Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Big Data Engineering (DJ19DSL604)
AY: 2022-23

Experiment 1

(Hadoop Ecosystem)

Name : Sarvagya Singh
SAPID : 60009200030
BATCH : K1

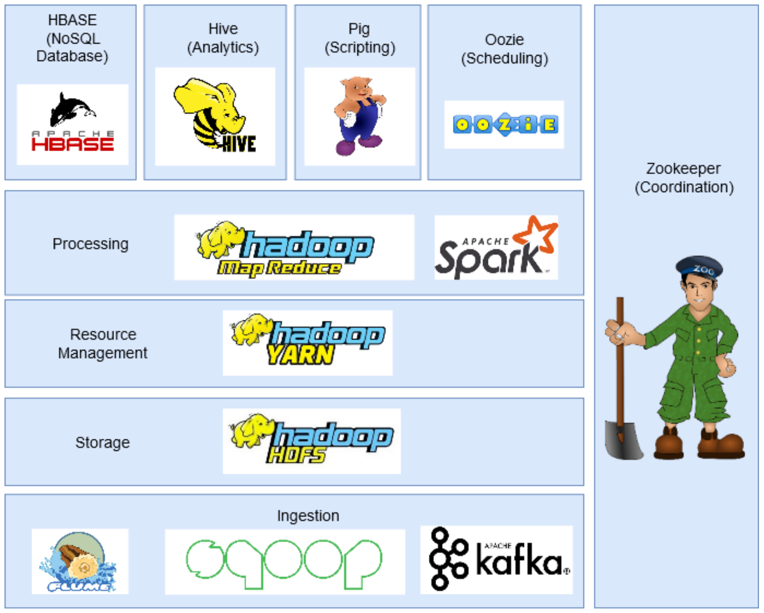
Aim: Hands on experience on Hadoop Ecosystem and MapReduce.

Theory:

1. Understanding inputs and outputs of MapReduce Concept of Hadoop:

Apache Hadoop is an open-source framework based on Google's file system that can deal with big data in a distributed environment. This distributed environment is built up of a cluster of machines that work closely together to give an impression of a single working machine.

Components of Hadoop Ecosystem:



2. The Map Tasks, grouping by Key, The Reduce Tasks, Combiners, Details of MapReduce Execution:

To handle Big Data, Hadoop relies on the MapReduce algorithm introduced by Google and makes it easy to distribute a job and run it in parallel in a cluster. It essentially divides a single task into multiple tasks and processes them on different machines.

The MapReduce framework operates on key-value pairs, that is, the framework views the input to the job as a set of key-value pairs and produces a set of key-value pair as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence, it is required to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

Both the input and output format of a MapReduce job are in the form of key-value pairs −

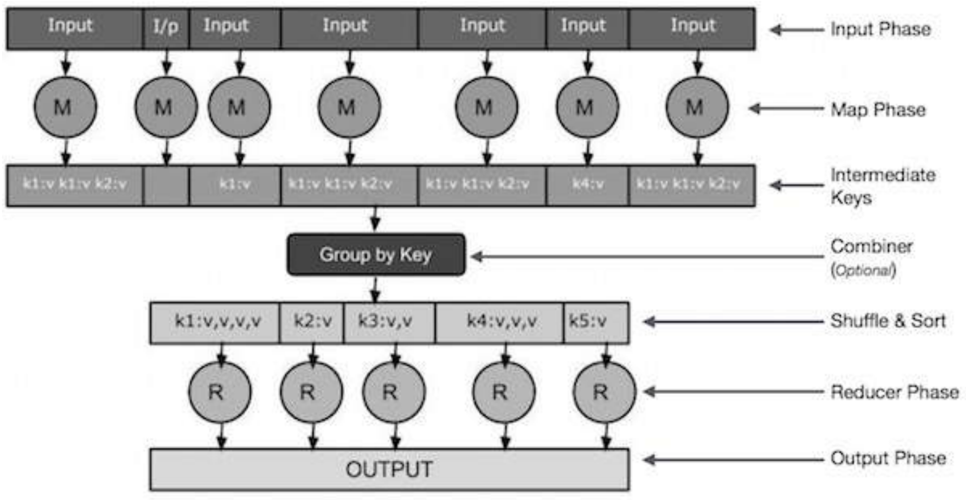
(Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3> (Output).

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

The MapReduce algorithm contains two important tasks, namely **Map** and **Reduce**.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

The reduce task is always performed after the map job.



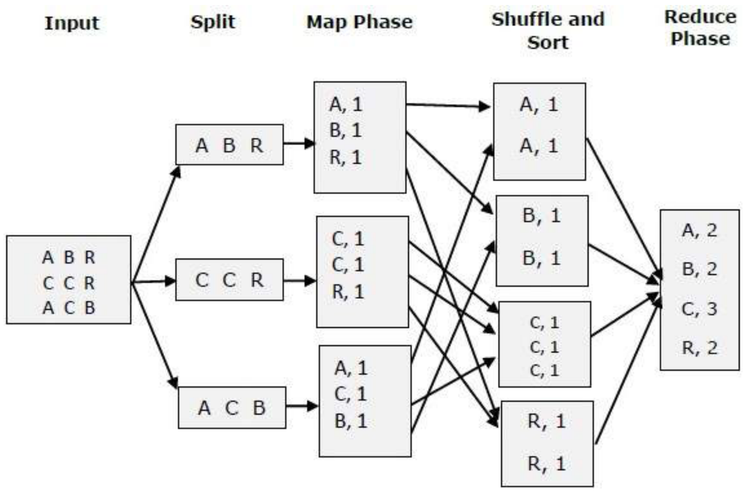
- **Input Phase −** Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map −** Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys −** They key-value pairs generated by the mapper are known as intermediate keys.

- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.

- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

- **Output Phase** − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.



Below is the simple structure of mapper, reducer and driver java files: (the highlighted text needs to be changed as per the problem statement)

Mapper Code:

```
// Importing all libraries
public class MapperClassName extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {

        // Map function
        public void map(LongWritable key, Text value, OutputCollector<Text,
                                    IntWritable> output, Reporter rep) throws IOException
```

```
          {

                    String line = value.toString();

                    // Write your Code here
                    output.collect(new Text(finalvariablethatcontainsdata), new IntWritable(1));
}
```

Reducer Code:

```
// Importing libraries
public class ReducerClassName extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable> {

          // Reduce function
          public void reduce(Text key, Iterator<IntWritable> value, OutputCollector<Text,
IntWritable> output, Reporter rep) throws IOException
          {

                    // Write your code here

                    output.collect(key, new IntWritable(finalvariablethatcontainsdata));
          }
}
```

Driver Code:

```
// Importing libraries

public class DriverClassName {
   public static void main(String[] args) {
      JobClient my_client = new JobClient();
      // Create a configuration object for the job
      JobConf job_conf = new JobConf(DriverClassName.class);

      // Set a name of the Job
      job_conf.setJobName("Nameofthejob");

      // Specify data type of output key and value
      job_conf.setOutputKeyClass(Text.class);
      job_conf.setOutputValueClass(IntWritable.class);

      // Specify names of Mapper and Reducer Class
      job_conf.setMapperClass(foldernamecontainingclassfiles.MapperClassName.class);
      job_conf.setReducerClass(foldernamecontainingclassfiles.ReducerClassName.class);

      // Specify formats of the data type of Input and output
```

```
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);

        // Set input and output directories using command line arguments,
        //arg[0] = name of input directory on HDFS, and arg[1] =  name of output directory to be
created to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Hadoop Distributed File System (HDFS) - Command line:

It is the storage component of Hadoop that stores data in the form of files.

Each file is divided into blocks of 128MB (configurable) and stores them on different machines in the cluster.

It has a master-slave architecture with two main components: Name Node and Data Node.

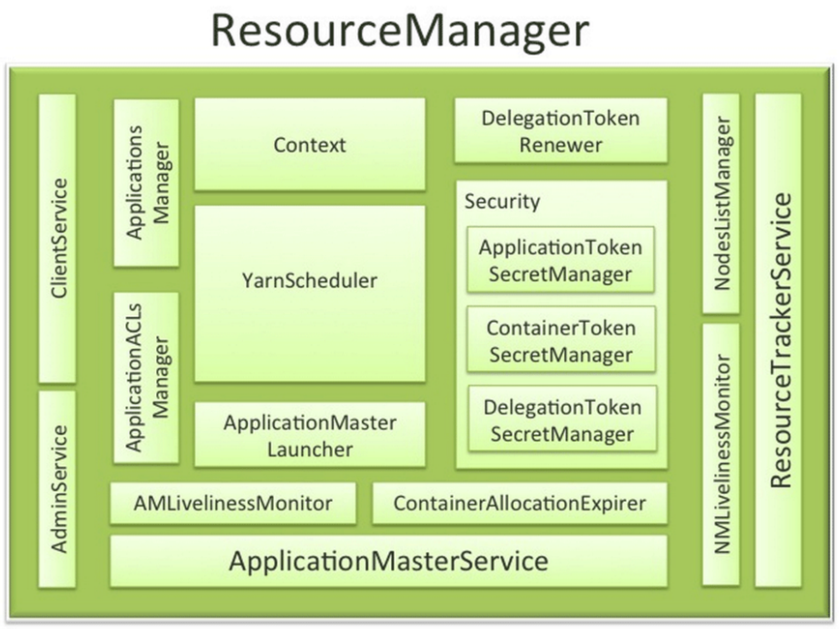
- **Name node** is the master node and there is only one per cluster. Its task is to know where each block belonging to a file is lying in the cluster
- **Data node** is the slave node that stores the blocks of data and there are more than one per cluster. Its task is to retrieve the data as and when required. It keeps in constant touch with the Name node through heartbeats

4. Overview of resource management – YARN:

YARN or Yet Another Resource Negotiator manages resources in the cluster and manages the applications over Hadoop. It allows data stored in HDFS to be processed and run by various data processing engines such as batch processing, stream processing, interactive processing, graph processing, and many more. This increases efficiency with the use of YARN.

The Resource Manager is the core component of YARN – Yet Another Resource Negotiator. In analogy, it occupies the place of JobTracker of MRV1. Hadoop YARN is designed to provide a generic and flexible framework to administer the computing resources in the Hadoop cluster.

In this direction, the YARN Resource Manager Service (RM) is the central controlling authority for resource management and makes allocation decisions ResourceManager has two main components: Scheduler and ApplicationsManager.



Lab Assignments:

1. Implement the following for the dataset given below:
   https://www.kaggle.com/datasets/narayan63/netflix-popular-movies-dataset
   a. Apply various HDFS commands on command line to get the appropriate file structure to run the MapReduce code.
   b. Write a Java MapReduce program to find out the list of movies who have been given ratings >= 8.
2. Implement a Java MapReduce program to display the occurrences of each character from a file.

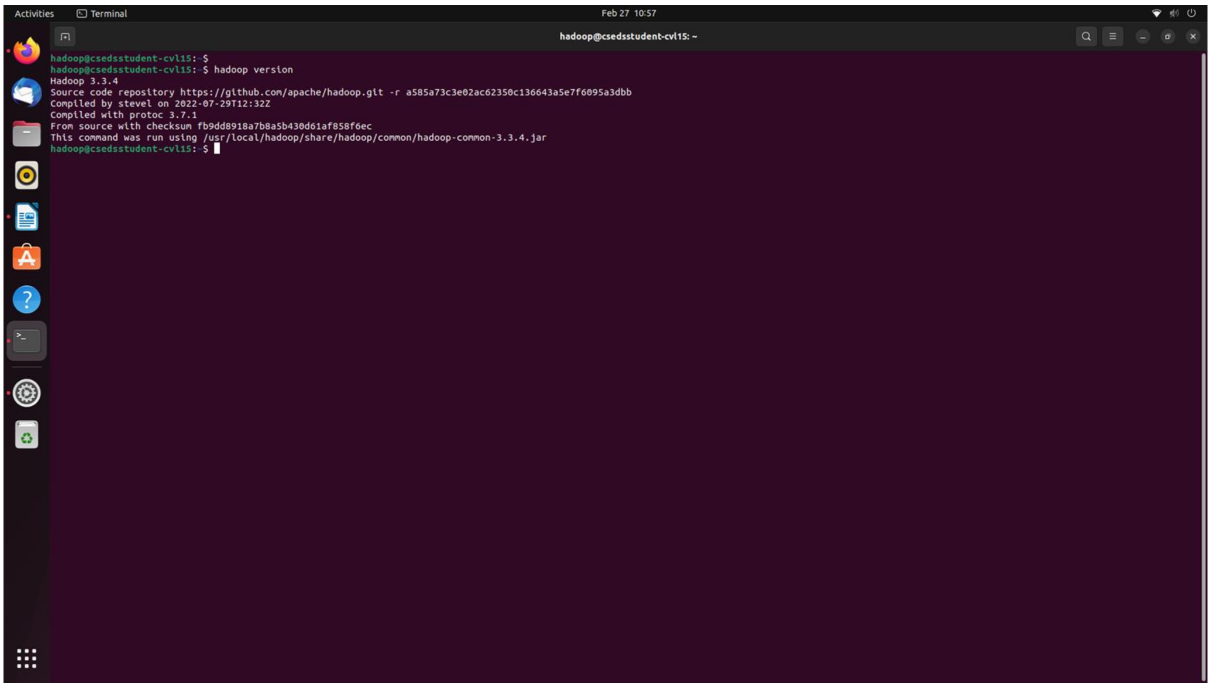
BIG DATA ENGINEERING

EXPERIMENT 1.a

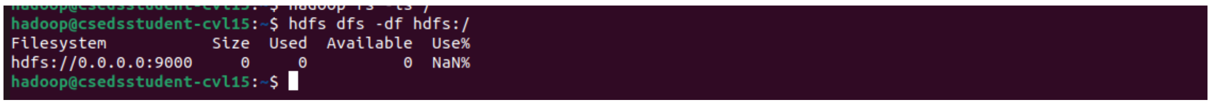
Name : Sarvagya Singh
SAPID : 60009200030
BATCH : K1

Aim: Hands on experience on Hadoop Ecosystem and MapReduce.

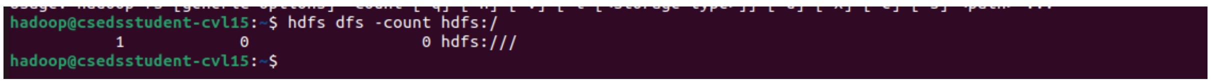
Implementation:



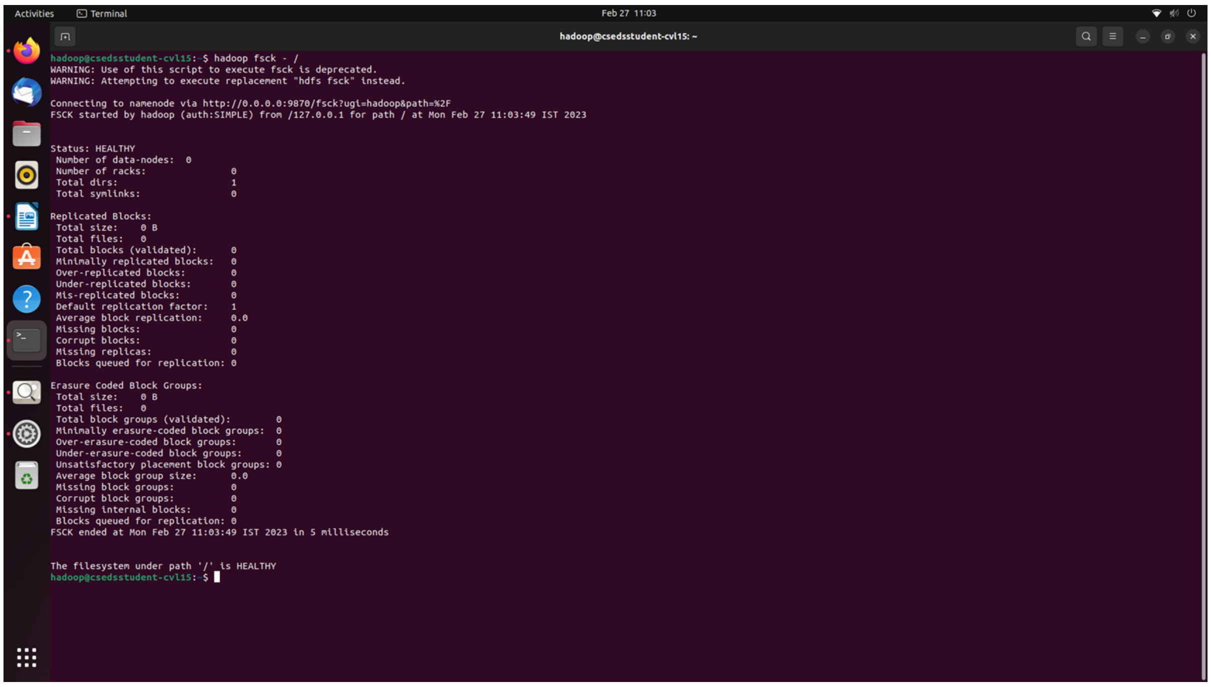
```
hadoop@csedsstudent-cvl15:~$
hadoop@csedsstudent-cvl15:~$ hadoop version
Hadoop 3.3.4
Source code repository https://github.com/apache/hadoop.git -r a585a73c3e02ac62350c136643a5e7f6095a3dbb
Compiled by stevel on 2022-07-29T12:32Z
Compiled with protoc 3.7.1
From source with checksum fb9dd8918a7b8a5b430d61af858f6ec
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.4.jar
hadoop@csedsstudent-cvl15:~$
```



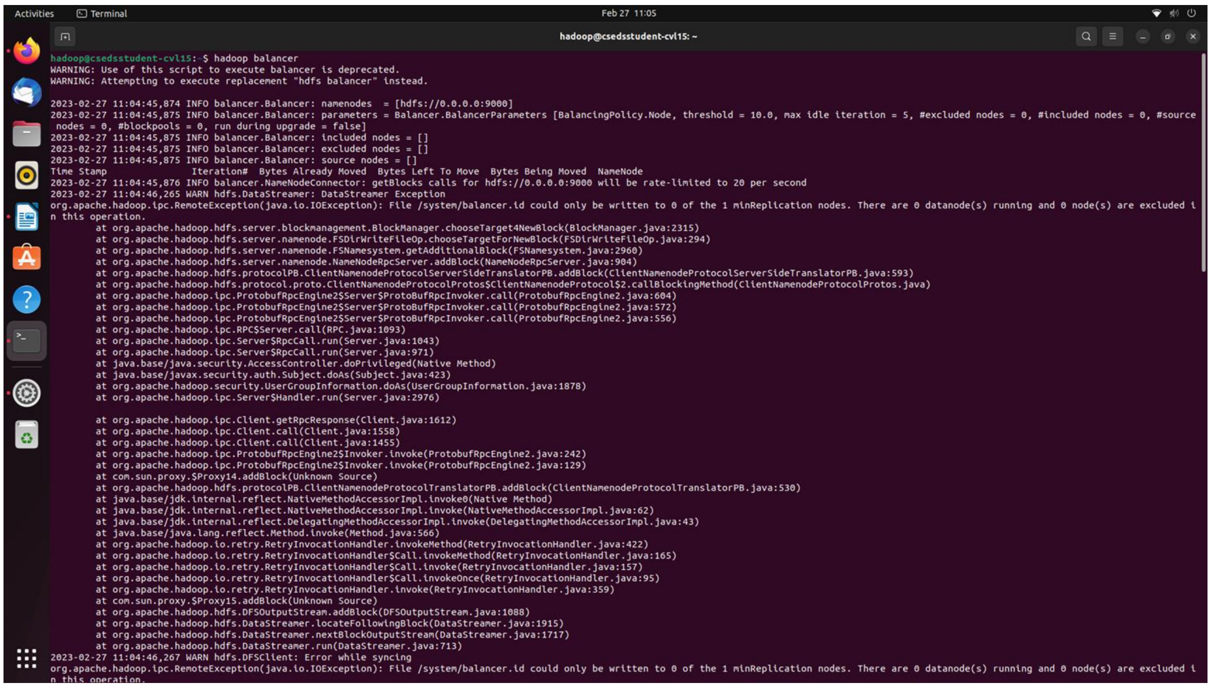
```
hadoop@csedsstudent-cvl15:~$ hdfs dfs -df hdfs:/
Filesystem            Size  Used  Available  Use%
hdfs://0.0.0.0:9000      0     0          0  NaN%
hadoop@csedsstudent-cvl15:~$
```



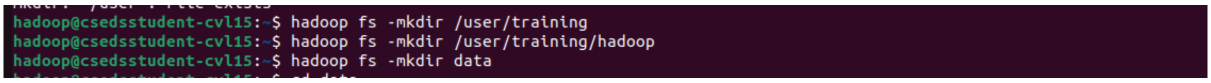
```
hadoop@csedsstudent-cvl15:~$ hdfs dfs -count hdfs:/
           1            0                  0 hdfs:///
hadoop@csedsstudent-cvl15:~$
```



```
hadoop@csedsstudent-cvl15:~$ hadoop fsck - /
WARNING: Use of this script to execute fsck is deprecated.
WARNING: Attempting to execute replacement "hdfs fsck" instead.

Connecting to namenode via http://0.0.0.0:9870/fsck?ugi=hadoop&path=%2F
FSCK started by hadoop (auth:SIMPLE) from /127.0.0.1 for path / at Mon Feb 27 11:03:49 IST 2023


Status: HEALTHY
 Number of data-nodes:  0
 Number of racks:               0
 Total dirs:                    1
 Total symlinks:                0

Replicated Blocks:
 Total size:    0 B
 Total files:   0
 Total blocks (validated):      0
 Minimally replicated blocks:   0
 Over-replicated blocks:        0
 Under-replicated blocks:       0
 Mis-replicated blocks:         0
 Default replication factor:    1
 Average block replication:     0.0
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              0
 Blocks queued for replication: 0

Erasure Coded Block Groups:
 Total size:    0 B
 Total files:   0
 Total block groups (validated):        0
 Minimally erasure-coded block groups:  0
 Over-erasure-coded block groups:       0
 Under-erasure-coded block groups:      0
 Unsatisfactory placement block groups: 0
 Average block group size:      0.0
 Missing block groups:          0
 Corrupt block groups:          0
 Missing internal blocks:       0
 Blocks queued for replication: 0
FSCK ended at Mon Feb 27 11:03:49 IST 2023 in 5 milliseconds


The filesystem under path '/' is HEALTHY
hadoop@csedsstudent-cvl15:~$
```



```
hadoop@csedsstudent-cvl15:~$ hadoop balancer
WARNING: Use of this script to execute balancer is deprecated.
WARNING: Attempting to execute replacement "hdfs balancer" instead.

2023-02-27 11:04:45,874 INFO balancer.Balancer: namenodes  = [hdfs://0.0.0.0:9000]
2023-02-27 11:04:45,875 INFO balancer.Balancer: parameters = Balancer.BalancerParameters [BalancingPolicy.Node, threshold = 10.0, max idle iteration = 5, #excluded nodes = 0, #included nodes = 0, #source
 nodes = 0, #blockpools = 0, run during upgrade = false]
2023-02-27 11:04:45,875 INFO balancer.Balancer: included nodes = []
2023-02-27 11:04:45,875 INFO balancer.Balancer: excluded nodes = []
2023-02-27 11:04:45,875 INFO balancer.Balancer: source nodes = []
Time Stamp               Iteration#  Bytes Already Moved  Bytes Left To Move  Bytes Being Moved  NameNode
2023-02-27 11:04:45,876 INFO balancer.NameNodeConnector: getBlocks calls for hdfs://0.0.0.0:9000 will be rate-limited to 20 per second
2023-02-27 11:04:46,265 WARN hdfs.DataStreamer: DataStreamer Exception
org.apache.hadoop.ipc.RemoteException(java.io.IOException): File /system/balancer.id could only be written to 0 of the 1 minReplication nodes. There are 0 datanode(s) running and 0 node(s) are excluded i
n this operation.
        at org.apache.hadoop.hdfs.server.blockmanagement.BlockManager.chooseTarget4NewBlock(BlockManager.java:2315)
        at org.apache.hadoop.hdfs.server.namenode.FSDirWriteFileOp.chooseTargetForNewBlock(FSDirWriteFileOp.java:294)
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.getAdditionalBlock(FSNamesystem.java:2960)
        at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.addBlock(NameNodeRpcServer.java:904)
        at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolServerSideTranslatorPB.addBlock(ClientNamenodeProtocolServerSideTranslatorPB.java:593)
        at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos$ClientNamenodeProtocol$2.callBlockingMethod(ClientNamenodeProtocolProtos.java)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine2.java:604)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine2.java:572)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Server$ProtoBufRpcInvoker.call(ProtobufRpcEngine2.java:556)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:1093)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:1043)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:971)
        at java.base/java.security.AccessController.doPrivileged(Native Method)
        at java.base/javax.security.auth.Subject.doAs(Subject.java:423)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1878)
        at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2976)

        at org.apache.hadoop.ipc.Client.getRpcResponse(Client.java:1612)
        at org.apache.hadoop.ipc.Client.call(Client.java:1558)
        at org.apache.hadoop.ipc.Client.call(Client.java:1455)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Invoker.invoke(ProtobufRpcEngine2.java:242)
        at org.apache.hadoop.ipc.ProtobufRpcEngine2$Invoker.invoke(ProtobufRpcEngine2.java:129)
        at com.sun.proxy.$Proxy14.addBlock(Unknown Source)
        at org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.addBlock(ClientNamenodeProtocolTranslatorPB.java:530)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:566)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInvocationHandler.java:422)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeMethod(RetryInvocationHandler.java:165)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invoke(RetryInvocationHandler.java:157)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeOnce(RetryInvocationHandler.java:95)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:359)
        at com.sun.proxy.$Proxy15.addBlock(Unknown Source)
        at org.apache.hadoop.hdfs.DFSOutputStream.addBlock(DFSOutputStream.java:1088)
        at org.apache.hadoop.hdfs.DataStreamer.locateFollowingBlock(DataStreamer.java:1915)
        at org.apache.hadoop.hdfs.DataStreamer.nextBlockOutputStream(DataStreamer.java:1717)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:713)
2023-02-27 11:04:46,267 WARN hdfs.DFSClient: Error while syncing
org.apache.hadoop.ipc.RemoteException(java.io.IOException): File /system/balancer.id could only be written to 0 of the 1 minReplication nodes. There are 0 datanode(s) running and 0 node(s) are excluded i
n this operation.
```



```
hadoop@csedsstudent-cvl15:~$ hadoop fs -mkdir /user/training
hadoop@csedsstudent-cvl15:~$ hadoop fs -mkdir /user/training/hadoop
hadoop@csedsstudent-cvl15:~$ hadoop fs -mkdir data
```

Conclusion: We have performed the various commands to achieve the given task.