ML1020 – Assignment 2
Nvidia Labs and Airflow Wordcount

Submitted by: Michael Vasiliou

Nvidia Labs Completion Certificate

NVIDIA DEEP LEARNING INSTITUTE

CERTIFICATE OF COMPETENCY

This NVIDIA DLI Certificate has been awarded to

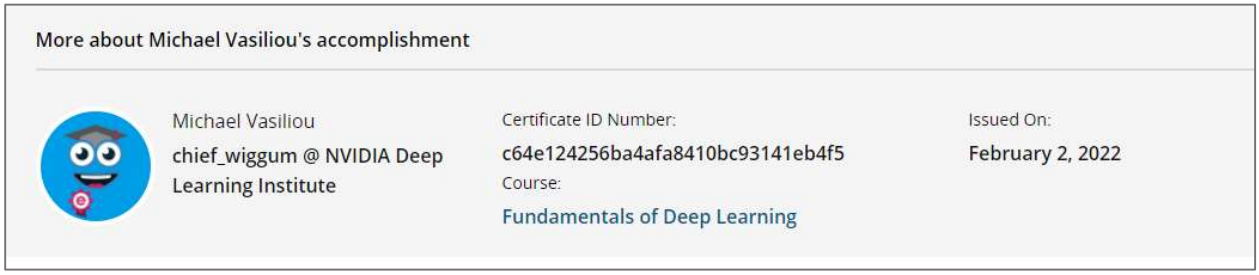
Michael Vasiliou

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

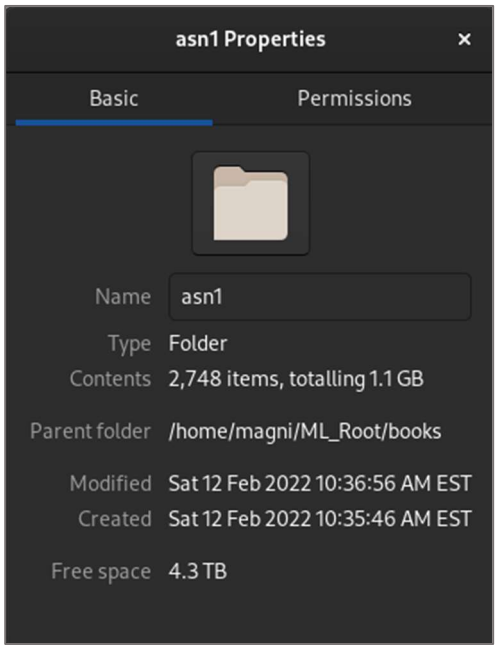
Course:
Fundamentals of Deep Learning

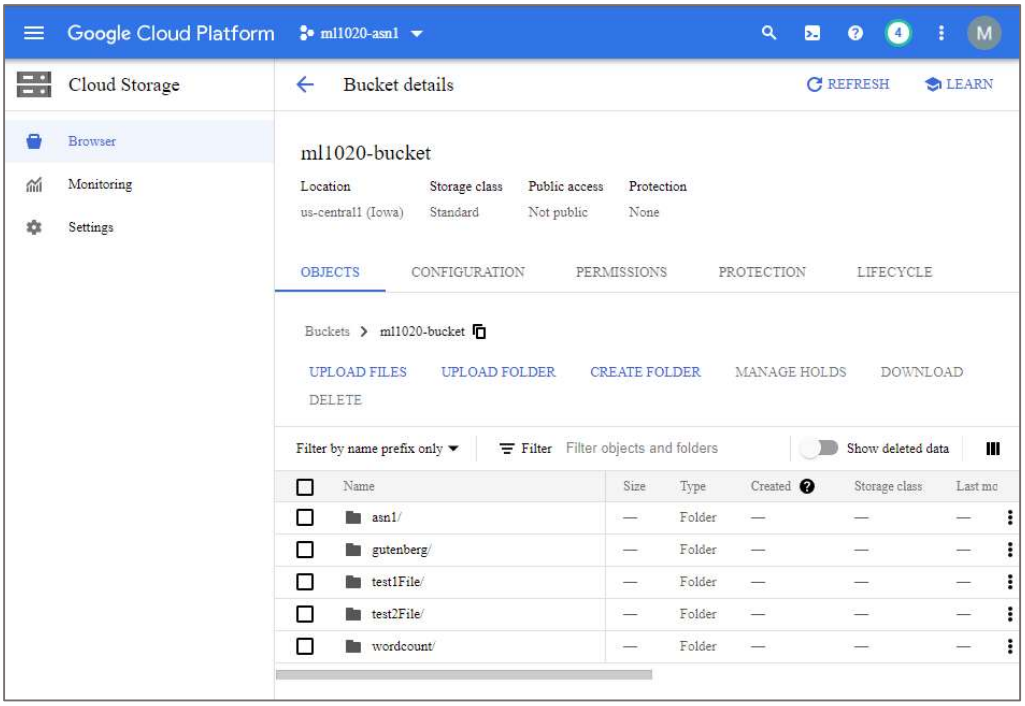
Issued On:
February 2, 2022

Deploy Wordcount using Airflow in GCP

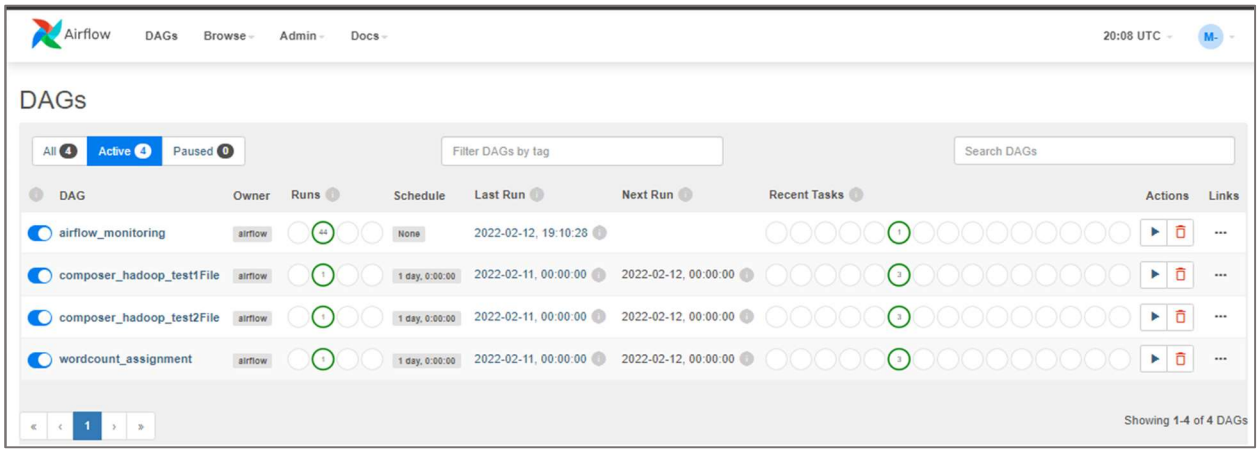
Dataset

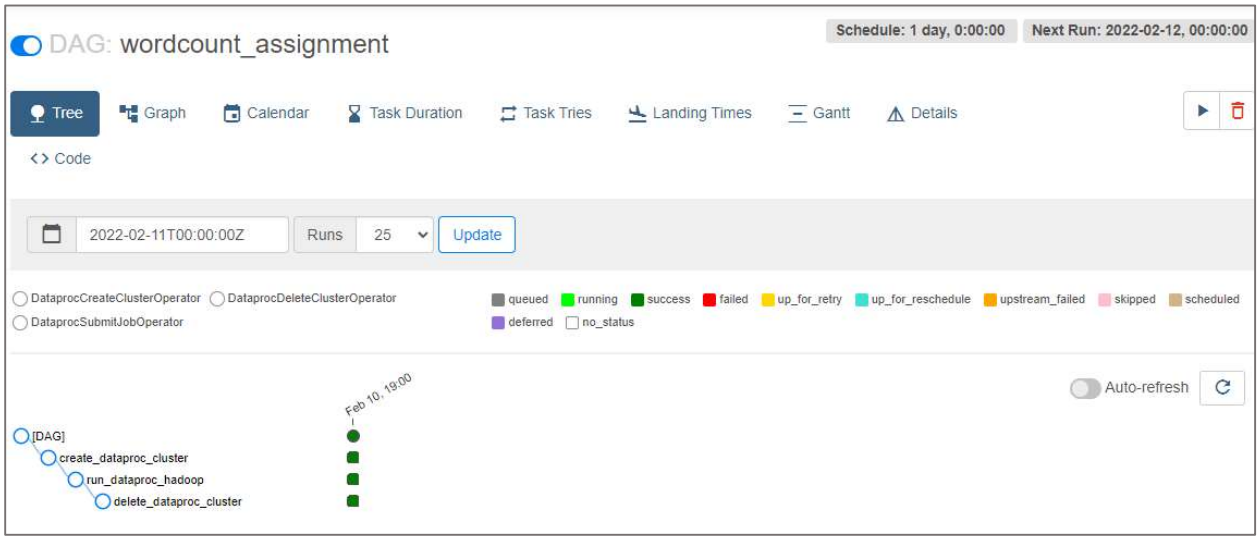
For the large dataset I used python to download books from Project Gutenberg. The download script is included at the end of the PDF. The final set of books used for the wordcount consisted of 2,748 files totaling over 1GB in size.

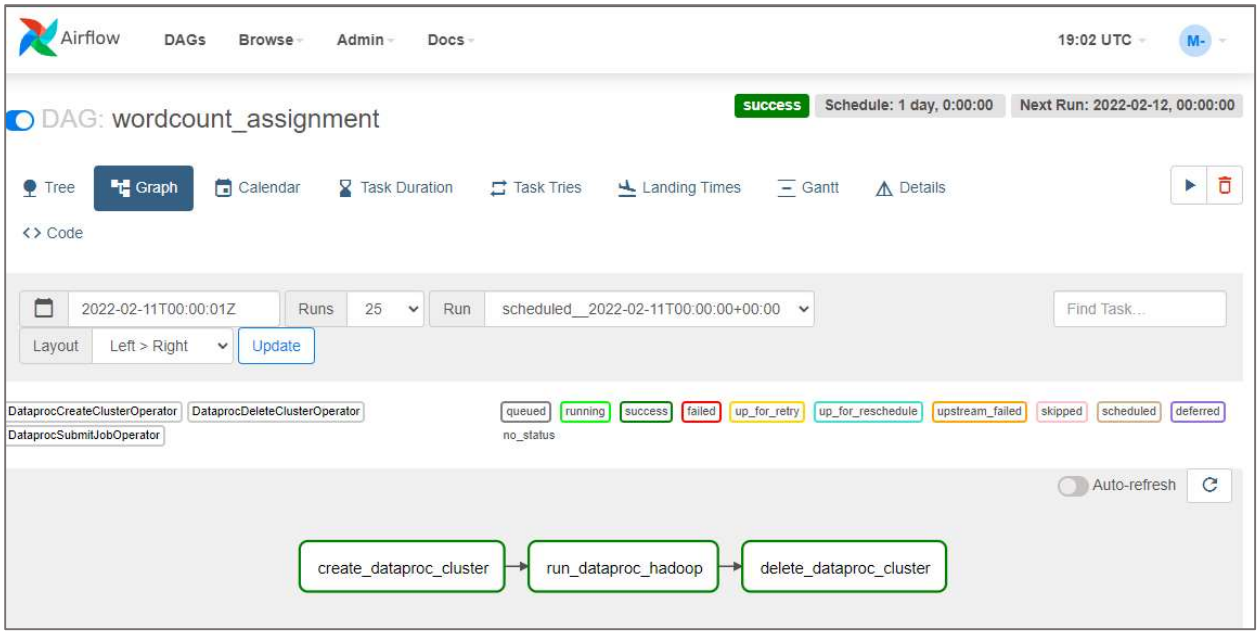


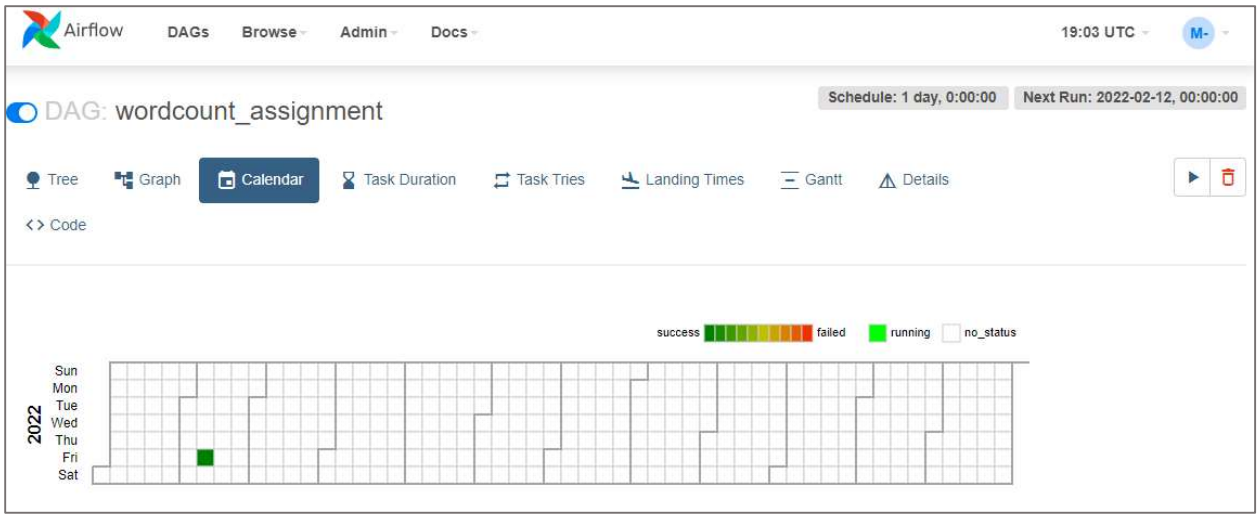


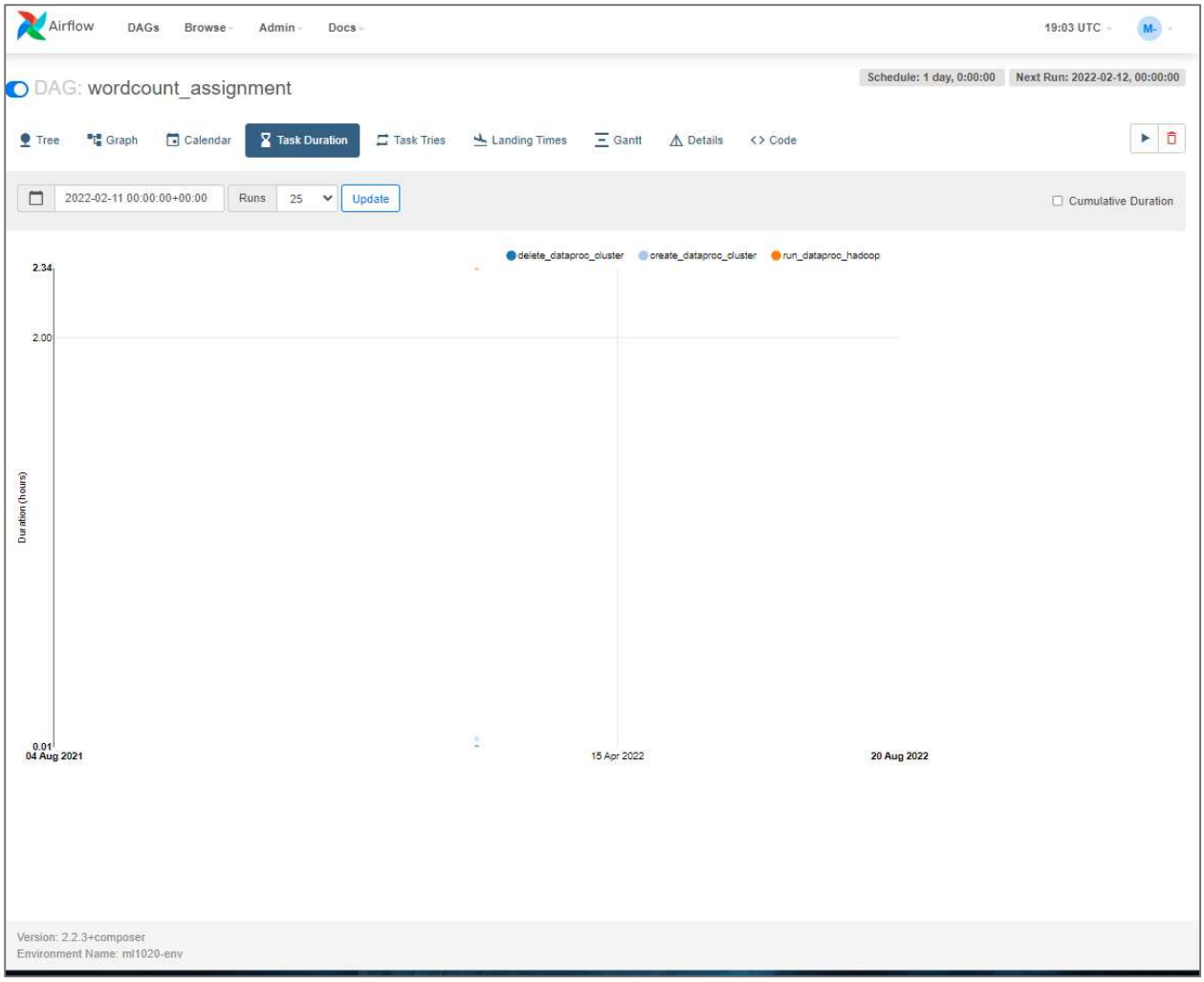
Airflow deployment

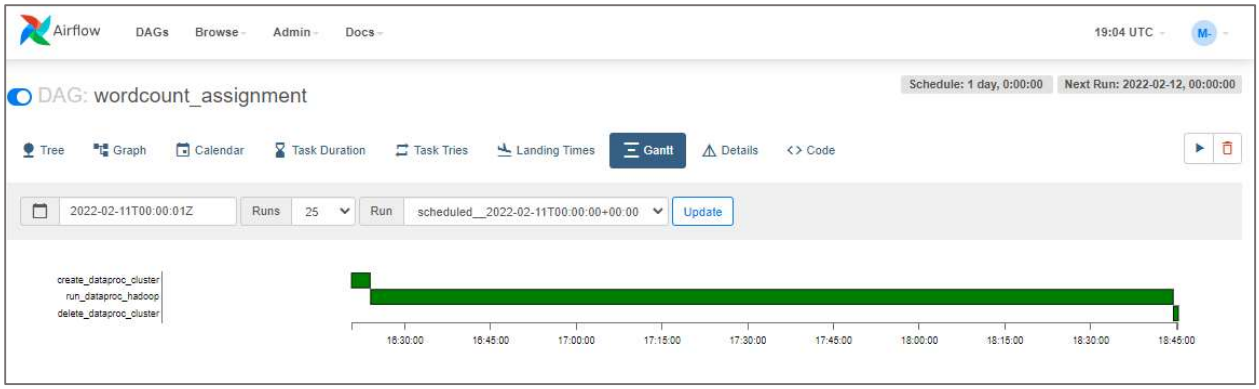


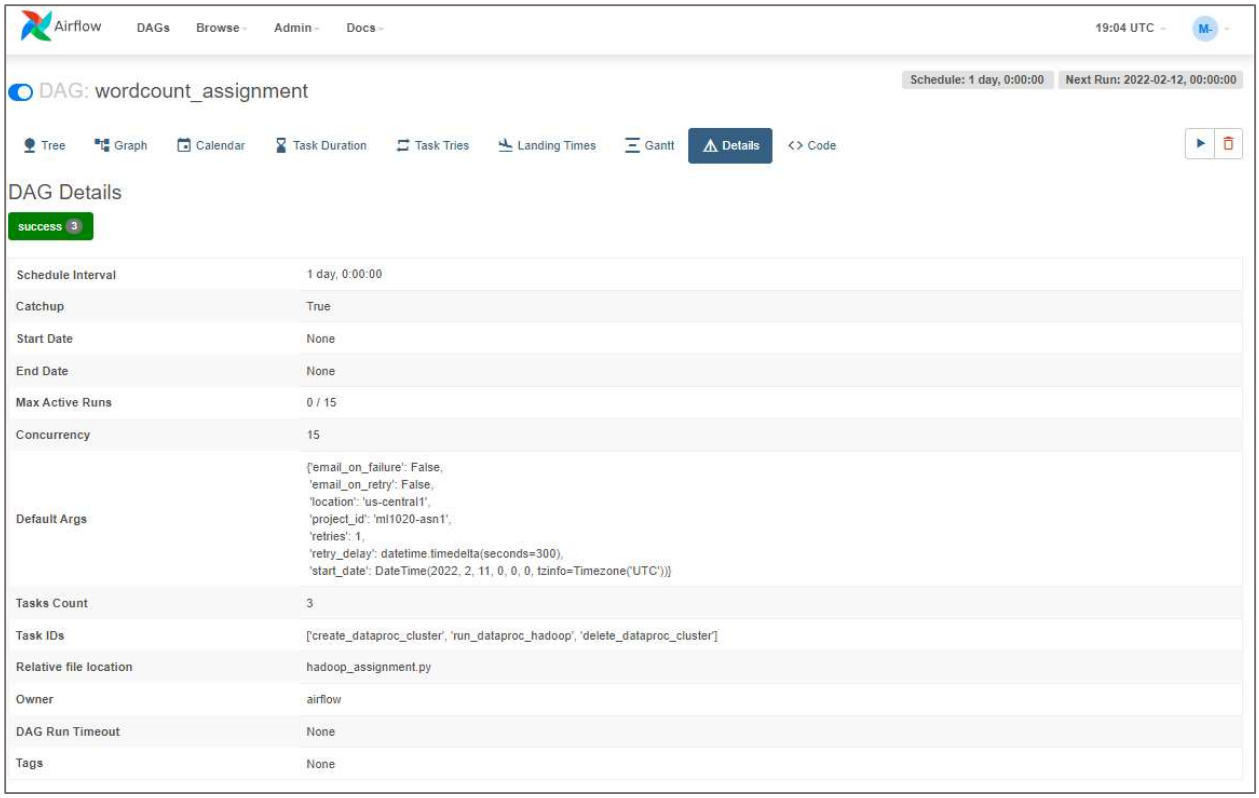



Airflow
DAGs
Browse
Admin
Docs
19:02 UTC
M-
DAG: wordcount_assignment
success
Schedule: 1 day, 0:00:00
Next Run: 2022-02-12, 00:00:00
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-02-11T00:00:01Z
Runs
25
Run
scheduled__2022-02-11T00:00:00+00:00
Find Task...
Layout
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
create_dataproc_cluster
run_dataproc_hadoop
delete_dataproc_cluster


Airflow
DAGs
Browse
Admin
Docs
19:03 UTC
M-
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
success
failed
running
no_status
2022
Sun
Mon
Tue
Wed
Thu
Fri
Sat


Airflow
DAGs
Browse
Admin
Docs
19:03 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
2022-02-11 00:00:00+00:00
Runs
25
Update
Cumulative Duration
delete_dataproc_cluster
create_dataproc_cluster
run_dataproc_hadoop
2.34
2.00
0.01
Duration (hours)
04 Aug 2021
15 Apr 2022
20 Aug 2022
Version: 2.2.3+composer
Environment Name: ml1020-env


Airflow
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
2022-02-11T00:00:01Z
Runs
25
Run
scheduled__2022-02-11T00:00:00+00:00
Update
create_dataproc_cluster
run_dataproc_hadoop
delete_dataproc_cluster
16:30:00
16:45:00
17:00:00
17:15:00
17:30:00
17:45:00
18:00:00
18:15:00
18:30:00
18:45:00



Wordcount DAG

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

```python
# ML1020 - Assignment 2 (DAG for wordcount)

"""Example Airflow DAG that creates a Cloud Dataproc cluster, runs the Hadoop
wordcount example, and deletes the cluster.

This DAG relies on three Airflow variables
https://airflow.apache.org/docs/apache-airflow/stable/concepts/variables.html
* gcp_project - Google Cloud Project to use for the Cloud Dataproc cluster.
* gce_zone - Google Compute Engine zone where Cloud Dataproc cluster should be
  created.
* gcs_bucket - Google Cloud Storage bucket to use for result of Hadoop job.
  See https://cloud.google.com/storage/docs/creating-buckets for creating a
  bucket.
"""

import datetime
import os

from airflow import models
from airflow.providers.google.cloud.operators import dataproc
from airflow.utils import trigger_rule

# Output file for Cloud Dataproc job.
# If you are running Airflow in more than one time zone
# see https://airflow.apache.org/docs/apache-airflow/stable/timezone.html
# for best practices
output_file = os.path.join(
    models.Variable.get('gcs_bucket'), 'wordcount',
    datetime.datetime.now().strftime('%Y%m%d-%H%M%S')) + os.sep
```

```
# Path to Hadoop wordcount example available on every Dataproc cluster.
WORDCOUNT_JAR = (
    'file:///usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar'
)
# Arguments to pass to Cloud Dataproc job.
#input_file = 'gs://pub/shakespeare/rose.txt'
input_file = 'gs://ml1020-bucket/asn1/*.txt'
wordcount_args = ['wordcount', input_file, output_file]

HADOOP_JOB = {
    "reference": {"project_id": models.Variable.get('gcp_project')},
    "placement": {"cluster_name": 'composer-hadoop-tutorial-cluster-{{ ds_nodash }}'},
    "hadoop_job": {
        "main_jar_file_uri": WORDCOUNT_JAR,
        "args": wordcount_args,
    },
}

CLUSTER_CONFIG = {
    "master_config": {
        "num_instances": 1,
        "machine_type_uri": "n1-standard-2"
    },
    "worker_config": {
        "num_instances": 2,
        "machine_type_uri": "n1-standard-2"
    },
}

yesterday = datetime.datetime.combine(
    datetime.datetime.today() - datetime.timedelta(1),
    datetime.datetime.min.time())

default_dag_args = {
```

```
    # Setting start date as yesterday starts the DAG immediately when it is
    # detected in the Cloud Storage bucket.
    'start_date': yesterday,
    # To email on failure or retry set 'email' arg to your email and enable
    # emailing here.
    'email_on_failure': False,
    'email_on_retry': False,
    # If a task fails, retry it once after waiting at least 5 minutes
    'retries': 1,
    'retry_delay': datetime.timedelta(minutes=5),
    'project_id': models.Variable.get('gcp_project'),
    'location': models.Variable.get('gce_region'),

}


with models.DAG(
        'wordcount_assignment',
        # Continue to run DAG once per day
        schedule_interval=datetime.timedelta(days=1),
        default_args=default_dag_args) as dag:

    # Create a Cloud Dataproc cluster.
    create_dataproc_cluster = dataproc.DataprocCreateClusterOperator(
        task_id='create_dataproc_cluster',
        # Give the cluster a unique name by appending the date scheduled.
        # See https://airflow.apache.org/docs/apache-airflow/stable/macros-ref.html
        cluster_name='composer-hadoop-tutorial-cluster-{{ ds_nodash }}',
        cluster_config=CLUSTER_CONFIG,
        region=models.Variable.get('gce_region'))

```

```
    # Run the Hadoop wordcount example installed on the Cloud Dataproc cluster
    # master node.
    run_wordcount = dataproc.DataprocSubmitJobOperator(
        task_id='run_dataproc_hadoop',
        job=HADOOP_JOB)

    # Delete Cloud Dataproc cluster.
    delete_dataproc_cluster = dataproc.DataprocDeleteClusterOperator(
        task_id='delete_dataproc_cluster',
        cluster_name='composer-hadoop-tutorial-cluster-{{ ds_nodash }}',
        region=models.Variable.get('gce_region'),
        # Setting trigger_rule to ALL_DONE causes the cluster to be deleted
        # even if the Dataproc job fails.
        trigger_rule=trigger_rule.TriggerRule.ALL_DONE)

    # Define DAG dependencies.
    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	ï¿½	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

```python
# -*- coding: utf-8 -*-

# Sergei Bugrov
# 7-9-17
#
# Downloads all available books in English language in .txt format from http://www.gutenberg.org,
# unpacks them from .zip archives, saves them to ../books/ folder, and deletes .zip files.
#
# usage : python gutenberg.py
#
# python version : 3.6.1

import requests, bs4, os, errno, zipfile, glob
from urllib.request import urlretrieve


def main():

    if not os.path.exists('books/'):
        try:
            os.makedirs('books/')
        except OSError as e:
            if e.errno != errno.EEXIST:
                raise

    # STEP 1. BUILD A LIST OF URLS

    urls_to_books = []

    if not os.path.exists('urls_to_books.txt'):

        page_w_books_url = 'http://www.gutenberg.org/robot/harvest?filetypes[]=txt&langs[]=en'

        while 1 == 1:

```

```
            is_last_page = False

            print('Reading page: ' + page_w_books_url)

            page_w_books = requests.get(page_w_books_url, timeout=20.0)

            if page_w_books:
                page_w_books = bs4.BeautifulSoup(page_w_books.text, "lxml")
                urls = [el.get('href') for el in page_w_books.select('body > p > a[href^="http://aleph.gutenberg.org/"]')]
                url_to_next_page = page_w_books.find_all('a', string='Next Page')

                if len(urls) > 0:
                    urls_to_books.append(urls)

                    if url_to_next_page[0]:
                        page_w_books_url = "http://www.gutenberg.org/robot/" + url_to_next_page[0].get('href')
                else:
                    is_last_page = True

            if is_last_page:
                break

        urls_to_books = [item for sublist in urls_to_books for item in sublist]

        # Backing up the list of URLs
        with open('urls_to_books.txt', 'w') as output:
            for u in urls_to_books:
                output.write('%s\n' % u)

    # STEP 2. DOWNLOAD BOOKS

```

```
    # If, at some point, Step 2 is interrupted due to unforeseen
    # circumstances (power outage, lost of internet connection), replace the number
    # (value of the variable url_num) below with the one you will find in the logfile.log
    # Example
    #       logfile.log : Unzipping file #99 books/10020.zip
    #       the number  : 99
    url_num = 0

    if os.path.exists('urls_to_books.txt') and len(urls_to_books) == 0:
        with open('urls_to_books.txt', 'r') as f:
            urls_to_books = f.read().splitlines()

    for url in urls_to_books[url_num:]:

        dst = 'books/' + url.split('/')[-1].split('.')[0].split('-')[0]

        with open('logfile.log', 'w') as f:
            f.write('Unzipping file #' + str(url_num) + ' ' + dst + '.zip' + '\n')

        if len(glob.glob(dst + '*')) == 0:
            urlretrieve(url, dst + '.zip')

            with zipfile.ZipFile(dst + '.zip', "r") as zip_ref:
                try:
                    zip_ref.extractall("books/")
                    print(str(url_num) + ' ' + dst + '.zip ' + 'unzipped successfully!')
                except NotImplementedError:
                    print(str(url_num) + ' Cannot unzip file:', dst)
```

```python

            os.remove(dst + '.zip')

        url_num += 1


if __name__ == '__main__':
    """
    The main function is called when gutenberg.py is run from the command line
    """

    main()
```