Engineer

Podman

a container engine that needs no daemon

What it is

Podman is a container engine for Linux that builds, pulls and runs the same OCI images Docker does, but with no central server in the middle. Each `podman` command does its work in its own process and exits. A started container keeps running under a small monitor process called conmon, and that monitor is a child of whatever launched it: a shell, a script, or systemd. The project describes itself as daemonless, open source and Linux-native, and says most users can alias `docker` to `podman` and carry on.

The name is short for pod manager. A pod, borrowed from Kubernetes, is a group of containers that share a network namespace and so reach each other on localhost. Podman can create pods by hand, and `podman kube play` starts containers and pods from a Kubernetes YAML file, which makes it a reasonable place to try a manifest before it goes near a cluster.

Two features set it apart in daily use. The first is rootless operation. An ordinary user can run containers, with the kernel's user namespaces mapping that user to UID 0 inside the container and the user's subordinate ID range, from `/etc/subuid` and `/etc/subgid`, to every other UID. Networking for rootless containers goes through pasta, a user-space network stack, since an unprivileged user has no right to rewrite the host's packet filter.

The second is Quadlet. A Quadlet file is a short declarative description of a container, volume, network, pod or image build, written in the same ini syntax as a systemd unit and saved under `/etc/containers/systemd/` for the system or `~/.config/containers/systemd/` for one user. At boot, and on every `systemctl daemon-reload`, a systemd generator turns `web.container` into an ordinary `web.service`. The older `podman generate systemd` command still works, and the project marks it deprecated in favour of Quadlet.

Use the concept in a decision

Put “What is Podman?” to work in a real decision. We can help relate the concept to the system you are building or buying. Packaged to behave identically on a laptop, on a test server and in production, and to scale up when a busy week arrives.

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 20 Docker apps on Google Cloud and AWS — 20 containerised applications deployed and watched

Ran 20 Docker apps on Google Cloud and AWS Get in touch →

Why it matters

The question Podman puts on the table is who owns the process tree. Docker manages containers through `dockerd`, with its own restart policies, logging and container health checks. On a server that already runs systemd, that is a second supervisor beside the first, and the two do not share a vocabulary.

Under Podman with Quadlet, systemd is the only supervisor. A container is a service like any other: `systemctl status` shows it, journald holds its output, `After=` and `Requires=` order it against the database it needs, and `Restart=` and the cgroup limits systemd already understands apply to it unchanged. An operator who knows how to read a unit file already knows how to read the container plane.

There is a security angle too. The Docker daemon runs as root unless its rootless mode is chosen, and Docker’s own security guide says only trusted users should be allowed to control it, because a container can be handed any directory on the host. Podman keeps no privileged daemon running between commands. A rootless setup narrows things further: a process that escapes its container lands as an unprivileged user.

Docker vs Podman is therefore less a contest of features than a question of fit. Docker is the smoother choice on a developer laptop with Docker Desktop, for teams whose workflow is built around Compose, and for tools that expect a Docker socket, though Podman can serve a compatible API on request. Podman tends to win on servers where systemd is already the source of truth, and where a daemon that rewrites the firewall is a liability rather than a convenience.

What it looks like in practice

Podman carries the container plane in two recent projects, and before that it was the debugging tool for a fleet built on Docker.

On one host, containers run as Quadlet units under systemd, with Podman in place of Docker. The reason is the firewall: Docker's default publishes container ports above the rules the hardening role installs, while these containers use the host's network with no published ports at all. The units run at the system level rather than rootless, and the page records that as a trade, simpler automation against the stricter isolation rootless would give, along with a note on which way to move if container escape ever matters more. Deployment is split from provisioning. Root sets up the deployment path once. Every later release is one fixed script that a user without privileges is allowed to run through a scoped rule, and the script does two things only: build the image and restart the service.

The same arrangement holds up on very little hardware. A single-core host with 512 MB of memory runs the company's public TLS endpoints, git forge and onion service, with backup automation implemented separately. Its container plane is Quadlet units under the same supervisor as everything else rather than a daemon with its own. Over a recorded day of samples, the available memory never fell below about 310 MB of the 464 MB usable.

Earlier, across twenty Docker-packaged applications spread over Google Cloud and AWS, the troubleshooting went through Podman, with Kubernetes used for the orchestration side. The images were the same; the tool around them was a choice.

Where it goes wrong

Most Podman trouble comes from expecting it to behave like Docker, or like a plain systemd unit, when it is a little of both.

An edited Quadlet file changes nothing. The generator runs at boot and on `systemctl daemon-reload`, so a change on disk is invisible until one of those happens. Then `systemctl enable` refuses the generated service, because systemd treats it as transient. Boot-time start belongs in the file's own `[Install]` section:

```
# /etc/containers/systemd/web.container
[Unit]
Description=Web front end

[Container]
Image=docker.io/library/nginx:1.27
Network=host

[Service]
Restart=always

[Install]
WantedBy=multi-user.target
```

After saving it, run `systemctl daemon-reload` and `systemctl start web.service`.

A rootless service stops when its owner logs out. User units live inside the user's own systemd manager, which ends with the last session. Running `loginctl enable-linger` for that user keeps the manager alive from boot, which any rootless service meant to survive a reboot needs.

A rootless web server asks for port 80 and is refused. Ports below 1024 require root or the `CAP_NET_BIND_SERVICE` capability, and that rule applies to the user Podman runs as. Publish a high port and put a proxy in front, or lower the kernel's `ip_unprivileged_port_start` setting on purpose, knowing it applies to every user on the host.

Files in a bind mount belong to a stranger. In rootless mode the user's own UID becomes root inside the container, and every other container UID maps into the subordinate range. A process running as UID 999 inside writes files the host sees as owned by something like 100998. `podman unshare` opens a shell inside the same mapping, where `chown` behaves as the container expects, and the `U` option on a mount changes ownership to match the container automatically.

Further reading

- What is Podman? — the project's own documentation, starting from its definition of the tool.
- podman-systemd.unit — the Quadlet reference: file types, search paths and every key.

- Basic Setup and Use of Podman in a Rootless environment — subordinate IDs, user namespaces and pasta networking.
- podman-unshare — entering the rootless user namespace to inspect or fix ownership.
- podman-generate-systemd — the older way to make units, and its deprecation in favour of Quadlet.

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 Podman?” 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/podman/