Containers

Podman

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.

Getting Started with Podman at NAS

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.

Preparing a Podman Image for Use on NAS Systems

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
```

Running Podman Containers on a NAS GPU Node

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)
   ```

Apptainer

Enabling User-Defined Software Stack with Apptainer

Apptainer, previously known as Singularity, is an open-source project under the Linux Foundation since November 2021.

Apptainer containers provide an application virtualization layer enabling mobility of compute via both application and environment portability. With Apptainer, you can build a user-defined software stack including an application and all its dependencies (libraries, etc.) that runs on any other Linux system where Apptainer is installed.

Note: Apptainer retains the look and feel of Singularity and uses the same Singularity Image Format (SIF) for the container images.

Main Apptainer commands include:

apptainer help
: To find help on available options and commands.

apptainer build
: To build an Apptainer image.

apptainer shell
: To run a shell within a container.

apptainer exec
: To run a command within a container.

apptainer run
: To run a user-defined default command within a container.

Many ready-to-use Apptainer images from multiple sources are available for you to grab with the `apptainer build` command and use on your workstation or on NAS systems. Introduction to Using Apptainer Build provides some basic information on using the `apptainer build` command.

If you have an existing Apptainer image that is ready to run, you should be able to run it on any platform where Apptainer is installed.

Apptainer is installed for use on NAS systems with either x86_64 or aarch64 architecture, and with either the TOSS4 or TOSS5 operating system. You can use Apptainer after loading one of these modules:

apptainer/x86-toss4
: For use on older NAS systems with x86_64 architecture and the TOSS4 operating system, including Electra (bro_ele, sky_ele), Aitken (rom_ait, mil_ait) and some Cabeus systems (mil_a100).

apptainer/x86-toss5
: For use on the latest NAS Athena system (tur_ath) with x86_64 architecture and the TOSS5 operating system.

apptainer/arm-toss5
: For use on the Cabeus GH200 system (gh200) with aarch64 architecture and the TOSS5 operating system.

See Running an Apptainer Container Image on NAS Systems to learn how to run your image on NAS systems.

References

For more information about using Apptainer, see the official Apptainer Container Documentation.

The information provided in this article can be used on your workstation to build an Apptainer image. Some information here is also useful if you want to grab a ready-to-use Apptainer image from online resources for use on NAS systems.

Note: To use Apptainer on a NAS host machine, load an appropriate Apptainer module (apptainer/x86-toss4, apptainer/x86-toss5, apptainer/arm-toss5) depending on which architecture (x86_64 or aarch64) and operating system (toss4 or toss5) are on the host. See the second half of Enabling User-Defined Software Stack with Apptainer for more information.

Building an Apptainer Image

To build an Apptainer image, use the following:

```
host% apptainer build [local options ...] image_path build_spec
```

The *image_path* target specifies the output from the build process. The output can be one of the two formats:

- Default: The compressed Apptainer read-only image format (i.e., the default SIF format).
- Sandbox: This is a read-write container within a directory structure.

To build a compressed Apptainer read-only image, use the default format:

```
host% apptainer build ...
```

To build a read/write container within a directory structure, use the sandbox option:

```
host% apptainer build --sandbox ...
```

Note: It is common to use the sandbox mode to develop the container, as the image is writable. You canshell into it and install software and dependencies (some may require root privilege), until you are satisfied that your container will meet your needs. You can then build the container as a default Apptainer image for production use. The default format cannot be altered after it is created.

Warning: While you can write to a sandbox on your local workstation, writing to a sandbox is not allowed on NAS systems. You should rely on the use of the Apptainer definition file to adjust what is included in the container.

A *build_spec* target can be local or remote.

Acceptable formats for **local** targets include:

DEF file
: Definition file, which contains a recipe for building a container. A sample DEF file is shown in Example 2 below. More samples can be found in Apptainer's GitHub.

buildkit
: A build context directory, containing a Dockerfile to build.

directory
: A directory structure containing a (ch)root filesystem.

image
: A local image on your machine (such as *.sif or those with legacy Singularity formats).

For **remote** targets, use one of the following acceptable uniform resource identifiers (URI):

library://
: An image library (no default).

docker://
: A Docker registry (default is Docker Hub).

shub://
: An Apptainer registry (default is Singularity Hub).

oras://
: An OCI registry that holds SIF files using ORAS.

ipfs://
: An IPFS cluster, using a HTTP gateway.

More details can be found here.

Examples for Building Apptainer Images

Example 1: Build an image from the GtHub Container Registry (GHCR) under Docker:

```
host% apptainer build lolcow.sif docker://ghcr.io/apptainer/lolcow
```

Example 2: Build an image from a definition (DEF) file by following these steps.

Note: This example is taken from Building containers from Apptainer definition files from the the Apptainer quickstart guide.

1. Create a sample definition file named lolcow.def with the following content.

```
BootStrap: docker
From: ubuntu:22.04

%post
  apt-get -y update
  apt-get -y install cowsay lolcat

%environment
  export LC_ALL=C
  export PATH=/usr/games:$PATH

%runscript
  date | cowsay | lolcat
```

Tip: For building Apptainer images with MPI libraries, see the examples in this Apptainer page. If you are building a container image from an existing base image located on a local system, use (for example):

```
Bootstrap: localimage
From: /path/to/your_local_image.sif

% post
 ....
```

2. Build a SIF file from the DEF file using:

```
host% apptainer build lolcow_from_def.sif lolcow.def
```

Example 3

Build a sandbox directory (named lolcow_from_sif) from the SIF file:

```
host% apptainer build --sandbox lolcow_from_sif  lolcow_from_def.sif
```

Example 4

Build a SIF image (lolcow_new.sif) from the sandbox directory:

```
host% apptainer build lolcow_new.sif lolcow_from_sif
```

If you have an existing Apptainer image that is ready to run, you should be able to run it on any platform where Apptainer is installed. This article provides information to help you run an Apptainer container image on NAS systems.

Note: A container is a runtime instance of an image. You can run multiple containers of the same image.

Unpacking the Image Tar File on NAS Systems

If you transferred your Apptainer container image to a NAS system as a compressed tar file, use the appropriate tool to uncompress and unpack the tar file. For example:

```
host% cd /nobackuppX/username/dir_containing_the_tar_file
host% tar -xzf your_image_name.tgz
```

where *X* in nobackupp*X* is one of [17-19,27-29]. Use the number of the /nobackup directory you are assigned to.

Accessing Apptainer at NAS

Apptainer is installed for use on NAS systems with either x86_64 or aarch64 architecture, and with either the TOSS4 or TOSS5 operating system. You can use Apptainer after loading one of these modules:

apptainer/x86-toss4
: For use on the front ends (PFEs, AFEs, and CFEs) and compute nodes of older NAS systems with x86_64 architecture and the TOSS4 operating system, including Electra (bro_ele, sky_ele), Aitken (rom_ait, mil_ait) and some Cabeus systems (mil_a100).

apptainer/x86-toss5
: For use on the front ends (ATHFEs) and compute nodes of the latest NAS Athena cluster (tur_ath) with x86_64 architecture and the TOSS5 operating system.

apptainer/arm-toss5
: For use on the front ends (CGHFEs) and compute nodes of the Cabeus GH200 system (gh200) with aarch64 architecture and the TOSS5 operating system

Here are example commands for the different environments listed above:

```
cfe01% module load apptainer/x86-toss4

athfe01% module load apptainer/x86-toss5

cghfe01% module load apptainer/arm-toss5
```

Running the Container Image

Depending on what you want to do, you can use either the shell, exec, or run command of Apptainer.

Warning: Use of the --fakeroot option is not allowed on NAS systems.

shell
: The shell command allows you to spawn a new Bourne shell within your container and interact with it as though it were a small virtual machine. To exit the shell, enter exit at the Apptainer prompt.

exec
: The exec command allows you to execute a custom command within a container.

run
: Apptainer containers may contain runscripts. These are user defined scripts that define the actions a container should perform when someone runs it. The runscript can be triggered with the run command, or simply by calling the container as though it were an executable.

For example, if you port a sandbox lolcow container (renamed from lolcow_from_sif as described in this article) to a NAS system, you may do:

```
host% apptainer shell lolcow
Apptainer> echo hello
hello

Apptainer> exit

host% apptainer exec lolcow which cowsay
/usr/games/cowsay
host% apptainer exec lolcow cowsay hello
 _______
< hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
```

```
              ||----w |
              ||    ||
host%

host% apptainer run lolcow
 _______________________________________
<Mon Dec 22 14:24:30 PST 2025 >
 ---------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
              ||----w |
              ||    ||
host%
```

Setting Up Environment Variables

You can modify the environment such as the $PATH within your container. For example:

```
host% apptainer shell lolcow
Apptainer> echo $PATH
/usr/games:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
Apptainer> export PATH=$PATH:/u/username/bin
Apptainer> echo $PATH
/usr/games:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u/username/bin
```

Accessing Files on NAS Systems

You can access files on NAS systems from within the container. By default, Apptainer bind-mounts your NAS $HOME (as /u/*username*), /tmp, and $PWD into your container at runtime. If you run the apptainer command from your /nobackupp*X*/$USER directory, the $PWD variable is then /nobackupp*X*/$USER and you can access the files and directories there.

Warning: The automatic mount for $PWD may fail if the path contains symlinks that resolve to different locations on the host versus inside the container. This will happen when $PWD is /nobackup/*username,* which is a symlink to the actual /nobackupp*X*/*username*. If you want to rely on Apptainer bind-mounting your $PWD when $PWD is your NAS nobackup directory, access it as /nobackupp*X*/*username* instead of /nobackup/*username*. To avoid this confusion, use the --bind option to bind-mount your /nobackup directory, as described below.

```
host% cd /nobackuppX/username/dir_containing_lolcow_image
host% apptainer shell lolcow
Apptainer> pwd
/nobackuppX/username/dir_containing_lolcow_image
Apptainer> cd /
Apptainer> ls
bin   dev    etc  lib lib64 media  nobackuppX   proc  run singularity  sys  u    var
boot  environment  home  lib32 libx32 mnt    opt     root  sbin srv      tmp  usr
```

You can specify additional directories to bind mount into your container with the --bind option.

Note: You can replace --bind with -B or set the environment variable APPTAINER_BIND or APPTINER_BINDPATH on the host system. See this Apptainer page for more information.

For example, if you run the apptainer command from your $HOME, and you want to be able to access files and directories under your /nobackupp*X*/$USER, you can do:

```
host% apptainer exec --bind /nobackuppX/username:/mnt lolcow ls /mnt
or
host% apptainer exec --bind /nobackup/username:/nobackup/username lolcow ls /nobackup/username
```

For read-only filesystems such as /nasa, optionally add :ro to the command as follows:

```
host% apptainer exec --bind /nasa:/nasa:ro lolcow ls /nasa
```

You can bind-mount multiple filesystems. For example,

```
host% apptainer shell --bind /nasa:/nasa:ro --bind /nobackuppX/username:/mnt lolcow
or
host% apptainer shell --bind /nasa:/nasa:ro,/nobackuppX/username:/mnt lolcow
Apptainer> cd /nasa
Apptainer> pwd
/nasa
Apptainer> cd /mnt
Apptainer> pwd
/mnt
Apptainer> exit
host%
```

Further Reading

Read this article for information on Using OpenMPI as host MPI for running containers in hybrid MPI mode on NAS systems.

Read this article for information on Using Cray MPICH as host MPI for running containers in hybrid MPI mode on Athena.

Using Cray MPICH As Host MPI For Running Containers in Hybrid MPI Mode on Athena

Running Apptainer in hybrid MPI mode allows you to call mpiexec or a similar launcher on the apptainer command itself when you execute an Apptainer container with an MPI code. The MPI process outside of the container will then work in tandem with a compatible MPI inside the container and the containerized MPI code to instantiate the job.

Currently, the only MPI library on Athena that supports the HPE Cray Slingshot 11 interconnect, for best communication performance, is the proprietary Cray MPICH. In contrast, containers typically include open-source MPICH, Intel MPI, or OpenMPI. This article provides information on how to use Cray MPICH on the Athena host to work with MPICH, Intel MPI, or OpenMPI inside a container for multi-node runs.

Note: For containers that use Intel MPI, it is possible to combine some of the techniques described in this article with the information in Using Cray PALS Without Cray MPICH to Launch Intel MPI Applications on Athena for multi-node container runs.

Recipe

Step One

To run your application, use a method that allows Cray MPICH to work with executables—either on the host or within a container—built with non-Cray-MPICH. As described in this article for MPI applications running natively on Athena, you can use one of the following two methods, depending on which flavor of MPI was used when the application was compiled:

- If you have an application that was built with standard MPICH or Intel MPI—which are Application Binary Interface (ABI)-compatible with Cray MPICH—use the cray-mpich-abi module.

  ```
  % module load cray-pals
  % module load cray-mpich-abi
  % module load hpe-libfabric
  % mpiexec [options] executable_built_with_mpich_or_intel_mpi
  ```

- If you have a **C/C++ (not Fortran)** application that was built with OpenMPI, use the Cray MPICH ABI runtime translator, mpixlate, to translate the executable from OpenMPI to Cray MPI.

  ```
  % module load cray-pals
  % module load cray-mpich
  % module load cray-mpixlate
  % module load hpe-libfabric
  % mpiexec [options] mpixlate -s ompi.40 -t cmpich.12 executable_built_with_openmpi
  ```

You will also use these methods later, when running an MPI application inside a container with a hybrid-MPI mode. An example is shown below in the sample script section.

Note: Starting June 15, 2026, a libfabric module is no longer loaded upon login. Instead, the libfabric library under/usr/lib64 will be used by default. However, for jobs that run with containers using the methods described in this article, you will need to load an hpe-libfabric module where the libfabric library is installed under /nasa instead of /usr/lib64 which causes collision with the container's /usr/lib64.

Step Two

Choose an appropriate Cray MPICH module that complies with the MPI standard supported by the MPI inside the container. Note that versions of MPI standard approved by the MPI Forum include 1.0, 1.1, 1.2, 1.3, 2.0, 2.1, 2.2, 3.0, 3.1, 4.0, 4.1 and 5.0.

Note: Currently, there are two Cray MPICH modules included in the Cray Programming Environment (CPE) 25.09 on Athena. If Athena is updated with a new CPE version, the Cray MPICH modules may change.

- The default cray-mpich/8.1.33 (default), based upon Argonne National Laboratory (ANL) MPICH 3.4a2, provides full support of the MPI-3.1 standard
- The cray-mpich/9.0.1, based on ANL MPICH 4.1.2, supports the MPI-4.0 standard

Be sure to load the appropriate Cray MPICH module depending on which MPI standard is required to support your application. For example, if the MPI library inside the container uses the MPI-4.0 standard, you will need to load the module cray-mpich/9.0.1 on the host.

Different flavors of MPI libraries available for public use are listed here:

Open-source MPICH

The version number of MPICH more or less reflects the version of MPI standard it supports.

- MPICH 3.0.x supports MPI-3.0
- MPICH 3.[1-4].x supports MPI-3.1
- MPICH 4.0.x supports MPI-4.0
- MPICH 4.[1-4].x supports MPI-4.1
- MPICH 5.0.x supports MPI-5.0

MPICH includes the mpichversion command to easily check the version number. When using mpichversion from a non-default version of cray-mpich, you need to prepend $CRAY_LD_LIBRARY_PATH (defined in the corresponding cray-mpich module) to $LD_LIBRARY_PATH.

For example, if cray-mpich/8.1.33 is the default on a system, and you want to use cray-mpich/9.0.1 whose modulefile includes the following prepend-path:

```
CRAY_LD_LIBRARY_PATH /opt/cray/pe/mpich/9.0.1/ofi/cray/20.0/lib: \
/opt/cray/pe/mpich/9.0.1/gtl/lib
```

then you must modify $LD_LIBRARY_PATH as demonstrated below:

```
host% module switch cray-mpich cray-mpich/9.0.1
host% which mpichversion
/opt/cray/pe/mpich/9.0.1/ofi/cray/20.0/bin/mpichversion
host% mpichversion
MPI VERSION : CRAY MPICH version 8.1.33.496 (ANL base 3.4a2)
^^ wrong version for cray-mpich/9.0.1 ^^
MPI BUILD INFO : Tue Jul 15 20:51 2025 (git hash b449595)

With bash,
host% export LD_LIBRARY_PATH="${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}"

With csh,
host% setenv LD_LIBRARY_PATH "${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}"
host% mpichversion
MPI VERSION : CRAY MPICH version 9.0.1.498 (ANL base 4.1.2)
^^ correct version for cray-mpich/9.0.1 ^^
MPI BUILD INFO : Wed Jul 16 9:35 2025 (git hash 0848216)
```

Intel MPI

Here are the Intel MPI versions and the MPI standard they support:

- Versions 2017 and later support MPI-3.1 standard
- Versions 2021.12 (distributed with oneAPI 2024.1) and later include support for MPI-4.0 features
- Versions 2021.16 (distributed with oneAPI 2025.2) and later provides complete support for MPI-4.1
- Version 2021.17 (distributed with oneAPI 2025.3) includes MPI-5.0 standard technical preview for C

OpenMPI

The OpenMPI version number does not reflect the MPI standard version it supports. For example,

- MPI-3.0 standard conformance was introduced since OpenMPI 1.8
- MPI-3.1 standard conformance was introduced since OpenMPI 2.0
- Partial support for MPI-4.0 started with OpenMPI 5.x

TIP: Use the command ompi_info to find the version of MPI API it fully supports.

Step Three

Ensure that the host's communication libraries are visible inside the container. This allows MPI jobs in the container to access Athena's hardware and take advantage of its high-performance network and multiple network related libraries.

Apptainer provides multiple environment variables for this purpose.

- **Set the Environment Variable APPTAINERENV_LD_LIBRARY_PATH on the Host**

  Use APPTAINERENV_LD_LIBRARY_PATH to set or override LD_LIBRARY_PATH inside the container. Ensure that the Libfabric library installed under the host's /nasa directory and the Cray MPICH, Process Management Interface (PMI) and Parallel Application Launch Service (PALS) libraries from the host's /opt/cray directory are defined in the container's library search path. If using the mpixlate utility, also add the path to the mpixlate library.

  The examples below show how to set this variable on the host. The examples assume that Athena uses CPE 25.09, so the available Cray MPICH versions are 8.1.33 and 9.0.1. For the path_to_libfabric_library shown in the examples, use the LD_LIBRARY_PATH output from the command module show hpe-libfabric.

  WARNING: The command lines shown in each section below are too long to be formatted as one line, so they are broken with a backslash (\). You should revert them back to a single long line in your script. There should be no space between entries of various libraries.

  Example 1: If a container was built with the GNU compiler and an open-source MPICH library that supports the MPI-3.1 standard:

  ```
  export APPTAINERENV_LD_LIBRARY_PATH= \
  "path_to_libfabric_library: \
  /opt/cray/pe/mpich/8.1.33/ofi/gnu/11.2/lib-abi-mpich: \
  /opt/cray/pe/lib64: \
  /opt/cray/pals/1.7/lib:/.singularity.d/libs"
  ```

  Note: The PMI libraries can be found under /opt/cray/pe/lib64 or /opt/cray/pe/pmi/default/lib.

  Example 2: If a container was built with an Intel compiler and an Intel MPI library that supports MPI-4.0 standard:

```
export APPTAINERENV_LD_LIBRARY_PATH= \
"path_to_libfabric_library: \
/opt/cray/pe/mpich/9.0.1/ofi/intel/2022.1/lib-abi-mpich: \
/opt/cray/pe/lib64: \
/opt/cray/pals/1.7/lib:/.singularity.d/libs"
```

Example 3: If a container was built with the GNU compiler and an OpenMPI library that supports MPI-3.1 standard:

```
export APPTAINERENV_LD_LIBRARY_PATH= \
"path_to_libfabric_library: \
/opt/cray/pe/mpixlate/1.0.9/ofi/gnu/11.2/lib: \
/opt/cray/pe/mpich/8.1.33/ofi/gnu/11.2/lib: \
/opt/cray/pe/lib64: \
/opt/cray/pals/1.7/lib:/.singularity.d/libs"
```

If other libraries are needed inside the container, add them just before /.singularity.d/libs.

- **Set the environment variable APPTAINER_CONTAINLIBS on the host**

  You can add the APPTAINER_CONTAINLIBS variable to specify a string of library file names (comma separated) to bind to the /.singularity.d/libs directory inside the container. This method is useful when a host library under (for example)/usr/lib64 is needed by the container, but binding the host /usr/lib64 to the container will collide with the container's /usr/lib64.

  For example, the /usr/lib64/libcxi.so.1 library on the host is needed in order for a container to utilize the Cray Slingshot 11 interconnect. To make this library available to the container (and other libraries that the application may need), you can set:

```
export APPTAINER_CONTAINLIBS="/usr/lib64/libcxi.so.1, \
/usr/lib64/libjson-c.so.5,/usr/lib64/libnl-3.so.200, \
/usr/lib64/libtinfo.so.6"
```

  When the utility mpixlate is used, additional libraries are needed:

```
export APPTAINER_CONTAINLIBS="/usr/lib64/libcxi.so.1, \
/usr/lib64/libjson-c.so.5,/usr/lib64/libnl-3.so.200,/usr/lib64/libtinfo.so.6, \
/usr/lib64/libcurl.so.4,/usr/lib64/libunistring.so.2,/usr/lib64/libldap.so.2, \
/usr/lib64/liblber.so.2,/usr/lib64/libsasl2.so.3,/usr/lib64/libcrypt.so.2"
```

  If you experience errors indicating you are missing some additional libraries, check whether they exist under /usr/lib64 on the host and add them to the list above.

Step Four

For large message sizes, you might need to set the MPICH_SMP_SINGLE_COPY_MODE variable to NONE. This variable controls the intranode communication mechanism in HPE Cray MPICH (and related Cray/TOSS MPI builds), specifically for large message transfers between CPU processes. If you encounter hangs or errors, you can try debugging by setting this variable to NONE to disable high-performance single-copy shared memory optimizations, forcing data movement through slower paths.

There is more than one way to set this variable to pass it into the container. Here is one option:

```
mpiexec [mpiexec options] \
--env MPICH_SMP_SINGLE_COPY_MODE NONE \
apptainer exec ...\
```

Step Five

If needed, use the NAS mbind.x tool—which does process/thread pinning—to bypass a known Cray PALS issue in which Cray MPICH copies the executable to /run/palsd/<some_hash>/executable to run multi-node jobs. This breaks codes that query the location of the executable to find files needed by the executable.

To bypass this issue, add mbind.x before the executable. The daemon palsd then copies mbind.x to /var/run/palsd/<some-hash>/mbind.x and runs the executable from its intended location.

When running an Apptainer container, the orginal executable is apptainer. In multi-node runs, a failure can occur where palsd copies this executable (apptainer) from its installed location under /nasa to /run/palsd/<some_hash>/apptainer on the compute node. During startup, apptainer then searches for apptainer.conf but cannot find it as /run/palsd/<some_hash>../etc/apptainer/apptainer.conf, causing the run to fail. To work around this issue, replace:

```
mpiexec [mpiexec options] apptainer exec ...
```

with

```
mpiexec [mpiexec options] --cpu-bind none \
/u/scicon/tools/bin/mbind.x apptainer exec ...
```

However, if the mpixlate utility is used, there is no need to use mbind.x.

```
mpiexec [mpiexec options] mpixlate -s ompi.40 -t \
cmpich.12 apptainer exec ...
```

Step Six

Bind-mount the /opt/cray, /run/palsd, /nobackup/*username*, and /nasa filesystems for use by the container. Specifically, add the following after the apptainer exec command:

- **-B /opt/cray:/opt/cray**

  A container may already include the /opt directory. However, unless the container was built with the licensed Cray software stack, the /opt/cray directory should not exist within the container. As stated earlier, many of the communication libraries on the host under /opt/cray are needed for running a container in hybrid MPI mode. Therefore, you should bind mount /opt/cray for use by the container.

- **-B /run/palsd:/run/palsd**

  The palsd is the daemon of Cray's PALS for managing, launching, and coordinating MPI processes. The /run/palsd directory is used by PALS as a temporary runtime directory for socket files and communication metadata. It is needed when running the container across nodes.

- **-B /nobackup/*username*:/nobackup/*username***

  This is needed if your container or anything you need to run the container is located under your /nobackup directory.

- **-B /nasa:/nasa**

  This is needed since the command apptainer and the libfabric library from an hpe-libfabric module that you load are is installed under /nasa on Athena.

Note: Your NAS $HOME filesystem is automatically bind mounted into the container as /u/*username*, not /home*X*/*username*.

Sample Script

The sample script below is for the following scenario:

1. Running a container across two Athena compute nodes with 256 MPI ranks running on each node
2. Using an open-source MPICH library which supports MPI-3.1 standard inside the container
3. An executable inside the container built with GNU and MPICH

For other scenarios, modify the script with information provided in the Recipe section.

TIP: Test your setup with a hello world MPI before trying your executable.

```
#PBS -lselect=2:ncpus=256:mpiprocs=256:model=tur_ath
#PBS -lwalltime=2:00:00
#PBS -q normal
#PBS -j oe
#PBS -koed
#
# Note: By default, cray-mpich/8.1.33 and cray-pals/1.7.1
# should have been automatically loaded when your job starts.
# Load an hpe-libfabric module where libfabric is installed under /nasa
# instead of using libfabric under /usr/lib64

module switch PrgEnv-cray PrgEnv-gnu
module load hpe-libfabric
module load apptainer-x86-toss5
which mpiexec
which apptainer

# replace setenv xxx yyy with export xxx=yyy if
# using bash shell

# For APPTAINERENV_LD_LIBRARY_PATH,
# replace "path_to_libfabric_library" with the one
# shown in the output of module show hpe-libfabric

# If you encounter an error complaining that libpmi
# is not found, add /opt/cray/pe/pmi/default/lib

# Replace "other_libraries_from_container" with
# the actual library names inside container
# The two setenv command lines below are too long to be formatted as one line,
# so they are broken with a backslash (\). You should revert them back to a single long
# line in your script. There should be no space between entries of various libraries.

setenv APPTAINERENV_LD_LIBRARY_PATH  \
"path_to_libfabric_library:/opt/cray/pe/mpich/8.1.33/ofi/gnu/11.2/ \
lib-abi-mpich:/opt/cray/pe/lib64:/opt/cray/pals/1.7/ \
lib:other_librarys_from_container:/.singularity.d/libs"

#It is possible that more libraries under /usr/lib64 are needed beyond the list below
```

```
setenv APPTAINER_CONTAINLIBS "/usr/lib64/libcxi.so.1,/usr/lib64/libjson-c.so.5, \
/usr/lib64/libnl-3.so.200,/usr/lib64/libtinfo.so.6"

mpiexec -l -np 512 --ppn 256 \
--env MPICH_SMP_SINGLE_COPY_MODE NONE \
--cpu-bind none /u/scicon/tools/bin/mbind.x \
apptainer exec \
-B /nobackup/username:/nobackup/username \
-B /opt/cray:/opt/cray \
-B /run/palsd:/run/palsd \
-B /nasa:/nasa \
/path/to/container_sif_file_or_sandbox \
/path/to/executable_inside_container
```

Using OpenMPI As Host MPI For Running Containers in Hybrid MPI Mode on NAS Systems

Running Apptainer in hybrid MPI mode allows you to call mpiexec or a similar launcher on the apptainer command itself, when you execute an Apptainer container with an MPI code. The MPI process outside of the container will then work in tandem with MPI inside the container and the containerized MPI code to instantiate the job.

Apptainer supports two main open-source implementations of MPI: OpenMPI and MPICH. This article provides information primarily on using OpenMPI 3.x and 4.x to run on some NAS systems, for example, Cabeus, Aitken, and Electra.

How OpenMPI Interacts with Apptainer in Hybrid Mode

The workflow described on the Apptainer website is duplicated here:

1. The MPI launcher (such as mpirun, mpiexec) is called by the resource manager or the user directly from a shell.
2. OpenMPI then calls the process management daemon (orted).
3. The orted process launches the Apptainer container requested by the launcher command.
4. Apptainer instantiates the container and namespace environment.
5. Apptainer then launches the MPI application within the container.
6. The MPI application launches and loads the OpenMPI libraries.
7. The OpenMPI libraries connect back to the orted process via the Process Management Interface (PMI).

At this point, the processes within the container run as they would normally, directly on the host.

Important: The MPI in the container must be compatible with the version of MPI available on the host. For best performance, the container's MPI should be configured for optimal use of the system hardware. The standard approach is to build a matching OpenMPI version on the host, as described in the next section.

Build Your Own OpenMPI on the Host

NAS systems typically do not provide an OpenMPI version compatible with your container. To build a compatible OpenMPI on the host, determine the container's MPI version and build configuration by running the ompi_info command inside the container. The following example shows OpenMPI version (4.1.2) and the configure options used inside a container.

Note: If your container includes an OpenMPI with version 5.x, you will need a compatible OpenMPI 5.x on the host.

```
nas_host% apptainer shell your_container.sif (or  your_container_sandbox )
Apptainer> ompi_info
                Open MPI: 4.1.2
...
  Configure command line: '--enable-orterun-prefix-by-default' '--with-verbs'
                          '--prefix=/usr/local/mpi'
...
```

To build your own OpenMPI on the host:

1. Download an OpenMPI software version that is the same as, or newer than, the version used inside the container.
2. Install the software in your /nobackup filesystem by using --prefix=/nobackup*X*/*username*/...

   Note: NAS users do not have permission to install OpenMPI in any system location, including the /usr/local directory.

3. Configure the OpenMPI with options similar to those used inside the container.

Notes:

- The --with-verbs option is typically unnecessary, because OpenMPI automatically detects and supports InfiniBand (IB) on NAS systems. (The --with-verbs option builds support for OpenFabrics verbs, previously known as "Open IB".)
- Do not include the --with-pmi option (where PMI means Process Management Interface) when building on NAS systems because PMI supports launching jobs with SLURM rather than PBS.

Test the Hybrid MPI Mode with a Simple MPI Code

Before running your MPI application in hybrid mode, compile a HelloWorldMPI code using the compiler and MPI library inside the container, and run it in hybrid mode. If the HelloWorldMPI code does not run properly, your MPI application won't either.

Note: If the apptainer exec -e option is used, it likely would result in every MPI rank acting as rank 0, as demonstrated in the following example, which uses a total of four MPI processes (from two nodes with two MPI processes on each node):

```
mpirun -np 4 -H r101i3n12:2,r101i3n13:2 \
apptainer exec -e your_container_sandbox $HOME/hello_world_mpi_built_inside_container
or
mpirun -np 4 -N 2 \
apptainer exec -e your_container_sandbox $HOME/hello_world_mpi_built_inside_container
```

```
Hello world from processor r101i3n12, rank 0 out of 1 processors
Hello world from processor r101i3n13, rank 0 out of 1 processors
Hello world from processor r101i3n12, rank 0 out of 1 processors
Hello world from processor r101i3n13, rank 0 out of 1 processors
```

Depending on the required setup—for example, removing the `-e` option and adding some MPI-related environment variables—a successful HelloWorldMPI run should produce:

```
Hello world from processor r101i3n12, rank 0 out of 4 processors
Hello world from processor r101i3n13, rank 2 out of 4 processors
Hello world from processor r101i3n12, rank 1 out of 4 processors
Hello world from processor r101i3n13, rank 3 out of 4 processors
```

Run the Application with OpenMPI in Hybrid Mode

The following is a more elaborate example showing how to run applications built with OpenMPI in hybrid MPI mode. A total of four MPI processes is used (from two compute nodes with two MPI processes on each node). The OpenMPI inside the container is installed in the /usr/local/mpi directory.

```
mpiexec -np 4 -H r101i3n12:2,r101i3n13:2 --mca btl vader,openib,self \
apptainer exec -B /nasa:/nasa -B /nobackupp[X]/[username]:/nobackupp[X]/[username] \
--env OPAL_PREFIX="/usr/local/mpi" --env PMIX_INSTALL_PREFIX="/usr/local/mpi" \
[your_container_sandbox] a.out
or
mpiexec -np 4 -N 2 --mca btl vader,openib,self \
apptainer exec -B /nasa:/nasa -B /nobackupp[X]/[username]:/nobackupp[X]/[username] \
--env OPAL_PREFIX="/usr/local/mpi" --env PMIX_INSTALL_PREFIX="/usr/local/mpi" \
[your_container_sandbox] a.out
```

- The `-H` option is used to properly list the compute nodes to invoke processes on (you will need to find the hostnames of these nodes, such as r101i3n12 and r101i3n13). Without the `-H` option, all of the processes would run on the first node, possibly oversubscribing the node.
- The `-N` option specifies the number of processes per host.
- If the host and container OpenMPI libraries were built to support `openib`, use `--mca btl vader,openib,self` to explicitly use the `vader`, `openib`, and `self` byte transfer layer (BTL) components of the modular component architecture (MCA) to transfer data between two processes.

  Read the next section to learn more about the BTL components.

- The `-B` option is used to bind mount the NAS /nobackup filesystem and the /nasa directory, which is necessary if any required software is installed in those locations (such as GCC and OpenMPI on the host).
- The `apptainer exec --env` option is used to properly set the OPAL_PREFIX and PMIX_INSTALL_PREFIX environment variables to the MPI installation inside the container and ensure they are seen by every MPI process on every node. This option may be needed to resolve conflicts between the host and the container.

  Alternatively, instead of using the `--env` option, you can define the variables APPTAINERENV_OPAL_PREFIX and APPTAINERENV_PMIX_INSTALL_PREFIX before invoking `mpiexec`:

  ```
  with bash:
  export APPTAINERENV_OPAL_PREFIX=/usr/local/mpi
  export APPTAINERENV_PMIX_INSTALL_PREFIX=/usr/local/mpi

  with csh:
  setenv APPTAINERENV_OPAL_PREFIX /usr/local/mpi
  setenv APPTAINERENV_PMIX_INSTALL_PREFIX /usr/local/mpi

  mpiexec -np 4 ...
  ```

  Note: On the host environment, the OPAL_PREFIX and PMIX_INSTALL_PREFIX variables still need to be set to the MPI installation on the host in order for `mpiexec` to work properly.

For GPUs: Read this article if you plan to run containers on NVIDIA GPUs.

More Information About the OpenMPI MCA BTL

You can see which BTL components are supported in your OpenMPI installation by running the `ompi_info` command. For example:

```
Apptainer> ompi_info | grep 'MCA btl'
                 MCA btl: self (MCA v2.1.0, API v3.1.0, Component v4.1.2)
                 MCA btl: openib (MCA v2.1.0, API v3.1.0, Component v4.1.2)
                 MCA btl: tcp (MCA v2.1.0, API v3.1.0, Component v4.1.2)
                 MCA btl: vader (MCA v2.1.0, API v3.1.0, Component v4.1.2)
```

These components are described below:

- The self component is used for loopback communication (i.e., when an MPI process sends to itself).
- The openib component uses OpenFabrics Alliance verbs to support InfiniBand, RDMA over Converged Ethernet (RoCE), and RDMA over Internet Protocol networks (iWARP).
- The tcp component uses TCP-based communications over IP interfaces and networks.

- The vader component uses shared memory.

You typically don't need to specify BTL components on the mpirun command line, because OpenMPI automatically selects the best option for each communication. Manual selection is only useful when you want to use a specific BTL, or to override a default such as ^openib (which disables OpenIB) in your OpenMPI modulefile and you want to be sure to useopenib to prevent slow TCP communication or possible timeouts in some containerized applications.

Notes:

- If you are specifying BTL components explicitly, be sure to includeself. If the self component is not specified, OpenMPI may be unable to complete send-to-self scenarios.
- If using OpenMPI 5.x, the available --mca BTL components will be different. For example, vader is renamed to sm and openib is removed. You may also need to specify other --mca frameworks such as pml (point-to-point messaging layer), etc.

Check if You are Using a NAS-Instrumented mpirun or mpiexec

If you are using an NAS-built MPI modulefile on the host, thempirun and mpiexec commands may be instrumented. The originals are available as mpirun.real and mpiexec.real.

If the instrumented mpirun and mpiexec on the host fail to run the container, try using mpirun.real or mpiexec.real instead.

TIP: SLURM vs. PBS: Because many online container examples are written for SLURM, you may need to adjust their usage instructions to work correctly under PBS

References

- Apptainer and MPI Applications
- OpenMPI Frequently Asked Questions (for OpebMPI 4.x and earlier)