Modules & Application Software

The number of software packages that are installed together with the operating system on cluster nodes is kept light on purpose. Additional packages and applications are provided by a module system, which enables you to easily customise your working environment on the cluster. This module system is called Lmod¹⁾. Furthermore, we can provide different versions of the software which you can use on demand. Loading a module, software specific settings are applied, e.g. changing environment variables like PATH, LD_LIBRARY_PATH and MANPATH.

Alternatively, you can manage software packages on the cluster yourself by building software from source, by means of EasyBuild or by using Singularity containers. Python packages can also be installed using the Conda manager. The first three possibilities, in addition to the module system, are described in the current section, whereas Conda usage in the cluster is explained in this section.

We have adopted a systematic software naming and versioning convention in conjunction with the software installation system EasyBuild 2).

Software installation on the cluster utilizes a hierarchical software module naming scheme. This means that the command module avail does not display all installed software modules right away. Instead, only the modules that are immediately available for loading are displayed. More modules become available after their prerequisite modules are loaded. Specifically, loading a compiler module or MPI implementation module will make available all the software built with those applications. This way, he hope the prerequisites for certain software become apparent.

At the top level of the module hierarchy, there are modules for compilers, toolchains and software applications that come as a binary and thus do not depend on compilers. Toolchain modules organize compilers, MPI implementations and numerical libraries. Currently the following toolchain modules are available:

- Compiler only toolchains
 - GCC: GCC updated binutils
 - iccifort: Intel compilers, GCC
- Compiler & MPI toolchains
 - ∘ gompi: GCC, OpenMPI
 - ∘ iimpi: iccifort, Intel MPI
 - ∘ iompi: iccifort, OpenMPI
- Compiler & MPI & numerical libraries toolchains
 - ∘ foss: gompi, OpenBLAS, FFTW, ScaLAPACK
 - ∘ intel: iimpi, Intel MKL
 - ∘ iomkl: iompi, Intel MKL

Note that Intel compilers newer than 2020.x are provided by the toolchain module intel-compilers. It is strongly recommended to use this module as after 2023 the intel compiler modules iccifort will be removed.

Working with modules

This section explains how to use software modules.

List the entire list of possible modules

```
module spider
```

The same in a more compact list

```
module -t spider
```

Search for specific modules that have "string" in their name

```
module spider string
```

Detailed information about a particular version of a module (including instructions on how to load the module)

```
module spider name/version
```

Searches for all module names and descriptions that contain the specified string

```
module key string
```

List modules immediately available to load

```
module avail
```

Some software modules are hidden from the avail and spider commands. These are mostly modules for system library packages that other user applications depend on. To list hidden modules, you may provide the -show-hidden option to the avail and spider commands:

```
module --show-hidden avail
module --show-hidden spider
```

A hidden module has a dot (.) in front of its version numbers (eg. zlib/.1.2.8).

List currently loaded modules

```
module list
```

Load a specific version of a module

```
module load name/version
```

If only a name is given, the command will load the default version which is marked with a (D) in the

module avail listing (usually the latest version). Loading a module may automatically load other modules it depends on.

It is not possible to load two versions of the same module at the same time.

To switch between different modules

module swap old new

To unload the specified module from the current environment

module unload name

To clean your environment of all loaded modules

module purge

Show what environment variables the module will set

module show name/version

Save the current list of modules to "name" collection for later use

module save name

Restore modules from collection "name"

module restore name

List of saved collections

module savelist

To get the complete list of options provided by Lmod through the command module type the following

module help

Exercise: Working with modules

As an example, we show how to load the gnuplot module.

List loaded modules

module list

No modules loaded

Find available gnuplot versions

```
module -t spider gnuplot
gnuplot/4.6.0
gnuplot/5.0.3
```

Determine how to load the selected gnuplot/5.0.3 module

```
module spider gnuplot/5.0.3

gnuplot: gnuplot/5.0.3

Description:
Portable interactive, function plotting utility - Homepage:
http://gnuplot.sourceforge.net/

This module can only be loaded through the following modules:

GCC/4.9.3-2.25 OpenMPI/1.10.2

Help:
Portable interactive, function plotting utility - Homepage:
http://gnuplot.sourceforge.net/
```

Load required modules

```
Module for GCCcore, version .4.9.3 loaded
Module for binutils, version .2.25 loaded
Module for GCC, version 4.9.3-2.25 loaded
Module for numactl, version .2.0.11 loaded
Module for hwloc, version .1.11.2 loaded
Module for OpenMPI, version 1.10.2 loaded
```

And finally load the selected gnuplot module

In order to simplify the procedure of loading the gnuplot module, the current list of loaded modules can be saved in a "mygnuplot" collection (the name string "mygnuplot" is, of course, arbitrary) and then loaded again when needed as follows:

Save loaded modules to "mygnuplot"

```
module save mygnuplot

Saved current collection of modules to: mygnuplot
```

If "mygnuplot" not is specified, the name "default" will be used.

Remove all loaded modules (or open a new shell)

List currently loaded modules. This selection is empty now.

```
module list
No modules loaded
```

List saved collections

```
module savelist

Named collection list:
   1) mygnuplot
```

Load gnuplot module again

List of available software

In this section, you will find user guides for some of the software packages installed in the cluster. The guides provided can, of course, not replace documentation that comes with the application. Please read that as well.

A wide variety of application software is available in the cluster system. These applications are located on a central storage system that is accessible by the module system Lmod via an NFS export. Issue the command module spider on the cluster system or visit the page for a comprehensive list of available software. If you really need a different version of an already installed application, or one that is currently not installed, please get in touch. The main prerequisite for use of a software within the cluster system is its availability for Linux. Furthermore, if the application requires a license, we need to clarify additional questions.

Some select Windows applications can also be executed on the cluster system with the help of Wine or Singularity containers. For information on Singularity, see Singularity Containers.

A current list of available software

Usage instructions

- Abaqus
- ANSYS / CFX
- COMSOL
- Conda
- Conda
- CPMD
- FEKO
- Jupyter in the cluster
- MATLAB
- Moose
- mpiFileUtils
- NFFT

Build software from source code

Note: We recommend using EasyBuild (see next section) if you want to make your software's build process reproducible and accessible through a module environment that EasyBuild automatically creates. EasyBuild comes with pre-configured recipes for installing thousands of scientific applications.

Sub-clusters of the cluster system have different CPU architectures that provide different instruction set capabilities/extensions. The command lcpuarchs (available on the login nodes) lists all available

CPU types.

```
login03:~$ lcpuarchs -v
CPU arch names
                       Cluster partitions
- - - - - - - - - - - - -
CPU arch names
                       Cluster partitions
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                       LUIS[smp,helena]
sse
                       FCH[]
                       LUIS[dumbo]
avx
                       FCH[iazd,isu,itp]
                       LUIS[haku,lena,smp]
avx2
                       FCH[ai,gih,isd,iqo,iwes,pci,fi,imuk]
avx512
                       LUIS[gpu,gpu.test,amo,taurus,vis]
FCH[tnt,isd,stahl,enos,pcikoe,pcikoe,isu,phd,phdgpu,muk,fi,itp]
CPU of this machine: avx2
For more verbose output type: lcpuarchs -vv
```

The technical sequence of these architectures is (oldest)-sse-avx-avx2-avx512-(newest). Cpus capable of executing instructions from newer instruction sets usually are able to execute commands from older extensions, so e.g. avx512 includes sse.

Software compiled to use a newer cpu instruction set will usually not typically abort with an "Illegal instruction" error when run on an older cpu. The important message here is that compilers may automatically set flags for the platform you are currently working on. If you compile your program on a node providing avx512 instructions (e. g. the amo sub-cluster) using the gcc compiler option - march=native, the program will usually not run on older nodes that are only equipped with cpus providing, say, avx instructions. To check which instruction set extensions a cpu architecture provides, you can run the command "Iscpu", which lists them in the "flags" section.

This section explains how to build a software on the cluster system to avoid the aforementioned issue if you want to be able to submit jobs to all compute nodes without specifying the CPU type. Beware, though, that compatibility usually comes at a price and allowing a compiler to use the newer instruction sets will usually boost performance. Depending on your workload, the effects/speedup on newer cpu architectures may even be called "drastic". But there's usually no better way to tell than testing.

In the example below we want to compile a sample software my-soft in version 3.1.

In your HOME (or, perhaps better, in your \$SOFTWARE directory, if all members of your project want to use the software) directory, create build/install directories for each available CPU architecture listed by the command lcpuarchs -s, as well as a directory source to storing the installation sources

Note: you can usually refer to a variable using \$variable_name and all will be well. In the following examples, however, we demonstrate the use of curly brackets around the variable's designator, which

will ensure proper separation of variables even in case of ambiguities (which in theory could occur in long paths composed out of several variables). For all normal purposes, \$HOME and \${HOME} or \$LUIS_CPU_ARCH and \${LUIS_CPU_ARCH} will be equivalent. If, however, you use spaces in your directories (like dir="my directory"), this will not be sufficient, you'll then also need to put double quotation marks "around the variable (like in cd "\${dir}").

```
login03:~$ mkdir -p ${HOME}/sw/source
login03:~$ eval "mkdir -p ${HOME}/sw/{$(lcpuarchs -ss)}/my-
soft/3.1/{build,install}"
```

Copy software installation archive to the source directory

```
login03:~$ mv my-soft-3.1.tar.gz ${HOME}/sw/source
```

Build my-soft for each available CPU architecture by submitting an interactive job to each compute node type requesting the proper CPU type. For example, to compile my-soft for avx512 nodes, first submit an interactive job requesting the avx512 feature:

```
login03:~$ salloc --nodes=1 --constraint=CPU_ARCH:avx512 --cpus-per-task=4
--time=6:00:00 --mem=16G
```

Then unpack and build the software. Note the environment variable \${LUIS_CPU_ARCH} that contains the cpu instruction set of the compute node reserved.

```
taurus-n034:~$ tar -zxvf ${HOME}/sw/source/my-soft-3.1.tgz -C
$HOME/sw/${LUIS_CPU_ARCH}/my-soft/3.1/build
taurus-n034:~$ cd ${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/build
taurus-n034:~$ ./configure --prefix=${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install && make && make install
```

Finally, use the environment variable \${LUIS_CPU_ARCH} in your job scripts to access the correct installation path of my-soft executable for the current compute node. Note that you may need to set/update the \${LD_LIBRARY_PATH} environment variable to point to the location of your software's shared libraries.

my-soft-job.sh

```
#!/bin/bash -l
#SBATCH --job-name=my-soft
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --mem=60G
#SBATCH --time=12:00:00
#SBATCH --constraint="[CPU_ARCH:avx512|CPU_ARCH:avx2]"
#SBATCH --output my-soft-job_%j.out
#SBATCH --error my-soft-job_%j.err
#SBATCH --mail-user=myemail@....uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
```

```
# Change to work dir
cd ${SLURM_SUBMIT_DIR}

# Load modules
module load my_necessary_modules

# run my_soft
export LD_LIBRARY_PATH=${HOME}/sw/${LUIS_CPU_ARCH}/my-
soft/3.1/install/lib:${LD_LIBRARY_PATH}
srun $HOME/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install/bin/my-soft.exe --
input file.input
```

You can certainly consider combining the software build and execution steps into a single batch job script. However, it is recommended that you first perform the build steps **interactively** before adding them to a job script to ensure the software compiles without errors. For example, such a job script might look like this:

my-soft-job.sh

```
#!/bin/bash -l
#SBATCH -- job-name=my-soft
#SBATCH -- nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --mem=120G
#SBATCH --time=12:00:00
#SBATCH --constraint=CPU ARCH:avx512
#SBATCH -- output my-soft-job %j.out
#SBATCH --error my-soft-job %j.err
#SBATCH --mail-user=myemail@....uni-hannover.de
#SBATCH --mail-type=BEGIN, END, FAIL
# Change to work dir
cd ${SLURM SUBMIT DIR}
# Load modules
module load my necessary modules
# install software if the executable does not exist
[ -e "${HOME}/sw/${LUIS CPU ARCH}/my-soft/3.1/install/bin/my-soft.exe"
] || {
 mkdir -p ${HOME}/sw/${LUIS_CPU_ARCH}/mysof/3.1/{build,install}
  tar -zxvf ${HOME}/sw/source/my-soft-3.1.tgz -C
${HOME}/sw/${LUIS CPU ARCH}/my-soft/3.1/build
  cd $HOME/sw/${LUIS CPU ARCH}/my-soft/3.1/build
  ./configure --prefix=${HOME}/sw/${LUIS CPU ARCH}/my-soft/3.1/install
  make
 make install
```

```
# run my_soft
export LD LIBRARY PATH=${HOME}/sw/${LUIS CPU ARCH}/my-
soft/3.1/install/lib:${LD LIBRARY PATH}
srun ${HOME}/sw/${LUIS CPU ARCH}/my-soft/3.1/install/bin/my-soft.exe --
input file.input
```

EasyBuild

Note: If you want to manually build the software from source code, please refer to the section above.

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way.

EasyBuild framework

The EasyBuild framework is available in the cluster through the module EasyBuild-custom. This module defines the location of the EasyBuild configuration files, recipes and installation directories. You can load the module using the command:

```
module load EasyBuild-custom
```

EasyBuild software and modules will be installed by default under the following directory:

```
$HOME/my.soft/software/${LUIS CPU ARCH}
$HOME/my.soft/modules/${LUIS CPU ARCH}
```

Here, the variable ARCH, which stores the CPU type of the machine on which the above module load command was executed, will currently be either haswell, sandybridge or skylake. The command lcpuarchs executed on the cluster login nodes lists all currently available values of ARCH. You can override the default software and module installation directory, and the location of your EasyBuild configuration files (MY_EASYBUILD_REPOSITORY) by exporting the following environment variables before loading the EasyBuild module:

```
export EASYBUILD INSTALLPATH=/your/preferred/installation/dir
export MY EASYBUILD REPOSITORY=/your/easybuild/repository/dir
module load EasyBuild-custom
```

If other project members should also have access to the software, the recommended location is a subdirectory in \$SOFTWARE.

How to build your software

After you load the EasyBuild environment as explained in the section above, you will have the

command eb available to build your code using EasyBuild. If you want to build the code using a given configuration <filename>.eb and resolving dependencies, you will use the flag -r as in the example below:

```
eb <filename>.eb -r
```

The build command just needs the configuration file name with the extension .eb and not the full path, provided that the configuration file is in your search path: the command eb --show-config will print the variable robot-paths that holds the search path. More options are available - please have a look at the short help message typing eb -h. For instance, using the search flag -S, you can check if any EasyBuild configuration file already exists for a given program name:

```
eb -S  program_name>
```

You will be able to load the modules created by EasyBuild in the directory defined by the EASYBUILD INSTALLPATH variable using the following commands:

```
module use $EASYBUILD_INSTALLPATH/modules/${LUIS_CPU_ARCH}/all
module load <modulename>/version
```

The command module use will prepend the selected directory to your MODULEPATH environment variable, therefore the command module avail will show modules of your software as well.

If you want the software module to be automatically available when opening a new shell in the cluster, modify your ~/. bashrc file as follows:

```
echo 'export EASYBUILD_INSTALLPATH=/your/preferred/installation/dir' >>
~/.bashrc
echo 'module use $EASYBUILD_INSTALLPATH/modules/${LUIS_CPU_ARCH}/all' >>
~/.bashrc
```

Note that to preserve the dollar sign in the second line above, the string must be enclosed in single quotes.

Further Reading

- EasyBuild documentation
- Easyconfigs repository

Apptainer Containers (replaces Singularity)



Apptainer will replace Singularity on the the LUH-Clusters. Currently you can use both commands apptainer and singularity because the last one is a symlink to apptainer. This may change in the future.

Please note: This instruction has been written for Apptainer 1.3.3-*

Please note: If you would like to fully manage your apptainer container images directly on the cluster, including build and/or modify actions, please contact us and ask for the permission "apptainer fakeroot" to be added to your account (because you will need it).

Apptainer containers on the cluster

Apptainer enables users to execute containers on High-Performance Computing (HPC) cluster like they are native programs or scripts on a host computer. For example, if the cluster system is running CentOS Linux, but your application runs in Ubuntu, you can create an Ubuntu container image, install your application into that image, copy the image to an approved location on the cluster and run your application using Apptainer in its native Ubuntu environment.

The main advantage of Apptainer is that containers are executed as an unprivileged user on the cluster system and, besides the local storage TMPDIR, they can access the network storage systems like HOME, BIGWORK and PROJECT, as well as GPUs that the host machine is equipped with.

Additionally, Apptainer properly integrates with the Message Passing Interface (MPI), and utilizes communication fabrics such as InfiniBand and Intel Omni-Path.

If you want to create a container and set up an environment for your jobs, we recommend that you start by reading the Apptainer documentation. The basic steps to get started are described below.

Building Apptainer container using a recipe file

If you already have a pre-build container ready for use, you can simply upload the container image to the cluster and execute it. See the section below about running container images.

Below we will describe how to build a new or modify an existing container directly on the cluster. A container image can be created from scratch using a recipe file, or fetched from some remote container repository. In this sub-section, we will illustrate a recipe file method. In the next one, we will take a glance at remote container repositories.

Using a Apptainer recipe file is the recommended way to create containers if you want to build reproducible container images. This example recipe file builds a RockyLinux 9 container:

rocky9.def

```
BootStrap: yum
OSVersion: 9
MirrorURL: https://ftp.uni-
hannover.de/rocky/%{OSVERSION}/BaseOS/$basearch/os
Include: yum

%setup
    echo "This section runs on the host outside the container during
bootstrap"
```

```
%post
  echo "This section runs inside the container during bootstrap"
  # install packages in the container
 yum -y groupinstall "Development Tools"
  yum -y install wget vim python3 epel-release
 yum -y install python3-pip
  # install tensorflow
  pip3 install --upgrade tensorflow
 # enable access to BIGWORK and PROJECT storage on the cluster system
 mkdir -p /bigwork /project
%runscript
  echo "This is what happens when you run the container"
 echo "Arguments received: $*"
  exec /usr/bin/python3 "$@"
%test
  echo "This test will be run at the very end of the bootstrapping
process"
  /usr/bin/python3 --version
```

This recipe file uses the yum bootstrap module to bootstrap the core operation system, RockyLinux 9, within the container. For other bootstrap modules (e.g., docker) and details on apptainer recipe files, refer to the online documentation.

The next step is to build a container image on one of the cluster login servers.

Note: your account must be authorized to use the --fakeroot option. Please contact us at cluster-help@luis.uni-hannover.de.

Note: Currently, the --fakeroot option is enabled only on the cluster login nodes.

```
username@login01$ apptainer build --fakeroot rocky9.sif rocky9.def
```

This creates an image file named rocky9.sif. By default, apptainer containers are built as read-only SIF(Apptainer Image Format) image files. Having a container in the form of a file makes it easier to transfer it to other locations both within the cluster and outside of it. Additionally, a SIF file can be signed and verified.

Note that a container as the SIF file can be built on any storage of the cluster you have a write access to. However, it is recommended to build containers either in your \$BIGWORK or in some directory under /tmp (or use the variable \$MY_APPTAINER) on the login nodes.

Note: Containers located only under the paths \$BIGWORK, \$SOFTWARE and /tmp are allowed to be executed using shell, run or exec commands, see the section below,

 $upuate: \\ 2025/03/24 \ guide: modules_and_application_software \ https://docs.cluster.uni-hannover.de/doku.php/guide/modules_and_application_software \ https://docs.cluster.uni-hannover.de/doku.php/guide/modules_and_application_application_application_application_application_application_application_application_application_application_application_application_application_application_applica$ 13:36

The latest version of the apptainer command can be used directly on any cluster node without prior activation.

Downloading containers from external repositories

Another easy way to obtain and use a Apptainer container is to retrieve pre-build images directly from external repositories. Popular repositories are Docker Hub or Apptainer Library. You can go there and search if they have a container that meets your needs. For docker images, use the search form at Docker Hub instead.

In the following example we will pull the latest python container from Docker Hub and save it in a file named python_latest.sif:

```
username@login01$ apptainer pull docker://python:latest
```

The build sub-command can also be used to download images, where you can additionally specify your preferred container file name:

```
username@login01$ apptainer build my-ubuntu22.04.sif
library://library/default/ubuntu:22.04
```

How to modify existing Apptainer images

First you should check if you really need to modify the container image. For example, if you are using Python in an image and simply need to add new packages via pip you can do that without modifying the image by running pip in the container with the --user option.

To modify an existing SIF container file, you need to first convert it to a writable sandbox format.

Please note: Since the -- fakeroot option of the shell and build sub-commands does not work with container sandbox when the container is located on a shared storage such as BIGWORK, PROJECT or HOME, the container sandbox must be stored locally on the login nodes. We recommend using the /tmp directory (or variable \$MY APPTAINER) which has sufficient capacity.

```
username@login01$ cd $MY APPTAINER
username@login01$ apptainer build --sandbox rocky9-sandbox rocky9.sif
```

The build command above creates a sandbox directory called rocky9-sandbox which you can then shell into in writable mode and modify the container as desired:

```
username@login01$ apptainer shell --writable --fakeroot rocky9-sandbox
Apptainer> yum install -qy python3-matplotlib
```

After making all desired changes, you exit the container and convert the sandbox back to the SIF file using:

```
Apptainer> exit username@login01$ apptainer build -F --fakeroot rocky9.sif rocky9-sandbox
```

Note: you can try to remove the sandbox directory *rocky9-sandbox* afterward but there might be a few files you can not delete due to the namespace mappings that happens. The daily /tmp cleaner job will eventually clean it up.

Running container images

Please note: In order to run a Apptainer container, the container SIF file or sandbox directory must be located either in your \$BIGWORK, in your group's \$SOFTWARE or in the /tmp directory.

There are four ways to run a container under Apptainer.

If you simple call the container image as an executable or use the Apptainer run sub-command it will carry out instructions in the %runscript section of the container recipe file:

How to call the container SIF file:

```
username@login01:~$ ./rocky9.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

Use the run sub-command:

```
username@login01:~$ apptainer run rocky9.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

The Apptainer exec sub-command lets you execute an arbitrary command within your container instead of just the %runscript. For example, to get the content of file /etc/os-release inside the container:

```
username@login01:~$ apptainer exec rocky9.sif cat /etc/os-release
NAME="Rocky Linux"
VERSION="8.4 (Green Obsidian)"
....
```

The Apptainer shell sub-command invokes an interactive shell within a container. Note the Apptainer> prompt within the shell in the example below:

```
username@login01:$ apptainer shell rocky9.sif
Apptainer>
```

Note that all three sub-commands shell, exec and run let you execute a container directly from remote repository without first downloading it on the cluster. For example, to run an one-liner "Hello World" ruby program:

```
username@login01:$ apptainer exec library://sylabs/examples/ruby ruby -e
'puts "Hello World!"'
Hello World!
```

Please note: You can access (read & write mode) your HOME, BIGWORK and PROJECT (only login nodes) storage from inside your container. In addition, the /tmp (or TMPDIR on compute nodes) directory of a host machine is automatically mounted in a container. Additional mounts can be specified using the --bind option of the exec, run and shell sub-commands, see apptainer run --help.

Apptainer & parallel MPI applications

In order to containerize your parallel MPI application and run it properly on the cluster system you have to provide MPI library stack inside your container. In addition, the userspace driver for Mellanox InfiniBand HCAs should be installed in the container to utilize cluster InfiniBand fabric as a MPI transport layer.

This example Apptainer recipe file ubuntu-openmpi.def retrieves an Ubuntu container from Docker Hub, and installs required MPI and InfiniBand packages:

Ubuntu 20.04

ubuntu-openmpi.def

```
BootStrap: docker
From: ubuntu:focal

%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common libibverbs1 libmlx4-1

# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project

# enable access to /scratch dir. required by mpi jobs
mkdir -p /scratch
```

Ubuntu 22.x - 24.x

ubuntu-openmpi.def

BootStrap: docker

```
%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common ibverbs-providers
# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project
# enable access to /scratch dir. required by mpi jobs
mkdir -p /scratch
```

Once you have built the image file ubuntu-openmpi.sif as explained in the previous sections, your MPI application can be run as follows (assuming you have already reserved a number of cluster compute nodes):

```
module load GCC/10.2.0 OpenMPI/4.0.5
mpirun apptainer exec ubuntu-openmpi.sif /path/to/your/parallel-mpi-app
```

The above lines can be entered at the command line of an interactive session, or can also be inserted into a batch job script.

Further Reading

- Apptainer home page
- Apptainer Library
- Docker Hub

1

https://lmod.readthedocs.io/en/latest/010_user.html

https://easybuild.readthedocs.io/en/latest/

From:

https://docs.cluster.uni-hannover.de/ - Cluster Docs

Permanent link:

https://docs.cluster.uni-hannover.de/doku.php/guide/modules and application software

Last update: 2025/03/24 13:36

