



airflow를 minikube환경에서 helm으로 설치하는 방법에 대해서 알려줘. values.yaml 작성해서 dag 파일 설정하는것 까지.(dag파일은 로컬 경로로 할거야)

좋아요. Minikube에서 Helm으로 Airflow(2.9.2)를 깔고, **로컬 경로의 DAG 파일**을 바로 마운트해서 쓰는 방법을 처음부터 끝까지 정리해 드릴게요. 현업에서 바로 따라 하기 좋게 “값(YAML) + 명령” 형태로요.

0) 준비물 체크

- Minikube가 실행 중이어야 합니다. (리소스는 최소 `--cpus=4 --memory=8g`)
- kubectl/helm 설치
- 로컬 DAG 디렉터리(예: `~/airflow/dags`)에 테스트 DAG 1개

```
bash

minikube start --cpus=4 --memory=8192
kubectl create ns airflow
helm repo add apache-airflow https://airflow.apache.org
helm repo update
```

1) 로컬 DAG 디렉터리 → Minikube 노드로 마운트

Minikube는 노드(가상머신) 파일시스템과 내 PC의 디렉터리를 연결할 수 있습니다. Airflow 컨테이너 내부 유저 UID는 통상 **50000(airflow)**이므로, 읽기 권한 문제를 피하려고 **uid/gid를 지정**해서 마운트해 둘게요.

이 명령은 **별도 터미널**에서 계속 실행 상태로 유지됩니다.

```
bash

# 터미널 A (계속 켜두기)
mkdir -p ~/airflow/dags
# 예시 DAG 하나
cat > ~/airflow/dags/example_local_dag.py <<'PY'
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

with DAG(
    dag_id="example_local_dag",
    start_date=datetime(2024,1,1),
    schedule=None,
    catchup=False,
    tags=["local"],
) as dag:
    t1 = BashOperator(
        task_id="echo_hello",
        bash_command="echo 'Hello from local DAG!'"
    )
PY

# 로컬 디렉터리 ~/airflow/dags 를 노드 경로 /data/airflow-dags 로 마운트
```

```
# airflow(UID 50000)가 읽을 수 있도록 권한 설정
minikube mount ~/airflow/dags:/data/airflow-dags --uid=50000 --gid=0
```

2) DAG용 PV/PVC 생성(hostPath)

위에서 노드 경로 `/data/airflow-dags`로 뚫었으니, 이 경로를 **hostPath PV**로 만들고 PVC로 묶습니다. (읽기 전용으로 써도 되고, RW로 뒀도 됩니다. 여기선 RW로 예시)

```
yaml

# airflow-dags-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: airflow-dags-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ""
  hostPath:
    path: /data/airflow-dags
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: airflow-dags-pvc
  namespace: airflow
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: ""
  volumeName: airflow-dags-pv
```

적용:

```
bash

kubectl apply -f airflow-dags-pv.yaml
```

팁: Minikube는 단일 노드라 **hostPath**가 심플합니다. 멀티노드/클라우드에선 NFS/CSI, 또는 Git Sync를 더 권장해요.

3) Helm 값 파일(values.yaml) 작성

- **Executor**: Minikube에선 의존성이 간단한 **LocalExecutor**가 편합니다. (Redis 불필요)
- **PostgreSQL**: 차트 내 내장 Postgres 사용(개발용)
- **DAG 볼륨**: 방금 만든 `airflow-dags-pvc`를 `/opt/airflow/dags`에 마운트
- **관리자 계정 자동 생성**
- 예제에서는 Airflow 이미지 태그를 **2.9.2**로 고정

```
yaml

# values.yaml
images:
  airflow:
    repository: apache/airflow
    tag: "2.9.2"

executor: "LocalExecutor"

# 기본 설정 (필요한 것만)
airflow:
  config:
    core:
      load_examples: "False" # 예제 DAG 불러오지 않기
  users:
    - username: admin
      password: admin
      role: Admin
      email: admin@example.com
      firstName: Admin
      lastName: User

# 웹서버 서비스는 ClusterIP로 두고, 포트포워딩로 접근
webserver:
  service:
    type: ClusterIP

# 데이터베이스/브로커
postgresql:
  enabled: true
redis:
  enabled: false # LocalExecutor는 Redis 불필요

# DAG 볼륨: 기존 PVC를 /opt/airflow/dags로 마운트
dags:
  persistence:
    enabled: true
    existingClaim: "airflow-dags-pvc"
    subPath: "" # 서브경로 필요 없으면 빈 값
    # readOnly: true # 읽기 전용으로 두고 싶으면 주석 해제

# 잡/마이그레이션
migrateDatabaseJob:
  enabled: true
createUserJob:
  enabled: true
```

필요 시 **triggerer**, **workers** 등 컴포넌트 설정을 추가로 손볼 수 있습니다. LocalExecutor에선 워커가 따로 띄지 않고 스케줄러가 로컬에서 태스크를 처리합니다.

4) 설치

```
bash

helm upgrade --install airflow apache-airflow/airflow \
  --namespace airflow \
  -f values.yaml
```

상태 확인:

```
bash

kubectl get pods -n airflow
```

초기 DB 마이그레이션과 사용자 생성 작업이 끝나면 **webserver**, **scheduler**, **triggerer**가 Running이 됩니다.

5) 접속(포트 포워딩)

```
bash

kubectl port-forward -n airflow svc/airflow-webserver 8080:8080
```

브라우저에서 **http://localhost:8080** → **admin/admin** 로그인

DAGs 화면에 **example_local_dag**가 보이면 성공입니다. (리스트에 뜨는 데 몇십 초 걸릴 수 있어요)

6) 로컬에서 DAG 수정 → 즉시 반영

- 지금 구성은 로컬 디렉터리(**~/airflow/dags**) → **Minikube** 노드(**/data/airflow-dags**) → **PVC/PV** → **/opt/airflow/dags**로 직결 마운트라서, 로컬 파일을 저장하면 곧바로 Web UI에 반영됩니다.
- 바뀐 DAG가 안 보이면:
 - 파일 확장자 **.py** 맞는지
 - 파서 오류 없는지 (Web UI > Browse > DAG Parser Errors)
 - 스케줄러 로그 확인
 - 필요 시 웹서버/스케줄러 재시작

```
bash

kubectl logs -n airflow deploy/airflow-scheduler
kubectl rollout restart deploy/airflow-webserver -n airflow
kubectl rollout restart deploy/airflow-scheduler -n airflow
```

7) 간단 실행 테스트

Web UI에서 **example_local_dag** → “Trigger DAG” → “Graph”에서 태스크 상태 확인
또는 CLI:

```
bash

# 웹서버/스케줄러 컨테이너 안에서 실행하고 싶다면:
kubectl exec -it -n airflow deploy/airflow-scheduler -- bash -lc 'airflow dags list && airflow
dags trigger example_local_dag'
```

흔한 문제 & 빠른 진단

1. DAG이 안 보임

- 파서 오류 확인: Web UI → *Browse* → *DAG Parser Errors*
- 스케줄러 로그 확인
- `load_examples=False`라 기본 예제는 안 뜹니다(의도)

2. 권한 문제

- `minikube mount`에 `--uid=500000 --gid=0`를 꼭 줬는지 확인
- 필요 시 PVC를 `ReadOnly`로 두는 것도 한 방법

3. Pods가 Pending

- PVC 바인딩 확인: `kubectl get pvc -n airflow`
- PV와 PVC의 `storageClassName/volumeName` 일치 체크

4. 포트 접근 문제

- 이번 가이드는 Ingress 없이 **포트포워딩** 기반
- Ingress가 필요하면 `webserver.ingress.enabled: true`로 바꾸고 도메인/인증서 설정 추가

참고: GitSync 방식이 나올 때

- 팀원이 여러 명이거나 멀티노드/클라우드 환경이라면 Git 리포에서 자동 동기화하는 **`**dags.gitSync**`**가 더 안전합니다.
- 하지만 지금은 “로컬 경로를 바로 쓰고 싶다”가 목표라 `hostPath + minikube mount`가 가장 간단합니다.

필요하면 이 구성을 **Ingress(회사 도메인)**, **Helm 차트 프로파일 분리(dev/prod)**, **KubernetesExecutor**(위커 스케일아웃) 쪽으로 확장하는 템플릿도 붙여 드리겠습니다. 지금은 일단 로컬 경로 DAG까지 잘 뜨게 만드는 데 집중했어요.



sample dag활성화는 어떻게 하지?

Airflow에서 “샘플 DAG 활성화”라는 말은 보통 두 가지 케이스가 있어요.

1. Helm 설치 시 `load_examples` 옵션

제가 아까 드린 `values.yaml`에서는 이렇게 되어 있었죠:

```
yaml

airflow:
  config:
    core:
      load_examples: "False"
```

여기를 "True"로 바꿔주면, Airflow 설치 시 자동으로 제공되는 **example_bash_operator, tutorial** 같은 예제 DAG 들이 Web UI에 나타납니다.

변경 후 적용:

```
bash

helm upgrade airflow apache-airflow/airflow -n airflow -f values.yaml
```

2. Web UI에서 특정 DAG 활성화

Airflow Web UI → **DAGs** 화면에서 기본 상태는 "Paused"입니다.

- **Toggle Switch** (ON/OFF 스위치)를 클릭해서 켜야 활성화돼요.
- 켜면 스케줄러가 DAG 실행 일정을 반영합니다.
- 즉, 리스트에 보이지만 회색(Paused)인 경우엔 수동으로 켜줘야 해요.

👉 정리하자면:

- **샘플 DAG이 아예 안 보인다** → `load_examples=True`로 Helm 설정 변경 필요
- **보이긴 하는데 안 돈다** → Web UI에서 토글을 눌러 활성화해야 함

혹시 원하시는 게 "예제 DAG이 처음부터 자동으로 켜진 상태로 올라오게" 하는 건가요? 아니면 그냥 지금 Web UI에서 보이는 회색 상태를 켜는 법이 궁금하신 건가요?



dag파일이 스케줄러 노드에 생성할때만 인식이 되는게, 이게 정상인거야?

좋은 포인트 짚으셨어요. Airflow가 DAG을 인식하는 방식 때문에 그렇게 보이실 수 있어요. 정리해 드리면:

DAG 인식 동작 원리

- 스케줄러 프로세스가 `/opt/airflow/dags` 경로를 주기적으로 스캔해서 DAG 파일을 파싱합니다.
- 웹서버는 직접 DAG을 읽는 게 아니라, **스케줄러가 DB에 등록한 DAG 메타데이터**를 보여주는 구조예요.
- 그래서 결국 DAG을 "등록/인식"하는 주체는 스케줄러 하나뿐입니다.

그래서 생기는 현상

- DAG 파일이 PVC로 마운트되어 있더라도, **스케줄러 Pod**가 그 파일을 볼 수 있어야만 DAG이 인식됩니다.
- 웹서버만 PVC를 마운트하고 스케줄러에는 안 돼 있으면 → DAG이 절대 안 보임.
- DAG을 추가/수정해도 스케줄러가 해당 경로를 스캔해야만 Web UI에서 반영됩니다.

정상 동작 맞춤

네, 말씀하신 대로 ***스케줄러 노드(혹은 스케줄러 Pod)에서만 DAG이 인식되는 것***이 정상이에요.
다른 Pod(웹서버, 워커)는 DAG 코드가 굳이 필요하지 않고, 실행할 때 Task 인스턴스만 DB에서 가져오기 때문이에요.

확인 포인트

1. Helm values.yaml에서 DAG PVC가 **스케줄러(scheduler)**, **웹서버(webserver)**, **트리거러(triggerer)** 모두에 마운트되도록 설정했는지
(공식 차트는 **dags.persistence** 옵션이 있으면 공통으로 마운트해 줍니다.)
2. 그래도 반영이 안 된다면, **scheduler** Pod 로그 확인

```
bash

kubectl logs deploy/airflow-scheduler -n airflow
```

DAG 파싱 오류가 있으면 DAG이 아예 안 보이기도 해요.

👉 한마디로 요약하면:

DAG이 스케줄러에서만 인식되는 건 Airflow 아키텍처상 정상이에요. 웹서버는 DAG 파일을 직접 읽지 않고, 스케줄러 DB 반영 결과만 보여주거든요.

혹시 지금 상황이 ***스케줄러 Pod이 재시작되거나 새로 생성될 때만 DAG이 잡히고, 그 외에는 수정해도 반영이 안 된다***를 말씀하시는 건가요? 아니면 단순히 "스케줄러만 DAG을 인식한다"는 구조적인 차이를 여쭙본 건가요?



dag파일을 생성했을때, 최대한 빨리 인식되게 하려면?

좋은 질문이에요. "새 DAG/수정된 DAG을 최대한 빨리 잡히게" 하는 데는 몇 가지 레버가 있습니다. 빠른 것부터, 설정까지 한 번에 정리해드릴게요.

당장 효과 보는 3가지

1. 파일 mtime 갱신(= touch)

- 스케줄러는 파일 변경시각을 근거로 파싱 대상 여부를 판단해요.
- 로컬에서 저장했는데 바로 안 뜨면, 해당 .py에 한 번 더 저장/touch 하세요.

```
bash

# 로컬
touch ~/airflow/dags/my_new_dag.py
```

Minikube mount를 쓰면 mtime 전달이 가끔 지연되는데, touch가 체감상 가장 즉각적입니다.

2. 스케줄러 재시작 (강제 파싱 트리거)

```
bash

kubectl rollout restart deploy/airflow-scheduler -n airflow
```

- 무식하지만 확실한 방법. 새/변경 DAG이 바로 재파싱됩니다.

3. 강제 리시리얼라이즈(웹이 더 빨리 반영)

```
bash

kubectl exec -it -n airflow deploy/airflow-scheduler -- \
  bash -lc 'airflow dags reserialize --include-paused --clear-only'
```

- 스케줄러가 DAG 메타를 DB에 직렬화하는 주기를 기다리지 않고, 웹서버가 읽을 최신 스냅샷을 강제로 갱신합니다.

근본 원리: 어떤 주기를 줄여야 빨라지는가

Airflow는 “파일 발견 → 파싱 → 직렬화(DB 반영) → 웹서버가 표시” 흐름이에요. 그래서 **아래 인터벌**을 줄이면 빨라집니다. (기본값은 개발자 입장에서 다소 길어요)

values.yaml 예시 (인터벌 타이트하게)

```
yaml

airflow:
  config:
    # (1) DAG 디렉터리 스캔 주기: 새 파일을 얼마나 자주 찾을지
    core:
      dag_dir_list_interval: "10"          # 기본 300 초인 환경 많음 → 10초로
      min_serialized_dag_update_interval: "5"  # 웹서버가 볼 직렬화 주기↓

    # (2) 스케줄러 파싱 루프 관련
    scheduler:
      processor_poll_interval: "1"          # 파서 프로세스 상태 폴링 주기↓
      min_file_process_interval: "5"        # 동일 파일 재파싱까지 최소 대기↓
      parsing_processes: "4"                # 동시 파싱 프로세스↑ (파일 많을 때 유리)
```

주의: 버전에 따라 섹션([core] vs [scheduler])이 달랐던 적이 있어요. Airflow 2.9.2 기준으로 위 키들은 정상 작동합니다. 값을 너무 낮추면 CPU를 꽤 씹니다. Minikube면 **parsing_processes**는 2~4 정도가 무난해요.

적용:

```
bash

helm upgrade airflow apache-airflow/airflow -n airflow -f values.yaml
```

GitSync를 쓰는 경우(참고)

- `dags.gitSync`를 쓰면 **git 동기화 주기**(예: `GIT_SYNC_PERIOD`)도 병목이에요.
- 이 값(예: 10s)과 위 Airflow 인터벌들이 모두 짧아야 전체 체감이 빨라집니다.
- 로컬 마운트(minikube mount)보다 안정적이고 멀티노드에서도 일관됩니다.

체크리스트 (느릴 때 자주 걸리는 포인트)

- 스케줄러 Pod에도 **DAG PVC가 마운트** 되었는가? (웹서버만 마운트되어 있으면 영원히 안 잡힘)
- 파싱 에러가 없는가? Web UI → *Browse* → *DAG Parser Errors* 확인.
- **minikube mount 지연**: 가끔 이벤트 전파가 느립니다. `touch` + 위 인터벌 조정이 해법.
- **대용량 리포/파일 수 많음**: `parsing_processes` ↑, 불필요한 디렉터리는 `.airflowignore`로 제외.

`.airflowignore` 예:

```
shell

# /opt/airflow/dags/.airflowignore
tests/
old/
*.tmp
```

한 줄 결론

- **즉시**: `touch` → `airflow dags reserialize` → (그래도 답답하면) `scheduler` 재시작
- **지속적으로**: `dag_dir_list_interval`, `min_file_process_interval`, `min_serialized_dag_update_interval`를 짧게, `parsing_processes`는 적당히 크게

원하시면 지금 쓰시는 `values.yaml` 붙여주시면, 위 인터벌을 부작용 최소화 선에서 딱 맞게 튜닝해 드릴게요.