EECS 6893 Big Data Analytics HW4 Data Analytics Pipeline

Nov 29th, 2022
Lijie Huang
lh3158

Task 1 Helloworld

Q1.1

Read through the tutorial slides and install Airflow either on your local laptop or on a VM of GCP. You can also use google cloud composer if you know how to use that.

(1)

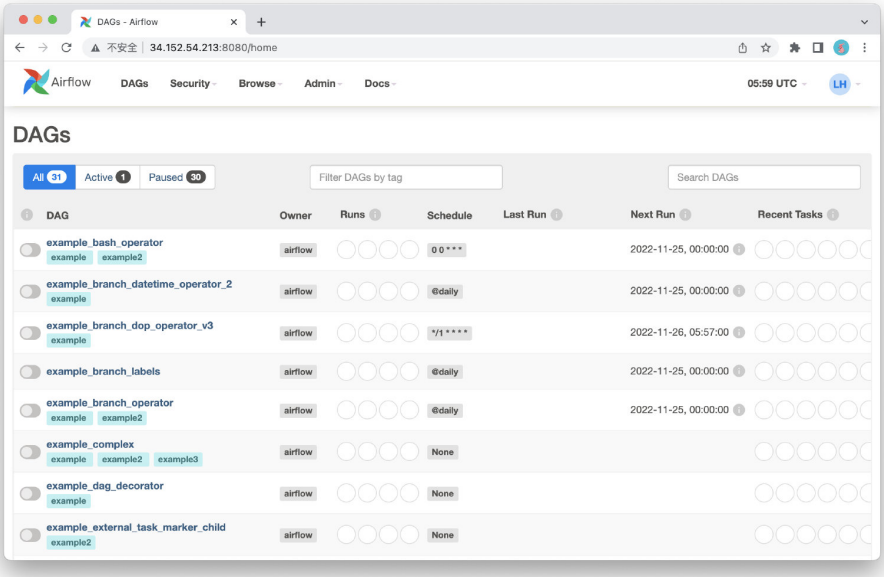
Provide screenshots of terminals after you successfully start the webserver and scheduler.

```
(airflow) youthtoday0@test:~$ airflow webserver --port 8080
/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/configuration.py:276: Deprecation
Warning: distutils Version classes are deprecated. Use packaging.version instead.
  if StrictVersion(sqlite3.sqlite_version) < StrictVersion(min_sqlite_version):
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2022-11-26 05:54:00,502] {dagbag.py:500} INFO - Filling up the DagBag from /dev/null
[2022-11-26 05:54:00,841] {manager.py:512} WARNING - Refused to delete permission view, assoc with role exists
DAG Runs.can_create Admin
Traceback (most recent call last):
  File "/home/youthtoday0/miniconda3/envs/airflow/bin/airflow", line 8, in <module>
    sys.exit(main())
  File "/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/__main__.py", line 48, in
 main
    args.func(args)
  File "/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/cli/cli_parser.py", line
48, in command
    return func(*args, **kwargs)
  File "/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/utils/cli.py", line 92, i
n wrapper
    return f(*args, **kwargs)
  File "/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/cli/commands/webserver_co
mmand.py", line 368, in webserver
    check_if_pidfile_process_is_running(pid_file=pid_file, process_name="webserver")
  File "/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/utils/process_utils.py",
line 272, in check_if_pidfile_process_is_running
    raise AirflowException(f"The {process_name} is already running under PID {pid}.")
airflow.exceptions.AirflowException: The webserver is already running under PID 20476.
```

```
(airflow) youthtoday0@test:~$ airflow scheduler
/home/youthtoday0/miniconda3/envs/airflow/lib/python3.8/site-packages/airflow/configuration.py:276: Deprecation
Warning: distutils Version classes are deprecated. Use packaging.version instead.
  if StrictVersion(sqlite3.sqlite_version) < StrictVersion(min_sqlite_version):
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2022-11-26 04:38:40,097] {scheduler_job.py:596} INFO - Starting the scheduler
[2022-11-26 04:38:40,098] {scheduler_job.py:601} INFO - Processing each file at most -1 times
[2022-11-26 04:38:40 +0000] [20657] [INFO] Starting gunicorn 20.1.0
[2022-11-26 04:38:40 +0000] [20657] [INFO] Listening at: http://0.0.0.0:8793 (20657)
[2022-11-26 04:38:40 +0000] [20657] [INFO] Using worker: sync
[2022-11-26 04:38:40,108] {manager.py:163} INFO - Launched DagFileProcessorManager with pid: 20658
[2022-11-26 04:38:40,112] {scheduler_job.py:1115} INFO - Resetting orphaned tasks for active dag runs
[2022-11-26 04:38:40 +0000] [20659] [INFO] Booting worker with pid: 20659
[2022-11-26 04:38:40,131] {settings.py:52} INFO - Configured default timezone Timezone('UTC')
[2022-11-26 04:38:40,149] {manager.py:431} WARNING - Because we cannot use more than 1 thread (parsing_processe
s = 2 ) when using sqlite. So we set parallelism to 1.
[2022-11-26 04:38:40 +0000] [20660] [INFO] Booting worker with pid: 20660
[2022-11-26 04:43:40,317] {scheduler_job.py:1115} INFO - Resetting orphaned tasks for active dag runs
[2022-11-26 04:47:24,701] {dag.py:2921} INFO - Setting next_dagrun for helloworld to 2022-11-26T04:47:05.480580
+00:00
[2022-11-26 04:47:24,753] {scheduler_job.py:288} INFO - 1 tasks up for execution:
        <TaskInstance: helloworld.t1 scheduled__2022-11-25T04:47:05.480580+00:00 [scheduled]>
[2022-11-26 04:47:24,755] {scheduler_job.py:317} INFO - Figuring out tasks to run in Pool(name=default_pool) wi
th 128 open slots and 1 task instances ready to be queued
[2022-11-26 04:47:24,755] {scheduler_job.py:345} INFO - DAG helloworld has 0/16 running and queued tasks
[2022-11-26 04:47:24,755] {scheduler_job.py:410} INFO - Setting the following tasks to queued state:
        <TaskInstance: helloworld.t1 scheduled__2022-11-25T04:47:05.480580+00:00 [scheduled]>
[2022-11-26 04:47:24,757] {scheduler_job.py:450} INFO - Sending TaskInstanceKey(dag_id='helloworld', task_id='t
1', run_id='scheduled__2022-11-25T04:47:05.480580+00:00', try_number=1) to executor with priority 7 and queue d
efault
[2022-11-26 04:47:24,757] {base_executor.py:82} INFO - Adding to queue: ['airflow', 'tasks', 'run', 'helloworld
', 't1', 'scheduled__2022-11-25T04:47:05.480580+00:00', '--local', '--subdir', 'DAGS_FOLDER/helloworld.py']
[2022-11-26 04:47:24,764] {sequential_executor.py:59} INFO - Executing command: ['airflow', 'tasks', 'run', 'he
lloworld', 't1', 'scheduled__2022-11-25T04:47:05.480580+00:00', '--local', '--subdir', 'DAGS_FOLDER/helloworld.
py']
```

(2)

Provide screenshots of the web browser after you successfully login and see the DAGs.



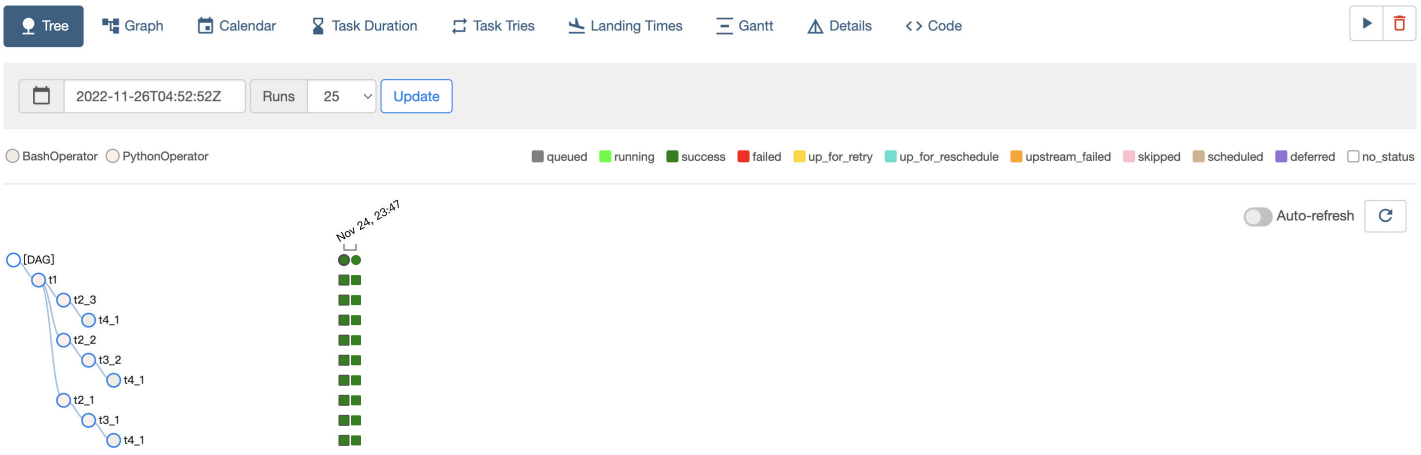
Q1.2

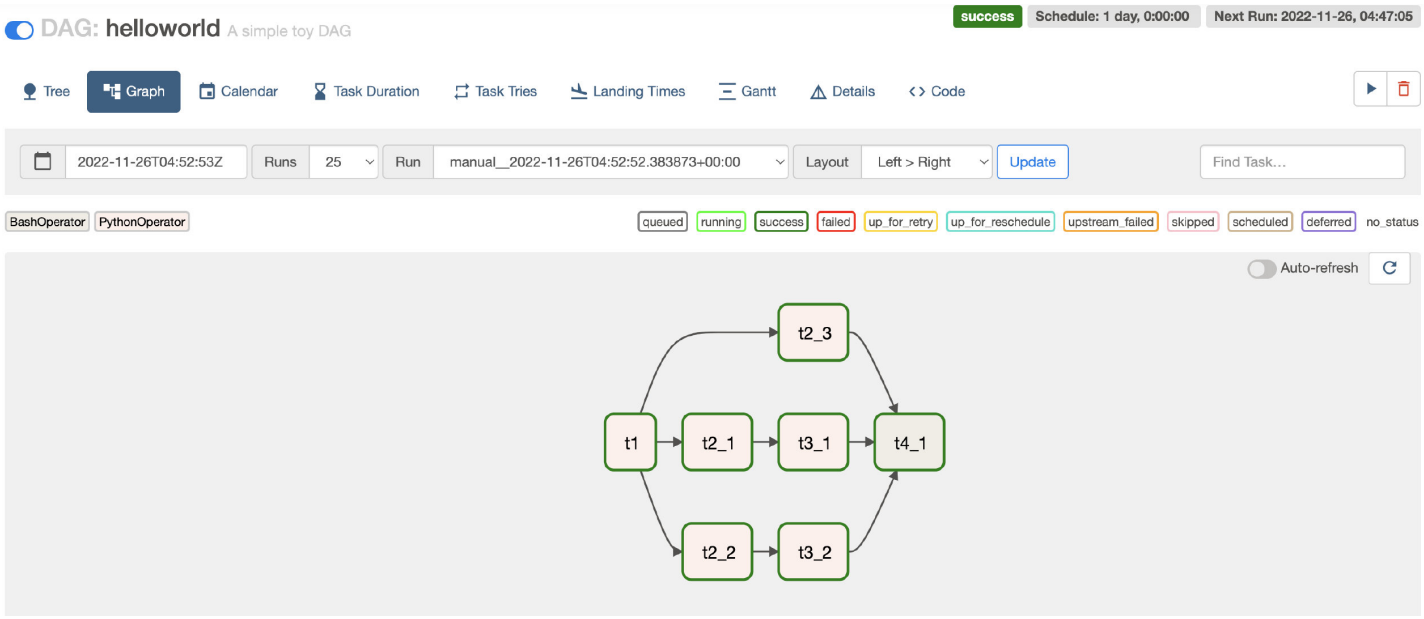
Run helloworld with SequentialExecutor and LocalExecutor.

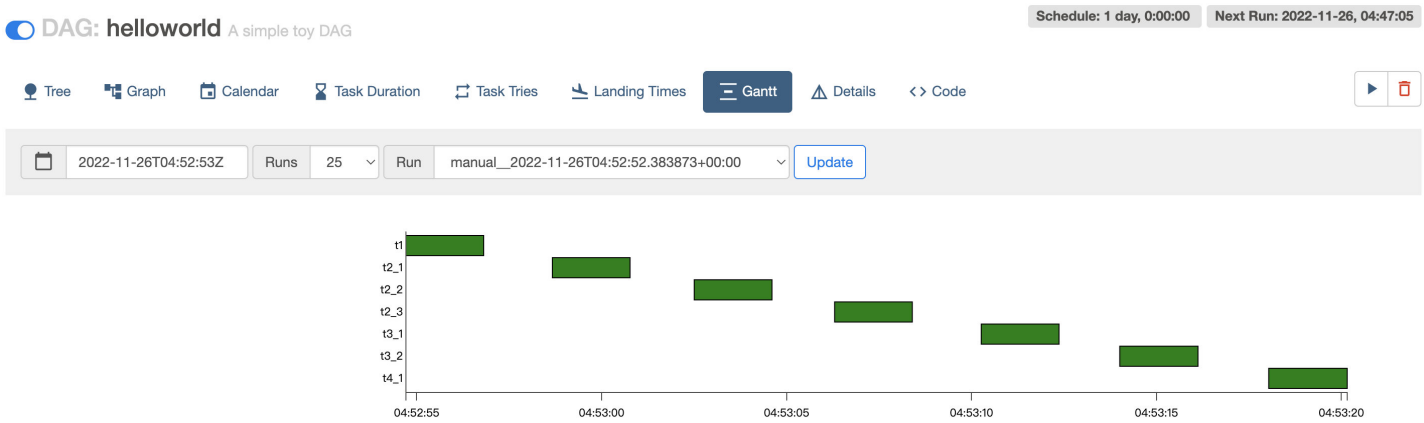
(1)

Provide screenshots of Tree, Graph, and Gantt of each executor.

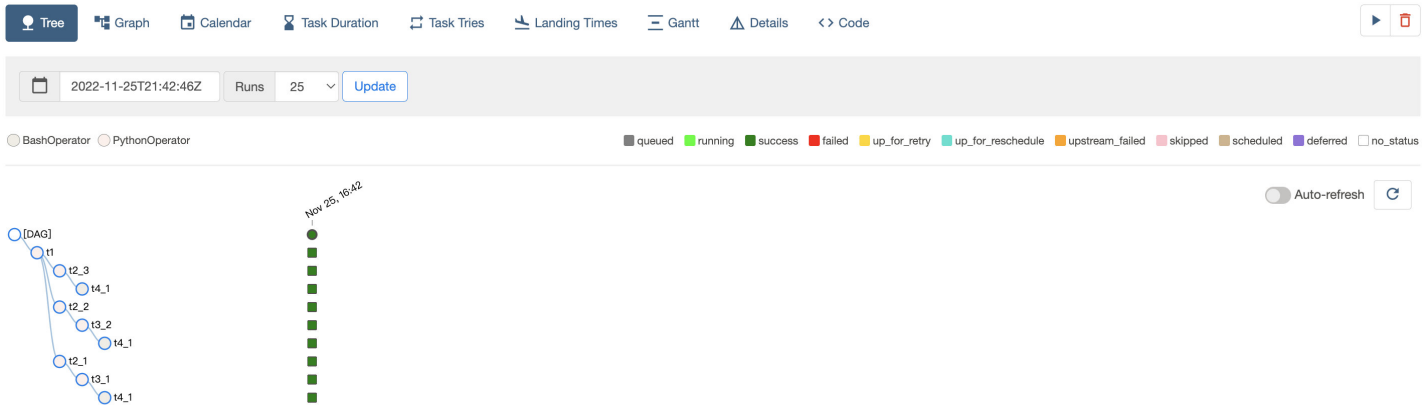
Answer:
For SequentialExecutor

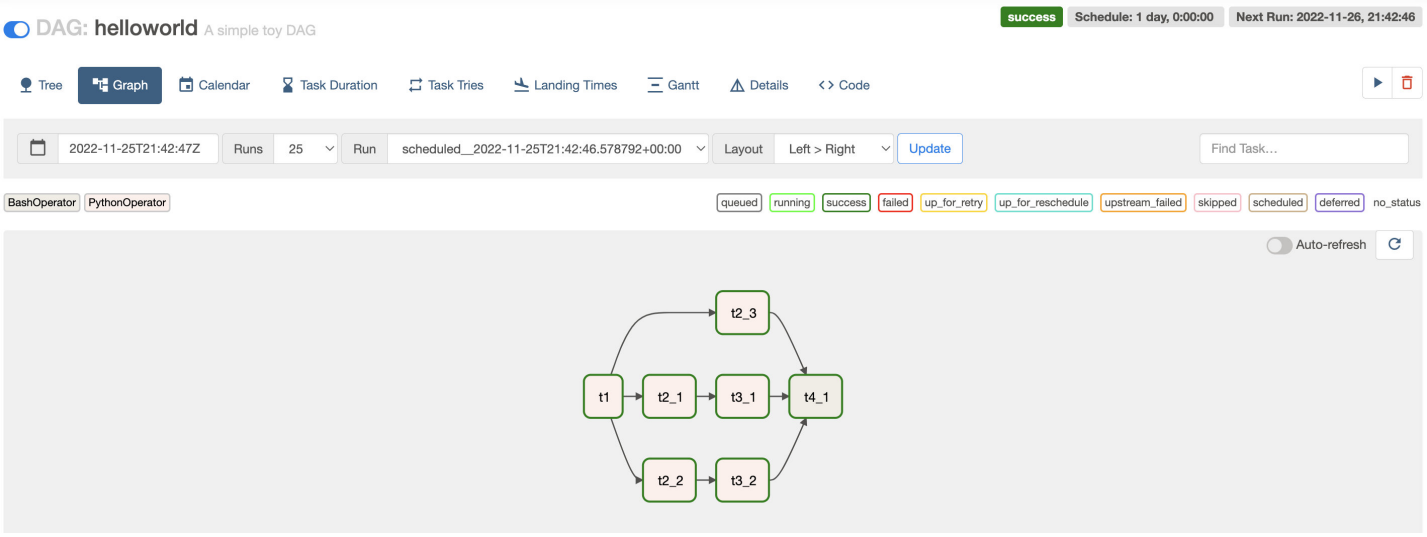


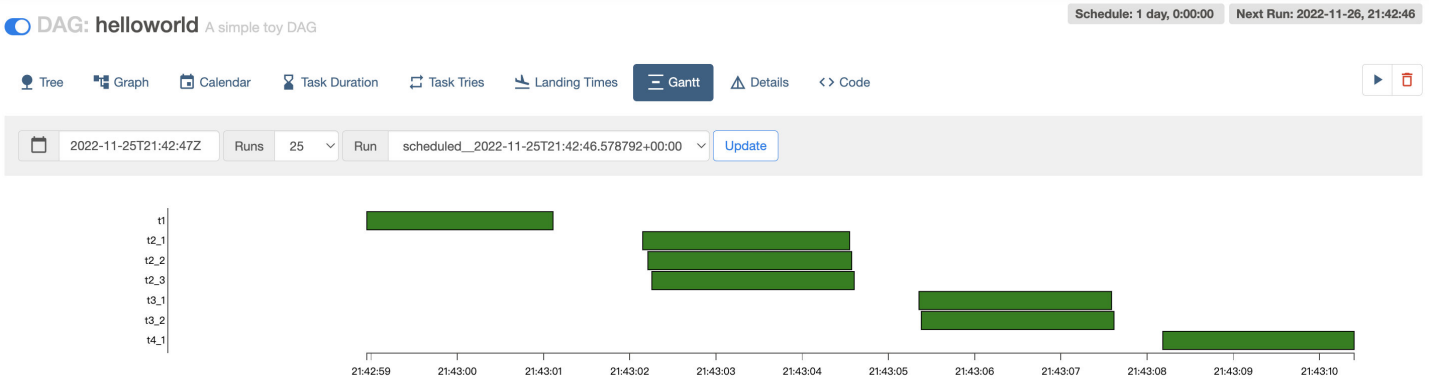




For LocalExecutor





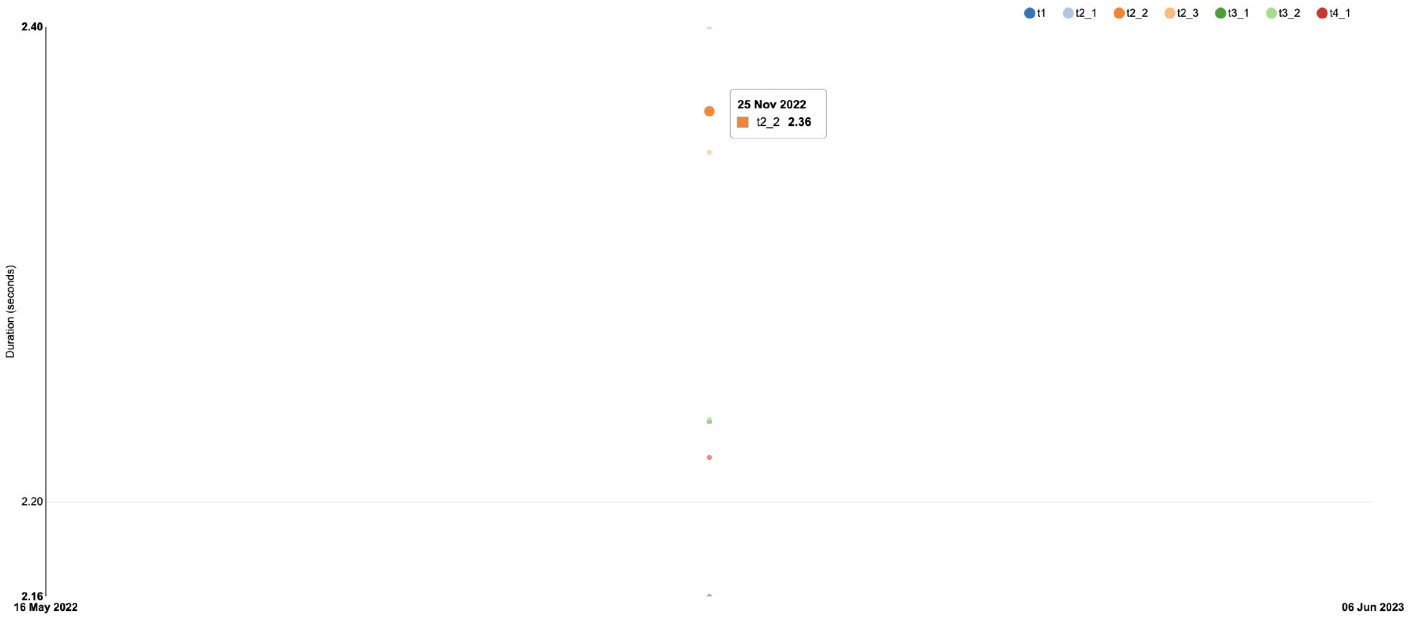


(2)

Explore other features and visualizations you can find in the Airflow UI. Choose two features/visualizations (other than tree, graph, and Gantt), explain their functions and how they help monitor and troubleshoot the pipeline, use helloword as an example.

Answer:

Task Duration: The duration of your different tasks over the past N runs. This view lets you find outliers and quickly understand where the time is spent in your DAG over many runs. For example, in helloworld case, I can see t2_2 spend the most time to execute.



DAG Details: It helps you to know the real time details about the program, including max active runs, concurrency and tasks count. For example, in helloworld case, I can see the concurrency is 16 and the total tasks are 7. All tasks are successfully finished.

DAG Details

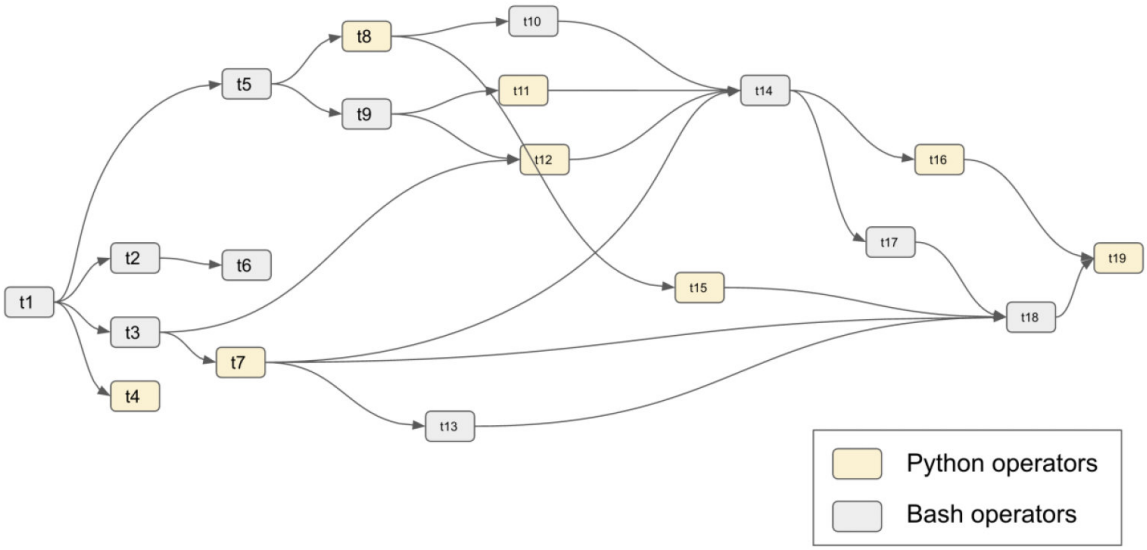
success 7

Schedule Interval	1 day, 0:00:00
Catchup	False
Started	True
End Date	None
Max Active Runs	0 / 16
Concurrency	16
Default Args	{'depends_on_past': False, 'email': ['yl4860@columbia.edu'], 'email_on_failure': False, 'email_on_retry': False, 'owner': 'yunhang', 'retries': 1, 'retry_delay': datetime.timedelta(seconds=30)}
Tasks Count	7
Task IDs	['t1', 't2_1', 't2_2', 't2_3', 't3_1', 't3_2', 't4_1']
Relative file location	helloworld.py
Owner	yunhang
DAG Run Timeout	None
Tags	example

Task 2 Build workflows

Q2.1

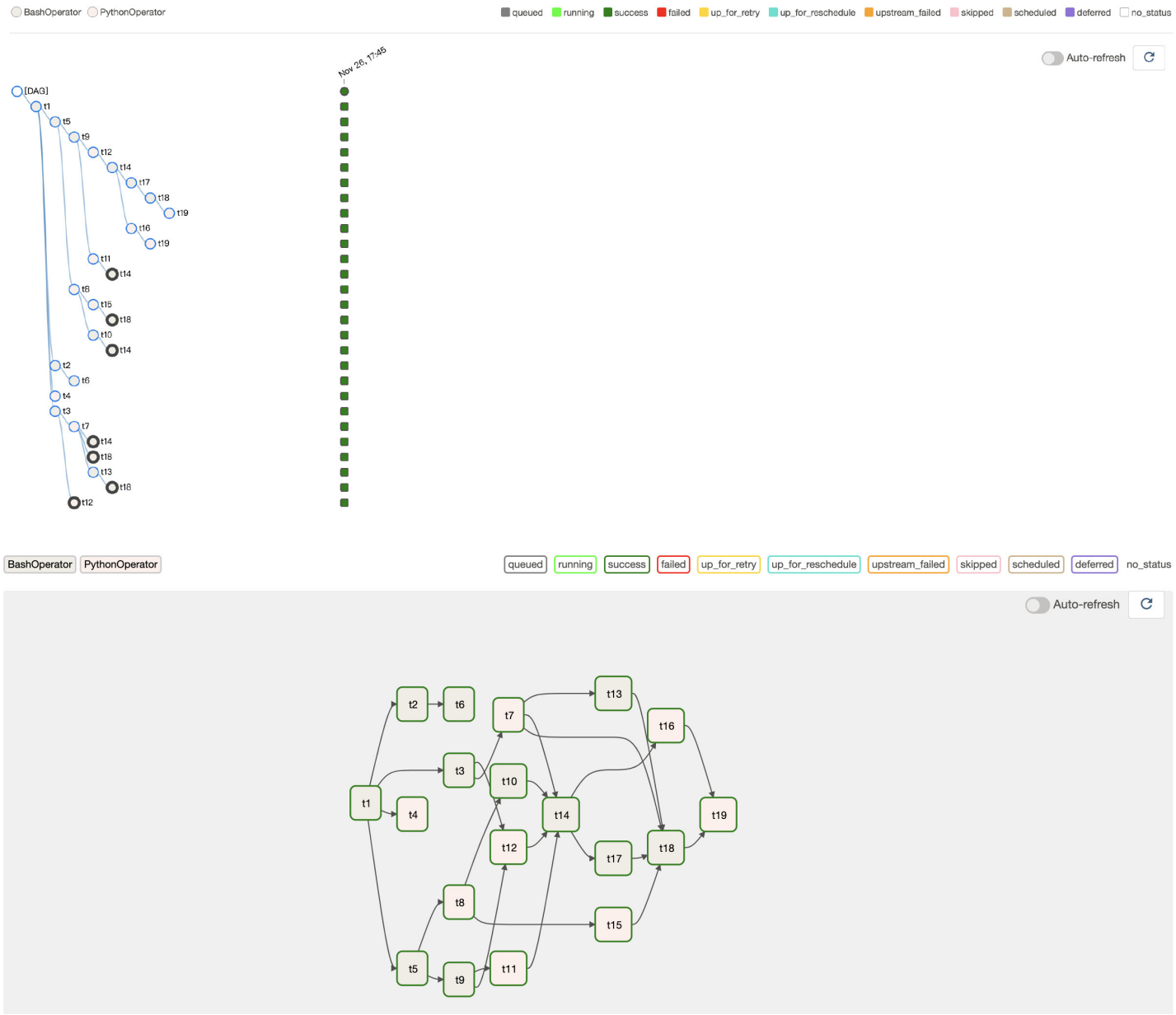
Implement the DAG below



For each kind of operator, use at least 3 different commands. For example, you can choose sleep, print, count functions for Python operators, and echo, run bash script, run python file for Bash operators.

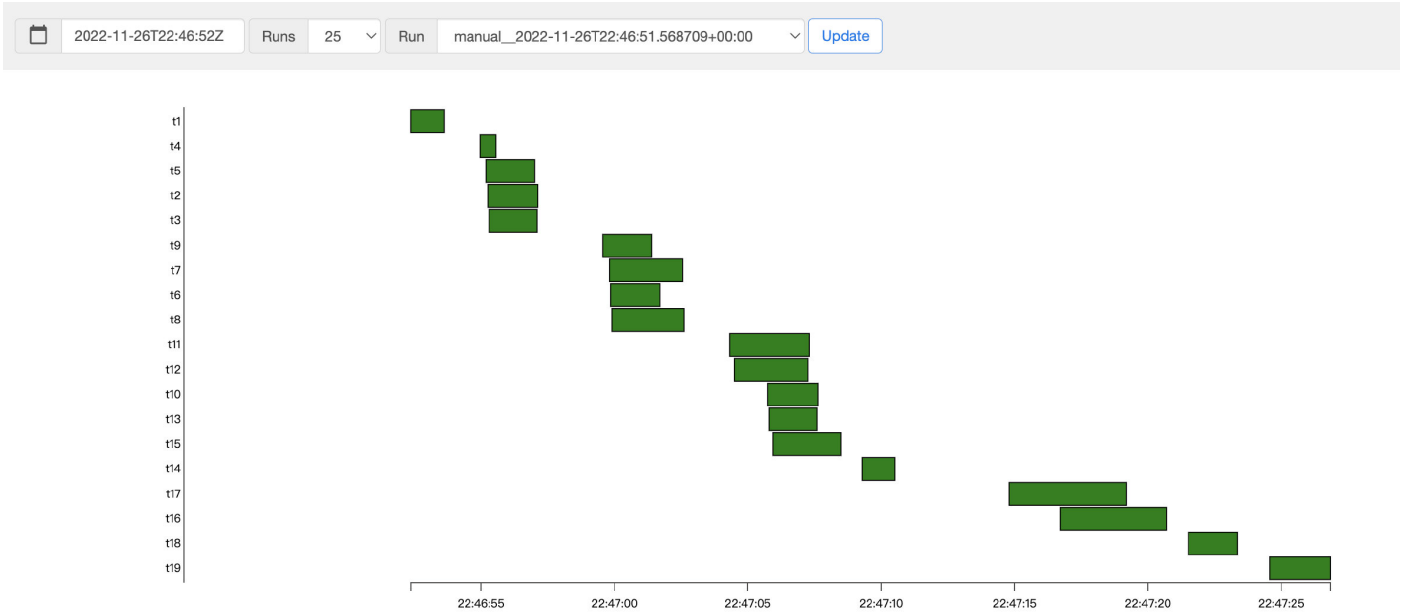
(1)

Provide screenshots of Tree and Graph in airflow. (10 pts)



(2)

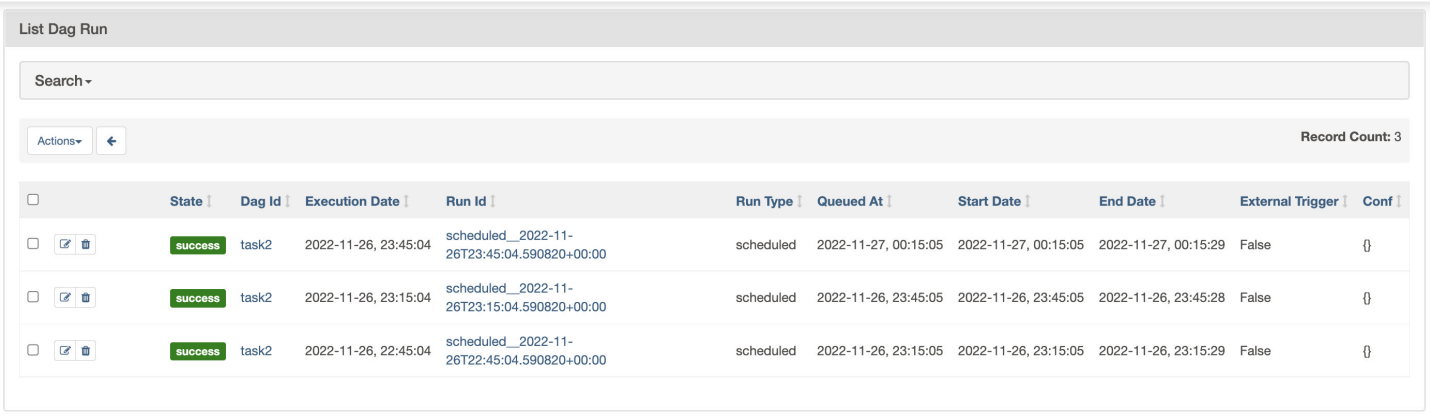
Manually trigger the DAG, and provide screenshots of Gantt. (10 pts)



(3)

Schedule the first run immediately and running the program every 30 minutes. Describe how you decide the start date and schedule interval. Provide screenshots of running history after two repeats (first run + 2 repeats). On your browser, you can find the running history. (5 pts)

Answer:
The scheduler runs the job one 'schedule_interval' AFTER the start date, at the END of the period. So I set the param 'start_date' to datetime(2021, 1, 1), which means the job starts immediately. Another param 'schedule_interval' determins the period betwin two repeating jobs. So I set 'schedule_interval' to 30 minitues as timedelta(minutes=30).



State	Dag Id	Execution Date	Run Id	Run Type	Queued At	Start Date	End Date	External Trigger	Conf
success	task2	2022-11-26, 23:45:04	scheduled__2022-11-26T23:45:04.590820+00:00	scheduled	2022-11-27, 00:15:05	2022-11-27, 00:15:05	2022-11-27, 00:15:29	False	{}
success	task2	2022-11-26, 23:15:04	scheduled__2022-11-26T23:15:04.590820+00:00	scheduled	2022-11-26, 23:45:05	2022-11-26, 23:45:05	2022-11-26, 23:45:28	False	{}
success	task2	2022-11-26, 22:45:04	scheduled__2022-11-26T22:45:04.590820+00:00	scheduled	2022-11-26, 23:15:05	2022-11-26, 23:15:05	2022-11-26, 23:15:29	False	{}

Q2.2

Stock price fetching, prediction, and storage every day (25 pts)
(1) Schedule fetching the stock price of [AAPL, GOOGL, FB, MSFT, AMZN] at 7:00 AM every day. Use Yahoo! Finance data downloader https://pypi.org/project/yfinance/.
(2) Preprocess data if you think necessary.
(3) Train/update 5 linear regression models for stock price prediction for these 5 corporates. Each linear model takes the "open price", "high price", "low price", "close price", "volume" of the corporate in the past ten days as the features and predicts the "high price" for the next day.
(4) Every day if you get new data, calculate the relative errors, i.e., (prediction yesterday - actual price today) / actual price today, and update the date today and 5 errors into a table, e.g., a csv file.
(5) Provide screenshots of your code. Describe briefly how to build this workflow, e.g., what the DAG is, how you manage the cross tasks communication, how you setup scheduler...

Answer:

```
from datetime import datetime as dt, timedelta
import datetime
import yfinance as yf
from airflow import DAG
from airflow.operators.python import PythonOperator
from sklearn import linear_model
import csv

##################################################
# DEFINE PYTHON FUNCTIONS
##################################################
aapl_err = 0
googl_err = 0
meta_err = 0
msft_err = 0
amzn_err = 0
work_day = True
today = datetime.date.today()

def predict_high(stock_name, today):
    stock = yf.Ticker(stock_name)
    data = stock.history(start=today+datetime.timedelta(days=-30), end=today, interval="1d")
    size = data.shape[0]
    data_x = data.iloc[size-11:size-1, 0:4]
    data_y = data.iloc[size-10:size, 1]
    regr = linear_model.LinearRegression()
    regr.fit(data_x, data_y)
    test_x = [data.iloc[size-1, 0:4]]
    pred = regr.predict(test_x)
    return pred


def get_error(stock_name, today):
    pred = predict_high(stock_name, today)
```

```
    stock = yf.Ticker(stock_name)
    data = stock.history(start=today + datetime.timedelta(days=-30), end=today, interval="1d")
    size = data.shape[0]
    gt = data.iloc[size - 1, 2]
    error = (pred[0] - gt) / gt
    return error

def is_workday():
    global work_day
    if yf.Ticker('aapl').history(start=today + datetime.timedelta(days=-1), end=today,
interval="1d").empty:
        work_day = False

def process(**kwargs):
    stock_name = kwargs['key']
    if stock_name == "aapl":
        global aapl_err
        aapl_err = get_error(stock_name, today)
    elif stock_name == "googl":
        global googl_err
        googl_err = get_error(stock_name, today)
    elif stock_name == "meta":
        global meta_err
        meta_err = get_error(stock_name, today)
    elif stock_name == "msft":
        global msft_err
        msft_err = get_error(stock_name, today)
    else:
        global amzn_err
        amzn_err = get_error(stock_name, today)


def write_to_csv():
    errors = [today + datetime.timedelta(days=-1), aapl_err, googl_err, meta_err, msft_err,
amzn_err]
    with open('text.csv', mode='a', encoding='UTF8', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(errors)

########################################
# DEFINE AIRFLOW DAG (SETTINGS + SCHEDULE)
########################################

default_args = {
    'owner': 'lijie',
    'depends_on_past': False,
    'email': ['lh3158@columbia.edu'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(seconds=30),
}

with DAG(
        'stock_airflow',
        default_args=default_args,
        description='A simple toy DAG',
        schedule_interval=timedelta(days=1),
        start_date=dt(2021, 1, 1, 7, 0),
```

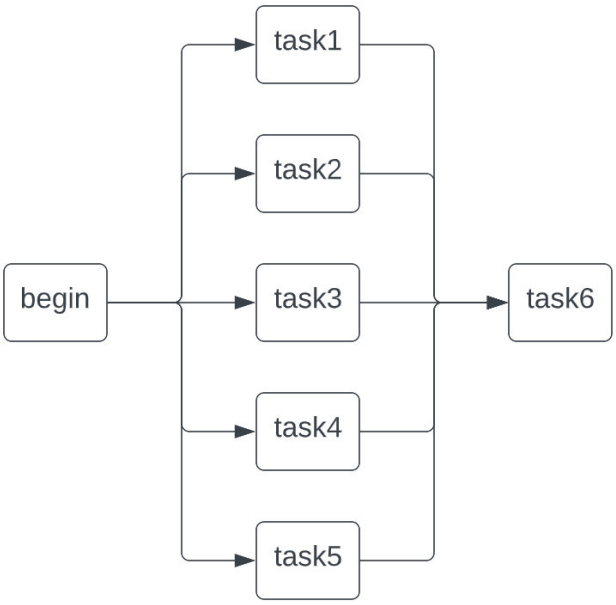
```
        catchup=False,
        tags=['example'],
) as dag:
    ########################################
    # DEFINE AIRFLOW OPERATORS
    ########################################

    begin = PythonOperator(
        task_id='work_day',
        python_callable=is_workday,
    )

    task1 = PythonOperator(
        task_id='aapl',
        python_callable=process,
        op_kwargs={'key': 'aapl'},
    )

    task2 = PythonOperator(
        task_id='googl',
        python_callable=process,
        op_kwargs={'key': 'googl'},
    )

    task3 = PythonOperator(
        task_id='meta',
        python_callable=process,
        op_kwargs={'key': 'meta'},
    )

    task4 = PythonOperator(
        task_id='msft',
        python_callable=process,
        op_kwargs={'key': 'msft'},
    )

    task5 = PythonOperator(
        task_id='amzn',
        python_callable=process,
        op_kwargs={'key': 'amzn'},
    )

    task6 = PythonOperator(
        task_id='write_csv',
        python_callable=write_to_csv,
    )

    ########################################
    # DEFINE TASKS HIERARCHY
    ########################################
    # task dependencies

    begin >> [task1, task2, task3, task4, task5]
    [task1, task2, task3, task4, task5] >> task6
```

Generated csv file so far.

```
Date,AAPL,GOOGL,META,MSFT,AMZN
2022-11-21,0.01791437061686096,0.04688281052818313,0.02186313089392508,0.034105371221516155,0.054382873412080805
2022-11-22,0.024057163909544658,0.022603908504109815,0.03116919242101665,0.010641240321213379,0.020179212629915844
2022-11-23,0.03314506402910893,0.0276264555553422,0.02307629950614651,0.005676944478117878,0.05212825431297204
2022-11-25,0.030950586401783577,0.017195168893616282,0.023942959429422272,-0.001867678741490763,0.02224018880468904
2022-11-28,0.05847457807138958,0.02382018443490338,0.02503780981637945,0.013471405180366311,0.011825165835434866
```

Firstly, task "begin" decides whether yesterday is a valid work day with trading data. If yesterday is not a work day, the global variable "work_day" would be changed to False. After task "begin" finishes, task1 to task5 would start simutaniously, calculating the prediction and the error between prediction and ground true of 5 company respectively. Each task's query stock is decided by the passing parameters. After the error of each company's stock is calculated, the error would be written into corresponding global variables. After all 5 tasks finishes, the last task would collect 5 error variables and write them into the csv file if the "work_day" is True.

To make the job run at 7 am every morning, I make the settings be schedule_interval = timedelta(days=1) and start_date = datetime(2021, 1, 1, 7, 0).



Task 3 Written parts (15 pts)

Q3.1

Answer the question (5 pts)

(1)

What are the pros and cons of SequentialExecutor, LocalExecutor, CeleryExecutor, KubernetesExecutor? (10%)

Answer:
SequentialExecutor

- Pros
  - It's simple and straightforward to set up.
  - It's a good way to test DAGs while they're being developed.
- Cons
  - It isn't scalable.
  - It is not possible to perform many tasks at the same time.
  - Unsuitable for use in production

LocalExecutor

- Pros
  - It's straightforward and easy to set up.
  - It's cheap and resource light.
  - It still offers parallelism.
- Cons
  - It's less scalable.
  - It's dependent on a single point of failure.

CeleryExecutor

- Pros
  - High availability
  - Built for horizontal scaling
  - Worker Termination Grace Period
- Cons

    - It's pricier
    - It takes some work to set up
    - Worker maintenance

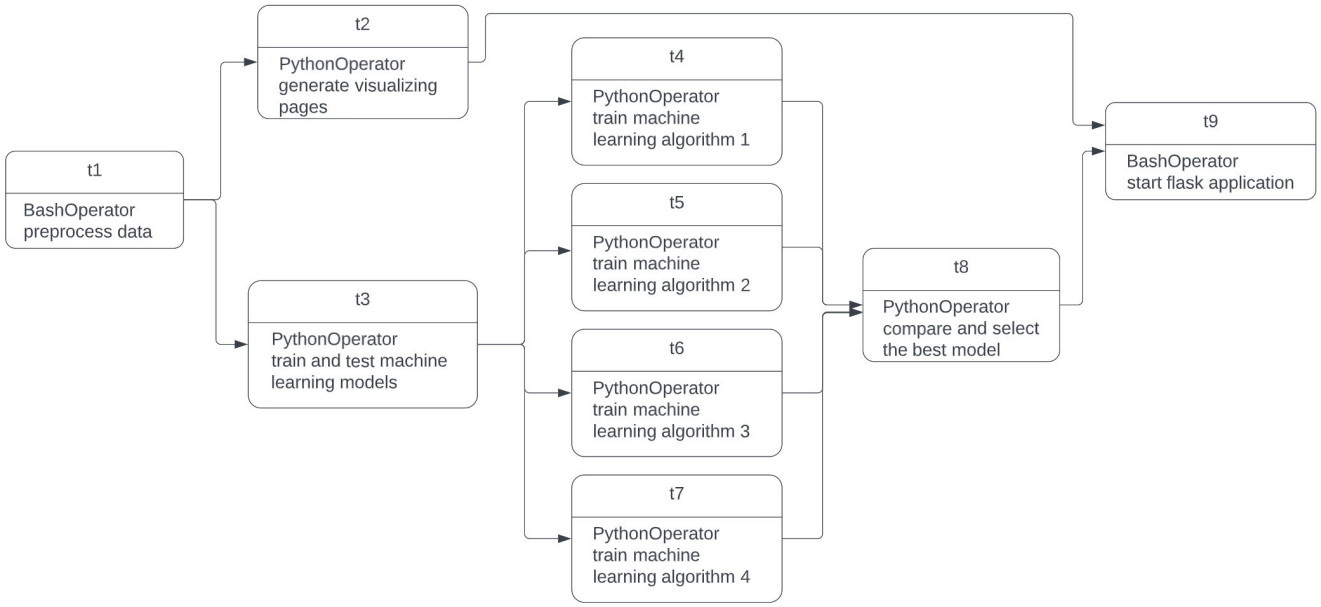
KubernetesExecutor

- Pros
    - Cost and resource efficient
    - Fault tolerant
    - Task-level configurations
    - No interruption to running tasks if a deploy is pushed
- Cons
    - Kubernetes familiarity as a potential barrier to entry
    - An overhead of a few extra seconds per task for a pod to spin up

Q3.2

Draw the DAG of your group project (10 pts)
(1) Formulate it into at least 5 tasks
(2) Task names (functions) and their dependencies
(3) How do you schedule your tasks?

Answer:



We use a fixed dataset, so the airflow job only needs to execute once. The dependencies of the tasks are shown in the picture above.