

Airflow-uid-쓰기-이슈



airflow사용자가 airflow에서 50000 번이야?



에어플로우에서 `airflow`라는 리눅스 계정이 자동으로 만들어질 때 UID(사용자 ID)가 **50000**으로 잡히는 걸 물어보시는 거죠?

- 기본 `apache/airflow` 공식 도커 이미지를 보면, `airflow`라는 계정을 UID=50000, GID=0(root 그룹)으로 만들어둡니다.

```
RUN groupadd --gid 0 airflow \  
&& useradd --uid 50000 --gid 0 --shell /bin/bash --create-home airflow
```

- 그래서 컨테이너 안에서 `id airflow` 하면 보통 이렇게 나와요:

```
uid=50000(airflow) gid=0(root) groups=0(root)
```

즉, 로컬 리눅스에서 새로 계정을 만든 게 아니라 공식 이미지에서 이미 정해둔 값이라서 50000이 붙는 겁니다.

👉 궁금한 게, 이게 단순히 "왜 50000이냐" 같은 호기심인지, 아니면 PVC 마운트 같은 데서 권한 문제 (UID mismatch) 때문에 신경 쓰이는 상황인가요?



그럼 공유 디렉토리(nfs)에서 2001번 uid 로 755 권한이 만들어져 있는데, airflow 사용자가 write 할려면 어떻게 하지?



좋은 포인트 짚으셨어요. 상황을 정리하면:

- NFS 공유 디렉토리
 - `owner uid=2001`, 권한 `755`
 - 즉 **2001 사용자만 쓰기 가능**, 다른 UID는 읽기만 가능
- Airflow 컨테이너 내부

- ``airflow`` 계정이 ``uid=50000``

그래서 ``50000`` → ``2001`` mismatch 때문에 쓰기 권한 거부가 나는 거죠.

여기서 선택지는 몇 가지가 있어요:

1. 컨테이너 실행 시 UID 매핑

Pod 스펙에서 ``securityContext.runAsUser: 2001`` 를 지정해서 Airflow 프로세스를 2001 UID로 실행하게 만들 수 있습니다.

```
securityContext:
  runAsUser: 2001
  runAsGroup: 2001
  fsGroup: 2001
```

이러면 컨테이너 안에서 ``airflow`` 대신 UID 2001이 쓰기를 할 수 있게 돼요.

다만, 공식 이미지의 기본 ``airflow`` 사용자 대신 그냥 UID만 바꿔 실행하는 방식이라 약간 꿈수 느낌이 있습니다.

2. NFS 디렉토리 권한 완화

공유 디렉토리 퍼미션을 ``775``로 바꾸고, ``airflow`` UID(50000)를 해당 그룹에 넣는 방법.
즉, NFS 서버 쪽에서:

```
chgrp airflow_group /nfs/path
chmod 775 /nfs/path
```

그리고 마운트 시 ``fsGroup: <그룹ID>``를 지정하면, Pod 내부에서 50000 UID라도 그룹 권한으로 쓰기 가능해집니다.

3. Airflow 이미지 커스텀

``airflow`` 계정의 UID를 ``2001``로 덮어씌운 이미지를 새로 빌드하는 방법:

```
FROM apache/airflow:2.9.2
RUN usermod -u 2001 airflow && groupmod -g 2001 airflow
```

이러면 컨테이너 안 `airflow` 계정 UID=2001이 되어, NFS 소유자랑 맞아떨어지죠.
하지만 다른 시스템과 충돌할 수 있으니 주의가 필요합니다.

🔑 정리하면:

- **빠른 해결:** Pod `securityContext`에서 `runAsUser: 2001` 지정
- **운영 안정성:** NFS 쪽에서 권한을 그룹 기반으로 풀고, Pod에 `fsGroup` 적용

혹시 지금 쓰시는 Airflow DAG에서 **공유 디렉토리** 쓰기는 ETL 결과물 저장용인가요? 아니면 단순 로그/캐시 용도인가요? 용도에 따라 제가 권장하는 방법이 달라질 수 있습니다.



"2. NFS 디렉토리 권한 완화" 에 대해서 상세히 알려줘.



1m 53s 동안 생각함

- 사용자의 환경은 RHEL9과 Kubernetes 1.31.1, ArgoCD, Helm, Airflow 2.9.2이며, NFS 디렉토리는 uid 2001에 755 권한 설정되어 있습니다. Airflow가 NFS에 글쓰기 하기 위해서는, 디렉토리 권한을 그룹 쓰기 가능(775)으로 변경하고, 그룹 소유권을 공유 그룹으로 설정해야 합니다. 단계별로 uid/gid를 확인하고, NFS 서버에서 그룹을 생성 후 권한 변경, setgid 비트 적용 및 기본 ACL을 설정해야 합니다.
- `runAsGroup`은 컨테이너 프로세스의 주 그룹 ID를 설정하고, `fsGroup`은 포드에 마운트된 볼륨의 파일에 적용되는 그룹을 설정합니다. `fsGroup`은 지정된 GID에 쓰기 권한을 부여하는 것이 일반적이지만, NFS에서는 `root_squash` 설정에 따라 파일 소유권 변경이 불가능할 수 있습니다.
- Kubernetes에서 `fsGroup`은 kubelet이 볼륨 마운트 시 파일 시스템이 지원하면 GID에 맞게 `chown`과 `chmod`를 시도합니다. 그러나 NFS에서 `root_squash`가 설정되면 권한 변경이 불가능한 경우가 있습니다. 이를 해결하려면 NFS 디렉토리의 그룹을 변경하고, `fsGroup`과 `supplementalGroups`를 조합해 권한을 설정하는 방법을 고려해야 합니다.
- 먼저, 원하는 GID를 선택하고 이를 `fsGroup`으로 설정하여 컨테이너 프로세스가 그룹 멤버십을 갖도록 합니다. 그 후, NFS 공유 경로에서 해당 GID로 소유권을 변경하고, 디렉토리에 `g+w` 및 `setgid` 비트를 적용합니다. 또한, 기본 ACL을 설정할 수 있습니다. Kubernetes에서 `fsGroup`을 Helm 값에 설정하고, 필요한 경우 `supplementalGroups`도 추가합니다.



- 목표는 Airflow(UID 50000)가 NFS 디렉터리에 쓰기 권한을 가지도록 설정하는 것입니다. 이를 위해 그룹 쓰기 권한이 설정된 공유 그룹을 사용합니다. NFS 서버에서 그룹을 설정하고 권한을 조정하는 방법을 설명합니다. 또한 Helm 차트에서 `fsGroup` 설정을 제공하고, 권한 설정 및 검증 방법을 제시합니다.

^ 이크

좋습니다. “그룹 쓰기”로 푸는 방법을 끝까지 깔끔하게 정리해볼게요. 핵심은 **NFS 공유 폴더를 특정 그룹(GID)에 소유시키고, 파드가 그 그룹으로 들어오게 해서(group 권한) 쓰기**를 허용하는 겁니다.

목표 상태

- NFS 서버의 디렉터리: `소유 그룹 = <GID>` 이고 디렉터리 권한 **2775**, 파일 권한 664 이상
- 쿠버네티스 파드: **그룹 ID(<GID>)**로 들어와서 해당 경로에 쓰기 가능
- 신규로 생기는 하위 파일/디렉터리도 **같은 그룹으로 자동 상속(setgid + 기본 ACL)**

1) NFS 서버 쪽 준비

아래에서 ``는 공유 루트(예: `/export/datalake`), ``는 Airflow/작업 파드가 공유할 그룹 ID입니다. (이미 `uid=2001` 사용자가 소유라면, 그 사용자의 **그룹 GID**를 그대로 쓰거나, 별도 공유그룹(예: `3001`)을 새로 만들어도 됩니다.)

1. (선택) 공유 그룹 만들기

```
# 새 그룹을 만들 경우
sudo groupadd -g 3001 datalake_writers
# 이미 적당한 그룹이 있으면 이 단계 생략, <GID>만 그 그룹의 GID로 맞춤
```

2. 공유 디렉터리의 그룹 소유권을 ``로 변경

```
sudo chgrp -R <GID> <PATH>
```

3. 디렉터리에 **setgid 비트**와 그룹 쓰기 권한 주기

setgid(2xxx)를 켜면 그 디렉터리 안에서 새로 만들어지는 파일/폴더의 **그룹이 디렉터리의 그룹으로 상속됩니다.**

```
# 디렉터리는 2775 (rwxrwsr-x)
sudo find <PATH> -type d -exec chmod 2775 {} +
# 기존 파일은 그룹 쓰기 허용(664)로 맞추는 것을 권장
sudo find <PATH> -type f -exec chmod 664 {} +
```

4. 기본 **ACL**로 앞으로 생성될 항목에도 그룹 쓰기 상속

umask가 0022인 워크로드가 많아서 g+w가 빠지는 경우가 흔합니다. 기본 ACL을 걸어두면 “기본 값”으로 g+rw가 내려갑니다(단, 프로세스의 umask가 더 강하게 제한하면 영향을 받습니다).

```
# 현재/기존 항목에 그룹 rwx 부여
sudo setfacl -R -m g:<GID>:rwx <PATH>

# 기본(default) ACL: 앞으로 새로 만들어질 항목에 기본으로 g:rwx
sudo setfacl -R -m d:g:<GID>:rwx <PATH>

# 확인
getfacl <PATH> | sed -n '1,40p'
```

5. /etc/exports 점검

- 일반적으로 ``root_squash`` 유지(보안상 권장). 이 방식은 **root** 권한이 없어도 그룹 권한으로 쓰기가 목적이므로 ``no_root_squash``는 불필요합니다.
- ACL을 쓸 거면 서버/클라이언트가 **ACL 지원**(보통 XFS/EXT4 기본)이고, NFS는 v4.x를 권장합니다.
예) ``rw,sync,root_squash`` 등. (환경 규정에 맞춰 유지)

2) 쿠버네티스(Helm/ArgoCD) 쪽 설정

핵심은 파드가 `<GID>`를 그룹으로 가지고 마운트하도록 하는 것입니다.

(A) 일반 파드/디플로이먼트 예시

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: airflow-web
spec:
  template:
    spec:
      securityContext:
        # 볼륨 루트 소유권 조정 시도 + 파드에 보조그룹으로 <GID> 부여
        fsGroup: <GID>
        fsGroupChangePolicy: "OnRootMismatch" # 전체 재귀 chown 방지(성능)
      containers:
```

```

- name: web
image: apache/airflow:2.9.2
securityContext:
  runAsUser: 50000          # airflow UID 그대로
  runAsGroup: <GID>        # 컨테이너 주 그룹을 <GID>로(권장)
  allowPrivilegeEscalation: false
volumeMounts:
- name: datalake
  mountPath: /datalake
volumes:
- name: datalake
  persistentVolumeClaim:
    claimName: datalake-pvc

```

포인트

- `runAsGroup: <GID>`: 컨테이너 프로세스의 주 그룹을 `<GID>`로. 새 파일의 GID가 `<GID>`로 잡히는 데 도움이됩니다.
- `fsGroup: <GID>`: 파드에 **보조그룹**을 부여하고(파일 접근 가능), 볼륨 마운트시 kubelet이 소유권 정렬을 시도합니다(루트 스쿼시 환경에선 소유권 변경이 실패할 수 있지만, **그룹 권한만으로 충분하면 문제없음**).
- 디렉터리에 `setgid`가 걸려 있으므로 새로 만드는 항목의 그룹이 `<GID>`로 상속됩니다.

(B) Airflow + KubernetesPodOperator(KPO) 작업에도 적용

DAG의 KPO에서 `pod_override`로 보안 컨텍스트를 내려주세요. (예시)

```

from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import
KubernetesPodOperator

write_to_nfs = KubernetesPodOperator(
    task_id="write_to_nfs",
    name="write-to-nfs",
    namespace="airflow",
    image="python:3.11",
    cmds=["bash", "-lc"],
    arguments=[
        # umask 0002로 그룹 쓰기 유지 + 테스트 작성
        "umask 0002 && echo hello > /datalake/test.txt && ls -l
        /datalake/test.txt"
    ],

```

```

volumes=[{
    "name": "datalake",
    "persistentVolumeClaim": {"claimName": "datalake-pvc"}
}],
volume_mounts=[{
    "name": "datalake",
    "mountPath": "/datalake"
}],
pod_override={
    "securityContext": {
        "fsGroup": <GID>,
        "fsGroupChangePolicy": "OnRootMismatch"
    },
    "containers": [{
        "name": "base",
        "securityContext": {
            "runAsUser": 50000,
            "runAsGroup": <GID>
        }
    }]
},
)

```

Airflow Helm 차트에서는 컴포넌트별(`web`, `scheduler`, `workers`, `triggerer`) `securityContext`/`podSecurityContext` 값을 values.yaml로 내려주시면 됩니다. 키 이름은 사용 중인 차트 버전 구조에 맞춰 배치하세요.

3) umask와 파일 권한 유지

- 컨테이너 기본 `umask`가 `0022`이면 새 파일이 `644` (g+w 빠짐)로 생깁니다.
- 그룹이 수정할 가능성이 있으면 `umask 0002` (파일 664/디렉터리 775)로 맞추는 걸 권장합니다.
 - 엔트리포인트 스크립트에서 `umask 0002` 수행
 - 또는 KPO `arguments`/커맨드에 포함
- 기본 ACL을 깔아도 **umask**가 더 보수적이면 g+w가 빠질 수 있으니, **setgid + umask 0002** 콤보가 가장 예측 가능합니다.

4) 빠른 점검 체크리스트

1. 서버

```
# 소유 그룹과 권한 확인
ls -ld <PATH>
getfacl <PATH> | head

# 하위 항목 샘플 확인
find <PATH> -maxdepth 2 -printf "%M %u:%g %p\n" | head
```

2. 파드

```
kubectl exec -it <pod> -- bash -lc '
id
umask
echo "test" > /datalake/_probe.txt
ls -l /datalake/_probe.txt
stat -c "%U:%G %a %n" /datalake/_probe.txt
'
```

- 기대: 그룹이 `**<GID>**`로 표시되고, 쓰기/수정이 자연스럽게 됩니다.

5) 혼한 함정과 팁

- **root_squash**: 기본값 유지(권장). 이 방식은 root로 chown 할 일이 없이 **그룹 권한**으로 해결합니다.
- **fsGroup 실패 로그**: NFS에서 소유권 변경(chown)이 막혀 경고가 찍히기도 합니다. 그래도 **그룹 권한**이 맞으면 실제 쓰기는 정상입니다.
- 기존 파일이 **640/600**처럼 닫혀 있으면, 한 번 `**chmod 664**` 또는 `**setfacl -R -m g:<GID>:rw**`로 정리해 주세요.
- **NFS v4 권장**: ACL/권한 동작이 v3보다 일관적인 편입니다.
- 여러 팀이 공유하면, 그룹을 “업무 영역 단위”로 나누고 `**<GID>**`를 다른 영역과 분리하는 게 권한 관리가 편합니다.

결론(추천 세팅)

- 서버: `**<PATH>**`에 **그룹=<GID>**, `**chmod 2775**`(디렉터리), `**chmod 664**`(파일), `**setfacl -m g:<GID>:rwx -m d:g:<GID>:rwx**`
- 파드: `**runAsGroup: <GID>**`, `**fsGroup: <GID>**`, `**fsGroupChangePolicy: OnRootMismatch**`

- 작업: ``umask 0002`` 유지
- 보안: ``root_squash`` 유지, UID 바꾸지 않음

이렇게 하면 ``airflow(uid=50000)`` 그대로 쓰면서도 **그룹 권한으로 NFS 쓰기**가 안정적으로 풀립니다. 필요하시면 values.yaml 구조에 맞춘 실제 조각(당신이 쓰는 그 Helm 차트 키)에 맞춰 바로 넣을 수 있게 변환해드릴게요.



"1. 컨테이너 실행 시 UID 매핑"에 대해서 자세히 알려줘



좋아요. “컨테이너 실행 시 UID 매핑(runAsUser)”로 NFS의 **uid=2001** 소유 디렉터리에 ``airflow``(기본 uid=50000)도 쓰기가 되게 만드는 방법을 처음부터 끝까지 정리해드릴게요.

핵심 한 줄: **파드를 2001 유저/그룹으로 실행**시키고(= 파일 생성·수정 주체가 2001), 필요하면 **fsGroup**으로 볼륨 접근권을 보조해줍니다.

0) 전제 점검

- NFS 서버의 쓰기 대상 경로 소유자: **uid=2001** (예: ``/export/datalake``)
- NFS는 보통 ``root_squash``가 켜져 있음 → 컨테이너에서 root로 ``chown`` 해도 적용 안 됨
⇒ 애초에 **2001로 실행**해서 “소유자 권한”으로 쓰게 하는 게 깔끔합니다.

1) 쿠버네티스 기본 예시 (Deployment)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: airflow-web
  namespace: airflow
spec:
  template:
    spec:
      # 파드 레벨 보안 컨텍스트
      securityContext:
        fsGroup: 2001
```

볼륨(파일시스템) 접근에 보조 그룹 부여



```

    fsGroupChangePolicy: OnRootMismatch # 전체 재귀 chown 방지(성능 최적화)
containers:
- name: web
  image: apache/airflow:2.9.2
  securityContext:
    runAsUser: 2001 # 컨테이너 프로세스 UID = 2001
    runAsGroup: 2001 # 기본 GID = 2001 (신규 파일 GID)
    allowPrivilegeEscalation: false
    runAsNonRoot: true
  volumeMounts:
  - name: datalake
    mountPath: /datalake # NFS 마운트 위치
volumes:
- name: datalake
  persistentVolumeClaim:
    claimName: datalake-pvc

```

포인트

- ``runAsUser: 2001`` → 컨테이너의 실행 계정 자체가 **2001**이므로 NFS의 2001 소유 디렉터리에 쓰기 가능.
- ``runAsGroup: 2001`` → 생성 파일·디렉터리 GID가 2001로 붙어 **그룹 상속**이 수월.
- ``fsGroup: 2001`` → 파드에 보조 그룹으로 2001을 부여. 일부 스토리지/마운트 권한 접근에 도움.

*NFS가 ``root_squash``라면, `initContainer`로 `root` 권한으로 ``chown`` 하려는 트릭은 실패합니다.
****UID 매핑(2001로 실행)**은 `squash` 영향을 받지 않아 가장 단순·직관적입니다.***

2) Airflow Helm 차트(values.yaml) 반영

Airflow 2.9.2(공식 Helm 차트 기준)라면 컴포넌트별로 아래처럼 넣습니다. (ArgoCD는 이 ``values.yaml``을 GitOps로 배포)

```

# values.yaml (여시)
web:
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001

```

```

    allowPrivilegeEscalation: false
    runAsNonRoot: true

scheduler:
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
    allowPrivilegeEscalation: false
    runAsNonRoot: true

workers:
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
    allowPrivilegeEscalation: false
    runAsNonRoot: true

triggerer:
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
    allowPrivilegeEscalation: false
    runAsNonRoot: true

```

(필요 시) flower 등도 동일 패턴

차트 버전에 따라 키 이름이 `securityContext` / `podSecurityContext`의 위치가 조금 다를 수 있어요.

쓰시는 차트 스키마에 맞게 동일 의미로 배치해주시면 됩니다.

3) KubernetesExecutor / KPO 작업에도 2001 적용

3-1) KubernetesExecutor 기본 pod 템플릿

K8sExecutor가 생성하는 작업 파드도 2001로 띄우려면 **pod_template_file**에 명시합니다.

``pod_template.yaml``:

```
apiVersion: v1
kind: Pod
spec:
  securityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  containers:
  - name: base
    securityContext:
      runAsUser: 2001
      runAsGroup: 2001
```

Airflow 설정(환경변수):

```
AIRFLOW__KUBERNETES__POD_TEMPLATE_FILE=/opt/airflow/pod_template.yaml
```

차트에서 ``env``나 ``extraEnv``로 설정하거나, ``config/airflow.cfg`` 구성 파일로 내려도 됩니다.

3-2) KubernetesPodOperator(KPO)

개별 태스크에 직접 지정:

```
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import
KubernetesPodOperator

task = KubernetesPodOperator(
    task_id="write_nfs",
    name="write-nfs",
    namespace="airflow",
    image="python:3.11",
    cmds=["bash", "-lc"],
    arguments=["umask 0002 && echo hello > /datalake/ok.txt && ls -l
/datalake/ok.txt"],
    volumes=[{"name": "datalake", "persistentVolumeClaim": {"claimName":
"datalake-pvc"}}],
    volume_mounts=[{"name": "datalake", "mountPath": "/datalake"}],
```

```

    pod_override={
      "securityContext": {"fsGroup": 2001, "fsGroupChangePolicy":
"OnRootMismatch"},
      "containers": [{
        "name": "base",
        "securityContext": {"runAsUser": 2001, "runAsGroup": 2001}
      }]
    },
  )

```

4) umask/권한 디테일

- 컨테이너 기본 `umask`가 `0022`면 새 파일이 `644`로 생성되어 ****그룹 쓰기(g+w)****가 빠질 수 있습니다.
 - 협업/후속 수정이 필요하면 `umask 0002` 권장(파일 664, 디렉터리 775).
 - 엔트리포인트 스크립트나 KPO `arguments`에서 `umask 0002`로 고정.
- NFS 디렉터리에 ****setgid(2775)****가 걸려 있으면 새 항목의 ****그룹이 상위 디렉터리의 GID(=2001)****로 자동 상속됩니다. (이건 서버 쪽 설정 이슈이지만, 있으면 훨씬 깔끔)

5) 검증 절차

배포 후, 아무 한 파드에서:

```

kubectl exec -it <pod> -n airflow -- bash -lc '
id
umask
echo "probe" > /datalake/_probe.txt
ls -l /datalake/_probe.txt
stat -c "%u:%g %a %n" /datalake/_probe.txt
'

```

기대 결과

- `uid=2001 gid=2001`
- `_probe.txt`가 **2001:2001**이고 퍼미션이 `-rw-rw-r--`(664)이면 베스트.

6) 혼한 이슈/주의점

1. \$AIRFLOW_HOME 권한

- 기본은 ``/opt/airflow``. 이미지 내부 파일 소유가 50000 기준일 수 있음.
- 보통 읽기엔 문제 없고, **로그/데이터 경로는 PVC로 마운트**하므로 현업에선 큰 문제 드뭅니다.
- 그래도 특정 플러그인/로컬 DB 등으로 ``/opt/airflow``에 쓰기를 한다면, 해당 경로를 PVC로 빼거나 권한(또는 ``fsGroup``)을 확보하세요.

2. root_squash 환경에서 chown 안 먹힘

- expected behavior. ****UID 매핑(2001로 실행)****이면 chown 없어도 정상 동작합니다.

3. runAsGroup 바꾸면 root 그룹 접근 상실?

- Airflow 이미지의 필수 파일은 일반적으로 world-readable입니다.
- 그래도 특정 경로가 ``root:root 750`` 같은 경우가 있으면 접근 불가가 날 수 있으니, 실제 파이프라인 기준으로 한 번 점검하세요. 필요하다면 ``supplementalGroups``로 추가 그룹을 부여할 수도 있습니다(권장되진 않음).

4. KubernetesExecutor/KPO마다 보안컨텍스트 따로

- 웹/스케줄러/워커/트리거러 컴포넌트와 **작업 파드(K8sExecutor/KPO)** 설정이 분리됩니다.
- “모든 작업이 2001”을 원하면 **pod_template_file** 또는 **KPO의 pod_override**를 잊지 말기.

7) 언제 이 방법이 ‘최선’인가?

- **NFS가 팀 표준**이고, 디렉터리 소유가 이미 **uid=2001**로 고정되어 있음.
- 운영 정책상 ``root_squash`` 유지가 필수.
- 이미지 변경 없이 **운영 배포(yaml)만으로** 문제를 풀고 싶음.

반대로, NFS 서버 권한을 그룹 기반(2775/setgid/ACL)으로 잘 설계했다면 “그룹 권한 방식(=fsGroup 중심)”이 더 유연합니다. 그래도 “UID 매핑”은 빠르게 확실하게 먹히는 해법이라 현장에서 자주 씁니다.

필요하시면 지금 쓰시는 Airflow Helm 차트 값 구조(키 이름)로 바로 붙여 넣는 **values.yaml** 조각까지 맞춰서 드릴게요. 현재 ArgoCD로 싱크 중이니, ``Application``의 ``values``나 ``valuesFiles``에 넣는 위치만 알려주시면 거기에 맞춰 정리해드립니다.



"3. Airflow 이미지 커스텀" 에 대해 상세히 알려줘.



18s 동안 생각함

- The earlier assistant suggested using groupadd with GID 0 for the 'airflow' group, which duplicates the root group's GID. In Linux, it's unusual to have multiple group names mapping to the same GID, though it's possible technically. I'll clarify: to change the primary group, we should use usermod and update file ownerships accordingly.

🕒 완료

좋습니다. “이미지 커스텀으로 `airflow` 계정의 UID를 2001로 바꿔서 NFS(소유자 uid=2001)에 그대로 쓰기” 방법을 처음부터 끝까지 정리해드릴게요.

핵심은 베이스 이미지의 `airflow(uid=50000, gid=0)`를 `uid=2001, gid=2001`로 바꾸고 필요한 디렉터리의 소유권을 맞춰주는 것입니다.

1) 최소 동작 버전(가볍게)

```
# Airflow 베이스 이미지는 실제 레지스트리 경로로 교체
FROM apache/airflow:2.9.2-python3.11

# 1) 그룹(2001) 만들고
RUN groupadd -g 2001 airflowgrp \
# 2) 사용자 UID/GID를 2001로 변경
&& usermod -u 2001 -g 2001 airflow \
# 3) 홈/작업 디렉터리 주요 경로 소유권 정리
&& chown -R 2001:2001 /home/airflow /opt/airflow

USER airflow
WORKDIR /opt/airflow
```

- 이걸로 컨테이너 안 `airflow` 사용자가 **uid=2001**이 됩니다.
- NFS에 마운트만 잘 되면(예: `/dags`, `/logs`, `/plugins`, `/data` 등) 소유자 권한으로 쓰기가 가능해집니다.

팁: NFS가 `root_squash`라도 상관없습니다. 우리는 애초에 **2001** 사용자로 실행하므로 chown이 필요 없습니다.

2) 운영형 버전(조금 더 안전하게)

Airflow가 실제로 쓰는 경로를 폭넓게 잡고 권한도 함께 정리합니다. (이미지마다 조금 다르지만 아래가 보통 안전합니다)

```
FROM apache/airflow:2.9.2-python3.11

# 필요한 패키지(옵션): acl, tini 등
# RUN apt-get update && apt-get install -y --no-install-recommends acl && rm -rf /var/lib/apt/lists/*

# GID/UID 변경
RUN groupadd -g 2001 airflowgrp \
  && usermod -u 2001 -g 2001 airflow \
  && mkdir -p /opt/airflow/{dags,logs,plugins} \
    /home/airflow/.cache/pip \
    /var/lib/airflow \
  && chown -R 2001:2001 /opt/airflow /home/airflow /var/lib/airflow

# (선택) 기본 umask 개선 - 그룹 쓰기 유지(664/775)
# set -o errexit; set -o nounset; 등은 취향
RUN echo 'umask 0002' >> /etc/profile.d/airflow-umask.sh

USER airflow
WORKDIR /opt/airflow
```

- `/opt/airflow/{dags,logs,plugins}`는 Helm 차트에서 PVC로 마운트하는 경우가 많습니다. 그래도 **이미지 레벨**에서 소유권을 맞춰두면 초기 부팅/헬스체크에서 불필요한 권한 이슈를 줄입니다.
- `umask 0002`는 협업(그룹 쓰기)을 고려한 설정입니다.

3) DAG/로그/플러그인 경로를 PVC로 쓸 때

Helm values에서 보통 다음처럼 마운트합니다(차트 구조에 맞춰 조정):

```
dags:
  persistence:
```



```
enabled: true
existingClaim: dags-pvc
# mountPath: /opt/airflow/dags (차트 기본 경로 확인)
```

logs:

```
persistence:
  enabled: true
  existingClaim: logs-pvc
```

plugins:

```
persistence:
  enabled: true
  existingClaim: plugins-pvc
```

- 이 경우 PVC 자체의 파일 소유/권한은 **스토리지 제공자** 쪽 설정에 좌우됩니다.
- 우리가 이미지에서 UID=2001로 맞췄다면, PVC도 **소유자 또는 그룹 기반으로 쓰기 가능하게** 준비되어야 합니다.
 - (소유자 정책) PVC 내 기본 소유자가 2001이면 그대로 OK
 - (그룹 정책) PVC 디렉터리 쪽에 ``setgid(2775)` + `fsGroup`/`runAsGroup`` 조합을 쓰면 협업에 더 유연

4) ArgoCD/Helm 연동 시 흔한 설정

커스텀 이미지를 배포한 뒤, 보안 컨텍스트는 크게 건드릴 게 없습니다. 그래도 다음 정도는 추천:

```
# values.yaml 예시
web:
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
    runAsNonRoot: true
    allowPrivilegeEscalation: false
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch

scheduler:
  securityContext:
    runAsUser: 2001
```

```

    runAsGroup: 2001
    runAsNonRoot: true
    allowPrivilegeEscalation: false
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch

```

workers/triggerer/flower 등도 동일 패턴

- 이미 이미지가 2001로 바뀌어도, **작업 파드(KubernetesExecutor/KPO)**는 별도 파드이니 **pod_template_file** 또는 KPO `pod_override`로 2001을 명시해주는 걸 잊지 마세요.

5) KPO/Executor에도 일관 적용

KubernetesExecutor

`pod_template.yaml`:

```

apiVersion: v1
kind: Pod
spec:
  securityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch
  containers:
  - name: base
    securityContext:
      runAsUser: 2001
      runAsGroup: 2001

```

`AIRFLOW__KUBERNETES__POD_TEMPLATE_FILE=/opt/airflow/pod_template.yaml`

KubernetesPodOperator

```

KubernetesPodOperator(
    ...,
    pod_override={
        "securityContext": {"fsGroup": 2001, "fsGroupChangePolicy":
"OnRootMismatch"},
        "containers": [{

```

```

    "name": "base",
    "securityContext": {"runAsUser": 2001, "runAsGroup": 2001}
  }]
},
)

```

6) 검증 커맨드(배포 후)

```

kubectl exec -it <pod> -n airflow -- bash -lc '
id
umask
echo hello > /opt/airflow/logs/_probe.txt
ls -l /opt/airflow/logs/_probe.txt
stat -c "%u:%g %a %n" /opt/airflow/logs/_probe.txt
'

```

- 기대: `uid=2001 gid=2001`, 파일 `2001:2001`, 권한 `664` 또는 `rw-rw-r--`.

7) 자주 겪는 함정과 회피책

- 베이스 이미지의 그룹이 **GID 0(root)**인 문제
 - 기본 Airflow 이미지는 `airflow` 사용자의 기본 그룹이 **0(root)**입니다. 우리가 `-g 2001`로 바꾸면 **root** 그룹 권한에 기대던 동작이 드물게 깨질 수 있습니다. 보통은 문제 없지만, 사내 커스텀 스크립트가 `/root` 전용 권한을 가정한다면 점검이 필요합니다.
- 이미지 전체 **chown**은 비용 큼
 - `chown -R`는 레이어를 크게 만들고 빌드 시간도 길어집니다. 실제로 쓰는 경로(`/opt/airflow`, `/home/airflow`) 위주로 최소화하세요.
- 플러그인/드라이버가 외부 경로에 쓰기
 - Spark/StarRocks/임시 디렉터리(`/tmp` 등) 권한도 가끔 이슈입니다. 보통 world-writable(`1777`)이라 문제 없지만, 사내 hardened 이미지면 한번 확인하세요.
- OpenShift처럼 “임의 UID” 정책의 클러스터
 - 그런 환경이면 이미지 내 고정 UID 전략보다 **fsGroup/RunAsGroup** 기반이 더 적합할 수 있습니다. (당신 환경은 RHEL9 일반 K8s라 고정 UID 전략 문제없어 보입니다.)
- **umask**에 잡아먹히는 **g+w**

- 협업 파일을 남길 태스크라면 ``umask 0002``(또는 기본 ACL)로 그룹 쓰기 보장하세요.

8) 요약(언제 이 방법이 좋다)

- 사내 NFS가 uid=2001 소유로 굳어 있고, Airflow/작업 파드가 그 소유자로 그대로 쓰기를 해야 한다.
- 클러스터 정책에 제약이 없고, 이미지 한 번 커스텀해두면 운영이 편해진다는 판단일 때.

만약 팀 단위 공유/협업이 많다면, “그룹 기반(2775+ACL)+`fsGroup`” 방식(=이전에 설명드린 2번)이 장기적으로 더 유연합니다. 그래도 이미지 커스텀은 가장 직관적이고 확실한 해법이라 ETL 파이프라인에서 자주 채택합니다.

필요하시면 당신이 쓰는 Helm 차트/ArgoCD `Application` 구조에 맞춰 `values.yaml` 조각과 `Dockerfile`를 바로 붙여 쓸 수 있게 딱 맞춰서 다시 뽑아드릴게요.



3번이 좋은것 같은데.

아래 처럼 한잖아, 근데 이미 airflow 사용자가 있지 않아?

GID/UID 변경

```
RUN groupadd -g 2001 airflowgrp \  
&& usermod -u 2001 -g 2001 airflow \
```



네, 맞아요. 이미 `airflow` 사용자는 베이스 이미지에 존재합니다(기본 UID=50000, 기본 GID=0/root).

지금 적으신 ``usermod -u 2001 -g 2001 airflow``는 그 “기존” 사용자의 UID/GID를 2001로 바꾸는 명령이에요.

딱 두 가지만 조심하면 됩니다:

1. 그룹(2001) 존재/충돌 처리
2. UID/GID 변경 뒤, 필요한 경로의 소유권 재정렬(chown)

아래처럼 “운영형 안전 스니펫”으로 가시면 깔끔합니다.



권장 Dockerfile 스니펫 (충돌 안전 + 최소 chown)

```
FROM apache/airflow:2.9.2-python3.11

USER root
SHELL ["/bin/bash", "-euxo", "pipefail", "-c"]

# 변수
ARG TARGET_UID=2001
ARG TARGET_GID=2001
ARG AIRFLOW_USER=airflow
ARG AIRFLOW_HOME=/opt/airflow

# 1) 그룹 준비: TARGET_GID가 없으면 만들고, 있으면 재사용
# 이미 같은 GID를 가진 그룹명이 있으면 그 그룹을 사용
RUN if ! getent group "${TARGET_GID}" >/dev/null; then \
    groupadd -g "${TARGET_GID}" airflowgrp; \
fi

# 2) airflow의 GID를 TARGET_GID로 변경
# - airflow 그룹이 원래 GID=0(root)인 이미지가 많습니다.
# - 이미 TARGET_GID에 어떤 그룹이 있으면 그 그룹으로 변경됨
RUN groupmod -g "${TARGET_GID}" "$(getent passwd ${AIRFLOW_USER} | cut -d: -f1)"

# 3) airflow의 UID를 TARGET_UID로 변경
RUN usermod -u "${TARGET_UID}" -g "${TARGET_GID}" "${AIRFLOW_USER}"

# 4) 자주 쓰는 경로만 최소 chown (전체 루트에서 -R은 레이어 커지고 느립니다)
# 필요에 따라 추가: /home/airflow, /var/lib/airflow 등
RUN mkdir -p ${AIRFLOW_HOME}/{dags,logs,plugins}
/home/${AIRFLOW_USER}/.cache/pip \
&& chown -R ${TARGET_UID}:${TARGET_GID} ${AIRFLOW_HOME} /home/${AIRFLOW_USER}

# 5) (선택) 협업 고려 시 그룹 쓰기 유지
RUN echo 'umask 0002' > /etc/profile.d/airflow-umask.sh

USER ${AIRFLOW_USER}
WORKDIR ${AIRFLOW_HOME}
```

포인트

- `groupmod`를 써서 기존 **airflow** 그룹의 **GID**를 바꾸는 게 가장 일관됩니다.
- 이미 `GID=2001`인 그룹이 존재하면 그 그룹으로 붙습니다. (위 스니펫은 `groupadd` 전 `getent group`으로 충돌 방지)
- 모든 파일을 `chown -R /`로 손대지 마세요. 꼭 쓰는 경로만 정리(`/opt/airflow`,
/home/airflow` 등). 빌드가 가벼워집니다.`

왜 **chown**이 필요하죠?

`airflow`의 UID/GID가 바뀌면, 이미지 레이어 내에 기존 **UID=50000**으로 소유된 파일들이 남아 있을 수 있어요.

Airflow가 실제로 쓰기(**write**) 하는 경로(보통 `/opt/airflow/logs`, 플러그인 캐시 등)만 새 **UID/GID로 chown**하면 충분합니다.

DAG/LOG/PLUGINS를 PVC로 마운트한다면, 컨테이너 내부 기본 파일은 “읽기 전용”이면 그 자체로 문제될 일이 거의 없습니다.

values.yaml 쪽(실행 컨텍스트)도 맞춰주면 안정적

이미지에 UID/GID를 2001로 바꿨더라도, 파드 실행 컨텍스트도 정리해두면 더 예측 가능해요.

```
web:
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
    runAsNonRoot: true
    allowPrivilegeEscalation: false
  podSecurityContext:
    fsGroup: 2001
    fsGroupChangePolicy: OnRootMismatch

scheduler:
  securityContext:
    runAsUser: 2001
    runAsGroup: 2001
  podSecurityContext:
    fsGroup: 2001
```

```
fsGroupChangePolicy: OnRootMismatch
```

```
# workers/triggerer/flower도 동일 패턴
```

작업 파드(KubernetesExecutor/KPO)도 **pod_template_file**이나 **pod_override**로 **runAsUser/Group=2001**을 주는 걸 잊지 마세요. (Airflow 본체와 작업 파드는 별개로 뜹니다.)

컨테이너 안에서 검증

```
id airflow
getent passwd airflow
getent group 2001 || true
ls -ld /opt/airflow /opt/airflow/logs /home/airflow
```

- 기대: `uid=2001(airflow) gid=2001(...)`
- `/opt/airflow`와 `/home/airflow` 소유자가 `2001:2001`

흔한 질문/함정 정리

- **Q. 원래 airflow 그룹이 GID=0(root)였는데 바뀌도 되나요?**
대부분 문제 없습니다. 이미지 내부 파일 권한은 대개 world-readable이고, 실행엔 지장 없어요.
혹시 특이하게 root 그룹 권한을 가정한 스크립트가 있다면 점검 필요.
- **Q. NFS가 root_squash인데 chown이 안 먹어요**
괜찮습니다. 우리는 컨테이너 내부 실행 계정을 **2001로 고정**했기 때문에, NFS 소유자(2001)로 바로 쓰기 됩니다.
- **Q. 전체 경로를 재귀 chown 해야 하나요?**
아니요. 쓰는 경로만 최소한으로 처리하세요. 나머지는 PVC 마운트가 덮어씁니다.
- **Q. OpenShift처럼 임의 UID 정책이면?**
그런 환경은 고정 UID 전략이 제약될 수 있어 **fsGroup 중심**이 낫습니다. (지금은 RHEL9 일반 K8s라 현재 선택 OK)

필요하시면 지금 운영 중인 Helm 차트 스키마(키 이름 구조)에 딱 맞춘 `values.yaml` 조각과, 사내 Nexus 경로 기준의 `FROM nexus.../apache/airflow:2.9.2-...` 형태로 바로 붙여 쓸 `Dockerfile` 버전으로 다시 정리해드릴게요.



그러면 베이스 airflow 는 놔두고, dag 실행할때 kpo로 해서 실행할때 새롭게 만든 이미지(# GID/UID 변경
 RUN groupadd -g 2001 airflowgrp \
 && usermod -u 2001 -g 2001 airflow \)를 띄우는것도 방법
 아니야?
 목적은 dag에서 nfs의 특정 path 에 2001 유저로 되어 있는 권한
 에 write하는 거니깐.



맞습니다. 그 방식, 깔끔해요.

Airflow 본체(Web/Scheduler/Worker)는 그대로 두고, DAG 태스크만 KPO로 “UID=2001 이미지”를 써서 실행하면 목적(특정 NFS 경로에 2001 소유로 쓰기) 정확히 달성됩니다.

아래처럼 정리해둘게요.

어떻게 구성하나요?

1) 태스크 전용 커스텀 이미지

베이스 Airflow 이미지는 그대로 두고, 작업 파드에서 쓸 전용 이미지를 따로 빌드합니다.

```
# 예: python 유틸 이미지에 UID/GID=2001 세팅
FROM python:3.11-slim

# 안전모드 셸
SHELL ["/bin/bash", "-euxo", "pipefail", "-c"]

# airflow와 상관없는 "작업 전용 유저"여도 OK (이름은 아무거나)
ARG TARGET_UID=2001
ARG TARGET_GID=2001
ARG USER_NAME=worker

# 그룹/유저 생성 (이미 존재하면 재사용)
RUN if ! getent group "${TARGET_GID}" >/dev/null; then groupadd -g
"${TARGET_GID}" appgrp; fi \
&& if ! id -u "${USER_NAME}" >/dev/null 2>&1; then useradd -u "${TARGET_UID}"
-g "${TARGET_GID}" -m "${USER_NAME}"; fi \
```



```
&& mkdir -p /work && chown -R ${TARGET_UID}:${TARGET_GID} /work \
&& echo 'umask 0002' > /etc/profile.d/umask.sh
```

```
USER ${USER_NAME}
WORKDIR /work
```

포인트

- *Airflow 이미지가 아니어도 됩니다. 태스크에 필요한 **runtime**(python, bash, spark 등)만 담으면 충분합니다.*
- *UID/GID를 **2001**로 고정 → NFS(소유자 2001)에서 소유자 권한으로 바로 쓰기.*
- *협업/권한 유지 위해 `umask 0002` 추천.*

원하시면 `FROM apache/airflow:2.9.2-python3.11` 기반으로 만들 수도 있어요. 다만 KPO는 **Airflow 런타임 불필요**한 경우가 많아 더 가벼운 이미지가 이점입니다.

2) KPO에서 전용 이미지 사용 + 보안 컨텍스트

DAG에서 그 태스크만 해당 이미지를 사용하고, 필요한 PVC(NFS)만 마운트합니다.

```
from airflow import DAG
from airflow.utils.dates import days_ago
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import
KubernetesPodOperator

with DAG(
    dag_id="nfs_write_with_uid2001",
    start_date=days_ago(1),
    schedule=None,
    catchup=False,
) as dag:

    write_nfs = KubernetesPodOperator(
        task_id="write_nfs",
        name="write-nfs",
        namespace="airflow", # 환경에 맞게
        image="nexus.local:5000/team/py311-uid2001:latest", # 방금 만든 전용 이미
        cmds=["bash", "-lc"],
        arguments=[
            # umask로 g+w 유지; 테스트 파일 작성
```

```

        "umask 0002 && echo 'hello' > /data/nfs/test.txt && ls -l
/data/nfs/test.txt && id && umask"
    ],
    volumes=[{
        "name": "datalake",
        "persistentVolumeClaim": {"claimName": "datalake-pvc"} # NFS PVC
    }],
    volume_mounts=[{
        "name": "datalake",
        "mountPath": "/data/nfs" # 원하는 경로
    }],
    # 중요: 보안 컨텍스트
    pod_override={
        "securityContext": {
            "fsGroup": 2001, # 접근/소유권 상속 보조
            "fsGroupChangePolicy": "OnRootMismatch"
        },
        "containers": [{
            "name": "base",
            "securityContext": {
                "runAsUser": 2001, # 컨테이너 프로세스 UID
                "runAsGroup": 2001, # 기본 GID
                "allowPrivilegeEscalation": False,
            }
        }]
    },
    # 레지스트리 인증이 필요하면:
    # image_pull_secrets=[k8s.V1LocalObjectReference(name="regcred")]
)

```

포인트

- **이미지 자체가 2001**이더라도, `runAsUser/runAsGroup=2001`을 명시하면 더 예측 가능해요 (클러스터 정책/SC에 따라 기본 USER가 무시되는 경우도 있음).
- `fsGroup=2001`은 NFS 마운트 접근 보조 및 그룹 상속에 도움. (서버 쪽에 `setgid/ACL`이 걸려 있으면 더 매끄럽습니다.)
- 이 태스크만 전용 이미지와 권한을 씁니다. Airflow 본체에는 영향 없음.

3) 언제 이 접근이 최적?



- Airflow 본체(웹/스케줄러/워커)는 **현행 운영 그대로** 유지하고,
- **일부 DAG 태스크만** 특정 NFS 경로에 **2001 소유자로** 쓰기가 필요할 때.
- 팀/업무별로 서로 다른 “쓰기 정책”이 필요할 때도 태스크 단위로 이미지/권한을 갈라 쓰기 편합니다.

4) 체크리스트

- ☐ **NFS 서버**: 대상 디렉터리 소유자 uid=2001, 필요 시 setgid(2775)/ACL 기본값 구성(선택)
- ☐ **커스텀 작업 이미지**: UID/GID=2001, ``umask 0002``
- ☐ **KPO**: 전용 이미지 사용, ``runAsUser=2001``, ``runAsGroup=2001``, ``fsGroup=2001``, PVC 마운트
- ☐ **검증**:

```
kubectl exec -it <task-pod> -- bash -lc 'id; umask; stat -c "%u:%g %a %n" /data/nfs/test.txt'
```

기대: ``2001:2001``, 권한 664(또는 g+w 유지)

5) 장단점 요약

- **장점**
 - Airflow 코어는 손대지 않음(업그레이드/운영 리스크 ↓).
 - 태스크별로 runtime·권한·패키지를 독립 관리 가능(데이터레이크 파이프라인에 유리).
 - ``root_squash`` 환경에서도 ****소유자(2001)****로 바로 쓰기.
- **주의**
 - 이미지가 여러 개 생김 → 태스크별 **이미지 버전 관리** 필요.
 - 같은 DAG에 **다른 권한 정책**을 쓰면 pod_template/KPO 설정 혼재 주의.

원하시면 지금 쓰시는 **ArgoCD Application + Helm values** 구조에 딱 맞춘 예시(YAML 조각 & Dockerfile)를 바로 뽑아드릴게요.

레지스트리 경로, PVC 이름, 네임스페이스만 알려주시면 실행 가능한 상태로 맞춰드립니다.