BLUE WATERS-GEO

MAPPING AND MODELING THE WORLD

Third Party Software Management

Brett Bode, Assistant Project Director
brett@Illinois.edu

Third Party Software

- As soon as you allow user access to your system you will receive requests for additional software installations!
  - You should have a defined process for reviewing the request!
  - If you can satisfy the requests via your OS distribution do it!
  - Is it your job to install science applications?
    - On BWs we consider if a requested package is likely to be useful by more than one or two users, if so we may take on the installation of the software into a space available to all users, otherwise we will advise the user on how to install it themselves.
  - It is also common for there to be requests for versions of base software newer than provided with the OS.
    - Compilers, Python, git, etc
    - This is particularly common with “enterprise” linux distributions and as your system “ages”.

Options for Managing Local Installations

- Just install it in /usr/local/ or some other local directory.
  - Must replicate the install on each node, at least for the runtime components
  - What if you need multiple versions of the same tool?
  - As the number of installations increase possibly subtle conflicts are likely!
  - How do you handle the local installs when the base OS is upgrade?
- Install it in a directory on a globally available file system
  - Solves the replication problem, but not the others.
  - **Best practice**: use a globally available directory and define a software installation group that allows installations without needing administrative privileges.

How do you solve the other issues?

- We need a management tool that allows specified directories to be added to the appropriate search paths on demand.
- Also needs a way to express conflicts and dependencies.
- Finally a tool that automates the process of building and installing in a reproducible way would allow clean rebuilds upon demand.

- The Environment Modules tool addresses the first two!

Modules/LMod

- Modules is a simple tool that simplifies isolated software installation by:
  - Packages up the environment settings needed to add a directory tree to your search path (include, library, binary search…)
  - Includes package dependencies and conflicts

- LMod is a newer implementation of an Environment Modules system
  - Designed to be more efficient and scalable
  - Few other nice additions

Toolchains

- One of the challenges in managing third party software is that mixing libraries built with different compilers can lead to issues.
- One solution is to start with each compiler and build all the major libraries with that compiler.
  - The result is a toolchain = compiler + math libraries + communication libraries
  - If you have several compilers then the challenge is building each piece of software for each toolchain.
  - On Blue Waters Cray provides GNU, Cray, PGI and Intel compilers and associated libraries.
    - Need an automated solution to building software against each toolchain!

easybuild

http://easybuild.readthedocs.org/en/latest/Introduction.html

- a **flexible framework** for building/installing (scientific) software
- fully **automates** software builds
- divert from the standard configure / make / make install with custom procedures
- allows for easily **reproducing** previous builds
- keep the software build recipes/specifications **simple and human-readable**
- supports **co-existence of versions/builds** via dedicated installation prefix and module files
- enables **sharing** with the HPC community (win-win situation)
- automagic **dependency resolution**
- **retain logs** for traceability of the build processes
- Almost 2000 packages
  - Compilers, MPI, libraries, tools (git, autotools, etc), languages (R, Python), full applications

Look for something like WRF:

```
eb -S wrf
== temporary log file in case of crash /tmp/eb-fAJkDl/easybuild-BIeOFd.log
== Searching (case-insensitive) for 'wrf' in /sw/EasyBuild/software/EasyBuild/2.2.0/lib/python2.6/site-packages/easybuild_easyconfigs-2.2.0-py2.6.egg/easybuild/easyconfigs
CFGS1=/sw/EasyBuild/software/EasyBuild/2.2.0/lib/python2.6/site-packages/easybuild_easyconfigs-2.2.0-py2.6.egg/easybuild/easyconfigs/w/WRF
 * $CFGS1/WRF-3.3.1-goalf-1.1.0-no-OFED-dmpar.eb
 * $CFGS1/WRF-3.3.1-goolf-1.4.10-dmpar.eb
 * $CFGS1/WRF-3.3.1-ictce-3.2.2.u3-dmpar.eb
 * $CFGS1/WRF-3.3.1-ictce-5.3.0-dmpar.eb
 * $CFGS1/WRF-3.3.1_GCC-build_fix.patch
 ---snip---
 * $CFGS1/WRF-3.6.1-intel-2014b-dmpar.eb
 * $CFGS1/WRF-3.6.1-intel-2015a-dmpar.eb
 * $CFGS1/WRF-3.6.1_known_problems.patch
 * $CFGS1/WRF-3.6.1_netCDF-Fortran_separate_path.patch
 * $CFGS1/WRF_FC-output-spec_fix.patch
 * $CFGS1/WRF_netCDF-Fortran_separate_path.patch
 * $CFGS1/WRF_no-GCC-graphite-loop-opts.patch
 * $CFGS1/WRF_parallel_build_fix.patch
 * $CFGS1/WRF_tests_limit-runtimes.patch
== temporary log file(s) /tmp/eb-fAJkDl/easybuild-BIeOFd.log* have been removed.
== temporary directory /tmp/eb-fAJkDl has been removed.
```

woah, that's a lot of choices. What do they all mean?

first, note at the top it tells you where all the configs are stored.

The naming scheme is PackageName-VersionOfPackage-ToolChainName-ToolChainVersion-Extra. What is a toolchain?

What are all those tool chains?

```
eb --list-toolchains
List of known toolchains (toolchainname: module[,module...]):
        ClangGCC: Clang, GCC
        CrayCCE: PrgEnv-cray, fftw
        CrayGNU: PrgEnv-gnu, fftw
        CrayIntel: PrgEnv-intel, fftw
        GCC: GCC
        GNU: GCC
        cgmpich: Clang, GCC, MPICH
        cgmpolf: BLACS, Clang, FFTW, GCC, MPICH, OpenBLAS, ScaLAPACK
        cgmvapich2: Clang, GCC, MVAPICH2
        cgmvolf: BLACS, Clang, FFTW, GCC, MVAPICH2, OpenBLAS, ScaLAPACK
        cgompi: Clang, GCC, OpenMPI
        cgoolf: BLACS, Clang, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
        dummy:
        foss: BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
        gcccuda: CUDA, GCC
        gimkl: GCC, imkl, impi
        gimpi: GCC, impi
        gmacml: ACML, BLACS, FFTW, GCC, MVAPICH2, ScaLAPACK
        gmpich: GCC, MPICH
        gmpich2: GCC, MPICH2
        gmpolf: BLACS, FFTW, GCC, MPICH, OpenBLAS, ScaLAPACK
        gmvapich2: GCC, MVAPICH2
        gmvolf: BLACS, FFTW, GCC, MVAPICH2, OpenBLAS, ScaLAPACK
        goalf: ATLAS, BLACS, FFTW, GCC, OpenMPI, ScaLAPACK
        gompi: GCC, OpenMPI
        gompic: CUDA, GCC, OpenMPI
        goolf: BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
        goolfc: BLACS, CUDA, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
        gpsmpi: GCC, psmpi
        gpsolf: BLACS, FFTW, GCC, OpenBLAS, ScaLAPACK, psmpi
```

EasyBuild Toolchains

- Tool chains are a combination of compiler and various runtime libraries, usually an MPI library and some Math libraries. You can define your own combination!
- The tool chain provides a known base to build upon and is where some of the reproducibility comes from. You will certainly not make use of all of the tool chains, but you may use more than one. Some obviously wrapper installed binaries (Intel compiler) while the Gnu ones build everything from scratch.
- The stock tool chains have four base compilers, Clang, GCC, Intel and Cray (only relevant for Cray systems) combined with MPICH, OpenMPI, and Intel MPI along with libraries such as OpenBLAS, ScaLAPACK and FFTW.
- The tool chain with the broadest use is goolf – (Gnu compiler, OpenMPI, OpenBLAS, ScaLAPACK, and FFTW)
- There is a convention of adding “-no-OFED” to tool chains built without InfiniBand support.

Spack

https://spack.io

- a **flexible framework** for building/installing (scientific) software
- fully **automates** software builds
- divert from the standard configure / make / make install with custom procedures
- keep the software build recipes/specifications **simple and human-readable**
- supports **co-existence of versions/builds** via dedicated installation prefix and module files
- enables **sharing** with the HPC community (win-win situation)
- automagic **dependency resolution**
- **retain logs** for traceability of the build processes
- Over 4000 packages

Installation Location

- Both EasyBuild and Spack are user-level tools so any user can use them to install software into the paths that they own.
  - A best practice is to test the build and installation of all software built in this fashion in an admins own directory before installing into a system-wide path.
  - Any accessible directory can be added to your modules path (module add <path>).

- Once the installation is tested and proven it is then installed in a path that is automatically added to all account's shells and available on all nodes.

EasyBuild versus Spack

- Both frameworks automate installation, manage dependencies and build modules a wide variety of software from compilers to full applications.
- Both are open source and written in Python.
- Both are user-level tools so can be used by individuals or by system admins to install software for general use.
- Both use a local source tar file cache – though it is intended to be persistent for EasyBuild. Useful to manually populate the source tarballs to avoid the automatic download when needed.
- EasyBuild has an emphasis on reproducibility while Spack emphasizes ease of use.
  - That makes Spack a more popular choice, but YMMV.

Questions?

- Email help+bw@ncsa.illinois.edu