OpenHPC (v2.0)
Cluster Building Recipes

OpenSUSE Leap 15.2 Base OS

Warewulf/SLURM Edition for Linux* (aarch64)

Document Last Update: 2020-10-06
Document Revision: 1e970dd16

Legal Notice

Contents

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 OpenSUSE Leap 15.2 and is subsequently configured to provision the remaining *compute* nodes with Warewulf 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.

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

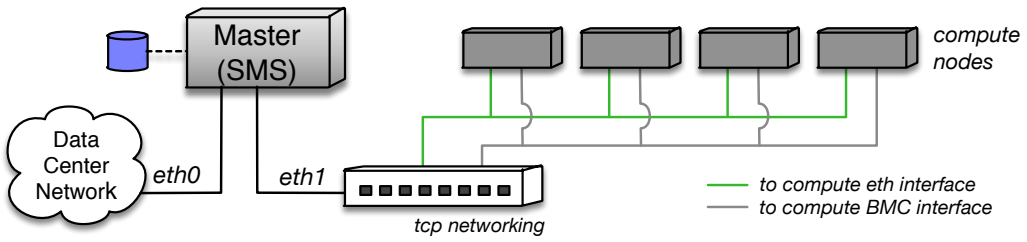

Figure 1: Overview of physical cluster architecture.

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.

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
- `${sms_eth_internal}` # Internal Ethernet interface on SMS
- `${eth_provision}` # Provisioning interface for computes
- `${internal_netmask}` # Subnet netmask for internal network
- `${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:

- `${kargs}` # Kernel boot arguments
- `${nagios_web_password}` # Nagios web access password

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 OpenSUSE Leap 15.2 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 OpenSUSE Leap 15.2 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 Warewulf, 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. If installed, the default firewall service can be disabled as follows:

```
[sms]# systemctl disable SuSEfirewall2
[sms]# systemctl stop SuSEfirewall2
```

3 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.

3.1 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.

```
[sms]# rpm -ivh http://repos.openhpc.community/OpenHPC/2/Leap_15/aarch64/ohpc-release-2-1.leap15.aarch64.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/2.0

In addition to the OpenHPC package repository, the *master* host also requires access to the standard distro repositories in order to resolve necessary dependencies.

3.2 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.

3.3 Add provisioning services on *master* node

With the OpenHPC repository enabled, we can now begin adding desired components onto the *master* server. This repository provides a number of aliases that group logical components together in order to help aid in this process. For reference, a complete list of available group aliases and RPM packages available via OpenHPC are provided in Appendix E. To add support for provisioning services, the following commands illustrate addition of a common base package followed by the Warewulf provisioning system.

```
# Install base meta-packages
[sms]# zypper -n install ohpc-base
[sms]# zypper -n install ohpc-warewulf
```

Tip

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

```
[sms]# ipmitool -E -I lanplus -H ${bmc_ipaddr} -U root chassis bootdev pxe options=persistent
```

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 "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
```

3.4 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]# zypper -n install ohpc-slurm-server

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

# Identify resource manager hostname on master host
[sms]# perl -pi -e "s/ControlMachine=\S+/ControlMachine=${sms_name}/" /etc/slurm/slurm.conf
```

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 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 particular hardware. 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.

3.5 Complete basic Warewulf setup for *master* node

At this point, all of the packages necessary to use Warewulf on the *master* host should be installed. Next, we need to update several configuration files in order to allow Warewulf to work with OpenSUSE Leap 15.2 and to support local provisioning using a second private interface (refer to Figure 1).

Tip

By default, Warewulf is configured to provision over the `eth1` interface and the steps below include updating this setting to override with a potentially alternatively-named interface specified by `${sms_eth_internal}`.

```
# Configure Warewulf provisioning to use desired internal interface
[sms]# perl -pi -e "s/device = eth1/device = ${sms_eth_internal}/" /etc/warewulf/provision.conf

# Configure DHCP server to use desired internal interface
[sms]# perl -pi -e "s/^DHCPD_INTERFACE=\S+/DHCPD_INTERFACE=${sms_eth_internal}/" /etc/sysconfig/dhcpd

# Enable tftp service for compute node image distribution
[sms]# perl -pi -e "s/^\s+disable\s+= yes/ disable = no/" /etc/xinetd.d/tftp

# Configure Warewulf to use the default SLES tftp location
[sms]# perl -pi -e "s#\#tftpdir = /var/lib/#tftpdir = /srv/#" /etc/warewulf/provision.conf

# Update Warewulf http configuration to use the SLES version of Apache modules
[sms]# export MODFILE=/etc/apache2/conf.d/warewulf-httpd.conf
[sms]# perl -pi -e "s#modules/mod_perl.so\$#/usr/lib64/apache2/mod_perl.so#" $MODFILE
[sms]# perl -pi -e "s#modules/mod_version.so\$#/usr/lib64/apache2/mod_version.so#" $MODFILE

# 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}

# Restart/enable relevant services to support provisioning
[sms]# systemctl enable mysql.service
[sms]# systemctl restart mysql
[sms]# systemctl enable apache2.service
[sms]# systemctl restart apache2
[sms]# systemctl enable dhcpd.service
[sms]# systemctl enable tftp.socket
[sms]# systemctl start tftp.socket
```

```
# Set chunk size for database
[sms]# perl -pi -e "s/^# database chunk.*$/database chunk size = 10000000/" /etc/warewulf/database.conf
```

3.6 Define *compute* image for provisioning

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

3.6.1 Build initial BOS image

The OpenHPC build of Warewulf includes specific enhancements enabling support for OpenSUSE Leap 15.2. The following steps illustrate the process to build a minimal, default image for use with Warewulf. We begin by creating a directory structure on the *master* host that will represent the root filesystem of the compute node. The default location for this example is in `/opt/ohpc/admin/images/leap15.2`.

```
# Define chroot location
[sms]# export CHROOT=/opt/ohpc/admin/images/leap15.2
# Build initial chroot image
[sms]# mkdir -p -m 755 $CHROOT                      # create chroot housing dir
[sms]# mkdir -m 755 $CHROOT/dev                     # create chroot /dev dir
[sms]# mknod -m 666 $CHROOT/dev/zero c 1 5          # create /dev/zero device
[sms]# wwmkchroot -v opensuse-15.2 $CHROOT          # create base image
# Enable OpenHPC repo in chroot
[sms]# cp -p /etc/zypp/repos.d/OpenHPC*.repo $CHROOT/etc/zypp/repos.d
# Import GPG keys for chroot repository usage
```

```
[sms]# zypper -n --root $CHROOT --no-gpg-checks --gpg-auto-import-keys refresh
```

3.6.2 Add OpenHPC components

The `wwmkchroot` process used in the previous step is designed to provide a minimal OpenSUSE Leap 15.2 configuration. Next, we add additional components to include resource management client services, NTP support, and other additional packages to support the default OpenHPC environment. This process augments the chroot-based install performed by `wwmkchroot` to modify the base provisioning image and will access the BOS and OpenHPC repositories to resolve package install requests. We begin by installing a few common base packages:

```
# Install compute node base meta-package
[sms]# zypper -n --root $CHROOT install ohpc-base-compute
```

To access the remote repositories by hostname (and not IP addresses), the chroot environment needs to be updated to enable DNS resolution. Assuming that the *master* host has a working DNS configuration in place, the chroot environment can be updated with a copy of the configuration as follows:

```
[sms]# cp -p /etc/resolv.conf $CHROOT/etc/resolv.conf
```

Now, we can include additional required components to the compute instance including resource manager client, NTP, and development environment modules support.

```
# Import GPG keys for chroot repository usage
[sms]# zypper -n --root $CHROOT --no-gpg-checks --gpg-auto-import-keys refresh

# copy credential files into $CHROOT to ensure consistent uid/gids for slurm/munge at
# install. Note that these will be synchronized with future updates via the provisioning system.
[sms]# cp /etc/passwd /etc/group $CHROOT/etc

# Add SLURM client support meta-package and enable munge
[sms]# zypper -n --root $CHROOT install ohpc-slurm-client
[sms]# chroot $CHROOT systemctl enable munge

# Register Slurm server with computes (using "configless" option)
[sms]# echo SLURMD_OPTIONS="--conf-server ${sms_ip}" > $CHROOT/etc/sysconfig/slurmd

# Provide udev rules to enable /dev/ipath access for InfiniPath devices
[sms]# cp /opt/ohpc/pub/examples/udev/60-ipath.rules $CHROOT/etc/udev/rules.d/

# Add Network Time Protocol (NTP) support
[sms]# zypper -n --root $CHROOT install chrony
[sms]# chroot $CHROOT systemctl enable chrony
# Identify master host as local NTP server
[sms]# echo "server ${sms_ip}" >> $CHROOT/etc/chrony.conf

# Add kernel drivers
[sms]# zypper -n --root $CHROOT install kernel-default

# Include modules user environment
[sms]# zypper -n --root $CHROOT install lmod-ohpc

# Enable ssh access
[sms]# chroot $CHROOT systemctl enable sshd.service

# Remove default hostname to allow WW to provision network names
[sms]# mv $CHROOT/etc/hostname $CHROOT/etc/hostname.orig
```

3.6.3 Customize system configuration

Prior to assembling the image, it is advantageous to perform any additional customization within the chroot environment created for the desired *compute* instance. The following steps document the process to add a local *ssh* key created by Warewulf to support remote access, identify the resource manager server, configure NTP for compute resources, and enable 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.

```
# Initialize warewulf database and ssh_keys
[sms]# wwinit database
[sms]# wwinit ssh_keys

# Add NFS client mounts of /home and /opt/ohpc/pub to base image
[sms]# echo "${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0" >> $CHROOT/etc/fstab
[sms]# echo "${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev 0 0" >> $CHROOT/etc/fstab

# 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
```

3.6.4 Additional Customization (*optional*)

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

- Restrict ssh access to compute resources
- Enable syslog forwarding
- Add Nagios Core monitoring
- 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.

3.6.4.1 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 within the *compute* image, issue the following:

```
[sms]# echo "account  required    pam_slurm.so" >> $CHROOT/etc/pam.d/sshd
```

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

```
# Configure SMS to receive messages and reload rsyslog configuration
[sms]# echo 'module(load="imudp")' >> /etc/rsyslog.d/ohpc.conf
[sms]# echo 'input(type="imudp" port="514")' >> /etc/rsyslog.d/ohpc.conf
```

```
[sms]# systemctl restart rsyslog

# Define compute node forwarding destination
[sms]# echo "*.* @${sms_ip}:514" >> $CHROOT/etc/rsyslog.conf
[sms]# echo "Target=\"${sms_ip}\" Protocol=\"udp\"" >> $CHROOT/etc/rsyslog.conf

# Disable most local logging on computes. Emergency and boot logs will remain on the compute nodes
[sms]# perl -pi -e "s/^\*\.info/\\#\*\.info/" $CHROOT/etc/rsyslog.conf
[sms]# perl -pi -e "s/^authpriv/\\#authpriv/" $CHROOT/etc/rsyslog.conf
[sms]# perl -pi -e "s/^mail/\\#mail/" $CHROOT/etc/rsyslog.conf
[sms]# perl -pi -e "s/^cron/\\#cron/" $CHROOT/etc/rsyslog.conf
[sms]# perl -pi -e "s/^uucp/\\#uucp/" $CHROOT/etc/rsyslog.conf
```

3.6.4.3 Add Nagios monitoring Nagios is an open source infrastructure network monitoring package designed to watch servers, switches, and various services and offers user-defined alerting facilities for monitoring various aspects of an HPC cluster. The core Nagios daemon and a variety of monitoring plugins are provided by the underlying OS distro and the following commands can be used to install and configure a Nagios server on the *master* node, and add the facility to run tests and gather metrics from provisioned *compute* nodes. This simple configuration example is intended to be illustrative to walk through defining a compute host group and enabling an ssh check for the computes. Users are encouraged to consult Nagios documentation for more information and can install additional plugins as desired on login nodes, service nodes, or compute hosts.

```
# Install nagios, nrep, and all available plugins on master host
[sms]# zypper -n install nagios nrpe monitoring-plugins-*

# Install nrpe and an example plugin into compute node image
[sms]# zypper -n --root $CHROOT install nrpe monitoring-plugins-ssh

# Enable and configure Nagios NRPE daemon in compute image
[sms]# chroot $CHROOT systemctl enable nrpe
[sms]# perl -pi -e "s/^allowed_hosts=/# allowed_hosts=/" $CHROOT/etc/nagios/nrpe.cfg
[sms]# echo "nrpe : ${sms_ip} : ALLOW" >> $CHROOT/etc/hosts.allow
[sms]# echo "nrpe : ALL : DENY"        >> $CHROOT/etc/hosts.allow

# Copy example Nagios config file to define a compute group and ssh check
# (note: edit as desired to add all desired compute hosts)
[sms]# cp /opt/ohpc/pub/examples/nagios/compute.cfg /etc/nagios/objects
# Register the config file with nagios
[sms]# echo "cfg_file=/etc/nagios/objects/compute.cfg" >> /etc/nagios/nagios.cfg

# Update location of mail binary for alert commands
[sms]# perl -pi -e "s/ \/bin\/mail/ \/usr\/bin\/mailx/g" /etc/nagios/objects/commands.cfg

# Update email address of contact for alerts
[sms]# perl -pi -e "s/nagios\@localhost/root\@${sms_name}/" /etc/nagios/objects/contacts.cfg

# Add check_ssh command for remote hosts
[sms]# echo command[check_ssh]=/usr/lib64/nagios/plugins/check_ssh localhost $CHROOT/etc/nagios/nrpe.cfg

# define password for nagiosadmin to be able to connect to web interface
[sms]# htpasswd -bc /etc/nagios/passwd nagiosadmin ${nagios_web_password}

# Enable Nagios on master, and configure
[sms]# systemctl enable nagios
[sms]# systemctl start nagios
# Update permissions on ping command to allow nagios user to execute
[sms]# chmod u+s `which ping`
```

3.6.4.4 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]# zypper -n 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
```

3.6.4.5 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]# zypper -n 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
```

3.6.4.6 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]# zypper -n 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
```

Note that an additional kernel boot option is typically necessary to enable serial console output. This option is highlighted in §3.7.4 after compute nodes have been registered with the provisioning system.

3.6.4.7 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]# zypper -n install nhc-ohpc
[sms]# zypper -n --root $CHROOT 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
```

3.6.5 Import files

The Warewulf system includes functionality to import arbitrary files from the provisioning server for distribution to managed hosts. This is one way to distribute user credentials to *compute* nodes. To import local file-based credentials, issue the following:

```
[sms]# wwsh file import /etc/passwd
[sms]# wwsh file import /etc/group
[sms]# wwsh file import /etc/shadow
```

Similarly, to import the cryptographic key that is required by the *munge* authentication library to be available on every host in the resource management pool, issue the following:

```
[sms]# wwsh file import /etc/munge/munge.key
```

3.7 Finalizing provisioning configuration

Warewulf employs a two-stage boot process for provisioning nodes via creation of a bootstrap image that is used to initialize the process, and a virtual node file system capsule containing the full system image. This section highlights creation of the necessary provisioning images, followed by the registration of desired compute nodes.

3.7.1 Assemble bootstrap image

The bootstrap image includes the runtime kernel and associated modules, as well as some simple scripts to complete the provisioning process. The following commands highlight the inclusion of additional drivers and creation of the bootstrap image based on the running kernel.

```
# (Optional) Include drivers from kernel updates; needed if enabling additional kernel modules on computes
[sms]# export WW_CONF=/etc/warewulf/bootstrap.conf
[sms]# echo "drivers += updates/kernel/" >> $WW_CONF

# Build bootstrap image
[sms]# wwbootstrap `uname -r`
```

3.7.2 Assemble Virtual Node File System (VNFS) image

With the local site customizations in place, the following step uses the `wwvnfs` command to assemble a VNFS capsule from the chroot environment defined for the *compute* instance.

```
[sms]# wwvnfs --chroot $CHROOT
```

3.7.3 Register nodes for provisioning

In preparation for provisioning, we can now define the desired network settings for four example compute nodes with the underlying provisioning system and restart the `dhcp` service. Note the use of variable names for the desired compute hostnames, node IPs, and MAC addresses which should be modified to accommodate local settings and hardware. By default, Warewulf uses network interface names of the `eth#` variety and adds kernel boot arguments to maintain this scheme on newer kernels. Consequently, when specifying the desired provisioning interface via the `$eth_provision` variable, it should follow this convention. Alternatively, if you prefer to use the predictable network interface naming scheme (e.g. names like `en4s0f0`), additional steps are included to alter the default kernel boot arguments and take the `eth#` named interface down after bootstrapping so the normal init process can bring it up again using the desired name.

The final step in this process associates the VNFS image assembled in previous steps with the newly defined compute nodes, utilizing the user credential files and munge key that were imported in §3.6.5.

```
# Set provisioning interface as the default networking device
[sms]# echo "GATEWAYDEV=${eth_provision}" > /tmp/network.$$
[sms]# wwsh -y file import /tmp/network.$$ --name network
[sms]# wwsh -y file set network --path /etc/sysconfig/network --mode=0644 --uid=0

# Add nodes to Warewulf data store
[sms]# for ((i=0; i<$num_computes; i++)) ; do
              wwsh -y node new ${c_name[i]} --ipaddr=${c_ip[i]} --hwaddr=${c_mac[i]} -D ${eth_provision}
       done
```

```
# Additional step required if desiring to use predictable network interface
# naming schemes (e.g. en4s0f0). Skip if using eth# style names.
[sms]# export kargs="${kargs} net.ifnames=1,biosdevname=1"
[sms]# wwsh provision set --postnetdown=1 "${compute_regex}"
```

```
# Define provisioning image for hosts
[sms]# wwsh -y provision set "${compute_regex}" --vnfs=leap15.2 --bootstrap=`uname -r` \
   --files=dynamic_hosts,passwd,group,shadow,munge.key,network
```

Tip

Warewulf includes a utility named **wwnodescan** to automatically register new compute nodes versus the outlined node-addition approach which requires hardware MAC addresses to be gathered in advance. With **wwnodescan**, nodes will be added to the Warewulf database in the order in which their DHCP requests are received by the master, so care must be taken to boot nodes in the order one wishes to see preserved in the Warewulf database. The IP address provided will be incremented after each node is found, and the utility will exit after all specified nodes have been found. Example usage is highlighted below:

```
[sms]# wwnodescan --netdev=${eth_provision} --ipaddr=${c_ip[0]} --netmask=${internal_netmask} \
    --vnfs=leap15.2 --bootstrap=`uname -r` --listen=${sms_eth_internal} ${c_name[0]}-${c_name[3]}
```

```
# Restart dhcp / update PXE
[sms]# systemctl restart dhcpd
[sms]# wwsh pxe update
```

3.7.4 Optional kernel arguments

If you chose to enable ConMan in §3.6.4.6, additional warewulf configuration is needed as follows:

```
# Define node kernel arguments to support SOL console
[sms]# wwsh -y provision set "${compute_regex}" --console=ttyS1,115200
```

If any components have added to the boot time kernel command line arguments for the compute nodes, the following command is required to store the configuration in Warewulf:

```
# Set optional compute node kernel command line arguments.
[sms]# wwsh -y provision set "${compute_regex}" --kargs="${kargs}"
```

3.7.5 Optionally configure stateful provisioning

Warewulf normally defaults to running the assembled VNFS image out of system memory in a *stateless* configuration. Alternatively, Warewulf can also be used to partition and format persistent storage such that the VNFS image can be installed locally to disk in a *stateful* manner. This does, however, require that a boot loader (GRUB) be added to the image as follows:

```
# Add GRUB2 bootloader and re-assemble VNFS image
[sms]# zypper -n --root $CHROOT install grub2
[sms]# wwvnfs --chroot $CHROOT
```

Enabling stateful nodes also requires additional site-specific, disk-related parameters in the Warewulf configuration, and several example partitioning scripts are provided in the distribution.

```
# Select (and customize) appropriate parted layout example
[sms]# cp /etc/warewulf/filesystem/examples/gpt_example.cmds /etc/warewulf/filesystem/gpt.cmds
[sms]# wwsh provision set --filesystem=gpt "${compute_regex}"
[sms]# wwsh provision set --bootloader=sda "${compute_regex}"
```

Tip

Those provisioning compute nodes in UEFI mode will install a slightly different set of packages in to the VNFS. Warewulf also provides an example EFI filesystem layout.

```
# Add GRUB2 bootloader and re-assemble VNFS image
[sms]# zypper -n --root $CHROOT install grub2-efi grub2-efi-modules
[sms]# wwvnfs --chroot $CHROOT
[sms]# cp /etc/warewulf/filesystem/examples/efi_example.cmds /etc/warewulf/filesystem/efi.cmds
[sms]# wwsh provision set --filesystem=efi "${compute_regex}"
[sms]# wwsh provision set --bootloader=sda "${compute_regex}"
```

Upon subsequent reboot of the modified nodes, Warewulf will partition and format the disk to host the desired VNFS image. Once the image is installed to disk, warewulf can be configured to use the nodes' local storage as the boot device.

```
# Configure local boot (after successful provisioning)
[sms]# wwsh provision set --bootlocal=normal "${compute_regex}"
```

3.8 Boot compute nodes

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

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

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

```
[sms]# pdsh -w c[1-4] uptime
c1  05:03am up   0:02,  0 users, load average: 0.20, 0.13, 0.05
c2  05:03am up   0:02,  0 users, load average: 0.20, 0.14, 0.06
c3  05:03am up   0:02,  0 users, load average: 0.19, 0.15, 0.06
c4  05:03am up   0:02,  0 users, load average: 0.15, 0.12, 0.05
```

Tip

While the `pxelinux.0` and `lpxelinux.0` files that ship with Warewulf to enable network boot support a wide range of hardware, some hosts may boot more reliably or faster using the BOS versions provided via the `syslinux` package. If you encounter PXE issues, consider replacing the `pxelinux.0` and `lpxelinux.0` files supplied with `warewulf-provision-ohpc` with versions from `syslinux`.

4 Install OpenHPC Development Components

The install procedure outlined in §3 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.

4.1 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]# zypper -n install ohpc-autotools

[sms]# zypper -n install EasyBuild-ohpc
[sms]# zypper -n install hwloc-ohpc
[sms]# zypper -n install spack-ohpc
[sms]# zypper -n install valgrind-ohpc
```

4.2 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]# zypper -n install gnu9-compilers-ohpc
```

4.3 MPI Stacks

For MPI development and runtime support, OpenHPC provides pre-packaged builds for two MPI families that are compatible with both ethernet and high-speed fabrics. These MPI stacks can be installed as follows:

```
[sms]# zypper -n install openmpi4-gnu9-ohpc mpich-ofi-gnu9-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 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]# zypper -n install mpich-ucx-gnu9-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/gnu9 ---------------------
   mpich/3.3.2-ofi  mpich/3.3.2-ucx (D)
```

4.4 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]# zypper -n install ohpc-gnu9-perf-tools
```

4.5 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]# zypper -n install lmod-defaults-gnu9-openmpi4-ohpc
```

Tip

If you want to change the default environment from the suggestion above, OpenHPC also provides additional default options using the GNU compiler toolchain with multiple MPICH variants or MVAPICH2. Relevant lmod-default packages names are as follows:

- lmod-defaults-gnu9-mpich-ofi-ohpc
- lmod-defaults-gnu9-mpich-ucx-ohpc

4.6 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 §4.3.

```
[sms]# zypper search -t package petsc-gnu7-*-ohpc
Loading repository data...
Reading installed packages...

S | Name                    | Summary
--+-------------------------+-------------------------------------------------------+--------
  | petsc-gnu7-mpich-ohpc    | Portable Extensible Toolkit for Scientific Computation | package
  | petsc-gnu7-openmpi3-ohpc | Portable Extensible Toolkit for Scientific Computation | package
```

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 §3.6.3, 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]# zypper -n install ohpc-gnu9-serial-libs
[sms]# zypper -n install ohpc-gnu9-io-libs
[sms]# zypper -n install ohpc-gnu9-python-libs
[sms]# zypper -n install ohpc-gnu9-runtimes
```

```
# Install parallel lib meta-packages for all available MPI toolchains
[sms]# zypper -n install ohpc-gnu9-mpich-parallel-libs
[sms]# zypper -n install ohpc-gnu9-openmpi4-parallel-libs
```

4.7 Optional Development Tool Builds

In addition to the 3rd party development libraries built using the open source toolchains mentioned in §4.6, OpenHPC also provides a subset of *optional* builds compatible with the Arm Compiler for Linux. These packages provide a similar hierarchical user environment experience as other compiler families present in OpenHPC. To take advantage of the available builds, the OpenHPC variant of the Arm Compiler for Linux (and any required licenses) must be downloaded and installed separately. See the following for more information on obtaining this toolchain:

https://developer.arm.com/tools-and-software/server-and-hpc/downloads/arm-allinea-studio/openhpc

Tip

As noted in §3.6.3, the default installation path for OpenHPC (`/opt/ohpc/pub`) is exported over NFS from the *master* to the compute nodes, but the Arm Linux compiler installer defaults to a path of `/opt/arm`. To make the Arm Linux compiler available to the compute nodes one must either customize the installation path to be within `/opt/ohpc/pub` (e.g. using the `-i /opt/ohpc/pub/arm` option with the Arm installer), or alternatively, add an additional NFS export for `/opt/arm` that is mounted on desired compute nodes.

Once installed locally, the OpenHPC compatible packages can be installed using standard package manager semantics after installation of a compatibility package. To enable all 3rd party builds available in OpenHPC that are compatible with Arm Linux Compiler, issue the following:

```
# Install OpenHPC compatibility packages (requires prior installation of Arm Linux Compiler)
[sms]# zypper -n install arm1-compilers-devel-ohpc
```

```
# Install 3rd party libraries/tools meta-packages built with Arm vendor toolchain
[sms]# zypper -n install ohpc-arm1-serial-libs
[sms]# zypper -n install ohpc-arm1-io-libs
[sms]# zypper -n install ohpc-arm1-perf-tools

[sms]# zypper -n install ohpc-arm1-mpich-parallel-libs
[sms]# zypper -n install ohpc-arm1-openmpi4-parallel-libs
```

5 Resource Manager Startup

In section §3, the Slurm resource manager was installed and configured for use on both the *master* host and *compute* node instances. With the cluster nodes up and functional, we can now startup the resource manager services in preparation for running user jobs. 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]# pdsh -w $compute_prefix[1-4] systemctl start munge
[sms]# pdsh -w $compute_prefix[1-4] systemctl start slurmd
```

6 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
```

Warewulf installs a utility on the compute nodes to automatically synchronize known files from the provisioning server at five minute intervals. In this recipe, recall that we previously registered credential files with Warewulf (e.g. passwd, group, and shadow) so that these files would be propagated during compute node imaging. However, with the addition of a new "test" user above, the files have been outdated and we need to update the Warewulf database to incorporate the additions. This re-sync process can be accomplished as follows:

```
[sms]# wwsh file resync passwd shadow group
```

Tip

After re-syncing to notify Warewulf of file modifications made on the *master* host, it should take approximately 5 minutes for the changes to propagate. However, you can also manually pull the changes from compute nodes via the following:

```
[sms]# pdsh -w $compute_prefix[1-4] /warewulf/bin/wwgetfiles
```

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.

6.1 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 ~]$ srun -n 8 -N 2 --pty /bin/bash

[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	srun –pty /bin/bash
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]

6.2 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]# zypper -n 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/leap15/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/leap15/aarch64/warewulf/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 v2, you will first need to update your local OpenHPC repository configuration to point against the v2 release (or update your locally hosted mirror). Refer to §3.1 for more details on enabling the latest repository. In contrast, when upgrading between micro releases on the same branch (e.g. from v2 to 2.2), there is no need to adjust local package manager configurations when using the public repository as rolling updates are pre-configured.

1. (Optional) Ensure repo metadata is current (on head node and in chroot location(s)). 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]# zypper clean -a
[sms]# zypper --root $CHROOT clean
```

2. Upgrade master (SMS) node

```
[sms]# zypper -n up "*-ohpc"

# Any new Base OS provided dependencies can be installed by
# updating the ohpc-base metapackage
[sms]# zypper -n up "ohpc-base"
```

> **Tip**
>
> The version of Warewulf included in OpenHPC v1.3.6 added a new architecture-specific package containing iPXE files. Upgraders will need to install this package on the SMS node.
>
> ```
> [sms]# zypper -n install warewulf-provision-server-ipxe-aarch64-ohpc
> ```

3. Upgrade packages in compute image

```
[sms]# zypper -n --root $CHROOT up "*-ohpc"

# Any new compute-node Base OS provided dependencies can be installed by
# updating the ohpc-base-compute metapackage
[sms]# zypper -n --root $CHROOT up "ohpc-base-compute"
```

4. Rebuild image(s)

```
[sms]# wwvnfs --chroot $CHROOT
```

In the case where packages were upgraded within the chroot compute image, you will need to reboot the compute nodes when convenient to enable the changes.

B.1 New component variants

As newer variants of key compiler/MPI stacks are released, OpenHPC will periodically add toolchains enabling the latest variant. To stay consistent throughout the build hierarchy, minimize recompilation requirements for existing binaries, and allow for multiple variants to coexist, unique delimiters are used to distinguish RPM package names and module hierarchy.

In the case of a fresh install, OpenHPC recipes default to installation of the latest toolchains available in a given release branch. However, if upgrading a previously installed system, administrators can *opt-in* to enable new variants as they become available. To illustrate this point, consider the previous OpenHPC 1.3.5 release as an example which contained GCC 7.3.0 along with runtimes and libraries compiled with this toolchain. In the case where an admin would like to enable the newer "gnu8" toolchain, installation of these additions is simplified with the use of OpenHPC's meta-packages (see Table 1 in Appendix E). The following example illustrates adding the complete "gnu8" toolchain. Note that we leverage the convenience meta-packages containing MPI-dependent builds, and we also update the modules environment to make it the default.

```
# Install GCC 8.x-compiled meta-packages with dependencies
[sms]# zypper -n install ohpc-gnu8-perf-tools \
                         ohpc-gnu8-io-libs \
                         ohpc-gnu8-python-libs \
                         ohpc-gnu8-runtimes \
                         ohpc-gnu8-serial \
                         ohpc-gnu8-parallel-libs

# Update default environment
[sms]# zypper -n remove lmod-defaults-gnu7-openmpi3-ohpc
[sms]# zypper -n install lmod-defaults-gnu8-openmpi3-ohpc
```

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]# zypper -n 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. 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........... : aarch64
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 `zypper` 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.

Tip

Packages modified locally will not be cryptographically signed like the packages in the OpenHPC repository. Zypper requires and additional flag (`--no-gpg-checks`) to install unsigned packages.

```
# Install rpm-build package from base OS distro
sms:~ # zypper -n install rpm-build

# Download SRPM from OpenHPC repository and install locally
sms001:~ # zypper source-install fftw-gnu9-openmpi4-ohpc

# Modify spec file as desired
sms001:~ # cd ~/rpmbuild/SPECS
sms:~rpmbuild/SPECS # perl -pi -e "s/enable-static=no/enable-static=yes/" fftw.spec

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

# Install the new package
sms ~rpmbuild/SPECS # zypper --no-gpg-checks in \
       ../RPMS/aarch64/fftw-gnu9-openmpi4-static-ohpc.2.0-3.3.8-6.1.aarch64.rpm

# The new module file appears along side the default
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 1.

Table 1: **Available OpenHPC Meta-packages**

Group Name	Description
ohpc-arm1-io-libs	Collection of IO library builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-mpich-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the MPICH runtime.
ohpc-arm1-openmpi4-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the openmpi4 runtime.
ohpc-arm1-perf-tools	Collection of performance tool builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-serial-libs	Collection of serial library builds for use with the Arm Compiler for Linux toolchain.
ohpc-autotools	Collection of GNU autotools packages.
ohpc-base	Collection of base packages.
ohpc-base-compute	Collection of compute node base packages.
ohpc-gnu9-geopm	Global Extensible Open Power Manager for use with GNU compiler toolchain.
ohpc-gnu9-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu9-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu9-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu9-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu9-openmpi4-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu9-openmpi4-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu9-openmpi4-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu9-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu9-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu9-python-libs	Collection of python related library builds for use with GNU compiler toolchain.
ohpc-gnu9-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu9-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu9-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-slurm-client	Collection of client packages for SLURM.
ohpc-slurm-server	Collection of server packages for SLURM.
ohpc-warewulf	Collection of base packages for Warewulf provisioning.

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

- Administrative tools (Table 2)
- Provisioning (Table 3)
- Resource management (Table 4)
- Compiler families (Table 5)
- MPI families (Table 6)
- Development tools (Table 7)
- Performance analysis tools (Table 8)
- IO Libraries (Table 9)
- Runtimes (Table 10)
- Serial/Threaded Libraries (Table 11)
- Parallel Libraries (Table 12)

Table 2: **Administrative Tools**

RPM Package Name	Version	Info/URL
conman-ohpc	0.3.0	ConMan: The Console Manager. http://dun.github.io/conman
docs-ohpc	2.0.0	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
lmod-defaults-arm1-mpich-ohpc lmod-defaults-arm1-openmpi4-ohpc lmod-defaults-gnu9-mpich-ofi-ohpc lmod-defaults-gnu9-mpich-ucx-ohpc lmod-defaults-gnu9-openmpi4-ohpc	2.0	OpenHPC default login environments. https://github.com/openhpc/ohpc
lmod-ohpc	8.2.10	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	1.4.2	LBNL Node Health Check. https://github.com/mej/nhc
ohpc-release	2	OpenHPC release files. https://github.com/openhpc/ohpc
pdsh-ohpc	2.34	Parallel remote shell program. https://github.com/chaos/pdsh
prun-ohpc	2.0	Convenience utility for parallel job launch. https://github.com/openhpc/ohpc
test-suite-ohpc	2.0.0	Integration test suite for OpenHPC. https://github.com/openhpc/ohpc/tests

Table 3: **Provisioning**

RPM Package Name	Version	Info/URL
warewulf-common-ohpc	3.9.0	Scalable systems management suite for high performance clusters. http://warewulf.lbl.gov
warewulf-ipmi-ohpc-initramfs-aarch64	3.9.0	Warewulf - Add IPMI to aarch64 initramfs. http://warewulf.lbl.gov
warewulf-ipmi-ohpc-initramfs-x86_64	3.9.0	Warewulf - Add IPMI to x86_64 initramfs. http://warewulf.lbl.gov
warewulf-provision-ohpc-gpl_sources	3.9.0	Warewulf - GPL sources used in Warewulf provisioning. http://warewulf.lbl.gov
warewulf-cluster-ohpc	3.9.0	Warewulf - HPC cluster configuration. http://warewulf.lbl.gov
warewulf-ipmi-ohpc	3.9.0	Warewulf - IPMI support. http://warewulf.lbl.gov
warewulf-common-ohpc-localdb	3.9.0	Warewulf - Install local database server. http://warewulf.lbl.gov
warewulf-provision-ohpc	3.9.0	Warewulf - System provisioning core. http://warewulf.lbl.gov
warewulf-provision-ohpc-server	3.9.0	Warewulf - System provisioning server. http://warewulf.lbl.gov
warewulf-vnfs-ohpc	3.9.0	Warewulf - Virtual Node File System support. http://warewulf.lbl.gov
warewulf-provision-ohpc-server-ipxe-aarch64	3.9.0	Warewulf - iPXE Bootloader for aarch64. http://warewulf.lbl.gov
warewulf-provision-ohpc-server-ipxe-x86_64	3.9.0	Warewulf - iPXE Bootloader for x86_64. http://warewulf.lbl.gov
warewulf-provision-ohpc-initramfs-aarch64	3.9.0	Warewulf - initramfs base for aarch64. http://warewulf.lbl.gov
warewulf-provision-ohpc-initramfs-x86_64	3.9.0	Warewulf - initramfs base for x86_64. http://warewulf.lbl.gov

Table 4: **Resource Management**

RPM Package Name	Version	Info/URL
openpbs-execution-ohpc	20.0.0	OpenPBS for an execution host. http://www.openpbs.org
openpbs-client-ohpc	20.0.0	OpenPBS for a client host. http://www.openpbs.org
openpbs-server-ohpc	20.0.0	OpenPBS for a server host. http://www.openpbs.org
pmix-ohpc	3.1.4	An extended/exascale implementation of PMI. https://pmix.github.io/pmix
slurm-devel-ohpc	20.02.5	Development package for Slurm. https://slurm.schedmd.com
slurm-example-configs-ohpc	20.02.5	Example config files for Slurm. https://slurm.schedmd.com
slurm-pam_slurm-ohpc	20.02.5	PAM module for restricting access to compute nodes via Slurm. https://slurm.schedmd.com
slurm-perlapi-ohpc	20.02.5	Perl API to Slurm. https://slurm.schedmd.com
slurm-contribs-ohpc	20.02.5	Perl tool to print Slurm job state information. https://slurm.schedmd.com
slurm-ohpc	20.02.5	Slurm Workload Manager. https://slurm.schedmd.com
slurm-slurmd-ohpc	20.02.5	Slurm compute node daemon. https://slurm.schedmd.com
slurm-slurmctld-ohpc	20.02.5	Slurm controller daemon. https://slurm.schedmd.com
slurm-slurmdbd-ohpc	20.02.5	Slurm database daemon. https://slurm.schedmd.com
slurm-libpmi-ohpc	20.02.5	Slurmś implementation of the pmi libraries. https://slurm.schedmd.com
slurm-torque-ohpc	20.02.5	Torque/PBS wrappers for transition from Torque/PBS to Slurm. https://slurm.schedmd.com
slurm-openlava-ohpc	20.02.5	openlava/LSF wrappers for transition from OpenLava/LSF to Slurm. https://slurm.schedmd.com

Table 5: **Compiler Families**

RPM Package Name	Version	Info/URL
arm1-compilers-devel-ohpc	2.0	OpenHPC compatibility package for Arm HPC compiler. https://github.com/openhpc/ohpc
gnu9-compilers-ohpc	9.3.0	The GNU C Compiler and Support Files. http://gcc.gnu.org

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

RPM Package Name	Version	Info/URL
libfabric-ohpc	1.10.1	User-space RDMA Fabric Interfaces. http://www.github.com/ofiwg/libfabric
mpich-ofi-arm1-ohpc mpich-ofi-gnu9-ohpc	3.3.2	MPICH MPI implementation. http://www.mpich.org
mpich-ucx-arm1-ohpc mpich-ucx-gnu9-ohpc	3.3.2	MPICH MPI implementation. http://www.mpich.org
openmpi4-arm1-ohpc openmpi4-gnu9-ohpc	4.0.4	A powerful implementation of MPI/SHMEM. http://www.open-mpi.org
ucx-ohpc	1.8.0	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	4.3.0	Software build and installation framework. https://easybuilders.github.io/easybuild
automake-ohpc	1.16.1	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake
autoconf-ohpc	2.69	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
cmake-ohpc	3.16.2	CMake is an open-source, cross-platform family of tools designed to build, test and package software. https://cmake.org
hwloc-ohpc	2.1.0	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-gnu9-mpich-ohpc python3-scipy-gnu9-openmpi4-ohpc	1.5.1	Scientific Tools for Python. http://www.scipy.org
python3-numpy-gnu9-ohpc	1.19.0	NumPy array processing for numbers, strings, records and objects. https://github.com/numpy/numpy
python3-mpi4py-gnu9-mpich-ohpc python3-mpi4py-gnu9-openmpi4-ohpc	3.0.3	Python bindings for the Message Passing Interface (MPI) standard. https://bitbucket.org/mpi4py/mpi4py
spack-ohpc	0.15.0	HPC software package management. https://github.com/LLNL/spack
valgrind-ohpc	3.15.0	Valgrind Memory Debugger. http://www.valgrind.org

Table 8: **Performance Analysis Tools**

RPM Package Name	Version	Info/URL
dimemas-arm1-mpich-ohpc dimemas-arm1-openmpi4-ohpc dimemas-gnu9-mpich-ohpc dimemas-gnu9-openmpi4-ohpc	5.4.2	Dimemas tool. https://tools.bsc.es
extrae-gnu9-mpich-ohpc extrae-gnu9-openmpi4-ohpc	3.7.0	Extrae tool. https://tools.bsc.es
imb-gnu9-mpich-ohpc imb-gnu9-openmpi4-ohpc	2019.6	Intel MPI Benchmarks (IMB). https://software.intel.com/en-us/articles/intel-mpi-benchmarks
omb-arm1-mpich-ohpc omb-arm1-openmpi4-ohpc omb-gnu9-mpich-ohpc omb-gnu9-openmpi4-ohpc	5.6.2	OSU Micro-benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks
paraver-ohpc	4.8.2	Paraver. https://tools.bsc.es
papi-ohpc	5.7.0	Performance Application Programming Interface. http://icl.cs.utk.edu/papi
pdtoolkit-arm1-ohpc pdtoolkit-gnu9-ohpc	3.25.1	PDT is a framework for analyzing source code. http://www.cs.uoregon.edu/Research/pdt
scalasca-arm1-mpich-ohpc scalasca-gnu9-mpich-ohpc scalasca-gnu9-openmpi4-ohpc	2.5	Toolset for performance analysis of large-scale parallel applications. http://www.scalasca.org
scorep-arm1-mpich-ohpc scorep-gnu9-mpich-ohpc scorep-gnu9-openmpi4-ohpc	6.0	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.vi-hps.org/projects/score-p
tau-gnu9-mpich-ohpc tau-gnu9-openmpi4-ohpc	2.29	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
adios-gnu9-mpich-ohpc adios-gnu9-openmpi4-ohpc	1.13.1	The Adaptable IO System (ADIOS). http://www.olcf.ornl.gov/center-projects/adios
hdf5-arm1-ohpc hdf5-gnu9-ohpc	1.10.6	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx-gnu9-mpich-ohpc netcdf-cxx-gnu9-openmpi4-ohpc	4.3.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran-gnu9-mpich-ohpc netcdf-fortran-gnu9-openmpi4-ohpc	4.5.2	Fortran Libraries for NetCDF. http://www.unidata.ucar.edu/software/netcdf
netcdf-gnu9-mpich-ohpc netcdf-gnu9-openmpi4-ohpc	4.7.3	C Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
phdf5-gnu9-mpich-ohpc phdf5-gnu9-openmpi4-ohpc	1.10.6	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
pnetcdf-arm1-mpich-ohpc pnetcdf-arm1-openmpi4-ohpc pnetcdf-gnu9-mpich-ohpc pnetcdf-gnu9-openmpi4-ohpc	1.12.1	A Parallel NetCDF library (PnetCDF). http://cucis.ece.northwestern.edu/projects/PnetCDF
sionlib-arm1-mpich-ohpc sionlib-arm1-openmpi4-ohpc sionlib-gnu9-mpich-ohpc sionlib-gnu9-openmpi4-ohpc	1.7.4	Scalable I/O Library for Parallel Access to Task-Local Files. http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.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
singularity-ohpc	3.4.2	Application and environment virtualization. https://www.sylabs.io/singularity

Table 11: **Serial/Threaded Libraries**

RPM Package Name	Version	Info/URL
R-gnu9-ohpc	3.6.3	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
gsl-arm1-ohpc gsl-gnu9-ohpc	2.6	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis-arm1-ohpc metis-gnu9-ohpc	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
openblas-gnu9-ohpc	0.3.7	An optimized BLAS library based on GotoBLAS2. http://www.openblas.net
plasma-arm1-ohpc plasma-gnu9-ohpc	2.8.0	Parallel Linear Algebra Software for Multicore Architectures. https://bitbucket.org/icl/plasma
scotch-arm1-ohpc scotch-gnu9-ohpc	6.0.6	Graph, mesh and hypergraph partitioning library. http://www.labri.fr/perso/pelegrin/scotch
superlu-arm1-ohpc superlu-gnu9-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-arm1-mpich-ohpc boost-arm1-openmpi4-ohpc boost-gnu9-mpich-ohpc boost-gnu9-openmpi4-ohpc	1.73.0	Boost free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw-arm1-mpich-ohpc fftw-arm1-openmpi4-ohpc fftw-gnu9-mpich-ohpc fftw-gnu9-openmpi4-ohpc	3.3.8	A Fast Fourier Transform library. http://www.fftw.org
hypre-arm1-mpich-ohpc hypre-arm1-openmpi4-ohpc hypre-gnu9-mpich-ohpc hypre-gnu9-openmpi4-ohpc	2.18.1	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mfem-gnu9-mpich-ohpc mfem-gnu9-openmpi4-ohpc	4.1	Lightweight, general, scalable C++ library for finite element methods. http://mfem.org
mumps-arm1-mpich-ohpc mumps-arm1-openmpi4-ohpc mumps-gnu9-mpich-ohpc mumps-gnu9-openmpi4-ohpc	5.2.1	A MUltifrontal Massively Parallel Sparse direct Solver. http://mumps.enseeiht.fr
opencoarrays-gnu9-mpich-ohpc opencoarrays-gnu9-openmpi4-ohpc	2.8.0	ABI to leverage the parallel programming features of the Fortran 2018 DIS. http://www.opencoarrays.org
petsc-gnu9-mpich-ohpc petsc-gnu9-openmpi4-ohpc	3.13.1	Portable Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc
ptscotch-arm1-mpich-ohpc ptscotch-arm1-openmpi4-ohpc ptscotch-gnu9-mpich-ohpc ptscotch-gnu9-openmpi4-ohpc	6.0.6	Graph, mesh and hypergraph partitioning library using MPI. http://www.labri.fr/perso/pelegrin/scotch
scalapack-arm1-mpich-ohpc scalapack-arm1-openmpi4-ohpc scalapack-gnu9-mpich-ohpc scalapack-gnu9-openmpi4-ohpc	2.1.0	A subset of LAPACK routines redesigned for heterogenous computing. http://www.netlib.org/lapack-dev
slepc-gnu9-mpich-ohpc slepc-gnu9-openmpi4-ohpc	3.13.2	A library for solving large scale sparse eigenvalue problems. http://slepc.upv.es
superlu_dist-arm1-mpich-ohpc superlu_dist-arm1-openmpi4-ohpc superlu_dist-gnu9-mpich-ohpc superlu_dist-gnu9-openmpi4-ohpc	6.1.1	A general purpose library for the direct solution of linear equations. http://crd-legacy.lbl.gov/~xiaoye/SuperLU
trilinos-gnu9-mpich-ohpc trilinos-gnu9-openmpi4-ohpc	13.0.0	A collection of libraries of numerical algorithms. https://trilinos.org

F Package Signatures

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

```
Fingerprint:  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
```