Project: Creating and Automating a Set of Data Pipelines with Airflow

สิ่งที่คาดหวังในโปรเจคนี้

1. นำโค้ดจาก [Project: Building a Data Modeling with Postgres (SQL)]() มาสร้าง data pipeline โดยใช้ Airflow
2. มีการเขียน documentation อธิบายสิ่งที่ตัวเองทำลงไป รวมไปถึงการออกแบบ data model
3. มี instruction ในการรันโค้ดของตัวเอง

gitpod /workspace/swu-ds525/05-creating-and-scheduling-data-pipelines (main) $ docker-compose up

วิธีทำ Over All

① import DAG, time-zone

② สร้าง Data pipeline ชื่อ "MyDag"

③ save file python

④ กด ก็จะขึ้นที่ Airflow รอสักพักจะมีขึ้นที่ Airflow

t1 >> t2

⑤ bash operator, python operator

⑥ grid view

อันบนคือ บนล่าง

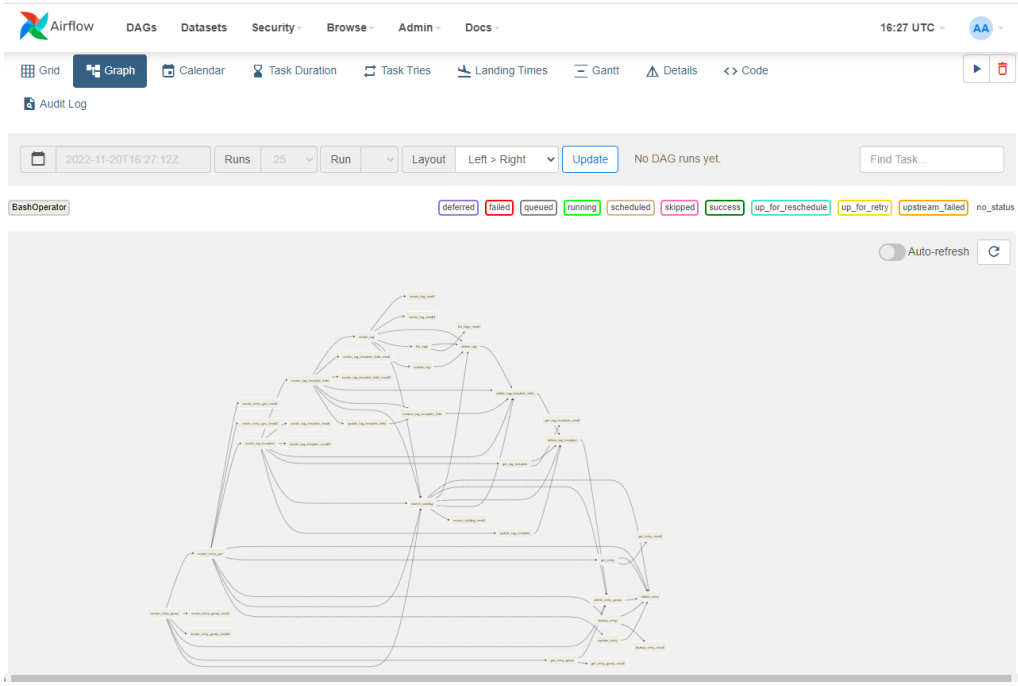
triger 1 ที่เกิด 1 dag run

clear task

t1 → details → clear → confirm

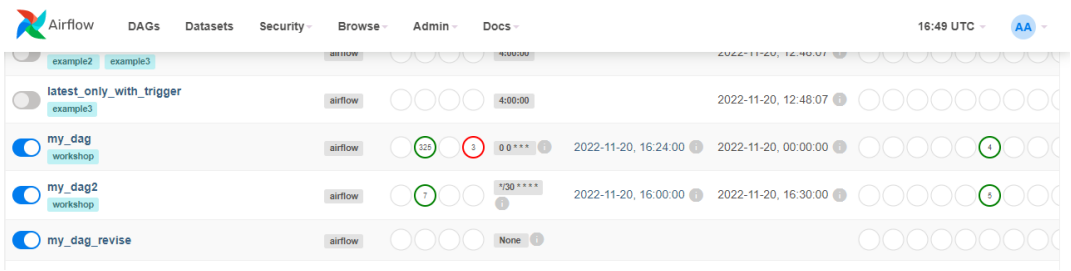
จะ run ใหม่อีกรอบ

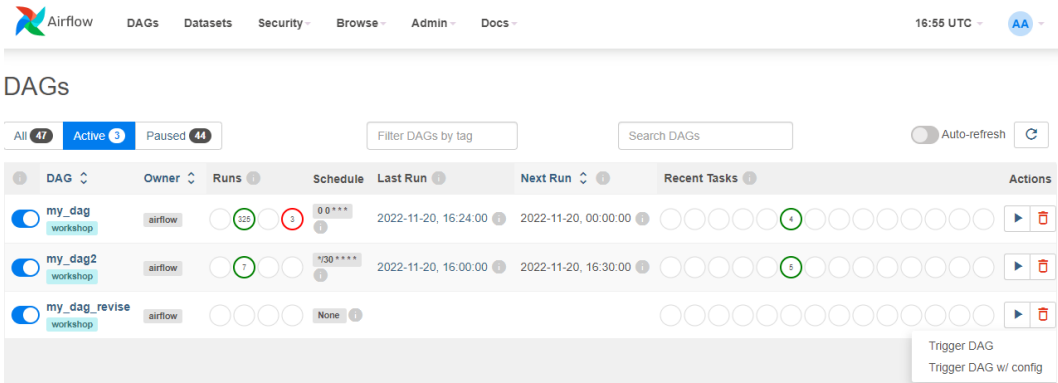
cronitor → cron expression เพื่อจะ set ให้ run ตอนไหน - เมื่อไหร่

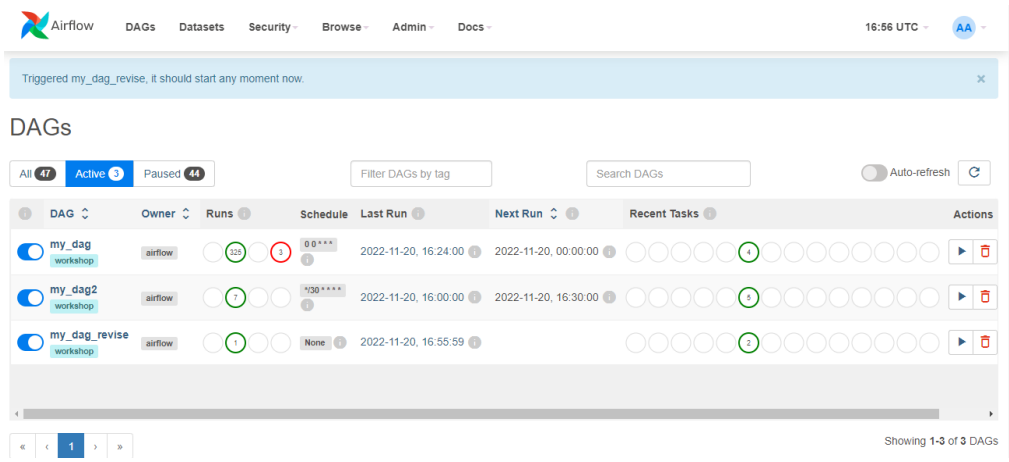


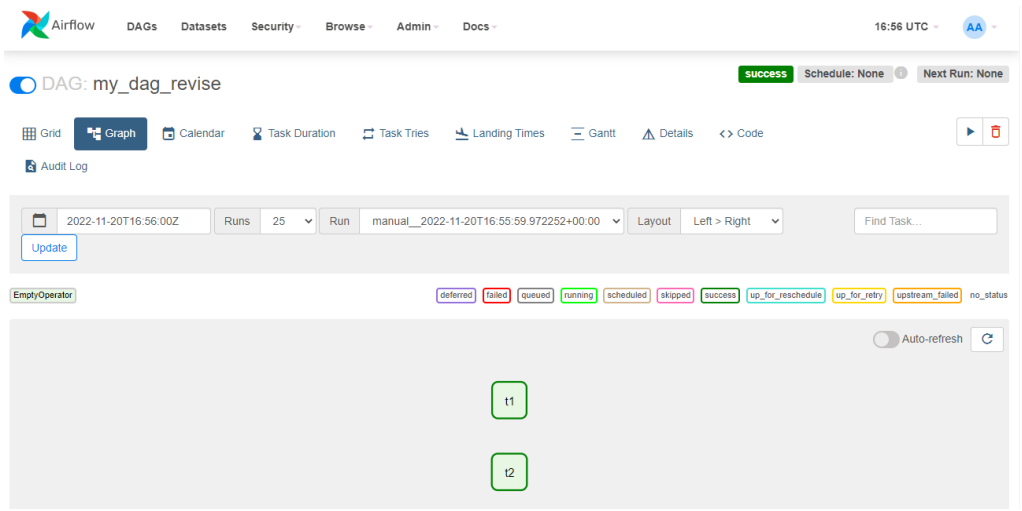
05-creating-and-scheduling-data-pipelines > dags > my_dag_revise.py

```python
#import DAG เป็น pipeline
from airflow import DAG
#import Timezone
from airflow.utils import timezone
#step 2 ใช้ EmptyOperator เลย import EmptyOperator เข้ามา
from airflow.operators.empty import EmptyOperator


#context manager เป็นการประกาศหัว
#"my_dag" ชื่อเดียวกับชื่อ file
#start date 2022, 10, 8
# schedule = None ยังไม่schedule
# step10
# schedule
# schedule เที่ยงคืน คือ 0
#v1  schedule = None ยังไม่set schedule
with DAG(
    "my_dag",
    start_date = timezone.datetime(2022, 10, 8),
    schedule = None,
):
   t1 = EmptyOperator( task_id = "t1")
   t2 = EmptyOperator( task_id = "t2")
```


Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
16:49 UTC
AA
example2
example3
latest_only_with_trigger
example3
airflow
4:00:00
2022-11-20, 12:48:07
my_dag
workshop
airflow
0 0 * * *
2022-11-20, 16:24:00
2022-11-20, 00:00:00
my_dag2
workshop
airflow
*/30 * * * *
2022-11-20, 16:00:00
2022-11-20, 16:30:00
my_dag_revise
airflow
None

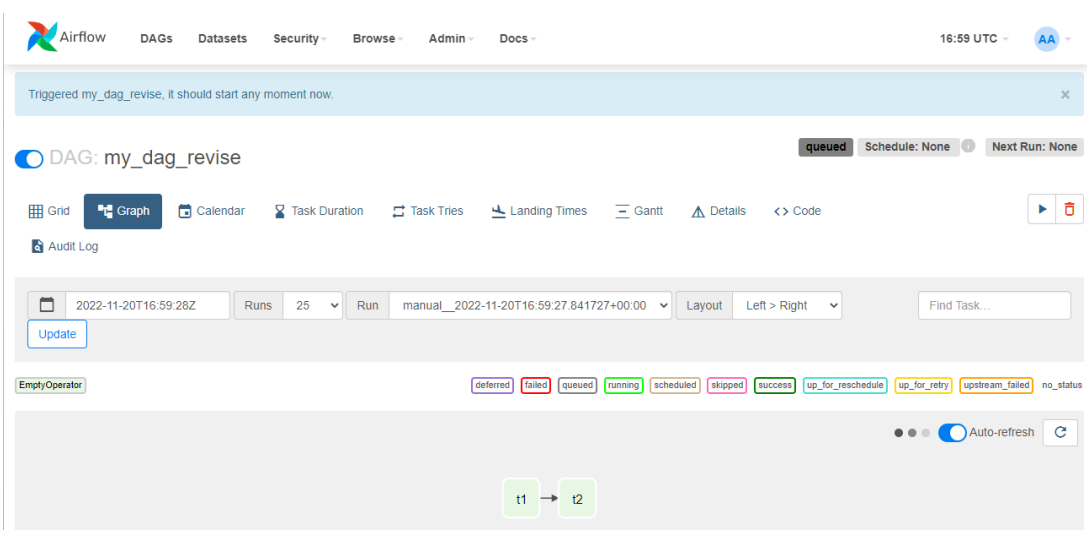

Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
16:55 UTC
AA
DAGs
All 47
Active 3
Paused 44
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
Actions
my_dag
workshop
airflow
0 0 * * *
2022-11-20, 16:24:00
2022-11-20, 00:00:00
my_dag2
workshop
airflow
*/30 * * * *
2022-11-20, 16:00:00
2022-11-20, 16:30:00
my_dag_revise
workshop
airflow
None
Trigger DAG
Trigger DAG w/ config


Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
16:56 UTC
AA
Triggered my_dag_revise, it should start any moment now.
DAGs
All 47
Active 3
Paused 44
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
Actions
my_dag
workshop
airflow
0 0 * * *
2022-11-20, 16:24:00
2022-11-20, 00:00:00
my_dag2
workshop
airflow
*/30 * * * *
2022-11-20, 16:00:00
2022-11-20, 16:30:00
my_dag_revise
workshop
airflow
None
2022-11-20, 16:55:59
Showing 1-3 of 3 DAGs


Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
16:56 UTC
AA
DAG: my_dag_revise
success
Schedule: None
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
2022-11-20T16:56:00Z
Runs
25
Run
manual__2022-11-20T16:55:59.972252+00:00
Layout
Left > Right
Find Task...
Update
EmptyOperator
deferred
failed
queued
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
t1
t2

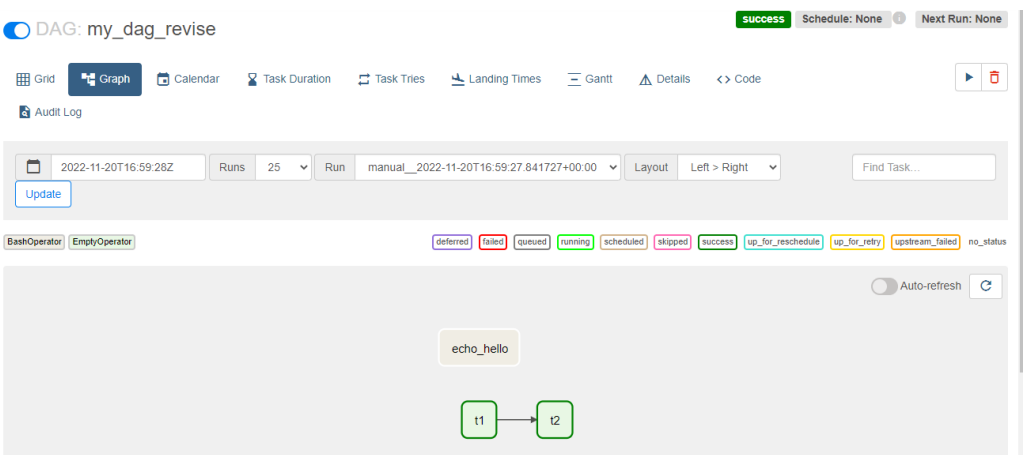
05-creating-and-scheduling-data-pipelines > dags > my_dag_revise.py

```python
#import DAG เป็น pipeline
from airflow import DAG
#import Timezone
from airflow.utils import timezone
#step 2 ใช้ EmptyOperator เลย import EmptyOperator เข้ามา
from airflow.operators.empty import EmptyOperator


#context manager เป็นการประกาศหัว
#"my_dag" ชื่อเดียวกับชื่อ file
#start date 2022, 10, 8
# schedule = None ยังไม่schedule
# step10
# schedule
# schedule เที่ยงคืน คือ 0
# tags =['workshop'] จะทำให้เป็นชัดเจน
#v1  schedule = None ยังไม่set schedule
with DAG(
    "my_dag_revise",
    start_date = timezone.datetime(2022, 10, 8),
    schedule = None,
    tags =["workshop"],
):
    t1 = EmptyOperator( task_id = "t1")
    t2 = EmptyOperator( task_id = "t2")

#t1 รันก่อน t2
    t1 >> t2

```



05-creating-and-scheduling-data-pipelines > dags > my_dag_revise.py

```python
# ):
#     t1 = EmptyOperator( task_id = "t1")
#     t2 = EmptyOperator( task_id = "t2")

# #t1 รันก่อน t2
#     t1 >> t2

#v2
with DAG(
    "my_dag_revise",
    start_date = timezone.datetime(2022, 10, 8),
    schedule = None,
    tags =["workshop"],
):
    t1 = EmptyOperator( task_id = "t1")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command= "echo 'hello'",
    )

    t2 = EmptyOperator( task_id = "t2")

#t1 รันก่อน t2
    t1 >> t2
```




```python
#t1 รันก่อน t2
    t1 >> echo_hello >> t2
```

DAG: my_dag_revise

Schedule: None | Next Run: None

Grid | Graph | Calendar | Task Duration | Task Tries | Landing Times | Gantt | Details | Code | Audit Log

11/20/2022 05:12:27 PM | 25 | All Run Types | All Run States | Clear Filters

deferred | failed | queued | running | scheduled | skipped | success | up_for_reschedule | up_for_retry | upstream_failed | no_status

Auto-refresh

Duration
00:00:03
00:00:01
00:00:00

t1
echo_hello
t2

DAG
my_dag_revise

DAG Details

DAG Runs Summary	
Total Runs Displayed	3
Total success	3
First Run Start	2022-11-20, 16:56:00 UTC
Last Run Start	2022-11-20, 17:12:19 UTC
Max Run Duration	00:00:03
Mean Run Duration	00:00:01
Min Run Duration	00:00:00
DAG Summary	
Total Tasks	3
EmptyOperators	2
BashOperator	1

Airflow | DAGs | Datasets | Security | Browse | Admin | Docs | 17:14 UTC | AA

Triggered my_dag_revise, it should start any moment now.

DAG: my_dag_revise

Schedule: None | Next Run: None

Grid | Graph | Calendar | Task Duration | Task Tries | Landing Times | Gantt | Details | Code | Audit Log

11/20/2022 05:14:10 PM | 25 | All Run Types | All Run States | Clear Filters

deferred | failed | queued | running | scheduled | skipped | success | up_for_reschedule | up_for_retry | upstream_failed | no_status

Auto-refresh

Duration
Nov 20, 16:55
00:00:03
00:00:01
00:00:00

t1
echo_hello
t2

DAG my_dag_revise / Run 2022-11-20, 17:12:18 UTC / Task echo_hello

Task Instance Details | Rendered Template | Log | XCom | List Instances, all runs | Filter Upstream

Details | Logs

Task Actions

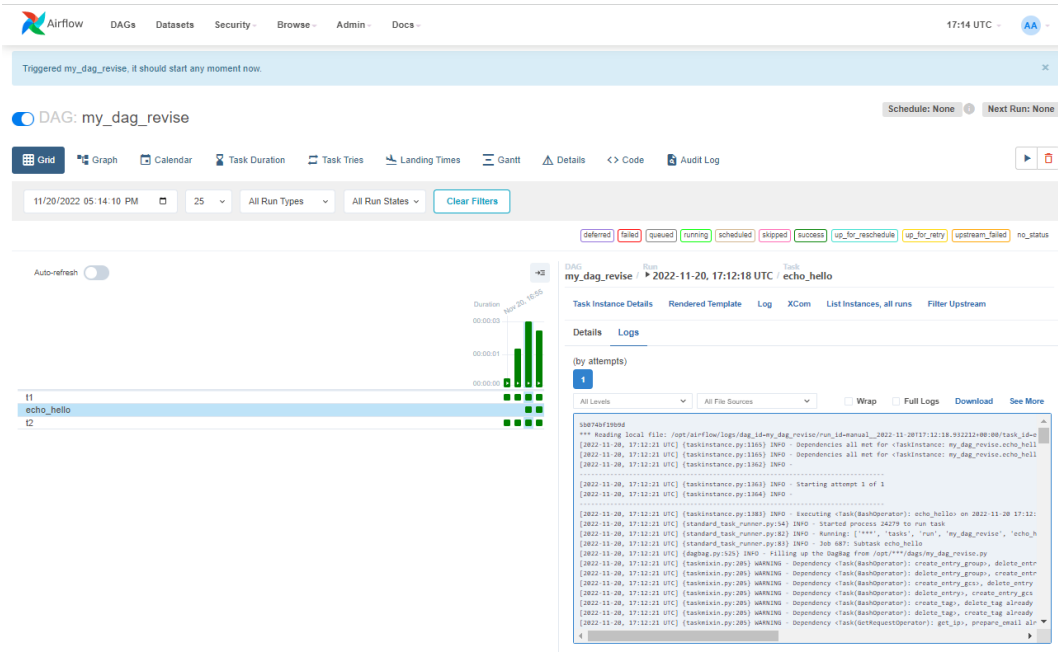
Ignore All Deps | Ignore Task State | Ignore Task Deps | Run

Past | Future | Upstream | Downstream | Recursive | Failed | Clear

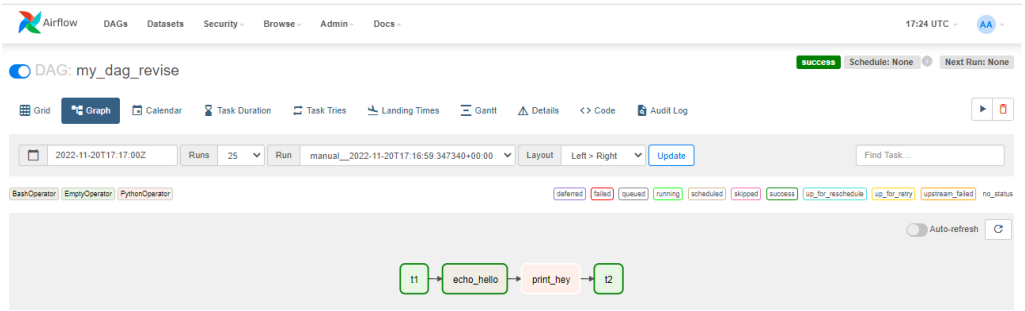
Past | Future | Upstream | Downstream | Mark Failed

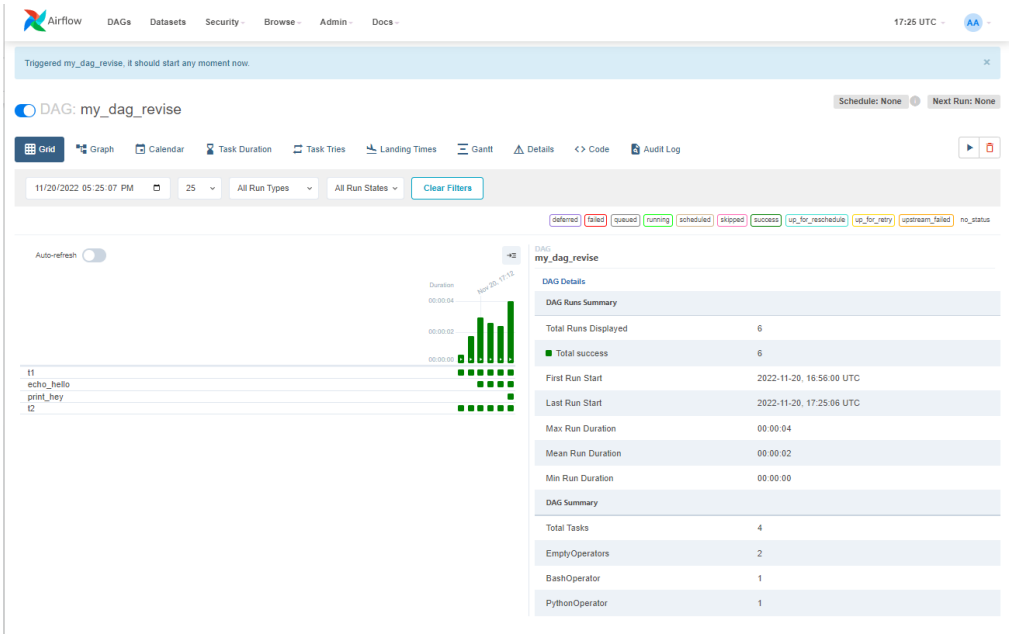
Past | Future | Upstream | Downstream | Mark Success

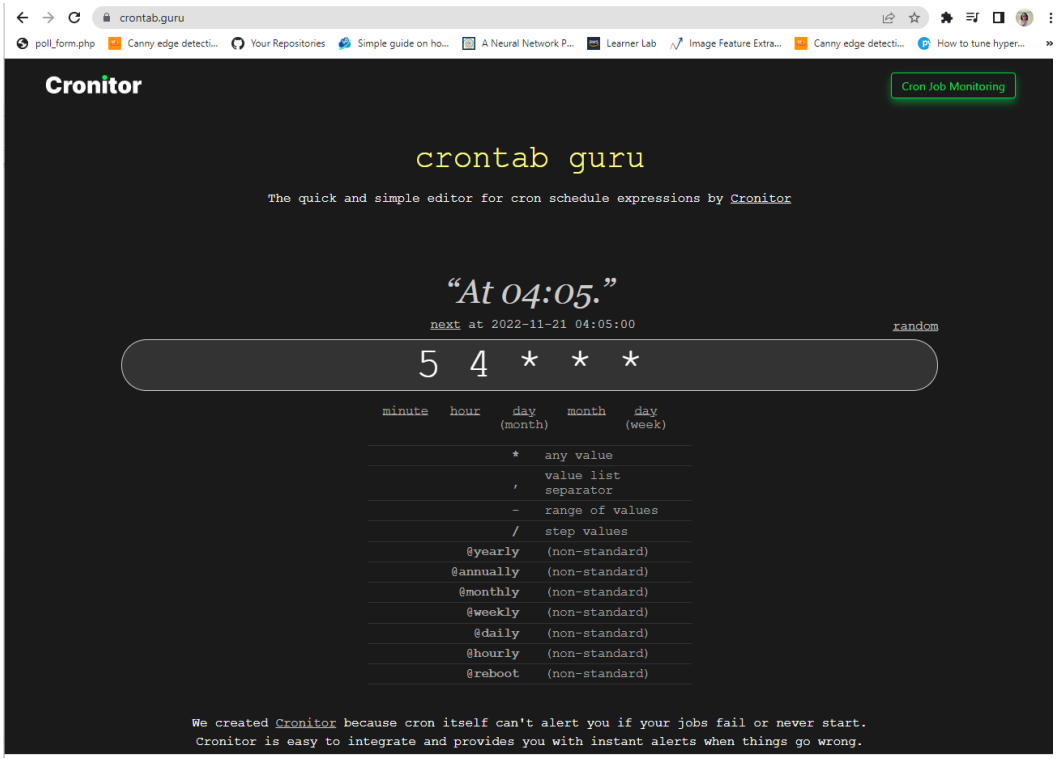
Status	success
Task ID	echo_hello
Run ID	manual__2022-11-20T17:12:18.932212+00:00
Operator	BashOperator
Duration	00:00:00
Started	2022-11-20, 17:12:21 UTC
Ended	2022-11-20, 17:12:22 UTC




```python
#v3
with DAG(
    "my_dag_revise",
    start_date = timezone.datetime(2022, 10, 8),
    schedule = None,
    tags =["workshop"],
):
    t1 = EmptyOperator( task_id = "t1")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command= "echo 'hello'",
    )
    def _print_hey():
        print("Hey!")

    print_hey = PythonOperator(
        task_id = "print_hey",
        python_callable = _print_hey,
    )

    t2 = EmptyOperator( task_id = "t2")

#t1 รันก่อน t2
    t1 >> echo_hello >> print_hey >> t2
```




Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
17:25 UTC
Triggered my_dag_revise, it should start any moment now.
DAG: my_dag_revise
Schedule: None
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
11/20/2022 05:25:07 PM
25
All Run Types
All Run States
Clear Filters
deferred
failed
queued
running
scheduled
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
Duration
00:00:04
00:00:02
00:00:00
t1
echo_hello
print_hey
t2
DAG
my_dag_revise
DAG Details
DAG Runs Summary
Total Runs Displayed 6
Total success 6
First Run Start 2022-11-20, 16:56:00 UTC
Last Run Start 2022-11-20, 17:25:06 UTC
Max Run Duration 00:00:04
Mean Run Duration 00:00:02
Min Run Duration 00:00:00
DAG Summary
Total Tasks 4
EmptyOperators 2
BashOperator 1
PythonOperator 1


crontab.guru
poll_form.php
Canny edge detecti...
Your Repositories
Simple guide on ho...
A Neural Network P...
Learner Lab
Image Feature Extra...
Canny edge detecti...
How to tune hyper...
Cronitor
Cron Job Monitoring
crontab guru
The quick and simple editor for cron schedule expressions by Cronitor
"At 04:05."
next at 2022-11-21 04:05:00
random
5 4 * * *
minute hour day (month) month day (week)
* any value
, value list separator
- range of values
/ step values
@yearly (non-standard)
@annually (non-standard)
@monthly (non-standard)
@weekly (non-standard)
@daily (non-standard)
@hourly (non-standard)
@reboot (non-standard)
We created Cronitor because cron itself can't alert you if your jobs fail or never start.
Cronitor is easy to integrate and provides you with instant alerts when things go wrong.

crontab.guru/#*_*_*_*_*

poll_form.php | Canny edge detecti... | Your Repositories | Simple guide on ho... | A Neural Network P... | Learner Lab | Image Feature Extra... | Canny edge detecti... | How to tune hyper...

Cronitor Cron Job Monitoring

crontab guru

The quick and simple editor for cron schedule expressions by Cronitor

"At every minute."

next at 2022-11-21 00:29:00 random

```
*  *  *  *  *
```

minute	hour	day (month)	month	day (week)

*	any value
,	value list separator
-	range of values
/	step values
@yearly	(non-standard)
@annually	(non-standard)
@monthly	(non-standard)
@weekly	(non-standard)
@daily	(non-standard)
@hourly	(non-standard)
@reboot	(non-standard)

We created Cronitor because cron itself can't alert you if your jobs fail or never start.
Cronitor is easy to integrate and provides you with instant alerts when things go wrong.

crontab.guru/#30_8_*_*_*

poll_form.php | Canny edge detecti... | Your Repositories | Simple guide on ho... | A Neural Network P... | Learner Lab | Image Feature Extra... | Canny edge detecti... | How to tune hyper...

Cronitor Cron Job Monitoring

crontab guru

The quick and simple editor for cron schedule expressions by Cronitor

"At 08:30."

next at 2022-11-21 08:30:00 random

```
30  8  *  *  *
```

minute	hour	day (month)	month	day (week)

*	any value
,	value list separator
-	range of values
/	step values
@yearly	(non-standard)
@annually	(non-standard)
@monthly	(non-standard)
@weekly	(non-standard)
@daily	(non-standard)
@hourly	(non-standard)
@reboot	(non-standard)

We created Cronitor because cron itself can't alert you if your jobs fail or never start.
Cronitor is easy to integrate and provides you with instant alerts when things go wrong.

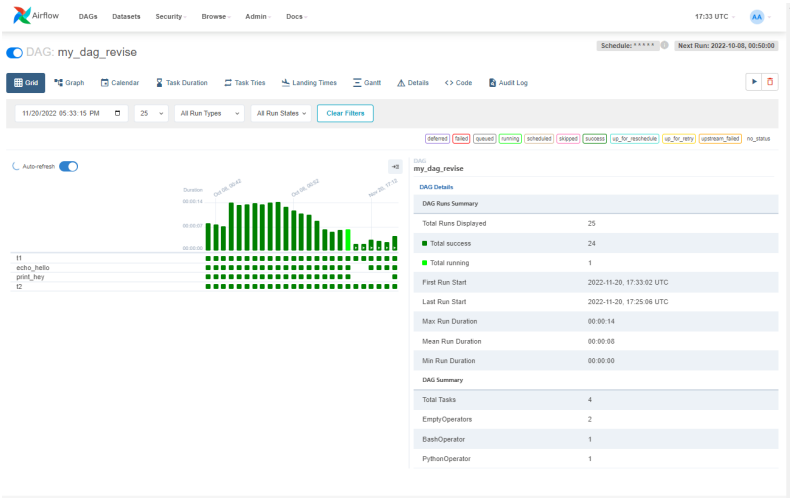
"At 17:00 on day-of-month 16."

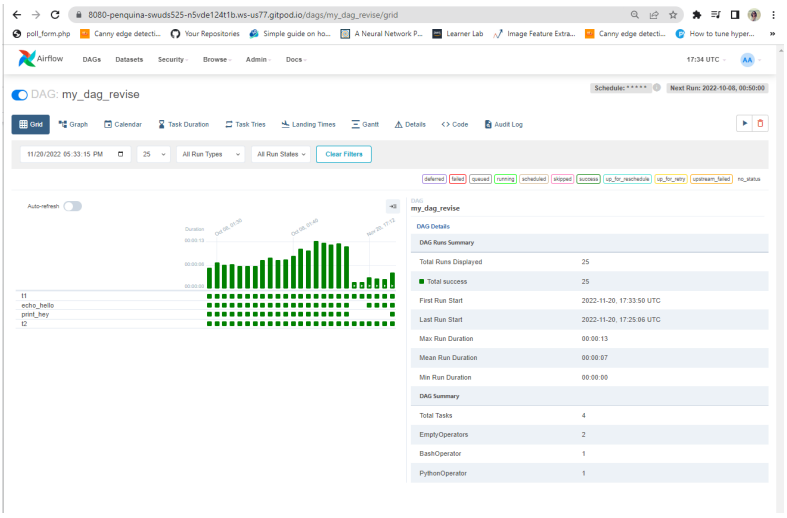
next at 2022-12-16 17:00:00 random

```
0  17  16  *  *
```

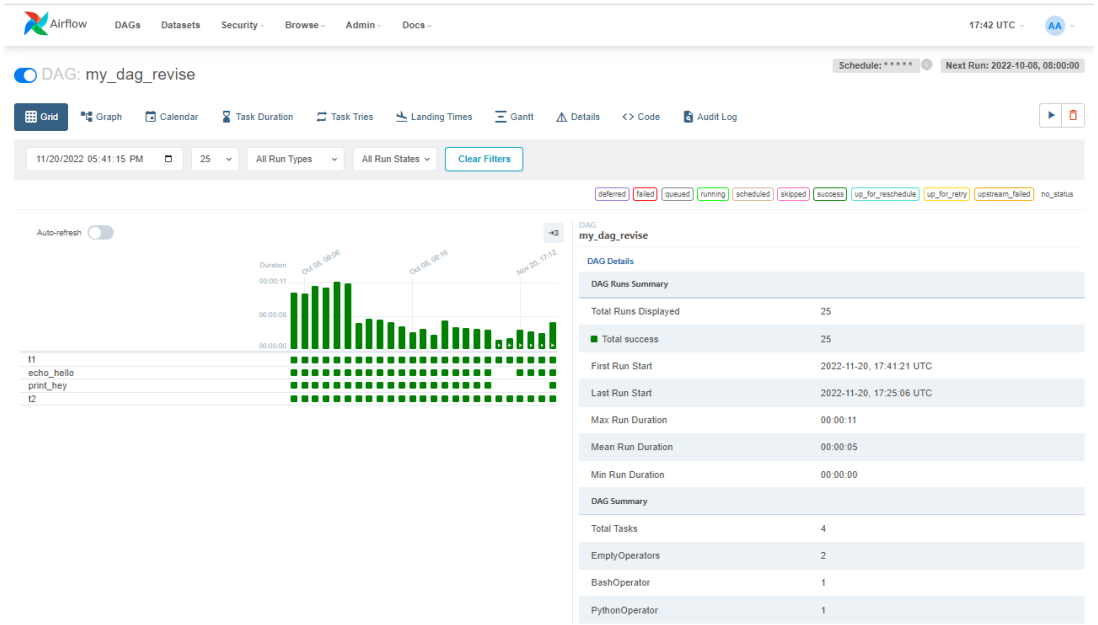
minute	hour	day (month)	month	day (week)

```python
with DAG(
    "my_dag_revise",
    start_date = timezone.datetime(2022, 10, 8),
    schedule = "* * * * *",
    tags =["workshop"],
):
    t1 = EmptyOperator( task_id = "t1")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command= "echo 'hello'",
    )
    def _print_hey():
        print("Hey!")

    print_hey = PythonOperator(
        task_id = "print_hey",
        python_callable = _print_hey,
    )

    t2 = EmptyOperator( task_id = "t2")

#t1 รันก่อน t2
    t1 >> echo_hello >> print_hey >> t2
```

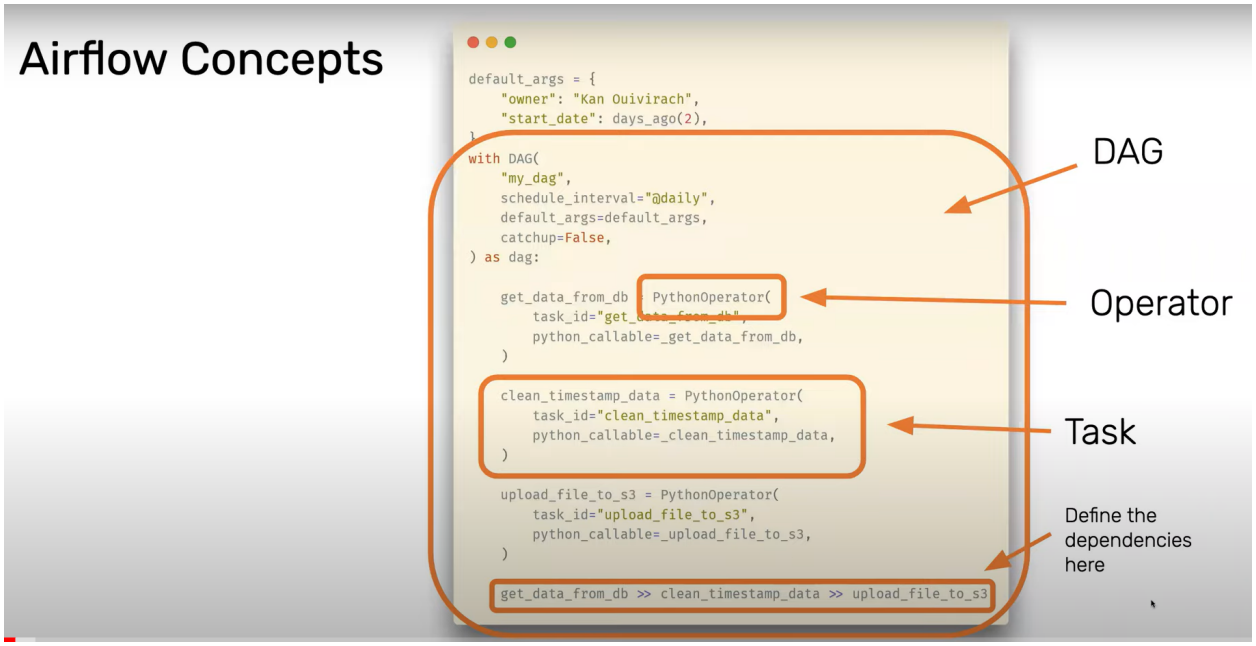




```
#t1 รันก่อน t2
    #t1 >> echo_hello >> print_hey >> t2
    t1 >> [ echo_hello, print_hey] >> t2
```



Week9 ช่วงทบทวนweek 8

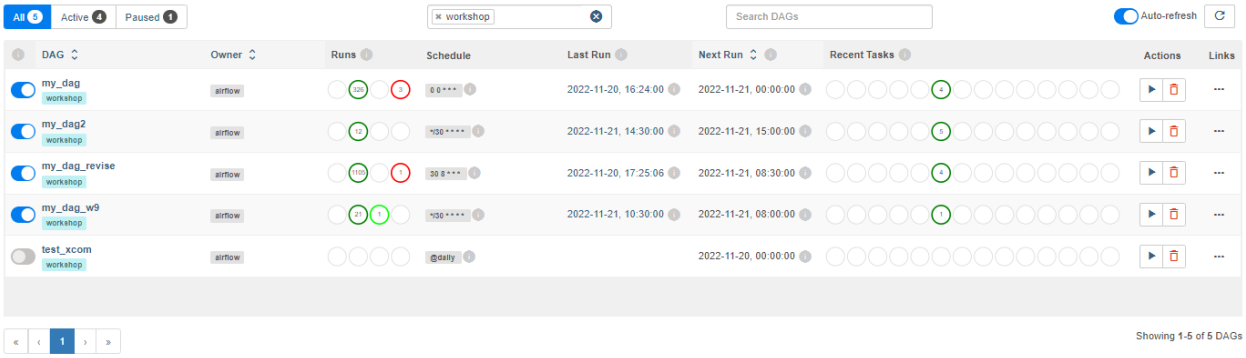


Let's Build this Airflow DAG

DAG runs every 30 minutes

1. EmptyOperator: start
2. BashOperator: echo_hello
3. PythonOperator: say_hello
4. PythonOperator: print_log_messages
5. EmptyOperator: end

DAGs

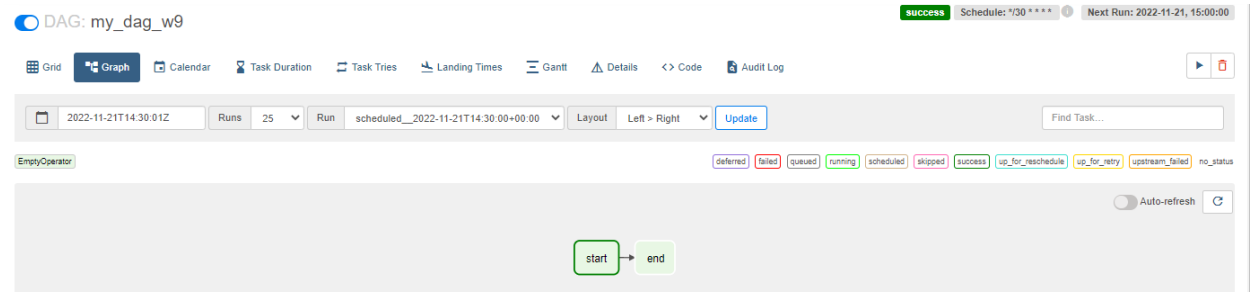


05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py

```python
from airflow import DAG
from airflow.utils import timezone
from airflow.operators.empty import EmptyOperator

with DAG(
    "my_dag_w9",
    start_date = timezone.datetime(2022,11,21),
    schedule = "*/30 * * * *",
    tags = ["workshop"],
) as dag:

    start = EmptyOperator(task_id = "start")

```

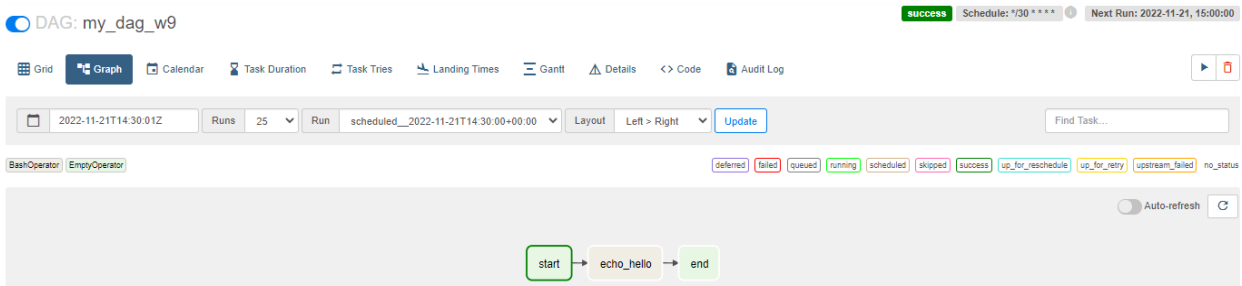
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py

```python
from airflow import DAG
from airflow.utils import timezone
from airflow.operators.empty import EmptyOperator

with DAG(
    "my_dag_w9",
    start_date = timezone.datetime(2022,11,21),
    schedule = "*/30 * * * *",
    tags = ["workshop"],
) as dag:

    start = EmptyOperator(task_id = "start")

    end =  EmptyOperator(task_id = "end")

    start >> end
```



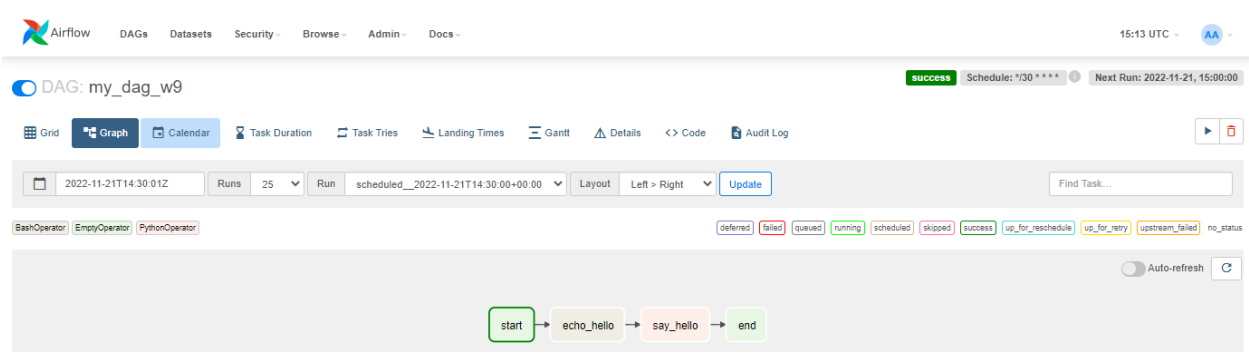
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py

```python
from airflow import DAG
from airflow.utils import timezone
from airflow.operators.empty import EmptyOperator
from airflow.operators.bash import BashOperator

with DAG(
    "my_dag_w9",
    start_date = timezone.datetime(2022,11,21),
    schedule = "*/30 * * * *",
    tags = ["workshop"],
) as dag:

    start = EmptyOperator(task_id = "start")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command = "echo 'hello'",
    )

    end =  EmptyOperator(task_id = "end")

    start >> echo_hello >>end

```



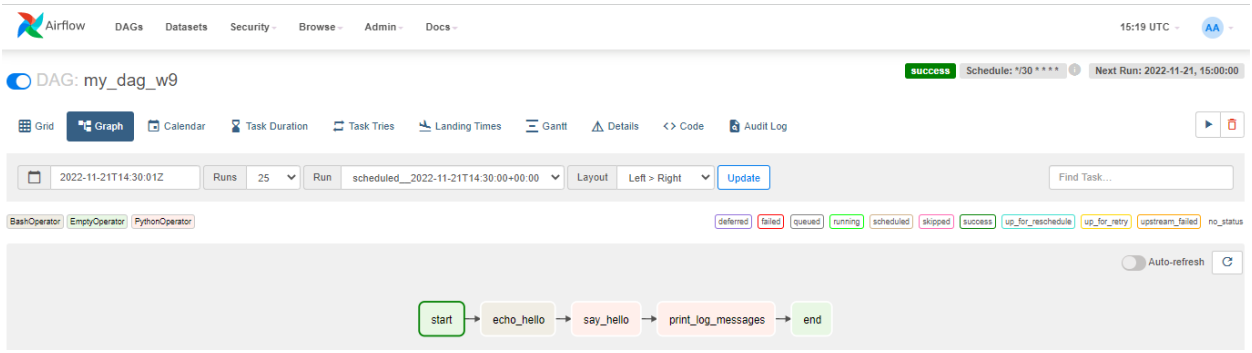
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py

```python
from airflow import DAG
from airflow.utils import timezone
from airflow.operators.empty import EmptyOperator
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator

#ชื่อฟังก์ชั่นต้องไม่เหมือนชื่อtask เลยใส่ _ นำหน้า

def _say_hello():
    print("Hello")

with DAG(
    "my_dag_w9",
    start_date = timezone.datetime(2022,11,21),
    schedule = "*/30 * * * *",
    tags = ["workshop"],
) as dag:

    start = EmptyOperator(task_id = "start")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command = "echo 'hello'",
    )

    say_hello = PythonOperator(
        task_id = "say_hello",
        python_callable = _say_hello,

    )

    end =  EmptyOperator(task_id = "end")

    start >> echo_hello >> say_hello >> end
```

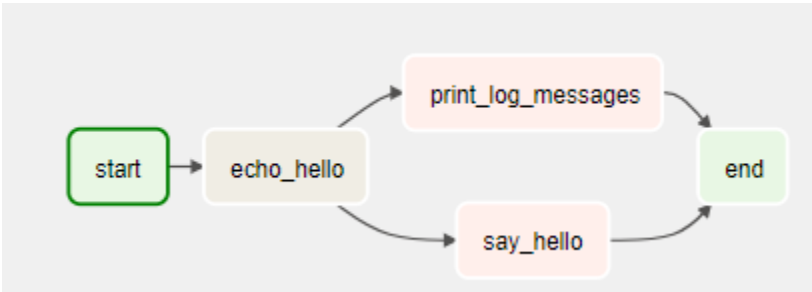


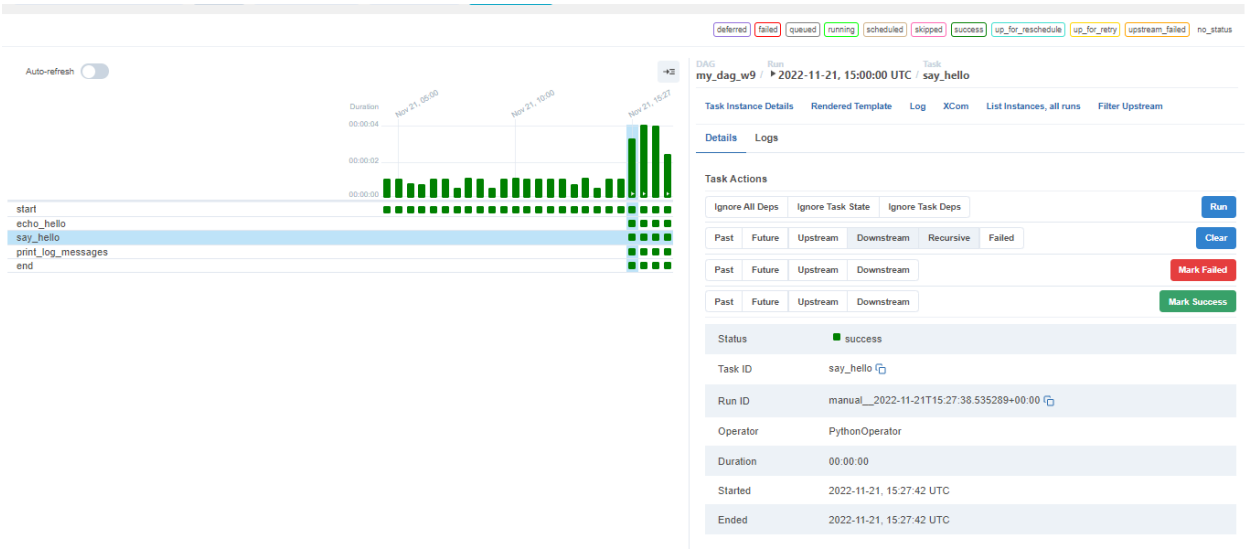
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py

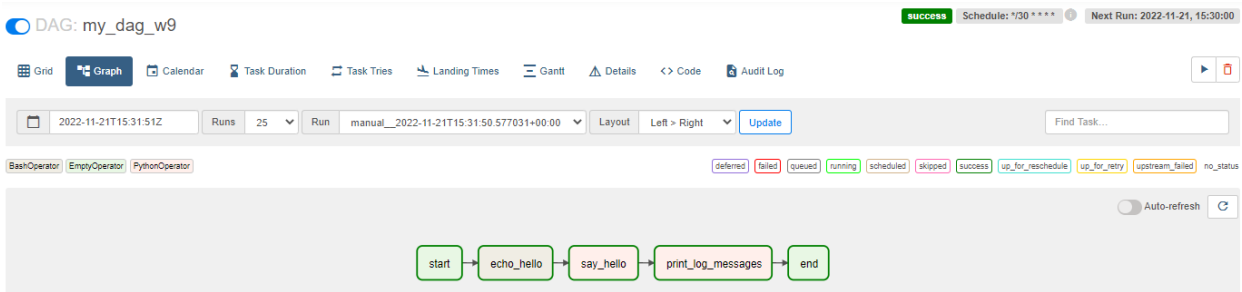
```python
import logging

from airflow import DAG
from airflow.utils import timezone
from airflow.operators.empty import EmptyOperator
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator

#ชื่อฟังก์ชั่นต้องไม่เหมือนชื่อtask เลยใส่ _ นำหน้า

def _say_hello():
    print("Hello")

#logging มีระดับการlog => info, debug แล้วแต่การทำงาน
def _print_log_messages():
    logging.info("Hello from Log")

with DAG(
    "my_dag_w9",
    start_date = timezone.datetime(2022,11,21),
    schedule = "*/30 * * * *",
    tags = ["workshop"],
) as dag:

    start = EmptyOperator(task_id = "start")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command = "echo 'hello'",
    )

    say_hello = PythonOperator(
        task_id = "say_hello",
        python_callable = _say_hello,

    )

    print_log_messages = PythonOperator(
        task_id = "print_log_messages",
        python_callable = _print_log_messages,

    )

    end =  EmptyOperator(task_id = "end")

    start >> echo_hello >> say_hello >> print_log_messages >> end
```




```
#    start >> echo_hello >> say_hello >> print_log_messages >> end
     start >> echo_hello >> [ say_hello , print_log_messages ] >> end
```







Week 9

Good Data Pipeline

It should have these 4 characteristics

1. **Reproducible:** deterministic and idempotent
2. **Future proof:** backfilling and versioning
3. **Fault tolerance:** automatic retry of failed tasks
4. **Transparent:** clarity of where data are

Idempotent

```
fruits = ["Apple", "Orange", "Grape"]
print(fruits)


def add(fruit):
    fruits.append(fruit)


add("Pineapple")
print(fruits)
```

This code changes the value stored in fruits 

Idempotent

```python
fruits = ["Apple", "Orange", "Grape"]
print(fruits)


def add(fruit):
    fruits.append(fruit)


add("Pineapple")
print(fruits)
```

```python
fruits = ["Apple", "Orange", "Grape"]


def add_new(fruit):
    return fruits + [fruit]


new_fruits = add_new("Pineapple")
print(fruits)
print(new_fruits)
```

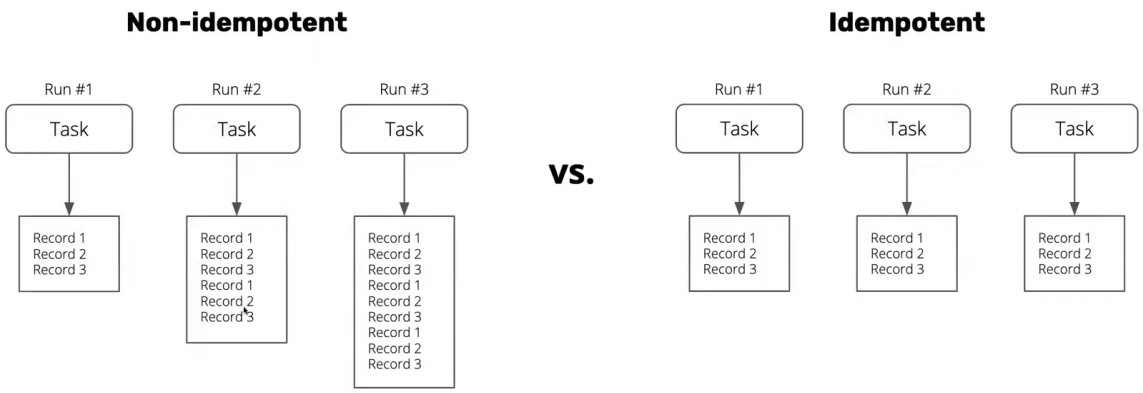
This code changes the value stored in fruits 🙅🏻

This code does **NOT** change the value stored in fruits ✅

```python
fruits = ["Apple", "Orange", "Grape"]
def add(fruit):
    fruits.append(fruit)

add("Pineapple")
print (" Not Idempotent ")
print(fruits)

add("Pineapple")
add("Pineapple")
add("Pineapple")
add("Pineapple")
print(fruits)

fruits2 = ["Apple", "Orange", "Grape"]
def add_new(fruit2):
    return fruits2 + [fruit2]

new_fruits = add_new("Pineapple")
print (" Idempotent ")
print(new_fruits)

new_fruits = add_new("Pineapple")
new_fruits = add_new("Pineapple")
new_fruits = add_new("Pineapple")
new_fruits = add_new("Pineapple")
print(new_fruits)
```

```
gitpod /workspace/swu-ds525/05-creating-and-scheduling-data-pipelines/dags (main) $ python fruits.py
 Not Idempotent
['Apple', 'Orange', 'Grape', 'Pineapple']
['Apple', 'Orange', 'Grape', 'Pineapple', 'Pineapple', 'Pineapple', 'Pineapple', 'Pineapple']
 Idempotent
['Apple', 'Orange', 'Grape', 'Pineapple']
['Apple', 'Orange', 'Grape', 'Pineapple']
```

Non-idempotent vs. Idempotent



Making Pipeline Idempotent

```
from airflow.providers.http.operators import SimpleHttpOperator


with DAG(...) as dag:
    get_data_from_api = SimpleHttpOperator(
        task_id="get_data_from_api",
        endpoint="https://api.somewhere.io/data.json"
                 "?start={{ execution_date }}"
                 "&end={{ execution_date }}",
    )
```

Templating and Macros

Way to pass dynamic data to your DAGs at runtime

Use cases:

- Turning tasks idempotent
- Timestamp for incremental ETL
- Custom user defined parameters for complex operators

See the default variables and macros in the Macros reference in Airflow docs