Introduction to EasyBuild

Getting Scientific Software Installed With Ease

Kenneth Hoste
HPC-UGent, Ghent University, Belgium
kenneth.hoste@ugent.be

http://users.ugent.be/~kehoste/EasyBuild_HPCAC_intro_20160323.pdf

March 23rd 2016 – HPC Advisory Council conference – Lugano

whoami

- PhD in Computer Science from Ghent University (Belgium)
- joined HPC-UGent team in October 2010
- main tasks: user support & training, *software installations*
- inherited maintenance of EasyBuild in 2011
- slowly also became lead developer & release manager

- e-mail: `kenneth.hoste@ugent.be`
- Twitter: `@kehoste`
- GitHub: `https://github.com/boegel`
- IRC (Freenode): `boegel`
- Google+: `kenneth.hoste@gmail.com`

HPC-UGent in a nutshell

`http://www.ugent.be/hpc` – `http://www.vscentrum.be`

- HPC team at central IT dept. of Ghent University (Belgium)
- 9+1 team members: 1 manager, ∼3 user support, ∼6 sysadmin
- 4+1 Tier2 clusters + one Tier1 cluster (8.5k cores)
- ∼1.8k user accounts, across all scientific domains
- tasks: hardware, system administration, user support/training, ...

- member of Flemish Supercomputer Centre (VSC)
  virtual centre, collaboration between Flemish university associations

Tasks for HPC user support teams

- resolving problems that occur when using the HPC system(s)
  - *"I lost my private key/password, and now I can't log in. Help?"*
  - *"My job crashed, and I have no idea why. What happened?"*
  - *"My stuff doesn't work anymore, and I didn't change a thing!"*
- answering questions, from simple to very technical
- **installing (scientific) software tools/libraries/applications**
- helping users improve their workflow (not necessarily by request)
- training: Linux basics, OpenMP, MPI, Python, etc.
- ~~performance analysis and optimisation of large scientific applications~~
- ~~consultancy services w.r.t. developing scientific software~~

Installing scientific software for users

Typical way in which scientific software is installed for users:

- by user request: new software, version upgrades, more variants, ...
- on a (shared NFS) filesystem available on every workernode
- specifically targetted to the HPC cluster it will be used on
  - **built from source** (if possible)
  - separate installation per cluster
  - **highly optimized** for system architecture
    - linked with heavily tuned libraries (MPI, BLAS, LAPACK, ...)
    - built with (equivalent of) `-march=native`/`-xHost`
- rebuild when updates for compilers/libraries become available
- installations remain available during lifetime of system
- accompanying *module file* is provided for easy access

Environment modules

- canonical way of giving users access to installed (scientific) software
- used on most HPC systems ($>$ 80%[1]), since mid 90's
- module file specifies changes to user environment (in Tcl/Lua subset)
- modules tool applies those changes to the current session (!)
- **easy interface for users**:
  - available software: '`module avail [name]`'
  - prepare environment: '`module load <name>/<version>`'
  - show loaded modules: '`module list`'
  - rollback changes to environment: '`module unload <name>`'
  - start afresh: '`module purge`'
- Tcl-based environment modules system is most prevalent (for now)
- Lmod: Lua-based modules tool, *vastly* improves user experience

(1) `http://hpcugent.github.io/easybuild/files/SC15_BoF_Getting_Scientific_Software_Installed.pdf`

“Please install <software> on the HPC?”

The most common type of support request from users is to install (scientific) software; this covers over 25% of support tickets at HPC-UGent.

Installing (lots of) *scientific* software is:

- error-prone, trial-and-error
- tedious, hard to get right
- repetitive & boring (well...)
- time-consuming (hours, days, even weeks)
- frustrating (e.g., dependency hell)
- sometimes simply not worth the effort...

Common issues with scientific software

Researchers focus on the *science* behind the software they implement, and care little about tools, build procedure, portability, ...

Scientists are not software developers or sysadmins (nor should they be).

"If we would know what we are doing, it wouldn't be called 'research'."

This results in:

- 'incorrect' use of build tools
- use of non-standard build tools (or broken ones)
- incomplete build procedure, e.g., no configure or install step
- interactive installation scripts
- hardcoded parameters (compilers, libraries, paths, ...)
- poor/outdated/missing/incorrect documentation
- dependency (version) hell

Prime example I: WRF

Weather Research and Forecasting Model (`http://www.wrf-model.org`)
(one of the top 5 applications on Blue Waters)

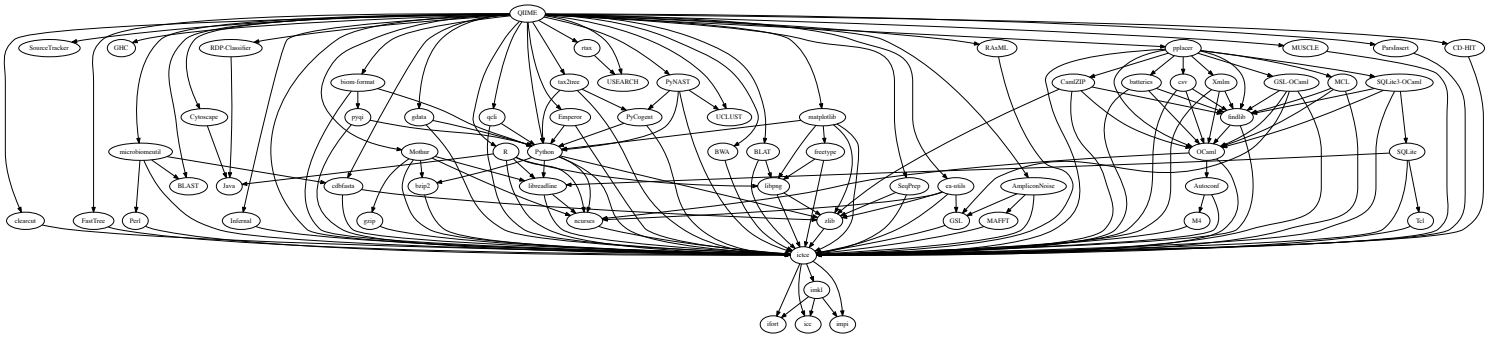
- dozen dependencies: netCDF (C, Fortran), HDF5, tcsh, JasPer, ...
- known issues in last release are (only) documented on website
  no patch file provided, infrequent bugfix releases
- interactive '`configure`' script :(
- resulting `configure.wrf` needs work:
  fix hardcoded settings (compilers, libraries, ...), tweaking of options
- custom '`compile`' script (wraps around '`make`')
  building in parallel is broken without fixing the `Makefile`
- no actual installation step

Wouldn't it be nice to build & install WRF with a single command?

`http://easybuild.readthedocs.org/en/latest/Typical_workflow_example_with_WRF.html`

Prime example II: QIIME

QIIME: Quantitative Insights Into Microbial Ecology (`http://qiime.org/`)



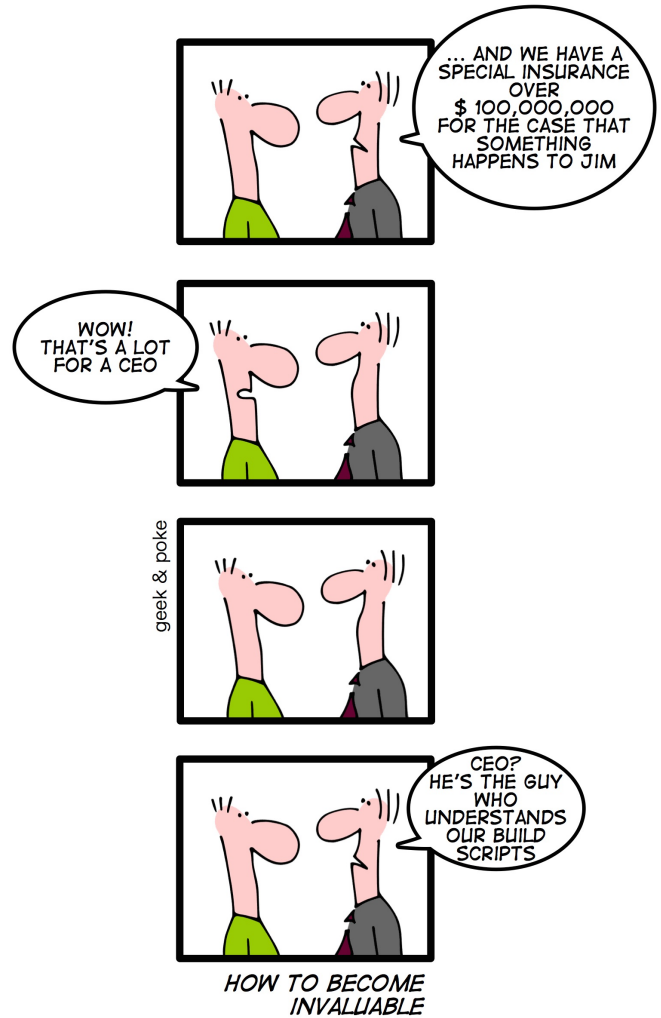
- scientific research domain: bioinformatics ...
- 59 dependencies in total (*without* compiler toolchain), some optional
  - depends on Haskell (GHC), Java, Python, R, Perl, OCaml, ...
  - several deps use a non-standard build procedure (in various degrees)
- *very* picky about dependency versions (e.g., *must* be Python v2.7.*3*)
- took us several weeks to get it installed (using Intel compilers!)...
- **...now we can (re)build/install it all with a single command!**

(disclaimer: support for QIIME not included yet in latest EasyBuild release)

Houston, we have a problem

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

- huge burden on HPC user support teams
- researchers lose lots of time (waiting)
- sites typically resort to in-house scripting
- very little collaboration among HPC sites :(



What about existing tools?

Existing tools are *not* well suited to scientific software and HPC systems.

- package managers: **yum** (RPMs), **apt-get** (.deb), ...
- **Homebrew** (Mac OS X), `http://brew.sh/`
- **Linuxbrew**, `http://brew.sh/linuxbrew/`
- **Portage** (Gentoo), `http://wiki.gentoo.org/wiki/Project:Portage`
- **pkgsrc** (NetBSD & (a lot) more), `http://pkgsrc.org/`
- **Nix**, `http://nixos.org/nix`
- **GNU Guix**, `https://www.gnu.org/s/guix`

Common problems:

- usually poor support for multiple versions/builds side-by-side
- not flexible enough to deal with idiosyncrasies of scientific software
- little support for scientific software, other compilers (not GCC), MPI

EasyBuild: building software with ease

http://hpcugent.github.io/easybuild/

- framework for installing (scientific) software on HPC systems
- collection of Python packages and modules
- in-house since 2009, open-source (GPLv2) since 2012
- now: thriving community; actively contributing, driving development
- new release every 6–8 weeks (latest: EasyBuild v2.7.0, Mar 20th 2016)
- supports over 850 different software packages

  including CP2K, GAMESS-US, GROMACS, NAMD, NWChem, OpenFOAM, PETSc, QuantumESPRESSO, WRF, WPS, ...
- well documented: http://easybuild.readthedocs.org

EasyBuild: feature highlights

- fully **autonomously** building and installing (scientific) software
  - automatic dependency resolution
  - automatic generation of module files (Tcl or Lua syntax)
- thorough **logging** of executed build/install procedure
- **archiving** of build specifications ('*easyconfig files*')
- highly **configurable**, via config files/environment/command line
- **dynamically extendable** with additional *easyblocks*, *toolchains*, etc.
- support for **custom module naming schemes** (incl. hierarchical)
- **comprehensively tested**: lots of unit tests, regression testing, ...
- actively developed, **collaboration** between various HPC sites
- worldwide **community**

EasyBuild terminology

- EasyBuild *framework*
  - core of EasyBuild: Python modules & packages
  - provides supporting functionality for building and installing software
- *easyblock*
  - a Python module, 'plugin' for the EasyBuild framework
  - implements a (generic) software build/install procedure
- *easyconfig* file (`*.eb`)
  - build specification: software name/version, compiler toolchain, etc.
- compiler *toolchain*
  - compilers with accompanying libraries (MPI, BLAS/LAPACK, ...)

Putting it all together

The EasyBuild *framework* leverages *easyblocks* to automatically build and install (scientific) software using a particular *compiler toolchain*, as specified by one or more *easyconfig files*.

EasyBuild: system requirements

- Linux x86_64 HPC systems is main target platform (for now
  - Red Hat-based systems (Scientific Linux, CentOS, RHEL, . . .)
  - also other Linux distros: Debian, Ubuntu, OpenSUSE, SLES, . . .
  - kind of works on OS X, but not really a target platform
  - *no* Windows support (and none planned)
  - **stable support for Cray systems since EasyBuild v2.7.0**
  - support for Linux@POWER systems is being looked into (by TAMU)
- Python v2.6.x or more recent v2.x (not Python 3 compatible (yet))
- a modules tool:
  - latest release of Tcl/C environment modules (version 3.2.10);
  - or one of the Tcl-only versions of environment modules;
  - or a recent version of *Lmod* (5.6.3 or more recent) **(recommended!)**
- (a system C/C++ compiler, to get started)

'Quick' demo for the impatient

```
eb HPL-2.1-foss-2016a.eb --robot
```

- **downloads** all required sources (best effort)
- **builds/installs** *foss* toolchain (be patient) + HPL on top of it

  foss: GCC, OpenMPI, LAPACK, OpenBLAS, FFTW, ScaLAPACK

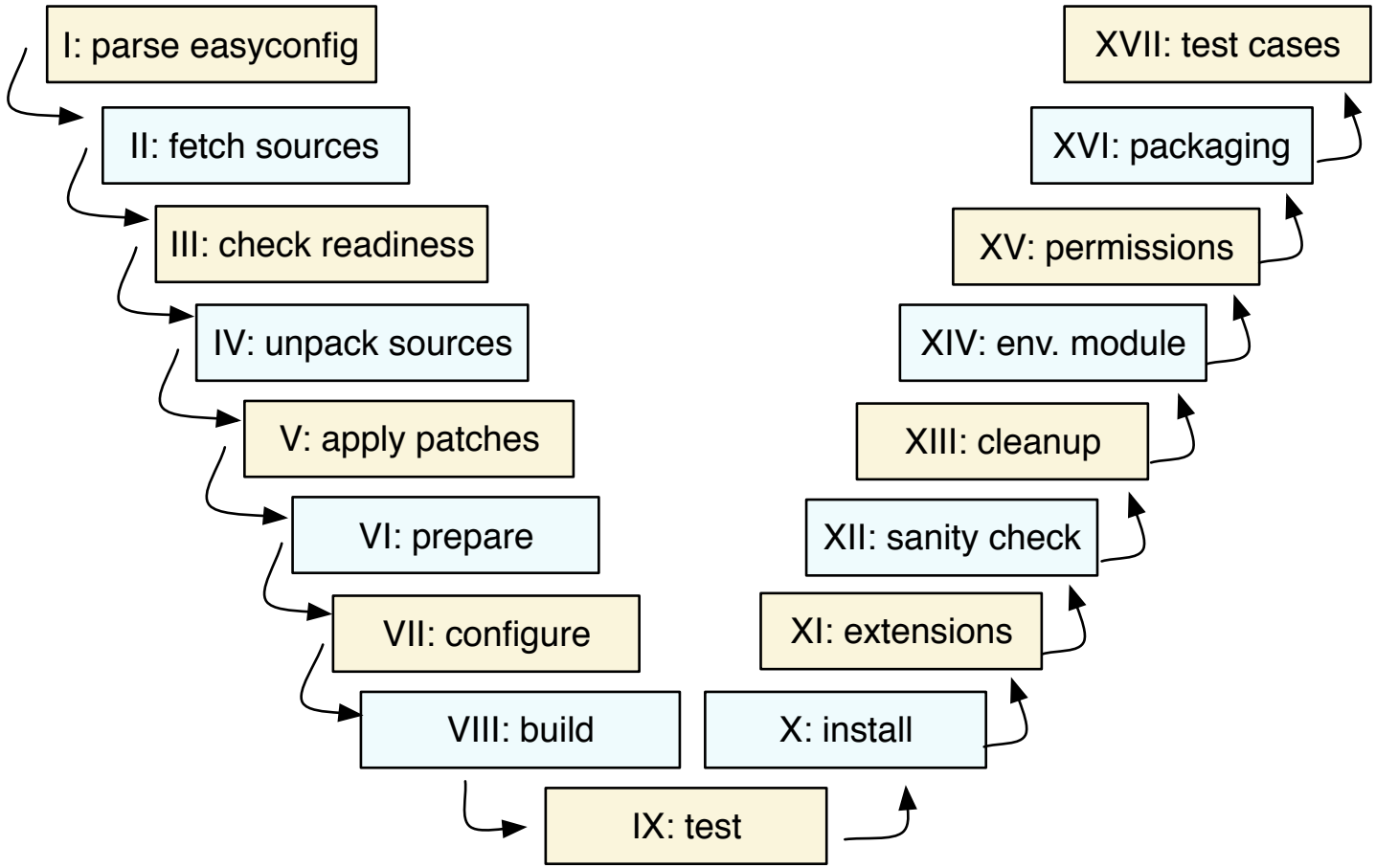
  note: requires `libibverbs` to be available
- **generates module file** for each installed software package

Example 'eb' output

```
$ eb GCC-4.9.3.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig GCC-4.9.3.eb
== building and installing GCC/4.9.3...
== 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/GCC/4...
== Build succeeded for 1 out of 1
== Temporary log file(s) /tmp/eb-GyvPHx/easybuild-U1TkEI.log* have been removed.
== Temporary directory /tmp/eb-GyvPHx has been removed.
```

Step-wise install procedure

build and install procedure as implemented by EasyBuild



most of these steps can be customised if required,
via *easyconfig parameters* or a *custom easyblock*

What EasyBuild is (not)

EasyBuild is **not**:

- YABT (Yet Another Build Tool)
- a replacement for your favorite package manager
- a magic solution to all your (installation) problems

EasyBuild can be (and maybe already *is*) a:

- proper way of installing scientific software
- uniform interface that wraps around software build procedures
- huge time-saver, by automating tedious/boring/repetitive tasks
- way to provide a *consistent* software stack to your users
- expert system for software installation on HPC systems
- platform for collaboration with HPC sites world-wide
- tool to empower users to manage their *own* software stack

EasyBuild: statistics

EasyBuild v2.7.0 *(Mar'16)*

- $\sim 25,000$ LoC in framework (17 Python packages, 160 Python modules)

  $+ \sim 5,000$ LoC in `vsc-base` (option parsing/logging)
  $+ \sim 12,500$ LoC more in unit tests
  $\Longrightarrow \sim 42,500$ LoC in total

- 194 easyblocks in total ($\sim 18,000$ Loc)

  165 software-specific easyblocks
  29 generic easyblocks

- 909 different software packages supported
  (incl. toolchains & bundles)

  bio: 203, tools: 123, vis: 99, devel: 78, lib: 77, math: 54,
  data: 53, toolchain: 38, chem: 38, lang: 32, numlib: 25,
  perf: 22, system: 21, cae: 16, compiler: 14, mpi: 11, phys: 6

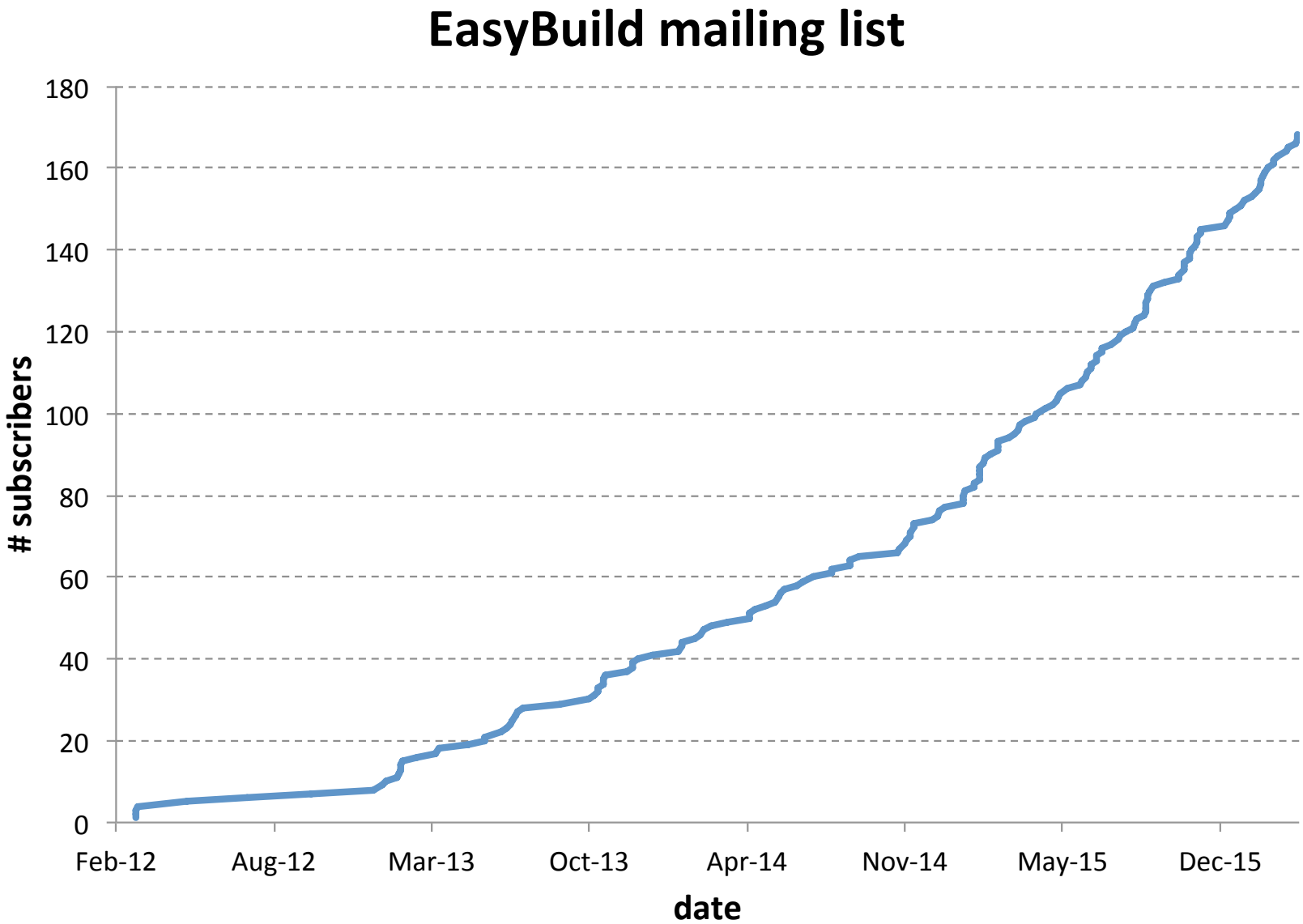
- 5,580 easyconfig files: different versions/variants, toolchains, ...

EasyBuild community

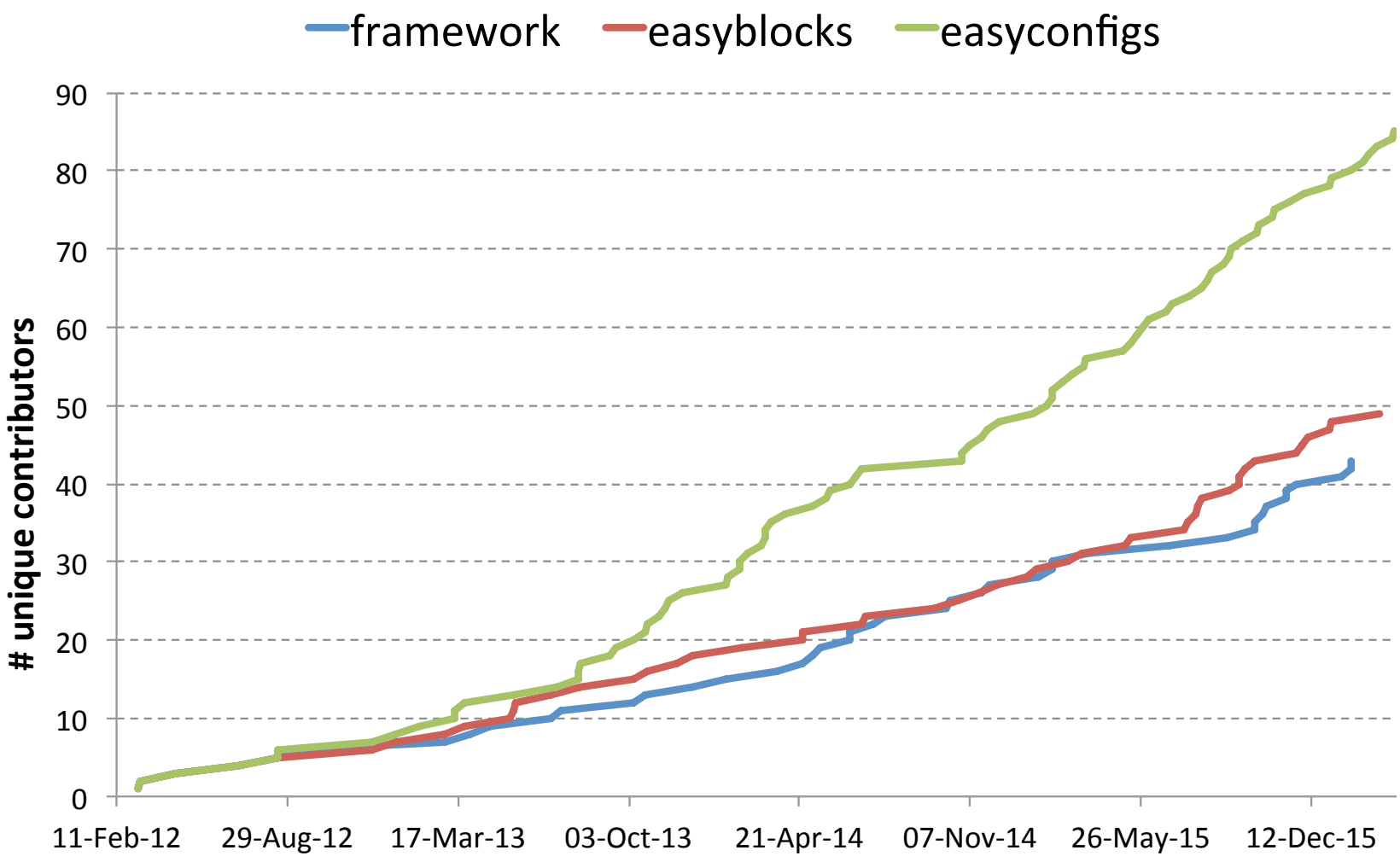
- Ghent University & partners in Flemish Supercomputing Centre
- Jülich Supercomputing Centre (JSC) – Germany
- Swiss National Supercomputing Centre (CSCS) – Switzerland
- (small) sites across Europe: Luxembourg, Cyprus, Switzerland, UK, ...
- US sites: Stanford University, University of Colorado Boulder, ...
- ...and all across the world: New Zealand, Australia, Cuba, ...
- and also commercial companies: Bayer (Germany), *****, ...

EasyBuild community by numbers

- 6 '*Getting Scientific Software Installed*' BoF sessions at ISC/SC
- 10 two/three-day EasyBuild hackathons + 1 user meeting
- ∼20-25 'active' souls on the `#easybuild` IRC channel
- a couple of dozen of HPC sites using it around the world
- 47 EasyBuild conference calls
- 168 subscribers to the EasyBuild mailing list
- framework: 966 merged PRs (50 open)
- easyblocks: 634 merged PRs (51 open)
- easyconfigs: 1,937 merged PRs (325 open)


EasyBuild mailing list
subscribers
0
20
40
60
80
100
120
140
160
180
Feb-12
Aug-12
Mar-13
Oct-13
Apr-14
Nov-14
May-15
Dec-15
date

EasyBuild contributors



Recent projects similar to EasyBuild

- **Spack** (LLNL) - `http://scalability-llnl.github.io/spack/`
- **Maali** (Pawsey) - `https://github.com/chrisbpawsey/maali/`
- **Smithy** (NICS, ORNL) - `http://anthonydigirolamo.github.io/smithy/`

Major differences with EasyBuild:

- slightly different approach
- smaller community
- fewer supported software packages
- missing features
- less flexibility
- not so powerful (except Spack?)

All have expressed interest in cross-community collaboration.

HUST'14 paper

Modern Scientific Software Management Using EasyBuild and Lmod

Markus Geimer (JSC)

Kenneth Hoste (HPC-UGent)

Robert McLay (TACC)

`http://hpcugent.github.io/easybuild/files/hust14_paper.pdf`

- paper at HPC User Support Tools workshop (HUST'14 @ SC14)
- explains basics of module tools, EasyBuild and Lmod
- highlights issues with current approaches in software installation
- advocates use of a hierarchical module naming scheme
- presents EasyBuild and Lmod as adequate tools for (automated) software/module management on HPC systems

EasyBuild: future work

- support more (scientific) software (never-ending story?)
- further extend documentation: generic easyblocks, easyblocks API
- support for more Lmod-specific features
  - module families
  - module properties & sticky modules
- stable support for 'subtoolchain'-aware dependency resolution
- (even) better integration with GitHub
- support for RPATH-style linking of libraries
- 'fat' easyconfig format (YAML-based?)
- join forces with Spack (LLNL)?

Do you want to know more?

- EasyBuild website: `http://hpcugent.github.io/easybuild`
- EasyBuild documentation: `http://easybuild.readthedocs.org`
- stable EasyBuild releases: `http://pypi.python.org/pypi/easybuild`

  EasyBuild framework: `http://pypi.python.org/pypi/easybuild-framework`
  easyblocks: `http://pypi.python.org/pypi/easybuild-easyblocks`
  easyconfigs `http://pypi.python.org/pypi/easybuild-easyconfigs`
- source repositories on GitHub

  EasyBuild meta package + docs: `https://github.com/hpcugent/easybuild`
  EasyBuild framework: `https://github.com/hpcugent/easybuild-framework`
  easyblocks: `https://github.com/hpcugent/easybuild-easyblocks`
  easyconfigs: `https://github.com/hpcugent/easybuild-easyconfigs`
- EasyBuild mailing list: easybuild@lists.ugent.be

  `https://lists.ugent.be/wws/subscribe/easybuild`
- Twitter: @easy_build
- IRC: `#easybuild` on `chat.freenode.net`

Why I like Lmod and why you should too!

How we learned about Lmod

- EasyBuild makes it very easy to install lots of software/modules quickly
- we started wondering how we could organise our modules tree better
- Lmod and the module hierarchy idea allow to deal with this

And then we discovered a whole bunch of other interesting features...

Lmod: a modern modules tool

`https://tacc.utexas.edu/research-development/tacc-projects/lmod`

- developed by Dr. Robert McLay (TACC, UT Austin)
- created to properly support module hierarchies
- available since Oct'08, *actively developed*, frequent stable releases
- well documented: `http://lmod.readthedocs.org`
- drop-in alternative for Tcl-based module tools (a few edge cases)
- written in Lua, consumes module files in both Tcl and Lua syntax
- (vastly) improves user experience, without hindering experts
- highly community-driven development

Lmod: feature highlights

- module hierarchy-aware design and functionality
  - searching across entire module tree with 'module `spider`'
  - automatic reloading of dependent modules on '`module swap`'
  - marking missing dependent modules as inactive after '`module swap`'
- caching of module files, for responsive subcommands (e.g., `avail`)
- site-customizable behavior via provided hooks
- `ml` command ('`ml`' is '`module list`', '`ml GCC`' is '`module load GCC`', ...)
- load/unload shortcuts via + and -
- various other useful/advanced features, including:
  - case-insensitive '`avail`' subcommand
  - can send subcommand output to stdout (rather than to stderr)
  - defining module families (e.g., '`compiler`', '`mpi`')
  - assigning properties to modules (e.g., '`Phi-aware`')
  - stack-based definition of environment variables (using `pushenv`)
  - user-definable collections of modules (`module save`)
  - and a lot more ...

Example

Behold: the power of the Lmod command line using '`ml`' command:

- see which modules are loaded ('`module list`')
- change to different part of module hierarchy by swapping compilers
- recheck which modules are loaded

```
$ ml
Currently loaded modules:
 1) GCC/4.8.2   2) MPICH/3.1.1   3) FFTW/3.3.2

$ ml -GCC Clang
The following have been reloaded:
 1) FFTW/3.3.2   2) MPICH/3.1.1

$ ml
Currently loaded modules:
 1) Clang/3.4   2) MPICH/3.1.1   3) FFTW/3.3.2
```