Using  to create  ingularity container images

Kenneth Hoste (HPC-UGent)

kenneth.hoste@ugent.be

20190612 - Singularity workshop @ HPCKP'19 (Barcelona)

https://users.ugent.be/~kehoste/EasyBuild_20190612_Singularity_workshop.pdf

https://github.com/boegel/easybuild-singularity-tutorial

https://easybuilders.github.io/easybuild

https://easybuild.readthedocs.io

GHENT UNIVERSITY

https://www.ugent.be/hpc

https://www.vscentrum.be

Vlaanderen is computing

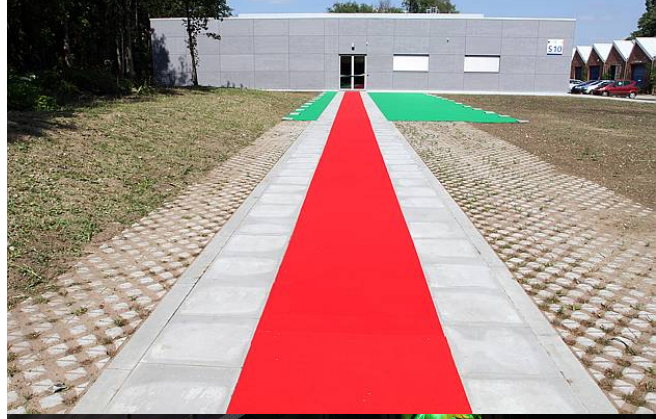
whoami

kenneth.hoste@ugent.be
@boegel *(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  **lead developer & release manager**
- likes family, beer, loud music, FOSS, helping people, dad jokes, stickers, ...
- doesn't like CMake, SCons, Bazel, TensorFlow, OpenFOAM, ...

HPC-UGent

GHENT UNIVERSITY



- 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

Vlaanderen
is computing

Outline

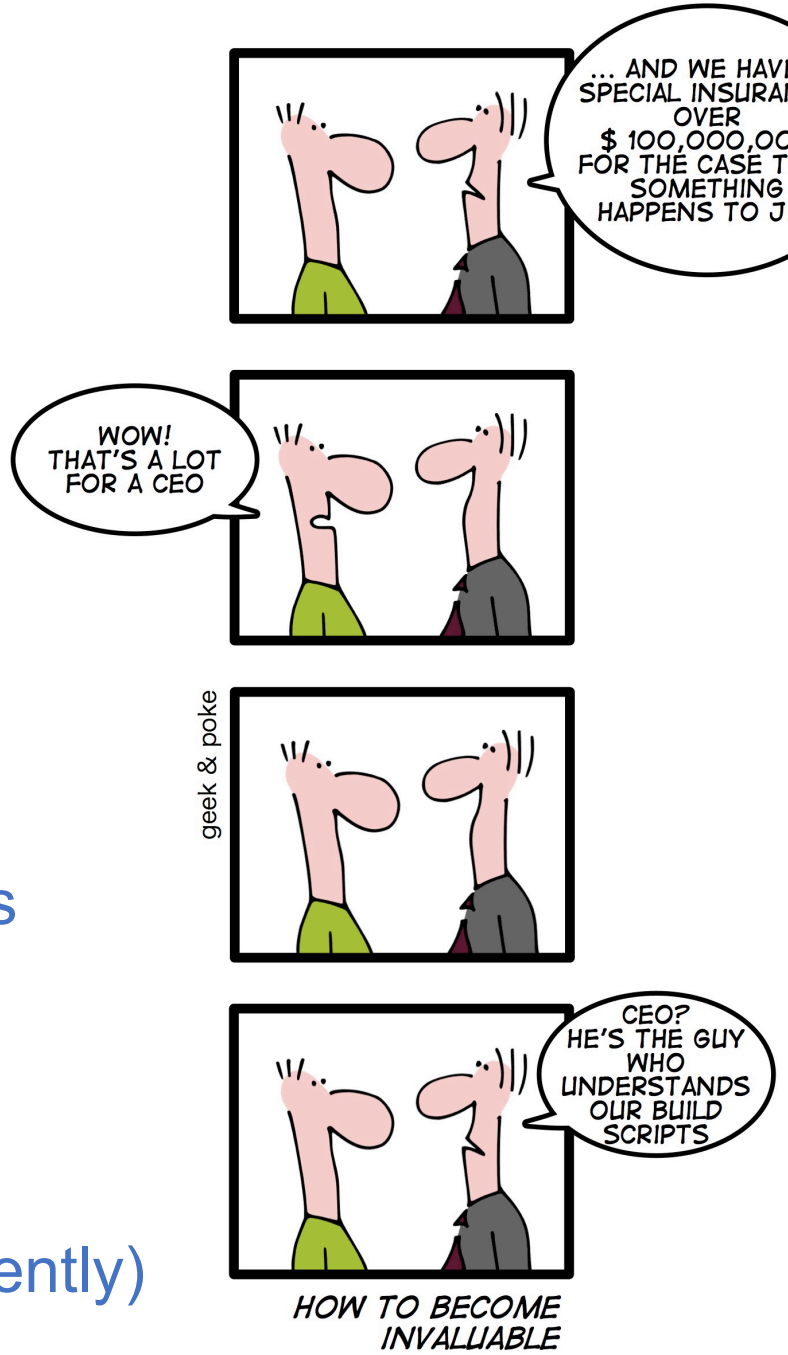
- Quick introduction to 
- Installing and configuration
- Basic usage
- Beyond the basics
- *(coffee break)*
- Using  to create  ingularity container images
  - Using `eb` in Singularity container recipes
  - Generating Singularity container recipes/images via `eb`

See also EasyBuild/Singularity tutorial at
https://github.com/boegel/easybuild-singularity-tutorial

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)



http://geekandpoke.typepad.com/geekandpoke/2010/05/how-to-become-invaluable.html

Prime example: TensorFlow

popular open-source software for Deep Learning (https://www.tensorflow.org)

- auto-installs most dependencies, but not all... (Python, CUDA, ...)
- '`configure`' script is a custom interactive script
  - silent configuration possible, but only if you *know* which `$TF_*` environment variables to set!
- uses Bazel (http://bazel.io) as build tool (there ~~is~~ was a contributed CMake alternative...)
  - *resets environment*, may result in unsetting important env. variables (e.g., `$PYTHONPATH`)
  - quite different from other build tools; e.g. to build TensorFlow:
    ```
    bazel build --config=opt //tensorflow/tools/pip_package:build_pip_package
    ```
  - `--config=opt`, `-c opt` and `-copt=...` <= these 3 options all mean different things...
- installation via '`pip install`' of locally built Python wheel file (`.whl`)

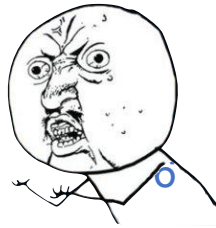
Prime example: TensorFlow

popular open-source software for Deep Learning (https://www.tensorflow.org)

- auto-installs most dependencies, but not all... (Python, CUDA, ...)
- '`configure`' script is a custom interactive script
  - silent configuration possible, but only if you *know* which `$TF_*` environment variables to set!
- uses Bazel (http://bazel.io) as build tool (there ~~is~~ was a contributed CMake alternative...)
  - *resets environment*, may result in unsetting important env. variables (e.g., `$PYTHONPATH`)
  - quite different from other build tools; e.g. to build TensorFlow:

    ```
    bazel build --config=opt //tensorflow/tools/pip_package:build_pip_package
    ```

    No, that's not a typo...
  - `--config=opt`, `-c opt` and `-copt=...` <= these 3 options all mean different things...
- installation via '`pip install`' of locally built Python wheel file (`.whl`)



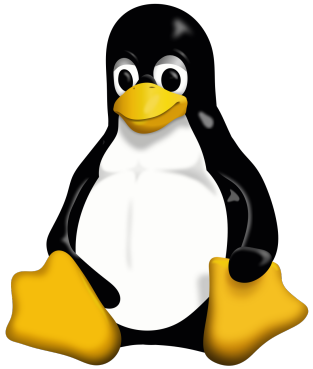
Installing (scientific) software on HPC systems

- key aspects:
  - large-scale **multi-user** system (100s to 1000s of active users)
  - **wide variety of users**, scientific domains, use cases, ...
  - results in **huge software collection** to (centrally) maintain & update
  - **different hardware architectures** (old & new processor generations)
  - important **system libraries** (IB network, GPU, ...) should also be taken into account
  - **conflicting user requirements/expectations** make things even more complex
    - stable software versions vs bleeding edge, Python 2 vs 3, ...
  - performance is key, so software preferably **built from source !**

- **user-friendly interface should be provided to users** to leverage installed software
  - well-established solution is the "environment modules" tool (**Lmod** is quite popular now)
  - `module avail example`, `module load example/1.2.3`, `module list`, ...

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)

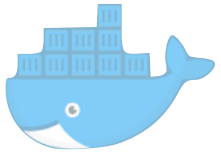
- very popular tool for installing scientific software, but mainly **targeted to individual users**
- **not well suited for centrally managing software** installations on multi-user HPC systems
- installing software with `conda` *usually* results in running **generically optimised binaries**
- hard to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

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 (Singularity/Docker)



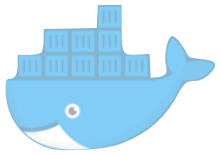
- **lack of attention to optimising for underlying hardware** ("mobility of compute")
- integration with system resources (MPI, CUDA, ...) is *still a problem*
- **someone still needs to be build/maintain/update the container image you need/want !**

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

CONDA (installation tool for Anaconda distribution)

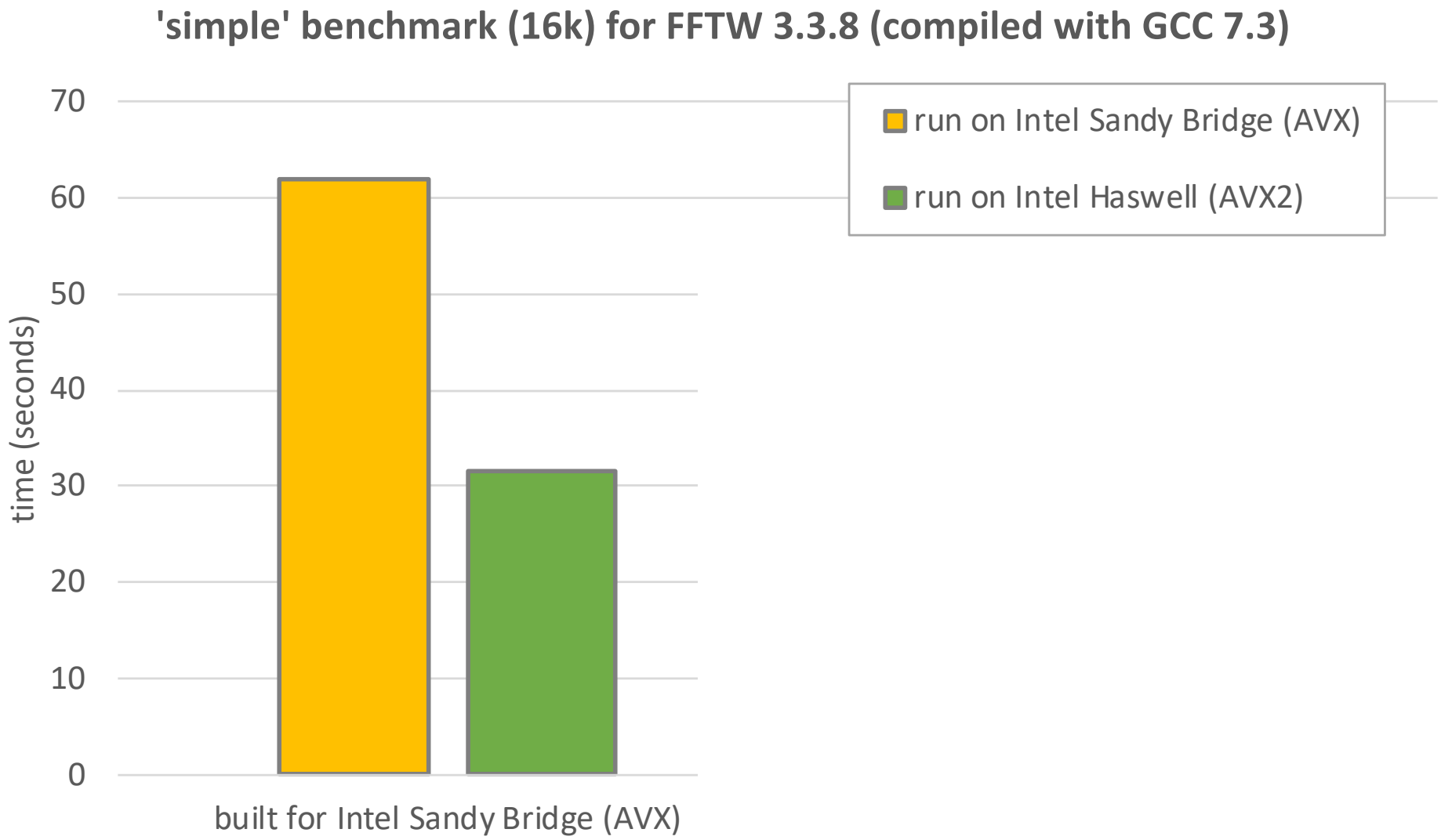
- very popular tool for installing scientific software, but mainly **targeted to individual users**
- **not well suited for centrally managing software** installations on multi-user HPC systems
- installing software with `conda` *usually* results in running **generically optimised binaries**
- hard to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

Containers (Singularity/Docker)



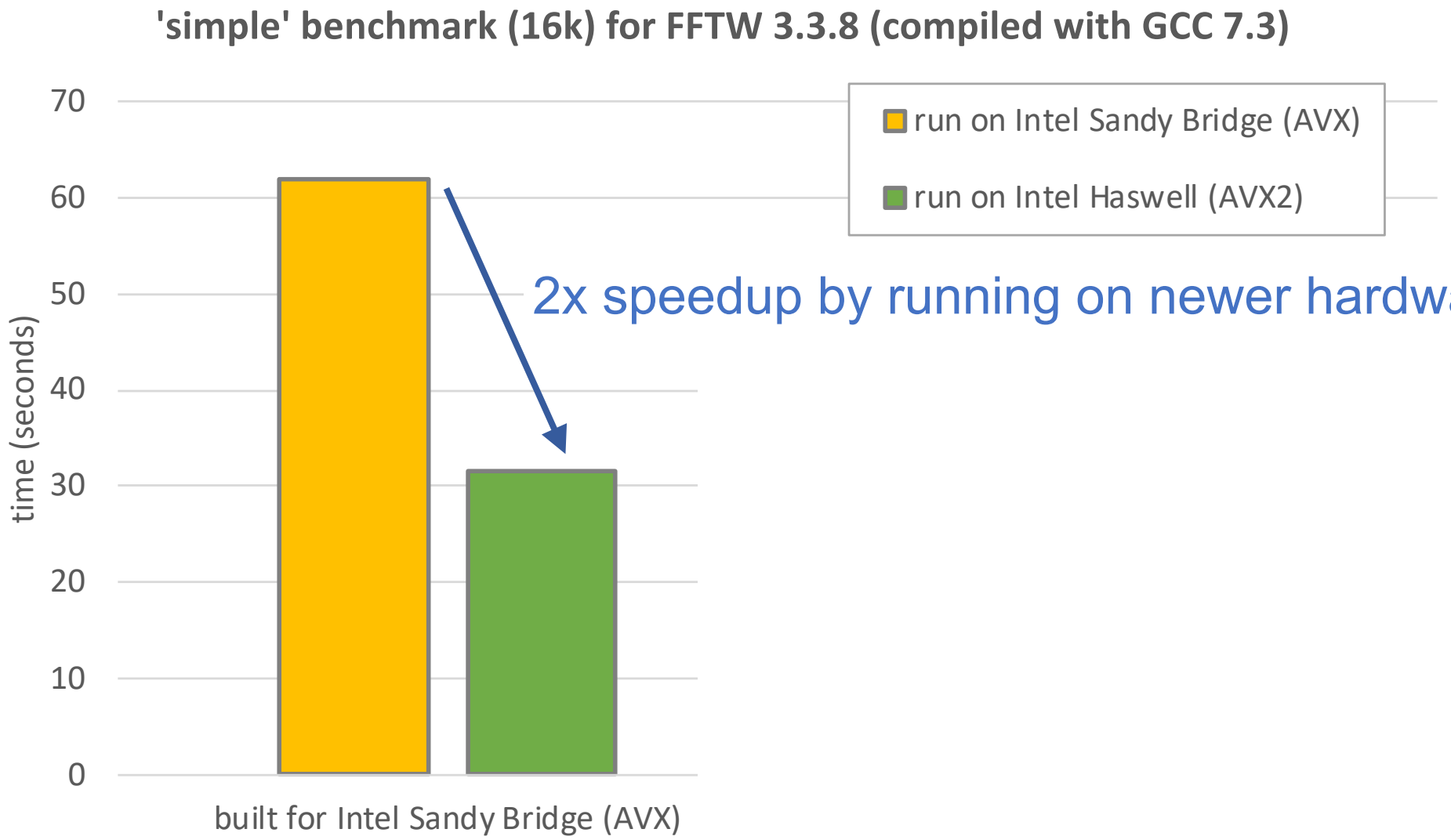
- **lack of attention to optimising for underlying hardware** ("mobility of compute")
- integration with system resources (MPI, CUDA, ...) is *still a problem*
- **someone still needs to be build/maintain/update the container image you need/want !**

Performance is key in HPC



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

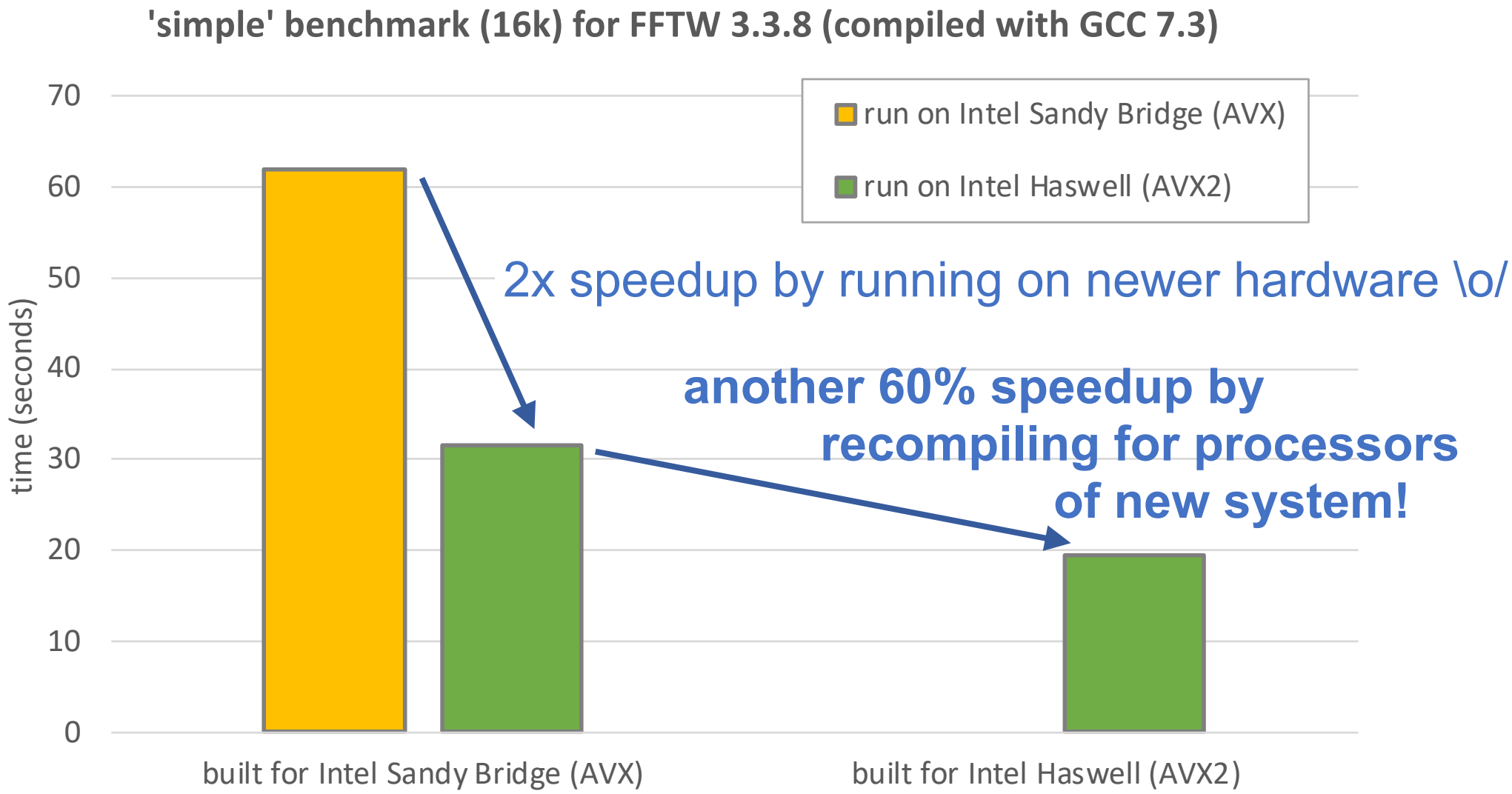
Performance is key in HPC



2x speedup by running on newer hardware \o/

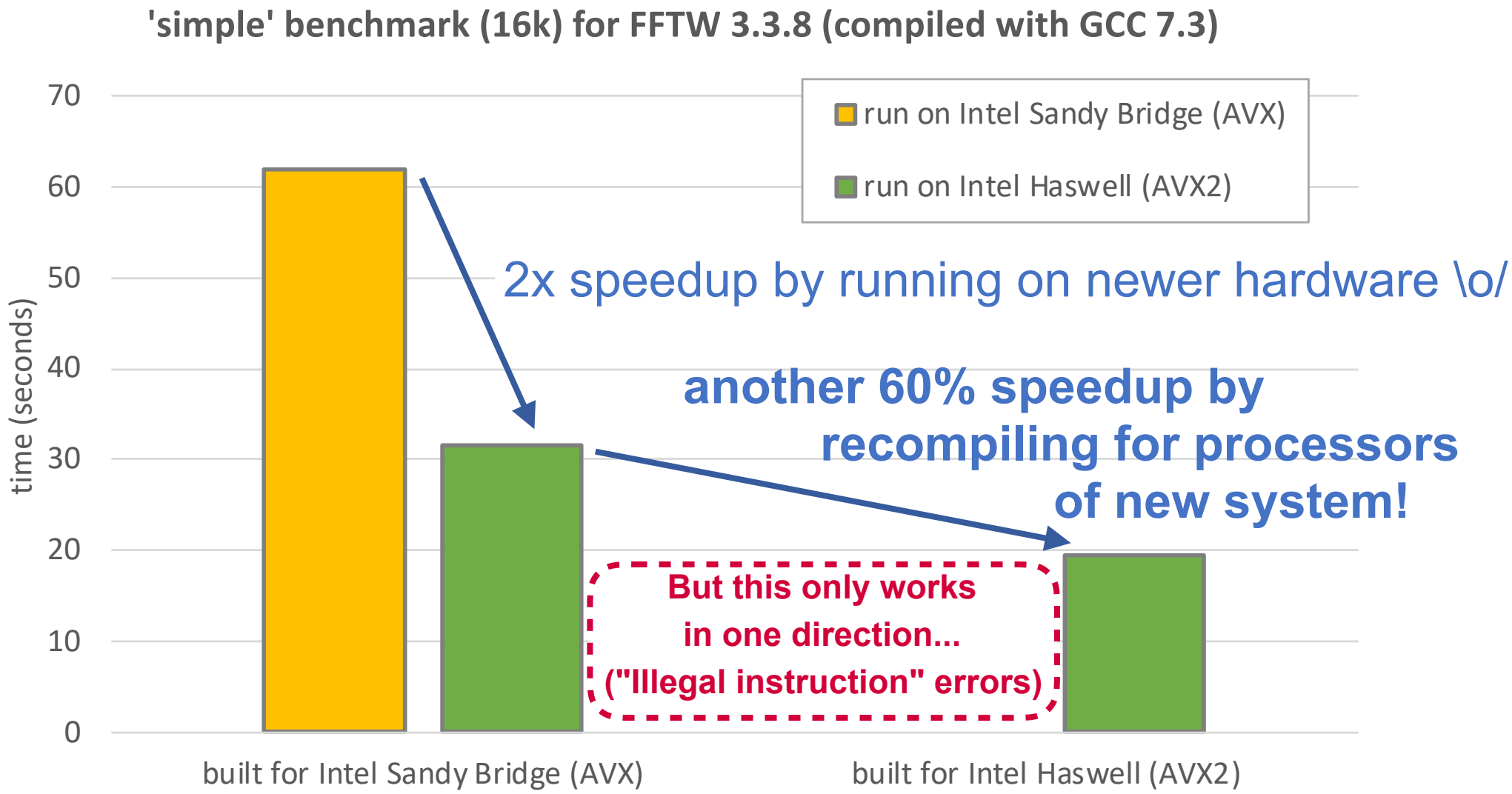
(FFTW 3.3.8 installed in Singularity container, built from source)

Performance is key in HPC



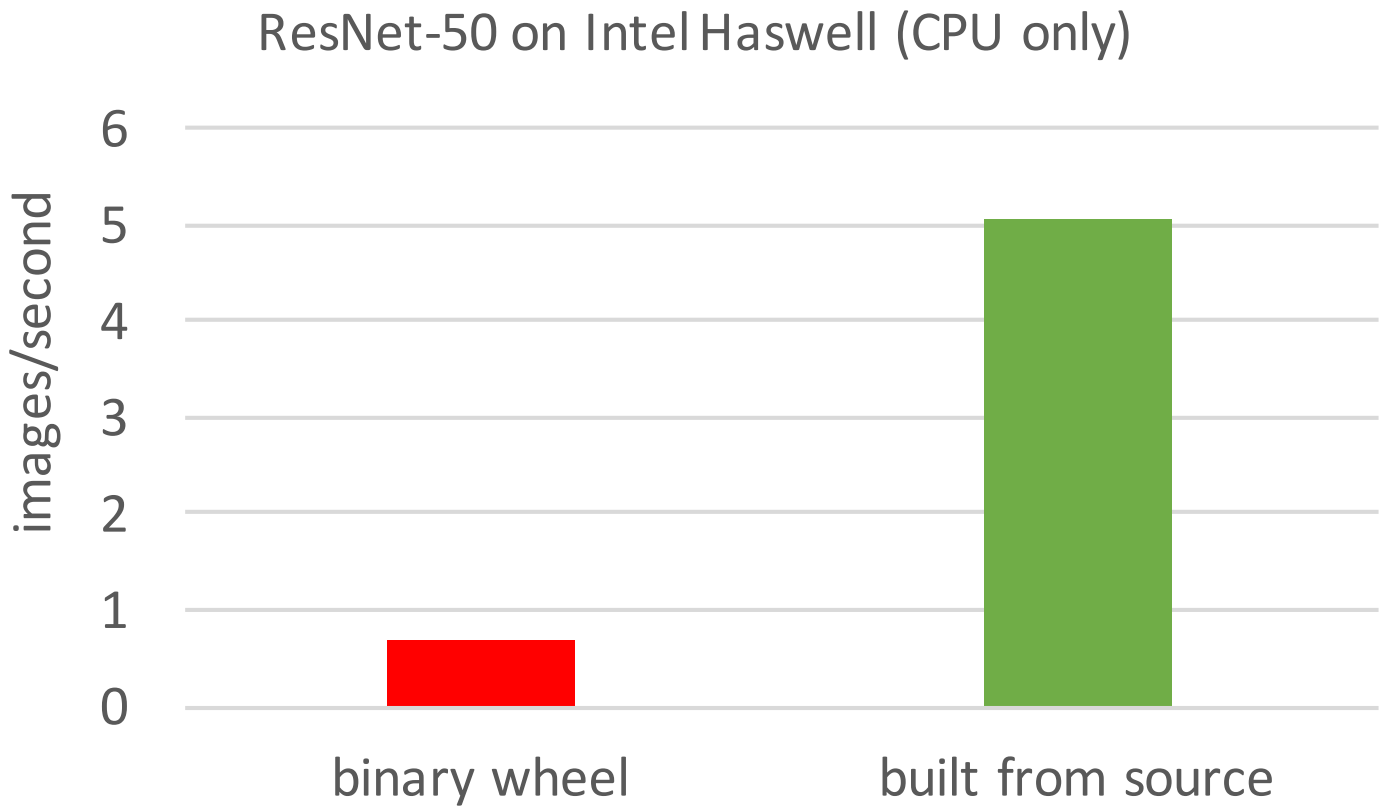
(FFTW 3.3.8 installed in Singularity container, built from source)

Performance is key in HPC



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

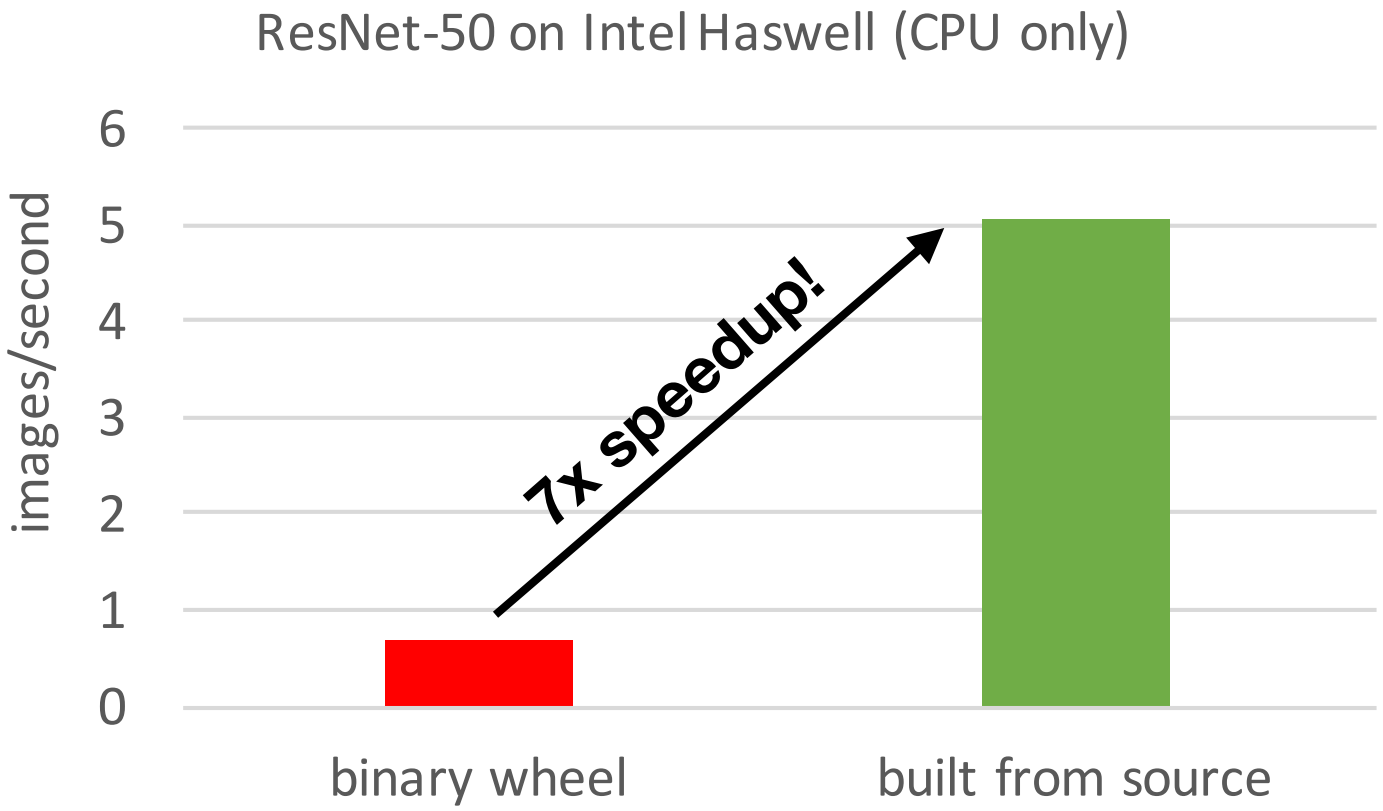
"pip install tensorflow" works fine for me...



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

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

"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 *may* be 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)
- **default is to build/optimise specifically for host architecture**
- implemented in Python 2, requires a modules tool (Lmod is recommended)
- available under GPLv2 license via PyPI, GitHub
- supports different compilers & MPI libraries, x86_64/ARM/POWER, ...
- currently supports 1,780 different software packages (+ extensions)
- originally started by HPC-UGent, now active & helpful *worldwide* community

10 years of easybuild

- project start by Stijn De Weirdt (HPC-UGent) in summer 2009
- stayed in-house for years, first public release in April 2012
- **stable release (EasyBuild v1.0) on November 13th 2012, during SC'12**
- intention was to get feedback, but gradually a community emerged around it...
- **frequent stable releases** since then (latest: EasyBuild v3.9.2, June 9th 2019)
- community-driven development: bug reports, feature requests, contributions

Supported software

https://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html

- latest EasyBuild (v3.9.2) supports installing **1,780 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 PE (GNU, Intel, CCE, PGI), CUDA
  - MPI libraries: OpenMPI, Intel MPI, MPICH, MPICH2, MVAPICH2, Cray MPI, ...
  - BLAS/LAPACK libraries: Intel MKL, OpenBLAS, ScaLAPACK, BLIS, Cray LibSci, ...

terminology

https://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)

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.

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, Cray PE, ...
- active & helpful worldwide **community**

What 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  can be for you

- 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

- 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 build toolchain compiler)
- installation procedure (pick one):
  - `pip install easybuild` (or equivalent thereof using other Python installation tool)
  - or use bootstrap script to install an `EasyBuild` module (see docs for more info)

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

Updating 

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
```

```
$ eb --version
This is EasyBuild 3.8.1 (framework: 3.8.1, easyblocks: 3.8.1) on host example.

$ eb --install-latest-eb-release
...
$ module load EasyBuild/3.9.2
$ eb --version
This is EasyBuild 3.9.2 (framework: 3.9.2, easyblocks: 3.9.2) on host example.
$ which eb
/software/EasyBuild/3.9.2/bin/eb
```

Configuring 

https://easybuild.readthedocs.io/en/latest/Configuration.html

To configure EasyBuild, you can specify settings via 3 configuration levels:

configuration files, environment variables, command line options

- you can mix these configuration levels as you see fit
- *all* configuration options are supported on all 3 configuration levels, no exceptions
- '`prefix`' configuration option determines path for software installations & generated module files, build directories, source files, generated container recipes/images, ...
- overview of all available configuration options via "`eb --help`"

Configuring  : default configuration

https://easybuild.readthedocs.io/en/latest/Configuration.html

It is highly recommended to first configure EasyBuild to your preferences!

Default configuration:

- location for source files & patches: `$HOME/.local/easybuild/sources`
- location for build directories: `$HOME/.local/easybuild/build`
- installation prefix: `$HOME/.local/easybuild/{software,modules}`
- container recipes/images: `$HOME/.local/easybuild/containers`
- package: `$HOME/.local/easybuild/packages`
- modules tool: Lmod, module syntax: Lua
- module naming scheme: `<name>/<version><toolchain><versionsuffix>`

Configuring  via configuration files

https://easybuild.readthedocs.io/en/latest/Configuration.html

Example:

```
[config]

prefix=/home/example
```

- system-level: `/etc/easybuild.d/*.cfg`
- user-level: `$HOME/.config/easybuild/config.cfg`
- see also output of "`eb --show-default-configfiles`"
- locations specified via '`configfiles`' configuration option are also considered
- INI format with named sections
- output of "`eb --confighelp`" can be used as a starting point
- settings specified via configuration files override default settings
- settings in configuration files are overruled by settings specified via environment variables or command line options

Configuring  via environment variables

https://easybuild.readthedocs.io/en/latest/Configuration.html

- '`example`' configuration option can be specified via `$EASYBUILD_EXAMPLE`
  - `$EASYBUILD_` + upper case name of configuration option (each '`-`' becomes '`_`')
- example:

```
$ export EASYBUILD_PREFIX=/home/example
```

- settings specified via environment variables:
  - override settings in configuration files
  - are overruled by command line options

Configuring  via command line options

https://easybuild.readthedocs.io/en/latest/Configuration.html

- each configuration setting can also be specified via an option to the '`eb`' command
- example:

```
$ eb --prefix=/home/example ...
```

- command line options override any other configuration level
- usually used to enable configuration settings that correspond to 'actions'
- or to specify a particular configuration option just once

Inspecting the current  configuration

https://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

```
$ 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
```

Basic usage

https://easybuild.readthedocs.io/en/latest/Using_the_EasyBuild_command_line.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 dependencies are available using `--missing`/`-M`:

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

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

```
eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb -r
```

Installing TensorFlow from source with **one command**...

```
$ eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
```

Installing TensorFlow from source with **one command**...


```
$ eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== building and installing TensorFlow/1.13.1-foss-2019a-Python-3.7.2...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
```

Installing TensorFlow from source with **one command**...

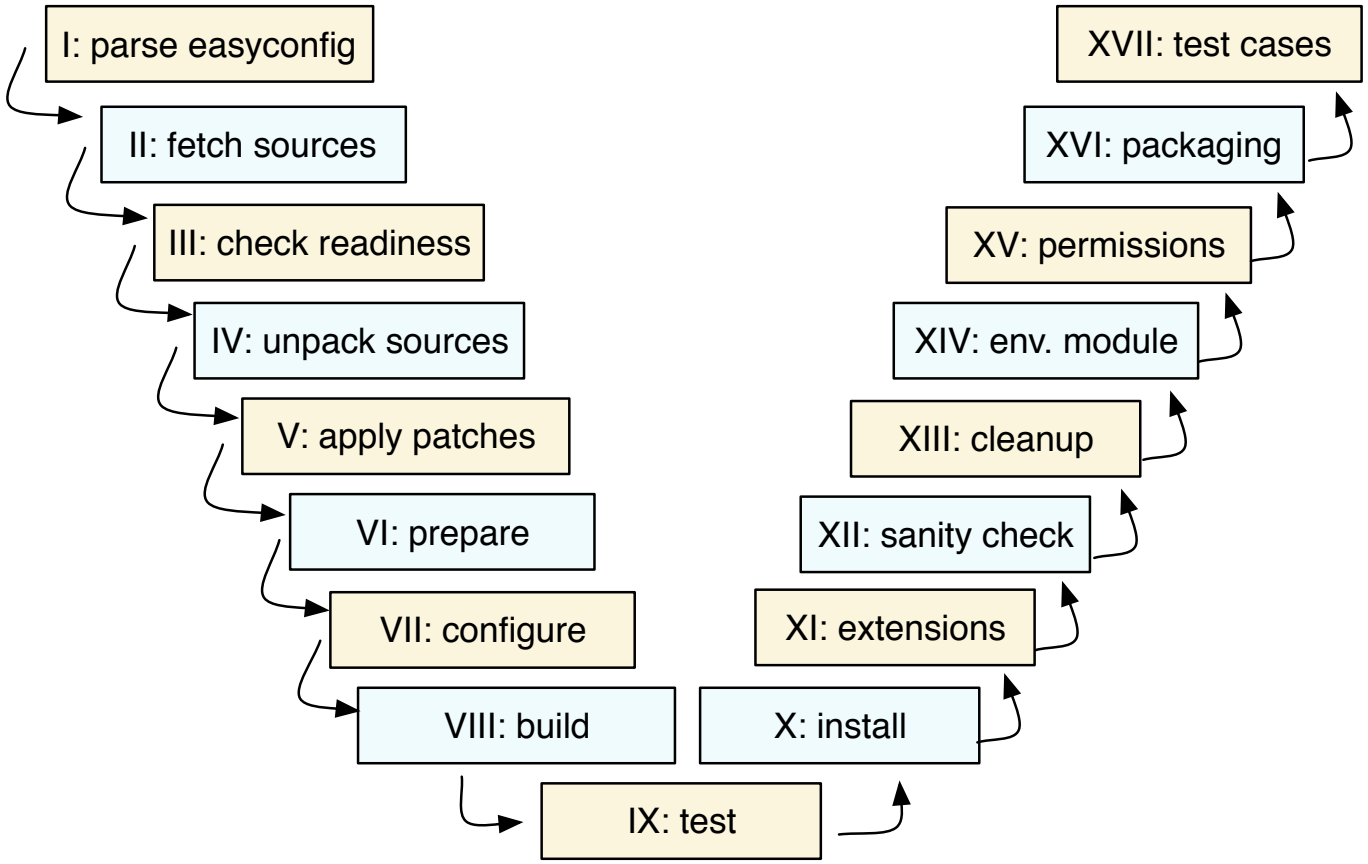

```
$ eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== building and installing TensorFlow/1.13.1-foss-2019a-Python-3.7.2...
== fetching files...
== creating build dir, resetting environment...
== unpacking...
== patching...
== preparing...
== configuring...
== building...
== testing...
== installing...
== taking care of extensions...
== postprocessing...
== sanity checking...
```

Installing TensorFlow from source with **one command**...


```
$ eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== temporary log file in case of crash /tmp/eb-GyvPHx/easybuild-U1TkEI.log
== processing EasyBuild easyconfig TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
== building and installing TensorFlow/1.13.1-foss-2019a-Python-3.7.2...
== 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-U1TkEI.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

Getting started: installing a compiler toolchain

https://easybuild.readthedocs.io/en/latest/Common-toolchains.html

- most easyconfig files use a particular compiler toolchain
- commonly used toolchains are:
  - `foss`: GCC, OpenMPI, OpenBLAS + ScaLAPACK, FFTW
  - `intel`: Intel C/C++/Fortran compilers, Intel MPI, Intel Math Kernal Library (MKL)
- compiler toolchain must be installed first (for example via "`eb --robot`")
- this may take a while, depending on available system resources...
- system compilers & MPI/BLAS/LAPACK libraries are usually not used
- tools provided via EasyBuild are strongly preferred (makes reproducing installations easier)

Typical workflow

https://easybuild.readthedocs.io/en/latest/Typical_workflow_example_with_WRF.html

- determine easyconfig file(s) to install
  - use `eb --search` to see which easyconfig files are available
  - or see https://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html
- check which required dependencies are not installed yet (no module installed yet)
  - use `eb --dry-run` or `eb --missing`
- **specify easyconfig(s) to `eb` command to install** (`--robot` to install missing deps)
- once installation is completed, use `module load` to start using the software

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

If the available easyconfig files don't exactly match your needs,

you can instruct EasyBuild to tweak an existing easyconfig file before installing it.

- tweaking the software version

```
eb example-1.2.3-foss-2019a.eb --try-software-version 2.3.4
```

- tweaking the compiler toolchain to use (can be combined with `--robot`):

```
eb example-1.2.3-foss-2018b.eb --try-toolchain-version 2019a
```

```
eb example-1.2.3-intel-2018b.eb --try-toolchain foss,2019a
```

No guarantees on success!

Transparency of performed install procedure (1)

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//$source
  * http://www.mmm.ucar.edu/wrf/src//$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 "tcsh ./compile -j 4 wrf"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsh ./compile -j 4 em_real"
  (in /home/example/eb/software/WRF/3.8.0-intel-2016b-dmpar/WRF-3.8.0)
  running command "tcsh ./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
    }
```

Transparency of performed install procedure (2)

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 files with output of each command
- timing information for each command (start time + how long it took to run)
- 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
...
```

Log files

https://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

The log is file is copied to software installation directory for future reference,
and can be used to debug 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:
...
```

Controlling architecture-specific optimisation

https://easybuild.readthedocs.io/en/latest/Controlling_compiler_optimization_flags.html

By default, EasyBuild targets the processor architecture of the build host.

- `-march=native` when using GCC
- `-xHost` when using Intel compilers
- this can be changed via the `--optarch` configuration option
  - `--optarch "<compiler flags>"`
  - `--optarch "GCC:<GCC compiler flags>;Intel:<Intel compiler flags>"`
  - `--optarch=GENERIC`

You should take this into account when creating container images via EasyBuild, since EasyBuild's default behaviour clashes with the "mobilty of compute" idea...

Easyconfig files as build specifications

https://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)

Easyconfig files as build specifications

https://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


```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Example easyconfig file

software name and version

software metadata

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Example easyconfig file

software name and version

software metadata

toolchain name & version

sources & patches

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Example easyconfig file

software name and version

software metadata

toolchain name & version

sources & patches

list of (build) dependencies

note: all versions are *fixed*!

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Example easyconfig file

software name and version

build variant (specific to WRF)
('dmpar': distributed, MPI)

software metadata

toolchain name & version

sources & patches

list of (build) dependencies
note: all versions are *fixed*!

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

Example easyconfig file

no easyblock specified, which implies using a software-specific easyblock (`EB_WRF`)

software name and version

build variant (specific to WRF)
('dmpar': distributed, MPI)

software metadata

toolchain name & version

sources & patches

list of (build) dependencies
note: all versions are *fixed*!

```
name = 'WRF'
version = '3.8.0'

buildtype = 'dmpar'  # custom parameter for WRF
versionsuffix = '-' + buildtype  # part of module name

homepage = 'http://www.wrf-model.org'
description = "Weather Research and Forecasting (WRF) Model"

toolchain = {'name': 'intel', 'version': '2016b'}

source_urls = ['http://www.mmm.ucar.edu/wrf/src/']
sources = ['%(name)sV%(version_major_minor)s.TAR.gz']
patches = ['WRF-%(version)s_known_problems.patch']

builddependencies = [('tcsh', '6.20.00')]
dependencies = [
    ('JasPer', '2.0.10'),
    ('netCDF', '4.4.1'),
    ('netCDF-Fortran', '4.4.4'),
]
```

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***
  - 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***

Common generic easyblocks

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`[1] 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`, ...)
- current 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

(1) FOSS: Free and Open Source Software

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

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

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

Making Scientific Software Installation Reproducible On Cray Systems Using EasyBuild

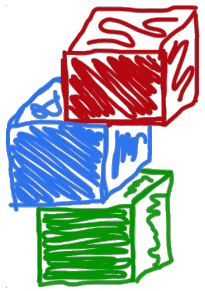
Petar Forai (IMP), Guilherme Peretti-Pezzi (CSCS), Kenneth Hoste (HPC-UGent)

https://cug.org/proceedings/cug2016_proceedings/includes/files/pap145.pdf

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



easybuild

Singularity

Questions?

Kenneth Hoste (HPC-UGent)

kenneth.hoste@ugent.be

20190612 - Singularity workshop @ HPCKP'19 (Barcelona)

https://users.ugent.be/~kehoste/EasyBuild_20190612_Singularity_workshop.pdf

https://github.com/boegel/easybuild-singularity-tutorial

https://easybuilders.github.io/easybuild

https://easybuild.readthedocs.io

Outline

- Quick introduction to 
- Installing and configuration
- Basic usage
- Beyond the basics
- *(coffee break)*
- Using  to create  ingularity container images
  - Using `eb` in Singularity container recipes
  - Generating Singularity container recipes/images via `eb`

See also EasyBuild/Singularity tutorial at
https://github.com/boegel/easybuild-singularity-tutorial

Using  in Singularity container recipes

The `eb` command can be used to install software in a Singularity container

Important things to take into account:

- requirements must be in place: Python 2, modules tool, C++ compiler, ...
- EasyBuild must be installed and properly configured in the container
- a non-root user should be used to run the `eb` command
- if Lmod is used, the Lmod cache must be updated after installing software
- `%environment` section must be set up properly to pick up installed software
- EasyBuild will optimise for host architecture by default,
  so the resulting container image will likely be not very portable

Installing  requirements in container

(note: container image is assumed to be using CentOS 7)

EasyBuild requirements can installed via `yum install` in `%post` section:

```
% post

# EPEL is required for Lmod
yum install -y epel-release
# EasyBuild requirements + pip Python installation tool
yum install -y python setuptools Lmod python-pip
# various utilities
yum install -y bzip2 gzip tar zip unzip xz patch make file git which
# C/C++ compiler (for building GCC) + Perl modules (for building Autotools)
yum install -y gcc-c++ perl-Data-Dumper perl-Thread-Queue
# OpenSSL (for CMake, Python) + Infiniband support libraries (for OpenMPI)
yum install -y rdma-core-devel
yum install -y openssl-devel
```

Installing  in container

The latest version of EasyBuild can be installed system-wide via `pip install` :

```
% post
...

# install EasyBuild using pip
pip install 'vsc-install<0.11.4' 'vsc-base<2.9.0'
pip install easybuild
```

Note: we install an older version of the `vsc-install` and `vsc-base` Python packages required by EasyBuild here to avoid problems with some additional Python packages that are required for the most recent versions of `vsc-install` and `vsc-base` .

Create dedicated user to run 'eb' command

Although you can run eb with admin privileges, it is not recommended.

(+ it requires configuring EasyBuild with `--allow-use-as-root-and-accept-consequences`)

Hence, we create a dedicated `easybuild` user.

In addition, we create two directories owned by this user:

- `/app`: will serve as installation prefix for software/modules
- `/scratch`: will serve as parent dir. for build directories, downloaded sources, ...

```
% post
...
useradd easybuild
mkdir -p /app /scratch
chown easybuild:easybuild -R /app /scratch
```

Configure Lmod in container

Lmod should be configured to use a directory in `/app` as location for its cache:

```
% post
...

# configure Lmod to use /apps/lmodcache as location for its cache
cat > /etc/lmodrc.lua << EOF
scDescriptT = {
  {
    ["dir"]       = "/app/lmodcache",
    ["timestamp"] = "/app/lmodcache/timestamp",
  },
}
EOF
```

Configuring  in container

Before installing software, we must properly configure EasyBuild, after switching to the `easybuild` user that will be used to run the '`eb`' command:

```
% post
...

# change to 'easybuild' user
su - easybuild

# use /scratch as general prefix, used for sources, build directories, etc.
export EASYBUILD_PREFIX=/scratch

# install software & modules into /app/{software,modules}
export EASYBUILD_INSTALLPATH=/app
```

Installing software in container using  easybuild

To install software, we simply use the `eb` command with the `--robot` option, and specify one or more easyconfig files:

```
% post
...

# change to 'easybuild' user
su - easybuild

# configure EasyBuild
...

eb TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb --robot
```

The software + accompanying modules will be installed in `/app` in the container.

Updating the Lmod cache after installing software

Although not strictly needed, we should update the Lmod cache after installing software to ensure a good user experience:

```
% post
...

# change to 'easybuild' user
su - easybuild

# configure EasyBuild and install software
...

# update Lmod cache
mkdir -p /app/lmodcache
$LMOD_DIR/update_lmod_system_cache_files -d /app/lmodcache
-t /app/lmodcache/timestamp /app/modules/all
```

Cleaning things up...

To conclude the post section, we clean things up in /scratch (anything still there is assumed to be no longer needed):

```
% post
...

# exit from 'easybuild' user
exit

# cleanup, everything in /scratch is assumed to be temporary
rm -rf /scratch/*
```

Setting up the environment in the container

The environment in the container should be properly set up:

- Lmod must be properly initialised (?)
- any (loaded) modules from outside the container should be discarded
- the modules for the installed software should be loaded
  such that the software is ready for use

This is needed to ensure that things work properly
when the container image is used via `singularity shell/exec/run`.

Customising the `%environment` section

```
%environment

# make sure that 'module' and 'ml' commands are defined
source /etc/profile

# purge any modules that may be loaded outside container
module --force purge
# avoid picking up modules from outside of container
module unuse $MODULEPATH

# pick up modules installed in /app
module use /app/modules/all
# load module(s) corresponding to installed software
module load TensorFlow/1.13.1-foss-2019a-Python-3.7.2
```

Can't we make this... easier?

Generating container recipes/images using 

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

Container recipes can be *generated* by using `eb --containerize` ,
which is a lot more convenient than having to puzzle them together manually...

Support for Singularity containers was significantly improved in EasyBuild v3.9.2,
but it is still an **experimental feature**

- requires configuring EasyBuild with `--experimental` ,
  or setting the `$EASYBUILD_EXPERIMENTAL` environment variable
- this functionality may change in future EasyBuild releases (not stable yet)
- **feedback, bug reports, suggestions for enhancements are very welcome!**

Container support: the basics

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

experimental feature

- `eb --containerize` will generate a **container recipe**
  (rather than actually performing any installations)
- Singularity is used by default, Docker also supported
- if `--container-build-image` is specified, **container image** will be built too
  (required `sudo` privileges!)
- **container configuration** can be controlled via `--container-config`
- container recipes & images are created in the "container path"
  (see output of `eb --show-config`)

Building container from scratch

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

experimental feature

To build a container from scratch, you can use a distro bootstrap agent:

```
bootstrap={arch,busybox,debootstrap,yum,zypper}
```

Note: currently only `yum` is well supported (due to hardcoded `yum install` commands).

A couple more keywords are supported for further specify things:

- `osversion`: operating system version
- `mirrorurl`: URI to download Linux distro from
  (default is CentOS with `bootstrap=yum`)
- `include`: additional OS packages to install
  (`yum` is default with `bootstrap=yum`)

Using an existing container image as base

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

experimental feature

To build a container image using an existing one as a base,

you can use an image-based bootstrap agent:

```
bootstrap={docker,library,localimage,shub}
```

- **`localimage`: locally available container image**
- `docker`: download container image from Docker Hub
- `library`: download container image from Sylabs Container Library
- `shub`: download container image from Singularity Hub

Using either of these requires to also specify the `from` keyword to specify the container image to use.

Example: TensorFlow in CentOS 7 container

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

experimental feature

```
# configure EasyBuild to allow using experimental features
export EASYBUILD_EXPERIMENTAL=1
# specify container configuration (CentOS 7 from scratch)
export EASYBUILD_CONTAINER_CONFIG="bootstrap=yum,osversion=7"
# instruct EasyBuild to create container recipe that installs
# TensorFlow (incl. all dependencies)
eb --containerize TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
# build container image
sudo singularity build TF.sif $HOME/containers/*TensorFlow*
```

Example: stacking container images (step 1/3)

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

experimental feature

Build a container image for the `foss/2019a` toolchain,
with the intention to use it as a base for other container images:

```
# configure EasyBuild to allow using experimental features
export EASYBUILD_EXPERIMENTAL=1
# specify container configuration (CentOS 7 from scratch)
export EASYBUILD_CONTAINER_CONFIG="bootstrap=yum,osversion=7"
# build container image for foss/2019a toolchain
eb --containerize foss-2019a.eb --container-build-image
```

Example: stacking container images (step 2/3)

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

experimental feature

Build a container image for `SciPy-bundle` built with the `foss/2019a` toolchain, using `foss-2019a.sif` as a base.

```
# specify container configuration: use foss-2019a.sif as a base
export EASYBUILD_CONTAINER_CONFIG="bootstrap=localimage,
from=/home/example/containers/foss-2019a.sif"
# build container image for SciPy-bundle
export EASYBUILD_EXPERIMENTAL=1 EASYBUILD_CONTAINER_BUILD_IMAGE=1
eb --containerize SciPy-bundle-2019.03-foss-2019a.eb
```

Example: stacking container images (step 3/3)

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

experimental feature

Build a container image for `TensorFlow` using `SciPy-bundle-2019.03-foss-2019a.sif` as a base.

```
# specify container configuration: use SciPy-bundle image as a base
export EASYBUILD_CONTAINER_CONFIG="bootstrap=localimage,
from=/home/example/containers/SciPy-bundle-2019.03-foss-2019a.sif"
# build container image for TensorFlow
export EASYBUILD_EXPERIMENTAL=1 EASYBUILD_CONTAINER_BUILD_IMAGE=1
eb --containerize TensorFlow-1.13.1-foss-2019a-Python-3.7.2.eb
```

Seeding in source files

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

experimental feature

- for some installations, source files may need to be provided
- automatic download failed, or is simply not supported (licensing issues, ...)
- container recipes generated via EasyBuild consider `/tmp/easybuild/sources/` as a fallback path in the "source path"
- can be used to 'seed' source files from host into container
- required for Java, which is a dependency of Bazel (TensorFlow installation tool):

```
cp jdk-8u212-linux-x64.tar.gz /tmp/easybuild/sources/
```

Can we think of something better?

Known limitations/issues

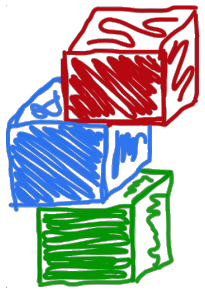
https://easybuild.readthedocs.io/en/latest/Containers.html

experimental feature

https://github.com/easybuilders/easybuild-framework/labels/containers

There are a couple of known limitations/issues with the current implementation:

- hardcoded `yum install` commands (regardless of Linux distro being used)
- no container metadata (description/labels)
- access to EasyBuild log files for failing installations is still problematic
- no effort was made yet to limit the size of the container images
  (removing build-only dependencies, zipping installation logs, ...)
- can we track for which processor architecture the container image was built?



easybuild

Singularity

Questions?

Kenneth Hoste (HPC-UGent)

kenneth.hoste@ugent.be

20190612 - Singularity workshop @ HPCKP'19 (Barcelona)

https://users.ugent.be/~kehoste/EasyBuild_20190612_Singularity_workshop.pdf

https://github.com/boegel/easybuild-singularity-tutorial

https://easybuilders.github.io/easybuild

https://easybuild.readthedocs.io