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 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:/sta
ble/x${ID}_${VERSION_ID}/ /" | sudo tee /etc/apt/sources.list.d/podman.list

curl -fsSL
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/sta
ble/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
```

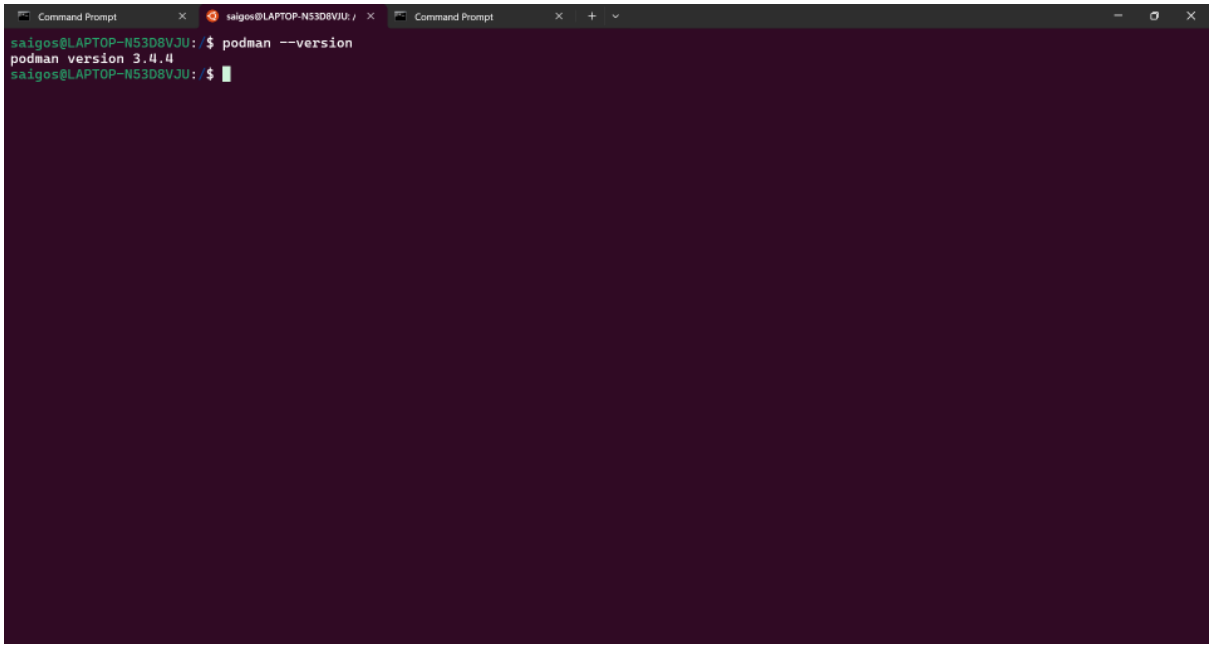
Leap users may need to enable the containers module depending on their version.

Checking if installation is complete

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

```
podman --version
```

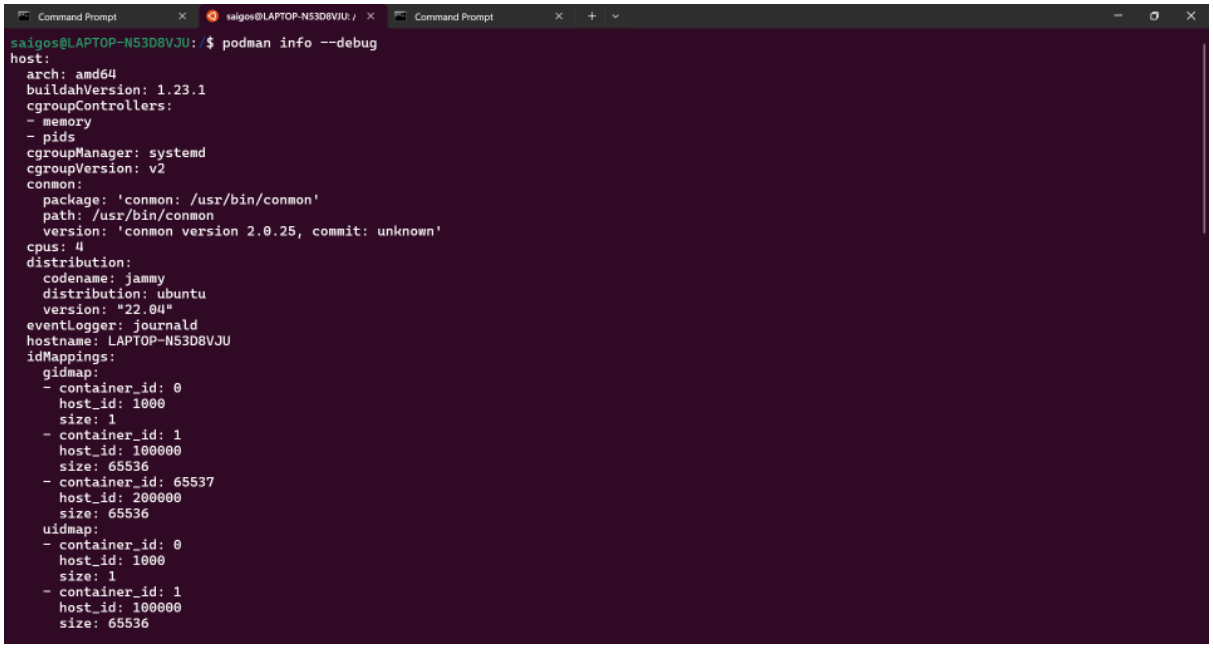
If you see ‘podman’ and the version, it means podman is successfully installed.



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
```



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
```

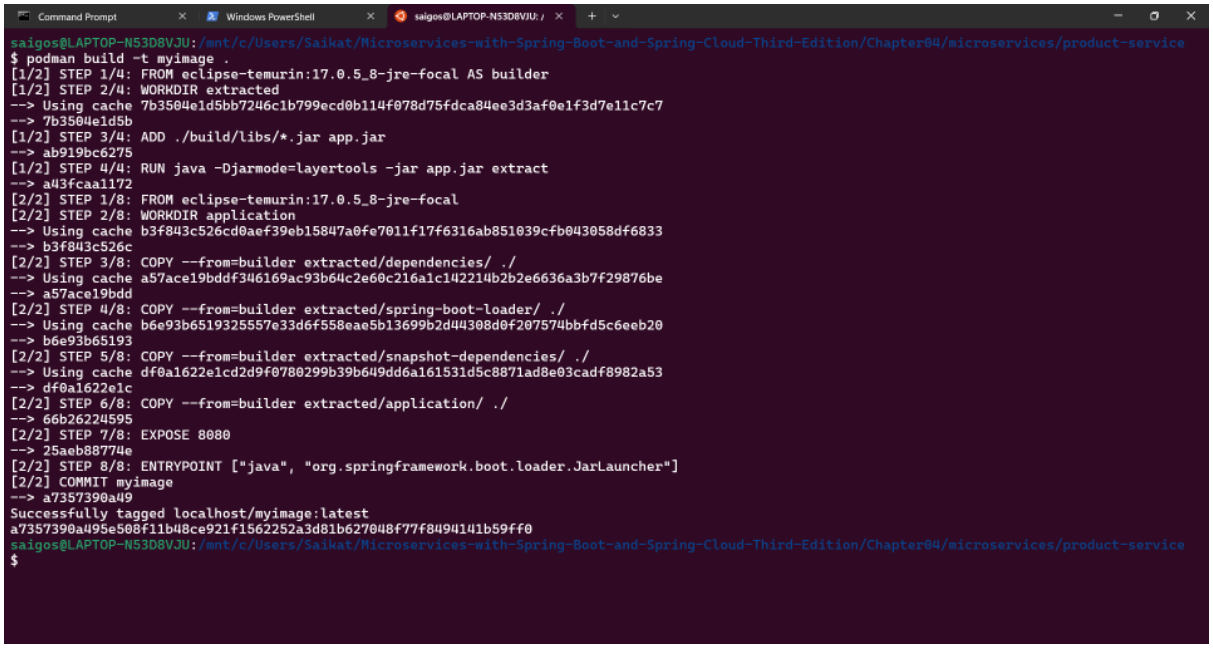
This replaces Docker's daemon autostart behavior with `systemd` management. It also gives you standard logs, restarts, and dependency management.

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.

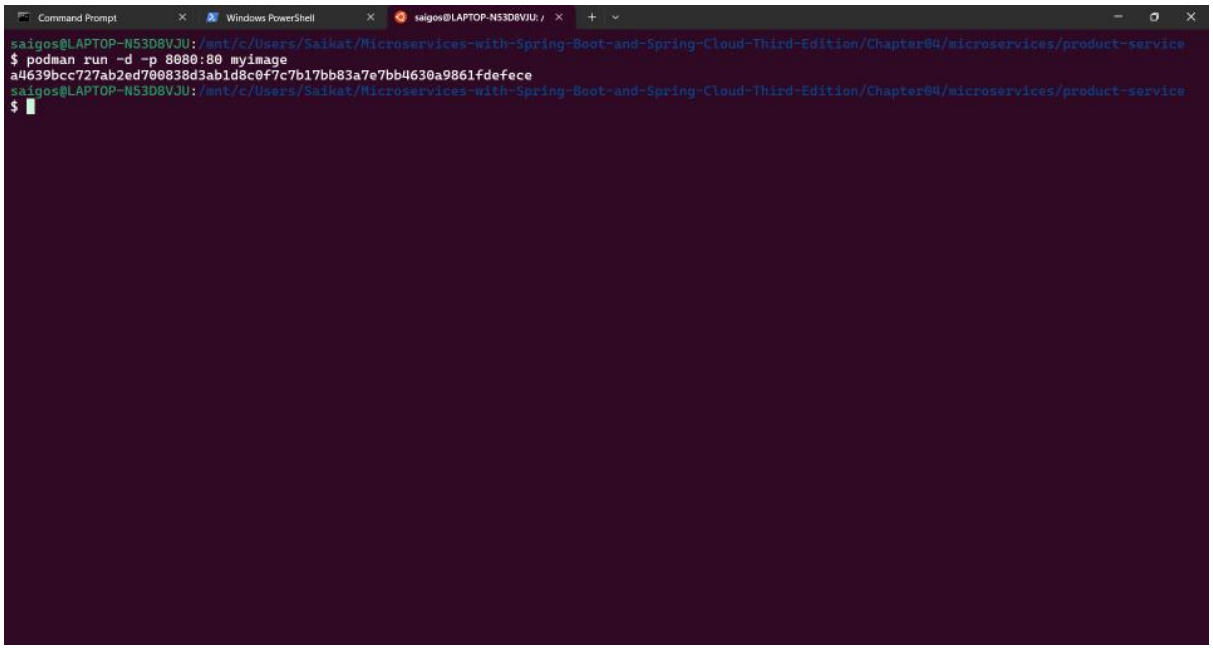
Build an image

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



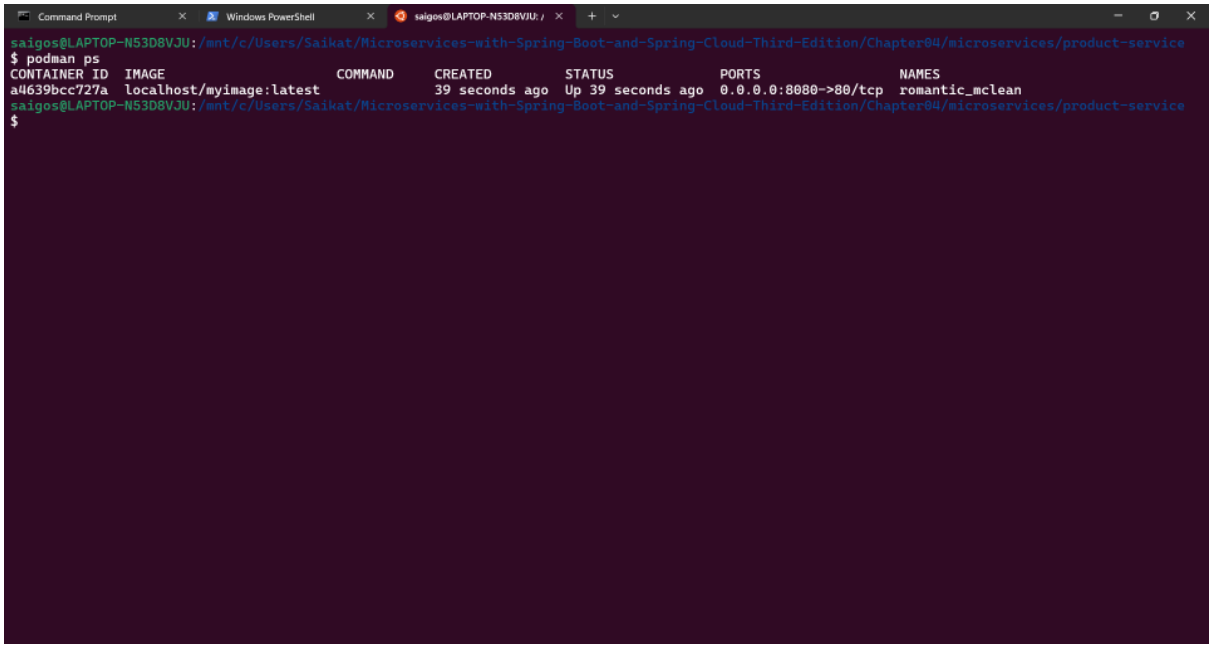
Run a container

`podman run -d -p 8080:80 myimage`



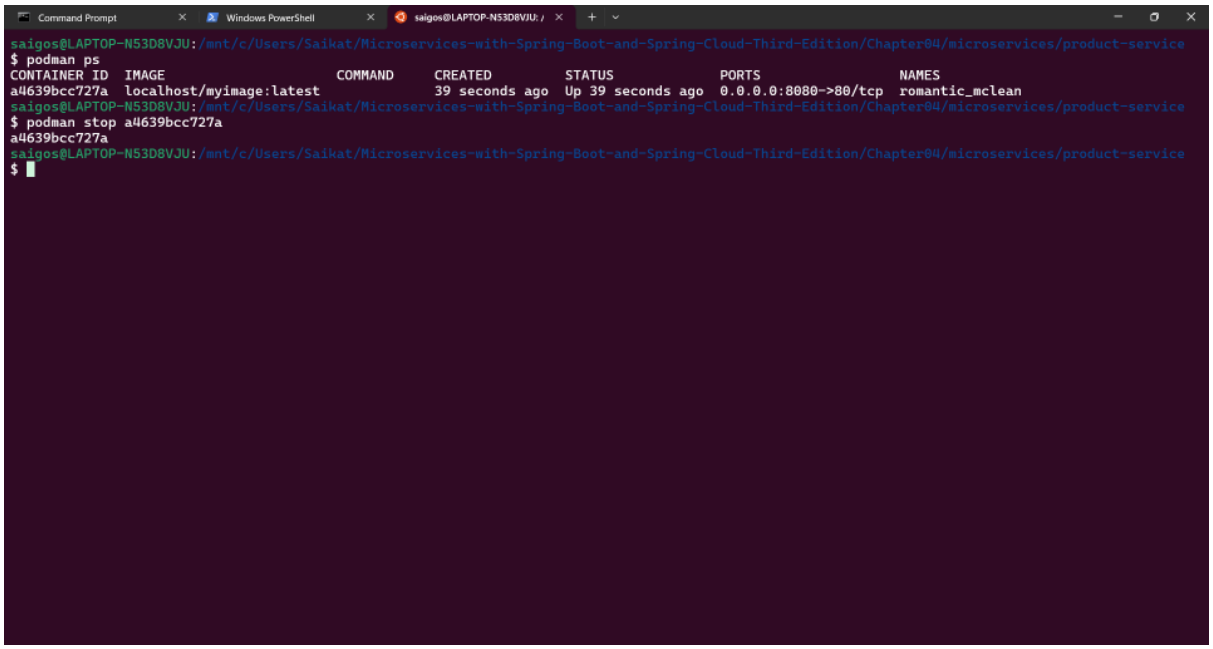
View running containers

`podman ps`



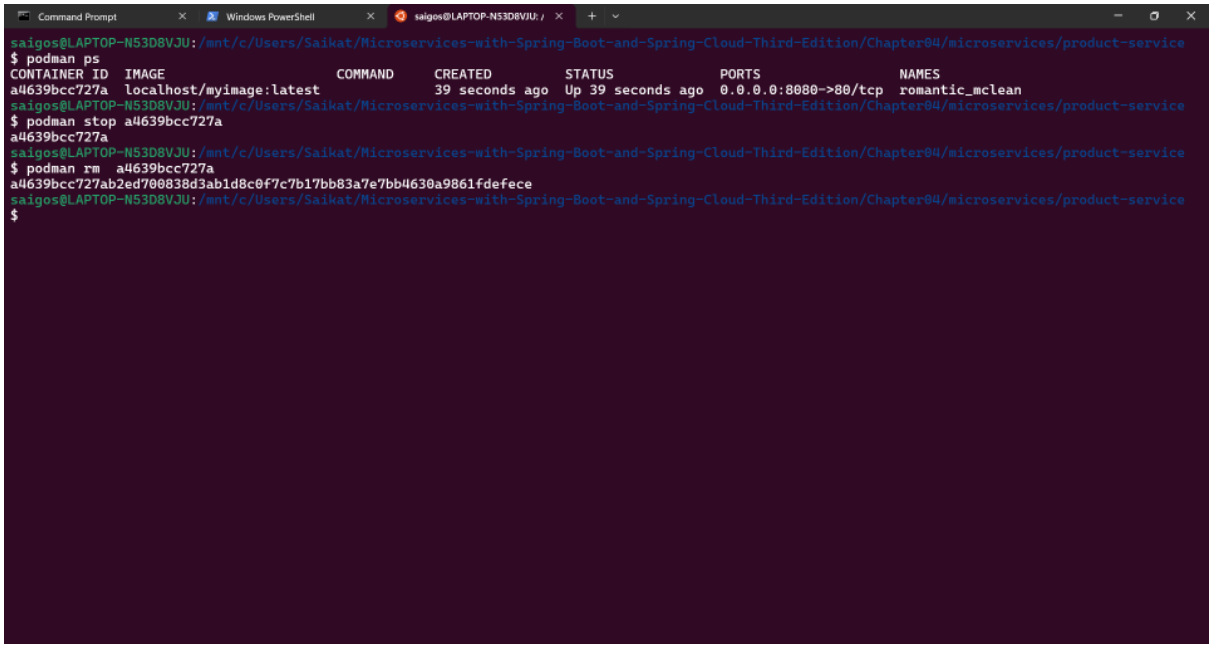
Stop a container

`podman stop <id>`



Remove a container

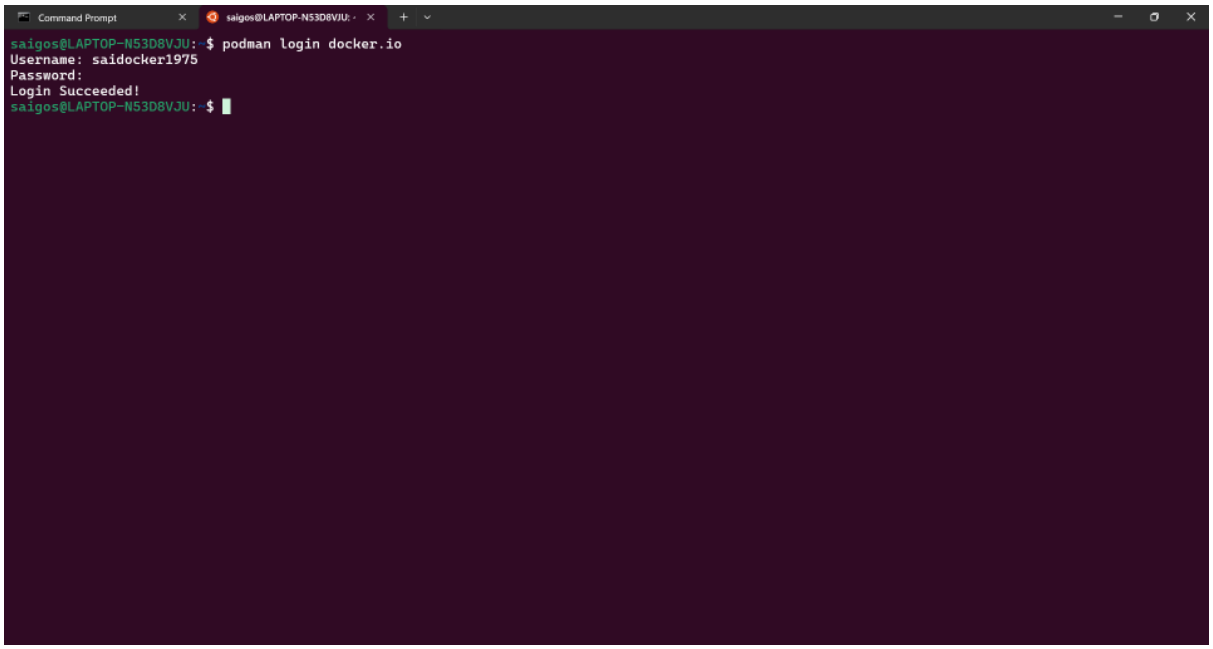
`podman rm <id>`



Login to registry

If your registry requires authentication:

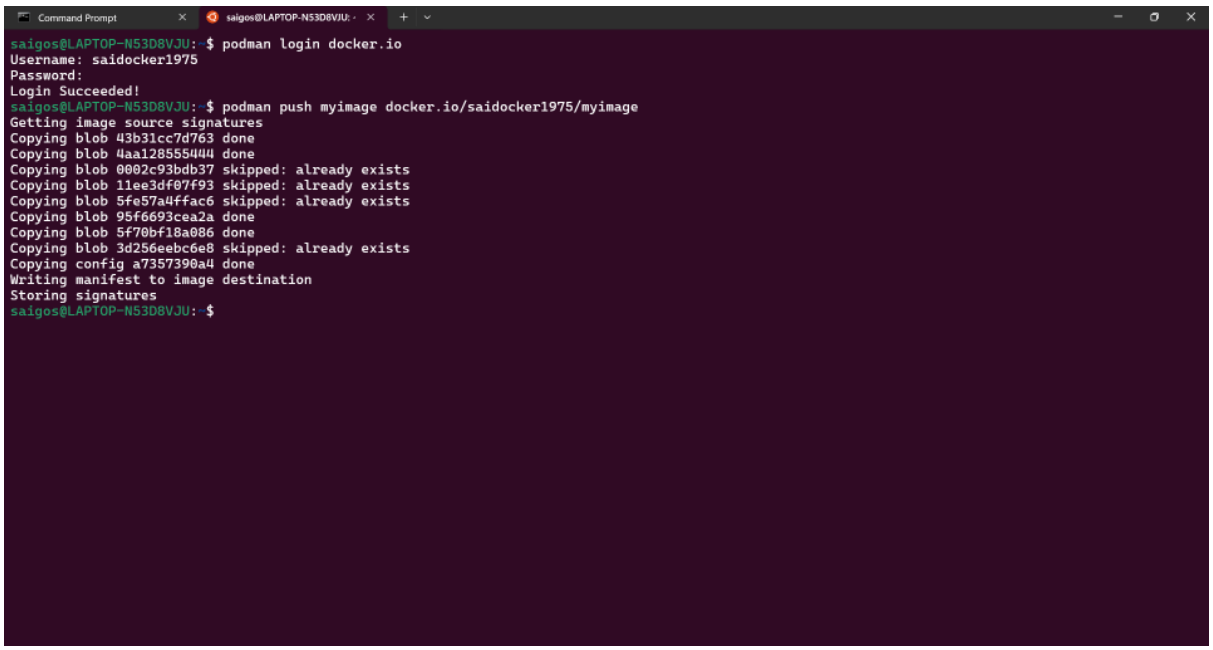
```
podman login docker.io
```



Push to a registry

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

Here `username` is the username for Docker.



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.