



OpenHPC: Community Building Blocks for HPC Systems

Karl W. Schulz, Ph.D.
Enterprise High Performance Computing Group, Intel

FOSDEM'16, HPC, Big Data and Data Science Devroom
Brussels, Belgium ♦ January 31, 2016

Outline

- Motivation for this new community effort
- Community vision and participants
- Current components overview - hierarchical approach
- Development Infrastructure
 - packaging conventions
 - build system
 - documentation
- Integration Testing
- Participation/Future timeline

Motivation for Community Effort

Many sites spend considerable effort aggregating a large suite of open-source projects to provide a capable HPC environment for their users:

- necessary to obtain HPC focused packages that are either absent or do not keep pace from Linux distro providers
- local packaging or customization frequently tries to give software versioning access to users (e.g. via modules or similar equivalent)

They frequently leverage a mix of external and in-house tools for:

- provisioning, software installations/upgrades, config management schemes, and system diagnostics mechanisms.
- although the functionality is similar, the implementations across sites is often different which can lead to duplication of effort

On the developer side, many successful projects must engage in continual triage and debugging regarding configuration and installation issues on HPC systems

Motivation for Community Effort (cont.)

Q: Can we provide community value by focusing first on one of the least common denominators?
--> *the configuration and packaging of HPC software*

- Focus on the common packages that have significant usage overlap
 - establish some basic packaging guidelines
 - embed hierarchal support (to deal with development toolchain permutations)
- Define binary distribution and upgrade mechanisms
 - leverage significant experience from the distro communities
 - include dependency resolution
- Do not be prescriptive on choice of config management or architecture
 - just like users have their favorite editors (mine is of course emacs, the one to rule them all), many sites have their favorite config management system and administration policies in place
 - instead, foster community with example installation recipes
 - include validation efforts on reference systems

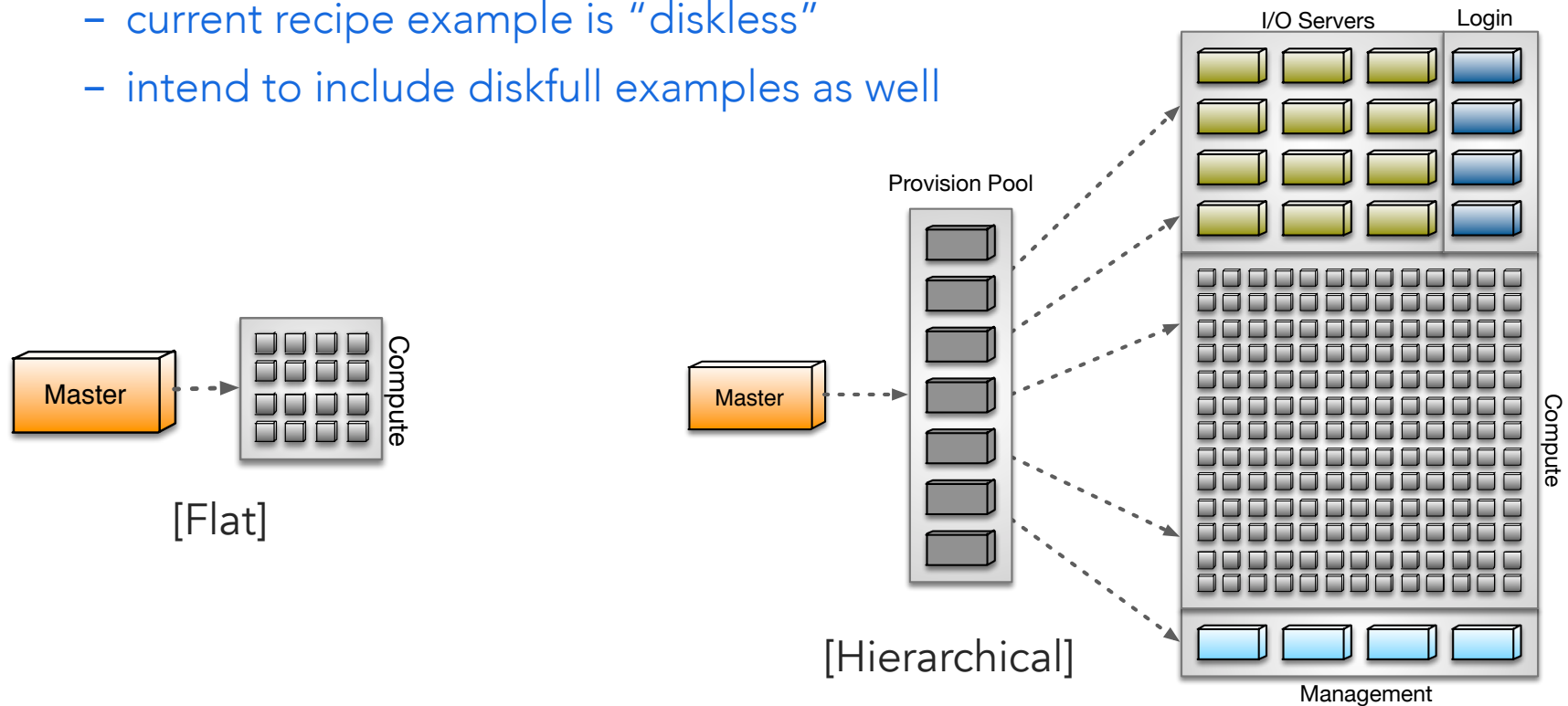
Assuming success above, expand further over time:

- explore definition of interface points between key software components
- what functional areas make sense to abstract for more seamless component swapping?
- inclusion of future research efforts (e.g. new runtime systems, programming models)

System Architecture

Intention over time is to provide reference design(s) with use cases for big and small system designs:

- usual designation of node types by function
- current recipe example is "diskless"
- intend to include diskfull examples as well



OpenHPC: Early Community Members



OEMs



HPC Sites

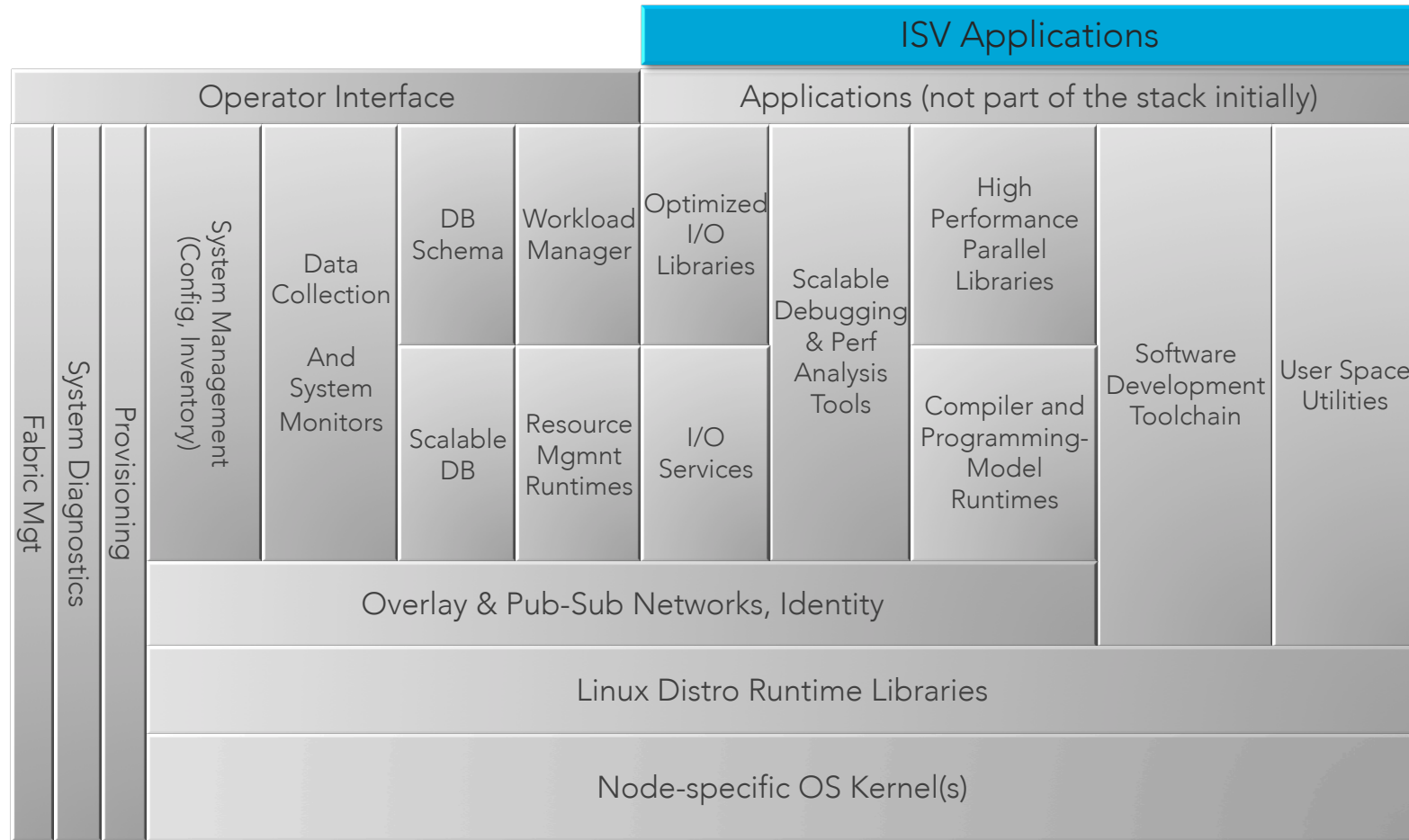


ISV-OSVs



Component Overview

SW Block Diagram of Typical HPC System



OpenHPC 1.0 - Initial starting components

Functional Areas	Components
Base OS	CentOS 7.1
Administrative Tools	Conman, Ganglia, Intel Cluster Checker**, Lmod, LosF, Nagios, pdsh, prun, EasyBuild
Provisioning	Warewulf
Resource Mgmt.	SLURM, Munge
I/O Services	Lustre client (community version)
Numerical/ Scientific Libraries	Boost, GSL, FFTW, Metis, PETSc, Trilinos, Hypre, SuperLU, Mumps, Intel MKL**
I/O Libraries	HDF5 (pHDF5), NetCDF (including C++ and Fortran interfaces), Adios
Compiler Families	GNU (gcc, g++, gfortran), Intel Parallel Studio XE (icc, icpc, ifort)**
MPI Families	MVAPICH2, OpenMPI, Intel MPI**
Development Tools	Autotools (autoconf, automake, libtool), Valgrind, R, SciPy/NumPy, Intel Inspector**
Performance Tools	PAPI, IMB, mpiP, pdtoolkit TAU, Intel Advisor**, Trace Analyzer and Collector**, Vtune Amplifier**

Notes:

- Additional dependencies that are not provided by the BaseOS or community repos (e.g. EPEL) are also included
- 3rd Party libraries are built for each compiler/MPI family (6 combinations typically)
- Resulting repository currently comprised of ~250 RPMs

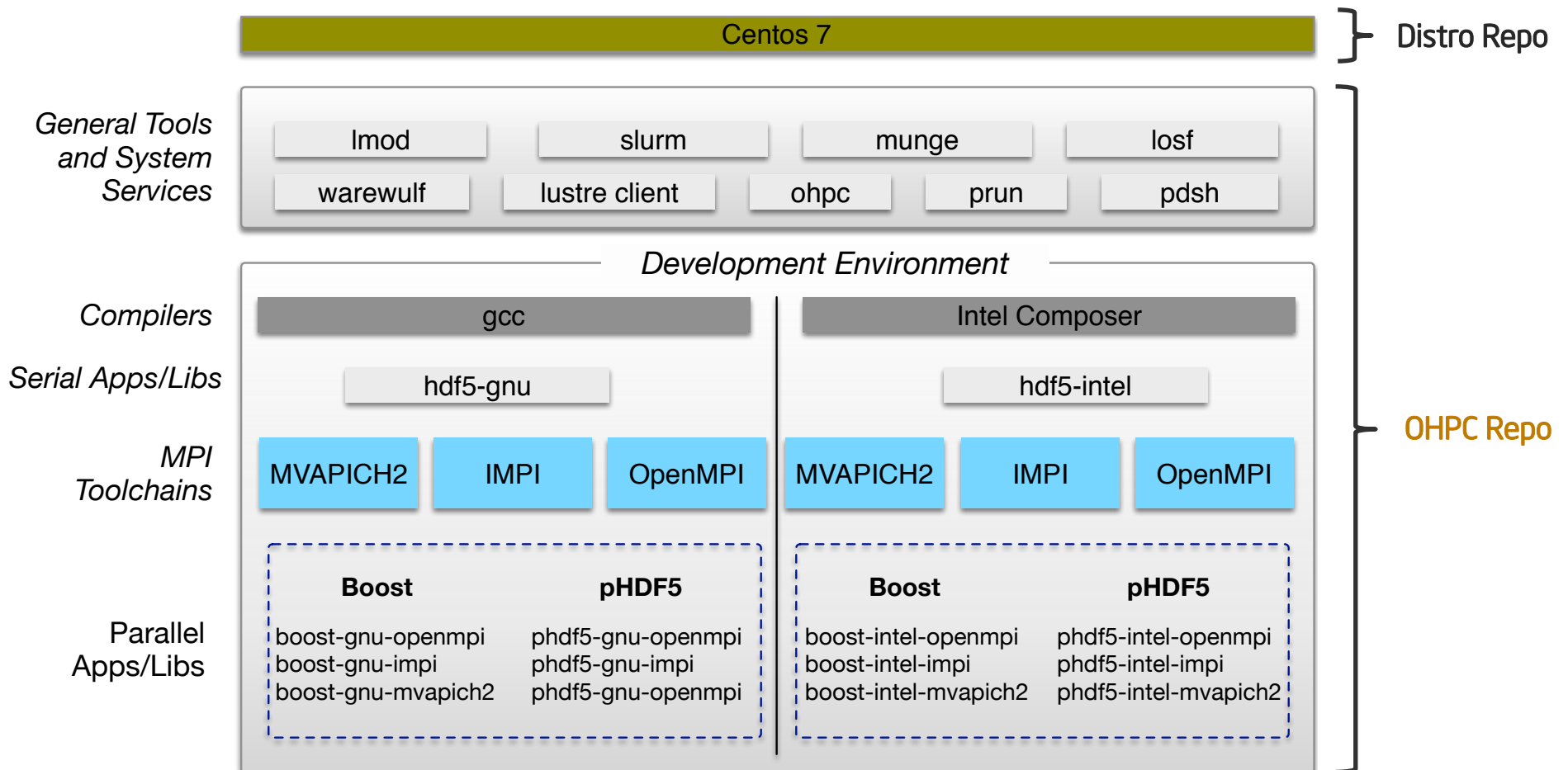
** Bring your own license model

OpenHPC++ - Potential future efforts

Functional Areas	Components	Contributions by:
Base OS	CentOS 7.1, McKernel , Kitten , mOS	RIKEN , Sandia , Intel
AdminTools	Conman, Ganglia, Intel Cluster Checker**, Lmod, LosF, Nagios, pdsh, prun, EasyBuild, ORCM	Intel
Provisioning	Warewulf, xCAT	Community
Resource Mgmt.	SLURM, Munge, ParaStation mgmt , PMIx , PBS Pro	ParTec , community , Altair
Cross Cutting	OpenStack HPC suitable components	Cray
Runtimes	OpenMP, OmpSs , OCR	BSC , Intel
I/O Services	Lustre client (community version)	
Numerical/ Scientific Libs	Boost, GSL, FFTW, Metis, PETSc, Trilinos, Hypr, SuperLU, Mumps, Intel MKL**	
I/O Libraries	HDF5 (pHDF5), NetCDF (including C++ and Fortran interfaces), Adios	
Compiler Families	GNU (gcc, g++, gfortran), Intel Parallel Studio XE (icc, icpc, ifort)**	
MPI Families	MVAPICH2, OpenMPI, Intel MPI**, MPICH , ParaStation MPI	Argonne , ParTec
Development Tools	Autotools (autoconf, automake, libtool), Valgrind, R, SciPy/NumPy, Intel Inspector **	
Performance Tools	PAPI, Intel IMB, mpiP, pdtoolkit TAU, Intel Advisor**, Intel Trace Analyzer and Collector**, Intel Vtune Amplifier**, Paraver , Scalasca	BSC , Jülich

** Bring your own license model

Hierarchical Overlay for OpenHPC software



Hierarchical Software - the user experience

- The compiler and MPI stack component hierarchy is reflected in the user environment
- End user sees compatible software build based on the currently loaded environment
- If the user switches between compiler families (or MPI families), environment updates automatically:
 - supported via modules (Lmod implementation - built-in notion of “families”)
 - module configuration built into RPM packaging

```
$ module list
Currently Loaded Modules:
  1) intel/16.0.0.069  2) openmpi/1.8.4  3) ohpc  4) boost/1.58.0

$ module avail
----- /opt/ohpc/pub/moduledeps/gnu-openmpi -----
  boost/1.58.0  phdf5/1.8.14
----- /opt/ohpc/pub/moduledeps/intel-----
  hdf5/1.8.14  impi/5.1.0.069  mvapich2/2.1  openmpi/1.8.4
----- /opt/ohpc/pub/modulefiles -----
  autotools  fsp  gnu/4.9.2  intel/16.0.0.69

$ echo $BOOST_LIB
/opt/ohpc/pub/libs/intel/openmpi/boost/1.58.0/lib

$ module swap intel gnu
Due to MODULEPATH changes the following have been reloaded:
  1) boost/1.58.0  2) openmpi/1.8.4

$ echo $BOOST_LIB
/opt/ohpc/pub/libs/gnu/openmpi/boost/1.58.0/lib
```

Development Infrastructure

OpenHPC Development Infrastructure

What are we using to get the job done....?

The usual software engineering stuff:

- GitHub (SCM and issue tracking/planning)
- Continuous Integration (CI) Testing (Jenkins)
- Documentation (Latex)

Capable build/packaging system

- We are targeting a common delivery/access mechanism that adopts Linux sysadmin familiarity - ie. **yum/zypper** repositories for supported distros
 - ultimately delivering RPMs
 - [base] + [update] repositories to support life-cycle management
- Require flexible system to manage builds for multiple distros, multiple compiler/MPI family combinations, and dependencies across packages
- Have engineered a system using Open Build Service (OBS) which is supported by back-end git
 - git houses .spec files, tarballs, and custom patches
 - OBS performs automated builds (more on that to come)



<https://github.com/openhpc/ohpc>



<https://build.openhpc.community>



LATEX

Build System - OBS

<https://build.openhpc.community>

OpenHPC Build Service Log In

openHPC

Welcome to the OpenHPC build service and community package repository. This community build infrastructure uses the [Open Build Service](#) to automate the build and release of a variety of RPMs under the auspices of the OpenHPC project. When combined with a matching base OS install, the collection of assembled tools and development packages can be used to deploy HPC Linux clusters. Additional information regarding the project can be found at:

- [openhpc.community](#) (General information)
- [GitHub](#) (Developer resources)
- [Mailing Lists](#) (Support)

The Open Build Service (OBS)

The [Open Build Service \(OBS\)](#) is an open and complete distribution development platform that provides a transparent infrastructure for development of Linux distributions, used by openSUSE, MeeGo and other distributions. It also supports Fedora, Debian, Ubuntu, RedHat and other Linux distributions.

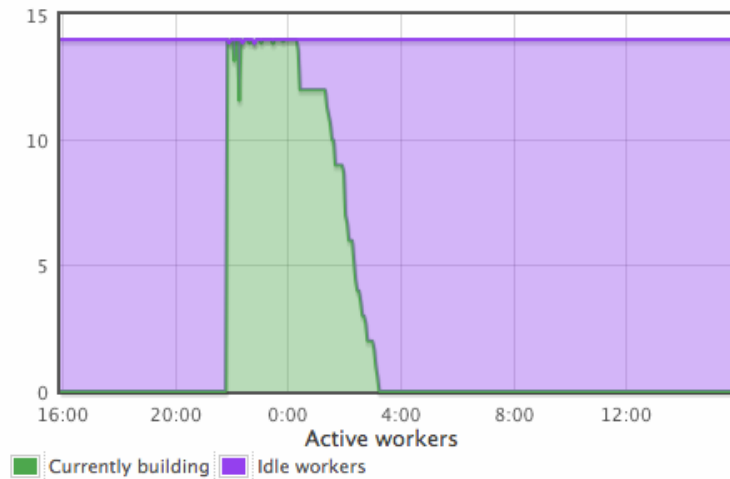
The OBS is developed under the umbrella of the [openSUSE project](#). Please find further informations on the [openSUSE Project wiki pages](#).

[All Projects](#) [Search](#) [Status Monitor](#)

OpenHPC:1.0:Factory	1 day ago
warewulf-ipmi	1 day ago
warewulf-provision	1 day ago
warewulf-vnfs	1 day ago
warewulf-nhc	1 day ago
valgrind	1 day ago

- Using the **Open Build Service (OBS)** to manage build process (end-to-end)
- OBS can drive builds for multiple repositories
- Builds carried out in chroot environment
- Generates binary and src rpms
- Publishes/maintains corresponding package repository

Build System - OBS



- OBS manages dependency resolution and rebuilds all downstream packages
- Community build infrastructure hosted in the cloud - can add additional build servers dynamically when needed
- Currently takes about 6 hours to build repository from scratch for CentOS7
- Biggest contributor is Trilinos

- Leveraging ability within OBS to link related packages
- Convenient for packages with **compiler** and **MPI** dependencies
- Single commit drives all package permutations
- OBS builds automatically triggered via git commit hooks

Snippets from METIS OBS config

```
$ ls metis-*
metis-gnu:
_service

metis-intel:
_link

$ cat metis-intel/_link
<link project='ForestPeak:15.31:Factory' package='metis-gnu'>
<patches>
  <topadd>%define compiler_family intel</topadd>
</patches>
</link>
EOF
```

Anatomy of a .spec file

- Motivation is to have single input to drive multiple output configurations
- Leverage variety of macros to aid in hierarchical builds

Install Path Macros

```
$ cat OHPC_macros
%define OHPC_BUILD 1
%define PROJ_NAME      ohpc
%define OHPC_HOME      /opt/{PROJ_NAME}
%define OHPC_ADMIN     %{OHPC_HOME}/admin
%define OHPC_PUB       %{OHPC_HOME}/pub
%define OHPC_COMPILERS %{OHPC_PUB}/compiler
%define OHPC_MPI_STACKS %{OHPC_PUB}/mpi
%define OHPC_APPS      %{OHPC_PUB}/apps
%define OHPC_LIBS      %{OHPC_PUB}/libs
%define OHPC_MODULES   %{OHPC_PUB}/modulefiles
%define OHPC_MODULEDEPS %{OHPC_PUB}/moduledeps
```

MPI macros

```
$ cat OHPC_setup_mpi
if [ -z "$OHPC_MPI_FAMILY" ]; then
    echo "Unknown OHPC_MPI_FAMILY"
    exit 1
fi

if [ "$OHPC_MPI_FAMILY" = "openmpi" ]; then
    module load openmpi
elif [ "$OHPC_MPI_FAMILY" = "impi" ]; then
    module load impi
elif [ "$OHPC_MPI_FAMILY" = "mvapich2" ]; then
    module load mvapich2
else
    echo "Unsupported OHPC_MPI_FAMILY -> $OHPC_MPI_FAMILY"
    exit 1
fi
```


Anatomy of a .spec file

Snippet from METIS .spec file

- Default family choice defined, but can be overridden
- Family dependencies embedded for package managers

```
%include %[_sourcedir]/OHPC_macros

#-ohpc-header-comp-begin-----

# OpenHPC convention: the default assumes the gnu compiler family;
# however, this can be overridden by specifying the compiler_family
# variable via rpmbuild or other mechanisms.

%{!?compiler_family: %define compiler_family gnu}

# Compiler dependencies
BuildRequires: lmod%{PROJ_DELIM} coreutils
%if %{compiler_family} == gnu
BuildRequires: gnu-compilers%{PROJ_DELIM}
Requires:      gnu-compilers%{PROJ_DELIM}
%endif
%if %{compiler_family} == intel
BuildRequires: gcc-c++ intel-compilers-devel%{PROJ_DELIM}
Requires:      gcc-c++ intel-compilers-devel%{PROJ_DELIM}
%if 0%{?OHPC_BUILD}
BuildRequires: intel_licenses
%endif
%endif

#-ohpc-header-comp-end-----
```

Anatomy of a .spec file

Snippet from METIS .spec file

- Hierarchical module files encoded within package build
- Provide consistent set of module environment variables
 - METIS_DIR
 - METIS_BIN
 - METIS_LIB
 - METIS_INC

```
# OpenHPC module file
%{__mkdir} -p %{buildroot}%{OHPC_MODULEDEPS}/%{compiler_family}/%{pname}
%{__cat} << EOF > %{buildroot}/%{OHPC_MODULEDEPS}/%{compiler_family}/%{pname}/%{version}
##Module1.0#####

proc ModulesHelp { } {

puts stderr " "
puts stderr "This module loads the %{PNAME} library built with the %{compiler_family}
compiler toolchain."
puts stderr "\nVersion %{version}\n"

}
module-whatis "Name: %{PNAME} built with %{compiler_family} toolchain"
module-whatis "Version: %{version}"
module-whatis "Category: runtime library"
module-whatis "Description: %{summary}"
module-whatis "%{url}"

set      version                %{version}

prepend-path  PATH                %{install_path}/bin
prepend-path  INCLUDE             %{install_path}/include
prepend-path  LD_LIBRARY_PATH     %{install_path}/lib

setenv       %{PNAME}_DIR        %{install_path}
setenv       %{PNAME}_BIN        %{install_path}/bin
setenv       %{PNAME}_LIB        %{install_path}/lib
setenv       %{PNAME}_INC        %{install_path}/include
EOF
```

Documentation Overview



OpenHPC (v1.0)
Cluster Building Recipes

CentOS7.1 Base OS
Base Linux Edition*

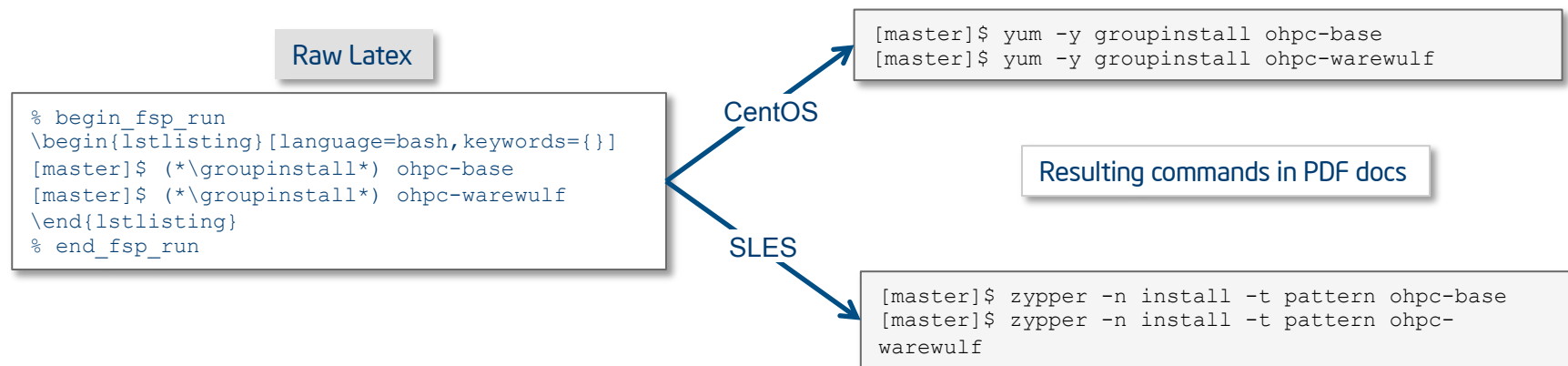
Document Last Update: 2015-11-12
Document Revision: bf8c471

Contents

1 Introduction	5
1.1 Target Audience	5
1.2 Requirements/Assumptions	5
1.3 Bring your own license	6
1.4 Inputs	6
2 Install Base Operating System (BOS)	7
3 Install OpenHPC Components	7
3.1 Enable OpenHPC repository for local use	7
3.2 Installation template	8
3.3 Add provisioning services on <i>master</i> node	8
3.4 Add resource management services on <i>master</i> node	9
3.5 Add InfiniBand support services on <i>master</i> node	9
3.6 Complete basic Warewulf setup for <i>master</i> node	9
3.7 Define <i>compute</i> image for provisioning	10
3.7.1 Build initial BOS image	10
3.7.2 Add OpenHPC components	11
3.7.3 Customize system configuration	11
3.7.4 Additional Customizations (<i>optional</i>)	12
3.7.4.1 Increase locked memory limits	12
3.7.4.2 Enable ssh control via resource manager	13
3.7.4.3 Add Cluster Checker	13
3.7.4.4 Add Lustre client	13
3.7.4.5 Add Nagios monitoring	14
3.7.4.6 Add Ganglia monitoring	14
3.7.4.7 Enable forwarding of system logs	15
3.7.5 Import files	15
3.8 Finalizing provisioning configuration	16
3.8.1 Assemble bootstrap image	16
3.8.2 Assemble Virtual Node File System (VNFS) image	16
3.8.3 Register nodes for provisioning	16
3.9 Boot compute nodes	17
4 Install OpenHPC Development Components	17
4.1 Development Tools	18
4.2 Compilers	18
4.3 Performance Tools	18
4.4 MPI Stacks	18
4.5 Setup default development environment	19
4.6 3rd Party Libraries and Tools	19
5 Resource Manager Startup	20
6 Run a Test Job	21
6.1 Interactive execution	21
6.2 Batch execution	22

Documentation Overview

- We all know documentation can be an afterthought and is difficult to validate
- System install is critical and the first thing an end-user will experience (good or bad)
 - consequently, approach taken is to treat documentation as a first-class verification citizen
 - difficult to do this with something like MS Word
 - more approachable with a typesetting documentation format like Latex which is what we have adopted to date
- Endeavor to minimize repetition (for example, across distros)
 - package manager commands abstracted so that single input can support multiple distros



Basic Cluster Install Example

- Starting install guide/ recipe targeted for flat hierarchy
- Leverages image-based provisioner (Warewulf)
 - PXE boot (stateless)
 - optionally connect external Lustre file system
- Obviously need hardware-specific information to support (remote) bare-metal provisioning

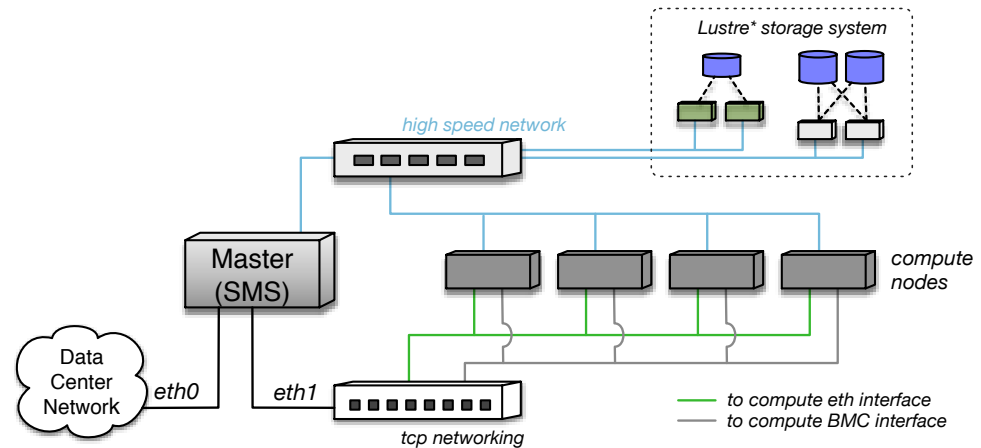


Figure 1: Overview of physical cluster architecture.

```

• ${ohpc_repo} # OpenHPC repo location
• ${sms_name} # Hostname for SMS server
• ${sms_ip} # Internal IP address on SMS server
• ${sms_eth_internal} # Internal Ethernet interface on SMS
• ${eth_provision} # Provisioning interface for computes
• ${internal_netmask} # Subnet netmask for internal network
• ${ntp_server} # Local ntp server for time synchronization
• ${bmc_username} # BMC username for use by IPMI
• ${bmc_password} # BMC password for use by IPMI
• ${c_ip[0]}, ${c_ip[1]}, ... # Desired compute node addresses
• ${c_bmc[0]}, ${c_bmc[1]}, ... # BMC addresses for computes
• ${c_mac[0]}, ${c_mac[1]}, ... # MAC addresses for computes
• ${compute_regex} # Regex for matching compute node names (e.g. c*)

Optional:
• ${mgs_fs_name} # Lustre MGS mount name
• ${sms_ipoib} # IPoIB address for SMS server
• ${ipoib_netmask} # Subnet netmask for internal IPoIB
• ${c_ipoib[0]}, ${c_ipoib[1]}, ... # IPoIB addresses for computes
    
```

Basic Cluster Install: SMS

- Example recipe assumes base operating system is first installed on chosen master (SMS) host
- Then, enable OpenHPC repo and install desired components
- Convenience aliases are provided to group related functionality

```
[sms]# yum -y groupinstall ohpc-base  
[sms]# yum -y groupinstall ohpc-warewulf
```

Add provisioning components

```
[sms]# yum -y groupinstall ohpc-slurm-server
```

Add SLURM server components

```
[sms]# cp /opt/ohpc/pub/examples/network/centos/ifcfg-ib0 /etc/sysconfig/network-scripts  
  
# Define local IPoIB address and netmask  
[sms]# perl -pi -e "s/master_ipoib/${sms_ipoib}/" /etc/sysconfig/network-scripts/ifcfg-ib0  
[sms]# perl -pi -e "s/ipoib_netmask/${ipoib_netmask}/" /etc/sysconfig/network-scripts/ifcfg-ib0  
  
# Initiate ib0  
[sms]# ifup ib0
```

Example optional configuration enabling IPoIB (e.g. to support Lustre over IB)

Basic Cluster Install Example

- Recipe guides necessarily have a number of things to “cut-and-paste” if you want to reproduce them
- Recall that we want to use the documentation during the validation process
 - Cull out relevant commands automatically for use during CI testing
 - Seemed reasonable to make available as a script, so there is a template starting script available with the documentation RPM, can be used for local installation and customization

1. Install the `docs-ohpc` package

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

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

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

3. Update `input.local` with desired settings

4. Copy the template installation script which contains command-line instructions culled from this guide.

```
[sms]# cp -p /opt/ohpc/pub/doc/recipes/vanilla/recipe.sh .
```

5. Review and edit `recipe.sh` to suite.

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

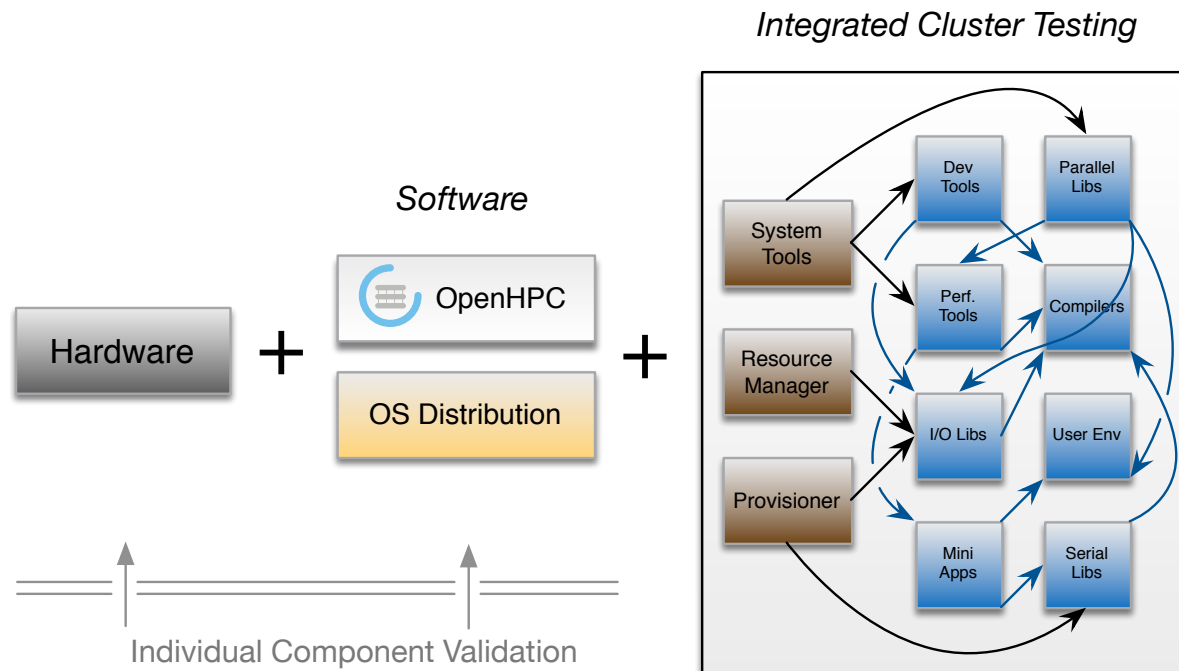
```
[sms]# export OHPC_INPUT_LOCAL=./input.local  
[sms]# ./recipe.sh
```

Integration Testing

Integration/Test/Validation

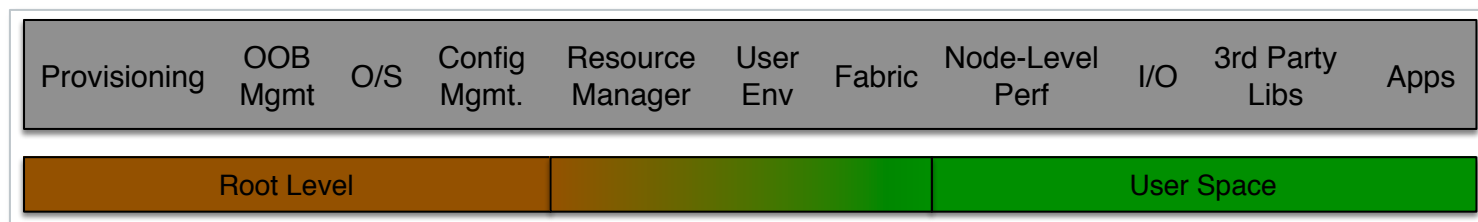
Testing intent is to build upon existing validation efforts and augment component-level validation with targeted cluster-validation and scaling initiatives including:

- install recipes
- cross-package interaction
- development environment
- scale/performance testing
- mimic use cases common in HPC deployments
- upgrade mechanism



Integration/Test/Validation

- To facilitate global efforts in diagnostics/validation, we have devised a standalone integration test infrastructure
- Intent was to create families of tests that could be used during:
 - initial install process (can we build a cluster?)
 - post-install process (does it work?)
 - developing tests that touch all of the major components (can we compile against 3rd party libraries, will they execute under resource manager, etc)
- Requires both root-level and user-level tests:
 - Includes build and test of benchmark kernels that are not part of install RPMs
 - miniFE
 - miniDFT
 - HPCG
 - Parallel research kernels (PRK)



Post Install Integration Tests - Overview

Global testing harness includes a number of embedded subcomponents:

- harness is autotools based
- major components have configuration options to enable/disable
- this example shows root-level tests

Example ./configure output

```
----- SUMMARY -----
Package version..... : cmt-0.11.0
Build user..... : root
Build host..... : master4-centos71.localdomain
Configure date..... : 2015-07-06 06:39
Build architecture..... : x86_64-unknown-linux-gnu

Submodule Configuration:
  Base operating system..... : enabled
  Out of band tools..... : enabled
  Hardware benchmarks..... : enabled
  Cluster checker..... : enabled
  Lustre client..... : enabled
  ORCM data collection..... : enabled
```

Root

Post Install Integration Tests - Overview

Example ./configure output

Non-root

This example highlights user-level tests:

- end user tests need to touch all of the supported compiler and MPI families -
- we abstract this to repeat the tests with different compiler/MPI environments:
 - gcc/Intel compiler toolchains
 - Intel, OpenMPI, MVAPICH2 MPI families

```
Package version..... : cmt-0.11.0
Build user..... : kwschulz
Build host..... : master4-centos71.localdomain

Submodule Configuration:

User Environment:
  RMS test harness..... : enabled
  Munge..... : enabled
  Apps..... : enabled
  Compilers..... : enabled
  MPI..... : enabled
  HSN..... : enabled
  Modules..... : enabled
  OOM..... : enabled
Dev Tools:
  Intel Inspector..... : enabled
  Valgrind..... : enabled
  R base package..... : enabled
  TBB..... : enabled
  CILK..... : enabled
Performance Tools:
  mpiP Profiler..... : enabled
  Intel Advisor XE..... : enabled
  Intel Trace Analyzer..... : enabled
  Intel Vtune Amplifier..... : enabled
  Papi..... : enabled
  PETSc..... : enabled
  TAU..... : enabled
Libraries:
  Boost ..... : enabled
  Boost MPI..... : enabled
  FFTW..... : enabled
  GSL..... : enabled
  HDF5..... : enabled
  HYPRE..... : enabled
  IMB..... : enabled
  Metis..... : enabled
  MUMPS..... : enabled
  NetCDF..... : enabled
  Numpy..... : enabled
  PHDF5..... : enabled
  Scipy..... : enabled
  Trilinos ..... : enabled
```

Integration Tests - Let's see one submodule test in action

Lmod user environment

- These are examples that primarily test interactive commands
- We are using the Bash Automated Testing System (Bats) for these tests
 - a TAP-complaint framework for Bash
 - available on GitHub
- We have extended Bats to:
 - create Junit output for parsing into Jenkins CI environment
 - capture execution runtimes


```
./interactive_commands
✓ [modules] module purge
✓ [modules] module list
✓ [modules] module help
✓ [modules] module load/unload
✓ [modules] module whatis
✓ [modules] module swap
✓ [modules] path updated

7 tests, 0 failures

$ ./rm_execution
✓ [modules] env variable passes through (slurm)
✓ [modules] loaded module passes through (slurm)
✓ [module] module commands available (slurm)
✓ [module] module load propagates thru RMS (slurm)

4 tests, 0 failures
```

Lmod submodule

Test Result : modules  **Jenkins**

0 failures (±0)

11 tests (±0)
Took 8.9 sec.

All Tests

Test name	Duration	Status
[module] module commands available (slurm)	0.49 sec	Passed
[module] module load propagates thru RMS (slurm)	0.62 sec	Passed
[modules] env variable passes through (slurm)	0.44 sec	Passed
[modules] loaded module passes through (slurm)	0.54 sec	Passed
[modules] module help	0.47 sec	Passed
[modules] module list	0.37 sec	Passed
[modules] module load/unload	2.9 sec	Passed
[modules] module purge	0.14 sec	Passed
[modules] module swap	0.94 sec	Passed
[modules] module whatis	0.46 sec	Passed
[modules] path updated	1.4 sec	Passed

Snapshot of Complete Integration Suite

Currently running in excess of 1,700 tests within CI environment

Test Result

0 failures (± 0)

1,706 tests (± 0)
Took 1 hr 10 min.

All Tests

Package \uparrow	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
UserLevelTests	1 hr 10 min	0		0		1669		1669	
RootLevelTests	0 ms	0		0		37		37	

Tested on Mellanox FDR and True Scale

Quick summary on current release

1.0 Release	
# of defined convenience groups	27
# of provided RPMs in Repo	255
Repo size	2.5 G
Size of example compute image	~300 MB
Integration test runtime (4-node cluster)	~1 hr 20 min
# of root-level integration tests	37
# of user-level integration tests	1,765

- Community collateral
 - [Info Site](http://openhpc.community): <http://openhpc.community>
 - [GitHub](https://github.com/openhpc/ohpc): <https://github.com/openhpc/ohpc>
 - [Build](https://build.openhpc.community): <https://build.openhpc.community>
 - [Yum Repo](http://build.openhpc.community/OpenHPC:/1.0/CentOS_7.1/OpenHPC:1.0.repo): http://build.openhpc.community/OpenHPC:/1.0/CentOS_7.1/OpenHPC:1.0.repo
 - *Future: CI visibility*
- Participation
 - mailing lists for questions, developer interactions and announcements: <http://openhpc.community/support/mail-lists/>
 - suggest additional components for selection; better yet, help with integrating and testing new components
 - share site knowledge through development of customized usage/install recipes
 - host/contribute test infrastructure



Thanks for your Time - Questions?

karl.w.schulz@intel.com

<http://openhpc.community>

<https://github.com/openhpc/ohpc>

<https://build.openhpc.community> (repo)