Engineer

systemd

the thing that starts everything else on Linux

What it is

On nearly every mainstream Linux distribution, process ID 1 is systemd: the system and service manager that brings the machine up at boot, starts and watches its services while it runs, and takes everything down in order at shutdown. The project spells its name in lower case, never SystemD or System D, and the trailing d is the old Unix mark of a daemon. It belongs to the family of init systems, alongside the SysV init scripts and Upstart it replaced on most distributions, OpenRC and runit, which some smaller distributions still choose, and launchd, which does the same job on macOS.

Everything systemd manages is a unit, described in a short text file of sections and key-value pairs. Its manual counts 11 types: service, socket, target, device, mount, automount, timer, swap, path, slice and scope. A service runs a program. A timer starts another unit on a schedule. A socket holds a listening port and starts its service when the first connection arrives. A target groups units into a state the machine can reach, such as a multi-user system with networking. Requirements and ordering are separate settings, so “this needs that” and “this starts after that” are two different lines, and systemd turns them into one transaction of start and stop jobs that it can check before it runs.

Two design choices set it apart from the shell-script init systems before it. Each service runs in its own Linux control group, so the manager tracks the processes in that group and can stop them together, limit their memory and account for their CPU. And the project is a suite as well as a manager: its own site lists a logging daemon, the journal, along with components for logins, network configuration, time synchronisation and name resolution. Distributions choose which of those extras they enable; the manager at PID 1 is the part that is close to universal.

Use the concept in a decision

Put “What is systemd?” to work in a real decision. We can help relate the concept to the system you are building or buying. Day-to-day care of servers, mail, accounts and updates, so the things everybody relies on stay boring.

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. Ran 10 PostgreSQL and MS SQL servers on Ubuntu — 10 database servers deployed and kept healthy

Ran 10 PostgreSQL and MS SQL servers on Ubuntu Get in touch →

Why it matters

On a Linux server the question is rarely whether to run systemd, because the distribution already chose it. The real decision is whether to use it as the supervisor and scheduler, or to stack another process manager, a container daemon’s restart policy or cron on top of it.

Using it directly puts start order, restart policy, logs, resource limits and sandboxing in one file per service, read by the manager that is already running, without a second supervisor process. A unit can cap memory with `MemoryMax=`, make the operating system’s own directories read-only for its service with `ProtectSystem=`, and run under a throwaway user with `DynamicUser=`, and `systemd-analyze security` scores how exposed each service is on a scale from 0.0 to 10.0. A job started by a timer, unlike a crontab line, logs to the journal, takes the same sandboxing as any service, and shows its last and next run in `systemctl list-timers`. The skill also transfers: an engineer who can read a unit file can operate most Linux servers, whichever distribution they run.

The costs are real too. Units are Linux-only, so a team that also ships to macOS writes launchd property lists, and one that ships to the BSDs writes rc scripts. Inside most containers there is no systemd at all: the application itself is PID 1, and a restart policy belongs to whatever runs the container. Where containers and plain services share a host, Podman’s Quadlet narrows that gap by turning a container description into an ordinary service unit, so both kinds of workload sit under one supervisor.

What it looks like in practice

The company’s own platform leans on systemd in three places.

In the September 2026 implementation record, the production host described in running the company's public services on one 512 MB host was a single-core machine with about 464 MB of usable memory. Resident on it were a web server holding TLS for seven domains, the git forge, the onion service, one server each for Gemini and Gopher, and intrusion banning. Backup automation is implemented separately; that does not establish active encrypted backups. At that size every resident process had to justify its memory, so systemd was the supervisor, with no second supervisor process layered over it, and the containers ran as Quadlet units under the same manager instead of under a container daemon of their own. When the question was whether the host could also carry an onion service, a recorded day of measured samples showed available memory never falling below about 310 MB of 464.

The backup and restore-test implementation uses systemd units for a database dump where configured, an encrypted deduplicated restic snapshot to storage at another provider over SFTP, retention pruning and an integrity check. A separate monthly workflow restores a known file and compares its bytes, and backup and restore have separate success signals. The timers depend on a configured repository; the implementation record dated 2026-10-27 says they were disabled without one and records no completed live restore. The work also met one of systemd's sharper edges, described below: the passphrase lives in a protected file the units read, because a value passed through the unit's environment has its escapes processed, and a backslash could silently change it.

The tech stack draws the general line: plain systemd units by default, with Docker and Kubernetes kept for the workloads that earn them.

Where it goes wrong

An edited unit keeps running with its old settings. systemd reads unit files into memory, so a change on disk does nothing until `systemctl daemon-reload` rebuilds the manager's view, and `systemctl cat` can show a file that differs from what the running manager uses. Make changes with `systemctl edit`, which reloads for you, or reload and then restart the service.

A secret reaches the service with a character missing. Values in `Environment=` go through systemd's own quoting rules, where C-style escapes are processed and a literal backslash has to be doubled. systemd's own documentation also

warns that environment variables are visible to unprivileged clients over D-Bus. Put secrets in a file and pass it with `LoadCredential=`, which hands the service a read-only copy that only its user and root can read.

A package upgrade quietly undoes a local fix. Edits made to the vendor's file under `/usr/lib/systemd/system/` are overwritten by the next package update. Put changes in a drop-in under `/etc/systemd/system/<unit>.d/`, which is merged on top and survives upgrades. Command lists add up rather than replace, so a drop-in that changes the start command first clears it with an empty assignment:

```
# /etc/systemd/system/app.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/local/bin/app --config /etc/app/app.toml
Restart=on-failure
```

A nightly job skips the night the server was off. A calendar timer that was due while the machine was down simply waits for its next slot, because `Persistent=` defaults to false. Set `Persistent=true` on backups and other jobs that must run, and the missed run fires when the timer is active again, subject to any configured random delay.

A crashed service stays down. `Restart=` defaults to `no`, so a service that dies is left dead until someone notices. Set `Restart=on-failure` for long-running services, and know that once a unit hits its start limit, the manager gives up on it, and a manual start has to wait for the interval to pass or for `systemctl reset-failed`.

Further reading

- systemd(1) — the manager itself: the unit types, the boot sequence and the signals it answers.
- systemd.unit(5) — unit files, dependencies, start limits and the drop-in directories that override them.
- systemd.service(5) — service types, restart policy and how command lists reset.
- systemd.exec(5) — the execution environment, sandboxing options and credentials.
- systemd.timer(5) — calendar and monotonic timers, including `Persistent=`.

- systemctl(1) — the command line for starting, editing, reloading and inspecting units.

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 systemd?” 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/systemd/