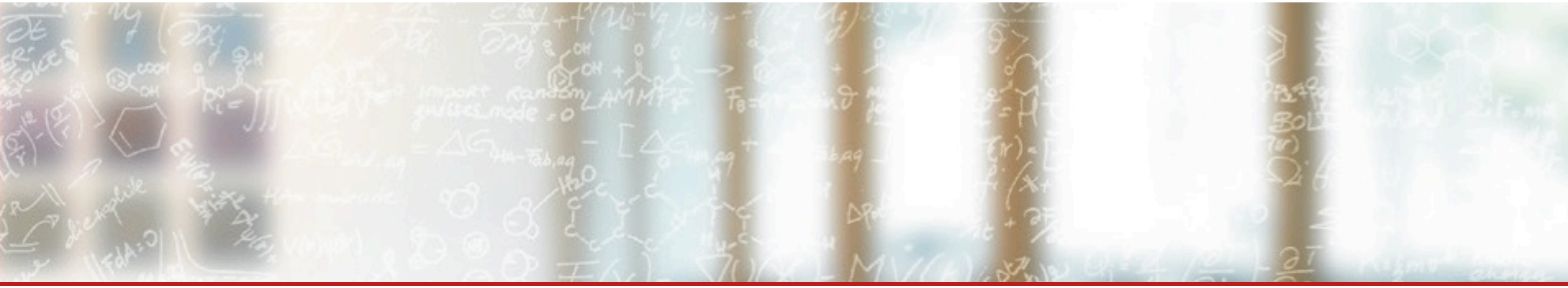
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich



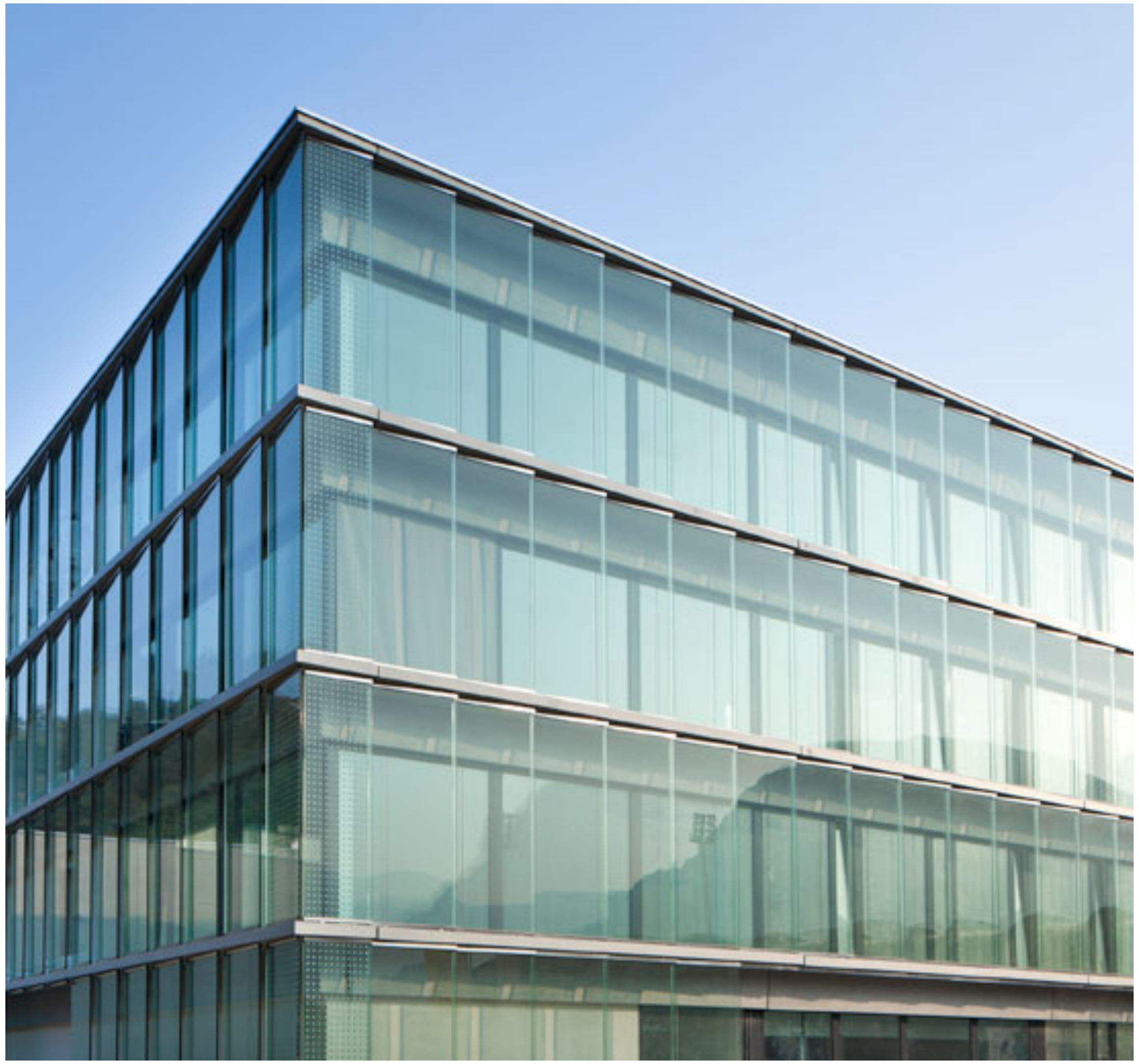
New EasyBuild workflow for CSCS

Technical Seminar

Guilherme Peretti-Pezzi, CSCS

July 14, 2015

Outline



- Automatic building tools & EasyBuild Intro
- EasyBuild setup @ CSCS
- Proposed workflow

HPC Building tools: basic properties and goals

- Perform automatic builds of scientific software
  - Including the dependencies and underlying software stack (libraries and compilers)
- Enable reproducibility
  - Set up once, rebuild easily (for maintenances and multiple deployments)
- **Improve portability**
  - Minimizes dependencies on the system software
    - A full programming environment can be deployed only requiring installed gcc, binutils & python
- Simplify upgrades
  - Trying new software versions is trivial
- Increase possibilities at user level (= without sudo)
  - Bleeding edge software co-exist with conservative (supported) packages
  - Users are able to build their own programming environments
    - And use it on different systems/sites

EasyBuild

- By far the most popular and active HPC building tool
  - 500+ supported applications
  - 40+ toolchains
  - Jülich Supercomputer Centre, Flemish Supercomputer Centre, sciCORE/UniBas, Stanford Univ., Univ. of Auckland, Bayer AG, Texas A&M, IMB (Austria), Univ. of Luxembourg, Cyprus Institute

- Community oriented
  - Shared (tested) build recipes among HPC centers
  - Standard toolchains are influenced by the users
  - Mailing lists, IRC channel, GitHub repository

- Can ease the deployment across systems
  - Ability to easily provide a uniform set of compilers & libraries (including versions)
  - Even across different sites
    - With a minimal coordination a ‘common’ toolchain can be supported

Some of the stock EasyBuild toolchains

- ClangGCC: Clang, GCC
- CrayCCE: PrgEnv-cray, fftw
- **CrayGNU: PrgEnv-gnu, fftw**
- CrayIntel: PrgEnv-intel, fftw
- GCC: GCC
- cgmpich: Clang, GCC, MPICH
- cgmvapich2: Clang, GCC, MVAPICH2
- cgompi: Clang, GCC, OpenMPI
- **dummy: (system libs and compilers)**
- **foss: BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK**
- gcccuda: CUDA, GCC
- **gmvapich2: GCC, MVAPICH2**
- gmvolf: BLACS, FFTW, GCC, MVAPICH2, OpenBLAS, ScaLAPACK
- gompic: CUDA, GCC, OpenMPI
- gpsolf: BLACS, FFTW, GCC, OpenBLAS, ScaLAPACK, psmpi
- iccifort: icc, ifort
- ictce: icc, ifort, imkl, impi
- intel: icc, ifort, imkl, impi
- iomkl: OpenMPI, icc, ifort, imkl
- iqacml: ACML, BLACS, FFTW, QLogicMPI, ScaLAPACK, icc, ifort

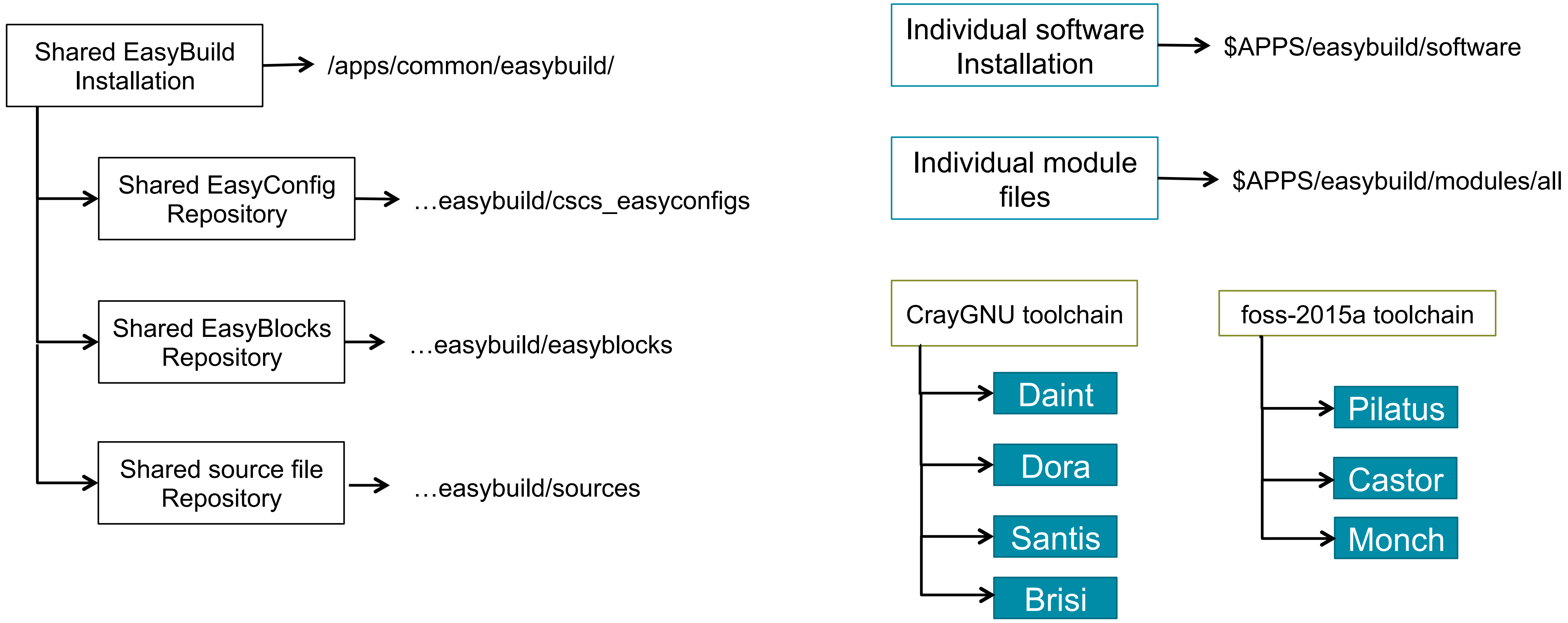
Upcoming feature (EB 2.2):
subtoolchain

Full list available with:
eb --list-toolchains

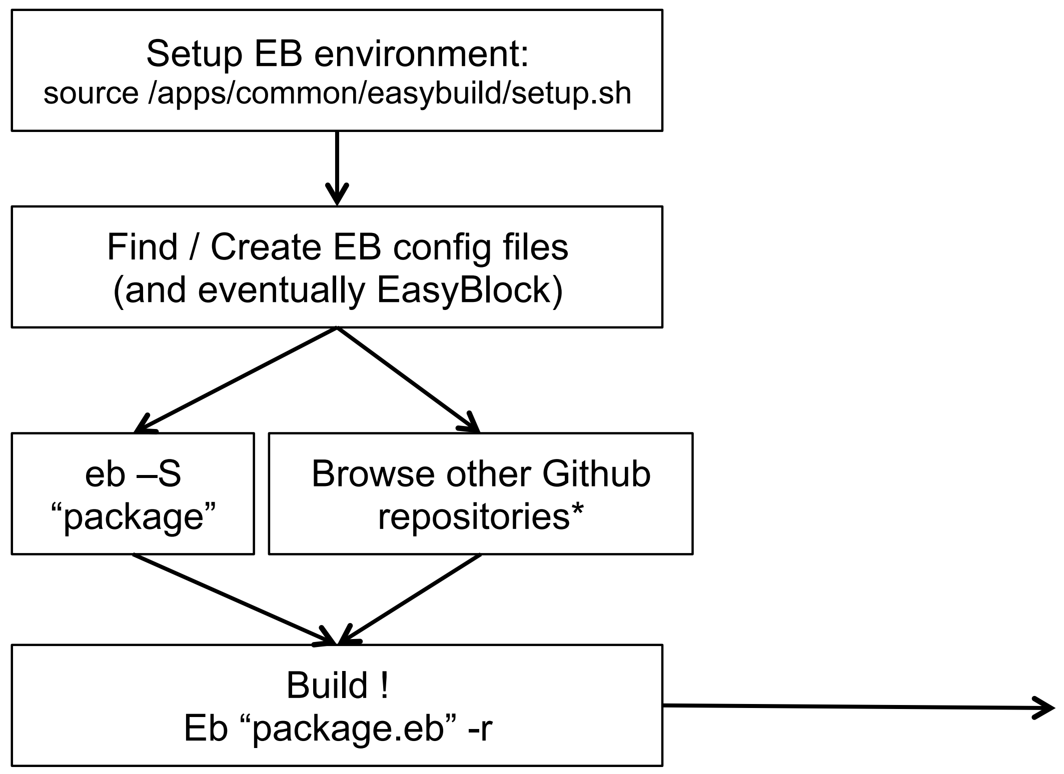
EasyBuild in a nutshell

	Role	When to modify/create a new instance
EasyConfigs (text files)	**Describe one build:** Name, versions, source, build instructions, easyblock, toolchain, dependencies	Standard builds (most cases): Configure/Make, Cmake, Bash script, and all the supported EasyBlocks
EasyBlocks (Python files)	**Describe general build instructions:** translate the information on easyconfig files to command lines	When special steps are required and are not covered by existing easyblocks nor by the easyconfig syntax
Toolchains (Python files)	**Define a programming environment:** compilers and (eventually) required libraries	Existing toolchains do not provide the required set of compilers and libraries
Framework (Python files)	**Orchestrate the building process:** easyconfig parsing, logging, module creation	When new features are needed or the general behavior needs to be tailored

EasyBuild setup @ CSCS


Shared EasyBuild Installation
/apps/common/easybuild/
Shared EasyConfig Repository
…easybuild/cscs_easyconfigs
Shared EasyBlocks Repository
…easybuild/easyblocks
Shared source file Repository
…easybuild/sources
Individual software Installation
$APPS/easybuild/software
Individual module files
$APPS/easybuild/modules/all
CrayGNU toolchain
Daint
Dora
Santis
Brisi
foss-2015a toolchain
Pilatus
Castor
Monch

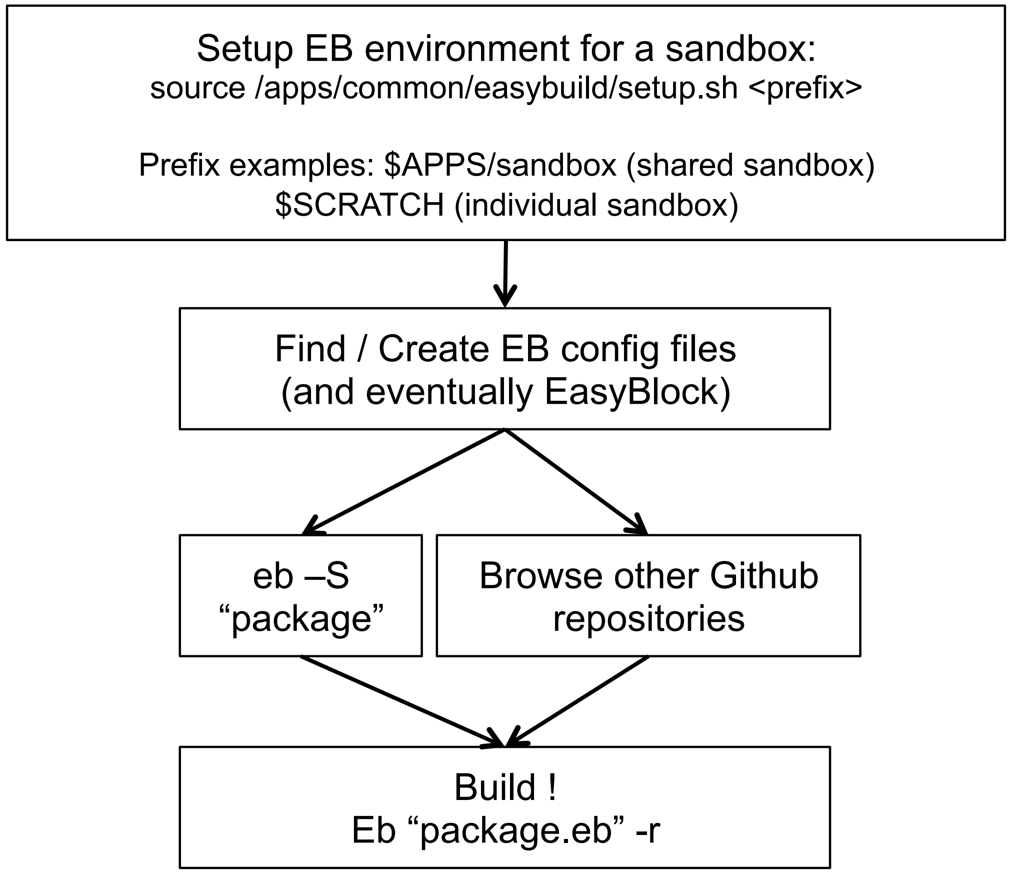
Proposed EasyBuild workflow for production (SCS)


Setup EB environment:
source /apps/common/easybuild/setup.sh
Find / Create EB config files
(and eventually EasyBlock)
eb –S
“package”
Browse other Github
repositories*
Build !
Eb “package.eb” -r

- Build (+dependencies)
- Install
- Create module files
- If successful
  - Commit easyconfig file to CSCS Git repository!!

*Links on the last slide

Proposed EasyBuild workflow for development (usable by all CSCS)


Setup EB environment for a sandbox:
source /apps/common/easybuild/setup.sh <prefix>
Prefix examples: $APPS/sandbox (shared sandbox)
$SCRATCH (individual sandbox)
Find / Create EB config files
(and eventually EasyBlock)
eb –S
“package”
Browse other Github
repositories
Build !
Eb “package.eb” -r

Creating new EasyConfig files

- Copy existing one and manually set the desired:
  - Toolchain
  - Version
  - Dependencies
    - System (‘EXTERNAL_MODULE’)
    - Modules built with EasyBuild (will match toolchain and toolchain version)

- Use the eb to automatically tweak existing EasyConfig files:
  - If the software version is available with another toolchain
    - eb package.eb --try-toolchain=new-toolchain,version
  - If you wish to update an existing version
    - eb --try-software-version=version
  - For more options
    - eb -H

EasyConfig file example: netCDF

```
name = 'netCDF’ # Will use easyblock netCDF.py
version = '4.3.3.1'

homepage = 'http://www.unidata.ucar.edu/software/netcdf/'
description = """NetCDF (network Common Data Form) is a set of software libraries  and machine-independent data formats that ….. scientific data."""

toolchain = {'name': 'foss', 'version': '2015a'}
toolchainopts = {'pic': True, 'usempi': True}

sources = [SOURCELOWER_TAR_GZ]
source_urls = ['http://www.unidata.ucar.edu/downloads/netcdf/ftp/']

dependencies = [('HDF5', '1.8.15')]     # runtime dependencies
builddependencies = [   ('CMake', '3.0.0'),     ('Doxygen', '1.8.7'),     ('cURL', '7.37.1'), ] # build only dependencies (not added on the module file)
configopts = [ "-DCURL_LIBRARY=$EBROOTCURL/lib/libcurl.so -DCURL_INCLUDE_DIR=$EBROOTCURL/include -DBUILD_SHARED_LIBS=ON",]

sanity_check_paths = { 'files': ['lib64/libnetcdf.so'],     'dirs': [], }
moduleclass = 'data'
```

List of all available EasyConfig parameters:
```
$ eb -a
```

Python use case

- Suported modules for Python 2 and 3
  - Setuptools 17.1.1, Pip 7.0.3, Nose 1.3.7, Numpy 1.9.2, Scipy 0.15.1, mpi4py 1.3.1, Cython 0.22, Six 1.9.0, Virtualenv 13.0.3, pandas 0.16.2, h5py 2.5.0 (serial/parallel), Matplotlib 1.4.3, pyCuda 2015.1, netcdf4 1.1.8

- Example Easyconfig files (for Python 2.7.10 on Cray)
  - Python-2.7.10-CrayGNU-5.2.40.eb
  - matplotlib-1.4.3-CrayGNU-5.2.40-Python-2.7.10.eb
  - netcdf4-python-1.1.8-CrayGNU-5.2.40-Python-2.7.10.eb
  - h5py-2.5.0-CrayGNU-5.2.40-Python-2.7.10-parallel.eb
  - h5py-2.5.0-CrayGNU-5.2.40-Python-2.7.10-serial.eb
  - pycuda-2015.1-CrayGNU-5.2.40-Python-2.7.10.eb
- Easyblocks
  - h5py.py, netcdf_python.py, pycuda.py

Now available on:
- Daint, Dora, Santis, Brisi (CrayGNU)
- Pilatus, Castor (foss)

Main differences from current (manual) installation

- Currently everyone has a different way of creating install recipes
  - Text file, shell script, publicly available or highly secret

- New method encourages everyone to follow a standard procedure
  - Straightforward for most cases
    - New versions of libraries and compilers can be easily created by using existing files
  - For new toolchains and non-standard builds
    - Some basic knowledge of Python and understanding of the EB framework is needed

- Modules are associated by default to a toolchain
  - Conflicts will appear if users try to load modules from different toolchains
    - Current modules (in general) are more permissive

Technical Tips & Tricks

- When a build fails
  - Check log (on /tmp, full path is shown on stdout)
  - If you know how to fix
    - modify the easyconfig file accordingly and re-run eb
  - If not
    - Go to build dir (by default under /dev/shm/'username')
    - Manually load the required modules and manually retry the build (to debug)
      - For example taking the full "./configure " command line from the log
  - Once you figure out how to fix, modify the easyconfig file and re-run eb
    - If you cannot find a solution with EasyBuild,
    - You can always install manually
      - Preferably changing the prefix

- When builds succeed
  - Logs and configuration files can be found inside the installation directory
    - Install_dir/easybuild

Final thoughts

- Current installation is ready for application level
  - Validation with Python use case (including modules)
    - Daint, Dora, Santis, Brisi, Pilatus, Castor

- Continuous validation techniques can be easily applied
  - Testing builds across all systems with Jenkins
  - Changes/errors on the PrgEnv can be detected early

- In order to get the most out of EasyBuild
  - We need to have consistent PrgEnv on most systems
    - OK on Cray systems
    - Not true on non-Cray
      - (easily achieved with EasyBuild)

Next steps (SCS)

- Start trying out EB for answering tickets requesting new software
  - Testing and feedback are very welcome

- Agree on a toolchain for non-Cray systems
  - Officially unsupported?

  1. Stock toolchain
     - Default “foss” toolchain works just fine for Python use case
     - May be not optimal for other apps (for example concerning MPI)
  2. Tailored toolchain using existing PrgEnv (supported by HPC Operations team)
     - Possible, but
       - Requires more work tweaking the EB framework
       - We might end up with a different toolchain on each system
  3. Quick alternative solution
     - Use a stock toolchain and only tailor the compilation parameters
       - (this approach was tried on the Storm-CH)

Links

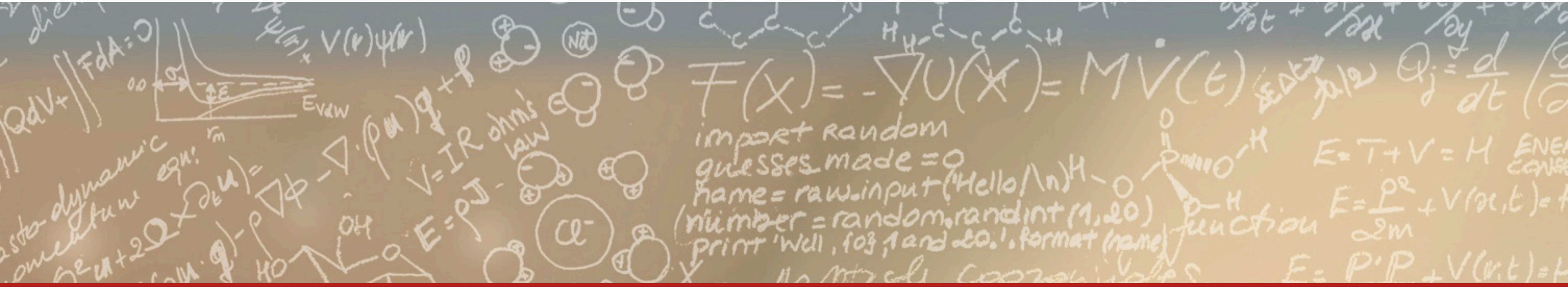
- Easybuild Documentation
  - GitHub
    - https://github.com/hpcugent/easybuild
  - Workflow example (WRF)
    - http://easybuild.readthedocs.org/en/latest/Typical_workflow_example_with_WRF.html

- CSCS Internal doc
  - https://github.com/eth-cscs/tools/wiki/EasyBuild-at-CSCS

- Additional easyconfig files repositories
  - Development EasyBuild branch
    - https://github.com/hpcugent/easybuild-easyconfigs/tree/develop
  - Successful production builds at CSCS
    - https://github.com/eth-cscs/tools/tree/master/easybuild/ebfiles_repo

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.