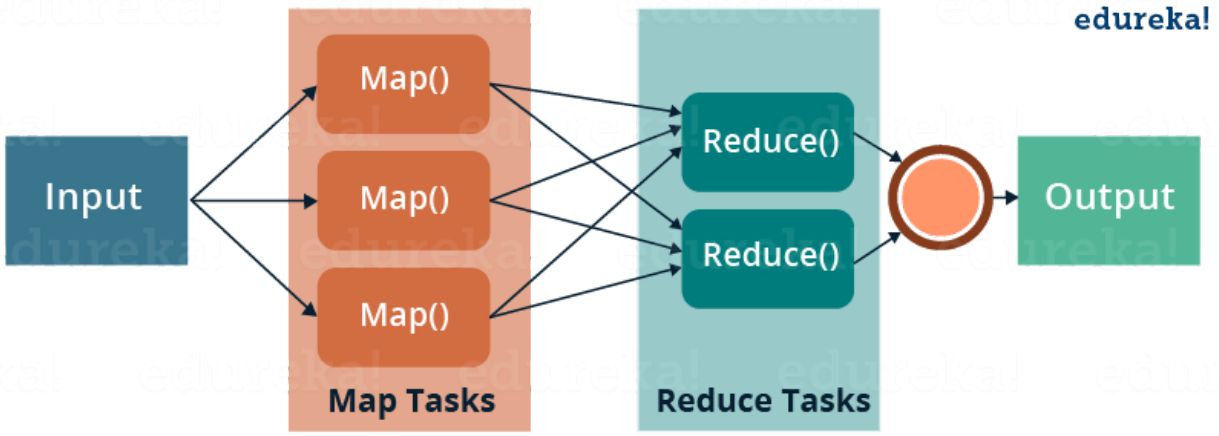
EXPERIMENT 5

WORDCOUNT Program in Hadoop

THEORY

What is MapReduce?



MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.

- MapReduce consists of two distinct tasks – Map and Reduce.
- As the name MapReduce suggests, the reducer phase takes place after the mapper phase has been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.
- Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

Let us understand more about MapReduce and its components. MapReduce majorly has the following three Classes. They are,

Mapper Class

The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop's Mapper store saves this intermediate data into the local disk.

- Input Split

It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.

- RecordReader

It interacts with the Input split and converts the obtained data in the form of Key-Value Pairs.

Reducer Class

The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.

Driver Class

The major component in a MapReduce job is a Driver Class. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

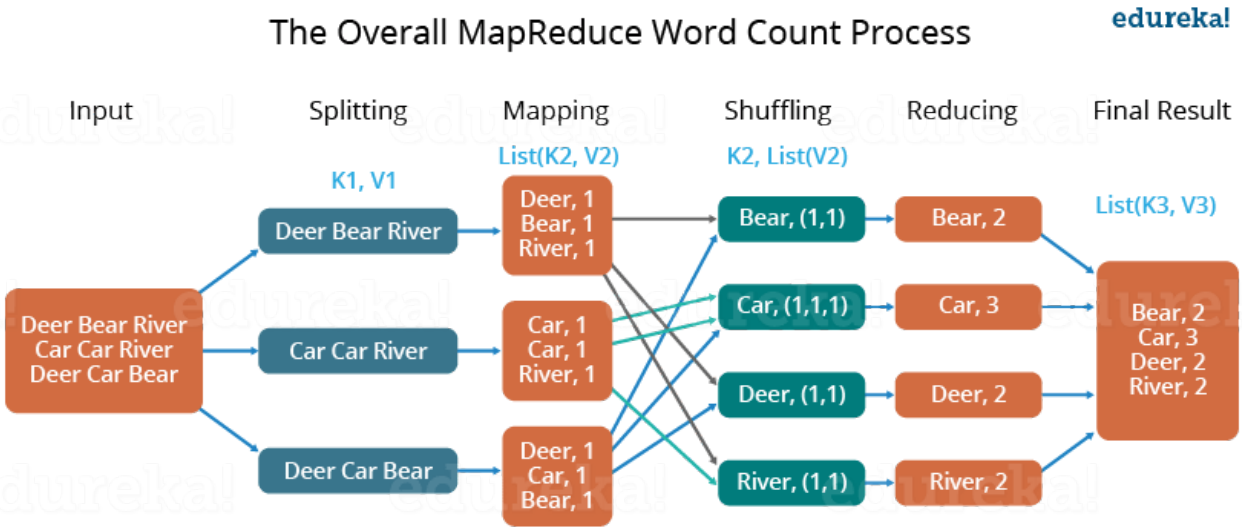
Meanwhile, you may go through this MapReduce Tutorial video where our expert from Hadoop online training has discussed all the concepts related to MapReduce has been clearly explained using examples:

MapReduce Tutorial: A Word Count Example of MapReduce

Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pairs will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.

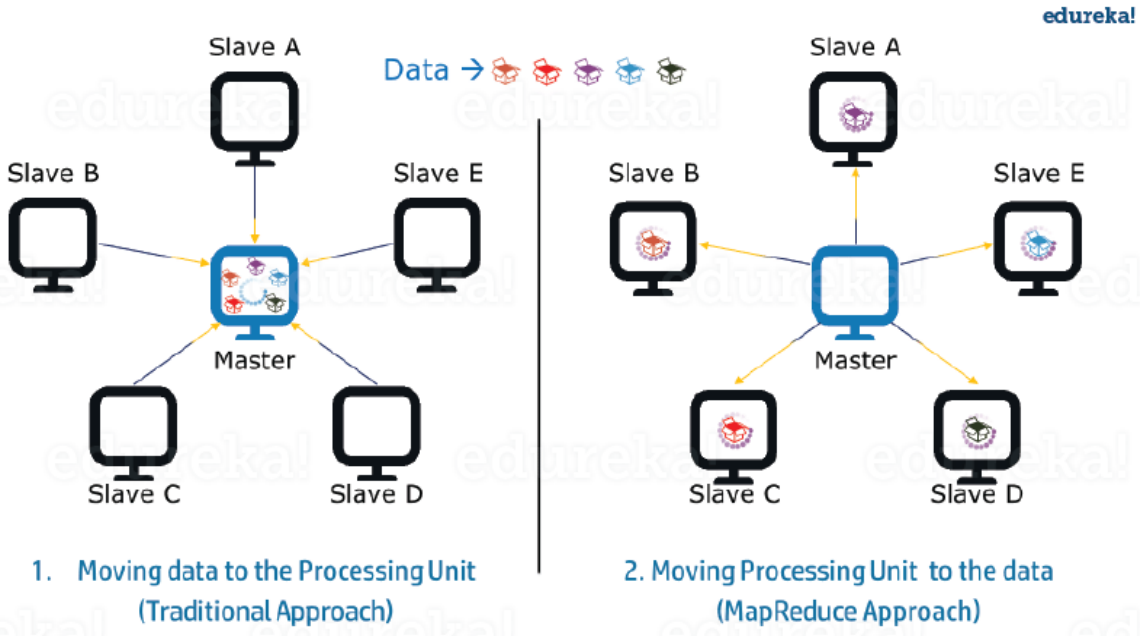
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

MapReduce Tutorial: Advantages of MapReduce

The two biggest advantages of MapReduce are:

1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on the Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount as shown in the figure below



2. Data Locality:

Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to

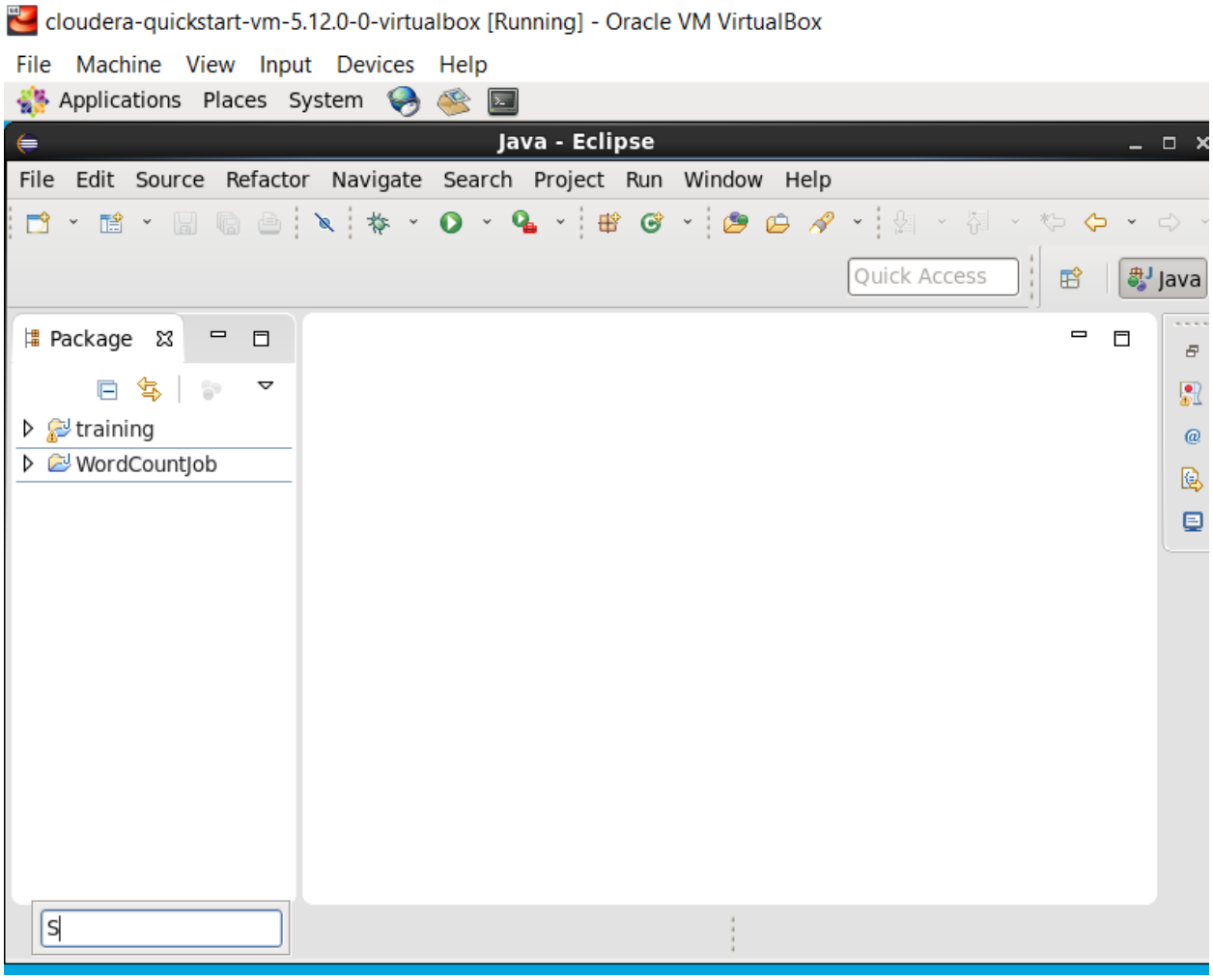
the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- The master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image, the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

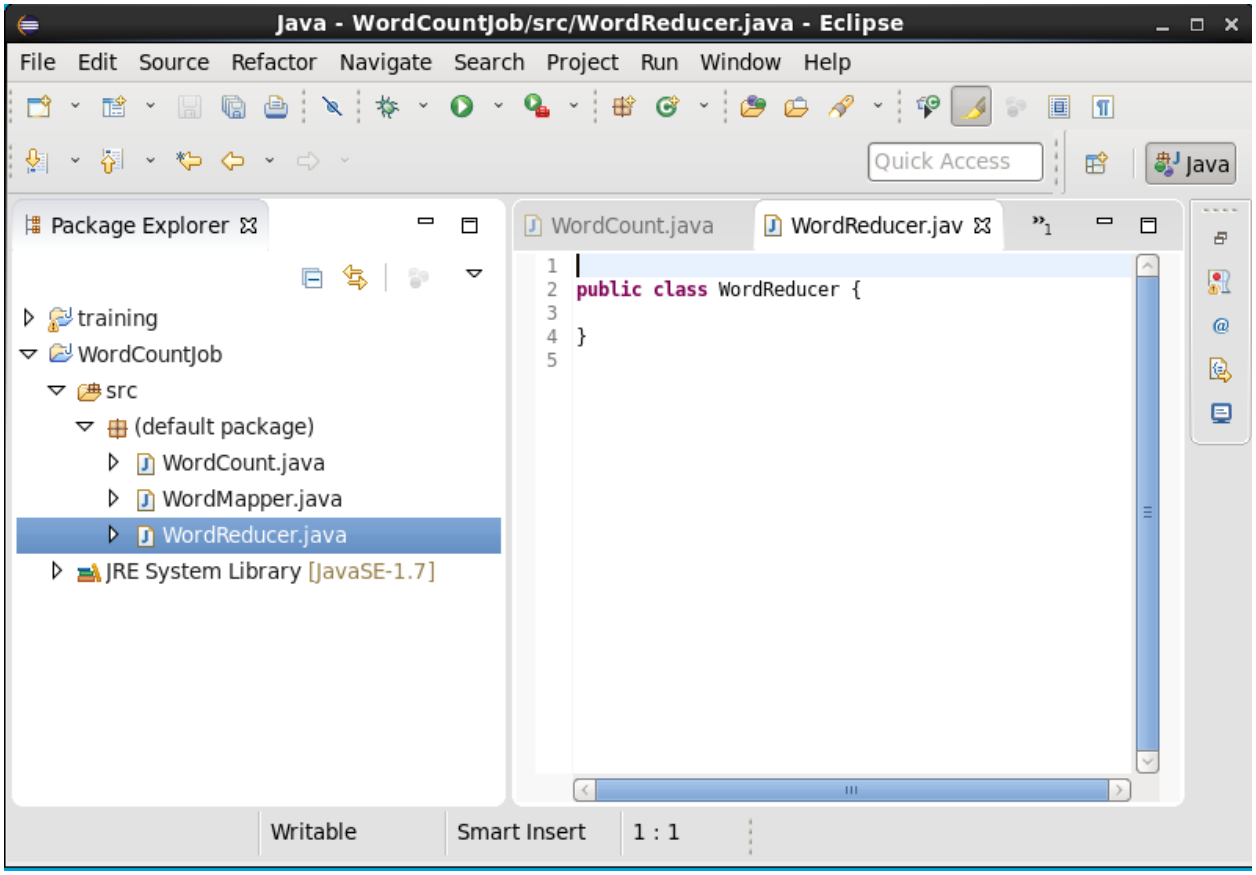
- It is very cost-effective to move the processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

STEP 1 :
Create New Java Project in Eclipse
Name project : WordCounJob
click : finish



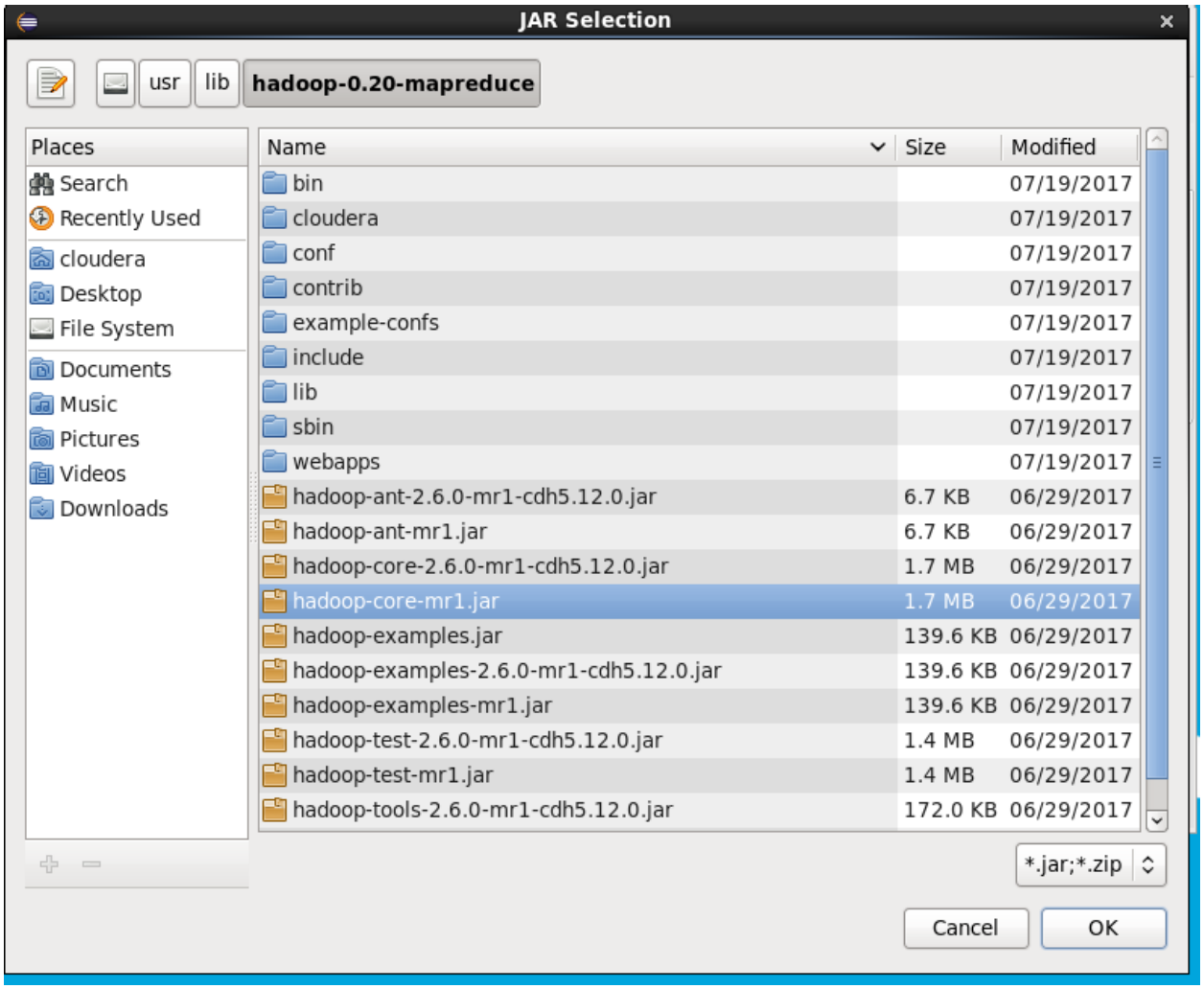
STEP 2:

Right Click on project -> New -> Class
create class files as
Name : WordCount
Name : WordMapper
Name : WordReducer

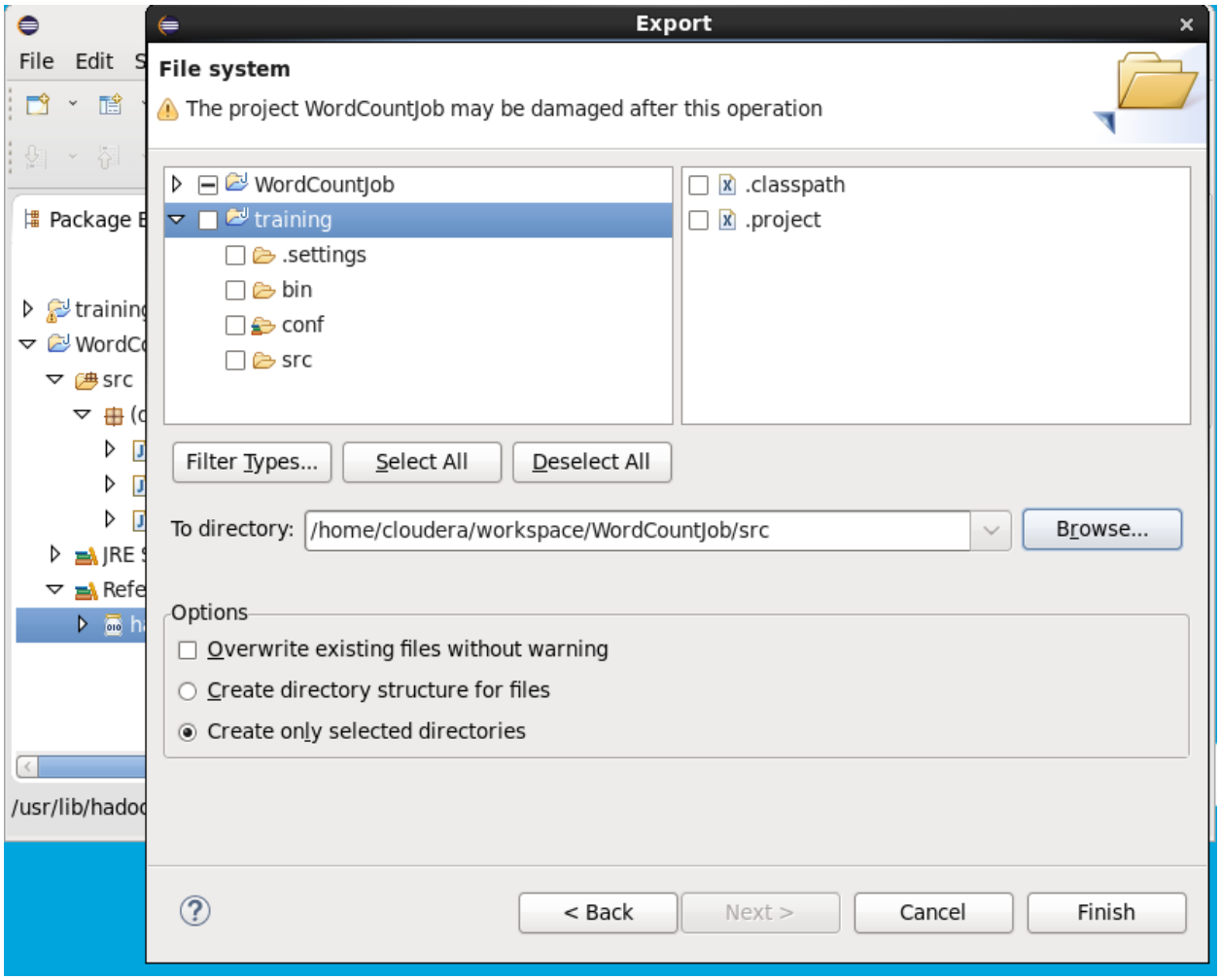


STEP 3 :

Right Click Project -> build path -> add expertnal archievs -> filesystem -> usr -> lib -> hadoop-0.20 -> hadoop-core.jar
click add



STEP 4 : export the jar file in same folder (training/workspace/WordCountJob/src)



STEP 5: open terminal
[training@localhost ~]$ ls -l

```
[cloudera@quickstart ~]$ ls -l
total 252
drwxrwxr-x 2 cloudera cloudera  4096 Aug 26 03:29 BDApracs
-rwxrwxr-x 1 cloudera cloudera  5387 Jul 19  2017 cloudera-manager
-rwxrwxr-x 1 cloudera cloudera  9964 Jul 19  2017 cm_api.py
drwxrwxr-x 2 cloudera cloudera  4096 Aug 28 01:24 data
-rw-rw-r-- 1 cloudera cloudera    67 Aug 27 00:31 demo.txt
drwxrwxr-x 3 cloudera cloudera  4096 Aug 26 03:44 Desktop
drwxrwxr-x 4 cloudera cloudera  4096 Jul 19  2017 Documents
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Downloads
drwxrwsr-x 9 cloudera cloudera  4096 Aug 29 04:02 eclipse
-rw-rw-r-- 1 cloudera cloudera 53655 Jul 19  2017 enterprise-deployment.json
-rw-rw-r-- 1 cloudera cloudera 50515 Jul 19  2017 express-deployment.json
-rwxrwxr-x 1 cloudera cloudera  5007 Jul 19  2017 kerberos
drwxrwxr-x 2 cloudera cloudera  4096 Jul 19  2017 lib
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Music
drwxrwxr-x 2 cloudera cloudera  4096 Aug 26 23:29 myfirstdata
-rwxrwxr-x 1 cloudera cloudera  4228 Jul 19  2017 parcels
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Pictures
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Public
-rw-rw-r-- 1 cloudera cloudera 18353 Aug 26 23:46 registercopy.java
-rw-rw-r-- 1 cloudera cloudera 18281 Aug 26 23:36 register.java
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Templates
drwxrwxr-x 2 cloudera cloudera  4096 Aug 29 00:11 test
drwxrwxr-x 2 cloudera cloudera  4096 Aug 27 00:33 training
drwxr-xr-x 2 cloudera cloudera  4096 Aug 26 03:21 Videos
drwxrwxr-x 6 cloudera cloudera  4096 Aug 29 05:26 workspace
[cloudera@quickstart ~]$
```

STEP 6: create sample file
[training@localhost ~]$ cat WCFile.txt
Hello we are doing BDA pracs
Hello I'm a student
Put the file in Hadoop

```
[cloudera@quickstart workspace]$ cat WCFile.txt
Hello we are doing BDA pracs
Hello I'm a student
[cloudera@quickstart workspace]$
```

STEP 7 : Put the content of sample.txt to samplehadoop.txt
[training@localhost ~]$ hadoop fs -put WCFile.txt WCFile.txt
[training@localhost ~]$ hadoop fs -ls

```
[cloudera@quickstart workspace]$ hadoop fs -put WCFile.txt WCFile.txt
[cloudera@quickstart workspace]$ hadoop fs -ls
Found 4 items
-rw-r--r--   1 cloudera cloudera         49 2022-08-29 05:32 WCFile.txt
drwxr-xr-x   - cloudera cloudera          0 2022-08-26 03:41 hadoop
drwxr-xr-x   - cloudera cloudera          0 2022-08-26 04:08 raunak
drwxr-xr-x   - cloudera cloudera          0 2022-08-26 04:11 test.txt
[cloudera@quickstart workspace]$
```

STEP 8 : change the directory to
[training@localhost ~]$ cd /home/training/workspace/WordCount/src

```
[cloudera@quickstart ~]$ cd workspace
[cloudera@quickstart workspace]$ cd WordCount
[cloudera@quickstart WordCount]$ cd src
[cloudera@quickstart src]$ ls
WCDriver.java  WCMapper.java  WCReducer.java
[cloudera@quickstart src]$
```

STEP 9: Listing the contents of that directory
[cloudera@quickstart src]$ ls
WCDriver.java WCMapper.java WCReducer.java

```
[cloudera@quickstart src]$ ls
WCDriver.java  WCMapper.java  WCReducer.java
[cloudera@quickstart src]$
```

STEP 10 : Run the wordcount program and collect the output in WCOutput
[cloudera@quickstart workspace]$ hadoop jar WordCount.jar WCDriver WCFile.txt WCOutput

```
[cloudera@quickstart workspace]$ hadoop jar WordCount.jar WCDriver WCFile.txt WC
Output
22/08/29 05:37:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
22/08/29 05:37:50 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
22/08/29 05:37:50 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
22/08/29 05:37:51 INFO mapred.FileInputFormat: Total input paths to process : 1
22/08/29 05:37:51 INFO mapreduce.JobSubmitter: number of splits:2
22/08/29 05:37:51 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_16
61770860486_0001
22/08/29 05:37:52 INFO impl.YarnClientImpl: Submitted application application_16
61770860486_0001
22/08/29 05:37:53 INFO mapreduce.Job: The url to track the job: http://quickstar
t.cloudera:8088/proxy/application_1661770860486_0001/
22/08/29 05:37:53 INFO mapreduce.Job: Running job: job_1661770860486_0001
22/08/29 05:38:08 INFO mapreduce.Job: Job job_1661770860486_0001 running in uber
 mode : false
22/08/29 05:38:08 INFO mapreduce.Job:  map 0% reduce 0%
22/08/29 05:38:21 INFO mapreduce.Job:  map 50% reduce 0%
22/08/29 05:38:22 INFO mapreduce.Job:  map 100% reduce 0%
22/08/29 05:38:28 INFO mapreduce.Job:  map 100% reduce 100%
22/08/29 05:38:29 INFO mapreduce.Job: Job job_1661770860486_0001 completed succe
ssfully
22/08/29 05:38:30 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=115
                FILE: Number of bytes written=375712
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=290
```

STEP 11 : chk the output file

[cloudera@quickstart workspace]$ hadoop fs -ls WCOutput

```
[cloudera@quickstart workspace]$ hadoop fs -ls WCOutput
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2022-08-29 05:38 WCOutput/_SUCCESS
-rw-r--r--   1 cloudera cloudera         61 2022-08-29 05:38 WCOutput/part-00000
```

STEP 12: Displaying the Output
[*cloudera@quickstart workspace*]$ hadoop fs -cat WCOutput/part-00000

```
[cloudera@quickstart workspace]$ hadoop fs -cat WCOutput/part-00000
BDA     1
Hello   2
I'm     1
a       1
are     1
doing   1
pracs   1
student 1
we      1
[cloudera@quickstart workspace]$
```