

데이터사이언스를 위한 지식관리시스템

14주차: Airflow(Workflow Management)

Airflow

- 워크플로우 관리도구 입니다
- 데이터 파이프라인, 배치 작업, ETL 등 반복적/복잡한 작업 흐름 관리
- 하나의 워크플로우를 DAG라는 것으로 정의하여 관리합니다
- 스케줄링 기반의 트리거, 온디맨드 트리거, 이벤트 기반의 트리거를 지원합니다
- Airflow의 히스토리: <https://airflow.apache.org/docs/apache-airflow/stable/project.html>
- 2025년 현재의 Airflow는 3.x 입니다
- Airflow 3.0이 2025년에 릴리즈 되면서 UI에 많은 변화가 있었고 향상된 사용자 경험을 제공하게 됩니다

DAG

- 방향성이 있고 순환이 없는 그래프입니다
- 즉, 작업의 시작과 끝이 있는 워크플로우를 의미합니다
- 이것이 Airflow에서는 하나의 실행 단위가 됩니다
- DAG 내에서는 task들이 있고 이들이 방향성을 갖고 연결됩니다
- 그 연결에는 순환이 없어야 합니다

Airflow Vs Cron

- Cron이라는 스케줄러를 이미 배웠는데 Airflow를 다시 하는 이유입니다
- Cron은 스케줄이 되면 무조건 실행합니다 -> 중복/충돌 가능합니다
- 이를 방지하기 위해 pid파일을 남기는 방식도 고려해 보았지만 빈틈은 있습니다
- Airflow는 DAG + Scheduler로 Task 순서를 보장, 중복 방지하는 메커니즘을 제공합니다

Airflow의 설치

- Airflow 3.1.3은 pip를 이용하여 간편하게 설치할 수 있습니다
- <https://airflow.apache.org/docs/apache-airflow/stable/installation/installing-from-pypi.html> 를 참고하여 설치합니다
- WSL 또는 클라우드 환경에서 사용 가능하며 개별 인스턴스로 설치, 구동이 가능하니 개별 계정으로 모두 개별 설치를 진행해보겠습니다
- 설치 후 airflow 라고 실행해봅니다
- 오른쪽과 같이 나오면 설치가 된 것입니다

```
Usage: airflow [-h] GROUP_OR_COMMAND ...

Positional Arguments:
  GROUP_OR_COMMAND

Groups
  assets          Manage assets
  backfill        Manage backfills
  config          View configuration
  connections     Manage connections
  dags            Manage DAGs
  db              Database operations
  db-manager      Manage externally connected database managers
  jobs            Manage jobs
  pools           Manage pools
  providers       Display providers
  tasks           Manage tasks
  variables       Manage variables

Commands:
  api-server      Start an Airflow API server instance
  cheat-sheet     Display cheat sheet
  dag-processor   Start a dag processor instance
  info            Show information about current Airflow and environment
  kerberos        Start a kerberos ticket renewer
```

Airflow의 프로세스 구성요소

- API server (실행명령: airflow api-server --port 포트번호)
 - 주의사항: api-server 구동시에 port번호를 다르게 구성하겠습니다
 - 충돌 방지를 위해 port번호는 user계정의 번호와 동일하게 하겠습니다

```
(base) ubuntu@uos-bigdata:~$ airflow api-server --port 8022
Please confirm database initialize (or wait 4 seconds to skip it). Are you sure? [y/N]
2025-12-05T22:07:30.961704Z [info      ] Log template table does not exist (added in 2.3.
-----
-----|_|( )-----/---/-----| /| / / /
-----/|_|/_/-----/---/-----/ \ | /| / /
-----/ /| / / / /-----/ / / / / / | /| / /
-----/ /| / / / /-----/ / / / / / / | /| / /
2025-12-05T22:07:30.974485Z [info      ] Running the uvicorn with:
Apps: all
Workers: 1
Host: 0.0.0.0:8022
Timeout: 120
Logfiles: -
```

Port	Forwarded Address
● 3306	localhost:3306
● 5432	localhost:5432
○ 7474	localhost:7474
○ 7687	localhost:7687
○ 8022	localhost:8022

터널링도 잊지 않고 해줍니다

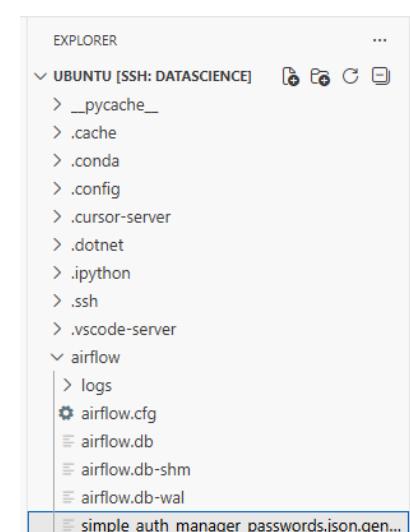
Airflow의 프로세스 구성요소

- DAG processor (실행명령: airflow dag-processor)
 - Scheduler (실행명령: 아래 실행명령에서 복사하여 포트번호를 바꿉니다)
 - Scheduler의 포트번호는 각자 할당된 번호로 뒤의 2자리를 바꿉니다
 - 주의사항: foreground모드로 돌기 때문에 터미널을 별도로 띄워서 진행합니다

```
export AIRFLOW_LOGGING_WORKER_LOG_SERVER_PORT=8722  
airflow scheduler
```

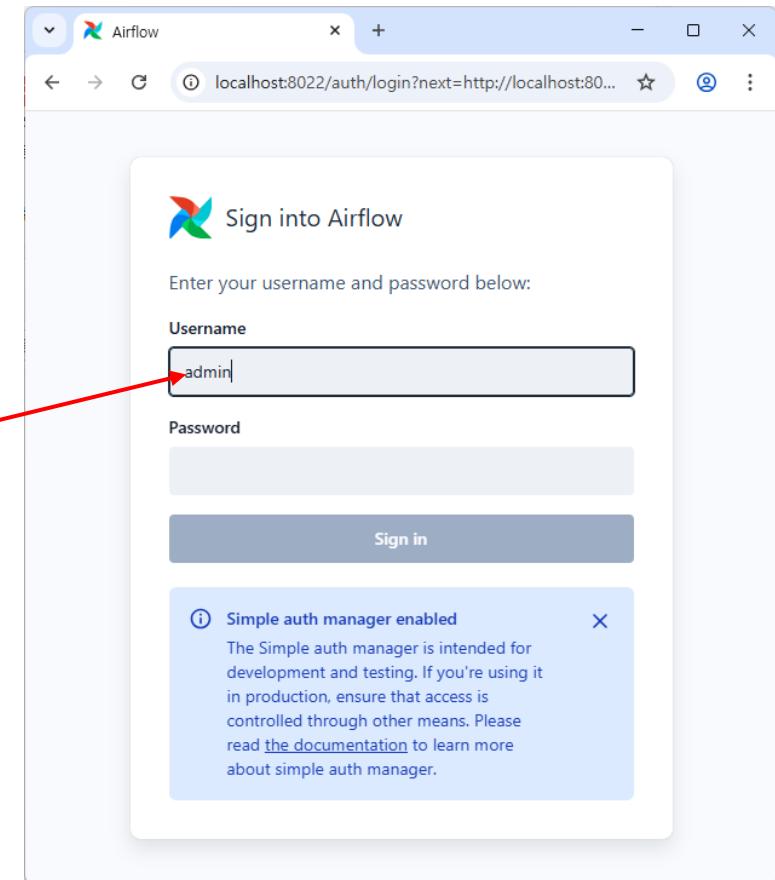
Airflow의 사용자 UI 접속

- 기본 8080 포트를 사용하나 각자 변경하신 포트로 접속합니다
- `http://localhost:<각자의포트번호>`
- 사용자명은 admin 입니다
- 패스워드는 아래의 경로에서 확인할 수 있습니다
- 이제부터는 별도 첨부된 ipynb파일을 사용합니다
- 각 코드셀에 있는 내용을 py 파일로 만들면서 DAG를 만들 것입니다



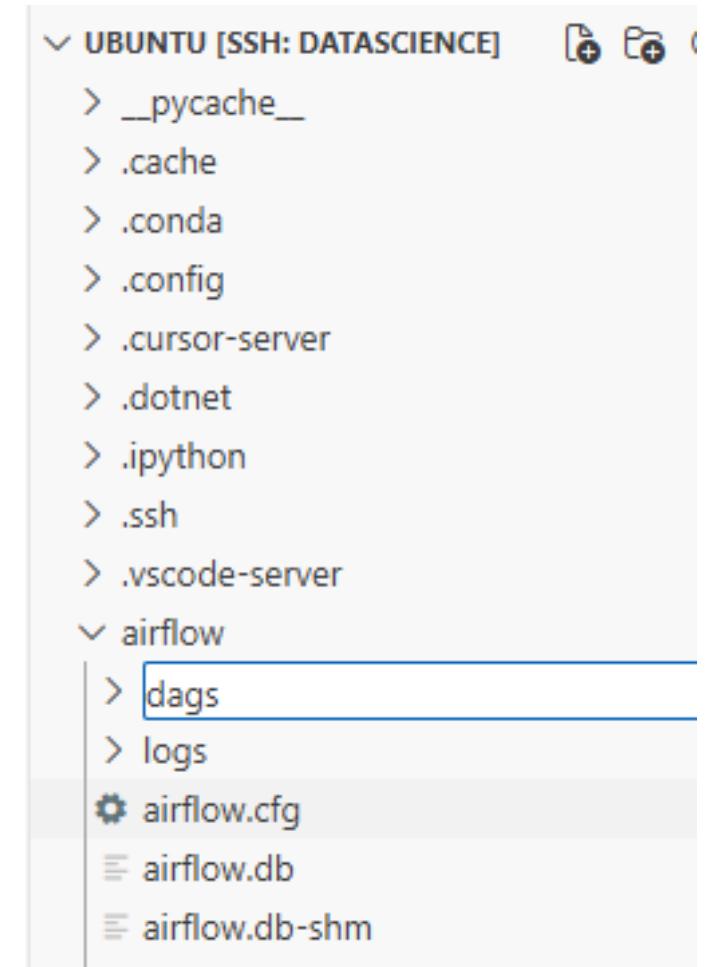
The screenshot shows a Jupyter Notebook interface with an 'EXPLORER' sidebar on the left. Under the 'UBUNTU [SSH: DATASCIENCE]' section, there is a tree view of files and directories. One of the files listed is 'simple_auth_manager_passwords.json.generated'. A red arrow points from this file name to the corresponding line in the code editor.

```
simple_auth_manager_passwords.json.generated
airflow > simple_auth_manager_passwords.json.get
1 {"admin": "Kg8sE3XSnHWZKHug"}
```



DAG 등록

- 1개의 task로 등록된 DAG를 등록해보겠습니다
- 샘플코드는 Airflow UI에 연결하여 찾을 수 있습니다
- 우선 py파일을 DAG로 등록하는 위치는 오른쪽과 같습니다
- airflow라는 디렉토리는 만들어졌을 것입니다
- 그 아래에 dags라는 디렉토리를 만듭니다
- dags 디렉토리에 py파일을 만들 것입니다
- 이제부터 그 py파일을 만드는 방법을 알아보겠습니다



Airflow의 Sample DAG들

- 아래와 같이 Dags탭을 확인합니다
- 80개의 샘플 DAG들이 등록되어 있습니다
- 하나를 클릭하면 아래 오른쪽과 같이 DAG의 상세 화면을 볼 수 있고 코드 탭이 있습니다

Dags

80 Dags

Params Trigger UI params, example

Params UI tutorial params, ui, example

```
from pathlib import Path
from airflow.sdk import DAG, Param, TriggerRule,
# [START params_trigger]
with DAG(
    dag_id=Path(__file__).stem,
    dag_display_name="Params Trigger UI",
    description=__doc__.partition(".")[0],
    doc_md=__doc__,
    schedule=None,
    start_date=datetime.datetime(2022, 3, 4),
    catchup=False,
    tags=["example", "params"],
    params={
        "names": Param(
            f"India", "Martha", "Thomas"
        )
    }
) as dag:
```

Operator

- Dags탭에서 example_python_operator 를 검색해보겠습니다
- 그리고 해당 DAG의 코드탭에 진입합니다
- 아래 그림을 보고 코드를 따라 타이핑 하시는 것보다는 UI에 있는 코드 템을 참조하여 아래 그림과 같은 코드가 있는 부분을 찾아봅니다

```
with DAG(
    dag_id="example_python_operator",
    schedule=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:
    # [START howto_operator_python]
    def print_context(ds=None, **kwargs):
        """Print the Airflow context and ds variable from the context."""
        print("::group::All kwargs")
        pprint(kwargs)
        print("::endgroup::")
        print("::group::Context variable ds")
        print(ds)
        print("::endgroup::")
        return "Whatever you return gets printed in the logs"

    run_this = PythonOperator(task_id="print_the_context", python_callable=print_context)
    # [END howto_operator_python]
```

task

- 이렇게 PythonOperator들로 선언된 변수가 있습니다
- 이들을 task라고 합니다
- 각 task를 >> 로 연결합니다

```
run_this = PythonOperator(task_id="print_the_context", python_callable=print_context)
# [END howto_operator_python]

# [START howto_operator_python_render_sql]
def log_sql(**kwargs):
    log.info("Python task decorator query: %s", str(kwargs["templates_dict"]["query"]))

log_the_sql = PythonOperator(
    task_id="log_sql_query",
    python_callable=log_sql,
    templates_dict={"query": "sql/sample.sql"},
    templates_exts=[".sql"],
)
# [END howto_operator_python_render_sql]

# [START howto_operator_python_kwargs]
# Generate 5 sleeping tasks, sleeping from 0.0 to 0.4 seconds respectively
def my_sleeping_function(random_base):
    """This is a function that will run within the DAG execution"""
    time.sleep(random_base)

for i in range(5):
    sleeping_task = PythonOperator(
        task_id=f"sleep_for_{i}", python_callable=my_sleeping_function, op_kwargs={"random_base": i / 10}
    )

run_this >> log_the_sql >> sleeping_task
```

The code snippet illustrates the declaration of three PythonOperator objects: `run_this`, `log_the_sql`, and `sleeping_task`. A red box highlights the first line of each operator's definition. A large red rectangle encloses the entire block of code. Red arrows point from the highlighted lines to the corresponding operator definitions and then to the sequence of connections at the bottom.