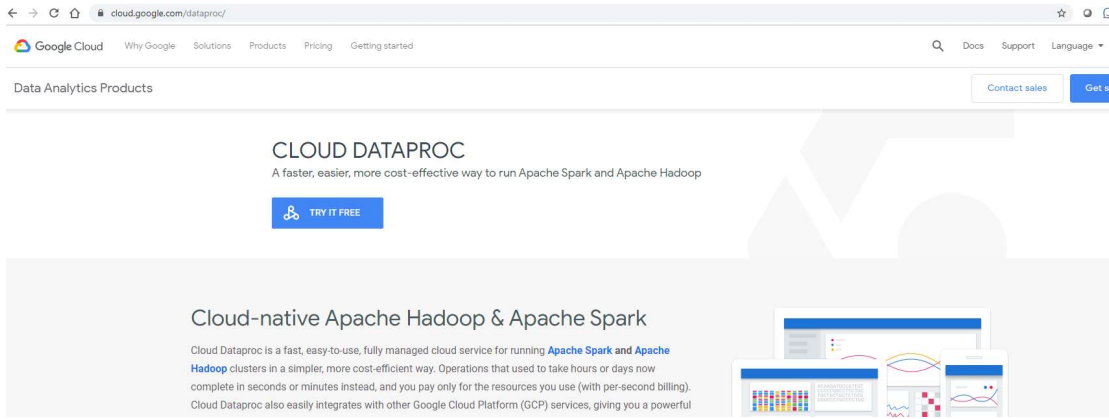


# Creating an Inverted Index using Hadoop

## Using Google Cloud Dataproc

1. You can read about GCP Dataproc and create your free trial account using this URL:  
<https://cloud.google.com/dataproc/>



**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

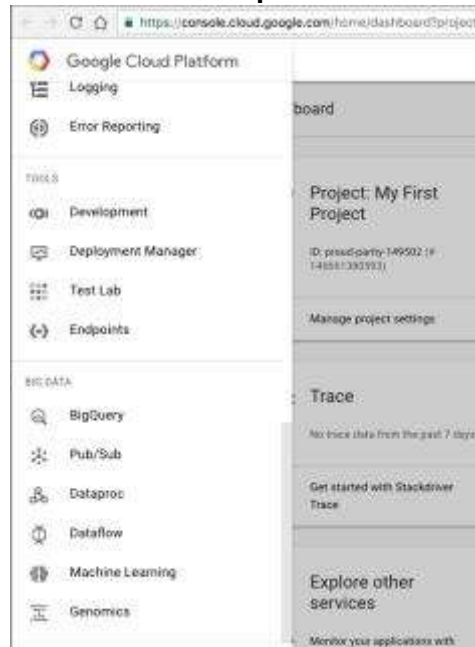


Figure 2: Google Cloud Platform APIs

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.

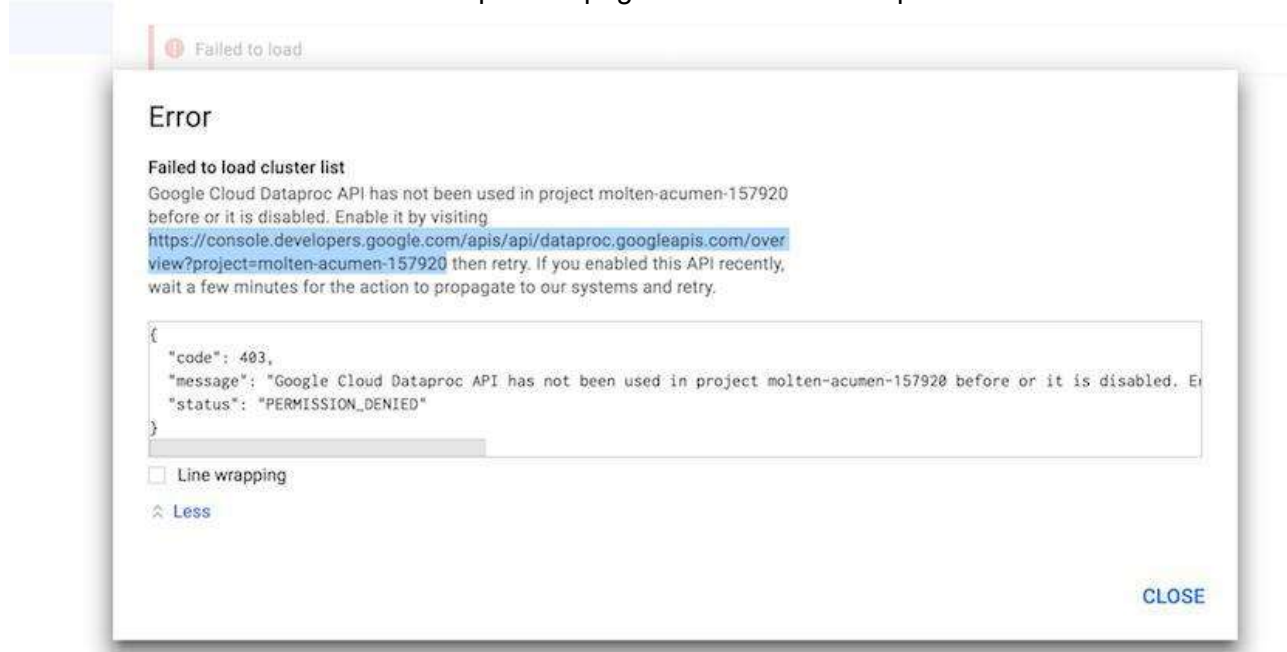


Figure 3: Error caused when trying to create a cluster for the first time

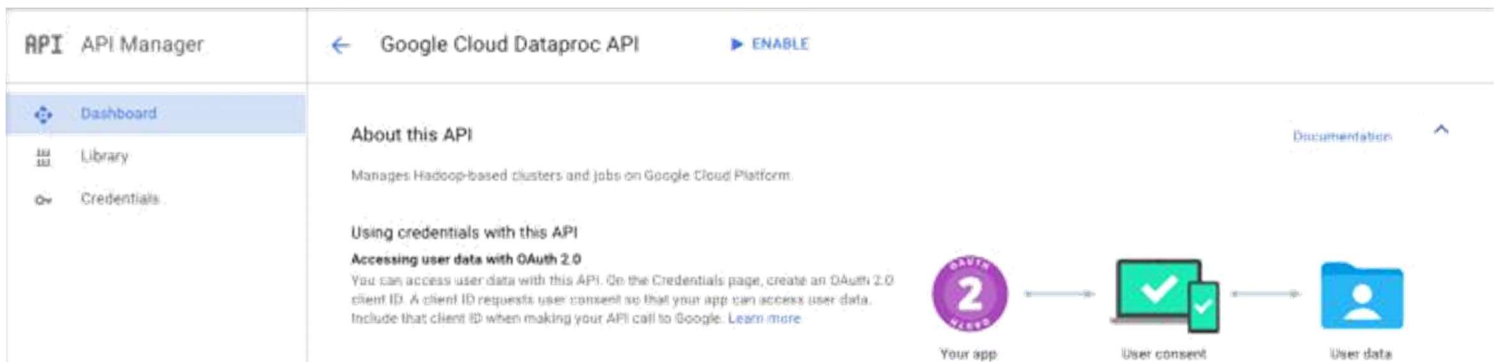


Figure 4: Enabling the Dataproc API

- 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).

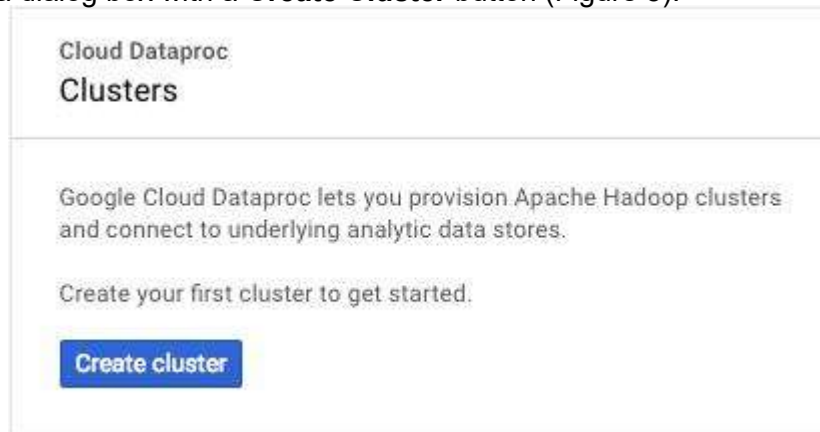


Figure 5: This is what you see once the API is enabled

- 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.

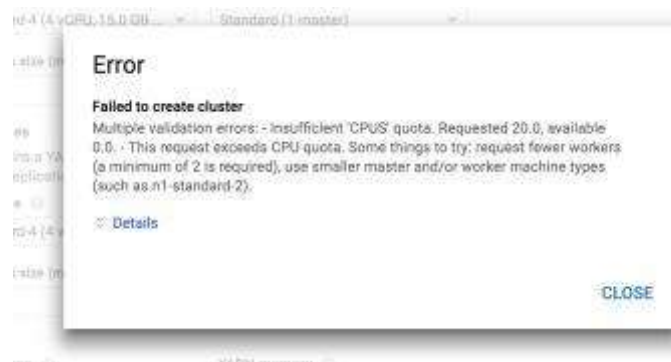


Figure 6: Insufficient CPU Quota error

Google Cloud Platform usc.edu test2

Cloud Dataproc

← Create a cluster

**Name**  
hadoop-cluster-1

**Zone**  
us-west1-a

**Master node**  
Contains the YARN Resource Manager, HDFS NameNode, and all job drivers

**Machine type**  
n1-standard-4 (4 vCPU, 15.0 GB ...)

**Cluster mode**  
Standard (1 master)

**Primary disk size (minimum 10 GB)**  
32 GB

**Worker nodes**  
Each contains a YARN NodeManager and a HDFS DataNode.  
The HDFS replication factor is 2.

**Machine type**  
n1-standard-4 (4 vCPU, 15.0 GB ...)

**Nodes (minimum 2)**  
3

**Primary disk size (minimum 10 GB)**  
32 GB

**Local SSDs (0-8)**  
0 x 375 GB

**YARN cores**  
12

**YARN memory**  
36.0 GB

Preemptible workers, bucket, network, version, initialization, & access options

Create Cancel

Equivalent REST or command line

Figure 7: Screen for setting up a cluster

- 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.

← Cluster details REFRESH DELETE

hadoop-cluster-1

Overview Jobs VM Instances Configuration

Name	Role	SSH
hadoop-cluster-1-m	Master	SSH
hadoop-cluster-1-w-0	Worker	
hadoop-cluster-1-w-1	Worker	
hadoop-cluster-1-w-2	Worker	

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.

```
SSH_AUTH_SOCK=/tmp/ssh-7A9Ga0Buk/agent.2227
DATAPROC_MASTER_COMPONENTS=hadoop-hdfs-namenode hadoop-yarn-resourceanagier mysql-server
MAIL=/var/mail/adasari
PATH=/usr/lib/jvm/java-8-openjdk-amd64/bin:/usr/lib/jvm/java-8-openjdk-amd64/bin:/usr/lib/jvm/java-8-openjdk-amd64/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
PWD=/home/adasari
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
LANG=en_US.UTF-8
DATAPROC_COMMON_COMPONENTS=openjdk-8-jdk libansi-java python-numpy libmvsq1-java hadoop-client hive pig spark-core spark-
```


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 to the Storage Bucket

For this project you will be creating an Inverted Index of words occurring in a set of English libraries. These libraries 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. Upload the data files attached with the project
  - b. 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.

 Cloud Dataproc

Clusters

Jobs

Clusters

CREATE CLUSTER

REFRESH

DELETE

<input type="checkbox"/>	Name ^	Zone	Total worker nodes	Cloud Storage staging bucket	Created	Status
<input checked="" type="checkbox"/>	hadoop-cluster-1	us-west1-a	3	<a href="#">dataproc-db7503d1-dede-4d40-908e-396370d93558-us</a>	Feb 19, 2017, 2:39:53 PM	Running

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 data folders individually. This will take a while, but there will be a progress bar. You may not see this progress bar as soon as you start the upload but, it will show up eventually.

Storage

Browser

Transfer

Settings

Browser

UPLOAD FILES

UPLOAD FOLDER

CREATE FOLDER

REFRESH

SHARE PUBLICLY

DELETE

Buckets / dataproc-7aba06d8-34a0-43a8-ba28-9292932bee8d-us

Filter by prefix...


<input type="checkbox"/>	Name	Size	Type	Storage class	Last modified	Share publicly
<input type="checkbox"/>	 google-cloud-dataproc-metainfo/	-	Folder	-	-	

Figure 10: Cloud Storage Bucket.

## Inverted Index Implementation using Map-Reduce

Now that you have the cluster and the data 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**).

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:

```
/*
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 occurrences 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);
    }
}

```

:

DocumentID	document
------------	----------

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.

## Creating a jar for your code (Sample Example)

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**".

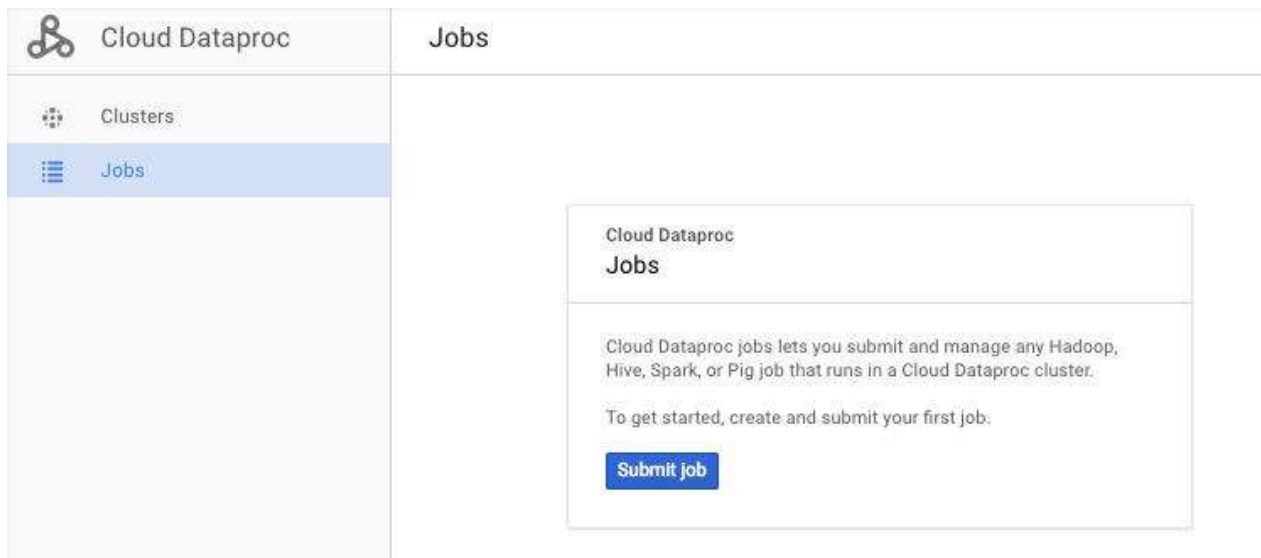


Figure 12: Dataproc jobs section

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

Cloud Dataproc

Submit a job

Clusters

Jobs

Cluster

hadoop-cluster-1

Job type

Hadoop

Jar files (Optional)

gs://dataproc-db7603d1-dede-4d40-908e-396370d93558-us/JAR/invertedindex.jar

Enter file path, for example, hdfs://example/example.jar

Main class or jar

InvertedIndexJob

Arguments (Optional)

gs://dataproc-db7603d1-dede-4d40-908e-396370d93558-us/dev\_data

gs://dataproc-db7603d1-dede-4d40-908e-396370d93558-us/output

Press <Return> to add more arguments

Properties (Optional)

+ Add item

Labels (Optional)

+ Add item

Submit Cancel

Equivalent REST

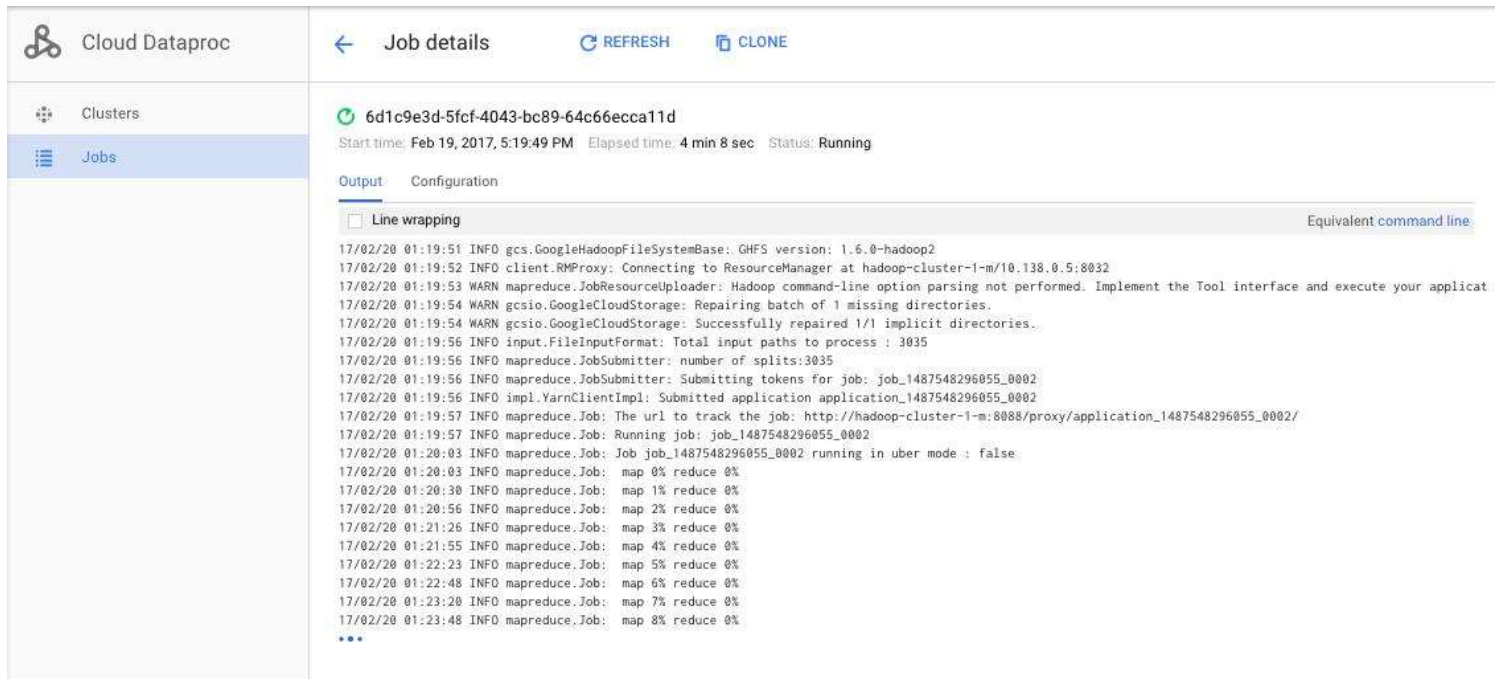
Figure 13: Job submission details

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

Job ID	Type	Cluster	Start time	Elapsed time	Status
<a href="#">46ec16fa-5303-41ba-bb04-168fd5bbc57a</a>	Hadoop	hadoop-cluster-1	Feb 19, 2017, 5:14:20 PM	1 min 16 sec	Succeeded

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.



The screenshot shows the Cloud Dataproc interface. On the left, there's a sidebar with 'Clusters' and 'Jobs'. The 'Jobs' section is selected. The main area displays 'Job details' for job ID '6d1c9e3d-5fcf-4043-bc89-64c66ecca11d'. The job is in 'Running' status, started on Feb 19, 2017, at 5:19:49 PM, and has elapsed 4 minutes and 8 seconds. Below the job details, there's a tab for 'Output' which is selected. It shows a log of Hadoop job progress, including information about the client, ResourceManager, and the progress of map and reduce tasks. The log shows the job is running in uber mode and has completed 8% of the map tasks.

Figure 14: Job progress

- 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.
- 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 libraries.

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)

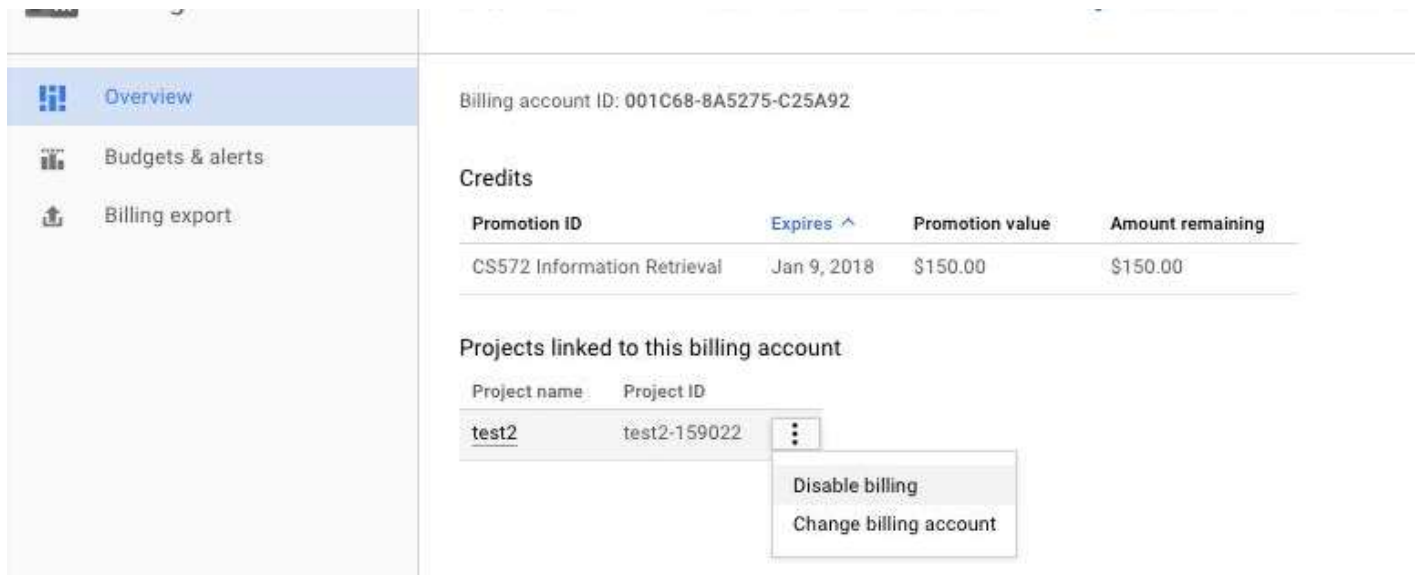


Figure 15: Disabling the billing for the cluster.

### Enable Billing:

**Option 1:** When you navigate to the billing section you will be prompted to select the billing account. Make sure to use your free trial account

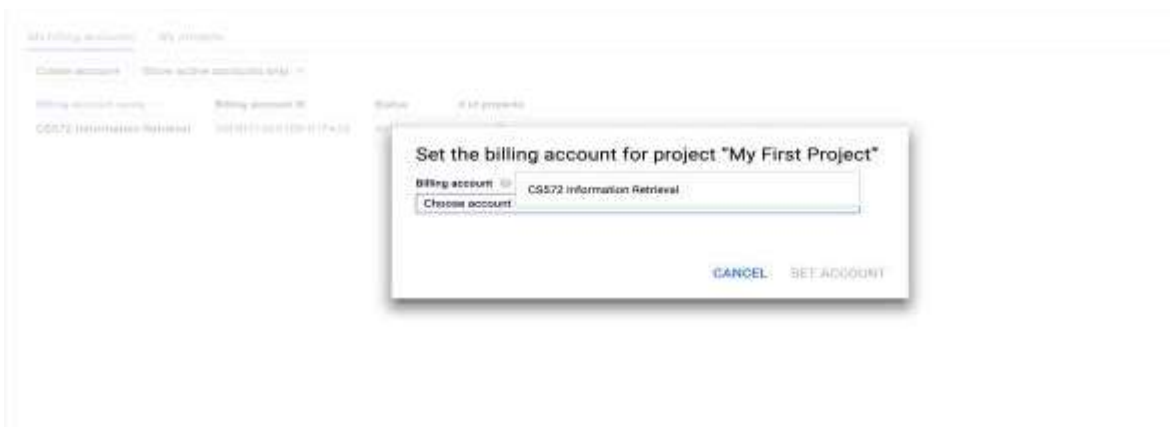


Figure 16: Select the account



## Option 2:

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

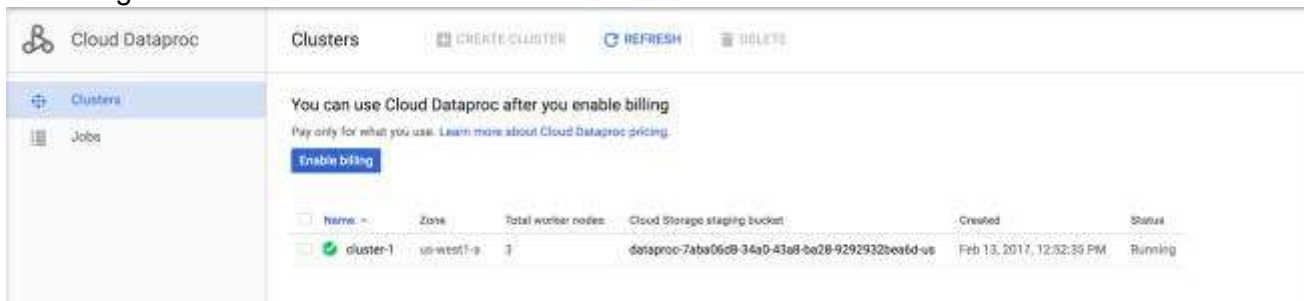


Figure 17: Enable billing

**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.

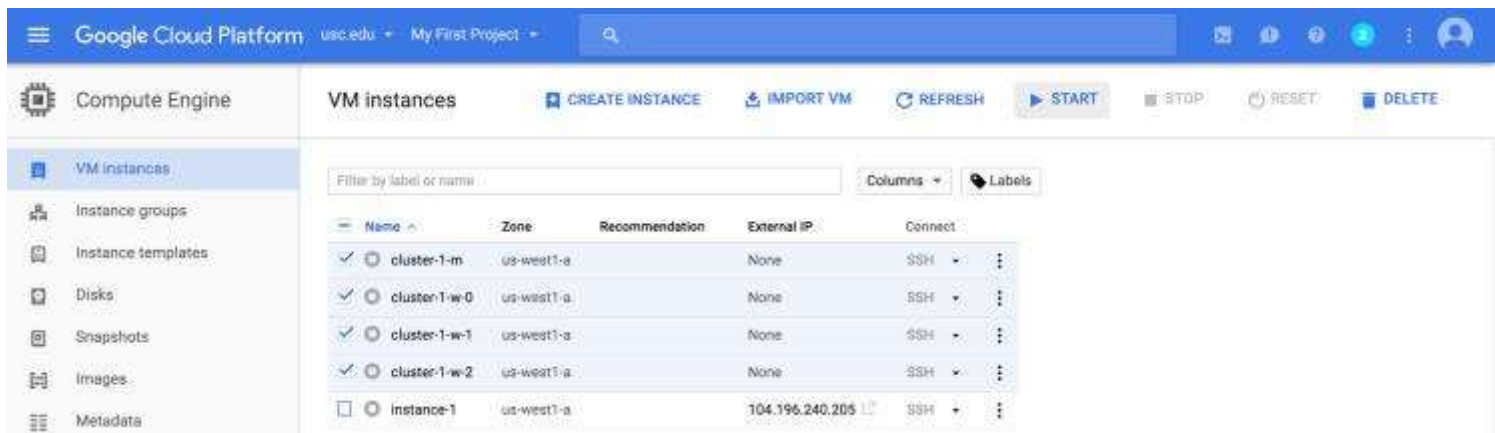


Figure 18: Select all virtual machines associated with the cluster.

## 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.



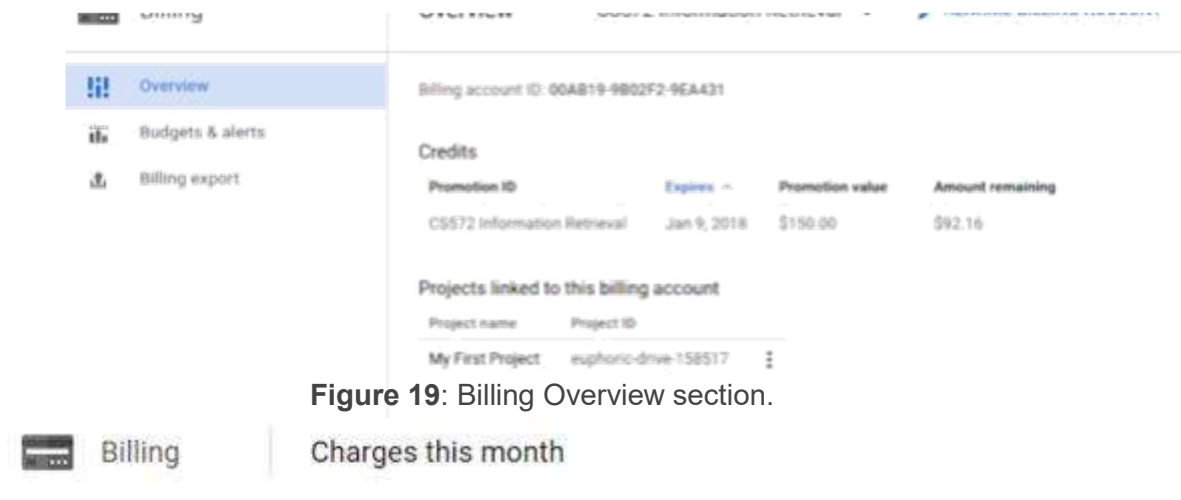


Figure 19: Billing Overview section.

[Go to billing](#)

Product	Resource	Usage	Amount
Google Compute	Standard Intel N1 4 VCPU running in JP	11,280 Minutes	\$49.86
Google Compute	Google Cloud Dataproc running on VM Image CORE	45,120 Minutes	\$7.52
Google Compute	Storage Pd Capacity Jp	8.95 GB-month	\$0.47
Google Compute	Storage Pd Capacity Jp	Credit applied	-\$0.47
Google Compute	Google Cloud Dataproc running on VM Image CORE	Credit applied	-\$7.52
Google Compute	Standard Intel N1 4 VCPU running in JP	Credit applied	-\$49.86
*Estimated charges before taxes, updated daily			Total: \$0.00

Figure 20: Cluster usage cost

## Connecting Your Local Application to GCP DataProc Jobs

You can check the following URL that provides details on how to connect to your DataProc MapReduce jobs from your local java application: <https://developers.google.com/api-client-library/java/apis/dataproc/v1>