EECS E6893 Big Data Analytics

Homework 4 Report

Name: Yi Zhang

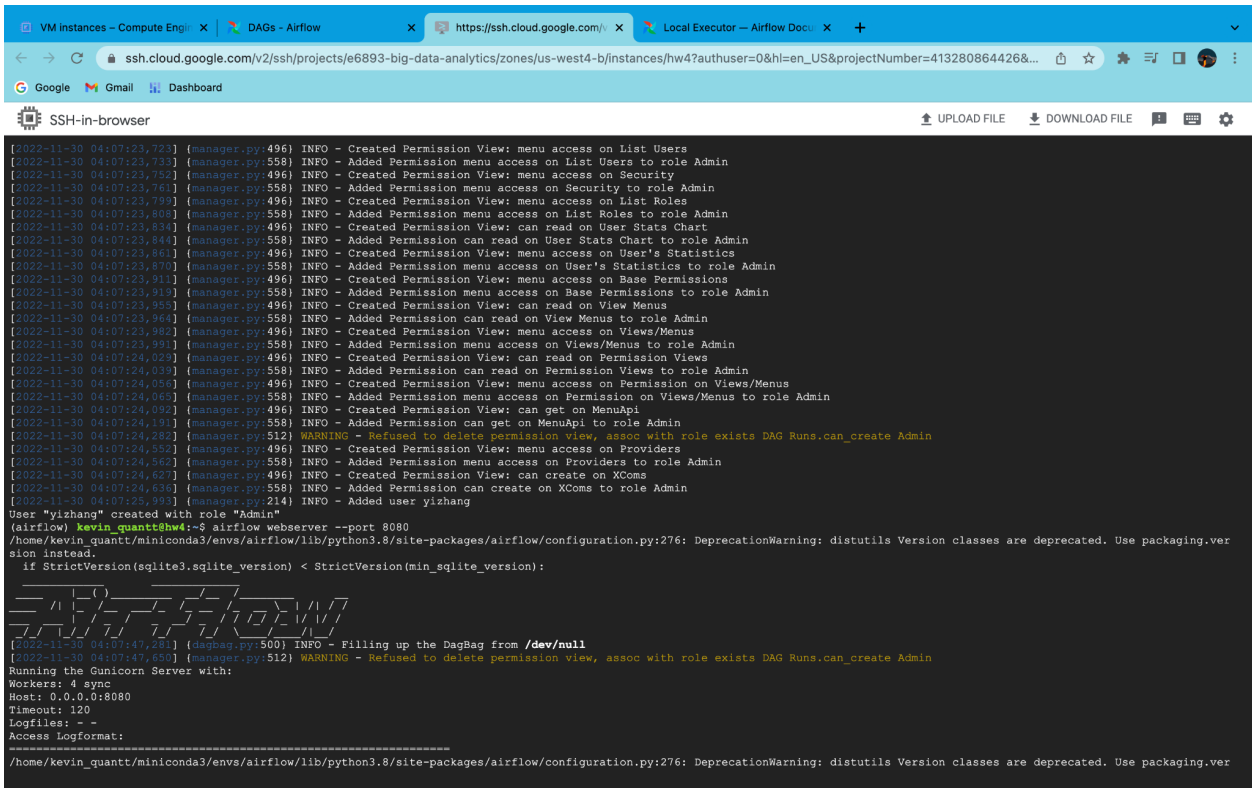
Uni: yz4130

November 2022

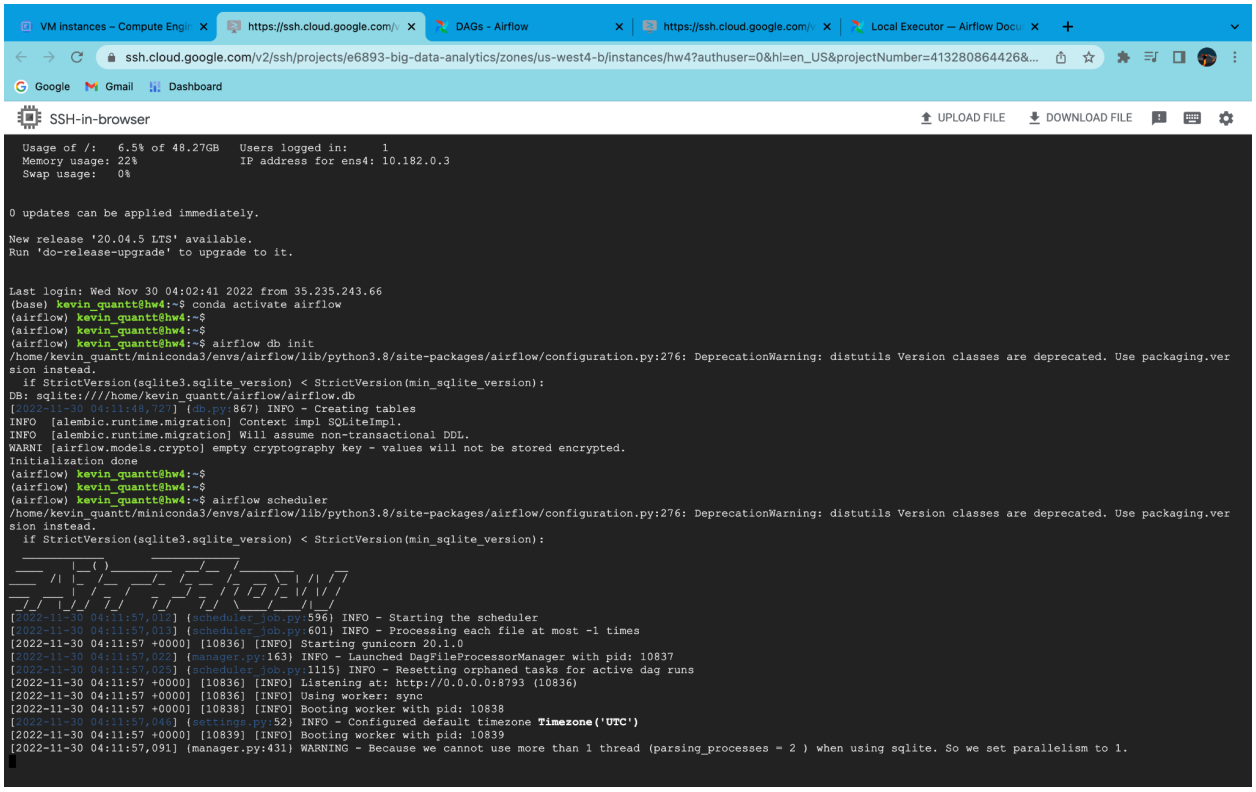
Task 1 Hello World

1.1

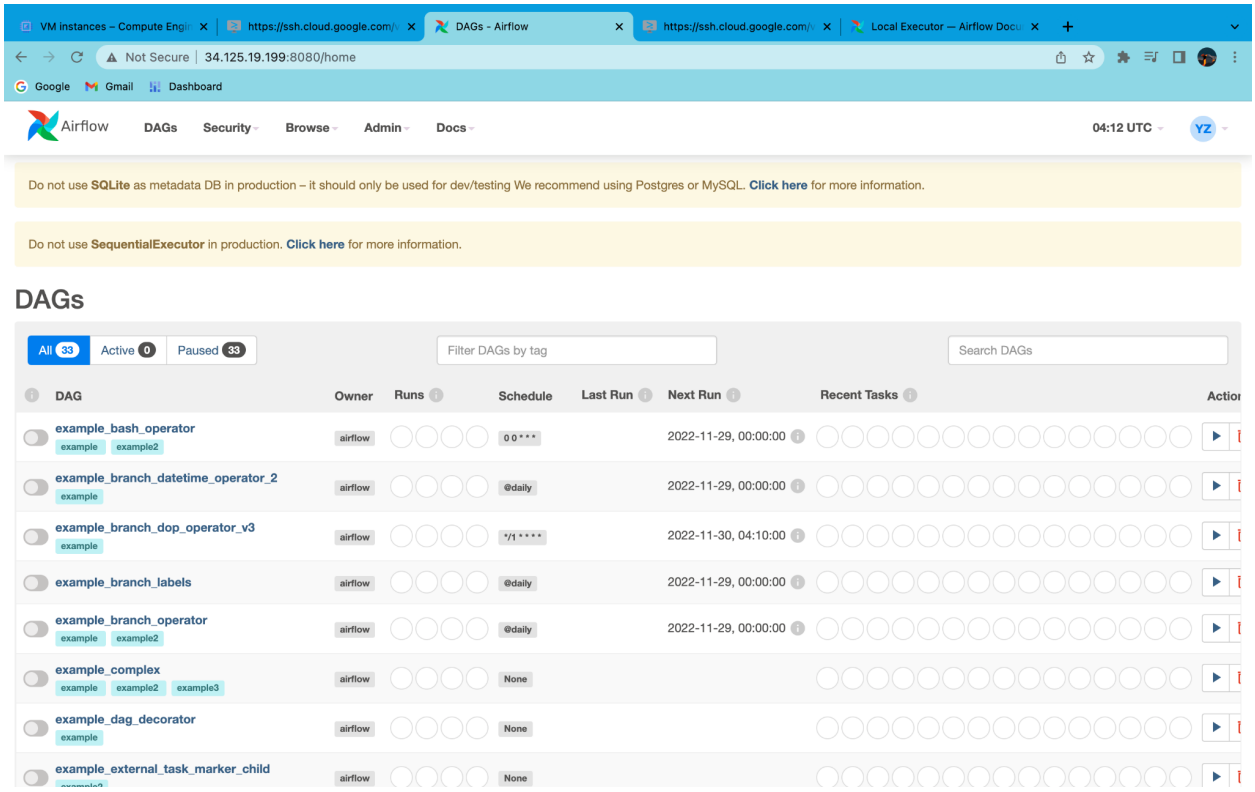
(1) Start web server



Start scheduler



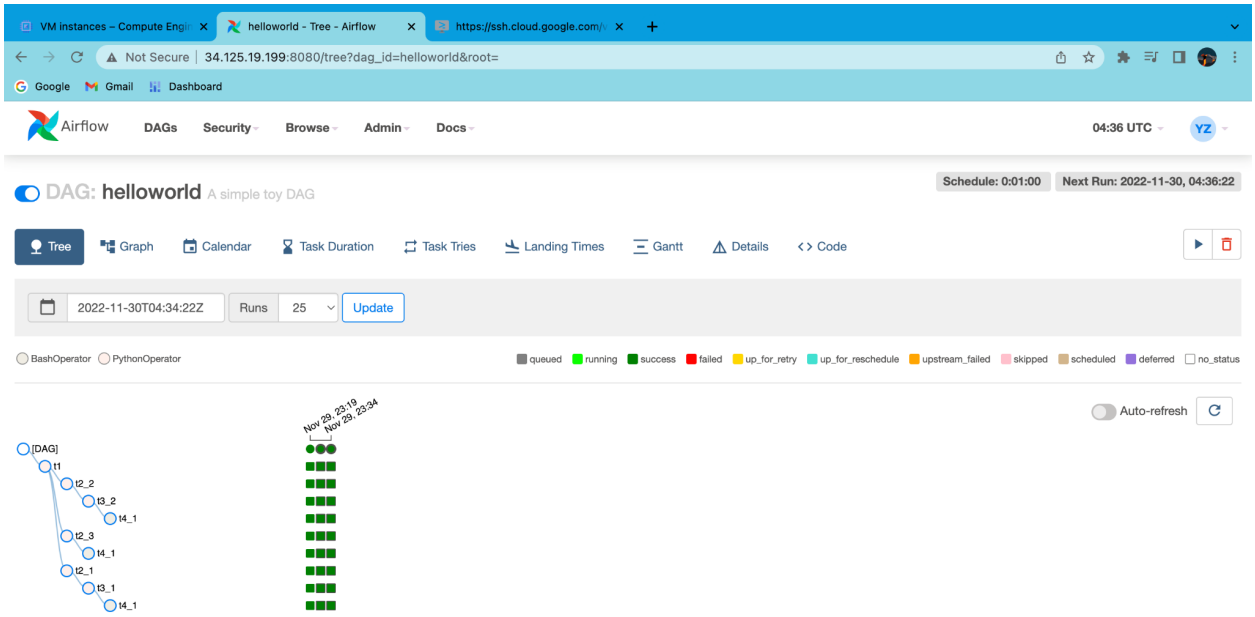
(2) Start web server and login, pictures of DAGs



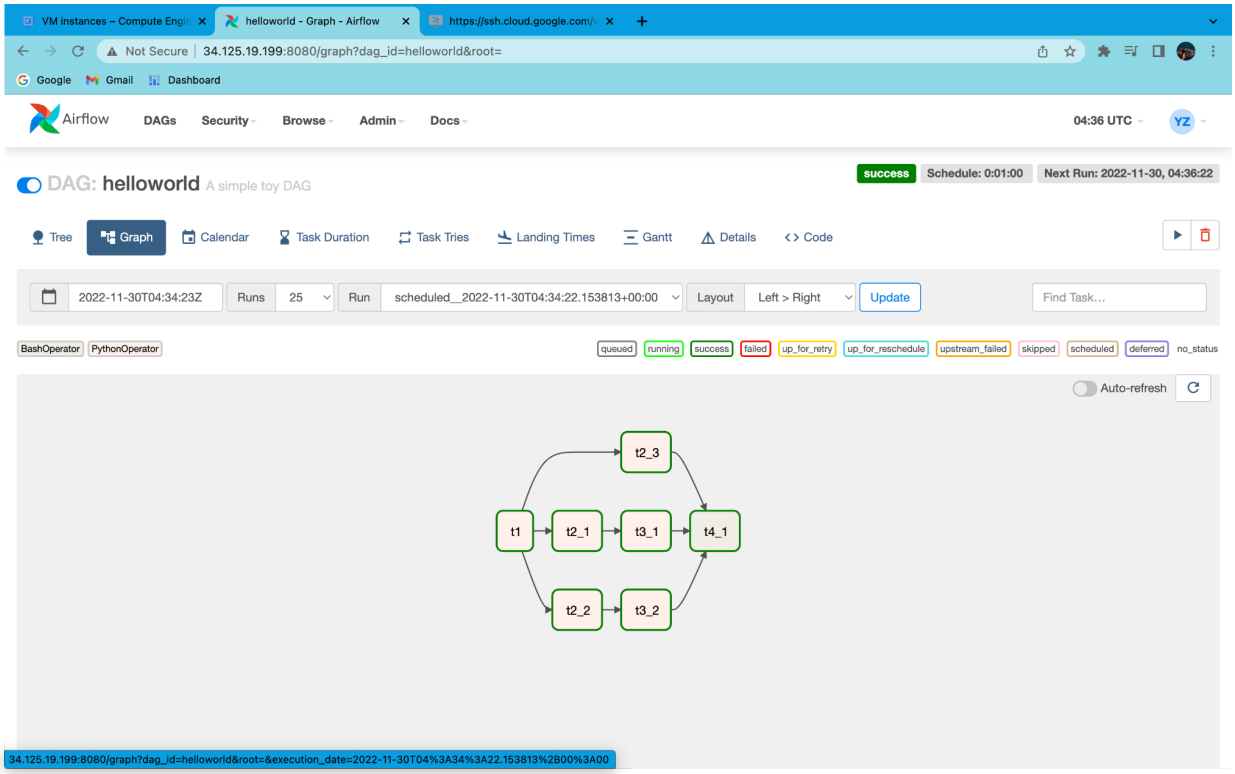
1.2

(1) The following results are based on sequential executors, which is the default setting of airflow.

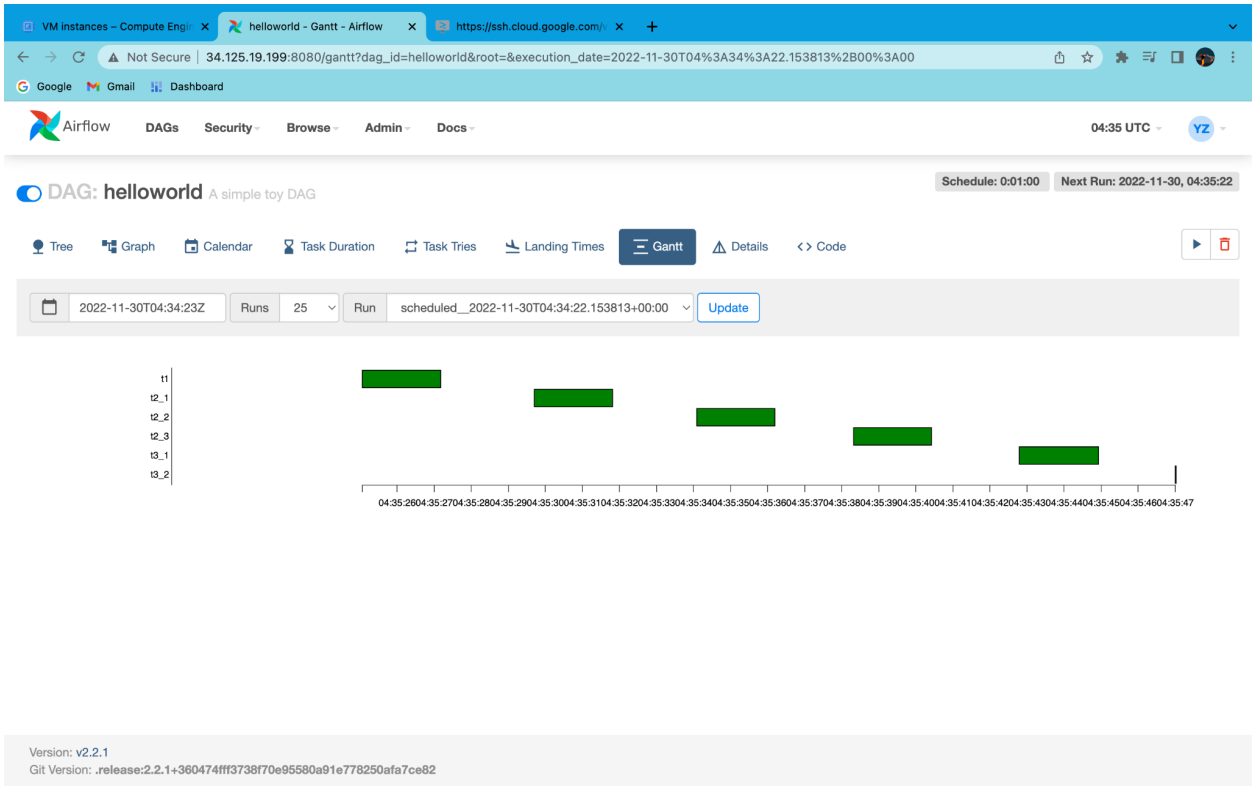
Tree


helloworld - Tree - Airflow
Not Secure | 34.125.19.199:8080/tree?dag_id=helloworld&root=
Airflow
DAGs
Security
Browse
Admin
Docs
04:36 UTC
DAG: helloworld A simple toy DAG
Schedule: 0:01:00
Next Run: 2022-11-30, 04:36:22
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-30T04:34:22Z
Runs
25
Update
BashOperator
PythonOperator
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
Nov 29, 23:19
Nov 29, 23:34
[DAG]
t1
t2_2
t3_2
t4_1
t2_3
t4_1
t2_1
t3_1
t4_1

Graph

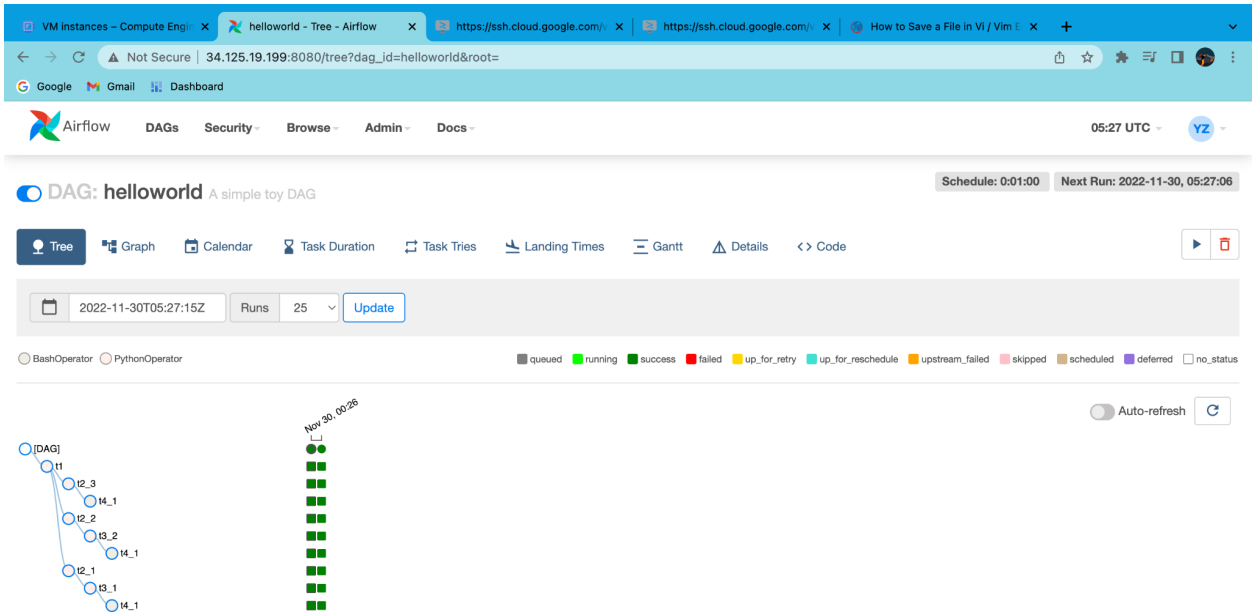

helloworld - Graph - Airflow
Not Secure | 34.125.19.199:8080/graph?dag_id=helloworld&root=
Airflow
DAGs
Security
Browse
Admin
Docs
04:36 UTC
DAG: helloworld A simple toy DAG
success
Schedule: 0:01:00
Next Run: 2022-11-30, 04:36:22
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-30T04:34:23Z
Runs
25
Run
scheduled__2022-11-30T04:34:22.153813+00:00
Layout
Left > Right
Update
Find Task...
BashOperator
PythonOperator
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
t2_3
t1
t2_1
t3_1
t4_1
t2_2
t3_2
34.125.19.199:8080/graph?dag_id=helloworld&root=&execution_date=2022-11-30T04%3A34%3A22.153813%2B00%3A00

Gannt

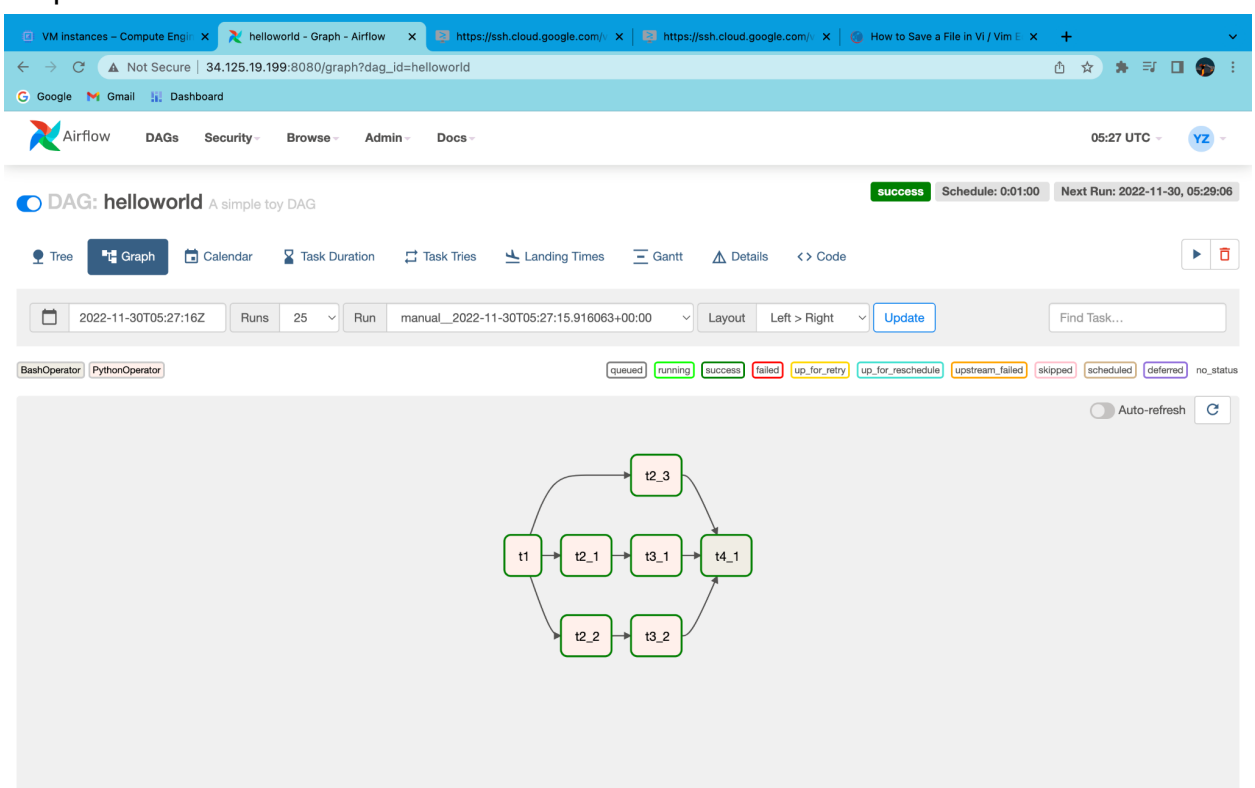


The following results are based on the local executor

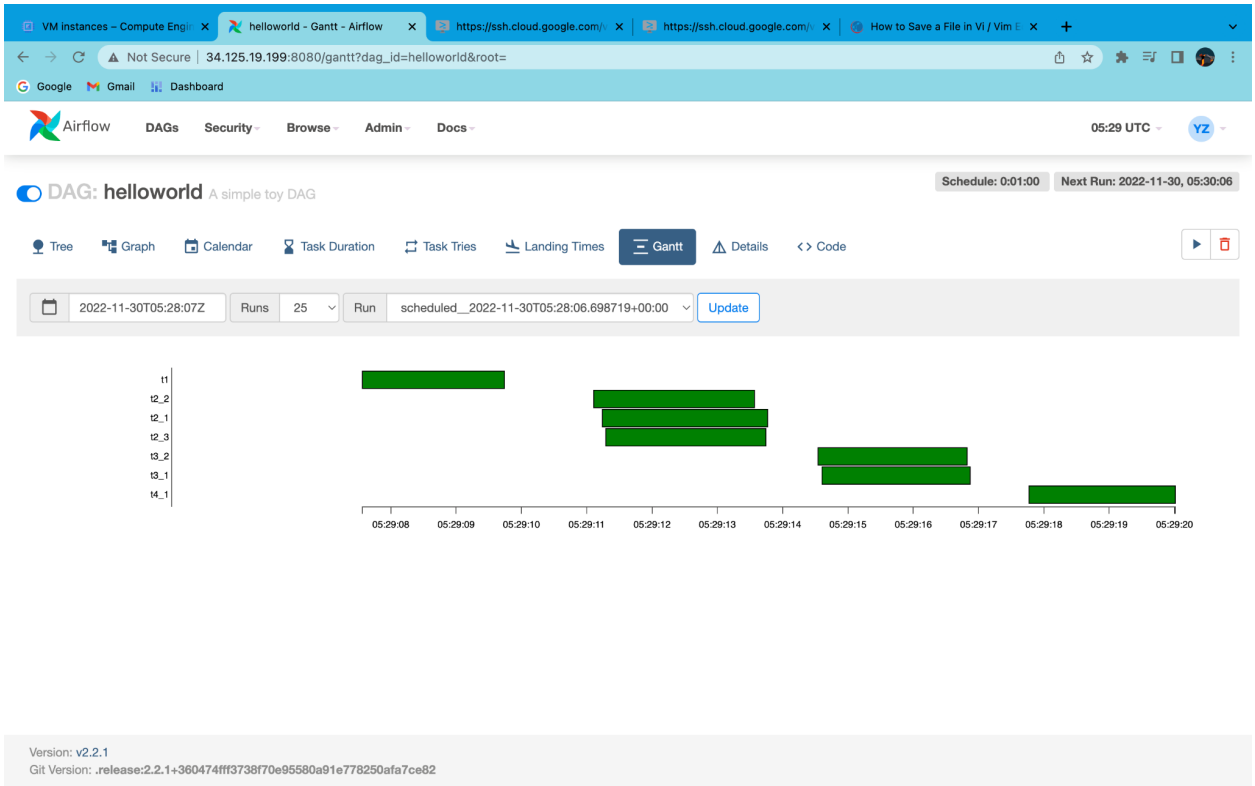
Tree


VM instances – Compute Engin
helloworld - Tree - Airflow
https://ssh.cloud.google.com/
https://ssh.cloud.google.com/
How to Save a File in Vi / Vim E
Not Secure | 34.125.19.199:8080/tree?dag_id=helloworld&root=
Google
Gmail
Dashboard
Airflow
DAGs
Security
Browse
Admin
Docs
05:27 UTC
YZ
DAG: helloworld A simple toy DAG
Schedule: 0:01:00
Next Run: 2022-11-30, 05:27:06
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-30T05:27:15Z
Runs
25
Update
BashOperator
PythonOperator
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
Nov 30, 00:26
[DAG]
t1
t2_3
t4_1
t2_2
t3_2
t4_1
t2_1
t3_1
t4_1

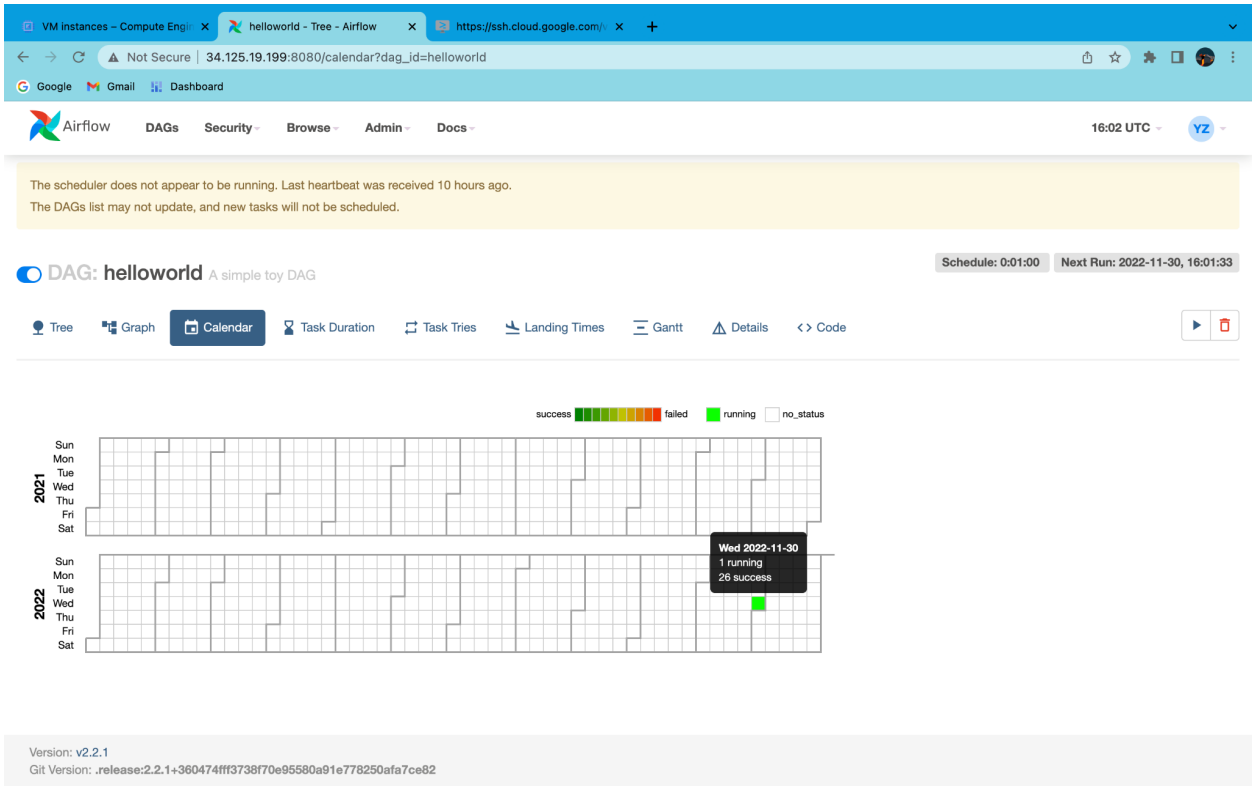
Graph


VM instances – Compute Engin
helloworld - Graph - Airflow
https://ssh.cloud.google.com/
https://ssh.cloud.google.com/
How to Save a File in Vi / Vim E
Not Secure | 34.125.19.199:8080/graph?dag_id=helloworld
Google
Gmail
Dashboard
Airflow
DAGs
Security
Browse
Admin
Docs
05:27 UTC
YZ
DAG: helloworld A simple toy DAG
success
Schedule: 0:01:00
Next Run: 2022-11-30, 05:29:06
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-30T05:27:16Z
Runs
25
Run
manual__2022-11-30T05:27:15.916063+00:00
Layout
Left > Right
Update
Find Task...
BashOperator
PythonOperator
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
t2_3
t1
t2_1
t3_1
t4_1
t2_2
t3_2

Gantt

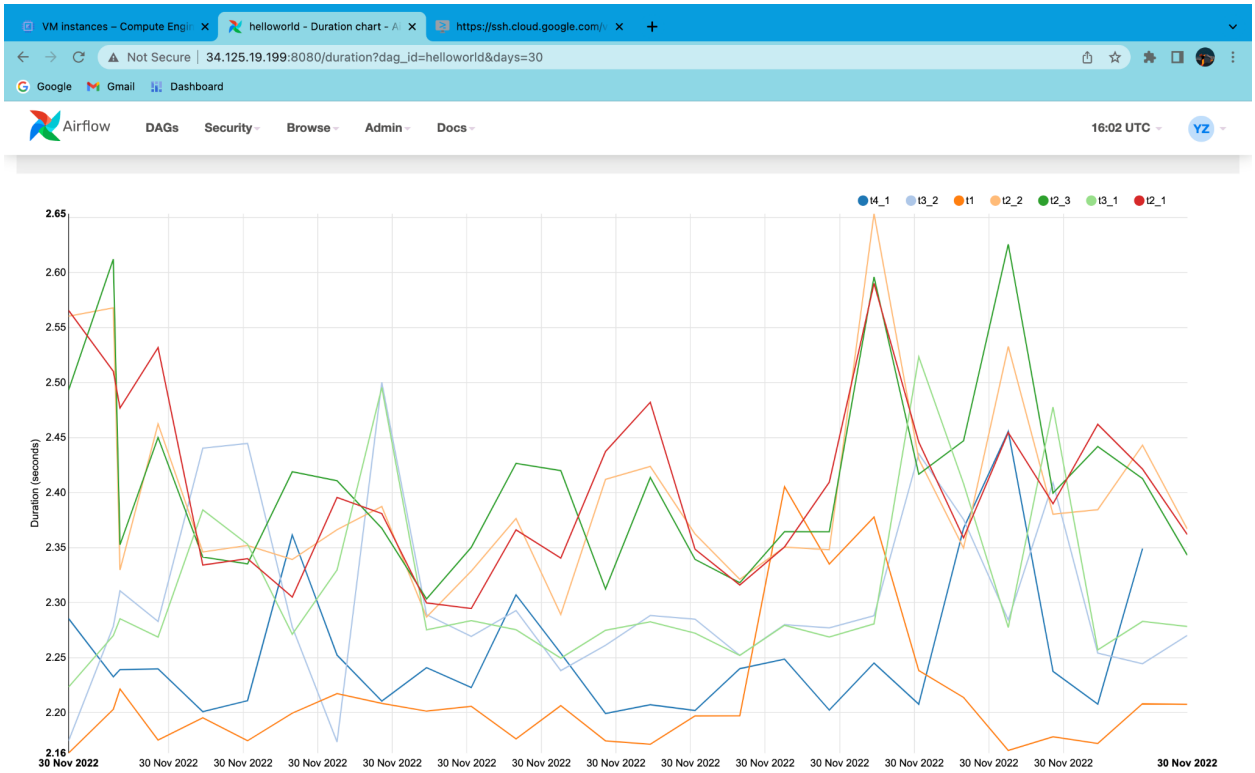


(2) Calendar visualization is another way to monitor the overall workflow, which shows the task's status on different dates within a year. This is helpful when the task schedule is heavily related to the date.



Task Duration uses line charts to demonstrate the time consumed by each task with respect to time. It is convenient to use this feature to compare the runtime of

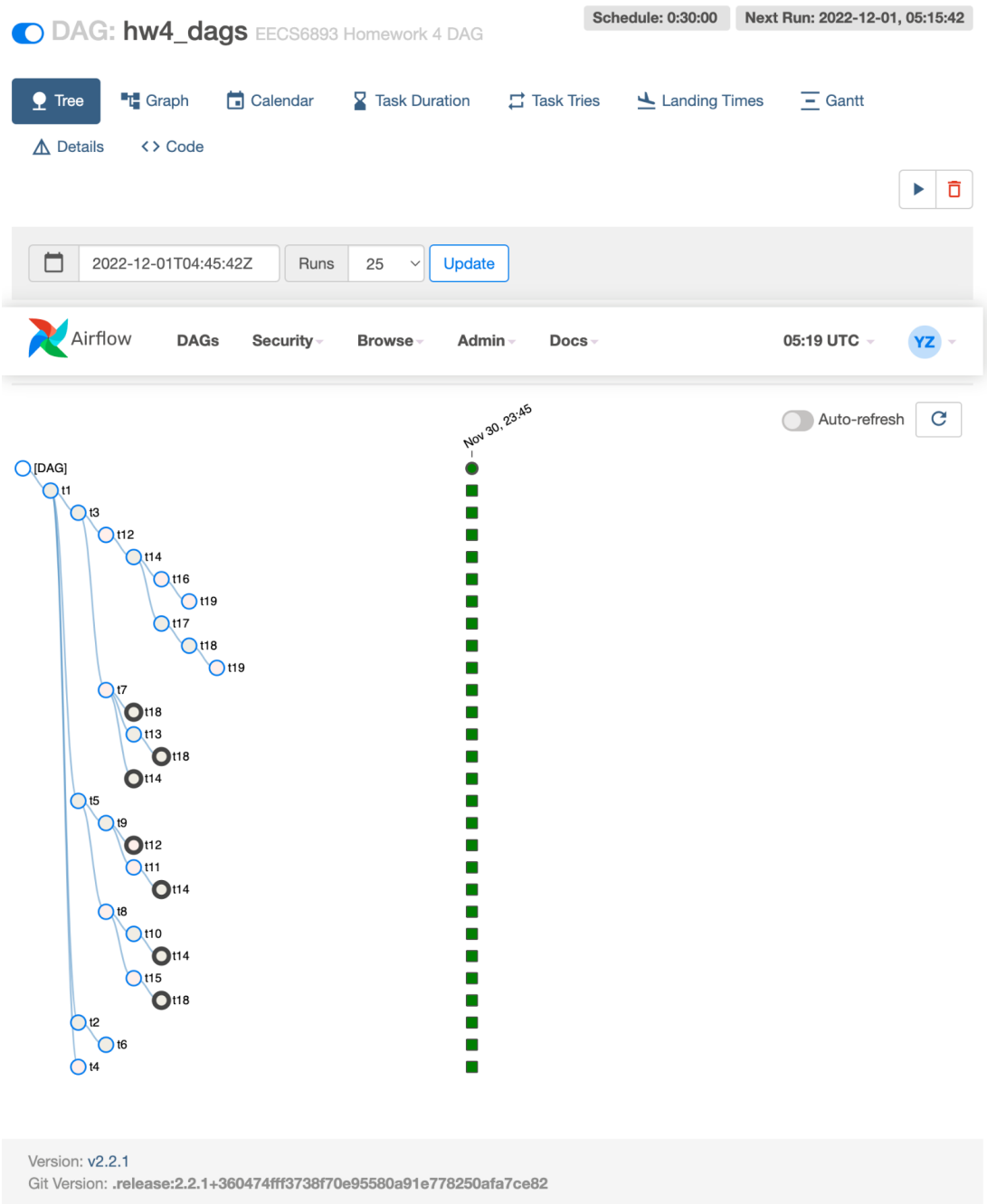
each task.



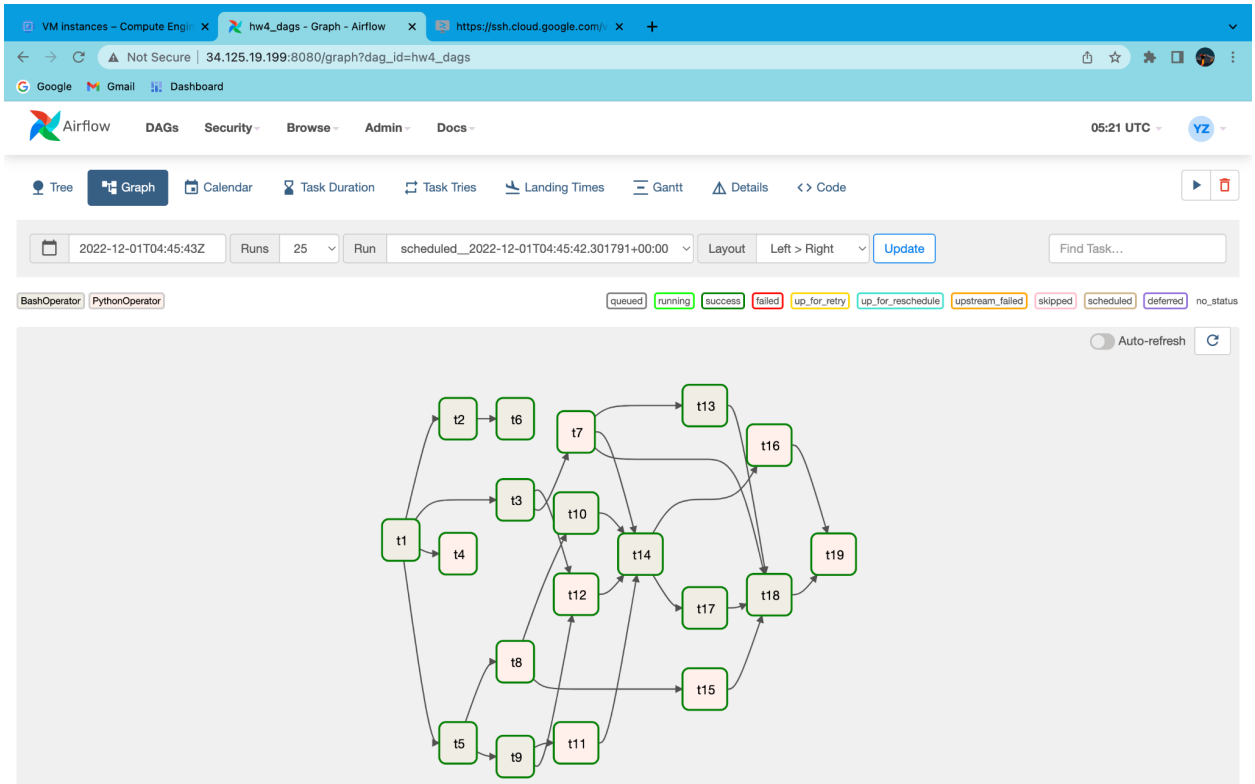
Task 2 Build Workflows

2.1

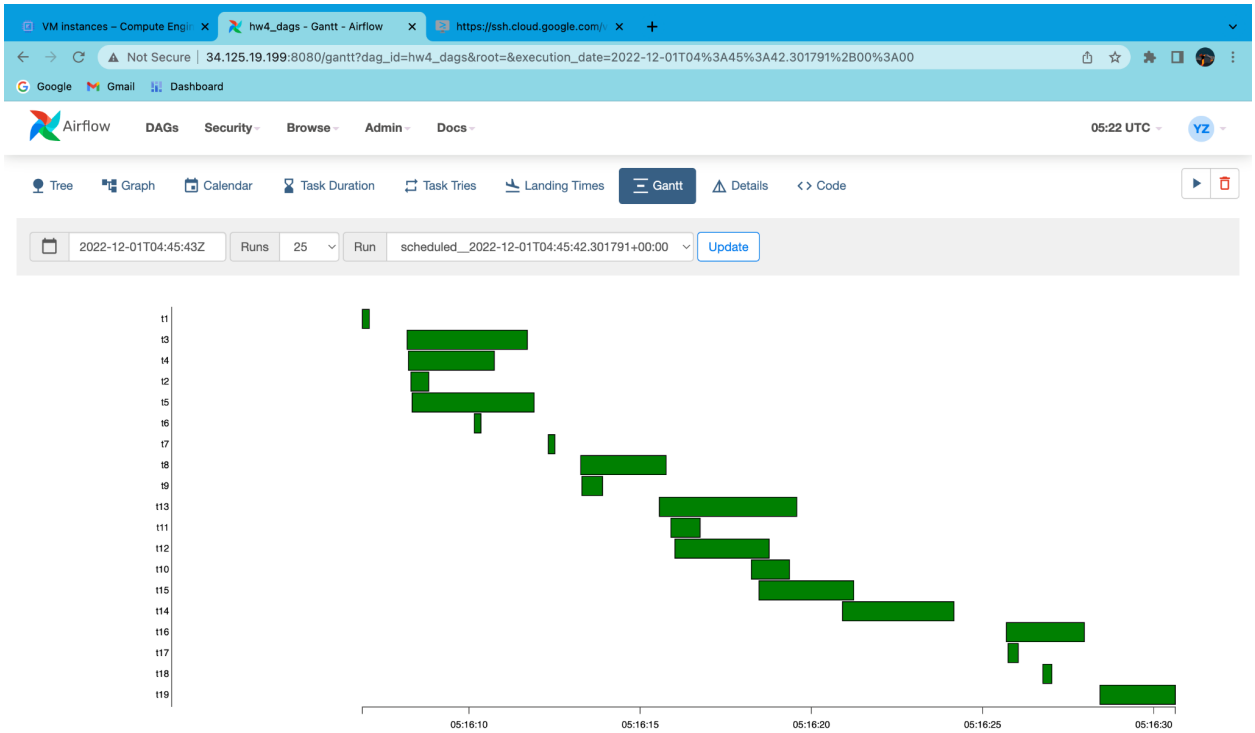
(1) Tree


DAG: hw4_dags EECS6893 Homework 4 DAG
Schedule: 0:30:00
Next Run: 2022-12-01, 05:15:42
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-12-01T04:45:42Z
Runs
25
Update
Airflow
DAGs
Security
Browse
Admin
Docs
05:19 UTC
YZ
Auto-refresh
Nov 30, 23:45
[DAG]
t1
t3
t12
t14
t16
t19
t17
t18
t19
t7
t18
t13
t18
t14
t5
t9
t12
t11
t14
t8
t10
t14
t15
t18
t2
t6
t4
Version: v2.2.1
Git Version: .release:2.2.1+360474fff3738f70e95580a91e778250afa7ce82

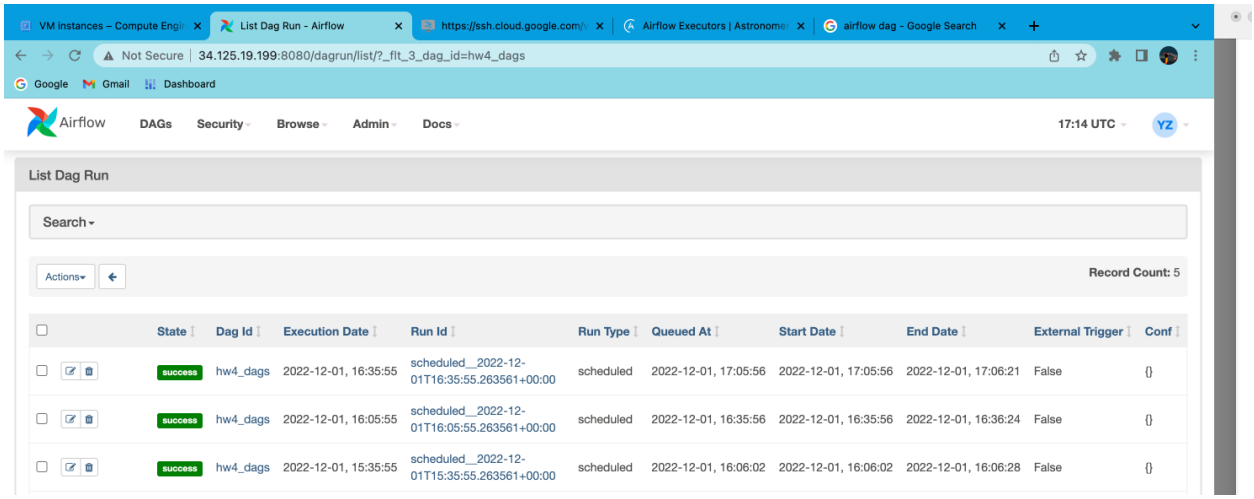
Graph


VM instances – Compute Engin
hw4_dags - Graph - Airflow
https://ssh.cloud.google.com/
Not Secure | 34.125.19.199:8080/graph?dag_id=hw4_dags
Google
Gmail
Dashboard
Airflow
DAGs
Security
Browse
Admin
Docs
05:21 UTC
YZ
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-12-01T04:45:43Z
Runs
25
Run
scheduled__2022-12-01T04:45:42.301791+00:00
Layout
Left > Right
Update
Find Task...
BashOperator
PythonOperator
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
t15
t16
t17
t18
t19

(2) Gantt


VM instances – Compute Engin
hw4_dags - Gantt - Airflow
https://ssh.cloud.google.com/
Not Secure | 34.125.19.199:8080/gantt?dag_id=hw4_dags&root=&execution_date=2022-12-01T04%3A45%3A42.301791%2B00%3A00
Google
Gmail
Dashboard
Airflow
DAGs
Security
Browse
Admin
Docs
05:22 UTC
YZ
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-12-01T04:45:43Z
Runs
25
Run
scheduled__2022-12-01T04:45:42.301791+00:00
Update
t1
t3
t4
t2
t5
t6
t7
t8
t9
t13
t11
t12
t10
t15
t14
t16
t17
t18
t19
05:16:10
05:16:15
05:16:20
05:16:25
05:16:30

(3) I simply choose the date that I started to do the project as the start date. The running history is as follows



State	Dag Id	Execution Date	Run Id	Run Type	Queued At	Start Date	End Date	External Trigger	Conf
success	hw4_dags	2022-12-01, 16:35:55	scheduled__2022-12-01T16:35:55.263561+00:00	scheduled	2022-12-01, 17:05:56	2022-12-01, 17:05:56	2022-12-01, 17:06:21	False	{}
success	hw4_dags	2022-12-01, 16:05:55	scheduled__2022-12-01T16:05:55.263561+00:00	scheduled	2022-12-01, 16:35:56	2022-12-01, 16:35:56	2022-12-01, 16:36:24	False	{}
success	hw4_dags	2022-12-01, 15:35:55	scheduled__2022-12-01T15:35:55.263561+00:00	scheduled	2022-12-01, 16:06:02	2022-12-01, 16:06:02	2022-12-01, 16:06:28	False	{}

2.2

(1) First download all history data of each stock.

(2) In order to make the model more accurate, I first drop null value data and then utilized the MinMaxScaler to transform the data to the same scale

(3) Next step is to train the regression model. After obtaining the the estimated coefficient, I am able to give the prediction of next day's high given all the information up to now, which I stored in a global data frame variable called 'prediction'

(Notes: The three steps above can be done by the function below)

```python
# train linear regression model and predict next day 'high' given current
def train_predict(**kwargs):
    global prediction, stock_df, error
    names = kwargs['names']
    stock_df = yf.download(names, group_by='ticker').dropna()
    df = stock_df
    scaler = MinMaxScaler()
    result = []
    for n in names:
        data = df[n].drop('Adj Close', axis=1)
        data = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)
        y = data['High'][10:]
        X = np.zeros((y.size, 50))
        for i in range(y.size):
            X[i, :] = data[i: i+10].values.reshape(-1)
        model = sm.OLS(y, X).fit()
        data['High'][10:] = model.predict()
        inversed_data = pd.DataFrame(scaler.inverse_transform(data), columns=data.columns)
        result.append(inversed_data['High'][len(inversed_data['High'])-1])

    prediction.loc[len(prediction)] = result
```

To illustrate the result, the log of from airflow is as below

VM instances – Compute Engine | https://ssh.cloud.google.com/v | https://ssh.cloud.google.com/v | stock_dags - Airflow

Not Secure | 34.125.19.199:8080/log?dag_id=stock_dags&task_id=predict&execution_date=2022-12-01T21%3A31%3A41.688825%2B00%3A00

Google Gmail Dashboard

Airflow DAGs Security Browse Admin Docs 21:34 UTC YZ

```
AIRFLOW_CTX_EXECUTION_DATE=2022-12-01T21:31:41.688825+00:00
AIRFLOW_CTX_DAG_RUN_ID=scheduled__2022-12-01T21:31:41.688825+00:00
[2022-12-02, 21:31:53 UTC] {logging_mixin.py:109} INFO -
[                                        0%                                        ]
[2022-12-02, 21:31:53 UTC] {logging_mixin.py:109} INFO -
[*******************   40%                                        ]  2 of 5 completed
[2022-12-02, 21:31:54 UTC] {logging_mixin.py:109} INFO -
[************************60%*****                                  ]  3 of 5 completed
[2022-12-02, 21:31:54 UTC] {logging_mixin.py:109} INFO -
[************************80%*************                          ]  4 of 5 completed
[2022-12-02, 21:31:54 UTC] {logging_mixin.py:109} INFO -
[************************100%************************************]  5 of 5 completed
[2022-12-02, 21:31:54 UTC] {logging_mixin.py:109} INFO -
[2022-12-02, 21:31:59 UTC] {logging_mixin.py:109} INFO -                    GOOGL                                                  ...       AMZN
                 Open        High         Low       Close   Adj Close  ...       High        Low      Close  Adj Close       Volume
Date                                                                    ...
2012-05-18   15.643143   15.826326   14.932432   15.025025   15.025025  ...  10.981500  10.640500  10.692500  10.692500  104634000.0
2012-05-21   15.027778   15.407658   15.015015   15.368118   15.368118  ...  10.999000  10.641000  10.905500  10.905500   71596000.0
2012-05-22   15.351351   15.360611   14.914915   15.035035   15.035035  ...  10.943500  10.698000  10.766500  10.766500   74662000.0
2012-05-23   15.056306   15.255255   14.942943   15.251752   15.251752  ...  10.877500  10.559000  10.864000  10.864000   84876000.0
2012-05-24   15.244244   15.313313   14.986737   15.106607   15.106607  ...  10.883000  10.635000  10.762000  10.762000   62822000.0
...                ...         ...         ...         ...         ...  ...        ...        ...        ...        ...          ...
2022-11-28   97.040001   97.580002   95.610001   96.050003   96.050003  ...  96.400002  93.430000  93.949997  93.949997   74943100.0
2022-11-29   95.730003   96.120003   94.110001   95.190002   95.190002  ...  94.410004  91.440002  92.419998  92.419998   65567300.0
2022-11-30   94.820000  101.040001   94.419998  100.989998  100.989998  ...  96.540001  91.529999  96.540001  96.540001  102805800.0
2022-12-01  101.019997  102.250000  100.250000  100.989998  100.989998  ...  97.230003  94.919998  95.500000  95.500000   68381200.0
2022-12-02   99.050003  100.769997   98.900002  100.440002  100.440002  ...  95.360001  93.779999  94.129997  94.129997   72096414.0

[2654 rows x 30 columns]
[2022-12-02, 21:31:59 UTC] {logging_mixin.py:109} INFO - [149.30119602889238, 102.53966911750494, 121.7160769672079, 256.6254464670454, 96.94579232107337]
[2022-12-02, 21:31:59 UTC] {logging_mixin.py:109} INFO -          AAPL       GOOGL        META        MSFT       AMZN
0  149.301196  102.539669  121.716077  256.625446  96.945792
[2022-12-02, 21:31:59 UTC] {python.py:152} INFO - Done. Returned value was: None
[2022-12-02, 21:31:59 UTC] {taskinstance.py:1270} INFO - Marking task as SUCCESS. dag_id=stock_dags, task_id=predict, execution_date=20221201T213141, start_date=20221202T213151, e
[2022-12-02, 21:32:00 UTC] {local_task_job.py:154} INFO - Task exited with return code 0
[2022-12-02, 21:32:00 UTC] {local_task_job.py:264} INFO - 1 downstream tasks scheduled from follow-on schedule check
```

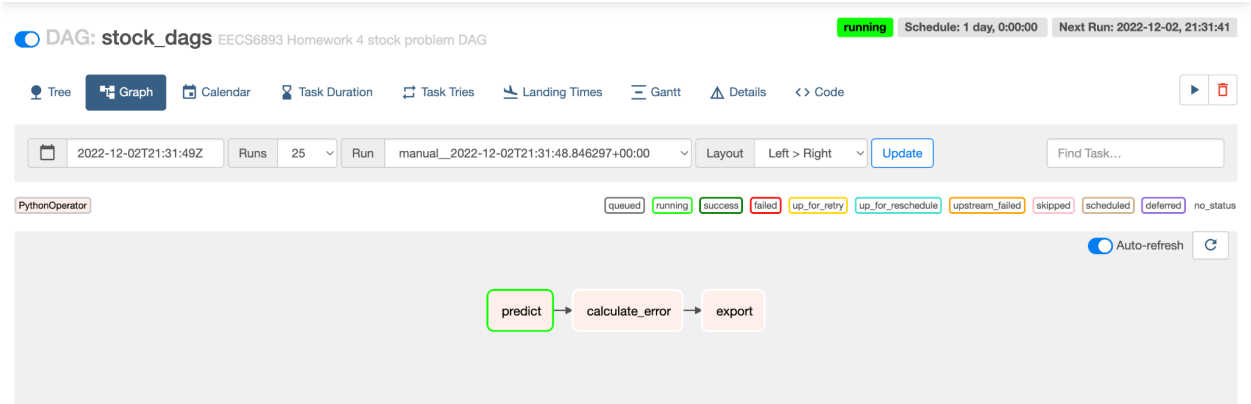
(4) Every time we have the updated data (downloaded when executing the prediction step), we can calculate the relative error of the last prediction. Similarly, I stored the error of each prediction into a global data frame variable called ‘error’

```python
# calculate the relative error of the prediction
def calculate(**kwargs):
    names = kwargs['names']
    df = kwargs['df']
    global error, prediction, stock_df
    if len(prediction) == 0 or not df:
        return
    actual = df.iloc[-1, :]
    reletative_error = []
    for n in names:
        reletative_error.append(prediction.iloc[-1, :][n] -  actual[n]['High']) / actual[n]['High']
    error.loc[len(error)] = reletative_error
```

If we want to export the data frame, simply use the pandas function ‘to_csv’ to export the data to the location we want, which can be done as below

```python
# export the error dataframe into a csv file to a path
def export(**kwargs):
    error_path = kwargs['error']
    prediction_path = kwargs['prediction']
    error.to_csv(error_path)
    prediction.to_csv(prediction_path)
```

(5) The DAG workflow is simple, first download data and give predictions, next is to calculate the error and then export data if necessary.



The cross tasks communication is done by using several global variables here. Of course an advanced way to do it is to use the Xcom mechanism, which can pass the updated data from 'predict' to 'calculate_error' step. The scheduler and other DAG arguments are set by below

```
args = {
    'owner': 'yizhang',
    'depends_on_past': False,
    'email': ['yz4130@columbia.edu'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(seconds=30),
    'schedule _interval': '0 7 * * *',
    'start_date': datetime(2022, 11, 20)
}
```

Task 3 Written Questions

3.1

Result is summarized in the table below

Executors	Pros	Cons
Sequential	Simple order to execute the tasks, workflow is easy	Time-consuming, can not harness the machine

	to understand and debug	computing power
Local	Execute the tasks in parallel, speed up the process, relatively cheap	Less scalable
Celery	Most mature version, build for horizontally scaling, offer worker termination grace period	Require infrastructure support, i.e. Celery and its backend, expensive
Kubernetes	Cost efficient, fault tolerance, less work for scheduler	Need knowledge of Kubernetes, workers need extra seconds to start tasks

3.2

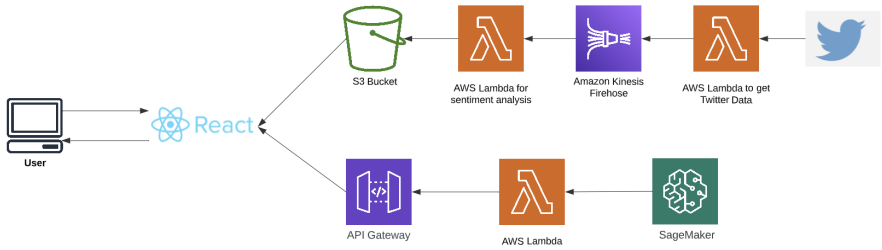
Wildfire has always been a major natural disaster especially in recent years as global warming becomes a key driver in more frequent wildfires.

The goal of our project is to provide insights on the behavior of the wildfires by exploratory data analysis and **predicting the cause and severity of wildfires based on environmental/geographical information**, and thus to help us preemptively reduce future wildfire risks.

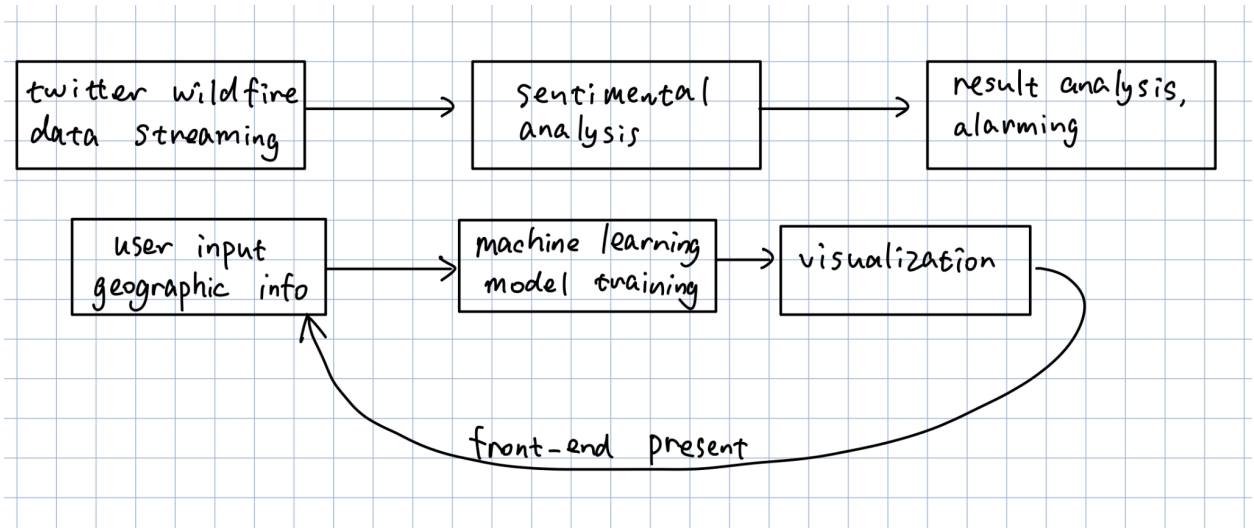
The novelty of this project lies in that it utilizes twitter API to provide sentiment analysis based on the disaster tweets in specific geolocations, which helps to predict people's reactions towards the wildfire.

(1) Based on the following workflow, we can summarize the project into

a. Visualization (wildfire information based on user input)
b. Machine learning prediction for wildfire cause and severity
c. Real-time twitter data (wildfire related hashtags) streaming
d. Sentimental analysis
e. Result analysis, real-time wildfire alarming



The DAG graph will look like below



(2) For twitter data streaming, given its real-time nature, it is scheduled to be repeated at a relatively high frequency. But since wildfire is not a very frequent disaster compared to other accidents, the best interval should be in the unit of hour.
For the machine learning part, since the database is updated at a much lower frequency than twitter data, the interval should be in the unit of month.