

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

สิ่งที่คาดหวังในโปรเจคนี้

- นำโคดจาก [Project: Building a Data Modeling with Postgres \(SQL\)](#) มาสร้าง data pipeline โดยใช้ Airflow
- มีการเขียน documentation อธิบายสิ่งที่ตัวเองทำลงไว้ รวมไปถึงการออกแบบ data model
- มี instruction ในการรันโค้ดของตัวเอง

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

ขั้นตอน Overall

① Import DAG, timezone  
② สร้าง Data pipeline ชื่อ "MyDag"

③ save file python

④ กด รันเพลย์ท์ Airflow จนไฟเขียวชี้เขียวที่ Airflow

$t_1 > t_2$

⑤ bash operator, python operator

⑥ gridview

ดูตาม กำหนดเวลา

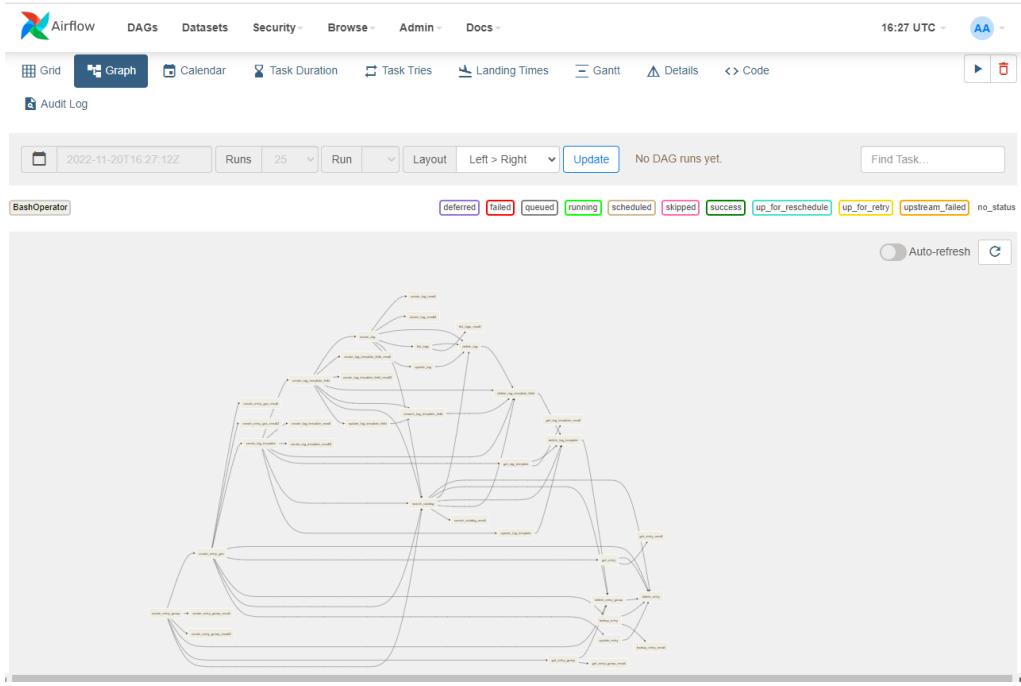
trigger 1 ที่ต้อง 1 dag run

clear task

$t_1 \rightarrow \text{details} \rightarrow \text{clear} \rightarrow \text{confirm}$

as run failed or error

cronitor  $\rightarrow$  expression ที่จะ set ให้ run 自動化 - ผ่านไป 24 ชั่วโมง



05-creating-and-scheduling-data-pipelines > dags > my\_dag\_revise.py

```

1  #import DAG เป็น pipeline
2  from airflow import DAG
3  #import Timezone
4  from airflow.utils import timezone
5  #step 2 ใช้ EmptyOperator เลย import EmptyOperator เข้ามา
6  from airflow.operators.empty import EmptyOperator
7
8
9  #context manager เป็นการประกาศหัว
10 #""my_dag"" ชื่อเดียวกับชื่อ file
11 #start_date 2022, 10, 8
12 # schedule = None ปั้งไม่schedule
13 # step10
14 # schedule
15 # schedule เที่ยงคืน คือ 0
16 #v1 schedule = None ปั้งไม่set schedule
17 with DAG(
18     "my_dag",
19     start_date = timezone.datetime(2022, 10, 8),
20     schedule = None,
21 ):
22     t1 = EmptyOperator( task_id = "t1")
23     t2 = EmptyOperator( task_id = "t2")

```

16:49 UTC AA

DAG	Last Run	Next Run	Recent Tasks
example2	2022-11-20, 12:40:07		
latest_only_with_trigger	2022-11-20, 12:40:07		
my_dag	2022-11-20, 16:24:00	2022-11-20, 00:00:00	1
my_dag2	2022-11-20, 16:00:00	2022-11-20, 16:30:00	1
my_dag_revise	None		

16:55 UTC AA

## DAGs

All 47 Active 3 Paused 44

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions
my_dag	airflow	325	0 0 * * *	2022-11-20, 16:24:00	2022-11-20, 00:00:00	1	
my_dag2	airflow	7	*30 * * * *	2022-11-20, 16:00:00	2022-11-20, 16:30:00	1	
my_dag_revise	airflow	0	None	2022-11-20, 16:55:59			

Trigger DAG Trigger DAG w/ config

16:56 UTC AA

Triggered my\_dag\_revise, it should start any moment now.

## DAGs

All 47 Active 3 Paused 44

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions
my_dag	airflow	325	0 0 * * *	2022-11-20, 16:24:00	2022-11-20, 00:00:00	1	
my_dag2	airflow	7	*30 * * * *	2022-11-20, 16:00:00	2022-11-20, 16:30:00	1	
my_dag_revise	airflow	1	None	2022-11-20, 16:55:59		2	

Showing 1-3 of 3 DAGs

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

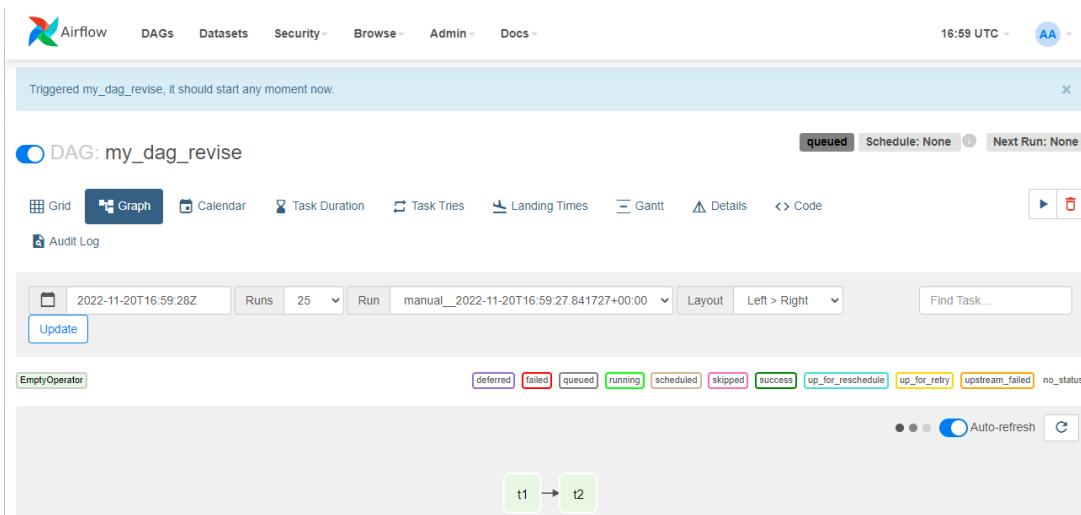
```

graph TD
    t1[task t1]
    t2[task t2]
    t1 --> t2
  
```

```

05-creating-and-scheduling-data-pipelines > dags > my_dag_revise.py
1 #import DAG เป็น pipeline
2 from airflow import DAG
3 #import Timezone
4 from airflow.utils import timezone
5 #step 2 ใช้ EmptyOperator เลย import EmptyOperator เข้ามา
6 from airflow.operators.empty import EmptyOperator
7
8
9 #context manager เป็นการประกาศหัว
10 #""my_dag"" ชื่อเดียวกับชื่อ file
11 #start date 2022, 10, 8
12 # schedule = None ยังไม่กำหนด
13 # step10
14 # schedule
15 # schedule เพียงคืน คือ 0
16 # tags =['workshop'] จะทำให้เป็นชุดเดน
17 #v1 schedule = None ยังไม่กำหนด
18 with DAG(
19     "my_dag_revise",
20     start_date = timezone.datetime(2022, 10, 8),
21     schedule = None,
22     tags =["workshop"],
23 ):
24     t1 = EmptyOperator( task_id = "t1")
25     t2 = EmptyOperator( task_id = "t2")
26
27 #t1 รันก่อน t2
28     t1 >> t2
29

```



```

05-creating-and-scheduling-data-pipelines > dags > my_dag_revise.py
25  # ):
26  #     t1 = EmptyOperator( task_id = "t1")
27  #     t2 = EmptyOperator( task_id = "t2")
28
29  # #t1 依赖于 t2
30  #     t1 >> t2
31
32  #v2
33  with DAG(
34      "my_dag_revise",
35      start_date = timezone.datetime(2022, 10, 8),
36      schedule = None,
37      tags =["workshop"],
38  ):
39      t1 = EmptyOperator( task_id = "t1")
40
41      echo_hello = BashOperator(
42          task_id = "echo_hello",
43          bash_command= "echo 'hello'",
44      )
45
46      t2 = EmptyOperator( task_id = "t2")
47
48  #t1 依赖于 t2
49  |     t1 >> t2

```

DAG: my\_dag\_revise

Grid  Graph  Calendar  Task Duration  Task Tries  Landing Times  Gantt  Details  Code ▶ ⚡

Audit Log

2022-11-20T16:59:28Z Runs 25 Run manual\_2022-11-20T16:59:27.841727+00:00 Layout Left > Right Find Task...

Update

BashOperator EmptyOperator deferred failed queued running scheduled skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

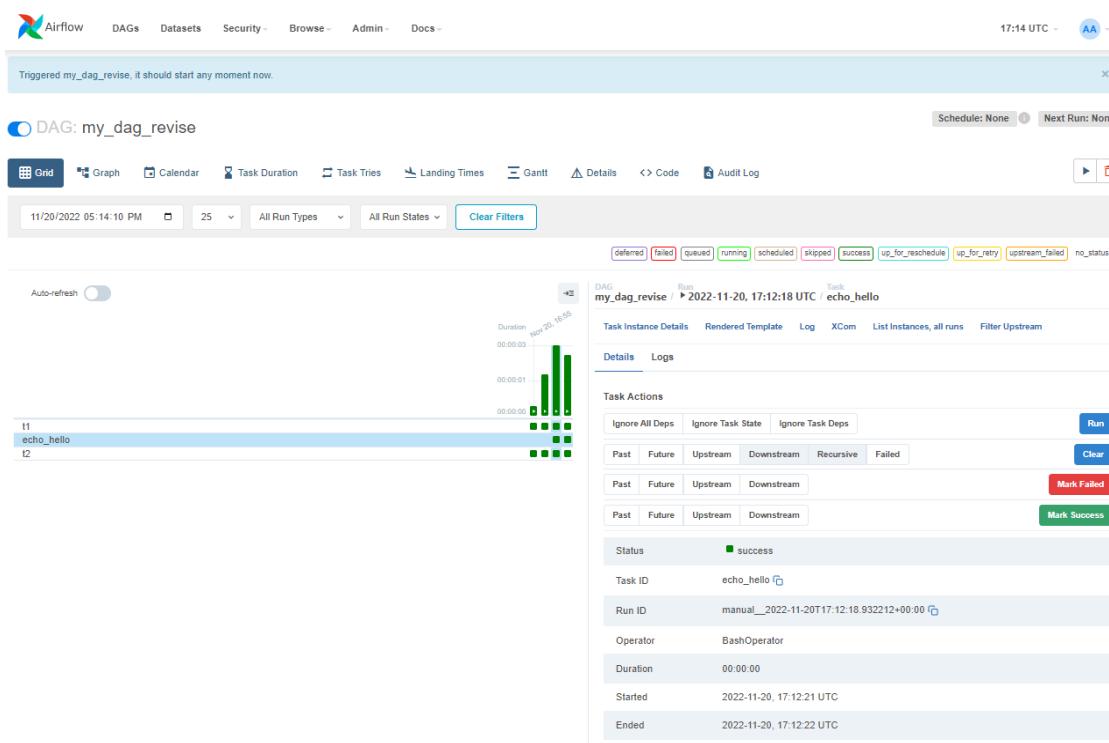
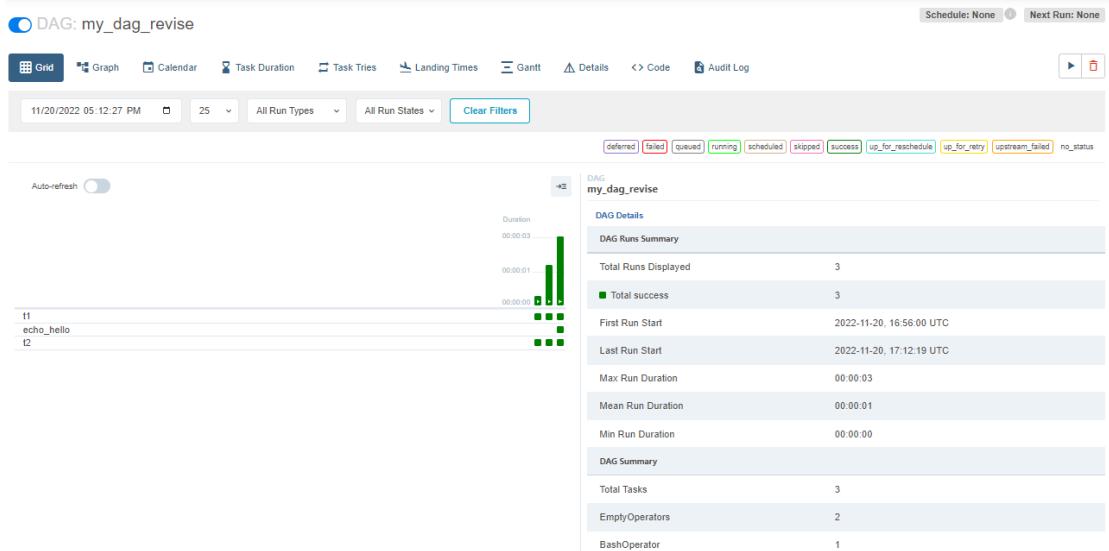
Auto-refresh ↻ C

```

graph LR
    t1[t1] --> echo_hello[echo_hello]
    echo_hello --> t2[t2]

```

#t1 依赖于 t2  
| t1 >> echo\_hello >> t2



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

Duration: Net 20:16:55

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

Details | Logs

(by attempts)

1

All Levels | All File Sources | Wrap | Full Logs | Download | See More

```
130794f1390d
*** Reading local file /opt/airflow/logs/dag_id=echo_dag_revise/run_id=manual_2022-11-20T17:12:18.93222+00:00/task_id=0
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO - Dependencies all met for <TaskInstance: my_dag_revise.echo_hello>
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO - Dependencies all met for <TaskInstance: my_dag_revise.echo_hello>
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO -
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO - Starting attempt 1 of 1
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO -
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO - Executing task (BashOperator): echo_hello> on 2022-11-20 17:12:
[2022-11-20, 17:12:21 UTC] (standard_task_runner.py:14) INFO - Started process 24279 to run task
[2022-11-20, 17:12:21 UTC] (standard_task_runner.py:14) INFO - Running: ['***', 'task', 'run', 'my_dag_revise', 'echo_h
[2022-11-20, 17:12:21 UTC] (standard_task_runner.py:14) INFO - Job 487: Subtask echo_hello>
[2022-11-20, 17:12:21 UTC] (taskinstance.py:116) INFO - Cleaning up the temporary directory for task my_dag_revise.py
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): create_entry_group>, delete_ent
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): delete_entry_group>, create_ent
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): create_entry_group>, dele
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): delete_entry>, create_ent
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): create_tag>, delete_ta
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): delete_tag>, create_ta
[2022-11-20, 17:12:21 UTC] (taskinstance.py:29) WARNING - Dependency <Task(BashOperator): get_id>, prepare_email
```

```
#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
```

17:24 UTC | AA

DAG: my\_dag\_revise

Schedule: None | Next Run: None

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

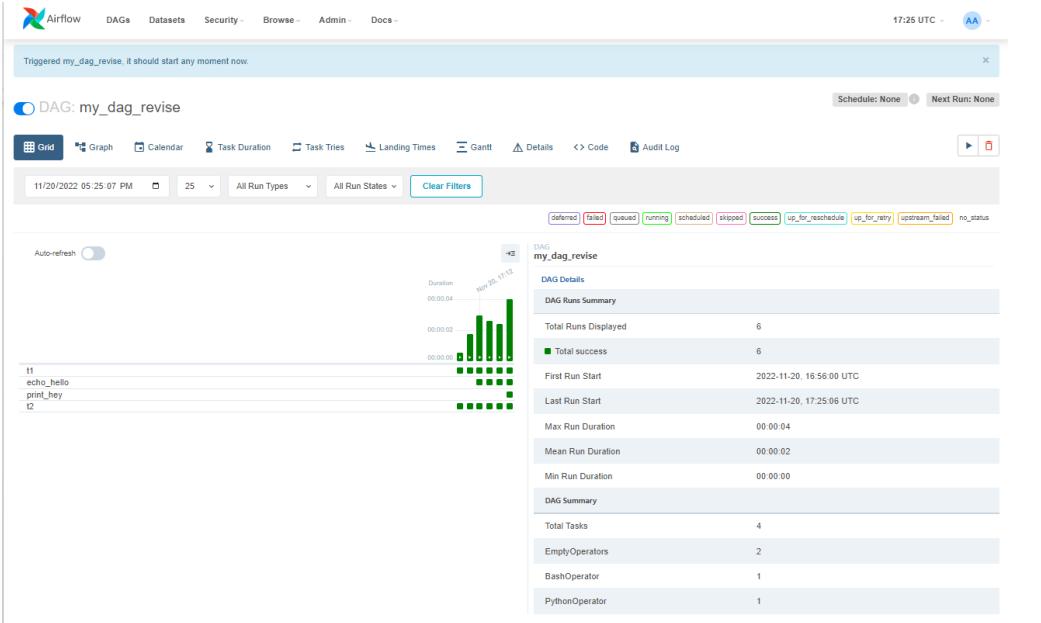
2022-11-20T17:17:00Z | Run | manual\_2022-11-20T17:16:59.347340+00:00 | Layout | Left > Right | Update | Find Task...

EmptyOperator | EmptyOperator | PythonOperator

defered failed queued running scheduled skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh

```
graph LR
    t1[t1] --> echo_hello[echo_hello]
    echo_hello --> print_hey[print_hey]
    print_hey --> t2[t2]
```



**Cronitor**

**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	day (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 detect... Your Repositories Simple guide on ho... A Neural Network P... Learner Lab Image Feature Extra... Canny edge detect... 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	day
(month)				(week)
*				
,				
-				
/				
@yearly				
@annually				
@monthly				
@weekly				
@daily				
@hourly				
@reboot				

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 detect... Your Repositories Simple guide on ho... A Neural Network P... Learner Lab Image Feature Extra... Canny edge detect... 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	day
(month)				(week)
*				
,				
-				
/				
@yearly				
@annually				
@monthly				
@weekly				
@daily				
@hourly				
@reboot				

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#0\_17\_16\_\*\_\*

poll\_form.php Canny edge detect... Your Repositories Simple guide on ho... A Neural Network P... Learner Lab Image Feature Extra... Canny edge detect... How to tune hyper...

## “At 17:00 on day-of-month 16.”

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

0 17 16 \* \*

minute	hour	day	month	day
(month)				(week)
0	17	16	*	*

```

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

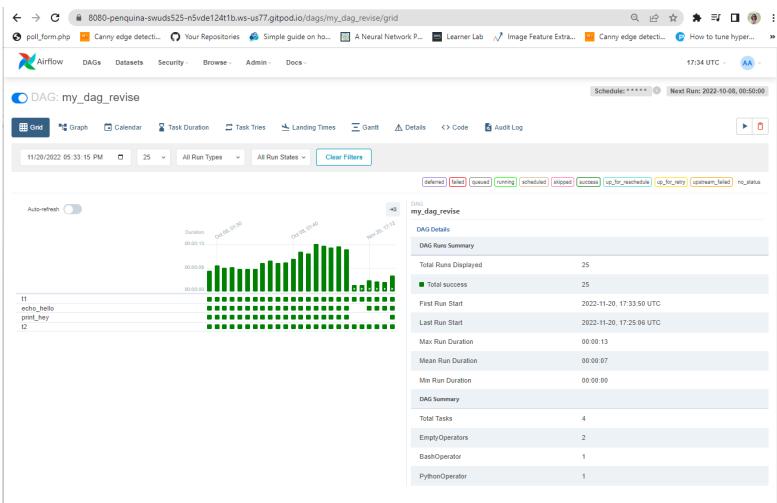
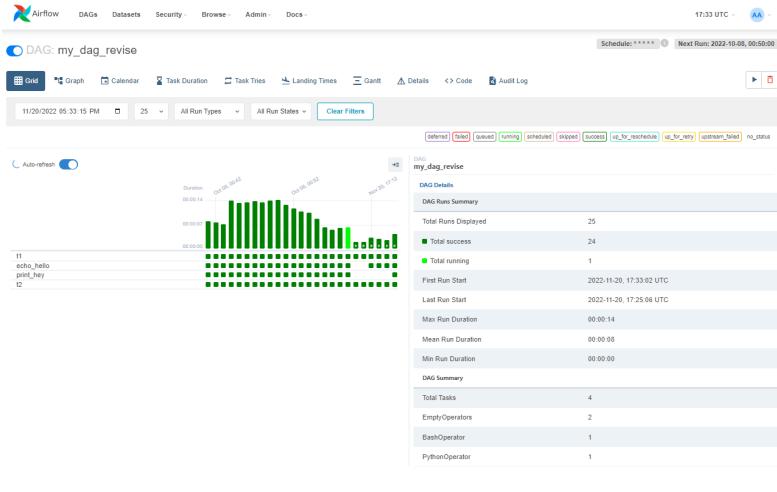
    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command= "echo 'hello'",
    )
    def _print_hey():
        print("Hey!")

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

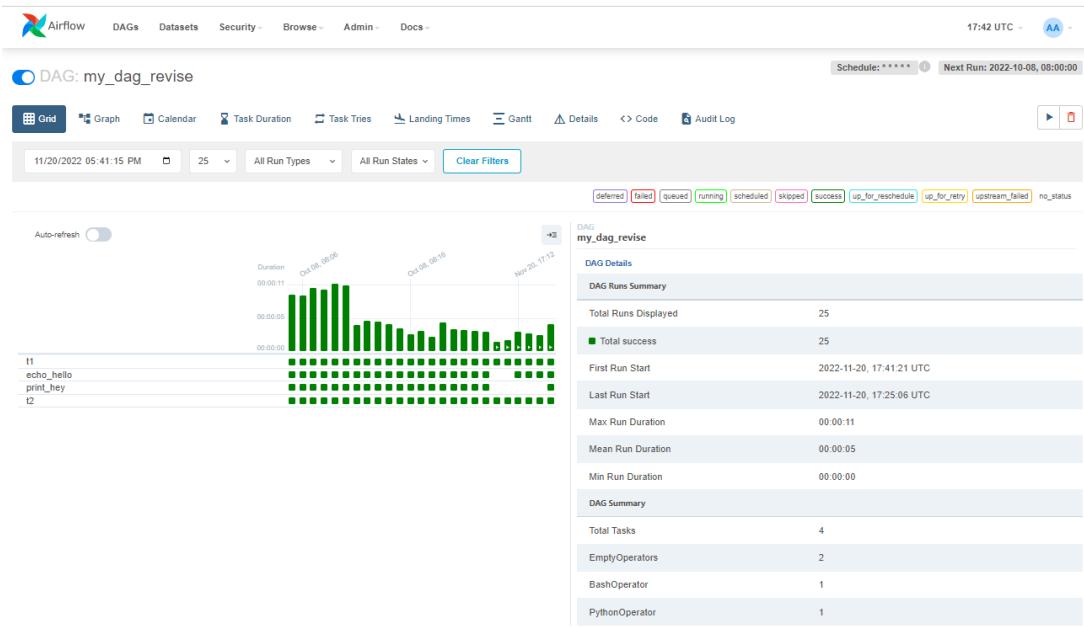
    t2 = EmptyOperator( task_id = "t2")

#t1 នឹងរាយ t2
    t1 >> echo_hello >> print_hey >> t2

```



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




---

Week9 ช่วงทบทวน week 8

## Airflow Concepts

The screenshot shows a Python script for an Airflow DAG. The code defines a DAG named 'my\_dag' with a schedule interval of '@daily'. It contains three tasks: 'get\_data\_from\_db', 'clean\_timestamp\_data', and 'upload\_file\_to\_s3'. Arrows indicate dependencies between the tasks. Annotations on the right side explain the components:

- DAG**: Points to the outermost curly braces defining the DAG.
- Operator**: Points to the first task definition, 'get\_data\_from\_db'.
- Task**: Points to the second task definition, 'clean\_timestamp\_data'.
- Define the dependencies here**: Points to the final line of code showing the task dependencies: `get_data_from_db >> clean_timestamp_data >> upload_file_to_s3`.

```
default_args = {
    "owner": "Kan Ouivirach",
    "start_date": days_ago(2),
}

with DAG(
    "my_dag",
    schedule_interval="@daily",
    default_args=default_args,
    catchup=False,
) as dag:

    get_data_from_db = PythonOperator(
        task_id="get_data_from_db",
        python_callable=_get_data_from_db,
    )

    clean_timestamp_data = PythonOperator(
        task_id="clean_timestamp_data",
        python_callable=_clean_timestamp_data,
    )

    upload_file_to_s3 = PythonOperator(
        task_id="upload_file_to_s3",
        python_callable=_upload_file_to_s3,
    )

    get_data_from_db >> clean_timestamp_data >> upload_file_to_s3
```

## 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

All (5)	Active (0)	Paused (1)	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
my_dag [workshop]	airflow	0	0/30 * * * *	2022-11-20, 16:24:00	2022-11-21, 00:00:00	0/30 * * * *	0/30 * * * *		
my_dag2 [workshop]	airflow	1	*/30 * * * *	2022-11-21, 14:30:00	2022-11-21, 15:00:00	*/30 * * * *	*/30 * * * *		
my_dag_revise [workshop]	airflow	0	0/8 * * * *	2022-11-20, 17:25:06	2022-11-21, 08:30:00	0/8 * * * *	0/8 * * * *		
my_dag_w9 [workshop]	airflow	0	*/30 * * * *	2022-11-21, 10:30:00	2022-11-21, 08:00:00	*/30 * * * *	*/30 * * * *		
test_xcom [workshop]	airflow	0	daily		2022-11-20, 00:00:00		*/30 * * * *		

Showing 1-5 of 5 DAGs

05-creating-and-scheduling-data-pipelines > dags > my\_dag\_w9.py

```
1  from airflow import DAG
2  from airflow.utils import timezone
3  from airflow.operators.empty import EmptyOperator
4
5  with DAG(
6      "my_dag_w9",
7      start_date = timezone.datetime(2022,11,21),
8      schedule = "*/30 * * * *",
9      tags = ["workshop"],
10 ) as dag:
11
12     start = EmptyOperator(task_id = "start")
13
```

05-creating-and-scheduling-data-pipelines > dags > my\_dag\_w9.py

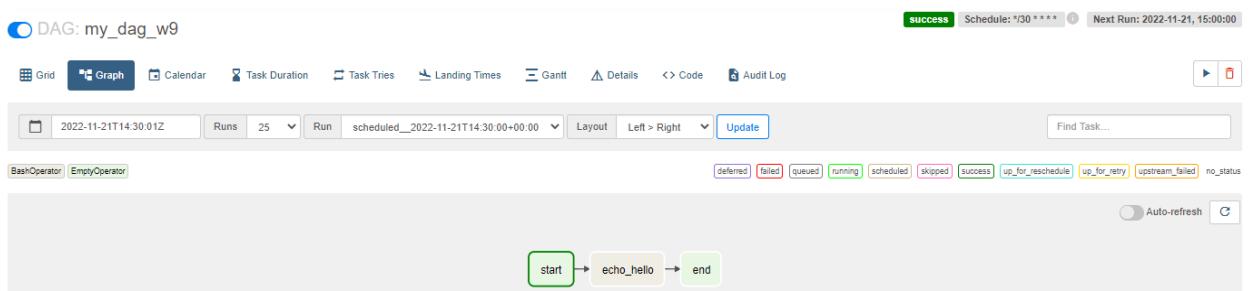
```
1  from airflow import DAG
2  from airflow.utils import timezone
3  from airflow.operators.empty import EmptyOperator
4
5  with DAG(
6      "my_dag_w9",
7      start_date = timezone.datetime(2022,11,21),
8      schedule = "*/30 * * * *",
9      tags = ["workshop"],
10 ) as dag:
11
12     start = EmptyOperator(task_id = "start")
13
14     end =  EmptyOperator(task_id = "end")
15
16     start >> end
```

DAG: my\_dag\_w9

```

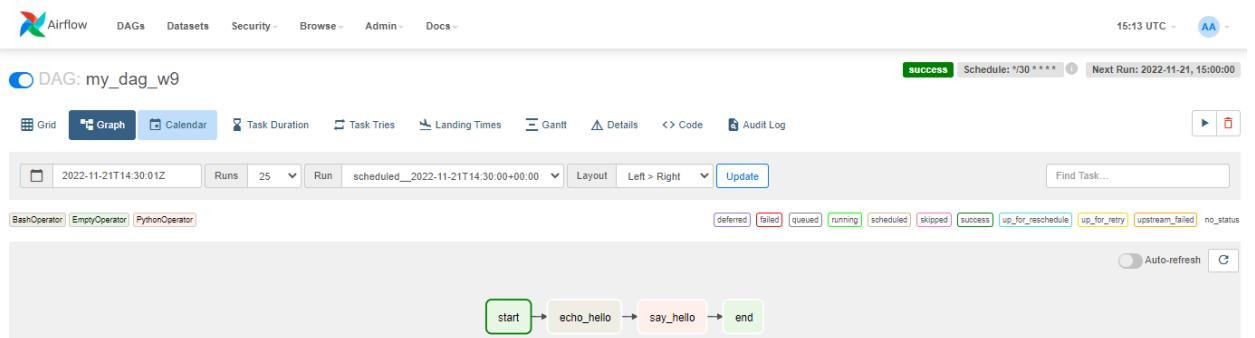
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py
1  from airflow import DAG
2  from airflow.utils import timezone
3  from airflow.operators.empty import EmptyOperator
4  from airflow.operators.bash import BashOperator
5
6  with DAG(
7      "my_dag_w9",
8      start_date = timezone.datetime(2022,11,21),
9      schedule = "*/30 * * * *",
10     tags = ["workshop"],
11 ) as dag:
12
13     start = EmptyOperator(task_id = "start")
14
15     echo_hello = BashOperator(
16         task_id = "echo_hello",
17         bash_command = "echo 'hello'",
18     )
19
20     end = EmptyOperator(task_id = "end")
21
22     start >> echo_hello >> end
23

```



05-creating-and-scheduling-data-pipelines > dags > my\_dag\_w9.py

```
1  from airflow import DAG
2  from airflow.utils import timezone
3  from airflow.operators.empty import EmptyOperator
4  from airflow.operators.bash import BashOperator
5  from airflow.operators.python import PythonOperator
6
7  # ข้อพึงกշั่นต้องไม่เหมือนชื่อ task เลยใส่ _ หน้าหัว
8
9  def _say_hello():
10     print("Hello")
11
12 with DAG(
13     "my_dag_w9",
14     start_date = timezone.datetime(2022,11,21),
15     schedule = "*/30 * * * *",
16     tags = ["workshop"],
17 ) as dag:
18
19     start = EmptyOperator(task_id = "start")
20
21     echo_hello = BashOperator(
22         task_id = "echo_hello",
23         bash_command = "echo 'hello'",
24     )
25
26     say_hello = PythonOperator(
27         task_id = "say_hello",
28         python_callable = _say_hello,
29     )
30
31     end = EmptyOperator(task_id = "end")
32
33     start >> echo_hello >> say_hello >> end
34
```



```
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py
 1 import logging
 2
 3 from airflow import DAG
 4 from airflow.utils import timezone
 5 from airflow.operators.empty import EmptyOperator
 6 from airflow.operators.bash import BashOperator
 7 from airflow.operators.python import PythonOperator
 8
 9 #ชื่อฟังก์ชันต้องไม่เหมือนชื่อ task เมนูสี _ นำหน้า
10
11 def _say_hello():
12     print("Hello")
13
14 #logging มีระดับการ log => info, debug และแต่ละการที่ทำงาน
15 def _print_log_messages():
16     logging.info("Hello from Log")
17
18 with DAG(
19     "my_dag_w9",
20     start_date = timezone.datetime(2022,11,21),
21     schedule = "*/*30 * * * *",
22     tags = ["workshop"],
23 ) as dag:
24
25     start = EmptyOperator(task_id = "start")
26
27     echo_hello = BashOperator(
28         task_id = "echo_hello",
29         bash_command = "echo 'hello'", )
30
31
32     say_hello = PythonOperator(
33         task_id = "say_hello",
34         python_callable = _say_hello,
35     )
36
37
38     print_log_messages = PythonOperator(
39         task_id = "print_log_messages",
40         python_callable = _print_log_messages,
41     )
42
43
44     end = EmptyOperator(task_id = "end")
45
46     start >> echo_hello >> say_hello >> print_log_messages >> end
```

15:19 UTC - AA -

DAG: my\_dag\_w9

success Schedule: \*30 \* \* \* \* Next Run: 2022-11-21, 15:00:00

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

2022-11-21T14:30:01Z Runs 25 Run scheduled\_2022-11-21T14:30:00+00:00 Layout Left > Right Update Find Task...

BashOperator EmptyOperator PythonOperator deferred failed queued running scheduled skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh C

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

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
- Past Future Upstream Downstream
- Past Future Upstream Downstream
- Mark Failed
- Mark Success

Status success

Task ID say\_hello

Run ID manual\_2022-11-21T15:27:38.535289+00:00

Operator PythonOperator

Duration 00:00:00

Started 2022-11-21, 15:27:42 UTC

Ended 2022-11-21, 15:27:42 UTC

DAG: my\_dag\_w9

success Schedule: \*30 \* \* \* \* Next Run: 2022-11-21, 15:30:00

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

2022-11-21T15:31:51Z Runs 25 Run manual\_2022-11-21T15:31:50.577031+00:00 Layout Left > Right Update Find Task...

BashOperator EmptyOperator PythonOperator deferred failed queued running scheduled skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

Auto-refresh C

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

# 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

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

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

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

This code changes the value stored in fruits



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

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

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

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



```
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]

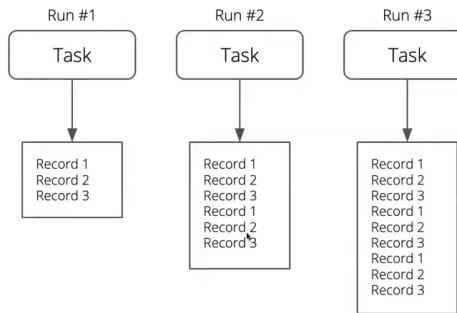
new_fruits = add_new("Pineapple")
print (" Idempotent ")
print(new_fruits)

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']

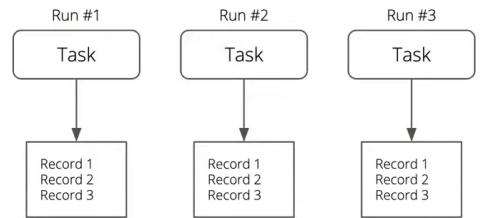
# Non-idempotent vs. Idempotent

## 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

# airflow template macro

The screenshot shows the Apache Airflow documentation website. At the top, there is a navigation bar with links to Community, Meetups, Documentation, Use-cases, Announcements, Blog, and Ecosystem. Below the navigation bar, there is a search bar with the placeholder "Search docs" and a magnifying glass icon. To the left of the search bar, the text "Version: 2.4.1" is displayed. On the far left, there is a sidebar titled "CONTENT" containing a list of links to various Airflow documentation pages, such as Overview, Project, License, Quick Start, Installation, Upgrading from 1.10 to 2, Tutorials, How-to Guides, UI / Screenshots, Concepts, Executor, DAG Runs, Plugins, Security, Logging & Monitoring, Time Zones, Using the CLI, Integration, Kubernetes, Lineage, Listeners, DAG Serialization, Modules Management, Release Policies, Release Notes, Best Practices, Production Deployment, FAQ, and Privacy Notice. Below this, there is a section titled "REFERENCES" with links to Operators and hooks and CLI. The main content area has a breadcrumb navigation path "Home / Templates reference". The title "Templates reference" is bolded. Below the title, there is a note: "Variables, macros and filters can be used in templates (see the [Jinja Templating](#) section)". Another note states: "The following come for free out of the box with Airflow. Additional custom macros can be added globally through [Plugins](#), or at a DAG level through the `DAG.user_defined_macros` argument.". A section titled "Variables" is present, stating: "The Airflow engine passes a few variables by default that are accessible in all templates". A table titled "Variables" provides a detailed list of these variables:

Variable	Description
<code>{{ data_interval_start }}</code>	Start of the data interval ( <code>pendulum.DateTime</code> ).
<code>{{ data_interval_end }}</code>	End of the data interval ( <code>pendulum.DateTime</code> ).
<code>{{ ds }}</code>	The DAG run's logical date as <code>YYYY-MM-DD</code> . Same as <code>{{ dag_run.logical_date   ds }}</code> .
<code>{{ ds_nodash }}</code>	Same as <code>{{ dag_run.logical_date   ds_nodash }}</code> .
<code>{{ ts }}</code>	Same as <code>{{ dag_run.logical_date   ts }}</code> . Example: <code>2018-01-01T00:00:00+00:00</code> .
<code>{{ ts_nodash_with_tz }}</code>	Same as <code>{{ dag_run.logical_date   ts_nodash_with_tz }}</code> . Example: <code>20180101T000000+0000</code> .
<code>{{ ts_nodash }}</code>	Same as <code>{{ dag_run.logical_date   ts_nodash }}</code> . Example: <code>20180101T000000</code> .
<code>{{ prev_data_interval_start_success }}</code>	Start of the data interval from prior successful DAG run ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ prev_data_interval_end_success }}</code>	End of the data interval from prior successful DAG run ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ prev_start_date_success }}</code>	Start date from prior successful dag run (if available) ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ dag }}</code>	The DAG object.
<code>{{ task }}</code>	The Task object.
<code>{{ macros }}</code>	A reference to the <code>macros</code> package, described below.

## Variables

The Airflow engine passes a few variables by default that are accessible in all templates

Variable	Description
<code>{{ data_interval_start }}</code>	Start of the data interval ( <code>pendulum.DateTime</code> ).
<code>{{ data_interval_end }}</code>	End of the data interval ( <code>pendulum.DateTime</code> ).
<code>{{ ds }}</code>	The DAG run's logical date as <code>YYYY-MM-DD</code> . Same as <code>{{ dag_run.logical_date   ds }}</code> .
<code>{{ ds_nodash }}</code>	Same as <code>{{ dag_run.logical_date   ds_nodash }}</code> .
<code>{{ ts }}</code>	Same as <code>{{ dag_run.logical_date   ts }}</code> . Example: <code>2018-01-01T00:00:00+00:00</code> .
<code>{{ ts_nodash_with_tz }}</code>	Same as <code>{{ dag_run.logical_date   ts_nodash_with_tz }}</code> . Example: <code>20180101T000000+0000</code>
<code>{{ ts_nodash }}</code>	Same as <code>{{ dag_run.logical_date   ts_nodash }}</code> . Example: <code>20180101T000000</code> .
<code>{{ prev_data_interval_start_success }}</code>	Start of the data interval from prior successful DAG run ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ prev_data_interval_end_success }}</code>	End of the data interval from prior successful DAG run ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ prev_start_date_success }}</code>	Start date from prior successful dag run (if available) ( <code>pendulum.DateTime</code> or <code>None</code> ).
<code>{{ dag }}</code>	The DAG object.
<code>{{ task }}</code>	The Task object.

```
05-creating-and-scheduling-data-pipelines > dags > my_dag_w9.py
 1 import logging
 2
 3 from airflow import DAG
 4 from airflow.utils import timezone
 5 from airflow.operators.empty import EmptyOperator
 6 from airflow.operators.bash import BashOperator
 7 from airflow.operators.python import PythonOperator
 8
 9 #ชื่อฟังก์ชันต้องไม่เหมือนชื่อ task เลยใช่ _ นำหน้า
10
11 def _say_hello():
12     print("Hello")
13
14 #logging มีระดับของ log => info, debug และการทิ้งงาน
15 def _print_log_messages():
16     logging.info("Hello from Log")
17
18 with DAG(
19     "my_dag_w9",
20     start_date = timezone.datetime(2022,11,21),
21     schedule = "*/*/*/*/*",
22     tags = ["workshop"],
23     catchup = False,
24 ) as dag:
25
26     start = EmptyOperator(task_id = "start")
27
28     echo_hello = BashOperator(
29         task_id = "echo_hello",
30         bash_command = "echo 'hello' on {{ ds }}",
31     )
32
33     say_hello = PythonOperator(
34         task_id = "say_hello",
35         python_callable = _say_hello,
36     )
37
38     print_log_messages = PythonOperator(
39         task_id = "print_log_messages",
40         python_callable = _print_log_messages,
41     )
42
43     end = EmptyOperator(task_id = "end")
44
45     start >> echo_hello >> say_hello >> print_log_messages >> end
46
47 #    start >> echo_hello >> [ say_hello , print_log_messages ] >> end
48
49
50
```

Airflow DAGs Datasets Security Browse Admin Docs 16:13 UTC AA

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

11/21/2022 04:11:59 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 21, 06:00 Nov 21, 11:00 Nov 21, 15:00

00:00:07  
00:00:03  
00:00:00

start echo\_hello say\_hello print\_log\_messages end

DAG my\_dag\_w9 / Run 2022-11-21, 16:00:00 UTC / Task echo\_hello

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

Details Logs

(by attempts) 1

All Levels All File Sources Wrap Full Logs Download See More

```
[2022-11-21, 16:12:01 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:12:01 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:12:01 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:12:01 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:12:01 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:12:01 UTC] [task_command.py:384] INFO - Running <TaskInstance: my_dag_w9.echo_hello.manual_2022-11-21T16:12:01>[2022-11-21, 16:12:01 UTC] [taskinstance.py:1592] INFO - Exporting the following env vars:[AIRFLOW_CTX_DAG_ID=my_dag_w9][AIRFLOW_CTX_TASK_ID=echo_hello][AIRFLOW_CTX_EXECUTION_DATE=2022-11-21T16:11:58.955654+00:00][AIRFLOW_CTX_TRY_NUMBER=1][AIRFLOW_CTX_DAG_RUN_ID=manual_2022-11-21T16:11:58.955654+00:00][2022-11-21, 16:12:01 UTC] [subprocess.py:63] INFO - Tmp dir root location: /tmp[2022-11-21, 16:12:01 UTC] [subprocess.py:75] INFO - Running command: ['>>> echo "Hello" on 2022-11-21'...][2022-11-21, 16:12:01 UTC] [subprocess.py:86] INFO - Output: Hello[2022-11-21, 16:12:01 UTC] [subprocess.py:87] INFO - Command exited with return code 0[2022-11-21, 16:12:01 UTC] [taskinstance.py:1406] INFO - Marking task as SUCCESS, dag_id=my_dag_w9, task_id#echo_hello, ex...[2022-11-21, 16:12:01 UTC] [local_task_job.py:164] INFO - Task exited with return code 0[2022-11-21, 16:12:01 UTC] [local_task_job.py:273] INFO - 1 downstream tasks scheduled from follow-on schedule check
```

```
def _say_hello(**context):
    print(context)
    print("Hello")
```

Airflow DAGs Datasets Security Browse Admin Docs 16:18 UTC AA

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

11/21/2022 04:18:32 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 21, 06:30 Nov 21, 11:30 Nov 21, 15:30

00:00:07  
00:00:03  
00:00:00

start echo\_hello say\_hello print\_log\_messages end

DAG my\_dag\_w9 / Run 2022-11-21, 16:00:00 UTC / Task say\_hello

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

Details Logs

(by attempts) 1

All Levels All File Sources Wrap Full Logs Download See More

```
[2022-11-21, 16:18:35 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; load, transform, load alre...[2022-11-21, 16:18:35 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:18:35 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; load, transform, load alre...[2022-11-21, 16:18:35 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:18:35 UTC] [taskmixin.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; transform, load alre...[2022-11-21, 16:18:35 UTC] [task_command.py:205] WARNING - Dependency <Task_PythonDecoratedOperator>; load, transform, load alre...[2022-11-21, 16:18:35 UTC] [taskinstance.py:1592] INFO - Running <TaskInstance: my_dag_w9.say_hello.manual_2022-11-21T16:18:35>[2022-11-21, 16:18:35 UTC] [taskinstance.py:1592] INFO - Exporting the following env vars:[AIRFLOW_CTX_DAG_ID=my_dag_w9][AIRFLOW_CTX_TASK_ID=say_hello][AIRFLOW_CTX_EXECUTION_DATE=2022-11-21T16:18:31.924719+00:00][AIRFLOW_CTX_TRY_NUMBER=1][AIRFLOW_CTX_DAG_RUN_ID=manual_2022-11-21T16:18:31.924719+00:00][2022-11-21, 16:18:35 UTC] [log_subscriber.py:117] INFO - Configuring AirflowConfigParser object at 0x7f33...[2022-11-21, 16:18:35 UTC] [log_subscriber.py:117] INFO - Hello[2022-11-21, 16:18:36 UTC] [python.py:177] INFO - Done. Returned value was: None[2022-11-21, 16:18:36 UTC] [taskinstance.py:1406] INFO - Marking task as SUCCESS, dag_id=my_dag_w9, task_id=say_hello, ex...[2022-11-21, 16:18:36 UTC] [local_task_job.py:164] INFO - Task exited with return code 0[2022-11-21, 16:18:36 UTC] [local_task_job.py:273] INFO - 1 downstream tasks scheduled from follow-on schedule check
```

The screenshot shows the Airflow web interface. At the top, there are navigation links: DAGs, Datasets, Security, Browse, Admin, Docs, and a timestamp of 16:23 UTC. On the far right, there's a user icon.

The main area displays a DAG run timeline for the DAG 'my\_dag\_v9'. The timeline shows tasks starting at 00:00:00 on Nov 21, 11:30, and ending at 00:00:07 on Nov 21, 15:31. The tasks listed are: start, echo\_hello, say\_hello, print\_log\_messages, and end. Each task is represented by a bar indicating its duration and execution status.

To the right of the timeline, there's a detailed view of the 'say\_hello' task. It includes tabs for Task Instance Details, Rendered Template, Log, XCom, List Instances, all runs, and Filter Upstream. The 'Logs' tab is selected, showing log entries from All Levels and All File Sources. The logs detail the execution of the task, including dependency information and environment variable exports.

```
def _say_hello(**context):
    print(context)
    datestamp = context["ds"]
    print(f"Hello! on {datestamp}")
```

## BashOperator airflow

All

Shopping

Videos

News

Images

More

About 28,800 results (0.36 seconds)

<https://airflow.apache.org> › howto › operator › bash

⋮

## [BashOperator — Airflow Documentation](#)

Use the **BashOperator** to execute commands in a Bash shell. `airflow/example_dags/example_bash_operator.py`[source]. `run_this = BashOperator ...`

You've visited this page 4 times. Last visit: 10/13/22

The screenshot shows the Apache Airflow documentation for the `airflow.operators.bash` module. The top navigation bar includes links for Community, Meetups, Documentation, Use-cases, Announcements, Blog, and Ecosystem. A sidebar on the left lists various documentation sections such as Overview, Project, License, Quick Start, Installation, Upgrading from 1.10 to 2, Tutorials, How-to Guides, UI / Screenshots, Concepts, Executor, DAG Runs, Plugins, Security, Logging & Monitoring, Time Zones, Using the CLI, Integration, Kubernetes, Lineage, Listeners, DAG Serialization, Modules Management, Release Policies, Release Notes, Best Practices, Production Deployment, FAQ, Privacy Notice, and References. The main content area displays the `airflow.operators.bash` module contents, showing the `BashOperator` class definition and its parameters. The right sidebar shows the module structure for `airflow.operators.bash`, including classes like `BashOperator`, `template_fields`, `template_field`, `renderers`, `template_ext`, `ui_color`, `subprocess_hook`, `get_env`, `execute`, and `on_kill`.

```
class airflow.operators.bash.BashOperator(¶, bash_command, env=None, append_env=False, output_encoding='utf-8', skip_exit_code=99, cwd=None, **kwargs) [source]
Bases: airflow.models.baseoperator.BaseOperator
Execute a Bash script command or set of commands.

One shot
For more information on how to use this operator, take a look at the guide: BashOperator

If BaseOperator.do_xcom_push is True, the last line written to stdout will also be pushed to an XCom when the bash command completes

Parameters
• bash_command (str) – The command, set of commands or reference to a bash script (must be '.sh') to be executed. (templated)
• env (dict(str, str) | None) – If env is not None, it must be a dict that defines the environment variables for the new process; these are used instead of inheriting the current process environment, which is the default behavior. (templated)
• append_env (bool) – If False(default) uses the environment variables passed in env params and does not inherit the current process environment. If True, inherits the environment variables from current passes and then environment variable passed by the user will either update the existing inherited environment variables or the new variables gets appended to it
• output_encoding (str) – Output encoding of bash command
• skip_exit_code (int) – If task exits with this exit code, leave the task in skipped state (default: 99). If set to None, any non-zero exit code will be treated as a failure.
• cwd (str | None) – Working directory to execute the command in. If None (default), the command is run in a temporary directory.
```

`template_fields:Sequence[str] = ['templates_dict', 'op_args', 'op_kwargs']` [source] ↗

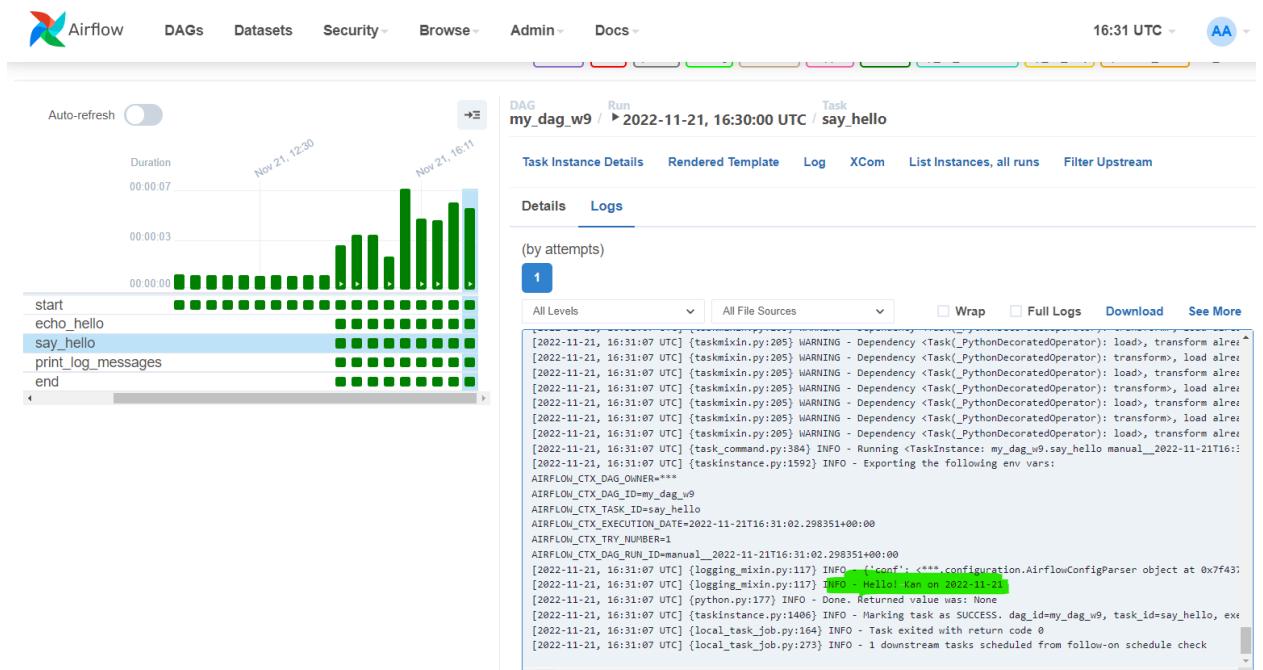
```
def _say_hello( name="", **context ):
    print(context)
    datestamp = context["ds"]
    print(f"Hello! {name} on {datestamp}")

#logging มีระดับการ log => info, debug และแม่ก้ารที่งาน
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"],
    catchup = False,
) as dag:
    start = EmptyOperator(task_id = "start")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command = "echo 'hello' on {{ ds }}"
    )

# op_kwarg รับข้อมูลเป็น dict
    say_hello = PythonOperator(
        task_id = "say_hello",
        python_callable = _say_hello,
        op_kwargs = {
            "name": "Kan",
        }
    )
```



```
def _say_hello( name="", **context ):
    print(context)
    datestamp = context["ds"]
    print(f"Hello! {name} on {datestamp}")

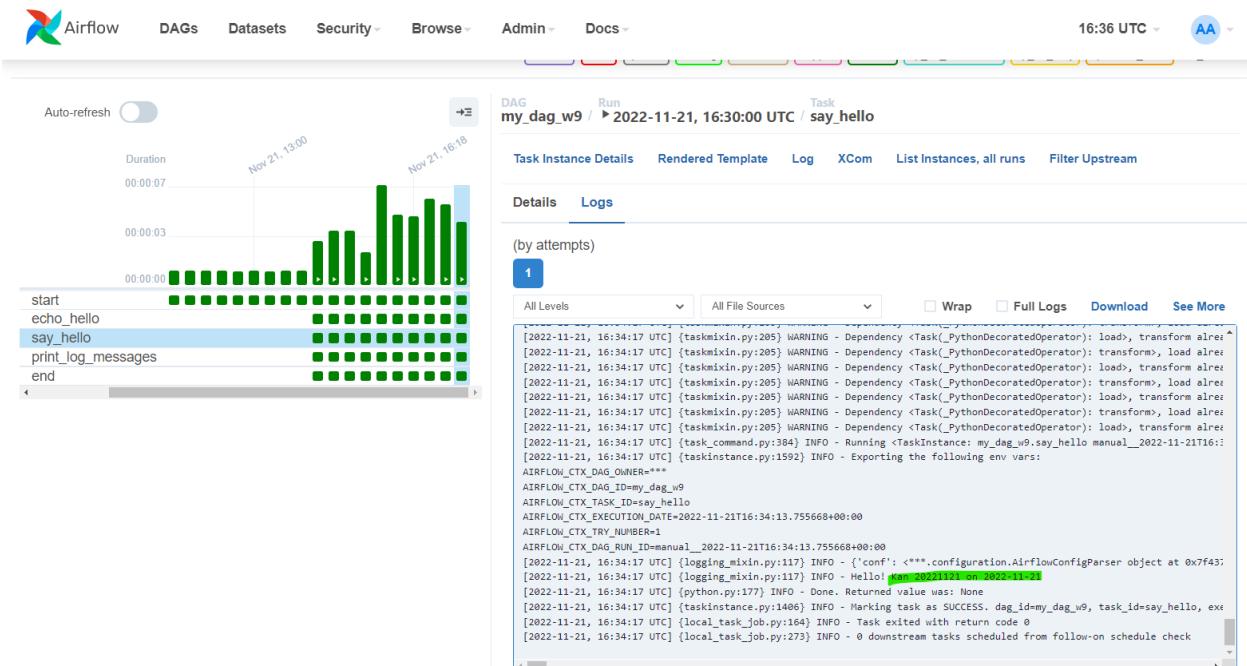
#logging မြန်တံ့ခါန log => info, debug မေးမားမြန်တံ့ခါနရှိခဲ့ခြင်း
def _print_log_messages():
    logging.info("Hello from Log")

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

    start = EmptyOperator(task_id = "start")

    echo_hello = BashOperator(
        task_id = "echo_hello",
        bash_command = "echo 'hello' on {{ ds }}",
    )

    # op_kwargs သည်ပုံစံများဖြစ်သော dict
    say_hello = PythonOperator(
        task_id = "say_hello",
        python_callable = _say_hello,
        op_kwargs = {
            "name": f"Kan {{ ds_nodash }}",
        }
    )
```



```

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

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

CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES

a85f837c40cc apache/airflow:2.4.1 "/usr/bin/dumb-init ..." 3 hours ago Up 3
hours (healthy) 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
05-creating-and-scheduling-data-pipelines-airflow-webserver-1

109ae6d1e50c apache/airflow:2.4.1 "/usr/bin/dumb-init ..." 3 hours ago Up 3
hours (healthy) 8080/tcp
05-creating-and-scheduling-data-pipelines-airflow-triggerer-1

26bbda0b7c3a apache/airflow:2.4.1 "/usr/bin/dumb-init ..." 3 hours ago Up 3
hours (healthy) 8080/tcp
05-creating-and-scheduling-data-pipelines-airflow-scheduler-1

65acd4300294 adminer:4.8.1 "entrypoint.sh docke..." 3 hours ago Up 3
hours 0.0.0.0:8088->8080/tcp, :::8088->8080/tcp
05-creating-and-scheduling-data-pipelines-adminer-1

2e74f59056c5 postgres:13 "docker-entrypoint.s..." 3 hours ago Up 3
hours 5432/tcp
05-creating-and-scheduling-data-pipelines-warehouse-1

b7b0d86547cd postgres:13 "docker-entrypoint.s..." 3 hours ago Up 3
hours (healthy) 5432/tcp
05-creating-and-scheduling-data-pipelines-postgres-1

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

05-creating-and-scheduling-data-pipelines-adminer-1
05-creating-and-scheduling-data-pipelines-airflow-triggerer-1

05-creating-and-scheduling-data-pipelines-airflow-scheduler-1
05-creating-and-scheduling-data-pipelines-airflow-webserver-1

```

```
gitpod /workspace/swu-ds525/05-creating-and-scheduling-data-pipelines (main) $  
docker exec 05-creating-and-scheduling-data-pipelines-airflow-scheduler-1 bash
```

```
default@26bbda0b7c3a:/opt/airflow$ airflow dags
```

```
default@26bbda0b7c3a:/opt/airflow$ airflow dags list
```

```
default@26bbda0b7c3a:/opt/airflow$ airflow dags backfill -s 2022-11-15 -e  
2022-11-22 test_xcom
```

These tasks are running:

DAG ID	Task ID	Run ID	Try number
-----	-----	-----	-----

These tasks have failed:

DAG ID	Task ID	Run ID	Try number
-----	-----	-----	-----

These tasks are skipped:

DAG ID	Task ID	Run ID	Try number
-----	-----	-----	-----

These tasks are deadlocked:

DAG ID	Task ID	Run ID	Try number
-----	-----	-----	-----
test_xcom	pull	backfill__2022-11-22T00:00:00+00:00	1
test_xcom	push	backfill__2022-11-22T00:00:00+00:00	1