OpenHPC (v3.2)
Cluster Building Recipes

Rocky 9.4 Base OS

confluent/SLURM Edition for Linux* (x86_64)

Document Last Update: 2024-09-08
Document Revision: d731775a2

Legal Notice

Contents

1 Introduction **5**
- 1.1 Target Audience . . . 5
- 1.2 Requirements/Assumptions . . . 5
- 1.3 Inputs . . . 6

2 Install Base Operating System (BOS) **7**

3 Install Confluent and Provision Nodes with BOS **8**
- 3.1 Enable Confluent repository for local use . . . 8
- 3.2 Add provisioning services on *master* node . . . 8
- 3.3 Complete basic Confluent setup for *master* node . . . 8
- 3.4 Define *compute* image for provisioning . . . 9
  - 3.4.1 Build initial BOS image . . . 9
- 3.5 Add compute nodes into Confluent database . . . 9
- 3.6 Boot compute nodes . . . 10

4 Install OpenHPC Components **11**
- 4.1 Enable OpenHPC repository for local use . . . 11
- 4.2 Installation template . . . 12
- 4.3 Setup time synchronization service on *master* node . . . 12
- 4.4 Add resource management services on *master* node . . . 12
- 4.5 Optionally add InfiniBand support services on *master* node . . . 13
- 4.6 Optionally add Omni-Path support services on *master* node . . . 14
  - 4.6.1 Add OpenHPC components . . . 14
  - 4.6.2 Customize system configuration . . . 15
  - 4.6.3 Additional Customization (*optional*) . . . 16
    - 4.6.3.1 Increase locked memory limits . . . 16
    - 4.6.3.2 Enable ssh control via resource manager . . . 16
    - 4.6.3.3 Add Lustre client . . . 16
    - 4.6.3.4 Add ClusterShell . . . 17
    - 4.6.3.5 Add *genders* . . . 18
    - 4.6.3.6 Add Magpie . . . 18
    - 4.6.3.7 Add ConMan . . . 18
    - 4.6.3.8 Add NHC . . . 18

5 Install OpenHPC Development Components **19**
- 5.1 Development Tools . . . 19
- 5.2 Compilers . . . 19
- 5.3 MPI Stacks . . . 19
- 5.4 Performance Tools . . . 20
- 5.5 Setup default development environment . . . 20
- 5.6 3rd Party Libraries and Tools . . . 21
- 5.7 Optional Development Tool Builds . . . 22

6 Resource Manager Startup **23**

7 Run a Test Job **23**
- 7.1 Interactive execution . . . 24
- 7.2 Batch execution . . . 25

Appendices **26**
A Installation Template . . . 26
B Upgrading OpenHPC Packages . . . 27
C Integration Test Suite . . . 28
D Customization . . . 30
D.1 Adding local Lmod modules to OpenHPC hierarchy . . . 30
D.2 Rebuilding Packages from Source . . . 31
E Package Manifest . . . 32
F Package Signatures . . . 48

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 stateless 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:

```
[sms]# 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 *master*, and four *compute* nodes. The *master* node serves as the overall system management server (SMS) and is provisioned with Rocky 9.4 and is subsequently configured to provision the remaining *compute* nodes with confluent in a stateless configuration. The terms *master* and SMS are used interchangeably in this guide. For power management, we assume that the compute node baseboard management controllers (BMCs) are available via IPMI from the chosen master host. For file systems, we assume that the chosen master server will host an NFS file system that is made available to the compute nodes. Installation information is also discussed to optionally mount a parallel file system and in this case, the parallel file system is assumed to exist previously.

An outline of the physical architecture discussed is shown in Figure 1 and highlights the high-level networking configuration. The *master* host 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

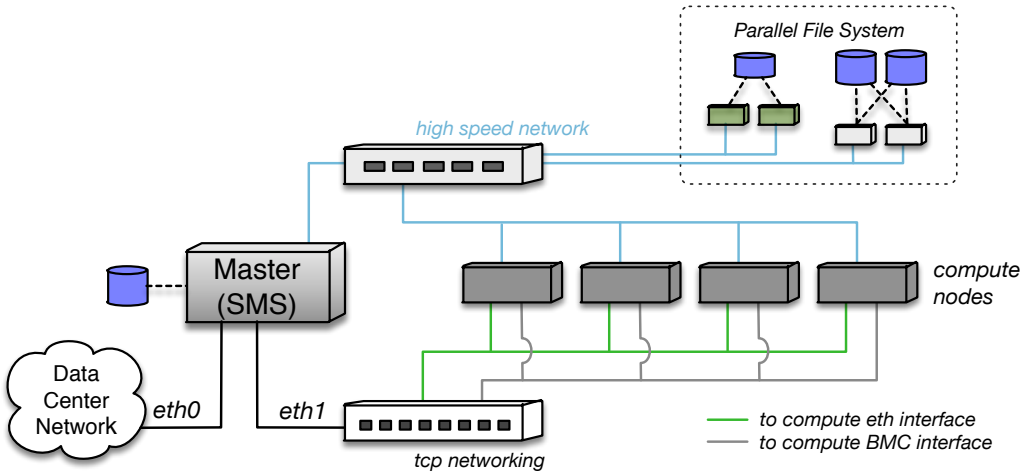

Figure 1: Overview of physical cluster architecture.

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 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.

In addition to the IP networking, there is an optional high-speed network (InfiniBand or Omni-Path in this recipe) that is also connected to each of the hosts. This high speed network is used for application message passing and optionally for parallel file system connectivity as well (e.g. to existing Lustre or BeeGFS storage targets).

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.

- ${sms_name} # Hostname for SMS server
- ${sms_ip} # Internal IP address on SMS server
- ${sms_eth_internal} # Internal Ethernet interface on SMS
- ${internal_netmask} # Subnet netmask for internal network
- ${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:

- ${sysmgmtd_host} # BeeGFS System Management host name
- ${mgs_fs_name} # Lustre MGS mount name
- ${sms_ipoib} # IPoIB address for SMS server
- ${ipoib_netmask} # Subnet netmask for internal IPoIB
- ${c_ipoib[0]}, ${c_ipoib[1]}, ... # IPoIB addresses for computes

2 Install Base Operating System (BOS)

In an external setting, installing the desired BOS on a *master* SMS host typically involves booting from a DVD ISO image on a new server. With this approach, insert the Rocky 9.4 DVD, power cycle the host, and follow the distro provided directions to install the BOS on your chosen *master* host. Alternatively, if choosing to use a pre-installed server, please verify that it is provisioned with the required Rocky 9.4 distribution.

Prior to beginning the installation process of OpenHPC components, several additional considerations are noted here for the SMS host configuration. First, the installation recipe herein assumes that the SMS host name is resolvable locally. Depending on the manner in which you installed the BOS, there may be an adequate entry already defined in `/etc/hosts`. If not, the following addition can be used to identify your SMS host.

```
[sms]# echo ${sms_ip} ${sms_name} >> /etc/hosts
```

While it is theoretically possible to enable SELinux on a cluster provisioned with confluent, 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 *master* SMS host. 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.

Finally, provisioning services rely on DHCP, TFTP, and HTTP network protocols. Depending on the local BOS configuration on the SMS host, default firewall rules may prohibit these services. Consequently, this recipe assumes that the local firewall running on the SMS host 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:

```
[sms]# systemctl disable firewalld
[sms]# systemctl stop firewalld
```

3 Install Confluent and Provision Nodes with BOS

Installation is accomplished in two steps: First, a generic OS image is installed on *compute* nodes and then, once the nodes are up and running, OpenHPC components are added to both the SMS and the nodes at the same time.

3.1 Enable Confluent repository for local use

To begin, enable use of the public Confluent repository by adding it to the local list of available package repositories. This also requires network access from your *master* server to the internet, or alternatively, that the repository be mirrored locally. In this case, we use the network.

```
[sms]# dnf -y install https://hpc.lenovo.com/yum/latest/el9/x86_64/lenovo-hpc-yum-1-1.x86_64.rpm
```

3.2 Add provisioning services on *master* node

With Confluent repository enabled, issue the following install the provisioning service on *master* node

```
[sms]# dnf -y install lenovo-confluent
[sms]# dnf -y install tftp-server

# enable Confluent and its tools for use in current shell
[sms]# systemctl enable confluent --now
[sms]# systemctl enable httpd --now
[sms]# systemctl enable tftp.socket --now
[sms]# source /etc/profile.d/confluent_env.sh
```

3.3 Complete basic Confluent setup for *master* node

At this point, all of the packages necessary to use Confluent on the *master* host should be installed. Next, we enable support for local provisioning using a second private interface (refer to Figure 1)

```
# Enable internal interface for provisioning
[sms]# ip link set dev ${sms_eth_internal} up
[sms]# ip address add ${sms_ip}/${internal_netmask} broadcast + dev ${sms_eth_internal}
```

Confluent requires a network domain name specification for system-wide name resolution. This value can be set to match your local DNS schema or given a unique identifier such as 'local'. A default group called everything is automatically added to every node. It provides a method to indicate global settings. Attributes may all be specified on the command line, and an example set could be:

```
[sms]# nodegroupattrib everything deployment.useinsecureprotocols=${deployment_protocols} dns.domain=${dns_domain}
[sms]# nodegroupattrib everything dns.servers=${dns_servers} net.ipv4_gateway=${ipv4_gateway}
```

We will also define

3.4 Define *compute* image for provisioning

With the provisioning services enabled, the next step is to define a system image that can subsequently be used to provision one or more *compute* nodes. The following subsections highlight this process.

3.4.1 Build initial BOS image

The following steps illustrate the process to build a minimal, default image for use with Confluent. To begin, you will first need to have a local copy of the ISO image available for the underlying OS. In this recipe, the relevant ISO image is `Rocky-9.4-x86_64-dvd.iso` (available from the Rocky download page). We initialize the image creation process using the `osdeploy` command assuming that the necessary ISO image is available locally in `${iso_path}` as follows:

The `osdeploy initialize` command is used to prepare a confluent server to deploy deploy operating systems. For first time setup, run osdeploy initialize interactively to be walked through the various options using: `osdeploy initialize -i`

```
[sms]# osdeploy initialize -${initialize_options}
[sms]# osdeploy import ${iso_path}
```

Once completed, OS image should be available for use within Confluent. These can be queried via:

```
# Query available images
[sms]# osdeploy list
Distributions:
  rocky-8.5-x86_64
  rocky-9.4-x86_64
Profiles:
  rhel-9.4-x86_64-default
  rocky-8.5-x86_64-default
```

If needing to copy files from the sms node to the compute nodes during deployment, this can be done by modifying the syncfiles file that is created when `osdeploy import` command is run. For an environment that has no DNS server and needs to have /etc/hosts file synced amongst all the nodes, the following command should be run.

```
[sms]# echo "/etc/hosts -> /etc/hosts" >> /var/lib/confluent/public/os/rocky-9.4-x86_64-default/syncfiles
```

3.5 Add compute nodes into Confluent database

Next, we add *compute* nodes and define their properties as attributes in Confluent database. These hosts are grouped logically into a group named *compute* to facilitate group-level commands used later in the recipe. The compute group has to be defined first before we can add any nodes to the group using the nodegroup define command. Note the use of variable names for the desired compute hostnames, node IPs, MAC addresses, and BMC login credentials, which should be modified to accommodate local settings and hardware. To enable serial console access via Confluent, console.method property is also defined.

```
#define the compute group
[sms]# nodegroupdefine compute

# Define nodes as objects in confluent database
[sms]# for ((i=0; i<$num_computes; i++)) ; do
```

```
nodedefine ${c_name[$i]} groups=everything,compute hardwaremanagement.manager=${c_bmc[$i]} \
secret.hardwaremanagementuser=${bmc_username} secret.hardwaremanagementpassword=${bmc_password} \
net.hwaddr=${c_mac[$i]} net.ipv4_address=${c_ip[$i]}
        done
```

Tip

Defining nodes one-by-one, as done above, is only efficient for a small number of nodes. For larger node counts, Confluent provides capabilities for automated detection and configuration. Consult the Confluent Hardware Discovery & Define Node Guide.

If enabling *optional* IPoIB functionality (e.g. to support Lustre over InfiniBand), additional settings are required to define the IPoIB network with Confluent and specify desired IP settings for each compute. This can be accomplished as follows for the *ib0* interface:

```
# Register desired IPoIB IPs per compute
[sms]# for ((i=0; i<$num_computes; i++)) ; do
               nodeattrib ${c_name[i]} net.ib0.ipv4_address=${c_ipoib[i]}/${ipoib_netmask}
        done
```

confluent2hosts can be used to help generate /etc/hosts entries for a noderange. It can read from the confluent db, using -a. In this mode, each net.value.attribute group is pulled together into hosts lines. ipv4 and ipv6 address fields are associated with the corresponding hostname attributes.

```
# Add nodes to /etc/hosts
[sms]# confluent2hosts -a compute
```

With the desired compute nodes and domain identified, the remaining steps in the provisioning configuration process are to define the provisioning mode and image for the *compute* group and use Confluent commands to complete configuration for network services like DNS and DHCP. These tasks are accomplished as follows:

```
# Associate desired provisioning image for computes
[sms]# nodedeploy -n compute -p rocky-9.4-x86_64-default
```

3.6 Boot compute nodes

Prior to booting the *compute* hosts, we configure them to use PXE as their next boot mode. After the initial PXE, ensuing boots will return to using the default boot device specified in the BIOS.

```
[sms]# nodesetboot compute network
```

At this point, the *master* server should be able to boot the newly defined compute nodes. This is done by using the `nodepower` Confluent command leveraging IPMI protocol set up during the the *compute* node definition in § 3.5. The following power cycles each of the desired hosts.

```
[sms]# nodepower compute boot
```

Once kicked off, the boot process should take about 5-10 minutes (depending on BIOS post times). You can monitor the provisioning by using the `nodeconsole` command, which displays serial console for a selected node. Note that the escape sequence is `CTRL-e c .` typed sequentially.

Successful provisioning can be verified by a parallel command on the compute nodes. The Confluent-provided `nodeshell` command, which uses Confluent node names and groups. For example, to run a command on the newly imaged compute hosts using `nodeshell`, execute the following:

```
[sms]# nodeshell compute uptime
c1:  12:56:50 up 14 min, 0 users, load average: 0.00, 0.01, 0.04
c2:  12:56:50 up 13 min, 0 users, load average: 0.00, 0.02, 0.05
c3:  12:56:50 up 14 min, 0 users, load average: 0.00, 0.02, 0.05
c4:  12:56:50 up 14 min, 0 users, load average: 0.00, 0.01, 0.04
```

4 Install OpenHPC Components

With the BOS installed and booted, the next step is to add desired OpenHPC packages onto the *master* server in order to provide provisioning and resource management services for the rest of the cluster. The following subsections highlight this process.

4.1 Enable OpenHPC repository for local use

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 *master* server 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.

```
# Add OpenHPC repo
[sms]# dnf -y install http://repos.openhpc.community/OpenHPC/3/EL_9/x86_64/ohpc-release-3-1.el9.x86_64.rpm
```

Tip

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/3.2

In addition to the OpenHPC package repository, the *master* host also requires access to the standard base OS distro repositories in order to resolve necessary dependencies. For Rocky 9.4, the requirements are to have access to the BaseOS, Appstream, Extras, CRB, and EPEL repositories for which mirrors are freely available online:

- Rocky-9 (e.g. http://download.rockylinux.org/pub/rocky/9/)
- EPEL 9 (e.g. http://download.fedoraproject.org/pub/epel/9/)

The public EPEL repository will be enabled automatically upon installation of the `ohpc-release` package. Note that this does depend on the Rocky Extras repository, which is shipped with Rocky 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:

```
[sms]# dnf install dnf-plugins-core
[sms]# dnf config-manager --set-enabled crb
```

```
[sms]# dnf -y install epel-release

# Enable crb on sms
[sms]# /usr/bin/crb enable
```

Now OpenHPC packages can be installed. To add the base package on the SMS issue the following

```
[sms]# dnf -y install ohpc-base
```

4.2 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 (§ 1.3) and is provided as a convenience for administrators. For additional information on accessing this script, please see Appendix A.

4.3 Setup time synchronization service on *master* node

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 SMS host with a specific server `${ntp_server}`, and allow this server to serve as a local time server for the cluster, issue the following:

```
[sms]# systemctl enable chronyd.service
[sms]# echo "local stratum 10" >> /etc/chrony.conf
[sms]# echo "server ${ntp_server}" >> /etc/chrony.conf
[sms]# echo "allow all" >> /etc/chrony.conf
[sms]# systemctl restart chronyd
```

Tip

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 SMS host. 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
```

4.4 Add resource management services on *master* node

OpenHPC provides multiple options for distributed resource management. The following command adds the Slurm workload manager server components to the chosen *master* host. Note that client-side components will be added to the corresponding compute image in a subsequent step.

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

# Use ohpc-provided file for starting SLURM configuration
[sms]# cp /etc/slurm/slurm.conf.ohpc /etc/slurm/slurm.conf
# Setup default cgroups file
[sms]# cp /etc/slurm/cgroup.conf.example /etc/slurm/cgroup.conf

# Identify resource manager hostname on master host
[sms]# perl -pi -e "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

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.

Other versions of this guide are available that describe installation of alternate resource management systems, and they can be found in the `docs-ohpc` package.

4.5 Optionally add InfiniBand support services on *master* node

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

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

Tip

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 9.4 provides the `opensm` package should you choose to run it on the *master* 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 (see §4.6.3.3 for additional details), then having IPoIB enabled is a requirement for the Lustre client. OpenHPC provides a template configuration file to aid in setting up an *ib0* interface on the *master* host. To use, copy the template provided and update the `${sms_ipoib}` and `${ipoib_netmask}` entries to match local desired settings (alter ib0 naming as appropriate if system contains dual-ported or multiple HCAs).

```
[sms]# cp /opt/ohpc/pub/examples/network/centos/ifcfg-ib0 /etc/sysconfig/network-scripts

# Define local IPoIB address and netmask
[sms]# perl -pi -e "s/master_ipoib/${sms_ipoib}/" /etc/sysconfig/network-scripts/ifcfg-ib0
[sms]# perl -pi -e "s/ipoib_netmask/${ipoib_netmask}/" /etc/sysconfig/network-scripts/ifcfg-ib0

# configure NetworkManager to *not* override local /etc/resolv.conf
[sms]# echo "[main]" >  /etc/NetworkManager/conf.d/90-dns-none.conf
[sms]# echo "dns=none" >> /etc/NetworkManager/conf.d/90-dns-none.conf
# Start up NetworkManager to initiate ib0
[sms]# systemctl start NetworkManager
```

4.6 Optionally add Omni-Path support services on *master* node

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

```
[sms]# dnf -y install opa-basic-tools
```

> **Tip**
>
> Omni-Path 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 9.4 provides the `opa-fm` package should you choose to run it on the *master* node.

4.6.1 Add OpenHPC components

The next step is adding OpenHPC components to the *compute* nodes that at this point are running basic OSes. This process will leverage two Confluent-provided commands: `nodeshell` to run `dnf` installer on all the nodes in parallel and `nodersync` to distribute configuration files from the SMS to the *compute* nodes. To do this, repositories on the *compute* nodes need to be configured properly.

Confluent has automatically setup an OS repository on the SMS and configured the nodes to use it, but it has also enabled online OS repositories.
Next, we alse add the OHPC repo to the compute nodes §4.1

```
# Add OpenHPC repo
[sms]# nodeshell compute dnf -y install http://repos.openhpc.community/OpenHPC/3/EL_9/x86_64/ohpc-release-3-1.el9.x86_64.rpm
```

The *compute* nodes also need access to the EPEL repository, a required dependency for OpenHPC packages.

```
# Add epel repo
[sms]# nodeshell compute dnf -y install epel-release
```

Additionally, a workaround is needed for OpenHPC documentation files, which are installed into a read-only NFS share /opt/ohpc/pub. Any package attempting to write to that directory will fail to install. The following prevents that by directing `rpm` not to install documentation files on the *compute* nodes:

```
[sms]# nodeshell compute echo -e %_excludedocs 1 \>\> ~/.rpmmacros
```

Now OpenHPC and other cluster-related software components can be installed on the nodes. The first step is to install a base compute package:

```
# Install compute node base meta-package
[sms]# nodeshell compute dnf -y install ohpc-base-compute
```

Next, we can include additional components:

```
# Add Slurm client support meta-package
[sms]# nodeshell compute dnf -y install ohpc-slurm-client

# Add Network Time Protocol (NTP) support
[sms]# nodeshell compute dnf -y install ntp

# Add kernel drivers
[sms]# nodeshell compute dnf -y install kernel

# Enable crb
[sms]# nodeshell compute /usr/bin/crb enable

# Include nfs-utils
[sms]# nodeshell compute dnf -y install nfs-utils

# Include modules user environment
[sms]# nodeshell compute dnf -y install lmod-ohpc
```

```
# Optionally add IB support and enable
[sms]# nodeshell compute dnf -y groupinstall "InfiniBand Support"
```

4.6.2 Customize system configuration

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 *master* host in this example configuration.

```
# Disable /tftpboot and /install export entries
[sms]# perl -pi -e "s|/tftpboot|#/tftpboot|" /etc/exports
[sms]# perl -pi -e "s|/install|#/install|" /etc/exports

# Export /home and OpenHPC public packages from master server
[sms]# echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >> /etc/exports
[sms]# echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports
[sms]# exportfs -a
[sms]# systemctl restart nfs-server
[sms]# systemctl enable nfs-server


# Create NFS client mounts of /home and /opt/ohpc/pub on compute hosts
[sms]# nodeshell compute echo \
       "\""${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0"\"" \>\> /etc/fstab
[sms]# nodeshell compute echo \
```

```
       "\""${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev 0 0"\"" \>\> /etc/fstab
[sms]# nodeshell compute systemctl restart nfs

# Mount NFS shares
[sms]# nodeshell compute mount /home
[sms]# nodeshell compute mkdir -p /opt/ohpc/pub
[sms]# nodeshell compute mount /opt/ohpc/pub
```

4.6.3 Additional Customization (*optional*)

This section highlights common additional customizations that can *optionally* be applied to the local cluster environment. These customizations include:

- Add InfiniBand or Omni-Path drivers
- Increase memlock limits
- Restrict ssh access to compute resources
- Add BeeGFS client
- Add Lustre client
- Add ClusterShell
- Add *mrsh*
- Add *genders*
- Add ConMan
- Add GEOPM

Details on the steps required for each of these customizations are discussed further in the following sections.

4.6.3.1 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 updating the `/etc/security/limits.conf` file and this should be performed on all job submission hosts. In this recipe, jobs are submitted from the *master* host, and the following commands can be used to update the maximum locked memory settings on both the master host and compute nodes:

```
# Update memlock settings on master
[sms]# perl -pi -e 's/# End of file/\* soft memlock unlimited\n$&/s' /etc/security/limits.conf
[sms]# perl -pi -e 's/# End of file/\* hard memlock unlimited\n$&/s' /etc/security/limits.conf
# Update memlock settings on compute
[sms]# nodeshell compute perl -pi -e "'s/# End of file/\* soft memlock unlimited\n$&/s' \
          /etc/security/limits.conf"
[sms]# nodeshell compute perl -pi -e "'s/# End of file/\* hard memlock unlimited\n$&/s' \
          /etc/security/limits.conf"
```

4.6.3.2 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:

```
[sms]# nodeshell compute echo "\""account required pam_slurm.so"\"" \>\> /etc/pam.d/sshd
```

4.6.3.3 Add Lustre client To add Lustre client support on the cluster, it necessary to install the client and associated modules on each host needing to access a Lustre file system. In this recipe, it is assumed that the Lustre file system is hosted by servers that are pre-existing and are not part of the install process. Outlining the variety of Lustre client mounting options is beyond the scope of this document, but the general requirement is to add a mount entry for the desired file system that defines the management server (MGS) and underlying network transport protocol. To add client mounts on both the *master* server and *compute* image, the following commands can be used. Note that the Lustre file system to be mounted is identified

by the ${mgs_fs_name} variable. In this example, the file system is configured to be mounted locally as /mnt/lustre.

```
# Add Lustre client software to master host
[sms]# dnf -y install lustre-client-ohpc
```

```
# Enable lustre on compute nodes
[sms]# nodeshell compute dnf -y install lustre-client-ohpc

# Create mount point and file system mount
[sms]# nodeshell compute mkdir /mnt/lustre
[sms]# nodeshell compute echo \
       "\""${mgs_fs_name} /mnt/lustre lustre defaults,_netdev,localflock,retry=2 0 0"\"" \>\> /etc/fstab
```

Tip

The suggested mount options shown for Lustre leverage the `localflock` option. This is a Lustre-specific setting that enables client-local flock support. It is much faster than cluster-wide flock, but if you have an application requiring cluster-wide, coherent file locks, use the standard `flock` attribute instead.

The default underlying network type used by Lustre is *tcp*. If your external Lustre file system is to be mounted using a network type other than *tcp*, additional configuration files are necessary to identify the desired network type. The example below illustrates creation of modprobe configuration files instructing Lustre to use an InfiniBand network with the **o2ib** LNET driver attached to `ib0`. Note that these modifications are made to both the *master* host and *compute* nodes.

```
[sms]# echo "options lnet networks=o2ib(ib0)" >> /etc/modprobe.d/lustre.conf
[sms]# nodeshell compute echo "\""options lnet networks=o2ib\(ib0\)"\"" \>\> /etc/modprobe.d/lustre.conf
```

With the Lustre configuration complete, the client can be mounted on the *master* and *compute* hosts as follows:

```
[sms]# mkdir /mnt/lustre
[sms]# mount -t lustre -o localflock ${mgs_fs_name} /mnt/lustre

# Mount on compute nodes
[sms]# nodeshell compute mount /mnt/lustre
```

4.6.3.4 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
[sms]# dnf -y install clustershell

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

4.6.3.5 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
[sms]# dnf -y install genders-ohpc

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

4.6.3.6 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
[sms]# dnf -y install magpie-ohpc
```

4.6.3.7 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
[sms]# dnf -y install conman-ohpc

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

# Enable and start conman
[sms]# systemctl enable conman
[sms]# systemctl start conman
```

4.6.3.8 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 master and compute nodes
[sms]# dnf -y install nhc-ohpc
[sms]# nodeshell compute dnf -y install nhc-ohpc
```

```
# Register as SLURM's health check program
[sms]# echo "HealthCheckProgram=/usr/sbin/nhc" >> /etc/slurm/slurm.conf
[sms]# echo "HealthCheckInterval=300" >> /etc/slurm/slurm.conf # execute every five minutes
```

5 Install OpenHPC Development Components

The install procedure outlined in §4 highlighted the steps necessary to install a *master* host, 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.

5.1 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
[sms]# dnf -y install ohpc-autotools

[sms]# dnf -y install EasyBuild-ohpc
[sms]# dnf -y install hwloc-ohpc
[sms]# dnf -y install spack-ohpc
[sms]# dnf -y install valgrind-ohpc
```

5.2 Compilers

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

```
[sms]# dnf -y install gnu14-compilers-ohpc
```

5.3 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 Table 1. The command that follows installs a starting set of MPI families that are compatible with both ethernet and high-speed fabrics.

```
[sms]# dnf -y install openmpi5-pmix-gnu14-ohpc mpich-ofi-gnu14-ohpc
```

Note that OpenHPC 2.x introduces the use of two related transport layers for the MPICH and OpenMPI builds that support 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

Table 1: Available MPI variants

	Ethernet (TCP)	InfiniBand	Intel® Omni-Path
MPICH (ofi)	✓	✓	✓
MPICH (ucx)	✓	✓	✓
MVAPICH2		✓	
MVAPICH2 (psm2)			✓
OpenMPI (ofi/ucx)	✓	✓	✓

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:

```
[sms]# dnf -y install mpich-ucx-gnu14-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.

```
[sms]# module avail mpich

-------------------- /opt/ohpc/pub/moduledeps/gnu14---------------------
   mpich/3.4.3-ofi  mpich/3.4.3-ucx (D)
```

If your system includes InfiniBand and you enabled underlying support in §4.5 and §4.6.3, an additional MVAPICH2 family is available for use:

```
[sms]# dnf -y install mvapich2-gnu14-ohpc
```

Alternatively, if your system includes Intel® Omni-Path, use the (psm2) variant of MVAPICH2 instead:

```
[sms]# dnf -y install mvapich2-psm2-gnu14-ohpc
```

5.4 Performance Tools

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

```
# Install perf-tools meta-package
[sms]# dnf -y install ohpc-gnu14-perf-tools
```

5.5 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.

```
[sms]# dnf -y install lmod-defaults-gnu14-openmpi5-ohpc
```

Tip

If you want to change the default environment from the suggestion above, OpenHPC also provides the GNU compiler toolchain with the MPICH and MVAPICH2 stacks:

- lmod-defaults-gnu14-mpich-ofi-ohpc
- lmod-defaults-gnu14-mpich-ucx-ohpc
- lmod-defaults-gnu14-mvapich2-ohpc

5.6 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 §5.3.

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 §4.6.2, so the packages will be immediately available for use on the cluster after installation on the *master* host.

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 Appendix E 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
[sms]# dnf -y install ohpc-gnu14-serial-libs
[sms]# dnf -y install ohpc-gnu14-io-libs
[sms]# dnf -y install ohpc-gnu14-python-libs
```

```
[sms]# dnf -y install ohpc-gnu14-runtimes
```

```
# Install parallel lib meta-packages for all available MPI toolchains
[sms]# dnf -y install ohpc-gnu14-mpich-parallel-libs
[sms]# dnf -y install ohpc-gnu14-openmpi5-parallel-libs
```

5.7 Optional Development Tool Builds

In addition to the 3rd party development libraries built using the open source toolchains mentioned in §5.6, OpenHPC also provides *optional* compatible builds for use with the compilers and MPI stack included in newer versions of the Intel® oneAPI HPC Toolkit (using the *classic* compiler variants). These packages provide a similar hierarchical user environment experience as other compiler and MPI families present in OpenHPC.

To take advantage of the available builds, OpenHPC provides a convenience package to enable the oneAPI repository locally along with compatibility packages that integrate oneAPI-generated compiler and MPI modulefiles within the standard OpenHPC user environment. To enable the Intel® oneAPI repository and install minimum compiler and MPI requirements for OpenHPC packaging, issue the following:

```
# Enable Intel oneAPI and install OpenHPC compatibility packages
[sms]# dnf -y install intel-oneapi-toolkit-release-ohpc
[sms]# rpm --import https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
[sms]# dnf -y install intel-compilers-devel-ohpc
[sms]# dnf -y install intel-mpi-devel-ohpc
```

Tip

As noted in §4.6.2, the default installation path for OpenHPC (`/opt/ohpc/pub`) is exported over NFS from the *master* to the compute nodes, but the Intel® oneAPI HPC Toolkit packages install to a top-level path of `/opt/intel`. To make the Intel® compilers available to the compute nodes one must add an additional NFS export for `/opt/intel` that is mounted on desired compute nodes.

To enable all 3rd party builds available in OpenHPC that are compatible with the Intel® oneAPI classic compiler suite, issue the following:

```
# Optionally, choose the Omni-Path enabled build for MVAPICH2. Otherwise, skip to retain IB variant
[sms]# dnf -y install mvapich2-psm2-intel-ohpc
```

```
# Install 3rd party libraries/tools meta-packages built with Intel toolchain
[sms]# dnf -y install openmpi5-pmix-intel-ohpc
[sms]# dnf -y install ohpc-intel-serial-libs
[sms]# dnf -y install ohpc-intel-geopm
[sms]# dnf -y install ohpc-intel-io-libs
[sms]# dnf -y install ohpc-intel-perf-tools
[sms]# dnf -y install ohpc-intel-python3-libs
[sms]# dnf -y install ohpc-intel-mpich-parallel-libs
[sms]# dnf -y install ohpc-intel-mvapich2-parallel-libs
[sms]# dnf -y install ohpc-intel-openmpi5-parallel-libs
[sms]# dnf -y install ohpc-intel-impi-parallel-libs
```

6 Resource Manager Startup

In section §4, the Slurm resource manager was installed and configured on the *master* host. Slurm clients were also installed, but have not been configured yet. To do so, Slurm and `munge` configuration files need to be copied to the nodes. This can be accomplished as follows:

```
[sms]# nodersync /etc/slurm/slurm.conf compute:/etc/slurm/slurm.conf
[sms]# nodersync /etc/munge/munge.key compute:/etc/munge/munge.key
```

With Slurm configured, we can now startup the resource manager services in preparation for running user jobs. Generally, this is a two-step process that requires starting up the controller daemons on the *master* host and the client daemons on each of the *compute* hosts. 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. The following commands can be used to startup the necessary services to support resource management under Slurm.

```
# Start munge and slurm controller on master host
[sms]# systemctl enable munge
[sms]# systemctl enable slurmctld
[sms]# systemctl start munge
[sms]# systemctl start slurmctld

# Start slurm clients on compute hosts
[sms]# nodeshell compute systemctl enable munge
[sms]# nodeshell compute systemctl enable slurmd
[sms]# nodeshell compute systemctl start munge
[sms]# nodeshell compute systemctl start slurmd
```

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:

```
[sms]# 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 Slurm user id is the same on the SMS and *compute* nodes. You may also consult Slurm Troubleshooting Guide.

7 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 *master* host that can be used to run an example job.

```
[sms]# useradd -m test
```

Next, the user's credentials need to be distributed across the cluster. Confluent's `nodeappy` has a merge functionality that adds new entries into credential files on *compute* nodes:

```
# Create a sync file for pushing user credentials to the nodes
[sms]# echo "/etc/passwd -> /etc/passwd" >> /var/lib/confluent/public/os/rocky-9.4-x86_64-default/syncfiles
[sms]# echo "/etc/group -> /etc/group" >> /var/lib/confluent/public/os/rocky-9.4-x86_64-default/syncfiles

# Use Confluent to distribute credentials to nodes
[sms]# nodeapply -F compute
```

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.

7.1 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
[sms]# 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]

7.2 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 §1.3. 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 SMS *master* host 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

```
[sms]# dnf -y install docs-ohpc
```

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

```
[sms]# cp /opt/ohpc/pub/doc/recipes/rocky9/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.

```
[sms]# cp -p /opt/ohpc/pub/doc/recipes/rocky9/x86_64/confluent/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.

```
[sms]# export OHPC_INPUT_LOCAL=./input.local
[sms]# ./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 v3, you will first need to update your local OpenHPC repository configuration to point against the v3 release (or update your locally hosted mirror). Refer to §4.1 for more details on enabling the latest repository. In contrast, when upgrading between micro releases on the same branch (e.g. from v3 to 3.2), there is no need to adjust local package manager configurations when using the public repository as rolling updates are pre-configured.

The initial step, when using a local mirror as described in §4.1, is downloading a new tarball from `http://build.openhpc.community/dist`. After this, move (or remove) the old repository directory, unpack the new tarball and run the `make_repo.sh` script. In the following, substitute `x.x.x` with the desired version number.

```
[sms]# wget http://repos.openhpc.community/dist/x.x.x/OpenHPC-x.x.x.EL_9.x86_64.tar
[sms]# mv $ohpc_repo_dir "$ohpc_repo_dir"_old
[sms]# mkdir -p $ohpc_repo_dir
[sms]# tar xvf OpenHPC-x.x.x.EL_9.x86_64.tar -C $ohpc_repo_dir
[sms]# $ohpc_repo_dir/make_repo.sh
```

The *compute* hosts will pickup new packages automatically as long as the repository location on the SMS stays the same. If you prefer to use a different location, setup repositories on the nodes as described in §4.1.

The actual update is performed as follows:

1. (Optional) Ensure repo metadata is current (on head and compute nodes.) 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.

   ```
   [sms]# dnf clean expire-cache
   [sms]# nodeshell compute dnf clean expire-cache
   ```

2. Upgrade master (SMS) node

   ```
   [sms]# dnf -y upgrade "*-ohpc"
   ```

3. Upgrade packages on compute nodes

   ```
   [sms]# nodeshell compute dnf -y upgrade "*-ohpc"
   ```

4. Rebuild image(s)

Note that to update running services such as a resource manager a service restart (or a full reboot) is required.

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.

```
[sms]# 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).

```
[sms]# 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........... : x86_64
Compiler Families............ : gnu9
MPI Families................. : mpich mvapich2 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/mvapich2)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/mvapich2)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/mvapich2)
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/openmpi4)
PASS rm_execution (exit status: 0)
```

D Customization

D.1 Adding local Lmod modules to OpenHPC hierarchy

Locally installed applications can easily be integrated in to OpenHPC systems by following the Lmod convention laid out by the provided packages. 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 `lmod` application should pick them up automatically.

```
[sms]# mkdir /opt/ohpc/pub/modulefiles/example1
[sms]# cp /opt/ohpc/pub/examples/example.modulefile \
   /opt/ohpc/pub/modulefiles/example1/1.0
[sms]# mkdir /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2
[sms]# cp /opt/ohpc/pub/examples/example-mpi-dependent.modulefile \
   /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2/1.0
[sms]# module avail

---------------------------------- /opt/ohpc/pub/moduledeps/gnu7-openmpi3 ----------------------------------
   adios/1.12.0   imb/2018.0        netcdf-fortran/4.4.4  ptscotch/6.0.4   sionlib/1.7.1
   boost/1.65.1   mpi4py/2.0.0      netcdf/4.4.1.1        scalapack/2.0.2  slepc/3.7.4
   example2/1.0   mpiP/3.4.1        petsc/3.7.6           scalasca/2.3.1   superlu_dist/4.2
   fftw/3.3.6     mumps/5.1.1       phdf5/1.10.1          scipy/0.19.1     tau/2.26.1
   hypre/2.11.2   netcdf-cxx/4.3.0  pnetcdf/1.8.1         scorep/3.1       trilinos/12.10.1

------------------------------------- /opt/ohpc/pub/moduledeps/gnu7 -------------------------------------
   R/3.4.2       metis/5.1.0    ocr/1.0.1            pdtoolkit/3.24  superlu/5.2.1
   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           valgrind/3.13.0
   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

Users of OpenHPC may find it desirable to rebuild one of the supplied packages to apply build customizations or satisfy local requirements. 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
[test@sms ~]$ sudo dnf -y install rpm-build dnf-plugins-core

# Install FFTW's build dependencies
[test@sms ~]$ sudo dnf builddep fftw-gnu9-openmpi4-ohpc

# Download SRPM from OpenHPC repository and install locally
[test@sms ~]$ dnf download --source fftw-gnu9-openmpi4-ohpc
[test@sms ~]$ rpm -i ./fftw-gnu9-openmpi4-ohpc-3.3.8-5.1.ohpc.2.0.src.rpm

# Modify spec file as desired
[test@sms ~]$ cd ~/rpmbuild/SPECS
[test@sms ~rpmbuild/SPECS]$ perl -pi -e "s/enable-static=no/enable-static=yes/" fftw.spec

# Increment RPM release so the package manager will see an update
[test@sms ~rpmbuild/SPECS]$ perl -pi -e "s/Release: 5.1/Release: 6.1/" fftw.spec

# Rebuild binary RPM. Note that additional directives can be specified to modify build
[test@sms ~rpmbuild/SPECS]$ rpmbuild -bb --define "OHPC_CFLAGS '-O3 -mtune=native'" \
       --define "OHPC_CUSTOM_DELIM static" fftw.spec

# Install the new package
[test@sms ~rpmbuild/SPECS]$ sudo dnf -y install \
       ../RPMS/x86_64/fftw-gnu9-openmpi4-static-ohpc.2.0-3.3.8-6.1.x86_64.rpm

# The new module file appears along side the default
[test@sms ~]$ module avail fftw

--------------------------/opt/ohpc/pub/moduledeps/gnu9-openmpi4 ---------------------------
   fftw/3.3.8-static  fftw/3.3.8 (D)
```

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 Table 2.

Table 2: **Available OpenHPC Meta-packages**

Group Name	Description
ohpc-autotools	Collection of GNU autotools packages.
ohpc-base	Collection of base packages.
ohpc-base-compute	Collection of compute node base packages.
ohpc-gnu12-geopm	Global Extensible Open Power Manager for use with GNU compiler toolchain.
ohpc-gnu12-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu12-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu12-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu12-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu12-mvapich2-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu12-mvapich2-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu12-mvapich2-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu12-openmpi4-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu12-openmpi4-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu12-openmpi4-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu12-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu12-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu12-python-libs	Collection of python related library builds for use with GNU compiler toolchain.
ohpc-gnu12-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu12-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu12-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-gnu13-geopm	Global Extensible Open Power Manager for use with GNU compiler toolchain.
ohpc-gnu13-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu13-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu13-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu13-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu13-mvapich2-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu13-mvapich2-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu13-mvapich2-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MVAPICH2 runtime.
ohpc-gnu13-openmpi5-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu13-openmpi5-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu13-openmpi5-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu13-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu13-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu13-python-libs	Collection of python related library builds for use with GNU compiler toolchain.
ohpc-gnu13-python3-libs	Collection of python3 related library builds for use with GNU compiler

Table 2 (cont): **Available OpenHPC Meta-packages**

Group Name	Description
ohpc-intel-mvapich2-parallel-libs	Collection of parallel library builds for use with Intel(R) oneAPI Toolkit and the MVAPICH2 runtime.
ohpc-intel-mvapich2-perf-tools	Collection of performance tool builds for use with Intel(R) oneAPI Toolkit compiler toolchain and the MVAPICH2 runtime.
ohpc-intel-openmpi4-io-libs	Collection of IO library builds for use with Intel(R) oneAPI Toolkit and OpenMPI runtime.
ohpc-intel-openmpi4-parallel-libs	Collection of parallel library builds for use with Intel(R) oneAPI Toolkit and the OpenMPI runtime.
ohpc-intel-openmpi4-perf-tools	Collection of performance tool builds for use with Intel(R) oneAPI Toolkit compiler toolchain and the OpenMPI runtime.
ohpc-intel-openmpi5-io-libs	Collection of IO library builds for use with Intel(R) oneAPI Toolkit and OpenMPI runtime.
ohpc-intel-openmpi5-parallel-libs	Collection of parallel library builds for use with Intel(R) oneAPI Toolkit and the OpenMPI runtime.
ohpc-intel-openmpi5-perf-tools	Collection of performance tool builds for use with Intel(R) oneAPI Toolkit compiler toolchain and the OpenMPI runtime.
ohpc-intel-perf-tools	Collection of performance tool builds for use with Intel(R) oneAPI Toolkit.
ohpc-intel-python3-libs	Collection of python3 related library builds for use with Intel(R) oneAPI Toolkit.
ohpc-intel-serial-libs	Collection of serial library builds for use with Intel(R) oneAPI Toolkit.
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.

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 §5.6. The relevant package groupings and associated Table references are as follows:

- Administrative tools (Table 3)
- Resource management (Table 4)
- Compiler families (Table 5)
- MPI families (Table 6)
- Development tools (Table 7)
- Performance analysis tools (Table 8)
- Lustre (Table **??**)
- IO Libraries (Table 9)
- Runtimes (Table 10)
- Serial/Threaded Libraries (Table 11)
- Parallel Libraries (Table 12)

Table 3: **Administrative Tools**

RPM Package Name	Version	Info/URL
conman-ohpc	0.3.1	ConMan: The Console Manager. http://dun.github.io/conman
docs-ohpc	3.1.0	OpenHPC documentation. https://github.com/openhpc/ohpc
examples-ohpc	2.0	Example source code and templates for use within OpenHPC environment. https://github.com/openhpc/ohpc
genders-ohpc	1.27	Static cluster configuration database. https://github.com/chaos/genders
hpc-workspace-ohpc	1.4.0	Temporary workspace management. https://github.com/holgerBerger/hpc-workspace
lmod-defaults-gnu12-impi-ohpc lmod-defaults-gnu12-mpich-ofi-ohpc lmod-defaults-gnu12-mpich-ucx-ohpc lmod-defaults-gnu12-mvapich2-ohpc lmod-defaults-gnu12-openmpi4-ohpc lmod-defaults-gnu13-impi-ohpc lmod-defaults-gnu13-mvapich2-ohpc lmod-defaults-gnu13-openmpi5-ohpc lmod-defaults-intel-impi-ohpc lmod-defaults-intel-mvapich2-ohpc lmod-defaults-intel-openmpi4-ohpc lmod-defaults-intel-openmpi5-ohpc	2.0	OpenHPC default login environments. https://github.com/openhpc/ohpc
lmod-ohpc	8.7.37	Lua based Modules (lmod). https://github.com/TACC/Lmod
losf-ohpc	0.56.0	A Linux operating system framework for managing HPC clusters. https://github.com/hpcsi/losf
mrsh-ohpc	2.12	Remote shell program that uses munge authentication. https://github.com/chaos/mrsh
nhc-ohpc	1.4.3	LBNL Node Health Check. https://github.com/mej/nhc
ohpc-release	3	OpenHPC release files. https://github.com/openhpc/ohpc
pdsh-ohpc	2.35	Parallel remote shell program. https://github.com/chaos/pdsh
prun-ohpc	2.2	Convenience utility for parallel job launch. https://github.com/openhpc/ohpc
test-suite-ohpc	3.0.0	Integration test suite for OpenHPC. https://github.com/openhpc/ohpc/tests

Table 4: **Resource Management**

RPM Package Name	Version	Info/URL
magpie-ohpc	3.0	Scripts for running Big Data software in HPC environments. https://github.com/LLNL/magpie
openpbs-execution-ohpc	22.05.11	OpenPBS for an execution host. http://www.openpbs.org
openpbs-client-ohpc	22.05.11	OpenPBS for a client host. http://www.openpbs.org
openpbs-server-ohpc	22.05.11	OpenPBS for a server host. http://www.openpbs.org
pmix-ohpc	4.2.9	An extended/exascale implementation of PMI. https://pmix.github.io/pmix
slurm-devel-ohpc	23.11.6	Development package for Slurm. https://slurm.schedmd.com
slurm-example-configs-ohpc	23.11.6	Example config files for Slurm. https://slurm.schedmd.com
slurm-sview-ohpc	23.11.6	Graphical user interface to view and modify Slurm state. https://slurm.schedmd.com
slurm-pam_slurm-ohpc	23.11.6	PAM module for restricting access to compute nodes via Slurm. https://slurm.schedmd.com
slurm-perlapi-ohpc	23.11.6	Perl API to Slurm. https://slurm.schedmd.com
slurm-contribs-ohpc	23.11.6	Perl tool to print Slurm job state information. https://slurm.schedmd.com
slurm-ohpc	23.11.6	Slurm Workload Manager. https://slurm.schedmd.com
slurm-sackd-ohpc	23.11.6	Slurm authentication daemon. https://slurm.schedmd.com
slurm-slurmd-ohpc	23.11.6	Slurm compute node daemon. https://slurm.schedmd.com
slurm-slurmctld-ohpc	23.11.6	Slurm controller daemon. https://slurm.schedmd.com
slurm-slurmdbd-ohpc	23.11.6	Slurm database daemon. https://slurm.schedmd.com
slurm-libpmi-ohpc	23.11.6	Slurmś implementation of the pmi libraries. https://slurm.schedmd.com
slurm-torque-ohpc	23.11.6	Torque/PBS wrappers for transition from Torque/PBS to Slurm. https://slurm.schedmd.com
slurm-openlava-ohpc	23.11.6	openlava/LSF wrappers for transition from OpenLava/LSF to Slurm. https://slurm.schedmd.com

Table 5: **Compiler Families**

RPM Package Name	Version	Info/URL
gnu12-compilers-ohpc	12.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org
gnu13-compilers-ohpc	13.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org
intel-oneapi-toolkit-release-ohpc	2024.0	Intel(R) oneAPI HPC Toolkit Repository Setup. https://github.com/openhpc/ohpc
intel-psxe-compilers-devel-ohpc	2024.0	OpenHPC compatibility package for Intel(R) Parallel Studio XE. https://github.com/openhpc/ohpc
intel-compilers-devel-ohpc	2024.0	OpenHPC compatibility package for Intel(R) oneAPI HPC Toolkit. https://github.com/openhpc/ohpc

Table 6: **MPI Families / Communication Libraries**

RPM Package Name	Version	Info/URL
intel-psxe-mpi-devel-ohpc	2024.0	OpenHPC compatibility package for Intel(R) MPI Library. https://github.com/openhpc/ohpc
intel-mpi-devel-ohpc	2024.0	OpenHPC compatibility package for Intel(R) oneAPI MPI Library. https://github.com/openhpc/ohpc
libfabric-ohpc	1.18.0	User-space RDMA Fabric Interfaces. http://www.github.com/ofiwg/libfabric
mpich-ofi-gnu12-ohpc mpich-ofi-gnu13-ohpc mpich-ofi-intel-ohpc	3.4.3	MPICH MPI implementation. http://www.mpich.org
mpich-ucx-gnu12-ohpc mpich-ucx-gnu13-ohpc	3.4.3	MPICH MPI implementation. http://www.mpich.org
mvapich2-gnu12-ohpc mvapich2-gnu13-ohpc mvapich2-intel-ohpc	2.3.7	OSU MVAPICH2 MPI implementation. http://mvapich.cse.ohio-state.edu
openmpi4-gnu12-ohpc openmpi4-gnu13-ohpc openmpi4-intel-ohpc openmpi4-pmix-gnu12-ohpc openmpi4-pmix-intel-ohpc	4.1.5	A powerful implementation of MPI/SHMEM. http://www.open-mpi.org
openmpi5-gnu13-ohpc openmpi5-intel-ohpc openmpi5-pmix-gnu13-ohpc openmpi5-pmix-intel-ohpc	5.0.3	A powerful implementation of MPI/SHMEM. http://www.open-mpi.org
ucx-ohpc	1.15.0	UCX is a communication library implementing high-performance messaging. http://www.openucx.org

Table 7: **Development Tools**

RPM Package Name	Version	Info/URL
EasyBuild-ohpc	4.9.1	Software build and installation framework. https://easybuilders.github.io/easybuild
automake-ohpc	1.16.5	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake
autoconf-ohpc	2.71	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
cmake-ohpc	3.24.2	CMake is an open-source, cross-platform family of tools designed to build, test and package software. https://cmake.org
hwloc-ohpc	2.9.3	Portable Hardware Locality. http://www.open-mpi.org/projects/hwloc
libtool-ohpc	2.4.6	The GNU Portable Library Tool. http://www.gnu.org/software/libtool
python3-scipy-gnu12-impi-ohpc python3-scipy-gnu12-mpich-ohpc python3-scipy-gnu12-mvapich2-ohpc python3-scipy-gnu12-openmpi4-ohpc	1.5.4	Scientific Tools for Python. http://www.scipy.org
python3.11-scipy-gnu13-impi-ohpc python3.11-scipy-gnu13-mpich-ohpc python3.11-scipy-gnu13-mvapich2-ohpc python3.11-scipy-gnu13-openmpi5-ohpc	1.5.4	Scientific Tools for Python. http://www.scipy.org
python3-numpy-gnu12-ohpc python3-numpy-intel-ohpc	1.19.5	NumPy array processing for numbers, strings, records and objects. https://github.com/numpy/numpy
python3.11-numpy-gnu13-ohpc python3.11-numpy-intel-ohpc	1.26.4	NumPy array processing for numbers, strings, records and objects. https://github.com/numpy/numpy
python3-mpi4py-gnu12-impi-ohpc python3-mpi4py-gnu12-mpich-ohpc python3-mpi4py-gnu12-mvapich2-ohpc python3-mpi4py-gnu12-openmpi4-ohpc python3-mpi4py-intel-impi-ohpc python3-mpi4py-intel-mpich-ohpc python3-mpi4py-intel-mvapich2-ohpc python3-mpi4py-intel-openmpi4-ohpc	3.1.4	Python bindings for the Message Passing Interface (MPI) standard. https://github.com/mpi4py/mpi4py
python3.11-mpi4py-gnu13-impi-ohpc python3.11-mpi4py-gnu13-mpich-ohpc python3.11-mpi4py-gnu13-mvapich2-ohpc python3.11-mpi4py-gnu13-openmpi5-ohpc python3.11-mpi4py-intel-impi-ohpc python3.11-mpi4py-intel-mpich-ohpc python3.11-mpi4py-intel-mvapich2-ohpc python3.11-mpi4py-intel-openmpi5-ohpc	3.1.5	Python bindings for the Message Passing Interface (MPI) standard. https://github.com/mpi4py/mpi4py
spack-ohpc	0.21.2	HPC software package management. https://github.com/spack/spack
valgrind-ohpc	3.20.0	Valgrind Memory Debugger. http://www.valgrind.org

Table 8: **Performance Analysis Tools**

RPM Package Name	Version	Info/URL
dimemas-gnu12-impi-ohpc dimemas-gnu12-mpich-ohpc dimemas-gnu12-mvapich2-ohpc dimemas-gnu12-openmpi4-ohpc dimemas-gnu13-impi-ohpc dimemas-gnu13-mpich-ohpc dimemas-gnu13-mvapich2-ohpc dimemas-gnu13-openmpi5-ohpc dimemas-intel-impi-ohpc dimemas-intel-mpich-ohpc dimemas-intel-mvapich2-ohpc dimemas-intel-openmpi4-ohpc dimemas-intel-openmpi5-ohpc	5.4.2	Dimemas tool. https://tools.bsc.es
extrae-gnu12-impi-ohpc extrae-gnu12-mpich-ohpc extrae-gnu12-mvapich2-ohpc extrae-gnu12-openmpi4-ohpc extrae-gnu13-impi-ohpc extrae-gnu13-mpich-ohpc extrae-gnu13-mvapich2-ohpc extrae-gnu13-openmpi5-ohpc extrae-intel-impi-ohpc extrae-intel-mpich-ohpc extrae-intel-mvapich2-ohpc extrae-intel-openmpi4-ohpc extrae-intel-openmpi5-ohpc	3.8.3	Extrae tool. https://tools.bsc.es
geopm-gnu12-impi-ohpc geopm-gnu12-mpich-ohpc geopm-gnu12-mvapich2-ohpc geopm-gnu12-openmpi4-ohpc geopm-gnu13-impi-ohpc geopm-gnu13-mpich-ohpc geopm-gnu13-mvapich2-ohpc geopm-gnu13-openmpi5-ohpc geopm-intel-impi-ohpc geopm-intel-mpich-ohpc geopm-intel-mvapich2-ohpc geopm-intel-openmpi4-ohpc geopm-intel-openmpi5-ohpc	1.1.0	Global Extensible Open Power Manager. https://geopm.github.io
imb-gnu12-impi-ohpc imb-gnu12-mpich-ohpc imb-gnu12-mvapich2-ohpc imb-gnu12-openmpi4-ohpc imb-gnu13-impi-ohpc imb-gnu13-mpich-ohpc imb-gnu13-mvapich2-ohpc imb-gnu13-openmpi5-ohpc imb-intel-impi-ohpc imb-intel-mpich-ohpc imb-intel-mvapich2-ohpc imb-intel-openmpi4-ohpc imb-intel-openmpi5-ohpc	2021.3	Intel MPI Benchmarks (IMB). https://software.intel.com/en-us/articles/intel-mpi-benchmarks
likwid-gnu12-ohpc	5.2.2	Performance tools for the Linux console. https://github.com/RRZE-HPC/likwid
likwid-gnu13-ohpc likwid-intel-ohpc	5.3.0	
omb-gnu12-impi-ohpc omb-gnu12-mpich-ohpc omb-gnu12-mvapich2-ohpc omb-gnu12-openmpi4-ohpc	6.1	

Table 8 (cont): **Performance Analysis Tools**

RPM Package Name	Version	Info/URL
pdtoolkit-gnu13-ohpc pdtoolkit-intel-ohpc		
scalasca-gnu12-impi-ohpc scalasca-gnu12-mpich-ohpc scalasca-gnu12-mvapich2-ohpc scalasca-gnu12-openmpi4-ohpc scalasca-gnu13-impi-ohpc scalasca-gnu13-mpich-ohpc scalasca-gnu13-mvapich2-ohpc scalasca-gnu13-openmpi5-ohpc scalasca-intel-impi-ohpc scalasca-intel-mpich-ohpc scalasca-intel-mvapich2-ohpc scalasca-intel-openmpi4-ohpc scalasca-intel-openmpi5-ohpc	2.5	Toolset for performance analysis of large-scale parallel applications. http://www.scalasca.org
scorep-gnu12-impi-ohpc scorep-gnu12-mpich-ohpc scorep-gnu12-mvapich2-ohpc scorep-gnu12-openmpi4-ohpc scorep-gnu13-impi-ohpc scorep-gnu13-mpich-ohpc scorep-gnu13-mvapich2-ohpc scorep-gnu13-openmpi5-ohpc scorep-intel-impi-ohpc scorep-intel-mpich-ohpc scorep-intel-mvapich2-ohpc scorep-intel-openmpi4-ohpc scorep-intel-openmpi5-ohpc	7.1	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.vi-hps.org/projects/score-p
tau-gnu12-impi-ohpc tau-gnu12-mpich-ohpc tau-gnu12-mvapich2-ohpc tau-gnu12-openmpi4-ohpc tau-gnu13-impi-ohpc tau-gnu13-mpich-ohpc tau-gnu13-mvapich2-ohpc tau-gnu13-openmpi5-ohpc tau-intel-impi-ohpc tau-intel-mpich-ohpc tau-intel-mvapich2-ohpc tau-intel-openmpi4-ohpc tau-intel-openmpi5-ohpc	2.31.1	Tuning and Analysis Utilities Profiling Package. http://www.cs.uoregon.edu/research/tau/home.php

Table 9: **IO Libraries**

RPM Package Name	Version	Info/URL
adios2-gnu12-impi-ohpc adios2-gnu12-mpich-ohpc adios2-gnu12-mvapich2-ohpc adios2-gnu12-openmpi4-ohpc adios2-gnu13-impi-ohpc adios2-gnu13-mpich-ohpc adios2-gnu13-mvapich2-ohpc adios2-gnu13-openmpi5-ohpc adios2-intel-impi-ohpc adios2-intel-mpich-ohpc adios2-intel-mvapich2-ohpc adios2-intel-openmpi4-ohpc adios2-intel-openmpi5-ohpc	2.8.3	The Adaptable IO System v2 (ADIOS2). https://adios2.readthedocs.io/en/latest/index.html
hdf5-gnu12-ohpc hdf5-gnu13-ohpc hdf5-intel-ohpc	1.14.0	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx-gnu12-impi-ohpc netcdf-cxx-gnu12-mpich-ohpc netcdf-cxx-gnu12-mvapich2-ohpc netcdf-cxx-gnu12-openmpi4-ohpc netcdf-cxx-gnu13-impi-ohpc netcdf-cxx-gnu13-mpich-ohpc netcdf-cxx-gnu13-mvapich2-ohpc	4.3.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-cxx-gnu13-ohpc netcdf-cxx-gnu13-openmpi5-ohpc netcdf-cxx-intel-impi-ohpc netcdf-cxx-intel-mpich-ohpc netcdf-cxx-intel-mvapich2-ohpc netcdf-cxx-intel-ohpc netcdf-cxx-intel-openmpi4-ohpc netcdf-cxx-intel-openmpi5-ohpc	4.3.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran-gnu12-impi-ohpc netcdf-fortran-gnu12-mpich-ohpc netcdf-fortran-gnu12-mvapich2-ohpc netcdf-fortran-gnu12-openmpi4-ohpc netcdf-fortran-intel-openmpi4-ohpc	4.6.0	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran-gnu13-impi-ohpc netcdf-fortran-gnu13-mpich-ohpc netcdf-fortran-gnu13-mvapich2-ohpc	4.6.1	
netcdf-fortran-gnu13-ohpc netcdf-fortran-gnu13-openmpi5-ohpc netcdf-fortran-intel-impi-ohpc netcdf-fortran-intel-mpich-ohpc netcdf-fortran-intel-mvapich2-ohpc netcdf-fortran-intel-ohpc netcdf-fortran-intel-openmpi5-ohpc	4.6.1	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf

Table 9 (cont): **IO Libraries**

RPM Package Name	Version	Info/URL
pnetcdf-gnu13-impi-ohpc pnetcdf-gnu13-mpich-ohpc pnetcdf-gnu13-mvapich2-ohpc pnetcdf-gnu13-openmpi5-ohpc pnetcdf-intel-impi-ohpc pnetcdf-intel-mpich-ohpc pnetcdf-intel-mvapich2-ohpc pnetcdf-intel-openmpi4-ohpc pnetcdf-intel-openmpi5-ohpc		
sionlib-gnu12-impi-ohpc sionlib-gnu12-mpich-ohpc sionlib-gnu12-mvapich2-ohpc sionlib-gnu12-openmpi4-ohpc sionlib-gnu13-impi-ohpc sionlib-gnu13-mpich-ohpc sionlib-gnu13-mvapich2-ohpc sionlib-gnu13-openmpi5-ohpc sionlib-intel-impi-ohpc sionlib-intel-mpich-ohpc sionlib-intel-mvapich2-ohpc sionlib-intel-openmpi4-ohpc sionlib-intel-openmpi5-ohpc	1.7.7	Scalable I/O Library for Parallel Access to Task-Local Files. https://apps.fz-juelich.de/jsc/sionlib/docu/index.html

Table 10: **Runtimes**

RPM Package Name	Version	Info/URL
charliecloud-ohpc	0.15	Lightweight user-defined software stacks for high-performance computing. https://hpc.github.io/charliecloud

Table 11: **Serial/Threaded Libraries**

RPM Package Name	Version	Info/URL
R-gnu12-ohpc R-gnu13-ohpc	4.2.1	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
gsl-gnu12-ohpc gsl-gnu13-ohpc gsl-intel-ohpc	2.7.1	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis-gnu12-ohpc metis-gnu13-ohpc metis-intel-ohpc	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
openblas-gnu12-ohpc openblas-gnu13-ohpc	0.3.21	An optimized BLAS library based on GotoBLAS2. http://www.openblas.net
plasma-gnu12-ohpc plasma-gnu13-ohpc plasma-intel-ohpc	21.8.29	Parallel Linear Algebra Software for Multicore Architectures. https://icl.utk.edu/plasma/overview/index.html
scotch-gnu12-ohpc scotch-gnu13-ohpc scotch-intel-ohpc	6.0.6	Graph, mesh and hypergraph partitioning library. http://www.labri.fr/perso/pelegrin/scotch
superlu-gnu12-ohpc superlu-gnu13-ohpc superlu-intel-ohpc	5.2.1	A general purpose library for the direct solution of linear equations. http://crd.lbl.gov/~xiaoye/SuperLU

Table 12: **Parallel Libraries**

RPM Package Name	Version	Info/URL
boost-gnu12-impi-ohpc boost-gnu12-mpich-ohpc boost-gnu12-mvapich2-ohpc boost-gnu12-openmpi4-ohpc boost-gnu13-impi-ohpc boost-gnu13-mpich-ohpc boost-gnu13-mvapich2-ohpc boost-gnu13-openmpi5-ohpc boost-intel-impi-ohpc boost-intel-mpich-ohpc boost-intel-mvapich2-ohpc boost-intel-openmpi4-ohpc boost-intel-openmpi5-ohpc	1.81.0	Free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw-gnu12-impi-ohpc fftw-gnu12-mpich-ohpc fftw-gnu12-mvapich2-ohpc fftw-gnu12-openmpi4-ohpc fftw-gnu13-impi-ohpc fftw-gnu13-mpich-ohpc fftw-gnu13-mvapich2-ohpc fftw-gnu13-openmpi5-ohpc fftw-intel-impi-ohpc fftw-intel-mpich-ohpc fftw-intel-mvapich2-ohpc fftw-intel-openmpi4-ohpc fftw-intel-openmpi5-ohpc	3.3.10	A Fast Fourier Transform library. http://www.fftw.org
hypre-gnu12-impi-ohpc hypre-gnu12-mpich-ohpc hypre-gnu12-mvapich2-ohpc hypre-gnu12-openmpi4-ohpc hypre-gnu13-impi-ohpc hypre-gnu13-mpich-ohpc hypre-gnu13-mvapich2-ohpc hypre-gnu13-openmpi5-ohpc hypre-intel-impi-ohpc hypre-intel-mpich-ohpc hypre-intel-mvapich2-ohpc hypre-intel-openmpi4-ohpc hypre-intel-openmpi5-ohpc	2.18.1	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mfem-gnu12-impi-ohpc mfem-gnu12-mpich-ohpc mfem-gnu12-mvapich2-ohpc mfem-gnu12-openmpi4-ohpc mfem-gnu13-impi-ohpc mfem-gnu13-mpich-ohpc mfem-gnu13-mvapich2-ohpc mfem-gnu13-openmpi5-ohpc mfem-intel-impi-ohpc mfem-intel-mpich-ohpc mfem-intel-mvapich2-ohpc mfem-intel-openmpi4-ohpc mfem-intel-openmpi5-ohpc	4.4	Lightweight, general, scalable C++ library for finite element methods. http://mfem.org
mumps-gnu12-impi-ohpc mumps-gnu12-mpich-ohpc mumps-gnu12-mvapich2-ohpc mumps-gnu12-openmpi4-ohpc		

	5.2.1	A MUltifrontal Massively Parallel Sparse direct Solver. http://graal.ens-lyon.fr/MUMPS

Table 12 (cont): **Parallel Libraries**

RPM Package Name	Version	Info/URL
mumps-gnu13-impi-ohpc mumps-gnu13-mpich-ohpc mumps-gnu13-mvapich2-ohpc mumps-gnu13-openmpi5-ohpc mumps-intel-impi-ohpc mumps-intel-mpich-ohpc mumps-intel-mvapich2-ohpc mumps-intel-openmpi4-ohpc mumps-intel-openmpi5-ohpc		
opencoarrays-gnu12-impi-ohpc opencoarrays-gnu12-mpich-ohpc opencoarrays-gnu12-mvapich2-ohpc opencoarrays-gnu12-openmpi4-ohpc opencoarrays-gnu13-impi-ohpc opencoarrays-gnu13-mpich-ohpc opencoarrays-gnu13-mvapich2-ohpc opencoarrays-gnu13-openmpi5-ohpc	2.10.0	ABI to leverage the parallel programming features of the Fortran 2018 DIS. http://www.opencoarrays.org
petsc-gnu12-impi-ohpc petsc-gnu12-mpich-ohpc petsc-gnu12-mvapich2-ohpc petsc-gnu12-openmpi4-ohpc petsc-gnu13-impi-ohpc petsc-gnu13-mpich-ohpc petsc-gnu13-mvapich2-ohpc petsc-gnu13-openmpi5-ohpc petsc-intel-impi-ohpc petsc-intel-mpich-ohpc petsc-intel-mvapich2-ohpc petsc-intel-openmpi4-ohpc petsc-intel-openmpi5-ohpc	3.18.1	Portable Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc
ptscotch-gnu12-impi-ohpc ptscotch-gnu12-mpich-ohpc ptscotch-gnu12-mvapich2-ohpc ptscotch-gnu12-openmpi4-ohpc ptscotch-gnu13-impi-ohpc ptscotch-gnu13-mpich-ohpc ptscotch-gnu13-mvapich2-ohpc ptscotch-gnu13-openmpi5-ohpc ptscotch-intel-impi-ohpc ptscotch-intel-mpich-ohpc ptscotch-intel-mvapich2-ohpc ptscotch-intel-openmpi4-ohpc ptscotch-intel-openmpi5-ohpc	7.0.1	Graph, mesh and hypergraph partitioning library using MPI. http://www.labri.fr/perso/pelegrin/scotch
scalapack-gnu12-impi-ohpc scalapack-gnu12-mpich-ohpc scalapack-gnu12-mvapich2-ohpc scalapack-gnu12-openmpi4-ohpc scalapack-gnu13-impi-ohpc scalapack-gnu13-mpich-ohpc scalapack-gnu13-mvapich2-ohpc	2.2.0	A subset of LAPACK routines redesigned for heterogeneous computing. https://netlib.org/scalapack

Table 12 (cont): **Parallel Libraries**

RPM Package Name	Version	Info/URL
superlu_dist-gnu13-impi-ohpc superlu_dist-gnu13-mpich-ohpc superlu_dist-gnu13-mvapich2-ohpc superlu_dist-gnu13-openmpi5-ohpc superlu_dist-intel-impi-ohpc superlu_dist-intel-mpich-ohpc superlu_dist-intel-mvapich2-ohpc superlu_dist-intel-openmpi4-ohpc superlu_dist-intel-openmpi5-ohpc		
trilinos-gnu12-impi-ohpc trilinos-gnu12-mpich-ohpc trilinos-gnu12-mvapich2-ohpc trilinos-gnu12-openmpi4-ohpc trilinos-gnu13-impi-ohpc trilinos-gnu13-mpich-ohpc trilinos-gnu13-mvapich2-ohpc trilinos-gnu13-openmpi5-ohpc trilinos-intel-impi-ohpc trilinos-intel-mpich-ohpc trilinos-intel-mvapich2-ohpc trilinos-intel-openmpi5-ohpc	13.4.0	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:  5392 744D 3C54 3ED5 7847 65E6 8A30 6019 DA565C6C
```

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.

```
[sms]# rpm --checksig -v docs-ohpc-*.rpm
docs-ohpc-2.0.0-72.1.ohpc.2.0.x86_64.rpm:
    Header V3 RSA/SHA1 Signature, key ID da565c6c: OK
    Header SHA256 digest: OK
    Header SHA1 digest: OK
    Payload SHA256 digest: OK
    V3 RSA/SHA1 Signature, key ID da565c6c: OK
    MD5 digest: OK
```