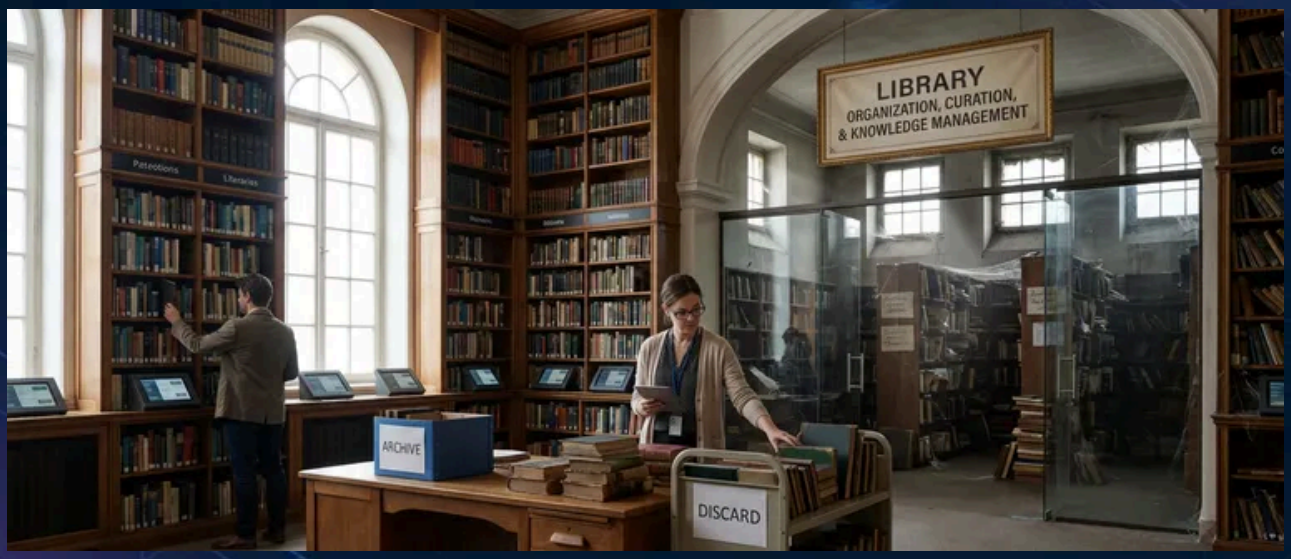
Dashboard Rot: Pruning Grafana for Actionability



Published on March 2, 2025

Webstack Builders

Table of Contents

It's 3 AM and the payments service is down. The on-call engineer opens Grafana, searches for "payments," and finds 47 results. Some are personal experiments from an engineer who left two years ago. Some are labeled "Copy of Copy of Payments Dashboard." Some show metrics from a version of the service that was deprecated in 2022. Ten minutes into the incident, they still haven't found the authoritative payments dashboard.

This scenario plays out constantly. Grafana makes it trivially easy to create dashboards — as it should — but nobody deletes them. Over time, organizations accumulate hundreds of dashboards: unowned, outdated, or unused. Engineers waste time searching through noise, critical dashboards get buried in clutter, and the storage costs pile up. Worse, stale dashboards actively mislead. That "Payments Health" dashboard showing all green? It's pulling from a datasource that was decommissioned six months ago. The proliferation that seemed harmless has now extended MTTR (Mean Time To Recovery).

> **⚠ WARNING**
>
> Every dashboard you create is a promise to maintain it. Dashboards without owners become misleading artifacts — they show stale data, use deprecated metrics, and waste time during incidents when accuracy matters most.

The Dashboard Lifecycle Problem

How Dashboards Proliferate

Dashboard accumulation isn't anyone's fault — it's a natural consequence of how teams work. Every dashboard enters your inventory through a legitimate path, and almost none ever leave.

New services get dashboards

When a team launches a service, they create a dashboard for it. This is correct behavior. But when the service is deprecated or merged into another, the dashboard stays. Nobody thinks to delete it because nobody's tracking which dashboards map to which services.

Incidents spawn debug dashboards

During an outage, someone creates "DEBUG - payment latency investigation" to correlate metrics. The incident gets resolved. The dashboard sits there forever because deleting it feels like destroying evidence. Six months later, there are forty "DEBUG-" dashboards, and nobody remembers what any of them were investigating.

Cloning is easier than customizing

An engineer wants the payments dashboard but with their team's specific filters. They clone it. Now there are two dashboards. The original gets updated; the clone diverges. Next quarter, there are seven variants, all slightly different, none clearly authoritative.

Teams reorganize but dashboards don't

When Platform Team A splits into Platform Infra and Platform Services, nobody updates dashboard ownership. The dashboards from the old team become orphans. People are reluctant to touch them because "Platform Team A" might still need them — even though that team no longer exists.

Nobody deletes anything

Deletion feels risky. What if someone needs it? What if there's data we can't recreate? The safe choice is always to keep it. Multiply this instinct across an organization and you guarantee accumulation.

Each of these patterns has a different expected lifecycle, but they all share the same outcome: permanent residency in your Grafana instance. Understanding the intended lifespan of each type is key to building cleanup policies that make sense — you wouldn't apply the same retention rules to a service overview dashboard and an incident debug dashboard.

Dashboard Type	Creation Frequency	Typical Lifespan	Deletion Rate
Service overview	Once per service	Years	< 5% ever deleted
Incident debug	Every incident	Should be days	< 1% deleted
Feature launch	Per feature	Should be weeks	< 10% deleted

Dashboard Type	Creation Frequency	Typical Lifespan	Deletion Rate
Personal exploration	Frequently	Should be hours	< 5% deleted
Team clones	Per team	Forever (diverges)	Never

Dashboard types and their lifecycle patterns.

The asymmetry is stark: creating a dashboard takes minutes, deciding to delete one feels like it requires a committee. This asymmetry guarantees accumulation unless you actively counteract it with process.

The Hidden Cost of Dashboard Accumulation

Dashboards aren't free. They consume resources even when nobody's looking at them.

Data source pressure

Every panel issues queries to your backend — Prometheus, InfluxDB, your SQL warehouse. A dashboard with 20 panels viewed by 50 people triggers 1,000 queries every refresh cycle. Multiply that by hundreds of dashboards, and your metrics infrastructure spends significant capacity serving dashboards that provide no value.

Server-side overhead

Grafana proxies every query, handles authentication, and sometimes performs transformations like joining data from multiple sources. Unused dashboards still consume compute resources when anyone stumbles across them or when scheduled reports run.

Alerting burden

Grafana-managed alerts linked to dashboard panels run their queries continuously, checking for firing conditions. That DEBUG dashboard from last quarter's incident? Its five alert rules are still running in the background, querying your metrics database every minute, even though nobody remembers what they were alerting on.

The resource cost of 500 dashboards versus 200 dashboards is real. During an incident — exactly when you need your observability infrastructure performing well — unused dashboards compete for the same query capacity as the critical ones you're actually trying to use.

> **INFO**
>
> The lifecycle reality: creation is easy (click, clone, copy), maintenance is rare (only when something breaks), review is never (who has time?), and deletion is almost never (too risky). Without deliberate intervention, your dashboard count only goes up.

Measuring Dashboard Health

You can't make evidence-based decisions about what to keep or delete without usage data. The first step in any hygiene program is instrumenting your Grafana instance to track who's looking at what.

What to Track

For each dashboard, you need four categories of metrics:

View metrics

Tell you whether anyone actually looks at this dashboard. Track views in the last 30 days, unique viewers (one person looking 50 times is different from 50 people looking once), and the last viewed date. A dashboard that hasn't been viewed in 90 days is a strong deletion candidate.

Edit metrics

Indicate maintenance activity. Track the last edit date, total edit count (the version number), and who made the last edit. A dashboard that was created two years ago and never edited is likely stale — either it's perfect (unlikely) or it's been abandoned.

Ownership metadata

Identifies who's responsible. Look for owner tags, team assignments, and the original creator. Dashboards without clear ownership are the hardest to make decisions about because nobody feels empowered to delete them.

Quality indicators

Flag technical problems. Count broken panels (those referencing deleted datasources), total panel count (excessive complexity suggests the dashboard should be split), and check for common naming anti-patterns like "Copy of" or "test" in the title.

Grafana Enterprise includes usage statistics out of the box. For Grafana OSS, you can track views through API (Application Programming Interface) audit logs, a reverse proxy that logs dashboard access, or custom middleware. The implementation details matter less than having **some** data to work with.

Calculating Health Scores

Raw metrics are useful but overwhelming when you have hundreds of dashboards. A composite health score (0-100) lets you quickly identify candidates for review.

A reasonable weighting:

`Usage score (35%)`	**Based on views and unique viewers in the last 30 days.** Zero views scores zero; 100+ views with 10+ unique viewers scores full marks.
`Freshness score (25%)`	**Based on time since last edit.** Edited in the last 30 days scores 100; over a year without edits scores 0.
`Ownership score (25%)`	**Has an assigned owner?** Has a team tag? Has any tags at all? Each adds points.
`Quality score (15%)`	**Penalize broken panels heavily (they actively mislead),** excessive complexity (30+ panels), and suspiciously simple dashboards (fewer than 3 panels might be incomplete).

Beyond the numeric score, flag dashboards with specific problems: “no-recent-views,” “stale,” “no-owner,” “broken-panels,” “unmaintained,” “likely-clone” (title contains “copy”), “possibly-temporary” (title contains “test”), or “incident-artifact” (title contains “debug”).

These flags help prioritize review. A dashboard scoring 40 with “broken-panels” needs immediate attention — it’s actively misleading anyone who looks at it. A dashboard scoring 40 with “no-recent-views” can wait for the next cleanup cycle.

> ✓ SUCCESS
>
> Automate usage tracking from day one. Without data, hygiene decisions become political debates instead of evidence-based choices. “Nobody uses this dashboard” is much more persuasive when you can prove it with numbers.

Dashboards as Code

Grafana stores dashboard definitions in its database by default — SQLite for small deployments, PostgreSQL or MySQL for larger ones. This works fine for casual use, but it creates problems at scale: no audit trail beyond Grafana’s internal version history, no review process for changes, and no way to roll back across multiple dashboards simultaneously.

The alternative is treating dashboards as code. Instead of editing in the browser, you define dashboards in a repository and deploy them through CI/CD.

- **Generation tools**
  Writing raw JSON is tedious and error-prone. Tools like Jsonnet (with the Grafonnet library), Python (with grafanalib), or Terraform's Grafana provider let you define dashboards programmatically. You get variables, loops, and functions — create a template once, generate dashboards for all your services.
- **Version control workflow**
  Dashboard changes go through pull requests. Someone reviews the diff. Changes deploy automatically through provisioning or the Grafana API. You get audit history, easy rollback, and the ability to see exactly what changed when a dashboard stopped working.
- **When to use which approach**
  Browser editing is fine for personal exploration dashboards and one-off debugging. Code-managed dashboards are better for anything canonical — service health dashboards, SLO dashboards, team operational views. The rule of thumb: if multiple people rely on it, it should be in version control.

Grafana’s built-in version history still works for code-provisioned dashboards. If someone makes an emergency fix in the browser, you can see the diff and decide whether to backport it to the code or revert it.

The Hygiene Process

Health scores tell you which dashboards need attention. A governance process ensures that attention actually happens. Without formal processes, cleanup becomes a heroic individual effort that burns people out.

Governance Policies

Before you can hold people accountable, you need clear expectations. Document these upfront:

1. **Ownership requirements**
   Every dashboard must have an owner. Owners keep dashboards accurate, respond to questions, update or delete when services change, and hand off ownership when leaving a team. Ownership isn't optional — it's the foundation of accountability.

2. **Folder structure**
   Organize folders by domain. Team dashboards go under team names. Service dashboards go under service names. SLO dashboards get their own area. Incident investigation dashboards live in a flat "Incidents" folder with auto-expiration. Personal exploration dashboards live in personal folders and are exempt from most governance (but not from usage tracking).

3. **Naming conventions**
   Use a consistent pattern like "[Service/Team] - [Purpose] - [Scope]." Good examples: "Payments - API Performance - Production" or "SLO - Checkout Success Rate - P99." Bad examples: "New Dashboard," "Copy of Copy of...," "Test123," or "DEBUG - delete me." Names that violate the pattern are automatic review candidates.

4. **Tagging requirements**
   Require `owner:{username}` and `team:{team-name}` tags on every dashboard. Recommend additional tags for service, environment, and purpose. Tags make bulk operations possible — you can't email all owners of payment-related dashboards if there's no consistent way to identify them.

The Automated Hygiene Cycle

Manual hygiene doesn't scale. The goal is a system that runs itself with minimal human intervention:

Daily

Collect usage metrics. Store them for historical analysis.

Weekly

Calculate health scores. Generate reports. Take automated actions based on score thresholds.

For dashboards scoring above 50, no action needed — they're healthy enough.

For dashboards scoring between 30 and 50, send a warning email to the owner. Give them two weeks to respond with one of three options: keep (with documented justification), delete it themselves, or ignore the warning (which escalates to their team lead after 30 days).

For dashboards scoring below 30, the automation takes over. After the two-week warning period with no response, export the dashboard JSON (JavaScript Object Notation) to cold storage and delete from Grafana. Notify the owner that their dashboard was archived. If they need it back, they can claim it within 90 days. After 90 days, permanent deletion.

> ⚠ **WARNING**
>
> Always archive before deleting. A 90-day cold storage period catches the "but we need that during quarterly reviews!" case. Export the JSON (JavaScript Object Notation), store it somewhere searchable, and only permanently delete after the retention period with no claims.

This creates accountability. Dashboard owners either maintain their dashboards or lose them. The system is predictable and fair — everyone knows the rules, and the automation applies those rules consistently.

Actionable Dashboard Design

Cleaning up existing dashboards is only half the battle. Without standards for **new** dashboards, you'll be right back where you started in a year. The goal is dashboards that drive action, not dashboards that display data.

Six Questions for Every Dashboard

Before creating or keeping a dashboard, it should pass these tests:

1. **Does it answer a specific question?**
   Good: "Is checkout latency within SLO?" Bad: "Here are some checkout metrics." If you can't state the question in one sentence, the dashboard is too unfocused.

2. **Does every panel have clear thresholds?**
   Good: "Red when p99 > 500ms, yellow when > 300ms." Bad: "Shows p99 latency (interpret as you wish)." Without defined good/bad ranges, viewers have to guess what "normal" looks like.

3. **Does it drive specific actions?**
   Good: "If red, scale horizontally or investigate slow queries." Bad: "If something looks wrong, investigate." Every alert state should have a documented response, ideally linked to a runbook.

4. **Does the time range match the use case?**
   Good: "Last hour for operational monitoring, last 30 days for trend analysis." Bad: "Default 6-hour range for everything." Operational dashboards need tight windows; capacity planning dashboards need long ones.

5. **Does it load quickly?**
   Good: "Renders in under 3 seconds." Bad: "Takes 30 seconds, times out during incidents." A slow dashboard is useless when you need it most. If it can't load under pressure, simplify the queries.

6. **Does it have a single purpose?**
   Good: "Service health overview OR deep-dive debugging." Bad: "Everything about the service on one page." If you can't describe the purpose without using "and," split it into multiple dashboards.

Anti-Patterns to Eliminate

Four dashboard smells that signal problems:

Vanity dashboards

Show metrics that look impressive but drive no action. The classic example is "Total requests served"—an ever-increasing counter that tells you nothing useful. Replace with rate of change or comparison to baseline.

Kitchen sink dashboards

Have too many panels with no clear narrative. Fifty panels showing every available metric helps no one. Split into an overview dashboard plus detail dashboards for specific investigations.

Wall of green dashboards

Have thresholds set so permissively that nothing ever triggers. If everything always looks fine, the dashboard provides no signal. Calibrate thresholds to actual SLOs and historical variance.

Mystery meat dashboards

Have panels with no titles, unlabeled axes, or missing legends. A graph with y-axis labeled "value" is useless without context. Every panel needs a title, description, and unit labels.

The Template Approach

Rather than policing dashboard standards manually, encode them in templates. A well-designed service health template includes:

SLO Status row
Error budget remaining (gauge with percentage thresholds), 30-day rolling availability (stat with SLO target threshold)

Traffic & Latency row
Request rate with anomaly alerting, latency distribution showing p50/p95/p99 with threshold lines

Errors row
Error rate as percentage with target threshold, error breakdown by type (pie chart for quick triage)

Resources row
CPU and memory utilization (watch for gradual increases), instance count compared to expected baseline

Links section
Deep-dive dashboard, runbook URL, log explorer query

When creating a new service dashboard is as simple as filling in `${service_name}` and `${team}` variables, developers naturally produce dashboards that follow your standards. Templates also make bulk updates possible — change the template, regenerate all derived dashboards.

> **INFO**
>
> Templates enforce consistency and encode best practices. When the path of least resistance produces good dashboards, you don't need to police standards — they emerge naturally.

Pruning Strategies

With standards for new dashboards in place, let's address the existing inventory. Health scores identify **which** dashboards need attention. A decision framework determines **what** to do with each one.

The Decision Tree

Start with the most important question: has anyone viewed this dashboard in the last 90 days?

If no	Check for an assigned owner. No owner and no views? Delete it — this is an orphaned, unused dashboard. If there is an owner, contact them. If they confirm they need it despite no views (perhaps it's quarterly reporting), document the justification and keep it. If they don't need it, delete it.
If yes, but barely (fewer than 5 views per month)	Check if it's a single viewer. If one person created it and they're the only one viewing it, it's a personal dashboard — move it to their Personal folder where it won't clutter team searches. If a single viewer is using someone else's dashboard, investigate why only one person finds it useful. If multiple people view it even occasionally, keep it — there's value there.

If it's actively viewed	Check for broken panels (panels referencing deleted datasources). Fix or remove them — broken panels actively mislead anyone looking at the dashboard. Then check for duplicates. If this dashboard is substantially similar to another with more usage, consolidate them: keep the more-used one, redirect users of the other, then delete it. If it's healthy and not a duplicate, keep it.

Five possible outcomes for every dashboard:

#	Outcome	Criteria	Action
1	Delete	No views, no owner, or owner confirms unneeded	Remove permanently
2	Move	Personal use only, low value to team	Move to Personal folder
3	Consolidate	Duplicate of higher-usage dashboard	Merge content, delete duplicate
4	Fix	Has broken panels	Repair or remove broken panels
5	Keep	Healthy, used, not duplicate	No action needed

Finding Duplicates

Clone proliferation creates dashboards with similar titles: “Payments Dashboard,” “Payments Dashboard - Copy,” “Payments Dashboard (team B),” “payments-prod-v2.” Automated duplicate detection using title similarity (threshold around 80% match) groups these into clusters. For each cluster, pick a canonical version — typically the one with the most views and most recent edits — and consolidate the others into it.

Consolidation isn’t always deletion. Sometimes the variants have useful additions. Merge any unique panels into the canonical dashboard, update any bookmarks or links, then delete the variants.

Running Cleanup Campaigns

While the automated hygiene cycle handles ongoing maintenance, a quarterly cleanup campaign creates urgency and clears the backlog.

Four weeks before

Run the usage analysis and generate health scores. Identify all dashboards below threshold. Generate reports by team so leads know what's coming. Draft the campaign communication.

Two weeks before

Send preview reports to team leads. Announce the campaign dates, process, and deadlines. Open a window for exception requests — if someone needs to keep a low-scoring dashboard, they need to explain why now, not during the campaign.

Campaign week one

Send individual notifications to every dashboard owner with flagged dashboards. Open a self-service portal where owners can mark dashboards as "keep," "delete," or "archive." Send daily reminders to non-responders. Hold office hours for questions.

Campaign week two

Final warnings. Escalate to team leads for stragglers. Most owners respond in week one once they realize the deadline is real.

Week three

Auto-archive everything unclaimed. Send notifications with restore instructions.

Week seven

Delete archived dashboards with no restore requests. Publish results.

Campaign Phase	Duration	Expected Reduction
Owner self-service	2 weeks	15-25%
Auto-archive	1 week	10-20%
Post-archive claims	4 weeks	-5% (restorations)
Net result	~7 weeks	20-40% reduction

Typical cleanup campaign results.

> ✓ SUCCESS
>
> Run cleanup campaigns quarterly. A single big cleanup is less effective than regular maintenance. Teams that know cleanup is coming in 3 months naturally keep things tidier and delete their experiments before the campaign starts.

Organizational Change

Building a Deletion Culture

Process alone won't fix dashboard rot. People hoard dashboards because deletion feels risky and there's no reward for tidiness. You need to change the incentives.

- **Celebrate deletion**
  Make cleanup visible and valued. A monthly "cleanest team" recognition, where the team with the highest ratio of views-to-dashboards wins bragging rights, creates positive peer pressure. Track dashboard count as a negative metric — lower is better. Create a deletion hall of fame for the biggest cleanups. The message: deleting unused dashboards is responsible stewardship, not laziness.

- **Reduce creation friction**
  This sounds counterintuitive, but easier creation leads to less hoarding. When templates make creating a new service dashboard trivial, people stop clinging to old ones "just in case." When Explore mode handles ad-hoc analysis, people don't create permanent dashboards for temporary questions. When ephemeral dashboards auto-expire after a week, debugging artifacts clean themselves up. If you can recreate it easily, you don't need to keep it.

- **Manage ownership transitions**
  When people leave teams, their dashboards become orphans. Include dashboard ownership in offboarding checklists. Assign team-level ownership for critical dashboards so they survive individual departures. Run automatic orphan detection after someone leaves — their dashboards should get flagged for review within a week, not discovered during the next cleanup campaign. Dashboards are team assets, not personal artifacts.

- **Schedule maintenance time**
  If cleanup only happens when someone volunteers, it won't happen. Build it into regular cadences: monthly dashboard review as a standing agenda item in team meetings (takes 5 minutes), quarterly cleanup sprints as a defined team activity, annual comprehensive audit for anything that slipped through.

Maintenance is part of the job, not extra work.

What to Measure

Good metrics focus on outcomes, not outputs:

Engagement rate

Dashboards viewed per week divided by total dashboards. Higher is better — it means your dashboards are actually useful.

Average health score

Tracks portfolio quality over time. Should trend upward as you clean up and maintain.

Time to find the right dashboard during an incident

The metric that actually matters. If engineers can't find the right dashboard quickly, your observability isn't helping when it counts.

Dashboard load time p95

Slow dashboards get abandoned. Track this to catch complexity creep.

Bad metrics to avoid: total dashboard count (encourages hoarding to keep the number stable), dashboards created per month (encourages proliferation), panels per dashboard (encourages complexity without context).

Handling Resistance

Every cleanup effort encounters the same objections. Have responses ready:

"I might need it someday."

Archive, don't delete. We'll keep the JSON for 90 days. If you need it, you can restore it. In three years of cleanups, we've had two restore requests.

"It's not hurting anyone."

It is. During the last incident, engineers wasted 8 minutes finding the right dashboard among 47 search results. That's 8 minutes of extended outage. Document your own incident data to make this argument concrete.

"I don't have time." The hygiene automation handles 80% of cases. You only need to review dashboards flagged for your attention — usually 5-10 per quarter. Most reviews take under a minute each.
"Someone else might use it." Usage data shows nobody has viewed this in 6 months. If someone needs it, they'll create their own — which they'd do anyway since this one is outdated.
"It's my personal dashboard." Great! Move it to your Personal folder. Personal dashboards are exempt from cleanup but won't clutter team search results.
"The process is too bureaucratic." For most dashboards, it's one click: keep, archive, or delete. The detailed process is only for disputed cases. 90% of owners complete review in under 5 minutes.

> **WARNING**
>
> The biggest obstacle to dashboard hygiene isn't technical — it's cultural. People hoard dashboards because deletion feels risky and there's no reward for tidiness. Address the incentives, not just the process.

Conclusion

Dashboard hygiene isn't a one-time project. It's ongoing maintenance, like any other operational practice.

The core practices:

- **Track usage from day one** — You can't make evidence-based decisions without data.
- **Calculate health scores objectively** — Removes politics from cleanup decisions.
- **Automate the warning-archive-delete pipeline** — Manual processes don't scale.

- **Design actionable dashboards that answer specific questions** — Prevention is cheaper than cleanup.
- **Run regular cleanup campaigns** — Creates urgency and clears backlogs.
- **Build a culture where deletion is celebrated** — Process without culture change fails.

The goal isn't the smallest possible dashboard count. It's ensuring that every dashboard that exists is accurate, maintained, and serves a clear purpose. When an engineer searches for "payments" during an incident, they should find three relevant dashboards, not thirty stale ones.

> **SUCCESS**
>
> If you do nothing else: implement usage tracking and run one cleanup campaign. Those two actions will give you the data to make informed decisions and the momentum to build a sustainable hygiene practice.

> **INFO**
>
> A dashboard that nobody looks at is worse than no dashboard at all. It takes up search results, misleads anyone who stumbles on it, and creates the illusion of observability without the reality. Delete with confidence.

https://www.webstackbuilders.com/articles/grafana-dashboard-hygiene-pruning-actionable-metrics