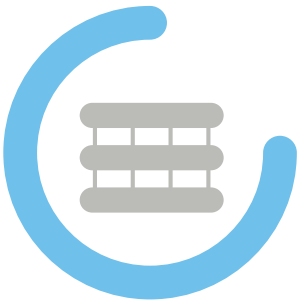

OpenHPC (v4.2)

Cluster Building Recipes

Rocky 10 Base OS

OpenCHAMI/Slurm Edition for Linux* (aarch64)



Document Last Update: 2026-11-07

Document Revision: 8da3709294470827ef75371c5af706c0feed9447

Legal Notice

Copyright © 2016-2026, OpenHPC, a Linux Foundation Collaborative Project. All rights reserved.

This documentation is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0.



Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contents

Legal Notice 2

1 Introduction 7

1.1 Target Audience . . . 7
1.2 Requirements/Assumptions . . . 7
1.3 Inputs . . . 8
1.4 Installation Template . . . 9

2 Install Base Operating System 9

2.1 Install OS . . . 9
2.2 Configure HTTPS Proxy (Optional) . . . 9
2.3 Enable EL Package Repositories . . . 10
2.4 Configure Firewall . . . 10
2.5 Configure NTP . . . 10
2.6 Generate SSH Key Pair . . . 11
2.7 Setup NFS Server . . . 11
2.8 Optionally add InfiniBand support services on *head node* . . . 11
2.9 Optionally add RoCE support on *head node* . . . 12
2.10 Optionally add Omni-Path support services on *head node* . . . 12

3 Install OpenHPC Components 13

3.1 Enable OpenHPC Repository . . . 13
3.2 Install OpenHPC Base Packages . . . 14

4 Install Slurm 14

4.1 Add Slurm on Head Node . . . 14

5 Install OpenCHAMI Provisioner 15

5.1 Install OpenCHAMI . . . 15
5.2 Certificate Creation . . . 15
5.3 Setting no proxy . . . 16
5.4 Setup DHCP . . . 16
5.5 Complete basic OpenCHAMI setup for *head node* . . . 16
5.6 s3cmd Setup . . . 18
5.7 OpenCHAMI Client Install . . . 20
5.8 Starting OpenCHAMI . . . 20
5.9 Limiting Access to cloud-init . . . 20

5.10 MPI on the Head Node . . . 22
5.11 Compute Image Configuration . . . 23
5.12 Configure Slurm Client . . . 23
5.13 Building the Base OS Image . . . 23

6 Additional Customization **29**
6.1 Enable InfiniBand drivers . . . 29
6.2 Enable RoCE drivers . . . 29
6.3 Enable Omni-Path drivers . . . 29
6.4 Increase locked memory limits . . . 29
6.5 Enable ssh control via resource manager . . . 30
6.6 NVIDIA GPU Drivers . . . 30
6.7 Enable System Log Forwarding . . . 31
6.8 Cluster Admin Tools . . . 32
6.8.1 Add ClusterShell . . . 32
6.8.2 Add genders . . . 32
6.8.3 Add ConMan . . . 33
6.8.4 Add NHC . . . 33
6.9 Add Magpie . . . 34
6.10 Charliecloud . . . 34

7 Deploy Cluster **34**
7.1 Rebuild Image . . . 34
7.2 Setting Boot Parameters . . . 35
7.3 Building cloud-init Image . . . 36
7.3.1 Create Default Configuration . . . 36
7.3.2 Configure Munge . . . 36
7.4 Optionally configure IPoIB on *compute nodes* . . . 37
7.5 Node Discovery . . . 38
7.6 Verify Configuration Before Boot . . . 40
7.7 Boot Compute Nodes . . . 41
7.8 Wait for Compute Nodes . . . 41
7.9 Start Slurm . . . 42
7.10 Start Slurm on the Compute Nodes . . . 43

8 Install OpenHPC Development Components **43**
8.1 Development Tools . . . 43
8.2 Compilers . . . 43
8.3 MPI Stacks . . . 43
8.4 Performance Tools . . . 44
8.5 Setup default development environment . . . 44
8.6 3rd Party Libraries and Tools . . . 45

9 Post Provisioning **46**
9.1 Check Cluster Status . . . 46
9.2 Add User . . . 46
9.3 Update NHC . . . 46

10 Test Cluster **46**
10.1 Run a Test Job . . . 47
10.2 Interactive Execution . . . 47
10.3 Batch execution . . . 48

Appendices **49**

A Installation Template **49**

B Upgrading OpenHPC Packages **50**

C Integration Test Suite **51**

D Customization **54**
D.1 Adding local environment modules to OpenHPC hierarchy . . . 54
D.2 Rebuilding Packages from Source . . . 55

E Package Manifest **56**
E.1 Available OpenHPC Meta-packages . . . 56
E.2 RPM Packages . . . 57
E.2.1 Administrative Tools . . . 58
E.2.2 Resource Management . . . 59
E.2.3 Compiler Families . . . 60
E.2.4 MPI Families / Communication Libraries . . . 60
E.2.5 Development Tools . . . 60
E.2.6 Performance Analysis Tools . . . 61
E.2.7 IO Libraries . . . 61

E.2.8 Distro Packages 62
E.2.9 Runtimes 62
E.2.10 Serial/Threaded Libraries 62
E.2.11 Parallel Libraries 63

F Package Signatures **64**

1 Introduction

This guide presents a simple cluster installation procedure using components from the OpenHPC software stack. OpenHPC represents an aggregation of a number of common ingredients required to deploy and manage an HPC Linux* cluster including provisioning tools, resource management, I/O clients, development tools, and a variety of scientific libraries. These packages have been pre-built with HPC integration in mind while conforming to common Linux distribution standards. The documentation herein is intended to be reasonably generic, but uses the underlying motivation of a small, 4-node cluster installation to define a step-by-step process. Several optional customizations are included and the intent is that these collective instructions can be modified as needed for local site customizations.

Base Linux Edition: this edition of the guide highlights installation without the use of a companion configuration management system and directly uses distro-provided package management tools for component selection. The steps that follow also highlight specific changes to system configuration files that are required as part of the cluster install process.

1.1 Target Audience

This guide is targeted at experienced Linux system administrators for HPC environments. Knowledge of software package management, system networking, and PXE booting is assumed. Command-line input examples are highlighted throughout this guide via the following syntax:

```
echo "OpenHPC hello world"
```

Unless specified otherwise, the examples presented are executed with elevated (root) privileges. The examples also presume use of the BASH login shell, though the equivalent commands in other shells can be substituted. In addition to specific command-line instructions called out in this guide, an alternate convention is used to highlight potentially useful tips or optional configuration options. These tips are highlighted via the following format:

Tip

Life is a tale told by an idiot, full of sound and fury signifying nothing. –Willy Shakes

1.2 Requirements/Assumptions

This installation recipe assumes the availability of a single *head node*, and four *compute* nodes. The *head node* serves as the overall system management server (SMS) and is provisioned with Rocky 10 and is subsequently configured to provision the remaining *compute* nodes with OpenCHAMI in a stateless configuration. For power management, we assume that the compute node baseboard management controllers (BMCs) are available via IPMI from the chosen *head node*. For file systems, we assume that the chosen *head node* will host an NFS file system that is made available to the compute nodes.

An outline of the physical architecture discussed is shown in the figure above and highlights the high-level networking configuration. The *head node* requires at least two Ethernet interfaces with *eth0* connected to the local data center network and *eth1* used to provision and manage the cluster backend (note that these interface names are examples and may be different depending on local settings and OS conventions). Two logical IP interfaces are expected to each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management. The second is used to connect to each host's BMC and is used for power management and remote console access. Physical connectivity for these two logical

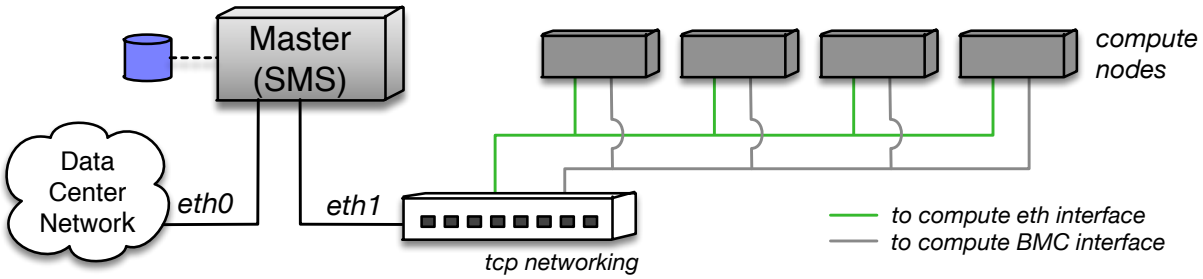

Figure 1: Overview of physical cluster architecture

IP networks is often accommodated via separate cabling and switching infrastructure; however, an alternate configuration can also be accommodated via the use of a shared NIC, which runs a packet filter to divert management packets between the host and BMC. Independent of the actual networking configuration it is recommended to have additional security boundaries like a firewall to protect the network interfaces from the Internet.

1.3 Inputs

As this recipe details installing a cluster starting from bare-metal, there is a requirement to define IP addresses and gather hardware MAC addresses in order to support a controlled provisioning process. These values are necessarily unique to the hardware being used, and this document uses variable substitution (`${variable}`) in the command-line examples that follow to highlight where local site inputs are required. A summary of the required and optional variables used throughout this recipe are presented below. Note that while the example definitions above correspond to a small 4-node compute subsystem, the compute parameters are defined in array format to accommodate logical extension to larger node counts.

Required variables:

- `${sms_name}` - Hostname for head node
- `${sms_ip}` - Internal IP address on head node
- `${sms_eth_internal}` - Internal Ethernet interface on head node
- `${internal_network}` - Subnet network address for internal network
- `${internal_netmask}` - Subnet netmask for internal network
- `${dns_domain}` - DNS domain name for cluster (e.g. `local`)
- `${ntp_server}` - Local ntp server for time synchronization
- `${bmc_username}` - BMC username for use by IPMI
- `${bmc_password}` - BMC password for use by IPMI
- `${num_computes}` - Total # of desired compute nodes
- `${c_ip[0]}`, `${c_ip[1]}`, … - Desired compute node addresses
- `${c_bmc[0]}`, `${c_bmc[1]}`, … - BMC addresses for computes
- `${c_mac[0]}`, `${c_mac[1]}`, … - MAC addresses for computes

- `${c_name[0]}`, `${c_name[1]}`, ... - Host names for computes
- `${compute_regex}` - Regex matching all compute node names (e.g. `c*`)
- `${compute_prefix}` - Prefix for compute node names (e.g. `c`)

Optional variables:

- `${modules_package}` - Module tool installed on the head node and the computes: `lmod-ohpc` (Lmod, the default) or `environment-modules-ohpc` (Environment Modules)

1.4 Installation Template

The collection of command-line instructions that follow in this guide, when combined with local site inputs, can be used to implement a bare-metal system installation and configuration. The format of these commands is intended to be usable via direct cut and paste (with variable substitution for site-specific settings). Alternatively, the OpenHPC documentation package (`docs-ohpc`) includes a template script which includes a summary of all of the commands used herein. This script can be used in conjunction with a simple text file to define the local site variables defined in the previous section (see Requirements/Assumptions) and is provided as a convenience for administrators. For additional information on accessing this script, please see the Automation Appendix.

2 Install Base Operating System

2.1 Install OS

In an external setting, installing the desired Base OS on a *head node* typically involves booting from a DVD ISO image on a new server. With this approach, insert the Rocky 10 DVD, power cycle the host, and follow the distro provided directions to install the Base OS on your chosen *head node*. Alternatively, if choosing to use a pre-installed server, please verify that it is provisioned with the required Rocky 10 distribution.

Prior to beginning the installation process of OpenHPC components, several additional considerations are noted here for the head node configuration. First, the installation recipe herein assumes that the head node name is resolvable locally. Depending on the manner in which you installed the Base OS, there may be an adequate entry already defined in `/etc/hosts`. If not, add one so the head node resolves itself by name (this guide relies on it for name-based commands):

```
echo ${sms_ip} ${sms_name} >> /etc/hosts
```

While it is theoretically possible to enable SELinux on a cluster provisioned with OpenCHAMI, doing so is beyond the scope of this document. Even the use of permissive mode can be problematic and we therefore recommend disabling SELinux on the *head node*. If SELinux components are installed locally, the `selinuxenabled` command can be used to determine if SELinux is currently enabled. If enabled, consult the distro documentation for information on how to disable.

2.2 Configure HTTPS Proxy (Optional)

If your environment requires an HTTPS caching proxy for external network access, configure it here before installing any packages.

2.3 Enable EL Package Repositories

In addition to the OpenHPC package repository, the *head node* also requires access to the standard base OS distro repositories in order to resolve necessary dependencies. For Rocky 10, the requirements are to have access to the BaseOS, Appstream, Extras, CRB, and EPEL repositories for which mirrors are freely available online:

- Rocky-10 (e.g. http://download.rockylinux.org/pub/rocky/10/)
- EPEL 10 (e.g. http://download.fedoraproject.org/pub/epel/10/)

Although the public EPEL repository would be enabled automatically upon installation of the `ohpc-release` package, we install it now. Note that this does depend on the Rocky 10 Extras repository, which is shipped with Rocky 10 and is typically enabled by default. In contrast, the CRB repository is typically disabled in a standard install, but can be enabled from EPEL as follows:

```
dnf -y install epel-release dnf-plugins-core
dnf config-manager --set-enabled crb
```

2.4 Configure Firewall

Provisioning services rely on DHCP, TFTP, and HTTP network protocols. Depending on the local Base OS configuration on the head node, default firewall rules may prohibit these services. Consequently, this recipe assumes that the local firewall running on the head node is disabled (it is still recommended to have additional security boundaries like a firewall to protect the cluster from the Internet). If installed, the default firewall service can be disabled as follows:

```
systemctl disable --now firewalld.service || true
```

2.5 Configure NTP

HPC systems rely on synchronized clocks throughout the system and the NTP protocol can be used to facilitate this synchronization. To enable NTP services on the head node with a specific server `${ntp_server}`, and allow this server to serve as a local time server for the cluster, issue the following:

```
dnf -y install chrony
systemctl enable chronyd.service
echo "local stratum 10" >> /etc/chrony.conf
echo "server ${ntp_server}" >> /etc/chrony.conf
echo "allow all" >> /etc/chrony.conf
systemctl restart chronyd.service
```

Tip

Note: Note that the "allow all" option specified for the chrony time daemon allows all servers on the local network to be able to synchronize with the head node. Alternatively, you can restrict access to fixed IP ranges and an example config line allowing access to a local class B subnet is as follows:

```
allow 192.168.0.0/16
```

2.6 Generate SSH Key Pair

An SSH key pair on the *head node* enables passwordless access to compute nodes and is useful for automating cluster configuration tasks.

If an ed25519 key does not already exist, generate one:

```
[[ -f ~/.ssh/id_ed25519 ]] || ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -N ''
```

Tip

Note: The key is generated without a passphrase (`-N ''`) to allow unattended use in scripts and automated workflows. Ensure that access to the head node is appropriately restricted.

2.7 Setup NFS Server

Here we set up NFS mounting of a `$HOME` file system and the public OpenHPC install path (`/opt/ohpc/pub`) that will be hosted by the *head node* in this example configuration.

```
# Install and Start NFS server
dnf -y install nfs-utils
systemctl start nfs-server.service

# Create OpenHPC public packages directory
mkdir -p /opt/ohpc/pub

# Export /home and OpenHPC public packages from head node
echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >> /etc/exports
echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports
exportfs -a

# Restart and enable nfs server
systemctl restart nfs-server.service
systemctl enable nfs-server.service
```

2.8 Optionally add InfiniBand support services on *head node*

The following command adds OFED and PSM support using base distro-provided drivers to the chosen *head node*.

```
dnf -y groupinstall "InfiniBand Support"
# Load IB services
udevadm trigger --type=devices --action=add
systemctl restart rdma-load-modules@infiniband.service
```

Tip

Note: InfiniBand networks require a subnet management service that can typically be run on either an administrative node, or on the switch itself. The optimal placement and configuration of the subnet manager is beyond the scope of this document, but Rocky 10 provides the `opensm` package should you choose to run it on the *head node*.

With the InfiniBand drivers included, you can also enable (optional) IPoIB functionality which provides a mechanism to send IP packets over the IB network. If you plan to mount a Lustre file system over InfiniBand, then having IPoIB enabled is a requirement for the Lustre client. The following creates a NetworkManager connection for the IPoIB interface `${ipoib_dev}` on the *head node*, with the address `${sms_ipoib}` and subnet prefix `${ipoib_prefix_length}`. Set `${ipoib_dev}` to the name the system gives the interface, for example `ibp3s0` under predictable interface naming, and add one connection per port if the system contains dual-ported or multiple HCAs.

```
# Define the local IPoIB interface, address, and subnet prefix
nmcli connection add type infiniband con-name ipoib ifname "${ipoib_dev}" \
    ipv4.method manual ipv4.addresses "${sms_ipoib}/${ipoib_prefix_length}"

# Bring up the IPoIB interface
nmcli connection up ipoib
```

2.9 Optionally add RoCE support on *head node*

RoCE (RDMA over Converged Ethernet) runs RDMA over Ethernet NICs that support it, for example NVIDIA ConnectX "Dx" models. It uses the same distro-provided RDMA stack as InfiniBand, but it needs no subnet manager and no IPoIB: RDMA traffic is addressed by the Ethernet interface's own IP address. This recipe runs RoCE over the internal cluster network. The following command adds the RDMA stack to the *head node*. If you enabled InfiniBand above, the stack is already installed and the script skips this step.

```
dnf -y groupinstall "InfiniBand Support"
# Load the RDMA drivers for the RoCE NIC
udevadm trigger --type=devices --action=add
systemctl restart rdma-load-modules@rdma.service
```

RoCE takes its own MTU (up to 4096 bytes) from the Ethernet MTU: the largest that fits with room for headers. An Ethernet MTU of 1500 gives RoCE 1024, and exactly 4096 gives only 2048; about 4200 or more (9000 is common) gives the full 4096. RoCE works at the default MTU, and a larger one adds a little bandwidth. Every host and switch port on the network must carry the larger frames.

This recipe leaves the MTU unchanged. To use a larger one, set the same value on the *head node*'s internal interface and on the computes' interfaces in your OpenCHAMI network configuration.

2.10 Optionally add Omni-Path support services on *head node*

The following command adds Omni-Path support using base distro-provided drivers to the chosen *head node*.

```
dnf -y install opa-basic-tools
```

Tip

Note: OmniPath networks require a subnet management service that can typically be run on either an administrative node, or on the switch itself. The optimal placement and configuration of the subnet manager is beyond the scope of this document, but Rocky 10 provides the `opa-fm` package should you choose to run it on the *head node*.

Tip

Note: Many server BIOS configurations have PXE network booting configured as the primary option in the boot order by default. If your compute nodes have a different device as the first in the sequence, the `ipmitool` utility can be used to enable PXE.

```
ipmitool -E -I lanplus -H ${bmc_ipaddr} -U root chassis \
    bootdev pxe options=persistent
```

3 Install OpenHPC Components

3.1 Enable OpenHPC Repository

With the Base OS installed and booted, the next step is to add desired OpenHPC packages onto the *head node* in order to provide provisioning and resource management services for the rest of the cluster.

To begin, enable use of the OpenHPC repository by adding it to the local list of available package repositories. Note that this requires network access from your *head node* to the OpenHPC repository, or alternatively, that the OpenHPC repository be mirrored locally. In cases where network external connectivity is available, OpenHPC provides an `ohpc-release` package that includes GPG keys for package signing and enabling the repository. The example which follows illustrates installation of the `ohpc-release` package directly from the OpenHPC build server.

```
dnf -y install http://repos.openhpc.community/OpenHPC/4/EL_10/aarch64/\
ohpc-release-4-1.el10.aarch64.rpm
```

Tip

Note: Many sites may find it useful or necessary to maintain a local copy of the OpenHPC repositories. To facilitate this need, standalone tar archives are provided – one containing a repository of binary packages as well as any available updates, and one containing a repository of source RPMS. The tar files also contain a simple bash script to configure the package manager to use the local repository after download. To use, simply unpack the tarball where you would like to host the local repository and execute the `make_repo.sh` script. Tar files for this release can be found at http://repos.openhpc.community/dist/4.2

3.2 Install OpenHPC Base Packages

Now OpenHPC packages can be installed. The `ohpc-base` meta-package requires a module tool to manage the user environment, and OpenHPC provides two of them: Lmod in the `lmod-ohpc` package and Environment Modules in the `environment-modules-ohpc` package. Both read the OpenHPC module hierarchy and provide the `module` command. The `${modules_package}` variable in `input.local` selects the one installed on the head node and, later in this guide, on the compute nodes; it defaults to `lmod-ohpc`. To add the base package and the module tool on the head node issue the following:

```
dnf -y install ohpc-base ${modules_package:-lmod-ohpc}
```

Tip

Naming the module tool keeps the choice explicit; `ohpc-base` alone normally pulls in Lmod, which it suggests. The two tools can also be installed side by side: the startup scripts `/etc/profile.d/modules.sh` and `modules.csh`, which define the `module` command, and `/usr/bin/modulecmd` are links managed with `alternatives`, and `environment-modules-ohpc` takes precedence over `lmod-ohpc` when both are present. Run `alternatives --config modules.sh` to select the active tool.

4 Install Slurm

4.1 Add Slurm on Head Node

The following command adds the Slurm workload manager server components to the chosen *head node*. Note that client-side components will be added to the corresponding compute image in a subsequent step. Note that Slurm leverages the use of the *munge* library to provide authentication services and this daemon also needs to be running on all hosts within the resource management pool.

```
# Install slurm server meta-package
dnf -y install ohpc-slurm-server

# Use ohpc-provided file for starting SLURM configuration
cp /etc/slurm/slurm.conf.ohpc /etc/slurm/slurm.conf

# Setup default cgroups file
cp /etc/slurm/cgroup.conf.example /etc/slurm/cgroup.conf

# Identify resource manager hostname on head node
sed -i "s/SlurmctldHost=\S\+/SlurmctldHost=${sms_name}/" \
    /etc/slurm/slurm.conf
```

There are a wide variety of configuration options and plugins available for Slurm and the example config file illustrated above targets a fairly basic installation. In particular, job completion data will be stored in a text file (`/var/log/slurm_jobcomp.log`) that can be used to log simple accounting information. Sites who desire more detailed information, or want to aggregate accounting data from multiple clusters, will likely want to enable the database accounting back-end. This requires a number of additional local modifications (on top of installing `slurm-slurmdbd-ohpc`), and users are advised to consult the online documentation for more detailed information on setting up a database configuration for Slurm.

Tip

Note: SLURM requires enumeration of the physical hardware characteristics for compute nodes under its control. In particular, three configuration parameters combine to define consumable compute resources: *Sockets*, *CoresPerSocket*, and *ThreadsPerCore*. The default configuration file provided via OpenHPC assumes the nodes are named c1-c4 and are dual-socket, 8 cores per socket, and two threads per core for this 4-node example. If this does not reflect your local hardware, please update the configuration file at `/etc/slurm/slurm.conf` accordingly to match your nodes names and particular hardware. Be sure to run `scontrol reconfigure` to notify SLURM of the changes. Note that the SLURM project provides an easy-to-use online configuration tool that can be accessed here.

5 Install OpenCHAMI Provisioner

Installation is completed in a few parts. First we build the images that the *compute* nodes will boot off of. Then we configure the cloud-init steps to add a local account, mount remote file systems, and inject the munge key into the OS. OpenHPC components will be added to both the head node and compute nodes. The head node will get the components during install, and compute nodes will get them during image builds.

OpenCHAMI uses an Infrastructure as Code tool called image-builder to translate YAML files to system images in layers to build out reusable file system layers like container images. This enables the rapid rebuild of images in smaller more usable and readable layers. OpenCHAMI uses SquashFS images over S3 to serve to nodes, and Container images through OCI.

5.1 Install OpenCHAMI

To begin we need to add the OpenCHAMI RPM to the head node, by adding it to the list of available packages locally. This requires network access from the *head node* to the internet. We then source `/etc/profile.d/openchami.sh` to update the shell.

```
dnf -y install podman buildah ansible-core nfs-utils tcpdump tftp curl yq jq
rpm --import \
    https://github.com/OpenCHAMI/release/releases/download/\
v0.1.7/repo_public.asc
dnf -y install \
    https://github.com/OpenCHAMI/release/releases/download/\
v0.1.7/openchami-0.1.7.rpm
```

```
source /etc/profile.d/openchami.sh
```

5.2 Certificate Creation

With OpenCHAMI a minimal open source certificate authority from smallstep is needed. The included automation initializes the CA on first startup. We can immediately download a certificate into the system trust bundle on the host for trusting all subsequent OpenCHAMI certificates. Notably, the certificate authority features ACME for automatic certificate rotation.

```
openchami-certificate-update update ${sms_name}
```

5.3 Setting no proxy

OpenCHAMI runs its own internal proxy for services which may not interact well if you run another httpd server. Here we set the no proxy variable to prevent lookups. The list contains the service names as well as the head node's own name and address, so that requests to local services never go through a site proxy.

```
export additional_no_proxy="${sms_name},${sms_ip},localhost,127.0.0.1,\
opaal,opaal-idp,smd,smd-init,hydra,cloud-init,acme-deploy,bss,haproxy,\
postgres,step-ca,configurator"
if [ -z "$no_proxy" ]; then
    export no_proxy="$additional_no_proxy"
else
    export no_proxy="$no_proxy,$additional_no_proxy"
fi
if [ -z "$NO_PROXY" ]; then
    export NO_PROXY="$additional_no_proxy"
else
    export NO_PROXY="$NO_PROXY,$additional_no_proxy"
fi
```

5.4 Setup DHCP

OpenCHAMI uses CoreDHCP as the DHCP service to provide IP addresses to nodes. There is a default configuration in place that we need to overwrite.

If your router address is different than the internal network address .254 you will need to change this file on your own. CoreDHCP has been configured to only provide addresses to nodes it knows. There are some extra configuration changes you can make to change that.

```
cat > /etc/openchami/configs/coredhcp.yaml <<EOF
server4:
 plugins:
    - server_id: ${sms_ip}
    - router: ${ipv4_gateway}
    - dns: ${dns_servers}
    - netmask: ${internal_netmask}
    - coresmd: https://${sms_name}:8443 http://${sms_ip}:8081 \
/root_ca/root_ca.crt 30s 1h false
EOF
```

5.5 Complete basic OpenCHAMI setup for *head node*

At this point, all of the packages necessary to use OpenCHAMI on the *head node* should be installed. Next, we create and enable the Container Registry and S3 Object store. These are optional if you have an existing S3 or Container Registry available for use at your local site.

Tip

The S3 and registry containers are configured as Podman Quadlets, which interface with systemd allowing you to perform normal systemd operations on them like:

```
systemctl status registry.service
systemctl restart seaweedfs.service
journalctl -u registry.service
```

First create local directories for container storage.

```
mkdir -p /opt/ohpc/admin/data/oci
mkdir -p /opt/ohpc/admin/data/s3
```

Now create the container registry quadlet definition.

```
cat > /etc/containers/systemd/registry.container <<EOF
# registry.container
[Unit]
Description=Image OCI Registry
After=network-online.target
Requires=network-online.target

[Container]
ContainerName=registry
HostName=registry
Image=docker.io/library/registry:latest
Volume=/opt/ohpc/admin/data/oci:/var/lib/registry:Z
PublishPort=5000:5000

[Service]
TimeoutStartSec=0
Restart=always

[Install]
WantedBy=multi-user.target
EOF
```

Then create the local S3 object store quadlet definition. MinIO is no longer usable here: current MinIO container images are no longer published under an open source license. SeaweedFS is used instead, since its S3 gateway is compatible enough with the S3 API that none of the later `s3cmd` steps in this guide (bucket creation, ACLs, bucket policies, anonymous object reads) need to change.

```
cat > /etc/containers/systemd/seaweedfs.container <<EOF
[Unit]
Description=SeaweedFS S3
After=local-fs.target network-online.target
Wants=local-fs.target network-online.target

[Container]
ContainerName=seaweedfs
Image=docker.io/chrislusf/seaweedfs:4.50
# Volumes
Volume=/opt/ohpc/admin/data/s3:/data:Z

# Ports
PublishPort=9000:9000
PublishPort=9091:23646

# Environment Variables
Environment=AWS_ACCESS_KEY_ID=admin
Environment=AWS_SECRET_ACCESS_KEY=password1234

# Command to run in container
Exec=mini -dir=/data -s3.port=9000

[Service]
Restart=always

[Install]
WantedBy=multi-user.target
EOF
```

We need to reload the system daemon, and to allow the SeaweedFS, and OCI Registry containers to start.

```
systemctl daemon-reload
systemctl start seaweedfs.service
systemctl start registry.service
```

Configure ssh

```
# Handle SSH to compute nodes
echo -e "Host ${compute_prefix}*\n\tStrictHostKeyChecking=no" \
    >> /etc/ssh/ssh_config.d/40-ohpc.conf
```

5.6 s3cmd Setup

In this recipe OpenCHAMI uses *S3* to store and serve images and we need to add configuration files that allow us to access and manage the *S3* storage layer for *EFI* and boot image storage.

First install a CLI tool we can use to interact with the S3 service.

```
dnf -y install s3cmd
```

Now we'll create a config file.

```
cat > ~/.s3cfg <<EOF
# Setup endpoint
host_base = ${sms_name}:9000
host_bucket = ${sms_name}:9000
bucket_location = us-east-1
use_https = False

# Setup access keys
access_key = admin
secret_key = password1234

# Enable S3 v4 signature APIs
signature_v2 = False
EOF
```

Now we can use the tool to create the buckets we'll need and set ACLs to public. The S3 API answers a few seconds after `seaweedfs.service` starts, so wait until `s3cmd ls` succeeds before creating the buckets.

```
s3cmd mb s3://efi
s3cmd setacl s3://efi --acl-public
s3cmd mb s3://boot-images
s3cmd setacl s3://boot-images --acl-public
```

The nodes need to be able to read from the S3 bucket without authentication so we need to create some read policies.

```
mkdir -p /opt/ohpc/admin/images
cat > /opt/ohpc/admin/images/public-read-boot.json <<EOF
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
     "Principal":"*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::boot-images/*"]
    }
  ]
}
EOF
cat > /opt/ohpc/admin/images/public-read-efi.json <<EOF
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":"*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::efi/*"]
    }
  ]
}
EOF
```

Lastly we can now apply these policies. Objects stored in the buckets should be publicly available.

```
s3cmd setpolicy /opt/ohpc/admin/images/public-read-boot.json \
    s3://boot-images \
    --host=${sms_ip}:9000 --host-bucket=${sms_ip}:9000
s3cmd setpolicy /opt/ohpc/admin/images/public-read-efi.json \
    s3://efi \
    --host=${sms_ip}:9000 --host-bucket=${sms_ip}:9000
```

5.7 OpenCHAMI Client Install

OpenCHAMI is very configurable, and it has a client that allows you to do most of the configuration via the command line, so installing that client makes management much easier.

```
dnf -y install \
    https://github.com/OpenCHAMI/ochami/releases/download/\
v0.7.2/ochami_0.7.2_linux_arm64.rpm
mkdir -p /etc/ochami && touch /etc/ochami/config.yaml
ochami config cluster set --system --default \
    cluster cluster.uri https://${sms_name}:8443
```

5.8 Starting OpenCHAMI

Starting OpenCHAMI can take a while especially when the node images are being downloaded for the first time. The status of all OpenCHAMI services can be checked with:

```
systemctl list-dependencies openchami.target
```

By default the cloud-init server keeps its groups and defaults in memory, so they are lost whenever its container restarts. This drop-in for its quadlet stores them in a database file on its `cloud-init-data` volume instead:

```
mkdir -p /etc/containers/systemd/cloud-init-server.container.d
cat > /etc/containers/systemd/cloud-init-server.container.d/ohpc-storage.conf <<EOF
[Container]
Environment=STORAGE_BACKEND=quack
Environment=DB_PATH=/cloud-init/cloud-init.db
EOF
systemctl daemon-reload
```

```
systemctl enable --now openchami.target
```

5.9 Limiting Access to cloud-init

The cloud-init server gives each node its configuration, including the munge key, and its admin API changes that configuration for every node. haproxy publishes it on ports 8081 and 8443 of the head node. The steps below have haproxy answer `/cloud-init` only from privileged source ports (1-1023), which only root can use, and `/cloud-init/admin` only from the head node itself. podman's published ports keep the client's address and source port, so haproxy sees them as the client sent them.

Root on the compute nodes gets a privileged source port from the nftables rules in the compute image (see Building the Base OS Image). Root on the head node gets one from the rules below, so `ochami cloud-init` commands keep working when run as root. `ochami` connects to `https://${sms_name}:8443`, so the rules cover every IPv4 address `${sms_name}` resolves to, as well as `${sms_ip}` (a container head, for example, also maps the name to its container address); the first step below collects them and stops if the name resolves to IPv6, which the rules don't cover. They don't work as another user, or through `127.0.0.1`.

This is the same trust model as Warewulf's secure mode. It assumes ports below 1024 are root-only on the head and compute nodes, so both pin `net.ipv4.ip_unprivileged_port_start` at 1024, and that root on any other host on the provisioning network is trusted.

```
if getent ahosts "${sms_name}" | awk '{print $1}' | grep -q ':'; then
    echo "ERROR: ${sms_name} resolves to IPv6; use IPv4 only for it" >&2
    exit 1
fi
sms_head_addrs=$( (echo "${sms_ip}"; getent ahostsv4 "${sms_name}" \
    | awk '{print $1}') | sort -u | paste -sd, -)
echo "net.ipv4.ip_unprivileged_port_start = 1024" \
    > /etc/sysctl.d/90-ohpc-privports.conf
sysctl -q -p /etc/sysctl.d/90-ohpc-privports.conf
```

First add the access rules to haproxy's `openchami` frontend, next to the `/cloud-init` path rule, check the configuration, and reload haproxy in place. haproxy's master process re-reads its configuration on `SIGUSR2`. Restarting `haproxy.service` instead also restarts the other services in `openchami.target`. Right after start-up those services can still be restarting, so the check and reload are retried until haproxy's container stays up long enough for both.

```
haproxy_cfg=/etc/openchami/configs/haproxy.cfg
acl_file=/opt/ohpc/admin/haproxy/cloud-init-access.cfg
mkdir -p /opt/ohpc/admin/haproxy
cat > "${acl_file}" <<EOF
  # OpenHPC: /cloud-init only from privileged source ports, and
  # /cloud-init/admin only from the head node
  acl SRC_privport src_port 1:1023
  acl SRC_head src ${sms_ip}
  acl PATH_cloud-init-admin path_beg -i /cloud-init/admin
  http-request deny deny_status 403 if PATH_cloud-init !SRC_privport
  http-request deny deny_status 403 if PATH_cloud-init-admin !SRC_head
EOF
if ! grep -q 'acl SRC_privport' "${haproxy_cfg}"; then
    sed -i "/^ *acl PATH_cloud-init path_beg/r ${acl_file}" "${haproxy_cfg}"
fi
grep -q 'acl SRC_privport' "${haproxy_cfg}" \
    || { echo "ERROR: cloud-init access rules not in ${haproxy_cfg}" >&2; exit 1; }
for ((t=0; ; t++)); do
    podman exec haproxy haproxy -c -q -f /usr/local/etc/haproxy/haproxy.cfg \
        && podman kill --signal USR2 haproxy >/dev/null && break
    (( t < 12 )) || { echo "haproxy: reload FAILED"; exit 1; }
    echo "Waiting for haproxy..."
    sleep 5
done
echo "haproxy: reloaded"
```

Then the head node's own rules. Root's connections to ports 8081 and 8443 on `${sms_ip}` leave from

`${sms_ip}` with a source port between 600 and 1023. Other users can't reach the OpenCHAMI container networks directly, which would bypass haproxy. The networks' subnets are read from podman. A dedicated unit loads the rules at boot, not `nftables.service`: stopping or reloading that service flushes the whole ruleset, podman's own rules included.

```
podman_nets=$(podman network inspect \
    --format '{{range .Subnets}}{{.Subnet}} {{end}}' \
    $(podman network ls --format '{{.Name}}' | grep '^openchami-') \
    | tr ' ' '\n' | grep '^[0-9.]*/[0-9]*$' | paste -sd, -)
if [[ -z "${podman_nets}" ]]; then
    echo "ERROR: no OpenCHAMI podman network subnets found" >&2
    exit 1
fi
cat > /etc/nftables/ohpc-sport.nft <<EOF
table inet ohpc_sport
delete table inet ohpc_sport
table inet ohpc_sport {
    chain output {
        type filter hook output priority mangle; policy accept;
        ip daddr { ${sms_head_addrs} } tcp dport { 8081, 8443 } meta skuid 0 \\
            meta mark set 0x421
        ip daddr { ${podman_nets} } meta skuid != 0 \\
            reject with icmpx admin-prohibited
    }
    chain postrouting {
        type nat hook postrouting priority srcnat; policy accept;
        meta mark 0x421 meta l4proto tcp snat ip to ${sms_ip}:600-1023
    }
}
EOF
cat > /etc/systemd/system/ohpc-sport.service <<EOF
[Unit]
Description=OpenHPC source-port rules for cloud-init
After=nftables.service
Before=network-pre.target haproxy.service
Wants=network-pre.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/sbin/nft -f /etc/nftables/ohpc-sport.nft
ExecStop=/usr/sbin/nft delete table inet ohpc_sport

[Install]
WantedBy=multi-user.target
EOF
systemctl daemon-reload
systemctl enable --now ohpc-sport.service
```

5.10 MPI on the Head Node

The OpenCHAMI services run in podman networks, so the head node carries their bridge interfaces (`podman0`, `podman1`, …) next to its real ones. libfabric's TCP-based providers enumerate every interface that is up, and

MPICH can pick one of the bridges as its default NIC. An MPI program started on the head node then fails with an error such as:

```
OFI call tsenddata failed (default nic=podman3: No such file or directory)
```

Jobs run through the resource manager are not affected, because the compute nodes carry no podman interfaces. Running MPI programs on the head node itself is unusual; the OpenHPC test suite does it, and pins those tests to libfabric's *tcp* provider on the loopback interface. Pinning the interface alone does not help, because MPICH can select another provider, for example *sockets*, which ignores `FI_TCP_IFACE`. For an interactive MPI run on the head node, set both for that session:

```
export FI_PROVIDER=tcp
export FI_TCP_IFACE=${sms_eth_internal}
```

Tip

Note: job launchers forward the environment, so these variables set in a login shell also reach the jobs submitted from it. On the compute nodes they would force the tcp provider onto the named interface, which disables a faster fabric. Do not set them permanently, for example in `/etc/profile.d`.

5.11 Compute Image Configuration

We first create a directory for the compute image configuration files.

```
# Create compute image configuration directory
mkdir -v -p /opt/ohpc/admin/images/data
```

5.12 Configure Slurm Client

We now configure Slurm `slurmd`

```
# Register Slurm server with computes (using "configless" option)
echo SLURMD_OPTIONS="--conf-server ${sms_ip}" \
    > /opt/ohpc/admin/images/data/slurmd
```

5.13 Building the Base OS Image

The Image Builder is designed to use layered building of system images, as shown below to enable the use of different layers which can be modified and added to build out different capabilities into image layers so a change of one layer does not require a change and rebuild of all other layers.

Tip

The image-build is an optional piece of the OpenCHAMI software stack. If you have an existing image build process that creates the necessary boot artifacts you can use that instead.

First let's define the lowest layer of the image which disables the installation of documentation files to reduce the size of the image:

```
echo -e "%_excludedocs 1" >> /opt/ohpc/admin/images/data/rpmmacros
cat > /opt/ohpc/admin/images/nodocs.yaml <<EOF
options:
  layer_type: 'base'
  name: 'rocky10-nodocs'
  publish_tags: '10'
  pkg_manager: 'dnf'
  parent: 'scratch'
  publish_registry: '${sms_name}:5000/${compute_prefix}'
  registry_opts_push:
    - '--tls-verify=false'
repos:
  - alias: 'rocky10_BaseOS'
    url: 'https://dl.rockylinux.org/pub/rocky/10/BaseOS/aarch64/os/'
    gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-10'
  - alias: 'rocky10_AppStream'
    url: 'https://dl.rockylinux.org/pub/rocky/10/AppStream/aarch64/os/'
    gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-10'
packages:
  - bash
  - dnf
  - dnf-plugins-core
copyfiles:
  - src: '/data/rpmmacros'
    dest: '/root/.rpmmacros'
EOF
```

Build this first layer:

```
podman run \
    --rm \
    --device /dev/fuse \
    --network host \
    -v /opt/ohpc/admin/images/data:/data \
    -v /opt/ohpc/admin/images/nodocs.yaml:/home/builder/config.yaml \
    ghcr.io/openchami/image-build:latest image-build \
        --config config.yaml \
        --log-level INFO
```

Now let's define a base layer we can use to build on. This base layer will install common packages and also create our initramfs. In this recipe it's created to build an initramfs that supports liveOS so the image will boot into memory.

```
cat > /opt/ohpc/admin/images/base.yaml <<EOF
options:
  layer_type: 'base'
  name: 'rocky10-base'
  publish_tags: '10'
  pkg_manager: 'dnf'
  parent: '${sms_name}:5000/${compute_prefix}/rocky10-nodocs:10'
  publish_registry: '${sms_name}:5000/${compute_prefix}'
  registry_opts_pull:
    - '--tls-verify=false'
  registry_opts_push:
    - '--tls-verify=false'
repos:
  - alias: 'rocky10_BaseOS'
    url: 'https://dl.rockylinux.org/pub/rocky/10/BaseOS/aarch64/os/'
    gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-10'
  - alias: 'rocky10_AppStream'
    url: 'https://dl.rockylinux.org/pub/rocky/10/AppStream/aarch64/os/'
    gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-10'
package_groups:
  - 'Minimal Install'
  - 'Development Tools'
packages:
  - kernel
  - wget
  - dracut-live
  - cloud-init
  - nftables
  - chrony
  - rsyslog
  - sudo
cmds:
  - cmd: >
      DRACUT_NO_XATTR=1 dracut --add "dmsquash-live livenet network-manager"
      --kver "\$(basename /lib/modules/*)"
      -N -f --logfile /tmp/dracut.log 2>/dev/null
    loglevel: INFO
  - cmd: 'echo DRACUT LOG:; cat /tmp/dracut.log'
    loglevel: INFO
EOF
```

Now let's build this base layer. This will take about 5 minutes but we should not have to rebuild it very often.

```
podman run \
    --rm \
    --device /dev/fuse \
    --network host \
    -v /opt/ohpc/admin/images/base.yaml:/home/builder/config.yaml \
    ghcr.io/openchami/image-build:latest image-build \
        --config config.yaml \
        --log-level DEBUG
```

Now we can build on top of the base layer by setting it to be the parent of the compute-base layer.

```
cat > /opt/ohpc/admin/images/compute-base.yaml << EOF
options:
  layer_type: 'base'
  name: 'compute-base'
  publish_tags:
    - '10'
  pkg_manager: 'dnf'
  parent: '${sms_name}:5000/${compute_prefix}/rocky10-base:10'
  publish_registry: '${sms_name}:5000/${compute_prefix}'
  registry_opts_pull:
    - '--tls-verify=false'
  registry_opts_push:
    - '--tls-verify=false'
repos:
  - alias: 'rocky10_CRB'
    url: 'https://dl.rockylinux.org/pub/rocky/10/CRB/aarch64/os/'
    gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-10'
  - alias: 'Epel10'
    url: 'https://dl.fedoraproject.org/pub/epel/10z/Everything/aarch64/'
    gpg: 'https://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-10'
package_groups:
  - 'InfiniBand Support'
packages:
  - vim
  - nfs-utils
  - tcpdump
  - traceroute
  - git
  - http://repos.openhpc.community/OpenHPC/4/EL_10/aarch64/\
ohpc-release-4-1.el10.aarch64.rpm
EOF
```

Then we can build it. Any modifications to the above configuration will only require a rebuild of this layer.

```
podman run \
    --rm \
    --device /dev/fuse \
    --network host \
    -v /opt/ohpc/admin/images/compute-base.yaml:/home/builder/config.yaml \
    ghcr.io/openchami/image-build:latest image-build \
        --config config.yaml \
        --log-level DEBUG
```

You can keep adding layers as you see fit, and branch off if so desired. We will use the next layer to add additional packages later on so no need to build it just yet.

The head node answers cloud-init requests only from privileged source ports (see Limiting Access to cloud-init). The production layer installs these nftables rules as the image's `/etc/sysconfig/nftables.conf`: they give root's connections to ports 8081 and 8443 on the head node a source port between 600 and 1023, which other users cannot get. `nftables.service` loads them before the network comes up, so they are in place before cloud-init fetches the node's configuration. This relies on ports below 1024 being root-only, so the layer also pins `net.ipv4.ip_unprivileged_port_start` (which covers IPv6 too) at its default of 1024: a site that lowers it, for example for rootless containers, would let any user on a compute node fetch its configuration.

```
sms_addrs="${sms_ip}"
if [[ ${enable_ipoib} -eq 1 ]]; then
    sms_addrs="${sms_addrs}, ${sms_ipoib}"
fi
cat > /opt/ohpc/admin/images/data/ohpc-sport.nft <<EOF
table inet ohpc_sport {
    chain output {
        type filter hook output priority mangle; policy accept;
        ip daddr { ${sms_addrs} } tcp dport { 8081, 8443 } meta skuid 0 \\
            meta mark set 0x421
    }
    chain postrouting {
        type nat hook postrouting priority srcnat; policy accept;
        meta mark 0x421 meta l4proto tcp masquerade to :600-1023
    }
}
EOF
echo "net.ipv4.ip_unprivileged_port_start = 1024" \
    > /opt/ohpc/admin/images/data/90-ohpc-privports.conf
```

```
cp /etc/hosts /opt/ohpc/admin/images/data
cat > /opt/ohpc/admin/images/compute-prod.yaml << EOF
options:
  layer_type: 'base'
  name: 'compute-prod'
  publish_tags:
    - '10'
  pkg_manager: 'dnf'
  parent: '${sms_name}:5000/${compute_prefix}/compute-base:10'
  publish_registry: '${sms_name}:5000/${compute_prefix}'
  registry_opts_pull:
    - '--tls-verify=false'
  registry_opts_push:
    - '--tls-verify=false'
  # Publish SquashFS image to local S3
  publish_s3: 'http://${sms_name}:9000'
  s3_prefix: 'compute/base/'
  s3_bucket: 'boot-images'
repos:
  - alias: 'OpenHPC'
    url: 'http://repos.openhpc.community/OpenHPC/4/EL_10'
    gpg: 'file:///etc/pki/rpm-gpg/RPM-GPG-KEY-OpenHPC-4'
  - alias: 'OpenHPC-updates'
    url: 'http://repos.openhpc.community/OpenHPC/4/updates/EL_10'
    gpg: 'file:///etc/pki/rpm-gpg/RPM-GPG-KEY-OpenHPC-4'
copyfiles:
  - src: '/data/hosts'
    dest: '/etc/hosts'
  - src: '/data/slurmd'
    dest: '/etc/sysconfig/slurmd'
  - src: '/data/ohpc-sport.nft'
    dest: '/etc/sysconfig/nftables.conf'
  - src: '/data/90-ohpc-privports.conf'
    dest: '/etc/sysctl.d/90-ohpc-privports.conf'
packages:
  - ohpc-base-compute
  - ohpc-slurm-client
  - pdsh-ohpc
  - ${modules_package:-lmod-ohpc}
cmds:
  - cmd: systemctl disable firewalld.service
    loglevel: INFO
  - cmd: >-
      echo '${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0'
      >> /etc/fstab
    loglevel: INFO
  - cmd: >-
      echo '${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev,nosuid 0 0'
      >> /etc/fstab
    loglevel: INFO
  - cmd: echo 'server ${sms_ip} iburst' >> /etc/chrony.conf
    loglevel: INFO
  - cmd: mkdir -p /opt/ohpc/pub
    loglevel: INFO
  - cmd: echo "account required pam_slurm.so" >> /etc/pam.d/sshd
    loglevel: INFO
  - cmd: systemctl enable munge.service slurmd.service
    loglevel: INFO
  - cmd: systemctl enable nftables.service
    loglevel: INFO
```

6 Additional Customization

This chapter highlights common additional customizations that can *optionally* be applied to the local cluster environment. Details on the steps required for each of these customizations are discussed further in the following sections.

6.1 Enable InfiniBand drivers

If your compute resources support InfiniBand, the following commands add OFED and PSM support using base distro-provided drivers to the compute image.

```
# Add IB support and enable
yq -i '.package_groups += ["InfiniBand Support"]' \
    /opt/ohpc/admin/images/compute-prod.yaml
```

6.2 Enable RoCE drivers

If your compute resources support RoCE, the following commands add the same RDMA stack to the compute image. If you enabled InfiniBand drivers above, the stack is already there and the script skips the install.

```
# Add RDMA support
yq -i '.package_groups += ["InfiniBand Support"]' \
    /opt/ohpc/admin/images/compute-prod.yaml
```

To run MPI over RoCE, select the RDMA device and port for UCX, for example `UCX_NET_DEVICES=mlx5_1:1`; `rdma link` lists each RDMA device with its network interface.

6.3 Enable Omni-Path drivers

If your compute resources support Omni-Path, the following commands add OPA support using base distro-provided drivers to the compute image.

```
# Add OPA support and enable
yq -i '.packages += ["opa-basic-tools"]' \
    /opt/ohpc/admin/images/compute-prod.yaml
yq -i '.packages += ["libpsm2"]' \
    /opt/ohpc/admin/images/compute-prod.yaml
```

6.4 Increase locked memory limits

In order to utilize InfiniBand or Omni-Path as the underlying high speed interconnect, it is generally necessary to increase the locked memory settings for system users. This can be accomplished by adding the `/etc/security/limits.d/40-ohpc-limits.conf` file and this should be performed on all job submission hosts. In this recipe, jobs are submitted from the *head node*, and the following commands can be used to update the maximum locked memory settings on both the *head node* and compute nodes:

```
# Update memlock settings on head node
echo '* soft memlock unlimited' >> \
  /etc/security/limits.d/40-ohpc-limits.conf
echo '* hard memlock unlimited' >> \
  /etc/security/limits.d/40-ohpc-limits.conf

# Update memlock settings on compute
C="echo '* soft memlock unlimited' >> /etc/security/limits.d/40-ohpc-limits.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
C="echo '* hard memlock unlimited' >> /etc/security/limits.d/40-ohpc-limits.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
```

6.5 Enable ssh control via resource manager

An additional optional customization that is recommended is to restrict `ssh` access on compute nodes to only allow access by users who have an active job associated with the node. This can be enabled via the use of a pluggable authentication module (PAM) provided as part of the Slurm package installs. To enable this feature on *compute* nodes, issue the following:

```
C="echo 'account    required     pam_slurm.so' >> /etc/pam.d/sshd" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
```

6.6 NVIDIA GPU Drivers

To add the optional NVIDIA GPU driver to the compute nodes, an additional external dnf repository provided by NVIDIA must be configured. Once the repository is configured, the GPU driver needs to be installed on the compute image and the corresponding toolkit installed on the head node.

OpenHPC provides a convenience package to enable the NVIDIA repository locally along with compatibility packages that integrate the NVIDIA HPC SDK within the standard OpenHPC user environment.

Tip

Note: The driver is NVIDIA's open GPU kernel module (`kmod-nvidia-open-dkms`), the only kernel module in NVIDIA's EL10 repository. It supports GPUs of the Turing architecture and newer.

```
# Add NVIDIA GPU driver repository to the head node
dnf -y install cuda-repo-ohpc

# Add NVIDIA GPU driver repository to the compute nodes
yq -i '.packages += ["cuda-repo-ohpc"]' \
    /opt/ohpc/admin/images/compute-prod.yaml

# Install the GPU driver on the compute nodes
yq -i '.packages += ["nvidia-driver-cuda", "kmod-nvidia-open-dkms"]' \
    /opt/ohpc/admin/images/compute-prod.yaml

# Build the driver for the image's kernel (DKMS otherwise targets the running kernel)
C="dkms autoinstall -k \$(ls /lib/modules | sort -V | tail -n 1)" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml

# Enable DKMS service to automatically rebuild driver
C="systemctl enable dkms.service" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml

# Install the toolkit on the head node
dnf -y install cuda-devel-ohpc
```

6.7 Enable System Log Forwarding

It is often desirable to consolidate system logging information for the cluster in a central location, both to provide easy access to the data, and to reduce the impact of storing data inside the compute node's memory footprint if it is stateless. The following commands highlight the steps necessary to configure compute nodes to forward their logs to the head node, and to allow the head node to accept these log requests.

```
# Configure head node to receive messages and reload rsyslog configuration
echo 'module(load="imudp")' >> /etc/rsyslog.d/ohpc.conf
echo 'input(type="imudp" port="514")' >> /etc/rsyslog.d/ohpc.conf
systemctl restart rsyslog.service

# Define compute node forwarding destination
C="echo '*.* @${sms_ip}:514' > /etc/rsyslog.d/ohpc-forward.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml

# Disable most local logging on computes.
# Emergency and boot logs will remain on the compute nodes
C="sed -i 's/^\*\.info/#\*\.info/' /etc/rsyslog.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
C="sed -i 's/^authpriv/#authpriv/' /etc/rsyslog.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
C="sed -i 's/^mail/#mail/' /etc/rsyslog.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
C="sed -i 's/^cron/#cron/' /etc/rsyslog.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
C="sed -i 's/^uucp/#uucp/' /etc/rsyslog.conf" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml

# Let the compute nodes pick up the new configuration
C="systemctl enable rsyslog.service" \
  yq -i '.cmds += [{"cmd": strenv(C)}]' /opt/ohpc/admin/images/compute-prod.yaml
```

6.8 Cluster Admin Tools

6.8.1 Add ClusterShell

ClusterShell is an event-based Python library to execute commands in parallel across cluster nodes. Installation and basic configuration defining three node groups (*adm*, *compute*, and *all*) is as follows:

```
# Install ClusterShell
dnf -y install clustershell

# Setup node definitions
mv /etc/clustershell/groups.d/local.cfg \
   /etc/clustershell/groups.d/local.cfg.orig
echo "adm: ${sms_name}" > \
  /etc/clustershell/groups.d/local.cfg
echo "compute: ${compute_prefix}[1-${num_computes}]" >> \
  /etc/clustershell/groups.d/local.cfg
echo "all: @adm,@compute" >> \
  /etc/clustershell/groups.d/local.cfg
```

6.8.2 Add genders

genders is a static cluster configuration database or node typing database used for cluster configuration management. Other tools and users can access the genders database in order to make decisions about where an action, or even what action, is appropriate based on associated types or "genders". Values may also be

assigned to and retrieved from a *gender* to provide further granularity. The following example highlights installation and configuration of two genders: *compute* and *bmc*.

```
# Install genders
dnf -y install genders-ohpc

# Generate a sample genders file
echo -e "${sms_name}\tsms" > /etc/genders
for ((i=0; i<num_computes; i++)) ; do
          echo -e "${c_name[$i]}\tcompute,bmc=${c_bmc[$i]}"
    done >> /etc/genders
```

6.8.3 Add ConMan

conman is a serial console management program designed to support a large number of console devices and simultaneous users. It supports logging console device output and connecting to compute node consoles via IPMI serial-over-lan. Installation and example configuration is outlined below.

```
# Install conman to provide a front-end to compute consoles and log output
dnf -y install conman-ohpc

# Configure conman for computes
# (note your IPMI password is required for console access)
for ((i=0; i<num_computes; i++)) ; do
    echo -n 'CONSOLE name="'"${c_name[$i]}"'" dev="ipmi:'"${c_bmc[$i]}"'" '
    echo -n 'ipmiopts="'U:"${bmc_username}",P:
    echo "${IPMI_PASSWORD:-undefined}",W:solpayloadsize'"'
    done >> /etc/conman.conf

# Enable and start conman
systemctl enable conman.service
systemctl start conman.service
```

6.8.4 Add NHC

Resource managers often provide for a periodic "node health check" to be performed on each compute node to verify that the node is working properly. Nodes which are determined to be "unhealthy" can be marked as down or offline so as to prevent jobs from being scheduled or run on them. This helps increase the reliability and throughput of a cluster by reducing preventable job failures due to misconfiguration, hardware failure, etc. OpenHPC distributes `nhc` to fulfill this requirement.

In a typical scenario, the `nhc` driver script is run periodically on each compute node by the resource manager client daemon. It loads its configuration file to determine which checks are to be run on the current node (based on its hostname). Each matching check is run, and if a failure is encountered, `nhc` will exit with an error message describing the problem. It can also be configured to mark nodes offline so that the scheduler will not assign jobs to bad nodes, reducing the risk of system-induced job failures.

```
# Install NHC on head and compute nodes
dnf -y install nhc-ohpc
yq -i '.packages += ["nhc-ohpc"]' \
    /opt/ohpc/admin/images/compute-prod.yaml
```

```
# Register as SLURM's health check program
echo "HealthCheckProgram=/usr/sbin/nhc" >> /etc/slurm/slurm.conf

# execute every five minutes
echo "HealthCheckInterval=300" >> /etc/slurm/slurm.conf
```

6.9 Add Magpie

Magpie contains a number of scripts to aid in running a variety of big data software frameworks within HPC queuing environments. Examples include Hadoop, Spark, Hbase, Storm, Pig, Mahout, Phoenix, Kafka, Zeppelin, and Zookeeper. Consult the online repository for more information on using these scripts; basic installation is outlined as follows:

```
# Install magpie
dnf -y install magpie-ohpc
```

6.10 Charliecloud

Typical Charliecloud workflows are based around Docker containers, but it is not strictly necessary to install Docker itself on the HPC resource. A common pattern is to build the Docker container on a laptop or VM and upload the result to the cluster for use with Charliecloud. More information can be found at https://hpc.github.io/charliecloud/

7 Deploy Cluster

We now set (or refresh) the secret token used for `ochami` commands. Stop here if no token came back (for example because the request to the token service was sent through a proxy); every `ochami` call below would otherwise fail later, less clearly.

```
CLUSTER_ACCESS_TOKEN=$(gen_access_token)
if [[ -z "${CLUSTER_ACCESS_TOKEN}" ]]; then
    echo "ERROR: gen_access_token returned an empty token" >&2
    exit 1
fi
export CLUSTER_ACCESS_TOKEN
```

7.1 Rebuild Image

Rebuild the final compute node image with all optional packages.

```
podman run \
    --rm \
    --device /dev/fuse \
    --network host \
    -e S3_ACCESS=admin \
    -e S3_SECRET=password1234 \
    -v /opt/ohpc/admin/images/data:/data \
    -v /opt/ohpc/admin/images/compute-prod.yaml:/home/builder/config.yaml:z \
    ghcr.io/openchami/image-build:latest \
        image-build \
        --config config.yaml
```

7.2 Setting Boot Parameters

Boot Script Service (BSS) is what provides the path to vmlinuz and the initrd, and provides the kernel parameters to the node during iPXE.

First check that the boot images are available.

```
s3cmd ls -Hr s3://boot-images/
```

Get some data from our images to create our BSS payload.

```
export s3Output=$(s3cmd ls -Hr s3://boot-images/)
export os=$(echo "$s3Output" | awk '{print $4}' | grep 'boot-images/compute')
export ramfs=$(echo "$s3Output" | awk '{print $4}' | grep 'initram')
export hdrs=$(echo "$s3Output" | awk '{print $4}' | grep 'vmlinuz')
export params="ds=nocloud-net;s=http://${sms_ip}:8081/cloud-init/ \
nomodeset ro root=live:http://${sms_ip}:9000/${os:5} ip=dhcp \
overlayroot=tmpfs overlayroot_cfgdisk=disabled apparmor=0 selinux=0 \
console=ttyS0,115200 ip6=off cloud-init=enabled"
```

Create a YAML payload for BSS.

```
mkdir -p /opt/ohpc/admin/nodes
cat > /opt/ohpc/admin/nodes/compute-boot.yaml <<EOF
kernel: 'http://${sms_ip}:9000/${hdrs:5}'
initrd: 'http://${sms_ip}:9000/${ramfs:5}'
params: '${params}'
macs:
EOF
for((i=0; i < $num_computes; i++)); do
    echo "  - ${c_mac[$i]}" >> \
        /opt/ohpc/admin/nodes/compute-boot.yaml
done
```

And then finally upload our payload.

```
ochami -k bss boot params set \
    -f yaml -d @/opt/ohpc/admin/nodes/compute-boot.yaml
```

7.3 Building cloud-init Image

OpenCHAMI uses cloud-init for post-boot node configuration. We will establish some basic cloud-init configurations here.

7.3.1 Create Default Configuration

Let's create a directory to hold our default cloud-init configuration and create an SSH key pair that we can use for SSH access later.

```
mkdir -p /opt/ohpc/admin/cloud-init
ssh-keygen -t ed25519 -q -f "$HOME/.ssh/id_openchami" -N ""
cat >> "$HOME/.ssh/config" <<EOF

Host ${compute_prefix}*
    IdentityFile ~/.ssh/id_openchami
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
EOF
```

Now we can define the payload to update the defaults.

```
cat > /opt/ohpc/admin/cloud-init/defaults.yaml <<EOF
base-url: "http://${sms_ip}:8081/cloud-init"
cluster-name: "CLUSTER"
nid-length: 1
public-keys: []
short-name: "${compute_prefix}"
EOF
for i in /root/.ssh/id*pub; do
    key=$(cat "$i") yq '.public-keys += [strenv(key)]' \
        -i /opt/ohpc/admin/cloud-init/defaults.yaml
done
```

And then finally upload our payload.

```
ochami -k cloud-init defaults set \
    -f yaml -d @/opt/ohpc/admin/cloud-init/defaults.yaml
```

7.3.2 Configure Munge

OpenCHAMI provides a cloud-init server that can be used, in conjunction with the cloud-init client, to provide post-boot configurations. The configuration below is a minimal configuration that should at least let the root user login using the public key and distribute the munge key file to all compute nodes.

The head node answers cloud-init only from privileged source ports (see Limiting Access to cloud-init). The `bootcmd` below adds the same limit on the compute nodes themselves: only root can reach ports 8081 and 8443 on the head node.

```
munge_key=$(cat /etc/munge/munge.key | base64 -w 0)
export munge_key
sms_addrs="${sms_ip}"
if [[ ${enable_ipoib} -eq 1 ]]; then
    sms_addrs="${sms_addrs}, ${sms_ipoib}"
fi
cat > /opt/ohpc/admin/cloud-init/compute.yaml <<EOF
- name: compute
  description: "compute template"
  file:
    encoding: plain
    content: |
      ## template: jinja
      #cloud-config
      merge_how:
      - name: list
        settings: [append]
      - name: dict
        settings: [no_replace, recurse_list]
      users:
        - name: root
          ssh_authorized_keys: {{ ds.meta_data.instance_data.v1.public_keys }}
      bootcmd:
        - nft add table inet ohpc
        - nft 'add chain inet ohpc output { type filter hook output priority 0 ; }'
        - >-
          nft 'add rule inet ohpc output ip daddr { ${sms_addrs} }
          tcp dport { 8081, 8443 } meta skuid != 0 reject'
      write_files:
        - content: ${munge_key}
          path: /etc/munge/munge.key
          permissions: '0400'
          owner: munge:munge
          encoding: base64
      runcmd:
        - systemctl restart munge.service
EOF
```

Once you are done updating cloud-init you can post the file to the cloud-init server.

```
ochami -k cloud-init group set \
    -f yaml -d @/opt/ohpc/admin/cloud-init/compute.yaml
```

7.4 Optionally configure IPoIB on *compute nodes*

To use IPoIB on the compute nodes (required if planning to mount Lustre/BeeGFS over IB; see Optionally add InfiniBand support services on *head node*), give each compute its own address `${c_ipoib[i]}` on the IPoIB interface `${ipoib_dev}`, with the `${ipoib_netmask}` netmask (`${ipoib_prefix_length}` prefix). These are per-node settings, so they belong in the OpenCHAMI node definitions:

OpenCHAMI delivers per-node configuration through cloud-init groups. Give each compute its own group, whose cloud-config writes a NetworkManager connection for `${ipoib_dev}` with that node's address; Net-

workManager brings it up when the interface appears. Each node joins its group when it is discovered (see Node Discovery):

```
# One cloud-init group per compute with its IPoIB connection
for ((i=0; i<num_computes; i++)) ; do
    gf=/opt/ohpc/admin/cloud-init/ipoib-${c_name[$i]}.yaml
    cat > "${gf}" <<EOF
- name: ipoib-${c_name[$i]}
  description: "IPoIB on ${c_name[$i]}"
  file:
    encoding: plain
    content: |
      #cloud-config
      merge_how:
      - name: list
        settings: [append]
      - name: dict
        settings: [no_replace, recurse_list]
      write_files:
        - path: /etc/NetworkManager/system-connections/${ipoib_dev}.nmconnection
          permissions: '0600'
          content: |
            [connection]
            id=${ipoib_dev}
            type=infiniband
            interface-name=${ipoib_dev}

            [infiniband]
            transport-mode=datagram

            [ipv4]
            method=manual
            address1=${c_ipoib[$i]}/${ipoib_prefix_length}
      runcmd:
        - nmcli connection reload
EOF
    ochami -k cloud-init group set -f yaml -d @"${gf}"
done
```

7.5 Node Discovery

OpenCHAMI can do dynamic discovery, but here we will do node discovery by YAML file. The YAML file can be imported by the discover command creating a way for adding nodes that do not have redfish capabilities, or when you just want to provide a list of nodes.

Create the `nodes.yaml` file the static discovery will use to populate SMD. Also add the compute nodes to `/etc/hosts` for SSH access, with both the short and the fully qualified name, `<name> <name>.${dns_domain}`, as xCAT's `makehosts` and Confluent's `confluent2hosts` write them; tools such as the test suite's `koomie_cf` find hosts only by a dotted name.

```
mkdir -p /opt/ohpc/admin/nodes
nf=/opt/ohpc/admin/nodes/nodes.yaml

## Add BMCs
echo "bmcs:" > $nf
for((i=0; i < $num_computes; i++)); do
    nid=$((i+1))
    printf -v mac '%02x:%02x' "$(((nid >> 8) & 0xff))" "$((nid & 0xff))"
    cat >> $nf << EOF
- xname: x1000c0s0b${nid}
  ip: ${c_bmc[$i]}
  mac: 02:00:00:00:${mac}
EOF
done

## Add nodes
echo "nodes:" >> $nf
for((i=0; i < $num_computes; i++)); do
    nid=$((i+1))
    cat >> $nf << EOF
- name: ${c_name[$i]}
  nid: ${nid}
  xname: x1000c0s0b${nid}n0
  bmc: x1000c0s0b${nid}
  groups:
  - compute
  interfaces:
  - mac_addr: ${c_mac[$i]}
    ip_addrs:
    - name: internal
      ip_addr: ${c_ip[$i]}
EOF
done

## Add nodes to /etc/hosts
for((i=0; i < $num_computes; i++)); do
    echo "${c_ip[$i]} ${c_name[$i]} ${c_name[$i]}.${dns_domain:-local}" >> /etc/hosts
done
```

If IPoIB is configured on the compute nodes, add each one to its IPoIB group.

```
yq -i '.nodes[] |= (.groups += ["ipoib-" + .name])' \
    /opt/ohpc/admin/nodes/nodes.yaml
```

Now use the generated file to populate the SMD database.

```
ochami discover static \
    -f yaml -d @/opt/ohpc/admin/nodes/nodes.yaml
```

`ochami discover static` populates SMD components, EthernetInterfaces (under the correct node xname), and group membership from the `nodes.yaml` file in a single step.

7.6 Verify Configuration Before Boot

Before booting compute nodes, verify that BSS and cloud-init are configured correctly. These checks catch the most common failures before a wasted boot cycle.

First, confirm that BSS has the correct kernel parameters stored:

```
ochami -k bss boot params get \
    | grep -q "ds=nocloud-net" \
    && echo "BSS: OK" \
    || { echo "BSS: FAILED - check boot params"; exit 1; }
```

Next, confirm that the cloud-init default configuration was uploaded successfully:

```
ochami -k cloud-init defaults get \
    | grep -q "base-url" \
    && echo "cloud-init defaults: OK" \
    || { echo "cloud-init defaults: FAILED"; exit 1; }
```

Next, confirm that the compute group cloud-init configuration is present. This is what delivers the SSH keys and munge key to the node at boot:

```
ochami -k cloud-init group get raw compute \
    | grep -q "compute" \
    && echo "cloud-init group: OK" \
    || { echo "cloud-init group: FAILED"; exit 1; }
```

Finally, confirm that the cloud-init server resolves each compute node by its IP address. This is the lookup the node itself performs at boot, and the only check here that has to run after node discovery.

The server answers `/cloud-init/meta-data` from an in-memory map of SMD's Ethernet interfaces that it refreshes on a timer, so a node registered a moment ago is not visible to it yet. A node that boots into that window gets no cloud-init data at all, which means no root SSH key and no munge key, and it never becomes reachable. Wait for the lookup to succeed before powering the nodes on.

The check below asks the cloud-init server the question the node will ask, with `X-Forwarded-For` carrying the node's address so that the server resolves the same entry. It asks the server's container directly rather than through haproxy, because haproxy replaces `X-Forwarded-For` with the address the request came from, so through it the server would look up the head node instead. The node's own request through haproxy is tested when it boots. The container's address exists only on the head node, so the check tells curl to ignore an `http_proxy` set in the environment; a proxy cannot reach the podman network and the request would time out there.

```
cloud_init_ip=$(podman inspect --format \
    '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
    cloud-init-server)
for ((i=0; i<num_computes; i++)); do
    for ((t=0; ; t++)) ; do
        curl -fsS -m 5 -o /dev/null --noproxy '*' \
            -H "X-Forwarded-For: ${c_ip[$i]}" \
            "http://${cloud_init_ip}:27777/meta-data" && break
        (( t < 18 )) || {
            echo "cloud-init node: FAILED for ${c_name[$i]}"
            exit 1
        }
        echo "Waiting for cloud-init to resolve ${c_name[$i]}..."
        sleep 5
    done
done
echo "cloud-init node: OK"
```

7.7 Boot Compute Nodes

At this point, the *head node* should be able to boot the newly defined compute nodes. Assuming that the compute node BIOS settings are configured to boot over PXE, all that is required to initiate the provisioning process is to power cycle each of the desired hosts using IPMI access. The following commands use the `ipmitool` utility to initiate power resets on each of the four compute hosts. Note that the utility requires that the `IPMI_PASSWORD` environment variable be set with the local BMC password in order to work interactively.

```
for ((i=0; i<num_computes; i++)) ; do
  ipmitool -E -I lanplus -H "${c_bmc[$i]}" -U "${bmc_username}" \
    -P "${bmc_password}" chassis power reset
done
```

Once kicked off, the boot process should take less than 5 minutes (depending on BIOS post times) and you can verify that the compute hosts are available via ssh, or via parallel ssh tools to multiple hosts. For example, to run a command on the newly imaged compute hosts using `pdsh`, execute the following:

```
pdsh -w ${compute_prefix}[1-${num_computes}] uptime
c1  05:03am  up   0:02,  0 users,  load average: 0.20, 0.13, 0.05
c2  05:03am  up   0:02,  0 users,  load average: 0.20, 0.14, 0.06
c3  05:03am  up   0:02,  0 users,  load average: 0.19, 0.15, 0.06
c4  05:03am  up   0:02,  0 users,  load average: 0.15, 0.12, 0.05
```

7.8 Wait for Compute Nodes

Wait for all compute nodes to become accessible via SSH, for up to `node_boot_timeout` seconds. A node that is not reachable by then is not slow, it has failed to provision, and the checks below cannot tell you anything useful about it.

```
deadline=$(( SECONDS + ${node_boot_timeout:-600} ))
while true ; do
    pdsh -S -w "${compute_prefix}[1-${num_computes}]" true \
        >/dev/null 2>&1 && { echo "compute nodes: OK"; break; }
    if (( SECONDS >= deadline )); then
        echo "compute nodes: FAILED, no SSH in ${node_boot_timeout:-600}s"
        echo "Check the node console, and the cloud-init server with"
        echo "  podman logs cloud-init-server"
        exit 1
    fi
    echo "Waiting for compute nodes..."
    sleep 10
done
```

`sshd` answers before cloud-init has finished: its final stage still has to write the munge key and run the `runcmd` restarts. Wait for it to complete on every node, so that the resource manager steps below do not restart daemons that cloud-init is restarting at the same time.

`cloud-init status --wait` blocks for as long as cloud-init takes, with no limit of its own, so bound it on the node with `cloud_init_timeout`. The test on the exit status fails this step when cloud-init timed out (`timeout` reports 124) or failed (1), and lets a run that finished with recoverable errors (2) through: any `cc_*` module warning produces one, and a cluster whose nodes are up and usable should not stop here for that. `$?` and `$rc` are escaped so that the compute node expands them rather than the head node.

```
pdsh -S -w "${compute_prefix}[1-${num_computes}]" \
    "timeout ${cloud_init_timeout:-600} cloud-init status --wait; \
    rc=\$?; [ \$rc -eq 0 ] || [ \$rc -eq 2 ]" \
    || { echo "cloud-init: FAILED or did not finish in time"; exit 1; }
```

If IPoIB is configured, check that it came up on every compute: NetworkManager activates each node's connection once the interface and its InfiniBand link appear.

```
for ((t=0; ; t++)) ; do
    pdsh -S -w "${compute_prefix}[1-${num_computes}]" \
        "nmcli -g GENERAL.STATE connection show ${ipoib_dev} | grep -qx activated" \
        >/dev/null 2>&1 && { echo "compute IPoIB: OK"; break; }
    (( t < 30 )) || { echo "compute IPoIB: FAILED"; exit 1; }
    echo "Waiting for IPoIB on the compute nodes..."
    sleep 10
done
```

7.9 Start Slurm

In an earlier section, the Slurm resource manager was installed and configured for use on both the *head node* and *compute* node instances. With the cluster nodes up and functional, we can now startup the resource manager services in preparation for running user jobs.

```
# Start munge and slurm controller on head node
systemctl enable --now munge.service
systemctl enable --now slurmctld.service
```

Running systems may need to restart `slurmctld` to pickup any changes.

7.10 Start Slurm on the Compute Nodes

We now directly (re)start Slurm on the compute nodes.

```
# Start slurm clients on compute hosts
pdsh -w "${compute_prefix}[1-${num_computes}]" systemctl restart munge.service
pdsh -w "${compute_prefix}[1-${num_computes}]" systemctl restart slurmd.service
```

8 Install OpenHPC Development Components

The install procedure outlined in Install OpenHPC Components highlighted the steps necessary to install a *head node*, assemble and customize a *compute* image, and provision several compute hosts from bare-metal. With these steps completed, additional OpenHPC-provided packages can now be added to support a flexible HPC development environment including development tools, C/C++/FORTRAN compilers, MPI stacks, and a variety of 3rd party libraries. The following subsections highlight the additional software installation procedures.

8.1 Development Tools

To aid in general development efforts, OpenHPC provides recent versions of the GNU autotools collection, the Valgrind memory debugger, EasyBuild, and Spack. These can be installed as follows:

```
# Install autotools meta-package
dnf -y install ohpc-autotools
dnf -y install EasyBuild-ohpc
dnf -y install hwloc-ohpc
dnf -y install spack-ohpc
dnf -y install valgrind-ohpc
```

8.2 Compilers

OpenHPC presently packages the GNU compiler toolchain integrated with the underlying environment modules system in a hierarchical fashion. The modules system will conditionally present compiler-dependent software based on the toolchain currently loaded.

```
dnf -y install gnu16-compilers-ohpc
```

8.3 MPI Stacks

For MPI development and runtime support, OpenHPC provides pre-packaged builds for a variety of MPI families and transport layers. Currently available options and their applicability to various network transports are summarized in the Available MPI variants table below. The command that follows installs a starting set of MPI families that are compatible with both ethernet and high-speed fabrics.

Table: Available MPI builds

	Ethernet (TCP)	InfiniBand
MPICH	✓	

	Ethernet (TCP)	InfiniBand
OpenMPI	✓	✓

```
dnf -y install openmpi5-pmix-gnu16-ohpc mpich-ofi-gnu16-ohpc
```

Note that the MPICH and OpenMPI builds support two related transport layers for a variety of underlying fabrics: UCX (Unified Communication X) and OFI (OpenFabrics interfaces). In the case of OpenMPI, a monolithic build is provided which supports both transports and end-users can customize their runtime preferences with environment variables. For MPICH, two separate builds are provided and the example above highlighted installing the `ofi` variant. However, the packaging is designed such that both versions can be installed simultaneously and users can switch between the two via normal module command semantics. Alternatively, a site can choose to install the `ucx` variant instead as a drop-in MPICH replacement:

```
dnf -y install mpich-ucx-gnu16-ohpc
```

In the case where both MPICH variants are installed, two modules will be visible in the end-user environment and an example of this configuration is highlighted is below.

```
module avail mpich

-------------------- /opt/ohpc/pub/moduledeps/gnu16----------------------
   mpich/3.4.3-ofi    mpich/3.4.3-ucx (D)
```

8.4 Performance Tools

OpenHPC provides a variety of open-source tools to aid in application performance analysis (refer to Package Manifest for a listing of available packages). This group of tools can be installed as follows:

```
# Install perf-tools meta-package
dnf -y install ohpc-gnu16-perf-tools
```

8.5 Setup default development environment

System users often find it convenient to have a default development environment in place so that compilation can be performed directly for parallel programs requiring MPI. This setup can be conveniently enabled via modules and the OpenHPC modules environment is pre-configured to load an `ohpc` module on login (if present). The following package install provides a default environment that enables autotools, the GNU compiler toolchain, and the OpenMPI stack.

```
dnf -y install modules-defaults-gnu16-openmpi5-ohpc
```

Tip

If you want to change the default environment from the suggestion above, OpenHPC also provides additional default options using the GNU compiler toolchain with multiple MPICH variants. Relevant modules-default packages names are as follows:

- modules-defaults-gnu16-mpich-ofi-ohpc
- modules-defaults-gnu16-mpich-ucx-ohpc

8.6 3rd Party Libraries and Tools

OpenHPC provides pre-packaged builds for a number of popular open-source tools and libraries used by HPC applications and developers. For example, OpenHPC provides builds for FFTW and HDF5 (including serial and parallel I/O support), and the GNU Scientific Library (GSL). Again, multiple builds of each package are available in the OpenHPC repository to support multiple compiler and MPI family combinations where appropriate. Note, however, that not all combinatorial permutations may be available for components where there are known license incompatibilities. The general naming convention for builds provided by OpenHPC is to append the compiler and MPI family name that the library was built against directly into the package name. For example, libraries that do not require MPI as part of the build process adopt the following RPM name:

```
package-<compiler_family>-ohpc-<package_version>-<release>.rpm
```

Packages that do require MPI as part of the build expand upon this convention to additionally include the MPI family name as follows:

```
package-<compiler_family>-<mpi_family>-ohpc-<package_version>-<release>.rpm
```

To illustrate this further, the command below queries the locally configured repositories to identify all of the available PETSc packages that were built with the GNU toolchain. The resulting output that is included shows that pre-built versions are available for each of the supported MPI families presented in MPI Stacks.

```
dnf search petsc-gnu16 ohpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
============================ N/S matched: petsc-gnu16, ohpc ============================
petsc-gnu16-mpich-ohpc.x86_64 : Portable Extensible Toolkit for Scientific Computation
petsc-gnu16-openmpi5-ohpc.x86_64 : Portable Extensible Toolkit for Scientific Comp...
```

Tip

OpenHPC-provided 3rd party builds are configured to be installed into a common top-level repository so that they can be easily exported to desired hosts within the cluster. This common top-level path (`/opt/ohpc/pub`) was previously configured to be mounted on **compute** nodes in an earlier section, so the packages will be immediately available for use on the cluster after installation on the **head node**.

For convenience, OpenHPC provides package aliases for these 3rd party libraries and utilities that can be used to install available libraries for use with the GNU compiler family toolchain. For parallel libraries, aliases are grouped by MPI family toolchain so that administrators can choose a subset should they favor a

particular MPI stack. Please refer to the Package Manifest appendix for a more detailed listing of all available packages in each of these functional areas. To install all available package offerings within OpenHPC, issue the following:

```
# Install 3rd party libraries/tools meta-packages built with GNU toolchain
dnf -y install ohpc-gnu16-serial-libs
dnf -y install ohpc-gnu16-io-libs
dnf -y install ohpc-gnu16-python-libs
dnf -y install ohpc-gnu16-runtimes
```

```
# Install parallel lib meta-packages for all available MPI toolchains
dnf -y install ohpc-gnu16-mpich-parallel-libs
dnf -y install ohpc-gnu16-openmpi5-parallel-libs
```

9 Post Provisioning

9.1 Check Cluster Status

After this, check status of the nodes within Slurm by using the `sinfo` command. All compute nodes should be in an *idle* state (without asterisk). If the state is reported as *unknown*, the following might help:

```
scontrol update partition=normal state=idle
```

In case of additional Slurm issues, ensure that the configuration file fits your hardware and that it is identical across the nodes. Also, verify that the Slurm user id is the same on the head node and *compute* nodes. You may also consult Slurm Troubleshooting Guide.

9.2 Add User

We will now add a new user to the cluster. This will be used later to run a test job.

```
useradd -m test
```

Since OpenCHAMI uses NFS-mounted `/home`, the new user's home directory will be immediately available on all compute nodes.

9.3 Update NHC

Generate NHC configuration file based on compute node environment

```
pdsh -w c1 "/usr/sbin/nhc-genconf -H '*' -c -" | dshbak -c
```

10 Test Cluster

We now run some simple tests on the cluster to ensure it is operational.

10.1 Run a Test Job

With the resource manager enabled for production usage, users should now be able to run jobs. To demonstrate this, we will add a "test" user on the **head** node that can be used to run an example job.

OpenHPC includes a simple "hello-world" MPI application in the `/opt/ohpc/pub/examples` directory that can be used for this quick compilation and execution. OpenHPC also provides a companion job-launch utility named `prun` that is installed in concert with the pre-packaged MPI toolchains. This convenience script provides a mechanism to abstract job launch across different resource managers and MPI stacks such that a single launch command can be used for parallel job launch in a variety of OpenHPC environments. It also provides a centralizing mechanism for administrators to customize desired environment settings for their users.

10.2 Interactive Execution

To use the newly created "test" account to compile and execute the application **interactively** through the resource manager, execute the following (note the use of `prun` for parallel job launch which summarizes the underlying native job launch mechanism being used):

```
# Switch to "test" user
su - test

# Compile MPI "hello world" example
[test@sms ~]$ mpicc -O3 /opt/ohpc/pub/examples/mpi/hello.c

# Submit interactive job request and use prun to launch executable
[test@sms ~]$ salloc -n 8 -N 2

[test@c1 ~]$ prun ./a.out

[prun] Master compute host = c1
[prun] Resource manager = slurm
[prun] Launch cmd = mpiexec.hydra -bootstrap slurm ./a.out

 Hello, world (8 procs total)
    --> Process #   0 of   8 is alive. -> c1
    --> Process #   4 of   8 is alive. -> c2
    --> Process #   1 of   8 is alive. -> c1
    --> Process #   5 of   8 is alive. -> c2
    --> Process #   2 of   8 is alive. -> c1
    --> Process #   6 of   8 is alive. -> c2
    --> Process #   3 of   8 is alive. -> c1
    --> Process #   7 of   8 is alive. -> c2
```

Tip

The following table provides approximate command equivalences between SLURM and OpenPBS:

Command	OpenPBS	SLURM
Submit *batch* job	qsub [job script]	sbatch [job script]
Request *interactive* shell	qsub -I /bin/bash	salloc
Delete job	qdel [job id]	scancel [job id]
Queue status	qstat -q	sinfo
Job status	qstat -f [job id]	scontrol show job [job id]
Node status	pbsnodes [node name]	scontrol show node [node id]

10.3 Batch execution

For batch execution, OpenHPC provides a simple job script for reference (also housed in the `/opt/ohpc/pub/examples` directory. This example script can be used as a starting point for submitting batch jobs to the resource manager and the example below illustrates use of the script to submit a batch job for execution using the same executable referenced in the previous interactive example.

```
# Copy example job script
[test@sms ~]$ cp /opt/ohpc/pub/examples/slurm/job.mpi .

# Examine contents (and edit to set desired job sizing characteristics)
[test@sms ~]$ cat job.mpi
#!/bin/bash

#SBATCH -J test                # Job name
#SBATCH -o job.%j.out          # Name of stdout output file (%j expands to %jobId)
#SBATCH -N 2                   # Total number of nodes requested
#SBATCH -n 16                  # Total number of mpi tasks #requested
#SBATCH -t 01:30:00            # Run time (hh:mm:ss) - 1.5 hours

# Launch MPI-based executable

prun ./a.out

# Submit job for batch execution
[test@sms ~]$ sbatch job.mpi
Submitted batch job 339
```

Tip

The use of the `%j` option in the example batch job script shown is a convenient way to track application output on an individual job basis. The `%j` token is replaced with the Slurm job allocation number once assigned (job #339 in this example).

Appendices

A Installation Template

This appendix highlights the availability of a companion installation script that is included with OpenHPC documentation. This script, when combined with local site inputs, can be used to implement a starting recipe for bare-metal system installation and configuration. This template script is used during validation efforts to test cluster installations and is provided as a convenience for administrators as a starting point for potential site customization.

Tip

Note that the template script provided is intended for use during initial installation and is not designed for repeated execution. If modifications are required after using the script initially, we recommend running the relevant subset of commands interactively.

The template script relies on the use of a simple text file to define local site variables that were outlined in the Inputs section. By default, the template installation script attempts to use local variable settings sourced from the `/opt/ohpc/pub/doc/recipes/vanilla/input.local` file, however, this choice can be overridden by the use of the `${OHPC_INPUT_LOCAL}` environment variable. The template install script is intended for execution on the **head node** and is installed as part of the `docs-ohpc` package into `/opt/ohpc/pub/doc/recipes/vanilla/recipe.sh`. After enabling the OpenHPC repository and reviewing the guide for additional information on the intent of the commands, the general starting approach for using this template is as follows:

1. Install the `docs-ohpc` package

```
dnf -y install docs-ohpc
```

2. Copy the provided template input file to use as a starting point to define local site settings:

```
cp /opt/ohpc/pub/doc/recipes/rocky10/input.local input.local
```

3. Update `input.local` with desired settings
4. Copy the template installation script which contains command-line instructions culled from this guide.

```
cp -p /opt/ohpc/pub/doc/recipes/rocky10/aarch64/openchami/slurm/recipe.sh .
```

5. Review and edit `recipe.sh` to suite.
6. Use environment variable to define local input file and execute `recipe.sh` to perform a local installation.

```
export OHPC_INPUT_LOCAL=./input.local
./recipe.sh
```

B Upgrading OpenHPC Packages

As newer OpenHPC releases are made available, users are encouraged to upgrade their locally installed packages against the latest repository versions to obtain access to bug fixes and newer component versions. This can be accomplished with the underlying package manager as OpenHPC packaging maintains versioning state across releases. Also, package builds available from the OpenHPC repositories have "`-ohpc`" appended to their names so that wild cards can be used as a simple way to obtain updates. The following general procedure highlights a method for upgrading existing installations. When upgrading from a minor release older than v4, you will first need to update your local OpenHPC repository configuration to point against the v4 release (or update your locally hosted mirror). Refer to an earlier section for more details on enabling the latest repository. In contrast, when upgrading between micro releases on the same branch (e.g. from v4 to 4.2), there is no need to adjust local package manager configurations when using the public repository as rolling updates are pre-configured.

1. (Optional) Ensure repo metadata is current on the head node. Package managers will naturally do this on their own over time, but if you are wanting to access updates immediately after a new release, the following can be used to sync to the latest.

```
dnf clean expire-cache
```

2. Upgrade head node

```
dnf -y upgrade "*-ohpc"

# Any new Base OS provided dependencies can be installed by
# updating the ohpc-base metapackage
dnf -y upgrade "ohpc-base"
```

3. Rebuild compute image layers

Update the image layer YAML configurations as needed, then rebuild the production compute image and reboot compute nodes to apply the changes.

```
podman run \
    --rm \
    --device /dev/fuse \
    --network host \
    -e S3_ACCESS=admin \
    -e S3_SECRET=password1234 \
    -v /opt/ohpc/admin/images/data:/data \
    -v /opt/ohpc/admin/images/compute-prod.yaml:/home/builder/config.yaml \
    ghcr.io/openchami/image-build:latest \
        image-build \
        --config config.yaml
```

After the image is rebuilt and published to S3, reboot the compute nodes to apply the changes.

C Integration Test Suite

This appendix details the installation and basic use of the integration test suite used to support OpenHPC releases. This suite is not intended to replace the validation performed by component development teams, but is instead, devised to confirm component builds are functional and interoperable within the modular OpenHPC environment. The test suite is generally organized by components and the OpenHPC CI workflow relies on running the full suite using Jenkins to test multiple OS configurations and installation recipes. To facilitate customization and running of the test suite locally, we provide these tests in a standalone RPM.

```
dnf -y install test-suite-ohpc
```

The RPM installation creates a user named `ohpc-test` to house the test suite and provide an isolated environment for execution. Configuration of the test suite is done using standard GNU autotools semantics and the BATS shell-testing framework is used to execute and log a number of individual unit tests. Some tests require privileged execution, so a different combination of tests will be enabled depending on which user executes the top-level `configure` script. Non-privileged tests requiring execution on one or more compute nodes are submitted as jobs through the SLURM resource manager. The tests are further divided into "short" and "long" run categories. The short run configuration is a subset of approximately 180 tests to demonstrate basic functionality of key components (e.g. MPI stacks) and should complete in 10-20 minutes. The long run (around 1000 tests) is comprehensive and can take an hour or more to complete.

Most components can be tested individually, but a default configuration is setup to enable collective testing. To test an isolated component, use the `configure` option to disable all tests, then re-enable the desired test to run. The `--help` option to `configure` will display all possible tests. By default, the test suite will endeavor to run tests for multiple MPI stacks where applicable. To restrict tests to only a subset of MPI families, use the `--with-mpi-families` option (e.g. `--with-mpi-families="openmpi4"`). Example output is shown below (some output is omitted for the sake of brevity).

```
su - ohpc-test
[test@sms ~]$ cd tests
[test@sms ~]$ ./configure --disable-all --enable-fftw
checking for a BSD-compatible install... /bin/install -c
checking whether build environment is sane... yes

----------------------------------- SUMMARY -----------------------------------

Package version.............. : test-suite-2.0.0

Build user................... : ohpc-test
Build host................... : sms001
Configure date............... : 2020-10-05 08:22
Build architecture........... : aarch64
Compiler Families............ : gnu9
MPI Families................. : mpich openmpi4
Python Families.............. : python3
Resource manager ............ : SLURM
Test suite configuration..... : short

Libraries:
    Adios ................... : disabled
    Boost ................... : disabled
    Boost MPI................ : disabled
    FFTW..................... : enabled
    GSL...................... : disabled
    HDF5..................... : disabled
    HYPRE.................... : disabled
```

Many OpenHPC components exist in multiple flavors to support multiple compiler and MPI runtime permutations, and the test suite takes this in to account by iterating through these combinations by default. If `make check` is executed from the top-level test directory, all configured compiler and MPI permutations of a library will be exercised. The following highlights the execution of the FFTW related tests that were enabled in the previous step.

```
[test@sms ~]$ make check
make --no-print-directory check-TESTS
PASS: libs/fftw/ohpc-tests/test_mpi_families
============================================================================
Testsuite summary for test-suite 2.0.0
============================================================================
# TOTAL: 1
# PASS:  1
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
[test@sms ~]$ cat libs/fftw/tests/family-gnu*/rm_execution.log
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (SLURM/gnu9/mpich)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/mpich)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/mpich)
PASS rm_execution (exit status: 0)
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (SLURM/gnu9/openmpi4)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/openmpi4)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/...
PASS rm_execution (exit status: 0)
```

D Customization

D.1 Adding local environment modules to OpenHPC hierarchy

Administrators may wish to add locally built software packages to the OpenHPC module hierarchy. This can be accomplished by creating module files in the appropriate locations under `/opt/ohpc/pub/moduledeps` or `/opt/ohpc/pub/modulefiles`. Two sample module files are included in the `examples-ohpc` package—one representing an application with no compiler or MPI runtime dependencies, and one dependent on OpenMPI and the GNU toolchain. Simply copy these files to the prescribed locations, and the `module` command should pick them up automatically. The example output below is from Lmod; Environment Modules lists the same modules in its own format.

```
# Create a simple example module
mkdir /opt/ohpc/pub/modulefiles/example1
cp /opt/ohpc/pub/examples/example.modulefile \
    /opt/ohpc/pub/modulefiles/example1/1.0

# Create an example module with a dependency
mkdir /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2
cp /opt/ohpc/pub/examples/example-mpi-dependent.modulefile \
    /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2/1.0

# Show modules
module avail
```

Example Output

```
---------------------- /opt/ohpc/pub/moduledeps/gnu7-openmpi3 -----------------------
   adios/1.12.0    imb/2018.0          netcdf-fortran/4.4.4    ptscotch/6.0.4
   boost/1.65.1    mpi4py/2.0.0        netcdf/4.4.1.1          scalapack/2.0.2
   example2/1.0    mpiP/3.4.1          petsc/3.7.6             scalasca/2.3.1
   fftw/3.3.6      mumps/5.1.1         phdf5/1.10.1            scipy/0.19.1
   hypre/2.11.2    netcdf-cxx/4.3.0    pnetcdf/1.8.1           scorep/3.1

--------------------------- /opt/ohpc/pub/moduledeps/gnu7 ---------------------------
   R/3.4.2        metis/5.1.0     ocr/1.0.1              pdtoolkit/3.24
   gsl/2.4        mpich/3.2       openblas/0.2.20        plasma/2.8.0
   hdf5/1.10.1    numpy/1.13.1    openmpi3/3.0.0  (L)    scotch/6.0.4

-------------------------- /opt/ohpc/admin/modulefiles --------------------------
   spack/0.10.0

------------------------------ /opt/ohpc/pub/modulefiles ------------------------------
   EasyBuild/3.4.1         cmake/3.9.2         hwloc/1.11.8        pmix/1.2.3
   autotools        (L)    example1/1.0 (L)    llvm5/5.0.0         prun/1.2        (L)
   clustershell/1.8        gnu7/7.2.0   (L)    ohpc         (L)    singularity/2.4

  Where:
   L:  Module is loaded

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of
the "keys".
```

D.2 Rebuilding Packages from Source

OpenHPC packages can be rebuilt from source using the source RPMs available from the OpenHPC repository. This allows administrators to customize package builds for their specific needs. One way to accomplish this is to install the appropriate source RPM, modify the spec file as needed, and rebuild to obtain an updated binary RPM. OpenHPC spec files contain macros to facilitate local customizations of compiler, compilation flags and MPI family. A brief example using the FFTW library is highlighted below. Note that the source RPMs can be downloaded from the community repository server at http://repos.openhpc.community via a web browser or directly via dnf as highlighted below. In this example we make an explicit change to FFTW's configuration, as well as modifying the CFLAGS environment variable. The package is also tagged with an additional delimiter to allow easy co-installation and use.

```
# Install rpm-build package and dnf tools from base OS distro
sudo dnf -y install rpm-build dnf-plugins-core

# Install FFTW's build dependencies
sudo dnf builddep fftw-gnu16-openmpi5-ohpc

# Download SRPM from OpenHPC repository and install locally
dnf download --source fftw-gnu16-openmpi5-ohpc
rpm -i ./fftw-gnu16-openmpi5-ohpc-*.rpm

# Modify spec file as desired
cd ~/rpmbuild/SPECS
sed -i "s/enable-static=no/enable-static=yes/" fftw.spec

# Increment RPM release so the package manager will see an update
sed -i "s/Release:   400.ohpc.3.1/Release:   400.ohpc.3.2/" fftw.spec

# Rebuild binary RPM. Note that additional directives can be specified to modify build
rpmbuild -bb --define "OHPC_CFLAGS '-O3 -mtune=native'" \
        --define "OHPC_CUSTOM_DELIM static" fftw.spec

# Install the new package
sudo dnf -y install \
        ../RPMS/$(uname -m)/fftw-gnu16-openmpi5-static-ohpc-*.$(uname -m).rpm
```

The new module file now appears along side the default.

```
$ module -t spider fftw
fftw/3.3.10-static
fftw/3.3.10
```

E Package Manifest

This appendix provides a summary of available meta-package groupings and all of the individual RPM packages that are available as part of this OpenHPC release. The meta-packages provide a mechanism to group related collections of RPMs by functionality and provide a convenience mechanism for installation. A list of the available meta-packages and a brief description is presented in the table below.

E.1 Available OpenHPC Meta-packages

Group Name	Description
ohpc-arm1-io-libs	Collection of IO library builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-mpich-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the MPICH runtime.
ohpc-arm1-openmpi5-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the openmpi5 runtime.
ohpc-arm1-perf-tools	Collection of performance tool builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-python3-libs	Collection of python3 related library builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-serial-libs	Collection of serial library builds for use with the Arm Compiler for Linux toolchain.
ohpc-autotools	Collection of GNU autotools packages.
ohpc-base	Collection of base packages.
ohpc-base-compute	Collection of compute node base packages.
ohpc-flux-client	Collection of client packages for Flux.
ohpc-flux-server	Collection of server packages for Flux.
ohpc-gnu15-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu15-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-openmpi5-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-openmpi5-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-openmpi5-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu15-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu15-python-libs	Collection of python related library builds for use with GNU compiler toolchain.

Group Name	Description
ohpc-gnu15-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu15-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu15-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-gnu16-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu16-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-openmpi5-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-openmpi5-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-openmpi5-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu16-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu16-python-libs	Collection of python related library builds for use with GNU compiler toolchain.
ohpc-gnu16-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu16-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu16-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-slurm-client	Collection of client packages for SLURM.
ohpc-slurm-server	Collection of server packages for SLURM.
ohpc-warewulf	Collection of base packages for Warewulf provisioning.

E.2 RPM Packages

What follows next in this Appendix is a series of tables that summarize the underlying RPM packages available in this OpenHPC release. These packages are organized by groupings based on their general functionality and each table provides information for the specific RPM name, version, brief summary, and the web URL where additional information can be obtained for the component. Note that many of the 3rd party community libraries that are pre-packaged with OpenHPC are built using multiple compiler and MPI families. In these cases, the RPM package name includes delimiters identifying the development environment for which each package build is targeted. Additional information on the OpenHPC package naming scheme is presented in Section 3rd Party Packages. The relevant package groupings and associated references are as follows:

- Administrative Tools

- Resource Management
- Compiler Families
- MPI Families / Communication Libraries
- Development Tools
- Performance Analysis Tools
- IO Libraries
- Distro Packages
- Runtimes
- Serial/Threaded Libraries
- Parallel Libraries

E.2.1 Administrative Tools

RPM Package Name	Version	Info/URL
conman	0.3.1	ConMan: The Console Manager. http://dun.github.io/conman
docs	4.0.0	OpenHPC documentation. https://github.com/openhpc/ohpc
docs	4.2.0	OpenHPC documentation. https://github.com/openhpc/ohpc
environment-modules	5.7.0	Provides dynamic modification of a user's environment. https://envmodules.io
examples	2.0	Example source code and templates for use within OpenHPC environment. https://github.com/openhpc/ohpc
genders	1.32	Static cluster configuration database. https://github.com/chaos/genders
hpc-workspace	1.6.3	Temporary workspace management. https://github.com/holgerBerger/hpc-workspace
lmod-defaults	2.0	OpenHPC default login environments. https://github.com/openhpc/ohpc
lmod	9.4.5	Lua based Modules (lmod). https://github.com/TACC/Lmod
losf	0.56.0	A Linux operating system framework for managing HPC clusters. https://github.com/hpcsi/losf
modules-defaults	2.1	OpenHPC default login environments. https://github.com/openhpc/ohpc
nhc	1.4.3	LBNL Node Health Check. https://github.com/mej/nhc
ohpc-release	4	OpenHPC release files. https://github.com/openhpc/ohpc
pdsh	2.36	Parallel remote shell program. https://github.com/chaos/pdsh
prun	3	Convenience utility for parallel job launch. https://github.com/openhpc/ohpc

RPM Package Name	Version	Info/URL
test-suite	4.2.0	Integration test suite for OpenHPC. https://github.com/openhpc/ohpc

E.2.2 Resource Management

RPM Package Name	Version	Info/URL
flux-core	0.89.0	Core components of the Flux resource manager framework. https://github.com/flux-framework/flux-core
flux	0.7.1	PMIx support for the Flux resource manager. https://github.com/flux-framework/flux-pmix
flux-sched	0.55.0	Fluxion graph-based scheduler for the Flux resource manager. https://github.com/flux-framework/flux-sched
flux-security	0.15.0	Independent security component for the Flux resource manager. https://github.com/flux-framework/flux-security
magpie	3.2	Scripts for running Big Data software in HPC environments. https://github.com/LLNL/magpie
openpbs-client	23.06.06	OpenPBS for a client host. http://www.openpbs.org
openpbs-execution	23.06.06	OpenPBS for an execution host. http://www.openpbs.org
openpbs-server	23.06.06	OpenPBS for a server host. http://www.openpbs.org
pmix	4.2.9	An extended/exascale implementation of PMI. https://pmix.org
slurm-contribs	26.05.5	Perl tool to print Slurm job state information. https://slurm.schedmd.com
slurm-devel	26.05.5	Development package for Slurm. https://slurm.schedmd.com
slurm-example-configs	26.05.5	Example config files for Slurm. https://slurm.schedmd.com
slurm-libpmi	26.05.5	Slurm's implementation of the pmi libraries. https://slurm.schedmd.com
slurm	26.05.5	Slurm Workload Manager. https://slurm.schedmd.com
slurm-openlava	26.05.5	openlava/LSF wrappers for transition from OpenLava/LSF to Slurm. https://slurm.schedmd.com
slurm-pam_slurm	26.05.5	PAM module for restricting access to compute nodes via Slurm. https://slurm.schedmd.com
slurm-perlapi	26.05.5	Perl API to Slurm. https://slurm.schedmd.com
slurm-sackd	26.05.5	Slurm authentication daemon. https://slurm.schedmd.com
slurm-slurmctld	26.05.5	Slurm controller daemon. https://slurm.schedmd.com
slurm-slurmd	26.05.5	Slurm compute node daemon. https://slurm.schedmd.com
slurm-slurmdbd	26.05.5	Slurm database daemon. https://slurm.schedmd.com
slurm-sview	26.05.5	Graphical user interface to view and modify Slurm state. https://slurm.schedmd.com
slurm-torque	26.05.5	Torque/PBS wrappers for transition from Torque/PBS to Slurm. https://slurm.schedmd.com

E.2.3 Compiler Families

RPM Package Name	Version	Info/URL
gnu15-compilers	15.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org
gnu16-compilers	16.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org

E.2.4 MPI Families / Communication Libraries

RPM Package Name	Version	Info/URL
mpich	5.0.1	MPICH MPI implementation. http://www.mpich.org
openmpi5	5.0.10	A powerful implementation of MPI/SHMEM. http://www.open-mpi.org
ucx	1.22.0	UCX is a communication library implementing high-performance messaging. http://www.openucx.org

E.2.5 Development Tools

RPM Package Name	Version	Info/URL
EasyBuild	5.4.1	Software build and installation framework. https://easybuild.io
autoconf	2.71	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
automake	1.16.5	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake
cmake	4.4.4	CMake is an open-source, cross-platform family of tools designed to build, test and package software. https://cmake.org
cuda-devel	25.9	OpenHPC compatibility package for cuda. https://github.com/openhpc/ohpc
cuda-repo	25.9	Cuda toolkit online repository. https://github.com/openhpc/ohpc
hwloc	2.15.0	Portable Hardware Locality. http://www.open-mpi.org/projects/hwloc
libtool	2.4.6	The GNU Portable Library Tool. http://www.gnu.org/software/libtool
python3-mpi4py	4.1.1	Python bindings for the Message Passing Interface (MPI) standard. https://github.com/mpi4py/mpi4py
python3-numpy	2.4.4	NumPy array processing for numbers, strings, records and objects. https://github.com/numpy/numpy
spack	1.2.2	HPC software package management. https://github.com/spack/spack
valgrind	3.27.1	Valgrind Memory Debugger. http://www.valgrind.org

E.2.6 Performance Analysis Tools

RPM Package Name	Version	Info/URL
dimemas	5.5.0	Dimemas tool. https://tools.bsc.es
extrae	5.0.4	Extrae tool. https://tools.bsc.es
imb	2021.11	Intel MPI Benchmarks (IMB). https://software.intel.com/en-us/articles/intel-mpi-benchmarks
likwid	5.5.1	Performance tools for the Linux console. https://github.com/RRZE-HPC/likwid
omb	7.5.2	OSU Micro-benchmarks. https://mvapich.cse.ohio-state.edu/benchmarks
papi	7.2.0	Performance Application Programming Interface. http://icl.cs.utk.edu/papi
paraver	4.12.0	Paraver. https://tools.bsc.es
pdtoolkit	3.25.1	PDT is a framework for analyzing source code. http://www.cs.uoregon.edu/Research/pdt
scalasca	2.6.2	Toolset for performance analysis of large-scale parallel applications. http://www.scalasca.org
scorep	9.4	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.vi-hps.org/projects/score-p
tau	2.35.1	Tuning and Analysis Utilities Profiling Package. http://www.cs.uoregon.edu/research/tau/home.php

E.2.7 IO Libraries

RPM Package Name	Version	Info/URL
adios2	2.12.0	The Adaptable IO System v2 (ADIOS2). https://adios2.readthedocs.io/en/latest/index.html
cubew	4.9.1	CUBE Uniform Behavioral Encoding generic presentation writer component. http://www.scalasca.org/software/cube-4.x/download.html
hdf5	2.1.1	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx	4.3.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran	4.6.2	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf	4.10.0	C Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
otf2	3.1.1	Open Trace Format 2 library. http://score-p.org
phdf5	2.1.1	A general purpose library and file format for storing scientific data (parallel version). http://www.hdfgroup.org/HDF5
pnetcdf	1.14.1	A Parallel NetCDF library (PnetCDF). http://cucis.ece.northwestern.edu/projects/PnetCDF

RPM Package Name	Version	Info/URL
sionlib	1.7.7	Scalable I/O Library for Parallel Access to Task-Local Files. https://apps.fz-juelich.de/jsc/sionlib/docu/index.html

E.2.8 Distro Packages

RPM Package Name	Version	Info/URL
python3-Cython	3.3.0	The Cython compiler for writing C extensions for the Python language. http://www.cython.org

E.2.9 Runtimes

RPM Package Name	Version	Info/URL
charliecloud	0.45.2	Lightweight user-defined software stacks for high-performance computing. https://charliecloud.io

E.2.10 Serial/Threaded Libraries

RPM Package Name	Version	Info/URL
R	4.5.3	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
cubelib	4.9.1	CUBE Uniform Behavioral Encoding generic presentation library component. http://www.scalasca.org/software/cube-4.x/download.html
gotcha	1.0.8	A library for wrapping function calls to shared libraries. https://github.com/llnl/gotcha
gsl	2.8	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
opari2	2.0.9	An OpenMP runtime performance measurement instrumenter. https://www.vi-hps.org/projects/score-p
openblas	0.3.32	An optimized BLAS library based on GotoBLAS2. http://www.openblas.net
plasma	25.5.27	Parallel Linear Algebra Software for Multicore Architectures. https://github.com/icl-utk-edu/plasma
scotch	7.0.11	Graph, mesh and hypergraph partitioning library. https://gitlab.inria.fr/scotch/scotch
superlu	7.0.1	A general purpose library for the direct solution of linear equations. http://crd.lbl.gov/~xiaoye/SuperLU

E.2.11 Parallel Libraries

RPM Package Name	Version	Info/URL
boost	1.90.0	Free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw	3.3.10	A Fast Fourier Transform library. http://www.fftw.org
hypre	3.1.0	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mfem	4.9	Lightweight, general, scalable C++ library for finite element methods. http://mfem.org
mumps	5.8.2	A MUltifrontal Massively Parallel Sparse direct Solver. https://mumps-solver.org
petsc	3.25.0	Portable Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc
ptscotch	7.0.11	Graph, mesh and hypergraph partitioning library using MPI. https://gitlab.inria.fr/scotch/scotch
scalapack	2.2.3	A subset of LAPACK routines redesigned for heterogeneous computing. https://netlib.org/scalapack
slepc	3.25.0	A library for solving large scale sparse eigenvalue problems. http://slepc.upv.es
superlu_dist	9.2.1	A general purpose library for the direct solution of linear equations. https://portal.nersc.gov/project/sparse/superlu
trilinos	17.0.0	A collection of libraries of numerical algorithms. https://trilinos.org

F Package Signatures

All of the RPMs provided via the OpenHPC repository are signed with a GPG signature. By default, the underlying package managers will verify these signatures during installation to ensure that packages have not been altered. The RPMs can also be manually verified and the public signing key fingerprint for the latest repository is shown below:

```
Fingerprint: 5E33 3CA3 A1BD BBC9 DF14 9D74 09AD FAE4 **D722A692**
```

The following command can be used to verify an RPM once it has been downloaded locally by confirming if the package is signed, and if so, indicating which key was used to sign it. The example below highlights usage for a local copy of the `docs-ohpc` package and illustrates how the **key ID** matches the fingerprint shown above.

```
rpm --checksig -v ohpc-release-4-1.el10.x86_64.rpm
ohpc-release-4-1.el10.x86_64.rpm:
    Header V3 RSA/SHA256 Signature, key ID d722a692: OK
    Header SHA256 digest: OK
    Header SHA1 digest: OK
    Payload SHA256 digest: OK
    V3 RSA/SHA256 Signature, key ID d722a692: OK
    MD5 digest: OK
```