Introduction to EasyBuild

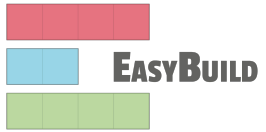
EasyBuild/EESSI/Spack tutorial/hackathon @ Bologna

19 May 2026

Kenneth Hoste (Ghent University, Belgium)

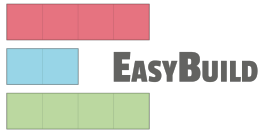
kenneth.hoste@ugent.be

Show of hands



- Who has heard about EasyBuild before?
- Who knows what EasyBuild is?
- Who has used EasyBuild to install software?
- Who is currently using EasyBuild to install software?
- Who is *only* using EasyBuild to install software?

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
- 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)
  - Over 110 stable releasese since then (latest: EasyBuild 5.3.0)
  - Worldwide community has grown around it (>1,100 members on EasyBuild Slack)

https://easybuild.io

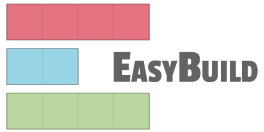
https://docs.easybuild.io

https://blog.easybuild.io

https://github.com/easybuilders

https://easybuild.io/join-slack

EasyBuild in a nutshell

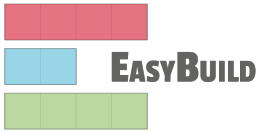


- **Tool** to provide a ***consistent and well performing*** scientific software stack
- **Uniform interface** for installing scientific software on HPC systems
- Saves time by ***automating*** tedious, boring and repetitive tasks
- Can **empower researchers** to self-manage their software stack
- **A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

Key features of EasyBuild (1/2)

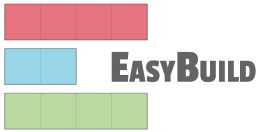
- Supports fully **autonomously** installing (scientific) software, including dependencies, generating environment module files, ...
- **No admin privileges are required** (only write permission to installation prefix)
- **Highly configurable**, easy to extend, support for hooks, easy customisation
-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

Key features of EasyBuild (2/2)



- **Integration** with various other tools: Lmod, Slurm, Singularity/Apptainer, FPM, …
- **Actively developed and supported by worldwide community**
- **Frequent stable releases** since 2012 (roughly 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, …

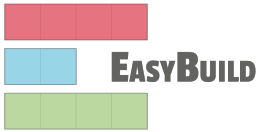
Community effort

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

What EasyBuild is *not*

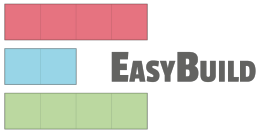
- EasyBuild is **not YABT (Yet Another Build Tool)**
  - It does not try to replace `CMake`, `make`, `pip`, etc.
  - It wraps around those tools and automates installation procedures
- EasyBuild does **not replace traditional Linux package managers** (`yum`, `dnf`, `apt`, ...)
  - You should still install some software via OS package manager
  - Anything that is run with admin privileges and should be updated in-place (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



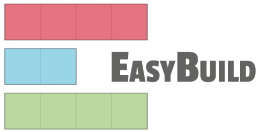
- It is important to briefly explain some terminology often used in EasyBuild
- Some concepts are specific to EasyBuild: easyblocks, easyconfigs, easystacks, …
- Overloaded terms are clarified: modules, extensions, toolchains, …

EasyBuild terminology speed run: framework



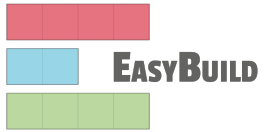
- The EasyBuild framework is the **core of EasyBuild**
- **Collection of Python modules**, organised in packages
- Implements **common functionality** for building and installing software
- Defines abstract installation procedure, in steps (configure, build, test, install, ...)
- Support for applying 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 speed run: easyblock



- A **Python module** that implements steps of installation procedure (as defined by framework)
  - In some sense: a “plugin” to the EasyBuild framework, or a script that leverages EasyBuild framework
- **Generic easyblocks** for “standard” procedures: `cmake`/`make`/`make install`, Python packages, etc.
- **Software-specific easyblocks** for complex software (OpenFOAM, PyTorch, 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`)

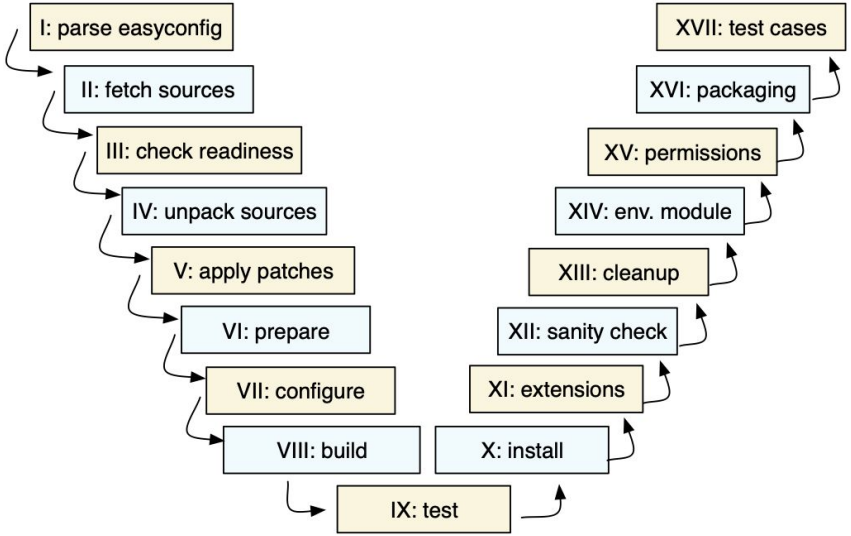
EasyBuild terminology speed run: easyconfig file



- "Build recipe"
- Text file (with `.eb` extension) that specifies what EasyBuild should install (in Python syntax)
- **Collection of values for easyconfig parameters** (key-value definitions),

  usually very little to no logic (cfr. easyblock)
- Also specifies which easyblock to use (directly, or indirectly via software name)
- 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

Step-wise installation procedure



- The EasyBuild framework specifies **step-wise installation procedure**, but leaves some steps unimplemented
- An easyblock defines the implementation, can override or extend some installation steps where needed
- An easyconfig file provides the details for a particular installation: software version, dependencies, toolchain, ...

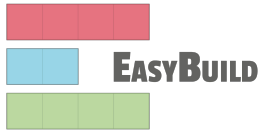
EasyBuild terminology speed run: easystack file

- New concept since EasyBuild v4.3.2 (Dec'20), stable since EasyBuild 5.0 (Mar'25)
- Concise description of software stack to be installed, in YAML syntax
- Basically **specifies a set of easyconfig files**
- Specific EasyBuild configuration options can be used per easyconfig file
- Minimal example, with a single easyconfig file and a custom configuration options

```
easyconfigs:
- example.eb:
    options:
      from-commit: d3adb33f  # use easyconfig from specific commit
```

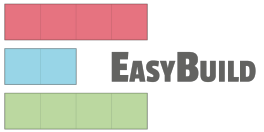
- More info: docs.easybuild.io/easystack-files

EasyBuild terminology speed run: extensions



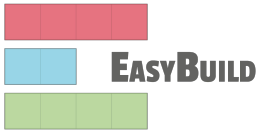
- **Additional software that can be installed *on top* of other software**
- Common examples: Python packages, Perl modules, R libraries, …
- *Extensions* is the general term we use for this type of software packages
- Can be installed in different ways:
  - As a stand-alone software packages (separate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

EasyBuild terminology speed run: dependencies



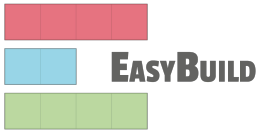
- Software that is **required to build/install or run other software**
- **Build dependencies**: only required when building/installing software (not to use it)
  - Examples: CMake, pip, pkg-config, …
- **Dependencies**: (also) required to use the installed software
  - Examples: Python, Perl, R, OpenBLAS, FFTW, …

EasyBuild terminology speed run: toolchains



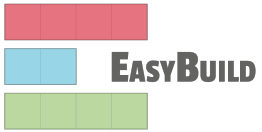
- **Compiler toolchain: set of compilers + libraries for MPI, BLAS/LAPACK, FFT, …**
- Toolchain component: a part of a toolchain (compiler component, etc.)
- **Full toolchain**: C/C++/Fortran compilers + libraries for MPI, BLAS/LAPACK, FFT
- **Subtoolchain** (partial toolchain): compiler-only, only compiler + 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 speed run: 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

EasyBuild terminology: bringing it all together

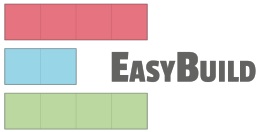


The EasyBuild **framework** leverages **easyblocks** to automatically build and install (scientific) software, potentially including additional **extensions**, using a particular compiler **toolchain**, as specified in **easyconfig files** which each define a set of **easyconfig parameters**.

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.

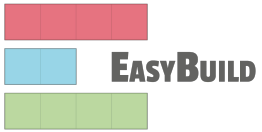
An **easystack** file can be used to specify a collection of software to install with EasyBuild.

Installing EasyBuild: requirements



- **Linux** as operating system (CentOS, RHEL, Ubuntu, Debian, SLES, ...)
  - EasyBuild framework also works on macOS, but support for installing software is very basic
- **Python** 3.6+ (Python 3.9+ recommended)
  - Only Python standard library is required for core functionality of EasyBuild
- An **environment modules tool** (`module` command)
  - Default is Lua-based Lmod implementation, highly recommended!
  - Tcl-based implementation (Environment Modules) is also supported
- Noteworthy optional dependencies:
  - `archspec` Python package (CPU detection)
  - `rich` Python package (colors, progress bars)

Installing EasyBuild: different options



- Option 1: Installing EasyBuild using a standard Python installation tool like `pip`
  - `pip install easybuild`
  - … or a variant thereof (`pip3 install --user`, using `virtualenv`, etc.)
  - May require additional commands, for example to update environment
- Option 2: Installing EasyBuild as a module, with EasyBuild
  - 2-step “bootstrap” procedure, via temporary EasyBuild installation using `pip`
- Option 3: Development setup, using `develop` branch from Git repositories
  - Clone GitHub repositories:

    `easybuilders/easybuild-{framework,easyblocks,easyconfigs}`
  - Update `$PATH` and `$PYTHONPATH` environment variables

Installing EasyBuild: pip install in Python venv

```
eb-demo $ python3 -m venv eb-env
eb-demo $ source eb-env/bin/activate
(eb-env) eb-demo $ pip install --upgrade pip
…
Successfully installed pip-25.1.1
(eb-env) eb-demo $ pip install easybuild archspec rich
Collecting easybuild
…
Installing collected packages: easybuild-framework, easybuild-easyconfigs,
easybuild-easyblocks, easybuild, archspec, rich, …
Successfully installed archspec-0.2.6 easybuild-5.3.0 easybuild-easyblocks-5.3.0
easybuild-easyconfigs-5.3.0 easybuild-framework-5.3.0 rich-15.0.0 …
…
(eb-env) eb-demo $ eb --version
This is EasyBuild 5.3.0 (framework: 5.3.0, easyblocks: 5.3.0) on host example.
```

Verifying the EasyBuild installation

- Check EasyBuild version:

```
eb --version
```

- Show help output (incl. long list of ~300 supported configuration options)

```
eb --help
```

- Show the current (default) EasyBuild configuration, only most important or changed settings:

```
eb --show-config    # or use –show-full-config to see all settings
```

- Show system information:

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

- Show (optional) dependencies:

```
eb --check-eb-deps
```

Updating EasyBuild (with pip or EasyBuild)

- Updating EasyBuild (in-place), if it was installed with pip:

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

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

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

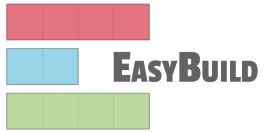
```
eb --install-latest-eb-release
```

  (you may need to install wheel first: `pip install wheel`)

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

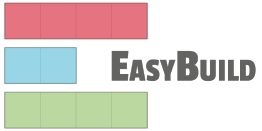
```
module load EasyBuild/5.3.0
```

Configuring EasyBuild



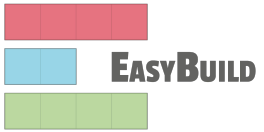
- EasyBuild should work fine out-of-the-box if you are using Lmod as modules tool
- ... but it will (ab)use `$HOME/.local/easybuild` to install software into, etc.
- **It is *strongly* recommended to configure EasyBuild properly!**
- Main questions you should ask yourself:
  - Where should EasyBuild install software (incl. module files)?
  - Where should auto-downloaded sources be stored?
  - Which (local) 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 (~300) is available via `eb --help`

Configuration levels

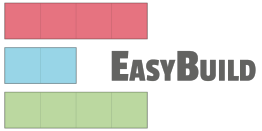


- There are 3 different configuration levels in EasyBuild:
  - **Configuration files**
  - **Environment variables**
  - **Command line options to the `eb` command**

- Each configuration setting can be specified via each “level” (no exceptions!)

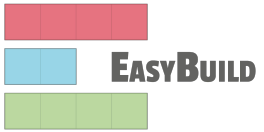
- Hierarchical configuration:
  - Configuration files override default settings
  - Environment variables override configuration files
  - eb command line options override environment variables

EasyBuild configuration files



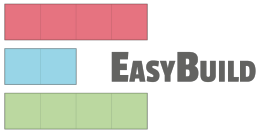
- EasyBuild configuration files are in standard INI format (`key=value`)
- EasyBuild considers multiple locations for configuration files:
  - User-level: `$HOME/.config/easybuild/config.cfg` (or via `$XDG_CONFIG_HOME`)
  - System-level: `/etc/xdg/easybuild.d/*.cfg` (or via `$XDG_CONFIG_DIRS`)
  - See 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



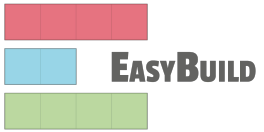
- Very convenient way to configure EasyBuild
- **There is an `$EASYBUILD_*` environment variable for each configuration setting**
  - Use all capital letters
  - Replace every dash (–) character with an underscore (_)
  - Prefix with `EASYBUILD_`
  - Example: `module-syntax` → `$EASYBUILD_MODULE_SYNTAX`
- 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

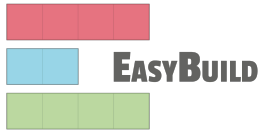


- It can be difficult to remember how EasyBuild was configured
- Output produced by **`eb --show-config`** is useful to remind you
- Shows configuration settings that are different from default
- Always shows a couple of key configuration settings
- Also shows on which level each configuration setting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

Inspecting the current configuration: example

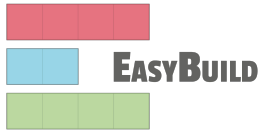
```
$ cat $HOME/config.cfg
[config]
prefix=$HOME/easybuild
buildpath=/tmp/$USER

$ export EASYBUILD_CONFIGFILES=$HOME/config.cfg

$ eb --installpath=/tmp/$USER --show-config
# Current EasyBuild configuration
# (C: command line argument, D: default value,
#  E: environment variable, F: configuration file)
buildpath       (F) = /tmp/ec2-user
configfiles     (E) = /home/ec2-user/config.cfg
containerpath   (F) = /home/ec2-user/easybuild/containers
installpath     (C) = /tmp/ec2-user
packagepath     (F) = /home/ec2-user/easybuild/packages
prefix          (F) = /home/ec2-user/easybuild
repositorypath  (F) = /home/ec2-user/easybuild/ebfiles_repo
robot-paths     (D) = /home/ec2-user/eb-env/easybuild/easyconfigs
rpath           (D) = True
sourcepath      (F) = /home/ec2-user/easybuild/sources
```

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:

```
$ eb --search BCFtools
```

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 BCFtools-1.18-GCC-12.3.0.eb
easyblock = 'ConfigureMake'

name = 'BCFtools'
version = '1.18'

homepage = 'https://www.htslib.org/'
description = """Samtools is a suite of programs for interacting with high-throughput
sequencing data.
 BCFtools - Reading/writing BCF2/VCF/gVCF files and calling/filtering/summarising SNP and
short indel sequence
 variants"""

toolchain = {'name': 'GCC', 'version': '12.3.0'}
toolchainopts = {'pic': True}
…
```

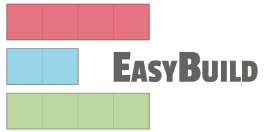
Checking dependencies via `eb --dry-run`

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

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

```
$ eb BCFtools-1.18-GCC-12.3.0.eb -Dr
 ...
 * [x] $CFGS/x/XZ/XZ-5.4.2-GCCcore-12.3.0.eb (module: XZ/5.4.2-GCCcore-12.3.0)
 * [x] $CFGS/g/GSL/GSL-2.7-GCC-12.3.0.eb (module: GSL/2.7-GCC-12.3.0)
 * [ ] $CFGS/h/HTSlib/HTSlib-1.18-GCC-12.3.0.eb (module: HTSlib/1.18-GCC-12.3.0)
 * [ ] $CFGS/b/BCFtools/BCFtools-1.18-GCC-12.3.0.eb (module:
BCFtools/1.18-GCC-12.3.0)
```

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

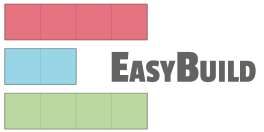


To check which dependencies are still *missing*, use `eb --missing` (or `eb -M`):

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

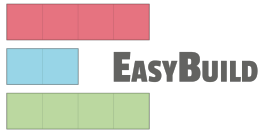
```
$ eb BCFtools-1.18-GCC-12.3.0.eb -M

2 out of 23 required modules missing:

* HTSlib-1.18-GCC-12.3.0.eb (HTSlib-1.18-GCC-12.3.0.eb)

* BCFtools/1.18-GCC-12.3.0 (BCFtools-1.18-GCC-12.3.0.eb)
```

Inspecting software install procedures



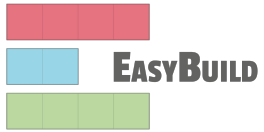
- EasyBuild can quickly unveil how it *would* install an easyconfig file, without actually installing it
- Via `eb --extended-dry-run` (or `eb -x`)
- Produces detailed output in a matter of seconds
- Software is not actually installed, all shell commands and file operations are skipped!
- Some guesses and assumptions are made, so output 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



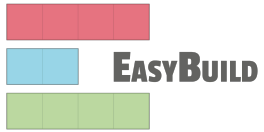

```
$ eb Boost-1.82.0-GCC-12.3.0.eb -x
...

preparing... [DRY RUN]

[prepare_step method]
Defining build environment, based on toolchain (options) and specified dependencies...

Loading toolchain module...

module load GCCcore/13.2.0 [SIMULATED]
module load binutils/2.40-GCCcore-13.2.0 [SIMULATED]
module load GCC/13.2.0 [SIMULATED]

Loading modules for dependencies...
…
```

Inspecting software install procedures: example



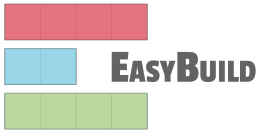

```
$ eb Boost-1.82.0-GCC-12.3.0.eb -x
...
Defining build environment...
  ...
  export CXX='g++'
  export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno -fPIC'
  ...

configuring... [DRY RUN]

[configure_step method]
  running shell command "./bootstrap.sh --with-toolset=gcc
--prefix=/home/user/software/Boost/1.82.0-GCC-12.3.0 --without-libraries=python,mpi"
  (in /tmp/cvanleeuwe/build/Boost/1.82.0/GCC-12.3.0)
  ...
```

Inspecting software install procedures: example



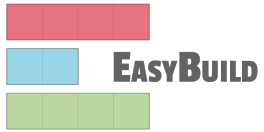

```
$ eb Boost-1.82.0-GCC-12.3.0.eb -x
...

[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)

...
```

Installing software with EasyBuild



- To install software with EasyBuild, just run the `eb` command with an easyconfig file:
  - `eb BCFtools-1.18-GCC-12.3.0.eb`
- If any dependencies are still missing, you will need to also use `--robot`:
  - `eb SAMtools-1.18-GCC-12.3.0.eb --robot`
- Details while the installation is running via trace output (enabled by default since EasyBuild v5.0)
  - `eb BCFtools-1.18-GCC-12.3.0.eb --robot --trace`
- To reinstall software, use `eb --rebuild` (or `eb --force`)

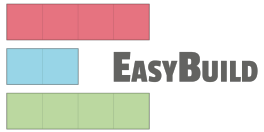
Using software installed with EasyBuild



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/1.18-GCC-12.3.0
```

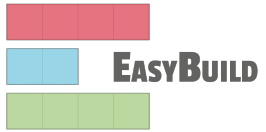
Stacking software installations



- 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 $HOME/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

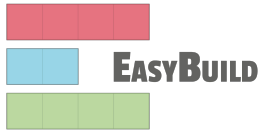
Troubleshooting: error messages

- 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: shell command 'make ...' failed
with exit code 2 in build step for BCFtools-troubleshooting.eb (took 3 secs)
```

- It may take a bit of effort to figure out the *actual* underlying 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:

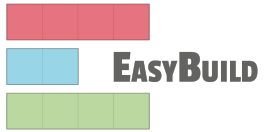
```
== 2025-05-19 08:43:21,688 run.py:500 INFO Running shell command 'make  -j 16
CFLAGS="-O2 -faster"' in /tmp/ec2-user/BCFtools/1.18/GCC-12.3.0/bcftools-1.18
```

- Different steps of installation procedure are clearly marked:

```
== 2025-05-19 08:43:21,817 example INFO Starting sanity check step
```

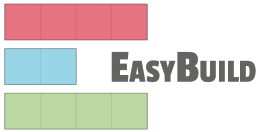
- To find actual problem for a failing shell command, look for patterns like:
  - ERROR
  - Error 1
  - error:
  - failure
  - not found
  - No such file or directory
  - Segmentation fault

Troubleshooting: inspecting the build directory



- EasyBuild leaves the build directory in place when the installation failed
- 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 with EasyBuild v5.0



- **EasyBuild v5.0 makes troubleshooting failing installations significantly easier**
- When a shell command run by EasyBuild fails:
  - The problem will be reported in a more user-friendly way
  - You can quickly inspect (only) the output of that command
  - A script is generated to start an **interactive shell session** to debug “in context”: in the correct working directory + prepared build environment
- This was made possible by switching to the new `run_shell_cmd` function

Improved error reporting in EasyBuild v5.0

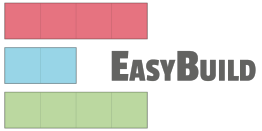
EasyBuild 5.0 produces clearer error messages when a shell command failed:

```
ERROR: Shell command failed!
       full command                -> make  -j 8 LDFLAGS='-lfast'
       exit code                   -> 2
       called from                 -> 'build_step' function in /.../easyblocks/generic/configuremake.py (line 357)
       working directory           -> /tmp/ec2-user/kenneth/easybuild/build/BCFtools/1.18/GCC-12.3.0/bcftools-1.18
       output (stdout + stderr)  -> /tmp/eb-i61vle8x/run-shell-cmd-output/make-1ynysa6f/out.txt
       interactive shell script  -> /tmp/eb-i61vle8x/run-shell-cmd-output/make-1ynysa6f/cmd.sh
```

- Colors to draw attention to the most important parts of the error message
- File with (only) command output + path to build directory are easy to find
- **Auto-generated `cmd.sh` script starts interactive subshell in correct build environment!**

This is powered by the new `run_shell_cmd` function that EasyBuild uses to run shell commands, which took a lot of effort, partially because all ~240 easyblocks has to be updated to use `run_shell_cmd`.

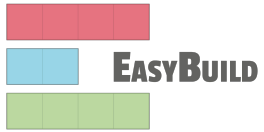
Troubleshooting: integration with LLMs (PoC)



- Proof-of-concept integration with LLMs, implemented as an EasyBuild hook (May 2025)
- https://github.com/boegel/easybuild-llm
- Feed output of failing shell command to a (local) LLM, served by Ollama,

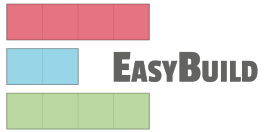
  ask it to provide a concise explanation of what went wrong.
- Next steps could be:
  - Using a more capable (local) LLM
  - Ask to explain to a human how to fix the problem
  - Agentic loop to change easyconfig file until installation works...

Troubleshooting: integration with LLMs (PoC)



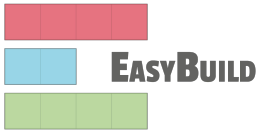

```
$ export EB_LLM_MODEL='gemma3:1b'
$ eb --hooks eb-llm-hooks.py example.eb
...
Shell command 'make  -j 8' failed! (exit code 127)
Large Language Model 'gemma3:1b' explains it as follows:

> The error message "command not found" indicates that the shell cannot locate the
> `make` executable. This is because the shell is trying to execute a command,
> but it doesn't know where to find it.
>
> The issue is that the `make` executable is not in the system's `PATH`
> environment variable. This means the shell isn't aware of the location where
> `make` is located.

*** NOTE: the text above was produced by an AI model, it may not be fully accurate! ***
(time spent querying LLM: 3.885 sec | tokens used: input=149, output=89)

ERROR: Shell command failed!
    full command              ->  make  -j 8
    exit code                 ->  127
    called from               ->  'build_step' function in …/easybuild/easyblocks/generic/configuremake.py
(line 382)
    working directory         ->  /tmp/build/example/1.2.3/system-system/example-1.2.3
    output (stdout + stderr)  ->  /tmp/eb-d3adb33f/run-shell-cmd-output/make-c0ff33/out.txt
    interactive shell script  ->  /tmp/eb-d3adb33f/run-shell-cmd-output/make-c0ff33/cmd.sh
```

Adding support for additional software



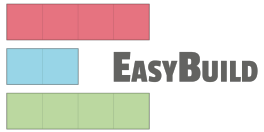
- Every installation performed by EasyBuild requires an **easyconfig file**
- Easyconfig files can be:
  - Included with EasyBuild itself (or obtained elsewhere)
  - Derived from an existing easyconfig (manually or automatic)
  - Created from scratch
- Most easyconfigs leverage a generic easyblock
-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

Easyblocks vs easyconfigs



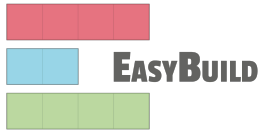
- When can you get away with using an easyconfig leveraging a generic easyblock?
- When is a software-specific easyblock really required?
- Easyblocks are *"implement once and forget"*
- Easyconfig files leveraging a generic easyblock can become (too) complicated
- Reasons to consider implementing a 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



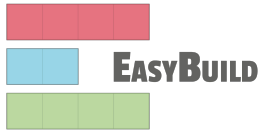
- Collection of easyconfig parameter definitions (Python syntax), collectively specify what to install
- Some easyconfig parameters are **mandatory**, and must always be defined:
  `name`, `version`, `homepage`, `description`, `toolchain`
- Commonly used easyconfig parameters (but strictly speaking not required):
  - `easyblock` (by default derived from software name)
  - `versionsuffix`
  - `source_urls`, `sources`, `patches`, `checksums`
  - `dependencies`, `builddependencies`
  - `preconfigopts`, `configopts`, `prebuildopts`, `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,12.3.0`
- 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:

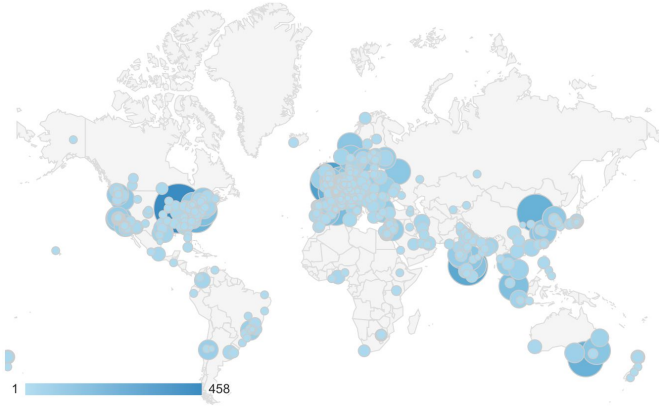
```
$ eb --copy-ec BCFtools-1.18-GCC-12.3.0.eb BCFtools.eb
...
BCFtools-1.18-GCC-12.3.0.eb copied to BCFtools.eb
```

Exercise on creating easyconfig file from scratch

- Step-wise example + exercise of creating an easyconfig file from scratch
- For fictitious software packages: `eb-tutorial` + `py-eb-tutorial`
- Sources available at https://github.com/easybuilders/easybuild-tutorial/tree/main/docs/files
- **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"
```

The EasyBuild community

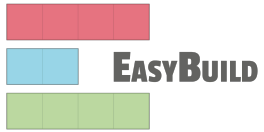


EASYBUILD

- Documentation is read all over the world
- HPC sites, consortia, and companies
- Slack: >1000 members,

  ~180 active members per week
- Bi-weekly online conf calls + yearly user meeting

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:
  docs.easybuild.io/integration-with-github

Opening a pull request in 1, 2, 3

```
$ mv sklearn.eb scikit-learn-1.4.2-gfbf-2023a.eb
$ mv scikit-learn*.eb easybuild/easyconfigs/s/scikit-learn
$ git checkout develop && git pull upstream develop
$ git checkout -b scikit_learn_142_gfbf_2023a
$ git add easybuild/easyconfigs/s/scikit-learn
$ git commit -m "{data}[gfbf/2023a] scikit-learn v1.4.2"
$ git push origin scikit_learn_142_gfbf_2023a
```

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

one single `eb` command

no git commands

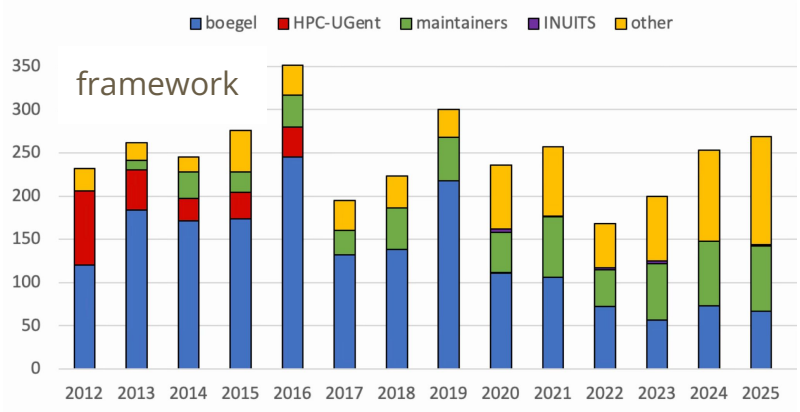
no GitHub interaction

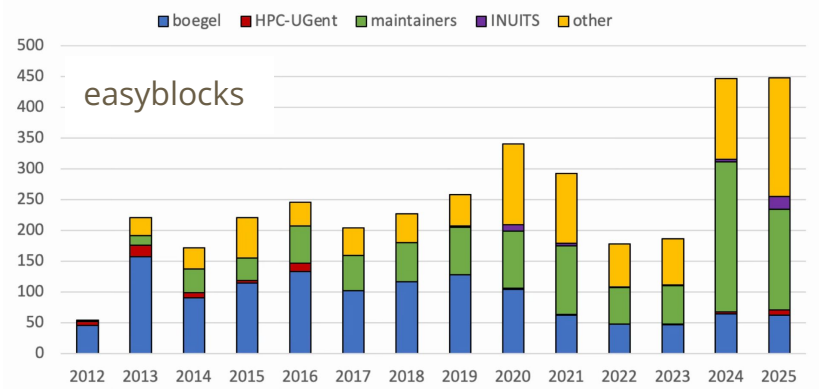
metadata is automatically derived from easyconfig

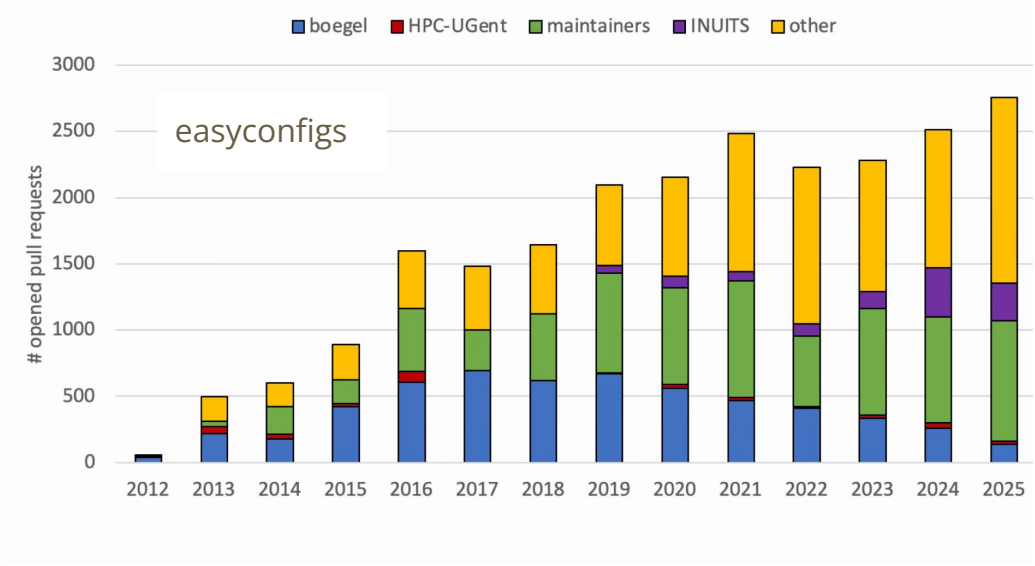
saves a lot of time!

`eb --new-pr sklearn.eb`

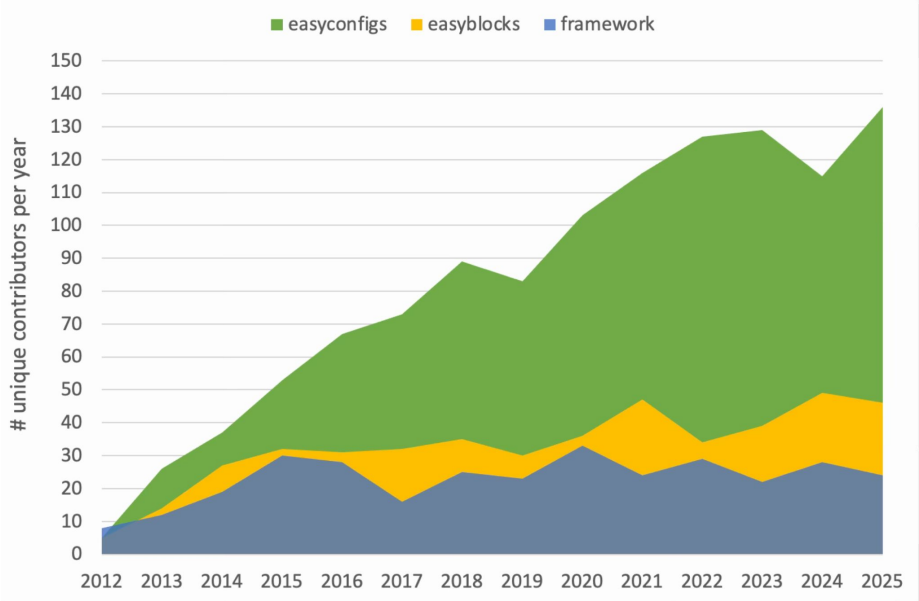
Contributions to EasyBuild (per repo/year)

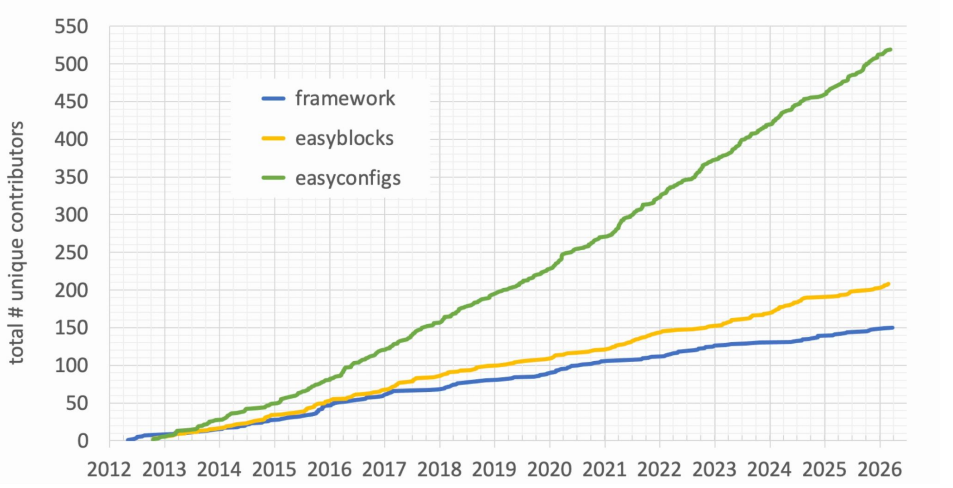




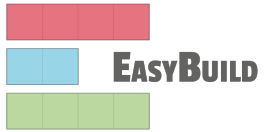


EasyBuild Contributors (per year + since dawn of time)



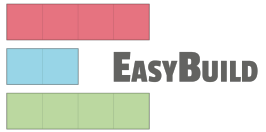


Customizing EasyBuild via Hooks



- Hooks allow you to customize EasyBuild easily and consistently
- Set of Python functions that are automatically picked up by EasyBuild
- Can be used to ”hook” custom code into specific installation steps
- Make EasyBuild use your hooks via `hooks` configuration option
- Examples:
  - Inject or tweak configuration options
  - Change toolchain definitions
  - Custom checks to ensure that site policies are taken into account
- Extensively documented: docs.easybuild.io/hooks

Hooks: examples



- EUM'22 talk by Alex: Building a heterogeneous MPI stack with EasyBuild

  https://easybuild.io/eum22/#eb-mpi

- `contrib/hooks` subdirectory in easybuild-framework GitHub repository:

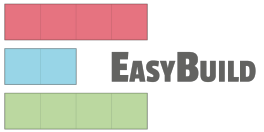
  https://github.com/easybuilders/easybuild-framework/tree/develop/contrib/hooks

Hooks: examples

Ensure that software is installed with a specific license group:

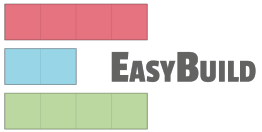
```
def parse_hook(self, *args, **kwargs):
    if self.name == 'example':
        # use correct license group for software ‘example’
        self['group'] = 'licensed_users_example'
```

Implementing Easyblocks



- An easyblock may be required for more complex software installations
- This requires some Python skills, and familiarity with EasyBuild framework
- A software-specific easyblock can derived from a generic easyblock
- Focus is usually on configure/build/installs steps of installation procedure
- See also https://docs.easybuild.io/implementing-easyblocks

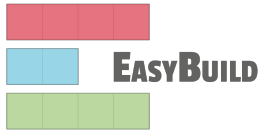
Submitting Installations as Slurm Jobs



- EasyBuild can *distribute* the installation of a software stack as jobs on a cluster
- Slurm is the default job backend in EasyBuild v5.x
- Use "`eb ... --job --robot`" to submit software installations

  to be performed with EasyBuild as Slurm jobs
- See also https://docs.easybuild.io/submitting-jobs

EasyBuild v5.0

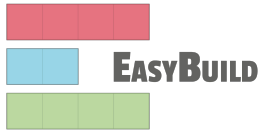


- **Released on 18 March 2025**
- Concludes a development effort that was started in March 2023 (103 weeks)
- Development done in separate `5.0.x` branches, kept in sync with `develop`
- 1,364 merged pull requests

  (framework: 245, easyblocks: 345, easyconfigs: 804)
- **There will be no more EasyBuild 4.x releases,**
  so you must migrate to EasyBuild v5.x!

https://docs.easybuild.io/easybuild-v5

EasyBuild v5.0: Breaking changes

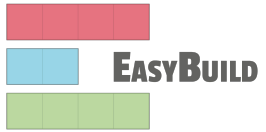


- **Python 3.6+ is required** to run EasyBuild v5.0.0
  - Python 2.7 no longer supported to *run* EasyBuild with (EOL since 2020)

- Updated version requirement for modules tool being used:
  - For Lmod version >= 8.0 is required
  - For Environment Modules version >= 4.3.0 is required

https://docs.easybuild.io/easybuild-v5

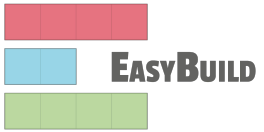
EasyBuild v5.0: Changed defaults



- **RPATH linking** is enabled by default
- Trace output is enabled by default
- `extensions` statement is included by default in generated modules
- `depends_on` is used by default for dependencies in generated modules
- Slurm is used as default job backend
- Default maximum build parallelism is set to 16
- `use_pip` + `sanity_pip_check` enabled by default for `PythonPackage` easyblock
- `CMakeMake` easyblock sets `LIBDIR` configuration option to `lib` by default

https://docs.easybuild.io/easybuild-v5

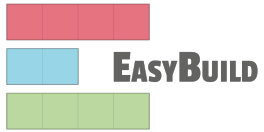
EasyBuild v5.0: Changed behaviour *(selected)*



- `--robot` (`-r`) is no longer enabled by default when using `--dry-run` (`-D`) => Use `eb -Dr`
- Verifying of checksums was moved from from `source` to `fetch` step, to include it with `--fetch`
- `lib` to `lib64` symlink (and vice versa) created before running `postinstallcmds`
- Parsing order for files in `$XDG_CONFIG_DIRS` is reversed + default value is fixed (`/etc/xdg`)
- Unresolved templates in easyconfig parameters are not allowed by default
- Don't automatically prepend a dash (`-`) to first compiler option (relevant for `optarch`)
- Run sanity checks commands from an empty tmdpir rather than the software install directory
- Only allow use of `rpath` toolchain option when `system` toolchain is used

https://docs.easybuild.io/easybuild-v5

EasyBuild v5.0: Enhancements (1/2)

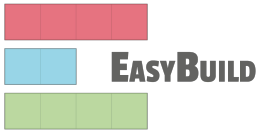


- **New function to run shell commands: `run_shell_cmd`**
- **Interactive debugging of failing shell commands via `env.sh` and `cmd.sh` scripts**
- New collection of easyconfig templates
- Support for installing extensions in parallel stable (no longer experimental)
- Easystack support stable (no longer experimental)
- **Reproducible tarballs for sources created via `git_config` (across Linux & macOS!)**
- New home for the archive of easyconfigs: easybuilders/easybuild-easyconfigs-archive
- Granular exit codes (exit 22 when sanity check fails, exit 31 for missing dependency, ...)
- Copy build directory and/or log file(s) if installation failed to path specified

  via `--failed-install-build-dirs-path` or `--failed-install-logs-path`
- **Specify changes that should be made by generated module files via `module_load_environment`**

https://docs.easybuild.io/easybuild-v5

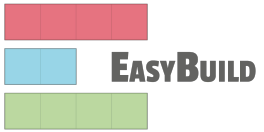
EasyBuild v5.0: Enhancements (2/2)



- Add support for alternate easyconfig parameters/templates/constants
- `keep-debug-symbols` configuration option to set default value of `debug` toolchain option
- Provide control over how generated modules update search path for header files (`$CPATH` or not)
- Provide control over how EasyBuild specifies path to header files during installation
- Provide control over how EasyBuild specifies path to libraries during installation
- Support not using `$PYTHONPATH` to specify the location of installed Python packages
- Revamp of easyconfig parameter `modextrapaths`
- Detect Fortran `.mod` files in installations using `GCCcore` toolchain
- **Let `ConfigureMake` generic easyblock error out on unrecognized configure options**
- Require `download_instructions` for non-public sources

https://docs.easybuild.io/easybuild-v5

EasyBuild v5.0: Removed functionality

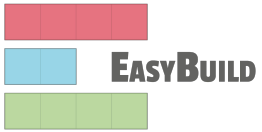


Features that were deprecated in EasyBuild 4.x have been removed:

- EasyBuild bootstrap script
- Experimental support for .yeb easyconfig
- Configuration settings: `accept-eula`, `wait-on-lock` (replaced by equivalent settings)
- Removed functions: `is_generic_easyblock`, `copytree`, `rmtree2`
- Removed methods: `EasyBlock.fetch_extension_sources`, `Toolchain.add_dependencies`
- `mod_exists_regex_template` option in `ModulesTool.exist` method
- Removed options for various methods and functions, like `use_git_am` option for `apply_patch`
- `dummy` toolchain (replaced with `system` toolchain)
- Support for 32-bit targets

https://docs.easybuild.io/easybuild-v5

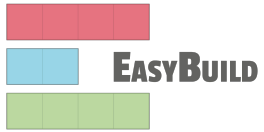
EasyBuild v5.0: Deprecated functionality (1/2)



- **`parallel` easyconfig parameter**
- **`run_cmd` and `run_cmd_qa` functions (replaced with `run_shell_cmd`)**
- `source` step (renamed to `extract`)
- `post_install_step` method in EasyBlock class (renamed to `post_processing_step`)
- Various methods in EasyBlock class: `make_module_req_guess`, `run`, `prerun`, `postrun`, `run_async`
- `easybuild.tools.py2vs3` module (no longer useful since Python 2 is no longer supported)
- Older checksum types

https://docs.easybuild.io/easybuild-v5

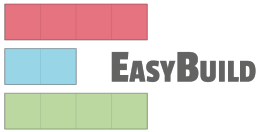
EasyBuild v5.0: Deprecated functionality (2/2)



- `EnvironmentModulesC` or `EnvironmentModulesTcl` modules tools
- GC3Pie as job backend
- Using `optarch` value without leading dash
- `COMPILER*_FLAGS` attributes in `Compiler` class
- Easyconfig parameters: `modextrapaths_append`, `allow_append_abs_path`, `allow_prepend_abs_path`

https://docs.easybuild.io/easybuild-v5

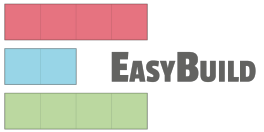
EasyBuild v5.1.0 (May 2025)



- Support for data installations
- **CUDA device code sanity check**
- Improved trace output
- Support for using environment variables in value used in modextravars
- Improvements to `Cargo` easyblock
- Run `pip check` only once for PythonBundle and Python installations
- **Various fixes for custom easyblock for LLVM**
- `foss/2025a` and `intel/2025a` common toolchains
- EasyBuild v5.1.2: Support for amdgcn-capabilities conf. option + easyconfig parameter

https://docs.easybuild.io/release-notes

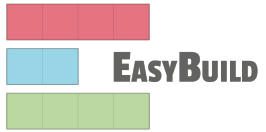
EasyBuild v5.2.0 (Dec 2025)



- Support for **NVHPC toolchain** with nvidia-compilers, NVHPCX, NVBLAS, and NVScaLAPACK
- Support for **LLVM-based toolchains** `lfoss` (OpenMPI) and `lmpflf` (MPICH)
- `--keep-going` option to keep going after an installation failed
- Print total runtime of all builds
- Pass dependencies to `toolchain.prepare` when setting up build environment for extensions
- Also pick up on toolchain components in `Toolchain._add_dependency_variables`
- **Support for using Lmod 9.0**
- **Deprecate support for running EasyBuild with Python < 3.9**
- Make command environment of shell commands more discoverable in the log

https://docs.easybuild.io/release-notes

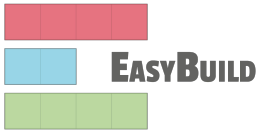
EasyBuild v5.3.0 (Apr 2026)



- Make EasyBuild plugin-able through entrypoints (https://docs.easybuild.io/entrypoints)
- `--fetch-all` command line option
- Support for **ROCm-based toolchains** (rocm-compilers, rompi, rfbf, rfoss);
- Add (experimental) **integration with `bwrap`** (see `--bwrap` and `--bwrap-installpath`)
- Add rich colors to `print_msg`
- Add generic `ROCmComponent` easyblock to build & install ROCm components
- Include `/usr/local` Python package directory for system-level Python package installations
- Enhance `PythonPackage` and `PythonBundle` easyblocks to verify Python package names and versions with `pip list`
- Update of (common) toolchains: `foss/2026.1`, `intel/2026.1`, `lfoss/2026.1`

https://docs.easybuild.io/release-notes

EasyBuild v6.0



- **ETA March 2027** (~2 years after last major EasyBuild release)
- Expected changes (subject to change):
  - **Python 3.9+ required** (+ recent version of Lmod/Environment Modules)
  - **Improved consistency in naming** of easyconfig parameters,

    EasyBuild configuration options, etc.
    - Already partially supported in EasyBuild 5.0.0,

      for example: `configure_opts` instead of `configopts`
  - **Modern `eb` command line interface**, based on `Click` or `Typer`