EASYBUILD

EESSI

EUROPEAN ENVIRONMENT FOR SCIENTIFIC SOFTWARE INSTALLATIONS

Introduction to EasyBuild & EESSI

CARLA 2025 - DevOps School for HPC, 22 Sept 2025

https://carlaconference.org/devops-school-for-hpc

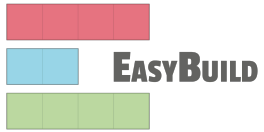
Kenneth Hoste (HPC @ Ghent University, BE) – *kenneth.hoste@ugent.be*

whoami

kenneth.hoste@ugent.be
@boegel
@boegel.bsky.social
mast.hpc.social/@boegel

- Masters & PhD in Computer Science from Ghent University (Belgium)
- Joined HPC-UGent team in October 2010
- Main tasks: user support & training, software installations
- Slowly also became EasyBuild lead developer & release manager
- Now also involved with EESSI and MultiXscale EuroHPC Centre-of-Excellence
- Likes family, beer, loud music, FOSS, helping people, dad jokes, stickers, ...
- Doesn't like C++, CMake, SCons, Bazel, TensorFlow, OpenFOAM, ...

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)
  - Worldwide community has grown around it since then! (>1,000 members on EasyBuild Slack)

https://easybuild.io

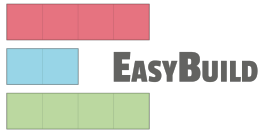
https://docs.easybuild.io

https://blog.easybuild.io

https://github.com/easybuilders

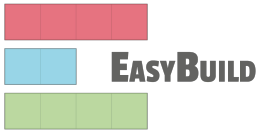
https://easybuild.io/join-slack

EasyBuild in a nutshell



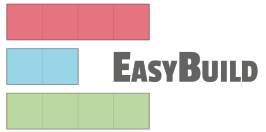
- **Tool** to provide a ***consistent and well performing*** scientific software stack
- Uniform interface for installing scientific software on HPC systems
- Saves time by ***automating*** tedious, boring and repetitive tasks
- Can empower scientific researchers to self-manage their software stack
-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

Key features of EasyBuild (1/2)



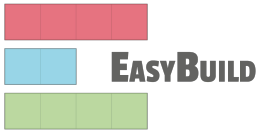
- Supports fully **autonomously** installing (scientific) software, including dependencies, generating environment module files, ...
- **No admin privileges are required** (only write permission to installation prefix)
- **Highly configurable**, easy to extend, support for hooks, easy customisation
- 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, …

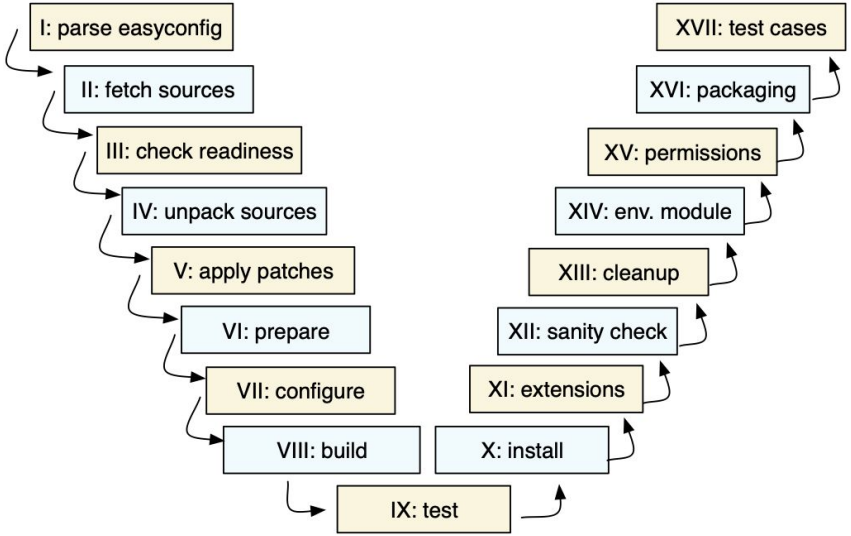
Community effort

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

What EasyBuild is *not*

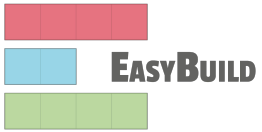
- EasyBuild is **not YABT (Yet Another Build Tool)**
  - It does not try to replace `CMake`, `make`, `pip`, etc.
  - It wraps around those tools and automates installation procedures
- EasyBuild does **not replace traditional Linux package managers** (`yum`, `dnf`, `apt`, ...)
  - You should still install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

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
- Easyconfig file provides the details (software version, dependencies, toolchain, ...)

EasyBuild terminology in a nutshell



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.

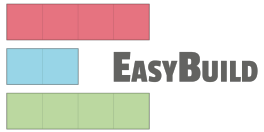
EasyBuild v5.0

- **Released on 18 March 2025**
- Concludes a development effort that was started in March 2023 (103 weeks)
- 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!
- Including backwards-incompatible changes, changes in default configuration, …
- Overview of changes in EasyBuild v5.0: https://docs.easybuild.io/easybuild-v5

Installing EasyBuild: requirements

- **Linux** as operating system (CentOS, RHEL, Ubuntu, Debian, SLES, ...)
  - EasyBuild also works on macOS, but support 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!
    - On RHEL-based Linux: easy to install Lmod via EPEL
  - Tcl-based implementation (Environment Modules) is 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
  - 2-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: pip install in Python venv

```
$ python3 -m venv eb-env
$ source eb-env/bin/activate

(eb-env) $ pip install easybuild archspec rich
Collecting easybuild
…
Installing collected packages: easybuild-framework, easybuild-easyconfigs,
easybuild-easyblocks, easybuild, archspec, rich, …
Successfully installed archspec-0.2.5 easybuild-5.0.0
easybuild-easyblocks-5.0.0 easybuild-easyconfigs-5.0.0
easybuild-framework-5.0.0 rich-14.0.0 …
…
(eb-env) $ eb --version
This is EasyBuild 5.1.1 (framework: 5.1.1, easyblocks: 5.1.1) on host …
```

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
```

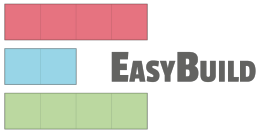
- Show system information:

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

- Check required (and optional) dependencies:

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

Updating EasyBuild (with pip or 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:

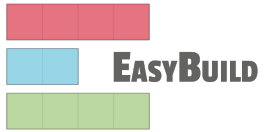
```
eb --install-latest-eb-release
```

  (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:

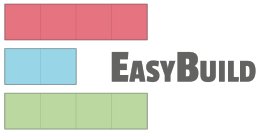
```
module load EasyBuild/5.1.1
```

Configuring EasyBuild



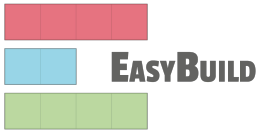
- EasyBuild should work fine out-of-the-box if you are using Lmod as modules tool
- ... but it will (ab)use `$HOME/.local/easybuild` to install software into, etc.
- **It is *strongly* recommended to configure EasyBuild properly!**
- Main questions you should ask yourself:
  - Where should EasyBuild 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 (~280) is available via `eb --help`

Configuration levels

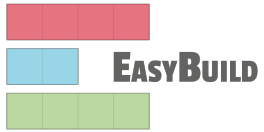


- There are 3 different configuration levels in EasyBuild:
  - **Configuration files** (see `eb --show-default-configfiles`)
  - **Environment variables** (`$EASYBUILD_XYZ`)
  - **Command line options to the `eb` command**

- 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

Inspecting the current configuration

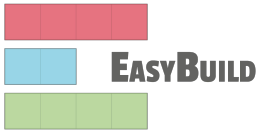


- It can be difficult to remember how EasyBuild was configured
- Output produced by **`eb --show-config`** is useful to remind you
- Shows configuration settings that are different from default
- Always shows a couple of key configuration settings
- Also shows on which level each configuration setting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



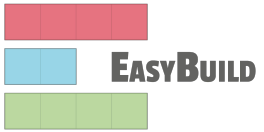
- **Use `eb` command to run EasyBuild**
- Software to install is usually specified via name(s) of easyconfig file(s), or easystack file
- `--robot` (`-r`) option is required to also install missing dependencies (and toolchain)
- Typical workflow:
  - Find or create easyconfig files to install desired 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)

Search for easyconfigs via `eb --search`

To search for easyconfig files to install (case-insensitive), use `eb --search`

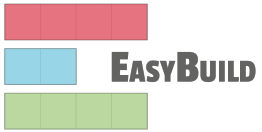
```
$ eb --search bcftools
== found valid index for /home/ec2-user/eb-env/easybuild/easyconfigs, so using it...
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.12-GCC-10.3.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.14-GCC-11.2.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.15.1-GCC-11.3.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.17-GCC-12.2.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.18-GCC-12.3.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.19-GCC-13.2.0.eb
 * /home/example/eb-env/easybuild/easyconfigs/b/BCFtools/BCFtools-1.21-GCC-13.3.0.eb
```

Installing software with EasyBuild



- To install software with EasyBuild, just run the `eb` command:
  - `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`
- More details while the installation is running via trace output (default in EasyBuild v5.x)
  - `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
```

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}
…
```

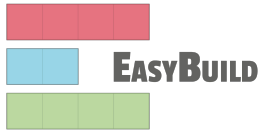
Checking dependencies via `eb --dry-run`

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)
 * [x] $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`

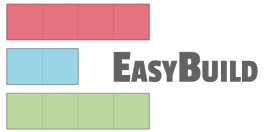


To check which dependencies are still *missing*, use `eb --missing` (or `eb -M`):

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

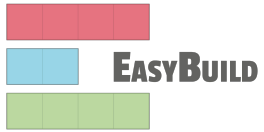
```
$ eb BCFtools-1.18-GCC-12.3.0.eb --missing

1 out of 23 required modules missing:

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

Inspecting software install procedures



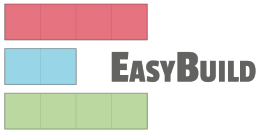
- EasyBuild can quickly unveil how exactly it *would* install an easyconfig file
- Via `eb --extended-dry-run` (or `eb -x`)
- Produces detailed output in a matter of seconds
- **Software is not actually installed, shell commands + file operations are skipped!**
- Some guesses and assumptions are made, 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



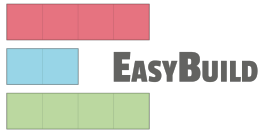

```
$ eb Boost-1.82.0-GCC-12.3.0.eb -x
...
Defining build environment...
  ...
  export CXX='g++'
  export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno -fPIC'
  ...

configuring... [DRY RUN]

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

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: log files

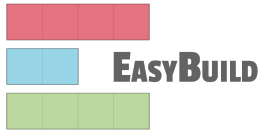


- EasyBuild keeps track of the installation in a detailed log file
- During the installation, it is stored in a temporary directory:

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

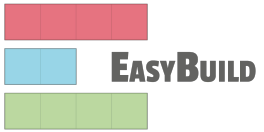
- Includes executed shell commands and output, build environment, etc.
- More detailed log file when debug mode is enabled (`debug` configuration setting)
- There is a log file per EasyBuild session, and one per performed installation
- **When an installation completes successfully,**
  the log file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

Troubleshooting: 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.x



- **EasyBuild v5.x 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
- Made possible by switching to new `run_shell_cmd` function

https://docs.easybuild.io/interactive-debugging-failing-shell-commands

Improved error reporting in EasyBuild v5.x

EasyBuild 5.x 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!**

https://docs.easybuild.io/interactive-debugging-failing-shell-commands

Adding support for additional software

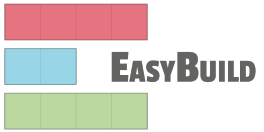
- **Every installation performed by EasyBuild requires an easyconfig file**
- Easyconfig files can be:
  - Included with EasyBuild itself (or obtained elsewhere)
  - Derived from an existing easyconfig (manually or automatic)
  - Created from scratch
- Most easyconfigs leverage a generic easyblock
-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

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`

https://docs.easybuild.io/writing-easyconfig-files

Easyblocks vs easyconfigs



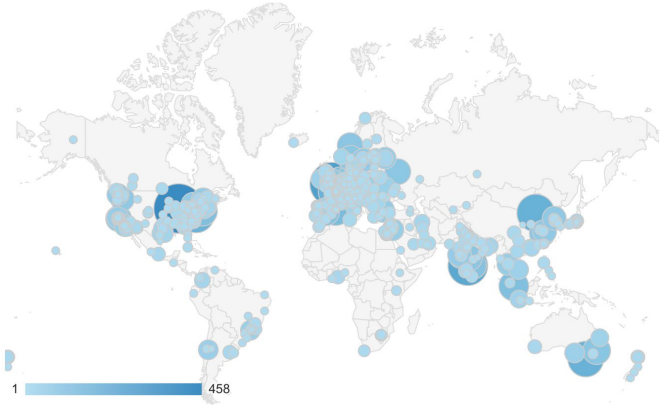
- When can you get away with using an easyconfig leveraging a generic easyblock?
- When is a software-specific easyblock really required?
- Easyblocks are *“implement once and forget”*
- Easyconfig files leveraging a generic easyblock can become too complicated (subjective)
- Reasons to consider implementing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

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

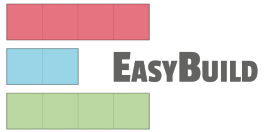


EASYBUILD

- 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

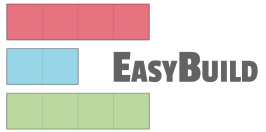
no GitHub interaction

metadata is automatically derived from easyconfig

saves a lot of time!

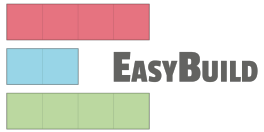
`eb --new-pr sklearn.eb`

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:

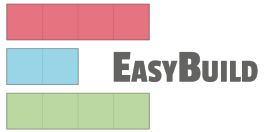
  https://github.com/easybuilders/easybuild-framework/tree/develop/contrib/hooks

Hooks: examples

Ensure that software is installed with a specific license group:

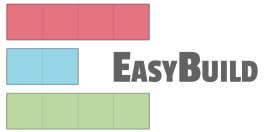
```
def parse_hook(self, *args, **kwargs):
    if self.name == 'example':
        # use correct license group for software ‘example’
        self['group'] = 'licensed_users_example'
```

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

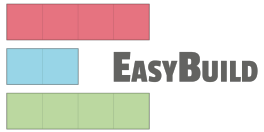
Submitting Installations as Slurm Jobs



- 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

Questions?

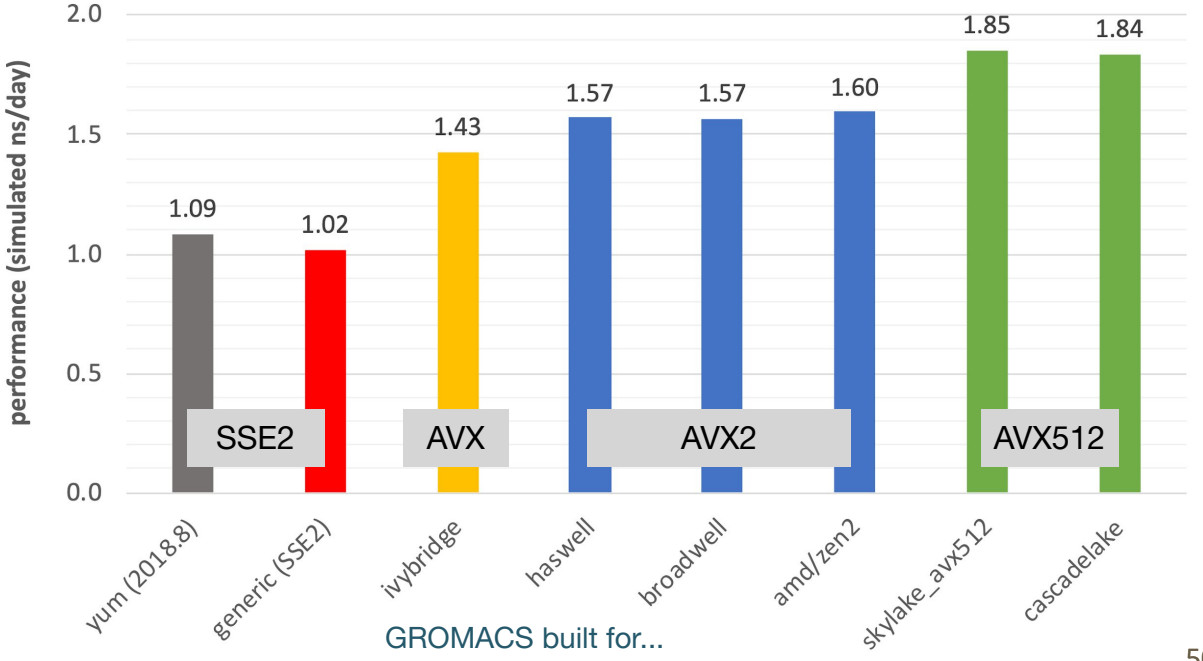


- Website: https://easybuild.io
- Documentation: https://docs.easybuild.io
- Tutorials: https://tutorial.easybuild.io
- EasyBuild User Meeting: https://easybuild.io/eum (slides+recording of *all* talks available!)
- Getting help:
  - Mailing list: https://lists.ugent.be/wws/subscribe/easybuild
  - **Slack: https://easybuild.slack.com - https://easybuild.io/join-slack**
  - Bi-weekly conference calls: https://github.com/easybuilders/easybuild/wiki/Conference-calls

Optimized scientific software installations

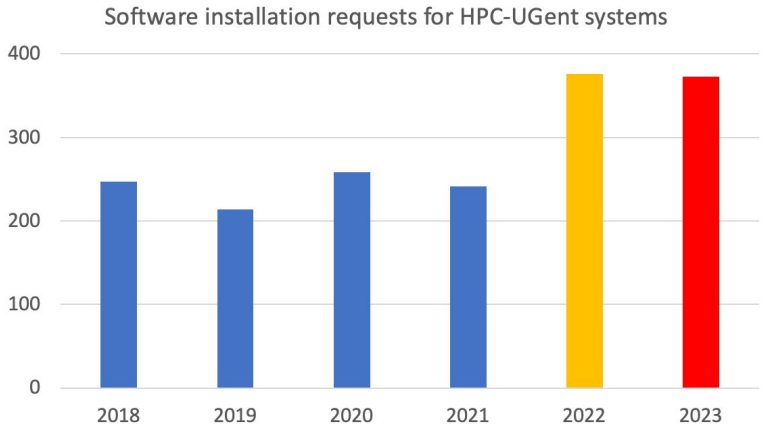
- Software should be optimized for the system it will run on (keep the P in HPC!)
- Impact on performance is often significant for scientific software!

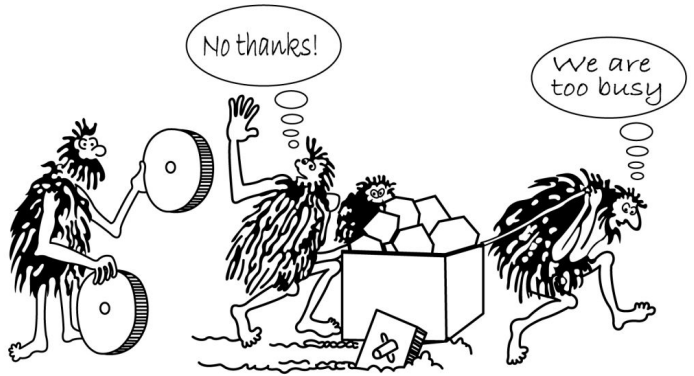
- Example: GROMACS 2020.1 (PRACE benchmark, Test Case B)
- Metric: (simulated) ns/day, higher is better
- Test system: dual-socket Intel Xeon Gold 6420 (Cascade Lake, 2x18 cores)
- **Performance of different GROMACS binaries, on exact same hardware/OS**



The changing landscape of scientific computing

- **Explosion of available scientific software** applications (bioinformatics, AI boom, …)
- Increasing interest in **cloud** for scientific computing (flexibility!)
- **Increasing variety in processor (micro)architectures** beyond Intel & AMD:
  Arm is ~~coming~~ already here (see Fugaku, JUPITER, …), RISC-V is coming (soon?)
- In strong contrast: available (wo)manpower **in HPC support teams is (still) limited**…





What if you no longer have to install

*a **broad range of scientific software***

from scratch on every laptop, HPC cluster,

or cloud instance you use or maintain,

without compromising on performance?

EESSI in a nutshell

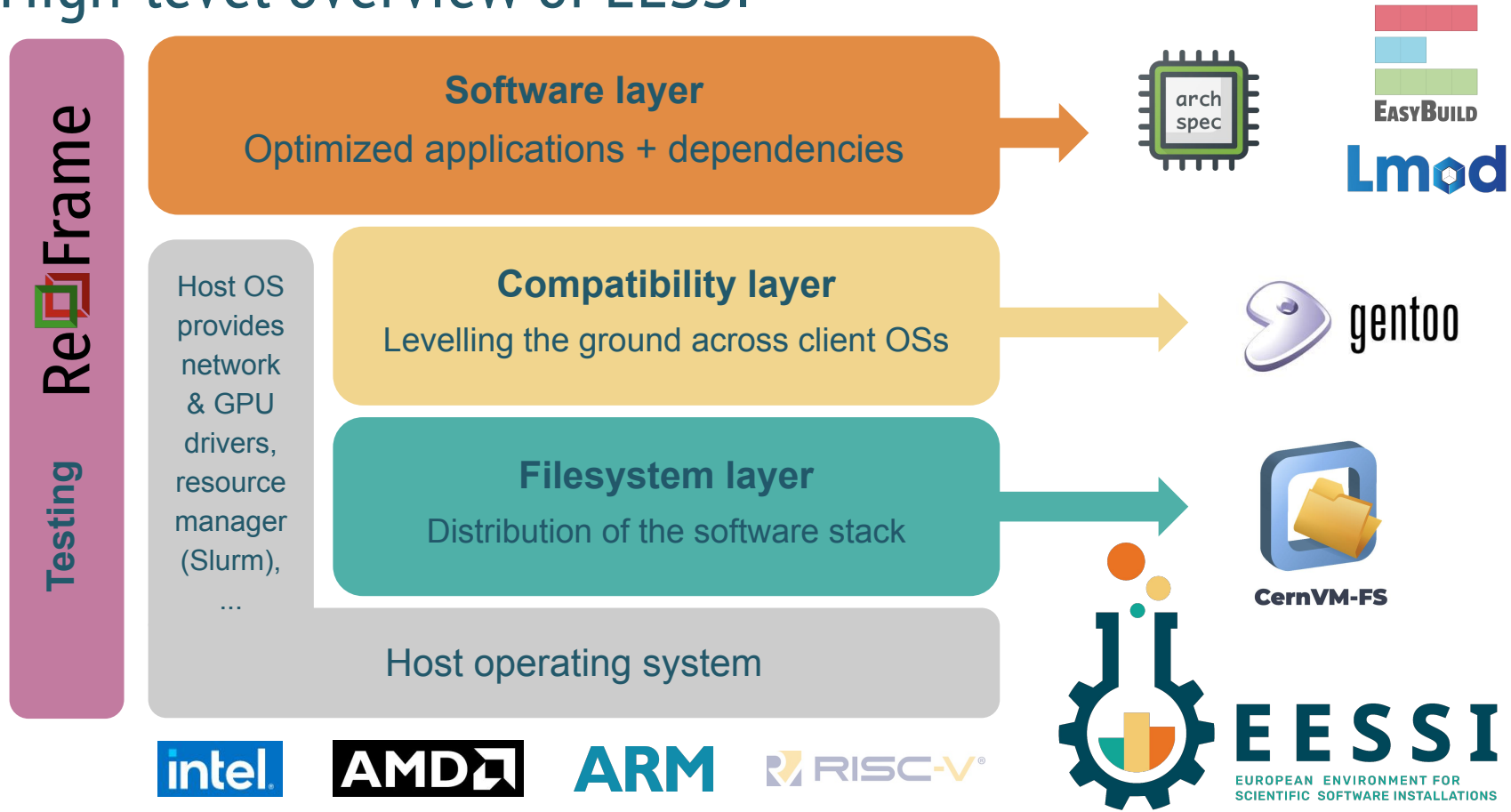
- *European Environment for Scientific Software Installations (EESSI)*
- **Shared repository of (optimized!) scientific software *installations***
- Avoid duplicate work across (HPC) sites by collaborating on a shared software stack
- Uniform way of providing software to users, regardless of the system they use!
- Should work on any Linux OS and system architecture
  - From laptops and personal workstations to HPC clusters and cloud
  - Support for different CPUs, interconnects, GPUs, etc.
- **Focus on performance, automation, testing, collaboration**

https://www.eessi.io/docs/

Major goals of EESSI

- Providing a truly **uniform software stack**
  - Use the (exact) same software environment everywhere
  - **Without sacrificing performance** for “mobility of compute”
    (like is typically done with containers/conda)
- **Avoid duplicate work** (for researchers, HPC support teams, sysadmins, …)
  - Tools that automate software installation process
    (EasyBuild, Spack) are not sufficient anymore
  - Go beyond sharing build recipes => work towards a shared software stack
- Facilitate HPC training, development of (scientific) software, …

High-level overview of EESSI



EESSI ingredients

Compatibility layer

Abstraction from the host operating system

Filesystem Layer

Global distribution of software installations via CernVM-FS

EESSI

EUROPEAN ENVIRONMENT FOR SCIENTIFIC SOFTWARE INSTALLATIONS

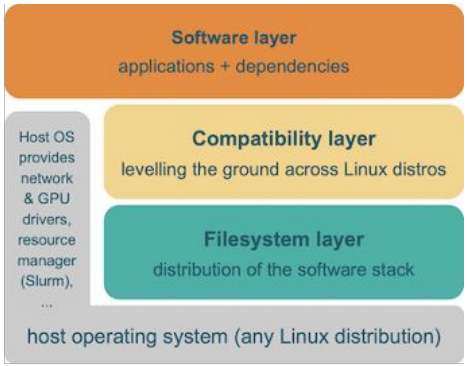
Software Layer

Optimized software installations for specific CPU microarchitectures

Intuitive user interface: module avail, module load, …

Automatic selection of best suited part of software stack for CPU microarchitectures

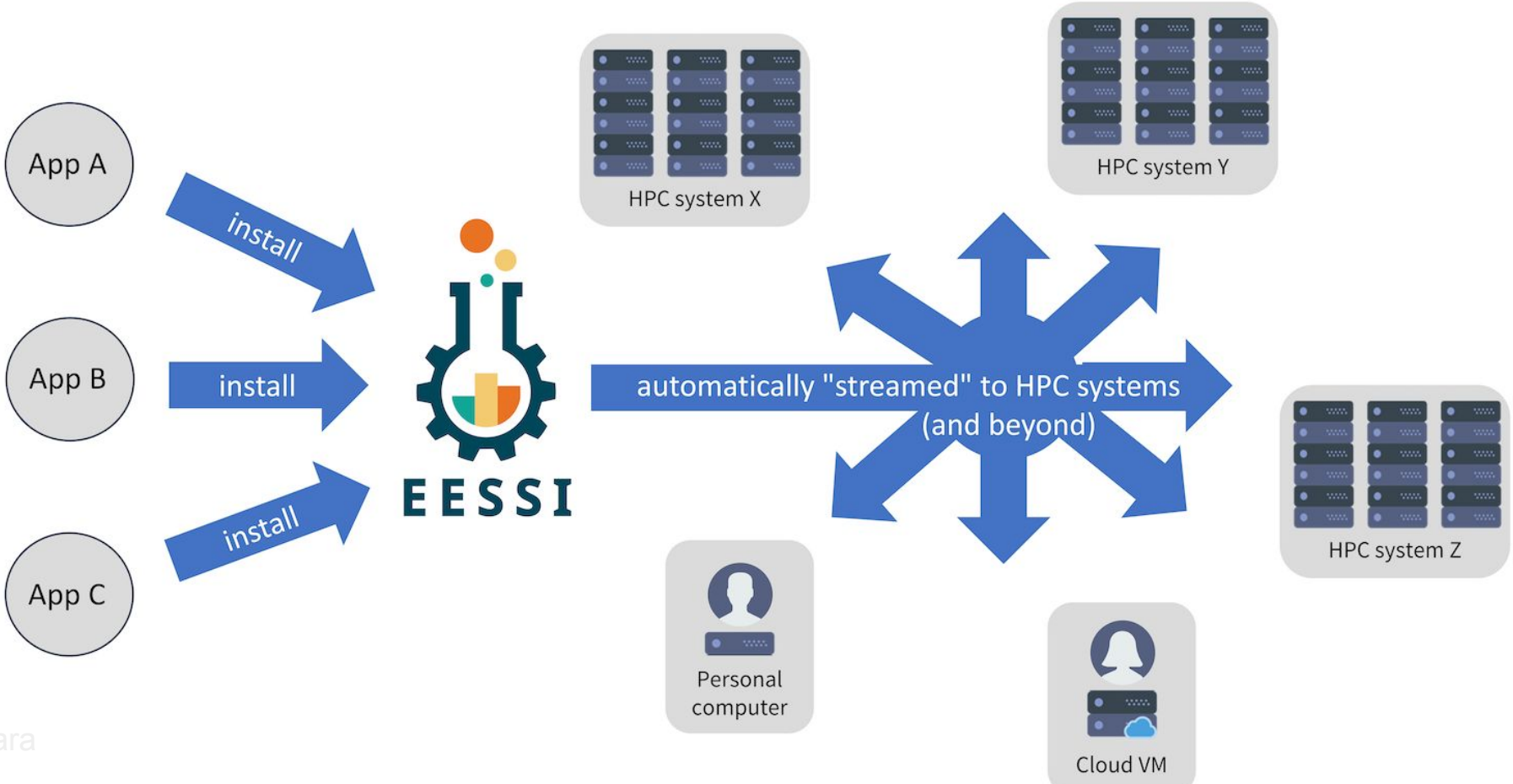
How does EESSI work?



- Software installations included in EESSI are:
  - Automatically **“streamed in” on demand** (via CernVM-FS)
  - Built to be **independent of the host operating system**

    “Containers without the containing”
  - **Optimized** for specific CPU generations + specific GPU types
- Initialization script **auto-detects** CPU + GPU of the system

EESSI as a shared software stack

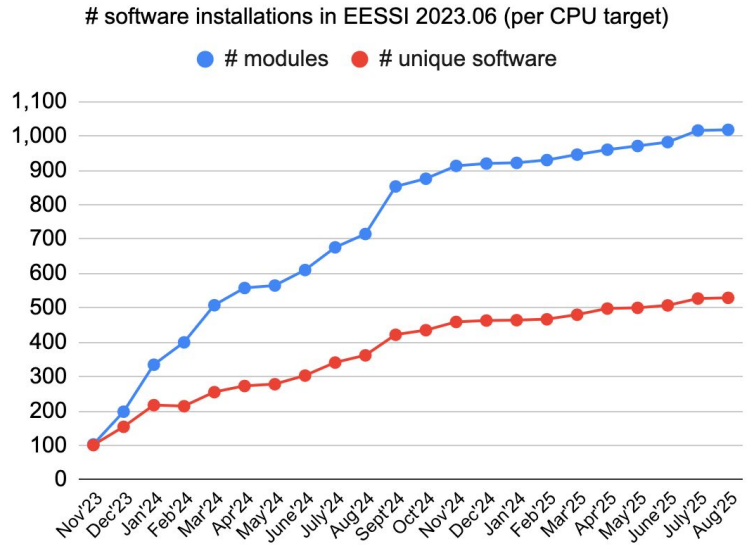


Overview of available software in EESSI

Currently > 1,000 software installations available per supported CPU target

via `software.eessi.io` CernVM-FS repository; increasing every week

- **13** (+1) **supported CPU targets** (x86_64 + Arm), see https://eessi.io/docs/software_layer/cpu_targets
- **Over 500 different software packages**, excl. extensions: Python packages, R libraries
- **Over 15,000 software installations in total**
- Including ESPResSo, GROMACS, LAMMPS, OpenFOAM, PyTorch, R, QuantumESPRESSO, TensorFlow, waLBerla, WRF, ...
- eessi.io/docs/available_software/overview



- Using `foss/2023a` and `foss/2023b` toolchains in EESSI 2023.06
- Using `foss/2024a` and `foss/2025a` toolchains in EESSI 2025.06

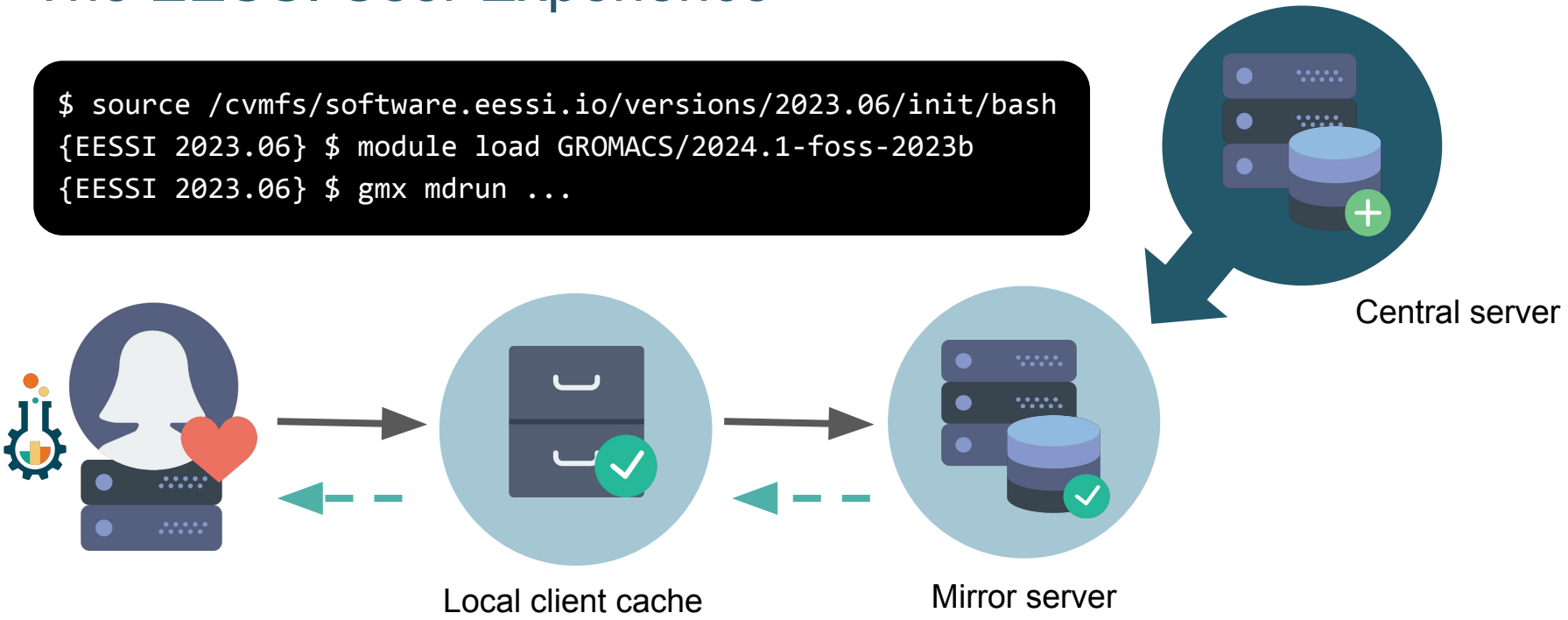
Getting access EESSI via CernVM-FS (demo)

```
# Native installation
# Installation commands for RHEL-based distros
# like CentOS, Rocky Linux, Almalinux, Fedora, …

# install CernVM-FS
sudo yum install -y
https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest.noarch.rpm
sudo yum install -y cvmfs

# create client configuration file for CernVM-FS
# (no proxy, 10GB local CernVM-FS client cache)
sudo bash -c "echo 'CVMFS_CLIENT_PROFILE="single"' > /etc/cvmfs/default.local"
sudo bash -c "echo 'CVMFS_QUOTA_LIMIT=10000' >> /etc/cvmfs/default.local"

# Make sure that EESSI CernVM-FS repository is accessible
sudo cvmfs_config setup
```

Alternative ways of accessing EESSI are available, via a container image, via cvmfsexec, …
eessi.io/docs/getting_access/native_installation - eessi.io/docs/getting_access/eessi_container

The EESSI User Experience

```
$ source /cvmfs/software.eessi.io/versions/2023.06/init/bash
{EESSI 2023.06} $ module load GROMACS/2024.1-foss-2023b
{EESSI 2023.06} $ gmx mdrun ...
```



EESSI provides **on-demand streaming**

of (scientific) software (like music, TV-series, …)

Using EESSI (demo)

eessi.io/docs/using_eessi/eessi_demos

```
/cvmfs/software.eessi.io/versions/2023.06/software
`-- linux
    |-- aarch64
    |   |-- a64fx
    |   |-- generic
    |   |-- neoverse_n1
    |   |-- neoverse_v1
    |   `-- nvidia/grace
    `-- x86_64
        |-- amd
        |   |-- zen2
        |   |-- zen3
        |   `-- zen4
        |-- generic
        `-- intel
            |-- cascadelake
            |-- haswell
            |-- icelake
            |-- haswell
            `-- sapphirerapids
                |-- modules
                `-- software
```

```
$ source /cvmfs/software.eessi.io/versions/2023.06/init/bash
Found EESSI pilot repo @
/cvmfs/software.eessi.io/versions/2023.06!
archdetect says x86_64/amd/zen3
Using x86_64/amd/zen3 as software subdirectory
...
Environment set up to use EESSI pilot software stack, have fun!

{EESSI 2023.06} $ module load R/4.3.2-gfbf-2023a

{EESSI 2023.06} $ which R
/cvmfs/software.eessi.io/versions/2023.06/software/linux/x86_64/
amd/zen3/software/R/4.3.2-gfbf-2023a/bin/R

{EESSI 2023.06} $ R --version
R version 4.3.2
```

Automatically detects CPU microarchitecture

Website: https://eessi.io

Join our Slack channel (see join link on website)

Documentation: https://eessi.io/docs

Blog: https://eessi.io/docs/blog

GitHub: https://github.com/EESSI

Paper (open access): https://doi.org/10.1002/spe.3075

EESSI YouTube channel

Bi-monthly online meetings

(first Thursday, odd months, 2pm CEST)

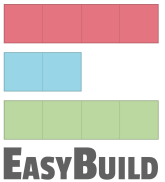
Webinar series: Different aspects of EESSI

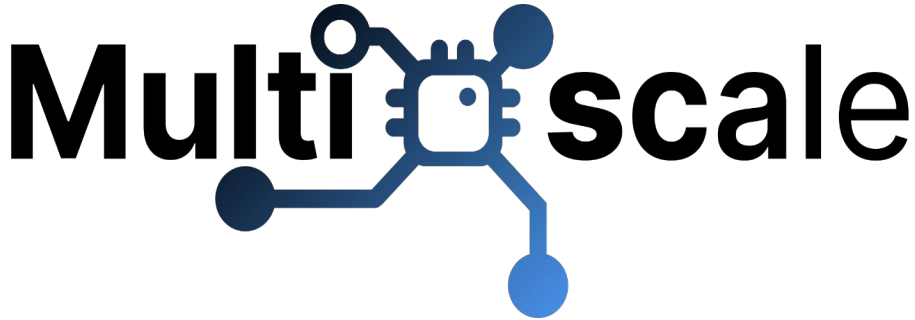
5 Mondays in a row May-June 2025

https://eessi.io/docs/training/2025/webinar-series-2025Q2

- Introduction to EESSI
- Introduction to CernVM-FS
- Introduction to EasyBuild
- EESSI for CI/CD *(26 May)*
- Using EESSI as the base for a system stack *(2 June)*

Slides + recordings available





Co-funded by the European Union

EuroHPC Joint Undertaking

Web page: multixscale.eu

Facebook: MultiXscale

Twitter: @MultiXscale

LinkedIn: MultiXscale

BlueSky: MultiXscale

KEMIJSKI INŠTITUT

UNIVERSITAT DE BARCELONA

Universität Stuttgart

UNIVERSITAS BERGENSIS

SORBONNE UNIVERSITÉ

Université de Toulouse

Consiglio Nazionale delle Ricerche

MAX-PLANCK-GESELLSCHAFT

HPC Now!

university of groningen

SURF

JÜLICH Forschungszentrum

GHENT UNIVERSITY

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

LEONARDO

iit ISTITUTO ITALIANO DI TECNOLOGIA

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and countries participating in the project under grant agreement No 101093169.