The use of LUMI supercomputer

The use of LUMI supercomputer

- Computing environment (CE) and available software libraries and tools

- HPC resources allocation, SLURM (guided examples)

- Scratch and Project storage (guided examples)

CE, Libraries, Tools

- Modules
- Programming Environments
- Software and Libraries

Modules

Module environments

- Modules are commonly used on HPC systems to enable users to create custom environments and select between multiple versions of applications => non-standard installation locations
- 3 “module” systems in use
  - Original module tool written in C with modules in Tcl, not really developed anymore
  - New implementation entirely in Tcl with many new features, developed at INRIA - Not supported by HPE Cray
  - Lmod, an implementation in Lua with native module files in Lua
- The LUST software team chose Lmod for LUMI

Exploring modules with Lmod

- Contrary to some other module systems, not all modules are immediately available for loading
  - Installed modules: All modules on the system that can be loaded one way or another
  - Available modules: Can be loaded without first loading another module
- Examples in the HPE Cray PE:
  - `cray-mpich` can only be loaded if a compiler module and network target module are loaded
  - Many of the performance monitoring tools only become available after loading `perftools-base`
  - `cray-fftw` only becomes available when a processor target module is loaded
- Tools
  - `module avail` is for searching in the available modules
  - `module spider` and `module keyword` are for searching in the installed modules

Benefits of a hierarchy

- When well designed, you get some protection from loading modules that do not work together well
  - Only partially implemented on LUMI
  - Ask Helpdesk when you suspect a clash in premade modules

When “swapping” a module that makes other modules available with a different one, Lmod will try to look for equivalent modules in the new hierarchy

- Example: Try `module load PrgEnv-gnu` in the default login environment and see what happens

Module names and families

- In Lmod you cannot have two modules with the same name loaded together
  - On LUMI, when loading a new module the other one with the same name will be automatically unloaded
  - Automatic protection from conflicts
- Extension: family concept: No two modules of the same family can be loaded together
  - E.g., make compilers member of the family “compiler”
  - On LUMI, the conflicting module of the same family will be unloaded automatically

Extensions

- It would not make sense to have a separate module for each of the hundreds of R packages or tens of Python packages a software stack may contain.
  - Would actually also create a performance problem due to excess metadata access and long PATH variables
  - Bundle related packages in a single module
- Lmod solution: A module can define a list of extensions, basically other packages provided by the module.
  - And the regular commands can be used to search for these
  - Unfortunately not used in the HPE Cray PE cray-python and cray-R modules

module spider

- `module spider` : Long list of all installed software with short description
  - Will also look into modules for “extensions” and show those also, marked with an “E”
- `module spider fftw` : Look for the FFTW libraries on the system
- `module spider fftw/3.3.10.1`: Look for this specific version
  - But this immediately show the problem
    - Some of the lines don’t make much sense (see later)
    - Some options are missing also

module spider for a regular package

- `module spider gnuplot` : Shows all versions of gnuplot on the system
- `module spider gnuplot/5.4.3-cpeGNU-22.08` : Shows help information for the specific module, including what should be done to make the module available

module spider for extensions

- No example in the default Cray modules, so examples come from the LUMI software stacks
- `module spider CMake`
- `module spider CMake/3.24.0` : Will tell you which module contains this version of CMake and how to load it

module keyword

- Currently not yet very useful due to a bug in Cray Lmod
- It searches in the module short description and help for the keyword.
  - E.g., try
    `module keyword https`
- We do try to put enough information in the modules to make this a suitable additional way to discover software that is already installed on the system

module available

- Lists modules that are available for loading
- Likely compatible with currently loaded modules
- List of installed modules is bigger – see `module spider`

Sticky modules and module purge

- On some systems, you will be taught to avoid `module purge` (which unloads all modules)
- Sticky modules are modules that are not unloaded by `module purge`, but reloaded.
    - They can be force-unloaded with `module --force purge` and `module --force unload`
- Used on LUMI for the software stacks and modules that set the display style of the modules
    - But keep in mind that the modules are reloaded, so any change to modules that are loaded by these modules will be wiped out.

Changing how the module list is displayed

- You may have noticed that you don’t see directories in the module view but descriptive texts
- This can be changed by loading a module
  - `ModuleLabel/label` : The default view
  - `ModuleLabel/PEhierarchy` : Descriptive texts, but the PE hierarchy is unfolded
  - `ModuleLabel/system` : Module directories
- Turn colour on or off using `ModuleColour/on` or `ModuleColour/off`
- Show some hidden modules with `ModulePowerUser/LUMI`
  - This will also show undocumented/unsupported modules!
- More customisation possible via LMOD environment variables

Getting help

- `module help` is the command to get help information for available modules
  - Without further arguments: help about the module command
  - We do try to add a bit more help information about what a module provides to the modules than default EasyBuild or Spack installations tend to do.
- Examples:

```
module help cray-mpich
module help cray-python/3.9.4.2
module help buildtools/22.08
```

`module whatis` can produce a short description

```
module whatis cray-python/3.9.4.2
```

A note on caching

- Large module system = lots of small module files = Lustre not very happy
  - But Lmod does use caches by default
  - Currently no system cache, only a user cache in $HOME/.lmod.d
- Cache refreshed automatically every 24 hours
  - You'll notice when the `spider` or `available` commands are slow
  - But you may need to clean the cache after installing new software as on LUMI Lmod does not always detect the change
- Also clear the cache if you notice very strange answers from `module spider`.
  - Looks like the HPE Cray PE sometimes causes cache problems

A note on other commands

- `module load`, `module unload`, `module list` are fairly standard commands and the basic operation is the same in all module systems
  - Note that `module list` may also show inactive modules: Were loaded at some point but got unloaded when a module closer to the root of the hierarchy got unloaded
- `module swap`:
  - Equivalent to an unload followed by a load
  - For two modules of the same family `module swap` is more efficient as Lmod does not first have to discover the family conflict
  - And it is not essential as LUMI has autoswap enabled

Programming Environments

Programming Environments (PEs)

- They are an intrinsic part of an HPC system
  - Programs are preferably installed from sources to generate binaries optimised for the system. When running, the build environment needs to be at least partially recreated.
  - Even when installing from prebuild binaries, some modules might still be needed
  - e.g. you may inject an optimised MPI library

The Operating System

- The login nodes run SUSE Linux Enterprise Server 15 SP3
- The compute nodes run Cray OS, a restricted version of SUSE with some daemons inactive or configured differently and Cray’s way of accessing file systems.
  - Goal is to limit OS jitter for better scalability of large applications
  - On the GPU nodes there was still the need to reserve 1 core for OS and drivers
  - This also implies that some software may not be able to run on the compute nodes
    - E.g., no /run/user/$UID

Programming models

- C/C++ and Fortran provided by several compilers in the PE
- MPI and OpenSHMEM for distributed memory, RCCL
- OpenMP, also for GPU programming
- HIP, AMD alternative for CUDA
- OpenACC only in Cray Fortran
- Commitment to OpenCL unclear
- NO CUDA

Development environment on LUMI

- Compilers: Cray Compiling Environment, GNU compiler collection (AMD support not enabled), AMD AOCC and ROCm compilers
- Cray Scientific and Math Libraries: FFTW, BLAS+Lapack, …
- Cray Message Passing Toolkit
- Additional tools to integrate everything and offer hugepages support
- Cray Performance Measurement and Analysis Tools
- Cray Debugging Support Tools
- Python and R

Cray Compiling Environment

- Default compiler on most Cray systems
  - Designed for scientific software in an HPC environment
  - LLVM-based backend with extensions by HPE Cray for vectorization and share memory parallelization
- Standards support:
  - C/C++ compiler essentially Clang/LLVM
  - Fortran is HPE Cray’s own frontend, supporting most of Fortran 2018
  - OpenMP support including offload: Full 4.5, partial 5.0/5.1 and working on more, see `man intro_openmp`
  - OpenACC support only in Fortran: 2.0, partial 2.x/3.x, see `man intro_openacc`
  - PGAS: UPC 1.2 and Fortran 2008 coarray support
  - MPI bindings

Scientific and math libraries

- LibSci
  - BLAS (Basic Linear Algebra Subroutines) and CBLAS (C interface wrappers)
  - LAPACK and LAPACKE (C interface to LAPACK)
  - IRT (Iterative Refinement Toolkit)
  - BLACS (Basic Linear Algebra Communication Subprograms) and ScaLAPACK
- LibSci_ACC: A subset of GPU-optimized routines from LibSci
- FFTW3: Fastest Fourier Transforms in the West
- Data libraries: netCDF and HDF5

Cray MPI

- Derived from ANL MPICH 3.4
- With tweaks for Cray
  - Improved algorithms for many collectives
  - Asynchronous progress engine for overlap of computation and communications
  - Customizable collective buffering when using MPI-IO
  - Optimized Remote Memory Access (one-sided) fully supported including passive RMA
- GPU-aware communications
- Support for Fortran 2008 bindings
- Full MPI 3.1 support except for dynamic process management and MPI_LONG_DOUBLE/MPI_C_LONG_DOUBLE_COMPLEX for CCE
- No mpirun/mpiexec, but Slurm srun as the process starter
- Layered on libfabric with the Cassini provider, and a GPU Transfer Library

Lmod

- The HPE Cray PE is configured through modules
  - On LUMI we use Lmod
- Basic module commands are similar in all 3 implementations of modules:
  - `module avail` : list available modules
  - `module list` : Show loaded modules
  - `module load` : To load a module
- Lmod supports a hierarchical module system: Distinguishes between available modules and installed modules
  - Some modules only become available after loading another module
  - Can be used to support multiple configurations with a single module name and version
  - Used extensively in the programming environment

Compiler wrappers

- The HPE Cray PE compilers are usually used through compiler wrappers:
  - `cc` for C
  - `CC` for C++
  - `ftn` for Fortran
- Compiler selected based on the modules loaded
- These select the target CPU and GPU architectures based on target modules
- Some libraries are linked in automatically when the corresponding module is loaded
  - MPI: There is no `mpicc`, `mpiCC`, `mpif90`, etc.
  - LibSci and FFTW
  - netCDF and HDF5
- You can see what the wrapper does by adding `--craype-verbose`

Selecting the version of the CPE

- Release numbers are of the type yy.dd, e.g., 22.08 for the release made in August 2022.
- There is always a default version assigned by the sysadmins
- The cpe module allows to change the default version
  - `module load cpe/21.12`
  - This module will try to switch loaded PE modules to the version corresponding to that PE version, but it sometimes fails due to bugs in that module. Loading it twice in separate `module load` commands fixes the issue.
  - Will produce a warning when unloading, but this is only a warning.
- Not needed when using the LUMI stacks, see later

The target modules

- CPU:
  - `craype-x86-rome`: Works everywhere, CPU in the login nodes and data and visualisation partition
  - `craype-x86-milan`: LUMI-C compute nodes
  - `crapye-x86-trento`: LUMI-G CPU
- GPU:
  - `craype-accel-host`: Will tell some compilers to compile for the host instead
  - `craype-accel-amd-gfx90a`: The MI200 series used in LUMI-G
- Network:
  - `craype-network-ofi`: Needed for the Slingshot 11 interconnect
  - `craype-network-none`: Omits network-specific libraries
- Compiler wrappers have corresponding options to overwrite these settings

LUMI

PrgEnv and compiler modules

- The PrgEnv-* modules load compiler, MPI and LibSci (and some other stuff)

PrgEnv	Description	Comp. mod.	Compilers
PrgEnv-cray	Cray compilation Environment	cce	craycc, crayCC, crayftn
PrgEnv-gnu	GNU Compiler Collection	gcc	gcc, g++, gfortran
PrgEnv-aocc	AMD Optimizing Compilers (CPU only)	aocc	clang, clang++, flang
PrgEnv-amd	AMD ROCm LLVM compilers (GPU support)	amd	amdclang, amdclang++, amdflang

- `rocm` module needed when using PrgEnv-cray or PrgEnv-gnu on the GPUs

Getting help

PrgEnv	C	C++	Fortran
PrgEnv-cray	`man craycc`	`man crayCC`	`man crayftn`
PrgEnv-gnu	`man gcc`	`man g++`	`man gfortran`
PrgEnv-aocc/PrgEnv-amd	-	-	-
Wrappers	`man cc`	`man CC`	`man ftn`

- `--help` flag, e.g., for compilers
- `-dumpversion` (wrappers), `--version` (many compiler commands) for version information

Other modules

- cray-mpich for MPI
- cray-libsci for LibSci
- cray-fftw for the Cray FFTW library
- cray-netcdf, cray-netcdf-hdf5parallel, cray-parallel-netcdf, cray-hdf5, cray-hdf5-parallel
- cray-python with a selection of packages including mpi4py, NumPy, SciPy and pandas
- Cray-R

Warning 1: You do not always get what you expect…

- The default behaviour of the Cray PE is to use default versions of libraries at runtime
  - The versions of these libraries do not correspond to the modules loaded
  - Many applications will run without reconstructing the compile environment
  - If the default version on the system changes, the behaviour of your application might change…
- Solution: Prepend LD_LIBRARY_PATH with CRAY_LD_LIBRARY_PATH:
  `export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:$LD_LIBRARY_PATH`
  - Experimental module `lumi-CrayPath` can be used to manage this
- Or use rpath linking when building:
  `export CRAY_ADD_RPATH=yes`

Warning 2: Order matters

- Some modules are only available when others are loaded first
- Some examples
  - cray-fftw only available when a processor target module is loaded
  - cray-mpich requires both craype-network-ofi and a compiler to be loaded
  - cray-hdf5 requires a compiler module to be loaded and cray-netcdf in turn requires cray-hdf5
  - And there are several other examples
- See next presentation to learn how to find modules

Software and Libraries

Software stack design considerations

LUMI

- Very leading edge and inhomogeneous machine (new interconnect, new GPU architecture with an immature software ecosystem, some NVIDIA GPUs for visualisation, a mix of zen2 and zen3)
  - Need to remain agile
- Users that come to LUMI from 11 different channels (not counting subchannels), with different expectations
- Small central support team considering the expected number of projects and users and the tasks the support team has
  - But contributions from local support teams
- Cray Programming Environment is a key part of our system
- Need for customised setups
  - Everybody wants a central stack as long as their software is in there but not much more
  - Look at the success of conda, Python virtual environments, containers, …

The LUMI solution

- Software organised in extensible software stacks based on a particular release of the PE
  - Many base libraries and some packages already pre-installed
  - Easy way to install additional packages in project space
- Modules managed by Lmod
  - More powerful than the (old) Modules Environment
  - Powerful features to search for modules
- EasyBuild is our primary tool for software installations
  - But uses HPE Cray specific toolchains
  - Offer a library of installation recipes
  - User installations integrate seamlessly with the central stack
  - We do have a Spack setup but don’t do development in Spack ourselves

Policies

- Bring-your-own-license except for a selection of tools that are useful to a larger community
  - One downside of the distributed user management is that LUST does not even have the information needed to determine if a particular userid can use a particular software license
  - Even for software on the system, users remain responsible for checking the license
- LUST tries to help with installations of recent software but porting or bug fixing is not available from LUST
  - Not all Linux or even supercomputer software will work on LUMI
  - LUST is too small a team to do all software installations, so don’t count on it to do all the work
- Conda, (large) Python installations need to go in containers
  - LUST offers a container-based wrapper (lumi-container-wrapper) to do that

Organisation: Software stacks

- **CrayEnv:** Cray environment with some additional tools pushed in through EasyBuild
- **LUMI** stacks, each one corresponding to a particular release of the PE
  - Work with the Cray PE modules , but accessed through a replacement for the PrgEnv-* modules
  - Tuned versions for the 3 ~~4~~ types of hardware: zen2 (login, large memory nodes), zen3 (LUMI-C compute nodes), ~~zen2 + NVIDIA GPU (visualisation partition)~~, zen3 + MI250X (LUMI-G GPU partition)
- **spack:** Install software with Spack using compilers from the PE
  - Offered as-is for users who know Spack, but we do not do development in Spack
- Far future: Stack based on common EB toolchains as-is for LUMI-C

Accessing the Cray PE on LUMI
3 different ways

- Very bare environment available directly after login
  - What you can expect on a typical Cray system
  - Few tools as only the base OS image is available
  - User fully responsible for managing the target modules
- **CrayEnv**
  - “Enriched” Cray PE environment
  - Takes care of managing the target modules: (re)loading CrayEnv will reload an optimal set for the node you’re on
  - Some additional tools, e.g., newer build tools (offered here and not in the bare environment as we need to avoid conflicts with other software stacks)
  - Otherwise used in the way discussed in this course

Accessing the Cray PE on LUMI
3 different ways

- **LUMI** software stack
  - Each stack based on a particular release of the HPE Cray PE
    - Other modules are accessible but hidden from the default view
  - Better not to use the PrgEnv modules but the EasyBuild LUMI toolchains

HPE Cray PE	LUMI toolchain	
PrgEnv-cray	cpeCray	Cray Compiling Environment
PrgEnv-gnu	cpeGNU	GNU C/C++ and Fortran
PrgEnv-aocc	cpeAOCC	AMD CPU compilers
PrgEnv-amd	cpeAMD	AMD ROCm GPU compilers (LUMI-G only)

  - Environment in which we install most software (mostly with EasyBuild)

Accessing the Cray PE on LUMI
The LUMI software stack

- The LUMI software stack uses two levels of modules
  - LUMI/22.08, LUMI/22.11: Versions of the LUMI stack
  - partition/L, partition/C, partition/G (~~and future partition/D~~): To select software optimised for the respective LUMI partition
    - partition/L is for both the login nodes and the large memory nodes (4TB)
  - Hidden partition/common for software that is available everywhere, but be careful using it for your own installs
  - When (re)loaded, the LUMI module will load the best matching partition module.
  - So be careful in job scripts: When your job starts, the environment will be that of the login nodes, but if you trigger a reload of the LUMI module it will be that of the compute node!

Installing software on HPC systems

- Software on an HPC system is rarely installed from RPM
  - Generic RPMs not optimised for the specific CPU
  - Generic RPMs may not work with the specific LUMI environment (SlingShot interconnect, kernel modules, resource manager)
  - Multi-user system so usually no “one version fits all”
  - Need a small system image as nodes are diskless
- Spack and EasyBuild are the two most popular HPC-specific software build and installation frameworks
  - Usually install from sources to adapt the software to the underlying hardware and OS
  - Installation instructions in a way that can be communicated and executed easily
  - Make software available via modules
  - Dependency handling compatible with modules

Extending the LUMI stack with EasyBuild

- Fully integrated in the LUMI software stack
  - Load the LUMI module and modules should appear in your module view
  - EasyBuild-user module to install packages in your user space
  - Will use existing modules for dependencies if those are already on the system or in your personal/project stack
- EasyBuild built-in easyconfigs do not work on LUMI, not even on LUMI-C
  - GNU-based toolchains: Would give problems with MPI
  - Intel-based toolchains: Intel compilers and AMD CPUs are a problematic cocktail
- Library of recipes that we made in the LUMI-EasyBuild-contrib GitHub repository
  - EasyBuild-user will find a copy on the system or in your install
  - List of recipes in lumi-supercomputer.github.io/LUMI-EasyBuild-docs

EasyBuild recipes - easyconfigs

- Build recipe for an individual package
  - Relies on either a generic or a specific installation process provided by an easyblock
- Steps
  - Downloading sources and patches
  - Typical configure – build – (test) – install process
  - Extensions mechanism for perl/python/R packages
  - Some simple checks
  - Creation of the module
- All have several parameters in the easyconfig file

The toolchain concept

- A set of compiler, MPI implementation and basic math libraries
  - Simplified concept on LUMI as there is no hierarchy as on some other EasyBuild systems
- These are the cpeCray, cpeGNU, cpeAOCC and cpeAMD modules mentioned before!

HPE Cray PE	LUMI toolchain	
PrgEnv-cray	cpeCray	Cray Compiling Environment
PrgEnv-gnu	cpeGNU	GNU C/C++ and Fortran
PrgEnv-aocc	cpeAOCC	AMD CPU compilers
PrgEnv-amd	cpeAMD	AMD ROCm GPU compilers (LUMI-G only)

The toolchain concept (2)

- Special toolchain: SYSTEM to use the system compiler
  - Does not fully function in the same way as the other toolchains when it comes to dependency handling
  - Used on LUMI for CrayEnv and some packages with few dependencies
- It is not possible to load packages from different cpe toolchains at the same time
  - EasyBuild restriction, because mixing libraries compiled with different compilers does not always work
- Packages compiled with one cpe toolchain can be loaded together with packages compiled with the SYSTEM toolchain
  - But we do avoid mixing them when linking

easyconfig names and module names

GROMACS-2021.4-cpeCray-21.12-PLUMED-2.8.0-CPU.eb

Additional information

Toolchain name and version (missing for SYSTEM)

Version of the package

Name of the package

Module: GROMACS/2021.4-cpeCray-21.12-PLUMED-2.8.0-CPU

Installing
Step 1: Where to install

- Default location is `$HOME/EasyBuild`
- But better is to install in your project directory for the whole project
  - `export EBU_USER_PREFIX=/project/project_465000000/EasyBuild`
  - Set this *before* loading the `LUMI` module
  - All users of the software tree have to set this environment variable to use the software tree
- Then load the `LUMI` module, `partition` module and `EasyBuild-user` module
  - In many cases, cross-compilation is possible by loading a different partition module than the one auto-loaded by `LUMI`
  - Though cross-compilation is currently problematic for GPU code

Installing

Step 2: Configure the environment

- Load the modules for the LUMI software stack and partition that you want to use. E.g.,
```
module load LUMI/22.08 partition/C
```
- Load the EasyBuild-user module to make EasyBuild available and to configure it for installing software in the chosen stack and partition:
```
module load EasyBuild-user
```

Installing
Step 3: Install the software

- Let’s, e.g., install GROMACS
  - Search if GROMACS build recipes are available
    ```
    eb --search GROMACS
    eb –S GROMACS
    ```
    This will be improved in the future.
  - Let’s take GROMACS-2021.4-cpeCray-22.08-PLUMED-2.8.0-CPU.eb:
    ```
    eb GROMACS-2021.4-cpeCray-22.08-PLUMED-2.8.0-CPU.eb -D
    eb GROMACS-2021.4-cpeCray-22.08-PLUMED-2.8.0-CPU.eb -r
    ```
- Now the module should be available
  ```
  module avail GROMACS
  ```

Installing

Step 3: Install the software - Note

- Note: Sometimes the module does not show up immediately. This is because Lmod keeps a cache and fails to detect that the cache is outdated.
  - Remove `$HOME/.lmod.d/.cache`
    ```
    rm -rf $HOME/.lmod.d/.cache
    ```
  - We've seen rare cases where internal Lmod data structures where corrupt and logging out and in again was needed
- Installing this way is 100% equivalent to an installation in the central software tree. The application is compiled in exactly the same way as we would do and served from the same file systems.

More advanced work

- You can also install some EasyBuild recipes that you got from support and are in the current directory (preferably one without subdirectories):
  `eb my_recipe.eb -r .`
  - Note the dot after the –r to tell EasyBuild to also look for dependencies in the current directory (and its subdirectories)
- In some cases you will have to download the sources by hand, e.g., for VASP, which is then at the same time a way for us to ensure that you have a license for VASP. E.g.,
  - `eb --search VASP`
  - Then from the directory with the VASP sources:
    `eb VASP-6.3.2-cpeGNU-22.08.eb -r .`

More advanced work (2): Repositories

- It is possible to have your own clone of the LUMI-EasyBuild-contrib repo in your $EBU_USER_PREFIX subdirectory if you want the latest and greatest before it is in the centrally maintained repository

  -
    ```
    cd $EBU_USER_PREFIX
    git clone https://github.com/Lumi-supercomputer/LUMI-EasyBuild-contrib.git
    ```
- It is also possible to maintain your own repo
  - The directory should be `$EBU_USER_PREFIX/UserRepo` (but of course on GitHub the repository can have a different name)
  - Structure should be compatible with EasyBuild: easyconfig files go in `$EBU_USER_PREFIX/easybuild/easyconfigs`

More advanced work (3): Reproducibility

- EasyBuild will keep a copy of the sources in `$EBU_USER_PREFIX/sources`
- EasyBuild also keeps copies of all installed easyconfig files in two locations:
  - In `$EBU_USER_PREFIX/ebrepo_files`
    - And note that EasyBuild will use this version if you try to reinstall and did not delete this version first!
    - This ensures that the information that EasyBuild has about the installed application is compatible with what's in the module files
  - With the installed software (in `$EBU_USER_PREFIX/SW`) in a subdirectory called easybuild
    This is meant to have all information about how EasyBuild installed the application and to help in reproducing

Additional tips&tricks

- Updating version: Often some trivial changes in the EasyConfig (`.eb`) file
  - Checksums may be annoying: Use `--ignore-checksums` with the `eb` command
- Updating to a new toolchain:
  - Be careful, it is more than changing one number
  - Versions of preinstalled dependencies should be changed and EasyConfig files of other dependencies also checked
- LUMI Software Library at lumi-supercomputer.github.io/LUMI-EasyBuild-docs
  - For most packages, pointers to the license
  - User documentation gives info about the use of the package, or restrictions
  - Technical documentation aimed at users who want more information about how we build the package

EasyBuild training for advanced users and developers

- EasyBuild web site: easybuild.io
- Generic EasyBuild training materials on easybuilders.github.io/easybuild-tutorial.
- Training for CSC and local support organisations: Most up-to-date version of the training materials on klust.github.io/easybuild-tutorial.

Resources

- More information:

  https://docs.lumi-supercomputer.eu/firststeps/getstarted/

- Helpdesk: LUMI User Support Team (LUST) :

  https://docs.lumi-supercomputer.eu/helpdesk/

Based on materials developed by LUST

How to access LUMI

Presented by Jan Vicherek, IT4I