EECS 6893: Big Data Analytics
HW4
Yutao Zhou UNI: yz4359

Task 1 Helloworld

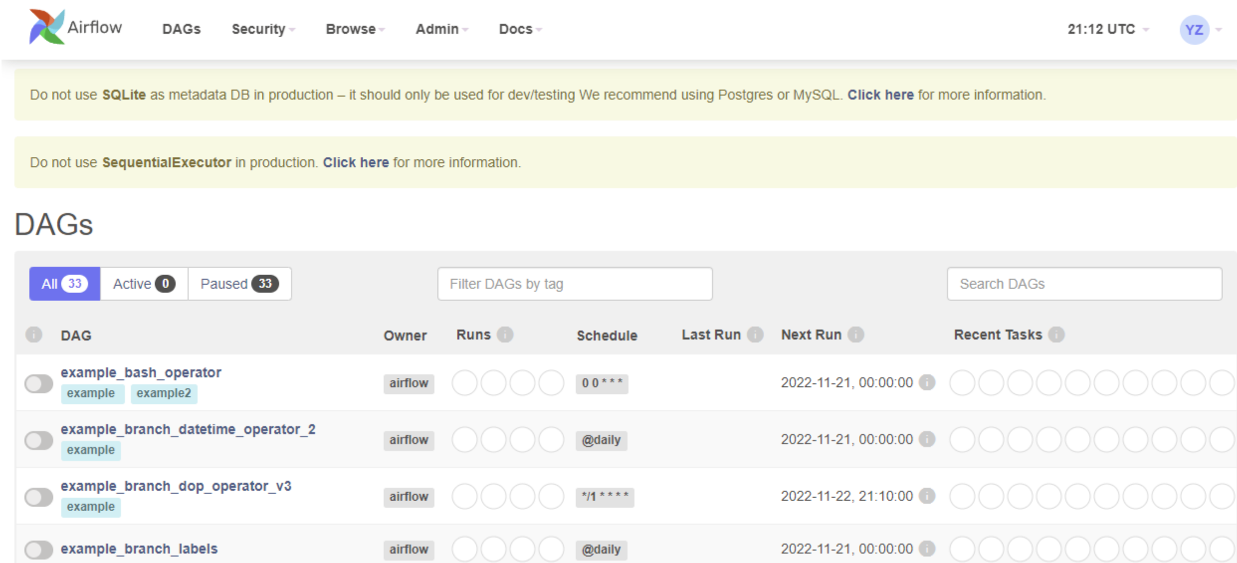
Q1.1

(1)

```
(airflow) yz4359@hw4airflow:~$ airflow webserver --port 8080
/home/yz4359/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/configuration.py:276: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  if StrictVersion(sqlite3.sqlite_version) < StrictVersion(min_sqlite_version):
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2022-11-22 21:09:28,339] {dagbag.py:500} INFO - Filling up the DagBag from /dev/null
[2022-11-22 21:09:28,788] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.can_create Admin
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8080
Timeout: 120
Logfiles: - -
Access Logformat:
=================================================================
/home/yz4359/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/configuration.py:276: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  if StrictVersion(sqlite3.sqlite_version) < StrictVersion(min_sqlite_version):
[2022-11-22 21:09:31 +0000] [9455] [INFO] Starting gunicorn 20.1.0
[2022-11-22 21:09:31 +0000] [9455] [INFO] Listening at: http://0.0.0.0:8080 (9455)
[2022-11-22 21:09:31 +0000] [9455] [INFO] Using worker: sync
[2022-11-22 21:09:31 +0000] [9457] [INFO] Booting worker with pid: 9457
[2022-11-22 21:09:31 +0000] [9458] [INFO] Booting worker with pid: 9458
[2022-11-22 21:09:31 +0000] [9459] [INFO] Booting worker with pid: 9459
[2022-11-22 21:09:32 +0000] [9460] [INFO] Booting worker with pid: 9460
[2022-11-22 21:09:35 +0000] [9455] [INFO] Handling signal: winch
[2022-11-22 21:09:35,716] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.can_create Admin
[2022-11-22 21:09:35,868] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.can_create Admin
[2022-11-22 21:09:36,034] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.can_create Admin
[2022-11-22 21:09:36,078] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists DAG Runs.can_create Admin
```

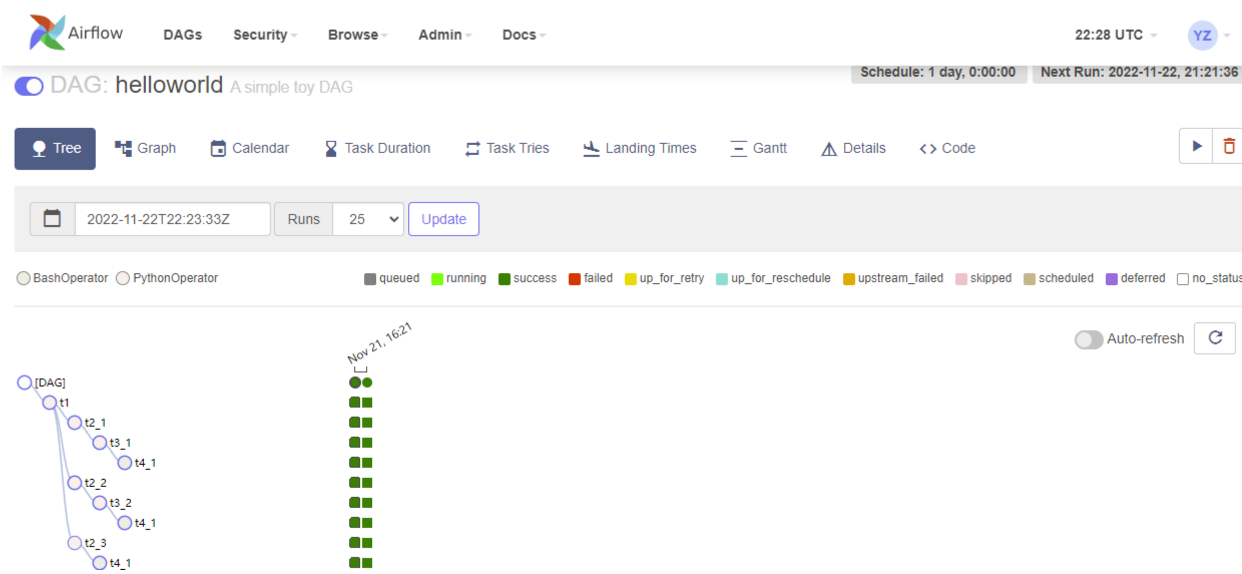
```
(airflow) yz4359@hw4airflow:~$ airflow scheduler
/home/yz4359/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/configuration.py:276: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  if StrictVersion(sqlite3.sqlite_version) < StrictVersion(min_sqlite_version):
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2022-11-22 21:10:25,272] {scheduler_job.py:596} INFO - Starting the scheduler
[2022-11-22 21:10:25,273] {scheduler_job.py:601} INFO - Processing each file at most -1 times
[2022-11-22 21:10:25 +0000] [9467] [INFO] Starting gunicorn 20.1.0
[2022-11-22 21:10:25,284] {manager.py:163} INFO - Launched DagFileProcessorManager with pid: 9468
[2022-11-22 21:10:25 +0000] [9467] [INFO] Listening at: http://0.0.0.0:8793 (9467)
[2022-11-22 21:10:25 +0000] [9467] [INFO] Using worker: sync
[2022-11-22 21:10:25,288] {scheduler_job.py:1115} INFO - Resetting orphaned tasks for active dag runs
[2022-11-22 21:10:25 +0000] [9469] [INFO] Booting worker with pid: 9469
[2022-11-22 21:10:25,309] {settings.py:52} INFO - Configured default timezone Timezone('UTC')
[2022-11-22 21:10:25,332] {manager.py:431} WARNING - Because we cannot use more than 1 thread (parsing_processes = 2 ) when using sqlite. So we set parallelism to 1.
[2022-11-22 21:10:25 +0000] [9470] [INFO] Booting worker with pid: 9470
[2022-11-22 21:10:28 +0000] [9467] [INFO] Handling signal: winch
```

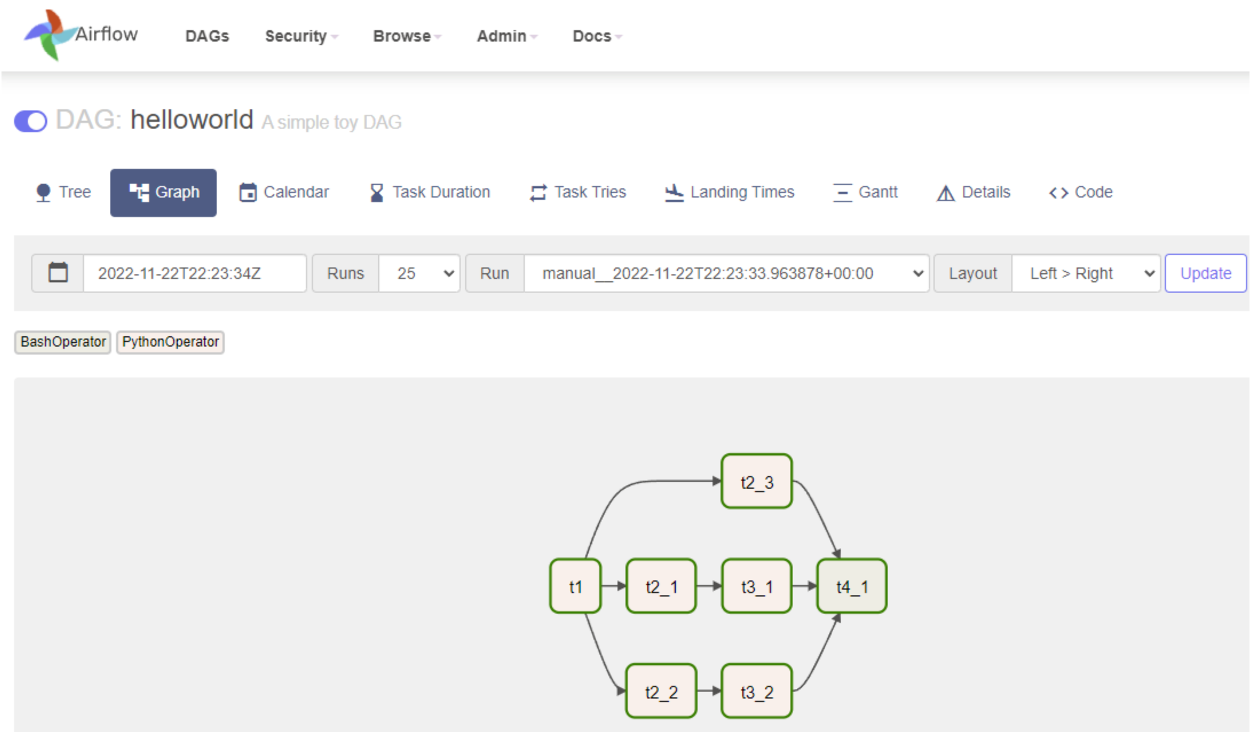
(2)

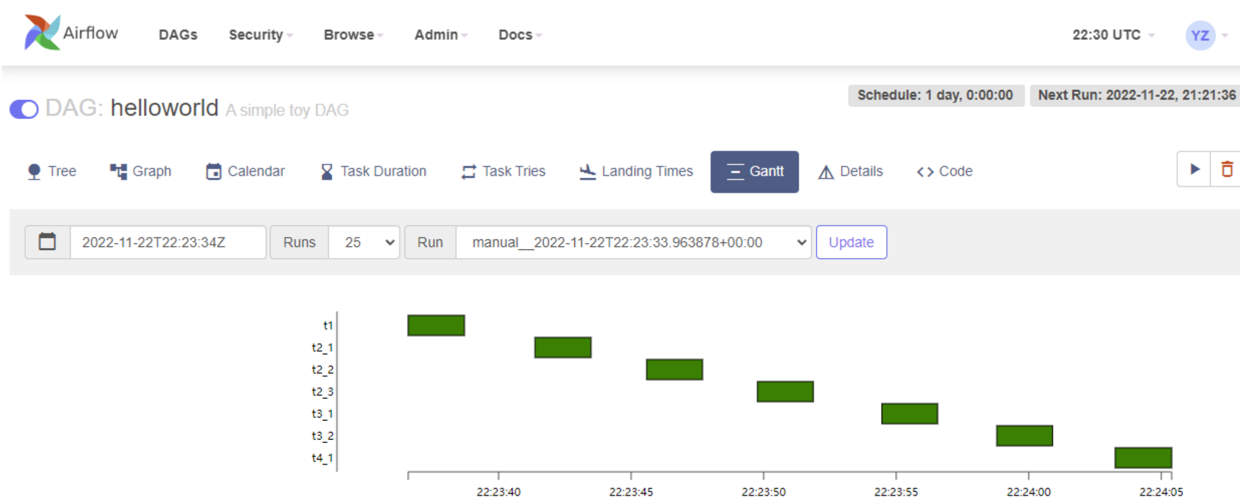


Q1.2

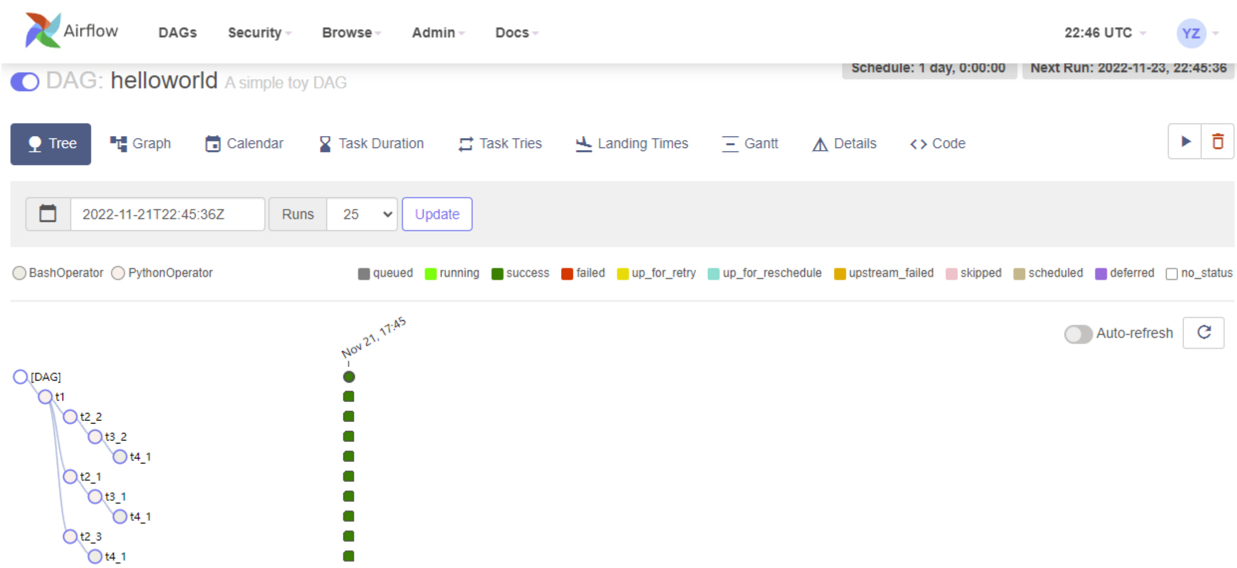
(1)SequentialExecutor

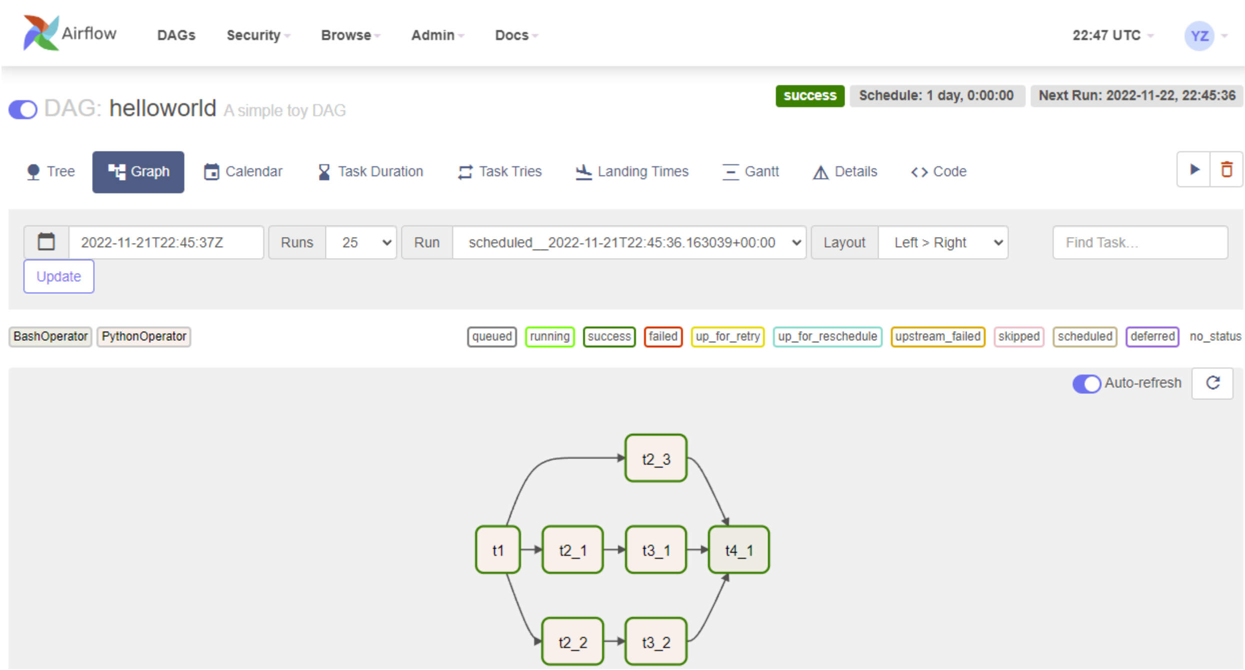


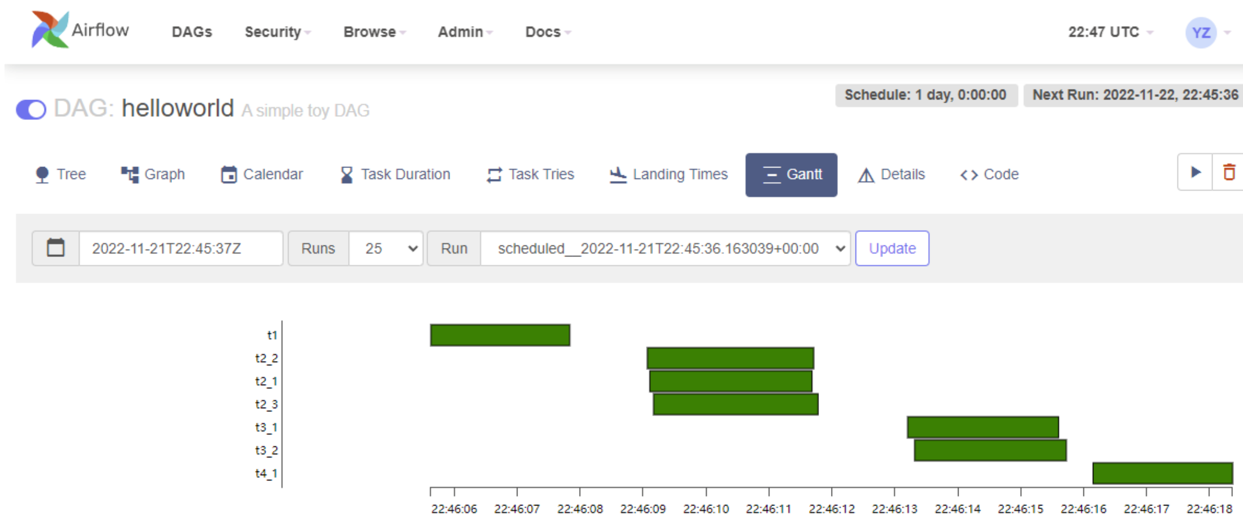




(1)LocalExecutor


Airflow
DAGs
Security
Browse
Admin
Docs
22:46 UTC
YZ
DAG: helloworld A simple toy DAG
Schedule: 1 day, 0:00:00
Next Run: 2022-11-23, 22:45:36
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-21T22:45:36Z
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 21, 17:45
[DAG]
t1
t2_2
t3_2
t4_1
t2_1
t3_1
t4_1
t2_3
t4_1


Airflow
DAGs
Security
Browse
Admin
Docs
22:47 UTC
YZ
DAG: helloworld A simple toy DAG
success
Schedule: 1 day, 0:00:00
Next Run: 2022-11-22, 22:45:36
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-21T22:45:37Z
Runs
25
Run
scheduled__2022-11-21T22:45:36.163039+00:00
Layout
Left > Right
Find Task...
Update
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



(2)

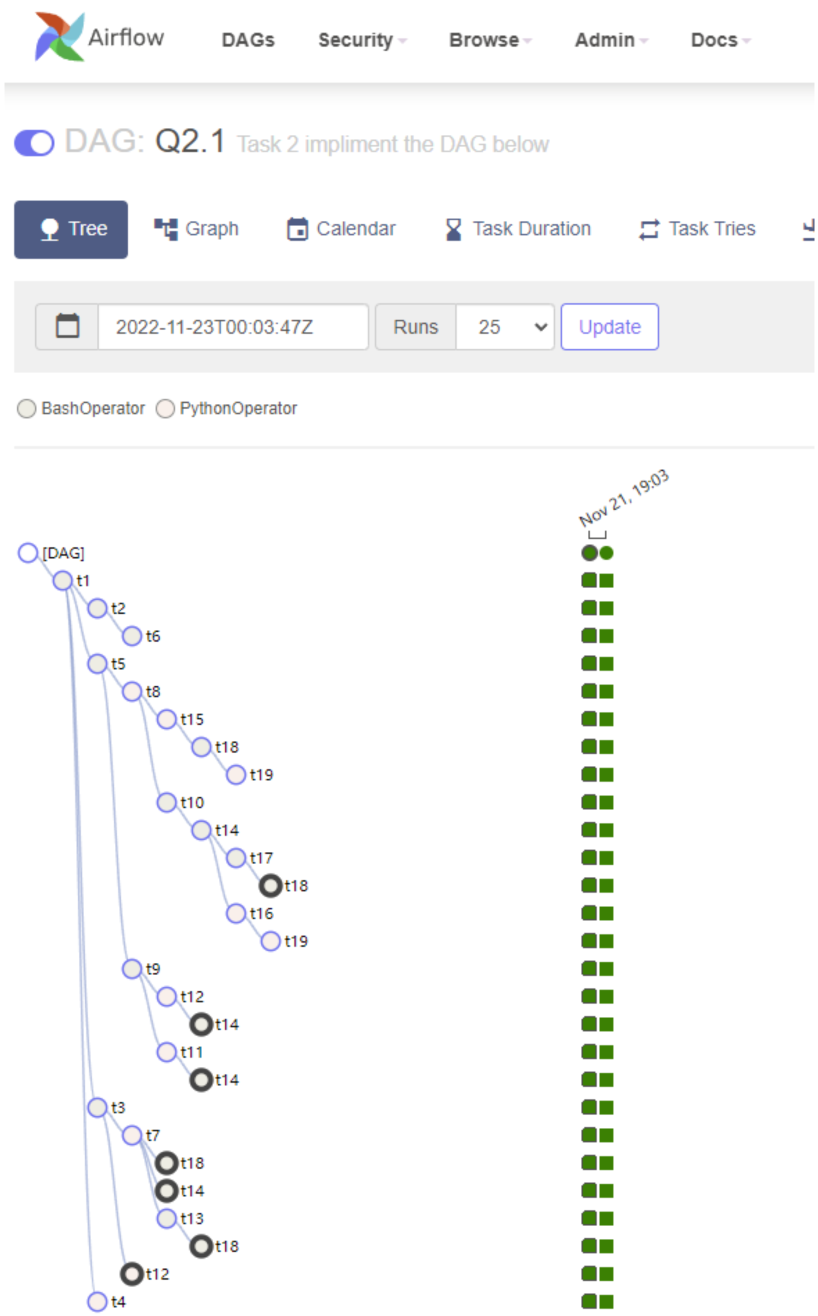
Calender: This is a great features because it allows me to see the history states of job in one place. If we have a large project and fatch data for many days, this function make it clear on which day have successfully executed and which day fail. So, we could use thai data to monitor the pipeline.

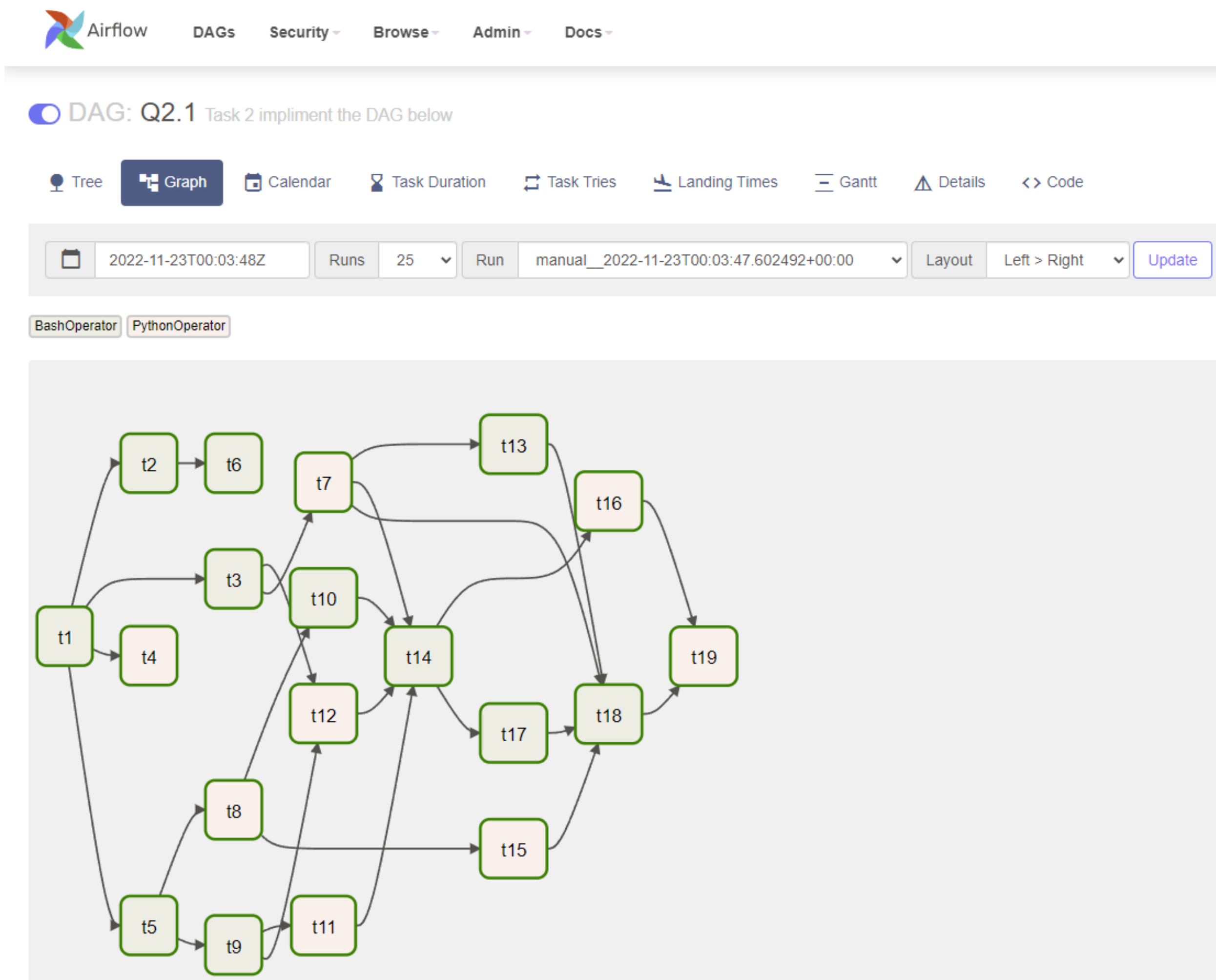
Task Duration: This is a great features because it shows the duration of each sub task in our tree or graph. So, we will know sipecifically which task cost what amount of time. This is great for debugging. For example, when we accidentally wrote a infinite loop or wrtote some bugs that takes a really long time. This would make it clear to identify which task is wrong. Also, this could be used for optimize the runtime.

Task 2 Build workflows

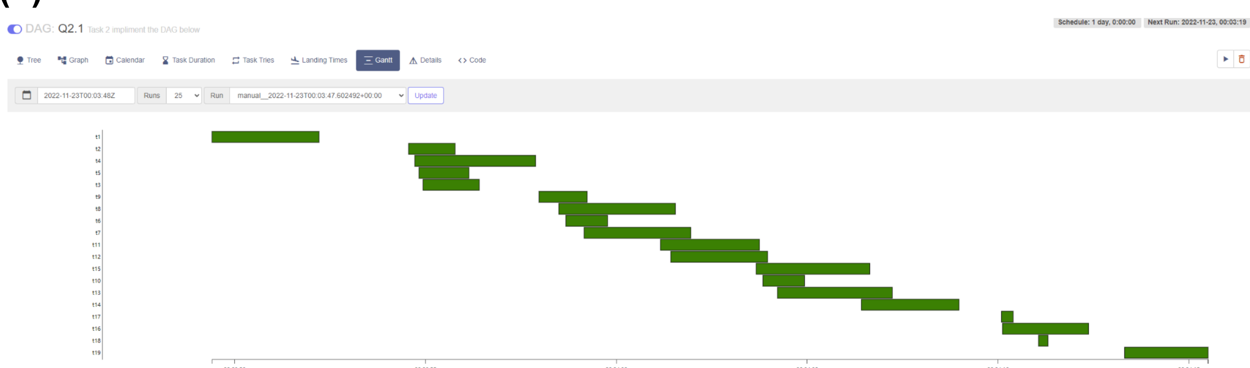
Q2.1

(1)

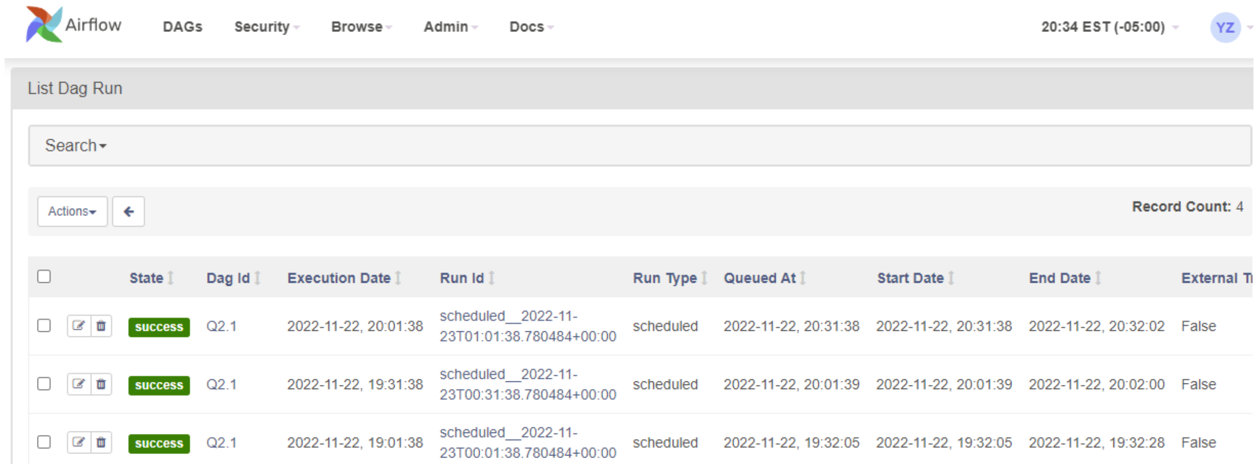

Airflow
DAGs
Security
Browse
Admin
Docs
DAG: Q2.1 Task 2 impliment the DAG below
Tree
Graph
Calendar
Task Duration
Task Tries
2022-11-23T00:03:47Z
Runs
25
Update
BashOperator
PythonOperator
Nov 21, 19:03
[DAG]
t1
t2
t6
t5
t8
t15
t18
t19
t10
t14
t17
t18
t16
t19
t9
t12
t14
t11
t14
t3
t7
t18
t14
t13
t18
t12
t4


Airflow
DAGs
Security
Browse
Admin
Docs
DAG: Q2.1 Task 2 impliment the DAG below
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
2022-11-23T00:03:48Z
Runs
25
Run
manual__2022-11-23T00:03:47.602492+00:00
Layout
Left > Right
Update
BashOperator
PythonOperator
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
t15
t16
t17
t18
t19

(2)


DAG: Q2.1

(3)



Airflow DAGs Security Browse Admin Docs 20:34 EST (-05:00) YZ

List Dag Run

Search

Actions

Record Count: 4

	State	Dag Id	Execution Date	Run Id	Run Type	Queued At	Start Date	End Date	External Tr
	success	Q2.1	2022-11-22, 20:01:38	scheduled__2022-11-23T01:01:38.780484+00:00	scheduled	2022-11-22, 20:31:38	2022-11-22, 20:31:38	2022-11-22, 20:32:02	False
	success	Q2.1	2022-11-22, 19:31:38	scheduled__2022-11-23T00:31:38.780484+00:00	scheduled	2022-11-22, 20:01:39	2022-11-22, 20:01:39	2022-11-22, 20:02:00	False
	success	Q2.1	2022-11-22, 19:01:38	scheduled__2022-11-23T00:01:38.780484+00:00	scheduled	2022-11-22, 19:32:05	2022-11-22, 19:32:05	2022-11-22, 19:32:28	False

I set the start date be the date and time right now(I first change the time zone on my vm to New York) so that it would tart immediately. I set the schedule interval to be 30minutes.

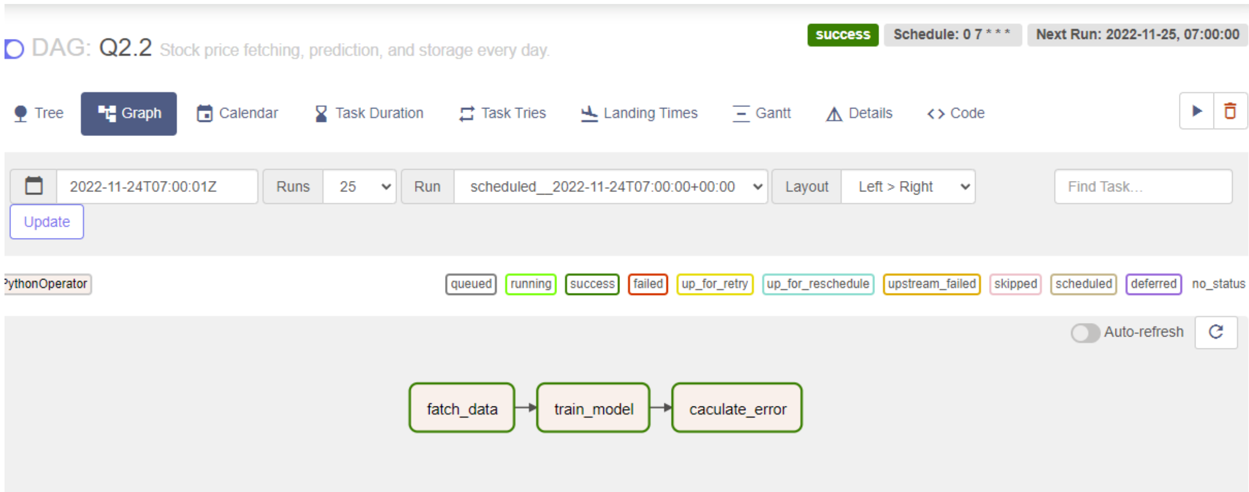
```
with DAG(
    'Q2.1',
    default_args=default_args,
    description='Task 2 impliment the DAG below',
    schedule_interval=timedelta(minutes=30),
    start_date=datetime(2022, 11, 22, 19, 35),
    catchup=False,
    tags=['homework'],
) as dag:
```

Q2.2

I build this workflow by separating our goal to three parts. First, I need to fetch data and push these data with xcom. Second, I need to train the model and get prediction for next day and save it to a file called 'predict.csv'. Third, I need to calculate the error between yesterday's prediction and today's price. I then saved all errors to another file called 'errors.csv'.

I manage cross task communication by using xcom_push and xcom_pull. These command will push key and value pairs to dictionaries in each task. I could access values in each dictionary with task id and keys.

I setup scheduler by specifying ‘start date’ and ‘schedule interval’ when setting up DAG. I let the schedule interval be 7 am UTC each day. Here is my DAG and code.



success	Q2.2	2022-11-29, 07:00:00	scheduled__2022-11-29T07:00:00+00:00	scheduled	2022-11-30, 07:00:00	2022-11-30, 07:00:00	2022-11-30, 07:00:14	False	{}
success	Q2.2	2022-11-28, 07:00:00	scheduled__2022-11-28T07:00:00+00:00	scheduled	2022-11-29, 07:00:00	2022-11-29, 07:00:00	2022-11-29, 07:00:12	False	{}
success	Q2.2	2022-11-27, 07:00:00	scheduled__2022-11-27T07:00:00+00:00	scheduled	2022-11-28, 07:00:00	2022-11-28, 07:00:00	2022-11-28, 07:00:12	False	{}
success	Q2.2	2022-11-26, 07:00:00	scheduled__2022-11-26T07:00:00+00:00	scheduled	2022-11-27, 07:00:00	2022-11-27, 07:00:00	2022-11-27, 07:00:11	False	{}
success	Q2.2	2022-11-25, 07:00:00	scheduled__2022-11-25T07:00:00+00:00	scheduled	2022-11-26, 07:00:00	2022-11-26, 07:00:00	2022-11-26, 07:00:13	False	{}
success	Q2.2	2022-11-24, 07:00:00	scheduled__2022-11-24T07:00:00+00:00	scheduled	2022-11-25, 07:00:00	2022-11-25, 07:00:00	2022-11-25, 07:00:14	False	{}

```python
from datetime import datetime, timedelta, date
from textwrap import dedent
import time
import yfinance as yf
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
from csv import writer, reader
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator

def fatch_data(**context):
    symbles = ["AAPL", "GOOGL", "META", "MSFT", "AMZN"]
    for s in symbles:
        tickers = yf.Tickers(s)
        data = tickers.tickers[s].info
        open_price = data["open"]
        high_price = data["dayHigh"]
        low_price = data["dayLow"]
        close_price = data["previousClose"]
        volume = data["volume"]
        context['ti'].xcom_push(key=s, value=[open_price, high_price, low_price, close_price, volume])

def train_model(**context):
    symbles = ["AAPL", "GOOGL", "META", "MSFT", "AMZN"]
    all_predict = []
    for s in symbles:
        todayPrice = context['ti'].xcom_pull(key=s,task_ids="fatch_data")
        data = yf.download(tickers = s, period = "11d", interval = "1d")
        data = data.drop(columns=["Adj Close"])
        x = data[:-1]
        y = data["High"][1:]
        reg = LinearRegression().fit(x, y)
        print(f"Current model accuracy: {reg.score(x, y)}")
        features = pd.DataFrame([todayPrice])
        prediction = reg.predict(features)[0]
        print(f"Prediction for next day's high: {prediction}")
        all_predict.append(prediction)
```

```python
            context['ti'].xcom_push(key=s, value=prediction)
    today = date.today()
    d = today.strftime("%m/%d/%y")
    write = [d] + all_predict
    with open(f'predict.csv','a') as file:
        writer_object = writer(file)
        writer_object.writerow(write)
        file.close()

def caculate_error(**context):
    symbles = ["AAPL", "GOOGL", "META", "MSFT", "AMZN"]
    errors = []
    predictions = []
    today = date.today()
    d = today.strftime("%m/%d/%y")
    yesterday = today - timedelta(days = 1)
    yesterday = yesterday.strftime("%m/%d/%y")
    with open('predict.csv', 'r') as file:
        reader_object = reader(file)
        for row in reader_object:
            if row and row[0] == yesterday:
                predictions = row[1:]
                break
    print(f"Yesterday's prediction: {predictions}")
    for i in range(len(symbles)):
        high_price = context['ti'].xcom_pull(key=symbles[i],task_ids="fatch_data")[1]
        error = (float(predictions[i]) - float(high_price)) / float(high_price)
        errors.append(error)
    write = [d] + errors
    with open(f'errors.csv','a') as file:
        writer_object = writer(file)
        writer_object.writerow(write)
        file.close()
    print(f"Today's error: {write}")

default_args = {
    'owner': 'yutao',
    'depends_on_past': False,
    'email': ['yz4359@columbia.edu'],
```

```python
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(seconds=3),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}

with DAG(
    'Q2.2',
    default_args=default_args,
    description='Stock price fetching, prediction, and storage every day.',
    schedule_interval='0 7 * * *',
    start_date=datetime(2022, 11, 24),
    catchup=False,
    tags=['homework'],
) as dag:

    fatch_data = PythonOperator(
        task_id='fatch_data',
        python_callable=fatch_data,
        retries=3,
        provide_context=True
    )

    caculate_error = PythonOperator(
        task_id='caculate_error',
        python_callable=caculate_error,
```

```
            retries=3,
            provide_context=True
    )

    train_model = PythonOperator(
        task_id='train_model',
        python_callable=train_model,
        retries=3,
        provide_context=True
    )

    fatch_data >> train_model
    train_model >> caculate_error
```

Task 3 Written parts

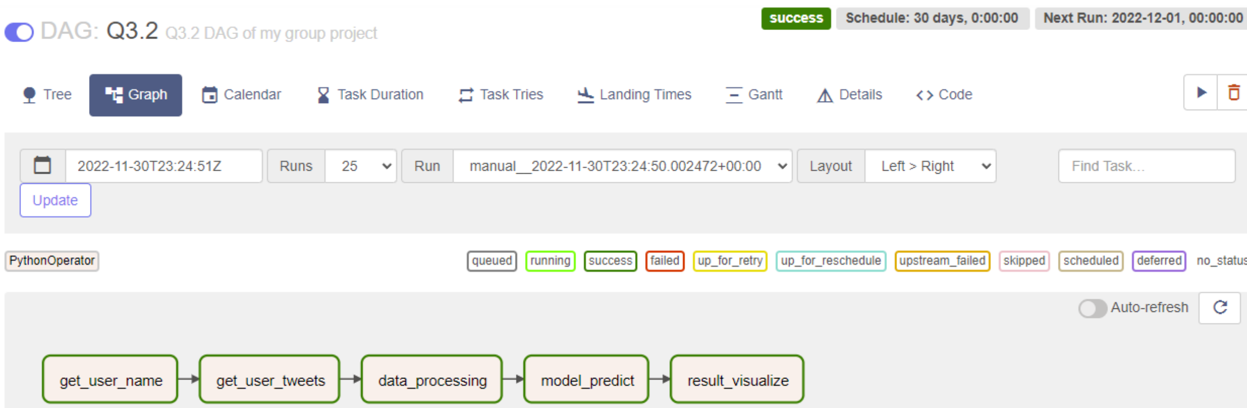
Q3.1

(1)

Executor	Pros	Cons
SequentialExecutor	Every task is in sequential. It is very easy to debug and write code since there can be only one task executing at any given time(even branch task from same root).	When there are many task or some task have many dependency, SequentialExecutorw ould be really slow. A task will wait for all of its depended task finish executing one by one to continue.
LocalExecutor	Execute task in parallel. It would be a	This would be harder for debugging and

	lot faster and utilizing more computational resource when executing task.	programing. We do not know which task will finish executing before hand. As a result we have to carefully write dependency and make sure task would not collide with each other(read and write same file at same time). We have to use lock when necessary to avoid collision.
CeleryExecutor	Most mature option because it is the oldest adoption. Many resource online since a lot of people and company are using it.	It require infrastructure support to work. Need Celery and Celery's backend.
KubernetesExecutor	Able to use different docker images for different task. More flexibility. Works well with Kubernetes ecosystem.	Not as robest as other executor because it is newer.

Q3.2



(3)

We will schedule our tasks to run every month(every 30 days as shown in the screen shoot).