

ML1020 – Assignment 2

Nvidia Labs and Airflow Wordcount

Submitted by: Michael Vasiliou

Nvidia Labs Completion Certificate

 DEEP
LEARNING
INSTITUTE

CERTIFICATE OF COMPETENCY

This NVIDIA DLI Certificate has been awarded to

Michael Vasiliou

for the successful completion of
Fundamentals of Deep Learning



Will Ramey
Senior Director, Developer Programs

February 2, 2022

More about Michael Vasiliou's accomplishment



Michael Vasiliou
chief_wiggum @ NVIDIA Deep
Learning Institute

Certificate ID Number:
c64e124256ba4afa8410bc93141eb4f5

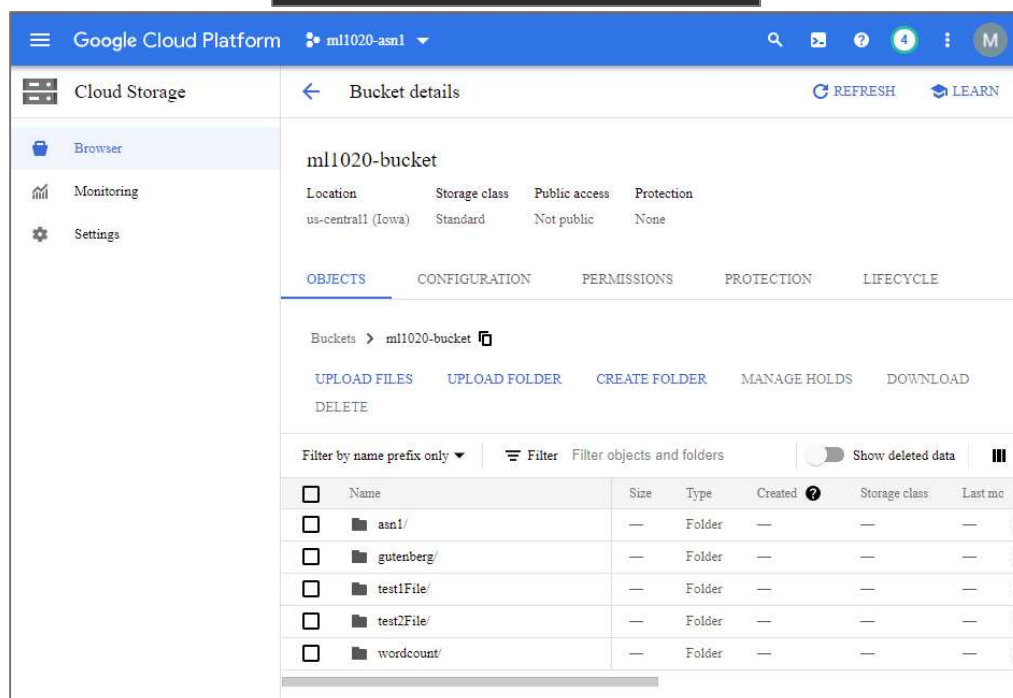
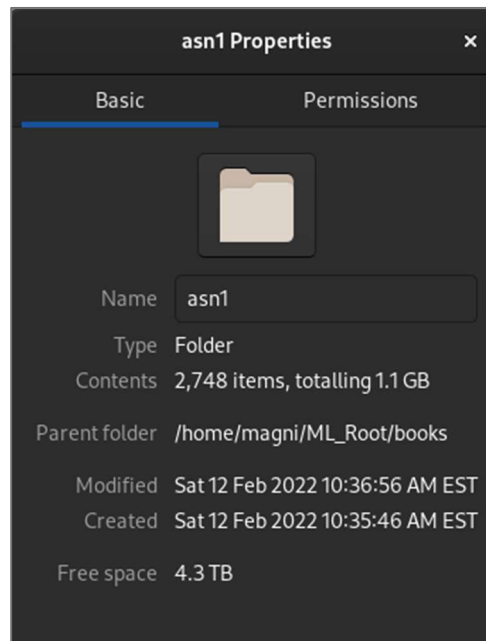
Course:
Fundamentals of Deep Learning

Issued On:
February 2, 2022

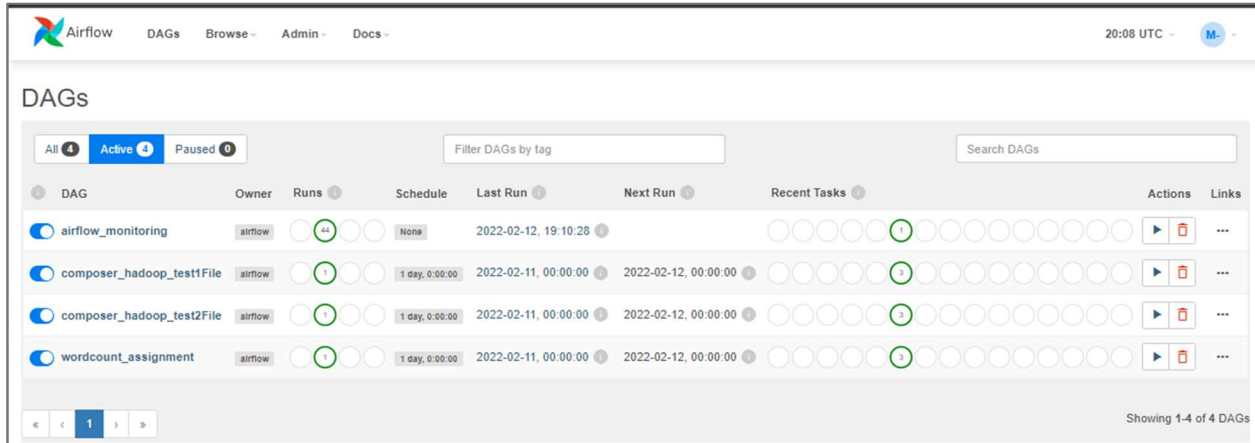
Deploy Wordcount using Airflow in GCP

Dataset

For the large dataset I used python to download books from Project Gutenberg. The download script is included at the end of the PDF. The final set of books used for the wordcount consisted of 2,748 files totaling over 1GB in size.

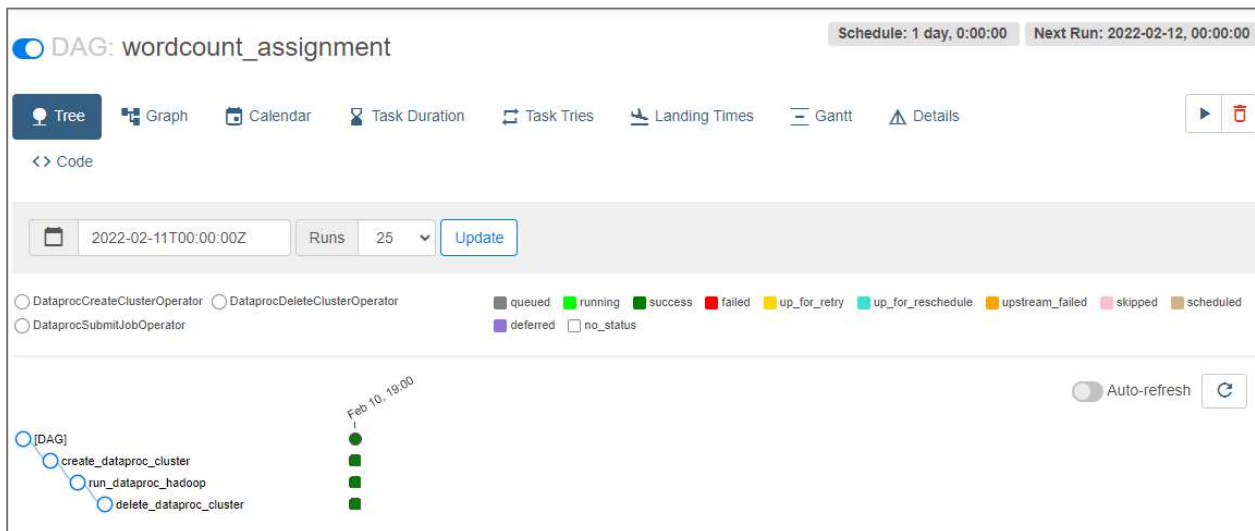


Airflow deployment



The screenshot shows the Airflow web interface's DAGs overview page. At the top, there are navigation links for DAGs, Browse, Admin, and Docs. The status bar indicates the time is 20:08 UTC. The main section is titled 'DAGs' and includes filters for 'All' (4), 'Active' (4), and 'Paused' (0). A search bar for DAGs is also present. Below the filters is a table listing DAGs with columns for DAG name, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. The table shows four DAGs: 'airflow_monitoring', 'composer_hadoop_test1File', 'composer_hadoop_test2File', and 'wordcount_assignment'. Each DAG has a status icon, a run count, a schedule, and a list of recent task statuses. The 'wordcount_assignment' DAG is highlighted as the selected one.

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
airflow_monitoring	airflow	1	None	2022-02-12, 19:10:28		1	[Play] [Stop]	...
composer_hadoop_test1File	airflow	1	1 day, 0:00:00	2022-02-11, 00:00:00	2022-02-12, 00:00:00	1	[Play] [Stop]	...
composer_hadoop_test2File	airflow	1	1 day, 0:00:00	2022-02-11, 00:00:00	2022-02-12, 00:00:00	1	[Play] [Stop]	...
wordcount_assignment	airflow	1	1 day, 0:00:00	2022-02-11, 00:00:00	2022-02-12, 00:00:00	1	[Play] [Stop]	...



The screenshot shows the details page for the 'wordcount_assignment' DAG. The top bar indicates the schedule is '1 day, 0:00:00' and the next run is '2022-02-12, 00:00:00'. Below the title, there are tabs for Tree, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, and Details. The 'Tree' tab is selected. A date picker shows '2022-02-11T00:00:00Z' and a dropdown for 'Runs' is set to '25'. An 'Update' button is next to it. A legend at the bottom shows various task statuses: queued, running, success, failed, up_for_retry, up_for_reschedule, upstream_failed, skipped, scheduled, deferred, and no_status. The DAG tree shows the following tasks: [DAG], create_dataproc_cluster, run_dataproc_hadoop, and delete_dataproc_cluster. A vertical timeline on the right shows the execution progress, with a label 'Feb 10, 19:00'.

DAG: wordcount_assignment Schedule: 1 day, 0:00:00 Next Run: 2022-02-12, 00:00:00

Tree Graph Calendar Task Duration Task Tries Landing Times Gantt Details

<> Code

2022-02-11T00:00:00Z Runs 25 Update

DataprocCreateClusterOperator DataprocDeleteClusterOperator DataprocSubmitJobOperator

queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred no_status

Feb 10, 19:00

[DAG] create_dataproc_cluster run_dataproc_hadoop delete_dataproc_cluster

Auto-refresh

Airflow DAGs Browse Admin Docs 19:02 UTC M-

DAG: wordcount_assignment success Schedule: 1 day, 0:00:00 Next Run: 2022-02-12, 00:00:00

Tree Graph **Calendar** Task Duration Task Tries Landing Times Gantt Details

<> Code

2022-02-11T00:00:01Z Runs 25 Run scheduled__2022-02-11T00:00:00+00:00 Find Task...

Layout Left > Right Update

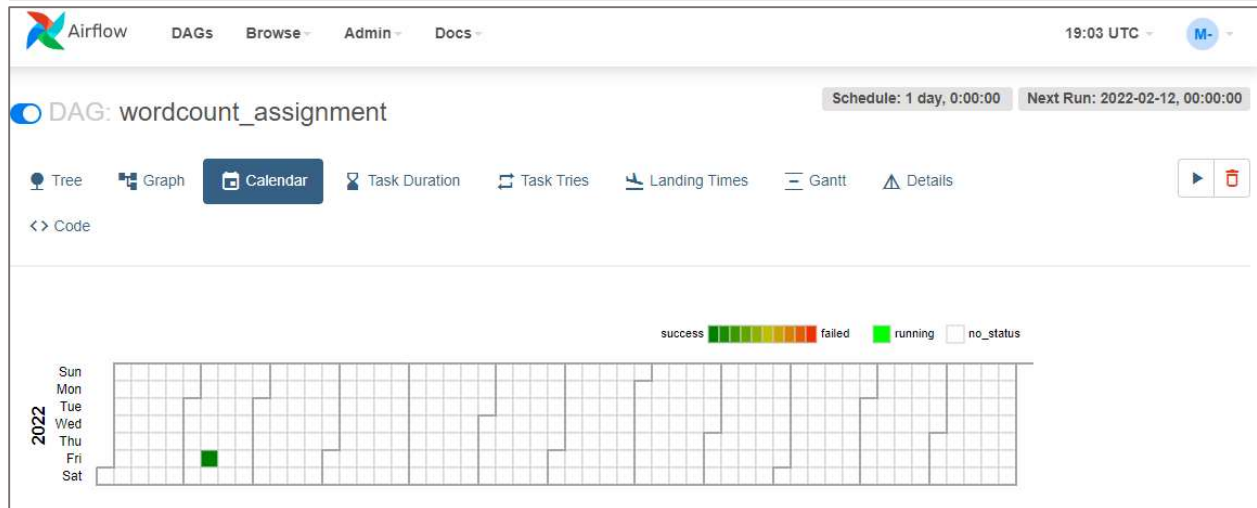
DataprocCreateClusterOperator DataprocDeleteClusterOperator
 DataprocSubmitJobOperator

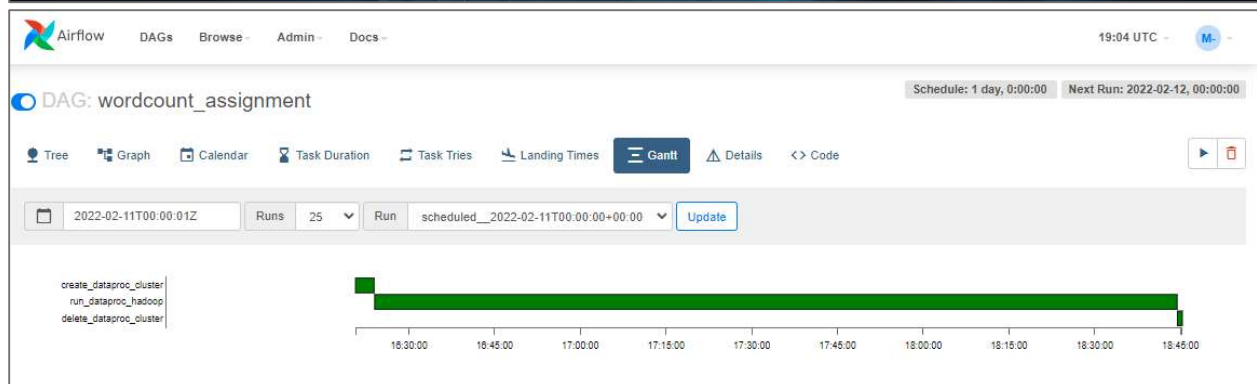
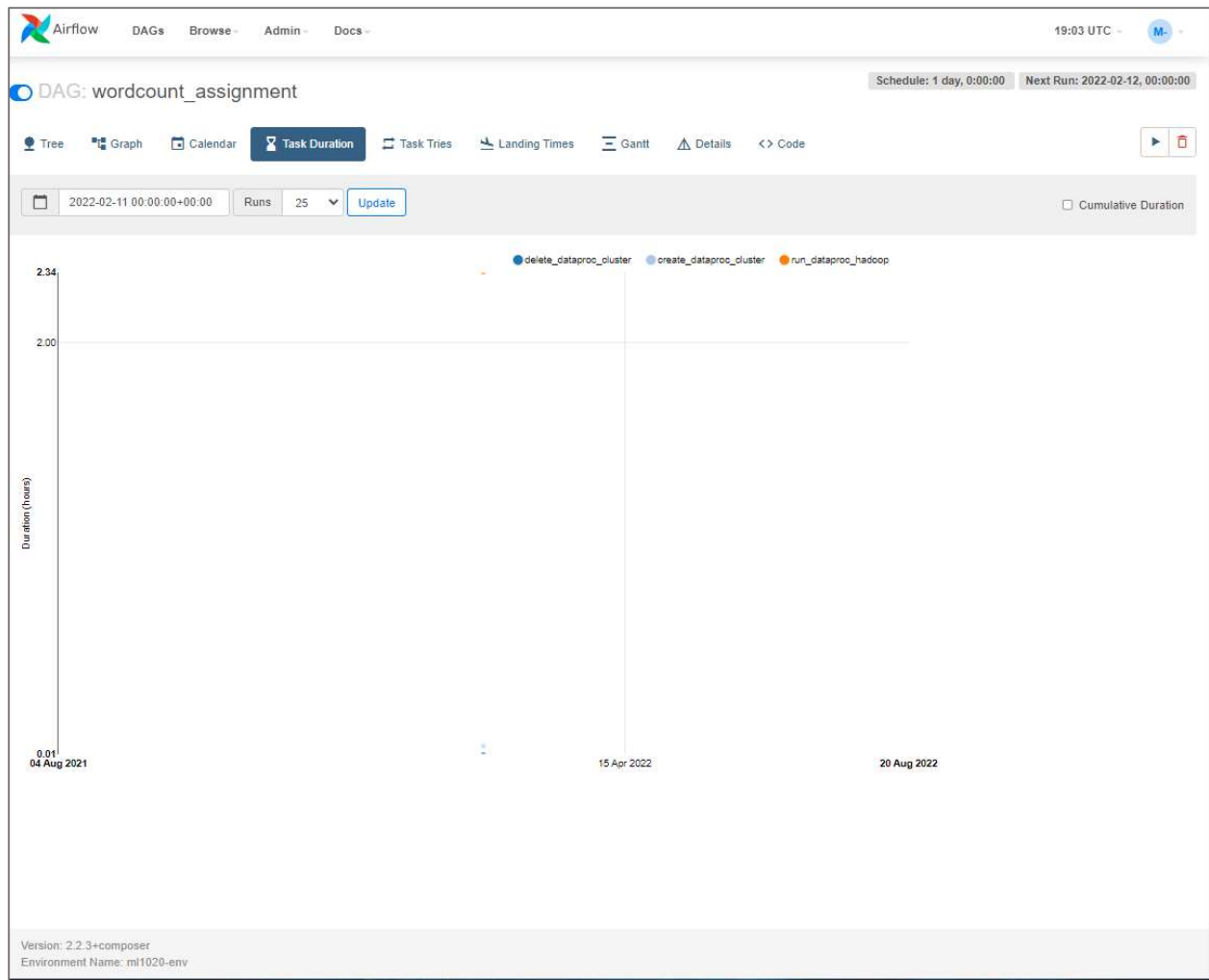
queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred
 no_status

Auto-refresh

```

graph LR
    A[create_dataproc_cluster] --> B[run_dataproc_hadoop]
    B --> C[delete_dataproc_cluster]
  
```





[DAGs](#)
[Browse](#)
[Admin](#)
[Docs](#)

19:04 UTC

DAG: wordcount_assignment

Schedule: 1 day, 0:00:00 Next Run: 2022-02-12, 00:00:00

[Tree](#)
[Graph](#)
[Calendar](#)
[Task Duration](#)
[Task Tries](#)
[Landing Times](#)
[Gantt](#)
[Details](#)
[Code](#)

DAG Details

success 3

Schedule Interval	1 day, 0:00:00
Catchup	True
Start Date	None
End Date	None
Max Active Runs	0 / 15
Concurrency	15
Default Args	{'email_on_failure': False, 'email_on_retry': False, 'location': 'us-central1', 'project_id': 'ml1020-asnt1', 'retries': 1, 'retry_delay': datetime.timedelta(seconds=300), 'start_date': DateTime(2022, 2, 11, 0, 0, 0, tzinfo=Timezone('UTC'))}
Tasks Count	3
Task IDs	['create_dataproc_cluster', 'run_dataproc_hadoop', 'delete_dataproc_cluster']
Relative file location	hadoop_assignment.py
Owner	airflow
DAG Run Timeout	None
Tags	None

Wordcount DAG

The wordcount DAG I used was based on a tutorial from google :
<https://cloud.google.com/composer/docs/tutorials/hadoop-wordcount-job>

```
1 # ML1020 - Assignment 2 (DAG for wordcount)
2
3 """Example Airflow DAG that creates a Cloud Dataproc
4 cluster, runs the Hadoop
5 wordcount example, and deletes the cluster.
6
7 This DAG relies on three Airflow variables
8 https://airflow.apache.org/docs/apache-airflow/stable/
9 concepts/variables.html
10 * gcp_project - Google Cloud Project to use for the
11 Cloud Dataproc cluster.
12 * gce_zone - Google Compute Engine zone where Cloud
13 Dataproc cluster should be
14 created.
15 * gcs_bucket - Google Cloud Storage bucket to use for
16 result of Hadoop job.
17 See https://cloud.google.com/storage/docs/creating-
18 buckets for creating a
19 bucket.
20 """
21
22 import datetime
23 import os
24
25 from airflow import models
26 from airflow.providers.google.cloud.operators import
27     dataproc
28 from airflow.utils import trigger_rule
29
30 # Output file for Cloud Dataproc job.
31 # If you are running Airflow in more than one time zone
32 # see https://airflow.apache.org/docs/apache-airflow/
33 # stable/timezone.html
34 # for best practices
35 output_file = os.path.join(
36     models.Variable.get('gcs_bucket'), 'wordcount',
37     datetime.datetime.now().strftime('%Y%m%d-%H%M%S'
38 )) + os.sep
```

```
30 # Path to Hadoop wordcount example available on every
    Dataproc cluster.
31 WORDCOUNT_JAR = (
32     'file:///usr/lib/hadoop-mapreduce/hadoop-mapreduce-
    examples.jar'
33 )
34 # Arguments to pass to Cloud Dataproc job.
35 #input_file = 'gs://pub/shakespeare/rose.txt'
36 input_file = 'gs://ml1020-bucket/asn1/*.txt'
37 wordcount_args = ['wordcount', input_file, output_file]
38
39 HADOOP_JOB = {
40     "reference": {"project_id": models.Variable.get('
    gcp_project')},
41     "placement": {"cluster_name": 'composer-hadoop-
    tutorial-cluster-{{ ds_nodash }}'},
42     "hadoop_job": {
43         "main_jar_file_uri": WORDCOUNT_JAR,
44         "args": wordcount_args,
45     },
46 }
47
48 CLUSTER_CONFIG = {
49     "master_config": {
50         "num_instances": 1,
51         "machine_type_uri": "n1-standard-2"
52     },
53     "worker_config": {
54         "num_instances": 2,
55         "machine_type_uri": "n1-standard-2"
56     },
57 }
58
59 yesterday = datetime.datetime.combine(
60     datetime.datetime.today() - datetime.timedelta(1),
61     datetime.datetime.min.time())
62
63 default_dag_args = {
```



```

64     # Setting start date as yesterday starts the DAG
        immediately when it is
65     # detected in the Cloud Storage bucket.
66     'start_date': yesterday,
67     # To email on failure or retry set 'email' arg to
        your email and enable
68     # emailing here.
69     'email_on_failure': False,
70     'email_on_retry': False,
71     # If a task fails, retry it once after waiting at
        least 5 minutes
72     'retries': 1,
73     'retry_delay': datetime.timedelta(minutes=5),
74     'project_id': models.Variable.get('gcp_project'),
75     'location': models.Variable.get('gce_region'),
76
77 }
78
79
80 with models.DAG(
81     'wordcount_assignment',
82     # Continue to run DAG once per day
83     schedule_interval=datetime.timedelta(days=1),
84     default_args=default_dag_args) as dag:
85
86     # Create a Cloud Dataproc cluster.
87     create_dataproc_cluster = dataproc.
        DataprocCreateClusterOperator(
88         task_id='create_dataproc_cluster',
89         # Give the cluster a unique name by appending
            the date scheduled.
90         # See https://airflow.apache.org/docs/apache-
            airflow/stable/macros-ref.html
91         cluster_name='composer-hadoop-tutorial-cluster
            -{{ ds_nodash }}',
92         cluster_config=CLUSTER_CONFIG,
93         region=models.Variable.get('gce_region'))
94

```

```
95     # Run the Hadoop wordcount example installed on  
    the Cloud Dataproc cluster  
96     # master node.  
97     run_wordcount = dataproc.DataprocSubmitJobOperator  
    (  
98         task_id='run_dataproc_hadoop',  
99         job=HADOOP_JOB)  
100  
101     # Delete Cloud Dataproc cluster.  
102     delete_dataproc_cluster = dataproc.  
    DataprocDeleteClusterOperator(  
103         task_id='delete_dataproc_cluster',  
104         cluster_name='composer-hadoop-tutorial-cluster  
-{{ ds_nodash }}',  
105         region=models.Variable.get('gce_region'),  
106         # Setting trigger_rule to ALL_DONE causes the  
    cluster to be deleted  
107         # even if the Dataproc job fails.  
108         trigger_rule=trigger_rule.TriggerRule.ALL_DONE  
    )  
109  
110     # Define DAG dependencies.  
111     create_dataproc_cluster >> run_wordcount >>  
    delete_dataproc_cluster
```

Top 100 Words generated through wordcount

Rank	Word	Frequency
1	the	10448736
2	of	6332524
3	and	5466904
4	to	4908685
5	a	3478486
6	in	2984453
7	that	1904329
8	was	1829951
9	I	1742820
10	his	1530557
11	he	1480257
12	with	1461410
13	for	1282440
14	is	1241116
15	as	1219036
16	had	1158791
17	it	1111759
18	by	1015730
19	not	1003920
20	be	979678
21	at	956803
22	The	916316
23	on	883587
24	you	846019
25	which	835139
26	her	811104
27	have	760316
28	from	748115
29	or	721939
30	this	709206
31	but	695898
32	my	639447
33	all	617144
34	were	601831
35	they	583038
36	she	576905
37	are	543268
38	their	528297
39	an	486094
40	him	470263
41	so	468620
42	we	449173
43	one	439543

Top 100 Words generated through wordcount

Rank	Word	Frequency
44	who	439134
45	would	431521
46	been	426547
47	no	389545
48	He	376006
49	will	360991
50	me	350011
51	when	336980
52	it's	335833
53	any	325539
54	if	323063
55	more	303836
56	into	290228
57	there	288286
58	said	287961
59	them	286386
60	has	282995
61	our	282370
62	could	281939
63	than	277181
64	your	274238
65	out	272286
66	what	270611
67	very	270530
68	It	270511
69	do	264589
70	up	264569
71	some	263236
72	upon	259387
73	its	246077
74	other	239258
75	about	236328
76	only	230184
77	And	229084
78	may	216507
79	Project	214766
80	should	214479
81	did	214087
82	little	212882
83	But	212351
84	great	208568
85	such	205828
86	like	204911

Top 100 Words generated through wordcount

Rank	Word	Frequency
87	can	204441
88	made	204240
89	must	194762
90	these	188782
91	man	186121
92	two	181996
93	time	179984
94	Mr.	179729
95	much	177884
96	after	174424
97	In	173923
98	most	170795
99	before	169584
100	where	169391

```
1 # -*- coding: utf-8 -*-
2
3 # Sergei Bugrov
4 # 7-9-17
5 #
6 # Downloads all available books in English language in
7 # .txt format from http://www.gutenberg.org,
8 # unpacks them from .zip archives, saves them to ../
9 # books/ folder, and deletes .zip files.
10 #
11 # usage : python gutenberg.py
12 #
13 # python version : 3.6.1
14
15 import requests, bs4, os, errno, zipfile, glob
16 from urllib.request import urlretrieve
17
18 def main():
19     if not os.path.exists('books/'):
20         try:
21             os.makedirs('books/')
22         except OSError as e:
23             if e.errno != errno.EEXIST:
24                 raise
25
26     # STEP 1. BUILD A LIST OF URLS
27
28     urls_to_books = []
29
30     if not os.path.exists('urls_to_books.txt'):
31
32         page_w_books_url = 'http://www.gutenberg.org/
33 robot/harvest?filetypes[]=txt&langs[]=en'
34
35         while 1 == 1:
```

```
36         is_last_page = False
37
38         print('Reading page: ' + page_w_books_url)
39
40         page_w_books = requests.get(
41             page_w_books_url, timeout=20.0)
42
43         if page_w_books:
44             page_w_books = bs4.BeautifulSoup(
45                 page_w_books.text, "lxml")
46             urls = [el.get('href') for el in
47                     page_w_books.select('body > p > a[href^="http://aleph.
48                     gutenberg.org/"]')]
49             url_to_next_page = page_w_books.
50             find_all('a', string='Next Page')
51
52             if len(urls) > 0:
53                 urls_to_books.append(urls)
54
55                 if url_to_next_page[0]:
56                     page_w_books_url = "http://www.
57                     gutenberg.org/robot/" + url_to_next_page[0].get('href')
58                 else:
59                     is_last_page = True
60
61             if is_last_page:
62                 break
63
64             urls_to_books = [item for sublist in
65                             urls_to_books for item in sublist]
66
67             # Backing up the list of URLs
68             with open('urls_to_books.txt', 'w') as output:
69                 for u in urls_to_books:
70                     output.write('%s\n' % u)
71
72             # STEP 2. DOWNLOAD BOOKS
```

```

67      # If, at some point, Step 2 is interrupted due to
        unforeseen
68      # circumstances (power outage, lost of internet
        connection), replace the number
69      # (value of the variable url_num) below with the
        one you will find in the logfile.log
70      # Example
71      #      logfile.log : Unzipping file #99 books/
        10020.zip
72      #      the number : 99
73      url_num = 0
74
75      if os.path.exists('urls_to_books.txt') and len(
        urls_to_books) == 0:
76          with open('urls_to_books.txt', 'r') as f:
77              urls_to_books = f.read().splitlines()
78
79      for url in urls_to_books[url_num:]:
80
81          dst = 'books/' + url.split('/')[ -1].split('.')[
        ][0].split('-')[0]
82
83          with open('logfile.log', 'w') as f:
84              f.write('Unzipping file #' + str(url_num
        ) + ' ' + dst + '.zip' + '\n')
85
86          if len(glob.glob(dst + '*')) == 0:
87              urlretrieve(url, dst + '.zip')
88
89          with zipfile.ZipFile(dst + '.zip', "r") as
        zip_ref:
90              try:
91                  zip_ref.extractall("books/")
92                  print(str(url_num) + ' ' + dst +
        '.zip ' + 'unzipped successfully!')
93              except NotImplementedError:
94                  print(str(url_num) + ' Cannot
        unzip file:', dst)

```



```
95
96         os.remove(dst + '.zip')
97
98         url_num += 1
99
100
101 if __name__ == '__main__':
102     """
103     The main function is called when gutenberg.py is
104     run from the command line
105     """
106     main()
```