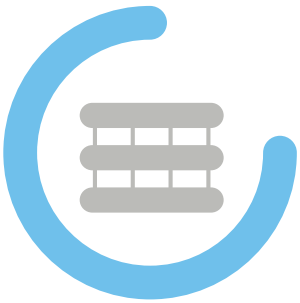


OpenHPC (v4.2)

Cluster Building Recipes

Rocky 10 Base OS

Warewulf/Flux Edition for Linux* (aarch64)



Document Last Update: 2026-06-05

Document Revision: d6b49a4be65ae379e2870143be2c2aae52aed5cd

Legal Notice

Copyright © 2016-2026, 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

Legal Notice 2

1 Introduction 6

1.1 Target Audience . . . 6
1.2 Requirements/Assumptions . . . 6
1.3 Inputs . . . 7
1.4 Installation Template . . . 8

2 Install Base Operating System 8

2.1 Install OS . . . 8
2.2 Configure HTTPS Proxy (Optional) . . . 8
2.3 Enable EL Package Repositories . . . 9
2.4 Configure Firewall . . . 9
2.5 Configure NTP . . . 9
2.6 Optionally add InfiniBand support services on *head node* . . . 10
2.7 Optionally add RoCE support on *head node* . . . 10
2.8 Optionally add Omni-Path support services on *head node* . . . 11

3 Install OpenHPC Components 11

3.1 Enable OpenHPC Repository . . . 11
3.2 Install OpenHPC Base Packages . . . 12

4 Install Flux 12

4.1 Add Flux on Head Node . . . 12

5 Install Warewulf Provisioner 14

5.1 Install Warewulf on *Head Node* . . . 14
5.2 Complete basic Warewulf setup for *head node* . . . 14
5.3 Create Compute Node Image . . . 17
5.4 Add OpenHPC Components . . . 17
5.5 Import files . . . 18
5.6 Configure Flux . . . 19
5.7 Finalizing Provisioning Configuration . . . 20
5.7.1 Two-Stage Provisioning (optional) . . . 20
5.7.2 Build Image and Overlays . . . 21

6 Additional Customization **22**
6.1 Enable InfiniBand drivers . . . 22
6.2 Enable RoCE drivers . . . 22
6.3 Enable Omni-Path drivers . . . 22
6.4 Increase locked memory limits . . . 22
6.5 NVIDIA GPU Drivers . . . 23
6.6 Enable System Log Forwarding . . . 24
6.7 Cluster Admin Tools . . . 24
6.7.1 Add ClusterShell . . . 24
6.7.2 Add genders . . . 25
6.7.3 Add ConMan . . . 25
6.7.4 Add NHC . . . 25
6.8 Add Magpie . . . 26
6.9 Charliecloud . . . 26

7 Deploy Cluster **26**
7.1 Register Compute Nodes for Provisioning . . . 26
7.2 Optionally configure IPoIB on *compute nodes* . . . 26
7.3 Rebuild the Image and Overlays . . . 27
7.4 Start Flux . . . 27
7.5 Boot Compute Nodes . . . 27
7.6 Wait for Compute Nodes . . . 28

8 Install OpenHPC Development Components **28**
8.1 Development Tools . . . 28
8.2 Compilers . . . 29
8.3 MPI Stacks . . . 29
8.4 Performance Tools . . . 29
8.5 Setup default development environment . . . 30
8.6 3rd Party Libraries and Tools . . . 30

9 Post Provisioning **31**
9.1 Check Cluster Status . . . 31
9.2 Add User . . . 31
9.3 Update NHC . . . 32

10 Test Cluster **32**
10.1 Run a Test Job . . . 32
10.2 Interactive Execution . . . 32
10.3 Batch execution . . . 33

Appendices **35**

A Installation Template **35**

B Upgrading OpenHPC Packages **36**

C Integration Test Suite **37**

D Customization **40**
D.1 Adding local environment modules to OpenHPC hierarchy . . . 40
D.2 Rebuilding Packages from Source . . . 41

E Package Manifest **42**
E.1 Available OpenHPC Meta-packages . . . 42
E.2 RPM Packages . . . 43
E.2.1 Administrative Tools . . . 44
E.2.2 Provisioning . . . 45
E.2.3 Resource Management . . . 45
E.2.4 Compiler Families . . . 46
E.2.5 MPI Families / Communication Libraries . . . 46
E.2.6 Development Tools . . . 46
E.2.7 Performance Analysis Tools . . . 47
E.2.8 IO Libraries . . . 47
E.2.9 Distro Packages . . . 48
E.2.10 Runtimes . . . 48
E.2.11 Serial/Threaded Libraries . . . 48
E.2.12 Parallel Libraries . . . 49

F Package Signatures **50**

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 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:

```
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*, and four *compute* nodes. The *head node* serves as the overall system management server (SMS) and is provisioned with Rocky 10 and is subsequently configured to provision the remaining *compute* nodes with Warewulf in a stateless configuration. For power management, we assume that the compute node baseboard management controllers (BMCs) are available via IPMI from the chosen *head node*. For file systems, we assume that the chosen *head node* will host an NFS file system that is made available to the compute nodes.

An outline of the physical architecture discussed is shown in the figure above and highlights the high-level networking configuration. The *head node* requires at least two Ethernet interfaces with *eth0* connected to the local data center network and *eth1* used to provision and manage the cluster backend (note that these interface names are examples and may be different depending on local settings and OS conventions). Two logical IP interfaces are expected to each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management. The second is used to connect to each host's BMC and is used for power management and remote console access. Physical connectivity for these two logical

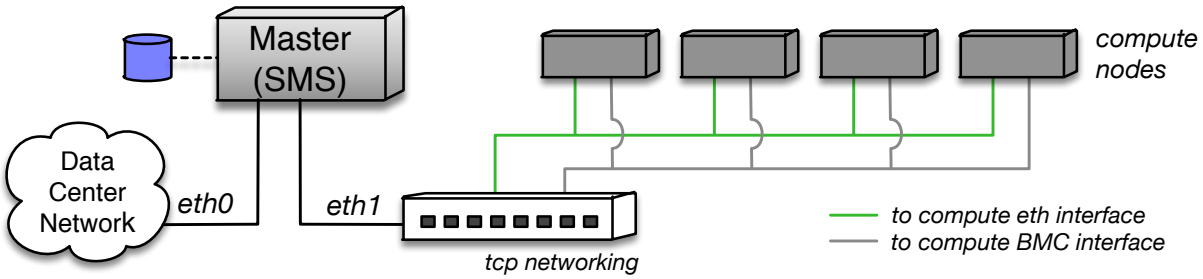


Figure 1: Overview of physical cluster architecture

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.

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.

Required variables:

- `${sms_name}` - Hostname for head node
- `${sms_ip}` - Internal IP address on head node
- `${sms_eth_internal}` - Internal Ethernet interface on head node
- `${eth_provision}` - Provisioning interface for computes
- `${internal_network}` - Subnet network address for internal network
- `${internal_netmask}` - Subnet netmask for internal network
- `${ipv4_gateway}` - Default gateway for the internal network
- `${dns_servers}` - DNS resolver for the internal network
- `${ntp_server}` - Local ntp server for time synchronization
- `${bmc_username}` - BMC username for use by IPMI
- `${bmc_password}` - BMC password for use by IPMI
- `${num_computes}` - Total # of desired compute nodes
- `${c_ip[0]}`, `${c_ip[1]}`, … - Desired compute node addresses

- `${c_bmc[0]}`, `${c_bmc[1]}`, ... - BMC addresses for computes
- `${c_mac[0]}`, `${c_mac[1]}`, ... - MAC addresses for computes
- `${c_name[0]}`, `${c_name[1]}`, ... - Host names for computes
- `${compute_regex}` - Regex matching all compute node names (e.g. `c*`)
- `${compute_prefix}` - Prefix for compute node names (e.g. `c`)
- `${node_disk}` - Device path for disk to be used for provision-to-disk

Optional variables:

- `${modules_package}` - Module tool installed on the head node and the computes: `lmod-ohpc` (Lmod, the default) or `environment-modules-ohpc` (Environment Modules)
- `${roce_mtu}` - Ethernet MTU for RoCE on the internal network (empty leaves it unchanged; 4200 or more gives RoCE MTU 4096)

1.4 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 (see Requirements/Assumptions) and is provided as a convenience for administrators. For additional information on accessing this script, please see the Automation Appendix.

2 Install Base Operating System

2.1 Install OS

In an external setting, installing the desired Base OS on a *head node* typically involves booting from a DVD ISO image on a new server. With this approach, insert the Rocky 10 DVD, power cycle the host, and follow the distro provided directions to install the Base OS on your chosen *head node*. Alternatively, if choosing to use a pre-installed server, please verify that it is provisioned with the required Rocky 10 distribution.

While it is theoretically possible to enable SELinux on a cluster provisioned with Warewulf, doing so is beyond the scope of this document. Even the use of permissive mode can be problematic and we therefore recommend disabling SELinux on the *head node*. 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.

2.2 Configure HTTPS Proxy (Optional)

If your environment requires an HTTPS caching proxy for external network access, configure it here before installing any packages.

2.3 Enable EL Package Repositories

In addition to the OpenHPC package repository, the *head node* also requires access to the standard base OS distro repositories in order to resolve necessary dependencies. For Rocky 10, the requirements are to have access to the BaseOS, Appstream, Extras, CRB, and EPEL repositories for which mirrors are freely available online:

- Rocky-10 (e.g. http://download.rockylinux.org/pub/rocky/10/)
- EPEL 10 (e.g. http://download.fedoraproject.org/pub/epel/10/)

Although the public EPEL repository would be enabled automatically upon installation of the `ohpc-release` package, we install it now. Note that this does depend on the Rocky 10 Extras repository, which is shipped with Rocky 10 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:

```
dnf -y install epel-release dnf-plugins-core
dnf config-manager --set-enabled crb
```

2.4 Configure Firewall

Provisioning services rely on DHCP, TFTP, and HTTP network protocols. Depending on the local Base OS configuration on the head node, default firewall rules may prohibit these services. Consequently, this recipe assumes that the local firewall running on the head node 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:

```
systemctl disable --now firewalld.service || true
```

2.5 Configure NTP

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

```
dnf -y install chrony
systemctl enable chronyd.service
echo "local stratum 10" >> /etc/chrony.conf
echo "server ${ntp_server}" >> /etc/chrony.conf
echo "allow all" >> /etc/chrony.conf
systemctl restart chronyd.service
```

Tip

Note: 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 head node. 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
```

2.6 Optionally add InfiniBand support services on *head node*

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

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

Tip

Note: InfiniBand networks require a subnet management service that can typically be run on either an administrative node, or on the switch itself. The optimal placement and configuration of the subnet manager is beyond the scope of this document, but Rocky 10 provides the `opensm` package should you choose to run it on the *head node*.

With the InfiniBand drivers included, you can also enable (optional) IPoIB functionality which provides a mechanism to send IP packets over the IB network. If you plan to mount a Lustre file system over InfiniBand, then having IPoIB enabled is a requirement for the Lustre client. The following creates a NetworkManager connection for the IPoIB interface `${ipoib_dev}` on the *head node*, with the address `${sms_ipoib}` and subnet prefix `${ipoib_prefix_length}`. Set `${ipoib_dev}` to the name the system gives the interface, for example `ibp3s0` under predictable interface naming, and add one connection per port if the system contains dual-ported or multiple HCAs.

```
# Define the local IPoIB interface, address, and subnet prefix
nmcli connection add type infiniband con-name ipoib ifname "${ipoib_dev}" \
    ipv4.method manual ipv4.addresses "${sms_ipoib}/${ipoib_prefix_length}"

# Bring up the IPoIB interface
nmcli connection up ipoib
```

2.7 Optionally add RoCE support on *head node*

RoCE (RDMA over Converged Ethernet) runs RDMA over Ethernet NICs that support it, for example NVIDIA ConnectX "Dx" models. It uses the same distro-provided RDMA stack as InfiniBand, but it needs no subnet manager and no IPoIB: RDMA traffic is addressed by the Ethernet interface's own IP address. This recipe runs RoCE over the internal cluster network. The following command adds the RDMA stack to the *head node*. If you enabled InfiniBand above, the stack is already installed and the script skips this step.

```
dnf -y groupinstall "InfiniBand Support"
# Load the RDMA drivers for the RoCE NIC
udevadm trigger --type=devices --action=add
systemctl restart rdma-load-modules@rdma.service
```

RoCE takes its own MTU (up to 4096 bytes) from the Ethernet MTU: the largest that fits with room for headers. An Ethernet MTU of 1500 gives RoCE 1024, and exactly 4096 gives only 2048; about 4200 or more (9000 is common) gives the full 4096. RoCE works at the default MTU, and a larger one adds a little bandwidth. Every host and switch port on the network must carry the larger frames.

To use one, set `${roce_mtu}`. The following sets it on the *head node*'s internal interface, now and across reboots; Enable RoCE drivers sets the same value on the computes.

```
# Set the Ethernet MTU on the internal interface
nmcli connection modify \
    "$(nmcli -g GENERAL.CONNECTION device show "${sms_eth_internal}")" \
    802-3-ethernet.mtu "${roce_mtu}"
ip link set dev "${sms_eth_internal}" mtu "${roce_mtu}"
```

2.8 Optionally add Omni-Path support services on *head node*

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

```
dnf -y install opa-basic-tools
```

Tip

Note: OmniPath networks require a subnet management service that can typically be run on either an administrative node, or on the switch itself. The optimal placement and configuration of the subnet manager is beyond the scope of this document, but Rocky 10 provides the `opa-fm` package should you choose to run it on the *head node*.

Tip

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

```
ipmitool -E -I lanplus -H ${bmc_ipaddr} -U root chassis \
    bootdev pxe options=persistent
```

3 Install OpenHPC Components

3.1 Enable OpenHPC Repository

With the Base OS installed and booted, the next step is to add desired OpenHPC packages onto the *head node* in order to provide provisioning and resource management services for the rest of the cluster.

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 *head node* to the OpenHPC repository, or alternatively, that the OpenHPC repository be mirrored locally. In cases where network external connectivity is available, OpenHPC provides an `ohpc-release` package that includes GPG keys for package signing and enabling the repository. The example which follows illustrates installation of the `ohpc-release` package directly from the OpenHPC build server.

```
dnf -y install http://repos.openhpc.community/OpenHPC/4/EL_10/aarch64/\
ohpc-release-4-1.el10.aarch64.rpm
```

Tip

Note: Many sites may find it useful or necessary to maintain a local copy of the OpenHPC repositories. To facilitate this need, standalone tar archives are provided – one containing a repository of binary packages as well as any available updates, and one containing a repository of source RPMS. The tar files also contain a simple bash script to configure the package manager to use the local repository after download. To use, simply unpack the tarball where you would like to host the local repository and execute the `make_repo.sh` script. Tar files for this release can be found at http://repos.openhpc.community/dist/4.2

3.2 Install OpenHPC Base Packages

Now OpenHPC packages can be installed. The `ohpc-base` meta-package requires a module tool to manage the user environment, and OpenHPC provides two of them: Lmod in the `lmod-ohpc` package and Environment Modules in the `environment-modules-ohpc` package. Both read the OpenHPC module hierarchy and provide the `module` command. The `${modules_package}` variable in `input.local` selects the one installed on the head node and, later in this guide, on the compute nodes; it defaults to `lmod-ohpc`. To add the base package and the module tool on the head node issue the following:

```
dnf -y install ohpc-base ${modules_package:-lmod-ohpc}
```

Tip

Naming the module tool keeps the choice explicit; `ohpc-base` alone normally pulls in Lmod, which it suggests. The two tools can also be installed side by side: the startup scripts `/etc/profile.d/modules.sh` and `modules.csh`, which define the `module` command, and `/usr/bin/modulecmd` are links managed with `alternatives`, and `environment-modules-ohpc` takes precedence over `lmod-ohpc` when both are present. Run `alternatives --config modules.sh` to select the active tool.

4 Install Flux

4.1 Add Flux on Head Node

The following command adds the Flux framework to the chosen *head node*: `flux-core` (the message broker and job manager), `flux-sched` (the Fluxion graph-based scheduler), `flux-security` (the job request signing library and the setuid IMP that starts job processes as the submitting user) and `flux-pmix` (PMIx support, which OpenMPI jobs need to bootstrap under Flux). The latter also installs `pmix-ohpc` under `/opt/ohpc`, which the compute nodes mount from the head node. Note that the components needed by the compute nodes will be added to the corresponding compute image in a subsequent step. Flux runs as a *system*

instance: one Flux broker per node, all connected in a tree-based overlay network rooted at the head node. Note that Flux leverages the use of the *munge* library to sign job requests and this daemon also needs to be running on all hosts within the resource management pool.

```
# Install Flux server meta-package
dnf -y install ohpc-flux-server
```

The connections between the Flux brokers are authenticated and encrypted using a CURVE certificate that has to be identical on all nodes and must only be readable by the `flux` user. The certificate is generated as the `flux` user and then moved into place, as `flux` has no write access to `/etc/flux/system`:

```
runuser -u flux -- flux keygen /tmp/curve.cert
mv /tmp/curve.cert /etc/flux/system/curve.cert
chown flux:flux /etc/flux/system/curve.cert
chmod 0400 /etc/flux/system/curve.cert
```

Next, configure the Flux system instance. The `bootstrap` table lists all hosts in rank order: the *head node* is rank 0 and the parent of all compute nodes, which connect to it on TCP port 8050. The `exec` and `access` tables let users other than the `flux` instance owner run jobs through the IMP, and the `resource` table withholds the *head node* (rank 0) from scheduling. Since the same configuration is used on all nodes, it will also be made available to the compute nodes in a subsequent step.

```
cat > /etc/flux/system/conf.d/system.toml <<EOF
# Path to the IMP used to start jobs as the submitting user
[exec]
imp = "/usr/libexec/flux/flux-imp"

# Allow users other than the instance owner to run jobs, and give root
# owner privileges for administration
[access]
allow-guest-user = true
allow-root-owner = true

# Overlay network: hosts in rank order; the head node is rank 0 and the
# parent of all compute nodes
[bootstrap]
curve_cert = "/etc/flux/system/curve.cert"

[[bootstrap.hosts]]
host = "${sms_name}"
bind = "tcp://${sms_ip}:8050"
connect = "tcp://${sms_ip}:8050"

[[bootstrap.hosts]]
host = "${compute_prefix}[1-${num_computes}]"

# Do not schedule jobs on the head node (rank 0). The rank is used
# instead of the hostname, as hostnames cannot be mapped to ranks
# before the resources of all nodes have been discovered.
[resource]
exclude = "0"
EOF
```

Finally, configure the IMP. As the only part of Flux that runs with root privileges, it has its own configuration,

which has to be owned by `root` and must not be writable by other users. It only allows the `flux` user to start jobs on behalf of the submitting user, and only with the installed Flux job shell:

```
cat > /etc/flux/imp/conf.d/imp.toml <<'EOF'
[exec]
allowed-users = [ "flux" ]
allowed-shells = [ "/usr/libexec/flux/flux-shell" ]
EOF
chmod 0644 /etc/flux/imp/conf.d/imp.toml
```

Tip

Note: Each `host` entry in the `bootstrap` table must exactly match the output of the `hostname` command on that node, which is how each Flux broker determines its own rank. With the configuration above, the resources of each compute node are discovered via hwloc as its broker joins the instance, and the full resource inventory is only defined once all compute nodes have joined. Production systems will typically describe their resources statically instead, using `[[resource.config]]` entries or an R file generated with `flux R`, so that jobs can be scheduled even while some nodes are down. Changes to the `bootstrap` table only take effect after restarting Flux on all nodes. Please refer to the Flux Administrator's Guide and `flux-config-resource(5)` for more detailed information.

5 Install Warewulf Provisioner

5.1 Install Warewulf on *Head Node*

To add support for provisioning services, we install the Warewulf provisioning system packages. Then the main Warewulf configuration files are edited to reflect the environment. The `yq` package is also installed to aid in configuring Warewulf `yaml` files.

```
# Make sure required variables are set
: "${sms_eth_internal:?ERROR: sms_eth_internal not set}"
: "${internal_network:?ERROR: internal_network not set}"

# Install Warewulf
dnf -y install warewulf-ohpc yq
```

5.2 Complete basic Warewulf setup for *head node*

At this point, all of the packages necessary to use Warewulf on the *head node* should be installed. Next, we need to update the configuration to allow Warewulf to work with Rocky 10, and to support local provisioning using a second private interface (refer to Figure 1).

warewulfd serves each node a system overlay and a runtime overlay. With `secure: true` in `warewulf.conf`, it serves the runtime overlay only to requests from privileged source ports (1-1023), which only root can use. The system overlay is served to any client, so files that only root on a node may read go in runtime overlays: the munge key, and the SSH host keys, which Warewulf's `default` profile puts in the system overlay. A node reads this setting from its system overlay when it boots, so the steps below set it before warewulfd first

starts and before any node boots. A node booted before the change fetches its runtime overlay again only after a reboot.

```
# Edit the warewulf.conf file to use appropriate interface and settings
yq -i '.ipaddr = "'"${sms_ip}"'"' /etc/warewulf/warewulf.conf
yq -i '.netmask = "'"${internal_netmask}"'"' /etc/warewulf/warewulf.conf
yq -i '.network = "'"${internal_network}"'"' /etc/warewulf/warewulf.conf
yq -i '.dhcp["range start"] = "'"${internal_network}"'"' \
    /etc/warewulf/warewulf.conf
yq -i '.dhcp["range end"] = "static"' /etc/warewulf/warewulf.conf
yq -i '.dhcp.template = "static"' /etc/warewulf/warewulf.conf

# Edit the nodes.conf to mount /opt on boot
sed -i "s/defaults,noauto,nofail,ro/defaults,nofail,ro/" \
    /etc/warewulf/nodes.conf

# Turn on debugging messages
yq -i '.nodeprofiles.default.kernel.args -= ["quiet"]' \
    /etc/warewulf/nodes.conf
echo "log-debug" >> /etc/dnsmasq.d/ww4-debug.conf

# Serve the runtime overlay only to privileged source ports (root)
yq -i '.warewulf.secure = true' /etc/warewulf/warewulf.conf

# Enable and start the warewulfd service before using wwctl.
systemctl enable --now warewulfd.service

# Create a new "nodes" profile and inherit the "default" profile
wwctl profile add nodes --profile default --comment "Nodes profile"

# Create a new "nodeconfig" overlay to store node
# configuration files, inherit the timezone settings
# from the SMS, and use the syncuser overlay.
# The "secrets" runtime overlay holds files only root may read.
# Its /etc is made 0755 here: "import --parents" would give it the
# mode of the first key imported (0500), and the nodes' /etc with it.
# The SSH host keys move from the system overlays to the runtime
# overlays ("~" removes an overlay inherited from "default").
wwctl overlay create nodeconfig
wwctl overlay create secrets
wwctl overlay mkdir secrets /etc
wwctl profile set --yes nodes \
    --system-overlays nodeconfig,localtime,~ssh.host_keys \
    --runtime-overlays syncuser,secrets,ssh.host_keys

# Set default network configuration
wwctl profile set -y nodes --netname=default --netdev="${eth_provision}"
wwctl profile set -y nodes --netname=default --netmask="${internal_netmask}"
wwctl profile set -y nodes --netname=default --gateway="${ipv4_gateway}"
wwctl profile set -y nodes --netname=default --nettagadd=DNS="${dns_servers}"

# Configuring Warewulf will restart/enable relevant services to support provisioning
wwctl configure --all

# Generate ssh keys (usually generated on login)
bash /etc/profile.d/ssh_setup.sh
```

5.3 Create Compute Node Image

First, we create a minimal compute node image. Warewulf supports using container images as the base file system for provisioning, and it can import these directly from an OCI registry like Docker Hub. Container images must be created especially for use with Warewulf since they need to include things like a kernel and an init system. In this example we will import our base image from a set maintained by the Warewulf community on the GitHub container registry.

The `wwctl image exec` command runs the commands below it, these commands also be run interactively one a time with the command `wwctl image shell rocky-10`. You can add `/bin/false` as the last command to prevent the image from rebuilding (it will show an error) and rebuild later with the `wwctl image build` command.

```
# Import the base image from Warewulf
wwctl image import docker://ghcr.io/warewulf/warewulf-rockylinux:10 \
    rocky-10 --syncuser

# Define chroot location
CHROOT=$(wwctl image show rocky-10)
export CHROOT
```

```
# Enable OpenHPC inside image and update image.
# Disable image build on exit, rebuild later.
wwctl image exec --build=false rocky-10 -- /bin/bash -ex <<- EOF
  dnf -y install dnf-utils
  dnf -y install http://repos.openhpc.community/OpenHPC/4/EL_10/\
aarch64/ohpc-release-4-1.el10.aarch64.rpm
  dnf -y update
EOF
```

5.4 Add OpenHPC Components

Next, we add additional components to include resource management client services, NTP support, and other additional packages to support the default OpenHPC environment. This process modifies the base provisioning image and will access the Base OS and OpenHPC repositories to resolve package install requests.

The instructions below are designed to be copied and pasted all at once into a terminal. Alternatively you can run `wwctl image shell rocky-10` to "run" the node image interactively and run the commands one at a time, this method is recommended.

We begin by installing a few common base packages:

```
# Install compute node base meta-package
wwctl image exec --build=false rocky-10 -- /bin/bash -ex <<- EOF

  dnf -y install epel-release

  dnf -y install ohpc-base-compute
EOF
```

Now, we can include additional required components to the compute instance including resource manager client, NTP, and development environment modules support. The module tool is the one selected with `${modules_package}`, as on the head node.

Adding packages can be done by entering the image with `wwctl image shell`, `wwctl image exec`, or using a CHROOT.

```
wwctl image exec --build=false rocky-10 -- /bin/bash -ex <<- EOF
  # Add Flux client support meta-package
  dnf -y install ohpc-flux-client

  # Remove the munge key the packages generate: the nodes get the
  # head node's key from the runtime overlay
  rm -f /etc/munge/munge.key

  # Enable services to start on boot
  systemctl enable munge.service
  systemctl enable flux.service

  # Add Network Time Protocol (NTP) support
  dnf -y install chrony

  # Include modules user environment
  dnf -y install ${modules_package:-lmod-ohpc}
EOF
```

5.5 Import files

The Warewulf system includes functionality to import arbitrary files from the provisioning server for distribution to managed hosts through a system called "overlays". Some files, like `/etc/passwd`, and `/etc/hosts` handled in this way by default. Here we add directories and files to the `nodesconfig` overlay that is applied to all nodes.

The SSH host keys come in the runtime overlay, which `wwclient` applies on the node. With two-stage provisioning the node's initramfs applies it first, before systemd starts; when the node boots in one stage from iPXE, `wwclient` is the first to apply it. A systemd drop-in has sshd and its host key generation start after `wwclient`, so sshd serves the host keys from the head node rather than new ones. With two-stage provisioning it delays sshd by a few seconds.

```
# Add the following to support unprivileged user namespaces for tools like Apptainer
wwctl overlay import --parents nodeconfig /etc/subuid
wwctl overlay import --parents nodeconfig /etc/subgid

# Identify head node as local NTP server, configure it with a template with a Tag
wwctl overlay import --parents nodeconfig \
 /opt/ohpc/pub/examples/chrony.conf.ww /etc/chrony.conf.ww
wwctl profile set --yes nodes --tagadd ntpserver="${sms_ip}"

# Configure systemd and NetworkManger to wait for the network to be fully up.
wwctl overlay import --parents nodeconfig \
 /opt/ohpc/pub/examples/network/NetworkManager-wait-online.service.d/override.conf \
 /etc/systemd/system/NetworkManager-wait-online.service.d/override.conf

# Start sshd after wwclient has applied the runtime overlay (SSH host keys)
mkdir -p /opt/ohpc/admin/warewulf
printf '[Unit]\nWants=wwclient.service\nAfter=wwclient.service\n' \
    > /opt/ohpc/admin/warewulf/wait-wwclient.conf
for unit in sshd.service sshd-keygen@.service; do
    wwctl overlay import --parents nodeconfig \
        /opt/ohpc/admin/warewulf/wait-wwclient.conf \
        /etc/systemd/system/${unit}.d/wait-wwclient.conf
done
```

5.6 Configure Flux

Similarly, the Flux brokers on the compute nodes need the same system instance configuration, CURVE certificate and IMP configuration as the *head node*, as well as the cryptographic key that is required by the *munge* authentication library to be available on every host in the resource management pool. The CURVE certificate holds the system instance's secret key, so it goes with the munge key in the `secrets` runtime overlay, which warewulfd serves only to root on the nodes. A systemd drop-in has munge and the Flux broker start after `wwclient` has applied it. To import these, issue the following:

```
# Configure the Flux system instance and the IMP
wwctl overlay import --parents nodeconfig /etc/flux/system/conf.d/system.toml
wwctl overlay import --parents nodeconfig /etc/flux/imp/conf.d/imp.toml

# The CURVE certificate in the runtime overlay, readable by flux only.
# Make its directories 0755 first: "import --parents" would give them
# the file's mode (0500), which the nodes' /etc/flux would then take.
wwctl overlay mkdir secrets /etc/flux/system
wwctl overlay import --parents secrets /etc/flux/system/curve.cert
wwctl overlay chown secrets /etc/flux/system/curve.cert \
    "$(id -u flux):$(id -g flux)"
wwctl overlay chmod secrets /etc/flux/system/curve.cert 0400

# Configure munge: the key in the runtime overlay, readable by munge only
wwctl overlay import --parents secrets /etc/munge/munge.key
wwctl overlay chown secrets /etc/munge/munge.key "$(id -u munge):$(id -g munge)"
wwctl overlay chown secrets /etc/munge "$(id -u munge):$(id -g munge)"
wwctl overlay chmod secrets /etc/munge 0700
wwctl overlay chmod secrets /etc/munge/munge.key 0400

# Start munge and flux after wwclient has applied the runtime overlay
# (munge key, CURVE certificate)
for unit in munge.service flux.service; do
    wwctl overlay import --parents nodeconfig \
        /opt/ohpc/admin/warewulf/wait-wwclient.conf \
        /etc/systemd/system/${unit}.d/wait-wwclient.conf
done
```

5.7 Finalizing Provisioning Configuration

Warewulf provisions a node with an image then customizes it with overlays. This section highlights creation of the node image and overlays, followed by the registration of desired compute nodes.

5.7.1 Two-Stage Provisioning (optional)

Warewulf optionally supports two-stage provisioning to ramfs or disk on the compute nodes. The nodes are still stateless, but the image is copied to disk during the provisioning process and is reprovisioned on every boot. This can be useful for cases when the compute node images are large, for example GPU drivers, or the compute nodes have limited memory.

To configure two-stage provisioning, install Dracut and ignition (ignition can also be used to provision swap and local storage (see Warewulf documentation for details) on the compute node. The example below provisions to ramfs, which is the default. To provision to disk, see the next section. To aid in debugging, it is helpful to add `rd.shell` to the kernel arguments.

```
## Add rd.shell to kernel arguments in the "default" profile.
yq -i '.nodeprofiles.default.kernel.args += ["rd.shell"]' \
  /etc/warewulf/nodes.conf
```

```
## Install Dracut in the image
wwctl image exec --build=false rocky-10 -- /usr/bin/dnf install -y \
  warewulf-ohpc-dracut

## Configure Dracut. Note the leading and trailing spaces
# in the quotes in "add_dracutmodules"
echo 'hostonly="no"' > "$CHROOT"/etc/dracut.conf.d/wwinit.conf
echo 'add_dracutmodules+=" wwinit "' \
  >> "$CHROOT"/etc/dracut.conf.d/wwinit.conf

## Enable Dracut boot for compute nodes that use the "nodes" profile
wwctl profile set --yes nodes --tagadd IPXEMenuEntry=dracut
```

Optionally, configure the compute node to provision to disk. Two stage provisioning (previous step) must also be enabled. The compute node will use, **and erase**, the disk specified by `node_disk`.

```
## Install disk tools
wwctl image exec --build=false rocky-10 -- /usr/bin/dnf install -y \
  ignition gdisk

## Add ignition Dracut module.
echo 'add_dracutmodules+=" ignition "' \
  >> "$CHROOT"/etc/dracut.conf.d/wwinit.conf

## Create the target "rootfs" partition and filesystem
wwctl profile set --yes nodes \
  --diskname "${node_disk}" --diskwipe \
  --partname rootfs --partcreate --partnumber 1 \
  --fsname rootfs --fswipe --fsformat ext4 --fspath /

## Enable provision-to-disk for compute nodes that use the "nodes" profile
wwctl profile set nodes --yes --root=/dev/disk/by-partlabel/rootfs
```

5.7.2 Build Image and Overlays

The bootstrap image includes the runtime kernel and associated modules, as well as some simple scripts to complete the provisioning process. We explicitly rebuild Dracut and the image at the end because rebuilding the image is disabled in earlier steps. It is good practice to rebuild the overlays when after configuration changes.

```
## Build the Dracut initramfs
wwctl image exec --build=false rocky-10 -- /usr/bin/dracut --force \
  --regenerate-all
```

```
# Build image
wwctl image build rocky-10
wwctl overlay build
```

6 Additional Customization

This chapter highlights common additional customizations that can *optionally* be applied to the local cluster environment. Details on the steps required for each of these customizations are discussed further in the following sections.

6.1 Enable InfiniBand drivers

If your compute resources support InfiniBand, the following commands add OFED and PSM support using base distro-provided drivers to the compute image.

```
# Add IB support and enable
dnf -y --installroot="$CHROOT" groupinstall "InfiniBand Support"
```

6.2 Enable RoCE drivers

If your compute resources support RoCE, the following commands add the same RDMA stack to the compute image. If you enabled InfiniBand drivers above, the stack is already there and the script skips the install.

```
# Add RDMA support
dnf -y --installroot="$CHROOT" groupinstall "InfiniBand Support"
```

If you set `${roce_mtu}`, the following gives the computes' provisioning interface the same MTU as the *head node*'s internal interface.

```
wwctl profile set -y nodes --netname=default --mtu="${roce_mtu}"
```

To run MPI over RoCE, select the RDMA device and port for UCX, for example `UCX_NET_DEVICES=mlx5_1:1`; `rdma link` lists each RDMA device with its network interface.

6.3 Enable Omni-Path drivers

If your compute resources support Omni-Path, the following commands add OPA support using base distro-provided drivers to the compute image.

```
# Add OPA support and enable
dnf -y --installroot="$CHROOT" install opa-basic-tools
dnf -y --installroot="$CHROOT" install libpsm2
```

6.4 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 *head node*, and the following commands can be used to update the maximum locked memory settings on both the *head node* and compute nodes:

```
# Update memlock settings on head node
echo '* soft memlock unlimited' >> \
  /etc/security/limits.d/40-ohpc-limits.conf
echo '* hard memlock unlimited' >> \
  /etc/security/limits.d/40-ohpc-limits.conf

# Update memlock settings on compute
echo '* soft memlock unlimited' >> \
    "$CHROOT/etc/security/limits.d/40-ohpc-limits.conf"
echo '* hard memlock unlimited' >> \
    "$CHROOT/etc/security/limits.d/40-ohpc-limits.conf"
```

6.5 NVIDIA GPU Drivers

To add the optional NVIDIA GPU driver to the compute nodes, an additional external dnf repository provided by NVIDIA must be configured. Once the repository is configured, the GPU driver needs to be installed on the compute image and the corresponding toolkit installed on the head node.

OpenHPC provides a convenience package to enable the NVIDIA repository locally along with compatibility packages that integrate the NVIDIA HPC SDK within the standard OpenHPC user environment.

Tip

Note: The driver is NVIDIA's open GPU kernel module (`kmod-nvidia-open-dkms`), the only kernel module in NVIDIA's EL10 repository. It supports GPUs of the Turing architecture and newer.

```
# Add NVIDIA GPU driver repository to the head node
dnf -y install cuda-repo-ohpc

# Add NVIDIA GPU driver repository to the compute nodes
dnf -y --installroot="$CHROOT" install cuda-repo-ohpc

# Install the GPU driver on the compute nodes
dnf -y --installroot="$CHROOT" install nvidia-driver-cuda kmod-nvidia-open-dkms

# Build the driver for the image's kernel (DKMS otherwise targets the running kernel)
wwctl image exec --build=false rocky-10 -- /bin/sh -c \
  "dkms autoinstall -k \$(ls /lib/modules | sort -V | tail -n 1)"

# Enable DKMS service to automatically rebuild driver
wwctl image exec --build=false rocky-10 -- /bin/sh -c \
  "systemctl enable dkms.service"

# Install the toolkit on the head node
dnf -y install cuda-devel-ohpc
```

6.6 Enable System Log Forwarding

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 compute node's memory footprint if it is stateless. The following commands highlight the steps necessary to configure compute nodes to forward their logs to the head node, and to allow the head node to accept these log requests.

```
# Configure head node to receive messages and reload rsyslog configuration
echo 'module(load="imudp")' >> /etc/rsyslog.d/ohpc.conf
echo 'input(type="imudp" port="514")' >> /etc/rsyslog.d/ohpc.conf
systemctl restart rsyslog.service

# Define compute node forwarding destination
wwctl image exec --build=false rocky-10 -- /bin/sh -c \
  "echo '*.* @${sms_ip}:514' > /etc/rsyslog.d/ohpc-forward.conf"

# Disable most local logging on computes.
# Emergency and boot logs will remain on the compute nodes
sed -i 's/^\*\.info/#\*\.info/' \
    "$CHROOT/etc/rsyslog.conf"
sed -i 's/^authpriv/#authpriv/' \
    "$CHROOT/etc/rsyslog.conf"
sed -i 's/^mail/#mail/' \
    "$CHROOT/etc/rsyslog.conf"
sed -i 's/^cron/#cron/' \
    "$CHROOT/etc/rsyslog.conf"
sed -i 's/^uucp/#uucp/' \
    "$CHROOT/etc/rsyslog.conf"

# Let the compute nodes pick up the new configuration
wwctl image exec --build=false rocky-10 -- /bin/sh -c \
  "systemctl enable rsyslog.service"
```

6.7 Cluster Admin Tools

6.7.1 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
dnf -y install clustershell

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

6.7.2 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
dnf -y install genders-ohpc

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

6.7.3 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
dnf -y install conman-ohpc

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

# Enable and start conman
systemctl enable conman.service
systemctl start conman.service
```

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

6.7.4 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 head and compute nodes
dnf -y install nhc-ohpc
dnf -y --installroot="$CHROOT" install nhc-ohpc
```

6.8 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
dnf -y install magpie-ohpc
```

6.9 Charliecloud

Typical Charliecloud workflows are based around Docker containers, but it is not strictly necessary to install Docker itself on the HPC resource. A common pattern is to build the Docker container on a laptop or VM and upload the result to the cluster for use with Charliecloud. More information can be found at https://hpc.github.io/charliecloud/

7 Deploy Cluster

7.1 Register Compute Nodes for Provisioning

Nodes can be registered for provisioning using the following:

```
for ((i=0; i<num_computes; i++)) ; do
    wwctl node add --image=rocky-10 --profile=nodes --netname=default \
        --ipaddr="${c_ip[$i]}" --hwaddr="${c_mac[$i]}" "${c_name[i]}"
done
```

7.2 Optionally configure IPoIB on *compute nodes*

To use IPoIB on the compute nodes (required if planning to mount Lustre/BeeGFS over IB; see Optionally add InfiniBand support services on *head node*), give each compute its own address `${c_ipoib[i]}` on the IPoIB interface `${ipoib_dev}`, with the `${ipoib_netmask}` netmask (`${ipoib_prefix_length}` prefix). These are per-node settings, so they belong in the Warewulf node definitions:

Warewulf's NetworkManager overlay configures the interface on each node from a network device of type `infiniband`:

```
for ((i=0; i<num_computes; i++)) ; do
    wwctl node set --yes "${c_name[$i]}" --netname=ib --type=infiniband \
        --netdev="${ipoib_dev}" --ipaddr="${c_ipoib[$i]}" \
        --netmask="${ipoib_netmask}"
done
```

Tip

Note: If, once you boot the nodes, you find that the IB interface has a name other than `${ipoib_dev}` and hence was not assigned the given IP, you can fix it by setting `--hwaddr` to the **literal last 6 bytes** of the interface's hardware address (the trailing 6 octets of the 20-byte IPoIB address, *not* the 48-bit InfiniBand "compatibility MAC"), so that the udev.netname overlay can set the persistent interface name. This does not make it PXE/DHCP-boot over IB.

7.3 Rebuild the Image and Overlays

Rebuild the image to incorporate any optional packages installed into the chroot during customization, then rebuild the overlays and update the Warewulf configuration.

```
# Rebuild image to include any customization chroot changes
wwctl image build rocky-10

# build the overlays for all the nodes
wwctl overlay build

# Update Warewulf configure
wwctl configure --all
```

7.4 Start Flux

In an earlier section, the Flux resource manager was installed and configured for use on both the *head node* 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. The Flux broker on the *head node* is rank 0 of the system instance; the brokers on the *compute* nodes are started on boot and connect to it.

```
# Start munge and the Flux system instance on head node
systemctl enable --now munge.service
systemctl enable --now flux.service
```

Running systems can pick up most configuration changes with `flux config reload`, except for changes to the `bootstrap` table, which require restarting `flux.service` on all nodes.

7.5 Boot Compute Nodes

At this point, the *head node* 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.

```
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:

```
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
```

7.6 Wait for Compute Nodes

Continue after the compute nodes boot and complete their install. A node has booted when `systemd` has finished starting it, and reports `running`, or `degraded` if a unit failed. To check one node, and to list its failed units:

```
ssh ${compute_prefix}1 systemctl is-system-running
ssh ${compute_prefix}1 systemctl --failed
```

A node that never answers has failed to provision; check its console.

8 Install OpenHPC Development Components

The install procedure outlined in Install OpenHPC Components highlighted the steps necessary to install a *head node*, 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.

8.1 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
dnf -y install ohpc-autotools
dnf -y install EasyBuild-ohpc
dnf -y install hwloc-ohpc
dnf -y install spack-ohpc
dnf -y install valgrind-ohpc
```

8.2 Compilers

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

```
dnf -y install gnu16-compilers-ohpc
```

8.3 MPI Stacks

For MPI development and runtime support, OpenHPC provides pre-packaged builds for a variety of MPI families and transport layers. Currently available options and their applicability to various network transports are summarized in the Available MPI variants table below. The command that follows installs a starting set of MPI families that are compatible with both ethernet and high-speed fabrics.

Table: Available MPI builds

	Ethernet (TCP)	InfiniBand
MPICH	✓	
OpenMPI	✓	✓

```
dnf -y install openmpi5-pmix-gnu16-ohpc mpich-ofi-gnu16-ohpc
```

Note that the MPICH and OpenMPI builds support two related transport layers for 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:

```
dnf -y install mpich-ucx-gnu16-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.

```
module avail mpich

-------------------- /opt/ohpc/pub/moduledeps/gnu16---------------------
   mpich/3.4.3-ofi    mpich/3.4.3-ucx (D)
```

8.4 Performance Tools

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

```
# Install perf-tools meta-package
dnf -y install ohpc-gnu16-perf-tools
```

8.5 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.

```
dnf -y install modules-defaults-gnu16-openmpi5-ohpc
```

Tip

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

- modules-defaults-gnu16-mpich-ofi-ohpc
- modules-defaults-gnu16-mpich-ucx-ohpc

8.6 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 MPI Stacks.

```
dnf search petsc-gnu16 ohpc
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
============================ N/S matched: petsc-gnu16, ohpc ============================
petsc-gnu16-mpich-ohpc.x86_64 : Portable Extensible Toolkit for Scientific Computation
petsc-gnu16-openmpi5-ohpc.x86_64 : Portable Extensible Toolkit for Scientific Comp...
```

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 an earlier section, so the packages will be immediately available for use on the cluster after installation on the **head node**.

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 the Package Manifest 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
dnf -y install ohpc-gnu16-serial-libs
dnf -y install ohpc-gnu16-io-libs
dnf -y install ohpc-gnu16-python-libs
dnf -y install ohpc-gnu16-runtimes
```

```
# Install parallel lib meta-packages for all available MPI toolchains
dnf -y install ohpc-gnu16-mpich-parallel-libs
dnf -y install ohpc-gnu16-openmpi5-parallel-libs
```

9 Post Provisioning

9.1 Check Cluster Status

After this, check the status of the Flux system instance. `flux overlay status` shows whether the broker on each compute node has connected to the *head node*, and `flux resource status` shows the state of each node as seen by the scheduler. All compute nodes should be listed as *avail*, while the *head node* is listed as *exclude* since it has been withheld from scheduling:

```
flux overlay status
flux resource status
```

Continue after every compute node is free for jobs. To check:

```
flux resource list -s free
```

In case of additional Flux issues, ensure that `munge.service` and `flux.service` are running on the affected nodes, that the hostname of each node exactly matches its entry in the `hosts` array of `/etc/flux/system/conf.d/system.toml`, and that the configuration and CURVE certificate are identical across the nodes. You may also consult the Flux Administrator's Guide.

9.2 Add User

We will now add a new user to the cluster. This will be used later to run a test job.

```
useradd -m test
```

Warewulf's `wwclient` on each compute node pulls the runtime overlay, which carries the user and group files, every `update interval` seconds set in `warewulf.conf`. Rebuild the overlay, then have `wwclient` pull it now rather than at its next interval:

```
wwctl overlay build
pdsh -w "${compute_prefix}[1-${num_computes}]" systemctl reload wwclient
```

Continue after the user is present on every compute node. To check one node:

```
ssh ${compute_prefix}1 getent passwd test
```

9.3 Update NHC

Generate NHC configuration file based on compute node environment

```
pdsh -w c1 "/usr/sbin/nhc-genconf -H '*' -c -" | dshbak -c
```

10 Test Cluster

We now run some simple tests on the cluster to ensure it is operational.

10.1 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 **head** node that can be used to run an example job.

OpenHPC includes a simple "hello-world" MPI application in the `/opt/ohpc/pub/examples` directory that can be used for this quick compilation and execution. OpenHPC also provides a companion job-launch utility named `prun` that is installed in concert with the pre-packaged MPI toolchains. This convenience script provides a mechanism to abstract job launch across different resource managers and MPI stacks such that a single launch command can be used for parallel job launch in a variety of OpenHPC environments. It also provides a centralizing mechanism for administrators to customize desired environment settings for their users.

10.2 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). Under Flux, `prun` launches one MPI task per allocated node unless the number of tasks is set via `FLUX_JOB_NTASKS`:

```
# Switch to "test" user
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 ~]$ flux alloc -N 2 -n 8

[test@c1 ~]$ FLUX_JOB_NTASKS=8 prun ./a.out

[prun] Master compute host = c1
[prun] Resource manager = flux
[prun] Launch cmd = flux run -N 2 -n 8 -o pmi=pmix ./a.out (family=openmpi5)

 Hello, world (8 procs total)
    --> Process #   0 of   8 is alive. -> c1
    --> Process #   4 of   8 is alive. -> c2
    --> Process #   1 of   8 is alive. -> c1
    --> Process #   5 of   8 is alive. -> c2
    --> Process #   2 of   8 is alive. -> c1
    --> Process #   6 of   8 is alive. -> c2
    --> Process #   3 of   8 is alive. -> c1
    --> Process #   7 of   8 is alive. -> c2
```

Tip

The following table provides approximate command equivalences between Flux and SLURM:

Command	Flux	SLURM
Submit *batch* job	flux batch [job script]	sbatch [job script]
Request *interactive* shell	flux alloc	salloc
Launch parallel tasks	flux run	srun
Delete job	flux cancel [job id]	scancel [job id]
Queue status	flux queue list	sinfo
Job status	flux jobs [job id]	scontrol show job [job id]
Node status	flux resource status	scontrol show node [node id]

10.3 Batch execution

For batch execution, a job script with Flux directives can be submitted to the resource manager. The example below illustrates creating such a script and submitting a batch job for execution using the same executable referenced in the previous interactive example.

```
# Create job script (and edit to set desired job sizing characteristics)
[test@sms ~]$ cat > job.mpi <<'EOF'
#!/bin/bash
#----------------------------------------------------------
# Job name
#flux: --job-name=test

# Name of stdout output file ({{id}} expands to the job ID)
#flux: --output=job.{{id}}.out

# Total number of nodes and task slots requested
#flux: -N 2
#flux: -n 16

# Run time - 1.5 hours
#flux: -t 90m
#----------------------------------------------------------

# Launch MPI-based executable, one MPI task per requested slot
export FLUX_JOB_NTASKS=16
prun ./a.out
EOF

# Submit job for batch execution
[test@sms ~]$ flux batch job.mpi
f2kQbuWdGb5
```

Tip

The use of the `{{id}}` token in the example batch job script shown is a convenient way to track application output on an individual job basis. The token is replaced with the Flux job ID once assigned, so the output of job `f2kQbuWdGb5` in this example is written to `job.f2kQbuWdGb5.out`. The status of the job can be followed with `flux jobs`.

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 the Inputs section. 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 **head node** 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

```
dnf -y install docs-ohpc
```

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

```
cp /opt/ohpc/pub/doc/recipes/rocky10/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.

```
cp -p /opt/ohpc/pub/doc/recipes/rocky10/aarch64/warewulf/flux/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.

```
export OHPC_INPUT_LOCAL=./input.local
./recipe.sh
```

B Upgrading OpenHPC Packages

As newer OpenHPC releases are made available, users are encouraged to upgrade their locally installed packages against the latest repository versions to obtain access to bug fixes and newer component versions. This can be accomplished with the underlying package manager as OpenHPC packaging maintains versioning state across releases. Also, package builds available from the OpenHPC repositories have "`-ohpc`" appended to their names so that wild cards can be used as a simple way to obtain updates. The following general procedure highlights a method for upgrading existing installations. When upgrading from a minor release older than v4, you will first need to update your local OpenHPC repository configuration to point against the v4 release (or update your locally hosted mirror). Refer to an earlier section for more details on enabling the latest repository. In contrast, when upgrading between micro releases on the same branch (e.g. from v4 to 4.2), there is no need to adjust local package manager configurations when using the public repository as rolling updates are pre-configured.

1. (Optional) Ensure repo metadata is current (on head node and in chroot location(s)). Package managers will naturally do this on their own over time, but if you are wanting to access updates immediately after a new release, the following can be used to sync to the latest.

```
dnf clean expire-cache
dnf --installroot="$CHROOT" clean expire-cache
```

2. Upgrade head node

```
dnf -y upgrade "*-ohpc"

# Any new Base OS provided dependencies can be installed by
# updating the ohpc-base metapackage
dnf -y upgrade "ohpc-base"
```

3. Upgrade packages in compute image

```
dnf -y --installroot="$CHROOT" upgrade "*-ohpc"

# Any new compute-node Base OS provided dependencies can be installed by
# updating the ohpc-base-compute metapackage
dnf -y --installroot="$CHROOT" upgrade "ohpc-base-compute"
```

4. Rebuild image(s)

The upgrade can recreate a munge key in the image (the munge key reaches the nodes in the `secrets` runtime overlay instead), so remove it before rebuilding:

```
rm -f ${CHROOT}/etc/munge/munge.key
wwvnfs --chroot $CHROOT
```

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

C Integration Test Suite

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

```
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 FLUX 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).

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

----------------------------------- SUMMARY -----------------------------------

Package version.............. : test-suite-2.0.0

Build user................... : ohpc-test
Build host................... : sms001
Configure date............... : 2020-10-05 08:22
Build architecture........... : aarch64
Compiler Families............ : gnu9
MPI Families................. : mpich openmpi4
Python Families.............. : python3
Resource manager ............ : FLUX
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 (FLUX/gnu9/mpich)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (FLUX/gnu9/mpich)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (FLUX/gnu9/mpich)
PASS rm_execution (exit status: 0)
1..3
ok 1 [libs/FFTW] Serial C binary runs under resource manager (FLUX/gnu9/openmpi4)
ok 2 [libs/FFTW] MPI C binary runs under resource manager (FLUX/gnu9/openmpi4)
ok 3 [libs/FFTW] Serial Fortran binary runs under resource manager (FLUX/gnu9/...
PASS rm_execution (exit status: 0)
```

D Customization

D.1 Adding local environment modules to OpenHPC hierarchy

Administrators may wish to add locally built software packages to the OpenHPC module hierarchy. This can be accomplished by creating module files in the appropriate locations under `/opt/ohpc/pub/moduledeps` or `/opt/ohpc/pub/modulefiles`. 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 `module` command should pick them up automatically. The example output below is from Lmod; Environment Modules lists the same modules in its own format.

```
# Create a simple example module
mkdir /opt/ohpc/pub/modulefiles/example1
cp /opt/ohpc/pub/examples/example.modulefile \
    /opt/ohpc/pub/modulefiles/example1/1.0

# Create an example module with a dependency
mkdir /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2
cp /opt/ohpc/pub/examples/example-mpi-dependent.modulefile \
    /opt/ohpc/pub/moduledeps/gnu7-openmpi3/example2/1.0

# Show modules
module avail
```

Example Output

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

--------------------------- /opt/ohpc/pub/moduledeps/gnu7 ---------------------------
   R/3.4.2        metis/5.1.0     ocr/1.0.1              pdtoolkit/3.24
   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
   autotools        (L)    example1/1.0 (L)    llvm5/5.0.0         prun/1.2        (L)
   clustershell/1.8        gnu7/7.2.0   (L)    ohpc         (L)    singularity/2.4

  Where:
   L:  Module is loaded

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

D.2 Rebuilding Packages from Source

OpenHPC packages can be rebuilt from source using the source RPMs available from the OpenHPC repository. This allows administrators to customize package builds for their specific needs. 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
sudo dnf -y install rpm-build dnf-plugins-core

# Install FFTW's build dependencies
sudo dnf builddep fftw-gnu16-openmpi5-ohpc

# Download SRPM from OpenHPC repository and install locally
dnf download --source fftw-gnu16-openmpi5-ohpc
rpm -i ./fftw-gnu16-openmpi5-ohpc-*.rpm

# Modify spec file as desired
cd ~/rpmbuild/SPECS
sed -i "s/enable-static=no/enable-static=yes/" fftw.spec

# Increment RPM release so the package manager will see an update
sed -i "s/Release:   400.ohpc.3.1/Release:   400.ohpc.3.2/" fftw.spec

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

# Install the new package
sudo dnf -y install \
        ../RPMS/$(uname -m)/fftw-gnu16-openmpi5-static-ohpc-*.$(uname -m).rpm
```

The new module file now appears along side the default.

```
$ module -t spider fftw
fftw/3.3.10-static
fftw/3.3.10
```

E Package Manifest

This appendix provides a summary of available meta-package groupings and all of the individual RPM packages that are available as part of this OpenHPC release. The meta-packages provide a mechanism to group related collections of RPMs by functionality and provide a convenience mechanism for installation. A list of the available meta-packages and a brief description is presented in the table below.

E.1 Available OpenHPC Meta-packages

Group Name	Description
ohpc-arm1-io-libs	Collection of IO library builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-mpich-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the MPICH runtime.
ohpc-arm1-openmpi5-parallel-libs	Collection of parallel library builds for use with the Arm Compiler for Linux and the openmpi5 runtime.
ohpc-arm1-perf-tools	Collection of performance tool builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-python3-libs	Collection of python3 related library builds for use with the Arm Compiler for Linux toolchain.
ohpc-arm1-serial-libs	Collection of serial library builds for use with the Arm Compiler for Linux toolchain.
ohpc-autotools	Collection of GNU autotools packages.
ohpc-base	Collection of base packages.
ohpc-base-compute	Collection of compute node base packages.
ohpc-flux-client	Collection of client packages for Flux.
ohpc-flux-server	Collection of server packages for Flux.
ohpc-gnu15-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu15-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu15-openmpi5-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-openmpi5-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-openmpi5-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu15-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu15-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu15-python-libs	Collection of python related library builds for use with GNU compiler toolchain.

Group Name	Description
ohpc-gnu15-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu15-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu15-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-gnu16-io-libs	Collection of IO library builds for use with GNU compiler toolchain.
ohpc-gnu16-mpich-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-mpich-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-mpich-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the MPICH runtime.
ohpc-gnu16-openmpi5-io-libs	Collection of IO library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-openmpi5-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-openmpi5-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain and the OpenMPI runtime.
ohpc-gnu16-parallel-libs	Collection of parallel library builds for use with GNU compiler toolchain.
ohpc-gnu16-perf-tools	Collection of performance tool builds for use with GNU compiler toolchain.
ohpc-gnu16-python-libs	Collection of python related library builds for use with GNU compiler toolchain.
ohpc-gnu16-python3-libs	Collection of python3 related library builds for use with GNU compiler toolchain.
ohpc-gnu16-runtimes	Collection of runtimes for use with GNU compiler toolchain.
ohpc-gnu16-serial-libs	Collection of serial library builds for use with GNU compiler toolchain.
ohpc-slurm-client	Collection of client packages for SLURM.
ohpc-slurm-server	Collection of server packages for SLURM.
ohpc-warewulf	Collection of base packages for Warewulf provisioning.

E.2 RPM Packages

What follows next in this Appendix is a series of tables that summarize the underlying RPM packages available in this OpenHPC release. These packages are organized by groupings based on their general functionality and each table provides information for the specific RPM name, version, brief summary, and the web URL where additional information can be obtained for the component. Note that many of the 3rd party community libraries that are pre-packaged with OpenHPC are built using multiple compiler and MPI families. In these cases, the RPM package name includes delimiters identifying the development environment for which each package build is targeted. Additional information on the OpenHPC package naming scheme is presented in Section 3rd Party Packages. The relevant package groupings and associated references are as follows:

- Administrative Tools

- Provisioning
- Resource Management
- Compiler Families
- MPI Families / Communication Libraries
- Development Tools
- Performance Analysis Tools
- IO Libraries
- Distro Packages
- Runtimes
- Serial/Threaded Libraries
- Parallel Libraries

E.2.1 Administrative Tools

RPM Package Name	Version	Info/URL
conman	0.3.1	ConMan: The Console Manager. http://dun.github.io/conman
docs	4.0.0	OpenHPC documentation. https://github.com/openhpc/ohpc
docs	4.2.0	OpenHPC documentation. https://github.com/openhpc/ohpc
environment-modules	5.7.0	Provides dynamic modification of a user's environment. https://envmodules.io
examples	2.0	Example source code and templates for use within OpenHPC environment. https://github.com/openhpc/ohpc
genders	1.32	Static cluster configuration database. https://github.com/chaos/genders
hpc-workspace	1.6.3	Temporary workspace management. https://github.com/holgerBerger/hpc-workspace
lmod-defaults	2.0	OpenHPC default login environments. https://github.com/openhpc/ohpc
lmod	9.4.5	Lua based Modules (lmod). https://github.com/TACC/Lmod
losf	0.56.0	A Linux operating system framework for managing HPC clusters. https://github.com/hpcsi/losf
modules-defaults	2.1	OpenHPC default login environments. https://github.com/openhpc/ohpc
nhc	1.4.3	LBNL Node Health Check. https://github.com/mej/nhc
ohpc-release	4	OpenHPC release files. https://github.com/openhpc/ohpc
pdsh	2.36	Parallel remote shell program. https://github.com/chaos/pdsh

RPM Package Name	Version	Info/URL
prun	3	Convenience utility for parallel job launch. https://github.com/openhpc/ohpc
test-suite	4.2.0	Integration test suite for OpenHPC. https://github.com/openhpc/ohpc

E.2.2 Provisioning

RPM Package Name	Version	Info/URL
warewulf	4.7.1	A provisioning system for large clusters of bare metal and/or virtual systems. https://github.com/warewulf/warewulf
warewulf-ohpc-dracut	4.7.1	dracut module for loading a Warewulf image. https://github.com/warewulf/warewulf
warewulf-ohpc-sos	4.7.1	sos plugin for Warewulf. https://github.com/warewulf/warewulf

E.2.3 Resource Management

RPM Package Name	Version	Info/URL
flux-core	0.89.0	Core components of the Flux resource manager framework. https://github.com/flux-framework/flux-core
flux	0.7.1	PMIx support for the Flux resource manager. https://github.com/flux-framework/flux-pmix
flux-sched	0.55.0	Fluxion graph-based scheduler for the Flux resource manager. https://github.com/flux-framework/flux-sched
flux-security	0.15.0	Independent security component for the Flux resource manager. https://github.com/flux-framework/flux-security
magpie	3.2	Scripts for running Big Data software in HPC environments. https://github.com/LLNL/magpie
openpbs-client	23.06.06	OpenPBS for a client host. http://www.openpbs.org
openpbs-execution	23.06.06	OpenPBS for an execution host. http://www.openpbs.org
openpbs-server	23.06.06	OpenPBS for a server host. http://www.openpbs.org
pmix	4.2.9	An extended/exascale implementation of PMI. https://pmix.org
slurm-contribs	26.05.5	Perl tool to print Slurm job state information. https://slurm.schedmd.com
slurm-devel	26.05.5	Development package for Slurm. https://slurm.schedmd.com
slurm-example-configs	26.05.5	Example config files for Slurm. https://slurm.schedmd.com
slurm-libpmi	26.05.5	Slurm's implementation of the pmi libraries. https://slurm.schedmd.com
slurm	26.05.5	Slurm Workload Manager. https://slurm.schedmd.com

RPM Package Name	Version	Info/URL
slurm-openlava	26.05.5	openlava/LSF wrappers for transition from OpenLava/LSF to Slurm. https://slurm.schedmd.com
slurm-pam_slurm	26.05.5	PAM module for restricting access to compute nodes via Slurm. https://slurm.schedmd.com
slurm-perlapi	26.05.5	Perl API to Slurm. https://slurm.schedmd.com
slurm-sackd	26.05.5	Slurm authentication daemon. https://slurm.schedmd.com
slurm-slurmctld	26.05.5	Slurm controller daemon. https://slurm.schedmd.com
slurm-slurmd	26.05.5	Slurm compute node daemon. https://slurm.schedmd.com
slurm-slurmdbd	26.05.5	Slurm database daemon. https://slurm.schedmd.com
slurm-sview	26.05.5	Graphical user interface to view and modify Slurm state. https://slurm.schedmd.com
slurm-torque	26.05.5	Torque/PBS wrappers for transition from Torque/PBS to Slurm. https://slurm.schedmd.com

E.2.4 Compiler Families

RPM Package Name	Version	Info/URL
gnu15-compilers	15.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org
gnu16-compilers	16.2.0	The GNU C Compiler and Support Files. http://gcc.gnu.org

E.2.5 MPI Families / Communication Libraries

RPM Package Name	Version	Info/URL
mpich	5.0.1	MPICH MPI implementation. http://www.mpich.org
openmpi5	5.0.10	A powerful implementation of MPI/SHMEM. http://www.open-mpi.org
ucx	1.22.0	UCX is a communication library implementing high-performance messaging. http://www.openucx.org

E.2.6 Development Tools

RPM Package Name	Version	Info/URL
EasyBuild	5.4.1	Software build and installation framework. https://easybuild.io
autoconf	2.71	A GNU tool for automatically configuring source code. http://www.gnu.org/software/autoconf
automake	1.16.5	A GNU tool for automatically creating Makefiles. http://www.gnu.org/software/automake

RPM Package Name	Version	Info/URL
cmake	4.4.4	CMake is an open-source, cross-platform family of tools designed to build, test and package software. https://cmake.org
cuda-devel	25.9	OpenHPC compatibility package for cuda. https://github.com/openhpc/ohpc
cuda-repo	25.9	Cuda toolkit online repository. https://github.com/openhpc/ohpc
hwloc	2.15.0	Portable Hardware Locality. http://www.open-mpi.org/projects/hwloc
libtool	2.4.6	The GNU Portable Library Tool. http://www.gnu.org/software/libtool
python3-mpi4py	4.1.1	Python bindings for the Message Passing Interface (MPI) standard. https://github.com/mpi4py/mpi4py
python3-numpy	2.4.4	NumPy array processing for numbers, strings, records and objects. https://github.com/numpy/numpy
spack	1.2.2	HPC software package management. https://github.com/spack/spack
valgrind	3.27.1	Valgrind Memory Debugger. http://www.valgrind.org

E.2.7 Performance Analysis Tools

RPM Package Name	Version	Info/URL
dimemas	5.5.0	Dimemas tool. https://tools.bsc.es
extrae	5.0.4	Extrae tool. https://tools.bsc.es
imb	2021.11	Intel MPI Benchmarks (IMB). https://software.intel.com/en-us/articles/intel-mpi-benchmarks
likwid	5.5.1	Performance tools for the Linux console. https://github.com/RRZE-HPC/likwid
omb	7.5.2	OSU Micro-benchmarks. https://mvapich.cse.ohio-state.edu/benchmarks
papi	7.2.0	Performance Application Programming Interface. http://icl.cs.utk.edu/papi
paraver	4.12.0	Paraver. https://tools.bsc.es
pdtoolkit	3.25.1	PDT is a framework for analyzing source code. http://www.cs.uoregon.edu/Research/pdt
scalasca	2.6.2	Toolset for performance analysis of large-scale parallel applications. http://www.scalasca.org
scorep	9.4	Scalable Performance Measurement Infrastructure for Parallel Codes. http://www.vi-hps.org/projects/score-p
tau	2.35.1	Tuning and Analysis Utilities Profiling Package. http://www.cs.uoregon.edu/research/tau/home.php

E.2.8 IO Libraries

RPM Package Name	Version	Info/URL
adios2	2.12.0	The Adaptable IO System v2 (ADIOS2). https://adios2.readthedocs.io/en/latest/index.html
cubew	4.9.1	CUBE Uniform Behavioral Encoding generic presentation writer component. http://www.scalasca.org/software/cube-4.x/download.html
hdf5	2.1.1	A general purpose library and file format for storing scientific data. http://www.hdfgroup.org/HDF5
netcdf-cxx	4.3.1	C++ Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf-fortran	4.6.2	Fortran Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
netcdf	4.10.0	C Libraries for the Unidata network Common Data Form. http://www.unidata.ucar.edu/software/netcdf
otf2	3.1.1	Open Trace Format 2 library. http://score-p.org
phdf5	2.1.1	A general purpose library and file format for storing scientific data (parallel version). http://www.hdfgroup.org/HDF5
pnetcdf	1.14.1	A Parallel NetCDF library (PnetCDF). http://cucis.ece.northwestern.edu/projects/PnetCDF
sionlib	1.7.7	Scalable I/O Library for Parallel Access to Task-Local Files. https://apps.fz-juelich.de/jsc/sionlib/docu/index.html

E.2.9 Distro Packages

RPM Package Name	Version	Info/URL
python3-Cython	3.3.0	The Cython compiler for writing C extensions for the Python language. http://www.cython.org

E.2.10 Runtimes

RPM Package Name	Version	Info/URL
charliecloud	0.45.2	Lightweight user-defined software stacks for high-performance computing. https://charliecloud.io

E.2.11 Serial/Threaded Libraries

RPM Package Name	Version	Info/URL
R	4.5.3	R is a language and environment for statistical computing and graphics (S-Plus like). http://www.r-project.org
cubelib	4.9.1	CUBE Uniform Behavioral Encoding generic presentation library component. http://www.scalasca.org/software/cube-4.x/download.html

RPM Package Name	Version	Info/URL
gotcha	1.0.8	A library for wrapping function calls to shared libraries. https://github.com/llnl/gotcha
gsl	2.8	GNU Scientific Library (GSL). http://www.gnu.org/software/gsl
metis	5.1.0	Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
opari2	2.0.9	An OpenMP runtime performance measurement instrumenter. https://www.vi-hps.org/projects/score-p
openblas	0.3.32	An optimized BLAS library based on GotoBLAS2. http://www.openblas.net
plasma	25.5.27	Parallel Linear Algebra Software for Multicore Architectures. https://github.com/icl-utk-edu/plasma
scotch	7.0.11	Graph, mesh and hypergraph partitioning library. https://gitlab.inria.fr/scotch/scotch
superlu	7.0.1	A general purpose library for the direct solution of linear equations. http://crd.lbl.gov/~xiaoye/SuperLU

E.2.12 Parallel Libraries

RPM Package Name	Version	Info/URL
boost	1.90.0	Free peer-reviewed portable C++ source libraries. http://www.boost.org
fftw	3.3.10	A Fast Fourier Transform library. http://www.fftw.org
hypre	3.1.0	Scalable algorithms for solving linear systems of equations. http://www.llnl.gov/casc/hypre
mfem	4.9	Lightweight, general, scalable C++ library for finite element methods. http://mfem.org
mumps	5.8.2	A MUltifrontal Massively Parallel Sparse direct Solver. https://mumps-solver.org
petsc	3.25.0	Portable Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc
ptscotch	7.0.11	Graph, mesh and hypergraph partitioning library using MPI. https://gitlab.inria.fr/scotch/scotch
scalapack	2.2.3	A subset of LAPACK routines redesigned for heterogeneous computing. https://netlib.org/scalapack
slepc	3.25.0	A library for solving large scale sparse eigenvalue problems. http://slepc.upv.es
superlu_dist	9.2.1	A general purpose library for the direct solution of linear equations. https://portal.nersc.gov/project/sparse/superlu
trilinos	17.0.0	A collection of libraries of numerical algorithms. https://trilinos.org

F Package Signatures

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

```
Fingerprint: 5E33 3CA3 A1BD BBC9 DF14 9D74 09AD FAE4 **D722A692**
```

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.

```
rpm --checksig -v ohpc-release-4-1.el10.x86_64.rpm
ohpc-release-4-1.el10.x86_64.rpm:
    Header V3 RSA/SHA256 Signature, key ID d722a692: OK
    Header SHA256 digest: OK
    Header SHA1 digest: OK
    Payload SHA256 digest: OK
    V3 RSA/SHA256 Signature, key ID d722a692: OK
    MD5 digest: OK
```