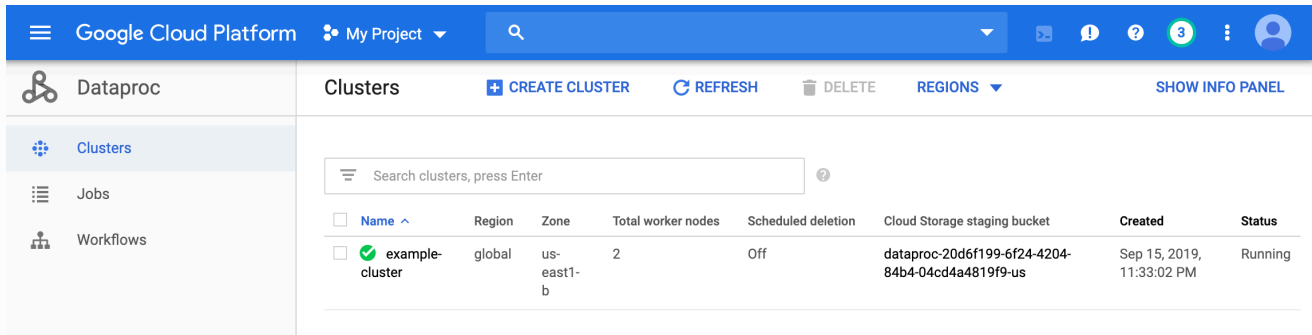


HOMEWORK 0 (E6893)

Jing Qian (jq2282)

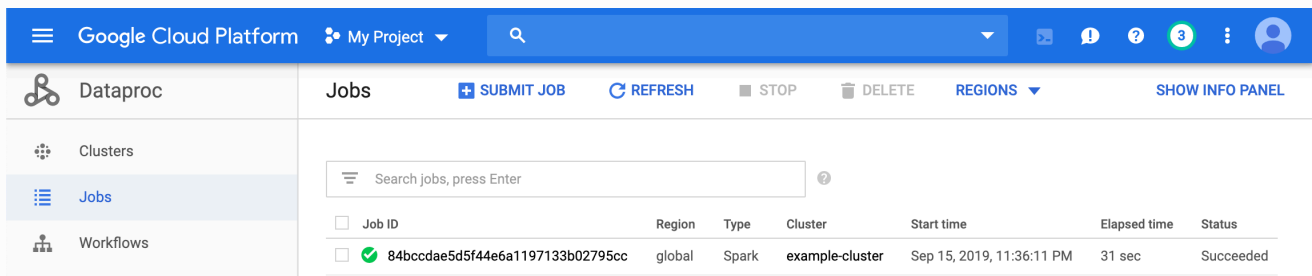
1. Warm-up exercises

1) Screenshots of exercise 3:



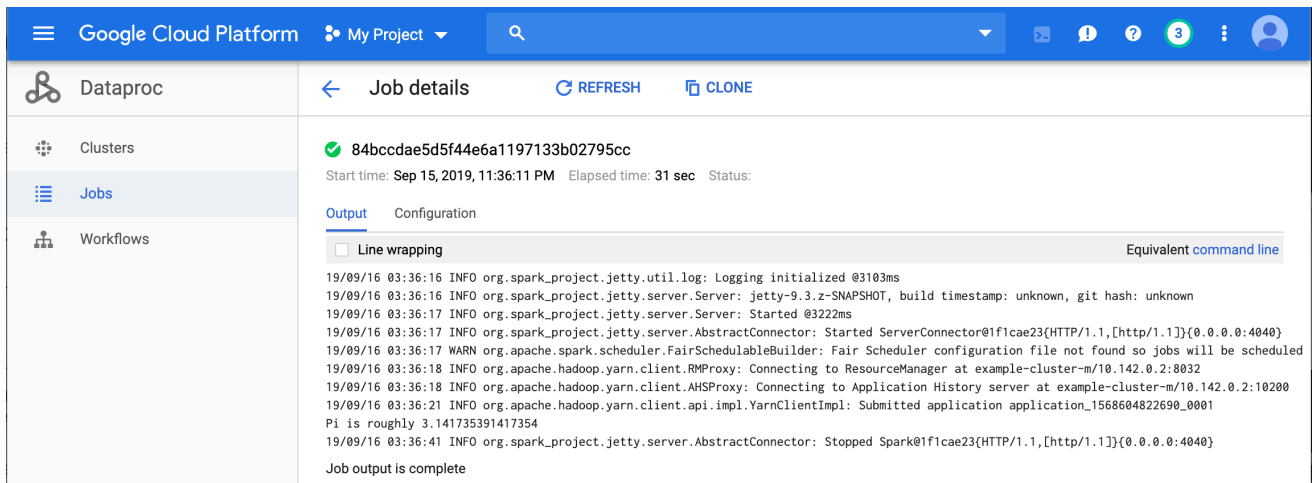
This screenshot shows the Google Cloud Platform interface for the 'Clusters' page. The top navigation bar includes 'Google Cloud Platform', 'My Project', a search bar, and notification icons. The left sidebar shows 'Dataproc' as the selected service, with 'Clusters' highlighted. The main content area has a 'Clusters' header with buttons for 'CREATE CLUSTER', 'REFRESH', 'DELETE', 'REGIONS', and 'SHOW INFO PANEL'. Below this is a search bar and a table of clusters.

<input type="checkbox"/>	Name ^	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage staging bucket	Created	Status
<input checked="" type="checkbox"/>	example-cluster	global	us-east1-b	2	Off	dataproc-20d6f199-6f24-4204-84b4-04cd4a4819f9-us	Sep 15, 2019, 11:33:02 PM	Running



This screenshot shows the Google Cloud Platform interface for the 'Jobs' page. The top navigation bar is the same as the previous screenshot. The left sidebar shows 'Dataproc' as the selected service, with 'Jobs' highlighted. The main content area has a 'Jobs' header with buttons for 'SUBMIT JOB', 'REFRESH', 'STOP', 'DELETE', 'REGIONS', and 'SHOW INFO PANEL'. Below this is a search bar and a table of jobs.

<input type="checkbox"/>	Job ID	Region	Type	Cluster	Start time	Elapsed time	Status
<input checked="" type="checkbox"/>	84bccdae5d5f44e6a1197133b02795cc	global	Spark	example-cluster	Sep 15, 2019, 11:36:11 PM	31 sec	Succeeded



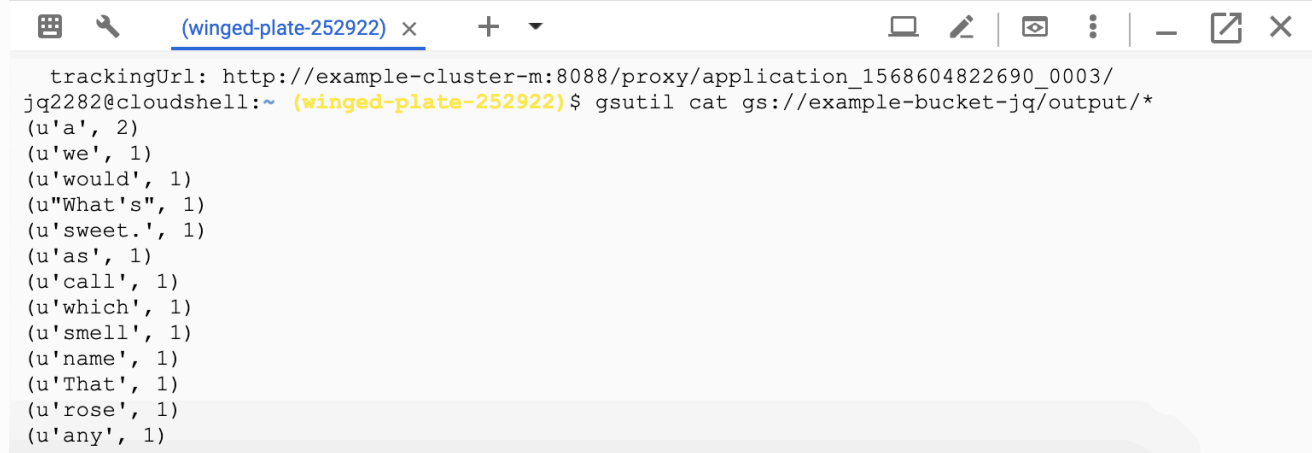
This screenshot shows the Google Cloud Platform interface for the 'Job details' page. The top navigation bar is the same as the previous screenshots. The left sidebar shows 'Dataproc' as the selected service, with 'Jobs' highlighted. The main content area has a 'Job details' header with buttons for 'REFRESH' and 'CLONE'. Below this is the job ID '84bccdae5d5f44e6a1197133b02795cc' and its status 'Succeeded'. The 'Output' tab is selected, showing a log of messages.

Start time: Sep 15, 2019, 11:36:11 PM Elapsed time: 31 sec Status: Succeeded

Output Configuration

☐ Line wrapping Equivalent command line

```
19/09/16 03:36:16 INFO org.spark_project.jetty.util.log: Logging initialized @3103ms
19/09/16 03:36:16 INFO org.spark_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown
19/09/16 03:36:17 INFO org.spark_project.jetty.server.Server: Started @3222ms
19/09/16 03:36:17 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@1f1cae23(HTTP/1.1,[http/1.1]){0.0.0.0:4040}
19/09/16 03:36:17 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair Scheduler configuration file not found so jobs will be scheduled
19/09/16 03:36:18 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at example-cluster-m/10.142.0.2:8032
19/09/16 03:36:18 INFO org.apache.hadoop.yarn.client.AHSPProxy: Connecting to Application History server at example-cluster-m/10.142.0.2:10200
19/09/16 03:36:21 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1568604822690_0001
Pi is roughly 3.141735391417354
19/09/16 03:36:41 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@1f1cae23(HTTP/1.1,[http/1.1]){0.0.0.0:4040}
Job output is complete
```

A screenshot of a terminal window with a title bar containing icons for window management and a tab labeled "(winged-plate-252922) x". The terminal shows a shell prompt "jq2282@cloudshell:~" followed by the command "gsutil cat gs://example-bucket-jq/output/*". The output consists of 15 lines of text, each representing a word and its frequency in parentheses. The words are: 'a' (2), 'we' (1), 'would' (1), 'What's' (1), 'sweet.' (1), 'as' (1), 'call' (1), 'which' (1), 'smell' (1), 'name' (1), 'That' (1), 'rose' (1), and 'any' (1).

```
trackingUrl: http://example-cluster-m:8088/proxy/application_1568604822690_0003/
jq2282@cloudshell:~ (winged-plate-252922) $ gsutil cat gs://example-bucket-jq/output/*
(u'a', 2)
(u'we', 1)
(u'would', 1)
(u'What's", 1)
(u'sweet.', 1)
(u'as', 1)
(u'call', 1)
(u'which', 1)
(u'smell', 1)
(u'name', 1)
(u'That', 1)
(u'rose', 1)
(u'any', 1)
```

2)

Transformations in Exercise3: filter().

Actions in Exercise3: count(). The RDD operation that triggers the program to execute is the actions and hence "count()".

Transformations in Exercise4: flatMap(), map(), reduceByKey().

Actions in Exercise4: saveAsTextFile(). The RDD operation that triggers the program to execute is the actions and hence "saveAsTextFile()".

*Exercise 3 is an inside example and corresponding code is found at: <https://spark.apache.org/examples.html>. The code for Exercise 4 is provided in the given link: <https://cloud.google.com/dataproc/docs/tutorials/gcs-connector-spark-tutorial>.

2. NYC Bike expert

1) There are 843 unique station_ids in this dataset.

Google Cloud Platform My Project

BigQuery FEATURES & INFO SHORTCUTS + COMPOSE NEW QUERY

Query history

Query editor

```
1 SELECT Count(DISTINCT station_id) FROM Bikes.Bikes
```

Valid.

Run Save query Save view Schedule query More

This query will process 6.6 KB when run.

Query results SAVE RESULTS EXPLORE WITH DATA STUDIO

Query complete (1.7 sec elapsed, 6.6 KB processed)

Job information Results JSON Execution details

Row	Count
1	843

2) The largest capacity for a station is 79. The *station_id* of stations that have the largest capacity are: 445, 422, 501.

Google Cloud Platform My Project

BigQuery FEATURES & INFO SHORTCUTS + COMPOSE NEW QUERY

Query history

Query editor

```
1 SELECT station_id, capacity
2 FROM Bikes.Bikes
3 ORDER BY capacity DESC
4 LIMIT 5
```

Valid.

Run Save query Save view Schedule query More

This query will process 13.2 KB when run.

Query results SAVE RESULTS EXPLORE WITH DATA STUDIO

Query complete (0.5 sec elapsed, 13.2 KB processed)

Job information Results JSON Execution details

Row	station_id	capacity
1	422	79
2	501	79
3	445	79
4	442	77

3) The total number of bikes available in region_id 70 is 394.

Google Cloud Platform My Project

BigQuery FEATURES & INFO SHORTCUTS + COMPOSE NEW QUERY

Query history

Query editor

```

1 SELECT Sum(num_bikes_available)
2 FROM Bikes.Bikes AS B
3 WHERE B.region_id = 70

```

Valid.

Run Save query Save view Schedule query More

This query will process 13.2 KB when run.

Query results SAVE RESULTS EXPLORE WITH DATA STUDIO

Query complete (0.3 sec elapsed, 13.2 KB processed)

Job information Results JSON Execution details

Row	f0_
1	394

3. Understanding William Shakespeare

1) The top 5 frequent words without any text preprocessing are: (('the', 620), ('and', 427), ('of', 396), ('to', 367), ('I', 326)).

Jupyter Shakespeare (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark

Run Code

```

In [22]: # Import text from bucket shakes0
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
url = "gs://shakes0/shakes.txt"
txt = sc.textFile(url)

In [25]: # Method 1: learn from wordCounts, without text preprocessing
words = txt.flatMap(lambda line: line.split())
# words.collect()
wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda count1, c
wordCounts.takeOrdered(5, key=lambda x: -x[1])

Out[25]: [(u'the', 620), (u'and', 427), (u'of', 396), (u'to', 367), (u'I', 326)]

```

2) Top 5 frequent words by filtering out stop words provided by NLTK package are: (('I', 346), ('And', 170), ('not', 165), ('with', 143), ('be', 138)). Notice here I also removed the punctuations or the counted words will include punctuations.

```
In [26]: # Method 2: filtering out stop words provided by NLTK package
import nltk

def tokenizer(x):
    from nltk.tokenize import RegexpTokenizer
    token = RegexpTokenizer(r'\w+')
    return token.tokenize(x)

tokens = txt.flatMap(tokenizer)
```

```
In [27]: nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = tokens.filter(lambda word : word[0] not in stop_words)

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
```

```
In [28]: wordCount2 = tokens.map(lambda word: (word, 1)).reduceByKey(lambda count1,
wordCount2.takeOrdered(5, key=lambda x: -x[1])
```

```
Out[28]: [(u'I', 346), (u'And', 170), (u'not', 165), (u'with', 143), (u'be', 138)]
```

The result will differ if we change the splitting method, like following:

```
In [2]: # Import text from bucket
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
url = "gs://shakespe/shakes.txt"
txt = sc.textFile(url)

# Method 2.2: filter stopwords with nltk but with different splitting method
words = txt.flatMap(lambda line: line.split())
```

```
In [3]: import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
words = words.filter(lambda word : word[0] not in stop_words)
wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda count1, c
wordCounts.takeOrdered(5, key=lambda x: -x[1])

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
```

```
Out[3]: [(u'I', 326), (u'And', 169), (u'not', 142), (u'with', 141), (u'Macb.', 13
7)]
```