PSNA COLLEGE OF ENGINEERING AND TECHNOLOGY
KOTHANDARAMAN NAGAR, DINDIGUL – 624622 TAMILNADU.

RECORD NOTE BOOK

Reg.No. | | | | | | | | | | | |

Certify that this is the bonafide record of work done by
Mr./Ms. ____
Of the ________________ Semester _____________________________
Branch during the year ____________________________________ in the
______________________________________ Laboratory.

Staff-in-charge   Head of the Department

____

Submitted for the university practical Examination on_______________20

Internal Examiner   External Examiner

INDEX

S. No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks

Ex. No.: 1 Downloading and installing Hadoop; Understanding different Hadoop modes,
Date: Startup scripts, Configuration files.

Aim:

To download and install hadoop and to explore Hadoop's modes, startup scripts, and configuration files for effective management and operation.

Introduction:

Hadoop is a free, open-source, and Java-based software framework used for the storage and processing of large datasets on clusters of machines. It uses HDFS to store its data and process these data using MapReduce. It is an ecosystem of Big Data tools that are primarily used for data mining and machine learning. Apache Hadoop 3.3 comes with noticeable improvements and many bug fixes over the previous releases. It has four major components such as Hadoop Common, HDFS, YARN, and MapReduce.

Different Modes of Hadoop Installation:

Local (Standalone) Mode: In this mode, Hadoop is installed and run on a single machine, and it operates without the Hadoop Distributed File System (HDFS). It is primarily used for testing and development purposes. In Local mode, data is processed directly from the local file system, and there is no distribution of data or computation across multiple nodes. It is the simplest mode to set up and work with since it doesn't involve the complexities of distributed processing.

Pseudo-Distributed Mode: In this mode, Hadoop is installed on a single machine, but it simulates a fully distributed environment by running each Hadoop component (such as Name Node, Data Node, Resource Manager, and Node Manager) in separate Java processes. HDFS is used in this mode, allowing you to store and process data across multiple directories on the same machine, replicating data as specified in the configuration files. Pseudo-distributed mode is often used for testing and development when a small-scale, multi-node environment is required.

Fully-Distributed Mode: In this mode, Hadoop is installed on a cluster of machines, each serving a specific role in the Hadoop ecosystem. The HDFS data is distributed across multiple nodes in the cluster to ensure fault tolerance and scalability. Data processing tasks are distributed across the cluster to achieve parallel processing and high performance. Fully-distributed mode is used in production environments where large-scale data processing and storage are necessary.

Single Node Hadoop Cluster Installation Procedure:

Step 1 – Installing Java:

Hadoop is written in Java and supports only Java version 8. Hadoop version 3.3 and the latest also support Java 11 runtime as well as Java 8. The commands given below are used to install packages on Linux systems that use the Advanced Package Tool (APT). OpenJDK 11 can be installed from the default apt repositories:

```
$ sudo apt update

$ sudo apt install openjdk-11-jdk
```

Once installed, verify the installed version of Java with the following command:

```
$ java -version
```

The following output can be shown:

```
openjdk version "11.0.11" 2021-04-20

OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04)

OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode, sharing)
```

You can find the location of the JAVA_HOME directory by running the following command. Java Location will be required later in this installation.

```
$ dirname $(dirname $(readlink -f $(which java)))
```

Step 2 – Create a Hadoop User:

It is a good idea to create a separate user to run Hadoop for security reasons. Run the following command to create a new user with name hadoop:

```
$ sudo adduser hadoop
```

Provide and confirm the new password as shown below:

```
Adding user `hadoop' ...

Adding new group `hadoop' (1002) ...

Adding new user `hadoop' (1002) with group `hadoop' ...

Creating home directory `/home/hadoop' ...

Copying files from `/etc/skel' ...

New password:

Retype new password:

passwd: password updated successfully

Changing the user information for hadoop

Enter the new value, or press ENTER for the default

        Full Name []:

        Room Number []:

        Work Phone []:

        Home Phone []:

        Other []:

Is the information correct? [Y/n] y
```

Step 3- Configure SSH Key based Authentication:

When you install Hadoop on a distributed environment, such as setting up a Hadoop cluster with multiple nodes, you will need to establish password less SSH communication between the nodes. This means that you won't need to enter a password each time you establish an SSH connection, making the communication between nodes more efficient and suitable for automated processes.

Next, you will need to configure password less SSH authentication for the local system.

First, change the user to Hadoop with the following command:

```
$ su - hadoop
```

Next, run the following command to generate Public and Private Key Pairs where -t option is used to specify the type of encryption algorithm used to generate the SSH key pair.

```
$ ssh-keygen -t rsa
```

You will be asked to enter the filename. Just press Enter to complete the process:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:QSa2syeISwP0hD+UXxxi0j9MSOrjKDGIbkfbM3ejyIk hadoop@ubuntu20
The key's randomart image is:
+---[RSA 3072]----+
| ..o++=.+        |
|..oo++.O         |
|. oo. B .        |
|o..+ o * .       |
|= ++o o S        |
|.++o+  o         |
|.+.+ + . o       |
|o . o * o .      |
|   E + .         |
+----[SHA256]-----+
```

Next, append the generated public keys from id_rsa.pub to authorized_keys and set proper permission: The following command is used to add the public key of an SSH key pair to the authorized_keys file on a remote server. Once your public key is added to the authorized_keys file on the server, you can log in without a password. When you attempt to SSH into the remote server, your local SSH client uses your private key to authenticate itself, and the remote server verifies this by checking if your public key is present in the authorized_keys file. If the key matches and authentication is successful, you'll be granted access without entering a password. And chmod command is used to set the file permissions on the authorized_keys file in the ~/.ssh directory.

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

$ chmod 640 ~/.ssh/authorized_keys
```

Next, verify the password less SSH authentication with the following command:

```
$ ssh localhost
```

You will be asked to authenticate hosts by adding RSA keys to known hosts. Type yes and hit Enter to authenticate the localhost:

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.

ECDSA key fingerprint is
SHA256:JFqDVbM3zTPhUPgD5oMJ4ClviH6tzIRZ2GD3BdNqGMQ.

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Step 4- Installing Hadoop:

Use the following command to download Hadoop 3.3.4 where wget is used to download the files from WWW or various network protocols.

```
$ wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz
```

Once you’ve downloaded the file, you can unzip it to a folder on your hard drive.

```
$ tar xzf hadoop-3.3.4.tar.gz
```

Rename the extracted folder using mv command to remove version information. This is an optional step, but if you don’t want to rename, then adjust the remaining configuration paths.

```
$ mv hadoop-3.3.4 hadoop
```

Next, you will need to configure Hadoop and Java Environment Variables in .bashrc file. Open the ~/.bashrc file in your favorite text editor:

```
$ nano ~/.bashrc
```

Append the below lines to the file. You can find the JAVA_HOME location by running dirname $(dirname $(readlink -f $(which java))) command on the terminal. Then save the file and close it.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/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 HADOOP_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”
```

Then, activate the environment variables with the following command:

```
$ source ~/.bashrc
```

Next, open the Hadoop environment variable file to configure JAVA_HOME in hadoop-env.sh file.

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

Search for the “export JAVA_HOME” and configure it with the value of Java Location. Save and close the file when you are finished.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Step 5- Configuring Hadoop:

First, you will need to create the namenode and datanode directories inside Hadoop home directory:

Run the following command to create both directories where The -p option in the mkdir command is used to create parent directories along with the specified directory path.

```
$ mkdir -p ~/hadoopdata/hdfs/namenode

$ mkdir -p ~/hadoopdata/hdfs/datanode
```

Next, edit the core-site.xml file and update with your system hostname

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

Change the following name as per your system hostname: Then, save and close the file. The following commands sets the default filesystem for Hadoop to be the HDFS instance running on localhost at port

9000. It means that when Hadoop clients (e.g., MapReduce jobs, Spark applications) interact with the filesystem, they will use hdfs://localhost:9000 as the default location for reading and writing data.

```
<configuration>

   <property>

      <name>fs.defaultFS</name>

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

   </property>

</configuration>
```

Then, edit the hdfs-site.xml file:

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

Change the NameNode and DataNode directory paths as shown below: Then save and close the file.

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

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

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

Then, edit the mapred-site.xml file:

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

Make the following changes: Then save and close the file. The following command sets the MapReduce framework to YARN, meaning that MapReduce jobs will be executed using the YARN resource management system. YARN is a key component of Hadoop that manages resources and schedules tasks across the cluster, allowing for efficient and scalable processing of MapReduce jobs

```
<configuration>
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
</configuration>
```

Then, edit the yarn-site.xml file:

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

Make the following changes: Then save and close the file. The MapReduce Shuffle service is a critical component of the Hadoop MapReduce framework. It handles the data shuffling phase in MapReduce jobs, where the data produced by the mappers is sorted and transferred to the reducers for further processing. The MapReduce Shuffle service runs as an auxiliary service on each YARN NodeManager, allowing efficient and optimized data transfer during the MapReduce job execution.

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

Step 6- Start Hadoop Cluster

Before starting the Hadoop cluster. You will need to format the Namenode as a hadoop user. Run the following command to format the Hadoop Namenode.

```
$ hdfs namenode -format
```

Once the namenode directory is successfully formatted with hdfs file system, you will see the message "**Storage directory /home/hadoop/hadoopdata/hdfs/namenode** has been successfully formatted ".

```
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop.tecadmin.net/127.0.1.1
************************************************************/
```

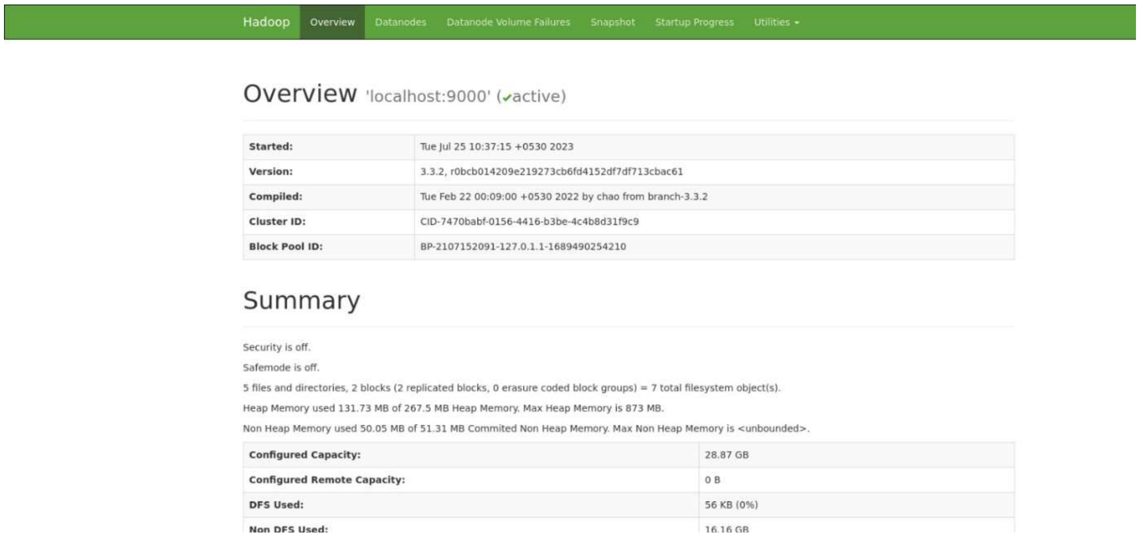
Then start the Hadoop cluster with the following command

```
$ start-all.sh
```

Then you can see the following output:

```
WARNING: Attempting to start all Apache Hadoop daemons as hadoop 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 [Ubuntu]
Starting resourcemanager
Starting nodemanagers
```

Once all the services started, you can access the Hadoop at: http://localhost:9870



Overview 'localhost:9000' (✓active)

Started:	Tue Jul 25 10:37:15 +0530 2023
Version:	3.3.2, r0bcb014209e219273cb6fd4152df7df713cbac61
Compiled:	Tue Feb 22 00:09:00 +0530 2022 by chao from branch-3.3.2
Cluster ID:	CID-7470babf-0156-4416-b3be-4c4b8d31f9c9
Block Pool ID:	BP-2107152091-127.0.1.1-1689490254210

Summary

Security is off.

Safemode is off.

5 files and directories, 2 blocks (2 replicated blocks, 0 erasure coded block groups) = 7 total filesystem object(s).

Heap Memory used 131.73 MB of 267.5 MB Heap Memory. Max Heap Memory is 873 MB.

Non Heap Memory used 50.05 MB of 51.31 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	28.87 GB
Configured Remote Capacity:	0 B
DFS Used:	56 KB (0%)
Non DFS Used:	16.16 GB

Result:

Thus, hadoop has been installed successfully and different Hadoop's modes, startup scripts, and configuration files have been explored.

Ex. No.: 2 **Hadoop Implementation of file management tasks, such as Adding files and**
Date: **directories, retrieving files and Deleting files**

Aim:
To implement file management tasks, such as Adding files and directories, retrieving files and Deleting files in Hadoop.

Description:
Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing. HDFS follows the master-slave architecture and it has the following elements.

Namenode:
The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode:
The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block:
Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Read Operations in HDFS:
To read any file from the HDFS, Users have to interact with the NameNode as it stores the metadata about the DataNodes. The user gets a token from the NameNode and that specifies the address where the data is stored. Then, user can put a read request to NameNode for a particular block location through distributed file systems. The NameNode will then check user's privilege to access the DataNode and allow that user to read the address block if the access is valid.

Write Operation in HDFS:
Similar to the read operation, the HDFS write operation is used to write the file to a particular address through the NameNode. This NameNode provides the slave address where the client/user can write or add data. After writing to the block location, the slave replicates that block and copies it to another slave location using factor 3 replication. The salve is then reverted back to the client for authentication.

Commands to add, retrieve and delete files from HDFS:

Starting HDFS:

Format the configured HDFS file system and then open the namenode (HDFS server) and execute the following HDFS command.

```
$ hadoop namenode -format
```

Start the distributed file system and follow the command listed below to start the namenode as well as the data nodes in cluster.

```
$ start-dfs.sh
```

Inserting Data into HDFS:

Step 1: Use the following command to create an input directory where -mkdir is used to create the directory and '/' (forward slash indicates root directory) and '/abcd' indicates the directory 'abcd' will be created directly under the root directory of HDFS.

```
Syntax:
$ hadoop fs -mkdir  /dir-name
Example:
$ hadoop fs -mkdir  /abcd
```

Step 2: Transfer the file from local file system to hadoop file system using 'put' command where '/home/hadoop/input/1.txt' is the source path of the file that users want to copy. '/abcd' is the destination path in HDFS where users want to place the copied file.

```
Syntax:
$ hadoop fs -put  /path/to/local/filename.txt   /hdfs/destination/path
Example:
$ hadoop fs -put  /home/hadoop/input/1.txt   /abcd
```

Step 3: Now, transferred file can be verified using the following command where 'ls' is used to list the contents of directory 'abcd'.

```
$ hadoop fs -ls  /abcd
```

Retrieving files from HDFS:

Step 1: To retrieve file from HDFS to local system, use the following command where 'get' is used to retrieve the file from HDFS. It will copy the file "1.txt" from the HDFS directory /abcd and the copied file will be placed in the /home/hadoop/input directory on local machine.

Syntax:

```
$ hadoop fs -get  /hdfs/path/filename.txt  /path/to/local
```

Example:

```
$ hadoop fs -get  /abcd/1.txt  /home/hadoop/input
```

Step 2: The 'cat' command is used to view the content of the file stored in HDFS directly in the terminal.

```
$ hadoop fs -cat  /abcd/1.txt
```

Deleting files from HDFS:

Step 1: To delete a file from HDFS, use 'rm' command as shown below in which it deletes the file '1.txt' stored in HDFS. This command is a permanent action, and the file cannot be recovered once it is deleted.

Syntax:

```
$ hadoop fs -rm  /hdfs/path/filename.txt
```

Example:

```
$ hadoop fs -rm  /abcd/1.txt
```

Step 2: To delete a directory and its content from HDFS, use 'rm' and 'r' commands as shown below in which 'r' performs removal operation recursively. It will recursively traverse the directory structure starting from the /abcd directory in HDFS. It will delete all the files and subdirectories within the specified directory, as well as the directory itself. The removal is performed in a cascading manner, meaning that all contents within the directory, including subdirectories and their contents, will be deleted. Once the command completes, the "abcd" directory and its entire contents will be permanently removed from HDFS.

```
$ hadoop fs -rm -r  /abcd
```

Shutting down the HDFS:

Shut down the HDFS using following command.

```
$ stop-dfs.sh
```

Result:

The file management tasks, such as Adding files and directories, retrieving files and deleting files have been implemented in Hadoop.

Ex. No.: 3 **Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm**
Date:

Aim:
To run a basic word count map-reduce program to understand map reduce paradigm.

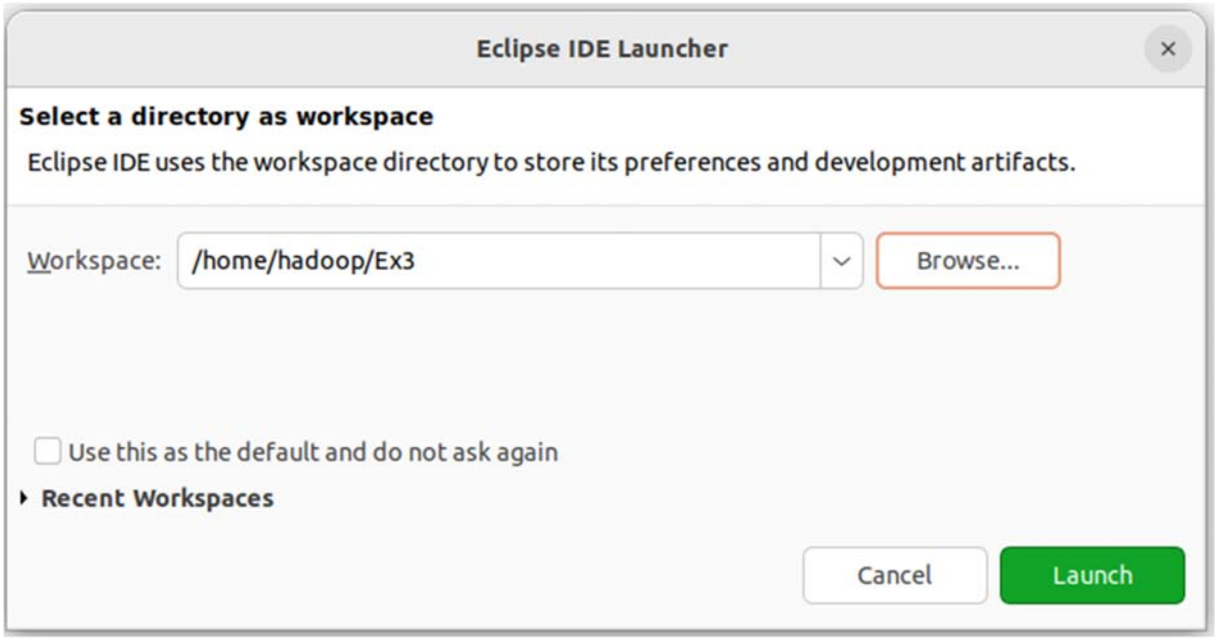
Map-Reduce Function in Hadoop:
MapReduce is a Java-based, distributed execution framework within the Apache Hadoop Ecosystem. Using MapReduce, we can concurrently split and process petabytes of data in parallel. This whole process is supported by two main tasks: mapping and reducing. This programming model is highly dependent on key-value pairs for processing.

- **Mapping:** This process takes an input in the form of key-value pairs and produces another set of intermediate key-value pairs after processing the input.
- **Reducing:** This process takes the output from the map task and further processes it into even smaller and possibly readable chunks of data. However, the outcome is still in the form of key-value pairs.

Procedure:
Step 1: Install Eclipse using following commands.

```
$ sudo apt update
$ sudo apt upgrade
snap –version
snap install --classic eclipse
```

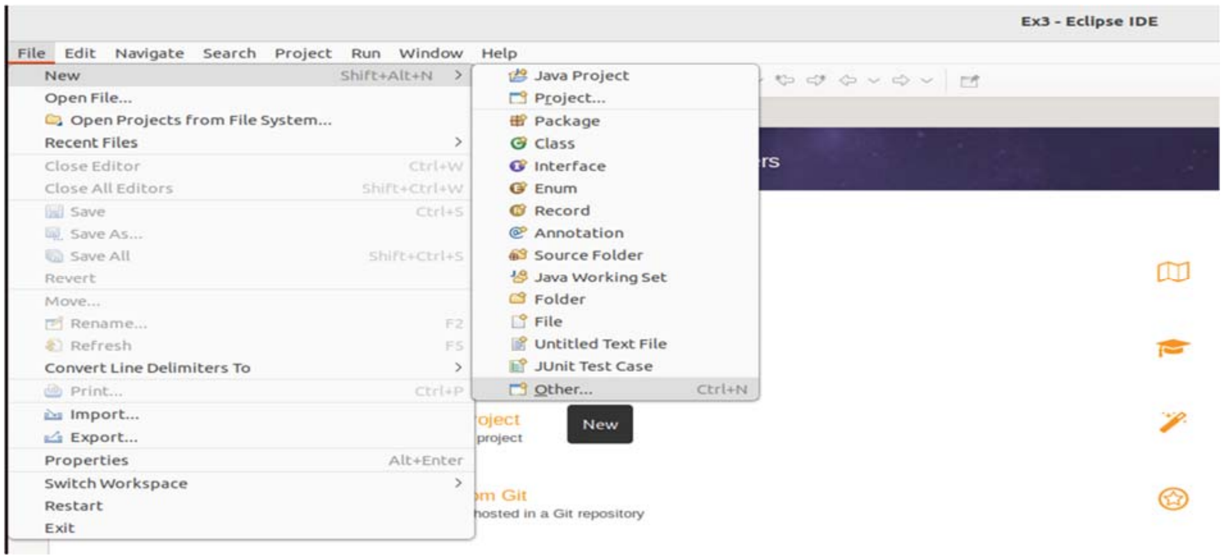
Step 2: Open Eclipse and create a workspace as shown below: Here directory'Ex3' is created in the root and it act as workspace for the project.

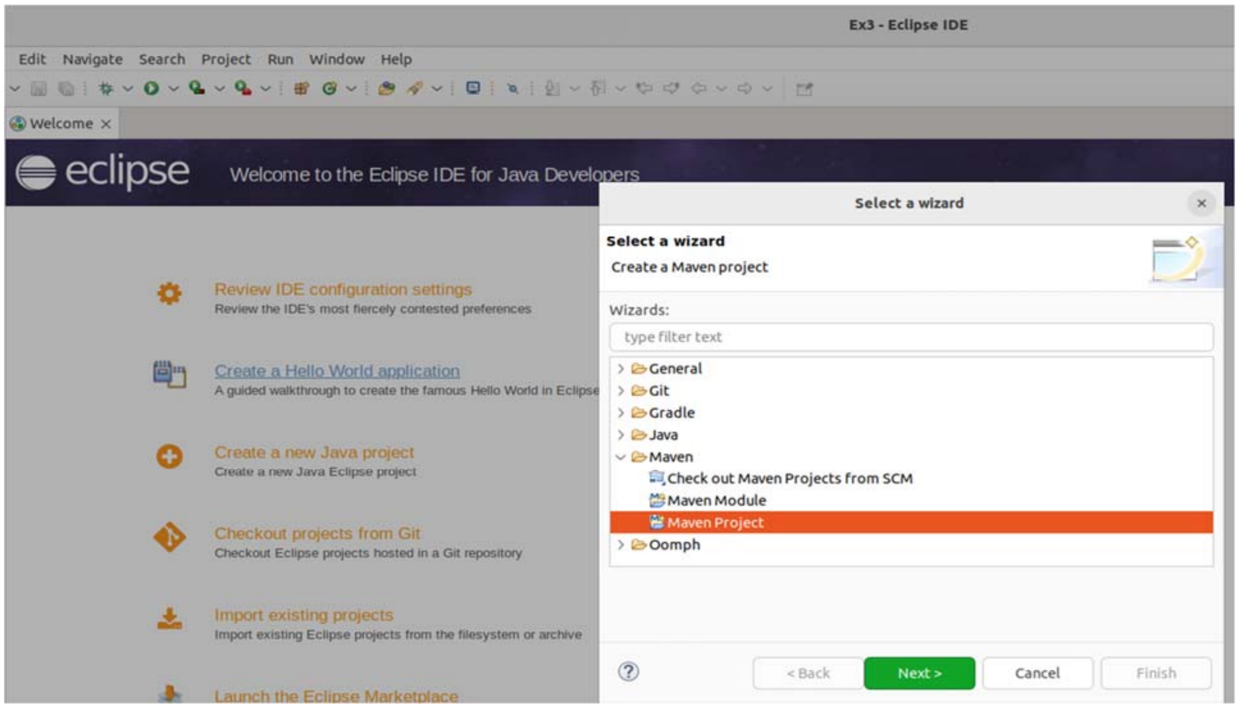

Eclipse IDE Launcher

Select a directory as workspace
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/hadoop/Ex3

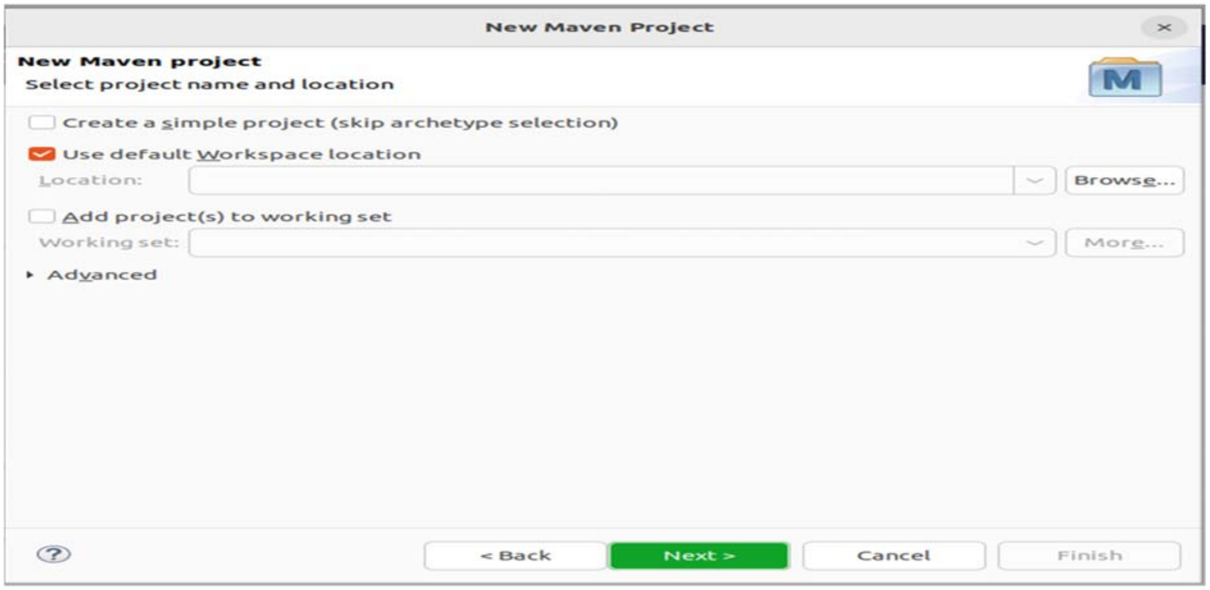
Step 3: Now, New Maven Project is created by using the steps given below:




Ex3 - Eclipse IDE
Edit Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for Java Developers
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse
Create a new Java project
Create a new Java Eclipse project
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Launch the Eclipse Marketplace
Select a wizard
Select a wizard
Create a Maven project
Wizards:
type filter text
General
Git
Gradle
Java
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
Oomph
< Back
Next >
Cancel
Finish

Maven simplifies the process of managing external libraries and dependencies required for your MapReduce projects. You can specify the dependencies in the project's pom.xml (Project Object Model) file, and Maven will automatically download and include the necessary JAR files in your project. This is helpful because Hadoop has a complex ecosystem with many related libraries and dependencies.

Step 5: Once Maven Project is selected, select default workspace location as shown below:



New Maven Project

New Maven project

Select project name and location

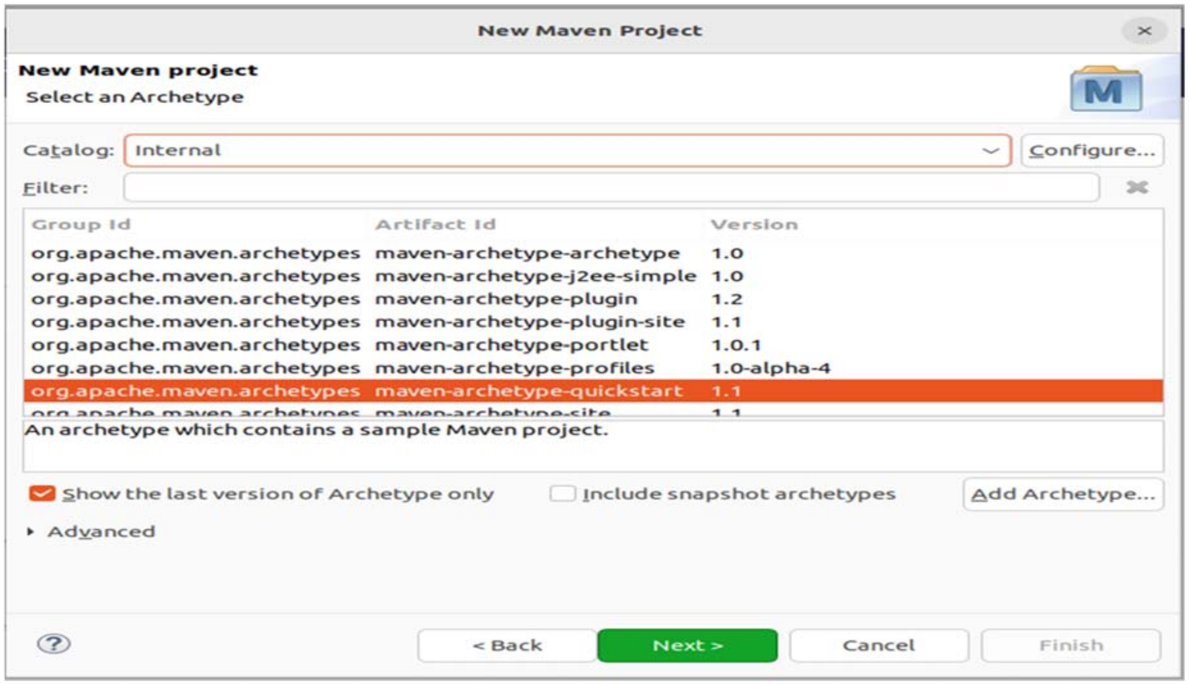
- [ ] Create a simple project (skip archetype selection)
- [x] Use default Workspace location

Location:

- [ ] Add project(s) to working set

Working set:

Step 6: Then select an archetype which is predefined project template that you can use as a starting point for creating new projects. The maven-archetype-quickstart archetype is commonly chosen because it provides a simple and standard project structure that is suitable for many Java applications.



New Maven Project

New Maven project

Select an Archetype

Catalog: Internal

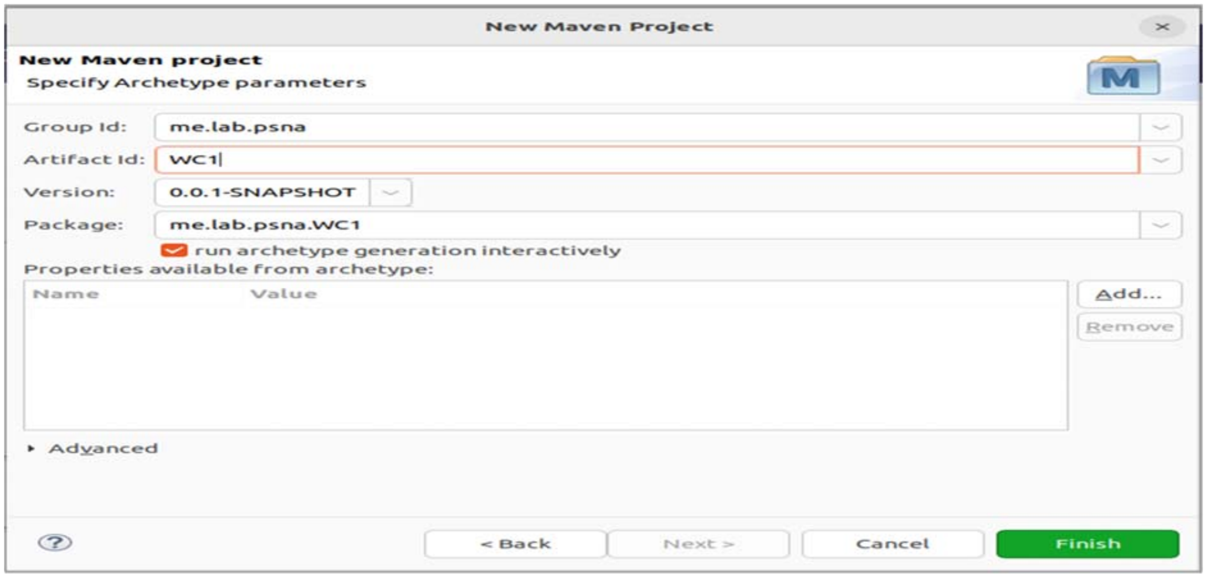
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1

An archetype which contains a sample Maven project.

- [x] Show the last version of Archetype only
- [ ] Include snapshot archetypes

Step 7: Create Group ID and Artifact ID in maven using the steps given below: In Maven, the groupId and artifactId are essential elements that help uniquely identify and organize your projects and artifacts. They are part of the Project Object Model (POM) configuration and play a crucial role in managing project dependencies, versioning, and project organization.



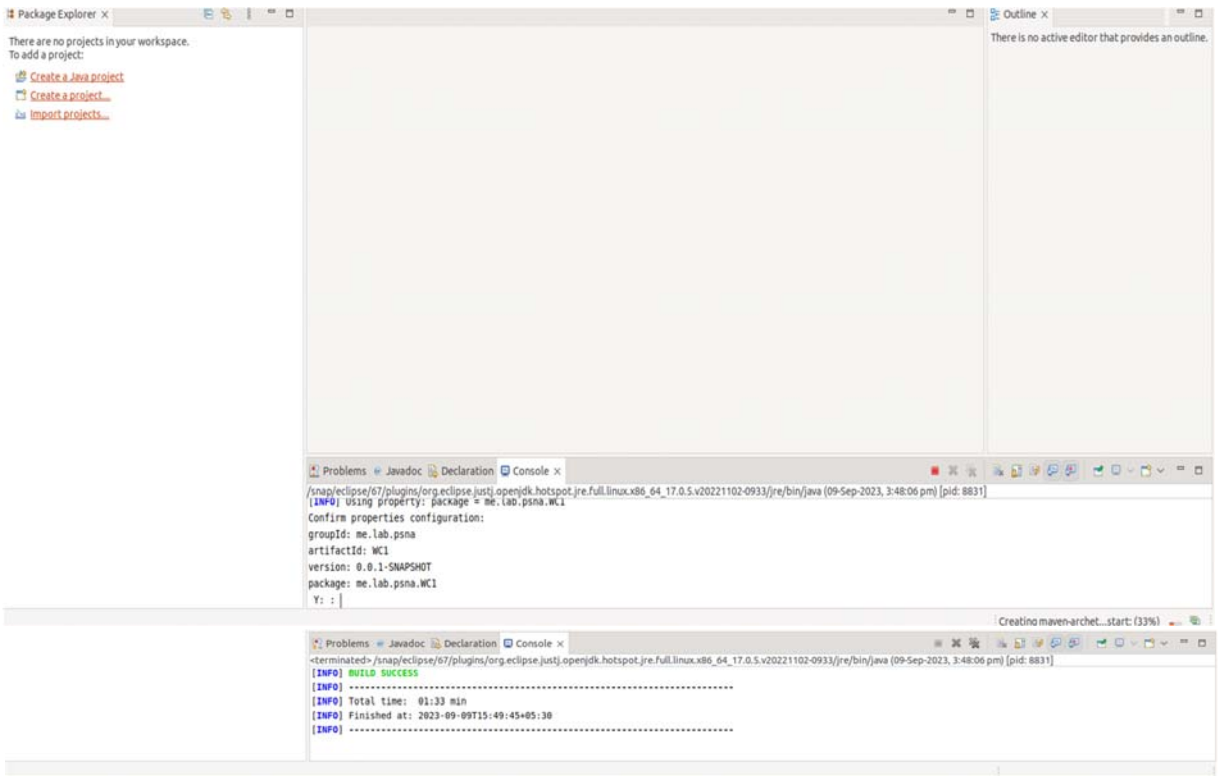
New Maven Project

New Maven project
Specify Archetype parameters

Group Id: me.lab.psna
Artifact Id: WC1
Version: 0.0.1-SNAPSHOT
Package: me.lab.psna.WC1
[✓] run archetype generation interactively
Properties available from archetype:

Name	Value

Step 8: Now, Maven Project is created in Eclipse by giving 'Y' and you can see the build success message in Console as shown below.



There are no projects in your workspace.
To add a project:

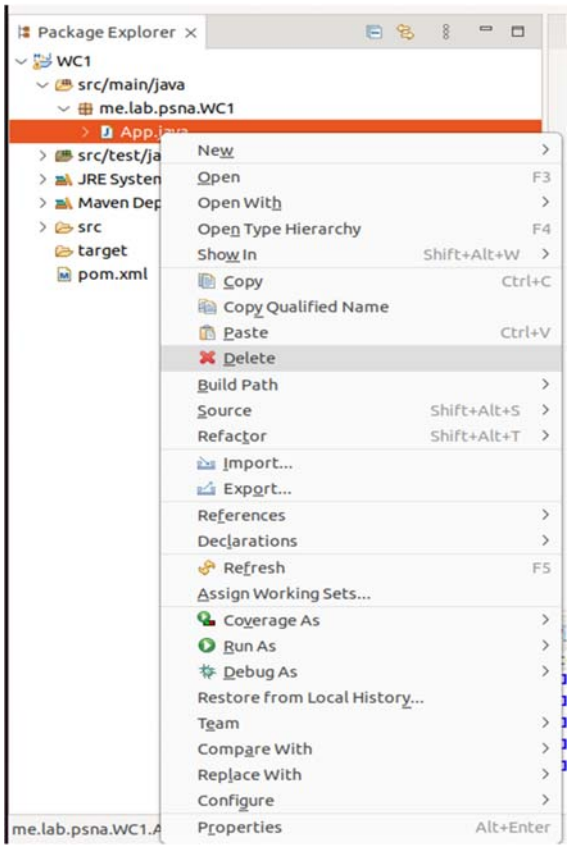
There is no active editor that provides an outline.

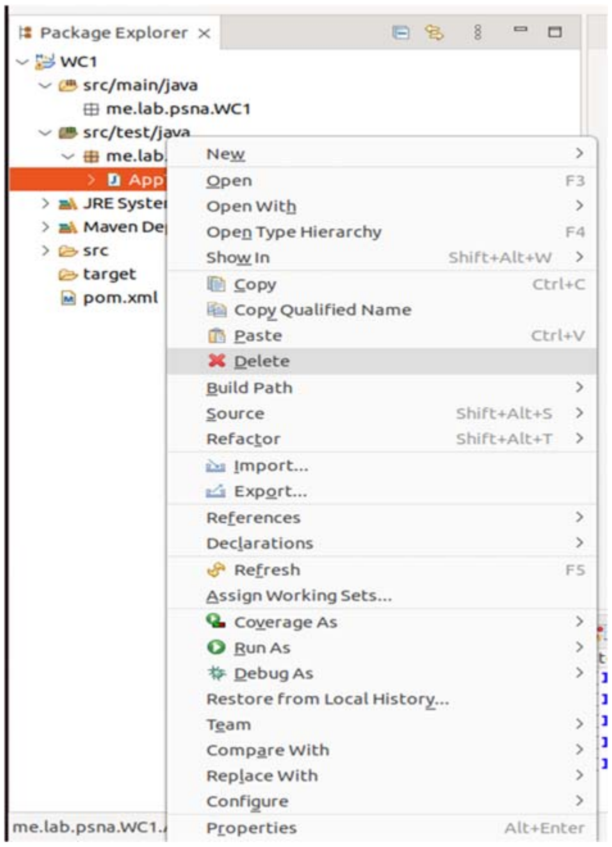
```
/snap/eclipse/67/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java (09-Sep-2023, 3:48:06 pm) [pid: 8831]
[INFO] Using property: package = me.lab.psna.WC1
Confirm properties configuration:
groupId: me.lab.psna
artifactId: WC1
version: 0.0.1-SNAPSHOT
package: me.lab.psna.WC1
 Y: :
```

Creating maven-archet...start: (33%)

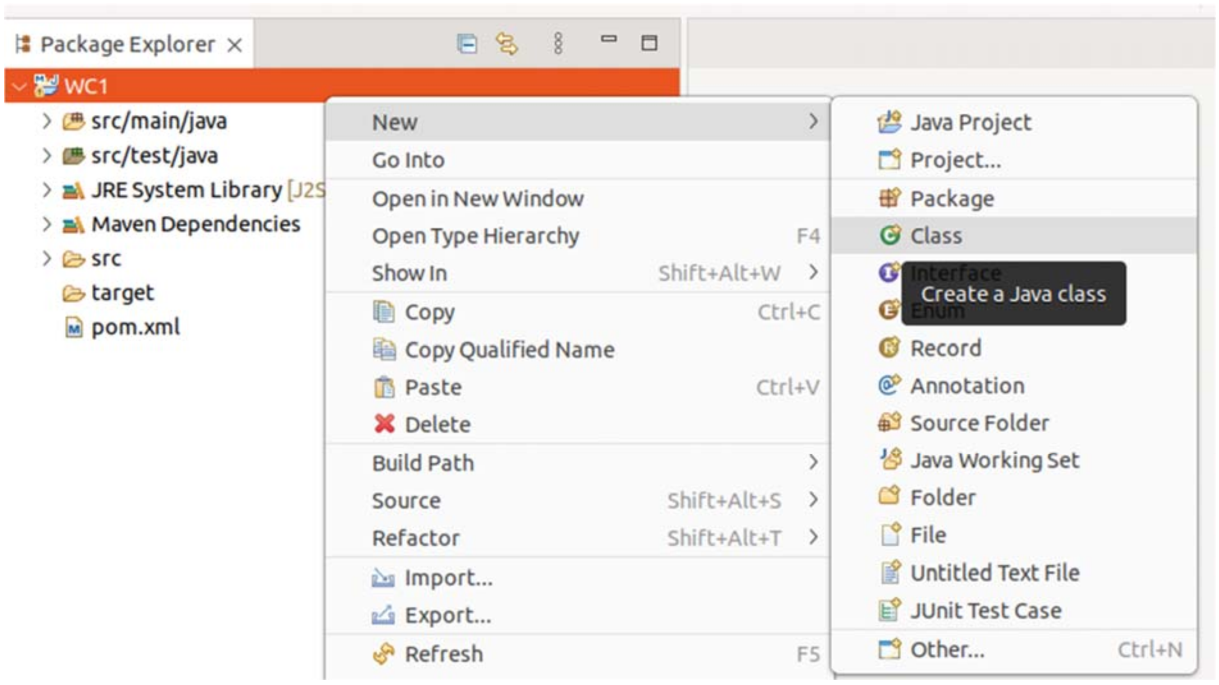
```
<terminated> /snap/eclipse/67/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java (09-Sep-2023, 3:48:06 pm) [pid: 8831]
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:33 min
[INFO] Finished at: 2023-09-09T15:49:45+05:30
[INFO] ------------------------------------------------------------------------
```

Step 9: Now, New project is shown in eclipse with the name as given in artifact ID. And then remove the default files app.java from the package WC1 as shown below:

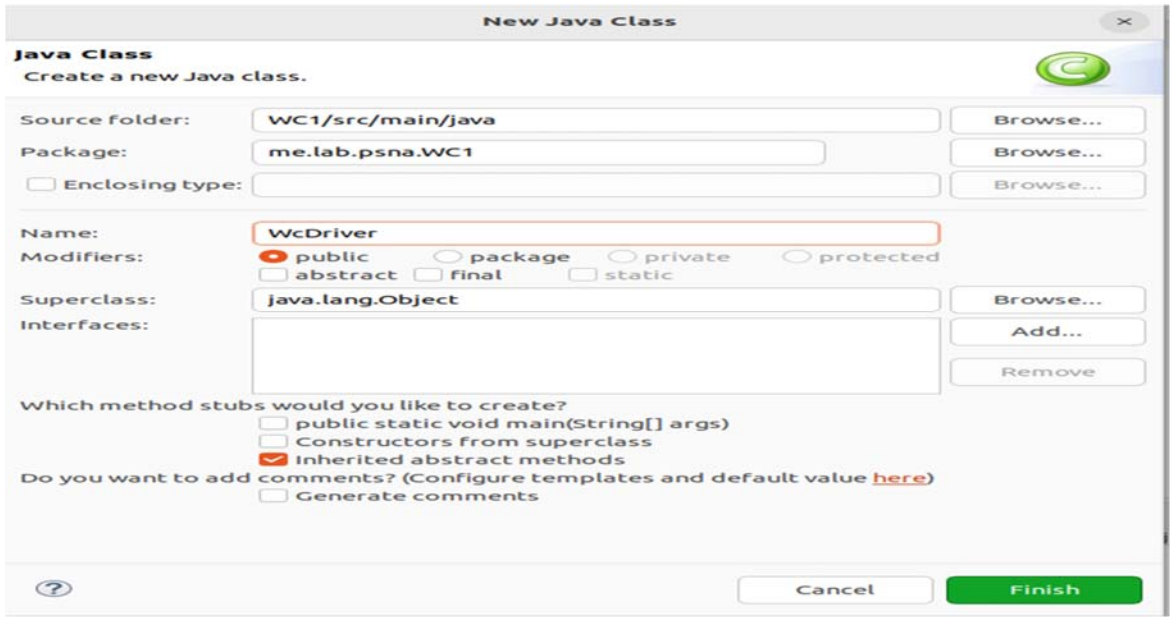




Step 10: Now create java programs using the steps given below:



Step 11: File name is given using the steps given below. Likewise, three files can be created such as WcDriver, WcMapper and WcReducer.


New Java Class

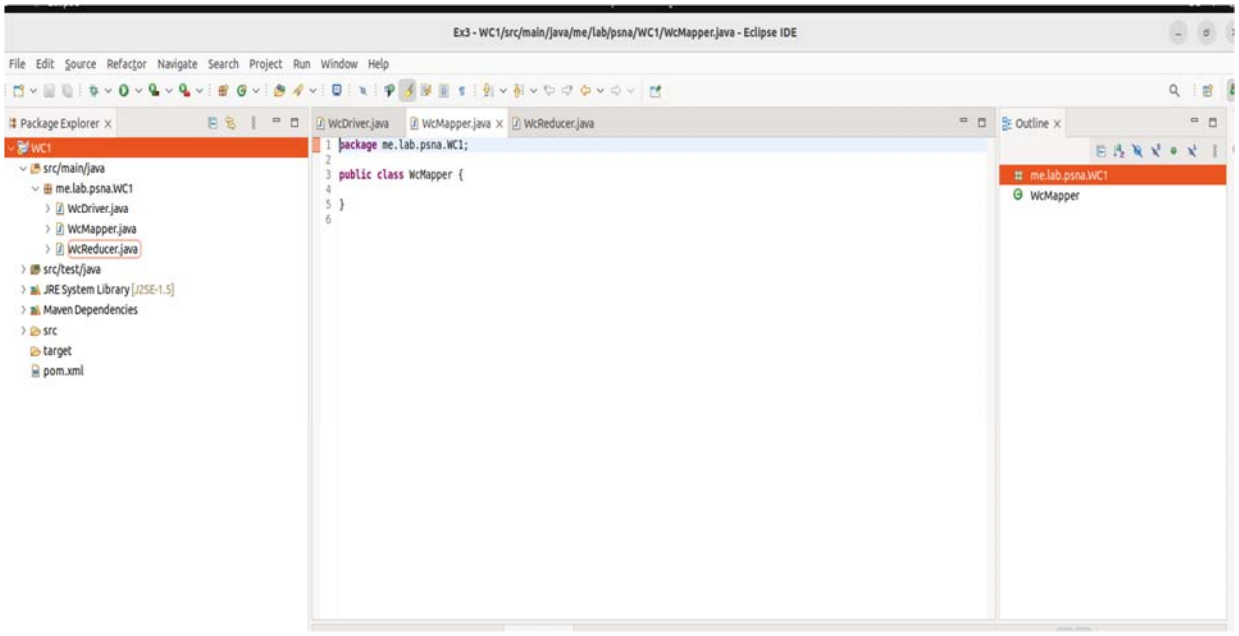
Java Class
Create a new Java class.

Source folder: WC1/src/main/java
Package: me.lab.psna.WC1
☐ Enclosing type:

Name: WcDriver
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

Step 12: Then write a program under each tab as shown below:


Ex3 - WC1/src/main/java/me/lab/psna/WC1/WcMapper.java - Eclipse IDE

```
package me.lab.psna.WC1;

public class WcMapper {

}
```

WcDriver.Java

```
package me.lab.psna.WC1
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class WcDriver extends Configured implements Tool{
public static void main(String[] args) throws Exception {
int exitCode=ToolRunner.run(new WcDriver(), args);
System.out.println("Exit code :"+exitCode);
System.exit(exitCode);
 }
public int run(String[] arg0) throws Exception {
Job job=Job.getInstance();
job.setJobName("Word Count");
job.setJarByClass(WcDriver.class);
job.setMapperClass(WcMapper.class);
job.setReducerClass(WcReducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(arg0[0]));
FileOutputFormat.setOutputPath(job, new Path(arg0[1]));
int ecode=job.waitForCompletion(true) ? 0:1;
return ecode;
}
}
```

WcMapper.java

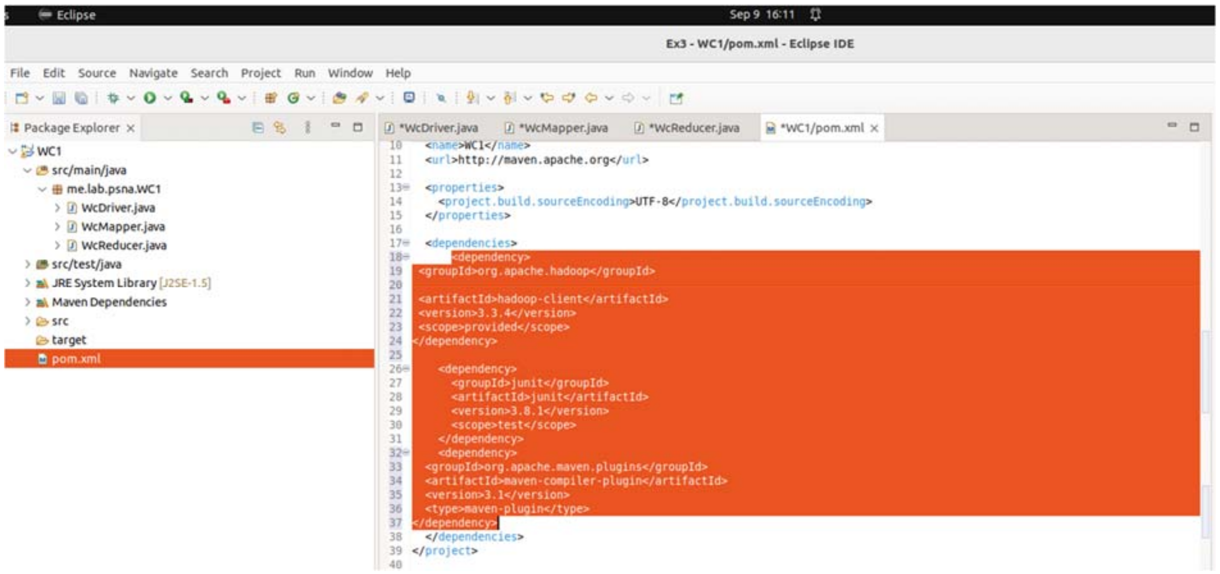
```
package me.lab.psna.WC1
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
public class WcMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
 @Override
 public void map(LongWritable key,Text value,Context context) throws
 IOException,InterruptedException{
 String line=value.toString();
 String[] words=line.split(" ");
 for(String word:words){
 context.write(new Text(word),new IntWritable(1));
 }
 }
 }
```

WcReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WcReducer extends Reducer<Text,IntWritable,Text,IntWritable>{
@Override
public void reduce(Text key,Iterable<IntWritable> values,Context context) throws
IOException,InterruptedException{
int sum=0;
for(IntWritable value:values){
sum+=value.get();
```

```
 }
 context.write(key, new IntWritable(sum));
 }
}
```

Step 12: Write the following dependencies in pom.xml file as shown below:



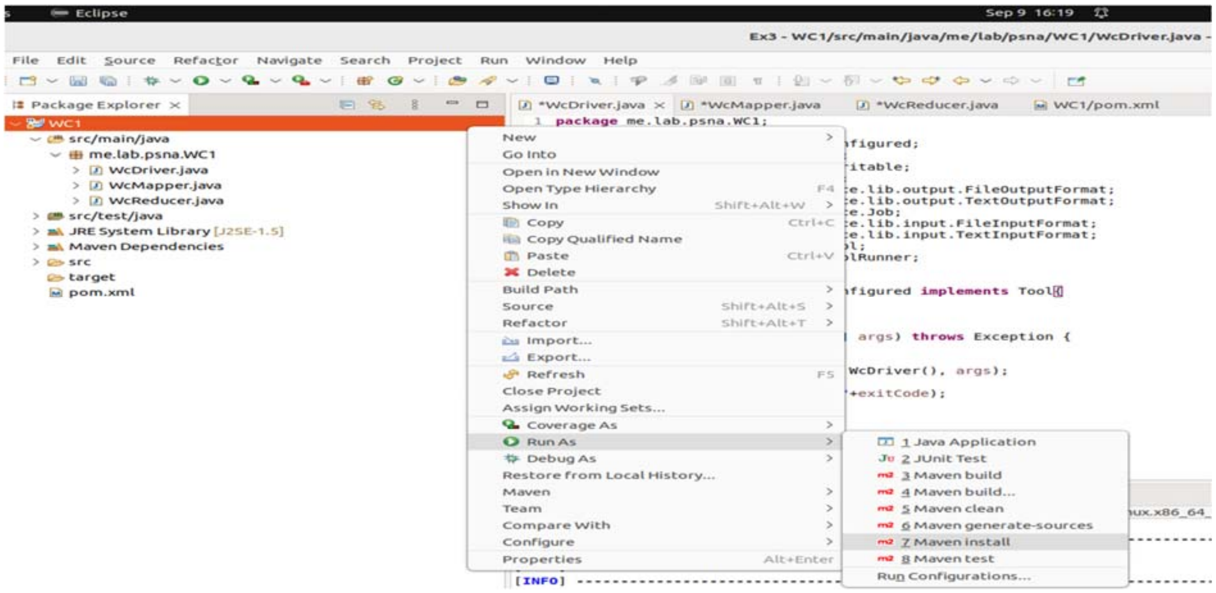
```
  <name>WC1</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
      <dependency>
<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>
<version>3.3.4</version>
<scope>provided</scope>
</dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <type>maven-plugin</type>
</dependency>
  </dependencies>
</project>
```

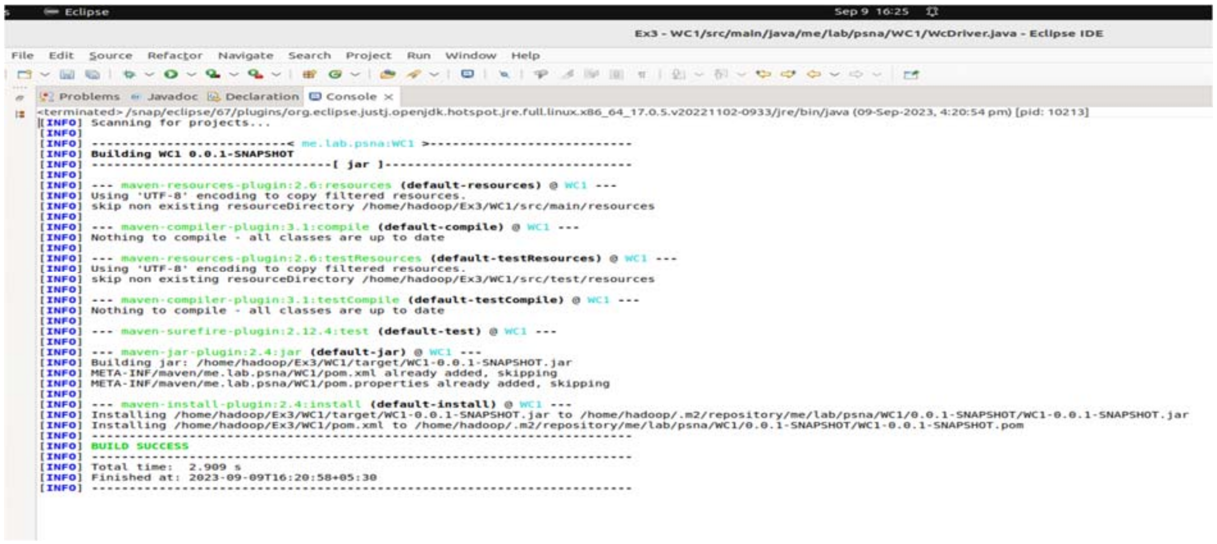
pom.xml:

```
<dependencies>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-client</artifactId>
<version>3.3.4</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<type>maven-plugin</type>
</dependency>
</dependencies>
```

Step 13: Now JAR file is created using the steps given below: The generated JAR file typically contains your compiled Java code, resources, and other necessary files which is stored in the workspace created in step 2.

```
/home/hadoop/Ex3/WC1/WC1-0.0.1-SNAPSHOT.jar
```




Eclipse
Ex3 - WC1/src/main/java/me/lab/psna/WC1/WcDriver.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Console

```
<terminated> /snap/eclipse/67/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java (09-Sep-2023, 4:20:54 pm) [pid: 10213]
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< me.lab.psna:WC1 >--------------------------
[INFO] Building WC1 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]--------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ WC1 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/hadoop/Ex3/WC1/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ WC1 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ WC1 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/hadoop/Ex3/WC1/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ WC1 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ WC1 ---
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ WC1 ---
[INFO] Building jar: /home/hadoop/Ex3/WC1/target/WC1-0.0.1-SNAPSHOT.jar
[INFO] META-INF/maven/me.lab.psna/WC1/pom.xml already added, skipping
[INFO] META-INF/maven/me.lab.psna/WC1/pom.properties already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ WC1 ---
[INFO] Installing /home/hadoop/Ex3/WC1/target/WC1-0.0.1-SNAPSHOT.jar to /home/hadoop/.m2/repository/me/lab/psna/WC1/0.0.1-SNAPSHOT/WC1-0.0.1-SNAPSHOT.jar
[INFO] Installing /home/hadoop/Ex3/WC1/pom.xml to /home/hadoop/.m2/repository/me/lab/psna/WC1/0.0.1-SNAPSHOT/WC1-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.909 s
[INFO] Finished at: 2023-09-09T16:20:58+05:30
[INFO] ------------------------------------------------------------------------
```

Step 14: Start all hadoop daemons using the command given below in the terminal:

```
$ start-all.sh
```

Step 15: Create input directory using the command given below:

```
$ hadoop fs -mkdir /user/wc/input
```

Step 16: Now copy the input file ‘1.txt’ to the input directory using the following command.

```
$ hadoop fs -put '/home/hadoop/Sample/1'  /user/wc/input
```

Sample Input:
1.txt:
Apple
Banana
Cat
Cat
Dog

Step 17: Open mapred-site.xml in nano editor and edit the file with following content.

```
$ nano mapred-site.xml

<property>
<name>yarn.app.mapreduce.am.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop
</value>
</property>
<property>
<name>mapreduce.map.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop</value>
</property>
<property>
<name>mapreduce.reduce.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop</value>
</property>
```

Step 18: Then, run the java program in hadoop using the command given below:

- **hadoop jar:** This is the command to submit a Hadoop MapReduce job. It tells Hadoop to execute a Java JAR file that contains the MapReduce application.
- **/home/hadoop/Ex3/WC1/target/WC1-0.0.1-SNAPSHOT.jar**: The JAR file likely contains your MapReduce driver code, configuration, and any required dependencies.
- **me/lab/psna/WC1/WcDriver:** It specifies the class that contains the main method where the MapReduce job configuration and execution are defined.
- **/user/wc/input/1:** It specifies the path where the input data for your MapReduce job is located.
- **/user/wc/output/:** It specifies the path where output of your MapReduce job will be stored.

```
$ hadoop jar '/home/hadoop/Ex3/WC1/target/WC1-0.0.1-SNAPSHOT.jar'
me/lab/psna/WC1/WcDriver   /user/wc/input/1   /user/wc/output/
```

Step 19: Now, the output from output directory can be read by using the command given below: In Hadoop MapReduce, the part-r-00000 file name is a convention for the output files generated by the reducer tasks as part of the MapReduce job.

```
$ hadoop fs -cat /user/wc/output/part-r-00000

Apple        1
Banana       1
Cat          2
Dog          1
```

Result: Thus map-reduce program for basic word count problem has been implemented.

Ex. No.: 4 **Implement of Matrix Multiplication with Hadoop Map Reduce**
Date:

Aim:
To implement the Matrix Multiplication program using Hadoop Map Reduce.

Map-Reduce Function in Hadoop:
This MapReduce program leverages parallel processing to efficiently compute the matrix product by distributing the workload across multiple mappers and reducers. It divides the input matrices into smaller blocks, processes them in parallel, and aggregates the results to obtain the final matrix.
Mapper Phase:
- Input matrices A and B are divided into smaller blocks.
- Mappers process these blocks, emitting key-value pairs for each element.
- The key is (i, j) to represent the element's position in the result matrix C.
- The value includes the element's value and a label indicating whether it comes from matrix A or B.

Shuffle and Sort Phase:
- The framework groups and sorts key-value pairs by their (i, j) keys, ensuring elements with the same indices are together.

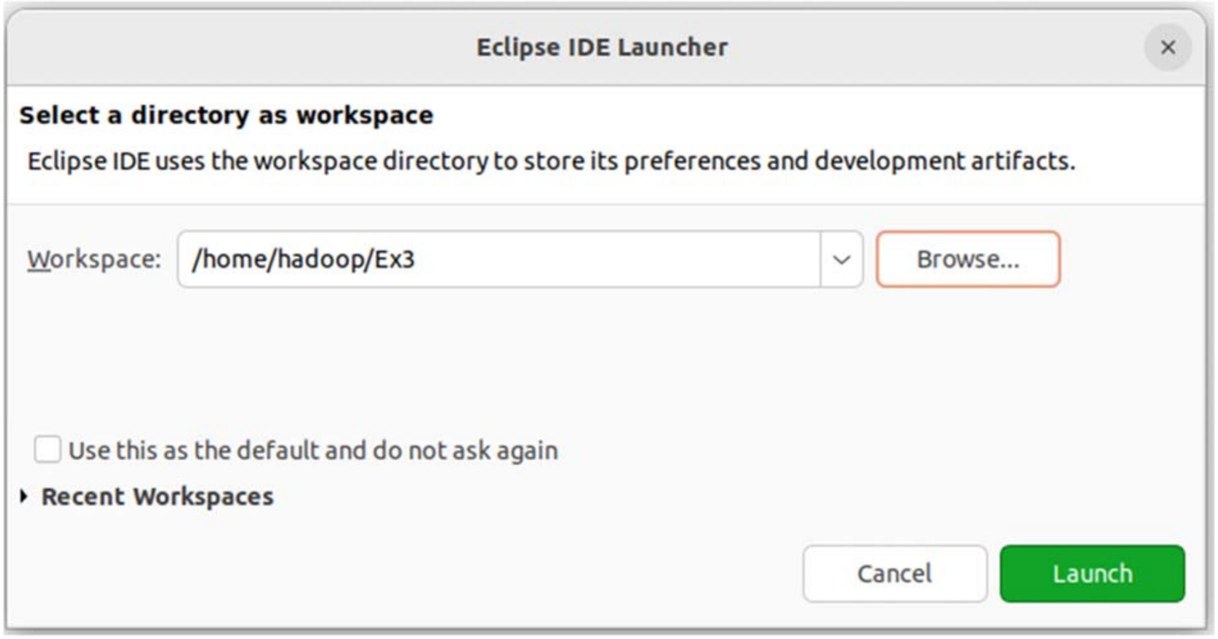
Reducer Phase:
- Reducers receive grouped elements for the same (i, j) indices.
- They perform the actual matrix multiplication for C[i][j] by multiplying A and B elements and summing the results.
- The result is emitted as a key-value pair with no key, and the value is the final C[i][j] value.

Output:
- The output consists of key-value pairs representing the (i, j) indices of matrix C and their corresponding values.

Procedure:
Step 1: Install Eclipse using following commands.

```
$ sudo apt update
$ sudo apt upgrade
snap –version
snap install --classic eclipse
```

Step 2: Open Eclipse and create a workspace as shown below: Here directory’Ex3’ is created in the root and it act as workspace for the project.

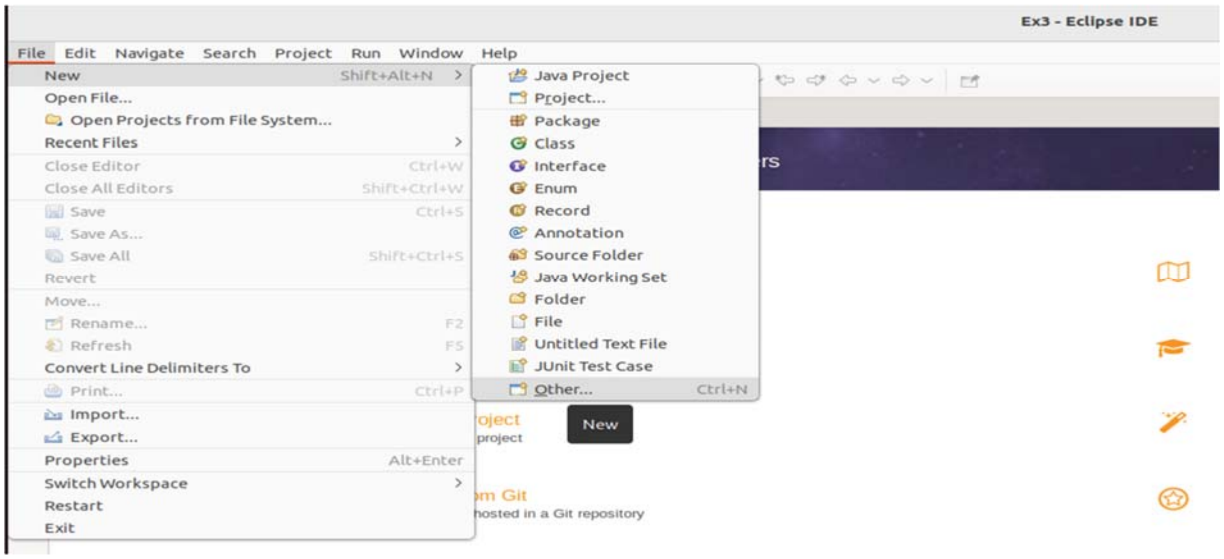

Eclipse IDE Launcher

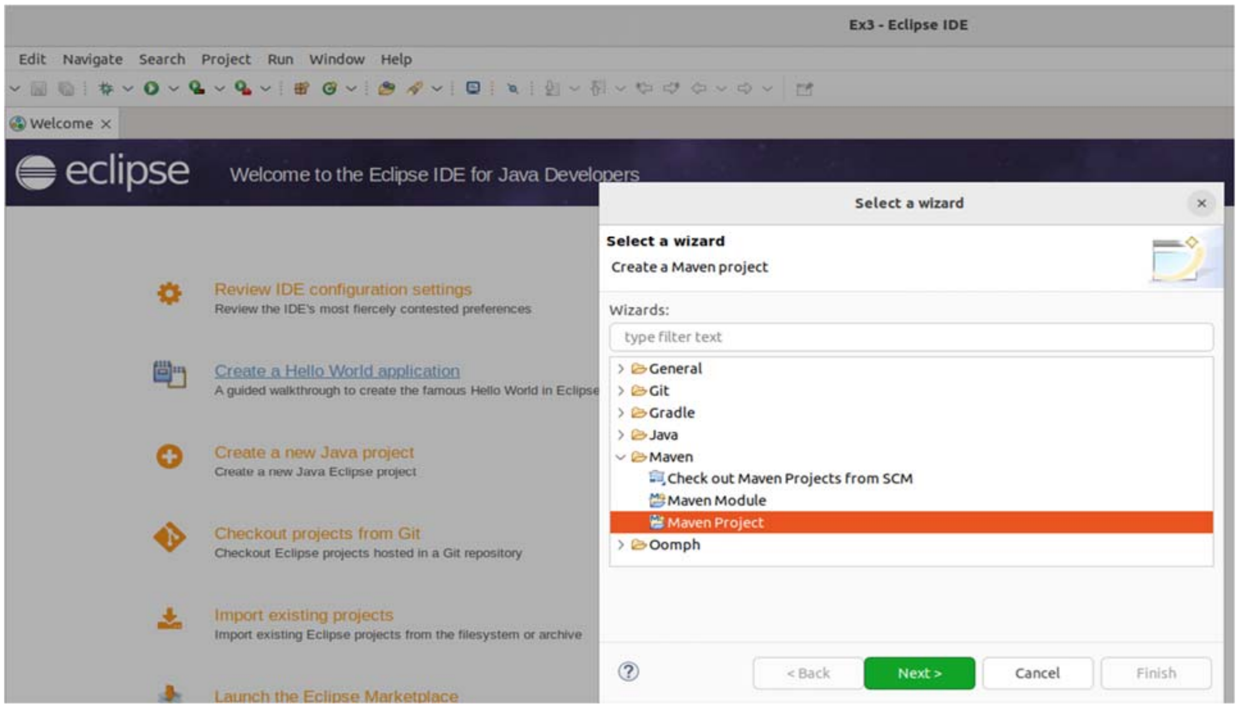
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/hadoop/Ex3

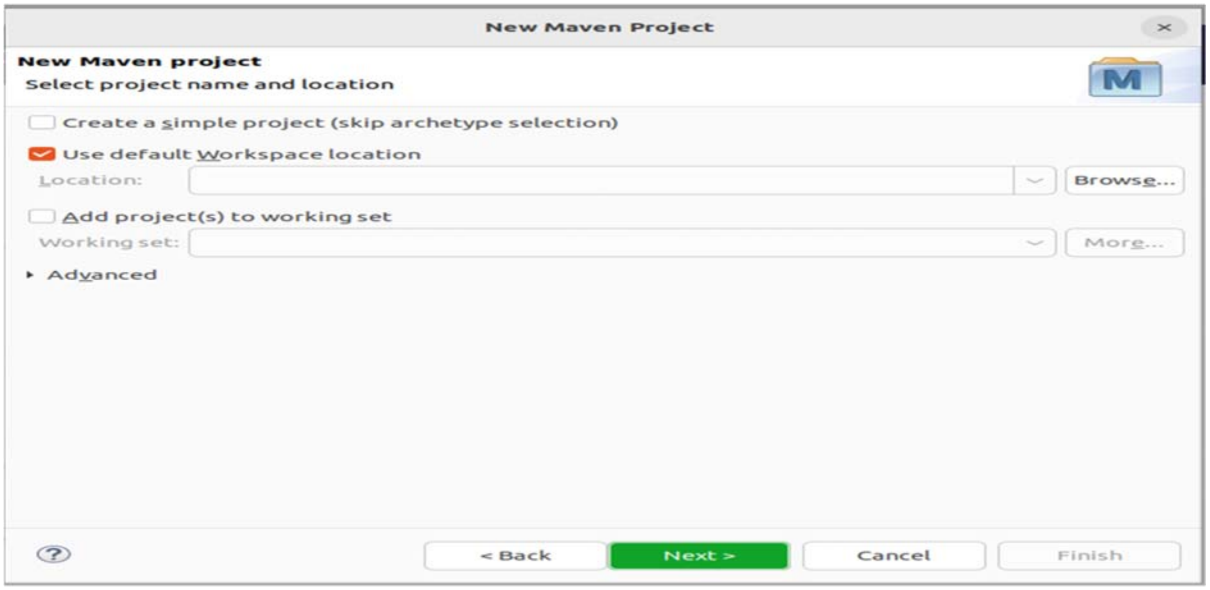
Step 3: Now, New Maven Project is created by using the steps given below:



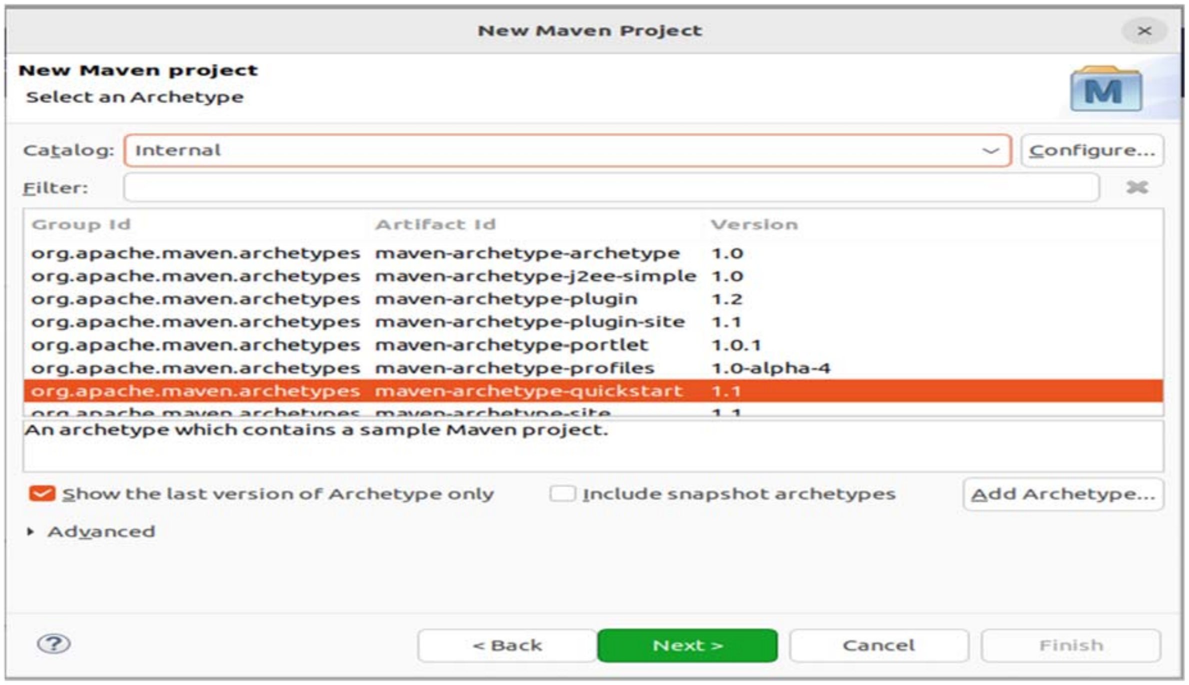

Ex3 - Eclipse IDE
Edit Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for Java Developers
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse
Create a new Java project
Create a new Java Eclipse project
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Launch the Eclipse Marketplace
Select a wizard
Select a wizard
Create a Maven project
Wizards:
type filter text
General
Git
Gradle
Java
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
Oomph
< Back
Next >
Cancel
Finish

Maven simplifies the process of managing external libraries and dependencies required for your MapReduce projects. You can specify the dependencies in the project's pom.xml (Project Object Model) file, and Maven will automatically download and include the necessary JAR files in your project. This is helpful because Hadoop has a complex ecosystem with many related libraries and dependencies.

Step 4: Once Maven Project is selected, select default workspace location as shown below:


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Cancel
Finish

Step 5: Then select an archetype which is predefined project template that you can use as a starting point for creating new projects. The maven-archetype-quickstart archetype is commonly chosen because it provides a simple and standard project structure that is suitable for many Java applications.



Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1

An archetype which contains a sample Maven project.

Step 6: Create Group ID and Artifact ID in maven project.

Step 7: Now, Maven Project is created in Eclipse by giving ‘Y’ and you can see the build success message in Console.

Step 8: Now, New project is shown in eclipse with the name as given in artifact ID. And then remove the default files app.java from the package.

Step 9: Now create java program with the name of MatrixMultiplication.

Step 10: Then write a program in MatrixMultiplication tab.

MatrixMultiplication.java

```
import java.io.IOException;
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 MatrixMultiplication {
public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {
private Text outputKey = new Text();
private Text outputValue = new Text();
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
String[] parts = value.toString().split(" ");
String matrixName = parts[0];
int row = Integer.parseInt(parts[1]);
int col = Integer.parseInt(parts[2]);
int matrixValue = Integer.parseInt(parts[3]);
outputKey.set(row + " " + col);
outputValue.set(matrixName + " " + matrixValue);
context.write(outputKey, outputValue);
}
}
public static class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
int[] matrixA = new int[3]; // Assuming 3x3 matrices
int[] matrixB = new int[3];
// Parse the key to get row and column
String[] parts = key.toString().split(" ");
int row = Integer.parseInt(parts[0]);
int col = Integer.parseInt(parts[1]);
for (Text val : values) {
String[] valParts = val.toString().split(" ");
String matrixName = valParts[0];
int value = Integer.parseInt(valParts[1]);
if (matrixName.equals("A")) {
matrixA[col] = value;
}
else if (matrixName.equals("B")) {
matrixB[row] = value;
}
}
int dotProduct = 0;
for (int i = 0; i < 3; i++) {
dotProduct += matrixA[i] * matrixB[i];
}
result.set(dotProduct);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "MatrixMultiplication");
job.setJarByClass(MatrixMultiplication.class);
job.setMapperClass(MatrixMapper.class);
job.setReducerClass(MatrixReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```
Step 11: Write the following dependencies in pom.xml file:
pom.xml:
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
  <groupId>me.psna.lab</groupId>
  <artifactId>martix</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>martix</name>
  <url>http://maven.apache.org</url>
<build>
	<finalName>${project.name}</finalName>
	<plugins>
		<plugin>
			<artifactId>maven-compiler-plugin</artifactId>
			<version>3.1</version>
			<configuration>
				<source>1.7</source>
				<target>1.7</target>
			</configuration>
		</plugin>
	</plugins>
</build>
  <properties>
	<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
	  <dependency>
		<groupId>org.apache.hadoop</groupId>
		 <artifactId>hadoop-client</artifactId>
		<version>3.3.4</version>
		<scope>provided</scope>
	</dependency>
	<dependency>
		<groupId>org.apache.maven.plugins</groupId>
		<artifactId>maven-compiler-plugin</artifactId>
		<version>3.1</version>
		<type>maven-plugin</type>
	</dependency>
	<dependency>
		<groupId>junit</groupId>
		<artifactId>junit</artifactId>
```

```
                        <version>3.8.1</version>
                         <scope>test</scope>
            </dependency>
  </dependencies>
  </project>
```

Step 12: Now JAR file is created by deploying Matrixmultiplication.java: The generated JAR file typically contains your compiled Java code, resources, and other necessary files which is stored in the workspace created in step 2.

Step 13: Start all hadoop daemons using the command given below in the terminal:

```
$ start-all.sh
```

Step 14: Create input directory using the command given below:

```
$ hadoop fs -mkdir /user/mm/input
```

Step 15: Now copy the input file 'A.txt' and 'B.txt' to the input directory using the following command.

```
$ hadoop fs -put '/home/hadoop/Sample/A'  /user/mm/input
$ hadoop fs -put '/home/hadoop/Sample/B'  /user/mm/input
```

Sample Input:

A.txt:	**B.txt:**
A 0 0 2	B 0 0 5
A 0 1 3	B 0 1 6
A 0 2 1	B 0 2 7
A 1 0 1	B 1 0 8
A 1 1 4	B 1 1 9
A 1 2 2	B 1 2 1
A 2 0 5	B 2 0 2
A 2 1 6	B 2 1 3
A 2 2 7	B 2 2 4

Step 16: Open mapred-site.xml in nano editor and edit the file with following content.

```
$ nano mapred-site.xml

<property>
<name>yarn.app.mapreduce.am.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop
</value>
</property>
<property>
<name>mapreduce.map.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop</value>
</property>
<property>
<name>mapreduce.reduce.env</name>
<value>HADOOP_MAPRED_HOME=/home/hadoop/hadoop</value>
</property>
```

Step 17: Then, run the java program in hadoop using the command given below:

```
$ hadoop jar '/home/hadoop/Ex3/MM/target/MM-0.0.1-SNAPSHOT.jar'
me/lab/psna/MM/MatrixMultiplication  /user/mm/input   /user/mm/output
```

Step 18: Now, the output from output directory can be read by using the command given below: In Hadoop MapReduce, the part-r-00000 file name is a convention for the output files generated by the reducer tasks as part of the MapReduce job.

```
$ hadoop fs -cat /user/mm/output/part-r-00000

0 0     10
0 1     0
0 2     0
1 0     0
1 1     36
1 2     0
2 0     0
2 1     0
2 2     28
```

Result: Thus map-reduce program for matrix multiplication has been implemented.

Ex. No. 5 **Installation of MongoDB along with practice examples**
Date:

Aim:

To install MongoDB and perform CRUD operations in MongoDB

Description:

NoSQL, which stands for "Not Only SQL," is a category of database systems that diverges from traditional relational database management systems (RDBMS). NoSQL databases are designed to handle large volumes of unstructured or semi-structured data, providing more flexibility and scalability than traditional relational databases. The term "NoSQL" doesn't imply a complete rejection of SQL (Structured Query Language) but rather suggests that these databases offer additional or alternative data storage and retrieval mechanisms beyond what SQL-based systems provide. Here some features of NoSQL databases.

- **Schema-less:** Unlike RDBMS, NoSQL databases are schema-less, meaning they don't require a predefined schema for data. This flexibility allows developers to insert data without the need for a rigid, predefined structure.
- **Scalability:** NoSQL databases are designed to scale horizontally, distributing data across multiple servers or nodes. This horizontal scalability allows them to handle large amounts of data and high traffic loads efficiently.

Types of NoSQL Databases:

- **Document-oriented databases:** Store data in flexible, JSON-like documents. Examples include MongoDB.
- **Key-value stores:** Use a simple key-value pairing for data storage. Examples include Redis and Amazon DynamoDB.
- **Column-family stores:** Organize data into columns instead of rows. Apache Cassandra is an example.
- **Graph databases:** Optimize for storing and querying graph-like structures. Neo4j is an example.

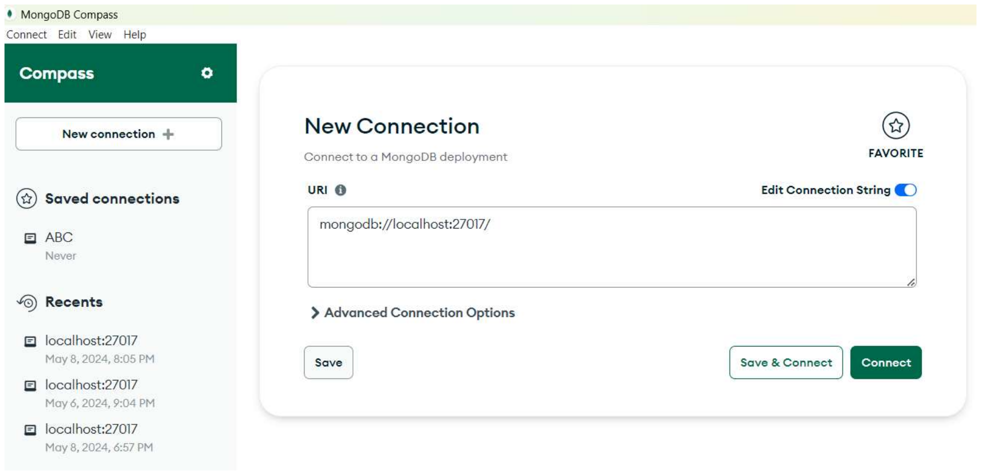
To Create Documents in MongoDB:

MongoDB is a popular open-source, NoSQL, document-oriented database. Unlike traditional relational databases, MongoDB doesn't store data in tables with rows and columns; instead, it stores data in flexible, JSON-like documents. This flexibility allows for easier scalability and handling of unstructured or semi-structured data.

In MongoDB, a collection is a grouping of MongoDB documents. Collections are analogous to tables in relational databases, but with some key differences. Collections don't enforce a schema, meaning documents within a collection can have different fields and structures. This flexibility makes MongoDB well-suited for evolving data models and applications where the schema may change over time.

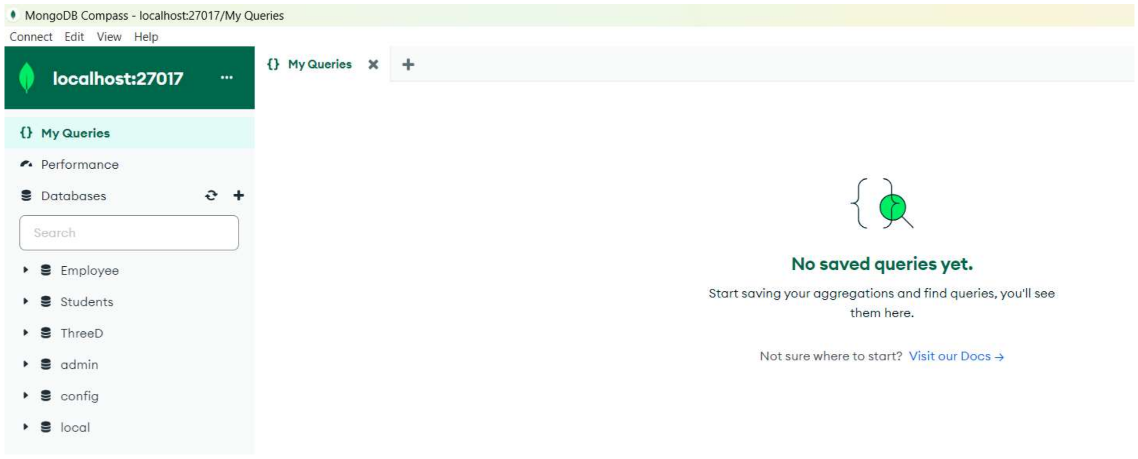
Step1: Download and Install MongoDB Compass using the link
https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-7.0.8-signed.msi

Step2: Once Installed, the following window will appear. Then create a connection with MongoDB using the button ‘Connect’.

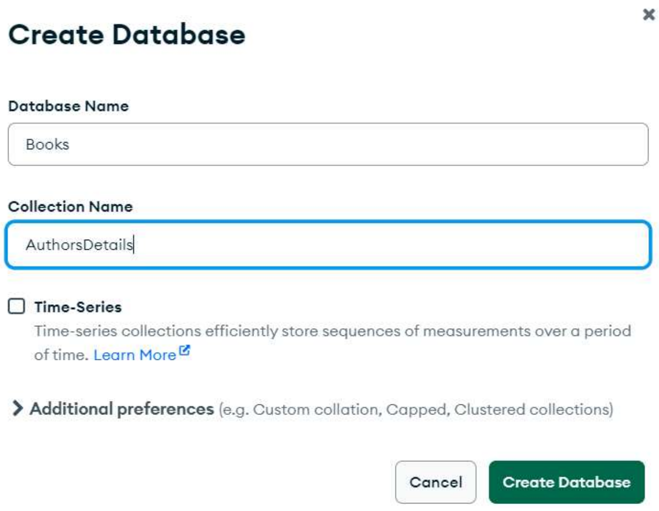


mongodb://localhost:27017/

Step3: After the connection has been created, the following window will be appeared.


MongoDB Compass - localhost:27017/My Queries
Connect Edit View Help
localhost:27017
My Queries
Performance
Databases
Search
Employee
Students
ThreeD
admin
config
local
{} My Queries
No saved queries yet.
Start saving your aggregations and find queries, you'll see them here.
Not sure where to start? Visit our Docs →

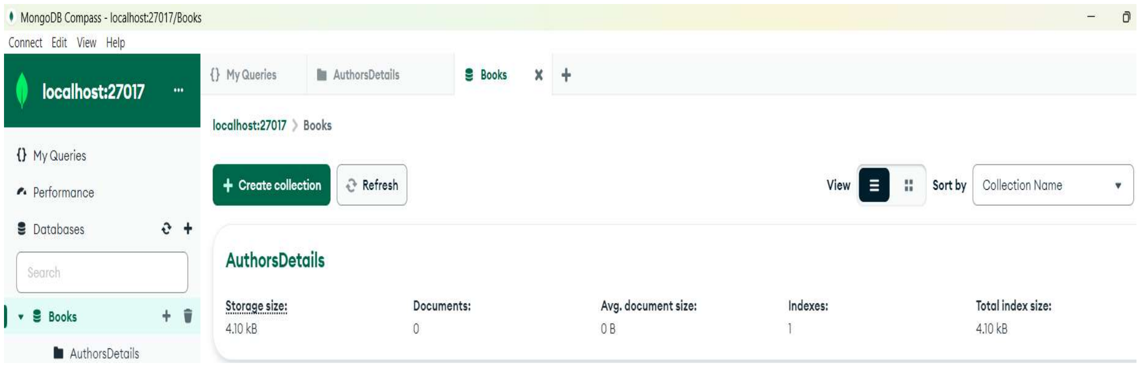
Step4: Using '+' button we can create the databases and collections using the options given below:


Create Database
Database Name
Collection Name
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Database

Database Name: Books

Collection Name: AuthorsDetails

Step5: The created tables shown in the list of databases as shown below:



AuthorsDetails

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
4.10 kB	0	0 B	1	4.10 kB

Step6: Now, download the VS code using the given link and write the python code for interacting with MongoDB.

https://code.visualstudio.com/

Step7: Once installed, search MongoDB for VS Code v1.6.0 extensions and install it.

Step8: After installing MongoDB extension, open it and establish a connection with MongoDB compass (Server) using the URI shown in step 2.

Step9: Use "Create New Playground" option in VScode to write the simple code for creating, storing the data in MongoDB server.

Step10: Simple code for performing CRUD operations in MongoDB.

Creating a New Collection: To create a new collection, createCollection() is used.

```
db.createCollection("AuthorsDetails")
```

Output:

```
{
 "ok": 1
}
```

(i) Create (Insert): To insert new documents into a MongoDB collection, you can use the insert_one() which adds single document to a collection or insert_many() which adds multiple documents to a collection.

```
use('Books')
db.AuthorsDetails.insertMany([
   {
     Name : 'Meera',
     Age  : 36,
     Books :"DBMS"
   },
   {
     Name : 'raja',
     Age  : 39,
    Books :"CNS"
   }
]);
```

Output:

```
{
  "acknowledged": true,
  "insertedIds": {
    "0": {
      "$oid": "663ce29016fe1a344e17bffe"
    },
    "1": {
      "$oid": "663ce29016fe1a344e17bfff"
    }
  }
}
```

- "acknowledged": true: This indicates that the operation was acknowledged by the MongoDB server, meaning it was successfully executed.
- "insertedIds": This is an object containing the IDs of the documents that were inserted into the collection.
- "0" and "1": These are the keys representing the index positions of the inserted documents. In this case, two documents were inserted, so they are indexed starting from 0.
- "$oid": This is a BSON type for ObjectID, MongoDB's default type for document IDs.
- "663ce29016fe1a344e17bffe" and "663ce29016fe1a344e17bfff": These are the actual ObjectIDs assigned to the inserted documents.

(ii) Read (Find): To retrieve documents from a MongoDB collection, you can use the find() which retrieves all the documents that match the query or findOne() method which retrieves single document that match the query.

```
use('Books')
db.AuthorsDetails.find({Books : 'CNS'})
```

Output: It will retrieve the details of authors who has written the book ‘CNS’.

```
[
  {
    "_id": {
      "$oid": "663ce29016fe1a344e17bfff"
    },
    "Name": "raja",
    "Age": 39,
    "Books": "CNS"
  }
]
```

(iii) Update: To update existing documents in a MongoDB collection, you can use the updateOne() or updateMany() or replaceOne().

updateOne() or updateMany() methods will update the specific fields and any fields that are not part of the update query will remain unchanged.

replaceOne() method will replace the entire document and any fields that are not part of the replace query will be removed.

```
use('Books')
db. AuthorsDetails.updateOne(
   {Name : "raja"},
   {$set : {Books : 'CN'}}
);
```

Output:

```
{
  "acknowledged": true,
  "insertedId": null,
  "matchedCount": 1,
  "modifiedCount": 1,
  "upsertedCount": 0
}
```

After Performing updateOne operation, the document will be

```
[
  {
    "_id": {
      "$oid": "66f654d1a0c2451afa9f0bcf"
    },
    "Name": "raja",
    "Age": 39,
    "Books": "CN"
  }
]
```

- "insertedId": null: This field indicates that no new document was inserted as part of this update operation.
- "matchedCount": 1: This indicates that one document matched the specified filter criteria for the update operation. MongoDB found a document in the collection that matched the query.
- "modifiedCount": 1: This indicates that one document was successfully modified (updated) as part of the operation. This means that the update operation found a matching document and successfully applied the update.
- "upsertedCount": 0: This indicates that no new documents were inserted as a result of an upsert operation. An upsert operation would insert a new document if no matching document is found based on the provided filter criteria.

```
use('Books')
db. AuthorsDetails.replaceOne(
    {Name : "raja"},
    {Books : 'CN'}
);
```

After performing replaceOne(), the document will be

```
[
  {
    "_id": {
      "$oid": "66f654d1a0c2451afa9f0bcf"
    },
    "Books": "CN"
  }
]
```

It removes other two fields such as Name and Age fields that are not specified in the replaceOne() query.

(iv) Delete: To remove documents from a MongoDB collection, you can use the delete_one() or delete_many() methods.

```
use('Books')
db. AuthorsDetails.deleteOne(
    {Name : "raja"},
    {$set : {Books : 'CN'}}
);
```

Output:

```
{
  "acknowledged": true,
  "deletedCount": 1
}
```

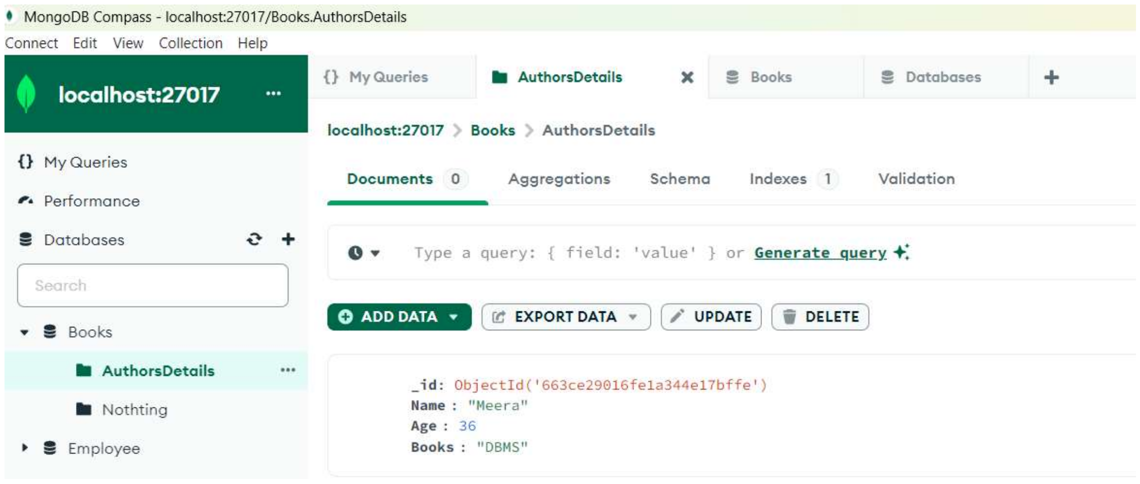
Removing Entire Collection: To remove entire collection, drop () can be used.

```
use("Books")
 db.AuthorDetails.drop()
```

Output:

```
true
```

Step 11: Once insertion, deletion and update operations are performed, the changes will be reflected in MongoDB (server) as shown below:



```
_id: ObjectId('663ce29016fe1a344e17bffe')
Name : "Meera"
Age : 36
Books : "DBMS"
```

Result:

Thus, MongoDB has been installed and CRUD operations have been performed.

Ex. No.: 6 **Installation of Spark along with Practice Examples**
Date

Aim:

To download and install spark and run simple spark application.

Introduction:

Apache Spark is an open-source, distributed computing system primarily used for big data processing, analytics, and machine learning. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. While Hadoop uses a disk-based system (HDFS) for storing and processing data, which can be slow, Spark processes data in-memory, making it much faster, especially for repeated operations or iterative tasks. Spark is also more versatile, as it supports batch processing, real-time data streaming, machine learning, and graph processing in one unified platform, whereas Hadoop focuses mostly on batch processing. Additionally, Spark's user-friendly APIs in languages like Python and Java make it easier to work with compared to Hadoop's more complex MapReduce model. Overall, Spark is beneficial because it provides faster processing, supports real-time applications, and is easier to use, making it a more efficient choice for many data-intensive tasks.

Installation Procedure:

Step 1 – Installing Java:

Java is required for Spark installation because Apache Spark is written in Scala, a language that runs on the Java Virtual Machine (JVM). Since Scala is built on top of the JVM, Spark relies on the JVM to execute its code. Java provides the necessary environment (runtime) for Spark to operate and manage system resources, memory, and other low-level functionalities.

```
$ sudo apt update

$ sudo apt install openjdk-11-jdk
```

Once installed, verify the installed version of Java with the following command:

```
$ java -version
```

The following output can be shown:

```
$ dirname $(dirname $(readlink -f $(which java)))

OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.20.04)

OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode, sharing)
```

Step 2 – Install Apache Spark:

Download Spark from the official website: https://dlcdn.apache.org/spark/spark-3.5.3/spark-3.5.3-bin-hadoop3.tgz

Spark can interact with Hadoop's HDFS (Hadoop Distributed File System) and YARN (Yet Another Resource Negotiator) for data storage and resource management. The mention of "Hadoop 3" in URL means that this version of Spark is compiled to work seamlessly with Hadoop 3.x environments.

Once you've downloaded the file, you can unzip it to a folder on your hard drive.

```
$ tar -xvf  '/home/hadoop/Downloads/spark-3.5.3-bin-hadoop3.tgz'
```

Rename the extracted folder using mv command to remove version information. This is an optional step, but if you don't want to rename, then adjust the remaining configuration paths.

```
$ mv spark-3.5.3-bin-hadoop3 spark
```

Next, you will need to configure Spark and Java Environment Variables in. bashrc file. Open the ~/.bashrc file in your favorite text editor:

```
$ nano ~/.bashrc
```

Append the below lines to the file. Then save the file and close it.

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export SPARK_HOME=/home/hadoop/spark
export PATH=$PATH:$SPARK_HOME/sbin:$SPARK_HOME/bin
```

Then, activate the environment variables with the following command:

```
$ source ~/.bashrc
```

Then, start the master node using start-master.sh which is a key script used to launch and manage the master node in Spark's standalone cluster mode, allowing for efficient resource distribution and job scheduling across a Spark cluster.

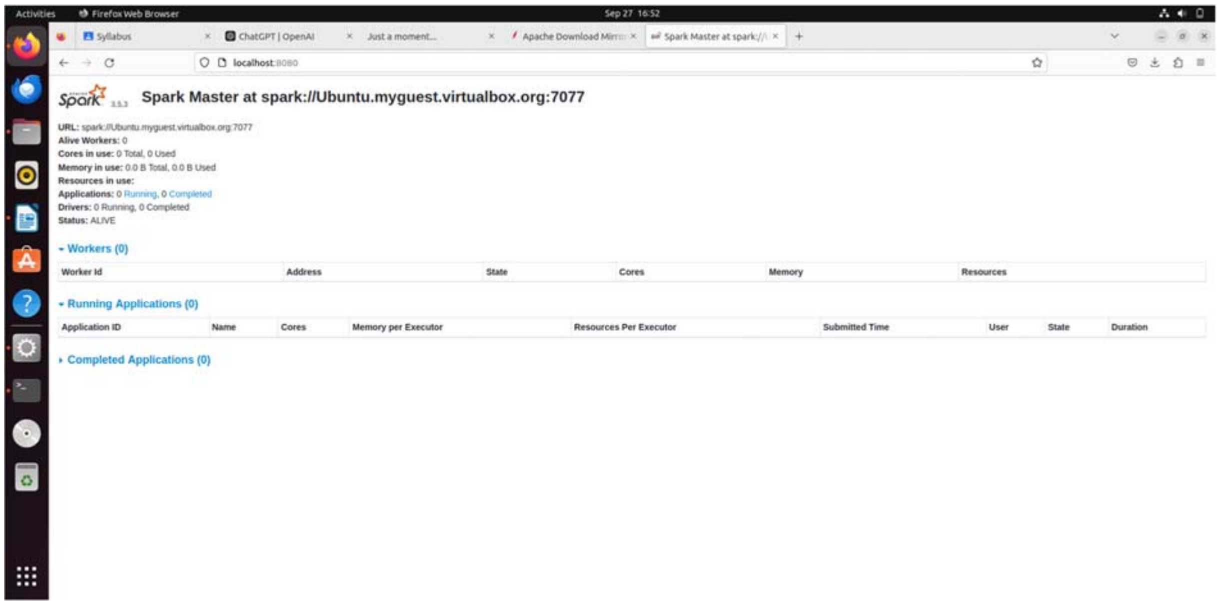
```
$ start-master.sh

starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/spark/logs/spark-hadoop-org.apache.spark.deploy.master.Master-1-Ubuntu.out
```

Then start the worker node using start-slave.sh which is a script used to start worker node (or "slave") in Spark's standalone cluster mode. A worker node is responsible for executing tasks assigned by the Spark master node. The worker node connects to the Spark master located at spark://localhost:7077. Once the worker node is connected, the master will manage and distribute jobs and tasks to this worker. After the worker node connects, you can view it on the Spark Web UI (typically at http://localhost:8080 for the master) under the "Workers" section.

```
$ start-slave.sh spark://localhost:7077

starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/spark/logs/spark-hadoop-org.apache.spark.deploy.worker.Worker-1-Ubuntu.out
```

After starting the master node, open a web browser and type http://localhost:8080

It is the default web interface (Web UI) for the Apache Spark master node. When you start the Spark master using the start-master.sh script, Spark launches a web-based dashboard that runs on port 8080 of the local machine (localhost). It’s a helpful tool for tracking job progress, understanding resource usage, and identifying potential issues with your Spark applications.



Spark Master at spark://Ubuntu.myguest.virtualbox.org:7077

URL: spark://Ubuntu.myguest.virtualbox.org:7077
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (0)

Worker Id	Address	State	Cores	Memory	Resources

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (0)

Step 3 – Simple Spark Application

If you want to use Spark with Python (PySpark), make sure Python is installed on your system. You can install it with:

```
$ sudo apt-get install python3
```

Then, install pyspark using pip command. PySpark is an interface for Apache Spark in Python, allowing developers to use Spark's powerful data processing capabilities while leveraging Python's simple syntax. It enables efficient handling of large datasets, integration with machine learning libraries, and the ability to perform distributed computing tasks easily.

```
$ pip install pyspark
```

Open nano editor with file name as shown below to write te program for performing word count problem in spark.

```
$ nano simpleprogram.py
```

simpleprogram.py:

```
spark = SparkSession.builder \
    .appName("Word Count") \
    .master("local") \
    .getOrCreate()
```

```
# Read the text file
text_file = spark.read.text('/home/hadoop/input')

# Split each line into words, flatten the list, and count the occurrences
word_counts = text_file.selectExpr("explode(split(value, ' ')) as word") \
                    .groupBy("word") \
                    .count() \
                    .orderBy("count", ascending=False)

# Show the results
word_counts.show()

# Stop the Spark session
spark.stop()
```

In the above program,

- SparkSession: This line creates a Spark session, which is the entry point to using Spark's functionality.
- appName("Word Count"): Sets the name of the application to "Word Count".
- master("local"): Specifies that the application will run locally on your machine, using all available cores.
- getOrCreate(): This method creates a new Spark session if one doesn't exist or returns the existing session.
- text_file.selectExpr("explode(split(value, ' ')) as word") : This line processes each line of the input DataFrame
- split(value, ' '): Splits each line (string) into a list of words using spaces as separators.
- explode(...): Converts each element in the list into a separate row. So, if a line has multiple words, each word will be in its own row.
- as word: Renames the resulting column to word.
- .groupBy("word") : It organizes the data so that all rows with the same word are collected together. For example, if Apple appears multiple times, all occurrences of Apple will be in one group.
- .count(): After grouping, it calculates how many times each unique word appears in the DataFrame. The result will show the word along with its count.
- .orderBy("count", ascending=False): It arranges the resulting DataFrame in descending order, so the words with the highest counts appear first. This makes it easy to see which words are the most common.
- spark.stop() stops the Spark session, releasing resources allocated to the application.

Input.txt:

```
Apple
Banana
Cat
Apple
```

Then, run the spark application using spark-submit simpleprogram.py which is used to submit a Spark application to a Spark cluster for execution. Spark reads the application code and executes it. This could involve reading data, processing it, and writing results

```
$ spark-submit simpleprogram.py


24/09/27 16:17:17 INFO CodeGenerator: Code generated in 32.404007 ms
+------+-----+
|  word|count|
+------+-----+
| Apple|    2|
|Banana|    1|
|   Cat|    1|
+------+-----+
24/09/27 16:17:17 INFO SparkContext: SparkContext is stopping with exitCode 0.

24/09/27 16:17:17 INFO SparkUI: Stopped Spark web UI at
http://[2409:4072:6e10:5a5a:a250:9452:91a7:1b96]:4040

24/09/27 16:17:18 INFO ShutdownHookManager: Deleting directory /tmp/spark-
3cec4b87-ba0d-4bb6-85cb-2617486d
```

Result:

Thus, spark has been installed and simple application using spark has been executed successfully.

**Ex. No.:7	Installation of HBase, Installing thrift along with Practice examples**
Date:

Aim:
To install HBase and Thrift and practice basic HBase commands in a Hadoop Environment.

Description:
HBase is an open-source, distributed, and scalable NoSQL database that is part of the Apache Hadoop ecosystem. It is designed to store and manage large volumes of sparse data, which makes it well-suited for handling massive amounts of data with high write and read throughput. Here are some key features and characteristics of HBase:

Features of HBase

- **Columnar Store:** HBase uses a column-family-based data model for efficient storage.
- **Scalability:** HBase can easily expand by adding more servers for handling large datasets.
- **Consistency:** It ensures strong consistency for data within a single row.
- **Automatic Sharding:** Data is split into regions for even distribution and parallel processing.
- **Master-Slave Architecture:** HBase has a master node for cluster management and region servers for data handling.
- **High Write Throughput:** It excels in write-heavy applications like time-series data.
- **Schema-on-Read:** HBase allows flexible schema adaptation without predefined structures.
- **Integration with Hadoop:** It seamlessly integrates with Hadoop tools for big data processing.
- **Compression and Bloom Filters:** Reduces storage and enhances query performance.
- **High Availability:** Ensures data accessibility through automatic failover mechanisms.

HBASE Installation Procedure:

Step 1: Access your Ubuntu command line and download the compressed Hbase files using wget command.

```
$ wget https://dlcdn.apache.org/hbase/2.5.6/hbase-2.5.6-hadoop3-bin.tar.gz
```

Step 2: Once the download process is complete, Untar the compressed Hbase package.

```
$ tar xzf hbase-2.5.6-hadoop3-bin.tar.gz
```

Step 3: The HBase files are now located in the apache-hive-3.1.2-bin directory. It can be seen by using ls command.

```
$ ls
hbase-2.5.6-hadoop3-bin  Demo      Desktop    Downloads  Ex4     hadoop-3.3.2.tar.gz
metastore_db  Pictures  run     snap      Videos
```

Step 4: Insert JAVA PATH in the hbase-env.sh file located in the **hbase-2.5.6-hadoop3-bin** folder.

```
# JDK11+ JShell
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Step 5: Append the following HBase environment variables to the end of the .bashrc file.

```
$ nano ~/.bashrc

export HBASE_HOME=/home/hadoop/hbase-2.5.6-hadoop3
export PATH=$PATH:$HBASE_HOME/bin
```

Step 6: Save and exit the .bashrc file once you add the Hbase variables. Apply the changes to the current environment with the following command.

```
$ source ~/.bashrc
```

Step 7: Open hbase-site.xml and add the following content.

```
<property>
<name>hbase.rootdir</name>
<value>file:///home/hadoop/HBASE/hbase</value>
</property>

<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/home/hadoop/HBASE/zookeeper</value>
</property>
```

Step 8: Start Hbase Master using the command given below in the terminal.

```
$ cd hbase-2.5.6-hadoop3/bin

hadoop@Ubuntu:~/hbase-2.5.6-hadoop3/bin$ start-hbase.sh

SLF4J: Class path contains multiple SLF4J bindings.
……………………………………………………………………………………..

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

running master, logging to /home/hadoop/hbase-2.5.6-hadoop3/logs/hbase-hadoop-master-
Ubuntu.out

$ jps

16546 HMaster

16687 Jps
```

Step 9: Start Hbase Shell by using the command given below:

```
$ hbase shell

SLF4J: Class path contains multiple SLF4J bindings.
…………………………………………………………..

HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.5.6-hadoop3, r6bac842797dc26bedb7adc0759358e4c8fd5a992, Sat Oct 14 23:50:27
PDT 2023
Took 0.0027 seconds
hbase:001:0>
```

Step 10 : In this hbase shell, the tables can be created by using ‘create command’.

```
hbase:001:0> create 'emp', 'adata','bdata'
 // It shows following output.

2023-10-29 11:45:58,772 INFO  [main] client.HBaseAdmin
(HBaseAdmin.java:postOperationResult(3591)) - Operation: CREATE, Table
Name: default:emp, procId: 9 completed

Created table emp

Took 3.8332 seconds

=> Hbase::Table – emp

hbase:002:0> put 'emp', 'row1', 'adata:name', 'John'
hbase:003:0> scan ‘emp’

//It shows the following output

ROW                        COLUMN+CELL

 row1                    column=adata:name, timestamp=1698582937405, value=John

1 row(s)

hbase:004:0> delete 'emp', 'row1' // It deletes the row1

hbase:005:0> scan ‘emp’

//It shows the following output

ROW                      COLUMN+CELL

0 row(s)

hbase:005:0> disable ‘emp’

hbase:006:0> drop ‘emp’

//It shows the following output

Took 2.3456 seconds
```

Thrift API:
Apache Thrift is a software framework used in Apache HBase to facilitate communication between the HBase client and the HBase server. It acts as a serialization and remote procedure call (RPC) framework, enabling clients to interact with HBase's distributed database system.

Thrift Installation Procedure:

Step 1: Download the compressed Thrift files using wget command.

```
$ wget https://dlcdn.apache.org/thrift/0.19.0/thrift-0.19.0.tar.gz
```

Step 2: Once the download process is complete, Untar the compressed thrift package.

```
$ tar xzf thrift-0.19.0.tar.gz
```

Step 3: Now install following packages to include essential development libraries and tools required to build and compile Thrift and its associated components.

```
$ sudo apt update -y

$ sudo apt-get install libboost-dev libboost-test-dev libboost-program-options-dev libevent-dev automake libtool flex bison pkg-config g++ libssl-dev


sudo apt-get install automake bison flex g++ git libboost-all-dev libevent-dev libssl-dev libtool make pkg-config


sudo apt-get install automake bison flex g++ git libboost-all-dev libevent-dev libssl-dev libtool make pkg-config
```

Step 4: Now, locate bootstrap.sh file in the 'thrift-0.19.0' folder and run the following commands: It is a common step in building open-source software from source code. It makes the project ready for compilation on the specific system and ensures that all necessary components are in place, making it easier to configure and build Thrift correctly.

```
$ ./bootstrap.sh
```

Step 5: In configure command, the output shows which programming language libraries and features of Thrift will be included or excluded during the build process. In this case, it's configuring Thrift to build C++, Python, and Python 3 libraries while excluding others like PHP, Ruby, Rust, and Swift.

```
$ ./configure

// It shows following output

thrift 0.19.0

Building C (GLib) Library .... : no
Building C++ Library ......... : yes
Building PHP Library ......... : no
Building Python Library ...... : yes
Building Py3 Library ......... : yes
Building Ruby Library ........ : no
Building Rust Library ........ : no
Building Swift Library ....... : no
```

Step 6: Run 'sudo make' would perform the compilation of the Thrift source code using the make build automation tool with superuser privileges granted by sudo.

```
$ sudo make
// It shows following output
make  all-recursive
…………………….
make[2]: Entering directory '/home/hadoop/thrift-0.19.0'
make[2]: Leaving directory '/home/hadoop/thrift-0.19.0'
make[1]: Leaving directory '/home/hadoop/thrift-0.19.0'
```

Step 7: once you've successfully installed Thrift using the ‘sudo make install command’, the Thrift software becomes available and usable by any user on your system.

```
$ sudo make install
// It shows following output
Making install in compiler/cpp
…………………….
make[2]: Entering directory '/home/hadoop/thrift-0.19.0'
make[2]: Leaving directory '/home/hadoop/thrift-0.19.0'
make[1]: Leaving directory '/home/hadoop/thrift-0.19.0'

make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/hadoop/thrift-0.19.0'
make[1]: Leaving directory '/home/hadoop/thrift-0.19.0'
```

Step 8: Check the successful installation of thrift by using ‘version’ command.

```
$ thrift -version
Thrift version 0.19.0
```

Integration of HBase and Thrift

Step 1: The next step is to intstall ‘pip’ the package manager to install python packages and libraries. ‘pip3 install thrift’ is a command used to install the Thrift Python library using the pip package manager for Python 3.

```
$ sudo apt update -y
$ sudo apt install python3-pip
$ pip3 install thrift
```

Step 2: HappyBase is a Python library that provides a convenient and high-level interface to interact with HBase, a NoSQL database. It can be installed by using the command given below:

```
$ pip3 install happybase
```

Step 3: Locate hbase-2.5.6-hadoop3/bin folder and start the Hbase Master using the command given below.

```
$ start-hbase.sh

// It shows following output

running master, logging to /home/hadoop/hbase-2.5.6-hadoop3/logs/hbase-hadoop-
master-Ubuntu.out
```

Step 4: Now, Start the Thrift Server by using the command given below:

```
$ hbase-daemon.sh start thrift

// It shows following output

running thrift, logging to /home/hadoop/hbase-2.5.6-hadoop3/logs/hbase-hadoop-
thrift-Ubuntu.out

$jps

27673 HMaster
28205 Jps
28063 ThriftServer
```

Step 5: Open python terminal and type the following content which retrieve the tables created in Hbase using Thrift.

'import happybase'- HappyBase is a Python client library for Apache HBase, which allows you to interact with HBase databases from your Python code.

'connection = happybase.Connection('127.0.0.1', 9090)'- This establishes a connection to your local HBase instance running on IP address 127.0.0.1 and port 9090.

'connection.tables()' - is a method provided by HappyBase that queries the HBase database for the list of tables and returns them as a list.

```
$ python3

>>>import happybase

>>>connection = happybase.Connection('127.0.0.1',9090)

>>> print(connection.tables())

// It shows the list of tables created in HBase

[b 'emp']

//You have one table in your HBase instance, and it is named "emp." When you call
connection.tables(), returns a list of table names as bytes. In Python 3, byte strings
are represented as bytes objects with a prefix 'b'.
```

Result:

Thus, installation of HBase and Thrift have been done and basic HBase commands were executed in Hadoop Environment.