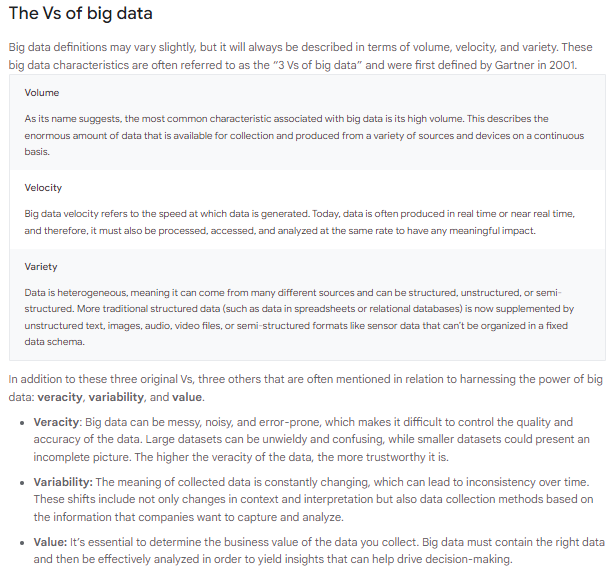
Big data

Big data refers to extremely large and diverse collections of structured, unstructured, and semi-structured data that continues to grow exponentially over time. These datasets are so huge and complex in volume, velocity, and variety, that traditional data management systems cannot store, process, and analyze them.

Link to article I have read:: <https://cloud.google.com/learn/what-is-big-data#:~:text=Big%20data%20describes%20large%20and,problems%20and%20make%20informed%20decisions>.



HADOOP::

What is Hadoop?

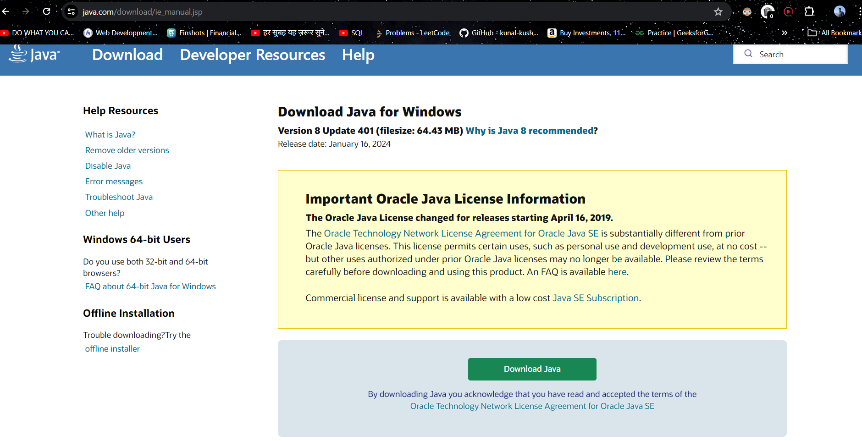
Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyze massive datasets in parallel more quickly.

Basically Hadoop uses distributed computing model

**Installation and setup Hadoop:**

**Hadoop Is java based to for that we need to install JDK and JRE**

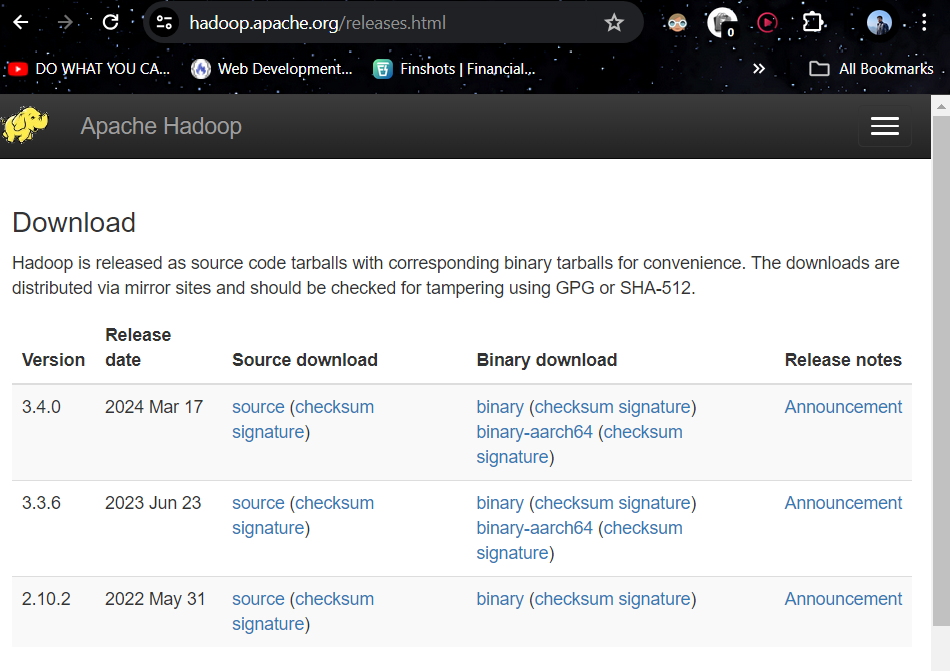
1. JRE 1.8 — Offline installer for JRE
2. [Java Development Kit — 1.8](https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html#license-lightbox)
3. A Software for Un-Zipping like [7Zip](https://www.7-zip.org/download.html) or [Win Rar](https://www.win-rar.com/download.html?L=0)  
   \* I will be using a 64-bit windows for the process, please check and download the version supported by your system x86 or x64 for all the software.



Now Hadoop file to download:

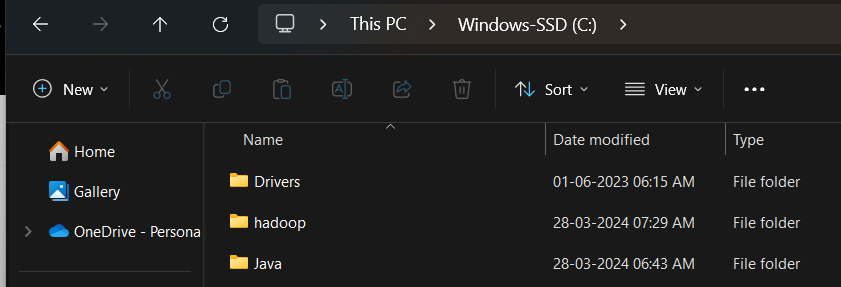
I am downloading 3.3.6 version of Hadoop from

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

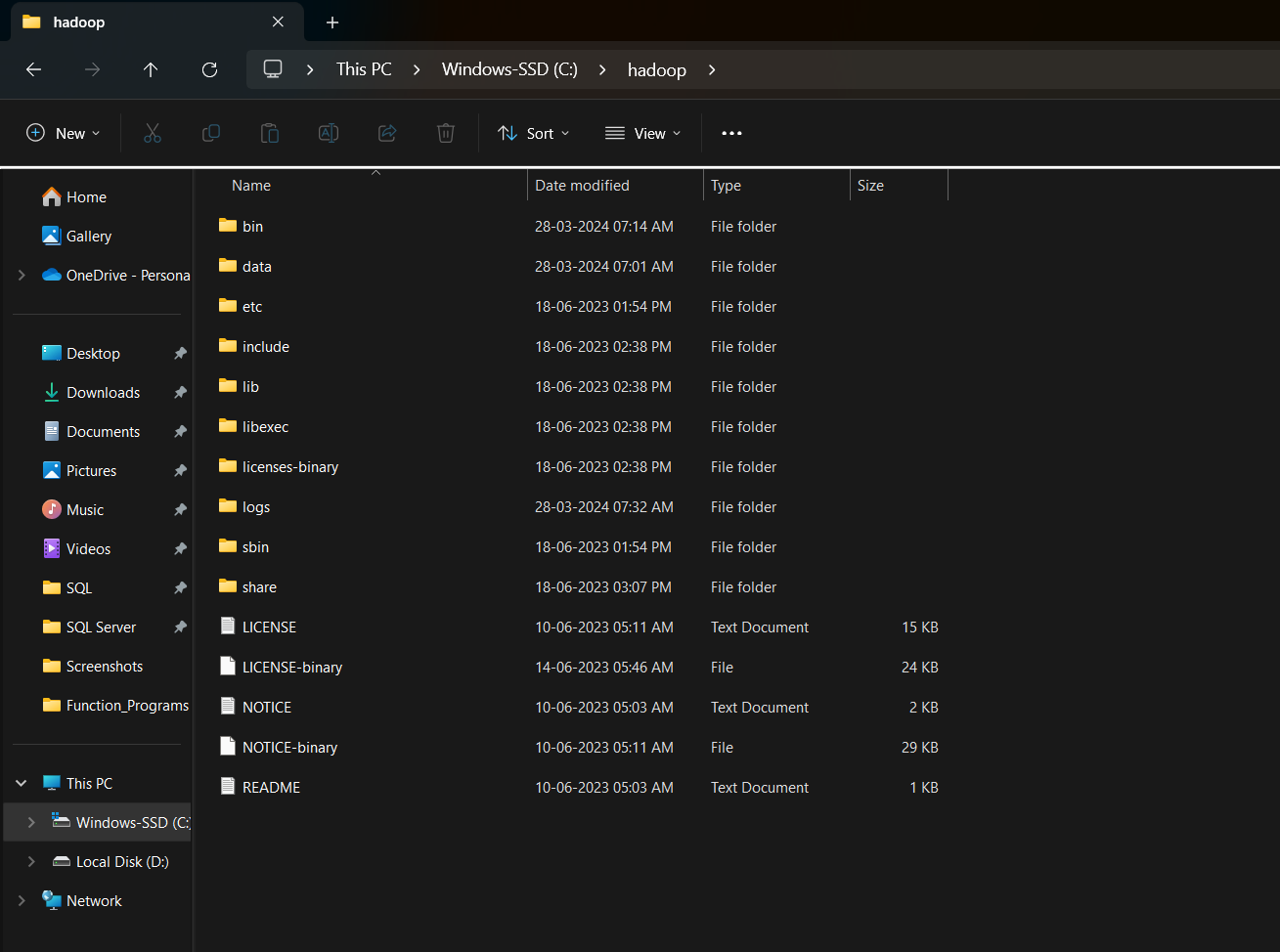


**2. Unzip and Install Hadoop**

After Downloading the Hadoop, we need to Unzip the hadoop-3.3.6.tar.gz file.

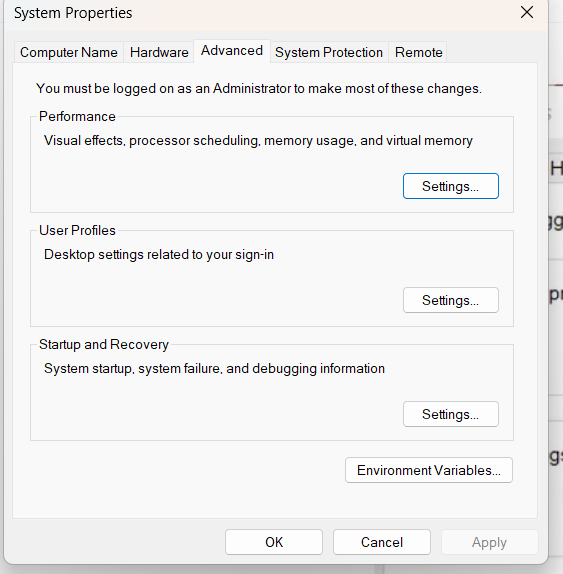


Once extracted, we would get a new file hadoop-2.9.2.tar.  
Now, once again we need to extract this tar file.



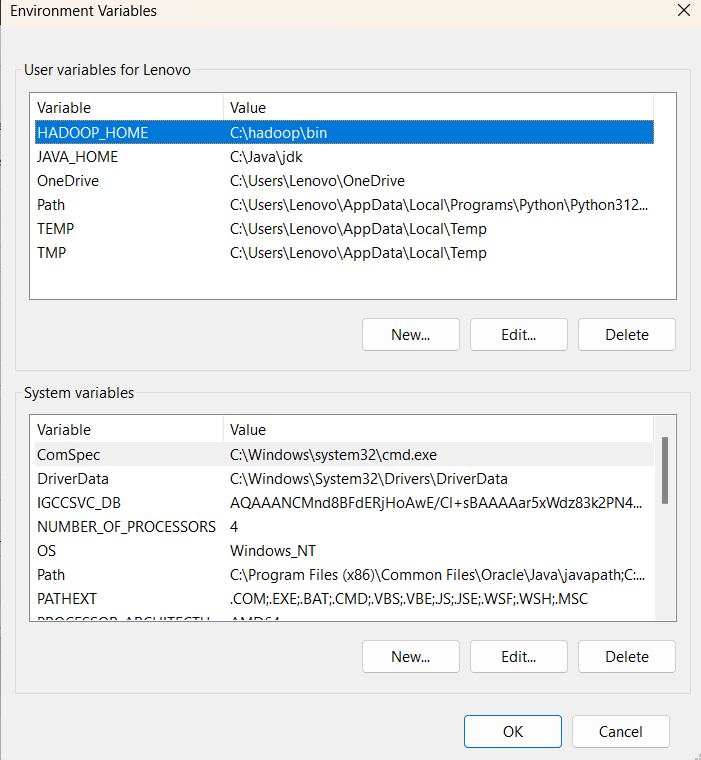
* Please note while creating folders, DO NOT ADD SPACES IN BETWEEN THE FOLDER NAME.(it can cause issues later)
* I have placed my Hadoop in C: drive

# 3. Setting Up Environment Variables

Another important step in setting up a work environment is to set your Systems environment variable.To edit environment variables, go to Control Panel > System > click on the “Advanced system settings” link

**Setting JAVA\_HOME & HADOOP\_HOME**

* Open environment Variable and click on “New” in “User Variable”



* Now as shown, add JAVA\_HOME in variable name and path of Java(jdk) in Variable Value.
* Click OK and we are half done with setting JAVA\_HOME.

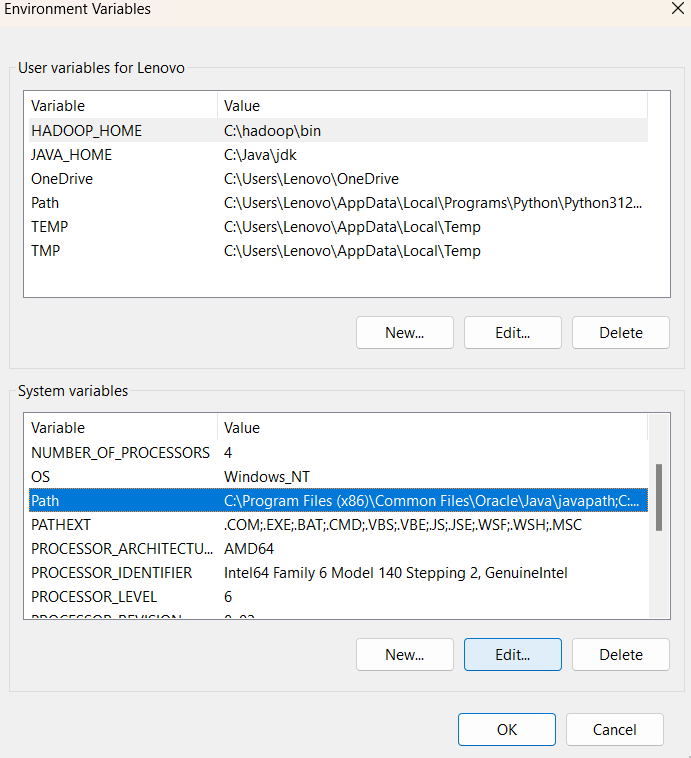
**FOR HADOOP**

* Now as shown, add HADOOP\_HOME in variable name and path of Hadoop folder in Variable Value.
* Click OK and we are half done with setting HADOOP\_HOME.

**Setting Path Variable**

* Last step in setting Environment variable is setting Path in System Variable.

Select Path variable in the system variables and click on Edit



* Now we need to add these paths to Path Variable one by one:-  
  \* %JAVA\_HOME%\bin  
  \* %HADOOP\_HOME%\bin  
  \* %HADOOP\_HOME%\sbin
* Click OK and OK. & we are done with Setting Environment Variables.

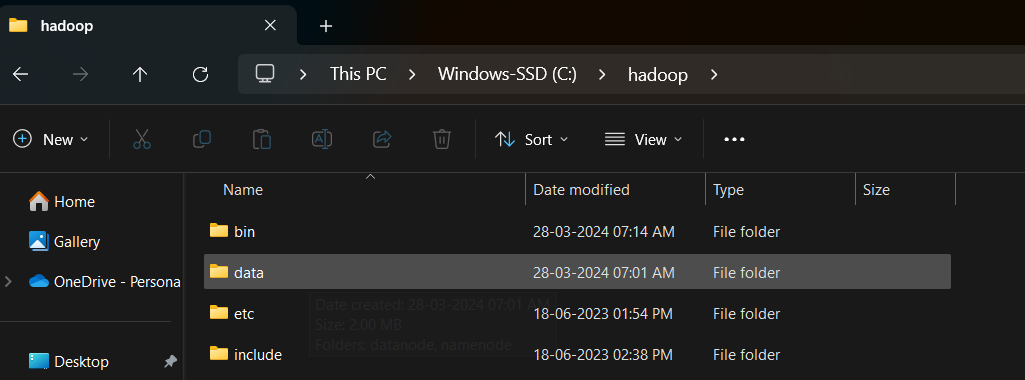
# Editing Hadoop files

Once we have configured the environment variables next step is to configure Hadoop. It has 3 parts:-

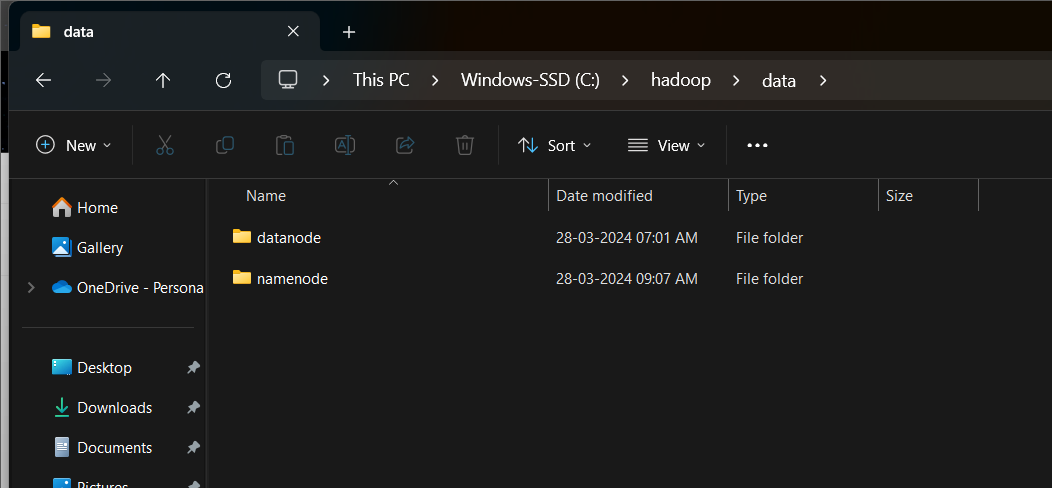
**4.1 Creating Folders**

We need to create a folder data in the hadoop directory, and 2 sub folders namenode and datanode

* Create **DATA folder**in the Hadoop directory



Once DATA folder is created, we need to create 2 new folders namely, namenode and datanode inside the data folder

These folders are important because files on HDFS resides inside the datanode.

**Editing Configuration Files:**

Now we need to edit the following config files in hadoop for configuring it :-

(We can find these files in Hadoop -> etc -> hadoop)

\* core-site.xml

\* hdfs-site.xml

\* mapred-site.xml

\* yarn-site.xml

\* hadoop-env.cmd

4.2.1 Editing core-site.xml

Right click on the file, select edit and paste the following content within <configuration> </configuration> tags.

Note:- Below part already has the configuration tag, we need to copy only the part inside it.

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

4.2.2 Editing hdfs-site.xml

Right click on the file, select edit and paste the following content within <configuration></configuration>tags.

Note:- Below part already has the configuration tag, we need to copy only the part inside it.

Also replace PATH~1 and PATH~2 with the path of namenode and datanode folder that we created recently(step 4.1).

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>PATH~1\namenode</value>

<final>true</final>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>PATH~2\datanode</value>

<final>true</final>

</property>

</configuration>

4.2.3 Editing mapred-site.xml

Right click on the file, select edit and paste the following content within <configuration> </configuration> tags.

Note:- Below part already has the configuration tag, we need to copy only the part inside it.

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

4.2.4 Editing yarn-site.xml

Right click on the file, select edit and paste the following content within <configuration> </configuration> tags.

Note:- Below part already has the configuration tag, we need to copy only the part inside it.

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

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

</configuration>

4.2.5 Verifying hadoop-env.cmd

Right click on the file, select edit and check if the JAVA\_HOME is set correctly or not.

We can replace the JAVA\_HOME variable in the file with your actual JAVA\_HOME that we configured in the System Variable.

set JAVA\_HOME=%JAVA\_HOME%

OR

set JAVA\_HOME="C:\Program Files\Java\jdk1.8.0\_221"

4.3 Replacing bin

Last step in configuring the hadoop is to download and replace the bin folder.

\* Go to this GitHub Repo and download the bin folder as a zip.

\* Extract the zip and copy all the files present under bin folder to %HADOOP\_HOME%\bin

**Testing Setup**

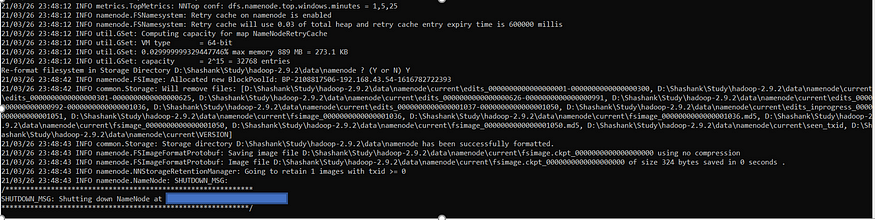
Congratulation..!!!!!  
We are done with the setting up the Hadoop in our System.

Now we need to check if everything works smoothly…

**5.1 Formatting Namenode**

Before starting hadoop we need to format the namenode for this we need to start a NEW Command Prompt and run below command

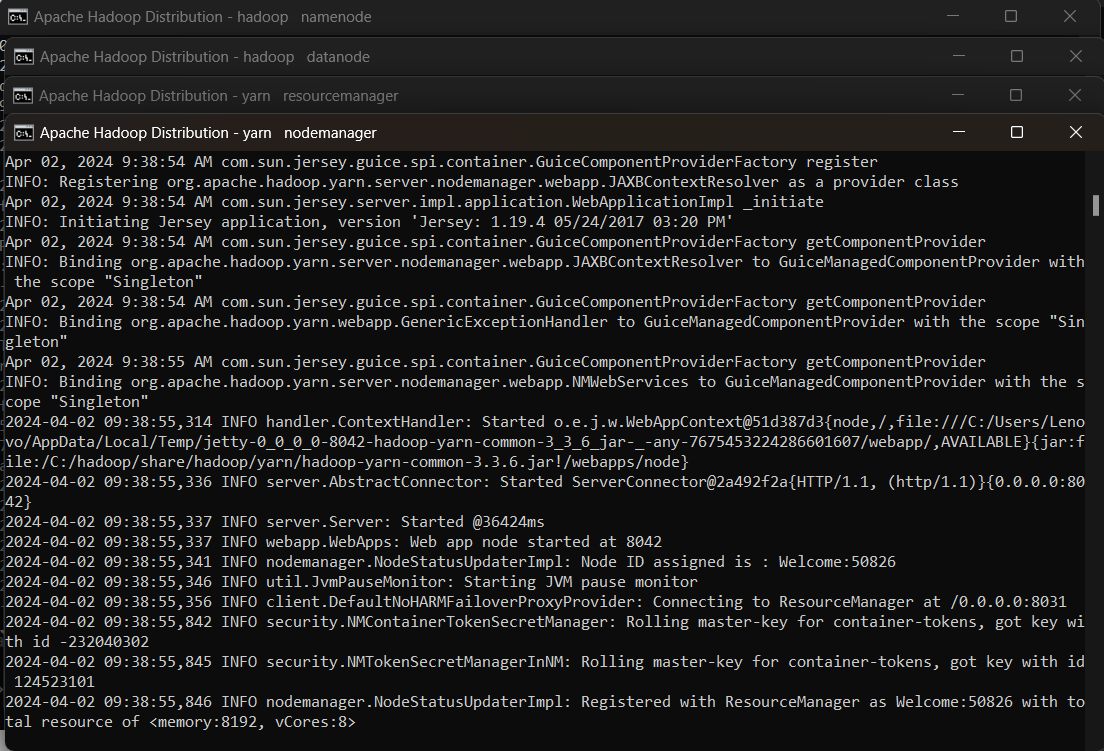
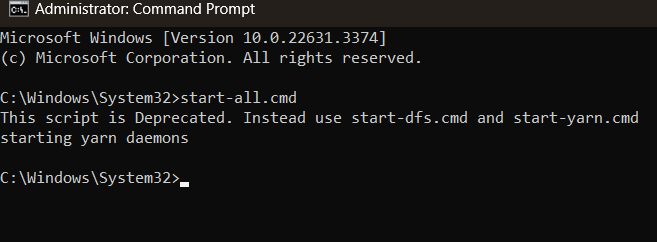
hadoop namenode -format



**Launching Hadoop**

Now we need to start a new Command Prompt remember to run it as administrator to avoid permission issues and execute below commands

**start-all.cmd**



**RUNNING HADOOP ON WEB:**

**Use localhost.9000**

**And localhost.8080**

**DONE!!!!!!!**

**Hadoop Components::**

**Sure, let's delve into each components of the Hadoop ecosystem in detail:**

**1. \*\*Hadoop Distributed File System (HDFS)\*\*:**

**- HDFS is the primary storage system in Hadoop, designed to store large volumes of data reliably and efficiently across a cluster of commodity hardware.**

**- It follows a master-slave architecture, with a single NameNode serving as the master and multiple DataNodes serving as slaves.**

**- The NameNode manages the file system namespace and metadata, while DataNodes store the actual data blocks.**

**- HDFS provides high throughput, fault tolerance, and scalability by replicating data blocks across multiple DataNodes and supporting parallel data access.**

**2. \*\*MapReduce\*\*:**

**- MapReduce is a programming model and processing engine for distributed data processing in Hadoop.**

**- It allows users to write parallelizable data processing tasks in the form of map and reduce functions, which are executed across the cluster.**

**- MapReduce jobs are typically divided into map tasks, which process input data in parallel and produce intermediate key-value pairs, and reduce tasks, which aggregate and process the intermediate results to produce the final output.**

**- MapReduce provides fault tolerance, data locality, and automatic parallelization of tasks, making it suitable for batch processing of large datasets.**

**3. \*\*YARN (Yet Another Resource Negotiator)\*\*:**

**- YARN is the resource management and job scheduling framework in Hadoop, introduced in Hadoop 2.x to address limitations of the original MapReduce framework.**

**- It decouples resource management from job scheduling, allowing multiple data processing engines (e.g., MapReduce, Apache Spark, Apache Flink) to run on the same Hadoop cluster.**

**- YARN consists of a ResourceManager (master) and multiple NodeManagers (slaves), which manage cluster resources and execute application containers, respectively.**

**- It supports dynamic resource allocation, fine-grained resource scheduling, and multi-tenancy, enabling efficient resource utilization and isolation across different workloads.**

**4. \*\*Hadoop Common\*\*:**

**- Hadoop Common is a set of common utilities and libraries shared by other Hadoop components.**

**- It provides core functionality such as filesystem abstractions, I/O operations, network communication, and security features.**

**- Hadoop Common includes libraries like Hadoop Distributed Copy (hadoop-distcp) for distributed data copying, Hadoop RPC for remote procedure calls, and Hadoop Auth for authentication and authorization.**

**5. \*\*Other Projects\*\*:**

**- The Hadoop ecosystem includes various related projects and tools that extend the functionality of Hadoop and provide additional features for data management, analytics, and machine learning.**

**- Some notable projects include Apache Hive for data warehousing and SQL-like querying, Apache Pig for data flow scripting, Apache HBase for real-time NoSQL data storage, Apache Spark for in-memory data processing, Apache Kafka for real-time data streaming, Apache Flume and Apache Sqoop for data ingestion and integration, and many others.**

**Together, these components form a comprehensive platform for building and deploying data pipelines, enabling organizations to store, process, and analyze large volumes of data efficiently and cost-effectively.**

[**https://www.guru99.com/introduction-to-mapreduce.html**](https://www.guru99.com/introduction-to-mapreduce.html)

[**https://www.guru99.com/learn-hadoop-in-10-minutes.html**](https://www.guru99.com/learn-hadoop-in-10-minutes.html)

**Hadoop architecture::**

**Sor for that I refer to   
this medium article where I learned about how hadpoo got intorduce and how it updated with time**

**HADOOP 1X, 2X, 3X.**

[**https://medium.com/@sujathamudadla1213/explain-hadoop-1-2-3-architechture-in-depth-95fd71e13bc6**](https://medium.com/@sujathamudadla1213/explain-hadoop-1-2-3-architechture-in-depth-95fd71e13bc6)

**Create a small file on your machine. Put that on HDFS. Check the number of blocks created.**

* 1. **Create a Small File:**

**Create a small text file on your local machine using any text editor. For example, you can create a file named sample.txt with some sample content.**

* 1. **Put the File on HDFS:**

**Use the hadoop fs -put command to copy the file from your local file system to HDFS. You need to specify the source file (local path) and the destination path in HDFS.**

**hadoop fs -put /path/to/local/sample.txt /user/your\_username/sample.txt**

**Replace /path/to/local/sample.txt with the path to your local file and your\_username with your Hadoop username.**

* 1. **Check the Number of Blocks Created:**

**You can use the hadoop fsck command to check the number of blocks created for the file in HDFS.**

**hdfs fsck /user/your\_username/sample.txt -files -blocks**

**hdfs fsck /user/your\_username/sample.txt -files -blocks**

**This command will display detailed information about the file, including the number of blocks created.**

**Alternatively, you can use the Hadoop web interface to check the number of blocks. Open your web browser and navigate to the HDFS NameNode web interface (http://localhost:9870). Then, navigate to the file browser section and find your file (/user/your\_username/sample.txt). It should display the number of blocks associated with the file.**

**That's it! You've created a small file on your local machine, put it on HDFS, and checked the number of blocks created for the file in HDFS.**

Get a bigger file (1 GB plus). Put that on HDFS. Check the blocks created again. It should be in parts of 128 MB.

1. Set the Block Size in Hadoop Configuration:

You'll need to set the desired block size in your Hadoop configuration. This involves modifying the hdfs-site.xml file, specifically the dfs.blocksize property. You can set this property to 128 MB (or 128m in Hadoop's configuration syntax). The hdfs-site.xml file is typically located in the etc/hadoop directory of your Hadoop installation.

Here's an example of how you can set the block size in hdfs-site.xml:

**<property>**

**<name>dfs.blocksize</name>**

**<value>128m</value>**

**</property>**

1. **Restart HDFS:** After modifying the configuration file, restart HDFS for the changes to take effect. You can do this by running the following command:

**stop-dfs.sh**

**start-dfs.sh**

1. **Put the File with the New Block Size:** Finally, upload the file to HDFS again, and it will use the new block size:

**hdfs dfs -put "C:/Users/Lenovo/PYTHON/Lytx-CFP-DataEngg-AWS/themovie.mkv" /user/movie/themovie.mkv**

**WORDCOUNT PROBLEM:**

* **First create a text file in local machine**
* **Then** Create a directory in HDFS, where to kept text file.  
  $ hdfs dfs -mkdir /test
* Upload the data.txt file on HDFS in the specific directory.  
  $ hdfs dfs -put /C/data.txt /test
* Then reat that file and open it into browser using Hadoop
* Then apply mapreduce on it and peform tasks
* Then see how map reduce work on that file and display it on to browser

WORD COUNT USING PYTHON :

MAPPER CODE :

#!/usr/bin/env python

import sys

for line in sys.stdin:

line = line.strip()

words = line.split()

for word in words:

print ('%s\t%s' % (word, 1))

REDUCER CODE:

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word, count = line.split('\t', 1)

count = int(count)

if current\_word == word:

current\_count += count

else:

if current\_word:

print ('%s\t%s' % (current\_word, current\_count))

current\_count = count

current\_word = word

if current\_word == word:

print('%s\t%s' % (current\_word, current\_count))

WordCount program using Hadoop Streaming:

**hadoop jar %HADOOP\_HOME%\share\hadoop\tools\lib\hadoop-streaming-\*.jar ^**

**-file "C:\Users\Lenovo\PYTHON\Lytx-CFP-DataEngg-AWS\hadoop\mapper.py" -mapper "C:\Users\Lenovo\PYTHON\Lytx-CFP-DataEngg-AWS\hadoop\mapper.py" ^**

**-file "C:\Users\Lenovo\PYTHON\Lytx-CFP-DataEngg-AWS\hadoop\reducer.py" -reducer "C:\Users\Lenovo\PYTHON\Lytx-CFP-DataEngg-AWS\hadoop\reducer.py" ^**

**-input /user/your\_username/input\_file.txt -output /user/your\_username/output\_directory**