> 쿠버네티스 환경에서 airflow를 helm을 통해서 설치했거든. airflow를 모니터링 하기위해서 grafana를 설정 중에 있는데, promethous 데이터 소스를 통해서 리소스 모니터링 대시보드를 만들고 있는데, 오류는 없어 보이는데, 데이터가 나오지 않는데, 어떻게 디버깅 하는게 좋을까?

데이터가 "없어 보이는" 상태는 보통 세 군데 중 하나에서 막힙니다:
① Airflow가 메트릭을 제대로 **내보내지 않음** → ② Prometheus가 그 **엔드포인트를 스크랩하지 못함** → ③ Grafana에서 **쿼리/대시보드가 메트릭 이름과 안 맞음**.
아래 순서대로 점검하시면 원인 거의 잡힙니다.

---

1) Airflow가 메트릭을 실제로 내보내는지 확인

Airflow는 기본적으로 Prometheus를 직접 내보내지 않습니다. 보통 두 가지 경로 중 하나를 사용하죠.

- **StatsD** → **statsd-exporter** → **Prometheus** (Airflow 공식 문서도 StatsD/Otel을 안내) Apache Airflow
- **Airflow Prometheus Exporter**(예: robinhood/airflow-prometheus-exporter) 컨테이너로 노출 GitHub

헬름으로 설치했다면 values에서 다음 중 하나가 활성인지부터 확인하세요(차트마다 키 이름은 다를 수 있습니다. 공식 차트 파라미터 표도 같이 보세요). Apache Airflow

- `statsd.enabled: true` (Airflow 컨테이너에 StatsD 클라이언트 설정 포함)
- 또는 Exporter 사이드카/별도 배포가 있고, **/metrics** 같은 HTTP 엔드포인트가 실제 뜨는지

빠른 체크:

```
bash

# Airflow 웹서버/스케줄러 파드 안에서 statsd 설정 확인(있는 경우)
kubectl -n <ns> exec -it deploy/airflow-webserver -- printenv | grep -i STATSD

# Exporter가 있다면 파드 내부 포트 확인
kubectl -n <ns> get pods -l app=airflow -o wide
kubectl -n <ns> port-forward <exporter-pod> 9102:9102
curl -s http://127.0.0.1:9102/metrics | head
```

여기서 응답이 없다면 1단계에서 막힌 겁니다. (Airflow는 '그냥 두면' Prometheus가 보지 못합니다. StatsD 경로나 Exporter를 켜야 합니다.) Stack Overflow

---

2) Prometheus가 스크랩 중인지 확인(가장 흔한 막힘)

Prometheus가 해당 서비스를 "타깃"으로 인지해야 합니다.

2-1. Service/Endpoints가 제대로 있는지

```bash
kubectl -n <ns> get svc -l app=airflow -o wide
kubectl -n <ns> describe svc <airflow-metrics-svc>
kubectl -n <ns> get endpoints <airflow-metrics-svc> -o yaml
```

- 포트 이름이 **`metrics`** 또는 지정한 스킴과 맞는지(HTTP/TCP)
- 엔드포인트가 비어있지 않은지

2-2. Prometheus UI에서 Targets/ServiceDiscovery 확인

```bash
# prometheus-server 접근(예: kube-prometheus-stack 기준)
kubectl -n monitoring port-forward svc/prometheus-k8s 9090:9090
# 브라우저에서 http://localhost:9090/targets , /service-discovery
```

- Airflow 관련 타깃이 **UP**인지, 에러 메시지는 없는지
- 오퍼레이터(=kube-prometheus-stack)를 쓰면 **ServiceMonitor/ PodMonitor** 라벨 셀렉터가 Prometheus 설정의 **`release`** 라벨과 **정확히 일치**해야 스크랩됩니다. (여기 틀려서 'No data'가 엄청 자주 납니다) Medium

예시(ServiceMonitor – 셀렉터/네임스페이스는 환경에 맞게 조정):

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: airflow-metrics
  namespace: monitoring
  labels:
    release: prometheus   # ← Prometheus Operator가 보는 release 라벨과 일치해야 함
spec:
  namespaceSelector:
    matchNames: ["<airflow-namespace>"]
  selector:
    matchLabels:
      app: airflow
      component: metrics
  endpoints:
    - port: metrics       # Service의 port 이름과 동일
      path: /metrics
      interval: 30s
```

2-3. 기본 K8s 메트릭도 안 뜨면, 인프라 쪽 점검

- **kube-state-metrics**가 죽어 있으면 네임스페이스/파드 수준 지표 자체가 비니까, 대시보드 전반이 'No data'가 됩니다. 상태부터 보세요. Stack Overflow
- node-exporter/스크레이프 잡 버전 차이로 **메트릭 이름이 바뀌어**도 빈 화면이 나옵니다(예: **`_bytes/_total`** 접미사 차이). Stack Overflow

---

3) Grafana에서 쿼리 자체를 검증

대시보드만 보면 헷갈립니다. **Explore**에서 생짜 PromQL로 확인하세요.

- `up{job=~".*airflow.*"}` : 해당 잡이 스크랩되고 있으면 1이 떠야 정상
- `sum(rate(container_cpu_usage_seconds_total{namespace="<airflow-ns>"}[5m]))` : 클러스터 메트릭이 살아있는지 확인(프로메테우스/eks/kube-st랙에 따라 메트릭 이름 조금 다를 수 있음)
- 패널 우측 상단 메뉴 → **Debug metrics for this panel**로 패널이 참조하는 메트릭/에러를 바로 확인 가능합니다. Grafana Labs

---

흔한 원인 → 빠른 처방

- **Airflow가 Prometheus로 직접 노출된 줄 안 경우** → StatsD 설정을 켜고 statsd-exporter를 붙이거나, **Airflow Prometheus Exporter**를 사이드카/별도 서비스로 배치하세요. Apache AirflowGitHub
- **ServiceMonitor 라벨 불일치**(가장 잦음) → `metadata.labels.release`가 Prometheus Operator의 `--selector`와 정확히 맞는지 재배포. Medium
- **kube-state-metrics 미동작** → 파드 재시작/권한 확인 후 재수집. Stack Overflow
- **대시보드-메트릭 이름 불일치** → 패널 PromQL을 최신 메트릭 이름으로 교체. (예: `node_memory_Buffers` → `node_memory_Buffers_bytes`, `node_disk_bytes_written` → `node_disk_written_bytes_total`) Stack Overflow

---

체크리스트(복사-붙여넣기용)

1. Airflow 값(yaml)에서 **StatsD 또는 Exporter**가 켜졌는지 확인. 응답: `curl http://<pod>:<port>/metrics`에 줄줄 나온다. Apache AirflowGitHub
2. Airflow 메트릭 **Service/Endpoints**가 유효한가(포트 이름 `metrics`, Endpoints 비지 않음).
3. Prometheus **/targets**에서 Airflow 타깃이 **UP**인가(오류 메시지 확인).
4. kube-prometheus-stack라면 **ServiceMonitor 라벨 셀렉터 일치** 확인. Medium
5. Grafana **Explore**에서 `up{job=~".*airflow.*"}`로 생존 신호 확인. Grafana Labs
6. **kube-state-metrics**/node-exporter 동작 확인(클러스터 공통 메트릭이 떠야 함). Stack Overflow
7. 대시보드 패널에서 **Debug metrics**로 메트릭 이름/라벨 미스매치 수정. Grafana Labs

---

원하시면, 지금 쓰는 헬름 values 일부(메트릭 관련 블록과 라벨, 사용 중인 Prometheus 스택 이름)만 붙여주세요. 그 상태에서 **Service/ServiceMonitor 샘플**을 해당 환경에 맞춰 바로 깔끔하게 맞춰드릴게요.

> 운영환경이라서 붙혀넣기가 힘드네. 디버깅 하는 방법을 잘 알려주면 내가 해야할 것 같아.

좋습니다. 붙여넣기 없이도 한 바퀴 점검 가능한 “현장 디버깅 루틴”을 드릴게요.
핵심은 **(A) Airflow가 메트릭을 내보내는가 → (B) Prometheus가 스크랩하는가 → (C) Grafana가 제대로 조회하는가** 순서로 끊어서 보는 겁니다.

0) 준비 — 내가 뭘 쓰고 있는지부터 확정

- 네임스페이스: **`AIRFLOW_NS`**, **`MON_NS`** (monitoring) 같은 식으로 마음속에 고정하세요.
- Prometheus가 **kube-prometheus-stack(=Operator)** 인지, 아니면 **단독 Prometheus** 인지 먼저 구분.
  - Operator면 *ServiceMonitor/PodMonitor* 자원으로 스크랩합니다.
  - 단독이면 Service에 **scrape 어노테이션**을 붙여 스크랩합니다.

A. Airflow가 메트릭을 "정말" 내보내는지

Airflow는 기본으로 Prometheus에 직접 안 내보냅니다. 보통 둘 중 하나죠:

- **statsd → statsd-exporter → Prometheus**
- **airflow-prometheus-exporter** 같은 HTTP **`/metrics`** 엔드포인트

A-1. 엔드포인트 현장 확인

```bash
bash

# 후보 파드/서비스 보기
kubectl -n AIRFLOW_NS get pods -o wide
kubectl -n AIRFLOW_NS get svc  -o wide

# 메트릭 포트가 있는 파드 하나 집어서 로컬로 터널
kubectl -n AIRFLOW_NS port-forward POD_NAME 9102:9102   # 포트 번호는 환경에 맞게
# 다른 터미널에서
curl -s http://127.0.0.1:9102/metrics | head -n 20
```

- 텍스트가 줄줄 나오면 **A 통과**.
- 연결 실패/빈 응답이면 **Airflow 측 설정 문제**입니다. (statsd/exporter 프로세스가 안 떠 있거나 포트 미노출)

  팁: 포트가 헷갈리면 **`kubectl -n AIRFLOW_NS describe pod POD_NAME | grep -i -A2 'Ports'`** 로 컨테이너 포트를 먼저 봅니다.

B. Prometheus가 그 엔드포인트를 스크랩하는지

B-1. Service/Endpoints 유효성

```bash
bash

# 메트릭 서비스 후보 찾기(라벨로 좁히면 더 좋습니다)
kubectl -n AIRFLOW_NS get svc -o wide

# 서비스가 파드를 제대로 가리키는지(엔드포인트가 비었는지) 확인
kubectl -n AIRFLOW_NS get endpoints SVC_NAME -o yaml
```

- **Endpoints가 비었으면** 셀렉터 라벨이 안 맞는 겁니다. Service **`spec.selector`** ↔ Pod **`metadata.labels`** 한 쌍을 맞추세요.

- Prometheus가 **포트 이름**을 찾게 만드는 경우가 많습니다. Service의 포트 이름을 **`metrics`**처럼 명시하세요.

B-2. 스크랩 구성 방식별 체크

(1) kube-prometheus-stack(Operator)일 때

```
bash

# ServiceMonitor가 있는지
kubectl -n MON_NS get servicemonitor | grep -i airflow

# 내용 확인
kubectl -n MON_NS get servicemonitor airflow-metrics -o yaml
```

- **`spec.selector.matchLabels`**가 **메트릭 Service의 라벨**과 맞는지
- **`spec.endpoints[].port`**가 **Service의 포트 이름**과 같은지
- **가장 흔한 실수**: ServiceMonitor에 붙인 라벨(예: **`release: prometheus`**)이 Prometheus(Operator)가 감시하는 selector와 **불일치**
  → Prometheus 리소스의 **`spec.serviceMonitorSelector`**(또는 Helm release 라벨)와 값이 정확히 같아야 합니다.

(2) 단독 Prometheus일 때

Service에 어노테이션이 필요할 수 있습니다:

```
yaml

metadata:
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/path:   "/metrics"
    prometheus.io/port:   "9102"
```

붙인 뒤 **`endpoints`**가 살아있는지 다시 확인하세요.

B-3. Prometheus에서 직접 확인(가장 확실)

```
bash

# Prometheus UI 열기(서비스 이름은 환경에 맞게)
kubectl -n MON_NS port-forward svc/prometheus-k8s 9090:9090
# 브라우저: http://127.0.0.1:9090/targets  (Airflow 관련 타깃이 UP인지)
# 브라우저: http://127.0.0.1:9090/service-discovery (발견은 되는지)
```

- 타깃이 **DOWN**이면 에러 메시지를 그대로 읽고(401/403, 타임아웃, DNS 실패 등) 그 지점 수습.
- **라벨 매칭 오류**면 아예 타깃 목록에 안 뜹니다 → ServiceMonitor/어노테이션 재점검.

C. Grafana 쿼리가 맞는지

대시보드 말고 **Explore** 탭에서 생짜로 확인하면 헷갈림이 줄어요.

C-1. 생존 신호

```
arduino

up{job=~".*airflow.*"}
```

- 1이 나오면 Prometheus는 스크랩 중입니다(최소한 네트워크는 됨).

C-2. Airflow/컨테이너 리소스 예시

(메트릭 이름은 배포 스택에 따라 조금 다릅니다)

```
csharp

# 네임스페이스의 CPU 사용률(컨테이너 합)
sum by (namespace) (rate(container_cpu_usage_seconds_total{namespace="AIRFLOW_NS"}[5m]))
```

값이 0만 나오면 **스크랩은 되는데 해당 라벨 조합이 없음** → 라벨(네임스페이스/워크로드명) 필터를 느슨하게 풀어 보세요.

C-3. 패널 자체 디버깅

- 패널 우측 상단 ▸ **Query inspector / Debug** 열어서 **실제 PromQL**과 **에러** 확인.
- **데이터 소스**가 맞는지(간혹 다른 Prometheus를 가리킵니다).
- **시간 범위**를 넉넉히(예: Last 6 hours). 운영은 Asia/Seoul이니 로컬 타임존/UTC 헷갈림 주의.
- 자동 새로고침을 잠깐 꺼두고 수동 새로고침으로 상태 고정.

네트워크/보안에 부딪혔을 때 체크

- **NetworkPolicy**가 Prometheus → Airflow 파드의 메트릭 포트를 막고 있지 않은지
  → 임시로 동일 네임스페이스에 busybox 띄워 **`wget -qO- http://POD_IP:PORT/metrics`** 해보면 빨리 분리됩니다.
- **TLS/인증**이 필요한 엔드포인트인지(베이직 auth/Ingress 경유 등). Prometheus **`scheme: https`**, **`bearerToken`**, **`tlsConfig`**가 필요한지 확인.
- **서비스 DNS** 대신 **엔드포인트 IP**로 쏴서 DNS 이슈를 빠르게 제외.

15분 짜리 "막히면 그대로 따라하기" 스크립트

1. **엔드포인트 살아있나**

```
bash

kubectl -n AIRFLOW_NS get pods -o wide
kubectl -n AIRFLOW_NS describe pod POD | sed -n '1,120p' | grep -i -A2 'Port'
kubectl -n AIRFLOW_NS port-forward POD 9102:9102
curl -s localhost:9102/metrics | head
```

2. **서비스/엔드포인트 연결됐나**

```
bash

kubectl -n AIRFLOW_NS get svc -o wide
kubectl -n AIRFLOW_NS get endpoints SVC -o yaml | sed -n '1,120p'
```

3. **(Operator 기준) ServiceMonitor 라벨/포트 일치하나**

```
bash

kubectl -n MON_NS get servicemonitor | grep -i airflow || echo "없음"
kubectl -n MON_NS get servicemonitor airflow-metrics -o yaml | sed -n '1,200p'
```

4. **Prometheus에서 타깃 상태 확인**

```
bash

kubectl -n MON_NS port-forward svc/prometheus-k8s 9090:9090
# 브라우저 /targets, /service-discovery 확인
```

5. **Grafana Explore에서**

```
arduino

up{job=~".*airflow.*"}
sum(rate(container_cpu_usage_seconds_total{namespace="AIRFLOW_NS"}[5m]))
```

6. **여전히 빈값이면**

- 라벨을 점점 풀면서(네임스페이스 조건 제거 → 잡 이름만) 데이터가 나타나는 지점 찾기
- 대시보드 내 패널 PromQL을 그 결과에 맞춰 조정

자주 터지는 포인트 — 체크리스트로 마무리

- ☐ Airflow 메트릭 엔드포인트가 **실제로 응답**한다.
- ☐ Service **`selector`** ↔ Pod **`labels`**가 **정확히 매치**, Endpoints **비어있지 않다**.
- ☐ (Operator) ServiceMonitor의 **라벨/포트/패스**가 Service와 **1:1로 일치**한다.
- ☐ Prometheus **`/targets`**에서 상태가 **UP**이다(아니면 에러 메시지 보고 즉시 수습).
- ☐ Grafana **데이터 소스**/시간 범위/쿼리 라벨이 환경과 맞다.

필요하시면, 지금 환경이 **Operator형인지 단독형인지**, 그리고 "메트릭이 어디서 나오는지(statsd-exporter인지 exporter 컨테이너인지)"만 알려주세요. 그 정보만으로도 **딱 맞는 Service/ServiceMonitor 샘플**을 만들어 드릴 수 있습니다.
걱정 마세요. 이 루틴 타고 돌면 어디서 막혔는지 금방 드러납니다.