



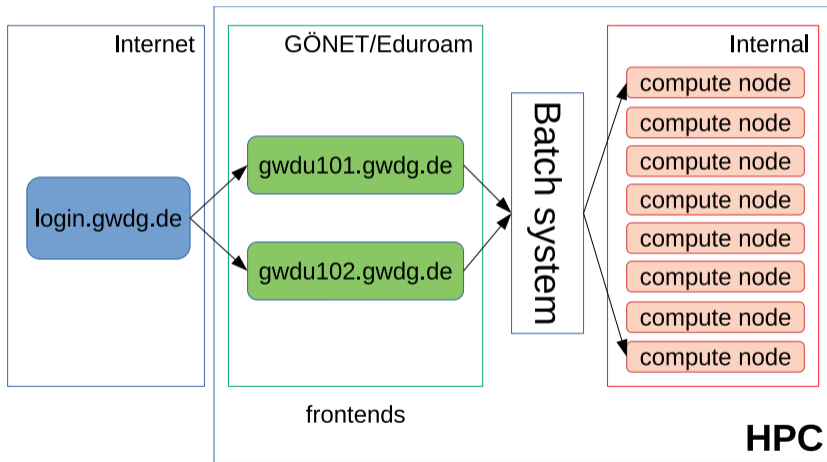
Kevin Lüdemann, Azat Khuziyakhmetov

Cluster introduction

Table of contents

- 1 Hardware overview
- 2 Modules and Containers
- 3 Compiling Software

Hardware and Network



Structure

Two sites:

- Modular Data Center (MDC)

- ▶ Frontends: login-mdc.hpc.gwdg.de (gwdu101 and gwdu102)
- ▶ EmmyP2
- ▶ SCC medium

- RZGö

- ▶ Frontend: glogin.hpc.gwdg.de
- ▶ Nodes: EmmyP3, Grete, KISSKI,
- ▶ SCC and SCC-GPU

Filesystem

■ 2 filesystems

1 **HOME** filesystem

2 **SCRATCH** filesystem

■ HOME

- ▶ Stores your *permanent* data.
- ▶ There is a quota. It could be extended on request.
- ▶ Has a backup mechanism.

■ SCRATCH

- ▶ Stores your data used for computations or projects.
- ▶ Fast and large filesystem.
- ▶ No quota, but also no backup.

Filesystem /local

- **local** filesystem is NOT shared, but fast (SSDs).
- Use it for temporal data on every node
- The size of it rather small

```
bash-4.2$ df -h /local
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda6	78G	57M	74G	1%	/local

- Location is in the variable `TMP_LOCAL`

```
bash-4.2$ echo $TMP_LOCAL  
/local/jobs/15287707/
```

Data archiving

Archive location

- Personal archive is located at `/usr/users/a/USERNAME`
- You can get the path from `$AHOME` variable

Usage

- It is necessary to compress directories as tar or zip files
- if you want to archive directory data, call

```
tar -czvf $AHOME/data.tgz data
```

or faster (uses 4 cores and faster compression)

```
PIGZ="-1 -p 4 -R" tar -I pigz -cvf $AHOME/data.tgz data
```

Exercises

- connect to the frontends
- check your HOME quota
- check out the scratch file system. How big are they, how much space is currently available?
- You downloaded a large genome database (100GB) from NCBI. Where would you store it and why?
- use scratch and archive:
 - ▶ Create a project directory on scratch
 - ▶ Add some files in it (e.g. `date > file1.txt`)
 - ▶ Compress the folder and send to archive

Time: 10 minutes

The workflow with /scratch filesystem

- The Scratch filesystem is **NOT** a permanent storage
- Recommended workflow
 - ▶ Create directory for your project /scratch/users/\$USER/PROJECT
 - ▶ Copy all necessary data there
 - ▶ Run your compute jobs
 - ▶ After completion of your jobs, save important results, that you need for further work to your home directory
 - ▶ Delete all temporary files and broken runs
 - ▶ Move the rest of the directory, that you want to keep for reference, into the archive and delete it from Scratch

```
tar -czvf $AHOME/PRJ.tar.xz /scratch/users/$USER/PROJECT  
rm -rf /scratch/users/$USER/PROJECT
```

Data transfer. Usage

■ SCP

- ▶ *works on Linux, macOS, and latest Windows*

```
scp -rp {SRC-DIR} {USER}@glogin.hpc.gwdg.de:{DST-DIR}
```

- ▶ to transfer back, simply swap the arguments

```
scp -rp {USER}@glogin.hpc.gwdg.de:{SRC-DIR} {DST-DIR}
```

■ Filezilla

- ▶ *works on all platforms. GUI. Open source software.*

■ Rsync

- ▶ *works on Linux, macOS*

```
rsync -avvH {SRC-DIR} {USER}@glogin.hpc.gwdg.de:{DST-DIR}
```

- ▶ to transfer back, simply swap the arguments

```
rsync -avvH {USER}@glogin.hpc.gwdg.de:{SRC-DIR} {DST-DIR}
```

The modules system

Problem:

- HPC Systems have a complex software ecosystem
 - ▶ different versions needed
 - ▶ complicated compiler requirements
 - ▶ library dependencies
- Package manager (yum, apt, etc.) cannot satisfy these requirements
- Compilation can be complicated

Solution:

- We compile/install software as necessary
- Make the software available with “modules”

The modules system

- `module avail` find a list of installed modules
- `module list` list of currently loaded modules
- `module load software/version`
- `module purge` unload all modules
- `module unload software` unload a single module
- Most of the modules just append or prepend a path to `PATH` and `MANPATH` variables.
- Or set default variables to be found by compiler/configure scripts at compile time.

CPU architecture specific modules

- Software provided as modules are compiled for specific CPU architecture: Cascadelake or Haswell.
- Names of these modules are the same, the correct version is loaded depending on the node you(your jobs) are.
- If you compile your software for specific architecture, check the modules you are using with `module whatis` command. It contains the "Target".

```
> gwdu103 ~ > module whatis gromacs
> ...
> gromacs/2020.4      : Target : haswell
> gwdu101 ~ > module whatis gromacs
> ...
> gromacs/2020.4      : Target : cascadelake
```

Apptainer containers

Apptainer is the containerization system, just like Docker. However, we don't provide Docker in HPC for security reasons.

Usage

To load apptainer use the modules

```
module load apptainer
```

You can run either native apptainer or Docker images.

```
apptainer run library://sylabscs/examples/lolcow
```

With Docker image

```
apptainer run docker://godlovedc/lolcow
```

Some software packages provide Docker or Apptainer images, if they do it will be easier to run them as containers.

Exercises

- Have a look at the available modules.
 - ▶ Load a module and see how your environment changes.
 - ▶ Log in and log out again. Are the modules still loaded?
- Run an Apptainer container.

Time: 5 Minutes

Why Compiling?

- Compiling means to create an executable – or a library – from the source code
- GWDG cannot install all software required by users (see modules for what is available)
- Scientific software is often only available as source code
- Compiling on the target system often yields better performance
- Prepackaged software typically requires administrator (root) privileges ...

Recipe: wget and tar

Using wget and tar to prepare the source code

```
> mkdir $HOME/build  
> cd $HOME/build  
> wget <tarball URL>  
> tar xvzf <name-version>.tar.gz  
> cd <name-version>
```

Compiling (or “Building”) the Software

- Standard method: “./configure; make; [make check; make install]”
- Without root privileges: “- -prefix” at configuration

About "--prefix"

- "--prefix" is used to specify the base directory for your software
- use `./configure --prefix=DIR` to **install directly in DIR**.
- e.g. `./configure --prefix=$HOME/software/<name-version>` to **install into a software specific directory**.

Recipe: Basic Building and Installing

Building and installing software into a specific directory

```
> cd $HOME; mkdir software
> cd $HOME/build/<name-version>
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
> ln -s $HOME/software/<name-version>/bin/* $HOME/bin
> ln -s $HOME/software/<name-version>/lib/* $HOME/lib
> ln -s $HOME/software/<name-version>/include/* $HOME/include
```

Compilers

- The GNU compilers (`gcc`, `gfortran`) are the standard compilers in Linux
- Other compilers are often faster, especially for Fortran code
- Recommended for overall performance: Intel compilers (`icc`, `ifort`)

Recipe: Using Intel Compilers

Building and installing software with Intel compilers

```
> module load intel
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Intel Math Kernel Library (MKL)

- A (shared) library is a collection of thematically related subroutines ready to use in a program
- The process of connecting a library to the (compiled) program is called linking
- Intel's Math Kernel Library provides performance optimized linear algebra and Fourier transform functions

Recipe: Using the MKL

Example: linking programs to MKL

```
> module load intel
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> module load intel-parallel-studio
> export CPPFLAGS="-I${MKLR00T}/include -I${MKLR00T}/include/fftw"
> export LDFLAGS="-L${MKLR00T}/lib/intel64 -lmkl_intel_lp64\
> -lmkl_sequential -lmkl_core -lpthread -lm"
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Use Intel MKL Link Line Advisor!

https:

[//software.intel.com/en-us/articles/intel-mkl-link-line-advisor](https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor)

Exercises: Compile your own editor!

In this exercise, you will download and compile the latest version of the nano editor

- Download the latest version of nano using `wget` or `curl`
- extract it using `tar`
- create a build directory
- configure the software with a prefix
- compile and install it
- test it!

Time: 10 Minutes