

# Airflow를 이용한 데이터 Workflow 관리

---



# About me



dba.kim@gmail.com

## YoungHeon (Roy) Kim

DBA - Riot Games

대한민국 | 인터넷

현재 Riot Games

미전 NAVER Business platform, LG CNS Co., Ltd.

### 경력 사항



DBA

Riot Games

2015년 5월 - 현재 · (3년 9개월)



차장

NAVER Business platform

2007년 4월 - 2015년 5월 · (8년 2개월)



대리

LG CNS Co., Ltd.

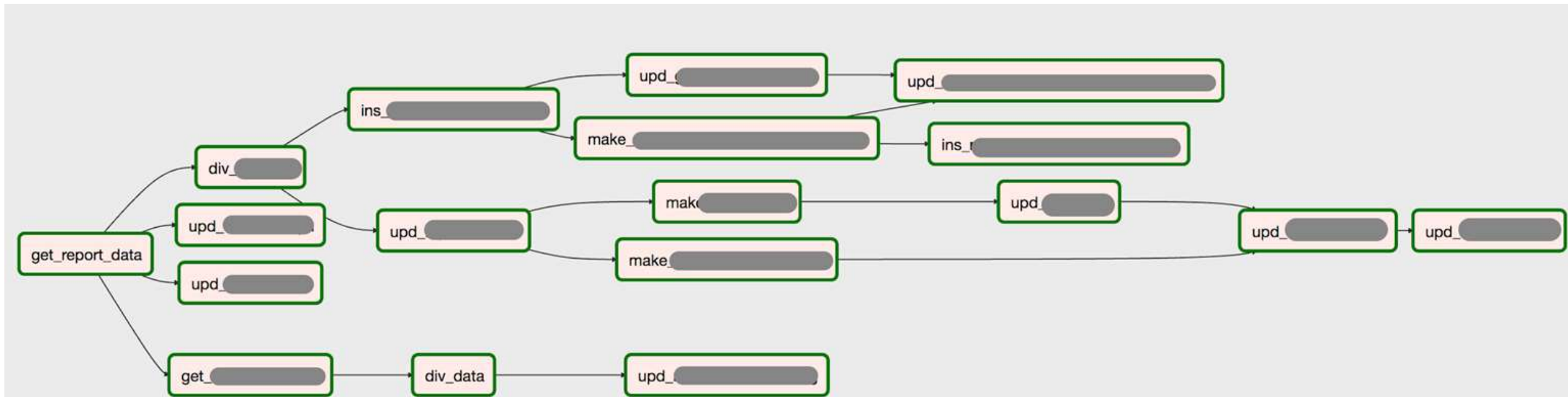
2001년 2월 - 2007년 4월 · (6년 3개월)

# Airflow

Workflow 관리를 위한 플랫폼  
airbnb에서 개발  
현재 아파치 incubating project

# 왜 Airflow 인가?

데이터 처리를 위한 복잡한 단계를 flow diagram 형태로 한눈에 볼 수 있다.



# 왜 Airflow 인가?

Python 기반으로 비교적 쉽게 각 task를 작성할 수 있다.

## dags/task1.py

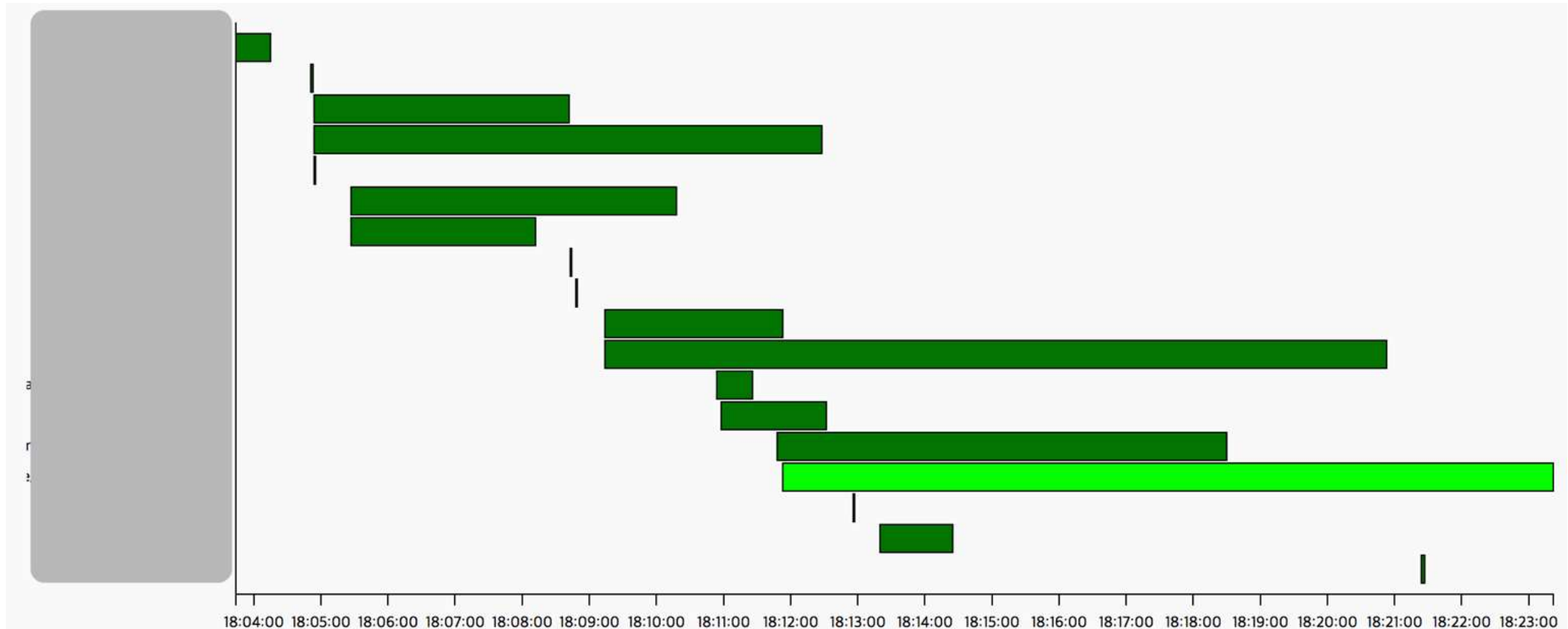
```
t1= PythonOperator(  
    task_id='get_report_data',  
    python_callable=get_report_data,  
    provide_context=True,  
    op_kwargs={'target_day':target_day},  
    dag=dag  
)
```

## scripts/task1\_lib.py

```
def get_report_data(**kwargs):  
    target_day=kwargs['target_day']  
    stmt= """SELECT ~~~~""" %(target_day)  
    a_cursor.execute(stmt)
```

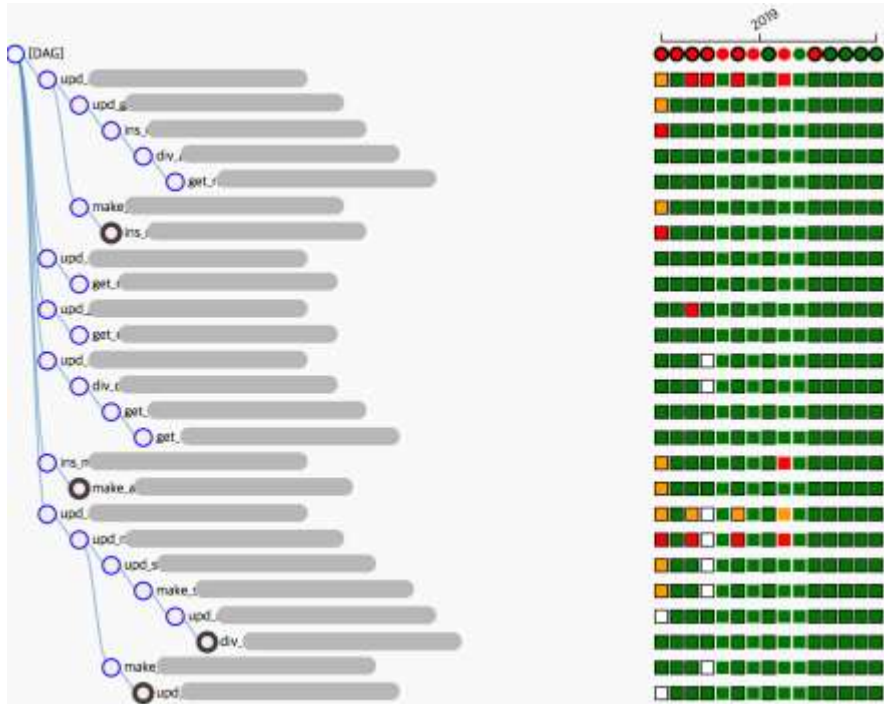
# 왜 Airflow 인가?

각 task의 실행 시간을 한눈에 볼 수 있다.









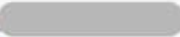






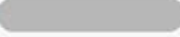










# 왜 Airflow 인가?

각 task의 실행 이력을 한눈에 볼 수 있고  
필요한 경우 특정 task만 실행할 수 있다.



# 왜 Airflow 인가?

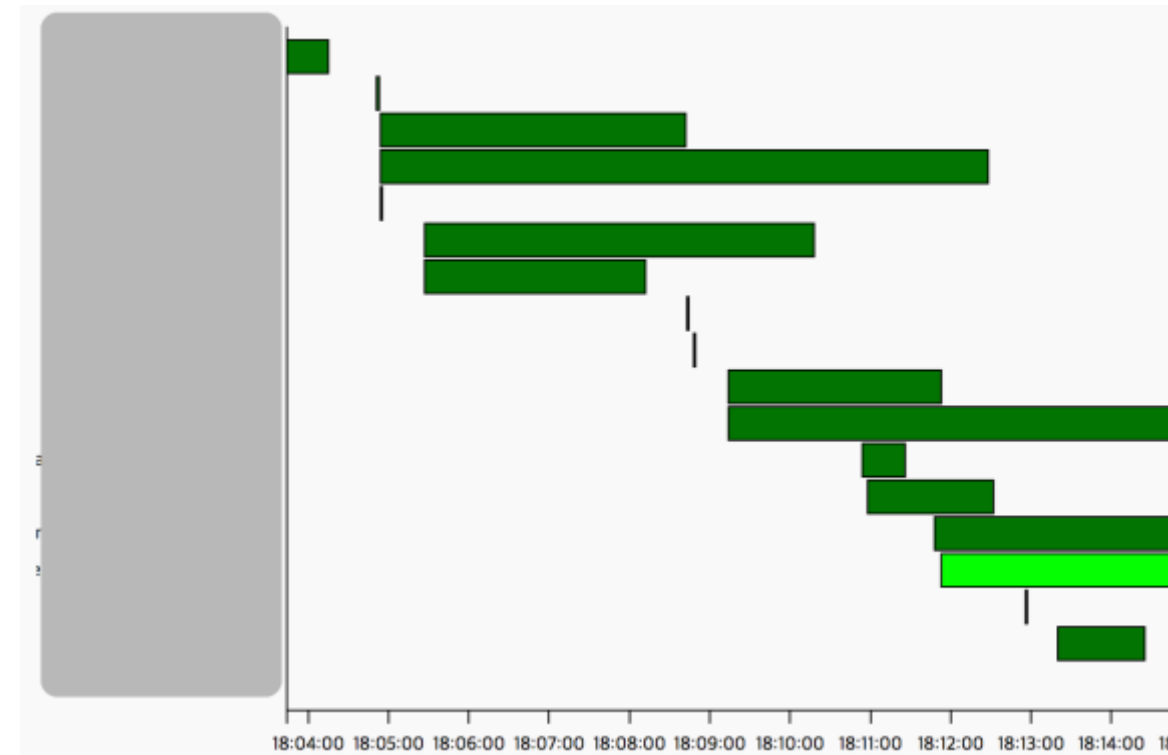
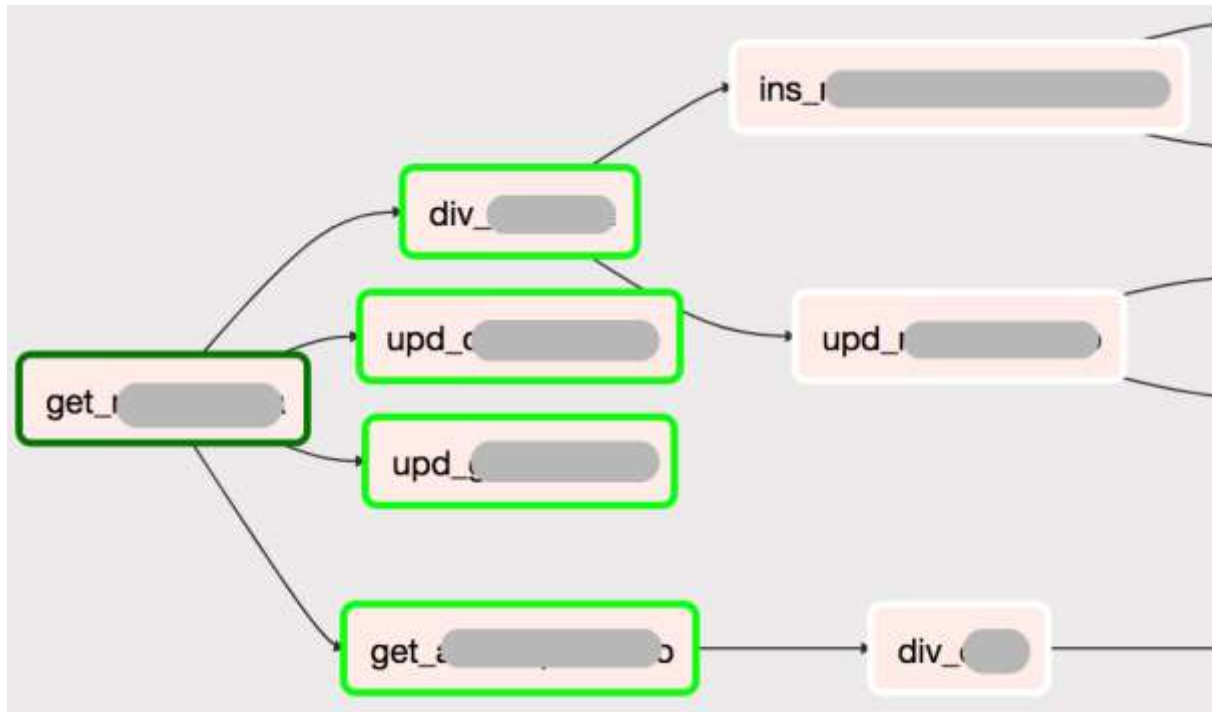
쉬운 job 스케줄 관리 및 전체 job의 실행  
상황을 볼 수 있다.

		DAG	Schedule	Owner	Recent Tasks 	Last Run 	DAG Runs 	Links
		auto_ 	00 04 ***	airflow		2019-01-13 19:00 		
		inten_ 	00 05 ***	airflow		2019-01-13 20:00 		
		python_test	00 01 ***	airflow		2019-01-12 16:00 		

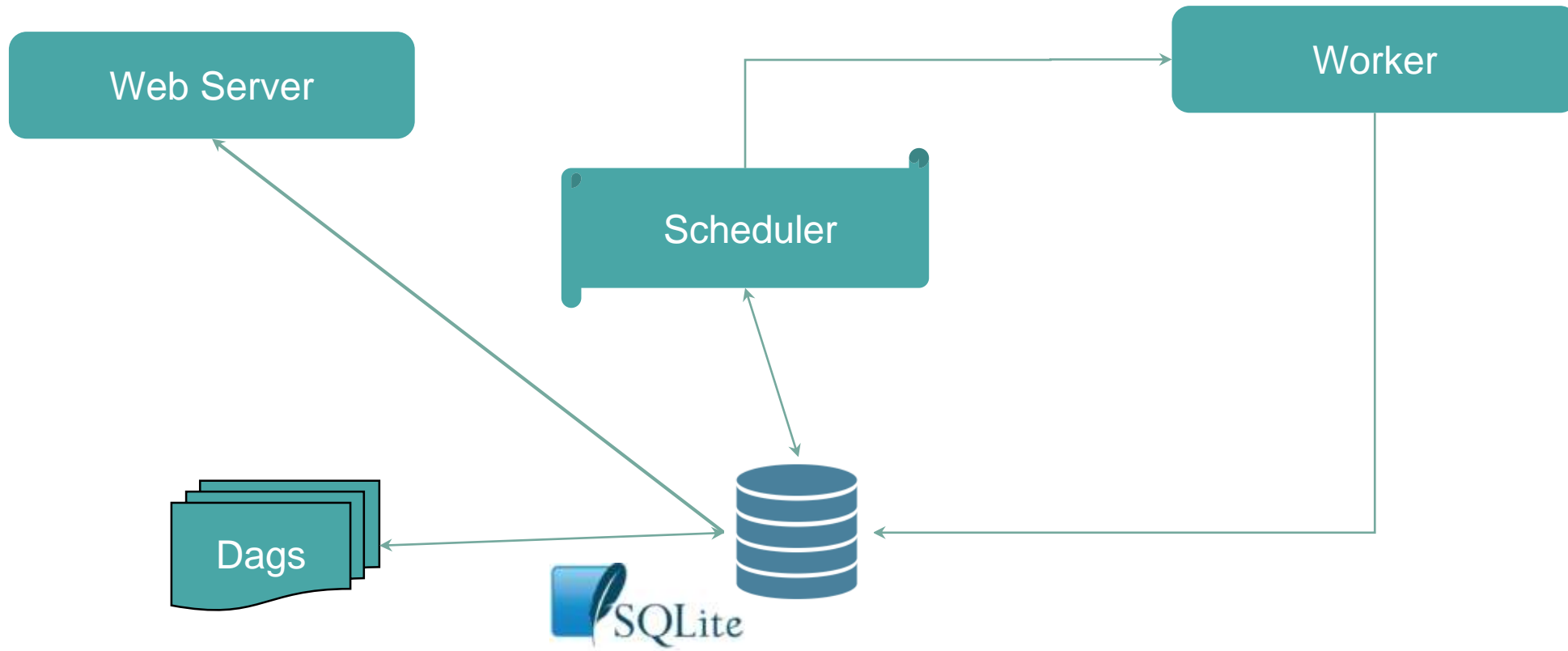


# 왜 Airflow 인가?

각 task를 병렬로 실행할 수 있다.



# Airflow 기본 아키텍처



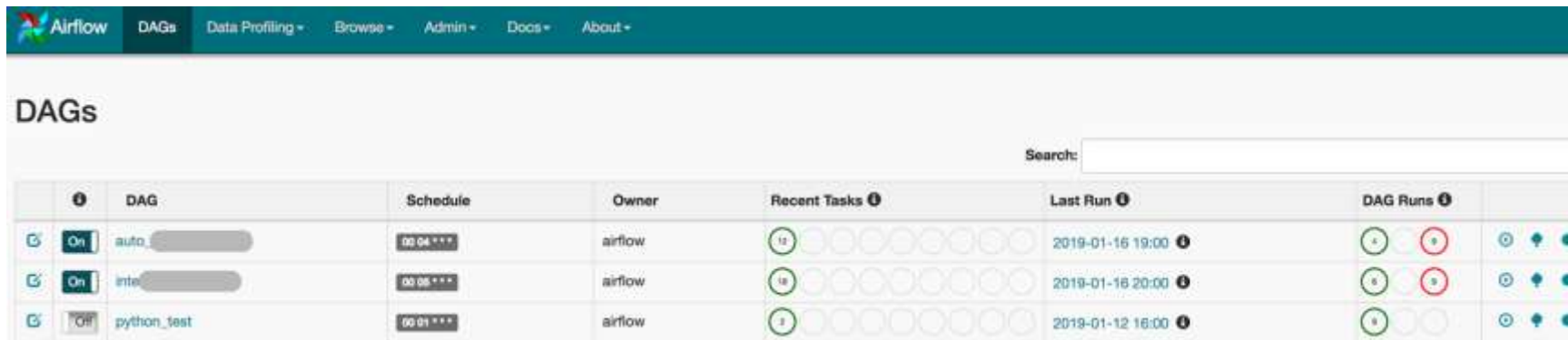
# Airflow 설치

```
#pip install apache-airflow
```

```
#airflow initdb
```

```
#airflow webserver -p 8080
```

웹 브라우저에서 접속 :<http://localhost:8080>



The screenshot shows the Apache Airflow web interface. The top navigation bar includes links for DAGs, Data Profiling, Browse, Admin, Docs, and About. The main section is titled 'DAGs' and features a search bar. Below the search bar is a table listing the DAGs. The table has columns for a status icon, DAG name, Schedule, Owner, Recent Tasks, Last Run, and DAG Runs. Three DAGs are listed: 'auto', 'inte', and 'python\_test'. The 'auto' and 'inte' DAGs are 'On' and have recent task runs, while 'python\_test' is 'Off'.

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs
	auto	00 04 ***	airflow		2019-01-16 19:00	
	inte	00 05 ***	airflow		2019-01-16 20:00	
	python_test	00 01 ***	airflow		2019-01-12 16:00	

# Dags 작성

DAG : **D**irected **A**cyclic **G**raph

Task들의 집합

여러개의 Task들이 순서와

Dependency를 갖는다

~/airflow/dags 디렉토리에 저장

# Dags 작성

## Python Operator

```
t1= PythonOperator(  
    task_id='get_report_data',  
    python_callable=get_report_data,  
    provide_context=True,  
    op_kwargs={'target_day':target_day},  
    dag=dag  
)
```

# Dags 작성

## Python Operator

Parameter	설명
task_id	각 Task를 구분하기 위한 Unique한 Task명
python_callable	실제 호출된 python 함수명
provide_context	python 함수 호출 시 해당 함수에서 사용될 수 있는 기본적인 argument값을 넘겨줄 지 여부
op_kwargs	기본 argument 외에 추가로 넘겨줄 파라미터 정의
dag	default dag 명

# Dags 작성

작성된 Dag는 변경된 사항을 반영하기 위해서 airflow의 scheduler에 의해서 주기적으로 호출되므로 파일이 너무 크거나 복잡해지지 않도록 만든다

Dag에서는 실제 처리 내용을 정의하기 보다는 처리될 순서가 정의될 수 있도록 하고 실제 처리내용은 별도 파일(lib)에서 호출될 수 있도록 했다

# Dags 작성

~/airflow/dags/**sample\_dag.py**

```
execfile('~/.airflow/scripts/sample_dag_lib.py')

t1= PythonOperator(
    task_id='get_report_data',
    python_callable=get_report_data,
    .....
)
t2= PythonOperator(
    task_id='upd_add_data',
    python_callable='upd_add_data',
    .....
)
t2.set_upstream(t1)
```

~/airflow/scripts/**sample\_dag\_**  
**lib.py**

```
def get_report_data(**kwargs):
    .....
    .....

def upd_add_data(**kwargs):
    .....
    .....
)
```



# Dags 작성(xcom)

Task 간의 데이터 전달은 xcom을 사용

*# 값을 넘겨줄 때*

```
def push_value(**kwargs):  
    val_a=1  
    ti=kwargs['ti']  
    ti.xcom_push(key='push_value1', value=val_a)
```






*# 값을 받을 때*

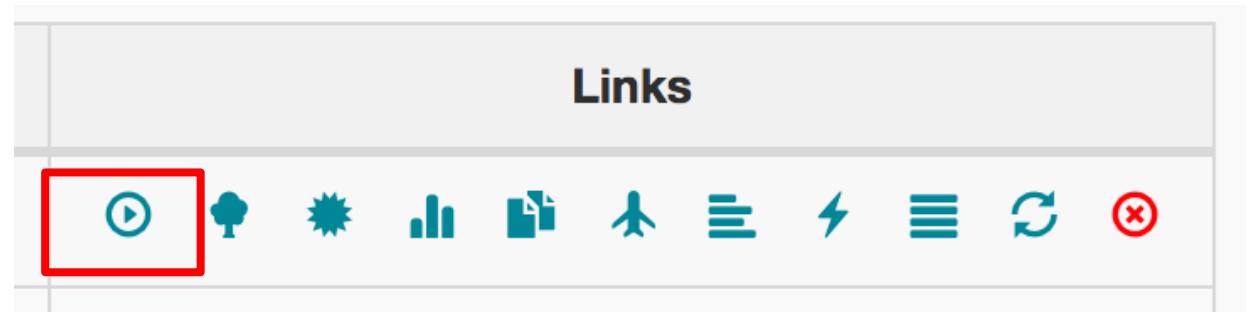
```
def get_value(**kwargs):  
    ti=kwargs['ti']  
    val_b=ti.xcom_pull(task_ids='push_value', key='push_value1')
```

# Scheduler 및 Job 실행

```
#airflow scheduler
```

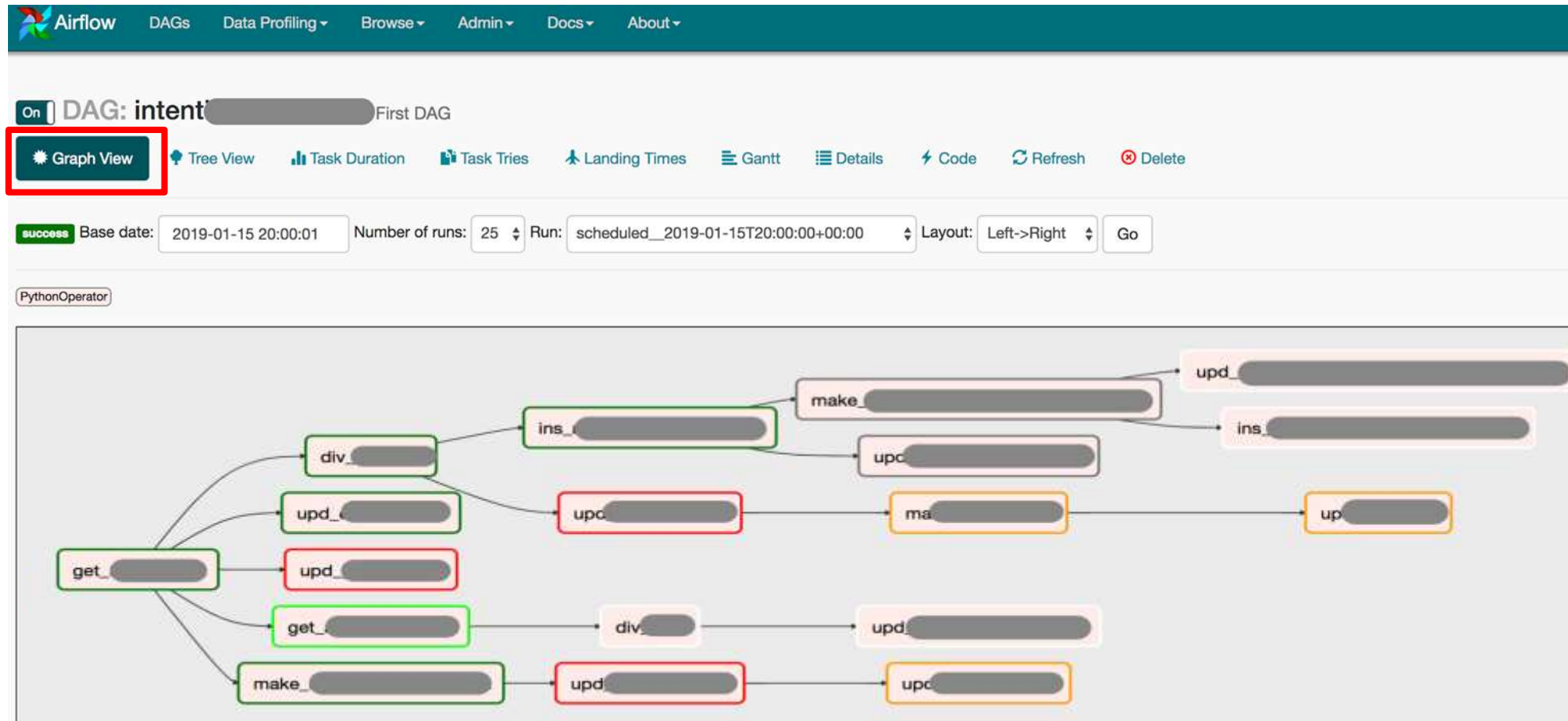
웹 페이지에서 생성된 Dag를 enable 시키고 수동으로 수행시킬 수 있다

	i	DAG
	<input checked="" type="checkbox"/>	auto_ 
	<input checked="" type="checkbox"/>	intent 
	<input type="checkbox"/>	python_test



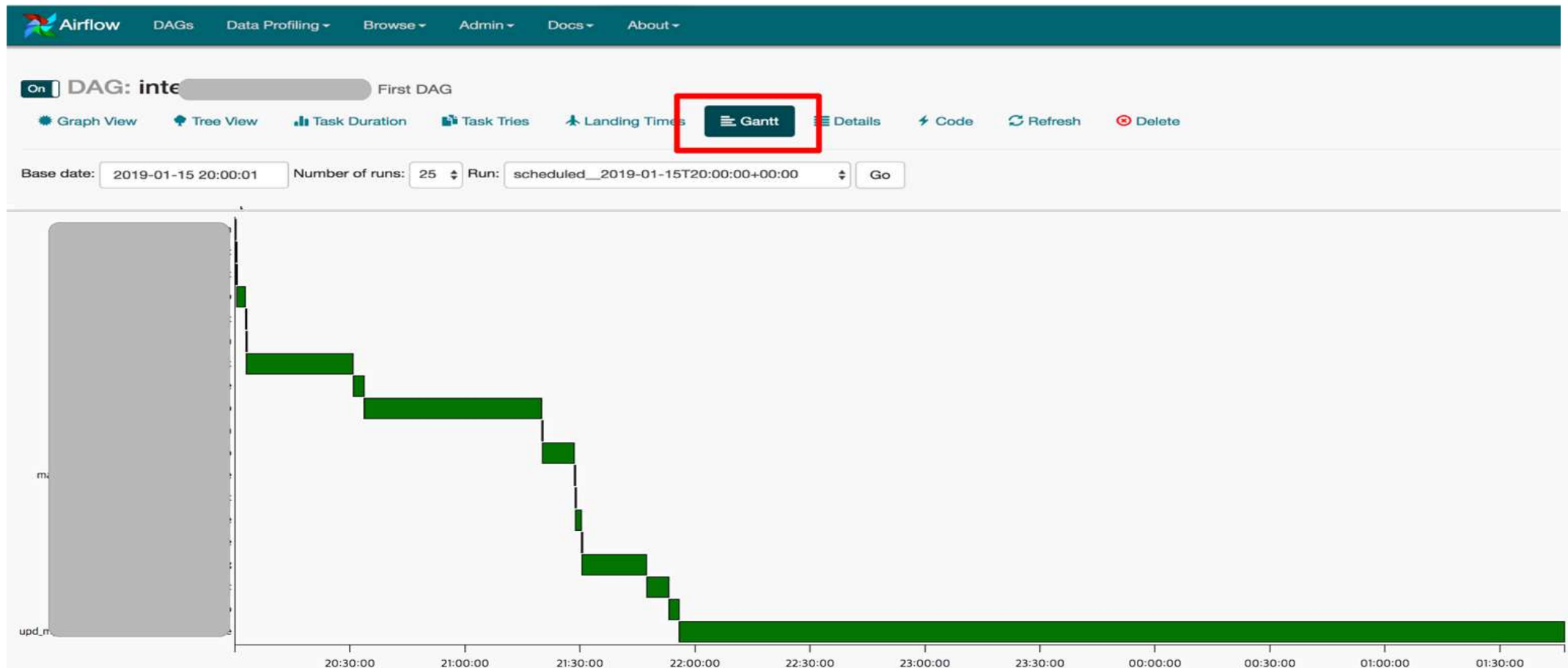
# Task 실행 상태 확인(Graph view)

수행중인 Dag를 클릭한 후 Graph View에서 실행 상태를 볼 수 있다



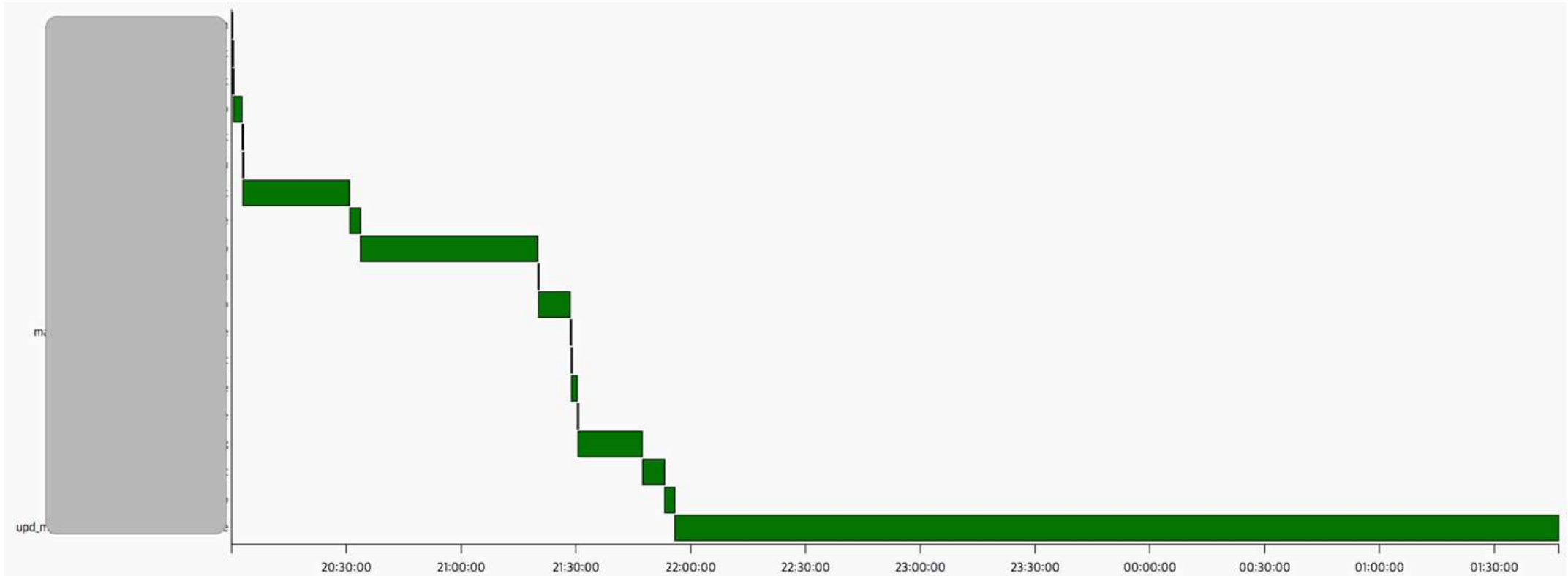
# Task 실행 상태 확인(Gant view)

Gant View에서 각 Task의 실행에 소요된 시간을 볼 수 있다



# Task의 병렬 실행

But...각 Task가 **병렬로 실행되지 않고 Sequential하게 실행되는** 것을 볼 수 있다

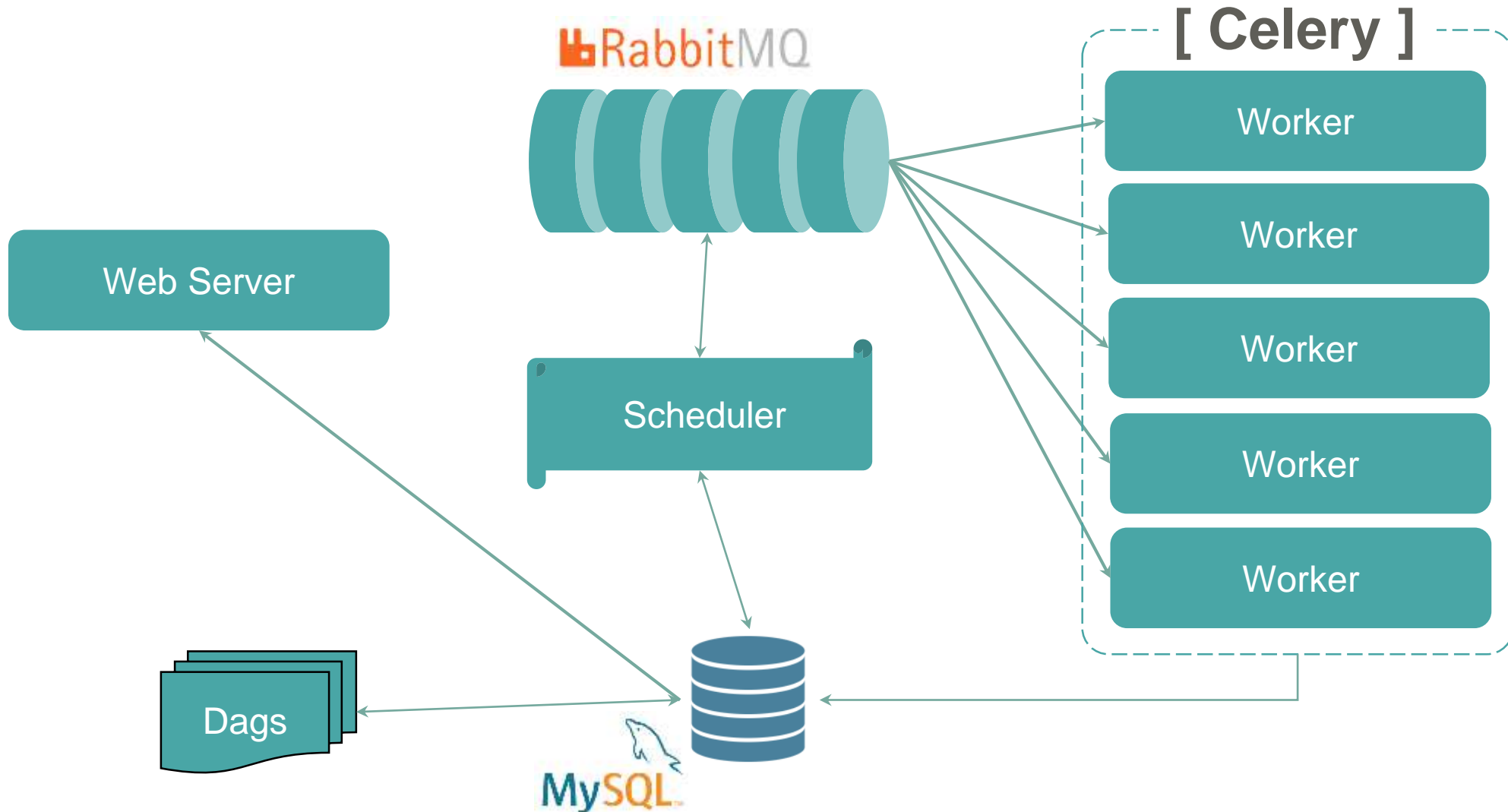


# Task의 병렬 실행

Task를 병렬로 실행하려면 아래 조건이 필요하다

- 실제 Task를 실행하는 airflow의 worker를 sequential executor가 아닌 **celery executor**를 사용해야 함
- celery executor 사용을 위해서는 Broker 가 필요한데 이를 위해 **RabbitMQ**나 **Redis**가 필요함
- airflow의 meta store로 sqlite가 아닌 **mysql**이나 **postgresql**을 사용해야 함

# 병렬 실행을 위한 Airflow 아키텍처



# MySQL 설치 및 설정

```
# yum install mysql.x86_64
```

```
mysql> create database airflow;
```

```
mysql> create user airflow@'10.xx.xx.xx' identified by 'airflow_pwd';
```

```
mysql> grant ALL PRIVILEGES ON airflow.* to airflow@'10.xx.xx.xx';
```

## my.cnf에 아래 설정 추가

```
explicit_defaults_for_timestamp = 1
```

```
max_allowed_packet = 30M
```



# MySQL 설치 및 설정

## MySQL start 및 airflow db 초기화

```
# /etc/init.d/mysql start  
# airflow initdb
```

## xcom으로 전달되는 데이터의 크기 확장을 위해 컬럼 타입 변경

```
mysql> alter table airflow.xcom modify value LONGBLOB;
```

# RabbitMQ 설치 및 설정

```
# yum install rabbitmq-server.noarch
# rabbitmq-server
# rabbitmqctl add_user airflow airflow
Creating user "airflow" ...
...done.
# rabbitmqctl add_vhost airflow_vhost
Creating vhost "airflow_vhost" ...
...done.
# rabbitmqctl set_user_tags airflow airflow
Setting tags for user "airflow" to [airflow] ...
...done.
# rabbitmqctl set_permissions -p airflow_vhost airflow ".*" ".*" ".*"
Setting permissions for user "airflow" in vhost "airflow_vhost" ...
...done.
```

# airflow.cfg 설정

```
# vi ~/airflow/airflow.cfg
```

```
# executor를 default인 SequentialExecutor에서 CeleryExecutor로 변경  
executor = CeleryExecutor
```

```
# db connection을 sqlite에서 mysql로 변경  
sql_alchemy_conn = mysql://airflow:airflow@10.xx.xx.xx/airflow
```

```
# Celery broker URL을 rabbitmq로 설정  
broker_url = amqp://airflow:airflow@10.xx.xx.xx:5672/airflow_vhost
```

```
# Celery가 job을 수행한 결과를 저장한 db metastore 설정  
celery_result_backend = db+mysql://airflow:airflow@10.xx.xx.xx:3306/airflow
```

# airflow restart

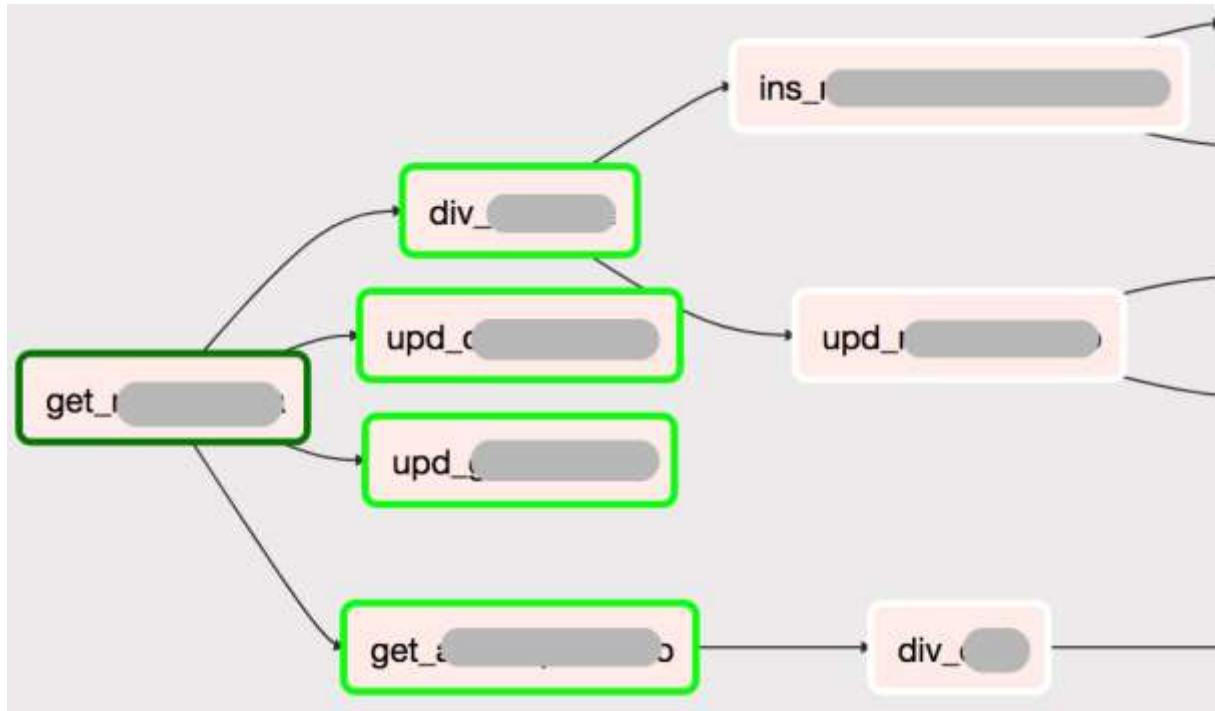
```
# airflow webserver -p 8080
```

```
-- celeryExecutor는 별도 worker 실행이 필요하다
```

```
# airflow worker
```

```
# airflow scheduler
```

# Task 병렬 실행 확인



# But.. 병렬 실행으로 인한 경합 발생

## Sequential 실행 시에 발생하지 않던 DB Lock 발생

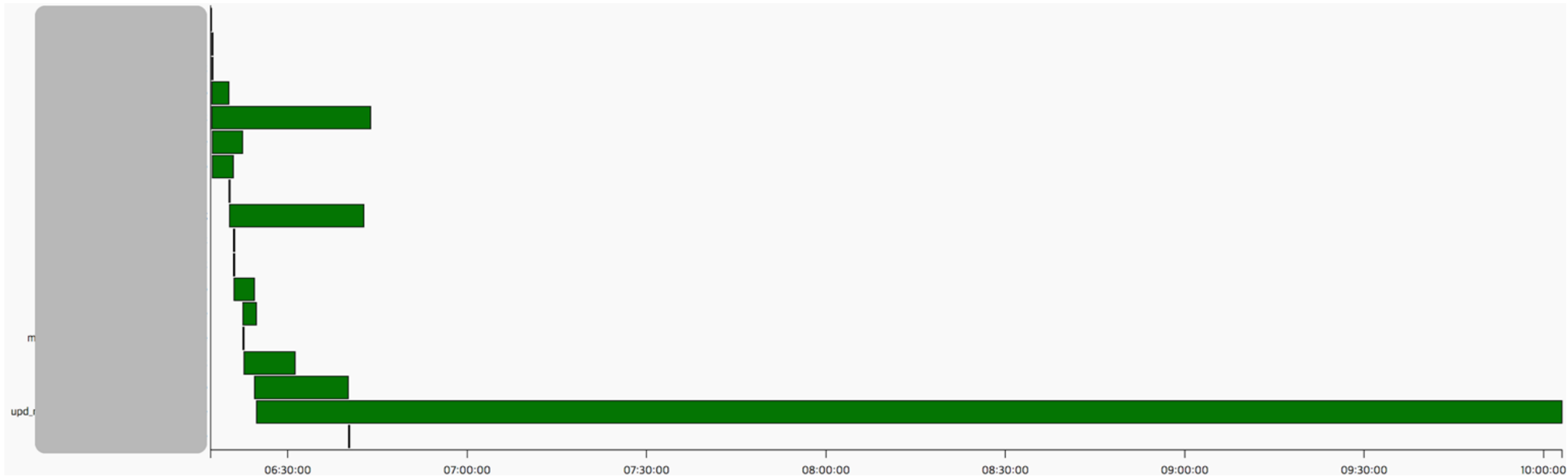
```
[2019-01-07 11:14:47,723] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,723] {base_task_runner.py:95} INFO - Subtask: [2019-01-07 11:14:47,682] {models.py:1417} ERROR - (1205, 'Lock wait timeout
[2019-01-07 11:14:47,724] {base_task_runner.py:95} INFO - Subtask: Traceback (most recent call last):
[2019-01-07 11:14:47,724] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,724] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,724] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,725] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,725] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,725] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,725] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,725] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,726] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,726] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,726] {base_task_runner.py:95} INFO - Subtask:
[2019-01-07 11:14:47,727] {base_task_runner.py:95} INFO - Subtask: OperationalError: (1205, 'Lock wait timeout exceeded; try restarting transac
[2019-01-07 11:14:47,727] {base_task_runner.py:95} INFO - Subtask: [2019-01-07 11:14:47,689] {models.py:1441} INFO - Marking task as FAILED.
[2019-01-07 11:14:47,799] {base_task_runner.py:95} INFO - Subtask: [2019-01-07 11:14:47,798] {models.py:1462} ERROR - (1205, 'Lock wait timeout
[2019-01-07 11:14:47,799] {base_task_runner.py:95} INFO - Subtask: Traceback (most recent call last):
```

해당 작업의 특성 상 특정 일자 데이터 범위를 조회/추가/변경 하는 로직으로 동일 테이블의 Task를 병렬로 수행 시 Lock경합이 발생함

각 Task의 실행 순서를 조정해서 Lock 문제 해결

# Task 병렬 수행 및 경합 해소 확인

Gant view에서 각 Task가 병렬로 수행됨을 확인할 수 있다



# Airflow 적용 후 개선점

- 데이터 처리 **Flow**를 쉽게 파악할 수 있다
- 데이터 처리를 위한 **배치**를 통합해서 관리할 수 있다
- 웹페이지를 통해 매일 **배치 수행 현황**을 쉽게 확인할 수 있다
- 배치 실패 시에 특정 Task만 별도로 수행함으로써 **문제를 쉽게 해결**할 수 있다



Thank you