easybuild

building software with ease

PyBUG meeting @ Ghent
lightning talk - Oct. 1st 2013

kenneth.hoste@ugent.be
easybuild@lists.ugent.be
About HPC UGent:

- central contact for HPC at Ghent University
- part of central IT department (DICT)
- member of Flemish supercomputer centre (VSC)
  - collaboration between Flemish university associations

- seven Tier2 systems, one Tier1 system
  - Top500: #119 (June’12), #163 (Nov’12), #239 (June’13)
- team consists of 7 FTEs
- tasks include system administration of HPC infrastructure, user training, user support, ...
Building scientific software is... fun!

Scientists focus on the functionality of their software, not on portability, build system, ...

Common issues with build procedures of scientific software:

- **incomplete**, e.g. no install step
- requiring human **interaction**
- heavily **customised** and **non-standard**
- uses **hard-coded** settings
- poor and/or outdated **documentation**

**Very time-consuming** for user support teams!
building from source is preferred in an HPC environment

performance is critical, instruction selection is key (e.g. AVX)

not a lot of packaged scientific software available (RPMs, ...)

requires huge effort, which is duplicated across distros

existing build tools are

hard to maintain (e.g., bash scripts)

stand-alone, no reuse of previous efforts

OS-dependent (HomeBrew, *Ports, ...)

custom to (groups of) software packages
e.g., Dorsal (DOLFIN), gmkpack (ALADIN)
Building software with ease

easybuild

a software build and installation framework

- written in **Python**
- developed in-house (HPC-UGent) for 2.5 years
- open-source (**GPLv2**) since April 2012
- **stable API** since Nov. 2012 (v1.0.0)
- latest release: v1.7.0 (v1.8.0 due this week)
- continuously enhanced and extended

http://hpcugent.github.com/easybuild
Installing EasyBuild

$ easy_install --user easybuild
error: option --user not recognized (only for recent setuptools)

You should be using pip!

$ pip install --user easybuild
pip: No such file or directory (pip not installed)

Just use --prefix with easy_install!

$ easy_install --prefix=$HOME easybuild

$ export PATH=$HOME/bin:$PATH

$ eb --version

ERROR: Failed to locate EasyBuild's main script
(PYTHONPATH not set correctly)
Easily install EasyBuild by bootstrapping it.

https://github.com/hpcugent/easybuild/wiki/Bootstrapping-EasyBuild

$ wget http://hpcugent.github.com/easybuild/bootstrap_eb.py
$ python bootstrap_eb.py $HOME

This will install EasyBuild with EasyBuild, and produce a module:

$ export MODULEPATH=$HOME/modules/all:$MODULEPATH
$ module load EasyBuild/1.7.0
$ eb --version

This is EasyBuild 1.7.0 (framework: 1.7.0, easyblocks: 1.7.0)
easybuild

High-level design

easybuild

tools
asyncprocess
build_log
cfg
environment
filetools
module_generator
modules
ordereddict
parallelbuild
pbs_job
repository
systemtools
variables
version

toolchain

toolchains

framework
easyblock
EasyBlock
...
extension
Extension
easyconfig
EasyConfig
...

easyblocks
generic
configuremake
ConfigureMake
cmakemake
CMakeMake
...
armadillo
EB_Armadillo
cp2k
EB_CP2K
wrf
EB_WRF
...

scripts

20

10

6

4

2

0

8
Step-wise install procedure

build and install procedure as implemented by EasyBuild

most of these steps can be customized if required
Features

- **logging** and archiving
  - entire build process is logged thoroughly, logs stored in install dir
  - easyconfig file used for build is archived (file/svn/git repo)

- **automatic dependency resolution**
  - software stack be built with a single command, using --robot

- running **interactive installers autonomously**
  - by passing a Q&A Python dictionary to the run_cmd_qa function

- building software in **parallel**
  - e.g., on a (PBS) cluster, by using --job

- comprehensive **testing**: unit tests, regression testing
Comprehensive testing

- unit tests are run automagically by Jenkins
- regression test results are pulled in
- publicly accessible: https://jenkins1.ugent.be/view/EasyBuild
List of supported software (v1.7.0)

329 different software packages (1,631 example easyconfigs)

ABAQUS ABINIT ABySS ACML ALADIN ALLPATHS-LG AMOS ASE ATLAS AnalyzeFMRI Armadillo Autoconf Automake a2ps ant aria2 BFAST BLACS BLAST BLAT BWA BamTools Bash BiSearch BioPerl Biopython Bison Bonnie ++ Boost Bowtie Bowtie2 bam2fastq bbFTP bbcp bbftpPRO beagle-lib binutils biodeps byacc bzip2 CBLAS CCfits CD-HIT CFITSIO CGAL CLHEP CMake CP2K CPLEX CRF++ CUDA CVXOPT Chapel Clang ClangGCC ClustalW2 Courscrew Cufflinks Cython cURL cairo ccache cflow cgdbl cgmpich cgmpolf cgmvapich2 cgmvolf cgompi cgoolf DL_POLY_Classic DOLFIN Diffutils Docutils Doxygen ECore ELinks EMBOSS EPD ESMF ESPResSo EasyBuild Eigen expat FASTA FASTX-Toolkit FCM FFC FFTW FIAT FLUENT FRC_align FSL Ferret FreeSurfer findutils flex fmri fontconfig freeglut freetype GATE GATK GCC GDAL GDB GEOS GHC GLIMMER GLPK GLib GMP GPAW GROMACS GSL Geant4 Greenlet g2lib g2lib gawk gettext git glproto gmacl gmvapich2 gmvolf gnuplot gnutls goalf gompi google-sparsehash goolf gooFLC gperf grlib_api guile gzip HDF HDF5 HH-suite HMMER HPCBIOS_Bioinfo HPCBIOS_Debuggers HPCBIOS_LifeSciences HPCBIOS_Math HPCBIOS_Profilers HPL Harminv Hypre h5py h5utils horton hwloc Infernal Inspector Instant Iperf IronPython icc iccfort iccfort ifort iqmmpi imkl imp_iomkl ipp icqcmkl itac JUnit JasPer Java Junia2 LAPACK LZO LibTIFF Libint Lftp libct libdrm libffi libgtextutils libharu libibmad libibumad libibverbs libidn libint2 libmatheval libpciaccess libpq libthread-stubs libbrepline libsmm libtool libungif libxsmllibxml libxml2 libxslt libyaml likwid lxml M4 MATLAB MCL MDP MEME METIS MPFR MPICH MTL4 MUMmer MUSCLE MVAPICH2 Maple MariaDB Meep Mercurial Mesa MetaVelvet Mono Motthor MrBayes MyMediaLite make makedepend matplotlib mc molmod mpi4py mpiBLAST NAMD NCBI-Toolkit NCL NEURON NWChem nano ncurses netCDF netCDF-C++ netCDF-Fortran nettle ns numacltl numexpr numpy ORCA Oases Oger OpenBLAS OpenFOAM OpenIFS OpenMPI OpenPGM OpenSSL orthomcl otcl PAML PAPI PCRE PETSc PLINK PSI ParMETIS Pasha Perl Primer3 PyYAML PyZMQ Python pandas parallel paycheck petsc4py phonopy pixman pkg-config problog pyTables python-meeple QLogicMPI Qt QuantumESPRESSO R RAXML RCS RNAz ROOT Rosetta SAMtools SCOOP SCOTCH SCons SHRIMP SLEPc SOAPdenovo SQLite SWIG ScaLAPACK ScientificPython Shapely Sphinx Stacks Stow SuiteSparse Szip scikit-learn scipy setup tools symy Tar Tcl Theano TiCCutils TiMBL TinySVM Tk TopHat Tornado TotalView Trilinos Trinity tbb tclcl tcs h UDUNITs UFC UFL util-linux VSC-tools VTK VTune Valgrind Velvet ViennaRNA Viper WIEN2k WPS WRF wiki2beamer XCRYSDen XML XML-LibXML XML-Simple xcb-proto xorg-macros xproto YAML-Syck YamCha Yasm yaff ZeroMQ zlib zsh zsync
easybuild

building software with ease

Do you want to know more?

website: http://hpcugent.github.com/easybuild

GitHub: https://github.com/hpcugent/easybuild[-framework|-easyblocks|-easyconfigs]

PyPi: http://pypi.python.org/pypi/easybuild[-framework|-easyblocks|-easyconfigs]

mailing list: easybuild@lists.ugent.be

Twitter: @easy_build

IRC: #easybuild on freenode.net
building and installing **WRF** *(Weather Research and Forecasting Model)*

- http://www.wrf-model.org
- complex(ish) dependency graph
- very non-standard build procedure
  - interactive configure script (!)
  - resulting `configure.wrf` needs work (hardcoding, tweaking of options, ...)
- compile script (wraps around `make`)
- no actual installation step
building and installing **WRF** *(Weather Research and Forecasting Model)*

- easyblock that comes with EasyBuild implements build procedure
  - running `configure` script **autonomously**
  - **building** with compile and **patching** `configure.wrf`
  - **testing** build with standard included tests/benchmarks
- various example easyconfig files available
  - different versions, toolchains, build options, ...
- building and installing WRF becomes child’s play, for example:

```
  eb --software=WRF,3.4 --toolchain-name=ictce --robot
```
**Use case: WRF - easyblock (1/3)**

imports, class constructor, custom easyconfig parameter

```python
1 import fileinput, os, re, sys
2 import easybuild.tools.environment as env
3 from easybuild.easyblocks.netcdf import set_netcdf_env_vars
4 from easybuild.framework.easyblock import EasyBlock
5 from easybuild.framework.easyconfig import MANDATORY
6 from easybuild.tools.filetools import patch_perl_script_autoflush, run_cmd, run_cmd_qa
7 from easybuild.tools.modules import get_software_root
8
9 class EB_WRF(EasyBlock):
10     def __init__(self, *args, **kwargs):
11         super(EB_WRF, self).__init__(*args, **kwargs)
12         self.build_in_installdir = True
13
14 @staticmethod
15 def extra_options():
16     extra_vars = [(['buildtype'], [None, "Type of build (e.g., dmpar, dm+sm).", MANDATORY])]
17     return EasyBlock.extra_options(extra_vars)
```

**class definition**

**class constructor, specify building in installation dir**

**define custom easyconfig parameters**
def configure_step(self):
    # prepare to configure
    set_netcdf_env_vars(self.log)

    jasper = get_software_root('JasPer')
    if jasper:
        jasperlibdir = os.path.join(jasper, 'lib')
        env.setvar('JASPERINC', os.path.join(jasper, 'include'))
        env.setvar('JASPERLIB', jasperlibdir)

    env.setvar('WRFIO_NCD_LARGE_FILE_SUPPORT', '1')

    patch_perl_script_autoflush(os.path.join('arch', 'Config_new.pl'))

    known_build_types = ['serial', 'smpar', 'dmpar', 'dm+sm']
    self.parallel_build_types = ['dmpar', 'smpar', 'dm+sm']
    bt = self.cfg['buildtype']

    if not bt in known_build_types:
        self.log.error("Unknown build type: '%s' (supported: %s)" % (bt, known_build_types))
Use case: WRF - easyblock (2/3)
configuration (part 2/2)

```python
# run configure script
bt_option = "Linux x86_64 i486 i586 i686, ifort compiler with icc"
bt_question = "\s*(?P<nr>[0-9]+).\s*%s\s*(%s)" % (bt_option, bt)

cmd = "./configure"
qa = {
    "(1= basic, 2= preset moves, 3= vortex following) [default 1]": "1",
    "(0= no nesting, 1= basic, 2= preset moves, 3= vortex following) [default 0]": "0"
}
std_qa = {r"%s.*\n(.*\n)*Enter selection\s*\[[0-9]+-[0-9]+\]\s*: " % bt_question: "%(nr)s"}

run_cmd_qa(cmd, qa, no_qa=[], std_qa=std_qa, log_all=True, simple=True)

# patch configure.wrf
cfgfile = 'configure.wrf'

comps = {
    'SCC': os.getenv('CC'),
    'SFC': os.getenv('F90'),
    'CCOMP': os.getenv('CC'),
    'DM_FC': os.getenv('MPIF90'),
    'DM_CC': "%s -DMPIZ_SUPPORT" % os.getenv('MPICC'),
}

for line in fileinput.input(cfgfile, inplace=1, backup='.orig.comps'):
    for (k, v) in comps.items() :
        line = re.sub(r"\^%(s\s*=%s)*\$" % k, r"\1 \%s" % v, line)
    sys.stdout.write(line)
```
def build_step(self):
    # build WRF using the compile script
    par = self.cfg['parallel']
    cmd = './compile -j %d wrf' % par
    run_cmd(cmd, log_all=True, simple=True, log_output=True)

    # build two test cases to produce ideal.exe and real.exe
    for test in ['em_real', 'em_b_wave']:
        cmd = './compile -j %d %s' % (par, test)
        run_cmd(cmd, log_all=True, simple=True, log_output=True)

def install_step(self):
    pass
Use case: installing WRF

specify build details in easyconfig file (.eb)

```
name = 'WRF'
version = '3.4'
homepage = 'http://www.wrf-model.org'
description = 'Weather Research and Forecasting'
toolchain = {'name': 'ictce', 'version': '3.2.2.u3'}
toolchainopts = {'opt': False, 'optarch': False}
sources = ['%s%s.TAR.gz' % (name, version)]
patches = ['WRF_parallel_build_fix.patch',
           'WRF-3.4_known_problems.patch',
           'WRF_tests_limit-runtimes.patch',
           'WRF_netCDF-Fortran_separate_path.patch']
dependencies = [('JasPer', '1.900.1'),
                ('netCDF', '4.2'),
                ('netCDF-Fortran', '4.2')]
buildtype = 'dmpar'
```

```
eb WRF-3.4-ictce-3.2.2.u3-dmpar.eb --robot
```
easybuild

building software with ease

PyBUG meeting @ Ghent
lightning talk - Oct. 1st 2013

kenneth.hoste@ugent.be
easybuild@lists.ugent.be