The CI/CD Runner Survival Guide

Stop Your GitOps From Falling Apart at 2am

TacitSoft LLC
Fractional DevOps for Startups

v1.0.8-a4f2cec

Scan to Learn More
tacitsoft.dev

Feedback? github.com/tacitness/guides/issues
joel@tacitsoft.dev

Contents

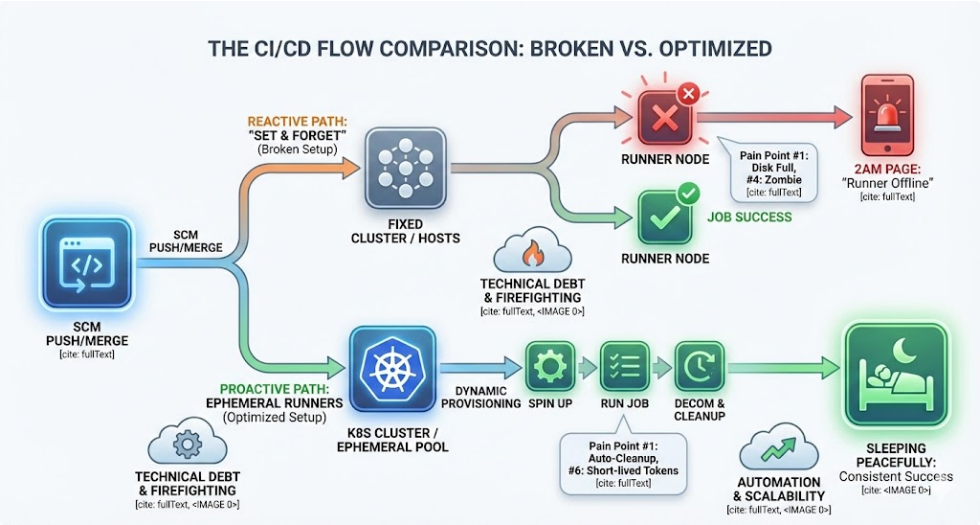
The CI/CD Runner Survival Guide

Stop Your GitOps From Falling Apart at 2am

By Joel Hanger, TacitSoft… Fractional DevOps for SaaS Teams

Your CI/CD runners are probably a mess. Not because you're bad at your job… because runners are "set and forget" infrastructure that nobody owns until they break.

I audit DevOps setups for a living. Here are the six pain points I find in almost every organization, and exactly how to fix them.



Pain Point #1: Disk Space Death Spiral

Warning Signs

- CI/CD jobs fail with “no space left on device” at 2am
- Docker builds take progressively longer over weeks
- Runners become unresponsive during large builds
- `df -h` shows `/var/lib/docker` at 95%+

Why This Happens

Docker’s default behavior keeps all images and layers until you explicitly remove them. Self-hosted runners don’t auto-clean. Build artifacts pile up. Nobody set up cleanup… because when it was set up, disk was plentiful.

The Fix

For GitHub Actions self-hosted runners:

```
# Add to crontab (runs daily at 3am)
0 3 * * * docker system prune -af --volumes
```

For GitLab runners:

```
# In config.toml
[[runners]]
  [runners.docker]
    # Auto-remove containers after job
    disable_cache = false
    # Keep only 3 most recent images
    pull_policy = "if-not-present"
```

Better solution: Ephemeral runners

GitHub Actions ephemeral runners create a fresh environment per job. More compute cost, zero state drift.

```
# In your workflow
runs-on: [self-hosted, ephemeral]
```

References:

- GitHub: Using ephemeral runners for autoscaling
- GitLab: Runner executors

Pain Point #2: “Runner Offline” Mystery

Warning Signs

- Jobs sit in queue for hours with no progress
- Team finds out about stuck deploys from clients
- `gh api .../runners` shows runners as “offline” you didn’t know about
- Random jobs fail with “no runners available”

Why This Happens

Runners can go offline for dozens of reasons… network issues, disk full, OOM kills, systemd failures. Without monitoring, you’re blind. Most teams have zero alerting on runner health.

The Fix

For GitHub Actions:

```
# Check runner status via API
curl -H "Authorization: token YOUR_PAT" \
  https://api.github.com/repos/OWNER/REPO/actions/runners \
  | jq '.runners[] | {name: .name, status: .status}'
```

For GitLab runners, enable Prometheus metrics:

```
# In config.toml
listen_address = ":9252"
```

Then scrape with Prometheus:

```
# prometheus.yml
scrape_configs:
  - job_name: 'gitlab-runner'
    static_configs:
      - targets: ['runner1:9252', 'runner2:9252']
```

Key metrics to alert on:

- `gitlab_runner_jobs_total` - Job throughput
- `gitlab_runner_errors_total` - Error rate
- `gitlab_runner_concurrent` - Capacity utilization

Alerting rule example:

```yaml
# Prometheus alerting rule
- alert: RunnerOffline
  expr: up{job="gitlab-runner"} == 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "GitLab Runner {{ $labels.instance }} is offline"
```

References:

- GitLab: Monitor GitLab Runner usage
- GitHub: Monitoring self-hosted runners

Pain Point #3: Secrets Accidentally Logged

Warning Signs

- `echo $API_KEY` or `printenv` in CI workflow files
- Secrets visible in plain text in job logs
- No pre-commit hooks checking for credentials
- Workflow files skip code review ("it's just CI config")

Why This Happens

CI debugging is painful. People take shortcuts. Code review doesn't catch workflow files as thoroughly as application code. And once a secret is in logs, it's in logs forever… visible to anyone with repo access.

The Fix

Prevention: Pre-commit secret scanning

Install gitleaks as a pre-commit hook:

```yaml
# .pre-commit-config.yaml
repos:
  - repo: https://github.com/gitleaks/gitleaks
    rev: v8.24.2
    hooks:
      - id: gitleaks
```

Detection: Scan your repos regularly

```
# Install gitleaks
brew install gitleaks

# Scan current repo
gitleaks git -v --report-path gitleaks-report.json
```

Alternative: TruffleHog (more detectors, validates if secrets are live)

```
# Install
brew install trufflehog

# Scan with verification
trufflehog git https://github.com/your-org/your-repo --results=verified
```

TruffleHog supports 800+ secret types and can verify if leaked credentials are still active.

CI Integration (GitHub Actions):

```yaml
- name: Secret Scanning
  uses: trufflesecurity/trufflehog@main
  with:
    extra_args: --results=verified,unknown
```

References:

- Gitleaks: github.com/gitleaks/gitleaks (24.8k stars)
- TruffleHog: github.com/trufflesecurity/trufflehog (24.4k stars)

Pain Point #4: Zombie Runners

Warning Signs

- 15 runners registered, 4 actually work
- Jobs randomly fail depending on which runner picks them up
- `gh api .../runners` shows runners you don't recognize
- "Who manages the runners?" gets shrugs

Why This Happens

Runners get registered on VMs that later get terminated. IP addresses change. Someone "temporarily" disables a runner and forgets. Without an inventory process, you're running on luck.

The Fix

Audit your runners regularly:

```
# GitHub - List all runners
gh api repos/{owner}/{repo}/actions/runners \
  --jq '.runners[] | "\(.name): \(.status)"'

# GitLab - Check runner status
curl --header "PRIVATE-TOKEN: $GITLAB_TOKEN" \
  "https://gitlab.com/api/v4/runners?status=offline"
```

Automate cleanup of stale runners:

```
#!/bin/bash
# Remove offline GitHub runners older than 7 days
OFFLINE_RUNNERS=$(
  gh api repos/{owner}/{repo}/actions/runners \
    --jq '.runners[] | select(.status=="offline") | .id'
)

for id in $OFFLINE_RUNNERS; do
  gh api -X DELETE \
    repos/{owner}/{repo}/actions/runners/$id
done
```

Runner inventory dashboard:

Use `node_exporter` on each runner host, then build a Grafana dashboard showing:

- Runner name

- Online/offline status
- Last job timestamp
- Disk/memory/CPU utilization

References:

- GitHub API: Self-hosted runners REST API
- GitLab API: Runners API

Pain Point #5: Label Soup (Jobs Hit Wrong Runners)

Warning Signs

- Jobs run on the wrong runner type (GPU job hits CPU-only machine)
- "No runners available" when runners are clearly online
- Inconsistent build times depending on which runner picks up the job
- Labels like `linux`, `docker`, `prod`, `test` with no documentation

Why This Happens

Runner labels (GitHub) and tags (GitLab) control job routing. Without a labeling strategy, chaos ensues:

- **Default labels aren't enough**: `self-hosted`, `linux`, `x64` don't distinguish your GPU build server from your test VM
- **Labels accumulate without cleanup**: Someone adds `docker` to one runner, forgets the others
- **No label documentation**: What does `prod-runner` actually mean? Who knows?
- **Wildcard matching**: GitLab's `Run untagged jobs` checkbox causes jobs to land anywhere

The Fix

Define a labeling taxonomy:

```
# Example: Consistent label schema
# Format: [self-hosted, OS, arch, purpose, capabilities...]

# Production deployment runners
runs-on: [self-hosted, linux, x64, deploy, prod]

# GPU-enabled build runners
runs-on: [self-hosted, linux, x64, build, gpu]

# Test runners (ephemeral, disposable)
runs-on: [self-hosted, linux, x64, test, ephemeral]
```

For GitLab, use tags strategically:

```
# .gitlab-ci.yml
build:
  tags:
    - docker
```


```
    - linux
    - gpu  # Only runners with ALL tags can run this job
  script:
    - make build
```

Audit your labels:

```
# GitHub - List all runners with their labels
gh api repos/{owner}/{repo}/actions/runners \
  --jq '.runners[] |
    {name: .name, labels: [.labels[].name]}'

# GitLab - List runners with tags
curl --header "PRIVATE-TOKEN: $GITLAB_TOKEN" \
  "https://gitlab.com/api/v4/runners?tag_list=docker,linux"
```

Disable “Run untagged jobs” on critical runners:

For GitLab, edit each runner and uncheck “Run untagged jobs”. This prevents random jobs from landing on your production deployment runner.

Document your labels in your README:

```
## Runner Labels

| Label | Meaning |
|-------|---------|
| `deploy` | Can deploy to production |
| `gpu` | Has NVIDIA GPU for ML builds |
| `ephemeral` | Destroyed after each job |
| `docker` | Has Docker installed |
```

References:

- GitHub: Using labels to route jobs
- GitLab: Control jobs with tags

Pain Point #6: Token Expiration Surprise

Warning Signs

- Deploys suddenly stop with auth errors
- Runner shows “offline” but the VM is fine
- Nobody knows when tokens were created
- Emergency token rotation at 2am

Why This Happens

Runner authentication tokens expire after 90 days (GitHub) or on revocation (GitLab). Token expiration is silent… no warning emails, no countdown in the UI. You find out when jobs fail.

The Fix

Track token creation dates:

```
# GitHub - Check runner token age
gh api repos/{owner}/{repo}/actions/runners \
  --jq '.runners[] |
    {name: .name, created: .created_at}'
```

Set calendar reminders:

- Day 60: Warning - tokens expire in 30 days
- Day 80: Critical - tokens expire in 10 days
- Day 85: Emergency - rotate NOW

Better: Use short-lived tokens with automation

For GitHub Actions, use Just-in-Time (JIT) runners:

```
# Generate JIT token valid for 1 hour
gh api \
  repos/{owner}/{repo}/actions/runners/registration-token \
  --method POST --jq '.token'
```

For GitLab, use runner authentication tokens:

```
# Rotate runner token
curl --request POST \
  --header "PRIVATE-TOKEN: $GITLAB_TOKEN" \
  "https://gitlab.com/api/v4/runners/" \
  "$RUNNER_ID/reset_authentication_token"
```

References:

- GitHub: Autoscaling with self-hosted runners
- GitLab: Runner authentication tokens

The Root Cause: GitOps Isn’t “Set and Forget”

Every one of these pain points comes from the same root cause: **CI/CD gets set up once, then ignored until it breaks.**

Your runners are infrastructure. They need:

- Monitoring and alerting
- Regular maintenance windows
- Capacity planning
- Security scanning
- Documentation

If you wouldn’t run a production database without monitoring, why would you run your deployment pipeline that way?

Quick Wins Checklist

Copy this and check off what you've done:

- ☐ Docker cleanup cron job on all runners
- ☐ Prometheus/metrics enabled on runners
- ☐ Alerting for runner offline status
- ☐ Pre-commit hook with gitleaks installed
- ☐ Monthly runner inventory audit scheduled
- ☐ Token expiration calendar reminders set
- ☐ Runbook documenting how to add/remove runners

Need Help?

If your CI/CD is held together with duct tape and prayers, let's talk.

Book a free 30-minute audit call:
📅 cal.com/joelhanger/discovery

I'll look at your setup, identify the biggest risks, and give you a prioritized fix list… whether you hire me or not.

More resources:

- 📕 5 Red Flags Guide - Warning signs in any codebase
- 🌐 tacitsoft.dev - Fractional DevOps for SaaS teams

Joel Hanger is a fractional DevOps consultant helping SaaS teams (10-50 employees) build infrastructure that doesn't page you at 2am. Based in Cheyenne, WY.

v1.0.8 | Feb 2026 | Ref: a4f2cec | tacitsoft.dev