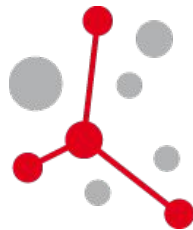


easybuild



ISC High Performance
The HPC Event.

EasyBuild tutorial

ISC'22

Sebastian Achilles (JSC) - Kurt Lust (Univ. of Antwerp)

Alan O'Cais (CECAM) - Kenneth Hoste (HPC-UGent)

29 May 2022

<https://easybuild.io/tutorial/isc22>

- [14:00-14:10] **Practical information w.r.t. prepared environment for hands-on**
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

Practical information

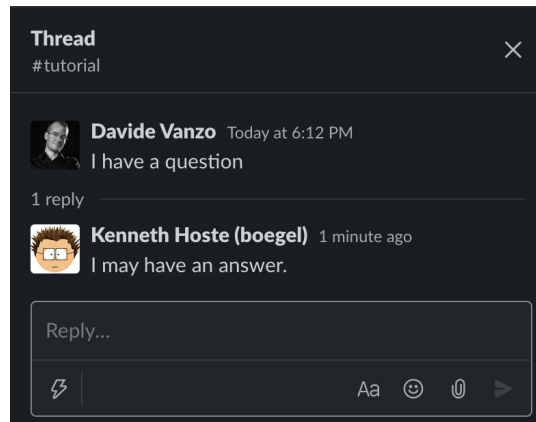
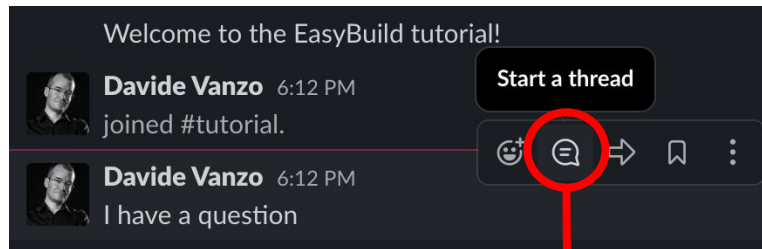


- **Sunday 29th of May 2022, 14:00 - 18:00 CEST**
- Tutorial website: <https://easybuild.io/tutorial/isc22>
- Please join the [#tutorial-isc22](#) channel in the EasyBuild Slack to ask questions!
- Prepared environment for hands-on demos & exercises

Q&A via dedicated channel in EasyBuild Slack



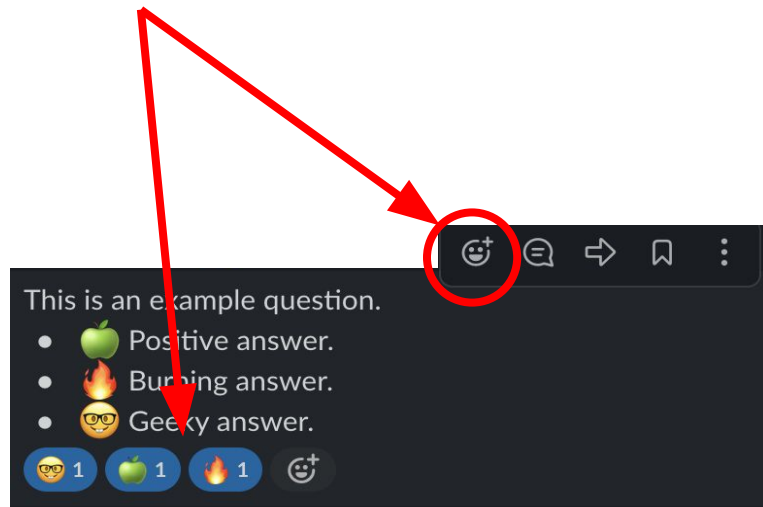
- Questions or problems?
Speak up in #tutorial-isc22 on EasyBuild Slack!
- Join via <https://easybuild.io/join-slack>
- Use threads to avoid overflowing the channel!



Emoji polls in Slack



- Small polls will be posted in the #tutorial-isc22 Slack channel.
- **Vote** for one (or more) answers using the corresponding emoji !



Prepared environment



- Small Rocky 8 cluster (in the cloud)
- **You need to create an account!**
 - Signup: <https://mokey.isc22.learnhpc.eu/auth/signup>
 - Accounts will only be approved for access on 29 May 2022, so **please record your username/password !**
 - [Reset Password](#) link does **not** work, instead raise any login problem in Slack
- Access via ssh or web browser (**pick one and stick to it!**)
 - Shell access: `ssh isc22.learnhpc.eu`
 - Via browser: <https://isc22.learnhpc.eu>
- System will be up until the end of the conference (18:15 CEST, Thursday 2 June 2022)

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] **Introduction to EasyBuild: scope & terminology**
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

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
 - First released publicly in Apr'12 (version 0.5)
 - EasyBuild 1.0.0 released in Nov'12 (during SC12)
 - Worldwide community has grown around it since then!

<https://easybuild.io>

<https://docs.easybuild.io>

<https://github.com/easybuilders>

<https://easybuild.io/join-slack>

Twitter: [@easy_build](#)

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

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 installation prefix)
- 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)

Key features of EasyBuild (2/2)



- Integrates with various other tools (Lmod, Singularity, FPM, Slurm, GC3Pie, ...)
- **Actively developed and supported by worldwide community**
- **Frequent stable releases** since 2012 (every 6 - 8 weeks)
- **Comprehensive testing:** unit tests, testing contributions, regression testing
- **Various support channels** (mailing list, Slack, conf calls) + yearly user meetings

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
- 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

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 may still run into compiler errors (unless somebody worked around it already)

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, ...

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 implemented currently
- More info: <https://docs.easybuild.io/en/latest/Easystack-files.html>

EasyBuild terminology: extensions



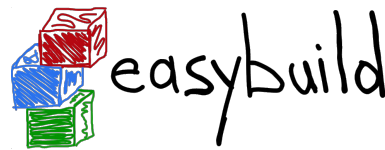
- **Additional software that can be installed *on top* of other software**
- Common examples: Python packages, Perl modules, R libraries, ...
- *Extensions* is the general term we use for this type of software packages
- Can be installed in different ways:
 - As a stand-alone software packages (separate module)
 - In a bundle together with other extensions
 - As an actual extension, to provide a “batteries included” installation

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
 - Examples: glibc, OpenBLAS, FFTW, ...
- Currently in EasyBuild: no distinction between link-time and run-time dependencies

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 toolchains in EasyBuild community:
 - `foss`: GCC + OpenMPI + (FlexiBLAS +) OpenBLAS + FFTW
 - `intel`: Intel compilers + Intel MPI + Intel MKL

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

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.

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] **Installing & configuring EasyBuild + basic usage**
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

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

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

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=$TMPDIR/lib/python3.9/site-packages:$PYTHONPATH  
  
# instruct EasyBuild to use python3 command  
  
export EB_PYTHON=python3
```

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
```

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
```

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)?

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 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
- *Full* list of EasyBuild configuration settings (~270) 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

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!**

\$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

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`

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`

Inspecting the current configuration: 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
```

Minimal EasyBuild configuration for hands-on



- **Use home directory as main prefix directory**

(location for installed software, downloaded sources, ...)

```
export EASYBUILD_PREFIX=$HOME/easybuild
```

- **Use *local* temporary directory for build directories** (important!)

```
export EASYBUILD_BUILDPATH=/tmp/$USER
```

- **Ensure prepared software stack is visible** via “module avail”

```
module use /easybuild/modules/all
```

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)

Specifying easyconfigs to use



- There are 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

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 TensorFlow-2.6.0-foss-2021a.eb
```

```
easyblock = 'PythonBundle'
```

```
name = 'TensorFlow'
```

```
version = '2.6.0'
```

```
homepage = 'https://www.tensorflow.org/'
```

```
description = "An open-source software library for Machine Intelligence"
```

```
toolchain = {'name': 'foss', 'version': '2021a'}
```

```
toolchainopts = {'pic': True}
```

```
...
```

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-GCC-11.2.0.eb -D
```

```
...
* [x] $CFGS/n/ncurses/ncurses-6.2-GCCcore-11.2.0.eb (module: ncurses/6.2-GCCcore-11.2.0)
* [x] $CFGS/p/pkg-config/pkg-config-0.29.2.eb (module: pkg-config/0.29.2)
* [x] $CFGS/o/OpenSSL/OpenSSL-1.1.eb (module: OpenSSL/1.1)
* [x] $CFGS/c/cURL/cURL-7.78.0-GCCcore-11.2.0.eb (module: cURL/7.78.0-GCCcore-11.2.0)
* [ ] $CFGS/s/SAMtools/SAMtools-1.14-GCC-11.2.0.eb (module: SAMtools/1.14-GCC-11.2.0)
```

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 PyTables-3.6.1-foss-2021b.eb -M
```

```
3 out of 69 required modules missing:
```

- * LZO/2.10-GCCcore-11.2.0 (LZO-2.10-GCCcore-11.2.0.eb)
- * Blosc/1.21.1-GCCcore-11.2.0 (Blosc-1.21.1-GCCcore-11.2.0.eb)
- * PyTables/3.6.1-foss-2021b (PyTables-3.6.1-foss-2021b.eb)

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!

Inspecting software install procedures: example



```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
```

```
...
```

```
preparing... [DRY RUN]
```

```
[prepare_step method]
```

```
Defining build environment, based on toolchain (options) and specified dependencies...
```

```
Loading toolchain module...
```

```
module load GCC/11.2.0
```

```
Loading modules for dependencies...
```

```
module load bzip2/1.0.8-GCCcore-11.2.0
```

```
module load zlib/1.2.11-GCCcore-11.2.0
```

```
module load XZ/5.2.5-GCCcore-11.2.0
```

Inspecting software install procedures: example



```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
```

```
...
```

```
Defining build environment...
```

```
...
```

```
export CXX='g++'
```

```
export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno -fPIC'
```

```
...
```

```
configuring... [DRY RUN]
```

```
[configure_step method]
```

```
running command "./bootstrap.sh --with-toolset=gcc
```

```
--prefix=/tmp/example/Boost/1.77.0-GCC-11.2.0 --without-libraries=python,mpi"
```

```
(in /tmp/example/build/Boost/1.77.0/GCC-11.2.0/Boost-1.77.0)
```

Inspecting software install procedures: example



```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
```

```
...
```

```
[sanity_check_step method]
```

```
Sanity check paths - file ['files']
```

```
  * lib/libboost_system-mt-x64.so
```

```
  * lib/libboost_system.so
```

```
  * lib/libboost_thread-mt-x64.so
```

```
Sanity check paths - (non-empty) directory ['dirs']
```

```
  * include/boost
```

```
Sanity check commands
```

```
  (none)
```

```
...
```

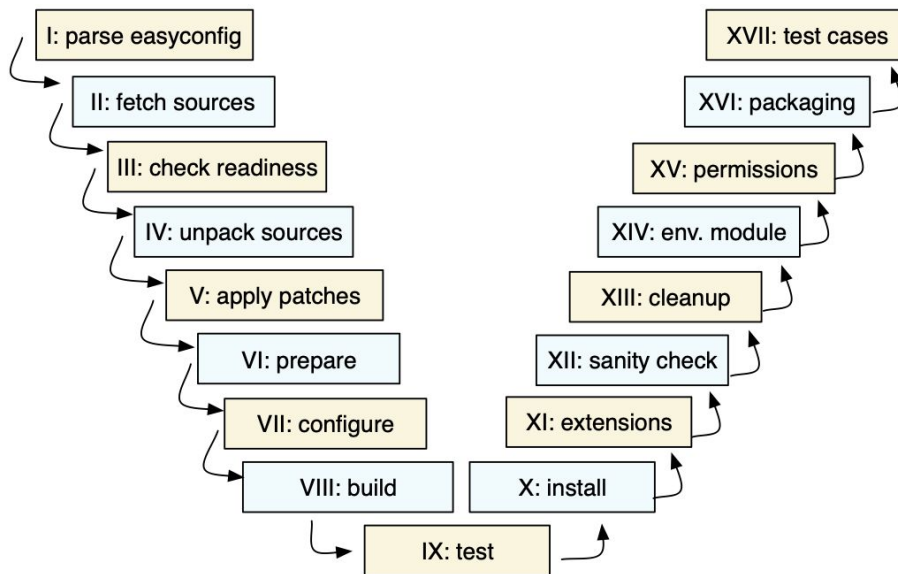

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] **Installing software with EasyBuild + troubleshooting**
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

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

Using software installed with EasyBuild



To use the software you installed with EasyBuild, load the corresponding module:

```
# inform modules tool about modules installed with EasyBuild

module use $HOME/easybuild/modules/all

# 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
```

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
```

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 files
 - 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

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**

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:

```
== 2022-05-25 13:11:19,968 run.py:222 INFO running cmd:  make -j 9
```

- Different steps of installation procedure are clearly marked:

```
== 2022-05-25 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
- For example:
 - Check `config.log` when `configure` command failed
 - Check `CMakeFiles/CMakeError.log` when `cmake` command failed (good luck...)

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/2.0.3/GCC-8.5.0): build failed (first 300 chars):  
Couldn't find file subread-2.0.3-source.tar.gz anywhere, and downloading  
it didn't work either...
```

```
Paths attempted (in order): ...
```

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] **Adding support for additional software**
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

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...

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

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)
 - `versionsuffix`
 - `source_urls`, `sources`, `patches`, `checksums`
 - `dependencies`, `builddependencies`
 - `preconfigopts`, `configopts`, `prebuiltopts`, `buildopts`, `preinstallopts`, `installopts`
 - `sanity_check_paths`, `sanity_check_commands`

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,11.2.0`
- Changing specific easyconfig parameters (limited): `eb --try-amend ...`
- 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.

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-1.14-GCC-11.2.0.eb SAMtools.eb
```

```
...
```

```
SAMtools-1.14-GCC-11.2.0.eb copied to SAMtools.eb
```

Hands-on: creating easyconfig files



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

```
name = 'eb-tutorial'
```

```
version = '1.0.1'
```

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

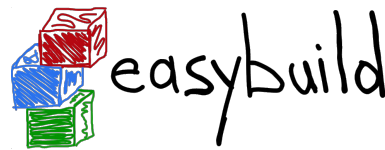
```
description = "EasyBuild tutorial example"
```

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] **(coffee break)**
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

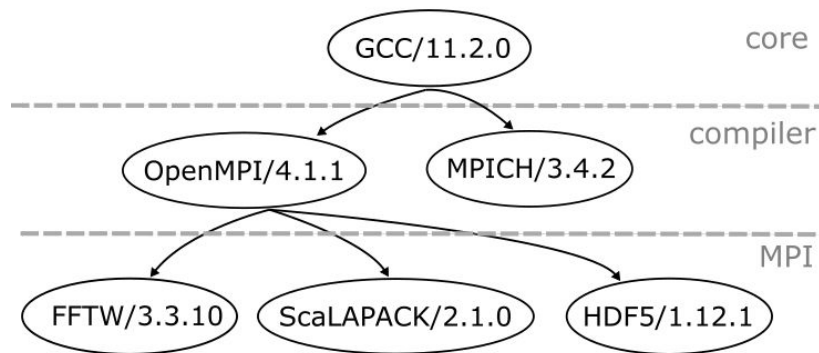


- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] **Module naming schemes (incl. hierarchical)**
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

Flat vs hierarchical module naming schemes



- Handful of supported module naming schemes (MNS), EasyBuildMNS is the default
- Flat module naming scheme (like EasyBuildMNS)
 - Clear mapping of easyconfig filename to name of generated module file
 - All modules immediately available for loading
- Hierarchical scheme typically has 3 levels
 - **core** level for things like compilers
 - **compiler** level
 - **MPI** level
 - Use “gateway modules” to access different levels



Pros and cons of using a flat vs hierarchical MNS



- Flat MNS
 - ± all modules visible (can be overwhelming)
 - + guaranteed unique
 - long module names that can be confusing
 - potential compatibility issues unless you are careful
- Hierarchical MNS
 - + short/clean module names (and no visible toolchains)
 - ± less visible modules (need to use `module spider+module avail`)
 - ± automatic swapping with Lmod when changing compiler/mpi
 - + modules that can be loaded are compatible with each other
 - requires gateway modules which might have little meaning for users

Custom module naming schemes with EasyBuild



- You can also create your own module naming scheme (e.g., lower-case only)
 - Implement Python class that derives from the general `ModuleNamingScheme` class
 - Best to start from one of the existing schemes
 - There are (a lot) more things to tweak with hierarchical module naming schemes
- To configure EasyBuild to use your custom module naming scheme:

```
export EASYBUILD_INCLUDE_MODULE_NAMING_SCHEMES=$HOME/easybuild/example_mns.py
export EASYBUILD_MODULE_NAMING_SCHEME=ExampleMNS
```

- Use dry-run mode to test it, e.g.,

```
eb SciPy-bundle-2021.10-foss-2021b.eb -D
```

Hands-on example: installing HDF5 in an HMNS



- **We must avoid mixing modules from a flat and hierarchical MNS!**

```
module unuse $MODULEPATH
```

- Configure our setup to reuse the existing software installations

```
export EASYBUILD_INSTALLPATH_SOFTWARE=/easybuild/software
```

```
export EASYBUILD_MODULE_NAMING_SCHEME=HierarchicalMNS
```

```
export EASYBUILD_INSTALLPATH_MODULES=$HOME/hmns/modules
```

- Re-generate all modules for HDF5 using the new scheme (42 modules)

```
eb HDF5-1.12.1-gompi-2021b.eb --robot --module-only
```

- Explore the new hierarchy

```
module use $HOME/hmns/modules/all/Core
```


- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] **Integration of EasyBuild in JSC, EESSI, and LUMI**
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)



by Sebastian Achilles



- JSC is a German supercomputing centre since 1987
 - About 250 experts for all aspects of supercomputing and simulation sciences



- JSC is a German supercomputing centre since 1987
 - About 250 experts for all aspects of supercomputing and simulation sciences
- Currently 3 primary systems:
 - JUWELS - 70 Petaflops, #8 in Top500 ([modular supercomputing](#))
 - JURECA-DC - 3.54 (CPU) + 14.98 (GPU) + 5 (KNL) Petaflops
 - JUSUF - AMD, V100 GPU. Interactive workflows and community services



EasyBuild at JSC



- Used for production software stack at JSC since 2014



EasyBuild at JSC



- Used for production software stack at JSC since 2014
- Geared towards average user experience
 - Hide lots of indirect software
 - Lots of toolchains => Module hierarchy
 - Renaming some modules, Lmod tweaks



EasyBuild at JSC



- Used for production software stack at JSC since 2014
- Geared towards average user experience
 - Hide lots of indirect software
 - Lots of toolchains => Module hierarchy
 - Renaming some modules, Lmod tweaks
- Custom MNS, toolchains, easyconfigs, easyblocks
 - Maintenance and contribution issue
 - Working hard to minimise this



Upgrading and retiring software



- Provide latest software to new projects by default
 - **Stages** concept
 - Updates once per year
 - Encourages users to adopt latest software & dependencies (performance, bug fixes,...)



Upgrading and retiring software



- Provide latest software to new projects by default
 - **Stages** concept
 - Updates once per year
 - Encourages users to adopt latest software & dependencies (performance, bug fixes,...)
- Give indirect access to "retired" software



Leveraging hooks for users & maintainers



- Very powerful alternative to customisations
 - Much more automated and flexible
 - Easier to maintain (particularly for easyconfigs)



Leveraging hooks for users & maintainers



- Very powerful alternative to customisations
 - Much more automated and flexible
 - Easier to maintain (particularly for easyconfigs)
- Hooks to enable user space installations
 - Guide people on how to do this “properly”
 - Installation hierarchy: system → group → user



Integration of EasyBuild in EESSI



Presented by Sebastian Achilles

Optimised scientific software everywhere without building or tuning: that's EESSI!

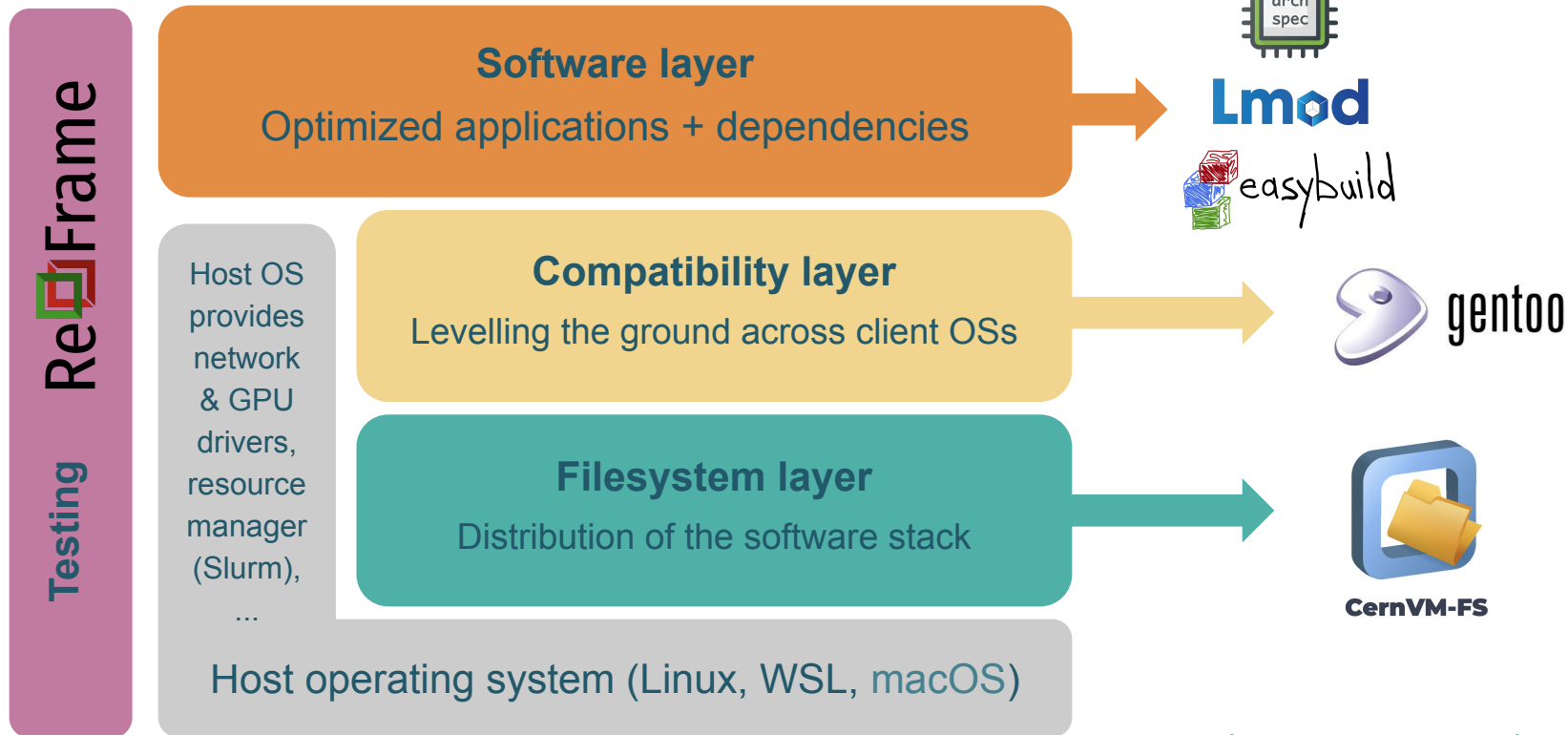


- The challenge:
 - Same software everywhere (HPC, Cloud, servers, laptops)
 - Optimized for specific CPUs, well tested, works on different OSs
 - Plug 'n play, limited setup
- The solution: EESSI - European Environment for Scientific Software Installations
 - "Streams" (scientific) software installations on-demand
 - Any machine, anywhere, nearly instantly available

<https://www.eessi-hpc.org>

<https://eessi.github.io/docs>

High-level overview of the EESSI project



<https://www.eessi-hpc.org>

<https://eessi.github.io/docs>

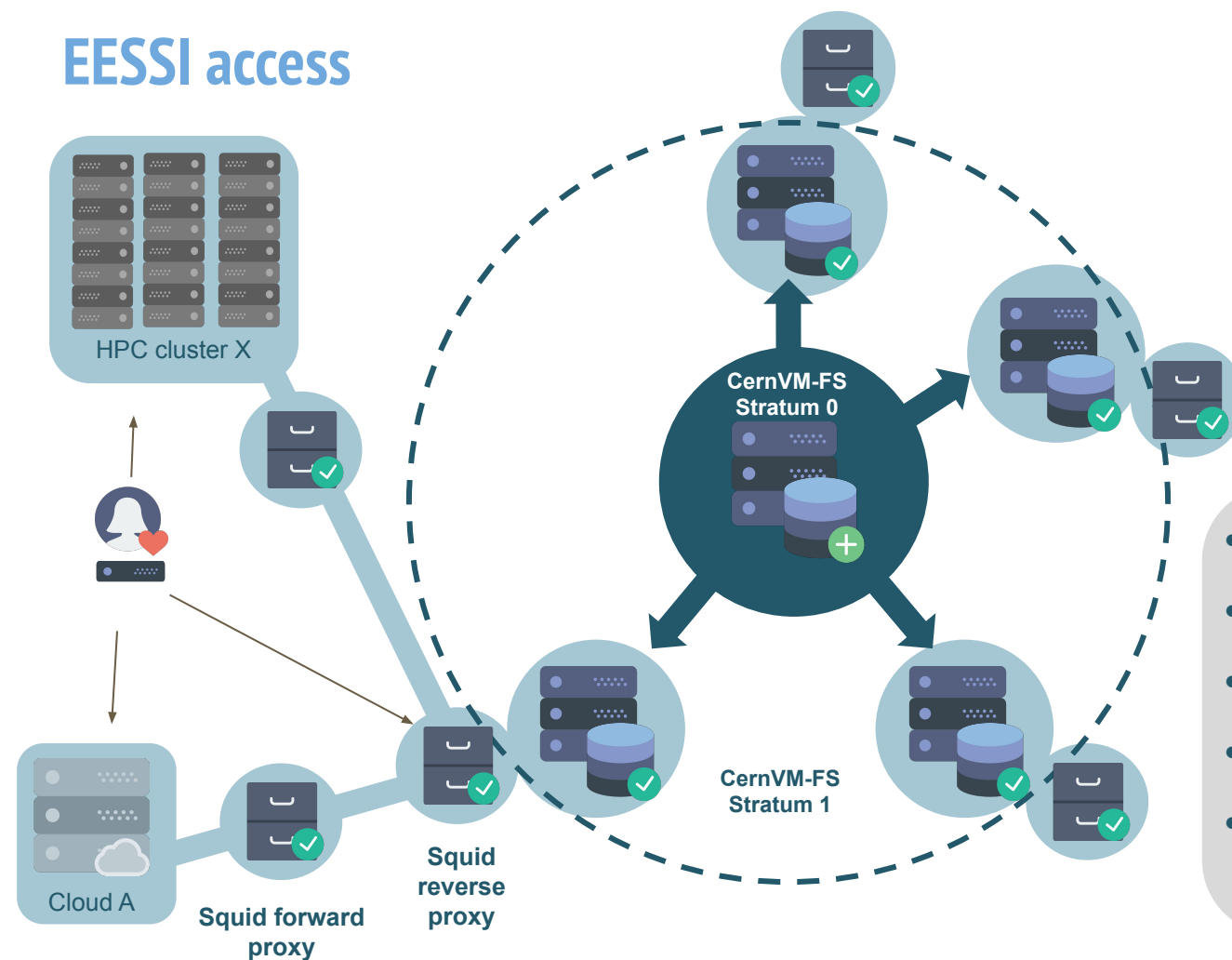
EESSI access



CernVM-FS

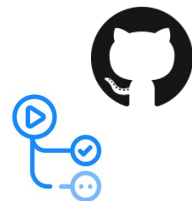
<https://cvmfs.readthedocs.io>

- Global distribution of software installations
- Centrally managed software stack
- Redundant network of “mirrors”
- Multiple levels of caching
- **Same software stack everywhere:**
laptops, HPC clusters, cloud VMs, ...



Leveraging EESSI in different scenarios

- If EESSI is already available: just set up your environment by sourcing a script
- As a **system administrator**, to make EESSI available:
 - Only need to install CernVM-FS + EESSI configuration package
 - Should also consider setting up squid proxy and maybe own Stratum-1 server
- As an **end user on an HPC system**, to access EESSI without having admin rights:
 - Run a container image via Singularity that includes CernVM-FS to access EESSI
 - See instructions at <https://eessi.github.io/docs/pilot>
- As a **software developer** in a CI environment like GitHub Actions
 - Use EESSI action in your workflow to leverage the available software in your tests





- EESSI ***not ready for production yet***, but testing + feedback is welcome!
- Website: <https://www.eessi-hpc.org>
- Documentation: <https://eessi.github.io/docs>
- Introduction to EESSI (EUM'21): https://www.youtube.com/watch?v=1CXwzIW_MsU
- **Join the EESSI mailing list and Slack:** <https://www.eessi-hpc.org/join>
- Monthly update meetings, open to join for anyone interested
<https://github.com/EESSI/meetings/wiki>
- EESSI hackathons (Dec'21 + Jan'22, plans for more):
<https://github.com/EESSI/hackathons>

Integration of EasyBuild in LUMI

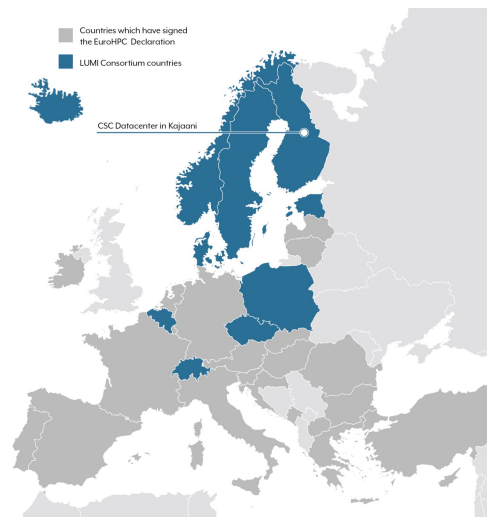


by Kurt Lust

LUMI User Support Team (LUST) & University of Antwerp

What is LUMI?

- LUMI is one of the EuroHPC JU pre-exascale systems
- Located in the CSC Kajaani data centre, hosted by a consortium of 10 countries who shared the investment with EuroHPC JU.
- HPE Cray EX system using SUSE Linux/COS and the HPE Cray Programming Environment (PE)
- Compute resources:
 - GPU partition: 2560 nodes with 1 AMD Trento CPU and 4 AMD MI250X GPUs
 - GPU-first node, SlingShot 11 interconnect attached to the GPUs
 - Cache-coherent unified memory
 - CPU partition: 1536 nodes with 2 64-core AMD Zen3 CPUs
 - Small interactive data analysis and visualisation partition (8 CPU-only nodes and 8 nodes with NVIDIA GPU)
 - OpenShift/Kubernetes partition
 - Lustre storage + Ceph object storage



Challenges

- Integrate with the HPE Cray PE, which is installed with the OS and not with the user applications stack.
 - EasyBuild common toolchains pose problems and have little support for AMD GPUs
- Heterogeneous environment and fast evolving software
 - Software stack updates measured in updates/year rather than years/update
- Distributed support effort
 - Central LUMI User Support Team only 9 FTE, and they are employees of institutions in the consortium countries and not of CSC
 - Consortium countries should also provide support
- Combining distributed user management with a small central support team with little access to user data creates a software license management nightmare
- Need for customisation

Building block 1: Lmod to organise software stacks

- Versatile and well supported in EasyBuild (and Spack), and by the HPE Cray PE.
- Used Lmod hierarchy to implement software stacks
 - CrayEnv: "enriched" Cray Programming Environment
 - Management of Cray PE target modules
 - Some additional tools on top of the OS
 - LUMI software stack: 2-level hierarchy
 - Versions aligned with the versions of the Cray PE (21.08, 21.12)
 - Second level: partition module loads stack for a particular architecture
 - Automatic selection of the partition module, but can be overwritten, e.g., for cross-compiling
 - Meta-partitions for special needs, e.g., software installed once for all architectures

Building block 2: EasyBuild

- EasyBuild gives a very precise description of the installation process
 - Hence a good way of passing installation instructions to someone
- Configuration of each individual installation fully described by easyconfig file, not by command line arguments
- Configuration module integrates EasyBuild with the LUMI software stack
 - Environment variable points to the user installation
 - User installation in the module search path
 - 3 EasyBuild configuration modules configure EasyBuild to install software in the right location:
 - EasyBuild-production, EasyBuild-infrastructure : system stack
 - EasyBuild-user for the user configuration
- Fix the version of EasyBuild for each software stack

Custom toolchains

- Common toolchains not fit for use on LUMI
 - Have support for the Cray and AMD compilers
 - Open MPI (foss) difficult to configure for LUMI, and no AMD GPU support anyway
 - Intel compilers have become a problem on AMD hardware
- Implement custom toolchains on top of Cray PE compilers
 - Build on CSCS implementation and an older implementation included in EasyBuild, but made several refinements
 - Compilers etc. not installed through EasyBuild
 - Replace the top level Cray PE module (PrgEnv-*) with one generated and managed through EasyBuild but otherwise use modules on the system

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:

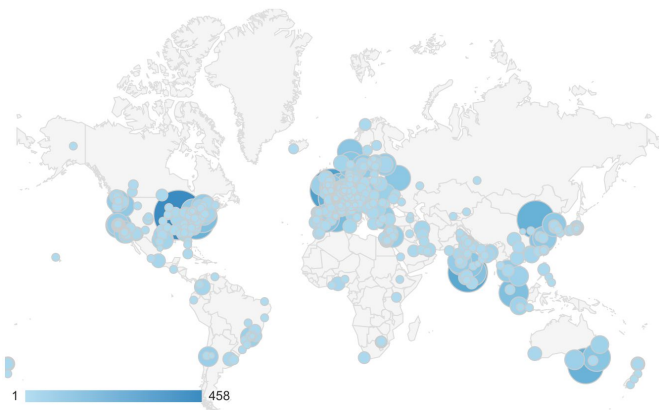
```
dependencies = [('cray-fftw', EXTERNAL_MODULE)]  
dependencies = [('cray-fftw/3.3.8.12', EXTERNAL_MODULE)]
```
- But metadata can be added through various mechanisms
 - Default metadata definition file included with EasyBuild (outdated)
 - Own metadata definition files
 - Discovery mechanism: EasyBuild recognises certain environment variables used by Cray modules

Software-specific easyblocks

- Probably the major nuisance when using EasyBuild on Cray systems
 - Several easyblocks contain code that only recognises certain compiler toolchains and abort for others
 - Some easyblocks detect dependencies through module names rather than EBROOT/EBVERSION variables and hence may fail for external modules
- Maintenance is an issue
 - Contributing back no guarantee that the support is maintained as testing is impossible in the EasyBuild test environment
 - But then you have to track changes yourself
- Tend to follow the CSCS approach and use generic easyblocks wherever possible, and "fatter" easyconfig files.

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] **The EasyBuild community + contributing to EasyBuild**
- [17:45-18:00] Q&A + closing remarks (incl. quick comparison with other tools)

The EasyBuild community



EasyBuild User Meeting 2020 (Barcelona)



- Documentation is read all over the world
- HPC sites, consortia, and companies
- Slack: >600 members, ~110 active members per week, 277K messages
- Regular online conf calls... and we even meet in person sometimes!



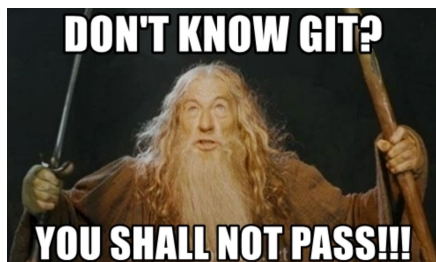
Contributing to EasyBuild



There are several ways to contribute to EasyBuild, including:

- Providing feedback (positive or negative)
- 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 requires some additional configuration, incl. providing a GitHub token
- **Enables creating, updating, reviewing pull requests using `eb` command!**
- Makes testing contributions very easy (~2,500 easyconfig pull requests per year!)
- Extensively documented:

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

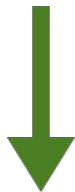
Opening a pull request in 1, 2, 3



```
$ 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`

- [14:00-14:10] Practical information w.r.t. prepared environment for hands-on
- [14:10-14:30] Introduction to EasyBuild: scope & terminology
- [14:30-14:50] Installing & configuring EasyBuild + basic usage
- [14:50-15:30] Installing software with EasyBuild + troubleshooting
- [15:30-16:00] Adding support for additional software
- [16:00-16:30] *(coffee break)*
- [16:30-16:50] Module naming schemes (incl. hierarchical)
- [16:50-17:30] Integration of EasyBuild in JSC, EESSI, and LUMI
- [17:30-17:45] The EasyBuild community + contributing to EasyBuild
- [17:45-18:00] **Q&A + closing remarks (incl. quick comparison with other tools)**

Topics we didn't cover...



- Implementing easyblocks
- Using easystack to install a whole stack at once
- Using RPATH linking
- Using EasyBuild as a library
- Implementing hooks to customize EasyBuild
- Submitting installations as jobs on a cluster
- Building Docker/Singularity container images with EasyBuild (experimental)

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



- **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 somewhat outdated) comparison, see https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists

Questions?



- Website: <https://easybuild.io>
- Documentation: <https://docs.easybuild.io>
- Tutorials: <https://easybuild.io/tutorial>
- Yearly EasyBuild User Meeting: <https://easybuild.io/eum>
- Getting help:
 - Mailing list: <https://lists.ugent.be/www/subscribe/easybuild>
 - Slack: <https://easybuild.slack.com> - <https://easybuild.io/join-slack>
 - Bi-weekly conference calls: <https://github.com/easybuilders/easybuild/wiki/Conference-calls>