The 500 builds of 300 applications in the HeLmod repository will at least get you started on a full suite of scientific applications

Aaron Kitzmiller[1], John Brunelle[2], Michele Clamp[1], James Cuff[3]

[1] Informatics and Scientific Applications, Faculty of Arts and Sciences, Harvard University. Email: aaron_kitzmiller@harvard.edu

[2] Google, Mountain View, California.

[3] Research Computing, Faculty of Arts and Sciences, Harvard University.

Project Website: http://rc.fas.harvard.edu/helmod
Source Code: https://github.com/fasrc/helmod
License: GPL v2.0 (https://www.gnu.org/licenses/gpl-2.0.html)

Scientific research of any significant scale is fundamentally dependent on free, open source software. While much of this software can be downloaded and run relatively easily on single user systems, many tools have complex, parallel or optimized code that must be compiled *in situ*. Large, shared high performance / high throughput clusters have additional deployment challenges.

The Harvard FAS Odyssey cluster accumulated more than 2000 application and library installs managed by a conventional Linux module system in its first 4 years of existence. These module installations had a number of problems including dependency conflicts and irreproducible build conditions. Migration to the TACC Lmod system alleviates module conflicts, but adds additional burdens and does not solve reproducibility issues.

The Harvard Extensions for Lmod deployment (HeLmod) system enhances a standard rpm-based build tool suite with macros and scripts that automate the build of scientific software for compiler and MPI branches of an Lmod deployment. Builds are captured in rpm spec files, and checked in to a freely available github repository. Over the past year, more than 500 builds have been generated for more than 300 applications. By capturing the requirement for reproducible builds, these spec files demonstrate strategies for many challenging software situations including 1) building for multiple compilers, 2) non-standard library locations, 3) commercial software, 4) scripting interactive builds, and 5) patching broken software and build components. These spec files can either be used directly or as a guide to solving troublesome builds in other systems.

Module dependencies and build metadata are captured in easily parsable text files. These files are used to populate web tools for browsing and search and the generation of dependency lists that enable automated rebuilding against new operating systems, compiler versions, etc.

In addition to spec files and rpms, the system is currently being expanded to provide Dockerfiles and images for general use.