Engineer

Monitoring

watching a running system so it tells you first

What it is

Monitoring is the continuous reading of what a live system reports about itself: something keeps holding those reports against lines someone drew and decides when a person needs to hear about it. Google's Site Reliability Engineering book defines monitoring as the collection, processing, aggregation and display of quantitative data about a system in real time: how many requests arrive and of which kinds, how many fail and how, how long they take, and how long each server has been running. The aim is an order of events: the system says it is in trouble before a user does.

The data comes from two directions. White-box monitoring reads what a system exposes about its own insides, such as a metrics endpoint, a queue length or a count of retries, so it can see trouble building before anyone feels it. Black-box monitoring stands outside and behaves like a user: it requests a page, resolves a name or completes a login, and reports whether that worked and how long it took. It sees only what is already broken, which is what makes it a trustworthy reason to wake someone.

Most of what a monitoring system keeps is numbers sampled over time, because numbers are cheap to store and easy to hold against a threshold. The same SRE chapter names four of them as the place to start for anything that serves users: latency, the time a request takes; traffic, how much demand arrives; errors, the share of requests that fail; and saturation, how close the busiest resource is to full.

Around the numbers sit two outputs. A dashboard is for looking, when someone already suspects something. An alert is for interrupting, when everyone is busy with something else. In the Prometheus ecosystem the server evaluates the rules, and Alertmanager groups, deduplicates and routes whatever fires.

Two neighbours are easy to confuse with it. Logging writes down individual events, one line per thing that happened, and is what someone reads after an alert to find out why. Observability, as the OpenTelemetry project frames it, is the wider ability to ask a system new questions from the outside. Monitoring is the narrower and older practice: questions decided ahead of time, asked continuously, with an answer that can page a person.

Use the concept in a decision

Put “What is monitoring?” 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

The alternative is hearing about trouble from the people it hurts, at whatever hour they notice. Monitoring moves that moment earlier and hands it to someone who can act.

The scarce resource is attention rather than software. A page interrupts work during the day and sleep at night, and the SRE book is blunt about what happens when pages come too often: people skim them, second-guess them and finally ignore them, sometimes including the real one hidden in the noise. The Prometheus project’s guidance fits in a sentence: “keep alerting simple, alert on symptoms, have good consoles to allow pinpointing causes, and avoid having pages where there is nothing to do”. Every rule that can wake someone is a promise that, when it fires, a person has something urgent to do.

That turns the central decision into one about what users feel. A disk at 85% is a cause; a checkout that fails is a symptom. Alerting on symptoms keeps the rule count small, because there are far fewer ways for users to be hurt than ways for machines to misbehave. Causes still belong on dashboards and in tickets, where they help diagnosis without waking anyone. Capacity is the honest exception: a resource close to full hurts nobody today, but it will need a person soon.

Then comes the question of running it or renting it. Running it means a time-series store, an alert router and someone who keeps both healthy. A hosted service takes that work away and bills by the volume sent, so every new host shows up on an invoice. The SRE book notes that even at Google, a team of 10–12 engineers typically has one or two whose main job is monitoring. Either way it is a system with its own upkeep, to be sized to the system it watches.

What it looks like in practice

Two of the founder's earlier roles and the company's own host show the same idea at very different sizes.

From 2015 to 2022, in an Infrastructure Engineer role, the job was 24/7 support for about 1,000 IPTV/OTT streaming servers, plus a good number of client-owned systems, for customers in China, the US and Germany. With customers in three time zones there was no quiet hour, and a problem at 3am local time was primetime somewhere else. Much of the work was noticing something drifting before it became an outage, and problems were dealt with before they reached a viewer's screen. On a service like that, success looks like nothing happening, and monitoring is what lets a team know that rather than hope it.

In a Database Engineer role from 2022 to 2024, across 100 PostgreSQL, MS SQL and BigQuery databases, automated monitoring reduced downtime by 50%.

Monitoring the company's own public-services host means working at the other end of the scale. A single 512 MB machine runs those public services, with about 464 MB of it usable, and memory is the argument that settles every design decision on it. Reporting there is a pull rather than an agent, a pipeline and a dashboard: seven commands that read the host and render Markdown, change nothing, and run only when someone asks. The onion-service decision shows the method: once the question came up, the host was sampled for a day and the readings settled it. In those recorded samples, the low point for available memory was about 310 MB of 464, swap held at 2.6% and the processor idled 99.7% of the time. At that size, monitoring is a measurement taken at the moment a decision needs one.

Where it goes wrong

The pager goes off so often that people stop reading it. Each incident leaves a rule behind, many of them fire on causes that clear by themselves, and within months the real page gets skimmed past with the rest. Page only on what users would notice, at one point in the stack, and send everything else to a ticket queue or a dashboard. Then let the condition hold for a while before it fires, so a blip that clears in a minute wakes nobody:

```
groups:
    - name: checkout
      rules:
          - alert: CheckoutErrorsHigh
            expr: job:checkout_errors:ratio5m > 0.02
            for: 10m
            labels:
                severity: page
```

The monitoring stops, and its silence reads as good news. A rule that never fires looks exactly like a system that never breaks. If the Prometheus server, the Alertmanager or the mail relay goes down, every alert stops at once. Prometheus's own advice is to alert on the monitoring itself, and better still to test the whole path end to end, so one check proves an alert can travel from rule to inbox. Scheduled jobs fall into the same trap: a backup that stops running raises no error, it simply stops leaving traces. Page when the last success is too old, allowing at least two full runs; for an hour-long job that runs every 4 hours, the project suggests 10 hours:

```
time() - backup_last_success_timestamp_seconds > 10 * 3600
```

The average says all is well while some users wait. Averages are easy to graph, and they hide the tail. The SRE book's example is a service averaging 100 ms at 1,000 requests a second, where 1% of requests can easily take 5 seconds; when a page depends on several such services, one backend's 99th percentile can become the frontend's median. Fast failures make it worse, because a quick error drags the average down. Record latency in buckets, alert on a high percentile, and track failed requests apart from successful ones.

Every internal number is green while users cannot reach the service. White-box metrics come from inside the system, so they share its blind spots: an expired certificate, a broken DNS record or a firewall change can leave every one of them healthy. A black-box check from a machine outside the system's network catches what is otherwise invisible, and keeps working when the internal monitoring goes down with everything else.

Further reading

- Prometheus: Overview — what a metrics-based monitoring system collects, which parts make it up, and where per-event precision calls for something else.
- Alerting — the Prometheus project's guidance on what to page on, from online services and batch jobs to monitoring the monitoring.
- Alertmanager — how alerts are deduplicated, grouped, silenced, inhibited and routed to a receiver.
- Monitoring Distributed Systems — chapter 6 of Google's Site Reliability Engineering book: symptoms and causes, the four golden signals, and the tail.
- Observability primer — OpenTelemetry's account of signals, SLIs and SLOs, and where observability reaches past monitoring.

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 monitoring?” 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/monitoring/