Hadoop Windows Setup

First check if wsl is available or not using - `wsl --status`

if not enable it in windows features and prompt `wsl --install` in terminal

It will automaticall download lated Ubuntu.

Now open and follow these:

Open Ubuntu

```
akhira@AKHIRA-PC:~$ cd //
akhira@AKHIRA-PC://$ ls
bin                boot  etc   init  lib.usr-is-merged  lost+found  mnt  proc  run   sbin.usr-is-merged  srv  tmp  var
bin.usr-is-merged  dev   home  lib   lib64              media       opt  root  sbin  snap                sys  usr
```

Phase 1: Prerequisites and Setup

Step 1: Update your System

Why: We need to make sure your Ubuntu package list is up to date so we download the correct versions of software. **Consequence:** If skipped, you might download broken or obsolete dependencies.

Run this command:

```
sudo apt update && sudo apt upgrade -y
```

Step 2: Install Java (OpenJDK 8)

Why: Hadoop is written in Java. It strictly requires the Java Development Kit (JDK) to run. While newer versions of Java exist, **OpenJDK 8** is the industry standard for Hadoop stability. **Consequence:** If you use a different version (like Java 17 or 21), Hadoop scripts may crash or throw syntax errors.

```
sudo apt install openjdk-8-jdk -y
```

Verify the installation:

```
java -version
```

You should see output mentioning `openjdk version "1.8.0...`"`._

```
akhira@AKHIRA-PC://$ java -version
openjdk version "1.8.0_472"
OpenJDK Runtime Environment (build 1.8.0_472-8u472-ga-1~24.04-b08)
OpenJDK 64-Bit Server VM (build 25.472-b08, mixed mode)
```

Step 3: Install and Configure SSH

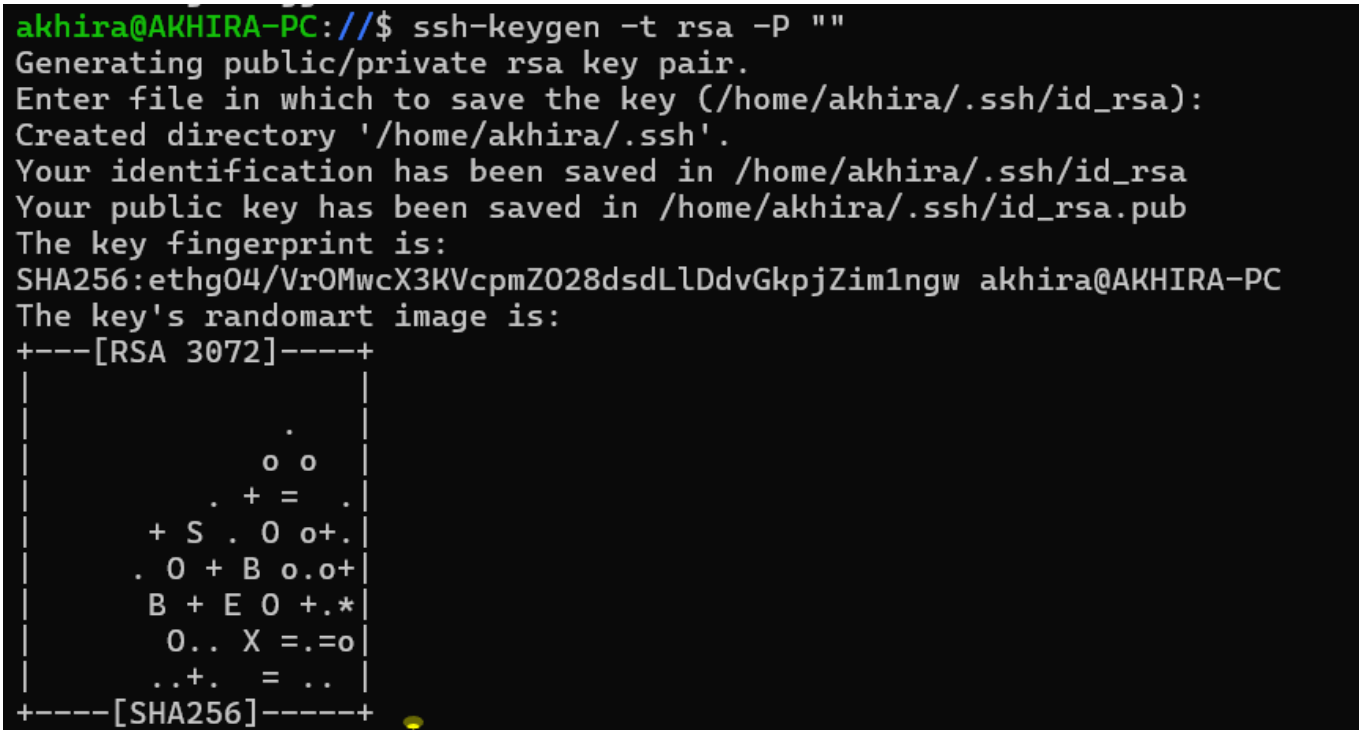
Why: Hadoop requires SSH (Secure Shell) to manage its nodes. Even though you are on one machine, the "NameNode" (manager) uses SSH to talk to the "DataNode" (worker). **Consequence:** If SSH is not configured, Hadoop will fail to start with "Connection Refused" errors.

1. **Install SSH Server:**

```
sudo apt install openssh-server -y
```

2. **Generate an SSH Key:** (Press **Enter** for all prompts to accept defaults and no password).

```
ssh-keygen -t rsa -P ""
```



3. **Authorize the Key:** We are adding your own key to your own "authorized list" so you can connect to yourself without typing a password every time.

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

4. **Start the SSH Service (Specific to WSL):** In standard Linux, this starts automatically. In WSL, you often have to start it manually.

```
sudo service ssh start
```

5. **Test it:**

```
ssh localhost
```

If it asks "Are you sure...?", type `yes`. If you get a command prompt without being asked for a password, it works! Type `exit` to close the SSH connection.

```
akhira@AKHIRA-PC://$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
akhira@AKHIRA-PC://$ sudo service ssh start
akhira@AKHIRA-PC://$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ED25519 key fingerprint is SHA256:ZIl4j8kzKcZu9s2v4FVl68gMKaPnfwflB5PQOzsHwDA.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.6.87.2-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Wed Nov 26 15:04:44 UTC 2025

  System load:  0.06                Processes:             32
  Usage of /:   0.2% of 1006.85GB   Users logged in:       1
  Memory usage: 5%                  IPv4 address for eth0: 172.23.137.198
  Swap usage:   0%
```

Phase 2: Downloading and Installing Hadoop

Step 4: Download Hadoop

Why: We need the actual software files. We will download Hadoop version 3.3.6 (stable).

```
sudo wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
```

```
akhira@AKHIRA-PC://usr/local$ sudo wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
[sudo] password for akhira:
--2025-11-26 15:08:55--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 135.181.214.104, 88.99.208.237, 2a01:4f9:3a:2c57::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|135.181.214.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: ‘hadoop-3.3.6.tar.gz’

hadoop-3.3.6.tar.gz                 100%[=============================================================================>] 696.28M  1.59MB/s    in 10m 14s

2025-11-26 15:19:11 (1.13 MB/s) - ‘hadoop-3.3.6.tar.gz’ saved [730107476/730107476]
```

Step 5: Extract the Files

Why: The file is a compressed "tarball" (like a .zip). We need to unzip it. **Consequence:** You cannot run the program from inside the compressed file.

```
sudo tar -xzvf hadoop-3.3.6.tar.gz
```

Now, let's move it to a cleaner location (optional but recommended):

```
sudo mv hadoop-3.3.6 /usr/local/hadoop
```

```
akhira@AKHIRA-PC://usr/local$ sudo mv hadoop-3.3.6 /usr/local/hadoop
```

Step 6: Configure Environment Variables (Crucial Step)

Why: We need to tell Ubuntu where to find the `hadoop` commands and, importantly, tell Hadoop exactly where Java lives. This is where we ensure **Anaconda doesn't interfere.**

1. **Find your Java Path:** Run this command and copy the output:

```
readlink -f /usr/bin/java | sed "s:bin/java::"
```

```
akhira@AKHIRA-PC://usr/local$ readlink -f /usr/bin/java | sed "s:bin/java::"
/usr/lib/jvm/java-8-openjdk-amd64/jre/
```

It usually looks like: `/usr/lib/jvm/java-8-openjdk-amd64/jre/`

2. **Open your bash configuration:**

```
nano ~/.bashrc
```

3. **Add the following to the BOTTOM of the file:** (Copy and paste this block. **Note:** Ensure `JAVA_HOME` matches the path you found above, usually removing the trailing `/jre/` part if you want the full JDK, but for now, let's stick to the OpenJDK 8 default location).

```
# Hadoop Variables
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
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
```

4.
    1. **Save and Exit:** Press `CTRL+X`, `y`, then `Enter`.
5. **Apply Changes:**

```
source ~/.bashrc
```

Phase 3: Configuration Files (The Complex Part)

We need to edit 4 XML files located in `/usr/local/hadoop/etc/hadoop/`. **Why:** This tells Hadoop "You are running on one machine" (Pseudo-distributed) rather than a real cluster.

File 1: hadoop-env.sh

Why: To hardcode the Java path so Hadoop never gets confused by Anaconda.

1. Open the file:

```
sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

2. Find the line that says `export JAVA_HOME=` (it might be commented out with a `#`). Change it to:

```
###
# Generic settings for HADOOP
###

# Technically, the only required environment variable is JAVA_HOME.
# All others are optional.  However, the defaults are probably not
# preferred.  Many sites configure these options outside of Hadoop,
# such as in /etc/profile.d

# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
# export JAVA_HOME=
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
###
# Generic settings for HADOOP
###

# Technically, the only required environment variable is JAVA_HOME.
# All others are optional.  However, the defaults are probably not
# preferred.  Many sites configure these options outside of Hadoop,
# such as in /etc/profile.d

# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

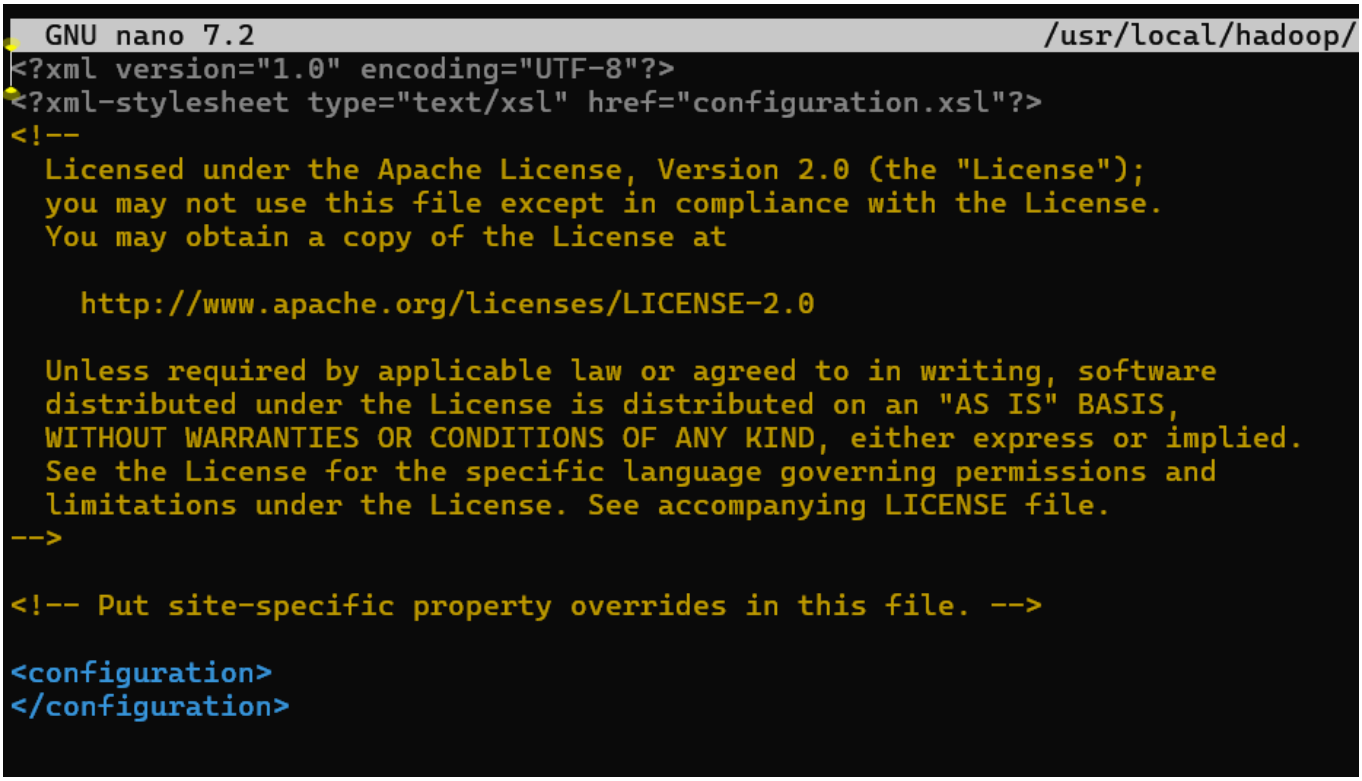
(Save : Ctrl + X, y, Enter)

File 2: core-site.xml

Why: Tells Hadoop where the "NameNode" (the master) runs.

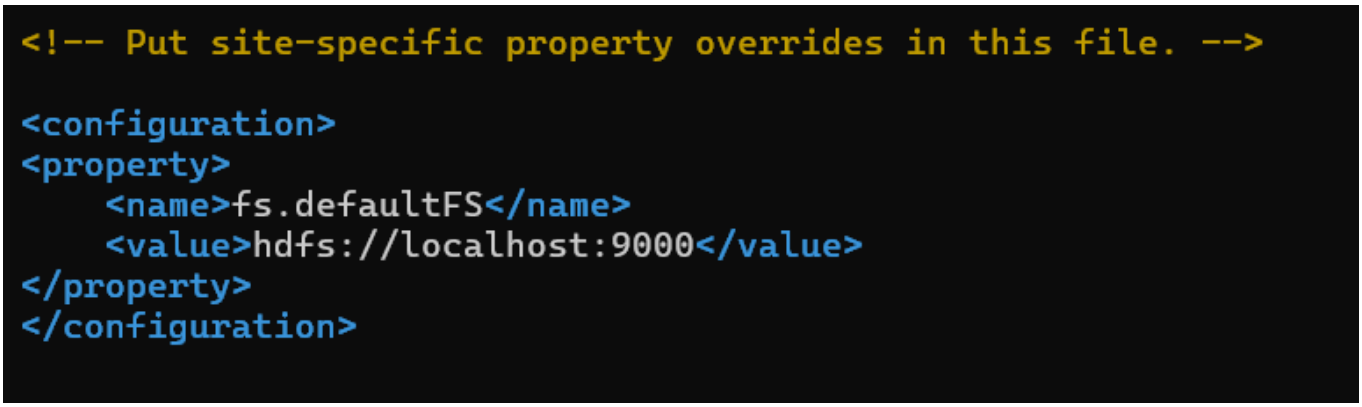
1. Open the file:

```
sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```



2. Paste this **between** the `<configuration>` and `</configuration>` tags:

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



File 3: hdfs-site.xml

Why: Configuration for the Hadoop Distributed File System (HDFS). We set replication to 1 because we only have 1 machine. If we default to 3 (standard), Hadoop will crash looking for 2 other missing machines.

1. Open the file:

```
sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

2. Paste this between the configuration tags:

```
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>/usr/local/hadoop/data/nameNode</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>/usr/local/hadoop/data/dataNode</value>
</property>
```

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

<configuration>
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>/usr/local/hadoop/data/nameNode</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>/usr/local/hadoop/data/dataNode</value>
</property>
</configuration>
```

Important: We pointed to directories that don't exist yet. Let's create them:

```
sudo mkdir -p /usr/local/hadoop/data/nameNode
```

```
sudo mkdir -p /usr/local/hadoop/data/dataNode
```

```
sudo chown -R $USER:$USER /usr/local/hadoop
```

```
akhira@AKHIRA-PC://usr/local$ sudo mkdir -p /usr/local/hadoop/data/nameNode
akhira@AKHIRA-PC://usr/local$ sudo mkdir -p /usr/local/hadoop/data/dataNode
akhira@AKHIRA-PC://usr/local$ sudo chown -R $USER:$USER /usr/local/hadoop
```

File 4: mapred-site.xml

Why: Tells Hadoop we are using YARN (Yet Another Resource Negotiator) to manage jobs.

1. Open the file:

```
sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

2. Paste between configuration tags:

```
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
</property>
```

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

<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
</property>
</configuration>
```

File 5: yarn-site.xml

Why: Configures the NodeManager.

1. Open the file:

```
sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

```
<configuration>

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

</configuration>
```

2. Paste between configuration tags:

```
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
```

```
<configuration>

<!-- Site specific YARN configuration properties -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
</configuration>
```

Phase 4: Formatting and Starting

Step 7: Format the NameNode

Why: This initializes the file system. Think of this like formatting a new USB stick before you can save files to it. **Consequence:** If you do this on a system that already has data, **it deletes everything**. Since this is a fresh install, do it once. **Do not do this repeatedly in the future or you will lose data.**

```
hdfs namenode -format
```

Look for a message near the end saying `Storage directory ... has been successfully formatted.`

Step 8: Start Hadoop

Why: We need to start the "daemons" (background processes).

1. **Start DFS (Distributed File System):**

```
start-dfs.sh
```

```
akhira@AKHIRA-PC://usr/local$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [AKHIRA-PC]
AKHIRA-PC: Warning: Permanently added 'akhira-pc' (ED25519) to the list of known hosts.
2025-11-26 15:33:35,565 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

2. **Start YARN (Resource Manager):**

```
start-yarn.sh
```

```
akhira@AKHIRA-PC://usr/local$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```

(If you get errors about "pdsh", run: `sudo apt install pdsh` and reload).

Step 9: Verify Installation

Why: To check if the processes are actually running.

Type:

```
jps
```

Desired Output: You should see a list like this (numbers will vary):

```
1234 NameNode
2345 DataNode
3456 ResourceManager
4567 NodeManager
5678 SecondaryNameNode
6789 Jps
```

```
akhira@AKHIRA-PC://usr/local$ jps
6528 NodeManager
6147 SecondaryNameNode
5796 NameNode
6900 Jps
6389 ResourceManager
5919 DataNode
```

If you see all of these, congratulations! Hadoop is running on your WSL2 machine.

Troubleshooting & Changes while Executing

1. **SSH Issue:** Since you are on WSL, if you restart your computer, the SSH service stops. Before running `start-dfs.sh` next time, always run `sudo service ssh start` first.
2. **Permission Denied:** If you get permission errors, ensure you ran the `chown` command in Step 6 (under hdfs-site.xml section) correctly.
3. **Accessing the UI:** You can now open your Windows Browser (Chrome/Edge) and go to: `http://localhost:9870` to see the Hadoop Web Interface.

Phase 1: Download and Install Software

Pig Setup

This phase downloads Apache Pig and Apache Hive binaries from official Apache repositories and extracts them to system directories.

Key Decision: We use Hive 3.1.3 (from Archive) instead of the latest Hive 4.0 because version 4.0 removes MapReduce support, which is essential for your current environment.

1.1 Download Apache Pig Binary

```
sudo wget -c https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
```

- -c flag resumes interrupted downloads
- Downloads 0.17.0 stable release from Apache's main repository

1.2 Extract and Move to System Directory

```
sudo tar -xzvf pig-0.17.0.tar.gz
```

```
sudo mv pig-0.17.0 /usr/local/pig
```

- tar -xzvf extracts the compressed archive
- Moves to /usr/local/pig for system-wide access
- Requires sudo privileges for /usr/local/ directory

Hive Setup

2.1 Download Apache Hive Binary from Archive

```
sudo wget https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz
```

- Uses Archive link instead of current releases
- Hive 3.1.3 maintains full MapReduce compatibility
- Latest Hive versions remove MapReduce, causing incompatibility with WSL2 environments

2.2 Extract and Move to System Directory

```
sudo tar -xzvf apache-hive-3.1.3-bin.tar.gz
```

```
sudo mv apache-hive-3.1.3-bin /usr/local/hive
```

- Creates /usr/local/hive installation directory
- Ensures both tools are in standard system locations for easy PATH management

If not working, then

Option 2: The "Windows" Method (The 100% Fix)

Since you are on WSL, your Ubuntu and Windows systems share files. We can download it using your Windows Browser (Chrome/Edge) and simply move it into Ubuntu.

Step 1: Download in Windows Click this link to download the file to your Windows **Downloads** folder: https://archive.apache.org/dist/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz

Step 2: Copy it into Ubuntu Once the download finishes, run this command in your Ubuntu terminal. *(Replace `YourWindowsUsername` with your actual Windows user name. If you aren't sure, type `ls /mnt/c/Users/` to see the list).*

```
sudo cp /mnt/c/Users/YourWindowsUsername/Downloads/apache-hive-3.1.3-bin.tar.gz /usr/local/
```

Step 3: Verify Check if the file arrived:

```
ls /usr/local/apache-hive-3.1.3-bin.tar.gz
```


```
sudo tar -xzvf apache-hive-3.1.3-bin.tar.gz
```

```
sudo mv apache-hive-3.1.3-bin hive
```

Phase 2: Environment Variables

Ubuntu needs to know the installation paths of Pig and Hive. This section configures your shell environment to automatically locate these tools and their dependencies

3.1 Open the Configuration File

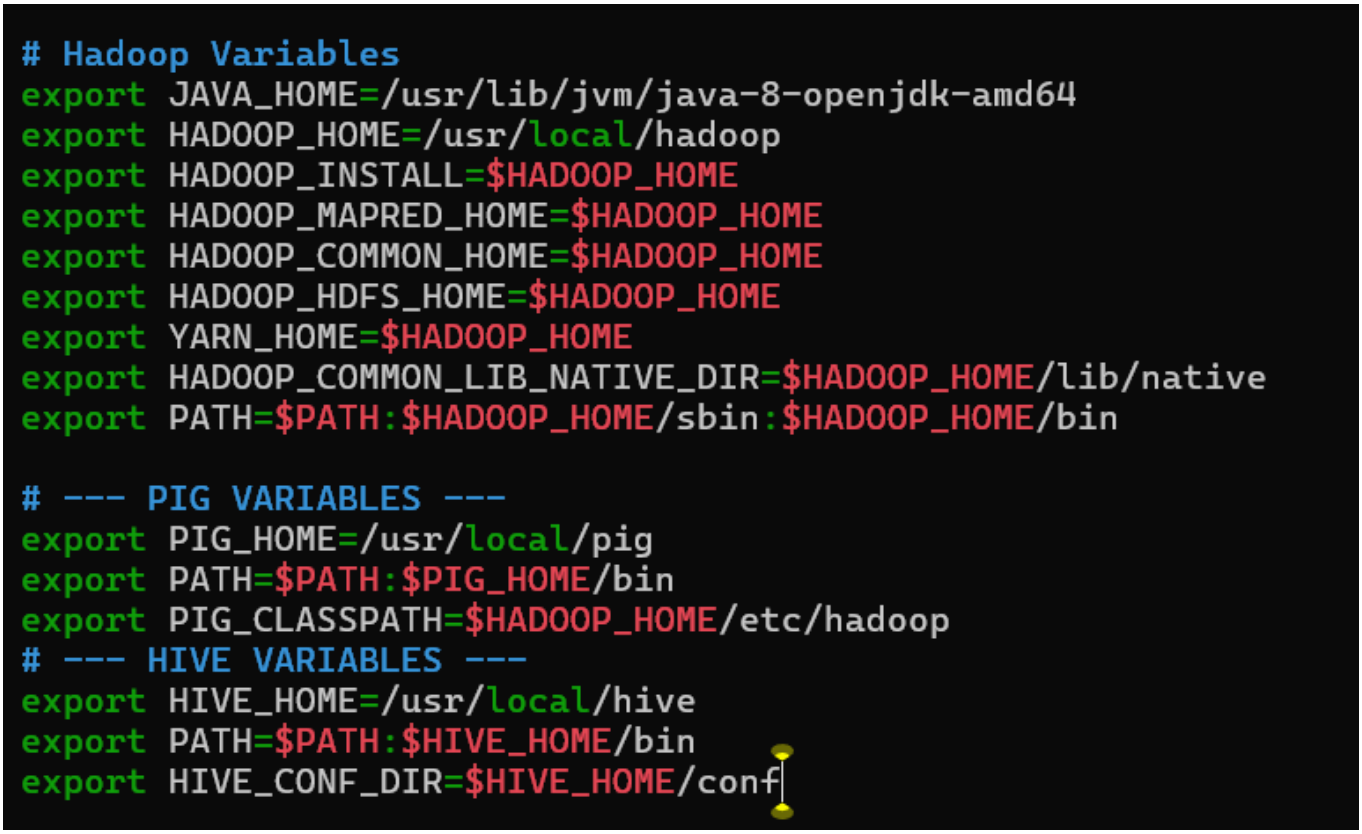
```
nano ~/.bashrc
```

- ~/.bashrc is the shell configuration file loaded on every terminal session
- nano opens a simple text editor
- Changes here persist across sessions

3.2 Add Environment Variables

Append these lines to the end of ~/.bashrc:

```
# --- PIG VARIABLES ---
export PIG_HOME=/usr/local/pig
export PATH=$PATH:$PIG_HOME/bin
export PIG_CLASSPATH=$HADOOP_HOME/etc/hadoop
# --- HIVE VARIABLES ---
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export HIVE_CONF_DIR=$HIVE_HOME/conf
```



3.3 Apply Changes Immediately

```
source ~/.bashrc
```

- source reloads the configuration file without restarting the terminal

- Changes take effect immediately in the current session

```
echo $PIG_HOME
echo $HIVE_HOME
```

Both should return their respective installation paths.

Phase 3: Critical Configuration Fixes

Problem

Hive 3.1.3 bundles Guava 19.0, but Hadoop 3.3.6 requires Guava 27.0. This version mismatch causes:

- ClassNotFoundException crashes
- Runtime library conflicts
- Metastore initialization failures

Solution

```
rm $HIVE_HOME/lib/guava-19.0.jar
```

```
cp $HADOOP_HOME/share/hadoop/common/lib/guava-27.0-jre.jar $HIVE_HOME/lib/
```

- Removes the conflicting older Guava version from Hive
- Copies the correct Guava version from Hadoop's library
- Ensures both systems use the same library version

Verification

```
ls -la $HIVE_HOME/lib/guava*.jar
```

Should show guava-27.0-jre.jar only

Configure hive-site.xml

problem

Hive needs to know where to store metadata. Without this configuration, it cannot initialize the metastore.

Solution: Create and Configure the File

5.1 Create the Configuration File

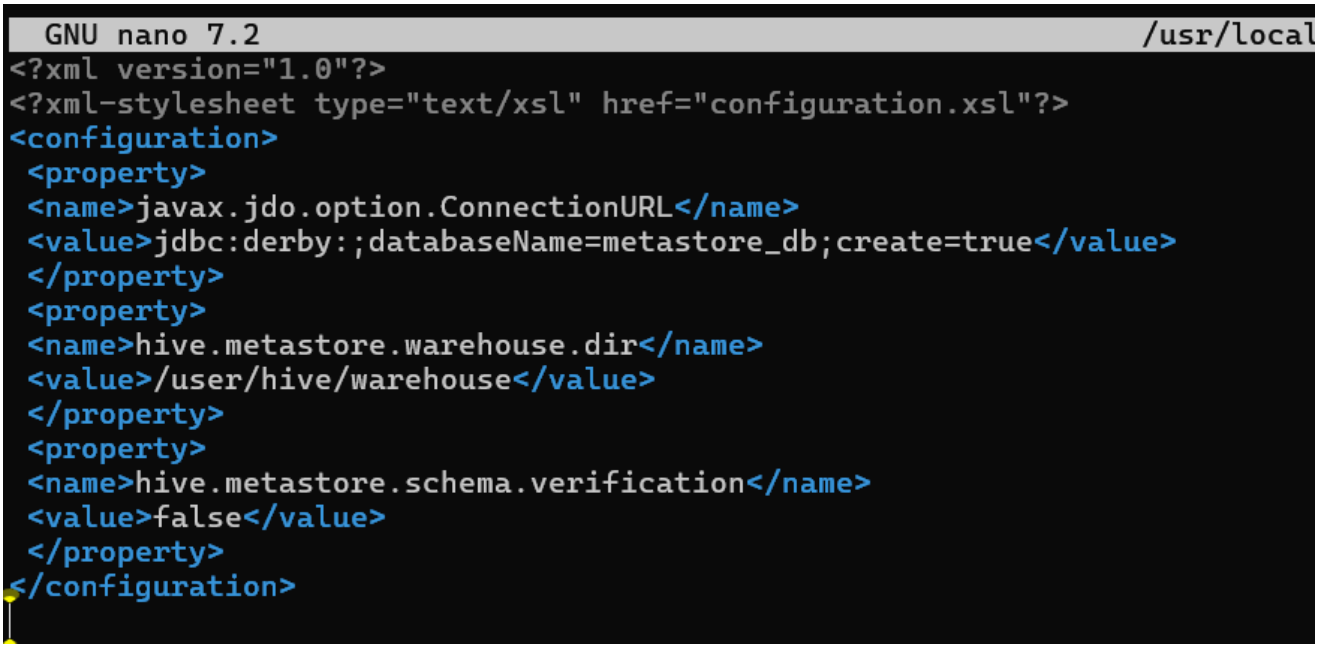
```
nano $HIVE_HOME/conf/hive-site.xml
```

5.2 Insert Configuration Content

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
 <property>
 <name>javax.jdo.option.ConnectionURL</name>
 <value>jdbc:derby:;databaseName=metastore_db;create=true</value>
 </property>
 <property>
 <name>hive.metastore.warehouse.dir</name>
 <value>/user/hive/warehouse</value>
 </property>
 <property>
 <name>hive.metastore.schema.verification</name>
 <value>false</value>
 </property>
</configuration>
```

- Derby is an embedded SQL database suitable for local Hive installations
- Creates metastore database in your home directory

- Simplifies setup without requiring external database servers



Configure yarn-site.xml (The WSL2 Fix)

Problem

WSL2 environments have limited memory reporting. Without these fixes:

- "Job Failed" errors appear without root cause
- Virtual memory checks cause unnecessary failures
- YARN cannot locate required libraries (ClassNotFoundException)

Solution: Update YARN Configuration

6.1 Retrieve Your Hadoop Classpath

Run this command and copy the entire output string:

```
hadoop classpath
```

6.2 Edit yarn-site.xml

```
nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

6.3 Add/Update These Properties

Insert these inside the `<configuration>` tags:

```
<property>
 <name>yarn.nodemanager.vmem-check-enabled</name>
 <value>false</value>
</property>
<property>
 <name>yarn.nodemanager.pmem-check-enabled</name>
 <value>false</value>
</property>
<property>
 <name>yarn.application.classpath</name>
 <value>PASTE_YOUR_LONG_CLASSPATH_STRING_HERE</value>
</property>
```

```
<configuration>

<!-- Site specific YARN configuration properties -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
 <name>yarn.nodemanager.vmem-check-enabled</name>
 <value>false</value>
</property>
<property>
 <name>yarn.nodemanager.pmem-check-enabled</name>
 <value>false</value>
</property>
<property>
 <name>yarn.application.classpath</name>
 <value>PASTE_YOUR_LONG_CLASSPATH_STRING_HERE</value>
</property>
</configuration>
```

Configure mapred-site.xml

Problem

MapReduce workers don't know where to find Hadoop binaries, causing:

- "HADOOP_MAPRED_HOME not found" errors
- Map/Reduce task failures
- Container launch failures

Solution

7.1 Edit Configuration File

```
nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

7.2 Add These Properties

Insert inside the `<configuration>` tags:

```
<property>
 <name>yarn.app.mapreduce.am.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
```

```
</property>
<property>
 <name>mapreduce.map.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
 <name>mapreduce.reduce.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
```

- All three specify the same Hadoop path: /usr/local/hadoop
- Ensures consistency across all MapReduce processes
- Each process inherits these environment variables at startup

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

<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
</property>
<property>
 <name>yarn.app.mapreduce.am.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
 <name>mapreduce.map.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
 <name>mapreduce.reduce.env</name>
 <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
</configuration>
```

Phase 4: Initialization

This phase starts Hadoop services and initializes the Hive metastore database. These steps must be executed in order

Step 8: Start Hadoop Services

```
start-dfs.sh
```

```
start-yarn.sh
```

- start-dfs.sh starts the Distributed File System (NameNode and DataNode)
- start-yarn.sh starts the resource manager and node managers
- Hadoop must be running before creating HDFS directories

Verification

```
jps
```

Expected Outcome

```
1234 NameNode
5678 DataNode
9012 ResourceManager
3456 NodeManager
7890 Jps
```

All five processes should appear.

Step 9: Create HDFS Directories

HDFS directories store Hive data and temporary files. These must exist with proper permissions before Hive can operate.

9.1 Create Hive Warehouse Directory

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

- -p creates parent directories if they don't exist
- /user/hive/warehouse is the standard location for Hive table data

9.2 Create Temporary Directory

```
hdfs dfs -mkdir -p /tmp
```

- /tmp is used by Hive and Pig for intermediate processing
- Must be writable by all users

9.3 Set Write Permissions on Warehouse

```
hdfs dfs -chmod g+w /user/hive/warehouse
```

- g+w grants write permission to the group
- Allows Hive metastore to write table data

9.4 Set Full Permissions on /tmp

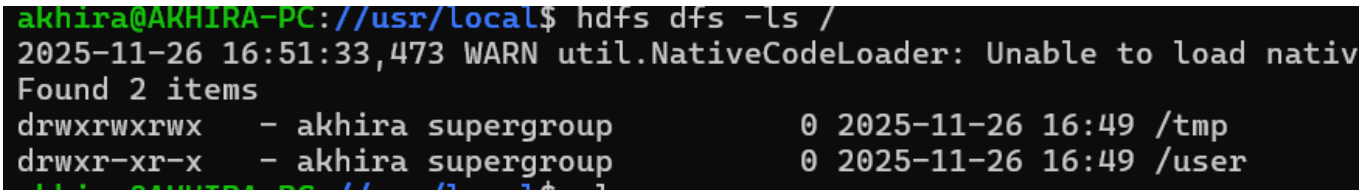
```
hdfs dfs -chmod -R 777 /tmp
```

- -R recursively applies to all subdirectories
- 777 grants read/write/execute to owner, group, and others
- Prevents "Permission Denied" errors during job execution

Verification

```
hdfs dfs -ls /
```

Should show both /user and /tmp directories.



Step 10: Initialize Hive Metastore

Run this command from your home directory (not from /usr/local/hive or any other location).

```
cd ~
```

10.1 Clear Any Corrupted Previous Database

```
rm -rf metastore_db
```

- Removes any existing metastore database that might be corrupted
- Derby creates a new clean database on initialization
- Safe to run even if the directory doesn't exist

10.2 Initialize Hive Schema

```
schematool -dbType derby -initSchema
```

- schematool is a Hive utility that manages database schemas

- -dbType derby specifies using Derby (embedded database)
- -initSchema creates the initial schema for Hive

```
akhira@AKHIRA-PC://usr/local$ cd //
akhira@AKHIRA-PC://$ cd ~
akhira@AKHIRA-PC:~$ rm -rf metastore_db
akhira@AKHIRA-PC:~$ schematool -dbType derby -initSchema
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Metastore connection URL:        jdbc:derby:;databaseName=metastore_db;create=true
Metastore Connection Driver :    org.apache.derby.jdbc.EmbeddedDriver
Metastore connection User:       APP
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.derby.sql
Initialization script completed
schemaTool completed
akhira@AKHIRA-PC:~$
```

If you see errors like "Database already exists":

- Run rm -rf metastore_db again
- Try schematool -dbType derby -initSchema again

Verification:

```
ls -la metastore_db
```

Should show a directory containing Derby database files.