster.

It runs as a server cluster, in the DMZ (demilitarized zone - situated between a trusted network and an untrusted network, providing physical isolation between the two), isolating the Hadoop cluster from the rest of the corporate network.

Its main feature is the provision of a security perimeter for Hadoop REST APIs, restricting the number of network endpoints required to access the Hadoop cluster.

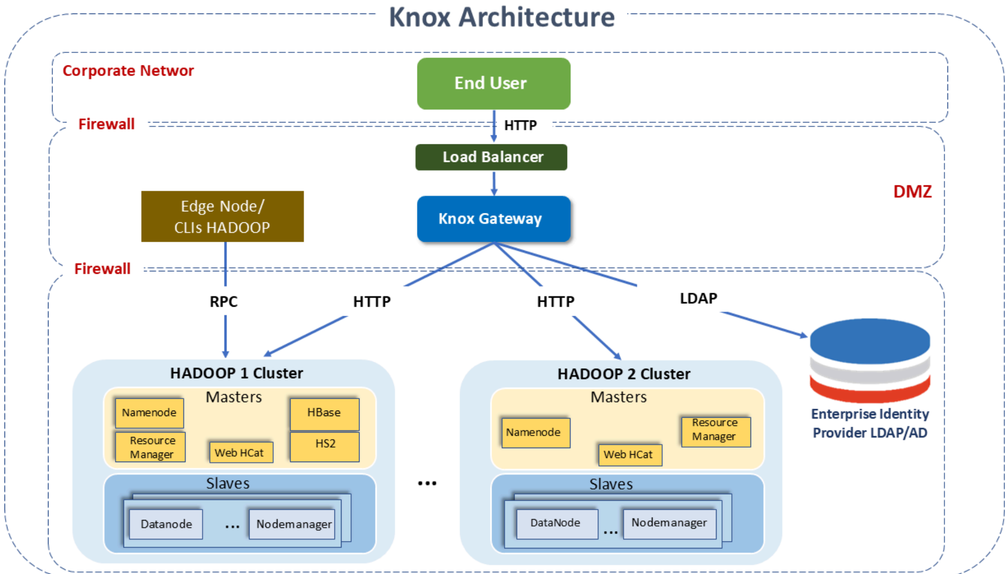
As a result, it "hides" the topology of the Hadoop cluster. At the network perimeter, it provides a single point of authentication and token verification.

Knox can be used with both Kerberized (protected by Kerberos) and non-Kerberized clusters.

In an enterprise solution employing Kerberos-protected clusters, it provides security that integrates well with enterprise identity management solutions, and as mentioned earlier, protects the implementation details of the Hadoop cluster and simplifies the number of services with which a client needs to interact.

The deployment architecture of the Knox Gateway can be understood in the following diagram:

© Copyright Tecnisys 2026. All rights reserved.



Knox Architecture

How Apache Knox Works

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

It provides a single point of access for REST and HTTP interactions with Apache Hadoop clusters. To this end, it offers three groups of user-oriented services:

- **Proxy Services:** Provides access to Apache Hadoop through HTTP resource proxying.
- **Authentication Services:** Authentication for REST API access, as well as WebSSO flow for UIs. LDAP/AD, Header based PreAuth, Kerberos, SAML, OAuth are all available options.
- **DSL/SDK Client Services:** Client development can be done with scripting through DSL or Knox Shell classes directly as SDK. The interactive script environment KnoxShell combines the interactive groovy shell with the Knox Shell SDK classes for an interaction with data from the deployed Hadoop cluster.

The Knox Gateway was designed as a reverse proxy with consideration for plugability in areas of policy enforcement, through the providers and back-end services to which it proxies requests.

Policy enforcement covers authentication/federation rules, authorization, auditing, dispatch, host mapping, and content rewriting. Policy is enforced through a chain of providers that are defined in the topology deployment descriptor for each Hadoop

© Copyright Tecnisys 2026. All rights reserved.

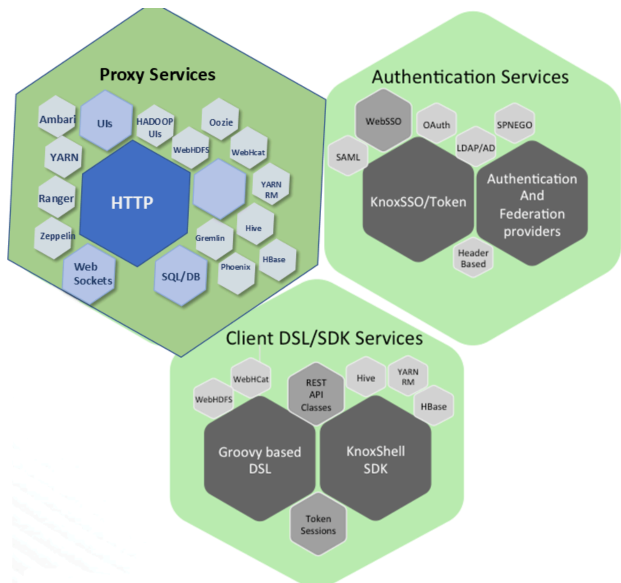
cluster controlled by Knox. The cluster definition is also made in the deployment descriptor and provides the Knox Gateway with the layout of the cluster for the purpose of routing and converting between user-facing and internal cluster URLs.

Each Hadoop cluster protected by Knox has its set of REST API represented by a single application context path specific to the cluster. This allows Knox to protect multiple clusters and present to the REST API consumer a single endpoint for access.

With the writing of the deployment descriptor in the topologies directory of the Knox installation, a new Hadoop cluster definition is processed, policy enforcement providers are configured, and the application context path is made available for use by REST API consumers.

Knox also complements Kerberos protection of clusters well, providing appropriate network isolation a

- integrating well with enterprise identity management solutions.
- protecting the deployment details of the cluster (hosts, ports).
- simplifying the number of services with which clients need to interact.



Knox Services

Apache Knox Resources

- **Configuration for New Services and UIs:** Apache Knox provides a targeted method of configuration to add new routing services. This enables new Hadoop REST APIs to be quickly and easily incorporated. This functionality was added in release 0.6.0.

© Copyright Tecnisys 2026. All rights reserved.

- **Homepage:** Apache Knox provides a HomePage that can be used as a "front-door" to implementations and resources that are published for access through it. It is a good alternative for distributing a link to the administrative interface and obtaining Quick Links.

- **Authentication:** The authentication function providers are responsible for collecting the credentials presented by the API consumer, validating them, and communicating the success or failure of the authentication to the client or the rest of the provider chain.

  Knox comes with the Shiro authentication provider, which leverages the Apache Shiro project to authenticate BASIC credentials in an LDAP user repository. Support for OpenLDAP, Apache DS, and Microsoft Active Directory is available.

- **Federation/SSO:** For clients that require credentials to be presented to a limited set of trusted entities within the enterprise, Knox can be configured to federate the authenticated identity from an external authentication event. This is done through providers with the federation function. The set of ready-to-use federation providers include:

  - **Standard KnoxSSO Form-Based IDP:** The default KnoxSSO configuration provides a form-based authentication mechanism that leverages Shiro authentication to authenticate in LDAP/AD with credentials collected from a form-based challenge.

  - **PAC4J:** The pac4j provider adds authentication and federation capabilities such as SAML, CAS, OpenID Connect, Google, Twitter, etc.

  - **HeaderPreAuth:** A mechanism for propagating identity through HTTP Headers that specifies the username and group for the authenticated user. This has been used for use cases like SiteMinder and IBM Tivoli Access Manager.

- **Knox SSO:** The Knox SSO service is an integration service that provides a normalized SSO token to represent the authenticated user.

  This token is generally used for WebSSO resources for participating UIs and their consumption of Hadoop REST APIs.

© Copyright Tecnisys 2026. All rights reserved.

Knox SSO abstracts the actual identity provider integration from the participating applications so that they need only be "aware" of the KnoxSSO cookie.

The token is presented by the browser as a cookie, and applications participating in the KnoxSSO integration can cryptographically validate the presented token and remain independent of the underlying SSO integration.

- **Authorization:** The authorization function is used by providers that make access decisions for requested resources based on the effective user identity context.

  This identity context is determined by the authentication provider and the identity assertion provider mapping rules.

  The evaluation of the user identity context and group principals against a set of access policies is done by the authorization provider to determine whether access should be granted to the user for the requested resource.

  Knox comes with an ACL-based authorization provider that evaluates rules that include username, groups, and IP addresses. These ACLs are bound and protect resources at the service level. That is, they protect access to the Hadoop services themselves, based on the user, group, and remote IP address.

- **Auditing:** Auditing allows determining which actions were performed by whom over a specific period.

  The installation is built on an extension of the Log4j framework and can be extended by replacing the "out-of-the-box" implementation with another.

Recommendations and Best Practices

- Integration of Knox with Apache Ranger is recommended to check the permissions of users who wish to access the cluster resources.
- Enabling SSL is highly recommended. In the case of disconnected Hive connections via Apache Knox, it is recommended to modify the connection timeout and the maximum number of connections in the gateway site configuration file.
- To improve response times through the Apache Knox configuration, configure the following properties in the gateway.site: `gateway.metrics.enabled=false`, `gateway.jmx.metrics.reporting.enabled=false`, `gateway.graphite.metrics.reporting.enabled=false`.

© Copyright Tecnisys 2026. All rights reserved.

Supported Services by Knox

The following services of the Hadoop Ecosystem have integration with the Knox Gateway: (these services can be consulted on the community page - item "Supported Apache Hadoop Services" / "Supported Apache Hadoop ecosystem UIs")

Supported Components by Knox

Component	SSO	Proxy (API)	Proxy (UI)
Ambari	YES	YES	YES
Métricas Ambari/Grafana			
Atlas	YES	YES	YES
HBase		YES	
HDFS			YES
Hive(via JDBC)		YES	
Hive(via WebHCat)		YES	
MapReduce2	YES		YES
Zookeeper	YES	YES	YES
Spark2/Spark History Server	YES		YES
WebHCat		YES	
WebHDFS		YES	
YARN	YES	YES	YES

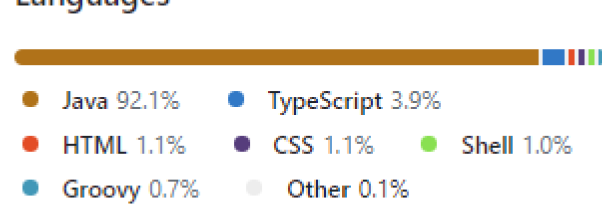
© Copyright Tecnisys 2026. All rights reserved.

Component	SSO	Proxy (API)	Proxy (UI)
Zeppelin	YES	YES	YES

TDP v2 - Knox - Services Supported by Knox with Proxy and SSO (Kerberized or non-Kerberized Clusters)

Apache Knox Project Details

Apache Knox was developed in Java.



Languages of Knox

Sources:

- Apache Knox Community
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Livy

Spark Session Management

Apache Livy is an open-source service platform that provides an easy way to interact with an Apache Spark cluster through a REST API. It essentially acts as a bridge between applications and Apache Spark, allowing users and applications to submit and manage Spark jobs remotely and interactively. Livy is designed to simplify the process of submitting Spark jobs, managing the complexity and details of communication with the Spark *cluster*.

Livy is ideal for scenarios requiring programmatic submission of Spark jobs, such as in batch data processing systems, large-scale data analysis, and integrations in data science environments.

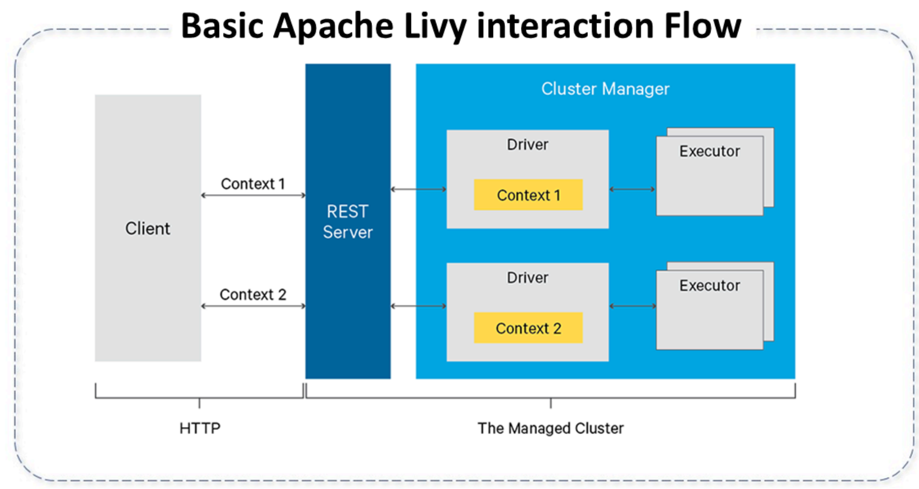
Apache Livy was initially developed by Cloudera as a solution to simplify interaction with Spark *clusters*. From the beginning, Livy's focus has been to provide a RESTful interface to facilitate the submission and management of Spark jobs on data *clusters*, especially in environments where direct interaction with Spark is complex.

The evolution of Livy aligns with the development of Apache Spark, reflecting the growing need for tools that facilitate access to powerful data processing capabilities.

Features of Apache Livy

- **Spark Job Submission**: Allows the submission and management of Spark jobs through a REST API;
- **Multi-language Support**: Supports jobs written in Scala, Python, and R;
- **Session Management**: Manages Spark sessions, facilitating reuse and optimizing resource use;
- **Security and Access Control**: Includes features to ensure data security and access to the *cluster*;
- **Integration with Hadoop YARN**: Facilitates resource management in *clusters* through integration with YARN.

© Copyright Tecnisys 2026. All rights reserved.

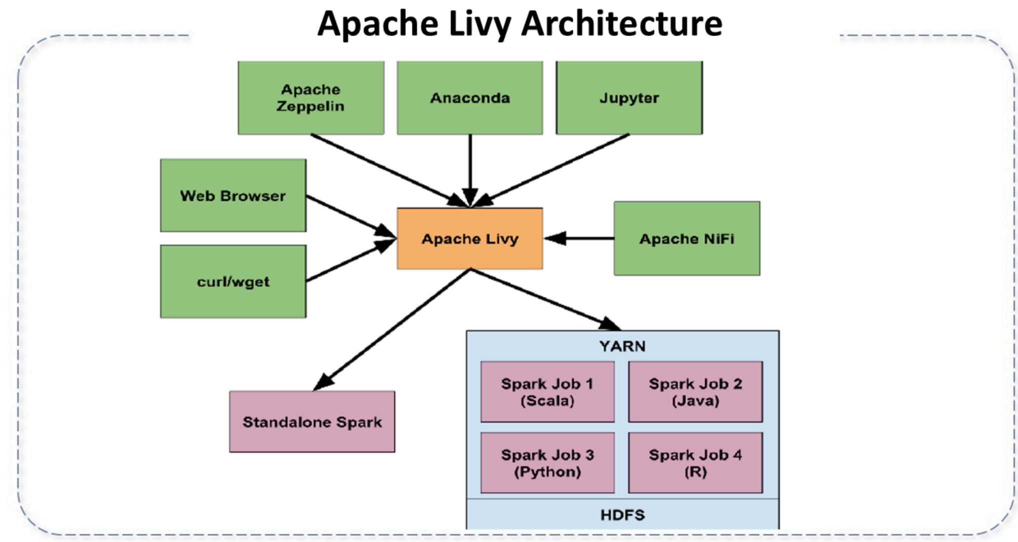


Basic Flow Apache Livy - job submission through Livy

Architecture of Apache Livy

The main components of Apache Livy are:

- **REST Server**: Provides a REST interface for the submission and management of Spark jobs.
- **Session Manager**: Responsible for starting, maintaining, and ending Spark sessions.
- **Integration with YARN**: Manages the *cluster* resources efficiently.
- **Programming Interface**: Provides APIs to facilitate the submission and control of Spark jobs.



© Copyright Tecnisys 2026. All rights reserved.

Architecture of Apache Livy

Best Practices in Using Apache Livy

- **Efficient Session Management**: Manage Spark sessions to optimize resource use;
- **Security**: Implement robust security controls;
- **Monitoring**: Monitor the performance of sessions and the *cluster*;
- **Documentation and Integration**: Keep documentation up to date on the integration of Livy with other systems;
- **Updates and Compatibility**: Stay informed about new versions of Livy and Spark.

When Not to Use Apache Livy

- **Low Latency Environments**: Livy may not be ideal for scenarios that require very low latency;
- **Simple and Isolated Jobs**: For Spark jobs that do not require complex integration, using Livy may be overkill.

Development Project Details

Apache Livy is primarily developed in Java. This means that an understanding of Java is beneficial for contributing to the project or integrating it more effectively into existing systems. Additionally, understanding Java programming paradigms can help optimize interaction with Livy and implement specific customizations as needed.

Sources:

- Livy.apache.org

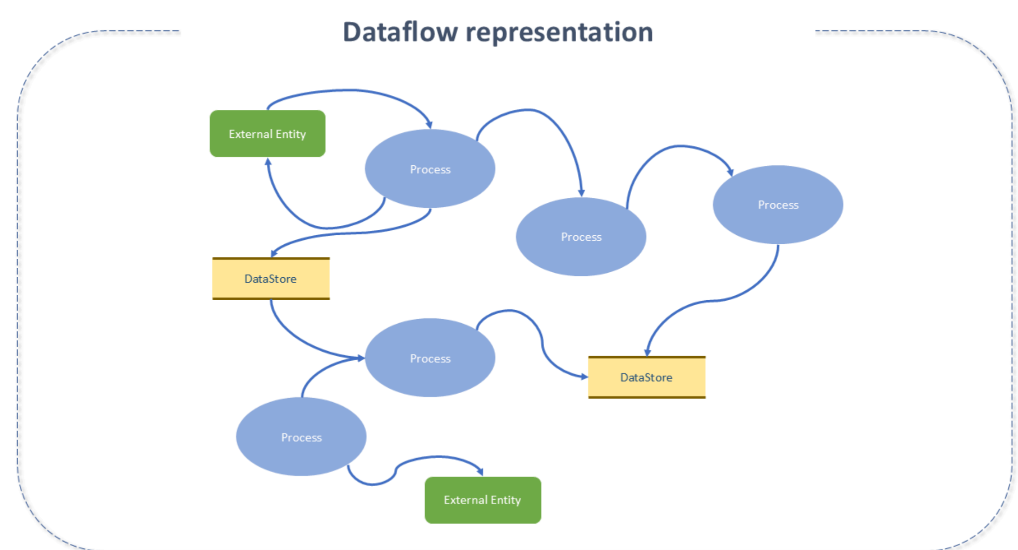
© Copyright Tecnisys 2026. All rights reserved.

Apache NiFi

Data Flow Automation

Data flow automation is a software paradigm based on the idea of computing as a directed graph, representing the automated and managed flow of information between systems.

Data flow automation helps in capturing, building, and collaborating at scale through automated tools, ensuring greater system efficiency through optimized and controllable reports.



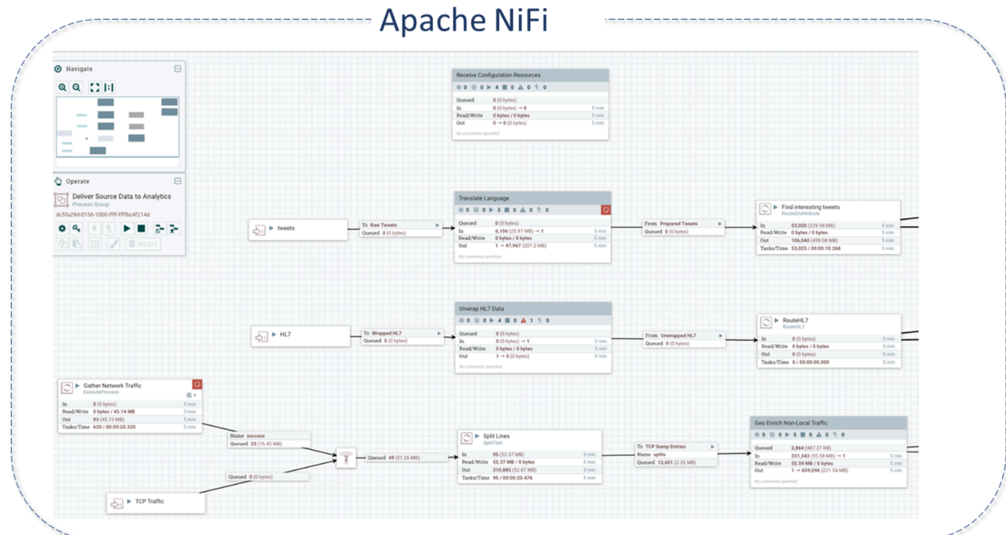
Representation of a Data Flow

Features of Apache NiFi

NiFi was built to automate data flows, support and meet the level of rigor necessary to achieve compliance, privacy, and security in data exchange between systems.

NiFi has become especially valuable for edge solutions that have emerged in recent years, for which data flow is vital, such as SOA (Service Oriented Architecture), APIs (Application Protocol Integration), IoT (Internet of Things), and Big Data.

© Copyright Tecnisys 2026. All rights reserved.



NiFi Interface

The fundamental design concepts of Apache NiFi are closely related to the ideas of Flow Based Programming - fbp.

NiFi Term	FBP Term	Description
FlowFile	Packet Information	The *FlowFile* represents each object moved between/within systems.
FlowFile Processor	Black Box	It is the Processors that actually perform the work.
Connection	*Bounded Buffer*	Connections provide the actual link between processors.
Flow Controller	*Scheduler*	Provides threads for extensions to execute and manages the schedule for when extensions receive resources for execution.

© Copyright Tecnisys 2026. All rights reserved.

Table D - NiFi - Concepts Associated with FBP

This design model makes the NiFi platform very effective for building powerful and scalable data flows, enabling:

- Visual creation and management of directed graphs (with directions associated with each edge) of processors.
- High throughput and a natural data buffer, despite fluctuations in processing and flow rates.
- Highly concurrent models, with the ability to execute several distinct tasks simultaneously (or apparently simultaneous), allowing the developer to disregard the typical complexities of this topic.
- The development of cohesive, low-coupling components, reusable in other contexts, and testable units.
- The simplification of critical data functions, such as back-pressure and pressure release, becoming natural and intuitive (through the constrained connections feature).
- Error handling as naturally as the happy path.
- Easy understanding and tracking of points where data "enters" and "exits" the system.

Apache NiFi Architecture

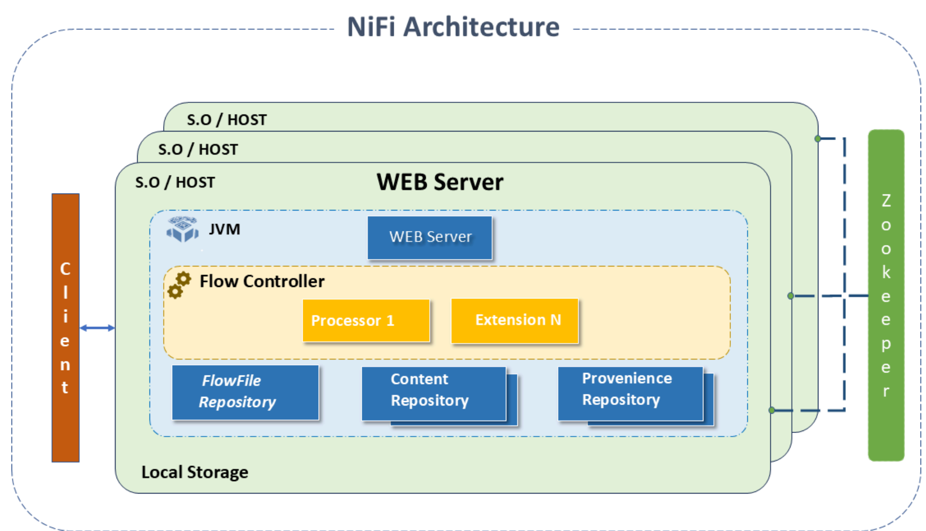
NiFi runs within a JVM virtual machine, on a host operating system. Its main components in the JVM are:

- **Web Server**: Executes visual control and monitors events. Hosts the HTTP-based command and control API.
- **Flow Controller**: The "brain" of the operation. Provides threads for extensions to execute and manages the schedule for when extensions receive resources for execution.
- **Extensions**: Plugins that allow interaction between NiFi and other Systems. NiFi provides several "extension" points for developers to add functionalities. All operate and execute within the JVM:
    - **Processor**: The processor interface provides access to FlowFiles, their attributes, and contents.
    - **ReportingTask**: Provides metrics, monitoring information, and the internal state of NiFi to endpoints such as log files, email, and remote web services.
    - **ParameterProvider**: Provides parameters for use by external sources.

© Copyright Tecnisys 2026. All rights reserved.

- **ControllerService**: Provides a mechanism to create services shared among all Processors, ReportingTasks, and other ControllerServices in a single JVM.
- **FlowFilePriorizer**: Provides a mechanism by which FlowFiles in a queue can be prioritized or sorted for subsequent processing in a more effective order in case of specific use.
- **AuthorityProvider**: Determines what privileges and roles, if any, should be granted to a given user.

- **FlowFile Repository**: Where NiFi keeps/tracks the status of active FlowFiles. The repository implementation is pluggable. The default is a persistent write-ahead log located on a specific disk partition.
- **Content Repository**: Where data "in transit" is kept. It is pluggable. The default approach is a simple mechanism that stores data blocks on the file system. More than one file system storage location can be specified to involve different partitions in reducing contention on any single volume.
- **Provenance Repository**: Stores all information about the provenance of data that circulates through the system. The repository is pluggable, with the default implementation using one or more physical disk volumes. Within each location, event data is indexed and enabled for search.



NiFi Architecture

NiFi is also enabled to operate within a Cluster.

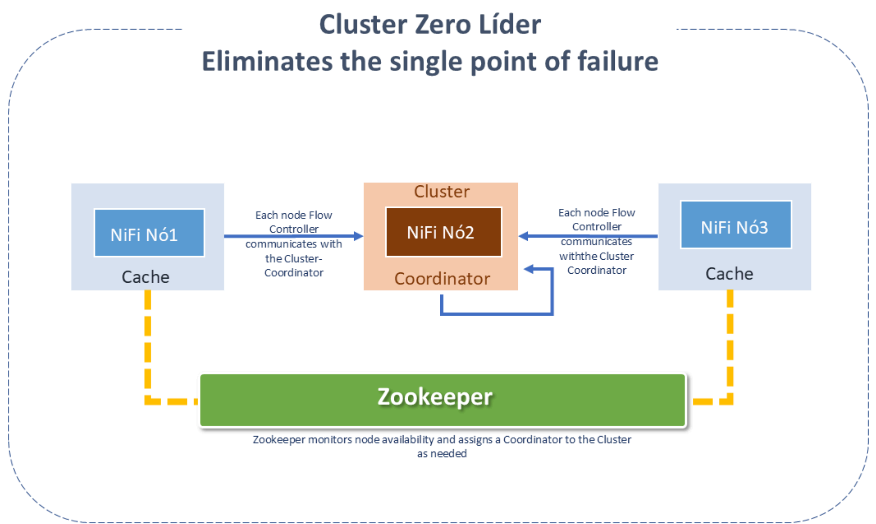
Since NiFi version 1.0, NiFi employs the Zero-Leader paradigm: each NiFi Cluster node performs the same tasks on the data, but each operates on a distinct data set.

© Copyright Tecnisys 2026. All rights reserved.

Apache Zookeeper elects a single node as the Coordinator. All Cluster nodes report heartbeat and status to the Coordinator, who is responsible for disconnecting and connecting nodes.

Additionally, every Cluster has a primary Node, also elected by Zookeeper.

Failover is handled automatically by Zookeeper.



Zero Leader Cluster

Just like the Dataflow manager, interaction with the NiFi Cluster can be done through the user interface (UI) of any node. Any changes are replicated to all nodes in the Cluster, allowing multiple entry points.

Performance Expectations

Apache NiFi was designed to fully leverage the resources of the host system it is operating on. This resource maximization is particularly strong concerning CPU and disk.

I/O:

Transfer rates or latency may vary depending on system configuration. Considering there are pluggable approaches for most NiFi subsystems, performance will depend on implementation. Consider using ready-to-use default implementations.

© Copyright Tecnisys 2026. All rights reserved.

TIP

We can assume a read/write rate of 50MB per second on modest RAID disks within a common server. For large classes of data flows, NiFi is enabled to achieve 100MB per second or more of throughput, as linear growth is expected for each physical partition and content repository added to NiFi.

CPU:

The flow controller allocates and manages threads for processors. It acts as an engine dictating when a processor will receive the thread to execute.

Additionally, it allows adding controller services, which facilitate resource management, such as database connections or cloud service provider credentials.

The controller services are daemons (executed in the background) and provide configuration, resources, and parameters for processors to execute.

TIP

The ideal number of threads to be used depends on the resources of the host system in terms of the number of cores, if that system is running other services, and the nature of the processing in the flow. For heavy I/O flows, it is reasonable to provide many dozens of threads.

RAM:

NiFi resides in the JVM and is limited by the memory space provided by it. JVM garbage collection becomes a very important factor both in restricting the total heap size and in optimizing the application's execution over time.

TIP

NiFi jobs can be I/O intensive when reading the same content regularly. Configure a sufficiently large disk to optimize performance.

Most Interesting Features of Apache NiFi

© Copyright Tecnisys 2026. All rights reserved.

NiFi Registry

It is a subproject of Apache NiFi - a complementary application that provides "centralized location" for storing and managing resources shared by one or more NiFi or MiNiFi instances.

It offers the following features:

- Implementation of Flow Registry to store and manage versioned flows.
- Integration with NiFi to allow storage, retention, and updating of versioned flows from the Flow Registry.
- Registry administration for defining users, groups, and policies.

The first implementation of the Registry supports versioned flows. Data flows at the process group level created in NiFi can be version-controlled and stored in a registry. The registry organizes where the flows are stored and manages permissions to access, create, modify, or delete them.

The NiFi Registry UI displays the available shared resources and provides mechanisms to create and manage users/groups, buckets, and policies. Once the NiFi Registry is installed, a compatible web browser can be used to view the UI.

NOTE

Buckets are containers that store and organize versioned items, such as flows and bundles (binary artifacts containing one or more extensions that can be run in NiFi or MiNiFi).

MiNiFi

It is a subproject of Apache NiFi - a complementary data collection approach that supplements the core principles of NiFi in data flow management, with a focus on data collection at the point of its creation.

Apache NiFi MiNiFi provides the following features:

- Small size and low resource consumption.
- Centralized agent management.
- Generation of data provenance with the entire information custody chain.
- Integration with NiFi for subsequent data flow management.

© Copyright Tecnisys 2026. All rights reserved.

Flow Management

- **Guaranteed Delivery**: NiFi's main philosophy is that, even at a very high scale, guaranteed delivery is the rule. This is achieved through the effective use of WAL (write-ahead log) and persistent content repository. Together they are designed to enable high transaction rates, effective load distribution, copy-on-write, and to "play" with the strengths of traditional read-writes on the disk.
- **Data Buffering with Back Pressure and Pressure Release**: NiFi supports buffering all queue data and has the ability to provide back pressure to those queues that have reached their limits or to "age" data when it reaches a specified age (its value has perished).
- **Prioritized Queuing**: NiFi allows configuring one or more prioritization schemes for retrieving data from a queue. The default is "oldest first," but there are times when "newer" data, larger, or some other custom scheme is the rule.
- **Flow Specific QoS**: (latency vs throughput, low tolerance, etc.): There are points in the Dataflow where data is absolutely critical and intolerant to failures. There are also times when it must be processed and delivered in seconds to generate some value. NiFi enables refining the flow configuration.

Ease of Use

- **Visual Command and Control**: Dataflows can become complex. Providing good visual expression can help reduce complexity and identify areas that need to be simplified. NiFi enables visual establishment of Dataflows in real-time. Changes to the dataflow take effect immediately. Changes are refined and isolated to the impacted components. It is not necessary to interrupt an entire flow or set of flows just to make a specific modification.
- **Templates**: Dataflows tend to be highly pattern-oriented, and although there are many different paths to solve a problem, sharing best practices by subject matter experts benefits everyone, as well as allowing the experts themselves to benefit from others' collaboration.
- **Data Provenance**: NiFi automatically records, indexes, and makes provenance data available as objects flow through the system, even in fan-in, fan-out, transformations, and others. This information becomes extremely critical in supporting compliance, troubleshooting, optimizations, and other scenarios.
- **Recovery/continuous buffering of refined history**: The NiFi content repository is designed to act as a continuous history buffer. Data is removed from the content repository only when it "ages" or more space is needed. This combined with provenance capability creates an incredibly useful base to enable "click-to-content,"

© Copyright Tecnisys 2026. All rights reserved.

content download, replay, all at a specific point in the object's lifecycle, which can span generations.

Security

- **System to System**: A Dataflow is only good if it is secure. NiFi, at any point in the Dataflow, offers secure exchange using encryption protocols such as two-way SSL. Additionally, it allows the flow to encrypt and decrypt content and use shared keys or other mechanisms on both sides of the sender/receiver equation.
- **User to System**: NiFi allows two-way SSL authentication and provides pluggable authorization to control user access at specific levels (read-only, dataflow manager, administrator). If a user enters a confidential property (such as a password) in the flow, it will be immediately encrypted on the server side and never exposed on the client side, even in its encrypted form.
- **Multi-tenant Authorization**: The level of authority of a Dataflow is applied to each component, enabling the admin user to have a refined level of access control. This means that each NiFi cluster can handle the requirements of one or more organizations. Compared to isolated topologies, multi-tenant authorization allows a self-service model for dataflow management and enables each team or organization to manage flows with full awareness of the rest of the flow, for which they do not have access.

Extensible Architecture

- **Extension**: NiFi is built for extension, and as such, it is a platform on which Dataflow processes can be executed and interact in a repetitive and predictable manner.
- **Classloader Isolation**: For any component-based system, dependency problems can occur. NiFi addresses this by providing a custom class-loading model, ensuring that each extension package is exposed to a very limited set of dependencies. As a result, extensions can be built without worrying about conflicts with other extensions. The concept of these extension packages is called "NiFi files."
- **Site-to-Site Communication Protocol**: The preferred communication protocol between NiFi instances is the NiFi Site-to-Site (s2s) Protocol. S2S facilitates the transfer of data from one NiFi instance to another efficiently and securely. NiFi client libraries can be easily built and bundled into other applications or devices to communicate with NiFi via S2S. Both socket-based and HTTP(s) protocols are supported in S2S as the underlying transport protocol, making it possible to incorporate a proxy server into S2S communication.

© Copyright Tecnisys 2026. All rights reserved.

Flexible Scaling Model

- **Scale-out (Clustering)**: NiFi is designed for horizontal scalability through the use of clustering multiple nodes. If a single node is provisioned and configured to handle hundreds of MB per second, a modest cluster can be configured to handle GB per second. This brings load balancing and failover challenges between NiFi and the systems from which data is obtained. The use of asynchronous queue-based protocols, such as messaging services, kafka, etc., can help. The site-to-site feature is also very effective, as it is a protocol that allows NiFi and a client (including another Cluster) to talk to each other, share loading information, and exchange data through specific authorized ports.
- **Scale-up and down**: NiFi is also designed for very flexible scale-up and scale-down. In terms of throughput, from the NiFi framework perspective, it is possible to increase the number of simultaneous tasks in the processor on the scheduling tab during configuration. This allows more processes to be executed simultaneously, providing great throughput. On the other end of the spectrum, NiFi can be scaled to run on edge devices where hardware resources are limited.

Difference between Apache Airflow and Apache NiFi

By nature, Airflow is an orchestration framework and not a data processing framework.

While NiFi's main goal is related to the stream processing category, automating data transfer between two systems, Airflow is more related to workflow management.

It is important to note that the two tools are not mutually exclusive and both offer interesting features that help solve data silos within the organization.

NiFi is a perfect tool for Big Data. There is no better choice when it comes to the "configure and forget" pipeline type.

Airflow, on the other hand, is perfect for scheduling specific tasks, configuring dependencies, and managing programmatic workflows.

It allows for easy visualization of dependencies, code, trigger tasks, progress, logs, and success status of data pipelines.

Best Practices for Apache NiFi

Separate Environments

© Copyright Tecnisys 2026. All rights reserved.

- **Separate environments for development**: It is essential to maintain separate environments for development, testing, and production to ensure data integrity and system stability.

Consider the User

One of the most important concepts in developing a "processor" or any other component is the user experience:

- Documentation should always be provided so that everyone can use the component easily.
- Consistency (naming conventions), simplicity, and clarity are fundamental principles to make this experience adequate.

Cohesion and Reusability

To create a unique and cohesive unit, developers are tempted to combine several functions into a single processor.

Adopting the approach of formatting data for a specific endpoint and then sending it to this point in the same processor may not be advantageous because:

- It can make the processor very complex.
- If the processor cannot communicate with a remote service, it will forward the data to a failure relationship and will be responsible for translating the data again, and again...
- If we have 5 different processors translating input data into a new format before sending it, we will have a large amount of duplicated code. If the scheme changes, many processors must be updated.
- These intermediate data are discarded when the processor finishes sending them to the remote service. The intermediate format may be useful for other processors.

NOTE

To avoid these problems and make processors more reusable, a processor should always follow the principle: "do one thing and do it well."

It should be divided into two separate processors: one to convert the data from format X to Y and another to send data to the remote resource.

© Copyright Tecnisys 2026. All rights reserved.

Naming Conventions

To provide a consistent appearance to users, it is advisable for processors to maintain standard naming conventions:

- Processors that extract data from a remote system are named:
  - `Get<Service>` (search data from arbitrary sources via known protocol, like GetHTTP, GetFTP) or
  - `Get<Protocol>` (search data from a known service like GetKafka).
- Processors that send data to a remote system are named:
  - `Put<Service>` or
  - `Put<Protocol>`.
- Relationship names are lowercase and use spaces to delineate words.
- Property names should use meaningful words, like a book title.

Processor Behavior Annotations

When creating a processor, the developer must provide hints to the framework on how to use it more efficiently.
This is done by applying annotations to the processor class.
The annotations that can be applied are in three sub-packages of *org.apache.nifi.annotation*:

- ***documentation* subpackage**: Provides documentation to the user.
- ***lifecycle* subpackage**: Instructs the framework on which methods to call on the processor to respond to appropriate lifecycle events.
- ***behavior* subpackage**: Helps the framework understand how to interact with the processor in terms of scheduling and general behavior.

The following annotations from the package can be used to modify how the framework will handle your processor (for more details, click here):

- **EventDriven**: Instructs the framework that the processor can be scheduled using the event-driven scheduling strategy.
- **SideEffectFree**: Indicates that the processor has no external side effects to NiFi.
- **SupportsBatching**: Indicates that it is okay for the framework to batch multiple ProcessSession commits into a single commit.
- **TriggerSerially**: Prevents the user from scheduling more than one concurrent thread to run the *onTrigger* method at a time.
- **PrimaryNodeOnly**: Restricts the processor's execution to only the Primary Node.

© Copyright Tecnisys 2026. All rights reserved.

- **TriggerWhenAnyDestinationAvailable**: Indicates that the processor should run if any relationship is "available" even if one of the queues is full.
- **TriggerWhenEmpty**: Ignores the size of the input queues and triggers the processor regardless of whether there is data in an input queue or not.
- **InputRequirement**: By providing a value through this annotation (INPUT_REQUIRED, INPUT_ALLOWED, or INPUT_FORBIDDEN), the framework will know when it should be invalidated or whether the user should be able to establish a connection to the processor.

Data Buffering

NiFi provides a generic data processing capability. Data can be in any format.
Processors are usually scaled with multiple threads.
A common developer mistake is to buffer all the content of a FlowFile in memory.
Unless absolutely necessary, this should be avoided, especially if the data format is known.

TIP

Instead of buffering this data in memory, it is advisable to evaluate the data as it is streamed from the Content Repository (i.e., the inputStream content provided by the ProcessSession.read callback).

Apache NiFi Project Details

Apache NiFi is developed in Java.

Languages

Java 87.8% JavaScript 6.7%
Groovy 2.5% HTML 1.9%
CSS 0.5% Shell 0.2% Other 0.4%

NiFi Languages

TDP Kubernetes

© Copyright Tecnisys 2026. All rights reserved.

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **1.28.0**, distributed via Helm Chart `tdp-nifi` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources

- NiFi.apache.org
- NiFi.apache.org/registry
- cwiki.apache.org

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone

Corporate Research Platform

This feature is only available starting from version 2.3.

Apache Ozone

Apache Ozone

Object Storage

Ozone is a redundant and distributed object storage system optimized for Big Data workloads. The primary design focus of Ozone is scalability, aiming to scale to billions of objects.

Ozone separates namespace management from block space management, which allows it to scale much more effectively. The namespace is managed by a daemon called the Ozone Manager (OM), while the block space is managed by the Storage Container Manager (SCM).

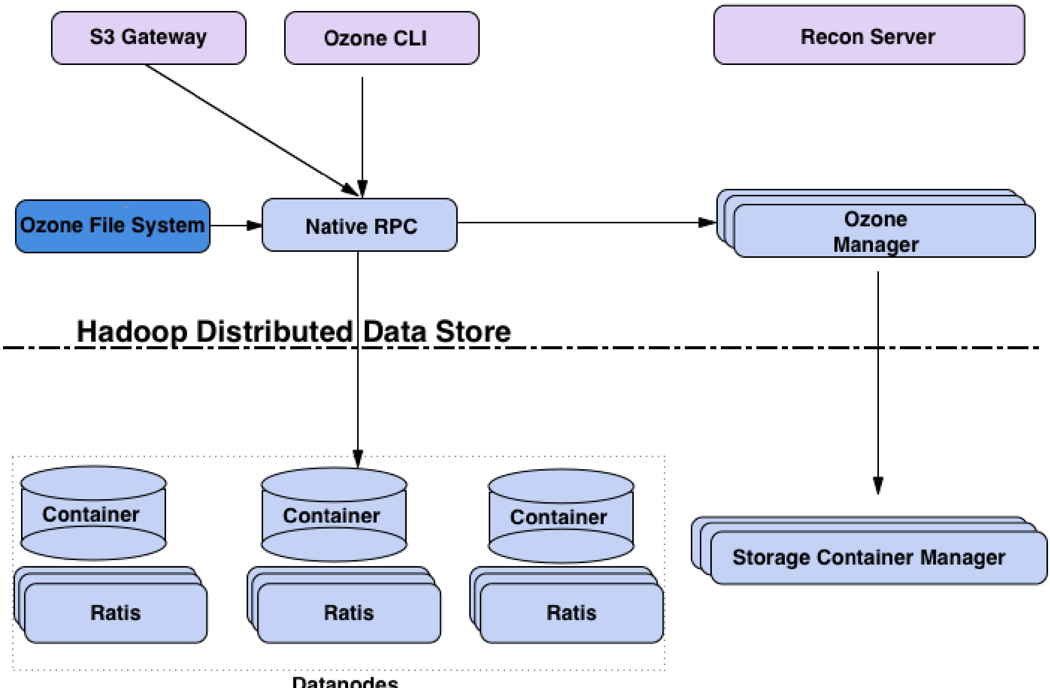
Ozone consists of volumes, buckets, and keys. A volume is similar to a home directory in the Ozone ecosystem. Only an administrator can create it.

Volumes are used to store buckets. Once a volume is created, users can create as many buckets as needed. Ozone stores data as keys that reside within these buckets.

The Ozone namespace comprises multiple storage volumes, which are also used as the basis for storage accounting.

© Copyright Tecnisys 2026. All rights reserved.

The block diagram below illustrates the key components of Ozone.

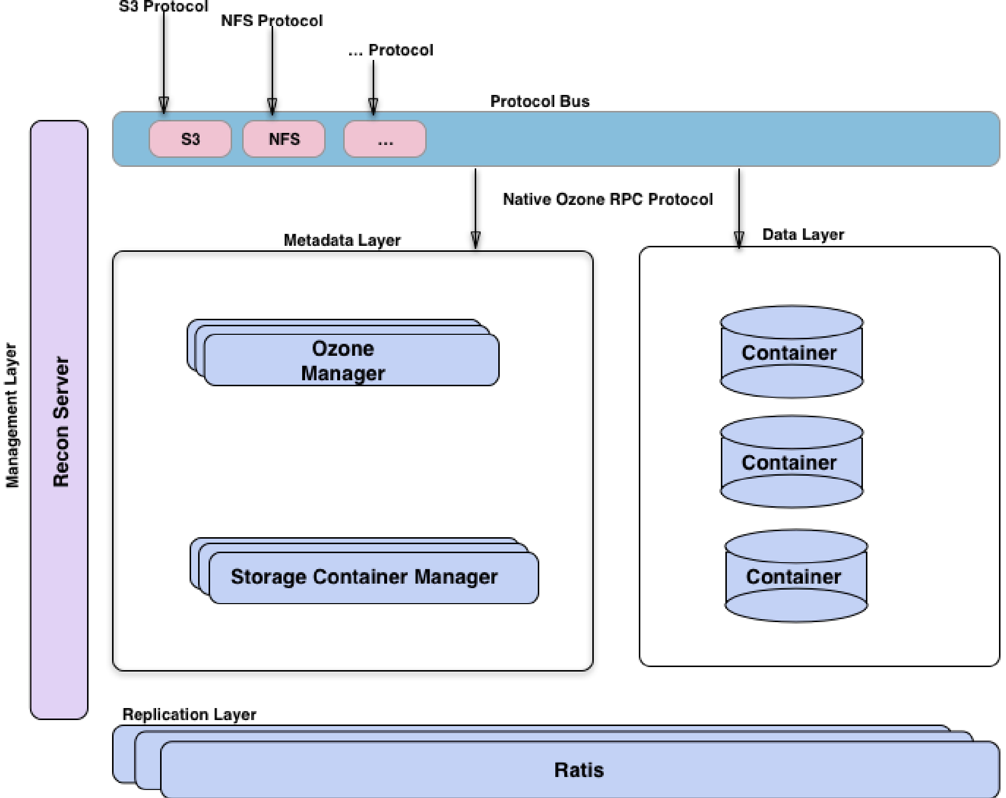


Data Storage - Ozone

The Ozone Manager is responsible for namespace management, the Storage Container Manager handles the physical and data layers, and Recon serves as the management interface for Ozone.

Different Perspectives

© Copyright Tecnisys 2026. All rights reserved.



Ozone Functional

Any distributed system can be viewed from different perspectives. One way to look at Ozone is by considering the Ozone Manager as a namespace service built on top of HDDS, a distributed block storage system.

Another way to visualize Ozone is through its functional layers. It has a metadata management layer, which consists of the Ozone Manager and the Storage Container Manager.

There is also a data storage layer, which comprises the data nodes managed by the SCM.

The replication layer, provided by Apache Ratis, is used to replicate metadata (OM and SCM) and to ensure consistency when modifying data on the data nodes.

A management server called Recon communicates with all other Ozone components, providing a unified management API and UX for Ozone.

© Copyright Tecnisys 2026. All rights reserved.

Ozone features a protocol bus that allows it to be extended via additional protocols. Currently, it supports the S3 protocol, which is built through the protocol bus. The protocol bus provides a generic framework that enables the implementation of new filesystem or object storage protocols that interact with the O3 Native protocol.

Apache Ozone Architecture

Apache Ozone's architecture is designed to manage billions of objects with scalability and resilience. It separates namespace management from block management, using the following key components:

- **Ozone Manager (OM)**:
  - Manages the namespace (volumes, buckets, and keys).
  - Uses the Ratis protocol to ensure consistency in distributed clusters.
- **Storage Container Manager (SCM)**:
  - Manages containers, which are the replication unit.
  - Coordinates data placement and replication across DataNodes.
- **DataNodes**:
  - Physically store data in containers.
  - Ensure high availability and efficient replication.
- **Recon**:
  - Monitoring and analysis tool for the Ozone cluster.
  - Collects detailed metrics for administration and diagnostics.
- **Ratis Protocol**:
  - Implements distributed consensus to ensure consistent replication.

Ozone Operations Workflow

- **Data Write**:
  - The client requests block allocation from OM to write data to DataNodes.
  - Data is written directly to DataNodes and replicated as needed.
- **Data Read**:
  - The client requests OM to locate data.

© Copyright Tecnisys 2026. All rights reserved.

    - Data is retrieved directly from DataNodes.
- **Data Replication**:
    - SCM manages replication across DataNodes to ensure fault tolerance.

Performance Expectations

Ozone employs advanced techniques to achieve high performance, such as:

- **RAFT replication for open containers**.
- **Asynchronous replication for closed containers (cold data)**.

Apache Ozone Features

Apache Ozone provides a robust set of features to meet massive data storage demands. Highlights include:

- **S3 Compatibility**:
  APIs that enable integration with cloud-based systems.
  Ideal for data migrations and hybrid architectures.

- **Hadoop Integration**:
  A direct alternative to HDFS, requiring no adaptations for frameworks like Apache Spark, Hive, and YARN.

- **High Availability**:
  Based on the Hadoop Distributed Data Store (HDDS), ensures consistent replication and fault tolerance.

- **Multimodal Support**:
  Allows storage as a file system or objects, depending on the use case.

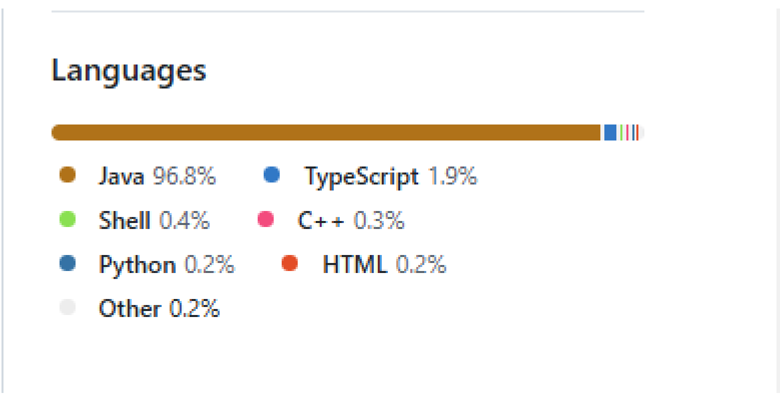
- **Horizontal Scalability**:
  Supports billions of objects, making it ideal for Big Data and Cloud-Native applications.

- **Topology Awareness**:
  Optimizes read/write pipelines based on node placement in the cluster.

Apache Ozone Project Details

© Copyright Tecnisys 2026. All rights reserved.

Apache Ozone is primarily built in Java, the foundational language for the entire Hadoop ecosystem, including HDFS, YARN, and other related components. This choice enables seamless integration with Hadoop and associated frameworks such as Spark, Hive, and MapReduce.



Apache Ozone Language

Sources

[Apache Software Foundation]

© Copyright Tecnisys 2026. All rights reserved.

Apache Ranger

Data Security

Once the user's identity has been established through authentication, security needs to ensure which actions or services that identity can access.

Data security refers to the protection measures employed to safeguard digital information against unauthorized access, corruption, or theft throughout its lifecycle, preserving its confidentiality, integrity, and availability.

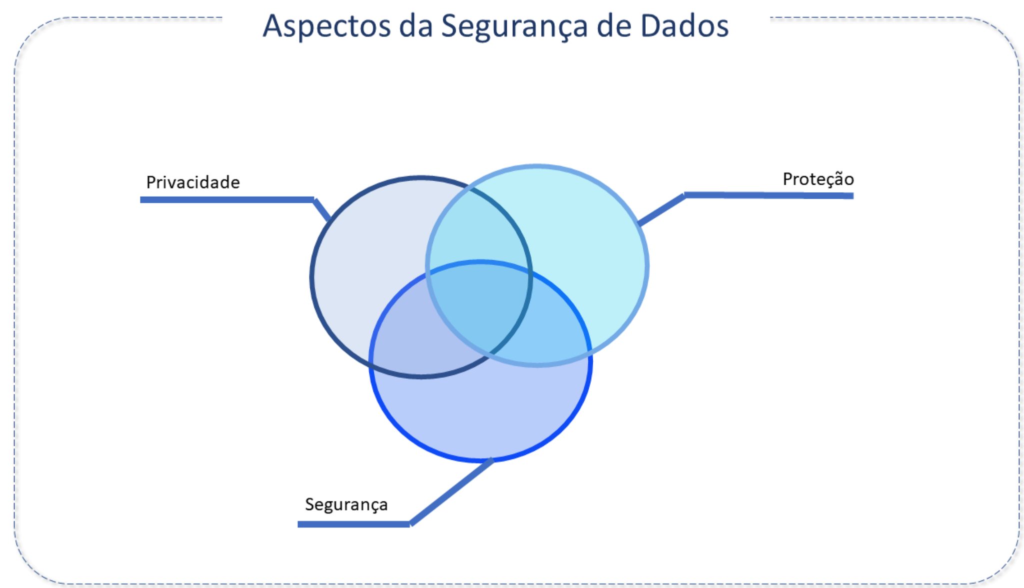
Digital transformation has profoundly changed how today's businesses operate and compete. The vast volume of data they create, handle, and store makes environments more complex, expanding the attack surface and making it more challenging to monitor and protect.

At the same time, the growing public demand for data protection initiatives, with several new privacy regulations, joins the long-standing security provisions. The value of data has never been greater. The loss of trade secrets or intellectual property can affect innovation and profitability.

Additionally, there are increasing challenges inherent to the complexity of today's distributed and hybrid environments.

Fortunately, there are available cybersecurity tools whose protection can reduce security risks when used as part of a security audit. These tools must know where the data resides, track who has access to it, and block high-risk activities.

© Copyright Tecnisys 2026. All rights reserved.



Aspects of Data Security

Apache Ranger Features

Apache Ranger is a framework designed to enable, monitor, and manage data security comprehensively on the Hadoop Platform.

With the implementation of Apache YARN, the Hadoop Platform started to support a broader data-lake architecture, allowing the execution of multiple workloads in a "multitenant" environment, which consequently demanded a more robust data security structure, with support and monitoring of data access and centralized management of security policies.

Apache Ranger was created with the following objectives:

- **Centralized security administration:**
  To enable the management of all security-related tasks in a central UI or using REST APIs.
- **Fine-grained authorization:**
  To perform actions and operations with Hadoop components or tools and management through a centralized administration tool.
- **Standardize authorization methods:**
  Across all Hadoop components.
- **Enhance support for different authorization methods:**
  Role-based access control, attribute-based access control, etc.

© Copyright Tecnisys 2026. All rights reserved.

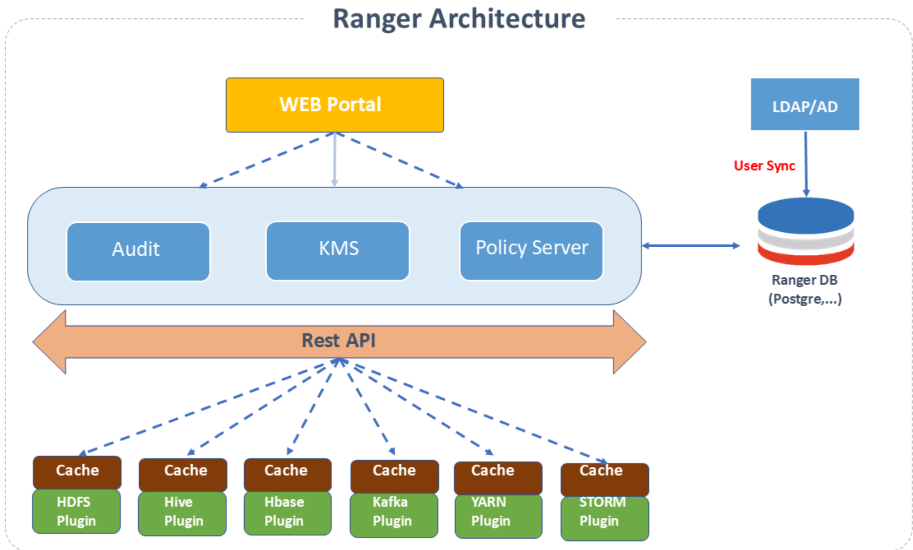
- **Centralize auditing:**
  Centralization of user access auditing and administrative actions (related to security) across all Hadoop components.

Apache Ranger Architecture

The main components of Apache Ranger are:

- **Ranger Admin Server/Portal:**
  All policies are centrally managed through a web portal that acts as a central interface for security management and allows defining repositories, creating and updating policies, managing users and groups, defining audit policies, and viewing audit activities. The portal itself has three distinct parts:
  - **Auditing:**
    Monitors user activities at the resource level and searches audit log entries based on some filters.
  - **Policy Manager:**
    Adds or modifies policies for groups or users.
    Specifies which users are admins, who can access or modify policies.
  - **KMS:**
    Used to store keys used in HDFS data encryption.
- **Ranger Plugins:**
  These are Java programs embedded in the processes of the cluster components. These plugins retrieve policies from a central server and store them locally, acting as an authorization module and evaluating user requests against the obtained security policies. Additionally, it sends this data to the audit server.

© Copyright Tecnisys 2026. All rights reserved.



Ranger Architecture

Services Supported by Apache Ranger

Currently, the following services are supported:

Component	Y/N
HDFS	YES
Hive	YES
HBase	YES
Storm	YES
Knox	YES
SolR	YES
Kafka	YES
YARN	YES

© Copyright Tecnisys 2026. All rights reserved.

Component	Y/N
Ozone	YES
Kudu	YES
Kylin	YES
NiFi	YES
NiFi Registry	YES
Sqoop *(deprecated)*	YES
Atlas	YES
ElasticSearch	YES
Presto	YES
Schema Registry	YES

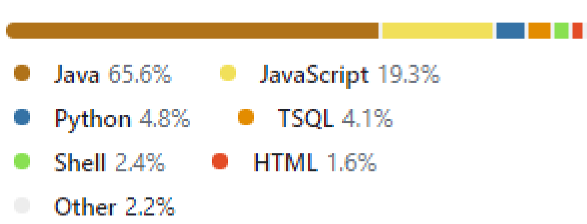
Best Practices for Apache Ranger

- For HDFS authorization, change the HDFS *umask* from 022 to 077 to prevent any new file or folder from being accessed by anyone other than the owner.
- Auditing in Apache Ranger can be controlled as a policy. Configure SSL for all enabled plugins.
- A default policy is created when Apache Ranger is installed via Ambari, for all files and directories in HDFS with auditing enabled. Ambari uses this policy to perform a smoke test via "Ambari QA" to verify HDFS services. A similar policy to enable auditing across all files and folders should be created if administrators disable this default policy.

Apache Ranger Project Details

Apache Ranger was predominantly developed in Java.

© Copyright Tecnisys 2026. All rights reserved.



Languages of Ranger

TDP Kubernetes

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **2.7.0**, distributed via Helm Chart `tdp-ranger` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources:

- Ranger Community
- Wiki
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Ranger KMS

Cryptographic Key Manager

A cryptographic key manager (KMS) is a solution or service designed to manage and protect encryption keys used in data and communications security. They play a crucial role in ensuring the confidentiality and integrity of sensitive data.

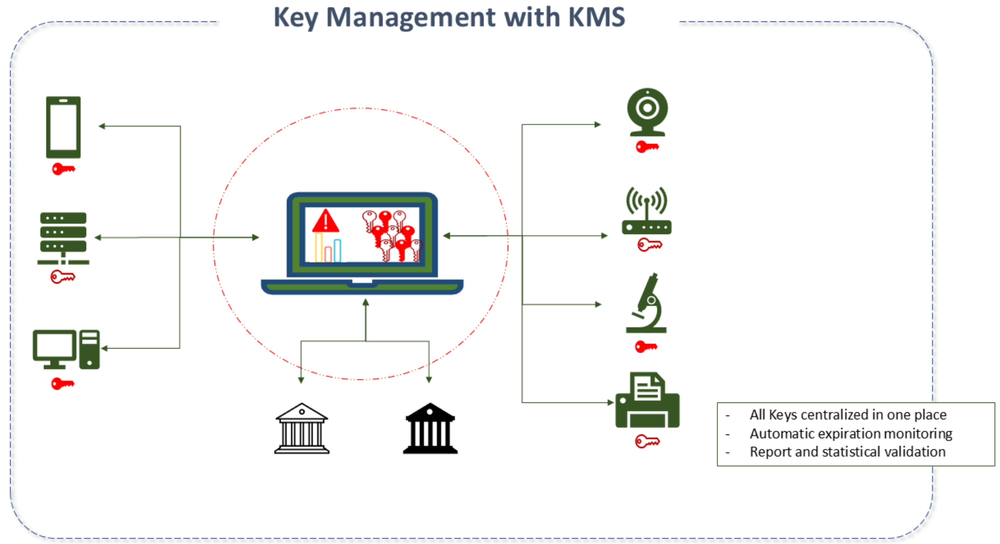
The idea of managing cryptographic keys dates back to the early days of cryptography, but the term "KMS" became more relevant with the increased use of encryption technologies in enterprise environments.

Among its relevant features, we highlight:

- **Secure key storage:** KMSs keep cryptographic keys in a highly secure environment, such as Hardware Security Modules (HSMs) or encrypted software;
- **Key generation:** They can create strong and random cryptographic keys;
- **Key distribution:** KMSs securely distribute keys to systems and applications that need them;
- **Access control:** They allow strict control over who can access and use the keys;
- **Auditing:** They record key-related activities for auditing and compliance purposes;
- **Security policies:** They allow the enforcement of security policies, such as key rotation and key expiration.

The architecture of a KMS varies, but generally consists of a central key management component, a secure key repository, and access control mechanisms. HSMs are often used to store keys securely, protecting them from unauthorized access. Security policies are applied to determine who can access and use the keys.

© Copyright Tecnisys 2026. All rights reserved.



Basic architecture of a KMS

Main Types in the Market:

- **HSMs (Hardware Security Modules):** Dedicated physical devices for storing and managing cryptographic keys in a highly secure way;
- **Cloud KMS:** Cloud-based services that offer key management as a service, such as AWS Key Management Service (KMS) from Amazon Web Services;
- **On-Premises KMS:** Software solutions that can be deployed on local infrastructure or in private cloud environments;
- **Third-Party KMS:** Many security vendors offer their own KMS solutions for use in conjunction with their products.

KMSs are essential to ensure the security of sensitive data, protecting encryption keys against internal and external threats. They play a fundamental role in enterprise environments and are widely adopted in sectors that require a high level of security, such as healthcare, finance, and government.

Apache Ranger Key Management Service (KMS)

Ranger KMS is part of the Apache Ranger ecosystem that provides key management and encryption services to protect sensitive data in Hadoop clusters and related storage systems. It is an open-source tool designed to simplify cryptographic key management and ensure compliance with security policies and regulations.

Ranger KMS is an important tool for organizations dealing with sensitive data in Hadoop clusters and distributed storage systems, helping to protect it through cryptographic key

© Copyright Tecnisys 2026. All rights reserved.

management and security policy enforcement. It plays a fundamental role in the Apache Ranger ecosystem and is widely used in Big Data environments.

Ranger KMS Features

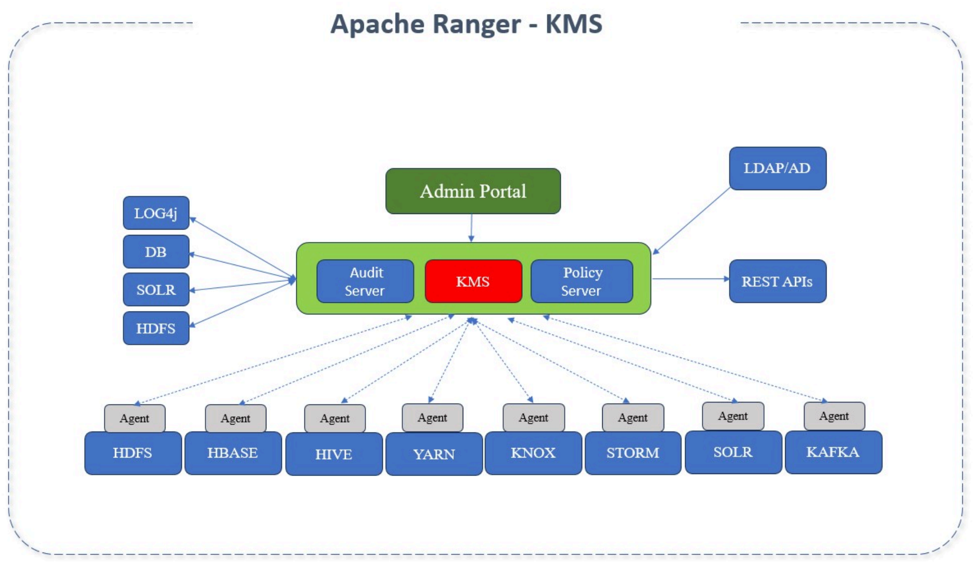
Among its main features, we highlight:

- **Secure management** of cryptographic keys;
- **Integration with Hadoop systems** and other storage systems;
- **Policy-based access control** for keys;
- **Auditing and logging of key-related activities**;
- **Support** for various encryption algorithms;
- **Integration with Apache Ranger** for security policy enforcement;
- **Extensibility** to support additional storage systems.

Ranger KMS Architecture

The architecture of Ranger KMS includes the following main components:

- **Key Management Service:** Central service that manages cryptographic keys;
- **Key Database:** Securely stores cryptographic keys;
- **Policy Engine:** Applies security policies to control access to keys;
- **Audit Log:** Records all key-related activities for audit purposes;
- **Ranger Admin:** The Apache Ranger administration interface for configuring security policies.



Ranger KMS Architecture

© Copyright Tecnisys 2026. All rights reserved.

Ranger KMS Resources

- **Advanced Security:** Strengthens data security by encrypting sensitive information.
- **Facilitates Compliance:** Simplifies compliance with data security regulations.
- **Fine-grained Control:** Provides more detailed control over who can access and manage cryptographic keys.
- **Logging and Monitoring:** Facilitates logging and monitoring of activities for auditing and compliance purposes.
- **Easy Integration:** Can be seamlessly integrated with Hadoop systems and other Big Data technologies.

Best Practices for Ranger KMS

When using Ranger KMS (Key Management Service) or any other cryptographic key management system, it is essential to follow security best practices to ensure adequate protection of the keys and the sensitive data they protect. Here are some best practices when using Ranger KMS:

- **Clear Security Policies:** Define clear security policies that describe who has access to keys and under what conditions.
- **Granular Access Control:** Ensure that only authorized people and systems can access cryptographic keys.
- **Regular Key Rotation:** Establish policies for regular key rotation, ensuring that keys do not become obsolete and vulnerable.
- **Activity Auditing:** Enable activity auditing in Ranger KMS to record and monitor all key-related operations.
- **Physical and Logical Protection:** Maintain a high level of physical and logical protection around Ranger KMS components, including Hardware Security Modules (HSMs) if used.
- **Updates and Patches:** Keep Ranger KMS updated with the latest security updates and patches to fix known vulnerabilities.
- **Segregation of Duties:** Implement segregation of duties to ensure that key management responsibilities are distributed among multiple people or teams.
- **Continuous Monitoring:** Establish a continuous monitoring system to detect suspicious or unauthorized key-related activities.
- **Communication Encryption:** Ensure that all communications between Ranger KMS components are encrypted to protect data in transit.
- **Training and Awareness:** Provide adequate training to Ranger KMS administrators and users to ensure correct and secure use of the system.

© Copyright Tecnisys 2026. All rights reserved.

- **Backup and Recovery:** Implement backup and recovery policies to ensure that keys are not lost in case of system failure.
- **Disaster Recovery Testing:** Conduct regular disaster recovery tests to ensure you can restore keys in case of a major failure.
- **Compliance with Regulations:** Ensure that your Ranger KMS usage practices comply with relevant data security regulations for your organization and industry.
- **Periodic Policy Review:** Conduct periodic reviews of security policies and update them as needed to meet the ever-evolving security needs.
- **Least Privilege Access:** Follow the principle of "least privilege access," granting access only to the keys and cryptographic information that users or systems need to perform their functions.
- **Incident Management:** Establish incident management procedures to effectively respond to any security breach or suspicious event.
- **Vulnerability Assessment:** Conduct regular vulnerability assessments to identify and mitigate potential security threats.

Remember that security best practices should be tailored to your organization's specific needs and applicable regulations. It is essential to stay up-to-date on security best practices and adjust your policies and procedures according to changes in the threat landscape.

Ranger KMS Project Details

Ranger KMS (Key Management Service) is primarily written in the Java programming language. The Apache ecosystem, of which Apache Ranger is a part, is largely based on Java, and this extends to the Ranger KMS source code. The use of Java allows Ranger KMS to run on various platforms and operating systems compatible with Java, making it highly portable and scalable.

Sources:

- Apache Community

© Copyright Tecnisys 2026. All rights reserved.

Apache Solr

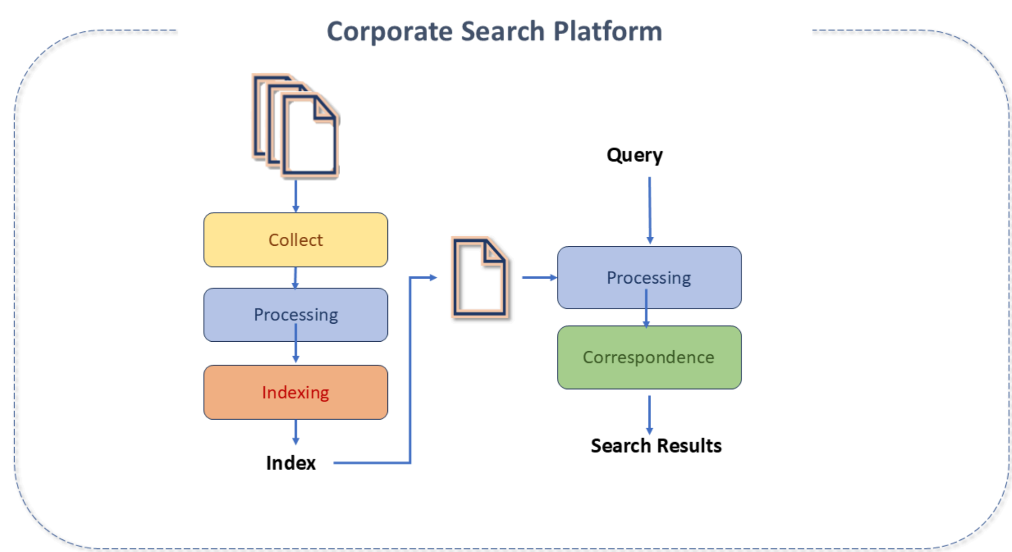
Corporate Search Platform

Enterprise Search Platforms have the function of creating a secure, powerful and easy-to-use search function.

When applied to business contexts, they generally integrate with business intelligence and data management solutions, and are used to "cleanse" and structure data, making information easier to locate, and can extract information from different sources, such as CRM, ERP, etc.

To qualify as a corporate search platform, the product must:

- Handle information from different sources, data types and formats.
- Index or archive data.
- Provide intelligent search options.
- Offer an interface for searching and retrieving data.
- Allow users to refine their searches using advanced filters.
- Set permissions for access to information.



Corporate Search Platform

© Copyright Tecnisys 2026. All rights reserved.

Apache Solr features

Apache Solr is an open source enterprise search platform, built on top of Apache Lucene, designed for document retrieval.

Apache Solr was created in 2004 by Yonik Seeley, at CNET Networks, as an internal project to add search capabilities to the company's website.

- In January 2006, its source code was donated to the Apache Software Foundation and, by 2007, it was already considered a top-level autonomous project (TLP), having grown steadily with accumulated resources, attracting users, collaborators and *commiters*.
- It was released by Apache in 2008, in version 1.3, and in 2010 it was merged with Lucene (its versioning scheme was changed in 2011 to match Lucene's).
- In 2020, *Bloomberg* donated the Solr operator to the Lucene/Solr project, which made it possible to deploy and run Solr on Kubernetes.
- In 2021 Solr was established as a separate Apache project (TLP), independent of Lucene. Its first independent version was 9.0.

Solr is widely used for enterprise research and analysis use cases. It is developed openly and collaboratively by the Apache Solr Project at the Apache Software Foundation. It has a very active development community with regular releases.

Its main features include:

- **Advanced Full-Text Search Features**. Enabling advanced search features, including phrase matching, use of *wildcards* to make searches more flexible, as well as supporting joins and groupings on any type of data.

- **Optimized for high-volume traffic**. Proven on extremely large scales around the world.

- **Open interfaces:** Based on XML, JSON and HTTP standards.

- **Comprehensive administration interfaces:** Featuring an integrated responsive administrative user interface to make it easier to control your instances.

- **Easy monitoring:** Publishing metric data loads via JMX.

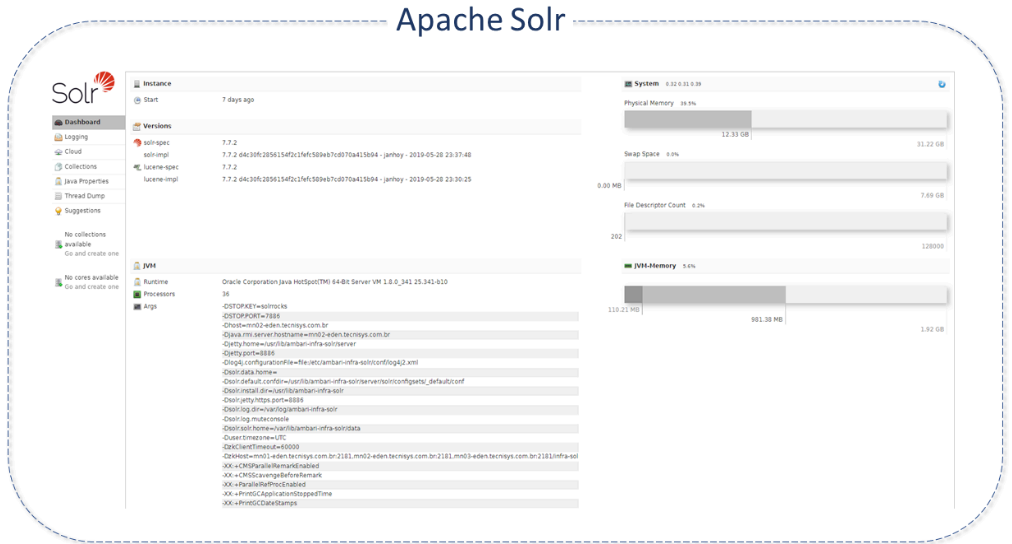
- **Highly scalable and fault-tolerant:**

© Copyright Tecnisys 2026. All rights reserved.

  - Built on Apache Zookeeper, it makes it easy to scale up or down.
  - Offers replication, distribution, rebalancing and fault tolerance.
- **Flexible and adaptable with simple configuration:** Flexible schema configurations, allowing almost any type of data to be stored in Solr.

- **Real-time _near indexing:** Allowing changes to be seen at any time.

- **Extensible plugin architecture.** Publishes many extension points that make it easy to incorporate index and query time plugins, allowing your code to be changed at will (as it is an open-source license).



Apache Solr interface

Apache Solr Architecture

Apache Solr is designed for scalability and fault tolerance. It runs as a standalone text search server.

Apache Solr runs on top of the Lucene software library which is, in fact, the engine that powers it. The Solr core is an **instance of the Lucene index**, where all the configuration files needed to use it are located. The Lucene Java Search Library is used for indexing and full-text searching and has HTTP/XML and JSON REST-like APIs that make Solr **usable in the most popular programming languages**.

Its external configuration allows it to adapt to many types of applications without Java coding and its plug-in architecture supports more advanced customizations.

© Copyright Tecnisys 2026. All rights reserved.

SOLR queries are simple HTTP request URLs and the responses are a structured document in JSON, XML, CSV or other format. The documents (*indexing*) are placed in it via JSON, XML, CSV or binary over HTTP. It is queried via HTTP GET and receives JSON, XML, CSV or binary results.

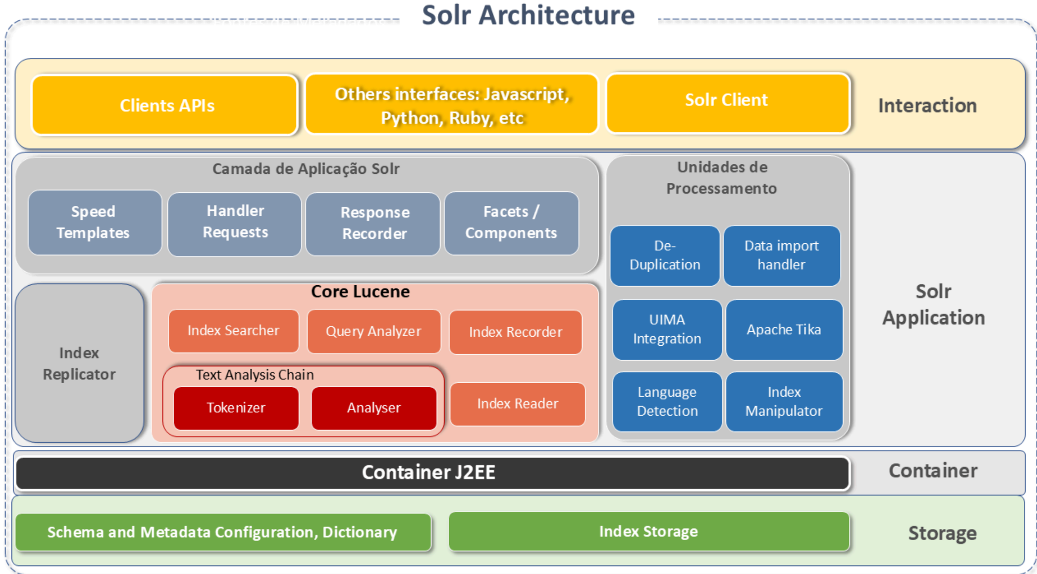
Solr runs on a **non-slave architecture**, as **every node is its own master**. The nodes use Zookeeper to learn about the state of the *Cluster* and each node (JVM) can host several cores (where Lucene runs).

Queries are configured and managed in the *schema.xml* and *solrconfig.xml* files.

The main components of the Apache Solr architecture are:

- RequestHandler: Requests received in Apache Solr are processed by the *Request Handler*. The request can be a query or an index update. Based on the request, the *request handler* is selected. The handler is usually mapped to an *end-point* URI through which the request will be fulfilled.
- Search Resource: Provides the search facility to Apache Solr. This can be *spell checking*, querying, *faceting*, *hit highlighting*, etc.
- Query Parsers. They analyze queries and check for syntax errors. After analysis, they translate the queries into a format understood by Lucene. Solr supports several query parsers.
- Response Writer: This is the component that generates the formatted output for the user's query. The supported formats are XML, JSON, CSV, etc. There are different *response writers* for each type of response.
- Parser / tokenizer: Lucene recognizes data in the form of *tokens*. Apache Solr analyzes the content, divides it into *tokens* and passes these *tokens* on to Lucene. An Analyzer in Apache Solr examines the text of fields and generates a "token stream". A tokenizer divides the *token stream* prepared for the parser into *tokens*.
- Update Request Processors: Every *update* request will execute through a set of *plugins* (signing, logging, indexing), collectively known as an "update request processor". This processor is responsible for modifications such as "field removal, field addition, etc."

© Copyright Tecnisys 2026. All rights reserved.



Apache Solr architecture

Apache Solr terminology

- Cluster Solr nodes operating in coordination with each other via Zookeeper and managed as a unit. A *Cluster* can contain several collections.

- Collections:** One or more documents grouped together in a single logical index using a single configuration and Schema. In the cloud, the collection can be divided into multiple logical fragments, which in turn can be distributed over several nodes or on a single node.

- Node In the Solr cloud, each unique Solr instance is considered a node.

Commit: Enables permanent changes to the index.

- Core:** An individual Solr instance (representing a logical index). Several cores can run on a single node.

- Shard(fragment): This is a logical partition of the collection that holds a subset of its documents, so that each document in a collection is contained in exactly one Shard. When a document is sent to a node for indexing, the system first determines which "shard" the document belongs to, and then which node hosts its Leader, forwarding the document to this leader for indexing, which in turn forwards it to all the other replicas.

© Copyright Tecnisys 2026. All rights reserved.

- Replication: When the index is too large for a single machine and there is a volume of queries that single shards can't keep up with, it's time to replicate each shard. In the Solr core, the replica is a copy of the *shard* that is executed on a node.

Leader: This is the *shard* replica that distributes the *Solr Cloud* requests to the other replicas.

Apache Solr features

Solr offers rich and flexible search capabilities. Its main features include:

Text search: Powered by Lucene, Solr allows matching features such as phrases, *wildcards*, *joins*, clusters and more on any type of data.

- Highlighting: Allows document fragments that match the query to be included in your answer.

- Faceting: Organizing search results into categories based on indexed terms.

- Real-time indexing:: Solr allows you to create an index with many different fields or input types. A Solr index can accept data from many different sources, such as XML, CSV or data extracted from tables in databases and files in common formats such as Word or PDF.

- Result Collapsing and Expanding: Groups documents (collapsing the result set) according to parameters and provides access to the document in the "collapsed" grouping for use in a "display" or application.

- SpellChecking: Designed to provide online query suggestions based on other similar terms.

- Database integration

- NoSQL features:** The *realtime get unique-key* feature allows retrieval of the latest version of any documents without the cost of reopening a searcher. This is especially useful when using Solr as a NoSQL data store and not just a search index.

  Real Time Get depends on the *update log* feature, which is enabled by default and can be configured in `solrconfig.xml`.

© Copyright Tecnisys 2026. All rights reserved.

Some of the most famous sites on the Internet use Solr, such as Macy, Ebay, Zappo.

Best Practices for Apache Solr

- Indexed fields:** The use of cloud storage can be greatly optimized by controlling the number of indexed fields.
  Indexes represent a critical factor between quality, performance and cost, and too many can degrade performance and increase costs.
  In the field definition, try not to mark fields as *indexed=true* if they are not used in a query.
  The number of indexed fields increases:
  - Memory usage during indexing.
  - Segment merging time.
  - Optimization time.
  - The size of the index.
- Storing fields: Retrieving stored fields from the query result can be costly. This cost is affected by the number of bytes stored per document. The higher the byte count, the more sparse the distribution of documents on disk - which requires more I/O bandwidth to retrieve the desired fields.

  These costs increase in cloud implementations. Obviously, their use must weigh up quality, performance and cost.

- Use of the Solr Cache: Solr caches various types of information to ensure that similar queries are not repeated unnecessarily.
  The Document Cache helps improve the time it takes to answer requests.
  There are three main types of cache:
  - *Query cache*: stores document IDs returned by queries.
  - *Filter cache*: stores filters created by Solr in response to filters added to queries.
  - *Document cache*: stores the fields of the document requested when query results are displayed.
- Autowarming: Enabling *autowarming* can significantly increase the performance of a search for any of the three cache types.
  By increasing *autowarmCount*, Solr will pre-fill or *autowarm* the cache with the cache objects created by the search result.

© Copyright Tecnisys 2026. All rights reserved.

This setting indicates the number of items in *cache* that will be copied to the new "searcher".

- Streaming Expressions: Solr offers a simple but powerful *streaming* processing language for the Solr Cloud.
  Streaming expressions are a set of functions that can be combined to perform many parallel computing tasks.

- JVM Memory Heap: The *heap* is an area of memory in which application objects, created from classes (with reference semantics), are stored. Objects that are referenced somewhere are allocated there.
  This area is managed by the *garbage collector* and occupies space as needed. At some point in the future, the GC will release the spaces when the data in a given portion is no longer needed.
  In the case of JVM (Java Virtual Machine) memory, much of the allocation is already reserved in advance by the GC, which tries to manage the memory in the best possible way.
  The *heap* settings of JVM memory directly affect the use of system resources.
  In a cloud environment, resource usage and costs can increase rapidly, which can directly affect performance and the success or failure of a search implementation.

  - Adding enough space Several tools, usually included in standard Java installations, show the minimum amount of memory consumed by the JVM when running Solr (including *jconsole*).
    Without enough memory for *heap*, the JVM can increase resource consumption due to frequent execution of the *garbage collection* process.
    An additional 25% to 50% of the minimum memory required to operate at maximum performance is recommended.

  O.S. memory:
  Care must be taken when allocating *heap JVM* memory, as Solr makes extensive use of the *org.apache.lucene.store.MMapDirectory* component.
  This component takes advantage of the Operating System (OS) that is used by the *MMapDirectory* component.
  In this way, the more *heap JVM* memory is allocated, the less memory is available for the Operating System, which also impairs search performance.

It is necessary to find a balance.

The ideal size of the JVM memory heap:
A range between 8 and 16 Gb is adequate. It is recommended to start with the smallest

© Copyright Tecnisys 2026. All rights reserved.

value and run tests, monitoring memory usage and gradually adjusting the size until the ideal size is reached.

- Garbage Collector:** This is a term used for the process in which memory freed in a Java program is returned to the memory pool for reuse.
  The collector used before Java version 9 was ParallelGCf. The G1GC Collector operates in a simultaneous multi-threaded manner and addresses latency problems that exist in ParallelGC.
  Its use is therefore recommended.
  In addition, the Solr control script comes with a set of pre-configured Java garbage collection configurations that work well for many different workloads. But these settings may not work so well for a particular use.
  In this case, the GC settings can be changed, which should also be done with the GC_TUNE variable in /etc/default/solr.in.sh.

Recommended fine-tuning of the production configuration

- Memory and GC settings
  By default, *script bin/solr* sets the maximum Java heap size to 512M (-Xmx512m), which is a reasonable number to start working with Solr.
  For production, it is appropriate to adjust the maximum size based on the memory requirements of the search applications. Values between 10 and 20 Gb are common. To change these settings, use the *SOLR_JAVA_MEM* variable:

Terminal input

```
solr_java_mem="-Xms10g -Xmx10g"
```

- Shutdown_ due to lack of memory: The *script* bin/solr registers *script bin/oom_solr.sh* to be called by the JVM if an *OutOfMemory* error occurs, which will issue a *kill -9* to the Solr process.
  This behavior is recommended when running in SolrCloud mode so that Zookeeper is immediately notified that a node is showing an unrecoverable error.
  It is recommended to inspect the contents of *script* /opt/solr/bin/oom_solr.sh to familiarize yourself with the actions that *script* will perform if called by the JVM.

- SolrCloud mode:
  To run Solr in SolrCloud mode you need to set the ZK_HOST in the include file to

© Copyright Tecnisys 2026. All rights reserved.

point to the Zookeeper assembly.
Running the embedded Zookeeper is not supported in production environments. If a Zookeeper set is hosted, for example, on three hosts on the default client port 2181(zk1, zk2, zk3), you will need to set:

When the *ZK_HOST* variable is set, Solr will start in cloud mode.

- Zookeeper:
  When using a Zookeeper instance shared by other systems, it is recommended to isolate the *SolrCloud znode* "tree" using Zookeeper's *chroot* support.
  To ensure that all *znodes* created by *SolrCloud* are stored in /solr, for example, use:

Before using a *chroot* for the first time, you need to create the root path (znode) in Zookeeper using the Solr control script. The *mkroot* command can be used:

- Hostname Solr:
  Use the SOLR_HOST variable in the include file to set the hostname of the Solr server.

© Copyright Tecnisys 2026. All rights reserved.

This is especially recommended when running in SolrCloud mode, as it determines the node's address when it registers with Zookeeper.

- Replacing settings in solrconfig.xml:
  Solr allows configuration properties to be overridden using Java system properties passed at startup via the *-Dproperty=value* syntax.

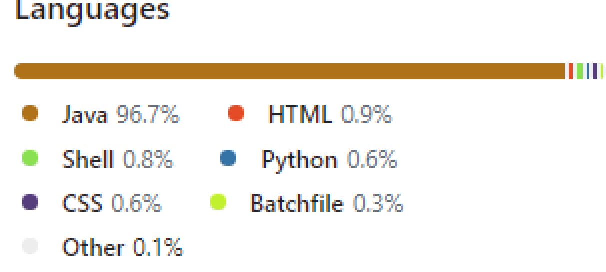
- Running multiple Solr nodes per host: *bin/solr* is capable of running multiple instances on one machine, but for a typical installation, this configuration is not recommended.
  Extra CPU and memory resources will be required for each additional instance.

WARNING

This rule does not apply when discussing extreme scalability. Running several Solr nodes on one host is a good alternative when you want to reduce the need for extremely large heaps. For more details see here.

Apache Solr language

Solr is written in Java.



Solr languages

Sources

- whismy Apache Solr
- Apache Wiki
- Apache.Solr.org
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Spark

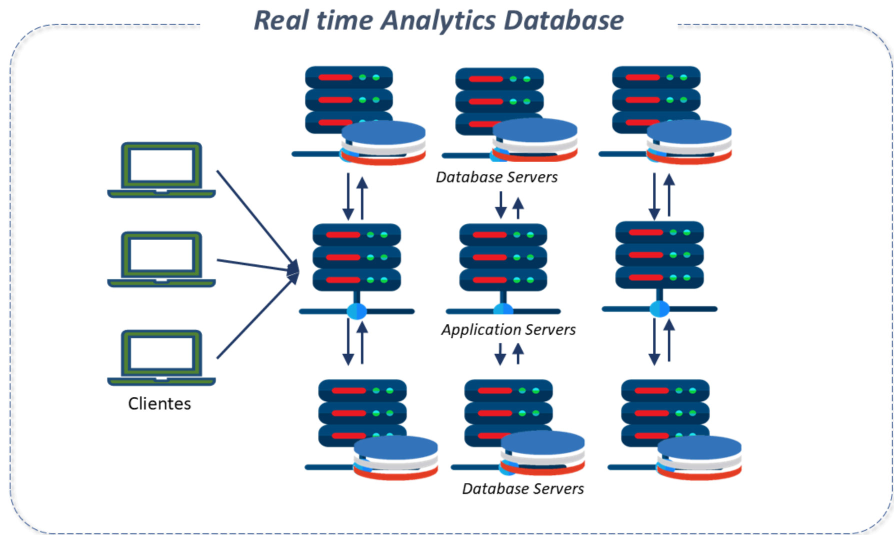
Distributed Computing Platform

Distributed Computing

With the evolution of computer networks, a new computational paradigm emerged and became extremely powerful: the possibility of distributing processing across different computers.

Distributed computing allows the partitioning and specialization of computational tasks, according to the nature and function of each computer.

This paradigm emerged to solve a major problem in computing, which is the need for computers with sufficient processing power to analyze the large volume of data available today.



Distributed Computing

Apache Spark Features

Apache Spark is a unified analytics engine for large-scale data processing in distributed computing.

© Copyright Tecnisys 2026. All rights reserved.

Created in 2009 at the University of California by Berkeley's AMPLab, Spark quickly gained a large community, leading to its adoption by the Apache Software Foundation in 2013.

Developers from over 300 companies helped implement it, and a vast community with more than 1,200 developers from hundreds of organizations continue to contribute to its ongoing refinement.

It is used by organizations in a range of sectors and has one of the largest existing developer communities.

The power of Spark comes from its in-memory processing.

It uses a distributed set of nodes with a lot of memory and compact data encoding, along with an optimized query planner to minimize execution time and memory demand.

By performing in-memory calculations, it can process up to 100 times faster than disk-processing frameworks.

It is ideal for processing large volumes of data in Analytics, training models for machine learning and AI.

Additionally, Spark runs a stack of native machine learning libraries and graph processing and SQL-like data structures, enabling exceptional performance.

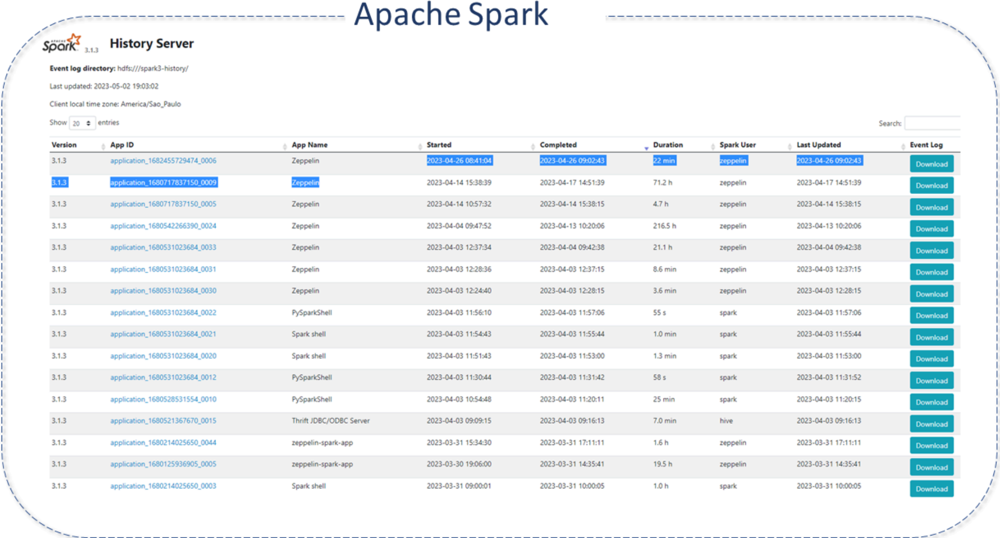
It features over 80 high-level operators, making it easy to create parallel applications.

Spark shares some similarities with Hadoop. Both are open-source frameworks for processing analytical data, live at the Apache Software Foundation, contain machine learning libraries, and can be programmed in several different languages.

However, Spark extends the number of possible calculations with Hadoop, enhancing Hadoop's native data processing component, MapReduce.

Spark uses the Hadoop Distributed File System (HDFS) infrastructure but improves its functionalities and provides additional tools, such as implementing applications in a Hadoop Cluster (with SIMR - Spark Inside MapReduce) or YARN.

© Copyright Tecnisys 2026. All rights reserved.



Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.1.3	application_1682455729474_0006	Zeppelin	2023-04-26 08:41:04	2023-04-26 09:02:43	22 min	zeppelin	2023-04-26 09:02:43	Download
3.1.3	application_1680717837150_0009	Zeppelin	2023-04-14 15:38:39	2023-04-17 14:51:39	71.2 h	zeppelin	2023-04-17 14:51:39	Download
3.1.3	application_1680717837150_0005	Zeppelin	2023-04-14 10:57:32	2023-04-14 15:38:15	4.7 h	zeppelin	2023-04-14 15:38:15	Download
3.1.3	application_1680542266390_0024	Zeppelin	2023-04-04 09:47:52	2023-04-13 10:20:06	216.5 h	zeppelin	2023-04-13 10:20:06	Download
3.1.3	application_1680531023684_0033	Zeppelin	2023-04-03 12:37:34	2023-04-04 09:42:38	21.1 h	zeppelin	2023-04-04 09:42:38	Download
3.1.3	application_1680531023684_0031	Zeppelin	2023-04-03 12:28:36	2023-04-03 12:37:15	8.6 min	zeppelin	2023-04-03 12:37:15	Download
3.1.3	application_1680531023684_0030	Zeppelin	2023-04-03 12:24:40	2023-04-03 12:28:15	3.6 min	zeppelin	2023-04-03 12:28:15	Download
3.1.3	application_1680531023684_0022	PySparkShell	2023-04-03 11:56:10	2023-04-03 11:57:06	55 s	spark	2023-04-03 11:57:06	Download
3.1.3	application_1680531023684_0021	Spark shell	2023-04-03 11:54:43	2023-04-03 11:55:44	1.0 min	spark	2023-04-03 11:55:44	Download
3.1.3	application_1680531023684_0020	Spark shell	2023-04-03 11:51:43	2023-04-03 11:53:00	1.3 min	spark	2023-04-03 11:53:00	Download
3.1.3	application_1680531023684_0012	PySparkShell	2023-04-03 11:30:44	2023-04-03 11:31:42	58 s	spark	2023-04-03 11:31:52	Download
3.1.3	application_1680528531554_0010	PySparkShell	2023-04-03 10:54:48	2023-04-03 11:20:11	25 min	spark	2023-04-03 11:20:15	Download
3.1.3	application_1680521367670_0015	Thrift JDBC/ODBC Server	2023-04-03 09:09:15	2023-04-03 09:16:13	7.0 min	hive	2023-04-03 09:16:13	Download
3.1.3	application_1680214025650_0044	zeppelin-spark-app	2023-03-31 15:34:30	2023-03-31 17:11:11	1.6 h	zeppelin	2023-03-31 17:11:11	Download
3.1.3	application_1680125936905_0005	zeppelin-spark-app	2023-03-30 19:06:00	2023-03-31 14:35:41	19.5 h	zeppelin	2023-03-31 14:35:41	Download
3.1.3	application_1680214025650_0003	Spark shell	2023-03-31 09:00:01	2023-03-31 10:00:05	1.0 h	spark	2023-03-31 10:00:05	Download

Apache Spark Interface

Apache Spark Architecture

Apache Spark is a distributed processing engine that operates on the coordinator/worker principle.

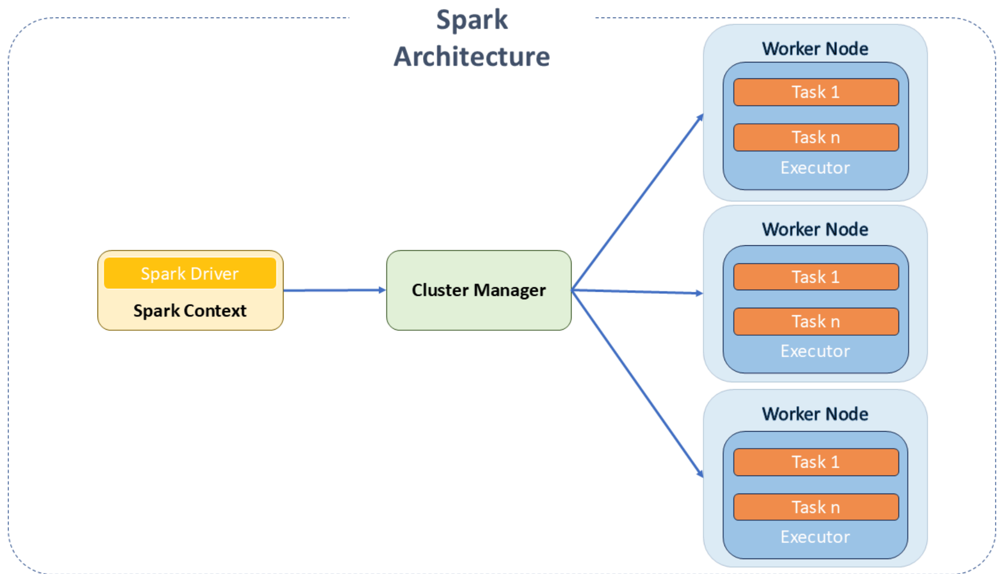
Its architecture consists of the following main components:

- **Spark Driver:** It is the master of the Spark architecture. It is the main application that manages the creation and execution of the processing defined by the programmer.

- Cluster Manager: An optional component only necessary if Spark is run in a distributed manner. It is responsible for managing the machines that will be used as workers.

- **Spark Workers:** These are the machines that actually execute the tasks sent by the Driver Program. If Spark is run locally, the machine can act as both Driver and Worker.

© Copyright Tecnisys 2026. All rights reserved.



Spark Architecture

Fundamental Components of the Spark Programming Model

- Resilient Distributed Datasets (RDD): The main object of the Spark programming model. It is in these objects that the data is processed. They store the data in memory to perform various operations such as loading, transforming, and actions (calculations, writing, filtering, joining, and map-reduce) on the data. They abstract a set of distributed objects in the Cluster, usually executed in main memory. They can be stored in traditional file systems, HDFS, and some NoSQL databases like HBase.

- Operations: Represent transformations (grouping, filtering, mapping data) or actions (counting and persistence) performed on an RDD. A Spark program is typically defined as a sequence of transformations or actions performed on a dataset.

- Spark Context: The context is the object that connects Spark to the program being developed. It can be accessed as a variable in a program.

Apache Spark Libraries

In addition to APIs, there are libraries that make up its ecosystem and provide additional capabilities:

- Spark Streaming: Can be used to process streaming data in real-time based on microbatch computing. For this, it uses DStream, which is basically a series of RDDs to process data in real-time. It is scalable, has high throughput, fault-tolerant, and

© Copyright Tecnisys 2026. All rights reserved.

supports batch or streaming workloads. Spark Streaming allows reading/writing from/to Kafka topics in text, csv, avro, and json formats.

- Spark SQL: Provides the ability to expose Spark datasets through a JDBC API. This allows executing SQL-style queries on these datasets, making use of traditional BI and visualization tools. Additionally, it allows the use of ETL to extract data in different formats (Json, Parquet, or Database), transform them and expose them for ad-hoc queries.

- Spark MLlib: Spark's machine learning library, consisting of learning algorithms, including classification, regression, clustering, collaborative filtering, and dimensionality reduction.

- Spark GraphX: A new API for Spark for graphs and parallel computation. Simply put, it extends Spark RDDs for graphs. To support graph computation, it exposes a set of fundamental operators (subgraphs and adjacent vertices), as well as an optimized variant of Pregel. Additionally, it includes a growing collection of algorithms to simplify graph analysis tasks.

- Shared Variables: Spark offers two types of shared variables to make it more efficient when running on Clusters:

    - **Broadcast:** Read-only variables that are cached on all nodes of the Cluster for access or use by tasks. Instead of sending the data with each task, Spark distributes the broadcast variables to the machine using efficient broadcast algorithms to reduce communication costs.

    - **Accumulator:** Shared variables added only through an associative and commutative operation, used to perform counters (similar to MapReduce counters) or sum operations.

Apache Spark Workflow

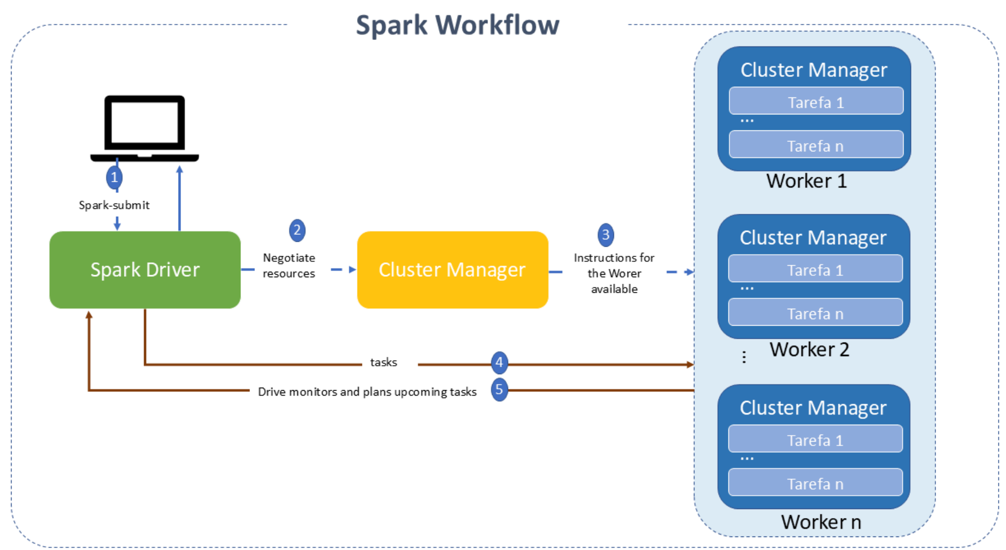
Its lifecycle involves several intermediate steps, each responsible for handling specific responsibilities.

- The process begins with the job submission by the client, using the *spark-submit* option.
- The *main* class, specified during job submission, is called, and the Spark driver program is started on the master node, responsible for managing the application's

© Copyright Tecnisys 2026. All rights reserved.

lifecycle.

- The driver program requests resources from the Cluster manager to start the Executors based on the application's configuration.
- The Cluster manager activates the executor on the worker node on behalf of the Spark driver, which now takes ownership of the application's lifecycle.
- The Spark driver creates a DAG based on the RDD. The task is then divided into stages. The Spark driver sends tasks to the executor, which executes them.
- The executor sends a task completion request to the driver through the Cluster manager. After all tasks are completed on all executors, the driver sends a completion status to the Cluster manager.



Spark Workflow

Other Features of Apache Spark

- Spark can access **variable data sources** and run on **various platforms**, including Hadoop.
- Provides **high-level functional APIs** in Java, Scala, Python, and R for:
  - **large-scale data manipulation**
  - **in-memory** data caching
  - **reusing datasets**
- Supports **various formats and sets of APIs** to handle any type of data in distributed mode.
- Offers an **optimized engine with support for general execution graphs**.
- Utilizes the concept of **direct acyclic graph (DAG)**, through which it is possible to develop pipelines composed of several complex stages.

© Copyright Tecnisys 2026. All rights reserved.

- The ability to store data *in-memory* and *near real-time* processing makes Spark **faster than the MapReduce framework** and provides an **advantage for iterative use cases** where the same dataset is used multiple times in different executions.

Best Practices for Apache Spark

- **Use Dataframe/Dataset over RDD**: RDD serializes and deserializes whenever it distributes data between Clusters. These operations are very costly. On the other hand, Dataframe stores data as binaries, using off-heap storage, without the need for serialization and deserialization of data in distribution to Clusters, making it a great advantage over RDD.

- **Use Coalesce to Reduce Number of Partitions**: Whenever it is necessary to reduce the number of partitions, use coalesce, as it makes the minimum data movement across the partition. On the other hand, repartition recreates the entire partition, making data movement very high. To increase the number of partitions, we have to use repartition.

- **Use Serialized Data Formats**: Generally, whether it's a streaming or batch job, Spark writes the calculated results to an output file, and another Spark job consumes it, performs some calculations, and writes it again to an output file. In this scenario, using a serialized file format, such as Parquet, gives us a significant advantage over CSV and JSON formats.

- **Avoid User-Defined Functions**: Use Spark's pre-built functions whenever possible. UDFs (User Defined Functions) are a black box for Spark, preventing it from applying optimizations. Thus, we lose all optimization features offered by Spark's Dataframe/Dataset.

- **Cache Memory Data**: Whenever we perform a sequence of Dataframe transformations and need to repeatedly use an intermediate Dataframe for additional calculations, Spark provides a feature to store a specific DF in memory in the form of a cache.

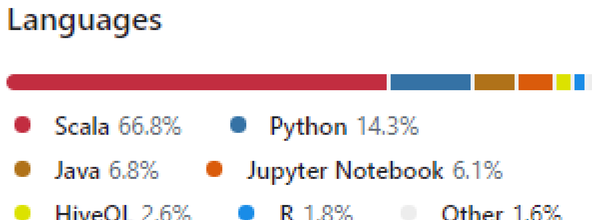
Apache Spark Project Details

Spark is written in the Scala language and runs on a Java virtual machine. It currently supports the following languages for application development:

- Scala

© Copyright Tecnisys 2026. All rights reserved.

- Java
- Python
- Clojure
- R



Spark Languages

TDP Kubernetes

AVAILABLE IN TDP KUBERNETES

This component is also available in the **TDP Kubernetes** edition since version 3.0.

The current version is **4.0.0**, distributed via Helm Chart `tdp-spark` v3.0.1.

For configuration details, see the TDP Kubernetes documentation.

Sources

- Apache.org
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Sqoop

Data Transfer

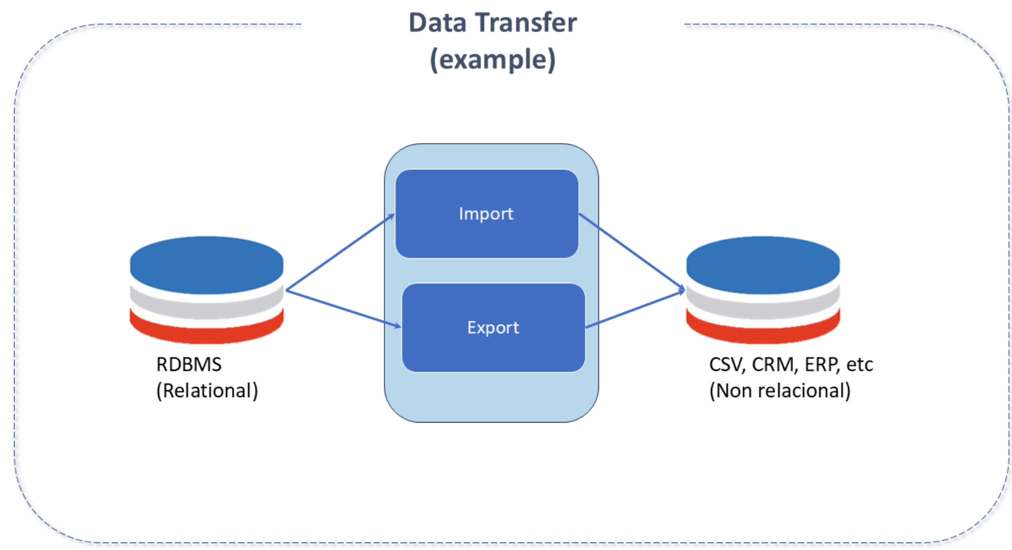
Este componente foi aposentado pela Apache Software Foundation e não recebe mais atualizações ou suporte.

Este componente será removido a partir do TDP 3.1.0.

Before being processed by a machine learning system, data needs to be imported. Likewise, they need to be exported to other applications before being used externally.

A data transfer engine enables the movement "to" or "from" different storage devices. The actual import work is done by processing mechanisms, which run the import jobs and then persist the imported data into the storage device.

Unlike other data processing systems, where the input data conforms to a schema and is almost always structured, the source data for a machine learning system can include a mix of sources and formats.



Data Transfer

Features of Apache Sqoop

© Copyright Tecnisys 2026. All rights reserved.

Apache Sqoop is a **command-line tool** designed to facilitate **bulk data transfer** between Apache Hadoop and structured datastores such as relational databases (RDBMS).

The integration of these environments is Sqoop's role, and its main purpose is to **import/export data** between the relational environment and Hadoop.

Data stored in external databases cannot be directly accessed by MapReduce applications. This approach would put the system at the *Cluster* nodes at high risk of stress.

Sqoop simplifies loading large amounts of data from RDBMS to Hadoop, addressing this issue.

Most of the process is automated by Sqoop, which relies on the database to specify the data import structure. It imports and exports data using the MapReduce architecture, which provides a parallel and fault-tolerant approach.

Sqoop graduated from the incubator in March 2012, becoming a top-level project at Apache.

Some of its features include:

- Reads tables row by row and writes the file to HDFS.
- Imports data and metadata from relational databases directly into Hive.
- Provides parallel and fault-tolerant processing by using MapReduce in import/export activities.
- Uses the YARN framework for data import and export, making it fault-tolerant with parallelism.
- Allows selecting the range of columns to be imported.
- Allows specifying delimiters and file formats.
- Parallelizes database connections by executing SQL commands like SELECT (import) and Insert/Update (export).
- The default format for imported files in HDFS is CSV.
- Data type conversion:
- Imports individual tables or entire databases into HDFS files.
- A generic JDBC connector is provided to connect to any database supporting the JDBC standard. It has several plugins for connection to PostgreSQL, Oracle, Teradata, Netezza, Vertica, DB2, SQL Server, and MySQL.
- Creates Java classes that allow users to interact with the imported data.

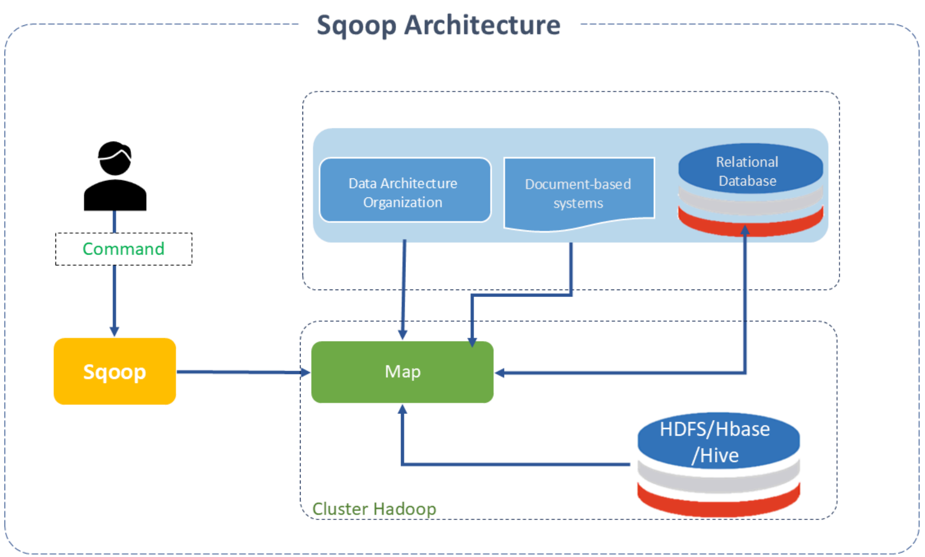
© Copyright Tecnisys 2026. All rights reserved.

Architecture of Apache Sqoop

Sqoop provides a command-line interface for end-users and can also be accessed via the Java API.

Data migration between Sqoop Hadoop and an external storage system is possible through **Sqoop connectors**, which enable it to use various well-known relational databases such as MySQL, PostgreSQL, Oracle, etc. Each of these connections can communicate with the DBMS to which it is linked.

During the execution of Sqoop, the transferred dataset is divided into several parts, and a map-only job is created with distinct mappers responsible for loading each partition. Sqoop uses the database information to deduce data types, handling each record securely.



Sqoop Architecture

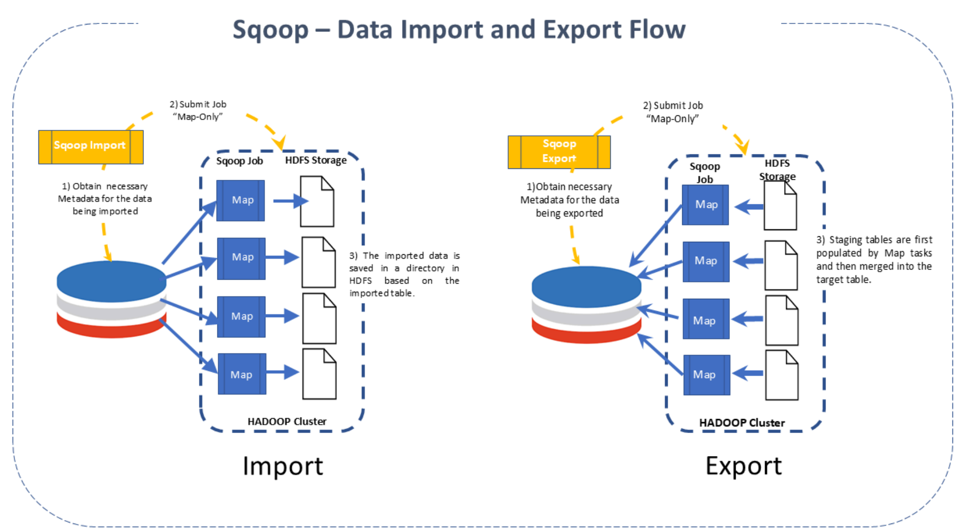
Sqoop is therefore composed of two main operations:

- Sqoop Imports
  This procedure is performed with the *Sqoop import* command. Each record loaded into the Hadoop DBMS as a single record is stored in text files as part of the Hadoop structure. When importing data, it is also possible to load and split Hive. Sqoop also allows incremental data import.

- Sqoop Exports

© Copyright Tecnisys 2026. All rights reserved.

Facilitates the task with the *export* command, which performs the operation in the opposite direction. Data is transferred from the Hadoop file system to the relational DBMS. Before completing the operation, the exported data is transformed into records.



Sqoop Flow

- [Apache Sqoop Resources]
  - It is possible to import the **results of an SQL query into HDFS with Sqoop**.
  - Offers **connectors for most RDBMS**, such as MySQL, Microsoft SQL Server, PostgreSQL, etc.
  - **Supports Kerberos network authentication protocol**, allowing nodes to authenticate users while communicating securely over an insecure network.
  - With a single command, Sqoop **can reload the entire table or specific sections** of the table.

When to use Apache Sqoop

- Processing OLTP databases using a Big Data tool.
- Integration of OLTP with Hadoop.
- Data ingestion into Hadoop.

How it works

The command entered by the user is parsed by Sqoop and executes the Hadoop *Map* only to import or export data, as the *Reduce* phase is only necessary when aggregations are required.

© Copyright Tecnisys 2026. All rights reserved.

Sqoop parses the arguments entered on the command line and prepares the *Map* task. A map job that runs several mappers depends on the number defined by the user on the command line.

During an import, each map task receives a portion of the data to be imported based on the command line.

Sqoop distributes the data evenly among the mappers to ensure high performance.

Then each mapper creates a connection with the JDBC database.

Best Practices for Apache Sqoop

- **Import to binary format:** Although imports to CSV file format are easy to test, some issues may arise when text stored in the database uses special characters. Importing to a binary format like Avro will avoid this problem and can speed up processing in Hadoop.

- **Control parallelism:** Sqoop works in the MapReduce programming model. It imports and exports data from most relational databases in parallel. The number of *Map* tasks per job determines this parallelism. Controlling parallelism allows dealing with database load and performance. There are two ways to explore parallelism in Sqoop:

  - **Changing the number of mappers:** Typical Sqoop jobs start with four mappers by default. To optimize performance, it is recommended to increase the Map tasks (parallel processes) to an integer value of 8 or 16. This can show a performance increase in some databases. Using *-m* or *--num--mappers* you can define the degree of parallelism in Sqoop.

    Terminal input

    ```
    Sqoop import \
    --connect jdbc:postgresql://postgresql.example.com/Sqoop \
    --username Sqoop \
    --password Sqoop \
    --table _table_name_ \
    --num-mappers 10
    ```

© Copyright Tecnisys 2026. All rights reserved.

- **Splitting by query:** When performing parallel imports, Sqoop needs a criterion to split the workload. It uses a *split column* to divide the workload. By default, it will identify the primary key column (if present) in a table and use it as the split column. The low and high values for the split column are retrieved from the database, and the map tasks operate on uniformly sized components of the total range. The *split-by* parameter divides the data in the column evenly based on the number of specified mappers. The *split by* syntax is:

Terminal input

```
Sqoop import \
--connect jdbc:postgresql://postgresql.example.com/_database_name_ \
--username _username_ \
--password _password_ \
--table _table_name_ \
--split-by _id_field_
```

WARNING

The number of *map* tasks must be less than the maximum number of possible parallel database connections. The increase in the degree of parallelism should be less than that available within your MapReduce Cluster.

- **Control data transfer process:** A popular method of improving performance is managing the path where we import and export data. Below we summarize some paths. To see the existing arguments, check Tables "Table 3" and "Table 29" -> import/export control arguments, in the User Guide

  - **Batch:** Which means that SQL statements can be batched when exporting data.

NOTE

The JDBC interface provides an API for batching in a prepared statement with multiple sets of values. This API is present in all JDBC drivers because it is required by the JDBC interface.

© Copyright Tecnisys 2026. All rights reserved.

Batch is disabled by default in Sqoop. Enable JDBC batching using the *batch* parameter.

Terminal input

```
Sqoop export \
--connect jdbc:postgresql://postgresql.example.com/_database_name_ \
--username _username_ \
--password _password_ \
--table _table_name_ \
--export-dir /data/_database_name_ \
--batch
```

- **Fetch size:** The default number of records that can be imported at once is 1,000. This can be altered by the *fetch-size* parameter, which is used to specify the number of records that Sqoop can import at once.

  Terminal input

  ```
  --connect jdbc:postgresql://postgresql.example.com/_database_name_ \
  --username _username_ \
  --password _password_ \
  --table _table_name_ \
  --fetch-size=n
  ```

  where n represents the number of entries Sqoop should fetch at a time.

  Based on available memory and bandwidth, the value of the *fetch-size* parameter can be increased relative to the volume of data that needs to be read.

- Direct Mode:
  By default, the Sqoop import process uses JDBC, which provides reasonable support. However, some databases may achieve higher performance by using database-specific utilities optimized to provide the best possible transfer speed, placing less pressure on the database server.

  By providing the *--direct* argument, Sqoop is forced to attempt to use the direct import channel. This channel may perform better than using JDBC.

© Copyright Tecnisys 2026. All rights reserved.

Terminal input

```
Sqoop import \
--connect jdbc:postgresql://postgresql.example.com/_database_name_ \
--username _username_ \
--password _password_ \
--table _table_name_ \
--direct
```

WARNING

There are several limitations that come with this faster import. Not all databases have native utilities available, and this mode is not available for all databases.

Sqoop has direct support for MySQL and PostgreSQL.

- **Custom Split Queries:** As seen, *split by* distributes data evenly for import. If the column has non-uniform values, the boundary query can be used if we do not get the desired results by only using the *split-by* argument. Ideally, configure the boundary query parameter as min(id) and max(id) along with the table name.

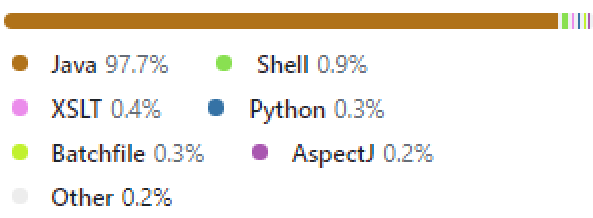
Terminal input

```
Sqoop import \
--connect jdbc:postgresql://postgresql.example.com/_database_name_ \
--username _username_ \
--password _password_ \
--query 'SELECT... FROM... JOIN ... USING ... WHERE $CONDITIONS' \
--split-by id \
--target-dir _table_name_ \
--boundary-query "select min(id), max(id) from _normalized_table_name_"
```

Apache Sqoop Project Details

Apache Sqoop was developed in JAVA.

© Copyright Tecnisys 2026. All rights reserved.



Languages of Sqoop

Technical Notes

RETIRED COMPONENT — REMOVAL PLANNED FOR TDP 3.1.0

Apache Sqoop has been **retired by the Apache Software Foundation** and moved to the Apache Attic, the official repository for inactive projects. This means the project no longer receives new releases, security fixes, or community support.

The last stable version available is **1.4.7** (released in 2017).

Impact for TDP users:

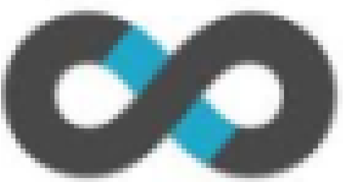
- Sqoop remains functional in current platform versions, but is not recommended for new projects.
- **Starting from TDP 3.1.0, this component will be removed from the platform.**

Sources:

- Apache Sqoop
- Apache Attic — Sqoop

© Copyright Tecnisys 2026. All rights reserved.

Apache Superset



Visualization

Data visualization consists of the graphical representation of information and data. Visual elements such as charts and maps facilitate the storytelling about specific data, making it more comprehensible, highlighting trends, exceptions, and generating new insights.

With the advent of "Big Data," it has become an incredibly relevant tool for interpreting and understanding the volume of data generated daily.

There are various tools for data visualization, among them is Apache Superset, a simple, easy-to-use tool that offers a range of options for all skill levels.

It is one of the best tools for data exploration and machine learning. Additionally, it offers a very user-friendly interface at a more affordable cost.

Features of Apache Superset

Built on popular open-source technologies such as JDBC and H2O, Apache Superset offers robust features for data visualization, exploration, and analysis. It is a fast, lightweight, intuitive business intelligence (BI) web application, filled with options that facilitate data exploration and visualization for users with any skill set.

It was created by Maxime Beauchemin, a data engineer, CEO, and founder of PRESET, who used an internal Airbnb hackathon (an event that brings together programmers, designers, and other software development professionals for a programming marathon) to create a BI tool from scratch.

The project, initially called "Caravel," became Apache Superset. Quickly adopted by dozens of companies, it increasingly took on more use cases. It was established as a complete open-source project and incubated with the Apache Software Foundation in 2016. Today, it is the leading open-source analytics platform, with one of the fastest-growing communities on GitHub.

© Copyright Tecnisys 2026. All rights reserved.

Among its main features, we can cite:

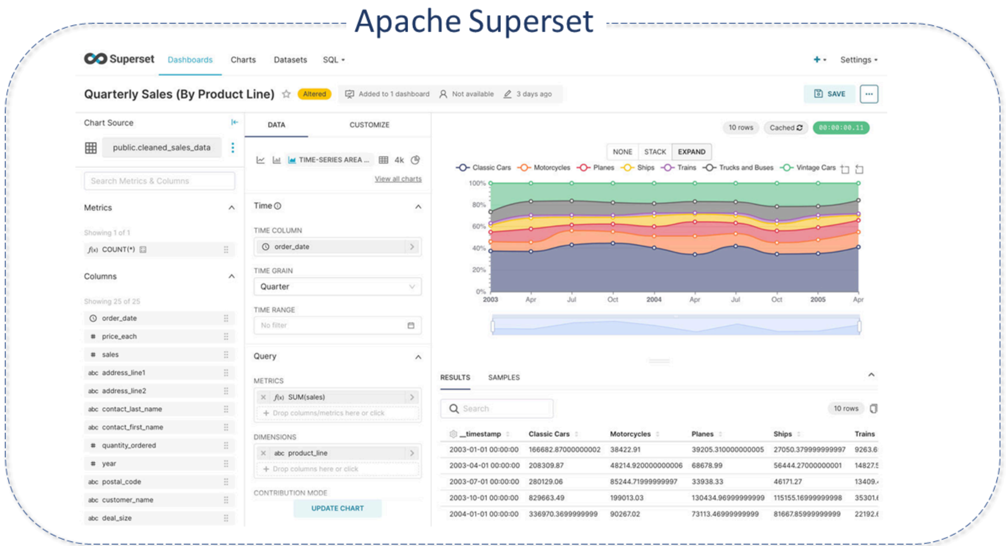
- **Intuitive Interface**: Allows visualizing data sets and creating interactive dashboards.
- **SQL IDE**: Facilitates the preparation of data for visualization, including a rich metadata browser.
- **Security**: It is one of its main advantages, as it offers complete control over data access. Allows adding users to the Database, providing access, and tracking behaviors.
- **Light Semantic Layer**: Empowers data analysts to quickly define custom dimensions and metrics.
- **Support for SQL Databases**: Compatible with most databases that use SQL.
- **Cache and Asynchronous Queries**: Improves performance by reducing the direct query load on databases.
- **Extensibility**: With complete access control to data, allowing the configuration of complex rules on who can access which resources and data sets.
- **Integration with major authentication backends**: Such as OpenID, LDAP, OAuth, Remote_user, etc.
- **Ability to add custom visualization plugins**.
- **API for programmatic customization**.
- **Cloud Native Architecture**: Designed for high availability and scalability in distributed environments.
    - Designed for High Availability and Scale in large and distributed environments.
    - Flexible in the choice of:
        - Web Server (Gunicorn, Nginx, Apache),
        - Metadata Database (MySQL, Postgres, MariaDB, etc),
        - Message Queue (Redis, RabbitMQ, SQS, etc),
        - Results Backend (S3, Redis, Memcached, etc),
        - Cache Layer (Memcached, Redis, etc).
- **Works well within Containers**.

© Copyright Tecnisys 2026. All rights reserved.

- **Allows the creation of Interactive Queries**: With selection of database, tables, and schema.

- **Requires no coding knowledge**: Designed for people who do not know code, such as business and financial analysts.

- **Accessible via Web and App**: That operate independently.



Apache Superset Interface

Architecture of Apache Superset

Apache Superset operates with a data-centered architecture, using a Dataset-Centric methodology that promotes the use of datasets similar to a Dataframe, that is, a tabular structure enriched with a subset of semantic features.

Apache Superset can be run in sequential mode for quick queries or distributed mode, where it distributes queries among workers.

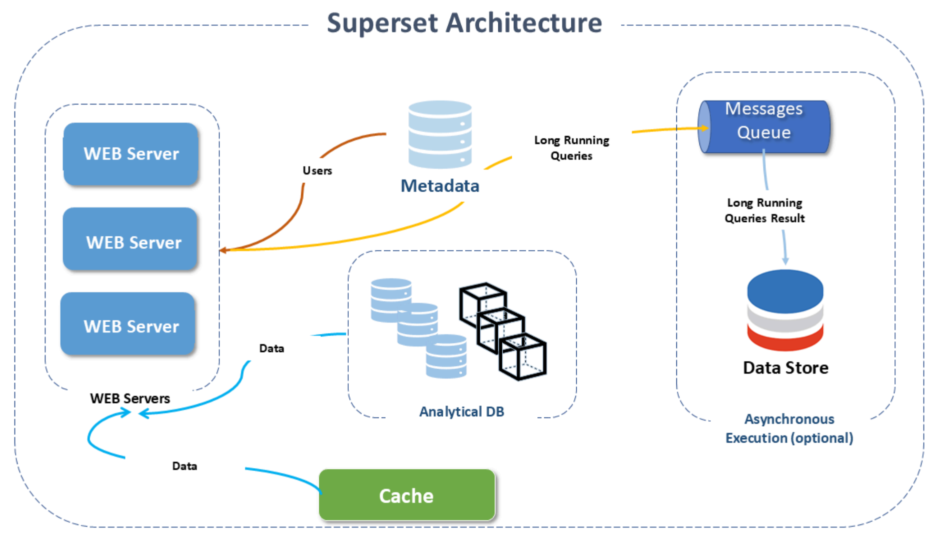
The main components involved in the Apache Superset solution are:

- **Web Servers**: Flask Python app used to connect to any database. Superset allows the choice of the Web Server and integrates with various options, such as Gunicorn, NGINX, Apache HTTP. The Superset web servers and the optional Superset Celery workers can expand to as many servers as needed.
- **Metadata Database**: (Metadata Database): Superset allows the choice of the metadata database engine and integrates with various options, such as MySQL,

© Copyright Tecnisys 2026. All rights reserved.

Postgres, MariaDB, etc.

- **Cache Layer**: Superset allows the choice of the cache layer and integrates with various options like Memcached, Redis, etc.
- **Message Queue for async queries**: Superset allows the choice of the message queue and integrates with various options like S3, Redis, Memcached, etc.
- **Results Backend**: For storing and retrieving query results.
- **Dashboard and Slices**: The Dashboard is a user interface that allows viewing various charts and data. Each section within the Dashboard is called a Slice, which in turn can be in various formats: text, chart, or, for example, a function.
- **SQL Lab**: SQL Lab is an SQL IDE with a wide range of features, with which it is possible to convert data into charts, for example.



Apache Superset Architecture

Superset works very well with metric and statistics services like NewRelic, StatsD, DataDog, and has the capability to perform analytical workloads on the most popular Database technologies.

Currently, it runs at scale in many companies, for example, in the Airbnb production environment, where, within Kubernetes, it serves more than 600 active daily users who view over 100,000 charts per day.

The main sectors and companies that adopt Superset can be seen here.

Resources of Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

Superset is enriched with features that support everything from creating dynamic visualizations to complex analyses, offering:

- Custom visualizations to explore and understand the data.
- SQL queries on the SQL Tab for data investigation.
- Code-free visualization building, or the SQL IDE for quick data integration and analysis.
- Lightweight and scalable data ingestion that works on existing data infrastructure, without requiring a separate ingestion layer.
- Basic semantic layer, where it is possible to control how data sources are displayed and handled.

Integration with Databases

Superset provides functionalities for connecting with various databases. It connects with almost all major databases, which facilitates the visualization and analysis of your data. It is compatible with Apache Spark SQL, PostgreSQL, Google Sheets, Amazon Athena, Amazon Redshift, Azure MS SQL, etc.

Types of Visualization

Apache Superset provides a wide variety of charts, tables, and layouts. Some examples are:

- Scatter Plot.
- Grids.
- Polygons.
- Path.
- Screen Grids.
- Arcs.

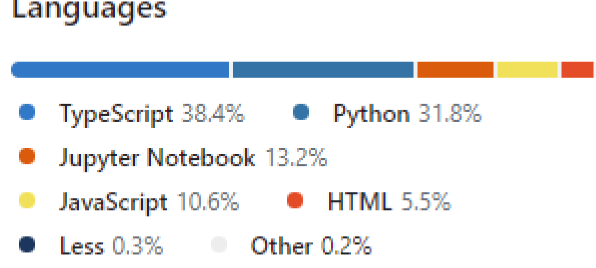
Details of Project Apache Superset

Developed predominantly in Python, Apache Superset also uses technologies such as Typescript and Flask App Builder internally. It supports Python version > 3.6 and can be installed in a variety of methods. The most common are:

- **locally**: where Python must be installed first and then pip installs the dependencies.
- **virtually**: Installation in a virtual environment is strongly recommended. Pyenv-virtualenv can be installed if pyenv is being used.

© Copyright Tecnisys 2026. All rights reserved.

- **Docker**: The simplest way to experiment with Superset locally is using Docker and Docker Compose on Linux or Mac OSX.



Languages of Superset

TDP Kubernetes

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **5.0.0**, distributed via Helm Chart `tdp-superset` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources

- Apache Superset Community
- GitHub Apache Superset

© Copyright Tecnisys 2026. All rights reserved.

Trino

Distributed Query Engine

Trino icon

This feature is only available starting from version 2.3.

A distributed query engine is designed to execute complex SQL queries on large volumes of data spread across multiple sources. It eliminates the need to move or duplicate data, allowing analytics to be performed directly where the data is stored, increasing efficiency and scalability.

Key Features of a Distributed Query Engine:

- **Parallel Execution:** Breaks queries into smaller tasks, which are processed simultaneously across different nodes in a cluster, optimizing execution time.
- **Connection to Multiple Data Sources:** Provides integration with data lakes, data warehouses, and relational databases, enabling federated query execution.
- **SQL as a Universal Language:** Analysts can perform queries using SQL without the need to learn specific languages.
- **Optimized Performance:** Uses techniques like predicate pushdown and in-memory storage to reduce latency and maximize performance.

Common use cases for distributed query engines include:

- Federated queries across multiple data sources.
- Real-time analysis of large volumes of data.
- Unifying data lakes and data warehouses to simplify governance and auditing.

Its basic operation involves:

1. The client sends a query to the engine.
2. The engine interprets the query, creates an execution plan, and distributes tasks to the *workers*.
3. Each *worker* processes its part, accessing data sources directly.
4. The results are consolidated and sent back to the client.

Mass Data Storage

Mass Data Storage

© Copyright Tecnisys 2026. All rights reserved.

Key Features of Trino

Trino is an **open-source** distributed query engine designed to execute SQL queries on large volumes of data stored in diverse sources such as data lakes, data warehouses, and relational databases.

Initially developed as Presto by Facebook engineers to meet its internal data analysis needs, Trino emerged in 2020 after a split from the Presto Foundation. Since then, it has become a benchmark for high-performance solutions in distributed data analytics.

Key Features:

- **Speed:** Designed for low-latency analytics, Trino uses highly parallel and distributed execution to process queries efficiently.
- **Horizontal Scalability:** Enables adding workers to increase processing capacity, capable of handling workloads at the exabyte scale, such as in large data lakes and data warehouses.
- **Simplicity:** Compatible with ANSI SQL, facilitating integration with BI tools like R, Tableau, Power BI, Superset, and more.
- **Versatility:** Supports interactive ad hoc analyses, long-running batch queries, and high-volume applications, ensuring sub-second response times in critical scenarios.
- **Local Analysis:** Queries data directly from sources like Hadoop, S3, Cassandra, and MySQL, eliminating the need to copy or move data, simplifying processes and reducing errors.
- **Federated Queries:** Enables executing queries across multiple data sources, such as HDFS, S3, relational databases, and data warehouses.
- **High Performance:** Its architecture is optimized for interactive workloads, ensuring minimal latency.
- **Extensibility:** Supports custom connectors, allowing integration with new data sources.
- **Reliability:** Widely used in mission-critical operations, such as financial reporting for public markets, by some of the largest global organizations.
- **Support for Various Formats:** Compatible with formats like Parquet, ORC, Avro, JSON, and CSV.
- **Open Community:** Developed under the leadership of the Trino Software Foundation, a nonprofit organization.

Trino Architecture

© Copyright Tecnisys 2026. All rights reserved.

Trino is a distributed query engine that processes data in parallel across multiple servers. Trino Clusters Servers are classified as Coordinators and Workers.

The following sections describe the main components of Trino's architecture.

Cluster

A Trino cluster consists of multiple nodes, including a Coordinator and zero or more Workers. Users connect to the Coordinator through SQL query tools. The Coordinator orchestrates tasks among the Workers and accesses connected data sources through configured catalogs.

Each query is processed as a stateful operation. The Coordinator distributes the workload among the Workers in parallel. Each node runs a single JVM instance, with additional parallelization using threads.

Node

A *Node* in Trino refers to any server within a cluster running a Trino process. It typically corresponds to a single machine since only one Trino process is recommended per machine.

Coordinator

The Coordinator is the central server responsible for parsing instructions, planning queries, and managing Worker nodes. Acting as the "brain" of the cluster, it tracks Worker activity, coordinates query execution, and communicates with clients and Workers via the REST API.

For development or testing, a single Trino instance can be configured to function as both a Coordinator and a Worker.

Worker

A Worker is a server responsible for executing tasks and processing data. Workers fetch data from connectors, exchange intermediate data, and communicate with the Coordinator via the REST API. Upon startup, a Worker registers with the Coordinator's discovery server for task allocation.

Client

Clients connect to Trino to send SQL queries and retrieve results. They can access configured data sources through catalogs and include tools such as command-line

© Copyright Tecnisys 2026. All rights reserved.

interfaces, desktop applications, and web-based systems. Some clients also support interactive query authoring, visualizations, and reports.

Data Source

Trino supports querying various data sources, including data lakes, relational databases, and key-value stores. Access to these data sources is configured using catalogs, which define the necessary connectors, credentials, and other parameters.

Below are the fundamental concepts associated with Data Sources in Trino:

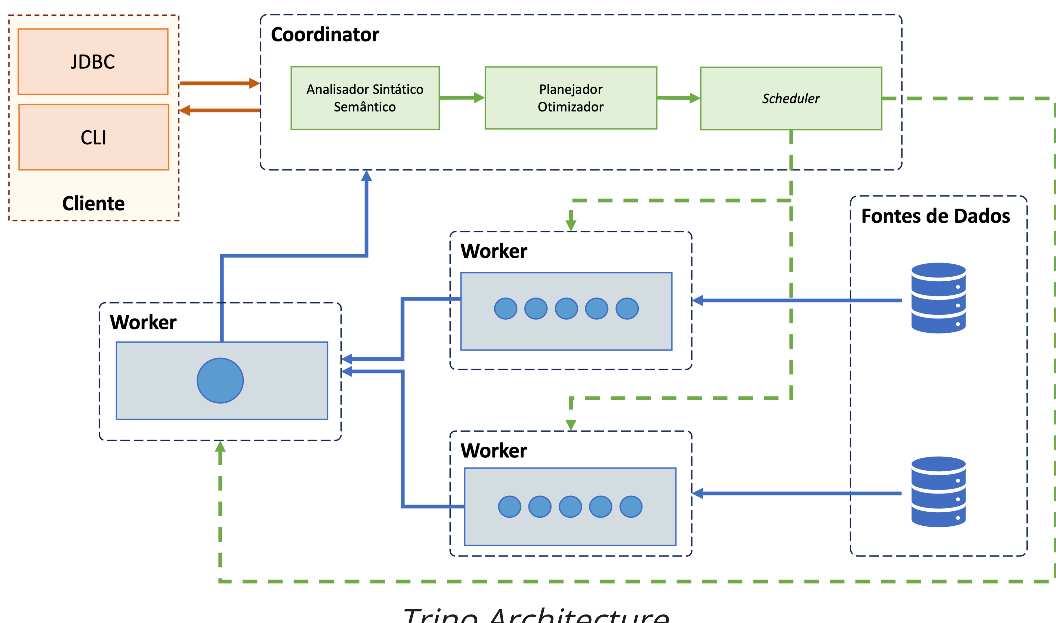
- **Connector:** Connectors allow Trino to interact with specific data sources, functioning like database drivers. Examples include connectors for Hive, Iceberg, PostgreSQL, MySQL, and Snowflake. Each catalog in Trino is associated with a connector.

- **Catalog:** A catalog is a collection of configuration properties for accessing a data source. Catalogs are defined in property files stored in Trino's configuration directory. A catalog can contain schemas and tables, enabling access to multiple data sources within a single cluster.

- **Schema:** Schemas organize tables and other objects within a catalog. They correspond to similar concepts in databases like Hive and MySQL.

- **Table:** A table consists of unordered rows organized into named columns with specific types. Tables are accessed by fully qualified names rooted in catalogs.

© Copyright Tecnisys 2026. All rights reserved.



Trino Architecture

Trino Query Execution Model

Trino executes SQL statements by transforming them into distributed queries executed across the cluster.

Below are more details about the terms used in Trino's query execution model:

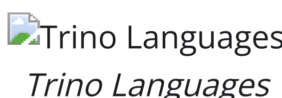
- **Statement:** A *statement* refers to the SQL text sent to Trino. It is converted into a query plan during execution.
- **Query:** A *query* encompasses the components and configuration needed to execute a statement. It includes stages, tasks, splits, and operators.
- **Stage:** *Stages* represent sections of a distributed query plan, organized hierarchically. Each query has a root stage that aggregates outputs from other stages.
- **Task:** *Tasks* execute stages in parallel on Workers. They operate on splits and involve multiple drivers to process data.
- **Split:** A *split* is a segment of a larger dataset processed by a task. The Coordinator assigns *splits* to tasks based on availability.

© Copyright Tecnisys 2026. All rights reserved.

- **Driver:** Drivers are the smallest units of parallelism, combining operators to process data within a task.
- **Operator:** Operators perform transformations on data, such as table scans or filters, and are combined within drivers.
- ***Exchanges***: *Exchanges* transfer data between nodes during query execution, allowing communication between stages through output buffers and exchange clients.

Trino Project Details

Trino was developed in **Java**, leveraging the robustness of the JVM for distributed processing.



Trino Languages

TDP Kubernetes

> **AVAILABLE IN TDP KUBERNETES**
>
> This component is also available in the **TDP Kubernetes** edition since version 3.0.
>
> The current version is **478**, distributed via Helm Chart `tdp-trino` v3.0.1.
>
> For configuration details, see the TDP Kubernetes documentation.

Sources: Trino - Overview

© Copyright Tecnisys 2026. All rights reserved.

Apache Zeppelin

Notebook

Typically used by data scientists in experiments and exploratory tasks, the notebook is an interactive computing tool that allows users to write and execute code, visualize results, and share insights.

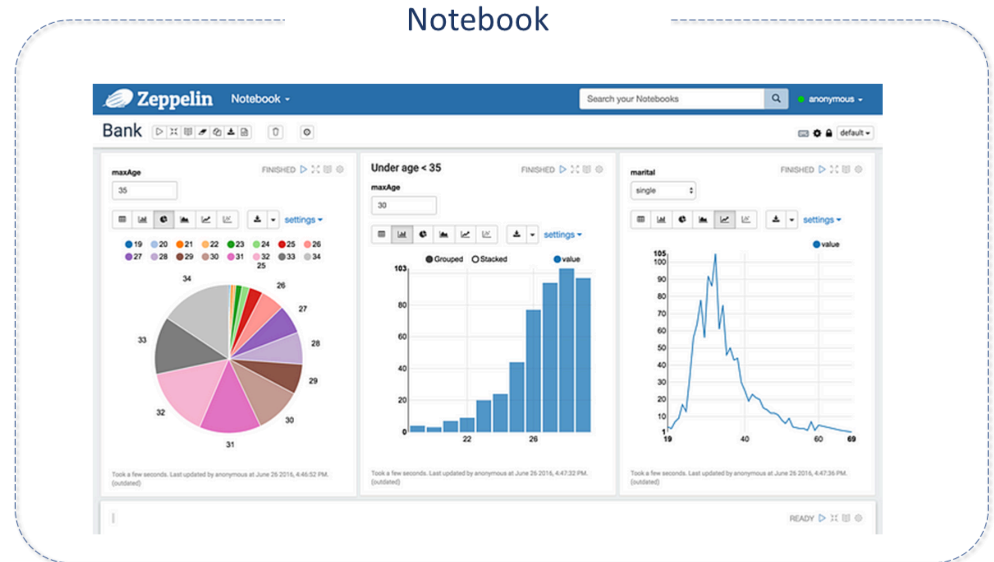
The concept was created by Stephan Wolfram, a computer scientist and physicist, who introduced Mathematica - the first computational notebook interface. Since then, the tool has proliferated and moved from academia to industry.

- **Apache Zeppelin** is a web-based "notebook" that provides ingestion, exploration, visualization, sharing, and interactive collaboration capabilities for Hadoop and Spark.

Apache Zeppelin began as a project named Zeppelin, from the company ZEPL (formerly known as NFLabs), led by Moon Sool Lee. In 2014, it became an incubator project at the Apache Software Foundation, and soon after, in 2016, it became a top-level project at Apache.

According to its creators, the term "notebook" given to Zeppelin is an analogy to a notebook, basing its functions on "paragraphs."

© Copyright Tecnisys 2026. All rights reserved.



Notebook

Apache Zeppelin Features

Zeppelin's interactive notebooks allow engineers, analysts, and data scientists to optimize their work with data. In this way, **it can be seen as an interface that connects users with the technologies they want to use for data processing**.

Zeppelin is very useful **for working interactively with data science workflows**, developing, organizing, and executing **analyses and** visualizing their **results**, without the need to use the command line or query Cluster details.

It also offers **support for multiple language backends** and a **growing ecosystem of data sources**.

- **Multiple language backends**: The software stands out for its **ability to integrate various other technologies through a functionality called *interpreter***, which is a layer for backend integration (working behind the application), already having more than 20 interpreters in its official distribution package.

  Among the various interpreters it supports, we can mention Apache Spark, Python, JDBC, Markdown, and Shell.

- **Integration with Apache Spark**: Zeppelin is integrated with Apache Spark. There is no need to create a separate module, plugin, or library for it. This integration provides:

© Copyright Tecnisys 2026. All rights reserved.

    - Automatic SparkContext and SQLContext.
    - Runtime dependency JAR loading from the local filesystem or Maven repository.
    - Job cancellation with progress display.
- **Data Visualization**: Some basic charts can also be used.

  Visualizations are not limited to SPARKSQL queries.

  Any output from any backend language can be recognized and visualized.

    - **Dynamic Charts**: Apache Zeppelin aggregates values and displays them in dynamic charts. With a simple drag-and-drop, it is possible to create a chart with multiple aggregated values, including sum, count, average, minimums, and maximums.

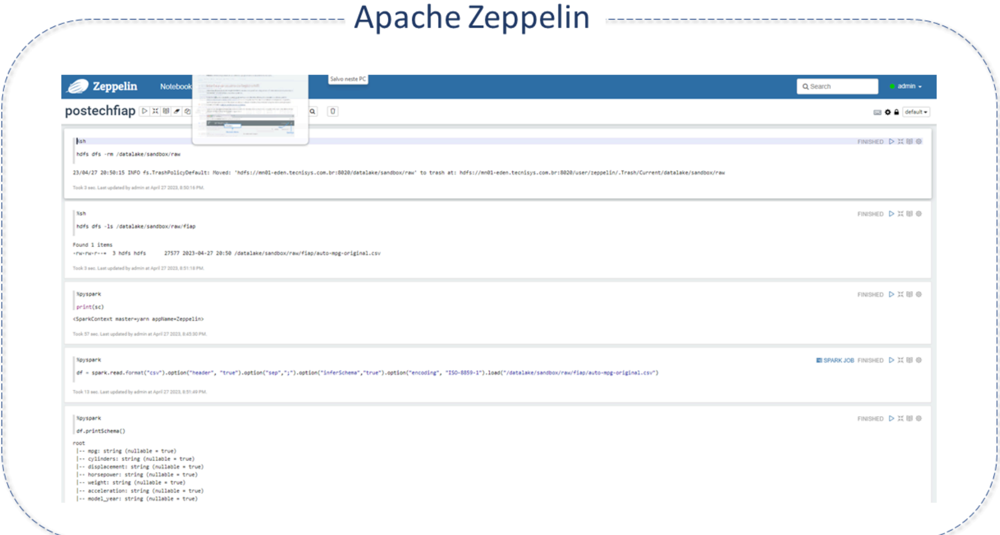
    - **Dynamic Forms**: Apache Zeppelin can dynamically create some input forms in your notebook.

- **Notebook and Paragraph Collaboration**: The notebook URL can be shared, and then Zeppelin can broadcast all changes in real-time, as well as collaborate like Google Docs.

  It can also provide a URL to display only the result, on a page without menu or button, within the Notebook, which can be easily embedded as an iframe on the user's site.

- **100% Open Source**: Apache Zeppelin is licensed under Apache2. It has a very active development community.

© Copyright Tecnisys 2026. All rights reserved.



Zeppelin Interface

Apache Zeppelin Architecture

Apache Zeppelin is divided into 03 layers:

- **Front-end**: From a Web Browser, the user interacts with Zeppelin's *Frontend*, which is **based on *AngularJS*** (a platform for building web applications based on *Ecmascript*) **and *Twitter Bootstrap*** (a *CSS* framework) that make the web application interface more fluid and dynamic.

  The *front-end* layer **communicates with the Apache Zeppelin server** through two possible interfaces:

  - **REST**: An architectural style to define **HTTP protocol constraints and properties.**
  - **Web Socket**: A technology for **bidirectional communication over *full-duplex* channels** - transmitter and receiver can simultaneously transmit data in both directions over a single TCP socket.
- **Zeppelin Server**: The server **operates on a Virtual Machine** (JVM - Java Virtual Machine) which also acts as the Notebook interpreter.

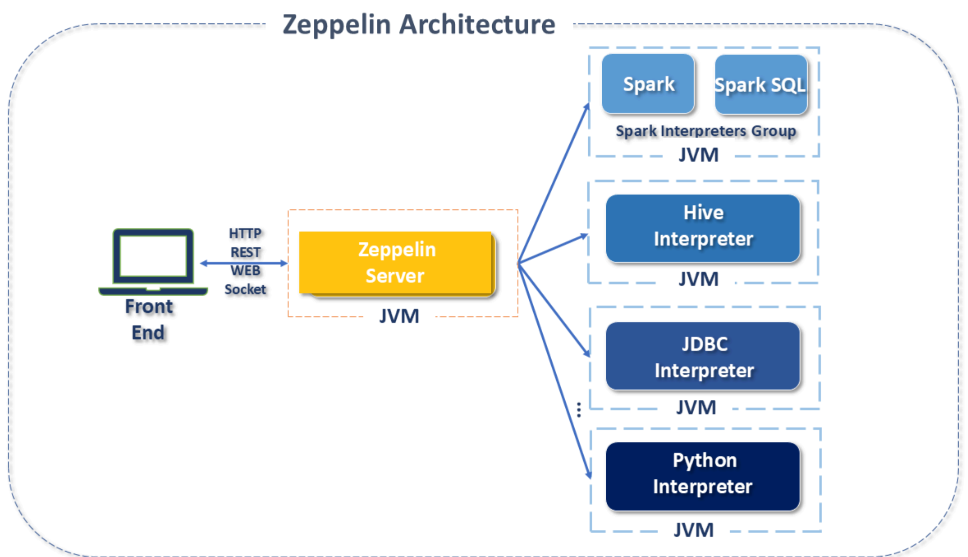
- **Interpreter**: The interpreter **communicates with a program running in Zeppelin's background via *Apache Thrift***, a technology that allows **defining data types and service interfaces** in a simple definition file.

© Copyright Tecnisys 2026. All rights reserved.

Taking this file as input, the compiler generates the code to be used to easily create clients and servers that communicate easily between programming languages.

The interpreter is a feature that makes Zeppelin "pluggable" to other technologies. Each interpreter process belongs to a group of interpreters that act as a start and stop unit of the interpreter.

To learn about all the interpreters that Apache Zeppelin supports, click https://zeppelin.apache.org/docs/latest/#available-interpreters.



Zeppelin Architecture

Data Visualization with Zeppelin

The Notebook is composed of **paragraphs**.

Each paragraph is a box that **receives some type of predefined script in the interpreters**.

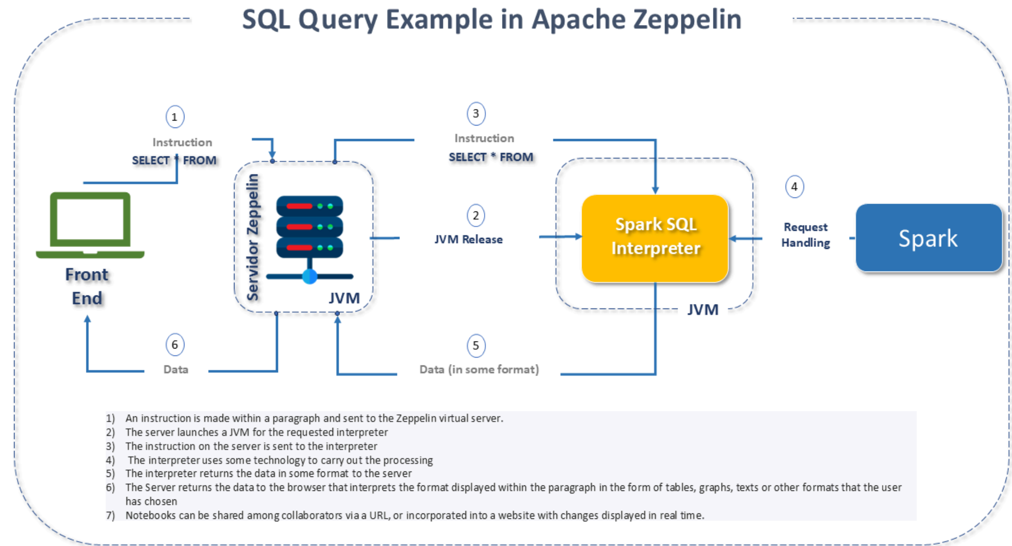
The interpreted text has a "%" mark to determine the interpreter and script to be executed.

Through the interface built with *Angular* and *Bootstrap*, the user can customize their visualization, placing paragraphs in columns to allow simultaneous display of the result.

The interaction between the user, the tool, and the data is provided by the front-end.

© Copyright Tecnisys 2026. All rights reserved.

The figure below is an example of Zeppelin's workflow, using a Spark Backend to handle an SQL query.



Zeppelin Example

Apache Zeppelin Best Practices

- **Installation and Versions**: It is recommended to install **Zeppelin with Ambari** and always use the **latest version** of Zeppelin, ensuring alignment with security and stability fixes.

- **Implementation Choices**: Although any node can be selected, the best place to install Zeppelin is on a **gateway node** when the Cluster firewall is turned off and protected externally.

- **Hardware Requirements**: **More memory and more cores** always benefit performance: a minimum of 64 GB and 8 cores is recommended. Number of users: A Zeppelin node can support 8 to 10 users. For more users, multiple instances can be configured.

- **Security**: Like any software, **security depends on the threat matrix and deployment options**:

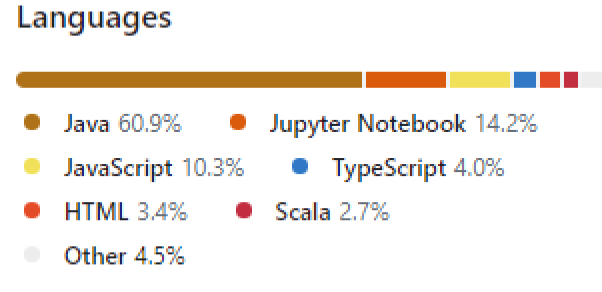
    - Authentication
    - Kerberize the Cluster using Ambari
    - Configure Zeppelin to leverage corporate LDAP

© Copyright Tecnisys 2026. All rights reserved.

  - Do not use Zeppelin's local user-based authentication except for demonstrations.
- **Interpreters**:
  - **Avoid using the Shell interpreter**, as the security isolation is not ideal.
  - **Do not use the interpreter UI for representations**. It works only for Livy and JDBC (Hive) interpreters.
  - Users should **restart their own interpreter sessions** from the button on the *Notebook page, and not on the interpreter page*, which would restart sessions for all users.
  - **Leverage the Livy interpreter** for Spark jobs on the Cluster, as it provides optimal identity propagation.
  - **Choosing the interpreters**: By default, Zeppelin will register and show all interpreters under the $ZEPPELIN_HOME/interpreters folder. However, there is an **option to specify which interpreters should be included or excluded** through the *zeppelin.interpreter.include* and *zeppelin_interpreter.exclude* properties. Only one of them can be specified.
  - It is possible to criate a new Interpreter, and the task is simple. Just extend the abstract class *org.apache.zeppelin.interpreter* and implement some methods.

Apache Zeppelin Project Details

Apache Zeppelin is based on JVM, functioning as a web application through Jetty and allows paragraphs in notebooks to be written in dozens of different languages such as Scala, Python, R, Markdown, and SQL.



Zeppelin Languages

© Copyright Tecnisys 2026. All rights reserved.

Sources:

- Apache Zeppelin

© Copyright Tecnisys 2026. All rights reserved.

Apache Zookeeper

Centralized Coordination Service

Distributed applications consist of multiple software components that operate simultaneously on various scalable physical servers, potentially spanning hundreds or thousands of machines.

Obviously, this system is subject to hardware failures, crashes, system failures, communication failures, and so on. These are failures that do not follow a pattern and, therefore, make it difficult to apply fault tolerance code to the application logic or system design.

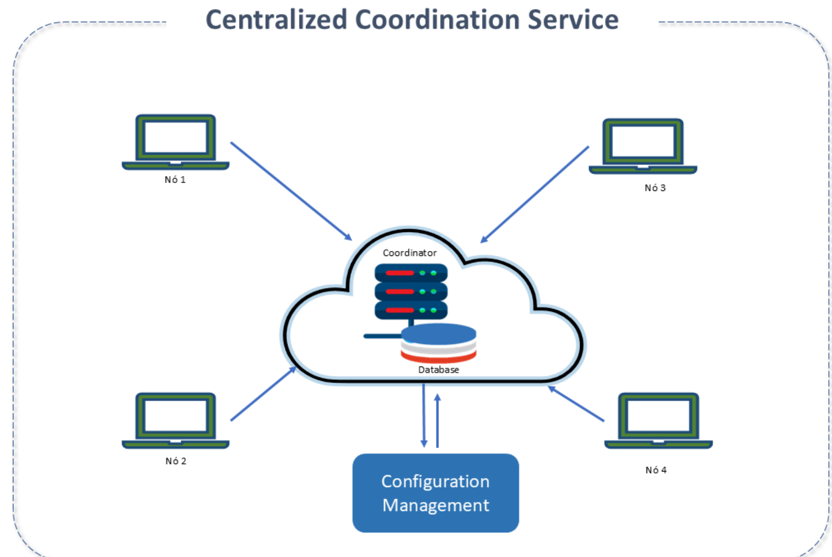
Additionally, achieving correct, fast, and scalable Cluster coordination is difficult and often prone to errors, potentially leading to inconsistencies in the Cluster.

Centralized coordination ensures the maintenance of the consistent "state" and performance of these systems (such as computers and software), avoiding undocumented changes in the environment and helping to avoid performance problems, inconsistencies, or compliance issues.

Performing these tasks manually is complex work in large systems like Big Data. It can involve thousands of components for each application.

For this, there are tools that provide a plan and a single version and the desired state of the organization's systems, as well as visibility of any configuration changes, through audit trails and change tracking.

© Copyright Tecnisys 2026. All rights reserved.



Centralized Coordination Service

Apache Zookeeper Features

Apache Zookeeper is a volunteer open-source project by Apache, originally developed by YAHOO and now widely used by major organizations such as Yahoo, Netflix, and Facebook (a complete list of organizations and projects that use Zookeeper can be seen here).

It is an open-source centralized service for the coordination of distributed applications.

It provides a set of "primitives" accessible through simple APIs on which applications can be built to implement high-level services, making less complicated:

- Cluster association operations (detection of node exit or joining).
- Distributed synchronization (locks and barriers).
- Naming (identifying nodes in a Cluster by name, similar to DNS).
- Configuration management (most recent and updated configuration information of the system for a joining node).

The main intention of Apache Zookeeper is to allow application developers to focus on business logic and rely entirely on Zookeeper for correct coordination:

© Copyright Tecnisys 2026. All rights reserved.

- Unlike conventional file systems, Zookeeper provides **high throughput and low latency**.
- It runs on a collection of machines and is designed for **high availability**, avoiding the introduction of points of failure in systems.
- Order is very important. Its **ordering** allows sophisticated synchronization primitives to be implemented on the client side.

  All updates are ordered. Each update is marked with a number reflecting this order, called *zxid* (Zookeeper Transaction Id), which is unique for each update.

  Reads (and watches) are ordered concerning updates. Read responses are stamped with the last *zxid* processed by the server that serves the read.
- The **performance** aspects of Zookeeper allow it to be used in large distributed systems.

  Performance and **scalability** are achieved through the watches mechanism, which allows clients to register to receive notifications whenever a *znode* undergoes some change (creation or deletion, changes in data held by the *znode*, creation or deletion of one of the children in a _znode's subtree).
- It facilitates loosely coupled interactions.

Apache Zookeeper Architecture

Zookeeper is a distributed and highly reliable centralized coordination application, following the client-server model and operating on a set of replicated servers, known as an ensemble, similar to the management of services such as DNS.

Zookeeper has a **simple architecture**, with a standard hierarchical namespace (which resembles the Unix filesystem) of data registers, called *znodes* (each node of the tree), making it an efficient solution for configuration information management.

Clients are nodes that **consume** services, and **servers** are nodes that **provide** services.

Paths to the nodes are expressed as **canonical paths** (shortest access to a file from the root directory), absolute, and separated by slashes.

© Copyright Tecnisys 2026. All rights reserved.

There are no relative references.

The main difference between Zookeeper and a standard file system is that **every *znode* can have associated data as children** (every file can also be a directory and vice versa), and *znodes* are limited to the volume of data they hold.

Zookeeper was designed to **store coordination data**: status information, configuration, location information, etc.

This type of meta-information is usually measured in kilobytes, if not bytes.

Therefore, it has an **integrated "sanity" check** of 1M, to prevent it from being used as a large data store, but in general, it is used to store much smaller data parts.

The main components of Zookeeper's architecture are:

- **Clients**: that connect to the service, using Zookeeper client library APIs (responsible for the interaction of an application with the Zookeeper service), through any member of the Ensemble.

  They can send and receive requests and responses, as well as notifications and heartbeats, through a TCP connection.

  If the connection is interrupted, the client will connect to a different server.

  When it connects for the first time, the first server will set up a session for the client.

  Read requests are processed locally on the server to which it is connected and served from the local replicas of each server's database.

  If the request registers a watch on a *znode*, this will also be tracked locally on the server.

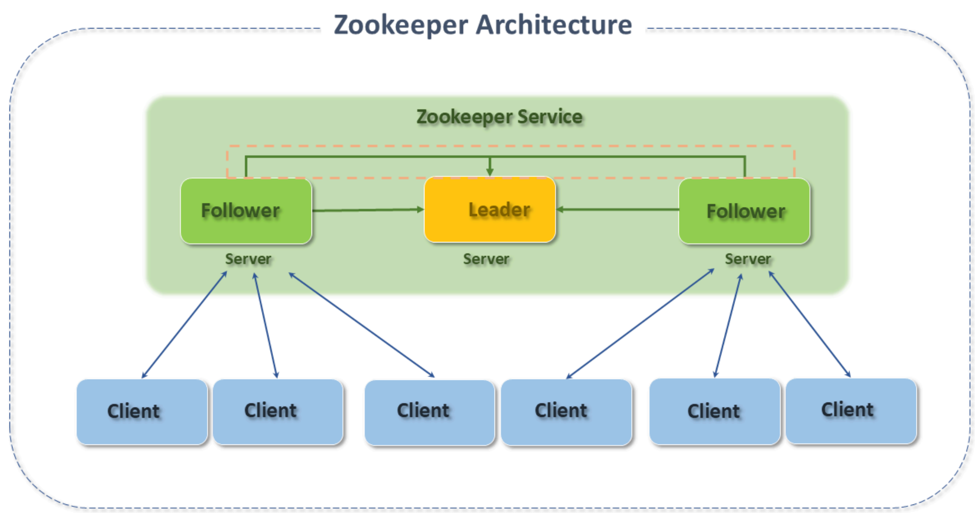
  Write requests, which change the state of the service, are processed by an agreement protocol and forwarded to other Zookeeper servers. They pass consensus before a response is generated.

  Synchronization requests are also forwarded to another server, but they do not pass consensus.

© Copyright Tecnisys 2026. All rights reserved.

- **Leader**: is the server that automatically recovers failed nodes. All client write requests are forwarded to the Leader.

- **Followers**: are non-leader servers that receive message proposals from the leader and agree to the delivery of messages.



Zookeeper Architecture

The messaging layer handles leader replacement in case of failure and synchronization between followers and leaders.

Zookeeper uses the **standard atomic messaging protocol** to ensure data consistency among nodes throughout the system. This ensures that local replicas never diverge.

After receiving a data change request, the leader writes the data to disk and then to memory.

Zookeeper follows a **shared and hierarchical storage model** similar to a traditional file system. The abstractions provided by the service allow *znodes* to be manipulated simply and efficiently.

The Zookeeper service is **replicated across the set of machines that comprise it**.

These machines maintain an in-memory image of the data tree along with transaction logs and snapshots in persistent storage. Once the data is held in memory, it achieves high throughput and low latency.

© Copyright Tecnisys 2026. All rights reserved.

The disadvantage is that the size of the database that Zookeeper can manage is limited by memory.

This limitation is another reason to keep the amount of data stored in *znodes* small.

The servers that make up the Zookeeper service must know each other.

As long as the majority is available, Zookeeper will be available. Clients must also know the list of servers because they create an identifier for the service using this list.

Znodes

The Zookeeper namespace is composed of data registers known as znodes, similar to files and directories. The Znode maintains the statistical structure that includes version number for data changes, ACL changes, timestamps, to allow cache validations and coordinated updates. Every time a *znode's* data changes, the version number increases.

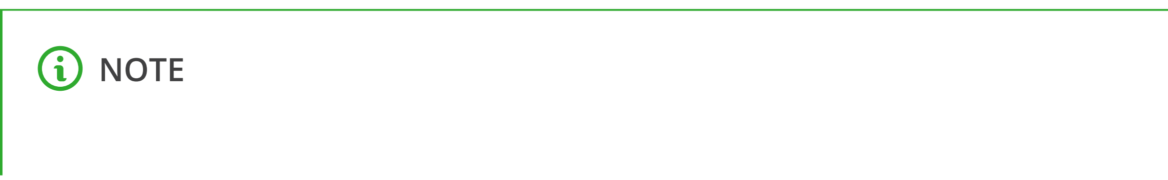
There are two types of *Znodes*:

- **Persistent**: Persistent Znodes only stop being part of the namespace when they are deleted. They exist to store data that needs to be highly available and accessible by all components of a distributed application.

- **Ephemeral**: Ephemeral *Znodes* are deleted by Zookeeper when the client's session ends, which can occur due to a failure disconnection or a connection termination. Although linked to the client's session, they are visible to all clients, depending on the associated Access Control List (ACL) policy. They can also be deleted by the creating client or any other authorized one.

  Ephemeral *znodes* can be used to build distributed applications where it is necessary for components to know each other's state or the state of constituent resources.

- **Sequential**: There is a third node, pointed out by some as another type of *znode* - the sequential. However, this *znode* should be understood as a qualifier of the two main ones.



© Copyright Tecnisys 2026. All rights reserved.

> A sequence number is assigned by Zookeeper as part of its name during creation. The value of a counter maintained by the parent *znode* is appended to the name.

These *znodes* are used to implement a global distributed queue because sequence numbers can impose an ordering. Also, to design a locking service for a distributed application.

Best Practices with Apache Zookeeper

Things to Avoid:

There are some issues that should be avoided through proper Zookeeper configuration:

- **Inconsistent server list**: The Zookeeper server list used by clients must match the Zookeeper server list that each Zookeeper server has. The client list must be a subset of the actual list. Additionally, the server lists in each Zookeeper server configuration file must be consistent with each other.

- **Incorrect transaction log placement**: The most critical part for Zookeeper performance is the transaction log. Zookeeper synchronizes transactions with the media before returning a response. A dedicated transaction log device is key to good and consistent performance.

  The log should never be placed on a busy device as this will impair performance. If there is only one storage device, we suggest placing the trace files on NFS and increasing the snapshotCount. This does not eliminate the problem but can mitigate it.

- **Swap to disk**:

  - Special care is needed when configuring the Java heap size. The situation where Zookeeper performs a swap to disk should be avoided. Everything is ordered, so if processing a request "swaps" to disk, all other requests in the queue are likely to do the same.

  - It is advisable to be conservative in estimates: With 4G of RAM, a maximum Java heap size should not exceed 3G, for example, because the operating system and cache also need memory.

© Copyright Tecnisys 2026. All rights reserved.

- The best practice is to perform load tests, ensuring to stay well below the limit that could cause a swap.

- System monitoring like vmstat can be used to monitor virtual memory statistics and decide on the optimal memory size depending on the application's needs. In any case, always avoid swapping.

Things to do:

- It is advisable to periodically clean the Zookeeper data directory if the autopurge option is not enabled. This directory contains the snapshot and transactional log files.

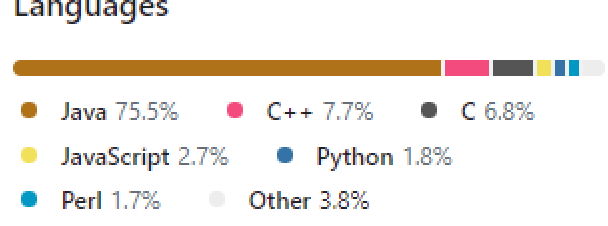
- Additionally, it is important to assess the need for a backup, which, being Zookeeper a replicated service, is only necessary on one of the ensemble servers.

- Zookeeper uses Apache log4j as the logging infrastructure. It is advisable to set up automatic rollover using the built-in Log4j feature for Zookeeper logs as the log files grow.

Zookeeper Project Details

Zookeeper is implemented in **Java** and offers **bindings for various other programming languages** such as C, Java, Perl, and Python.

The full list of available client bindings in the community can be accessed here.



Zookeeper Languages

Sources:

- Wiki.Apache
- Zookeeper.apache.org
- Zookeeper GitHub

© Copyright Tecnisys 2026. All rights reserved.

PART II - QUICK START

© Copyright Tecnisys 2026. All rights reserved.

TDP Sandbox

Error loading this section: GET http://127.0.0.1:3002/tdp/en/commons/sandbox -> 404

© Copyright Tecnisys 2026. All rights reserved.

PART III - INSTALLATION

© Copyright Tecnisys 2026. All rights reserved.

Minimum Requirements

The minimum requirements for implementing a *Big Data Cluster* may vary depending on the desired services or the technical and organizational needs of each environment.

Minimum Number of Machines

To minimally meet the replication and high availability requirements of the Platform's main components, we suggest:

- 07 machines (*physical or virtual*), as follows:
  - 03 machines dedicated to control, management, and administration components (*Masters*)
  - 03 machines dedicated to storage, operation, and execution components (*Workers*)
  - 01 machine dedicated to Apache Ambari and other utility components (*Utility* or *Edge*)

NOTE

Clusters with a reduced number of machines can be implemented as long as the technical requirements, sizing, and processing capacity of each component are respected. For example, a *Cluster* exclusively for the Apache Kafka service can initially be implemented with just 03 machines.

WARNING

It is recommended that the machines be exclusively used for the *Big Data Cluster*.

The amount of computing resources (CPU, RAM, disk, etc.) indicated for each machine requires the evaluation of several factors, such as the services to be installed, desired workloads, expected data volume, among others.

If you need support, contact us through the Support Area.

© Copyright Tecnisys 2026. All rights reserved.

NOTE

For downloading the installation packages, 30GB of free disk space is required.

Supported Virtualizers

TDP can be installed on physical or virtual machines. The supported hypervisors are:

- VMware vSphere Hypervisor (ESXi) 5.1 and later versions;
- Microsoft Hyper-V Server 2012 and later versions; and
- Oracle VirtualBox 5.0 and later versions.

Supported Operating Systems

Currently, TDP is available for the following operating systems:

- CentOS 7.5 and higher minor versions;
- CentOS 8.x and 9.x;
- Red Hat Enterprise Linux 7.5 and higher minor versions;
- Red Hat Enterprise Linux 8.x and 9.x;
- Rocky Linux 8.x and 9.x; and
- AlmaLinux 8.x and 9.x.

Support for additional Linux distributions will be available soon.

Supported Browsers

To access the graphical interfaces of the components in a *TDP Cluster*, we recommend using the following browsers:

- Google Chrome 96 or later versions;
- Mozilla Firefox 94 or later versions; and
- Microsoft Edge 96 or later versions.

© Copyright Tecnisys 2026. All rights reserved.

Support Tools

The tools listed below can facilitate the deployment and administration activities of the *Big Data Cluster*.

- **epel release** - A set of open-source packages that provides various complementary software for Enterprise Linux distributions, such as administration tools, development, monitoring, and other activities.
- **telnet** - A network tool that enables remote connection between two machines, assisting in testing and troubleshooting communication issues between servers.
- **net tools** - A package with several useful network tools (e.g. *netstat*) for scanning IP ranges, route analysis, "ping", identifying connections, among other activities.
- **wget** - A tool used for requesting, or downloading, content and files from servers on the internet. Supports downloads via FTP, SFTP, HTML, and HTTPS.
- **vim** - A versatile text editor. An enhanced version of the "vi" text editor.

Instructions

To install them, run the command below:

Terminal input

```
yum install epel-release telnet net-tools wget vim
```

© Copyright Tecnisys 2026. All rights reserved.

```
(06.08.2024 19:11:24)[root@big-tdp-220-02 ~]$ yum install epel-release telnet net-tools wget vim
Rocky Linux 8 - AppStream                                                     5.2 kB/s | 4.8 kB     00:00
Rocky Linux 8 - BaseOS                                                        5.0 kB/s | 4.3 kB     00:00
Rocky Linux 8 - Extras                                                        3.5 kB/s | 3.1 kB     00:00
Extra Packages for Enterprise Linux 8 - x86_64                                 74 kB/s |  92 kB     00:01
Package epel-release-8-18.el8.noarch is already installed.
Package telnet-1:0.17-76.el8.x86_64 is already installed.
Package net-tools-2.0-0.52.20160912git.el8.x86_64 is already installed.
Package wget-1.19.5-11.el8.x86_64 is already installed.
Package vim-enhanced-2:8.0.1763-19.el8_6.4.x86_64 is already installed.
Dependencies resolved.
==============================================================================================================
 Package                          Architecture           Version                    Repository           Size
==============================================================================================================
Upgrading:
 epel-release                     noarch                 8-20.el8                   epel                 24 k

Transaction Summary
==============================================================================================================
Upgrade  1 Package

Total download size: 24 k
Is this ok [y/N]: y
Downloading Packages:
epel-release-8-20.el8.noarch.rpm                                               74 kB/s |  24 kB     00:00
--------------------------------------------------------------------------------------------------------------
Total                                                                          14 kB/s |  24 kB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                                    1/1
  Running scriptlet: epel-release-8-20.el8.noarch                                                       1/1
  Upgrading        : epel-release-8-20.el8.noarch                                                       1/2
  Running scriptlet: epel-release-8-20.el8.noarch                                                       1/2
  Cleanup          : epel-release-8-18.el8.noarch                                                       2/2
  Running scriptlet: epel-release-8-18.el8.noarch                                                       2/2
  Verifying        : epel-release-8-20.el8.noarch                                                       1/2
  Verifying        : epel-release-8-18.el8.noarch                                                       2/2

Upgraded:
  epel-release-8-20.el8.noarch

Complete!
```

Figure 1 - Installation of support tools

© Copyright Tecnisys 2026. All rights reserved.

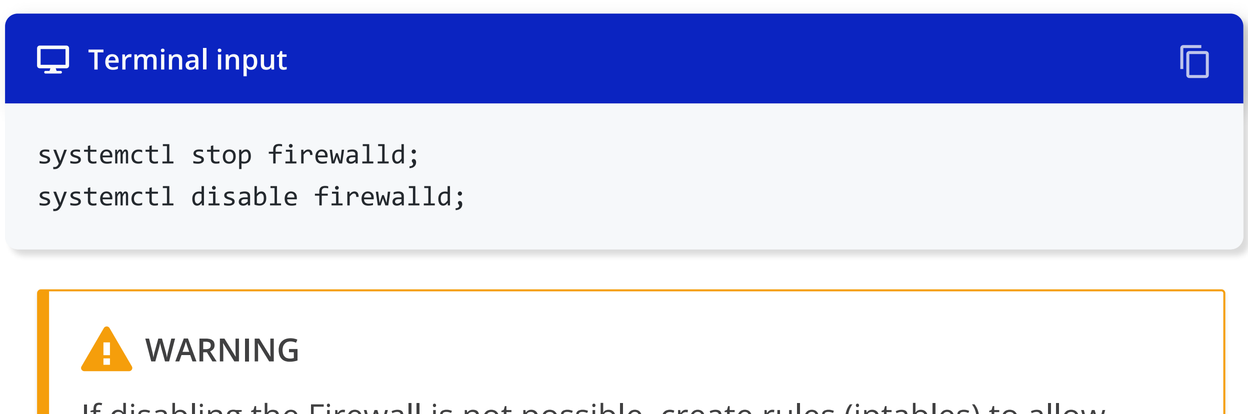
Environment Preparation

The following procedures should be performed on all machines in the *Big Data Cluster*.

Firewall Deactivation

Instructions

1. Disable the Firewall service:



Terminal input

```
systemctl stop firewalld;
systemctl disable firewalld;
```

> **WARNING**
>
> If disabling the Firewall is not possible, create rules (iptables) to allow communication between the machines on the ports used by the TDP Platform services.

SELinux Deactivation

Instructions

1. Temporarily disable SELinux:

Terminal input

```
setenforce 0;
```

2. Permanently disable SELinux:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
sed -i --follow-symlinks "s+SELINUX=enforcing+SELINUX=disabled+g"
/etc/selinux/config;
```

3. Restart the machine to apply the permanent SELinux deactivation.

Kernel Parameter Configuration

Instructions

1. Minimize the use of swap space:

```
echo 'vm.swappiness=1' >> /etc/sysctl.conf;
sysctl -p /etc/sysctl.conf;
```

> **NOTE**
>
> The Linux kernel provides a tunable setting that controls how frequently the swap space on disk is used, called *swappiness*. A *swappiness* value of zero means that the disk will be avoided unless absolutely necessary (when the server runs out of memory), while a *swappiness* value of 100 means that programs will use swap space almost instantly. Reducing the *swappiness* value reduces the likelihood of the Linux kernel sending an application's memory to swap space. Swap space is extremely slower than memory because it uses the disk instead of RAM. When processes' memory is swapped to disk, they can experience pauses, which can cause problems in services, such as Apache Zookeeper.

```
(07.08.2024 18:03:33)[root@big-tdp-220-02 ~]$ echo 'vm.swappiness=1' >> /etc/sysctl.conf
(07.08.2024 18:03:40)[root@big-tdp-220-02 ~]$ sysctl -p /etc/sysctl.conf
net.ipv6.conf.all.disable_ipv6 = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
net.ipv6.conf.all.disable_ipv6 = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.swappiness = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1

(07.08.2024 18:03:48)[root@big-tdp-220-02 ~]$
```

© Copyright Tecnisys 2026. All rights reserved.

Figure 1 - Swap minimize

2. Change the default behavior of RAM memory allocation:

Terminal input

```
echo 'vm.overcommit_memory=1' >> /etc/sysctl.conf;
sysctl -p /etc/sysctl.conf;
```

> **NOTE**
>
> In *Big Data* environments, where it is common to have machines with large amounts of RAM, we recommend setting *overcommit_memory* to 1. Unlike the value 0, which uses a heuristic approach to memory requests (*malloc*), the value 1 assumes there is always sufficient physical memory, significantly improving the performance of memory-intensive tasks.

```
## Changing the Default RAM Allocation Behavior
(07.08.2024 18:04:08)[root@big-tdp-220-02 ~]$ echo 'vm.overcommit_memory=1' >> /etc/sysctl.conf
(07.08.2024 18:04:15)[root@big-tdp-220-02 ~]$ sysctl -p /etc/sysctl.conf
net.ipv6.conf.all.disable_ipv6 = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
net.ipv6.conf.all.disable_ipv6 = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.swappiness = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
vm.swappiness = 1
vm.overcommit_memory = 1
(07.08.2024 18:04:15)[root@big-tdp-220-02 ~]$
```

Figure 2 - Change RAM default

3. Disable Transparent Huge Pages (THP):

Terminal input

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled;
echo never > /sys/kernel/mm/transparent_hugepage/defrag;
echo "echo never > /sys/kernel/mm/transparent_hugepage/enabled" >>
/etc/rc.local;
echo "echo never > /sys/kernel/mm/transparent_hugepage/defrag" >>
/etc/rc.local;
```

© Copyright Tecnisys 2026. All rights reserved.

NOTE

Many Linux distributions offer THP as a low-complexity option to increase the size of memory blocks/pages (from 4KB to 2MB or 1GB) and to enable the management of many gigabytes, and even terabytes, of RAM. However, workloads in *Big Data* environments often perform poorly with THP enabled, because they tend to have sparse rather than contiguous memory access patterns, thus overloading the CPU.

```
## Disabling the Transparent Huge Pages (THP)
(07.08.2024 18:04:53)[root@big-tdp-220-02 ~]$ echo never > /sys/kernel/mm/transparent_hugepage/enabled
(07.08.2024 18:05:03)[root@big-tdp-220-02 ~]$ echo never > /sys/kernel/mm/transparent_hugepage/defrag
(07.08.2024 18:05:11)[root@big-tdp-220-02 ~]$ echo "echo never > /sys/kernel/mm/transparent_hugepage/enabled" >> /etc/rc.local
(07.08.2024 18:05:19)[root@big-tdp-220-02 ~]$ echo "echo never > /sys/kernel/mm/transparent_hugepage/defrag" >> /etc/rc.local
(07.08.2024 18:05:21)[root@big-tdp-220-02 ~]$
```

Figure 3 - THP disable

Time Synchronization

Instructions

1. Install a time synchronization service (NTP) to ensure that the date and time of the machines are always synchronized, preventing inconsistencies in data and services. In this example, we will use Chrony:

Terminal input

```
yum install chrony -y
systemctl enable chronyd
systemctl start chronyd
```

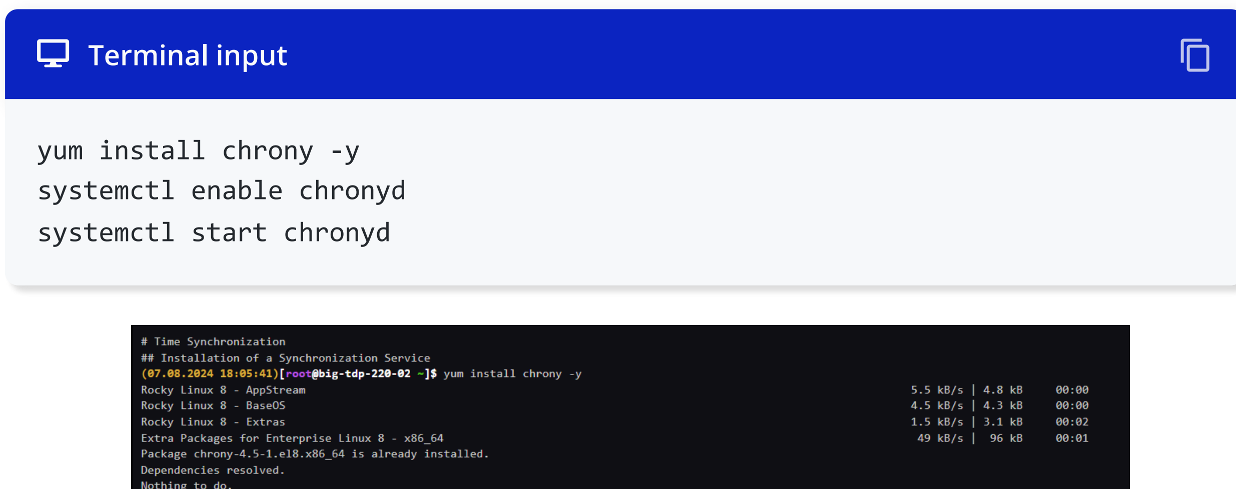


Figure 4 - Install time synchronization

2. Configure Chrony to synchronize with appropriate NTP servers:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
vi /etc/chrony.conf

# Add or adjust the NTP servers as needed

server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
```

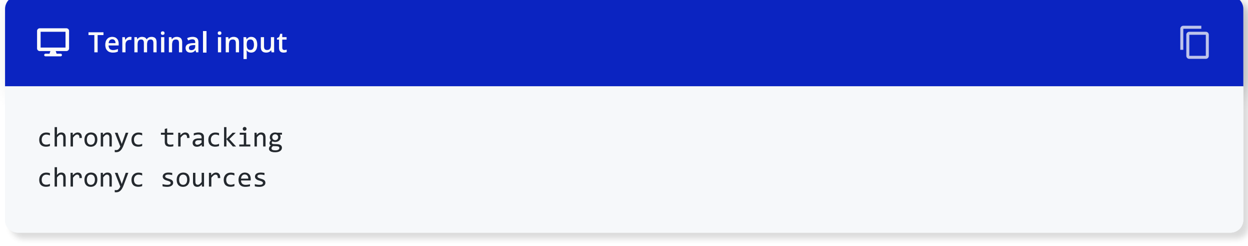
> **NOTE**
>
> To save and exit the vi editor, press ESC, type :wq, and press Enter.

3. Restart the service to apply the new configurations:

Terminal input

```
systemctl restart chronyd
```

4. Check the synchronization status:



Terminal input

```
chronyc tracking
chronyc sources
```

© Copyright Tecnisys 2026. All rights reserved.

```
## Chrony Configuration for Synchronization with Appropriate NTP Servers
(07.08.2024 18:06:24)[root@big-tdp-220-02 ~]$ vi /etc/chrony.conf
===> Add or adjust the NTP servers as needed, then save and exit the editor  (<ESC>:wq<ENTER><===)

server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
(07.08.2024 18:06:24)[root@big-tdp-220-02 ~]$

#Service Restart to Apply the New Configurations
(07.08.2024 18:08:13)[root@big-tdp-220-02 ~]$ systemctl restart chronyd
(07.08.2024 18:08:14)[root@big-tdp-220-02 ~]$

## Verification of the Synchronization Status
(07.08.2024 18:08:29)[root@big-tdp-220-02 ~]$ chronyc tracking
Reference ID    : 92A43042 (cortex.pads.ufrj.br)
Stratum         : 2
Ref time (UTC)  : Wed Aug 07 18:08:25 2024
System time     : 0.000000526 seconds slow of NTP time
Last offset     : +0.001021679 seconds
RMS offset      : 0.001021679 seconds
Frequency       : 501.882 ppm slow
Residual freq   : +128.144 ppm
Skew            : 2.026 ppm
Root delay      : 0.036760874 seconds
Root dispersion : 0.003537497 seconds
Update interval : 1.0 seconds
Leap status     : Normal
(07.08.2024 18:08:37)[root@big-tdp-220-02 ~]$ chronyc sources
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^+ lrtest2.ntp.ifsc.usp.br       2   6    17    12  +1552us[+1552us] +/-   20ms
^- a.st1.ntp.br                  1   6    17    12  -1435us[-1435us] +/-   16ms
^- gps.nu.ntp.br                 1   6    17    13   +604us[ +604us] +/-   14ms
^* cortex.pads.ufrj.br           1   6    17    14  +1043us[+2065us] +/-   20ms
^+ b.ntp.netplanety.com.br       2   6    17    14  +2778us[+3800us] +/-   49ms
^- sa-north-1.clearnet.pw        2   6    17    13  +2083us[+2083us] +/-   39ms
^- time.cloudflare.com           3   6    17    13    +11ms[  +11ms] +/-   74ms
(07.08.2024 18:08:40)[root@big-tdp-220-02 ~]$
```

Figure 5 - Configure Chrony

© Copyright Tecnisys 2026. All rights reserved.

Installation Packages

All TDP installation packages, including direct dependencies, are available on the Tecnisys Public Package Repository. To access them, use the login credentials you set during your *free* registration on the Tecnisys website.

NOTE

In a deployment with limited or restricted Internet access, it is possible to download the necessary files and create a local package repository. To do this, follow the steps below. Otherwise, proceed to Download the Installation Script.

Downloading the Packages

Instructions

To download the installation packages, follow these steps:

1. Register for free on the Tecnisys website;
2. Access the Tecnisys Public Package Repository;
3. Click the *Sign out* button located in the upper right corner of the page;
4. Enter your login credentials (the same as for the Tecnisys website);
5. Click the *Browse* option located in the left-hand menu; and
6. In the central navigation area, access the desired directory to download the files.

© Copyright Tecnisys 2026. All rights reserved.

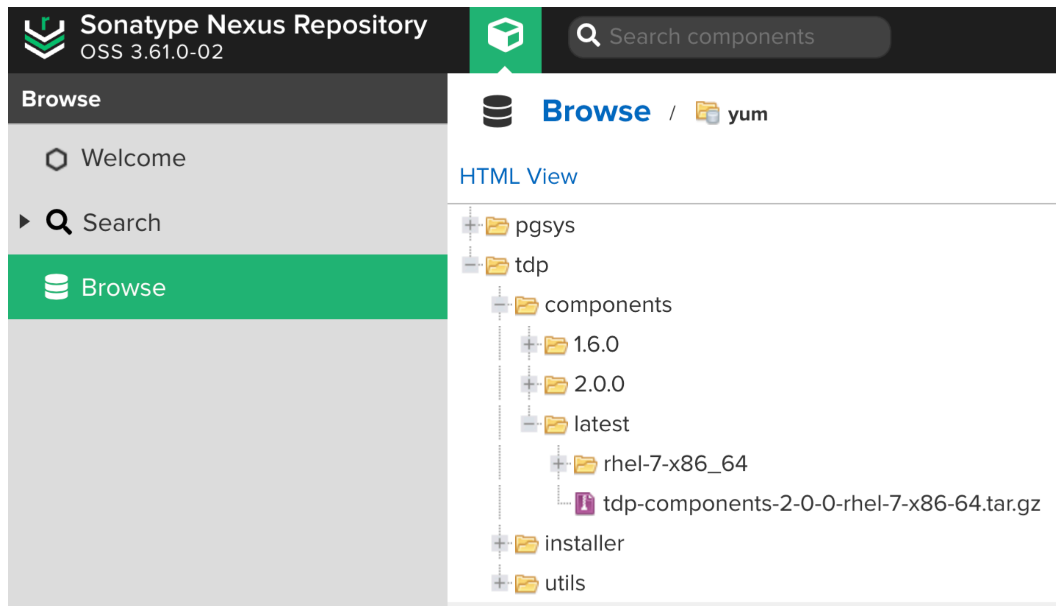


Figure 1 - Tecnisys Public Package Repository

It is also possible to download the packages directly from the terminal, provided the access credentials are included in the request:

- Example using curl

Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/tdp-ambari-2-2-el-9-x86-
64.tar.gz -o tdp-ambari-2-2-el-9-x86-64.tar.gz
```

Terminal input

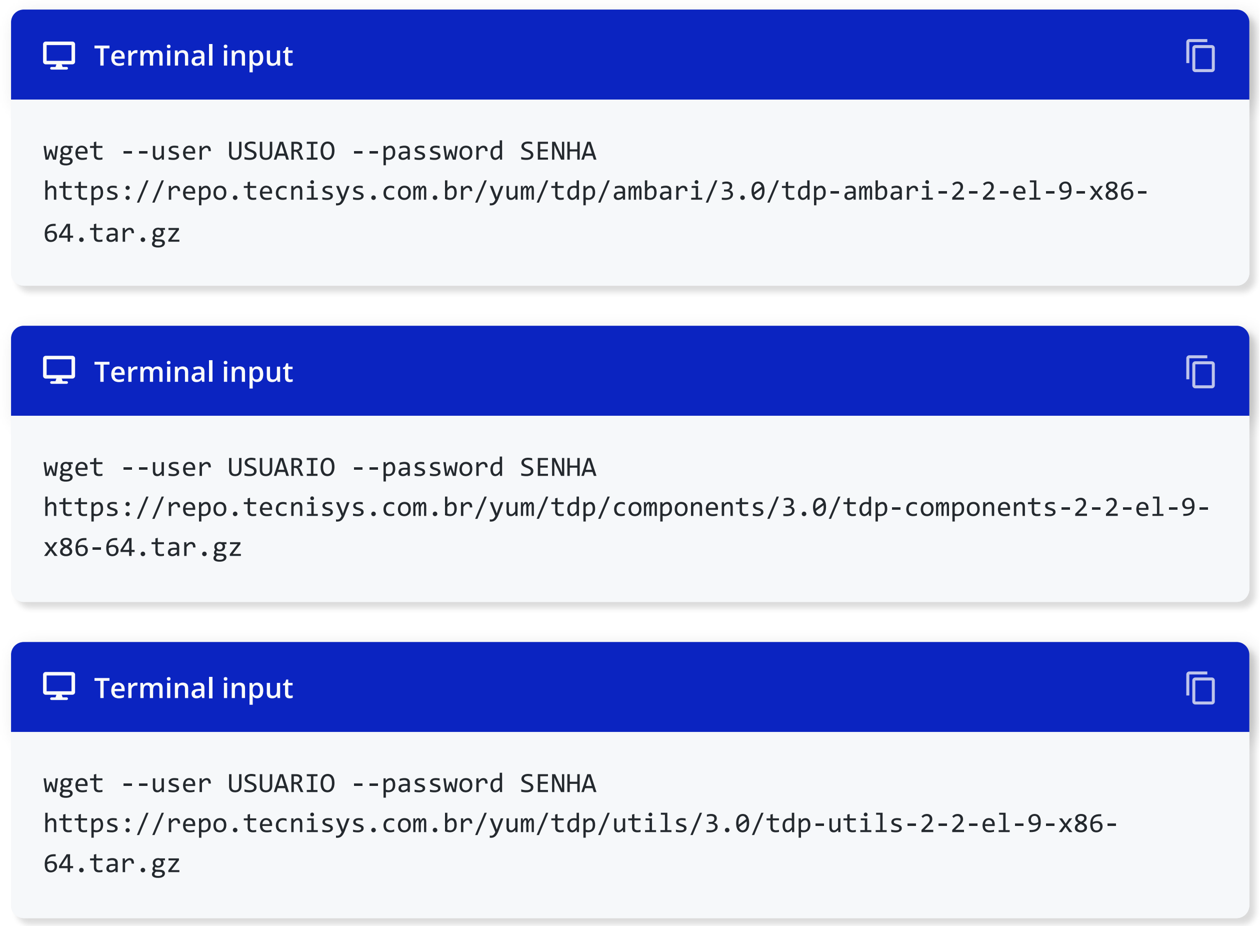
```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/components/3.0/tdp-components-2-2-el-9-
x86-64.tar.gz -o tdp-components-2-2-el-9-x86-64.tar.gz
```

Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/utils/3.0/tdp-utils-2-2-el-9-x86-
64.tar.gz -o tdp-utils-2-2-el-9-x86-64.tar.gz
```

© Copyright Tecnisys 2026. All rights reserved.

- Example using wget



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/tdp-ambari-2-2-el-9-x86-64.tar.gz
```

Terminal input

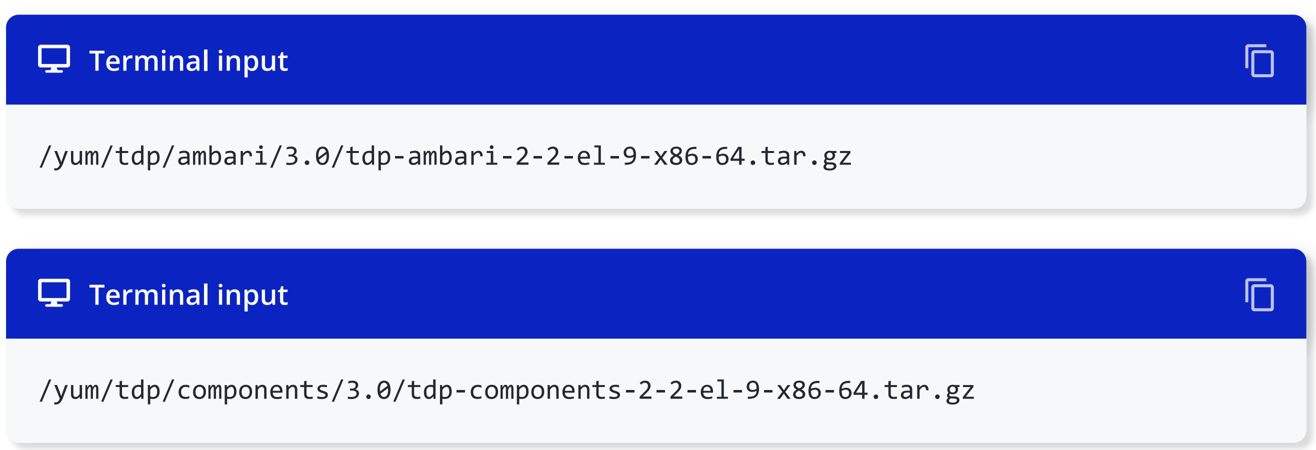
```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/components/3.0/tdp-components-2-2-el-9-x86-64.tar.gz
```

Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/utils/3.0/tdp-utils-2-2-el-9-x86-64.tar.gz
```

Remember to replace *USERNAME* and *PASSWORD* with your registered username and password on the Tecnisys website.

All installation packages for components and utilities can be downloaded at once via compressed files (tar.gz). For example:



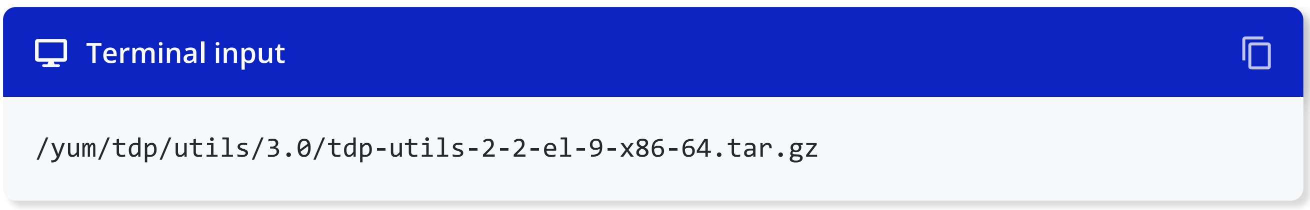
Terminal input

```
/yum/tdp/ambari/3.0/tdp-ambari-2-2-el-9-x86-64.tar.gz
```

Terminal input

```
/yum/tdp/components/3.0/tdp-components-2-2-el-9-x86-64.tar.gz
```

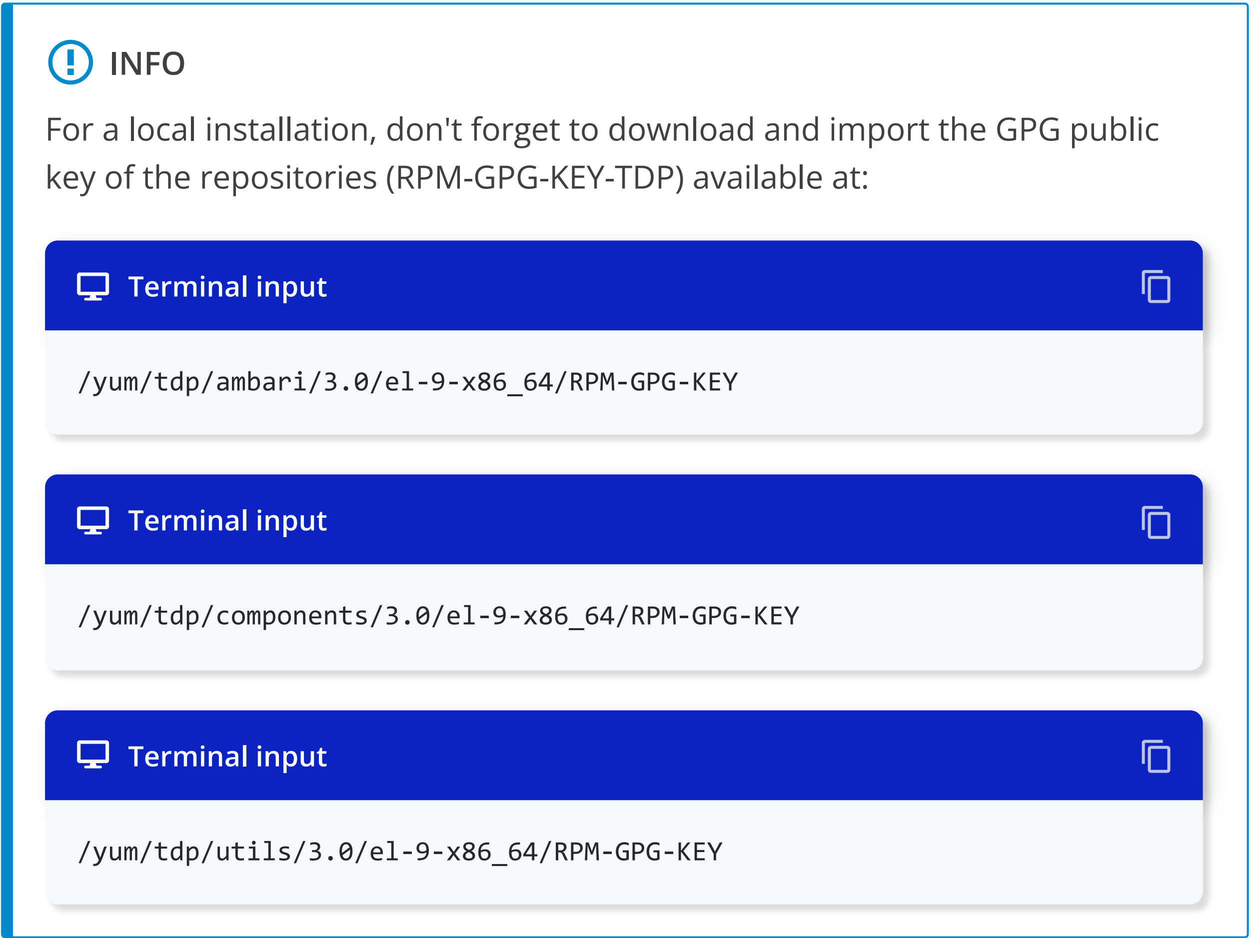
© Copyright Tecnisys 2026. All rights reserved.



Terminal input

```
/yum/tdp/utils/3.0/tdp-utils-2-2-el-9-x86-64.tar.gz
```

> **TIP**
>
> You can verify the integrity of the file by checking the Checksum attributes (sha512, sha256, sha1, and md5) displayed when you click on it.



> **INFO**
>
> For a local installation, don't forget to download and import the GPG public key of the repositories (RPM-GPG-KEY-TDP) available at:
>
> Terminal input
>
> ```
> /yum/tdp/ambari/3.0/el-9-x86_64/RPM-GPG-KEY
> ```
>
> Terminal input
>
> ```
> /yum/tdp/components/3.0/el-9-x86_64/RPM-GPG-KEY
> ```
>
> Terminal input
>
> ```
> /yum/tdp/utils/3.0/el-9-x86_64/RPM-GPG-KEY
> ```

The compressed files (tar.gz) of the installation packages for components and utilities already include the GPG public key.

Creating a Local Package Repositorio

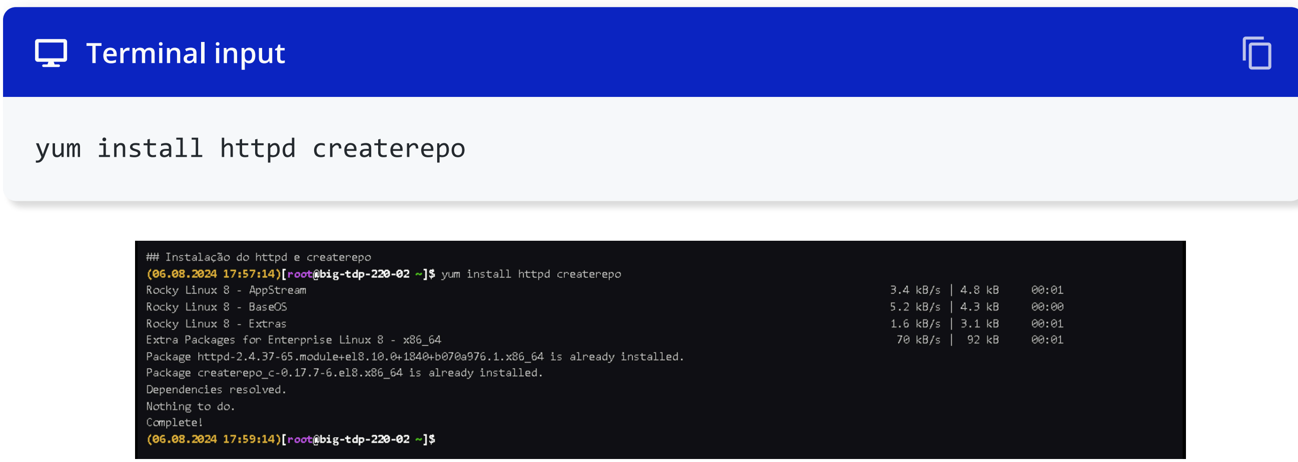
Instructions

© Copyright Tecnisys 2026. All rights reserved.

In a deployment without Internet access, or with restricted access, first download the installation packages through a machine with Internet access, and transfer those files to your *Big Data* environment.

If a local package repository does not yet exist, it can be created as follows:

1. Install the HTTPD and createrepo tools:

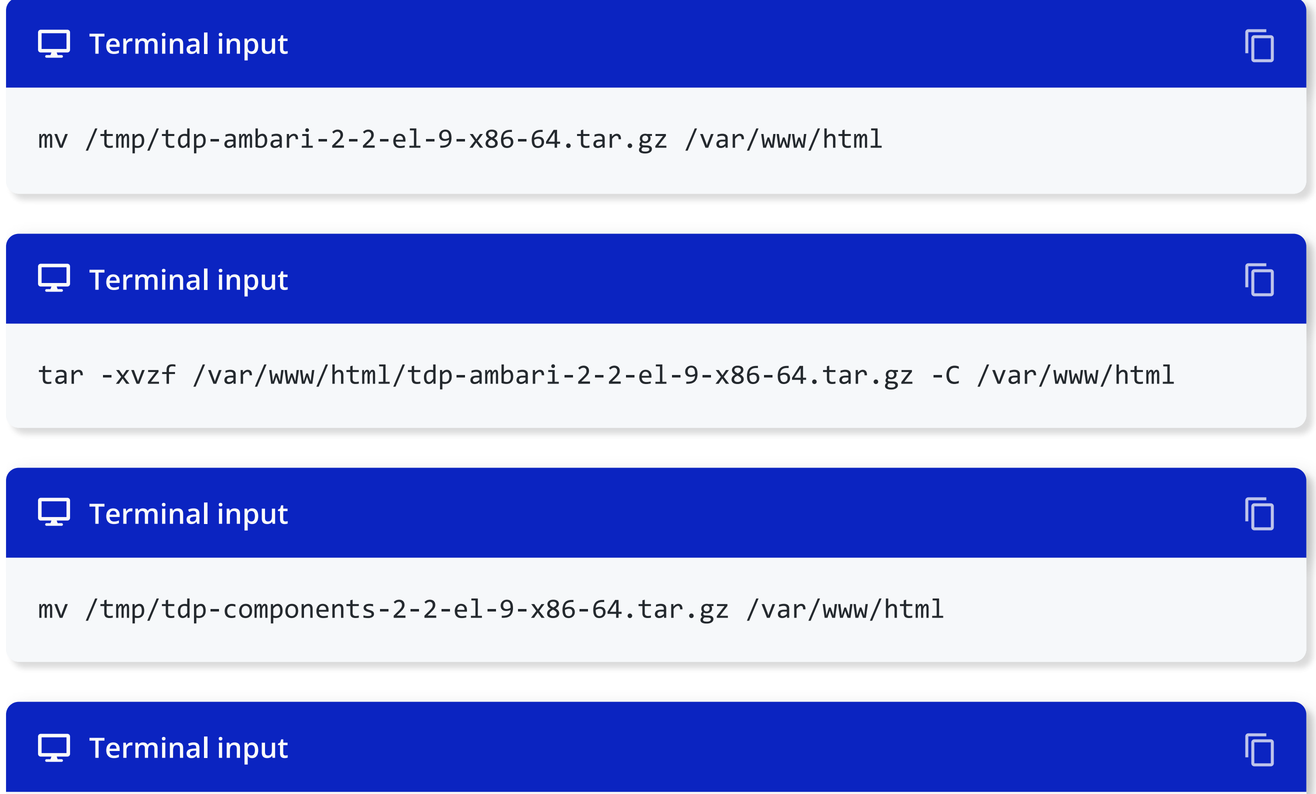


Terminal input

```
yum install httpd createrepo
```


Figure 2 - Createrepo and httpd installation

2. Create the package repository directory (ambari, components, and utils). For example:



Terminal input

```
mv /tmp/tdp-ambari-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-ambari-2-2-el-9-x86-64.tar.gz -C /var/www/html
```

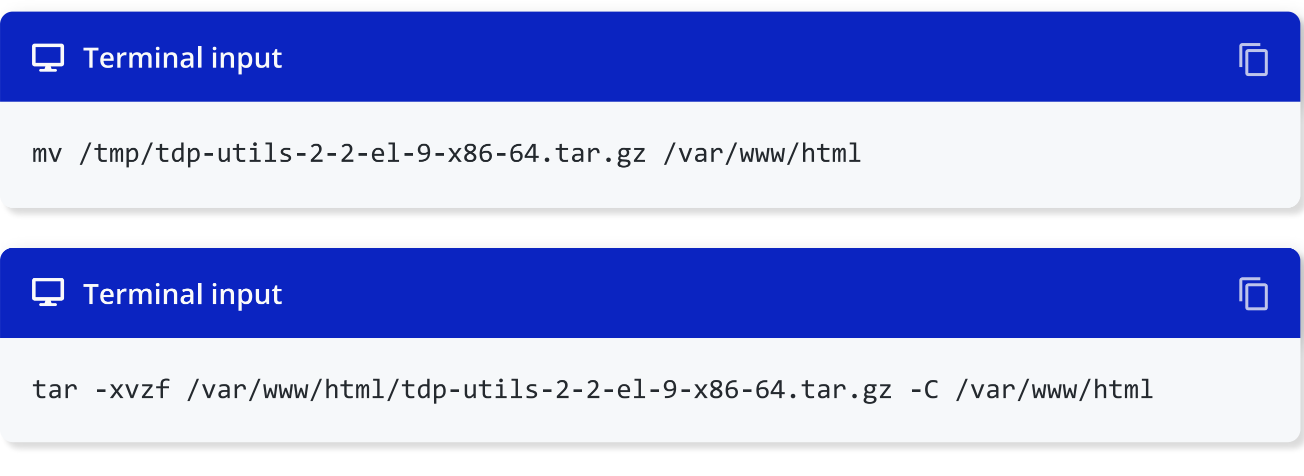
Terminal input

```
mv /tmp/tdp-components-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
tar -xvzf /var/www/html/tdp-components-2-2-el-9-x86-64.tar.gz -C /var/www/html
```



Terminal input

```
mv /tmp/tdp-utils-2-2-el-9-x86-64.tar.gz /var/www/html
```

Terminal input

```
tar -xvzf /var/www/html/tdp-utils-2-2-el-9-x86-64.tar.gz -C /var/www/html
```

```
## Criação do diretório do repositório de pacotes
(06.08.2024 17:57:37)[root@big-tdp-220-02 ~]$ mv /tmp/tdp-ambari-2-2-0-el-9-x86-64.tar.gz /var/www/html
(06.08.2024 17:57:41)[root@big-tdp-220-02 ~]$ tar -xvzf /var/www/html/tdp-ambari-2-2-0-el-9-x86-64.tar.gz -C /var/www/html
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/RPM-GPG-KEY/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/RPM-GPG-KEY/RPM-GPG-KEY-TDP
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/ambari-agent-3.0.0.0-1.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/ambari-server-3.0.0.0-1.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/ambari-server-spi-3.0.0.0-1.noarch.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/md5/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/md5/ambari-agent-3.0.0.0-1.x86_64.rpm.md5
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/md5/ambari-server-3.0.0.0-1.x86_64.rpm.md5
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/ambari/md5/ambari-server-spi-3.0.0.0-1.noarch.rpm.md5
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/postgresql/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/postgresql/postgresql14-14.7-1PGDG.rhel9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/postgresql/postgresql14-devel-14.7-1PGDG.rhel9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/postgresql/postgresql14-libs-14.7-1PGDG.rhel9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/postgresql/postgresql14-server-14.7-1PGDG.rhel9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/tdp-select/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/tdp-select/tdp-select-0.3-1.el9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/tdp-select/tdp-conf-select-0.3-1.el9.x86_64.rpm
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/tecnisys-support-agent/
repo/yum/tdp/ambari/2.2.0/el-9-x86_64/tecnisys-support-agent/tecnisys-support-agent-1.0.0-0.x86_64.rpm
(06.08.2024 17:58:36)[root@big-tdp-220-02 ~]$ mv /tmp/tdp-components-2-2-0-el-9-x86-64.tar.gz /var/www/html
(06.08.2024 17:58:39)[root@big-tdp-220-02 ~]$ tar -xvzf /var/www/html/tdp-components-2-2-0-el-9-x86-64.tar.gz -C /var/www/html
repo/yum/tdp/components/2.2.0/el-9-x86_64/
repo/yum/tdp/components/2.2.0/el-9-x86_64/RPM-GPG-KEY/
repo/yum/tdp/components/2.2.0/el-9-x86_64/RPM-GPG-KEY/RPM-GPG-KEY-TDP
repo/yum/tdp/components/2.2.0/el-9-x86_64/ambari-metrics/
repo/yum/tdp/components/2.2.0/el-9-x86_64/ambari-metrics/ambari-metrics-hadoop-sink-3.1.0-1.x86_64.rpm
repo/yum/tdp/components/2.2.0/el-9-x86_64/ambari-metrics/ambari-metrics-monitor-3.1.0-1.x86_64.rpm
repo/yum/tdp/components/2.2.0/el-9-x86_64/ambari-metrics/ambari-metrics-collector-3.1.0-1.x86_64.rpm
repo/yum/tdp/components/2.2.0/el-9-x86_64/ambari-metrics/ambari-metrics-grafana-3.1.0-1.x86_64.rpm
...
(06.08.2024 18:15:07)[root@big-tdp-220-02 ~]$ mv /tmp/tdp-utils-2-2-0-el-9-x86-64.tar.gz /var/www/html
(06.08.2024 18:15:11)[root@big-tdp-220-02 ~]$ tar -xvzf /var/www/html/tdp-utils-2-2-0-el-9-x86-64.tar.gz -C /var/www/html
repo/yum/tdp/utils/2.2.0/el-9-x86_64/
repo/yum/tdp/utils/2.2.0/el-9-x86_64/redhat-lsb-core-4.1-56.el9.x86_64.rpm
...
(06.08.2024 18:15:20)[root@big-tdp-220-02 ~]$
```

Figure 3 - Repository Creation of repository directory

3. Create the repository package index:

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/ambari/3.0/el-9-x86_64/
```

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
createrepo /var/www/html/repo/yum/tdp/components/3.0/el-9-x86_64/
```

Terminal input

```
createrepo /var/www/html/repo/yum/tdp/utils/3.0/el-9-x86_64/
```

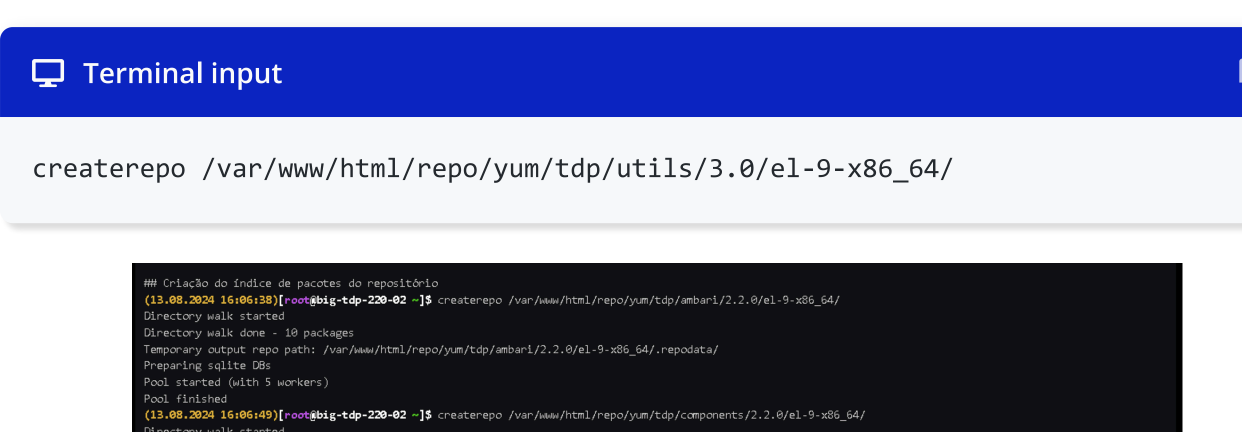


Figure 4 - Creating the package index

4. Add the repository alias in the HTTPD service configuration file:

Terminal input

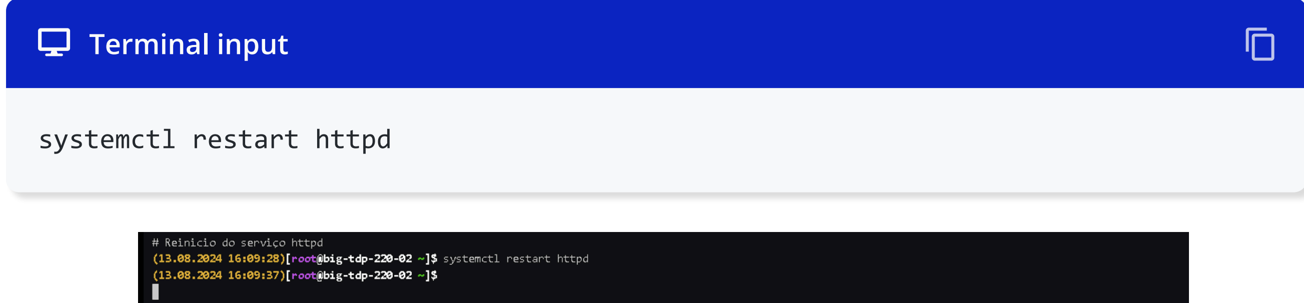
```
cat << EOF   /etc/httpd/conf/httpd.conf
Alias /repo /var/www/html/repo
  <Directory /var/www/html/repo>
      Options Indexes FollowSymLinks
      AllowOverride None
      Require all granted
  </Directory>
  EOF
```

© Copyright Tecnisys 2026. All rights reserved.

```
## Adição do alias do repositório no arquivo de configuração do serviço HTTPD
(13.08.2024 16:08:50)[root@big-tdp-220-02 ~]$ cat << EOF   /etc/httpd/conf/httpd.conf
> Alias /repo /var/www/html/repo
> <Directory /var/www/html/repo>
>     Options Indexes FollowSymLinks
>     AllowOverride None
>     Require all granted
> </Directory>
> EOF
...
#
# EnableMMAP and EnableSendfile: On systems that support it,
# memory-mapping or the sendfile syscall may be used to deliver
# files.  This usually improves server performance, but must
# be turned off when serving from networked-mounted
# filesystems or if support for these functions is otherwise
# broken on your system.
# Defaults if commented: EnableMMAP On, EnableSendfile Off
#
#EnableMMAP off
EnableSendfile on

# Supplemental configuration
#
# Load config files in the "/etc/httpd/conf.d" directory, if any.
IncludeOptional conf.d/*.conf
(13.08.2024 16:08:59)[root@big-tdp-220-02 ~]$
```

Figure 5 - Add repository alias to HTTPD configuration file

5. Restart the HTTPD service:



Terminal input

```
systemctl restart httpd
```

```
# Reinício do serviço httpd
(13.08.2024 16:09:28)[root@big-tdp-220-02 ~]$ systemctl restart httpd
(13.08.2024 16:09:37)[root@big-tdp-220-02 ~]$
```

Figure 6 - Service restart

Confirm the availability and organization of the installation packages by accessing the address http://localhost/repo. Replace 'localhost' with the IP or *hostname* of the package repository machine if accessing from another machine.

Download the Installation Script

Instructions

We have developed a customizable shell script to assist you in the first stage of the *Big Data Cluster* deployment process, the installation of the Apache Ambari component.

NOTE

We recommend that the installation of Apache Ambari (Ambari Server and Ambari Web) be performed on a *Utility* or *Edge* type machine.

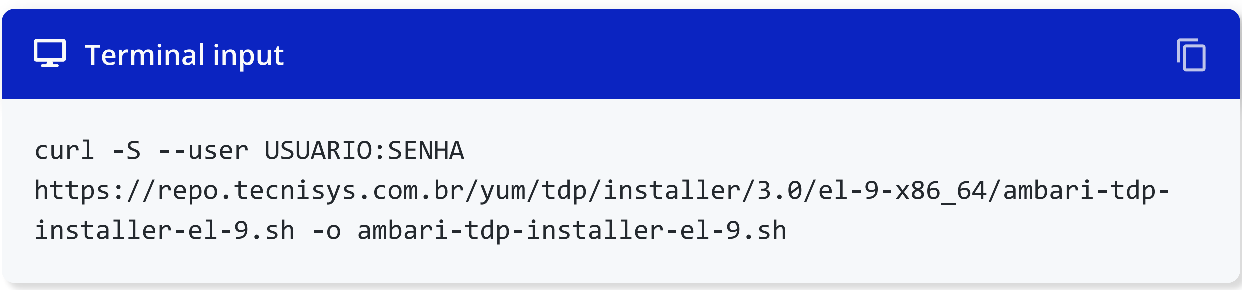
© Copyright Tecnisys 2026. All rights reserved.

This script is available in the `/yum/tdp/installer` directory of the Tecnisys Public Package Repository.

Initially, the script should be downloaded onto the machine where the Ambari Server will be installed. However, if the registration of the machines that will compose the *Big Data Cluster* is to be done manually (without SSH key exchange), as described in Host Registration, it is recommended that the script be downloaded onto all machines in the *Cluster* for the installation of the Ambari Agent.

The installation script can be downloaded directly from the terminal, provided the access credentials are included in the request:

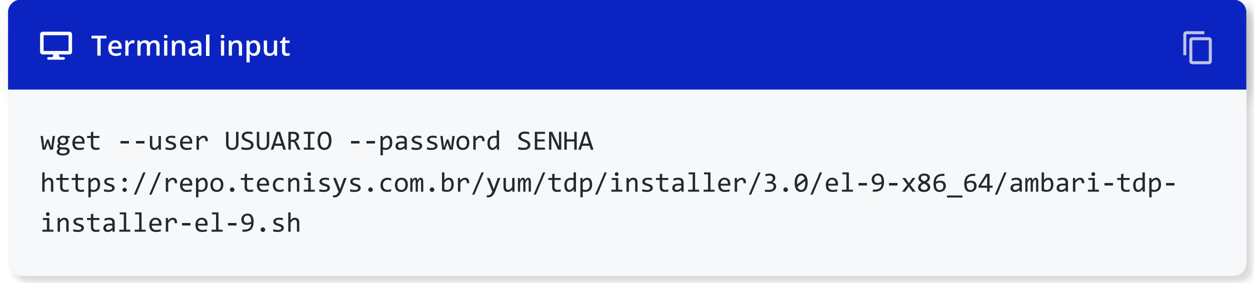
- Example using curl



Terminal input

```
curl -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh -o ambari-tdp-installer-el-9.sh
```

- Example using wget



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh
```

Remember to replace *USERNAME* and *PASSWORD* with your registered username and password on the Tecnisys website.

© Copyright Tecnisys 2026. All rights reserved.

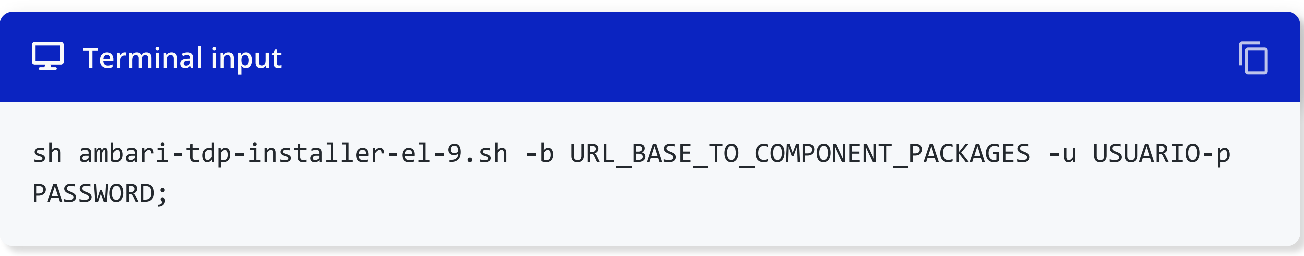
Apache Ambari Installation

Once the minimum requirements are met, all machines in the environment are prepared, and the installation packages are available, we proceed to the installation of the Apache Ambari service.

Once installed, Apache Ambari will be responsible for creating the *Big Data Cluster*, either through its web page or via REST API.

Installing the Ambari Server

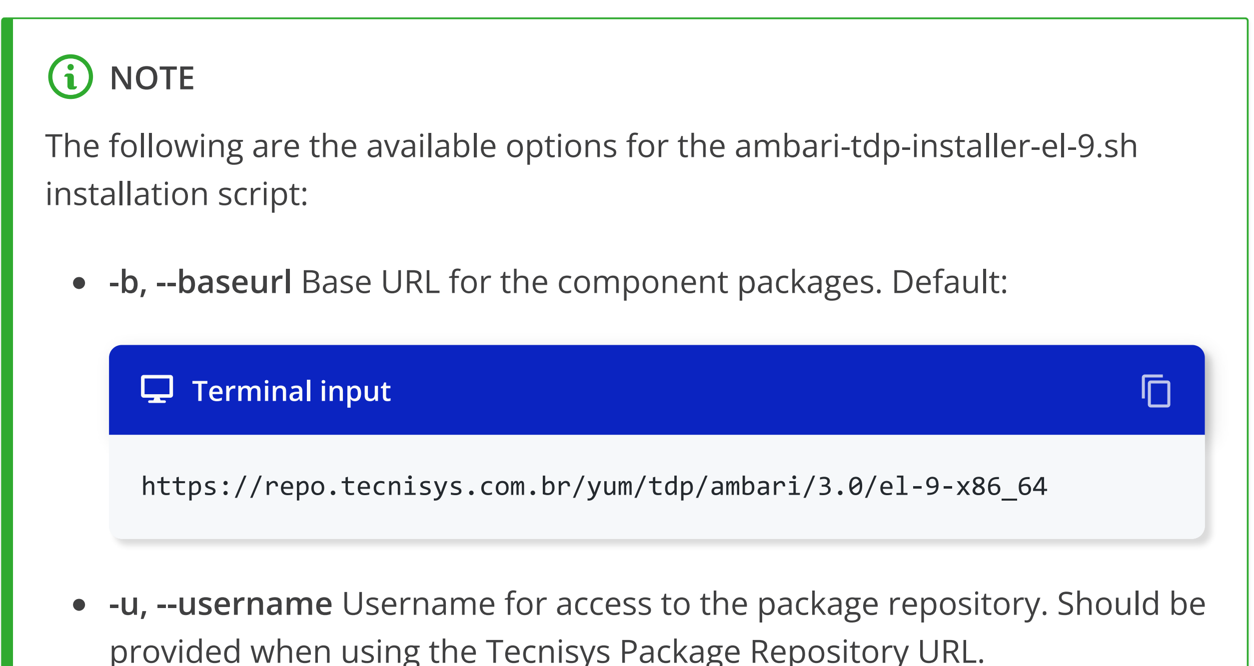
Run the command below to install the Ambari Server via the installation script on a *utility* or *edge* machine [1]:



Terminal input

```
sh ambari-tdp-installer-el-9.sh -b URL_BASE_TO_COMPONENT_PACKAGES -u USUARIO-p
PASSWORD;
```

The USERNAME (-u) and PASSWORD (-p) parameters are optional and should only be provided when the package repository requires access credentials, as is the case with the Tecnisys Pacage Public Repository.



NOTE

The following are the available options for the ambari-tdp-installer-el-9.sh installation script:

- **-b, --baseurl** Base URL for the component packages. Default:

  Terminal input

  ```
  https://repo.tecnisys.com.br/yum/tdp/ambari/3.0/el-9-x86_64
  ```

- **-u, --username** Username for access to the package repository. Should be provided when using the Tecnisys Package Repository URL.

© Copyright Tecnisys 2026. All rights reserved.

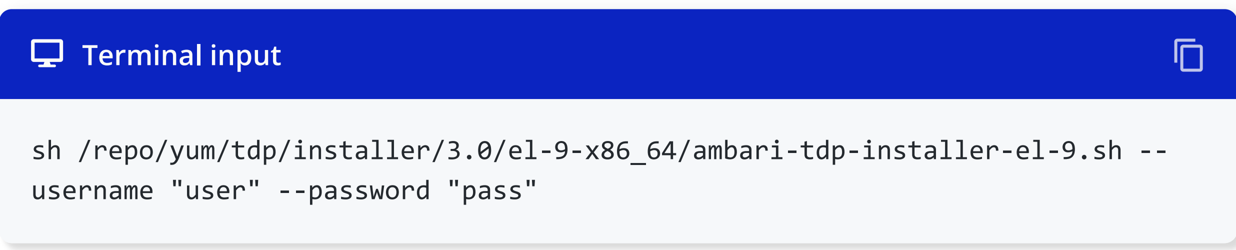
- **-p, --password** Password for access to the package repository. Should be provided when using the Tecnisys Package Repository URL.
- **-c, --component (agent | server | all)** [default: `server`] Apache Ambari component to be installed.

Ambari Server Installation Scenarios

Instructions

Below, we provide examples for different scenarios:

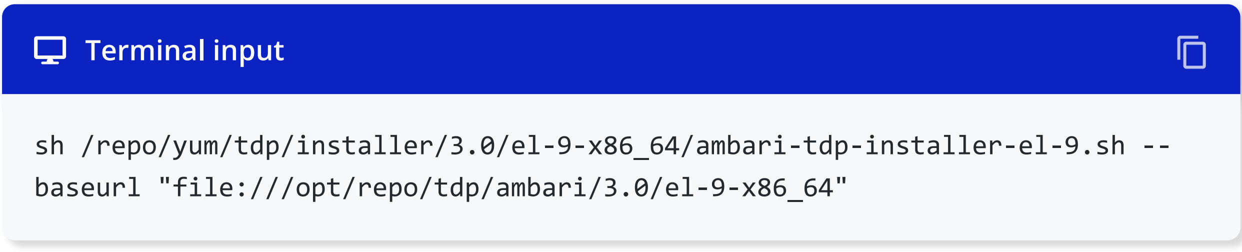
- **Installation using the Tecnisys Public Package Repository:**



```
sh /repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --username "user" --password "pass"
```

The access credentials *(username and password)* are set when you register for free on the Tecnisys Site.

- **Installation using a local package repository:**



```
sh /repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --baseurl "file:///opt/repo/tdp/ambari/3.0/el-9-x86_64"
```

- **Installation using a local package repository on the machine with IP 192.168.32.100 and accessible via HTTP:**

```
/repo/yum/tdp/installer/3.0/el-9-x86_64/ambari-tdp-installer-el-9.sh --baseurl "https://192.168.32.100/tdp/ambari/3.0/el-9-x86_64"
```

© Copyright Tecnisys 2026. All rights reserved.

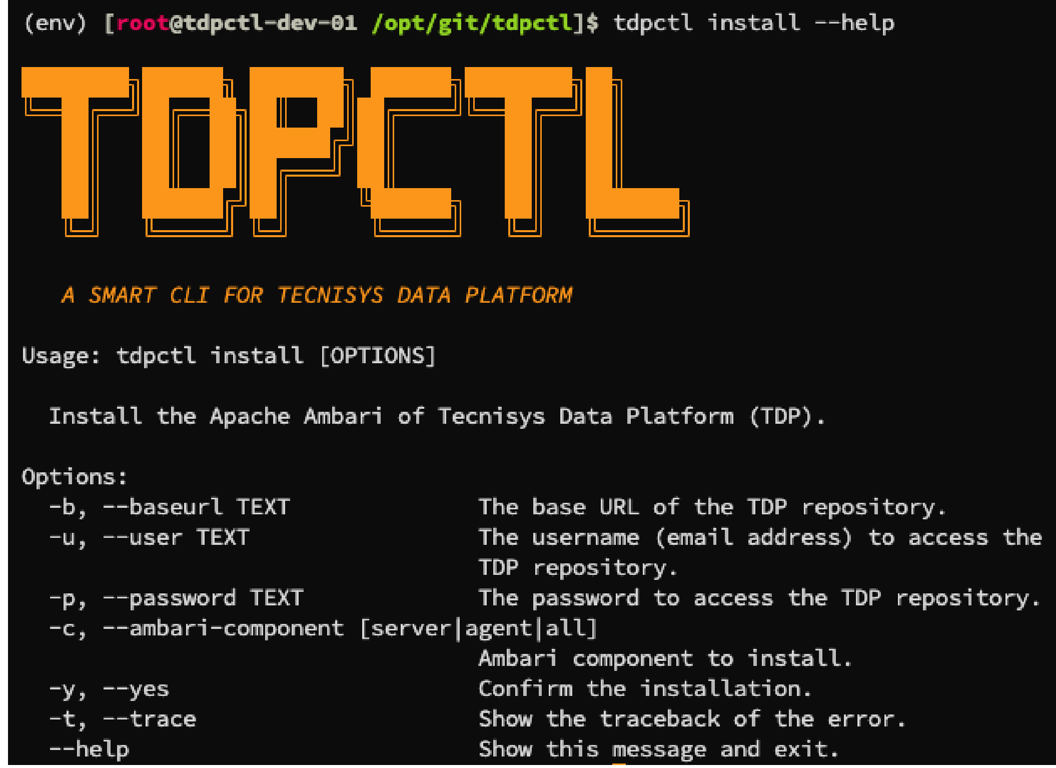
Installation via tdpctl

Ambari Server installation can also be performed through tdpctl, the TDP Command Line Interface:

Terminal input

```
tdpctl install --help
```

The available options, as well as the information required to install Ambari Server, are the same as those requested by the installation script.



```
(env) [root@tdpctl-dev-01 /opt/git/tdpctl]$ tdpctl install --help
TDPCTL

  A SMART CLI FOR TECNISYS DATA PLATFORM

Usage: tdpctl install [OPTIONS]

  Install the Apache Ambari of Tecnisys Data Platform (TDP).

Options:
  -b, --baseurl TEXT              The base URL of the TDP repository.
  -u, --user TEXT                 The username (email address) to access the
                                  TDP repository.
  -p, --password TEXT             The password to access the TDP repository.
  -c, --ambari-component [server|agent|all]
                                  Ambari component to install.
  -y, --yes                       Confirm the installation.
  -t, --trace                     Show the traceback of the error.
  --help                          Show this message and exit.
```

Figure 1 - Options for the tdpctl install command

Note that it is possible to inform the desired component using the -c or --component (agent | server | all) option.

© Copyright Tecnisys 2026. All rights reserved.

Figure 2 - Ambari Server installation via tdpctl

To install tdpctl, follow the guidance available here.

Configurations

Instructions

Configuring the JDBC Driver

Configure the JDBC driver to be used by the Ambari Server to connect to the metadata database:

© Copyright Tecnisys 2026. All rights reserved.

> **TIP**
>
> The PostgreSQL JDBC driver is also available in the public/ambari/3.0.0.0 directory of the Tecnisys Package Repository.

Figure 3 - JDBC Configuration

Configuring the Ambari Server

Configure the Ambari Server:

Terminal input

```
ambari-server setup
```

Next, provide the requested information at the presented prompt:

1- If SELinux is active, a warning will be displayed. Confirm if SELinux can be switched to permissive mode and temporarily disabled. Default (y)

2- Confirm if you want to use a custom OS user for the Ambari Server daemon. Default (n)

> (a) - If yes (y), provide the desired user.

3- If the Firewall (iptables) is active, a warning will be displayed. Confirm if the ports used by Ambari are accessible. Default (y)

> (a) - If there is a port conflict, an error will be displayed, and the configuration will be aborted.

4- Choose the JDK to be used. Default (1) *Oracle JDK 1.8 + Java Cryptography Extension (JCE) Policy Files 8* (automatically downloaded from the *public/tdp/2.0.0/rhel-7-x86_64* directory of the `Tecnisys Package Repository.

© Copyright Tecnisys 2026. All rights reserved.

> (a) - If option (1) is chosen, confirm if you accept the usage license. Default (y)

> (b) - Choose option (2) to provide the JAVA_HOME path of a different JDK.

5- Confirm if you want the Ambari Server to download and install additional LZO compression packages. Default (n)

6- Confirm if you want to proceed with the Advanced Database Configuration. Default (n)

> (a) - If no (n), Ambari automatically initializes a PostgreSQL instance (an installation dependency of the Ambari Server), creates its own metadata database, schema, user, metadata tables, etc.

> (b) - If yes (y), connection information for the database will be requested, such as the Database Management System (DBMS), hostname, port, and Ambari database name, so that the Ambari Server can connect and automatically create its metadata tables and other structures.

© Copyright Tecnisys 2026. All rights reserved.

```
# Ambari Server Configuration
(14.08.2024 19:39:00)[root@big-tdp-220-02 ~]$ ambari-server setup
Using python
Setup ambari-server
Checking SELinux...
SELinux status is 'disabled'
SELinux status is 'disabled'
Customize user account for ambari-server daemon [y/n] (n)? n
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Checking JDK...
[1] OpenJDK 64-Bit Server VM (Temurin)(build 25.402-b06, mixed mode)
[2] Custom JDK
==============================================================================
Enter choice (1):
To download the Open JDK Policy Files you must accept the license terms found at https://github.com/openjdk/jdk/blob/master/LICENSE and not accepting will
cancel the Ambari Server setup and you must install the JDK and JCE files manually.
Do you accept the Code License Agreement [y/n] (y)? y
Downloading JDK from https://repo.tecnisys.com.br/repository/public/ambari/3.0.0.0/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz to /var/lib/ambari-serve
r/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz... 100% (98.2 MB of 98.2 MB)
Successfully downloaded JDK distribution to /var/lib/ambari-server/resources/OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
Installing JDK to /usr/lib/jvm/
Successfully installed JDK to /usr/lib/jvm/
Downloading JCE Policy archive from https://repo.tecnisys.com.br/public/ambari/3.0.0.0/jce_policy-8.zip to /var/lib/ambari-server/resources/jce_policy-8.zi
p

Successfully downloaded JCE Policy archive to /var/lib/ambari-server/resources/jce_policy-8.zip
Installing JCE policy...
Check JDK version for Ambari Server...
JDK version found: 8
Minimum JDK version is 8 for Ambari. Skipping to setup different JDK for Ambari Server.
Checking GPL software agreement...
GPL License for LZO: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html
Enable Ambari Server to download and install GPL Licensed LZO packages [y/n] (n)? y
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Configuring PostgreSQL...
Restarting PostgreSQL
Creating schema and user...
done.
Creating tables...
done.
Extracting system views...
ambari-admin-3.0.0.0-1.jar
ambari-views-package-3.0.0.0-1.jar
capacity-scheduler-3.0.0.0-1.jar
files-3.0.0.0-1.jar
pig-3.0.0.0-1.jar
wfmanager-3.0.0.0-1.jar

Ambari repo file doesn't contain latest json url, skipping repoinfos modification
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
```

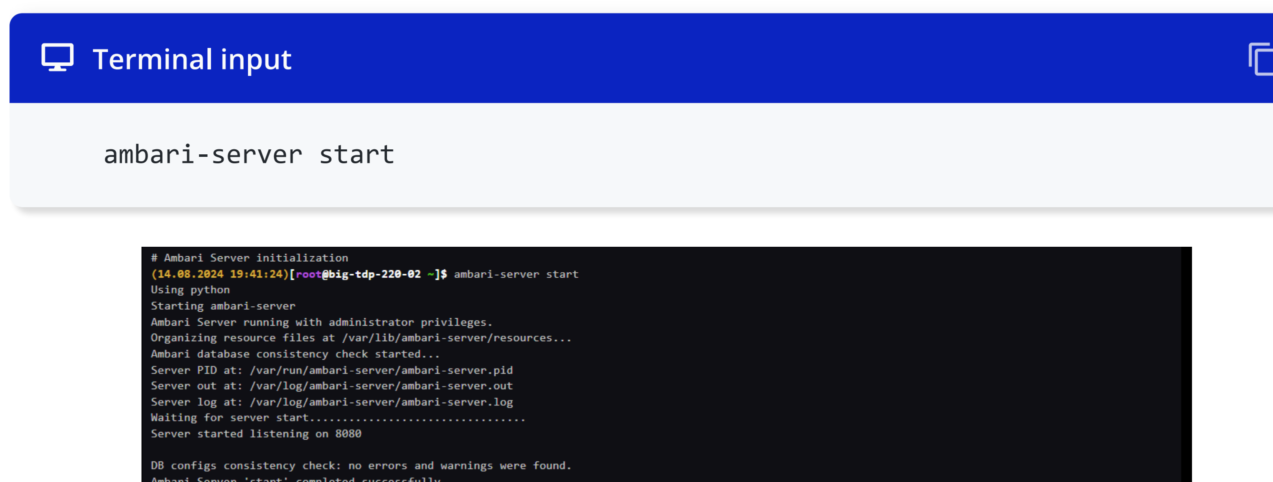
Figure 4 - Ambari Server Configuration

Starting the Ambari Server

Start the Ambari Server daemon:

Terminal input

```
ambari-server start
```



```
# Ambari Server initialization
(14.08.2024 19:41:24)[root@big-tdp-220-02 ~]$ ambari-server start
Using python
Starting ambari-server
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Ambari database consistency check started...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start................................
Server started listening on 8080

DB configs consistency check: no errors and warnings were found.
Ambari Server 'start' completed successfully.
```

Figure 5 - Ambari Server Initialization

© Copyright Tecnisys 2026. All rights reserved.

After the Ambari Server starts, the Ambari web page will be accessible by default on port 8080.

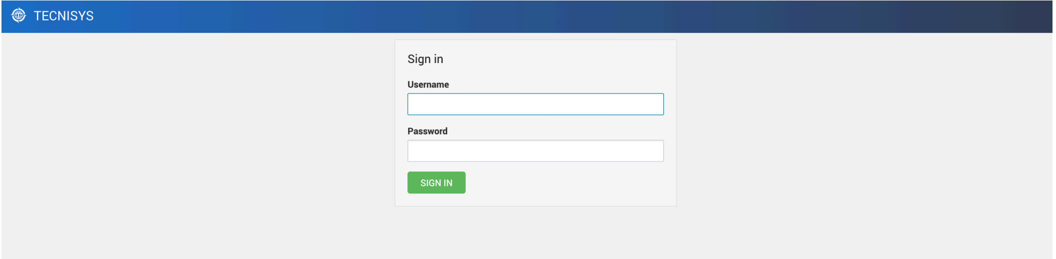


Figure 6 - Ambari Login Page

NOTE

If necessary, the Ambari web interface port can be changed in the `/etc/ambari-server/conf/ambari.properties` file.

Creating Metadata Databases

Many of the services/components of the TDP Platform require databases for metadata persistence.

Below, we provide guidelines for creating these databases in PostgreSQL (the default DBMS for the Ambari Server). If you have chosen a different DBMS for your environment, please refer to its official documentation.

WARNING

The passwords used in the following commands are intended only to exemplify the presented procedure. Please use strong passwords in your environment. Additionally, the user and database names are only suggestions, feel free to change them.

CAUTION

Pay attention to the collation (locale and encoding settings, for example: en_US.UTF-8 and pt_BR.UTF-8) of the metadata databases. Some components

© Copyright Tecnisys 2026. All rights reserved.

may not function correctly or may have issues with displaying, comparing, and sorting data and information if the collation is not configured correctly.

Creating the Apache Airflow Metadata Database

1. Create the airflow user:

Terminal input

```
su -c 'createuser -s -d -l airflow' - postgres;
```

2. Set the airflow user's password:

Terminal input

```
su -c "psql -U postgres -c \\"alter user airflow with password 'airflow'\\""
- postgres;
```

3. Create the airflow database:

Terminal input

```
su -c 'createdb -O airflow airflow' - postgres;
```

Creating the Apache Druid Metadata Database

1. Create the druid user:

Terminal input

```
su -c 'createuser -s -d -l druid' - postgres;
```

2. Set the druid user's password:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
su -c "psql -U postgres -c \\"alter user druid with password 'druid'\\"" -
postgres;
```

3. Create the druid database:

Terminal input

```
su -c 'createdb -O druid druid' - postgres;
```

Creating the Apache Hive Metadata Database

1. Create the hive user:

Terminal input

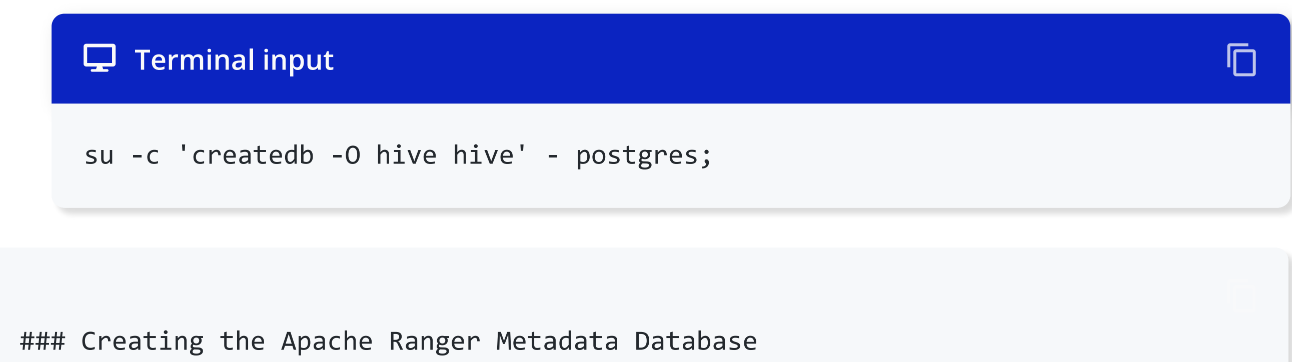
```
su -c 'createuser -s -d -l hive' - postgres;
```

2. Set the hive user's password:

Terminal input

```
su -c "psql -U postgres -c \\"alter user hive with password 'hive'\\"" -
postgres;
```

3. Create the hive database:



Terminal input

```
su -c 'createdb -O hive hive' - postgres;
```

```
### Creating the Apache Ranger Metadata Database

1. Create the ranger user:
```

© Copyright Tecnisys 2026. All rights reserved.

```bash title="Terminal input"
su -c 'createuser -s -d -l ranger' - postgres;
```

2. Set the ranger user's password:

Terminal input

```
su -c "psql -U postgres -c \\"alter user ranger with password 'ranger'\\"" - postgres;
```

3. Create the ranger database:

Terminal input

```
su -c 'createdb -O ranger ranger' - postgres;
```

Creating the Apache RangerKMS Metadata Database

1. Create the rangerkms user:

Terminal input

```
su -c 'createuser -s -d -l rangerkms' - postgres;
```

2. Set the rangerkms user's password:

Terminal input

```
su -c "psql -U postgres -c \\"alter user rangerkms with password 'rangerkms'\\"" - postgres;
```

3. Create the rangerkms database:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
su -c 'createdb -O rangerkms rangerkms' - postgres;
```

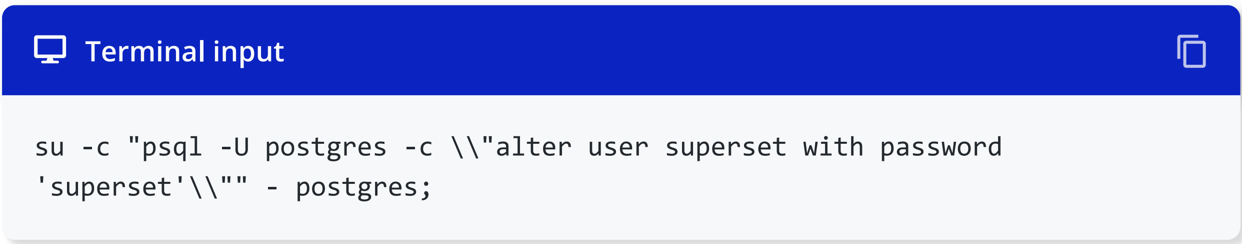
Creating the Apache Superset Metadata Database

1. Create the superset user:

Terminal input

```
su -c 'createuser -s -d -l superset' - postgres;
```

2. Set the superset user's password:



Terminal input

```
su -c "psql -U postgres -c \\"alter user superset with password
'superset'\\"" - postgres;
```

3. Create the superset database:

Terminal input

```
su -c 'createdb -O superset superset' - postgres;
```

```
# Metadata Database Creation - Airflow Example

## Airflow User Creation
(14.08.2024 19:42:27)[root@big-tdp-220-02 ~]$ su -c 'createuser -s -d -l airflow' - postgres
(14.08.2024 19:42:29)[root@big-tdp-220-02 ~]$

## Airflow User Password Setting
(14.08.2024 19:42:44)[root@big-tdp-220-02 ~]$ su -c "psql -U postgres -c \"alter user airflow with password 'airflow'\"" - postgres
ALTER ROLE

(14.08.2024 19:42:48)[root@big-tdp-220-02 ~]$

## Airflow Database Creation
(14.08.2024 19:43:01)[root@big-tdp-220-02 ~]$ su -c 'createdb -O airflow airflow' - postgres
(14.08.2024 19:43:01)[root@big-tdp-220-02 ~]$
```

Figure 7 - Ambari Server Setup - metadata creation

Configuring the Metadata Databases Instance

Below, we provide guidelines and code examples for configuring access rules and properties of PostgreSQL (the default DBMS for the Ambari Server). If you have chosen a different DBMS for your environment, please refer to its official documentation.

© Copyright Tecnisys 2026. All rights reserved.

NOTE

Check, and if necessary, change the major version of PostgreSQL present in the configuration file paths.

1. Add the other databases to the Ambari access rule:

```
sed -i
's/ambari,mapred/airflow,ambari,druid,hive,mapred,oozie,ranger,superset/g'
/var/lib/pgsql/14/data/pg_hba.conf
```

2. Configure the PostgreSQL instance to accept connections from all network interfaces:

```
sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g"
/var/lib/pgsql/14/data/postgresql.conf
```

3. Increase the maximum number of connections supported by the PostgreSQL instance:

```
sed -i "s/max_connections = 100/max_connections = 500/g"
/var/lib/pgsql/14/data/postgresql.conf
```

4. Restart the PostgreSQL service to apply all changes:

```
systemctl restart postgresql-14
```

© Copyright Tecnisys 2026. All rights reserved.

```
# Configuração da Instância dos Bancos de Metadados - exemplo

## Adição dos demais bancos de dados à regra de acesso do Ambari

(14.08.2024 19:50:13)[root@big-tdp-220-02 ~]$ sed -i 's/ambari,mapred/airflow,ambari,druid,hive,mapred,oozie,ranger,superset/g' /var/lib/pgsql/14/data/pg_h
ba.conf

(14.08.2024 19:51:20)[root@big-tdp-220-02 ~]$

## Configuração da instância PostgreSQL para receber conexões de todas as interfaces de rede
(14.08.2024 19:50:37)[root@big-tdp-220-02 ~]$ sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g" /var/lib/pgsql/14/data/postgressql.conf

(14.08.2024 19:51:43)[root@big-tdp-220-02 ~]$

## Incremento do número máximo de conexões suportadas pela instância PostgreSQL
(14.08.2024 19:50:57)[root@big-tdp-220-02 ~]$ sed -i "s/max_connections = 100/max_connections = 500/g" /var/lib/pgsql/14/data/postgresql.conf
(14.08.2024 19:51:04)[root@big-tdp-220-02 ~]$

## Reinicio do serviço do PostgreSQL para que todas as alterações sejam efetivadas
(14.08.2024 19:51:18)[root@big-tdp-220-02 ~]$ systemctl restart postgresql-14
(14.08.2024 19:51:28)[root@big-tdp-220-02 ~]$
(14.08.2024 19:51:30)[root@big-tdp-220-02 ~]$
```

Figure 8 - Metadata Configuration

WARNING

Assess the impact of these configurations on the security and performance of your environment. If you need support, Contact Us, we will be happy to assist you.

Footnotes

1. *Utility* machines are typically used for auxiliary tasks, such as cluster management, while *Edge* machines are dedicated to graphical interfaces or components used by users at the "edge" of the environment. ↩

© Copyright Tecnisys 2026. All rights reserved.

Cluster Creation and Component Installation

After installing the Apache Ambari service, the next step is to create the *Big Data Cluster*, including the installation of the desired services/components.

Let's Get Started! Use a browser to access the Ambari web interface available at the IP/hostname of the Ambari Server machine, port 8080. For example:

```
http://192.168.56.100:8080
```

TECNISYS
Sign in
Username
Password
SIGN IN

Figure 1 - Ambari login page

> **NOTE**
>
> By default, the access username/password are admin / admin, respectively.

Instructions

1. On the first access, a welcome page will be displayed. To start the *Cluster* deployment process, click the LAUNCH INSTALL WIZARD button:

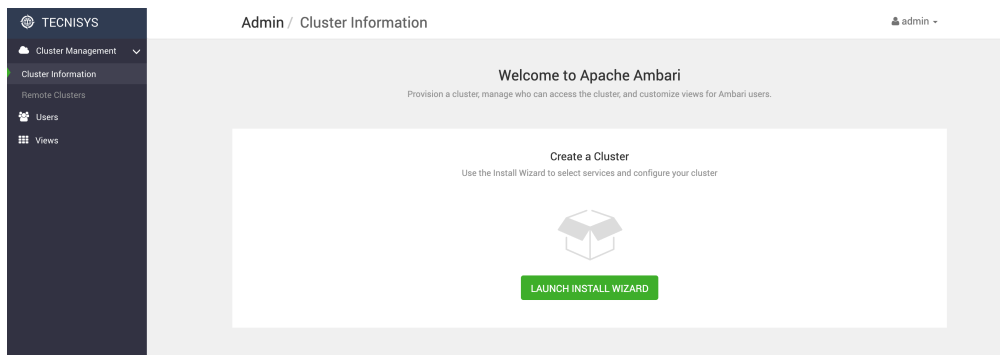


Figure 2 - Ambari Welcome Page

© Copyright Tecnisys 2026. All rights reserved.

2. Enter a name for the Cluster and click NEXT:

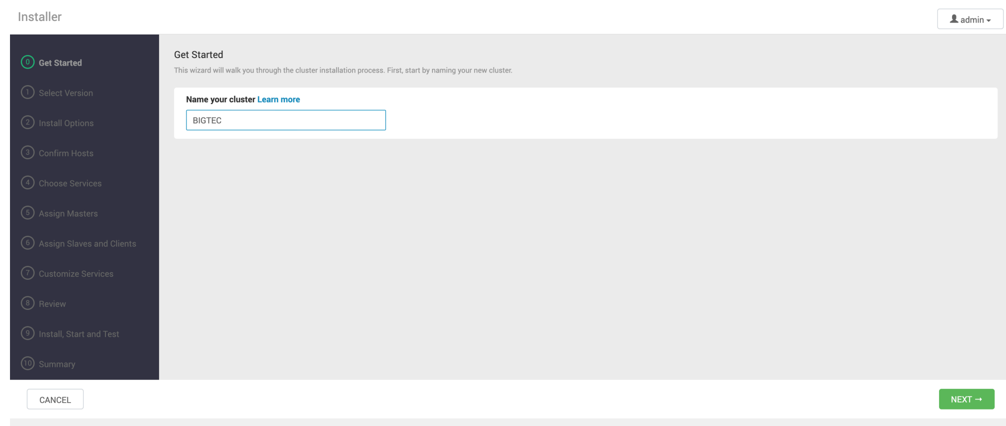


Figure 3 - Cluster Name

Version Selection

1. Select the desired TDP version:

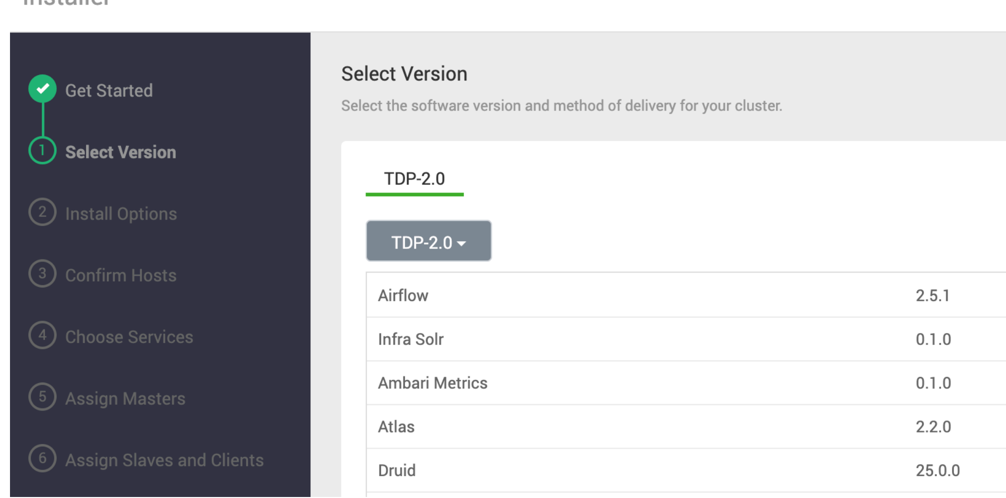


Figure 4 - TDP Version

2. Select the type of package repository (Public or Local) and enter the URL for *Components* (TDP-3.0) and *Utils* (TDP-UTILS-3.0):

© Copyright Tecnisys 2026. All rights reserved.

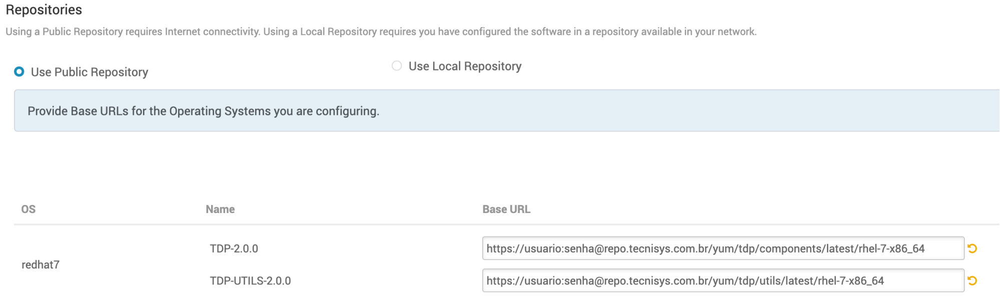


Figure 5 - Package Repositories

3. Then, click NEXT.

IMPORTANT

If you choose to use the Public Package Repository of Tecnisys, access credentials (username and password) must be entered directly in the URL, as shown in the image above.

Installation Options

1. In Target Hosts, enter the *Fully Qualified Domain Name* (FQDN) of the hosts (machines) that will make up the *Cluster*.

   The Ambari Server must have access to the machines provided. Make sure that the FQDN of the machines is correctly resolved, either through a DNS Server (recommended) or locally (via `/etc/hosts` file).

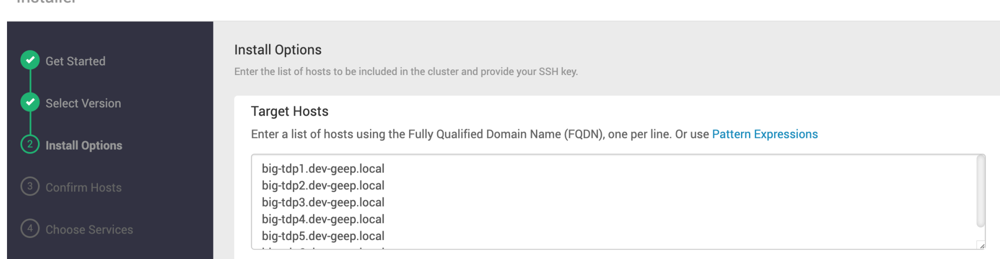


Figure 6 - Cluster Host Information

© Copyright Tecnisys 2026. All rights reserved.

> In Target Hosts, you can enter machines using *Pattern Expressions*. The example shown in the figure above would look like this: `big-tdp[1-7].dev-geep.local`.

2. In *Hosts Registration Information*, select how the *Cluster* machines will be registered.

    - If you choose to provide the private SSH key of the Ambari Server machine for automatic registration of the *Cluster* machines, paste its content into the text field below or upload the file. Then, confirm the username and SSH port to be used. Also, make sure that Trust Relationship (SSH key exchange) has been correctly established, allowing access to all machines via SSH without the need for the password of the Ambari Server's *daemon* user (by default, root).

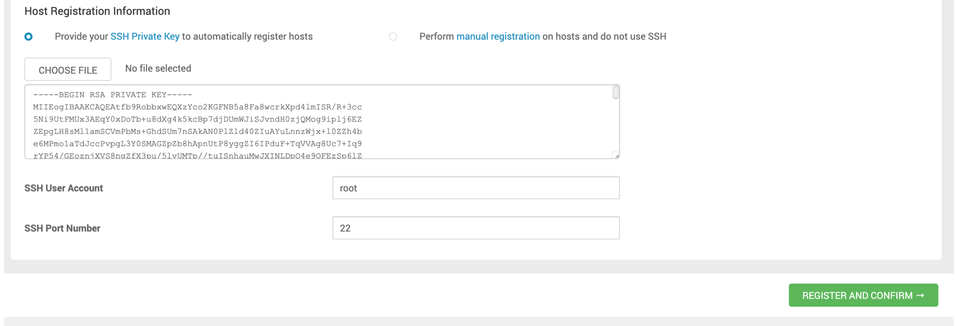
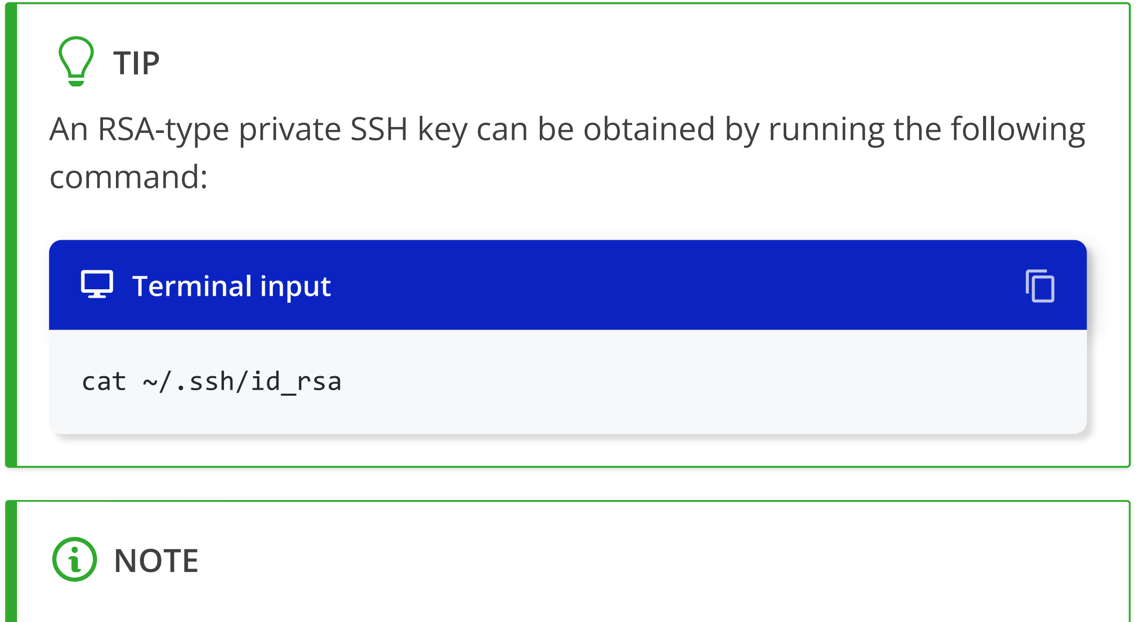


Figure 7 - Host Registration

    - If you prefer manual registration of machines, install the Ambari Agent on each machine before proceeding.



> **TIP**
>
> An RSA-type private SSH key can be obtained by running the following command:
>
> Terminal input
>
> ```
> cat ~/.ssh/id_rsa
> ```

> **NOTE**

© Copyright Tecnisys 2026. All rights reserved.

To manually install the Ambari Agent:

Terminal input

```
yum install ambari-agent
```

3. Then, click REGISTER AND CONFIRM.

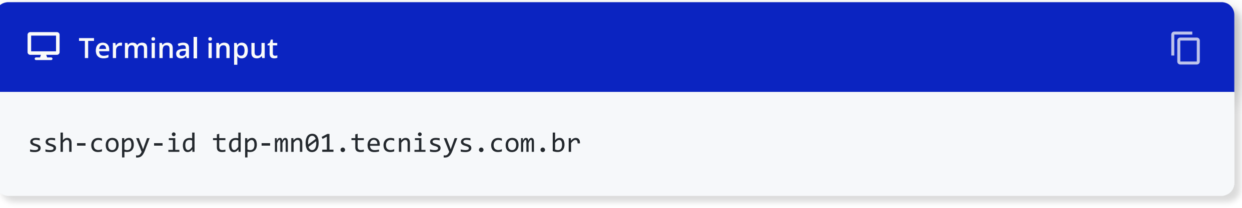
Trust Relationship Configuration

1. On the Ambari Server machine, generate a private SSH key:

Terminal input

```
ssh-keygen
```

2. Copy the SSH key to *ALL* machines in the *Cluster*. For example:



Terminal input

```
ssh-copy-id tdp-mn01.tecnisys.com.br
```

3. Test SSH access to *ALL* machines in the *Cluster* without the user password. For example:

Terminal input

```
ssh root@tdp-mn01.tecnisys.com.br
```

Host Confirmation

After installing the Ambari Agent on all machines specified in the previous step, Ambari performs a series of checks to ensure that the prerequisites have been met (JDK, Firewall, THP, among others).

© Copyright Tecnisys 2026. All rights reserved.

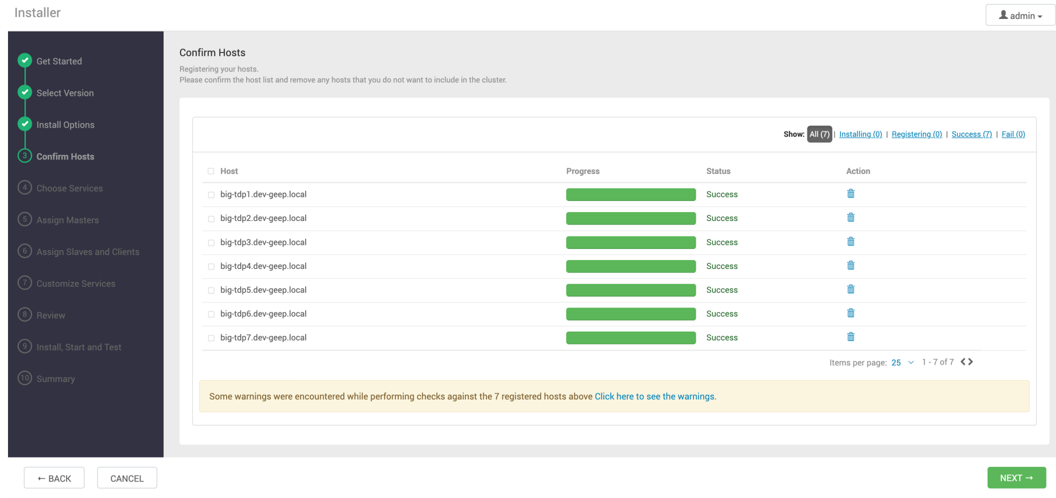


Figure 8 - Host Confirmation

Any errors need to be corrected, and the verification re-run to proceed.

Click NEXT to continue.

NOTE

Package Issues alerts related to already installed PostgreSQL packages can be ignored.

Service Selection

1. Select the service responsible for the Cluster storage layer.

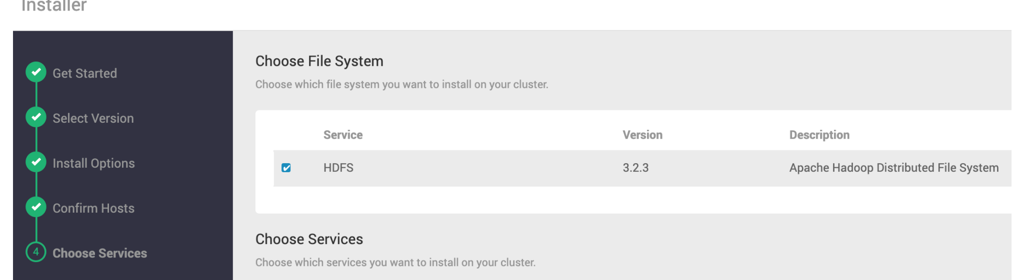


Figure 9 - Storage Layer Service Selection

2. Select the remaining Cluster services.

© Copyright Tecnisys 2026. All rights reserved.

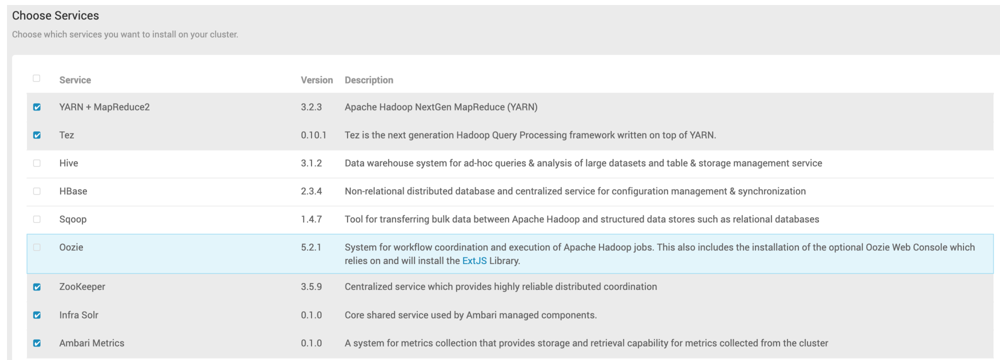


Figure 10 - Selection of Additional Cluster Services

TIP

We recommend initially selecting the basic services, such as YARN + MapReduce2, Tez, Zookeeper, Infra Solr, and Ambari Metrics. Additional services, if needed, can be added after the Cluster is created. This approach makes it easier to handle potential component installation issues.

NOTE

The Cluster requires certain services to operate fully, such as Apache Ranger for security and Apache Atlas for data governance. Therefore, Ambari will display alerts if any functionality is limited by the absence of a specific service. Ignore the alert (click the PROCEED ANYWAY button) if the service will be installed later or if you are aware of this limitation.

3. Then, click NEXT.

Assignment of Master Components

1. Specify the machine for each Master component (usually management and coordination components) of the selected services. On the right side of the page, you can see the organization of components by machine.

© Copyright Tecnisys 2026. All rights reserved.

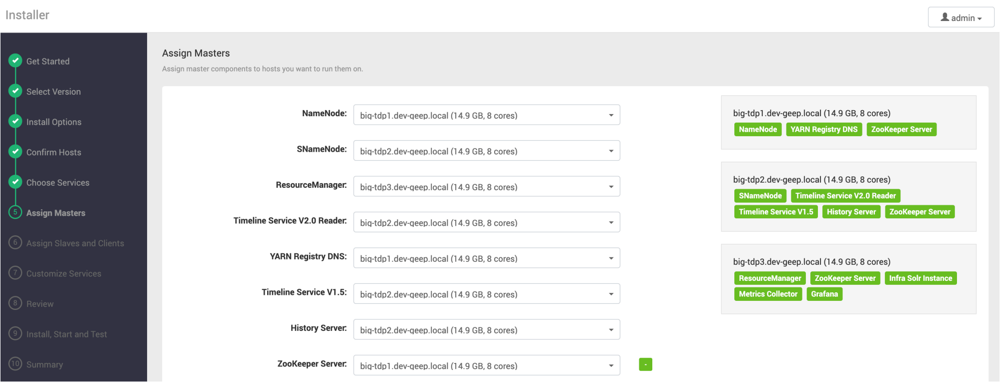


Figure 11 - Master Component Assignment

NOTE

The setup should consider each component's requirements and the available resources on each machine. Some recommendations include:

- Avoid installing non-*Edge* or *Gateway* services on the Ambari Server machine. If possible, dedicate a machine to the Ambari Server.
- Components responsible for high availability should be installed on separate machines, e.g., NameNode and Secondary NameNode (SNameNode).
- Install Zookeeper on an odd number of machines, more than one (01); that is, initially on at least 3 machines.

2. Then, click NEXT.

Assignment of Slave and Client Components

1. Specify the machines where Slave components (usually storage and processing components) and Clients will be installed.

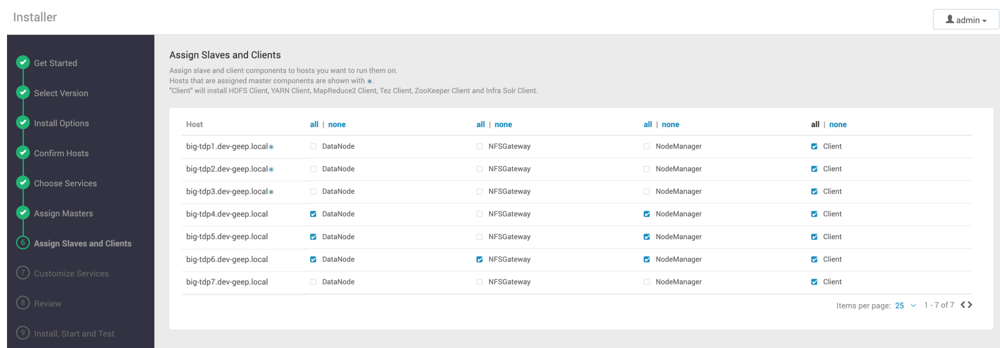


Figure 12 - Assignment of Slave and Client Components

> **NOTE**
>
> Whenever possible, avoid installing Slave components on Master component machines.

2. Then, click NEXT.

Service Customization

In this step, define access credentials, database connection data, directories, users, and other specific information necessary for each service installation.

Resolve any pending items in each section of this step and click NEXT to proceed.

Credentials

For illustration, we have Grafana, a component of Ambari Metrics, which requires setting up the administration username and password:

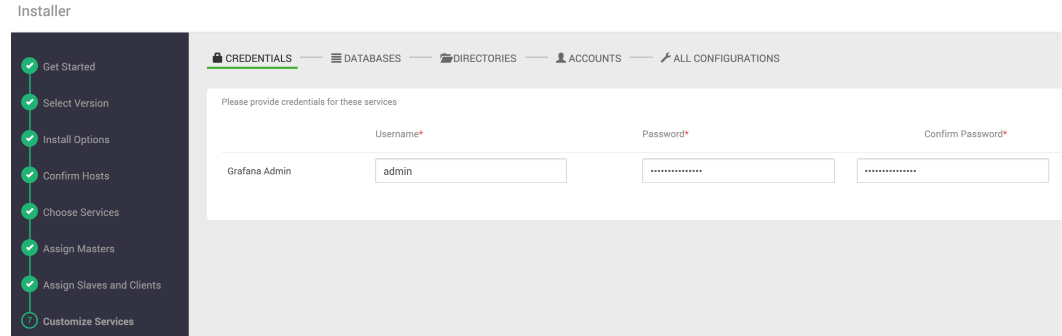


Figure 13 - Defining Grafana Administration Credentials

Databases

As an example, Hive requires a database to store metadata. Here, we provide the connection data for an existing PostgreSQL instance:

© Copyright Tecnisys 2026. All rights reserved.

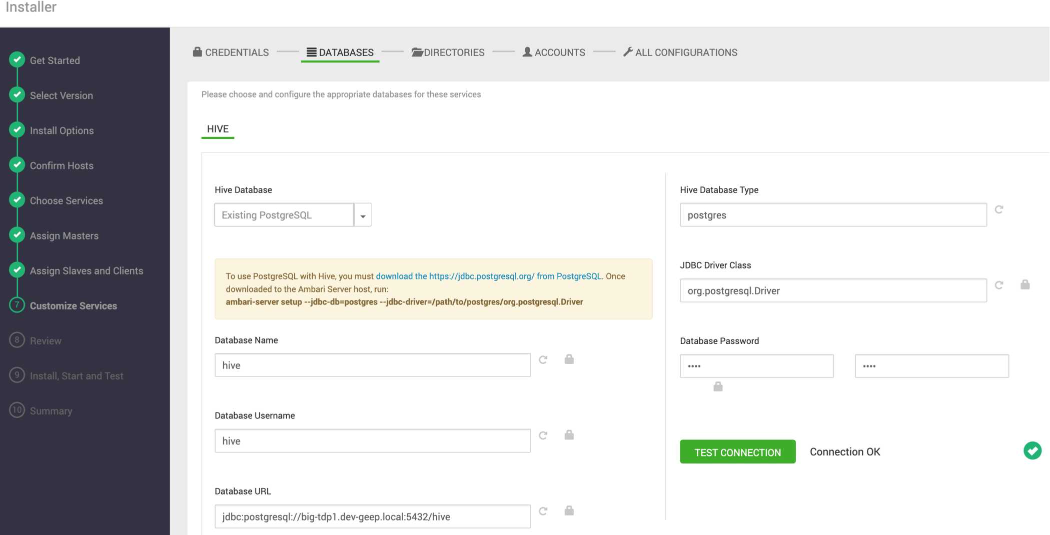


Figure 14 - Defining Hive Database Connection Data

NOTE

Click the TEST CONNECTION button to test the connection to the specified database.

Directories

In this section, you can customize the service directories, such as the DataNodes data directories, NameNode namespace directories, log directories, etc.

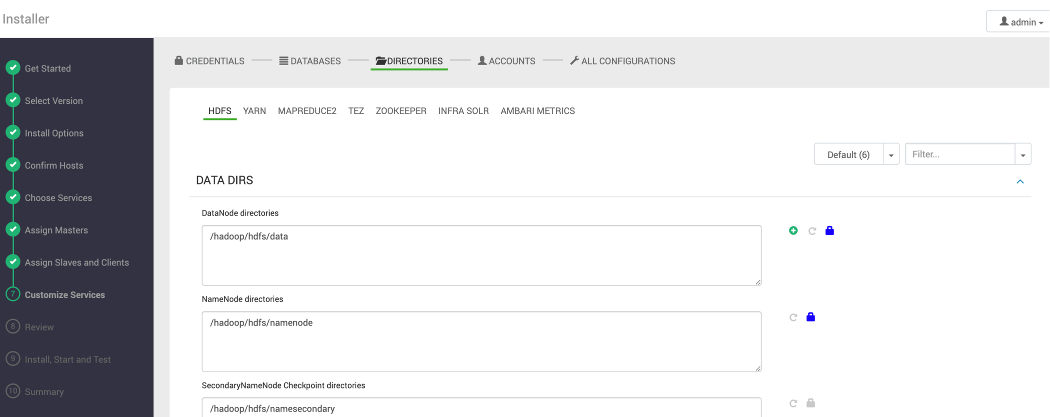


Figure 15 - Defining Service Directories

WARNING

If possible, use exclusive storage devices (disks, SSDs, etc.), volumes, and directories for DataNode, NameNode, JournalNode, NodeManager, Timeline

© Copyright Tecnisys 2026. All rights reserved.

Service, and Zookeeper files.

Service Users

In this section, you can customize the operating system users that will be created for each service.

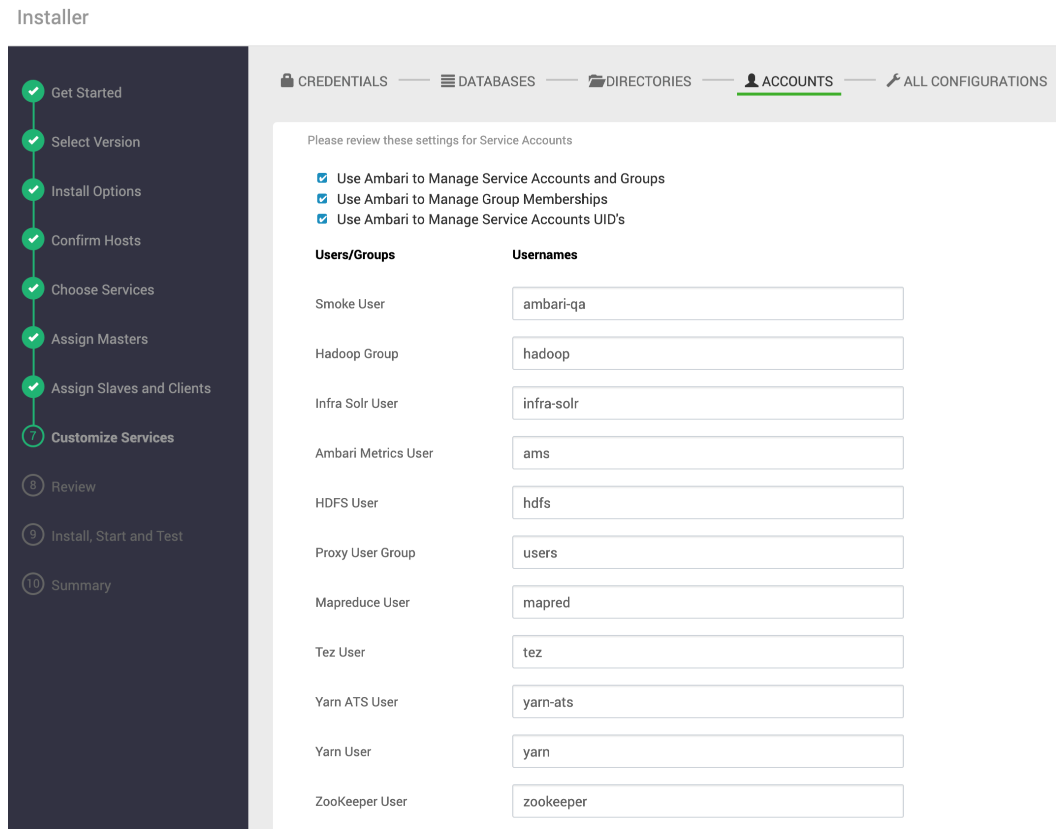


Figure 16 - Defining Service Users

All Configurations

This final section provides access to all configurations of the services to be installed. Review and adjust as needed.

© Copyright Tecnisys 2026. All rights reserved.

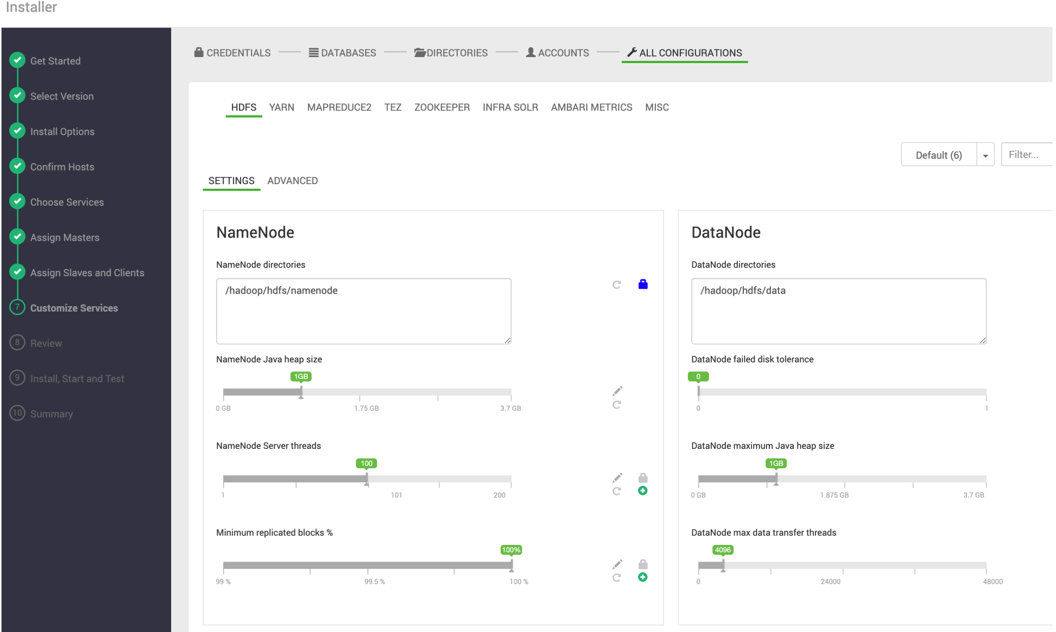


Figure 17 - All Service Configurations

If you missed any configuration, don't worry. After installation, all these settings can be modified through Ambari.

Configuration Review

In this final step before creating the Cluster, a review of the settings is presented. Carefully check all information, and if you need to change any settings, use the left-side navigation area to go back to the desired step.

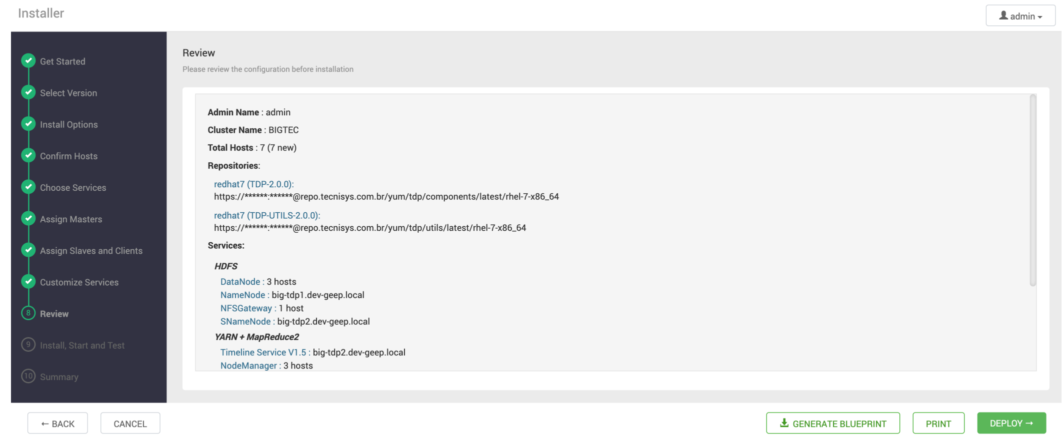
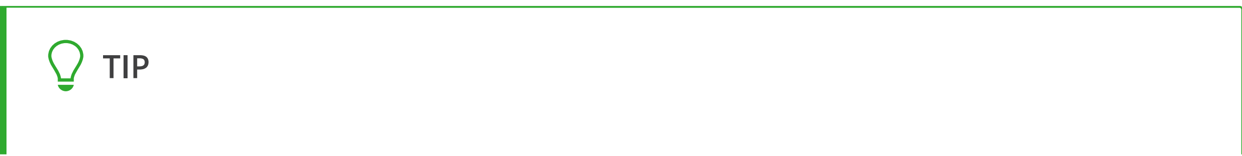


Figure 18 - Configuration Review Before Installation



© Copyright Tecnisys 2026. All rights reserved.

Use the PRINT button to generate an installation report and the GENERATE BLUEPRINT button to create an XML file with all defined settings, which can be used in the future to recreate the Cluster via the Ambari REST API.

To start the Cluster deployment, click the DEPLOY button.

Service Installation, Startup, and Testing

In this step, services will be installed, started, and tested, respecting each service's dependencies and integrations.

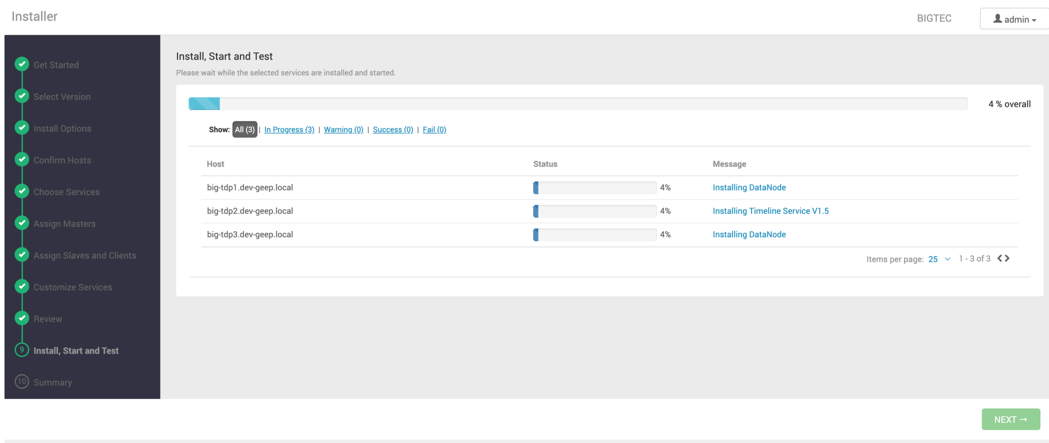


Figure 19 - Service Installation

Click the Message column link to view the scheduled tasks for each machine.

In case of failures, Ambari may pause the deployment, allowing you to resume after fixing the issue by clicking the RETRY button.

However, depending on the progress made, Ambari may complete the deployment and make the Cluster available as-is, even if not all components of a specific service have been successfully installed, started, or tested. In this case, after the Cluster is created, you can adjust configurations or reinstall only the problematic service through Ambari.

Once deployment is complete, click NEXT.

© Copyright Tecnisys 2026. All rights reserved.

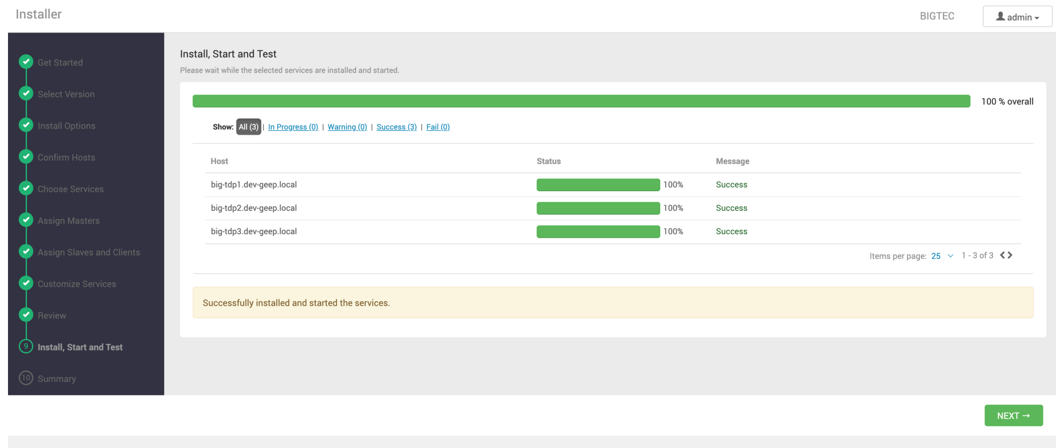


Figure 20 - Installation, Startup, and Testing Completed

Summary

In the final step of the process, an implementation summary is displayed.

Click the COMPLETE button to finalize the operation and access the administration area of the created *Cluster*.

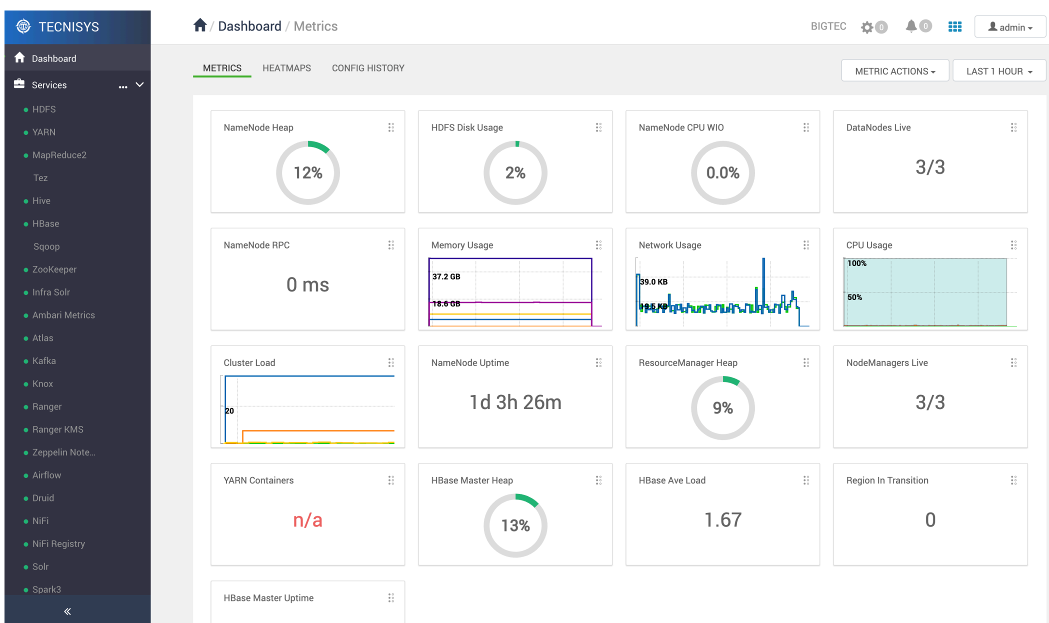


Figure 21 - Cluster Administration Area

© Copyright Tecnisys 2026. All rights reserved.

PART IV - UPDATE

© Copyright Tecnisys 2026. All rights reserved.

Update Flowchart

The following flowchart graphically represents the update process of a TDP Cluster.

In summary, the TDP update process is divided into three main stages:

- Update of Apache Ambari;
- Registration of the new version of the TDP *stack*; and
- Update of the components installed in the *Cluster*.

In the following sections, we will detail each of these stages.

© Copyright Tecnisys 2026. All rights reserved.

Preparação do Ambiente

Início

Verificar se todos os serviços instalados nos Clusters estão funcionais

Atualizar ou criar repositórios de pacotes para a nova versão

Atualizar cache do YUM

Atualização dos Pacotes de Instalação

Atualizar o Ambari Agent em todas as máquinas do Cluster

Atualizar o Ambari Server

Atualizar as propriedades e banco de metadados do Ambari

Atualização da versão atual da Stack

Alerta para a stack atual?

SIM

O Banco de Metadados está no PostgreSQL ?

NÃO

Atualizar a versão atual via ambari-server set-current

SIM

Atualizar a versão atual via UPDATE diretamente no banco de Metadados

NÃO

Registrar a nova Stack do TDP no Ambari

Atualização dos Componentes

Desabilitar o início automático (Auto Start) dos Serviços

Instalar a nova versão dos componentes da nova Stack

Este procedimento apenas realiza a cópia dos arquivos RPM para os hosts do Cluster

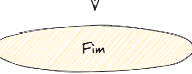
Realizar a atualização (UPGRADE) dos componentes do Cluster

Express Upgrade

Rolling Upgrade

Habilitar o inicio automático (Auto-Start) dos Serviços

© Copyright Tecnisys 2026. All rights reserved.



© Copyright Tecnisys 2026. All rights reserved.

Update Prerequisites

Below we present the minimum prerequisites for updating a TDP Cluster, which may vary according to the desired services or the technical and organizational needs of each environment.

If needed, request support at Support Area.

Disk Space

The update requires the installation of new packages, which demand additional disk space, either in the package repository or in the directories of the Cluster machines designated for binaries, configuration files, and libraries. For a safe update without incidents, we recommend a minimum of 100 GB of free disk space.

> WARNING
>
> Before installing the packages of the new version, check the available disk space to avoid file corruption, operation restart, among other problems.

Communication Network

Communication between the Cluster machines and the package repository must be guaranteed, whether through a local network or the internet. The speed and stability of the network directly influence the execution time and integrity of the operation.

Auto-Start Function Disabled

Before starting the component update, it is important to disable the Auto-Start function of services in Ambari. For this, follow these steps:

Instructions

- Select the `Service Auto-Start` option from the Ambari sidebar menu.
- Change the switch to `Disabled` in the `Auto Start Settings` option.

© Copyright Tecnisys 2026. All rights reserved.

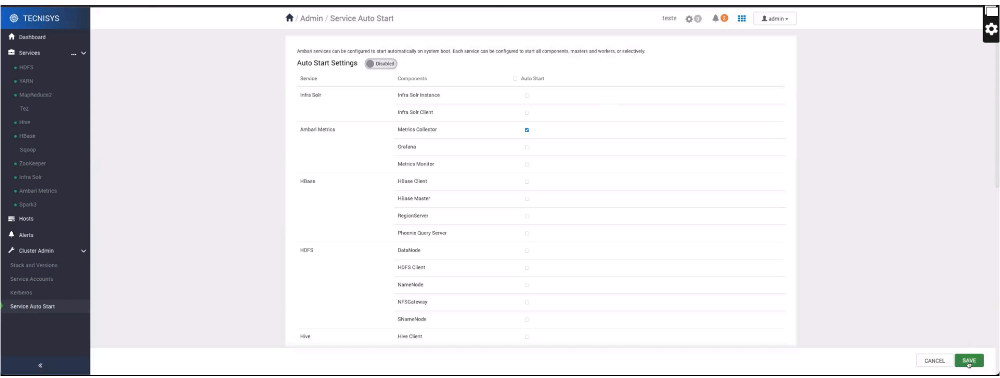


Figure 5 - Disabling Auto Start

- Click `Save` to save the change and confirm the operation.

> **NOTE**
>
> Once the update is completed, the Auto Start function can be enabled again.

Workloads Finished

We recommend that the main workloads be completed before the update. This avoids interrupting ongoing processes, ensuring data integrity and continuity of operations.

High Availability Services

The `Rolling Upgrade` type update is a process that allows updating a service without interrupting the operation of the Cluster. For this, it is necessary that the services are configured for high availability, ensuring the continuity of operations.

Backup of Metadata Databases

Various components of the TDP platform use metadata databases to store information about configurations, operations, among others.

Before starting the update, it is recommended to perform a backup of the metadata databases, thus ensuring the possibility of restoration in case of failures.

Additional Settings in Apache Ambari for Large Clusters

In a large Cluster, with dozens or hundreds of machines, some additional settings in Apache Ambari may be necessary to ensure the success of the update.

© Copyright Tecnisys 2026. All rights reserved.

Adjusting the Timeout for Package Installation

In a large Cluster, installing packages through Ambari can take a considerable amount of time. Therefore, to avoid timeout issues, increase the value of the `agent.package.install.task.timeout` parameter, located in the configuration file `/etc/ambari-server/conf/ambari.properties` of the Ambari Server machine. For this, follow these steps:

Instructions

1. Open the Ambari Server configuration file with a text editor:

Terminal input

```
vim /etc/ambari-server/conf/ambari.properties
```

1.1. Adjust the value of the `agent.package.install.task.timeout` property. For example, to 1 hour (3600 seconds):

Terminal input

```
agent.package.install.task.timeout=3600
```

1.2. Save the change and close the configuration file of the Ambari Server.

© Copyright Tecnisys 2026. All rights reserved.

```
(01.11.2024 13:52:26)[root@big-tdp-220-02 ~]$ vim /etc/ambari-server/conf/ambari.properties

# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

#
#Fri Nov 01 13:51:20 UTC 2024
agent.package.install.task.timeout=3600
agent.stack.retry.on_repo_unavailability=false
agent.stack.retry.tries=5
agent.task.timeout=900
agent.threadpool.size.max=25
ambari-server.user=root
ambari.python.wrap=ambari-python-wrap
bootstrap.dir=/var/run/ambari-server/bootstrap
bootstrap.script=/usr/lib/ambari-server/lib/ambari_server/bootstrap.py
bootstrap.setup_agent.script=/usr/lib/ambari-server/lib/ambari_server/setupAgent.py
check_database_skipped=false
client.threadpool.size.max=25
common.services.path=/var/lib/ambari-server/resources/common-services
custom.action.definitions=/var/lib/ambari-server/resources/custom_action_definitions
custom.postgres.jdbc.name=postgresql-42.2.16.jar
extensions.path=/var/lib/ambari-server/resources/extensions
gpl.license.accepted=false
http.cache-control=no-store
http.charset=utf-8
http.pragma=no-cache
http.strict-transport-security=max-age=31536000
http.x-content-type-options=nosniff
http.x-frame-options=DENY
http.x-xss-protection=1; mode=block
java.home=/usr/lib/jvm/jdk8u402-b06
:wq
```

Figure 6 - Adjust timeout

2. Restart the Ambari Server service:

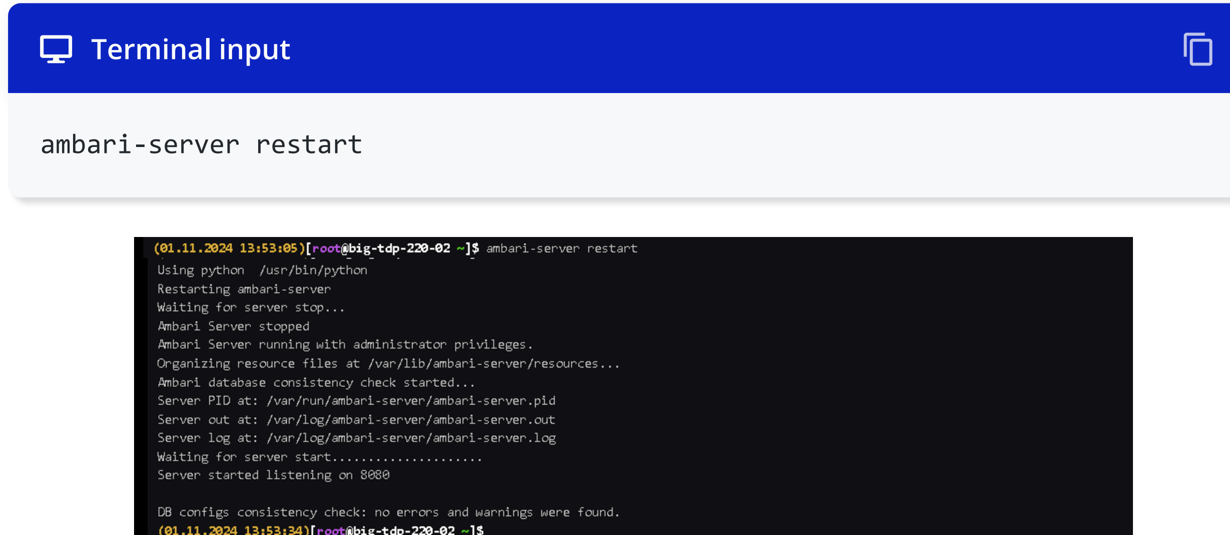


Figure 6 - Restart Ambari

Adjusting the NameNode Restart Time

In a large Cluster, the process of starting the NameNode can take a significant amount of time. The startup time depends not only on the available computing resources but

© Copyright Tecnisys 2026. All rights reserved.

also on the volume of data and network parameters.

To ensure that Ambari's requests to start the NameNode do not exceed the timeout during an update, configure the NameNode restart timeout parameter in Ambari, `upgrade.parameter.nn-restart.timeout`, in the file `/etc/ambari-server/conf/ambari.properties` of the Ambari Server machine.

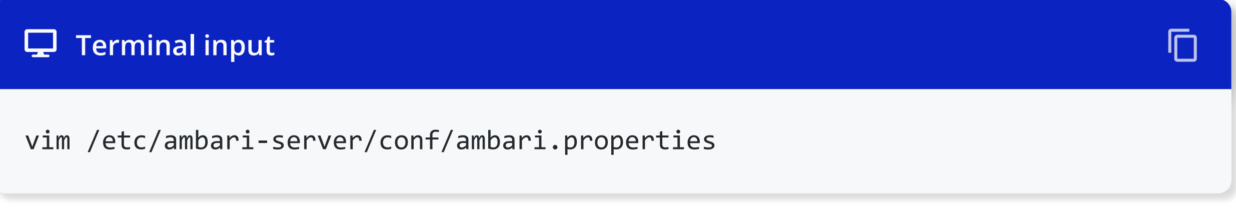
NOTE

If the parameter `upgrade.parameter.nn-restart.timeout` does not exist in the configuration file, add it.

Instructions

Initially, add 10% to the time (in seconds) normally required to restart the NameNode. Although there is no standard method to determine an appropriate value, the following guidance can be used. For example, 660 seconds (11 minutes) if the normal restart time is 600 seconds (10 minutes). For this, follow these steps:

1. Open the Ambari Server configuration file with a text editor:



Terminal input

```
vim /etc/ambari-server/conf/ambari.properties
```

1.1. Adjust the value of the property *upgrade.parameter.nn-restart.timeout*:

Terminal input

```
upgrade.parameter.nn-restart.timeout=660
```

1.2. Save the change and close the configuration file of the Ambari Server.

2. Restart the Ambari Server service:

Terminal input

© Copyright Tecnisys 2026. All rights reserved.

```
ambari-server restart
```

```
(01.11.2024 11:23:30)[root@big-tdp-220-02 ~]$ vim /etc/ambari-server/conf/ambari.properties
server.execution.scheduler.misfire.toleration.minutes=480
server.fqdn.service.url=http://169.254.169.254/latest/meta-data/public-hostname
server.http.session.inactive_timeout=1800
server.jdbc.connection-pool=internal
server.jdbc.database=postgres
server.jdbc.database_name=ambari
server.jdbc.postgres.schema=ambari
server.jdbc.user.name=ambari
server.jdbc.user.passwd=/etc/ambari-server/conf/password.dat
server.os_family=redhat9
server.os_type=rocky linux9
server.persistence.type=local
server.python.log.level=INFO
server.python.log.name=ambari-server-command.log
server.stages.parallel=true
server.task.timeout=1200
server.tmp.dir=/var/lib/ambari-server/data/tmp
server.version.file=/var/lib/ambari-server/resources/version
shared.resources.dir=/usr/lib/ambari-server/lib/ambari_commons/resources
skip.service.checks=false
stack.java.home=/usr/lib/jvm/jdk8u402-b06
stack.jce.name=jce_policy-8.zip
stack.jdk.name=OpenJDK8U-jdk_x64_linux_hotspot_8u402b06.tar.gz
stackadvisor.script=/var/lib/ambari-server/resources/scripts/stack_advisor.py
ulimit.open.files=65536
upgrade.parameter.nn-restart.timeout=660
user.inactivity.timeout.default=0
user.inactivity.timeout.role.readonly.default=0
views.ambari.request.connect.timeout.millis=30000
views.ambari.request.read.timeout.millis=45000
views.http.cache-control=no-store
views.http.charset=utf-8
views.http.pragma=no-cache
views.http.strict-transport-security=max-age=31536000
views.http.x-content-type-options=nosniff
views.http.x-frame-options=SAMEORIGIN
views.http.x-xss-protection=1; mode=block
views.request.connect.timeout.millis=5000
views.request.read.timeout.millis=10000
views.skip.home-directory-check.file-system.list=wasb,adls,adl
webapp.dir=/usr/lib/ambari-server/web

:wq

(01.11.2024 11:24:25)[root@big-tdp-220-02 ~]$ ambari-server restart
Using python  /usr/bin/python
Restarting ambari-server
Waiting for server stop...
Ambari Server stopped
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Ambari database consistency check started...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start....................
Server started listening on 8080

DB configs consistency check: no errors and warnings were found.
(01.11.2024 11:24:55)[root@big-tdp-220-02 ~]$
```

Figure 6 - Adjust namenode

© Copyright Tecnisys 2026. All rights reserved.

Apache Ambari Update

Prerequisites

> **Pré-Requisitos**
>
> - Creation or update of the local package repository for updates with restricted or no internet access.
> - Verification of the prerequisites for the update.

Updating Apache Ambari

Package Repository

Instructions

Carry out the following procedures on all Cluster machines to define a new package repository for Apache Ambari:

- Access the directory where the Ambari package repository file (*repo file*) is located:

```
cd /etc/yum.repos.d/
```

- Execute the command below to download the `ambari.repo` file and save it in the local directory /etc/yum.repos.d/, to be used by the YUM package manager.

```
wget -O /etc/yum.repos.d/ambari.repo
https://repo.tecnisys.com.br/repository/public/tdp/3.0/el-9-
```

© Copyright Tecnisys 2026. All rights reserved.

```
x86_64/ambari.repo
```

Terminal input

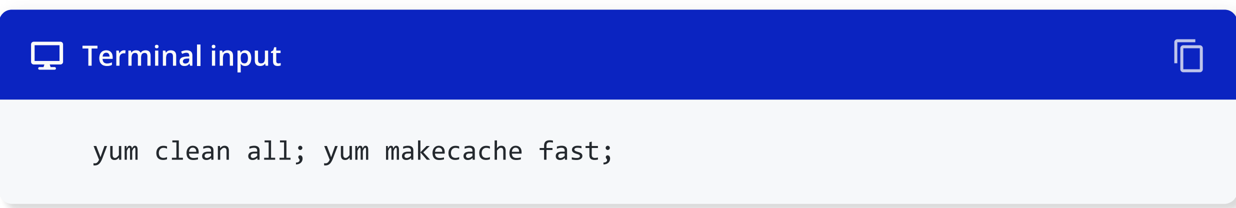
```
curl -o /etc/yum.repos.d/ambari.repo
https://repo.tecnisys.com.br/repository/public/tdp/3.0/el-9-
x86_64/ambari.repo
```

WARNING

The commands suggested below overwrite the file `/etc/yum.repos.d/ambari.repo`, if it already exists in the directory. If necessary, perform a *backup* of the file with the current definitions first.

- In the ambari.repo file, replace the USER and PASS variables, present in the URLs of the *baseurl* and *gpgkey* properties, respectively, with your user and password registered on the Tecnisys website . If using a local repository, adjust these URLs.

- After defining the package repository of the new TDP version, we recommend updating the yum/dnf *cache*:



Terminal input

```
yum clean all; yum makecache fast;
```

Updating the Ambari Agent

Instructions

With the new package repository defined, begin updating the Ambari Agent component.

WARNING

Communication between the Ambari Agent and the Ambari Server is suspended during the update. However, the services of other Cluster components continue to operate.

© Copyright Tecnisys 2026. All rights reserved.

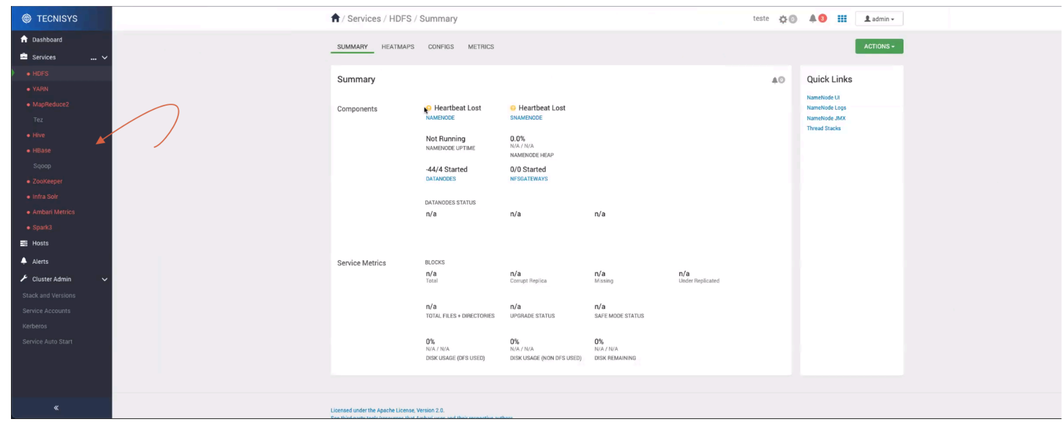


Figure 1 - Service situation during the Ambari Agent update

To update the Ambari Agent, perform the following procedure on all Cluster machines:

- Stop the Ambari Agent service:

Terminal input

```
ambari-agent stop
```

- Update the Ambari Agent binaries:

Terminal input

```
yum upgrade ambari-agent -y
```

NOTE

Any customizations of the `ambari-agent.ini` file can be recovered from the backup file generated: `ambari-agent.ini.old`

- Start the Ambari Agent service:

Terminal input

```
ambari-agent start
```

© Copyright Tecnisys 2026. All rights reserved.

Updating the Ambari Server

Instructions

After updating the Ambari Agent on all Cluster machines, proceed to update the Ambari Server.

WARNING

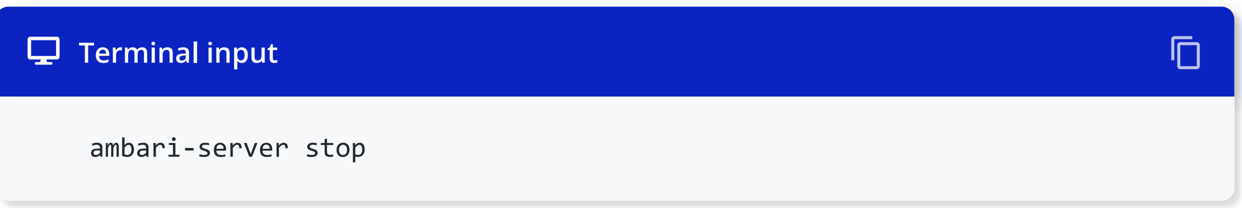
This operation requires stopping the Ambari Server service.

WARNING

It is recommended to perform a *backup* of the Ambari metadata database before starting this operation.

Carry out the following procedure only on the machine where the Ambari Server is installed:

- Stop the Ambari Server service:




```
ambari-server stop
```

- Update the Ambari Server binaries:

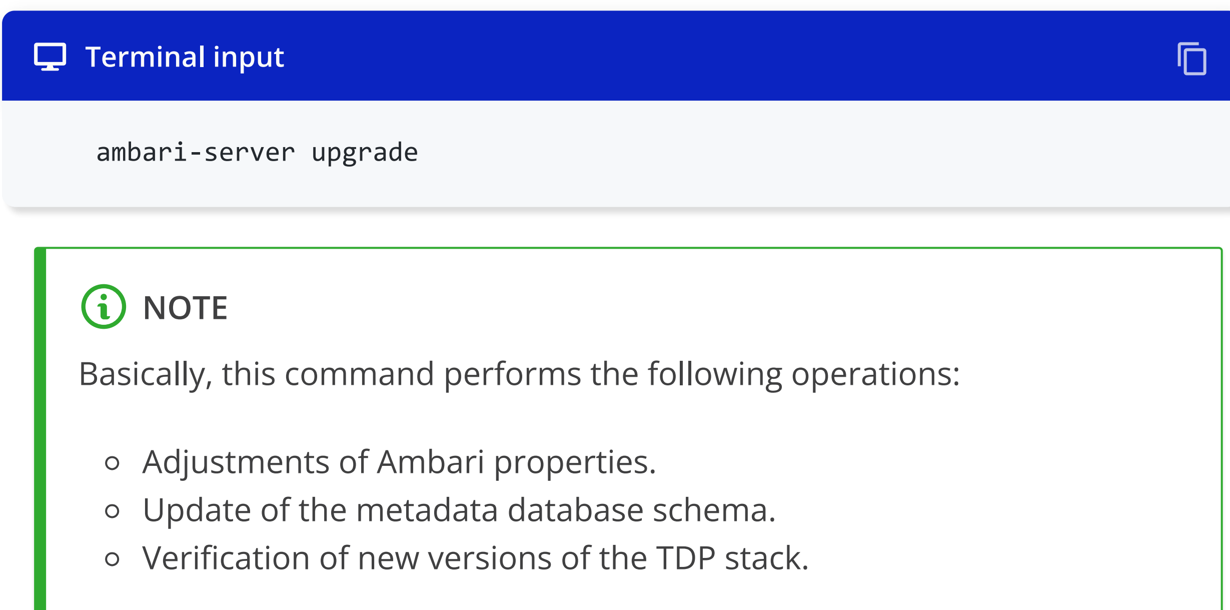
```
yum upgrade ambari-server -y
```

© Copyright Tecnisys 2026. All rights reserved.

```
TDP-GEEP-1
[root@tdp-geep-1 yum.repos.d]# yum upgrade ambari-server -y
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.uepg.br
 * epel: espejito.fder.edu.uy
 * extras: mirror.uepg.br
 * updates: mirror.uepg.br

   ...

 * epel: espejito.fder.edu.uy
 * extras: mirror.uepg.br
 * updates: mirror.uepg.br
Resolving Dependencies
--> Running transaction check
---> Package ambari-server.x86_64 0:2.7.6.1-0 will be updated
---> Package ambari-server.x86_64 0:2.7.6.2-0 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================
 Package              Arch            Version            Repository               Size
==========================================================================================
Updating:
 ambari-server        x86_64          2.7.6.2-0          Ambari-TDP-210           680 M

Transaction Summary
==========================================================================================
Upgrade  1 Package

Total download size: 680 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
ambari-server-2.7.6.2-0.x86_64.rpm           51% [=======================-          ]  49 MB/s | 353 MB  00:00:06 ETA
```

Figure 2 - Updating the Ambari Server binaries

- Update the Ambari Server metadata:



Terminal input

```
ambari-server upgrade
```

NOTE

Basically, this command performs the following operations:

- Adjustments of Ambari properties.
- Update of the metadata database schema.
- Verification of new versions of the TDP stack.

```
TDP-GEEP-1
[root@tdp-geep-1 yum.repos.d]# ambari-server upgrade
Using python  /usr/bin/python
Upgrading ambari-server
INFO: Upgrade Ambari Server
INFO: Updating Ambari Server properties in ambari.properties ...
INFO: Updating Ambari Server properties in ambari-env.sh ...
INFO: Original file ambari-env.sh kept
INFO: Fixing database objects owner
Ambari Server configured for Embedded Postgres. Confirm you have made a backup of the Ambari Server database [y/n] (n)? y
INFO: Upgrading database schema
INFO: Return code from schema upgrade command, retcode = 0
INFO: Console output from schema upgrade command:
INFO: {}


INFO: Schema upgrade completed
Adjusting ambari-server permissions and ownership...
Ambari repo file doesn't contain latest json url, skipping repoinfos modification
Ambari Server 'upgrade' completed successfully.
[root@tdp-geep-1 yum.repos.d]#
```

Figure 3 - Ambari Server Upgrade

© Copyright Tecnisys 2026. All rights reserved.

- Start the Ambari Server:

Terminal input

```
ambari-server start
```

- Check the new version of Ambari in the web interface by clicking on the user button, located at the top right of the page, and then on the *About* option.

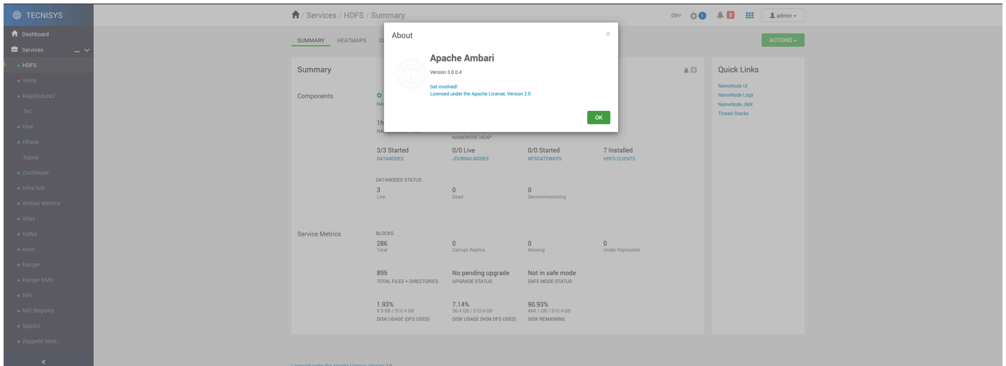


Figure 4 - Ambari Version

- Check the new version of the TDP stack in the web interface by clicking on the *Stack and Versions* option from the sidebar menu, and then on the *VERSIONS* tab.

> **IMPORTANT**
>
> If the alert *"Host component out of sync"* is displayed on the *VERSIONS* tab, perform the procedure defined in Updating the Current Stack Version to correct this inconsistency.

Updating the Current Stack Version

Instructions

If the alert "Host component out of sync" is displayed when checking the versions of the TDP stack in the Ambari web interface, carry out the detailed procedure in this section to correct this occasional inconsistency between the current stack version and the information defined in the Ambari metadata database.

© Copyright Tecnisys 2026. All rights reserved.

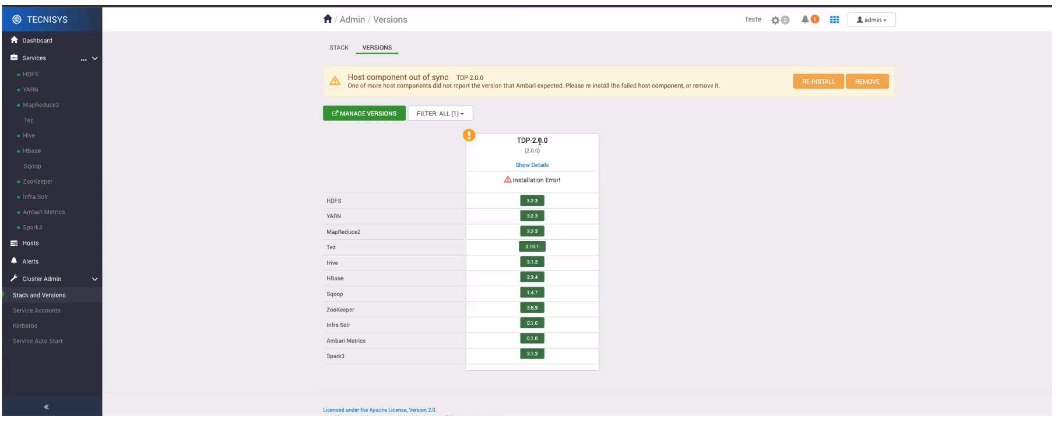


Figure 5 - Alert on the _VERSIONS_ tab

WARNING

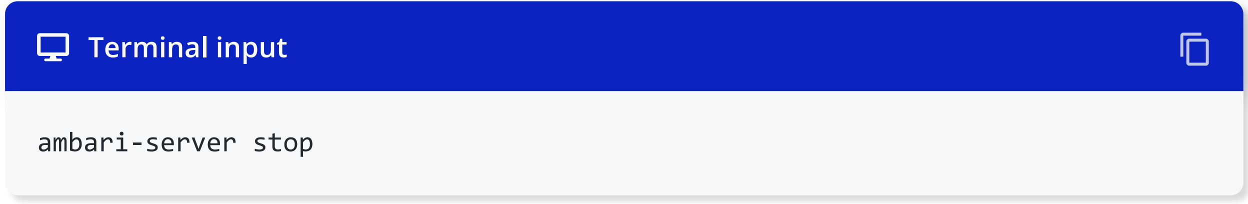
This operation requires stopping the Ambari Server service.

WARNING

It is recommended to perform a *backup* of the Ambari metadata database before starting this operation.

Carry out the following procedure on the machine where the Ambari Server is installed:

- Stop the Ambari Server:

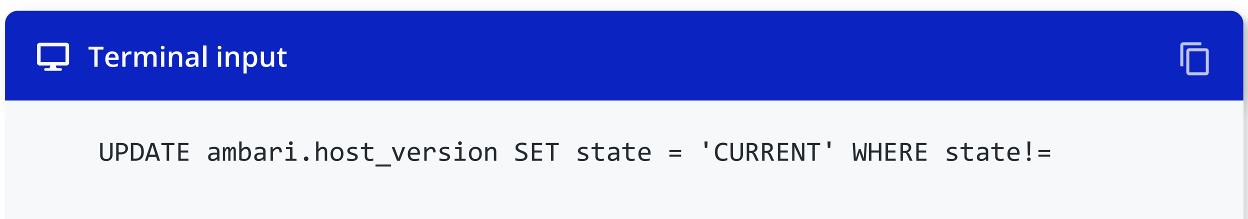


Terminal input

```
ambari-server stop
```

- Update the Ambari metadata database.

  If the metadata database is NOT on PostgreSQL, access the relevant database and execute the following command:



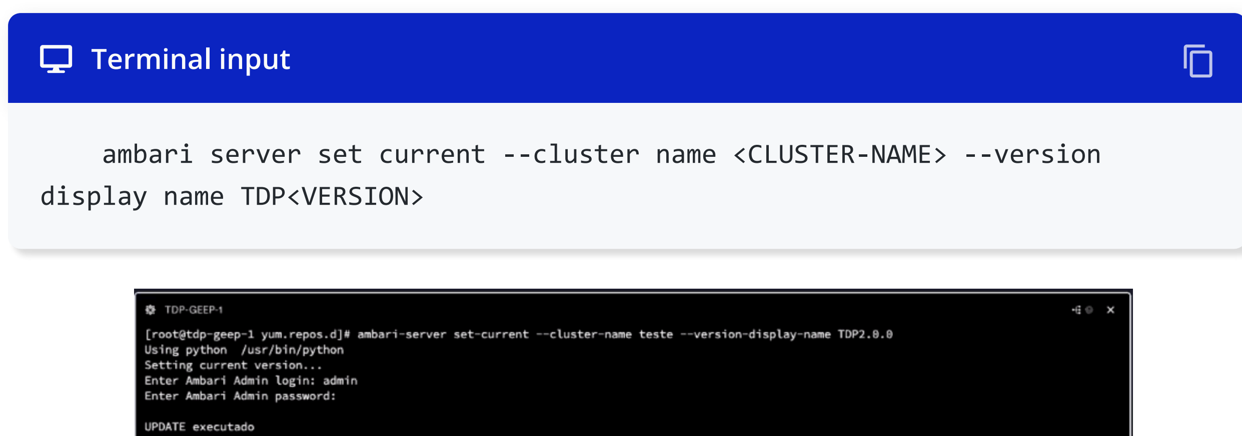
Terminal input

```
    UPDATE ambari.host_version SET state = 'CURRENT' WHERE state!=
```

© Copyright Tecnisys 2026. All rights reserved.

```
'CURRENT';
```

If the metadata database is on PostgreSQL:



Terminal input

```
    ambari server set current --cluster name <CLUSTER-NAME> --version
display name TDP<VERSION>
```


Figure 6 - Setting the current stack version

> **NOTE**
>
> Ensure that the CLUSTER-NAME and VERSION are correct by consulting the *Stack and Versions* page of the Ambari web interface.

- Start the Ambari Server:

Terminal input

```
ambari-server start
```

- Check if the inconsistency has been resolved by accessing the Ambari web interface, clicking on the *Stack and Versions* option from the sidebar menu, and then on the *VERSIONS* tab.

© Copyright Tecnisys 2026. All rights reserved.

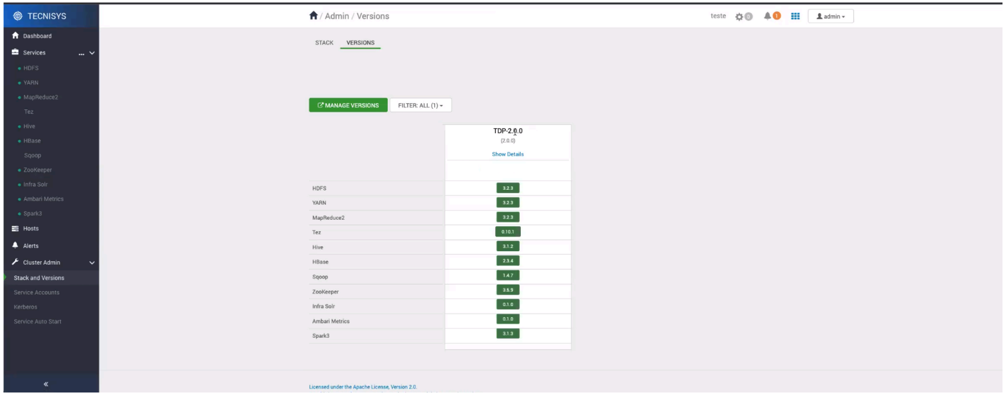


Figure 7 - Inconsistency resolved

© Copyright Tecnisys 2026. All rights reserved.

New Version Registration

Below, the procedures for registering the new version of the TDP stack in Ambari are presented:

- In the Ambari web interface, access the "Stack and Versions" page through the sidebar, click on the "VERSIONS" tab and then click on the "MANAGE VERSIONS" button.

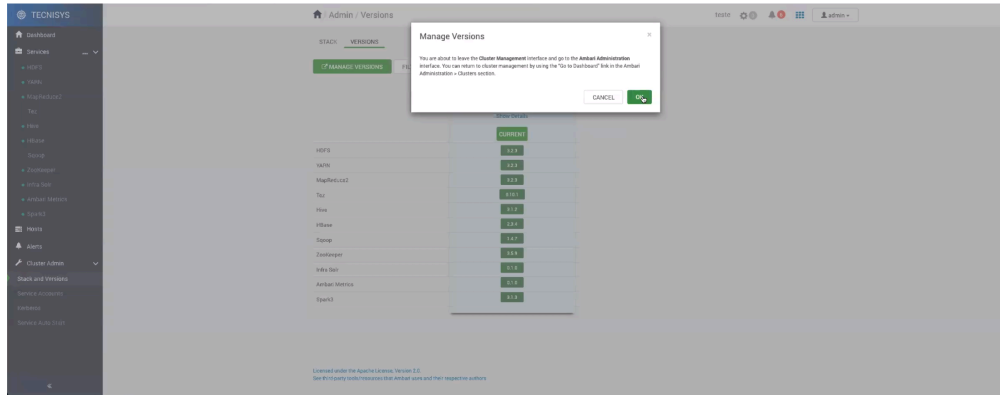


Figure 1 - Redirection to the version administration area

- Confirm the redirection to Ambari's version administration area.

- Select "Register Version".

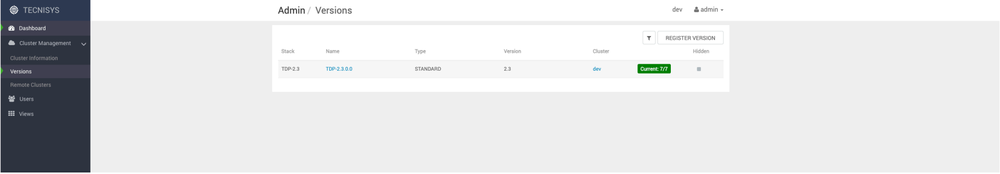


Figure 2 - Version administration area

- In the registration of the new version, provide the *Base URL* of the component repositories (TDP-3.0) and utilities (TDP-UTILS-3.0).

© Copyright Tecnisys 2026. All rights reserved.

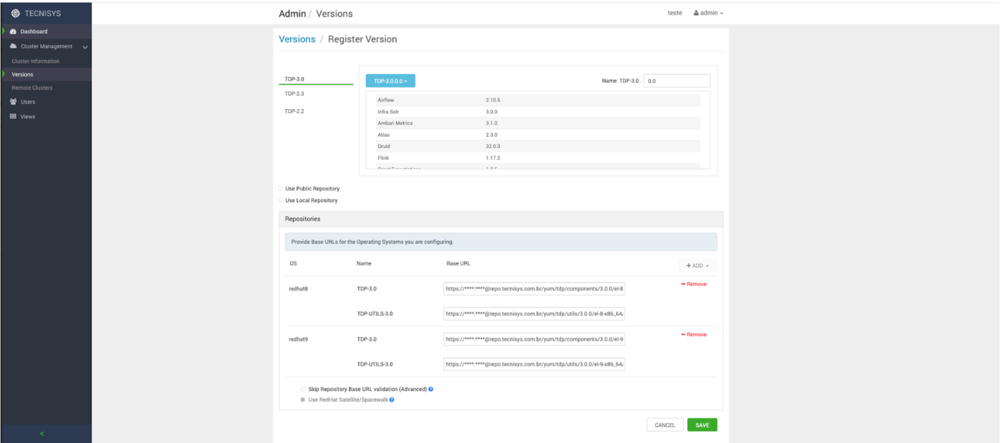


Figure 3 - Registering the new version

- Save the changes.
- Subsequently, the new version of the TDP stack will be available in the *VERSIONS* tab.

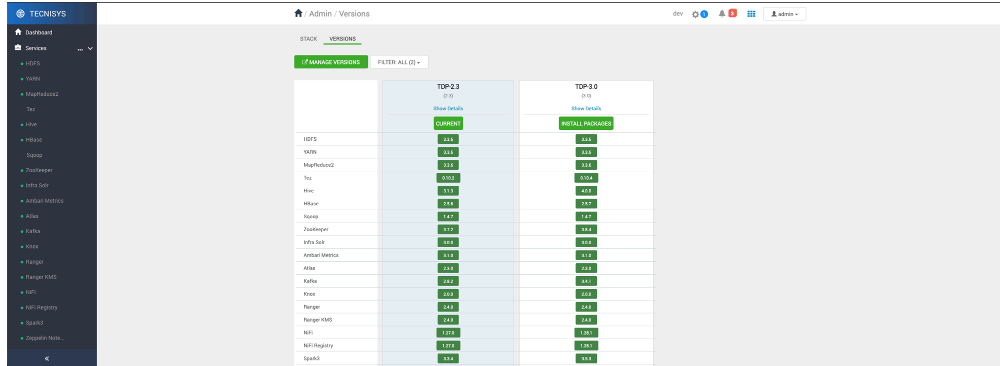


Figure 4 - New version available

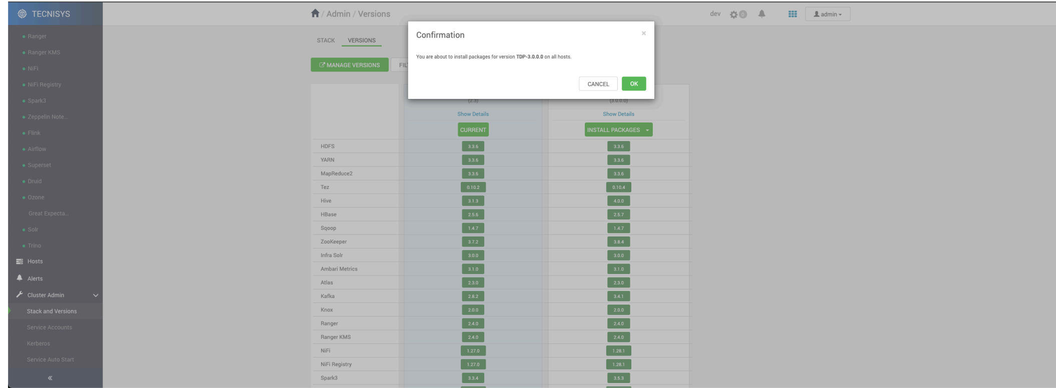
© Copyright Tecnisys 2026. All rights reserved.

Components Update

The component update is carried out through Ambari. The process is divided into two stages: installation of packages available in the new version and execution of the *upgrade*.

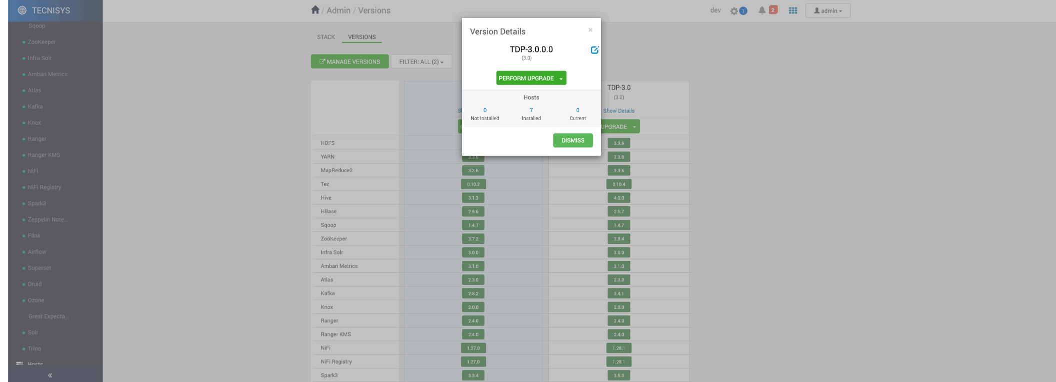
Package Installation

1. In Ambari, on the "Stack and Versions" page, VERSIONS tab, click on the INSTALL PACKAGES button of the new version.

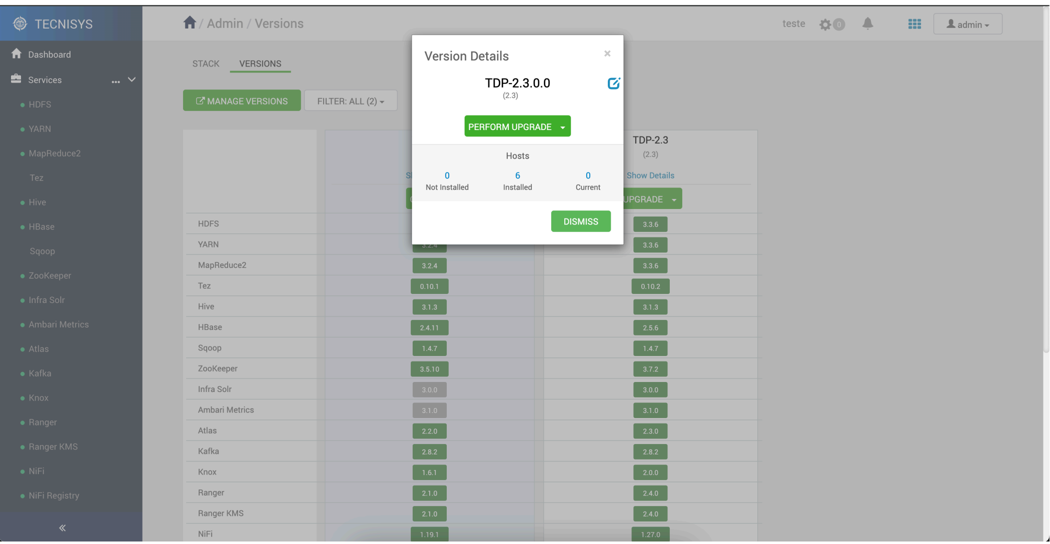


2. Confirm the installation of the packages.

3. Click on the "Show details" link for both versions to view the total number of machines with installed packages.



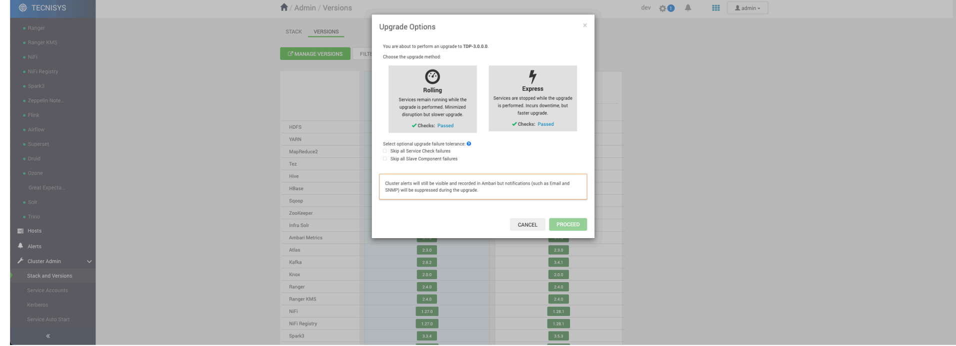
© Copyright Tecnisys 2026. All rights reserved.



NOTE

Up to this point, even though the packages of the new version may already be installed on the machines, the *Cluster* still operates with the binaries of the components from the current version ("CURRENT"). At this stage, any issues with the update are limited to the installation of the RPMs.

- From this point forward, two options are provided for carrying out the *Upgrade*:
  - *Express Upgrade*: This is the faster option and all services of the Cluster will be suspended for the update;
  - *Rolling Upgrade*: The update is carried out without completely stopping the services of the Cluster.



IMPORTANT

© Copyright Tecnisys 2026. All rights reserved.

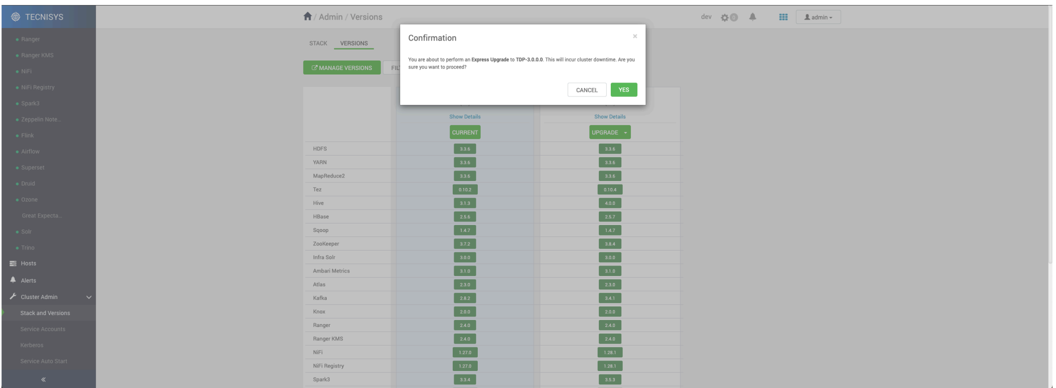
Before starting the update, ensure that all prerequisites have been met, the services are functional, and the Auto Start function is disabled in Ambari.

Execution of the Express Upgrade

The *Express Upgrade* is an option that allows the update of the Cluster components to a new version quickly and safely. However, this option requires the total shutdown of the Cluster services.

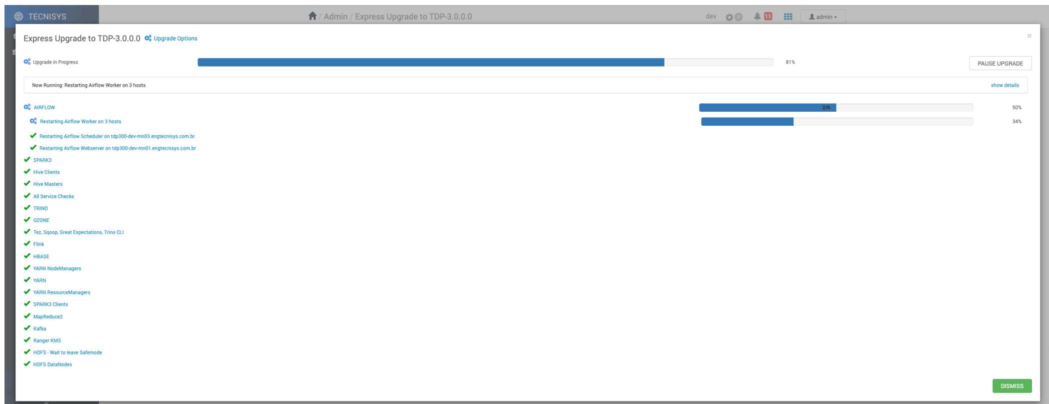
Once all prerequisites are met, follow these steps:

1. Click on the UPGRADE button of the new version.
2. Select the "Express" update option.
3. Click on "Proceed" to continue.
4. Confirm operation.



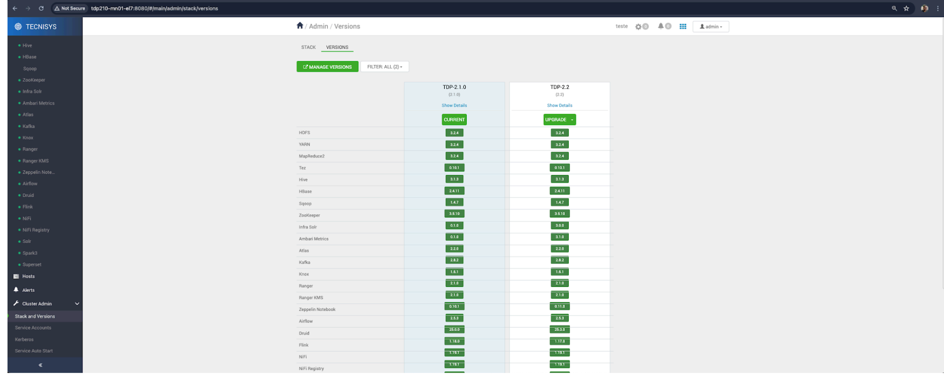
5. Monitor the update process and pay attention to the messages and requests displayed on the screen.

© Copyright Tecnisys 2026. All rights reserved.



WARNING

During the *Express Upgrade*, services are unavailable. The total downtime depends on various factors, such as the number of machines, the number of services, network speed, among others.



WARNING

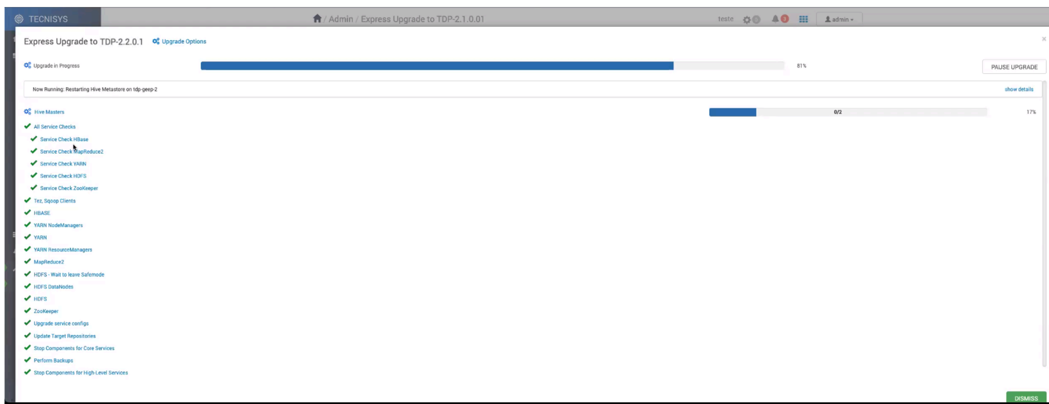
If the Apache Hive service is installed, during the update, you will be asked to perform a *backup* of the Hive Metastore database. This action is necessary to facilitate recovery in case of failure or *downgrade* of the version. Subsequently, back in Ambari, confirm the completion of the *backup* to proceed with the update.

TIP

© Copyright Tecnisys 2026. All rights reserved.

It is possible to verify whether the updated components are functional. For this, click on the "all service checks" link. This allows for the identification and correction of potential issues before the service verification stage (*service check*), carried out at the end of the update.

6. Monitor the verification of all updated services (*Service Check*).



7. Select one of the following options to conclude the update process:

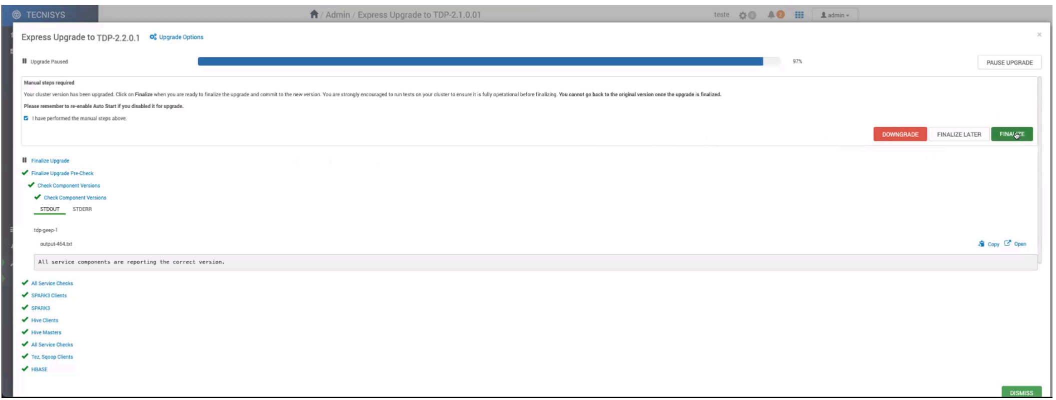
- *DOWNGRADE*: Allows reversing the update process. All components revert to their previous versions.
- *FINALIZE LATER*: Allows finalizing the update process at a later time, enabling the performance of tests and validations.
- *FINALIZE*: Finalizes the update process and sets the new version as "CURRENT".

WARNING

If Apache HBase is among the updated services, at the end of the upgrade you will be asked to confirm the deletion of the snapshot created during the process.

8. Click "Finalize" to finalize the atualization process.

© Copyright Tecnisys 2026. All rights reserved.



9. Click on "Proceed" to confirm the finalization of the update process.

Execution of the Rolling Upgrade

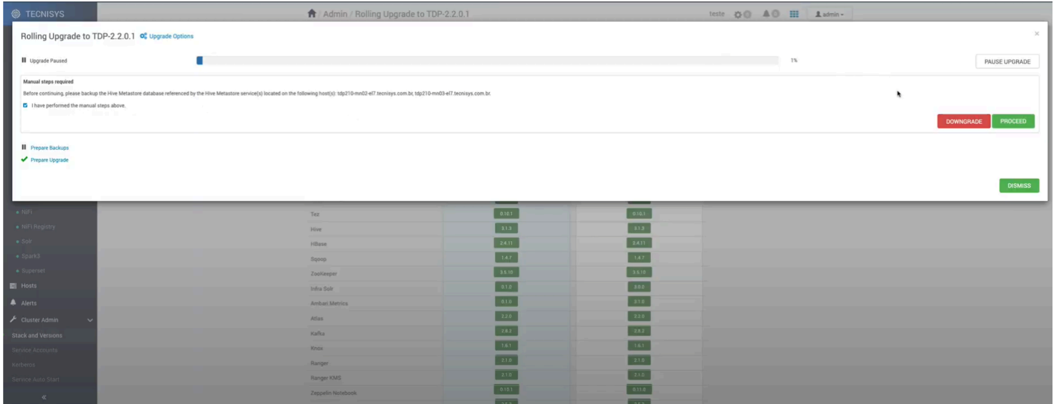
Although it is a longer option, the *Rolling Upgrade* allows the update of the Cluster components to a new version without completely interrupting the services. This is possible provided that the services are in high availability or that their components operate in a distributed architecture where there is no single point of coordination and control (*No Master*).

Once all prerequisites are met, follow these steps:

1. Click on the UPGRADE button of the new version.

2. Select and confirm the "Rolling Upgrade" option.

3. Confirm the execution of metadata database backups.



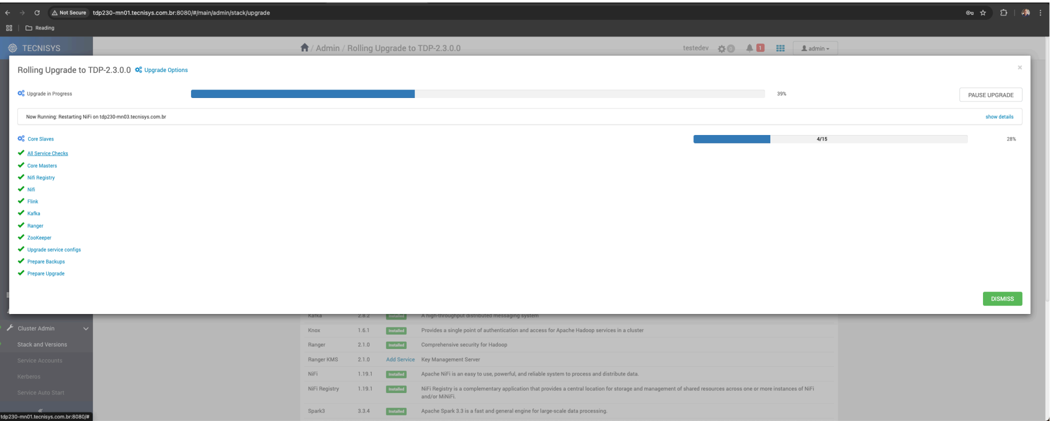
NOTE

© Copyright Tecnisys 2026. All rights reserved.

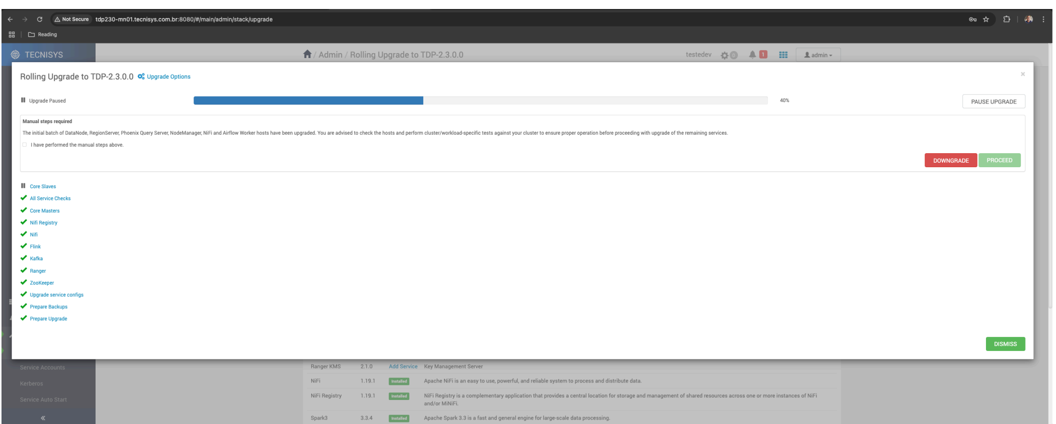
A series of settings will be automatically carried out during the update.

NOTE

At any time, it is possible to suspend the "Rolling Upgrade". For this, select the "Pause Upgrade" button. This allows you to pause the update to correct any irregularities or reverse the operation (*rollback*).



4. Confirm the completion of the manual steps (*steps*) requested.



TIP

It is possible to verify whether the updated components are functional. For this, click on the "all service checks" link. This allows for the identification and correction of potential issues before the service verification stage (*Service Check*), carried out at the end of the update.

© Copyright Tecnisys 2026. All rights reserved.

5. Monitor the verification of all updated services (*Service Check*).

6. Select one of the following options to conclude the update process:

- *DOWNGRADE*: Allows reversing the update process. All components revert to their previous versions.
- *FINALIZE LATER*: Allows finalizing the update process at a later time, enabling the performance of tests and validations.
- *FINALIZE*: Finalizes the update process and sets the new version as "CURRENT".

WARNING

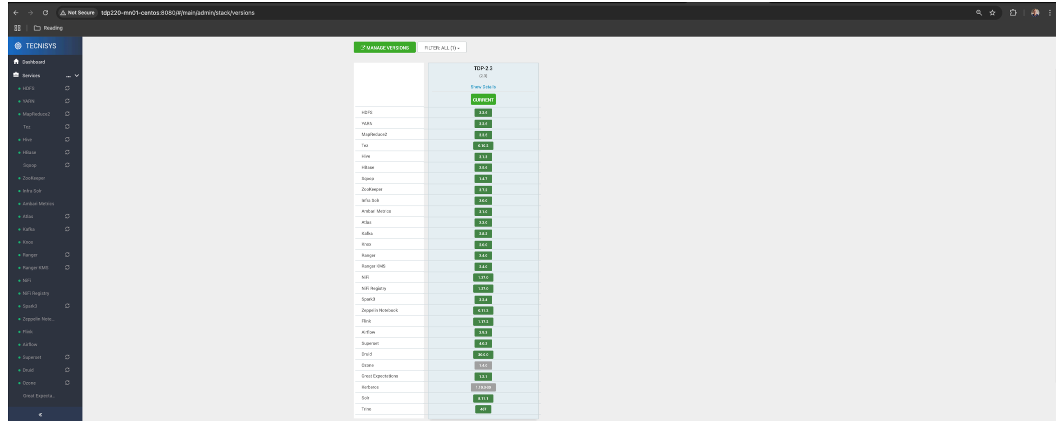
If Apache HBase is among the updated services, at the end of the *upgrade*, confirmation of the deletion of the *snapshot* created during the process will be requested.

7. Select "Finalize" to finalize the update process.

8. Click on "Proceed" to confirm the finalization of the update process.

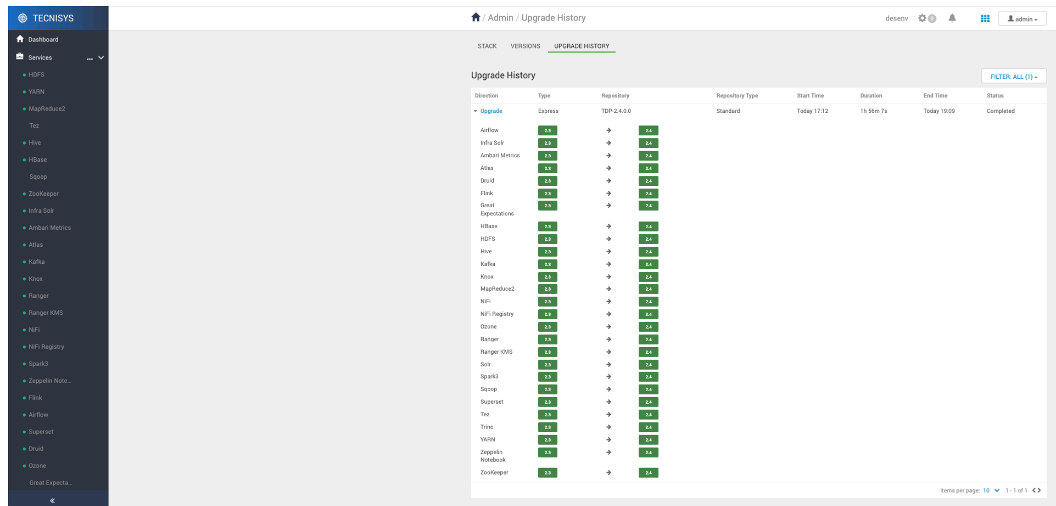
Conclusion of the Update

In the VERSIONS tab of the "Stack and Versions" page, the previous version has been removed and the new version is set as "CURRENT".

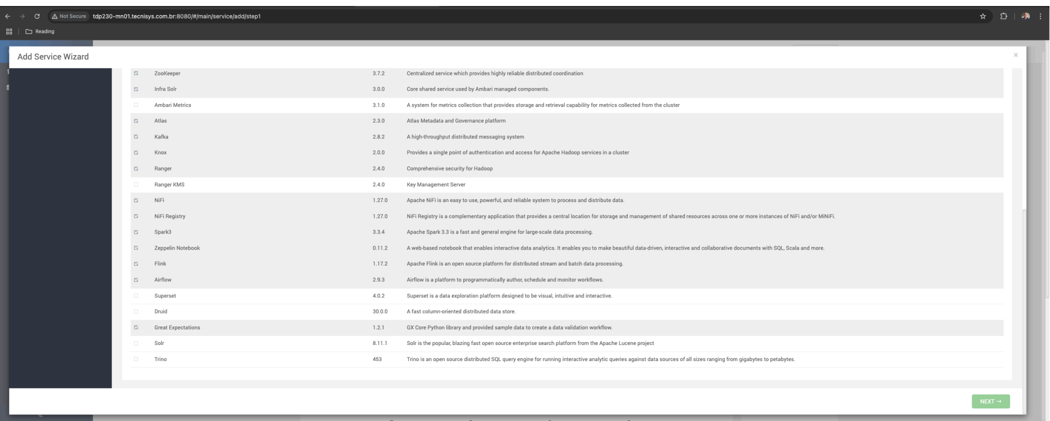


Note that the UPGRADE HISTORY tab has been added to the "Stack and Versions" page. This tab presents the update history of the cluster, including details such as the type of *upgrade* performed, the previous version, and the new version of the *stack* and its services.

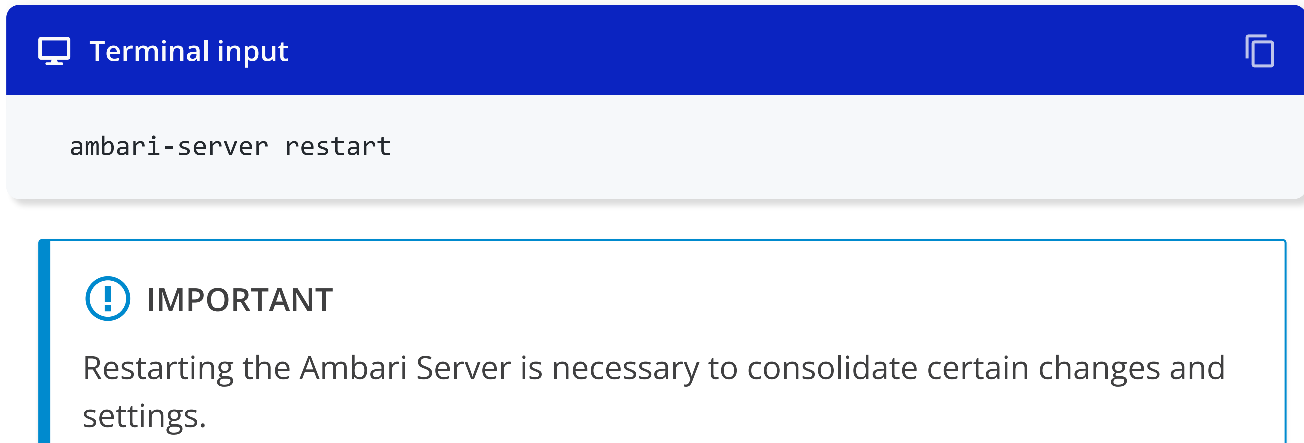
© Copyright Tecnisys 2026. All rights reserved.



Now, on the STACK tab of the "Stack and Versions" page, you can add the new services available in the new version of the TDP.



Finally, restart the Ambari Server service:


Terminal input

```
ambari-server restart
```

IMPORTANT

Restarting the Ambari Server is necessary to consolidate certain changes and settings.

© Copyright Tecnisys 2026. All rights reserved.

WARNING

After finishing, remember to reactivate the Automatic Start of Services by selecting, in the Ambari sidebar menu, the option "Service Auto-Start" and changing the "Auto Start Settings" switch to "Enabled".

© Copyright Tecnisys 2026. All rights reserved.

PART V - COMPONENTS

© Copyright Tecnisys 2026. All rights reserved.

Version Matrix

Components and versions available in TDP 2.2:

Operating System	SO Ver-sion	Archi-tecture	Component	Component Version
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Airflow	2.5.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ambari	3.0.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Atlas	2.2.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Druid	25.0.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Flink	1.17.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Hadoop	3.2.4
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache HBase	2.4.11

© Copyright Tecnisys 2026. All rights reserved.

Operating System	SO Ver-sion	Archi-tecture	Component	Component Version
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Hive	3.1.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Iceberg	1.5.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Kafka	2.8.2
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Knox	1.6.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Livy	0.7.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi	1.19.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache NiFi Registry	1.19.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Phoenix	5.1.3

© Copyright Tecnisys 2026. All rights reserved.

Operating System	SO Ver-sion	Archi-tecture	Component	Component Version
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Ranger	2.1.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Ranger KMS	2.1.0
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Solr	8.11.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Spark	3.3.4
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Sqoop	1.4.7
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Superset	2.1.3
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Tez	0.10.1
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zeppelin	0.11.0

© Copyright Tecnisys 2026. All rights reserved.

Operating System	SO Ver-sion	Archi-tecture	Component	Component Version
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Apache Zookeeper	3.5.10
CentOS, Red Hat, Rocky Linux, AlmaLinux	8.x e 9.x	x86-64	Delta Lake	2.3.0

Check the minimum requirements for installing the TDP here.

© Copyright Tecnisys 2026. All rights reserved.

PART VI - TECHNICAL NOTES

© Copyright Tecnisys 2026. All rights reserved.

Version Highlights

Platform	Version	Highlights
Tecnisys Data Platform	TDP 2.2	- Support for Enterprise Linux 8 and 9 operating systems (Rocky Linux, AlmaLinux, Red Hat, among others). - Apache Ambari with extensive support for Python 3. - Inclusion of HBase REST API. - Improvements in Hive authentication mechanism with LDAP. - Use of OpenJDK8 as the default JDK for Ambari. - Fixed issues with Pydruid library for Kerberos authentication. - Kafka-UI as a graphical interface for managing Kafka clusters. - Fixed jaas_conf of Spark for Kerberized environments. - Added hive.server2.leader.zookeeper.namespace property to Hive. - Added Rolling Upgrade option for new TDP stack versions.

© Copyright Tecnisys 2026. All rights reserved.

Platform	Version	Highlights
		- Improvements in the integration of various components with SSL/TLS enabled. - Improvements in the integration of Flink, Iceberg, and Zeppelin components. - Addition of ResourceManager UI in Knox topology. - Flink integration with Avro, JDBC, ORC, Parquet, CSV, Hive, Kafka, and Iceberg. - Spark 3.3.4 integration with Delta 2, Elastic, Iceberg, Kafka, PostgreSQL, Phoenix (HBase), and Solr. - Improvements in Ambari Metrics charts. - Updated components: Delta, Iceberg, Grafana (integrated with Ambari Metrics), Phoenix, and Zeppelin.

Table A. Platform Highlights

Component Highlights

See also the highlights of the updated components:

© Copyright Tecnisys 2026. All rights reserved.

Service	Version	Category	Highlights
Apache Ambari	Ambari 3.0.0.0	Administration	- Modernization of the interface, making it more intuitive and modern. - Improvements and bug fixes in Ambari Metrics. - Operation with Python 3 on EL 8 and 9 operating systems.
Apache Iceberg	Iceberg 1.5.0	Analytics	- Fixed issue in FileIO where an extra header request was made when reading manifests. - Marked HTTP 502 and 504 status codes as retryable in the REST Client. - Bug fixes and improvements in JDBC Catalog.
Apache Phoenix	Phoenix 5.1.3	Analytics	- Improvements in stability and bug fixes with the JDBC client. - Fixes related to transaction management to ensure greater data

© Copyright Tecnisys 2026. All rights reserved.

Service	Version	Category	Highlights
			consistency and integrity. - Query execution improvements, especially for high data volume scenarios, reducing response time and optimizing resource usage. - Improvements in handling and using secondary indexes to speed up read operations. - Updates to better support integration with Apache Spark, allowing the use of Phoenix as an SQL query layer for HBase data within Spark pipelines. - Greater compatibility with recent versions of HBase.
Apache Spark	Spark 3.3.4	Analytics, Data Science, Graph, Streaming	- Adjustments in the behavior of several SQL functions, such as percentile_disc and

© Copyright Tecnisys 2026. All rights reserved.

Service	Version	Category	Highlights
			percentile_approx (), to correctly handle NULL values. - Improvement in the behavior of SQL functions, including to_number, try_to_number, and handling of columns with GROUPING SETS. - Fixes in the behavior of expression push-down in DataSource V2 and binary comparison functions. - Improved thread management and structural integrity of CoarseGrainedExecutorBackend. - Fixes in performance issues related to TransportClientFactory and the use of await(). - Adjustments to improve compatibility with Kubernetes and

© Copyright Tecnisys 2026. All rights reserved.

Service	Version	Category	Highlights
			YARN environments, ensuring that resource allocation and token renewal are handled correctly. - Dependency updates, including the ORC version to 1.7.10.
Apache Zeppelin	Zeppelin 0.11.0	Notebook	- Default support for JDK 11, which improves compatibility and performance over previous versions. - Support for the latest versions of Apache Spark (3.5.0) and Apache Flink (1.17). - Python 3.9 is now the default version for the Python interpreter, ensuring better support and performance for Python scripts. - Dynamic creation of forms within notebooks to facilitate user interaction with data.

© Copyright Tecnisys 2026. All rights reserved.

Service	Version	Category	Highlights
			- Improvements in data visualization functionalities, including pivot charts and other interactive tools for data analysis. - Bug fixes and security improvements.

Table B. Component Highlights

© Copyright Tecnisys 2026. All rights reserved.

Added Components

The following components were added in TDP2.2

- Kafka UI

© Copyright Tecnisys 2026. All rights reserved.

Removed Components

The following components were removed in TDP2.2

- Apache Oozie

© Copyright Tecnisys 2026. All rights reserved.

FIXES

© Copyright Tecnisys 2026. All rights reserved.

Fix 202408001

Description of the Inconsistency or Failure

The update of the Chrome browser, version 127.0.6533.88/89, released on 07/30/2024, deprecated a presentation feature used by the Ambari Web component. As a result, we observed a layout break in the Ambari web interface pages when accessed through this browser.

Affected Services and Components

- Ambari 2.7.6.3: Ambari Web
- Ambari 3.0.0.0: Ambari Web

Related Issues

- AMBARI-7883
- AMBARI-7884

Impact

The central content of Ambari web interface pages is displayed below the side menu instead of next to it.

INFO

Component operations and functionalities available on the Ambari Web pages were not compromised.

Required Actions

To fix this issue, apply the corresponding fix for the affected Apache Ambari version, as per the instructions below.

Instructions

Apache Ambari 2.7.6.3

© Copyright Tecnisys 2026. All rights reserved.

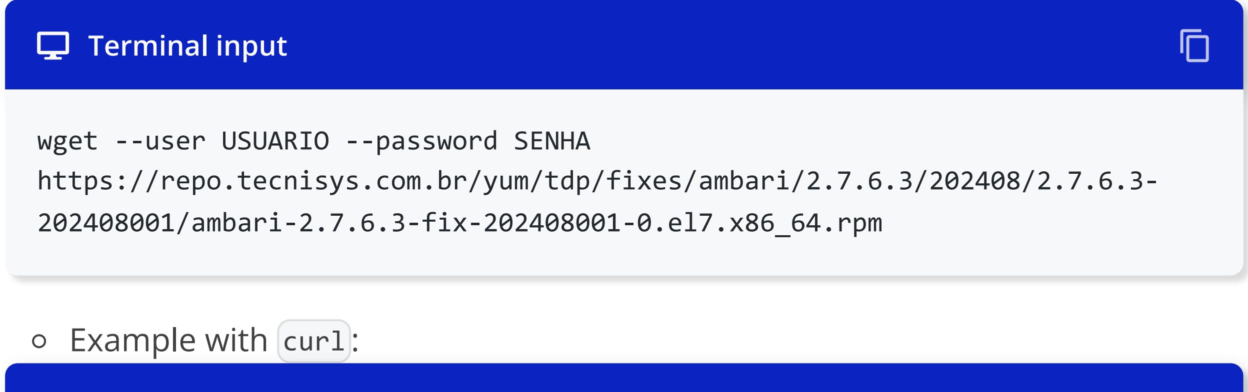
This fix resolves the layout break issue in the Apache Ambari 2.7.6.3 (Enterprise Linux 7) web interface:

Follow these steps on the Ambari Server machine to apply the fix:

1. **Download**

   Use `wget` or `curl` to download the RPM file from the Tecnisys public package repository.

   - Example with `wget`:



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/2.7.6.3/202408/2.7.6.3-
202408001/ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

   - Example with `curl`:

Terminal input

```
curl -O -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/2.7.6.3/202408/2.7.6.3-
202408001/ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

2. **Installation**

   Use `yum install` to install the downloaded fix on the Ambari Server machine:

Terminal input

```
sudo yum install ambari-2.7.6.3-fix-202408001-0.el7.x86_64.rpm
```

Apache Ambari 3.0.0.0

This fix resolves the layout break issue in the Apache Ambari 3.0.0.0 (Enterprise Linux 8 and 9) web interface:

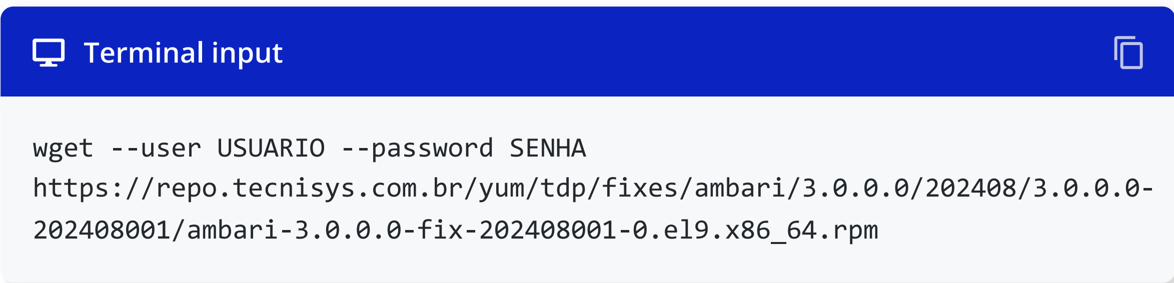
© Copyright Tecnisys 2026. All rights reserved.

Follow these steps on the Ambari Server machine to apply the fix:

1. **Download**

   Use `wget` or `curl` to download the RPM file from the Tecnisys public package repository.

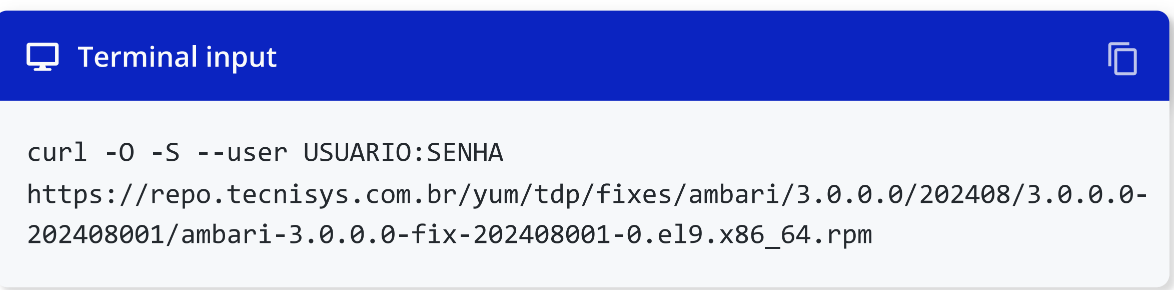
   - Example with `wget` for EL 9:



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-202408001/ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
```

   - Example with `curl` for EL 9:



Terminal input

```
curl -O -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-202408001/ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
```

2. **Installation**

   Use `dnf install` to install the downloaded fix:

Terminal input

```
sudo dnf install ambari-3.0.0.0-fix-202408001-0.el9.x86_64.rpm
```

Video with wget

Carregando...

Video with curl

© Copyright Tecnisys 2026. All rights reserved.

Carregando...

© Copyright Tecnisys 2026. All rights reserved.

Fix 202408002

Description of the Inconsistency or Failure

Apache Ambari displays the error `configparser.InterpolationSyntaxError` when attempting to download installation packages.

The issue was caused by the configparser's variable interpolation, which doesn't handle special characters in URLs well, such as `%40` (representation of @) used for authentication credentials.

Affected Services and Components

- Ambari 3.0.0.0: Ambari Web

Related Issues

- AMBARI-7907

Impact

It is not possible to download Apache Ambari installation packages when the repository URL contains special characters, thus requiring the creation of a local package repository.

Required Actions

To fix this issue, follow the instructions below.

Instructions

Apache Ambari 3.0.0.0

This fix resolves the configparser.InterpolationSyntaxError issue when downloading installation packages in Apache Ambari 3.0.0.0 (Enterprise Linux 8 and 9):

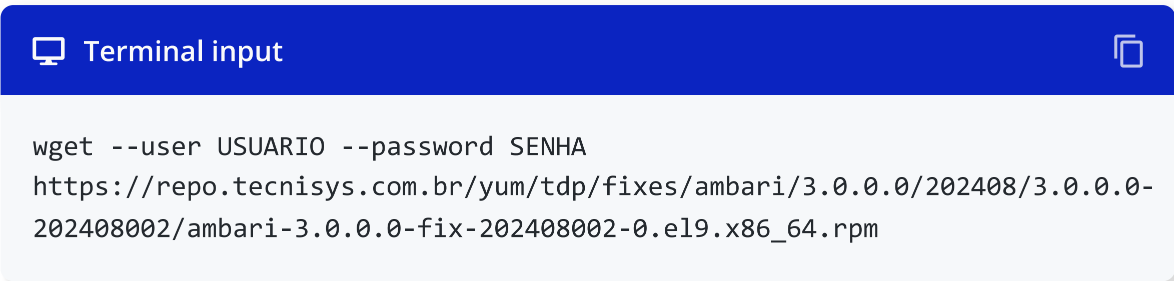
Follow these steps on all machines with Ambari Agent installed to apply the fix:

1. **Download**

© Copyright Tecnisys 2026. All rights reserved.

Use `wget` or `curl` to download the RPM file from the Tecnisys public package repository.

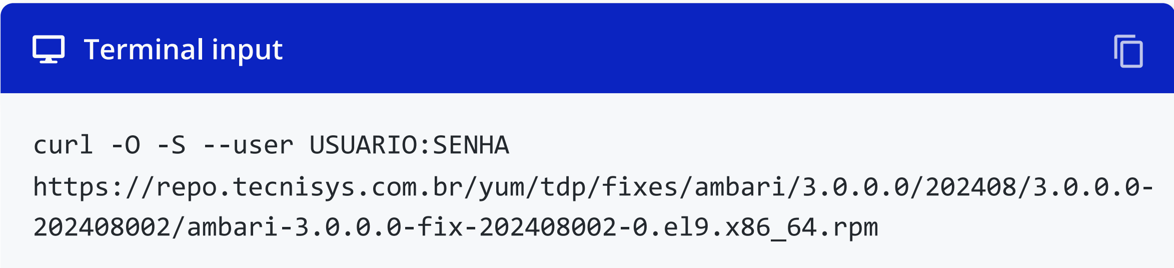
- Example with `wget` for EL 9:



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
202408002/ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
```

- Example with `curl` for EL 9:

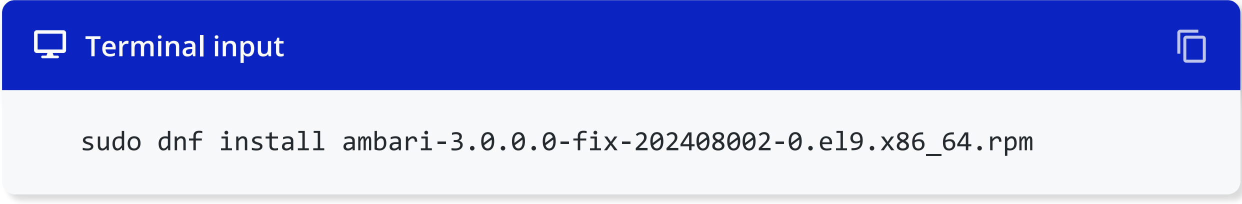


Terminal input

```
curl -O -S --user USUARIO:SENHA
https://repo.tecnisys.com.br/yum/tdp/fixes/ambari/3.0.0.0/202408/3.0.0.0-
202408002/ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
```

2. **Installation**

Use `dnf install` to install the downloaded fix:



Terminal input

```
sudo dnf install ambari-3.0.0.0-fix-202408002-0.el9.x86_64.rpm
```

Video with wget

Video with curl

© Copyright Tecnisys 2026. All rights reserved.

More Information about each TDP Platform component

The following provides more information about the implementations, fixes, and improvements made in TDP and its components:

Tecnisys Data Platform 2.2

Below are the issues resolved in this version:

Fixes:

- STACK-5599 Hive does not start automatically after changing the Active Namenode
- STACK-5609 Issues with the Scala version used by Spark interpreter in Zeppelin
- AMBARI-6929 Fix for issues in metric collection and display on EL 8 and 9 systems
- BIGTOP-7402 Pydruid does not connect with Kerberos via Superset
- STACK-7415 Error when enabling SSL in Atlas and Ranger
- AMBARI-7423 Fix for configuration issues with OpenJDK in Ambari
- STACK-7812 Spark does not generate `jaas_conf` in a Kerberized environment.

Improvements

- AMBARI-5998 Ambari updated to version 3.0.0.0
- STACK-7249 Added `hive.server2.leader.zookeeper.namespace` property to Hive
- BIGTOP-7376 Spark update
- STACK-7429 Job Schedule enabled by default in Airflow
- STACK-7430 Enriched data lineage in Atlas with metadata from Airflow
- STACK-7434 Support for Enterprise Linux 8 and 9 operating systems
- BIGTOP-7559 Improved integration of Flink, Iceberg, and Zeppelin components
- BIGTOP-7623 Updated Delta, Iceberg, Grafana, Phoenix, and Zeppelin components
- STACK-7704 Authentication configuration improvements for Hive LLAP

New Components, Features, and Functionalities

- STACK-7623 Added Rolling Upgrade option in Ambari
- STACK-7390 Kafka-UI as a graphical interface for managing Kafka clusters
- STACK-7834 Added HBase Rest API to Ambari

Apache Airflow

© Copyright Tecnisys 2026. All rights reserved.

Version: 2.5.3

- Release Notes
- GitHub

Apache Ambari

Version: 3.0.0.0

- Release Notes
- Jira
- GitHub

Apache Atlas

Version: 2.2.0

- Release Notes
- Jira
- GitHub

Delta-Lake

Version: 2.3.0

- Release Notes
- GitHub

Apache Druid

Version: 25.0.0

- Release Notes
- GitHub

Apache Flink

© Copyright Tecnisys 2026. All rights reserved.

Version: 1.17.2

- Release Notes
- Jira
- GitHub

Apache Hadoop 3.2.4

- Release Notes
- Hadoop HDFS Jira
- Hadoop YARN Jira
- Hadoop MapReduce Jira
- Hadoop Common Jira
- GitHub

Apache HBase

Version: 2.4.11

- Release Notes
- Jira
- GitHub

Apache Hive

Version: 3.1.3

- Release Notes
- Jira
- GitHub

Apache Iceberg

Version: 1.5.0

- Release Notes
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Kafka

Version: 2.8.2

- Release Notes
- Jira
- GitHub

Apache Knox

Version: 1.6.1

- Release Notes
- Jira
- GitHub

Apache NiFi

Version: 1.19.1

- Release Notes
- Jira
- GitHub

Apache Phoenix

Version: 5.1.3

- Release Notes
- Jira
- GitHub

Apache Ranger

Version: 2.1.0

© Copyright Tecnisys 2026. All rights reserved.

- Release Notes
- Jira
- GitHub

Apache Solr

Version: 8.11.1

- Release Notes
- Jira
- GitHub

Apache Spark

Version: 3.3.4

- Release Notes
- Jira
- GitHub

Apache Sqoop

Version: 1.4.7

- Release Notes
- Jira
- GitHub

Apache Superset

Version: 2.1.3

- Release Notes
- Jira
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

Apache Tez

Version: 0.10.1

- Release Notes
- Jira
- GitHub

Apache Zeppelin

Version: 0.11.0

- Release Notes
- Jira
- GitHub

Apache Zookeeper

Version: 3.5.10

- Release Notes
- Jira
- GitHub

© Copyright Tecnisys 2026. All rights reserved.

PART VII - ROADMAP

© Copyright Tecnisys 2026. All rights reserved.

Roadmap

This topic presents the highlights of the TDP versions, including the features planned for the upcoming semesters.

Current Version

- **Jupyter added:** Additional interactive notebook service.
- **OpenMetadata added:** Unified metadata management service.
- **ClickHouse added** [1]**:** OLAP database service for real-time data analysis.
- **TDP Kubernetes:** Cloud Native edition.
- **Component updates:** Latest versions of all components.

Planned Versions

- **HUE added:** Web interface for data exploration, querying, and management.
- **MLflow added:** Service for AI/LLM model lifecycle governance.
- **Component updates:** Latest versions of all components.

TDP 3.2 | 2027-S1

- **Milvus added:** High-performance vector database service for AI applications.
- **Agno added:** Framework for building multi-agent systems.
- **Component updates:** Latest versions of all components.

TDP 4.0 | 2027-S2

- **Tecnisys Data eXperience:** *Low code* layer for building BI, Data Science, and AI projects.
- **Component updates:** Latest versions of all components.

© Copyright Tecnisys 2026. All rights reserved.

Previous Versions

TDP 2.3 | 2025-S1

- **Trino added:** Data virtualization service.
- **Apache Ozone added:** Distributed and highly scalable object storage service.
- **Great Expectations added:** Data quality analysis service.
- **Component updates:** Latest versions of all components.

TDP 2.2 | 2024-S1

- **Kafka-UI added:** Graphical interface for monitoring and managing Kafka clusters.
- **Extended operating system support:** CentOS/RHEL 8+ and 9+, Rocky Linux 8+ and 9+, and AlmaLinux 8+ and 9+.
- **Component updates:** Latest versions of all components.

TDP 2.1 | 2023-S2

- **Apache Iceberg added:** Advanced table format for analytics requiring high performance with large data volumes.
- **Apache Flink added:** Distributed stream processing service.
- **Apache Ranger KMS added:** Encryption key management service.
- **Component updates:** Latest versions of all components.

TDP 2.0 | 2023-S1

- **Apache Airflow added:** Pipeline orchestration service.
- **Apache Druid added:** Distributed OLAP service.
- **Apache Superset added:** Data exploration and visualization service.
- **Delta Lake added:** Table format with full transactional support.
- **Component updates:** Latest versions of all components.

TDP 1.0 | 2022-S2

© Copyright Tecnisys 2026. All rights reserved.

- **Big Data platform based on the Hadoop ecosystem:** With integrated and centralized installation, configuration, and monitoring.
- **Exclusive package repository with direct dependencies:** Tecnisys Public Package Repository.
- **Operating system compatibility:** CentOS/RHEL 7.5

Documentação v3.0

Footnotes

1. Exclusive to the TDP Kubernetes edition. ↩

PART VIII - OTHER INFORMATION

© Copyright Tecnisys 2026. All rights reserved.

Support

How to get support for TDP?

Tecnisys offers basic and free support to all users registered on the Tecnisys website.

Register now to access our Support Area.

If your business requires priority support, SLAs, 24x7 technical availability, first response within 15 minutes, advanced monitoring, risk analysis, incident prevention, consulting, and other specialized services, contact us.

How to report a failure in TDP?

In the support area , you can open a ticket to report the failure and track its resolution.

What is the Tecnisys Support Agent (TSA)?

The Tecnisys Support Agent is a service installed alongside Apache Ambari to facilitate the identification of machines and data environments subscribed at the time of opening tickets in the support area .

The TSA service runs in the background and does not interfere with the performance of the operating system or the database. Additionally, this service does not require a direct Internet connection.

The machine identification file (*subscription.info*), generated daily by the TSA, is kept securely and encrypted in the */opt/tecnisys-support-agent* directory.

How to install the Tecnisys Support Agent (TSA) separately?

Package Download

Instructions

To install the Tecnisys Support Agent separately, follow the steps below:

1. Register for free on the Tecnisys website;

© Copyright Tecnisys 2026. All rights reserved.

2. Access the Tecnisys Public Package Repository;

3. Click the *Sign out* button located in the upper right corner of the page;

4. Enter your access credentials (the same as the Tecnisys website);

5. Click the *Browse* option located in the left sidebar;

6. In the central navigation area, access: *yum >> tecnisys >> tsa*

7. Select the installation package (`.rpm`) of the latest version available for your operating system;

8. Click the *Path* property link located in *Summary* on the right side of the page to start downloading the package.

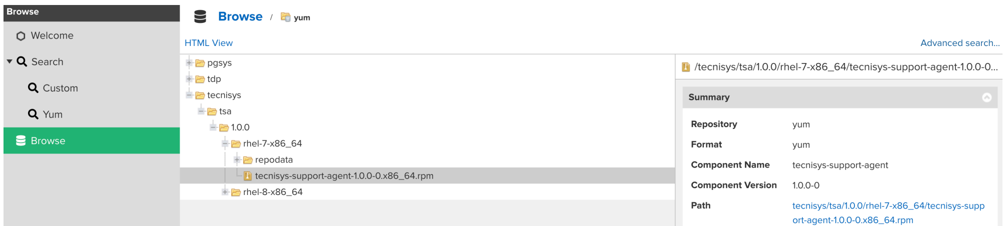


Figure 1 - Download the TSA installation package

Package Installation

Instructions

1. Install the TSA installation package on the subscribed machine, as shown in the example below:

```
rpm -ivh tecnisys-support-agent-1.0.0-0.x86_64.rpm
```

2. After the installation, run the following command to enable and start the TSA service:

© Copyright Tecnisys 2026. All rights reserved.

```
systemctl enable --now tecnisys-support-agent.service
```

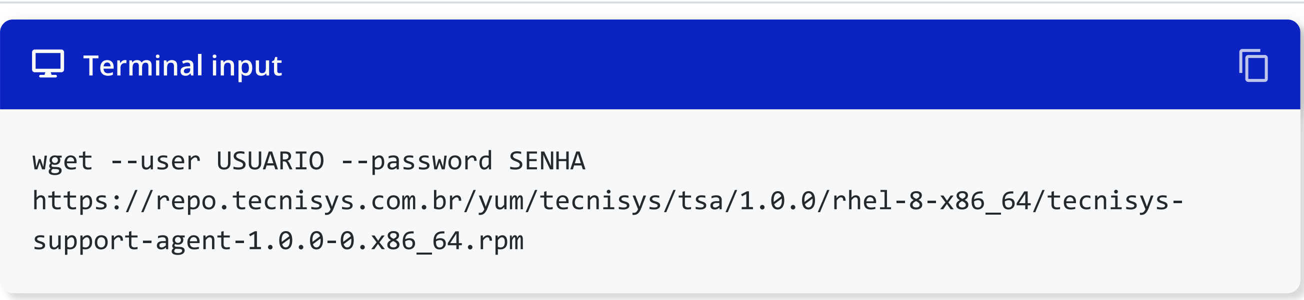
3. To check the status of the TSA service, run the following command:

Terminal input

```
systemctl status tecnisys-support-agent.service
```

It is also possible to download the TSA installation package directly from the terminal via curl or wget. See the example below:

Instructions



Terminal input

```
wget --user USUARIO --password SENHA
https://repo.tecnisys.com.br/yum/tecnisys/tsa/1.0.0/rhel-8-x86_64/tecnisys-
support-agent-1.0.0-0.x86_64.rpm
```

© Copyright Tecnisys 2026. All rights reserved.

Legal Information

This documentation is maintained by Tecnisys Informática e Assessoria Empresarial LTDA.

TDP (Tecnisys Data Platform) is a registered trademark of Tecnisys.

Tecnisys does not provide representations for its products.

TDP (Tecnisys Data Platform) incorporates software components from various communities, especially those under the terms of the Apache Software License. Other components may be incorporated under alternative terms. This information can be obtained directly from the license files available on each community's website.

For details on the components and services that make up Tecnisys' service portfolio, visit our support and sales page or contact us.

The use of the software and components described herein is subject to the permissions and limitations defined by the Apache Software License.

The information contained herein is subject to change without notice. Tecnisys reserves the right to include, exclude, or change any component described herein at any time and without notice.

Tecnisys disclaims any liability, warranty, or condition of any kind, express or implied, arising from the use of the products described herein, except as defined by law or expressly defined in a specific bilateral written contract. Nothing herein shall be construed as constituting an additional warranty.

Tecnisys does not guarantee the absence of interruptions, defects, errors, technical damages, editorial errors, protection against loss, corruption or unavailability of data, omissions, except as expressly defined in a specific bilateral written contract.

Copyright information can be found in the documentation accompanying each component and its specific version.

Contact

© Copyright Tecnisys 2026. All rights reserved.

Site: www.tecnisys.com.br

E-mail: faleconosco@tecnisys.com.br

Phone: +55 61 3039-9700

Address: Trecho 8, Lote 245, SIA, Brasília, DF, Brasil, CEP 71205-080

© Copyright Tecnisys 2026. All rights reserved.