

# Getting (your) scientific software installed using

Kenneth Hoste (HPC-UGent)

*kenneth.hoste@ugent.be*

**20190510 - CSCS Software Management Course**

*[https://users.ugent.be/~kehoste/EasyBuild\\_20190510\\_CSCS\\_software\\_management.pdf](https://users.ugent.be/~kehoste/EasyBuild_20190510_CSCS_software_management.pdf)*

*<https://easybuilders.github.io/easybuild>*

*<https://easybuild.readthedocs.io>*

# HPC-UGent



- part of central IT department of Ghent University (Belgium)
- centralised scientific computing services, training & support
- for researchers of UGent, industry & knowledge institutes
- 6 Tier-2 clusters (> 15k cores in total), ~2 PB shared storage
- 7+1 team members, > 3000 user accounts
- member of Flemish Supercomputer Centre (VSC)

<https://www.vscentrum.be>





# whoami

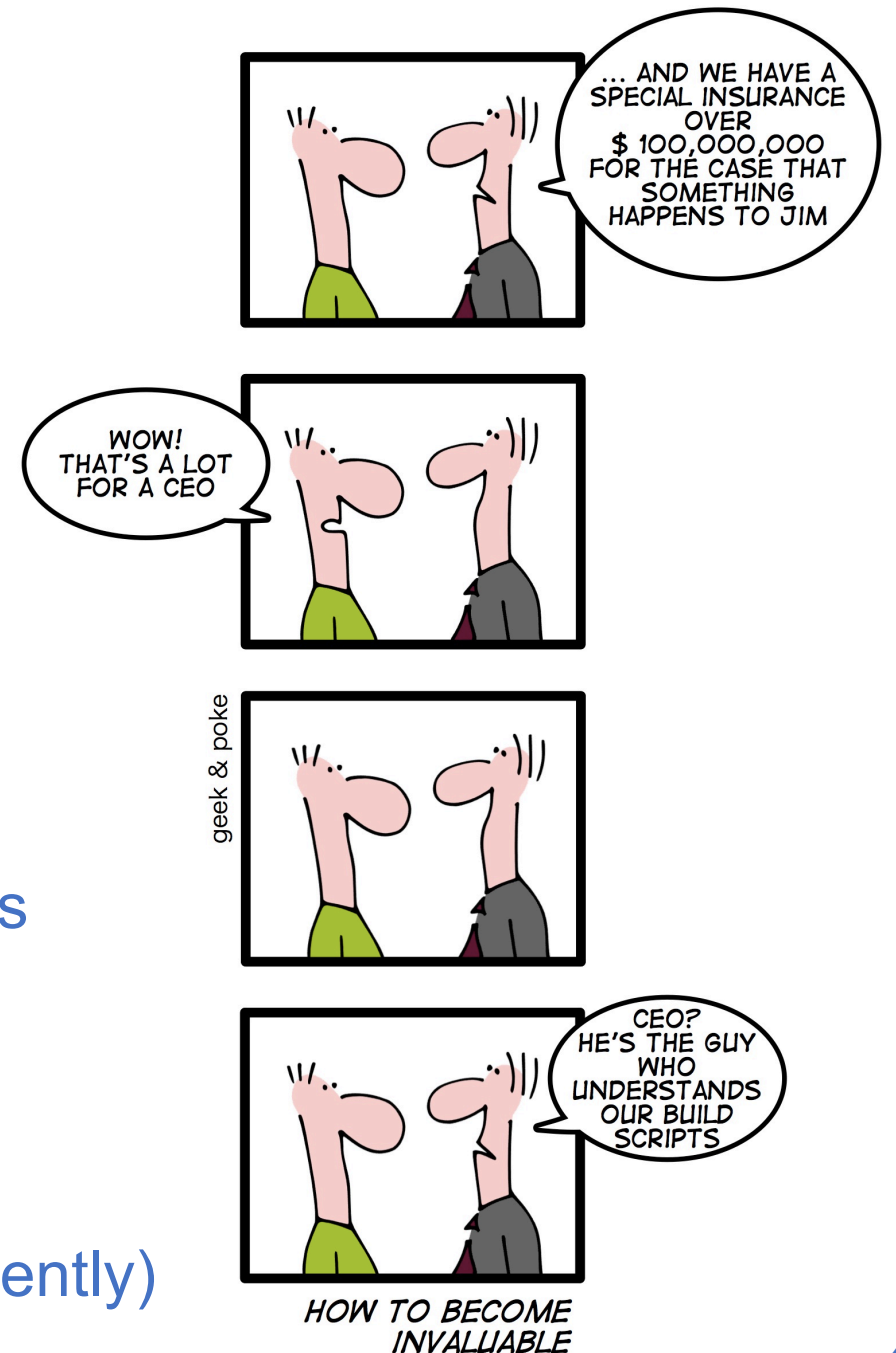
kenneth.hoste@ugent.be  
@boege1 (*GitHub, IRC, Slack*)  
@kehoste (*Twitter*)

- Masters & PhD in Computer Science from Ghent University
- PhD topic: machine learning applied to software performance, compilers, ...
- joined HPC-UGent team in October 2010
- main tasks: user support & training, *software installations*
- slowly also became  *easybuild* **lead developer & release manager**
- likes family, beer, loud music, FOSS, helping people, dad jokes, stickers, ...
- doesn't like CMake, SCons, Bazel, TensorFlow, OpenFOAM, ...

# Getting scientific software installed

Installation of scientific software is a tremendous problem for HPC sites all around the world.

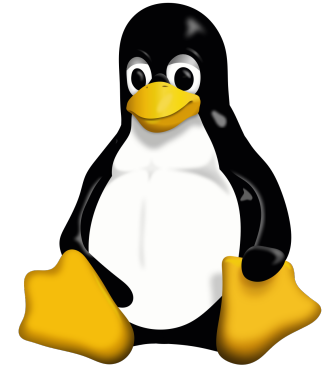
- ideally built from source (performance is key!)
- tedious, time-consuming, frustrating, sometimes simply not worth the (manual) effort...
- huge burden on researchers & HPC support teams
  - over 25% of support tickets at HPC-UGent, but consumes way more than 1/4th of support time...
- very little collaboration among HPC sites (until recently)



# What about existing software installation tools? (1/2)

- **traditional package managers in Linux(-like) operating systems**

- *yum* to install RPMs (Red Hat derivatives)
- *apt* to install .deb files (Debian & derivatives)
- *Homebrew* (macOS/Linux)
- *Portage* (Linux), *pkgsrc* (\*nix), ...



- **problems in context of scientific software & HPC:**

- not well suited to idiosyncrasies of scientific software
- not aware (enough) of MPI/BLAS/LAPACK/GPUs, other compilers (Intel, PGI, ...)
- bad fit for multi-user HPC systems in general
- little support for having multiple versions installed side-by-side
- strong focus on generically optimised binaries (portability is important to limit effort)

# What about existing software installation tools? (2/2)

 **CONDA** (installation tool for Anaconda distribution)

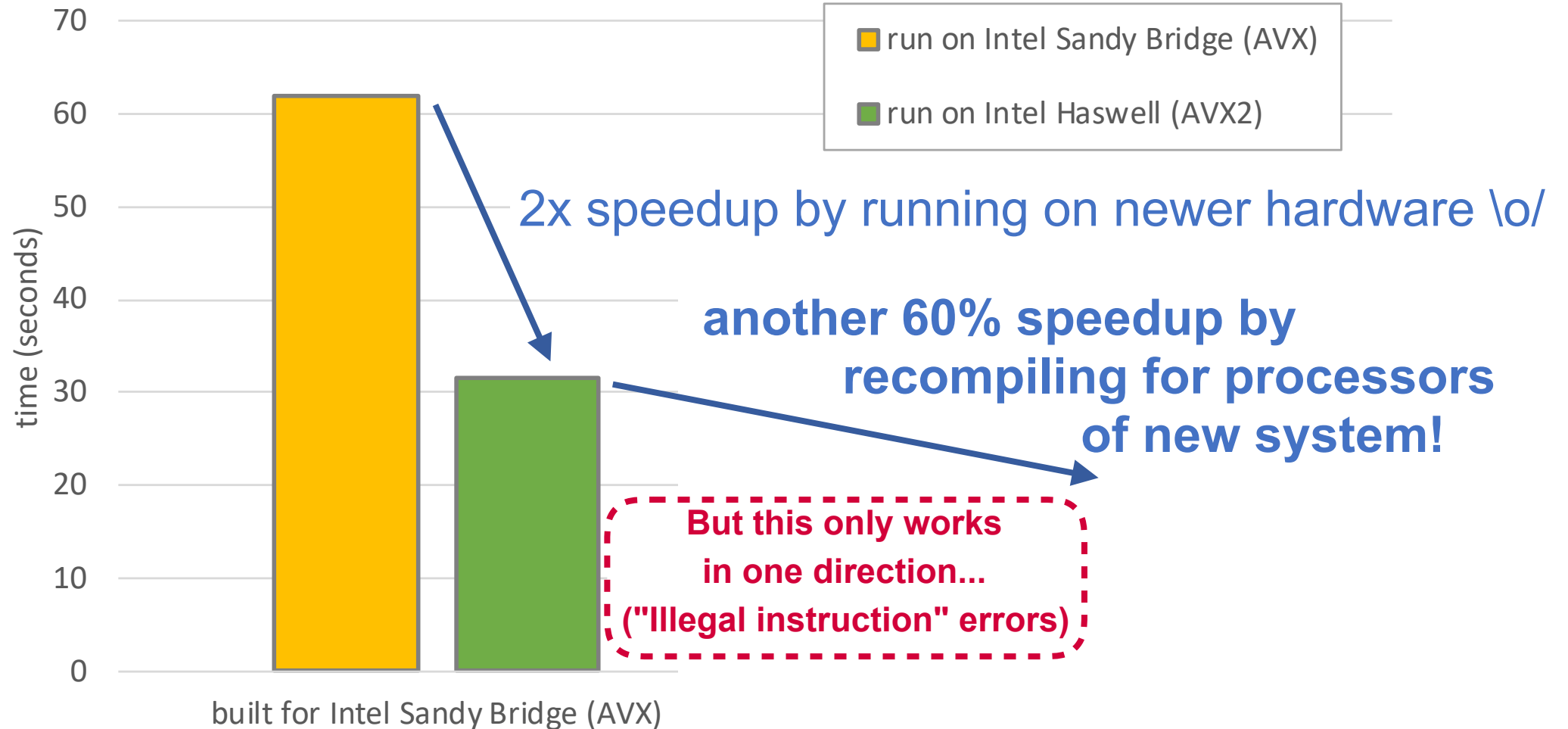
- very popular tool for installing scientific software, but mainly **targeted to individual users**
- **not well suited for centrally managing software** installations on multi-user HPC systems
- installing software with `conda` *usually* results in running **generically optimised binaries**
- hard to combine with software installed in other ways (e.g. centrally provided modules)

Containers (Docker/Singularity)  

- **lack of attention to optimising for underlying hardware** ("mobility of compute")
- **integration with system resources** (MPI, CUDA, ...) is *still a problem*
- **someone still needs to be build/maintain/update the container image you need/want !**
- in my opinion, containers are a *symptom*, not a cure...

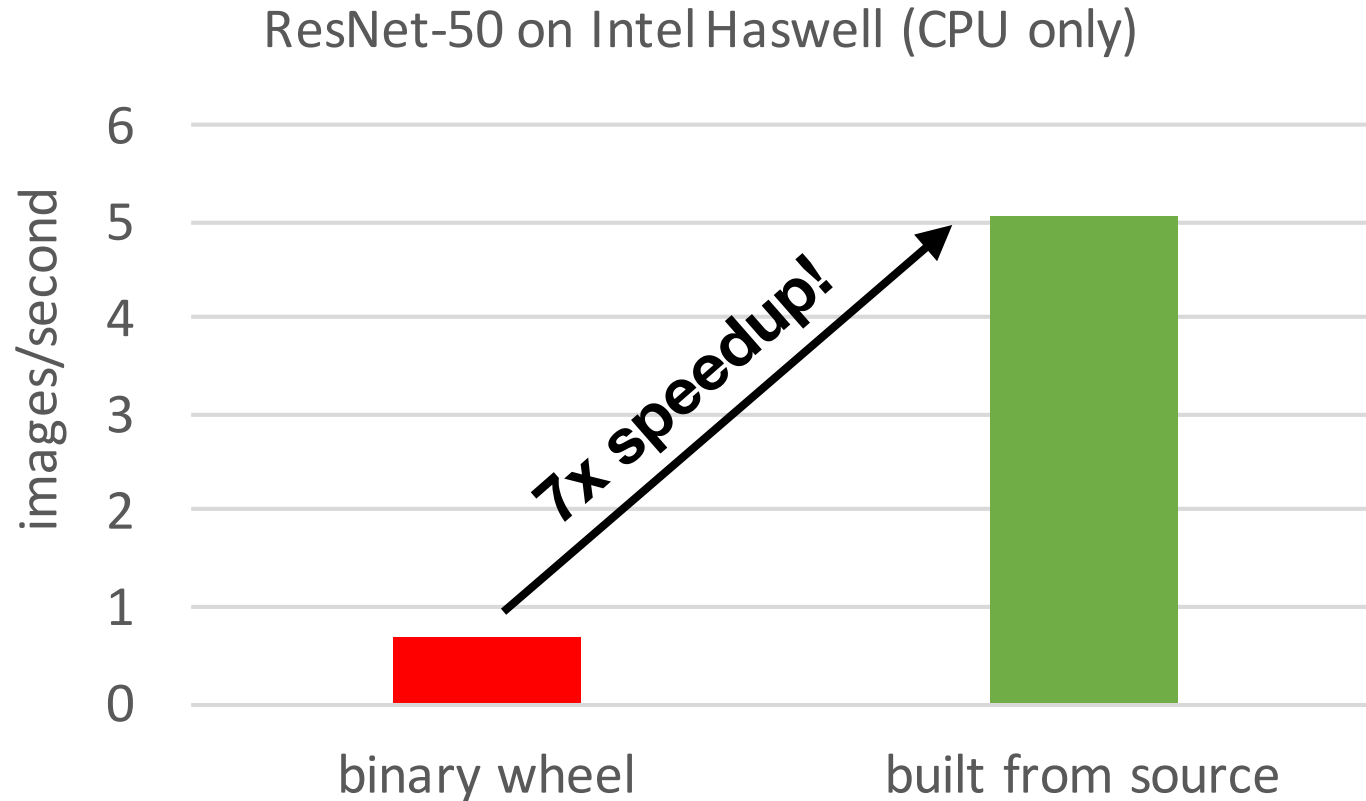
# Performance is key in HPC

'simple' benchmark (16k) for FFTW 3.3.8 (compiled with GCC 7.3)



(FFTW 3.3.8 installed in Singularity container, built from source)

# "pip install tensorflow" works fine for me...



*(disclaimer: old version of TensorFlow (1.4.x?), but point is still valid)*

- installing pre-built generically optimised binaries is easy
- ... but **you may be paying a *significant* prize w.r.t. performance**



<https://easybuilders.github.io/easybuild> - <https://easybuild.readthedocs.io>

- **framework for installing scientific software** (*built from source when possible*)
- strong focus on Linux & HPC systems (and hence also performance)
- **by default builds/optimises specifically for host architecture**
- implemented in Python 2, lead development by HPC-UGent
- available under GPLv2 license via PyPI, GitHub
- supports different compilers & MPI libraries, x86\_64/ARM/POWER, ...
- **good integration with Cray Programming Environment**
- active & helpful worldwide community

# Almost 10 years of easybuild

- in-house development at HPC-UGent since summer 2009
- first public release in April 2012 (EasyBuild v0.5)
- **first stable release on November 13th 2012, during SC'12 (EasyBuild v1.0)**
- intention was to get feedback, but gradually a community emerged around it...
- **frequent stable releases** since then (latest: EasyBuild v3.9.0, Apr 12th 2019)
- community-driven development: bug reports, feature requests, contributions



# Supported software



[http://easybuild.readthedocs.io/en/latest/version-specific/Supported\\_software.html](http://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html)

- latest EasyBuild (v3.9.0) supports installing **over 1,700 different software packages**
  - including CP2K, NAMD, NWChem, OpenFOAM, TensorFlow, WRF, ...
  - a lot of bioinformatics software is also supported out of the box
  - + >1,000 extensions: Python packages, R libraries, Perl modules, X11 libraries, ...
  - built from source when possible, optimised by host architecture by default
- diverse toolchain support:
  - compilers: GCC, Intel, Clang, PGI, IBM XL, **Cray GNU/Intel/CCE**, CUDA
  - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, **Cray MPI**, ...
  - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ScaLAPACK, BLIS, **Cray LibSci**, ...



# easybuild terminology

[http://easybuild.readthedocs.io/en/latest/Concepts\\_and\\_Terminology.html](http://easybuild.readthedocs.io/en/latest/Concepts_and_Terminology.html)

- **framework**

- core of EasyBuild: Python modules & packages
- provides supporting functionality for building/installing software, generating modules, ...

- **easyblock**

- a Python module that serves as a build script, 'plugin' for the EasyBuild framework
- implements a (generic or software-specific) build/install procedure

- **easyconfig file** (\*.eb): build specification; software name/version, compiler toolchain, etc.

- **(compiler) toolchain**: set of compilers + accompanying libraries (MPI, BLAS/LAPACK, ...)

- **extensions**: additional packages for a particular applications (e.g., Python, R)



# easybuild feature highlights (1)

- fully **autonomously** building and installing (scientific) software
  - automatic dependency resolution (via `--robot`)
  - automatic generation of environment module files (Tcl or Lua syntax)
- **no admin privileges required** (only write permission to installation target)
- thorough **logging** of executed build/install procedure
- **archiving** of easyconfigs, patches, easyblocks that were used
- highly **configurable**, via config files/environment/command line
- **dynamically extendable** with additional easyblocks, toolchains, etc.



## easybuild feature highlights (2)

- support for **custom module naming schemes** (incl. hierarchical)
- **transparency** via support for 'dry run' installation & trace output
- **comprehensively tested**: lots of unit tests, frequent regression testing, ...
- actively developed, **frequent stable releases**
- **collaboration** between various HPC sites large & small
- integration with Torque/SLURM, FPM, Singularity, Docker, ...
- worldwide **community**

# Integration with Cray Programming Environment



<https://easybuild.readthedocs.io/en/latest/Cray-support.html>

- initial (experimental) integration with Cray PE implemented April 2015
- declared stable in March 2016 (EasyBuild v2.7.0)
- custom Cray\* toolchains leverage Cray-provided modules:  
`PrgEnv/*`, `cray-libsci`, `cray-mpich`, ...
- `--optarch` controls which `craype-*` module is used in build environment
- leveraging of Cray-provided "external" modules via provided metadata
- careful handling of Cray-provided modules in build environment & toolchains
- **used by CSCS since 2015 on Piz Daint & other Cray systems**

# What easybuild is not

- EasyBuild is not **YABT** (Yet Another Build Tool)

it does *not* replace build tools like `cmake` or `make`; it wraps around them

- it is not a replacement for package managers (`yum`, `apt`, ...)

it leverages some tools & libraries provided by the OS (`glibc`, `OpenSSL`, `IB drivers`, ...)

- it is not a magic solution to all your (software installation) problems...

you will still run into compiler errors (unless somebody has already taken care of it)

# What easybuild is

- a **uniform interface** that wraps around software installation procedures
- a huge **time-saver**, by automating tedious/boring/repetitive tasks
- a way to provide a **consistent software stack** to your users
- an **expert system** for software installation on HPC systems
- a **platform for collaboration** with HPC sites worldwide
- a way to **empower *users* to self-manage their software stack** on HPC systems
- a tool that can be leveraged for **building *optimised* container images**

# Installing easybuild

<https://easybuild.readthedocs.io/en/latest/Installation.html>

**EasyBuild is already installed on Piz Daint!**

```
$ module load daint-mc EasyBuild-custom
```

<https://user.cscs.ch/computing/compilation/easybuild>

- requirements:
  - GNU/Linux system + Python 2.6 or 2.7 (Python 3 will be supported soon)
  - environment modules tool (default configuration requires Lmod)
  - a system C/C++ compiler (only really needed to install toolchains)
- installation procedure (pick one):
  - `pip install easybuild` (or equivalent thereof using other Python installation tool)
  - bootstrap script (recommended)

```
$ python bootstrap_eb.py $EASYBUILD_PREFIX
$ module use $EASYBUILD_PREFIX/modules/all
$ module load EasyBuild
```

# Updating easybuild

<https://easybuild.readthedocs.io/en/latest/Installation.html#updating-an-existing-easybuild-installation>

- new versions of EasyBuild 3.x are a drop-in replacement (for older EasyBuild 3.x)
  - upcoming EasyBuild 4.0 will have some minor backwards-incompatible changes
- recommend way to pick up new EasyBuild version & start using it:

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

```
kehoste@daint103:~> module load daint-gpu EasyBuild-custom
```

```
kehoste@daint103:~> eb --version
```

```
This is EasyBuild 3.8.1 (framework: 3.8.1, easyblocks: 3.8.1) on host daint103.
```

```
kehoste@daint103:~> eb --install-latest-eb-release
```

```
kehoste@daint103:~> module load EasyBuild/3.9.0
```

```
kehoste@daint103:~> eb --version
```

```
This is EasyBuild 3.9.0 (framework: 3.9.0, easyblocks: 3.9.0) on host daint103.
```

```
kehoste@daint103:~> which eb
```

```
/users/kehoste/easybuild/daint/haswell/software/EasyBuild/3.9.0/bin/eb
```

# Configuring

<http://easybuild.readthedocs.io/en/latest/Configuration.html>

**EasyBuild is pre-configured on Piz Daint!**

```
$ module load daint-mc EasyBuild-custom
```

<https://user.cscs.ch/computing/compilation/easybuild>

- default configuration (ab)uses `$HOME/.local/easybuild`
- configuring can be done via **config files**, **environment**, or **command line interface**
  - *all* configuration options are supported on all 3 levels
  - CLI overrides environment, which overrides configuration files
- easy to extend EasyBuild:
  - (also) use own easyconfigs repositories via `--robot-paths`
  - add additional easyblocks, toolchains, module naming schemes via `--include-*`
  - [http://easybuild.readthedocs.io/en/latest/Including\\_additional\\_Python\\_modules.html](http://easybuild.readthedocs.io/en/latest/Including_additional_Python_modules.html)

# Inspecting the current configuration for



<http://easybuild.readthedocs.io/en/latest/Configuration.html>

- use 'eb --show-config' to get an overview of the current configuration
- only shows a couple of important settings + anything different from default
- output indicates via which configuration level each setting was specified
- full list of settings for current configuration via 'eb --show-full-config'

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



# Default easybuild configuration on Piz Daint

(note: actual output has been trimmed for sake of clarity)

```
$ module load daint-mc EasyBuild-custom
```

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

**User-specific '/run' directory is used for build and temporary directories.**

```
#
```

```
# Current EasyBuild configuration
```

```
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
```

```
#
```

```
buildpath (E) = /run/user/23189/easybuild/build
```

```
external-modules-metadata (E) = /apps/common/UES/jenkins/production/easybuild/cray_external_...
```

```
hide-deps (F) = absl, ANTLR, APR, APR-util, Autoconf, Automake, Autotools, ...
```

```
hide-toolchains (F) = CrayCCE, CrayGNU, CrayIntel, CrayPGI, GCCcore, ...
```

```
include-easyblocks (E) = /apps/common/UES/jenkins/production/easybuild/easyblocks/*.py
```

```
installpath (E) = /users/kehoste/easybuild/daint/broadwell
```

```
module-syntax (F) = Tcl
```

```
modules-tool (E) = EnvironmentModulesC
```

```
optarch (E) = broadwell
```

```
prefix (E) = /users/kehoste/easybuild/daint/broadwell
```

```
repositorypath (E) = /users/kehoste/easybuild/daint/broadwell/ebfiles_repo
```

```
robot-paths (E) = /apps/common/UES/jenkins/production/easybuild/easyconfigs/, ...
```

```
sourcepath (E) = /users/kehoste/sources
```

```
tmpdir (E) = /run/user/23189/easybuild/tmp
```

# Default easybuild configuration on Piz Daint

(note: actual output has been trimmed for sake of clarity)

```
$ module load daint-mc EasyBuild-custom
```

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

```
#
```

```
# Current EasyBuild configuration
```

```
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
```

```
#
```

```
buildpath (E) = /run/user/23189/easybuild/build
```

```
external-modules-metadata (E) = /apps/common/UES/jenkins/production/easybuild/cray_external_...
```

```
hide-deps (F) = absl, ANTLR, APR, APR-util, Autoconf, Automake, Autotools, ...
```

```
hide-toolchains (F) = CrayCCE, CrayGNU, CrayIntel, CrayPGI, GCCcore, ...
```

```
include-easyblocks (E) = /apps/common/UES/jenkins/production/easybuild/easyblocks/*.py
```

```
installpath (E) = /users/kehoste/easybuild/daint/broadwell
```

```
module-syntax (F) = Tcl
```

```
modules-tool (E) = EnvironmentModulesC
```

```
optarch (E) = broadwell
```

```
prefix (E) = /users/kehoste/easybuild/daint/broadwell
```

```
repositorypath (E) = /users/kehoste/easybuild/daint/broadwell/ebfiles_repo
```

```
robot-paths (E) = /apps/common/UES/jenkins/production/easybuild/easyconfigs/, ...
```

```
sourcepath (E) = /users/kehoste/sources
```

```
tmpdir (E) = /run/user/23189/easybuild/tmp
```

Installation prefix for software & modules is set to `$HOME/easybuild/daint/{broadwell,haswell}`.

Sources are downloaded to `$HOME/sources`.

# Default easybuild configuration on Piz Daint

(note: actual output has been trimmed for sake of clarity)

```
$ module load daint-mc EasyBuild-custom
```

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

```
#
```

```
# Current EasyBuild configuration
```

```
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
```

```
#
```

```
buildpath (E) = /run/user/23189/easybuild/build
```

```
external-modules-metadata (E) = /apps/common/UES/jenkins/production/easybuild/cray_external_...
```

```
hide-deps (F) = absl, ANTLR, APR, APR-util, Autoconf, Automake, Autotools, ...
```

```
hide-toolchains (F) = CrayCCE, CrayGNU, CrayIntel, CrayPGI, GCCcore, ...
```

```
include-easyblocks (E) = /apps/common/UES/jenkins/production/easybuild/easyblocks/*.py
```

```
installpath (E) = /users/kehoste/easybuild/daint/broadwell
```

```
module-syntax (F) = Tcl
```

```
modules-tool (E) = EnvironmentModulesC
```

```
optarch (E) = broadwell
```

```
prefix (E) = /users/kehoste/easybuild/daint/broadwell
```

```
repositorypath (E) = /users/kehoste/easybuild/daint/broadwell/ebfiles_repo
```

```
robot-paths (E) = /apps/common/UES/jenkins/production/easybuild/easyconfigs/, ...
```

```
sourcepath (E) = /users/kehoste/sources
```

```
tmpdir (E) = /run/user/23189/easybuild/tmp
```

## Custom configuration for Piz Daint (1/2):

- metadata for Cray-provided external modules

- Tcl-based environment modules tool (Lmod is default)

- target optimisation architecture ('broadwell' or 'haswell')

# Default easybuild configuration on Piz Daint

(note: actual output has been trimmed for sake of clarity)

```
$ module load daint-mc EasyBuild-custom
```

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

```
#
```

```
# Current EasyBuild configuration
```

```
# (C: command line argument, D: default value, E: environment variable, F: configuration file)
```

```
#
```

```
buildpath (E) = /run/user/23189/easybuild/build
```

```
external-modules-metadata (E) = /apps/common/UES/jenkins/production/easybuild/cray_external_...
```

```
hide-deps (F) = absl, ANTLR, APR, APR-util, Autoconf, Automake, Autotools, ...
```

```
hide-toolchains (F) = CrayCCE, CrayGNU, CrayIntel, CrayPGI, GCCcore, ...
```

```
include-easyblocks (E) = /apps/common/UES/jenkins/production/easybuild/easyblocks/*.py
```

```
installpath (E) = /users/kehoste/easybuild/daint/broadwell
```

```
module-syntax (F) = Tcl
```

```
modules-tool (E) = EnvironmentModulesC
```

```
optarch (E) = broadwell
```

```
prefix (E) = /users/kehoste/easybuild/daint/broadwell
```

```
repositorypath (E) = /users/kehoste/easybuild/daint/broadwell/ebfiles_repo
```

```
robot-paths (E) = /apps/common/UES/jenkins/production/easybuild/easyconfigs/, ...
```

```
sourcepath (E) = /users/kehoste/sources
```

```
tmpdir (E) = /run/user/23189/easybuild/tmp
```

## Custom configuration for Piz Daint (2/2):

- hiding dependencies & toolchain (in module avail)

- customised easyblocks maintained by CSCS

- additional easyconfig files (using Cray\* toolchains)

# Basic usage



[http://easybuild.readthedocs.io/en/latest/Using\\_the\\_EasyBuild\\_command\\_line.html](http://easybuild.readthedocs.io/en/latest/Using_the_EasyBuild_command_line.html)

[http://easybuild.readthedocs.io/en/latest/Typical\\_workflow\\_example\\_with\\_WRF.html](http://easybuild.readthedocs.io/en/latest/Typical_workflow_example_with_WRF.html)

- specify software name/version and toolchain to 'eb' command
- usually done via easyconfig filename(s):

```
eb GCC-7.3.0-2.30.eb Clang-7.01-7.3.0-2.30.eb
```

- check whether required toolchain & dependencies are available using `--dry-run/-D`:

```
eb Python-3.7.0-foss-2018b.eb -D
```

- enable dependency resolution via `--robot/-r`:

```
eb TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb -r
```

# Installing TensorFlow from source with one command...

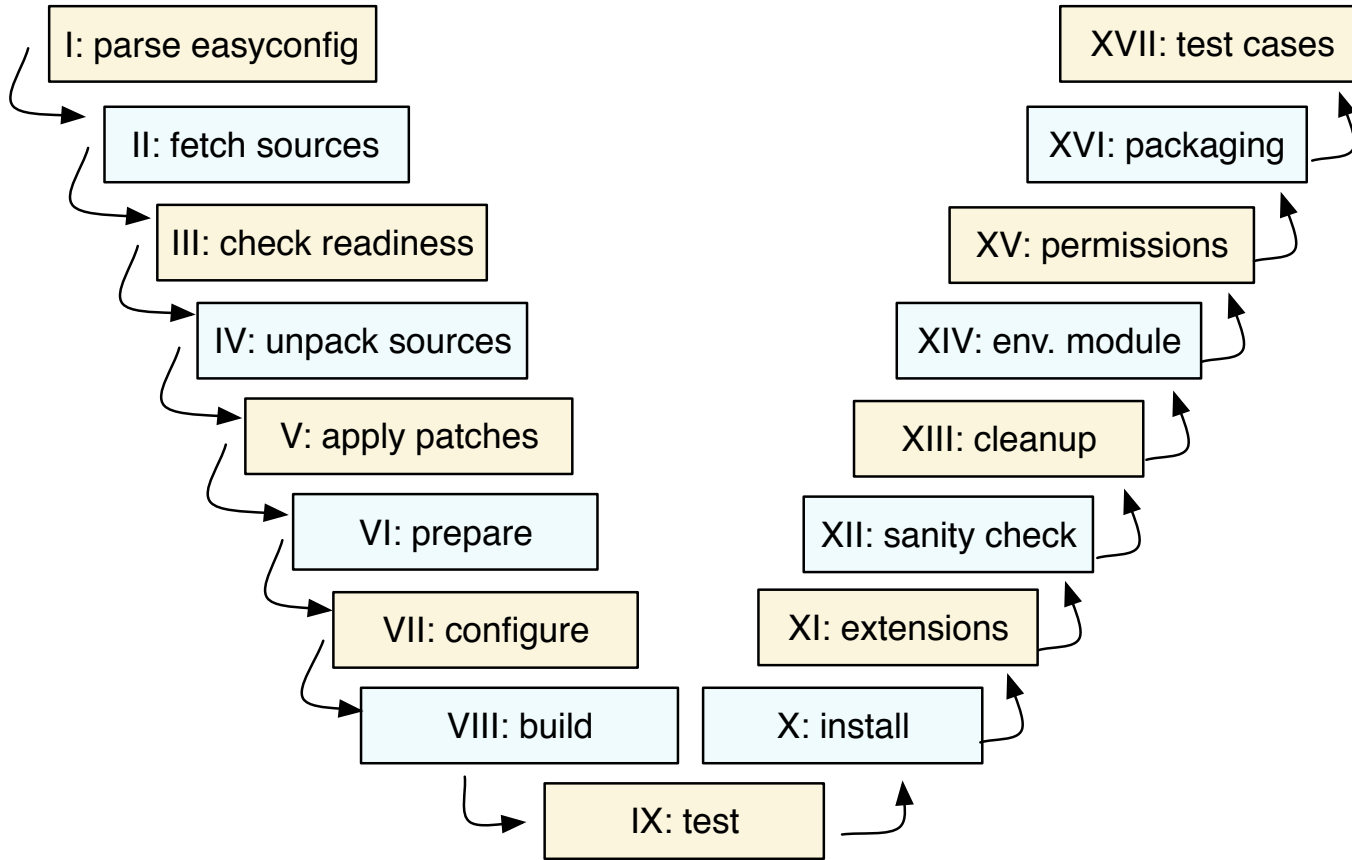


```
$ eb TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-UlTkeI.log
== processing EasyBuild easyconfig TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb
== building and installing TensorFlow/1.13.1-foss-2018b-Python-3.6.6...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
== cleaning up...
== creating module...
== permissions...
== packaging...
== COMPLETED: Installation ended successfully
== Results of the build can be found in the log file /opt/easybuild/software/Tensor...
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-GyvPHx/easybuild-UlTkeI.log* have been removed.
== Temporary directory /tmp/eb-GyvPHx has been removed.
```

# Step-wise installation procedure



EasyBuild performs a step-wise installation procedure for each software:



- download sources (best effort)
- set up build directory & environment
  - unpack sources (& apply patches)
  - load modules for toolchain & deps
  - define toolchain-related env vars (\$CC, \$CFLAGS, ...)
- configure, build, (test), install, (extensions)
- perform simple sanity check on installation
- generate environment module file

*each step can be customised via easyconfig parameters or an easyblock*

# Transparency of performed install procedure (1)



[https://easybuild.readthedocs.io/en/latest/Tracing\\_progress.html](https://easybuild.readthedocs.io/en/latest/Tracing_progress.html)

**eb --trace** shows more details while EasyBuild is performing installation(s)

- exact location of build and install directories
- list of source files & patches
- modules being loaded
- commands being executed
- pointer to temporary log file with output of commands
- timing information for each command (start time + how long it took)
- results of sanity check

# Example output of `eb --trace`



```
$ eb TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb --trace
...
== preparing...
  >> loading toolchain module: foss/2018b
  >> loading modules for build dependencies:
  >> * Bazel/0.20.0-GCCcore-7.3.0
  >> * protobuf/3.6.1-GCCcore-7.3.0
  >> loading modules for (runtime) dependencies:
  >> * Python/3.6.6-foss-2018b
  >> * wheel/0.31.1-foss-2018b-Python-3.6.6
  >> * h5py/2.8.0-foss-2018b-Python-3.6.6
  >> defining build environment for foss/2018b toolchain
...
== installing extension TensorFlow 1.13.1 (13/13)...
...
  >> running command:
      [started at: 2019-05-12 14:12:38]
      [output logged in /tmp/eb-pRHwkc/easybuild-run_cmd-SOINRV.log]
      bazel ... --jobs=6 --config=mkl //tensorflow/tools/pip_package:build_pip_package
  >> command completed: exit 0, ran in 00h41m22s
...
```

# Transparency of performed install procedure (2)



[https://easybuild.readthedocs.io/en/latest/Extended\\_dry\\_run.html](https://easybuild.readthedocs.io/en/latest/Extended_dry_run.html)

- **eb --extended-dry-run** (or '**eb -x**') reveals planned installation procedure
- runs in a matter of seconds
- shows commands that will be executed, build environment, generated module file, ...
- any errors that occur in used easyblock are ignored (but clearly reported)
- not 100% accurate since easyblock may require certain files to be present, etc.
- very useful when debugging easyblocks, instant feedback as a first pass
- implementation motivated by requests from the community
- helps to avoid impression that EasyBuild is a magic black box for installing software

# Example output of `--extended-dry-run` (1/3)



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
```

```
== temporary log file in case of crash /tmp/eb-Dh1wOp/easybuild-0bu9u9.log
```

```
== processing EasyBuild easyconfig /home/example/eb/easybuild-easyconfigs/easybuild/easyconfigs/w/WRF/WRF-3.8.0-intel-2016b-dmpar.eb
```

```
...
```

```
*** DRY RUN using 'EB_WRF' easyblock (easybuild.easyblocks.wrf @ /home/example/eb/easybuild-easyblocks/easybuild/easyblocks/w/wrf.py) ***
```

```
== building and installing WRF/3.8.0-intel-2016b-dmpar...  
fetching files... [DRY RUN]
```

```
[fetch_step method]
```

```
Available download URLs for sources/patches:
```

- \* [http://www2.mmm.ucar.edu/wrf/src//\\$source](http://www2.mmm.ucar.edu/wrf/src//$source)
- \* [http://www.mmm.ucar.edu/wrf/src//\\$source](http://www.mmm.ucar.edu/wrf/src//$source)

```
List of sources:
```

- \* WRFV3.8.0.TAR.gz will be downloaded to /home/example/eb/sources/w/WRF/WRFV3.8.0.TAR.gz

# Example output of `--extended-dry-run` (2/3)



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
...

building... [DRY RUN]

[build_step method]
  running command "tcsch ./compile -j 4 wrf"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsch ./compile -j 4 em_real"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsch ./compile -j 4 em_b_wave"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
...

[sanity_check_step method]
Sanity check paths - file ['files']
  * WRFV3/main/libwrflib.a
  * WRFV3/main/real.exe
  * WRFV3/main/wrf.exe
Sanity check paths - (non-empty) directory ['dirs']
  * WRFV3/main
  * WRFV3/run
Sanity check commands
  (none)
```

# Example output of `--extended-dry-run (3/3)`



```
$ eb WRF-3.8.0-intel-2016b-dmpar.eb -x
```

```
...
```

```
[make_module_step method]
```

```
Generating module file /home/example/eb/modules/all/WRF/3.8.0-intel-2016b-dmpar,  
with contents:
```

```
#!/Module  
proc ModulesHelp { } {  
    puts stderr { The Weather Research and Forecasting (WRF) Model }  
}  
module-whatis {Description: WRF - Homepage: http://www.wrf-model.org}  
  
set root /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar  
  
conflict WRF  
  
if { ![ is-loaded intel/2016b ] } {  
    module load intel/2016b  
}  
if { ![ is-loaded Jasper/1.900.1-intel-2016b ] } {  
    module load Jasper/1.900.1-intel-2016b  
}
```

# Log files



<http://easybuild.readthedocs.io/en/latest/Logfiles.html>

- **EasyBuild *thoroughly* logs the executed installation procedure**
  - active EasyBuild configuration
  - easyconfig file that was used
  - modules that were loaded + resulting changes to environment
  - defined environment variables
  - output + exit code of executed commands
  - informative log messages produced by easyblock
- log file is copied to software installation directory for future reference
- can be used to debug build problems or see how installation was performed exactly

# Log files: example



```
== 2019-05-13 13:34:31,906 main.EB_HPL INFO This is EasyBuild 3.9.0 (framework:
3.9.0, easyblocks: 3.9.0) on host example.
...
== 2019-05-13 13:34:35,503 main.EB_HPL INFO configuring...
== 2019-05-13 13:34:48,817 main.EB_HPL INFO Starting configure step
...
== 2019-05-13 13:34:48,823 main.EB_HPL INFO Running method configure_step part of
step configure
...
== 2019-05-13 13:34:48,823 main.run DEBUG run_cmd: running cmd /bin/bash
make_generic (in /tmp/easybuild_build/HPL/2.3/foss-2019a/hpl-2.3/setup)
== 2019-05-13 13:34:48,823 main.run DEBUG run_cmd: Command output will be logged
to /tmp/easybuild-W85p4r/easybuild-run_cmd-XoJwMY.log
== 2019-05-13 13:34:48,849 main.run INFO cmd "/bin/bash make_generic" exited with
exitcode 0 and output:
...
```

# Easyconfig files as build specifications



[http://easybuild.readthedocs.io/en/latest/Writing\\_easyconfig\\_files.html](http://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html)

- for each software installation, there is a corresponding easyconfig file
- **simple text files defining a set of easyconfig parameters** (in Python syntax)
- some are mandatory: software name/version, toolchain, metadata (homepage, descr.)
- other commonly used parameters:
  - easyblock to use
  - list of sources & patches
  - list of (build) dependencies
  - options for configure/build/install commands
  - files/directories that should be present, trivial commands that should work (sanity check)

## Overview of supported easyconfig parameters:

```
$ eb -a
```

**Include easyconfig parameters specific to a particular a particular easyblock:**

```
$ eb -a -e PythonPackage
```

# Example easyconfig file



no easyblock specified, which implies using a software-specific easyblock (EB\_WRF)

software name and version	←	<code>name = 'WRF'</code> <code>version = '3.8.0'</code>
build variant (specific to WRF) ( <code>'dmpar'</code> : distributed, MPI)	←	<code>buildtype = 'dmpar' # custom parameter for WRF</code> <code>versionsuffix = '-' + buildtype # part of module name</code>
software metadata	←	<code>homepage = 'http://www.wrf-model.org'</code> <code>description = "Weather Research and Forecasting (WRF) Model"</code>
toolchain name & version	←	<code>toolchain = {'name': 'intel', 'version': '2016b'}</code>
sources & patches	←	<code>source_urls = ['http://www.mmm.ucar.edu/wrf/src/']</code> <code>sources = ['%(name)sV%(version_major_minor)s.TAR.gz']</code> <code>patches = ['WRF-%(version)s_known_problems.patch']</code>
list of (build) dependencies <b>note: all versions are <i>fixed</i>!</b>	←	<code>builddependencies = [('tcsh', '6.20.00')]</code> <code>dependencies = [</code> <code>('JasPer', '2.0.10'),</code> <code>('netCDF', '4.4.1'),</code> <code>('netCDF-Fortran', '4.4.4'),</code> <code>]</code>

# Adding support for additional software



- for each software installation, an **easyconfig file** is *required*
  - see [https://easybuild.readthedocs.io/en/latest/Writing\\_easyconfig\\_files.html](https://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html)
  - existing easyconfig files can serve as examples
  - tweaked easyconfig files can be *generated* via `eb --try-*`
- for 'standard' installation procedures, a **generic easyblock** can be used
  - installation can be controlled where needed via easyconfig parameters
- for custom installation procedures, a **software-specific easyblock** is required
  - see <https://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html>

# Tweaking existing easyconfigs via `--try-*`



- simple common use cases include:
  - updating to a more recent software version
  - installing with a different compiler toolchain
- dedicated CLI options: `--try-software-version`, `--try-toolchain`, ...
- success is not guaranteed... (hence the 'try')

```
kehoste@daint104:~> module load daint-mc EasyBuild-custom/cscs
kehoste@daint104:~> module avail CMake
----- /apps/daint/UES/jenkins/6.0.UP07/mc/easybuild/modules/all -----
CMake/3.12.0
kehoste@daint104:~> eb CMake-3.12.0.eb --try-software-version 3.14.3
...
kehoste@daint104:~> module avail CMake
----- /users/kehoste/easybuild/daint/broadwell/modules/all -----
CMake/3.14.3
----- /apps/daint/UES/jenkins/6.0.UP07/mc/easybuild/modules/all -----
CMake/3.12.0
```

# Writing easyconfig files



[https://easybuild.readthedocs.io/en/latest/Writing\\_easyconfig\\_files.html](https://easybuild.readthedocs.io/en/latest/Writing_easyconfig_files.html)

- set of values for easyconfig parameters, written in Python syntax  
(strictly speaking order doesn't matter, but we maintain a preferred order & style)
- strictly required:  
`name, version, toolchain, homepage, description`
- commonly used:  
`easyblock, versionsuffix, source_urls, sources,`  
`(build)dependencies, (pre)configopts, (pre)buildopts,`  
`(pre)installopts, exts_list, sanity_check_paths, moduleclass`

# Common generic easyblocks



[http://easybuild.readthedocs.io/en/latest/version-specific/generic\\_easyblocks.html](http://easybuild.readthedocs.io/en/latest/version-specific/generic_easyblocks.html)

- **ConfigureMake**  
standard './configure' - 'make' - 'make install' installation procedure
- **CMakeMake**  
same as ConfigureMake, but using 'cmake' for configuring
- **PythonPackage**  
installing an individual Python package ('python setup.py install', 'pip install', ...)
- **PythonBundle**  
installing a bundle of Python packages
- **MakeCp**  
no (standard) configuration step, build with 'make', install by copying binaries/libraries
- **Tarball**: just unpack sources and copy everything to installation directory
- **Binary**: run binary installer (specified via 'install\_cmd' easyconfig parameter)

# Community (common) toolchains



<http://easybuild.readthedocs.io/en/latest/Common-toolchains.html>

- **intel and foss<sup>1</sup> toolchains** are most commonly used in EasyBuild community
- helps to focus efforts of HPC sites using one or both of these toolchains
- updated twice a year, clear versioning scheme: <year>{a,b} (2017b, 2018a, ...)
- latest version:
  - `foss/2019a`  
binutils 2.31.1, GCC 8.2, OpenMPI 3.1.3,  
OpenBLAS 0.3.5, FFTW 3.3.8
  - `intel/2019a`  
binutils 2.31.1 + GCC 8.2 as base  
Intel compilers 2019.1.144, Intel MPI 2018.4.274, Intel MKL 2019.1.144

**On Piz Daint, Cray\* toolchains are typically used:**

`CrayCCE/18.08`

`CrayGNU/18.08`

`CrayIntel/18.08`

`CrayPGI/18.08`

# Easyconfig files vs easyblocks



<http://easybuild.readthedocs.io/en/latest/Implementing-easyblocks.html>

- thin line between using custom easyblock and 'fat' easyconfig with generic easyblock
- custom easyblocks are "do once and forget", **central solution to build peculiarities**
- reasons to consider implementing a software-specific easyblock include:
  - 'critical' values for easyconfig parameters required to make installation succeed
  - toolchain-specific aspects of the build and installation procedure (e.g., configure options)
  - interactive commands that need to be run
  - custom (configure) options for dependencies
  - having to create or adjust specific (configuration) files
  - 'hackish' usage of a generic easyblock

# Demo: an easyconfig for Hello built with CMake



```
kehoste@daint104:~> module load daint-mc EasyBuild-custom
kehoste@daint104:~> ls
hello-cmake-1.0.tar.gz
kehoste@daint104:~> vim Hello.eb
kehoste@daint104:~> eb Hello.eb --trace

...
== configuring...
== building...
== testing...
== installing...

...
== sanity checking...
  >> file 'bin/hello' found: OK
  >> running command 'hello': OK
== cleaning up...

...
== COMPLETED: Installation ended successfully

kehoste@daint104:~> module load Hello/1.0-CrayGNU-18.08
kehoste@daint104:~> hello
Hello World!
```

# Demo: an easyconfig for Hello built with CMake



```
kehoste@daint104:~> cat Hello.eb
easyblock = 'CMakeMake'

name = 'Hello'
version = '1.0'

homepage = 'https://example.com'
description = "Hello world!"

toolchain = {'name': 'CrayGNU', 'version': '18.08'}

sources = ['hello-cmake-%(version)s.tar.gz']

builddependencies = [('CMake', '3.14.3', '', True)]

sanity_check_paths = {
    'files': ['bin/hello'],
    'dirs': [],
}
sanity_check_commands = ["hello"]

moduleclass = 'tools'
```

# Contributing to EasyBuild



<http://easybuild.readthedocs.io/en/latest/Contributing.html>

EasyBuild has improved significantly thanks to the community.

**You too can contribute back, by:**

- sending feedback
- reporting bugs
- joining the discussion (mailing lists, IRC/Slack, EasyBuild conf calls)
- sharing suggestions/ideas for enhancements & additional features
- contributing easyconfigs, enhancing easyblocks, adding support for new software...
- extending & enhancing documentation

# How to contribute easyconfig files



<https://easybuild.readthedocs.io/en/latest/Contributing.html>

- open pull request (PR) to develop branch of easybuild-easyconfigs repository
- keep PRs limited in scope (rule of thumb: max. 10 easyconfigs touched per PR)
- if possible, submit test report to show the easyconfig(s) works for you
- take into account failing unit tests (if any) and comments by reviewer (if any)
- be patient...
- enjoy eternal glory once your pull request has been merged!

# GitHub integration in

[https://easybuild.readthedocs.io/en/latest/Integration\\_with\\_GitHub.html](https://easybuild.readthedocs.io/en/latest/Integration_with_GitHub.html)

- goals:

- **automate contribution workflow**
- avoid direct interaction with git and



- initial implementation added support for:

- opening new PRs & updating existing PRs
- downloading & using easyconfigs from a PR
- uploading test reports to a PR

`eb --new-pr`

`eb --update-pr`

`eb --from-pr`

`eb --upload-test-report`

- limited to easyconfig files, for now...

- but implemented such that it is easy to extend to easyblocks, and even the EasyBuild framework itself

# 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
```

# The easybuild community



group picture  
4th EasyBuild User Meeting  
@ UC Louvain (Feb 2019)

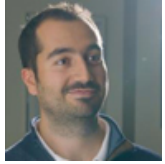


- EasyBuild community has been growing rapidly the last couple of years
- used by **hundreds of HPC sites and companies worldwide**,  
**incl. JSC, CSCS, SURFsara, Pfizer, ...**
- very welcoming & supportive to newcomers

mailing list: <https://lists.ugent.be/www/info/easybuild>  
Slack channel: [easybuild.slack.com](https://easybuild.slack.com)  
IRC: #easybuild on FreeNode



# easybuild maintainers



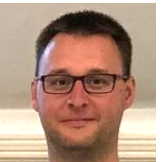
**Damian Alvarez** - @damianam  
(Jülich Supercomputing Centre)



**Miguel Dias Costa** - @migueldiascosta  
(National University of Singapore)



**Pablo Escobar** - @pescobar  
(sciCORE, University of Basel)



**Kenneth Hoste** - @boegel  
(HPC-UGent)



**Adam Huffman** - @verdurin  
(Big Data Institute, University of Oxford)



**Samuel Moors** - @smoors  
(Free University of Brussels)



**Alan O'Cais** - @ocaisa  
(Jülich Supercomputing Centre)



**Bart Oldeman** - @bartoldeman  
(ComputeCanada)



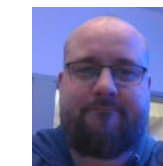
**Ward Poelmans** - @wpoely86  
(Free University of Brussels)



**Åke Sandgren** - @akesandgren  
(Umeå University, Sweden)



**Davide Vanzo** - @vanzod  
(Vanderbilt University)



**Mikael Öhman** - @micketeer  
(Chalmers University of Technology)

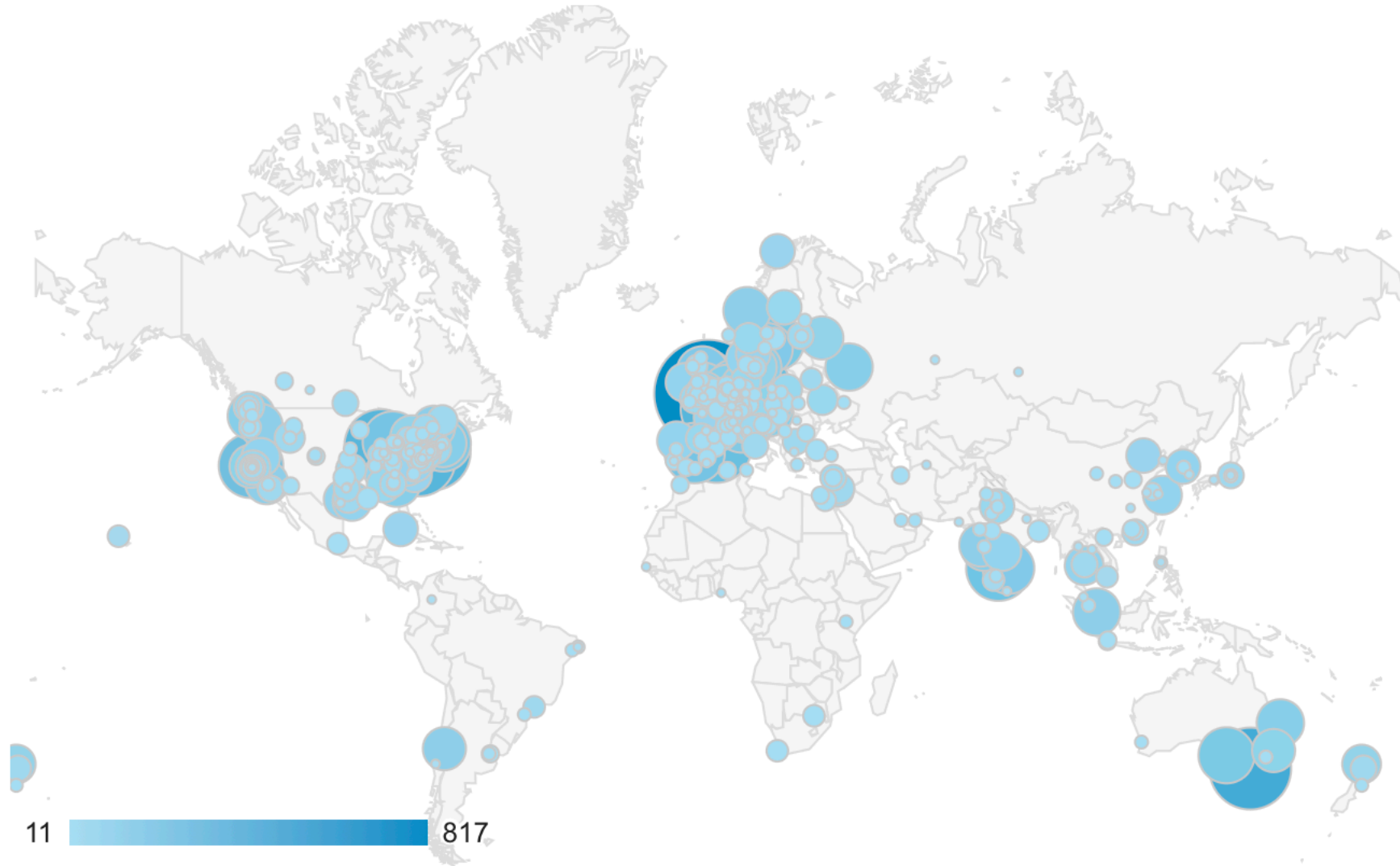
# EasyBuild documentation visits



EasyBuild documentation at <https://easybuild.readthedocs.io> hitting **~500 weekly visitors**

(source: Google Analytics)

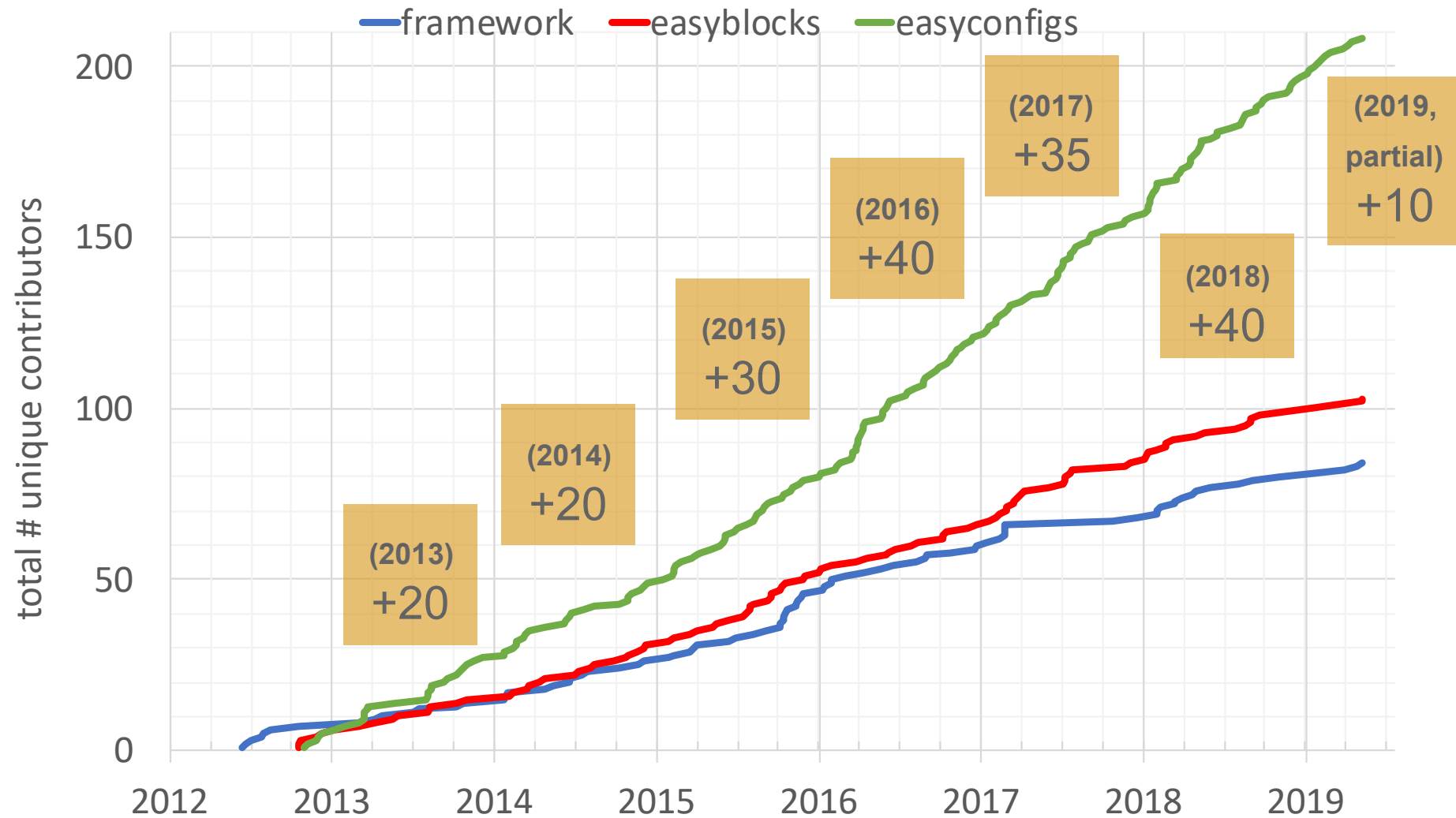
# The sun never sets on EasyBuild...



cities from where <https://easybuild.readthedocs.io> was visited at least 10 times during the last year

(source: Google Analytics)

# Unique contributors (total, per repository)



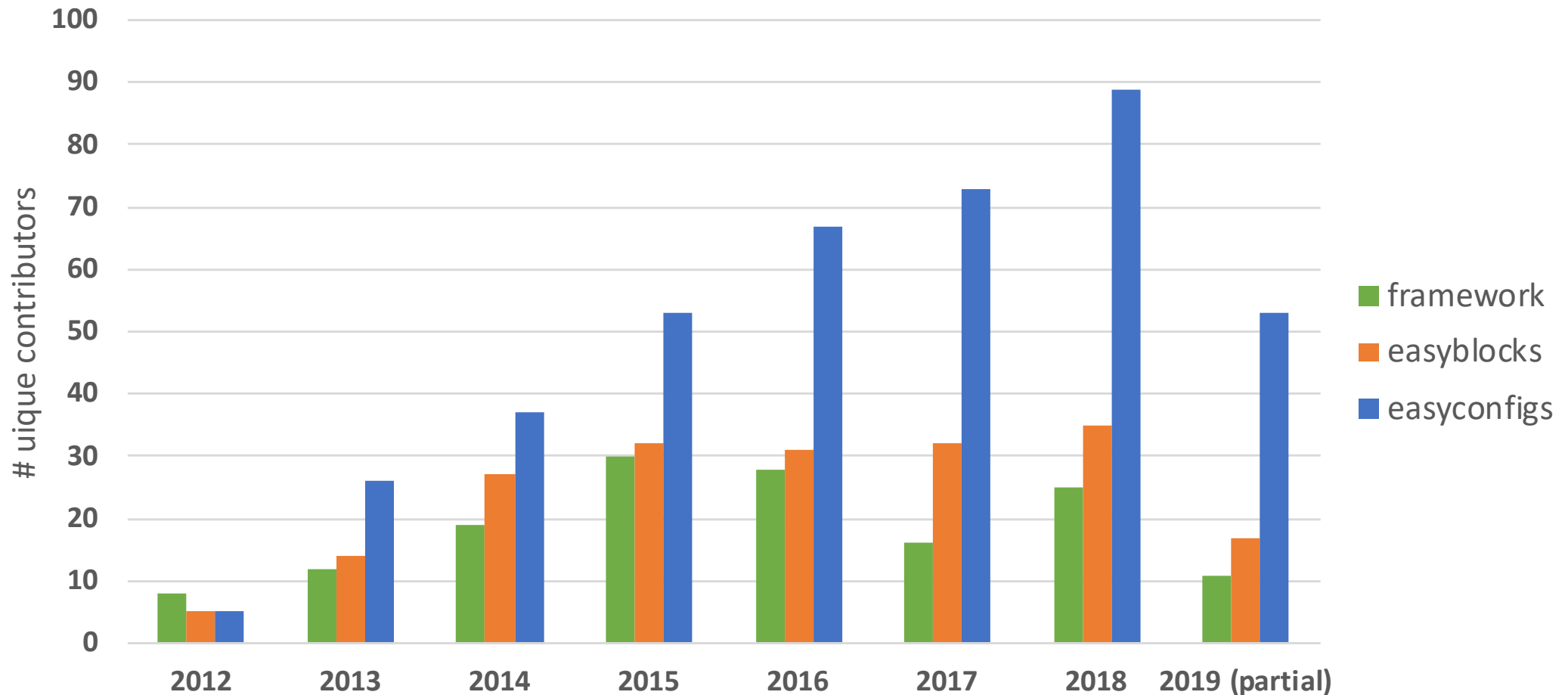
> 200 unique contributors for easyconfigs, > 100 for easyblocks, ~85 for framework

# Unique contributors (per repository, per year)



Group of easyconfigs contributors is growing every year (reaching 100 in 2019?).

Stable group of easyblocks contributors, fluctuating a bit for framework.



# Integration with Docker/Singularity



<https://easybuild.readthedocs.io/en/latest/Containers.html>

- (experimental) integration with Singularity since EasyBuild v3.6.0 (April 2018)
  - extended to also support Docker in EasyBuild v3.6.2 (July 2018)
- `eb --containerize` (or `'eb -C'`) generates recipe to build container image
- generated recipe uses 'eb' to build and install specified software in container image
- `--container-build-image` can be used to also build container image
  - makes eb call out to `'sudo singularity build'` (or `'sudo docker build'`)
- **current implementation is experimental, has some rough edges...**

# Integration with Docker/Singularity (demo)



<https://easybuild.readthedocs.io/en/latest/Containers.html>

```
$ export EASYBUILD_CONTAINER_BASE=shub:shahzebsiddiqui/eb-singularity:centos-7.4.1708
$ eb --experimental --containerize TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb
== temporary log file in case of crash /tmp/eb-sMtVBY/easybuild-1luKuM.log
== Singularity definition file created at /home/kehoste/containers/
Singularity.TensorFlow-1.13.1-foss-2018b-Python-3.6.6
== Temporary log file(s) /tmp/eb-sMtVBY/easybuild-1luKuM.log* have been removed.
== Temporary directory /tmp/eb-sMtVBY has been removed.
$ cat /home/kehoste/containers/Singularity.TensorFlow-1.13.1-foss-2018b-Python-3.6.6
Bootstrap: shub
From: shahzebsiddiqui/eb-singularity:centos-7.4.1708

%post
...
# change to 'easybuild' user
su - easybuild

eb TensorFlow-1.13.1-foss-2018b-Python-3.6.6.eb --robot --installpath=/app/ --prefix=/
scratch --tmpdir=/scratch/tmp

%environment
...
module load TensorFlow/1.13.1-foss-2018b-Python-3.6.6
```

# Other features that were not covered in detail...



<http://easybuild.readthedocs.io>

- letting users manage their software stack on top of centrally provided modules
- installing hidden modules, hiding certain dependencies & toolchains
- support for using RPATH linking
- partial installations: only (re)generate module file, install additional extensions
- submitting installations as jobs to an HPC cluster via `--job` (distributed installation!)
- creating packages (RPMs, ...) for software installations done with EasyBuild
- installing (bundles of) Python packages for multiple Python versions in a single prefix

# Outlook to easybuild v4.0

<https://github.com/easybuilders/easybuild/issues/447>

Ongoing effort, see '4.x' branches in EasyBuild repositories on GitHub!

- ✓ support for running EasyBuild on top of Python 3.5 (or more recent Python 3)
- ✓ no more required dependencies (vsc-base has been ingested, setuptools no longer needed)
- WIP deprecated 'dummy' toolchain, replaced with 'system' toolchain (mostly about renaming)
  - custom easyblock for OpenMPI with more sensible default configuration
- WIP switch to using `pip` by default for installing Python packages (?)
  - (hopefully) stable integration with Singularity & Docker
  - **ETA: summer 2019**



easybuild

vs



**Spack**

<https://spack.rtf.d.io>

- EasyBuild: GPLv2 license; Spack: MIT/Apache 2.0 license
- **no stable releases yet for Spack (< 1.0), EasyBuild is stable since 2012**
- on par w.r.t. amount of supported software (but differences w.r.t. which software)
- (originally) targeted for different use cases: HPC support teams vs developers
- fixed dependency/toolchain versions in EasyBuild vs flexible CLI in Spack
- running EasyBuild on top of Python 3 is not supported yet (but coming really soon!)
- macOS support in EasyBuild is limited (no toolchains/testing for macOS)
- both projects are backed by an active & supportive community!

**For a more detailed comparison,**

**see [https://archive.fosdem.org/2018/schedule/event/installing\\_software\\_for\\_scientists](https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists)**

# Papers on easybuild

## **Modern Scientific Software Management Using EasyBuild and Lmod**

Markus Geimer (JSC), Kenneth Hoste (HPC-UGent), Robert McLay (TACC)

[http://easybuilders.github.io/easybuild/files/hust14\\_paper.pdf](http://easybuilders.github.io/easybuild/files/hust14_paper.pdf)

## **Making Scientific Software Installation Reproducible On Cray Systems Using EasyBuild**

Petar Forai (IMP), Guilherme Peretti-Pezzi (CSCS), Kenneth Hoste (HPC-UGent)

[https://cug.org/proceedings/cug2016\\_proceedings/includes/files/pap145.pdf](https://cug.org/proceedings/cug2016_proceedings/includes/files/pap145.pdf)

## **Scientific Software Management in Real Life: Deployment of EasyBuild on a Large Scale System**

Damian Alvarez, Alan O’Cais, Markus Geimer (JSC), Kenneth Hoste (HPC-UGent)

<http://easybuilders.github.io/easybuild/files/eb-jsc-hust16.pdf>



# Questions?

Kenneth Hoste (HPC-UGent)

*kenneth.hoste@ugent.be*

**20190510 - CSCS Software Management Course**

*[https://users.ugent.be/~kehoste/EasyBuild\\_20190510\\_CSCS\\_software\\_management.pdf](https://users.ugent.be/~kehoste/EasyBuild_20190510_CSCS_software_management.pdf)*

*<https://easybuilders.github.io/easybuild>*

*<https://easybuild.readthedocs.io>*

# Extra slides

(only used in case of time/interest)

# Using a custom module naming scheme



- a couple of different module naming schemes are included in EasyBuild
  - see `--avail-module-naming-schemes`
  - specify active module naming scheme via `--module-naming-scheme`
  - default: `EasyBuildMNS (<name>/<version>-<toolchain>-<versionsuffix>)`
- **you can implement your own module naming scheme relatively easily**
  - specify how to compose module name using provided metadata
  - via Python module that defines custom derivative class of `ModuleNamingScheme`
  - make EasyBuild aware of it via `--include-module-naming-schemes`
- decouple naming of install dirs vs modules via `--fixed-installdir-naming-scheme`

# Flat module naming scheme



## legend

(not available)

(available)

(loaded)

- all modules are always available for loading
- long(er) module names
- 'module avail' may be overwhelming for users
- too easy to load incompatible modules together

GCC/5.3.0

GCC/6.1.0

OpenMPI/1.10.2-GCC-5.3.0

OpenMPI/2.1.0-GCC-5.3.0

OpenMPI/1.10.3-GCC-6.1.0

OpenMPI/2.1.0-GCC-6.1.0

FFTW/3.3.4-gompi-2016.04

FFTW/3.3.6-gompi-2016.04

FFTW/3.3.4-gompi-2016.07

FFTW/3.3.6-gompi-2016.07

# Hierarchical module naming scheme (1)



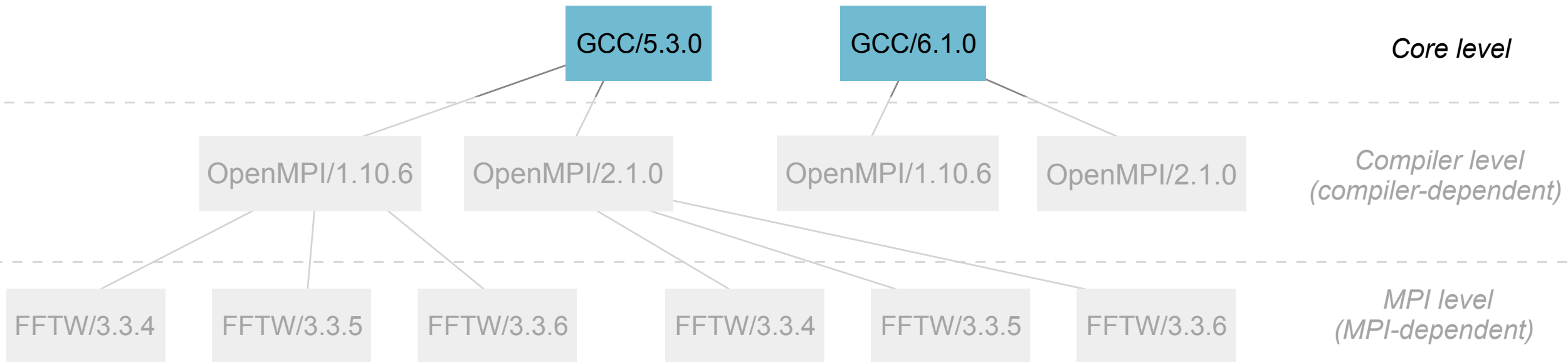
- modules are organised in a tree-like fashion
- initially, only 'core' modules are available for loading
- typically 3 hierarchy levels: core, compiler-dependent, MPI-dependent
- other modules are only visible via 'module spider'

*legend*

(not available)

(available)

(loaded)



# Hierarchical module naming scheme (2)



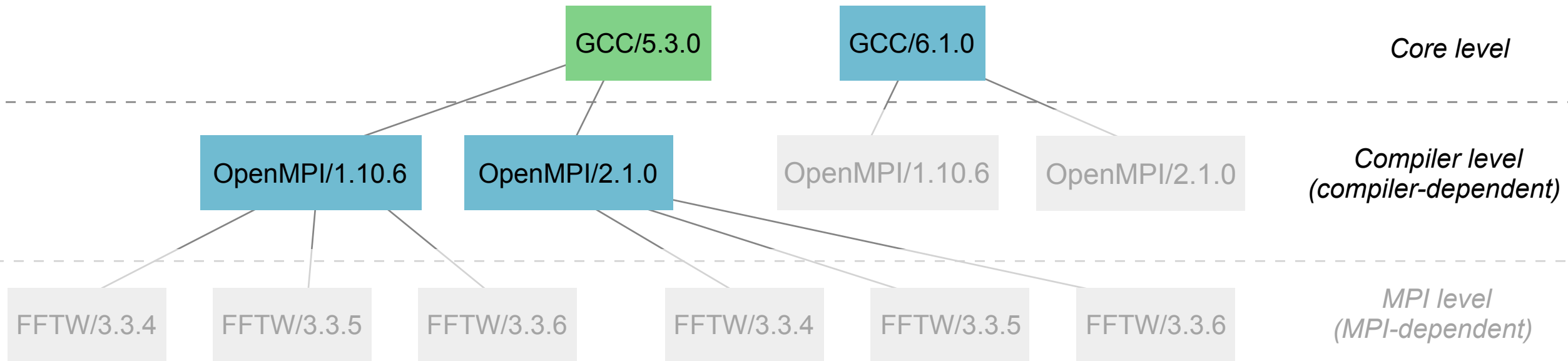
- Core modules may extend `$MODULEPATH` with an additional location
- loading a Core module may make more modules available
- in this example, loading a GCC module makes OpenMPI modules available

legend

(not available)

(available)

(loaded)



# Hierarchical module naming scheme (3)



- even more modules may be made available by loading other modules
- for example, loading an OpenMPI modules reveals MPI-dependent modules
- EasyBuild can organise modules in hierarchy for you!

*legend*

(not available)

(available)

(loaded)

