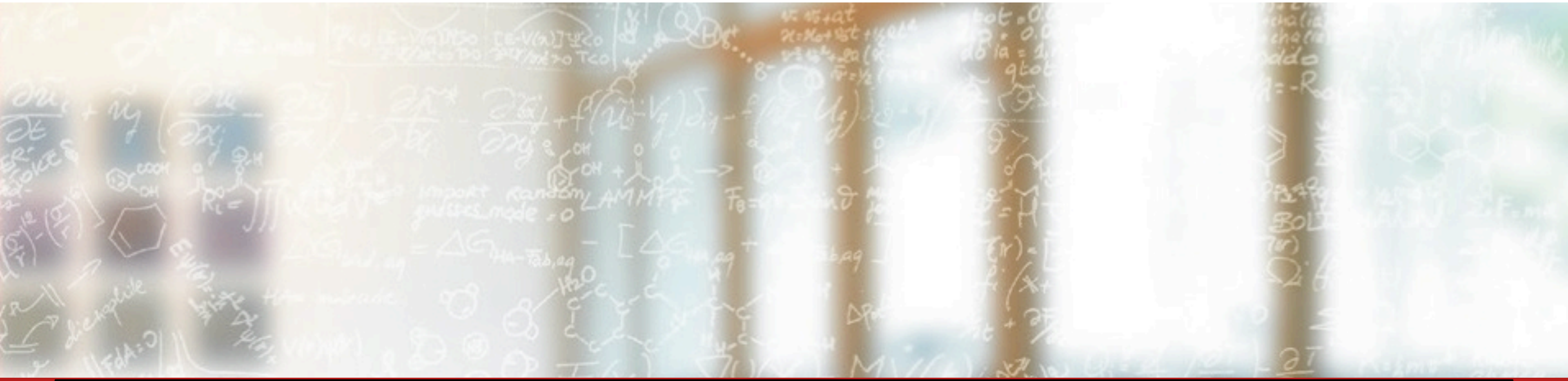
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

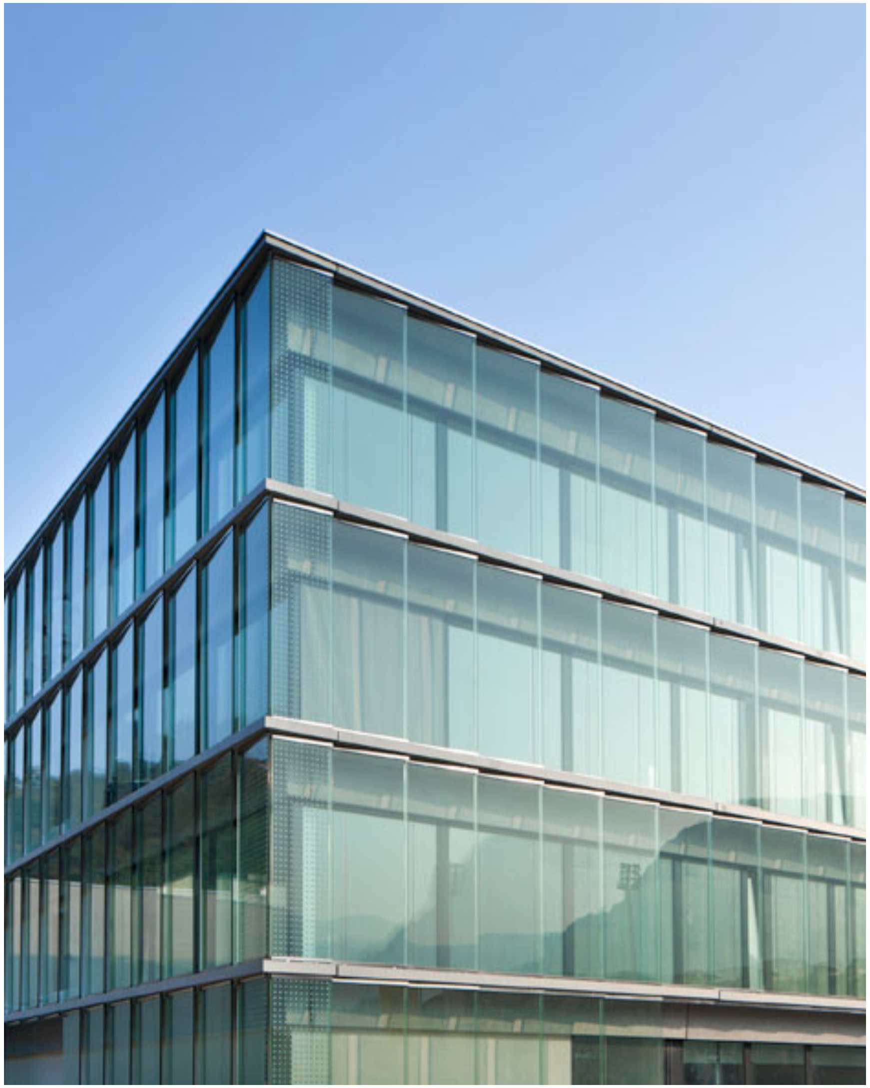


Automatic Building & Continuous Validation CSCS Roadmap

Guilherme Peretti-Pezzi
User Engagement and Support

11.06.2015 hpc-ch forum

Outline



- Background
- Automatic Building tools for HPC
- Continuous Validation
- CSCS preliminary use cases
- Hands on Demo

Evolution of requirements for scientific applications

- A typical requirement workflow in a galaxy not so far away
  - Application: Fortran / C
  - Numerical library: Fortran
  - Parallel API: MPI

- Nowadays
  - Application
    - Fortran / C / C++ / Python
  - Numerical libraries
    - Fortran / C
    - Python + scipy, numpy, matplotlib, h5py, mpi4py, fipy, netCDF4 + (...)
    - R + Modules
    - I/O libraries HDF, netcdf
  - Parallel API
    - MPI / PGAS / CUDA / OpenMP / OpenACC

Current goals @ CSCS

- Minimize the maintenance overhead
  - Number of supported applications is constantly increasing
  - Upgrades are becoming a headache
- Provide a uniform set of libraries and applications
  - On ALL of our systems
- Improve QoS by adopting continuous validation techniques
  - Unit/Non-regression testing
- Provide a continous integration platform to the users
  - Ease the development and improve quality of scientific software
- Challenges
  - Scale (number of systems, users and applications)
  - Implement consistent procedures within and across large teams

Standard package management tools

- Popular examples
  - Yum (RPM)
  - Deb
  - Homebrew
  - Portage
  - pkgsrc
  - Nix
- Known issues
  - Lack of isolation
  - Not adapted for handling multiple versions of the same software
  - Limited support for automatic building
  - Reproducibility
  - Lack of integration with modules environment
  - Error-prone (manual/interactive steps)

Smithy

- ORNL
- Ruby-based
- 100+ of supported packages
- Pros
  - Simplicity
    - One file describes all the building process
  - Archictecture agnostic
- Cons
  - Builds are described as a program-like syntax
  - Lack of community / users outside ORNL

Spack

- LLNL
- Python based
- 100+ of supported packages
- Pros
  - DSL for describing dependencies/workflow
    - Dependencies can be described without hardcoding version number
  - Archictecture agnostic
- Cons
  - Lack of community / users outside LLNL

EasyBuild

- HPC-Ugent
- Python based
- 600+ of supported packages
- Pros
  - Two-level dependency description
    - Easyconfig files for standard builds
    - Easyblocks (python code) for more complex
  - **Growing community**
    - sciCORE/UniBas, Flemish Supercomputer Centre, Julich Supercomputer Centre, Stanford Univ., Univ. of Auckland, Bayer AG, Texas A&M, IMB (Austria), Univ. of Luxembourg, Cyprus Institute
- Cons
  - One easyconfig file maps to one specific software and toolchain version
    - Thousands of files..
  - Focused on building toolchains from scratch
    - Non trivial procedure for creating/extending toolchains
  - Version 2.1.0 (03/2015) provides limited Cray support

Overview of Automatic Building tools for HPC

	Smithy	Spack	EasyBuild
Language	Ruby	Python	Python
# of packages	150+	100+	600+
High level Input file	No	Yes	Yes
Module file Generation	Yes	Yes	Yes

Testing EasyBuild 2.1.1 @ CSCS (as of 11.06.2015)

- Linux x64 clusters (pilatus, castor)
  - OK using foss-2015a toolchain and a couple of stock easyconfig files

- Cray XC30/XC40 (Piz Daint, Santis, Dora)
  - OK for stock Python-2.7.9 using CrayGNU toolchain
  - Created easyconfig file for Python-3.4.3
    - Modules scipy, numpy, matplotlib, mpi4py, virtualenv, pip, …

- Cray MeteoSwiss (Early access)
  - Requirements: GCC 4.8.2 + MVAPICH2
  - PrgEnv-gnu not available
  - Falling back to EB gmvapich2 toolchain
    - Updated to MVAPICH 2.1
  - Next steps: Intel + PGI

Continuous Integration/Validation Tools

- Cdash
  - From Cmake developers

- Bamboo
  - From Atlassian

- Jenkins
  - Formerly Hudson
  - Widely used

1st use case @ CSCS: Automatic builds

- Idea:
  - Regular builds (test) and automatic install on all systems
- EasyBuild
  - Cray: Python/2.7.9 + 3.4.3 + modules,
  - Gnu: Bison/3.0.2,flex/2.5.39, Cmake/3.0.0, HDF5/1.8.15, netCDF/4.3.3.1
- Jenkins

All | **EasyBuild** | +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
		CrayGNU Unit Test on Daint	3 days 0 hr - #1	N/A	13 min	
		EasyBuild Apps on daint	2 days 23 hr - #9	3 days 0 hr - #7	49 min	
		EasyBuild apps on Pilatus	2 days 22 hr - #7	N/A	1 hr 22 min	
		EasyBuild Apps on Santis	3 days 0 hr - #4	N/A	2 min 19 sec	

2nd use case @ CSCS: DCA++

- Idea
  - Provide required libraries with EasyBuild
  - Regularly run builds, unit and performance tests

- EasyBuild (TBD)
  - Libraries: HDF5, NFFT, SPGLIB

- Google Mock + Ctest
  - Unit testing (at application level)
    - Sequential
    - Parallel (MPI)

- Jenkins
  - Github repository integration
  - Automatic build of App + dependencies
  - Automatic execution of unit test
    - Triggered by git commits

3rd Use case @ CSCS: Non-regression suite

- Current state
  - Bash scripts for simple tests
    - Hardware/System
    - Compilation environment sanity
    - Performance
    - Manually triggered

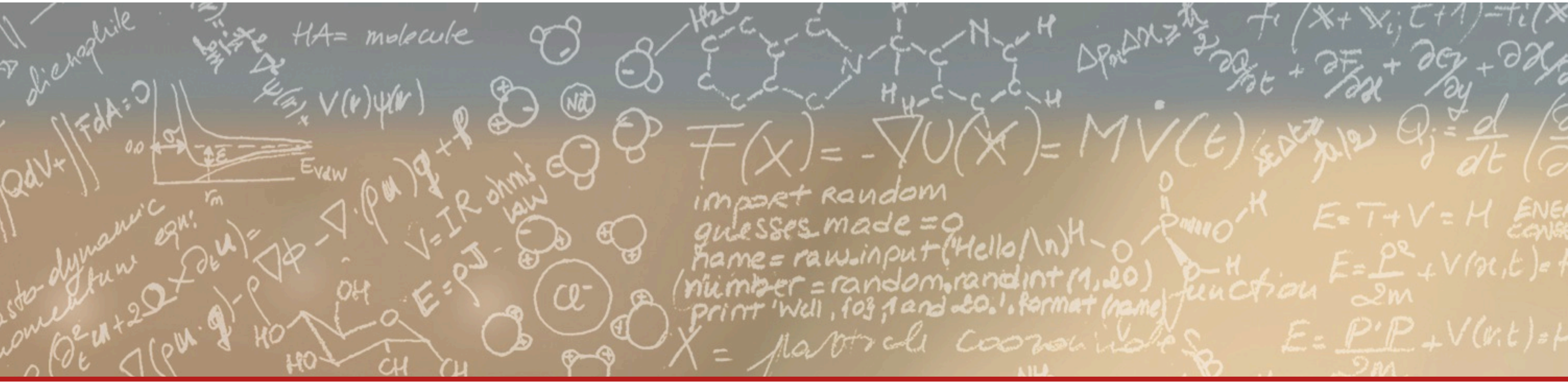
- Next steps
  - EasyBuild + Jenkins integration
  - Extend suite for testing for more complex applications
    - Automatic building
      - To ease the work when upgrading or performing maintenance
  - Monitor “user experience”
    - Periodic performance tests of (key) applications
    - For example: FS and Slurm response times

EB + Jenkins Workflow @ Piz Daint

- Setup EasyBuild
  - $ source $APPS/easybuild/setup.sh
- Create a new EasyConfig file using CrayGNU toolchain
- Build and test application
  - $ eb newApp.eb -r
- Move .eb file to shared folder for nightly builds
  - /apps/common/easybuild/cscs_easyconfigs/
    - CRAY_XC30 (for Piz Daint, Piz Dora and Santis)
    - GNU for non-Cray (Pilatus, Castor, …)
- Day 2
  - Check Jenkins build output on all systems
  - If OK copy new .eb file to automatic deployment folder

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.