OpenHPC (v1.1)
Cluster Building Recipes

SLES12 Base OS
Base Linux* Edition

Document Last Update: 2016-03-09
Document Revision: 7e16cb3

Legal Notice

Redistribution and use of the enclosed documentation ("Documentation"), with or without modification, are permitted provided that the following conditions are met:

- Redistributions of this Documentation in whatever format (e.g. PDF, HTML, RTF, shell script, PostScript, etc.), must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Intel Corporation nor the names of its contributors may be used to endorse or promote products derived from this Documentation without specific prior written permission.

Except as expressly provided above, no license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

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 and represent a mix of open-source components combined with Intel development and analysis tools (e.g. Intel® Parallel Studio XE). 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

If you don't know where you are going, you might wind up someplace else. –Yogi Berra

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 SLES12 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.

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

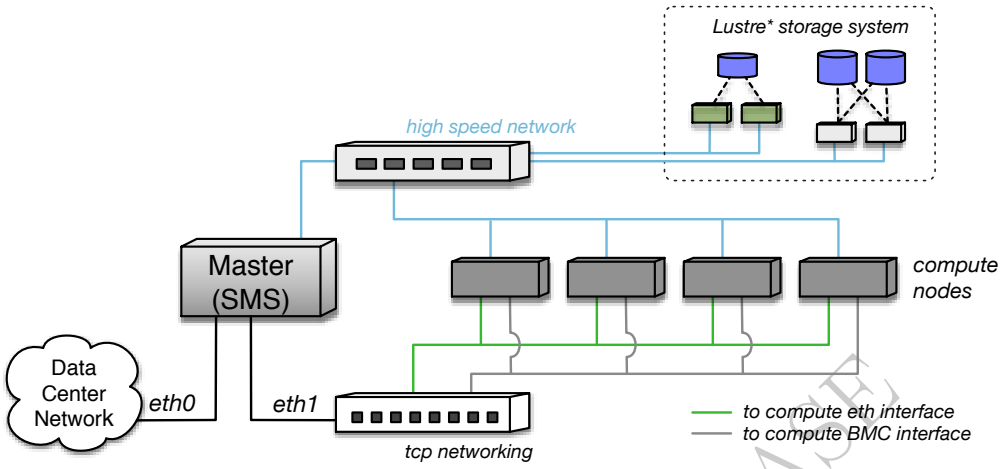

Figure 1: Overview of physical cluster architecture.

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.

1.3 Bring your own license

OpenHPC provides a variety of integrated, pre-packaged elements from the Intel® Parallel Studio XE software suite. Portions of the runtimes provided by the included compilers and MPI components are freely usable. However, in order to compile new binaries using these tools (or access other analysis tools like Intel® Trace Analyzer and Collector, Intel® Inspector, etc.), you will need to provide your own valid license and OpenHPC adopts a *bring-your-own-license* model. Note that licenses are provided free of charge for many categories of use. In particular, licenses for compilers and developments tools are provided at no cost to academic researchers or developers contributing to open-source software projects. More information on this program can be found at:

https://software.intel.com/en-us/qualify-for-free-software

1.4 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.

- `${ohpc_repo}` # OpenHPC repo location
- `${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
- `${bmc_password}` # BMC password for use by IPMI
- `${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 for matching 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

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 SLES12 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 SLES12 distribution.

Tip

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.

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 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:/1.0/SLES12/x86_64/ohpc-release-1.1-1.x86_64.rpm
```

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 SLES12, the requirements are to have access to both the base and SDK repositories:

- SLES12-12-0
- SLES12-12-sdk

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 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 (§ 1.4) 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 B. 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
```

Provisioning services with Warewulf rely on DHCP, TFTP, and HTTP network protocols. Depending on the local Base OS 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
```

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 ntp server `${ntp_server}`, issue the following:

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

3.4 Add resource management services on *master* node

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
```

Slurm requires the designation of a system user that runs the underlying resource management daemons. The default configuration file that is supplied with the OpenHPC build of Slurm identifies this `SlurmUser` to be a dedicated user named "slurm" and this user must exist. The following command can be used to add this user to the *master* server:

```
[sms]# useradd slurm
```

3.5 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
```

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 §3.7.4.4 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
```

3.6 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 SLES12 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
```


3.7 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.

3.7.1 Build initial BOS image

The OpenHPC build of Warewulf includes specific enhancements enabling support for SLES12. 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/sles12`. Note that Warewulf assumes access to an external repository (mirror.centos.org) during the `wwmkchroot` process which requires external network connectivity on your chosen *master* host. If this is not available, or if you prefer to access a locally cached mirror, the steps below include an example to update the OS mirror used by `wwmkchroot` to use an alternate location defined by a `${BOS_MIRROR}` environment variable.

```
# Define chroot location
```

```
[sms]# export CHROOT=/opt/ohpc/admin/images/sles12

# 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
```

3.7.2 Add OpenHPC components

The `wwmkchroot` process used in the previous step is designed to provide a minimal SLES12 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 uses the `chroot` command 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
```



```
# 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

# 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
```

3.7.3 Customize system configuration

Prior to assembling the image, it is advantageous to perform any additional customizations 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

# 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
```

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.

3.7.4 Additional Customizations (*optional*)

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

- Increase memlimits to support MPI jobs over InfiniBand
- Restrict ssh access to compute resources
- Add Cluster Checker
- Add Lustre client

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

3.7.4.1 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]# echo "* soft memlock unlimited" >> /etc/security/limits.conf
[sms]# echo "* hard memlock unlimited" >> /etc/security/limits.conf

# Update memlock settings within compute image
```

```
[sms]# echo "* soft memlock unlimited" >> $CHROOT/etc/security/limits.conf
[sms]# echo "* hard memlock unlimited" >> $CHROOT/etc/security/limits.conf
```

3.7.4.2 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
```

3.7.4.3 Add Cluster Checker The Intel® Cluster Checker provides a convenient suite of diagnostics that can be used to aid in isolating hardware and software problems on an installed cluster. This package can be optionally added to the SMS and compute image using the commands below. Note that a valid license file will also need to be installed in order to use this software (recall discussion in §1.3).

```
[sms]# zypper -n install intel-clck-ohpc
[sms]# zypper -n --root $CHROOT install intel-clck-ohpc
```

3.7.4.4 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 SLES12 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
```

3.7.4.5 Add stateful provisioning support

In addition to running the VNFS image from memory, Warewulf can also partition and format persistent storage on a node and maintain the VNFS state across reboots. This requires installing additional packages in the VNFS image to support a boot loader (GRUB), and it also requires some modification to the node definition in the warewulf data store.

```
# Add GRUB2 Bootloader
[sms]# zypper -n --root $CHROOT install grub2

# Re-assemble VNFS image
[sms]# wwvnfs -y --chroot $CHROOT
```

Stateful nodes require addition parameters in the Warewulf configuration. In particular, we instruct Warewulf where to install the GRUB bootloader, which disk to partition, which partition to format, and what the filesystem layout will look like. These changes are necessary for each compute node we wish to provision statefully.

```
# Update node object parameters
[sms]# export filesystems="mountpoint=/boot:dev=sda1:type=ext3:size=500," \
            "dev=sda2:type=swap:size=32768," \
            "mountpoint=/:dev=sda3:type=ext3:size=fill"
[sms]# for ((i=0; i<$num_computes; i++)) ; do
            wwsh -y object modify -s bootloader=sda ${c_name[$i]};
            wwsh -y object modify -s diskpartition=sda ${c_name[$i]};
            wwsh -y object modify -s diskformat=sda1,sda2,sda3 ${c_name[$i]};
            wwsh -y object modify -s filesystems="$filesystems" ${c_name[$i]};
         done
```

Upon the node's next reboot, Warewulf will partition and format the disk. It will also run grub2-mkconfig and grub2-install to place grub in the proscribed boot sector, and it will install the VNFS locally. In order

to ensure this process happens only once per node we must instruct warewulf to next boot from the local storage.

```
# Update node object parameters
[sms]# for ((i=0; i<$num_computes; i++)) ; do
           wwsh -y object modify -s bootlocal=EXIT ${c_name[$i]};
       done
```

Deleting the bootlocal object parameter will cause Warewulf once again to re-partition and format the local storage.

3.7.4.6 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
```

3.7.5 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. It also provides a convenient mechanism to provide necessary global configuration files required by services such as Slurm. 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 support for controlling IPoIB interfaces, 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
```

3.8 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.

3.8.1 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`
```

3.8.2 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
```

3.8.3 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 §3.7.5.

```
# Define four compute nodes and network settings
[sms]# wwsh -y node new ${c_name[0]} --ipaddr=${c_ip[0]} --hwaddr=${c_mac[0]} -D ${eth_provision}
[sms]# wwsh -y node new ${c_name[1]} --ipaddr=${c_ip[1]} --hwaddr=${c_mac[1]} -D ${eth_provision}
[sms]# wwsh -y node new ${c_name[2]} --ipaddr=${c_ip[2]} --hwaddr=${c_mac[2]} -D ${eth_provision}
[sms]# wwsh -y node new ${c_name[3]} --ipaddr=${c_ip[3]} --hwaddr=${c_mac[3]} -D ${eth_provision}
```

Tip

Warewulf ships with a utility to automatically register new compute nodes. 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.

```
[sms]# wwnodescan --netdev=${eth_provision} --ipaddr=${c_ip[0]} --netmask=${internal_netmask} \
    --vnfs=sles12 --bootstrap=`uname -r` ${c_name[0]}-${c_name[3]}
```

```
# Optionally register additional compute nodes
[sms]# for ((i=4; 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=sles12 --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
```

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

3.9 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]# ipmitool -E -I lanplus -H ${c_bmc[0]} -U ${bmc_username} chassis power reset # power cycle c1
[sms]# ipmitool -E -I lanplus -H ${c_bmc[1]} -U ${bmc_username} chassis power reset # power cycle c2
[sms]# ipmitool -E -I lanplus -H ${c_bmc[2]} -U ${bmc_username} chassis power reset # power cycle c3
[sms]# ipmitool -E -I lanplus -H ${c_bmc[3]} -U ${bmc_username} chassis power reset # power cycle c4

# Optionally boot additional compute nodes
[sms]# for ((i=4; 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 `[l]pxelinux.0` that ships with `warewulf-provision-ohpc` support a wide range of hardware some hosts may boot more reliably or faster using the BOS provided `[l]pxelinux.0` that comes from `syslinux-tftpboot`. Replace the `[l]pxelinux.0` from `warewulf-provision-ohpc` version if necessary.

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, including the addition of Intel licensed software (e.g. Composer compiler suite, Intel MPI). It is assumed that the end-site administrator will procure and install the necessary licenses in order to use the Intel proprietary software.

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 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
```

4.2 Compilers

OpenHPC presently packages two compiler families (GNU and Intel® Parallel Studio) that are integrated within the underlying modules-environment system in a hierarchical fashion. End users of an OpenHPC system can choose to access one compiler at a time and will be presented with additional compiler-dependent software as a function of which compiler toolchain is currently loaded. Each compiler toolchain can be installed separately and the following commands illustrate the installation of both along with any necessary dependencies:

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

Tip

OpenHPC provides the Intel® compiler development and runtime environment as a convenience, but a valid license is necessary to enable the compilers for local compilations. Please refer to §1.3 for more details.

4.3 MPI Stacks

For MPI development support, OpenHPC presently provides pre-packaged builds for three MPI families:

- MVAPICH2
- OpenMPI
- Intel MPI

For consistency, each of the open-source MPI families (OpenMPI and MVAPICH2) is built against each of the two supported compiler families resulting in total of four build combinations. The Intel MPI stack is also configured to support both the GNU and Intel compiler toolchain directly, but is packaged as a single RPM. Installation of all of the MPI family instances, can be accomplished via the following command. Note the use of wildcards (*) in this example in order to install both GNU and Intel builds for OpenMPI and MVAPICH2. In addition to installing available MPI toolchains, the example below includes installation of a suite of pre-built MPI benchmarks (IMB) that are also provided via OpenHPC.

```
[sms]# zypper -n install openmpi-*-ohpc mvapich2-*-ohpc intel-mpi-ohpc
[sms]# zypper -n install -t pattern ohpc-imb
```

4.4 Performance Tools

To aid in application performance analysis, OpenHPC provides a variety of open-source and Intel licensed software (see §1.3 for more details regarding Intel software). These can be installed as follows:

```
[sms]# zypper -n install papi-ohpc
[sms]# zypper -n install intel-itac-ohpc
[sms]# zypper -n install intel-vtune-ohpc
[sms]# zypper -n install intel-advisor-ohpc
[sms]# zypper -n install intel-inspector-ohpc
[sms]# zypper -n install -t pattern ohpc-mpiP
[sms]# zypper -n install -t pattern ohpc-tau
```

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 MVAPICH2 MPI stack.

```
[sms]# zypper -n install lmod-defaults-gnu-mvapich2-ohpc
```

Tip

If you want to change the default environment from the suggestion above, OpenHPC provides additional choices for each of the available compiler/MPI combinations. In particular, the following packages can be substituted instead:

- lmod-defaults-gnu-openmpi-ohpc
- lmod-defaults-gnu-impi-ohpc
- lmod-defaults-intel-mvapich2-ohpc
- lmod-defaults-intel-openmpi-ohpc
- lmod-defaults-intel-impi-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 FFTW 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 fftw-gnu-*-ohpc
Loading repository data...
Reading installed packages...

S | Name                | Summary                          | Type
--+---------------------+----------------------------------+--------
  | fftw-gnu-impi-ohpc   | A Fast Fourier Transform library | package
  | fftw-gnu-mvapich2-ohpc | A Fast Fourier Transform library | package
  | fftw-gnu-openmpi-ohpc | A Fast Fourier Transform library | 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 §3.7.3, 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 all of the available compiler/MPI family permutations. To install all of the available package offerings within OpenHPC, issue the following:

```
[sms]# zypper -n install -t pattern ohpc-adios
[sms]# zypper -n install -t pattern ohpc-boost
[sms]# zypper -n install -t pattern ohpc-fftw
[sms]# zypper -n install -t pattern ohpc-gsl
[sms]# zypper -n install -t pattern ohpc-hdf5
[sms]# zypper -n install -t pattern ohpc-hypre
[sms]# zypper -n install -t pattern ohpc-metis
[sms]# zypper -n install -t pattern ohpc-mumps
[sms]# zypper -n install -t pattern ohpc-netcdf
[sms]# zypper -n install -t pattern ohpc-numpy
[sms]# zypper -n install -t pattern ohpc-ocr
[sms]# zypper -n install -t pattern ohpc-openblas
[sms]# zypper -n install -t pattern ohpc-petsc
[sms]# zypper -n install -t pattern ohpc-phdf5
[sms]# zypper -n install -t pattern ohpc-scalapack
[sms]# zypper -n install -t pattern ohpc-scipy
[sms]# zypper -n install -t pattern ohpc-trilinos
```

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 munge and slurm clients on compute hosts
[sms]# pdsh -w c[1-4] systemctl start munge
[sms]# pdsh -w c[1-4] systemctl start slurmd
```

In the default configuration, the *compute* hosts will be initialized in an *unknown* state. To place the hosts into production such that they are eligible to schedule user jobs, issue the following:

```
[sms]# scontrol update nodename=c[1-4] state=idle
```


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 resync process can be accomplished as follows:

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

Tip

After resyncing 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.

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] 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
```

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 §1.4. By default, the template install 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 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
fsp-adios	FSP adios IO library builds for various compiler/MPI combinations.
fsp-autotools	Collection of GNU autotools packages.
fsp-base	FSP base packages.
fsp-boost	FSP Boost library builds for various compiler/MPI combinations.
fsp-fftw	FSP FFTW library builds for various compiler/MPI combinations.
fsp-gsl	ForestPeak GSL library builds for various compiler combinations.
fsp-hdf5	FSP HDF5 library builds for various compiler combinations.
fsp-hypre	FSP Hypre library builds for various compiler/MPI combinations.
fsp-imb	FSP builds for Intel IMB MPI benchmarks.
fsp-metis	ForestPeak METIS library builds for various compiler combinations.
fsp-mpiP	FSP mpiP profiling library builds for various compiler/MPI combinations.
fsp-mumps	FSP Mumps library builds for various compiler/MPI combinations.
fsp-netcdf	FSP NetCDF library builds for various compiler/MPI combinations.
fsp-numpy	ForestPeak Numpy python module builds for various compiler combinations.
fsp-petsc	FSP PETSC library builds for various compiler/MPI combinations.
fsp-phdf5	FSP HDF5 parallel library builds for various compiler/MPI combinations.
fsp-scipy	FSP Scientific Python library builds for various compiler/MPI combinations.
fsp-slurm-client	FSP client packages for SLURM.
fsp-slurm-server	FSP server packages for SLURM.
fsp-superlu_dist	FSP Superlu_dist library builds for various compiler/MPI combinations.
fsp-tau	FSP TAU library builds for various compiler/MPI combinations.
fsp-trilinos	FSP Trilinos library builds for various compiler/MPI combinations.
fsp-warewulf	Base packages for Warewulf provisioning.

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 §4.6. The relevant package groupings and associated Table reference 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)
- Serial Libraries (Table 12)
- Parallel Libraries (Table 13)

Table 2: **Administrative Tools**

RPM Package Name	Version	Info/URL
conman-fsp	0.2.7	ConMan: The Console Manager. http://conman.googlecode.com
docs-fsp	15.31	Forest Peak documentation.
examples-fsp	1.2	Example source code and templates for use within FSP environment.
intel-clck-fsp	2.2.2	Intel(R) Cluster Checker. http://intel.com/go/cluster
lmod-defaults-intel-fsp	1.1	FSP default login environment.
lmod-fsp	5.9.3	Lua based Modules (lmod). https://github.com/TACC/Lmod
losf-fsp	0.52.0	A Linux operating system framework for managing HPC clusters. https://github.com/hpcsi/losf
pdsh-fsp	2.31	Parallel remote shell program. http://sourceforge.net/projects/pdsh
prun-fsp	0.1.0	Convenience utility for parallel job launch.

Table 3: **Provisioning**

RPM Package Name	Version	Info/URL
warewulf-cluster-fsp	3.6	Tools used for clustering with Warewulf. http://warewulf.lbl.gov
warewulf-cluster-node-fsp	3.6	Tools used for clustering with Warewulf. http://warewulf.lbl.gov
warewulf-common-fsp	3.6	A suite of tools for clustering. http://warewulf.lbl.gov
warewulf-provision-fsp	3.6	Warewulf - Provisioning Module. http://warewulf.lbl.gov
warewulf-provision-server-fsp	3.6	Warewulf - Provisioning Module - Server. http://warewulf.lbl.gov
warewulf-vnfs-fsp	3.6	Warewulf VNFS Module. http://warewulf.lbl.gov

Table 4: **Resource Management**

RPM Package Name	Version	Info/URL
munge-fsp	0.5.11	MUNGE authentication service. https://munge.googlecode.com
slurm-devel-fsp	14.11.5	Development package for Slurm. http://slurm.schedmd.com
slurm-fsp	14.11.5	Slurm Workload Manager. http://slurm.schedmd.com
slurm-munge-fsp	14.11.5	Slurm authentication and crypto implementation using Munge. http://slurm.schedmd.com
slurm-perlapi-fsp	14.11.5	Perl API to Slurm. http://slurm.schedmd.com
slurm-plugins-fsp	14.11.5	Slurm plugins (loadable shared objects). http://slurm.schedmd.com
slurm-slurmdb-direct-fsp	14.11.5	Wrappers to write directly to the slurmdb. http://slurm.schedmd.com
slurm-slurmdbd-fsp	14.11.5	Slurm database daemon. http://slurm.schedmd.com
slurm-sql-fsp	14.11.5	Slurm SQL support. http://slurm.schedmd.com

Table 5: **Compiler Families**

RPM Package Name	Version	Info/URL
gnu-compilers-fsp	4.9.2	The GNU C Compiler and Support Files. http://gcc.gnu.org
intel-compilers-devel-fsp	16.0.069	Intel(R) Parallel Studio XE. http://www.intel.com/software/products
intel-compilers-fsp	16.0.069	Intel(R) Parallel Studio XE. http://www.intel.com/software/products

Table 6: **MPI Families**

RPM Package Name	Version	Info/URL
intel-mpi-devel-fsp	5.1.0.069	Intel(R) MPI Library for Linux* OS. https://software.intel.com/en-us/intel-mpi-library
intel-mpi-fsp	5.1.0.069	Intel(R) MPI Library for Linux* OS. https://software.intel.com/en-us/intel-mpi-library
mvapich2-gnu-fsp mvapich2-intel-fsp	2.1	OSU MVAPICH2 MPI implementation. http://mvapich.cse.ohio-state.edu/overview/mvapich2
openmpi-gnu-fsp openmpi-intel-fsp	1.8.4	A powerful implementation of MPI. http://www.open-mpi.org

Table 7: **Development Tools**

RPM Package Name	Version	Info/URL
R_base-fsp	3.2.0	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
autoconf-fsp	2.69	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
automake-fsp	1.15	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake
intel-inspector-fsp	16.0.1.407184	Intel(R) Inspector XE. http://www.intel.com/software/products
libtool-fsp	2.4.6	The GNU Portable Library Tool. http://www.gnu.org/software/libtool
python-numpy-gnu-fsp python-numpy-intel-fsp	1.9.2	NumPy array processing for numbers, strings, records and objects. http://sourceforge.net/projects/numpy
python-scipy-gnu-impi-fsp python-scipy-gnu-mvapich2-fsp python-scipy-gnu-openmpi-fsp	0.15.1	Scientific Tools for Python. http://www.scipy.org
valgrind-fsp	3.10.1	Valgrind Memory Debugger. http://www.valgrind.org

Table 8: **Performance Analysis Tools**

RPM Package Name	Version	Info/URL
imb-gnu-impi-fsp imb-gnu-mvapich2-fsp imb-gnu-openmpi-fsp imb-intel-impi-fsp imb-intel-mvapich2-fsp imb-intel-openmpi-fsp	4.0.2	Intel MPI Benchmarks (IMB). https://software.intel.com/en-us/articles/intel-mpi-benchmarks
intel-advisor-fsp	16.0.70.414655	Intel(R) Advisor XE. http://www.intel.com/software/products
intel-itac-fsp	9.1.0.010	Intel(R) Trace Analyzer and Collector. http://www.intel.com/software/products
intel-vtune-fsp	16.0.1.414512	Intel(R) VTune(TM) Amplifier XE 2015 Update 2. http://www.intel.com/software/products
mpiP-gnu-impi-fsp mpiP-gnu-mvapich2-fsp mpiP-gnu-openmpi-fsp mpiP-intel-impi-fsp mpiP-intel-mvapich2-fsp mpiP-intel-openmpi-fsp	3.4.1	mpiP: a lightweight profiling library for MPI applications. http://mpip.sourceforge.net
papi-fsp	5.4.1	Performance Application Programming Interface. http://icl.cs.utk.edu/papi
pdtoolkit-gnu-fsp pdtoolkit-intel-fsp	3.20	PDT is a framework for analyzing source code. http://www.cs.uoregon.edu/Research/pdt
tau-gnu-impi-fsp tau-gnu-mvapich2-fsp tau-gnu-openmpi-fsp tau-intel-impi-fsp tau-intel-mvapich2-fsp tau-intel-openmpi-fsp	2.24	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-filesystem-fsp	1.6.3	Lua library to Access Directories and Files. http://keplerproject.github.com/luafilesystem

Table 10: **Lustre**

RPM Package Name	Version	Info/URL
lustre-client-fsp	2.7.0	Lustre File System. https://wiki.hpdd.intel.com

Table 11: **IO Libraries**

RPM Package Name	Version	Info/URL
adios-gnu-impi-fsp adios-gnu-mvapich2-fsp adios-gnu-openmpi-fsp adios-intel-impi-fsp adios-intel-mvapich2-fsp adios-intel-openmpi-fsp	1.8.0	The Adaptable IO System (ADIOS). http://www.olcf.ornl.gov/center-projects/adios
hdf5-gnu-fsp hdf5-intel-fsp	1.8.14	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx-gnu-impi-fsp netcdf-cxx-gnu-mvapich2-fsp netcdf-cxx-gnu-openmpi-fsp netcdf-cxx-intel-impi-fsp netcdf-cxx-intel-mvapich2-fsp netcdf-cxx-intel-openmpi-fsp	4.2.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran-gnu-impi-fsp netcdf-fortran-gnu-mvapich2-fsp netcdf-fortran-gnu-openmpi-fsp netcdf-fortran-intel-impi-fsp netcdf-fortran-intel-mvapich2-fsp netcdf-fortran-intel-openmpi-fsp	4.4.2	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-gnu-impi-fsp netcdf-gnu-mvapich2-fsp netcdf-gnu-openmpi-fsp netcdf-intel-impi-fsp netcdf-intel-mvapich2-fsp netcdf-intel-openmpi-fsp	4.3.3	C Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
phdf5-gnu-impi-fsp phdf5-gnu-mvapich2-fsp phdf5-gnu-openmpi-fsp phdf5-intel-impi-fsp phdf5-intel-mvapich2-fsp phdf5-intel-openmpi-fsp	1.8.14	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5

Table 12: **Serial Libraries**

RPM Package Name	Version	Info/URL
gsl-gnu-fsp	1.16	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis-gnu-fsp metis-intel-fsp	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
superlu-gnu-fsp superlu-intel-fsp	4.3	A general purpose library for the direct solution of linear equations. http://crd.lbl.gov/ xiaoye/SuperLU

Table 13: **Parallel Libraries**

RPM Package Name	Version	Info/URL
boost-gnu-impi-fsp boost-gnu-mvapich2-fsp boost-gnu-openmpi-fsp boost-intel-impi-fsp boost-intel-mvapich2-fsp boost-intel-openmpi-fsp	1.58.0	Boost free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw-gnu-impi-fsp fftw-gnu-mvapich2-fsp fftw-gnu-openmpi-fsp	3.3.4	A Fast Fourier Transform library. http://www.fftw.org
hypre-gnu-impi-fsp hypre-gnu-mvapich2-fsp hypre-gnu-openmpi-fsp hypre-intel-impi-fsp hypre-intel-mvapich2-fsp hypre-intel-openmpi-fsp	2.10.0b	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mkl-blacs-gnu-openmpi-fsp	11.3	Intel(R) Math Kernel Library Basic Linear Algebra
mumps-gnu-impi-fsp mumps-gnu-mvapich2-fsp mumps-gnu-openmpi-fsp mumps-intel-impi-fsp mumps-intel-mvapich2-fsp mumps-intel-openmpi-fsp	5.0.0	Communication Subprograms. http://software.intel.com/en-us/intel-mkl A MUltifrontal Massively Parallel Sparse direct Solver. http://mumps.enseeiht.fr
petsc-gnu-impi-fsp petsc-gnu-mvapich2-fsp petsc-gnu-openmpi-fsp petsc-intel-impi-fsp petsc-intel-mvapich2-fsp petsc-intel-openmpi-fsp	3.5.3	Portable Extensible Toolkit for Scientific Computation. http://www-unix.mcs.anl.gov/petsc/petsc-as
superlu_dist-gnu-impi-fsp superlu_dist-gnu-mvapich2-fsp superlu_dist-gnu-openmpi-fsp superlu_dist-intel-impi-fsp superlu_dist-intel-mvapich2-fsp superlu_dist-intel-openmpi-fsp	4.0	A general purpose library for the direct solution of linear equations. http://crd-legacy.lbl.gov/ xiaoye/SuperLU
trilinos-gnu-impi-fsp trilinos-gnu-mvapich2-fsp trilinos-gnu-openmpi-fsp trilinos-intel-impi-fsp trilinos-intel-mvapich2-fsp trilinos-intel-openmpi-fsp	11.14.3	A collection of libraries of numerical algorithms. http://trilinos.sandia.gov/index.html

C 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)
```