



OpenHPC (v3.4) Cluster Building Recipes

Rocky[9.6 Base OS

OpenCHAMI/SLURM Edition for Linux* (x86_64)



Document Last Update: 2025-10-29

Document Revision: 060ee07274f72f2a629d45a50f5d93e38754bbea

Legal Notice

Copyright © 2016-2023, OpenHPC, a Linux Foundation Collaborative Project. All rights reserved.



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

Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Contents

1	Introduction	5
1.1	Target Audience	5
1.2	Requirements/Assumptions	5
1.3	Inputs	6
2	Install Base Operating System (BOS)	7
3	Install OpenCHAMI and Provision Nodes with SMD	8
3.1	OpenCHAMI prerequisites	8
3.2	Enable OpenCHAMI repository for local use	8
3.3	Setting no proxy	8
3.4	Setup DHCP	8
3.5	Complete basic OpenCHAMI setup for <i>master</i> node	9
3.6	Certificate Creation	10
3.7	Starting OpenCHAMI	10
3.8	OpenCHAMI Client Install	11
3.9	Node Discovery	11
4	Install OpenHPC Components	11
4.1	Enable OpenHPC repository for local use	11
4.2	Installation template	12
4.3	Setup time synchronization service on <i>master</i> node	12
4.4	Add resource management services on <i>master</i> node	13
5	OpenCHAMI Image Building	13
5.1	Working Directory	13
5.2	s3cmd Setup	14
5.3	Building the Base OS Image	15
5.3.1	Customize system configuration	18
5.3.1.1	Increase locked memory limits	18
5.3.1.2	Add Lustre client	19
5.3.1.3	Add GPU driver	19
5.3.1.4	Add ClusterShell	20
5.3.1.5	Add <i>genders</i>	20
5.3.1.6	Add Magpie	20
5.3.1.7	Add ConMan	21
5.3.1.8	Add NHC	21
5.3.1.9	Build Production Compute Image	21
5.4	Setting Boot Parameters	22
5.5	Building CloudInit Image	22
6	Munge Key Injection	23
6.1	Boot compute nodes	23
7	Install OpenHPC Development Components	24
7.1	Development Tools	24
7.2	Compilers	24
7.3	MPI Stacks	24
7.4	Performance Tools	25

7.5	Setup default development environment	26
7.6	3rd Party Libraries and Tools	26
7.7	Optional Development Tool Builds	27
8	Resource Manager Startup	28
Appendices		29
A	Installation Template	29
B	Integration Test Suite	30
C	Customization	32
C.1	Adding local Lmod modules to OpenHPC hierarchy	32
C.2	Rebuilding Packages from Source	33
D	Package Signatures	34

1 Introduction

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

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

1.1 Target Audience

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

```
[sms]# echo "OpenHPC hello world"
```

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

Tip

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

1.2 Requirements/Assumptions

This installation recipe assumes the availability of a single head node *master*, and four *compute* nodes. The *master* node serves as the overall system management server (SMS) and is provisioned with Rocky[9.6 and is subsequently configured to provision the remaining *compute* nodes with OpenCHAMI in a stateless configuration. The terms *master* and SMS are used interchangeably in this guide. For power management, we assume that the compute node baseboard management controllers (BMCs) are available via IPMI from the chosen master host. For file systems, we assume that the chosen master server will host an NFS file system that is made available to the compute nodes. Installation information is also discussed to optionally mount a parallel file system and in this case, the parallel file system is assumed to exist previously.

An outline of the physical architecture discussed is shown in Figure 1 and highlights the high-level networking configuration. The *master* host requires at least two Ethernet interfaces with *eth0* connected to the local data center network and *eth1* used to provision and manage the cluster backend (note that these interface names are examples and may be different depending on local settings and OS conventions). Two



Figure 1: Overview of physical cluster architecture.

logical IP interfaces are expected to each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management. The second is used to connect to each host's BMC and is used for power management and remote console access. Physical connectivity for these two logical IP networks is often accommodated via separate cabling and switching infrastructure; however, an alternate configuration can also be accommodated via the use of a shared NIC, which runs a packet filter to divert management packets between the host and BMC. Independent of the actual networking configuration it is recommended to have additional security boundaries like a firewall to protect the network interfaces from the Internet.

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

1.3 Inputs

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

- `${sms_name}` # Hostname for SMS server
- `${sms_ip}` # Internal IP address on SMS server
- `${sms_eth_internal}` # Internal Ethernet interface on SMS
- `${internal_network}` # Subnet network address for internal network
- `${internal_netmask}` # Subnet netmask for internal network
- `${ntp_server}` # Local ntp server for time synchronization
- `${bmc_username}` # BMC username for use by IPMI
- `${bmc_password}` # BMC password for use by IPMI
- `${num_compute}` # Total # of desired compute nodes
- `${c_ip[0]}, ${c_ip[1]}, ...` # Desired compute node addresses
- `${c_bmc[0]}, ${c_bmc[1]}, ...` # BMC addresses for computes
- `${c_mac[0]}, ${c_mac[1]}, ...` # MAC addresses for computes
- `${c_name[0]}, ${c_name[1]}, ...` # Host names for computes
- `${compute_regex}` # Regex matching all compute node names (e.g. "c*")
- `${compute_prefix}` # Prefix for compute node names (e.g. "c")

Optional:

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

2 Install Base Operating System (BOS)

In an external setting, installing the desired BOS on a *master* SMS host typically involves booting from a DVD ISO image on a new server. With this approach, insert the Rocky[9.6 DVD, power cycle the host, and follow the distro provided directions to install the BOS on your chosen *master* host. Alternatively, if choosing to use a pre-installed server, please verify that it is provisioned with the required Rocky[9.6 distribution.

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

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

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

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

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

3 Install OpenCHAMI and Provision Nodes with SMD

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

3.1 OpenCHAMI prerequisites

Add SMS name to `/etc/hosts` so we can make API calls by hostname

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

3.2 Enable OpenCHAMI repository for local use

To begin we need to add the OpenCHAMI RPM to the SMS node, by adding it to the list of available packages locally. This requires network access from the *master* node to the internet.

```
[sms]# dnf -y install podman buildah ansible-core nfs-utils tcpdump tftp curl yq
[sms]# rpm --import https://github.com/OpenCHAMI/release/releases/download/v3LONG/public_gpg_key.asc
[sms]# dnf -y install https://github.com/OpenCHAMI/release/releases/download/v3LONG/openchami-3LONG.rpm
```

3.3 Setting no proxy

OpenCHAMI runs its own internal proxy for services which may not interact well if you run another httpd server. Here we set the no proxy variable to prevent lookups.

```
[sms]# export additional_no_proxy="${sms_name},opaal,opaal-idp,smd,smd-init,hydra,cloud-init,acme-deploy,bss,haproxy,postgres,slurm"
[sms]# if [ -z "$no_proxy" ]; then
[sms]#     export no_proxy="$additional_no_proxy"
[sms]# else
[sms]#     export no_proxy="$no_proxy,$additional_no_proxy"
[sms]# fi
[sms]# if [ -z "$NO_PROXY" ]; then
[sms]#     export NO_PROXY="$additional_no_proxy"
[sms]# else
[sms]#     export NO_PROXY="$NO_PROXY,$additional_no_proxy"
[sms]# fi
```

3.4 Setup DHCP

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

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


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

3.5 Complete basic OpenCHAMI setup for *master* node

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

Tip

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

```
systemctl status registry
systemctl restart minio-server
journalctl -u registry
```

First create local directories for container storage

```
[sms]# mkdir -p /opt/ohpc/admin/data/oci
[sms]# mkdir -p /opt/ohpc/admin/data/s3
```

Now create the container registry quadlet definition

```
[sms]# cat > /etc/containers/systemd/registry.container <<EOF
[sms]# # registry.container
[sms]# [Unit]
[sms]# Description=Image OCI Registry
[sms]# After=network-online.target
[sms]# Requires=network-online.target
[sms]#
[sms]# [Container]
[sms]# ContainerName=registry
[sms]# HostName=registry
[sms]# Image=docker.io/library/registry:latest
[sms]# Volume=/opt/ohpc/admin/data/oci:/var/lib/registry:Z
[sms]# PublishPort=5000:5000

[sms]# [Service]
[sms]# TimeoutStartSec=0
[sms]# Restart=always
[sms]#
[sms]# [Install]
[sms]# WantedBy=multi-user.target
[sms]# EOF
```

Then create the local S3 object store quadlet definition

```
[sms]# cat > /etc/containers/systemd/minio.container <<EOF
[sms]# [Unit]
[sms]# Description=Minio S3
[sms]# After=local-fs.target network-online.target
[sms]# Wants=local-fs.target network-online.target
[sms]#
[sms]# [Container]
[sms]# ContainerName=minio-server
[sms]# Image=docker.io/minio/minio:latest
[sms]# # Volumes
[sms]# Volume=/opt/ohpc/admin/data/s3:/data:Z

[sms]# # Ports
[sms]# PublishPort=9000:9000
[sms]# PublishPort=9091:9001

[sms]# # Environment Variables
[sms]# Environment=MINIO_ROOT_USER=admin
[sms]# Environment=MINIO_ROOT_PASSWORD=password1234
[sms]#
[sms]# # Command to run in container
[sms]# Exec=server /data --console-address :9001
[sms]#
[sms]# [Service]
[sms]# Restart=always

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

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

```
[sms]# systemctl daemon-reload
[sms]# systemctl start minio.service
[sms]# systemctl start registry.service
```

3.6 Certificate Creation

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

```
[sms]# openchami-certificate-update update ${sms_name}
```

3.7 Starting OpenCHAMI

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

```
[sms]# systemctl list-dependencies openchami.target
```

```
[sms]# systemctl start openchami.target
```

3.8 OpenCHAMI Client Install

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

```
[sms]# dnf install -y \
    https://github.com/OpenCHAMI/ochami/releases/download/v0.3.4/ochami_0.3.4_linux_amd64.rpm
[sms]# yes | ochami config cluster set --system --default \
    ${compute_prefix} cluster.uri https://${sms_name}:8443
```

```
[sms]# systemctl restart opaal-idp opaal
[sms]# sed -e "s,FQDN|md5sum,RANDOM|sha512sum,g" -i /usr/libexec/openchami/ohpc-nodes.sh
```

3.9 Node Discovery

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

Create the nodes.yaml file the static discovery will use to populate SMD

```
[sms]# mkdir -p /opt/ohpc/admin/nodes
[sms]# export ${compute_prefix^^}_ACCESS_TOKEN=$(bash -lc 'gen_access_token')
[sms]# echo "nodes:" > /opt/ohpc/admin/nodes/nodes.yaml
[sms]# for((i=0; i < $num_computes; i++)); do
    /usr/libexec/openchami/ohpc-nodes.sh ${c_name[$i]} $i ${c_bmc[$i]} ${c_mac[$i]} ${c_ip[$i]}
done
```

Now use the generated file to populate the SMD database

```
[sms]# ochami discover static -f yaml -d @/opt/ohpc/admin/nodes/nodes.yaml
```

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

We need to enable the OpenHPC repository on the local list of available repositories. We then install the OPHC built applications such as the base and slurm server on the master node.

```
[sms]# dnf -y install http://repos.openhpc.community/OpenHPC/3/EL_9/x86_64/ohpc-release-3-1.el9.x86_64.rpm
[sms]# dnf -y install dnf-plugins-core
[sms]# dnf config-manager --set-enabled crb
```

In addition to the OpenHPC package repository, the *master* host also requires access to the standard base OS distro repositories in order to resolve necessary dependencies. For Rocky[9.6, the requirements are to have access to the BaseOS, Appstream, Extras, CRB, and EPEL repositories for which mirrors are freely available online:

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

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

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

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

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

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

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

4.2 Installation template

The collection of command-line instructions that follow in this guide, when combined with local site inputs, can be used to implement a bare-metal system installation and configuration. The format of these commands is intended to be usable via direct cut and paste (with variable substitution for site-specific settings). Alternatively, the OpenHPC documentation package (`docs-ohpc`) includes a template script which includes a summary of all of the commands used herein. This script can be used in conjunction with a simple text file to define the local site variables defined in the previous section (§ 1.3) and is provided as a convenience for administrators. For additional information on accessing this script, please see Appendix A.

4.3 Setup time synchronization service on *master* node

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

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

Tip

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

```
allow 192.168.0.0/16
```

4.4 Add resource management services on *master* node

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

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

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

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

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

Tip

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

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

5 OpenCHAMI Image Building

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

5.1 Working Directory

```
[sms]# mkdir -p /opt/ohpc/admin/images/data
[sms]# cd /opt/ohpc/admin/images
```

```
# Register Slurm server with computes (using "configless" option)
[sms]# echo SLURMD_OPTIONS="--conf-server ${sms_ip}" > /opt/ohpc/admin/images/data/slurmd
# Handle SSH to compute nodes
[sms]# echo -e "Host ${compute_prefix}*\n\tStrictHostKeyChecking=no" >> /etc/ssh/ssh_config.d/40-ohpc.conf
```

5.2 s3cmd Setup

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

First lets install a CLI tool we can use to interact with the Minio service

```
[sms]# dnf -y install s3cmd
```

Now we'll create a config file (maybe use a better passwd)

```
[sms]# cat > ~/.s3cfg <<EOF
[sms]# # Setup endpoint
[sms]# host_base = ${sms_name}:9000
[sms]# host_bucket = ${sms_name}:9000
[sms]# bucket_location = us-east-1
[sms]# use_https = False
[sms]#
[sms]# # Setup access keys
[sms]# access_key = admin
[sms]# secret_key = password1234
[sms]#
[sms]# # Enable S3 v4 signature APIs
[sms]# signature_v2 = False
[sms]# EOF
```

Now we can use the tool to create the buckets we'll need and set ACLs to public

```
[sms]# s3cmd mb s3://efi
[sms]# s3cmd setacl s3://efi --acl-public
[sms]# s3cmd mb s3://boot-images
[sms]# s3cmd setacl s3://boot-images --acl-public
```

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

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

```
[sms]# {
[sms]#   "Effect":"Allow",
[sms]#   "Principal": "*",
[sms]#   "Action":["s3:GetObject"],
[sms]#   "Resource":["arn:aws:s3:::efi/*"]
[sms]# }
[sms]# ]
[sms]# }
[sms]# EOF
```

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

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

5.3 Building the Base OS Image

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

Tip

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

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

```
[sms]# echo -e "%_excludedocs 1" >> /opt/ohpc/admin/images/data/rpmmacros
[sms]# cat > /opt/ohpc/admin/images/nodocs.yaml <<EOF
[sms]# options:
[sms]#   layer_type: 'base'
[sms]#   name: 'rocky-nodocs'
[sms]#   publish_tags: '9'
[sms]#   pkg_manager: 'dnf'
[sms]#   parent: 'scratch'
[sms]#   publish_registry: 'ohpc-lenovo-sms:5000/c'
[sms]#   registry_opts_push:
[sms]#     - '--tls-verify=false'
[sms]#   copyfiles:
[sms]#     - src: '/data/rpmmacros'
[sms]#       dest: '/root/.rpmmacros'
[sms]# EOF
```

Build this first layer:

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

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

```
[sms]# cat > /opt/ohpc/admin/images/base.yaml <<EOF
[sms]# options:
[sms]#   layer_type: 'base'
[sms]#   name: 'rocky-base'
[sms]#   publish_tags: '9'
[sms]#   pkg_manager: 'dnf'
[sms]#   parent: '${sms_name}:5000/${compute_prefix}/rocky-nodocs:9'
[sms]#   publish_registry: '${sms_name}:5000/${compute_prefix}'
[sms]#   registry_opts_pull:
[sms]#     - '--tls-verify=false'
[sms]#   registry_opts_push:
[sms]#     - '--tls-verify=false'
[sms]#   repos:
[sms]#     - alias: 'Rocky_9_BaseOS'
[sms]#       url: 'https://dl.rockylinux.org/pub/rocky/9/BaseOS/x86_64/os/'
[sms]#       gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-9'
[sms]#     - alias: 'Rocky_9_AppStream'
[sms]#       url: 'https://dl.rockylinux.org/pub/rocky/9/AppStream/x86_64/os/'
[sms]#       gpg: 'https://dl.rockylinux.org/pub/rocky/RPM-GPG-KEY-Rocky-9'
[sms]#   package_groups:
[sms]#     - 'Minimal Install'
[sms]#     - 'Development Tools'
[sms]#   packages:
[sms]#     - kernel
[sms]#     - wget
[sms]#     - dracut-live
[sms]#     - cloud-init
[sms]#     - chrony
[sms]#     - rsyslog
[sms]#     - sudo
[sms]#   cmds:
[sms]#     - cmd: >
[sms]#       dracut --add "dmsquash-live livenet network-manager"
[sms]#       --kver "\$(basename /lib/modules/*)"
[sms]#       -N -f --logfile /tmp/dracut.log 2>/dev/null
[sms]#     loglevel: INFO
[sms]#     - cmd: 'echo DRACUT LOG:; cat /tmp/dracut.log'
[sms]#     loglevel: INFO
[sms]# EOF
```

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

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

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

```
[sms]# cat > /opt/ohpc/admin/images/compute-base.yaml << EOF
[sms]# options:
```



```
[sms]# layer_type: 'base'
[sms]# name: 'compute-base'
[sms]# publish_tags:
[sms]#   - '9'
[sms]# pkg_manager: 'dnf'
[sms]# parent: '${sms_name}:5000/${compute_prefix}/rocky-base:9'
[sms]# publish_registry: '${sms_name}:5000/${compute_prefix}'
[sms]# registry_opts_pull:
[sms]#   - '--tls-verify=false'
[sms]# registry_opts_push:
[sms]#   - '--tls-verify=false'
[sms]# repos:
[sms]#   - alias: 'Epel9'
[sms]#     url: 'https://dl.fedoraproject.org/pub/epel/9/Everything/x86_64/'
[sms]#     gpg: 'https://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-9'
[sms]# package_groups:
[sms]#   - 'InfiniBand Support'
[sms]# packages:
[sms]#   - vim
[sms]#   - nfs-utils
[sms]#   - tcpdump
[sms]#   - traceroute
[sms]#   - git
[sms]#   - http://repos.openhpc.community/OpenHPC/3/EL_9/x86_64/ohpc-release-3-1.el9.x86_64.rpm
[sms]# cmds:
[sms]#   - cmd: /usr/bin/crb enable
[sms]#   loglevel: INFO
[sms]# EOF
```

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

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

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

```
[sms]# cp /etc/hosts /opt/ohpc/admin/images/data
[sms]# cat > /opt/ohpc/admin/images/compute-prod.yaml << EOF
[sms]# options:
[sms]#   layer_type: 'base'
[sms]#   name: 'compute-prod'
[sms]#   publish_tags:
[sms]#     - '9'
[sms]#   pkg_manager: 'dnf'
[sms]#   parent: '${sms_name}:5000/${compute_prefix}/compute-base:9'
[sms]#   publish_registry: '${sms_name}:5000/${compute_prefix}'
[sms]#   registry_opts_pull:
[sms]#     - '--tls-verify=false'
[sms]#   registry_opts_push:
[sms]#     - '--tls-verify=false'
[sms]#   # Publish SquashFS image to local S3
[sms]#   publish_s3: 'http://${sms_name}:9000'
[sms]#   s3_prefix: 'compute/base/'
```

```
[sms]# s3_bucket: 'boot-images'
[sms]#
[sms]# # Publish OCI image to container registry
[sms]# #
[sms]# # This is the only way to be able to reuse this image as
[sms]# # a parent for another image layer.
[sms]# publish_registry: '${sms_name}:5000/${compute_prefix}'
[sms]# registry_opts_push:
[sms]#   - '--tls-verify=false'
[sms]# copyfiles:
[sms]#   - src: '/data/hosts'
[sms]#     dest: '/etc/hosts'
[sms]#   - src: '/data/slurmd'
[sms]#     dest: '/etc/sysconfig/slurmd'
[sms]# packages:
[sms]#   - ohpc-base-compute
[sms]#   - ohpc-slurm-client
[sms]#   - pdsh-ohpc
[sms]#   - lmod-ohpc
[sms]# cmds:
[sms]#   - cmd: systemctl disable firewalld
[sms]#     loglevel: INFO
[sms]#   - cmd: echo '${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0' >> /etc/fstab
[sms]#     loglevel: INFO
[sms]#   - cmd: echo '${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev,nosuid 0 0' >> /etc/fstab
[sms]#     loglevel: INFO
[sms]#   - cmd: mkdir -p /opt/ohpc/pub
[sms]#     loglevel: INFO
[sms]#   - cmd: echo "account required pam_slurm.so" >> /etc/pam.d/sshd
[sms]#     loglevel: INFO
[sms]#   - cmd: systemctl enable munge slurmd
[sms]#     loglevel: INFO
[sms]#   - cmd: ln -sf ../usr/share/zoneinfo/UTC /etc/localtime
[sms]#     loglevel: INFO
[sms]# EOF
```

5.3.1 Customize system configuration

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

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

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

```
# Update memlock settings on SMS
[sms]# echo '* soft memlock unlimited' >> /etc/security/limits.d/40-ohpc-limits.conf
```

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

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

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

```
# Include Lustre client software in compute image
[sms]# yq -i '.packages += ["lustre-client-ohpc"]' /opt/ohpc/admin/images/compute-prod.yaml
```

Tip

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

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

```
[sms]# echo "options lnet networks=o2ib(ib0)" >> /etc/modprobe.d/lustre.conf
[sms]# yq -i '.cmds += [{"cmd":"echo \"options lnet networks=o2ib(ib0)\" >> /etc/modprobe.d/lustre.conf"}]' /opt/ohpc/admin/images/compute-prod.yaml
```

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
```

5.3.1.3 Add GPU driver If planning to install the NVIDIA toolkit, register the following additional path (`/opt/nvidia`) to share with the compute nodes:

```
# Add NVIDIA GPU driver repository to the SMS
[sms]# dnf -y install cuda-repo-ohpc
# Add NVIDIA GPU driver repository to the compute nodes
[sms]# yq -i '.repos += [{"alias": "cuda-rhel9-x86_64", "url": "https://developer.download.nvidia.com/compute/cuda/repos/rhel9/x86_64"}]' /etc/yum.repos.d/cuda-rhel9-x86_64.repo
# Install the GPU driver on the compute nodes
[sms]# yq -i '.modules += [{"install": ["nvidia-driver:latest-dkms"]}]' /opt/ohpc/admin/images/compute-prod.yaml
# Enable DKMS service to automatically rebuild driver
[sms]# yq -i '.cmds += [{"cmd": "systemctl enable dkms"}]' /opt/ohpc/admin/images/compute-prod.yaml
# Install the toolkit on the SMS
[sms]# dnf -y install cuda-devel-ohpc nvidia-driver-cuda
# (Optional) Setup NFS mount for /opt/nvidia
[sms]# echo "/opt/nvidia *(ro,no_subtree_check)" >> /etc/exports
[sms]# exportfs -a

# Create mount point and file system mount
[sms]# nodeshell compute mkdir /opt/nvidia
[sms]# nodeshell compute echo \
    "\"${sms_ip}:/opt/nvidia /opt/nvidia nfs nfsvers=3,nodev,nosuid 0 0\" \">> /etc/fstab
# Mount NFS shares
[sms]# nodeshell compute mount /opt/nvidia
```

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

```
# Install ClusterShell
[sms]# dnf -y install clustershell

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

5.3.1.5 Add genders *genders* is a static cluster configuration database or node typing database used for cluster configuration management. Other tools and users can access the *genders* database in order to make decisions about where an action, or even what action, is appropriate based on associated types or "genders". Values may also be assigned to and retrieved from a *gender* to provide further granularity. The following example highlights installation and configuration of two genders: *compute* and *bmc*.

```
# Install genders
[sms]# dnf -y install genders-ohpc

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

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

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

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

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

# Configure conman for computes (note your IPMI password is required for console access)
[sms]# for ((i=0; i<$num_compute; i++)) ; do
    echo -n 'CONSOLE name="'${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
```

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

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

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

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

5.3.1.9 Build Production Compute Image Rebuild Final Compute node image with all optional packages

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

```
image-build \
--config config.yaml
```

5.4 Setting Boot Parameters

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

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

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

Create a yaml payload for BSS

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

And then finally upload our payload

```
[sms]# ochami bss boot params set -f yaml -d @/opt/ohpc/admin/nodes/compute-boot.yaml
```

5.5 Building CloudInit Image

OpenCHAMI uses cloudinit for post boot node configuration. We will establish some basic cloudinit configurations here. But later we will build a cloudinit template that injects the munge key into the system at boot.

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

```
[sms]# mkdir -p /opt/ohpc/admin/cloud-init
[sms]# cd /opt/ohpc/admin/cloud-init
[sms]# ssh-keygen -t ed25519 -q -f "$HOME/.ssh/id_openchami" -N ""
[sms]# export master_key=$(cat ~/.ssh/id_openchami.pub)
```

Now we can define the payload to update the defaults

```
[sms]# cat > /opt/ohpc/admin/cloud-init/defaults.yaml <<EOF
[sms]# ---
[sms]# base-url: "http://${sms_ip}:8081/cloud-init"
[sms]# cluster-name: "${compute_prefix}^"
[sms]# nid-length: 1
```

```
[sms]# public-keys:
[sms]# - "${master_key}"
[sms]# short-name: "${compute_prefix}"
[sms]# EOF
[sms]# for i in /root/.ssh/id*pub; do \
    file=${i} yq '.public-keys +=loadstr(strenv(file))' -i /opt/ohpc/admin/cloud-init/defaults.yaml; \
done
```

And then finally upload our payload

```
[sms]# ochami cloud-init defaults set -f yaml -d @/opt/ohpc/admin/cloud-init/defaults.yaml
```

6 Munge Key Injection

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

```
[sms]# export munge_key=$(cat /etc/munge/munge.key | base64 -w 0)
[sms]# cat > /opt/ohpc/admin/cloud-init/compute.yaml <<EOF
[sms]# - name: compute
[sms]#   description: "compute template"
[sms]#   file:
[sms]#     encoding: plain
[sms]#     content: |
[sms]#       ## template: jinja
[sms]#       #cloud-config
[sms]#       merge_how:
[sms]#         - name: list
[sms]#           settings: [append]
[sms]#         - name: dict
[sms]#           settings: [no_replace, recurse_list]
[sms]#       users:
[sms]#         - name: root
[sms]#           ssh_authorized_keys: {{ ds.meta_data.instance_data.v1.public_keys }}
[sms]#       write_files:
[sms]#         - content: ${munge_key}
[sms]#           path: /etc/munge/munge.key
[sms]#           permissions: '0400'
[sms]#           owner: munge:munge
[sms]#           encoding: base64
[sms]# EOF
```

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

```
[sms]# ochami cloud-init group set -f yaml -d @/opt/ohpc/admin/cloud-init/compute.yaml
```

6.1 Boot compute nodes

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

requires that the `IPMI_PASSWORD` environment variable be set with the local BMC password in order to work interactively.

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

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

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

7 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.

7.1 Development Tools

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

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

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

7.2 Compilers

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

```
[sms]# dnf -y install gnu15-compilers-ohpc
```

7.3 MPI Stacks

For MPI development and runtime support, OpenHPC provides pre-packaged builds for a variety of MPI families and transport layers. Currently available options and their applicability to various network trans-

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

Table 1: Available MPI variants

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

```
[sms]# dnf -y install openmpi5-pmix-gnu15-ohpc mpich-ofi-gnu15-ohpc
```

Note that OpenHPC 2.x introduces the use of two related transport layers for the MPICH and OpenMPI builds that support a variety of underlying fabrics: [UCX](#) (Unified Communication X) and [OFI](#) (OpenFabrics interfaces). In the case of OpenMPI, a monolithic build is provided which supports both transports and end-users can customize their runtime preferences with environment variables. For MPICH, two separate builds are provided and the example above highlighted installing the ofi variant. However, the packaging is designed such that both versions can be installed simultaneously and users can switch between the two via normal module command semantics. Alternatively, a site can choose to install the ucx variant instead as a drop-in MPICH replacement:

```
[sms]# dnf -y install mpich-ucx-gnu15-ohpc
```

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

```
[sms]# module avail mpich
----- /opt/ohpc/pub/moduledeps/gnu15-----
mpich/3.4.3-ofi  mpich/3.4.3-ucx (D)
```

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

```
[sms]# dnf -y install mvapich2-gnu15-ohpc
```

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

```
[sms]# dnf -y install mvapich2-psm2-gnu15-ohpc
```

7.4 Performance Tools

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

```
# Install perf-tools meta-package
[sms]# dnf -y install ohpc-gnu15-perf-tools
```

7.5 Setup default development environment

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

```
[sms]# dnf -y install lmod-defaults-gnu15-openmpi5-ohpc
```

Tip

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

- `lmod-defaults-gnu15-mpich-ofi-ohpc`
- `lmod-defaults-gnu15-mpich-ucx-ohpc`
- `lmod-defaults-gnu15-mvapich2-ohpc`

7.6 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 §7.3.

Tip

OpenHPC-provided 3rd party builds are configured to be installed into a common top-level repository so that they can be easily exported to desired hosts within the cluster. This common top-level path (`/opt/ohpc/pub`) was previously configured to be mounted on *compute* nodes in §5.3.1, 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 ?? for a more detailed listing of all available packages in each of these functional areas. To install all available package offerings within OpenHPC, issue the following:

```
# Install 3rd party libraries/tools meta-packages built with GNU toolchain
[sms]# dnf -y install ohpc-gnu15-serial-libs
[sms]# dnf -y install ohpc-gnu15-io-libs
[sms]# dnf -y install ohpc-gnu15-python-libs
[sms]# dnf -y install ohpc-gnu15-runtimes
```

```
# Install parallel lib meta-packages for all available MPI toolchains
[sms]# dnf -y install ohpc-gnu15-mpich-parallel-libs
[sms]# dnf -y install ohpc-gnu15-openmpi5-parallel-libs
```

7.7 Optional Development Tool Builds

In addition to the 3rd party development libraries built using the open source toolchains mentioned in §7.6, OpenHPC also provides *optional* compatible builds for use with the compilers and MPI stack included in newer versions of the Intel® oneAPI HPC Toolkit (using the *classic* compiler variants). These packages provide a similar hierarchical user environment experience as other compiler and MPI families present in OpenHPC.

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

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

Tip

As noted in §5.3.1, the default installation path for OpenHPC (`/opt/ohpc/pub`) is exported over NFS from the *master* to the compute nodes, but the Intel® oneAPI HPC Toolkit packages install to a top-level path of `/opt/intel`. To make the Intel® compilers available to the compute nodes one must add an additional NFS export for `/opt/intel` that is mounted on desired compute nodes.

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

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

```
# Install 3rd party libraries/tools meta-packages built with Intel toolchain
[sms]# dnf -y install openmpi5-pmix-intel-ohpc
[sms]# dnf -y install ohpc-intel-serial-libs
```

```
[sms]# dnf -y install ohpc-intel-geopm
[sms]# dnf -y install ohpc-intel-io-libs
[sms]# dnf -y install ohpc-intel-perf-tools
[sms]# dnf -y install ohpc-intel-python3-libs
[sms]# dnf -y install ohpc-intel-mpich-parallel-libs
[sms]# dnf -y install ohpc-intel-mvapich2-parallel-libs
[sms]# dnf -y install ohpc-intel-openmpi5-parallel-libs
[sms]# dnf -y install ohpc-intel-impi-parallel-libs
```

8 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 ${compute_prefix}[1-${num_computes}] systemctl start munge
[sms]# pdsh -w ${compute_prefix}[1-${num_computes}] systemctl start slurmd
```

Appendices

A Installation Template

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

Tip

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

The template script relies on the use of a simple text file to define local site variables that were outlined in §1.3. By default, the template installation script attempts to use local variable settings sourced from the `/opt/ohpc/pub/doc/recipes/vanilla/input.local` file, however, this choice can be overridden by the use of the `${OHPC_INPUT_LOCAL}` environment variable. The template install script is intended for execution on the SMS *master* host and is installed as part of the `docs-ohpc` package into `/opt/ohpc/pub/doc/recipes/vanilla/recipe.sh`. After enabling the OpenHPC repository and reviewing the guide for additional information on the intent of the commands, the general starting approach for using this template is as follows:

1. Install the `docs-ohpc` package

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

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

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

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

```
[sms]# cp -p /opt/ohpc/pub/doc/recipes/rocky9/x86_64/openchami/slurm/recipe.sh .
```

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

```
[sms]# export OHPC_INPUT_LOCAL=./input.local  
[sms]# ./recipe.sh
```

B Integration Test Suite

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

```
[sms]# dnf -y install test-suite-ohpc
```

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

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

```
[sms]# su - ohpc-test
[test@sms ~]$ cd tests
[test@sms ~]$ ./configure --disable-all --enable-fftw
checking for a BSD-compatible install... /bin/install -c
checking whether build environment is sane... yes
...
----- SUMMARY -----
Package version..... : test-suite-2.0.0

Build user..... : ohpc-test
Build host..... : sms001
Configure date..... : 2020-10-05 08:22
Build architecture..... : x86_64
Compiler Families..... : gnu9
MPI Families..... : mpich mvapich2 openmpi4
Python Families..... : python3
Resource manager ..... : SLURM
Test suite configuration..... : short
...
Libraries:
  Adios ..... : disabled
  Boost ..... : disabled
  Boost MPI..... : disabled
  FFTW..... : enabled
  GSL..... : disabled
  HDF5..... : disabled
  HYPRE..... : disabled
...
```

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

```
[test@sms ~]$ make check
make --no-print-directory check-TESTS
PASS: libs/fftw/ohpc-tests/test_mpi_families
=====
Testsuite summary for test-suite 2.0.0
=====
# TOTAL: 1
# PASS: 1
# SKIP: 0
# XFAIL: 0
# FAIL: 0
# XPASS: 0
# ERROR: 0
=====
[test@sms ~]$ cat libs/fftw/tests/family-gnu*/rm_execution.log
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (SLURM/gnu9/mpich)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/mpich)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/mpich)
PASS rm_execution (exit status: 0)
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (SLURM/gnu9/mvapich2)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/mvapich2)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/mvapich2)
PASS rm_execution (exit status: 0)
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (SLURM/gnu9/openmpi4)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (SLURM/gnu9/openmpi4)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (SLURM/gnu9/openmpi4)
PASS rm_execution (exit status: 0)
```

C Customization

C.1 Adding local Lmod modules to OpenHPC hierarchy

Locally installed applications can easily be integrated in to OpenHPC systems by following the Lmod convention laid out by the provided packages. Two sample module files are included in the `examples-ohpc` package—one representing an application with no compiler or MPI runtime dependencies, and one dependent on OpenMPI and the GNU toolchain. Simply copy these files to the prescribed locations, and the `lmod` application should pick them up automatically.

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

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

----- /opt/ohpc/pub/moduledeps/gnu7 -----
R/3.4.2      metis/5.1.0     ocr/1.0.1            pdttoolkit/3.24  superlu/5.2.1
gsl/2.4      mpich/3.2       openblas/0.2.20      plasma/2.8.0
hdf5/1.10.1  numpy/1.13.1    openmpi3/3.0.0 (L)  scotch/6.0.4

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

----- /opt/ohpc/pub/modulefiles -----
EasyBuild/3.4.1  cmake/3.9.2  hwloc/1.11.8  pmix/1.2.3  valgrind/3.13.0
autotools (L)    example1/1.0 (L)  llvm5/5.0.0  prun/1.2 (L)
clustershell/1.8  gnu7/7.2.0 (L)  ohpc (L)    singularity/2.4

Where:
L: Module is loaded

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


C.2 Rebuilding Packages from Source

Users of OpenHPC may find it desirable to rebuild one of the supplied packages to apply build customizations or satisfy local requirements. One way to accomplish this is to install the appropriate source RPM, modify the spec file as needed, and rebuild to obtain an updated binary RPM. OpenHPC spec files contain macros to facilitate local customizations of compiler, compilation flags and MPI family. A brief example using the FFTW library is highlighted below. Note that the source RPMs can be downloaded from the community repository server at <http://repos.openhpc.community> via a web browser or directly via `dnf` as highlighted below. In this example we make an explicit change to FFTW's configuration, as well as modifying the `CFLAGS` environment variable. The package is also tagged with an additional delimiter to allow easy co-installation and use.

```
# Install rpm-build package and dnf tools from base OS distro
[test@sms ~]$ sudo dnf -y install rpm-build dnf-plugins-core

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

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

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

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

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

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

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

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

../.. /warewulf/slurm/manifest.tex

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: 5392 744D 3C54 3ED5 7847 65E6 8A30 6019 DA565C6C

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

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