Engineer

Service level objective

a measurable reliability target that a service must meet

What it is

A service level objective is a target for how well a service behaves, written so precisely that anyone can check whether it was met: a share of requests that must succeed, or must be answered fast enough, over a stated period. Almost everyone shortens it to SLO. Google's Site Reliability Engineering book defines it as a target value, or a range of values, for one level of service, as measured by an indicator.

That indicator is the service level indicator, or SLI: a quantity defined with care to describe one aspect of the service users actually get. Request latency is the usual first choice, with error rate and throughput close behind. The SRE Workbook recommends writing every SLI as a ratio, good events divided by all events, so that it runs from 0%, where nothing works, to 100%, where nothing is broken. The objective is then a line drawn on that scale: the indicator must stay on the right side of a target, or between a lower and an upper bound.

A complete objective names everything needed to check it. The book's example reads: 99% of Get RPC calls, averaged over one minute, complete in under 100 ms, measured across all backend servers. One indicator can carry several targets at once, such as 90% of calls under 1 ms, 99% under 10 ms and 99.9% under 100 ms, because one percentile rarely describes a whole distribution. The Workbook also separates the specification, the outcome that matters to users, from the implementation, the place it is measured: a server log, a prober running a browser, or code in the page itself. Each one sees different failures, and the choice belongs in the definition.

An SLO is not an SLA. A service level agreement is a contract with users, explicit or implicit, that attaches consequences to meeting or missing the objectives inside it, most visibly a rebate or a penalty. The book's test is to ask what happens

when the objective is missed: if nothing explicit follows, it is an SLO. A footnote adds that people who talk about a violated SLA nearly always mean a missed SLO, since a real breach of an SLA can end in court for breach of contract. Google Search, the book notes, has no SLA for the public, but an outage of it still costs reputation and advertising revenue.

Use the concept in a decision

Put “What is a service level objective?” 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

Without a stated objective, users invent their own. The SRE book describes both ways that goes wrong: people who trust a service more than they should, and prospective users who think it flakier than it is and stay away. Its example is Chubby, Google’s lock service for distributed systems. Global outages of it were so rare that other services quietly came to depend on it never failing, and those services broke on the rare occasions it did. The answer was to make the global instance meet its SLO without exceeding it by much: in any quarter where real failures had not pushed availability below the target, SRE took the system down on purpose.

The objective also settles the argument about how reliable is reliable enough. “100% reliability is the wrong target”, writes the SRE Workbook, and its reasons are practical. Components fail at the same time however they are built. The chain of networks and devices between a service and its customers fails too, so the customers would never see the 100%. Each extra nine costs more than the last while its value to users approaches zero. And since change is the leading cause of outages, a service that may never fail can never be updated either. Once the target is below 100%, the gap between the target and perfection is room to spend on releases, experiments and planned maintenance: the error budget, which is simply 100% minus the SLO. Someone has to own that trade between shipping features and staying up; the Workbook names the CTO in a small organisation and the product owner or product manager in a larger one.

Finally, the objective turns monitoring into decisions. The book describes a loop: measure the indicators, compare them with the objectives, decide whether anything needs doing, and do it. If latency is climbing and will cross the line within a few hours, the team checks whether the servers are CPU-bound and adds more of them. Without the objective, the book points out, nobody can say whether to act, or when.

What it looks like in practice

Take a checkout API and an objective written the way the Workbook suggests, with the indicator separate from where it is measured, the window stated and the reasoning kept beside it:

```
Service:    checkout API
SLI:        share of POST /checkout requests answered without a 5xx
            status, counted in the server's request log
Objective:  99.9% over a rolling 28-day window
Rationale:  observational, rounded down from four weeks of data
Owner:      product manager, with the on-call engineers
Review:     monthly until it settles, then quarterly
```

The window matters as much as the percentage. The Workbook prefers a rolling window to a calendar month, because users do not forget an outage on the last day of a month when the next one begins, and a whole number of weeks, so that every window holds the same number of weekends; four weeks is its general-purpose choice. That window is 28 × 24 × 60 = 40,320 minutes, and 0.1% of it is a little over 40 minutes of complete outage. On a 30-day month the same 99.9% allows 43.2 minutes, the figure in the book’s availability table.

Time is the cruder way to count, though. The SRE book explains that Google measures availability as the success rate of requests rather than as uptime, because a service spread across the world is seldom entirely up or entirely down. Counted in requests, the Workbook’s example runs: at 99.9%, 3 million requests over four weeks leave room for 3,000 failures, and an outage that causes 1,500 of them uses half the budget. Counting requests also prices the partial failures that uptime misses. At steady traffic, 20 minutes in which half of all requests fail costs as much as 10 minutes of total outage, which is roughly a quarter of the 28-day budget.

In Prometheus, with a request counter labelled by status, the indicator over the window is one ratio, modelled on the Workbook's own example:

```
sum(rate(http_requests_total{job="checkout", status!~"5.."}[28d]))
  /
sum(rate(http_requests_total{job="checkout"}[28d]))
```

On this site, the service levels page sits on the agreement side of the line. For a system the company runs on a client's behalf, the response time for an incident, the hours covered, the availability it is held to and the maintenance windows are written into the agreement for that system, and they are set per system because a nightly report and a payment service need different answers. The first-party error monitoring and tracing built by the founder as part of current work under NextMariner surfaces through a set of operator views, one of them for service level objectives. And the note on selling the last ticket exactly once turns a stock invariant into one: check it on a schedule, chart it and page on any drift, so that zero oversells is an objective with an alert behind it rather than a hope.

Where it goes wrong

The target is copied from last month's dashboard. Taking whatever the system achieves today feels safe, but the SRE book warns that a target adopted without reflection can lock a team into heroic efforts to keep meeting it, on a system that cannot improve without a redesign. Work back from what users need, and write the reasoning down beside the number: the Workbook asks for the rationale, and for whether the figures came from experiment or observation, even when they are frankly ad hoc. Review the document monthly at first and quarterly once it settles.

The objective is 100%, or a string of nines nobody has priced. A system that is always available, or scales without limit, is expensive to build and run, and better than users would notice. Start with a loose target and tighten it as you learn the system; the book prefers that to a strict target that later has to be relaxed because it proved out of reach.

An average stands in for what users feel. In the book's example a typical request takes about 50 ms while 5% of requests take twenty times as long, and a graph of the average shows nothing happening all day. Write latency objectives

as percentiles, or as a share of requests under a threshold such as 300 ms, never as a mean.

There are more objectives than anyone can defend. A dozen SLOs per service dilutes all of them. The book's test is blunt: if quoting an objective can never win an argument about priorities, it is probably not worth having. Keep the few that cover what users actually do.

Missing the objective changes nothing. The Workbook calls this an SLO without teeth: compliance becomes one more KPI on a report instead of a reason to decide anything. Agree in advance what happens when the budget is spent, and have the product manager, the developers and the SREs sign it. The Workbook's example error budget policy pauses ordinary changes while the service is outside its objective, allowing only the highest-priority bug fixes and security work, and says in so many words that the policy is not a punishment.

The SLO is the same number as the SLA. When the internal objective matches the contract, the first a team hears of trouble is the breach. The book recommends a tighter internal objective than the one advertised, as a buffer for chronic problems. The opposite mistake matters too: a service that runs far better than its objective teaches users to depend on that, so the book suggests occasional planned downtime, throttling, or a system that is no faster under light load.

The window is longer than the data. A 28-day objective computed on a Prometheus server that keeps 15 days of samples, the default when no retention is set, quietly measures 15 days and reports it as four weeks. Set the retention past the window before trusting the number:

```
prometheus --storage.tsdb.retention.time=35d
```

Further reading

- Service Level Objectives — chapter 4 of Google's Site Reliability Engineering book: indicators, objectives and agreements, and how to choose a target.
- Implementing SLOs — the SRE Workbook's chapter on SLIs as ratios, error budgets, windows and writing the objective down.

- Example Error Budget Policy — a complete policy for a fictional game service: when releases stop, and who settles a disagreement.
- Embracing Risk — why Google measures availability by request success rather than uptime, and what each step of reliability costs.
- Availability Table — how much downtime each level of availability allows per year, quarter, month, week, day and hour.
- Storage — how Prometheus keeps samples on disk, and the retention flags that decide how far back an SLO query can see.

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 a service level objective?” 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/service-level-objective/