Why Is Podman Recommended for Use at NAS?

Podman is an open-source Linux container management tool developed and maintained by Red Hat. It is a preferred container solution and an alternative to Singularity (currently disabled) for the NAS community mainly due to its daemonless and rootless approach. Specifically:

- Unlike Docker, Podman interacts directly with the `runc` container runtime and it does not use any root-owned daemon (such as `dockerd`) to pull, push, build images, or spawn containers. A daemon owned by root can gain privileges to perform unauthorized and harmful activities.
- Unlike Singularity, a popular HPC container technology that sometimes needs root privileges to build or run containers, Podman is designed to function without root. This makes Podman the preferred container to Singularity in the following ways:

    - **No Root-Owned "setuid" Binaries**

      setuid is a Linux permission that allows a user to execute a file with the permission of the owner of the file. A file set with `setuid` will have the `x` owner permission bit replaced with `s`. For example:

      ```
      -rwsr-xr-x      filename
      ```

      Although `setuid` is disallowed by default within Singularity containers, a root user can override this default. This approach is not allowed by the NAS security policy.

      However, Podman does not use root-owned setuid binaries.

      Note: Apptainer, an open-source project forked from Singularity, is designed with an architecture without the use of `setuid` binaries. As of December 2024, Apptainer is being evaluated to replace Singularity for use at NAS.

    - **User Namespaces Without "fakeroot"**

      A user namespace is a Linux feature that allows mapping of users in the container to different users in the host. When this feature is combined with the use of the Singularity `--fakeroot` or `-f` option, processes can have root privileges inside the container while being mapped to non-root users outside the container. This combination is useful when building or running a Singularity container that requires root privileges.

      Podman also uses user namespaces but operates without the need of fakeroot. A rootless Podman container maps the user's ID into the container as root of the user name namespace.

      Note: Proper support of namespaces is required not just in Podman, but also in the filesystem(s), network, and operating system to prevent security vulnerabilities.

In addition to its daemonless and rootless approach, Podman provides a command line interface similar to Docker, making it easy for users familiar with Docker to use Podman as a drop-in replacement. For example, simply replace `docker pull centos` with `podman pull centos`.

For more information about Podman, see the Podman website.

Podman is an experimental offering on NAS systems with the x86_64 architecture. This article helps you get started, provides a list of commands, and outlines the limitations of this offering.

For more information about Podman, see the Podman website.

Preparation Steps

1. Send an email to support@nas.nasa.gov to request access to Podman. A system administrator will create a directory for you under /nasa2 where images are to be stored. Podman images and containers can only be stored on a local filesystem or a filesystem that supports NFS version 4.2, such as /nasa2.

2. Create a directory called containers under $HOME/.config, if it does not already exist. Then, create a new storage.conf file under $HOME/.config/containers with the following contents.

   Note: Remember to replace *your_nas_username* with your NAS username.

   ```
   [storage]
   driver = "overlay"
   runroot = "/nasa2/your_nas_username/containers/runtime"
   graphroot = "/nasa2/your_nas_username/containers/storage"
   rootless_storage_path="$HOME/.local/share/containers/storage"
   [storage.options]
   additionalimagestores=['/nasa2/your_nas_username/podman']
   ignore_chown_errors = "true"
   mount_program = "/usr/bin/fuse-overlayfs"
   [storage.options.overlay]
   force_mask="755"
   mount_program = "/usr/bin/fuse-overlayfs
   ```

3. Create a directory named podman under your /nasa2/*your_nas_username* directory, using

   ```
   % mkdir /nasa2/your_nas_username/podman
   ```

   Note: A directory called /nasa2/*your_nas_username*/containers will be created automatically after your first use of Podman commands such as info, images, pull, or run.

4. Run this quick test on either your $HOME, /nobackup, or /nasa2 directory to make sure you are ready to use Podman:

   ```
   % podman run -it docker.io/sylabsio/lolcow
   Trying to pull docker.io/sylabsio/lolcow:latest...
   Getting image source signatures
   Copying blob 5ca731fc36c2 done   |
   Copying blob 16ec32c2132b done   |
   Copying config eb03f9d8dd done   |
   Writing manifest to image destination
   WARN[0017] Failed to mount subscriptions, skipping entry in
   /usr/share/containers/mounts.conf: getting host subscription data:
   failed to read subscriptions from "/usr/share/rhel/secrets":
   open /usr/share/rhel/secrets/rhsm/syspurpose/syspurpose.json: permission denied
    _____________________________
   < Wed Aug 14 18:53:27 UTC 2024 >
    -----------------------------
           \   ^__^
            \  (oo)\_______
               (__)\       )\/\
                  ||----w |
                  ||     ||
   ```

   Note: You may disregard the warning message about Red Hat subscription and permission (WARN[0017]).

Podman Commands

Podman can be used on a subset of NAS systems including some front ends (PFEs, AFEs, CFEs) andsome compute nodes.

You can find the location and version of Podman installed on NAS systems as follows:

```
% which podman
/usr/bin/podman

% podman --version
podman version 4.9.4-rhel

% podman version
Client:       Podman Engine
Version:      4.9.4-rhel
```

```
API Version:  4.9.4-rhel
Go Version:   go1.21.13 (Red Hat 1.21.13-2.module+el8.10.0+22329+6cd5c9c6)
Built:        Wed Sep 25 10:36:19 2024
OS/Arch:      linux/amd64
```

The Podman command line interface (CLI) is similar to Docker's CLI. For a complete list of Podman commands and the description of each command, do one of the following:

- % podman --help (or podman -h)
  % podman command_name --help (most command_name will allow podman command_name -h)

  For example,
  % podman run --help (the run command does not accept -h)

- Read the Podman Commands page on the Podman website.

Limitations

- Podman can be used on only a few NAS compute node types.
  - Node types where Podman will run: mil_ait, rom_ait, mil_a100, gh200
  - Node types where Podman will **NOT** run: bro_ele, sky_ele, cas_ait (due to rack leader and filesystem configuration issues which hinder the required UID and GID mapping).
- You can run Podman on a single node. Until approved by NAS security, running MPI across multiple nodes is not allowed.
- Access to the internet is only available from the NAS front-end nodes, but not from the compute nodes. Therefore, any Podman activities that require internet access can only be performed on the front-end nodes.
- NAS front-end nodes are shared among all users. When pulling or building an image on a busy front-end node, Podman may appear to stall.
- The total space of /nasa2 is 1,000 gigabytes (GB). It is shared by all users. Use the command df -h /nasa2 to find the amount of space used and available under /nasa2. Your storage area for Podman images and containers can be found at /nasa2/*your_nas_username*/containers/storage.

  Note: A container is a runtime instance of an image. You can run multiple containers of the same image. Please remove unwanted containers and images to save space. An image cannot be removed before any container associated with it—regardless of its status—has been removed.

  To find a container's ID and its status, stop running the container, and delete it:

  ```
  % podman container list --all  (or podman container ps --all or podman ps --all)
  % podman container stop containerID
  % podman rm containerID   (if necessary, podman rm --force containerID)
  ```

  To find an image's ID and delete it:

  ```
  % podman images       (to list images in local storage)
  % podman rmi imageID
  ```

- Due to compatibility issues with the current operating system (Red Hat 8.x) of NAS x86-64 systems, Podman versions 5.x and newer cannot be installed.

This article describes several ways of getting an image ready to use on NAS systems: pulling a pre-built image from a registry; building an image with a containerfile; and saving an image to a tar file.

Note: Pulling or building an image that requires internet access can only be done on a NAS front-end node.

Pulling a Pre-Built Image From a Registry

There are many publicly available registries that host pre-built images. You can pull an image from a registry using its fully qualified image name including the registry server, namespace, image name, and tag. For example:

```
% podman pull docker.io/library/python:latest
```

You can also find information about how an image was built. For example:

```
% podman inspect python
or
% podman inspect docker.io/library/python:latest
```

At NAS, the file /etc/containers/registries.conf includes a list of registries where you can search for images by using the podman search command:

- registry.access.redhat.com
- registry.redhat.io
- docker.io

For example, to search images related to Python in these three registries, run podman search python:

```
% podman search python
NAME                          DESCRIPTION
registry.access.redhat.com/   Python 3.3 platform for building and running...
openshift3/python-33-rhel7
...
registry.redhat.io/           Python 3.3 platform for building and running...
openshift3/python-33-rhel7
...
docker.io/library/python      Python is an interpreted, interactive, objec...
...
```

Then, you can pull one of the images shown in the search results. For example:

```
% podman pull registry.access.redhat.com/openshift3/python-33-rhel7
```

Note: The two Red Hat registries (registry.access.redhat.com and registry.redhat.io) hold essentially the same container images. However, some images in registry.redhat.io require an active Red Hat account; these are only available in that registry.

In addition, a few files under /etc/containers/registries.conf.d define short names for some images (mostly OS-related short names). For example, there is an entry "centos" = "quay.io/centos/centos" in the file 000-shortnames.cong. Therefore, the following command will pull the image centos from quay.io:

```
% podman pull centos
Resolved "centos" as an alias
  (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull quay.io/centos/centos:latest...
...
Getting image source signatures
Copying blob 7a0437f04f83 done   |
Copying config 300e315adb done   |
Writing manifest to image destination
300e315adb2f96afe5f0b2780b87f28ae95231fe3bdd1e16b9ba606307728f55
```

Note: By default, the podman pull command will pull an image with the architecture according to the host machine on which this command is run (for example, amd64). To explicitly specify an architecture, add the --arch option (for example, --arch=arm64). Keep in mind that an image with **amd64** architecture most likely will not run on a host of **arm64** architecture, and vice versa. To learn more, see this Red Hat blog post.

Building an Image With a Containerfile

You can build an image from scratch using a Containerfile, which is a configuration file that automates the process of creating an image. It uses the same syntax as a Dockerfile.

By default, to build an image, Podman looks for a file named Containerfile. It will fall back to Dockerfile if Containerfile does not exist. You can also explicitly specify the name of a Containerfile or Dockerfile, as shown in an example below.

Note: Images and containers are stored under /nasa2/your_nas_username/containers/storage.

For instructions on creating a Containerfile, read **man containerfile** (or this ubuntu man page).

Here is a Containerfile, modified from an example downloaded from GitHub, for building an image of lolcow:

```
FROM ubuntu:16.04

RUN apt update
RUN mkdir -p /usr/local/share/ca-certificates && chown root: \
/usr/local/share/ca-certificates
RUN apt-get install -y ca-certificates
RUN apt-get install -y fortune cowsay lolcat

ENV PATH /usr/games:${PATH}
ENV LC_ALL=C

ENTRYPOINT fortune | cowsay | lolcat
```

Note: A command line in the code box above is too long to be formatted as one line, so it is broken with a backslash (\).

Examples

Here are a few examples of building a lolcow image from the Containerfile.

Example 1

To build an image of lolcow if this Containerfile is in the current directory:

```
% podman build .
```

Example 2

To build an image of lolcow by specifying the location and name of the Containerfile, for example, $HOME/containerfile_test:

```
% podman build --file $HOME/containerfile_test
or
% podman build -f $HOME/containerfile_test
```

Example 3

To build an image from a Containerfile in the current directory and apply a tagged name to the image (for example, lolcow):

```
% podman build --tag lolcow .
or
% podman build -t lolcow .
```

If this image is successfully built, you will see it as "localhost/lolcow" under the REPOSITORY column when you run the `podman images` command. For example:

```
% podman images
REPOSITORY                TAG         IMAGE ID      CREATED         SIZE
localhost/lolcow          latest      76975188c890  22 seconds ago  249 MB
docker.io/library/ubuntu  16.04       b6f507652425  2 years ago     139 MB
```

When you run a container from the lolcow image, you will see something similar to the following:

```
% podman run 76975188c890
or
% podman run lolcow
or
% podman run localhost/lolcow

WARN[0000] Failed to mount subscriptions, skipping entry in
/usr/share/containers/mounts.conf: getting host subscription data:
failed to read subscriptions from "/usr/share/rhel/secrets":
open /usr/share/rhel/secrets/redhat.repo: permission denied
 _____________________________________
/ Q:Why do astronauts use Linux? ? A: Because \
\ they can't open windows in space.                /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
               ||----w |
               ||     ||
```

Note: Ignore the warning message.

TIP: When building an image without a tag, the image is identifiable only by its imageID. However, you can add a tag to an existing untagged image using the `podman tag` command. For example:

```
% podman tag 67aedfd8c12b lolcow2:24.11
```

Saving an Existing Image to a Tar File

You can save an existing image to a tar file. For example:

```
% podman image save -o lolcow.tar 76975188c890
or
% podman image save -o lolcow.tar lolcow
or
% podman image save -o lolcow.tar localhost/lolcow
```

The tar file lolcow.tar will be saved in the directory the above command was run.

If the image is removed, and later you want to load it from the tar file, do:

```
% podman load -i lolcow.tar
Getting image source signatures
Copying blob 47ef83afae74 done   |
....
Writing manifest to image destination
Loaded image: localhost/lolcow:latest

% podman images
REPOSITORY        TAG         IMAGE ID      CREATED         SIZE
localhost/lolcow  latest      76975188c890  10 minutes ago  249 MB

% podman run lolcow
/ Q: What is the sound of one cat \
\ napping? A: Mu.                 /
 ----------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Note: When running on a compute node, you may need to run the command `loginctl enable-linger your_nas_username` prior to running the podman command. Otherwise, you may encounter the following error:

```
Error: default OCI runtime "crun" not found: invalid argument
```

After you prepare a Podman image for use on NAS systems, you can run a command in a new or existing (running) container. You can also bind-mount your home and/or nobackup filesystems to a Podman container.

Note: After you complete your runs, be sure to remove unwanted containers or images as described in the **Limitations** section of Getting Started with Podman at NAS.

Introduction

Use the `podman run` command and its options, followed by an image name or image ID, to run a command in a new container from an image. The following options are commonly used:

- **--interactive (or -i):** to keep the standard input (STDIN) open even if not attached.
- **--tty or (-t):** to allocate a pseudo-TTY for container.
- **--rm:** to remove container and any anonymous unnamed volume associated with the container after exit.
- **--volume:** to bind mount a volume into the container.
- **--workdir (or -w):** to set working directory inside the container.
- **--env or (-e):** to set environment variables in container.

To see more options, do: `podman run --help`

To execute a command in a running container, use the `podman exec` command followed by its options and the name or ID of the container. Some of its options are the same as the `podman run` command options, such as -i, -t, -e, -w.

For more information, see: `podman exec -h`

The examples shown in this article use a centos image pulled from quay.io:

```
% podman pull centos
% podman images
REPOSITORY            TAG         IMAGE ID      CREATED      SIZE
quay.io/centos/centos  latest      300e315adb2f  3 years ago  217 MB
```

If you want information about how this image was created, do (for example):

```
% podman inspect centos
or
% podman inspect 300e315adb2f
```

Running a Command In a New Container From an Image

- To run a container in an interactive mode followed by running a few commands:

  ```
  % podman run -it centos
  # whoami
  root
  # pwd
  /
  # echo $HOME
  /root
  # ls /root
  anaconda-ks.cfg  anaconda-post.log  original-ks.cfg
  ```

  Notes:

  - At the container prompt #, you may also see the containerID, for example, 550df871ae08#.
  - This centos image was created with CMD in its Containerfile set to /bin/bash. Therefore, without a specified command after `podman run -it centos` in the command line, /bin/bash is run.
  - To exit out of this interactive mode, type `exit` or use Ctrl-D.

- To run a container with the ls command, then exit:

  ```
  % podman run centos ls
  ```

  or

  ```
  % podman run -it 300e315adb2f ls
  bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root ...
  ```

- To run a container and remove it after exiting:

  A container that exited is still stored in /nasa2/*your_nas_username*/containers. Add the option --rm if you want to remove it from

storage after exit.

```
% podman run --rm -it centos
#
```

If you have a container already running on a host, you can find its containerID and name using any of the following (for example):

```
% podman container list
```

or

```
% podman container ps
```

or

```
% podman ps

CONTAINER ID  IMAGE                           COMMAND  CREATED
550df871ae08  quay.io/centos/centos:latest  bash     4 minutes ago

STATUS         PORTS                   NAMES
Up 4 minutes                           cool_lederberg
```

Note: By default, the above commands only show containers that are running. To see all containers in all states, add --all or -a.

Bind-Mounting a Volume When Running a New Container

- To bind-mount your NAS /nobackup filesystem to the container and create a file *foo*:

  ```
  % podman run -v /nobackup/your_nas_username:/nobackup/your_nas_username -it centos
  # ls
  bin  dev  etc  home  lib  lib64  lost+found  media  mnt  nobackup  opt proc ...
  # cd /nobackup/your_nas_username
  # vi foo
  ```

- To bind-mount your NAS /nobackup filesystem as read-only, add :ro as follows:

  ```
  -v /nobackup/your_nas_username:/nobackup/your_nas_username:ro
  ```

- Most likely, an image already contains a directory named /home which is not your NAS home filesystem. To bind-mount your NAS home filesystem, use /u/*your_nas_username* or /homeX/*your_nas_username*, where X is a number associated with your NAS $HOME. You can find which homeX is yours with the command ls -ld /u/*your_nas_username* before bind-mounting it to the container. For example:

  ```
  % ls -ld /u/your_nas_username
  lrwxrwxrwx 1 root root 14 Feb  1  2018 /u/your_nas_username -> /home4/your_nas_username

  % podman run -v /home4/your_nas_username:/home4/your_nas_username -it centos
  # ls
  bin  dev  etc  home  home4  lib  lib64 lost+found  media  mnt opt  proc ...
  # cd /home4/your_nas_username
  ```

- To bind-mount both your NAS home and /nobackup filesystems, issue -v one time for your NAS home filesystem, and another -v for your NAS /nobackup filesystem.

Executing a Command In a Running Container

WARNING: This section does not apply to containers running on a compute node via PBS.

You can execute commands in a running container from the same host where it was started (for example, pfe25), but not from other hosts.

- Execute a command in the running container using either the name or the containerID of the container. For example:

  ```
  % podman exec cool_lederberg echo hello
  or
  % podman exec -it cool_lederberg echo hello
  or
  % podman exec -it 550df871ae08 echo hello
  hello
  ```

- Access bind-mounted filesystem in a running container. For example:

  ```
  % podman exec -it cool_lederberg ls /nobackup/your_nas_username
  ```

- Set the working directory to a bind-mounted filesystem in a running container. For example:

  ```
  % podman exec -w /nobackup/your_nas_username -it cool_lederberg bash
  # pwd
  /nobackup/your_nas_username
  ```

  Note: Similarly, you can set the working directory with the podman run command when starting a new container.

- Modify an environment variable in a running container:

  The variable PATH is used as an example.

  By default, PATH in a Podman container is set to: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

  The following example shows one method of modifying PATH. It assumes that the running container was created with your /homeX/*your_nas_user_name* bind-mounted and the containerID is 550df871ae08.

  - If you use csh:

    % setenv PATH "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/ \
    sbin:/bin:/**homeX/*your_nas_username*/bin**"

  - If you use bash:

    % export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/ \
    sbin:/bin:/**homeX/*your_nas_username*/bin**"

  Note: The command lines above are too long to be formatted as one line, so they are broken with a backslash (\).

  Use the -e option to set PATH for the container:

  % podman exec -e PATH=$PATH -it 550df871ae08 bash
  # echo $PATH
  /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin: \
  /bin:/**homeX/*your_nas_username*/bin**

  Note: Similarly, you can set environment variables with the `podman run` command when starting a new container.

Running a Container With a Workflow In a Script

Likely, you want to run multiple commands or a complicated workflow in a container. You can create a script for this purpose. For example, under your NAS $HOME, create a script, `test.sh`, which contains:

```
#!/bin/bash

whoami
pwd
ls
cd /homeX/your_nas_username
export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin: \
/usr/bin:/sbin:/bin:/homeX/your_nas_username/bin"
echo $PATH
```

Run `test.sh` to make sure the script can be executed:

```
% chmod u+x test.sh
```

When running a container, bind-mount your NAS $HOME and add `test.sh` as the command to be executed. For example:

```
% podman run -it -v /homeX/name:/homeX/name centos /homeX/name/test.sh
```

or

```
% podman run -it -v /homeX/name:/homeX/name -w /homeX/name centos ./test.sh
```

This article provides basic information on getting started with running a Podman container on one or more GPUs of a NAS Grace Hopper (gh200) or Milan A100 (mil_a100) node.

Note: Running Podman containers across nodes is not allowed per NAS security policy.

For demonstration, the image nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu22.04 is used in this article.

Before You Begin: Before pulling a Docker/Podman image from a remote repository, for example, the NVIDIA GPU Cloud (NGC) containers, read its online documentation (if available) to make sure that its CPU host architecture, GPU types supported (for example, H100, A100, and so on), and CUDA version are compatible with the GPU compute node you intend to run the container on.

Pulling an Image with the Correct Architecture

The container image you are interested in using might be built for one or both of these CPU host architectures:

- CPU host architecture on the **gh200** nodes: **arm64**
- CPU host architecture on the **mil_a100** nodes: **x86_64** or **amd64**

Before pulling a container image, check the manifest list of the container. In this example, the output of the following command line will show that both arm64 and amd64 host architectures are supported:

```
NAS_front_end% podman manifest inspect nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu22.04
```

Then, pull the image you want using the option --arch=xxx. For example, to pull an image with the arm64 architecture, run:

```
NAS_front_end% podman pull --arch=arm64 nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu22.04
```

You can also check the architecture after the image has been stored locally. For example:

```
NAS_front_end% podman images
REPOSITORY          TAG                    IMAGE ID      CREATED       SIZE
nvcr.io/nvidia/cuda 12.9.1-devel-ubuntu22.04  ec9b21c93944  2 weeks ago    8.77 GB

NAS_front_end% podman image inspect ec9b21c93944 | grep -i arch
```

Tip: You can modify the tag of the image with a different name to make its architecture obvious to you. For example, to tag an image with arm64, run:

```
NAS_front_end% podman tag ec9b21c93944 nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu22.04-arm64

NAS_front_end% podman images
REPOSITORY          TAG                          IMAGE ID      CREATED       SIZE
nvcr.io/nvidia/cuda  12.9.1-devel-ubuntu22.04        ec9b21c93944  2 weeks ago    8.77 GB
nvcr.io/nvidia/cuda  12.9.1-devel-ubuntu22.04-arm64  ec9b21c93944  2 weeks ago    8.77 GB

NAS_front_end% podman rmi nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu22.04

NAS_front_end% podman images
REPOSITORY          TAG                          IMAGE ID      CREATED       SIZE
nvcr.io/nvidia/cuda  12.9.1-devel-ubuntu22.04-arm64  ec9b21c93944  2 weeks ago    8.77 GB
```

Testing the Image with the NVIDIA System Management Interface (nvidia-smi)

1. Submit an interactive PBS job to access a GPU compute node. This example uses a gh200 node:

   ```
   NAS_front_end% qsub -I -q xxx -lselect=1:ncpus=72:ngpus=1:mem=460GB:model=gh200
   ```

2. Add all GPU devices on the node to the container.
   - For the gh200 nodes running with Red Hat 9.x and Podman 5.x, add to the podman run command:

     ```
     --gpus=all
     ```

   - For the mil_a100 nodes running with Red Hat 8.x and Podman 4.9x, add:

     ```
     --device nvidia.com/gpu=all
     or
     --device=nvidia.com/gpu=all
     ```

     Note: A mil_a100 node has a total of four GPUs. If you don't need all four GPUs, you can specify which one to use via a relative GPU ID or the GPU's universally unique identifier (UUID). For example, to add GPU 1 and GPU 3 (as seen on the host) to the container, use:

     ```
     --device=nvidia.com/gpu=1 --device=nvidia.com/gpu=3
     ```

     To use the UUIDs:

     ```
     --device nvidia.com/gpu=GPU-d4f2037f-1d66-de91-3f3c-1341443640b0
     ```

```
--device nvidia.com/gpu=GPU-2f2f99cf-a25c-bcdc-0389-fcc95a620c72
```

3. Verify that the container can see and use the GPU(s):

   Run the command `nvidia-smi -L` inside the container. For example, on a `gh200` node:

   ```
   GPU_node% podman run -it --rm --gpus=all ec9b21c93944 nvidia-smi -L

   ==========
   == CUDA ==
   ==========
   CUDA Version 12.9.1
   ...
   GPU 0: NVIDIA GH200 480GB (UUID: GPU-bf077a13-dcd7-7c4d-6ba5-15ba1dddf6f3)
   ```