Airflow 소개

데이터 파이프라인

- 일반적인 데이터 파이프라인은 원하는 결과를 얻기 위해 실행되는 여러 태스크 또는 동작으로 구성

데이터 파이프라인 예

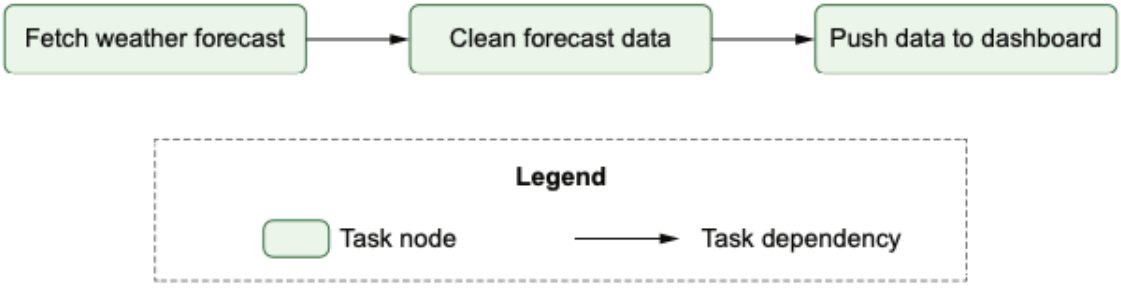
날씨 대시보드

1. 먼저 다른 시스템의 날씨 API를 통해 일기 예보 데이터를 가져옴
2. 서비스 목적에 맞도록 데이터를 정제하거나 변환(예: 온도를 화씨에서 섭씨로 변환)
3. 변환된 데이터를 날씨 대시보드로 전송

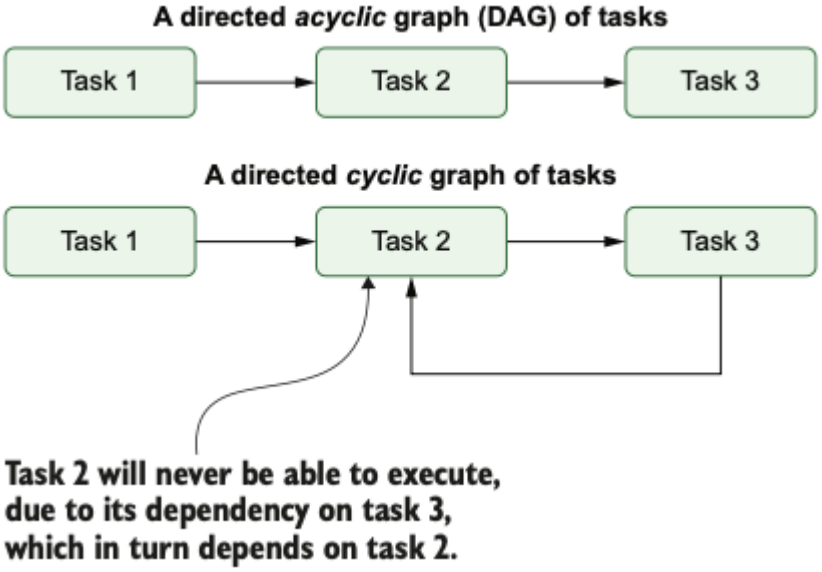
- 위의 파이프라인은 단순하지만 서로 다른 태스크로 구성되어 있음
- 그리고 각 태스크는 정해진 순서대로 진행되어야 함
- 만약 데이터를 가져오기 전에 변환을 시도한다면 의미가 없고, 데이터 변환 전에 새로운 데이터를 대시보드로 전송하면 안됨
- 따라서 정해진 순서대로 태스크를 적용해야 하는지 확인해야 함

데이터 파이프라인 그래프

- task간의 의존성을 명확하게 확인하는 방법 중 하나는, 데이터 파이프라인을 그래프로 표현하는 것
- 태스크는 노드로 표시
- 태스크 간의 의존성은 태스크 노드 간의 방향으로 표시
- 화살표의 최종 끝점은 태스크 A에서 태스크 B로 연결되고, 태스크 B가 시작되기 전에 태스크 A를 완료해야 한다는 것을 의미
- 이러한 형태의 그래프는 방향성을 가지기 때문에 방향성 그래프(directed graph)라고 함



- 이런 형태의 그래프는 일반적으로 방향성 비순환 그래프(Directed Acyclic Graph, DAG)라고 함
- 그래프는 화살표 방향성의 끝점(directed edge)을 포함하되 반복이나 순환을 허용하지 않음. 즉 비순환임
- 비순환 속성은 태스크 간의 순환 실행을 방지하기 때문에 매우 중요

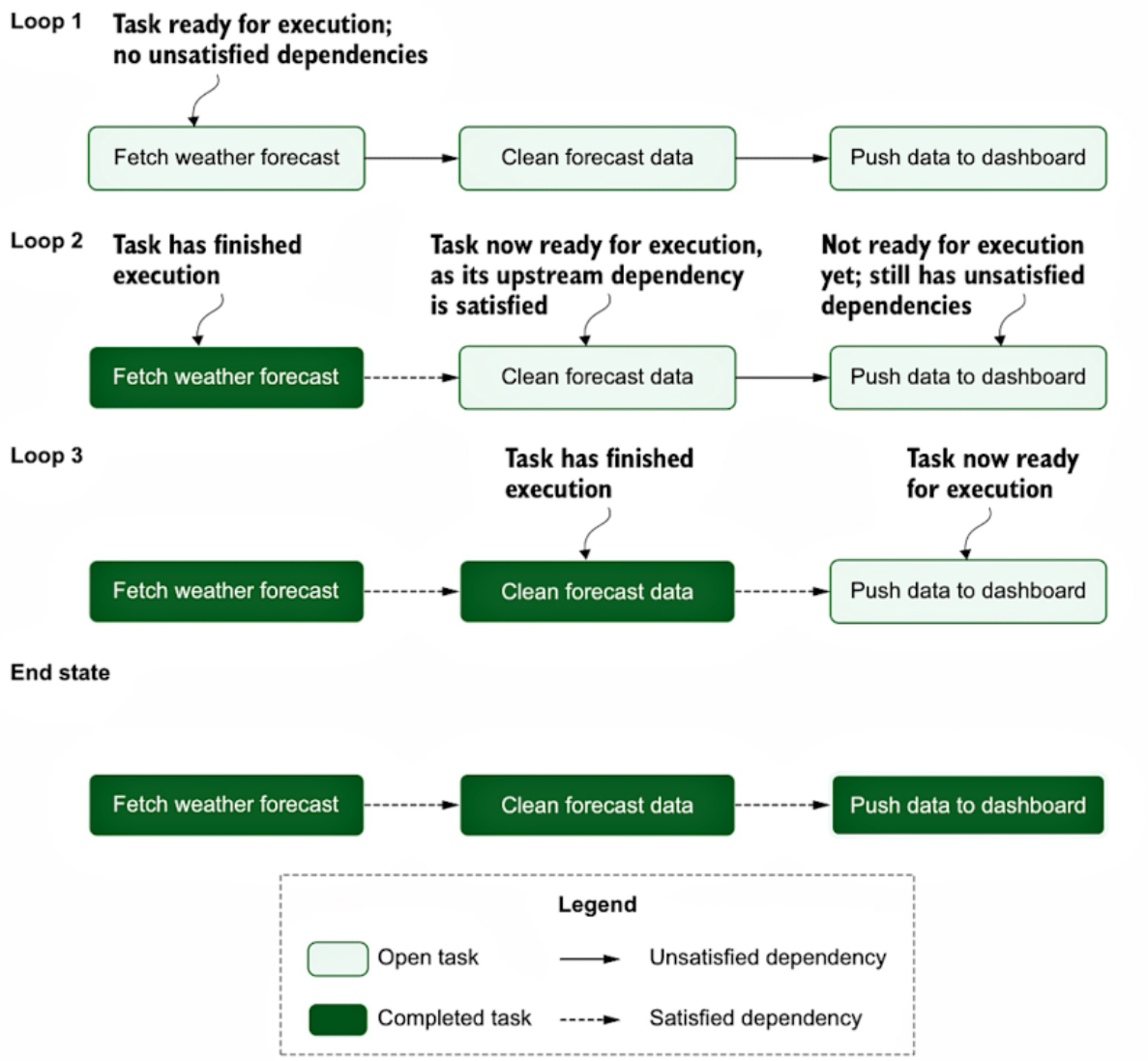


- cyclic 형태의 그림을 보면 태스크 2,3은 서로 의존하기 때문에 논리적 오류가 발생
- Airflow에서는 DAG의 비순환 속성은 태스크 그래프를 효율적으로 해결하고 실행하기 위해서 사용

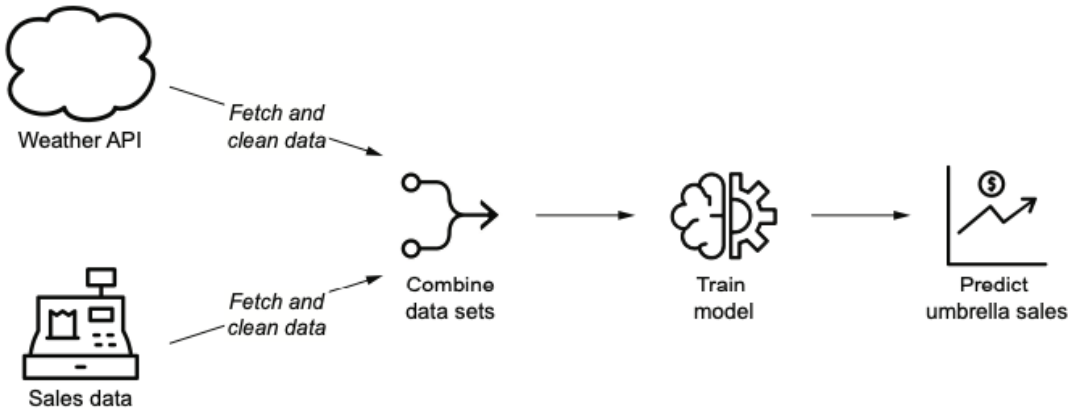
파이프라인 그래프 실행

1. 그래프 안에 태스크는 각각 open 상태(미완료)
    1. 각각의 화살표 끝점은 태스크를 향하며, 다음 태스크로 향하기 전에 이전 태스크가 완료되었는지 확인 (1단계)
    2. 태스크가 완료되면 다음에 실행해야 할 태스크를 대기열에 추가
2. 실행 대기열에 있는 태스크르 실행하고 태스크 수행이 완료되면 완료 표시

3. 그래프의 모든 태스크가 완료될 때까지 1단계로 돌아감



그래프 데이터 파이프라인의 특징



1. 판매 데이터 준비 과정
    1. 원천 시스템에서 판매 데이터 추출

    2. 요구 사항에 맞게 데이터 정제 및 변환
2. 날씨 데이터 준비 과정
    1. api로부터 날씨 데이터 가져오기
    2. 요구 사항에 맞게 데이터 정제 및 변환
3. 수요 예측 모델 생성을 위해 판매 및 날씨 데이터 세트를 결합하여 새로운 데이터 세트 생성
4. 생성된 데이터 세트를 이용해 머신러닝 모델 훈련
5. 머신러닝 모델을 배포하여 비즈니스에 사용

- 위의 내용으로 그래프 기반 스케줄을 작성

- 그래프를 이용하여 태스크를 병렬로 실행할 수 있음
- 만약 전체 작업을 하나의 모놀리식(monolithic) 스크립트하면 구현 초기에는 그다지 문제가 발생하지 않지만 파이프라인의 중간 태스크가 실패하면 전체 스크립트를 재실행해야 하기 때문에 비효율적
- 반면에 그래프 기반은 실패한 태스크만 재실행하면 되므로 효율적으로 구성할 수 있음

워크플로 매니저 프로그램

Name	Originated at[a]	Workflows defined in	Written in	Scheduling	Backfilling	User interface[b]	Installation platform	Horizontally scalable
Airflow	Airbnb	Python	Python	Yes	Yes	Yes	Anywhere	Yes
Argo	Applatix	YAML	Go	Third party[c]		Yes	Kubernetes	Yes
Azkaban	LinkedIn	YAML	Java	Yes	No	Yes	Anywhere	
Conductor	Netflix	JSON	Java	No		Yes	Anywhere	Yes
Luigi	Spotify	Python	Python	No	Yes	Yes	Anywhere	Yes
Make		Custom DSL	C	No	No	No	Anywhere	No
Metaflow	Netflix	Python	Python	No		No	Anywhere	Yes
Nifi	NSA	UI	Java	Yes	No	Yes	Anywhere	Yes
Oozie		XML	Java	Yes	Yes	Yes	Hadoop	Yes

a. Some tools were originally created by (ex-)employees of a company; however, all tools are open sourced and not represented by one single company.
b. The quality and features of user interfaces vary widely.
c. https://github.com/bitphy/argo-cron.

airflow

- 데이터 파이프라인을 구성하는데 사용하는 프레임웍
- 파이프라인 구축을 위한 배치 태스크(batch-oriented framework)
- 파이프라인이나 워크플로 태스크를 방향성 비순환 그래프(DAG)로 정의

구성 요소

Airflow 스케줄러

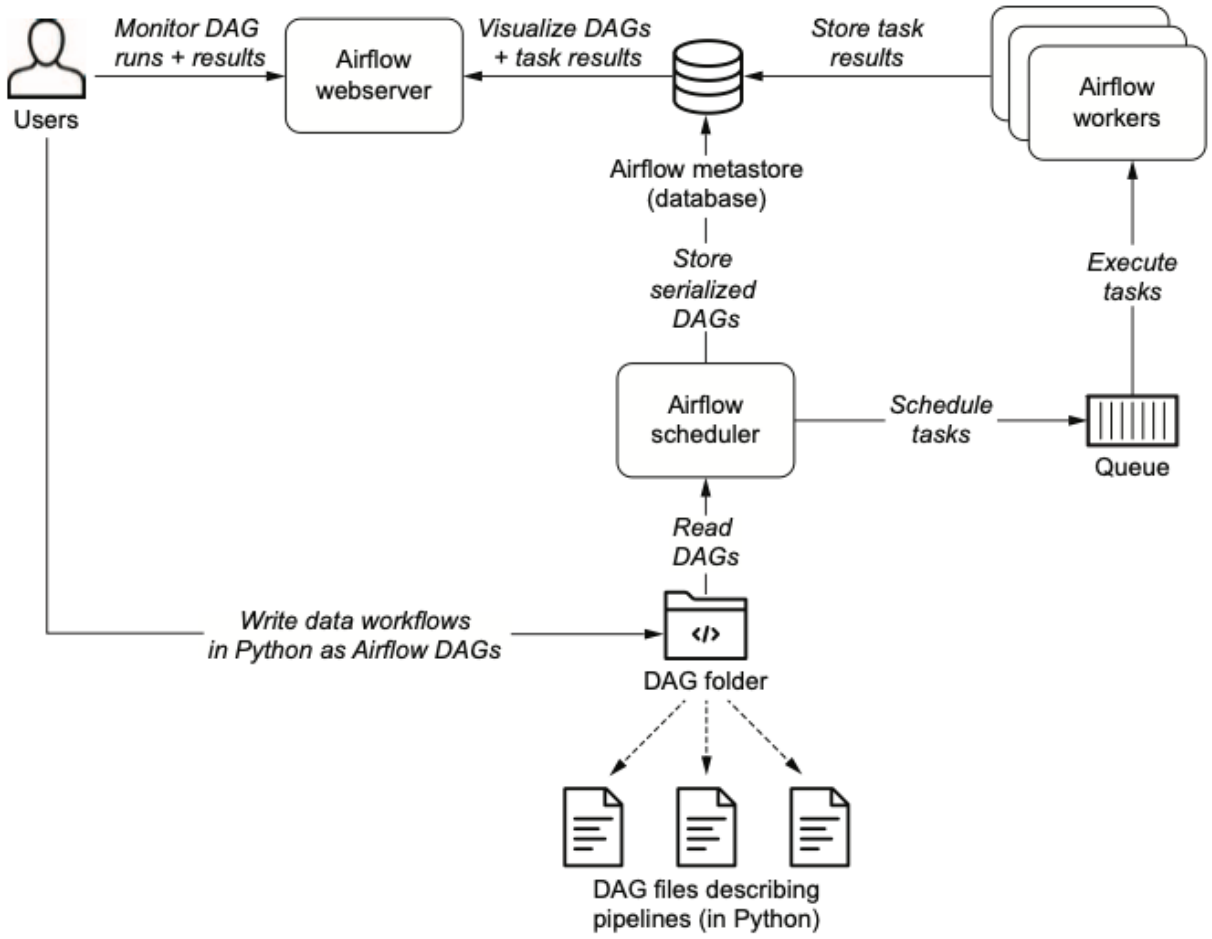
- DAG를 분석하고 현재 시점에서 DAG의 스케줄이 지난 경우 Airflow 워커에 DAG의 태스크를 예약함
- 파이프라인이 실행되는 시기와 방법을 결정하는 하는 곳

Airflow 워커

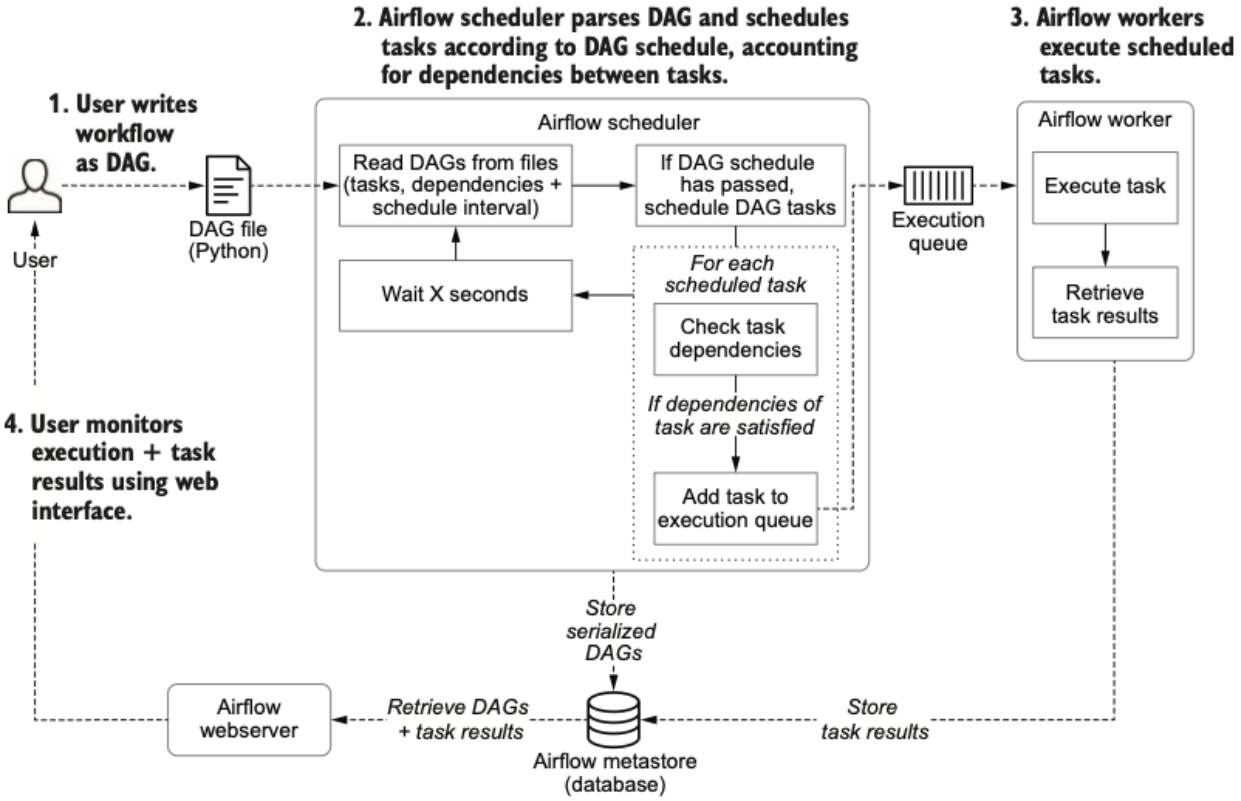
- 예약된 태스크를 선택하고 실행

Airflow 웹 서버

- 스케줄러에서 분석한 DAG를 시각화하고 DAG 실행과 결과를 확인할 수 있는 주요 인터페이스를 제공



1. 사용자가 DAG 워크플로를 작성하면, 스케줄러는 DAG 파일을 분석하고 각 DAG 태스크, 의존성 및 예약 주기를 확인
2. 스케줄러는 마지막 DAG까지 내용을 확인한 후 DAG의 예약 주기가 경과했는 확인
3. 예약된 각 태스크에 대해 스케줄러는 해당 태스크의 의존성을 확인. 의존성 태스크가 완료되지 않았따면 실행 대기열에 추가
4. 스케줄러는 1단계로 다시 돌아간 후 새로운 루프를 잠시 동안 대기



- 태스크가 실행 대기열에 추가되면 Airflow 워커의 풀(pool)의 워커가 태스크를 선택하고 실행
- 이 과정의 모든 결과는 Airflow의 메타스토어로 전달되어, 웹 인터페이스를 통해 태스크 진행 상황을 추적하고 로그를 확인할 수 있음

Airflow의 장점

- 파이썬 코드를 이용해 파이프라인을 구현할 수 있기 때문에 파이썬 언어에서 구현할 수 있는 대부분의 방법을 사용하여 복잡한 커스텀 파이프라인을 만들 수 있음
- 쉽게 확장이 가능하고 다양한 시스템과 통합이 가능
  - 다양한 유형의 데이터베이스, 클라우드 서비스 등과 통합할 수 있는 수 많은 애드온이 존재
- 수 많은 스케줄링 기법은 파이프라인을 정기적으로 실행하고 점진적 증분처리를 통해, 전체 파이프라인을 재실행할 필요 없는 효율적인 파이프라인 구축이 가능
- 백필 기능을 사용하면 과거 데이터를 손쉽게 재처리할 수 있기 때문에 코드를 변경한 후 재생성이 필요한 데이터 재처리가 가능
  - Backfilling(백필)
    - 하나의 플로(Airflow에서는 DAG)를 특정 옵션(기간) 기준으로 다시 실행할 수 있는 기능을 의미
    - 태스크가 며칠 동안 실패하거나 새롭게 만든 플로를 과거의 특정 시점부터 순차적으로 실행하고 싶을 때 수행

- 웹 인터페이스는 파이프라인 실행 결과를 모니터링할 수 있고 오류를 디버깅하기 위한 편리한 뷰를 제공

Airflow가 적합하지 않은 경우

- 배치 태스크를 실행하는 기능에 초점이 맞춰져 있기 때문에, 스트리밍 워크플로 및 해당 파이프라인 처리에 적합하지 않음
- 추가 및 삭제 태스크가 빈번한 동적 파이프라인의 경우에 적합하지 않을 수 있음
  - 구현은 가능하지만 웹 인터페이스는 DAG의 가장 최근 실행 버전에 대한 정의만 표현함
  - 따라서 실행되는 동안 구조가 변경되지 않을 파이프라인에 좀 더 적합함
- 파이썬 프로그래밍 경험이 전혀 없는 팀은 적합하지 않을 수 있음. Azure Data Factory와 같은 그래픽 사용자 인터페이스를 가진 툴이나 다른 정적 워크플로 정의가 가능한 솔루션이 더 합리적인 선택이 될 것임
- 파이썬 코드로 DAG를 작성하는 것은 파이프라인 규모가 커지면 굉장히 복잡해질 수 있음. 그렇기 때문에 장기적으로 DAG를 유지 관리하기 위해서는 초기 사용 시점에서부터 엄격한 관리가 필요