Big Data and Hadoop Course Guide

Course Overview

Explore the world of Big Data with our comprehensive course. Begin with an introduction to Big Data and Hadoop. Dive into the Hadoop ecosystem, understand its core components including HDFS and YARN, and explore the evolution from Hadoop 1.x to Hadoop 3.x. Master MapReduce, a fundamental programming paradigm in Big Data, and gain practical insights with real-world examples. By the end of this course, you'll have a solid foundation in Big Data and Hadoop, enabling you to work effectively in data-intensive environments.

Big Data, Hadoop, and MapReduce

1. Big Data Introduction

- **Definition:** Large and complex datasets that require specialized processing techniques.
- **Characteristics of Big Data (5Vs): It is based on 3V of Big Data - as per grtner research paper in 2001**
    - Volume
    - Velocity
    - Variety
    - Veracity: the Degree of trustworthiness.
    - Value

Benefits of BigData :

1.Improved decision making : Any organisation will want to be data driven.

2.Continuous intelligence: it allows us to integrate automated, real time data streaming with advanced data analytics.

3. Increased Agility and innovation:

4. Better Customer Experiences.

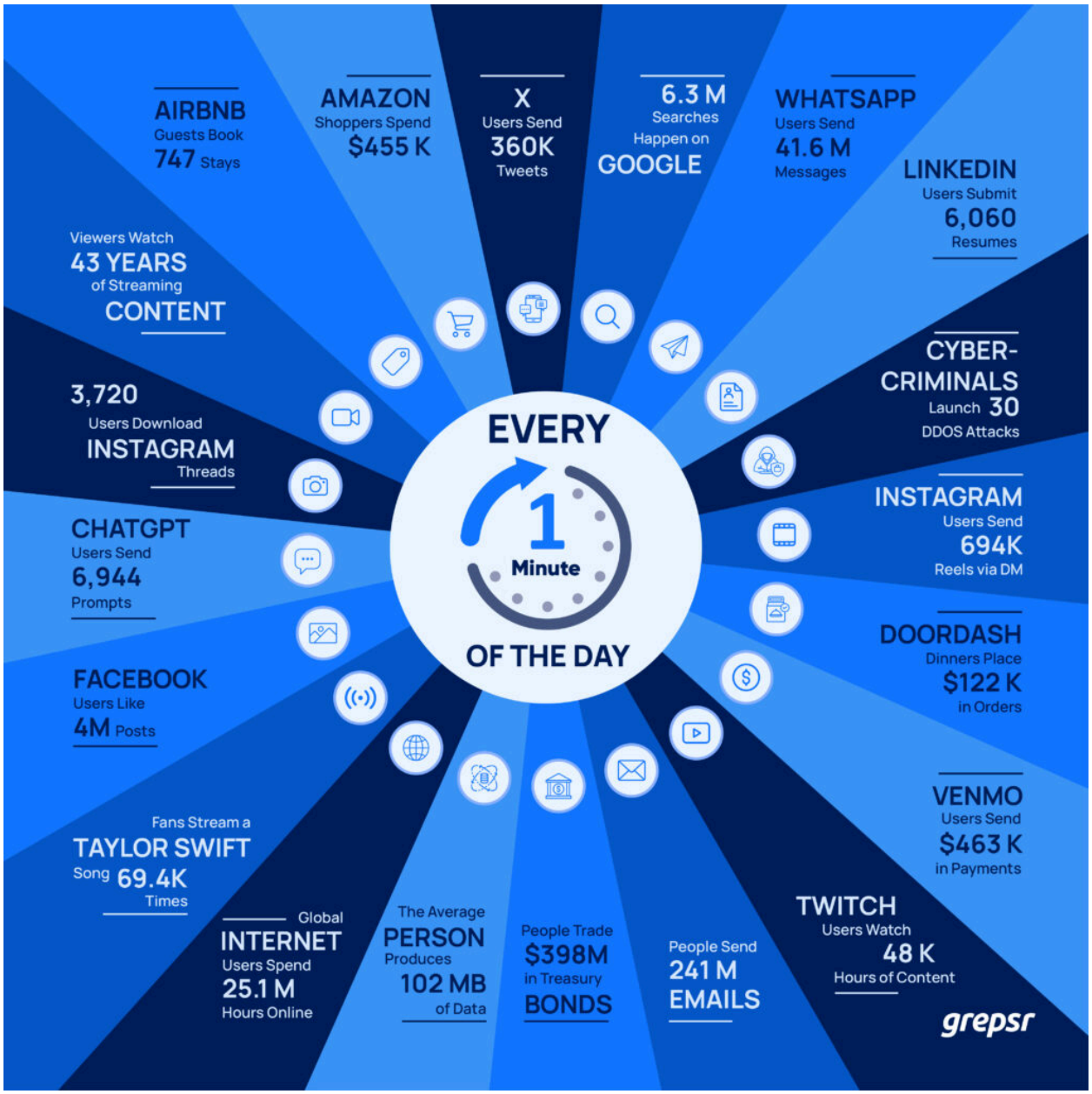
5. Improved risk management

6. More efficient operations: Data can be processed faster and generate insight.

Limitation/Challenges 👍

- Lack of Data talent and skills : Data Scientist, Data analyst, Data Engineers are in short supply.
- Speed of Data growth:
- Problems with data quality: raw data is messy and can be difficult to curate.

- Compliance violations: Big data contains lot of sensitive data and information, making it a ricky task to continuously ensure data processing and storagemeet data privacy and regulatory requirement.
- Integration complexity : Real time v/s batched data
- Security concerns :



- **Use Cases:**
    - Fraud detection in financial transactions

    - Predictive maintenance in manufacturing
    - Recommendation engines (e.g., Netflix, Amazon)
- **Real-World Stories:**
    - How Walmart optimizes inventory using Big Data
    - Google's real-time search query processing

How is data driven business performing ?

- 58% of companies that make data-based decisions are more likely to beat revenue targets than those that don't.
- Organizations with advanced insights-driven business capabilities are 2.8x more likely to report double-digit YOY growth.
- Data-driven organizations generate, on average, more than 30% growth per year.

2. Getting Started: Hadoop

- **Definition:** An open-source framework for storing and processing Big Data in a distributed environment.
- **How Hadoop Works ?**
    - Breaks down large workloads into smaller workloads that can be run in parallel
    - Uses Distributed storage to store data across multiple computers.
    - Replicates nodes in the cluster to protect against hardware or software failures

Hadoop benefits:

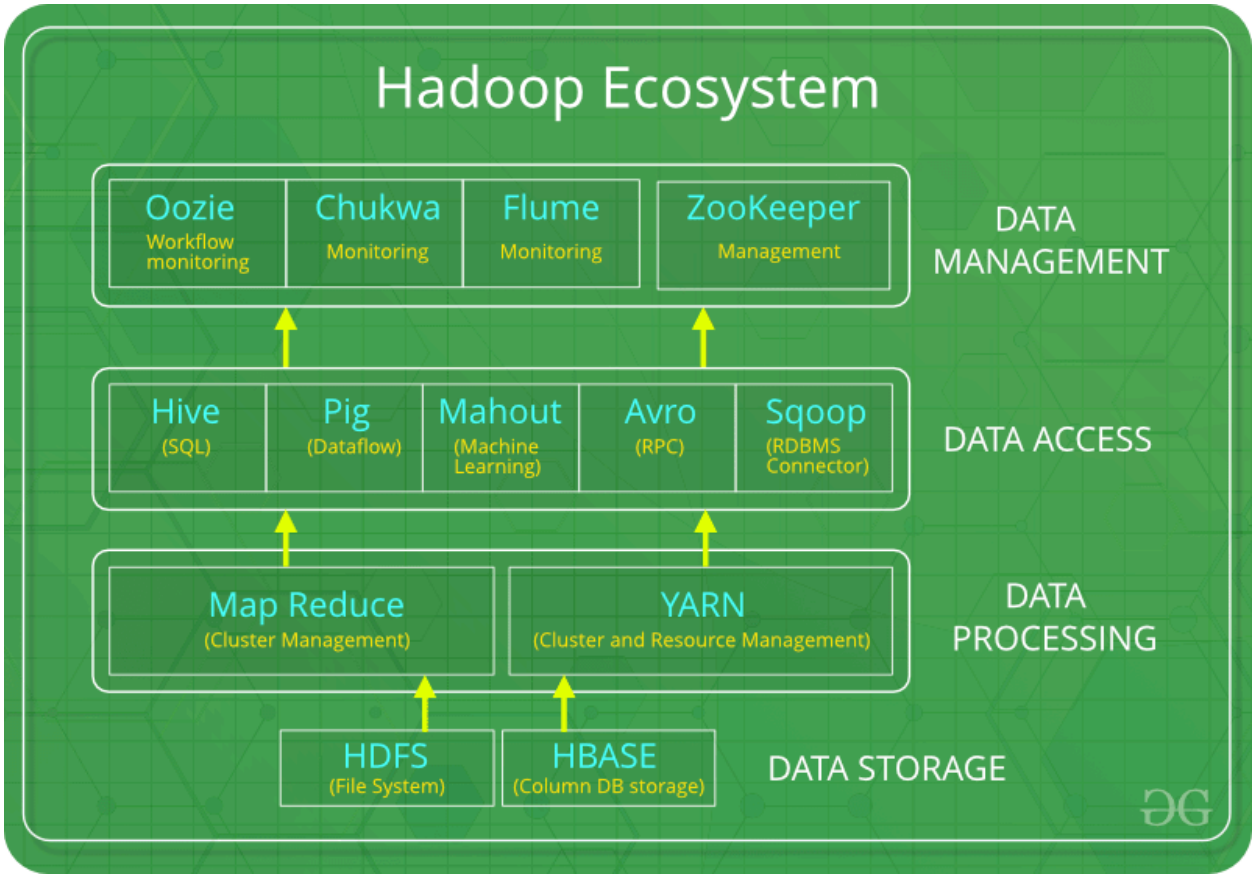
Scalability: it can handle large amount of data

Resilience: It can protect against hardware or software failures.

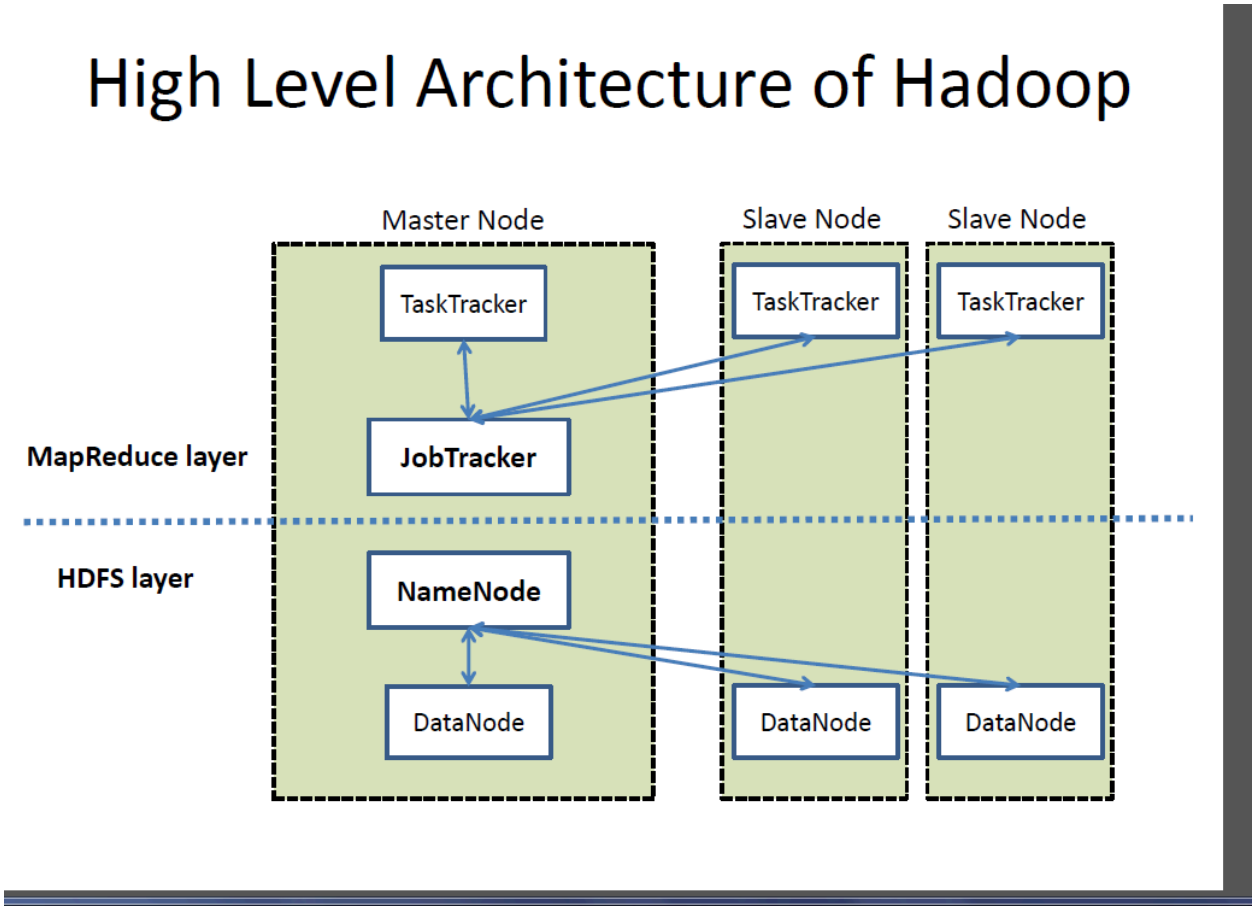
Flexibility: Can handle a variety of data, including structured, unstructured and semi-structured.

- **Why Hadoop?**
    - Handles structured and unstructured data efficiently
    - Cost-effective and scalable
    - Fault-tolerant system
- **Industries Using Hadoop:**
    - Healthcare
    - E-commerce
    - Banking and Finance

3. Hadoop Framework: Stepping into Hadoop



- **Core Components:**
  - HDFS (Hadoop Distributed File System)
  - YARN (Yet Another Resource Negotiator)
  - MapReduce (Processing Framework)
- **Key Features:**
  - Distributed Storage
  - Parallel Processing
  - Fault Tolerance
- **Best Practices:**
  - Efficient data partitioning
  - Proper replication strategy
  - Regular monitoring of Hadoop clusters



4. HDFS: What and Why?

- **Definition:** Distributed file system designed to run on commodity hardware.
- **Features:**
  - High throughput
  - Data replication
  - Scalability
- **Real-World Use Cases:**
  - Social media platforms storing vast amounts of data (e.g., Facebook, Twitter)
  - Weather forecasting models processing large datasets
- Key pointers of HDFS:
- Architecture : HDFS uses name node to manage file system metadata and datanode to store actual data for individual systems.
- Fault tolerance:Data redundancy across multi nodes that ensures data availability even if some nodes fail.

5. Hadoop 2.x - YARN

- **Introduction to YARN:**
  - Separates resource management and job scheduling
  - Allows multiple applications to run on Hadoop

Features	Hadoop 1.x	hadopp2.x	Hadoop 3.x
Architecture	Uses mapreduce as sole processing framework	Introduces YARN for resource management	Enhanced YARN for better resource allocations.
Resource management	Job Tracker and Task tracker causes scalability issues.	YARN	Improved YARN
Storage(HDFS)	Singe notebook - prone to failure	HA(High Availability)name Node	Execute Coding
Data performance	Only Mapreduceis supported	Mapreduce, Spark, Tez, HIVE etc are Support	Here we have better integration with Ai tooks
Scalability	It supports 4,00 nodes max	It support 20,000 make	It supports more than 50, 000
HDFS Block Size	64MB	128 MB	256mode
Compatibility	Limited Support from economic system	Sports modern tools like Spark, Flink, Tez and Flink	Fully compatible with cloud platform (AWS, Azure and GCP)

- **Components of YARN:**
  - Resource Manager
  - Node Manager
  - Application Master
- **Advantages:**
  - Better cluster utilization
  - Improved scalability
- **Use Cases:**
  - Multi-tenancy applications in cloud computing
  - Real-time data processing with Apache Spark

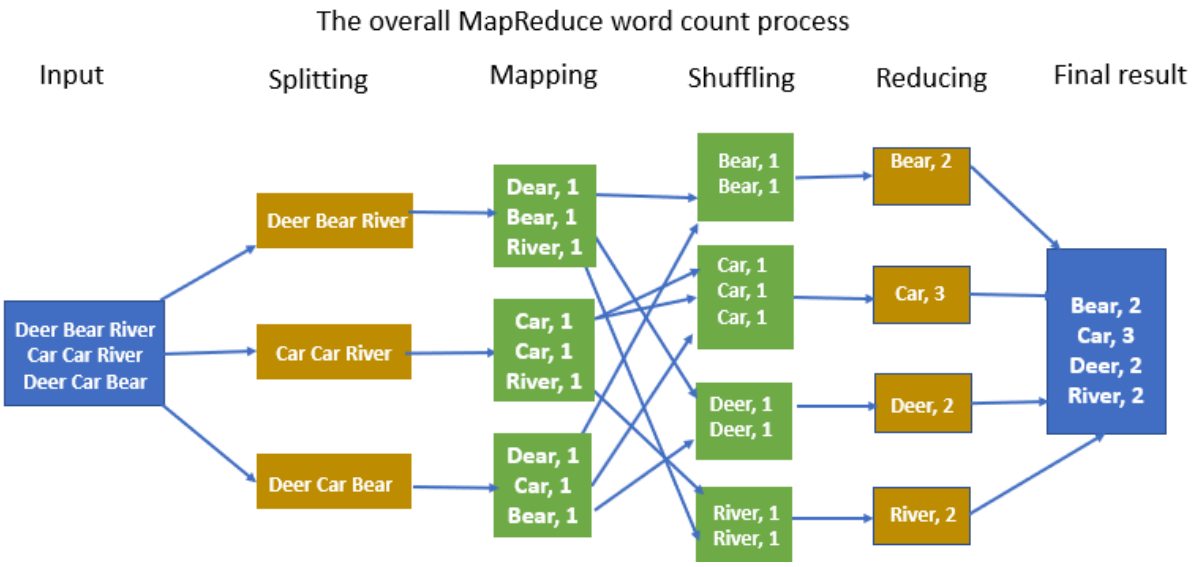
6. MapReduce: A Programming Paradigm

- **Definition:** A processing technique for parallel computation on large datasets.
- **Key Components:**
  - Mapper
  - Reducer
- **Steps in MapReduce Execution:**
  - Input Splitting
  - Mapping
  - Shuffling & Sorting
  - Reducing
  - Final Output
- **Best Practices:**

- Optimize mapper and reducer functions
- Use combiners to reduce data transfer
- Proper partitioning for load balancing

7. Understanding MapReduce with an Example

- **Example Scenario:**
  - Word count program using MapReduce



How MapReduce processes large log files

- **Industry Use Cases:**
  - Log analysis in cybersecurity
  - Sentiment analysis in social media

8. Hadoop 1.x vs Hadoop 2.x

- **Key Differences:**
  - Introduction of YARN in Hadoop 2.x
  - Improved resource management in Hadoop 2.x
  - Support for non-MapReduce applications
- **Advantages of Hadoop 2.x:**
  - Scalability
  - Better cluster utilization
  - High availability

9. Hadoop 3.x

- **Improvements Over Hadoop 2.x:**
  - Erasure coding for storage efficiency

- Support for multiple standby NameNodes
- Containerization with Docker

- **Use Cases:**
  - Large-scale machine learning models
  - Data lake implementation in enterprises
- **Best Practices:**
  - Upgrade strategy from Hadoop 2.x to 3.x
  - Optimized resource allocation
  - Efficient cluster monitoring

Conclusion

This course provides an in-depth understanding of Big Data, Hadoop, and MapReduce, covering foundational concepts, practical applications, and best practices. By the end of this training, participants will be well-equipped to handle data-intensive environments using Hadoop efficiently.

Case Studies on Big Data, Hadoop, and MapReduce

Case Study 1: Retail Industry – Personalized Customer Experience

Background

A leading e-commerce company wanted to improve customer experience by providing personalized recommendations. Traditional database systems struggled to handle the large volume of customer interactions, transactions, and product data in real time.

Application of Big Data and Hadoop

- **HDFS** stored vast amounts of structured and unstructured data, including clickstream data, user purchase history, and reviews.
- **MapReduce** processed large datasets to identify customer preferences and purchasing patterns.
- **Hadoop Ecosystem (Hive, HBase, Pig)** enabled fast querying and analysis of data.

Outcome

- Improved recommendation engine, resulting in a 35% increase in customer engagement.
- Reduced data processing time from hours to minutes.
- Enhanced customer retention and higher conversion rates.

Case Study 2: Healthcare – Predicting Disease Outbreaks

Background

A global health organization wanted to analyze patient records, social media data, and climate conditions to predict disease outbreaks in various regions.

Application of Big Data and Hadoop

- **Hadoop Framework** handled petabytes of medical records and climate data efficiently.
- **MapReduce** analyzed social media feeds to identify early signs of disease spread.
- **YARN (Yet Another Resource Negotiator)** managed and allocated resources dynamically for real-time analytics.

Outcome

- Early disease outbreak prediction with 85% accuracy.
- Faster response time for healthcare professionals.
- Reduced the spread of contagious diseases through early intervention.

Case Study 3: Banking and Finance – Fraud Detection

Background

A major bank needed to enhance its fraud detection system by analyzing millions of transactions in real time to detect fraudulent activities.

Application of Big Data and Hadoop

- **HDFS** stored historical transaction data for long-term analysis.
- **MapReduce** processed real-time transaction streams to identify fraudulent patterns.
- **Hadoop 2.x - YARN** ensured optimized resource allocation for simultaneous processing of fraud detection algorithms.

Outcome

- Fraud detection improved by 60%, reducing financial losses.
- Real-time alerts prevented fraudulent transactions before they were completed.
- Enhanced regulatory compliance and customer trust.

Case Study 4: Telecommunications – Network Optimization

Background

A telecom company faced frequent network failures and customer dissatisfaction due to poor service quality. They needed an efficient system to analyze call data records (CDRs) and network logs.

Application of Big Data and Hadoop

- **HDFS** stored massive amounts of CDRs and network logs.
- **MapReduce** analyzed dropped calls and latency issues.
- **Hadoop 3.x** provided better fault tolerance and storage efficiency.

Outcome

- 40% reduction in network downtime.
- Proactive maintenance prevented major outages.
- Improved customer satisfaction and reduced churn rate.

Case Study 5: Social Media – Sentiment Analysis

Background

A marketing firm wanted to analyze social media trends to understand customer sentiment toward a newly launched product.

Application of Big Data and Hadoop

- **HDFS** stored unstructured social media data from various platforms.
- **MapReduce** processed text data to extract sentiments (positive, negative, neutral).
- **Hadoop Ecosystem (Spark, Hive)** enabled real-time sentiment analysis.

Outcome

- Improved marketing strategies based on customer sentiment.
- Real-time feedback helped companies adjust their campaigns promptly.
- Increased brand awareness and customer engagement.

Conclusion

Big Data, Hadoop, and MapReduce are transforming industries by enabling organizations to analyze large-scale data efficiently. Whether it's personalized recommendations, fraud detection, or sentiment analysis, these technologies provide actionable insights that drive business success. By leveraging Hadoop's ecosystem, companies can store, process, and analyze vast amounts of data to make data-driven decisions in real time.

Preparing a Local System for Big Data, Hadoop, and MapReduce Demos

Before starting hands-on demos, ensure that your system is properly set up with the required tools. Follow these steps:

System Requirements

- **OS:** Windows 10/11, Linux (Ubuntu, CentOS), or macOS
- **RAM:** Minimum 8 GB (Recommended 16 GB for better performance)
- **Processor:** Intel i5/i7 (or AMD equivalent)
- **Storage:** At least 50 GB free space for Hadoop and datasets
- **Software:** Java 8+, Hadoop (latest stable version), Python (for additional data processing), VirtualBox (if using Cloudera or Hortonworks sandbox)

Steps to Set Up the Environment

1. **Install Java Development Kit (JDK 8 or higher)**
   - Download and install JDK from Oracle or OpenJDK
   - Set `JAVA_HOME` environment variable
2. **Install Hadoop (Standalone Mode for Local Setup)**
   - Download Hadoop from the Apache Hadoop website
   - Extract and configure `core-site.xml`, `hdfs-site.xml`, and `mapred-site.xml`
   - Format the HDFS and start the Hadoop services
3. **Set Up HDFS for Storage**
   - Initialize HDFS directory structure
   - Create user-specific directories and set permissions
4. **Set Up a Hadoop Cluster (Optional for Multi-Node Setup)**
   - Configure NameNode and DataNode settings
   - Set up YARN for resource management
5. **Install Additional Tools (Optional but Recommended)**
   - **Hive:** For SQL-based querying
   - **Spark:** For faster processing
   - **Sqoop & Flume:** For data ingestion
   - **Kafka:** For real-time data streaming

Python-Based Hands-on Demos

The following table presents short **Python-based** demos for each Big Data topic.

Topic	Demo Title	Scenario

Big Data Introduction	Analyzing Large CSV Data	Process a large CSV file to calculate the average of a column
Getting Started: Hadoop	Uploading a File to HDFS	Store a dataset in HDFS and retrieve it
Hadoop Framework	Word Count in Hadoop using Python	Count words in a large text file using Hadoop Streaming
HDFS: What and Why?	Checking File Storage in HDFS	List files and directories in HDFS
Hadoop 2.x - YARN	Submitting a Spark Job with Python	Run a simple PySpark transformation using YARN
MapReduce: A Programming Paradigm	Running MapReduce Job in Python	Process text data using Python and MRJob
Understanding MapReduce with an Example	Processing Log Files	Extract IP counts from a log file
Hadoop 1.x vs Hadoop 2.x	Performance Comparison of Word Count	Measure execution time in Hadoop 1.x vs 2.x
Hadoop 3.x Features	Storage Optimization with Erasure Coding	Enable erasure coding for better storage efficiency

HIVE & Spark

Features	**Hadoop(YARN/HDFS)**	**Mapreduce**	**HIVE**	**Spark**
Purpose	Storage and resource management	Bath data processing	SQL queries on hadoop	Fast Data processing (Batch + Streaming Data)
Processing speed	Slow (Disk Based)	Slow (Disk -Space)	Faser(Spark/TEz)	Vary Fast (IN-Memory)
Ease of use	Complex	Required JAVA or Python	SQL Based (Easy)	Easy with APis(Scala, Python, SQL)
Real time Support	No	No	No	Yes
Best use Case	Storing Big data	Processing Large logs	Datawarehousing (DWH)	Real-time Analytics

How all Bigdata models work together 👍
Step 1: Store the data in Hadoop HDFS(raw data from logs and Transaction.
Step 2: use Mapreduce to process data (Batch processing for large scale transformation)

Step 3: Query data with HIVE.(For SQL based Analytics on top of hadoop)
Step 4: Use Spark for faster Computation.(For real-time insight and machine learning.

Demo 1: Word counting using python

Install pyspark
Pip install pyspark

```
Step 1: Intilise Pyspark session
Step 2: Load Data
Step 3: Word Counting using spark transformation
Step 4: Print results
Step 5 Closing sparks session

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("BigDataDemo").getOrCreate()

data = ["bigdata hadoop spark", "hadoop hdfs hive", "spark mapreduce hadoop"] rdd =
spark.sparkContext.parallelize(data)


word_counts = ( rdd.flatMap(lambda line: line.split()) # Split into words .map(lambda word: (word, 1)) #
Create (word,1) pairs .reduceByKey(lambda a, b: a + b) # Aggregate counts )


print("Word Count using Spark:") for word, count in word_counts.collect(): print(f"{word}: {count}")


spark.stop()
```

Demo 2: Running SQl query on structured data using HIVE in python

Step 1: Create a sample CSV file for sales data
Step 2: Load the CSV into HIVE
Step 3: use python to execute SQl queries

```
from pyhive import hive

# Connect to Hive Server

conn = hive.Connection(host="localhost", port=10000, database="default")


# Create a Table in Hive
query_create = """ CREATE TABLE IF NOT EXISTS sales ( product STRING, price INT ) ROW
FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE; """ cursor = conn.cursor()
```

```
cursor.execute(query_create)

# Load Data into Hive Table (Assuming data is already loaded) # Query Data

query_select = "SELECT product, AVG(price) FROM sales GROUP BY product;"
cursor.execute(query_select)

# Fetch and Print Results
for row in cursor.fetchall(): print(row)

# Close Connection
cursor.close() conn.close()
```