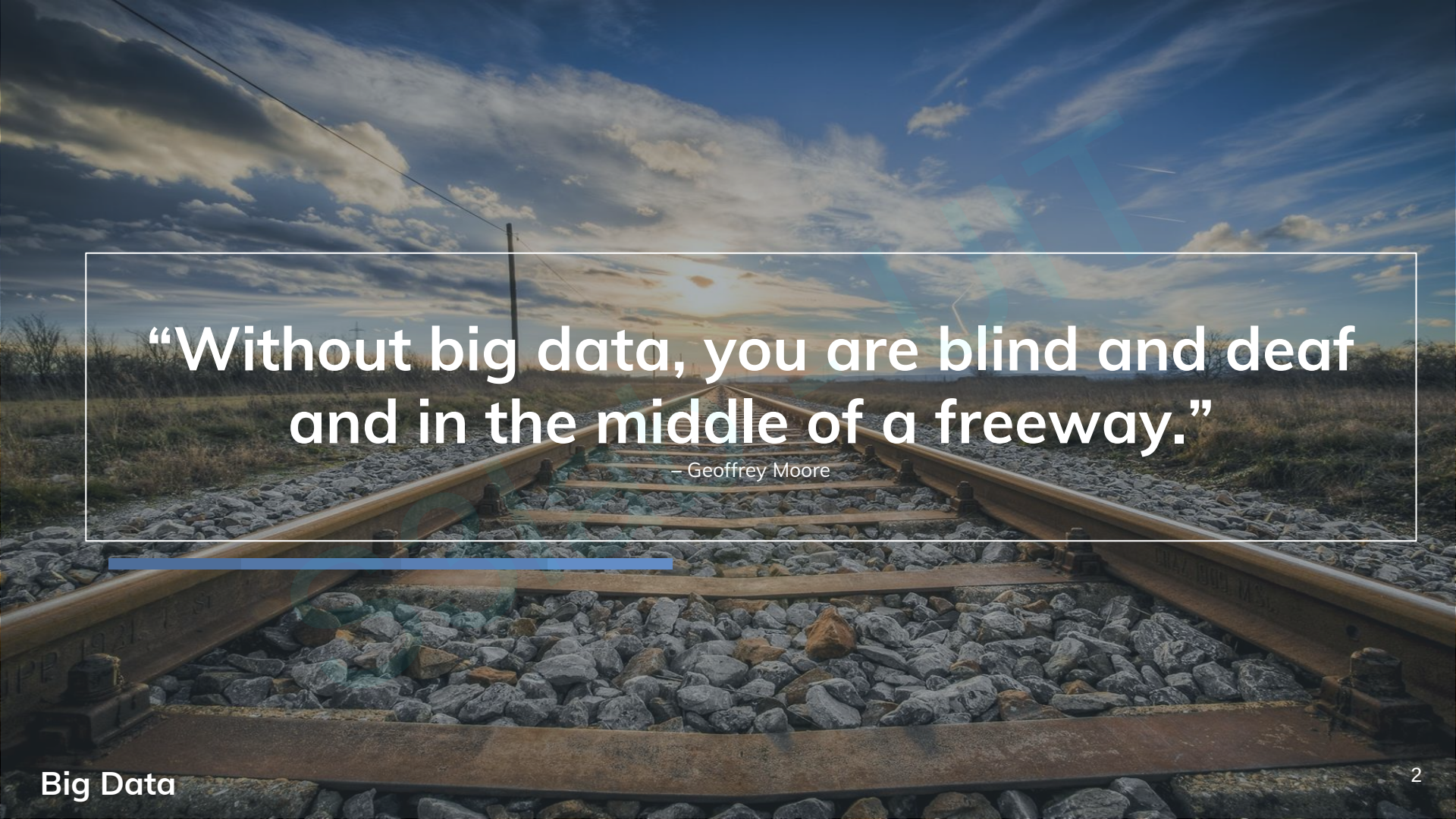
Big Data

Hadoop tutorial

Trong-Hop Do

S3Lab
Smart Software System Laboratory

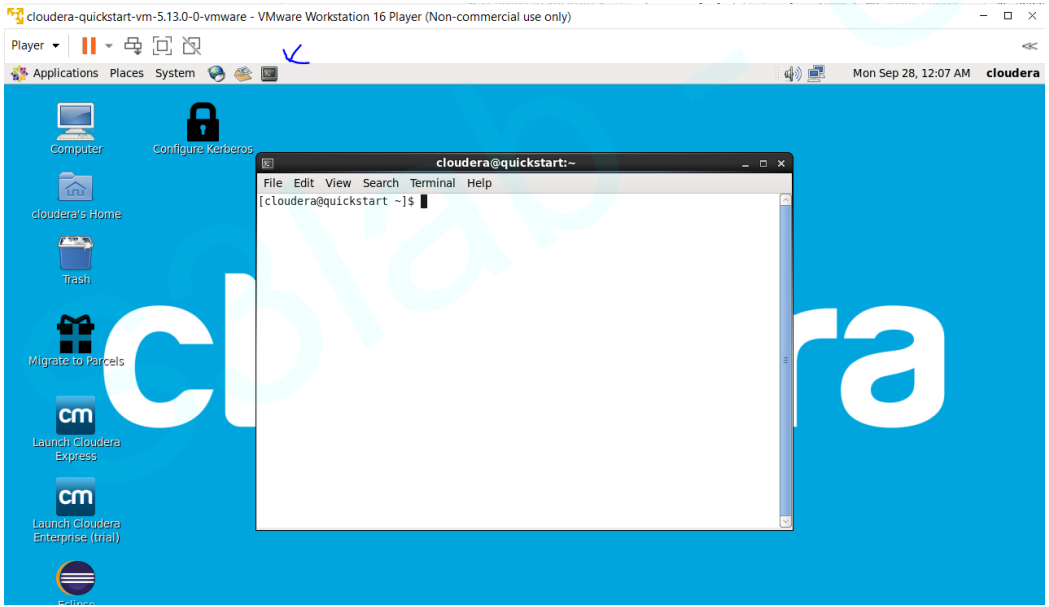

“Without big data, you are blind and deaf and in the middle of a freeway.”
– Geoffrey Moore

Hadoop Shell Commands

Hadoop Shell Commands to Manage HDFS

Open terminal

- In this exercise, we will practise generic HDFS commands and get ourselves familiar with Hadoop command line interface.
- Open Terminal on your VM, navigate to Application > System Tools > Terminal Or use the shortcut on desktop.



Hadoop Shell Commands to Manage HDFS

hdfs dfs

- View description of all the commands associated with FsShell subsystem
- Command: **hdfs dfs**

```
[cloudera@quickstart ~]$ hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] [-v] [-x] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] [-x] <path> ...]
        [-expunge]
        [-find <path> ... <expression> ...]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]
        [-mkdir [-p] <path> ...]
```

Hadoop Shell Commands to Manage HDFS

ls

- HDFS Command to display the list of Files and Directories in HDFS.
- Command: **hdfs dfs –ls**

```
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart ~]$
```

Hadoop Shell Commands to Manage HDFS

ls

- View the content of **root** directory.
- Command: **hdfs dfs –ls /**

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 7 items
drwxrwxrwx   - hdfs     supergroup          0 2017-10-23 10:29 /benchmarks
drwxr-xr-x   - cloudera supergroup          0 2020-09-26 05:36 /dataset
drwxr-xr-x   - hbase    supergroup          0 2020-09-27 23:42 /hbase
drwxr-xr-x   - solr     solr                0 2017-10-23 10:32 /solr
drwxrwxrwt   - hdfs     supergroup          0 2020-09-16 08:12 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2017-10-23 10:31 /user
drwxr-xr-x   - hdfs     supergroup          0 2017-10-23 10:31 /var
```

- Note: the directory structure in HDFS has nothing to do with the directory structure of the local filesystem, they are separate namespaces.

Hadoop Shell Commands to Manage HDFS

ls

- The **home** directory is **/user/cloudera/**
- Command: **hdfs dfs –ls** or hdfs **dfs –ls /user/cloudera**

```
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 4 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:07 inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:29 outputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart ~]$ hdfs dfs -ls /user/cloudera
Found 4 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 /user/cloudera/dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:07 /user/cloudera/inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:29 /user/cloudera/outputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 /user/cloudera/student
```

Hadoop Shell Commands to Manage HDFS

mkdir

- HDFS Command to create the directory in HDFS.
- Usage: hdfs dfs –mkdir directory_name
- Command: **hdfs dfs –mkdir dataset**

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir dataset
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 2 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 04:21 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart ~]$
```

Hadoop Shell Commands to Manage HDFS

touchz

- HDFS Command to create a file in HDFS with file size 0 bytes.
- Usage: hdfs dfs –touchz directory/filename
- Command: **hdfs dfs –touchz dataset/sample**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -touchz dataset/sample
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls dataset
Found 1 items
-rw-r--r--   1 cloudera cloudera          0 2020-09-29 03:13 dataset/sample
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

du

- HDFS Command to check the file size.
- Usage: hdfs dfs –du –s directory/filename
- Command: **hdfs dfs –du –s dataset/sample**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -du -s dataset/sample
0  0  dataset/sample
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

put

- HDFS Command to copy single source or multiple sources from local file system to the destination file system.
- Usage: hdfs dfs -put <localsrc> <destination>
- Command: **hdfs dfs –put data.txt dataset**

```
[cloudera@quickstart HadoopStreaming]$ echo 'This is a test file' > data.txt
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -put data.txt dataset
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls dataset
Found 2 items
-rw-r--r--   1 cloudera cloudera         20 2020-09-29 03:17 dataset/data.txt
-rw-r--r--   1 cloudera cloudera          0 2020-09-29 03:13 dataset/sample
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

cat

- HDFS Command that reads a file on HDFS and prints the content of that file to the standard output.

- Usage: hdfs dfs –cat /path/to/file_in_hdfs

- Command: **hdfs dfs –cat dataset/data.txt**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -cat dataset/data.txt
This is a test file
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

get

- HDFS Command to copy files from hdfs to the local file system.
- Usage: hdfs dfs -get <src> <localdst>
- Command: **hdfs dfs –get dataset/sample /home/cloudera**

```
[cloudera@quickstart ~]$ hdfs dfs -get /dataset/sample /home/cloudera
[cloudera@quickstart ~]$ ls /home/cloudera/
cloudera-manager        Documents                   lib       student.java
cm_api.py               Downloads                   Music     Templates
codegen_categories.java eclipse                     parcels   Videos
course.java             enterprise-deployment.json  Pictures  workspace
data.txt                express-deployment.json     Public
Desktop                 kerberos                    sample
[cloudera@quickstart ~]$
```

Hadoop Shell Commands to Manage HDFS

count

- HDFS Command to count the number of directories, files, and bytes under the paths that match the specified file pattern.

- Usage: hdfs dfs -count <path>

- Command: **hdfs dfs –count dataset**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -count dataset
           1            2                 20 dataset
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

rm

- HDFS Command to remove the file from HDFS.
- Usage: hdfs dfs –rm <path>
- Command: **hdfs dfs –rm dataset/sample**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -rm dataset/sample
Deleted dataset/sample
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls dataset
Found 1 items
-rw-r--r--   1 cloudera cloudera         20 2020-09-29 03:17 dataset/data.txt
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

rm -r

- HDFS Command to remove the entire directory and all of its content from HDFS.
- Usage: hdfs dfs -rm -r <path>
- Command: **hdfs dfs -rm -r dataset**

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls
Found 7 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-29 00:30 HSOutput
drwxr-xr-x   - cloudera cloudera          0 2020-09-28 23:13 HadoopStreaming
drwxr-xr-x   - cloudera cloudera          0 2020-09-28 05:17 ReduceJoin
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:07 inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:29 outputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -rm -r dataset
Deleted dataset
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls dataset
ls: `dataset': No such file or directory
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Shell Commands to Manage HDFS

cp

- HDFS Command to copy files from source to destination. This command allows multiple sources as well, in which case the destination must be a directory.
- Usage: hdfs dfs -cp <src> <dest>
- Command: **hdfs dfs -cp /user/cloudera/dataset/data.txt /user/cloudera/dataset/datacopy.txt**

```
[cloudera@quickstart ~]$ hdfs dfs -cp /user/cloudera/dataset/data.txt /user/clou
dera/dataset/datacopy.txt
[cloudera@quickstart ~]$ hdfs dfs -ls /user/cloudera/dataset
Found 2 items
-rw-r--r--   1 cloudera cloudera         20 2020-09-26 05:00 /user/cloudera/data
set/data.txt
-rw-r--r--   1 cloudera cloudera         20 2020-09-26 06:02 /user/cloudera/data
set/datacopy.txt
[cloudera@quickstart ~]$
```

Hadoop Shell Commands to Manage HDFS

rmdir

- HDFS Command to remove a empty directory.
- Usage: hdfs dfs -rmdir <path>
- Command: **hdfs dfs –rmdir /user/cloudera/dataset**

Hadoop Shell Commands to Manage HDFS

usage

- HDFS Command that returns the help for an individual command.
- Usage: hdfs dfs -usage <command>
- Command: **hdfs dfs -usage mkdir**
- By using usage command you can get information about any command

Hadoop Shell Commands to Manage HDFS

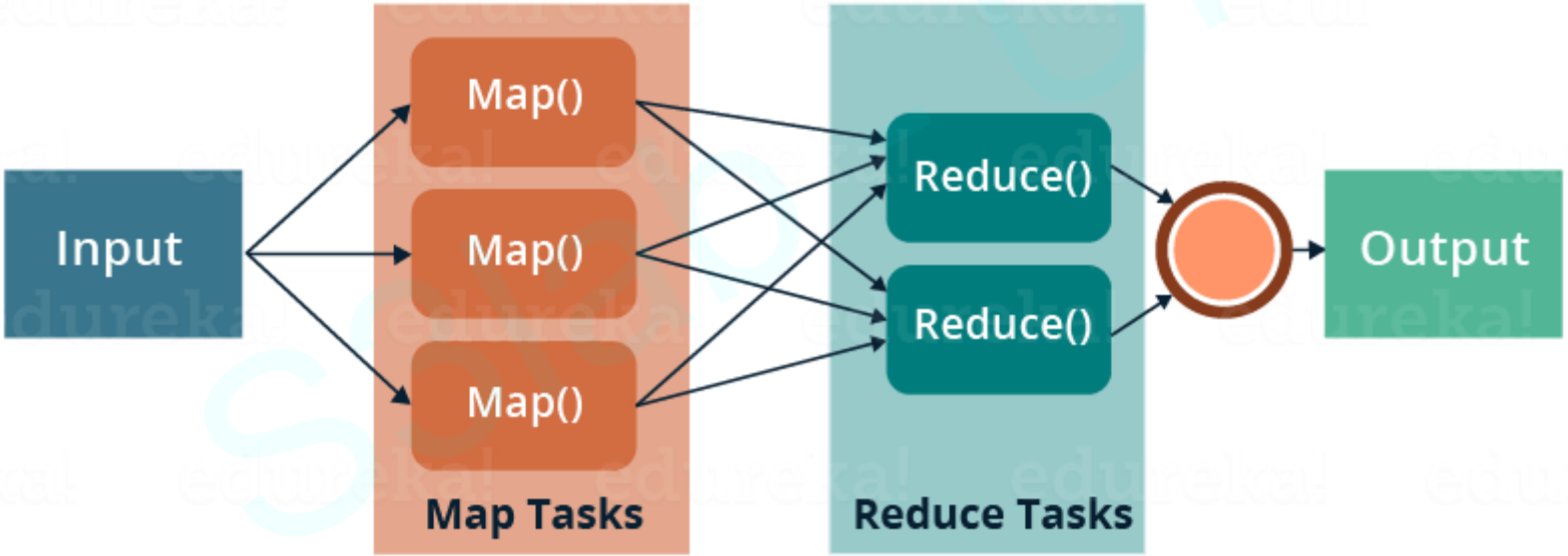
help

- HDFS Command that displays help for given command or all commands if none is specified.

- Command: **hdfs dfs -help**

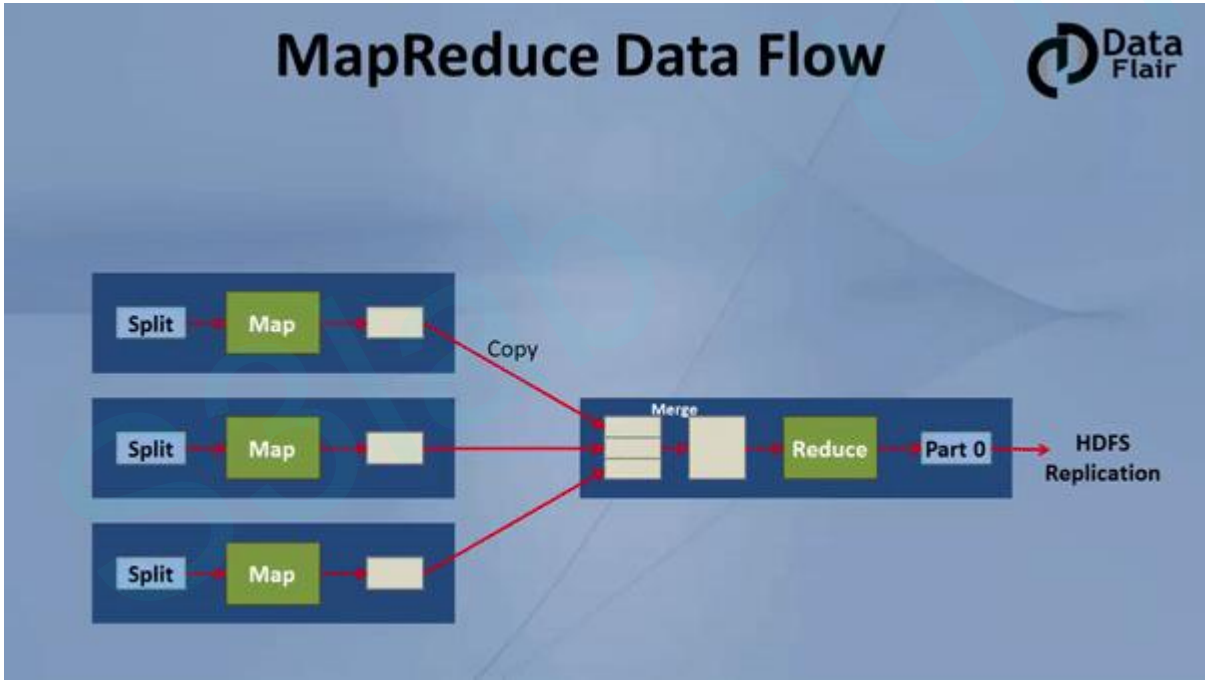
Word Count MapReduce

Hadoop MapReduce



Hadoop MapReduce

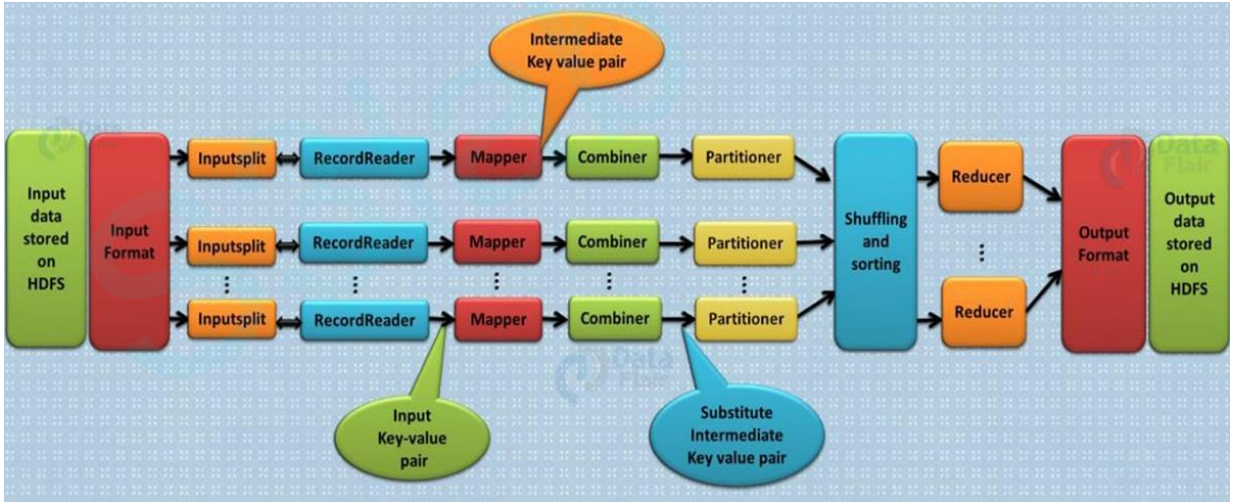
How Hadoop MapReduce work?



Hadoop MapReduce

Input Files

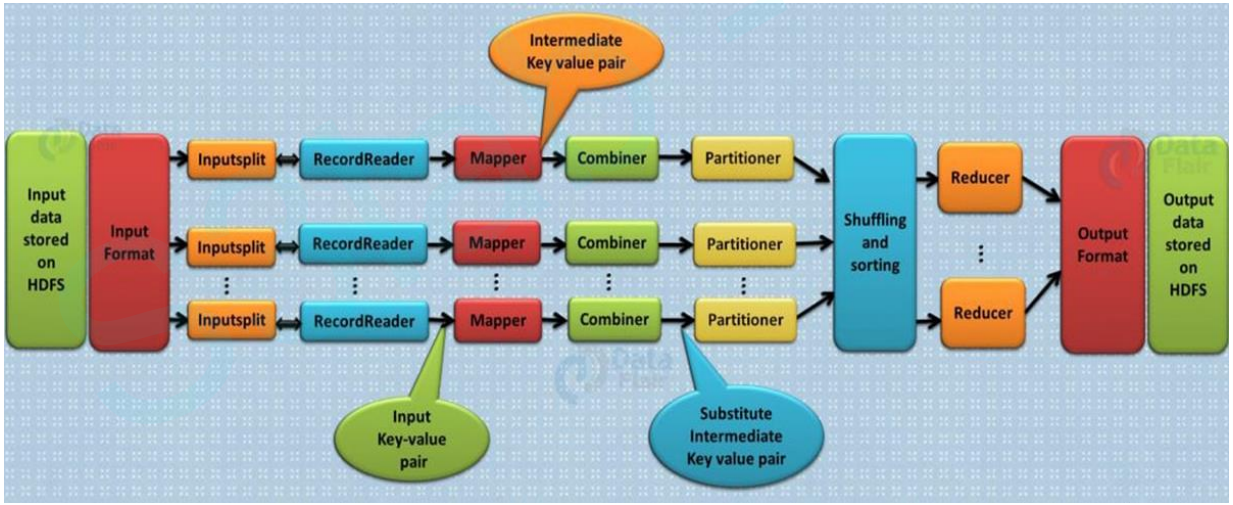
- The data for a MapReduce task is stored in input files, and input files typically lives in HDFS. The format of these files is arbitrary, while line-based log files and binary format can also be used.



Hadoop MapReduce

InputFormat

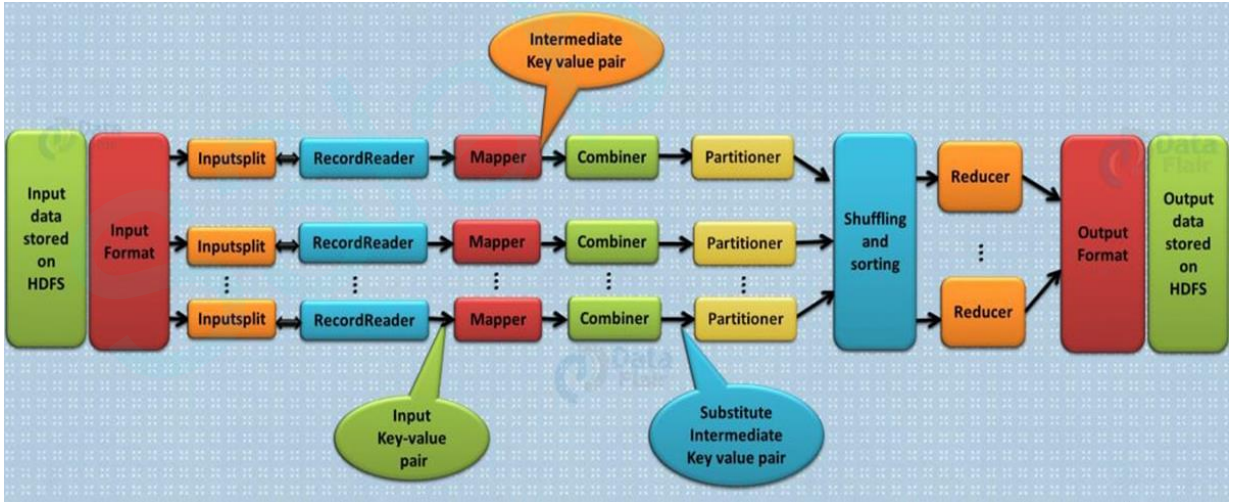
- Now, InputFormat defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.



Hadoop MapReduce

InputSplits

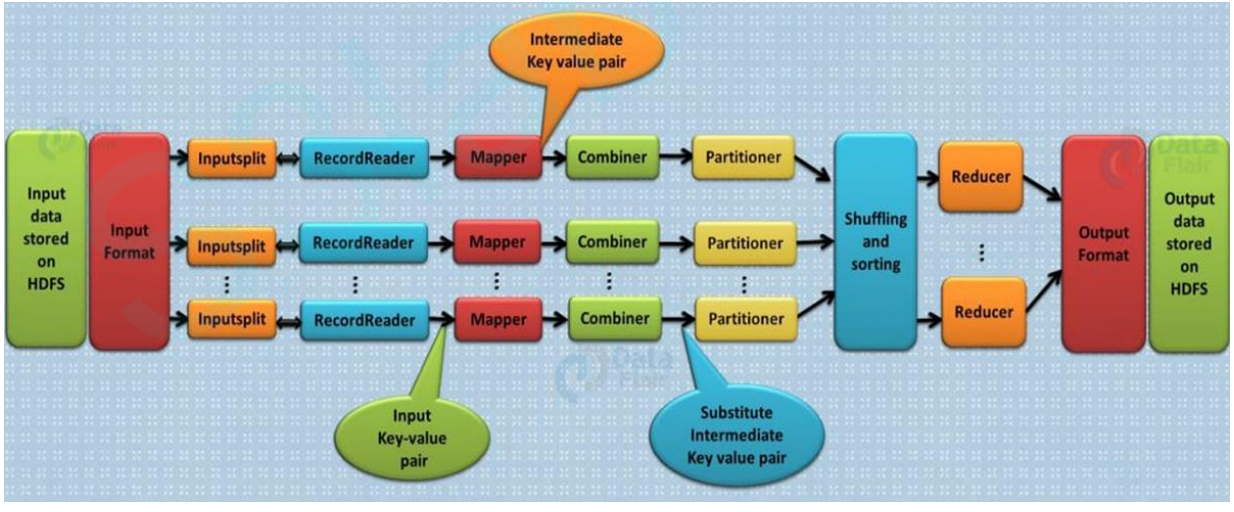
- It is created by InputFormat, logically represent the data which will be processed by an individual Mapper (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper.



Hadoop MapReduce

RecordReader

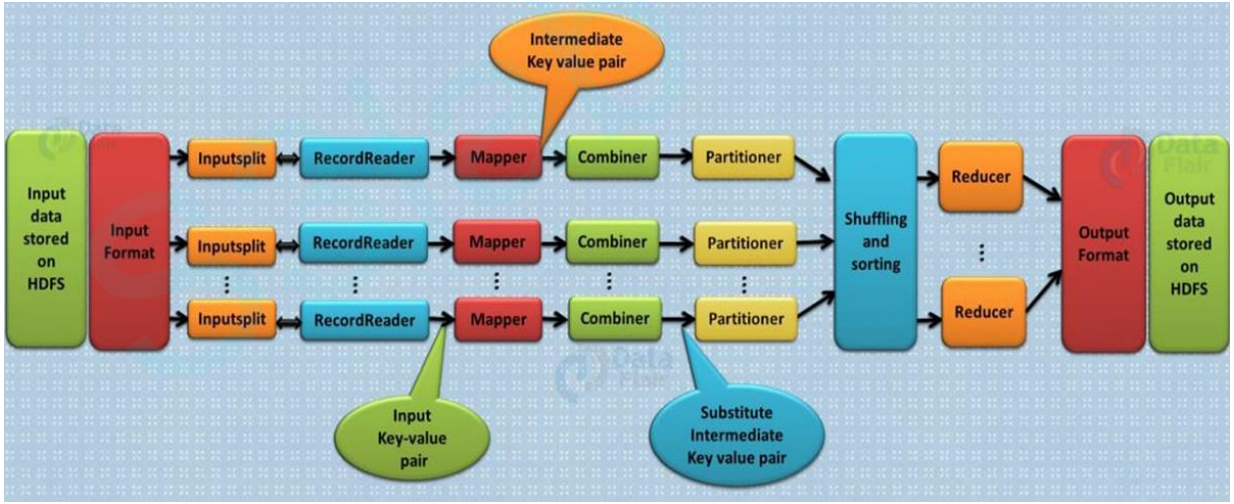
- It communicates with the **InputSplit** in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.



Hadoop MapReduce

Mapper

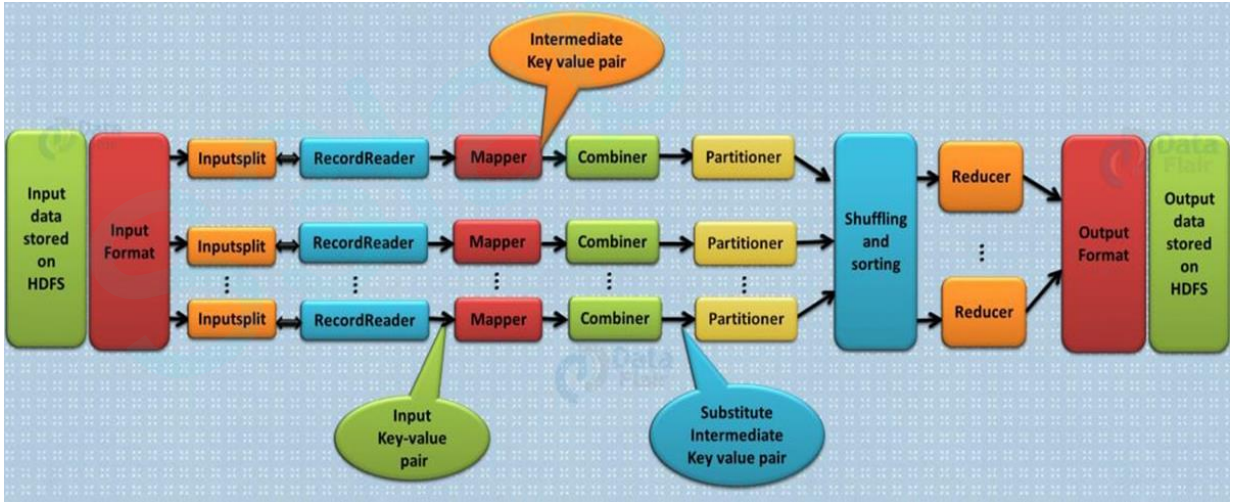
- It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process



Hadoop MapReduce

Combiner

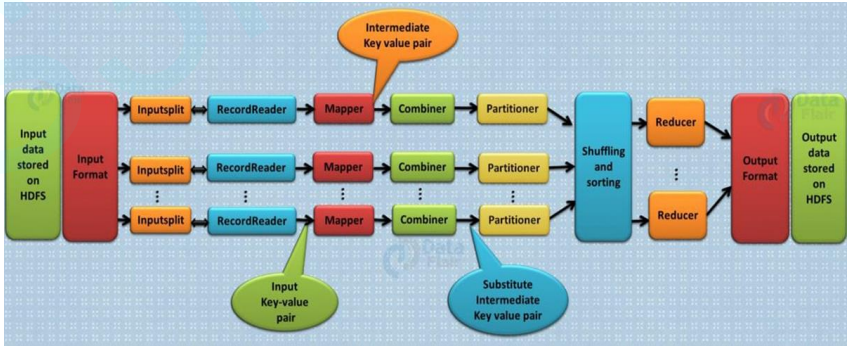
- The combiner is also known as ‘Mini-reducer’. Hadoop MapReduce Combiner performs local aggregation on the mappers’ output, which helps to minimize the data transfer between mapper and reducer (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work.



Hadoop MapReduce

Partitioner

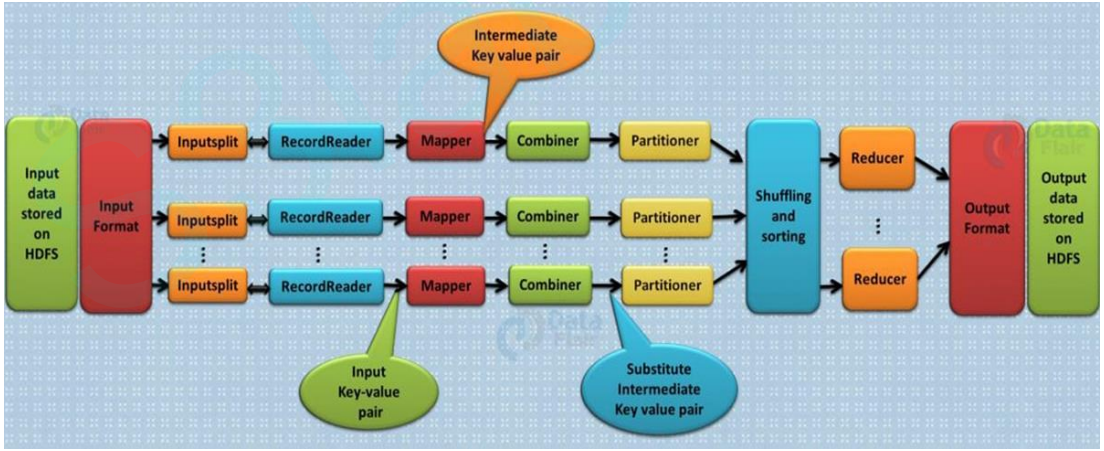
- Hadoop MapReduce, Partitioner comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).
- Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition.
- According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.



Hadoop MapReduce

Shuffling and Sorting

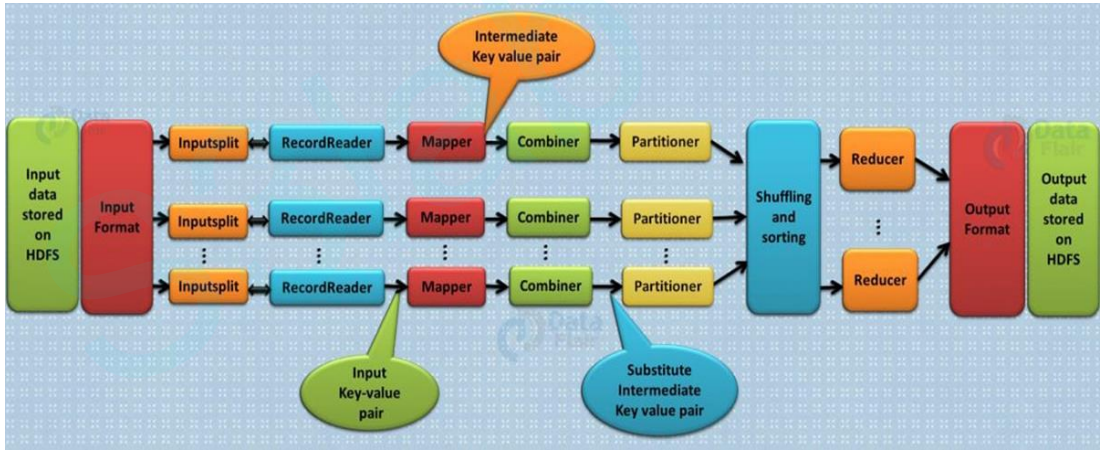
- Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.



Hadoop MapReduce

Reducer

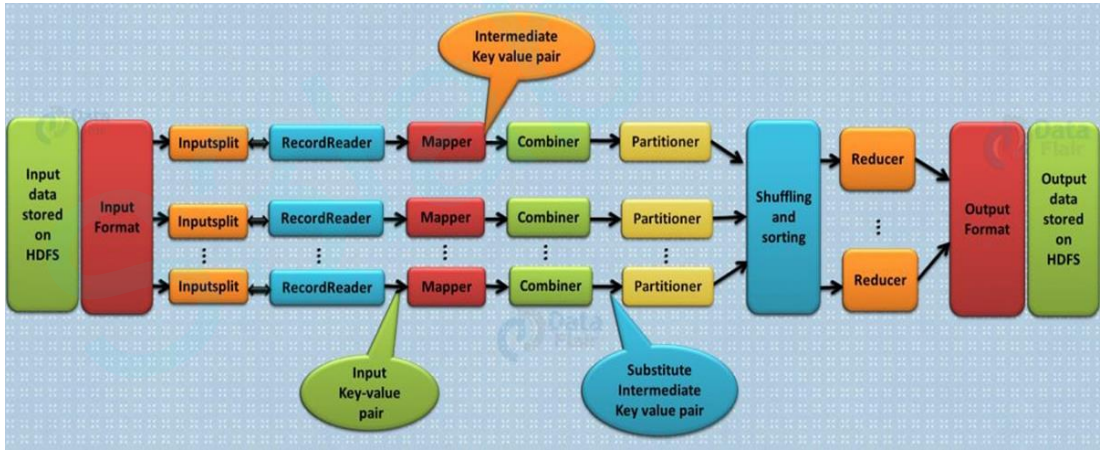
- It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.



Hadoop MapReduce

RecordWriter

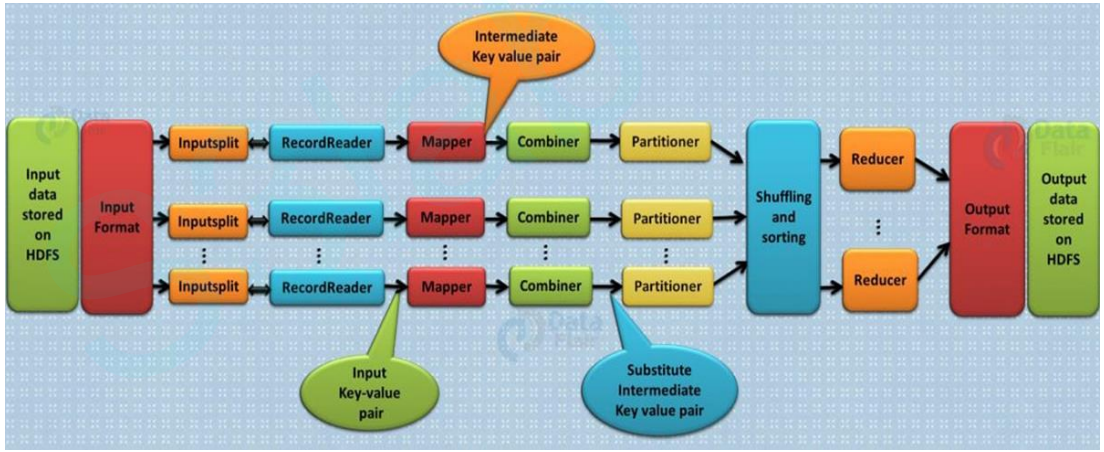
- It writes these output key-value pair from the Reducer phase to the output files.



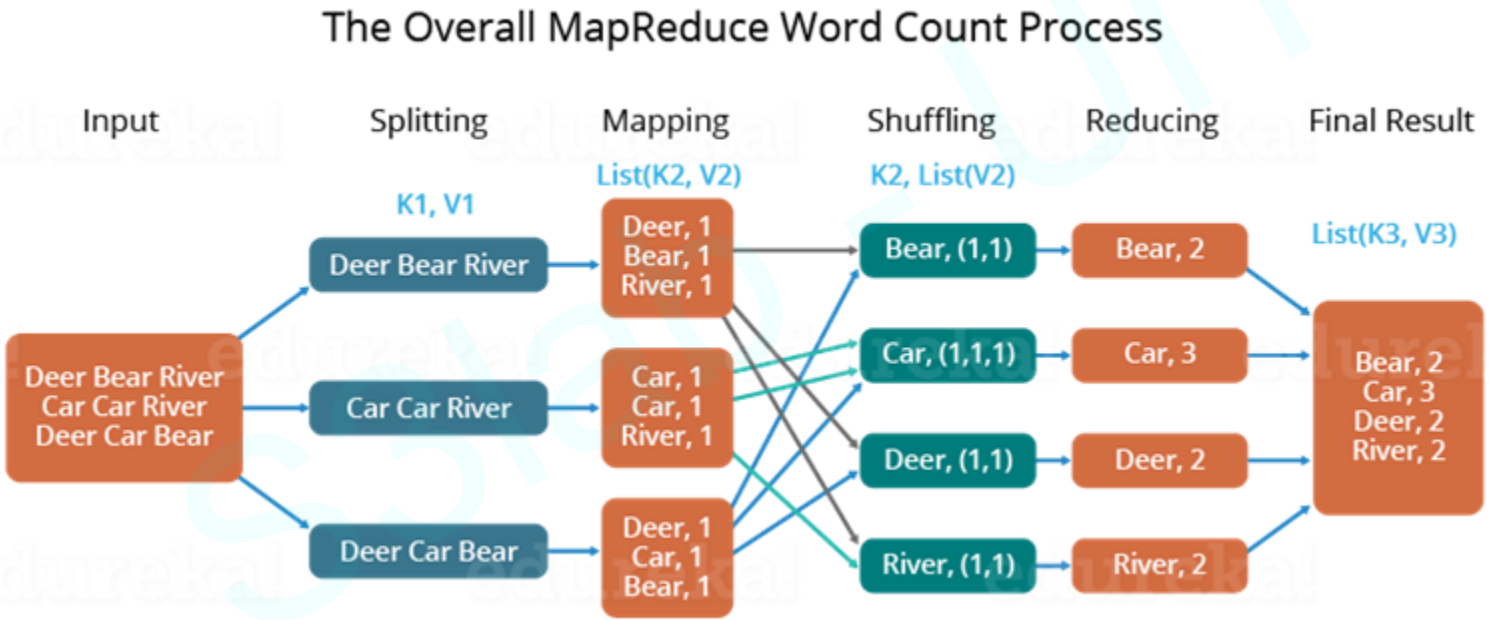
Hadoop MapReduce

OutputFormat

- The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer is written on HDFS by OutputFormat instances.
- Hence, in this manner, a Hadoop MapReduce works over the cluster.



Hadoop MapReduce



Hadoop MapReduce

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

Hadoop MapReduce

Mapper code

```
public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(Object key, Text value, Context context
                   ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
     }
   }
```

Hadoop MapReduce

Mapper code

```
public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }
 }
```

Hadoop MapReduce

Driver code

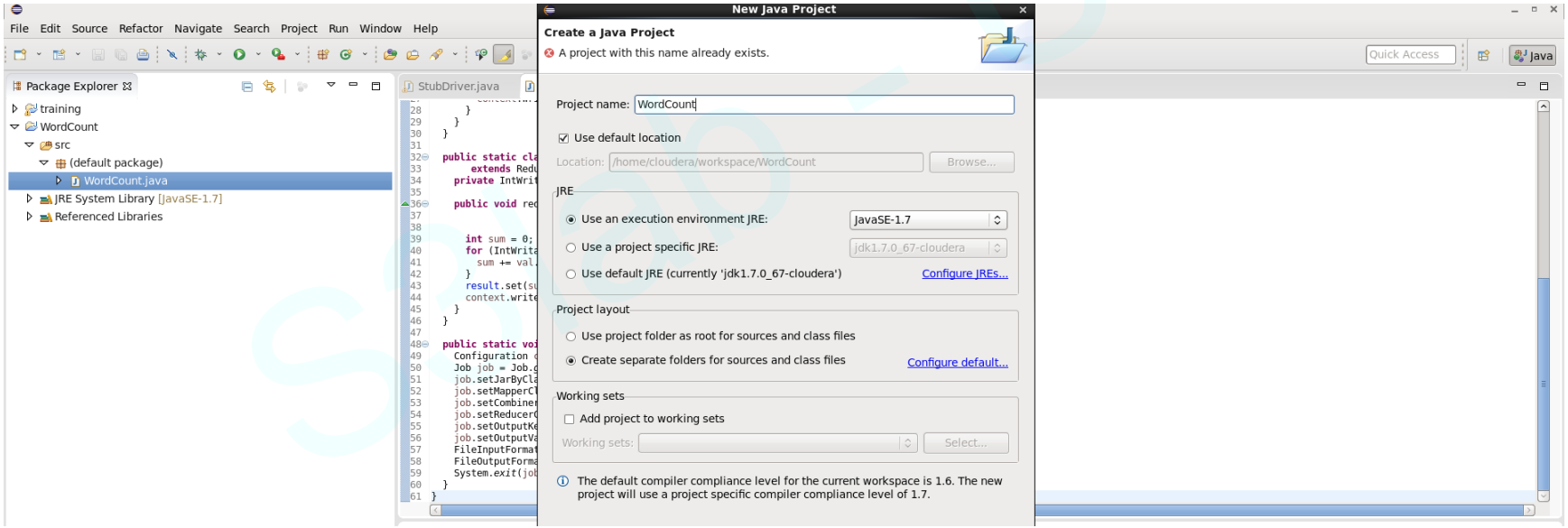
```
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

Word Count MapReduce

Word Count MapReduce Tutorial starts from here

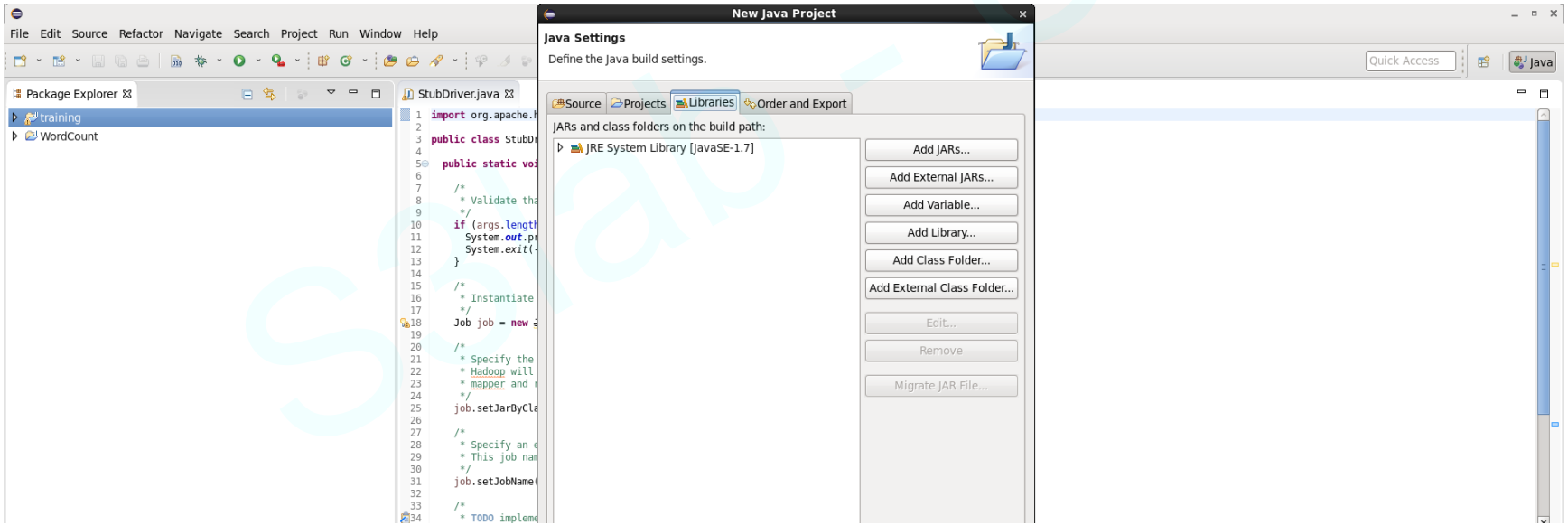
Word Count MapReduce

- Open Eclipse on Cloudera Quickstart VM and create a new Java Project



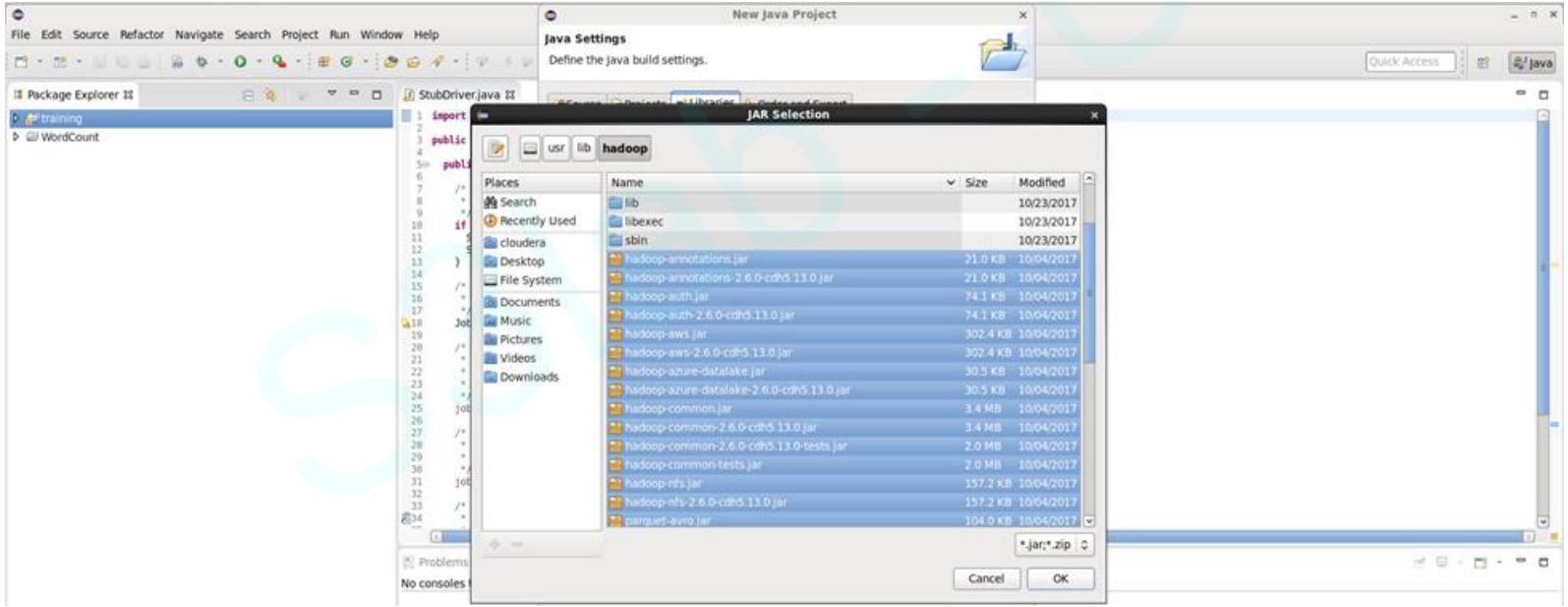
Word Count MapReduce

- Add Hadoop Libraries to project through **Add External JARS**



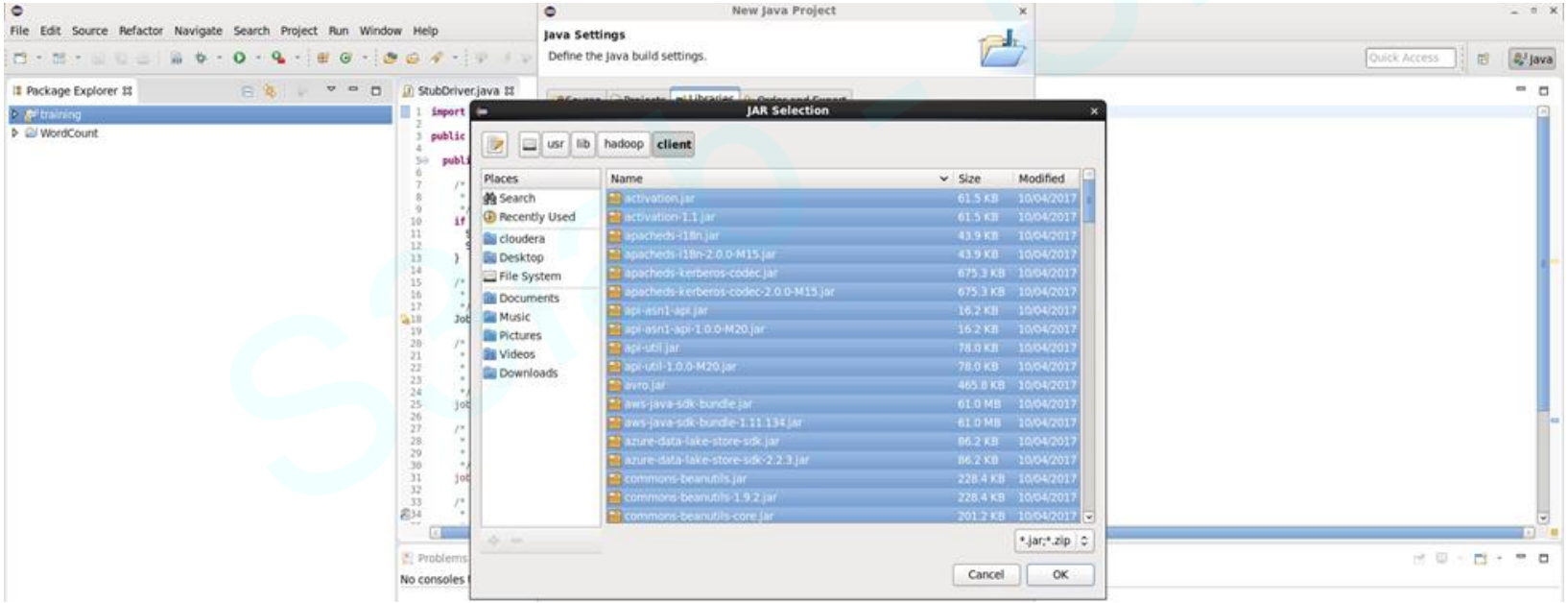
Word Count MapReduce

- Add all .jar files in the folder **usr/lib/hadoop**



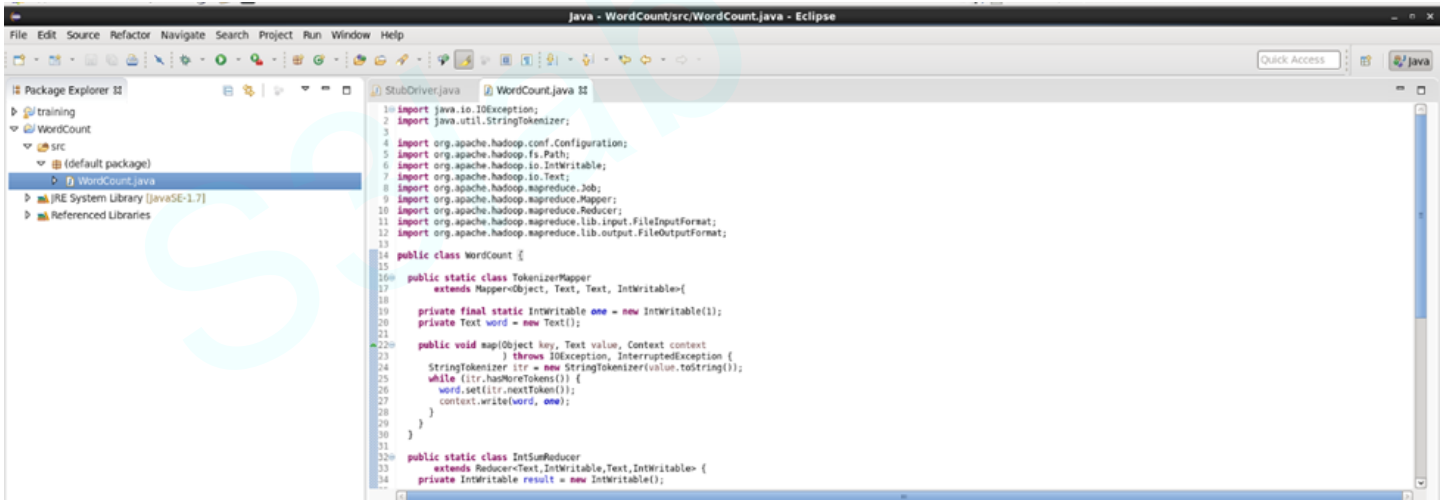
Word Count MapReduce

- Add all .jar files in the folder **usr/lib/hadoop/client**



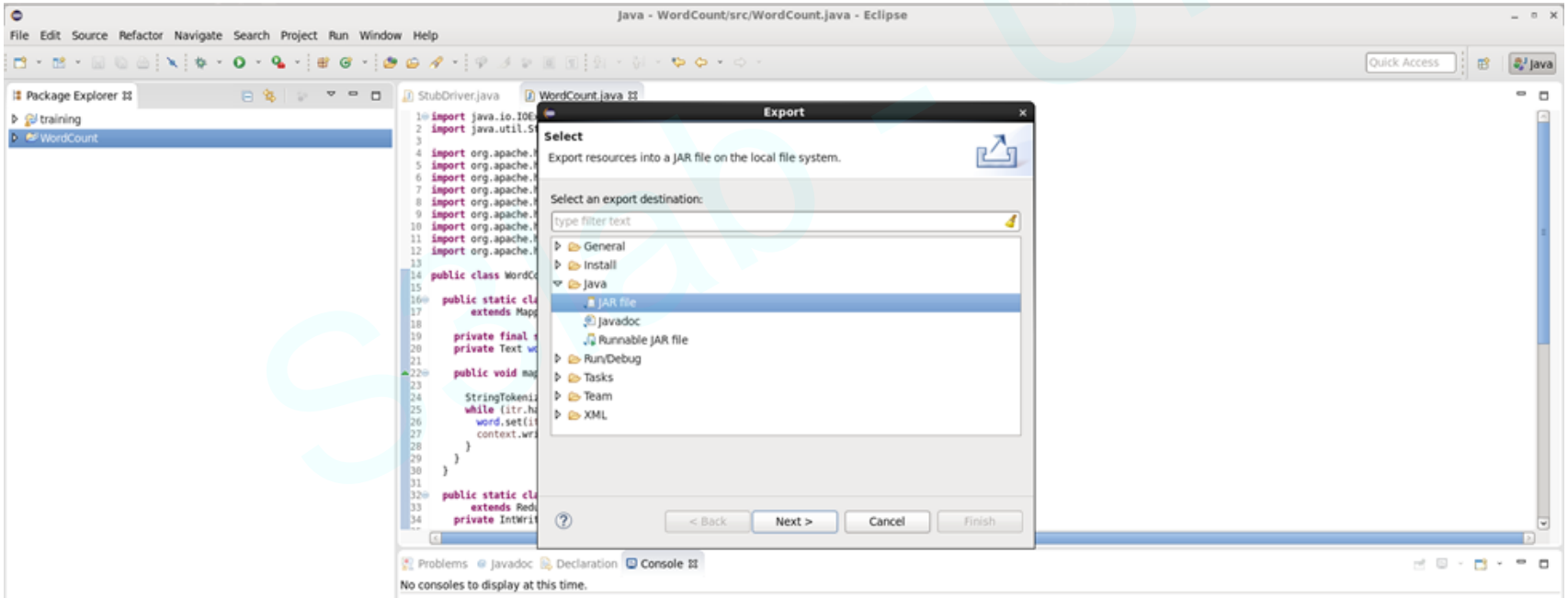
Word Count MapReduce

- Copy the source code of WordCount programe from https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html



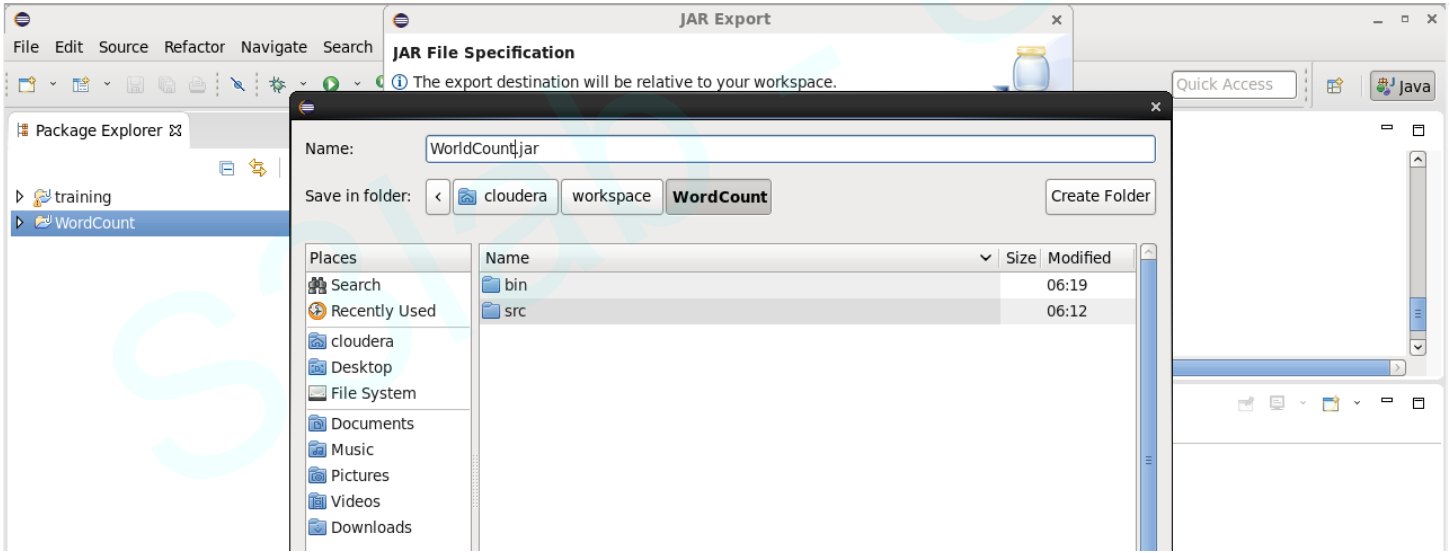
Word Count MapReduce

- Export to jar file



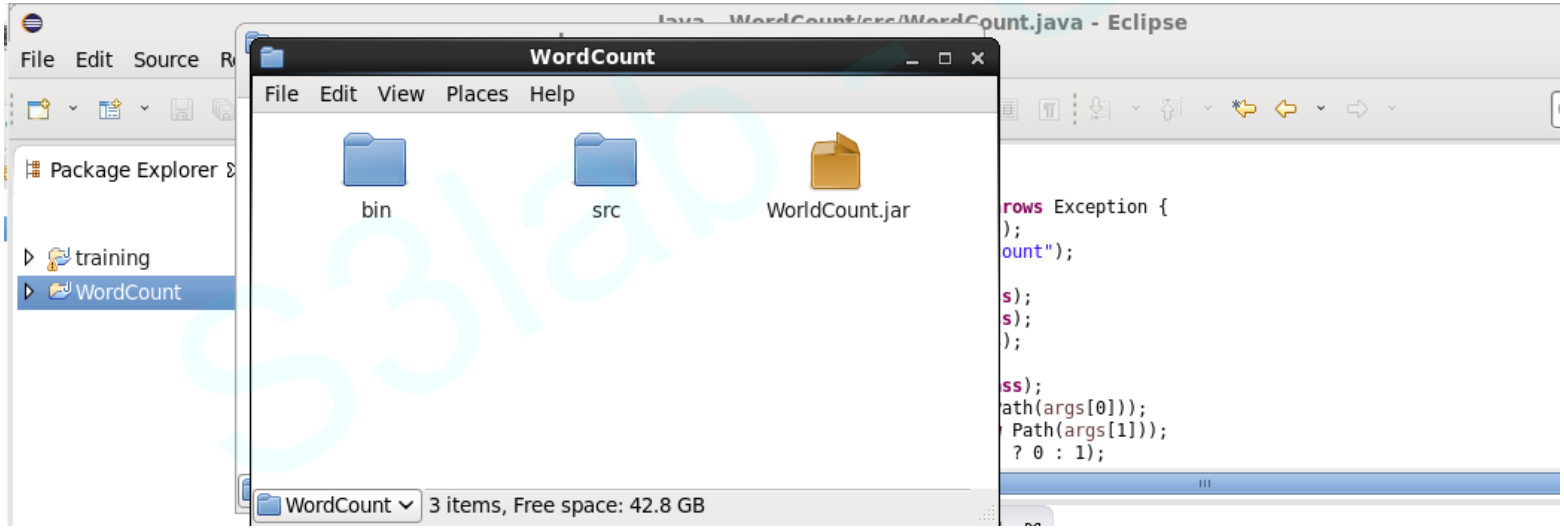
Hadoop Shell Commands to Manage HDFS

- Name the file **WordCount.jar** and place it inside **/home/cloudera/workspace/WordCount**



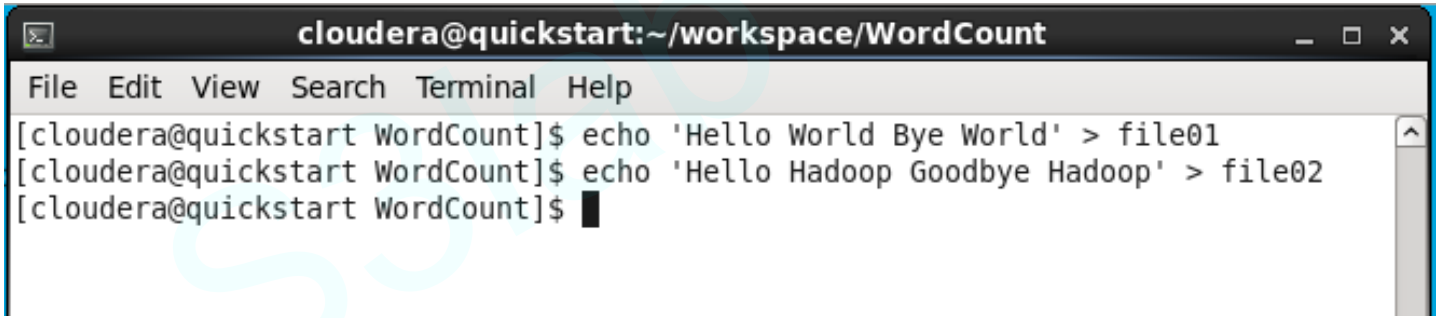
Hadoop Shell Commands to Manage HDFS

- Check if the **WordCount.jar** file has been created



Hadoop Shell Commands to Manage HDFS

- Create two text files inside **/home/cloudera/workspace/WordCount** folder



Hadoop Shell Commands to Manage HDFS

- Create input directory in Hadoop

```
[cloudera@quickstart WordCount]$ hdfs dfs -mkdir inputWC
[cloudera@quickstart WordCount]$ hdfs dfs -ls
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:00 inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart WordCount]$
```

Hadoop Shell Commands to Manage HDFS

- Move two text files to the input directory in Hadoop

```
[cloudera@quickstart WordCount]$ hdfs dfs -put file0* inputWC
[cloudera@quickstart WordCount]$ hdfs dfs -ls inputWC
Found 2 items
-rw-r--r--   1 cloudera cloudera         22 2020-09-27 07:07 inputWC/file01
-rw-r--r--   1 cloudera cloudera         28 2020-09-27 07:07 inputWC/file02
[cloudera@quickstart WordCount]$
```

Hadoop Shell Commands to Manage HDFS

- Check the content of the two files in Hadoop

```
[cloudera@quickstart WordCount]$ hdfs dfs -cat inputWC/file01
Hello World Bye World
[cloudera@quickstart WordCount]$ hdfs dfs -cat inputWC/file02
Hello Hadoop Goodbye Hadoop
[cloudera@quickstart WordCount]$
```

Hadoop Shell Commands to Manage HDFS

- Execute the JAR file on Hadoop

```
[cloudera@quickstart WordCount]$ hadoop jar WorldCount.jar WordCount inputWC outputWC
```

  - WordCount.jar jar: file to run
  - WordCount: name of the class
  - inputWC: input directory
  - outputWC: output directory

- This will create a directory **/user/cloudera/outputWC** and two files within it

Hadoop Shell Commands to Manage HDFS

- Check the output directory and the output files inside it

```
[cloudera@quickstart WordCount]$ hdfs dfs -ls
Found 4 items
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:07 inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:29 outputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart WordCount]$ hdfs dfs -ls outputWC
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2020-09-27 07:29 outputWC/_SUCCESS
-rw-r--r--   1 cloudera cloudera         41 2020-09-27 07:29 outputWC/part-r-00000
```

- Note: Job ran with only one Reducer, so there should be only one part-file

Hadoop Shell Commands to Manage HDFS

- Check the content of the output file

```
[cloudera@quickstart WordCount]$ hdfs dfs -cat outputWC/part*
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
[cloudera@quickstart WordCount]$
```

Hadoop Shell Commands to Manage HDFS

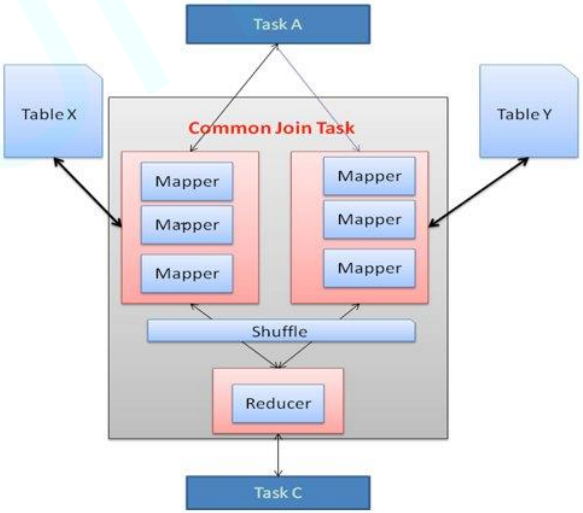
- If ssh localhost is not working and it shows, ssh not installed. Then please enter the following: > sudo apt-get install ssh Enter password if prompted.
- If there is an error while executing the command. Please check the variables JAVA_HOME and HADOOP_CLASSPATH. Later reset the values and proceed.
- If the is ClassNotFoundException, then please find the details in the link here: LINK. Or one could compile and save the .jar file within the same source directory.
- If an error like the following appears on trying to make directories in HDFS viz. then please run start-dfs.sh.

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -mkdir /user
mkdir: Call From hadoop/127.0.1.1 to localhost:9000 failed on connection exception: java.net.Con
nectException: Connection refused; For more details see:  http://wiki.apache.org/hadoop/Connecti
onRefused
```

Join in MapReduce

What is Join in MapReduce

- **Advantage** – Optimize Solution in terms of processing speed of the data.
- **Disadvantage** – Time Consuming for programmer, Non-ease mode of development due to 100's of lines of code and Availability of higher level frameworks like HIVE/PIG

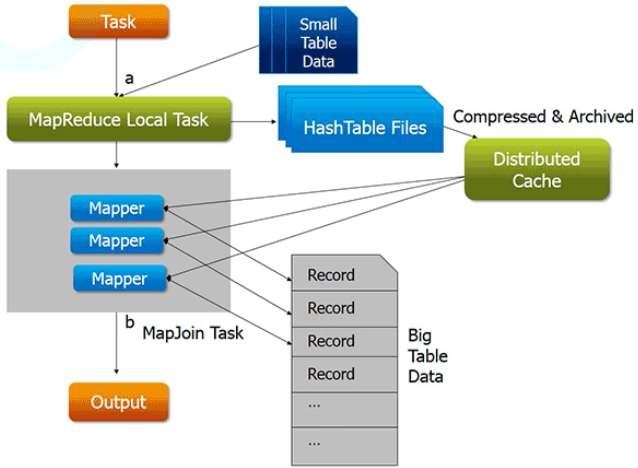


Type of Join in MapReduce

Map-Side Join

Read the data streams into the mappers and uses login within the mapper function to perform the join.

- **Where to use**: – When you have One large dataset and you need to join this with a small dataset. Also for Optimize performance.
- **Why**: – Smaller table will be loaded into memory and the join operation will happen during the mapper execution of large data set
- **Advantage**: Better performance.
- **Disadvantage**: Not Flexible i.e. can not be used if both data sets are large in size.
- **Note**: Reduce Side Join can also be used here but the performance will decrease.

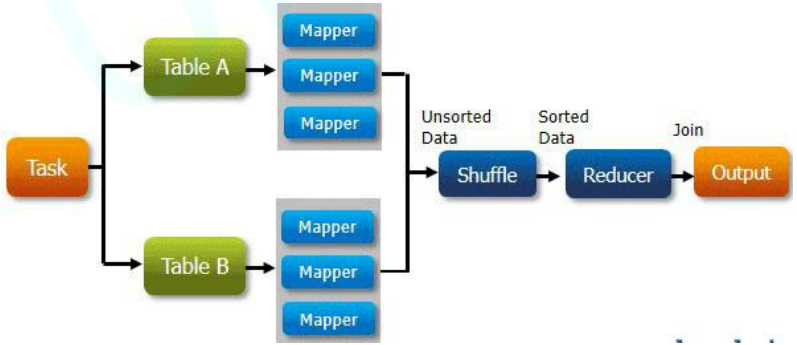


Type of Join in MapReduce

Reduce-Side Join

Process the multiple data streams through multiple map stages and perform the join at Reducer stage.

- **Where to use**: – When both the data sets are large.
  Why: – None of the dataset can be loaded in memory completely. Will have to process both tables separately and JOIN them at reducer side
- **Advantage**: Flexible and can be applied anywhere.
  Disadvantage: Poor performance in comparison with Map-side Joins
- **Note**: – Map Side Join can't be used here.



Reduce side Join

- Suppose that I have two separate datasets of a sports complex:
  - **cust_details:** It contains the details of the customer.
  - **transaction_details:** It contains the transaction record of the customer.
- Using these two datasets, I want to know the lifetime value of each customer. In doing so, I will be needing the following things:
  - The person’s name along with the frequency of the visits by that person.
  - The total amount spent by him/her for purchasing the equipment.

Reduce side Join

Cust ID	First Name	Last Name	Age	Profession
4000001	Kristina	Chung	55	Pilot
4000002	Paige	Chen	74	Teacher
4000003	Sherri	Melton	34	Firefighter
4000004	Gretchen	Hill	66	Engineer
.....	.....	.....	.....	.....

Fig: cust_details

Trans ID	Date	Cust ID	Amount	Game Type	Equipment	City	State	Mode
0000000	06-26-2011	4000001	40.33	Exercise & Fitness	Cardio Machine Accessories	Clarksville	Tennessee	credit
0000001	05-05-2011	4000002	198.44	Exercise & Fitness	Weightlifting Gloves	Long Beach	California	credit
0000002	06-17-2011	4000002	5.58	Exercise & Fitness	Weightlifting Machine Accessories	Anaheim	California	credit
0000003	06-14-2011	4000003	198.19	Gymnastics	Gymnastics Rings	Milwaukee	Wisconsin	credit
0000004	12-28-2011	4000002	98.81	Team Sports	Field Hockey	Nashville	Tennessee	credit
0000005	02-14-2011	4000004	193.63	Outdoor Recreation	Camping & Backpacking & Hiking	Chicago	Illinois	credit
0000006	10-17-2011	4000005	27.89	Puzzles	Jigsaw Puzzles	Charleston	South Carolina	credit
.....	.....	.....	.....	.....	.....	.....	.....	.....

Fig: transaction_details

Reduce side Join

Map phase: Mapper for customer

- Read the input taking one tuple at a time.
- Tokenize each word in that tuple and fetch the cust ID along with the name of the person.
- The cust ID will be the key of the key-value pair that the mapper will generate eventually.
- Add a tag “cust” to indicate that this input tuple is of cust_details type.
- Therefore, mapper for cust_details will produce following intermediate key-value pair:

Key – Value pair: [cust ID, cust name]

- Example: [4000001, cust Kristina], [4000002, cust Paige], etc.

Reduce side Join

Map phase: Mapper for transaction

- Fetch the amount value instead of name of the person.
- In this case, “tnxn” is used as a tag.
- Therefore, the cust ID will be the key of the key-value pair that the mapper will generate eventually.
- Finally, the output of mapper for transaction_details will be of the following format:

Key, Value Pair: [cust ID, tnxn amount]

- Example: [4000001, tnxn 40.33], [4000002, tnxn 198.44], etc.

Reduce side Join

Sorting and Shuffling Phase

- The sorting and shuffling phase will generate an array list of values corresponding to each key. In other words, it will put together all the values corresponding to each unique key in the intermediate key-value pair. The output of sorting and shuffling phase will be of the following format:

  Key – list of Values:

  {cust ID1 – [(cust name1), (tnxn amount1), (tnxn amount2), (tnxn amount3),…..]}

  {cust ID2 – [(cust name2), (tnxn amount1), (tnxn amount2), (tnxn amount3),…..]}

  ……

- Example:

  {4000001 – [(cust kristina), (tnxn 40.33), (tnxn 47.05),…]};

  {4000002 – [(cust paige), (tnxn 198.44), (tnxn 5.58),…]};

  ……

Reduce side Join

Reduce Phase

- The primary goal to perform this reduce-side join operation was to find out that how many times a particular customer has visited sports complex and the total amount spent by that very customer on different sports. Therefore, the final output should be of the following format:

Key – Value pair: [Name of the customer] (Key) – [total amount, frequency of the visit] (Value)

- Hence, the final output that my reducer will generate is given below:

Kristina, 651.05 8

Paige, 706.97 6

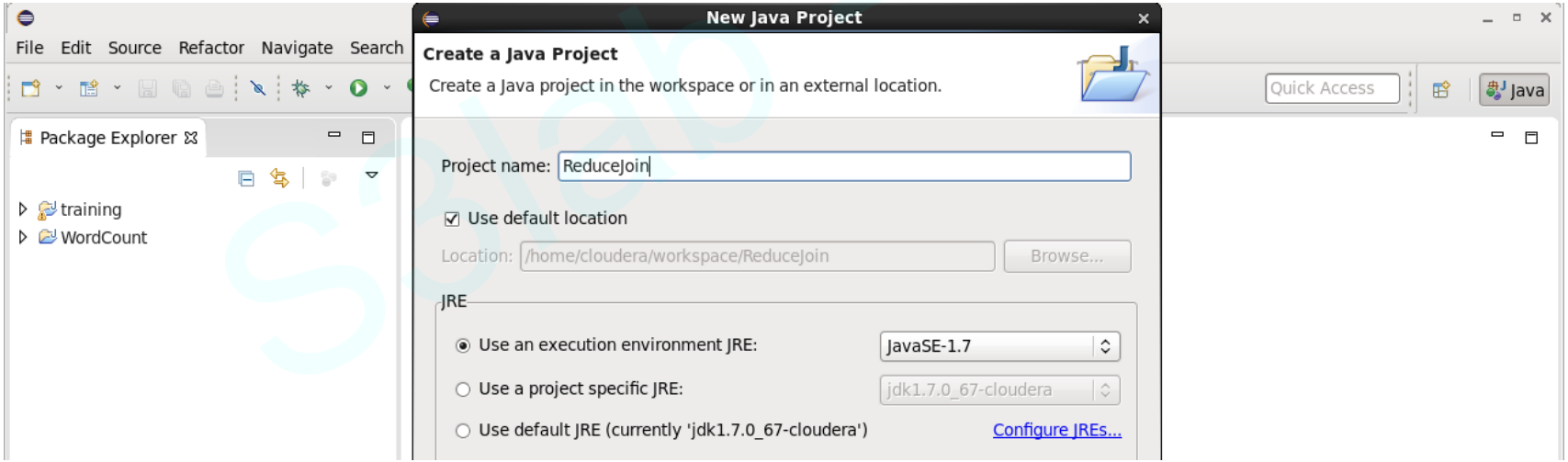
…..

Reduce side Join

Reduce side Join tutorial starts from here

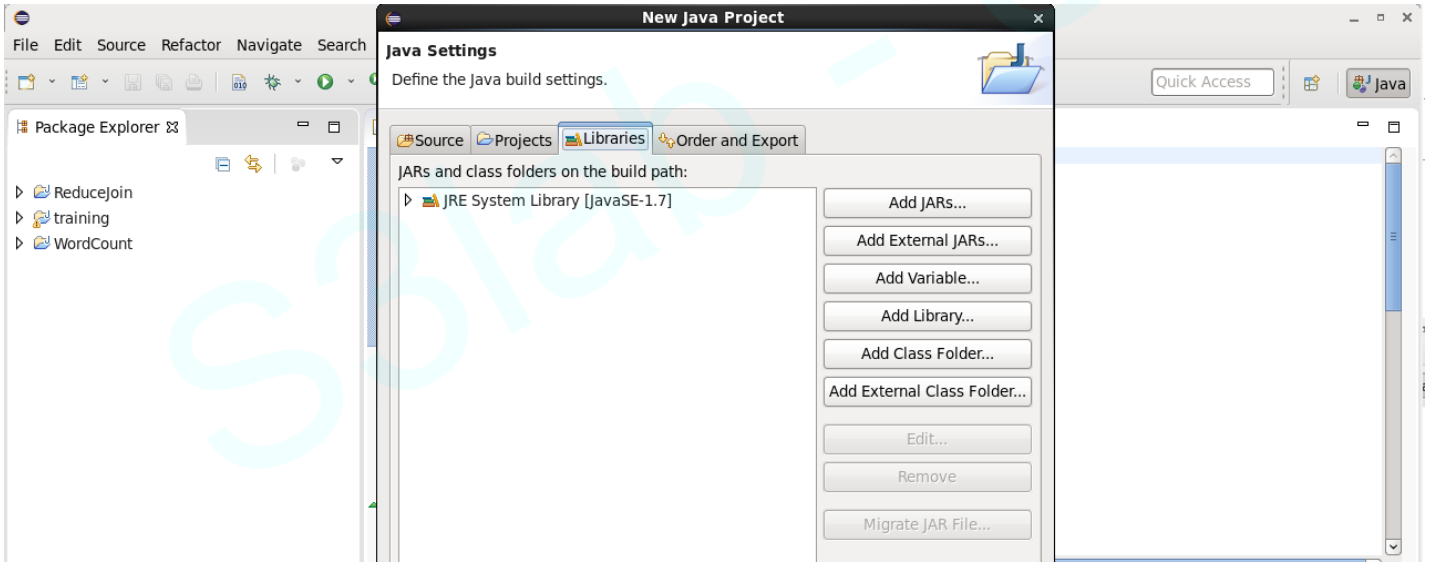
Reduce side Join

- Create Java Project in Eclipse then click Next.



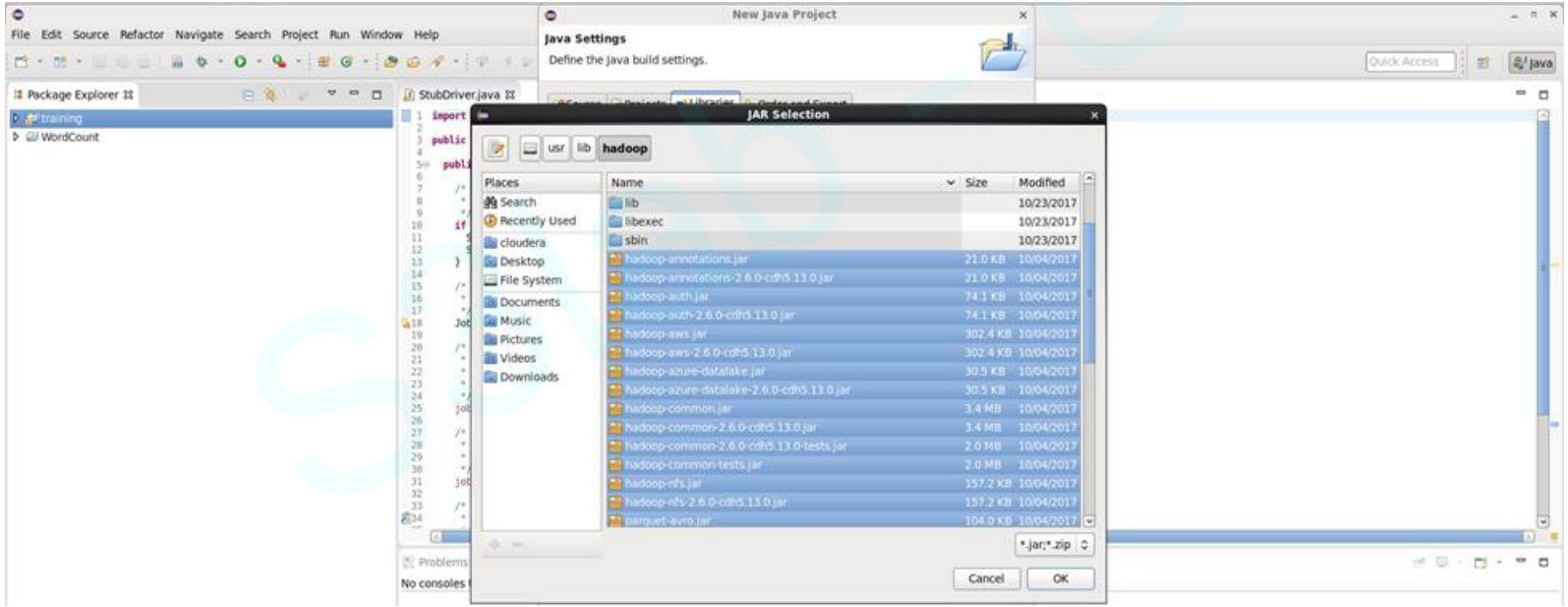
Reduce side Join

- Add External JARs



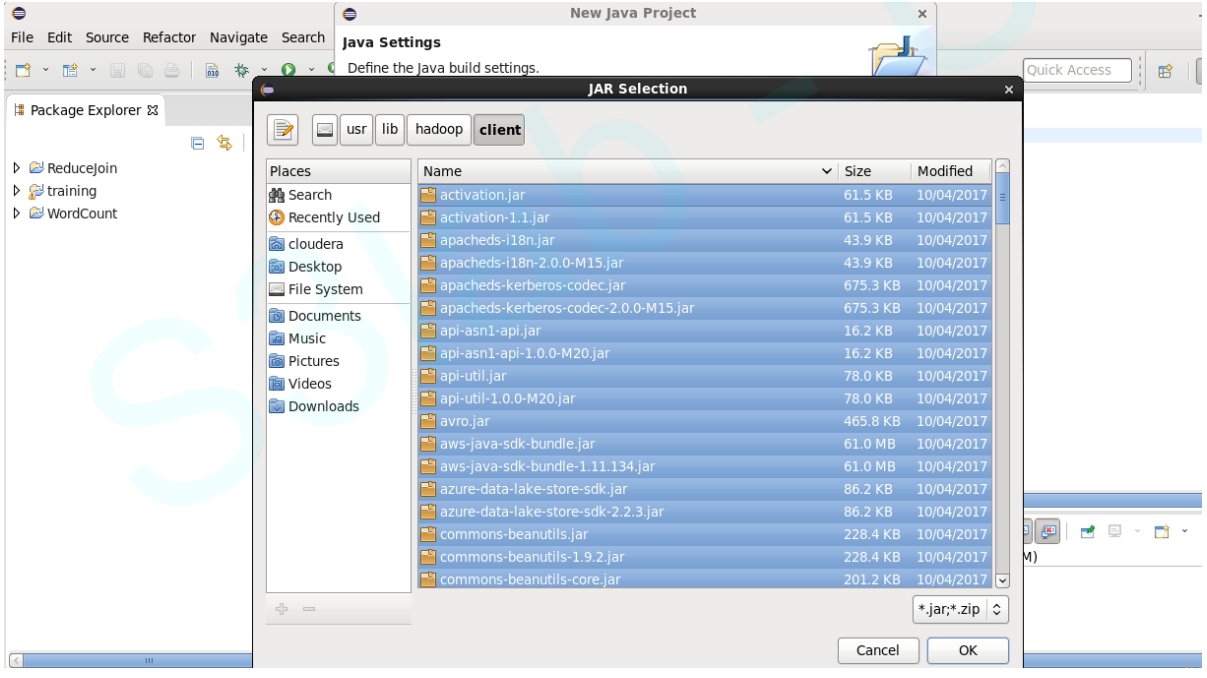
Reduce side Join

- Add all .jar files in the folder **usr/lib/hadoop**



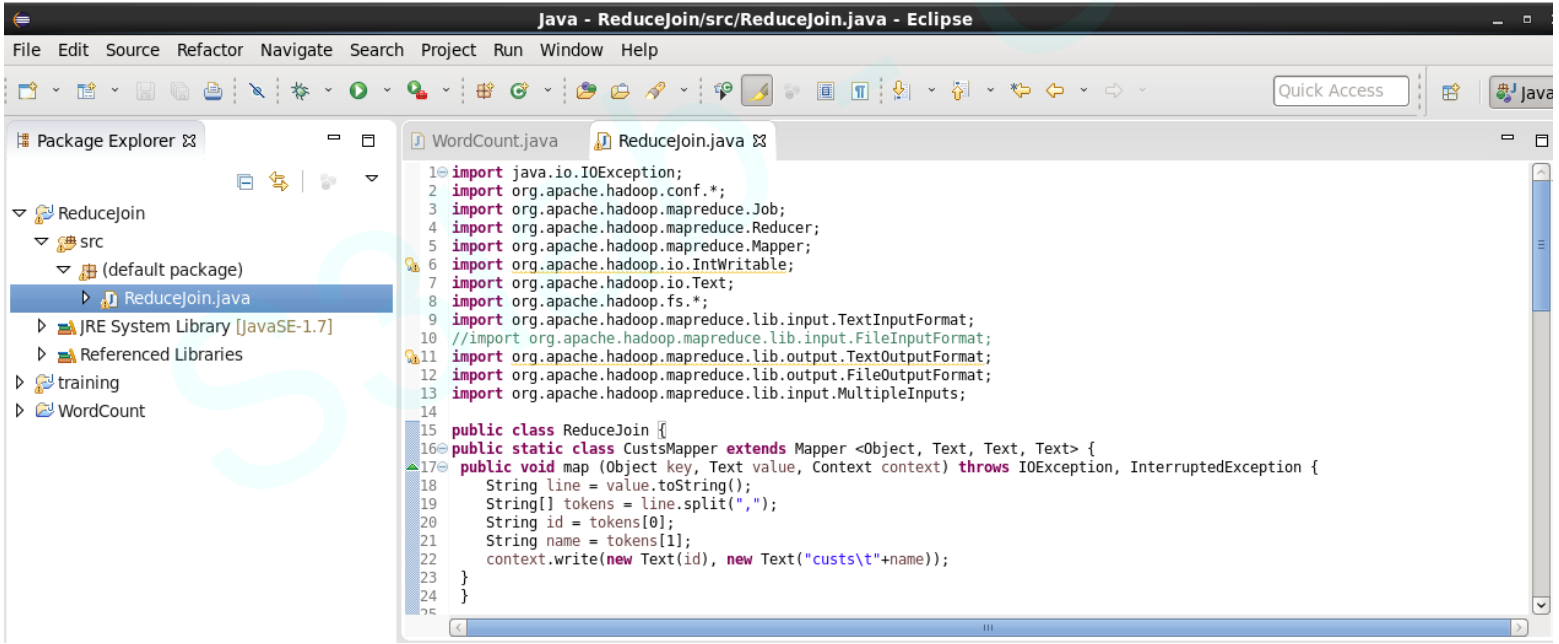
Reduce side Join

- Add all .jar files in the folder **usr/lib/hadoop/client**



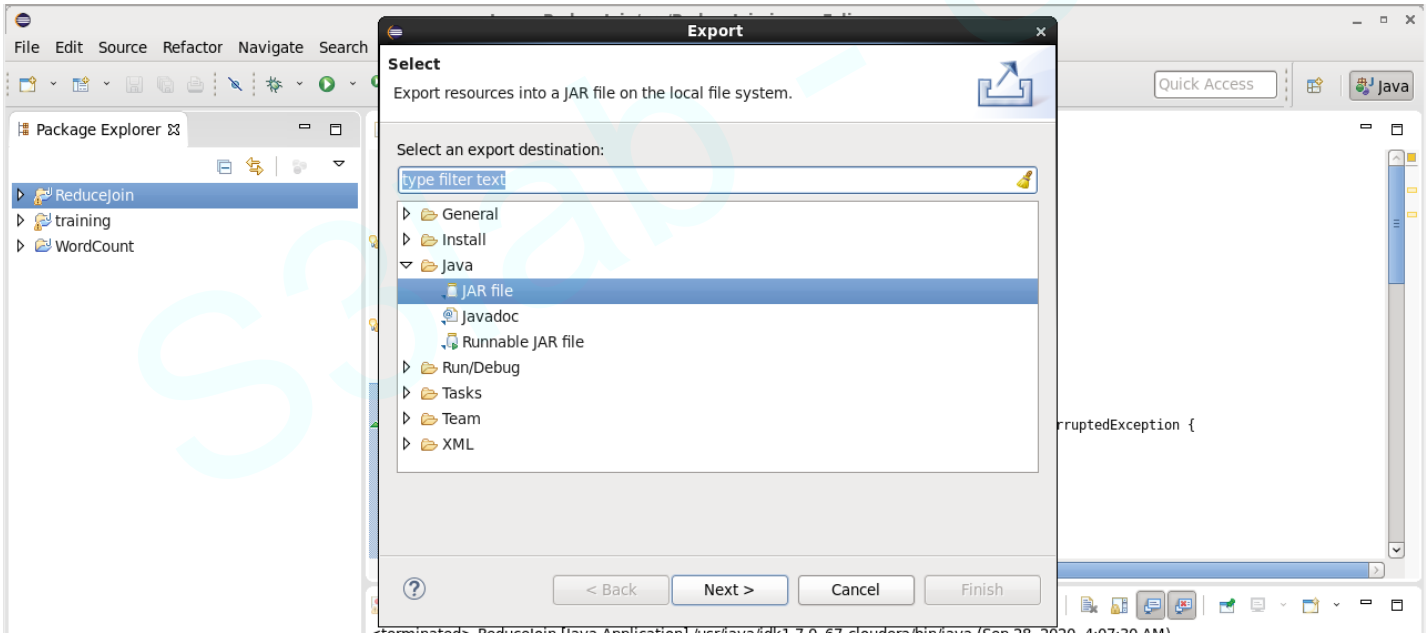
Reduce side Join

- Create new class ReduceJoin



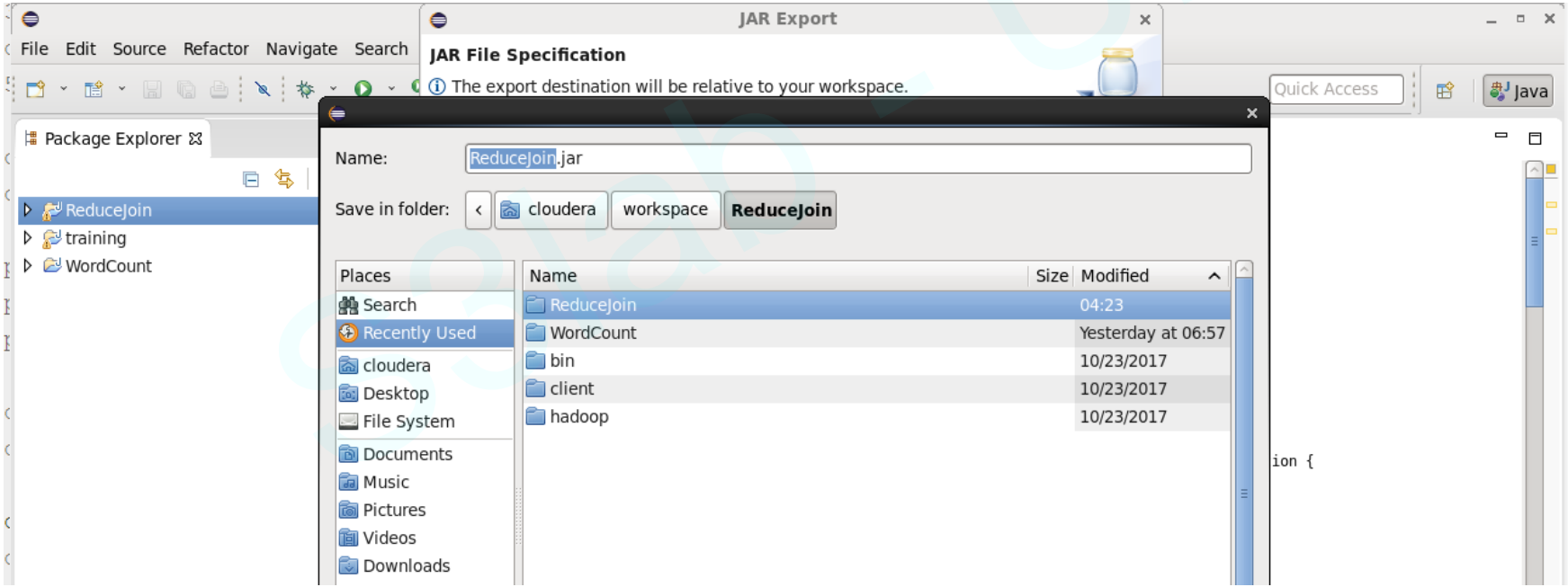
Reduce side Join

- Export to JAR file



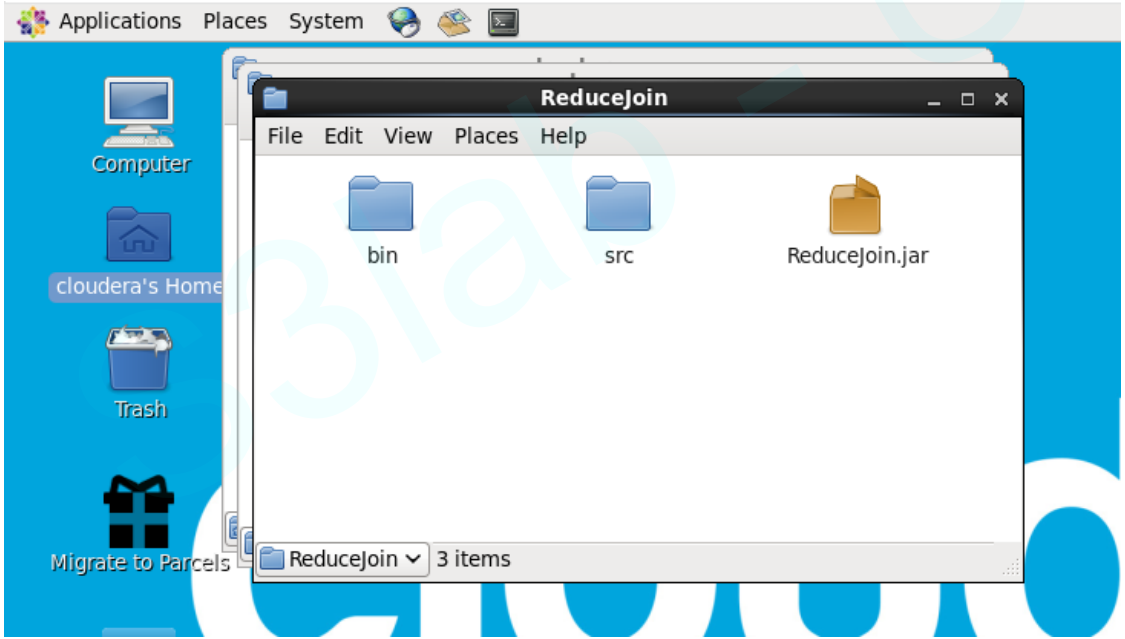
Reduce side Join

- Name the file **ReduceJoin.jar** and place it inside **/home/cloudera/workspace/ReduceJoin**



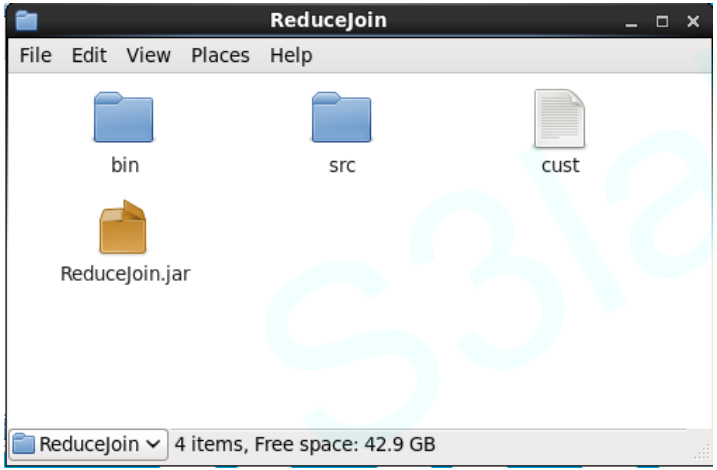
Reduce side Join

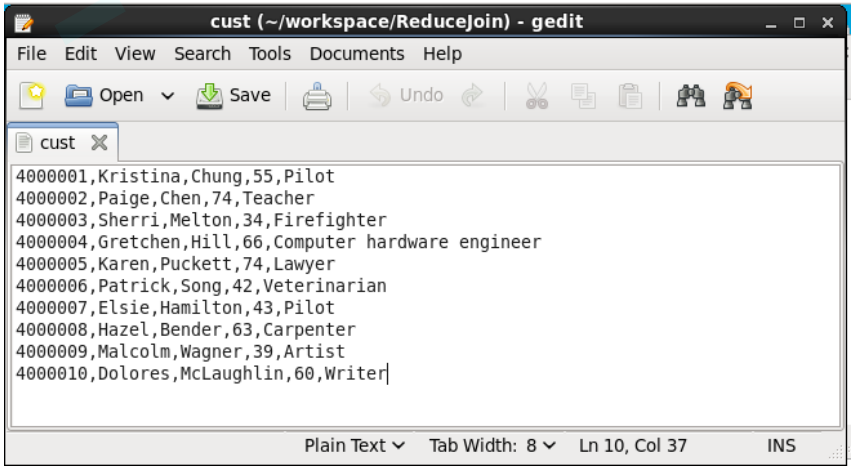
- Check if the JAR file has been created



Reduce side Join

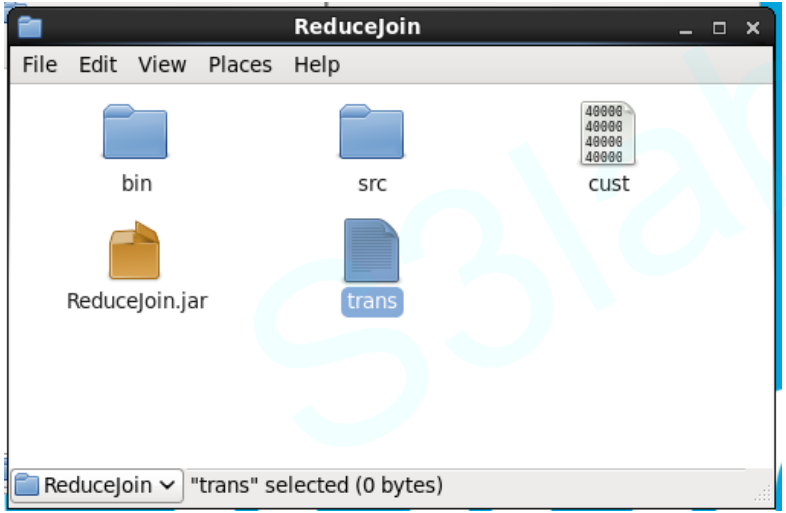
- Create input file for customer inside **/home/cloudera/workspace/ReduceJoin**

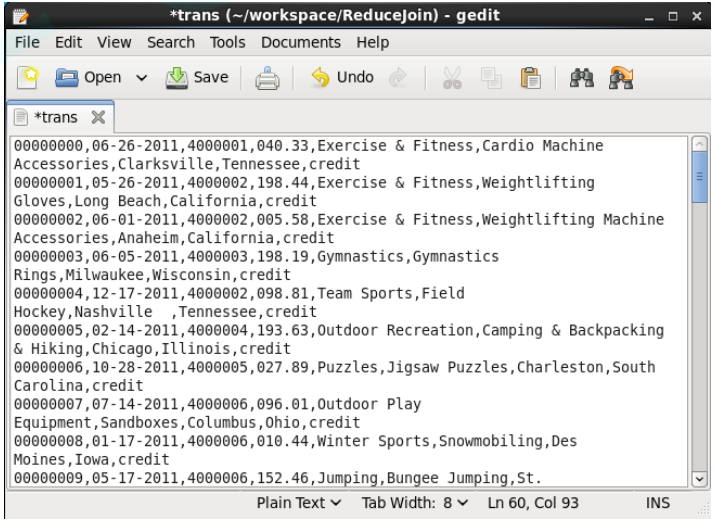




Reduce side Join

- Create input file for transaction inside **/home/cloudera/workspace/ReduceJoin**





Reduce side Join

- Create input directory in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir ReduceJoin
[cloudera@quickstart ~]$ hdfs dfs -mkdir ReduceJoin/input
```

Reduce side Join

- Move input files to HDFS

```
[cloudera@quickstart ~]$ cd /home/cloudera/workspace/ReduceJoin
[cloudera@quickstart ReduceJoin]$ hdfs dfs -put cust ReduceJoin/input
[cloudera@quickstart ReduceJoin]$ hdfs dfs -put trans ReduceJoin/input
[cloudera@quickstart ReduceJoin]$ hdfs dfs -ls Reduce/input
ls: `Reduce/input': No such file or directory
[cloudera@quickstart ReduceJoin]$ hdfs dfs -ls ReduceJoin/input
Found 2 items
-rw-r--r--   1 cloudera cloudera        356 2020-09-28 04:45 ReduceJoin/input/cust
-rw-r--r--   1 cloudera cloudera       5383 2020-09-28 04:46 ReduceJoin/input/trans
[cloudera@quickstart ReduceJoin]$
```

Reduce side Join

- Check the content of **cust** file

```
[cloudera@quickstart ReduceJoin]$ hdfs dfs -cat ReduceJoin/input/cust
4000001,Kristina,Chung,55,Pilot
4000002,Paige,Chen,74,Teacher
4000003,Sherri,Melton,34,Firefighter
4000004,Gretchen,Hill,66,Computer hardware engineer
4000005,Karen,Puckett,74,Lawyer
4000006,Patrick,Song,42,Veterinarian
4000007,Elsie,Hamilton,43,Pilot
4000008,Hazel,Bender,63,Carpenter
4000009,Malcolm,Wagner,39,Artist
4000010,Dolores,McLaughlin,60,Writer
[cloudera@quickstart ReduceJoin]$
```

Reduce side Join

- Submit the job:

hadoop jar ReduceJoin.jar ReduceJoin ReduceJoin/input/cust ReduceJoin/input/trans ReduceJoin/output

```
[cloudera@quickstart ReduceJoin]$ hadoop jar ReduceJoin.jar ReduceJoin ReduceJoin/input/cust ReduceJoin/input/trans  ReduceJoin/output
20/09/28 05:17:44 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
20/09/28 05:17:45 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and exe
cute your application with ToolRunner to remedy this.
20/09/28 05:17:45 INFO input.FileInputFormat: Total input paths to process : 1
20/09/28 05:17:45 INFO input.FileInputFormat: Total input paths to process : 1
20/09/28 05:17:45 INFO mapreduce.JobSubmitter: number of splits:2
20/09/28 05:17:45 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1601275309549_0001
20/09/28 05:17:46 INFO impl.YarnClientImpl: Submitted application application_1601275309549_0001
20/09/28 05:17:46 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1601275309549_0001/
20/09/28 05:17:46 INFO mapreduce.Job: Running job: job_1601275309549_0001
20/09/28 05:17:54 INFO mapreduce.Job: Job job_1601275309549_0001 running in uber mode : false
20/09/28 05:17:54 INFO mapreduce.Job:  map 0% reduce 0%
20/09/28 05:18:06 INFO mapreduce.Job:  map 100% reduce 0%
20/09/28 05:18:14 INFO mapreduce.Job:  map 100% reduce 100%
20/09/28 05:18:14 INFO mapreduce.Job: Job job_1601275309549_0001 completed successfully
```

Reduce side Join

- Check the output directory in HDFS

```
[cloudera@quickstart ReduceJoin]$ hdfs dfs -ls ReduceJoin/output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2020-09-28 05:18 ReduceJoin/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera        214 2020-09-28 05:18 ReduceJoin/output/part-r-00000
[cloudera@quickstart ReduceJoin]$
```

Reduce side Join

- Check the output directory in HDFS and the result inside it

```
[cloudera@quickstart ReduceJoin]$ hdfs dfs -ls ReduceJoin/output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2020-09-28 05:18 ReduceJoin/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera        214 2020-09-28 05:18 ReduceJoin/output/part-r-00000
[cloudera@quickstart ReduceJoin]$ hdfs dfs -cat ReduceJoin/output/part*
 Kristina       8 651.049999
 Paige  6 706.970007
 Sherri 3 527.589996
 Gretchen       5 337.060005
 Karen  5 325.150005
 Patrick        5 539.380010
 Elsie  6 699.550003
 Hazel  10 859.419990
 Malcolm        6 457.829996
 Dolores        6 447.090004
[cloudera@quickstart ReduceJoin]$
```

Assigment

- Write program to output Min, Max, and Mean age of users for each Game Type

Hadoop Streaming:

Writing A Hadoop MapReduce Program In Python

Hadoop Streaming:

What is Hadoop Streaming?

- Hadoop Streaming is a utility that comes with the Hadoop distribution. It can be used to execute programs for big data analysis. Hadoop streaming can be performed using languages like Python, Java, PHP, Scala, Perl, UNIX, and many more. The utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. For example:

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar
-input myInputDirs
-output myOutputDir
-mapper /bin/cat
-reducer /bin/wc
```

Hadoop Streaming:

What is Hadoop Streaming?

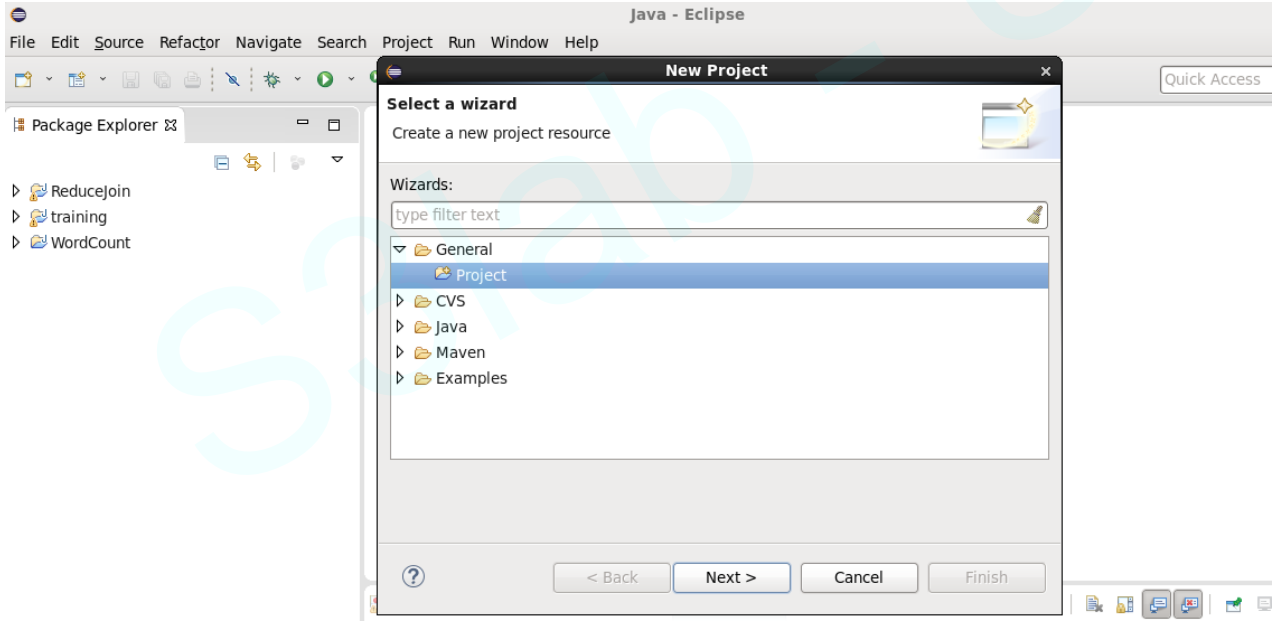
Parameter	Optional / Required	Description
-input directoryname or filename	Required	Input location for mapper
-output directoryname or filename	Required	Output location for mapper
-mapper executable	Required	Mapper executable
-reducer executable	Required	Reducer executable
-file filename	Optional	Make the mapper, reducer, or combiner executable available locally on the compute nodes

Hadoop Streaming:

Hadoop streaming tutorial start from here

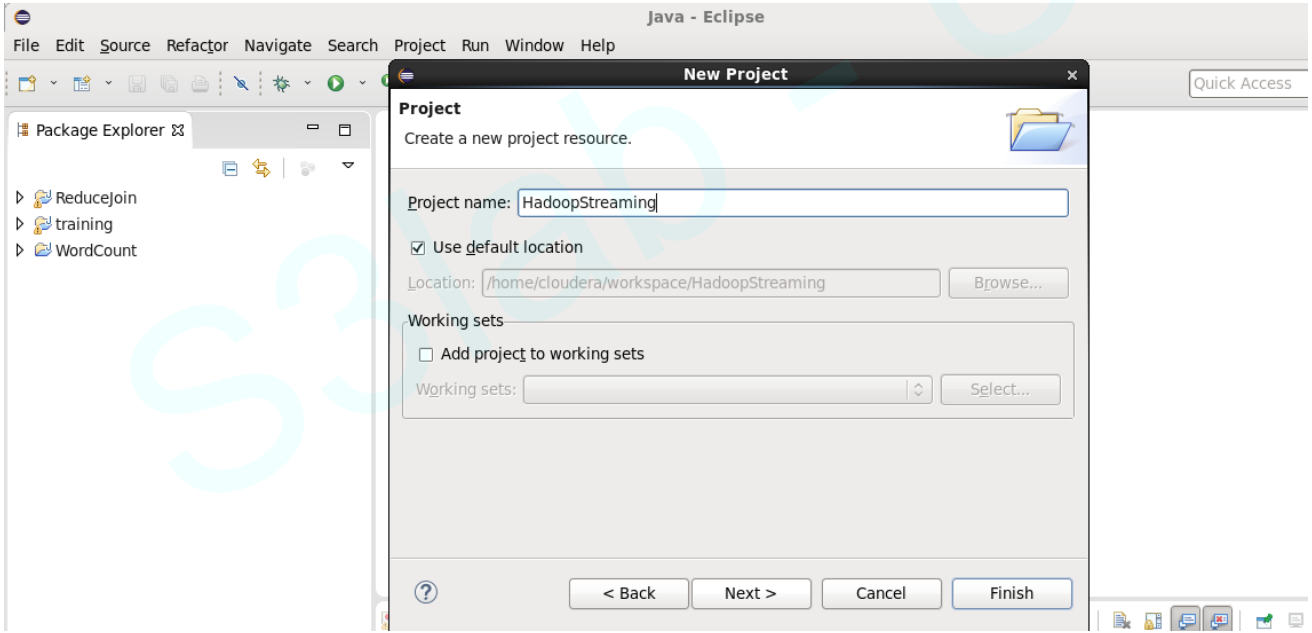
Hadoop Streaming:

- Create new General Project in Eclipse



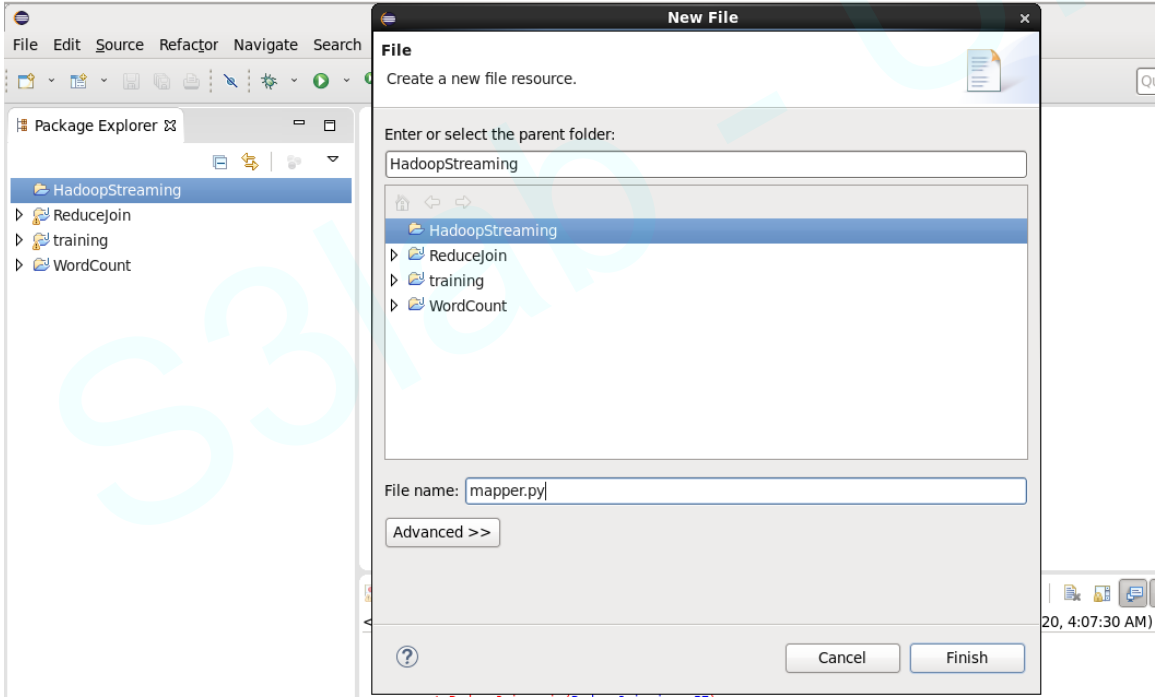
Hadoop Streaming:

- Name the project HadoopStreaming



Hadoop Streaming:

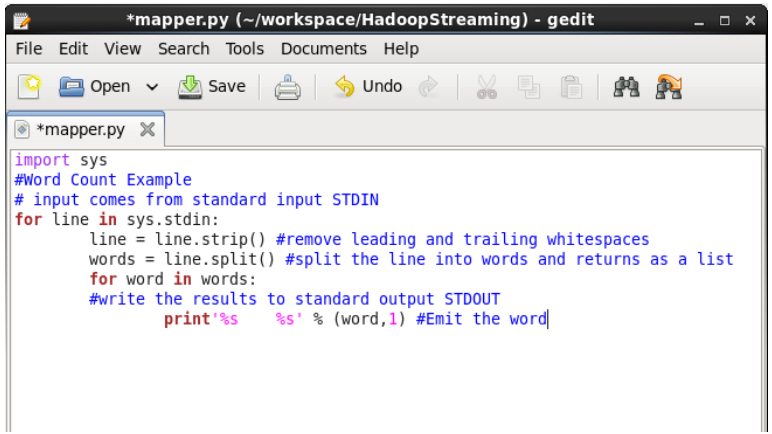
- Create new file **mapper.py**



Hadoop Streaming:

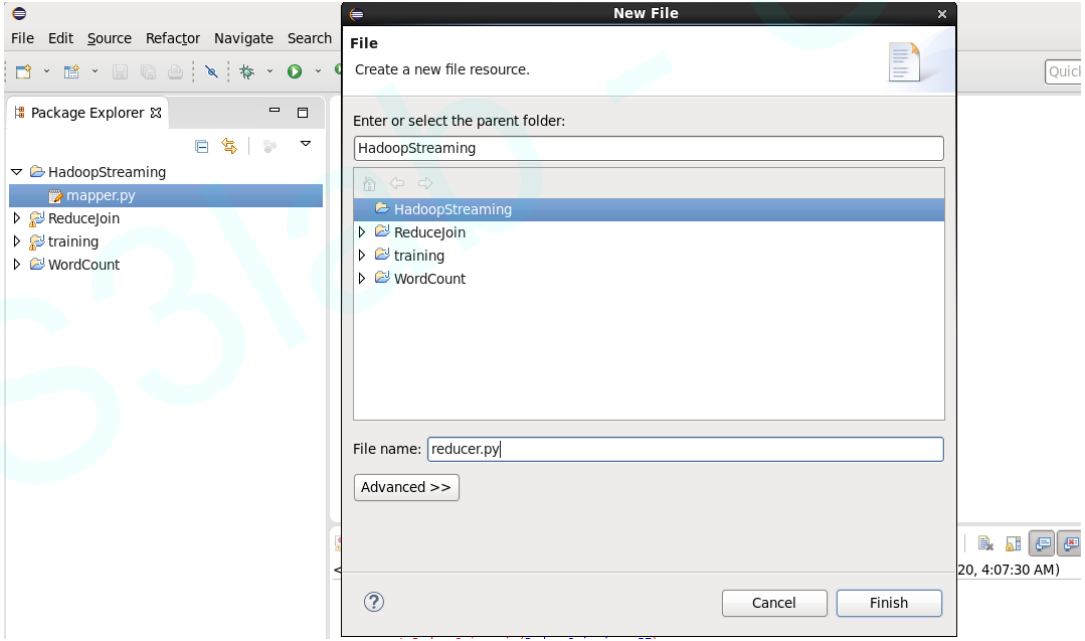
mapper.py

```
import sys
#Word Count Example
# input comes from standard input STDIN
for line in sys.stdin:
    line = line.strip() #remove leading and trailing whitespaces
    words = line.split() #split the line into words and returns as a list
    for word in words:
    #write the results to standard output STDOUT
          print'%s    %s' % (word,1) #Emit the word
```



Hadoop Streaming:

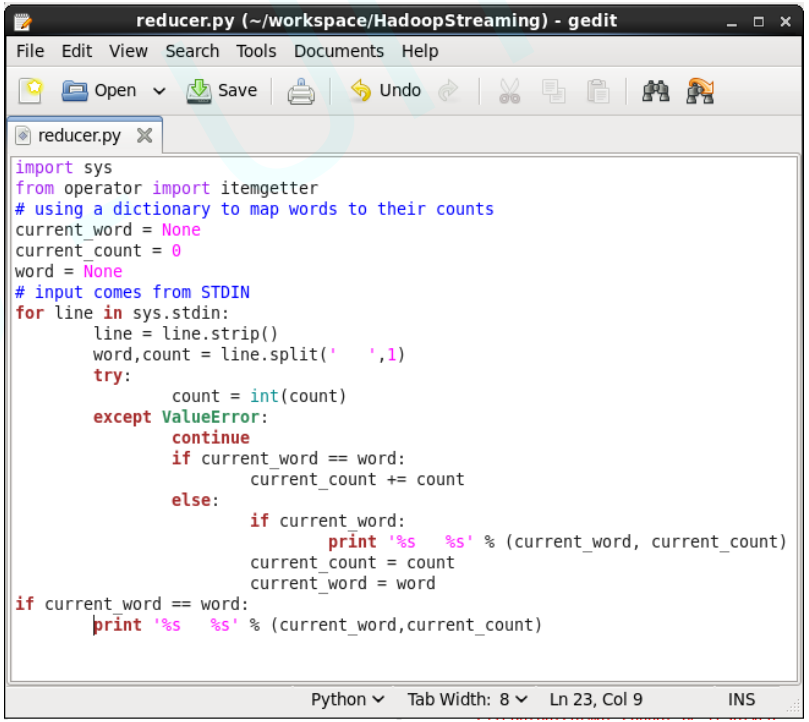
- Create new file **reducer.py**



Hadoop Streaming:

reducer.py

```
import sys
from operator import itemgetter
# using a dictionary to map words to their counts
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    word,count = line.split('    ',1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s    %s' % (current_word, current_count)
        current_count = count
        current_word = word
if current_word == word:
    print '%s    %s' % (current_word,current_count)
```



Hadoop Streaming:

- Locate the **HadoopStreaming** directory which contains **mapper.py** and **reducer.py**
- Create **word.txt** file

```
[cloudera@quickstart ~]$ cd /home/cloudera/workspace/HadoopStreaming
[cloudera@quickstart HadoopStreaming]$ echo 'Cat mouse lion deer Tiger lion Elephant lion deer' > word.txt
```

Hadoop Streaming:

- We can run mapper and reducer on local files (ex: word.txt). In order to run the Map and reduce on the Hadoop Distributed File System (HDFS), we need the Hadoop Streaming jar. So before we run the scripts on HDFS, let's run them locally to ensure that they are working fine.
- command: **cat word.txt | python mapper.py**

```
[cloudera@quickstart HadoopStreaming]$ cat word.txt | python mapper.py
Cat    1
mouse    1
lion    1
deer    1
Tiger    1
lion    1
Elephant    1
lion    1
deer    1
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Streaming:

- Run reducer.py
- command: **cat word.txt | python mapper.py | sort -k1,1 | python reducer.py**

```
[cloudera@quickstart HadoopStreaming]$ cat word.txt | python mapper.py | sort -k1,1 | python reducer.py
Cat   1
deer   2
Elephant   1
lion   3
mouse   1
Tiger   1
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Streaming:

Running the Python Code on Hadoop

- Create **HadoopStreaming** directory in HDFS
- Move **word.txt** to HDFS

```
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -mkdir HadoopStreaming
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -put word.txt  HadoopStreaming
```

Hadoop Streaming:

Running the Python Code on Hadoop

- So locate the Hadoop Streaming jar on your terminal and copy the path.
- Command: **ls /usr/lib/hadoop-mapreduce/hadoop-streaming.jar**

```
[cloudera@quickstart HadoopStreaming]$ ls /usr/lib/hadoop-mapreduce/hadoop-streaming.jar
/usr/lib/hadoop-mapreduce/hadoop-streaming.jar
[cloudera@quickstart HadoopStreaming]$
```

Hadoop Streaming:

Run the MapReduce job

- Command: **hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file /home/cloudera/workspace/HadoopStreaming/mapper.py -mapper 'python mapper.py' -file /home/cloudera/workspace/HadoopStreaming/reducer.py -reducer 'python reducer.py' -input HadoopStreaming/word.txt -output HSOutput**

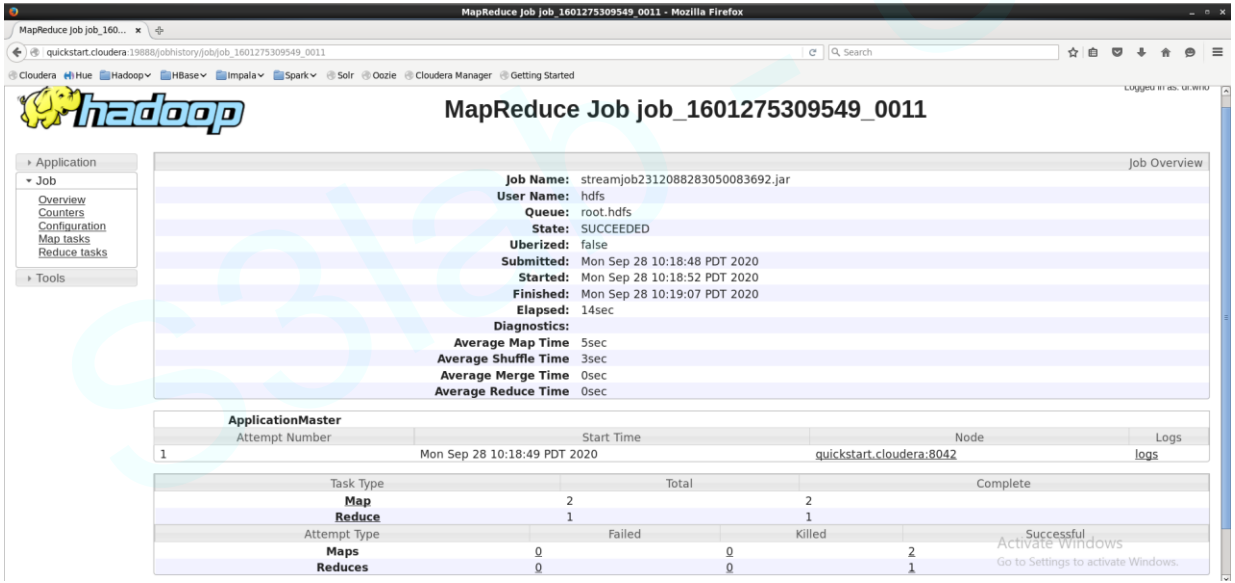
```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file /home/cloudera/workspace
/HadoopStreaming/mapper.py -mapper 'python mapper.py' -file /home/cloudera/workspace/HadoopStreaming/reducer.py -
reducer 'python reducer.py' -input HadoopStreaming/word.txt -output HSOutput
20/09/28 23:25:12 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [/home/cloudera/workspace/HadoopStreaming/mapper.py, /home/cloudera/workspace/HadoopStreaming/redu
cer.py] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob3457884780733202883.jar tm
pDir=null
20/09/28 23:25:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
20/09/28 23:25:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
20/09/28 23:25:14 INFO mapred.FileInputFormat: Total input paths to process : 1
20/09/28 23:25:14 INFO mapreduce.JobSubmitter: number of splits:2
20/09/28 23:25:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1601358520423_0002
```

Note: If we navigate to **HadoopStreaming** directory, we don't need to specify the full path **/home/cloudera/workspace/HadoopStreaming/mapper.py**

[cloudera@quickstart HadoopStreaming]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file mapper.py -mapper 'python mapper.py' -file reducer.py -reducer 'python reducer.py' -input HadoopStreaming/word.txt -output HSOutput

Hadoop Streaming:

- Hadoop provides a basic web interface for statistics and information



Hadoop Streaming:

- Check the output directory and the content of the result file

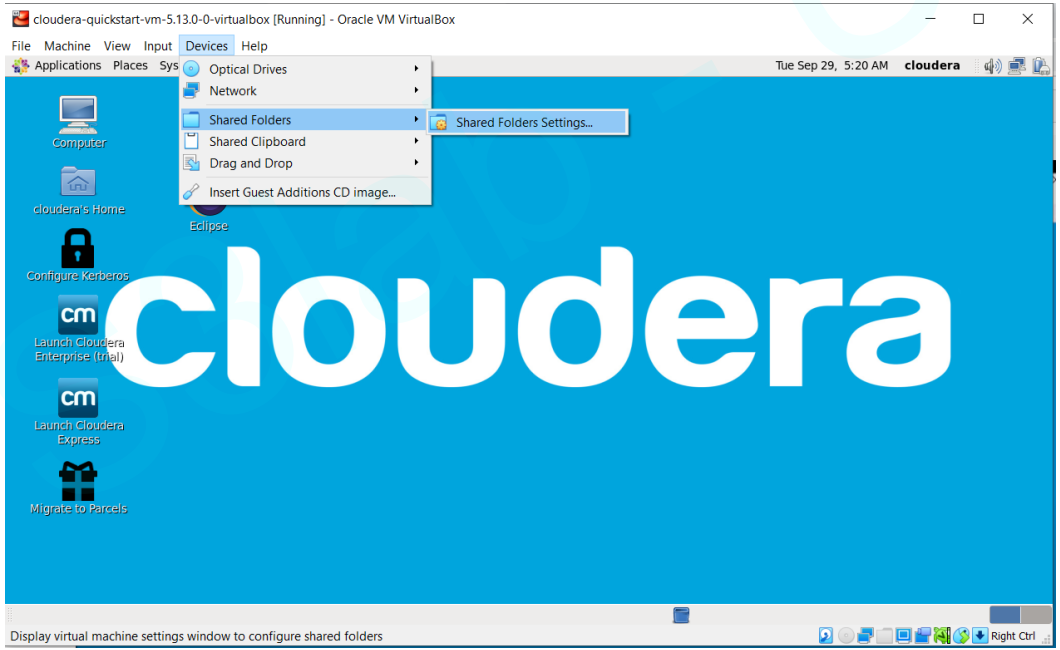
```
                Bytes Read=75
        File Output Format Counters
                Bytes Written=65
20/09/28 10:19:08 INFO streaming.StreamJob: Output directory: /user/cloudera/HSOutput
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls
Found 7 items
drwxr-xr-x   - hdfs     cloudera          0 2020-09-28 10:19 HSOutput
drwxr-xr-x   - cloudera cloudera          0 2020-09-28 10:03 HadoopStreaming
drwxr-xr-x   - cloudera cloudera          0 2020-09-28 05:17 ReduceJoin
drwxr-xr-x   - cloudera cloudera          0 2020-09-26 06:02 dataset
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:07 inputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-27 07:29 outputWC
drwxr-xr-x   - cloudera cloudera          0 2020-09-16 07:29 student
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -ls HSOutput
Found 2 items
-rw-r--r--   1 hdfs cloudera          0 2020-09-28 10:19 HSOutput/_SUCCESS
-rw-r--r--   1 hdfs cloudera         65 2020-09-28 10:19 HSOutput/part-00000
[cloudera@quickstart HadoopStreaming]$ hdfs dfs -cat HSOutput/part*
Cat   1
Elephant   1
Tiger   1
deer   2
lion   3
mouse   1
[cloudera@quickstart HadoopStreaming]$
```

Share Folders between VM and Host

- Just in case you’re finding the way to transfer files from Host to VM

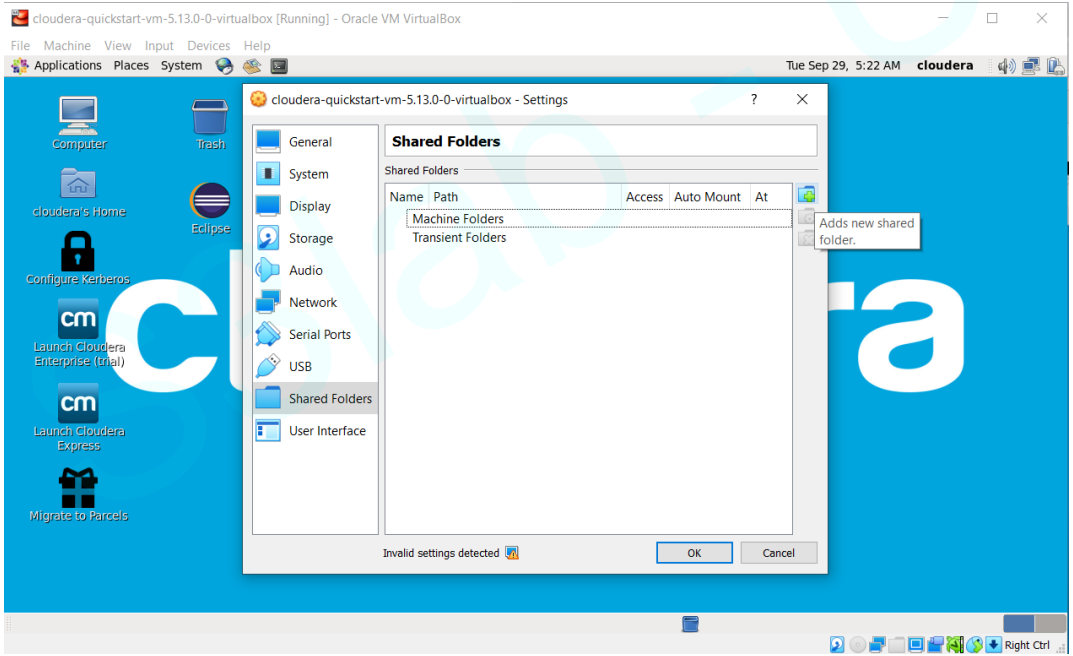
Share Folders between VM and Host

- Device -> Shared Folders -> Shared Folders Setting



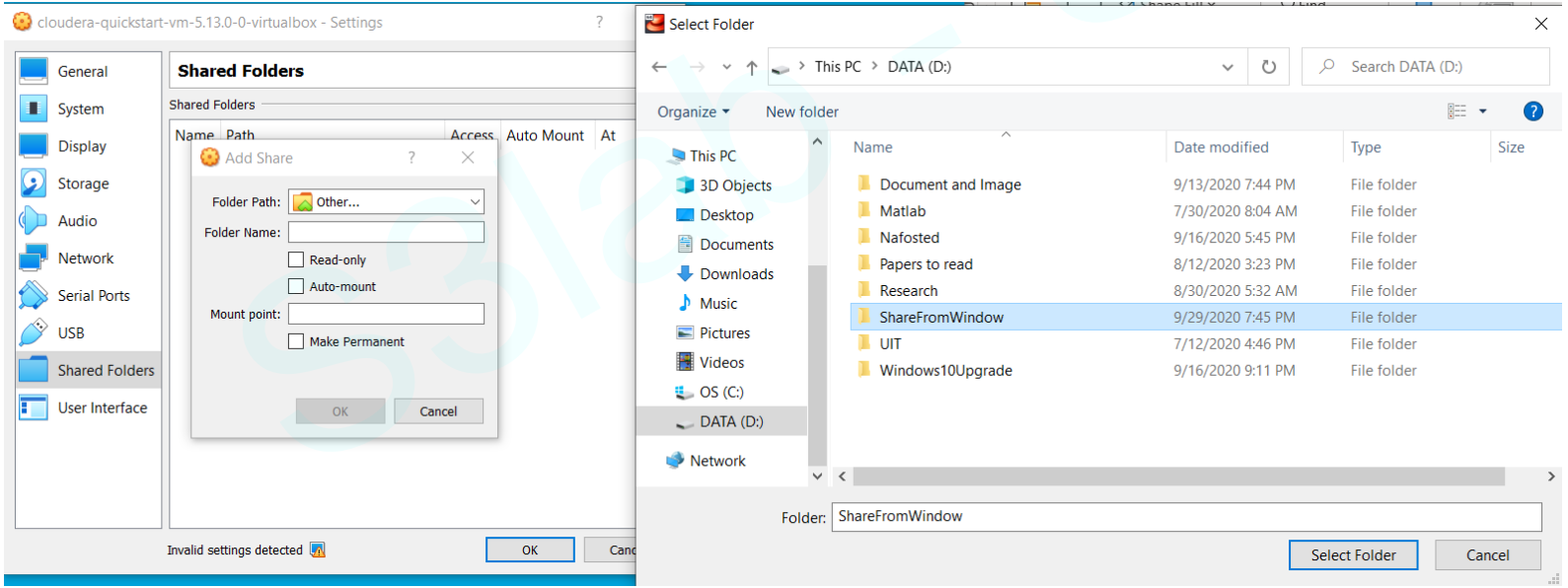
Share Folders between VM and Host

- Click Adds new shared folder



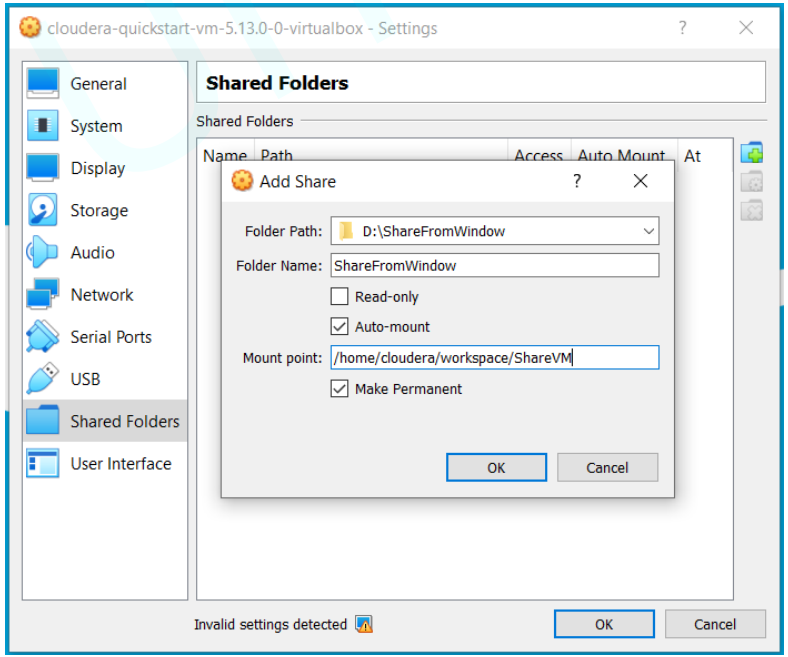
Share Folders between VM and Host

- Navigate to any folder on Window



Share Folders between VM and Host

- ***Folder Path*** - folder in the host (Windows 10)
- ***Folder Name*** - repeat the folder name from above
- ***Auto-mount*** - check this option
- ***Mount point*** - guest OS folder where the shared folder will mount (will be created if it does not exist)
- ***Make Permanent*** - check this option



Share Folders between VM and Host

- The ShareVM folder will be created (if not exist) after you reboot the VM.
- You can create the **ShareVM** folder in guest OS (Cloudera) by yourself (so no need to reboot)
- Command: **mkdir /home/cloudera/workspace/ShareVM**

```
[cloudera@quickstart ~]$ ls /home/cloudera
cloudera-manager          Downloads                  lib       student.java
cm_api.py                 eclipse                    Music     Templates
codegen_categories.java   enterprise-deployment.json parcels   Videos
Desktop                   express-deployment.json    Pictures  workspace
Documents                 kerberos                   Public
[cloudera@quickstart ~]$ ls /home/cloudera/workspace
training
[cloudera@quickstart ~]$ mkdir /home/cloudera/workspace/ShareVM
```

Share Folders between VM and Host

- Switch to root account (password: cloudera)
- Mount the shared folder:

Command: **Mount –t vboxsf ShareFromWindow /home/cloudera/workspace/ShareVM**

```
[cloudera@quickstart ~]$ mount -t vboxsf ShareFromWindow /home/cloudera/workspace/ShareVM
mount: only root can do that
[cloudera@quickstart ~]$ su
Password:
[root@quickstart cloudera]# mount -t vboxsf ShareFromWindow /home/cloudera/workspace/ShareVM
[root@quickstart cloudera]#
```

Share Folders between VM and Host

- Put some files to the shared folder in Window
- Click the newly created shortcut in Cloudera VM and check if the files appear