Practical – 1

Big Data in Education

Introduction

Big data has various applications in different domains. Some of the applications of big data analysis include machine learning, detecting hidden patterns, unknown correlations, market trends, and customer preferences. Big data analytics is an analysis technique that examines enormous data sets. Real-time big data applications are used in various domains such as healthcare, finance, retail, and transportation. Some examples of big data use cases in businesses include fraud detection, customer segmentation, and predictive maintenance.

Real-time big data applications are used in various domains such as healthcare, finance, retail, and transportation. In healthcare, realtime big data applications can be used for patient monitoring, disease surveillance, and drug discovery. In finance, big data can be used for fraud detection, algorithmic trading, and customer segmentation. In retail, big data can be used for inventory management, customer behavior analysis, and personalized marketing. In transportation, big data can be used for route optimization, predictive maintenance, and real-time traffic analysis. The use of big data in these domains can lead to improved efficiency, cost savings, and better decision-making.

The phrase "beyond the scope of this paper" is commonly used in research to indicate that the topic being discussed is not the main focus of the paper and that the reader should refer to other sources for more information. This phrase is used to avoid going off-topic and to direct the reader towards useful literature about the topic. It is important to use this phrase appropriately and to provide clear and concise explanations of the main topic being discussed in the paper.

When turning down a task that is beyond the scope of one's expertise, it is important to do so politely and professionally. One can explain the reasons why the task is beyond their scope and suggest alternative solutions such as delegating the task to someone else or providing resources for the task. It is important to be honest and transparent about one's limitations and to communicate clearly with the person requesting the task.

Big data has revolutionized various sectors, and education is no exception. The education sector generates a significant amount of digital data from various sources, including students, teachers, institutions, and digital tools. The data generated can be used to improve the education sector's overall efficiency, effectiveness, and accountability. In this report, we will explore the different types of digital big data generated in the education sector and how they are used.

Different Types of Data Generated in Education Domain

Student Data

Student data is one of the primary types of digital data generated in the education sector. It includes information such as student demographics, attendance records, grades, test scores, and disciplinary records. This data is often stored in Student Information Systems (SIS), Learning Management Systems (LMS), and other digital platforms. Student data can be used to track student progress, identify areas of improvement, and develop personalized learning plans. It can also be used to evaluate student performance and inform decisions related to educational policies.

Teaching Data

Teaching data is another critical type of digital data generated in the education sector. It includes information about teachers' instructional strategies, classroom management techniques, and assessment methods. This data is often collected through teacher evaluations, observation tools, and surveys. Teaching data can be used to improve teacher performance, inform professional development opportunities, and develop effective instructional practices. It can also be used to identify areas of improvement in teaching quality and inform policies related to teacher training and support.

Institutional Data

Institutional data includes information about educational institutions, including enrollment numbers, funding sources, and staffing data. It is often collected through institutional research and assessment offices and can be used to inform decisions related to educational policy, strategic planning, and resource allocation. Institutional data can also be used to evaluate institutional performance and inform accreditation processes.

Learning Analytics Data

Learning analytics data is a rapidly growing type of digital data generated in the education sector. It includes information about student interactions with digital learning platforms, such as LMS, online assessments, and educational apps. Learning analytics data can be used to develop personalized learning plans, identify areas of improvement in instructional design, and evaluate the effectiveness of digital tools. It can also be used to improve student engagement and retention rates.

Social Media Data

Social media data is another type of digital data generated in the education sector. It includes information about student and faculty interactions on social media platforms such as Facebook, Twitter, and Instagram. Social media data can be used to understand student and faculty perspectives on educational issues, inform decisions related to social media policies, and develop effective communication strategies.

Open Educational Resources Data

Open Educational Resources (OER) data includes information about the use of OER, including textbooks, videos, and other digital resources, in the education sector. OER data can be used to evaluate the effectiveness of open education initiatives, identify areas of improvement in instructional design, and inform decisions related to resource allocation.

Assessment Data

Assessment data includes information about student performance on standardized tests, local assessments, and formative assessments.

This data is often used to evaluate student achievement, identify areas of improvement in instructional design, and inform decisions related to educational policy. Assessment data can also be used to evaluate the effectiveness of educational interventions and identify students who may need additional support.

The education sector generates a significant amount of digital data from various sources. This data can be used to improve the education sector's overall efficiency, effectiveness, and accountability. The types of digital data generated in the education sector include student data, teaching data, institutional data, learning analytics data, social media data, open educational resources data, and assessment data. Each type of data has its unique uses and can be used to inform decisions related to educational policy, instructional design, and resource allocation. As the use of digital

Practical – 2

Implement the following machine learning technique Linear Regression Logistic Regression KMeans Clustering

Loading the Iris Dataset

```
[ ] import pandas as pd
    from sklearn.datasets import load_iris
    iris=load_iris()
```

```
[ ] iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
[ ] Y = iris.target
```

```
X = iris.data
```

```
[ ] # df = pd.DataFrame(iris.data)
    # df.columns = iris['feature_names']

    # df.head()
```

```
[ ]
```

Linear Rgression

Logistic Regression

KMeans Clustering

```
[ ] from sklearn.linear_model import LinearRegression
    from sklearn.linear_model import LogisticRegression
    from sklearn.cluster import KMeans
    from sklearn.model_selection import train_test_split
```

```
linearRegression = LinearRegression()
logistic = LogisticRegression(solver='lbfgs', max_iter=1000)
kmeans = KMeans(n_clusters=3)
```

Splitting the Data for Training and Testing

```
[ ] x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=0)
```

```
[ ] x_train.shape
```

```
(112, 4)
```

```
y_train.shape
```

```
(112,)
```

Linear Regression

```
[ ] linearRegression.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Logistic Regression

```
[ ] logistic.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

KMeans Clustering

```
kmeans.fit(X)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
  warnings.warn(
```

```
▾ KMeans
KMeans(n_clusters=3)
```

Outputs

```
[ ] ylinear = linearRegression.predict(x_test)
```

```
[ ] ylogistic = logistic.predict(x_test)
```

```
[ ] ykmeans = kmeans.fit_predict(X)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The defa
  warnings.warn(
```

```
ylinear
```

```
array([ 2.07872867,  0.9662282 , -0.16117412,  1.82229476, -0.03749929,
        2.28704244, -0.03604989,  1.30986735,  1.27147131,  1.10781204,
        1.59744796,  1.299921  ,  1.23731195,  1.32145191,  1.34954356,
       -0.11133487,  1.36886386,  1.2542803 ,  0.03401222, -0.05014733,
        1.82644819,  1.42764369,  0.09995305,  0.04048737,  1.59299693,
       -0.1147503 ,  0.15857194,  1.17003517,  0.9301028 ,  0.10397109,
        1.74160045,  1.45830398, -0.07070034,  1.62994357,  2.00546549,
        1.27901229, -0.04419114,  1.59151965])
```

```
[ ] import numpy as np
```

```
[ ] ylinear = abs(np.round(ylinear))
```

```
[ ] ylinear
```

```
array([2., 1., 0., 2., 0., 2., 0., 1., 1., 1., 2., 1., 1., 1., 1., 0., 1.,
       1., 0., 0., 2., 1., 0., 0., 2., 0., 0., 1., 1., 0., 2., 1., 0., 2.,
       2., 1., 0., 2.])
```

```
ylogistic
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])
```

```
[ ] ykmeans
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

Evaluation

```
[ ] from sklearn.metrics import accuracy_score
```

Linear Regression Accuracy

```
[ ] aclinear = accuracy_score(ylinear, y_test)
```

```
[ ] aclinear
```

```
0.9736842105263158
```

Logistic Regression Accuracy

```
[ ] aclogistic = accuracy_score(ylogistic, y_test)
```

```
[ ] aclogistic
```

```
0.9736842105263158
```

KMeans Clustering Accuracy

```
ackmeans = accuracy_score(ykmeans, Y)
```

```
[ ] ackmeans
```

```
0.24
```

Practical – 3

Setup Single-node hadoop cluster and apply HDFS command on single node hadoop cluster

To set up a single-node Hadoop cluster, you will need to follow these general steps:

Download and install Hadoop: Download the latest stable version of Hadoop from the official website and install it on your machine.

Apache Hadoop Download Documentation Community Development Help Apache Software Foundation

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
3.3.4	2022 Aug 8	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature)	Announcement
3.2.4	2022 Jul 22	source (checksum signature)	binary (checksum signature)	Announcement
2.10.2	2022 May 31	source (checksum signature)	binary (checksum signature)	Announcement

Configure Hadoop: Once the installation is complete, you need to configure Hadoop to work as a single-node cluster. This involves editing configuration files such as core-site.xml, hdfs-site.xml, and mapred-site.xml.

Core-site.xml Configuration

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Hdfs-site.xml

```xml
<configuration>

<!-- Site specific YARN configuration properties -->
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>

</configuration>
```

Yarn-site.xml

```xml
<configuration>
<property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:///D:/hadoop-3.3.4/data/namenode</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>/D:/hadoop-3.3.4/data/datanode</value>
</property>
</configuration>
```

Mapred-site.xml configuration

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

Hadoop-env.cmd configuration setting the path

```
@rem The java implementation to use.  Required.
set JAVA_HOME=%JAVA_HOME%
set JAVA_HOME=C:\Progra~1\OpenJDK\openjdk-11.0.14.1_1
```

Format the Hadoop file system: Before you can use Hadoop, you need to format the Hadoop Distributed File System (HDFS) using the command: hdfs namenode -format

```
D:\hadoop-3.3.4>hdfs namenode -format
2023-03-21 02:14:48,179 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = LAPTOP-DTDQGQDQ/192.168.1.4
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.4
STARTUP_MSG:   classpath = D:\hadoop-3.3.4\etc\hadoop;D:\hadoop-3.3.4\share\hadoop\
annotations-1.17.jar;D:\hadoop-3.3.4\share\hadoop\common\lib\asm-5.0.4.jar;D:\hadoop
3.3.4\share\hadoop\common\lib\checker-qual-2.5.2.jar;D:\hadoop-3.3.4\share\hadoop\c
```

Start the Hadoop cluster: To start the Hadoop cluster, run the command start-all.sh from the bin directory of your Hadoop installation.

Verify the Hadoop installation: Verify that Hadoop is installed correctly and the cluster is working by running the jps command to see a list of running processes.

```
D:\hadoop-3.3.4>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 /D:/hadoop-3.3.4/share/hadoop/common/hadoop-common-3.3.4.jar
```

```
D:\hadoop-3.3.4>java -version
java version "17" 2021-09-14 LTS
Java(TM) SE Runtime Environment (build 17+35-LTS-2724)
Java HotSpot(TM) 64-Bit Server VM (build 17+35-LTS-2724, mixed mode, sharing)
```

```
D:\hadoop-3.3.4>jps
1056 Jps
10004
11108 DataNode
12564 NodeManager
19876 ResourceManager
22040 NameNode
```

Once your Hadoop cluster is set up, you can start using HDFS commands. Here are some basic HDFS commands you can use:

hdfs dfs -ls /: List the files and directories in the root directory of HDFS.

```
D:\hadoop-3.3.4>hdfs dfs -ls /
Found 4 items
drwxr-xr-x   - gauta supergroup          0 2023-03-21 01:55 /input
drwxr-xr-x   - gauta supergroup          0 2023-03-21 01:43 /input_dir
drwxr-xr-x   - gauta supergroup          0 2023-03-21 01:59 /out
drwx------   - gauta supergroup          0 2023-03-21 01:58 /tmp
```

hdfs dfs -mkdir /test: Create a new directory called "test" in the root directory of HDFS.

```
D:\hadoop-3.3.4>hadoop fs -mkdir /input

D:\hadoop-3.3.4>hadoop fs -put D:/word.txt /input
```

hdfs dfs -put localfile.txt /test: Upload a local file called "localfile.txt" to the "test" directory in HDFS.

hdfs dfs -cat /test/localfile.txt: Display the contents of the file "localfile.txt" in the "test" directory.

```
D:\hadoop-3.3.4>hadoop fs -cat /out/*
hello   2
hi      2
```

hdfs dfs -rm /test/localfile.txt: Remove the file "localfile.txt" from the "test" directory.

```
D:\hadoop-3.3.4>hadoop fs -ls /input/
Found 1 items
-rw-r--r--   1 gauta supergroup         22 2023-03-21 01:55 /input/word.txt

D:\hadoop-3.3.4>hadoop dfs -cat /input/word.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
hello
hi
hello
hi
```

Practical – 4

Map Reduce Algorithm

To design a MapReduce algorithm that takes a large file of integers and generates the largest integer and the average of all integers, we can follow these steps:

Map Function: The map function will read each integer from the input file and emit a key-value pair where the key is a constant and the value is the integer itself. This will distribute the integers evenly among the reducers.

Reduce Function: The reduce function will receive a set of integers from the map function, and it will calculate the largest integer and the sum of all integers. It will also count the number of integers received. It will then emit two key-value pairs: one for the largest integer and another for the sum and count of all integers.

Combiner Function: We can use a combiner function that performs a local reduce operation on the key-value pairs emitted by the map function. This will reduce the amount of data that needs to be shuffled across the network.

Pseudo Code

```
Map (key, value):
   // key is ignored, value is an integer from the input file     emit("data", value)

Combiner (key, values):     largest =
max(values)
   sum = reduce(values, (x,y) -> x+y)     count =
size(values)
   emit("partial_results", (largest, sum, count))
Reduce (key, values):
   largest = max(values)     total_sum = 0
total_count = 0     for value in values:
sum, count = value        total_sum += sum
total_count += count     average =
total_sum / total_count     emit("result",
(largest, average))
```

Mapper Class

In this implementation, the mapper class is responsible for processing the input data and emitting intermediate key-value pairs. The mapper class is a Java class that extends the Mapper class of the Hadoop MapReduce framework and overrides the map() method. The map() method takes three arguments: a LongWritable key, a Text value, and a Context object.

The LongWritable key represents the byte offset of the current line in the input file. The Text value represents the content of the current line in the input file. The Context object is used to write the intermediate key-value pairs to the output.

The map() method first converts the Text value to an integer using the parseInt() method. It then creates a new IntWritable object with the integer value and sets this as the value of the output key-value pair. The output key is a constant value "data" which is represented by the static final Text object DATA_KEY.

Finally, the Context object's write() method is called with the DATA_KEY as the output key and the IntWritable object as the output value. This writes the intermediate key-value pair to the Hadoop MapReduce framework's output.

Overall, the mapper class processes the input data by extracting the integer value from each line and emitting a constant key-value pair where the key is "data" and the value is the integer value. This intermediate key-value pair is then passed to the combiner or reducer class for further processing.

```
public class Mapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private static final Text DATA_KEY = new Text("data");       private final
IntWritable valueWritable = new IntWritable();
   @Override
   public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {       int intValue =
Integer.parseInt(value.toString());       valueWritable.set(intValue);
      context.write(DATA_KEY, valueWritable);
   }
}
```

Combiner Class

The Combiner class is a type of reducer that is used to perform local aggregation of data before it is sent to the reducers. The purpose of the Combiner is to reduce the amount of data that needs to be transmitted between the Mapper and the Reducer, which can significantly improve the performance of the MapReduce job.

In this specific example, the Combiner class takes in key-value pairs, where the key is a Text object and the value is an IntWritable object. The output of the Combiner is also a key-value pair, where the key is a Text object and the value is a TupleWritable object.

The Text key is set to a constant value "partial_results". This key is used to indicate that the output of the Combiner contains partial results, rather than the final results of the MapReduce job.

The TupleWritable value contains three elements: the largest integer, the sum of all integers, and the count of integers. These values are calculated by iterating over the Iterable<IntWritable> values input to the reduce method. For each value, the Combiner updates the largest integer if necessary, adds the value to the sum, and increments the count.

Finally, the Combiner writes the output key-value pair to the Context object using the context.write() method. The output key is set to "partial_results", and the value is the TupleWritable containing the partial results.

```
public class Combiner extends Reducer<Text, IntWritable, Text,
TupleWritable> {
   private static final Text PARTIAL_RESULTS_KEY = new
Text("partial_results");
   private final TupleWritable tupleWritable = new TupleWritable();

   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {        int largest =
Integer.MIN_VALUE;        int sum = 0;
```

```
    int count = 0;
    for (IntWritable value : values) {              int
intValue = value.get();
        largest = Math.max(largest, intValue);
sum += intValue;           count++;
    }
    tupleWritable.set(0, new IntWritable(largest));
tupleWritable.set(1, new IntWritable(sum));        tupleWritable.set(2,
new IntWritable(count));
    context.write(PARTIAL_RESULTS_KEY, tupleWritable);
  }
}
```

Reducer Class

The Reducer class is responsible for taking the output produced by the Combiner class and computing the final result. In this case, the Reducer takes the intermediate results for each key and calculates the largest integer and the average of all integers.

The class extends the Reducer class and uses the following key-value pairs: (Text, TupleWritable) -> (Text, DoubleWritable).

The RESULT_KEY is a Text object that identifies the final result. The resultWritable object is a DoubleWritable object used to write the final result to the output.

In the reduce() method, the values Iterable contains all the intermediate values associated with a particular key. The intermediate values are in the form of TupleWritable objects containing the largest integer, the sum of all integers, and the count of all integers.

The method iterates through all the TupleWritable objects and extracts the partial largest integer, partial sum, and partial count values. It then updates the final largest integer, sum, and count values by adding the partial values from each TupleWritable.

Finally, the method calculates the average of all integers by dividing the sum by the count and sets it to the resultWritable object. It writes the average value to the output by using the RESULT_KEY. It then sets the largest integer to the resultWritable object and writes it to the output again using the RESULT_KEY.

In summary, the Reducer class aggregates the intermediate results and computes the final results by calculating the largest integer and average of all integers. It then writes the final results to the output

```
public class Reducer extends Reducer<Text, TupleWritable, Text,
DoubleWritable> {
   private static final Text RESULT_KEY = new Text("result");     private final
DoubleWritable resultWritable = new
DoubleWritable();

   @Override
   public void reduce(Text key, Iterable<TupleWritable> values, Context context)
throws IOException, InterruptedException {                         int largest =
Integer.MIN_VALUE;        int sum = 0;        int count = 0;        for (TupleWritable
value : values) {        int partialLargest = ((IntWritable) value.get(0)).get();        int
partialSum = ((IntWritable) value.get(1)).get();                    int partialCount =
((IntWritable) value.get(2)).get();        largest = Math.max(largest, partialLargest);
sum += partialSum;
        count += partialCount;
        }
        double average = (double) sum / count;
        resultWritable.set(average);
        context.write(RESULT_KEY, resultWritable);
resultWritable.set((double) largest);        context.write(RESULT_KEY,
resultWritable);
   }
```

Main Function

Configure the Job and run it.

```
public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();     Job job =
Job.getInstance(conf, "Integers");
job.setJarByClass(Integers.class);
job.setMapperClass(IntegersMapper.class);
job.setCombinerClass(IntegersCombiner.class);
job.setReducerClass(IntegersReducer.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); }
```

Practical – 5

Start all Hadoop services

```
C:\Windows\system32>d:

D:\>cd hadoop-3.3.4/sbin

D:\hadoop-3.3.4\sbin>start-dfs

D:\hadoop-3.3.4\sbin>start-yarn
starting yarn daemons
```

Checking that all the services our namenode and datanode is running perfectly or not.

```
D:\hadoop-3.3.4>jps
10004
11108 DataNode
12564 NodeManager
19876 ResourceManager
22980 Jps
22040 NameNode
```

Making the directory for our input File

Putting the file in the input directory of which we have to calculate the count of words into that file.

```
D:\hadoop-3.3.4>hadoop fs -mkdir /input

D:\hadoop-3.3.4>hadoop fs -put D:/word.txt /input
```

Places the input file in the hdfs environment and printing that input file content into console.

```
D:\hadoop-3.3.4>hadoop fs -ls /input/
Found 1 items
-rw-r--r--   1 gauta supergroup         22 2023-03-21 01:55 /input/word.txt

D:\hadoop-3.3.4>hadoop dfs -cat /input/word.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
hello
hi
hello
hi
```

Mapper Class

```
public class WordCountMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {

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

    public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);         while
(tokenizer.hasMoreTokens()) {            word.set(tokenizer.nextToken());
context.write(word, ONE);
        }
    }
}
```

Reducer Class

```
public class WordCountReducer 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 value : values) {             sum
+= value.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Configure the Mapper and Reducer Class

```
public class WordCount {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
job.setMapperClass(WordCountMapper.class);
```

```java
        job.setCombinerClass(WordCountReducer.class);
job.setReducerClass(WordCountReducer.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);
    }
}
```

We have already the Jar file for word count in hadoop example

So we are going to use them directly to count the number of words into our input file.

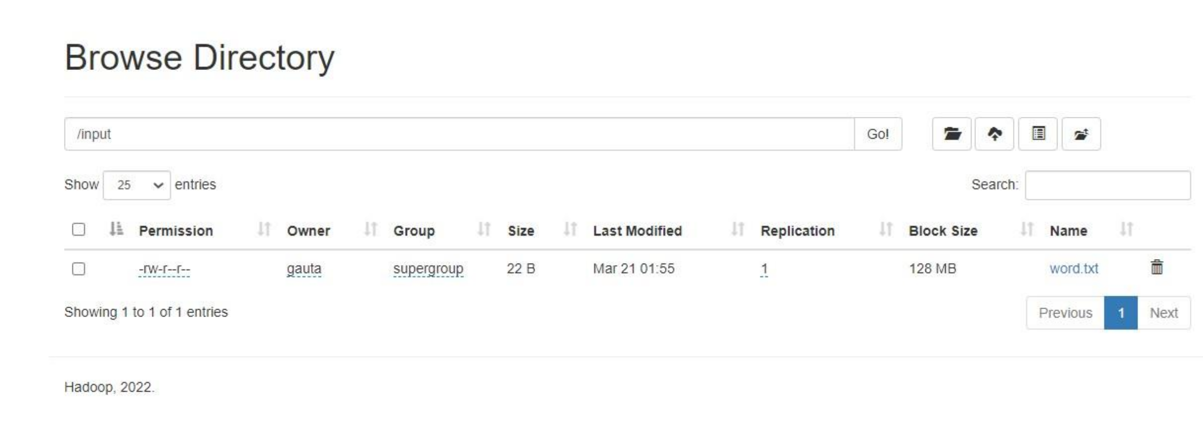
```
D:\hadoop-3.3.4>hadoop jar D:/hadoop-3.3.4/share\hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar wordcount /input /out
2023-03-21 01:58:58,597 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-03-21 01:59:00,092 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/gauta/.staging/job_1679343064205_0001
2023-03-21 01:59:00,450 INFO input.FileInputFormat: Total input files to process : 1
2023-03-21 01:59:00,584 INFO mapreduce.JobSubmitter: number of splits:1
2023-03-21 01:59:00,872 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1679343064205_0001
2023-03-21 01:59:00,873 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-21 01:59:01,125 INFO conf.Configuration: resource-types.xml not found
2023-03-21 01:59:01,126 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-03-21 01:59:01,871 INFO impl.YarnClientImpl: Submitted application application_1679343064205_0001
2023-03-21 01:59:02,314 INFO mapreduce.Job: The url to track the job: http://LAPTOP-DTDQGQDQ:8088/proxy/application_1679343064205_0001/
2023-03-21 01:59:02,322 INFO mapreduce.Job: Running job: job_1679343064205_0001
2023-03-21 01:59:14,638 INFO mapreduce.Job: Job job_1679343064205_0001 running in uber mode : false
2023-03-21 01:59:14,641 INFO mapreduce.Job:  map 0% reduce 0%
2023-03-21 01:59:19,780 INFO mapreduce.Job:  map 100% reduce 0%
2023-03-21 01:59:26,861 INFO mapreduce.Job:  map 100% reduce 100%
2023-03-21 01:59:27,897 INFO mapreduce.Job: Job job_1679343064205_0001 completed successfully
2023-03-21 01:59:27,999 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=27
                FILE: Number of bytes written=554513
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=123
                HDFS: Number of bytes written=13
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
```

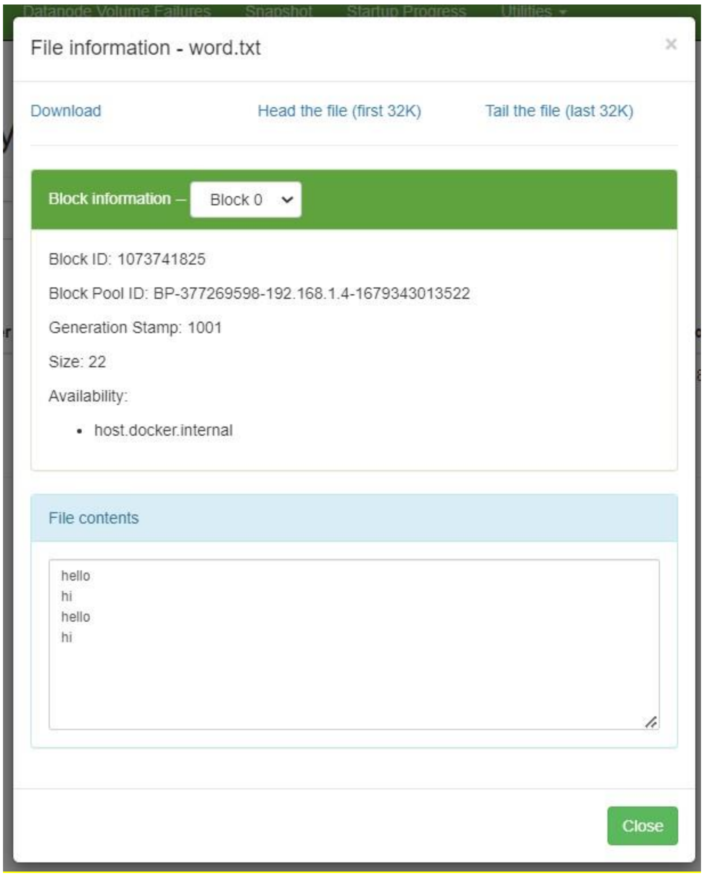
These are the output we get

```
D:\hadoop-3.3.4>hadoop fs -cat /out/*
hello   2
hi      2
```

Graphical User Interface

Input File





Output File

Browse Directory

/out Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	gauta	supergroup	0 B	Mar 21 01:59	1	128 MB	_SUCCESS
-rw-r--r--	gauta	supergroup	13 B	Mar 21 01:59	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

Browse Directory

/ Go!

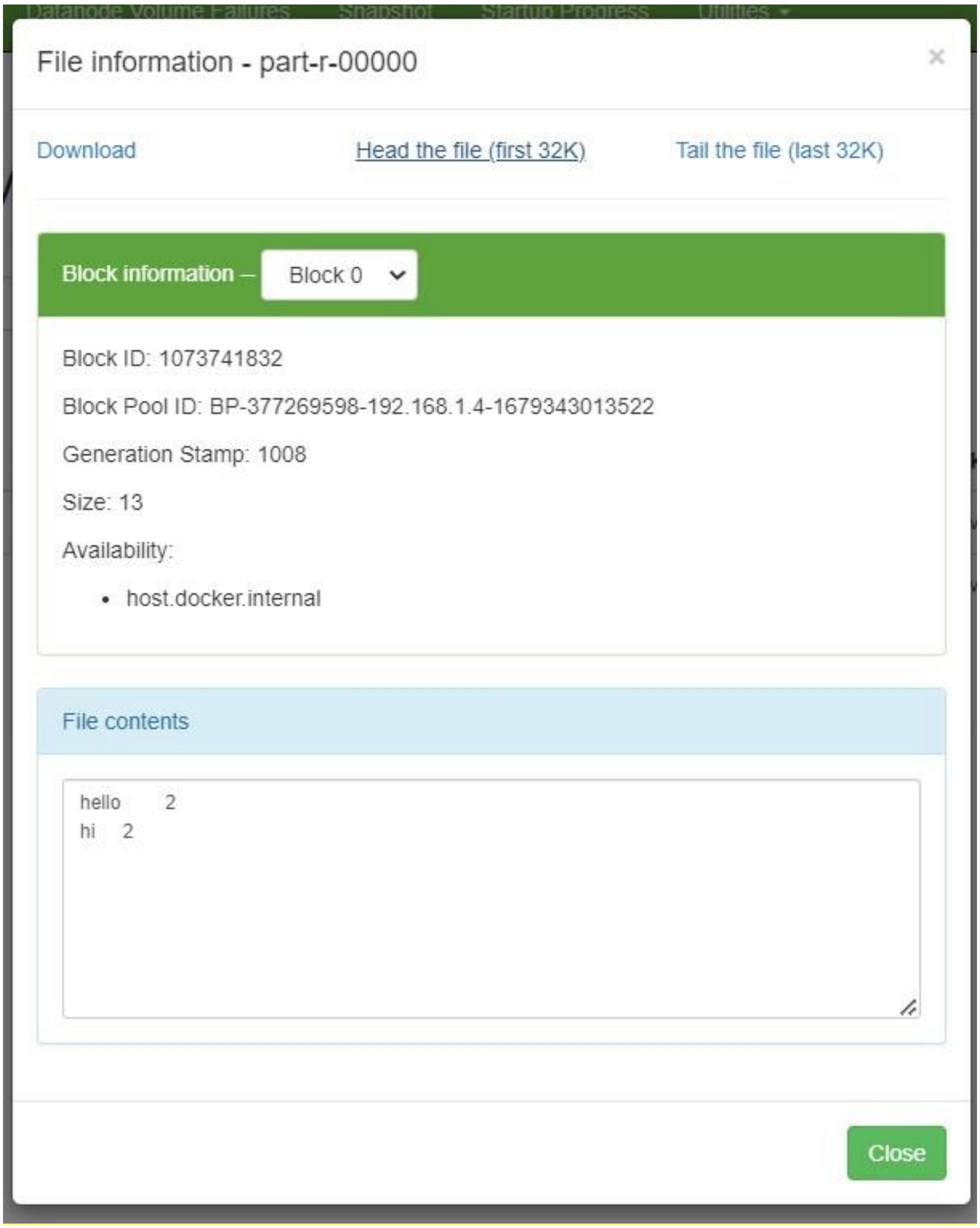
Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	gauta	supergroup	0 B	Mar 21 01:55	0	0 B	input
drwxr-xr-x	gauta	supergroup	0 B	Mar 21 01:43	0	0 B	input_dir
drwxr-xr-x	gauta	supergroup	0 B	Mar 21 01:59	0	0 B	out
drwx------	gauta	supergroup	0 B	Mar 21 01:58	0	0 B	tmp

Showing 1 to 4 of 4 entries

Previous 1 Next

Hadoop, 2022.



Practical – 6

Analytics of Mapper and Reducer Class

The choice of mapper and reducer can have a significant impact on the performance of the MapReduce job and the accuracy of the results. In the case of the algorithm for finding the largest integer and the average of all integers, there are different ways to implement the mapper and reducer functions, each with its advantages and drawbacks. Here are a few examples:

One mapper, one reducer: This is the simplest implementation, where the mapper reads the input file line by line, extracts the integer value, and outputs a key-value pair where the key is a constant value, and the value is an IntWritable. The reducer receives all the values associated with the constant key, and computes the largest integer and the average of all integers in the same way as the above implementation. This implementation has the advantage of simplicity, but it can be slow for large input files, as all the processing is done in a single reducer.

Mapper Implementation

The mapper reads the input file line by line, extracts the integer value, and outputs a key-value pair where the key is a constant value, and the value is an IntWritable.

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static Text constantKey = new Text("constant");      private final
static IntWritable integerWritable = new IntWritable();
    @Override
    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {        String line = value.toString();        int
integer = Integer.parseInt(line);        integerWritable.set(integer);
        context.write(constantKey, integerWritable);
    }
}
```

Reducer Implementation

The reducer receives all the values associated with the constant key, computes the largest integer and the average of all integers. This implementation is simple, but it can be slow for large input files, as all the processing is done in a single reducer.

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersReducer extends Reducer<Text, IntWritable, Text,
DoubleWritable> {
    private static final Text RESULT_KEY = new Text("result");
    private final DoubleWritable resultWritable = new
DoubleWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {        int max = Integer.MIN_VALUE;
        int sum = 0;        int
count = 0;

        for (IntWritable value : values) {              int
integer = value.get();            max = Math.max(max,
integer);
```

```
            sum += integer;            count++;
        }

        double average = (double) sum / count;        resultWritable.set(max + average);
        context.write(RESULT_KEY, resultWritable);
    }
}
```

Multiple mappers, one reducer: In this implementation, the input file is split into multiple chunks, and each mapper processes a chunk independently. The mapper reads the input file, extracts the integer value, and outputs a key-value pair where the key is a constant value, and the value is a TupleWritable containing three IntWritable values: the integer value itself, its count (which is 1), and its sum. The reducer receives all the values associated with the constant key, and computes the largest integer and the average of all integers in the same way as the above implementation. This implementation has the advantage of parallelism, as multiple mappers can process different parts of the input file simultaneously. However, the computation of the sum and count of integers in each mapper can be inefficient for small input files, as it adds overhead to the processing.

The input file is split into multiple chunks, and each mapper processes a chunk independently. The mapper reads the input file, extracts the integer value, and outputs a key-value pair where the key is a constant value, and the value is a TupleWritable containing three IntWritable values: the integer value itself, its count (which is 1), and its sum. The reducer receives all the values associated with the constant key, computes the largest integer and the average of all integers.

Mapper Implementation

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersMapper extends Mapper<LongWritable, Text, Text,
TupleWritable> {
    private final static Text constantKey = new Text("constant");     private final static
IntWritable one = new IntWritable(1);     private final static TupleWritable
tupleWritable = new TupleWritable();

    @Override
    public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
String line = value.toString();        int integer =
Integer.parseInt(line);
        IntWritable integerWritable = new IntWritable(integer);
tupleWritable.set(0, integerWritable);        tupleWritable.set(1, one);
        tupleWritable.set(2, integerWritable);        context.write(constantKey,
tupleWritable);
    }
}
```

This implementation has the advantage of parallelism, as multiple mappers can process different parts of the input file simultaneously. However, the computation of the sum and count of integers in each mapper can be inefficient for small input files, as it adds overhead to the processing.

Reducer Implementation

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersReducer extends Reducer<Text, TupleWritable, Text,
DoubleWritable> {
```

```
    private static final Text RESULT_KEY = new Text("result");
    private final DoubleWritable resultWritable = new
DoubleWritable();

    @Override
    public void reduce(Text key, Iterable<TupleWritable> values, Context context)
throws IOException, InterruptedException {         int max =
Integer.MIN_VALUE;
        int sum = 0;         int
count = 0;

        for (TupleWritable value : values) {
            int integer = ((IntWritable) value.get(0)).get();             int c =
((IntWritable) value.get(1)).get();             int s = ((IntWritable)
value.get(2)).get();
            max = Math.max(max, integer);             sum
+= s;             count += c;
        }

        double average = (double) sum / count;         resultWritable.set(max + average);
        context.write(RESULT_KEY, resultWritable);
    }
}
```

One mapper, multiple reducers: In this implementation, the input file is processed by a single mapper, which outputs a key-value pair for each integer value, where the key is a constant value, and the value is a TupleWritable containing the integer value itself and two IntWritable values representing its count (which is 1) and its sum. Each reducer receives a subset of the values associated with the constant key, and computes the largest integer and the average of all integers in the same way as the above implementation. This implementation has the advantage of parallelism, as multiple reducers can process different subsets of the intermediate values simultaneously. However, it can be slower than the previous implementations for small input files, as it requires more communication overhead between the mapper and the reducers.

The input file is processed by a single mapper, which outputs a keyvalue pair for each integer value, where the key is a constant value, and the value is a TupleWritable containing the integer value itself and two IntWritable values representing its count (which is 1) and its sum. Each reducer receives a subset of the values associated with the constant key, computes the largest integer and the average of all integers.

Mapper Implementation

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersMapper extends Mapper<LongWritable, Text, Text,
TupleWritable> {
    private final static Text constantKey = new Text("constant");     private final static
IntWritable one = new IntWritable(1);     private final static TupleWritable
tupleWritable = new TupleWritable();

    @Override
    public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
String line = value.toString();         int integer =
Integer.parseInt(line);
        IntWritable integerWritable = new IntWritable(integer);
tupleWritable.set(0, integerWritable);         tupleWritable.set(1, one);
        tupleWritable.set(2, integerWritable);         context.write(constantKey,
tupleWritable);
    }
}
```

This implementation has the advantage of parallelism, as multiple reducers can process different subsets of the intermediate values simultaneously. However, it can be slower than the second implementation, as the data transfer between the mapper and reducers can be costly.

Reducer Implementation

```
import java.io.IOException; import
org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

public class IntegersReducer extends Reducer<Text, TupleWritable, Text,
DoubleWritable> {
    private static final Text RESULT_KEY = new Text("result");
    private final DoubleWritable resultWritable = new
DoubleWritable();

    @Override
    public void reduce(Text key, Iterable<TupleWritable> values, Context context)
throws IOException, InterruptedException {        int max =
Integer.MIN_VALUE;
        int sum = 0;        int
count = 0;

        for (TupleWritable value : values) {
            int integer = ((IntWritable) value.get(0)).get();
```

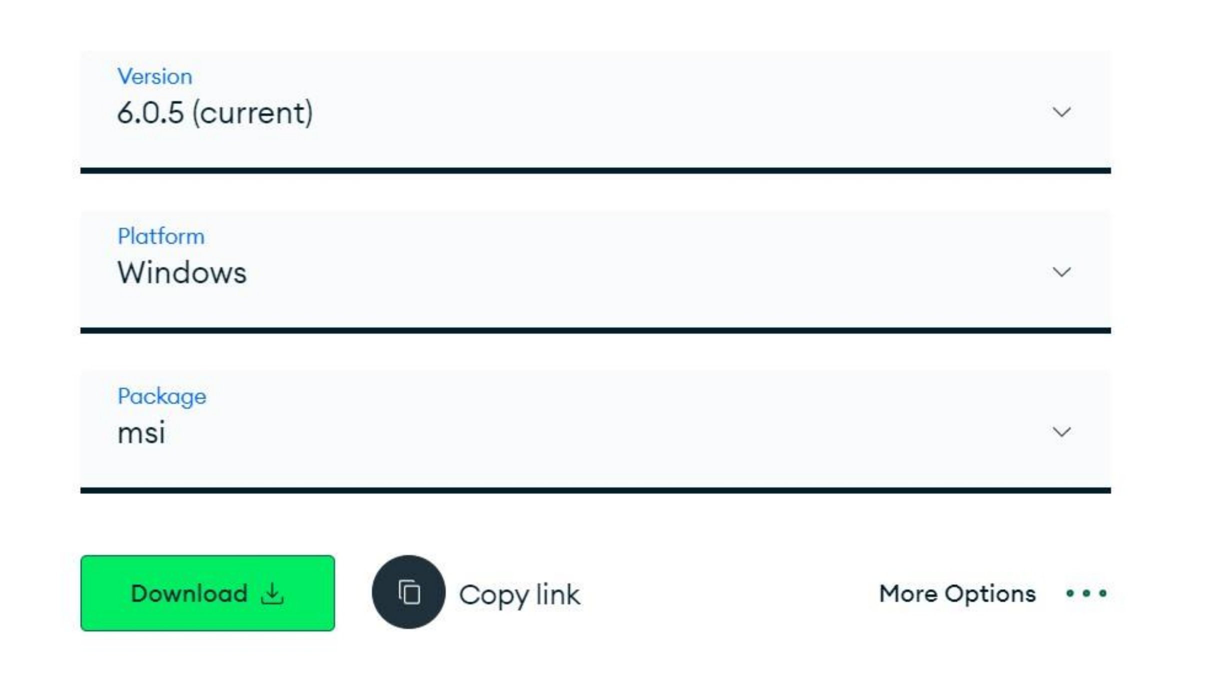
```
            int c = ((IntWritable) value.get(1)).get();             int s =
((IntWritable) value.get(2)).get();
            max = Math.max(max, integer);             sum
+= s;             count += c;
        }

        double average = (double) sum / count;         resultWritable.set(max + average);
        context.write(RESULT_KEY, resultWritable);
    }
}
```

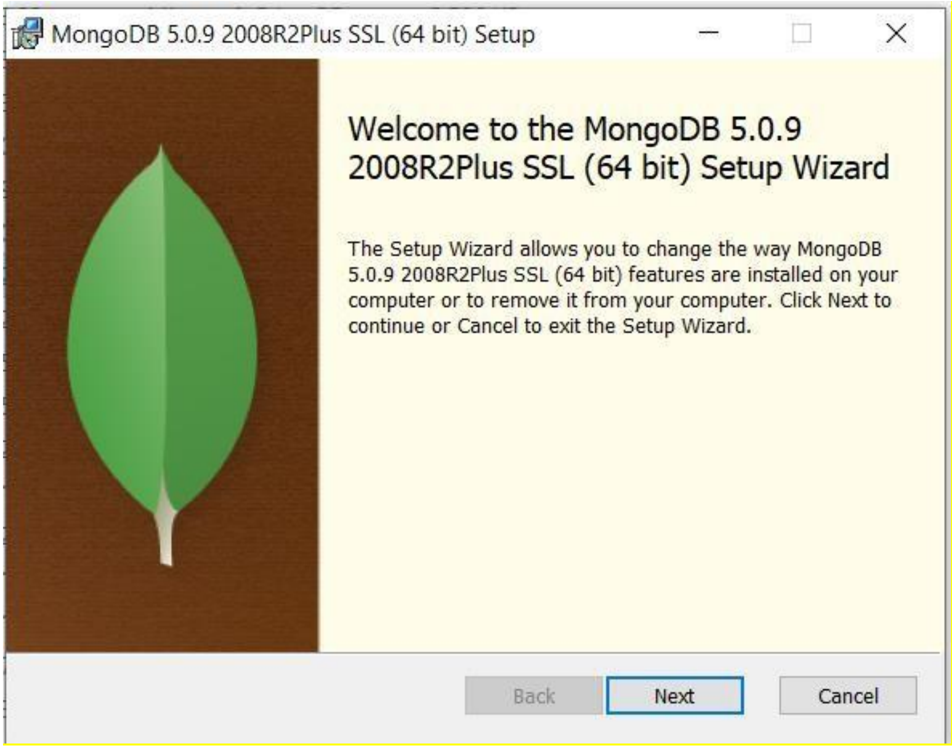
the choice of mapper and reducer can significantly affect the performance and accuracy of a MapReduce job. Each implementation has its advantages and drawbacks, and the best choice depends on the size of the input data and the available resources. A small input file may benefit from a single mapper and reducer implementation, while a large input file may benefit from a multiple mappers and one reducer implementation. Ultimately, the choice of implementation should aim to strike a balance between performance and accuracy.

Practical – 7

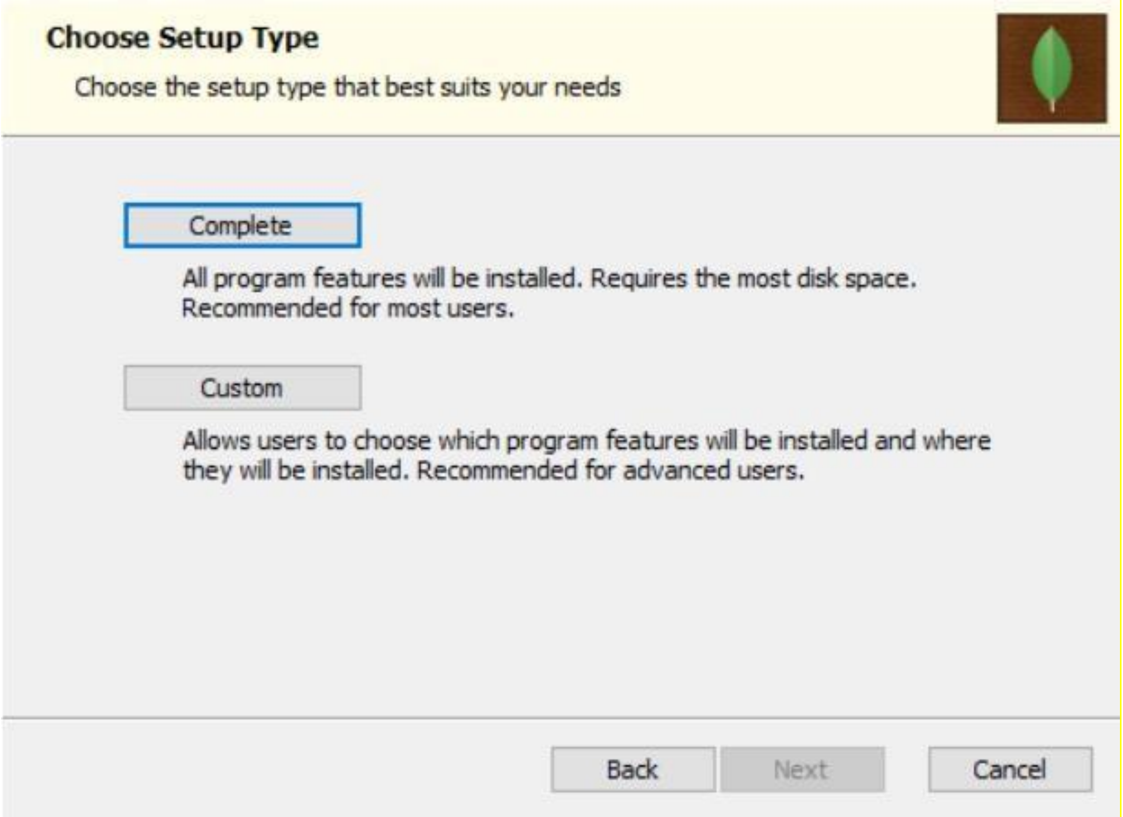
Download the MSI file for Mongodb

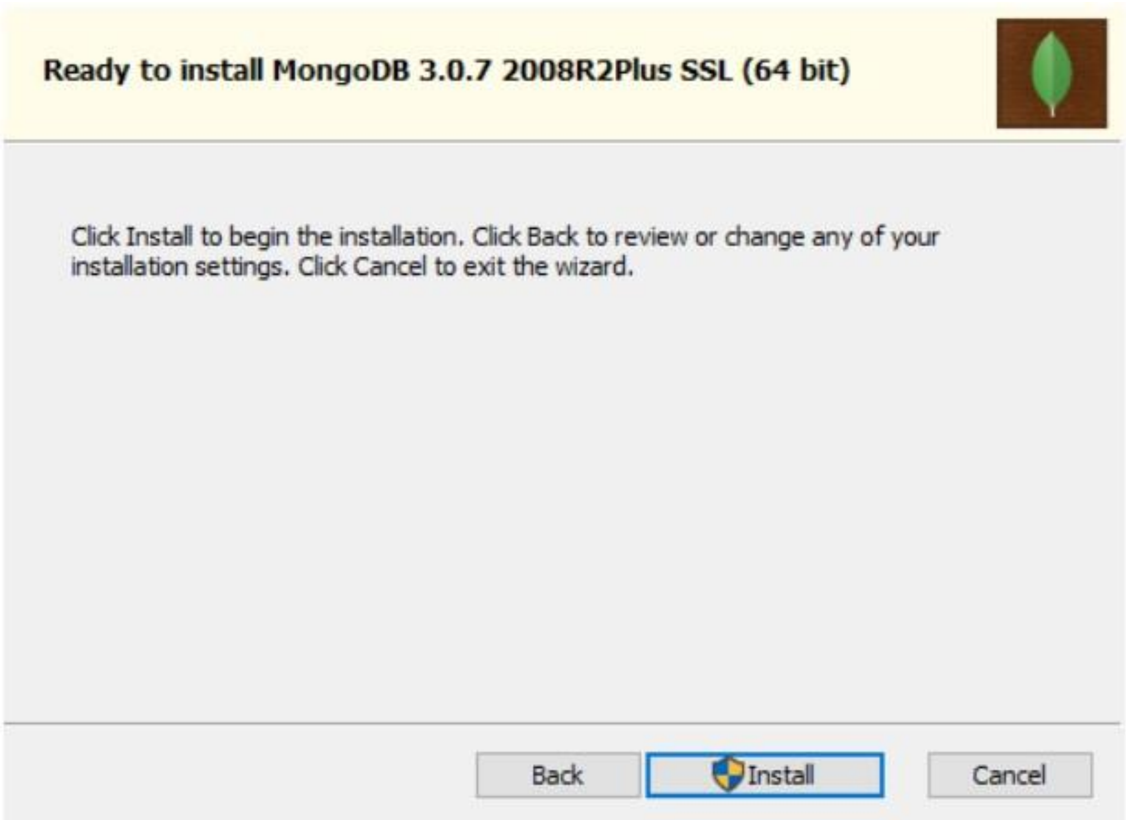


Run the MSI File

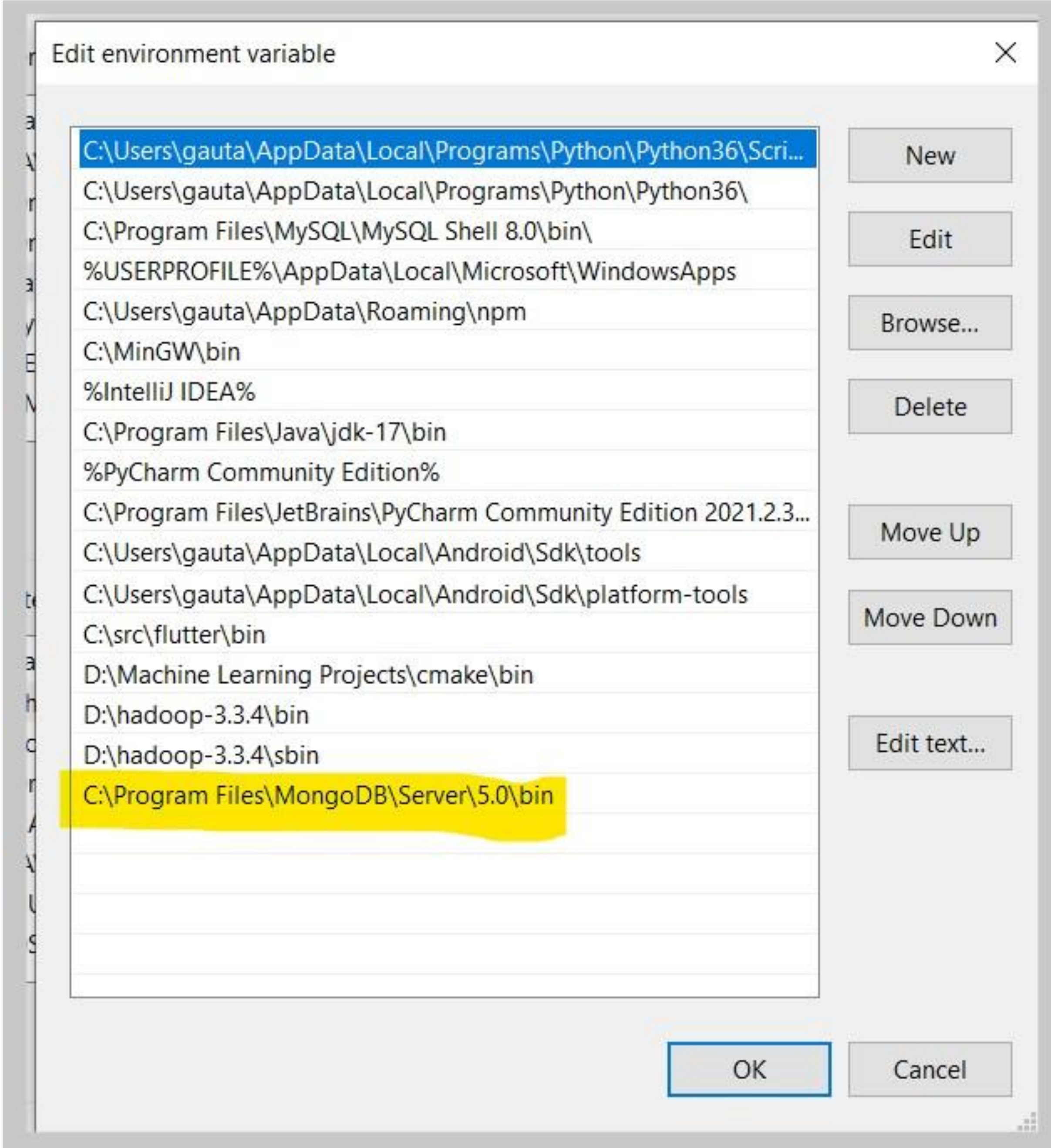


Configure it correctly and install successfully

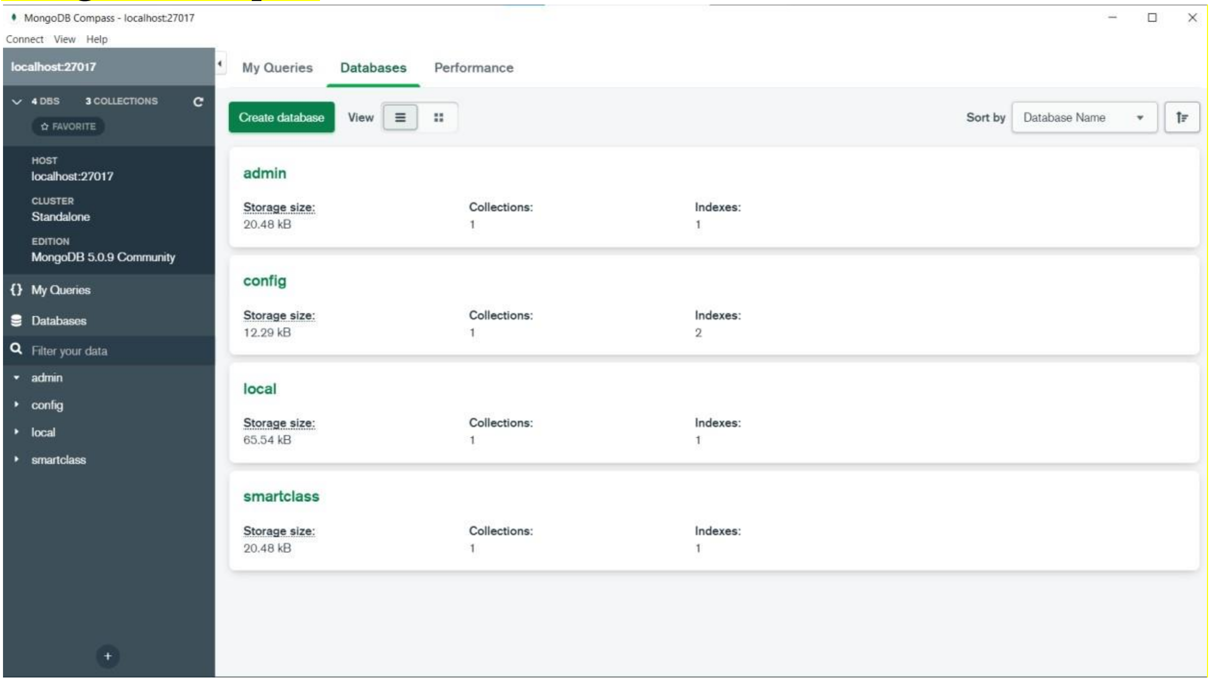




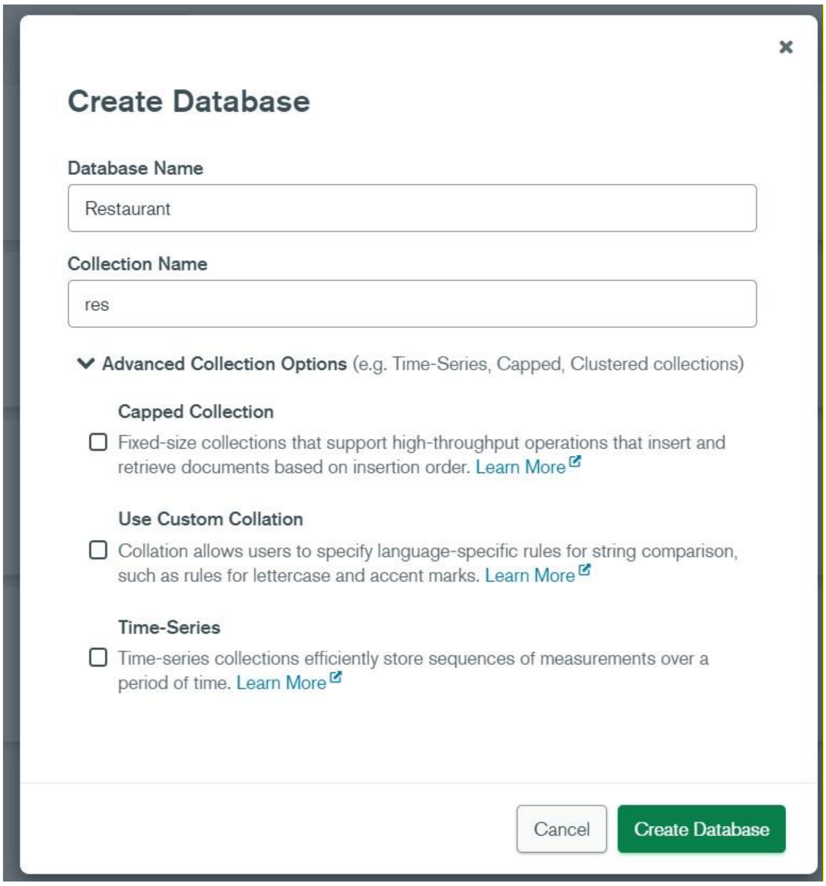
Setup the Environment



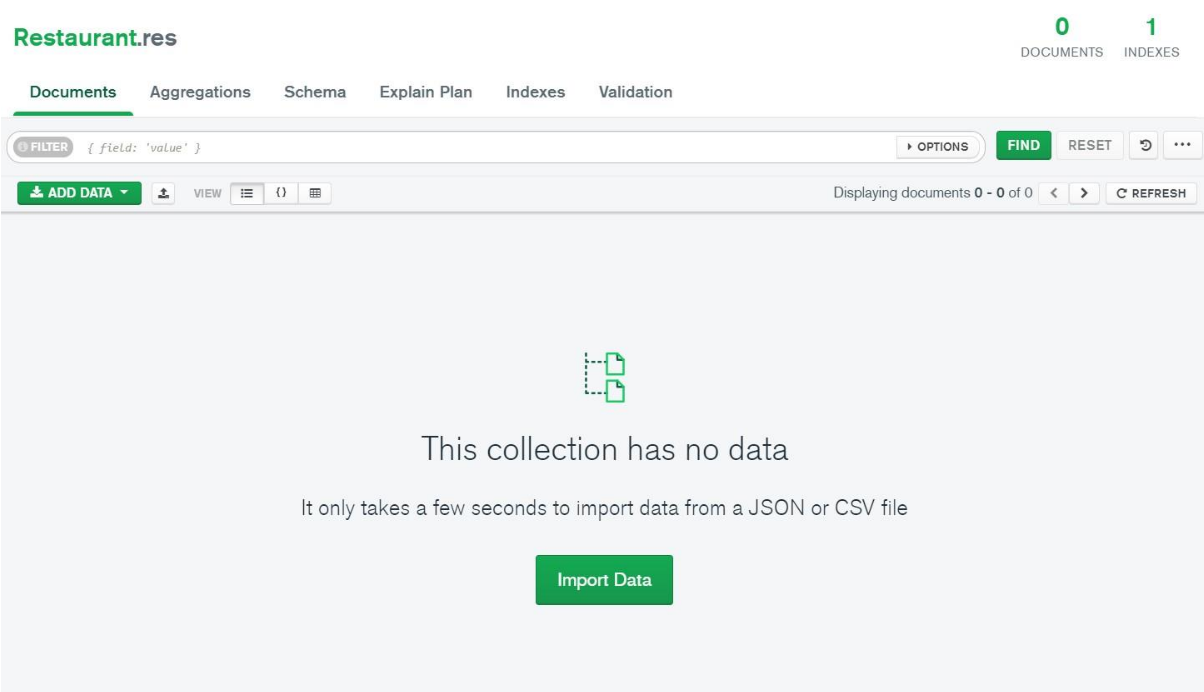
MongoDB Compass

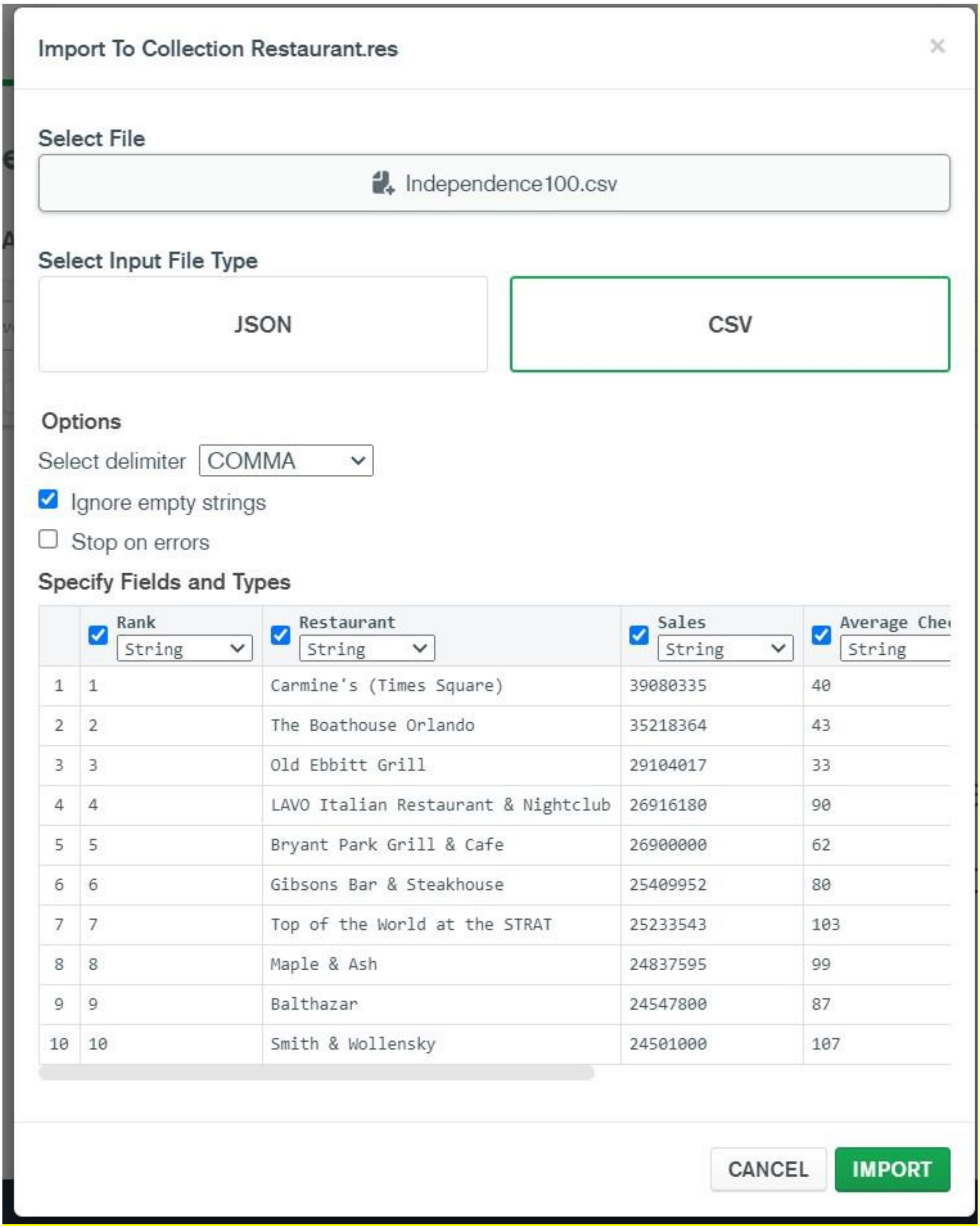


Create a Database



Importing the Restaurant Database from restaurant.csv

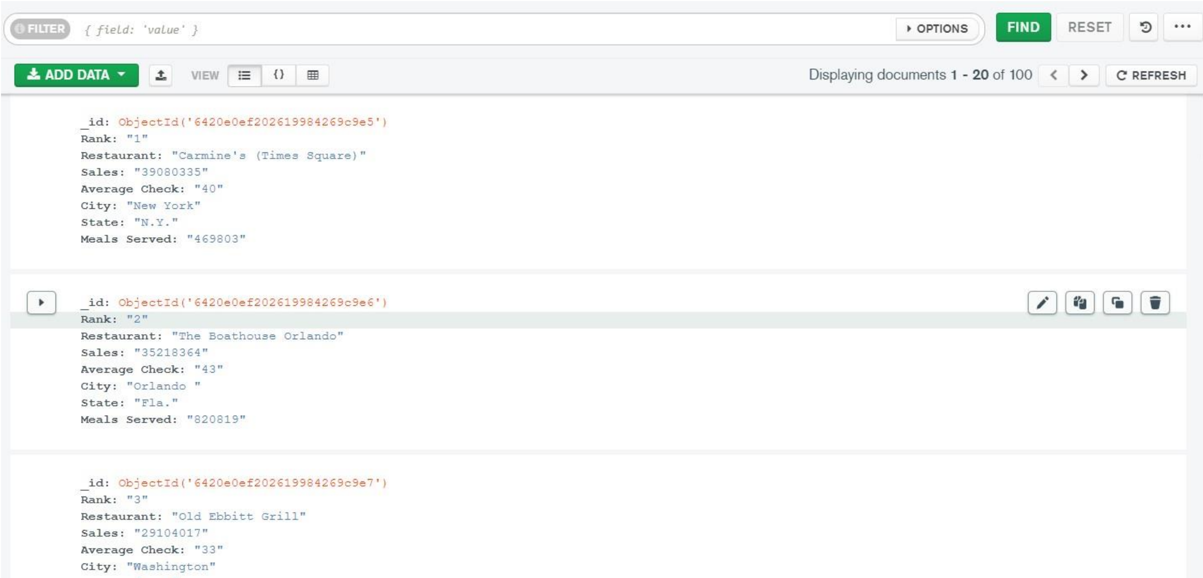




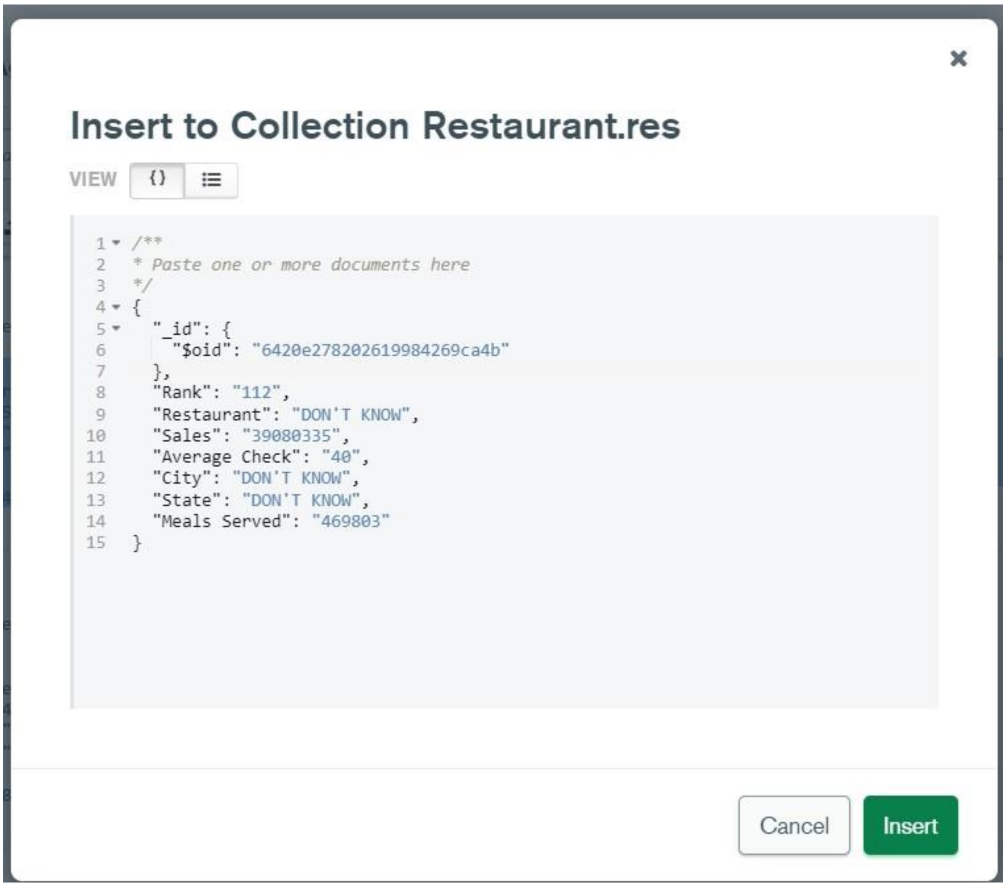
	Rank (String)	Restaurant (String)	Sales (String)	Average Che (String)
1	1	Carmine's (Times Square)	39080335	40
2	2	The Boathouse Orlando	35218364	43
3	3	Old Ebbitt Grill	29104017	33
4	4	LAVO Italian Restaurant & Nightclub	26916180	90
5	5	Bryant Park Grill & Cafe	26900000	62
6	6	Gibsons Bar & Steakhouse	25409952	80
7	7	Top of the World at the STRAT	25233543	103
8	8	Maple & Ash	24837595	99
9	9	Balthazar	24547800	87
10	10	Smith & Wollensky	24501000	107

CURD Operation using GUI

Reading the Database



Insert the new data in database



Updating the Data

```
{
  "_id": {
    "$oid": "6420e278202619984269ca4b"
  },
  "Rank": "112",
  "Restaurant": "DON'T KNOW",
  "Sales": "39080335",
  "Average Check": "40",
  "City": "DON'T KNOW",
  "State": "DON'T KNOW",
  "Meals Served": "43"
}
```

Document modified. CANCEL REPLACE

Deleting the Data

```
{
  "_id": {
    "$oid": "6420e278202619984269ca4b"
  },
  "Rank": "112",
  "Restaurant": "DON'T KNOW",
  "Sales": "39080335",
  "Average Check": "40",
  "City": "DON'T KNOW",
  "State": "DON'T KNOW",
  "Meals Served": "43"
}
```

Document flagged for deletion. CANCEL DELETE

CURD Operation Using Command Line

```
PS C:\Users\gauta> mongo
MongoDB shell version v5.0.9
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("69b11ca3-2c8e-40e8-ae5e-1be6f49b62f4") }
MongoDB server version: 5.0.9
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2023-03-26T11:47:43.385+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> _
```

```
> show dbs
Restaurant  0.000GB
admin       0.000GB
config      0.000GB
local       0.000GB
smartclass  0.000GB
> _
```

Inserting the Data

```
> db.res.insert([{
...   "Rank": "1",
...   "Restaurant": "Carmine's (Times Square)",
...   "Sales": "39080335",
...   "Average Check": "40",
...   "City": "New York",
...   "State": "N.Y.",
...   "Meals Served": "469803"
... }])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 1,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

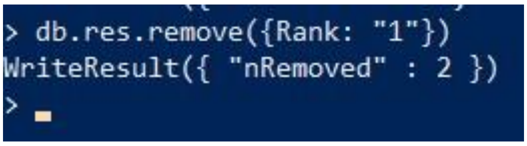
Reading the Data

```
> db.res.find().pretty()
{
        "_id" : ObjectId("6420e0ef202619984269c9e5"),
        "Rank" : "1",
        "Restaurant" : "Carmine's (Times Square)",
        "Sales" : "39080335",
        "Average Check" : "40",
        "City" : "New York",
        "State" : "N.Y.",
        "Meals Served" : "469803"
}
{
        "_id" : ObjectId("6420e0ef202619984269c9e6"),
        "Rank" : "2",
        "Restaurant" : "The Boathouse Orlando",
        "Sales" : "35218364",
        "Average Check" : "43",
        "City" : "Orlando ",
        "State" : "Fla.",
        "Meals Served" : "820819"
}
{
        "_id" : ObjectId("6420e0ef202619984269c9e7"),
        "Rank" : "3",
        "Restaurant" : "Old Ebbitt Grill",
        "Sales" : "29104017",
        "Average Check" : "33",
        "City" : "Washington",
        "State" : "D.C.",
        "Meals Served" : "892830"
```

Updating the Data

```
> db.res.update({Rank: "1"},
... {
...    "Rank": "1",
...    "Restaurant": "Carmine's (Times Square)",
...    "Sales": "39080335",
...    "Average Check": "40",
...    "City": "New York",
...    "State": "N.Y.",
...    "Meals Served": "469803"
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Removing the Data



Practical – 8

Cassandra

Cassandra is a distributed, open-source NoSQL database system that was originally developed by Facebook. It is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra is a column-family database, meaning that data is organized by columns rather than rows, and it provides features such as automatic data partitioning, replication, and fault tolerance.

Installation of Cassandra

Download and Setup the Cassandra

Releases

Latest GA Version

Download the latest Apache Cassandra 4.1 GA release:
Latest release on 2023-03-21
Maintained until 4.4.0 release (May-July 2025)

4.1.1

(pgp, sha256 and sha512)

Previous Stable Version

Download the latest Apache Cassandra 4.0 release:
Latest release on 2023-02-14
Maintained until 4.3.0 release (May-July 2024)

4.0.8

(pgp, sha256 and sha512)

Older Supported Releases

The following older Cassandra releases are still supported:

Download the latest Apache Cassandra 3.11 release:
Latest release on 2022-10-23
Maintained until 4.2.0 release (May-July 2023)

3.11.14

(pgp, sha256 and sha512)

Apache Cassandra 3.0
Latest release on 2022-10-23
Maintained until 4.2.0 release (May-July 2023)

3.0.28

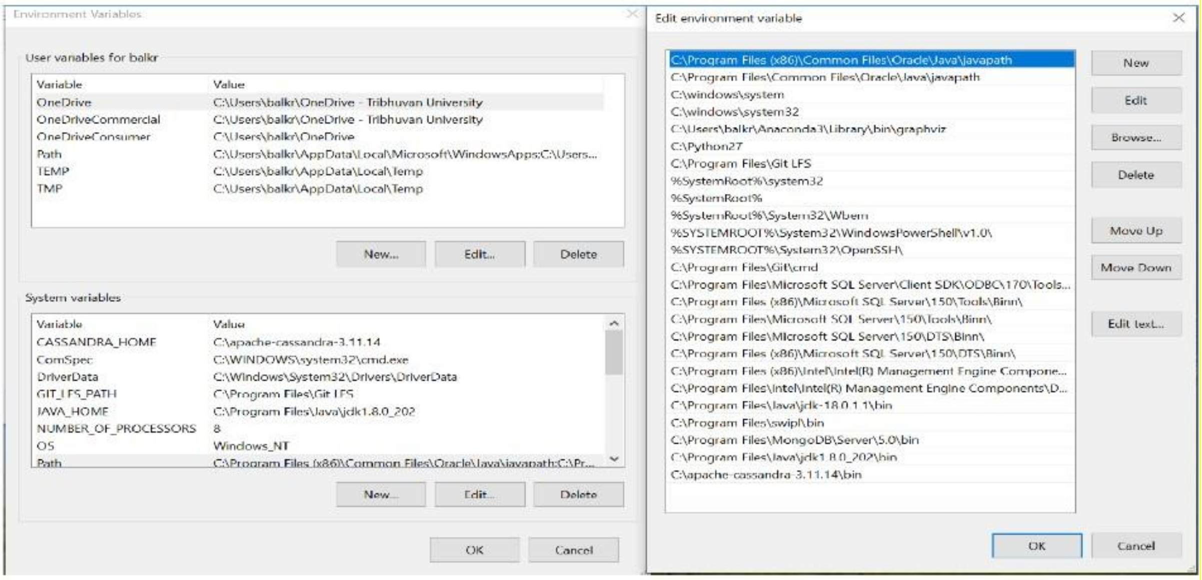
(pgp, sha256 and sha512)

Install JDK and set the path in the Environment Variable

Linux macOS **Windows**

Product/file description	File size	Download
x64 Compressed Archive	180.80 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.zip (sha256)
x64 Installer	159.94 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.exe (sha256)
x64 MSI Installer	158.72 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.msi (sha256)

Setting up environment Variable



CURD Operation in Cassandra

Help Command

```
cqlsh> help

Documented shell commands:
===========================
CAPTURE  CLS          COPY  DESCRIBE  EXPAND  LOGIN   SERIAL  SOURCE   UNICODE
CLEAR    CONSISTENCY  DESC  EXIT      HELP    PAGING  SHOW    TRACING

CQL help topics:
================
AGGREGATES               CREATE_KEYSPACE           DROP_TRIGGER      TEXT
ALTER_KEYSPACE           CREATE_MATERIALIZED_VIEW  DROP_TYPE         TIME
ALTER_MATERIALIZED_VIEW  CREATE_ROLE               DROP_USER         TIMESTAMP
ALTER_TABLE              CREATE_TABLE              FUNCTIONS         TRUNCATE
ALTER_TYPE               CREATE_TRIGGER            GRANT             TYPES
ALTER_USER               CREATE_TYPE               INSERT            UPDATE
APPLY                    CREATE_USER               INSERT_JSON       USE
ASCII                    DATE                      INT               UUID
BATCH                    DELETE                    JSON
BEGIN                    DROP_AGGREGATE            KEYWORDS
BLOB                     DROP_COLUMNFAMILY         LIST_PERMISSIONS
BOOLEAN                  DROP_FUNCTION             LIST_ROLES
COUNTER                  DROP_INDEX                LIST_USERS
CREATE_AGGREGATE         DROP_KEYSPACE             PERMISSIONS
CREATE_COLUMNFAMILY      DROP_MATERIALIZED_VIEW    REVOKE
CREATE_FUNCTION          DROP_ROLE                 SELECT
CREATE_INDEX             DROP_TABLE                SELECT_JSON

cqlsh>
cqlsh>
```

Create A database

```
cqlsh:student> CREATE TABLE Student_Info(
           ...     ID  int PRIMARY KEY,
           ...     Name varchar,
           ...     Age int,
           ...     Mobile varint,
           ...     Year int,
           ...     Address varchar
           ...     );
cqlsh:student> select * from Student_Info;

 id | address | age | mobile | name | year
----+---------+-----+--------+------+------

(0 rows)
```

Insert the data into database

```
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 2, 'Shyam',22,2072,'Madhyampath');
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 3, 'Hari',23,2073,'New Road');
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 4, 'Dinesh',22,2073,'Bagar');
cqlsh:student>
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 5, 'Shyam',23,2073,'Simpani');
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 6, 'Mahesh',24,2071,'Lakeside');
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 7, 'Biplav',23,2074,'Madhyampath');
cqlsh:student> INSERT INTO Student_Info (id, Name, Age,Year,Address ) VALUES ( 8, 'Bhimesh',22,2073,'Madhyampath');
cqlsh:student> select * from Student_info;

 id | address     | age | name    | year
----+-------------+-----+---------+------
  5 |     Simpani |  23 |   Shyam | 2073
  1 |       Bagar |  23 |     Ram | null
  8 | Madhyampath |  22 | Bhimesh | 2073
  2 | Madhyampath |  22 |   Shyam | 2072
  4 |       Bagar |  22 |  Dinesh | 2073
  7 | Madhyampath |  23 |  Biplav | 2074
  6 |    Lakeside |  24 |  Mahesh | 2071
  3 |    New Road |  23 |    Hari | 2073
```

Read the Data from Database

```
cqlsh:student> select * from Student_info where Address='Madhyampath' allow filtering;

 id | address     | age | name    | year
----+-------------+-----+---------+------
  8 | Madhyampath |  22 | Bhimesh | 2073
  2 | Madhyampath |  22 |   Shyam | 2072
  7 | Madhyampath |  23 |  Biplav | 2074
```

Update the Data in Database

```
cqlsh:student> update Student_Info set name='Rahul',year=2073 where sid=1;
cqlsh:student> select * from Student_Info;

 sid | address     | age | name    | year
-----+-------------+-----+---------+------
   5 |     Simpani |  23 |   Shyam | 2073
   1 |       Bagar |  23 |   Rahul | 2073
   8 | Madhyampath |  22 | Bhimesh | 2073
   2 | Madhyampath |  22 |   Shyam | 2072
   4 |       Bagar |  22 |  Dinesh | 2073
   7 | Madhyampath |  23 |  Biplav | 2074
   6 |    Lakeside |  24 |  Mahesh | 2071
   3 |    New Road |  23 |    Hari | 2073
```

Delete data from Database

```
cqlsh:student> delete year from Student_Info where sid=5;
cqlsh:student> select * from Student_Info;

 sid | address     | age | name    | year
-----+-------------+-----+---------+------
   5 |     Simpani |  23 |   Shyam | null
   1 |       Bagar |  23 |   Rahul | 2073
   8 | Madhyampath |  22 | Bhimesh | 2073
   2 | Madhyampath |  22 |   Shyam | 2072
   4 |       Bagar |  22 |  Dinesh | 2073
```

Practical – 9

K-Means Clustering Using MapReduce

K-means clustering is a popular algorithm for clustering data points into groups based on their similarity. When dealing with large datasets, it can be challenging to implement K-means clustering in main memory. One solution is to use MapReduce, a programming model for processing large datasets in a distributed manner[1].

Here are the general steps to implement K-means clustering using MapReduce:

1. Divide the dataset into smaller subsets that can fit into memory.

2. Initialize the centroids for each subset using a random or heuristic method.

3. For each subset, assign each data point to the nearest centroid.

4. Calculate the new centroids for each subset based on the assigned data points.

5. Merge the centroids from each subset and calculate the final centroids.

6. Repeat steps 3-5 until convergence.

Pseudo Code

```
centroids = k random sampled points from the dataset. do:
Map:
-     Given a point and the set of centroids.
-     Calculate the distance between the point and each centroid.
-     Emit the point and the closest centroid.
Reduce:
-     Given the centroid and the points belonging to its cluster.
-     Calculate the new centroid as the arithmetic mean position of the points.
-     Emit the new centroid. prev_centroids = centroids. centroids = new_centroids.
while prev_centroids - centroids > threshold.
```

Here's an example of implementing K-means clustering using MapReduce in Python with Apache Spark:

```
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans, KMeansModel from numpy import
array

# Initialize Spark context
sc = SparkContext("local", "K-means clustering")

# Load data from file data =
sc.textFile("data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Initialize K-means model
clusters = KMeans.train(parsedData, k=2, maxIterations=10,
initializationMode="random")

# Print the center of each cluster
print("Cluster centers:") for center in
clusters.clusterCenters:
    print(center)
```

Advantages of using MapReduce in KMeans Clustering

MapReduce is a programming framework for processing large-scale datasets by exploiting the parallelism among a cluster of computing nodes. K-means clustering is a popular algorithm for clustering data points into groups based on their similarity. There are several advantages of using MapReduce for K-means clustering:

1. Scalability: MapReduce allows K-means clustering to scale to large datasets by distributing the computation across multiple nodes in a cluster.

2. Fault tolerance: MapReduce provides fault tolerance by automatically handling node failures and re-executing failed tasks on other nodes.

3. Flexibility: MapReduce allows K-means clustering to be implemented on a variety of distributed computing platforms, such as Apache Hadoop or Apache Spark.

4. Reduced communication overhead: MapReduce reduces the communication overhead between nodes by partitioning the data and processing each partition independently.

5. Efficient computation: MapReduce allows K-means clustering to be computed efficiently by minimizing the amount of data that needs to be transferred between nodes.

Overall, using MapReduce for K-means clustering allows for efficient and scalable computation of large datasets, while providing fault tolerance and flexibility in implementation.

Mapper Class

```
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {          // Parse data point from input value
        Point point = new Point(value.toString());
        // Find nearest centroid
```

```
        int nearest = 0;
        double minDistance = Double.MAX_VALUE;           for (int i
= 0; i < k; i++) {
            double distance = point.distance(centroids.get(i));              if
(distance < minDistance) {
                nearest = i;
                minDistance = distance;
            }
        }
        // Emit key-value pair for nearest centroid          context.write(new
IntWritable(nearest), new
PointWritable(point, 1));
    }
```

Reducer Class

```
    protected void reduce(IntWritable key,
Iterable<PointWritable> values, Context context) throws
IOException, InterruptedException {
        // Initialize sum and count for cluster          Point sum =
new Point();
        int count = 0;

        // Accumulate sum and count for each data point in cluster
        for (PointWritable value : values) {             sum.add(value.getPoint());
```

```
            count += value.getCount();
        }

        // Calculate new centroid for cluster
        Point centroid = sum.divide(count);
        // Emit new centroid as key-value pair
        context.write(key, new PointWritable(centroid, count));         }
```

Complete code for KMeans Clustering using MapReduce

```
public class KMeansMR {

    public static class KMeansMapper extends Mapper<LongWritable, Text,
IntWritable, PointWritable> {

        private int k;
        private List<Point> centroids;

        @Override
        protected void setup(Context context) throws IOException,
InterruptedException {             super.setup(context);
            k = context.getConfiguration().getInt("k", 2);                 centroids = new
ArrayList<>();
            // Initialize centroids here          }
```

```
        @Override
        protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {             // Parse data point from input value
            Point point = new Point(value.toString());
            // Find nearest centroid             int
nearest = 0;
            double minDistance = Double.MAX_VALUE;             for (int i
= 0; i < k; i++) {
                double distance = point.distance(centroids.get(i));                 if (distance
< minDistance) {
                    nearest = i;
                    minDistance = distance;
                }
            }
            // Emit key-value pair for nearest centroid             context.write(new
IntWritable(nearest), new
PointWritable(point, 1));
        }
    }

    public static class KMeansReducer extends Reducer<IntWritable, PointWritable,
IntWritable, PointWritable> {

        @Override
```

```
    protected void reduce(IntWritable key,
Iterable<PointWritable> values, Context context) throws
IOException, InterruptedException {
        // Initialize sum and count for cluster            Point sum = new
Point();
        int count = 0;

        // Accumulate sum and count for each data point in cluster
        for    (PointWritable    value    :    values)    {
sum.add(value.getPoint());                              count  +=
value.getCount();
        }

        // Calculate new centroid for cluster
        Point centroid = sum.divide(count);
        // Emit new centroid as key-value pair
        context.write(key, new PointWritable(centroid, count));
    }
  }
}
```

Practical – 10

Case Study

Big Data Analytics on AWS Cloud

Amazon Web Services (AWS) is a cloud computing platform that provides a wide range of services for big data analytics. AWS offers a broad platform of managed services to help build, secure, and seamlessly scale end-to-end big data applications quickly and with ease. AWS provides infrastructure and tools to tackle big data projects, whether they require real-time streaming or batch data processing. AWS offers solutions for data ingestion, data cleansing, data analytics and visualization, and data archiving. AWS experts have built content to help beginners build a career or build their knowledge of data analytics in the AWS Cloud. This whitepaper examines some tools available on AWS for big data analytics and provides a good reference point when starting to design big data applications.

Benefits of Amazon AWS for Big Data Analytics

Using AWS for big data analytics provides several benefits, including ideal usage patterns, cost model, performance, durability and availability, scalability and elasticity, and interfaces. AWS allows users to build an entire analytics application to power their

business, scale a Hadoop cluster from zero to thousands of servers within just a few minutes, and then turn it off again when done. This means big data workloads can be processed in less time and at a lower cost. AWS also provides the most serverless options for data analytics in the cloud, including options for data warehousing, big data analytics, real-time data, data integration, and more. AWS manages the underlying infrastructure, allowing users to focus solely on their application. AWS provides a broad platform of managed services to help build, secure, and seamlessly scale endto-end big data applications quickly and with ease, whether applications require real-time streaming or batch data processing.

Scalability

One of the major scalability benefits of using AWS for big data analytics is the ability to easily scale up and down based on the amount of input data and the type of analysis. Analyzing large datasets requires significant compute capacity that can vary in size, and AWS's pay-as-you-go cloud computing model is ideally suited for this characteristic of big data workloads. With AWS, users can build virtually any big data application and scale a Hadoop cluster from zero to thousands of servers within just a few minutes, and then turn it off again when done. This means big data workloads can be processed in less time and at a lower cost. AWS provides a broad and deep portfolio of purpose-built analytics services optimized for unique analytics use cases, allowing users to easily move a portion of data from one data store to another and providing unified governance.

Big Data Analytics Options on AWS Data Warehousing:

Amazon Redshift, a cloud-based data warehouse service, to store its vast amounts of structured and unstructured data. Amazon Redshift provides a highly scalable, cost-effective solution that allows Amazon to store and analyze its data easily. Amazon also uses AWS Glue, a fully managed ETL (extract, transform, and load) service, to move data from various sources into Redshift. AWS Glue simplifies the process of preparing and loading data into Redshift, enabling Amazon to perform analytics on large data sets quickly.

Data Processing:

AWS Lambda, a serverless computing service, to process large amounts of data in real-time. Lambda allows Amazon to run code without provisioning or managing servers, enabling it to process data at scale quickly. Amazon also uses Amazon Kinesis, a platform for streaming data on AWS, to collect and process real-time data streams. Kinesis enables Amazon to process large amounts of data from multiple sources in real-time, allowing it to gain insights into customer behavior quickly.

Data Analytics:

Amazon EMR, a managed big data processing service, to perform data analytics on large data sets. EMR allows Amazon to process vast amounts of data using Apache Hadoop, Spark, and other big data processing frameworks. EMR provides a scalable, cost-effective solution for Amazon to process and analyze large amounts of data easily.

Data Storage:

Amazon S3, a highly scalable and secure object storage service, to store its vast amounts of unstructured data, such as customer reviews, clickstream data, and product information. S3 provides a cost-effective solution for Amazon to store and access its data easily. Amazon also uses Amazon Glacier, a low-cost storage service, to store archival data that is rarely accessed. Glacier provides a highly durable, cost-effective solution for Amazon to store data that is not frequently accessed.

Data Visualization:

Amazon QuickSight, a cloud-based business intelligence service, to create visualizations and dashboards for its data. QuickSight provides a user-friendly interface that allows Amazon to create visualizations easily and quickly. QuickSight also provides advanced analytics features, such as machine learning, anomaly detection, and forecasting, enabling Amazon to gain insights into its data quickly.

Cost Effectiveness in Big Data

AWS helps with cost-effectiveness in big data analytics by providing a pay-as-you-go cloud computing model, where applications can easily scale up and down based on the amount of input data and the type of analysis. This allows users to pay only for the resources they use, avoiding the need to over-provision and reducing costs. AWS also offers a wide range of big data analytics options, including Amazon Kinesis, AWS Lambda, Amazon EMR, AWS Glue, Amazon Machine Learning, Amazon DynamoDB, Amazon Redshift, Amazon Elasticsearch Service, Amazon QuickSight, Amazon EC2, and Amazon Athena, which are designed to be cost-effective and provide high performance and scalability. For example, Amazon Redshift is 3x faster and at least 50 percent less expensive than other cloud data warehouses, and Spark on Amazon EMR runs 1.7x faster than other cloud providers. AWS also provides ideal usage patterns, durability and availability, and anti-patterns to help users optimize their big data analytics workloads for costeffectiveness.

Implementation

Amazon EMR

Amazon Elastic MapReduce (EMR) is a cloud-based big data processing service provided by Amazon Web Services (AWS). It allows you to easily process large amounts of data using popular distributed computing technologies such as Hadoop, Spark, and
Hive, without the need for managing the underlying infrastructure.

EMR is built on top of Amazon EC2 and Amazon S3, and allows you to easily create, configure, and scale Hadoop clusters to process big data. EMR provides a simple web interface and a set of APIs to manage the lifecycle of the cluster and to submit jobs for processing.

Here are some key features of Amazon EMR:

Scalable: EMR allows you to easily scale the processing power of your Hadoop cluster up or down depending on your needs.

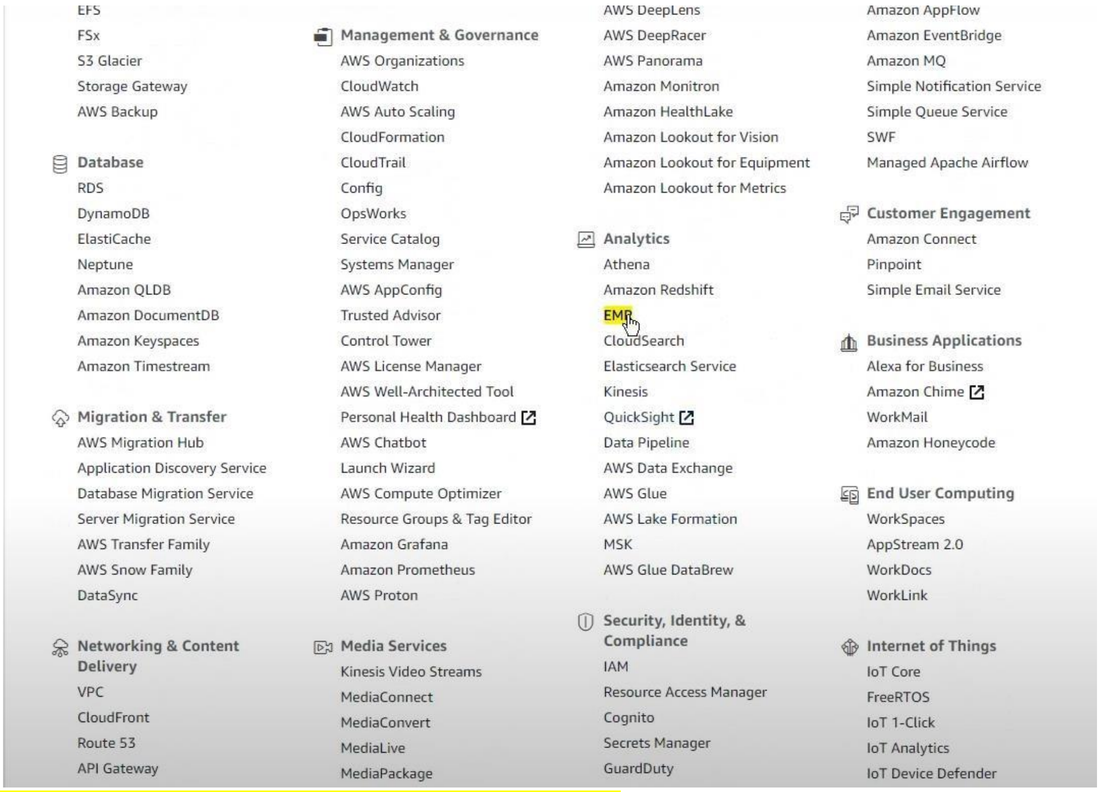
Easy to use: EMR provides a web-based console that makes it easy to create, configure, and manage Hadoop clusters. You can also use APIs to automate the management of your clusters.

Low cost: EMR provides a pay-as-you-go pricing model, which means you only pay for the resources you use.

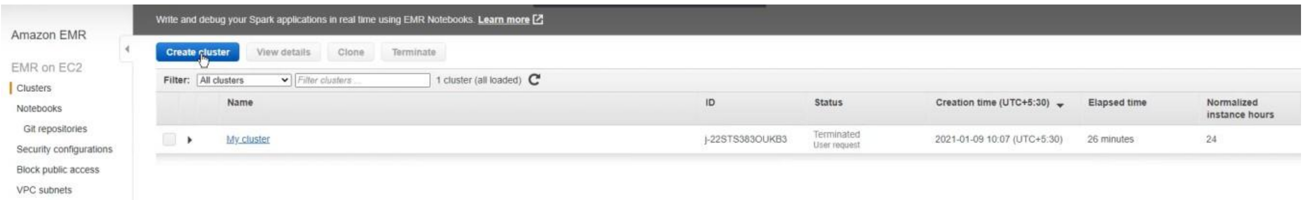
Secure: EMR integrates with AWS security services, such as IAM and KMS, to provide a secure environment for processing your data.

Integration with other AWS services: EMR integrates with other AWS services, such as Amazon S3, Amazon Redshift, and Amazon DynamoDB, to provide a seamless big data processing experience.

Amazon EMR

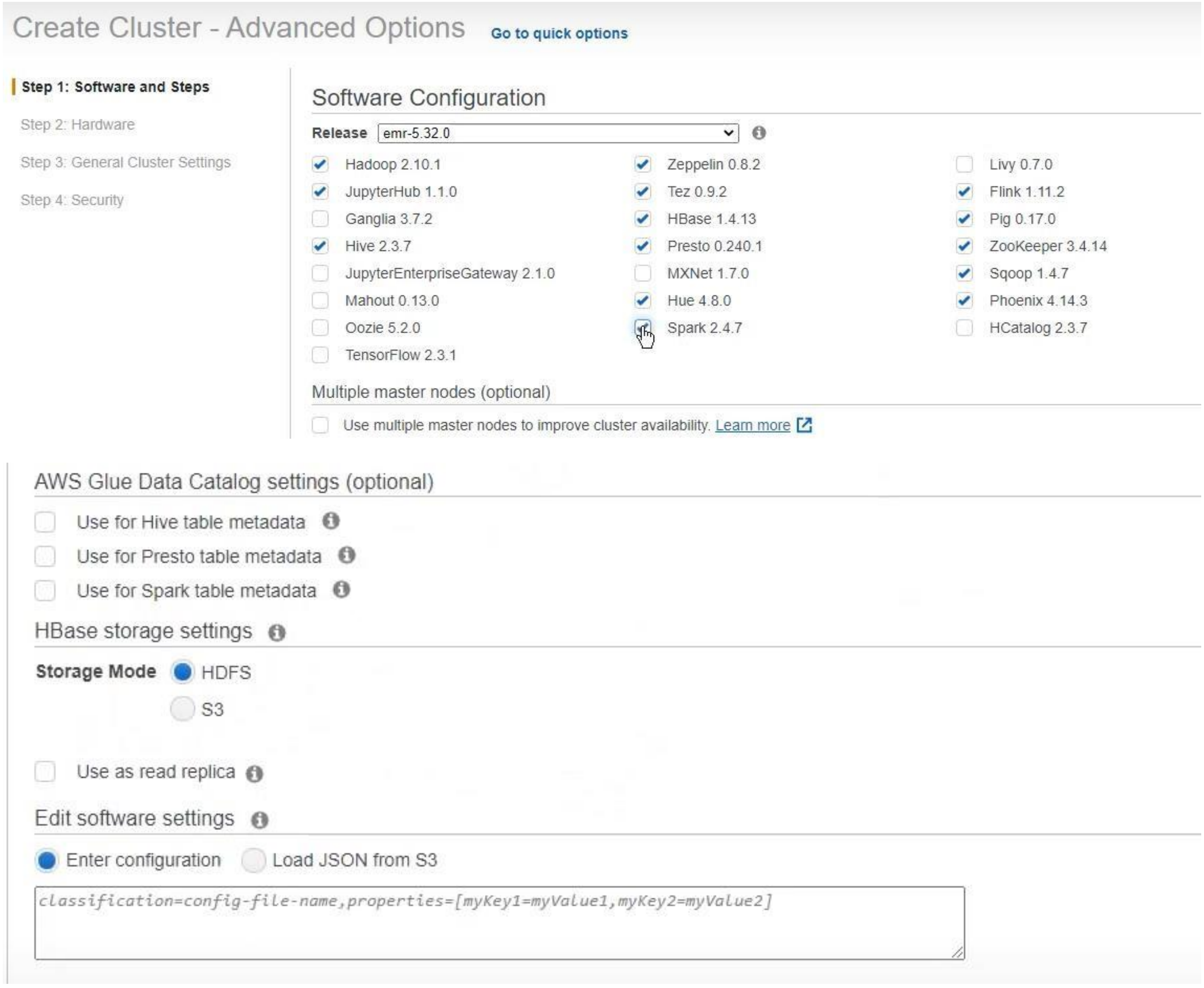


Creating Cluster for Big Data Analytics

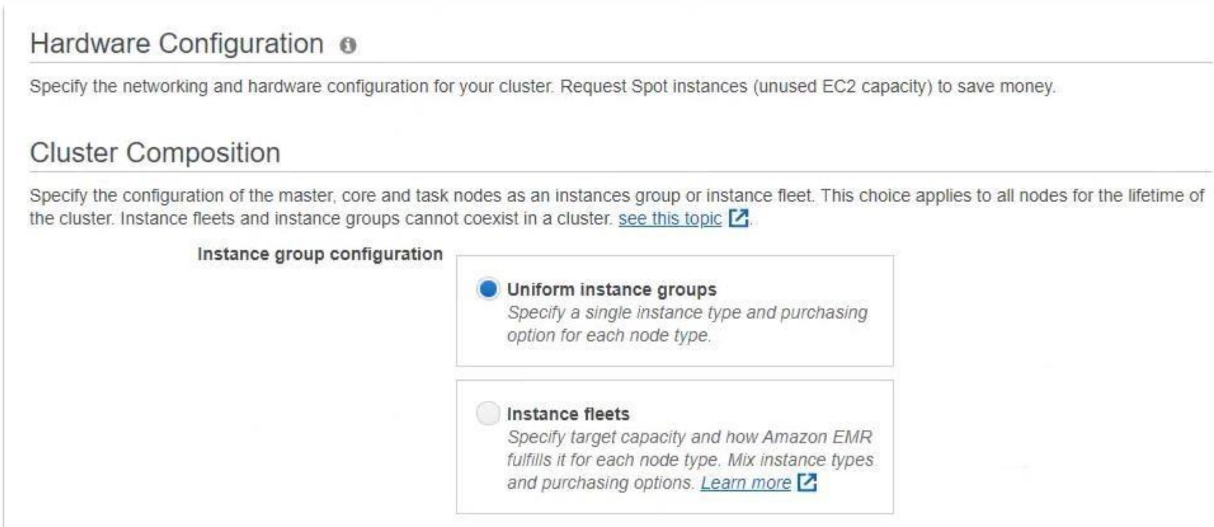


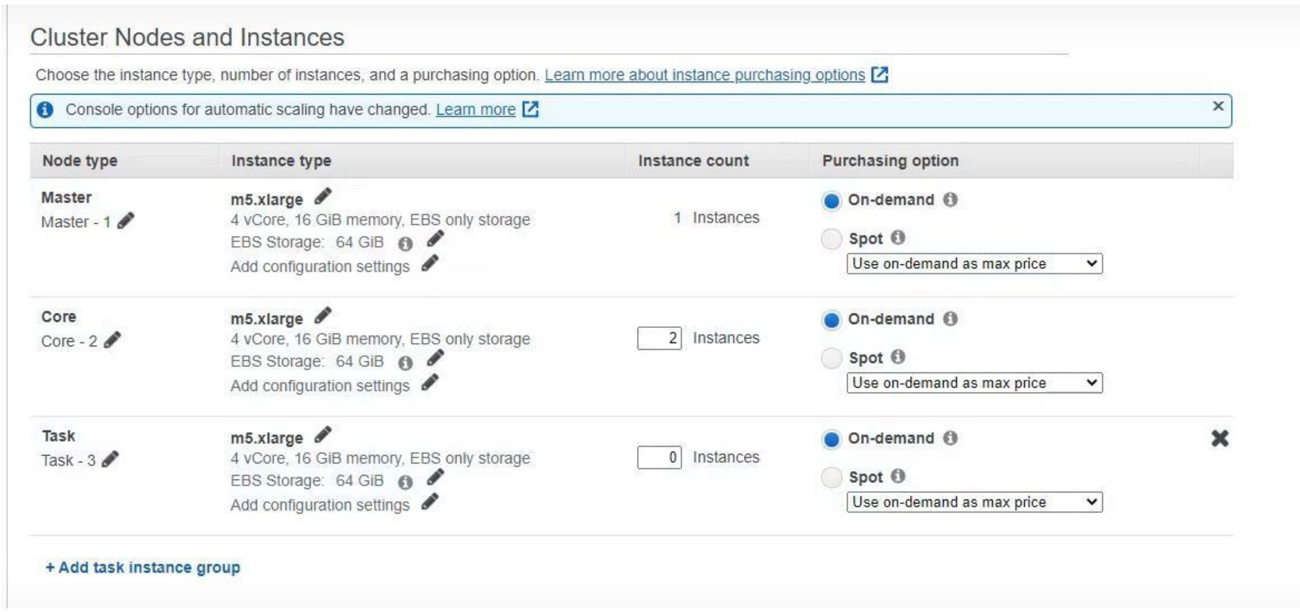
Configuring the Cluster Manually

Software configuration

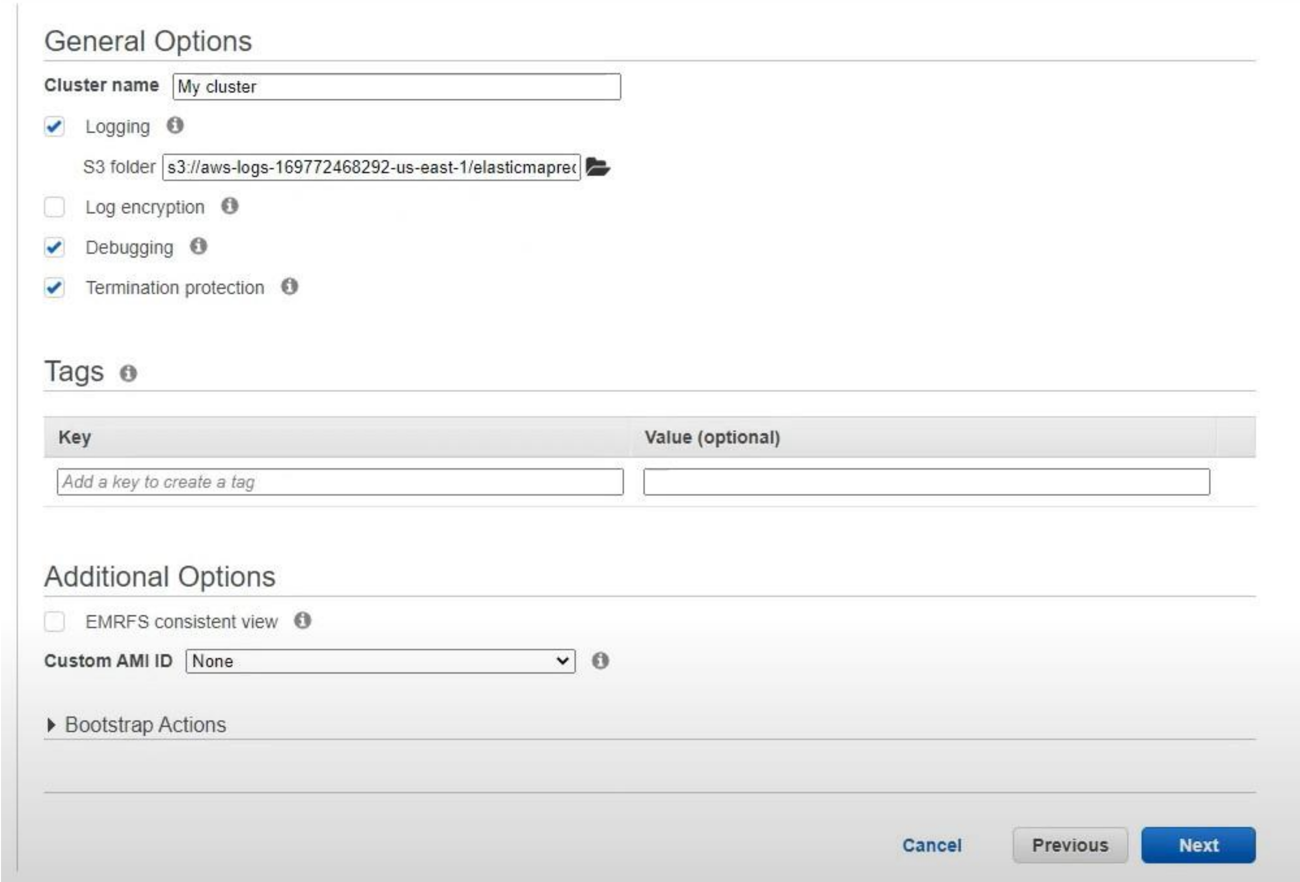


Hardware Configuration

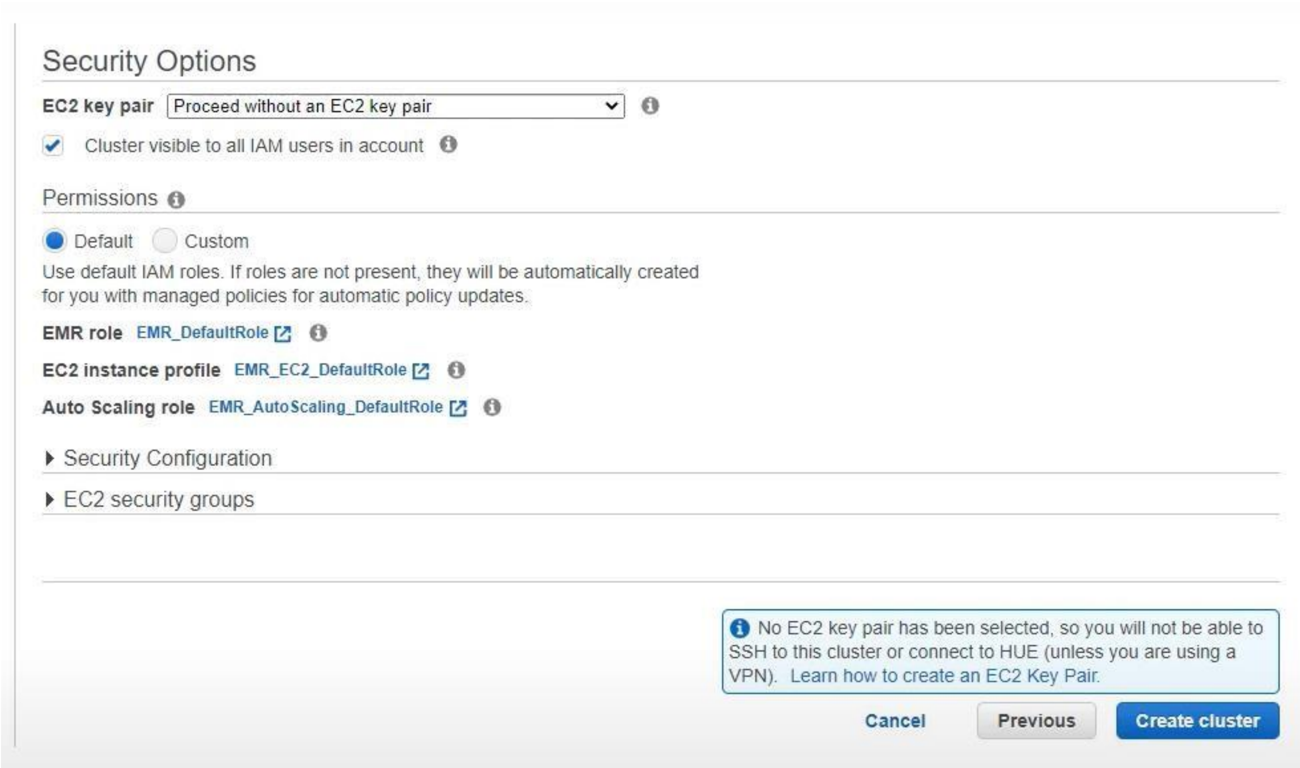




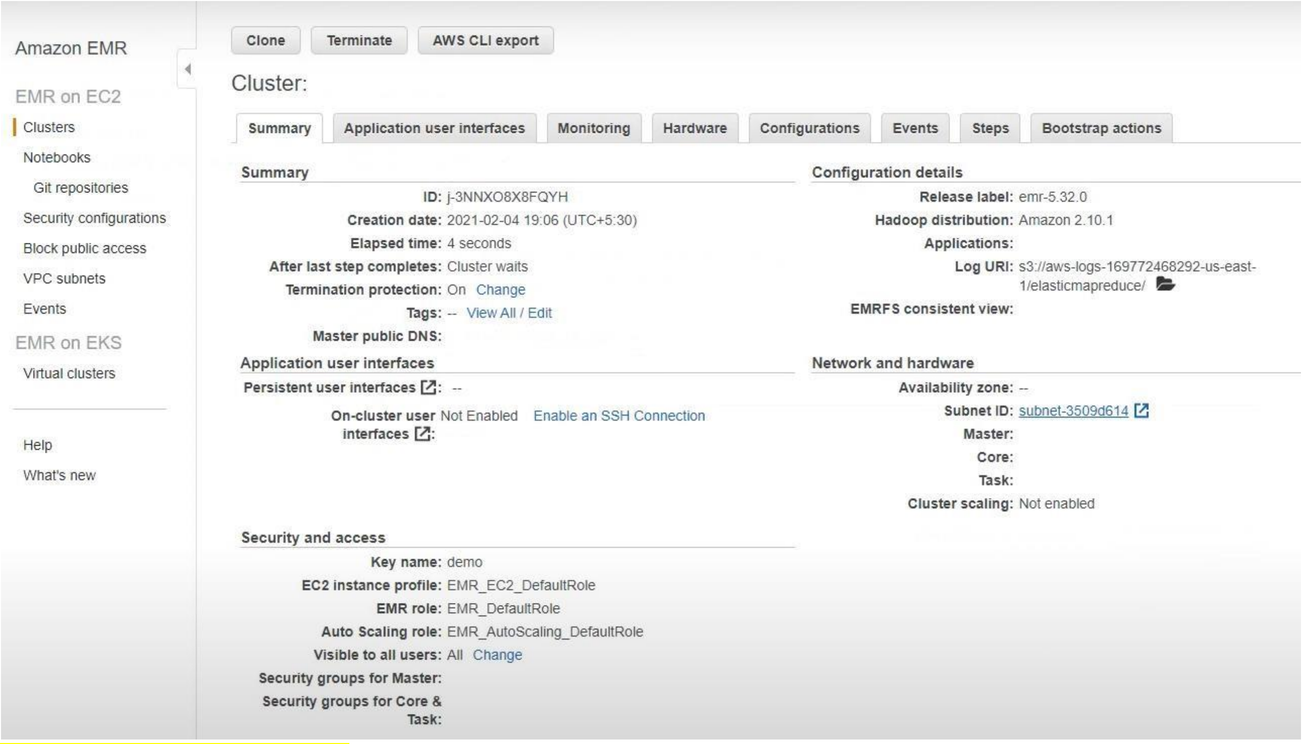
Setting



Security



Summary of the Cluster



Amazon EMR Console

So we can directly use different big data framework on the AWS Cloud using Amazon EMR.

Hive for the Data Storage

```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
18 package(s) needed for security, out of 61 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-91-246 ~]$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
```

Spark for the Analysis of the Big Data.

```
[hadoop@ip-172-31-91-246 ~]$ spark-shell
21/02/04 13:41:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform
e applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://ip-172-31-91-246.ec2.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1612446083842).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.7-amzn-0
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_272)
Type in expressions to have them evaluated.
Type :help for more information.
```