Grafana Troubleshooting Runbook Pack

A senior engineer's incident runbook for datasources, panels, alerting, auth, and the database behind Grafana. DevOps AI ToolKit · devopsaitoolkit.com

Grafana sits between your humans and your data: a browser talks to the Grafana server, the server proxies to datasources (Prometheus, Loki, SQL), reads its own config DB for dashboards/users/alerts, and evaluates alert rules on a schedule. When something breaks, the symptom (blank panel, red datasource, silent alert) is rarely where the fault is. Work from the outside in: confirm the server is healthy, then the datasource, then the query, then the config layer that feeds it. Read the server log first — Grafana is unusually good about logging the real cause with `lvl=error`.

> Examples assume a Linux package or Docker install with the server reachable as `grafana-1:3000` and Prometheus as `prometheus:9090`. Adjust paths for your install (`/etc/grafana`, `/var/lib/grafana`, container `/etc/grafana`). Run read-only checks first; only restart or edit provisioning with a change record.

1. First-Response Triage

```
# Is the server process up and what did it log last?
systemctl status grafana-server --no-pager          # or: docker logs --tail=120 grafana
journalctl -u grafana-server -n 120 --no-pager | grep -Ei "lvl=error|lvl=warn|panic"

# Health endpoint — no auth required, returns DB + version
curl -s http://grafana-1:3000/api/health
# {"commit":"...","database":"ok","version":"11.3.0"}

# Datasource inventory (needs auth: API token or basic auth)
curl -s -H "Authorization: Bearer $GRAFANA_TOKEN" http://grafana-1:3000/api/datasources \
  | jq -r '.[] | "\(.name)\t\(.type)\t\(.url)\t\(.access)"'
```

- `"database":"ok"` but panels fail → the config DB is fine; the problem is a datasource or query, not Grafana itself.
- `"database":"failed"` or the health call hangs → go straight to §8 (database).
- Note the `version` — half of "it's broken" reports are a known bug fixed in a later point release. Compare against the log line `Starting Grafana ... version=`.

2. Datasource Errors (Bad Gateway / timeout)

```
# Reproduce what Grafana's proxy does, from the Grafana host/container
curl -s -o /dev/null -w '%{http_code} %{time_total}s\n' http://prometheus:9090/-/healthy

# Test a datasource the same way the UI's "Save & test" button does (id from §1)
curl -s -H "Authorization: Bearer $GRAFANA_TOKEN" \
  http://grafana-1:3000/api/datasources/uid/<DS_UID>/health
```

```
# grafana.ini — proxy timeouts that surface as 502/504 on slow datasources
[dataproxy]
timeout = 30
dialTimeout = 10
keep_alive_seconds = 30
```

- `502 Bad Gateway` / `dial tcp: connection refused` → Grafana can reach nothing at that URL. DNS or the datasource is down, not Grafana.
- `504` / `context deadline exceeded` → the datasource answered too slowly; the query or the backend is the problem, raise `[dataproxy] timeout` only as a stopgap.
- **Proxy vs direct (Browser) access**: `access: proxy` means the Grafana *server* must reach the URL (use `prometheus:9090`); `access: direct` means the *browser* must — a common cause of "works in curl, fails in UI". Prefer proxy.
- TLS: `x509: certificate signed by unknown authority` → add the CA via `tlsCACert` in the datasource `jsonData` / `secureJsonData`, or (lab only) set `tlsSkipVerify: true`.

3. No Data in Panels

```
# Run the panel's exact PromQL against the datasource directly
curl -s "http://prometheus:9090/api/v1/query?query=up" | jq '.data.result | length'
```

- Use the panel's **Query Inspector** (Panel → Inspect → Query) to see the *actual* request Grafana sent and the raw JSON response. "No data" with a 200 and empty `result` = your query matched nothing, not a Grafana fault.
- **Time range**: the panel may be querying a window with no samples. Reset to "Last 6 hours" before blaming anything. Check the datasource/host timezone too.
- **Datasource UID mismatch**: a dashboard imported from elsewhere references a `datasource` UID that doesn't exist here → "Datasource

`<uid>` was not found". Repoint the panel or provision the datasource with a fixed `uid`.

- `too many outstanding requests`: this is Prometheus/backend concurrency, not Grafana. The backend is saturated — reduce panel count, widen the interval, or raise the backend's query concurrency.

---

4. Templating & Variables

```
# "Templating [X] failed to load" almost always = the variable's query datasource failed.
journalctl -u grafana-server -n 80 --no-pager | grep -Ei "template|variable|datasource.*not found"
```

- **"Templating failed to load"** on dashboard open → a variable's underlying query errored (bad datasource UID, datasource down, or a bad `label_values(...)` expression). Fix the variable's datasource before anything else on the board.
- Open **Dashboard settings → Variables**, edit the variable, and check the live preview of values. Empty preview = the query returns nothing for the current time range.
- **Chained variables**: variable B filters on `$A`. If A resolves to empty, B's query breaks. Verify A first, and confirm B's query actually references `$A` (e.g. `label_values(node_uname_info{job="$A"}, instance)`).
- After a datasource UID changes, variables referencing the old UID silently fail — re-select the datasource in each variable.

---

5. Alerting Not Firing / Mis-Routing

```
# Confirm unified alerting is on (legacy alerting was removed in Grafana 11)
grep -A3 "\[unified_alerting\]" /etc/grafana/grafana.ini
journalctl -u grafana-server -n 200 --no-pager | grep -Ei "alerting|scheduler|notifier|receiver"
```

```
[unified_alerting]
enabled = true
[unified_alerting.screenshots]
capture = false
```

- **Rule not evaluating**: check the rule's *Health* in Alerting → Alert rules. `error`/`nodata` means the query failed, not that the condition is false. A rule only fires after it stays `pending` for its full `for:` duration.
- **Contact point**: send a test notification from Alerting → Contact points. If the test fails, it's SMTP/webhook creds, not the rule.
- **Notification policies**: an alert routes by matching labels down the policy tree. A too-broad top-level route or a wrong `group_by` sends alerts to the wrong receiver or groups them into silence.
- **Silences & mute timings**: check Alerting → Silences for a stale silence swallowing the alert. This is the most common "why didn't it page" cause.

---

6. Provisioning Failures

```
# /etc/grafana/provisioning/datasources/prometheus.yaml
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    uid: prom-main          # pin the UID so dashboards resolve across installs
    isDefault: true
```

```
# /etc/grafana/provisioning/dashboards/main.yaml
apiVersion: 1
providers:
  - name: 'default'
    folder: 'Provisioned'
    type: file
    options:
      path: /var/lib/grafana/dashboards
```

```
# Provisioning is applied ONLY at startup/reload — errors go to the log
journalctl -u grafana-server -n 120 --no-pager | grep -Ei "provisioning|failed to provision"
ls -l /etc/grafana/provisioning/datasources/ /var/lib/grafana/dashboards/
```

- Provisioning changes need a **restart** (`systemctl restart grafana-server`) or an admin reload — editing the YAML alone does nothing.
- `permission denied` reading provisioning files → they must be readable by the `grafana` user (`chown grafana:grafana`, `chmod 640`). In Docker, the mounted files must be readable by UID 472.
- A provisioned object cannot be deleted from the UI ("cannot delete provisioned...") — remove it from YAML instead. Invalid YAML aborts that provider silently except for one log line, so always grep the log after a change.

---

7. Auth & SSO

```
# grafana.ini
[server]
root_url = https://grafana.example.com/     # MUST match the real external URL

[auth.generic_oauth]
enabled = true
client_id = grafana
auth_url = https://idp.example.com/authorize
token_url = https://idp.example.com/token
role_attribute_path = contains(roles[*], 'admin') && 'Admin' || 'Viewer'
```

```
journalctl -u grafana-server -n 150 --no-pager | grep -Ei "oauth|ldap|login|Invalid username|origin not allowed"
```

- **"Invalid username or password"** for a local user → reset it: `grafana-cli admin reset-admin-password '<newpass>'`.
- **LDAP/OAuth login fails** → the log names the reason (bad `role_attribute_path`, cert error, or user in no mapped org). Test LDAP with the built-in "Test LDAP" page under Server Admin.
- **Wrong org/role** after login → check `role_attribute_path` (OAuth) or LDAP `group_mappings`; a user with no mapping lands as `Viewer` or is rejected.
- **API access**: prefer **service accounts + tokens** over legacy API keys. `401` on the API = missing/expired token; `403` = token role too low.
- **"origin not allowed"** on POST/API from the UI → `root_url` (and `[security] cookie_samesite`) don't match the URL the browser used, usually a reverse-proxy misconfig.

---

8. Database Issues

```
# Which backend? sqlite3 (default), mysql, or postgres
grep -A5 "\[database\]" /etc/grafana/grafana.ini | grep -Ei "type|host|name"
journalctl -u grafana-server -n 120 --no-pager | grep -Ei "database|migration|locked|dial"
```

```
[database]
type = postgres
host = postgres:5432
name = grafana
user = grafana
# For SQLite under load, reduce lock contention:
# type = sqlite3
# cache_mode = private
```

- `database is locked` **(SQLite)** → two Grafana processes hitting one file, an NFS-mounted DB, or slow disk. SQLite does not support HA — move `/var/lib/grafana/grafana.db` to local disk, and never run two replicas against it.
- **Migration failure** on startup (`migration failed ... code=`) → usually a partial upgrade or a manually edited schema. Restore the DB from backup and re-run the target version; do not hand-edit tables.
- **HA / Postgres/MySQL backend** → all Grafana replicas must share one external DB. `dial tcp: connection refused` to the DB host means the backend is down or firewalled; Grafana will not start until it can migrate.

---

9. Plugins & Rendering

```
grafana-cli plugins ls
journalctl -u grafana-server -n 120 --no-pager | grep -Ei "plugin|signature|unsigned|renderer"
```

```
[plugins]
# Only for trusted internal/dev plugins — never blanket-allow in production
allow_loading_unsigned_plugins = my-internal-panel

[rendering]
server_url = http://renderer:8081/render
callback_url = http://grafana-1:3000/
```

- `plugin signature invalid/unsigned` → Grafana refuses to load it. Install the signed version, or (dev only) name it in `allow_loading_unsigned_plugins`.
- **PDF/PNG export or alert screenshots fail** ("image renderer plugin not found") → you need the **grafana-image-renderer** plugin or the standalone renderer service, plus `[rendering] server_url`. The renderer must be able to reach Grafana at `callback_url`.
- After any plugin change, restart the server — plugins load at startup only.

---

10. Performance & Memory

```
# OOM kills show up in the kernel log, not Grafana's
dmesg -T | grep -i "killed process .*grafana"
systemctl status grafana-server --no-pager | grep -Ei "memory|oom"

# Slow dashboard? Inspect → Query shows per-query timing; count concurrent panels.
grep -A3 "\[dataproxy\]" /etc/grafana/grafana.ini
```

```
[dataproxy]
row_limit = 1000000
[dashboards]
min_refresh_interval = 10s     # stop 1s auto-refresh dashboards hammering datasources
```

- **OOM** → a huge query result (unbounded SQL, high-cardinality PromQL) pulled into memory, or too many concurrent renders. Cap results and raise `min_refresh_interval`.
- **Slow dashboard load** → too many panels each firing a query. Use Query Inspector timings to find the slow panel; the fix is almost always the query or the datasource, not Grafana.
- **Query concurrency**: a wall of panels on 5s refresh can trigger `too many outstanding requests` at the datasource (§3). Widen refresh, reduce panels, or split the board.

---

11. Decision Tree

```
Grafana symptom?
 ├─ /api/health database != ok? ──── yes → §8 database (locked / migration / backend down)
 └─ no (server healthy)
     ├─ Datasource red / 502 / 504? ─ yes → §2 (reachability, access mode, TLS, dataproxy timeout)
     ├─ Panel "No data"? ──────────── yes → §3 (Query Inspector → time range → UID → backend concurrency)
     ├─ "Templating failed to load"? ─ yes → §4 (fix the variable's datasource/query first)
     ├─ Alert didn't fire/mis-routed? ─ yes → §5 (rule health → silence → notification policy → contact point)
     ├─ Provisioned change ignored? ── yes → §6 (restart to apply; check file perms + YAML in log)
     ├─ Can't log in / wrong role? ─── yes → §7 (root_url, role_attribute_path, service-account token)
     ├─ Plugin/PDF/PNG broken? ─────── yes → §9 (signature, image renderer, callback_url)
     └─ Slow / OOM? ────────────────── yes → §10 (cap results, min_refresh_interval, per-query timing)
```

---

12. Copy/Paste One-Shot Triage

```
G=http://grafana-1:3000; DS=http://prometheus:9090
echo "== health ==";        curl -s $G/api/health
echo "== version/log ==";   journalctl -u grafana-server -n 40 --no-pager | grep -Ei "version=|lvl=error"
echo "== datasource up =="; curl -s -o /dev/null -w "prometheus: %{http_code} %{time_total}s\n" $DS/-/healthy
echo "== datasources ==";   curl -s -H "Authorization: Bearer $GRAFANA_TOKEN" $G/api/datasources | jq -r '.[]|"\(.name)\t\(.type)\t\(.access)\t\
(.url)"'
echo "== provisioning ==";  journalctl -u grafana-server -n 120 --no-pager | grep -Ei "provisioning|failed to provision" | tail
echo "== alerting ==";      journalctl -u grafana-server -n 120 --no-pager | grep -Ei "scheduler|notifier|receiver" | tail
echo "== oom ==";           dmesg -T 2>/dev/null | grep -i "killed process .*grafana" | tail
```

---

13. Incident Notes Template

```
INCIDENT: Grafana outage / degradation
Started (UTC):        ____   Detected by: ____   Sev: ____
Impact:               (which dashboards / teams / is alerting affected?)
Server health:        [ ] /api/health database=____  version=____
Layer confirmed:      [ ] server  [ ] datasource  [ ] query/panel  [ ] variable
                      [ ] alerting  [ ] provisioning  [ ] auth/SSO  [ ] database  [ ] plugin/render  [ ] perf/OOM
Datasource evidence:  name ____  access ____  direct probe ____ (code/time)
Query evidence:       (Query Inspector) status ____  result count ____  time range ____
Alerting evidence:    rule health ____  silence? ____  policy/receiver ____
DB evidence:          type ____  locked/migration/backend ____
Action taken:         (config edited / restarted / provisioning fixed + who + when)
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (timeouts, refresh limits, HA DB, signed plugins, alerts on Grafana itself) ____
```

DevOps AI ToolKit · devopsaitoolkit.com — practical AI workflows for real cloud engineers. Free to use for your own and your employer's internal work; please don't resell or republish the pack itself. Commands are provided as-is; always validate against your own environment before running in production. When you're stuck, paste the errors into the free AI Incident Response assistant at devopsaitoolkit.com/dashboard/incident-response.