Chapter 6 : 워크플로 트리거

#워크플로 트리거하는 다른 방법

- (기존_chapter3) (특정)시간 간격 설정
  - 문자열(ex: **@daily**)
  - timedelta 객체(ex: **timedelta(days=3)**)
  - cron문자열(ex: **"30 14 * * * "**)
- 다른방법 필요한 경우 : (1) 특정 태스크 수행 후 트리거
  (2) 공유 드라이브에 파일 업로드
  (3) 개발자가 코드 push
  (4) Hive 테이블에 파티션 있을 때, 워크플로 실행 등등..

(ex) 데이터가 비정기적으로 공유 시, 데이터 수집(원천제공) 가능한 시간과 워크플로 실행까지 대기시간이 많이 소요될 수 있음.

|------Data(16:30)--------Data(18:00)---------**Workflow** 시작(02:00)------|

(1) (2) (3)

- (3) 워크플로 실행시간 전까지, (1)+(2) 데이터 수집 대기
- 이를 해결위해, Airflow 특수타입(서브클래스) Operator 인 **센서(sensor)** 활용
  - ➔ 특정 조건이 **true**인지 지속적으로 확인
  - ➔ **false**인 경우, true 될 때까지 or timeout 될 때까지 계속 확인

```
(ex) from airflow.sensors.filesystem import FileSensor

wait_for_supermarket_1=FileSeonsor(
    task_id="wait_for_supermarket_1",
    filepath="/data/supermarket/data.csv",
)
```

- **FileSensor**는 파일("/data/supermarket/data.csv") 존재하는지 확인
  - ➔ 존재 : **true** 반환
  - ➔ 미존재 : **false** 반환
    - ⇨ 센서 지정된 시간(ex: default값 = 60초)동안 대기 후 다시시도
    - ⇨ 1분에 한번씩, 주어진 파일 있는지 **포크(poke)** 함
    - ⇨ **Poking** : 센서 실행, 센서 상태 확인위해 Airflow에서 사용하는 이름
- DAG는 모두 **타임아웃 설정** 가능
- '**태스크 로그**'에서 센서_출력내용 확인 가능

|------ **Workflow** 시작(16:00)-----Data(16:30)--------Data(18:00)---------|

- 보통, DAG 시작시간을(Workflow 시작) 데이터 도착하는 시간 경계에 배치
- 센서는 가용성을 지속적으로 확인 -> 조건 충족 시, 다음 태스크 수행

#사용자 지정 조건 폴링

- 데이터 세트(여러 파일로 구성된) 처리시, 최종 업로드 파일 접미사(_SUCESS) 붙이면
  - ➔ FilesSeonsor는 '**글로빙**'을 사용해 파일 or 디렉터리 이름/패턴 일치시킴
- **PythonSeonsor** : PythonOperator 처럼 Python 콜러블 지원
  - ➔ 조건충족 : **true** 값 반환
  - ➔ 실패 : **false** 값 반환

(ex) PythonSensor 사용해 사용자 지정조건 구현

```
from pathlib import Path

from airflow.sensors.python import PythonSensor

def _wait_for_supermarket(supermarket_id):
    supermarket_path=Path( "/data/" + supermarket_id) // Path객체 초기화
    data_files=supermarket_path.glob("data-*.csv")      // data-*.csv파일 수집
    success_file=supermarket_path / "_SUCCESS"          // "_SUCCESS"파일수집
    return data_files and success_file.exists()         // 데이터 파일과 성공파일
                                                           있는지 확인 후 반환

wait_for_supermarket_1=PythonSensor(
    task_id="wait_for_supermarket_1",
    python_callable=_wait_for_supermarket,
    op_kwargs={"supermarket_id" : "supermarket1"},
    dag=dag,
)
```

#원할하지 않는 흐름의 센서 처리

- 최대시간 초과시 센서는 실패
- timeout 인수 허용 : 최대 실행 허용 시간(초) 지정
  - ➔ 포크 시작 시, 실행시간이 timeout 설정 값 초과할 경우 -> 센서는 실패 반환함
  - ➔ **센서 timeout** : 7일(보통)

 # DAG에서 최대 동시 태스크 수 설정

(ex) Dag=DAG(

Dag_id="korea_app",

Start_date=datetime(2019,1,1),

Schedule_interval="0 0 * * * ",

Concurrency=50, // 이 DAG는 동시에 50개 태스크 실행 허용

)

- ➢ DAG 클래스는 DAG 안에서 동시 실행되는 태스크 수 제어하는 인수 제공
- ➢ DAG 당 많은 태스크 동시 실행 시 -> 눈덩이 효과 발생 (**센서 데드룩** 현상)
  - ➔ 최대 태스크 제한에 도달하면, 전체 시스템 정지될 수 있음 -> **주의** !

[해결방안]

- ➢ 센서 클래스는 (1) poke or (2) reschedule 설정할 수 있는 **mode 인수** 존재
  - ➔ poke 설정 : 최대 태스크 제한 도달 시, 새로운 태스크 차단
    즉, 센서 태스크가 실행중인 동안 태스크 슬롯 차지
  - ➔ reschedule 설정 : 포크 동작 실행할 때만 슬롯 차지
    대기 시간 동안은 슬롯 차지 (x)
- ➢ 동시 태스크 수 제어 : **Airflow 전역 설정 옵션**으로 제어

 # 다른 DAG 트리거 (참고 : 128~129p)

(상황예시) 분석팀은 전체 파이프라인 실행 기다리기 보단, 자신들의 데이터 처리 직후 바로 사용하길 원하는 경우 존재

- **mode="reschedule"** 설정 : 포크 후 슬롯을 해제해, 다른 작업 실행 가능

(ex)
```
wait_for_supermarket_1=PythonSensor(
    task_id="wait_for_supermarket_1",
    python_callable=_wait_for_supermarket,
    op_kwargs={"supermarket_id" : "supermarket1"},
    mode="reschedule",
    dag=dag,
)
```

- **TriggerDagRunOperator** : 다른 DAG 트리거 (참고: 129p~131p)
  - ➔ **Trigger_dag_id 인수** 제공

#TriggerDagRunOperator로 백필 작업 (참고 : 131~132p)

#다른 DAG 상태 폴링 (참고 : 132~135p)

- **ExternalTaskSensor : 다른 DAG 태스크 지정 -> 해당 태스크 상태** 확인

#REAT/CLI 이용해 워크플로 시작 (참고 : 135~138p)

- 다른 DAG에서 DAG를 트리거하는 방법 外 **REST API** 및 **CLI** 통한 트리거
  - Airflow 외부에서 DAG 트리거
  - CI/CD 파이프라인 일부로 Airflow 외부에서 워크플로 시작하려는 경우에 해당