SStack: Software Stacks for easier and cleaner software builds on HPC

Strahinja Trecakov
trecakov@nmsu.edu
New Mexico State University
Las Cruces, USA

Nicholas Von Wolff
nvonwolf@nmsu.edu
New Mexico State University
Las Cruces, USA

Mohammad Al-Tahat
tahat@nmsu.edu
New Mexico State University
Las Cruces, USA

ABSTRACT

HPC system administrators and user support teams spend a considerable amount of time on software installation because most of them that come prepackaged in OS package managers may not be optimized for our compute resources and network or have desired compilation features. These installations can be complex due to specific versions of compilers, dependencies, and message passing interface (MPI) libraries that all together create disorganization and are inflexible to manage. This issue is even bigger at smaller size institutions with limited resources that support heterogeneous clusters and have to install software for different hardware configurations to achieve the best utilization and optimization. Moreover, many researchers want the freedom to manage their software stacks and use only package managers with which they are comfortable.

To lower the learning barriers for our users, enable ease of installing software for both site administrators and end users, bring a structured directory tree, and have a hierarchical structure of modules, we introduce SStack. SStack is a tool that enables the management of multiple package managers. It provides an easy way to build software with similar or specific site defaults, keeping the module hierarchy tied together and easy navigation.

CCS CONCEPTS

• **Software and its engineering → System administration**.

KEYWORDS

High-performance computing, package management, system administration, SStack

ACM Reference Format:
Strahinja Trecakov, Nicholas Von Wolff, and Mohammad Al-Tahat. 2024. SStack: Software Stacks for easier and cleaner software builds on HPC . In *In Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2024), November 17–22, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC-W '24, November 17–22, 2024, Atlanta, GA, USA
© 2024 ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 INTRODUCTION

The New Mexico State University (NMSU) supports around 500 users on one HPC cluster called Discovery. It is a heterogeneous cluster that grows in batches and consists of different hardware architectures [29]. Discovery currently has 1536 CPU cores, 32 GPUs, 17.5 Terabytes of RAM, 1.8 Petabytes of usable storage, and an HDR InfiniBand network. Due to a lack of a sustainable budget, our hardware life-cycle goes over 7 years and we have to do major upgrades to the cluster. In 2022, we went through migration and upgraded our cluster from CentOS 7 to RHEL 8 and with that change, all of our software stacks had to be rebuilt. We saw this as an opportunity to restructure how we build packages and create a better and easier process for both our users and system administrators.

Most HPC clusters are built as large Linux-based homogeneous clusters that offer low-latency, high-speed interconnects, the fastest processors on the market, and a great place for new scientific discoveries. Resource allocation and Job Management Systems have critical roles in every large-scale computing cluster. Slurm [30], HTCondor [24], PBS [19], and Torque [10] are the most commonly used schedulers that enable efficient utilization of computing resources. The usual life-cycle of these systems is 5 to 7 years, and during that time, they do not go under any major upgrades and mostly keep running the same major version of the OS. Managing scientific software on such systems is a challenge without a perfect solution. Users of HPCs run a diverse set of applications that have to be compiled and optimized for specific hardware in order to utilize its resources for peak efficiency. The scientific software stack that gets built on these systems usually has the same life-cycle as the hardware; however, new applications and their versions with different build recipes are added constantly [18, 27].

Many package management tools automate parts of the software build process that help system administrators and users keep software stacks up to date and ease up the installation of multiple versions and configurations [2–4, 12, 17, 21]. Not all package managers are able to utilize package builds; some require packages to be installed for each recipe, which creates unnecessary use of storage and management complexity. There are many challenges in the realm of HPC software. Some efforts to speed up and automate software stack re-deployments have already been made [16, 22, 25, 28]. Moreover, the HPC community has been provided with an E4S [20] software stack that helps accelerate the deployment and use of more than 80 popular HPC packages.

In this paper, we address our experience using the SStack tool that we have developed at the NMSU to manage our software stacks and bring more structure for both system administrators and end users. It is common for sites to have multiple installations of the same package managers just so they can avoid dependency, version, and compiler issues. With thousands of scientific packages on a cluster, different versions and configuration builds, usage of different package management tools, and different locations of these builds, it is hard to locate, modify, troubleshoot, and display those modules in a hierarchy where all module paths remain structured. The SStack tool brings us closer to solving the problem of having software installations scattered around a system by creating a structured directory tree and integration with Lmod to give a module hierarchy. This tool provides orchestration and quick deployment of package managers, easy discoverability of modules, and lowers the learning barriers for users. We like to believe that the SStack tool is a "spider web" that links the Wild West of HPC software together.

The remainder of this paper presents the software hierarchy and environment modules, an overview of the SStack tool, main features, use cases, and future work.

2 SOFTWARE HIERARCHY AND ENVIRONMENT MODULES

Software hierarchy is a very important piece on systems that host thousands of packages such as HPC clusters. The software hierarchy typically breaks down software into layers, where the bottom layer consists of hardware and OS as the foundation. Packages that provide systems services are the second layer. The next layer is a collection of software linked using modules that provide higher-level functionality such as user interface, data management, or algorithmic processing. The top layer of the software hierarchy represents packages that provide specific functionalities to end users such as scientific software. These packages are composed of multiple modules that make the whole software system work in unity and without issues.

HPC users want to have the latest complex and dynamic software versions available on clusters for their research. These packages are often challenging to install and maintain because of their dependencies, compilation options, libraries, and MPI distributions. Some applications have a life-cycle longer than the hardware [12], which makes it difficult to keep the software stack stable and up to date. Moreover, users who work on developing software or inspecting application performance [11] have to customize the source code or recompile the same piece of software with different optimization flags to benchmark its performance.

Having different versions of a package brought issues when running or installing new software because an executable has to determine the location of its dependencies and libraries to be able to run. If a package does not find its dependence, it will error out, however, if it finds a wrong dependency it can produce incorrect results. In the software engineering field, a software module is a reusable component of software that consists of all resources needed to perform its function. Modules are self-contained, which benefits us in the development, test, and maintenance phase, and allow us easier management of complex software systems and reuse of packages. Discovery of a large number of installed packages has been addressed by Environment Modules [15]. The Environment Modules system enables users to list packages on the system and *load* and *unload* them into the user's environment. This was revolutionary for users of HPC clusters and was later improved and rewritten in Lua language. The new, advanced Environment Module system has been named Lmod [5, 26] and has multiple essential improvements compared to other module systems.

For years, institutions that run HPC clusters have been working to improve software management overhead and develop tools that would help them in this task. Many package managers have been developed over the past 10 years and have helped system administrators and users install scientific software easier and without worrying much about dependencies[2, 12, 17, 21]. Some of those are widely used within the HPC community and have a large community of contributors. Conda [2] is a package and environment manager that runs on Windows, macOS, and Linux. It enables users to install and update packages and their dependencies in no time as well as to create environments. It supports many languages and is popular for Python packages. EasyBuild [21] is a framework that enables users to manage multiple versions of scientific software at once. It builds and installs software in parallel autonomously, creates module files for all installed packages, and uses Lmod to connect them together. One of the key features is automatic dependency resolution. Spack [17] is a package manager that enables ease of installation, configuration, upgrades, and removal of scientific software. It was designed for HPC environments, but can be used anywhere. Written in Python, Spack offers over 6900 package recipes and a large community of users. Another popular tool is Nix [12], a multi-platform package manager that provides binaries instead of automating builds. Its deployment is set on a directory that keeps the system structure clean. Upgrades to a new version of software do not interfere with the old version.

3 THE SOFTWARE STACK TOOL

Software hierarchy plays an important aspect on every system because by breaking down software into layers, developers, and users can easily test, maintain, reuse, and troubleshoot packages. Tools mentioned in the previous section help us install software quicker, their dependencies and recipes, and easier load/unload different paths and directories into user environments. However, not all of those tools provide an easy way to install common software build tools with sane or site-specific defaults for both system administrators and end users. Also, those tools do not provide an option to represent all software on an HPC cluster in a hierarchical structure. With this challenge in front of us, staff at NMSU's HPC team has developed SStack [1], the Software Stack tool, that is written in Python, and can be used by both site administrators and end users to install software in global(shared) locations and home/project directories. This tool enables the administration of various popular package managers in a simple and clean manner. The module path hierarchy will remain highly structured since it is identical in both locations.

3.1 Important Terminology and the SStack Hierarchy

The SStack tool has been developed to help install different software stacks on HPC clusters. We use below terminology when talking about the SStack.

[1]https://gitlab.com/nmsu_hpc/sstack

- SStack - name of this tool.
- Stack - a single instance or installation of a stack type.
- Stack Type - specific package manager deployed in stack.
- Module - a lua file used by the Lmod Module System.
- Module Tree - a collection of modules. Each SStack data root has a single module tree it organizes everything under.
- SStack Data Root or Path - top-level directory where SStack stores stacks, modules, and a stateful JSON database.

3.2 Installation

Installing the SStack is straightforward. Prerequisites for this tool are:

- Lmod ≥ 8.2.7
- curl, tar, bzip2, git, make
- writable and executable */tmp* and *$HOME* directories

We recommend installing the SStack tool as a stack controlled by sstack because in the backend it automatically integrates SStack with the module system without manual setup of module paths. A helper install script has been provided in our GitHub repository and is listed below. The SStack stack type leverages pixi to handle Python and its libraries. In Figure 1, we demonstrate various configurations of installing the SStack tool using a helper script.

```
# Install SStack (Install to default directory in ~/
     sstack)
curl -fsSL "https://gitlab.com/nmsu_hpc/sstack/-/raw/main
     /share/install_sstack_latest.sh"
| bash -s --

# Install to custom directory
curl -fsSL "https://gitlab.com/nmsu_hpc/sstack/-/raw/main
     /share/install_sstack_latest.sh"
| bash -s -- "/software/sstack"

# Install to custom directory and set custom stack name
## This can be used to reinstall a broken sstack
     installation
curl -fsSL "https://gitlab.com/nmsu_hpc/sstack/-/raw/main
     /share/install_sstack_latest.sh"
| bash -s -- "/software/sstack" "latest-el7"
```

Figure 1: Different install options.

The SStack tool offers a few different command options, such as: *'install'* to install a new software stack; *'modules'* to manage installed module files; *'remove'* to remove an existing software stack; *'show'* to list all installed stacks, available types, etc.; *'update'* to update a software stack and regenerate module file. Moreover, it includes an extensive help system where *'--help'* flag can be added to list different options, arguments, and subcommands.

When it comes to the SStack structure, every SStack instance consists of modules and stacks directories, and a stacks.json file. The modules directory holds lua files of each Stack Type on the system, while stacks directory is the parent location of all different stacks installed. stacks.json file holds all important information about the user's SStack tool. Figure 2 shows the directory tree structure of the SStack tool.

```
data_root (Default:~/sstack/osID_name)
 ├── modules
 │    └── stack_type
 │         └── stack_name.lua
 ├── stacks
 │    └── stack_type
 │         └── stack_name
 │              ├── stack_installation_data
 │              └── ...
 └── stacks.json
```

Figure 2: SStack directory tree structure.

3.3 SStack Types

The SStack tool enables users to have multiple software stacks of different types. Those stack types could be Custom, Conda, EasyBuild, Micormamba, Nix, pkgsrc, pixi, Singularity Registry HPC (SHPC), Spack and SStack.

The custom SStack type allows users to install packages manually while still keeping a tree structure and hierarchy of packages. For example, if we want to build a custom software version named *2023a*, we need to run *'sstack install -t custom -n 2023a'*. The SStack tool creates four directories (builds, modules, packages, sources) under *~/sstack/stacks/custom/2023a/*. In these directories, users can install their software and create module files for corresponding software in the *~/sstack/stacks/custom/2023a/modules/*. directory. To activate the module system to use module files in the mentioned directory, users will need to run *'module use ~/sstack/modules'*. Now, all modules found in the user's *~/sstack/stacks/custom/2023a/modules/* directory will be usable by the module system. The custom SStack type is great for containers.

Conda [2] is an open-source package management tool that enables users to quickly install software and create software environments. These software environments can contain different package versions and dependencies, that enable their users to work without worrying about package conflicts. SStacks makes the Conda module structure easier to navigate and allows users to create a module file for a Conda virtual environment. To do it, users will need to create a Lua file in their modules directory for this SStack type.

Micromamba [6] is a minimalistic distribution of Conda. It can be used to deliver custom Conda and Miniconda deployments with a small footprint and minimal dependencies. Users can easily install Micromamba with SStack and create virtual environments that are compatible with it, just like Conda users. Micromamba has conflicts with Conda, EasyBuild, and Spack, which means that those stacks cannot be loaded with the Micromamba stack.

Another package manager for Linux and Unix systems that SStack supports is Nix [13, 14]. It enables the installation of many software packages and configurations and ensures its reliability by placing its builds in the read-only Nix store with a hash prefix followed by the package name and version. Nix package manager was containerized with Apptainer for portability and made available through SStack Tool.

Pkgsrc is a package manager containing over 26,000 third-party software that is easily deployable on any Unix platform [8]. Binaries produced by pkgsrc can be used without compilation from the source. Using SStack to install pkgsrc binaries is simple and keeps software organized.

Pixi is a cross-platform (Windows, Mac, and Linux) and multi-language (Python, R, C/C++, Rust, Ruby, and many other languages) package manager and workflow tool built on the foundation of the Conda ecosystem [7]. It provides developers with an exceptional experience and enhances the Conda ecosystem with a project-focused approach that goes beyond the traditional emphasis on environments.

SStack works well with EasyBuild [21] and Spack [17] package managers. Both of these were designed to simplify complex builds and installations of software on HPC. There are many similarities between these two tools, however, Spack supports Linux and macOS, whereas EasyBuild Linux and Cray PE. They also differ in the command line interface and many other aspects.

Singularity Registry HPC [9] allows users to use containers that are optimized for performance in an HPC environment. It allows users to download pre-build container images that are ready to run software consistently across different systems. SHPC supports SingularityCE, Podman, and Docker container technologies and Lmod and Environment Modules module systems.

SStack is a tool to manage common HPC package managers, referred to as stacks, such as Spack, Conda, Easybuild, pkgsrc, and many more. It handles opinionated deployment, organization, and integration with these stacks. Lmod modules and hierarchy are leveraged to allow easy loading of these tools and their packages. Furthermore, the SStack module hierarchy allows for package/module searching across multiple stacks and even multiple installations of the SStack data root (home, project, global, etc.. directories).

```
$ module avail

------- /fs1/home/user/sstack/rhel_8/modules -------
  conda/2023a  pkgsrc/2024a  spack/2023a (D)

------- /fs1/software/sstack/rhel_8/modules --------
  conda/2022a  conda/2023a  conda/2024a (D)
  custom/2022a  custom/2023a  custom/2024a (D)
  pixi/2024a  spack/2022a  spack/2023a   sstack/main

----------------- /etc/modulefiles -----------------
  pmix/2.2.5  pmix/3.2.3  pmix/4.1.2 (D)

----------- /fs1/software/sstack/modules -----------
  os/centos_7_test (S)  os/centos_7 (S)
  os/rhel_8_test (S)  os/rhel_8 (S,L,D)

------ /usr/share/lmod/lmod/modulefiles/Core -------
  lmod  settarg
```

Figure 3: Module hierarchy.

4 USE CASES AND FEATURES

We deployed SStack to production at New Mexico State University in August of 2022 and since then we have seen an increase in our users utilizing this tool. The main advantage that the SStack brought to our users is a module hierarchy that is identical no matter of location where it has been installed. Moreover, all the module paths can remain highly structured and are easily discoverable. In figure 3, we represent available modules on a production system, where *conda/2023a* and *spack/2023a* stacks are installed in the user's home directory under *~/home/user/sstack/rhel_8* with the same directory tree structure as described in the previous section.

On the other hand, SStack enables users to distinguish what software has been compiled for what OS distribution. SStack has removed a lot of stress and troubleshooting time our customers had when loading the wrong OS distribution compiled binaries in their user environment. Using Linux container virtualization technology, called Apptainer (previously known as Singularity) [1, 23], we enable users to run deprecated legacy software compiled for CentOS7 run on RHEL8 by treating Singularity Image File(*.sif*) as an executable.

SStack enables site administrators to add software stacks periodically without confusing users about what stack they should load. The tool facilitates a clean and structural representation of all stacks and modules. In Figure 3, we can see Discovery's module hierarchy, different software stacks, and stack types. We have two software stacks, *os/centos_7* and *os/rhel_8* (which is also sticky, loaded, and default). The *os/rhel_8* global software stack has 10 stack types with their names based on the stack type and the year the stack type was created. Since HPC clusters often require multiple package manager installations tailored for specific hardware and/or compilers, these installations should be separated from other installations. The SStack tool supports multiple stacks of the same stack type as we can see in figure 3 as well. There are 6 different stack types out of 13 installed on the system under rhel_8 stack of which 3 are installed locally in the user's home directory and the rest are globally installed. This layout allows us to communicate to our users about a possible deprecation of the global software stack and administration-wise it makes it easy to decommission a stack without worrying about possible dependency issues. Moreover, the utilization of CentOS binaries via container technology becomes less confusing due to the requirement of adding the *os/centos_7* data root to the user's *$PATH* by running *'module load os/centos_7'*.

Figure 4 presents the global SStack directory tree on a production system with 5 different stack types and many stacks. The SStack tool enables users to deploy their software by the released date of the stack or any name that users would like to give to that stack.

4.1 Example of installing Spack stack using SStack tool

The SStack tool enables ease of installation and usage of different package managers as its stacks. Figure 5 shows detailed steps on how to install the Spack stack with stack name *2023a*, load the Spack stack that was previously created, install a *zlib* package using the Spack package manager, and use the *'module spider'* command to locate *zlib* module.

4.2 Module files and search

Module files are easily manageable by using *'sstack modules'* command. This command allows users to manage default modules for

```
~/sstack
├── centos7
│   ├── modules
│   │   └── spack
│   │       └── 2022a.lua
│   └── stacks
│       └── spack
│           └── 2022a
├── rhel8
│   ├── modules
│   │   ├── custom
│   │   │   ├── 2022a.lua
│   │   │   ├── 2023a.lua
│   │   │   └── 2024a.lua
│   │   ├── spack
│   │   │   ├── 2022a.lua
│   │   │   └── 2023a.lua
│   │   ├── pixi
│   │   │   └── 2024a.lua
│   │   ├── sstack
│   │   │   └── main.lua
│   │   └── conda
│   │       ├── 2022a.lua
│   │       ├── 2023a.lua
│   │       └── 2024a.lua
│   └── stacks
│       ├── custom
│       │   ├── 2022a
│       │   ├── 2023a
│       │   └── 2024a
│       │       ├── builds
│       │       ├── modules
│       │       ├── packages
│       │       └── sources
│       ├── spack
│       │   ├── 2022a
│       │   └── 2023a
│       ├── pixi
│       │   └── 2024a
│       ├── sstack
│       └── conda
│           ├── 2022a
│           ├── 2023a
│           └── 2024a
└── stacks.json
```

Figure 4: SStack directory tree structure on the Discovery cluster.

each stack type, regenerate stack module file, and display stack modules.

```
$ module use ~/sstack/modules
$ module load sstack
$ sstack install --name 2023a --type spack
...
Stack Successfully Installed!
+-------+-------+---------+----------------+
| Name  | Type  | Version | Path
+-------+-------+---------+----------------+
| 2023a | spack | 0.18.0  | ~/sstack/stacks/spack/2023a
+-------+-------+---------+----------------+

$ module load spack/2023a
$ spack --version
0.18.0 (c09bf37ff690c29779a342670cf8a171ad1b9233)

$ spack install zlib
[+] /home/username/sstack/stacks/spack/2023a/opt/spack/
     linux-fedora35-x86_64_v3/gcc-11.3.1/zlib-1.2.12-
     ojxmrh7kuyqszihdl5573b4pjadftkld

$ module spider zlib
--------------------------------------------------
  zlib: zlib/1.2.12-ojxmrh7
--------------------------------------------------

    You will need to load all module(s) on any one of the
     lines below before the "zlib/1.2.12-ojxmrh7" module
     is available to load.

      spack/2023a

    Help:
      A free, general-purpose, legally unencumbered
     lossless data-compression
      library.
```

Figure 5: Spack stack installation using SStack tool.

```
$ module spider python/3.10.8-2023a-gcc_12.2.0-hvrnktz

--------------------------------------------------
  python: python/3.10.8-2023a-gcc_12.2.0-hvrnktz
--------------------------------------------------

    You will need to load all module(s) on any one of the
     lines below before the "python/3.10.8-2023a-gcc_12
     .2.0-hvrnktz" module is available to load.

      spack/2023a  gcc/12.2.0-2023a-gcc_8.5.0-e643dqu

    Help:
      The Python programming language.
```

Figure 6: Module spider.

Modules are searchable via *'module spider'* command. For example, if a user searches for a Python package using *'module spider'* command, the Lmod will display different available versions of this package across all stacks on the system. This command also allows users to find more information about this specific package by appending the module's full name at the end of the command. As shown in Figure 6, the Lmod lists what stacks and modules need to be loaded before the Python package. By default, SStack adds the name of the stack *2023a* to the package name for easier recognition and usage. SStack supports multiple data roots and allows loads of multiple packages from different stacks and data roots. Users can use packages that are installed via Spack in their home directories and some that are globally installed simultaneously. This tool has some limits implemented and will prevent loading two stacks simultaneously which could cause dependency issues. On the other hand, if a user modifies the package manager configuration to generate module files in non-default locations, SStack will handle this as long as the package manager is aware of the module location.

To enable the searchability of local and global modules we have added the local path */home/user/sstack/osID_name/modules* and global path */sstack/osID_name/modules* to the $MODULEPATH of each user. The local path is listed first and will take precedence in the module hierarchy.

5 SUMMARY

With the rapid growth of HPC software and the increase in its complexity, management of these software stacks becomes time-consuming and difficult. In this paper, we have reviewed some software management tools and environment modules that are currently state-of-the-art and addressed some of the missing pieces. We introduced SStack, a software stack tool to manage package managers. It provides an ease of building common package managers with sane or site-specific defaults in a structural hierarchy. Moreover, using Lmod, we connected all the package managers and software installed using SStack and made them searchable across multiple stacks. This tool helped novice users manage their software stacks without needing extra help from system administrators to deal with package manager installations, dependencies, module configuration, and discoverability across stacks. SStack is available online at https://gitlab.com/nmsu_hpc/sstack and we would like to encourage other institutions to try it out.

ACKNOWLEDGMENTS

New Mexico State University Discovery is directly supported by the National Science Foundation (OAC-2019000), the Student Technology Advisory Committee, and New Mexico State University and benefits from inclusion in various grants DoD ARO-W911NF1810454; NSF EPSCoR OIA-1757207; Partnership for the Advancement of Cancer Research, supported in part by NCI grants U54 CA132383 (NMSU)). We thank all past and current members of the NMSU's High-Performance Computing Group for their hard work over the years.

REFERENCES

[1] 2024. *Apptainer.* https://apptainer.org/.
[2] 2024. *Conda.* https://docs.conda.io/en/latest/.
[3] 2024. *Hashdist.* http://github.com/hashdist/hashdist.
[4] 2024. *Homebrew.* http://brew.sh.
[5] 2024. *Lmod.* https://lmod.readthedocs.io/en/latest/.
[6] 2024. *Micromamba.* https://mamba.readthedocs.io/en/latest/.
[7] 2024. *pixi.* https://pixi.sh/latest/.
[8] 2024. *pkgsrc.* https://www.pkgsrc.org/.
[9] 2024. *SingularityHPC.* https://singularity-hpc.readthedocs.io/en/latest/.
[10] Inc Adaptive Computing Enterprises. 2023. TORQUE. In *Administrator Guide.* https://support.adaptivecomputing.com/torque-resource-manager-documentation/.
[11] Mohammad Al-Tahat, Strahinja Trecakov, and Jonathan Cook. 2022. AppEKG: A Simple Unifying View of HPC Applications in Production. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS).* 129–134. https://doi.org/10.1109/PMBS56514.2022.00017
[12] 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.* 1–6.
[13] Eelco Dolstra, Merijn De Jonge, Eelco Visser, et al. 2004. Nix: A Safe and Policy-Free System for Software Deployment.. In *LISA*, Vol. 4. 79–92.
[14] Eelco Dolstra and Andres Löh. 2008. NixOS: A purely functional Linux distribution. In *Proceedings of the 13th ACM SIGPLAN international conference on Functional programming.* 367–378.
[15] J. L. Furlani. 1991. Modules: Providing a flexible user environment. In *In Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V).* 141–152.
[16] Todd Gamblin and Daniel S. Katz. 2022. Overcoming Challenges to Continuous Integration in HPC. *Computing in Science & Engineering* 24, 6 (2022), 54–59. https://doi.org/10.1109/MCSE.2023.3263458
[17] 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, 1–12.
[18] Markus Geimer, Kenneth Hoste, and Robert McLay. 2014. Modern scientific software management using easybuild and lmod. In *2014 First International Workshop on HPC User Support Tools.* IEEE, 41–51.
[19] Robert L Henderson. 2005. Job scheduling under the portable batch system. In *Job Scheduling Strategies for Parallel Processing: IPPS'95 Workshop Santa Barbara, CA, USA, April 25, 1995 Proceedings.* Springer, 279–294.
[20] Michael A Heroux. 2019. *The Extreme-Scale Scientific Software Stack (E4S).* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).
[21] Kenneth Hoste, Jens Timmerman, Andy Georges, and Stijn De Weirdt. 2012. EasyBuild: Building Software with Ease. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis.* 572–582. https://doi.org/10.1109/SC.Companion.2012.81
[22] 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 *Practice and Experience in Advanced Research Computing 2019: 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
[23] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PloS one* 12, 5 (2017), e0177459.
[24] Michel J Litzkow, Miron Livny, and Matt W Mutka. 1987. *Condor-a hunter of idle workstations.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
[25] Fang Liu, Ronald Rahaman, Michael Weiner, 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 2023: Computing for the Common Good* (Portland, OR, USA) *(PEARC '23).* Association for Computing Machinery, New York, NY, USA, 68–74. https://doi.org/10.1145/3569951.3593607
[26] Robert McLay, Karl W Schulz, William L Barth, and Tommy Minyard. 2011. Best practices for the deployment and management of production HPC clusters. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 1–11.
[27] 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.* 1–6.
[28] 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: Seamless Creativity* (Pittsburgh, PA, USA) *(PEARC '18).* Association for Computing Machinery, New York, NY, USA, Article 6, 6 pages. https://doi.org/10.1145/3219104.3219147
[29] Strahinja Trecakov and Nicholas Von Wolff. 2021. Doing more with less: Growth, improvements, and management of NMSU's computing capabilities. In *Practice and Experience in Advanced Research Computing.* 1–4.
[30] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing.* Springer, 44–60.