

ArgoCD 경고 해결 방법



argocd에서 airflow 를 kubernetes 에 배포 했는데, 다음과 같은 warning 이 나는데, 지금 iclhk8s-prod-dl-ops-airflow-pdp-dev-airflow 이것을 사용중이거든.

SharedResourceWarning, ClusterRoleBinding/airflow-pre-checker-binding is part of applications argocd/iclhk8s-prod-dl-ops-airflow-pdp-dev-airflow and iclhk8s-prod-dl-ops-airflow-smartdq-dev-airflow

경고의 뜻부터 정리할게요.

SharedResourceWarning

Argo CD가 두 개의 Application이 **동일한 리소스**(여기서는 **ClusterRoleBinding/airflow-pre-checker-binding**)를 동시에 “소유”하려고 한다고 판단할 때 뜹니다.

지금 **iclhk8s-prod-dl-ops-airflow-pdp-dev-airflow** 와 **iclhk8s-prod-dl-ops-airflow-smartdq-dev-airflow** 가 둘 다 이 ClusterRoleBinding을 관리 대상으로 잡고 있어요. (ClusterRoleBinding은 클러스터 범위라 이름이 겹치면 바로 충돌 후보가 됩니다.)

핵심 선택지는 두 가지예요:

A안) 공유 리소스를 따로 분리(One owner)

“공유되는” ClusterRole/ClusterRoleBinding을 **별도 App/Release** 로 떼어내고, 각 Airflow 앱에서는 생성하지 않도록 꺼버립니다.

어떻게

1. “shared” 전용 Helm 차트(혹은 Kustomize)로 아래만 배포

- **ClusterRole**
- **ClusterRoleBinding** (예: **airflow-pre-checker-binding-global** 같이 명확한 이름)

2. 각 Airflow 앱(= 두 개의 dev 앱)에서는 해당 RBAC 생성 옵션을 **비활성화**

- 차트 값 예시(차트마다 키 이름 다름; 보통 이런 류가 있습니다):

```
yaml
rbac:
  create: false          # 또는 preChecker.rbac.create: false, web.rbac.create: false 등
```

3. Argo CD App-of-Apps나 Sync Wave로 **shared** → **각 Airflow** 순으로 적용되게 정렬

- shared 매니페스트에:

```
yaml

metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"
```

- 각 Airflow 앱에는 "1" 이상을 부여

장점

- 소유권 충돌이 깔끔히 사라짐(단일 오너).
- 이후 Airflow 앱을 몇 개 더 만들어도 충돌 없음.

주의

- 사용하는 Airflow 차트에서 "그 RBAC을 끌 수 있는 스위치"가 실제로 있는지 확인 필요.

B안) 리소스 이름을 각 릴리스마다 유니크하게(Per-release)

각 Airflow 릴리스가 서로 다른 이름의 ClusterRoleBinding을 만들게 합니다.
차트가 `fullnameoverride/nameoverride`를 RBAC 이름에도 반영한다면 가능해요.

예시

```
yaml

# pdp-dev
fullnameoverride: iclhk8s-prod-dl-ops-airflow-pdp-dev
# smartdq-dev
fullnameoverride: iclhk8s-prod-dl-ops-airflow-smartdq-dev
```

그 결과 RBAC 이름이 `iclhk8s-prod-dl-ops-airflow-pdp-dev-pre-checker-binding` 처럼 달라지면 충돌이 없어집니다.

장점

- 앱 분리 유지, 의존성 없음.

주의

- 경고의 대상 이름이 현재 `airflow-pre-checker-binding` 으로 고정돼 보입니다.
차트가 이름을 하드코딩했다면 B안은 불가 → A안으로 가야 합니다.

지금 당장 점검할 5분 체크리스트

1. 누가 붙잡고 있나 확인

```
bash

kubectl get clusterrolebinding airflow-pre-checker-binding -o yaml | \
  yq '.metadata.labels, .metadata.annotations'
```

여기서 `argocd.argoproj.io/instance`(또는 `app.kubernetes.io/instance`) 라벨/어노테이션이 두 앱 모두 찍혀 있거나, 한 앱이지만 둘 다 매니페스트에 포함되어 있으면 SharedResourceWarning이 납니다.

2. 차트 값에서 RBAC 토큰 찾기

Airflow 차트 values.yaml에서 아래 류의 스위치를 검색:

- `rbac.create`
- `precheck.rbac.create`, `pre-upgrade-check.rbac.create`
- `webserver.rbac.create`, `scheduler.rbac.create` ...(차트에 따라 다름)

3. 전략 결정

- 차트가 이름 변경 불가/하드코딩 → **A안(공유 리소스 분리)**
- 이름 변경이 RBAC에도 반영 가능 → **B안(릴리스별 유니크 이름)**

4. 마이그레이션 순서(권장 A안 기준)

- shared App 배포(ClusterRole/Binding만)
- 두 Airflow App values에서 RBAC 생성 `false` 로 변경 → Sync
- 기존 `airflow-pre-checker-binding` 에 찍힌 잘못된 instance 라벨이 있으면 한 번 삭제 후(리소스만), **shared App** 으로 재생성되게 하세요:

```
bash

kubectl delete clusterrolebinding airflow-pre-checker-binding
# 이후 shared 앱 Sync
```

- 두 Airflow 앱 Sync (이제 해당 RBAC은 만들지 않으니 경고 사라짐)

덧: 임시로 “무시”하는 건 근본 해결이 아님

`resource.exclusions` 나 `ignoreDifferences`로 경고를 덮을 수는 있지만, **소유권 충돌 자체는 남습니다.**
RBAC 같은 클러스터 스코프는 **단일 오너 원칙**으로 정리하는 게 안전합니다.

필요하시면, 지금 쓰시는 Airflow 차트(리포/버전) 알려주시면

- A안: “shared-rbac” 미니 차트/Kustomize 예시,
- B안: values 패치 예시(정확한 키로),
바로 손에 잡히게 만들어 드리겠습니다.



argocd에서 kubectl에 airflow를 배포했는데, airflow-webserver 에서 STATUS가 OutOfSync로 계속 나오는데, argocd에서 리소스를 diff해보면 checksum/webserver-secret-key 이부분이거든, 왜 그러지? 다른 리소스들은 모두 초록색(정상) 상태로 되거든.

이건 거의 100% “헬름이 매 렌더마다 만들어내는 랜덤 값” 때문입니다.

airflow-webserver Deployment 템플릿에 보통 이런 주석이 있어요:

annotations: checksum/webserver-secret-key: <해시>

이 해시는 **웹서버 시크릿(Flask SECRET_KEY 등)** 템플릿 내용을 해시한 값이고, 차트가 **randAlphaNum** 같은 함수로 키를 “자동 생성”하도록 돼 있으면 **매번 다른 값**이 렌더됩니다.

결과: 아르고가 “라이브=이전 해시, 디자인어드=새 해시”로 읽고 **OutOfSync**가 계속 뜹니다. (다른 리소스는 고정이어서 초록색)

간단히 정리하면:

- 증상: **Deployment/airflow-webserver**의 **spec.template.metadata.annotations.checksum/webserver-secret-key**만 계속 바뀜
- 원인: 차트가 **webserver secret key**를 매 렌더마다 새로 생성 → 체크섬이 매번 달라짐

빠른 확인(3분)

```
bash

# 1) 라이브에 박힌 체크섬
kubectl -n <ns> get deploy airflow-webserver \
  -o jsonpath='{.spec.template.metadata.annotations.checksum\webserver-secret-key}'; echo

# 2) 아르고가 렌더한 디자인어드(템플릿) 체크섬
# (CLI가 가능하다면)
argocd app manifests <APP_NAME> | \
  yq '. | select(.kind=="Deployment" and .metadata.name=="airflow-webserver") | \
    .spec.template.metadata.annotations."checksum/webserver-secret-key"'
```

두 값이 매번 달라지면 “자동 생성” 케이스가 맞습니다.

해결 전략

A안(권장): 시크릿을 고정해서 쓰기

1. 시크릿을 직접 만들어 둡니다.

```
bash

# 길고 랜덤한 키 하나 생성 (예: 64바이트)
KEY=$(openssl rand -base64 64)

kubectl -n <ns> create secret generic airflow-webserver-secret \
  --from-literal=webserver-secret-key="$KEY"
```

2. 헬름 values에서 “기존 시크릿 사용”으로 돌립니다. (차트마다 키 이름이 조금씩 다릅니다. 대표적으로 아래 형태 중 하나)

```
yaml

# 예시 1: existingSecret 패턴
webserver:
  existingSecret: airflow-webserver-secret
  existingSecretKey: webserver-secret-key
```

```
# 예시 2: 직접 값 고정 (차트가 지원하면)
webserver:
  secretKey: "여기에_고정_문자열" # 값을 명시하면 randAlphaNum 사용 안 함
```

정확한 values 키가 다르면 알려주세요. 쓰시는 Airflow 차트 리포/버전 보고 딱 맞는 키로 바꿔 드리겠습니다.

효과: 체크섬이 더 이상 매번 바뀌지 않아 **OutOfSync**가 사라집니다.
(시크릿 값이 고정 → 해시도 고정)

B안: 아르고에서 그 필드만 무시

근본 해결은 아니지만, 운영상 허용할 수 있으면 ignoreDifferences로 해당 경로를 제외할 수 있습니다.

```
yaml

# Argo CD Application spec에 추가
spec:
  ignoreDifferences:
    - group: apps
      kind: Deployment
      name: airflow-webserver
      namespace: <ns>
      jsonPointers:
        - /spec/template/metadata/annotations/checksum~1webserver-secret-key
```

참고: ~1는 슬래시 이스케이프입니다.

주의: 이렇게 하면 웹서버 시크릿이 진짜로 바뀌어야 할 때도 “차이 감지”가 안 됩니다. 가능하면 A안으로 가시는 게 안전합니다.

왜 웹서버 시크릿이 중요한가요?

Airflow 웹서버(Flask)는 세션/CSRF 보호를 위해 **SECRET_KEY**를 씁니다.
키가 바뀌면 모든 세션이 무효화되고, 쿠버네티스 롤링 시에도 예상치 못한 로그인 끊김이 생길 수 있어요.
운영에선 **한 번 정하고 고정**이 맞습니다.

결론

- **checksum/webserver-secret-key**만 튜다 = “렌더마다 바뀌는 시크릿” 신호.
- **가장 깔끔한 해결:** 시크릿을 직접 만들어 두고 차트가 그걸 참조하게 설정.
- 임시 땀빵: ignoreDifferences로 해당 경로 제외(권장 X).

차트 리포/버전만 알려주시면, 지금 쓰시는 values.yaml에 맞춰 **정확한 키 이름**으로 패치 블록을 바로 드리겠습니다.



a안으로 할려고 하는데 2.9.2 airflow values.yaml 속성이 없는것 같아. (existkey같은것)

좋아요, **A안(고정 키 사용)** 으로 가려면 “키를 차트에 고정”만 해주면 됩니다.

쓰시는 차트가 공식 **Apache Airflow Helm chart** 라면, values 에 다음 두 가지 방법 중 하나가 있어요.

1) values에 키를 직접 고정 (가장 단순)

```
yaml

# values.yaml
webserverSecretKey: "<여기에_강력한_랜덤_키_고정>"
```

키 생성 예:

```
bash

python3 -c 'import secrets; print(secrets.token_hex(16))'
```

공식 문서가 바로 이 방법을 권장합니다. 이렇게 넣으면 **checksum/webserver-secret-key**가 고정되어 ArgoCD OutOfSync가 사라집니다. [Apache Airflow](#)

2) 기존 Kubernetes Secret을 참조

values에 **Secret 이름**만 지정하는 방식도 지원돼요:

```
yaml

# values.yaml
webserverSecretKeySecretName: my-webserver-secret
```

그리고 시크릿을 미리 만들어 둡니다:

```
bash

kubectl -n <ns> create secret generic my-webserver-secret \
  --from-literal="webserver-secret-key=$(python3 -c 'import secrets;
print(secrets.token_hex(16))')"
```

해당 시크릿의 key 이름은 **webserver-secret-key** 여야 합니다. 이 경로도 공식 가이드에 명시돼 있어요. [Apache Airflow](#)

“내 values에 저 필드가 안 보인다” 체크 포인트

- 공식 차트면 파라미터 레퍼런스에 **webserverSecretKey**, **webserverSecretKeySecretName**, 그리고 이를 읽도록 하는 **내장 시크릿 연동 항목** (**enableBuiltInSecretEnvVars.AIRFLOW__WEBSERVER__SECRET_KEY**)이 있습니다. 검색 키 그대로 문서에 있어요. [Apache Airflow](#)

- 만약 **Bitnami 차트**를 쓰는 거라면 키 이름이 달라요(예: webserver 쪽 “secret key”를 chart 값으로 고정하거나, 기존 Secret을 참조). Bitnami도 다중 웹서버 시 **고정 키**를 요구합니다. 쓰는 차트 리포/버전을 알려주시면 해당 차트 문법으로 바로 블록 만들어 드리겠습니다. [Artifact Hub](#)

적용 순서(권장)

1. (위 1번 또는 2번 방식 중 하나 선택) values 수정/Secret 생성
2. ArgoCD에서 해당 App **Sync**
3. **Deployment/airflow-webserver**의 annotation 확인

```
bash

kubectl -n <ns> get deploy airflow-webserver \
-o jsonpath='{.spec.template.metadata.annotations.checksum\.\webserver-secret-key}'; echo
```

값이 더 이상 매 렌더마다 바뀌지 않으면 정상입니다.

원하시면 지금 쓰시는 **차트 리포/버전**과 현재 values 일부만 보여 주세요. 그 문법에 딱 맞춰서 최소 변경 패치(yq/kustomize/helm values)로 정리해 드릴게요.



1안으로 하니깐 warn없이 깔끔하다. 굿

깔끔하게 잡혔다니 속이 다 시원하네요.

checksum/webserver-secret-key가 매번 바뀌던 건 결국 그 랜덤 시크릿 때문이었거든요.

지금처럼 고정 값으로 잡아두면, ArgoCD 입장에서도 “이건 안정된 desired 상태”로 인식해서 불필요한 diff가 전혀 안 납니다.

이제부터는 OutOfSync가 뜨면 진짜 의미 있는 변경일 확률이 높을 거예요.

운영 중엔 그게 제일 중요하죠.



쿠버네티스에서 crictl 명령어대해 알려줘

좋은 질문이에요.

crictl은 **Kubernetes**가 내부에서 쓰는 컨테이너 런타임(**Container Runtime Interface, CRI**)을 직접 다루기 위한 CLI 도구입니다.

쉽게 말해, **kubectl**이 쿠버네티스 API 서버와 이야기하는 도구라면,

crictl은 **kubelet**이 컨테이너 런타임(**containerd, CRI-O, etc.**)과 이야기할 때의 언어를 직접 쓸 수 있게 해주는 도구예요.



역할 정리

Kubernetes 구조를 간단히 보면:

```
java
```

```
kubectl → apiserver → kubelet → container runtime (containerd / CRI-O)
```

여기서 **crictl**은 kubelet이 runtime에게 보내는 요청(CRI gRPC API)을 수동으로 실행하는 도구입니다. 그래서 디버깅이나 로우레벨 컨테이너 확인용으로 자주 씁니다.

기본 구성

보통 `/etc/crictl.yaml` 에 runtime 소켓을 지정해둡니다.

예:

```
yaml
```

```
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
timeout: 10
debug: false
```

주요 명령어

명령어	설명
crictl ps	실행 중인 컨테이너 목록 보기 (docker ps 유사)
crictl ps -a	모든 컨테이너 (종료된 포함)
crictl pods	PodSandbox 목록 보기 (kubectl get pods 와 다름 — kubelet 단위)
crictl images	이미지 목록 보기
crictl pull <image>	이미지 수동으로 받기
crictl inspect <container_id>	컨테이너 상세 정보
crictl logs <container_id>	컨테이너 로그 확인
crictl exec -it <container_id> /bin/bash	컨테이너 내부 진입 (kubectl exec 유사)
crictl stats	컨테이너별 리소스 사용량 보기
crictl rm <container_id>	컨테이너 제거
crictl rmp <pod_id>	PodSandbox 제거
crictl version	CRI 버전 확인

활용 예시

1. 노드 단위에서 직접 컨테이너 확인


```
bash

cricctl ps -a | grep airflow
```

2. Kubelet이 실제 어떤 컨테이너를 올렸는지 확인

```
bash

cricctl pods
```

3. Pod가 Pending이나 CrashLoop일 때 내부 런타임 레벨 확인

```
bash

cricctl inspect <container_id> | jq .
```

4. 이미지 캐시 상태 점검

```
bash

cricctl images
```

혼한 실수/주의점

- **cricctl**은 **kubectl**과 전혀 다른 층위입니다.
kubectl이 안 되는 노드에서도, kubelet과 runtime만 살아 있으면 **cricctl**로 확인 가능.
- Pod 이름이 아니라 **Pod ID**나 **Container ID** 기준으로 동작합니다.
- Docker 기반(Kubernetes 1.24 이전)에서는 잘 안 쓰고, **docker ps**로 대체되곤 했지만 containerd로 넘어가면서 **cricctl**이 표준 디버깅 도구가 됐습니다.

예시: Airflow pod 디버깅 시

Pod가 “ContainerCreating” 상태에서 멈춰 있다면,

```
bash

cricctl ps -a
cricctl pods | grep airflow
cricctl inspectp <pod_id>
cricctl inspect <container_id>
cricctl logs <container_id>
```

이렇게 runtime 레벨에서 상태를 직접 확인할 수 있습니다.
(kubelet 로그와 짝지어 보면 특히 유용합니다.)

원하시면 **cricctl**을 사용해서 특정 노드에서 **Airflow** 파드가 실제 어떻게 실행됐는지 디버깅하는 절차도 예시로 보여드릴 수 있어요.
그쪽으로 이어가볼까요?