Hewlett Packard Enterprise

HPE Cray Supercomputing User Services Software User Guide: CSM and HPCM on HPE Cray Supercomputing EX Systems (1.3.1) (S-8065)

Part Number: S-8065
Published: June 2025

HPE Cray Supercomputing User Services Software User Guide: CSM and HPCM on HPE Cray Supercomputing EX Systems (1.3.1) (S-8065)

Part Number: S-8065
Published: June 2025

Table of contents

Copyright and Version

© Copyright 2022-2025 Hewlett Packard Enterprise Development LP. All third-party marks are the property of their respective owners.

USS: 1.3.1-LocalBuild

Doc git hash: 6811163d2c8f7f0996c37783313da9b7e6f982b4

Generated: Thu Jun 26 2025

Source Availability

This product includes code licensed under certain open source licenses which require source compliance. The corresponding source for these components is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett Packard Enterprise Company. To obtain such source code, please check if the code is available in the HPE Software Center at **https://myenterpriselicense.hpe.com/cwp-ui/software** but, if not, contact your HPE service representative.

About the USS User Guide

This document provides usage information and relevant examples for the User Services Software (USS). It is intended for software developers, engineers, scientists, and other users of USS.

Release Information

This publication supports the following software, which is included during the installation process and can be manually configured later by a system administrator:

- Containers on compute nodes (COCN)
  - Podman
  - SingularityCE
- PBS Professional 2024.1.1
- Slurm 24.05.4

HPE supports this software only if it is installed on HPE Cray Supercomputing EX Systems with:

- CSM 1.6 and COS 25.3.x (based on SLES 15 SP6)
- HPCM 1.13 and one of the following operating systems:
  - SLES 15 SP6
  - RHEL 9.4 or RHEL 9.5
  - COS 25.3.x (based on SLES 15 SP6)

For CPE customers using the WLM support ISO, the following operating systems are supported:

- RHEL 9.4 (x86_64, aarch64, or a combination of both)
- RHEL 8.10 (x86_64 only)
- SLES 15 SP6

Example Software Commands

Examples in this document include specific software commands. However, it is beyond the scope of this guide to provide complete details for third-party software. Examples are provided with the caveat that they may be out of sync with changes made by the third-party vendor. For detailed information and documentation on this software, go to:

- Kubernetes - **https://kubernetes.io/docs/home**
- PBS Professional **https://community.altair.com/community?id=altair_product_documentation**
- Podman - **https://docs.podman.io/en/latest**
- SingularityCE - **https://sylabs.io/docs**
- Slurm - **https://slurm.schedmd.com**

Some examples in this document use user@hostname to refer to the application node, which can be a User Access Node (UAN), User Access Instance (UAI), or login node depending on the system. On a system running Cray System Management (CSM), an application node is a UAN or UAI. On a system running HPE Performance Cluster Manager (HPCM) software, an application node is a login node.

Containers on Compute Nodes (COCN) Usage

Subtopics

Introduction to COCN

The HPE Cray Supercomputing EX system can run HPC applications using containers on compute nodes (COCN). Containers are a tool that enables an application to be packaged with required libraries and environment settings. This allows user software to be isolated from the runtime platform in a way that improves portability and security. This section is primarily focused on Message Passing Interface (MPI) applications, and it describes how to build and run containerized applications using Cray MPI.

Prerequisites for Creating Containers

To run containers on the compute nodes of an HPE Cray Supercomputing EX system, users must be signed into the application node. On a system running Cray System Management (CSM), an application node is a User Access Node (UAN) or User Access Instance (UAI). On a system running HPE Performance Cluster Manager (HPCM) software, an application node is a login node.

Container Software

This publication describes running the following container software on HPE Cray Supercomputing EX systems:

- **Podman**

  Podman is a daemonless, open-source container engine that finds, shares, builds, runs, and deploys applications using Open Containers Initiative (OCI) containers and container images.

- **Apptainer**

  Apptainer is a free and open-source container platform that allows you to create and run applications in containers in a simple, portable, fast, and secure manner.

- **SingularityCE**

  SingularityCE is an open-source container platform that facilitates the creation of container images on one machine and their execution across cloud and HPC clusters. It is commonly used to execute AI and machine learning workloads on HPC cluster computers.

Build and Run an MPI Application Using Podman

This section describes how to build and run a Cray MPI application as a rootless user with Podman. The following procedure uses an x86_64 system installed with COS Base (based on SLES), and the example container image that is created uses a SLES base container image. On a system installed with RHEL, use the same commands listed in this procedure but use a base container image from RHEL instead. For more information on SLES base container images, refer to SUSE's Container Guide. For more information on RHEL base container images, refer to Red Hat's EcoSystem Catalog.

Subtopics

Prerequisites for Building and Running an MPI Application Using Podman

- Each compute node that is used to run the MPI application must have access to the container image.

  The container image built by Podman uses an overlay file system, and rootless Podman users use the fuse-overlayfs storage driver to access the storage. Lustre and other distributed file systems, such as Network File System (NFS), are not supported when running in rootless mode because these file systems do not understand user namespaces.

  To provide access to a shared container image on shared network storage from the compute nodes, a copy of the image is made to the network storage in a compressed, read-only SquashFS format. First, the container image is built from the application node on local storage in the OverlayFS format. Then, the image is migrated to shared network storage in the SquashFS format using the m2s utility. Finally, the MPI application is run on the compute nodes using a workload manager with Podman by using a command-line argument to specify the location of the shared image.

- COCN has been installed on an application node and compute nodes.

Prepare the MPI Application for Podman

Create and Build an MPI Application on the Application Node

1. SSH into the application node as a rootless user.

   ```
   ssh user@hostname
   ```

2. Create the test program in a file called mpi_hello.c.

```
user@hostname> cat mpi_hello.c
/* MPI hello world example */
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char **argv)
{
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  system("whoami");
  printf("Hello from rank %d\n", rank);
  MPI_Finalize();
  return 0;
}
```

3. Verify the correct modules are loaded, including cray-mpich.

```
user@hostname> which cc
/opt/cray/pe/craype/<CPE_VERSION>/bin/cc
user@hostname> module list
Currently Loaded Modulefiles:
  1) craype-x86-rome                  6) craype/<CPE_VERSION>
  2) libfabric/<LIBFABRIC_VERSION>          7) cray-dsmml/<DSMML_VERSION>
  3) craype-network-ofi                  8) cray-mpich/<MPICH_VERSION>
  4) perftools-base/<PERFTOOLS_VERSION>      9) cray-libsci/<LIBSCI_VERSION>
  5) cce/<CCE_VERSION>                    10) PrgEnv-cray/<PRGENV_VERSION>
```

4. Build the MPI application.

```
user@hostname> cc mpi_hello.c -o mpi_hello.x
```

Find and Copy the Shared Libraries Needed by the MPI Application

1. Use the ldd command to list the shared libraries needed.

```
user@hostname> ldd mpi_hello.x
linux-vdso.so.1 (0x00007ffcac3eb000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f52cda4e000)
libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12 (0x00007f52cb3da000)
libquadmath.so.0 => /usr/lib64/libquadmath.so.0 (0x00007f52cb394000)
libmodules.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2
(0x00007f52cb379000)
libfi.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2
(0x00007f52cadb3000)
libcraymath.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2
(0x00007f52caccd000)
libf.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2
(0x00007f52cac39000)
libu.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2
(0x00007f52cab2d000)
libcsup.so.1 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1
(0x00007f52cab25000)
libc.so.6 => /lib64/libc.so.6 (0x00007f52ca92e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f52cda72000)
libfabric.so.1 => /opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1
```

```
(0x00007f52ca839000)
libatomic.so.1 => /usr/lib64/libatomic.so.1 (0x00007f52ca82f000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f52ca80b000)
librt.so.1 => /lib64/librt.so.1 (0x00007f52ca801000)
libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0 (0x00007f52ca7de000)
libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0 (0x00007f52ca7bb000)
libm.so.6 => /lib64/libm.so.6 (0x00007f52ca66d000)
libgfortran.so.5 => /usr/lib64/libgfortran.so.5 (0x00007f52ca399000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f52ca154000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f52ca130000)
librdmacm.so.1 => /usr/lib64/librdmacm.so.1 (0x00007f52ca10f000)
libibverbs.so.1 => /usr/lib64/libibverbs.so.1 (0x00007f52ca0ec000)
libpals.so.0 => /opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0
(0x00007f52ca0e4000)
libnl-3.so.200 => /usr/lib64/libnl-3.so.200 (0x00007f52c9e00000)
libnl-route-3.so.200 => /usr/lib64/libnl-route-3.so.200 (0x00007f52c9a00000)
```

2. Find the shared libraries needed in the container image from the output of the ldd command.

   The SLES base container image used to create the MPI application image contains the libraries that are needed by the MPI application in /usr/lib64 and /lib64. The libraries in /opt/cray are not included in the SLES base container image.

3. Copy the shared libraries needed in the container image into a subdirectory.

```
user@hostname> mkdir libs
user@hostname> cp /opt/cray/pe/lib64/libmpi_cray.so.12 libs
user@hostname> cp
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2 libs
user@hostname> cp /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2 libs
user@hostname> cp
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2 libs
user@hostname> cp /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2 libs
user@hostname> cp /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2 libs
user@hostname> cp /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1
libs
user@hostname> cp /opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1
libs
user@hostname> cp /opt/cray/pe/lib64/libpmi.so.0 libs
user@hostname> cp /opt/cray/pe/lib64/libpmi2.so.0 libs
user@hostname> cp /opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0 libs
```

Build and Run the Podman Image

Build the Container Image on Local Container Image Storage

Build the container image on the local container image storage in $PODMAN_GRAPHROOT on the application node.

1. Set the location used for local container image storage for the rootless user.

   a. Find the location of the local storage directory for container image storage for rootless users.

      The default local storage directory used is /scratch/cocn/containers. Each rootless user uses a subdirectory in the /scratch/cocn/containers directory that is automatically created by Podman.

   b. Set an environment variable for the directory chosen for container storage for the rootless user.

```
user@hostname> export
PODMAN_GRAPHROOT=/scratch/cocn/containers/$USER/storage
```

2. Create the Dockerfile.

```
user@hostname> vim Dockerfile.mpi_hello
user@hostname> cat Dockerfile.mpi_hello
FROM registry.suse.com/bci/bci-base:latest
COPY libs/* /usr/local/lib
COPY mpi_hello.x /usr/local/bin
ENV PATH=/usr/local/bin:$PATH
ENV LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
ENV LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/host/usr/lib64
```

3. Build the Podman image.

```
user@hostname> podman \
--root $PODMAN_GRAPHROOT \
build -f Dockerfile.mpi_hello -t mpi_hello:1.0 .
```

4. List the Podman images.

```
user@hostname> podman \
--root $PODMAN_GRAPHROOT \
images
REPOSITORY                 TAG       IMAGE ID      CREATED         SIZE
localhost/mpi_hello         1.0       e3bd25fc994f  17 seconds ago  218 MB
registry.suse.com/bci/bci-base  latest    8030200b6c3d  4 days ago      123 MB
```

Migrate the Container Image from Local to Shared Container Image Storage

Migrate the container image to the shared container image storage in $SHARED_CONTAINERS_STORAGE.

1. Save the path chosen for storing container images in $SHARED_CONTAINERS_STORAGE.

   a. Find the location of the shared storage directory for container image storage for rootless users.

      The container image storage must be on shared network storage that is accessible from the application node and the compute nodes where Podman will be installed and configured. For example, you can use a directory on Lustre or NFS storage.

      The example in this section uses a Lustre shared storage directory at /lus/snx11010/cocn/containers. Each rootless user uses a subdirectory in the /lus/snx11010/cocn/containers directory. The subdirectory is automatically created when the image is migrated to shared storage using the following process.

   b. Set an environment variable for the directory chosen for container storage for the rootless user.

      ```
      user@hostname> export
      SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/storage
      ```

2. Initialize the shared container image storage.

   Before each rootless user migrates an image for the first time, initialize the shared container image storage. This step is not necessary for later migrations by the same user.

   ```
   user@hostname> m2s init $SHARED_CONTAINERS_STORAGE
   ```

3. Migrate the image to the SquashFS format on shared container image storage.

   ```
   user@hostname> m2s --root $PODMAN_GRAPHROOT mig mpi_hello:1.0
   $SHARED_CONTAINERS_STORAGE
   ```

4. List the container images on local and shared storage.

   The images on shared storage have the R/O flag set to true to indicate that they are read-only. These images are accessible from the compute nodes.

   ```
   user@hostname> podman \
   --root $PODMAN_GRAPHROOT \
   --storage-opt overlay.imagestore=$SHARED_CONTAINERS_STORAGE \
   images
   ```

```
REPOSITORY                    TAG        IMAGE ID      CREATED          SIZE       R/O
localhost/mpi_hello           1.0        e3bd25fc994f  11 minutes ago  218 MB     false
localhost/mpi_hello           1.0        e3bd25fc994f  11 minutes ago  218 MB     true
registry.suse.com/bci/bci-base  latest     8030200b6c3d  4 days ago      123 MB
false
```

Run the MPI Application

1. Run the MPI application in the container on two compute nodes using Slurm.

   For more information on Slurm, see the Slurm Usage section of this guide.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following command for HPE Slingshot 100GB NIC on two compute nodes.

     ```
     user@hostname> srun --ntasks-per-node=1 --nodelist=cn01,cn02 \
     podman \
     --storage-driver overlay \
     --root /tmp/$USER/storage \
     --storage-opt overlay.ignore_chown_errors=true \
     --storage-opt overlay.mount_program=/usr/bin/fuse-overlayfs-wrap \
     --storage-opt overlay.imagestore=$SHARED_CONTAINERS_STORAGE \
     run --rm \
     --userns=keep-id \
     --net=host --pid=host --ipc=host \
     --device /dev/infiniband/rdma_cm \
     --device /dev/infiniband/uverbs0 \
     --mount type=bind,src=/etc/libibverbs.d,target=/etc/libibverbs.d \
     --mount type=bind,src=/usr/lib64,target=/host/usr/lib64 \
     --mount type=bind,src=/var/run/munge,target=/var/run/munge \
     --mount type=bind,src=/var/spool/slurmd,target=/var/spool/slurmd \
     --env 'SLURM_*' \
     --env 'PALS_*' \
     --env 'PMI_*' \
     --env 'SLINGSHOT_*' \
     --env LD_LIBRARY_PATH=/usr/local/lib:/host/usr/lib64:/host/usr/lib64/libibverbs \
     mpi_hello:1.0 mpi_hello.x
     ```

   - Run the following command for HPE Slingshot 200GB NIC on two compute nodes.

     ```
     user@hostname> srun --ntasks-per-node=1 --nodelist=cn01,cn02 \
     podman \
     --storage-driver overlay \
     --root /tmp/$USER/storage \
     --storage-opt overlay.ignore_chown_errors=true \
     --storage-opt overlay.mount_program=/usr/bin/fuse-overlayfs-wrap \
     --storage-opt overlay.imagestore=$SHARED_CONTAINERS_STORAGE \
     run --rm \
     --userns=keep-id \
     --net=host --pid=host --ipc=host \
     --device /dev/cxi0 \
     --mount type=bind,src=/usr/lib64,target=/host/usr/lib64 \
     --mount type=bind,src=/var/run/munge,target=/var/run/munge \
     --mount type=bind,src=/var/spool/slurmd,target=/var/spool/slurmd \
     --env 'SLURM_*' \
     --env 'PALS_*' \
     --env 'PMI_*' \
     --env 'SLINGSHOT_*' \
     --env LD_LIBRARY_PATH=/usr/local/lib:/host/usr/lib64 \
     mpi_hello:1.0 mpi_hello.x
     ```

2. Run the MPI application in the container on two compute nodes using PBS.

   For more information on PBS, see the PBS Professional Usage section of this guide.

   a. Create a file with the Process Management Interface (PMI) environment variables.

      Because the PMI_CONTROL_FD environment variable should not be passed to Podman, you cannot use the --env 'PMI_*' argument with podman run. To avoid passing all of the needed PMI environment variables on the command line, you can use a file with the required variables. You must create this file on shared network storage because it must be available from the compute nodes.

      ```
      user@hostname> cat file-with-PMI-variables
      PMI_JOBID
      PMI_LOCAL_RANK
      PMI_LOCAL_SIZE
      PMI_RANK
      PMI_SHARED_SECRET
      PMI_SIZE
      PMI_UNIVERSE_SIZE
      ```

   b. Run the MPI application commands for the Slingshot version on your system.

      - Run the following commands for HPE Slingshot 100GB NIC on two compute nodes.

        ```
        user@hostname> qsub -I -l select=1:host=cn01+host=cn02,place=scatter
        user@cn01> export
        SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/storage
        user@cn01> module load cray-pals; module load cray-pmi
        user@cn01> mpiexec -n 2 \
        podman \
        --storage-driver overlay \
        --root /tmp/$USER/storage \
        --storage-opt overlay.ignore_chown_errors=true \
        --storage-opt overlay.mount_program=/usr/bin/fuse-overlayfs-wrap \
        --storage-opt overlay.imagestore=$SHARED_CONTAINERS_STORAGE \
        run --rm \
        --userns=keep-id \
        --net=host --pid=host --ipc=host \
        --device /dev/infiniband/rdma_cm \
        --device /dev/infiniband/uverbs0 \
        --mount type=bind,src=/etc/libibverbs.d,target=/etc/libibverbs.d \
        --mount type=bind,src=/usr/lib64,target=/host/usr/lib64 \
        --mount type=bind,src=/var/run/palsd,target=/var/run/palsd \
        --env 'PALS_*' \
        --env 'SLINGSHOT_*' \
        --env-file <full-path-to>/file-with-PMI-variables \
        --env LD_LIBRARY_PATH=/usr/local/lib:/host/usr/lib64:/host/usr/lib64/libibverbs \
        mpi_hello:1.0 mpi_hello.x
        ```

      - Run the following commands for HPE Slingshot 200GB NIC on two compute nodes.

        ```
        user@hostname> qsub -I -l select=1:host=cn01+host=cn02,place=scatter
        user@cn01> export
        SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/storage
        user@cn01> module load cray-pals; module load cray-pmi
        user@cn01> mpiexec -n 2 \
        podman \
        --storage-driver overlay \
        --root /tmp/$USER/storage \
        --storage-opt overlay.ignore_chown_errors=true \
        --storage-opt overlay.mount_program=/usr/bin/fuse-overlayfs-wrap \
        --storage-opt overlay.imagestore=$SHARED_CONTAINERS_STORAGE \
        ```

```
run --rm \
--userns=keep-id \
--net=host --pid=host --ipc=host \
--device /dev/cxi0 \
--mount type=bind,src=/usr/lib64,target=/host/usr/lib64 \
--mount type=bind,src=/var/run/palsd,target=/var/run/palsd \
--env 'PALS_*' \
--env 'SLINGSHOT_*' \
--env-file <full-path-to>/file-with-PMI-variables \
--env LD_LIBRARY_PATH=/usr/local/lib:/host/usr/lib64 \
mpi_hello:1.0 mpi_hello.x
```

Build and Run an MPI Application Using Apptainer

This section describes how to build and run a Cray MPI application as a rootless user with Apptainer. The following procedure uses an x86_64 system installed with COS Base (based on SLES), and the example container image that is created uses a SLES base container image. On a system installed with RHEL, use the same commands listed in this procedure but use a base container image from RHEL instead. For more information on SLES base container images, refer to SUSE's **Container Guide**. For more information on RHEL base container images, refer to Red Hat's **EcoSystem Catalog**.

Subtopics

Prerequisites for Building and Running an MPI Application Using Apptainer

- Each compute node used to run the MPI application must have access to the container image.

  The container image built by Apptainer uses the Singularity Image Format (SIF), which is a read-only SquashFS format. The SIF container images are supported on shared network storage provided by distributed file systems, such as NFS and Lustre.

  The SIF container image must be built from the application node on shared network storage accessible from the compute nodes. The MPI application is run on the compute nodes using a workload manager with Apptainer by using a command-line argument to specify the location of the shared image.

- Apptainer has been installed on an application node and compute nodes.

Prepare the MPI Application for Apptainer

Create and Build an MPI Application on the Application Node

1. SSH into the application node as a rootless user.

   ```
   ssh user@hostname
   ```

2. Create the test program in a file called mpi_hello.c.

   ```
   user@hostname> cat mpi_hello.c
   /* MPI hello world example */
   #include <stdio.h>
   #include <stdlib.h>
   #include <mpi.h>
   int main(int argc, char **argv)
   ```

```
{
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  system("whoami");
  printf("Hello from rank %d\n", rank);
  MPI_Finalize();
  return 0;
}
```

3. Verify the correct modules are loaded, including cray-mpich.

```
user@hostname> which cc
/opt/cray/pe/craype/<CPE_VERSION>/bin/cc
user@hostname> module list
Currently Loaded Modulefiles:
  1) craype-x86-rome                    6) craype/<CPE_VERSION>
  2) libfabric/<LIBFABRIC_VERSION>          7) cray-dsmml/<DSMML_VERSION>
  3) craype-network-ofi                  8) cray-mpich/<MPICH_VERSION>
  4) perftools-base/<PERFTOOLS_VERSION>      9) cray-libsci/<LIBSCI_VERSION>
  5) cce/<CCE_VERSION>                      10) PrgEnv-cray/<PRGENV_VERSION>
```

4. Build the MPI application.

```
user@hostname> cc mpi_hello.c -o mpi_hello.x
```

Find the Shared Libraries Needed by the MPI Application

1. Use ldd command to list the shared libraries needed.

```
user@hostname> ldd mpi_hello.x
linux-vdso.so.1 (0x00007ffcac3eb000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f52cda4e000)
libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12 (0x00007f52cb3da000)
libquadmath.so.0 => /usr/lib64/libquadmath.so.0 (0x00007f52cb394000)
libmodules.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2
(0x00007f52cb379000)
libfi.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2
(0x00007f52cadb3000)
libcraymath.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2
(0x00007f52caccd000)
libf.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2
(0x00007f52cac39000)
libu.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2
(0x00007f52cab2d000)
libcsup.so.1 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1
(0x00007f52cab25000)
libc.so.6 => /lib64/libc.so.6 (0x00007f52ca92e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f52cda72000)
libfabric.so.1 => /opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1
(0x00007f52ca839000)
libatomic.so.1 => /usr/lib64/libatomic.so.1 (0x00007f52ca82f000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f52ca80b000)
librt.so.1 => /lib64/librt.so.1 (0x00007f52ca801000)
libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0 (0x00007f52ca7de000)
libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0 (0x00007f52ca7bb000)
libm.so.6 => /lib64/libm.so.6 (0x00007f52ca66d000)
libgfortran.so.5 => /usr/lib64/libgfortran.so.5 (0x00007f52ca399000)
```

```
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f52ca154000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f52ca130000)
librdmacm.so.1 => /usr/lib64/librdmacm.so.1 (0x00007f52ca10f000)
libibverbs.so.1 => /usr/lib64/libibverbs.so.1 (0x00007f52ca0ec000)
libpals.so.0 => /opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0
(0x00007f52ca0e4000)
libnl-3.so.200 => /usr/lib64/libnl-3.so.200 (0x00007f52c9e00000)
libnl-route-3.so.200 => /usr/lib64/libnl-route-3.so.200 (0x00007f52c9a00000)
```

2. Find the shared libraries needed in the container image from the output of the ldd command.

   The SLES base container image that is used to create the SIF image contains the libraries that are needed by the MPI application in /usr/lib64 and /lib64. The libraries in /opt/cray are not included in the SLES base container image.

Build and Run the Apptainer Image

Set the Location for Shared SIF Container Image Storage

Save the path chosen for storing SIF container images in $SHARED_CONTAINERS_STORAGE.

1. Find the location of the shared storage directory for container image storage for rootless users.

   The container image storage must be on shared network storage that is accessible from the application node and the compute nodes where SingularityCE will be installed and configured. For example, you can use a directory on Lustre or NFS storage.

   The example used in this section for a Lustre shared storage directory is /lus/snx11010/cocn/containers. Each rootless user uses a subdirectory in the "/lus/snx11010/cocn/containers" directory.

2. Set an environment variable for the directory that will be used for storage of SIF images for the rootless user and create the directory.

```
user@hostname> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
user@hostname> mkdir -p $SHARED_CONTAINERS_STORAGE
```

Build and Run the Shared Container Image

There are two main approaches for executing MPI applications installed in a Apptainer container. The first approach is called the hybrid model, which uses a combination of libraries provided on the host and in the container to provide the MPI implementation. The second approach is called the bind model, which only uses the MPI implementation available on the host and does not include any MPI libraries in the container.

For more information on using Apptainer and MPI applications with the hybrid and bind models, refer to Apptainer's **User Guide**.

Select either the hybrid or bind model to build and run the Apptainer image.

hybrid Model

1. Create the Apptainer definition file.

   Create the definition file that will later be used to create the Apptainer image. Copy the list of shared libraries from the previous step into the container. In the following example, the shared libraries are copied to /usr/local/lib/ in the container, but they can be copied to any directory that is available in the container. The directory containing the shared libraries must be included in LD_LIBRARY_PATH.

```
user@hostname> vim mpi-hybrid_hello.def
user@hostname> cat mpi-hybrid_hello.def
Bootstrap: docker
From: registry.suse.com/bci/bci-base:latest
%files
/opt/cray/pe/lib64/libmpi_cray.so.12 /usr/local/lib/
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2 /usr/local/lib/
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2 /usr/local/lib/
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2 /usr/local/lib/
```

```
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2 /usr/local/lib/
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2 /usr/local/lib/
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1 /usr/local/lib/
/opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1 /usr/local/lib/
/opt/cray/pe/lib64/libpmi.so.0 /usr/local/lib/
/opt/cray/pe/lib64/libpmi2.so.0 /usr/local/lib/
/opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0 /usr/local/lib/
/path/to/app/mpi_hello.x /usr/local/bin/
%environment
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/host/usr/lib64:/host/usr/lib64/libibverbs
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib64:/host/lib64
```

2. Build the Apptainer image.

```
user@hostname> apptainer build --fakeroot \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif mpi-hybrid_hello.def
```

3. Run the MPI application in the container on the application node.

```
user@hostname> apptainer exec --fakeroot \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

4. Run the MPI application in the container on two compute nodes using Slurm.

   For more information on Slurm, see the Slurm Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following command for HPE Slingshot 100GB NIC on two compute nodes.

```
user@hostname> srun -N2 apptainer exec --fakeroot \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/spool/slurmd \
--bind /var/run/munge \
--bind /etc/libibverbs.d \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

   - Run the following command for HPE Slingshot 200GB NIC on two compute nodes.

```
user@hostname> srun -N2 apptainer exec --fakeroot \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/spool/slurmd \
--bind /var/run/munge \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

5. Run the MPI application in the container on two compute nodes using PBS.

   For more information on PBS, see the PBS Professional Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following commands for HPE Slingshot 100GB NIC on two compute nodes.

```
user@hostname> qsub -I -l select=2,place=scatter
```

```
user@cn> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/apptainer
user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
user@cn> mpiexec -n2 apptainer exec --fakeroot \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/run/palsd \
--bind /etc/libibverbs.d \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

- Run the following commands for HPE Slingshot 200GB NIC on two compute nodes.

```
user@hostname> qsub -I -l select=2,place=scatter
user@cn> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/apptainer
user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
user@cn> mpiexec -n2 apptainer exec --fakeroot \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/run/palsd \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

bind Model

1. Create the Apptainer definition file.

   Create the definition file that will later be used to create the Apptainer image. The shared libraries required by the MPI application are not copied into the container and will need to be bind mounted from the host. The Cray shared libraries are available on the host in /opt/cray/ and must be available in the container using $LD_LIBRARY_PATH.

```
user@hostname> vim mpi-bind_hello.def
user@hostname> cat mpi-bind_hello.def
Bootstrap: docker
From: registry.suse.com/bci/bci-base:latest
%files
/path/to/app/mpi_hello.x /usr/local/bin/
%environment
export LD_LIBRARY_PATH=/opt/cray/pe/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cray/pals/<PALS_VERSION>/lib
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cray/libfabric/<LIBFABRIC_VERSION>/lib
64
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/host/usr/lib64:/host/usr/lib64/libibverbs
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib64:/host/lib64
```

2. Build the Apptainer image.

```
user@hostname> apptainer build --fakeroot \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif mpi-bind_hello.def
```

3. Run the MPI application in the container on the application node.

   The /opt/cray directory is mounted into the container from the node.

```
user@hostname> apptainer exec --fakeroot \
--bind /opt/cray \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
```

4. Run the MPI application in the container on two compute nodes using Slurm.

For more information on Slurm, see the Slurm Usage section of this guide.

The --bind argument is used to specify the directories that are mounted into the container from the nodes.

Be sure to run the MPI application commands for the Slingshot version on your system.

- Run the following command for HPE Slingshot 100GB NIC on two compute nodes.

  ```
  user@hostname> srun -N2 apptainer exec --fakeroot \
  --bind /opt/cray \
  --bind /usr/lib64:/host/usr/lib64 \
  --bind /lib64:/host/lib64 \
  --bind /var/spool/slurmd \
  --bind /var/run/munge \
  --bind /etc/libibverbs.d \
  $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
  ```

- Run the following command for HPE Slingshot 200GB NIC on two compute nodes.

  ```
  user@hostname> srun -N2 apptainer exec --fakeroot \
  --bind /opt/cray \
  --bind /usr/lib64:/host/usr/lib64 \
  --bind /lib64:/host/lib64 \
  --bind /var/spool/slurmd \
  --bind /var/run/munge \
  $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
  ```

5. Run the MPI application in the container on two compute nodes using PBS.

   For more information on PBS, see the PBS Professional Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following commands for HPE Slingshot 100GB NIC on two compute nodes.

     ```
     user@hostname> qsub -I -l select=2,place=scatter
     user@cn> export
     SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/apptainer
     user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
     user@cn> mpiexec -n2 apptainer exec --fakeroot \
     --bind /opt/cray \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     --bind /var/run/palsd \
     --bind /etc/libibverbs.d \
     $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
     ```

   - Run the following commands for HPE Slingshot 200GB NIC on two compute nodes.

     ```
     user@hostname> qsub -I -l select=2,place=scatter
     user@cn> export
     SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/apptainer
     user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
     user@cn> mpiexec -n2 apptainer exec --fakeroot \
     --bind /opt/cray \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     --bind /var/run/palsd \
     --bind /etc/libibverbs.d \
     $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
     ```

Build and Run an MPI Application Using SingularityCE

This section describes how to build and run a Cray MPI application as a rootless user with SingularityCE. The following procedure uses an x86_64 system installed with COS Base (based on SLES), and the example container image that is created uses a SLES base container image. On a system installed with RHEL, use the same commands listed in this procedure but use a base container image from RHEL instead. For more information on SLES base container images, refer to SUSE's **Container Guide**. For more information on RHEL base container images, refer to Red Hat's **EcoSystem Catalog**.

Subtopics

Prerequisites for Building and Running an MPI Application Using SingularityCE

- Each compute node that is used to run the MPI application must have access to the container image.

  The container image built by SingularityCE uses the Singularity Image Format (SIF), which is a read-only SquashFS format. The SIF container images are supported on shared network storage provided by distributed file systems, such as NFS and Lustre.

  The SIF container image must be built from the application node on shared network storage accessible from the compute nodes. The MPI application is run on the compute nodes using a workload manager with SingularityCE by using a command-line argument to specify the location of the shared image.

- SingularityCE has been installed on an application node and compute nodes.

Prepare the MPI Application for SingularityCE

Create and Build an MPI Application on the Application Node

1. SSH into the application node as a rootless user.

   ```
   ssh user@hostname
   ```

2. Create the test program in a file called mpi_hello.c.

   ```
   user@hostname> cat mpi_hello.c
   /* MPI hello world example */
   #include <stdio.h>
   #include <stdlib.h>
   #include <mpi.h>
   int main(int argc, char **argv)
   {
     int rank;
     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     system("whoami");
     printf("Hello from rank %d\n", rank);
     MPI_Finalize();
     return 0;
   }
   ```

3. Verify the correct modules are loaded, including cray-mpich.

```
user@hostname> which cc
/opt/cray/pe/craype/<CPE_VERSION>/bin/cc
user@hostname> module list
Currently Loaded Modulefiles:
  1) craype-x86-rome                    6) craype/<CPE_VERSION>
  2) libfabric/<LIBFABRIC_VERSION>           7) cray-dsmml/<DSMML_VERSION>
  3) craype-network-ofi                    8) cray-mpich/<MPICH_VERSION>
  4) perftools-base/<PERFTOOLS_VERSION>      9) cray-libsci/<LIBSCI_VERSION>
  5) cce/<CCE_VERSION>                      10) PrgEnv-cray/<PRGENV_VERSION>
```

4. Build the MPI application.

```
user@hostname> cc mpi_hello.c -o mpi_hello.x
```

Find the Shared Libraries Needed by the MPI Application

1. Use ldd command to list the shared libraries needed.

```
user@hostname> ldd mpi_hello.x
linux-vdso.so.1 (0x00007ffcac3eb000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f52cda4e000)
libmpi_cray.so.12 => /opt/cray/pe/lib64/libmpi_cray.so.12 (0x00007f52cb3da000)
libquadmath.so.0 => /usr/lib64/libquadmath.so.0 (0x00007f52cb394000)
libmodules.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2
(0x00007f52cb379000)
libfi.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2
(0x00007f52cadb3000)
libcraymath.so.2 =>
/opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2
(0x00007f52caccd000)
libf.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2
(0x00007f52cac39000)
libu.so.2 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2
(0x00007f52cab2d000)
libcsup.so.1 => /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1
(0x00007f52cab25000)
libc.so.6 => /lib64/libc.so.6 (0x00007f52ca92e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f52cda72000)
libfabric.so.1 => /opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1
(0x00007f52ca839000)
libatomic.so.1 => /usr/lib64/libatomic.so.1 (0x00007f52ca82f000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f52ca80b000)
librt.so.1 => /lib64/librt.so.1 (0x00007f52ca801000)
libpmi.so.0 => /opt/cray/pe/lib64/libpmi.so.0 (0x00007f52ca7de000)
libpmi2.so.0 => /opt/cray/pe/lib64/libpmi2.so.0 (0x00007f52ca7bb000)
libm.so.6 => /lib64/libm.so.6 (0x00007f52ca66d000)
libgfortran.so.5 => /usr/lib64/libgfortran.so.5 (0x00007f52ca399000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007f52ca154000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f52ca130000)
librdmacm.so.1 => /usr/lib64/librdmacm.so.1 (0x00007f52ca10f000)
libibverbs.so.1 => /usr/lib64/libibverbs.so.1 (0x00007f52ca0ec000)
libpals.so.0 => /opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0
(0x00007f52ca0e4000)
libnl-3.so.200 => /usr/lib64/libnl-3.so.200 (0x00007f52c9e00000)
libnl-route-3.so.200 => /usr/lib64/libnl-route-3.so.200 (0x00007f52c9a00000)
```

2. Find the shared libraries needed in the container image from the output of the ldd command.

   The SLES base container image that is used to create the SIF image contains the libraries that are needed by the MPI application in

/usr/lib64 and /lib64. The libraries in /opt/cray are not included in the SLES base container image.

Build and Run the SingularityCE Image

Set the Location for Shared SIF Container Image Storage

Set the location for shared container image storage that will be used for SIF images in $SHARED_CONTAINERS_STORAGE.

1. Find the location of the shared storage directory for container image storage for rootless users.

   The container image storage must be on shared network storage that is accessible from the application node and the compute nodes where SingularityCE will be installed and configured. For example, you can use a directory on Lustre or NFS storage.

   The example in this section uses a Lustre shared storage directory at /lus/snx11010/cocn/containers. Each rootless user uses a subdirectory in the "/lus/snx11010/cocn/containers" directory.

2. Set an environment variable for the directory chosen for storage for SIF images for the rootless user and create the directory.

   ```
   user@hostname> export
   SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
   user@hostname> mkdir -p $SHARED_CONTAINERS_STORAGE
   ```

Build and Run the Shared Container Image

There are two main approaches for executing MPI applications installed in a SingularityCE container. The first approach is called the hybrid model, which uses a combination of libraries provided on the host and in the container to provide the MPI implementation. The second approach is called the bind model, which only uses the MPI implementation available on the host and does not include any MPI libraries in the container.

For more information on using SingularityCE and MPI applications with the hybrid and bind models, refer to Sylabs's **User Guide**.

Select either the hybrid or bind model to build and run the SingularityCE image.

hybrid Model

1. Create the SingularityCE definition file.

   Create the definition file that will later be used to create the SingularityCE image. Copy the list of shared libraries found in the previous step into the container. In the following example, the shared libraries are copied to /usr/local/lib/ in the container, but they can be copied to any directory that is available in the container. The directory containing the shared libraries must be included in LD_LIBRARY_PATH.

   ```
   user@hostname> vim mpi-hybrid_hello.def
   user@hostname> cat mpi-hybrid_hello.def
   Bootstrap: docker
   From: registry.suse.com/bci/bci-base:latest
   %files
   /opt/cray/pe/lib64/libmpi_cray.so.12 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libmodules.so.2 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libfi.so.2 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcraymath.so.2 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libf.so.2 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libu.so.2 /usr/local/lib/
   /opt/cray/pe/cce/<CCE_VERSION>/cce/x86_64/lib/libcsup.so.1 /usr/local/lib/
   /opt/cray/libfabric/<LIBFABRIC_VERSION>/lib64/libfabric.so.1 /usr/local/lib/
   /opt/cray/pe/lib64/libpmi.so.0 /usr/local/lib/
   /opt/cray/pe/lib64/libpmi2.so.0 /usr/local/lib/
   /opt/cray/pals/<PALS_VERSION>/lib/libpals.so.0 /usr/local/lib/
   /path/to/app/mpi_hello.x /usr/local/bin/
   %environment
   export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
   ```

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/host/usr/lib64:/host/usr/lib64/libibverbs
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib64:/host/lib64
```

2. Build the SingularityCE image.

   ```
   user@hostname> singularity build --fakeroot \
   $SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif mpi-hybrid_hello.def
   ```

3. Run the MPI application in the container on the application node.

   ```
   user@hostname> singularity exec \
   --bind /usr/lib64:/host/usr/lib64 \
   --bind /lib64:/host/lib64 \
   $SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
   ```

4. Run the MPI application in the container on two compute nodes using Slurm.

   For more information on Slurm, see the Slurm Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following command for HPE Slingshot 100GB NIC on two compute nodes.

     ```
     user@hostname> srun -N2 singularity exec \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     --bind /var/spool/slurmd \
     --bind /var/run/munge \
     --bind /etc/libibverbs.d \
     $SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
     ```

   - Run the following command for HPE Slingshot 200GB NIC on two compute nodes.

     ```
     user@hostname> srun -N2 singularity exec \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     --bind /var/spool/slurmd \
     --bind /var/run/munge \
     $SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
     ```

5. Run the MPI application in the container on two compute nodes using PBS.

   For more information on PBS, see the PBS Professional Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following commands for HPE Slingshot 100GB NIC on two compute nodes.

     ```
     user@hostname> qsub -I -l select=2,place=scatter
     user@cn> export
     SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
     user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
     user@cn> mpiexec -n2 singularity exec \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     --bind /var/run/palsd \
     --bind /etc/libibverbs.d \
     $SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
     ```

   - Run the following commands for HPE Slingshot 200GB NIC on two compute nodes.

```
user@hostname> qsub -I -l select=2,place=scatter
user@cn> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
user@cn> mpiexec -n2 singularity exec \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/run/palsd \
$SHARED_CONTAINERS_STORAGE/mpi-hybrid_hello.sif /usr/local/bin/mpi_hello.x
```

bind Model

1. Create the SingularityCE definition file.

   Create the definition file that will later be used to create the SingularityCE image. The shared libraries that are required by the MPI application are not copied into the container and will need to be bind mounted from the host. The Cray shared libraries are available on the host in /opt/cray/ and must be available in the container using $LD_LIBRARY_PATH.

   ```
   user@hostname> vim mpi-bind_hello.def
   user@hostname> cat mpi-bind_hello.def
   Bootstrap: docker
   From: registry.suse.com/bci/bci-base:latest
   %files
   /path/to/app/mpi_hello.x /usr/local/bin/
   %environment
   export LD_LIBRARY_PATH=/opt/cray/pe/lib64:$LD_LIBRARY_PATH
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cray/pals/<PALS_VERSION>/lib
   export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cray/libfabric/<LIBFABRIC_VERSION>/lib
   64
   export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/host/usr/lib64:/host/usr/lib64/libibverbs
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib64:/host/lib64
   ```

2. Build the SingularityCE image.

   ```
   user@hostname> singularity build --fakeroot \
   $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif mpi-bind_hello.def
   ```

3. Run the MPI application in the container on the application node.

   The /opt/cray directory is mounted into the container from the node.

   ```
   user@hostname> singularity exec \
   --bind /opt/cray \
   --bind /usr/lib64:/host/usr/lib64 \
   --bind /lib64:/host/lib64 \
   $SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
   ```

4. Run the MPI application in the container on two compute nodes using Slurm.

   For more information on Slurm, see the Slurm Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following command for HPE Slingshot 100GB NIC on two compute nodes.

     ```
     user@hostname> srun -N2 singularity exec \
     --bind /opt/cray \
     --bind /usr/lib64:/host/usr/lib64 \
     --bind /lib64:/host/lib64 \
     ```

```
--bind /var/spool/slurmd \
--bind /var/run/munge \
--bind /etc/libibverbs.d \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
```

- Run the following command for HPE Slingshot 200GB NIC on two compute nodes.

```
user@hostname> srun -N2 singularity exec \
--bind /opt/cray \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/spool/slurmd \
--bind /var/run/munge \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
```

5. Run the MPI application in the container on two compute nodes using PBS.

   For more information on PBS, see the PBS Professional Usage section of this guide.

   The --bind argument is used to specify the directories that are mounted into the container from the nodes.

   Be sure to run the MPI application commands for the Slingshot version on your system.

   - Run the following commands for HPE Slingshot 100GB NIC on two compute nodes.

```
user@hostname> qsub -I -l select=2,place=scatter
user@cn> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
user@cn> mpiexec -n2 singularity exec \
--bind /opt/cray \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/run/palsd \
--bind /etc/libibverbs.d \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
```

   - Run the following commands for HPE Slingshot 200GB NIC on two compute nodes.

```
user@hostname> qsub -I -l select=2,place=scatter
user@cn> export
SHARED_CONTAINERS_STORAGE=/lus/snx11010/cocn/containers/$USER/singularity
user@cn> module load PrgEnv-cray; module load cray-pals; module load cray-pmi
user@cn> mpiexec -n2 singularity exec \
--bind /opt/cray \
--bind /usr/lib64:/host/usr/lib64 \
--bind /lib64:/host/lib64 \
--bind /var/run/palsd \
--bind /etc/libibverbs.d \
$SHARED_CONTAINERS_STORAGE/mpi-bind_hello.sif /usr/local/bin/mpi_hello.x
```

PBS Professional Usage

Subtopics

Parallel Application Launch Service

The Parallel Application Launch Service (PALS) serves as a launcher for third-party workload managers (WLMs) that do not provide their own launchers, enabling parallel applications to be run as a unit on multiple compute nodes while coordinating their execution and providing whole-application reporting. PALS enables the WLM to control node usage and allocation, and runs on the compute node alongside the WLM daemon.

Control HPE Slingshot Network Resources Using PBS and PALS

You can control HPE Slingshot network resource allocation using the PALS mpiexec or aprun --network option. The network option consists of a list of semicolon-separated values:

Run an Application with PBS in Batch Mode

This procedure creates a launch script and submits it as a PBS job using PALS.

Run an Application with PBS in Interactive Mode

This procedure interactively submits a job to PBS using the PALS mpiexec command.

ATOM Energy Reports

Application Test Orchestration and Management (ATOM) is an HPE Cray Supercomputing EX service that workload managers use to manage jobs and tasks. ATOM runs job and application prolog and epilog tasks. Several predefined tasks are delivered with ATOM's default configuration, but administrators can add site-specific tasks, if desired. This section explains how to to review some of the data captured by these tasks.

Use the DRC2 Feature

Some systems run applications that need to use Remote Direct Memory Access (RDMA) communication between different HPC jobs and users. Use the DRC2 feature with a workload manager to allow this communication on a system.

Parallel Application Launch Service

The Parallel Application Launch Service (PALS) serves as a launcher for third-party workload managers (WLMs) that do not provide their own launchers, enabling parallel applications to be run as a unit on multiple compute nodes while coordinating their execution and providing whole-application reporting. PALS enables the WLM to control node usage and allocation, and runs on the compute node alongside the WLM daemon.

- At this time, PALS supports only the PBS workload manager.
- PALS supports the MPIR Process Acquisition Interface, which is used by tools such as debuggers and performance analyzers to locate MPI processes that are part of an MPI job.

Applications are launched by PALS from a PBS job using the mpiexec or aprun commands. The mpiexec command is similar to the mpiexec command used with other common distributions of the MPICH or Open MPI libraries.

PMIx Support

PALS supports a subset of the PMIx interface. To enable PMIx support, use the --pmi=pmix option or set the PALS_PMI=pmix environment variable. For more information on PMIx, see the **OpenPMIx documentation**.

PALS supports the following PMIx features:

Feature	PMIx Commands
Data access and sharing	PMIx_Put, PMIx_Commit, and PMIx_Get
Process creation	PMIx_Spawn
Publish/lookup operations	PMIx_Publish, PMIx_Lookup, and PMIx_Unpublish
Synchronization	PMIx_Fence

PALS provides the following non-reserved attributes for PMIx_Get:

- SLINGSHOT_DEVICES - Comma-separated PMIX_STRING with CXI device names
- SLINGSHOT_SVC_IDS - PMIX_DATA_ARRAY of CXI service IDs (PMIX_INT)
- SLINGSHOT_VNIS - PMIX_DATA_ARRAY of VNIs the application can use (PMIX_UINT16)

PALS does not support the following PMIx features:

Feature	PMIx Commands
Connecting and disconnecting processes	PMIx_Connect and PMIx_Disconnect
Credentials	PMIx_Get_credential and PMIx_Validate_credential
Event notification	PMIx_Register_event_handler and PMIx_Notify_event
Fabric support	PMIx_Fabric_register and PMIx_Fabric_update
Job management and reporting	PMIx_Allocation_request and PMIx_Job_control
Process sets and groups	PMIx_Group_invite and PMIx_Group_join
Query operations	PMIx_Query_info and PMIx_Resolve_peers
Tools	PMIx_tool_attach_to_server, PMIx_IOF_pull, and PMIx_IOF_push

Load cray-pals Module

Tip: Before running either mpiexec or aprun, modify the cray-pals module path to use the latest PALS modules. Then, load the cray-pals module to access the PALS commands.

- On systems with TCL modules:

  ```
  user@hostname> export MODULEPATH=/opt/cray/pals/modulefiles:$MODULEPATH
  user@hostname> module load cray-pals
  ```

- On systems with Lua modules:

  ```
  user@hostname> export
  MODULEPATH=/opt/cray/pals/lmod/modulefiles/core:$MODULEPATH
  user@hostname> module load cray-pals
  ```

Launch an Application

To launch an application from a PBS job, use the mpiexec command with common options.

```
user@hostname> mpiexec -n <npes> --ppn <pes_per_node> -d <cpus_per_pe>
a.out
```

For examples, see Run an Application with PBS in Batch Mode and Run an Application with PBS in Interactive Mode. Further details are also available in the mpiexec(1) and aprun(1) man pages when the cray-pals module is loaded.

mpiexec Command

Options and arguments for mpiexec are described in detail in the mpiexec(1) man page.

Tip: Note --cpu-bind and --mem-bind keyword details.

- The --cpu-bind (CPU binding) option is formatted as [verbose,]<keyword>[:arguments], where the valid keywords are:
  - none - No CPU binding
  - numa, socket, core, thread - Bind ranks to the specified hardware
  - depth - Bind ranks to the number of threads in the argument
  - list - Bind ranks to colon-separated rangelists of CPUs
  - mask - Bind ranks to comma-separate bitmasks of CPUs
- The --mem-bind (NUMA-mode memory binding) option is formatted as [verbose,]<keyword>[:arguments], where the valid keywords are:
  - none - No memory binding
  - local - Restrict each rank to use only its own NUMA node memory
  - list - Bind ranks to colon-separated rangelists of NUMA nodes

- mask - Bind ranks to comma-separate bitmasks of NUMA nodes

Commonly Used PALS Commands

In addition to mpiexec and aprun, the following PALS commands are commonly used in the command-line interface.

Tip: The NODE options are not required if the commands are run within the same PBS job as the application.

Tip: The cray-pals module must be updated and loaded before running any of these commands. For more information, see Load cray-pals Module. For complete details, see the palstat(1), palsig(1), palscmd(1), and palscp(1) man pages.

Command	Action
palstat -n <NODE>	List the applications currently running on a node.
palstat -n <NODE> <APID>	Gather information about the application <APID>.
palsig -n <NODE> -s <SIGNAL> <APID>	Send a signal to a running application. If <SIGNAL> is not specified, the default is SIGTERM, to terminate the application. In contrast, to stop a process, enter palsig -n <NODE> -s SIGSTOP <APID>. To continue a stopped process, enter palsig -n <NODE> -s SIGCONT <APID>.
palscmd -n <NODE> <APID> <CMD> <ARGS>	Run a command alongside an application.
palscp -n <NODE> -f <FILE> <APID>	Copy a file to an application's spool directory.

PALS Environment Variables

The following environment variables are set by PALS for each application rank in addition to other application environment variables.

Environment Variable	Purpose
ALPS_APP_DEPTH	The number of threads per process
ALPS_APP_ID	Unique application identifier
ALPS_APP_PE	This process's rank index
FI_CXI_COLL_FABRIC_MGR_URL	URL for allocating Slingshot collectives
FI_CXI_COLL_JOB_ID	The workload manager job ID
FI_CXI_COLL_JOB_STEP_ID	The workload manager job-step ID
FI_CXI_COLL_MCAST_TOKEN	Security token for using Slingshot collectives
FI_CXI_HWCOLL_ADDRS_PER_JOB	Number of Slingshot multicast addresses per job
FI_CXI_HWCOLL_MIN_NODES	Number of nodes per application to use collectives
PALS_APID	Unique application identifier
PALS_APINFO	Path to libpals application information file
PALS_DEPTH	The number of threads per process
PALS_FD	File descriptor for libpals communication
PALS_LOCAL_RANKID	This process is the N'th rank on this host
PALS_LOCAL_SIZE	The total number of ranks on just this host
PALS_NODEID	This process's node index
PALS_RANKID	This process's rank index
PALS_SPOOL_DIR	Local application spool directory for transferred files
PMI_JOBID	The workload manager job ID
PMI_LOCAL_RANK	This process is the N'th rank on this host
PMI_LOCAL_SIZE	The total number of ranks on just this host
PMI_RANK	This process's rank (PMI rank reordering may overrule this)
PMI_SHARED_SECRET	Key used to secure PMI communication
PMI_SIZE	The total number of ranks
PMI_UNIVERSE_SIZE	The maximum number of ranks for this application
SLINGSHOT_DEVICES	The set of Slingshot devices configured for use
SLINGSHOT_SVC_IDS	The Slingshot service ID for each corresponding device
SLINGSHOT_TCS	The Slingshot traffic classes they can use
SLINGSHOT_VNIS	The Slingshot security tokens they can use
ZE_AFFINITY_MASK	Visibility of Intel GPUs

Important: Due to the way PALS is integrated with PBS, the job-specific temporary directory (TMPDIR) will only be created on the job's head node. This can cause application failure if it tries to create temporary files or directories. To work around this problem, add export TMPDIR=/tmp to the job script before calling aprun or mpiexec.

Control HPE Slingshot Network Resources Using PBS and PALS

You can control HPE Slingshot network resource allocation using the PALS mpiexec or aprun --network option. The network option consists of a list of semicolon-separated values:

- def_<rsrc>=<val> - Per-CPU reserved allocation for this resource
- max_<rsrc>=<val> - Maximum per-node limit for this resource

The network resources are:

Resource	Description	Default	Maximum
acs	Addressing contexts	4 per-CPU	1022 per-node
cts	Counters	1 per-CPU	2047 per-node
eqs	Event queues	2 per-CPU	2047 per-node
les	List entries	16 per-CPU	16384 per-node
ptes	Portable table entries	6 per-CPU	2048 per-node
tles	Trigger list entries	1 per-CPU	2048 per-node
tgqs	Target command queues	1 per-CPU	512 per-node
txqs	Transmit command queues	2 per-CPU	1024 per-node

For example, mpiexec --network def_txqs=4;max_eqs=1024 reserves 4 transmit command queues per CPU and limits application usage to 1024 event queues per node.

Run an Application with PBS in Batch Mode

This procedure creates a launch script and submits it as a PBS job using PALS.

Important: Because of the way PALS is integrated with PBS, the job-specific temporary directory (TMPDIR) is only created on the head node of the job. This scenario can cause application failure if it tries to create temporary files or directories. To work around this problem, add export TMPDIR=/tmp to the job script before calling aprun or mpiexec.

Prerequisites

- PBS is installed and configured on the system.
- The application is compiled.

  For more information on creating MPI applications, visit the HPE Cray Programming Environment Online Documentation website .
- The cray-pals module is updated and loaded.

  For more information, see Load cray-pals Module.

Procedure

1. Change to the directory where the application is located.

   ```
   user@hostname> cd /lus/<USERNAME>
   ```

2. Create a launch script launch.sh.

   Important: If your login shell does not match the batch script shell (for example, your login shell is tcsh, but the batch script uses bash), the module environment might not be initialized. To fix this issue, add -l to the first line of the batch script (for example, #!/bin/bash -l).

   MPI: This example launch script is specific to the "Hello World" MPI application running on four nodes.

   ```
   #!/bin/bash
   #PBS -l walltime=00:00:30
   echo start job $(date)
   export MODULEPATH=/opt/cray/pals/modulefiles:$MODULEPATH
   module load cray-pals
   echo "mpiexec hostname"
   mpiexec hostname
   echo "mpiexec -n 4 /lus/<USERNAME>/hello_mpi"
   mpiexec -n4 /lus/<USERNAME>/mpi_hello.x
   echo end job $(date)
   ```

```
exit 0
```

3. Assign permissions to the launch.sh script to ensure it is executable.

   ```
   user@hostname> chmod u+x launch.sh
   ```

4. Launch the batch script.

   ```
   user@hostname> qsub -l select=4,place=scatter launch.sh
   ```

5. Check job output.

   ```
   user@hostname> cat launch.sh.o426757
   Hello from rank 3
   Hello from rank 2
   Hello from rank 1
   Hello from rank 0
   ```

Run an Application with PBS in Interactive Mode

This procedure interactively submits a job to PBS using the PALS mpiexec command.

Prerequisites

- PBS is installed and configured on the system.
- The application is compiled.

  For more information on creating MPI applications, visit the HPE Cray Programming Environment Online Documentation website .
- The cray-pals module is updated and loaded.

  For more information, see Load cray-pals Module.

Procedure

1. Initiate an interactive session.

   ```
   user@hostname> qsub -I
   qsub: waiting for job 4071.pbs-host to start
   qsub: job 4071.pbs-host ready
   user@hostname>
   ```

2. Load the PrgEnv-cray, cray-pals, and cray-pmi modules.

   ```
   user@hostname> export MODULEPATH=/opt/cray/pals/modulefiles:$MODULEPATH
   user@hostname> module load PrgEnv-cray cray-pals cray-pmi
   ```

3. Acquire information about mpiexec.

   ```
   user@hostname> type mpiexec
   mpiexec is /opt/cray/pe/pals/<version>/bin/mpiexec
   ```

4. Change to the directory where the application is located.

   ```
   user@hostname> cd /lus/<USERNAME>
   ```

5. Run the executable MPI program.

   ```
   user@hostname> mpiexec -n4 ./mpi_hello.x
   Hello from rank 1
   Hello from rank 2
   Hello from rank 3
   ```

```
Hello from rank 0
```

For an example using UCX instead of OFI:

```
user@hostname> module swap craype-network-ofi craype-network-ucx

Inactive Modules:
  1) cray-mpich

user@hostname> module swap cray-mpich cray-mpich-ucx
user@hostname> mpiexec -n4 ./mpi_hello.x
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
```

ATOM Energy Reports

Application Test Orchestration and Management (ATOM) is an HPE Cray Supercomputing EX service that workload managers use to manage jobs and tasks. ATOM runs job and application prolog and epilog tasks. Several predefined tasks are delivered with ATOM's default configuration, but administrators can add site-specific tasks, if desired. This section explains how to to review some of the data captured by these tasks.

When the ATOM energy plugin is configured, it captures per-node energy usage data whenever applications or workload manager jobs run and writes it in JSON format to one or more files named $ATOMD_ACCT_DIR/users/$UID/jobs/JOBID/nodes/NID/energy.json.

The /opt/cray/atom/sbin/energy_post.py tool provides several options for generating reports based on the data captured by the ATOM energy plugin; see energy_post.py --help for more information. For example:

- By default, energy_post.py merges data from the individual node files into a single aggregate report file.

```
nid00001:~ # /opt/cray/atom/sbin/energy_post.py
/lus/acct/users/0/jobs/<jobid>/nodes/*/
energy.json
{"energy_used": 212, "cpu_energy_used": 42, "memory_energy_used": 90,
"nodes": 2,
"nodes_throttled": 0, "nodes_cpu_throttled": 0,
"nodes_memory_throttled": 0,
"nodes_power_capped": 0, "min_power_cap": 0, "min_power_cap_count": 0,
"max_power_cap": 0, "max_power_cap_count": 0,
"nodes_with_changed_power_cap": 0}
```

- To compare the cpu_energy values shown in the raw data to the cpu_energy_used shown in the report, specify the --verbose option to view both per-node and aggregate data.

```
nid00001:~ # /opt/cray/atom/sbin/energy_post.py --verbose \
/lus/acct/users/0/jobs/<jobid>/nodes/*/energy.json
{"xname": "x5000c1s0b1n0", "nid": "5002", "energy_used": 104,
"cpu_energy_used": 19,
"memory_energy_used": 46, "start_power_cap": 0, "stop_power_cap": 0,
"cpu_throttled": 0,
"memory_throttled": 0, "changed_power_cap": 0}
{"xname": "x5000c1s0b1n1", "nid": "5003", "energy_used": 108,
"cpu_energy_used": 23,
"memory_energy_used": 44, "start_power_cap": 0, "stop_power_cap": 0,
"cpu_throttled": 0,
"memory_throttled": 0, "changed_power_cap": 0}
{"energy_used": 212, "cpu_energy_used": 42, "memory_energy_used": 90,
"nodes": 2,
"nodes_throttled": 0, "nodes_cpu_throttled": 0,
"nodes_memory_throttled": 0,
"nodes_power_capped": 0, "min_power_cap": 0, "min_power_cap_count": 0,
"max_power_cap": 0, "max_power_cap_count": 0,
"nodes_with_changed_power_cap": 0}
```

- To generate easier-to-read output, specify the --pretty option.

```
nid00001:~ # /opt/cray/atom/sbin/energy_post.py --pretty \
/lus/acct/users/0/jobs/<jobid>/nodes/*/energy.json
{'cpu_energy_used': 42,
 'energy_used': 212,
 'max_power_cap': 0,
 'max_power_cap_count': 0,
 'memory_energy_used': 90,
 'min_power_cap': 0,
 'min_power_cap_count': 0,
 'nodes': 2,
 'nodes_cpu_throttled': 0,
 'nodes_memory_throttled': 0,
 'nodes_power_capped': 0,
 'nodes_throttled': 0,
 'nodes_with_changed_power_cap': 0}
```

The ATOM energy plugin tracks the following data:

Counter	Description
cpu_energy_used	Total energy (joules) used in the CPU energy domain
energy_used	Total energy (joules) used across all domains
max_power_cap	Maximum nonzero power cap
max_power_cap_count	Nodes with the maximum nonzero power cap
memory_energy_used	Total energy (joules) used in the memory energy domain
min_power_cap	Minimum nonzero power cap
min_power_cap_count	Nodes with the minimum nonzero power cap
nodes	Nodes in job
nodes_cpu_throttled	Nodes experiencing CPU power/thermal throttling
nodes_memory_throttled	Nodes experiencing memory power/thermal throttling
nodes_power_capped	Nodes with nonzero power cap
nodes_throttled	Nodes experiencing either CPU power/thermal throttling or memory power/thermal throttling
nodes_with_changed_power_cap	Nodes with power caps that changed during execution

Use the DRC2 Feature

Some systems run applications that need to use Remote Direct Memory Access (RDMA) communication between different HPC jobs and users. Use the DRC2 feature with a workload manager to allow this communication on a system.

1. Include the drc2.h header file.

2. Call the DRC2 functions.

3. Link the application against libdrc2 by adding -ldrc2 to the compiler command.

For an example of DRC2 usage by applications or libraries:

```
#include <assert.h>
#include <drc2.h>
#include <stdlib.h>
int
main()
{
   drc2_errcode_t rc;
   drc2_set_t rset;
   char *token;
   void *auth_key;
   size_t keylen;

   rc = drc2_token_acquire("intercommunication", 0, 0, &token);
   assert(DRC2_SUCCESS == rc);

   rc = drc2_set_create(token, 0, 1, &rset);
   assert(DRC2_SUCCESS == rc);

   // Create a key for cxi1
   rc = drc2_create_ofi_key(rset, 1, &auth_key, &keylen);
   assert(DRC2_SUCCESS == rc);

   /*
    * Use the auth_key as the the key for the OFI domain_attr and
    * ep_attr authorization keys. You need separate copies for each, so
    * make a copy of the memory for the second one.
    */

   free(auth_key);

   rc = drc2_set_destroy(rset);
   assert(DRC2_SUCCESS == rc);

   return 0;
}
```

Slurm Usage

Subtopics

Control HPE Slingshot Network Resources Using Slurm
You can control network resource allocation with Slurm on systems with certain HPE Slingshot networks. Use the scontrol command to determine if this feature is enabled on your system.
Workload Manager Plugins
Various plugins allow the workload manager to interact with other parts of the HPE Cray Supercomputing EX system.
Run an Application with Slurm in Batch Mode
This procedure creates a launch script and submits it as a job using Slurm.
Run an Application with Slurm in Interactive Mode
This procedure launches an application using the Slurm srun command.
Use the DRC2 Feature
Some systems run applications that need to use Remote Direct Memory Access (RDMA) communication between different HPC jobs and users. Use the DRC2 feature with a workload manager to allow this communication on a system.

Control HPE Slingshot Network Resources Using Slurm

You can control network resource allocation with Slurm on systems with certain HPE Slingshot networks. Use the scontrol command to determine if this feature is enabled on your system.

```
user@hostname> scontrol show config |grep SwitchType
SwitchType                  = switch/hpe_slingshot
```

If SwitchType = switch/hpe_slingshot, this feature is enabled. For more information on the switch/hpe_slingshot plugin, see WLM Plugins.

Use the srun, sbatch, or salloc command with the --network option to configure network resource allocation for your jobs. The --network option consists of a comma-separated list of network resource modifiers and values in one of the following formats:

- modifier
- modifier=<val>
- modifier_<rsrc>=<val>

For example, the following command allocates a job VNI, reserves 4 transmit command queues per CPU, reserves 128 target command queues per node, and limits application usage to 1024 event queues per node.

```
user@hostname> srun --network
job_vni,def_txqs=4,res_tgqs=128,max_eqs=1024
```

The network resource modifiers are:

Modifier	**Description**
adjust_limits	slurmd sets an upper bound on network resource reservations by taking the per-NIC maximum resource quantities and subtracting the reserved or used values (whichever are higher) for all system network services. This is the default.
no_adjust_limits	slurmd calculates network resource reservations based only upon the per-resource configuration default and the number of tasks in the application. It does not set an upper bound on reservation requests based on resource usage of already-existing system network services. Consequently, setting this means more application launches might fail based on network resource exhaustion, but if the application absolutely needs certain amounts of resources to function, this option ensures it.
def_<rsrc>=<val>	Per-CPU reserved allocation for the selected resource.
res_<rsrc>=<val>	Per-node reserved allocation for the selected resource. This overrides the per-CPU allocation.
max_<rsrc>=<val>	Maximum per-node limit for the selected resource.
depth=<depth>	Multiplier for per-CPU resource allocation. Default is the number of reserved CPUs on the node.
job_vni	Allocates a job VNI for this job.
no_vni	Does not allocate any VNIs for this job, even if it is a multi-node job.
single_node_vni	Allocates a job VNI for this job, even if it is a single-node job.
tcs=<class1>[:<class2>]	Configures traffic classes for applications. The available traffic classes are DEDICATED_ACCESS, LOW_LATENCY, BULK_DATA, and BEST_EFFORT. The traffic classes can also be specified as TC_DEDICATED_ACCESS, TC_LOW_LATENCY, TC_BULK_DATA, and TC_BEST_EFFORT. Access to traffic classes can be controlled by site policy through a Slurm job submit plugin. Therefore, the job's traffic class request might be modified or rejected.

The network resources are:

Resource	Description	Default	Maximum
acs	Addressing contexts	4 per-CPU	1022 per-node
cts	Counters	1 per-CPU	2047 per-node
eqs	Event queues	2 per-CPU	2047 per-node
les	List entries	16 per-CPU	16384 per-node
ptes	Portable table entries	6 per-CPU	2048 per-node
tles	Trigger list entries	1 per-CPU	2048 per-node
tgqs	Target command queues	1 per-CPU	512 per-node
txqs	Transmit command queues	2 per-CPU	1024 per-node

Workload Manager Plugins

Various plugins allow the workload manager to interact with other parts of the HPE Cray Supercomputing EX system.

HPE Slingshot Plugin

The Slurm switch/hpe_slingshot plugin configures HPE Slingshot NICs to control access to Virtual Network Identifiers (VNIs), traffic classes (TCs), and NIC resources. It also provides information about the NIC configuration to the end-user application through environment variables. When each job step is complete, it removes the NIC configuration on compute nodes and returns allocated VNIs to the available pool for reuse.

The environment variables set by the switch/hpe_slingshot plugin are:

Environment Variable	Description
SLINGSHOT_DEVICES	Comma-separated list of NIC devices
SLINGSHOT_SVC_IDS	Comma-separated list of CXI service IDs, one per device
SLINGSHOT_TCS	Bitmask of allowed traffic classes
SLINGSHOT_VNIS	Comma-separated list of allowed VNIs

If the hardware-accelerated collectives feature is enabled, these additional environment variables are set:

Environment Variable	Description
FI_CXI_COLL_FABRIC_MGR_URL	URL to contact the Slingshot Fabric Manager
FI_CXI_COLL_JOB_ID	Job ID
FI_CXI_COLL_JOB_STEP_ID	Job step ID
FI_CXI_COLL_MCAST_TOKEN	Authentication token for contacting the Slingshot Fabric Manager
FI_CXI_HWCOLL_ADDRS_PER_JOB	Number of multicast addresses assigned to the job
FI_CXI_HWCOLL_MIN_NODES	Minimum number of nodes required to use hardware-accelerated collectives

Cray Shasta Plugin

The Slurm mpi/cray_shasta plugin provides information to Cray PMI used to set up communication between application ranks. It performs the following tasks:

1. Creates a job step-specific directory on each compute node under the slurmd spool directory.
2. Creates an apinfo file in the job step directory containing job step layout and networking information.
3. Sets Cray PMI-specific environment variables for each task.
4. Removes the apinfo file and job step-specific directory when the job step ends.

The environment variables set by the mpi/cray_shasta plugin are:

Environment Variable	Description
PALS_APID	Application ID, set to "<jobid>.<stepid>"
PALS_APINFO	Path to the apinfo file
PALS_LOCAL_RANKID	Node local rank ID, same as Slurm's SLURM_LOCALID
PALS_NODEID	Relative node ID of the current node, same as Slurm's SLURM_NODEID
PALS_RANKID	Global rank ID, same as Slurm's SLURM_PROCID
PALS_SPOOL_DIR	Path to the job step-specific directory containing the apinfo file
PMI_CONTROL_PORT	Port used by Cray PMI for remote communication within the port range set by MpiParams
PMI_JOBID	Job ID, same as Slurm's SLURM_JOB_ID
PMI_LOCAL_RANK	Node local rank ID, same as Slurm's SLURM_LOCALID
PMI_LOCAL_SIZE	Number of tasks on this node
PMI_RANK	Global rank ID, same as Slurm's SLURM_PROCID
PMI_SHARED_SECRET	Random 64-bit integer value, used to authenticate Cray PMI connections
PMI_SIZE	Number of tasks in the job step, same as Slurm's SLURM_NTASKS
PMI_UNIVERSE_SIZE	Maximum number of tasks in this job step (is set to the same number as PMI_SIZE)

Run an Application with Slurm in Batch Mode

This procedure creates a launch script and submits it as a job using Slurm.

Prerequisites

- Slurm is installed and configured on the system.
- The application is compiled.

  For more information on creating MPI applications, visit the HPE Cray Programming Environment Online Documentation website .

Procedure

1. Log in to the application node with the connection string.

   ```
   ssh user@hostname
   ```

2. Load CPE modules.

   - **MPI:** MPI modules are loaded by default.
   - **OpenSHMEM:** If OpenSHMEM is available, load the Cray OpenSHMEM modules.

     ```
     user@hostname> module load cray-openshmemx
     ```

   - **Cray DSMML:** If DSMML is available, the Cray DSMML modules are loaded by default.

3. Determine the default MPI type for Slurm.

   When using Cray MPICH, Slurm must use the cray_shasta MPI type.

   ```
   user@hostname> scontrol show config | grep MpiDefault
   ```

```
MpiDefault                = cray_shasta
```

If the default Slurm MPI type is not cray_shasta, either add the --mpi=cray_shasta option to each srun command or set the SLURM_MPI_TYPE environment variable to cray_shasta.

```
user@hostname> export SLURM_MPI_TYPE=cray_shasta
```

4. Change to the directory where the application is located.

```
user@hostname> cd /lus/<USERNAME>
```

5. Create a launch script.

   Important: If your login shell doesn't match the batch script shell (for example, your login shell is tcsh but the batch script uses bash), the module environment may not be initialized. To fix this, add -l to the first line of the batch script (for example, #!/bin/bash -l).

   To launch the application with sbatch, add srun to the launch script.

   - **MPI:** The following example is specific to the "Hello World" MPI application running on four nodes.

     ```
     #!/bin/bash
     #SBATCH -N4
     #SBATCH --ntasks-per-node=1
     ulimit -s unlimited # in case not set by default
     srun -N4 --ntasks-per-node=1 ./mpi_hello.x
     exit 0
     ```

   - **OpenSHMEM** (CSM systems only):

     ```
     #!/bin/bash
     #SBATCH --time=5
     #SBATCH --nodes=2
     #SBATCH --tasks-per-node=4
     module load cray-openshmemx
     srun -n8 ./shmem_hello.x
     exit 0
     ```

   - **DSMML** (CSM systems only):

     ```
     #!/bin/bash
     #SBATCH --time=5
     #SBATCH --nodes=1
     #SBATCH --tasks-per-node=1
     srun -n1 ./dsmml_example.x
     exit 0
     ```

     The DSMML batch script example is specific to the sample DSMML application code. It runs on a single node with one processing element.

     For more information, visit the HPE Cray Programming Environment Online Documentation website .

6. Assign permissions to the launch.sh script to ensure it is executable.

```
user@hostname> chmod u+x launch.sh
```

7. Launch the batch script.

```
user@hostname> sbatch launch.sh
Submitted batch job 1065736
```

8. Check job output.

   - **MPI:**

     ```
     user@hostname> cat slurm-1065736.out
     Hello from rank 1
     ```

```
Hello from rank 3
Hello from rank 0
Hello from rank 2
```

- **Troubleshoot:** Add ldd to the job script to check that the correct modules are loaded.

  ```
  user@hostname> ldd ./mpi_hello.x
  ```

Run an Application with Slurm in Interactive Mode

This procedure launches an application using the Slurm srun command.

Prerequisites

- Slurm is installed and configured on the system.
- The application is compiled.

  For more information on creating MPI applications, visit the HPE Cray Programming Environment Online Documentation website .

Procedure

1. Log in to the application node with the connection string.

   ```
   ssh user@hostname
   ```

2. Load CPE modules.

   - **MPI:** MPI modules are loaded by default.
   - **OpenSHMEM:** If OpenSHMEM is available, load the Cray OpenSHMEM modules.

     ```
     user@hostname> module load cray-openshmemx
     ```

3. Determine the default MPI type for Slurm.

   When using Cray MPICH, Slurm must use the cray_shasta MPI type.

   ```
   user@hostname> scontrol show config | grep MpiDefault
   MpiDefault              = cray_shasta
   ```

   If the default Slurm MPI type is not cray_shasta, either add the --mpi=cray_shasta option to each srun command or set the SLURM_MPI_TYPE environment variable to cray_shasta.

   ```
   user@hostname> export SLURM_MPI_TYPE=cray_shasta
   ```

4. Change to the directory where the application is located.

   ```
   user@hostname> cd /lus/<USERNAME>
   ```

5. Execute the application with srun.

   - **MPI:**

     ```
     user@username> srun -N<ranks> --ntasks-per-node=<number_tasks_per_node>
     ./<app_exe>
     ```

     For an example using OFI:

     ```
     user@hostname> srun -N4 --ntasks-per-node=1 ./mpi_hello.x
     Hello from rank 1
     Hello from rank 2
     Hello from rank 3
     ```

```
Hello from rank 0
```

For an example using UCX:

```
user@hostname> module swap craype-network-ofi craype-network-ucx
user@hostname> module swap cray-mpich cray-mpich-ucx
user@hostname> srun -N4 --ntasks-per-node=1 ./mpi_hello.x
Hello from rank 1
Hello from rank 2
Hello from rank 3
Hello from rank 0
```

- **OpenSHMEM** (CSM systems only):

```
user@hostname> srun -n<ranks> ./shmem_hello.x
```

For example:

```
user@hostname> srun -n8 ./shmem_hello.x
```

DVS Usage

Subtopics

DVS Environment Variables
By default, user environment variables allow client override of mount options specified during configuration. Reference the *HPE Cray Supercomputing User Services Software Administration Guide for HPE Performance Cluster Manager Software* or the *HPE Cray Supercomputing User Services Software Administration Guide: CSM on HPE Cray Supercomputing EX Systems* for more information on the operation of DVS and available mount options.

DVS Environment Variables

By default, user environment variables allow client override of mount options specified during configuration. Reference the *HPE Cray Supercomputing User Services Software Administration Guide for HPE Performance Cluster Manager Software* or the *HPE Cray Supercomputing User Services Software Administration Guide: CSM on HPE Cray Supercomputing EX Systems* for more information on the operation of DVS and available mount options.

By default, user environment variables allow client override of options specified during configuration and are evaluated whenever a file is opened by DVS. However, if the nouserenv option is included in the options setting of the client_mount configuration setting, then user environment variables are disabled for that client mount.

The following environment variables are for use in the default case:

Cray DVS User Environment Variables

Variable Name	Options	Purpose
DVS_ATOMIC	on|off	Overrides the atomic or noatomic mount options.
DVS_BLOCKSIZE	n	A nonzero number, n overrides the blksize mount option. **Note:** Unlike most other mount option and environment variable pairs, DVS_BLOCKSIZE and blksize have subtly different spellings.
DVS_CACHE	on\\\|off	Overrides the cache or nocache mount options. Exercise caution if using this variable.
DVS_CACHE_READ_SZ	n	A positive integer, n overrides the cache_read_sz mount option.
DVS_CLOSESYNC	on|off	Overrides the closesync or noclosesync mount options. **Note:** Periodic sync functions similarly to the DVS closesync mount option, but it is more efficient and is enabled by default. Hewlett Packard Enterprise recommends not using closesync or this associated environment variable.
DVS_DATASYNC	on|off	Overrides the datasync or nodatasync mount options. **Note:** Setting DVS_DATASYNC to on can slow down an application considerably. The periodic sync feature, enabled by default, is a better way to synchronize data. See the section "Periodic Sync Promotes Data and Application Resiliency" in *HPE Cray Supercomputing User Services Software Administration Guide* for your platform for more information.
DVS_DEFEROPENS	on|off	Overrides the deferopens or nodeferopens mount options.
DVS_KILLPROCESS	on|off	Overrides the killprocess or nokillprocess mount options.
DVS_MAXNODES	n	A nonzero number, n overrides the maxnodes mount option. The specified value of maxnodes must be greater than zero and less than or equal to the number of server nodes specified on the mount, otherwise the variable has no effect.

Low Noise Mode

Some application workloads show improved performance when compute node operating system tasks (sources of "OS noise") are migrated to one or more system CPUs that are excluded from application use. Low Noise Mode configures Linux features to achieve this configuration. For more information about Low Noise Mode, see the *HPE Cray Supercomputing User Services Software Administration Guide* for your platform.

Checking Low Noise Mode Status

Use the executable /usr/sbin/lnmctl to check the status of Low Noise Mode on a node. For example, invoke the executable as:

/usr/sbin/lnmctl --status

If Low Noise Mode is configured, the command produces output similar to:

```
Mode: full
CPU: [0, 63]
```

This indicates that Low Noise Mode is active in its "full" state with CPUs 0 and 63 as the "system" CPUs.

If Low Noise Mode is not configured, the command produces output similar to `File /var/run/lnm.status not found.`

Download HPE Cray Supercomputing EX Software

To download HPE Cray Supercomputing EX software, refer to the HPE Support Center or download it directly from My HPE Software Center. The HPE Support Center contains a wealth of documentation, training videos, knowledge articles, and alerts for HPE Cray Supercomputing EX systems. It provides the most detailed information about a release as well as direct links to product firmware, software, and patches available through My HPE Software Center.

Download Software through the HPE Support Center

HPE recommends downloading software through the HPE Support Center because of the many other resources available on the website.

1. Visit the HPE Cray Supercomputing EX product page on the HPE Support Center.
2. Search for specific product info, such as the full software name or recipe name and version.

   For example, search for "Slingshot 2.1" or "Cray System Software with CSM 24.3.0."
3. Find the desired software in the search results and select it to review details.
4. Select **Obtain Software** and select **Sign in Now** when prompted.

   If a customer's Entitlement Order Number (EON) is tied to specific hardware rather than software, the software is available without providing account credentials. Access the software instead by selecting **Download Software** and skip the next step in this procedure.
5. Enter account credentials when prompted and accept the HPE License Terms.

   To download software, customers must ensure their Entitlement Order Number (EON) is active under **My Contracts & Warranties** on My HPE Software Center. If customers have trouble with the EON or are not entitled to a product, they must contact their HPE contract administrator or sales representative for assistance.
6. Choose the needed software and documentation files to download and select **curl Copy** to access the files.

   Just like the software files, the documentation files change with each release. In addition to the official documentation, valuable information for a release is often available in files that include the phrase **README** in their name. Be sure to select and review these files in detail.

   HPE recommends the **curl Copy** option, which downloads a single text file with curl commands to use on the desired system. You must run the curl commands within 24 hours of downloading them or download new commands if more than 24 hours have passed.

   To validate the security of the downloads, you can later compare the files on the desired system against the checksums provided by HPE underneath each selected download.
7. Save the text file to a central location.
8. On the system where the software will be downloaded, run a shell script to execute the text file that includes the curl commands.

   For example:

   ```
   ncn-m001# bash -x <TEXT_FILE_PATH>
   ```

   The -x option in this example tracks the download progress of each curl command in the text file.

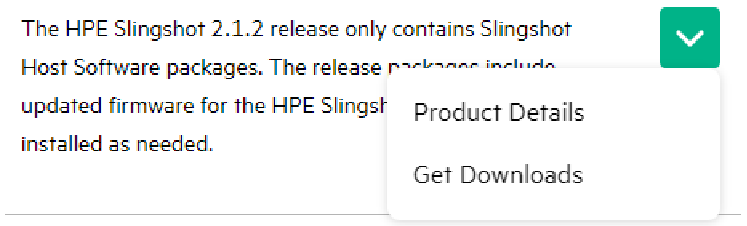
Download Software Directly from My HPE Software Center

Users already familiar with a release can save time by downloading software directly from My HPE Software Center.

1. Visit My HPE Software Center and select **Sign in**.
2. Enter account credentials when prompted and select **Software** in the left navigation bar.
3. Search for specific product info, such as the full software name or recipe name and version.

For example, search for "Slingshot 2.1" or "Cray System Software with CSM 24.3.0."

4. Find the desired software in the search results and review details by selecting **Product Details** under the **Action** column.



The HPE Slingshot 2.1.2 release only contains Slingshot Host Software packages. The release packages include updated firmware for the HPE Slingsh installed as needed.

5. Select **Go To Downloads Page** and accept the HPE License Terms.

   To download software, customers must ensure their Entitlement Order Number (EON) is active under **My Contracts & Warranties**. If customers have trouble with the EON or are not entitled to a product, they must contact their HPE contract administrator or sales representative for assistance.

6. Choose the needed software and documentation files to download and select **curl Copy** to access the files.

   Just like the software files, the documentation files change with each release. In addition to the official documentation, valuable information for a release is often available in files that include the phrase **README** in their name. Be sure to select and review these files in detail.

   HPE recommends the **curl Copy** option, which downloads a single text file with curl commands to use on the desired system. You must run the curl commands within 24 hours of downloading them or download new commands if more than 24 hours have passed.

   To validate the security of the downloads, you can later compare the files on the desired system against the checksums provided by HPE underneath each selected download.

7. Save the text file to a central location.

8. On the system where the software will be downloaded, run a shell script to execute the text file that includes the curl commands.

   For example:

   ```
   ncn-m001# bash -x <TEXT_FILE_PATH>
   ```

   The -x option in this example tracks the download progress of each curl command in the text file.

Copy and Paste Commands from this Guide

If experiencing issues copying and pasting commands from the PDF version of this guide, the following steps are recommended to ensure that the commands are copied and pasted correctly:

1. Copy a command from the PDF.

2. Paste it into a text-only tool, like a text editor.

3. Copy the command from the text-only tool and paste it into the console.

If entering commands manually, double-check them for correctness. Some commands may not render correctly in a PDF depending on the PDF viewer in use.

Alternatively, copy and paste commands from the HTML or Markdown versions of this guide.

Documentation Conventions

Several conventions have been used in the preparation of this documentation.

- Markdown Format
- File Formats
- Typographic Conventions
- Annotations for how we identify sections of the documentation that do not apply to all systems
- Command Prompt Conventions which describe the context for user, host, directory, chroot environment, or container environment

Subtopics

Markdown Format
This documentation is in Markdown format. Although much of it can be viewed with any text editor, a richer experience will come from using a tool which can render the Markdown to show different font sizes, the use of bold and italics formatting, inclusion of diagrams and screen shots as image files, and to follow navigational links within a topic file and to other files.
File Formats
Some of the installation instructions require updating files in JSON, YAML, or TOML format. These files should be updated with care since some file formats do not accept tab characters for indentation of lines. Only space characters are supported. Refer to online documentation to learn more about the syntax of JSON, YAML, and TOML files.
Typographic Conventions
This style indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.
Annotations
This repository may change annotations, for now, under the MarkDown governance these are the available annotations.
Command Prompt Conventions

Markdown Format

This documentation is in Markdown format. Although much of it can be viewed with any text editor, a richer experience will come from using a tool which can render the Markdown to show different font sizes, the use of bold and italics formatting, inclusion of diagrams and screen shots as image files, and to follow navigational links within a topic file and to other files.

There are many tools which can render the Markdown format to get these advantages. Any Internet search for Markdown tools will provide a long list of these tools. Some of the tools are better than others at displaying the images and allowing you to follow the navigational links.

HPE recommends disabling line wrapping when using the raw Markdown version of this publication. This helps with viewing the large tables in Markdown. Each publication is available in PDF, HTML, and Markdown format.

File Formats

Some of the installation instructions require updating files in JSON, YAML, or TOML format. These files should be updated with care since some file formats do not accept tab characters for indentation of lines. Only space characters are supported. Refer to online documentation to learn more about the syntax of JSON, YAML, and TOML files.

Typographic Conventions

This style indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.

\ (backslash) At the end of a command line, indicates the Linux shell line continuation character (lines joined by a backslash are parsed as a single line).

Annotations

This repository may change annotations, for now, under the MarkDown governance these are the available annotations.

You must use these to denote the right steps to the right audience.

These are context clues for steps, if they contain these, and you are not in that context you ought to skip them.

EXTERNAL USE

This tag should be used to highlight anything that an HPE Cray internal user should ignore or skip.

INTERNAL USE

This tag is used before any block of instruction or text that is only usable or recommended for internal HPE Cray systems.

External (GitHub or customer) should disregard these annotated blocks - they maybe contain useful information as an example but are not intended for their use.

Command Prompt Conventions

Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt.

- The root or super-user account always has the # character at the end of the prompt
- Any non-root account is indicated with account@hostname>. A non-privileged account is referred to as user.

Node abbreviations

The following list contains abbreviations for nodes used throughout this document.

- CN - compute Node
- NCN - Non Compute Node
- AN - Application Node (special type of NCN)
- UAN - User Access Node (special type of AN)
- PIT - Pre-Install Toolkit (initial node used as the inception node during software installation booted from the LiveCD)

Prompt	Description
ncn#	Run the command as root on any NCN, except an NCN which is functioning as an Application Node (AN), such as a UAN.
ncn-m#	Run the command as root on any NCN-M (NCN which is a Kubernetes master node).
ncn-m002#	Run the command as root on the specific NCN-M (NCN which is a Kubernetes master node) which has this hostname (ncn-m002).
ncn-w#	Run the command as root on any NCN-W (NCN which is a Kubernetes worker node).
ncn-w001#	Run the command as root on the specific NCN-W (NCN which is a Kubernetes master node) which has this hostname (ncn-w001).
ncn-s#	Run the command as root on any NCN-S (NCN which is a Utility Storage node).
ncn-s003#	Run the command as root on the specific NCN-S (NCN which is a Utility Storage node) which has this hostname (ncn-s003).
pit#	Run the command as root on the PIT node.
linux#	Run the command as root on a Linux host.
uan#	Run the command as root on any UAN.
uan01#	Run the command as root on hostname uan01.
user@uan>	Run the command as any non-root user on any UAN.
cn#	Run the command as root on any CN. Note that a CN will have a hostname of the form nid124356, that is "nid" and a six digit, zero padded number.
hostname#	Run the command as root on the specified hostname.
user@hostname>	Run the command as any non-root user on the specified hostname.

Command prompt inside chroot

If the chroot command is used, the prompt changes to indicate that it is inside a chroot environment on the system.

```
hostname# chroot /path/to/chroot
chroot-hostname#
```

Command prompt inside Kubernetes pod

If executing a shell inside a container of a Kubernetes pod where the pod name is $podName, the prompt changes to indicate that it is inside the pod. Not all shells are available within every pod, this is an example using a commonly available shell.

```
ncn# kubectl exec -it $podName /bin/sh
pod#
```

Command prompt inside image customization session

If using ssh during an image customization session, the prompt changes to indicate that it is inside the image customization environment (pod). This example uses $PORT and $HOST as environment variables with specific settings. When using chroot in this context the prompt will be different than the above chroot example.

```
hostname# ssh -p $PORT root@$HOST
root@POD# chroot /mnt/image/image-root
:/#
```

Directory path in command prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the cd command is used to change into the directory, and the directory is referenced with a period (.) to indicate the current directory.

Examples of prompts as they appear on the system:

```
hostname# cd /etc
hostname:/etc# cd /var/tmp
hostname:/var/tmp# ls ./file
hostname:/var/tmp# su - user
user@hostname:~> cd /usr/bin
user hostname:/usr/bin> ./command
```

Examples of prompts as they appear in this publication:

```
hostname# cd /etc
hostname# cd /var/tmp
hostname# ls ./file
hostname# su - user
user@hostname> cd /usr/bin
user@hostname> ./command
```

Command prompts for network switch configuration

The prompts when doing network switch configuration can vary widely depending on which vendor switch is being configured and the context of the item being configured on that switch. There may be two levels of user privilege which have different commands available and a special command to enter configuration mode.

Example of prompts as they appear in this publication:

Enter "setup" mode for the switch make and model, for example:

```
remote# ssh admin@sw-leaf-001
sw-leaf-001# enable
sw-leaf-001# configure terminal
sw-leaf-001(conf)#
```

Refer to the switch vendor OEM documentation for more information about configuring a specific switch.