Contents

[1 Introduction of Project 3](#_Toc499998818)

[1.1 Project Description 3](#_Toc499998819)

[1.2 Data Set Description 3](#_Toc499998820)

[2 Creation of Cloud Cluster on AWS EC2 4](#_Toc499998821)

[2.1 Three instances and configuration 4](#_Toc499998822)

[2.2 SSH to remote Linux 7](#_Toc499998823)

[3 Software Installation and Configuration 9](#_Toc499998824)

[3.1 Upload Software to Remote Machine 9](#_Toc499998825)

[3.2 JDK Installation and Configuration 10](#_Toc499998826)

[3.3 Hadoop Installation and Configuration 11](#_Toc499998827)

[4 Mapping between Host Name and IP 14](#_Toc499998828)

[5 SSH Configuration between Remote Machine and Synchronization 15](#_Toc499998829)

[5.1 SSH Configuration 15](#_Toc499998830)

[5.2 Machine Synchronization 15](#_Toc499998831)

[6 Start Hadoop and Check Status 17](#_Toc499998832)

[7 Parallel Programming for Association Mining 20](#_Toc499998833)

[Appendix Java File 24](#_Toc499998834)

# Introduction of Project

## Project Description

Working Environment: AWS (Amazon Web Service) EC2 + Hadoop + MapReduce

Mining Algorithm: Association Rules Mining based on Apriori

Programming Language: Java

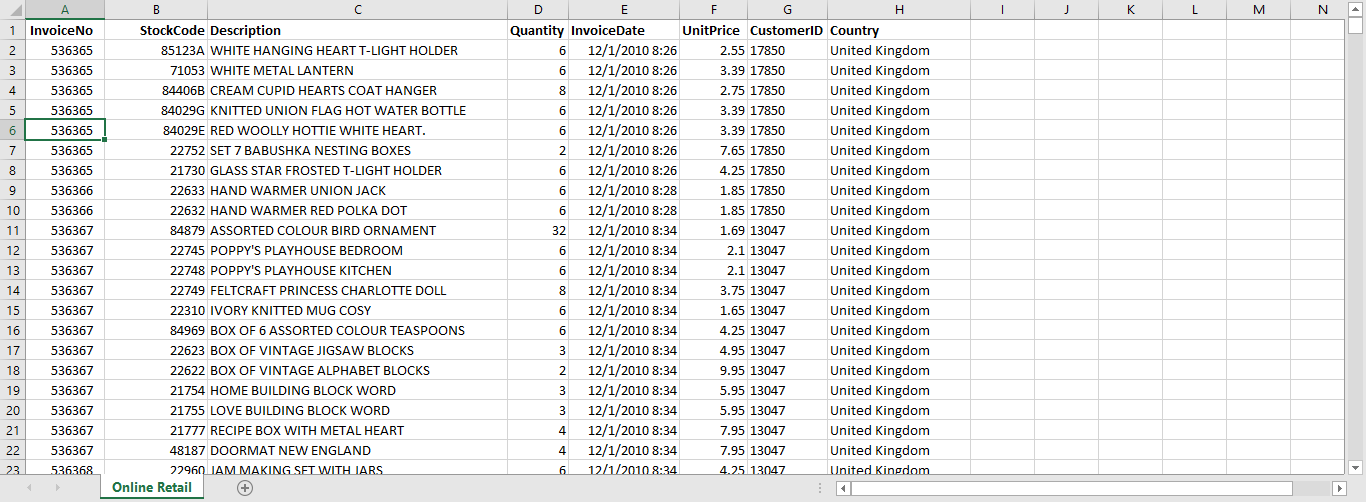
Operating System: Windows 10

Tools Software: PuTTY for SSH connection, WinSCP for uploading file from Windows to remote Linux

## Data Set Description

Data Set: Online-Retail.xlsx, <http://archive.ics.uci.edu/ml/machine-learning-databases/00352/>

This is a data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail store. The company mainly sells unique all-occasion gifts. Below are data details:



**Attribute Information (from the data set website):**

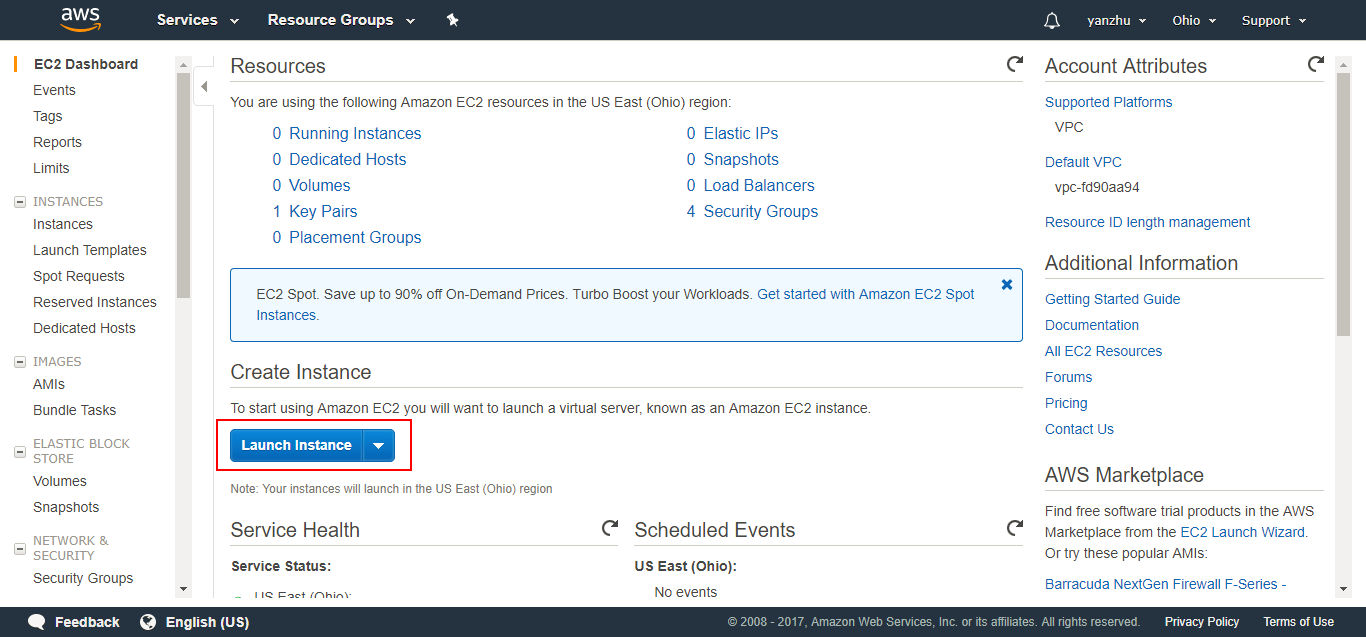
InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.   
StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.   
Description: Product (item) name. Nominal.   
Quantity: The quantities of each product (item) per transaction. Numeric.   
InvoiceDate: Invoice Date and time. Numeric, the day and time when each transaction was generated.   
UnitPrice: Unit price. Numeric, Product price per unit in sterling.   
CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.   
Country: Country name. Nominal, the name of the country where each customer resides.

I mainly use “InvoiceNo” and “StockCode” in my project.

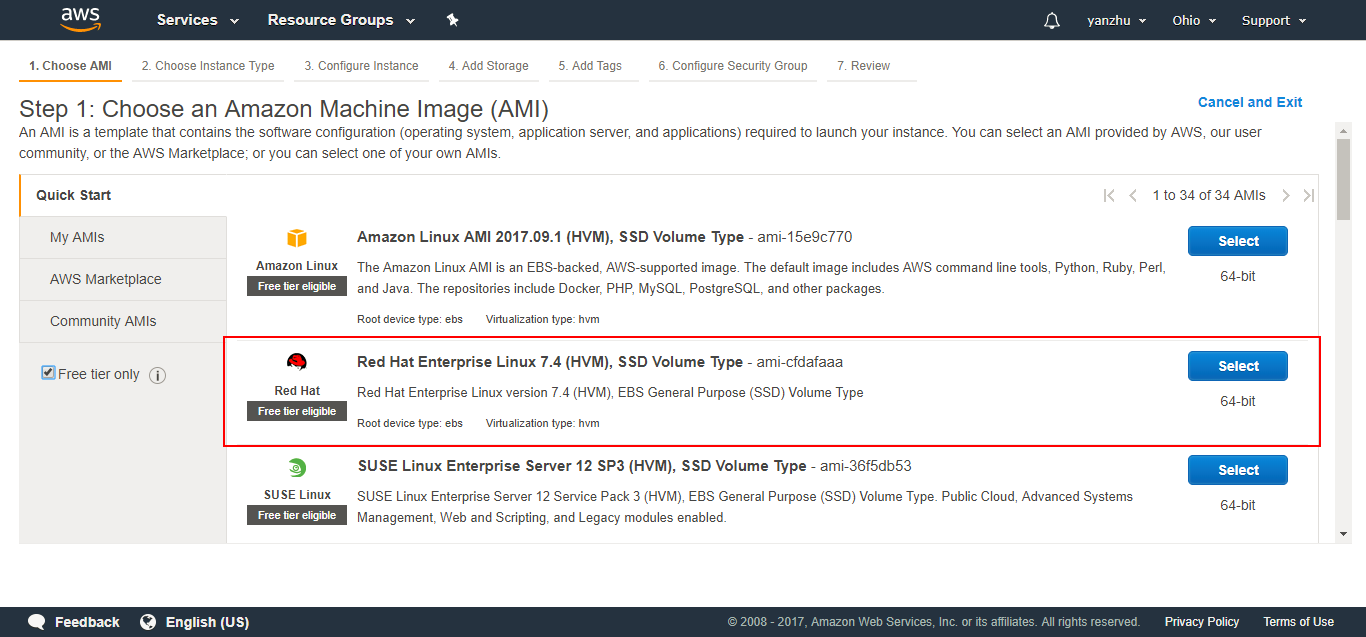
# Creation of Cloud Cluster on AWS EC2

## Three instances and configuration

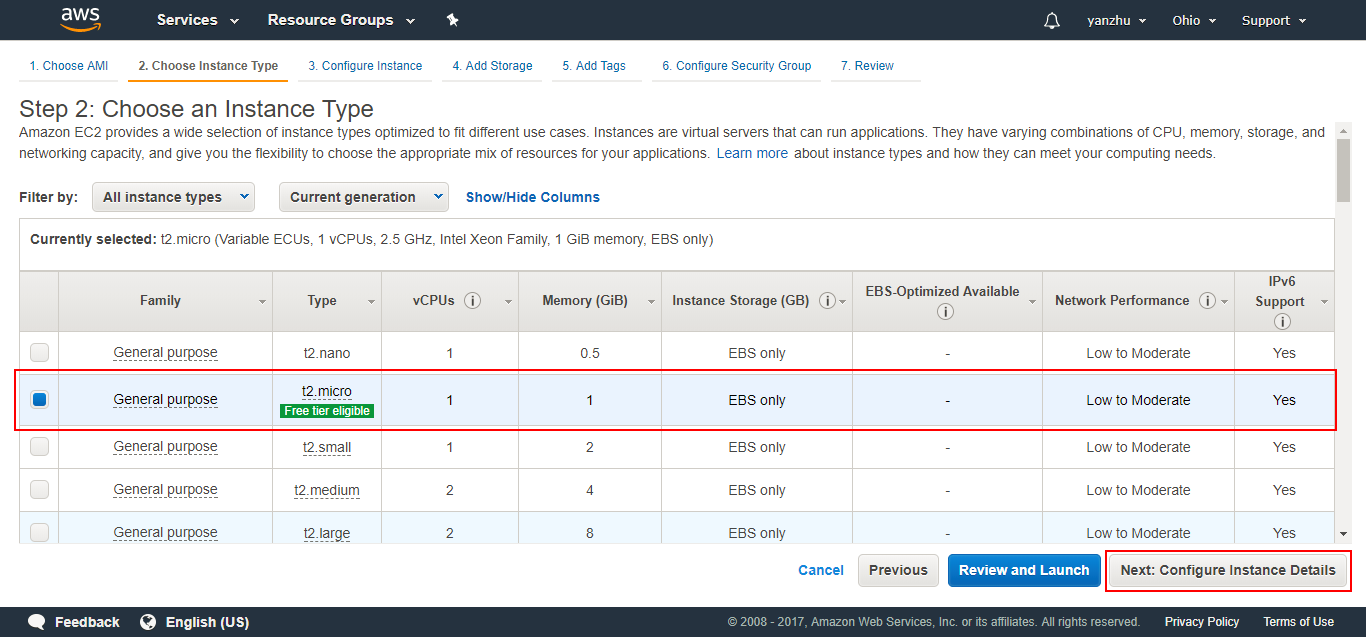
Go to EC2 Dashboard and click “Launch Instance”.



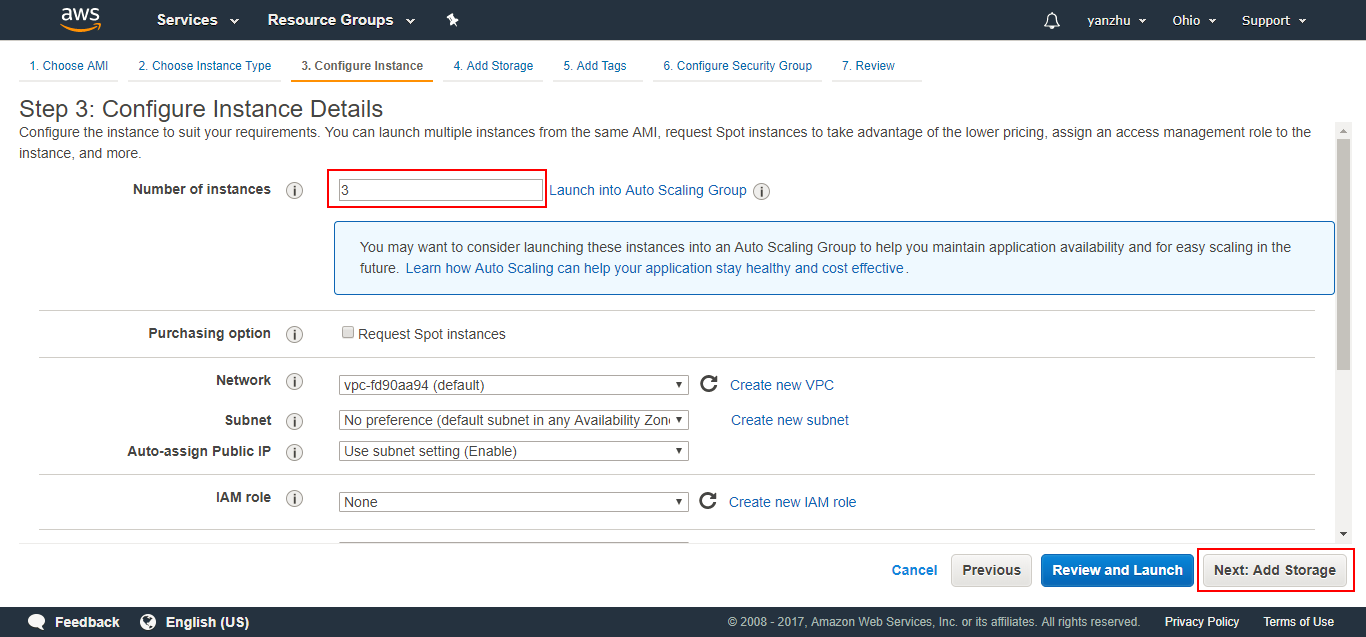
Choose the instance type, here I use RHEL (Red Hat Enterprise Linux 7.4) for my project:



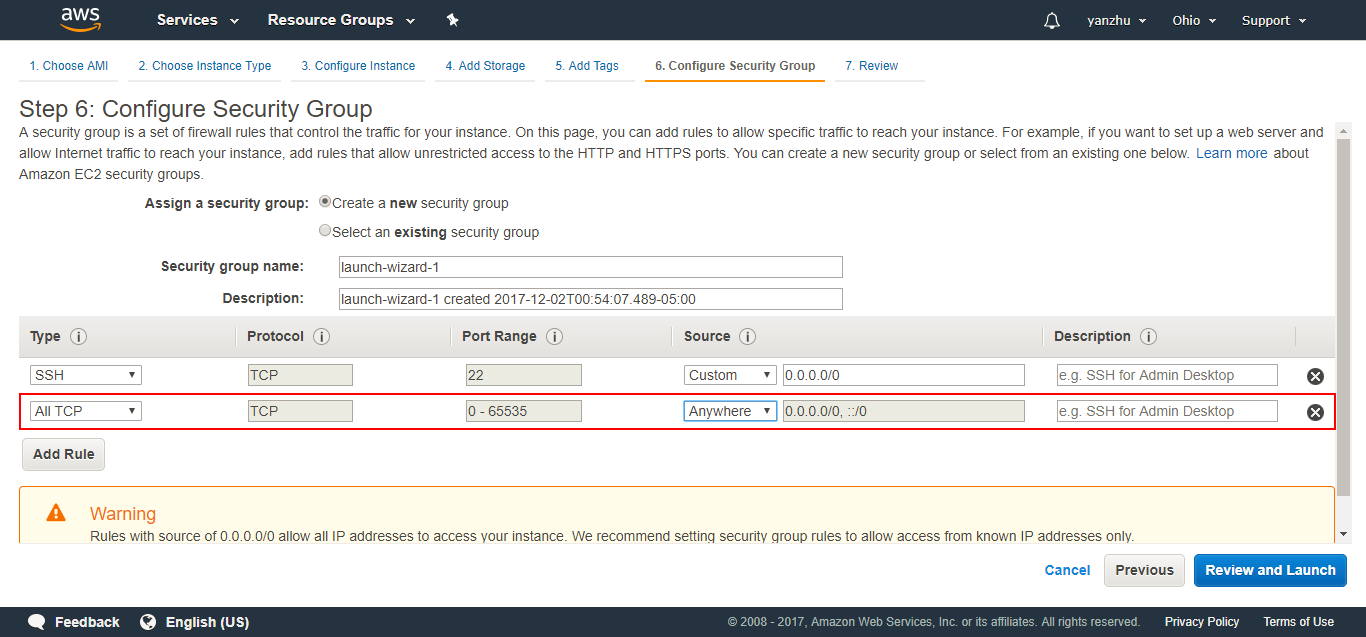
For my project, I can choose default resource for created instances, and then click “Next”:



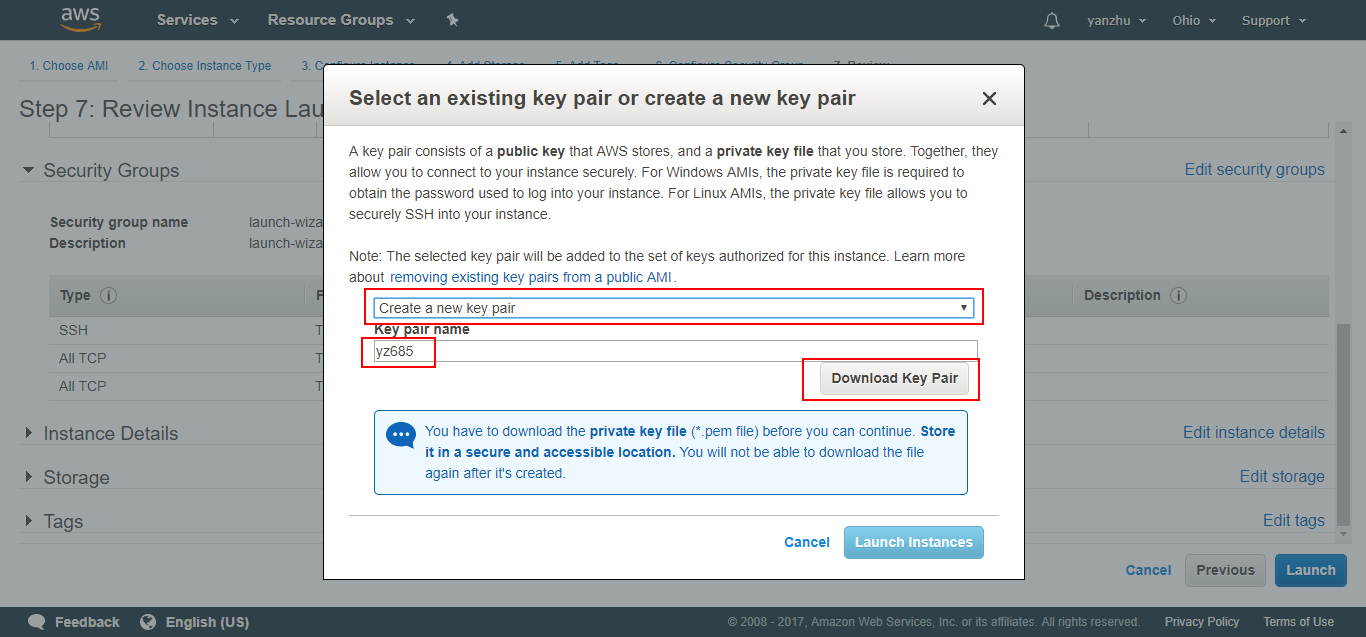
“Number of instances” is 3, then click “Next”:



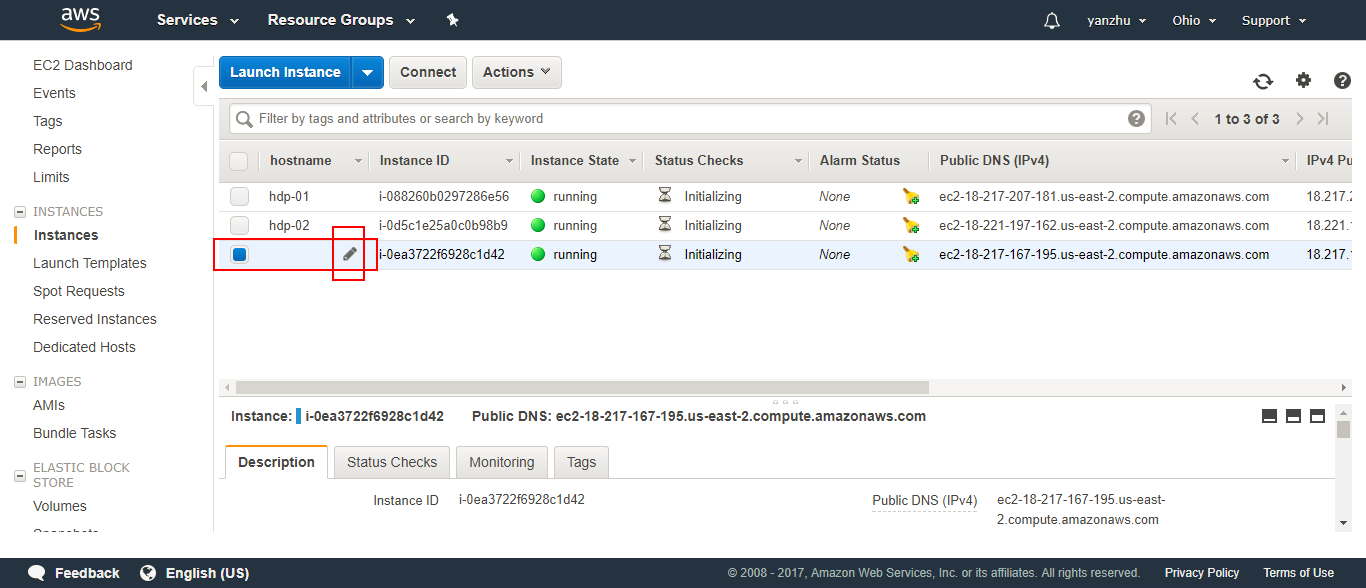
Choose default storage for each instance and click “Next”. At next page, I don’t need to add tags, so directly click “Next”. In “Configure Security Group” step, add new rules for using, because I need to use some TCP ports for Hadoop and other software, so here I open all ports just for my project:



For SSH connection, I need to create and download a key to local machine, and then click “Launch Instance”:



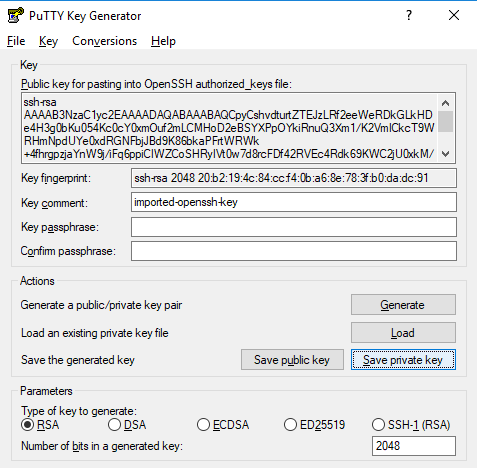
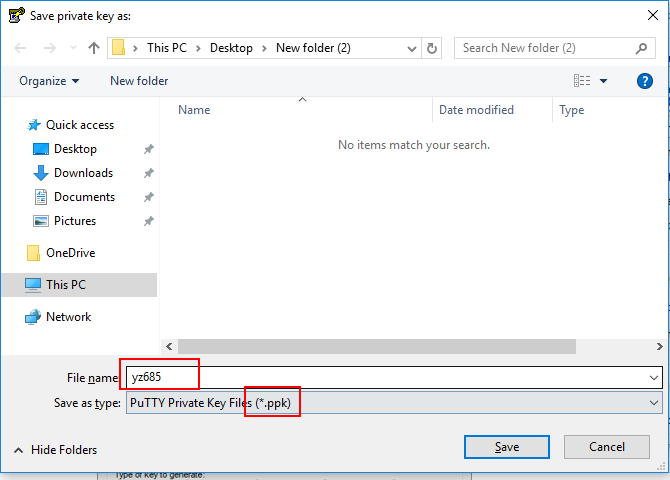
Optional operation: for convenience, I name hostname for three instances, they are hdp-01, hdp-02, hdp-03.



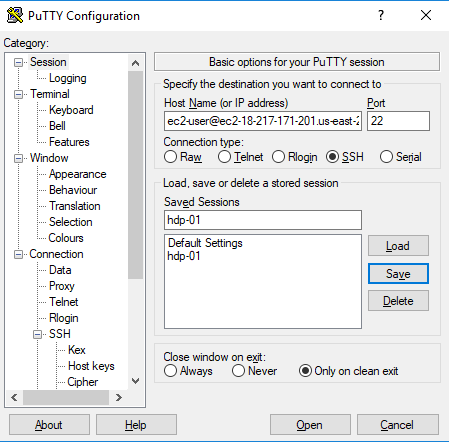
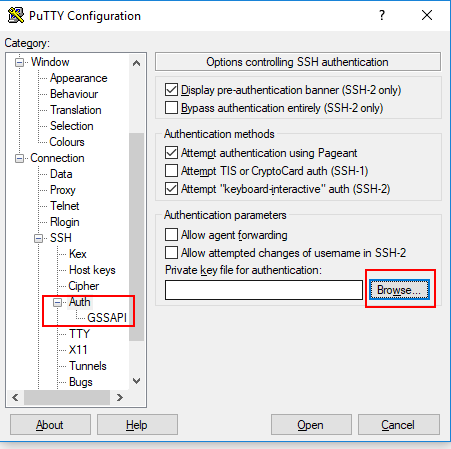
## SSH to remote Linux

Download PuTTY for windows, and then install it.

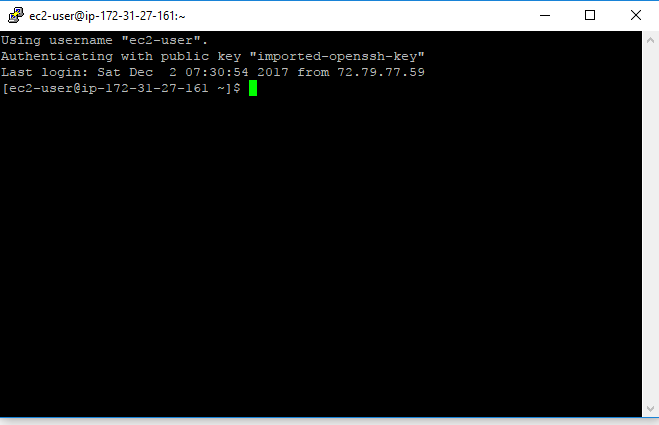
Because this software does not support .pem (from AWS), I will use PuTTYgen (a component of PuTTY) to convert .pem to .ppk, the method is like below, load .pem and save private key as .ppk :

The ppk file should has the same name as pem file. After this, launch PuTTY, and then build connection, first to add key, then set hostname, sava session for fast next using:

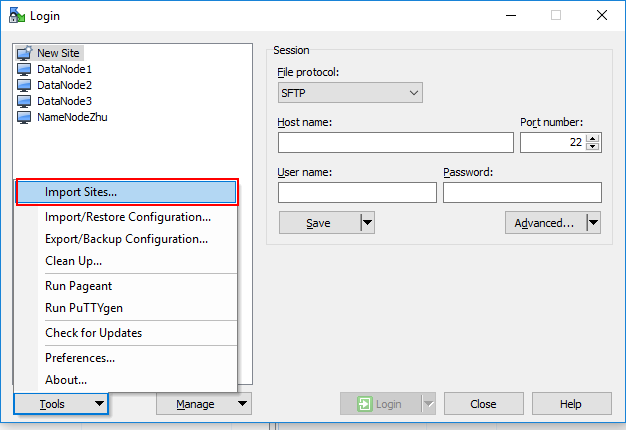
Now I can access remote Linux:



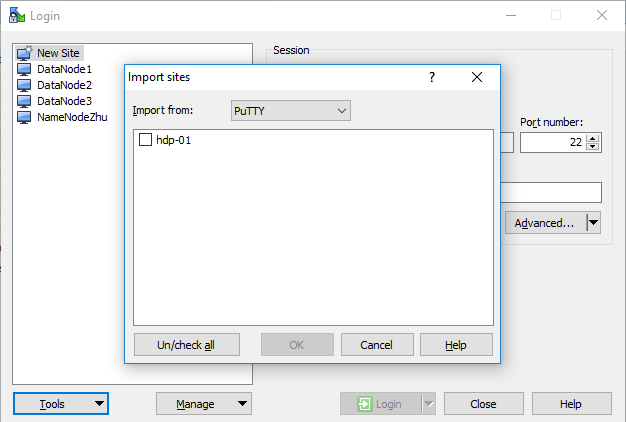
# Software Installation and Configuration

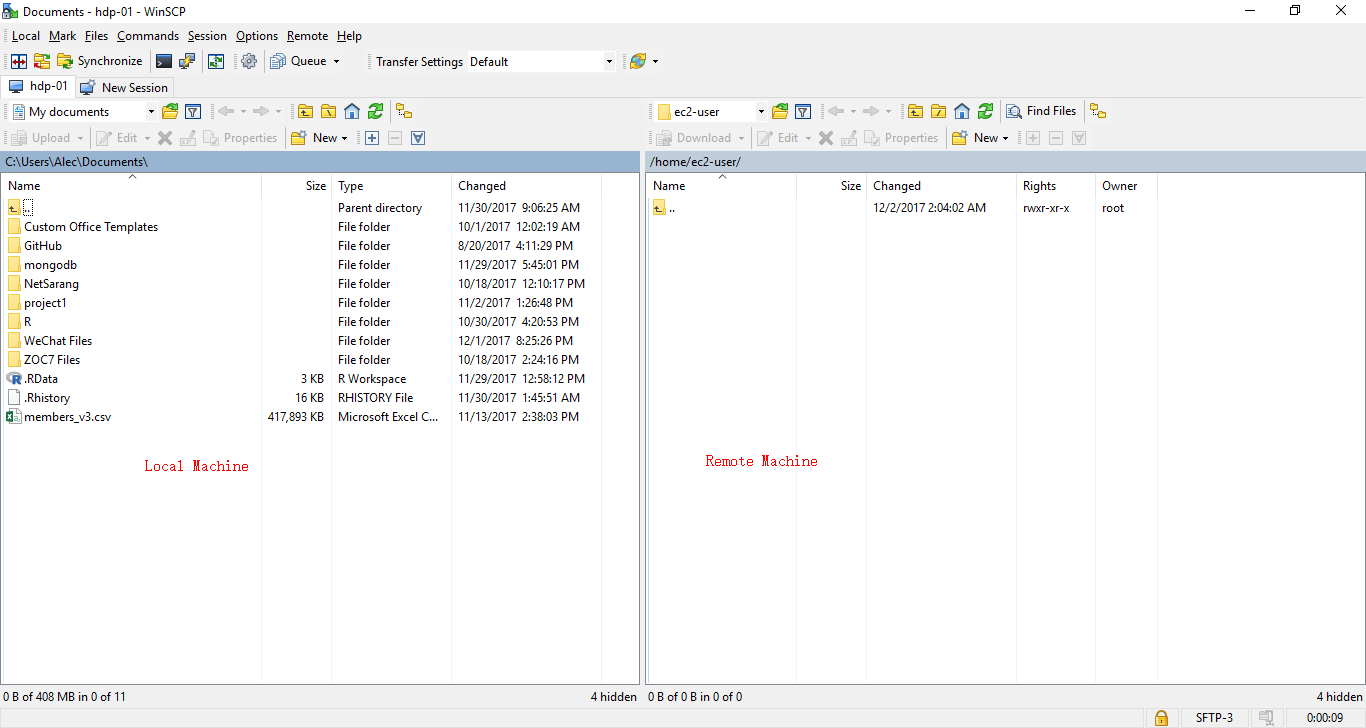
## Upload Software to Remote Machine

I upload software to remote Linux by using WinSCP on my operating system:



Because I have created connection at above step to SSH virtual host, now WinSCP can recognizes it and automatically connect to remote machine, click “Login”:





## JDK Installation and Configuration

Download jdk-8u151-linux-x64.tar.gz to local machine and then upload it to hdp-01 and run commands below:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ mkdir apps  [ec2-user@ip-172-31-27-161 ~]$ tar -zxvf jdk-8u151-linux-x64.tar.gz -C apps/  [ec2-user@ip-172-31-27-161 ~]$ sudo vi /etc/profile |

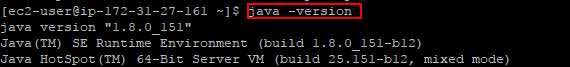
Add below to end of file:

|  |
| --- |
| export JAVA\_HOME=/home/ec2-user/apps/jdk1.8.0\_151  export PATH=$PATH:$JAVA\_HOME/bin |

Then get the configuration work:

|  |
| --- |
| source /etc/profile |

Check if I have install it successfully:



## Hadoop Installation and Configuration

* Download hadoop-2.8.2.tar.gz and upload it to remote machine, execute commands below:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ tar -zxvf hadoop-2.8.2.tar.gz -C apps/ |

Configure environment variable for Hadoop:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ cd /home/ec2-user/apps/hadoop-2.8.2/etc/hadoop/  [ec2-user@ip-172-31-27-161 hadoop]$ vi hadoop-env.sh |

Change file like below:

|  |
| --- |
| # The java implementation to use.  export JAVA\_HOME=/home/ec2-user/apps/jdk1.8.0\_151 |

* Specify Hadoop name node (Master node) by editing “core-site.xml”:

|  |
| --- |
| <configuration>  <property>  <name>fs.defaultFS</name>  <value>hdfs://hdp-01:9000</value>  </property>  </configuration> |

* Specify name node and data node storing directory by editing “hdfs-site.xml”:

|  |
| --- |
| <configuration>  <property>  <name>dfs.namenode.name.dir</name>  <value>/home/ec2-user/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>/home/ec2-user/dfs/data</value>  </property>  </configuration> |

* Specify resource allocation node by editing “yarn-site.xml”

|  |
| --- |
| <configuration>  <!-- Site specific YARN configuration properties -->  <property>  <name>yarn.resourcemanager.hostname</name>  <value>hdp-01</value>  </property>  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  </configuration> |

Explanation: resource manager is to allocate resource, node manager is to apply for resource to do computing. node manager is not necessarily same as data node.

Normally, data node is together with node manager to do computing quickly, when use bash command, like start-yarn.sh to launch all node manager, /etc/slaves in Hadoop is to decide which one can work as node manager.

* Specify MapReduce framework by editing “mapred-site.xml”

|  |
| --- |
| <configuration>  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  </configuration> |

* Specify slave nodes by editing “slaves”

|  |
| --- |
| localhost  hdp-02  hdp-03 |

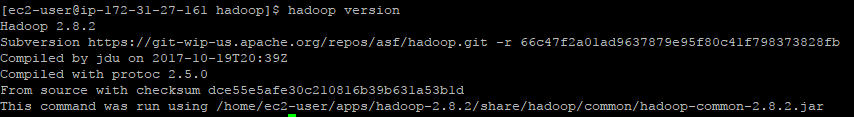
* Configure system environment variable for Hadoop:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 hadoop]$ sudo vi /etc/profile |

Then edit file like below:

|  |
| --- |
| export HADOOP\_HOME=/home/ec2-user/apps/hadoop-2.8.2  export PATH=$PATH:$JAVA\_HOME/bin:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin |

Source /etc/profile and input “hadoop version”, it is shown like below:



# Mapping between Host Name and IP

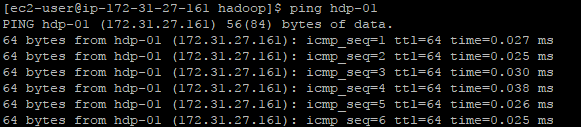
Execute command:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 hadoop]$ sudo vi /etc/hosts |

Edit file like below:

|  |
| --- |
| 172.31.27.161 hdp-01  172.31.27.228 hdp-02  172.31.24.48 hdp-03 |

Now I can use host name to label a machine:



# SSH Configuration between Remote Machine and Synchronization

## SSH Configuration

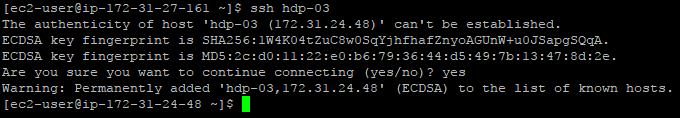
When launch instance, AWS creates same public key and write it into /home/ec2-user/.ssh/aauthorized\_keys, and I saved its paired private key to my machine. So now I use this key pair to connect my local PuTTY to remote AWS VM, certainly I can also use this pair to completely connect each VM through SSH, then we can use scp or other communication command.

I can use scp -i “key name” specify using private key, or upload private key to remote VM, and then mv “private key” /home/ec2-user/.ssh/id\_rsa, do not forget to rename it as id\_rsa.

Execute commands as below:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ mv yz685.pem /home/ec2-user/.ssh/id\_rsa  [ec2-user@ip-172-31-27-161 ~]$ chmod 400 /home/ec2-user/.ssh/id\_rsa |

Now I can access every virtual machine by ssh, use “exit” to return original machine:



## Machine Synchronization

* Synchronize mapping between host name and IP. because /etc/hosts need root auth, so we need to scp hosts to a temp file to remote machine:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ scp /etc/hosts hdp-02:~/temp |

Then SSH to hdp-02 and execute command:

|  |
| --- |
| [ec2-user@ip-172-31-27-228 ~]$ sudo mv temp /etc/hosts |

Run the same operation on hdp-03, now three host name mapping is completed.

* Synchronize profile on all three machines:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ scp /etc/profile hdp-02:~/temp |

Then SSH to hdp-02 and execute command:

|  |
| --- |
| [ec2-user@ip-172-31-27-228 ~]$ sudo mv temp /etc/profile |

Run the same operation on hdp-03, now three host name mapping is completed.

* Synchronize private key

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ scp /home/ec2-user/.ssh/id\_rsa hdp-02:/home/ec2-user/.ssh/id\_rsa  [ec2-user@ip-172-31-27-161 ~]$ scp /home/ec2-user/.ssh/id\_rsa hdp-03:/home/ec2-user/.ssh/id\_rsa |

Now three machine can access each other by SSH.

* Synchronize software

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ scp -r apps/ hdp-02:~/  [ec2-user@ip-172-31-27-161 ~]$ scp -r apps/ hdp-03:~/ |

# Start Hadoop and Check Status

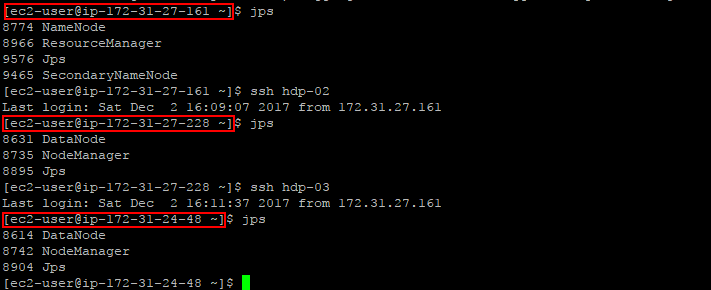
* Format name node for one time, input command:

|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ hadoop namenode -format |

* Restart all nodes and input in hdp-01 (name node):

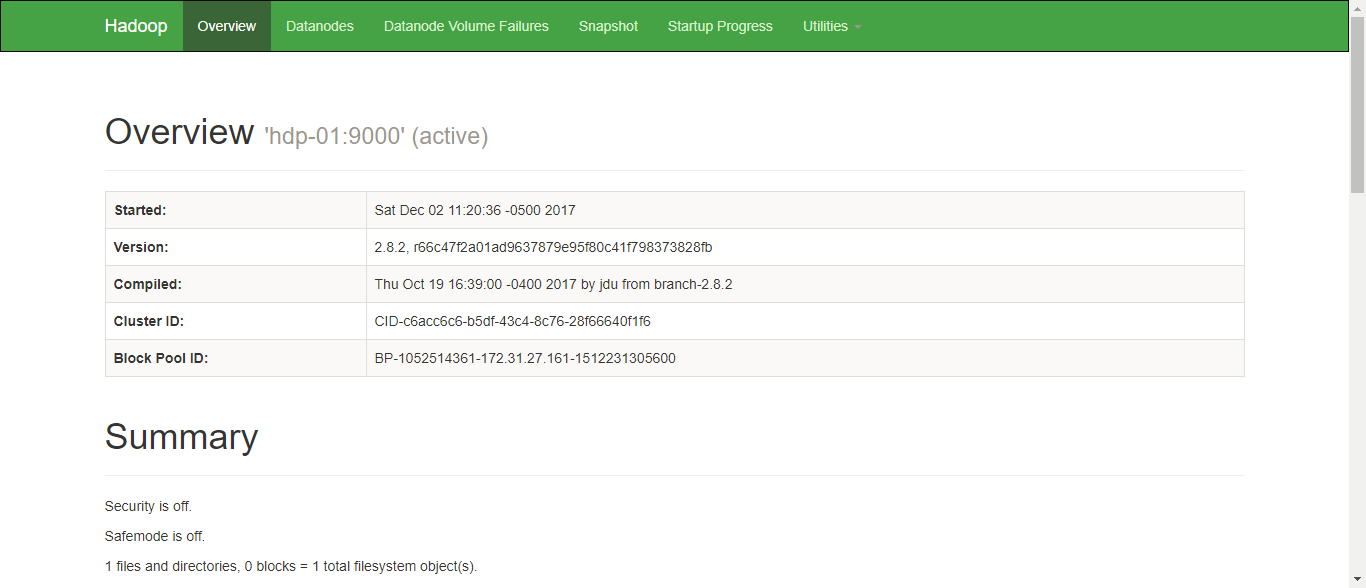
|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ start-dfs.sh  [ec2-user@ip-172-31-27-161 ~]$ start-yarn.sh |

* Check Hadoop cluster status by command, it is shown below:



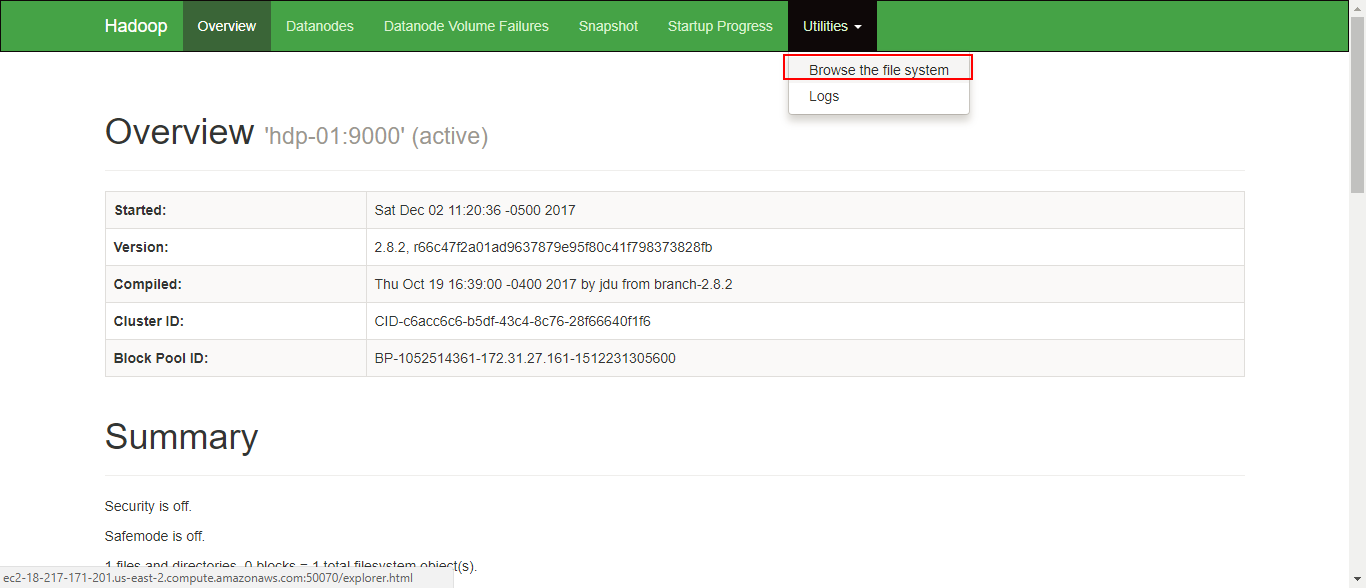
* Check Hadoop cluster status by AWS website:

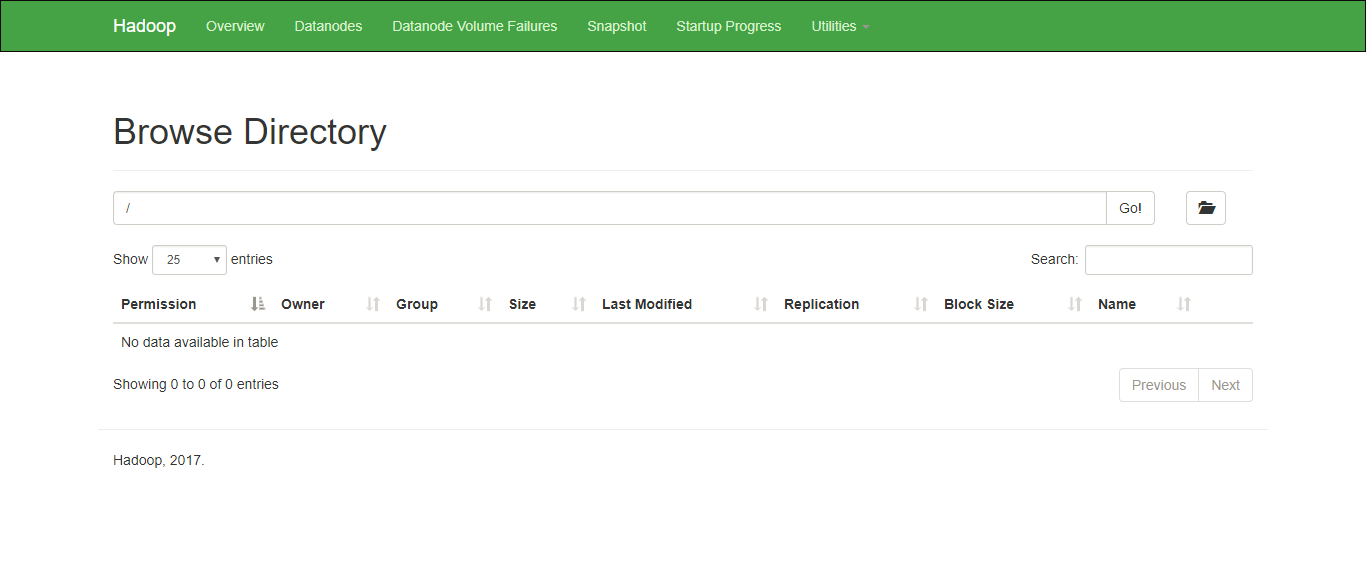
1. http://ec2-18-217-171-201.us-east-2.compute.amazonaws.com:50070



I can further check all nodes information and hdfs status:

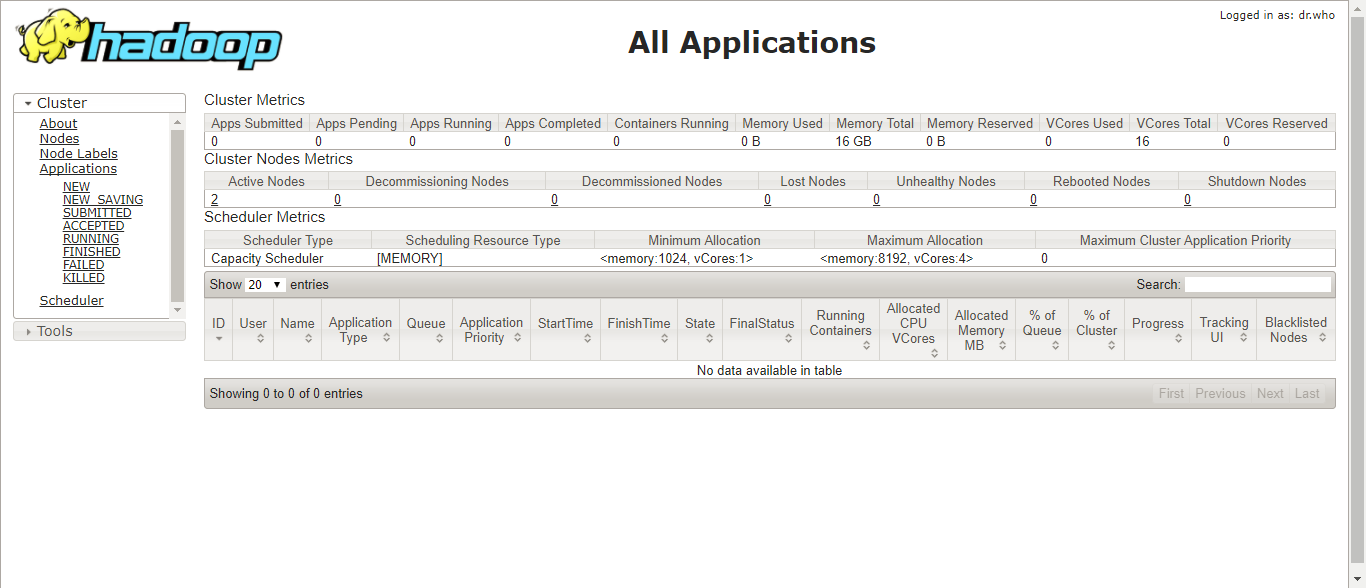




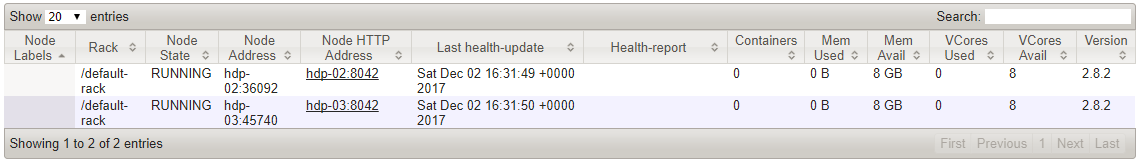


1. http://ec2-18-217-171-201.us-east-2.compute.amazonaws.com:8088/cluster

Below figure is yarn status:



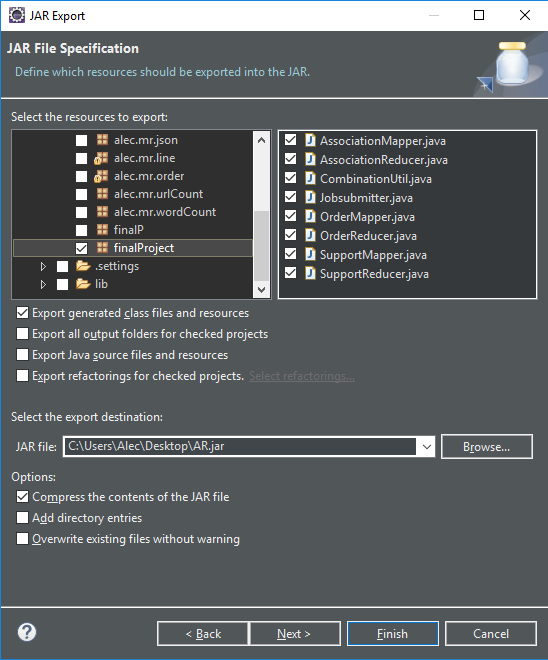
2 Active nodes:



# Parallel Programming for Association Mining

* Export jar file

Eclipse – export – JAR file, and then select which package I need to export as below:



* java file:

Jobsubmitter.java (main file)

CombinationUtil.java (tools class )

OrderMapper.java (combine original data set)

OrderReducer.java (combine original data set)

SupportMapper.java (calculate frequent set)

SupportReducer.java (calculate frequent set)

AssociationMapper.java (calculate association rules)

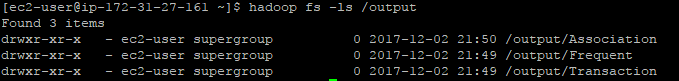
AssociationReducer.java (calculate association rules)

* Upload data set and jar file to hdp-01 and execute commands in hdp-01:

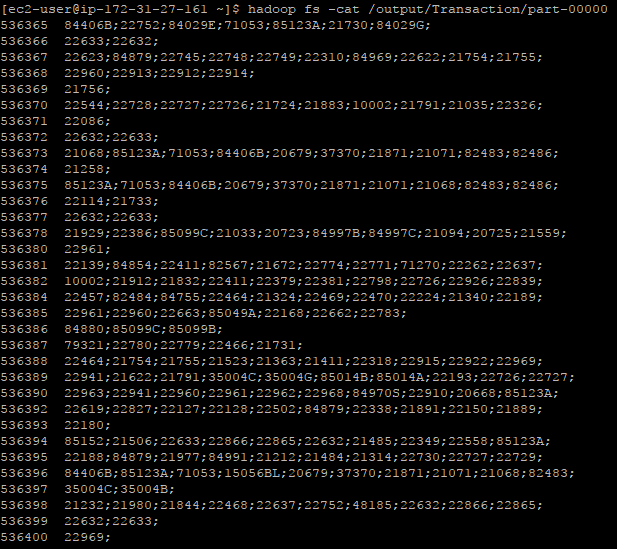
|  |
| --- |
| [ec2-user@ip-172-31-24-48 ~]$ hadoop fs -ls /  [ec2-user@ip-172-31-24-48 ~]$ hadoop fs -mkdir -p /AssociationRules/input  [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -put Online-Retail.csv /AssociationRules/input  [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -ls /AssociationRules/input  [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -rm -r /output  [ec2-user@ip-172-31-27-161 ~]$ hadoop jar AR.jar finalProject.Jobsubmitter /AssociationRules/input /output 500 10 0.05 0.2 |

* Check part results by using commands below:

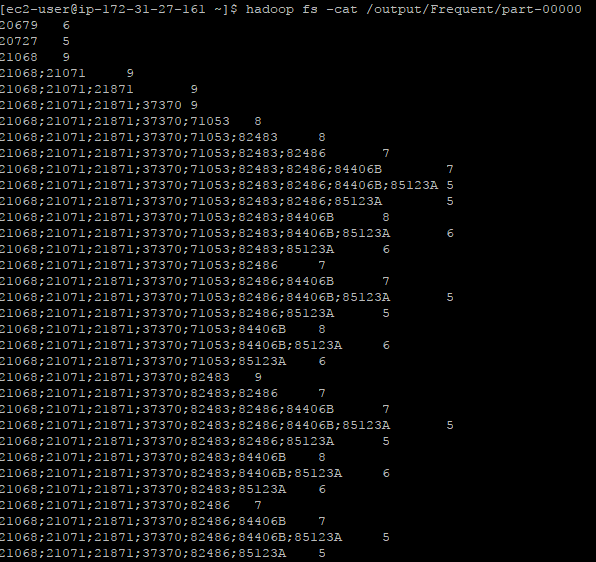
|  |
| --- |
| [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -ls /  [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -ls /output  [ec2-user@ip-172-31-27-161 ~]$ hadoop fs -cat /output/Transaction/part-00000 |



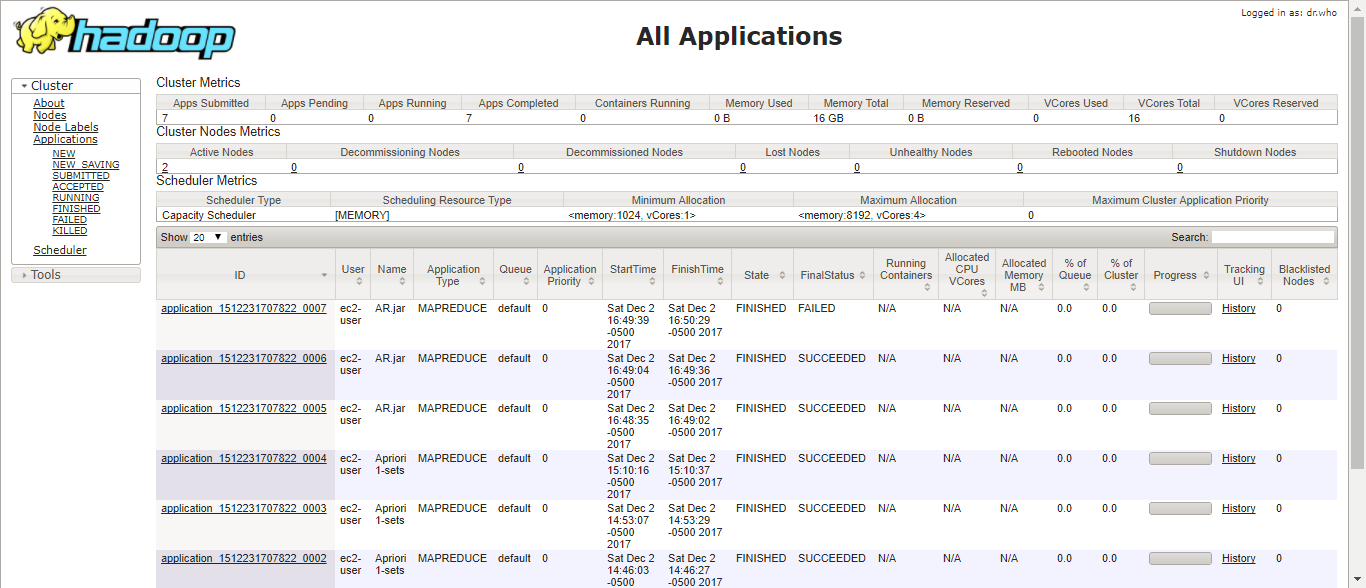
Combined order information:



Frequent set:



I can also check yarn status through website:



# Appendix Java File

* Jobsubmitter.java (main file)

|  |
| --- |
| package finalProject;  import org.apache.hadoop.conf.Configured;  import org.apache.hadoop.fs.FileStatus;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.FileInputFormat;  import org.apache.hadoop.mapred.JobConf;  import org.apache.hadoop.mapred.KeyValueTextInputFormat;  import org.apache.hadoop.mapred.RunningJob;  import org.apache.hadoop.mapred.FileOutputFormat;  import org.apache.hadoop.mapred.JobClient;  import org.apache.hadoop.util.Tool;  import org.apache.hadoop.util.ToolRunner;  /\*\*  \* @author Alec Main class for job submit  \*/  public class Jobsubmitter extends Configured implements Tool {  public static String SEPARATOR = ";";  public static void main(String[] args) throws Exception {  int res = ToolRunner.run(new Jobsubmitter(), args);  System.exit(res);  }  public int run(String[] args) throws Exception {  String INPUT\_DIR = args[0];  String OUTPUT\_DIR = args[1];  int totalTransaction = Integer.parseInt(args[2]);  int maxItemNum = Integer.parseInt(args[3]);  float minSupport = Float.parseFloat(args[4]);  float minConfidence = Float.parseFloat(args[5]);  JobConf cnfg = new JobConf(Jobsubmitter.class);  FileSystem fs = FileSystem.get(cnfg);  FileStatus[] statesList = fs.listStatus(new Path(INPUT\_DIR));  if (statesList != null) {  for (FileStatus statesFile : statesList) {  FileInputFormat.addInputPath(cnfg, statesFile.getPath());  }  }  FileOutputFormat.setOutputPath(cnfg, new Path(OUTPUT\_DIR + "/Transaction"));  cnfg.setInt("totalTransaction", totalTransaction);  cnfg.setInt("maxItemNum", maxItemNum);  cnfg.setFloat("minSupport", minSupport);  cnfg.setFloat("minConfidence", minConfidence);  cnfg.setMapperClass(OrderMapper.class);  cnfg.setReducerClass(OrderReducer.class);  cnfg.setMapOutputKeyClass(Text.class);  cnfg.setMapOutputValueClass(Text.class);  cnfg.setOutputKeyClass(Text.class);  cnfg.setOutputValueClass(Text.class);  RunningJob job = JobClient.runJob(cnfg);  job.waitForCompletion();  if (job.isSuccessful()) {  JobConf cnfgSupport = new JobConf(Jobsubmitter.class);  fs = FileSystem.get(cnfgSupport);  statesList = fs.listStatus(new Path(OUTPUT\_DIR + "/Transaction"));  if (statesList != null) {  for (FileStatus status : statesList) {  if (status.getLen() > 0) {  FileInputFormat.addInputPath(cnfgSupport, status.getPath());  }  }  }  FileOutputFormat.setOutputPath(cnfgSupport, new Path(OUTPUT\_DIR + "/Frequent"));  cnfgSupport.setInt("totalTransaction", totalTransaction);  cnfgSupport.setInt("maxItemNum", maxItemNum);  cnfgSupport.setFloat("minSupport", minSupport);  cnfgSupport.setFloat("minConfidence", minConfidence);  cnfgSupport.setMapperClass(SupportMapper.class);  cnfgSupport.setReducerClass(SupportReducer.class);  cnfgSupport.setInputFormat(KeyValueTextInputFormat.class);  cnfgSupport.setMapOutputKeyClass(Text.class);  cnfgSupport.setMapOutputValueClass(IntWritable.class);  cnfgSupport.setOutputKeyClass(Text.class);  cnfgSupport.setOutputValueClass(IntWritable.class);  job = JobClient.runJob(cnfgSupport);  job.waitForCompletion();  }  if (job.isSuccessful()) {  JobConf cnfgAssociation = new JobConf(Jobsubmitter.class);  fs = FileSystem.get(cnfgAssociation);  statesList = fs.listStatus(new Path(OUTPUT\_DIR + "/Frequent"));  if (statesList != null) {  for (FileStatus status : statesList) {  if (status.getLen() > 0) {  FileInputFormat.addInputPath(cnfgAssociation, status.getPath());  }  }  }  FileOutputFormat.setOutputPath(cnfgAssociation, new Path(OUTPUT\_DIR + "/Association"));  cnfgAssociation.setInt("totalTransaction", totalTransaction);  cnfgAssociation.setInt("maxItemNum", maxItemNum);  cnfgAssociation.setFloat("minSupport", minSupport);  cnfgAssociation.setFloat("minConfidence", minConfidence);  cnfgAssociation.setMapperClass(AssociationMapper.class);  cnfgAssociation.setReducerClass(AssociationReducer.class);  cnfgAssociation.setInputFormat(KeyValueTextInputFormat.class);  cnfgAssociation.setMapOutputKeyClass(Text.class);  cnfgAssociation.setMapOutputValueClass(IntWritable.class);  cnfgAssociation.setOutputKeyClass(Text.class);  cnfgAssociation.setOutputValueClass(Text.class);  job = JobClient.runJob(cnfgAssociation);  job.waitForCompletion();  }  return 0;  }  } |

* CombinationUtil.java

|  |
| --- |
| package finalProject;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author Alec  \*  \* utility class for combining result data set  \*  \*/  public class CombinationUtil {  private List<String> combinationStrArr;  public CombinationUtil() {  this.combinationStrArr = new ArrayList<String>();  }  public List<String> getCombination(String[] dataList, int n) {  for (int j = n; j <= dataList.length; j++) {  genCombination(dataList, 0, new String[j], 0);  }  return this.combinationStrArr;  }  public List<String> getCombination(List<String> dataList, int n) {  for (int j = n; j <= dataList.size(); j++) {  genCombination(dataList, 0, new String[j], 0);  }  return this.combinationStrArr;  }  private void genCombination(String[] dataList, int dataIndex, String[] resultList, int resultIndex) {  int resultLen = resultList.length;  int resultCount = resultIndex + 1;  if (resultCount > resultLen) {  String resultStr = "";  for (int i = 0; i < resultList.length; i++) {  if (i == resultList.length - 1) {  resultStr += resultList[i];  } else {  resultStr += resultList[i] + Jobsubmitter.SEPARATOR;  }  }  combinationStrArr.add(resultStr);  return;  }  for (int i = dataIndex; i < dataList.length + resultCount - resultLen; i++) {  resultList[resultIndex] = dataList[i];  genCombination(dataList, i + 1, resultList, resultIndex + 1);  }  }  private void genCombination(List<String> dataList, int dataIndex, String[] resultList, int resultIndex) {  int resultLen = resultList.length;  int resultCount = resultIndex + 1;  if (resultCount > resultLen) {  String resultStr = "";  for (int i = 0; i < resultList.length; i++) {  if (i == resultList.length - 1) {  resultStr += resultList[i];  } else {  resultStr += resultList[i] + Jobsubmitter.SEPARATOR;  }  }  combinationStrArr.add(resultStr);  return;  }  for (int i = dataIndex; i < dataList.size() + resultCount - resultLen; i++) {  resultList[resultIndex] = dataList.get(i);  genCombination(dataList, i + 1, resultList, resultIndex + 1);  }  }  } |

* OrderReducer.java

|  |
| --- |
| package finalProject;  import java.io.IOException;  import java.util.Iterator;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.JobConf;  import org.apache.hadoop.mapred.JobConfigurable;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reducer;  import org.apache.hadoop.mapred.Reporter;  /\*\*  \* @author Alec  \* combine all stock code with the same order (invoice number)  \*/  public class OrderReducer extends MapReduceBase implements Reducer<Text, Text, Text, Text>, JobConfigurable {  private int maxItemNum;    // get maxItemNum while loading main file  public void configure(JobConf conf) {  String s = conf.get("maxItemNum");  this.maxItemNum = Integer.parseInt(s);  }    // combine iterately  public void reduce(Text key, Iterator<Text> value, OutputCollector<Text, Text> output, Reporter r)  throws IOException {  String resultStr = "";  int i = 0;  while (value.hasNext()) {  // just compute orders with set max item number  if (i >= maxItemNum)  break;  resultStr += value.next().toString() + Jobsubmitter.SEPARATOR;  i++;  }  output.collect(key, new Text(resultStr));  }  } |

* OrderMapper.jar

|  |
| --- |
| package finalProject;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.Mapper;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reporter;  import java.io.IOException;  /\*\*  \* @author Alec  \*  \* split original records, key is invoice number and value is stock code  \*  \*/  public class OrderMapper extends MapReduceBase implements Mapper<Object, Text, Text, Text> {  public void map(Object key, Text value, OutputCollector<Text, Text> output, Reporter r) throws IOException {  String[] text = value.toString().split(",");  // avoid title  if (!text[0].equals("InvoiceNo") && text[1].length() != 0) {  output.collect(new Text(text[0]), new Text(text[1]));  }  }  } |

* SupportMapper.java

|  |
| --- |
| package finalProject;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.Mapper;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reporter;  import java.io.IOException;  import java.util.ArrayList;  import java.util.Collections;  public class SupportMapper extends MapReduceBase implements Mapper<Text, Text, Text, IntWritable> {  public void map(Text key, Text value, OutputCollector<Text, IntWritable> output, Reporter r) throws IOException {  String[] text = value.toString().split(Jobsubmitter.SEPARATOR);  ArrayList<String> itemList = new ArrayList<String>();  for (int i = 0; i < text.length; i++) {  // delete items duplication in one order  if (itemList.contains(text[i])) {  continue;  } else {  itemList.add(text[i]);  }  }  // order list  Collections.sort(itemList);  CombinationUtil combinationUtil = new CombinationUtil();  ArrayList<String> ItemSetList = (ArrayList<String>) combinationUtil.getCombination(itemList, 1);  for (int j = 0; j < ItemSetList.size(); j++) {  output.collect(new Text(ItemSetList.get(j)), new IntWritable(1));  }  }  } |

* SupportReducer.java

|  |
| --- |
| package finalProject;  import java.io.IOException;  import java.util.Iterator;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.JobConf;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reducer;  import org.apache.hadoop.mapred.Reporter;  public class SupportReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {  private int totalTransaction;  private float minSupport;  public void configure(JobConf conf) {  // get total transaction number and minimal support from main file  String s = conf.get("totalTransaction");  String m = conf.get("minSupport");  // type convertting  this.totalTransaction = Integer.parseInt(s);  this.minSupport = Float.parseFloat(m);  }  public void reduce(Text key, Iterator<IntWritable> value, OutputCollector<Text, IntWritable> output, Reporter r)  throws IOException {  int supportValue = 0;  while (value.hasNext()) {  supportValue += value.next().get();  }  // if items number satisfies minimal support, then output this item as frequent set  if (supportValue >= minSupport \* totalTransaction && key.toString().length() != 0) {  output.collect(key, new IntWritable(supportValue));  }  }  } |

* AssociationMapper.java

|  |
| --- |
| package finalProject;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.Mapper;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reporter;  import java.io.IOException;  public class AssociationMapper extends MapReduceBase implements Mapper<Object, Text, Text, IntWritable> {  public void map(Object key, Text value, OutputCollector<Text, IntWritable> output, Reporter r) throws IOException {  String[] text = key.toString().split(Jobsubmitter.SEPARATOR);  // output original data except frequent sets with one item  if (text.length > 1) {  output.collect(new Text(key.toString()), new IntWritable(Integer.parseInt(value.toString())));  }  }  } |

* AssociationReducer.java

|  |
| --- |
| package finalProject;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.HashMap;  import java.util.Iterator;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapred.JobConf;  import org.apache.hadoop.mapred.JobConfigurable;  import org.apache.hadoop.mapred.MapReduceBase;  import org.apache.hadoop.mapred.OutputCollector;  import org.apache.hadoop.mapred.Reducer;  import org.apache.hadoop.mapred.Reporter;  public class AssociationReducer extends MapReduceBase  implements Reducer<Text, IntWritable, Text, Text>, JobConfigurable {  private HashMap<String, Integer> map = new HashMap<String, Integer>();  private float minConfidence;  private int totalTransaction;  public void configure(JobConf conf) {  String c = conf.get("minConfidence");  String s = conf.get("totalTransaction");  this.minConfidence = Float.parseFloat(c);  this.totalTransaction = Integer.parseInt(s);  String dictFile = conf.get("fs.defaultFS") + "/home/ec2-user/output/Frequent/part-r-00000";  try {  Path pt = new Path(dictFile);  FileSystem fs = FileSystem.get(conf);  BufferedReader fis = new BufferedReader(new InputStreamReader(fs.open(pt)));  String currLine = null;  while ((currLine = fis.readLine()) != null) {  String[] items = currLine.split("\t");  if (items.length == 2) {  map.put(items[0], Integer.parseInt(items[1]));  }  }  fis.close();  fs.close();  } catch (IOException e) {  e.printStackTrace();  }  }  public void reduce(Text key, Iterator<IntWritable> value, OutputCollector<Text, Text> output, Reporter r)  throws IOException {  String[] items = key.toString().split(Jobsubmitter.SEPARATOR);  String outputKey = "";  String outputValue = "";  int implyingItemSetSupport;  int fullItemsetSupport = map.get(key.toString());  int confidence;  while (value.hasNext()) {  value.next();  if (items.length >= 2) {  for (int i = items.length - 1; i >= 0; i--) {  String impliedItem = items[i];  String implyingItems = "";  int count = 1;  for (int j = 0; j < items.length; j++) {  if (j != i) {  implyingItems += items[j];  if (count < items.length - 1)  implyingItems += Jobsubmitter.SEPARATOR;  count++;  }  }  if (map.containsKey(implyingItems)) {  implyingItemSetSupport = map.get(implyingItems);  confidence = (int) (fullItemsetSupport \* 1.0 / implyingItemSetSupport \* 100);  if (confidence >= this.minConfidence \* 100) {  outputKey = implyingItems + " -> " + impliedItem;  outputValue = "["  + String.format("%.2f", (fullItemsetSupport \* 1.0 / this.totalTransaction \* 100))  + "%, " + confidence + "%]";  output.collect(new Text(outputKey), new Text(outputValue));  }  }  }  }  }  }  } |