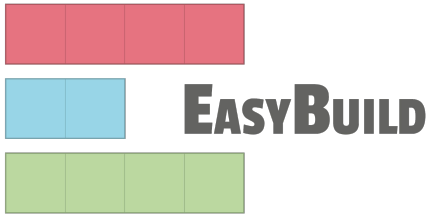
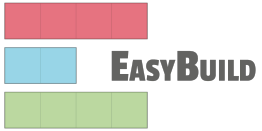


EasyBuild + EESSI tutorial

12+22 May 2023, Ghent (Belgium)

https://users.ugent.be/~kehoste/easybuild-tutorial-20230512.pdf

Practical information



- We will reuse the content of the April 2023 workshop on EasyBuild + EESSI:

  https://tutorial.easybuild.io/2023-eb-eessi-uk-workshop

- If you need help, consider asking questions in the EasyBuild Slack

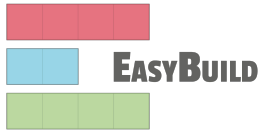
- We will use VSC account + HPC-UGent infrastructure for hands-on and exercises.

  There will be some minor differences in output of demos + hands-on

  because of this, because `/easybuild` does not exist!

  (also, ignore the “Practical information” part of the tutorial website)

What is EasyBuild?



- **EasyBuild is a software build and installation framework**
- Strong focus on scientific software, performance, and HPC systems
- Open source (GPLv2), implemented in Python (2.7, 3.5+)
- Brief history:
  - Created in-house at HPC-UGent in 2008
  - First released publicly in Apr'12 (version 0.5)
  - EasyBuild 1.0.0 released in Nov'12 (during SC12)
  - Worldwide community has grown around it since then!

https://easybuild.io

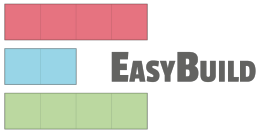
https://docs.easybuild.io

https://github.com/easybuilders

https://easybuild.io/join-slack

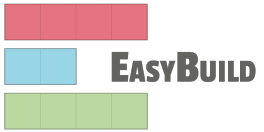
Twitter: @easy_build

EasyBuild in a nutshell



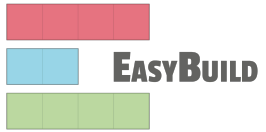
- **Tool to provide a *consistent and well performing* scientific software stack**
- Uniform interface for installing scientific software on HPC systems
- Saves time by *automating* tedious, boring and repetitive tasks
- Can empower scientific researchers to self-manage their software stack
- **A platform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

Key features of EasyBuild (1/2)



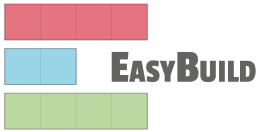
- Supports fully **autonomously** installing (scientific) software, including dependencies, generating environment module files, ...
- **No admin privileges are required** (only write permission to installation prefix)
- Highly configurable, easy to extend, support for hooks, easy customisation
- Detailed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

Key features of EasyBuild (2/2)



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

Focus points in EasyBuild



Performance

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

Reproducibility

- Compiler, libraries, and required dependencies are mostly controlled by EasyBuild
- Fixed software versions for compiler, libraries, (build) dependencies, …

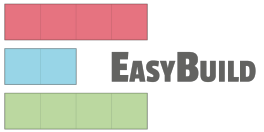
Community effort

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

What EasyBuild is *not*

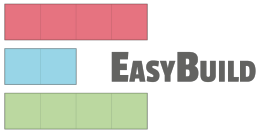
- EasyBuild is **not YABT (Yet Another Build Tool)**
  - It does not try to replace `CMake`, `make`, `pip`, etc.
  - It wraps around those tools and automates installation procedures
- EasyBuild does **not replace traditional Linux package managers** (`yum`, `dnf`, `apt`, ...)
  - You should still install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

EasyBuild terminology



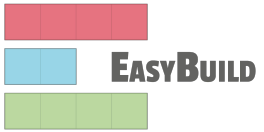
- It is important to briefly explain some terminology often used in EasyBuild
- Some concepts are specific to EasyBuild: easyblocks, easyconfigs, …
- Overloaded terms are clarified: modules, extensions, toolchains, …

EasyBuild terminology: framework



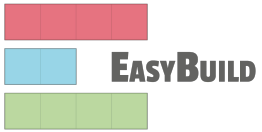
- The EasyBuild framework is the **core of EasyBuild**
- **Collection of Python modules**, organised in packages
- Implements **common functionality** for building and installing software
- Support for applying patches, running commands, generating module files, ...
- Examples: `easybuild.toolchains`,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

EasyBuild terminology: easyblock



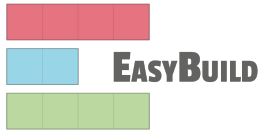
- A **Python module** that implements a specific software installation procedure
  - Can be viewed as a “plugin” to the EasyBuild framework
- **Generic easyblocks** for “standard” stuff: `cmake` + `make` + `make install`, Python packages, etc.
- **Software-specific easyblocks** for complex software (OpenFOAM,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

EasyBuild terminology: easyconfig file



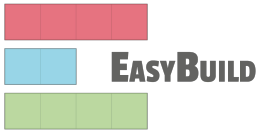
- Text file that specifies what EasyBuild should install (in Python syntax)
- **Collection of values for easyconfig parameters** (key-value definitions)
- Filename typically ends in '`.eb`'
- Specific filename is expected in some contexts (when resolving dependencies)
  - Should match with values for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

EasyBuild terminology: easystack file



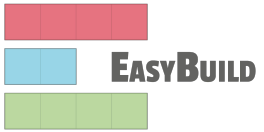
- New concept since EasyBuild v4.3.2 (Dec'20), **experimental feature**
- Concise description for software stack to be installed (in YAML syntax)
- Basically **specifies a set of easyconfig files** (+ associated info)
- Still a work-in-progress, only basic functionality implemented currently
- More info: https://docs.easybuild.io/en/latest/Easystack-files.html

EasyBuild terminology: extensions



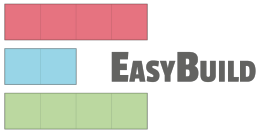
- **Additional software that can be installed *on top* of other software**
- Common examples: Python packages, Perl modules, R libraries, …
- *Extensions* is the general term we use for this type of software packages
- Can be installed in different ways:
  - As a stand-alone software packages (separate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

EasyBuild terminology: dependencies



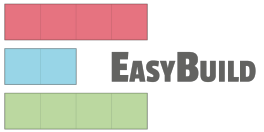
- Software that is **required to build/install or run other software**
- **Build dependencies**: only required when building/installing software (not to use it)
  - Examples: CMake, pip, pkg-config, …
- **Run-time dependencies**: (also) required to use the installed software
  - Examples: Python, Perl,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

EasyBuild terminology: toolchains



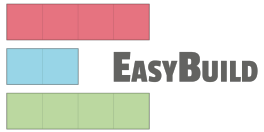
- **Compiler toolchain: set of compilers + libraries for MPI, BLAS/LAPACK, FFT, …**
- Toolchain component: a part of a toolchain (compiler component, etc.)
- **Full toolchain**: C/C++/Fortran compilers + libraries for MPI, BLAS/LAPACK, FFT
- **Subtoolchain** (partial toolchain): compiler-only, only compiler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

EasyBuild terminology: modules



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

Bringing all EasyBuild terminology together



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

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

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

Installing EasyBuild: requirements

- **Linux** as operating system (CentOS, RHEL, Ubuntu, Debian, SLES, ...)
  - EasyBuild also works on macOS, but support is very basic
- **Python** 2.7 or 3.5+
  - Only Python standard library is required for core functionality of EasyBuild
  - Using Python 3.6+ is highly recommended!
- An **environment modules tool** (`module` command)
  - Default is Lua-based Lmod implementation, highly recommended!
  - Tcl-based implementations are also supported

Installing EasyBuild: different options

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

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

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

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

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

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

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

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

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

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

Verifying the EasyBuild installation

- Check EasyBuild version:

```
eb --version
```

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

```
eb --help
```

- Show the current (default) EasyBuild configuration:

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

- Show system information:

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

Updating EasyBuild

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

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

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

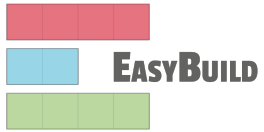
- Use current EasyBuild to install latest EasyBuild release as a module:

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

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

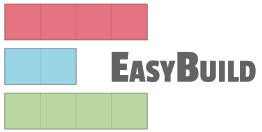
```
module load EasyBuild/4.7.1
```

Configuring EasyBuild



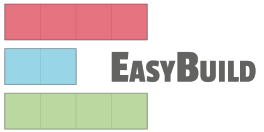
- EasyBuild should work fine out-of-the-box if you are using Lmod as modules tool
- ... but it will (ab)use `$HOME/.local/easybuild` to install software into, etc.
- It is ***strongly*** recommended to configure EasyBuild properly!
- Main questions you should ask yourself:
  - Where should EasyBuild 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 (~270) is available via `eb --help`

Configuration levels

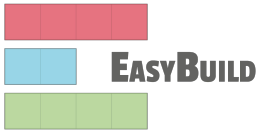


- There are 3 different configuration levels in EasyBuild:
  - **Configuration files**
  - **Environment variables**
  - **Command line options to the `eb` command**

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

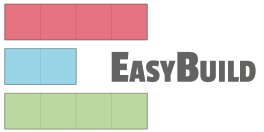
- Hierarchical configuration:
  - Configuration files override default settings
  - Environment variables override configuration files
  - eb command line options override environment variables

EasyBuild configuration files



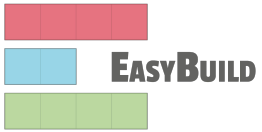
- EasyBuild configuration files are in standard INI format (`key=value`)
- EasyBuild considers multiple locations for configuration files:
  - User-level: `$HOME/.config/easybuild/config.cfg` (or via `$XDG_CONFIG_HOME`)
  - System-level: `/etc/easybuild.d/*.cfg` (or via `$XDG_CONFIG_DIRS`)
  - See output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

$EASYBUILD_* environment variables



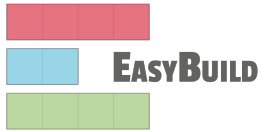
- Very convenient way to configure EasyBuild
- **There is an `$EASYBUILD_*` environment variable for each configuration setting**
  - Use all capital letters
  - Replace every dash (–) character with an underscore (_)
  - Prefix with `EASYBUILD_`
  - Example: `module-syntax` → `$EASYBUILD_MODULE_SYNTAX`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

Command line options for `eb` command



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

Inspecting the current configuration

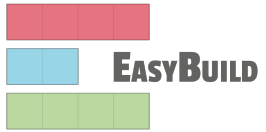


- It can be difficult to remember how EasyBuild was configured
- Output produced by **`eb --show-config`** is useful to remind you
- Shows configuration settings that are different from default
- Always shows a couple of key configuration settings
- Also shows on which level each configuration setting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

Inspecting the current configuration: example

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

$ export EASYBUILD_BUILDPATH=/tmp/$USER/build

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

Minimal EasyBuild configuration for hands-on



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

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

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

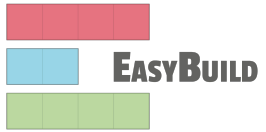
- **Use *local* temporary directory for build directories** (important!)

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

- **Ensure prepared software stack is visible** via "`module avail`":

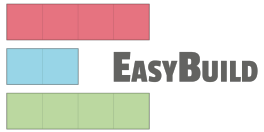
  ```
  module avail foss/2021b
  ```

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

Inspecting easyconfigs via `eb --show-ec`

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

```
$ eb --show-ec TensorFlow-2.6.0-foss-2021a.eb
easyblock = 'PythonBundle'

name = 'TensorFlow'
version = '2.6.0'

homepage = 'https://www.tensorflow.org/'
description = "An open-source software library for Machine Intelligence"

toolchain = {'name': 'foss', 'version': '2021a'}
toolchainopts = {'pic': True}
…
```

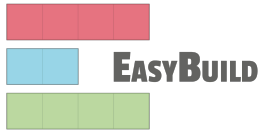
Checking dependencies via `eb --dry-run`

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

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

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

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

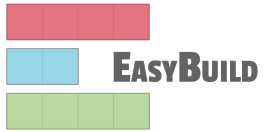


To check which dependencies are still *missing*, use `eb --missing` (or `eb -M`):

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

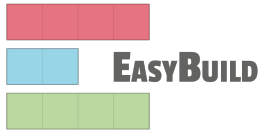
```
$ eb PyTables-3.6.1-foss-2021b.eb -M
3 out of 69 required modules missing:
* LZO/2.10-GCCcore-11.2.0 (LZO-2.10-GCCcore-11.2.0.eb)
* Blosc/1.21.1-GCCcore-11.2.0 (Blosc-1.21.1-GCCcore-11.2.0.eb)
* PyTables/3.6.1-foss-2021b (PyTables-3.6.1-foss-2021b.eb)
```

Inspecting software install procedures



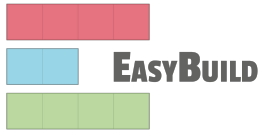
- EasyBuild can quickly unveil how exactly it *would* install an easyconfig file
- Via `eb --extended-dry-run` (or `eb -x`)
- Produces detailed output in a matter of seconds
- Software is not actually installed, all shell commands and file operations are skipped!
- Some guesses and assumptions are made,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

Inspecting software install procedures: example



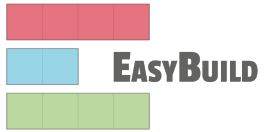

```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
...

preparing... [DRY RUN]

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

Loading toolchain module...

module load GCC/11.2.0

Loading modules for dependencies...

module load bzip2/1.0.8-GCCcore-11.2.0
module load zlib/1.2.11-GCCcore-11.2.0
module load XZ/5.2.5-GCCcore-11.2.0
```

Inspecting software install procedures: example




```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
...
Defining build environment...
  ...
  export CXX='g++'
  export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno -fPIC'
  ...

configuring... [DRY RUN]

[configure_step method]
  running command "./bootstrap.sh --with-toolset=gcc
  --prefix=/tmp/example/Boost/1.77.0-GCC-11.2.0 --without-libraries=python,mpi"
  (in /tmp/example/build/Boost/1.77.0/GCC-11.2.0/Boost-1.77.0)
```

Inspecting software install procedures: example

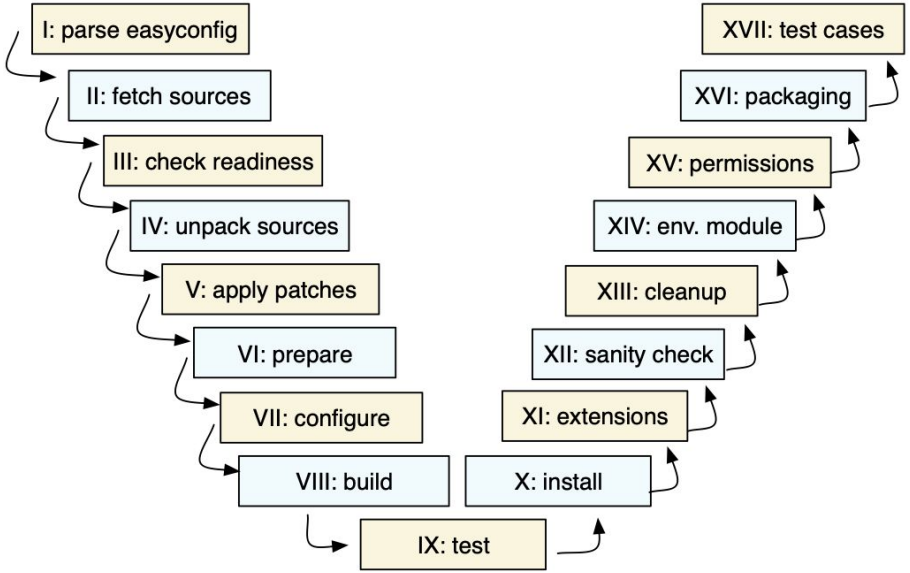



```
$ eb Boost-1.77.0-GCC-11.2.0.eb -x
...

[sanity_check_step method]
Sanity check paths - file ['files']
  * lib/libboost_system-mt-x64.so
  * lib/libboost_system.so
  * lib/libboost_thread-mt-x64.so
Sanity check paths - (non-empty) directory ['dirs']
  * include/boost
Sanity check commands
  (none)

...
```

Installing software with EasyBuild

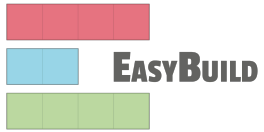
- To install software with EasyBuild, just run the `eb` command:
  - `eb SAMtools-1.14-GCC-11.2.0.eb`
- If any dependencies are still missing, you will need to also use `--robot`:
  - `eb BCFtools-1.14-GCC-11.2.0.eb --robot`
- To see more details while the installation is running, enable trace mode: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

Step-wise installation procedure



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

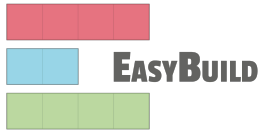
Using software installed with EasyBuild



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

```
# inform modules tool about modules installed with EasyBuild
module use $HOME/easybuild/modules/all
# check for available modules for BCFtools
module avail BCFtools
# load BCFtools module to "activate" the installation
module load BCFtools/1.14-GCC-11.2.0
```

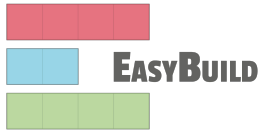
Stacking software installations



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

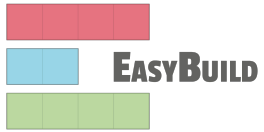
```
module use /easybuild/modules/all

eb --installpath $HOME/easybuild my-software.eb
```

Troubleshooting failing installations



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

Troubleshooting: error messages

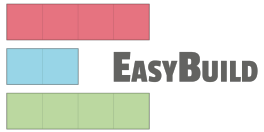


- When EasyBuild detects that something went wrong, it produces an error
- Very often due to a shell command that produced a non-zero exit code...
- Sometimes the problem is clear directly from the error message:

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

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

Troubleshooting: log files

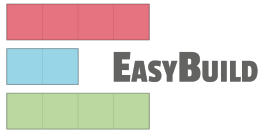


- EasyBuild keeps track of the installation in a detailed log file
- During the installation, it is stored in a temporary directory:

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

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

Troubleshooting: navigating log files



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

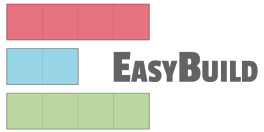
```
== 2022-05-25 13:11:19,968 run.py:222 INFO running cmd:  make -j 9
```

- Different steps of installation procedure are clearly marked:

```
== 2022-05-25 13:11:48,817 example INFO Starting sanity check step
```

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

Troubleshooting: inspecting the build directory

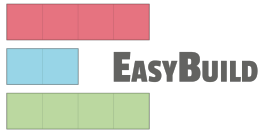


- EasyBuild leaves the build directory in place when the installation failed

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

- Can be useful to inspect the contents of the build directory for debugging
- For example:
  - Check `config.log` when `configure` command failed
  - Check `CMakeFiles/CMakeError.log` when `cmake` command failed (good luck...)

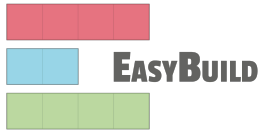
Troubleshooting: hands-on exercise



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

```
$ eb subread.eb
...
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/example/Subread/2.0.3/GCC-8.5.0): build failed (first 300 chars):
Couldn't find file subread-2.0.3-source.tar.gz anywhere, and downloading
it didn't work either...
Paths attempted (in order): ...
```

Adding support for additional software

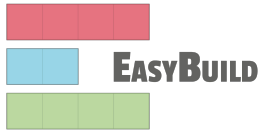


- Every installation performed by EasyBuild requires an easyconfig file
- Easyconfig files can be:
  - Included with EasyBuild itself (or obtained elsewhere)
  - Derived from an existing easyconfig (manually or automatic)
  - Created from scratch
- Most easyconfigs leverage a generic easyblock
- Sometimes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

Easyblocks vs easyconfigs

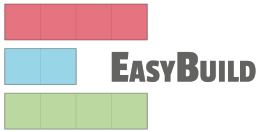
- When can you get away with using an easyconfig leveraging a generic easyblock?
- When is a software-specific easyblock really required?
- Easyblocks are “implement once and forget”
- Easyconfig files leveraging a generic easyblock can become too involved (subjective)
- Reasons to consider implementing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

Writing easyconfig files



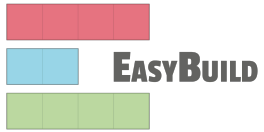
- Collection of easyconfig parameter definitions (Python syntax), collectively specify what to install
- Some easyconfig parameters are mandatory, and **must** always be defined:
  `name`, `version`, `homepage`, `description`, `toolchain`
- Commonly used easyconfig parameters (but strictly speaking not required):
  - `easyblock` (by default derived from software name)
  - `versionsuffix`
  - `source_urls`, `sources`, `patches`, `checksums`
  - `dependencies`, `builddependencies`
  - `preconfigopts`, `configopts`, `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,11.2.0`
- Changing specific easyconfig parameters (limited): `eb --try-amend ...`
- Note the “try” aspect: additional changes may be required to make installation work
- EasyBuild does save the so generated easyconfig files in the `easybuild` subdirectory of the software installation directory and in the easyconfig archive.

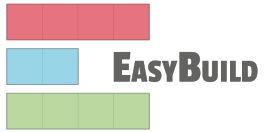
Copying easyconfig files



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

```
$ eb --copy-ec SAMtools-1.14-GCC-11.2.0.eb SAMtools.eb
...
SAMtools-1.14-GCC-11.2.0.eb copied to SAMtools.eb
```

Hands-on: creating easyconfig files

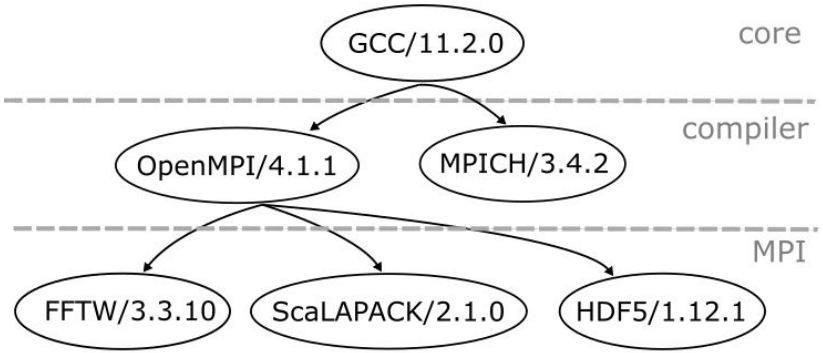


- Step-wise example + exercise of creating an easyconfig file from scratch
- For fictitious software packages: `eb-tutorial` + `py-eb-tutorial`
- **Great exercise to work through these yourself!**

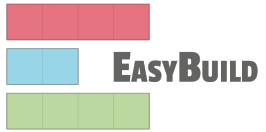
```
name = 'eb-tutorial'

version = '1.0.1'

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

description = "EasyBuild tutorial example"
```

Flat vs hierarchical module naming schemes

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

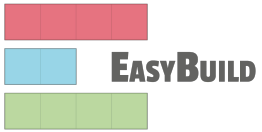


Pros and cons of using a flat vs hierarchical MNS



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

Custom module naming schemes with EasyBuild



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

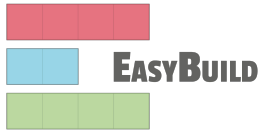
- To configure EasyBuild to use your custom module naming scheme:

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

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

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

Hands-on example: installing HDF5 in an HMNS



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

```
module unuse $MODULEPATH
```

- Configure our setup to reuse the existing software installations

```
export EASYBUILD_INSTALLPATH_SOFTWARE=/easybuild/software

export EASYBUILD_MODULE_NAMING_SCHEME=HierarchicalMNS

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

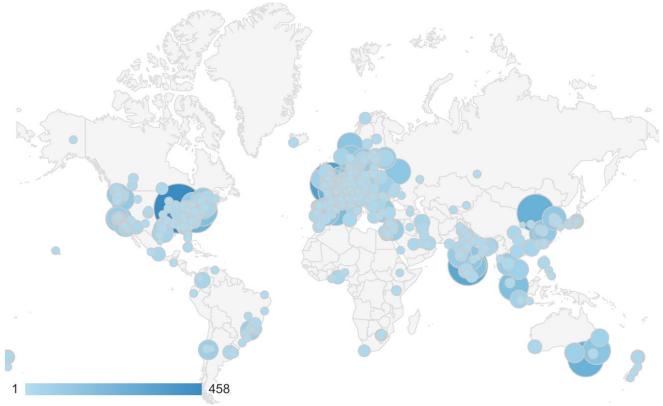
- Re-generate all modules for HDF5 using the new scheme (42 modules)

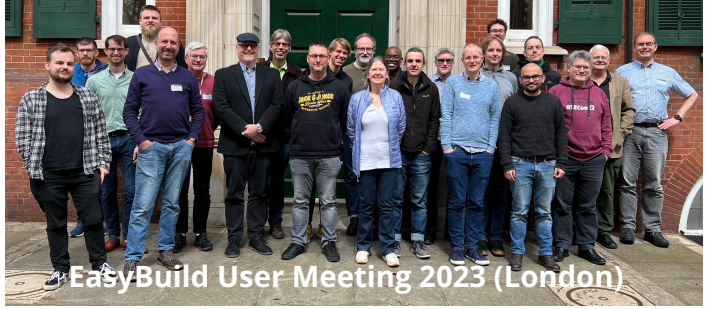
```
eb HDF5-1.12.1-gompi-2021b.eb --robot --module-only
```

- Explore the new hierarchy

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

The EasyBuild community



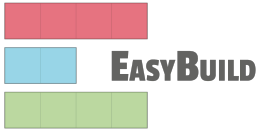


EASYBUILD

- Documentation is read all over the world
- HPC sites, consortia, and companies
- Slack: >700 members, ~180 active members per week, 311k messages
- Regular online conf calls… and we even meet in person sometimes!

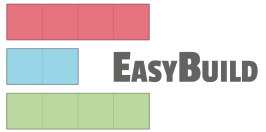
https://easybuilders.github.io/easybuild-tutorial/2022-isc22/community

Why Contribute Back?



- Creating PRs upstream: get reviews, suggestions from software installation experts
- Participating in the EasyBuild community: connect with HPC teams from all over the world
- Keeping in sync with the EasyBuild repository to maximally profit from upstream work:
  - New software recipes, new version of existing software
  - Bug fixes
  - Enhancements, additional functionality
  - Performance improvements

Contributing to EasyBuild



There are several ways to contribute to EasyBuild, including:

- Providing feedback (positive or negative)
- Reporting bugs
- Joining the discussions (mailing list, Slack, conf calls)
- Sharing suggestions/ideas for enhancements & additional features
- Contributing easyconfigs, enhancing easyblocks,

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

GitHub integration features

- EasyBuild has strong integration with GitHub, which facilitates contributions
- Some additional Python packages required for this: GitPython, keyring
- Also requires some additional configuration, incl. providing a GitHub token
- **Enables creating, updating, reviewing pull requests using `eb` command!**
- Makes testing contributions very easy (~2,500 easyconfig pull requests per year!)
- Extensively documented:
  https://docs.easybuild.io/integration-with-github

Opening a pull request in 1, 2, 3

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

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

one single `eb` command

no git commands

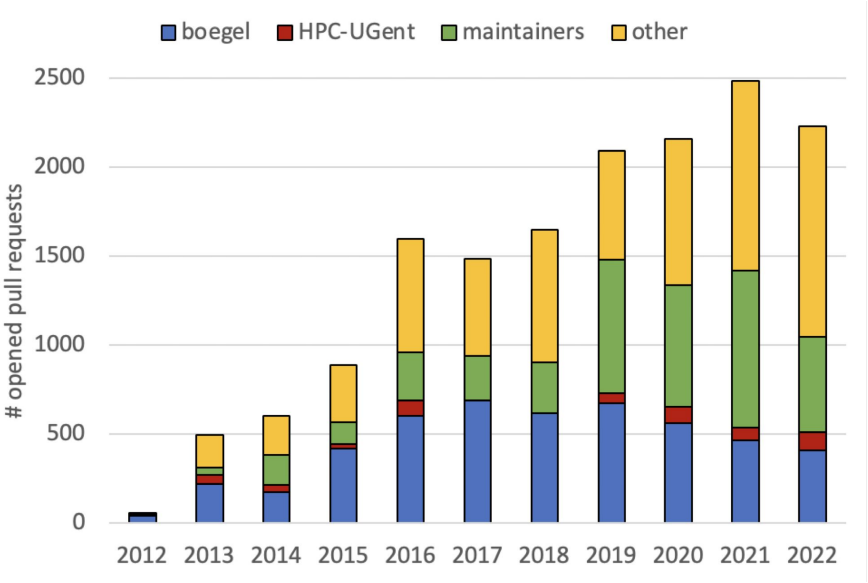
no GitHub interaction

metadata is automatically derived from easyconfig

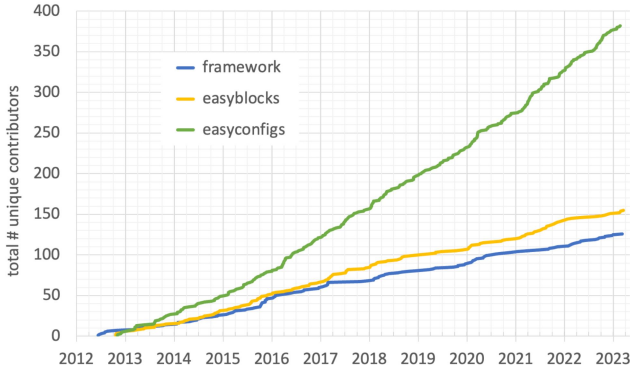
saves a lot of time!

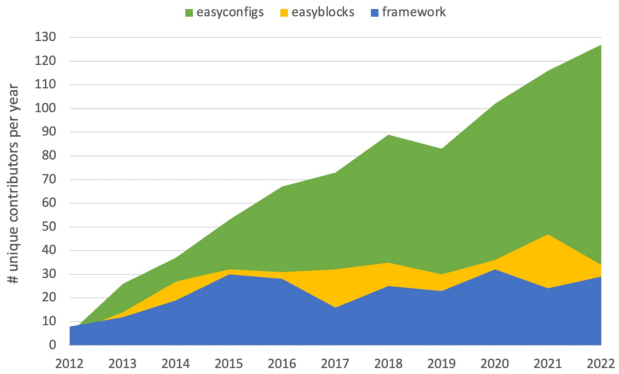
`eb --new-pr sklearn.eb`

EasyBuild Contributions & Contributors

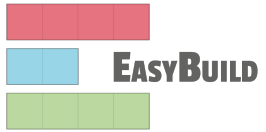


(only central easyconfigs repository)



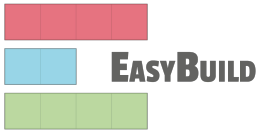


Customizing EasyBuild via Hooks



- 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: https://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:

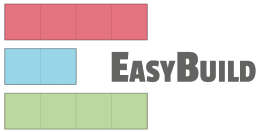
  https://github.com/easybuilders/easybuild-framework/tree/develop/contrib/hooks

Hooks: examples

Ensure that software is installed with a specific license group:

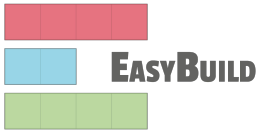
```
def parse_hook(self, *args, **kwargs):
    if self.name == 'Example':
        # use correct license group on Hortense
        if os.getenv('VSC_INSTITUTE_CLUSTER') == 'dodrio':
            self['group'] = 'gli_hortense_example'
```

Implementing Easyblocks



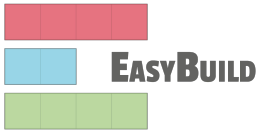
- An easyblock may be required for more complex software installations
- This requires some Python skills, and familiarity with EasyBuild framework
- A software-specific easyblock can derived from a generic easyblock
- Focus is usually on configure/build/installs steps of installation procedure
- See 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 most commonly used job backend that EasyBuild can use
- `export EASYBUILD_JOB_BACKEND=Slurm`
- Then use "`eb … --job --robot`"
- See also https://docs.easybuild.io/submitting-jobs

Using EasyBuild as a Python Library

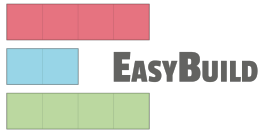


- You can use EasyBuild as a Python library: `from easybuild import …`
- Setting up the EasyBuild configuration first is required:

```
from easybuild.tools.options import set_up_configuration

set_up_configuration()
```

- You can write your own Python scripts that leverage EasyBuild!

Questions?



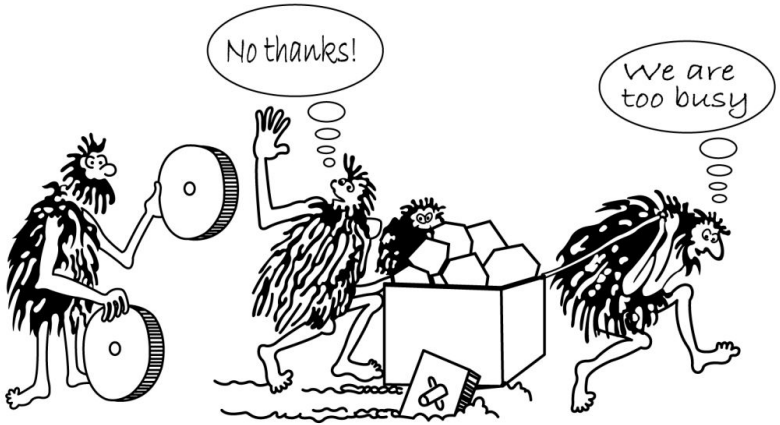
- Website: https://easybuild.io
- Documentation: https://docs.easybuild.io
- Tutorials: https://tutorial.easybuild.io
- Yearly EasyBuild User Meeting: https://easybuild.io/eum
- Getting help:
  - Mailing list: https://lists.ugent.be/wws/subscribe/easybuild
  - Slack: https://easybuild.slack.com - https://easybuild.io/join-slack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

Introduction to EESSI

EASYBUILD

EESSI

EUROPEAN ENVIRONMENT FOR
SCIENTIFIC SOFTWARE INSTALLATIONS



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 (+ WSL, and possibly macOS) 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**

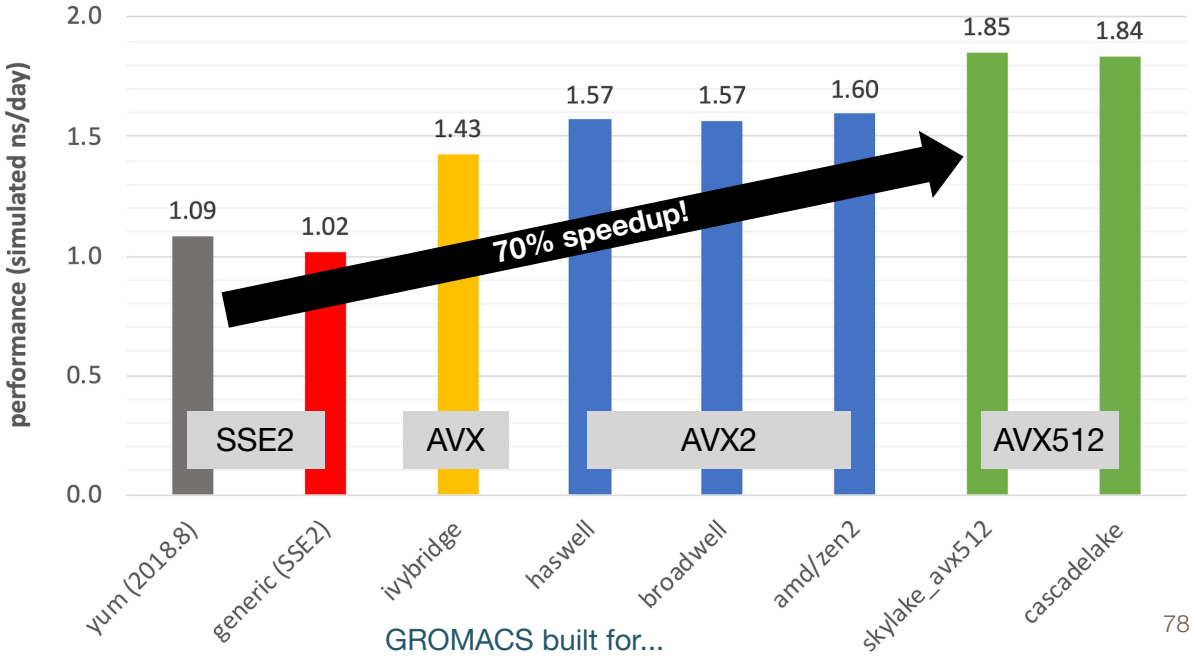
https://www.eessi-hpc.org

https://eessi.github.io/docs (try out the pilot setup!)

Optimized scientific software installations

- Software should be optimized for the system it will run on
- Impact on performance is often significant for scientific software

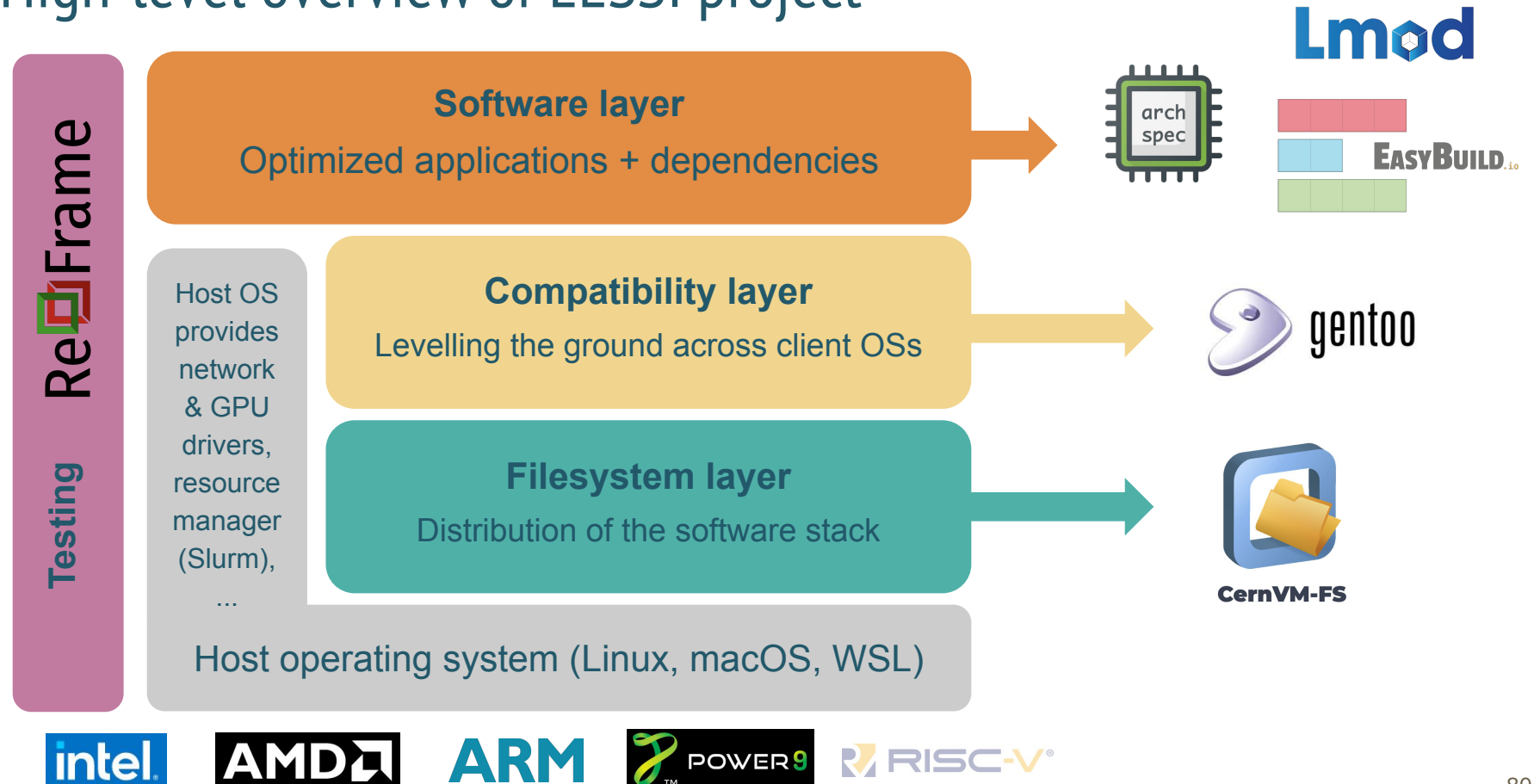
- Example: GROMACS 2020.1 (PRACE benchmark, Test Case B)
- Metric: (simulated) ns/day, higher is better
- Test system: dual-socket Intel Xeon Gold 6420 (Cascade Lake, 2x18 cores)
- **Performance of different GROMACS binaries, on exact same hardware/OS**



Major goals of EESSI

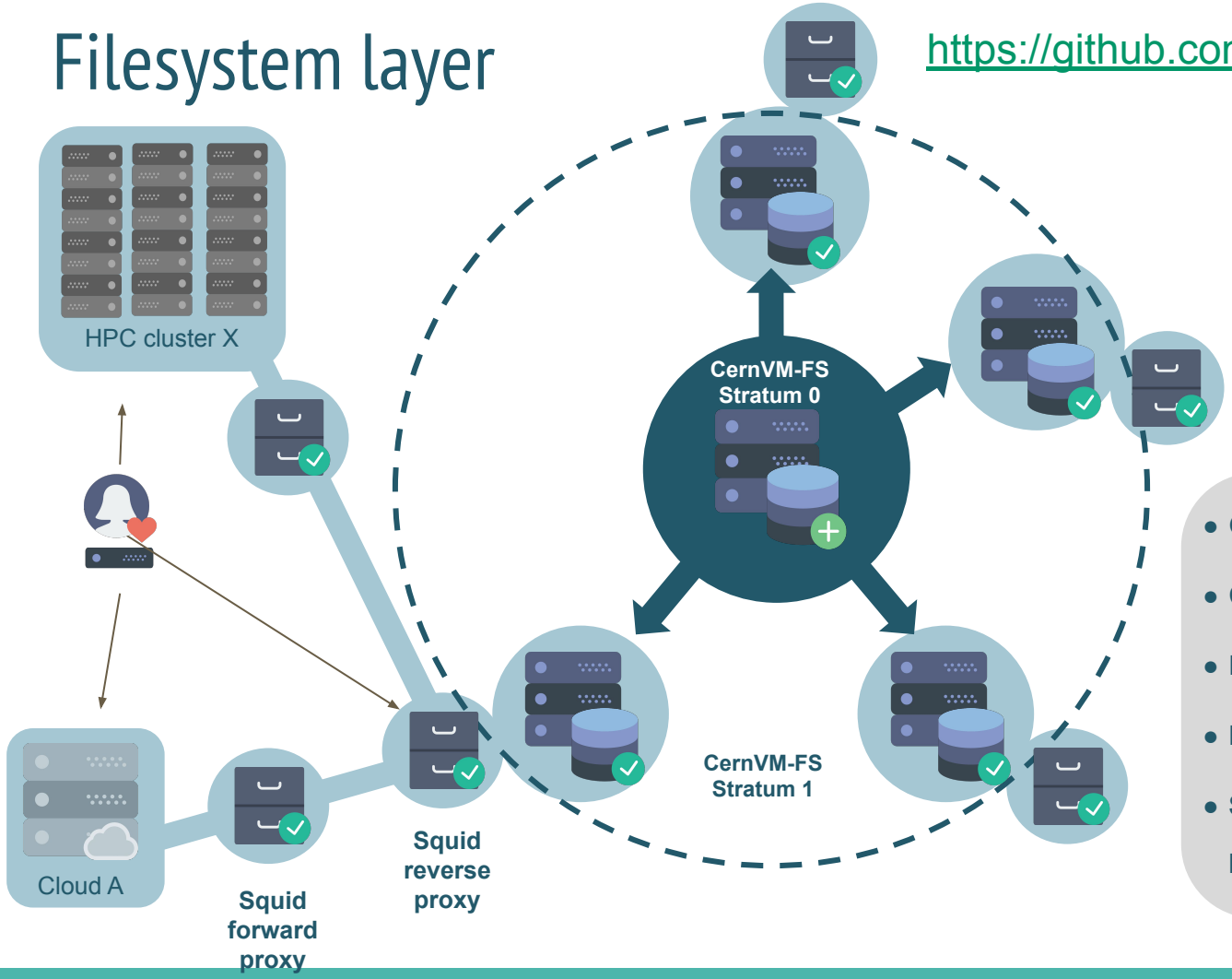
- **Avoid duplicate work** (for researchers, HPC support teams, …)
  - Tools that automate software installation process (EasyBuild, Spack) are not sufficient
  - Go beyond sharing build recipes => work towards a shared software stack
- Providing a truly **uniform software stack**
  - Use the (exact)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

High-level overview of EESSI project



Filesystem layer

https://github.com/EESSI/filesystem-layer



CernVM-FS

https://cvmfs.readthedocs.io

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

(icons via https://www.flaticon.com/authors/smashicons)

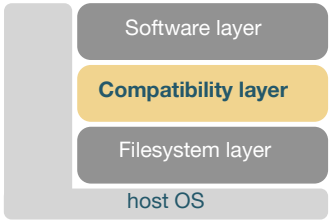
Compatibility layer

https://github.com/EESSI/compatibility-layer

- **Gentoo Prefix** installation (in `/cvmfs/.../compat/<os>/<arch>/`)
- Set of tools & libraries installed in non-standard location
- Limited to low-level stuff, incl. glibc (no Linux kernel or drivers)
  - Similar to the OS layer in container images
- Only targets a supported processor **family** (`aarch64`, `ppc64le`, `x86_64`, `riscv64`)
- **Levels the ground for different client operating systems** (Linux distros, later also macOS?)
- Currently in pilot repository:

```
/cvmfs/pilot.eessi-hpc.org/versions/2021.12/compat/linux/aarch64
/cvmfs/pilot.eessi-hpc.org/versions/2021.12/compat/linux/ppc64le
/cvmfs/pilot.eessi-hpc.org/versions/2021.12/compat/linux/x86_64
```

powered by



Software layer

https://github.com/EESSI/software-layer

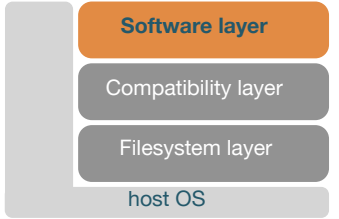
- Provides scientific software applications, libraries, and dependencies
- **Optimized for specific CPU microarchitectures** (Intel Haswell, ...)
  - Separate subdirectory/tree for each (in `/cvmfs/.../software/...`)
- **Leverages libraries** (like glibc) **from compatibility layer** (not from host OS)
- Installed with EasyBuild, incl. environment module files
- Lmod environment modules tool is used to access installations
- **Best subdirectory for host is selected automatically** via archspec

EESSI

powered by

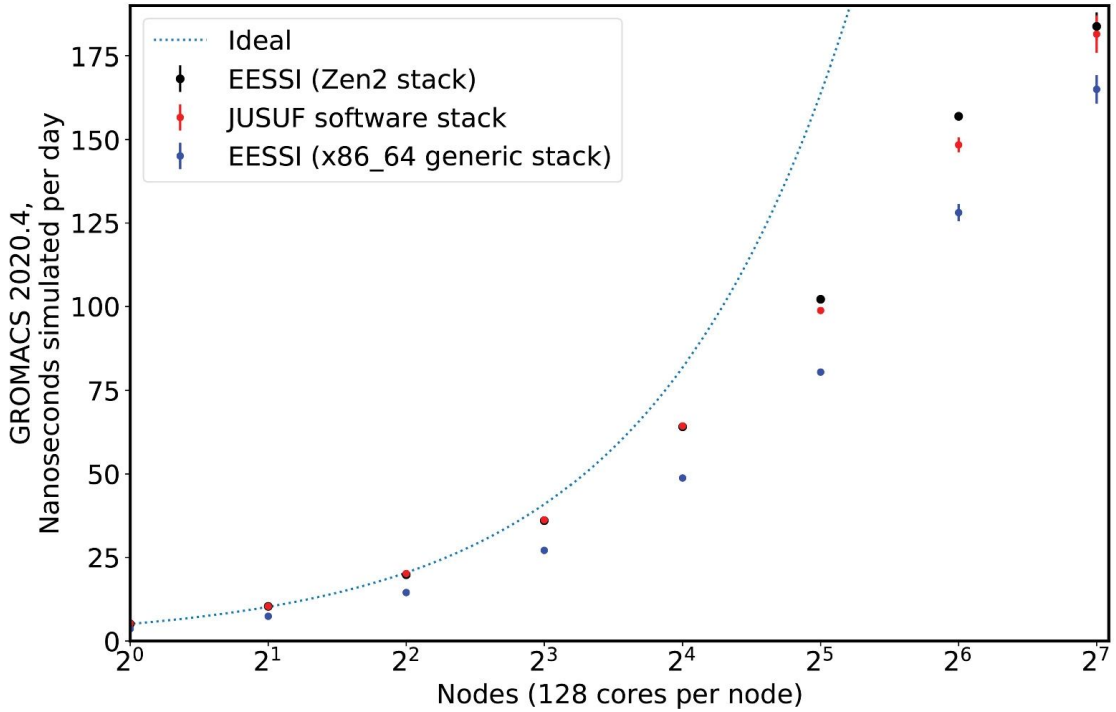
EasyBuild

Lmod



EESSI paper (open access)

doi.org/10.1002/spe.3075



Paper includes proof-of-concept performance evaluation compared to system software stack, performed at JUSUF @ JSC using GROMACS 2020.4, up to 16,384 cores (CPU-only)

Current status of EESSI

- Working **proof of concept** (see https://eessi.github.io/docs/pilot)
- Ansible playbooks, scripts, docs at `https://github.com/eessi`
- CernVM-FS: Stratum 0 @ Univ. of Groningen + four Stratum 1 servers
- Software (CPU-only): Bioconductor, GROMACS, OpenFOAM, R, TensorFlow, ...
- Hardware targets:
  - `{aarch64,ppc64le,x86_64}/generic`
  - `intel/{haswell,skylake_avx512}, amd/{zen2,zen3}, aarch64/{graviton2,graviton3), ppc64le/power9le`

- Supported by Azure and AWS: sponsored credits to develop necessary infrastructure

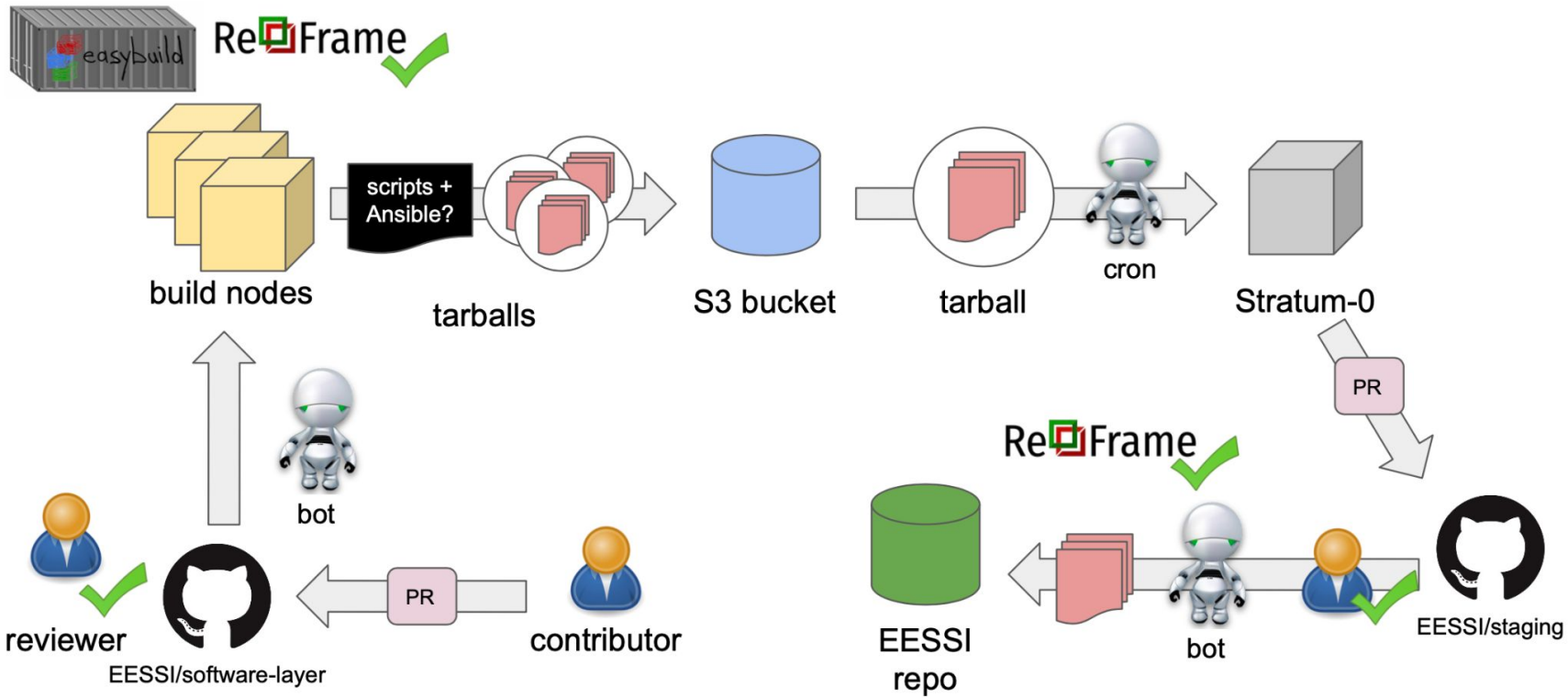
Adding software to EESSI (1/2)

- Current workflow:
  - Human runs software installation script in EESSI build container (on each target CPU arch.)
  - Human runs script to create tarball with added software installations + upload it to AWS S3 bucket
  - Cron script on CernVM-FS central server picks up new uploaded tarballs
  - Creates PR to (private) EESSI/staging repository on GitHub
  - Tarball is automatically ingested into EESSI pilot CernVM-FS repository when PR is merged
- Scripts available in https://github.com/EESSI/software-layer + https://github.com/EESSI/infrastructure
  - `install_software_layer.sh` to install EESSI software layer on top of compat layer
  - `build_container.sh` to easily run software installation script in EESSI build container
  - `create_tarball.sh` to create tarball for added installations (based on fuse-overlayfs upper dir)
  - `eessi-upload-to-staging` to upload into dedicated AWS S3 bucket (requires permissions)

Adding software to EESSI (2/2)

- Problems with current workflow:
  - Still way too manual and time-consuming: human babysitting + taking action
  - Doesn't allow (low-effort) contributions to EESSI software layer from people not familiar with workflow
  - Requires access to (growing) set of target CPUs
    - Different Intel/AMD CPU generations, Arm @ AWS, POWER9, soon also RISC-V?
    - In EESSI pilot v2021.12: `aarch64/generic`, `aarch64/graviton2`, `ppc64le/generic`, `ppc64le/power9`, `x86_64/generic`, `x86_64/amd/zen2`, `x86_64/amd/zen3`, `x86_64/intel/haswell`, `x86_64/intel/skylake_avx512`
  - Requires permissions to upload tarball into AWS S3 bucket for ingestion (who can we trust?)
  - How do we know that provided software builds are not tampered with in any way (knowingly or not)?

Goal: automated procedure with human oversight



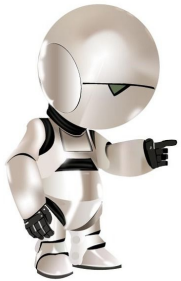
Towards a semi-automated workflow (1/2)

- Goal:
  - Allow contributors to propose additional software to include in EESSI
  - Ideally via a low effort interface: pull requests to GitHub
  - Automatic feedback on whether proposed integration into EESSI works
- Attention points: automation, performance, security, (minimal) human oversight, …
- Conditions for accepting contribution:
  - Software should work correctly in EESSI environment (compat layer, RPATH, long prefix, etc.)
  - Tests should be included to test end user applications (with ReFrame)
  - Software should build + tests should pass on all target CPUs (ideally)

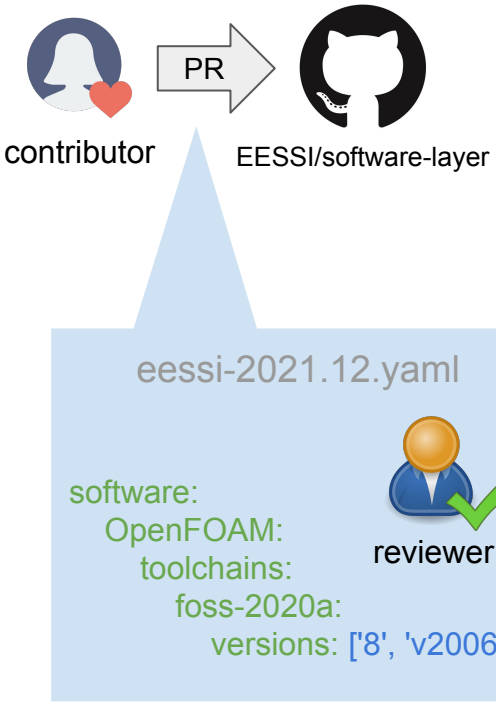
Towards a semi-automated workflow (2/2)

Implement a bot as a GitHub App:

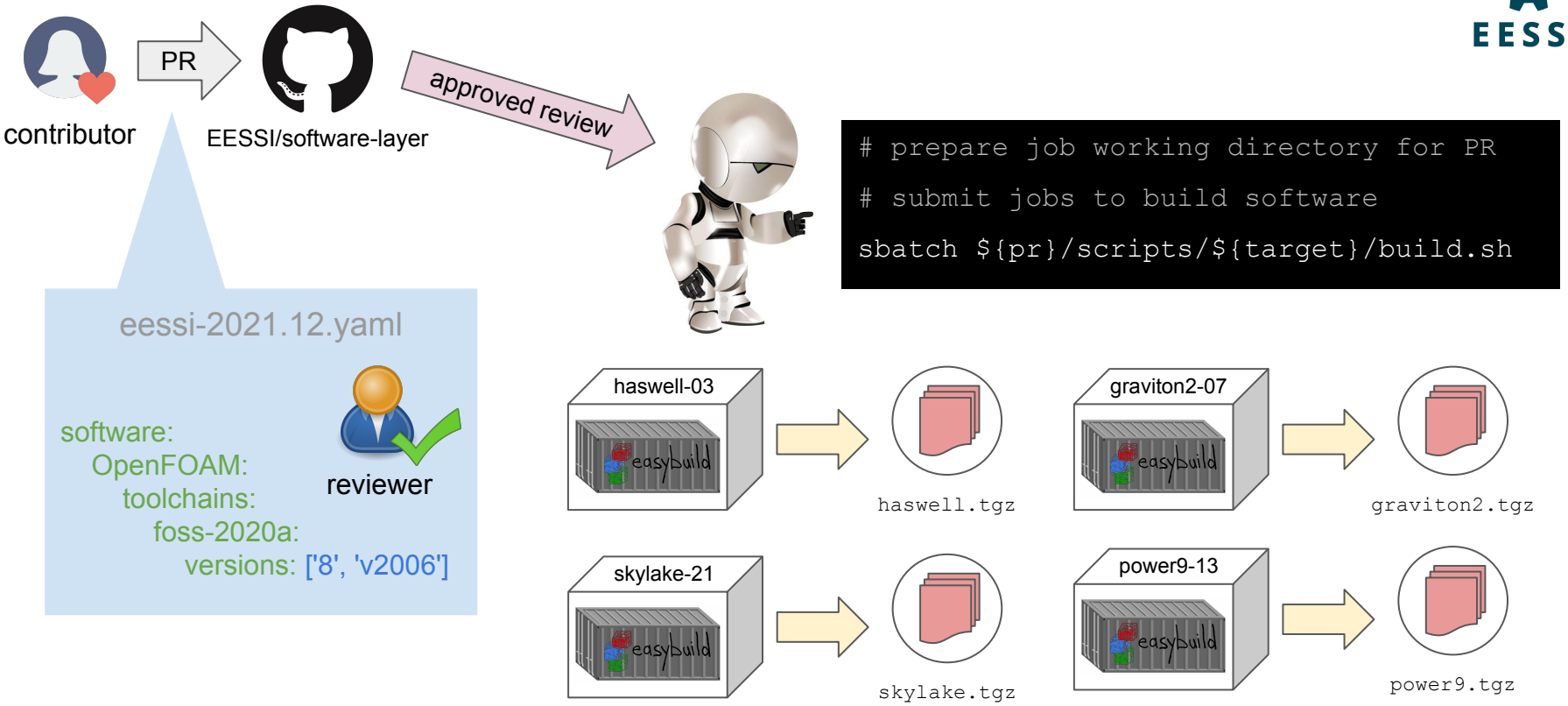
- In Python 3, using Flask (web app framework) + PyGithub (talk to GitHub API)
- Event-based bot that reacts to pull requests (PRs) to `EESSI/software-layer` repository
  - Events include: opening a PR, posting a comment, adding/removing a label, …
- Tasks:
  - Automatically build & install software for different target CPUs (no human intervention)
  - Using EESSI build container, on top of compat layer
  - Run tests to verify that software installation works (in different environment: OS, system, etc.)
  - Get software installations ingested into EESSI repository (after PR is merged?)



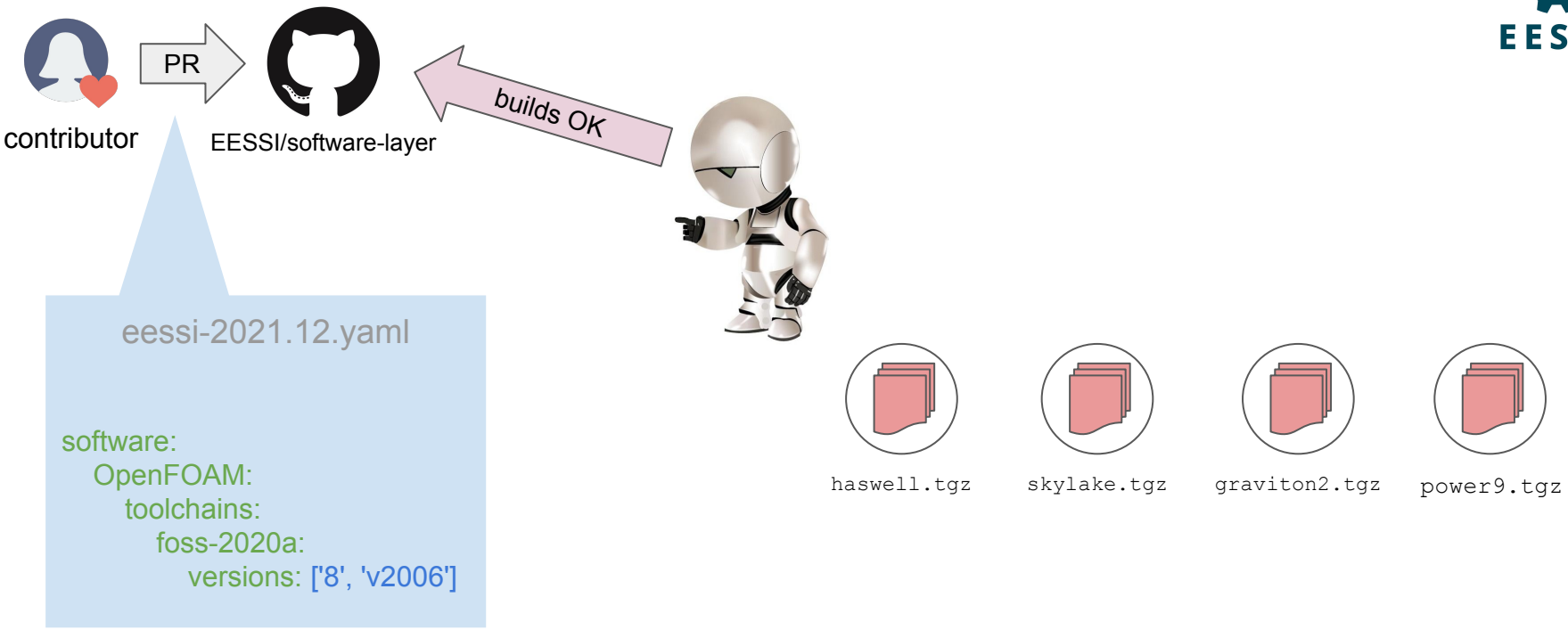
High-level overview of EESSI software bot



High-level overview of EESSI software bot

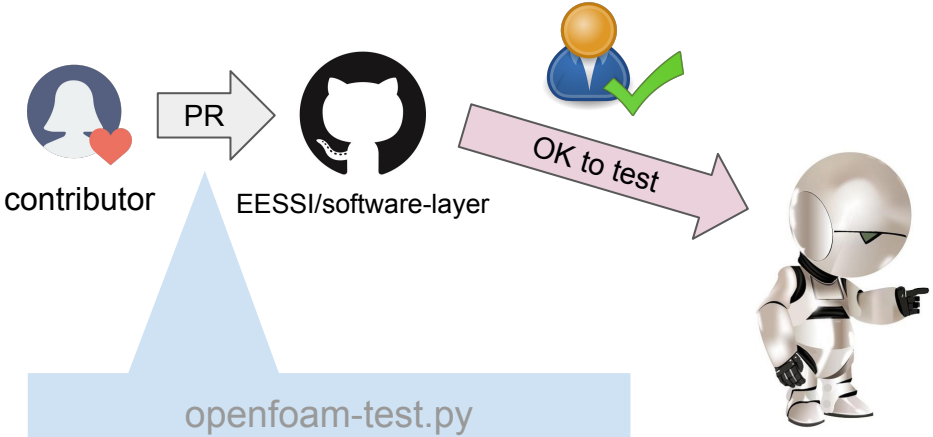


High-level overview of EESSI software bot



High-level overview of EESSI software bot

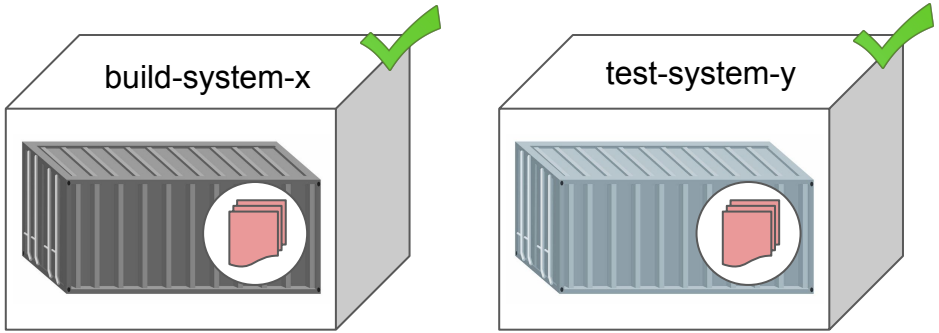
(simplified view)



openfoam-test.py

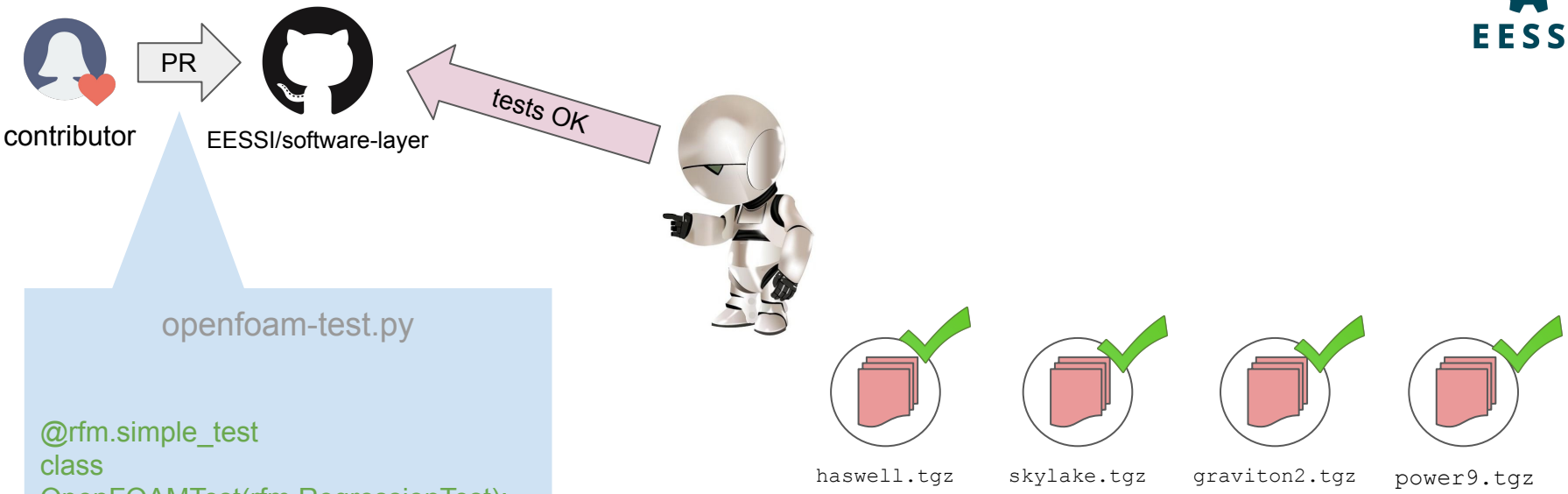
```
@rfm.simple_test
class
OpenFOAMTest(rfm.RegressionTest):
```

```
# submit jobs to test built software
sbatch ${pr}/scripts/${target}/test.sh
```

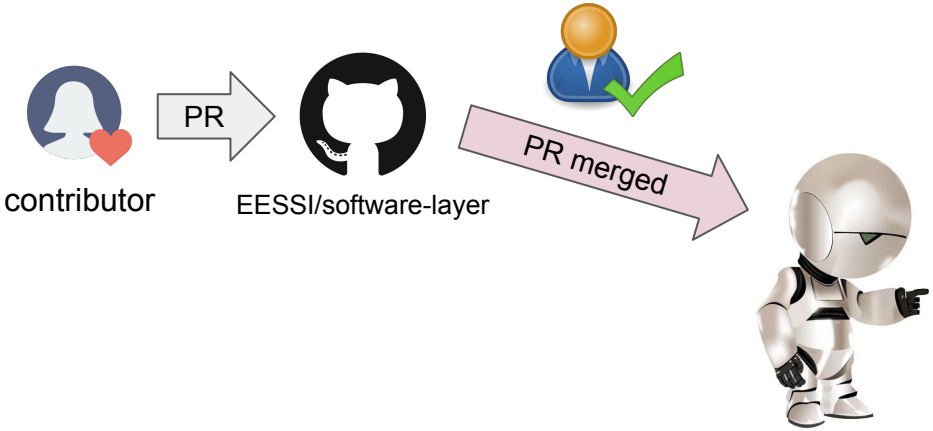


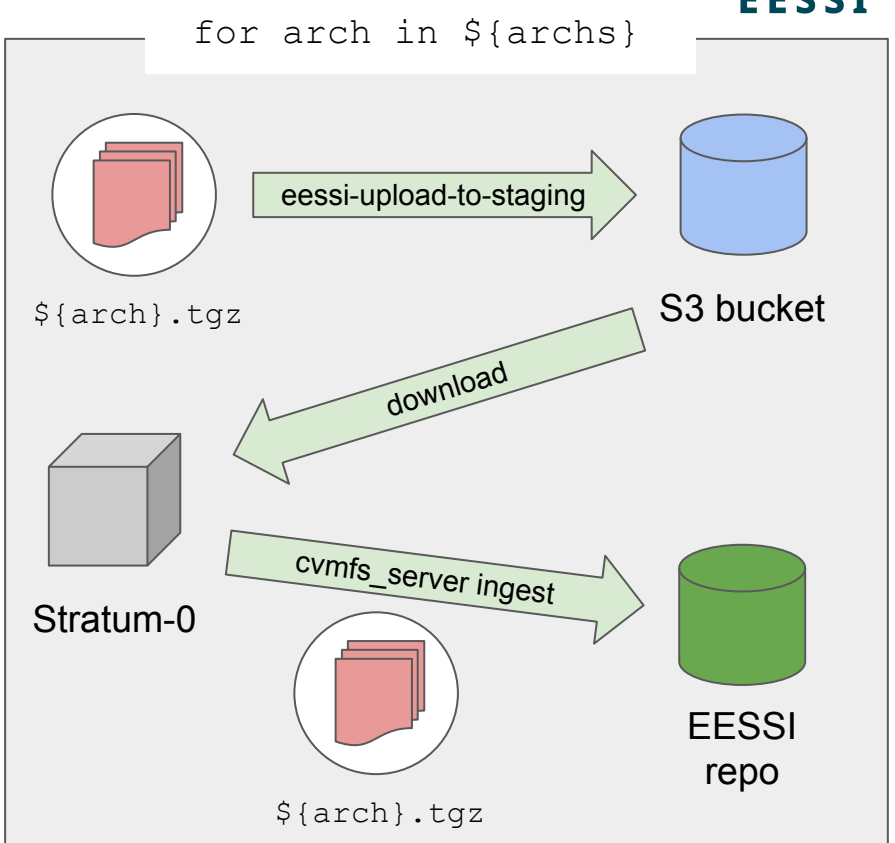
High-level overview of EESSI software bot

(simplified view)

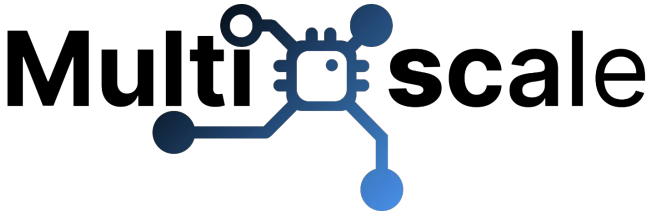


High-level overview of EESSI software bot





The MultiXscale EuroHPC Project



- EuroHPC Centre of Excellence
  - 4 year project, likely start Q1 2023
- Budget of ~6M EUR (50% EU funding, 50% national funding)
  - Roughly 50% of funding for EESSI-related activities
- Collaboration between EESSI and CECAM (total of 16 partners)
  - EESSI primarily addresses technical aspects
  - CECAM network provides scientific expertise
- Scientific target are multiscale simulations with 3 key use cases
  - Helicopter design and certification for civil transport
  - Battery applications to support the sustainable energy transition
  - Ultrasound for non-invasive diagnostics and biomedical applications
- https://www.multixscale.eu

EESSI

EUROPEAN ENVIRONMENT FOR
SCIENTIFIC SOFTWARE INSTALLATIONS

DEMO

Demo scenarios

https://github.com/EESSI/eessi-demo

- Demo 1: Using an “empty” **Ubuntu 22.04** VM in AWS (**Arm Graviton2**)
  - No CernVM-FS installed, EESSI not available yet, but only takes 2 min.
  - Requires admin rights (`sudo` to install extra packages)
  - Set up EESSI environment by sourcing init script
  - Running EESSI demo scripts

- Demo 2: On HPC-UGent infrastructure (**RHEL 8.6, AMD Rome**)
  - EESSI CernVM-FS repository readily available (by the friendly HPC-UGent sysadmins)
  - Leverage software installations provided by EESSI in job scripts
  - Anyone who has an account on the HPC-UGent infrastructure can do this!

Demo 1: Ubuntu 22.04 Arm VM in AWS (1/3)

- We need to: https://github.com/EESSI/eessi-demo
  - Install CernVM-FS packages
  - Install EESSI CernVM-FS configuration (`cvmfs-eessi-config*` package)
  - Set up minimal client configuration in `/etc/cvmfs/default.local`
- For production usage (especially large-scale), you should also:
  - Use a squid proxy, next to a local client cache (better start-up performance)
  - Set up your own Stratum-1 mirror server (protection against network disconnects)
  - Also recommended to “be a good citizen” in the EESSI CernVM-FS network

Demo 1: Ubuntu 22.04 Arm VM in AWS (2/3)

- Commands to install CernVM-FS + EESSI configuration for CernVM-FS
- Assumption: using Ubuntu as OS (only matters for `apt-get/dpkg` commands)

```
$ cat eessi-demo/scripts/install_cvmfs_eessi_Ubuntu.sh
sudo apt-get install lsb-release
wget https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest_all.deb
sudo dpkg -i cvmfs-release-latest_all.deb
sudo apt-get update
sudo apt-get install -y cvmfs

wget https://github.com/EESSI/filesystem-layer/releases/download/latest/cvmfs-config-eessi_latest_all.deb
sudo dpkg -i cvmfs-config-eessi_latest_all.deb

sudo bash -c "echo 'CVMFS_CLIENT_PROFILE="single"' > /etc/cvmfs/default.local"
sudo bash -c "echo 'CVMFS_QUOTA_LIMIT=10000' >> /etc/cvmfs/default.local"

sudo cvmfs_config setup
```

https://github.com/EESSI/eessi-demo

Demo 1: Ubuntu 22.04 Arm VM in AWS (3/3)

- Once CernVM-FS + EESSI configuration is installed, you're good to go!
- Set up EESSI environment by sourcing the init script, load modules, run.

https://github.com/EESSI/eessi-demo

```
$ ls /cvmfs/pilot.eessi-hpc.org
host_injections latest versions

$ source /cvmfs/pilot.eessi-hpc.org/latest/init/bash
...
Environment set up to use EESSI pilot software stack, have fun!

$ module avail GROMACS TensorFlow OpenFOAM Bioconductor

------ /cvmfs/pilot.eessi-hpc.org/versions/2021.12/software/linux/aarch64/graviton2/modules/all -------

      GROMACS/2020.1-foss-2020a-Python-3.8.2       OpenFOAM/9-foss-2021a                     (D)
      GROMACS/2020.4-foss-2020a-Python-3.8.2 (D)   R-bundle-Bioconductor/3.11-foss-2020a-R-4.0.0
      OpenFOAM/v2006-foss-2020a                    TensorFlow/2.3.1-foss-2020a-Python-3.8.2
      OpenFOAM/8-foss-2020a
```

Demo 2: On HPC-UGent infrastructure

- https://www.ugent.be/hpc/en/infrastructure
- OS: RHEL 8.6 - Slurm
- CPUs: mix of different generations of Intel and AMD CPUs
- Assumption: EESSI is already available to use
- HPC team has installed and configured CernVM-FS to provide access to EESSI
- Incl. properly setting up squid proxy (cache) + local Stratum-1 (caching + reliability)
- Researchers who have an HPC account can leverage software provided by EESSI
- **Just source EESSI init script, load modules, and you're ready to go!**

```
source /cvmfs/pilot.eessi-hpc.org/latest/init/bash
```

Try out EESSI yourself using Apptainer!

- Only Apptainer (or Singularity) is required to run the EESSI client container
- **Should work on any Linux distribution**, on Intel/AMD/Arm/POWER CPUs
- Detailed instructions available at https://eessi.github.io/docs/pilot

```
$ apptainer shell --fusemount "$EESSI_PILOT" docker://ghcr.io/eessi/client-pilot:centos7
...
Apptainer> ls /cvmfs/pilot.eessi-hpc.org/
2021.06  host_injections  latest  versions

Apptainer> source /cvmfs/pilot.eessi-hpc.org/latest/init/bash

Found EESSI pilot repo @ /cvmfs/pilot.eessi-hpc.org/versions/2021.12!
archspec says x86_64/amd/zen2
Using x86_64/amd/zen2 as software subdirectory.
Using /cvmfs/pilot.eessi-hpc.org/versions/2021.12/software/linux/x86_64/amd/zen2/modules/all as the
directory to be added to MODULEPATH.
Found Lmod configuration file at
/cvmfs/pilot.eessi-hpc.org/versions/2021.12/software/linux/x86_64/amd/zen2/.lmod/lmodrc.lua
Initializing Lmod...
Prepending /cvmfs/pilot.eessi-hpc.org/versions/2021.12/software/linux/x86_64/amd/zen2/modules/all to
$MODULEPATH...
```

Overview of use cases enabled by EESSI

- A uniform software stack across HPC clusters, clouds, servers, and laptops
- Can be leveraged in continuous integration (CI) environments
- Significantly facilitates setting up infrastructure for HPC training
- Enhanced collaboration with software developers and application experts
- Enable portable workflows

Also discussed in our open-access paper, available via [doi.org/10.1002/spe.3075](doi.org/10.1002/spe.3075)

EESSI provides a uniform software stack

- Main goal: **same software everywhere**: laptop, server, HPC, cloud, …
- Wide variety of systems supported
  - CPUs: x86_64 (Intel, AMD), aarch64 (Arm), ppc64le (POWER), riscv64 (soon…)
  - OS: any Linux distribution, Windows via WSL, macOS should be possible too
  - High-speed interconnects (Infiniband), GPUs, etc.
- Without compromising on software performance
  - Optimized software installations for specific CPU microarchitectures + auto-detection
  - Large contrast with generic binaries often used in containers
- Facilitates migrating from laptop to HPC, cloud bursting, …

Leveraging EESSI in CI environments

- EESSI can be used in CI environments like Jenkins, GitHub Actions, …
- We can provide:
  - Different compilers to test your software with
  - Required dependencies for your software
  - Additional tools like ReFrame, …
- Other than CernVM-FS, no software installations required
  - Everything that is actually needed is pulled in on-demand by CernVM-FS
- Significantly facilitates also running CI tests in other contexts (laptop, HPC, …)

Leveraging EESSI in CI environments

Accessing EESSI in a GitHub Actions workflow is very… easy:

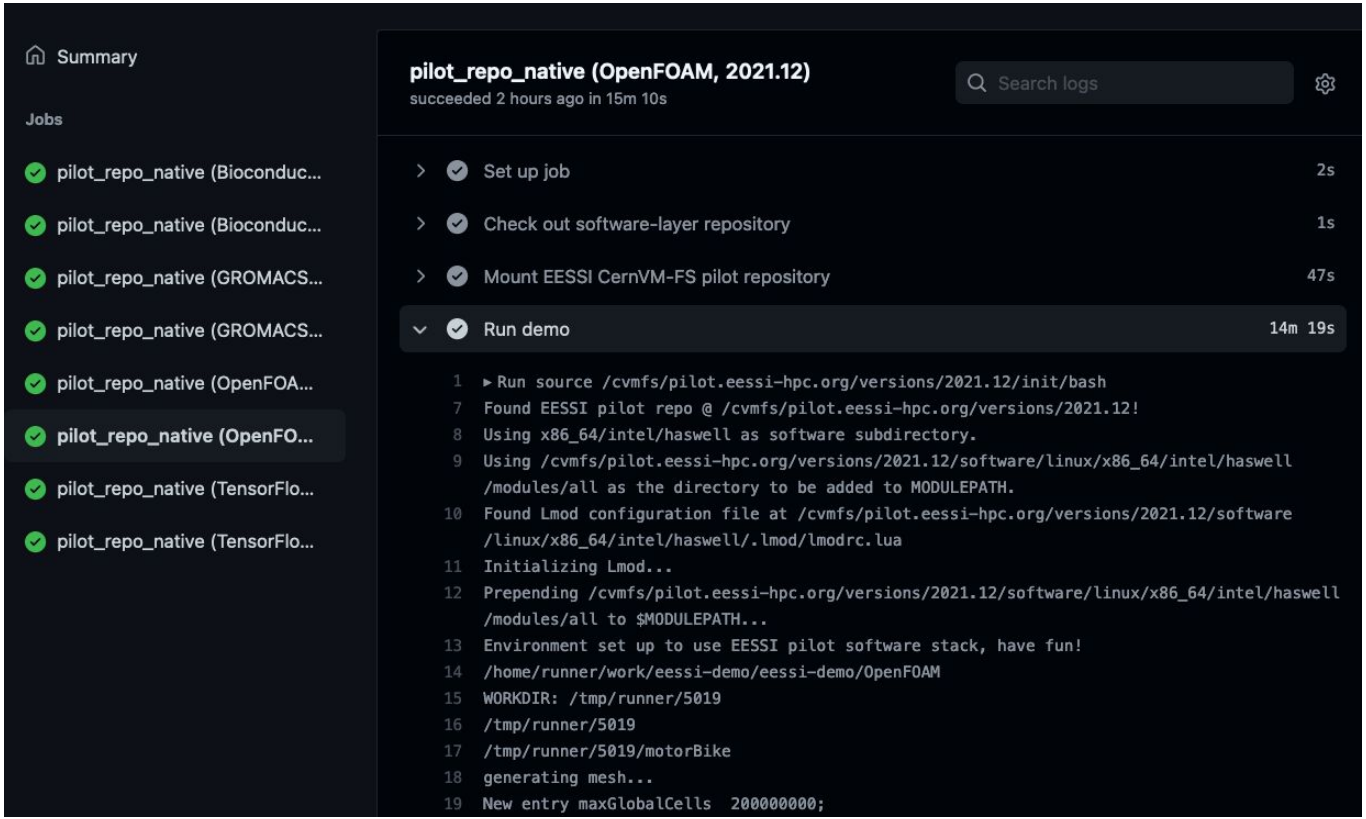
See it in action in the `eessi-demo` repository:

github.com/EESSI/eessi-demo/actions/workflows/pilot_repo_native.yml

github.com/EESSI/eessi-demo/blob/main/.github/workflows/pilot_repo_native.yml

```yaml
jobs:
  eessi:
    runs-on: ubuntu-20.04
    steps:
      - name: Check out repository
        uses: actions/checkout@v2
      - name: Mount EESSI CernVM-FS pilot repository
        uses: cvmfs-contrib/github-action-cvmfs@main
        with:
                # name of EESSI pilot repository
                cvmfs_repositories: pilot.eessi-hpc.org
                # EESSI configuration package (long download URL)
                cvmfs_config_package: https://…/latest/cvmfs-config-eessi_latest_all.deb
                # direct access to CernVM-FS network, no proxy
                cvmfs_http_proxy: DIRECT
      - name: Set up EESSI environment and run tests
        run: |
         source /cvmfs/pilot.eessi-hpc.org/versions/2021.12/init/bash
        ./run_tests.sh  # what the developer really cares about, just load modules for dependencies!
```

Leveraging EESSI in CI environments



https://github.com/EESSI/eessi-demo/actions/runs/3044103853/jobs/4904114694

Leveraging EESSI in CI environment (short version)

We also have an EESSI GitHub Action as a shorthand for this:

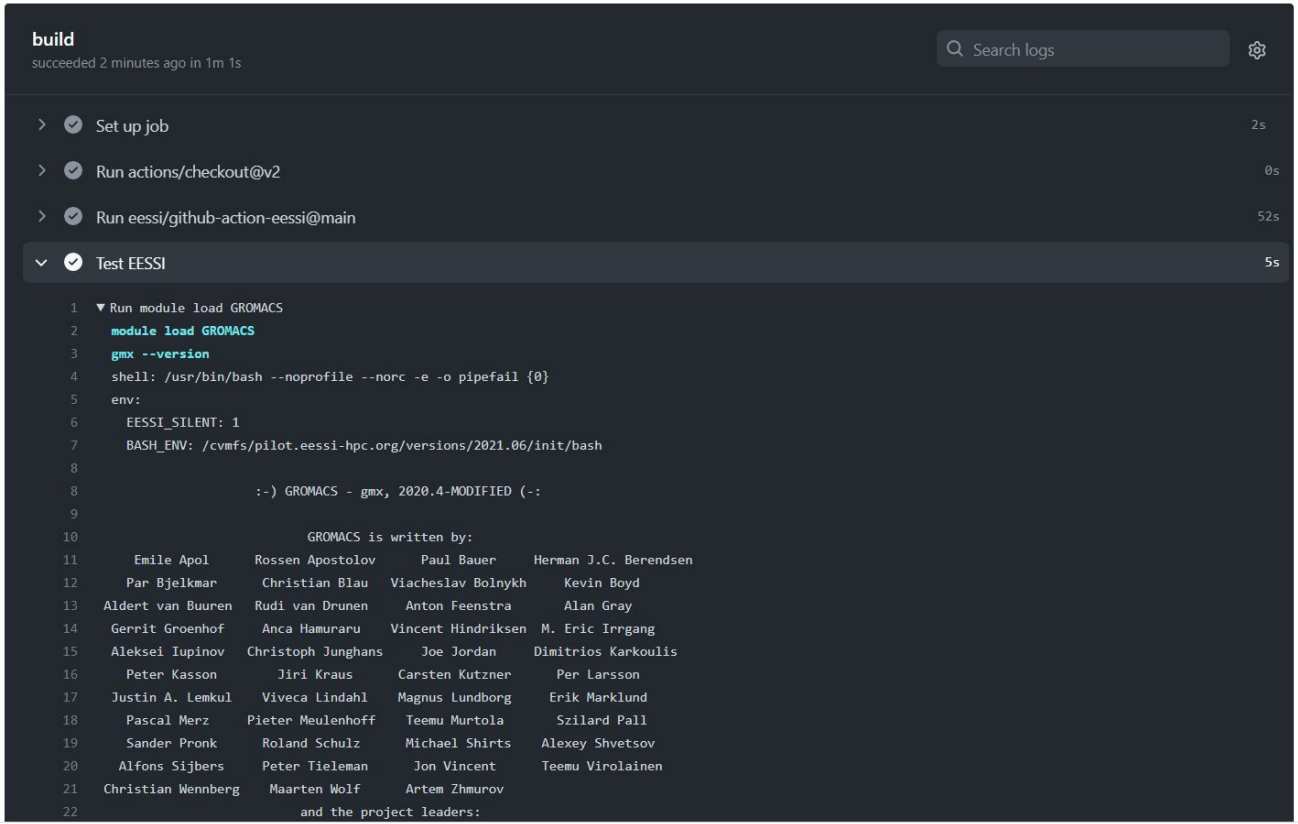
See it in action in the `github-essi-action` repository:

github.com/EESSI/github-action-eessi

github.com/EESSI/github-action-eessi/blob/main/.github/workflows/gromacs-usage.yml

```yaml
name: ubuntu_gromacs
on: [push, pull_request]
jobs:
 build:
   runs-on: ubuntu-latest
   steps:
   - uses: actions/checkout@v2
   - uses: eessi/github-action-eessi@main
     with:
       eessi_stack_version: '2021.06'
   - name: Test EESSI
     run: |
       module load GROMACS
       gmx --version
     shell: bash
```

Leveraging EESSI GitHub Action



https://github.com/EESSI/github-action-eessi/actions/runs/3044539257/jobs/4905040409

Facilitate HPC training

- EESSI can significantly reduce effort required to set up infrastructure for HPC training sessions (introductory, software-specific, …)
- Setting up a throwaway Slurm cluster in the cloud is easy via CitC or Magic Castle
- EESSI can provide (scientific) software that is required for the training
- Attendees can easily set up the *same* software environment later on their own system(s) by leveraging EESSI

Collaboration with software developers + experts

- A central software stack by/for the community opens new doors…
- We can work with software developers/experts to verify the installation
  - Check how installation is configured and built
  - Help to verify whether software is functional for different use cases
  - Show us how to do extensive testing of their software
  - Evaluate performance of the software, enable performance monitoring
  - **“Approved by developers” stamp for major applications included in EESSI**
- Relieve software developers from burden of getting their software installed
  - Remove need to provide pre-built binary packages?
- Developers can also leverage EESSI themselves: dependencies, CI, …

EESSI enables portable workflows

- Portable workflows are significantly easier when relying on EESSI
- They often involve running a broad set of tools, which all need to be available
- Workflows definitions (Snakemake, …) can be included in EESSI along with software
- Community-specific view on software provided by EESSI can be provided

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

Website: https://www.eessi-hpc.org

Join our mailing list & Slack channel

https://www.eessi-hpc.org/join

Documentation: https://eessi.github.io/docs

GitHub: https://github.com/eessi

Twitter: @eessi_hpc

YouTube channel (brand new!)

Monthly online meetings (first Thursday, 2pm CEST)