EECS-E6893 Big Data Analytics - HW4

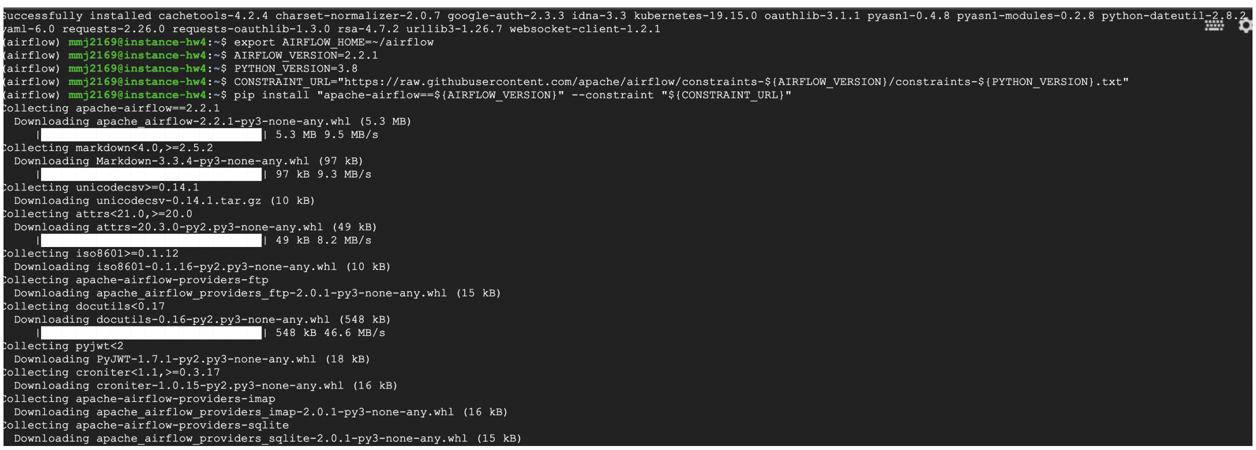
Name: Meghana Joshi

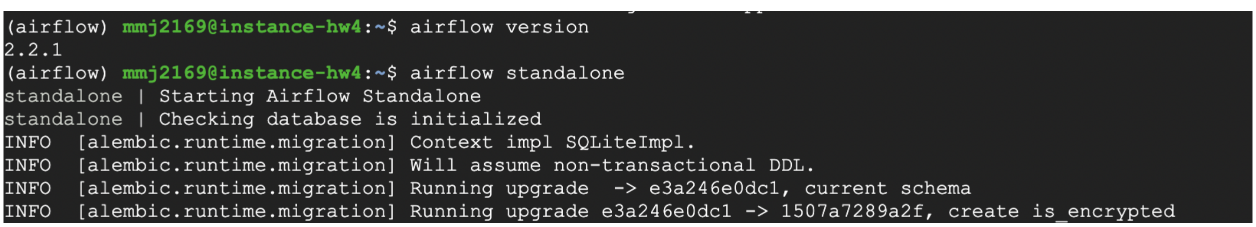
UNI: mmj2169

Task 1 : Helloworld

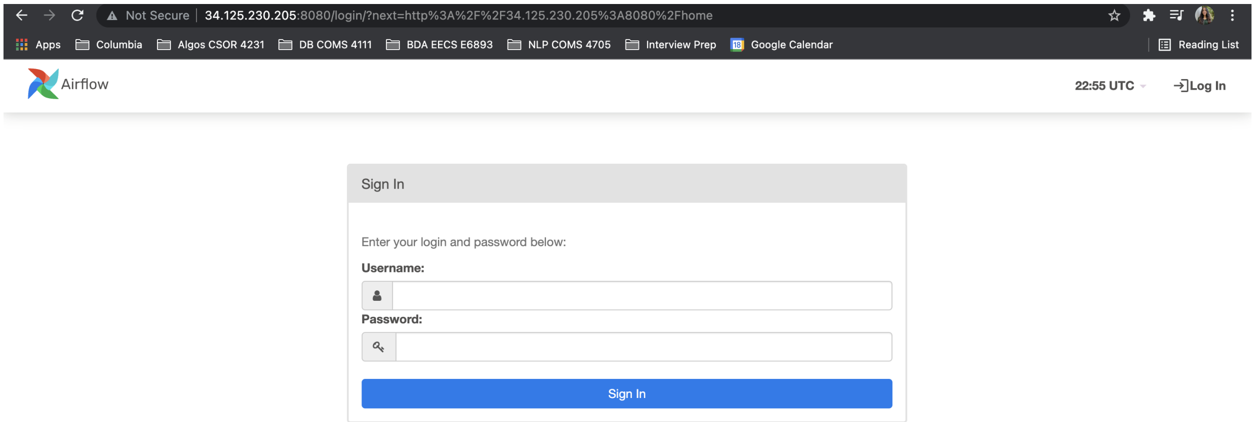
1.1) **<u>Airflow Installation</u>**

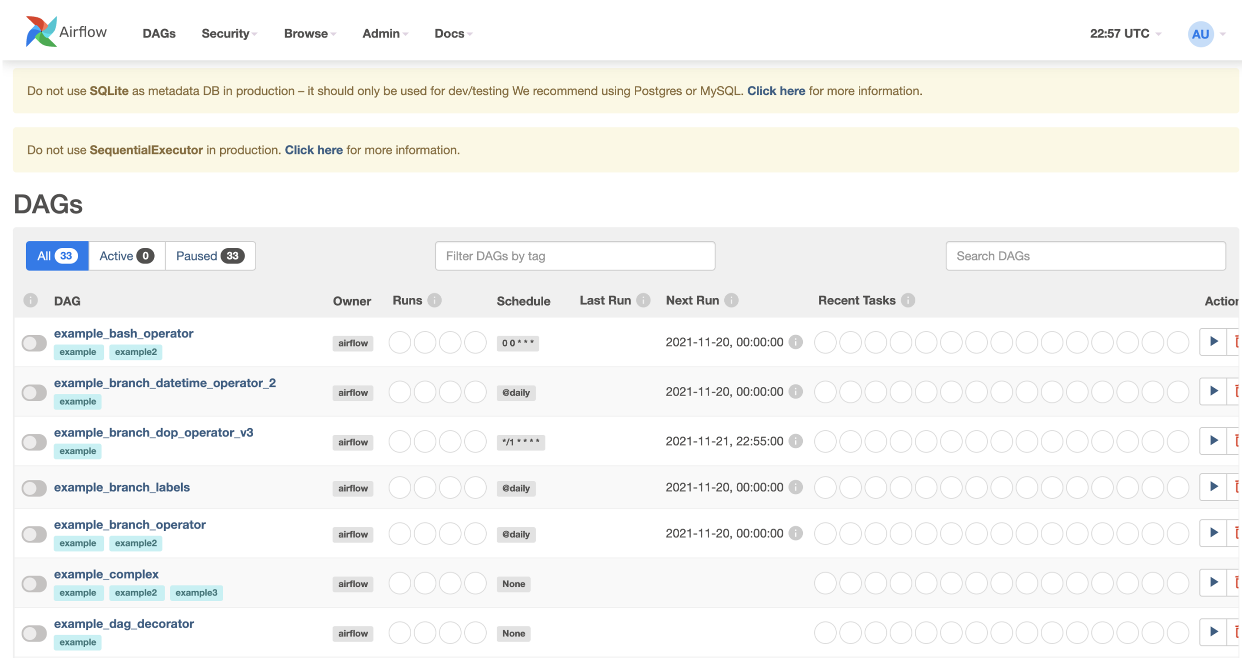
1. Screenshots of Terminal





2. Screenshots of Web Browser



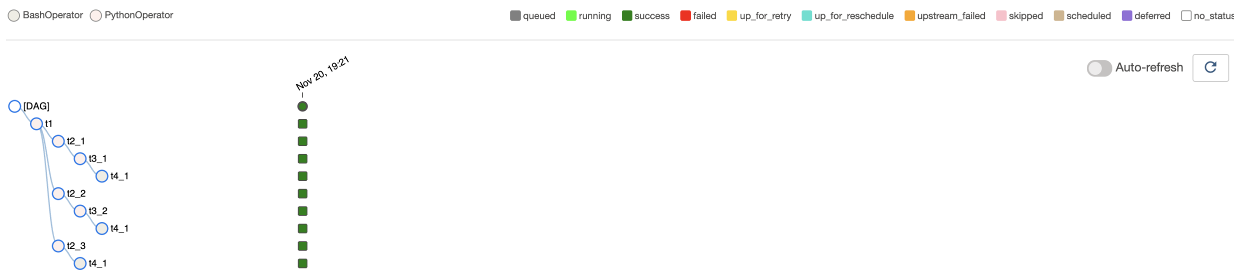


1.2) **SequentialExecutor and LocalExecutor**

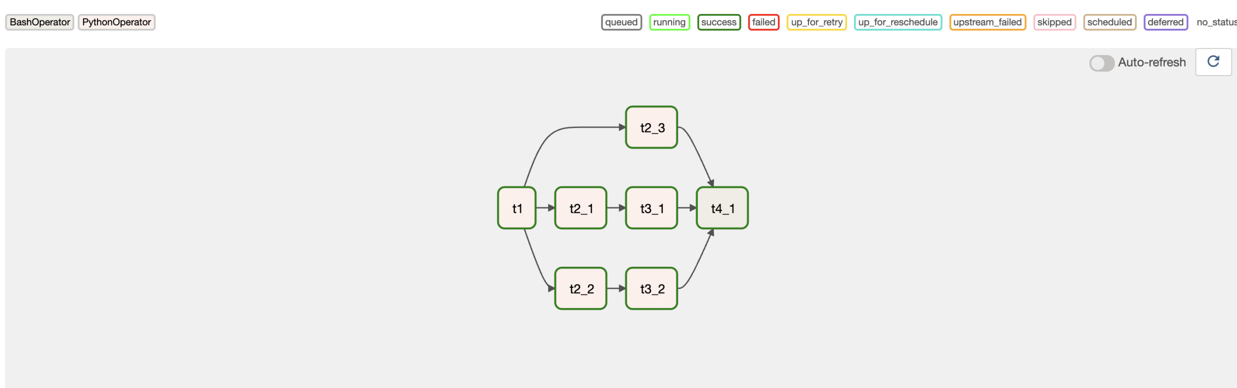
1) Tree, Graph, and Gantt of each executor

- SequentialExecutor:

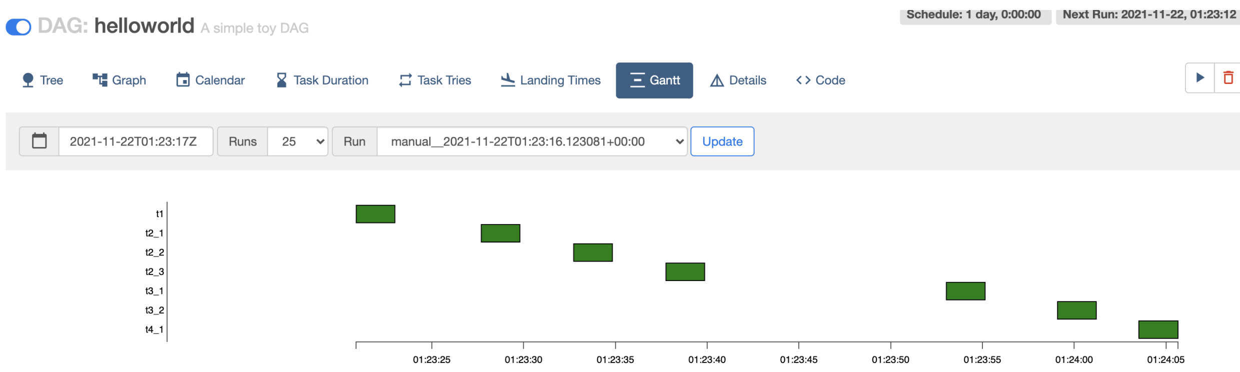
1. Tree



2. Graph

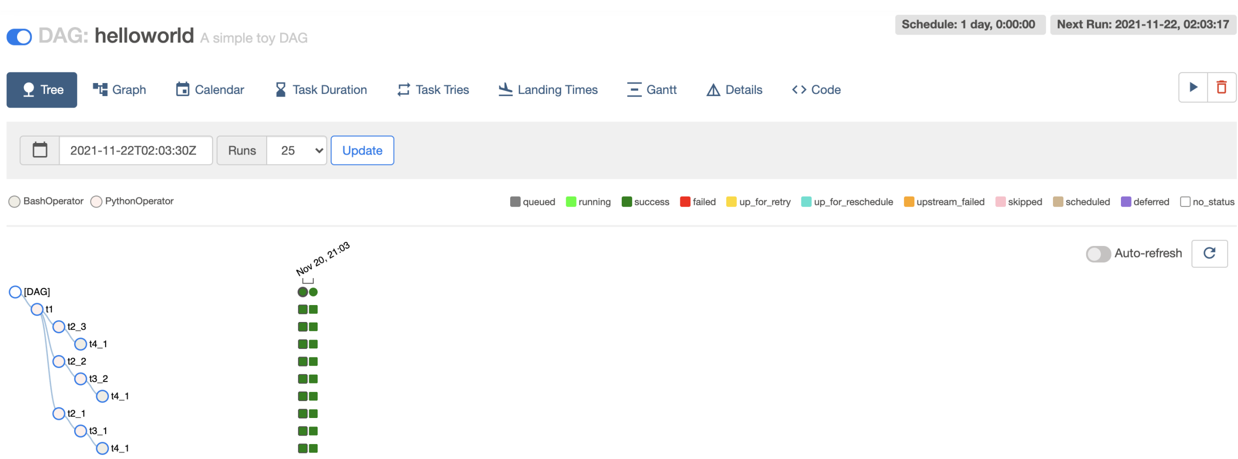


3. Gantt

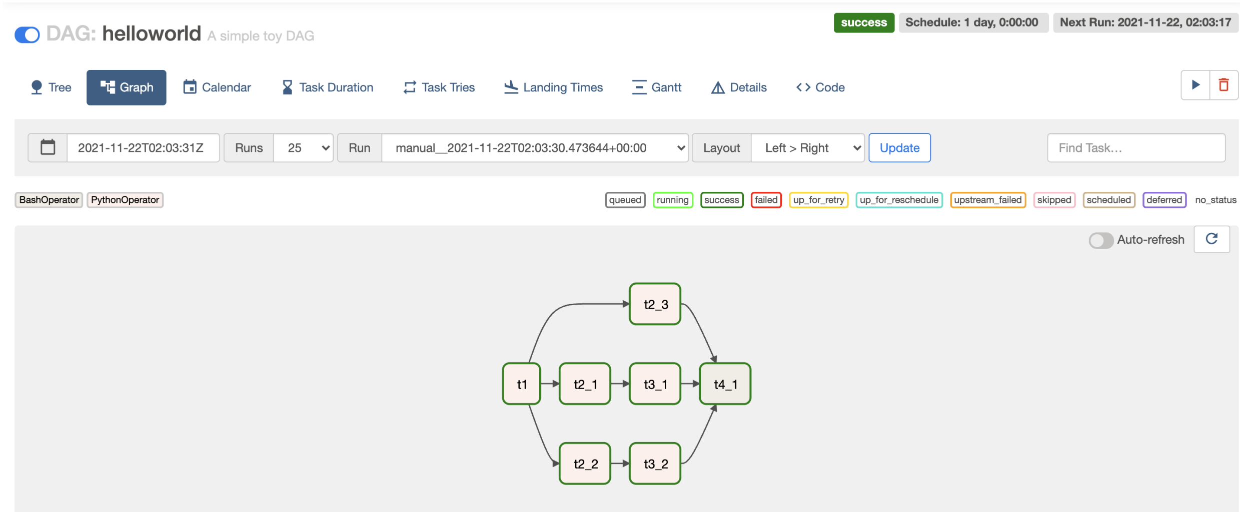


- LocalExecutor:

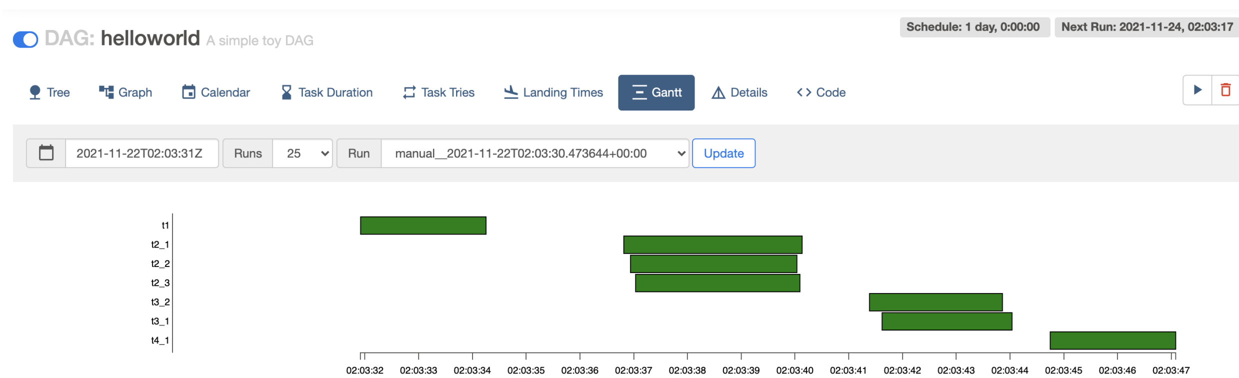
1. Tree



2. Graph



3. Gantt



2) **Additional Features/Visualizations:**

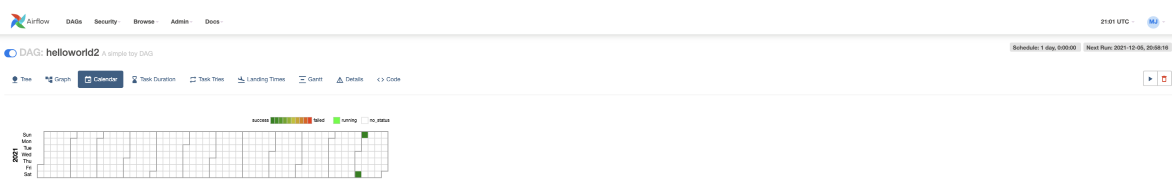
Apart from Tree, Graph and Gantt, the Airflow UI has additional features such as Calendar, Task Duration, Task Tries, and Landing Times which can be utilized to help monitor and troubleshoot the pipeline.

- Task Duration:

  The task duration shows the duration of different tasks over the past 'N' runs. This view is helpful since it allows visualization of outliers and helps to quickly identify where time is spent in the DAG over multiple runs. This information helps to further troubleshoot and identify which tasks can be combined and executed parallely.

- Calendar:

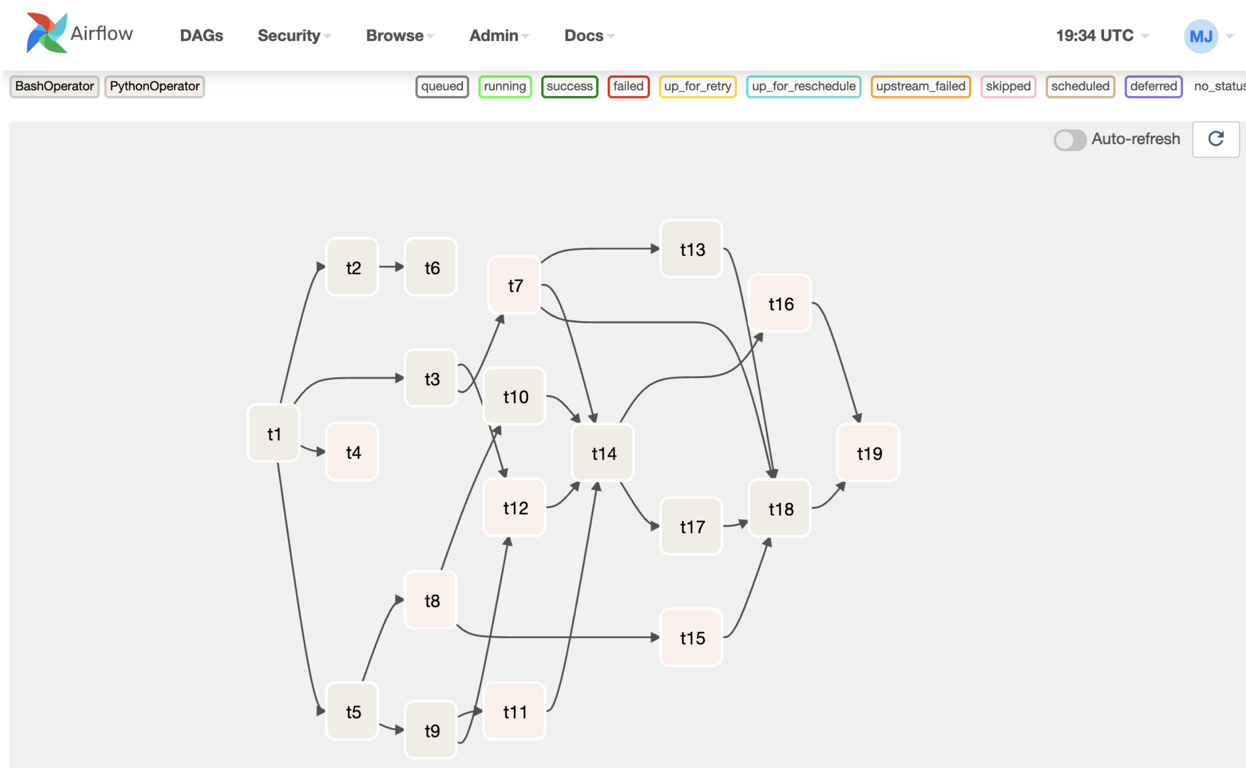
  The calendar view gives an outline of the entire DAGs history over months, or years. This allows for quick identification of the overall success/ failure rate of runs over time. This view is helpful since users can make informed decisions based on the success and failure rates of the runs.

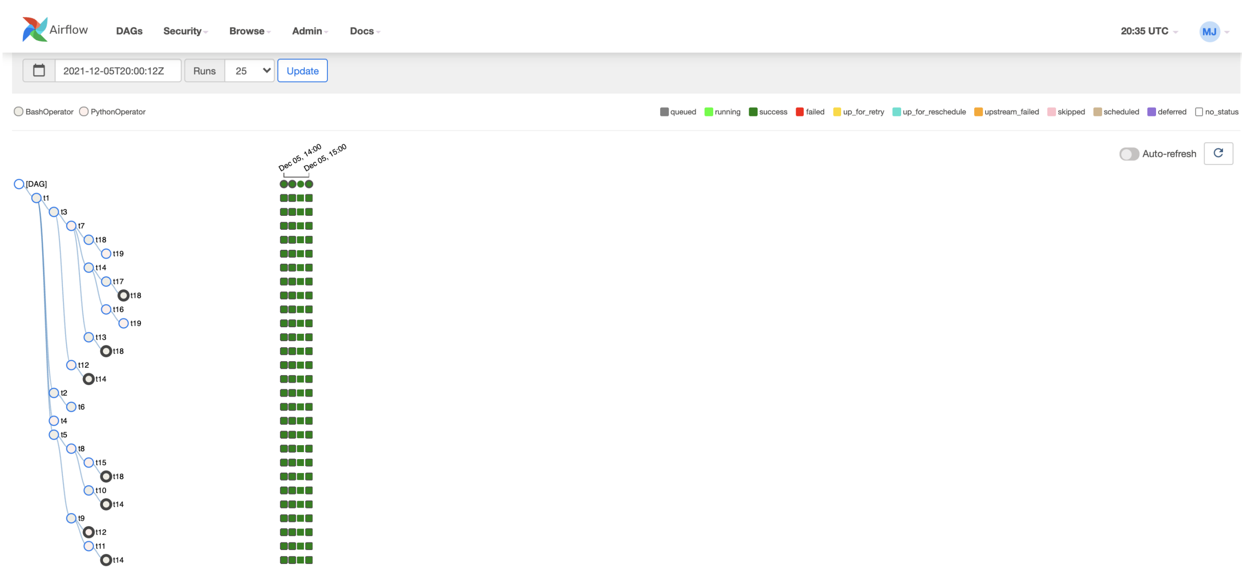


Task 2 : Building Workflows

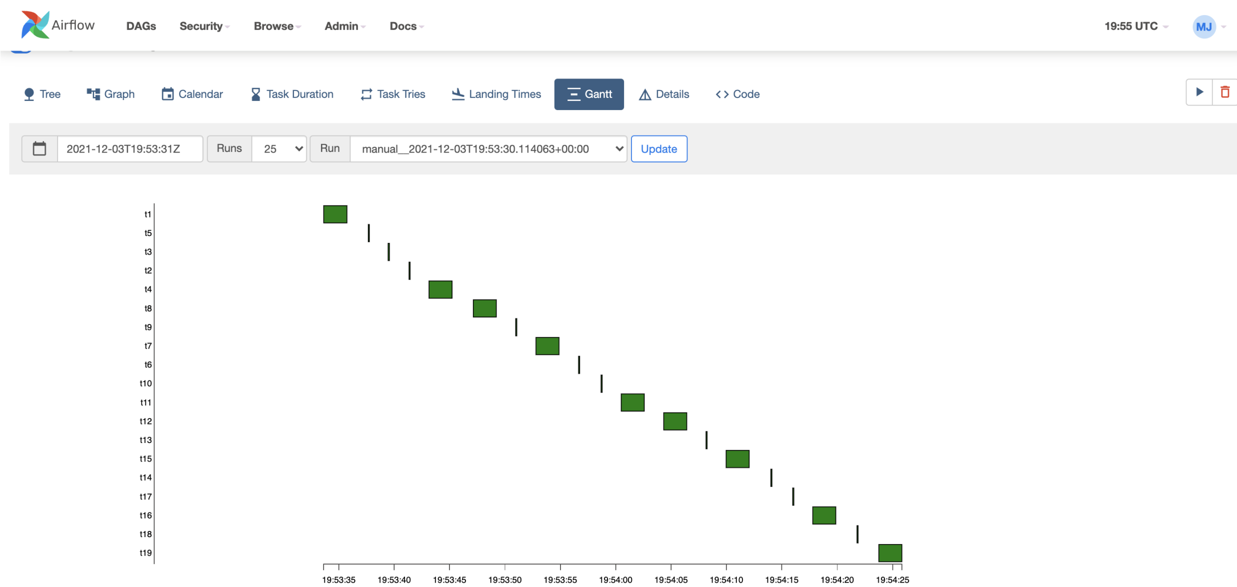
2.1) <u>**Implementing given DAG**</u>

1) Screenshots of Tree and Graph in airflow

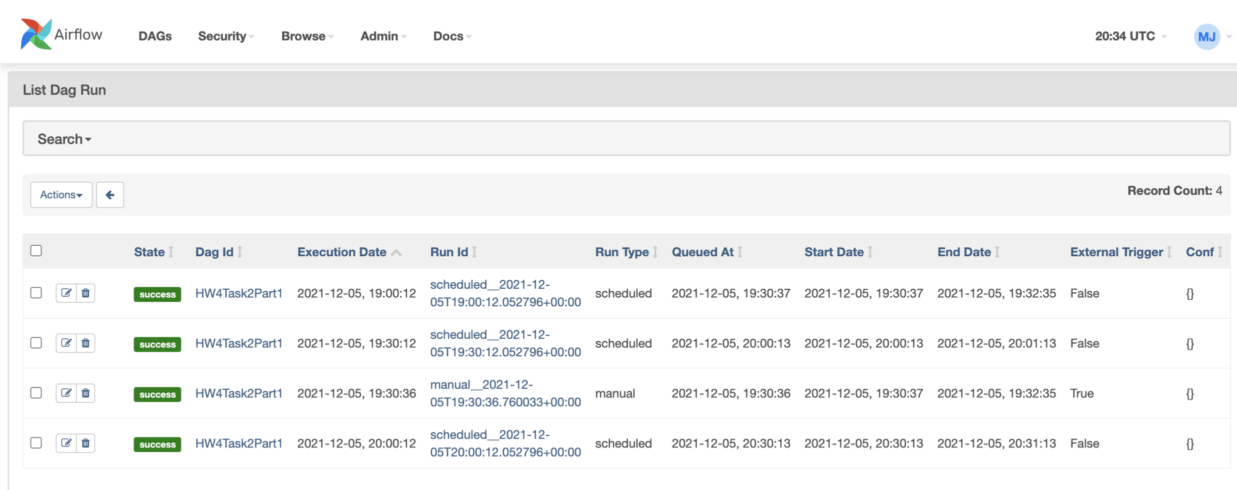




2) Screenshots of Gantt



3) Running History



State	Dag Id	Execution Date	Run Id	Run Type	Queued At	Start Date	End Date	External Trigger	Conf
success	HW4Task2Part1	2021-12-05, 19:00:12	scheduled__2021-12-05T19:00:12.052796+00:00	scheduled	2021-12-05, 19:30:37	2021-12-05, 19:30:37	2021-12-05, 19:32:35	False	{}
success	HW4Task2Part1	2021-12-05, 19:30:12	scheduled__2021-12-05T19:30:12.052796+00:00	scheduled	2021-12-05, 20:00:13	2021-12-05, 20:00:13	2021-12-05, 20:01:13	False	{}
success	HW4Task2Part1	2021-12-05, 19:30:36	manual__2021-12-05T19:30:36.760033+00:00	manual	2021-12-05, 19:30:36	2021-12-05, 19:30:37	2021-12-05, 19:32:35	True	{}
success	HW4Task2Part1	2021-12-05, 20:00:12	scheduled__2021-12-05T20:00:12.052796+00:00	scheduled	2021-12-05, 20:30:13	2021-12-05, 20:30:13	2021-12-05, 20:31:13	False	{}

2.2) <u>**Stock price fetching, prediction, and storage every day**</u>

Screenshots of Code:

- Function to Ingest the Data:

```
def data_ingestion(**kwargs):
    global tickers
    for ticker in tickers:
        end_date = kwargs['execution_date']
        df = yf.Ticker(ticker)
        stock_history = df.history(period='max', end=end_date.strftime("%Y-%m-%d"))
        print(stock_history)
        stock_history.to_pickle(f"./stock_data_{ticker}.pkl")
```

- Function to Pre-process the Data:

```
def preprocess(**kwargs):
    global tickers
    for ticker in tickers:
        stock_history = pd.read_pickle(f"./stock_data_{ticker}.pkl")
        stock_history.drop(columns=["Dividends","Stock Splits"],inplace=True)
        stock_history.reset_index(inplace=True)
        stock_history['Date'] = pd.to_datetime(stock_history.Date)
        stock_history.to_pickle(f"./stock_data_{ticker}.pkl")
```

- Function to Train the regression model:

```python
def train_model(**kwargs):
    global tickers
    for i,ticker in enumerate(tickers):
        stock_history = pd.read_pickle(f"./stock_data_{ticker}.pkl")
        end_date = kwargs['execution_date'].strftime("%Y-%m-%d")
        #print(stock_history.columns)

        if stock_history[stock_history['Date'] == end_date].shape[0] == 0:
            pass
        else:
            try:
                dfr=pd.read_csv(f'./{ticker}_error.csv')
                stock_history = stock_history.tail(11)
            except:
                stock_history = stock_history

            y = stock_history['High']
            x = stock_history[['Open', 'Low', 'Close', 'Volume']]

            train_x = x[:-1]
            train_y = y[:-1]
            test_x = x[-1:]
            test_y = y[-1:]

            #Linear Regression Model
            regression = LinearRegression()
            regression.fit(train_x, train_y)
            predicted = regression.predict(test_x)

            try:
                dfr   = pd.read_csv(f'./{ticker}_error.csv')
                dfr2  = pd.DataFrame({'Date':stock_history.Date.tail(1).values,'Actual_Price':test_y, 'Predicted_Price':predicted})
                dfr2['error'] = (dfr2["Predicted_Price"] - dfr2["Actual_Price"] )/(dfr2["Actual_Price"])
                df3 = pd.concat([dfr,dfr2])
                df3.to_csv(f'./{ticker}_error.csv',index=False)

            except:
                dfr=pd.DataFrame({'Date':stock_history.Date.tail(1).values,'Actual_Price':test_y, 'Predicted_Price':predicted})
                dfr['error'] = (dfr["Predicted_Price"] - dfr["Actual_Price"] )/(dfr["Actual_Price"])
                dfr.to_csv(f'./{ticker}_error.csv',index=False)
```

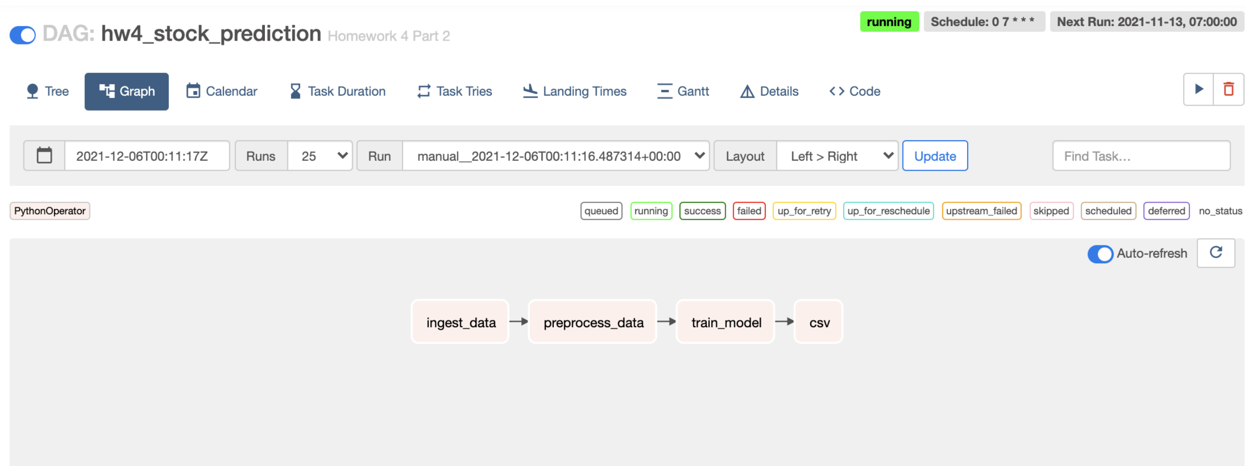
- DAG:

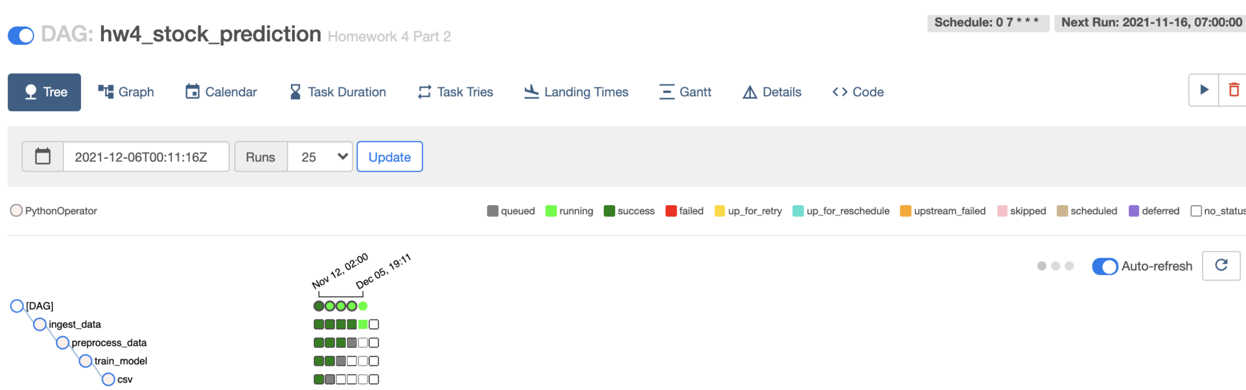
```python
with DAG(
    'hw4_stock_prediction',
    default_args=default_args,
    description='Homework 4 Part 2',
    schedule_interval='0 7 * * *',
    start_date=datetime(2021, 11, 12),
    end_date=datetime(2021, 11, 28),
    catchup=True,
    tags=['hw4'],
) as dag:


    task_ingest_data = PythonOperator(
        task_id='ingest_data',
        python_callable=data_ingestion,
        provide_context=True,
        dag=dag
    )

    task_preprocess_data = PythonOperator(
        task_id='preprocess_data',
        python_callable=preprocess,
        provide_context=True,
        dag=dag
    )

    task_train_model = PythonOperator(
        task_id='train_model',
        python_callable=train_model,
        provide_context=True,
        dag=dag
    )

    task_csv = PythonOperator(
        task_id='csv',
        python_callable=create_csv,
        provide_context=True,
        dag=dag
    )
```

- Task Dependencies:

```python
task_ingest_data >> task_preprocess_data >> task_train_model >> task_csv
```

Screenshots of DAG:





Output:

```
Schedule               Company  Actual_Price        Predicted_Price     error
2021-11-12 07:00:00 am AAPL     150.39999389648438  150.75210229367272  0.0023411463529093
2021-11-17 07:00:00 am AAPL     155.0               153.77794070807477  -0.0078842534962918
2021-11-19 07:00:00 am AAPL     161.02000427246094  159.97066898901772  -0.0065168007427675
2021-11-24 07:00:00 am AAPL     162.13999938964844  161.59674417833995  -0.0033505317216818
2021-11-12 07:00:00 am GOOGL    2977.0              2987.4074664654136  0.0034959578318486
2021-11-17 07:00:00 am GOOGL    2971.18994140625    2980.5896051683776  0.0031636024446416
2021-11-19 07:00:00 am GOOGL    3019.330078125      3012.3966934658592  -0.0022963321265776
2021-11-24 07:00:00 am GOOGL    2924.989990234375   2931.9043979936337  0.0023639081782651
2021-11-12 07:00:00 am FB       341.8599853515625   340.1429201646225   -0.005022714738533
2021-11-17 07:00:00 am FB       347.29998779296875  346.1638211612027   -0.0032714272148012
2021-11-19 07:00:00 am FB       352.1000061035156   349.78218837520217  -0.0065828392165151
2021-11-24 07:00:00 am FB       341.7799987792969   339.9625795379698   -0.005317511989637
2021-11-12 07:00:00 am MSFT     336.6141809542142   336.8024820629323   0.0005593974329433
2021-11-17 07:00:00 am MSFT     342.19000244140625  340.93658534287744  -0.0036629272906459
2021-11-19 07:00:00 am MSFT     345.1000061035156   346.844079138496    0.0050538191948255
2021-11-24 07:00:00 am MSFT     338.1600036621094   338.98149892141305  0.0024293093518076
2021-11-12 07:00:00 am AMZN     3540.72998046875    3549.747312160178   0.0025467436774814
2021-11-17 07:00:00 am AMZN     3587.25             3604.277399413689   0.004746644202018
2021-11-19 07:00:00 am AMZN     3762.14990234375    3780.8251196050815  0.0049639747873143
2021-11-24 07:00:00 am AMZN     3613.639892578125   3582.110083406826   -0.0087252216902011
```

Explanation:

In this DAG, I used four PythonOperators to run four Python functions. Each Python operator is a separate task that needs to be passed to *Airflow Sequential Executor* in order to achieve the expected result.

The four Python functions are:

1. A function to ingest the data
2. A function to preprocess the data
3. A function to train the model
4. A function to write errors into a csv file

A static start_date and end_date are specified in the code, with catchup = True to allow airflow to backfill the data. Additionally, schedule interval is set to * 7 * * * to schedule the task at 7am daily.

Task 3 : Written Parts

3.1) 1) What are the pros and cons of SequentialExecutor, LocalExecutor, CeleryExecutor,
KubernetesExecutor?

- SequentialExecutor: The Sequential Executor runs a single task instance at a time in a linear fashion with no parallelism functionality (A → B → C).
    - Pros:
        - The SequentialExecutor has the ability to identify a single point of failure, thus making it useful for debugging purposes.
        - It is simple and easy to setup.
    - Cons:

        - Sequential Executor is not recommended for any use cases that require more than a single task execution at a time.
        - It is not scalable.
        - Not suitable to be used in production.

- LocalExecutor: The LocalExecutor is similar to the Sequential Executor, except for the fact that it can run multiple tasks at a time.
    - Pros:
        - The LocalExecutor can run multiple tasks at a time.
        - It is good for running DAGs during development.
    - Cons:
        - It is not scalable.
        - Identifying a single point of failure can be difficult.
        - The LocalExecutor is not suitable for production.

- CeleryExecutor: It is often used for running distributed asynchronous python tasks. The CeleryExecutors has a fixed number of workers running to pick-up the tasks as they are scheduled.
    - Pros:
        - It provides scalability.
        - Since celery manages its workers, it can easily spawn new ones in case of a failure.
    - Cons:
        - In order to queue the task, celery requires RabbitMQ/ Redis, which is a duplication of effort since airflow already supports this.
        - The CeleryExecutor is comparatively complex.

- KubernetesExecutor: The KubernetesExecutor runs each task in an individual Kubernetes pod. Since it spawns worker pods on demand, it enables maximum usage of resources.
    - Pros:
        - It is a combination of the advantages of scalability and simplicity of CeleryExecutor and LocalExecutor.
        - There is fine-grained control over the resources allocated to tasks.
    - Cons:
        - As it is new airflow, there is a lock of proper documentation, which may cause the setup to be rather complicated.

3.2) DAG of Group project

1) <u>Tasks:</u>

1. Data Extraction - From StackExchange Data Dump
2. Data Cleaning - Fixing missing attribute fields, cleaning up text field, etc.
3. Data Pre-Processing
4. Feature Extraction - Extracting features from the text of the post body such as Metric Entropy, Average Term Entropy
5. Data Visualization - Visualization using Correlation heatmaps, jount plots and word clouds
6. LSTM Models - Models used for classification of a post as good quality or poor quality
7. Prediction Models - It is used to predict the target feature, i.e, quality
8. F1 Score - Performance metric is calculated
9. Kappa Cohen Metric - Calculation of metric used for inter-rater reliability
10. Mathews Correlation Coefficient - Calculation of a metric that produces a high score only if the prediction obtained good results in all of the four confusion matrix categories

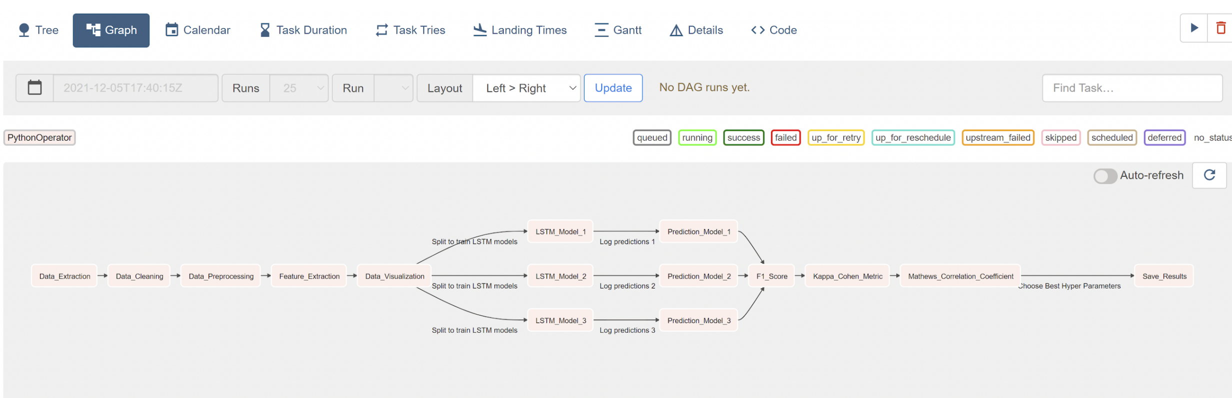
11. Save reults - Saving results which can be used for dynamic prediction

2) Task names (functions) and their dependencies:

Task Names:

1. Data Extraction
2. Data Cleaning
3. Data Pre-Processing
4. Feature Extraction
5. Data Visualization
6. LSTM Models
7. Prediction Models
8. F1 Score
9. Kappa Cohen Metric
10. Mathews Correlation Coefficient
11. Save reults

Dependencies:



3) Scheduling of Tasks:

The tasks are scheduled sequentially since the tasks need to be completed one after the other. For example, the model cannot be trained before the data is cleaned and pre-processed. However, the LSTM Models and Predictions can be trained and run in parallel, and later sequential execution continues.

Final DAG: