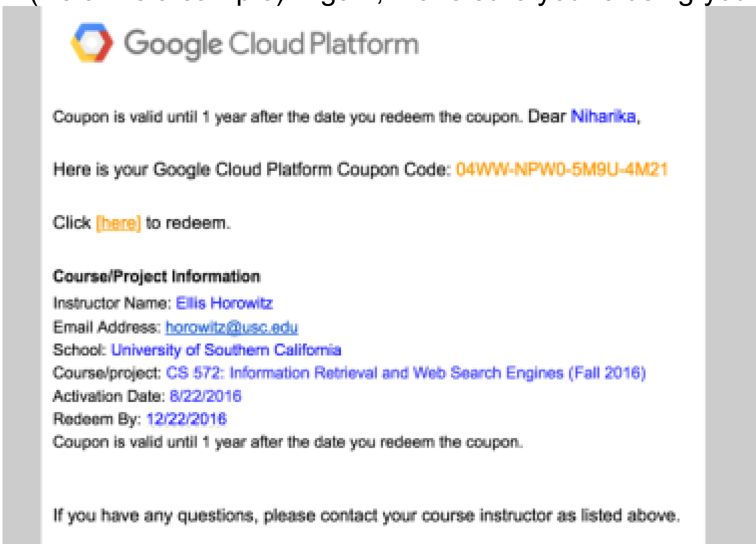
Creating an Inverted Index using Hadoop

Redeeming Google Cloud Credits

1. Go to https://goo.gl/gcpedu/zvmhM6 to redeem the $150 Google Cloud Platform Credit. Make sure you use your `.edu` email.
2. Follow the instructions provided. At the end of the process you will receive a Google Cloud Platform code in your email.(Below is a sample). Again, make sure you're using your USC Gmail account.


Google Cloud Platform

Coupon is valid until 1 year after the date you redeem the coupon. Dear Niharika,

Here is your Google Cloud Platform Coupon Code: 04WW-NPW0-5M9U-4M21

Click [here] to redeem.

Course/Project Information
Instructor Name: Ellis Horowitz
Email Address: horowitz@usc.edu
School: University of Southern California
Course/project: CS 572: Information Retrieval and Web Search Engines (Fall 2016)
Activation Date: 8/22/2016
Redeem By: 12/22/2016
Coupon is valid until 1 year after the date you redeem the coupon.

If you have any questions, please contact your course instructor as listed above.

At the final step click on [**here**] in the email message and you are directed to the Google Cloud Platform containing your coupon code and credit. Click on Accept and Continue. You should end up at the billing section of your account, with CSCI 572 noted at the top. **Note**: the home page for Google Cloud is https://console.cloud.google.com.

Setting up Your Initial Machine

Click on "Project" at the top of the window and either create a new project or select an existing one. For new projects choose a name. It may take a while to complete, but eventually you will be redirected to the Google cloud Dashboard.

Google has a large set of APIs, that will appear if you click on the menu immediately to the left of Google Cloud Platform. You will get a list that looks like Figure 2 below. Included in the BIG DATA category are: BigQuery, Pub/Sub, Dataproc, Dataflow, Machine Learning and Genomics. For this exercise we will use Dataproc. Using Dataproc we can quickly create a cluster of compute instances running Hadoop. The alternative to Dataproc would be to individually setup each compute node, install Hadoop on it, set up HDFS, set up master node, etc. Dataproc automates this grueling process for us. Follow the instructions below to create a Hadoop cluster using Dataproc.

Creating a Hadoop Cluster on Google Cloud Platform

1. Create a Google Dataproc Cluster. Select **Dataproc** from the navigation list on the left

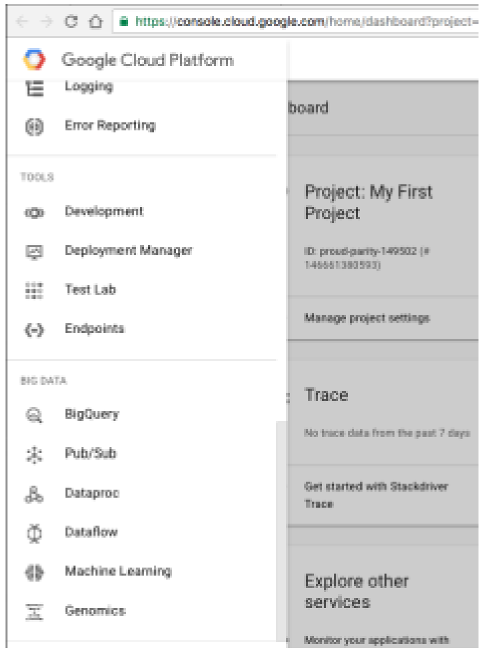


Figure 2: Google Cloud Platform APIs

2. If this is the first time you're using Dataproc then you'll encounter the error in the below screenshot (Figure 3). This means that your Google cloud account doesn't have the required API enabled. To enable the API copy the link in the error description and go to it. You will land on a page similar to the one in **Figure 4.** Click the **Enable** button at the top of the page to enable the Dataproc API.

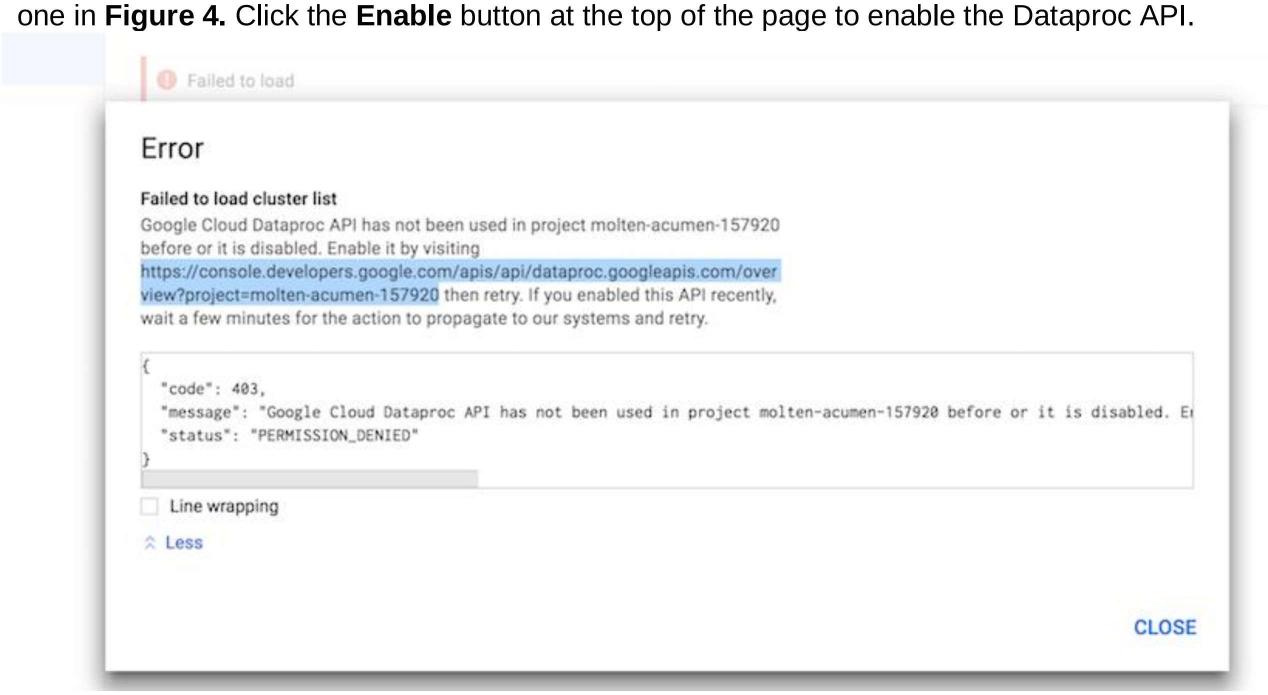


Figure 3: Error caused when trying to create a cluster for the first time

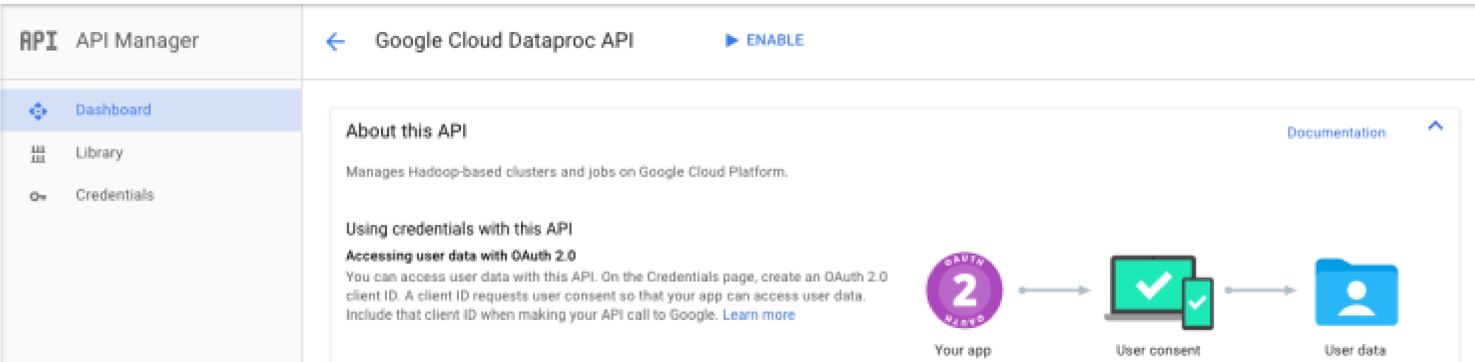


Figure 4: Enabling the Dataproc API

3. Once you've enabled the API go back to the page where you got the error earlier and reload the page. You'll now see a dialog box with a **Create Cluster** button (Figure 5).

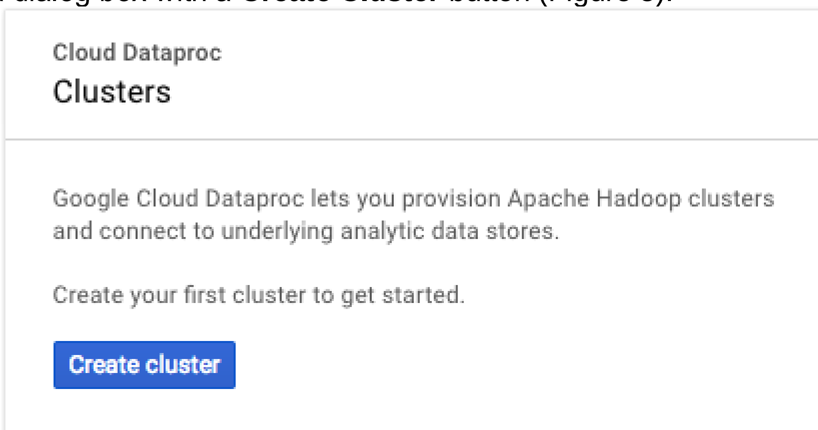


Figure 5: This is what you see once the API is enabled

4. Clicking on "**Create Cluster**" will take you to the cluster configuration section (Figure 7). Give any unique name to your cluster and select a **us-west** zone. You need to create a master and 3 worker nodes. Select the default configuration processors **(n1-standard-4 4vCPU 15 GB memory)** for each member and reduce the storage to **32 GB** HDD storage. Leave everything else default and click on "**Create**".
If you get an error (Figure 6) saying that you've exceeded your quota, reduce the number of worker nodes or choose a Machine Type(for master and worker) with fewer **vCPUs**. In rare cases you may get the error in **Figure 3** again. If so, simply follow the instructions in step 2 again. If all goes well your cluster will be created in a few minutes.

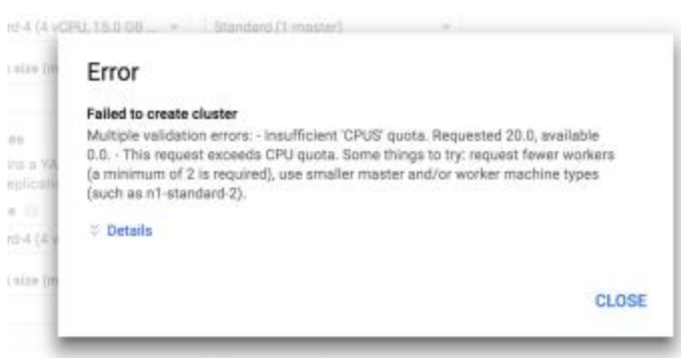


Figure 6: Insufficient CPU Quota error

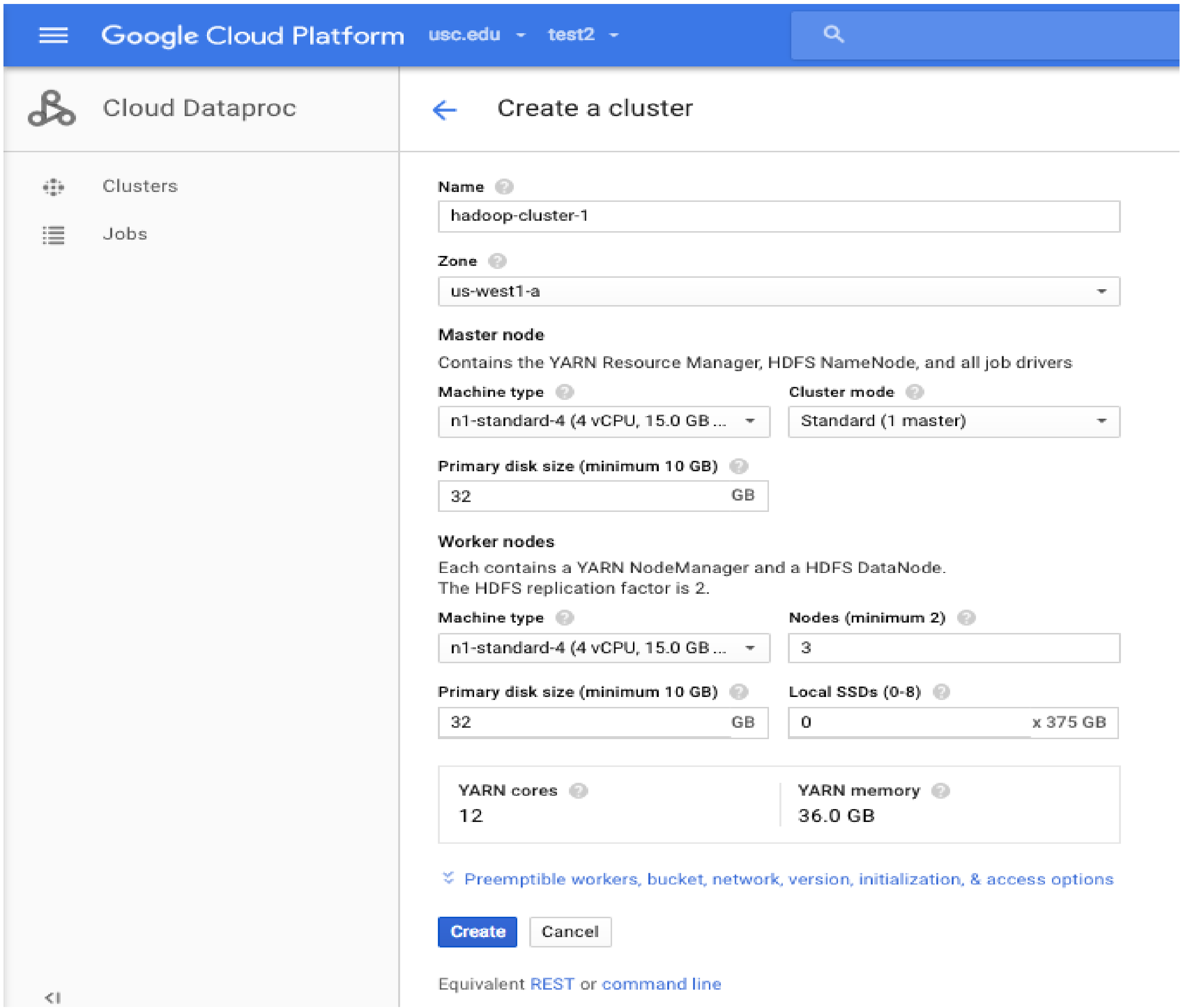


Figure 7: Screen for setting up a cluster

5. Now that the cluster is setup we'll have to configure it a little before we can run jobs on it. Select the cluster you just created from the list of clusters under the cloud Dataproc section on your console. Go to the **VM Instances** tab and click on the **SSH** button next to the instance with the **Master** Role. If you don't see the SSH button click the **Refresh** button on the top of the page.

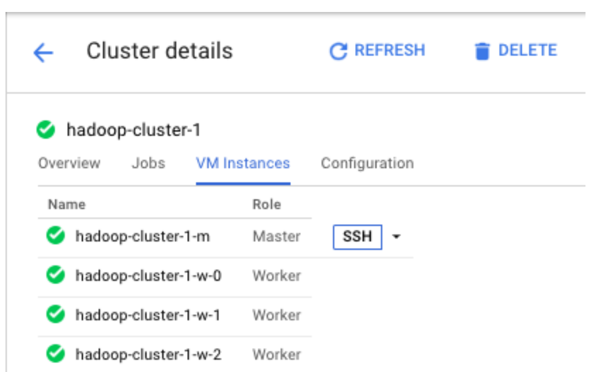


Figure 8: SSH into the master node.

6. Clicking on the **SSH** button will take you to a Command line Interface(CLI) like an xTerm or Terminal. All the commands in the following steps are to be entered in the CLI.
   There is no home directory on HDFS for the current user so set up the user directory on HDFS. So, we'll have to set this up before proceeding further. (To find out your user name run `whoami`)
   - `hadoop fs -mkdir -p /user/<your username here>`
7. Set up environment variables for JAVA and HADOOP_CLASSPATH. Please note that this step has to be done each time you open a new SSH terminal.
   - `JAVA_HOME` is already set-up. Do not change this.
   - `export PATH=${JAVA_HOME}/bin:${PATH}`
   - `export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar`

   To ensure that the environment variables are set, run the command `env`. You should see the path associated with `JAVA_HOME` in the `PATH` variable and a new variable called `HADOOP_CLASSPATH` as highlighted in the image below.

8. Run `hadoop fs -ls`
9. If there is no error this implies that your cluster was successfully set up. If you do encounter an error it's most likely due to a missing environment variable or user home directory not being set up right. Retrace steps 1 to 6 to fix this.

NOTE:

- Please **disable** the billing for the cluster when you are not using it. Leaving it running will cost extra credits. The cluster is billed based on how many hours it is running and not how much data it is processing. So, if you leave the billing enabled overnight on an idle cluster you will still incur significant charges.
- Click the on the top left corner in the Google console and go to the **Billing** section. Click the button next to the project you created initially, and select *disable billing*. Please do this whenever you are not working on the cluster.
- See the **"Enable and Disable Billing account"** section on **page 11** for detailed instructions on how to do this.

Upload Data(Books) to the Storage Bucket

For this project you will be creating an Inverted Index of words occurring in a set of English books. We'll be using a collection of 3,036 English books written by 142 authors acquired from [here](). This collection is a small subset of the Project Gutenberg corpus that was further cleaned for the purpose of this assignment.

These books will be placed in a bucket on your Google cloud storage and the Hadoop job will be instructed to read the input from this bucket.

1. Uploading the input data into the bucket
    a. Get the books from either of the links below
    http://www-scf.usc.edu/~csci572/2017Spring/hw3/DATA.zip

    https://drive.google.com/open?id=0B0rRqvGB9KPBU2pJX01uenh2NWc
    Use your USC account to get access to the data from the Google Drive link. The full data is around 385MB.

    b. Unzip the contents. You will find two folders inside named **'development'** and '**full data**'. Each of the folders contains the actual data(books) and a mapper file to map the docID to the file name. We suggest you use the development data initially while you are testing your code. Using the full data will take up to 2 hours for each run of the Map-Reduce job and you may risk spending all your cloud credits while testing the code.

    c. Click on 'Dataproc' in the left navigation menu under ☰. Next, locate the address of the default **Google cloud storage staging** bucket for your cluster. Underlined in blue in Figure-9 below. If you've previously disabled billing, you need to re-enable it before you can upload the data. Refer to the **"Enable and Disable Billing account"** section to see how to do this.

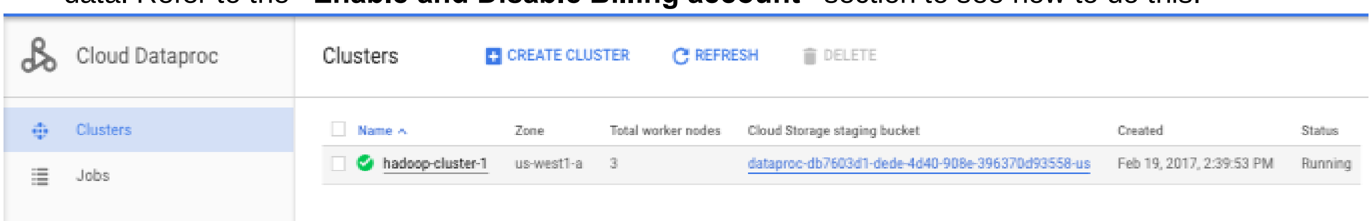


Figure 9: The default Cloud Storage bucket.

    d. Go to the storage section in the left navigation bar select your cluster's default bucket from the list of buckets. At the top you should see menu items UPLOAD FILES, UPLOAD FOLDER, CREATE FOLDER, etc. Click on the UPLOAD FOLDER button and upload the `dev_data` folder and `full_data` folder individually. This will take a while, but there will be a progress bar (Figure 11). You may not see this progress bar as soon as you start the upload but, it will show up eventually.

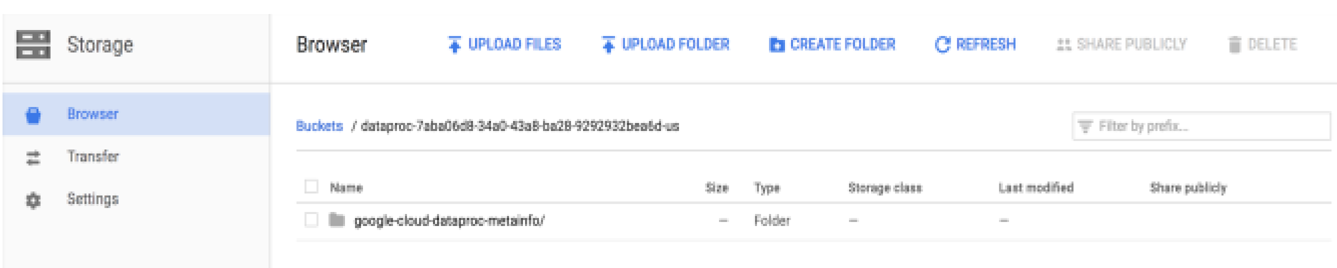


Figure 10: Cloud Storage Bucket.

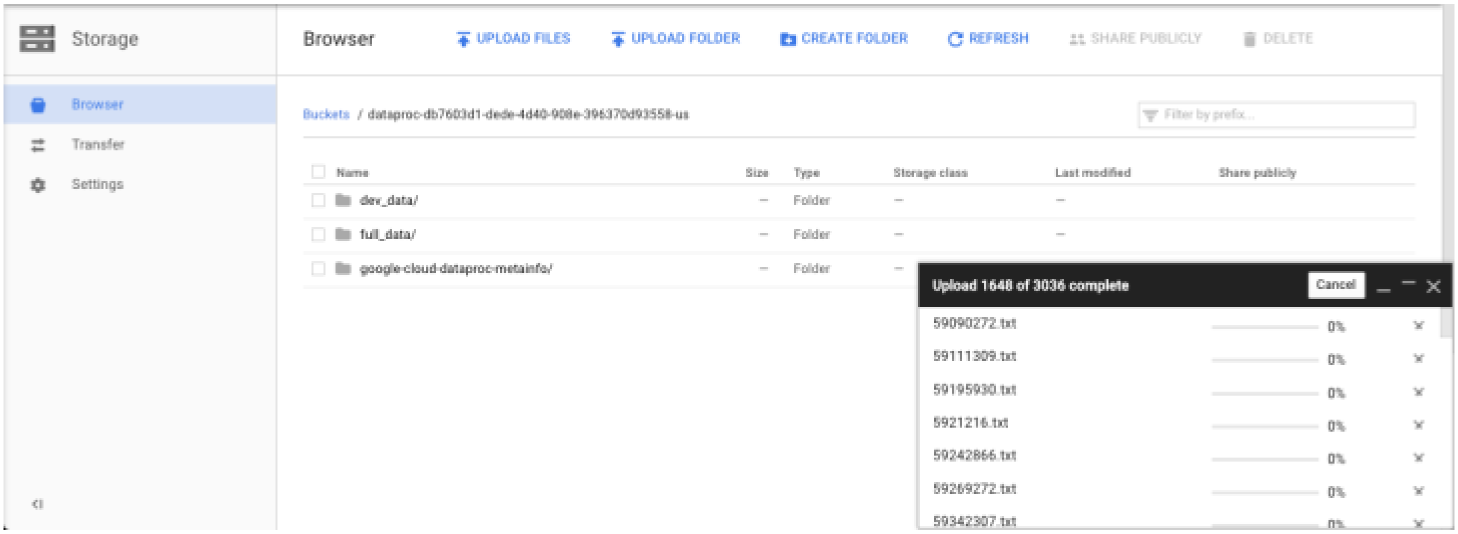


Figure 11: Progress of uploading

Inverted Index Implementation using Map-Reduce

Now that you have the cluster and the books in place, you need to write the actual code for the job. As of now, Google Cloud allows us to submit jobs via the UI, only if they are packaged as a jar file. The following steps are focussed on submitting a job written in Java via the Cloud console UI.

Refer to the below examples and write a Map-Reduce job in java that creates an Inverted Index given a collection of text files. You can very easily tweak a **word-count example** to create an inverted index instead (**Hint**: Change the mapper to output `word` **`docID`** instead of `word count` and in the reducer use a **HashMap**).

Examples of Map-Reduce Jobs

1. https://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
2. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

The example in the following pages explains a Hadoop word count implementation in detail. It takes one text file as input and returns the word count for every word in the file. Refer to the comments in the code for explanation.

The Mapper Class:

```java
/*
This is the Mapper class. It extends the Hadoop's Mapper class.
This maps input key/value pairs to a set of intermediate(output) key/value pairs.
Here our input key is a LongWritable and input value is a Text.
And the output key is a Text and value is an IntWritable.
*/
class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    /*
    Hadoop supported data types. This is a Hadoop specific datatype that is used to handle
    numbers and Strings in a hadoop environment. IntWritable and Text are used instead of
    Java's Integer and String datatypes.
    Here 'one' is the number of occurances of the 'word' and is set to the value 1 during the
    Map process.
    */
    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
    {
        //Reading input one line at a time and tokenizing.
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        //Iterating through all the words available in that line and forming the key value pair.
        while (tokenizer.hasMoreTokens())
        {
            word.set(tokenizer.nextToken());
            /*
            Sending to output collector(Context) which in-turn passes the output to Reducer.
            The output is as follows:
                'word1' 1
                'word1' 1
                'word2' 1
            */
            context.write(word, one);
        }
    }
}
```

The Reducer Class:

```
/*
This is the Reducer class. It extends the Hadoop's Reducer class.
This maps the intermediate key/value pairs we get from the mapper to a set
of output key/value pairs, where the key is the word and the value is the word's count.
Here our input key is a Text and input value is a IntWritable.
And the output key is a Text and value is an IntWritable.
*/
class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
    /*
    Reduce method collects the output of the Mapper and adds the 1's to get the word's count.
    */
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException
    {
        int sum = 0;
        /*
        Iterates through all the values available with a key and add them together and give the
        final result as the key and sum of its values
        */
        for (IntWritable value : values)
        {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Main Class

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.*;
public class WordCount
{
    public static void main(String[] args)
        throws IOException, ClassNotFoundException, InterruptedException {
        if (args.length != 2) {
            System.err.println("Usage: Word Count <input path> <output path>");
            System.exit(-1);
        }
        //Creating a Hadoop job and assigning a job name for identification.
        Job job = new Job();
        job.setJarByClass(WordCount.class);
        job.setJobName("Word Count");
        //The HDFS input and output directories to be fetched from the Dataproc job submission console.
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //Providing the mapper and reducer class names.
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        //Setting the job object with the data types of output key(Text) and value(IntWritable).
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.waitForCompletion(true);
    }
}
```

We've already cleaned up the input data so you don't have to worry about any stray characters. Each input file consists of exactly one book that has been cleared of `'\n\r'`, `'\n'` and all but one `'\t'`. The only `'\t'` separates the key(Document ID) from the value(Document). The input files are in a key value format as below:

```
DocumentID        document
```

Sample document:

```
51918182    four meetings  by henry james  1885   i saw her only four times but i remember them
vividly she made an impression upon me i thought her very pretty and very interestinga charming
specimen of a type i am very sorry to hear of her death and yet when i think of it why should i
be sorry the last time i saw her she was certainly notbut i will describe all our meetings in
order     i  the first one took place in the country at a little teaparty one snowy night it must
have been some seventeen years ago my friend latouche going to spend christmas with his mother
```

The mapper's output is expected to be as follows:

```
james   51918182
james   51918182
james   51918182
people  51918182
people 51918182
of  51918182
of  51918182
```

The above example indicates that the word `james` occurred 3 times in the document with docID `51918182` and `people` 2 times.

The reducer takes this as input, aggregates the word counts using a Hashmap and creates the Inverted index. The format of the index is as follows.

```
word  docID:count      docID:count      docID:count...
```

```
ably    9931985:1
abnegate    85886314:1  80811098:1
abney   31694096:3  15109590:1  38583612:1  98115965:98
abnormal    47943267:1  94435826:1  80942074:1
abroad  73713297:1  11200532:1
```

The above sample shows the inverted index created by the reducer. The docID's can be mapped to their document names using the docID2name.csv file in the download package.

To write the Hadoop java code you can use the **VI** or **nano** editors that come pre-installed on the master node. You can test your code on the cluster itself. Be sure to use the development data while testing the code. You are expected to write a simple Hadoop job. You can just tweak [this]() example if you'd like but, make sure you understand it first.

Creating a jar for your code

Now that your code for the job is ready we'll need to run it. The Google Cloud console requires us to upload a Map-Reduce job as a jar file. In the following example the Mapper and Reducer are in the same file called `InvertedIndexJob.java`. To create a jar for the Java class implemented please follow the instructions below. The following instructions were executed on the cluster's master node on the Google Cloud.

1. Say your Java Job file is called `InvertedIndex.java`. Create a JAR as follows:
   - `hadoop com.sun.tools.javac.Main InvertedIndexJob.java`

   If you get the following Note you can ignore them

   `Note: InvertedIndexJob.java uses or overrides a deprecated API.`

Note: Recompile with -Xlint:deprecation for details.

- `jar cf invertedindex.jar InvertedIndex*.class`

Now you have a jar file for your job. You need to place this jar file in the default cloud bucket of your cluster. Just create a folder called JAR on your bucket and upload it to that folder. If you created your jar file on the cluster's master node itself use the following commands to copy it to the `JAR` folder.

- `hadoop fs -copyFromLocal ./invertedindex.jar`
- `hadoop fs -cp ./invertedindex.jar gs://dataproc-69070.../JAR`

The highlighted part is the default bucket of your cluster. It needs to be prepended by the `gs://` to tell the Hadoop environment that it is a bucket and not a regular location on the filesystem.

Note: This is not the only way to package your code into a jar file. You can follow any method that will create a single jar file that can be uploaded to the google cloud.

Submitting the Hadoop job to your cluster

As mentioned before, a job can be submitted in two ways.

1. From the console's UI.
2. From the command line on the master node.

If you'd like to submit the job via the command line follow the instructions here

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Follow the instructions below to submit a job to the cluster via the console's UI.

1. Go to the "Jobs" section in the left navigation bar of the Dataproc page and click on "**Submit job**".

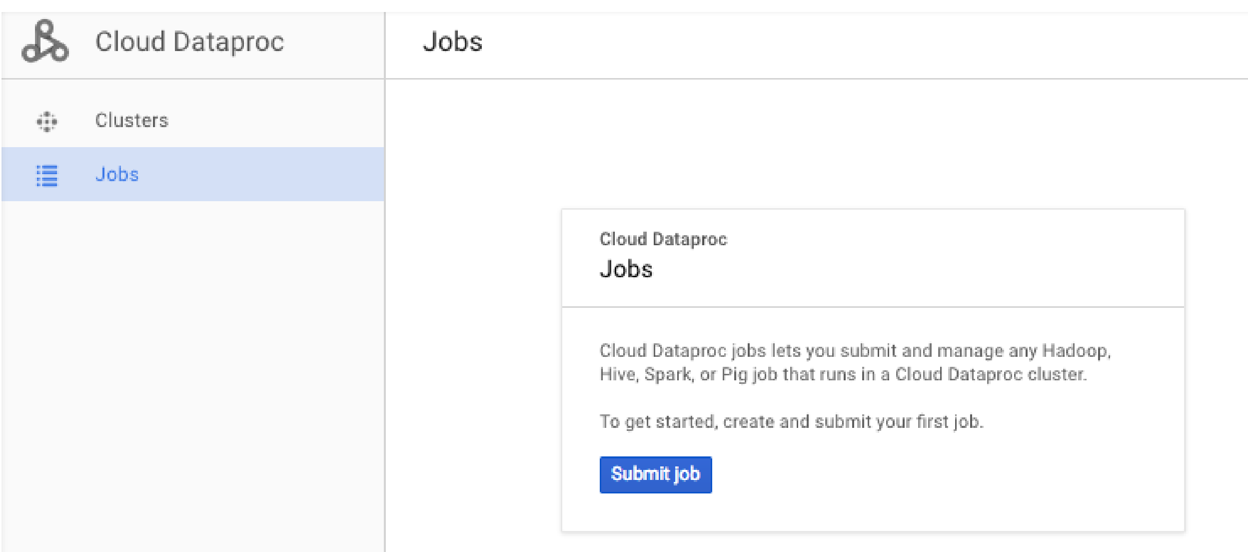


Figure 12: Dataproc jobs section

2. Fill the job parameters as follows (see Figure 13 for reference):
    - **Cluster**: Select the cluster you created
    - **Job Type**: Hadoop
    - **Jar File**: Full path to the jar file you uploaded earlier to the Google storage bucket. Don't forget the `gs://`
    - **Main Class or jar**: The name of the java class you wrote the mapper and reducer in.
    - **Arguments**: This takes two arguments
        i. **Input**: Path to the input data you uploaded
        ii. **Output**: Path to the storage bucket followed by a **new** folder name. The folder is created during execution. You will get an error if you give the name of an existing folder.
    - Leave the rest at their default settings

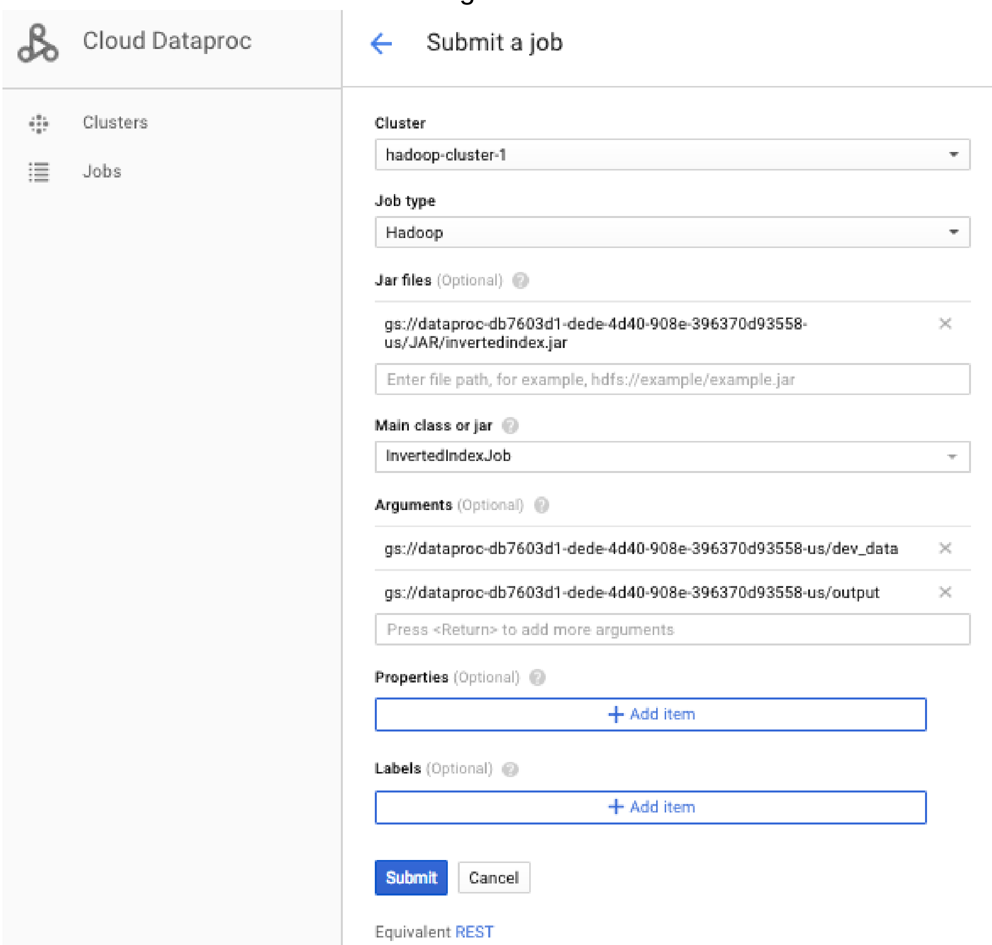


Figure 13: Job submission details

3. Submit Job. It will take quite a while. Please be patient. You can see the progress on the job's status section.

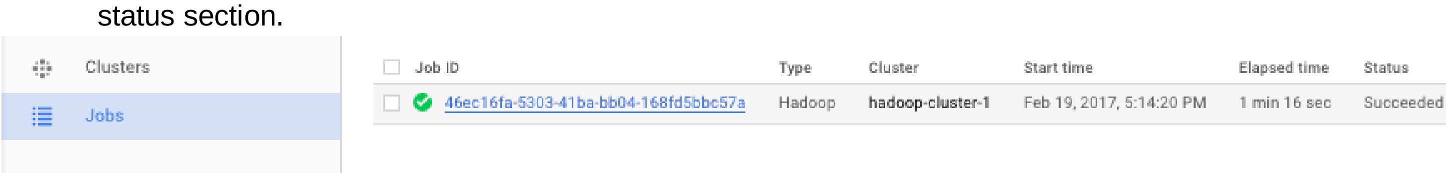


Figure 14: Job ID generated. Click it to view the status of the job.

NOTE: If you encounter a **Java.lang.Interrupted exception** you can safely ignore it. Your submission will still execute.

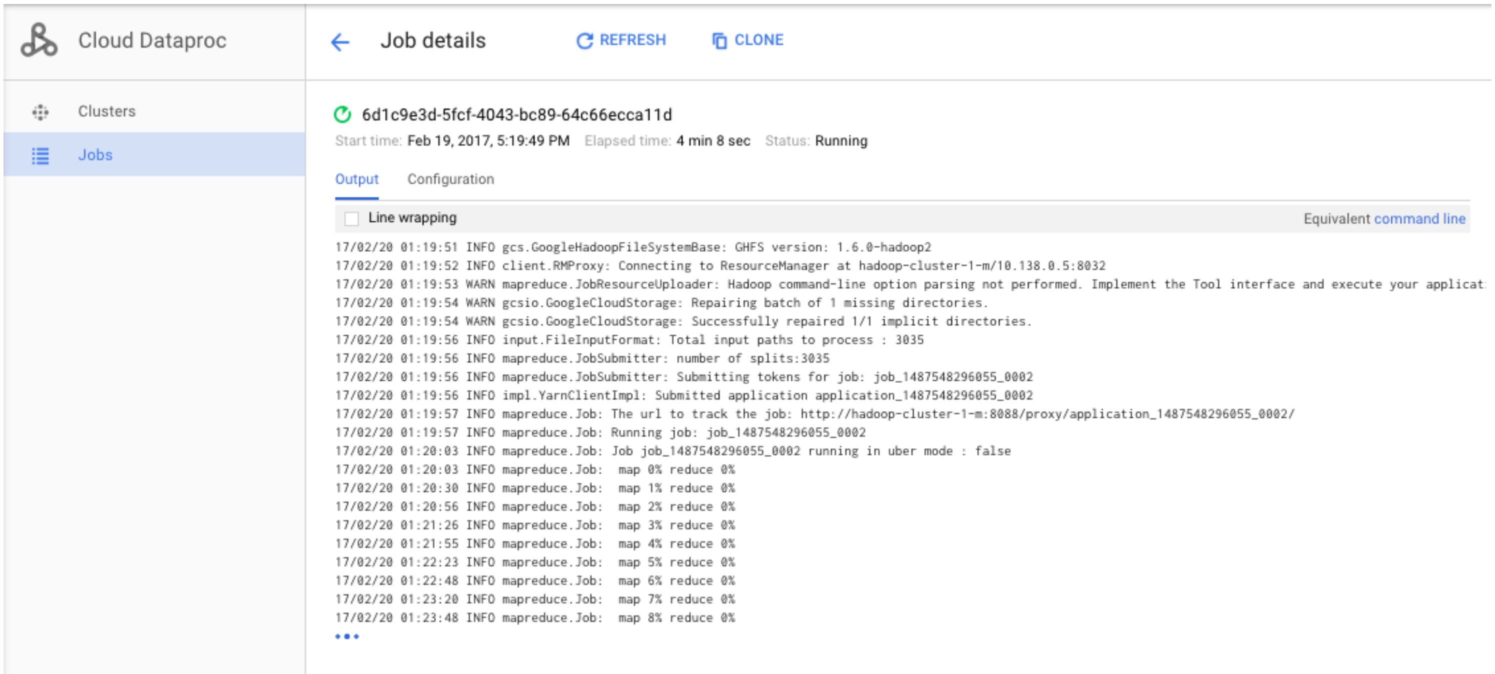


Figure 14: Job progress

4. Once the job executes copy all the log entries that were generated to a text file called `log.txt`. You need to submit this log along with the java code. You need to do this only for the job you run on the full data. No need to submit the logs for the dev_data.

5. The output files will be stored in the `output` folder on the bucket. If you open this folder you'll notice that the inverted index is in several segments.(Delete the _**SUCCESS** file in the folder before merging all the output files)

To merge the output files, run the following command in the master nodes command line(SSH)

- `hadoop fs -getmerge gs://dataproc-69070458-bbe2-.../output ./output.txt`
- `hadoop fs -copyFromLocal ./output.txt`
- `hadoop fs -cp ./output.txt gs://dataproc-69070458-bbe2-.../output.txt`

The output.txt file in the bucket contains the full Inverted Index for all the books.

Use `grep` to search for the words mentioned in the submissions section. Using grep is the fastest way to get the entries associated with the words.

For example to search for "`string`" use

```
grep -w '^string      ' fullindex.txt
```

Enabling and Disabling Billing accounts

We need to disable billing for the project (where the cluster was created) when we are not running the job to save some credits. Follow the steps below to disable and enable the billing for your project:

Disable Billing:

1. Click the navigation button on the top left .
2. Navigate to the billing section.
3. Click on Disable billing for the project you created.(See screenshot below)

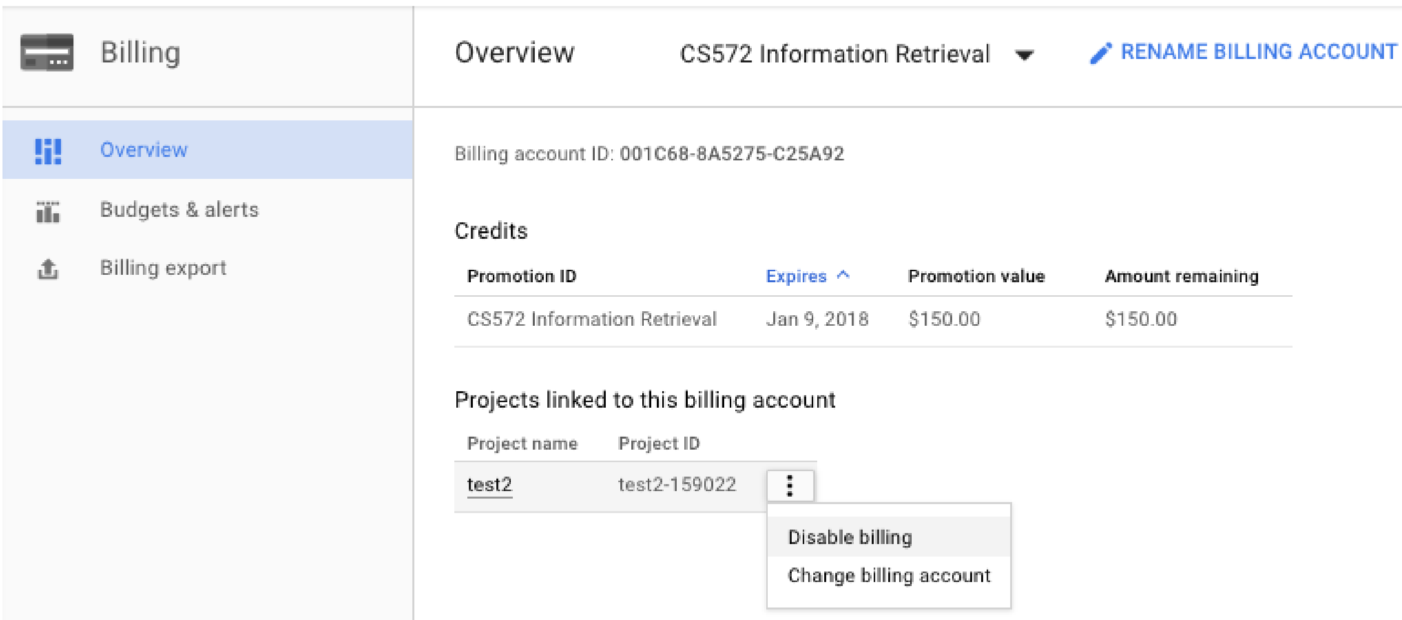


Figure 15: Disabling the billing for the cluster.

Enable Billing:

Option 1: When you navigate to the billing section you will be prompted to select the billing account. Select “CS572 Information Retrieval”. This billing account is created when you redeem the google credits.

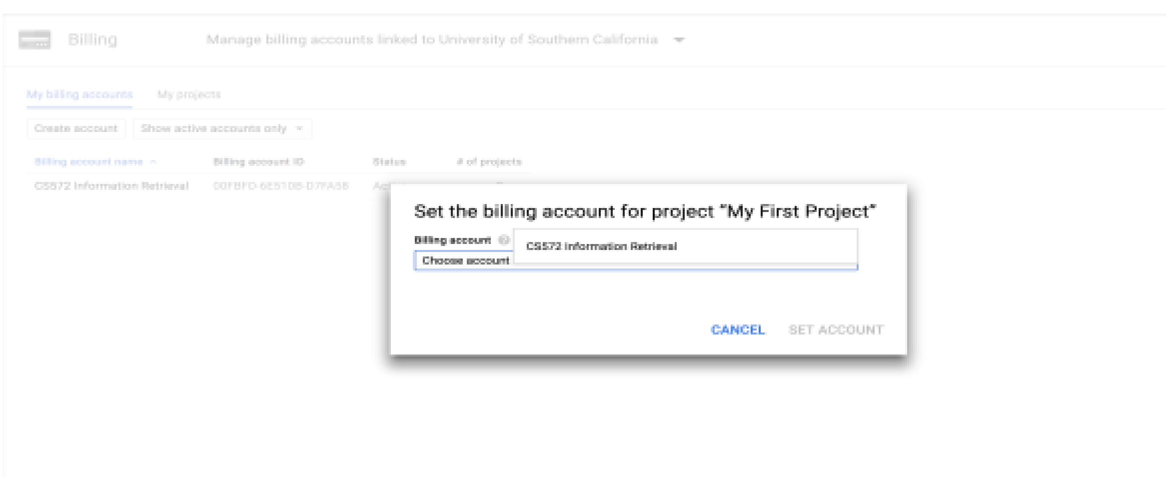


Figure 16: Select the account “CS572 Information Retrieval”

Option 2:

1. Navigate to the Dataproc section. You will see a screen similar to the figure below. Click on Enable billing.

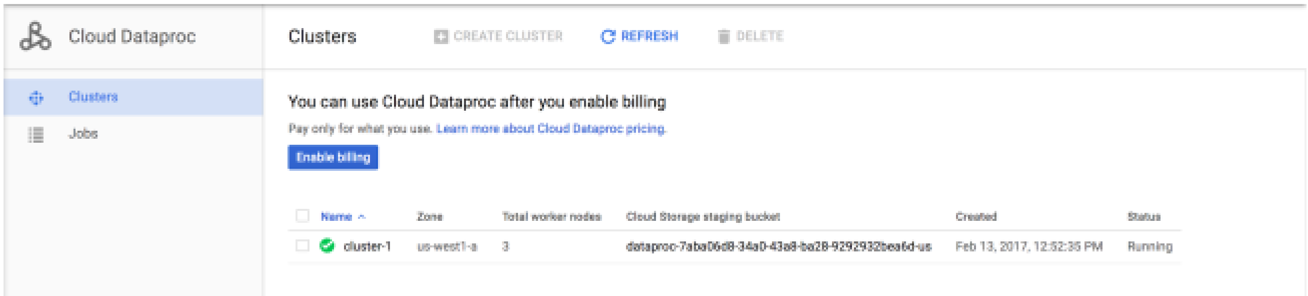


Figure 17: Enable billing

NOTE : Every time you disable and enable billing for a cluster, the Virtual Machines in the cluster don't start by themselves. We need to manually start the VMs. In the VM Instances section of the Cluster you might see all the VM's of the cluster disabled (See Figure 18). To enable the VM Instances, navigate to the Compute Engine section. Select all the instances corresponding to the cluster you created and click on the START button. Once activated navigate back to the Dataproc section to resume working on the cluster.

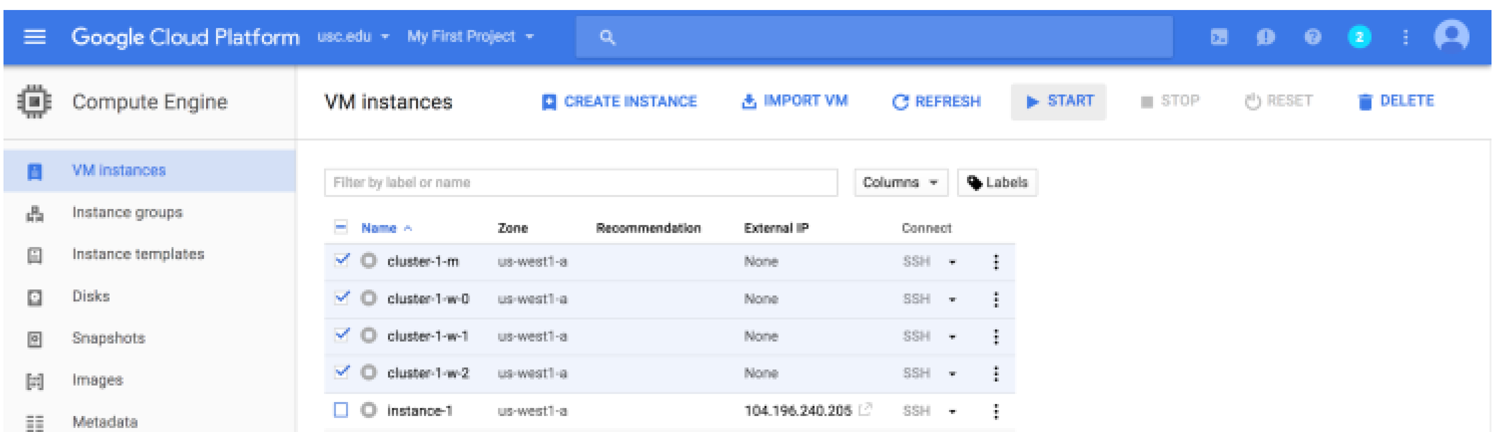


Figure 18: Select all virtual machines associated with the cluster.

Credits Spent:

To check how much you've been charged for your cluster, navigate to the Billing section and click on the project name in the Overview section (see Figure 19 & 20). We suggest you check this section at least once every 24 hours.

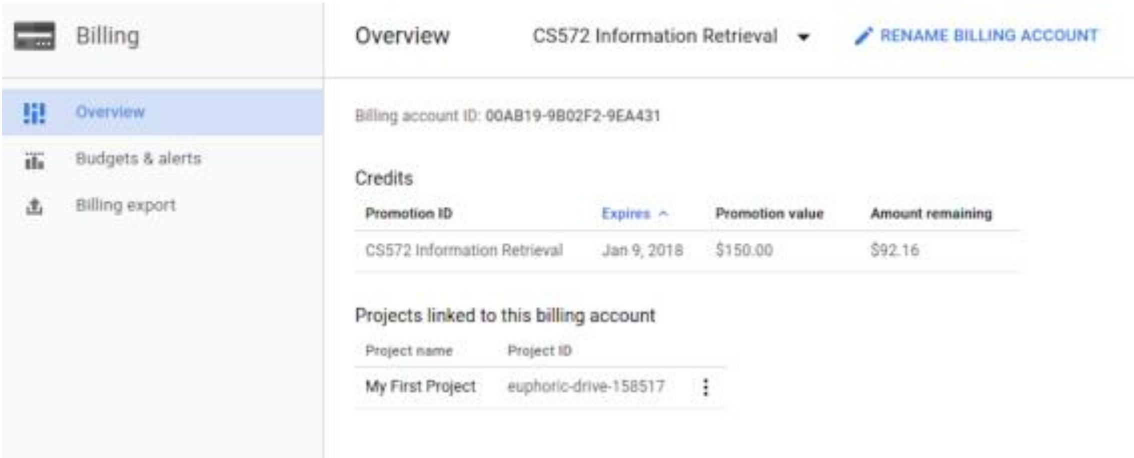


Figure 19: Billing Overview section.

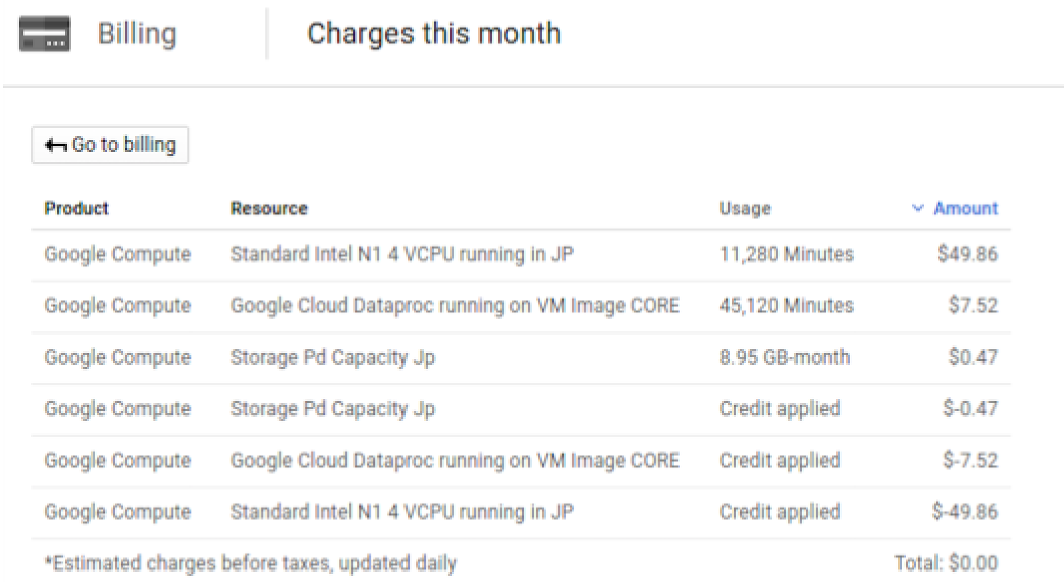


Figure 20: Cluster usage cost

Submission Instructions:

1. Include all the code that you have written(java) and the log file created for the full data job submission.
2. Also include the inverted index file for the book “Bram Stoker___The Jewel of Seven Stars.txt”
3. Create a text file named `index.txt` and include the index entries for the following words
    a. magnifiers
    b. livelihood
    c. thrilling
    d. fuel
    e. warning
    f. mars
    g. sawyer
    h. guaranty

    Add the full line from the index including the word itself.
4. **Do NOT submit your full index.**
5. Compress your code and the text file into a single zip archive and name it `index.zip`. Use a standard zip format and not zipx, rar, ace, etc.
6. To submit your file electronically to the csci572 account enter the following command from your UNIX prompt:

```
$ submit -user csci572 -tag hw3 index.zip
```