

Containers in HPC – Podman

UManitoba Spring 2025
High-Performance Computing And Cloud
Workshop

Stefano Ansaloni

University of Manitoba

May 23, 2025



**University
of Manitoba**



**Digital Research
Alliance** of Canada

Stefano Ansaloni

Cloud Computing Specialist at University of Manitoba
(part of the Advanced Research Computing team)

Software Developer and DevOps Specialist since 2017

Linux User/Admin since 2005

What is a container

From Wikipedia ([Containerization](#)):

Containerization is operating-system–level virtualization or application-level virtualization so that softwares can run in isolated user spaces called “containers” in any cloud or non-cloud environment, regardless of type or vendor.

Properties of containers

- ▶ Each container is a fully functional and portable computing environment surrounding the application and keeping it independent of other environments running in parallel
- ▶ Each container simulates a different software application and runs isolated processes (including configurations, libraries, and dependencies)
- ▶ Multiple containers share a common operating system kernel (operative system)

Terminology

Image	Archive (or number of archives – i.e. “layers”) of a filesystem tree along with metadata
Containerfile	Recipe for building an image, including OS and software within the image (e.g. <i>Dockerfile</i>)
Container	A running instance of an image (can be a computing process, or a service daemon)
Container Runtime	Lower level component responsible for reading the image and communicating with the host kernel to start containerized processes (e.g. <i>runc</i> , <i>crun</i>)
OCI (Open Container Initiative)	Open governance structure that creates open industry standards for container formats and runtimes
Registry	An online storage area for images (e.g. <i>DockerHub</i> , <i>Quay.io</i>)



Focusing On Podman

What is Podman

From Wikipedia ([Podman](#)):

Podman (pod manager) is an open source OCI-compliant container management tool from Red Hat used for handling containers, images, volumes, and pods; offering APIs for the lifecycle management of those components (the API is identical to the Docker API).

Why not Docker

Podman aims to provide a more secure and lightweight alternative to Docker:

- ▶ Daemonless ⇒ Don't rely on a process with root privileges to run containers
- ▶ Rootless containers ⇒ Run containers as regular users, without interacting with a root-owned daemon
- ▶ User namespaces ⇒ Careful use of kernel capabilities

Compatibility with Docker

For most use cases, Podman can be used as a “drop-in” replacement for Docker:

- ▶ Podman CLI syntax is almost the same as Docker's one
- ▶ Podman can use the same images as Docker
- ▶ Podman can use the same registries as Docker



Podman on HPC – Disclaimer

Podman is considered an advanced tool to be used only by experienced users when their workloads cannot be run using standard HPC programs/modules, or through Singularity/Apptainer (i.e. Podman should be the very last resort).

In any case, Podman must **NOT** be used:

- ▶ on login nodes
- ▶ to execute long-running services (e.g. daemons, databases, ...)

Podman on HPC

Podman is available as a module on national clusters and UManitoba HPC cluster (GreX).

HPC System	Command	Current version
National Clusters	<code>module load StdEnv/2023 podman</code>	4.9.5
GreX	<code>module load podman</code>	5.5.0

Podman on Grex

When on Grex, it is important to use the local version of Podman:

- ▶ local proxy cache for registries
- ▶ better default configuration
- ▶ newer version

Basic commands

Print version	<code>podman version</code>
Pull image	<code>podman pull <REGISTRY>/<NAME>:<TAG></code>
Delete image	<code>podman image rm </code>
List images	<code>podman image ls</code>
Create and start a container	<code>podman run [OPTS] [CMD [ARGS]]</code>
Execute command inside a running container	<code>podman exec [OPTS] <CNT> [CMD [ARGS]]</code>
Stop a running container	<code>podman stop <CNT></code>
Start a stopped container	<code>podman start <CNT></code>
Delete a container	<code>podman rm <CNT></code>
List containers	<code>podman ps [-a]</code>

Getting an image

From a registry

The “pull” subcommand can be used to download an image from an online registry (e.g. *DockerHub*) into a local (temporary) registry, that is available only on node where the *pull* has been performed and will be deleted when the job ends.

```
podman pull docker.io/alpine
```

Getting an image

Building your own

If it becomes necessary to build a custom image, it can be accomplished with the “build” subcommand.

By default the build image will be saved into a local (temporary) registry, that is available only on node where the *build* has been performed and will be deleted when the job ends.

To overcome this problem, the “--tag” / “-t” option can be specified using “docker-archive:” as the *tag* prefix.

```
podman build -t docker-archive:/path/to/custom.image \  
-f /path/to/Containerfile
```

Binding host directories

To let containers access data on the host (e.g. the user home directory), the “--volume” / “-v” option can be specified (multiple times) using the format “HOST_DIR:CONT_DIR”.

```
podman run -v $HOME:$HOME docker.io/alpine ls $HOME
```

When binding directories, the container will be able to read and write the content of those directories (if the user has the correct filesystem permissions).



Setting environment variables

To option “--env” / “-e” can be specified (multiple times) to set environment variables inside the container.

```
podman run -e V1="hi" -e V2="$USER" docker.io/alpine sh -c 'echo $V1 $V2'
```

If the value is not specified, Podman will map the host variable value into the container.

```
podman run -e USER docker.io/alpine sh -c 'echo $USER'
```

Demo

Running a **GROMACS** benchmark with Podman

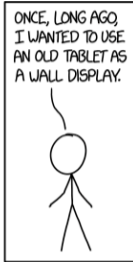
Running a GROMACS benchmark with Podman

Specifications

- Container Engine ⇒ Podman (on Grex)
- Image ⇒ nvcr.io/hpc/gromacs:2023.2
(from the [nVidia NGC Catalog](#))
- Software ⇒ GROMACS
- Benchmark ⇒ STMV

Is Podman a silver bullet?

- ▶ Is the same software already provided via modules-based HPC software stack?
- ▶ Running software from *well-built* images
 - ▶ Making or finding a suitable image is a bit of work
 - ▶ Bleeding-edge versions of programs could be poorly maintained/tested (including their images)
- ▶ Encapsulating software and sometimes data to reduce number of files (e.g. python or conda based programs)



Questions?

Thank you