Using EasyBuild and Continuous Integration for Deploying Scientific Applications on Large Scale Production Systems

HPC Advisory Council – Swiss Conference
Guilherme Peretti-Pezzi, CSCS
April 11, 2017
Using EasyBuild and Continuous Integration for Deploying Scientific Applications on Large Scale Production Systems

1. Introduction: Basic concepts & tools
   - Automating builds for HPC (EasyBuild, Spack, Nix, Guix)
   - Continuous Integration (CI) & Testing (Jenkins, Travis, Bamboo, CDash)
   - Collaborative development tools (Github, Gitlab, Bitbucket)

2. EasyBuild in a nutshell
   - Feature highlights
   - Production systems at CSCS running EasyBuild

3. Combining EasyBuild + CI + Github for testing and deploying builds
   - Overview of Piz Daint production setup
Issues when managing scientific software stacks (at scale)

- HPC software have particularly complex dependency list
- Building is very time consuming
  - Long compile times
    - Often require human intervention at some point
  - OS upgrades require re-building everything from scratch
    - And therefore they are often postponed
- Scattered and non-uniform build recipes
- Little collaboration among HPC centers & colleagues
- Reproducibility issues
  - Default module versions may change (ABI compatibility)
  - OS dependencies versions may change (example: zlib)
Issues when managing scientific software stacks (at scale)

- HPC software have particularly complex dependency list
  - Solution: Use a framework with dependency resolution
- Building is very time consuming
  - Solution: Use a framework that allows automation
- Scattered and non-uniform build recipes
  - Solution: Use a central repository for archiving build recipes (git)
- Little collaboration among HPC centers & colleagues
  - Solution: Adopt a collaborative on-line platform (Github)
- Reproducibility issues
  - Existing module versions (and defaults) may change
    - Solution: Always pin version numbers
  - OS dependencies may change (example: zlib)
    - Solutions
      - Use Continuous Integration
      - Minimize usage of OS deps (include them in the build recipe)
Basic concepts & tools (1/3)
Automating builds for HPC: approaches

- Building software from scratch
  - Pros
    - Binaries are optimized for the CPU architecture (instruction set)
    - Possible tuning to specific interconnect
  - Cons
    - Long build times
    - Potential issues due to differences in the base OS dependencies
  - Examples
    - EasyBuild, Spack

- Using (generic) pre-compiled binaries & containers
  - Pros
    - Portability: better isolation of OS dependencies (containers)
  - Cons
    - Not always optimized for CPU & interconnect
  - Examples
    - Nix, Guix, Shifter, Singularity
Continuous Integration (CI)
- Early detection of problems on complex systems
  - Achievable by performing regular builds
  - Can be extended to continuous delivery
    - Additional tests upon the provided software

Continuous Testing
- Useful for providing fast and continuous feedback on the contributions
  - Is typically based on unit testing

Popular tools
- Travis, Jenkins, CDash, Bamboo, gitlab-ci
Basic concepts & tools (3/3)
Collaborative development tools (Github, Gitlab, Bitbucket)

- Distributed version control (git server)
- Issues
  - bug tracking
  - feature requests
  - task management
- Wikis
- Pull/Merge requests
  - Code review and comments
- Integration with CI tools
- These tools can be useful also for managing build recipes
Table of Contents

1. Introduction: Basic concepts & tools
   - Automating builds for HPC (EasyBuild, Spack, Nix, Guix)
   - Continuous Integration (CI) & Testing (Jenkins, Travis, Bamboo, CDash)
   - Collaborative development tools (Github, Gitlab, Bitbucket)

2. EasyBuild in a nutshell
   - Feature highlights
   - Production systems at CSCS running EasyBuild

3. Combining EasyBuild + CI + Github for testing and deploying builds
   - Overview of Piz Daint production setup
EasyBuild: building software with ease

- framework for installing (scientific) software on HPC systems
- implemented as Python packages and modules
- started at University of Gent, Belgium, in 2009, open-source (GPLv2) since 2012
- now: thriving community; actively contributing, driving development
- new release every 6-8 weeks (latest: EasyBuild v3.1.2, Apr 4th 2017)
- supports over 1.100 different software packages
- including CP2K, GAMESS-US, GROMACS, NAMD, NWChem, OpenFOAM, PETSc, QuantumESPRESSO, WRF, WPS, . . .
- well documented: http://easybuild.readthedocs.io

http://hpcugent.github.io/easybuild/
EasyBuild: feature highlights

- Fully **autonomously** building and installing (not only scientific) software
  - automatic dependency resolution.
  - full integration with Environment Modules (Tcl or Lua syntax)
- thorough **logging** of executed build/install procedure
- **archiving** of build specifications (‘easyconfig files’)
- highly **configurable**, via config files/environment/command line
- **dynamically extendable** with additional easyblocks, toolchains, etc.
- **reproducibility** of the installation as one of the major design goals
- **comprehensively tested**: lots of unit tests, regression testing, …
- actively developed, **collaboration** between various HPC sites (worldwide **community**)
- extensive **transparency** through verbose dry-runs and preview of all steps involved in the installation.
EasyBuild terminology

- EasyBuild **Framework**
  - core of EasyBuild: Python modules & packages
  - provides supporting functionality for building and installing software

- easyblock
  - a Python module, ‘plugin’ for the EasyBuild framework
  - implements a (generic) software build/install procedure

- easyconfig file (*.eb)
  - build specification: software name/version, compiler toolchain, etc.

- compiler **toolchain**
  - compilers with accompanying libraries (MPI, BLAS/LAPACK, …)
EasyBuild vs Spack

### EasyBuild
- **Focus on Linux HPC systems**
  - Smooth integration with environment modules
- **Reproducibility**
  - Version pinned recipes
  - Avoids using system compilers
- **Stability**
  - Regular releases (with package testing)
    - 52 releases since 2012
  - Popular toolchains are validated worldwide
- **Production ready**
  - CSCS only required minor contributions to framework:
  - ![gppezzi](image)
    - 11 commits / 65 ++ / 24 --
- **Main limitations**
  - “How do I use a compiler and MPI combination that is not available on EasyBuild?”
  - Duplication:
    - Number of easyconfig files grows too fast!

### Spack
- **Multiplatform:** Also supports MacOS
  - Modules support was recently added
- **Flexibility**
  - Generic ‘version-less’ recipes
  - Good support for using system compilers
- **Bleeding edge**
  - 11 releases
    - v0.10.0: “With the next release, we will begin to run package tests in addition to unit tests.”
- **Can be used as package manager**
- **Main limitations**
  - Requires Python knowledge
  - “How do I know if the recipe is actually tested/optimized for my setup?”
    - Example: recipes on Cray that do not use libsci
One-Slide guide for creating EasyBuild recipes

- (1) Choose a toolchain
  - `eb --list-toolchains`
  - Most popular: foss (GCC/OpenMPI) & intel

- (2) Choose an easyblock
  - Most popular @ CSCS:
    - ConfigureMake, CMakeMake & CmdCp

- (3) Get inspired by recipes with similar build steps (Copy&Paste 😊)
  - Search by keyword: [https://github.com/hpcugent/easybuild-easyconfigs](https://github.com/hpcugent/easybuild-easyconfigs)
  - Search by app name: `eb --search 'software-name'`

- (4) Add dependencies
  - Create recipes for the missing ones
Easyconfig file example: freetype

name = 'freetype'
version = '2.6.2'

homepage = 'http://freetype.org'
description = """FreeType 2 is a software font engine that is designed to be small, efficient, highly customizable, and portable while capable of producing high-quality output (glyph images). It can be used in graphics libraries, display servers, font conversion tools, text image generation tools, and many other products as well.""

toolchain = {'name': 'CrayGNU', 'version': '2015.11'}

source_urls = [GNU_SAVANNAH_SOURCE]
sources = [SOURCE_TAR_GZ]

dependencies = [('libpng', '1.6.21')]

sanity_check_paths = {
    'files': ['bin/freetype-config', 'lib/libfreetype.a', 'lib/libfreetype.%s' % SHLIB_EXT, 'lib/pkgconfig/freetype2.pc'],
    'dirs': ['include/freetype2'],
}

moduleclass = 'vis'
Easyconfig file example: freetype

name = 'freetype'
version = '2.6.2'

homepage = 'http://freetype.org'
description = "FreeType 2 is a software font engine that is designed to be small, efficient, highly customizable, and portable while capable of producing high-quality output (glyph images). It can be used in graphics libraries, display servers, font conversion tools, text image generation tools, and many other products as well."

toolchain = {'name': 'CrayGNU', 'version': '2015.11'}

source_urls = [GNU_SAVANNAH_SOURCE]
sources = [SOURCE_TAR_GZ]

dependencies = [('libpng', '1.6.21')]

sanity_check_paths = {
    'files': ['bin/freetype-config', 'lib/libfreetype.a', 'lib/libfreetype.%s' % SHLIB_EXT, 'lib/pkgconfig/freetype2.pc'],
    'dirs': [include/freetype2],
}

moduleclass = 'vis'
Piz Kesch & Escha use case
(MeteoSwiss / Cray CS-Storm)

“Kesch” and “Es-cha” consist of identical systems (production and failover), each comprising:

Cray CS-Storm: 12 nodes
- 2 x Intel Haswell E5-2690v3 2.6 GHz 12-core CPUs per node
  - total of 24 E5-2690v3 processors
- 256 GB 2133 MHz DDR4 memory per node
  - total of 3 TB
- 8 NVIDIA® Tesla® K80 GPU devices per node
  - total of 192 GPUs

MeteoSwiss, the Swiss national weather forecasting service, hosts their dedicated production systems at Cray CS-Storm at CSCS, Lugano.
Using EasyBuild and Continuous Integration for Deploying Scientific Applications on Large Scale Production Systems

<table>
<thead>
<tr>
<th>Model</th>
<th>Cray XC50/XC40</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC50 Compute Nodes</td>
<td>Intel® Xeon® E5-2690 v3 (Haswell) @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB</td>
</tr>
<tr>
<td>XC40 Compute Nodes</td>
<td>Intel® Xeon® E5-2695 v4 (Broadwell) @ 2.10GHz (18 cores, 64/128 GB RAM)</td>
</tr>
<tr>
<td>Login Nodes</td>
<td>Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz (10 cores, 256 GB RAM)</td>
</tr>
<tr>
<td>Interconnect Configuration</td>
<td>Aries routing and communications ASIC, and Dragonfly network topology</td>
</tr>
<tr>
<td>Scratch capacity</td>
<td>6.2 PB (Lustre / Sonexion 3000)</td>
</tr>
</tbody>
</table>
Using EasyBuild and Continuous Integration for Deploying Scientific Applications on Large Scale Production Systems

**Piz Daint**

- #8 Top 500
  - #1 in Europe
  - 9,779.0 PFLOPS

- #2 Green 500
  - 7453.5 MFLOPS/W
Table of Contents

1. Introduction: Basic concepts & tools
   - Automating builds for HPC (EasyBuild, Spack, Nix, Guix)
   - Continuous Integration (CI) & Testing (Jenkins, Travis, Bamboo, CDash)
   - Collaborative development tools (Github, Gitlab, Bitbucket)

2. EasyBuild in a nutshell
   - Feature highlights
   - Production systems at CSCS running EasyBuild

3. Combining EasyBuild + CI + Github for testing and deploying builds
   - Overview of Piz Daint production setup
Automated deployment of scientific software

Production repository:

- All CSCS EasyBuild recipes
- List of supported recipes on Daint

Monitor changes

Deploy new recipes

Jenkins
Workflow for installing production software on Piz Daint

1. Create git branch with new EasyBuild recipe
2. Open a Pull request
3. Jenkins tests the recipe on Piz Daint
4. Recipe gets reviewed
5. Change requested?
   - Y: Recipe is merged to repository
   - N: Propose fix
8. Y: Jenkins deploys the software on production
   - N: Start feedback process
   - N: Propose fix
9. passes?
   - Y: Start feedback process
   - N: Propose fix

Pull Request example: CP2K (1/2)

Easyconfig file for CP2K with PLUMED plugin #102

Merged lucamar merged 2 commits into master from cp2k_plumed 21 days ago

Conversation 2 Commits 2 Files changed 1

lucamar commented 22 days ago

I had to shorten the BUILDPATH to build successfully:

```bash
export EASYBUILD_BUILDPATH=/dev/shm/lucamar
```

Easyconfig file for CP2K with PLUMED plugin

9fa4054

gppezzi commented 22 days ago

How about adding the buildpath fix as a comment to the easyconfig file?

Adding comment on buildpath

b1d05a4
Pull Request example: CP2K (1/2)

Easyconfig file for CP2K with PLUMED plugin #102

Merged lucamar merged 2 commits into master from cp2k_plumed 21 days ago

Conversation 2  Commits 2  Files changed 1

lucamar commented 22 days ago

I had to shorten the BUILDPATH to build successfully:

```bash
export EASYBUILD_BUILDPATH=/dev/shm/lucamar
```

Easyconfig file for CP2K with PLUMED plugin

gppezzi commented 22 days ago

How about adding the buildpath fix as a comment to the easyconfig file?

Adding comment on buildpath

Approved by reviewer

Tested by Jenkins passed
Pull Request example: CP2K (2/2)

ทย lucamar requested a review from gppezzi 21 days ago

ทย lucamar self-assigned this 21 days ago

gppezzi approved these changes 21 days ago

ทย lucamar merged commit 48de0fb into master 21 days ago

1 check passed
Jenkins + Github Integration: Jenkins server setup

- Setup remote slaves
  - By default jenkins will run jobs on the same host as the apache server
    - If you wish to perform builds on a remote host, you need to add ‘slave nodes’
    - Manage Jenkins > Manage nodes > new node

- Install Pull request builder plugin
  - Manage Jenkins > Manage Plugins > Available

- Github integration setup
  - Create a repository for archiving your easybuild recipes
  - Create a ‘jenkins’ user on github add to your repository with write permissions
    - (you don’t want to use your private Github account)
  - Jenkins > Manage Jenkins > Github Pull Request Builder
    - Add/test credentials
Jenkins + Github Integration:
Jenkins project setup

- Create a new ‘multi-configuration’ project
- Section “Source code management”
  - Add your github repository to “Repository URL”
  - Branch Specifier (blank for ‘any’): ${ghprbActualCommit}
- Section “Build Triggers”
  - Enable GH PR builder and set admin lists (github users white list)
- Section “Configuration Matrix”
  - Select node(s) under Slaves > Node/label
- Section “Build”
  - Add build step > Execute shell
    1. Select eb files to build, for example comparing with master for new/modified .ebs
       ```
       eb_list=$(git diff origin/master..HEAD --name-only --oneline --no-merges --diff-filter=ACMRTUXB)
       ```
    2. Setup EB and build
       ```
       for ebfile in $eb_list ;do
           eb $ebfile -r --force
       done
       ```
Thank you for your attention.