Engineer

Grafana

a dashboard tool for metrics and logs

What it is

Grafana turns numbers kept in other systems into dashboards. It sends a query to a data source such as Prometheus, Loki, PostgreSQL or MySQL, draws what comes back as a time-series graph, a table, a gauge or a heatmap, and raises an alert when a value crosses a line someone has set. Its own documentation puts it as software to query, visualise, alert on and explore metrics, logs and traces “wherever they are stored”, and the last four words carry the design.

The telemetry on the screen stays where it was collected. Each panel fetches its slice from the data source when the dashboard loads or refreshes, through one of more than 150 data source plugins, and Grafana keeps only its own configuration: dashboards, folders, users, permissions and alert rules. Those live in a small database that is SQLite by default, embedded in the binary, with MySQL or PostgreSQL recommended for production. A dashboard is a JSON document: panels arranged in rows, each panel a query plus a choice of how to draw it, and variables along the top that let one dashboard switch between hosts, environments or services from a drop-down.

Grafana Labs develops it, alongside Loki for logs, Tempo for traces and Mimir for long-term Prometheus storage, a set people often call the Grafana stack. On 20 April 2021 Grafana, Loki and Tempo moved from the Apache 2.0 licence to AGPLv3. Grafana comes as the open-source edition, as Grafana Enterprise under a commercial licence, and as Grafana Cloud, which Grafana Labs hosts and runs.

It is usually weighed against Kibana, which plays the same part in front of Elasticsearch, against the expression browser built into Prometheus, and against hosted observability suites that sell storage and screens as one product.

Business-intelligence tools such as Metabase or Power BI can read the same databases, but they are built for reports a person studies, while Grafana is built for screens that refresh every few seconds.

Use the concept in a decision

Put “What is Grafana?” to work in a real decision. We can help relate the concept to the system you are building or buying. Monitoring and alerts that speak up while a fault is still small, and the figures that show how often it stays that way.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Built the company's first observability dashboard — the company's first view of its own systems

Built the company's first observability dashboard Get in touch →

Why it matters

Installing Grafana is the easy part. It runs on a small machine and can be pointed at a database a team already has. The decisions that shape what it is worth come afterwards: who each screen is for, how dashboards are kept once there are dozens of them, and where it runs.

Because it only displays, adopting it moves no data. A team can keep Prometheus, PostgreSQL and its log store where they are, and could swap the display layer later without migrating a single series, so the lock-in and the bill both sit with the data sources. Suites that sell storage and display together make the opposite trade: one bill and one login, and data that is harder to take elsewhere.

The audience shapes a dashboard more than the tool does. An engineer on call wants a page that answers “is it broken, and where” at a glance, with links into detail. A room of people with different jobs needs fewer and larger panels, plain titles and a threshold line that says when to worry. The same Grafana serves both through different dashboards, and a page built for both at once tends to serve neither.

How dashboards are kept is the second decision. Grafana can load them from JSON files on disk and watch the folder for changes, which its documentation calls provisioning. In version control a dashboard gets review, history and a way back, and a new environment gets the same screens without anyone clicking through them. The cost is that editing in the browser stops being how changes are made, and the whole team has to accept that.

Licence and hosting come last. Under AGPLv3, whoever modifies Grafana and lets others use the modified version over a network has to offer them its source. A team running an unmodified Grafana for its own staff stands somewhere very different from a vendor shipping a patched one inside a product, and the vendor should read the licence with counsel. Self-hosting is a single binary until a second instance is needed. From then on the instances share a MySQL or PostgreSQL database, and Grafana Cloud is the way to hand that work to Grafana Labs instead.

What it looks like in practice

Both times Grafana appears in this company's work, it is the part people looked at, with Prometheus behind it.

In an Infrastructure Engineer role from 2022 to 2024, the starting point was a team that heard about problems after users did, because there was no single view of how its systems were doing. A Grafana and Prometheus monitoring dashboard gave it that view. The key services were instrumented, the metrics were chosen because they meant something rather than because they looked busy, and alert thresholds were set on them and surfaced where the team would see them while there was still time to act. Problems began to be caught before they escalated, reliability improved, and the team moved from firefighting to getting ahead of trouble.

The other case was built for a mixed audience. A dashboard on a large office TV was meant for technical and non-technical colleagues alike. Its second version, on Grafana and Prometheus, showed metrics picked with the DevOps and engineering teams, fed by data pipelines that took in performance figures from several systems, with alerts when a threshold was breached. Teams detected and resolved incidents 40% faster, and with the numbers in front of the whole office, the health of the systems became something people owned together.

A wall screen like that is read from across a room by people who will never edit a query. It rewards a few large panels, units in the titles, and colours that change only when someone should act, and Grafana leaves every one of those choices to whoever builds the dashboard.

Where it goes wrong

A dashboard grows until nobody reads it. Each incident adds a panel and nothing takes one away, so a year later the page holds forty small graphs and the one that matters sits below the fold. Give each dashboard one audience and one question, put the answer to “is it healthy” in the top row, and move detail to dashboards linked from there. A panel that nobody on the team can explain is a panel to delete.

A fix made in the browser disappears. The dashboard was provisioned from a file. With `allowUiUpdates` at its default of false, Grafana refuses the save with a “Cannot save provisioned dashboard” dialog. Set to true, it saves the change to its database and then overwrites it with the file the next time the file changes. Edit the file, or export the JSON from the save dialog and commit it, and keep the provider’s setting explicit:

```
apiVersion: 1
providers:
    - name: operations
      type: file
      allowUiUpdates: false
      options:
          path: /var/lib/grafana/dashboards
```

The wall screen slows down the system it watches. Every open browser, the TV included, re-runs every panel’s query on each refresh, and five seconds is the shortest refresh Grafana allows out of the box. Twenty panels on a five-second refresh is four queries a second from one screen. Refresh at the pace the data changes, which for Prometheus means the scrape interval or slower, raise the floor for everyone, and precompute heavy expressions with recording rules in the data source:

```
[dashboards]
min_refresh_interval = 30s
```

The same incident pages people twice. The rule lives in Prometheus, routed through Alertmanager, and again in Grafana Alerting. Or two Grafana instances each evaluate every rule, because alerting was left out when the second one was

added. Give every rule a single home, and follow Grafana's alerting high-availability steps before a second instance goes live.

Dashboards and users change depending on which server answers. Two Grafana instances behind a load balancer each kept their own SQLite file, so a dashboard saved on one is missing on the other and a user exists on only one of them. Grafana's own guide makes a shared MySQL or PostgreSQL database a requirement for high availability.

Further reading

- About Grafana — the project's overview of what Grafana does, its editions and the open-source projects around it.
- Data sources — the systems Grafana can query, and how each one is added and configured.
- Dashboards — panels, rows, variables and the JSON model a dashboard is stored as.
- Provision Grafana — data sources and dashboards from files, and what happens to edits made in the browser.
- Set up Grafana for high availability — the shared database a second instance needs, and the extra steps for alerting.
- Licensing — Grafana Labs' page on the 2021 move from Apache 2.0 to AGPLv3.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Grafana?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/grafana/