Semi-Automatic Hybrid Software Deployment Workflow in a Research Computing Center

Fang (Cherry) Liu
Georgia Institute of Technology
Atlanta, GA, U.S.A.
fang.liu@gatech.edu

Ronald Rahaman
Georgia Institute of Technology
Atlanta, GA, U.S.A.
rrahaman6@gatech.edu

Michael D. Weiner
Georgia Institute of Technology
Atlanta, GA, U.S.A.
mweiner3@gatech.edu

J. Eric Coulter
Georgia Institute of Technology
Atlanta, GA, U.S.A.
j.eric@gatech.edu

Deepa Phanish
Georgia Institute of Technology
Atlanta, GA, U.S.A.
deepa.phanish@gatech.edu

Jeffrey Valdez
Georgia Institute of Technology
Atlanta, GA, U.S.A.
valdez@gatech.edu

Semir Sarajlic
Georgia Institute of Technology
Atlanta, GA, U.S.A.
ssarajlic10@gmail.com

Ruben Lara
Georgia Institute of Technology
Atlanta, GA, U.S.A.
ruben.lara@oit.gatech.edu

Pam Buffington
Georgia Institute of Technology
Atlanta, GA, U.S.A.
pam@gatech.edu

ABSTRACT

The software stack is an essential component in any High Performance Computing (HPC) cluster, and it is critical to optimize the usage of the underlying computing resources while simultaneously providing the best user experience. Georgia Institute of Technology (GT) Partnership for an Advanced Computing Environment (PACE) maintains a heterogeneous system with around 2000 compute nodes across five different clusters on two different schedulers (Torque and Slurm) and two different parallel filesystems (GPFS and Lustre). This diversity calls for multiple software stacks that pose a significant challenge to delivering software support efficiently. Hardware upgrades, filesystem changes, and scheduler replacements all lead to software stack changes ranging from updating the MPI libraries to rebuilding the whole stack. In the past five years, PACE has rebuilt the software stack four times due to these events. To reduce the build time, Spack, GitLab, and ReFrame tools were adopted. Spack, an automatic software building tool, allows us to set up more than half of the software stack quickly; GitLab provides git tags and issues to handle *Dev-Test-Prod* workflow coordination within the team; and ReFrame provides a systematic way to test the whole software stack automatically. By integrating these three tools alongside manual software installations, we have created a semi-automatic hybrid software deployment workflow that is presented in this paper. We share our experience to contribute to the body of work and useful tradition of HPC centers sharing their practices with the community.

PEARC '23, July 23–27, 2023, Portland, OR, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9985-2/23/07.
https://doi.org/10.1145/3569951.3593607

CCS CONCEPTS

• **Software and its engineering** → *Programming teams.*

KEYWORDS

Research Computing Center, HPC Software stack, Spack Tool, automatic deployment, Reframe

ACM Reference Format:
Fang (Cherry) Liu, Ronald Rahaman, Michael D. Weiner, J. Eric Coulter, Deepa Phanish, Jeffrey Valdez, Semir Sarajlic, Ruben Lara, and Pam Buffington. 2023. Semi-Automatic Hybrid Software Deployment Workflow in a Research Computing Center. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3569951.3593607

1 INTRODUCTION

The Partnership for an Advanced Computing Environment (PACE) center at Georgia Institute of Technology operates as a provider for a broad spectrum of research computing resources to the Georgia Tech community (in addition to serving the national research community through collaborations with ACCESS [1] and the Open Science Grid [4, 9]). As a funded center for Georgia Tech, our primary focus always returns to providing quality research computing services, software, and infrastructure to approximately 2000 users we serve locally. We provide access to, and maintain, five High Performance Computing (HPC) systems that includes the Phoenix cluster [18] (ranked #277 on the Top500 November 2020 list), which are each heterogeneous clusters offering a variety of architectures and GPU devices. A key piece in providing quality service revolves around provisioning our systems with up-to-date scientific software that has been compiled with optimizations to match the architectures of all the hardware available on our local systems.

While system-level software management necessary to run an HPC system and other types of Cyberinfrastructure (CI) is itself a topic worthy of deep study, it is an area well served by many existing tools from Enterprise-grade Linux distributions, configuration management systems, and support contracts with scheduler providers; building the software itself correctly is generally not a deep concern. On the other hand, scientific applications are a diverse set of research software, developed by the community according to a wide variety of standards and practices. Providing these applications to users in a production environment is an ongoing challenge.

This is not to imply that the scientific software support ecosystem is utterly lacking in solutions to our common problems! There are many well-loved community tools that we utilize in our current workflow, e.g. Spack [13], Lmod [28], ReFrame [7] and GitLab Continuous Integration/Continuous Deployment (CI/CD) tools[5]. The research computing community has a long-standing tradition of sharing solutions and best practices that help deal with the many common problems presented by supporting computational researchers. Many of these efforts employ similar toolchains, such as Lmod and Easybuild[15], the RESIF framework [39] (which is actually a wrapper *around* EasyBuild and Lmod). Others, such as the in-house workflow at The Ohio Supercomputing Center[19], are closer to our own system, which also utilizes Git and ReFrame. Other sites have implemented more novel tools for the HPC world, such as Jenkins+Singularity at the University of Colorado[30] or the purely functional software management tool Nix[11] at the GRICAD HPC Center[6]. We *do not* assert that our site's entire workflow represents absolute best practice. On the contrary, our workflow continually evolves due to our improved knowledge. We hope some components of our workflow will be helpful to other centers. Most importantly, we hope that, by publishing our workflow and discussing the actual time taken to build packages across different iterations, we can contribute to and encourage the continued open exchange of ideas among HPC centers.

This work dives deeply into the motivation, policies, and tooling that represents our current workflow, which has risen out of continued community support needs, a transition from the Torque scheduler to Slurm, a growing team, and a growing stable of different types of hardware. We also discuss the amounts of time needed to build (and re-build) the software stack, which is rarely done in similar works.

2 HISTORY AND MOTIVATION

The HPC scientific software stack is critical for optimally using the available computing resources on a cluster, making software support inevitable in any Advanced Research Computing (ARC) center. In view of this, GT's PACE center provides a full software stack and necessary support for GT researchers and their collaborators. In the past, all software were built manually, and the *rsync* Linux command was used to deploy it onto the network-mounted filesystem so that each compute node could access the stack. The person who built the software was able to test and verify it only on the compilation machine, and hence, there was no cross-validation or queue-based testing before deployment.

The last complete manual build on PACE was done in 2018 when the system was upgraded from RHEL 6 to RHEL 7.4. This was accomplished with two full-time staff and multiple undergraduate students. Among 200 pieces of software in consideration, two-thirds were built in a span of five months before releasing them to the users. Due to the time-consuming nature of the workflow, we had to prioritize the installations using the high-priority software list generated from our previous work [3, 27]. Further software installations were handled based on requests from the users. In the process, we trained student assistants to learn the HPC system by compiling, installing, and testing the software. The stack thus deployed was used until the beginning of 2020.

After finishing migration to the RHEL 7.4 stack, we needed to build yet another software stack for new hardware in our new CODA [26, 35] data center, where we moved in the spring of 2019. We used this opportunity to adopt new tools to increase the team's productivity and reduce the time needed to establish a software stack on a bare-metal machine. We began two initiatives in this regard: 1) We chose the Spack tool [13], which aims to seamlessly manage scientific software dependencies. 2) We also decided to switch from TCL [31] modules to Lmod [24], since Lmod provides for easy handling of the MODULEPATH hierarchy problem. We successfully built a RedHat 7.6 stack when the new infrastructure was migrated to the new CODA data center. The stack contains 239 unique applications, with 115 applications built manually. Using the Spack tool for half of the software dramatically reduced the labor required. The new stack was released to users in January 2020 and is still actively used by GT researchers. PACE staff have been actively following Spack development and participating in the Spack Slack, user groups, and BoF sessions in conferences since 2019, and have built local expertise to support GT researchers on a daily basis. Also, using Spack has made it easier to port certain complex software suites — U.S. Department of Energy (DOE)'s Energy Exascale Earth System Model (e3sm) [29] for example — to PACE in a customized Spack branch.

When releasing the software stack in 2020, we faced the challenge of a rapidly growing Spack tool. In order to accommodate new software, we made multiple Spack-built software stacks co-existing through a carefully aligned Lmod system.

In 2022, we had to rebuild our software stack from scratch to facilitate our scheduler migration from Torque/Moab to Slurm. The center's lead research software engineer, Fang (Cherry) Liu, led the project with two new team members and a student assistant. In contrast to previous timelines, the whole process took just about a month to set up the initial software stack and complete sanity tests with the Reframe [7] suite. Research Computing Facilitator (RCF)s and Research Software Engineer (RSE)s completed additional software installation based on user requests. To date, 175 applications are installed, out of which 108 are built with Spack.

Not long after the Slurm stack was released, we needed to build yet another stack for newly-bought nodes, which added AMD CPUs and Nvidia GPUs to an Intel-based instructional cluster. This particular cluster was still on the Moab/Torque scheduler, since PACE is migrating systems gradually. For this addition, we had to build a separate AMD stack as the newly-built Slurm software stack cannot yet be deployed here, and the Torque stack — last built in 2020 and highly optimized towards Intel Cascade Lake — was incompatible with AMD nodes. Following our new process, we quickly rebuilt the AMD stack with basic compiler and HPC libraries using the Spack tool, providing a total of 53 packages. Since this is only a temporary solution until the instructional clusters' Slurm migration, we did not add any manually-built software into this stack.

The history above shows the dynamic nature of software management within an HPC center and the importance of preparedness for a continuously changing environment. Our center's experience in adopting Spack, along with modern software deployment practices, has increased our work efficiency and team collaboration by a great deal. The details of our newly- designed workflow are presented in Section 4. The transitions we have undertaken pave a clear path to further refine the usage of Spack for better serving PACE's mission. The best practice we learned through the past of years leads us to define the future road map in section 5.

3 SOFTWARE POLICY

PACE publishes a software policy[33] detailing how requests for additions to the software stack are handled. The policy ensures a sustainable, functional software stack is available across PACE's clusters. The software policy helps us to provide quality software support with minimal staff time.

In order for software to meet the criteria for installation, software should benefit at least five cluster users. It must be compatible with the operating system in use - currently Red Hat Enterprise Linux 7 - and may not require system-level changes (alternatively, a container may be employed). We install only current production code, not experimental or abandoned projects, in order to ensure the software can be maintained. Most packages are Free and Open-Source Software (FOSS), while some are licensed at the campus level or by certain faculty, in which case access may need to be restricted to those users with licenses.

In order to prioritize support and documentation efforts, PACE sorts software into tiers. Our data-driven analysis[3] maximizes the return on these efforts by analyzing historical records of batch jobs. "Tier 1" software, defined to cover all batch software needs of 75% of researchers on the cluster, is installed by us and the most widely used. These packages receive detailed usage guides in our documentation[32]. Approximately 30% of available packages fall in this category.

"Tier 2" software extends full coverage to 98% of researchers, representing another 60% of packages. Software in this tier is also installed and maintained by PACE and receives best-effort support. Documentation is provided for many of these packages as well, but at a lower priority.

Additional software not covered by the first two tiers is installed only upon request and is not automatically included when a full rebuild of the software stack occurs. These requests are evaluated based on the guidelines described above. Interactive software packages, including some that are used only through a GUI, may not appear in analysis of batch jobs and are therefore excluded from the tiers. These are also re-installed in new software stack builds upon request from researchers and receive priority if they were offered in the prior stack to avoid disruption to ongoing workflows. The more popular ones receive full documentation like Tier 2 software.

If a requested package does not meet the criteria for inclusion in our software stack, we provide assistance as needed with a self-installation in the researcher's own directories or in a shared directory for a research group.

We aim to provide a whole software stack refresh every two years to meet either OS upgrades or software tool upgrades requirements.

4 OVERALL WORKFLOW

For any brand-new stack, we start out by building compilers and Message Passing Interface (MPI) libraries using Spack in two flavors: the Mvapich2 [38] MPI library built with the GCC[10] compiler; and Mvapich2 built with the Intel compiler. Next, we build foundational libraries, e.g. HDF5 [16], NetCDF [37] and FFTW [12]. Because many software packages are not compatible with the Intel compiler, we built most of the software on the GCC stack. In the future, we plan to compare GCC and Intel-built software performance and optimize for better performance.

The end-to-end workflow shown in Figure 1 is a five-step process for ensuring a high-quality software stack before it goes into production. Each step of the workflow contains both *automated* and *manual* components.

Our workflow allows six people to work together asynchronously on the same software stack. This is uncommon in Advanced Research Computing (ARC) support, as in most cases, only one or two people take care of the entire software stack. Due to a restructuring of the PACE team, the addition of several new members, and resource requirements spurred by the scheduler migration, many new PACE RCFs began installing software in 2022. This meant that organized training efforts were needed. Knowledgeable team members offered training sessions, helped new colleagues work through examples, and provided guidance for any questions that arose, all while building a pool of talent. We created a clear, step-by-step documentation for how and where to install new packages, create modules, manage placement in the repository, and test before deployment. Most training initially focused on manual installations, with only a few working on Spack packages.

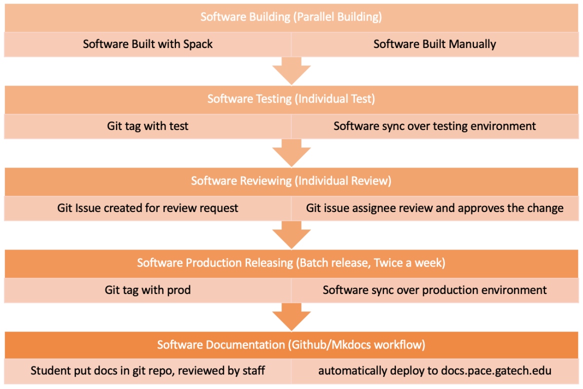

Software Building (Parallel Building)
Software Built with Spack
Software Built Manually
Software Testing (Individual Test)
Git tag with test
Software sync over testing environment
Software Reviewing (Individual Review)
Git Issue created for review request
Git issue assignee review and approves the change
Software Production Releasing (Batch release, Twice a week)
Git tag with prod
Software sync over production environment
Software Documentation (Github/Mkdocs workflow)
Student put docs in git repo, reviewed by staff
automatically deploy to docs.pace.gatech.edu

Figure 1: High-Level PACE Software Deployment Workflow

We follow five steps for each installation:

- The *Software Building* step includes the initial stack-building task and further distributing incoming user software requests among multiple team members. While the former mainly uses the Spack tool, the latter requests can be built either using Spack or manually on a case-by-case basis. One major criterion is the ease of installation: software with dependencies that are hard to resolve in Spack are moved to manual installation. Further, software that need timely addition of new sub-packages, e.g., LAMMPS [36], and software natively built with Conda [2], proceed with a manual route, the details of which are covered in section 4.1.

- The *Software Testing* step integrates the best practices in software Dev/Ops with git repositories. In our workflow, each individual software — either manually built or on the Spack stack — is in a separate git repo. When a new repo is built and ready for testing, we tag the repo with a suffix of *test*, so our custom-built tool synchronizes the given repo tag to the test environment as detailed in section 4.2. For testing a newly built initial Spack stack quickly, we use the ReFrame tool that can test around 77 software packages efficiently. A detailed description is in section 4.3.
- The *Software Reviewing* step allows the software installation to be peer- reviewed after testing. The requirements for this step are two-fold: 1) creating a git issue; and 2) assigning the git issue to another team member for review.
- The *Software Production* step tags the git repo with a suffix of *prod* and our custom-built tool synchronizes the given repo tag to the production environment. Further details are provided in section 4.2.
- The *Software Documentation* is mainly done by undergraduate assistants. We have a git repository for documentation that uses MkDocs [34] to generate the HTML web pages on a public website [32].

4.1 Hybrid Software Stack Building

We introduced Spack into PACE's software stack in 2019 and have successfully rebuilt it three times since then. We hope that our experience can help other centers with daily software management. Spack has many configuration files written in YAML format. Commonly used ones include *config.yaml*, *modules.yaml*, *packages.yaml* and *config.yaml*, which define the customization for each site's building preferences. To keep track of those configurations, we have created a *spack-config* git repository. The git branches allow us to support multiple spack versions and hardware architecture using a naming convention in the format [*arch*]-[*OS version*]-[*Spack version*]. Our previous work [25] demonstrates successful adaptation of Spack on PACE by extracting the Spack configuration files from Spack's central repo. You can find similar success stories at other sites as well [21].

The Spack environment [20] aims to group a set of build specs, allowing easy rebuild. It provides a great way to capture the build's provenance and promotes collaborations. We have utilized night Spack environments in the PACE Slurm stack:

- The **base_gcc_apps** environment builds applications using system default GCC 4.8.5
- The **base_gcc** environment builds GCC 10.3.0 version using the system default GCC, e.g., GCC 4.8.5 on RHEL 7.9
- The **gcc_apps** environment builds non-MPI applications using GCC 10.3.0
- The **gcc_mvapich** environment builds MVAPICH2 2.3.6 with GCC 10.3.0
- The **gcc_mv2_apps** environment builds applications with both MVAPICH2 2.3.6 and GCC 10.3.0
- The **base_intel** environment builds intel-parallel-studio@ cluster.2020.4 with base GCC
- The **intel_apps** environment builds non-MPI applications with Intel 2020.4
- The **intel_mvapich** environment environment builds MVAPICH2 2.3.6 with Intel 2020.4
- The **intel_mv2_apps** environment builds applications with both MVAPICH2 2.3.6 and Intel 2020.4

Figure 2 shows the spack spec for the **intel_mvapich** environment. This spec specifies systemdependent libraries in the *packages* section to avoid duplication. We have also used *when_possible* unify for the *concretizer* section to reduce duplication.

```
spack:
  specs:
  - mvapich2@2.3.6 fabrics=mrail file_systems=nfs,ufs process_managers=slurm +regcache
threads=multiple +wrapperrpath %gcc@10.3.0
    ^libxml2@2.9.13%gcc@4.8.5
    ^findutils@4.9.0%gcc@4.8.5
    ^libpciaccess@0.16%gcc@4.8.5
    ^pkgconf@1.8.0%gcc@4.8.5
    ^slurm@current%gcc@4.8.5
  mirrors: {}
  repos: []
  upstreams: {}
  concretizer:
    unify: when_possible
  view: false
  packages:
    pmi:
      version:
      - current
      buildable: false
      externals:
      - spec: pmi@current%gcc@4.8.5 arch=linux-rhel7-x86_64
        prefix: /opt/pmix/current
    slurm:
      version:
      - current
      buildable: false
      externals:
      - spec: slurm@current%gcc@4.8.5 arch=linux-rhel7-x86_64
        prefix: /opt/slurm/current
    rdma-core:
      buildable: false
      version:
      - 15
      externals:
      - spec: rdma-core@15%gcc@4.8.5 arch=linux-rhel7-x86_64
        prefix: /
```

Figure 2: The intel_mvapich environment spack.yaml (partial)

These night Spack environments empower the team to collaborate during the building process without much dependency between them, increasing productivity through parallel work. Three staff members simultaneously built the Slurm stack in the summer of 2022. One built the base GCC compiler and Non-MPI applications with GCC compilers. The second handled the Intel compiler and Intel-built Non-MPI applications. Once the base compilers were ready, the third started to build MVAPICH2 with the GCC compiler and GCC-build MPI applications. The first to finish non-MPI applications took care of building MVAPICH2 with the Intel compiler and Intel-build MPI applications. With three team members working together, it took about one month to install 108 packages using the newest Spack release 0.18.

All Spack configuration and environment specs are stored in the *spack-config* git repo, along with all utility scripts to make deployment of configuration and environment YAML files easier. We also have a wrapper script for building each environment to reduce repeated typing along with was a management script for timing all build steps. The most recent rebuild for our Slurm stack was done on a 24-core Intel Cascade Lake node with 2.70 GHz Xeon(R) Gold 6226 CPUs. We have captured the build time per Spack environment as shown in Table 1. The management script allows us to rebuild the whole stack automatically with just one click.

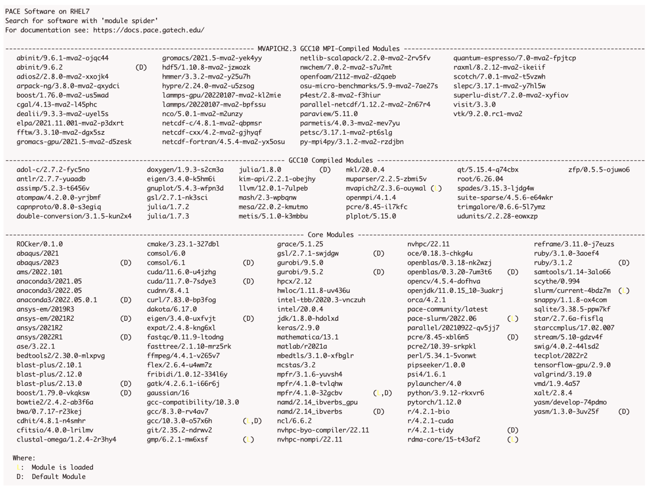
Before the software stack was released to researchers, other software had to be manually built, including licensed software like

Table 1: Spack Stack Building Time in Minutes

Build Target	base_gcc_apps	base_gcc	base_intel	intel_apps	intel_mv2_apps	gcc_apps	gcc_mv2_apps
Build Time	106	84	12	25	147	530	140

Gaussian, Ansys, Comsol, and Matlab. Since our Slurm migration employed a staggered approach, new software requests came in gradually as more compute nodes were moved from Torque to Slurm. We put all source packages in a shared location so that all historical build scripts and source codes are preserved. We continued to assign software installation requests in a round-robin fashion among multiple team members with further evaluation on each software to determine whether Spack was the right tool or not. Among all software requests since the initial release, there were more than 50 packages installed manually and around 10 installed in Spack, demonstrating that a hybrid approach is the best fit for PACE's needs.

We have used Lmod [28] to manage the software module hierarchy. We control package visibility through the folder structure in <*mpi*>/< *mpi-version*>/<*compiler*>/<*compiler-version*>. We also put Spack module files side-by-side with manually-built software module files to keep all module files in the same location. Figure 3 shows a list of currently available software on PACE systems.



```
PACE Software on RHEL7
Search for software with 'module spider'
For documentation see: https://docs.pace.gatech.edu/

------------------------------------------------ MVAPICH2.3 GCC10 MPI-Compiled Modules ------------------------------------------------
   abinit/9.6.1-mva2-ojqc44              gromacs/2021.5-mva2-yek4yy          netlib-scalapack/2.2.0-mva2-2rv5fv      quantum-espresso/7.0-mva2-fpjtcp
   abinit/9.6.2                   (D)    hdf5/1.10.8-mva2-jzwozk             nwchem/7.0.2-mva2-s7u7mt                raxml/8.2.12-mva2-ikeiif
   adios2/2.8.0-mva2-xxojk4              hmmer/3.3.2-mva2-y25u7h             openfoam/2112-mva2-d2qaeb               scotch/7.0.1-mva2-t5vzwh
   arpack-ng/3.8.0-mva2-qxydci           hypre/2.24.0-mva2-u5zsog            osu-micro-benchmarks/5.9-mva2-7ae27s    slepc/3.17.1-mva2-y7hl5w
   boost/1.76.0-mva2-us5wad              lammps-gpu/20220107-mva2-kl2mie     p4est/2.8-mva2-f3hiur                   superlu-dist/7.2.0-mva2-xyfiov
   cgal/4.13-mva2-l45phc                 lammps/20220107-mva2-bpfssu         parallel-netcdf/1.12.2-mva2-2n67r4      visit/3.3.0
   dealii/9.3.3-mva2-uyel5s              nco/5.0.1-mva2-m2unzy               paraview/5.11.0                         vtk/9.2.0.rc1-mva2
   elpa/2021.11.001-mva2-p3dxrt          netcdf-c/4.8.1-mva2-qbpmsr          parmetis/4.0.3-mva2-mev7yu
   fftw/3.3.10-mva2-dgx5sz               netcdf-cxx/4.2-mva2-gjhyqf          petsc/3.17.1-mva2-pt6slg
   gromacs-gpu/2021.5-mva2-d5zesk        netcdf-fortran/4.5.4-mva2-yx5osu    py-mpi4py/3.1.2-mva2-rzdjbn

---------------------------------------------------------- GCC10 Compiled Modules ----------------------------------------------------------
   adol-c/2.7.2-fyc5no               doxygen/1.9.3-s2cm3a    julia/1.8.0          (D)    mkl/20.0.4                   qt/5.15.4-q74cbx             zfp/0.5.5-ojuwo6
   antlr/2.7.7-yuaadb                eigen/3.4.0-k5hm6i      kim-api/2.2.1-obejhy        muparser/2.2.5-zbmi5v        root/6.26.04
   assimp/5.2.3-t6456v               gnuplot/5.4.3-wfpn3d    llvm/12.0.1-7ulpeb          mvapich2/2.3.6-ouywal (L)    spades/3.15.3-ljdg4w
   atompaw/4.2.0.0-yrjbmf            gsl/2.7.1-nk3sci        mash/2.3-wpbqnw             openmpi/4.1.4                suite-sparse/4.5.6-e64wkr
   capnproto/0.8.0-s3egiq            julia/1.7.2             mesa/22.0.2-kmutmo          pcre/8.45-il7kfc             trimgalore/0.6.6-5l7ymz
   double-conversion/3.1.5-kun2x4    julia/1.7.3             metis/5.1.0-k3mbbu          plplot/5.15.0                udunits/2.2.28-eowxzp

-------------------------------------------------------------- Core Modules --------------------------------------------------------------
   ROCker/0.1.0                      cmake/3.23.1-327dbl               grace/5.1.25                      nvhpc/22.11                      reframe/3.11.0-j7euzs
   abaqus/2021                       comsol/6.0                        gsl/2.7.1-swjdgw         (D)      oce/0.18.3-chkg4u                ruby/3.1.0-3aoef4
   abaqus/2023                (D)    comsol/6.1               (D)      gurobi/9.5.0                      openblas/0.3.18-nk2wzj           ruby/3.1.2            (D)
   ams/2022.101                      cuda/11.6.0-u4jzhg                gurobi/9.5.2             (D)      openblas/0.3.20-7um3t6   (D)     samtools/1.14-3alo66
   anaconda3/2021.05                 cuda/11.7.0-7sdye3       (D)      hpcx/2.12                         opencv/4.5.4-dofhva              scythe/0.994
   anaconda3/2022.05                 cudnn/8.4.1                       hwloc/1.11.8-uv436u               openjdk/11.0.15_10-3uakrj        slurm/current-4bdz7m  (L)
   anaconda3/2022.05.0.1      (D)    curl/7.83.0-bp3fog                intel-tbb/2020.3-vnczuh           orca/4.2.1                       snappy/1.1.8-ox4com
   ansys-em/2019R3                   dakota/6.17.0                     intel/20.0.4                      pace-community/latest            sqlite/3.38.5-ppw7kf
   ansys-em/2021R2            (D)    eigen/3.4.0-uxfvjt       (D)      jdk/1.8.0-hdolxd                  pace-slurm/2022.06       (L)     star/2.7.6a-fisflq
   ansys/2021R2                      expat/2.4.8-kng6xl                keras/2.9.0                       parallel/20210922-qv5jj7         starccmplus/17.02.007
   ansys/2022R1               (D)    fastqc/0.11.9-ltodng              mathematica/13.1                  pcre/8.45-xbl6m5         (D)     stream/5.10-gdzv4f
   ase/3.22.1                        fasttree/2.1.10-mrz5rk            matlab/r2021a                     pcre2/10.39-srkpkl               swig/4.0.2-44lsd2
   bedtools2/2.30.0-mlxpvg           ffmpeg/4.4.1-v265v7               mbedtls/3.1.0-xfbglr              perl/5.34.1-5vonwt               tecplot/2022r2
   blast-plus/2.10.1                 flex/2.6.4-u4wm7z                 mcstas/3.2                        pipseeker/1.0.0                  tensorflow-gpu/2.9.0
   blast-plus/2.12.0                 fribidi/1.0.12-334l6y             mpfr/3.1.6-yuvsh4                 psi4/1.6.1                       valgrind/3.19.0
   blast-plus/2.13.0          (D)    gatk/4.2.6.1-i66r6j               mpfr/4.1.0-tvlqhw                 pylauncher/4.0                   vmd/1.9.4a57
   boost/1.79.0-vkqksw        (D)    gaussian/16                       mpfr/4.1.0-32gcbv        (L,D)    python/3.9.12-rkxvr6             xalt/2.8.4
   bowtie2/2.4.2-ab3f6a              gcc-compatibility/10.3.0          namd/2.14_ibverbs_gpu             pytorch/1.12.0                   yasm/develop-74pdmo
   bwa/0.7.17-r23kej                 gcc/8.3.0-rv4av7                  namd/2.14_ibverbs        (D)      r/4.2.1-bio                      yasm/1.3.0-3uv25f     (D)
   cdhit/4.8.1-n4smhr                gcc/10.3.0-o57x6h        (L,D)    ncl/6.6.2                         r/4.2.1-cuda
   cfitsio/4.0.0-lrilmv              git/2.35.2-ndrwv2                 nvhpc-byo-compiler/22.11          r/4.2.1-tidy             (D)
   clustal-omega/1.2.4-2r3hy4        gmp/6.2.1-mw6xsf         (L)      nvhpc-nompi/22.11                 rdma-core/15-t43af2      (L)

  Where:
   L:  Module is loaded
   D:  Default Module
```

Figure 3: User View of PACE Software Stack Module Hierarchy

4.2 Software Stack Deployment

PACE has actively adopted modern software development practices in our daily operational workflow to improve efficiency and ensure quality of service. We mapped the *Dev-Test-Prod* workflow to the software deployment procedure and integrated GitLab with our workflow for productivity and collaboration. Figure 4 shows how software is deployed in the PACE system.

The **Compilation node** serves as a *Dev* environment, where RCFs and RSEs build the software. This node is configured with the same Operating System (OS) stack as the cluster's compute nodes.

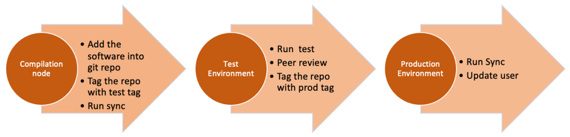


Figure 4: High-Level Software Deployment Workflow

This *Dev* environment also helps us to determine missing system libraries, which would be added to the compute nodes' OS stack later. In our system, Spack-built software are in one git repository; all module files are in a separate git repository; and each manually-built software is kept in a separate git repository. We observed some performance bottle-neck when committing large binary files and a large number of small files into git repository. E.g. it took a long time to commit Matlab and Anaconda. We started to explore some alternatives. After the software is built successfully, the new binaries and its build script are added to the given software's git repository and tagged with the *common.intel.test* tag. Next, the software is synced to the *Test* environment. To make this step more efficient, we have created these in-house tools:

- *pace-set-git-tag* is a bash script to set a tag for the repository with rollback capabilities. It needs to run within the git repository; one can either set an arbitrary tag for the given repository or a specially-formatted <*cluster*>.<*architecture*>.<*deployment*> tag. It saves the existing tag with <tag><current timestamp> for rollback assurance and then creates the new tag pointing to the current head. It can also remove old tags.
- *pace-status-packages* is a bash script doing QA checks for tags. In the past, we had two tags per cluster; five clusters together gave us ten tags. This tool allowed us to easily confirm the tags' correctness and whether they were pointing to the right git commit or not.
- *pace-sync-packages* is a bash script dealing with actual sync to the destination filesystem. PACE puts the software stack on the different volumes for the different stages: the *Dev* volume is on the local disk of the compilation node for quick building, while the *Test* and *Production* volumes are on network-mounted NetApp [14] storage. The following command syncs over all git repositories with the given tag to the *Test* environment: *pace-sync-packages –tag common.intel.test –path /storage/coda-apps/test* One can also choose to sync only one repository at a time, for which the sync script updates only the selected one as well as the modules repository, shortening the time to update.

Once the packages are synced over to the **Test Environment**, the person who installed the software conducts testing first, then creates a GitLab issue with the label *needs- review* along with detailed steps to run the test, and lastly assigns the issue to another person. A second person reviews the installation and changes the git issue label to *reviewed*, after which an RSE changes the git issue label to *prod-sync* and assigns it to the Cyberinfrastructure (CI) team. The use of Git issues, labels, and assignees lays out clear workflows across multiple teams, increasing work efficiency and quality.

Finally, the CI team syncs the new packages to the **Production Environment**, and the assigned RCF or RSE communicates to the user who requested the software, completing the end of the software deployment cycle.

Git and GitLab have increased our workflow efficiency overall, but for some of our packages, Git is not the most efficient method for distribution. We face two inefficient cases: repos with a very large number of small files; and repos with a small number of very large files. These cases are known to be problematic for Git, especially since they fall outside of the original use case for Git (tracking text files for source code in a software project). Fortunately, Git developers have made steady progress in improving both of these cases. For a large number of small files, the inefficiencies result from the index and untracked cache, and performance optimizations can be enabled with the `"feature.manyFiles"` [8] option in Git 2.29 and later. The object database becomes inefficient for a small number of large files, and the best solution is to move large files out of the object database. To support increasing demand for containers at PACE since 2020, a single Singularity images repository was created to install all containers. As there are 50 containers in the repo now, it slowed *git add/commit/status* processes greatly when a new SIF-formatted image is added, as each can be several GB. Git and GitLab offer native support for Git LFS (large file storage), which can store large objects in the LFS cache on the client side and the LFS store on server side. Deployment of Git LFS greatly increased speed for managing our container deployments.

4.3 Verification and Validation

ReFrame[7] is a new framework for writing regression tests for HPC systems. It aims to abstract away the complexity of the interactions with the underlying HPC system, allowing users to write quickly-portable regression tests, focusing only on the software functionality. We have used ReFrame to execute sanity regression tests on packages in the PACE *Test* Environment since 2020, configuring it for 139 different applications. Student assistants created all test cases, and the ReFrame instance has been integrated with our GitLab CI/CD pipeline. Each time a new test is added, new packages are verified in the *Test* environment. There was also a scheduled nightly test. We upgraded the ReFrame test suite with our 2022 stack, and only 77 applications have been included so far. Currently, the test suite can be run manually, and the integration with CI/CD is ongoing, as we integrate more test cases through the efforts of student assistants.

4.4 User-Facing Documentation

Most of our user-facing software documentation is created by student assistants, who are undergraduate students mentored by PACE RCFs. We document the most widely-used packages[3] with full details, include example Slurm scripts to launch jobs and simple example input files that researchers can run to learn how to use the software. Graphical applications are documented with screenshots, usually from our Open OnDemand[17] instance.

The students write documentation in markdown files, and we use mkdocs[34] to build a full html site of user documentation, which includes guides to cluster usage, storage options, policies, and more, alongside software guides. The markdown and html pages are maintained in a git repository, and the built site of html page is synced daily to the web hosting service, making the documentation publicly available on our website[32].

5 FUTURE ROAD MAP

In the future, our software stack must adapt to the increasing heterogeneity of our site's clusters. In response to our stakeholders, we are purchasing nodes with a larger variety of CPU and GPU architectures. We will require strategies to deploy several different builds of the same software with a minimal amount of staff effort. We hope to leverage Spack environments, spec matrices [22], and stacks [23] more effectively.

Currently, we compile our software stack for an instruction set that is common to all our CPUs. Specifically, we limit our software's optimizations to AVX2 instructions, since both our Intel Cascade Lake and AMD Zen 3 processors support it. This vastly simplifies deployment while losing the advantage of the AVX-512 instructions supported by Cascade Lake. To solve this, we plan to list both compiler vendor and CPU target in a spec matrix within a single environment. For our site, our spec matrix would have a spec list with `"gcc target=x86_64_v3"` for both Cascade Lake and Zen 3; and `"intel target=cascadelake"` for Cascade Lake only. It leaves room for integrating the AOCC compiler for our Zen 3 processors by extending the spec list with `"aocc target=zen3"`. For many applications, we will need to deploy separate builds for NVIDIA and AMD GPUs. In this case, separate environments may be more feasible. Not all packages have CUDA and/or ROCm support, so a full spec matrix would need many exceptions listed. While this is a well-supported capability in Spack, we've found it difficult to manage for the wide ranges of GPU support. Instead, it appears more feasible to have one NVIDIA environment that lists packages with only CUDA support and lists `"+cuda"` under `"packages:require:all"`. Likewise, a second AMD environment would list only ROCm packages and `"+rocm"` under `"packages:require:all"`. We also see there is an opportunity to make the production sync step fully automated by adding automatic testing before the scheduled file synchronization.

6 CONCLUSION

An Advanced Research Computing (ARC) facility like PACE can dramatically benefit from innovative tools for daily operational support. This paper details our modern workflow and software deployment life cycle at PACE, where we have demonstrated that automatic software building using Spack is a scalable and sustainable solution. Further, adopting a *Dev-Test-Prod* workflow has enabled parallel work distribution across a growing staff while ensuring high quality of software builds. Testing with ReFrame will further enable us to make the workflow fully automatic in the future. As stated in Section 5, we also plan to refine Spack configurations to continue supporting an increasingly heterogeneous set of hardware.

ACKNOWLEDGMENTS

We are grateful to the entire PACE team for their ideas, contributions, and feedback in this work. Special thanks go to Research Computing Facilitation team members Marian Zvada and Grigori Yourganov for their daily support to our users; and Matt Guidry for supporting production deployments. We also want to thank former PACE Research Software Engineers Nuyun Zhang, Kevin Manalo, and Christopher Stone for their valuable contribution to the 2020 software stack and setting up the software policies and procedures.

REFERENCES

[1] ACCESS. 2022. NSF: An Advanced Computing and Data Resource. https://access-ci.org/.

[2] Inc Anaconda. 2017. Conda: an open source package management system. Retrieved 03.03.2023 from https://docs.conda.io/en/latest/

[3] Mehmet Belgin, Tyler A. Perini, Fang (Cherry) Liu, Nuyun Zhang, Semir Sarajlic, Andre McNeill, Paul Manno, and Neil C. Bright. 2019. A data-driven support strategy for a sustainable research software repository. *Concurrency and Computation: Practice and Experience* 31, 20 (2019), e5338. https://doi.org/10.1002/cpe.5338 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5338 e5338 cpe.5338.

[4] Mehmet Belgin, Semir Sarajlic, Ruben Lara, Laura Cadonati, Nepomuk Otte, Ignacio J. Taboada, Gregory L. Beyer, Norman B. Bonner, Michael Brandon, Pam Buffington, J. Eric Coulter, Aaron Jezghani, David Leonard, Fang (Cherry) Liu, Paul D. Manno, Craig A. Moseley, Trever C. Nightingale, Ronald Rahaman, Kenneth J. Suda, Peter Wan, Michael D. Weiner, Deirdre Womack, Dan Zhou, Marian Zvada, Andre C. McNeill, Neil C. Bright, Robert W. Gardner, Paschalis Paschos, Lincoln Andrew Bryant, Judith Lorraine Stephen, James Alexander Clark, Peter F. Couvares, Brian Hua Lin, Todd Tannenbaum, and Gregory Thain. 2022. Buzzard: Georgia Tech's Foray into the Open Science Grid. In *PEARC22: Practice and Experience in Advanced Research Computing 22*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3491418.3535135

[5] GitLab B.V. 2023. GitLab CI/CD Documentation. https://docs.gitlab.com/ee/ci/.

[6] Bruno Bzeznik, Oliver Henriot, Valentin Reis, Olivier Richard, and Laure Tavard. 2017. Nix as HPC Package Management System. In *Proceedings of the Fourth International Workshop on HPC User Support Tools* (Denver, CO, USA) *(HUST'17)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3152493.3152556

[7] Swiss National Supercomputing Centre. 2016. ReFrame: A framework for writing regression tests for HPC systems. https://reframe-hpc.readthedocs.io/en/stable/.

[8] Git Community. 2021. feature.manyFiles in Git 2.29 Reference Manual. Retrieved 03.03.2023 from https://git-scm.com/docs/git-config/2.29.0#Documentation/git-config.txt-featuremanyFiles

[9] The OSG Consortium. 2023. NSF: Open Science Grid. https://osg-htc.org/.

[10] Free Software Foundation. 2023. GCC, the GNU Compiler Collection. Retrieved 03.03.2023 from https://gcc.gnu.org/

[11] NixOS Foundation. 2020. How the Nix Package Manager Works. https://nixos.org/guides/how-nix-works.html.

[12] Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[13] Todd Gamblin, Matthew LeGendre, Michael R Collette, Gregory L Lee, Adam Moody, Bronis R de Supinski, and Scott Futral. 2015. The Spack package manager: bringing order to HPC software chaos. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Austin, TX, 1–12.

[14] Gartner and Magic Quadrant. 2022. NetApp Storage System. Retrieved 03.03.2023 from https://www.netapp.com/

[15] Markus Geimer, Kenneth Hoste, and Robert McLay. 2014. Modern Scientific Software Management Using EasyBuild and Lmod. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST '14)*. IEEE Press, New Orleans, Louisiana, 41–51. https://doi.org/10.1109/HUST.2014.8

[16] The HDF Group. 2006. Hierarchical Data Formats (HDF5). https://www.hdfgroup.org/solutions/hdf5/.

[17] Dave Hudak, Doug Johnson, Alan Chalker, Jeremy Nicklas, Eric Franz, Trey Dockendorf, and Brian L. McMichael. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622. https://doi.org/10.21105/joss.00622

[18] Aaron Jezghani, Semir Sarajlic, Michael Brandon, Neil Bright, Mehmet Belgin, Gregory Beyer, Christopher Blanton, Pam Buffington, J. Eric Coulter, Ruben Lara, Lew Lefton, David Leonard, Fang (Cherry) Liu, Kevin Manalo, Paul Manno, Craig Moseley, Trever Nightingale, N. Bray Bonner, Ronald Rahaman, Christopher Stone, Kenneth J. Suda, Peter Wan, Michael D. Weiner, Deirdre Womack, Nuyun Zhang, and Dan Zhou. 2022. Phoenix: The Revival of Research Computing and the Launch of the New Cost Model at Georgia Tech. In *PEARC22: Practice and Experience in Advanced Research Computing 22*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3491418.3530767

[19] Samuel Khuvis, Zhi-Qiang You, Heechang Na, Scott Brozell, Eric Franz, Trey Dockendorf, Judith Gardiner, and Karen Tomko. 2019. A Continuous Integration-Based Framework for Software Management. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)* (Chicago, IL, USA) *(PEARC '19)*. Association for Computing Machinery, New York, NY, USA, Article 28, 7 pages. https://doi.org/10.1145/3332186.3332219

[20] Lawrence Livermore National Laboratory. 2013. Spack Environments (spack.yaml). Retrieved 03.03.2023 from https://spack.readthedocs.io/en/latest/environments.html

[21] Lawrence Livermore National Laboratory. 2021. Spack Configuration Files from Different Sites. Retrieved 03.03.2023 from https://github.com/spack/spack-configs

[22] Lawrence Livermore National Laboratory. 2023. Spec Matrices. Retrieved 03.03.2023 from https://spack.readthedocs.io/en/latest/environments.html#spec-matrices

[23] Lawrence Livermore National Laboratory. 2023. Stack Tutorial. Retrieved 03.03.2023 from https://spack-tutorial.readthedocs.io/en/latest/tutorial_stacks.html

[24] J Layton. 2015. Lmod–alternative environment modules. Retrieved 03.03.2023 from http://www.admin-magazine.com/HPC/Articles/Lmod-Alternative-Environment-Modules

[25] Fang (Cherry) Liu, Mehmet Belgin, Nuyun Zhang, Kevin Manalo, Ruben Lara, Christopher P. Stone, and Paul Manno. 2022. ProvBench: A performance provenance capturing framework for heterogeneous research computing environments. *Concurrency and Computation: Practice and Experience* 34, 10 (2022), e6820. https://doi.org/10.1002/cpe.6820

[26] Fang Cherry Liu, Michael D. Weiner, Kevin Manalo, Aaron Jezghani, Christopher J. Blanton, Christopher Stone, Kenneth Suda, Nuyun Zhang, Dan Zhou, Mehmet Belgin, Semir Sarajlic, and Ruben Lara. 2021. Human-in-the-Loop Automatic Data Migration for a Large Research Computing Data Center. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, Las Vegas, NV, 1752–1758. https://doi.org/10.1109/CSCI54926.2021.00068

[27] Fang Cherry Liu, Weijia Xu, Mehmet Belgin, Ruizhu Huang, and Blake C. Fleischer. 2017. Insights into Research Computing Operations Using Big Data-Powered Log Analysis. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact* (New Orleans, LA, USA) *(PEARC17)*. Association for Computing Machinery, New York, NY, USA, Article 31, 8 pages. https://doi.org/10.1145/3093338.3093351

[28] Lmod. 2021. A New Environment Module System. Retrieved 03.03.2023 from https://lmod.readthedocs.io/en/latest/

[29] U.S. Department of Energy. 2023. Energy Exascale Earth System Model. Retrieved 03.03.2023 from https://e3sm.org/

[30] Zebula Sampedro, Aaron Holt, and Thomas Hauser. 2018. Continuous Integration and Delivery for HPC: Using Singularity and Jenkins. In *Proceedings of the Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC '18)*. Association for Computing Machinery, New York, NY, USA, Article 6, 6 pages. https://doi.org/10.1145/3219104.3219147

[31] TCL. 1996. Environment Modules. Retrieved 03.03.2023 from https://modules.sourceforge.net/

[32] GT PACE Team. 2019. PACE Cluster Documentation. https://docs.pace.gatech.edu/.

[33] GT PACE Team. 2020. Introduction to PACE Software Repository. https://docs.pace.gatech.edu/software/softwarePolicy/.

[34] MkDocs Team. 2014. Project Documentation with Markdown. https://www.mkdocs.org/.

[35] Georgia Tech. 2019. Coda at Tech Square. Retrieved 03.03.2023 from https://coda.gatech.edu/

[36] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.* 271 (2022), 108171. https://doi.org/10.1016/j.cpc.2021.108171

[37] UCAR. 2023. Network Common Data Form (NetCDF). https://www.unidata.ucar.edu/software/netcdf/.

[38] The Ohio State University. 2001. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, RoCE, and Slingshot. Retrieved 03.03.2023 from https://mvapich.cse.ohio-state.edu/

[39] Sebastien Varrette, Emmanuel Kieffer, Frederic Pinel, Ezhilmathi Krishnasamy, Sarah Peter, Hyacinthe Cartiaux, and Xavier Besseron. 2021. RESIF 3.0: Toward a Flexible & Automated Management of User Software Environment on HPC Facility. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21)*. Association for Computing Machinery, New York, NY, USA, Article 33, 4 pages. https://doi.org/10.1145/3437359.3465600