OpenHPC (v1.2.1)
Cluster Building Recipes

SLES12SP1 Base OS

Warewulf/SLURM Edition for Linux* (aarch64)

[Tech Preview]

Document Last Update: 2017-01-24

Document Revision: ac5491c

Legal Notice

Contents

1 Tech Preview

This guide highlights the initial availability of OpenHPC packages targeted for use on 64-bit ARM-based architectures. This collection is being provided as a **Tech Preview** release initially, as there are some known issues around provisioning and a subset of development packages. A running log of errata and fixes can be found at the ARM Tech Preview Wiki.

The guide follows the general installation steps laid out in other companion OpenHPC recipes. However, the provisioning steps as outlined with Warewulf (most of the steps in § 4.3 thru 4.9) are not directly usable without additional modification to the PXE boot process. Consequently, users interested in leveraging packages from this Tech Preview are encouraged to enable the repo (see § 4.1) and install desired development components (§ 5) on top of systems were the underlying base OS is pre-installed. In addition, if multiple nodes are available, the SLURM resource manager can be used to schedule resources. Future OpenHPC releases will expand on this tech preview to include validated recipes for a bare-metal cluster install.

Known Package Issues:

- GSL: a small subset of tests performed with the GSL library failed precision related tests. This is currently attributed to the fact that the tests included in GSL are tuned for x86 which does 80-bit extended precision.
- PAPI: hardware counter availability may not be available depending on the underlying ARM platform.
- MPI: The hardware used for validating this Tech Preview release contained only ethernet. The available MPI stacks reflect this test environment.
- Hypre and SuperLU-dist: the libraries build, but when linking test applications unresolved symbols remain
- Nagios and Ganglia: don't work on SLES-12-SP1 due to missing PHP5 dependencies
- Lustre: since various ARM platforms require different kernels than the standard ones provided by the SLES-12-SP1 and CentOS-7.2 distributions, building a lustre client that would work for these specific platform configurations was beyond the scope of this release.
- Warewulf: the ARM Standard Base Boot Requirements and Standard Base System Architecture requires specific UEFI support during the boot process which doesn't seem to be compatible with the way warewulf currently auto-provisions worker nodes. There is a work-around, but it requires some manual intervention during installation and deployment of the nodes.

2 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 using a mix of open-source components. 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.

2.1 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

The solution is to increase the size of the manuals. –Mark V. Shaney

2.2 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 SLES12SP1 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. Installation information is also discussed to optionally include a Lustre file system mount and in this case, the Lustre file system is assumed to exist previously.

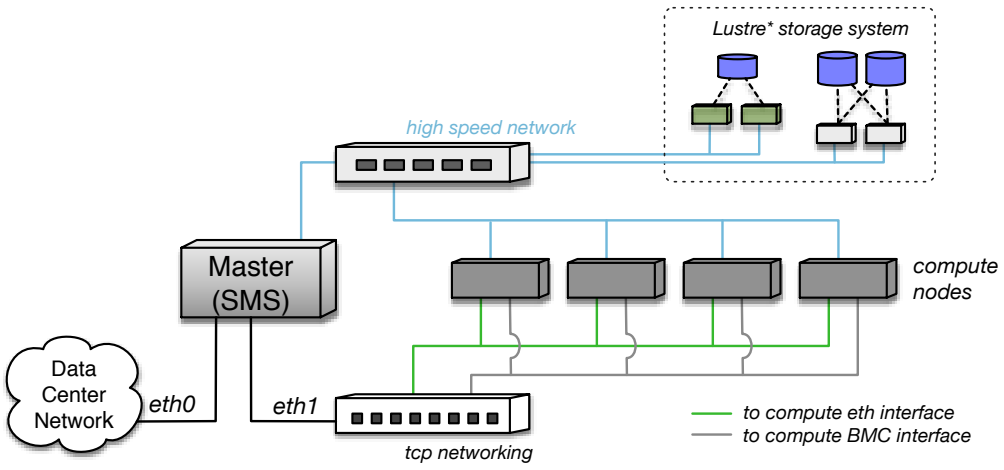

Figure 1: Overview of physical cluster architecture.

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 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.

In addition to the IP networking, there is a high-speed network (InfiniBand 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 Lustre connectivity as well.

2.3 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]}`, `${c_mac[1]}`, ... # MAC addresses for computes
- `${compute_regex}` # Regex matching all compute node names (e.g. "c*")
- `${compute_prefix}` # Prefix for compute node names (e.g. "c")

Optional:

- `${mgs_fs_name}` # Lustre MGS mount name
- `${sms_ipoib}` # IPoIB address for SMS server
- `${ipoib_netmask}` # Subnet netmask for internal IPoIB
- `${c_ipoib[0]}`, `${c_ipoib[1]}`, ... # IPoIB addresses for computes
- `${kargs}` # Kernel boot arguments

3 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 SLES12SP1 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 SLES12SP1 distribution.

Prior to beginning the installation process of OpenHPC components, several additional considerations are noted here for the SMS host configuration. First, the remaining installation recipe instructions herein assume that your 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
```

Next, while it is theoretically possible to provision a Warewulf cluster with SELinux enabled, 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
```

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 repository enablement. The example which follows illustrates installation of the `ohpc-release` package directly from the OpenHPC build server.

```
[sms]# rpm -ivh http://build.openhpc.community/OpenHPC:/1.2/SLE_12_SP1/aarch64/ohpc-release-1.2-1.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://build.openhpc.community/dist/1.2.1

In addition to the OpenHPC package repository, the *master* host also requires access to the standard distro repositories in order to resolve necessary dependencies. For SLES12SP1, the requirements are to have access to both the base and SDK repositories. In addition, add-on modules are needed to resolve dependencies for certain monitoring components. A summary of these requirements is as follows:

- SLES12-SP1-12.1-0
- SLES12-SP1-SDK-12.1-0
- Web and Scripting Module (*only required for nagios/ganglia)
- Server Monitoring Software (*only required for nagios/ganglia)

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

4.3 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 C. To add support for provisioning services, the following commands illustrate addition of a common base package followed by the Warewulf provisioning system.

```
[sms]# zypper -n install -t pattern ohpc-base
[sms]# zypper -n install -t pattern 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 typically 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}, issue the following:

```
[sms]# systemctl enable ntpd.service
[sms]# echo "server ${ntp_server}" >> /etc/ntp.conf
[sms]# systemctl restart ntpd
```

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.

```
[sms]# zypper -n install -t pattern ohpc-slurm-server
```

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

4.5 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]# zypper -n install libibverbs-runtime libmlx4-rdmav2 libipathverbs-rdmav2
[sms]# zypper -n install libibmad5 librdmacm1 rdma infinipath-psm dapl-devel dapl-utils

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

# Load IB drivers
[sms]# systemctl start rdma
```

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 SLES12SP1 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 §4.7.4.3 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/sles/ifcfg-ib0 /etc/sysconfig/network
```

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

# Initiate ib0
[sms]# ifup ib0
```

4.6 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 SLES12SP1 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 mod_perl
[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

# Enable http access for Warewulf cgi-bin directory to support newer apache syntax
[sms]# perl -pi -e "s/cgi-bin>\$/cgi-bin>\n Require all granted/" $MODFILE
[sms]# perl -pi -e "s/Allow from all/Require all granted/" $MODFILE
[sms]# perl -ni -e "print unless /^\s+Order allow,deny/" $MODFILE

# Enable internal interface for provisioning
[sms]# ifconfig ${sms_eth_internal} ${sms_ip} netmask ${internal_netmask} up

# Restart/enable relevant services to support provisioning
[sms]# systemctl restart xinetd
[sms]# systemctl enable mysql.service
[sms]# systemctl restart mysql
[sms]# systemctl enable apache2.service
[sms]# systemctl restart apache2
```

4.7 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.

4.7.1 Build initial BOS image

The OpenHPC build of Warewulf includes specific enhancements enabling support for SLES12SP1. 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/sles12sp1`.

Tip

Note that Warewulf requires access to a zypper repository during the `wwmkchroot` process. The easiest way to provide this is to mount the SUSE-SLE-12-SP1-Server ISO image on a web server that can communicate with the *master* host, set the `${BOS_MIRROR}` environment variable to that URL, and update the template file *prior* to running the `wwmkchroot` command below. For example:

```
# Override default OS repository - set BOS_MIRROR variable to desired repo location
[sms]# perl -pi -e "s#^ZYP_MIRROR=(\S+)#ZYP_MIRROR=${BOS_MIRROR}#" \
   /usr/lib/warewulf/wwmkchroot/sles-12.tmpl
```

```
# Define chroot location
[sms]# export CHROOT=/opt/ohpc/admin/images/sles12sp1

# 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 sles-12 $CHROOT                   # create base image
```

4.7.2 Add OpenHPC components

The `wwmkchroot` process used in the previous step is designed to provide a minimal SLES12SP1 configuration. Next, we add additional components to include resource management client services, InfiniBand drivers, and other additional packages to support the default OpenHPC environment. This process augments the chroot-based install performed by `wwmkchroot` to augment the base provisioning image and will access the BOS and OpenHPC repositories to resolve package install requests. 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 components to the compute instance using `zypper` to install into the chroot location defined previously:

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

# Add SLURM client support
[sms]# zypper -n --root $CHROOT install -t pattern ohpc-slurm-client

# Add IB support
[sms]# zypper -n --root $CHROOT install libibverbs-runtime libmlx4-rdmav2 libipathverbs-rdmav2
[sms]# zypper -n --root $CHROOT install libibmad5 librdmacm1 rdma infinipath-psm dapl-devel dapl-utils

# 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 ntp

# 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
```

4.7.3 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.

```
# add new cluster key to base image
[sms]# wwinit ssh_keys
[sms]# cat ~/.ssh/cluster.pub >> $CHROOT/root/.ssh/authorized_keys

# add NFS client mounts of /home and /opt/ohpc/pub to base image
[sms]# echo "${sms_ip}:/home /home nfs nfsvers=3,rsize=1024,wsize=1024,cto 0 0" >> $CHROOT/etc/fstab
[sms]# echo "${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3 0 0" >> $CHROOT/etc/fstab

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

# 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 nfsserver
[sms]# systemctl enable nfsserver
# Enable NTP time service on computes and identify master host as local NTP server
[sms]# chroot $CHROOT systemctl enable ntpd
[sms]# echo "server ${sms_ip}" >> $CHROOT/etc/ntp.conf
[sms]# cp /etc/ntp.keys $CHROOT/etc/ntp.keys
```

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.

4.7.4 Additional Customization (*optional*)

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

- Increase memlock limits
- Restrict ssh access to compute resources
- Add Cluster Checker
- Add Lustre client
- Add Nagios Core monitoring
- Add Ganglia monitoring
- Add ClusterShell
- Add *mrsh*
- Add *genders*
- Add ConMan

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

4.7.4.1 Increase locked memory limits In order to utilize InfiniBand 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 within the *compute* image and 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 the compute image:

```
# 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 within compute image
[sms]# perl -pi -e 's/# End of file/\* soft memlock unlimited\n$&/s' $CHROOT/etc/security/limits.conf
[sms]# perl -pi -e 's/# End of file/\* hard memlock unlimited\n$&/s' $CHROOT/etc/security/limits.conf
```

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

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

4.7.4.3 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`. Additionally, note that a default SLES12SP1 environment may not allow loading of the necessary Lustre kernel modules. Consequently, the example below includes steps which update the /etc/modprobe.d/10-unsupported-modules.conf file to allow loading of the necessary modules.

```
# Add Lustre client software to master host
[sms]# zypper -n install lustre-client-ohpc lustre-client-ohpc-modules

# Update configuration to allow Lustre modules to be loaded on master host
[sms]# perl -pi -e "s/^allow_unsupported_modules 0/allow_unsupported_modules 1/" \
    /etc/modprobe.d/10-unsupported-modules.conf
```

```
# Include Lustre client software in compute image
[sms]# zypper -n --root $CHROOT install lustre-client-ohpc lustre-client-ohpc-modules

# Update configuration to allow Lustre modules to be loaded on compute hosts
[sms]# perl -pi -e "s/^allow_unsupported_modules 0/allow_unsupported_modules 1/" \
    $CHROOT/etc/modprobe.d/10-unsupported-modules.conf

# Include mount point and file system mount in compute image
[sms]# mkdir $CHROOT/mnt/lustre
[sms]# echo "${mgs_fs_name} /mnt/lustre lustre defaults,_netdev,localflock 0 0" >> $CHROOT/etc/fstab
```

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* image.

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

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

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

4.7.4.4 Add Nagios monitoring Nagios is an open source infrastructure monitoring package that monitors servers, switches, applications, and services and offers user-defined alerting facilities. As provided by OpenHPC, it consists of a base monitoring daemon and a set of plug-ins for monitoring various aspects of an HPC cluster. 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.

```
# Install Nagios base, Remote Plugin Engine, and Plugins on master host
[sms]# zypper -n install -t pattern ohpc-nagios

# Install plugins into compute node image
[sms]# zypper -n --root $CHROOT install nagios-plugins-all-ohpc nrpe-ohpc

# Enable and configure NRPE 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 5666/tcp # NRPE"     >> $CHROOT/etc/services
[sms]# echo "nrpe : ${sms_ip} : ALLOW" >> $CHROOT/etc/hosts.allow
[sms]# echo "nrpe : ALL : DENY"        >> $CHROOT/etc/hosts.allow
[sms]# chroot $CHROOT /usr/sbin/useradd -c "NRPE user for the NRPE service" -d /var/run/nrpe \
       -r -g nrpe -s /sbin/nologin nrpe
[sms]# chroot $CHROOT /usr/sbin/groupadd -r nrpe

# Configure remote services to test on compute nodes
[sms]# mv /etc/nagios/conf.d/services.cfg.example /etc/nagios/conf.d/services.cfg

# Define compute nodes as hosts to monitor
[sms]# mv /etc/nagios/conf.d/hosts.cfg.example /etc/nagios/conf.d/hosts.cfg
[sms]# for ((i=0; i<$num_computes; i++)) ; do
            perl -pi -e "s/HOSTNAME$(($i+1))/${c_name[$i]}/ || s/HOST$(($i+1))_IP/${c_ip[$i]}/" \
            /etc/nagios/conf.d/hosts.cfg
          done

# 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

# Enable Nagios on master, and configure
[sms]# chkconfig nagios on
[sms]# systemctl start nagios
[sms]# chmod u+s `which ping`
```

4.7.4.5 Add Ganglia monitoring

Ganglia is a scalable distributed system monitoring tool for high-performance computing systems such as clusters and grids. It allows the user to remotely view live or historical statistics (such as CPU load averages or network utilization) for all machines running the *gmond* daemon. The following commands can be used to enable Ganglia to monitor both the *master* and *compute* hosts.

```
# Install Ganglia
[sms]# zypper -n install -t pattern ohpc-ganglia
[sms]# zypper -n --root $CHROOT install ganglia-gmond-ohpc

# Use example configuration script to enable unicast receiver on master host
[sms]# cp /opt/ohpc/pub/examples/ganglia/gmond.conf /etc/ganglia/gmond.conf
[sms]# perl -pi -e "s/<sms>/${sms_name}/" /etc/ganglia/gmond.conf

# Add configuration to compute image and provide gridname
[sms]# cp /etc/ganglia/gmond.conf $CHROOT/etc/ganglia/gmond.conf
[sms]# echo "gridname MySite" >> /etc/ganglia/gmetad.conf

# Start and enable Ganglia services
[sms]# systemctl enable gmond
[sms]# systemctl enable gmetad
[sms]# systemctl start gmond
[sms]# systemctl start gmetad
[sms]# chroot $CHROOT systemctl enable gmond

# Restart web server
[sms]# systemctl try-restart httpd
```

Once enabled and running, Ganglia should provide access to a web-based monitoring console on the *master* host. Read access to monitoring metrics will be enabled by default and can be accessed via a web browser. When running a web browser directly on the *master* host, the Ganglia top-level overview is available at http://localhost/ganglia. When accessing remotely, replace *localhost* with the chosen name of your master host (`${sms_name}`).

4.7.4.6 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-ohpc

# 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
```

4.7.4.7 Add *mrsh*

mrsh is a secure remote shell utility, like ssh, which uses *munge* for authentication and encryption. By using the *munge* installation used by Slurm, *mrsh* provides shell access to systems using the same *munge* key without having to track ssh keys. Like ssh, *mrsh* provides a remote copy command, *mrcp*, and can be used as a *rcmd* by *pdsh*. Example installation and configuration is as follows:

```
# Install mrsh
[sms]# zypper -n install mrsh-ohpc mrsh-rsh-compat-ohpc
```

```
[sms]# zypper -n --root $CHROOT install mrsh-ohpc mrsh-rsh-compat-ohpc mrsh-server-ohpc

# Identify mshell and mlogin in services file
[sms]# echo "mshell       21212/tcp               # mrshd" >> /etc/services
[sms]# echo "mlogin         541/tcp               # mrlogind" >> /etc/services

# Enable xinetd in compute node image
[sms]# chroot $CHROOT systemctl enable xinetd
```

4.7.4.8 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
```

4.7.4.9 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 §4.8.4 after compute nodes have been registered with the provisioning system.

4.7.4.10 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]# perl -pi -e "s/\\#\\\$ModLoad imudp/\\\$ModLoad imudp/" /etc/rsyslog.conf
[sms]# perl -pi -e "s/\\#\\\$UDPServerRun 514/\\\$UDPServerRun 514/" /etc/rsyslog.conf
[sms]# systemctl restart rsyslog

# Define compute node forwarding destination
[sms]# echo "*.* @${sms_ip}:514" >> $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
```

4.7.5 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* hosts. 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 global Slurm configuration file and 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/slurm/slurm.conf
[sms]# wwsh file import /etc/munge/munge.key
```

Finally, to add *optional* support for controlling IPoIB interfaces (see §4.5), OpenHPC includes a template file for Warewulf that can optionally be imported and used later to provision `ib0` network settings.
Finally, to add *optional* support for controlling IPoIB interfaces (see §4.5), OpenHPC includes a template file for Warewulf that can optionally be imported and used later to provision `ib0` network settings.

```
[sms]# wwsh file import /opt/ohpc/pub/examples/network/sles/ifcfg-ib0.ww
[sms]# wwsh -y file set ifcfg-ib0.ww --path=/etc/sysconfig/network/ifcfg-ib0
```

4.8 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.

4.8.1 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 Lustre drivers; needed if enabling Lustre client on computes
[sms]# export WW_CONF=/etc/warewulf/bootstrap.conf
[sms]# echo "drivers += updates/kernel/" >> $WW_CONF

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

4.8.2 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 -y --chroot $CHROOT
```

4.8.3 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. Included in these steps are commands to enable Warewulf to manage IPoIB settings and corresponding definitions of IPoIB addresses for the compute nodes. This is typically optional unless you are planning to include a Lustre client mount over InfiniBand. 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 §4.7.5.

```
# 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
```

```
# Define provisioning image for hosts
[sms]# wwsh -y provision set "${compute_regex}" --vnfs=sles12sp1 --bootstrap=`uname -r` \
    --files=dynamic_hosts,passwd,group,shadow,slurm.conf,munge.key
```

```
# Optionally define IPoIB network settings (required if planning to mount Lustre over IB)
[sms]# for ((i=0; i<$num_computes; i++)) ; do
            wwsh -y node set ${c_name[$i]} -D ib0 --ipaddr=${c_ipoib[$i]} --netmask=${ipoib_netmask}
       done
[sms]# wwsh -y provision set "${compute_regex}" --fileadd=ifcfg-ib0.ww
```

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=${internal_netmask} \
   --vnfs=sles12sp1 --bootstrap=`uname -r` --listen=${sms_eth_internal} ${c_name[0]}-${c_name[3]}
```

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

4.8.4 Optional kernel arguments

If you chose to enable ConMan in §4.7.4.9, additional boot-time kernel arguments are needed to enable serial console redirection. An example provisioning setting which adds to any other kernel arguments defined in ${kargs} is as follows:

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

4.8.5 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 -y --chroot $CHROOT
```

Enabling stateful nodes also requires additional site-specific, disk-related parameters in the Warewulf configuration. In the example that follows, the compute node configuration is updated to define where to install the GRUB bootloader, which disk to partition, which partition to format, and what the filesystem layout will look like.

```
# Update node object parameters
[sms]# export sda1="mountpoint=/boot:dev=sda1:type=ext3:size=500"
[sms]# export sda2="dev=sda2:type=swap:size=32768"
[sms]# export sda3="mountpoint=/:dev=sda3:type=ext3:size=fill"
[sms]# wwsh -y object modify -s bootloader=sda -t node "${compute_regex}"
[sms]# wwsh -y object modify -s diskpartition=sda -t node "${compute_regex}"
[sms]# wwsh -y object modify -s diskformat=sda1,sda2,sda3 -t node "${compute_regex}"
[sms]# wwsh -y object modify -s filesystems="$sda1,$sda2,$sda3" -t node "${compute_regex}"
```

Upon subsequent reboot of the modified nodes, Warewulf will partition and format the disk to host the desired VNFS image. Once installed to disk, Warewulf can be instructed to subsequently boot from local storage (alternatively, the BIOS boot option order could be updated to reflect a desire to boot from disk):

```
# Update node object parameters
[sms]# wwsh -y object modify -s bootlocal=EXIT -t node "${compute_regex}"
```

Deleting the bootlocal object parameter will cause Warewulf to once again reformat and re-install to local storage upon a new PXE boot request.

4.9 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} 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`.

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, Spack, and R. These can be installed as follows:

```
[sms]# zypper -n install -t pattern ohpc-autotools
[sms]# zypper -n install valgrind-ohpc
[sms]# zypper -n install EasyBuild-ohpc
[sms]# zypper -n install spack-ohpc
[sms]# zypper -n install R_base-ohpc
```

5.2 Compilers

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

```
[sms]# zypper -n install gnu-compilers-ohpc
```

5.3 MPI Stacks

For MPI development support, OpenHPC presently provides pre-packaged builds for two MPI families. Note that for this **Tech Preview** release, MPI tests have only been carried using an *ethernet* transport layer.

```
[sms]# zypper -n install openmpi-gnu-ohpc mpich-gnu-ohpc
```

5.4 Performance Tools

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

```
[sms]# zypper -n install -t pattern ohpc-perf-tools-gnu
```

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 MPI stack.

```
[sms]# zypper -n install lmod-defaults-gnu-openmpi-ohpc
```

Tip

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

- lmod-defaults-gnu-mpich-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.

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

S | Name                    | Summary                                                    | --------
--+-------------------------+------------------------------------------------------------+--------
i | petsc-gnu-mpich-ohpc    | Portable Extensible Toolkit for Scientific Computation | package
i | petsc-gnu-openmpi-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 §4.7.3, 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 C 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:

```
[sms]# zypper -n install -t pattern ohpc-serial-libs-gnu
[sms]# zypper -n install -t pattern ohpc-io-libs-gnu
[sms]# zypper -n install -t pattern ohpc-python-libs-gnu
[sms]# zypper -n install -t pattern ohpc-runtimes-gnu
```

```
# Install parallel libs for all available MPI toolchains
[sms]# zypper -n install -t pattern ohpc-parallel-libs-gnu-mpich
[sms]# zypper -n install -t pattern ohpc-parallel-libs-gnu-openmpi
```

6 Resource Manager Startup

In section §4, 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 c[1-4] systemctl start slurmd
```

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
```

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 c[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 script named `prun` that is installed in concert with the pre-packaged MPI toolchains. At present, OpenHPC is unable to include the PMI process management server normally included within Slurm which implies that `srun` cannot be use for MPI job launch. Instead, native job launch mechanisms provided by the MPI stacks are utilized and `prun` abstracts this process for the various stacks to retain a single launch command.

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 ~]$ 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 PBS Pro:

Command	PBS Pro	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]

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 §2.3. 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/vanilla/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/vanilla/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 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 specfile as needed, and rebuild to obtain an updated binary RPM. A brief example using the FFTW library is highlighted below. Note that the source RPMs can be downloaded from the community build server at https://build.openhpc.community via a web browser or directly via `rpm` as highlighted below. The OpenHPC build system design leverages several keywords to control the choice of compiler and MPI families for relevant development libraries and the `rpmbuild` example illustrates how to override the default mpi_family.

```
# Install rpm-build package from base OS distro
[test@sms ~]$ zypper -n install rpm-build

# Download SRPM from OpenHPC repository and install locally
[test@sms ~]$ rpm -i \
  http://build.openhpc.community/OpenHPC:/1.2/SLE_12_SP1/src/fftw-gnu-openmpi-ohpc-3.3.4-4.1.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 package manager will see an update
[test@sms ~rpmbuild/SPECS]$ perl -pi -e "s/Release: 4.1/Release: 5.1/" fftw.spec

# Rebuild binary RPM. Note that additional directives can be specified to modify build
[test@sms ~rpmbuild/SPECS]$ rpmbuild -bb --define "mpi_family mvapich2" fftw.spec

# As privileged user, install the new package
[sms]# zypper -n install ~test/rpmbuild/RPMS/x86_64/fftw-gnu-mvapich2-ohpc-3.3.4-5.1.rpm
```

C Package Manifest

This appendix provides a summary of available convenience groups and all of the underlying RPM packages that are available as part of this OpenHPC release. The convenience groups are aliases that the underlying package manager supports in order to provide a mechanism to group related collections of packages together. The collection of packages (along with any additional dependencies) can be installed by using the group name. A list of the available convenience groups and a brief description are presented in Table 1.

Table 1: **Available OpenHPC Convenience Groups**

Group Name	Description
ohpc-autotools	Collection of GNU autotools packages.
ohpc-base	OpenHPC base packages.
ohpc-ganglia	Collection of Ganglia monitoring and metrics packages.
ohpc-io-libs-gnu	OpenHPC IO library builds for use with GNU compiler toolchain.
ohpc-nagios	Collection of Nagios monitoring and metrics packages.
ohpc-parallel-libs-gnu-mpich	OpenHPC parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-parallel-libs-gnu-openmpi	OpenHPC parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-perf-tools-gnu	OpenHPC performance tool builds for use with GNU compiler toolchain.
ohpc-python-libs-gnu	OpenHPC python related library builds for use with GNU compiler toolchain.
ohpc-runtimes-gnu	OpenHPC runtimes for use with GNU compiler toolchain.
ohpc-serial-libs-gnu	OpenHPC serial library builds for use with GNU compiler toolchain.
ohpc-slurm-server	OpenHPC server packages for SLURM.
ohpc-warewulf	Collection of base packages for Warewulf provisioning.
ohpc-parallel-libs-gnu	OpenHPC parallel library builds for use with GNU compiler toolchain.
ohpc-slurm-client	OpenHPC client packages for SLURM.

What follows next in this Appendix are 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 contained 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 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)
- Distro support packages and dependencies (Table 9)
- IO Libraries (Table 11)
- Runtimes (Table 12)
- Serial Libraries (Table 13)
- Parallel Libraries (Table 14)

Table 2: **Administrative Tools**

RPM Package Name	Version	Info/URL
conman-ohpc	0.2.7	ConMan: The Console Manager. http://dun.github.io/conman
docs-ohpc	1.2.1	OpenHPC documentation. https://github.com/openhpc/ohpc
examples-ohpc	1.4	Example source code and templates for use within OpenHPC environment. https://github.com/openhpc/ohpc
ganglia-ohpc	3.7.2	Distributed Monitoring System. http://ganglia.sourceforge.net
genders-ohpc	1.22	Static cluster configuration database. https://github.com/chaos/genders
lmod-defaults-gnu-mpich-ohpc lmod-defaults-gnu-openmpi-ohpc	1.2	OpenHPC default login environments. https://github.com/openhpc/ohpc
lmod-ohpc	6.5.11	Lua based Modules (lmod). https://github.com/TACC/Lmod
losf-ohpc	0.53.0	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. None
nagios-ohpc	4.1.1	Nagios monitors hosts and services and yells if something breaks. http://www.nagios.org
nagios-plugins-ohpc	2.1.1	Host/service/network monitoring program plugins for Nagios. https://www.nagios-plugins.org
ohpc-release	1.2	OpenHPC release files. https://github.com/openhpc/ohpc
pdsh-ohpc	2.31	Parallel remote shell program. http://sourceforge.net/projects/pdsh
prun-ohpc	1.1	Convenience utility for parallel job launch. https://github.com/openhpc/ohpc

Table 3: **Provisioning**

RPM Package Name	Version	Info/URL
warewulf-cluster-node-ohpc	3.6	Tools used for clustering with Warewulf. http://warewulf.lbl.gov
warewulf-cluster-ohpc	3.6	Tools used for clustering with Warewulf. http://warewulf.lbl.gov
warewulf-common-ohpc	3.6	A suite of tools for clustering. http://warewulf.lbl.gov
warewulf-ipmi-ohpc	3.6	IPMI Module for Warewulf. http://warewulf.lbl.gov
warewulf-nhc-ohpc	1.4.1	Warewulf Node Health Check System. http://warewulf.lbl.gov
warewulf-provision-ohpc	3.6	Warewulf - Provisioning Module. http://warewulf.lbl.gov
warewulf-provision-server-ohpc	3.6	Warewulf - Provisioning Module - Server. http://warewulf.lbl.gov
warewulf-vnfs-ohpc	3.6	Warewulf VNFS Module. http://warewulf.lbl.gov

Table 4: **Resource Management**

RPM Package Name	Version	Info/URL
munge-ohpc	0.5.12	MUNGE authentication service. http://dun.github.io/munge
pbspro-client-ohpc	14.1.0	PBS Professional for a client host. https://github.com/pbspro/pbspro
pbspro-execution-ohpc	14.1.0	PBS Professional for an execution host. https://github.com/pbspro/pbspro
pbspro-server-ohpc	14.1.0	PBS Professional for a server host. https://github.com/pbspro/pbspro
slurm-devel-ohpc	16.05.8	Development package for Slurm. http://slurm.schedmd.com
slurm-munge-ohpc	16.05.8	Slurm authentication and crypto implementation using Munge. http://slurm.schedmd.com
slurm-ohpc	16.05.8	Slurm Workload Manager. http://slurm.schedmd.com
slurm-pam_slurm-ohpc	16.05.8	PAM module for restricting access to compute nodes via Slurm. http://slurm.schedmd.com
slurm-perlapi-ohpc	16.05.8	Perl API to Slurm. http://slurm.schedmd.com
slurm-plugins-ohpc	16.05.8	Slurm plugins (loadable shared objects). http://slurm.schedmd.com
slurm-sjobexit-ohpc	16.05.8	Slurm job exit code management tools. http://slurm.schedmd.com
slurm-sjstat-ohpc	16.05.8	Perl tool to print Slurm job state information. http://slurm.schedmd.com
slurm-slurmdb-direct-ohpc	16.05.8	Wrappers to write directly to the slurmdb. http://slurm.schedmd.com
slurm-slurmdbd-ohpc	16.05.8	Slurm database daemon. http://slurm.schedmd.com
slurm-sql-ohpc	16.05.8	Slurm SQL support. http://slurm.schedmd.com
slurm-torque-ohpc	16.05.8	Torque/PBS wrappers for transition from Torque/PBS to Slurm. http://slurm.schedmd.com

Table 5: **Compiler Families**

RPM Package Name	Version	Info/URL
gnu-compilers-ohpc	5.4.0	The GNU C Compiler and Support Files. http://gcc.gnu.org

Table 6: **MPI Families**

RPM Package Name	Version	Info/URL
mpich-gnu-ohpc	3.2	MPICH MPI implementation. http://www.mpich.org
openmpi-gnu-ohpc	1.10.4	A powerful implementation of MPI. http://www.open-mpi.org

Table 7: **Development Tools**

RPM Package Name	Version	Info/URL
autoconf-ohpc	2.69	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
automake-ohpc	1.15	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake
libtool-ohpc	2.4.6	The GNU Portable Library Tool. http://www.gnu.org/software/libtool
python-numpy-gnu-ohpc	1.11.1	NumPy array processing for numbers, strings, records and objects. http://sourceforge.net/projects/numpy
python-scipy-gnu-mpich-ohpc python-scipy-gnu-openmpi-ohpc	0.18.0	Scientific Tools for Python. http://www.scipy.org
R_base-ohpc	3.3.1	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
valgrind-ohpc	3.11.0	Valgrind Memory Debugger. http://www.valgrind.org

Table 8: **Performance Analysis Tools**

RPM Package Name	Version	Info/URL
imb-gnu-mpich-ohpc imb-gnu-openmpi-ohpc	4.1	Intel MPI Benchmarks (IMB). https:
mpiP-gnu-mpich-ohpc mpiP-gnu-openmpi-ohpc	3.4.1	mpiP: a lightweight profiling library for MPI applications. http://mpip.sourceforge.net
papi-ohpc	5.4.3	Performance Application Programming Interface. http://icl.cs.utk.edu/papi
pdtoolkit-gnu-ohpc	3.22	PDT is a framework for analyzing source code. http://www.cs.uoregon.edu/Research/pdt
scalasca-gnu-mpich-ohpc scalasca-gnu-openmpi-ohpc	2.3.1	Toolset for performance analysis of large-scale parallel applications. http://www.scalasca.org
scorep-gnu-mpich-ohpc scorep-gnu-openmpi-ohpc	3.0	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.vi-hps.org/projects/score-p
sionlib-gnu-mpich-ohpc sionlib-gnu-openmpi-ohpc	1.7.0	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html
tau-gnu-mpich-ohpc tau-gnu-openmpi-ohpc	2.26	Tuning and Analysis Utilities Profiling Package. http://www.cs.uoregon.edu/research/tau/home.php

Table 9: **Distro Support Packages/Dependencies**

RPM Package Name	Version	Info/URL
lua-bit-ohpc	1.0.2	Module for Lua which adds bitwise operations on numbers. http://bitop.luajit.org
lua-filesystem-ohpc	1.6.3	Lua library to Access Directories and Files. http://keplerproject.github.com/luafilesystem
lua-posix-ohpc	33.2.1	POSIX library for Lua. https://github.com/luaposix/luaposix

Table 10: **Lustre**

RPM Package Name	Version	Info/URL
lustre-client-ohpc	2.8.0	Lustre File System. https://wiki.hpdd.intel.com

Table 11: **IO Libraries**

RPM Package Name	Version	Info/URL
adios-gnu-mpich-ohpc adios-gnu-openmpi-ohpc	1.10.0	The Adaptable IO System (ADIOS). http://www.olcf.ornl.gov/center-projects/adios
hdf5-gnu-ohpc	1.8.17	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx-gnu-mpich-ohpc netcdf-cxx-gnu-openmpi-ohpc	4.2.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran-gnu-mpich-ohpc netcdf-fortran-gnu-openmpi-ohpc	4.4.4	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-gnu-mpich-ohpc netcdf-gnu-openmpi-ohpc	4.4.1	C Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
phdf5-gnu-mpich-ohpc phdf5-gnu-openmpi-ohpc	1.8.17	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5

Table 12: **Runtimes**

RPM Package Name	Version	Info/URL
ocr-gnu-ohpc	1.0.1	Open Community Runtime (OCR) for shared memory. https://xstack.exascale-tech.com/wiki

Table 13: **Serial Libraries**

RPM Package Name	Version	Info/URL
gsl-gnu-ohpc	2.2.1	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis-gnu-ohpc	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
openblas-gnu-ohpc	0.2.19	An optimized BLAS library based on GotoBLAS2. http://www.openblas.net
superlu-gnu-ohpc	5.2.1	A general purpose library for the direct solution of linear equations. http://crd.lbl.gov/~xiaoye/SuperLU

Table 14: **Parallel Libraries**

RPM Package Name	Version	Info/URL
boost-gnu-mpich-ohpc boost-gnu-openmpi-ohpc	1.61.0	Boost free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw-gnu-mpich-ohpc fftw-gnu-openmpi-ohpc	3.3.4	A Fast Fourier Transform library. http://www.fftw.org
hypre-gnu-mpich-ohpc hypre-gnu-openmpi-ohpc	2.10.1	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mumps-gnu-mpich-ohpc mumps-gnu-openmpi-ohpc	5.0.2	A MUltifrontal Massively Parallel Sparse direct Solver. http://mumps.enseeiht.fr
petsc-gnu-mpich-ohpc petsc-gnu-openmpi-ohpc	3.7.0	Portable Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc
superlu_dist-gnu-mpich-ohpc superlu_dist-gnu-openmpi-ohpc	4.2	A general purpose library for the direct solution of linear equations. http://crd-legacy.lbl.gov/~xiaoye/SuperLU
trilinos-gnu-mpich-ohpc trilinos-gnu-openmpi-ohpc	12.6.4	A collection of libraries of numerical algorithms. http://trilinos.sandia.gov/index.html

D 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:  DD5D 8CAA CB57 364F FCC2 D3AE C468 07FF 26CE 6884
```

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-1.0-1.1.x86_64.rpm:
    Header V3 RSA/SHA256 Signature, key ID 26ce6884: OK
    Header SHA1 digest: OK (c3873bf495c51d2ea6d3ef23ab88be105983c72c)
    V3 RSA/SHA256 Signature, key ID 26ce6884: OK
    MD5 digest: OK (43d067f33fb370e30a39789439ead238)
```