

# Minikube에서 Helm 설치



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



1m 1s 동안 생각함

- Airflow를 Minikube에 Helm으로 설치하고, local path에서 DAGs를 구성하는 방법을 설명해야겠어요. "dags.persistence.enabled: false"를 사용해 로컬 DAGs를 탑재하거나, "dags.gitSync", "dags.existingClaim", "dags.mountPath" 옵션을 사용해야 합니다. Minikube에서 로컬 디렉토리를 마운트하려면 이와 관련된 구성을 설정해야겠네요.
- 위의 과정을 정리하도록 할게요. Minikube에서 로컬 디렉토리를 사용할 수 있도록 설정하고, PVC와 PV를 사용하여 hostPath를 연결하는 방법을 안내해야 해요. 또한 Airflow 버전 2.9.2를 사용하고, LocalExecutor로 설정하고 PostgreSQL을 사용할 거라면 이를 전체적으로 구조화해야겠네요.
- 기본적으로 차트는 CeleryExecutor를 사용하지만, Minikube에서는 LocalExecutor가 더 나을 것 같아요 (Redis 없이 설정 가능). values.yaml에서 LocalExecutor를 설정하고 Redis와 워커는 비활성화합니다. DB 생성은 postgresql.enabled를 true로 설정하고, 다른 설정들도 필요한 값을 적어 넣으면 됩니다.
- Minikube와 Kubernetes 1.31.1을 사용하고, Airflow 2.9.2 버전의 차트를 사용할 때, 필요한 명령어와 YAML 코드를 제공할 예정이에요. PV 권한 설정 시, `securityContext fsGroup 0`이나 `extraInitContainers`로 소유권을 변경해야 하고, DAGs는 readOnly로 마운트 할 수 있어요.

☑ 완료

좋아요. Minikube에서 Helm으로 Airflow(2.9.2)를 깔고, 로컬 경로의 **DAG** 파일을 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



```
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/gid**를 지정해서 마운트해 둘게요.

이 명령은 **별도 터미널**에서 계속 실행 상태로 유지됩니다.

```
# 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)

```
# 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:

```
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

```
# 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, 포트포워딩으로 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: ""          # 서버경로 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. `LocalExecutor`에 선 워커가 따로 뜨지 않고 스케줄러가 로컬에서 태스크를 처리합니다.

## 4) 설치

```

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

```

상태 확인:

```
kubectl get pods -n airflow
```

초기 DB 마이그레이션과 사용자 생성 잡이 끝나면 ``webserver``, ``scheduler``, ``triggerer``가 Running이 됩니다.

## 5) 접속(포트 포워딩)

```
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 직접 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

```
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:

```
# 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=50000 --gid=0`를 꼭 줬는지 확인
- 필요 시 PVC를 ReadOnly로 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**(워커 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:

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



여기를 `"True"`로 바꿔주면, Airflow 설치 시 자동으로 제공되는 **example\_bash\_operator**, **tutorial** 같은 예제 DAG들이 Web UI에 나타납니다.

변경 후 적용:

```
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 정상인 거야?



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 메타데이터를 보여주는 구조예요.
- 그래서 결국 DAG을 “등록/인식”하는 주체는 스케줄러 하나뿐입니다.

---

## 그래서 생기는 현상

- DAG 파일이 PVC로 마운트되어 있더라도, 스케줄러 Pod가 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 정상이예요. 다른 Pod(웹서버, 워커)는 DAG 코드가 굳이 필요하지 않고, 실행할 때 Task 인스턴스만 DB에서 가져오기 때문예요.

---

## 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

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

DAG 파싱 오류가 있으면 DAG이 아예 안 보이기도 해요.

---

👉 한마디로 요약하면:

DAG이 스케줄러에서만 인식되는 건 Airflow 아키텍처상 정상이예요. 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?



32s 동안 생각함

- I need to figure out how to make Airflow pick up DAG changes faster. I'm considering adjusting parameters like ``min_file_process_interval``, ``dag_dir_list_interval``, and ``min_serialized_dag_update_interval``, all tied to scheduler parsing configs. There's also an option to trigger a manual parsing, which might be needed. The config names shifted a little over versions, like in 2.3, but I think the settings mostly remain in ``[core]`` and ``[scheduler]`` sections.
- I need to check if settings like ``[scheduler] min_serialized_dag_update_interval`` or ``[core] min_serialized_dag_update_interval`` should be adjusted. There's also behavior such as using ``kubectl exec`` to touch files or tweaking the Helm chart for quicker DAG updates. Reducing intervals like ``dag_dir_list_interval`` and ``min_file_process_interval`` will speed up things, but might increase CPU load, so I need to be cautious.

☑ 완료

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 ``.py``에 한 번 더 저장/``touch`` 하세요.

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

Minikube mount를 쓰면 mtime 전달이 가끔 지연되는데, `touch`가 체감상 가장 즉각적입니다.

## 2. 스케줄러 재시작 (강제 파싱 트리거)

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

- 무식하지만 확실한 방법. 새/변경 DAG이 바로 재파싱됩니다.

## 3. 강제 리시리얼라이즈(웹이 더 빨리 반영)

```
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)

```
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 씁니다. Minikube면 `parsing\_processes`는 2~4 정도가 무난해요.

적용:

```
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 **Pod**에도 **DAG PVC**가 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:

```
# /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.



airflow 이미지에 vi를 설치한 이미지를 사용하고싶어



12s 동안 생각함

- 사용자는 Airflow 2.9.2 버전에 VI를 설치한 커스텀 도커 이미지를 만들려는 것 같아요. 공식 이미지가 Debian 기반이라 apt-get을 사용하고, slim 이미지도 있지만, vi를 설치하려면 vim-tiny나 nano가 적합할 거예요. Dockerfile을 제공하고 registry에 푸시 후 values.yaml을 통해 해당 이미지를 설정해야 합니다.

👍 완료

원하시는 거 가능해요. 방법은 3가지 정도로 깔끔하게 정리해드릴게요. 보통은 커스텀 이미지 빌드가 제일 간단합니다.

## 1) 가장 권장: Airflow 베이스로 커스텀 이미지 빌드(vi 포함)

### Dockerfile

```
# Airflow 2.9.2 기준
FROM apache/airflow:2.9.2

USER root
# 용량/의존성 최소화하려면 vim-tiny 추천 (vim도 무방)
RUN apt-get update \
    && apt-get install -y --no-install-recommends vim-tiny \
    && rm -rf /var/lib/apt/lists/*
USER airflow
```

빌드 & 푸시:

```
# 레지스트리/태그는 환경에 맞게
docker build -t <REGISTRY>/<NAMESPACE>/airflow:2.9.2-vi .
docker push <REGISTRY>/<NAMESPACE>/airflow:2.9.2-vi
```

values.yaml에서 이미지 교체:

```
images:
  airflow:
    repository: <REGISTRY>/<NAMESPACE>/airflow
    tag: "2.9.2-vi"    # 위에서 푸시한 태그

# (선택) 만약 Helm 차트 버전에 따라 컴포넌트별 image override가 있다면 동일하게 맞춰 주세요.
# webserver: {}
```

```
# scheduler: {}  
# triggerer: {}
```

적용:

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

확인:

```
kubectl exec -it -n airflow deploy/airflow-scheduler -- bash -lc 'vi --version  
|| which vi'
```

참고: 공식 `apache/airflow` 이미지는 Debian 베이스라 `apt-get`이 맞습니다. alpine이 아니  
예요.

## 2) 빠른 임시 해킹: initContainer에서 설치(권장 X)

이미지 빌드가 어려울 때, Pod가 뜰 때마다 설치하게 만들 수도 있어요. 다만 **Pod** 기동이 느려지고 네트  
워크가 막힌 환경에선 실패합니다.

```
scheduler:  
  extraInitContainers:  
    - name: add-vi  
      image: debian:bookworm-slim  
      command: ["/bin/bash", "-lc"]  
      # Airflow 컨테이너가 Debian이므로 동일 계열 패키지를 공유할 수 없고,  
      # 보통은 sharedEmptyDir에 바이너리 넣고 마운트해서 우회합니다.  
      args:  
        - >  
          apt-get update &&  
          apt-get install -y --no-install-recommends vim-tiny &&  
          cp /usr/bin/vim.tiny /workdir/vi &&  
          chmod +x /workdir/vi  
      volumeMounts:  
        - name: vi-bin  
          mountPath: /workdir  
  
  extraVolumeMounts:  
    - name: vi-bin  
      mountPath: /usr/local/bin
```

```
readOnly: false

extraVolumes:
  - name: vi-bin
    emptyDir: {}
```

이 방식은 “/usr/local/bin/vi”처럼 심볼릭/복사로 우회하는 트릭입니다. 정식 운영엔 비추. 그냥 커스텀 이미지가 낫습니다.

### 3) 일회성 디버깅: `kubectl debug`로 에페메럴 컨테이너 붙이기

편집용 에디터가 당장 하나 필요할 때:

```
# 스케줄러 Pod에 디버그 컨테이너 붙이기(읽기/쓰기 네임스페이스 공유)
kubectl debug -n airflow pod/airflow-scheduler-xxxxx \
  -it --image=debian:bookworm-slim --target=airflow-scheduler
# 디버그 셸에서 vi 설치
apt-get update && apt-get install -y --no-install-recommends vim-tiny
vi /opt/airflow/dags/xxx.py
```

Pod 재시작하면 사라집니다. “지금 당장”용.

#### 작은 팁들

- **에디터 대안:** 이미지 용량을 더 줄이고 싶으면 `vim-tiny` 대신 `nano`도 가볍습니다.
- **권한:** 편집 대상이 DAG PVC라면, 컨테이너 내부에서 `airflow(UID 50000)`가 쓰기 가능한지 확인 필요.
- **운영 철학:** 컨테이너 내부 편집은 긴급 디버깅용. 실무에선 Git → CI → 이미지/동기화가 표준이에요. 지금은 “vi 있는 툴박스”가 필요하니 1번처럼 **베이크**해 두면 깔끔합니다.

원하시면 지금 쓰시는 `values.yaml` 보여주시면, **모든 컴포넌트**

**(webserver/scheduler/triggrer/worker)** 가 같은 커스텀 이미지를 사용하게끔 정확한 키 경로까지 맞춰 드릴게요.



에어플로우를 로컬에서 테스트용으로 나만 사용하는 거니깐 최소로 설치하는 방법



좋아요. “나 혼자 쓰는 테스트용”이면 **가장 가벼운 Helm 구성**으로 가면 됩니다. 핵심은  
`LocalExecutor + 내장 Postgres만` 쓰고 나머지는 다 끄는 것. 아래 그대로 쓰시면 돼요.

## 0) 준비

```
minikube start --cpus=2 --memory=4096
kubectl create ns airflow
helm repo add apache-airflow https://airflow.apache.org
helm repo update
```

## 1) 최소 values.yaml

- Executor는 **LocalExecutor** (Redis/Worker 불필요)
- **PostgreSQL만 켜**, 나머지(Flower/Triggerer/StatsD/Redis)는 전부 끄
- 리소스 아주 작게
- DAG는 “로컬 경로” 마운트(예: `~/airflow/dags` → minikube mount)

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

executor: "LocalExecutor"

airflow:
  config:
    core:
      load_examples: "False"
      # 스캔 주기 조금만 빠르게 (너무 낮추면 CPU 써요)
      dag_dir_list_interval: "20"
```



```

    scheduler:
      processor_poll_interval: "2"
      min_file_process_interval: "10"
  users:
    - username: admin
      password: admin
      role: Admin
      email: admin@example.com
      firstName: Admin
      lastName: User

  webserver:
    service:
      type: ClusterIP
    resources:
      requests: { cpu: "100m", memory: "256Mi" }
      limits: { cpu: "500m", memory: "512Mi" }

  scheduler:
    resources:
      requests: { cpu: "100m", memory: "256Mi" }
      limits: { cpu: "500m", memory: "512Mi" }

# deferrable operator 안 쓸 거면 꺼도 됨
  triggerer:
    enabled: false

# 모니터링/브로커 모두 끄
  flower:
    enabled: false
  redis:
    enabled: false
  statsd:
    enabled: false

# DB는 개발용으로 chart 내장 Postgres 사용(가장 단순)
  postgresql:
    enabled: true
    primary:
      resources:
        requests: { cpu: "50m", memory: "128Mi" }

```

```

    limits:    { cpu: "300m", memory: "256Mi" }

# DB 마이그레이션/유저 생성 잡만 켜기
migrateDatabaseJob:
    enabled: true
createUserJob:
    enabled: true

# DAG: PVC로 /opt/airflow/dags 마운트
dags:
    persistence:
        enabled: true
        existingClaim: "airflow-dags-pvc"

```

## 2) 로컬 DAG 경로 → Minikube 마운트 + PV/PVC

```

mkdir -p ~/airflow/dags
cat > ~/airflow/dags/mini_example.py <<'PY'
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

with DAG("mini_example", start_date=datetime(2024,1,1), schedule=None,
catchup=False):
    BashOperator(task_id="echo", bash_command="echo hi")
PY

```

```

# 터미널 하나에서 계속 유지
minikube mount ~/airflow/dags:/data/airflow-dags --uid=500000 --gid=0

```

```

# airflow-dags-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
    name: airflow-dags-pv
spec:
    capacity:
        storage: 256Mi
    accessModes: ["ReadWriteMany"]

```

```

storageClassName: ""
persistentVolumeReclaimPolicy: Retain
hostPath:
  path: /data/airflow-dags
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: airflow-dags-pvc
  namespace: airflow
spec:
  accessModes: ["ReadWriteMany"]
  resources:
    requests:
      storage: 256Mi
  storageClassName: ""
  volumeName: airflow-dags-pv

```

적용:

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

### 3) 설치

```

helm upgrade --install airflow apache-airflow/airflow \
  -n airflow -f values-minimal.yaml
kubectl get pods -n airflow

```

### 4) 접속(포트포워딩)

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

- 브라우저: <http://localhost:8080> ↗ (admin / admin)
- DAGs 화면에서 `mini\_example` 보이면 OK → 토글 켜고 실행

문제 없이 더 가볍게 쓰는 요령

- **Triggerer 비활성화**: deferrable operator 안 쓰면 지금처럼 끄기
- 리소스 더 줄이기: Minikube에서 버벅이면 web/scheduler의 limits를 더 낮춰도 됩니다
- 예제 **DAG 비활성화**: ``load_examples=False`` 유지
- **빠른 인식**: 파일 저장 후 안 뜨면 ``touch`` 한 번, 그래도 답답하면

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

## 더 미니멀? (참고)

- 쿠버네티스 말고 진짜 로컬만이라면 ``airflow standalone``(단일 프로세스 + SQLite)도 가능해요.
- 하지만 “Minikube에서 테스트”가 목표면 위 구성이 헬스체크/웹UI까지 포함한 **최소 안정 구도**입니다.

필요하시면 ``values-minimal.yaml``에서 **Ingress** 추가나 **\*\*커스텀 이미지(vi 포함)\*\***를 바로 섞은 버전으로도 정리해 드릴게요.



helm airflow 삭제.



Airflow를 Helm으로 설치했을 때 지우는 건 아주 단순합니다. 다만 **남는 리소스**(PVC, PV, Secret, ConfigMap 등)를 어디까지 정리할지가 관건이에요.

## 1) 기본 삭제 (Helm 릴리스만)

```
# 네임스페이스 airflow에 설치했다고 가정
helm uninstall airflow -n airflow
```

→ 이렇게 하면 Helm이 만든 Deployment/Service/Pod 등만 지워집니다.  
PVC, PV는 기본적으로 **남아있습니다** (데이터 보존 목적).

## 2) 네임스페이스ごと 싹 비우기 (테스트 환경이라면 깔끔)

```
kubectl delete ns airflow
```

→ 이렇게 하면 네임스페이스 아래 PVC, Secret, ConfigMap까지 같이 없어집니다.

→ 단, `PersistentVolume`은 cluster 스코프라 여전히 남을 수 있습니다.

### 3) PV까지 완전 삭제

만약 제가 예시로 만든 `airflow-dags-pv` 같은 hostPath PV도 없애고 싶으면:

```
kubectl delete pv airflow-dags-pv
```

### 4) 혹시 남은 것 확인

```
kubectl get all -n airflow  
kubectl get pvc -n airflow  
kubectl get pv | grep airflow
```

#### 👉 정리

- 테스트 용도로 다시 깔 거라면 `helm uninstall airflow -n airflow` 만 해도 충분
- 완전히 깨끗하게 초기화하고 싶으면 `kubectl delete ns airflow && kubectl delete pv <이름>`

혹시 원하시는 게 “테스트하다 꼬였으니 환경을 완전히 초기화” 쪽인가요, 아니면 단순히 릴리스만 제거하고 나중에 다시 설치하려는 건가요?