



airflow 모니터링에 prometheus + grafana 조합으로 많이 사용해?

네, 많이 씁니다.

Airflow는 워커/스케줄러/웹서버가 각각 독립 프로세스로 돌아가니까 리소스와 태스크 상태를 잘 보려면 메트릭 수집이 필요합니다.

보통 조합은 이렇게 가죠:

- **Airflow → Prometheus**

Airflow 2.x부터는 **statsd** 플러그인이나 직접 Prometheus exporter(**airflow-exporter**, **prometheus-statsd-exporter**)를 붙여서 메트릭을 노출합니다. 스케줄러 큐 길이, 태스크 성공/실패 카운트, DAG 런 타임, 워커 리소스 사용량 같은 걸 뽑아낼 수 있어요.

- **Prometheus → Grafana**

Prometheus가 긁어온 지표를 Grafana가 시각화합니다. DAG 런 성공률, 태스크 대기 시간, 워커별 CPU/메모리, 큐 지연 시간 같은 걸 대시보드로 만들어 쓰죠. 커뮤니티에서 만들어둔 Airflow 전용 대시보드 JSON도 많습니다.

실무에서 **Kubernetes + Helm** 환경이면 Prometheus Operator + Grafana를 묶어서 Airflow Helm 차트에 같이 붙이는 케이스가 흔해요. “Airflow 자체 로그 모니터링 → ELK/EFK”, “지표/알람 → Prometheus + Grafana” 이렇게 역할을 분리하는 식으로요.

👉 즉, Airflow 모니터링에 Prometheus + Grafana 조합은 거의 사실상 표준에 가깝습니다.

궁금한 건, 지금 원하시는 게 **리소스(CPU/메모리) 모니터링** 쪽인지, 아니면 **DAG/태스크 성공률 같은 운영 메트릭** 쪽인가요?



grafana 대시보드에 promethous 를 이용해서 만드는 방법 몇가지 템플릿 알려줘

좋죠. 운영에서 바로 써먹기 좋은 **Prometheus + Grafana 대시보드 템플릿**을 몇 가지 묶어 드릴게요.

당신 환경(쿠버네티스/RHEL9/Helm/ArgoCD, Airflow 중심)에 맞춰 **변수화**와 **PromQL 쿼리** 위주로 잡았습니다. 그대로 붙여서 쓰되, 지표 이름은 사용 중인 익스포터(**airflow-exporter** vs **statsd-exporter**)마다 조금씩 다르니 **패턴만 바꿔** 적용하시면 됩니다.

## 0) 공통 변수(Templating) 설정

Grafana → Dashboard settings → **Variables**에 아래 3개를 추가하세요.

### 1. \$datasource (Prometheus 선택 변수)

- Type: *Datasource*
- Type filter: *Prometheus*

### 2. \$namespace (쿠버네티스 네임스페이스)

- Type: *Query*
- Data source: **\$datasource**
- Query:

scss

```
label_values(kube_pod_info, namespace)
```

- Multi-value: ✓

### 3. \$dag (Airflow DAG ID; 익스포터에 따라 둘 중 하나 선택)

- Type: *Query*
- Data source: **\$datasource**
- Query (A: airflow-exporter 스타일):

scss

```
label_values(airflow_task_status, dag_id)
```

- (대안 B: statsd-exporter 경로 → 태그가 애매하면 대체)

scss

```
label_values({__name__=~"airflow_.*"}, dag_id)
```

- Include All: ✓ (없으면 전체 집계용)

팁: 먼저 **Explore**에서 `{__name__=~"airflow_.*"}`로 메트릭 이름을 훑어보고, 실제 환경 지표명을 골라 붙이세요.

## 1) Airflow 개요(성공률/지연/실패탭)

### 패널 1: 지난 24시간 태스크 성공률(%)

(A: airflow-exporter 방식)

bash

```
100 * sum(rate(airflow_task_status{status="success",dag_id=~"$dag"}[24h]))
/ sum(rate(airflow_task_status{dag_id=~"$dag"}[24h]))
```

(B: statsd-exporter에서 status 라벨이 다르면 성공/실패 계열명을 매칭해서 합산)

- 추천 시각화: *Stat* (Threshold: 95, 90)

### 패널 2: DAG 런 대기시간 p95 (큐 적체 감지)

(A: 히스토그램 버킷이 있을 때)

```
bash

histogram_quantile(
  0.95,
  sum by (le) (
    rate(airflow_dag_run_queue_duration_seconds_bucket{dag_id=~"$dag"}[$__rate_interval])
  )
)
```

(히스토그램이 없고 게이지라면 `quantile_over_time` 대체)

- 추천 시각화: *Time series*

## 패널 3: 실패 많은 DAG Top N

```
less

topk(10,
  sum(rate(airflow_task_status{status="failed"}[2h])) by (dag_id)
)
```

- 추천 시각화: *Table* 또는 *Bar gauge*

## 2) 스케줄러/익스큐터 헬스

### 패널 4: Exporter 타깃 헬스(UP)

```
scss

avg(up{job=~"airflow.*"}) by (job)
```

- job 라벨은 스크레이프 잡 이름에 맞게 수정(예: `airflow-scheduler`, `airflow-webserver`)

### 패널 5: 스케줄러 하트비트/루프시간

(환경에 따라 지표명이 다릅니다. 아래 중 실측되는 항목 사용)

```
scss

max_over_time(airflow_scheduler_heartbeat_seconds[$__interval])
```

또는

```
bash

rate(airflow_scheduler_loop_duration_seconds_sum[$__rate_interval])
/
rate(airflow_scheduler_loop_duration_seconds_count[$__rate_interval])
```

- 추천 시각화: *Time series* (상승 → 지연 증가)

### 3) DAG/태스크 지속시간 & SLA

#### 패널 6: 태스크 실행시간 p95 (초)

(A: 히스토그램)

```
bash

histogram_quantile(
  0.95,
  sum by (le, task_id) (
    rate(airflow_task_duration_seconds_bucket{dag_id=~"$dag"}[$__rate_interval])
  )
)
```

(B: Summary 계열)

```
bash

rate(airflow_task_duration_seconds_sum{dag_id=~"$dag"}[$__rate_interval])
/
rate(airflow_task_duration_seconds_count{dag_id=~"$dag"}[$__rate_interval])
```

#### 패널 7: SLA Miss 수(최근 24h)

(있다면)

```
bash

sum(increase(airflow_task_sla_miss_total{dag_id=~"$dag"}[24h]))
```

### 4) 워커/파드 리소스(쿠버네티스 관점)

kube-state-metrics / cAdvisor 메트릭 기준. 네임스페이스/파드 네이밍을 Airflow에 맞춰 필터하세요.

#### 패널 8: 워커 CPU 사용률 (코어)

```
bash

sum by (pod) (
  rate(container_cpu_usage_seconds_total{namespace=~"$namespace", pod=~".*airflow.*worker.*",
    image!=""}[$__rate_interval])
)
```

#### 패널 9: 워커 메모리 Working Set (GiB)

bash

```
sum by (pod) (
  container_memory_working_set_bytes{namespace=~"$namespace", pod=~".*airflow.*worker.*",
  image!=""}
) / 1024^3
```

## 패널 10: 파드 재시작 증가(최근 1h)

arduino

```
topk(10,
  increase(kube_pod_container_status_restarts_total{namespace=~"$namespace"}[1h])
)
```

## 5) 큐 적체(Celery/Rabbit/Redis 사용 시)

익스포터가 노출하는 라벨에 맞춰 조정하세요.

## 패널 11: Celery 큐 대기 메시지 수

(예시: RabbitMQ exporter)

arduino

```
sum(rabbitmq_queue_messages_ready{queue=~"celery.*"})
```

## 패널 12: 워커 활성/유휴 수

(예시: Celery exporter)

scss

```
sum(celery_workers_active) by (hostname)
sum(celery_workers_reserved) by (hostname)
```

## 6) 가져다 붙이는 패널 JSON 샘플 3종

Grafana → 패널 추가 → **Edit(JSON)** 에 붙여 넣기.

"\$datasource" 바인딩을 위해 `datasource.uid`를 `${DS_PROMETHEUS}`로 뒀습니다. (대시보드 상단 우측  
**"Dashboard settings → Variables → Datasource"**에서 만든 변수를 지정하세요.)

### (A) 태스크 성공률 Stat

json

```
{
  "type": "stat",
```

```

"title": "Airflow Task Success Rate (24h)",
"datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
"targets": [
  {
    "expr": "100 * sum(rate(airflow_task_status{status=\"success\", dag_id=~\"$dag\"}[24h])) / sum(rate(airflow_task_status{dag_id=~\"$dag\"}[24h]))",
    "refId": "A"
  }
],
"fieldConfig": {
  "defaults": {
    "unit": "percent",
    "thresholds": {
      "mode": "absolute",
      "steps": [
        {"color": "red", "value": null},
        {"color": "yellow", "value": 90},
        {"color": "green", "value": 95}
      ]
    }
  },
  "overrides": []
},
"options": {"reduceOptions": {"calcs": ["lastNotNull"]}}
}

```

## (B) DAG 큐 지연 p95 시계열

```

json

{
  "type": "timeseries",
  "title": "DAG Queue Latency p95",
  "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
  "targets": [
    {
      "expr": "histogram_quantile(0.95, sum by (le) (rate(airflow_dag_run_queue_duration_seconds_bucket{dag_id=~\"$dag\"}[ $__rate_interval ])))",
      "legendFormat": "p95",
      "refId": "A"
    }
  ],
  "fieldConfig": {"defaults": {"unit": "s"}}
}

```

## (C) 워커 메모리 Top 파드

```

json

{
  "type": "bargauge",
  "title": "Airflow Workers - Memory Working Set (GiB)",
  "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
  "targets": [
    {
      "expr": "topk(10, sum by (pod) (container_memory_working_set_bytes{namespace=~\"$namespace\", pod=~\".*airflow.*worker.*\", image!=\"\"} / 1024^3))",
      "refId": "A"
    }
  ],
  "fieldConfig": {"defaults": {"unit": "decgbytes"}}
}

```

## 7) 미니 대시보드 JSON (패널 4개짜리 한 장)

새 대시보드 → JSON Import → 아래 전체를 붙여 넣기.

붙인 뒤 **Variables**에서 **\$datasource/\$namespace/\$dag**만 점검하세요.

```
json

{
  "title": "Airflow - Quick Ops",
  "panels": [
    {
      "gridPos": {"h": 6, "w": 6, "x": 0, "y": 0},
      "type": "stat",
      "title": "Task Success Rate (24h)",
      "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
      "targets": [{"expr": "100 *
sum(rate(airflow_task_status{status=~\"success\" ,dag_id=~\"$dag\"}[24h])) /
sum(rate(airflow_task_status{dag_id=~\"$dag\"}[24h]))", "refId": "A"}],
      "fieldConfig": {"defaults": {"unit": "percent"}}
    },
    {
      "gridPos": {"h": 6, "w": 6, "x": 6, "y": 0},
      "type": "timeseries",
      "title": "DAG Queue Latency p95",
      "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
      "targets": [{"expr": "histogram_quantile(0.95, sum by (le)
(rate(airflow_dag_run_queue_duration_seconds_bucket{dag_id=~\"$dag\"}
[ $__rate_interval ])))", "refId": "A"}],
      "fieldConfig": {"defaults": {"unit": "s"}}
    },
    {
      "gridPos": {"h": 8, "w": 12, "x": 0, "y": 6},
      "type": "timeseries",
      "title": "Workers CPU (cores) by Pod",
      "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
      "targets": [{"expr": "sum by (pod)
(rate(container_cpu_usage_seconds_total{namespace=~\"$namespace\", pod=~\".*airflow.*worker.*\",
image!=\"\"}[ $__rate_interval ]))", "refId": "A"}]
    },
    {
      "gridPos": {"h": 8, "w": 12, "x": 0, "y": 14},
      "type": "bargauge",
      "title": "Pod Restarts ↑ (1h)",
      "datasource": {"type": "prometheus", "uid": "${DS_PROMETHEUS}"},
      "targets": [{"expr": "topk(10,
increase(kube_pod_container_status_restarts_total{namespace=~\"$namespace\"}
[1h]))", "refId": "A"}]
    }
  ],
  "templating": {
    "list": [
      {"name": "datasource", "type": "datasource", "query": "prometheus"},
      {"name": "namespace", "type": "query", "datasource":
{"type": "prometheus", "uid": "${DS_PROMETHEUS}", "query": "label_values(kube_pod_info,
namespace)", "multi": true},
      {"name": "dag", "type": "query", "datasource":
{"type": "prometheus", "uid": "${DS_PROMETHEUS}", "query": "label_values(airflow_task_status,
dag_id)", "includeAll": true}
    ]
  },
  "schemaVersion": 38
}
```

## 8) 알람(옵션, PrometheusRule 예시)

## (예) 성공률 하락

```
yaml

groups:
- name: airflow.reliability
  rules:
  - alert: AirflowTaskSuccessRateLow
    expr: (sum(rate(airflow_task_status{status="success"}[30m])) /
sum(rate(airflow_task_status[30m]))) < 0.95
    for: 10m
    labels: {severity: warning}
    annotations:
      summary: "Airflow task success rate < 95% (30m)"
```

## (예) 큐 지연 p95 급증

```
yaml

- name: airflow.latency
  rules:
  - alert: AirflowDAGQueueLatencyHigh
    expr: histogram_quantile(0.95, sum by (le)
(rate(airflow_dag_run_queue_duration_seconds_bucket[10m]))) > 60
    for: 10m
    labels: {severity: warning}
    annotations:
      summary: "DAG queue latency p95 > 60s (10m)"
```

## 마지막 체크포인트

- 익스포터 마다 **메트릭 이름/라벨**이 조금 달라요. 위 쿼리에서 **airflow\_\*** 지표명이 안 잡히면 **Explore**에서 **airflow**로 필터해 실제 이름을 먼저 확인하세요.
- 쿠버네티스 리소스 쪽은 **kube\_pod\_\***, **container\_\*** 계열이 표준이라 거의 그대로 맞습니다.
- 처음에는 “개요 1장(성공률/큐지연/CPU/재시작)”으로 시작 → 문제 구간이 보이면 **세부 대시보드**(DAG/태스크/스케줄러)로 파고드는 흐름이 제일 덜 스트레스 받습니다.

필요하시면, 지금 쓰는 익스포터(airflow-exporter인지, statsd→prometheus-statsd-exporter인지) 이름만 알려주세요. 거기에 딱 맞춰 **메트릭명 1:1 매칭 쿼리**로 손봐서 드릴게요.