

EXPERIMENT NO. 1 A (Group A)

Aim: Set up and Configuration Hadoop Using CloudEra/ Google Cloud BigQuery. Databricks Lakehouse Platform. Snowflake. Amazon Redshift

Theory:

Hadoop software can be installed in three modes of

Hadoop is a Java-based programming framework that supports the processing and storage of extremely large datasets on a cluster of inexpensive machines. It was the first major open source project in the big data playing field and is sponsored by the Apache Software Foundation.

Hadoop-2.7.3 is comprised of four main layers:

- **Hadoop Common** is the collection of utilities and libraries that support other Hadoop modules.
- **HDFS**, which stands for Hadoop Distributed File System, is responsible for persisting data to disk.
- **YARN**, short for Yet Another Resource Negotiator, is the "operating system" for HDFS.
- **MapReduce** is the original processing model for Hadoop clusters. It distributes work within the cluster or map, then organizes and reduces the results from the nodes into a response to a query. Many other processing models are available for the 2.x version of Hadoop.

Hadoop clusters are relatively complex to set up, so the project includes a stand-alone mode which is suitable for learning about Hadoop, performing simple operations, and debugging.

Procedure:

we'll install Hadoop in stand-alone mode and run one of the example MapReduce programs it includes to verify the installation.

Prerequisites:**Step1: Installing Java 8 version.**

Openjdk version "1.8.0_91"

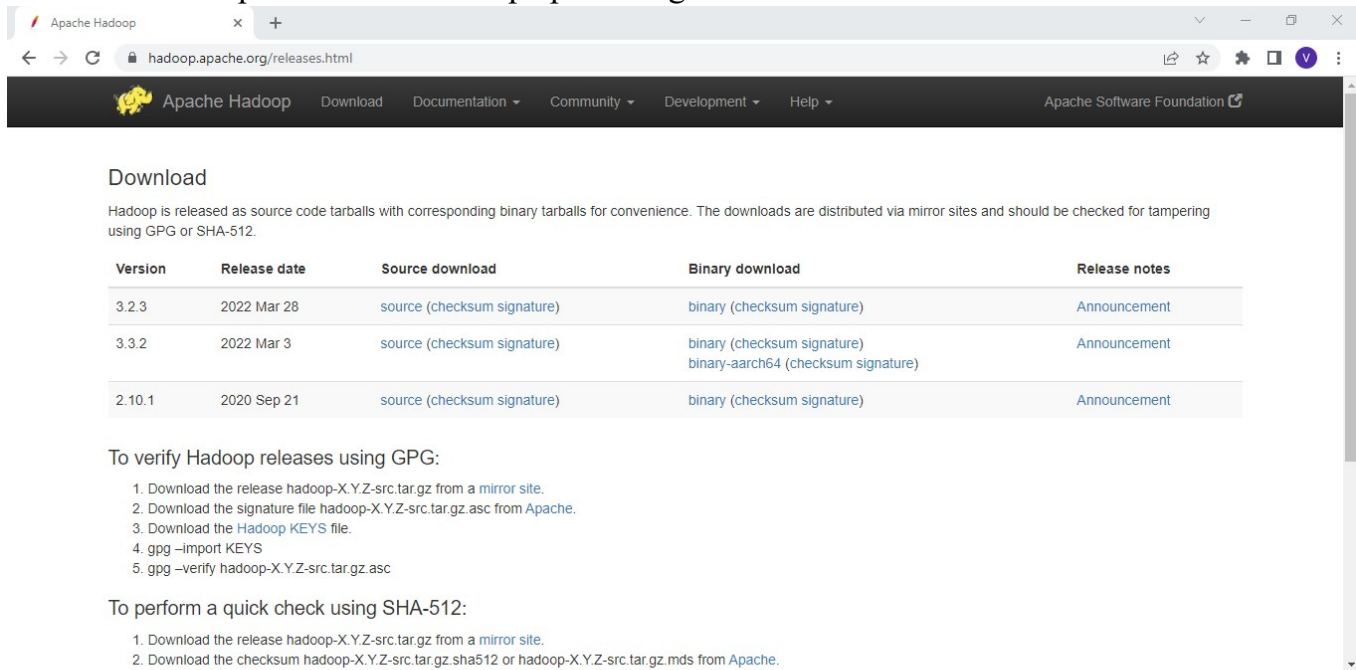
OpenJDK Runtime Environment (build 1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14)

OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode) This output verifies that OpenJDK has been successfully installed. **Note:** To set the path for environment variables. i.e. JAVA_HOME

Step2: Installing Hadoop

With Java in place, we'll visit the Apache Hadoop Releases page to find the most recent stable release. Follow the binary for the current release:

Download Hadoop from www.hadoop.apache.org



Apache Hadoop

hadoop.apache.org/releases.html

Download

Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512.

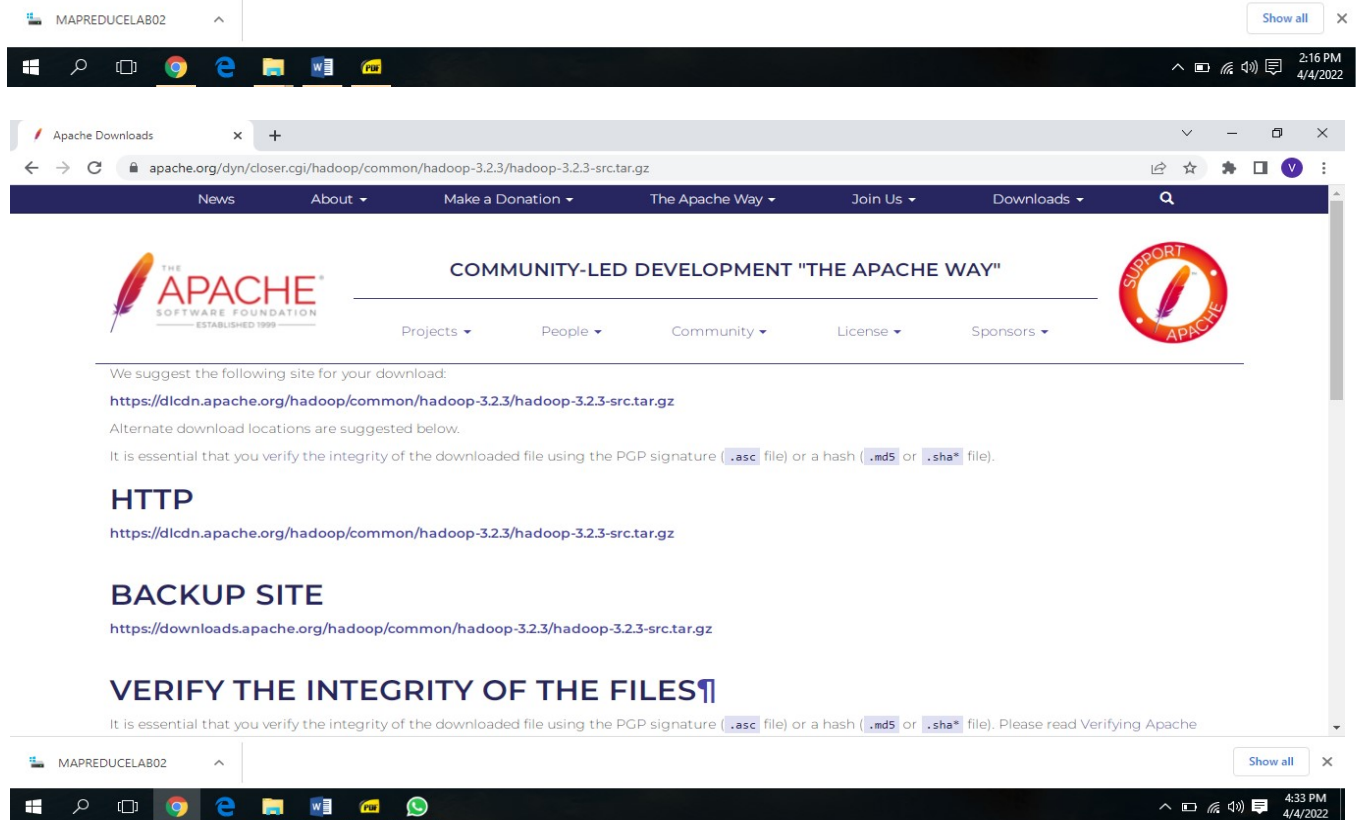
Version	Release date	Source download	Binary download	Release notes
3.2.3	2022 Mar 28	source (checksum signature)	binary (checksum signature)	Announcement
3.3.2	2022 Mar 3	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature)	Announcement
2.10.1	2020 Sep 21	source (checksum signature)	binary (checksum signature)	Announcement

To verify Hadoop releases using GPG:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the signature file `hadoop-X.Y.Z-src.tar.gz.asc` from Apache.
3. Download the **Hadoop KEYS** file.
4. `gpg --import KEYS`
5. `gpg --verify hadoop-X.Y.Z-src.tar.gz.asc`

To perform a quick check using SHA-512:

1. Download the release `hadoop-X.Y.Z-src.tar.gz` from a [mirror site](#).
2. Download the checksum `hadoop-X.Y.Z-src.tar.gz.sha512` or `hadoop-X.Y.Z-src.tar.gz.mds` from Apache.



Apache Downloads

apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.2.3/hadoop-3.2.3-src.tar.gz

News About Make a Donation The Apache Way Join Us Downloads

THE APACHE SOFTWARE FOUNDATION ESTABLISHED 1999

COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"

Projects People Community License Sponsors

We suggest the following site for your download:

<https://d1cdn.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3-src.tar.gz>

Alternate download locations are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file).

HTTP

<https://d1cdn.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3-src.tar.gz>

BACKUP SITE

<https://downloads.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3-src.tar.gz>

VERIFY THE INTEGRITY OF THE FILES

It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file). Please read [Verifying Apache](#)

Procedure to Run Hadoop

1. Install Apache Hadoop 2.2.0 in Microsoft Windows OS

If Apache Hadoop 2.2.0 is not already installed then follow the post Build, Install, Configure and Run Apache Hadoop 2.2.0 in Microsoft Windows OS.

2. Start HDFS (Namenode and Datanode) and YARN (Resource Manager and Node Manager)

Run following commands.

Command Prompt

C:\Users\abhijitg>cd c:\hadoop

c:\hadoop>sbin\start-dfs

c:\hadoop>sbin\start-yarn starting yarn

daemons

Namenode, Datanode, Resource Manager and Node Manager will be started in few minutes and ready to execute Hadoop **MapReduce** job in the Single Node (pseudo-distributed mode) cluster.

```

Apache Hadoop Distribution - hadoop namenode
infoPort=50075, ipcPort=50020, storageInfo=lv=-47;cid=CID-1a1d9f-efee-4d4e-9f03-2c23e5eb;nsid=28920020;c=0) storage DS-2081780230-50010-1383498502153
13/11/03 22:38:22 INFO net.NetworkTopology: Adding a new node: /default-rack/127.0.0.1:50010
13/11/03 22:38:22 INFO blockmanagement.BlockManager: BLOCK* processReport: Received first block report from 127.0.0.1:50010 after starting up or becoming active. Its block contents are no longer considered stale
13/11/03 22:38:22 INFO BlockStateChange: BLOCK* processReport: from DatanodeRegistration(127.0.0.1, storageID=DS-2081780230-50010-1383498502153, infoPort=50075, ipcPort=50020, storageInfo=lv=-47;cid=CID-1a1d9f-efee-4d4e-9f03-2c23e5eb;nsid=28920020;c=0), blocks: 0, processing time: 15 msec

Apache Hadoop Distribution - hadoop datanode
207-50010-1383498502153> service to localhost/127.0.0.1:9000
13/11/03 22:38:22 INFO datanode.DataNode: BlockReport of 0 blocks took 0 msec to generate and 94 msec for RPC and NN processing
13/11/03 22:38:22 INFO datanode.DataNode: sent block report, processed command: org.apache.hadoop.hdfs.server.protocol.FinalizeCommand@78a6195e
13/11/03 22:38:22 INFO util.GSet: Computing capacity for map BlockMap
13/11/03 22:38:22 INFO util.GSet: VM type = 64-bit
13/11/03 22:38:22 INFO util.GSet: 0.5% max memory = 888.9 MB
13/11/03 22:38:22 INFO util.GSet: capacity = 2^19 = 524288 entries
13/11/03 22:38:22 INFO datanode.BlockPool$SliceScanner: Periodic Block Verification
  
```

Resource Manager & Node Manager:

```

C:\Apache Hadoop Distribution - yarn resourcemanager
oop.yarn.server.api.ResourceManagerAdministrationProtocolPB to the server
13/11/03 22:48:14 INFO ipc.Server: IPC Server Responder: starting
13/11/03 22:48:14 INFO ipc.Server: IPC Server listener on 8033: starting
13/11/03 22:48:14 INFO util.RackResolver: Resolved ABHIJITG.██.com to /default-rack
13/11/03 22:48:14 INFO resourcemanager.ResourceTrackerService: NodeManager from node ABHIJITG.██.com(cmPort: 60092 httpPort: 8042) registered with capability: <memory:8192, vCores:8>, assigned nodeId ABHIJITG.██.com:60092
13/11/03 22:48:14 INFO rmnode.RMNodeImpl: ABHIJITG.██.com:60092 Node Transitioned from NEW to RUNNING
13/11/03 22:48:14 INFO capacity.CapacityScheduler: Added node ABHIJITG.██.com:60092 clusterResource: <memory:8192, vCores:8>

C:\Apache Hadoop Distribution - yarn nodemanager
13/11/03 22:48:13 INFO morthay.log: Started SelectChannelConnector00.0.0.0:8042
13/11/03 22:48:13 INFO webapp.WebApps: Web app /node started at 8042
13/11/03 22:48:14 INFO webapp.WebApps: Registered webapp guice modules
13/11/03 22:48:14 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8031
13/11/03 22:48:14 INFO security.NMContainerTokenSecretManager: Rolling master-key for container-tokens, got key with id 441918079
13/11/03 22:48:14 INFO security.NMTokenSecretManagerInNM: Rolling master-key for nm-tokens, got key with id :1221761938
13/11/03 22:48:14 INFO nodemanager.NodeStatusUpdaterImpl: Registered with ResourceManager as ABHIJITG.██.com:60092 with total resource of <memory:8192, vCores:8>
13/11/03 22:48:14 INFO nodemanager.NodeStatusUpdaterImpl: Notifying ContainerManager to unblock new container-requests
  
```

Run wordcount MapReduce job

Now we'll run **wordcount** MapReduce job available

in %HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar

Create a text file with some content. We'll pass this file as input to the **wordcount** MapReduce job for counting words.
C:\file1.txt

Install Hadoop

Run Hadoop Wordcount Mapreduce Example

Create a directory (say 'input') in HDFS to keep all the text files (say 'file1.txt') to be used for counting words.

C:\Users\abhijitg>cd c:\hadoop C:\hadoop>bin\hdfs dfs -mkdir input

Copy the text file(say 'file1.txt') from local disk to the newly created 'input' directory in HDFS.

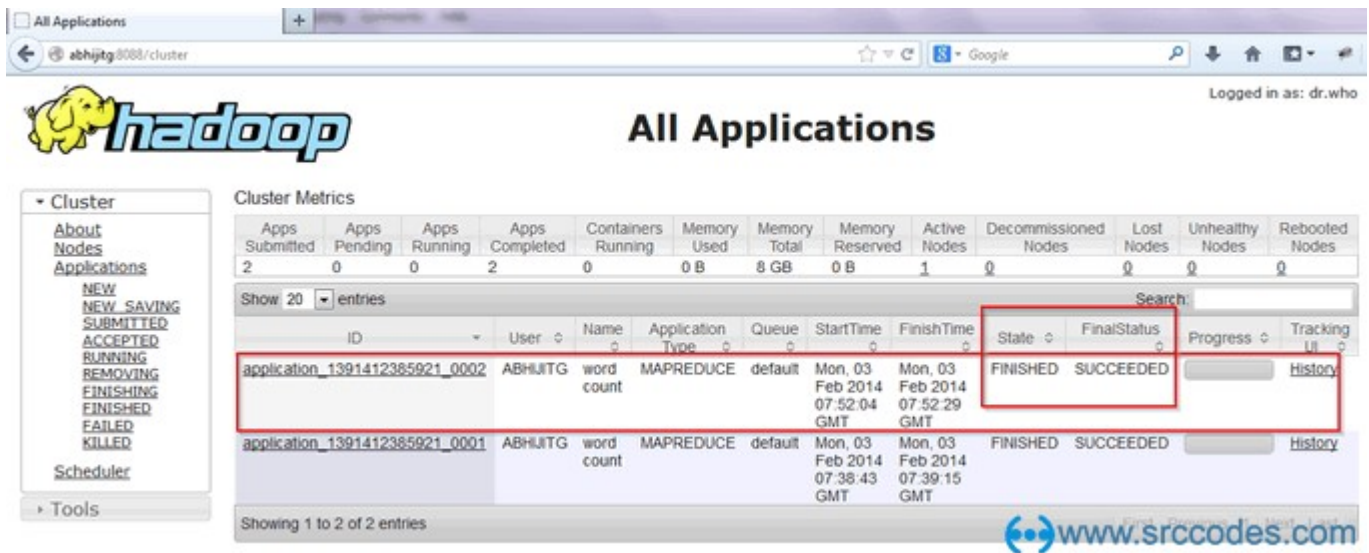
C:\hadoop>bin\hdfs dfs -copyFromLocal c:/file1.txt input

Check content of the copied file.

C:\hadoop>hdfs dfs -ls input

Found 1 items

-rw-r--r-- 1 ABHIJITG supergroup 55 2014-02-03 13:19 input/file1.txt



The screenshot shows the Hadoop web interface for 'All Applications'. The browser address bar shows 'abhi1tg8088/cluster'. The Hadoop logo is on the left. The main title is 'All Applications'. Below it, there's a 'Cluster Metrics' section with a table showing various metrics. A sidebar on the left contains navigation links like 'About', 'Nodes', 'Applications', and 'Scheduler'. The main content area displays a table of application entries. Two entries are visible, both with a 'FINISHED' state and 'SUCCEEDED' final status, highlighted by a red box. The bottom right corner has a watermark for 'www.srccodes.com'.

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	0	2	0	0 B	8 GB	0 B	1	0	0	0	0

Showing 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1391412385921_0002	ABHIJITG	word count	MAPREDUCE	default	Mon, 03 Feb 2014 07:52:04 GMT	Mon, 03 Feb 2014 07:52:29 GMT	FINISHED	SUCCEEDED		History
application_1391412385921_0001	ABHIJITG	word count	MAPREDUCE	default	Mon, 03 Feb 2014 07:38:43 GMT	Mon, 03 Feb 2014 07:39:15 GMT	FINISHED	SUCCEEDED		History

Showing 1 to 2 of 2 entries

www.srccodes.com

Result: We've installed Hadoop in stand-alone mode and verified it by running an example program it provided.

AIM: To Develop a MapReduce program to calculate the frequency of a given word in a given file

Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

Input

Set of data

Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN, BUS, buS, caR, CAR, car, BUS, TRAIN

Output

Convert into another set of data

(Key, Value)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1),

(TRAIN,1), (BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input Set of Tuples

(output of Map function)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1),

(TRAIN,1), (BUS,1),

(buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Output Converts into smaller set of tuples

(BUS,7), (CAR,7), (TRAIN,4)

Work Flow of Program

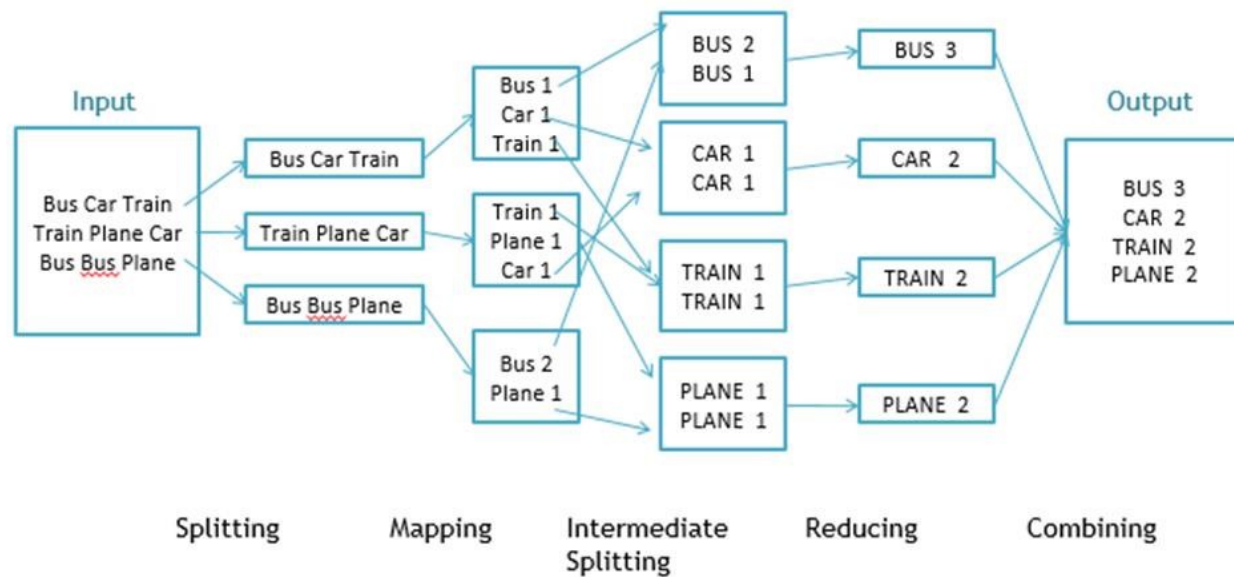


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').
2. **Mapping** – as explained above
3. **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on same cluster.
4. **Reduce** – it is nothing but mostly group by phase
5. **Combining** – The last phase where all the data (individual result set from each cluster) is combine together to form a Result

Now Let's See the Word Count Program in Java

Make sure that Hadoop is installed on your system with java idk

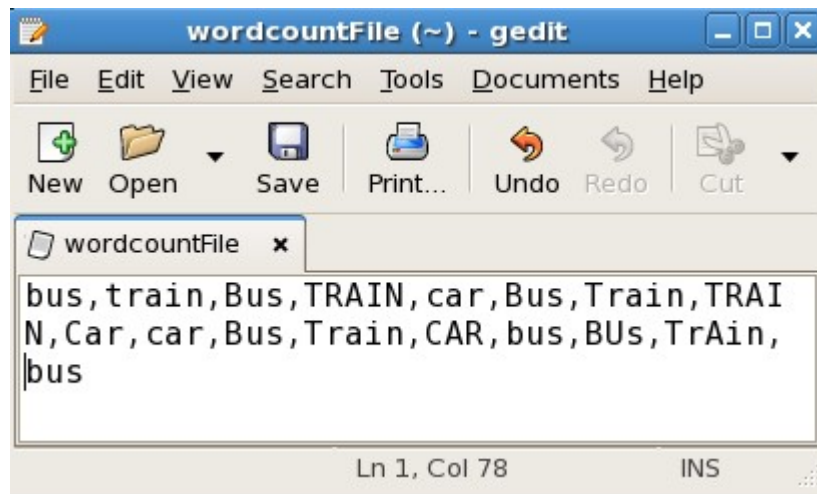
Steps to follow

Step 1. Open Eclipse > File > New > Java Project > (Name it – MRProgramsDemo) > Finish Step 2. Right Click > New > Package (Name it - PackageDemo) > Finish Step 3. Right Click on Package > New > Class (Name it - WordCount) Step 4. Add Following Reference Libraries – Right Click on Project > Build Path> Add External Archivals

- /usr/lib/hadoop-0.20/hadoop-core.jar
- Usr/lib/hadoop-0.20/lib/Commons-cli-1.2.jar

Program: Step 5. Type following Program :

```
package PackageDemo; 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.LongWritable; 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; import
org.apache.hadoop.util.GenericOptionsParser; public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class); FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException,
InterruptedException
{
String line = value.toString();
String[] words=line.split(","); for(String
word: words )
{
Text outputKey = new Text(word.toUpperCase().trim()); IntWritable
outputValue = new IntWritable(1); con.write(outputKey, outputFile);
}
} } }
```

To Move this into Hadoop directly, open the terminal and enter the following commands:
 [training@localhost ~]\$ hadoop fs -put wordcountFile wordCountFile

Run Jar file

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile
 PathToOutputDirectry)

[training@localhost ~]\$ Hadoop jar MRProgramsDemo.jar PackageDemo.WordCount
 wordCountFile MRDir1

Result: Open Result

```
[training@localhost ~]$ hadoop fs -ls MRDir1
Found 3 items
-rw-r--r-- 1 training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_SUCCESS
drwxr-xr-x - training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_logs
-rw-r--r-- 1 training supergroup
20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000
[training@localhost ~]$ hadoop fs -cat MRDir1/part-r-00000
BUS    7
CAR    4
TRAIN  6
```

AIM: To Develop a MapReduce program to find the grades of student's.

```
import java.util.Scanner; public
class JavaExample
{ public static void main(String args[])
{
/* This program assumes that the student has 6 subjects, thats why I
   have created the array of size 6. You can * change this as per the
                                   requirement. */

int marks[] = new int[6];
int i;
float total=0, avg;
Scanner scanner = new Scanner(System.in);
for(i=0; i<6; i++) {
System.out.print("Enter Marks of Subject"+(i+1)+":"); marks[i] =
scanner.nextInt(); total = total + marks[i];
}
scanner.close(); //Calculating
average here avg = total/6;
System.out.print("The student Grade is: ");
if(avg>=80)
{
System.out.print("A");
}
else if(avg>=60 && avg<80)
{
System.out.print("B");
}
else if(avg>=40 && avg<60)
{
System.out.print("C");
}
else {
System.out.print("D");
}
}
}
}
```

Expected Output:

Enter Marks of Subject1:40

Enter Marks of Subject2:80

Enter Marks of Subject3:80

Enter Marks of Subject4:40

Enter Marks of Subject5:60

Enter Marks of Subject6:60

The student Grade is: B

AIM: Develop a MapReduce program to analyze Titanic ship data and to find the average age of the people (both male and female) who died in the tragedy. How many persons are survived in each class.

The titanic data will be..

Column 1 : PassengerId	Column 2 : Survived (survived=0 & died=1)
Column 3 : Pclass	Column 4 : Name
Column 5 : Sex	Column 6 : Age
Column 7 : SibSp	Column 8 : Parch
Column 9 : Ticket	Column 10 : Fare
Column 11 : Cabin	Column 12 : Embarked

Description:

There have been huge disasters in the history of Map reduce, but the magnitude of the Titanic's disaster ranks as high as the depth it sank too. So much so that subsequent disasters have always been described as "titanic in proportion" – implying huge losses.

Anyone who has read about the Titanic, know that a perfect combination of natural events and human errors led to the sinking of the Titanic on its fateful maiden journey from Southampton to New York on April 14, 1912.

There have been several questions put forward to understand the cause/s of the tragedy – foremost among them is: What made it sink and even more intriguing How can a 46,000 ton ship sink to the depth of 13,000 feet in a matter of 3 hours? This is a mind boggling question indeed!

There have been as many inquiries as there have been questions raised and equally that many types of analysis methods applied to arrive at conclusions. But this blog is not about analyzing why or what made the Titanic sink – it is about analyzing the data that is present about the Titanic publicly. It actually uses Hadoop MapReduce to analyze and arrive at:

- The average age of the people (both male and female) who died in the tragedy using Hadoop MapReduce.
- How many persons survived – traveling class wise.

This blog is about analyzing the data of Titanic. This total analysis is performed in Hadoop MapReduce.

This Titanic data is publically available and the Titanic data set is described below under the heading Data Set Description.

Using that dataset we will perform some Analysis and will draw out some insights like finding the average age of male and females died in Titanic, Number of males and females died in each

compartment.**DATA SET DESCRIPTION**

Column 1 : PassengerI

Column 2 : Survived (survived=0 & died=1) Column 3 : Pclass

Column 4 : Name

Column 5 : Sex

Column 6 : Age

Column 7 : SibSp

Column 8 : Parch

Column 9 : Ticket

Column 10 : Fare

Column 11 : Cabin

Column 12 : Embarked

Mapper code:

```
public class Average_age {  
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {  
  
        private Text gender = new Text();  
  
        private IntWritable age = new IntWritable();  
  
        public void map(LongWritable key, Text value, Context context ) throws IOException,  
            InterruptedException {  
  
            String line = value.toString();  
  
            String str[]=line.split(",");  
  
            if(str.length>6){  
  
                gender.set(str[4]);  
  
                if((str[1].equals("0")) ){  
  
                    if(str[5].matches("\\d+")){  
  
                        int i=Integer.parseInt(str[5]);  
  
                        age.set(i);  
                    }  
                } }  
            context.write(gender, age)
```

```
}
}
```

Reducer Code:

```
public static class Reduce extends Reducer<Text,IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException { int sum = 0; int l=0; for (IntWritable val : values)
{ l+=1; sum += val.get();
} sum=sum/l; context.write(key, new
IntWritable(sum));
}
}
```

Configuration Code: job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

<https://github.com/kiran0541/Map-Reduce/blob/master/Average%20age%20of%20male%20and%20female%20people%20died%20in%20titanic>

Way to to execute the Jar file to get the result of the first problem statement: *hadoop jar average.jar /TitanicData.txt /avg_out*

Here ‘hadoop’ specifies we are running a Hadoop command and jar specifies which type of application we are running and average.jar is the jar file which we have created which consists the above source code and the path of the Input file name in our case it is TitanicData.txt and the output file where to store the output here we have given it as avg out.

Way to view the output:

hadoop dfs -cat /avg_out/part-r-000000

Here ‘hadoop’ specifies that we are running a Hadoop command and ‘dfs’ specifies that we are performing an operation related to Hadoop Distributed File System and ‘- cat’ is used to view the contents of a file and ‘avg_out/part-r-000000’ is the file where the output is stored. Part file is created by default by the TextInputFormat class of Hadoop.

Aim : Mongo DB: Installation and Creation of database and Collection CRUD Document: Insert, Query, Update and Delete Document.

Theory :

To get started with MongoDB, you have to install it in your system. You need to find and download the latest version of MongoDB, which will be compatible with your computer system. You can use this (<http://www.mongodb.org/downloads>) link and follow the instruction to install MongoDB in your PC.

The process of setting up MongoDB in different operating systems is also different, here various installation steps have been mentioned and according to your convenience, you can select it and follow it.

1 Install Mongo DB in Windows

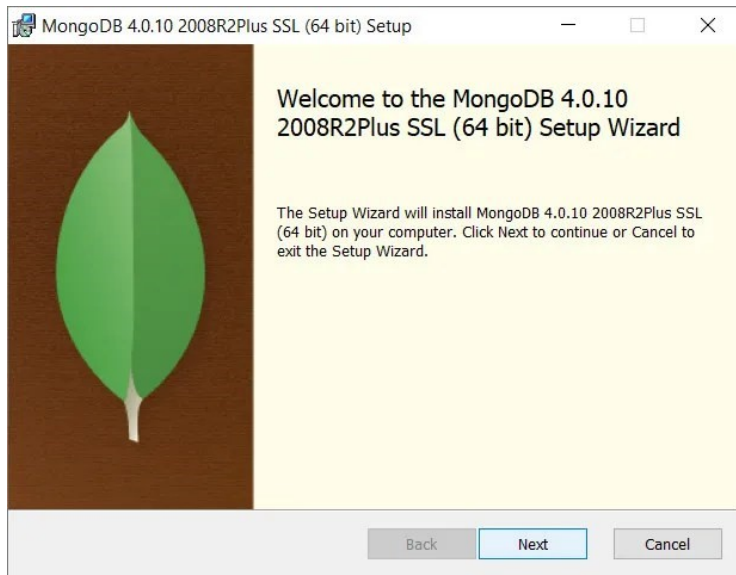
The website of MongoDB provides all the installation instructions, and MongoDB is supported by Windows, Linux as well as Mac OS.

It is to be noted that, MongoDB will not run in Windows XP; so you need to install higher versions of windows to use this database.

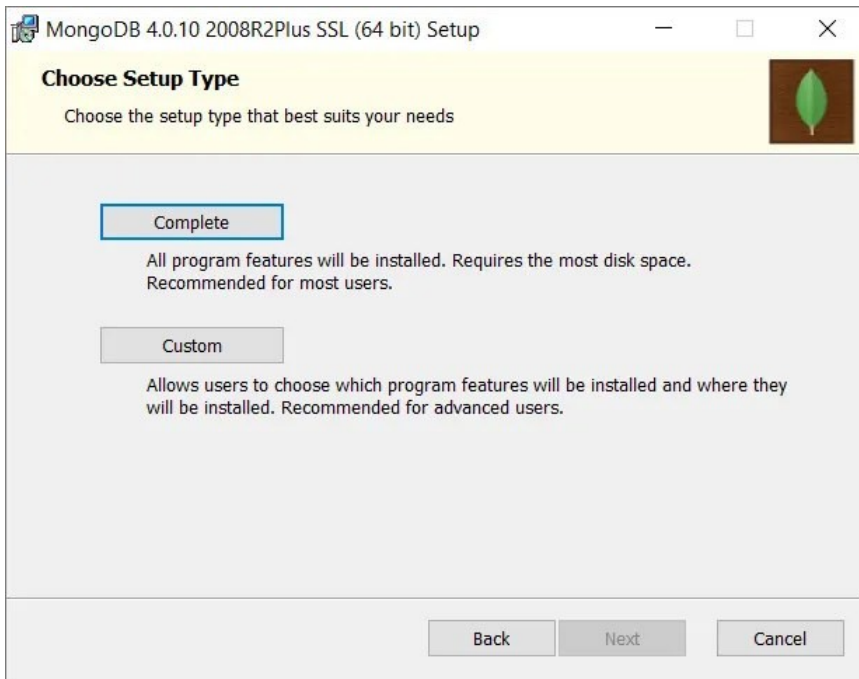
Once you visit the link (<http://www.mongodb.org/downloads>), click the download button.

1. Once the download is complete, double click this setup file to install it. Follow the steps:

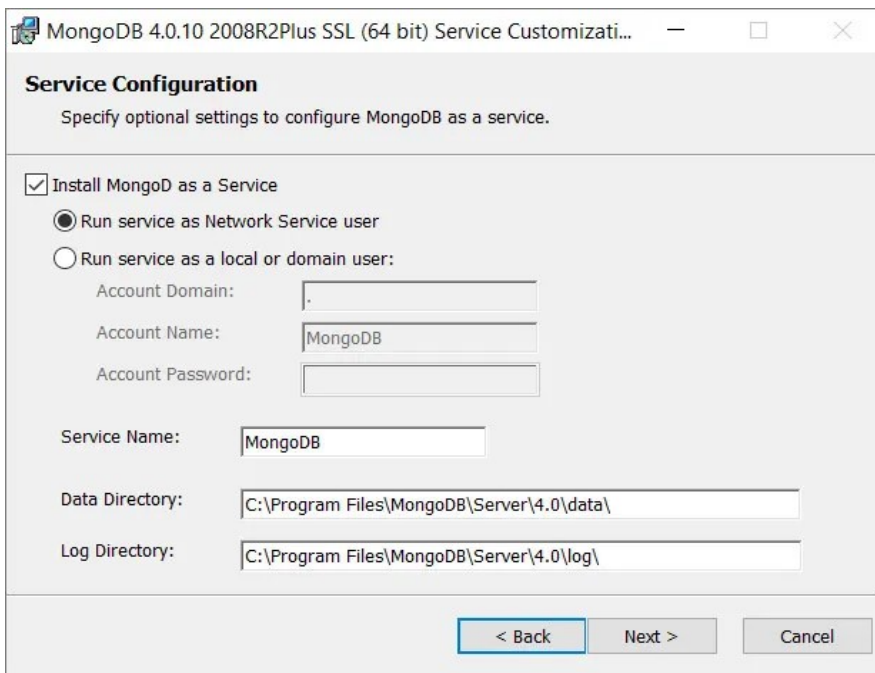
2



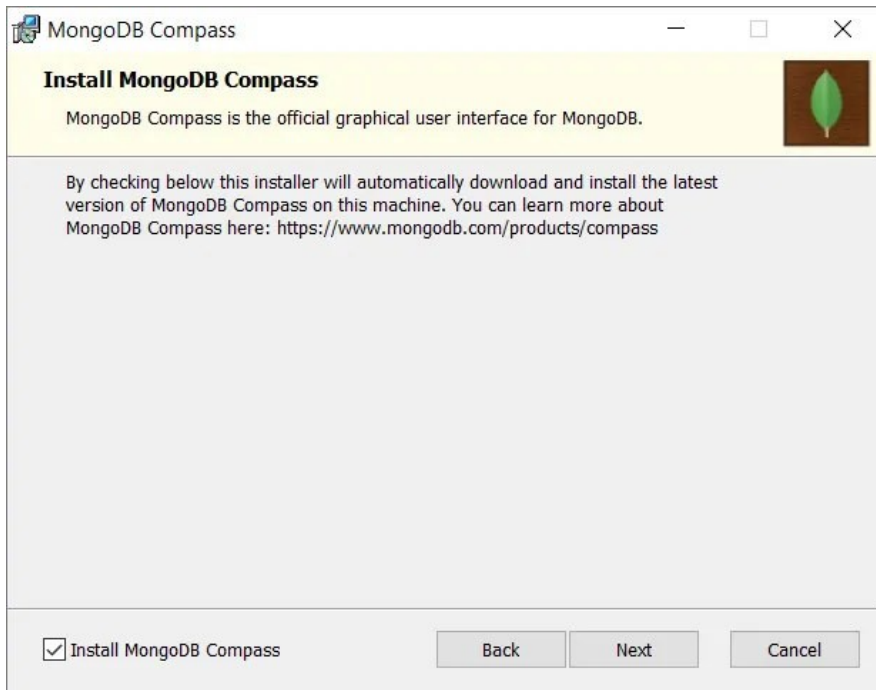
3. Now, choose Complete to install MongoDB completely.



4. Then, select the radio button "Run services as Network service user."



The setup system will also prompt you to install MongoDB Compass, which is MongoDB official graphical user interface (GUI). You can tick the checkbox to install that as well.



Once the installation is done completely, you need to start MongoDB and to do so follow the process:

1. Open Command Prompt.
2. Type: C:\Program Files\MongoDB\Server\4.0\bin
3. Now type the command simply: **mongod** to run the server.

In this way, you can start your MongoDB database. Now, for running MongoDB primary client system, you have to use the command:

```
C:\Program Files\MongoDB\Server\4.0\bin>mongo.exe
```

CRUD DOCUMENT

As we know, we can use MongoDB for a variety of purposes such as building an application (including web and mobile), data analysis, or as an administrator of a MongoDB database. In all of these cases, we must interact with the MongoDB server to perform specific operations such as entering new data into the application, updating data in the application, deleting data from the application, and reading the data of the application.

MongoDB provides a set of simple yet fundamental operations known as CRUD operations that will allow you to quickly interact with the MongoDB server.



1. **Create Operations** — these operations are used to insert or add new documents to the collection. If a collection does not exist, a new collection will be created in the database. MongoDB provides the following methods for performing and creating operations:

Method	Description
db.collection.insertOne()	It is used to insert a single document in the collection.
db.collection.insertMany()	It is used to insert multiple documents in the collection.

insertOne()

As the namesake, insertOne() allows you to insert one document into the collection. For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB. We then provide the

information we want to insert in the form of key-value pairs, establishing the Schema.

Example:

```
db.RecordsDB.insertOne({ name: "Marsh", age: "6 years", species: "Dog", ownerAddress:
"380 W. Fir Ave", chipped: true
})
```

If the create operation is successful, a new document is created. The function will return an object where “acknowledged” is “true” and “insertID” is the newly created “ObjectId.”

```
> db.RecordsDB.insertOne({
... name: "Marsh",
... age: "6 years",
... species: "Dog",
... ownerAddress: "380 W. Fir Ave",
... chipped: true
... })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5fd989674e6b9ceb8665c57d")
}
```

insertMany()

It's possible to insert multiple items at one time by calling the insertMany() method on the desired collection. In this case, we pass multiple items into our chosen collection (RecordsDB) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

Example:

```
db.RecordsDB.insertMany([ { name: "Marsh", age: "6 years", species: "Dog",
ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years",
species: "Cat", ownerAddress: "521 E. Cortland", chipped: true} ])
```

```
db.RecordsDB.insertMany([ { name: "Marsh", age: "6 years", species:
"Dog",
ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years", species: "Cat",
ownerAddress: "521 E. Cortland", chipped: true} ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5fd98ea9ce6e8850d88270b4"),
    ObjectId("5fd98ea9ce6e8850d88270b5")
  ]
}
```

2. Read Operations

You can give specific query filters and criteria to the read operations to indicate the documents you desire. More information on the possible query filters can be found in the MongoDB manual. Query modifiers can also be used to vary the number of results returned.

MongoDB offers two ways to read documents from a collection:

- [db.collection.find\(\)](#) • [db.collection.findOne\(\)](#) **find()**

In order to get all the documents from a collection, we can simply use the `find()` method on our chosen collection. Executing just the `find()` method with no arguments will return all records currently in the collection.

[db.RecordsDB.find](#) [findOne\(\)](#)

In order to get one document that satisfies the search criteria, we can simply use the `findOne()` method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. If no documents satisfy the search criteria, the function returns null. The function takes the following form of syntax.


```
db.{collection}.findOne({query}, {projection})
```

3 Update Operations

Update operations, like create operations, work on a single collection and are atomic at the document level. Filters and criteria are used to choose the documents to be updated during an update procedure.

You should be cautious while altering documents since alterations are permanent and cannot be reversed. This also applies to remove operations.

There are three techniques for updating documents in MongoDB CRUD:

- `db.collection.updateOne()`
- `db.collection.updateMany()` ● `db.collection.replaceOne()` **updateOne()**

With an update procedure, we may edit a single document and update an existing record. To do this, we use the `updateOne()` function on a specified collection, in this case "RecordsDB." To update a document, we pass two arguments to the method: an update filter and an update action.

The update filter specifies which items should be updated, and the update action specifies how those items should be updated. We start with the update filter. Then we utilise the "\$set" key and supply the values for the fields we wish to change. This function updates the first record that matches the specified filter.

updateMany()

`updateMany()` allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.

replaceOne()

The `replaceOne()` method is used to replace a single document in the specified collection. `replaceOne()` replaces the entire document, meaning fields in the old document not contained in the new will be lost.

4 Delete Operations

Delete operations, like update and create operations, work on a single collection. For a single document, delete actions are similarly atomic. You can provide delete actions with filters and criteria to indicate which documents from a collection you want to delete. The filter options use

the same syntax as the read operations.

MongoDB provides two ways for removing records from a collection:

- `db.collection.deleteOne()` • `db.collection.deleteMany()` **deleteOne()**

`deleteOne()` is used to remove a document from a specified collection on the MongoDB server. A filter criterion is used to specify the item to delete.

It deletes the first record that matches the provided filter.

`deleteMany()`

`deleteMany()` is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in `deleteOne()`

3.6 *Let us sum up*

- MongoDB is an open-source database that uses a document-oriented data model and a non-structured query language
- It is one of the most powerful NoSQL systems and databases around, today.
- The data model that MongoDB follows is a highly elastic one that lets you combine and store data of multivariate types without having to compromise on the powerful indexing options, data access, and validation rules.
- A group of database documents can be called a collection. The RDBMS equivalent to a collection is a table. The entire collection exists within a single database. There are no schemas when it comes

to collections. Inside the collection, various documents can have varied fields, but mostly the documents within a collection are meant for the same purpose or for serving the same end goal.

- A set of key-value pairs can be designated as a document. Documents are associated with dynamic schemas. The benefit of having dynamic schemas is that a document in a single collection does not have to possess the same structure or fields. Also, the common fields in a collection document can have varied types of data.