EasyBuild tutorial CSC'22

Kurt Lust (Univ. of Antwerp)

9-11 May 2022

https://klust.github.io/easybuild-tutorial/2022-CSC_and_LO

Introduction to EasyBuild

What is EasyBuild?

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

What is EasyBuild?

- **EasyBuild is a software build and installation framework**
- Strong focus on scientific software, performance, and HPC systems
- Open source (GPLv2), implemented in Python (2.7, 3.5+)
- Brief history:
  - Created in-house at HPC-UGent in 2008 as a tool for the interuniversity VSC project
  - First released publicly in Apr’11
  - EasyBuild 1.0 released in Nov’11 (during SC11)
  - Worldwide community has grown around it since then!

https://easybuild.io

https://docs.easybuild.io

https://github.com/easybuilders

https://easybuild.slack.com
(https://easybuild.io/join-slack)

Twitter: @easy_build

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

EasyBuild in a nutshell

- **Tool to provide a *consistent and well performing* scientific software stack**
- Uniform interface for installing scientific software on HPC systems
- Saves time by *automating* tedious, boring and repetitive tasks
- Can empower scientific researchers to self-manage their software stack
- **A platform for collaboration among HPC sites worldwide**
- Has become an “expert system” for installing scientific software

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

Key features of EasyBuild (1/2)

- Supports fully **autonomously** installing (scientific) software, including dependencies, generating environment module files, …
- **No admin privileges are required** (only write permission to install path)
- Highly configurable, easy to extend, support for hooks, easy customisation
- Detailed logging, fully transparent via support for “dry runs” and trace mode
- Support for using custom module naming schemes (incl. hierarchical)

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

Key features of EasyBuild (2/2)

- Integrates with various other tools (Lmod, Singularity, Slurm, …)
- **Actively developed and supported by worldwide community**
- **Frequent stable releases** since 2011 (every 6 - 8 weeks)
- **Comprehensive testing**: unit tests, testing contributions, regression testing
  - But no Cray test systems
- **Various support channels** (mailing list, Slack, conf calls) + yearly user meetings

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

Focus points in EasyBuild

Performance

- Strong preference for building software from source
- Software is optimized for the processor architecture of build host (by default)

Reproducibility

- Compiler, libraries, and required dependencies are mostly controlled by EasyBuild
  - Cray systems are an exception as EasyBuild interfaces with the Cray PE modules
- Fixed software versions for compiler, libraries, (build) dependencies, ...

Community effort

- Development is highly driven by EasyBuild community
- Lots of active contributors, integration with GitHub to facilitate contributions

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

What EasyBuild is *not*

- EasyBuild is **not YABT (Yet Another Build Tool)**
  - It does not try to replace CMake, make, pip, etc.
  - It wraps around those tools and automates installation procedures
- EasyBuild does **not replace traditional Linux package managers** (yum, dnf, apt, …)
  - You should still install some software via OS package manager: OpenSSL, Slurm, etc.
- EasyBuild is **not a magic solution** to all your (software installation) problems
  - You will still run into compiler errors (unless somebody worked around it already)

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_01_what_is_easybuild/

Introduction to EasyBuild

The Lmod module system

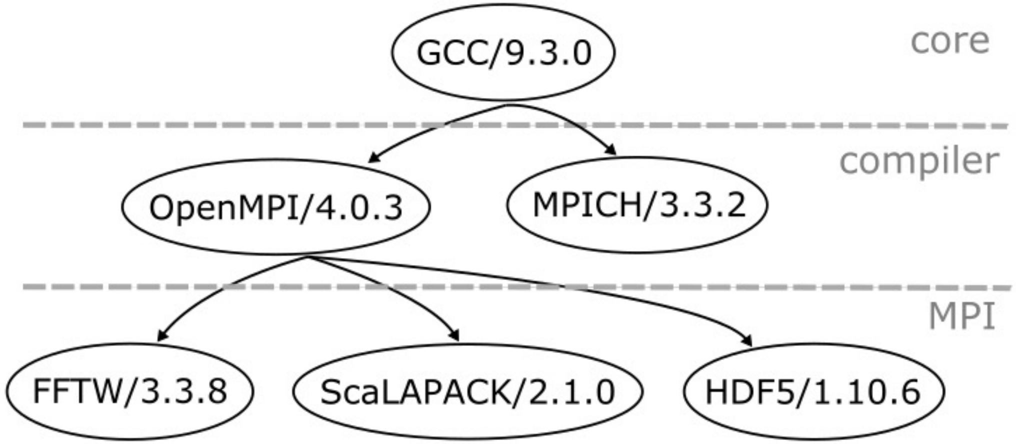
Lmod: LUA environment modules

- Choices
  - Environment modules 3: C implementation, Tcl modules, supported by Cray
  - Environment modules 4 or 5: Tcl implementation + modules, NOT supported by Cray
  - Lmod: LUA implementation + modules, supported by Cray (with some delay)
- Hierarchical module scheme
  - Unconventional hierarchy used in the Cray PE
  - Partly used to organise the software stacks on LUMI (versions and specific hardware)

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_02_Lmod/

Lmod hierarchy

- Model hierarchy: 3 levels
  - Core level: compilers
  - Compiler level: provides libraries that only need the compiler, and MPI modules
  - MPI level: Software that is compiled with MPI support



Lmod hierarchy

- Distinction between
  - Installed modules: All modules that can be loaded one way or another, sometimes by first loading other modules
    => module spider and module keyword
  - Available modules: The modules that can be loaded right away
    => module avail
- 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 avialable when a processor target module is loaded

Lmod hierarchy: Building blocks

- MODULEPATH determines which modules are available
    - Some modules change the MODULEPATH to add a new level of available modules
- “One name rule”
- “Family” concept: extension of the “one name rule”
    - No two modules of the same family can be loaded
    - Example on LUMI: The PrgEnv modules belong to the “PrgEnv” family
    - Example on mahti: Compiler modules aocc/3.2.0, gcc/9.4.0, gcc/11.2.0 belong to the “compiler” family
- Be careful with module naming to exploit this!

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 gnuplot : Shows all versions of gnuplot on the system
  module spider CMake
- module spider gnuplot/5.4.3-cpeGNU-21.12 : Shows help information for the specific module, including what should be done to make the module available
    - But this does not completely work with the Cray PE modules
- module spider CMake/3.22.2 : Will tell you which module contains CMake and how to load it

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_02_Lmod/

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
  - Thinking of proposing an extension to EasyBuild to make this a bit easier without getting ugly looking module help and whatis information

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, which implies that the target modules and partition module will be switched (back) to those for the current node.

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!

Introduction to EasyBuild

The HPE Cray PE

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_03_CPE/

HPE Cray PE components

- Cray compiler environments
    - Compilers preferably used through universal compiler wrappers
    - cc, CC, ftn commands
    - Behaviour depends on the loaded compiler module and target modules
- On LUMI:
    - Cray Compiling Environment (CCE): Clang/LLVM C/C++ and Cray Fortran front-end with LLVM-based backend
    - 3rd party: GNU
    - 3rd party: AMD Optimizing C/C++ and Fortran Compilers (AOCC)
    - 3rd party: AMD ROCm compilers

HPE Cray PE components (2)

- Cray Scientific and Math Library
  - LibSci with BLAS, LAPACK, ScaLAPACK, IRT
  - FFTW
  - HDF5 and NetCDF
  - Wrappers take care of adding the right compiler/linker flags based on loaded modules
- Cray Message Passing Toolkit
  - Libfabric-based with Cassini provider for SlingShot 11
  - UCX will no longer work after the late May LUMI upgrade
- DSMML, Cray Performance Analysis Tools, Cray Debugging Support Tools

Programming Environment Modules

- What they do is determined by a single configuration file
- When interfacing with EasyBuild replaced by an EasyBuild-controlled module

HPE Cray PE	Compiler module	LUMI stack
PrgEnv-cray	cce	cpeCray
PrgEnv-gnu	gcc	cpeGNU
PrgEnv-aocc	aocc	cpeAOCC
PrgEnv-amd	rocm	cpeAMD

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_03_CPE/

Choosing versions through the cpe module

- Loading cpe/yy.mm
  - Sets the default versions of the Cray PE modules to the versions that come with the particular HPE Cray PE release
  - Reloads already loaded PE modules to switch to the default version
- But buggy due to Cray bugs and Lmod limitations
  - Never load with other modules in a single module command
  - May need to load twice to switch all modules to the new version
- In the LUMI software stacks, the LUMI module takes part of this role over
  - needs to be improved
  - cpeCray/cpeGNU etc. modules always (re)load the right versions

Target modules

- craype-x86-* set the target architecture for CPU optimisation
- craype-accel-* set the target architecture for OpenMP offload
    - And dummy craype-accel-host
- craype-network-* set the communication library to be used by Cray MPICH.
- craype-hugepages* modules for Cray Huge Pages support (cce and gcc only)
- EasyBuild currently also uses the target modules rather than command line switches to set optimisation target architectures

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_03_CPE/

Unexpected behaviour

- Dynamic linking needed for system libraries and Cray PE libraries.
- But not all modules set LD_LIBRARY_PATH. Some set CRAY_LD_LIBRARY_PATH instead and will use by default fallback libraries in /opt/cray/pe/lib64
    - And these correspond to the default version of the Cray PE as set in the system
    - So the behaviour of a program may change after a change of default version of the PE

Introduction to EasyBuild

LUMI software stacks

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

Software stacks: 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 which is also supported by HPE Cray
    - 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 can help you with setting up Spack also, but this is not yet automated

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

LUMI 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 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)
  - Some software may be installed outside those stacks
- Far future: Stack based on common EB toolchains as-is
  - MPI may be the problem

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

3 ways to access the Cray PE on LUMI

- 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

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

3 ways to access the Cray PE on LUMI

- 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 LUMI toolchains

HPE Cray PE	LUMI toolchain	What?
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)

  - cpeXXX modules also load the MPI libraries and LibSci just as the PrgEnv-* modules
  - Environment in which we install most software

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

3 ways to access the Cray PE on LUMI

- The LUMI software stack uses two levels of modules
  - LUMI/21.08, LUMI/21.12: Versions of the LUMI stack
  - partition/L, partition/C, partition/EAP (and future partition/D, partition/G): To select software optimised for the respective LUMI partition
    - partition/L is for both the login nodes and the large memory nodes (4TB)
    - partition/EAP doesn't really have any software preinstalled (except for tools that we have everywhere)
  - 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.
  - Hence 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!

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

Partition module

- Targets for the partition modules:

Partition	CPU target	GPU target
partition/L	craype-x86-rome	craype-accel-host
partition/C	craype-x86-milan	craype-accel-host
partition/G	craype-x86-trento	craype-accel-amd-gfx90a
partition/D	craype-x86-rome	craype-accel-nvidia80
partition/EAP	craype-x86-rome	craype-accel-amd-gfx908

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_04_LUMI_software_stack/

Introduction to EasyBuild

Terminology

EasyBuild terminology

- It is important to briefly explain some terminology often used in EasyBuild
- Some concepts are specific to EasyBuild: easyblocks, easyconfigs, …
- Overloaded terms are clarified: modules, extensions, toolchains, …

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

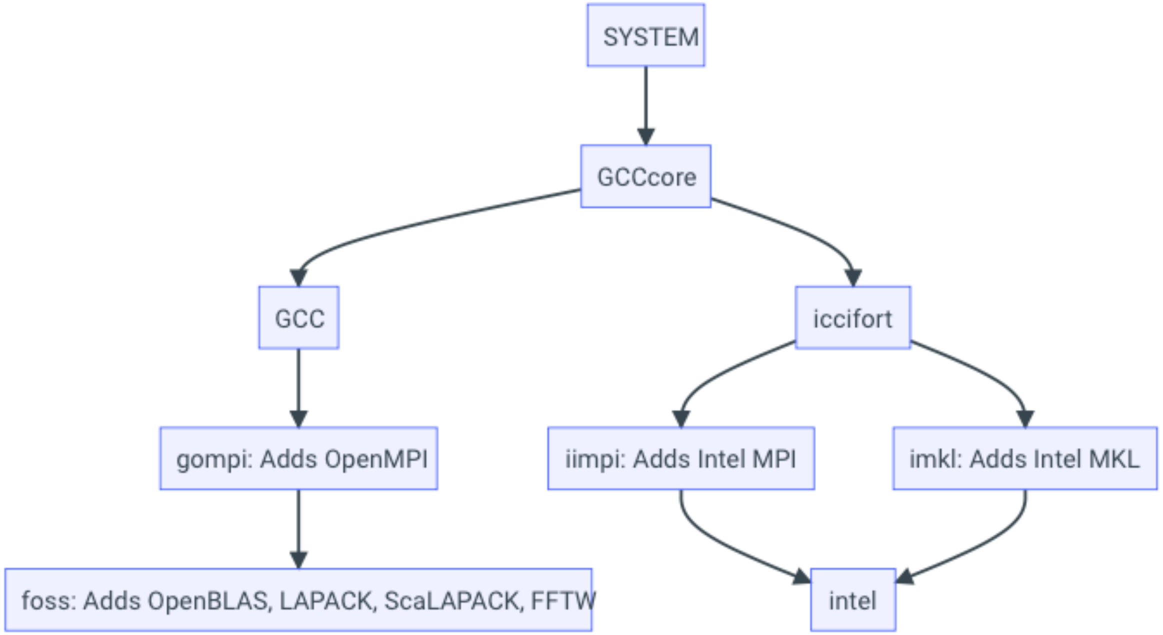
EasyBuild terminology: toolchains

- Compiler toolchain: set of compilers + libraries for MPI, BLAS/LAPACK, FFT, …
- Toolchain component: a part of a toolchain (compiler component, etc.)
- Full toolchain: C/C++/Fortran compilers + libraries for MPI, BLAS/LAPACK, FFT
- Subtoolchain (partial toolchain): compiler-only, only compiler + MPI, etc.
- System toolchain: use compilers (+ libraries) provided by the operating system
- Common toolchains: widely used toolchain in EasyBuild community:
    - `foss`: GCC + OpenMPI + (FlexiBLAS +) OpenBLAS + FFTW
    - `intel`: Intel compilers + Intel MPI + Intel MKL

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

EasyBuild terminology: toolchains

- Organised in a hierarchy



https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

EasyBuild terminology: framework

- The EasyBuild framework is the **core of EasyBuild**
- **Collection of Python modules**, organised in packages
- Implements **common functionality** for building and installing software
- Support for applying patches, running commands, generating module files, ...
- Examples: `easybuild.toolchains`, `easybuild.tools`, `...`
- Provides eb command, but can also be leveraged as a Python library
- GitHub repository: https://github.com/easybuilders/easybuild-framework

EasyBuild terminology: easyblock

- A Python module that implements a specific software installation procedure
  - Can be viewed as a “plugin” to the EasyBuild framework
- Generic easyblocks for “standard” stuff: cmake + make + make install, Python packages, etc.
- Software-specific easyblocks for complex software (OpenFOAM, TensorFlow, WRF, …)
- Installation procedure can be controlled via easyconfig parameters
  - Additional configure options, commands to run before/after build or install command, ...
  - Generic easyblock + handful of defined easyconfig parameters is sufficient to install a lot of software
- GitHub repository: https://github.com/easybuilders/easybuild-easyblocks
- Easyblocks do not need to be part of the EasyBuild installation (see `--include-easyblocks`)

EasyBuild terminology: easyconfig file

- Text file that specifies what EasyBuild should install (in Python syntax)
- **Collection of values for easyconfig parameters** (key-value definitions)
- Filename typically ends in '`.eb`'
- Specific filename is expected in some contexts (when resolving dependencies)
  - Should match with values for `name`, `version`, `toolchain`, `versionsuffix`
  - `<name>-<version>-<toolchain><versionsuffix>.eb`
- GitHub repository: https://github.com/easybuilders/easybuild-easyconfigs

EasyBuild terminology: easystack file

- New concept since EasyBuild v4.3.2 (Dec’20), **experimental feature**
- Concise description for software stack to be installed (in YAML syntax)
- Basically **specifies a set of easyconfig files** (+ associated info)
- Still a work-in-progress, only basic functionality currently
- More Info: https://docs.easybuild.io/en/latest/Easystack-files.html
- My personal experience: Still a bit buggy, but a promising way to organise (re-)installation of a software stack

EasyBuild terminology: extensions

- **Additional software that can be installed *on top* of other software**
- Common examples: Python packages, Perl modules, R libraries, …
- Extensions is the general term we use for this type of software packages
- Can be installed in different ways:
    - As a stand-alone software packages (separate module)
    - In a bundle together with other extensions
    - As an actual extension, to provide a “batteries included” installation
- Feature can work together with Lmod to be able to find extensions included in a module easily but turned off at the moment because of problems with Cray Lmod 8.3.1

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

EasyBuild terminology: dependencies

- Software that is required to build/install or run other software
- Build dependencies: only required when building/installing software (not to use it)
  - Examples: CMake, pip, pkg-config, ...
- Run-time dependencies: (also) required to use the installed software
  - Examples: Python, Perl, R, ...
- Link-time dependencies: libraries that are required by software to link to, when linked statically or using RPATH
  - Examples: glibc, OpenBLAS, FFTW, ...
- Currently in EasyBuild: no distinction between link-time and run-time dependencies

EasyBuild terminology: modules

- Very overloaded term: kernel modules, Python modules, Perl modules …
- In EasyBuild context: “module” usually refers to an environment module file
  - Shell-agnostic specification of how to “activate” a software installation
  - Expressed in Tcl or Lua syntax (scripting languages)
  - Consumed by a modules tool (Lmod, Environment Modules, …)
- Other types of modules will be qualified explicitly (Python modules, etc.)
- EasyBuild automatically generates a module file for each installation

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

Bringing all EasyBuild terminology together

The EasyBuild **framework** leverages **easyblocks** to automatically build and install (scientific) software, potentially including additional **extensions**, using a particular compiler **toolchain**, as specified in **easyconfig files** which each define a set of **easyconfig parameters**.

EasyBuild ensures that the specified **(build) dependencies** are in place, and automatically generates a set of (environment) **modules** that facilitate access to the installed software.

An **easystack** file can be used to specify a collection of software to install with EasyBuild.

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_05_terminology/

Introduction to EasyBuild

Installation

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_06_installation/

Installing EasyBuild: requirements

- **Linux** as operating system (CentOS, RHEL, Ubuntu, Debian, SLES, …)
    - EasyBuild also works on macOS, but support is very basic
- **Python** 2.7 or 3.5+
    - Only Python standard library is required for core functionality of EasyBuild
    - Using Python 3 is highly recommended!
- An **environment modules tool** (module command)
    - Default is Lua-based Lmod implementation, highly recommended!
    - Tcl-based implementations are also supported

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_06_installation/

Installing EasyBuild: different options

- Installing EasyBuild using a standard Python installation tool
  - `pip install easybuild`
  - … or a variant thereof (`pip3 install --user`, using `virtualenv`, etc.)
  - May require additional commands, for example to update environment
- **Installing EasyBuild as a module, with EasyBuild *(recommended!)***
  - 3-step “bootstrap” procedure, via temporary EasyBuild installation using pip
- Development setup
  - Clone GitHub repositories: `easybuilders/easybuild-{framework,easyblocks,easyconfigs}`
  - Update `$PATH` and `$PYTHONPATH` environment variables

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_06_installation/

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

- **Step 1: Use pip to obtain a temporary installation of EasyBuild**

```
export TMPDIR=/tmp/$USER/easybuild
pip3 install --prefix $TMPDIR easybuild
# update environment to use this temporary EasyBuild installation
export PATH=$TMPDIR/bin:$PATH
export PYTHONPATH=$TMPDIR/lib/python3.6/site-packages:$PYTHONPATH
# instruct EasyBuild to use python3 command
export EB_PYTHON=python3
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_06_installation/

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

- **Step 2: Use EasyBuild to install EasyBuild (as a module) in home directory**

```
eb --install-latest-eb-release --prefix $HOME/easybuild
# and then clean up the temporary EasyBuild installation
rm -r $TMPDIR
```

- **Step 3: Load EasyBuild module to use final installation**

```
module use $HOME/easybuild/modules/all
module load EasyBuild
```

Approach on LUMI

- Each version of the LUMI software stack is bootstrapped to ensure that it can be rebuild on an "empty" system
- Tend to fix the version of EasyBuild for each LUMI stack to ensure that a rebuild of the already installed software is possible
- Use the bootstrapping process
  - There is no `pip` in the system Python so we call the `setup.py` script through Python
  - Then use that version to do a proper install in `partition/common`, using the configuration modules

Verifying the EasyBuild installation

- Check EasyBuild version:

```
eb --version
```

- Show help output (incl. long list of supported configuration settings)

```
eb --help
```

- Show the current (default) EasyBuild configuration:

```
eb --show-config
```

- Show system information:

```
eb --show-system-info
```

Updating EasyBuild

- Updating EasyBuild (in-place) that was installed with pip:

```
pip install --upgrade easybuild
```

  (+ additional options like `--user`, or using `pip3`, depending on your setup)

- Use current EasyBuild to install latest EasyBuild release as a module:

```
eb --install-latest-eb-release
```

  - This is *not* an in-place update, but a new EasyBuild installation!
  - You need to load (or swap to) the corresponding module afterwards:

```
module load EasyBuild/4.5.4
```

Introduction to EasyBuild

Configuring EasyBuild

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Configuring EasyBuild

- EasyBuild should work fine out-of-the-box if you are using Lmod as modules tool
- … but it will (ab)use `$HOME/.local/easybuild` to install software into, etc.
- It is ***strongly*** recommended to configure EasyBuild properly!
- Main questions you should ask yourself:
  - Where should EasyBuild install software (incl. module files)?
  - Where should auto-downloaded sources be stored?
  - Which filesystem is best suited for software build directories (I/O-intensive)?

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Primary configuration settings

- Most important configuration settings: (strongly recommended to specify the ones in **bold!**)
  - Modules tool + syntax (`modules-tool` + `module-syntax`)
  - **Software + modules installation path** (`installpath`)*
  - **Location of software sources "cache"** (`sourcepath`)*
  - **Parent directory for software build (work) directories** (`buildpath`)*
  - Location of easyconfig files archive (`repositorypath`)*
  - Search path for easyconfig files (`robot-paths` + `robot`)
  - Module naming scheme (`module-naming-scheme`)
- Several locations* (+ others) can be controlled at once via `prefix` configuration setting
  - Defaults are as if `--prefix=$HOME/.local/easybuild`
- *Full* list of EasyBuild configuration settings (~250) is available via `eb --help`

Configuration levels

- There are 3 different configuration levels in EasyBuild:
    - **Configuration files**
    - **Environment variables**
    - **Command line options to the eb command**
- Each configuration setting can be specified via each “level” (no exceptions!)
- Hierarchical configuration:
    - Configuration files override default settings
    - Environment variables override configuration files
    - `eb` command line options override environment variables

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

EasyBuild configuration files

- EasyBuild configuration files are in standard INI format (key=value)
- EasyBuild considers multiple locations for configuration files:
  - User-level: `$HOME/.config/easybuild/config.cfg` (or via `$XDG_CONFIG_HOME`)
  - System-level: `/etc/easybuild.d/*.cfg` (or via `$XDG_CONFIG_DIRS`)
  - See output of `eb --show-default-configfiles`
- Output produced by `eb --confighelp` is a good starting point
- Typically for “do once and forget” static configuration (like modules tool to use, ...)
- **EasyBuild configuration files and easyconfig files are very different things!**

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

$EASYBUILD_* environment variables

- Very convenient way to configure EasyBuild
- **There is an $EASYBUILD_* environment variable for each configuration setting**
  - Use all capital letters
  - Replace every dash (-) character with an underscore (_)
  - Prefix with `EASYBUILD_`
  - Example: `module-syntax` → `$EASYBUILD_MODULE_SYNTAX`
- Common approach: using a shell script or module file to (dynamically) configure EasyBuild
  - Which is what we do on LUMI with `EasyBuild-user`, `EasyBuild-production` and `EasyBuild-infrastructure`

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Command line options for `eb` command

- **Configuration settings specified as command line option always "win"**
- Use double-dash + name of configuration setting, like `--module-syntax`
- Some options have a corresponding shorthand (`eb --robot` == `eb -r`)
- In some cases, only command line option really makes sense (like `eb --version`)
- Typically used to control configuration settings for current EasyBuild session;

  for example: `eb --installpath /tmp/$USER`

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Inspecting the current configuration

- It can be difficult to remember how EasyBuild was configured
- Output produced by `eb --show-config` is useful to remind you
  - Shows configuration settings that are different from default
  - Always shows a couple of key configuration settings
  - Also shows on which level each configuration setting was specified
- Full current configuration: `eb --show-full-config`

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Inspecting the current configuration: fictitious example

```
$ cat $HOME/.config/easybuild/config.cfg
[config]
prefix=/apps

$ export EASYBUILD_BUILDPATH=/tmp/$USER/build

$ eb --installpath=/tmp/$USER --show-config
# Current EasyBuild configuration
# (C: command line argument, D: default value,
#  E: environment variable, F: configuration file)
buildpath      (E) = /tmp/example/build
containerpath  (F) = /apps/containers
installpath    (C) = /tmp/example
packagepath    (F) = /apps/packages
prefix         (F) = /apps
repositorypath (F) = /apps/ebfiles_repo
robot-paths    (D) = /home/example/.local/easybuild/easyconfigs
sourcepath     (F) = /apps/sources
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_07_configuration/

Introduction to EasyBuild

Basic usage

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Basic usage of EasyBuild

- **Use eb command to run EasyBuild**
- Software to install is usually specified via name(s) of easyconfig file(s), or easystack file
- `--robot` (`-r`) option is required to also install missing dependencies (and toolchain)
- Typical workflow:
  - Find or create easyconfig files to install desired software
  - Inspect easyconfigs, check missing dependencies + planned installation procedure
  - Double check current EasyBuild configuration
  - Instruct EasyBuild to install software (while you enjoy a coffee… or two)

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Specifying easyconfigs to use

- There a different ways to specify to the eb command which easyconfigs to use
    - Specific relative/absolute paths to (directory with) easyconfig files
    - Names of easyconfig files (triggers EasyBuild to search for them)
    - Easystack file to specify a whole stack of software to install (via `eb --easystack`)
- Easyconfig filenames only matter when missing dependencies need to be installed
    - “Robot” mechanism searches based on dependency specs + easyconfig filename
- `eb --search` can be used to quickly search through available easyconfig files

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Searching for easyconfigs

- EasyBuild has 2 options to search for an easyconfig
  - `eb --search` : Output with full paths
  - `eb -S` : Output grouped per repository, common part of the path replaced with a variable

```
$ eb --search openfoam-9
 * /appl/lumi/LUMI-EasyBuild-contrib/easybuild/easyconfigs/o/OpenFOAM/OpenFOAM-9-cpeGNU-21.08.eb
 * /appl/lumi/LUMI-EasyBuild-contrib/easybuild/easyconfigs/o/OpenFOAM/OpenFOAM-9-cpeGNU-21.12.eb
$ eb -S openfoam-9
CFGS1=/appl/lumi/LUMI-EasyBuild-contrib/easybuild/easyconfigs/o/OpenFOAM
 * $CFGS1/OpenFOAM-9-cpeGNU-21.08.eb
 * $CFGS1/OpenFOAM-9-cpeGNU-21.12.eb
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Searching for easyconfigs

- Search can also use regular expressions
  - But be careful that bash does not expand special characters!

```
$ eb -S '^gromacs-2021.*cpeGNU.*'
CFGS1=/appl/lumi/LUMI-EasyBuild-contrib/easybuild/easyconfigs/g/GROMACS
 * $CFGS1/GROMACS-2021-cpeGNU-21.08-PLUMED-2.7.2-CPU.eb
 * $CFGS1/GROMACS-2021.3-cpeGNU-21.08-CPU.eb
 * $CFGS1/GROMACS-2021.4-cpeGNU-21.12-PLUMED-2.7.4-CPU.eb
 * $CFGS1/GROMACS-2021.4-cpeGNU-21.12-PLUMED-2.8.0-CPU.eb
 * $CFGS1/GROMACS-2021.5-cpeGNU-21.12-CPU.eb
```

- Note that the easyconfigs that come with EasyBuild are not included in the path used for search and dependency resolution.

Inspecting easyconfigs via `eb --show-ec`

- To see the contents of an easyconfig file, you can use eb --show-ec
- No need to know where it is located, EasyBuild will do that for you!

```
$ eb --show-ec bzip2-1.0.8-cpeCray-21.12.e
…
name =     'bzip2'
version = ‘1.0.8’

homepage = 'https://www.sourceware.org/bzip2/’
…
toolchain = {'name': 'cpeCray', 'version': '21.12'}
toolchainopts = {'pic': True}

source_urls = ['https://sourceware.org/pub/%(name)s/']
sources =     [SOURCE_TAR_GZ]
patches =     ['bzip2-%(version)s-pkgconfig-manpath.patch']
…
```

Checking dependencies via `eb --dry-run`

To check which dependencies are required, you can use `eb --dry-run` (or `eb -D`):

- Provides overview of all dependencies (both installed and missing)
- Including compiler toolchain and build dependencies

```
$ eb SAMtools-1.14-cpeGNU-21.12.eb -D
…
CFGS=/appl/lumi
 * [x] $CFGS/mgmt/ebrepo_files/LUMI-21.12/LUMI-common/buildtools/buildtools-21.12.eb (module: buildtools/21.12)
 * [x] $CFGS/mgmt/ebrepo_files/LUMI-21.12/LUMI-L/cpeGNU/cpeGNU-21.12.eb (module: cpeGNU/21.12)
 * [x] $CFGS/mgmt/ebrepo_files/LUMI-21.12/LUMI-L/ncurses/ncurses-6.2-cpeGNU-21.12.eb (module: ncurses/6.2-cpeGNU-21.12)
…
 * [x] $CFGS/mgmt/ebrepo_files/LUMI-21.12/LUMI-L/Brotli/Brotli-1.0.9-cpeGNU-21.12.eb (module: Brotli/1.0.9-cpeGNU-21.12)
 * [x] $CFGS/mgmt/ebrepo_files/LUMI-21.12/LUMI-L/cURL/cURL-7.78.0-cpeGNU-21.12.eb (module: cURL/7.78.0-cpeGNU-21.12)
 * [ ] $CFGS/LUMI-EasyBuild-contrib/easybuild/easyconfigs/h/HTSlib/HTSlib-1.14-cpeGNU-21.12.eb (module: HTSlib/1.14-
cpeGNU-21.12)
 * [ ] $CFGS/LUMI-EasyBuild-contrib/easybuild/easyconfigs/s/SAMtools/SAMtools-1.14-cpeGNU-21.12.eb (module:
SAMtools/1.14-cpeGNU-21.12)
```

Checking *missing* dependencies via `eb --missing`

To check which dependencies are still *missing*, use `eb --missing` (or `eb -M`):

- Takes into account available modules, only shows what is still missing

```
$ eb SAMtools-1.14-cpeGNU-21.12.eb -M
2 out of 11 required modules missing:

* HTSlib/1.14-cpeGNU-21.12 (HTSlib-1.14-cpeGNU-21.12.eb)
* SAMtools/1.14-cpeGNU-21.12 (SAMtools-1.14-cpeGNU-21.12.eb)
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Inspecting software install procedures

- EasyBuild can quickly unveil how exactly it *would* install an easyconfig file
- Via `eb --extended-dry-run` (or `eb -x`)
- Produces detailed output in a matter of seconds
- Software is not actually installed, all shell commands and file operations are skipped!
- Some guesses and assumptions are made, so it may not be 100% accurate...
- Any errors produced by the easyblock are reported as being ignored
- Very useful to evaluate changes to an easyconfig file or easyblock!

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Inspecting software install procedures: example

```
$ eb HTSlib-1.14-cpeGNU-21.12.eb -x
...
[prepare_step method]
Defining build environment, based on toolchain (options) and specified dependencies...

Loading toolchain module...

module load cpeGNU/21.12

Loading modules for dependencies...

module load buildtools/21.12
module load zlib/1.2.11-cpeGNU-21.12
module load bzip2/1.0.8-cpeGNU-21.12
...
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Inspecting software install procedures: example

```
$ eb HTSlib-1.14-cpeGNU-21.12.eb -x

...

Defining build environment...

...

  export CC='cc'

  export CFLAGS='-O2 -ftree-vectorize -fno-math-errno’

...

configuring... [DRY RUN]

[configure_step method]

 running command "./configure --prefix=/users/kurtlust/LUMI-user-appl/SW/LUMI-

21.12/L/HTSlib/1.14-cpeGNU-21.12"

  (in /run/user/10012026/easybuild/build/HTSlib/1.14/cpeGNU-21.12/HTSlib-1.14)
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Inspecting software install procedures: example

```
$ eb HTSlib-1.14-cpeGNU-21.12.eb -x
...
building... [DRY RUN]
[build_step method]
  running command "make  -j 256"
  (in /run/user/10012026/easybuild/build/HTSlib/1.14/cpeGNU-21.12/HTSlib-1.14)
testing... [DRY RUN]
[test_step method]
installing... [DRY RUN]
...
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Inspecting software install procedures: example

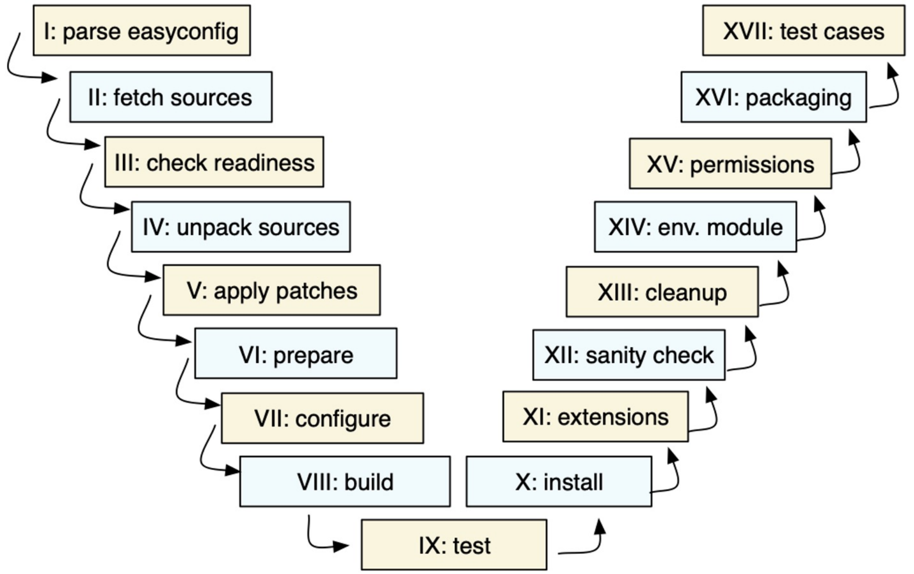
```
$ eb HTSlib-1.14-cpeGNU-21.12.eb -x
...
Sanity check paths - file ['files']
  * bin/bgzip
  * bin/tabix
  * lib/libhts.so
Sanity check paths - (non-empty) directory ['dirs']
  * include
Sanity check commands
  * bgzip --version
  * htsfile --version
  * tabix --version...
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Installing software with EasyBuild

- To install software with EasyBuild, just run the `eb` command:
  - `eb SAMtools-1.14-GCC-11.2.0.eb`
- If any dependencies are still missing, you will need to also use `--robot`:
  - `eb BCFtools-1.14-GCC-11.2.0.eb --robot`
- To see more details while the installation is running, enable trace mode:
  - `eb BCFtools-1.14-GCC-11.2.0.eb --robot --trace`
- To reinstall software, use `eb --rebuild` (or `eb --force`)

Step-wise installation procedure



- EasyBuild framework defines step-wise installation procedure, leaves some unimplemented
- Easyblock completes the implementation, override or extends installation steps where needed

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Using software installed with EasyBuild

- On LUMI modules are readily available (at least if the Lmod cache doesn't cause problems)
- Regular EasyBuild installation:

```
# inform modules tool about modules installed with EasyBuild
module use $HOME/easybuild/modules/all
```

- Then in both cases:

```
# check for available modules for BCFtools
module avail BCFtools
# load BCFtools module to "activate" the installation
module load BCFtools/1.14-GCC-11.2.0
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Stacking software installations

- It’s easy to “stack” software installed in different locations
- EasyBuild doesn’t care much where software is installed
- As long as the required modules are available to load, it can pick them up
- End users can easily manage a software stack on top of what’s installed centrally!

```
module use /easybuild/modules/all
eb --installpath $HOME/easybuild my-software.eb
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/1_Intro/1_08_basic_usage/

Using Easybuild

Troubleshooting

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_01_troubleshooting/

Troubleshooting failing installations

- Sometimes stuff still goes wrong...
- Being able to troubleshoot a failing installation is a useful/necessary skill
- Problems that occur include (but are not limited to):
  - Missing source or patch files
  - Checksum errors
  - Missing dependencies (perhaps overlooked required dependencies)
  - Failing shell commands (non-zero exit status)
  - Running out of memory or storage space
  - Compiler errors (or crashes)
- EasyBuild keeps a thorough log for each installation which is very helpful

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_01_troubleshooting/

Troubleshooting: error messages

- When EasyBuild detects that something went wrong, it produces an error
- Very often due to a shell command that produced a non-zero exit code...
- Sometimes the problem is clear directly from the error message:

```
== building...
== FAILED: Installation ended unsuccessfully (build directory: /tmp/example/example/1.0/GCC-
11.2.0):
build failed (first 300 chars): cmd "make" exited with exit code 2 and output:
/usr/bin/g++ -O2 -ftree-vectorize -march=native -std=c++14 -c -o core.o core.cpp
g++: error: unrecognized command line option '-std=c++14' (took 1 sec)
```

- In some cases, the error message itself does not reveal the problem...

Troubleshooting: log files

- EasyBuild keeps track of the installation in a detailed log file
- During the installation, it is stored in a temporary directory:

```
$ eb example.eb
== Temporary log file in case of crash /tmp/eb-r503td0j/easybuild-17flov9v.log
...
```

- Includes executed shell commands and output, build environment, etc.
- More detailed log file when debug mode is enabled (`debug` configuration setting)
- There is a log file per EasyBuild session, and one per performed installation
- **When an installation completes successfully,**
  the log file is copied to a subdirectory of the software installation directory

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_01_troubleshooting/

Troubleshooting: last log file

- EasyBuild has a nice trick to access the log file after a failed installation
  - `eb --last-log` returns the file name (including path) of that log file
  - So

```
vim $(eb --last-log)
```

Troubleshooting: navigating log files

- **EasyBuild log files are well structured, and fairly easy to search through**
- Example log message, showing prefix ("== "), timestamp, source location, log level:

```
== 2021-06-25 13:11:19,968 run.py:222 INFO running cmd:  make -j 9
```

- Different steps of installation procedure are clearly marked:

```
== 2021-06-25 13:11:48,817 example INFO Starting sanity check step
```

- To find actual problem for a failing shell command, look for patterns like:
  - ERROR
  - Error 1
  - error:
  - failure
  - not found
  - No such file or directory
  - Segmentation fault

Troubleshooting: inspecting the build directory

- EasyBuild leaves the build directory in place when the installation failed

```
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/build/example/1.0/GCC-11.2.0): build failed ...
```

- Can be useful to inspect the contents of the build directory for debugging
  - Rooted at `$EASYBUILD_BUILDPATH`
- For example:
  - Check `config.log` when `configure` command failed
  - Check `CMakeFiles/CMakeError.log` when `cmake` command failed

Troubleshooting: hands-on exercise

- **Highly recommended to try the exercise on tutorial website!**
- Try to fix the problems you encounter with the “broken” easyconfig file…

```
$ eb subread.eb
...
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/example/Subread/2.0.1/GCC-8.5.0): build failed (first 300 chars):
Couldn't find file subread-2.0.1-source.tar.gz anywhere, and downloading
it didn't work either...
Paths attempted (in order): ...
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_01_troubleshooting/

Using Easybuild

Creating easyconfig files

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Adding support for additional software

- Every installation performed by EasyBuild requires an easyconfig file
- Easyconfig files can be:
  - Included with EasyBuild itself (or obtained elsewhere)
  - Derived from an existing easyconfig (manually or automatic)
  - Created from scratch
- Most easyconfigs leverage a generic easyblock
- Sometimes using a custom software-specific easyblock makes sense...

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Easyblocks vs easyconfigs

- When can you get away with using an easyconfig leveraging a generic easyblock?
- When is a software-specific easyblock really required?
- Easyblocks are “implement once and forget”
- Easyconfig files leveraging a generic easyblock can become too involved (subjective)
- Reasons to consider implementing a custom easyblock:
  - 'critical' values for easyconfig parameters required to make installation succeed
  - custom (configure) options related to toolchain or included dependencies
  - interactive commands that need to be run
  - having to create or adjust specific (configuration) files
  - 'hackish' usage of a generic easyblock
  - complex or very non-standard installation procedure

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Writing easyconfig files

- Collection of easyconfig parameter definitions (Python syntax), collectively specify what to install
- Some easyconfig parameters are mandatory, and **must** always be defined: `name`, `version`, `homepage`, `description`, `toolchain`
- Commonly used easyconfig parameters (but strictly speaking not required):
  - `easyblock` (by default derived from software name)
  - `source_urls`, `sources`, `patches`, `checksums`
  - `dependencies`, `builddependencies`
  - `versionsuffix`
  - `preconfigopts`, `configopts`, `prebuildopts`, `buildopts`, `preinstallopts`, `installopts`
  - `sanity_check_paths`, `sanity_check_commands`

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Generating tweaked easyconfig files

- Trivial changes to existing easyconfig files can be done automatically
- Bumping software version: `eb example-1.0.eb --try-software-version 1.1`
- Changing toolchain (version): `eb example.eb --try-toolchain GCC,9.4.0`
- Changing specific easyconfig parameters (limited):

  ```
  eb --try-amend versionsuffix='-test'
  ```
- Note the "try" aspect: additional changes may be required to make installation work
- EasyBuild does save the so generated easyconfig files in the `easybuild` subdirectory of the software installation directory and in the easyconfig archive.

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Copying easyconfig files

- Small but useful feature: copy specified easyconfig file via `eb --copy-ec`
- Avoids the need to locate the file first via `eb --search`
- Typically used to create a new easyconfig using existing one as starting point
- Example:

```
$ eb --copy-ec SAMtools-1.11-GCC-10.2.0.eb SAMtools.eb
...
SAMtools-1.10-GCC-10.2.0.eb copied to SAMtools.eb
```

Hands-on: creating easyconfig files

- Step-wise example + exercise of creating an easyconfig file from scratch
- For a fictive software packages: eb-tutorial
- **Great exercise to work through these yourself!**

```
name = 'eb-tutorial'

version = '1.0.1'

homepage = 'https://easybuilders.github.io/easybuild-tutorial'

description = "EasyBuild tutorial example"
```

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_02_creating_easyconfig_files/

Using Easybuild

Using external modules from the Cray PE

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_03_external_modules/

External modules

- Modules not installed through EasyBuild
- Lack:
  - The metadata provided in modules generated by EasyBuild through the EBROOT and EBVERSION environment variables
  - A corresponding easyconfig file to tell EasyBuild about further dependencies
- Use:

```
dependencies = [('cray-fftw', EXTERNAL_MODULE)]
dependencies = [('cray-fftw/3.3.8.12', EXTERNAL_MODULE)]
```

- But metadata can be added through various mechanisms
  - Default metadata definition file included with EasyBuild (outdated)
  - Own metadata definition files
  - Discovery mechanism: EasyBuild recognises certain environment variables used by Cray modules

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_03_external_modules/

External modules: metadata

- External modules metadata file: `$EASYBUILD_EXTERNAL_MODULES_METADATA`
- Keys:
  - `name`: Equivalent EasyBuild module name
  - `version`: Software version provided by the module
  - `prefix`: Installation prefix of the software provided by the module
    - absolute path
    - or one that starts with the name of an environment variable specified by the module

```
[cray-fftw]
name = FFTW
prefix = FFTW_DIR/..
version = 3.3.8.10
```

Using Easybuild

Implementing easyblocks

- Text-only

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_04_implementing_easyblocks/

Advanced topics

Using EasyBuild as a library

- Text-only
- Investigating using EasyBuild this way to automatically generate documentation for recipes in our EasyBuild repository.

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/3_Advanced/3_01_easybuild_library/

Advanced topics

Using hooks to customise EasyBuild

- Not complete in these slides
- Used on LUMI to add LUST support as the `site_contacts` for all easyconfigs installed in the central installation.

Why hooks?

- Enforce site policies on easyconfig files
- Adding a parameter to the module if it is not present
    - E.g., on LUMI, site_contacts is added to all centrally installed software, pointing to the LUST support forms
- Modify the behaviour of a standard easyconfig file to adapt to the system while users can use the default easyconfig
    - Could use this to produce a working Open MPI setup on LUMI for the foss toolchain while users would be thinking they are using just the standard easyconfigs
    - Used at JSC to always take certain libraries from the system

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/2_Using/2_03_external_modules/

Advanced topics

Submitting installations as Slurm jobs

- Text-only
- Not really tested on LUMI, may need some compromises in the EasyBuild configuration

Advanced topics

Module naming schemes

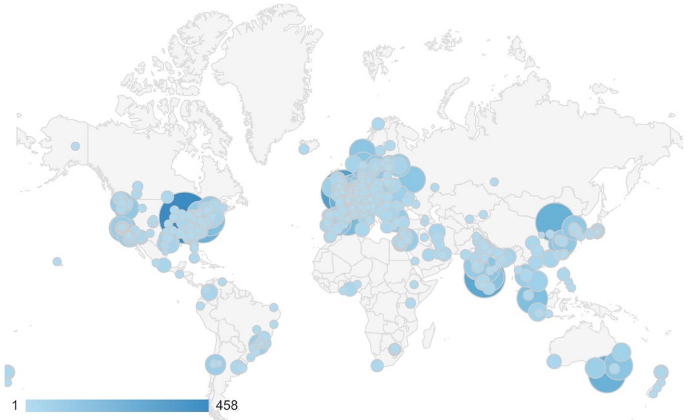
- Text-only

https://easybuilders.github.io/easybuild-tutorial/2022-CSC_and_LO/3_Advanced/3_04_module_naming_schemes/

Advanced topics

GitHub integration

The EasyBuild community



- Documentation read all over the world
- HPC sites, consortia, and companies
- Slack: >450 members, ~100 active members per week, 226K messages
- Regular online conf calls...and we even meet in person sometimes!

https://easybuilders.github.io/easybuild-tutorial/2021-isc21/community

Contributing to EasyBuild

There are several ways to contribute to EasyBuild, including:

- providing feedback
- reporting bugs
- joining the discussions (mailing list, Slack, conf calls)
- sharing suggestions/ideas for enhancements & additional features
- contributing easyconfigs, enhancing easyblocks,

  adding support for new software, implementing additional features, ...
- extending & enhancing documentation

GitHub integration features

- EasyBuild has strong integration with GitHub, which facilitates contributions
- Some additional Python packages required for this: GitPython, keyring
- Also required some additional configuration, incl. providing a GitHub token
- **Enables creating, updating, reviewing pull requests using eb command!**
- Makes testing contributions very easy (~2,000 easyconfig pull requests per year!)
- Extensively documented:

  https://docs.easybuild.io/en/latest/Integration_with_GitHub.html

Opening a pull request in 1, ~~2~~, ~~3~~

easybuild

```
$ mv sklearn.eb scikit-learn-0.19.1-intel-2017b-Python-3.6.3.eb
$ mv scikit-learn*.eb easybuild/easyconfigs/s/scikit-learn
$ git checkout develop && git pull upstream develop
$ git checkout -b scikit_learn_0191_intel_2017b
$ git add easybuild/easyconfigs/s/scikit-learn
$ git commit -m "{data}[intel/2017b] scikit-learn v0.19.1"
$ git push origin scikit_learn_0191_intel_2017b
```

+ log into GitHub to actually open the pull request (clickety, clickety...)

one single eb command

no git commands

no GitHub interaction

metadata is automatically derived from easyconfig

saves a lot of time!

```
eb --new-pr sklearn.eb
```

Topics we didn’t cover...

- Using RPATH linking
- Building Docker/Singularity container images with EasyBuild (experimental)

https://docs.easybuild.io - https://easybuild.io/tutorial

easybuild vs Spack

- **EasyBuild: GPLv2 license - Spack: MIT/Apache 2.0 license**
- no stable releases yet for Spack (< 1.0), EasyBuild is stable since 2012
- roughly on par w.r.t. amount of supported software (but differences w.r.t. which software)
- **targeted to different use cases: HPC support teams (EasyBuild) vs developers (Spack)**
- **fixed dependency/toolchain versions in EasyBuild vs flexible CLI in Spack**
- both support running on top of Python 2.7 and 3.5+
- macOS support in EasyBuild is limited (no toolchains/testing for macOS)
- **both projects are backed by an active & supportive community!**
- For a more detailed (but very outdated) comparison, see https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists

Just one more thing...

https://www.eessi-hpc.org   *https://eessi.github.io/docs*

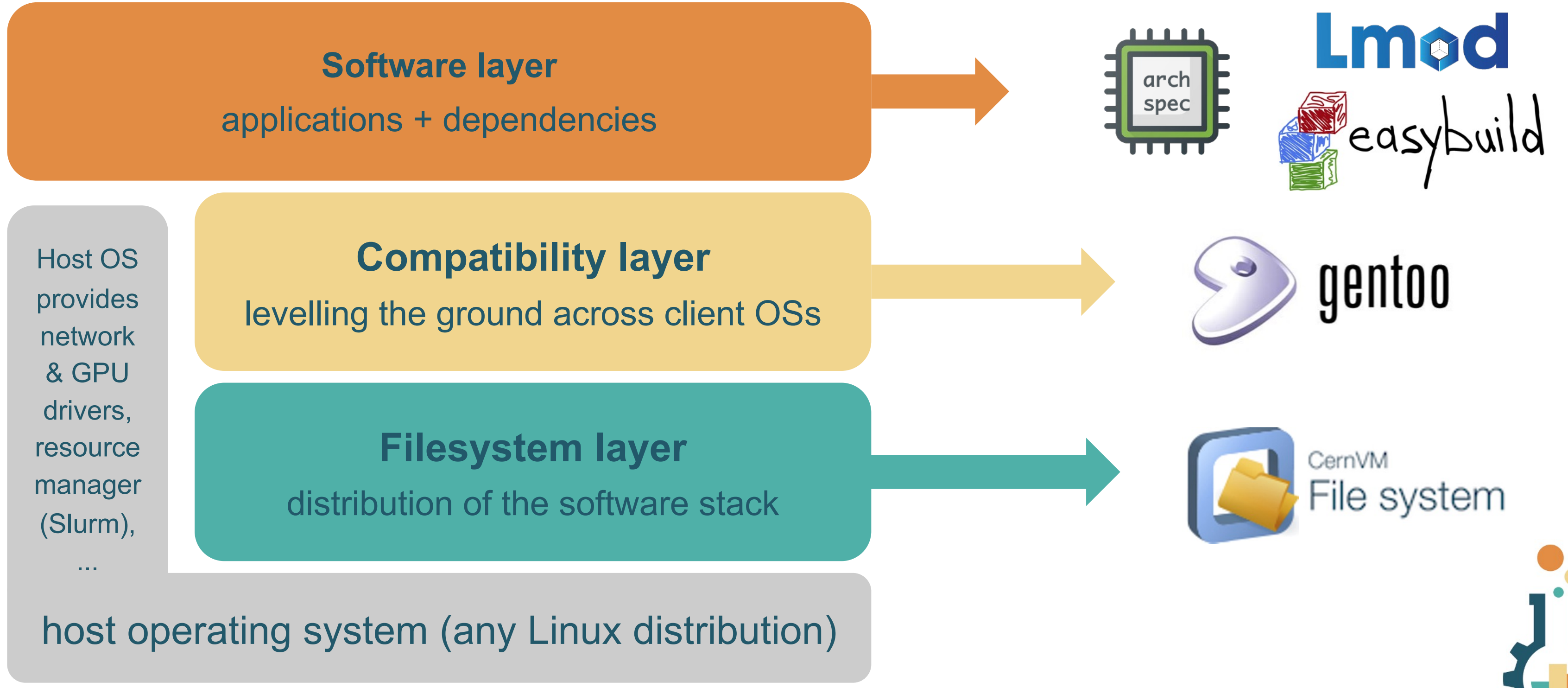
EESSI

- **European Environment for Scientific Software Installations (EESSI)**
- Collaboration between different European partners in HPC community
- Goal: building a **common** scientific software stack,
  for HPC systems & beyond (personal workstations, cloud instances, …)
- Heavily inspired by Compute Canada software stack
- Focus on performance, automation, testing, collaboration, ...

High-level overview of the EESSI project

https://www.eessi-hpc.org

https://eessi.github.io/docs


Software layer
applications + dependencies
arch spec
Lmod
easybuild
Host OS provides network & GPU drivers, resource manager (Slurm), ...
Compatibility layer
levelling the ground across client OSs
gentoo
Filesystem layer
distribution of the software stack
CernVM File system
host operating system (any Linux distribution)

EESSI

EESSI VS E4S

- EESSI is based on EasyBuild while E4S is based on Spack
- Different distribution mechanisms
  - EESSI strictly via CernVM FS
    - Can use CernVM FS from a container, but performance may be slow without close enough cache
  - E4S via build caches with binaries for multiple platforms, or rebuild otherwise + containers for Docker, singularity, Shifter and CharlieCloud