

UNIVERSITY OF BUEA



REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,
Buea, South West Region
CAMEROON
Tel: (237) 3332 21 34/3332 26 90
Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CEF 450: CLOUD COMPUTING AND SOA

HADOOP AS A SERVICE (HAAS): REVOLUTIONIZING BIG DATA PROCESSING.

BY:

GROUP 2(HADOOP)

SUPERVISOR:

DJEULA INES, PHD

University of Buea

2nd Semester 2023/2024 Academic Year

List of Participants

SN	Name	Matricule	Speciality
1	KOUE TEPE KENNETH	FE21A220	SOFTWARE
2	QUINUEL TABOT NDIP-AGBOR	FE21A300	SOFTWARE
3	NSIELA SUHBISI DENZEL	FE21A286	SOFTWARE
4	REOUTADE ROLAND	FE21A301	SOFTWARE
5	NGOBA STEVE JONES NTONG	FE21A259	SOFTWARE
6	NKEMZI FOLEFACK GIL	FE21A276	SOFTWARE
7	NKWENKAM JENNIFER	FE21A279	SOFTWARE
8	NTUI RAOUL NTUI NJOCK	FE21A288	SOFTWARE
9	NYANDO ONONGWENE	FE21A290	SOFTWARE
10	MBISHU FABRICE YENVEN	FE21A232	SOFTWARE
11	NKWO BRAINIE NGONDA	FE21A282	SOFTWARE
12	MBACHAM LOANA NING	FE21A228	SOFTWARE
13	OJONG-ENYANG OYERE	FE21A297	SOFTWARE
14	MBISHU FABRICE YENVEN	FE21A232	SOFTWARE
15	MBACHAM LOANA NING	FE21A228	SOFTWARE
16	MESSI II INNOCENT R.	FE21A236	SOFTWARE
17	NJIDA SALIFU	FE21A272	SOFTWARE
18	NYUYSEVER BORIS DINNYUY	FE21A295	SOFTWARE
19	NGUEDIA JEATSA JOYCE GRACE	FE21A263	SOFTWARE
21	KIMBI CYRIL BONGNYU	FE21A216	SOFTWARE
22	NYENTY PAUL EGBE	FE21A292	SOFTWARE
23	LIMA CHARLES KEHBUMA	FE21A225	SOFTWARE
24	NFOR WILLY LINWE	FE21A254	SOFTWARE
25	NZEMTEJUH SYLVANUS	FE21A296	SOFTWARE
26	NKWETI MANGEM ANGCELIA	FE21A280	SOFTWARE
27	METAGNE KAMGA MAIVA	FE21A237	SOFTWARE
28	MEWOABI NGUEFACK DORE	FE21A239	SOFTWARE
29	LONCHI JORDAN	FE21A226	SOFTWARE
30	NSEM CONFIDENT NJOCK	FE21A285	SOFTWARE
31	NKEMTCHOU PIANKE OLIVIER	FE21A275	SOFTWARE

32	MOKFEMBAM FABRICE KONGNYUY	FE21A240	SOFTWARE
33	NDONG HENRY NDANG	FE21A248	SOFTWARE
34	NOUPOUWO DONGMO STEPHANE MERCI	FE21A283	SOFTWARE
35	POKAM NGOUFFO TANEKOU	FE21A299	SOFTWARE
36	NGUEPI GNETEDEM PATERSON	FE21A264	SOFTWARE
37	NEGUE KWAHAM MAEL GRACE	FE21A252	SOFTWARE
38	NDIFON LEMUEL ASHU-MBI	FE21A247	SOFTWARE
39	NFOUA EUGENE MGBA	FE21A257	SOFTWARE
40	NGULEFAC JERRY MBUOH .	FE21A265	SOFTWARE
41	NGAGUEN NDJOMOU LOICE VANELLE	FE21A258	SOFTWARE
42	NGWASIRI RYAN TANIFORM ONGA	FE21A266	SOFTWARE
43	MBUNGAI GEORGE BERINYUY	FE21A234	SOFTWARE
44	NKENGBEZA DERICK	FE21A277	SOFTWARE
45	NEBA PRINCEWILL AMBE	FE21A251	SOFTWARE
46	MONDOA ROBERT NASOA NDIVE	FE21A241	SOFTWARE
47	LANGHEH MOHAMMED YIENEH	FE21A223	SOFTWARE
48	Mofako Beltus Edube	FE20A065	SOFTWARE
49	NYOCHEMBENG ENZO NKENGAFACK	FE21A293	SOFTWARE
50	NGONCHI RAMATOU YOLAND	FE21A260	SOFTWARE

ABSTRACT

In today's data-driven world, managing and analysing vast amounts of data efficiently has become a paramount challenge for organizations across various industries. Apache Hadoop, a robust and scalable open-source framework, emerges as a solution to tackle this challenge effectively. As a Platform as a Service (PaaS) offering, Hadoop provides a distributed computing environment that enables the storage, processing, and analysis of large datasets across clusters of commodity hardware.

This presentation aims to provide a comprehensive overview of Apache Hadoop as a PaaS solution, covering its environment, installation and configuration process, and the deployment of a simple application. We will explore the fundamental concepts behind Hadoop's architecture, its core components, and the distributed file system that underpins its operations. Additionally, we will delve into the step-by-step process of setting up a Hadoop cluster, configuring its various services, and ensuring seamless integration with existing infrastructure.

Furthermore, we will demonstrate the deployment of a simple application on the Hadoop cluster, showcasing the ease with which developers can leverage its powerful capabilities to process and analyze data at scale. Through this presentation, attendees will gain valuable insights into the potential of Hadoop as a PaaS solution, its key features, and its role in driving innovation and unlocking actionable insights from big data.

Table Of Content

List of Participants	2
ABSTRACT.....	4
INTRODUCTION	6
OBJECTIVES	6
SCOPE	6
WHAT IS HADOOP?.....	7
HADOOP ARCHITECTURE AND KEY COMPOENENT	8
Hadoop Architecture	8
What is Hadoop architecture?.....	8
What is Hadoop architecture used for?.....	9
Who uses Hadoop architecture?	10
Pros and cons of Hadoop architecture	10
pros.....	10
Cons	10
Components of Hadoop.....	11
Main compoenets.....	11
Hadoop HDFS - Hadoop Distributed File System (HDFS)	11
NameNode and DataNodes	14
Hadoop MapReduce	16
YARN (Yet Another Resource Negotiator)	21
Other MapReduce Components	22
Why Hadoop Matters.....	24
SETTING UP HADOOP ENVIRONMENT	25
Prerequisites for Linux	25
Procedure to setup the environment.....	25
DEPLOYING A SIMPLE APPLICATION ON THE HADOOP CLUSTER (java word count application)	33
Source code(a clean version will be attached during submission).....	33
Running The Application on Hadoop	33
Result And Discussion	35
CONCLUSION.....	36
REFERENCE.....	37

INTRODUCTION

In today's data-centric landscape, Apache Hadoop emerges as a robust and scalable open-source framework, addressing the challenge of efficiently managing and analysing vast datasets across various industries. As a Platform as a Service (PaaS) offering, Hadoop facilitates distributed computing, enabling storage, processing, and analysis of large datasets across commodity hardware clusters.

This presentation provides a comprehensive overview of Apache Hadoop as a PaaS solution, covering its environment, installation, and configuration process. Exploring fundamental concepts of Hadoop's architecture and core components, we navigate through the setup of a Hadoop cluster, configuring services, and integrating with existing infrastructure.

Moreover, we showcase the deployment of a simple application on the Hadoop cluster, illustrating developers' ability to harness its capabilities for scalable data processing and analysis. Attendees will gain insights into Hadoop's potential as a PaaS solution, its key features, and its role in driving innovation and extracting actionable insights from big data.

OBJECTIVES

By the end of this report, readers will grasp Apache Hadoop's essence as a Platform as a Service (PaaS) solution. They'll comprehend its architecture, installation, configuration, and application deployment. Armed with this understanding, they'll harness Hadoop's power to store, process, and analyze vast datasets effectively. Equipped with practical insights, they'll make informed decisions and foster innovation within their organizations.

SCOPE

This report explores Apache Hadoop as a Platform as a Service (PaaS) solution, covering its core components, installation process, and application deployment. Readers will gain insights into Hadoop's capabilities for managing and analysing large datasets efficiently. We use **Ubuntu** as a case study and we will not touch the other operating systems. Also, we will be simulating a **single cluster** (not multiple clusters) – due to **hardware limitations** on a **local machine and not on a cloud platform**

WHAT IS HADOOP?

Hadoop is a powerful open-source framework specifically designed to address the challenges of storing and processing large-scale data sets. At its core, Hadoop enables distributed storage and processing of massive amounts of data across a cluster of commodity hardware. This distributed architecture allows organizations to harness the combined computational power of multiple machines, resulting in improved scalability, fault tolerance, and performance.

The origins of Hadoop can be traced back to research papers published by Google on their MapReduce programming model and Google File System (GFS). Inspired by these papers, the creators of Hadoop sought to develop an open-source framework that could replicate the capabilities of Google's infrastructure for handling large-scale data processing tasks.

One of the key features of Hadoop is its scalability. Hadoop scales horizontally, meaning that additional machines can be seamlessly added to the cluster as the volume of data grows. This scalability enables organizations to handle increasing amounts of data without significant infrastructure investments.

Hadoop has emerged as the go-to solution for handling big data challenges across various industries. Its ability to efficiently store, process, and analyse massive datasets has made it an essential tool for organizations looking to extract valuable insights from their data. Whether it's processing log files, analysing social media data, or running complex machine learning algorithms, Hadoop provides a flexible and cost-effective platform for tackling diverse big data use cases.

HADOOP ARCHITECTURE AND KEY COMPOENENT

Hadoop Architecture

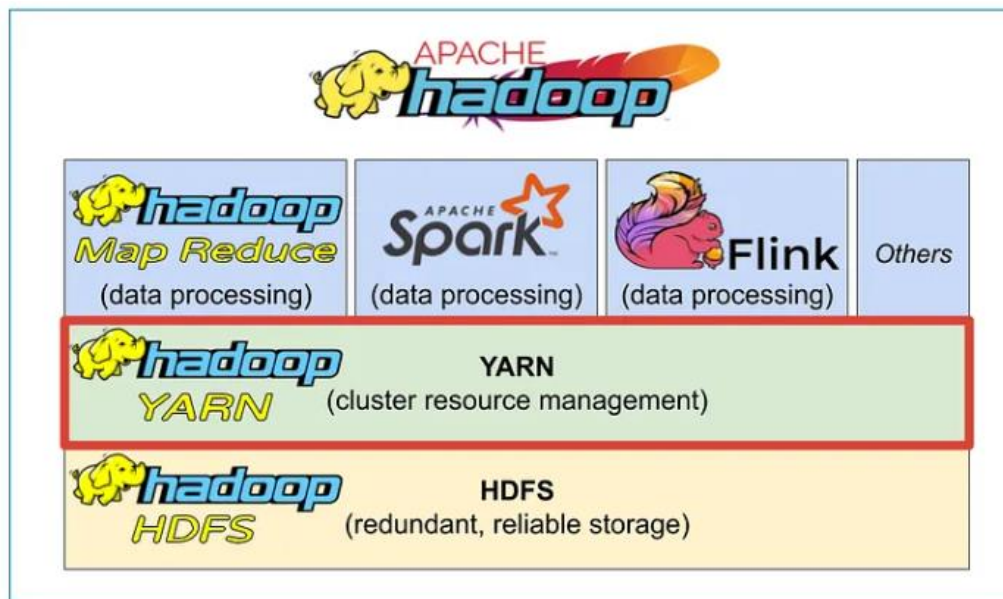


Figure 1: hadoop Architecture Illustration

Hadoop architecture operates as a framework built of multiple computers, known as clusters. It allows for the storage and analysis of large data sets.

Its unique architecture makes it key for processing immense, unstructured data sets quickly and efficiently, all while protecting your data from loss or corruption due to the design of the computer clusters. Additionally, you can combine the abilities of Hadoop with machine learning and AI to uncover patterns within large data sets while also formulating predictions.

What is Hadoop architecture?

Hadoop architecture comprises four modules: the Hadoop Distributed File System (HDFS), Yet Another Resource Negotiator (YARN), Hadoop Common, and MapReduce. These components work across a cluster of computers set up on the same network. Giant sets of data break up across the cluster for parallel processing. HDFS takes these data sets and spreads them across the cluster, duplicating the sets at least twice. This way, if one of your data sets becomes corrupted, it's not lost. Instead, the

file system replicates the remaining copies of the set so that it's always available in multiple places.

Once HDFS spreads the data set across the cluster, MapReduce processes and converts it, combining individual subsets into a more manageable data set. Finally, YARN monitors the cluster and assigns tasks as needed.

Hadoop architecture follows a leader/follower structure. One computer in the cluster is the NameNode, which directs processing, and the follower computers do the actual processing.

What is Hadoop architecture used for?

Hadoop architecture's main function is to process immense sets of big data for use in data analytics and data management. It's unique in that its architecture protects data from corruption or loss by replicating the sets across multiple computer clusters. It's capable of processing different file types at once. These clusters allow your business to process giant data sets quickly and efficiently.

Some other uses of Hadoop architecture include:

- **Data lakes:** Hadoop's ability to store data without needing to pre-process means you can partner this technology with a data lake, which stores huge amounts of unstructured data.
- **Data storage:** Hadoop allows you to store all different types of files, which allows your company to process data in ways that will support your business.
- **Big data analytics:** Hadoop offers robust data analytics through its efficient processing of immense data sets and parallel processing.
- **AI and machine learning:** Data professionals use the data processed by Hadoop to support machine learning tasks.
- **Risk management:** Industries with financial risks, such as banks or insurance companies, often use Hadoop as a risk management tool.

Who uses Hadoop architecture?

If you're a data professional, you might use Hadoop architecture to process immense data sets that would otherwise be impossible to parse. You can use Hadoop for its storage, processing, and big data analytics tools. Hadoop architecture has applications across various industries, including security, finance, health care, and retail.

Pros and cons of Hadoop architecture

pros

One benefit of Hadoop is the fact that it's incredibly scalable. You can easily increase the amount of clusters in the ecosystem by adding new computers without the need for expensive infrastructure. Hadoop's architecture is also designed to detect and address any failures without losing your data. For example, if a node—a computer server—within the cluster fails, the data on that node has not been lost because that same data has been stored on another node within the cluster. The analysis of your data can continue without interruption. Hadoop's other key benefit is that its ability to handle different types of structured and unstructured data makes it very flexible, simplifying storage and analysis processes.

Cons

Some cons of Hadoop architecture include the fact that Hadoop struggles to process lots of small files versus several immense ones. If your business tends to generate small data files, Hadoop might not be your best option. In addition, Hadoop's design is mainly for batch processing, and it cannot do any real-time, data-stream processing.

Components of Hadoop

Main components

Hadoop is a framework that uses distributed storage and parallel processing to store and manage Big Data. It is the most commonly used software to handle Big Data. There are three components of Hadoop.

1. Hadoop HDFS - Hadoop Distributed File System (HDFS) is the storage unit of Hadoop.
2. Hadoop MapReduce - Hadoop MapReduce is the processing unit of Hadoop.
3. Hadoop YARN - Hadoop YARN is a resource management unit of Hadoop.

Hadoop HDFS - Hadoop Distributed File System (HDFS)

Before head over to learn about the HDFS(Hadoop Distributed File System), we should know what actually the file system is. The file system is a kind of Data structure or method which we use in an operating system to manage file on disk space. This means it allows the user to keep maintain and retrieve data from the local disk.

An example of the windows file system is NTFS(New Technology File System) and FAT32(File Allocation Table 32). FAT32 is used in some older versions of windows but can be utilized on all versions of *windows xp*. Similarly like windows, we have ext3, ext4 kind of file system for Linux OS.

What is DFS?

DFS stands for the distributed file system, it is a concept of storing the file in multiple nodes in a distributed manner. DFS actually provides the Abstraction for a single large system whose storage is equal to the sum of storage of other nodes in a cluster.

Let's understand this with an example. Suppose you have a DFS comprises of 4 different machines each of size 10TB in that case you can store let say 30TB across this DFS as it provides you a combined Machine of size 40TB. The 30TB data is distributed among these Nodes in form of Blocks. Figure 1 below illustrate a distributed file system

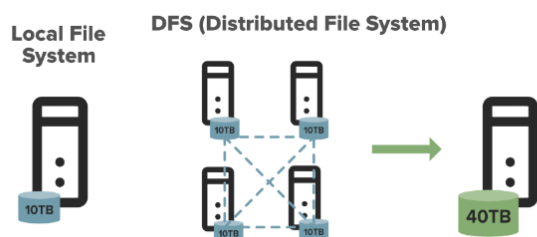


Figure 2: Distributed File system Illustration

Why We Need DFS?

You might be thinking that we can store a file of size 30TB in a single system then why we need this DFS. This is because the disk capacity of a system can only increase up to an extent. If somehow you manage the data on a single system then you'll face the processing problem, processing large datasets on a single machine is not efficient.

Let's understand this with an example. Suppose you have a file of size 40TB to process. On a single machine, it will take suppose 4hrs to process it completely but what if you use a DFS(Distributed File System). In that case, as you can see in the below image the File of size 40TB is distributed among the 4 nodes in a cluster each node stores the 10TB of file. As all these nodes are working simultaneously it will take the only 1 Hour to completely process it which is Fastest, that is why we need DFS.

Local File System Processing:

Distributed File System Processing:



Figure 3: Comparing Local file system with distributed file system

Assumptions and Goals

- **Hardware Failure**

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

- **Streaming Data Access**

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for

applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

- **Large Data Sets**

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

- **Simple Coherency Model**

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed except for appends and truncates. Appending the content to the end of the files is supported but cannot be updated at arbitrary point. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model.

- **“Moving Computation is Cheaper than Moving Data”**

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

- **Portability Across Heterogeneous Hardware and Software Platforms**

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

HDFS Architecture

HDFS is an Open source component of the Apache Software Foundation that manages data. HDFS has scalability, availability, and replication as key features. Name nodes, secondary name nodes, data nodes, checkpoint nodes, backup nodes, and blocks all make up the architecture of HDFS. HDFS is fault-tolerant and is replicated. Files are distributed across the cluster systems using the Name node and Data Nodes. The

primary difference between Hadoop and Apache **HBase** is that Apache HBase is a non-relational database and Apache Hadoop is a non-relational data store. Data is stored in a distributed manner in HDFS. There are two components of HDFS - **name node** and **data node**. While there is only one name node, there can be multiple data nodes.

HDFS is specially designed for storing huge datasets in commodity hardware. An enterprise version of a server costs roughly \$10,000 per terabyte for the full processor. In case you need to buy 100 of these enterprise version servers, it will go up to a million dollars.

Hadoop enables you to use commodity machines as your data nodes. This way, you don't have to spend millions of dollars just on your data nodes. However, the name node is always an enterprise server.

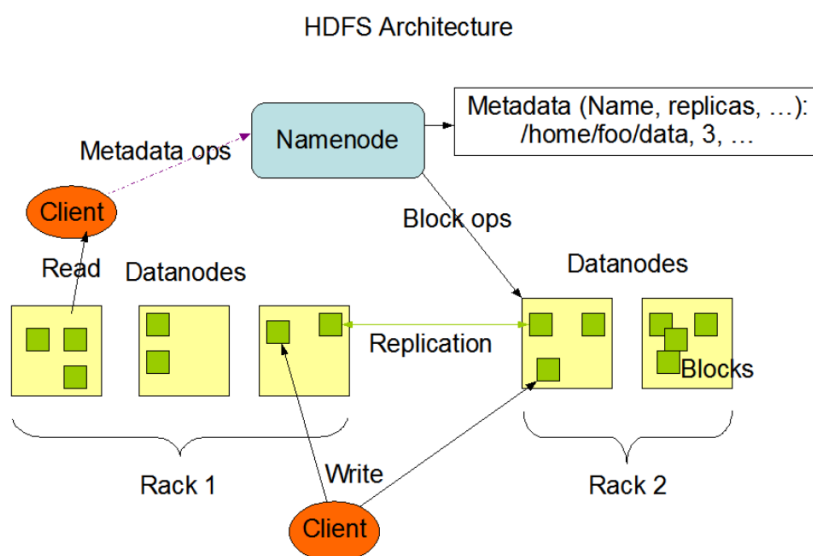


Figure 4: HDFS Architecture

NameNode and DataNodes

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

running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

NameNode

All the blocks on DataNodes are handled by NameNode, which is known as the master node. It performs the following functions:

1. Monitor and control all the DataNodes instances.
2. Permits the user to access a file.
3. Stores all of the block records on a DataNode instance.
4. EditLogs are committed to disk after every write operation to Name Node's data storage. The data is then replicated to all the other data nodes, including Data Node and Backup Data Node. In the event of a system failure, EditLogs can be manually recovered by Data Node.
5. All of the DataNodes' blocks must be alive in order for all of the blocks to be removed from the data nodes.
6. Therefore, every UpdateNode in a cluster is aware of every DataNode in the cluster, but only one of them is actively managing communication with all the DataNodes. Since every DataNode runs their own software, they are completely independent. Therefore, if a DataNode fails, the DataNode will be replaced by another DataNode.

DataNode

Every slave machine that contains data organises a DataNode. DataNode stores data in ext3 or ext4 file format on DataNodes. DataNodes do the following,

- DataNodes store every data.
- It handles all of the requested operations on files, such as reading file content and creating new data, as described above.
- All the instructions are followed, including scrubbing data on DataNodes, establishing partnerships, and so on.

Hadoop MapReduce

The Problem: How to Analyze Massive Datasets

Imagine you have terabytes of website logs tracking every single visitor interaction, and from here, you want to filter out some information, like which pages are most popular or where visitors drop off in your purchase funnel, etc.

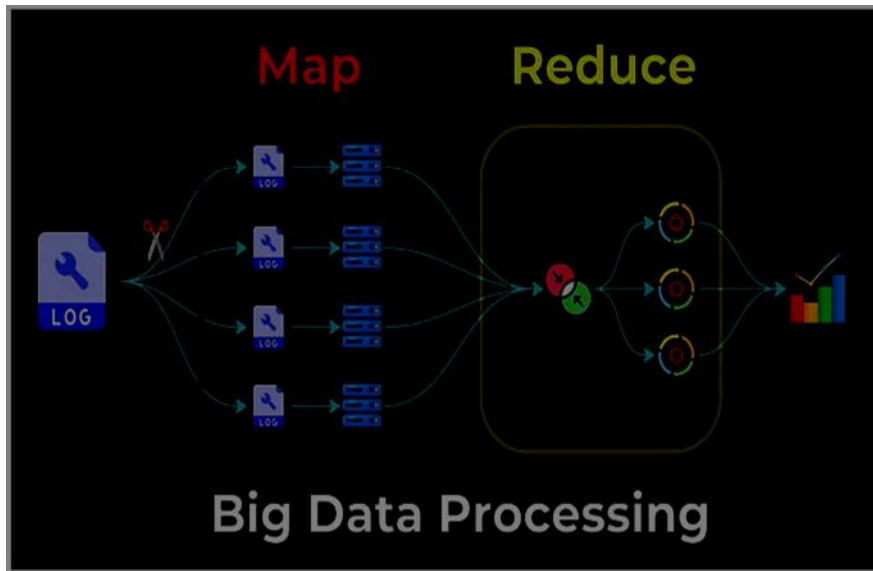


Figure 5: Big data Processing with MapReduce

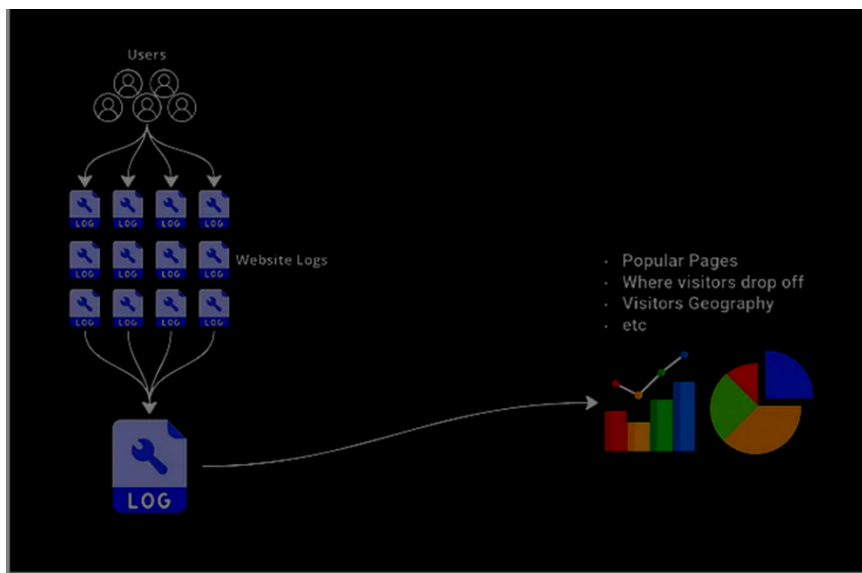


Figure 6: Big data Processing with Traditional tools

Traditional tools and databases are simply not designed for datasets of this scale. That's where MapReduce comes in.

What is MapReduce?

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

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

How MapReduce Handles Big Data?

MapReduce operates in two primary phases — The map phase and the reduce phase.

Map Phase

In the Map Phase, we first split these huge logs into smaller and manageable chunks. These chunks then get sent to different worker computers in a cluster.

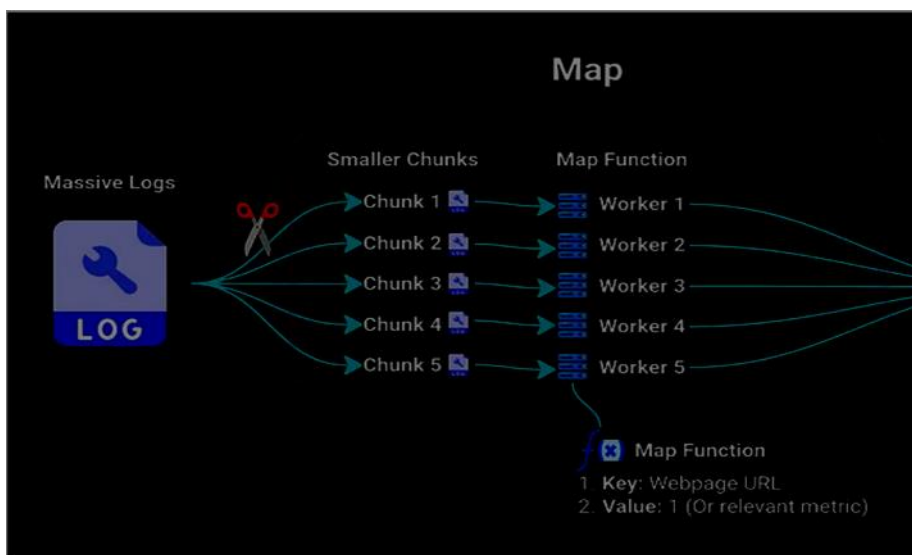


Figure 7: Map Phase of MapReduce Processing

Think of each worker as a separate server that handles its assigned chunk. It has a **Map Function** that extracts the key information: in our case, it will map the **keys**, which are the specific webpage visited, to the **values**, which, if we are counting visits, can be the number of visits to that page.

Reduce Phase

Then, we enter the reduce phase, where all the key-value pairs generated by the map phase are sorted and grouped by webpage ('key').

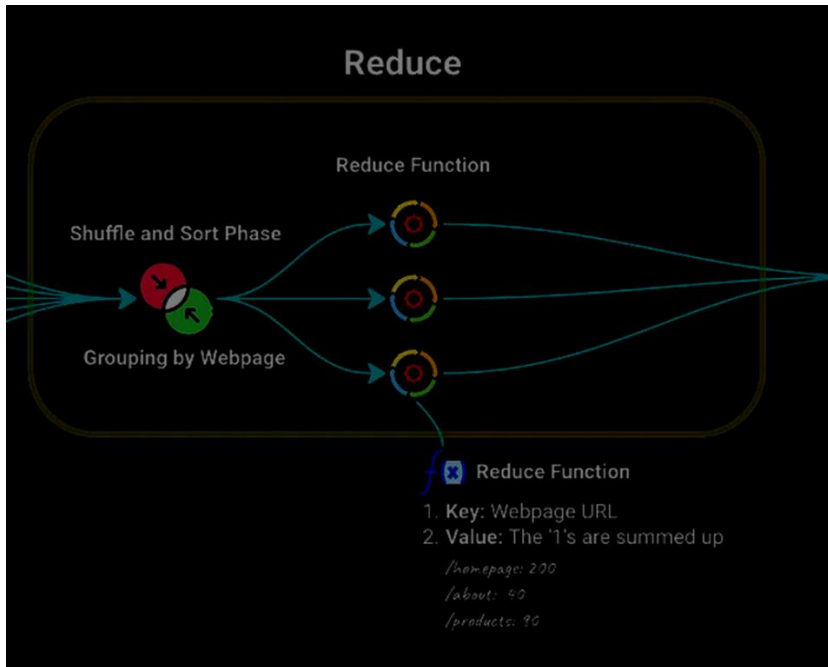


Figure 8: Reduce Phase of MapReduce Processing.

We forward those to the **Reduce Function**. For each unique webpage, it adds up the '1' values to find the total visits. It can also tackle more complex questions, such as average time spent, visitor demographics, etc.

And now, using this information, we can visualize it via charts and other visuals on the screen.

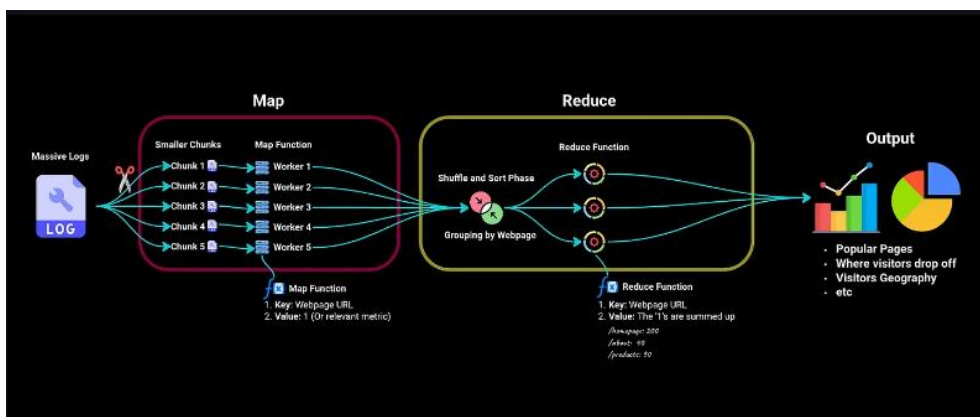


Figure 9: complete MapReduce Process

Benefits of MapReduce for Log Analysis

We get a couple of benefits when processing data with MapReduce:

- **Parallel Power:** Distributing the work makes processing much faster than a single computer could manage.
- **Scalability:** Got even more log data? Just add more computers to the cluster and MapReduce can keep up.
- **Fault Tolerance:** If a computer fails during a job, MapReduce automatically reassigns its work to other computers in the network. This ensures that all the tasks are completed successfully without interruption.

Batch vs. Stream Processing

To understand why MapReduce is so unique, let's quickly touch on batch versus stream processing:

Batch Processing

Batch Processing deals with data in large chunks that have already been collected. For example, if you search for a word in Google Docs or Microsoft Word in a large file, the data is available upfront, so it can be processed immediately.

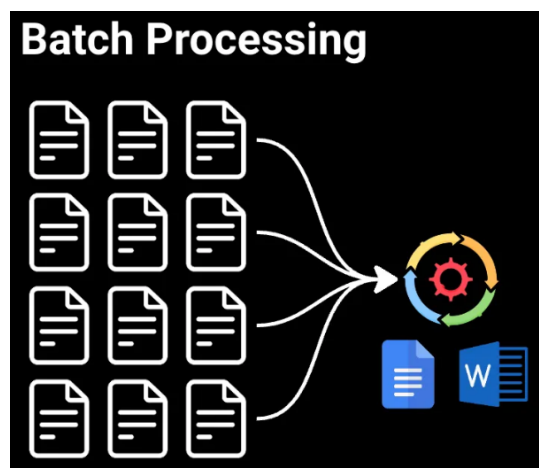


Figure 10: Mapreduce Batch Processing

This is useful for large datasets where immediate results aren't essential, such as when generating monthly sales reports, analyzing customer purchase history, or training machine learning models on data.

Streaming Processing

Stream Processing handles data as it arrives in a continuous flow. For example, when watching a YouTube video, you hit ‘play’, and it starts almost immediately. That’s because tiny pieces of video are sent to your computer in a continuous flow, letting you watch while the rest of the video is still being transmitted.

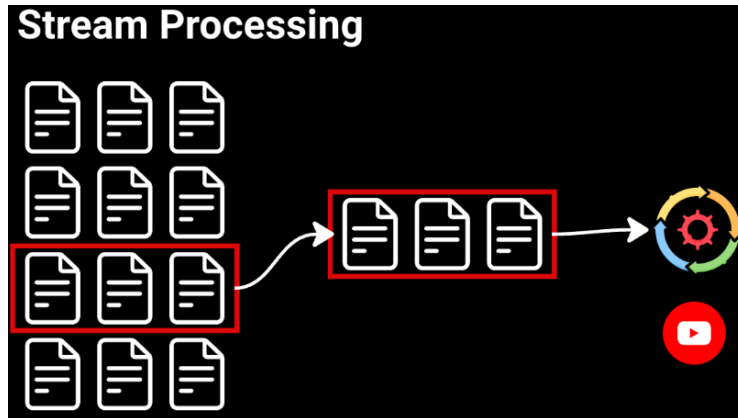


Figure 11: Stream processing illustration

What is the processing method used by MapReduce?

MapReduce is a batch-processing model because it operates on data that is already stored, not on a live continuous stream of incoming data. Input data needs to be divided and distributed before the Map phase of MapReduce even begins.

As you can imagine, Batch processing is slower than Streaming due to the accumulation of data before processing. But it’s generally simpler to set up and manage, while Stream Processing can be more complex due to the constant flow of data and the potential for errors or inconsistencies.

MapReduce Limitations and Modern Alternatives

While MapReduce was revolutionary, it has limitations in terms of speed and flexibility for iterative and complex data processing tasks. This is where tools like Apache Spark come in.

YARN (Yet Another Resource Negotiator)

YARN stands for “**Yet Another Resource Negotiator**“. It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0. YARN was described as a “*Redesigned Resource Manager*” at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing.

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

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

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

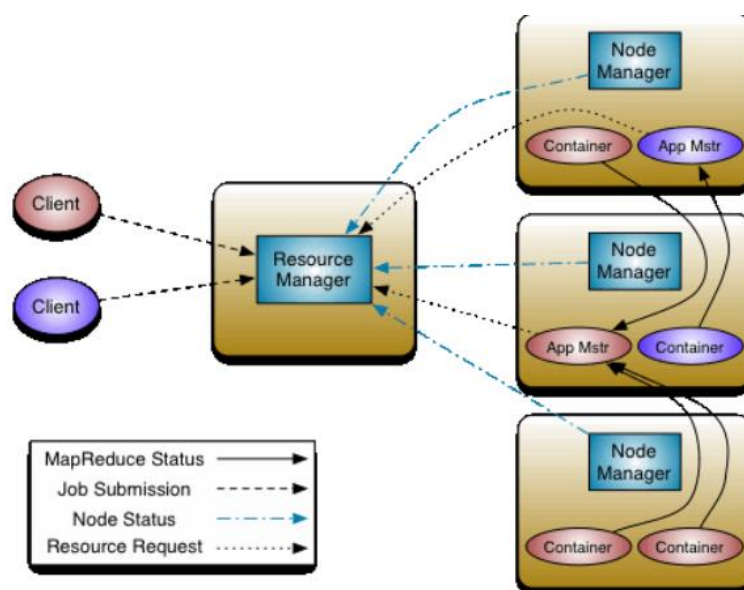


Figure 12: Yarn Architecture

The ResourceManager has two main components: Scheduler and ApplicationsManager.

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

The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the CapacityScheduler and the FairScheduler would be some examples of plug-ins.

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

Other MapReduce Components

Apache Spark

Spark leverages in-memory processing, meaning it keeps data in RAM for very fast calculations compared to MapReduce's reliance on disk storage. It handles a wider range of tasks, including SQL queries, machine learning, and real-time data processing (streaming).

➤ **Integration with Hadoop:**

- Spark can read data from HDFS (Hadoop Distributed File System) and process it in-memory.
- It leverages Hadoop's YARN for resource management and cluster coordination.

➤ **Use Cases with Hadoop:**

- Spark complements Hadoop by providing faster data processing and more expressive APIs.
- Ideal for ETL (Extract, Transform, Load) tasks, machine learning, and interactive analytics.
- Spark SQL integrates seamlessly with Hadoop Hive for querying structured data.

Apache Flink

Apache Flink is another powerful framework used for real-time data processing (stream processing). It offers similar capabilities to Spark Streaming, allowing for immediate analysis of data as it arrives. This is a specialized tool for scenarios requiring real-time data analysis, often used alongside Spark for a complete big data processing toolkit.

➤ **Integration with Hadoop:**

- Flink can read data from HDFS and process real-time streams.
- It can run on YARN alongside Hadoop.

➤ **Use Cases with Hadoop:**

- Flink is suitable for event-driven applications, stream processing, and complex event processing.
- It provides low-latency processing and exactly-once semantics.
- Use Flink for real-time analytics on Hadoop data.

Apache Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop, designed to facilitate querying and analysis of large datasets stored in Hadoop's HDFS. It provides a SQL-like query language called HiveQL, which allows users to write queries to extract insights from data.

➤ *Integration with Hadoop:*

- Hive can seamlessly integrate with Hadoop, leveraging its distributed storage and processing capabilities.
- It can read data from HDFS, process it using MapReduce or Tez, and store the results back to HDFS or other supported storage systems.

➤ *Use Cases with Hadoop:*

- Hive is ideal for data warehousing, ad-hoc querying, and analysis of structured and semi-structured data.
- It is commonly used for data exploration, reporting, and business intelligence applications.
- Hive's ability to process SQL-like queries makes it accessible to users familiar with SQL, enabling them to perform analytics without requiring extensive programming knowledge.

Apache HBase.

Apache HBase is a distributed, scalable, and consistent NoSQL database built on top of Hadoop's HDFS. It provides real-time read/write access to large datasets, making it suitable for applications that require low-latency data access and strong consistency guarantees.

➤ *Integration with Hadoop:*

- HBase is tightly integrated with Hadoop, running on the same cluster infrastructure and sharing the underlying HDFS for storage.
- It can be deployed alongside other Hadoop ecosystem components, such as HDFS, MapReduce, YARN, and Hive.

➤ *Use Cases with Hadoop:*

- HBase is well-suited for use cases requiring random, real-time access to large volumes of structured data.
- It is commonly used for applications like sensor data storage and analysis, time-series data storage, and online transaction processing (OLTP).
- HBase's ability to handle high-throughput, low-latency workloads makes it a preferred choice for applications demanding real-time data processing and analysis.

WHY HADOOP MATTERS

- **Scalability:** Hadoop scales horizontally. Need more power? Add more machines to the cluster.
- **Cost-Effective:** It runs on commodity hardware, saving you big bucks compared to proprietary solutions.
- **Community Love:** Hadoop is open-source, which means a global community of developers constantly improves it.
- **Versatility:** It's not just batch processing; Hadoop handles real-time data too.

SETTING UP HADOOP ENVIRONMENT

System Requirements

Before diving into the installation and configuration, ensure that your system meets the following requirements:

➤ **Hardware Specifications:**

- **RAM:** Minimum 8GB (16GB or more recommended).
- **CPU:** Multi-core processors (quad-core or higher).
- **Storage:** Sufficient disk space for data storage (HDFS).

➤ **Supported Operating Systems:**

- Hadoop primarily runs on Linux distributions (Ubuntu, CentOS, etc.).
- While Windows is also supported, Linux is recommended for production clusters due to better stability and performance.

Prerequisites for Linux

As linux is our simulation environment, we need the following requirements to be met:

- Java must be installed. Recommended Java versions are described at HadoopJavaVersions(<https://cwiki.apache.org/confluence/display/HA+DOOP/Hadoop+Java+Versions>)
- ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons if the optional start and stop scripts are to be used. Additionally, it is recommended that pdsh also be installed for better ssh resource management.

Procedure to setup the environment

- update the package index of available packages

```
: $ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://ppa.launchpad.net/maarten-fonville/android-studio/ubuntu focal InRelease [17.6 kB]
Get:3 http://ppa.launchpad.net/maarten-fonville/android-studio/ubuntu focal/main amd64 Packages [2,568 B]
Hit:4 http://cn.archive.ubuntu.com/ubuntu focal InRelease
Get:5 http://ppa.launchpad.net/maarten-fonville/android-studio/ubuntu focal/main Translation-en [340 B]
Get:6 http://cn.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main i386 Packages [737 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [2,856 kB]
Hit:9 http://cn.archive.ubuntu.com/ubuntu focal-backports InRelease
Get:10 http://cn.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [961 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [432 kB]
```

- install java runtime environment(JRE) and java compiler(Javac)
 - To install java compiler :

```
kenneth@kenneth-Latitude-E5440:~$ sudo apt install default-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  chromium-codecs-ffmpeg-extra gir1.2-goa-1.0 gstreamer1.0-vaapi
  libfwupdplugin1 libgstreamer-plugins-bad1.0-0 libva-wayland2 libxmb1
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  default-jdk-headless libice-dev libpthread-stubs0-dev libsm-dev libx11-6
  libx11-6:i386 libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
  openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre
  openjdk-11-jre-headless x11proto-core-dev x11proto-dev xorg-sgml-doctools
  xtrans-dev
```

- To install Java runtime environment

```
sudo apt install openjdk-11-jre
```

- Verify installation

```
kenneth@kenneth-Latitude-E5440:~$ javac --version
javac 11.0.22
```

- Add A new user with root(Sudo) privileges
 - Add user

```
kenneth@kenneth-Latitude-E5440:~$ sudo adduser apachehadoop
```

- Add the “apachehadoop” user to the sudo group.

```
kenneth@kenneth-Latitude-E5440:~$ sudo usermod -aG sudo apachehadoop
```

- Switch to the user(“apachehadoop” in our case)

```
kenneth@kenneth-Latitude-E5440:~$ sudo su - apachehadoop
[sudo] password for kenneth:
apachehadoop@kenneth-Latitude-E5440:~$
```

- Install and configure ssh
 - Install ssh client and server

```
apachehadoop@kenneth-Latitude-E5440:~$ apt install openssh-server openssh-client -y
```

- Generate public and private key pairs.

```
apachehadoop@kenneth-Latitude-E5440:~$ ssh-keygen -t rsa
```

- Add the generated public key from `id_rsa.pub` to `authorized_keys`.

```
apachehadoop@kenneth-Latitude-E5440:~$ sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```


- Change the permissions of the `authorized_keys` file.

```
apachehadoop@kenneth-Latitude-E5440:~$ sudo chmod 640 ~/.ssh/authorized_keys
```

- Verify if the password-less SSH is functional:

```
apachehadoop@kenneth-Latitude-E5440:~$ ssh localhost
```

- Install Apache Hadoop
 - Download the latest stable version of Hadoop. To get the latest version, go to Apache Hadoop (<https://downloads.apache.org/hadoop/common/stable/>) page. Right click and copy the link then use it in the command that follows

	hadoop-3.4.0-aarch64.tar.gz	2024-03-04 09:36	921M
	hadoop-3.4.0-aarch64.tar.gz.asc	2024-03-04 09:36	836
	hadoop-3.4.0-aarch64.tar.gz.sha512	2024-03-04 09:36	168
	hadoop-3.4.0-rat.txt	2024-03-04 09:36	2.2M
	hadoop-3.4.0-rat.txt.asc	2024-03-04 09:36	833
	hadoop-3.4.0-rat.txt.sha512	2024-03-04 09:36	161
	hadoop-3.4.0-site.tar.gz	2024-03-04 09:36	41M
	hadoop-3.4.0-site.tar.gz.asc	2024-03-04 09:36	833
	hadoop-3.4.0-site.tar.gz.sha512	2024-03-04 09:36	165
	hadoop-3.4.0-src.tar.gz	2024-03-04 09:36	37M
	hadoop-3.4.0-src.tar.gz.asc	2024-03-04 09:36	833
	hadoop-3.4.0-src.tar.gz.sha512	2024-03-04 09:36	164
	hadoop-3.4.0.tar.gz	2024-03-04 09:36	921M
	hadoop-3.4.0.tar.gz.asc	2024-03-04 09:36	833
	hadoop-3.4.0.tar.gz.sha512	2024-03-04 09:36	160

```

hadoop@kenneth-Latitude-E5440:~$ wget https://downloads.apache.org/hadoop/common/stable/hadoop-3.4.0.tar.gz
--2024-04-18 02:39:40-- https://downloads.apache.org/hadoop/common/stable/hadoop-3.4.0.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.208.237, 135.181.214.104, 2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.208.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 965537117 (921M) [application/x-gzip]
Saving to: 'hadoop-3.4.0.tar.gz'

hadoop-3.4.0.t 100% 920.81M  5.27MB/s   in 5m 12s

2024-04-18 02:44:54 (2.95 MB/s) - 'hadoop-3.4.0.tar.gz' saved [965537117/965537117]

```

- Extract the downloaded file:

```

[3980] password for kenneth:
apachehadoop@kenneth-Latitude-E5440:~$ tar -xvzf hadoop-3.4.0.tar.gz

```

- Move the extracted directory to the `/usr/local/hadoop` directory.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo mv hadoop-3.4.0 /usr/local/hadoop

```

- Create directory to store system logs.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo mkdir /usr/local/hadoop/logs

```

- Change the ownership of the hadoop directory.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo chown -R apachehadoop:apachehadoop /home/apachehadoop/hdfs

```

- Configure Hadoop

- Edit file `~/.bashrc` to configure the Hadoop environment variables.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo nano ~/.bashrc

```

- Add the following lines to the file. Save and close the file.

```
GNU nano 4.8 /home/apachehadoo
# Add an "alert" alias for long running commands. Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "[ $? = 0 ]" && echo terminal ||'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

- Activate the environment variables: `$ source ~/.bashrc`
- **Configure Java Environment Variables**
Hadoop has a lot of components that enable it to perform its core functions. To configure these components such as YARN, HDFS, MapReduce, and Hadoop-related project settings, you need to define Java environment variables in `hadoop-env.sh` configuration file.
 - Find the Java path: `$ which javac`
 - Find the OpenJDK directory.: `$ readlink -f /usr/bin/javac`
 - Edit the `hadoop-env.sh` file.

```
apachehadoop@kenneth-Latitude-E5440:~$ sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
GNU nano 4.8 /usr/local/hadoop/etc/hadoop/hadoop-env.sh
###
# Secure/privileged execution
###
#
# Out of the box, Hadoop uses jsvc from Apache Commons to launch daemons
# on privileged ports. This functionality can be replaced by providing
# custom functions. See hadoop-functions.sh for more information.
#
# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol. Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
# export JSVC_HOME=/usr/bin
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_CLASSPATH+=" $HADOOP_HOME/lib/*.jar"
```

- Browse to the hadoop lib directory: `$ cd /usr/local/hadoop/lib`
- Download the Javac activation file.

```
apachehadoop@kenneth-Latitude-E5440:/usr/local/hadoop/lib$ sudo wget https://jcenter.bintray.com/javac/activation/javac.activation-apl/1.2.0/javac.activation-apl-1.2.0.jar
```

- Verify the Hadoop version: `$ hadoop version`

- Edit the **core-site.xml** configuration file to specify the URL for your NameNode.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
GNU nano 4.8 /usr/local/hadoop/etc/hadoop/core-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
    <description>The default file system URI</description>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/tmp/hadoop-${user.name}</value>
    <description>temporary path URI</description>
  </property>
</configuration>

```

- Create a directory for storing node metadata and change the ownership to hadoop.

```

apachehadoop@kenneth-Latitude-E5440:~$ sudo mkdir -p /home/apachehadoop/hdfs/{namenode,datanode}
apachehadoop@kenneth-Latitude-E5440:~$ sudo chown -R apachehadoop:apachehadoop /home/apachehadoop/hdfs

```

- Edit **hdfs-site.xml** configuration file to define the location for storing node metadata, fs-image file.

```

9b9c9e99qoob@kenneth-Latitude-E5440:~$ nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml
GNU nano 4.8 /usr/local/hadoop/etc/hadoop/hdfs-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>

  <property>
    <name>dfs.name.dir</name>
    <value>file:///home/apachehadoop/hdfs/namenode</value>
  </property>

  <property>
    <name>dfs.data.dir</name>
    <value>file:///home/apachehadoop/hdfs/datanode</value>
  </property>
</configuration>

```

- Edit **mapred-site.xml** configuration file to define MapReduce values.

```
apachehadoop@kenneth-Latitude-E5440:~$ sudo nano /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

```
GNU nano 4.8 /usr/local/hadoop/etc/hadoop/mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobtracker.address</name>
    <value>localhost:54311</value>
  </property>
</configuration>
```

- Edit the **yarn-site.xml** configuration file and define YARN-related settings.

```
apachehadoop@kenneth-Latitude-E5440:~$ sudo nano /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

```
GNU nano 4.8 /usr/local/hadoop/etc/hadoop/yarn-site.xml
<?xml version="1.0"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<configuration>

  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>localhost</value>
  </property>

<!-- Site specific YARN configuration properties -->

</configuration>
```

- Validate the Hadoop configuration and format the HDFS NameNode.
hdfs namenode -format

- Start all hadoop services

```
apachehadoop@kenneth-Latitude-E5440:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as apachehadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [kenneth-Latitude-E5440]
Starting resourcemanager
Starting nodemanagers
```

- Check if the services have been successfully started

```
apachehadoop@kenneth-Latitude-E5440:~$ jps
17764 DataNode
22330 Jps
17994 SecondaryNameNode
18253 ResourceManager
18398 NodeManager
17615 NameNode
```

- Accessing Different components dashboard
 - The name node: <http://localhost:9870>

Activities Firefox Web Browser Apr 21 16:02

Nodes of the Hadoop J... Namenode X Hadoop UI Apache H Namenode in Namenode in Problem l Application a + -

← → ↻ localhost:9870/dfshealth.html#tab-overview

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Overview 'localhost:9000' (✓active)

Started:	Sun Apr 21 15:23:24 +0100 2024
Version:	3.4.0, rbd8b77f398f626bb7791783192ee7a5dfaec760
Compiled:	Mon Mar 04 07:35:00 +0100 2024 by root from (HEAD detached at release-3.4.0-RC3)
Cluster ID:	CID-9617eb66-c601-485fa443-e42a56cc8c51
Block Pool ID:	BP-429833572-127.0.1.1-1713709193602

Summary

Security is off.
Safemode is off.
23 files and directories, 6 blocks (6 replicated blocks, 0 erasure coded block groups) = 29 total filesystem object(s).
Heap Memory used 135.58 MB of 221 MB Heap Memory. Max Heap Memory is 1.92 GB.
Non Heap Memory used 67.9 MB of 70.9 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

○ The datanode: <http://localhost:9864>



DataNode on kenneth-Latitude-E5440:9866

Cluster ID:	CID-9617eb66-c601-485f-a443-e42a56cc8c51
Started:	Sun Apr 21 15:23:30 +0100 2024
Version:	3.4.0, rbd8b77f398f626bb7791783192ee7a5dfaee760

Block Pools

Namenode Address	Namenode HA State	Block Pool ID	Actor State	Last Heartbeat Sent	Last Heartbeat Response	Last Block Report	Last Block Report Size (Max Size)
localhost:9000	active	BP-429833572-127.0.1.1-1713709193602	RUNNING	1s	1s	41 minutes	0 B (128 MB)

Volume Information

○ The cluster matrices: <http://localhost:8088>

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
0	0	0	0	<1

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned
1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mb), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime
----	------	------	------------------	------------------	-------	----------------------	-----------	------------	------------

Showing 0 to 0 of 0 entries

DEPLOYING A SIMPLE APPLICATION ON THE HADOOP CLUSTER (java word count application)

Source code(a clean version will be attached during submission)

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

This source is stored in the Hadoop user home directory as **wordCount.java**

Running The Application on Hadoop

Let's compile the code to come out with a jar file

```
apachehadoop@kenneth-Latitude-E5440:~$ hadoop com.sun.tools.javac.Main WordCount.java
apachehadoop@kenneth-Latitude-E5440:~$ jar cf wc.jar WordCount*.class
```

You will have the following files created: in the home directory ofcourse



Let's create our local input folder(wordCountInput) and file(input1)

Create folder

```
apachehadoop@kenneth-Latitude-E5440:~$ mkdir wordCountInput
```

Create file and fill the content

```
apachehadoop@kenneth-Latitude-E5440:~/wordCountInput$ vi input1.txt
```

Let's create our local output folder(wordCountOutput) and file(output)

Create folded

```
apachehadoop@kenneth-Latitude-E5440:~$ mkdir wordCountOutput
```

Create an empty file

```
apachehadoop@kenneth-Latitude-E5440:~/wordCountOutput$ touch output.txt
```

Create remote(hdfs) folders and file then upload the local file to it

```
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -mkdir /user
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -mkdir /user/apachehadoop
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -mkdir /user/apachehadoop/wordCountInput
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -mkdir /user/apachehadoop/wordCountOutput
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -put wordCountInput/input1.txt /user/apachehadoop/wordCountInput
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -put wordCountInput/input2.txt /user/apachehadoop/wordCountInput
apachehadoop@kenneth-Latitude-E5440:~$ hdfs dfs -put wordCountOutput/output.txt /user/apachehadoop/wordCountOutput
```

Run the application:

```
apachehadoop@kenneth-Latitude-E5440:~$ hadoop jar wc.jar WordCount ./wordCountInput/input1.txt wordCountOutput/output.txt
2024-04-21 15:29:22,172 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at localhost/127.0.0.1:8032
2024-04-21 15:29:22,812 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and
execute your application with ToolRunner to remedy this.
2024-04-21 15:29:22,862 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/apachehadoop/.staging
job_1713709430459_0002
2024-04-21 15:29:23,334 INFO input.FileInputFormat: Total input files to process : 1
2024-04-21 15:29:23,516 INFO mapreduce.JobSubmitter: number of splits:1
2024-04-21 15:29:23,938 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1713709430459_0002
2024-04-21 15:29:23,938 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-04-21 15:29:24,218 INFO conf.Configuration: resource-types.xml not found
2024-04-21 15:29:24,219 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-04-21 15:29:24,808 INFO impl.YarnClientImpl: Submitted application application_1713709430459_0002
2024-04-21 15:29:24,928 INFO mapreduce.Job: The url to track the job: http://kenneth-Latitude-E5440:8088/proxy/application_1713709430459_0002
2024-04-21 15:29:24,929 INFO mapreduce.Job: Running job: job_1713709430459_0002
2024-04-21 15:29:36,279 INFO mapreduce.Job: Job job_1713709430459_0002 running in uber mode : false
2024-04-21 15:29:36,281 INFO mapreduce.Job: map 0% reduce 0%
2024-04-21 15:29:43,685 INFO mapreduce.Job: map 100% reduce 0%
2024-04-21 15:29:52,790 INFO mapreduce.Job: map 100% reduce 100%
2024-04-21 15:29:52,808 INFO mapreduce.Job: Job job_1713709430459_0002 completed successfully
2024-04-21 15:29:52,947 INFO mapreduce.Job: Counters: 54
```

	<pre>Map-Reduce Framework Map input records=1 Map output records=23 Map output bytes=220 Map output materialized bytes=231 Input split bytes=130 Combine input records=23 Combine output records=19 Reduce input groups=19 Reduce shuffle bytes=231 Reduce input records=19 Reduce output records=19 Spilled Records=38 Shuffled Maps =1 Failed Shuffles=0 Merged Map outputs=1 GC time elapsed (ms)=93 CPU time spent (ms)=2160 Physical memory (bytes) snapshot=527302656 Virtual memory (bytes) snapshot=5466943488 Total committed heap usage (bytes)=457179136 Peak Map Physical memory (bytes)=310640640 Peak Map Virtual memory (bytes)=2730278912 Peak Reduce Physical memory (bytes)=216662016 Peak Reduce Virtual memory (bytes)=2736664576 Shuffle Errors BAD_ID=0 CONNECTION=0 IO_ERROR=0 WRONG_LENGTH=0 WRONG_MAP=0 WRONG_REDUCE=0 File Input Format Counters Bytes Read=129 File Output Format Counters Bytes Written=149</pre>
<pre>File System Counters FILE: Number of bytes read=231 FILE: Number of bytes written=618425 FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=259 HDFS: Number of bytes written=149 HDFS: Number of read operations=8 HDFS: Number of large read operations=0 HDFS: Number of write operations=2 HDFS: Number of bytes read erasure-coded=0 Job Counters Launched map tasks=1 Launched reduce tasks=1 Data-local map tasks=1 Total time spent by all maps in occupied slots (ms)=4834 Total time spent by all reduces in occupied slots (ms)=5630 Total time spent by all map tasks (ms)=4834 Total time spent by all reduce tasks (ms)=5630 Total vcore-milliseconds taken by all map tasks=4834 Total vcore-milliseconds taken by all reduce tasks=5630 Total megabyte-milliseconds taken by all map tasks=4950016 Total megabyte-milliseconds taken by all reduce tasks=5765120</pre>	

Result And Discussion

Input was:

```
1 my name in kouete tepe kenneth, my group and i are working on hadoop and more precisely on apache hadoop with mapreduce cluster
```

And the output is:

1	and	2	
2	apache	1	
3	are	1	
4	cluster		1
5	group	1	
6	hadoop	2	
7	i	1	
8	in	1	
9	kenneth,		1
10	kouete	1	
11	mapreduce		1
12	more	1	
13	my	2	
14	name	1	
15	on	2	
16	precisely		1
17	tepe	1	
18	with	1	
19	working		1

The results of our exploration into Hadoop as a Platform as a Service (PaaS) solution reveal its pivotal role in modern data processing. Through an in-depth analysis of its environment presentation, installation, configuration, and application deployment process, we've unveiled the robustness and flexibility inherent in Hadoop's architecture. By leveraging its distributed storage and processing capabilities, organizations can effectively manage and analyse large-scale datasets, driving innovation and informed decision-making. Furthermore, our discussion highlights Hadoop's seamless integration with cloud computing environments, offering scalability and cost-effectiveness. Overall, the results underscore the transformative potential of Hadoop as a PaaS solution, empowering organizations to harness the power of big data and propel themselves forward in today's data-driven landscape.

CONCLUSION

Hadoop's significance in big data processing is undeniable, offering robust capabilities for storing, processing, and analyzing vast datasets. Throughout this presentation, we've explored its distributed architecture, seamless integration with cloud computing environments, and the step-by-step process from installation to deploying a simple application. By embracing Hadoop as a Platform as a Service (PaaS) solution, organizations unlock opportunities for innovation, empowering them to derive actionable insights and drive transformative changes in today's data-driven landscape.

REFERENCE

<https://docs.vultr.com/install-and-configure-apache-hadoop-on-ubuntu-20-04>

<https://downloads.apache.org/hadoop/common/stable/>

<https://www.youtube.com/watch?v=3oHwfKM0bNM>

[MapReduce Architecture - GeeksforGeeks: https://www.geeksforgeeks.org/mapreduce-architecture/](https://www.geeksforgeeks.org/mapreduce-architecture/)

<https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

<https://hadoop.apache.org/>