

Installing Podman 5.x and Migrating from Docker in Linux

Podman is a container tool in Linux, which is increasingly gaining popularity. Podman is secure, simple to use and its integration with systemd makes it popular amongst developers and system administrators. Podman 5.x (new release) is OCI-conformant and its interface is similar to Docker.

In this article, you will learn how to install Podman 5.x on a Linux system and migrate your Docker workloads.

Why should you use Podman ?

Podman is a container engine that follows the **OCI (Open Container Initiative)** specification for containers and images. It is a container engine that builds, runs, and manages containers. This means the container images and runtime formats it uses are fully compatible with other OCI-compliant tools like Docker and containerd. Because of this standardization, containers created with Podman can be shared, moved, or deployed across different platforms without modification.

It is compatible with Docker command line commands. Podman supports many of the same command-line arguments and workflows used in Docker, making it easy for Docker users to switch without relearning commands. This compatibility greatly reduces migration effort and helps developers maintain familiar workflows.

The difference lies in how Podman runs containers. The key distinction is in Podman's architecture and execution model. Unlike Docker, Podman runs containers **without a central daemon**, relying instead on a daemonless and rootless design. This approach improves security, gives each container its own supervised process, and avoids the single point of failure that comes with Docker's daemon.

Instead of running persistent daemons, it spawns containers as child processes. Podman is easier to debug, easier to secure, and simpler to integrate with standard Linux tools.

With Docker you often need elevated permissions to run containers or modify networking settings. Podman avoids that by letting you manage containers as an unprivileged user. That reduces the risk of accidental system wide changes and helps meet security requirements in restricted environments.

Podman integrates seamlessly with systemd. You can create unit files automatically, control container lifecycles with standard service commands, and rely on system boot ordering. This makes Podman especially appealing on servers where administrators already use `systemd` to control everything from databases to custom applications.

With Podman 5.x, performance has improved in areas like build speed, network setup, and layer caching. The project also sharpened Docker compatibility and refined tooling around volumes and pods. All of this makes migration easier than ever.

If your environment requires the very latest Podman, the upstream RPM repository from the Podman project is an option. Administrators in regulated environments should check with internal policy before enabling external package sources.

Installing Podman 5.x on Various Linux Distributions

Commands to install Podman 5.x :

- **For CentOS Stream and RHEL 9**

For CentOS Stream 9 or RHEL 9, install Podman from the AppStream repository. Type:

```
sudo dnf install podman
```

- **For Fedora, type:**

```
sudo dnf install podman
```

- **Ubuntu 22.04 and 24.04**

Ubuntu repositories lag behind upstream, so you should use the official Kubic repository provided by the Podman team. Type:

```
. /etc/os-release
echo "deb
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/x${ID}_${VERSION_ID}/" | sudo tee /etc/apt/sources.list.d/podman.list

curl -fsSL
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/x${ID}_${VERSION_ID}/Release.key | \
    gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/podman.gpg > /dev/null

sudo apt update
sudo apt install podman
```

- **Debian 12**

Debian Stable also benefits from the Kubic repository.

```
. /etc/os-release
```

```
echo "deb
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/sta
ble/Debian_${VERSION_ID}/ /" | \
sudo tee /etc/apt/sources.list.d/podman.list

curl -fsSL
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/sta
ble/Debian_${VERSION_ID}/Release.key | \
gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/podman.gpg > /dev/null

sudo apt update
sudo apt install podman
```

• Arch Linux

Arch users simply install from the main repository.

```
sudo pacman -S podman
```

• openSUSE Leap and Tumbleweed

Tumbleweed ships the latest Podman.

```
sudo zypper install podman
```

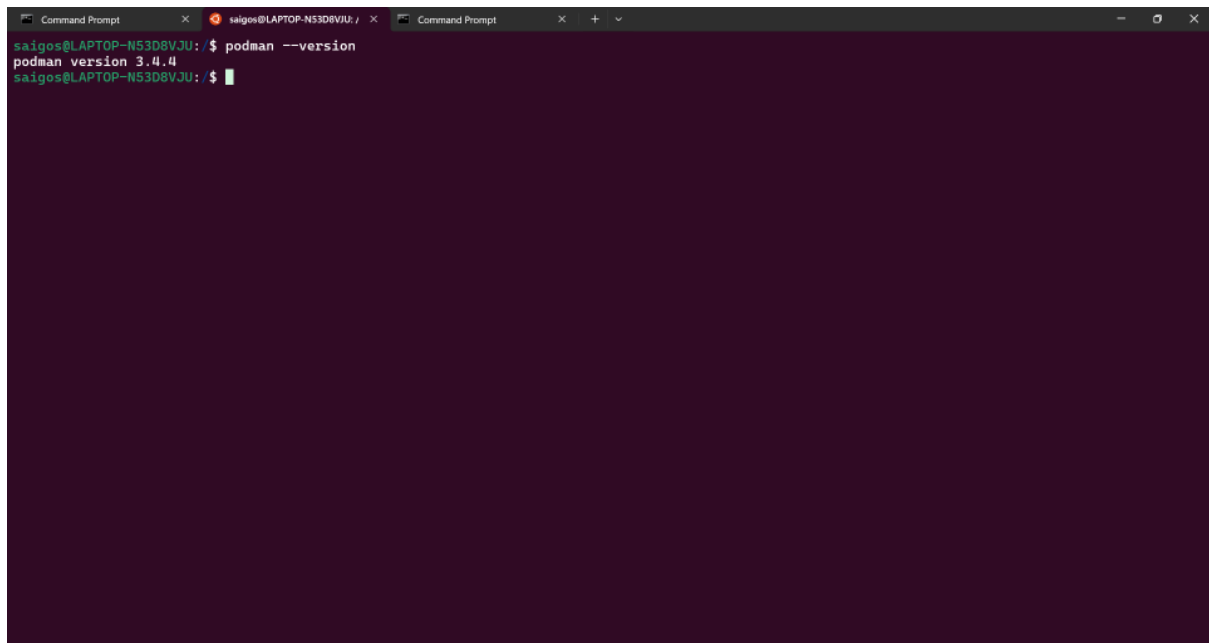
Leap users may need to enable the containers module depending on their version.

Checking if installation is complete

After installation, check if `podman` is installed with the following command:

```
podman --version
```

If you see ‘podman’ and the version, it means podman is successfully installed.

A screenshot of a terminal window with a dark purple background. The window has three tabs: 'Command Prompt', 'saigos@LAPTOP-N53D8VJU: /', and another 'Command Prompt'. The terminal text shows the user 'saigos@LAPTOP-N53D8VJU' running the command 'podman --version', which outputs 'podman version 3.4.4'. The prompt returns to 'saigos@LAPTOP-N53D8VJU: \$'.

Configuring Podman for ‘Rootless’ Use

Once installed, Podman is ready for rootless containers. What is ‘**rootless**’ ? Normally, container engines (like Docker) run a **daemon as root**. This means the container runtime has full system privileges, which can be risky if compromised. A **rootless container** runs entirely under a **non-root user account**. It uses **Linux user namespaces** and **subuid/subgid mappings** to give the container its own “fake root” inside, while mapping to unprivileged IDs outside. Inside the container, processes may appear to run as `root` (UID 0), but on the host they’re actually mapped to a non-root UID (like 100000)To check your environment, type:

```
podman info --debug
```

```
saigos@LAPTOP-N53D8VJU: / $ podman info --debug
host:
  arch: amd64
  buildahVersion: 1.23.1
  cgroupControllers:
  - memory
  - pids
  cgroupManager: systemd
  cgroupVersion: v2
  common:
    package: 'common: /usr/bin/common'
    path: /usr/bin/common
    version: 'common version 2.0.25, commit: unknown'
  cpus: 4
  distribution:
    codename: jammy
    distribution: ubuntu
    version: "22.04"
  eventlogger: journald
  hostname: LAPTOP-N53D8VJU
  idMappings:
    gidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
    - container_id: 65537
      host_id: 200000
      size: 65536
    uidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
```

Look under “host” to confirm that user namespaces and subuids/subgids are configured.

What Are subuid and subgid?

- subuid (**subordinate user IDs**) and subgid (**subordinate group IDs**) are ranges of **additional UIDs/GIDs** assigned to a user.
- They’re defined in `/etc/subuid` and `/etc/subgid`.
- These ranges allow a non-root user to **map container users** to IDs they control, without needing root privileges.

If you see errors about missing ranges or uids/guids missing, create them manually:

```
sudo usermod --add-subuids 100000-165535 $USER
sudo usermod --add-subgids 100000-165535 $USER
```

Log out and log back in. This gives Podman room to map user IDs inside containers.

To confirm rootless networking, run:

```
podman run --rm -p 8080:80 docker.io/nginx
```

Open a browser and visit <http://localhost:8080> to confirm the container is reachable. The browser shows:



Differences between Podman 5 and Docker

Podman supports a Docker compatible CLI, but there are some differences:

Pod Support

Podman has pods, which group containers that share namespaces. This is similar to Kubernetes pods. It becomes useful when you have multi container applications that need shared network stacks, shared IPC, or predictable startup ordering. Docker does not have pods.

Daemon

Docker relies on a background daemon that manages containers, networks, and storage. Podman does not. Each container is a child process you can inspect it with standard tools like `ps` and `kill`. If you rely on daemon settings in your workflow, you will need to adjust.

Networking

Podman uses the '**netavark**' and '**aardvark**' components for network stack management. In Podman, **Netavark** is the new Rust-based network setup tool, and **Aardvark** is its companion DNS server. Most port forwarding and bridge style networks behave like Docker, but Podman does not support every plugin used in Docker's custom networking ecosystem. If you use unusual network drivers, test them first.

Volumes

Docker has volumes. Podman does not have daemon-controlled volume directories.

Migrating from Docker to Podman 5.x: Step by Step

Follow the following steps to migrate from Docker to Podman 5.x:

1. Stop Docker

To prevent conflicts:

```
sudo systemctl stop docker
sudo systemctl disable docker
```

You can keep Docker installed if you want both tools available, but avoid running them at the same time because both try to manage `iptables` rules, image storage, and ports.

2. Import Existing Docker Images into Podman

Podman can use the same image formats as Docker. If you want Podman to read Docker's storage directly, you can copy or move the images. The simplest method is to export and import:

```
docker save myimage:latest | podman load
```

Note: 'myimage' is the name of your image. 'latest' refers to the tag. In my case, I used:

```
docker save mysql:8.4 | podman load
```

Repeat for each image you want to migrate.

You can also rebuild your images using `podman build`. If your Dockerfiles follow best practices, Podman should build them without changes.

3. Migrate Docker Compose

Podman does not use Docker Compose directly but supports `podman-compose`, which is a Python tool that handles Compose files.

Install:

```
sudo pip3 install podman-compose
```

From the directory, which contains the `docker-compose.yml` file run:

```
podman-compose up
```

Most Compose v2 files work as long as they avoid unsupported networks or volume drivers. If you want a smoother long-term experience, consider converting Compose workflows to systemd units or Kubernetes YAML using tools like `podman generate kube`.

4. Replace Docker Commands in Scripts

Because Podman mirrors Docker syntax, many scripts only need a simple alias.

Add this to your shell config:

```
alias docker=podman
```

This works for commands like:

```
docker run
docker build
docker push
```

If your script checks for the Docker daemon or uses `docker system` commands with daemon specific options, adjust those manually.

5. Convert Docker Managed Services to systemd

Podman can generate systemd units directly from containers.

For example:

```
podman generate systemd --name myapp --files --new
```

This creates a service file you can copy into:

```
sudo cp container-myapp.service /etc/systemd/system/
sudo systemctl enable --now container-myapp
```

This replaces Docker's daemon autostart behavior with `systemd` management. It also gives you standard logs, restarts, and dependency management.

Building and Running Containers with Podman 5.x

To reinforce that Podman works almost exactly like Docker, here are typical commands and their Podman equivalents.

Build an image

```
podman build -t myimage .
```



```
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman build -t myimage .
[1/2] STEP 1/4: FROM eclipse-temurin:17.0.5_8-jre-focal AS builder
[1/2] STEP 2/4: WORKDIR extracted
--> Using cache 7b3504e1d5bb7246c1b799ecd0b114f078d75fdca84ee3d3af0e1f3d7e11c7c7
--> 7b3504e1d5b
[1/2] STEP 3/4: ADD ./build/libs/*.jar app.jar
--> ab919bc6275
[1/2] STEP 4/4: RUN java -Djarmode=layertools -jar app.jar extract
--> a43fca1172
[2/2] STEP 1/8: FROM eclipse-temurin:17.0.5_8-jre-focal
[2/2] STEP 2/8: WORKDIR application
--> Using cache b3f843c526cd0aef39eb15847a0fe7011f17f6316ab851039cfb043058df6833
--> b3f843c526c
[2/2] STEP 3/8: COPY --from=builder extracted/dependencies/ ./
--> Using cache a57ace19bdd
a57ace19bddf346169ac93b64c2e60c216a1c142214b2b2e6636a3b7f29876be
[2/2] STEP 4/8: COPY --from=builder extracted/spring-boot-loader/ ./
--> Using cache b6e93b651932557e33d6f558eae5b13699b2d44308d0f207574bbfd5c6eeb20
--> b6e93b65193
[2/2] STEP 5/8: COPY --from=builder extracted/snapshot-dependencies/ ./
--> Using cache df0a1622e1cd2d9f0780299b39b649dd6a161531d5c8871ad8e03cadf8982a53
--> df0a1622e1c
[2/2] STEP 6/8: COPY --from=builder extracted/application/ ./
--> 66b26224595
[2/2] STEP 7/8: EXPOSE 8080
--> 25aeb88774e
[2/2] STEP 8/8: ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
[2/2] COMMIT myimage
--> a7357390a49
Successfully tagged localhost/myimage:latest
a7357390a495e508f11b48ce921f1562252a3d81b627048f77f8494141b59f9f0
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$
```

Run a container

podman run -d -p 8080:80 myimage

```
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman run -d -p 8080:80 myimage
a4639bcc727ab2ed700838d3ab1d8c0f7c7b17bb83a7e7bb4630a9861fdefece
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$
```

View running containers

podman ps

```
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                    NAMES
a4639bcc727a   localhost/myimage:latest            "java -jar product-... 39 seconds ago Up 39 seconds ago 0.0.0.0:8080->80/tcp    romantic_mclean
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$
```

Stop a container

podman stop <id>

```
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                    NAMES
a4639bcc727a   localhost/myimage:latest            "java -jar product-... 39 seconds ago Up 39 seconds ago 0.0.0.0:8080->80/tcp    romantic_mclean
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman stop a4639bcc727a
a4639bcc727a
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$
```

Remove a container

podman rm <id>

```
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                    NAMES
a4639bcc727a   localhost/myimage:latest           "..."                 39 seconds ago Up 39 seconds ago 0.0.0.0:8080->80/tcp    romantic_mclean
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman stop a4639bcc727a
a4639bcc727a
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$ podman rm a4639bcc727a
a4639bcc727ab2ed700838d3ab1d8c0f7c7b17bb83a7e7bb4630a9861fdefece
saigos@LAPTOP-N53D8VJU: /mnt/c/Users/Saikat/Microservices-with-Spring-Boot-and-Spring-Cloud-Third-Edition/Chapter04/microservices/product-service
$
```

Login to registry

If your registry requires authentication:

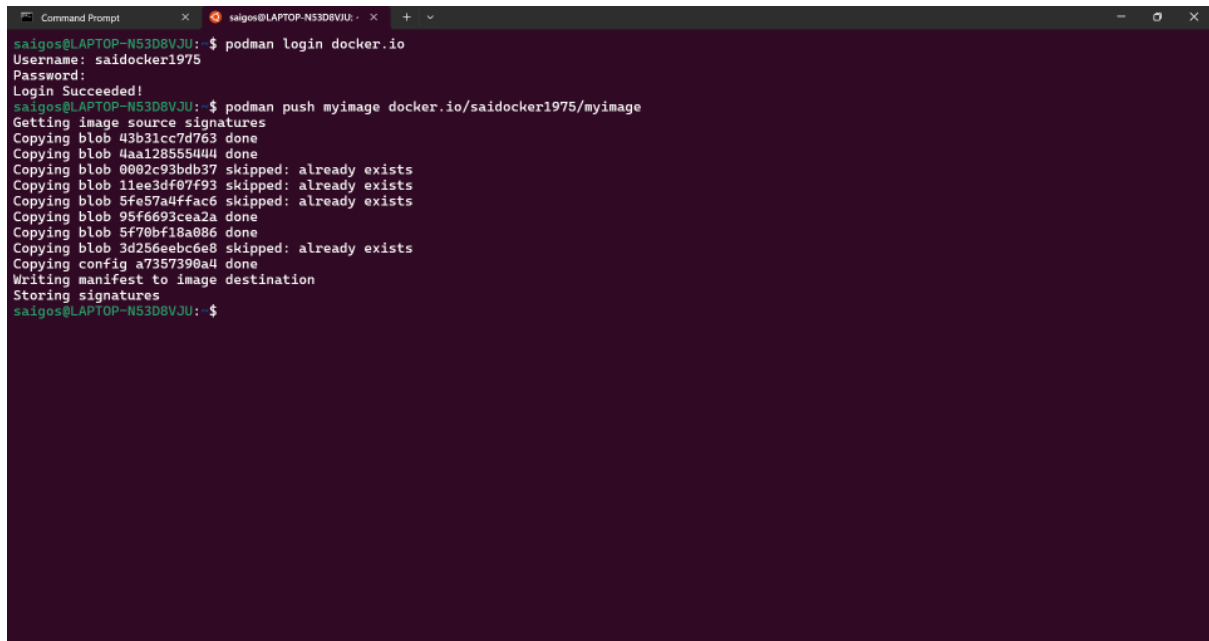
```
podman login docker.io
```

```
saigos@LAPTOP-N53D8VJU: $ podman login docker.io
Username: saidocker1975
Password:
Login Succeeded!
saigos@LAPTOP-N53D8VJU: $
```

Push to a registry

```
podman push myimage docker.io/username/myimage
```

Here `username` is the username for Docker.

A terminal window titled 'Command Prompt' with a tab for 'saigos@LAPTOP-N53D8VJU'. The terminal shows the following commands and output:

```
saigos@LAPTOP-N53D8VJU:~$ podman login docker.io
Username: saidocker1975
Password:
Login Succeeded!
saigos@LAPTOP-N53D8VJU:~$ podman push myimage docker.io/saidocker1975/myimage
Getting image source signatures
Copying blob 43b31cc7d763 done
Copying blob 4aa128555444 done
Copying blob 0002c93bdb37 skipped: already exists
Copying blob 11ee3df07f93 skipped: already exists
Copying blob 5fe57a4ffac6 skipped: already exists
Copying blob 95f6693cea2a done
Copying blob 5f70bf18a086 done
Copying blob 3d256eebc6e8 skipped: already exists
Copying config a7357390a4 done
Writing manifest to image destination
Storing signatures
saigos@LAPTOP-N53D8VJU:~$
```

Final Thoughts

Podman 5.x brings a stable, fast, and secure container engine that fits naturally into the Linux ecosystem. Its compatibility with Docker commands makes the learning curve short, and the migration path is straightforward for both developers and administrators. Once installed, Podman handles everything from simple container runs to multi-container pods, automatic `systemd` integration, and Kubernetes generation.

For most users, the transition takes less effort than expected. A few adjustments in networking, storage paths, and build scripts are usually enough to complete the switch. After that, you get a cleaner and safer environment with fewer moving parts and tighter integration with native Linux tools.

Teams often notice faster startup times and more predictable behavior during builds and deployments. Logs become simpler to trace, rootless workflows take some pressure off security teams, and the overall footprint feels lighter. As environments grow, Podman's pod model helps organize services without extra layers or plugins. You can script actions with familiar commands, automate routine tasks with `systemd` units, and keep everything consistent across development, staging, and production. The end result is a container stack that feels stable, transparent, and aligned with how Linux already works, which helps projects stay maintainable as they scale. This smoother alignment also reduces troubleshooting time and helps teams focus on building features instead of fighting tooling.