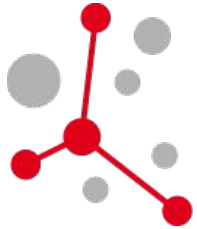


ISC High Performance
The HPC Event.

EasyBuild tutorial
ISC'21

Kenneth Hoste (HPC-UGent) - Alan O'Cais (JSC) - Bart Oldeman (Compute Canada)

June 25th 2021

https://easybuild.io/tutorial/isc21

Agenda

(all times are UTC)

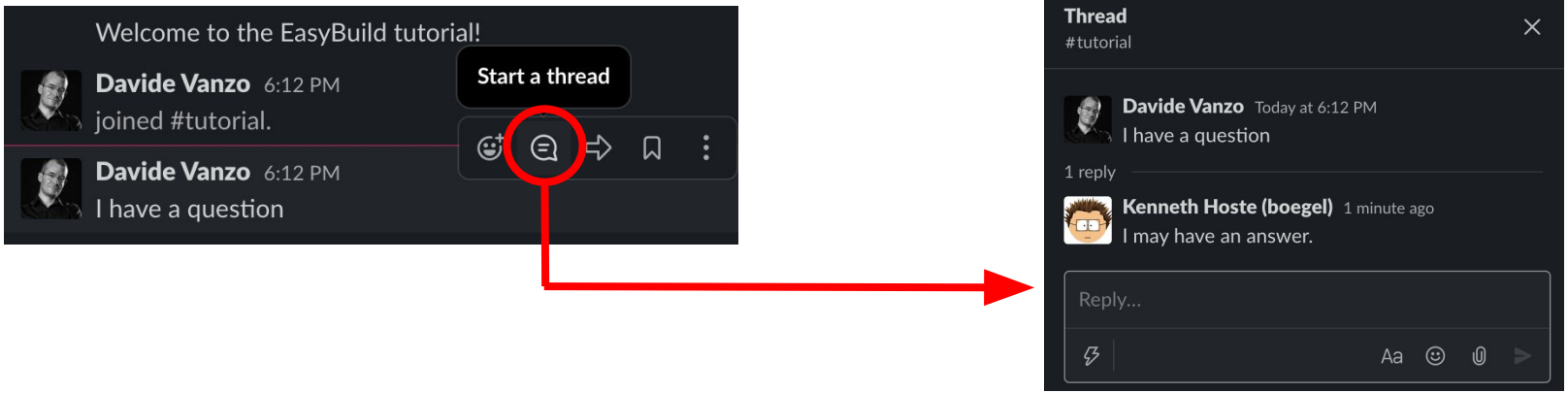
- [12:00-12:10] **Practical information w.r.t. prepared environment for hands-on examples**
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

Practical information

- **Friday June 25th 2021, 12:00 - 16:00 UTC**
- Tutorial website: https://easybuild.io/tutorial/isc21
- Please join the `#tutorial-isc21` channel in the EasyBuild Slack to ask questions!
- Prepared environment for hands-on demos & exercises

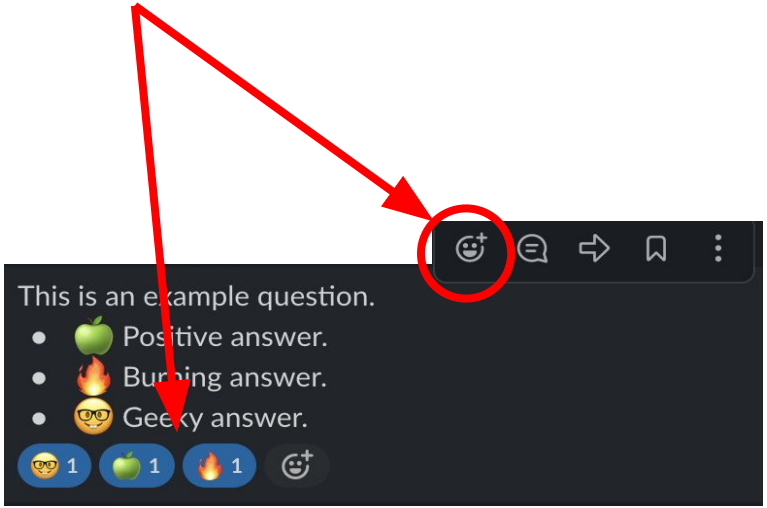
Q&A via dedicated channel in EasyBuild Slack

- Questions or problems?
  Speak up in `#tutorial-isc21` on EasyBuild Slack!
- **Join via https://easybuild.io/join-slack**
- **Use threads to avoid overflowing the channel!**



Emoji polls in Slack

- Small polls will be posted in the `#tutorial-isc21` Slack channel.
- **Vote** for one (or more) answers using the corresponding emoji !



Prepared environment

- Small CentOS 7 cluster (in the cloud)

- **You need to create an account!**
  - Signup: https://mokey.isc21.learnhpc.eu/auth/signup
  - Accounts will only be approved for access on 24/25 June 2021, so **please record your username/password !**

- Access via `ssh` or web browser:
  - Shell access: `ssh isc21.learnhpc.eu`
  - Via browser: https://isc21.learnhpc.eu

- System will be up until the end of the conference (18:15 CEST, Friday 2 July 2021)

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] **Introduction to EasyBuild: scope & terminology**
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

What is EasyBuild?

- **EasyBuild is a software build and installation framework**
- Strong focus on scientific software, performance, and HPC systems
- Open source (GPLv2), implemented in Python (2.7, 3.5+)
- Brief history:
  - Created in-house at HPC-UGent in 2008
  - First released publicly in Apr'11
  - EasyBuild 1.0 released in Nov'11 (during SC11)
  - Worldwide community has grown around it since then!

https://easybuild.io

https://docs.easybuild.io

https://github.com/easybuilders

https://easybuild.slack.com
(https://easybuild.io/join-slack)

Twitter: @easy_build

EasyBuild in a nutshell

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

Key features of EasyBuild (1/2)

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

Key features of EasyBuild (2/2)

- Integrates with various other tools (Lmod, Singularity, FPM, Slurm, GC3Pie, ...)
- **Actively developed and supported by worldwide community**
- **Frequent stable releases** since 2011 (every 6 - 8 weeks)
- **Comprehensive testing**: unit tests, testing contributions, regression testing
- **Various support channels** (mailing list, Slack, conf calls) + yearly user meetings

Focus points in EasyBuild

Performance

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

Reproducibility

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

Community effort

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

What EasyBuild is *not*

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

EasyBuild terminology

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

EasyBuild terminology: framework

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

EasyBuild terminology: easyblock

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

EasyBuild terminology: easyconfig file

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

EasyBuild terminology: easystack file

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

EasyBuild terminology: extensions

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

EasyBuild terminology: dependencies

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

EasyBuild terminology: toolchains

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

EasyBuild terminology: modules

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

Bringing all EasyBuild terminology together

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

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

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

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] **Installing & configuring EasyBuild + basic usage**
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

Installing EasyBuild: requirements

- **Linux** as operating system (CentOS, RHEL, Ubuntu, Debian, SLES, ...)
  - EasyBuild also works on macOS, but support is very basic
- **Python** 2.7 or 3.5+
  - Only Python standard library is required for core functionality of EasyBuild
  - Using Python 3 is highly recommended!
- An **environment modules tool** (`module` command)
  - Default is Lua-based Lmod implementation, highly recommended!
  - Tcl-based implementations are also supported

Installing EasyBuild: different options

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

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

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

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

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

Installing EasyBuild as a module (recommended)

3-step bootstrap procedure

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

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

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

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

Verifying the EasyBuild installation

- Check EasyBuild version:

```
eb --version
```

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

```
eb --help
```

- Show the current (default) EasyBuild configuration:

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

- Show system information:

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

Updating EasyBuild

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

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

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

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

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

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

```
module load EasyBuild/4.4.0
```

Configuring EasyBuild

- EasyBuild should work fine out-of-the-box if you are using Lmod as modules tool
- … but it will (ab)use `$HOME/.local/easybuild` to install software into, etc.
- It is ***strongly*** recommended to configure EasyBuild properly!
- Main questions you should ask yourself:
  - Where should EasyBuild install software (incl. module files)?
  - Where should auto-downloaded sources be stored?
  - Which filesystem is best suited for software build directories (I/O-intensive)?

Primary configuration settings

- Most important configuration settings: (strongly recommended to specify the ones in **bold**!)
  - Modules tool + syntax (`modules-tool` + `module-syntax`)
  - **Software + modules installation path** (`installpath`)*
  - **Location of software sources "cache"** (`sourcepath`)*
  - **Parent directory for software build directories** (`buildpath`)*
  - Location of easyconfig files archive (`repositorypath`)*
  - Search path for easyconfig files (`robot-paths` + `robot`)
  - Module naming scheme (`module-naming-scheme`)
- Several locations* (+ others) can be controlled at once via `prefix` configuration setting
- *Full* list of EasyBuild configuration settings (~250) is available via `eb --help`

Configuration levels

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

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

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

EasyBuild configuration files

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

$EASYBUILD_* environment variables

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

Command line options for `eb` command

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

Inspecting the current configuration

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

Inspecting the current configuration: example


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

$ export EASYBUILD_BUILDPATH=/tmp/$USER/build

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

Minimal EasyBuild configuration for hands-on

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

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

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

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

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

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

  ```
  module use /easybuild/modules/all
  ```

Basic usage of EasyBuild

- **Use `eb` command to run EasyBuild**
- Software to install is usually specified via name(s) of easyconfig file(s), or easystack file
- `--robot` (`-r`) option is required to also install missing dependencies (and toolchain)
- Typical workflow:
  - Find or create easyconfig files to install desired software
  - Inspect easyconfigs, check missing dependencies + planned installation procedure
  - Double check current EasyBuild configuration
  - Instruct EasyBuild to install software (while you enjoy a coffee… or two)

Specifying easyconfigs to use

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

Inspecting easyconfigs via `eb --show-ec`

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

```
$ eb --show-ec TensorFlow-2.4.1-foss-2020b.eb
easyblock = 'PythonBundle'

name = 'TensorFlow'
version = '2.4.1'

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

toolchain = {'name': 'foss', 'version': '2020b'}
toolchainopts = {'pic': True}
…
```

Checking dependencies via `eb --dry-run`

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

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

```
$ eb SAMtools-1.11-GCC-10.2.0.eb -D
 ...
 * [ ] $CFGS/x/XZ/XZ-5.2.5-GCCcore-10.2.0.eb (module: XZ/5.2.5-GCCcore-10.2.0)
 * [ ] $CFGS/c/cURL/cURL-7.72.0-GCCcore-10.2.0.eb (module: cURL/7.72.0-GCCcore-10.2.0)
 * [x] $CFGS/g/GCC/GCC-10.2.0.eb (module: GCC/10.2.0)
 * [x] $CFGS/n/ncurses/ncurses-6.2-GCCcore-10.2.0.eb (module: ncurses/6.2-GCCcore-10.2.0)
 * [ ] $CFGS/s/SAMtools/SAMtools-1.11-GCC-10.2.0.eb (module: SAMtools/1.11-GCC-10.2.0)
```

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

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

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

```
$ eb h5py-3.1.0-foss-2020b.eb -M

2 out of 61 required modules missing:

* pkg-config/0.29.2-GCCcore-10.2.0 (pkg-config-0.29.2-GCCcore-10.2.0.eb)

* h5py/3.1.0-foss-2020b (h5py-3.1.0-foss-2020b.eb)
```

Inspecting software install procedures

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

Inspecting software install procedures: example


```
$ eb Boost-1.74.0-GCC-10.2.0.eb -x
...

preparing... [DRY RUN]

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

Loading toolchain module...

module load GCC/10.2.0

Loading modules for dependencies...

module load bzip2/1.0.8-GCCcore-10.2.0
module load zlib/1.2.11-GCCcore-10.2.0
module load XZ/5.2.5-GCCcore-10.2.0
```

Inspecting software install procedures: example


```
$ eb Boost-1.74.0-GCC-10.2.0.eb -x
...
Defining build environment...
  ...
  export CXX='mpicxx'
  export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno -fPIC'
  ...

configuring... [DRY RUN]

[configure_step method]
  running command "./bootstrap.sh --with-toolset=gcc
  --prefix=/tmp/example/Boost/1.74.0/GCC-10.2.0/obj --without-libraries=python,mpi"
  (in /tmp/example/build/Boost/1.74.0/GCC-10.2.0/Boost-1.74.0)
```

Inspecting software install procedures: example


```
$ eb Boost-1.74.0-GCC-10.2.0.eb -x
...

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

...
```

Agenda

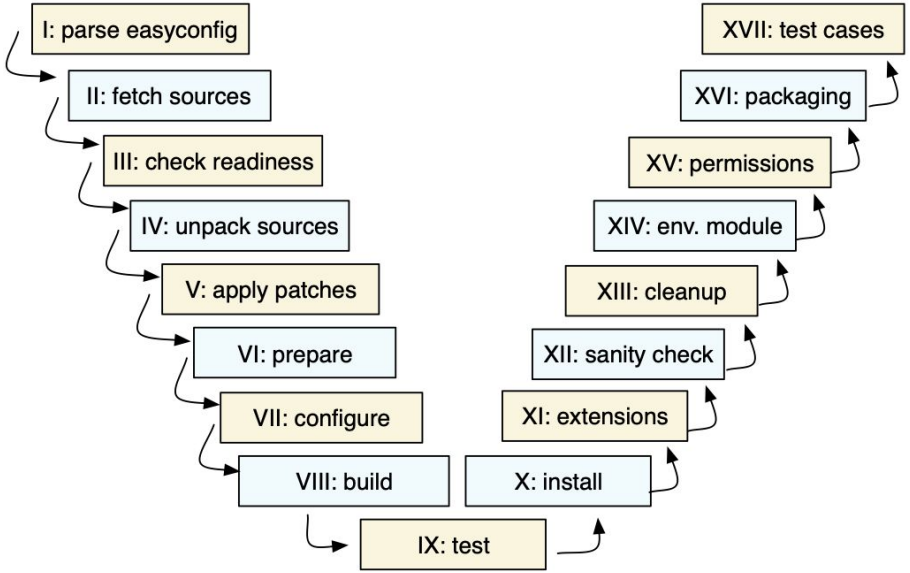
(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] **Installing software with EasyBuild + troubleshooting**
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

Installing software with EasyBuild

- To install software with EasyBuild, just run the `eb` command:
  - `eb SAMtools-1.11-GCC-10.2.0.eb`
- If any dependencies are still missing, you will need to also use `--robot`:
  - `eb BCFtools-1.11-GCC-10.2.0.eb --robot`
- To see more details while the installation is running, enable trace mode:
  - `eb BCFtools-1.11-GCC-10.2.0.eb --robot --trace`
- To reinstall software, use `eb --rebuild` (or `eb --force`)

Step-wise installation procedure



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

Using software installed with EasyBuild

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

```
# inform modules tool about modules installed with EasyBuild
module use $HOME/easybuild/modules/all
# check for available modules for BCFtools
module avail BCFtools
# load BCFtools module to "activate" the installation
module load BCFtools/1.11-GCC-10.2.0
```

Stacking software installations

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

```
module use /easybuild/modules/all

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

Troubleshooting failing installations

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

Troubleshooting: error messages

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

```
== building...
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/example/example/1.0/GCC-10.2.0):
build failed (first 300 chars): cmd "make" exited with exit code 2 and output:
/usr/bin/g++ -O2 -ftree-vectorize -march=native -std=c++14 -c -o core.o core.cpp
g++: error: unrecognized command line option '-std=c++14' (took 1 sec)
```

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

Troubleshooting: log files

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

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

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

Troubleshooting: navigating log files

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

```
== 2021-06-25 13:11:19,968 run.py:222 INFO running cmd:  make -j 9
```

- Different steps of installation procedure are clearly marked:

```
== 2021-06-25 13:11:48,817 example INFO Starting sanity check step
```

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

Troubleshooting: inspecting the build directory

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

```
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/build/example/1.0/GCC-10.2.0): build failed ...
```

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

Troubleshooting: hands-on exercise

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

```
$ eb subread.eb
...
== FAILED: Installation ended unsuccessfully (build directory:
/tmp/example/Subread/2.0.1/GCC-8.5.0): build failed (first 300 chars):
Couldn't find file subread-2.0.1-source.tar.gz anywhere, and downloading
it didn't work either...
Paths attempted (in order): ...
```

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] ***(coffee break)***
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

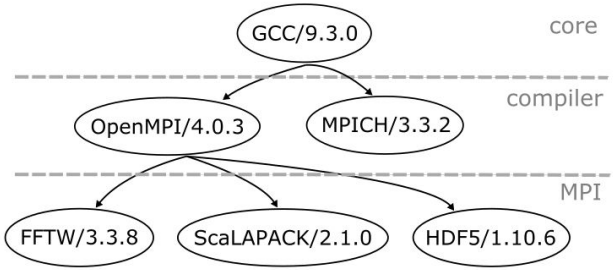
Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] **Module naming schemes (incl. hierarchical)**
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

Flat vs hierarchical module naming schemes

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



Pros and cons of using a flat vs hierarchical MNS

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

Custom module naming schemes with EasyBuild

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

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

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

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

```
eb SciPy-bundle-2020.11-foss-2020b-Python-2.7.18.eb -D
```

Hands-on example: installing HDF5 in an HMNS

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

```
module unuse $MODULEPATH
```

- Configure our setup to reuse the existing software installations

```
export EASYBUILD_INSTALLPATH_SOFTWARE=/easybuild/software

export EASYBUILD_MODULE_NAMING_SCHEME=HierarchicalMNS

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

- Re-generate all modules for HDF5 using the new scheme (41 modules)

```
eb HDF5-1.10.7-gompi-2020b.eb --robot --module-only
```

- Explore the new hierarchy

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

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] **Adding support for additional software**
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

Adding support for additional software

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

Easyblocks vs easyconfigs

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

Writing easyconfig files

- Collection of easyconfig parameter definitions (Python syntax), collectively specify what to install
- Some easyconfig parameters are mandatory, and **must** always be defined:
  `name`, `version`, `homepage`, `description`, `toolchain`
- Commonly used easyconfig parameters (but strictly speaking not required):
  - `easyblock` (by default derived from software name)
  - `source_urls`, `sources`, `patches`, `checksums`
  - `dependencies`, `builddependencies`
  - `preconfigopts`, `configopts`, `prebuildopts`, `buildopts`, `preinstallopts`, `installopts`
  - `sanity_check_paths`, `sanity_check_commands`

Generating tweaked easyconfig files

- Trivial changes to existing easyconfig files can be done automatically
- Bumping software version: `eb example-1.0.eb --try-software-version 1.1`
- Changing toolchain (version): `eb example.eb --try-toolchain GCC,9.4.0`
- Changing specific easyconfig parameters (limited): `eb --try-amend ...`
- Note the “try” aspect: additional changes may be required to make installation work

Copying easyconfig files

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

```
$ eb --copy-ec SAMtools-1.11-GCC-10.2.0.eb SAMtools.eb
...
SAMtools-1.10-GCC-10.2.0.eb copied to SAMtools.eb
```

Hands-on: creating easyconfig files

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

```
name = 'eb-tutorial'

version = '1.0.1'

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

description = "EasyBuild tutorial example"
```

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] **Use of EasyBuild in large scale production systems at JSC and Compute Canada**
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

EasyBuild at Jülich Supercomputing Centre

JÜLICH
Forschungszentrum

by Alan O'Cais

Jülich Supercomputing Centre

- JSC is a German supercomputing centre since 1987
  - About 200 experts for all aspects of supercomputing and simulation sciences

JÜLICH Forschungszentrum

Jülich Supercomputing Centre

- JSC is a German supercomputing centre since 1987
  - About 200 experts for all aspects of supercomputing and simulation sciences
- Currently 3 primary systems:
  - JUWELS - 73 Petaflops, #7 in Top500 (modular supercomputing)
  - JURECA-DC - 3.54 (CPU) + 14.98 (GPU) + 5 (KNL) Petaflops
  - JUSUF - AMD, V100 GPU. Interactive workflows and community services

JÜLICH Forschungszentrum

EasyBuild at JSC

- Used for production software stack at JSC since 2014

JÜLICH Forschungszentrum

EasyBuild at JSC

- Used for production software stack at JSC since 2014
- Geared towards average user experience
  - Hide lots of indirect software
  - Lots of toolchains => Module hierarchy
  - Renaming some modules, Lmod tweaks

EasyBuild at JSC

- Used for production software stack at JSC since 2014
- Geared towards average user experience
  - Hide lots of indirect software
  - Lots of toolchains => Module hierarchy
  - Renaming some modules, Lmod tweaks
- Custom MNS, toolchains, easyconfigs, easyblocks
  - Maintenance and contribution issue
  - Working hard to minimise this

JÜLICH Forschungszentrum

Upgrading and retiring software

- Provide latest software to new projects by default
  - ***Stages*** concept
  - Updates once per year
  - Encourages users to adopt latest software & dependencies (performance, bug fixes,...)

Upgrading and retiring software

- Provide latest software to new projects by default
  - ***Stages*** concept
  - Updates once per year
  - Encourages users to adopt latest software & dependencies (performance, bug fixes,...)
- Give indirect access to "retired" software

Leveraging hooks for users & maintainers

- Very powerful alternative to customisations
  - Much more automated and flexible
  - Easier to maintain (particularly for easyconfigs)

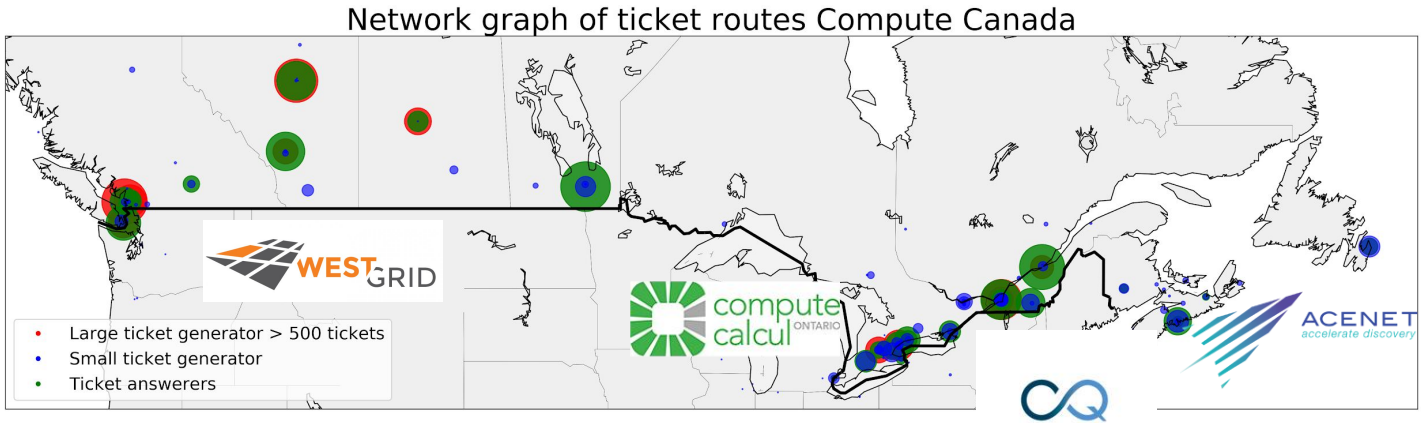
Leveraging hooks for users & maintainers

- Very powerful alternative to customisations
  - Much more automated and flexible
  - Easier to maintain (particularly for easyconfigs)
- Hooks to enable user space installations
  - Guide people on how to do this “properly”
  - Installation hierarchy: system → group → user

EasyBuild at Compute Canada

- by Bart Oldeman

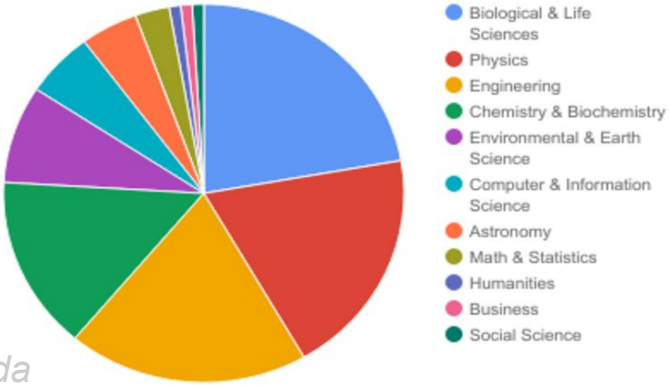
Compute Canada : the people



All research disciplines supported

Free access for any researcher at a Canadian institution

- 4 regional consortia
- 35 member institutions
- ~200 technical staff
- ~15,000 user accounts
  - 20% growth per year



Compute Canada : the hardware

5 major national systems
~15 legacy systems
270K cores, 2500 GPUs,
70 PB disk, 180 PB tape

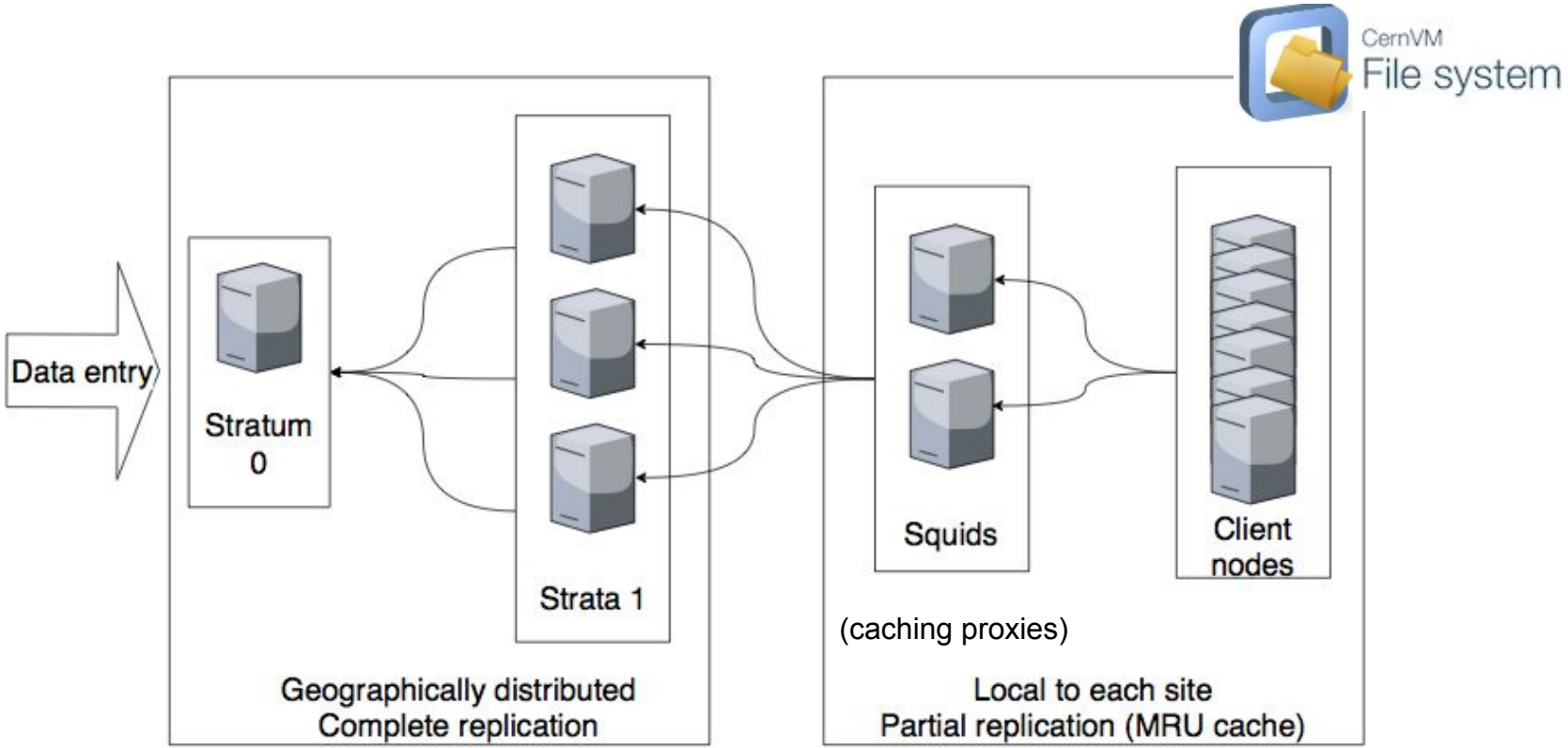
System	Type	Network	Production
Arbutus	Cloud	10 GbE	2016 H2
Cedar	General	OPA	2017 H1
Graham	General	EDR IB	2017 H1
Niagara	Large MPI	EDR IB	2018 H1
Béluga	General	EDR IB	2019 H1

Goal

Users should be presented with an interface that is as consistent and easy to use as possible across all sites. It should also offer optimal performance.

- Accessible on every site, reliably and performantly: need a distribution mechanism
  - CernVM-FS : CERN Virtual Machine File System
- Independent of the OS (Ubuntu, CentOS, Fedora, etc.)
  - Gentoo Prefix (used to be Nix)
- Automated, tracked, reproducible installation (humans are not so consistent)
  - **EasyBuild**
- Needs a module interface that scales well
  - Lmod with a hierarchical structure

CernVM-FS content delivery



Software: design overview

Easybuild layer: modules for Intel, NVHPC, OpenMPI, CUDA, MKL, high-level applications. Multiple architectures (sse3, avx, avx2, avx512)
`/cvmfs/soft.computecanada.ca/easybuild/{modules,software}/` ~~2017~~2020

Easybuild-generated modules around Nix profiles (GONE):
~~GCC, Eclipse, Qt+Perl+Python no longer~~
~~`/cvmfs/soft.computecanada.ca/nix/var/nix/profiles/[a-z]*`~~

Compatibility: ~~Nix~~ Gentoo Prefix layer: GNU libc, autotools, make, bash, cat, ls, awk, grep, etc.
`module gentoo/2020 =>`
`$EPREFIX=/cvmfs/soft.computecanada.ca/gentoo/2020, $EBROOTGENTOO=$EPREFIX/usr`

Gray area: ~~Slurm,~~ Lustre client libraries, IB/OmniPath/InfiniPath client libraries (all dependencies of OpenMPI). In Gentoo layer, but can be overridden using PATH & LD_LIBRARY_PATH.

OS kernel, daemons, drivers, libcuda, anything privileged (e.g. the sudo command): always local. Some legally restricted software too (VASP)

Compute Canada Software Stack

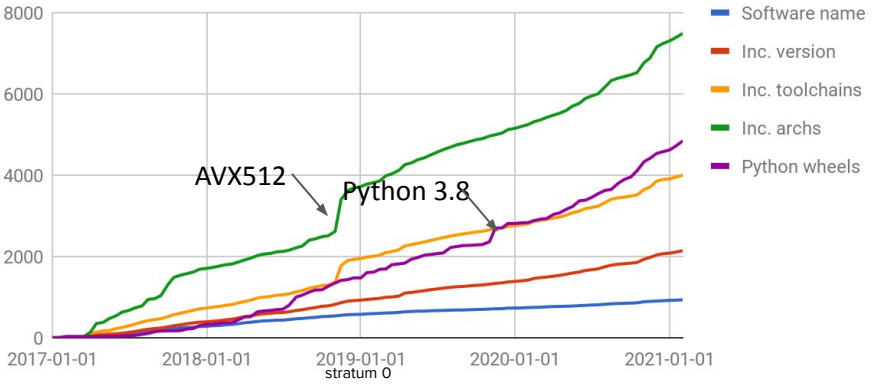
Available software

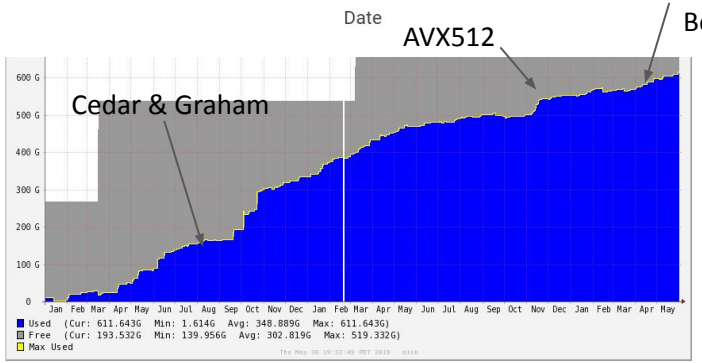
800+ scientific applications

6,000+ permutations of version/arch/toolchain

Type	Modules
AI	5
Bioinformatics	239
Chemistry	63
Data	19
Geo/Earth	23
Mathematics	82
MPI libraries	7
Physics	48
Various tools	176
Visualisation	28
Misc	38

Number of software packages available through modules and python wheels





600GB[90]

- Two major new clusters with Skylake CPUs
- Built new modules with AVX512 for most packages
- High deduplication
- Further details

https://easybuilders.github.io/easybuild-tutorial/2021-isc21/computecanada

Design choices / EasyBuild features

- Compatibility layer => filtering of a lot of dependencies (M4, cmake, etc.)
- Toolchains based combinations of
  - Intel/GCC, OpenMPI, MKL, Cuda
  - => gomkl(c)/ iomkl(c) toolchains
  - => We are (ab)using the --try-toolchain, --try-software-version, --try-update-deps
- Custom module naming scheme:
  - Hierarchical, lower case
  - No versionsuffix at all
  - Toolchains are hidden
- No $LD_LIBRARY_PATH, instead RPATH using wrapper for linker (ld).

Hooks

- Injecting custom configuration options for OpenMPI
- Injecting footer code in compiler and MPI modules to support installation in user's home directories
- Splitting the installation of Intel into redistributable and non-redistributable parts
- Stripping down Python modules (dropping extensions)

Handling Python

- Installing Python wrappers and side packages (PyQt5 with Qt5, OpenCV-python with OpenCV, etc.) whenever possible
- Using `multi_deps` so that modules are compatible with all versions of Python
- Not installing most Python packages as modules, but by providing Python wheels to users they can install using `pip` in virtual environments
- Not supporting Anaconda

You can use this too!

Mounting our software stack:

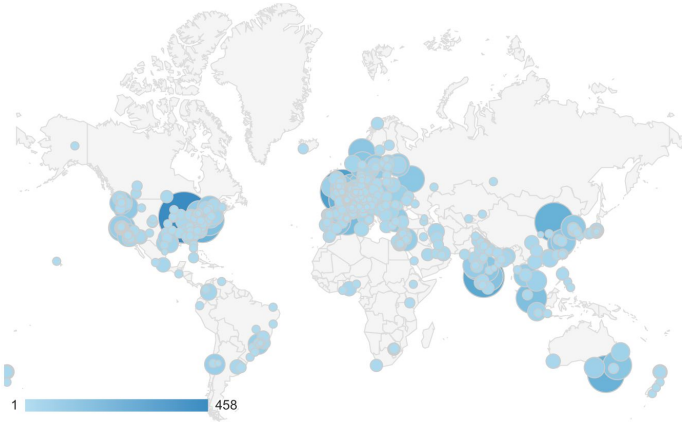
https://docs.computecanada.ca/wiki/Accessing_CVMFS

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] **The EasyBuild community + contributing to EasyBuild**
- [15:45-16:00] Q&A + closing remarks (incl. quick comparison with other tools)

The EasyBuild community



easybuild

- Documentation read all over the world
- HPC sites, consortia, and companies
- Slack: >450 members, ~100 active members per week, 226K messages
- Regular online conf calls...and we even meet in person sometimes!

https://easybuilders.github.io/easybuild-tutorial/2021-isc21/community

Contributing to EasyBuild

There are several ways to contribute to EasyBuild, including:

- providing feedback
- reporting bugs
- joining the discussions (mailing list, Slack, conf calls)
- sharing suggestions/ideas for enhancements & additional features
- contributing easyconfigs, enhancing easyblocks,

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

GitHub integration features

- EasyBuild has strong integration with GitHub, which facilitates contributions
- Some additional Python packages required for this: GitPython, keyring
- Also required some additional configuration, incl. providing a GitHub token
- **Enables creating, updating, reviewing pull requests using `eb` command!**
- Makes testing contributions very easy (~2,000 easyconfig pull requests per year!)
- Extensively documented:
  https://docs.easybuild.io/en/latest/Integration_with_GitHub.html

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`

Agenda

(all times are UTC)

- [12:00-12:10] Practical information w.r.t. prepared environment for hands-on examples
- [12:10-12:30] Introduction to EasyBuild: scope & terminology
- [12:30-13:00] Installing & configuring EasyBuild + basic usage
- [13:00-13:45] Installing software with EasyBuild + troubleshooting
- [13:45-14:15] *(coffee break)*
- [14:15-14:45] Module naming schemes (incl. hierarchical)
- [14:45-15:10] Adding support for additional software
- [15:10-15:30] Use of EasyBuild in large scale production systems at JSC and Compute Canada
- [15:30-15:45] The EasyBuild community + contributing to EasyBuild
- [15:45-16:00] **Q&A + closing remarks (incl. quick comparison with other tools)**

Topics we didn't cover...

- Implementing easyblocks
- Using RPATH linking
- Using EasyBuild as a library
- Implementing hooks to customize EasyBuild
- Submitting installations as jobs on a cluster
- Integration with the Cray Programming Environment
- Building Docker/Singularity container images with EasyBuild (experimental)

https://docs.easybuild.io - https://easybuild.io/tutorial

- **EasyBuild: GPLv2 license - Spack: MIT/Apache 2.0 license**
- no stable releases yet for Spack (< 1.0), EasyBuild is stable since 2012
- roughly on par w.r.t. amount of supported software (but differences w.r.t. which software)
- **targeted to different use cases: HPC support teams (EasyBuild) vs developers (Spack)**
- **fixed dependency/toolchain versions in EasyBuild vs flexible CLI in Spack**
- both support running on top of Python 2.7 and 3.5+
- macOS support in EasyBuild is limited (no toolchains/testing for macOS)
- **both projects are backed by an active & supportive community!**
- For a more detailed (but somewhat outdated) comparison, see https://archive.fosdem.org/2018/schedule/event/installing_software_for_scientists

Just one more thing...

https://www.eessi-hpc.org *https://eessi.github.io/docs*

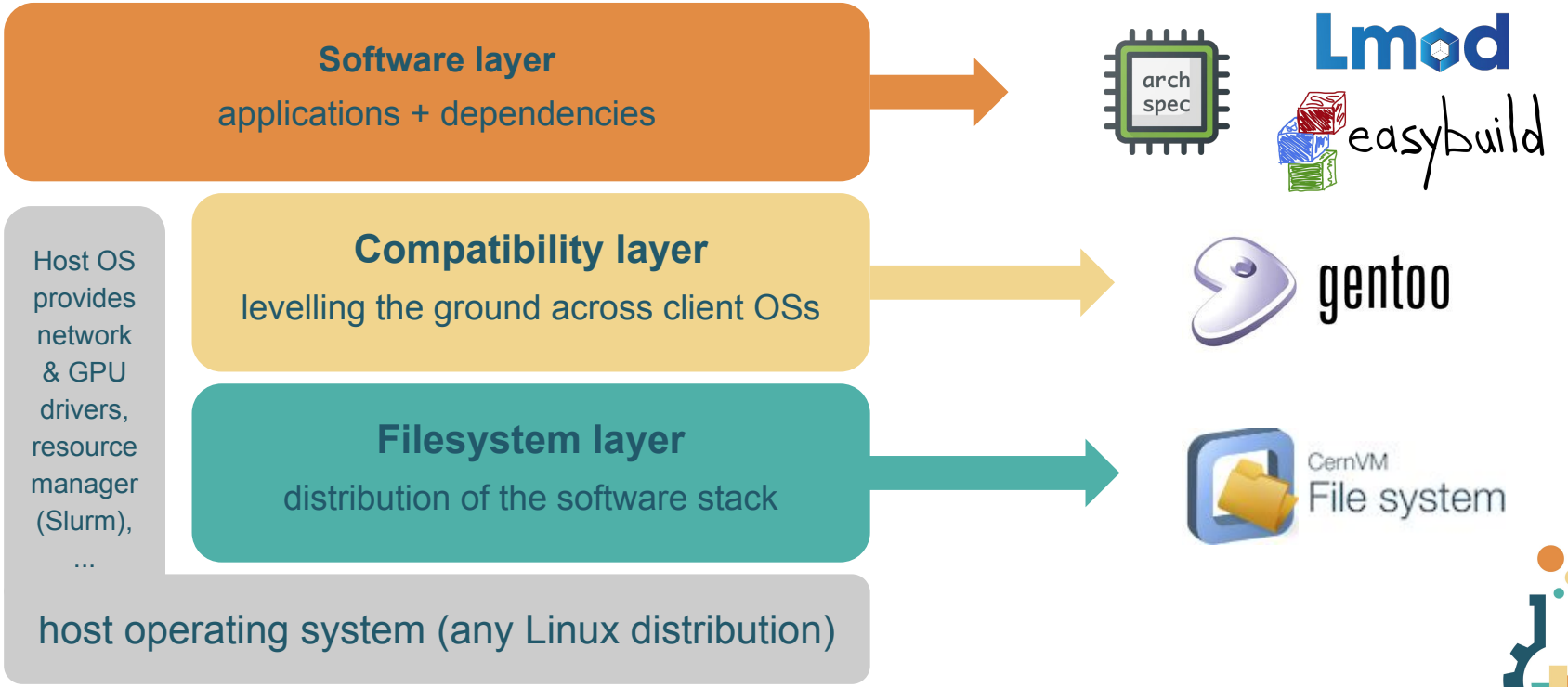
- **European Environment for Scientific Software Installations (EESSI)**
- Collaboration between different European partners in HPC community
- Goal: building a **common** scientific software stack,

  for HPC systems & beyond (personal workstations, cloud instances, ...)
- Heavily inspired by Compute Canada software stack
- Focus on performance, automation, testing, collaboration, ...

High-level overview of the EESSI project

https://www.eessi-hpc.org

https://eessi.github.io/docs



Questions?

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