REPUBLIC OF TURKEY

AKDENIZ UNIVERSITY

BUILDING OF A LINUX BASED LIGHTWEIGHT OPEN SOURCE BIG DATA DISTRIBUTION

Muhammed Numan INCE

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JANUARY 2020

ANTALYA

REPUBLIC OF TURKEY

AKDENIZ UNIVERSITY

BUILDING OF A LINUX BASED LIGHTWEIGHT OPEN SOURCE BIG DATA DISTRIBUTION

Muhammed Numan INCE

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JANUARY 2020

ANTALYA

REPUBLIC OF TURKEY

AKDENIZ UNIVERSITY

INSTITUTE OF NATURAL AND APPLIED SCIENCES

BUILDING OF A LINUX BASED LIGHTWEIGHT OPEN SOURCE BIG DATA DISTRIBUTION

Muhammed Numan INCE

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis unanimously accepted by the jury on 03/01/2020.

Prof. Dr. Melih GÜNAY (Supervisor)

Asst. Prof. Dr. Joseph LEDET

Asst. Prof. Dr. Umut TOSUN

ÖZET

LİNUX TABANLI HAFİF VE AÇIK KAYNAK KODLU BÜYÜK VERİ DAĞITIMI GERÇEKLEMESİ

Muhammed Numan İNCE

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Prof. Dr. Melih GÜNAY

Ocak 2020; 93 sayfa

Bu tez çalışması, Linux işletim sistemindeki veri mimarilerinin doğru kurulum, yapılandırma ve optimizasyon işlemlerini içerir. Çekirdek olarak alacağımız ve özelleştireceğimiz işletim sistemi, şu anda BAP (Bilimsel Araştırma Projeleri) tarafından desteklenen Milis Linux projesini taban alacaktır. Bu projede kullanılacak büyük veri platformu yaygın olarak kullanılan ve iyi bilinen Apache Hadoop dosya sistemini uyarlar. Apache Hadoop platformu, temel programlama çerçevesine sahip bilgisayar ve disk kümelerine büyük miktarda veri dağıtılmasını sağlar. Apache Hadoop'un üstüne topluluk tarafından işletilen veri işleme, güvenlik, erişim yönetimi, performans izleme ve diğer yardımcı yazılımlar kurulması bu tez kapsamında gerçeklenecektir. Kararlı bir Büyük Veri işleme yazılımı üretmek ve dağıtım yapmak için, yazılım paketlerinin en yeni ve uyumlu sürümlerinin bir araya getirilmesini sağlayan bir sistem kurulmalıdır. Bu tür yazılım seçim sürecinin geliştirilmesi ve otomasyonu ve birim testleriyle doğrulanması bu araştırmanın kapsamıdır. Bu araştırma dahilinde bir araya toplanan uygulamaların hazır kurulu olduğu bir sistem imajı hazırlanarak kullanıma hazır hale getirilecektir. Bu imaj canlı ortamda test edilebilir olduğu gibi sanal/fiziksel disklere kurulumu mümkün olacaktır.

ANAHTAR KELİMELER: Açık Kaynak, Büyük Veri, Linux, Milis İşletim Sistemi, Veri İşleme

JÜRİ: Prof. Dr. Melih GÜNAY
Dr. Öğr. Üyesi Joseph LEDET
Dr. Öğr. Üyesi Umut TOSUN

ABSTRACT

BUILDING OF A LINUX BASED LIGHTWEIGHT OPEN SOURCE BIG DATA DISTRIBUTION

Muhammed Numan INCE

MSc Thesis in Computer Engineering
Supervisor: Prof. Dr. Melih GUNAY
January 2020; 93 pages

The thesis covers the proper installation, configuration and optimization processes of the big data architectures for Linux operating system. The operating system that we will take as a core and customize is based on MILIS Operating System which is currently supported by Akdeniz University Scientific Research Projects Coordination Unit (BAP). Data platform and file system that is planned to be used in this project is well known and widely adapted Apache Hadoop. Apache Hadoop platform enables large volumes of data to be distributed across cluster of computer and disks with a basic programming framework. On top of Apache Hadoop, community-run data processing, security, access management, performance monitoring and other utility software is installed. In order to produce a stable Big Data processing software and distribution, a system should be put in place that includes the latest and compatible versions of the software packages to be assembled together. Development and automation of such software selection process and verification through unit tests is part of this research. In this research, a system image that contains the widely used big data modules is prepared for installation on virtual/physical disks.

KEYWORDS: Big Data, Data Processing, Linux, MILIS Linux, Open Source

COMMITTEE: Prof. Dr. Melih GUNAY
Asst. Prof. Dr. Joseph LEDET
Asst. Prof. Dr. Umut TOSUN

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Dr. Melih GUNAY who helped, guided and motivated me during this study.

I would like to thank all my teachers who trained me to serve for our country.

Special thanks to my family for their big love and support.

LIST OF CONTENTS

TEXT OF OATH

I declare that this study "Building Of A Linux Based Lightweight Open Source Big Data Distribution", which I present as master thesis, is in accordance with the academic rules and ethical conduct. I also declare that I cited and referenced all material and results that are not original to this work.

03/01/2020

Muhammed Numan INCE

ABBREVIATIONS

ACID	: Atomicity Consistency Isolation Durability
AWS	: Amazon Web Services
CDH	: Cloudera's Distribution Including Apache Hadoop
CLI	: Command Line Interface
CM	: Cloudera Manager
CQL	: Cassandra Query Language
EDH	: Enterprise Data Hub
ETL	: Extract Transform Load
GCP	: Google Cloud Platform
GNU	: Gnu Not Unix
HDFS	: Hadoop Distributed File System
HQL	: Hive Query Language
JVM	: Java Virtual Machine
LFS	: Linux From Scratch
ML	: Machine Learning
MPP	: Massive Parallel Processing
MPS	: MILIS Package System
NoSQL	: Not only Standardized Query Language
OLAP	: Online Analytical Processing
OLTP	: Online Transactional Processing
RDBMS	: Relational Database Management System
SQL	: Standardized Query Language

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

With the development of technology, all kinds of devices connected to the Internet have become data producers and therefore the concept of big data has become increasingly widespread. Data and processing are now increasingly taking place on the cloud systems; because of the high availability of the servers, their ease of creation, configuration and management, and decreased cost of capacity and CPU speed. As the volume of data in the cloud grows, the storage and processing of the data comes with some challenges. Therefore, these processes require data-driven professional environments for the handling of big data. For this purpose, open source Apache Hadoop and its add-on components are used frequently in cloud systems. The installation of the Hadoop infrastructure and its compatibility with other components is only possible through the proper deployment process. The package manager of the operating system in which it is hosted, ensures that the compatible versions of the Big Data frameworks and applications are installed and updated together.

Big Data refers to large data, structured or unstructured which traditional processing algorithms and/or techniques are unable to handle on (Taylor and Sakyi 2016). With technological advances and the rise of the Internet, scattered data, hitherto known as piles of information, has become more important. Software companies and scientific teams, who think that meaningful data can come out of this dump, have concentrated their R&D efforts on this issue and revealed what we call Big Data.

The term of Big Data actually refers to the way that all of these data are generated in different formats such as log files, photo/video archives, social media shares, and other records that we continuously record. Today, database experts classify existing data associated with them in relational databases. Company executives also make decisions with reports generated through reporting systems running in these databases. But there are many more data sets that we cannot fully classify and relate to. Until now, all of this information has been called a garbage dump, because it is very difficult to store this data in the databases.

Characteristics of big data are (IBM Big Data Hub, 2019):

- Volume – Size of data determines whether the data should be treated as big data

- Variety – It specifies structured or unstructured states of different types of data. According to previous approaches, databases and electronic documents are the only data source for applications, but today data analysis applications are based on data that can produce in different types and formats. This diversity of unstructured data brings some aspects for storing, mining and analyzing data.

- Velocity – It refers to the speed of generation of data. Big data velocity is related to social media content, business processes, application logs, sensor data etc. data sources. Data flow is continuous and intensive.

- Veracity – This indicates situations where data is inconsistent. Therefore, it prevents the process of using and managing data effectively.

Data is valuable, but only if it can be protected, processed, understood, and acted upon. The goal of harnessing big data is to offer real-time information that you can use to improve your business. Real-time information processing is one of the major goals for companies attempting to deliver value to their customers in a consistent and seamless manner and is one of the crucial features of edge computing. Insights from big data could allow you to cut costs, operate more efficiently, and discover new ways to boost profits and reach new customers.

A business or system may capture huge amounts of data from just about any source and make it possible to prepare for analysis for opportunity mining and business development by:

- cost reduction,

- operation streamlining,

- development of new products and services,

- decision support.

Big data combined with highly efficient and powerful analytic can be a tremendous boost for any business and can help to find root causes of setbacks and failures, discover customer buying habits and determine next steps to boost sales further and even discover fraudulent behaviour within the organization itself. That being said, stored data is only as

useful as the measures focus to ensure the integrity of the data, not to mention its security. This has become more of an issue in recent years, and there are several steps that can be taken in this direction to ensure data integrity and security starting from the operating system level.

Linux is an open source operating system that is widely used. Linux kernel is the core software that manages the hardware. Kernel and the user application space programs form the basis of the Linux Operating System. Linus Torvalds, while a student at the University of Helsinki, started this project as a hobby in 1991, but the development of kernel being as free and open source made it popular. With wide spread contribution, it became an alternative to Minix that is being used in academic environments. In fact, the term Linux is an inclusive term as all the components, programs, tools and services necessary for a functional operating system are included in this term. While some Linux distributions come with GNU (Gnu Not Unix) tools and call themselves GNU/Linux, others use different toolsets such as Android OS.

Economically, large data sets are collected and processed by scaling hardware. Here Linux is advantageous due to its clustering, high availability and I/O capabilities. This is what makes Linux a leader years ago in high performance computing (Bridgwater 2013). Linux kernel is open source/free and promotes similar licenses for the applications that run on it. Therefore, cloud services on Linux often have hardware costs rather than software. However, there are also cloud services operating in a closed-source model based on Linux ecosystem.

Structurally, Linux's open source design makes it possible to develop new techniques and algorithms to increase the computational power required. Other open-source architectures provide the flexibility to work together and allow computer resources to be collected to provide large amounts of data. Linux's modular design supports many devices, architectures and use cases. Hereby, publications from industry and academia claim Linux is the sensible platform for cloud computing (Bridgwater 2013).

Linux based cloud services became the frequent choice for Big Data storage and processing. Major public cloud providers such as from Microsoft Azure, Google Cloud Platform (GCP) and Amazon Web Services (AWS) has different flavors of Linux available. About 30 percent of the virtual machines that Microsoft Azure uses are based on Linux

platform (Nuggets 2018).

In Turkey, the research in the field of Big Data processing and management is new. Therefore, the management of these data storage and processing platforms are left to the global cloud server providers. At present, global companies that produce open source integrated solutions for Big Data are also purchased and privatized. For example, RedHat, which provides infrastructure service support for Linux operating system, was purchased by IBM. Likewise, HortonWorks, the company that provided the Big Data frameworks and technologies, hence ecosystem of Hadoop platform was purchased by its rival, Cloudera.

Considering the need of a Linux based open source Hadoop platform and software distribution, we proposed this study. We are motivated by the fact that; the market share of open source big data technologies likely to increase exponentially.

The thesis first explains the concept of big data platform basics and presents the state of art. Next, the building of a lightweight big data platform based on MILIS Linux distribution using Hadoop is reported. All components of platform is chosen to be open source.

2. LITERATURE REVIEW

2.1. Big Data Architecture

Processing of Big Data requires the right and compatible versions of the open source software from various sources to work in harmony with the correct configuration. Scalability and speed of processing depends on fine tuning of the memory and file I/O access of modules properly.

The following Figure. 2.1 shows the logical components that fit into a Big Data architecture.

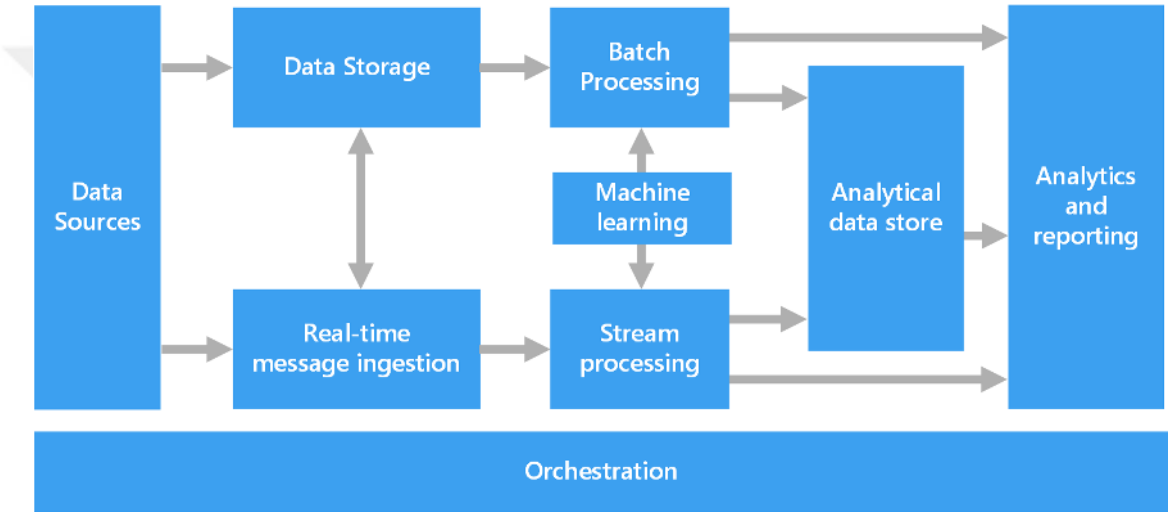


Figure 2.1. Big Data Architecture Components (Microsoft 2018)

Building blocks/modules of the Big Data software include the following;

- Service Monitoring and Coordination
- Data Streaming and Flow
- Data Cleaning and Ingestion
- Distributed File System and Storage
- Query Execution Engine
- Machine Learning and Real-Time Analytics
- Security

The relevant categories and the tools used in these categories are discussed in detail below.

2.1.1. Distributed File System

In Big Data, multiple computing nodes are important to handle data distributively. The idea of big data is based on how the system can be deployed and go beyond a single supercomputer. So distributing of big data means data across multiple nodes/clusters and to make use of computing power of each cluster node to process data. File system approach refresh itself with this distributing feature that is a system which can handle accessing of data across multiple clusters computers (Mahmoud 2017).

- **HDFS**: The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It is similar to existing distributed file systems. Main features of HDFS are highly fault-tolerant and deploy-able on low-cost hardware. HDFS is also good at high throughput accessing large data sets.

  HDFS has a master/slave architecture. NameNode(master) is responsible for declaring which node will send data or which node contains what is being looked for. The client can then connect to the DataNode (slave) and begin transferring data without being affected by the NameNode (Hadoop 2019). Figure 2.2 shows the HDFS architecture.

  HDFS applies write once read many models. So we cannot edit files already stored in HDFS, but we can append data by reopening the file. In Read-Write operation client first, interact with the NameNode. NameNode provides privileges so, the client can easily read and write data blocks into/from the respective datanodes.

- **MapReduce**: It is a programming approach for processing of large data. MapReduce programs are parallel in nature, thus instead of using very high-end servers to process huge data, MapReduce performs the same operation more efficiently on a cluster of commodity hardware. MapReduce has two programming stages: Map and Reduce. In the Map stage, the master node gets the data and breaks it down into smaller pieces and distributes it across to slave nodes. As the slave nodes complete

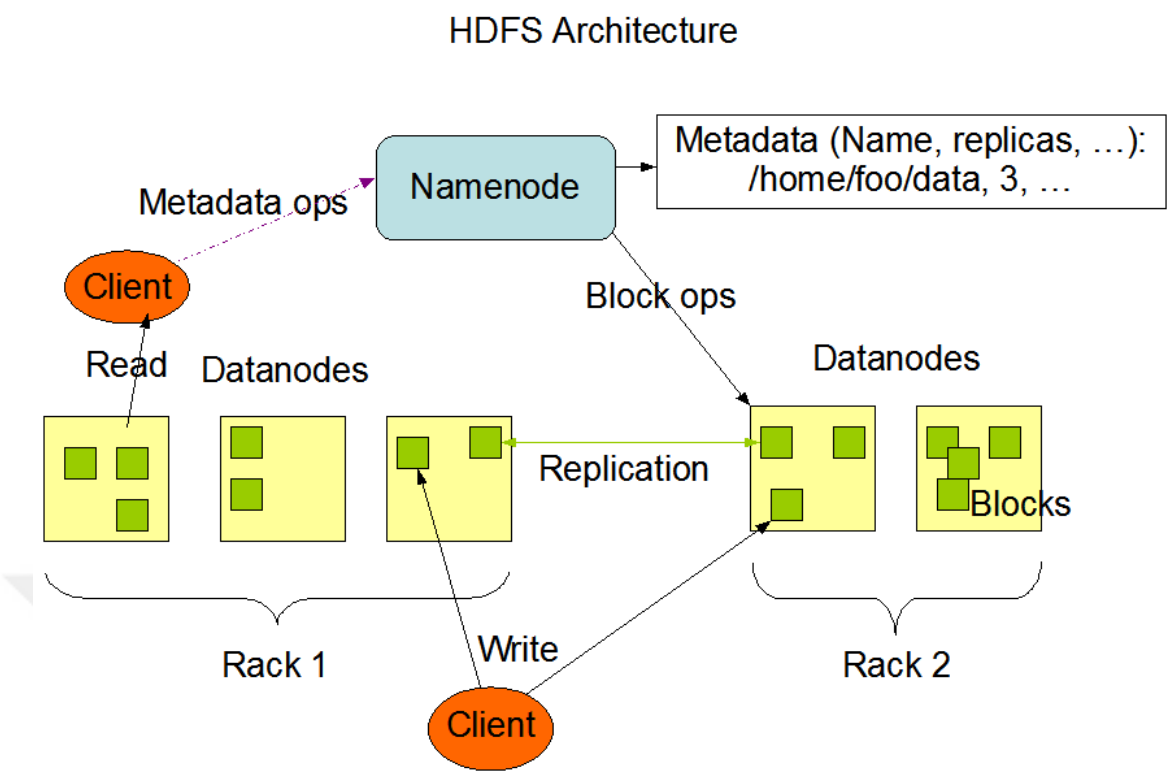


Figure 2.2. HDFS Architecture (Hadoop 2019)

these jobs, they return the result back to the master node. In the Reduce stage, the results obtained is combined based on the logic of the work.

If we simplify it, the data we want to take from the data analyzed in Map stage is drawn, and the analysis we want takes place on the data we take in Reduce stage.

In relational database systems (RDBMS); Map stage is like selecting fields according to select clause and filtering with where, Reduce stage is like making calculations on related data such as count, sum, having.

Map operations can be carried out independently of each other in parallel. In this way, large amounts of data can be read quickly by the nodes in the cluster simultaneously. That is, the more nodes in the cluster, the faster the process is. In the Reduce stage, data with the same key can be processed in parallel. Figure 2.3 shows that how MapReduce runs on HDFS. Hadoop also supports various programming languages for MapReduce.

- **Yarn**: It is the resource management and job scheduling component of Hadoop

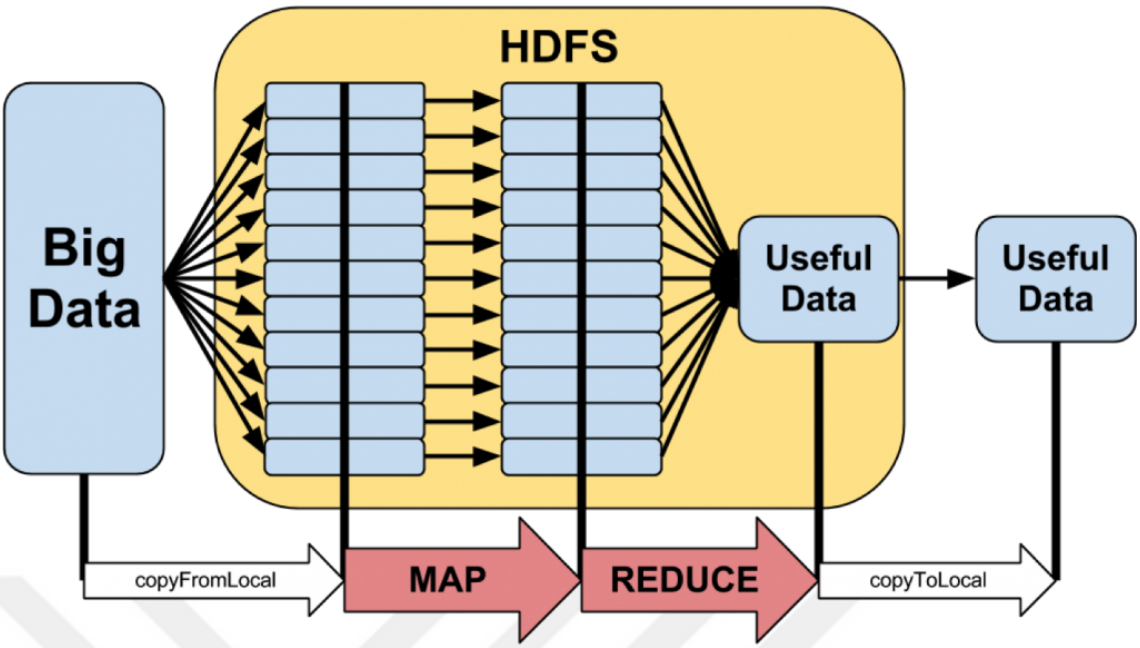


Figure 2.3. MapReduce Workflow (B3LAB 2016)

framework. It comes with built-in with Hadoop that manages system resources according to various applications running on Hadoop cluster and schedules tasking on different cluster nodes. Figure 2.4 shows the YARN Architecture. YARN Architecture has these core component.

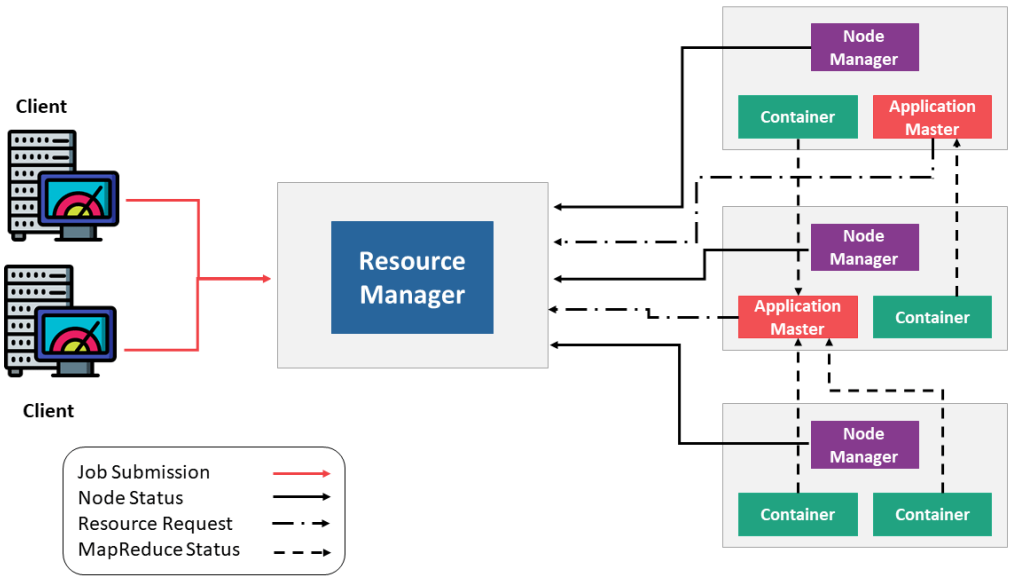


Figure 2.4. Yarn Architecture (Edureka 2018)

- Resource Manager: It does resource allocation in the cluster and runs on master node.
- Node Manager: It does execution of a task on slave(worker) nodes.
- Application Master: It manages user job lifecycle and resources.
- Container: It includes resources like that CPU, RAM, network etc.

Together with the YARN technology, MapReduce came as a decent engine that can run on other nodes. In most cases, users are replacing it with Spark to achieve faster performance in batch applications such as debugging, converting, and loading jobs. (Rouse 2018). Spark can also run stream processing applications on Hadoop clusters through YARN. YARN has also support for different databases and other SQL-on-Hadoop query engines. YARN offers resource utilization, high availability, scalability and performance improvements over MapReduce for more application support.

Figure 2.5 shows that how an application works on Hadoop:

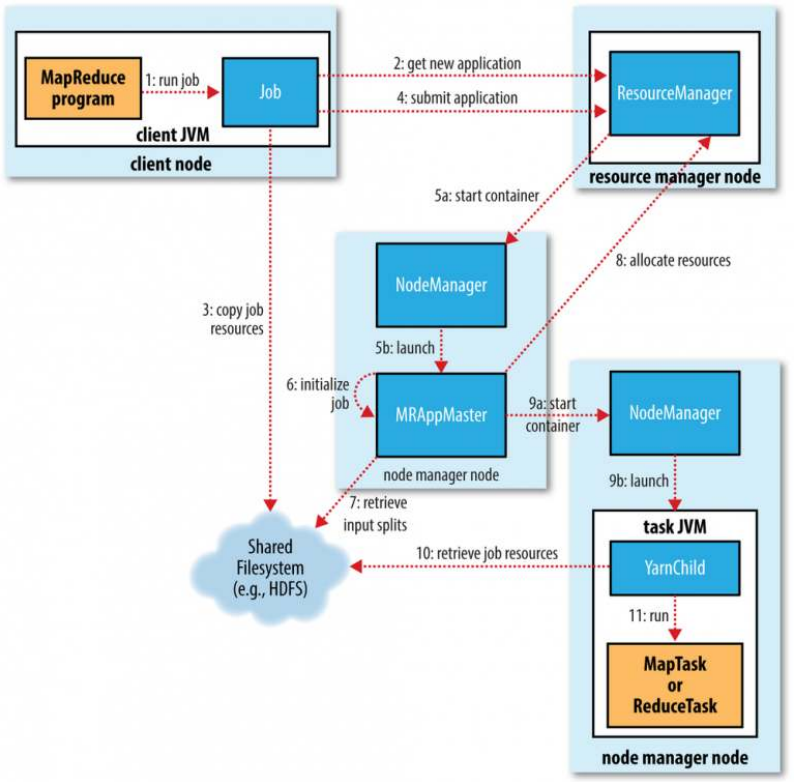


Figure 2.5. Running of an Application on Hadoop (White 2015)

2.1.2. Execution Engines

Execution Engine is used to communicate with Hadoop name and data nodes with job tracker to execute Hive Query on HDFS. It executes the execution plan that is created by the Query compiler.

- **TEZ**; uses complex directed acyclic graph (DAG) of tasks for data processing. It is currently built on YARN and uses it as resource manager (TEZ 2019, Shenoy 2014). Tez implements the MapReduce paradigm by expressing computations as dataflow graph. Traditionally, Hadoop handles large amounts of data by batch processing. When query processing with near-real-time performance is needed, MapReduce falls short. Tez addresses these uses cases by customization.

  Apache Hive and Pig get notable improvements in response times when they use Tez over MapReduce as the excution engine. Input, Processor and Output modules are core components of Tez to model user logic at each vertex of the dataflow graph. Data format is determined by Input & Output modules based on where it is written/read. The Processor keeps the data transformation logic. Tez only requires that Input, Processor and Output formats are compatible with each other. Tez has simple Java API to express a DAG of data processing (Cloudera 2019). The API has three components:

  - DAG – Each data processing job has a DAG object.
  - Vertex – User logic, resources and environment is defined in vertex. Vertex object is defined for each step in the job and added to DAG
  - Edge – Connection of consumer and producer vertices.

  In Tez, a job is divided into individual tasks on behalf of users, each task runs as a process through YARN. This approach brings some inherent costs for process start such as allocation of containers through the YARN resource manager Figure 2.6.

- **SPARK**: Apache Spark is designed for open-source cluster computing framework that handles real-time data processing. Spark increases the processing efficiency of

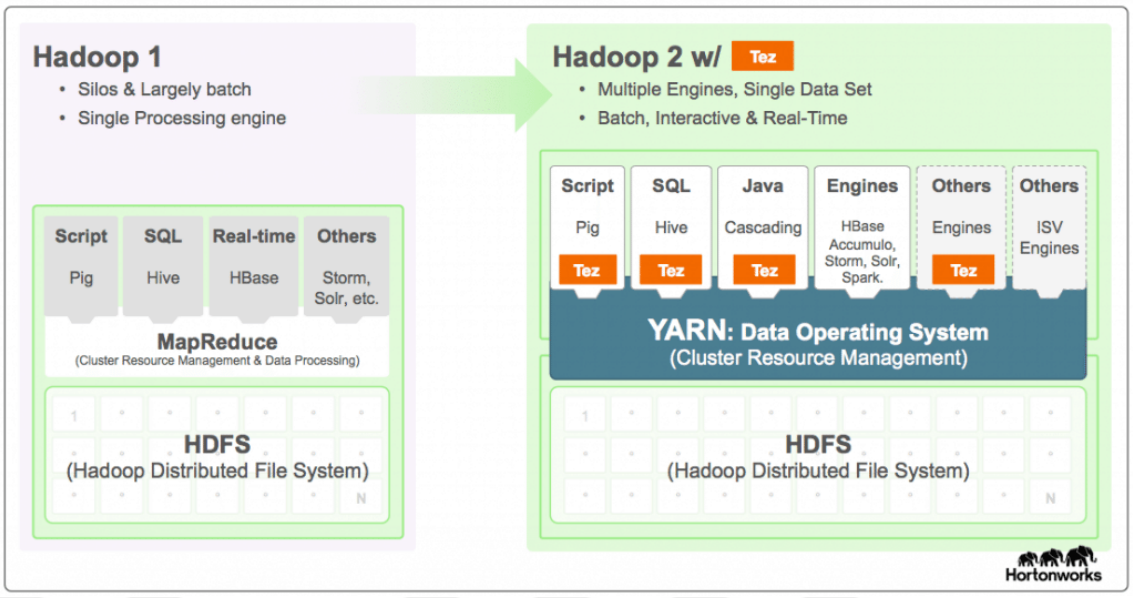


Figure 2.6. Apache Tez (Cloudera 2019)

an application through in-memory cluster computations. Spark provides an interface with implicit data parallelism and fault tolerance for programming clusters. Batch applications are designed to handle workloads including iterative algorithms, interactive queries, and flow. Figure 2.7 shows Apache Spark architecture.

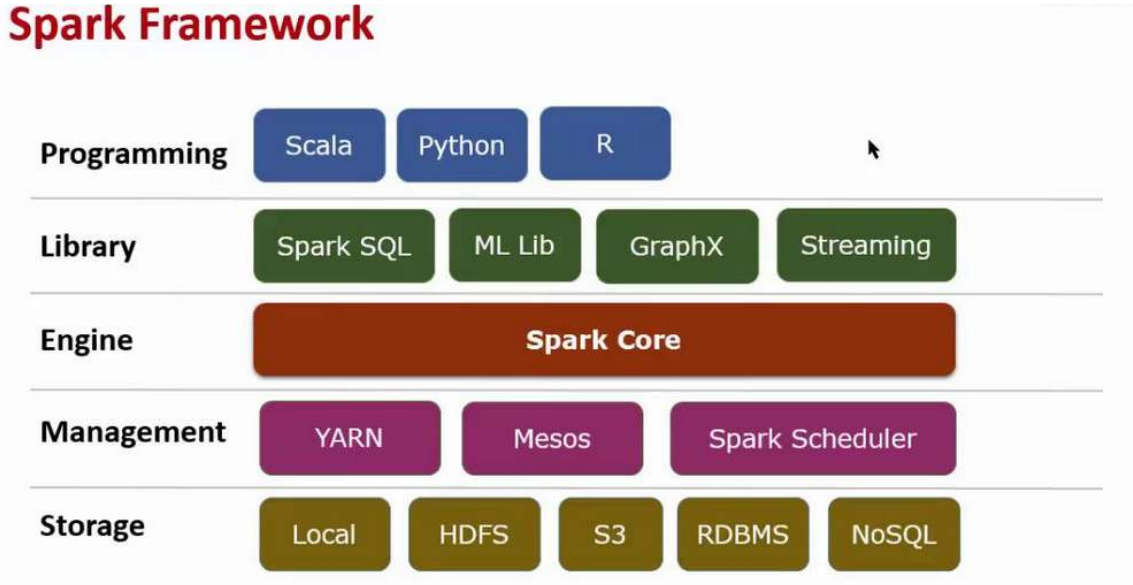


Figure 2.7. Apache Spark Architecture (V2 Maestros 2016)

- **Spark Core:** It is the basic execution infrastructure on which all other functions of the Spark platform are built. It is capable of in-memory processing to provide speed. A wide range of applications may be developed using Java, Scala, and Python APIs.
- **Spark SQL + Dataframes:** Spark SQL module is used in structured data processing. It has DataFrames to abstract data sets. With its distributed SQL query engine unaltered Hadoop Hive queries can run up to 100 times faster in existing environment. Spark SQL has also a strong integration with its ecosystem.
- **Streaming:** Many applications need not only batch data, but also the ability to process and analyze new data flowing in real-time. Working at the top of Spark, Spark Streaming combines Spark's fault tolerance and ease of use with powerful interactive and analytical applications on both streaming and historical data. HDFS integrates easily with a wide range of popular data sources including Kafka, Twitter and Apache Flume.
- **MLlib (Machine Learning):** Machine learning has came out as a important step in the processing of big data in business intelligence. MLLib, a scalable machine learning library, offers both qualified algorithms and speed (100 times over MapReduce). The library may be used as part of Spark applications in Java, Scala and Python.
- **GraphX - Graphical Calculation Engine:** It allows users to generate output in an interactive graph structure.

2.1.3. Coordinators

A coordination service is important for a distributed system because of system needs to ensure synchronization across its cluster. Coordinator behaves as a centralized repository where applications can exchange their data. Also it is responsible to keep the distributed system functionalities in its coordination, synchronization, serialization goals.

- **Zookeeper**: It is an Apache open source coordinator project. Zookeeper keeps configuration information and provides naming and synchronization over Hadoop clus-

ters. With its centralized structure, it aims to make management easier while ensuring reliable propagation of changes.

ZooKeeper has an infrastructure for cross-node synchronization by maintaining status type information in memory on ZooKeeper servers. A ZooKeeper server stores local log files about state of the entire system and related information. Multiple ZooKeeper servers support large Hadoop clusters, with a master server synchronization.

Within ZooKeeper, an application can create a znode, which is a file that persists in memory on the ZooKeeper servers. Any node in the cluster can update znode and register to be notified of changes to that znode. An application can synchronize its tasks across the distributed cluster by updating its status in a znode. Then znode sends information about specific node's status change. This cluster-wide status update mechanism is very important for serialization and management tasks across a large distributed set of servers (IBM 2019).
Figure 2.8 shows Zookeeper how works with clients.

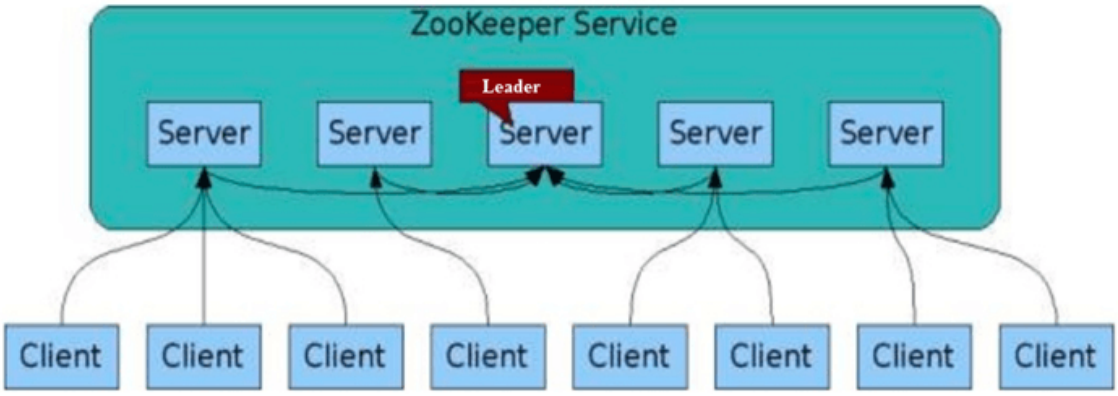


Figure 2.8. Zookeeper Arhitecture (Zookeeper 2019)

Some of the important reasons of using the Zookeeper are (Guru99 2019):

- Cooperation of server processes and mutual exclusion is being allowed
- A transaction process may end up with success or failure
- Data encoding according to specific rules
- Hierarchical namespace is maintained like a directory system

- Node joining or exiting updates status in real time
- More machines means more performance
- Leader election for more performed coordination

2.1.4. Monitoring

It is crucial to provide that system admins see the progress and status of each application running on the Hadoop cluster for monitoring. Web-based monitoring software is used where Hadoop components communicate with it using Rest API. Using agents these monitoring softwares collect data on nodes and other management services. Generally monitoring tools serve as a high-level component management module for big data platform like Ambari for Hortonworks/Cloudera.

- **Cloudera Manager**: Cloudera Manager is a coordinator application for CDH (Cloudera's Distribution Including Apache Hadoop) clusters. Cloudera Manager monitors deployment phases of components and pre-configures them to reduce administration costs. Cloudera Manager (CM) can operate the complete CDH stack and other services easily in addition to reporting, viewing, and logging.

  Cloudera Manager Server is the core component of the monitoring software as it integrates with other components including;

  - Agent - For process starting/stopping, settings, triggering deployments, and monitoring stuff done by agent that is installed to each host.
  - Management Service - Monitoring, reporting and alerting functions done by management service with roles.
  - Database - It stores configuration and monitoring records.
  - Cloudera Repository - Platform package repo.
  - Clients - Accessing points to interact with server.

Detailed architecture of Cloudera Manager is given in the following diagram Figure 2.9.

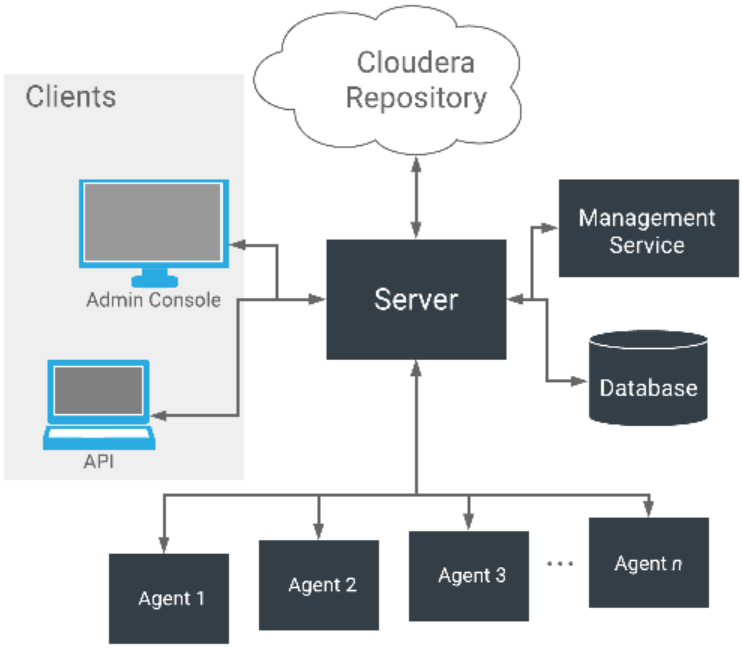


Figure 2.9. Cloudera Manager Architecture (Cloudera CM 2019)

- **Apachi Ambari**; is a software project that makes easy to manage and monitor Hadoop clusters. Ambari has REST API system that automate operations in the cluster. It shows the health of the Hadoop cluster using an interactive dashboard. It has secure and consistent interfaces for operatational control. Detailed architecture of Apache Ambari is given in Figure 2.10.

  Ambari core components are:

  - **Ambari Server**: Master server of Ambari. It has several entry points with different acceptable parameters. They are:
    - Software setup/upgrade
    - Daemon management
    - Ambari backup and restore
    - LDAP (Lightweight Direct Access Protocol)/PAM (Pluggable Authentication Module) /Kerberos management

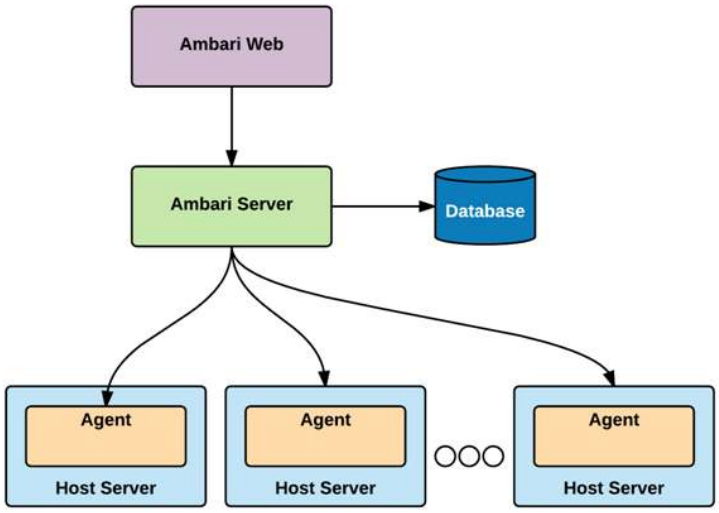


Figure 2.10. Apache Ambari Architecture (Intellipaat 2019)

- **Ambari Agent**; runs different jobs on cluster nodes and sends heartbeats.
- **Ambari Web UI** allows management of the cluster.
- **Database** Ambari supports several RDBMS for storing metainfo about components.

Features of Apache Ambari are (Intellipaat 2019):

- **Platform independent**: Apache Ambari runs on different operating systems and it can be installed via package managers.
- **Pluggable component/Extensibility**: It has modular infrastructure that can be customize for different components. Also extendable with different view components.
- **Failure recovery**: If something goes wrong during a run, the system should gracefully recover from it.
- **Security**: It syncs with LDAP over the cluster.
- **Version management**: Like a high-level package manager, Ambari manages big data components itself.

2.1.5. Data Collection

Big Data accumulates data from various internal and external sources. Data digestion from diverse sources is done either through batch or real-time processing. Below we list technologies to digest data from those sources.

- **Apache Kafka**; is an event streaming platform capable of collecting lots of data rapidly. Kafka, which started as a messaging queue, has an abstraction technique based on distributed commit log. When LinkedIn opened sourced Kafka in 2011, it has quickly emerged from messaging queue to a capable event streaming platform today (Confluent 2019).

  Apache has three key capabilities: publishing/subscribing of streams, storing in a fault-tolerant way, and processing streams immediately. Kafka is used also in Java ecosystem for performance metrics monitoring, website activity tracking, log aggregation, and real-time analytics.

  Kafka runs on single or multiple node that has scalable clustering features. A Kafka cluster, stores a stream of records in topics and each record has a key, a value, and a timestamp. Kafka manages four main APIs:

  - Producer: A stream of records can be publish to one or more topics.
  - Consumer: Subscribing topics can be multiple.
  - Streams: An application can behave as transformers between input and output streams.
  - Connector: Reusable producers or consumers can be used for connecting topics that already exist.

  Figure. 2.11 shows that Apache Kafka Cluster Architecture

- **Apache Flink**; is a distributed, low latency and high throughput stateful data processing/analytics engine for various data streams (Flink 2019). Flink has an impressive in-memory computation speed that may scale to thousands of cores and petabytes of data. Flink has a built-in stream and batch processing, state management, event-time processing semantics supports. It can be deployed on different resource

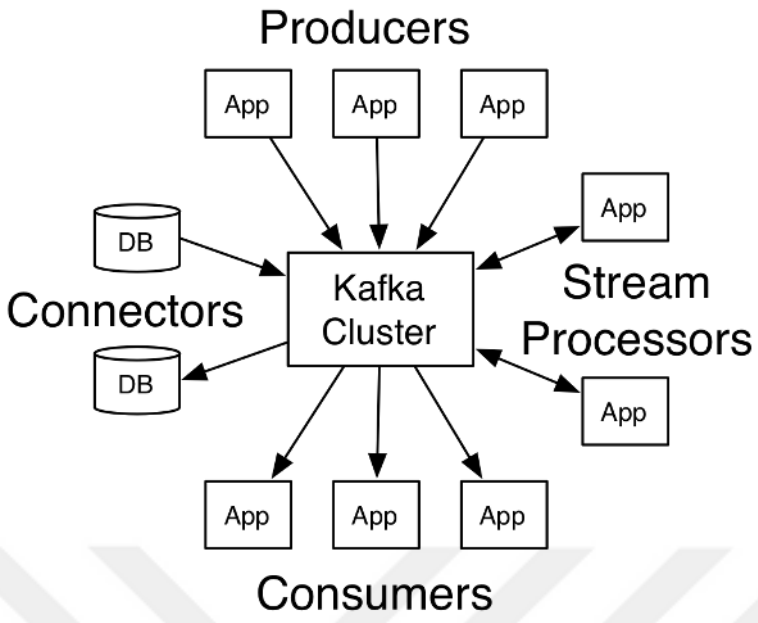


Figure 2.11. Apache Kafka Cluster Architecture (Confluent 2019)

managers such as YARN, Kubernetes and Mesos. Flink has not a single point of failure hence ensures high availability. Figure. 2.12 shows that Apache Flink Data Flow Architecture

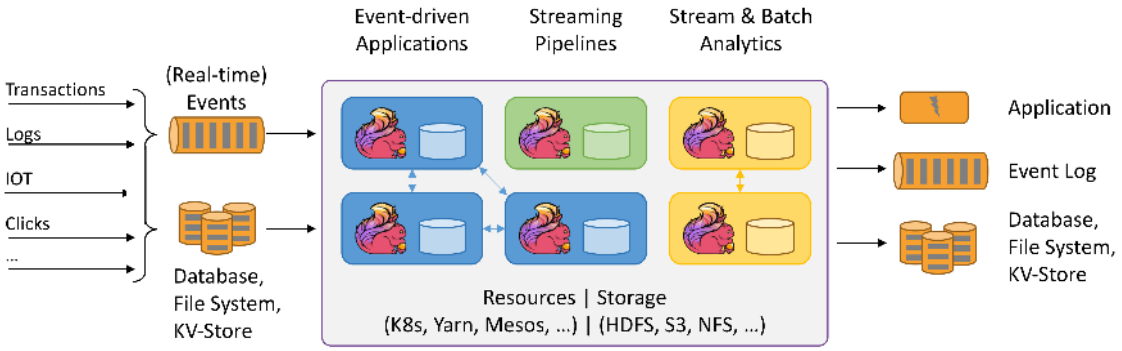


Figure 2.12. Apache Flink Data Flow Schema (Flink 2019)

Flink distributed stream processors not only restarts the flow in an error, but also tries to ensure that the current state is not lost through:

- Consistent Checkpoints: Recovery mechanism uses consistent checkpoints for recovering the states of an application when a failure occurs.
- Efficient Checkpoints: Check-pointing is quite expensive in an application that

maintains terabytes of state. Flink may perform asynchronous and incremental checkpoints which provides small latency.

- High-Availability Setup: The HA-mode works with ZooKeeper so Flink eliminates the single point of failure for its nodes.
- End-to-End Exactly-Once: Data is only written out exactly once in some storage systems while transactional sinks is used.
- Integration with Cluster Managers: Flink can be set up with resource managers like YARN, Kubernetes and Mesos.

- **Apache Flume**: Flume is a distributed service that collects large amounts of log data using streaming data flows. With configurable reliability and many fail-over and recovery mechanisms, it ensures consistency and high availability. Also its data model provides online analytic application.

  Figure 2.13 shows Apache Flume Data Flow Architecture.

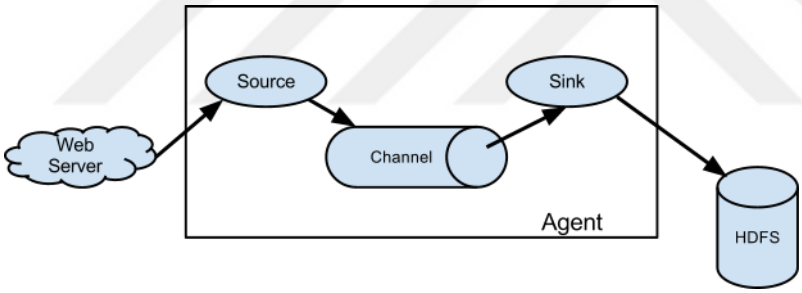


Figure 2.13. Apache Flume Data Flow Schema (Flume 2019)

Flume works as a JAVA application called agent and has 3 interconnected components:

- Source: Where the logs come to Flume,
- Channel: Location of logs in Flume,
- Sink (destination): Where to write the logs.

Basically, an agent set with 1 source, 1 channel and 1 sink, works to take logs from a single point and write them to a single location. It is possible to build complex models by running multiple channels/sinks or multiple agents. Flume is not limited

to the most common use as above. In order to receive events, it is possible to use different structures such as reading directly from source in various formats. Besides this; replication, multiplexing, fail-over or load-balancing scenarios may also be designed by setting multiple channels/sinks.

2.1.6. Data Integration

One of the most important steps in a big data project is combination of data from different sources and formats and then gives to users an integrated view of the accumulated data. Integration of big data provides decision makers valuable insight for their businesses. The process of integrating large data sets can be challenging and complex. Some of those challenges are; data uncertainty, synchronization between data sources, data ownership and privacy, storage and management of big data.

- **Sqoop**: In order to process the data stored in the relational database effectively on Hadoop, it must be transferred to HDFS. Sqoop is an open source, command-line interface tool designed to transfer data between relational databases and Hadoop. Already the name is derived from the words Sql-to-Hadoop.

  Sqoop supports all popular databases such as PostgreSQL, MySQL, Oracle, SQL Server. Since it is a Java implementation project like Hadoop, it can work with all databases accessible using JDBC. Sqoop may both import and export data from external sources. It is possible to import data from the database onto HDFS, either directly as a Hive table or to HBase. The advantage of Sqoop is that data transfer can be done in parallel with MapReduce tasks for speed. Sqoop also supports data transfer from/to MySQL and PostgreSQL without using JDBC (Ilter 2013).

2.1.7. Data Workflow

In order to implement the big data flow, we must first understand the data and processes in business. Use case and process diagrams in an organization helps understanding the data and the direction of data flow. The hook points of the streaming data are determined by these diagrams. However, workflows are task-oriented and may not be sufficient

by themselves for the identification of key attributes. Therefore, it should be studied in conjunction with the data model and ER diagrams for modules.

Implementation of data flow involves;

- Identification of the data sources.
- Ensuring access speeds and permits for the data.
- Determination of data types.
- Create/update data flow based on business goals.

Usually, these tasks are first performed manually. As things need to scale up, the processes are automated usually triggered through a cron job. Ultimately, you may reach a point where cron jobs may not guarantee a stable and robust performance. That's when you need a data-flow management tools.

- **Oozie**: It is a task scheduler for Hadoop ecosystem that runs the workflow of dependent jobs. Basic idea is using directed acyclic graphs for tasks, which may either run in parallel and sequentially in Hadoop. It has two parts:
  - Workflow engine: Storing and running of tasks
  - Coordinator engine: Running workflow jobs and coordinate them

  Oozie is scalable service that can manage the timely execution of thousands of workflows where each workflow has scores of tasks in a Hadoop cluster. Oozie workflow contains action and flow-control nodes. A workflow task, represented by an action node, may run a MapReduce job, move files into HDFS, import data using Sqoop, Hive or Pig jobs or run a shell script of a Java program.

  Figure 2.14 shows how Oozie works; Start-Node, initializes the workflow job. End-Node, signals end of the job. Error-Node assigns the occurrence of an error and is responsible for the error to be printed.

- **Airflow**: This is a platform to programmatically author, schedule and monitor workflows. Airflow started in October 2014 by Airbnb, joined the Apache Software Foundation's Incubator program in March 2016.

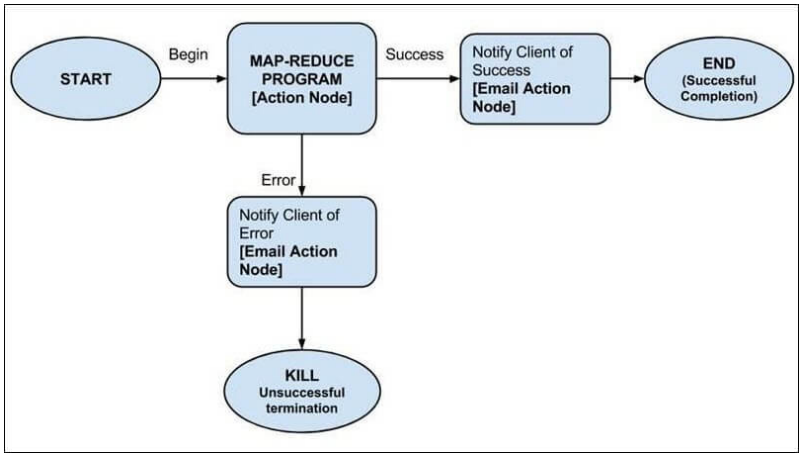


Figure 2.14. Apache Ooozie Architecture (Guru99 2019)

Airflow scheduler executes the DAGs of jobs on an array of workers while following the specified dependencies. Jobs implemented using Python scripts, allows them to be versionable, maintainable and open to collaboration.

Principles of Airflow (Airflow 2019):

- Dynamic: Pipelines are configured using Python code that allows instantiating them dynamically.
- Extensible: Defining custom operators, executors and extensions allows improved abstraction.
- Elegant: Pipelines are lean and explicit that can be parameterized via using Jinja templating engine.
- Scalable: It is modular and easily to scale.

- **Luigi**: Luigi is a Python application that allows building complex pipelines of batch jobs. It handles workflow management, dependency resolution, visualization, command line integration handling failures and much more. The aim of Luigi is the handling of long-running batch processes that typically takes time like importing/exporting of data to/from Hadoop, execution of machine learning algorithms, or anything that requires pipelining of sevaral tasks.

  Luigi manages much of the workflow, so that you can focus on the tasks and their dependencies (Luigi 2019). Luigi ensures all file system operations are atomic so

the pipeline will not crash in a state containing partial data. Figure 2.15 shows how Luigi may be used.

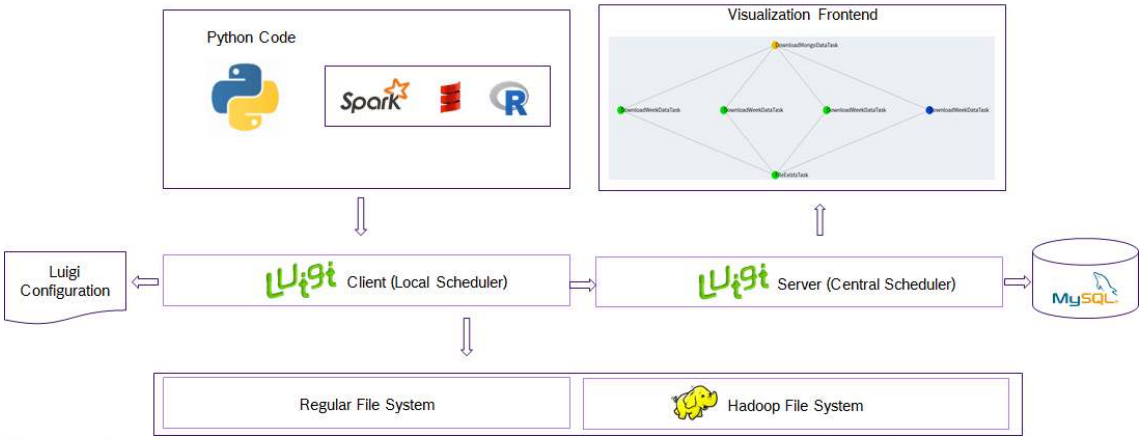


Figure 2.15. Luigi Workflow Example Usage (Lade 2017)

Luigi's one major advantage is that, it is not only for Hadoop therefore differs from Oozie. It may be extended with other kinds of tasks and their dependents. Advanced dependency graphs of tasks may include date algebra or recursive references to other versions of the same task using Luigi.

2.1.8. Data Flow/Ingestion

Data ingestion is the event of retrieval of data from data sources to data storage points. Data which is produced in real time and/or in batch should be ingested at a level that can handle this production frequency. Data collected from several sources and transmitted in different formats may require some transmissions during ingestion. Typically, periodic tasks are triggered by cronjob like task managers that requires configuration. This process can be managed by a program that can both resolve data flow and ingestion at the same time. For business intelligence and analytics applications, this data ingestion process is important step in providing decision support.

- **Apache NIFI**: It is an open source software for automating and managing the flow of data between systems. It comes with a Web User Interface for monitoring, creating and & controlling over data flows. It is powerful, immensely configurable and alterable data flow process that can modify data on the fly. It can be extensible and modular with adding custom components (NIFI 2019).

Apache Nifi features;

- Queuing data and buffering support
- Prioritization schemes is allowed
- Data sources connect via processors
- Only needs Java runtime to run
- Good at lack of connectivity
- Flow optimization
- Role-based authentication/authorization
- Simple function structure can be use to make more detailed flows
- Custom controller services and processors
- Communication over secure protocols, content encryption
- Fast development and testing
- Management of dependencies is easy with class loader isolation

Figure 2.16 shows Nifi architecture

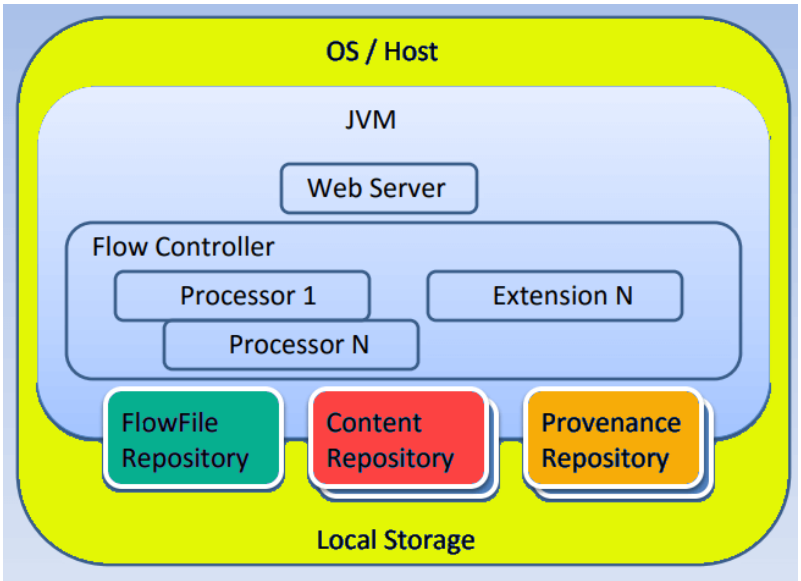


Figure 2.16. Apache Nifi Architecture (Guru99 2019)

- **StreamSets Data Collector**: StreamSets Data Collector is an open source, enterprise level and continuous big data ingestion infrastructure. It has user interface that lets developers, data scientist and data infrastructure teams to create data pipelines easily. Also StreamSets Data Collector reads from and writes to a large number of end-points to/from various data sources. You can use Javascript, Java Expression Language and Python to transform and process the data on the fly. It can setup data pipelines in cluster mode for fault tolerance. It scales out so it gives fine grained monitoring at each stage of the pipeline (StreamSets 2019).

  The existing StreamSets DataOps Platform (including StreamSets Data Collector, Control Hub, and Transformer among other products) is designed for enterprises that have a wide variety of data integration use cases and design patterns. Figure 2.17 shows StreamSet's big data architecture.

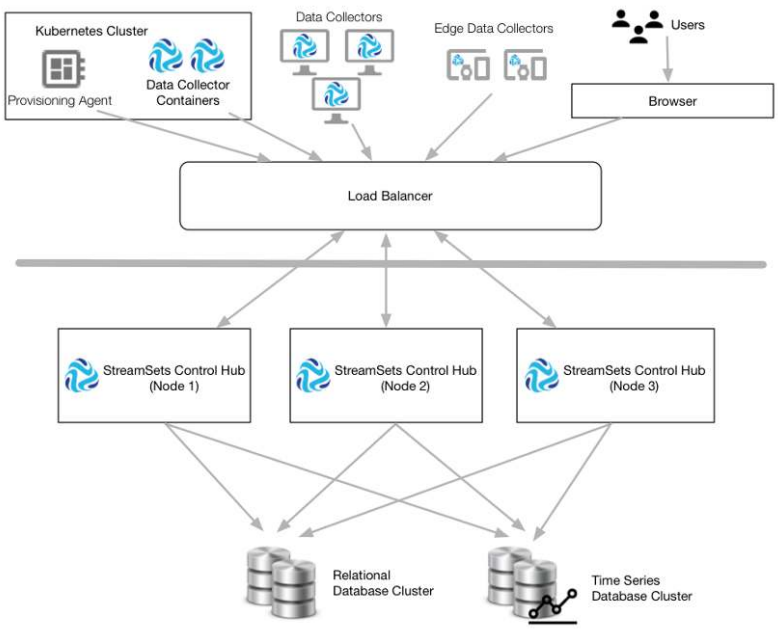


Figure 2.17. StreamSets Architecture (StreamSets 2019)

2.1.9. NoSQL Databases

NoSQL (Not Just the Standard Query Language) is a database design based on a wide range of data models, such as key-value, document, columnar and graphical formats, compared to traditional database approaches. The term NoSQL may have begun to

be implemented in the first RDMBS databases, but it has received its true terminological value from cloud studies beginning in 2000. Performance and scalability requirements in current implementations countered the need for rapid and rigid data consistency that RDBMS provides for enterprise enterprise applications. For working with distributed datasets, NoSQL databases are especially useful because of their desgn according to big data. NoSQL helps to deal and solve important requirements of big data attributes as follows.

- Volume: ACID (Atomicity, Consistency, Isolation, Durability) properties come cheaper with doing partition our data multiple places.
- Variety: Changing a schema is easier in NoSQL models.
- Velocity: Storing data in memory is more faster and cheaper today

Some categories of NoSQL database are:

- **Key-value stores** This structure has an array or data structure for each key. Our goal is to access data quickly through the keys. Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives. Some examples of this category Redis, Berkeley DB, Aerospike, MemchacheDB and Riak.
- **Document databases** In the document store type, the data is saved as document. For example, a JSON data can be accepted as a document. For content management and mobile applications use document databases. MongoDB, CouchDB, Couchbase Server, MarkLogic and DocumentDB are examples of document databases.
- **Wide-column stores** The structure here is similar to the classical database. There are tables, columns and rows. The biggest difference is that when we define the structure, we determine the fields that will be super columns and under these fields a different column can be found for each record. Cassandra, HBase, Google BigTable are examples of wide-column stores.
- **Graph stores** The general purpose of this structure is to hide the relationships between the data. Each data can be a node, and our main goal here is to define the

relationship between nodes together with their aspects. Some graph databases are AllegroGraph, IBM Graph, Titan and Neo4j.

Figure. 2.18 shows big family of NoSQLs.

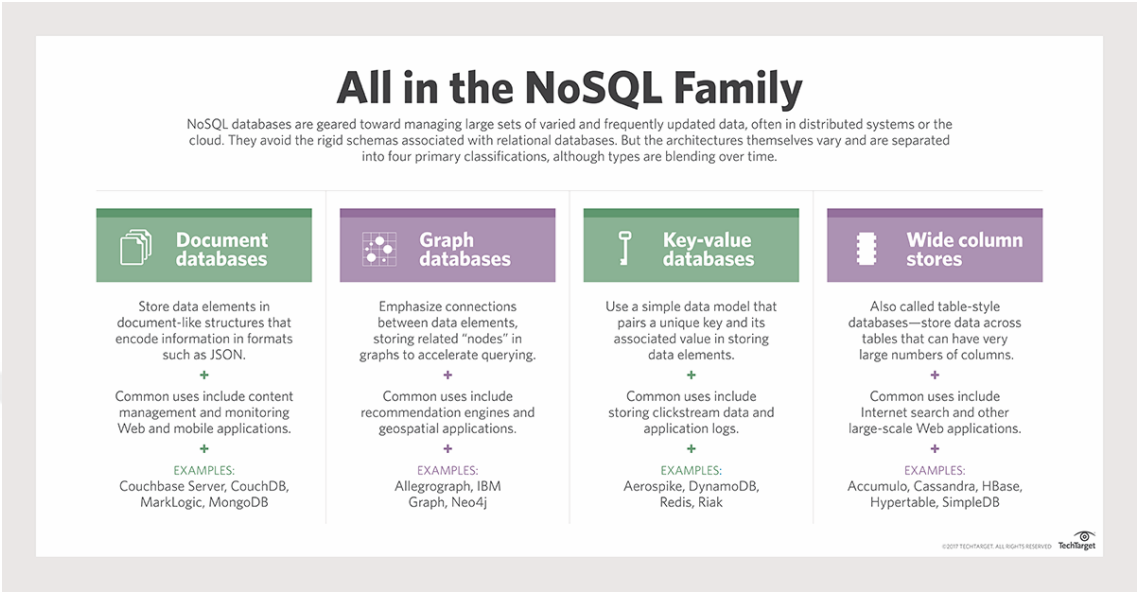


Figure 2.18. All in the NoSQL Family (TechTarget 2019)

2.1.10. Data Warehouse

A data warehouse is a integrated, subject-oriented with time, and non-volatile collection of data in support of getting valuable report. A Data Warehouse works as a central data storage where data arrives from various data sources. Data flows into a data warehouse from the transactional or relational way. And data may be in structured, unstructured or semi-structured.

The data in Data Warehouse is transformed, processed and ingested so that data can access through Business Intelligence tools, SQL clients, and spreadsheets. A data warehouse merges information coming from different sources into one consistent database. This helps to coming out of availability of data. Data warehousing also makes data mining that looks for patterns in the data that can brings higher benefits.

There are four components of data warehouses are:

- Load manager: It is responsible for preparing the data for entering into the Data warehouse.

- Warehouse Manager: It performs operations about indexes, views, aggregations, archiving etc. stuff.
- Query Manager: It operates of scheduling and execution of queries.
- End-user access tools: These tools about accessing to Warehouse via different protocols.
- **Apache Hive**: Apache Hive is a data warehouse and an ETL (extract-transform-load) tool which provides an SQL-like interface between the user and the Hadoop distributed file system (HDFS) which integrates Hadoop. It is built on top of Hadoop. It is a software project that provides data query and analysis. It facilitates reading, writing and handling wide datasets that stored in distributed storage and queried by Structure Query Language (SQL) syntax. It is not built for OLTP (Online Transactional Processing) workloads. It is frequently used for data warehousing tasks like data encapsulation, Ad-hoc queries, and analysis of huge datasets. It is designed to enhance scalability, extensibility, performance, fault-tolerance and loose-coupling with its input formats.

  Initially Hive is developed by Facebook and Amazon, Netflix and It delivers standard SQL functionality for analytics. Traditional SQL queries are written in the MapReduce Java API to execute SQL Application and SQL queries over distributed data. Hive provides portability as most data warehousing applications functions with SQL-based query languages like NoSQL.

  The main components of Hive and its interaction with the Hadoop is demonstrated in the Figure. 2.19 below:

  - User Interface – It provides an interface between user and Hive. Submitting queries and other operations done by UI.
  - Driver – Hive driver, handles sessions according to connected protocol.
  - Compiler – It processes queries and does execution plan.
  - Metastore – Hive stores tables meta information to HDFS as metadata that can be different schema. Metadata means data about data.

– Execution Engine – Compiler components does execution plan then execution engine performs the plan. Runs queries and brings result.

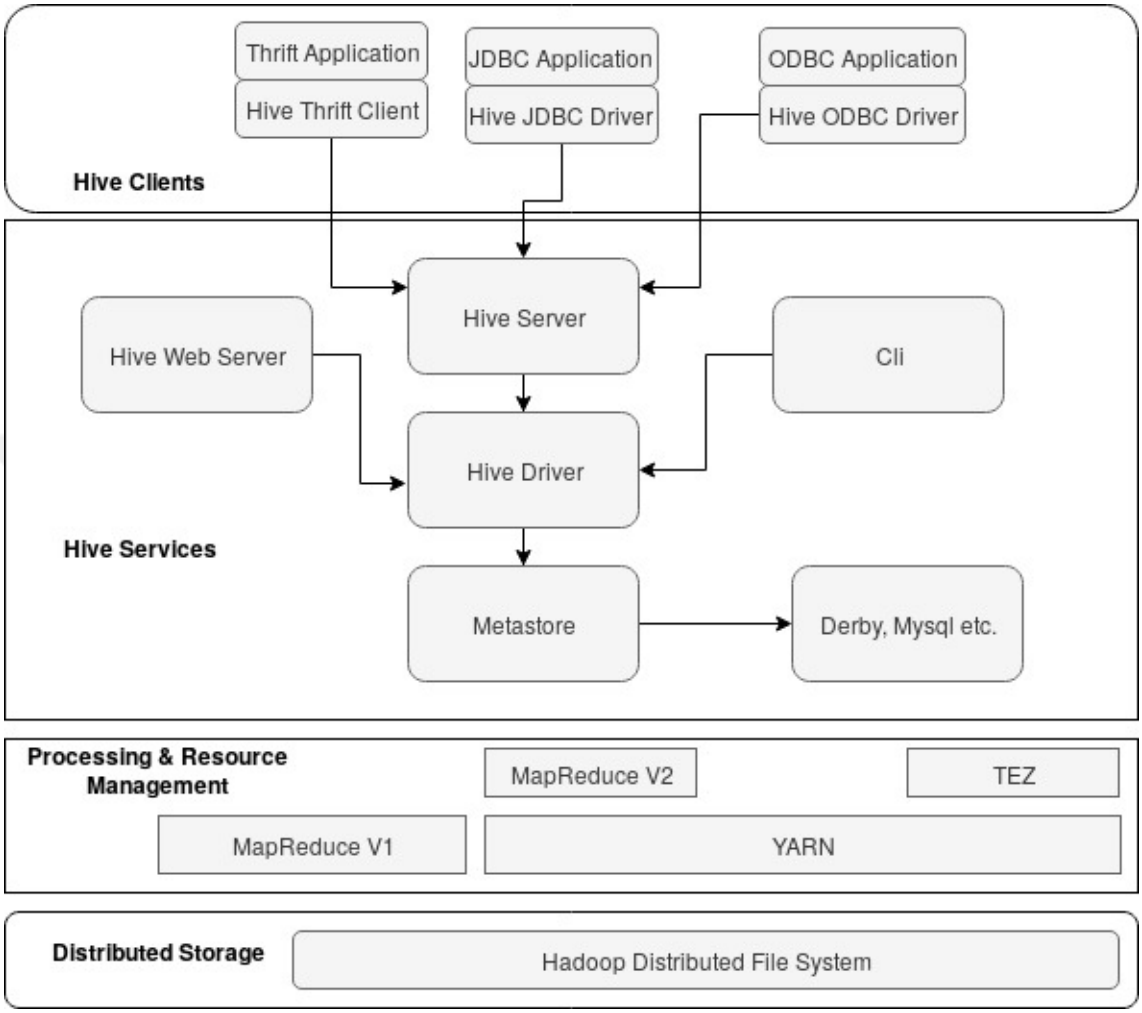


Figure 2.19. Architecture of Apache Hive

- **Apache Impala**: Impala is SQL query engine based on MPP (Massive Parallel Processing) technique for processing large volumes of data that is stored in Hadoop cluster. It is is written in C++ and Java (Impala 2019). Impala's operating approach can be defined as Distributed SQL Query Engine. Unlike Hive, Impala queries are not translated into MapReduce jobs. This eliminates time-consuming functions such as starting, coordinating, and restarting MapReduce jobs. It accesses data directly through distributed queries. The query engine analyzes the given query and calculates how to run it distributed, then runs the required queries on each node. After the queries, the results from each node are combined directly on the memory,

not on HDFS. Since the de-fragmentation is performed on memory, it works faster but requires sufficient memory in the system. During this process, a more functional and performance store format called Parquet is used.

Figure. 2.20 shows Apache Impala architecture,

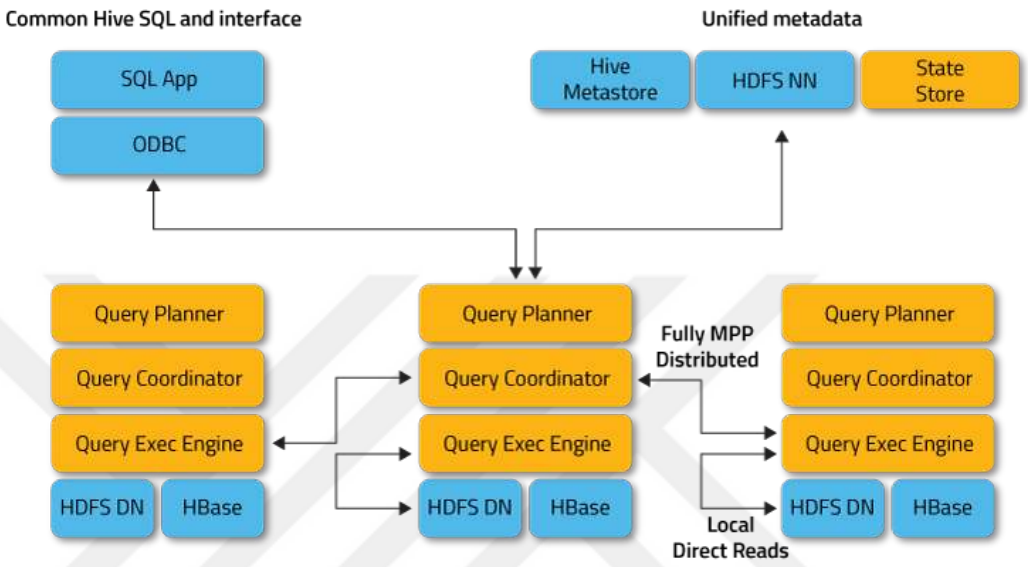


Figure 2.20. Architecture of Apache Impala (Impala 2019)

Impala combines performance of a traditional database and with the SQL support the flexibility and scalability of Hadoop, by utilizing standard components such as HDFS, HBase, Metastore, YARN and Sentry. With Impala, users can communicate to other databases with SQL queries in a faster way compared to other SQL engines like Hive. Impala supports at reading format Avro, RCFile and Parquet used by Hadoop. It reduces the latency of utilizing MapReduce and this makes Impala faster than Apache Hive but not for long-run queries.

Some significant advantages of Impala are:

- Processing data that is stored in HDFS at lightning-fast speed with traditional SQL.
- Data transformation and movement is not required for data stored on Hadoop
- Easy to access for various data stores.

  - Impala uses Parquet file format that is optimized file format for large-scale queries typical in data warehouse.

- **Presto**: It was originally designed at Facebook as they had a need to run interactive queries against large data warehouses in Hadoop. Specifically it fills this requirements about running fast queries against data warehouses storing petabytes of data. Presto works incredibly successfully from gigabytes to petabytes. One important feature of Preston does cross-query that means same queries can run diff rent kind of databases such as Mysql, Teradata, Cassandra, Oracle, and MongoDB. Also it processes these different sources with a single Presto Query. It is also a very powerful product on the data exploration side.

  Presto has some useful features:

  - Modular architecture.
  - Pluggable connectors
  - User-defined functions
  - Vectorized columnar processing.
  - Pipelined executions

  Presto's distributed system runs on Hadoop and uses a classic massively parallel processing (MPP) database management system. Its one coordinator master node works for synchronous with other workers. After submitting SQL queries via client to master daemon (coordinator), query engine parses queries, plan and schedule a distributed query plan across all its worker nodes. Presto has built-in SQL query syntax. Advanced queries, joins and aggregations support by Presto. The following diagram Figure. 2.21 illustrates the architecture of Presto.

  In additional, Presto uses memory usually in the context of the JVMs (Java Virtual Machine) itself, depending on query sizes and complexity of tasks you can allocate more or less memory to the JVMs. However, it does not use this memory to cache any data. Presto stores in-between data during the period of tasks in its buffer cache. However, it is not meant to serve as a caching solution or a persistent storage layer.

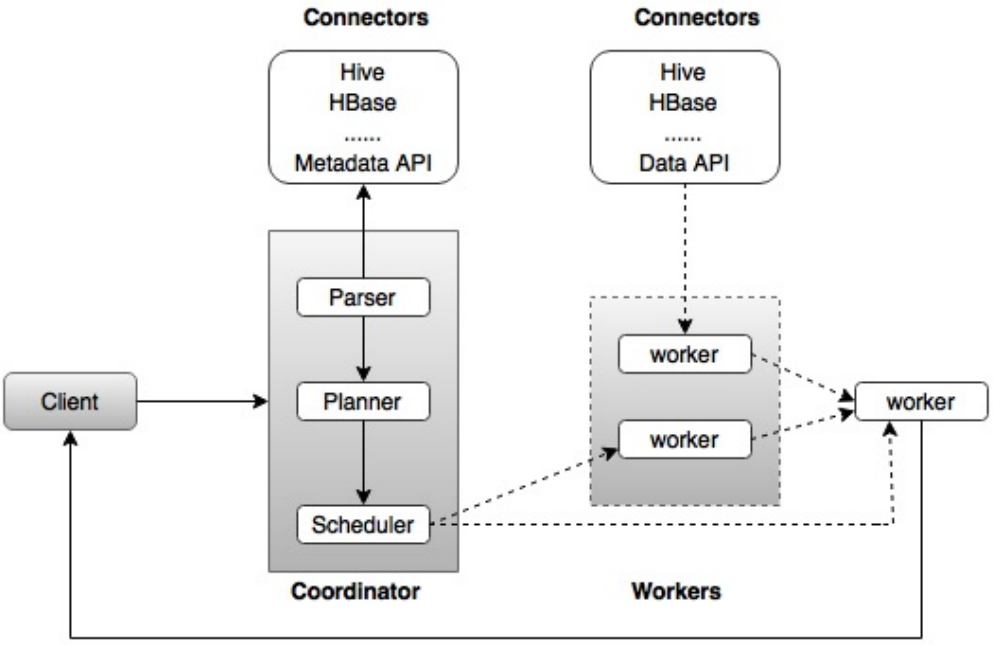


Figure 2.21. Architecture of Presto (PrestoDB 2019)

First focus of Presto's design is to be a execution engine that allows for querying against other separated data sources.

2.1.11. Real Time Analytics

Real-time data streaming and analytics is the process that is used for analyzing the huge amount of data at the moment it is used or produced. In this, we extract valuable information for the organization as soon as it's created or stored. In other words, we can say that real-time data streaming and analytics is a process that mainly focuses on the data produced or consumed or stored within a live environment. Let's take an example of analyzing the huge amount of data as it is produced within banks and branches, stock exchanges throughout the globe. The scope of analytics can be from multiple sources. We can import or fetch the data and store it within a system and can execute data analysis algorithms over it. And further, these analytics data is delivered to the users/administrator through an analytics dashboard. Real-time analytics can be used for below-listed purposes:

- To report historical data and current data concurrently.

- For receiving alerts on the basis of certain and predefined parameters.
- To build operational decisions and apply them on business processes or on other production activities based on real-time and on an ongoing basis.
- To apply pre-existing prescriptive models or predictive models.
- For outlook of real-time displays or dashboards in real time on constantly changing datasets.

Real time streaming also supported by distributed file systems such as Hadoop, S3 and other distributed file systems are supporting data processing in large volumes and on the other hand we can also query them using their different frameworks like Hive, Spark, Impala. On the other hand, organizations are looking for such type of platforms where they can look into business insights in real-time and act upon them in real-time. Alerting systems and platforms are also based on these real-time streams. But the effectiveness of these platform based on the data that we are processing in real-time.

- **Apache Spark**: Apache Spark is an unified analytic engine for large scale data processing. Basically, Apache Spark is a computing technology that is specially designed for faster computation. Spark is designed in order to cover batch applications, interactive queries, algorithms, and streaming. The main feature of spark is that it is in-memory cluster computing which means that this will increase the processing speed of an application.

  As real time processing, Spark applies structured streaming model that is new streaming model of the framework build on SQL engine. It came with Apache Spark 2.0 version, it ensures scallable, fault-tolerant, low latency and fast processing. Writing batch queries with streaming data is allowed with this implementation. Structured streaming provides new API for datasets/dataframes in Scala, Java, Python or R to express streaming aggregation, event-time windows, and stream-to-bath join.

  Figure. 2.22 shows that how Spark handles real time streaming.

- **Apache Flink**: Apache Flink is an open source stream processing framework for distributed, high performance and data accurate data streaming applications. Apache Flink also supports batch processing as a special case of stream processing.

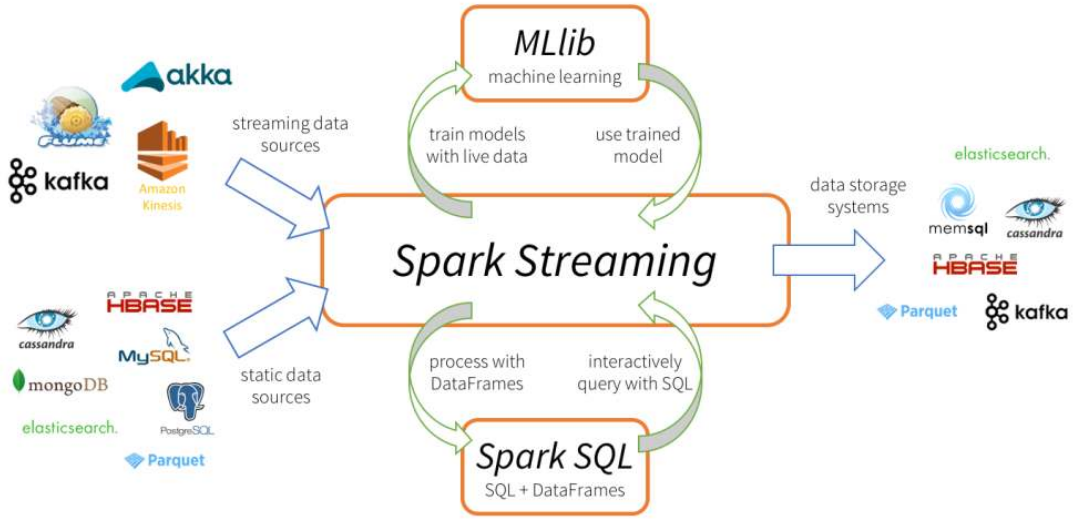


Figure 2.22. Real Time Streaming on Apache Spark (Spark 2019)

Analytical jobs extract information and insight from raw data. Traditionally, analytics are considered as batch queries or applications on limited data sets of recorded events. In order to add the latest data into the result of the analysis, it needs to be incorporated to the analyzed data set and the query or application is run again. The results are written to storage or produced as reports.

With a powerful stream processing engine, analytics can also be performed in a real-time way. Instead of reading finite data sets, streaming materials ingest real-time event streams and continuously produce and update results as events are consumed. The advantages of continuous streaming analytics compared to batch analytics are not bounded to a much lower latency from events to insight due to elimination of periodic import and query execution. Streaming queries do not have to deal with artificial boundaries in the input data which are caused by periodic imports and the limited nature of the input (Flink 2019).

Apache Flink good at streaming as shown in the Figure. 2.23

Flink provides good at providing continuous streaming as well as batch analytics. It has ANSI-SQL interface for batch and streaming queries. Computing of SQL queries has same results on real time streaming sets as well static datasets.

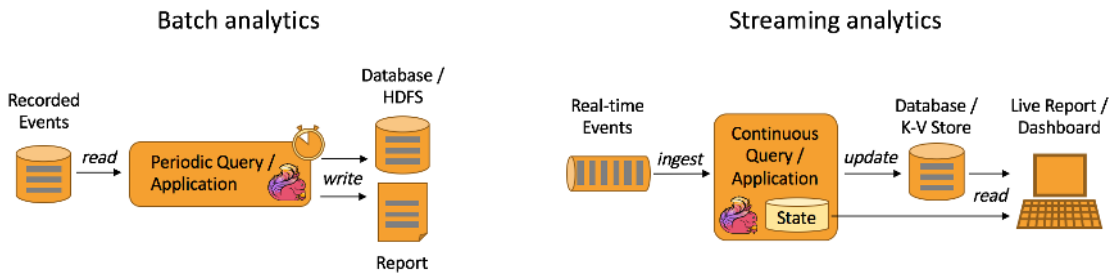


Figure 2.23. Streaming on Apache Flink (Flink 2019)

Rich support for user-defined functions ensures that custom code can be executed in SQL queries. If more logic needs, Flink's data APIs come with low-level control. Also Flink's Gelly library has algorithms and building blocks for large-scale and high-performance graph analytics on batch data sets.

- **Apache Druid**: It is a column-oriented distributed datastore. Druid stores its data in a columnar format. The data rules could have 100s or thousands of values. They could also have 100s or thousands of columns. In OLAP (Online Analytical Processing) queries what you query is a subset of those dimensions 5 or 20, etc. Being a column oriented store, it enables Druid to scan the columns necessary to answer the query. Druid supports sub-second query times because it is going to power an interactive dashboard. It utilizes various techniques, such as bit map indexes, dictionary encoding, data compression, query caching in order to provide sub second query times. Druid supports realtime streaming ingestion from almost any ETL pipeline. You can have pull based as well as push based ingestion. Druid supports automatic data summarization where it can pre-aggregate your data at the time of ingestion. Druid supports approximate histograms where we need to do fast approximate distinct counts. Druid is scalable to petabytes of data and highly available. Druid's core design combines ideas from OLAP databases, timeseries databases, and search systems to create a merged system for a broad range of use cases Figure. 2.24.

  Druid is complementary component for other big data streaming tools because of it places between a processing layer or storage and the end user, and serves analytic workloads as query layer. For this purpose, Druid supports multiple ingestion types.

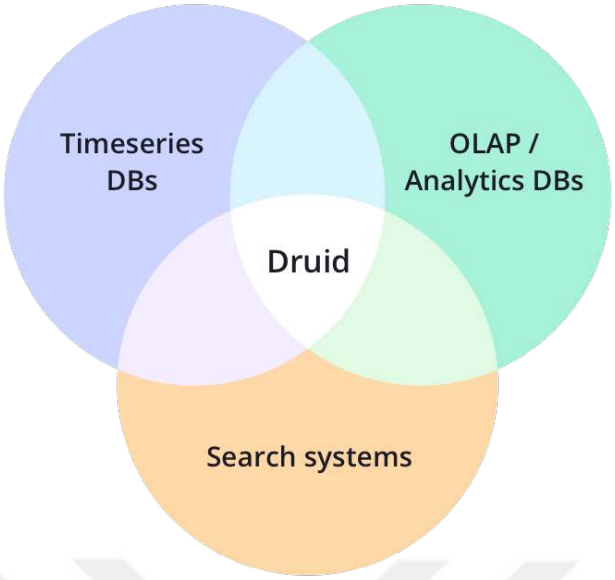


Figure 2.24. Core Design Ideas of Apache Druid (Druid 2019)

It connects to a data source, acts a message bus for streaming data loads or like a distributed filesystem for batch data loads. Druid converts raw data stored in a source to a more read-optimized segments that uses on indexing.

Druid stores data in columns with different column type and builds inverted indexes for string columns for performed searching and filtering. Like timeseries databases, Druid makes partitions of data by time to enable fast time-oriented querying. Druid can optionally pre-aggregate data as it is ingested that known as rollup, and can use for storage savings (Druid 2019).

Druid is based on microservice-based architecture, core services(querying, ingestion, coordination) of Druid can deploy on cluster nodes separately. Druid services can independently fail without impacting the operations of other services. In additionally, Druid also supports querying data with JSON via HTTP and SQL. Figure. 2.25 shows Druid's microservice architecture.

- **Apache Kudu**: Apache Kudu is a columnar storage manager developed for the Apache Hadoop platform. Kudu has similar Hadoop that runs on commodity hardware, can scale horizontally and supports highly available operation. Think of Kudu as the storage engine of a database. Alongside a query engine like Apache Impala or

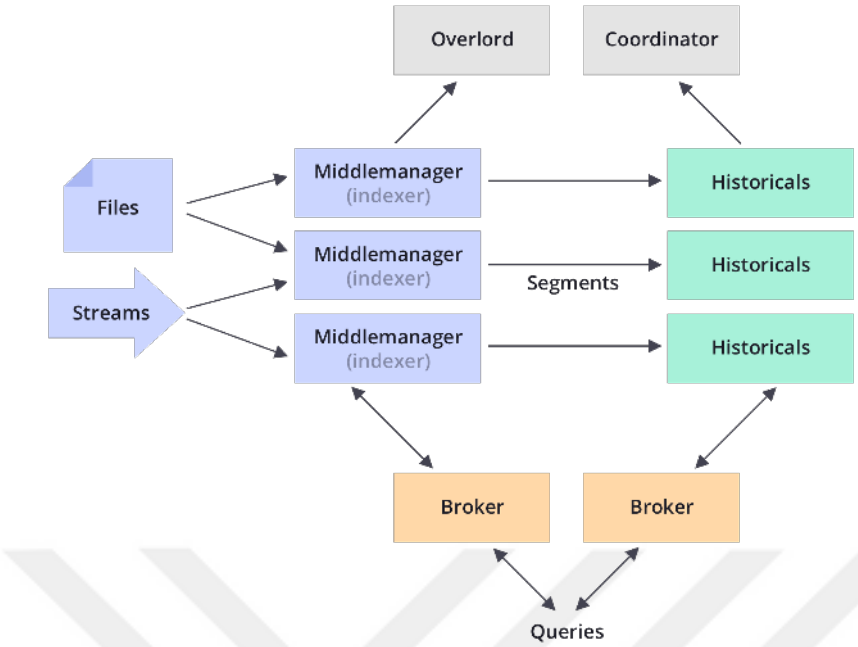


Figure 2.25. Druid Microservice Architecture (Druid 2019)

Apache Spark, it can be considered a complete solution for OLAP (Online Analytical Processing) workloads. Figure. 2.26 shows Kudu's position for analytics.

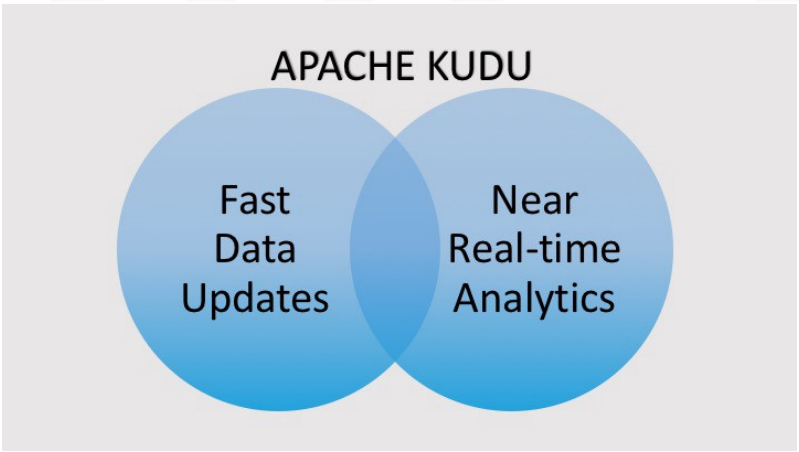


Figure 2.26. Apache Kudu (Kudu 2019)

A Kudu cluster's table storing like relational databases. A table can be as simple as an binary key and value, or as complex as a few hundred different strongly-typed attributes. Every table has a primary key in Kudu. This might be a single column like a unique value or a compound key tuple for a timeseries database. Rows can be performantly read, updated, or deleted by their primary key (Kudu 2019).

Kudu is a live storage system which supports low-latency millisecond-scale access to individual rows. For accessing to Kudu, you can choose between Java, C++, or Python APIs. And these APIs can be used for machine learning or analytics with batch access.

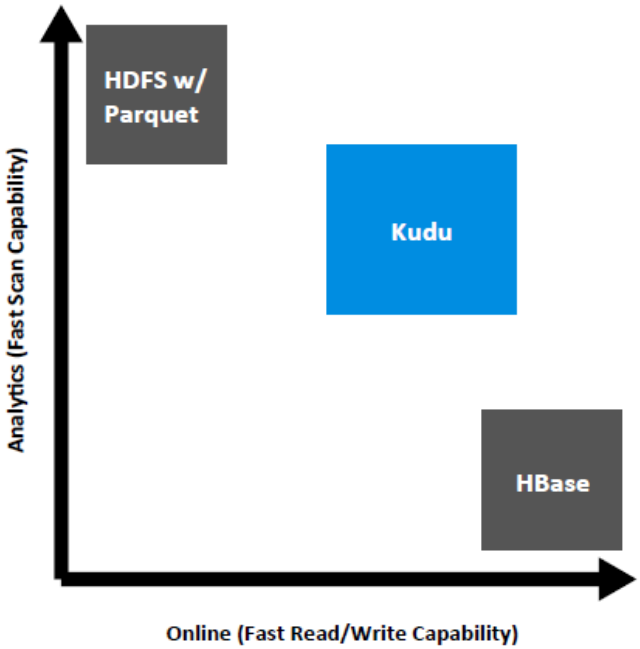


Figure 2.27. Apache Kudu Position (Kudu 2019)

Kudu was designed to fit in with the Hadoop ecosystem, and integrating it with other data processing frameworks is simple. It sits between HBase and HDFS with Parquet, attempting to remove the trade offs between quick scans and rapid random access. Figure. 2.27 shows position of Kudu between in Hbase and HDFS. Also stream data can be used from live real-time data sources with Java client, and then process it on the fly using Spark, Impala, or MapReduce. Kudu tables can be joined with data stored in other Hadoop storage such as HDFS or HBase. It can easily share data disks with HDFS DataNodes, and can manage in a RAM footprint as small as one gigabyte for light amount of work.

Kudu can be used to run analytical queries on Big Data. As Kudu is not yet ACID compliant — it lacks multi-row transactions, it cannot be used for all use-cases.

Kudu can be considered as an option for most of the existing systems where the storage layer is HDFS/HBase. HDFS does not support data updates, therefore Kudu is a good consideration for performing analytics on streaming data. Use cases like log analytics, network monitoring, payment fraud detection can all use Kudu. Some of the other use-cases that it fits well are around general data analytics, prediction and data integration.

2.1.12. Machine Learning

ML (Machine Learning) is based on algorithms that can learn from data without relying on rules-based programming. Big data is the type of data that may be supplied into the analytical system so that a ML model could improve the accuracy of its predictions that called as learn.

Machine learning doesn't involve any prior assumptions. Once they're provided with the required data, machine learning algorithms can process that data and identify patterns. Then those patterns can be used on new datasets. Generally, this technology is applied to high-dimensional datasets. It means the more data you can provide, the more accurate your predictions will be. And this is exactly where the power of big data comes in.

As the industry and sciences are experiencing a phenomenal rise in data generation, this scenario has presented a great opportunity for machine learning and big data to come together and create machine learning techniques which have the ability to manage modern data types by attaining computational and statistical intelligence for navigation of massive amounts of information with no or minimal human intervention.

Machines learn from extensive calculations performed over datasets, meaning the more the data, the more effective the learning. With the emergence of big data together with the advancements in computing technologies, machine learning has already evolved from that of the past. With the steadily increasing proliferation of big data analysis into machine learning, machines and devices will get smarter and should be able to perform in a more advanced manner. This will eventually lead to improvement and advancement in machine learning solutions.

- **Mahout**: Apache Mahout is an open source project, powerful, scalable machine-learning library that runs on top of Hadoop MapReduce and its background to ma-

nage huge volumes of data. Apache Mahout started as a sub-project of Apache's Lucene in 2008. In 2010, Mahout became a foremost project of Apache.

Figure. 2.28 shows three tiers of Apache Mahout Architecture.

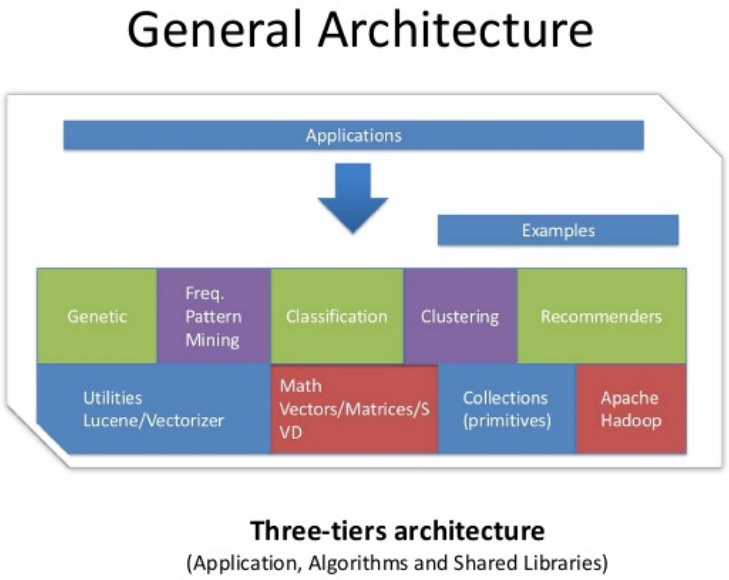


Figure 2.28. Apache Mahout General Architecture (Mahout 2019)

Some of machine learning algorithms implemented by Mahout are:

- Classification
- Recommendation
- Collaborative Filtering
- Clustering
- Topic Models
- Dimensionality Reduction

- **Apache Spark**: MLlib is Apache Spark's machine learning library. MLlib places Spark's APIs and interoperates with R and NumPy in Python libraries. Spark good at iterative computation, enabling MLlib to run fast. Also MLlib includes superiority algorithms that leverage iteration, and can give better results than the one-pass

estimations sometimes used on MapReduce. Figure. 2.22 shows how MLlib working on a Spark streaming data (Spark 2019).

Spark MLlib has tools such as:

- ML Algorithms: general ML algorithms (classification, clustering regression etc.)
- Pipelines: creating ML pipelines
- Persistence: store/load algorithms, pipelines and models
- Featurization: selection, extraction and transformation
- Utilities: data handling, statistics, linear algebra etc.

- **Apache Flink**: This is the ML library for Flink. FlinkML aims to provide scalable ML algorithms, an powerful API, and tools that help coding improvements in end-to-end ML systems. With Flink's scalable ETL capabilities that can be easily combined.

  Robust pipelines without the need to use yet another technology is easily developed for data ingestion and data munging. Developer can write ML pipelines quickly, using familiar programming concepts and terminology. Contrary to other data-processing systems, Flink implements in-memory data streaming, and natively executes iterative processing algorithms which usually use in ML. Flink exploits the streaming nature of Flink, and provide functionality designed specifically for data streams.(Flink 2019). Supported algorithms in Flink such as supervised learning, recommendation, unsupervised learning, outlier selection.

2.1.13. Security

Big data security refers to the means of prevention used to protect against any malicious activity. This understanding of security mainly focuses on data and its analytical operations. Like other types of cyber security, it can be sourced from online or offline attacks. In addition, big data requires a versatile security approach because the data is large and complex. A comprehensive security approach to the proper elimination of security risks includes.

- Data classification
- Clarifying of data access and interactions
- Data event correlation
- Application level control
- Device control and encryption
- Cloud storage control
- Secure network
- User control authentication/authorization
- **Apache Ranger**: This framework was developed to create and monitor security protocols for the hadoop ecosystem in general. With the development of YARN, Hadoop is now interactive with more applications. There should be a security procedure that can be used for multiple data access. In addition, the developed procedures should be followed in a centralized manner so that the Ranger provides us with this. It can use different authorization systems to achieve this (Ranger 2019).

  Some targets of Apache Ranger:

  - Web UI and Rest API for managing security tasks
  - Development of component-based safety modules
  - Creating a standard security procedure
  - Specific access mechanism such as role-based access control, feature-based access control

  Ranger provides access control across data access components with consistent policies. Security administrators can define these policies at the database, table, and file level. These policies can also be supported by dynamic conditional rules. In additiona, Ranger can integrate authorization models modularly, such as LDAP, Kerberos, Unix.

Components of Apache Ranger shows at Figure. 2.29:

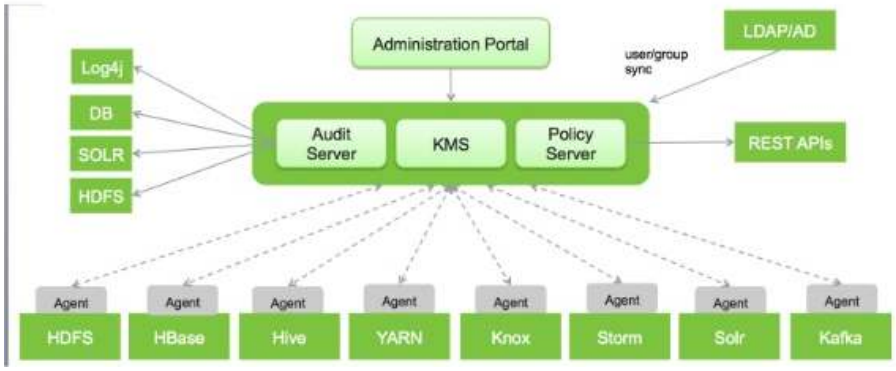


Figure 2.29. Apache Ranger Components (Ranger 2019)

- **Apache Sentry**: Apache Sentry is a role-based authorization component for Hadoop ecosystem. Sentry comes with the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster. Sentry currently works out of the box with HDFS, Apache Solr, Apache Hive and Impala

  Sentry runs with plugins for authorization engine for Hadoop components. So it can be write rules to validate an access requests for Hadoop resources. Sentry's modular structure can work with different kind of data models in Hadoop.

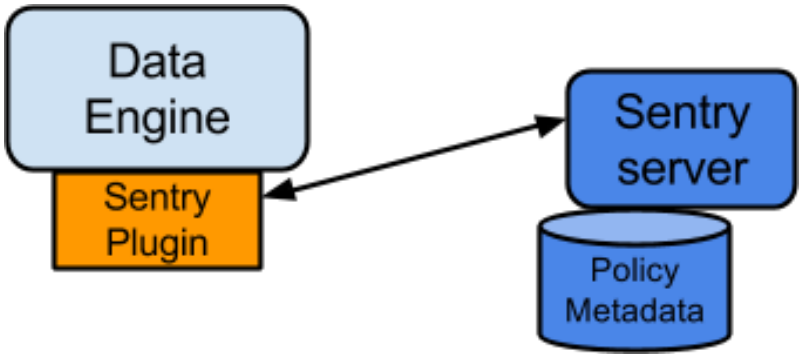


Figure 2.30. Apache Sentry Components (Sentry 2019)

Three components of authorization process in Sentry shows at Figure. 2.30:

  – Sentry Server: It manages of the retrieving and manipulating authorization metadata.

- Data Engine: This is a data processing application that uses for authorizing access to data or metadata resources.
- Sentry Plugin: It is responsible to manage metadata of authorization that stored in Sentry Server and runs within data engine. Also it contains policy engine for authorization that evaluates requests (Sentry 2019).

2.2. Related Works

Although Apache Hadoop is a common platform for both developers and users, it actually needs to create distributions for deployments. As with other Linux distribution projects, the management of the Hadoop ecosystem has similar way. First of all, big data distribution comes with some enterprise advantages that big data processing needs professional approaches. One of is the support functionality assists with technical solutions and turns the platform simple for users at various levels. Also big data distributions have feedback mechanism for patches, fixes, and bug detection. They also give custom attempts for extra add-on instruments.

To build a real data-driven enterprise, where decisions are taken based on data rather than relying on estimations, organizations need data management solution that offers not just data governance but also should be able to manage existing enterprise. There should be solid integration with the existing enterprise infrastructure as well.

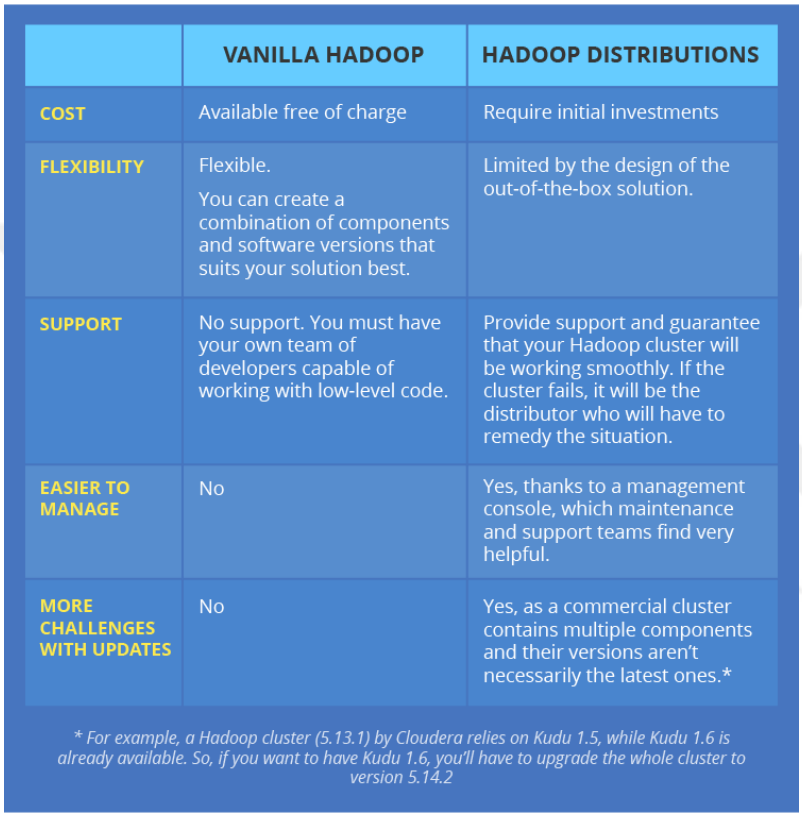
Adding new functionality to big data tasks is easier with Hadoop architecture and is convenient in structure. Thus, vendors implementing Hadoop's open-ended framework developed their code to enhance existing functionality. There is a important, and increasing, divergence between what the remaining distributions offer. This brings a number of challenges for users and developers alike (Chopping 2017):

- It is hard to maintenance and making guide for products
- Solutions should work on across all distributions.
- More distribution-specific projects causes heavily adopting while moving to different distributions.

Furthermore, the use of different tools between distributions causes a divergence. When planning a distribution deployment, it is very important to carefully select your Hadoop

distribution because later rollback can be difficult. If it is necessary to have equivalent options that make it difficult to choose from open source software, it can be questioned whether it serves end users and developers well between deployments.

Big data platform studies are usually based on the Hadoop ecosystem, but can also accommodate additional ecosystems. Besides of this Figure. 2.31 shows differences between vanilla Hadoop and Hadoop distributions.



	VANILLA HADOOP	HADOOP DISTRIBUTIONS
COST	Available free of charge	Require initial investments
FLEXIBILITY	Flexible. You can create a combination of components and software versions that suits your solution best.	Limited by the design of the out-of-the-box solution.
SUPPORT	No support. You must have your own team of developers capable of working with low-level code.	Provide support and guarantee that your Hadoop cluster will be working smoothly. If the cluster fails, it will be the distributor who will have to remedy the situation.
EASIER TO MANAGE	No	Yes, thanks to a management console, which maintenance and support teams find very helpful.
MORE CHALLENGES WITH UPDATES	No	Yes, as a commercial cluster contains multiple components and their versions aren't necessarily the latest ones.*

** For example, a Hadoop cluster (5.13.1) by Cloudera relies on Kudu 1.5, while Kudu 1.6 is already available. So, if you want to have Kudu 1.6, you'll have to upgrade the whole cluster to version 5.14.2*

Figure 2.31. Vanilla Hadoop vs Hadoop Distributions (Chopping 2017)

2.2.1. Hortonworks

In 2011, members of the Yahoo team in charge of the Hadoop project formed HortonWorks and has then quickly developed and opened the Hadoop distributions way. Hortonworks is one of the 100% open source Hadoop distribution of Hadoop with no proprietary software deployed with it. HortonWorks is a big Hadoop contributor, and its economic model based on selling exclusively support and training, instead of selling a license. This distribution is most consistent with Apache's Hadoop platform. By the way, in the end of 2018 Hortonworks merged with Cloudera (BusinessWire 2018).

HortonWorks platform includes these components:

- Distirbuted file system (HDFS/MapReduce)
- Querying (Apache Hive)
- Planning (Apache Oozie)
- Handling queries (Stinger)
- Searching (Apache Solr)
- Management and supervision (Apache Ambari)
- Integration services (Apache Sqoop)
- NoSQL (Apache HBase)
- Coordination (Apache Zookeeper)
- Script Platform (Apache Pig)
- Security (Apache Ranger)
- Learning (Apache Mahout)
- Distributed Log Management (Apache Flume)
- Metadata (Apache HCatalog)

2.2.2. Cloudera

Cloudera Inc. was founded by big data experts from Yahoo, Google, Oracle and Facebook in the year 2008. Cloudera was the first distribution Apache Hadoop based software and is still the largest organization and has widely user base with many customers. Also Cloudera provided some tools such as Cloudera Management for deployment management and Cloudera Search for searching. For example Cloudera Management suite provides the users reduced deployment time, real-time node counts etc. (Mindmajix 2018).

Cloudera, an enterprise data management company, introduced the concept of the enterprise data hub (EDH) that is a central system to store and work with all data. Also it is

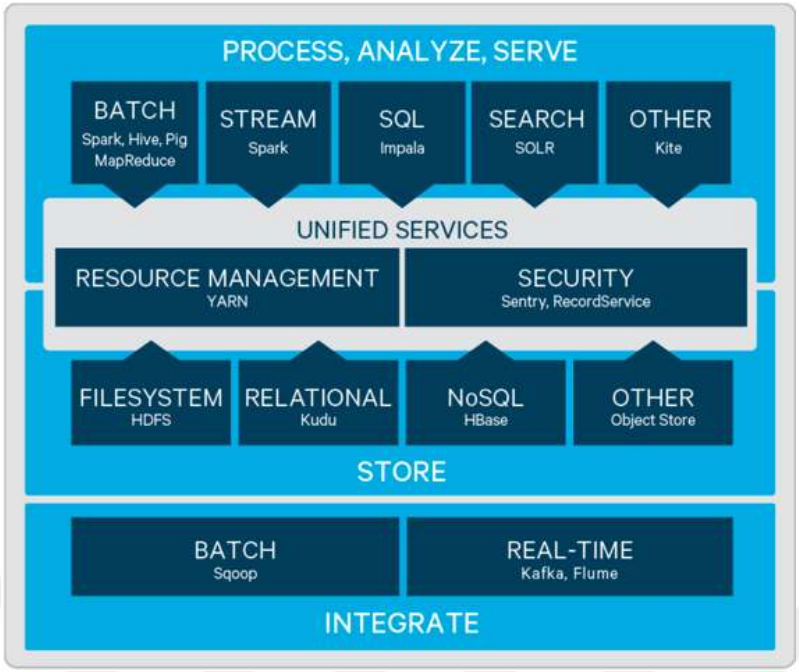


Figure 2.32. Cloudera Techonology Stack (Cloudera 2019)

flexible to run a variety of enterprise workloads for example, batch processing, interactive SQL, enterprise search, and advanced analytics while meeting enterprise requirements such as integrations to existing systems, security, governance, data protection, and management. The EDH is the improvement version of enterprise data management. Cloudera consists of the open source Cloudera Distribution including Apache Hadoop (CDH), a suite of management software and enterprise-class support. By the way, Cloudera announced completion of its merger with Hortonworks, in January 2019 (Cloudera 2019). So CDH and Hortonworks platform merged also in software stack.

2.2.3. MapR

The MapR Converged Data Platform is a solution for providing business value for data-driven companies. The main consept of MapR Hadoop distributions is, focuses on market needs because of market driven entity. Unlike Cloudera and Hortonworks, MapR Hadoop Distribution has a more distributed approach for storing metadata on the cluster nodes because of it uses different file system that called MapR File System (MapRFS) and does not use a NameNode architecture. Figure. 2.33 shows MapR Converged Data Platform.

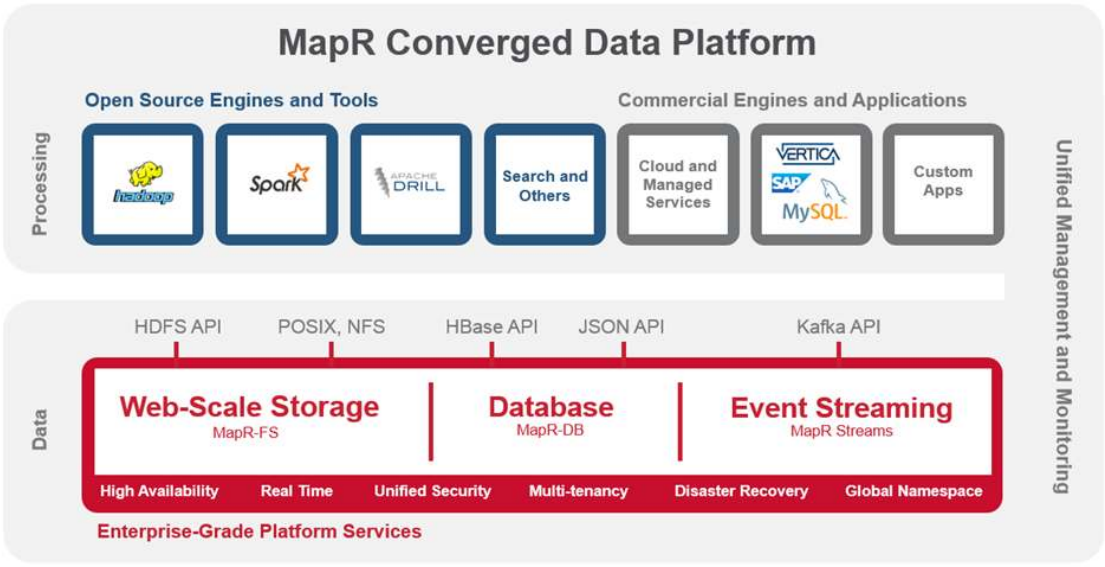


Figure 2.33. MapR Converged Data Platform (MAPR 2019)

MapR company has important product is MapR Data Platform (MDP). MapR Data Platform is software based data platform that can be install on commodity hardware for many use case of processing data. This platform has storage management, processing, and analysis of data for analytic applications and AI. It also provides security across the platform.

Besides of MDP, there is also MEP (MapR Ecosystem Pack) that a broad set of Hadoop ecosystem projects and standard APIs are tested in its process. MapR provides to customers a wide range of solutions meeting custom requirements with these features. Figure. 2.33 shows technology stack of MapR big data platform.

2.2.4. Amazon Web Services

The introduction of cloud service providers such as Amazon Web Services has been instrumental in the growth of big data applications. Amazon Web Services started in 2002, the first services of Amazon S3 cloud storage, SQS (Simple Queue Service) and EC2 were provided in 2006. With the integrated service system provided by Amazon Web Services, it can be use on data lake for analytical applications. (Reeves 2019).

Amazon Web Services make it easy to access big data without compromising governance and security with machine learning and analytic services that it provides. Also with

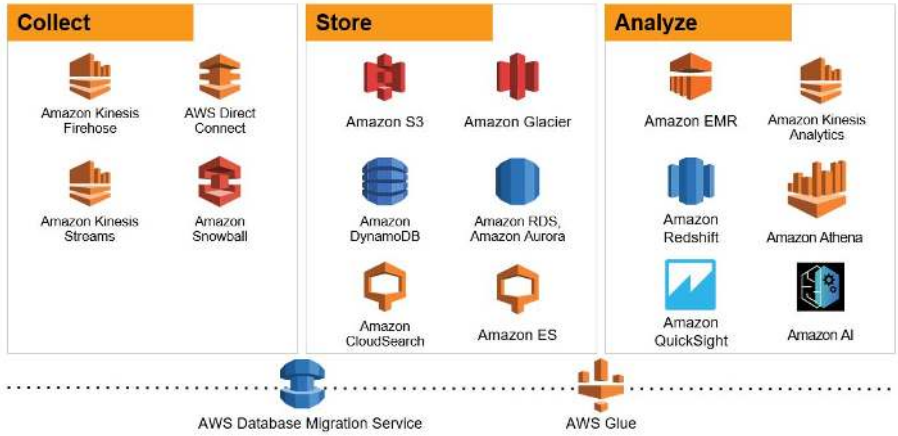


Figure 2.34. Amazon Web Services Big Data Platform (AWS 2019)

the advantage of flexible use of services, organizations do not need to undertake long-term investment in the use of the latest technologies, the service continues to be provided as it uses. As companies data grows, AWS also helps organizations to create more business value at low cost. Figure. 2.34 shows AWS big data ecosystem.

In addition, Amazon Web Services (AWS) has the Amazon Elastic MapReduce (EMR) service for large data processing and analysis. This service provides a low-configurable service to in-house block calculation operations that can be expanded with a different structure. Amazon EMR is based on Apache Hadoop, which supports the processing of large data sets on distributed nodes. Amazon EMR processes large data in the Hadoop cluster on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). The word elastic in the name of EMR refers to the dynamic resizing capability that allows it to increase or decrease resource use at any time, depending on demand. Amazon EMR is used for data analytics on web indexing, data storage, machine learning, log analysis, financial analysis, scientific simulation, bioinformatics and more. EMR also supports components such as Apache Spark, Presto, and Apache HBase for additional functionality.

2.2.5. Microsoft Azure Big Data Platform

In big data technologies, Azure has succeeded in painful points. Spinning up is possible to any technology in a VM that provides by Azure, so PaaS services are interested in the decisions are between only consistent and industry can get big data results consis-

tently with analytical solutions. The following Figure. 2.35 shows the main technologies that can be involved in a Big Data architecture in Azure.

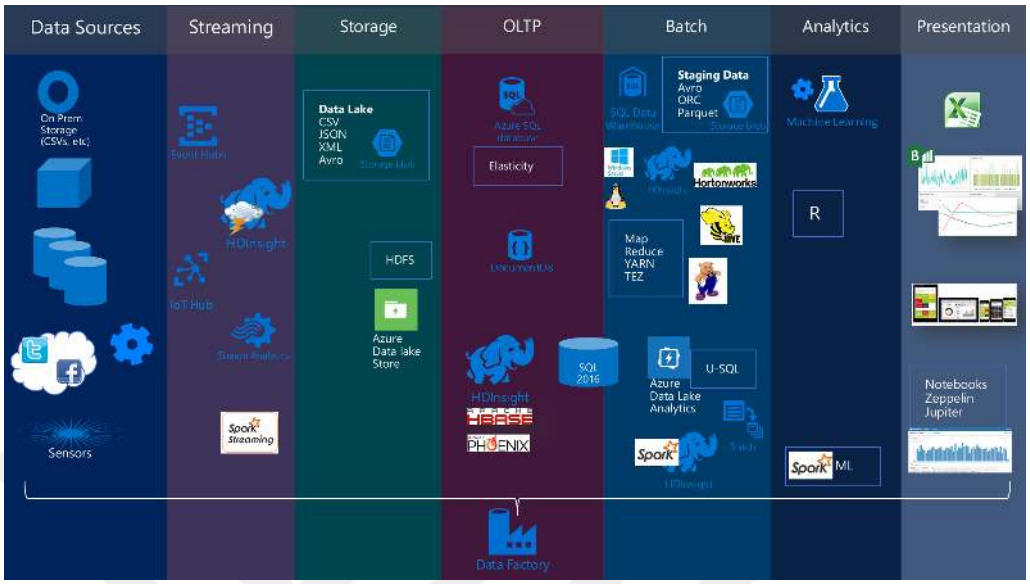


Figure 2.35. Microsoft Azure Big Data Architecture (Lester 2016)

These technologies are categorised in sections to understand them more clearly. Azure has many services that can be used in a big data architecture. We can separate them two categories:

- Managed services that contains Azure Data Lake Store, Azure Data Warehouse, Azure Data Lake Analytics, Azure Event Hub, Azure Stream Analytics, Azure Data Factory and Azure IoT Hub.
- Open source technologies based on the Apache Hadoop ecosystem.

These options are not fixed so many open source solutions can uses with Azure services.

2.2.6. Pivotal HD

Pivotal, the big data and cloud company that appeared from VMware and EMC in 2013, is open source its investments based on big data technologies and it is working collaboratively with IBM, Hortonworks and other companies on a Open Data Platform Project. Pivotal is still selling licenses and support for its Greenplum, HAWQ and GemFire database products, but it is also releasing the core code bases for those technologies as open source.

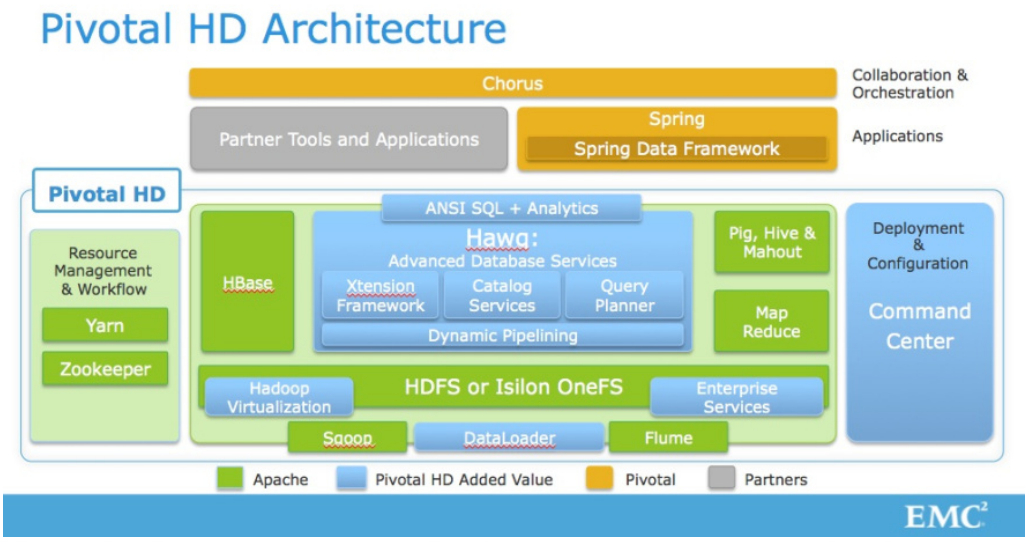


Figure 2.36. Pivotal HD Architecture (PivotalHD 2015)

In March 2013, Pivotal HD was announced based on Hadoop, containing a version of Hawq (Greenplum fork). There is also Pivotal HD Enterprise for commercially. The Figure. 2.36 below shows how each Apache and Pivotal components integrated in Pivotal HD.

2.2.7. IBM Cloud Big Data Platform

IBM team proposed and architecture that split into three major categories to address the data flows:

- Batch jobs execution
- Near real-time
- Real-time

All three workflows needs data ingestion. Preferred solution is all data collecting through Apache Kafka data across with different channels. Although existing platform already used Kafka and other ingestion sources, so suggested multiple ingestions into Kafka and leveraging it as a service on IBM Cloud through IBM Message Hub. Because Kafka ensures guaranteed delivery for streaming messages for real-time application. Since Kafka is an open-source software without license problem.

For IBM Streaming Analytics on IBM Cloud to combine real-time and near real-time application use cases for message processing. IBM Streams is technology comes with to scale for up many of users. Although IBM Streams is not open source, it supports the development of applications with Apache Beam, which is a programming model that allows users to create platform-agnostic streaming applications. Also it has an adapter to read/write from Kafka and can apply a machine-learning model to make predictions.

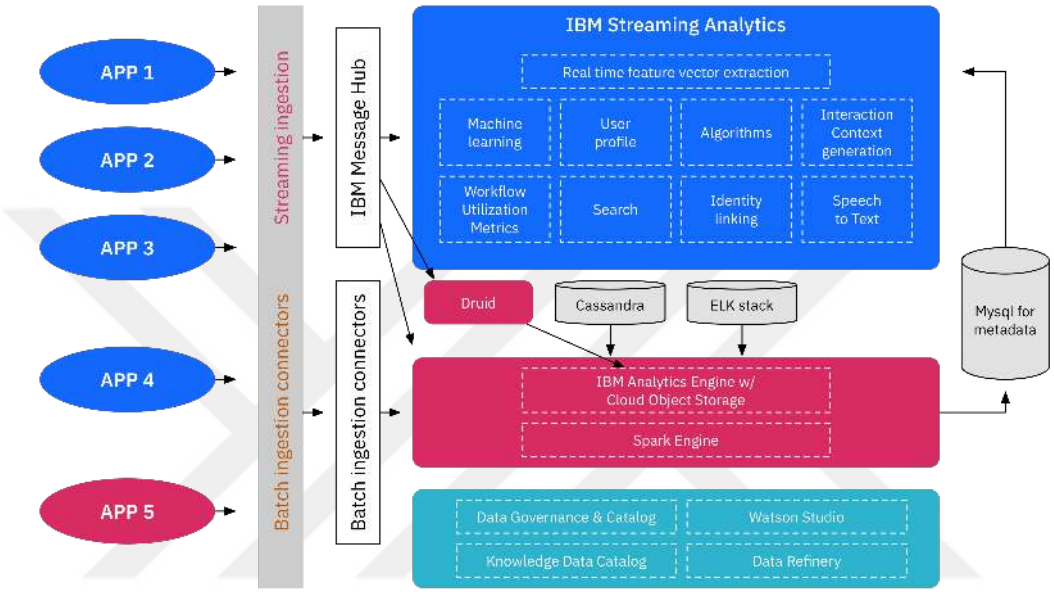


Figure 2.37. IBM Big Data Architecture (IBM 2019)

IBM Streaming Analytics process the messages in use case of real-time or near real-time data ingestion. Consumers of IBM Streams output will go into the downstream component with different components depending on the workflow. Additionally, the decision engine in the data pipeline will be triggered through the IBM Streams operator based on the type of event. The events data can be stored in a NoSQL database for historical purposes and for real-time/near real-time decision making. The client was already using in-memory database as a column data store, and it uses for OLAP queries, which can provide near real-time dashboarding capability for the interactive user applications needs (IBM 2019). Details are shown at Figure. 2.37

2.2.8. Google Cloud Platform

GCP (Google Cloud Platform) is a cloud information platform in which Google provides server infrastructure services used by customers such as big sites like Youtube. From simple to complex applications, different types of software can be developed through Google Cloud. This distribution of components comes with advantages, including consistency towards to fails and low latency by locating resources closer to clients. Because GCP has several kind of pyschical and virtual assets that are placed in Google's data centers across globe. Each data center location is in a global region. Also each region is a collection of zones, which are isolated from each other within the region (GCP 2019).

Cloud computing converts software and hardware products to services. These services provide access to the underlying resources. GCP has many services in available and they are growing. When you develop your application or web service on GCP, you can make combinations with these services and that provide the infrastructure you need, and then you can add your code and build in your service infrastructure.

3. MATERIAL AND METHOD

3.1. MILIS Linux Operating System

MILIS Linux base system is used as the foundation for the big data platform. Hadoop ecosystem reside on Linux userspace. MILIS, which is created from Scratch (LFS), includes a Package Manager(MPS - MILIS Package System) that is used to manage applications in userspace of the Operating System.

Base system of MILIS consists of approximately 100 packages including kernel, firmwares, built tools, compression, text processing, network, C/C++ compiler, disk, startup, git, basic security and authorization tools. Graphics server and desktop modules are not be needed for a base system. The applications hosted by the base system is sufficient for the access and management of software.

3.1.1. MPS - MILIS Packaging System

MPS is responsible for building the system from scratch and managing the packages. Package management also includes the process of compiling packages. Directive files are used for generating packages with their content. The directive files contain the headers, dependencies, source addresses, compilation and packaging procedures of a package. MPS also checks the update status of the complete system through the collection of directives. Directives lay at following levels.

- 0- Bootstrap directives
- 1- Base system directives
- 2- Common system directives
- 3- Test directives

3.2. Big Data Platform Design on MILIS Linux

For the big data platform, the infrastructure needs to be well structured. We choose Hadoop and its ecosystem as the foundation. Three basic approaches are used in the construction of the remaining softwares of the platform.

- **Components Repo:** Big data platform is driven by high-level component manager that can interact with the system package manager underneath. The management of components is achieved by the component manager either by command line or web interface such as Ambari in Hortonworks and Cloudera.

- **Package Repo:**Although a component manager administers the components, package manager plays an important role in managing these components at low-level. Hence, installation, removal and updates are handled by the package manager of Linux distribution.

- **Container:**It is the method of creating and using containers such as Docker or LXC (Linux Containers) as the base system which also includes big data software pre-installed.

Unlike other big data platforms such as Hortonworks, Cloudera, where components are managed with a special interface, in our system, MPS directly manages the components. Therefore, the proposed system is simpler.

The project is broken down to 3 phases:

Build Preparation: First directives of selected packages are prepared. Next, package instruction ("talimat") scripts are collected at a different directive level. They are stored in a Git version track repository. By adding a new big data level to build environment, packages are assembled.

Installing and Configuration: After producing packages, packages will upload to our big data package repo instead of base repo of MILIS. So we can manage separately related packages then when we add package repo to MPS config, we will able to be install our new packages. Configuration will do by manually to provide more stability.

Testing and Verification: Management is orchestrated by MPS but dependency of the big data software components is still needs to be tested. In the testing phase, components are tested individually and verified to work together. During testing, re-packaging or re-configuring occurred.

The components discussed in the literature section are the most used and current technology for big data. There are other software than those described above. However, as the scope of this study is to build a light-weight big data platform that has the widest adaptation, we limit the technologies to the ones discussed in literature review section.

For each of the technology that was integrated, the lifecycle included the compilation, installation, and testing phases starting with the source code for optimum packaging.

MILIS build process require each directive file (talimat) of the package to be committed into GIT repository. Components may need additional packages during build and/or runtime. During the built phase, dependencies of related packages are checked. MILIS built process which is managed by MPS is carried out in a container that is isolated from the host. Pre-installation and post-installation scripts setup the installation environment and configure the software afterwards respectively.

During the installation phase, packages are registered and uploaded to the Milis package repository. MPS eases installation, update and removal of the components of the big data platform. If an update is applied to a component, MPS manages to trigger dependent software as well.

In the following sections, directive preparation, compilation, installation and test steps of each big data component is explained in detail.

3.2.1. Distributed File System

Since our project is built on Hadoop ecosystem. HDFS is used as the main distributed file system. However, It is important to select the proper version of Hadoop as the dependent software may not work with the selection.

Hadoop is currently released in version 2.x and version 3.x. The reason for continuing 2.x is the need to be backward compatible with some dependent applications. Note that there are some important differences between Hadoop 2.x vs Hadoop 3.x (Data-Flair 2019) as listed below;

1. License
    - 2.x – Apache 2.0 and Open Source
    - 3.x – Apache 2.0 and Open Source

2. Java versions
    - 2.x – Minimum version of Java is 7.
    - 3.x – Minimum version of Java is 8.
3. Fault Tolerance
    - 2.x – Replication handles (which is wastage of space).
    - 3.x – Erasure coding handles.
4. Data Balancing
    - 2.x – For data, balancing uses HDFS balancer.
    - 3.x – For data, balancing uses Intra-data node balancer.
5. Storage
    - 2.x – As replication scheme uses 3.X mechanism and hence has 200% overhead in storage space (for 6 block has 18 block space because of replication scheme) (Hadoop 2019)
    - 3.x – Erasure encoding is supported in HDFS and has 50% overhead in storage space (for 6 block has 6 block space and 3 block parity)
6. YARN Timeline Service
    - 2.x – Old timeline service in usage
    - 3.x – Timeline service v2 that has more stability and scability
7. Default Ports Range
    - 2.x – Some default ports are in Linux short-lived port range so they may fail at booting
    - 3.x – Related ports out of in Linux short-lived port range, less likely to fail

Also 3.x has improved compatible file system attributes. Due to its advantages, we choose to base our distribution on Hadoop 3.x version. Directive file of Hadoop is as follows.

```
[paket]
tanim   = Apache Hadoop dağıtık dosya sistemi
paketci = milisarge
grup    = büyükveri
url     = http://hadoop.apache.org

[gerek]
derleme = maven openjdk8 protobuf2
calisma =

[kaynak]
1 = https://www.apache.org/dist/hadoop/core/
$isim-$surum/$isim-$surum-src.tar.gz

[sha256]
1 = 3bd31357c047bf85cf18356df1405b5f4c36
f87caa27f519fb8ef4fdd6750324

[derle]
betik = cd $isim-$surum-src
betik = export JAVA_HOME=/usr/lib/jvm/java-8-openjdk
betik = make-ca -g
betik = mvn -e package -Pdist -DskipTests
-Dtar -Dmaven.javadoc.skip=true -Dmaven.repo.local=/opt/m2

[pakur]
betik = cd $isim-$surum-src
betik = mkdir -p $PKG/usr/mdp
betik = cp -r hadoop-dist/target/* $PKG/usr/mdp
```

Hadoop needs protobuf 2.5.0 to compile which is added in the dependency-build section. At the end of build, we get a hadoop-3.2.1.tar.gz archive file under hadoop-dist/target

directory which is then placed under the package preparation directory. Following components are installed to /usr/mdp. MDP refers to MILIS Data Platform.

3.2.2. Cluster Coordinating

ZooKeeper is a highly available coordination service for Hadoop ecosystem and became the de-facto software for cordination. It is originated in the Hadoop project allows one to choose or implement custom locking, fail over, leader election, group membership and other coordination software on top of it. We will use 3.5.x version of Zookeeper. Directive file of Zookeeper is;

```
[paket]
tanim   = Güvenilir dağıtık koordinator uygulaması
paketci = milisarge
grup    = büyükveri
url     = http://zookeeper.apache.org

[gerek]
derleme =
calisma = openjdk8

[kaynak]
1 = https://archive.apache.org/dist/$isim/
$isim-$surum/apache-$isim-$surum-bin.tar.gz

[sha256]
1 = db24700e453f2ad4b6b789d553fe828ccc
eef3977a7e6b36389580ec92397a7e

[derle]
betik = cd apache-$isim-$surum-bin
betik = echo "derleme yok"
```

```
[pakur]
betik = cd apache-$isim-$surum-bin
betik = mkdir -p $PKG/usr/mdp/$isim
betik = cp -r bin conf lib $PKG/usr/mdp/$isim/
```

We choose 3.5.6 version of Zookeeper. Due to trouble in compiling zookeeper, binary version (JAR) of the Zookeeper software was deployed into repository.

3.2.3. Data Collection

We choose Apache Kafka as message queue and data collector due to its speed and high data processing capacity either in batch or real-time contributed to the large cache system on disk. In addition Kafka has wide support for a variety of programming languages. After Jun 2019, it released 2.3.0, so we will use 2.3.x version of Kafka (Kafka 2019). Directive file of Kafka is;

```
[paket]
tanim   = Yüksek verimli dağıtılmış mesajlaşma sistemi
paketci = milisarge
grup    = büyükveri
url     = https://kafka.apache.org

[gerek]
derleme =
calisma = openjdk8 zookeeper

[kaynak]
1 = https://archive.apache.org/dist/$isim/
$surum/kafka_2.12-$surum.tgz
dosya = $isim-profile.sh

[sha256]
1 = 5a3ddd4148371284693370d56f6f66c7
```

```
a86d86dd96c533447d2a94d176768d2e
2 = aa63f4be4f696a14279407f92cc13c6f
001cf3fe54664883f2c68247e8a1d867

[derle]
betik = cd kafka_2.12-$surum
betik = echo "derleme yok"

[pakur]
betik= cd kafka_2.12-$surum
betik= mkdir -p $PKG/usr/mdp/$isim
betik= cp -r bin config libs $PKG/usr/mdp/$isim/
betik= mkdir -p $PKG/etc/profile.d
betik= cp $SRC/$isim-profile.sh $PKG/etc/profile.d/$isim.sh
```

At sources section, there is a file for profiling Kafka under /etc/profile.d directory. It refers to KAFKA_HOME="/usr/mdp/kafka" so for other components and run binaries can be found easily using the *home* variable. Compiling Kafka with Gradle that is already in the package repository gave some sort of module error that we could not figure out in this version. Therefore, compiled version of Kafka with Scala 2.12 is deployed.

3.2.4. Execution Engine

As query execution engine; MapReduce, Tez and Spark are options. MapReduce comes as default, but is not preferred due to its disadvantage in speed in comparison to Tex and Spark.

Tez mainly depends on Yarn resource manager. Spark can run standalone without Yarn. Both of them uses directed acyclic graphs while processing data (Parker 2019). Spark may be installed on any Hadoop cluster. Therefore, Spark instead of Tez is chosen as we wanted to lightly couple our components with each other. Consequently, we can change any components with a better one if needed or desired. Spark has also advantages over Tez due to its real-time and machine learning support. Spark version 2.4.x which is compatible with Hadoop 3.x is chosen. Directive file of Spark is as follows.

```
[paket]
tanim   = Geniş ölçekli veri işleme motoru
paketci = milisarge
grup    = büyükveri
url     = https://spark.apache.org

[gerek]
derleme =
calisma = openjdk8 python

[kaynak]
1 = https://archive.apache.org/dist/spark/
spark-$surum/spark-$surum-bin-without-hadoop.tgz
dosya = $isim-profile.sh

[sha256]
1 = a245ddc05c19a3d8b6a225bb518b55c6e
4c78de1c5d576fbdc791c40d495ea56
2 = a30b65d2472e3439e433693b86cf0ebd2
48c125f841040541f148a6f45a1dfc3

[derle]
betik = echo "derleme yok"

[pakur]
betik = cd $isim-$surum-bin-without-hadoop
betik = mkdir -p $PKG/usr/mdp/$isim
betik = cp -r bin conf data examples
jars python sbin yarn $PKG/usr/mdp/$isim/
betik = mkdir -p $PKG/etc/profile.d
betik = cp $SRC/$isim-profile.sh
```

```
$PKG/etc/profile.d/$isim.sh
```

At sources (kaynak) section, there is a file about profiling Spark under /etc/profile.d directory. It is set to SPARK_HOME="/usr/mdp/spark". Other components and running binaries may work with Spark using SPARK's home variable that is set. Pre-compiled version of Spark without Hadoop is deployed which also includes Scala version 2.12.

3.2.5. Data Workflow

Data pipeline related tasks are first performed manually, then, as things need to scale up, the process is automated and triggered with cron like tool. Ultimately, we reached a point where the good old cronjob is not able to guarantee a stable and robust performance for the monitoring and managing of the big data flow progresses. It's simply not enough anymore. So we need a data flow management system. Oozie, Airflow and Luigi are job schedulers that was mentioned in the literature review section. Apache Oozie uses XML property files and a SQL database to log metadata for task management. Airflow and Luigi does task orchestration with python based scripts. Note that, when we look to community contributions of Oozie vs others. Airflow and Luigi has more contribution, likely due to support for python. When Airflow and Luigi is compared, Luigi achieves actual pipeline mechanism with a simpler concept (Peppoloni 2018). Luigi tasks on pipeline share input and output information and is connected together and has target-based approach. Also with its simplicity of defining tasks, we prefer to use Luigi version 2.8.x. However, Airflow may also be a good choice for data workflow.

3.2.6. Data Integration

For data integration, existing export tools of existing databases are used. Although Sqoop may seem advantageous, the latest version 1.99.x (Sqoop2) is not compatible with Hadoop 3.x. Due to integration issues, the use of Sqoop may be postponed for future studies.

3.2.7. Real-Time Analytics

For an analytics database, Druid may be a good choice due to reduced latency. Druid merges the best qualities of a column store and inverted indexing. Columnar storage is

better for analytical queries due to low I/O costs. Support for complex joins and comprehensive SQL is not the main objective of Druid. Apache Hive on the other hand tries just that (Cloudera 2019).

For the big data warehouse, we choice Apache in our design. A compatible low latency, analytical database is needed on top of Hive that is also able to work with Hive. Druid which can also be queried through Hive seem to be the natural fit. Additionally Apache Druid offers a simple programming interface that is easily accessible via HTTP. We deploy 0.16.x version of Druid in our package. Directive file of Druid is;

```
[paket]
tanim   = Apache Hive veri ambarı uygulaması
paketci = milisarge
grup    = büyükveri
url     = https://hive.apache.org

[gerek]
derleme = maven python-yaml
calisma = openjdk8 python

[kaynak]
1 = https://github.com/apache/incubator-druid/
archive/druid-$surum-incubating.tar.gz
dosya = $isim-profile.sh

[sha256]
1 = b8aa553d80ae624e44d2f7f9c144e3
39bb97dc9191ac025854804420ebc9e55e
2 = 3fe2d03eff76285d78bf3e9dbff9c8
286473deed7117133d498c2891ed353c24

[derle]
betik = cd incubator-druid-druid-$surum-incubating
```

```
betik = make-ca -g
betik = source /etc/profile.d/openjdk.sh
betik = pathappend $JAVA_HOME/bin
betik = export MAVEN_OPTS="-Xmx2g
-XX:ReservedCodeCacheSize=512m"
betik = mvn clean package -DskipTests
-Pdist -Dmaven.repo.local=/opt/m2/repository

[pakur]
betik  = cd incubator-druid-druid-$surum-incubating
betik = mkdir -p $PKG/usr/mdp
betik = tar xf distribution/target/
apache-druid-$surum-incubating-bin.tar.gz -C $PKG/usr/mdp/
betik = mv $PKG/usr/mdp/apache-
druid-0.16.0-incubating $PKG/usr/mdp/druid
betik = cp -r examples $PKG/usr/mdp/druid/
betik = mkdir -p $PKG/etc/profile.d
betik = cp $SRC/$isim-profile.sh
$PKG/etc/profile.d/$isim.sh
```

At sources(kaynak) section, there is a file about profiling Druid under /etc/profile.d directory. It refers to DRUID_HOME="/usr/mdp/druid" so for other components and running binaries can find it easily via Druid's home variable.

3.2.8. NoSQL Database

NoSQL stores unstructured data across multiple data nodes, as well as multiple servers. NoSQL distributed database infrastructures are the De-facto solution for the largest data warehouses. Among the NoSQL DB candidates, Hbase, Cassandra and MongoDB seem to be popular ones.

Unlike MongoDB, Cassandra has a ring architecture without a master which has advantages over master-slave systems. This means that all nodes has equal treatment in a cluster and a majority of nodes can be used to ensure quorum. Although Cassandra stores

data in columns and rows like a traditional RDBMS, it provides nimbleness in the sense that it allows rows to have different columns, and even allows a change in the format of the columns. Also Cassandra has Cassandra Query Language (CQL), similar to the traditional SQL syntax, and thus is easier to adapt for developers.

HBase shares several similarities with Cassandra, one major difference in its architecture is the use of a master-slave architecture. This also proves to be a single point of failure, as failing from one node to another can take time, which can also be a performance issues (Klein 2019). If we consider an always-available system and easy of use through a query language then the Cassandra is the right choice. With these advantages, we deploy 3.11.x version of Apache Cassandra. The directive file of Cassandra is:

```
[paket]
tanim   = Apache Cassandra NoSQL veritabanı
paketci = milisarge
grup    = büyükveri
url     = https://cassandra.apache.org

[gerek]
derleme = apache-ant
calisma = openjdk8 python procps-ng

[kaynak]
1 = https://www.apache.org/dist/cassandra/
3.11.5/apache-$isim-$surum-src.tar.gz
dosya= $isim-profile.sh

[sha256]
1 = d5363386b6ddbee59cf7dbbb2564a5f23c
a236e728eebce12daf83b5ccdd50d8
2 = c932b440260c5ef4c48ed91c752ca67d70
8b1333a834ea0006fc93504368886a
```

```
[derle]
betik = cd apache-$isim-$surum-src
betik = make-ca -g
betik = source /etc/profile.d/openjdk.sh
betik = pathappend $JAVA_HOME/bin
betik = ant

[pakur]
betik = cd apache-$isim-$surum-src
betik = mkdir -p $PKG/usr/mdp/$isim
betik = mv build/*.jar lib/
betik = cp -r bin conf interface lib pylib
tools $PKG/usr/mdp/$isim/
betik = mkdir -p $PKG/etc/profile.d
betik = cp $SRC/$isim-profile.sh
$PKG/etc/profile.d/$isim.sh
```

At sources(kaynak) section, there is a file for profiling Cassandra under /etc/profile.d directory. It refers to CASSANDRA_HOME="/usr/mdp/cassandra". Other components and running binaries can easily find Cassandra via its set *home* variable.

3.2.9. Data Warehouse

We choice Apache Hive as data warehouse technology, as it seem to play well with the Hadoop ecosystem. It has the following features.

- Built-in on top of HDFS as distributed architecture
- Includes SQL-like declarative language called HiveQL (HQL)
- Table structure seems like relational database
- Efficent data extract/transform/load (ETL) feature
- Supports variety of data formats. Also converting variety of format from to other within Hive.

Apache Hive is preferred technology for data lakes. Hive can be integrated with Druid and Spark and also data can be exchanged in a common space. We chose version 3.1.x which is compatible with Hadoop 3.x.y version. The directive file of Hive is:

```
[paket]
tanim   = Apache Hive veri ambarı uygulaması
paketci = milisarge
grup    = büyükveri
url     = https://hive.apache.org

[gerek]
derleme = maven
calisma = openjdk8 python

[kaynak]
1 = http://us.mirrors.quenda.co/apache/
hive/$isim-$surum/apache-$isim-$surum-src.tar.gz
dosya = $isim-profile.sh

[sha256]
1 = 5cfc0c988f6279737dad415835f3c8cf
492ae09ab09d13d529bb22b6068c3ee2
2 = d7151fdc918c631243dba10c252f2a90
1f4628ba8b954957a6de32d1a69a7b69

[derle]
betik = cd apache-$isim-$surum-src
betik = make-ca -g
betik = source /etc/profile.d/openjdk.sh
betik = pathappend $JAVA_HOME/bin
betik = export MAVEN_OPTS="-Xmx2g
 -XX:ReservedCodeCacheSize=512m"
```

```
betik = mvn clean package -DskipTests
-Pdist -Dmaven.repo.local=/opt/m2/repository

[pakur]
betik = cd apache-$isim-$surum-src/packaging/
target/apache-$isim-$surum-bin/apache-$isim-$surum-bin
betik = mkdir -p $PKG/usr/mdp/$isim
betik = cp -r bin conf examples jdbc lib
scripts hcatalog $PKG/usr/mdp/$isim/
betik = mkdir -p $PKG/etc/profile.d
betik = cp $SRC/$isim-profile.sh
$PKG/etc/profile.d/$isim.sh
```

At sources(kaynak) section, there is a file for profiling Hive under /etc/profile.d directory. It refers to HIVE_HOME="/usr/mdp/hive" Other components and running binaries can find Hive easily via its set its home variable. Hive is compiled from source and packaged under /usr/mdp successfully.

3.2.10. Machine Learning

Doing personalization, recommendation, and predictive insights on big data is quite hard with traditional methods. So we need a scaleable, fast and easy to use machine learning software. Apache Spark provides a general purpose machine learning library that is designed for simplicity, scalability, and easy integration with widely used other big data modules.

The key benefit of Spark MLlib is that, it allows data scientists to focus on their data problems and models instead of solving the complexities surrounding the distributed data infrastructure and configurations (Bradley et.al 2016). Also using Python, Spark can interact with Luigi to manage process and data flow. Spark being strong as an execution engine make its extension Spark MLlib to be a natural choice.

Figure 3.38. MILIS Big Data Platform Design

3.2.11. Management

From input to output, big data platform has to include various software working in harmony. In our study, we not only selected the most essential modules but also the least resource hungry and light weight for the job. In the distribution that is prepared, we aim to address the most common business requirements with minimal software while avoiding overlapping components. If host operating system can handle, we avoid to add yet another 3rd party software in areas such as security, process management and monitoring.

Ambari handles management and deployment of components. Ambari also monitors the system and reports statistics. In our approache, installing/removing of big data modules are through MILIS package manager. If pre/post configurations are needed in between installations, scripts written in Bash, Python and Lua programming languages are executed.

The applications on the platform can access the relational database to store their metadata. These metadata contain internal settings of components and their relationship to other components. For this purpose, we use Postgresql relational database which is already shipped with MILIS Linux repos as version 11.5.

Security includes authentication, authorization, audit tracking and data protection sections. For accomplishing these steps we use default Linux applications. Authentication can be handled by the Linux user management mechanism and ACL (Access Control Lists). Also there is Kerberos which is an authentication protocol that provides secure network login. Kerberos uses tickets that are encrypted which can help reduce the amount of times passwords need to be sent over the network. As authorization, Polkit application may be used where the clients may request authorization to carry out tasks defined in XML files (Die.net 2019). Encryption may be achieved by the OpenSSL application. Consequently, a security approach that blends the security tools in the system may be implemented.

Final MILIS Big Data Platform design is shown in the Figure 3.38.

3.3. Individual Modules Setup

In this study, the big data modules and software are compiled from source code, then packaged and tested on the MILIS Linux. Selected modules for the proposed big data platform is presented in Figure 3.38.

Mostly of modules compiled from source code. For some such as Kafka, Spark we used pre-compiled versions due to conflict with the systems pre-existing Gradle build tool. Packages are defined in directive file with bin suffix of the source code. Following the releases of these packages, compiling from source will be reconsidered.

Since MILIS big data platform is intended to provide functionality with few tools, it is lightweight. Note that a typical workflow for a big data application include, data collection, cleanup, storage, summarizing and analysis. We first report on configuration and testing of components.

3.3.1. Apache Hadoop Setup

First, we execute the following command after entering $HADOOP_HOME

```
$ bin/hadoop
```

This will display the help menu of Hadoop. The following example copies the conf directory to use as an input and then locates and calculates the given regular expression. Output will be in output folder.

```
$ mkdir input
$ cp etc/hadoop/*.xml input
$ bin/hadoop jar \
 share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar\
 grep input output 'dfs[a-z.]+'
$ cat output/*
```

If we see _SUCCESS files under output directory, MapReduce is running properly.

Hadoop can start in different modes. In this study, for proof of concept, pseudo-distribution is selected where Hadoop runs on a single-node where each Hadoop daemon runs in a separate Java process.

We chose these configurations; in etc/hadoop/core-site.xml:

```
<configuration><property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property></configuration>
```

In etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
      <name>dfs.datanode.data.dir</name>
      <value>/hdfs/data</value>
      <final>true</final>
    </property>
```

```
     <property>
      <name>dfs.namenode.name.dir</name>
      <value>/hdfs/namenode</value>
      <final>true</final>
    </property>

    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Before we should make sure to login to the localhost without a passphrase.

```
$ ssh localhost
```

If we can't login to localhost without a passphrase, then we need to add our id_rsa key to authorised_keys file.

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

To setup environment, also add the following lines to the etc/hadoop-env.sh

```
export JAVA_HOME="/usr/lib/jvm/java-8-openjdk"
export HDFS_NAMENODE_USER="hadoop"
export HDFS_DATANODE_USER="hadoop"
export HDFS_SECONDARYNAMENODE_USER="hadoop"
export YARN_RESOURCEMANAGER_USER="hadoop"
export YARN_NODEMANAGER_USER="hadoop"
export HADOOP_MAPRED_HOME="/usr/mdp/hadoop"
```

Using the following commands, we run the mapreduce example given before in a single Hadoop node at HDFS.

1. Prepare the filesystem of HDFS:

```
$ mkdir /hdfs && bin/hdfs namenode -format
```

2. Starting of name and data nodes:

```
$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [milistgn]
```

After starting daemons, We can check our Name node at http://localhost:9870/, it should be seen as in that Figure 3.39 and DataNode at http://localhost:9864/ as shown in Figure 3.40.

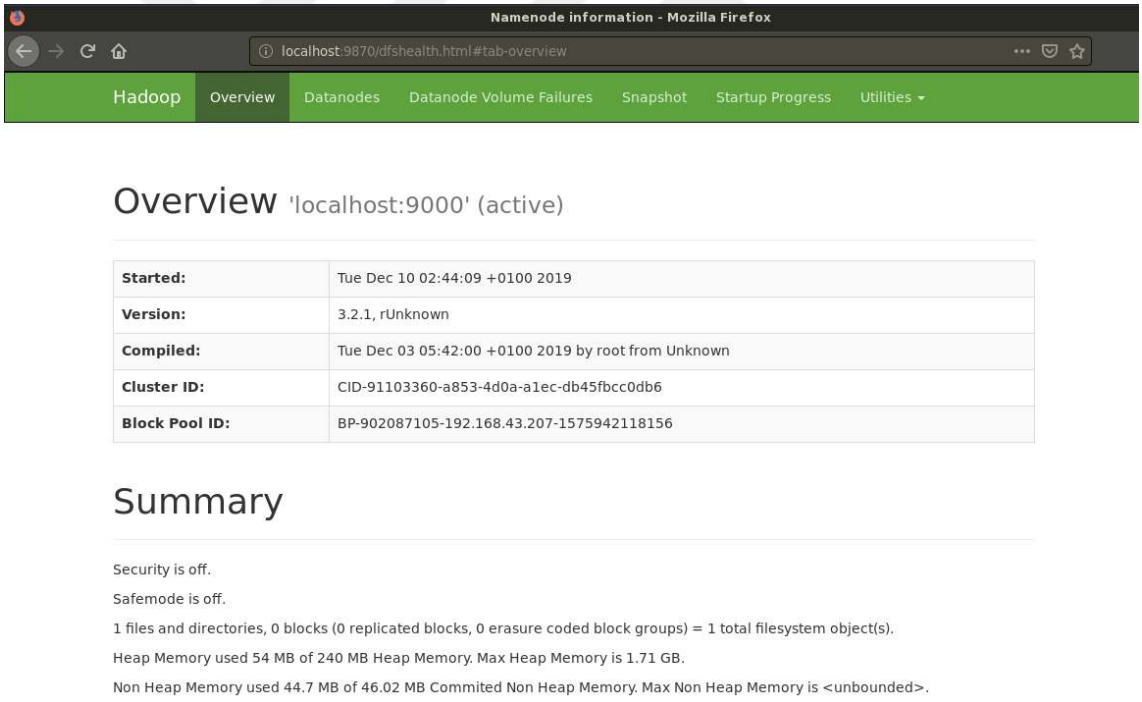


Figure 3.39. Hadoop NameNode Web UI

3. Prepare HDFS directories for MapReduce example:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/hadoop
```

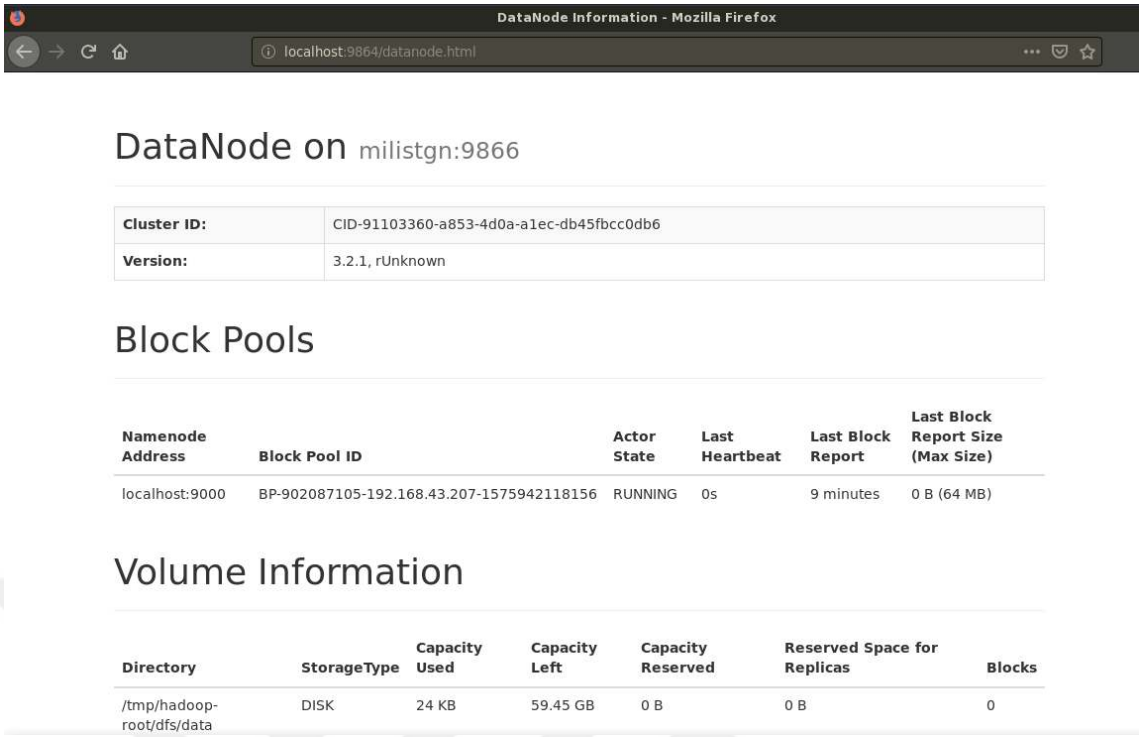


Figure 3.40. Hadoop DataNode Web UI

4. Transfer input files to HDFS:

```
$ bin/hdfs dfs -mkdir input
$ bin/hdfs dfs -put etc/hadoop/*.xml input
```

5. Execute the mapreduce example:

```
$ bin/hadoop jar share/hadoop/mapreduce/ \
 hadoop-mapreduce-examples-3.2.1.jar \
 grep input output 'dfs[a-z.]+'
```

6. Check output directory under HDFS and we should see _SUCCESS file:

```
$ bin/hdfs dfs -ls output
Found 2 items
-rw-r--r-- 3 root supergroup 0 2019-12-10
 02:59 output/_SUCCESS
-rw-r--r-- 3 root supergroup 11 2019-12-10
 02:59 output/part-r-00000
```

3.3.2. YARN Setup

The following commands will set the resource manager as YARN. In etc/hadoop/mapred-site.xml:

```
<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<property>
    <name>mapreduce.application.classpath</name>
    <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/
    *:$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/
    *</value>
</property>
</configuration>
```

In etc/hadoop/yarn-site.xml:

```
<configuration>
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.env-whitelist</name>
    <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,
    HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,
    HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
</property>
</configuration>
```

Starting of YARN daemons.

```
$ sbin/start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```

We can check YARN from web interface, it should be seen as in Figure 3.41.

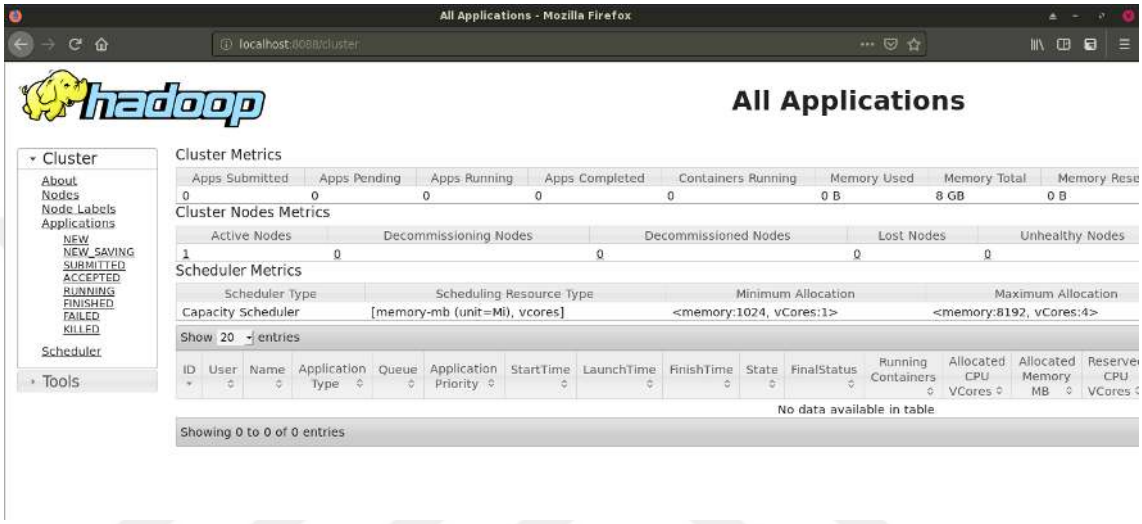


Figure 3.41. Hadoop Yarn Web UI

Note that, we can use start-all.sh/stop-all.sh script under sbin/ for starting and stopping of all daemons.

3.3.3. Apache Hive Setup

After instalation Hadoop, first we should prepare the system for Hive setup with the commands:

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir -p /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

We can run Hive shell with the following command:

```
$ $HIVE_HOME/bin/hive
```

We had two troubles running Hive. First, hive script had a bug dealing with locale in Bash. We add "export LC_ALL=C" to the beginning of "libexec/hadoop-functions" script file. Second one caused by the incompatibility of guava library, so we resolved this issue by using a newer version of the guava library with Hive as copied below:

```
$ rm /usr/mdp/hive/lib/guava-19.0.jar
$ cp /usr/mdp/hadoop/share/hadoop/common/lib/ \
 guava-27.0-jre.jar  /usr/mdp/hive/lib/
```

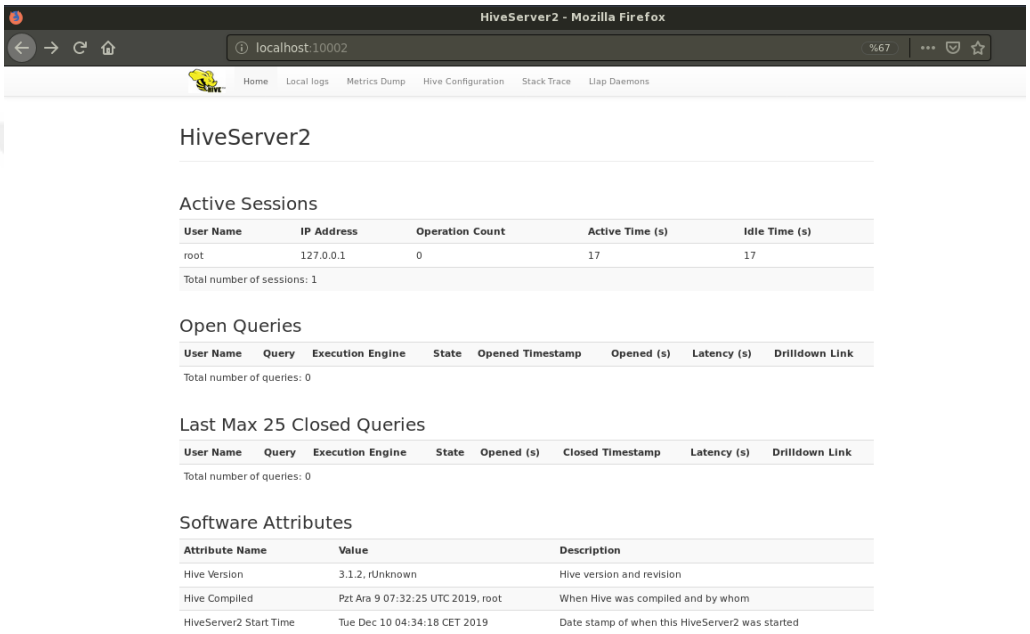


Figure 3.42. Hive Web UI

To setup Hive server with an initial schema, the following scripts were executed:

```
$ cd $HIVE_HOME
$ bin/schematool -dbType derby -initSchema
Initialization script completed
schemaTool completed
```

After this command metastore_db directory and derby.log will create under hive home. By the way we used Derby database for Hive metastore, it comes within Hive. To run HiveServer2:

```
$ $HIVE_HOME/bin/hiveserver2
```

Hive command line interface (Beeline) were run in terminal as follows.

```
$ $HIVE_HOME/bin/beeline -u jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 3.1.2 by Apache Hive
0: jdbc:hive2://localhost:10000>
```

Also we can verify Hive from the web interface as shown in Figure 3.42. We run the basic example to create a table and import data using SQL commands:

```
0: jdbc:hive2://localhost:10000> CREATE TABLE pokes \
 (foo INT, bar STRING);
0: jdbc:hive2://localhost:10000> ALTER TABLE pokes \
 ADD COLUMNS (new_col INT);
0: jdbc:hive2://localhost:10000> describe pokes;
+-----------+------------+----------+
| col_name  | data_type  | comment  |
+-----------+------------+----------+
| foo       | int        |          |
| bar       | string     |          |
| new_col   | int        |          |
+-----------+------------+----------+
0: jdbc:hive2://localhost:10000> LOAD DATA LOCAL INPATH
'./examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
0: jdbc:hive2://localhost:10000> select * from  pokes;
+------------+------------+----------------+
| pokes.foo  | pokes.bar  | pokes.new_col  |
+------------+------------+----------------+
| 238        | val_238    | NULL           |
| 86         | val_86     | NULL           |
```

```
| 311         | val_311     | NULL            |
.............................................
```

3.3.4. Apache Spark Setup

We installed Spark without Hadoop because we already have Hadoop installed before.

So we need to add Hadoop classpath to Spark's profile file.

```
# /etc/profile.d/spark.sh
export SPARK_DIST_CLASSPATH=
$($HADOOP_HOME/bin/hadoop classpath)
```

After that we can start a standalone master server with the following command:

```
$SPARK_HOME/sbin/start-master.sh
```

We can check Spark is running from the web interface at http://localhost:8080. We expect to see the screenshot shown in Figure 3.43. Also clients may connect to the master server via spark://localhost:7077. To run an interactive shell on Spark cluster use the following command:

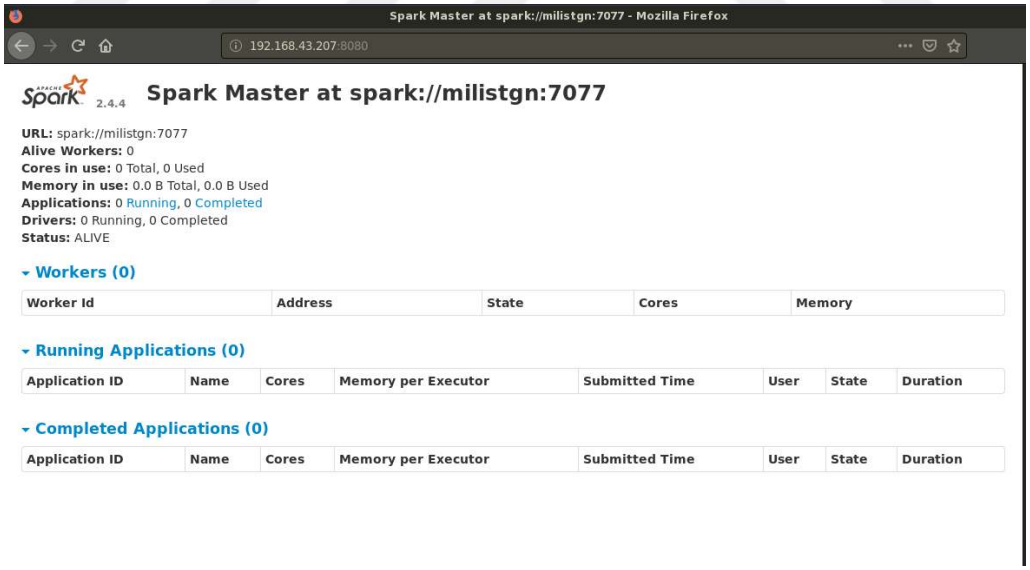


Figure 3.43. Spark Master Web UI

```
$ cd $SPARK_HOME
$ export TERM=xterm-color
$ bin/spark-shell --master spark://milistgn:7077
```

Spark gives the following error.

```
[ERROR] Failed to construct terminal; falling back to
unsupported java.lang.NumberFormatException:
For input string: "0x100"
```

So we adjusted TERM environment variable to fix the above error.

3.3.5. Apache Kafka Setup

With Kafka installation, Zookeeper is also installed due to its delegation as a coordinator between the brokers and consumers. Therefore, Zookeeper is configured first. There is already a sample config file under /usr/mdp/zookeeper/conf . We renamed it to zoo.cfg from zoo_sample.cfg and updated dataDir to /usr/mdp/zookeeper/data. After the configuration, we started the Zookeeper service using the following command:

```
$ $ZOOKEEPER_HOME/bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/mdp/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Zookeeper has CLI that connects to the Zookeeper server and returns the below response.

```
$ $ZOOKEEPER_HOME/bin/zkCli.sh
Connecting to localhost:2181
[zk: localhost:2181(CONNECTED) 0]
```

Next, we started Kafka:

```
$ cd $KAFKA_HOME
$ bin/kafka-server-start.sh config/server.properties
.................
[2019-12-23 02:12:34,124] INFO [KafkaServer id=0]
started (kafka.server.KafkaServer)
```

After starting Kafka broker, pub/sub features are tested. In message queue system, messages are handled via channels. In Kafka these are topics. So a topic is created as follows.

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic topictest
Created topic topictest.
```

Topic list may also be obtained:

```
$ bin/kafka-topics.sh --list --zookeeper localhost:2181
topictest
```

Now, we can send messages with:

```
$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic topictest
>hey
>test
>messages
```

In other terminal, we may consume messages with:

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic topictest --from-beginning
hey
test
messages
```

3.3.6. Apache Druid Setup

After installation of Apache Druid, Zookeper service is disabled as it has already been installed prior. In Druid test, we run "start-micro-quickstart profile" to edit "bin/verify-default-ports" and "conf/supervise/single-server/micro-quickstart.conf" files to use the existing Zookeeper. Then we can start Druid:

```
$ ./bin/start-micro-quickstart
Running command[coordinator-overlord]
Running command[broker]
Running command[router]
```

```
Running command[historical]
Running command[middleManager]
```

Once a single cluster had started, you may navigate to http://localhost:8888 to view through web interface. The Druid router process serves the Druid console as shown in Figure 3.44.

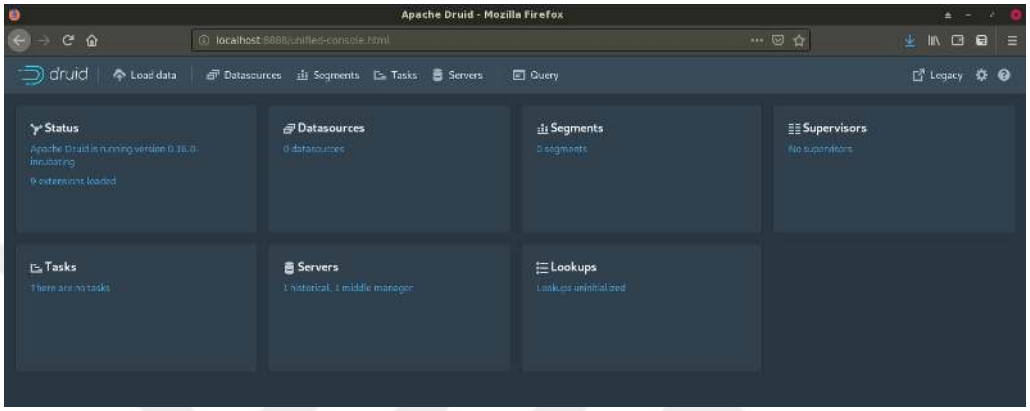


Figure 3.44. Druid Web Console

3.3.7. Cassandra Setup

Cassandra is started with a single node by not editing cassandra.yml file. Only, ANSI local is set.

```
$ LC_ALL=C bin/cassandra -f
```

Next, server is checked by starting cqlsh (CLI for Cassandra) in another terminal:

```
$ $CASSANDRA_HOME/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.5-SNAPSHOT |
CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

3.3.8. Apache Flink Setup

We can start a local cluster of Flink in $FLINK_HOME:

```
$ LC_ALL=C bin/start-cluster.sh
Starting cluster.
```

Flink web console interface is shown in Figure 3.45.

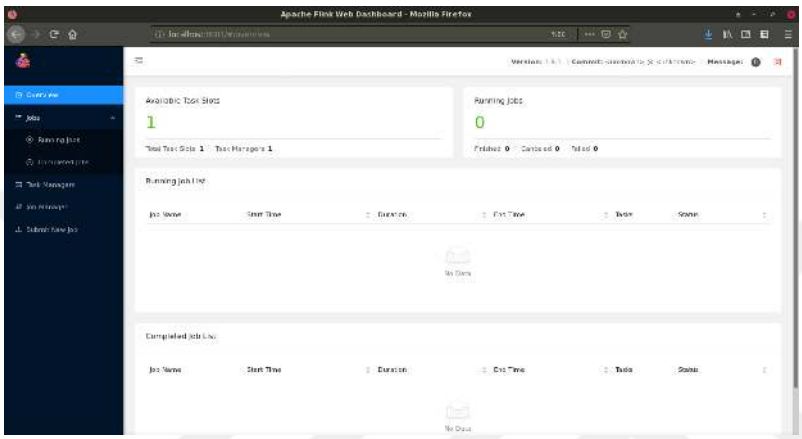


Figure 3.45. Flink Web Console

3.3.9. Luigi Setup

Luigi service is started as workflow and task management tool via:

```
$ luigid --background --port=8082 --logdir=logs
```

Web interface for Luigi is shown in Figure 3.46.

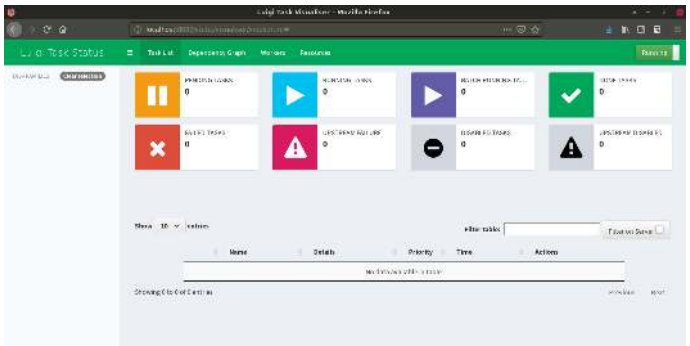


Figure 3.46. Luigi Web Console

4. RESULTS AND DISCUSSION

After all major big data components are installed the setup is verified through tests in 4 steps:

- **Building the packages:** Javac is the main compiler for big data applications that was selected. MILIS Linux comes with Javac from OpenJDK8 already installed. Also during the compilation of packages, tools such as Maven and Gradle download dependencies from global repositories. Gradle, had an issue with the existing version on MILIS. Therefore, packages that use Gradle to compile are not compiled but installed from pre-compiled library at this moment. The directive files of the big data components, as discussed in the Implementation section, were not very difficult to prepare. However, such task would require more effort in Debian, Fedora or Gentoo build system. During the building phase of packages, a dependency repository directory is mounted to isolated compilation environment so that internal dependencies are not downloaded repeatedly. Packages are generally configured to be installed under /usr/mdp. In addition, environment variables for defining home directories are inserted to /etc/profile.d at startup.

- **Configuration:**One of the most challenging phase of big data setup after package installation is the configuration. Because often individual components should be tuned to work most effectively either by themselves or in conjunction with other dependant software. For example, if Hive and Spark needs to be installed together, Hive needed to be configured to support Spark and during installation and setup also Spark needs to know certain setup configuration of Hive in order to ensure proper setup.

  Management tools such as Ambari not only monitors the health of the system during the operation, but also ensures proper installation and management of big data software and modules by checking the version compatibility of packages. It also provides alteration of settings from the web interface. In our study, we edited plain text files of configuration using text editors. However, console-based configuration and setting management has proven to be difficult. This is perhaps one of priority tasks that this proposed work needs to tackle in the future.

- **Testing:**Testing the applications is important to demonstrate the operability of the platform. We have done two kinds of tests; first one is individual component testing and second one is the system wide testing. In second test, compatibility between components are tested and adjustments and configuration are made modified to ensure system integrity.

- **Management:** All of the above processes are setup manually through the commands of the basic linux operating system. Management of versioning and settings are not covered in the current setup. Deployment and management of 7-8 big data modules manually may not be difficult currently, but as the number of applications increase there will be a need for a GUI. Managing of a big data system requirese not only capability in linux but also expertise in Hadoop ecosystem. It is our goal to develop a minimal management panel in a follow up study.

At the end of these steps we got resource consumption information for each module installed. Note that, each value of the metric is obtained under no load, standalone. As seen in Table 4.1. Druid and Cassandra has the highest memory requirement. In terms of disk size required for software setup, none of the modules need significant space. Having Cassandra and Druid using the most memory is typical for databases with OLAP characteristics. Our big data distribution based on MILIS has an image size of 1.9 GB. This is expected to be the lowest among the available for download.

Table 4.1. Size and standalone RAM usage information of components

	Package Size (MB)	Installed Size (MB)	RAM Usage (MB)
Hadoop 3.2.1	272	366	1050
Hive 3.1.2	294	347	600
Zookeeper 3.5.6	6	7	100
Spark 2.4.4	146	175	150
Druid 0.16.0	163	205	2860
Cassandra 3.11.5	29	42	2250
Kafka 2.3.1	55	59	400
Flink 1.9.1	227	286	235
Luigi 2.8.0	4	21	20
Summary	∼1200	∼1500	∼7650

5. CONCLUSION

Big data is increasingly becoming an important area of research and development for businesses and organizations. The reflection of this to the software world is the development of big data components and the construction of platforms hosting these components. Construction based on open source software is the dominant approach for these platforms. Although, these platforms are open source, it does not mean that they are easy to deploy, adapt and maintain. Neither, they are completely free.

As mentioned in the study, big data components work in coordination with other components. Being able to work from input to output in an integrated fashion brings the success of big data platforms. Such integration however requires the right combination of components to be selected, installed together and configured synchronously. This is not an easy task due to configuration complexity. Successful integration of each component into the platform, therefore requires distributed process management experience and programming capabilities.

The purpose of this study is to build a free, light-weight big data platform based on independently developed MILIS Linux. Using the MILIS build-system, compatible and essential components of the Big Data platform are selected, added to the package repository and verified to run. Within the scope of this thesis, the deployment part of this effort and the first step taken to show their interoperability are reported. The purposed platform includes the modules for the input and the processing of the big data. Big Data distribution that we deployed and confirmed comes with a live image that may be downloaded from https://mls.akdeniz.edu.tr/iso

Within the scope of this research, widely used modules for big data platform are researched, deployed, tested and compared in capability and performance. Considering the needs of the big data processing, a minimal set of best performing, least resource hungry and complex modules are identified and became part of the final distribution image. During the thesis; the methods, commands, and scripts used to create the big data platform software stack and distribution image are reported.

The big data distribution developed in this study, is aimed to be the preferred software for a data scientist either local now or cloud in the future.

6. REFERENCES

Taylor, K. and Sakyi 2016. Big Data: Understanding Big Data. *Arxiv* https://arxiv.org/abs/1601.04602

IBM Big Data Hub: https://www.ibmbigdatahub.com/infographic/four-vs-big-data [Last access date: 2019-11-30].

Bridgwater, A. 2013: https://www.computerweekly.com/blog/Open-Source-Insider/Why-Linux-is-the-powerhouse-for-big-data [Last access date: 2019-11-30].

Nuggets, T. 2018: https://www.cbtnuggets.com/blog/certifications/open-source/why-linux-runs-90-percent-of-the-public-cloud-workload [Last access date: 2019-11-30].

Microsoft 2018: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/ [Last access date: 2019-11-30].

Mahmoud, T. 2017: https://www.mindtory.com/an-introduction-to-distributed-file-sys [Last access date: 2019-11-30].

Hadoop, A. 2019 : https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#NameNode+and+DataNodes [Last access date: 2019-11-30].

B3LAB 2016: https://www.b3lab.org/mapreduce-nedir/ [Last access date: 2019-11-30].

Edureka 2018: https://www.edureka.co/blog/hadoop-yarn-tutorial/ [Last access date: 2019-11-30].

Rouse, M. 2018: https://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator [Last access date: 2019-11-30].

White, Tom. 2015: Hadoop:TheDefinitiveGuide,4thEdition2015 [Last access date: 2019-11-30].

TEZ, H. 2019: https://tez.apache.org/ [Last access date: 2019-11-30].

Flink, A. 2019: https://flink.apache.org [Last access date: 2019-11-30].

Shenoy, R. 2014: https://www.infoq.com/articles/apache-tez-saha-murthy/ [Last access date: 2019-11-30].

Cloudera 2019: https://www.cloudera.com/products/open-source/apache-hadoop/apache-tez.html [Last access date: 2019-11-30].

V2 Maestros, LLC. 2016: https://www.udemy.com/user/kumaranponnambalam/ [Last access date: 2019-11-30].

Zookeeper, A. 2015: https://zookeeper.apache.org/doc/r3.5.6/zookeeperOver.html [Last access date: 2019-11-30].

IBM 2019: https://www.ibm.com/analytics/hadoop/zookeeper [Last access date: 2019-11-30].

Guru99 2019: https://www.guru99.com/zookeeper-tutorial.html [Last access date: 2019-11-30].

Intellipaat 2019: https://intellipaat.com/blog/what-is-apache-ambari [Last access date: 2019-11-30].

Cloudera CM 2019: https://docs.cloudera.com/documentation/enterprise/5-7-x/topics/admin_cm_ha_deploy_arch.html [Last access date: 2019-11-30].

Confluent 2019: https://www.confluent.io/what-is-apache-kafka/ [Last access date: 2019-11-30].

StreamSets 2019: https://streamsets.com/documentation/controlhub/latest/onpremhelp/controlhub/UserGuide/GettingStarted/HighAvailability.html [Last access date: 2019-11-30].

Chopping, M. 2017: https://dzone.com/articles/hadoop-distributions-the-story [Last access date: 2019-11-30].

BusinessWire 2018: https://www.businesswire.com/news/home/20181003005869/en/Cloudera-Hortonworks-Announce-Merger-Create-World%E2%80%99s-Leading [Last access date: 2019-11-30].

Flume, A. 2019: https://flume.apache.org [Last access date: 2019-11-30].

Sqoop, A. 2019: https://sqoop.apache.org/ [Last access date: 2019-11-30].

Ilter, H. 2013: http://devveri.com/hadoop/sqoop-veritabani-hadoop-veri-aktarimi [Last access date: 2019-11-30].

Lade, P. 2017: https://medium.com/@prasanth_lade/luigi-all-you-need-to-know [Last access date: 2019-11-30].

PrestoDB 2019: https://prestodb.io/ [Last access date: 2019-11-30].

Cloudera 2019: https://www.cloudera.com/about/news-and-blogs/press-releases/2019-01-03-cloudera-and-hortonworks-complete-planned-merger.html [Last access date: 2019-11-30].

CDH: https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/cdh_intro.html [Last access date: 2019-11-30].

MapR: https://mapr.com/docs/51/MapROverview/c_overview_intro.html [Last access date: 2019-11-30].

AWS 2019: https://aws.amazon.com/tr/data-warehouse/ [Last access date: 2019-11-30].

Reeves, T. 2019: http://www.npvstaffing.com/big-data-on-amazon-web-services/ [Last access date: 2019-11-30].

Lester, R. 2016: https://blogs.msdn.microsoft.com/robinlester/2016/03/09/architecting-a-big-data-project-in-azure/ [Last access date: 2019-11-30].

PivotalHD 2015: https://gigaom.com/2015/02/17/pivotal-open-sources-its-hadoop-and-greenplum-tech-and-then-some/ [Last access date: 2019-11-30].

IBM 2019: https://www.ibm.com/cloud/blog/implementing-big-data-platform-cloud [Last access date: 2019-11-30].

GCP 2019: https://cloud.google.com/docs/overview/ [Last access date: 2019-11-30].

Data-Flair 2019: https://data-flair.training/blogs/hadoop-2-x-vs-hadoop-3-x-comp [Last access date: 2019-11-30].

Hadoop 2019: https://hadoop.apache.org/docs/r3.2.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html [Last access date: 2019-11-30].

Kafka 2019: https://kafka.apache.org/downloads [Last access date: 2019-11-30].

Parker, E. 2019: https://www.xplenty.com/blog/apache-spark-vs-tez-comparison/ [Last access date: 2019-11-30].

Peppoloni, L. 2018: https://towardsdatascience.com/data-pipelines-luigi-airflow [Last access date: 2019-11-30].

Airflow, A. 2019: https://airflow.apache.org/ [Last access date: 2019-11-30].

Luigi 2019: luigi.readthedocs.org [Last access date: 2019-11-30].

NIFI, A. 2019: http://nifi.apache.org/ [Last access date: 2019-11-30].

Guru99 2019: https://www.guru99.com/apache-nifi-tutorial.html [Last access date: 2019-11-30].

Impala, A. 2019: https://impala.apache.org/ [Last access date: 2019-11-30].

Kudu, A. 2019: https://kudu.apache.org/ [Last access date: 2019-11-30].

Ranger, A. 2019: https://ranger.apache.org/ [Last access date: 2019-11-30].

Druid, A. 2019: https://druid.apache.org/ [Last access date: 2019-11-30].

Mahout, A. 2019: http://mahout.apache.org/ [Last access date: 2019-11-30].

TechTarget 2019: https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL [Last access date: 2019-11-30].

Mindmajix 2018: https://mindmajix.com/cloudera-vs-hortonworks [Last access date: 2019-11-30].

Cloudera 2019: https://blog.cloudera.com/apache-hive-druid-part-1-3/ [Last access date: 2019-11-30].

Klein, E. 2019: https://logz.io/blog/nosql-database-comparison/ [Last access date: 2019-11-30].

Bradley, J., Meng, X., Lee D. 2016: https://www.infoworld.com/article/3031690/why-you-should-use-spark-for-machine-learning.html [Last access date: 2019-11-30].

Die.net 2019: https://linux.die.net/man/8/polkit [Last access date: 2019-11-30].

CURRICULUM VITAE

Muhammed Numan İNCE

muhammednince@gmail.com

EDUCATION

Master of Science 2018-2020	Akdeniz University Institute of Naturel Sciences, Department of Computer Engineering, Antalya
Bachelor of Science 2004-2008	Ege University Engineering Faculty, Department of Computer Engineering, Izmir

WORK EXPERIENCE

Computer Engineering 2008-2017	Incorporated Company IT Staff, Izmir

PUBLICATIONS

Papers delivered in International Conferences and Printed as a Proceedings

1- Gunay M., Ince M.N., Cetinkaya A., (2019). Apache Hive Performance Improvement Techniques for Relational Data. ?, 30(4), 1013-1020. Doi: 10.2298/FIL1604013C.