

**Software Deployment Process at NERSC:**  
**Deploying the Extreme-scale Scientific Software Stack**  
**(E4S) Using Spack at the National Energy Research**  
**Scientific Computing Center (NERSC)**

Shahzeb Siddiqui (shahzebsiddiqui@lbl.gov), Sameer Shende (sameer@cs.uoregon.edu)

National Energy Research Scientific Computing Center  
Lawrence Berkeley National Laboratory  
Berkeley, CA 94720

Report No. LBNL-2001458

Office of Advanced Scientific Computing Research  
Office of Science  
US Department of Energy

March 2022

This document was prepared as an account of work sponsored by the United States Government.

While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

## Table of Contents

<b>Motivation</b>	<b>4</b>
<b>Background</b>	<b>5</b>
<b>The Journey of Deploying Software</b>	<b>6</b>
Step 0 - Determine which system to deploy your software	7
Step 1 - Acquire Spack Configuration	8
Step 2 - Preparing Spack Configuration	9
Step 3 - Module Generation	14
Step 4 - Deployment Script	16
Step 5 - User Documentation	17
Step 6- Give back to the community	18
<b>Recent Developments</b>	<b>20</b>
Building E4S on Perlmutter	20
Automation	22
MPI Support	25
Container-based deployment of E4S	26
Testing E4S Post Deployment	28
<b>Conclusion</b>	<b>32</b>
How to Get Involved	33
<b>Acknowledgement</b>	<b>34</b>
<b>Bio</b>	<b>35</b>

## Motivation

One of the many benefits of using a high-performance computing (HPC) system at a Department of Energy (DOE) Office of Science (SC) HPC facility is the large number of software products, built and optimized for the system. The HPC center staff and HPC vendors provide optimized software such as libraries and even full scientific applications, ready to be used by users as building blocks to accelerate scientific discovery. Behind each provided packaged software module are a large number of decisions - which compiler, optimizations, variants/options - to build the software on the target system. And, even before the software gets deployed, the software must be developed, tested, and maintained, including deprecating old versions and ensuring compatibility across versions. The software lifecycle is complex and is further convoluted by a web of interdependencies on other software.

In an HPC environment, the software lifecycle is even more complicated and requires a community to address the many challenges. The Extreme-Scale Scientific Software Stack (E4S) is a community effort supported by the Exascale Computing Project (ECP) to provide an ecosystem of open source software packages for developing, deploying and running scientific applications on HPC platforms. E4S provides unified and consistent deployment through a collection of Spack packages which can be used by users, a development team, or site administrators at an HPC facility. E4S is a flexible software stack for HPC systems that enables an end-user to install a subset of E4S packages for their development purpose, a software development team to install their software product and integrate E4S spack builds into their CI/CD process, or a site-administrator to install E4S on bare-metal or container system-wide for all users.

The utility of E4S extends beyond providing ready-made recipes for building some or all of E4S on a particular machine. E4S releases can be used to identify reasonable default versions of software packages that are known to be interoperable. Recipes for building on a system can be leveraged as a starting point for similar or next-generation systems. For existing supported systems, E4S can be used to quickly learn how to add a new package to a deployment not previously supported on the system. Much of the value of E4S is not in specific recipes for specific systems, but rather in the general improvement of interoperability of packages that comprise the software ecosystem, or documenting where versions of packages are known not to be interoperable, as well as in documenting when certain packages are not yet ported to a platform.

In 2021, National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory released its first deployment of E4S/20.10 on the Cori supercomputer using the spack package manager. NERSC wants to leverage E4S to provide an advanced, performant, stable and supported HPC environment to its users. By being at the forefront of E4S deployment and testing, NERSC is able to provide feedback to ECP Software Technology teams with build failures during deployment so they can be fixed in future versions.

[NERSC has since also deployed E4S on its newest supercomputer, Perlmutter](#), a Cray Shasta supercomputer composed of 1,536 GPU-accelerated nodes with of [AMD EPYC 7713 \(Milan\)](#) CPUs and [NVIDIA A100](#) GPUs, a 35 PB all-flash Lustre scratch file system and the HPE Cray Slingshot 10 high speed interconnect. Perlmutter is currently (2022) being augmented with 3,072 dual-socket CPU only nodes updated to the Slingshot 11 network.

Even with software seemingly packaged and delivered with a bow like E4S, an actual system-wide deployment is complicated and requires many decisions to make for site-specific customizations, e.g. each system supports multiple compilers, compiler versions and MPI providers that impact how software is deployed. Here, we describe the steps and lessons learned to deploy the E4S software stack at NERSC to help users navigate their E4S deployment. The lessons learned can also guide future developers of packaged community software on development-to-deployment requirements.

SINCE THE INITIAL E4S DEPLOYMENT, NERSC CONTINUES TO DEPLOY NEW VERSIONS OF E4S ON CORI AND PERLMUTTER AND PROVIDES THE LATEST INFORMATION ON THE [E4S - NERSC DOCUMENTATION](#) PAGES.

## Background

E4S is a collection of 100+ top-level scientific software packages needed for scientific computing in high-performance computing (HPC) environments. E4S member packages must demonstrate compatibility with the E4S [community policies](#), including a production quality spack-based build and installation procedure. The Department of Energy Office of Science (DOE SC) [ASCR Facilities](#) (NERSC, OLCF and ALCF) are expected to build and deploy E4S on the pre-exascale systems, which helps to ensure a consistent programming environment for users across facilities.

The HPC centers interested in deploying E4S on their facility system(s) should consider how it aligns with their overall software update strategy, which takes into consideration planned system-wide upgrades that may require a rebuild of the full software stack. Leveraging planned disruptive events can minimize the overall system downtime for users. Deploying the entire E4S stack requires installing 500+ software packages, including software dependencies, using a single compiler. Installation scales linearly as one introduces additional compilers to build E4S. However, not all packages need to be installed and HPC centers should take the time to determine which packages are beneficial to their user community. At NERSC, we install a subset of the total E4S software stack system-wide, and of course users can install and configure individual E4S software packages on their own.

In close collaboration with the ASCR facilities, the E4S team created a well-defined release strategy that specifies a spack commit or branch along with a list of packages as part of the E4S release. The E4S team is committed to quarterly releases with new versions of each package. A release will contain spack configuration (spack.yaml) and reference commit, branch or tag of spack project to build E4S that will be available on GitHub at <https://github.com/E4S-Project/e4s>. E4S adopted the [Calendar Versioning](#) scheme (e.g. 22.02), with **YY.MM** format to indicate the year and month for E4S release. For more details on this discussion, please see <https://github.com/E4S-Project/e4s/issues/2>.

E4S also provides architecture-specific container releases and GPU-based images in Docker and Singularity image format. The [E4S Download](#) page describes how individual users may also download and install E4S without any system-level privileges. These deployment options provide flexibility to maximize user productivity. In general, users should consider the performance and portability trade-offs between using containers or building E4S with Spack, targeting the specific architecture.

# The Journey of Deploying Software

E4S is released quarterly, however facilities may choose to install it bi-annually or annually. Although E4S is intended to be easily deployed on systems, in practice, deploying E4S system-wide can be complicated, at least initially, involving a trial-and-error process of determining which components work on the target system. Each successive deployment can leverage the previous recipes and should shorten the time to deployment.

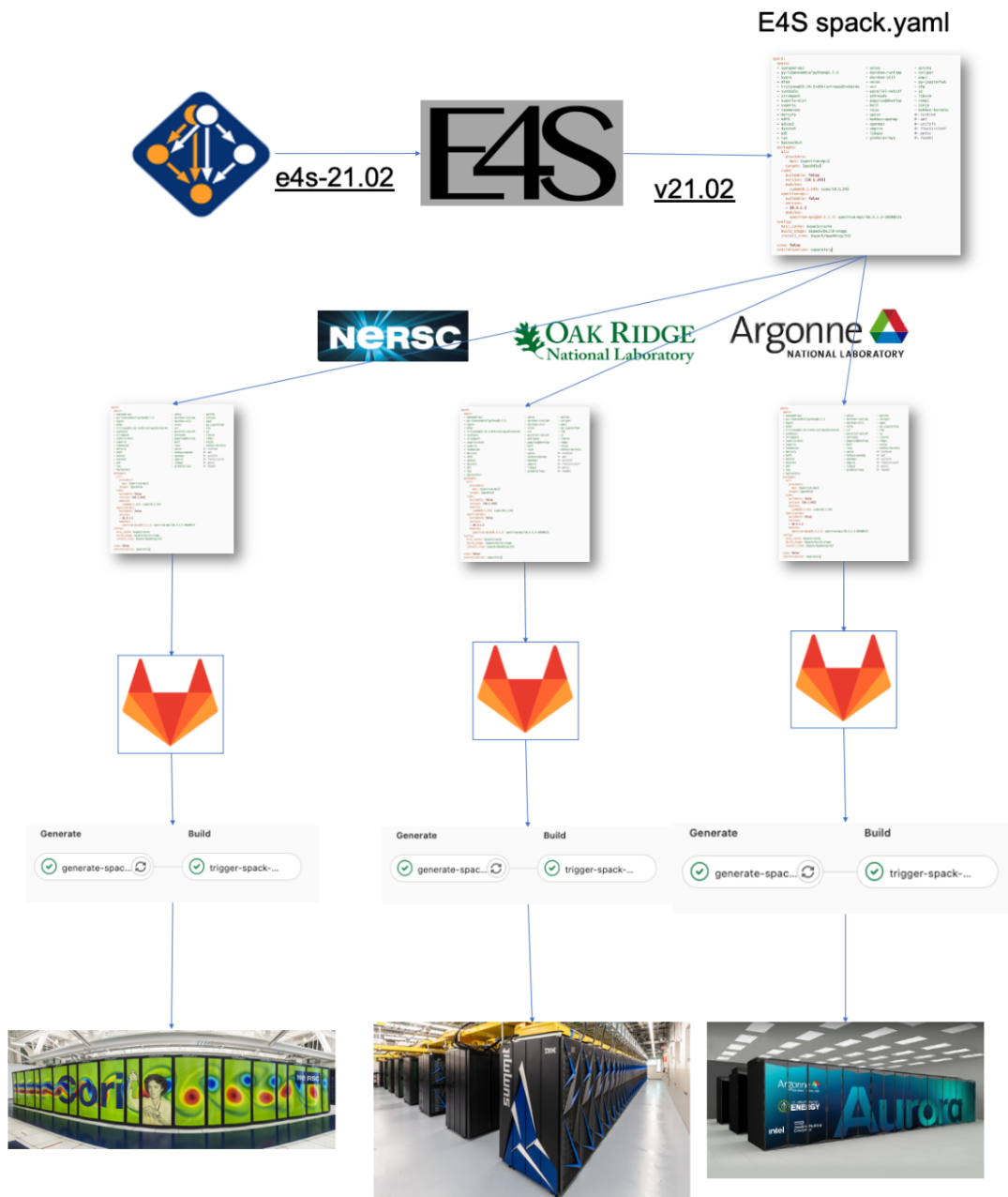


FIGURE 1. E4S software deployment process for the Office of Science HPC facilities (NERSC, OLCF, ALCF)

In Figure 1, we outline the E4S software deployment process at the ASCR facilities. The release begins with E4S providing a reference spack commit/branch along a reference [spack.yaml](#) that ASCR facilities will acquire when building E4S on their system. The reference spack.yaml was built on the University of Oregon HPC system and it's worth noting that simple copy/paste won't work, since site-specific customizations are needed to take into account differences in system architecture, available compilers, and operating system. Each facility will typically need to port the spack configuration for their system.

During the porting process, the facility will determine which packages to install, select their preferred compilers and specify package preference to optimize for their system. In an ideal world, with no build errors, building the E4S stack via **spack install**, which will build all packages from source, requires a few hours of build time depending on the size of the chosen packages. In practice, the build time can take substantially longer to address/debug any software build failures. We leverage Gitlab to automate the entire software deployment process which allows us to analyze pipeline logs once the stack is built and focus on any build failures. Without automation, one would have to run these steps manually on a terminal.

**Lesson Learned:** *Planning ahead based on site-specific resources and your community needs*

Deploying a software stack as large as E4S customized for a site can typically take up to 6-8 weeks from inception depending on the facility deployment process. At NERSC, the process is complete once software is deployed as modules supplemented with user documentation. A significant portion of the software deployment process is troubleshooting build errors that may arise from using different compilers and MPI implementations, and working towards a full stack build with no errors. To build the entire E4S stack, the number of software packages can be up to 500 including third party dependencies. At NERSC, we are working on cutting down this deployment process to 2-3 weeks by automating the deployment process and reducing the size of the E4S software stack we provide to end-users. With each version, we expect to leverage existing software patches to reduce the total time.

## Step 0 - Determine which system to deploy your software

The deployment process begins by determining which system(s) to install E4S and understanding the architecture of the systems. At NERSC, we have two production systems, Cori and Perlmutter that are our target systems for deploying E4S. For the first half of 2021, our focus was deploying E4S on Cori while Perlmutter was getting ready for initial acceptance.

Cori has two primary system partitions, Intel **Haswell** and **KNL**, along with Haswell login nodes. Currently, we build E4S on a Login node that is tuned to target the Haswell architecture which is compatible with the KNL nodes, although it is not optimized for the architecture. Given the significant time to build E4S for each node architecture, we decided to deploy E4S such that it will provide the greatest impact to our users and help us understand the deployment process for building E4S in preparation for Perlmutter.

Cori does not have any GPUs, therefore we were able to skip any E4S packages that require GPUs which typically require CUDA as a software dependency. Shown below is an overview of the Cori system architecture which can be found at <https://docs.nersc.gov/systems/cori/>.

### System Specification

System Partition	Processor	Clock Rate	Physical Cores Per Node	Threads/Core	Sockets Per Node	Memory Per Node
Login	<a href="#">Intel Xeon Processor E5-2698 v3</a>	2.3 GHz	32	2	2	515 GB
Haswell	<a href="#">Intel Xeon Processor E5-2698 v3</a>	2.3 GHz	32	2	2	128 GB
KNL	<a href="#">Intel Xeon Phi Processor 7250</a>	1.4 GHz	68	4	1	96 GB (DDR4), 16 GB (MCDRAM)
Large Memory	<a href="#">AMD EPYC 7302</a>	3.0 GHz	32	2	2	2 TB

**FIGURE 2. NERSC Cori system partitions**

## Step 1 - Acquire Spack Configuration

We acquire the spack configuration from E4S, such as the [e4s/21.02 spack configuration](#) which was released in Feb 2021. This usually means copying the content of spack.yaml and storing this in a git repository in NERSC gitlab server <https://software.nersc.gov>. The E4S team recommends using a per-release tagged (e.g., e4s-22.02) branch of Spack that has been validated with the E4S Spack configuration, and one that maintains release-specific patches as bugs are reported and fixed.

Some HPC facilities may choose to have a fork of spack in order to build E4S, which allows them to update the spack source code to apply their preferred changes without relying on the upstream branch. If one wants to maintain a fork, then one needs to have a deep understanding of the spack source code, or use a tagged branch, when troubleshooting builds. We don't maintain a fork of spack because this further complicates the deployment process to keep the fork in sync with upstream.

**Lesson Learned:** *You can't win them all - not all packages we planned to install are actually installed each release.*

During the spack builds, we discover build errors for certain software packages that did not support our preferred compilers and runtime libraries on the target platform. First we try to troubleshoot the build error by analyzing the build log to resolve the issue and if we require further assistance we contact the spack community via slack or directly reach out to the package maintainer. If we need more visibility into the issue we would report this to the spack issue tracker: <https://github.com/spack/spack/issues>. In the future, we intend to report E4S build issues at the E4S issue tracking system at



<https://github.com/E4S-Project/e4s/issues>, a one-stop site that will curate all E4S issues and coordinate with individual package maintainers.

Someone from the spack community will apply a fix for our issue as a pull request to the spack [develop](#) branch. However, we don't change our version of spack provided by E4S to satisfy build errors, instead we will defer these build errors in the next version of E4S. In the event of build failures, we try our best by experimenting with different build options to address build error, if all else fails we skip the build and document the issue. This process can be improved in the future as well.

## Step 2 - Preparing Spack Configuration

### Compiler Definition

The Cori system supports a wide range of compilers and versions, but for a large software stack deployment, we need to be more selective and determine the compilers we want to use for building E4S. On Cori, we select **Intel** and **GCC** as our preferred compilers. We have several versions of Intel and GCC compilers installed that are accessible via modules. We select one version of the compilers (a stable and widely used version) and define a compiler stanza in the spack configuration relevant for our system.

Shown below is our compiler stanza for the e4s/21.02 deployment. The compiler specs **intel@19.1.2.254** and **gcc@10.1.0** are the compilers used to build E4S. We use Cray compiler wrappers **cc**, **CC**, **ftn** for C, C++ and Fortran wrappers, respectively. The **modules** section informs what modules should be loaded when using the chosen compiler. On Cori we have **PrgEnv-intel** and **PrgEnv-gnu** modules which are Cray provided modules in order to use Intel and GCC compilers that use Cray PE wrappers.

```
compilers:
- compiler:
  spec: intel@19.1.2.254
  paths:
    cc: cc
    cxx: CC
    f77: ftn
    fc: ftn
  flags: {}
  operating_system: cnl7
  target: any
  modules:
  - PrgEnv-intel
  - intel/19.1.2.254
  environment: {unset: []}
  extra_rpaths: []
- compiler:
  spec: gcc@10.1.0
  paths:
    cc: cc
    cxx: CC
    f77: ftn
    fc: ftn
  operating_system: cnl7
  modules:
  - PrgEnv-gnu
  - gcc/10.1.0
```

**FIGURE 3. Spack Compiler Definition for E4S/21.02**

We plan to stick to one compiler version for gcc and intel compiler when building the E4S stack on Cori even though we may have several compilers installed on the system. In future deployments, we will incorporate Cray compilers (**PrgEnv-cray**) into the compiler suite when building E4S to broaden our software support across the three compilers.

**Lessons Learned:** *The compiler selection will impact which packages can be built.*

It's generally a good idea to use the gcc compiler since it is widely supported by the open-source community and most packages will be built successfully. You should target multiple compilers and versions when building a software stack, for instance we noticed that a few packages fail to build with gcc/10.1.0 but when we try a different gcc version they were built successfully.

If your HPC system is running the Cray Programming Environment, it is a good idea to use the Cray provided

compilers which are accessible via modules **PrgEnv-gnu**, **PrgEnv-intel**, **PrgEnv-cray**, and **PrgEnv-nvidia**. If your system supports NVIDIA GPUs, then it would make sense to use the [NVIDIA HPC SDK](#) (nvhpc) compiler.

### Package Selection

Although E4S has 100+ top-level software packages, not all are used by the NERSC user community. To avoid software bloat and reduce the amount of work in deployment, we need to determine which packages get installed along with the preferred compiler. In addition, each software package has a set of build options called **variants** that can be configured in the spack configuration (spack.yaml). The package versions that are provided by E4S typically are the latest versions of the software for the given spack release and we copy these versions provided in E4S release.

The variants are selected by inspecting each package via **spack info** to see applicable build options suitable for our system which requires insight into the specific system software stack. On Cori, generally we enable support for *openmp* and *mpi* when applicable. In some cases we take variants provided by E4S and copy them in our spack configuration. For example, when installing tasmanian we build the package as follows: **tasmanian@7.3 +blas +fortran +mpi +python +xsdkflags**. Each spack package comes with several variants along with default values for each variant. Shown below are the available variants for the tasmanian package.

**Variants:**

Name [Default]	When	Allowed values	Description
=====	====	=====	=====
amdgpu_target [none]	--	none, gfx908, gfx801, gfx1011, gfx802, gfx900, gfx1010, gfx701, gfx1012, gfx906, gfx803	AMD GPU architecture
blas [off]	--	on, off	add BLAS support to Tasmanian
build_type [Release]	[, ]	Debug, Release	CMake build type
cuda [off]	[, ]	on, off	add CUDA support to Tasmanian
cuda_arch [none]	--	none, 13, 61, 37, 32, 80, 86, 75, 21, 12, 72, 10, 70, 30, 11, 60, 50, 52, 53, 35, 20, 62	CUDA architecture
fortran [off]	--	on, off	add Fortran 90/95 interface to Tasmanian
ipo [off]	--	on, off	CMake interprocedural optimization
magma [off]	--	on, off	add UTK MAGMA support to Tasmanian
mpi [off]	--	on, off	add MPI support to Tasmanian
openmp [on]	--	on, off	add OpenMP support to Tasmanian
python [off]	--	on, off	add Python binding for Tasmanian
rocm [off]	[, ]	on, off	add ROCm support to Tasmanian
xsdkflags [off]	--	on, off	enable XSDK defaults for Tasmanian

**FIGURE 4.** List of variants for Tasmanian spack package

We define a definition name **e4s\_intel** and **e4s\_gcc** to map spack packages that will be installed with gcc and intel compiler. We skip some packages for various reasons, for instance we don't want *openmpi* installed in the E4S stack. Packages like *parallel-netcdf* and python extensions that start with **py-\*** are generally skipped. Some packages were skipped due to build failures.

```

definitions:
- intel_compiler: ['%intel@19.1.2.254']
- gcc_compiler: ['%gcc@10.1.0']
- e4s_intel:
  - adios2@2.7.1 +hdf5
  - aml@0.1.0
  - arborx@0.9-beta +openmp
  - bolt@2.0
  - caliper@2.5.0 +fortran
  - faodel@1.1906.1
  - flecsi@1.4 +cinch +caliper +graphviz +tutorial
  - flit@2.1.0
  - gasnet@2020.3.0 +udp
  - ginkgo@1.3.0
  - gotcha@1.0.3 +test
  - hdf5@1.10.7
  - hypre@2.20.0 +mixedint +superlu-dist +openmp
  - libnrm@0.1.0
  - libquo@1.3.1
  - mercury@2.0.0 +udreg
  - mfem@4.2.0 +examples +gnutls +gslib +lapack +libunwind +openmp +threadsafe +pumi +umpire
  - ninja@1.10.2
  - omega-h@9.32.5 ~trilinos
  - openpmc-api@0.13.2
  - papi@6.0.0.1 +example +static_tools +powercap +infiniband
  - papyrus@1.0.1
  - pdt@3.25.1 +pic
  - precice@2.2.0 +python
  - pumi@2.2.5 +fortran
  - qthreads@1.16 ~hwloc
  - raja@0.13.0 +tests
  - slepc@3.14.2
  - strumpack@5.1.1 +shared
  - sundials@5.7.0 +examples-cxx +hypre +klu +lapack
  - superlu@5.2.1
  - superlu-dist@6.4.0 +openmp
  - swig@4.0.2-fortran
  - tasmanian@7.3 +blas +fortran +mpi +python +xSDKflags
  - tau@2.30.1 +mpi ~pdt
  - turbine@1.2.3 +hdf5 +python
  - umap@2.1.0 +tests
  - umpire@4.1.2 +fortran +numa +openmp
  - upcxx@2020.10.0
  - zfp@0.5.5 +aligned +c +fortran +openmp +profile

```

```

- e4s_gcc:
  - darshan-runtime@3.2.1 +slurm
  - darshan-util@3.2.1 +bzip2
  - dyninst@10.2.1
  - legion@20.03.0
  - plasma@20.9.20
  - slate@2020.10.00 ~cuda

# skipping package
# - adios@1.13.1 +bzip2 +fortran +hdf5 +netcdf
# - kokkos-kernels@3.2.00 +mkl +openmp
# - kokkos@3.2.00 +compiler_warnings +deprecated_code +examples +hwloc +memkind +numactl +openmp +pic +tests
# - openmpi@4.0.5 +cxx +thread_multiple schedulers=slurm
# - parallel-netcdf@1.12.1 +burstbuffer
# - petsc@3.14.4 +X +fftw +jpeg +libpng +libyaml +memkind
# - py-jupyterhub@1.0.0
# - py-libensemble@0.7.1 +mpi +nlopt +petsc4py +scipy
# - py-petsc4py@3.14.1
# - trilinos@13.0.1

```

**FIGURE 5.** List of spec definitions to specify which spack packages to install

**Lesson Learned:** *Building software is an art, and beauty is in the eye of the installer*

It's worth noting that variant selection is an art which sometimes comes down to the installers preference and our selection may not be consistent with the developers or users preference. There is no universal selection for each package because this selection process depends on the system architecture, the compiler, and system software stack. The variant selection is an important aspect when building packages as it impacts how a package gets installed and our selection may not be optimal for all user needs. In certain situations, we reach out to developers for recommendations on the package variants. There are situations where certain variants are mutually exclusive, for instance some package X that can either support openmp or pthreads so doing **package X +openmp +pthreads** is not allowed.

### Package Preference

Most scientific software requires MPI, [BLAS](#), and [ScaLAPACK](#) as common dependencies when installing software, but on Cori we choose not to build these from source since they are not optimized for the system and are generally provided as vendor software (cray-libsci, intel-mkl, cray-mpich). This typically requires one to specify preferences to ensure spack doesn't use the default preference to build from source, and instead uses an alternative. In our spack configuration, we leverage **cray-libsci**, **intel-mkl** and **mpich** as preferences for mkl, mpi, blas and scalapack.

```
specs:
- matrix:
  - [$e4s_intel]
  - [$intel_compiler]
- matrix:
  - [$e4s_gcc]
  - [$gcc_compiler]
```

**FIGURE 6. Install specs based on definition list**

The spack documentation has a detailed summary on build customization which can be found at [https://spack.readthedocs.io/en/latest/build\\_settings.html](https://spack.readthedocs.io/en/latest/build_settings.html)

```
packages:
  all:
    compiler: [intel@19.1.2.254, gcc@10.1.0]
    target: [haswell]
    providers:
      mpi: [mpich]
      mkl: [cray-libsci, intel-mkl]
      blas: [cray-libsci, intel-mkl]
      scalapack: [cray-libsci, intel-mkl]
      pkgconfig: [pkg-config]
```

**FIGURE 7. Package Preference for compiler and spack providers**

```
cray-libsci:
  buildable: false
  externals:
  - spec: cray-libsci@19.06.1%intel
  modules:
  - cray-libsci/19.06.1
```

**FIGURE 8. Spack external definition for cray-libsci**

Certain packages like cray-libsci, intel-mkl, mpich are provided on our system which typically requires setting a [spack external](#) to ensure spack will leverage our preferred libraries provided by Cray. In the example below we define **cray-libsci** as an external module which maps to modulefile **cray-libsci/19.06.1** which is available on Cori.

**Lessons Learned:** *Grab a cup of coffee and take time to read the dependency tree*

During the package preference determination, we analyze output of **spack concretize** or run **spack spec** to see the dependency tree to determine if output seems reasonable. In this process, we analyze each package variant, the dependency tree such as what MPI wrapper, blas provider is used. This process is time consuming especially when one is trying to analyze output of the entire software stack, which can be hundreds of packages.

Figure 9 shows a concretized spec of hdf5, and its dependencies, we can see *hdf5 +mpi* is set in the concretized output which means build HDF5 with MPI support and this leads to the *cray-mpich* dependency which is an external package.

```
==> Concretized hdf5gcc9.3.0
- waf2r3 hdf5@1.12.0gcc9.3.0-cxx+fortran+h1-ipo-java+mpi+shared-axip-threads+safe+tools api=v18 build_type=RelWithDebInfo arch=cray-sles15-zen2
- tty4hbt *cmake@3.22.2gcc9.3.0-doc+ncurses+openssl+ownlibs-gt build_type=Release arch=cray-sles15-zen2
- yyij5y *ncurses@6.1gcc9.3.0-symbols+terminlib abi=none arch=cray-sles15-zen2
- mwq7u *cray-mpich@8.1.12gcc9.3.0 arch=cray-sles15-zen2
- nac4se *numactl@2.0.14gcc9.3.0
patches=4e1d78cbb85de25bad28705e748856033eaaafab92a66df383a3d7e00cc94,62fc8a8bf7665a60e8f4c93ebbd535647ceb74198f7afafec4e085a8825c006,ff37630df599cfabf0740518b91ec8daaf18e8f288b19adaae5364dc1f6b2296 arch=cray-sles15-zen2
- ly1b0zm *autoconf@2.69gcc9.3.0
patches=3c4492815463764976e92649f6c121ca0e033ee6efefc9be2af3253082c2,7793209b33013dc0f81208718c68440c5aae80e7alc4b8d33e382525af791a7,a49dd5bac3b62da0ff6688ab4d508d71dbd2f4f87e2a02321926346161bf3ee arch=cray-sles15-zen2
- v2y6pw *m81.4.18gcc9.3.0+sigsegv patches=3877ab548f88597ab2327a2230ee048d2d07ace1062efe81fc92e91b7f39cd00,fc9b61654a3ba1a8d6cd78ce087e7c9636c290bc8d2c299f9882d793b853c8 arch=cray-sles15-zen2
- 5ygh3lf *perl@5.34.0gcc9.3.0-cpan+shared+threads arch=cray-sles15-zen2
- cypj04r *berkeley-db@18.1.40gcc9.3.0-cxx+doc+sql patches=0231fcc4d5cfff05e5a348146a5af0e9a96428dc2176540d2c05af41de522 arch=cray-sles15-zen2
- 4hnguj *bzip2@1.0.6gcc9.3.0-debug-pie+shared arch=cray-sles15-zen2
- v12i23g *gdbm@1.19gcc9.3.0 arch=cray-sles15-zen2
- wog4ob2 *readline@7.0gcc9.3.0 arch=cray-sles15-zen2
- fuw9lba *zlib@1.2.11gcc9.3.0+optimize+pie+shared arch=cray-sles15-zen2
- qj22d5d *automake@1.16.5gcc9.3.0 arch=cray-sles15-zen2
- y9me47v *libtool@2.4.6gcc9.3.0 arch=cray-sles15-zen2
- cmi5wrf *pkgconf@1.0.0gcc9.3.0 arch=cray-sles15-zen2
```

FIGURE 9. Concretized output of hdf5

### Spack External Recommendation

Spack provides ability to reuse software pre-installed on system via [spack externals](#) to avoid reinstalling software that will never be used. You should consult output of `spack concretize -f` if you are in a spack environment or `spack spec <spec>` for one of packages to see list of dependencies. We have compiled a list of spack packages that should be external when building spack stacks on NERSC systems. You may run `spack external find <spec>` to update your spack.yaml however we recommend you always confirm your spack configuration with whats provided by system.

Spack Package	Description
bash	This is GNU Bourne Again Shell known as <code>bash</code> shell which is typically found in most linux systems. You can check supported shells by <code>cat /etc/shells</code> , you can check version of bash by running <code>/usr/bin/bash --version</code> . This package should not be installed as spack package
bzip2	<code>/usr/bin/bzip2</code> should be available on system. You can run <code>bzip2 --version</code> to check version. You can check if rpm is available by running <code>rpm -q --whatprovides \$(which bzip2)</code>
cpio	The GNU cpio is program to manage archives of files, this program is available at <code>/usr/bin/cpio</code> . This package should not be installed via spack
cray-libsci	cray-libsci is provided by Cray Programming Environment. You should see available module by running <code>module avail cray-libsci</code> . The cray-libsci package is typically installed in <code>/opt/cray/pe/libsci/</code> . For instance a modulefile <code>cray-libsci/21.08.1.2</code> should be external for package <code>cray-libsci@21.08.1.2</code> .
cray-mpich	cray-mpich is MPI wrapper based on mpich provided by Cray. You can find <code>cray-mpich</code> modulefile, this is typically installed at <code>/opt/cray/pe/mpich/</code> on cray machines. You should set this as external for packages that depend on MPI.
cuda	cuda is typically provided on NERSC but not as a system library ( <code>/usr/bin/nvcc</code> ) but usually through modules. You can find this typically in modules such as <code>module av cuda nvhpc nvidia cudatoolkit CUDA</code> . You should run <code>nvcc --version</code> to check version. The nvhpc compiler provides typically provides three versions of cuda for instance cuda packages by <code>nvhpc@21.9</code> are located <code>/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda</code> on Perlmutter. If you want to set a version@10.2 external then the path would be <code>/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/10.2/</code> . Sometimes you may not want cuda as external especially when one needs to install from source or you have incompatible between compiler and cuda version. For reference please see <a href="#">nvhpc release notes</a> . There is a more detailed breakdown of cuda compatibility requirement found at <a href="https://gist.github.com/ax3l/9489132">https://gist.github.com/ax3l/9489132</a> . Please check the cuda driver via <code>nvidia-smi</code> for driver compatibility.
coreutils	This package is install on Linux OS which provides common utilities like <code>cat</code> , <code>ls</code> , <code>echo</code> and many other utilities. You can check if coreutils is installed and its version by running <code>rpm -qi coreutils</code>
diffutils	This is the GNU diffutils package which provides <code>/usr/bin/diff</code> . This package should be external and will most likely get picked up when building a large software stack
findutils	The findutils package provides <code>find</code> and <code>xargs</code> command which is provided by OS. This is typically located in <code>/usr/bin/find</code> and <code>/usr/bin/xargs</code> . Please check the version of the utility or check the rpm version by running <code>rpm -qi findutils</code> .
git	git is typically provided on NERSC systems that can be found at <code>/usr/bin/git</code> . This should be an external, we don't need spack to install multiple versions of git that user will never need.
libfabric	The libfabric package is provided by Cray which can be searched by running <code>module av libfabric</code> . This is typically installed at <code>/opt/cray/libfabric/</code> .

FIGURE 10. Spack External Documentation with breakdown by spack package and description

In this process, we learned spack tries to do some interesting things like installing openssl, openssl, basic linux utilities or even a scheduler like Slurm based on concretization preferences, which are redundant or

unoptimized for Cori. These types of selections need intimate knowledge of the system stack along with analyzing output of *spack concretize* to see what gets installed. After several E4S software stack builds, we [documented](#) a list of externals shown in Figure 10 that should be set which is applicable for a NERSC system. Some of these externals may be applicable for your system.

## Step 3 - Module Generation

Most HPC systems nowadays leverage **modules** to allow the user to easily interface with the software stack and provide a consistent programming environment. A **modulefile** is a file that configures a software package such as PATH and LD\_LIBRARY\_PATH to configure the user environment in order to use the software with ease. Currently, there are two module systems in use, [Lmod](#) and [environment-modules](#) which provide a module implementation that is widely used in the HPC community. In environment-modules modulefiles are written [Tool Command Language \(TCL\)](#), while Lmod supports both TCL and Lua modules with preference for [Lua](#) modules.

Spack provides a mechanism to generate modules in TCL and Lua format based on the module system. On Cori we use environment-modules which support TCL based modules. During the module generation process, we inform spack on the format of the modulefile. We avoid hash in modules and prefer having modules in the format `{name}/{version}-{compiler.name}-{compiler.version}` which avoids module conflicts when a package like `hdf5@1.10.7` is installed with both compilers. Shown below is the spack configuration for modules along with output of the spack generated modules. It's worth noting we limit

```

modules:
  enable:
  - tcl
  tcl:
    blacklist_implicit: true
    hash_length: 0
    naming_scheme: '{name}/{version}-{compiler.name}-{compiler.version}'
  all:
    conflict:
    - '{name}'
    environment:
    set:
      '{name}_ROOT': '{prefix}'
  darshan-runtime:
    conflict:
    - 'darshan'
  darshan-util:
    conflict:
    - 'darshan'
  projections:
    all: '{name}/{version}-{compiler.name}-{compiler.version}'
----- /global/common/software/spackcp/e4s-21.02/modules/cray-cn17-haswell/ -----
adios2/2.7.1-intel-19.1.2.254      legion/20.03.0-gcc-10.1.0      raja/0.13.0-intel-19.1.2.254
aml/0.1.0-intel-19.1.2.254        libnm/0.1.0-intel-19.1.2.254  slate/2020.10.00-gcc-10.1.0
erbox/0.9-beta-intel-19.1.2.254  libquo/1.3.1-intel-19.1.2.254  slepc/3.14.2-intel-19.1.2.254
bolt/2.0-intel-19.1.2.254        mercury/2.0.0-intel-19.1.2.254 strumpack/5.1.1-intel-19.1.2.254
caliper/2.5.0-intel-19.1.2.254   mfm/4.2.0-intel-19.1.2.254    sundials/5.7.0-intel-19.1.2.254
darshan-runtime/3.2.1-gcc-10.1.0  ninja/1.10.2-intel-19.1.2.254  superlu-dist/6.4.0-intel-19.1.2.254
darshan-util/3.2.1-gcc-10.1.0    omega-h/9.32.5-intel-19.1.2.254 superlu-dist/6.4.0-intel-19.1.2.254
dyninst/10.2.1-gcc-10.1.0        openpm�-api/0.13.2-intel-19.1.2.254 swig/4.0.2-fortran-intel-19.1.2.254
faodel/1.1906.1-intel-19.1.2.254  papi/6.0.0.1-intel-19.1.2.254  tasmanian/7.3-intel-19.1.2.254
flecsi/1.4-intel-19.1.2.254      papyrus/1.0.1-intel-19.1.2.254 tau/2.30.1-intel-19.1.2.254
flit/2.1.0-intel-19.1.2.254      pdt/3.25.1-intel-19.1.2.254   turbine/1.2.3-intel-19.1.2.254
gasnet/2020.3.0-intel-19.1.2.254  plasma/20.9.20-gcc-10.1.0     umpire/4.1.2-intel-19.1.2.254
ginkgo/1.3.0-intel-19.1.2.254     precice/2.2.0-intel-19.1.2.254 upcxx/2020.10.0-intel-19.1.2.254
gotcha/1.0.3-intel-19.1.2.254    pumi/2.2.5-intel-19.1.2.254   zfp/0.5.5-intel-19.1.2.254
hypr/2.20.0-intel-19.1.2.254     qthreads/1.16-intel-19.1.2.254

```

our module generation to root specs and avoid generating modules for dependencies by setting **blacklist\_implicit: true** which avoids explosion of modules and higher likelihood of module conflicts. The **hash\_length** configures the number of hash characters to append to each modulefile. For every modulefile we set a conflict on the same name which adds a keyword **conflict** to each modulefile such that one can't load two instances at same time.

FIGURE 11. Spack configuration for module generation and output of generated modules

We provide an overarching modulefile to load the specific e4s stack version that corresponds to each release of E4S. For instance on Cori we have three versions of e4s as shown below. The modulefile will set up a spack instance used for deployment and update MODULEPATH with spack generated modules.

```
siddiq90@cori12> module av e4s
|
|----- /global/common/software/neresc/cle7/extra_modulefiles -
e4s/20.10 e4s/21.02 e4s/21.05
```

FIGURE 12. E4S modulefile used for accessing E4S stack

```
config:
install_tree:
  root: /global/common/software/spackcep/e4s-21.02/software
module_roots:
  tcl: /global/common/software/spackcep/e4s-21.02/modules/
```

FIGURE 13. Spack configuration for install root and module root

During production deployment we select the location where spack will install the software and modules which are defined using **install\_tree** and **module\_root**. We install software on a shared file system that is accessible on both login and compute nodes, in this case the root directory

`/global/common/software/spackcep` is available for us to perform E4S deployment. We organize each e4s deployments by version to support multiple releases.

**Lessons Learned:** *What's in a name? Just keep it short and sweet.*

We like to keep module names as short as possible, this means we don't include hash names during module generation which is the default behavior when spack generates modules. In our first deployment of E4S, we included hash names as shown below. This can be inconvenient for users as they have a very long modulename that they need to load such as **module load adios2/2.6.0-intel-19.1.2.254-n4dtk4qs** to load the adios2 package.

```
----- /global/common/software/spackcep/e4s-20.10/modules/cray-cn17-haswell/ -----
adiak/0.1.1-intel-19.1.2.254-mthjgtd2      kvtree/1.0.2-intel-19.1.2.254-ob5jw5ci      pumi/2.2.2-intel-19.1.2.254-u7ra5bw2
adios2/2.6.0-intel-19.1.2.254-n4dtk4qs    libbsd/0.10.0-intel-19.1.2.254-w4tvpvc2    py-cython/0.29.21-intel-19.1.2.254-yjdamuq7
adlxb/0.9.2-intel-19.1.2.254-xlmanu5      libfabric/1.11.0-intel-19.1.2.254-rilhysw  py-libensemble/0.7.0-intel-19.1.2.254-bn6b6rft
```

Since the e4s/21.02 deployment and all future releases we don't have any hash in modulefile naming. In the module generation step, we run **spack module tcl refresh** to build the TCL modules. During this process we can run into module conflicts which require unique module names. For example, **warpx/21.05** package has three modulefiles for the same version which require unique module names.

```
siddiq90@cori07> module av warpx
```

```
----- /global/common/software/spackcep/e4s-21.05/modules/cray-cn17-haswell/ -----
warpx/21.05-dims2  warpx/21.05-dims3  warpx/21.05-dimsRZ
```

This is because we have three instances of the warpx package that was installed with variants **dims=2**, **dims=3** and **dims=rz** so we have three unique module names to support all of the instances. Shown below are the warpx packages installed from our e4s/21.05 deployment.

```
siddiq90@cori07> spack find --format "{name}@{version} {compiler.name}@{compiler.version}: {variants}" warpx
warpx@21.05 %intel@19.1.3.304: +app-ascent-eb-ipo+lib+mpi+mpithreadmultiple+openmpi+psatd+qed-qedtablegen+shared+tprof build_type=RelWithDebInfo compute=omp dims=2 precision=double
warpx@21.05 %intel@19.1.3.304: +app-ascent-eb-ipo+lib+mpi+mpithreadmultiple+openmpi+psatd+qed-qedtablegen+shared+tprof build_type=RelWithDebInfo compute=omp dims=3 precision=double
warpx@21.05 %intel@19.1.3.304: +app-ascent-eb-ipo+lib+mpi+mpithreadmultiple+openmpi+psatd+qed-qedtablegen+shared+tprof build_type=RelWithDebInfo compute=omp dims=rz precision=double
```

If you are deploying multiple software stacks like E4S, it's a good idea to keep your software stack behind a meta-module like **e4s/21.02** as we did for our E4S deployment. With this approach, we are able to support multiple E4S deployments at same time which has the following benefits

- Minimize output of **module avail** at startup modules, one has to load the e4s module
- Site administrators can easily deprecate stack by removing modulefile and also adding notice in modulefile
- Avoid Name/Version conflicts in modulefile across different directories in MODULEPATH (i.e two modulefiles called **gcc/9.3.0**)
- Users will be forced to run **module load e4s/<version>** to access stack followed by loading some package (i.e **module load petsc**) compared to just **module load petsc** can cause scripts to break if site-administrators update the version or remove modulefile.

## Step 4 - Deployment Script

The deployment process for e4s/21.02 was initiated through GitLab CI by defining a gitlab job in [.gitlab-ci.yml](#). Shown in Figure 14 is the **deploy** job which does the production deployment. This process will clone spack into the production path and install specs from buildcache which we did in advance and generate the modulefiles. The deploy job is initiated once the entire stack can be rebuilt from source and pushed to buildcache. The gitlab configuration for e4s/21.02 can be found at <https://github.com/spack/spack-configs/blob/main/NERSC/cori/e4s-21.02/.gitlab-ci.yml>

```
deploy:
  stage: deploy
  tags: [cori]
  only:
    variables:
      - $DEPLOY_E4S == "True"
  script:
    - mkdir -p /global/common/software/spackecp/e4s-21.02
    - cd /global/common/software/spackecp/e4s-21.02
    - rm -rf spack
    - git clone $SPACK_REPOSITORY -b e4s-21.02
    - cd spack
    # Need this PR: https://github.com/spack/spack/pull/22500
    - git cherry-pick f67d477
    - source share/spack/setup-env.sh
    # copy all site configuration for spack instance
    - cp $CI_PROJECT_DIR/site_config/* $SPACK_ROOT/etc/spack
    # the other permission is set to 0 before copying. Adding 4 will ensure user can read the file
    - chmod 664 $SPACK_ROOT/etc/spack/*.yaml
    - export SPACK_GNUPGHOME=$HOME/.gnupg
    - spack buildcache update-index -d /global/common/software/spackecp/mirrors/cori-e4s-21.02
    - find /global/common/software/spackecp/mirrors/cori-e4s-21.02 -name *.spack -exec chmod o+r {} \;
    - chmod o+r /global/common/software/spackecp/mirrors/cori-e4s-21.02/build_cache/index.json
    - spack buildcache list -L
    - spack env create e4s $CI_PROJECT_DIR/prod/spack.yaml
    - spack env activate e4s
    - spack install --cache-only
    - spack module tcl refresh --delete-tree -y
    - spack find
```

FIGURE 14. Gitlab Deployment Job for e4s/21.02



## Step 5 - User Documentation

The last step for deployment is writing user documentation for our E4S stack on NERSC systems. Our home page for E4S is <https://docs.nersc.gov/applications/e4s/> where we have a separate documentation page per E4S release. The user documentation goes through peer review and further testing to ensure documentation is accurate. Shown below is a preview of our E4S documentation at NERSC, we have a subpage with documentation for each E4S stack.

**NERSC Documentation**

Home

Getting Started

Tutorials >

Accounts >

Iris >

Systems >

Storage Systems >

Connecting >

Environment >

Policies >

Development >

Developer Tools >

Running Jobs >

Applications >

AMBER

Abinit

BerkeleyGW

CP2K

CPMD

**E4S** >

22.02

21.11

21.05

E4S Spack Develop

Spack-Gitlab-Pipeline

GAMESSS

Gromacs

LAMMPS

Mathematica

MATLAB >

MOLPRO

NAMD

NCAR Graphics

NWChem

SIESTA

ParaView

### Extreme-scale Scientific Software Stack (E4S)

The [Extreme-scale Scientific Software Stack \(E4S\)](#) is a curated software stack from the [Spack](#) ecosystem that is continuously built and tested on several pre-exascale systems. E4S is composed of many open-source projects, including xSDK, dev-tools, math-libraries, compilers, and more. For a complete product list see [E4S Product Dictionary](#).

E4S is shipped as a container (Docker, Singularity, Shifter, CharlieCloud), a [buildcache](#), or a Spack manifest ( `spack.yaml` ). Currently, we focus on building E4S from source using a `spack.yaml` file provided by the E4S team from <https://github.com/E4S-Project/e4s>.

**Note**

We install as many packages from E4S provided in `spack.yaml` as possible. Some packages were intentionally skipped such as specs tied to `develOp` branches or packages built for GPUs. Some additional packages couldn't be installed or had concretization issues with our base compiler.

### E4S Support Timeline

This table outlines the support lifetime for each E4S version. The Release Date is marked on the day of E4S release when user documentation was live. Once E4S version has reached end of support we will remove E4S and corresponding modulefiles and user documentation. As we approach the **End of Support** for a particular release, we will communicate via email and modulefile will include a banner when loading the module.

**We recommend users to port their application and any scripts to the newer release.**

System	Version	Release Date	End of Support
Cori	20.10	Jan 15 <sup>th</sup> 2021	Mar 16 <sup>th</sup> 2022
Cori	21.02	Jun 11 <sup>th</sup> 2021	Mar 16 <sup>th</sup> 2022
Cori	21.05	Aug 23 <sup>th</sup> 2021	Oct 31 <sup>st</sup> 2022
Perlmutter	21.11	Jan 22 <sup>nd</sup> 2022	Mar 31 <sup>st</sup> 2023
Cori	22.02	Mar 16 <sup>th</sup> 2022	End of Life Cori (2023)

**Table of contents**

- E4S Support Timeline
- Point of Contact
- References
- Past Events

**FIGURE 15. NERSC E4S Documentation**

Recently, we outlined a support timeline for each E4S stack in order to deprecate older stacks in preference of newer versions. This process will entail removal of modulefile, user documentation and uninstall the software stack from the filesystem. Shown below is a preview of our [E4S Support Timeline](#).

## E4S Support Timeline

This table outlines the support lifetime for each E4S version. The Release Date is marked on the day of E4S release when user documentation was live. Once E4S version has reached end of support we will remove E4S and corresponding modulefiles and user documentation. As we approach the **End of Support** for a particular release, we will communicate via email and modulefile will include a banner when loading the module.

**We recommend users to port their application and any scripts to the newer release.**

System	Version	Release Date	End of Support
Cori	20.10	Jan 15 <sup>th</sup> 2021	Mar 16 <sup>th</sup> 2022
Cori	21.02	Jun 11 <sup>th</sup> 2021	Mar 16 <sup>th</sup> 2022
Cori	21.05	Aug 23 <sup>th</sup> 2021	Oct 31 <sup>st</sup> 2022
Perlmutter	21.11	Jan 22 <sup>nd</sup> 2022	Mar 31 <sup>st</sup> 2023
Cori	22.02	Mar 16 <sup>th</sup> 2022	End of Life Cori (2023)

FIGURE 16. E4S support timeline for each release on NERSC systems

## Step 6- Give back to the community

We contribute back our spack configuration to <https://github.com/spack/spack-configs> in addition we update the [E4S Facility Dashboard](#) as shown below. We want other HPC centers to contribute to this page ([https://e4s.readthedocs.io/en/latest/facility\\_e4s.html](https://e4s.readthedocs.io/en/latest/facility_e4s.html)) as they deploy E4S on their system so we can see where E4S has been deployed. The HPC community can benefit by seeing how other centers have deployed E4S by sharing their spack configuration.

## E4S Facility Dashboard

System	Institution	E4S Version	Total Installed Specs	Compiler
Perlmutter	NERSC	21.11	94	gcc@9.3.0 , nvhpc@21.9
Cori	NERSC	20.10	135	intel@19.1.2.254
Cori	NERSC	21.02	149	intel@19.1.2.254 , gcc@10.1.0
Cori	NERSC	21.05	157	intel@19.1.3.304
Cori	NERSC	22.02	385	gcc@11.2.0 , intel@19.1.2.254
Spock	ORNL	21.05	200	gcc@10.2.0 , gcc@10.3.0 , cce@1:
Spock	ORNL	21.08	1083	gcc@10.2.0 , gcc@10.3.0 , cce@1:
Summit	ORNL	21.05	632	gcc@8.3.1 , gcc@9.1.0 , gcc@9.3 0 , clang@12.0.0.-0
Summit	ORNL	21.08	2025	gcc@8.3.1 , gcc@9.1.0 , gcc@9.3 0 , clang@12.0.0.-0
Crusher	ORNL	21.08	1426	gcc@7.5.0 , gcc@9.3.0 , gcc@10.: rocm4.5.0
Arcticus	ANL	21.05	334	gcc@9.3.0
Arcticus	ANL	21.08	392	gcc@9.3.0 , oneapi@2021.4.0
Arcticus	ANL	21.11	426	gcc@9.3.0 , oneapi@2021.4.0

**FIGURE 17. Summary of E4S deployments at DOE facilities available in E4S Documentation**

We will communicate our E4S deployment release with our NERSC and ECP user-base through NERSC weekly emails and slack channel. This way we can coordinate efforts across the various ASCR facilities and share best practices.

## Recent Developments

E4S development and deployment at the ASCR facilities moves quickly. Here are details on some of the latest developments.

### Building E4S on Perlmutter

We will discuss some of our experiences building E4S on Perlmutter. In Oct 2021 we started the process of building the most recent version of e4s at that time. We picked e4s/21.11 as the preferred version which was released in Nov 2021. During this period we encountered several changes to the Cray Programming Environment (CPE) over the span of 3 months with CPE 21.08, 21.10, 21.11 and 21.12. These changes impact our compiler and package preference in our spack configuration. We went through 3 rebuilds of Perlmutter E4S/21.11 over the span of three months with one rebuild being performed in April 2022.

Our compiler choice for Perlmutter is gcc and nvhpc compiler and cray-mpich as our preferred MPI provider. One of the pain points was having to update the package external for cray packages such as cray-mpich. For instance, we noticed that CPE 21.10 had cray-mpich version 8.1.10 but CPE 21.12 provided version 8.1.12. Furthermore, we also had **cuda** modules which were changed to **cuda toolkit** which were provided by NERSC staff since we didn't have a standalone cuda module.

```

cray-mpich:
  buildable: false
  externals:
  - spec: cray-mpich@8.1.10 %nvhpc@21.7
    prefix: /opt/cray/pe/mpich/8.1.10/ofi/gnu/9.1
    modules:
    - cray-mpich/8.1.10
    - cuda/11.3.0
  - spec: cray-mpich@8.1.10 %gcc@9.3.0
    prefix: /opt/cray/pe/mpich/8.1.10/ofi/gnu/9.1
    modules:
    - cray-mpich/8.1.10
    - cuda/11.3.0

cray-mpich:
  buildable: false
  externals:
  - spec: cray-mpich@8.1.12 %gcc@9.3.0
    prefix: /opt/cray/pe/mpich/8.1.12/ofi/gnu/9.1
    modules:
    - cray-mpich/8.1.12
    - cudatoolkit/21.9_11.4
  - spec: cray-mpich@8.1.12 %nvhpc@21.9
    prefix: /opt/cray/pe/mpich/8.1.12/ofi/nvidia/20.7
    modules:
    - cray-mpich/8.1.12
    - cudatoolkit/21.9_11.4

cuda:
  buildable: false
  version: [11.3.0]
  externals:
  - spec: cuda@11.3.0
    prefix: /global/common/software/ner/scos1.3/cuda/11.3.0
    modules:
    - cuda/11.3.0

cuda:
  buildable: false
  version: [11.4.0]
  externals:
  - spec: cuda@11.4.0
    prefix: /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/11.4
    modules:
    - cudatoolkit/21.9_11.4

```

**FIGURE 18. Package preference for cray-mpich and cuda on Perlmutter for CPE 21.10 and 21.12**

Cray provides an NVHPC compiler provided by NVIDIA which typically comes with 3 versions of cuda in the same distribution. We ended up writing modulefiles for each cuda version mapping to the NVHPC compiler. For instance **cudatoolkit/21.3\_10.2** refers to cuda version 10.2 from the NVHPC 21.3 compiler.

```
siddiq90@login31> ml -t av cudatoolkit
/opt/cray/pe/lmod/modulefiles/core:
cudatoolkit/21.3_10.2
cudatoolkit/21.3_11.0
cudatoolkit/21.3_11.2
cudatoolkit/21.9_10.2
cudatoolkit/21.9_11.0
cudatoolkit/21.9_11.4
/opt/modulefiles:
cudatoolkit/21.3_10.2
cudatoolkit/21.3_11.0
cudatoolkit/21.3_11.2
cudatoolkit/21.9_10.2
cudatoolkit/21.9_11.0
cudatoolkit/21.9_11.4
```

Most recently, we removed older versions of **cudatoolkit** modulefile and changed the modulefile name format to exclude nvhpc version. These changes impacted our spack build and any system changes like changes to modulefile need to be synced with our spack configuration. In addition we recently added NVHPC 21.11 with intent of removing 21.9 in near future, so we are now planning to rebuild with the latest compiler. Our initial deployment used **gcc@9.3.0** compiler and **nvhpc@21.9** however, **gcc/9.3.0** modulefile was also removed, so now we are planning to use **gcc/11.2.0**. Shown below is our compiler definition on Perlmutter, with the left image showing our first iteration and image on right showing our updated compilers which we plan to use for our upcoming redeployment.

```
- compiler:
  spec: gcc@9.3.0
  paths:
    cc: /opt/cray/pe/craype/default/bin/cc
    cxx: /opt/cray/pe/craype/default/bin/CC
    f77: /opt/cray/pe/craype/default/bin/ftn
    fc: /opt/cray/pe/craype/default/bin/ftn
  flags: {}
  operating_system: sles15
  target: any
  modules:
    - PrgEnv-gnu
    - gcc/9.3.0
    - craype-x86-milan
    - libfabric
  #environment:
  # append_path:
  # LIBRARY_PATH: /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/math_libs/lib64
  # LD_LIBRARY_PATH: /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/math_libs/lib64
  # CPATH: /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/math_libs/include
- compiler:
  spec: nvhpc@21.9
  paths:
    cc: /opt/cray/pe/craype/default/bin/cc
    cxx: /opt/cray/pe/craype/default/bin/CC
    f77: /opt/cray/pe/craype/default/bin/ftn
    fc: /opt/cray/pe/craype/default/bin/ftn
  flags: {}
  operating_system: sles15
  target: any
  modules:
    - PrgEnv-nvidia
    - nvidia/21.9
    - craype-x86-milan
    - libfabric
- compiler:
  spec: nvhpc@21.11
  paths:
    cc: /opt/cray/pe/craype/default/bin/cc
    cxx: /opt/cray/pe/craype/default/bin/CC
    f77: /opt/cray/pe/craype/default/bin/ftn
    fc: /opt/cray/pe/craype/default/bin/ftn
  flags: {}
  operating_system: sles15
  target: any
  modules:
    - PrgEnv-nvidia
    - nvidia/21.11
    - craype-x86-milan
    - libfabric
- compiler:
  spec: gcc@11.2.0
  paths:
    cc: cc
    cxx: CC
    f77: ftn
    fc: ftn
  flags: {}
  operating_system: sles15
  target: any
  modules:
    - PrgEnv-gnu
    - gcc/11.2.0
    - craype-x86-milan
    - libfabric
  extra_rpaths: []
```

FIGURE 19. Compiler definition of gcc and nvhpc compiler on Perlmutter

On Jan 22, 2022 we released E4S/21.11 on Perlmutter which was our first deployment of E4S that was based on CPE 21.12. The initial release consisted of 94 packages, we provided TCL and Lua based modules generated by spack. We created a user-facing modulefile **e4s/21.11-tcl** and **e4s/21.11-lmod** that

can be used to access the same software stack but the main difference being the way modulefiles were presented. For more details on this stack see <https://docs.nersc.gov/applications/e4s/perlmutter/21.11/>.

In parallel to our standard deployment, we provide a containerized deployment of E4S/21.11 on Perlmutter which is a container image provided by the E4S team as part of their release process. The base image is an Ubuntu container using gcc@9.3.0 compiler, we provide this as an alternative to our software stack.

**Lessons Learned:** *The first time on any system can be challenging - have patience*

Perlmutter has gone through several changes in the past several months including upgrades to new CPE, we anticipate a few rebuilds will be required until Perlmutter is stable which is expected when bringing a new system into production.

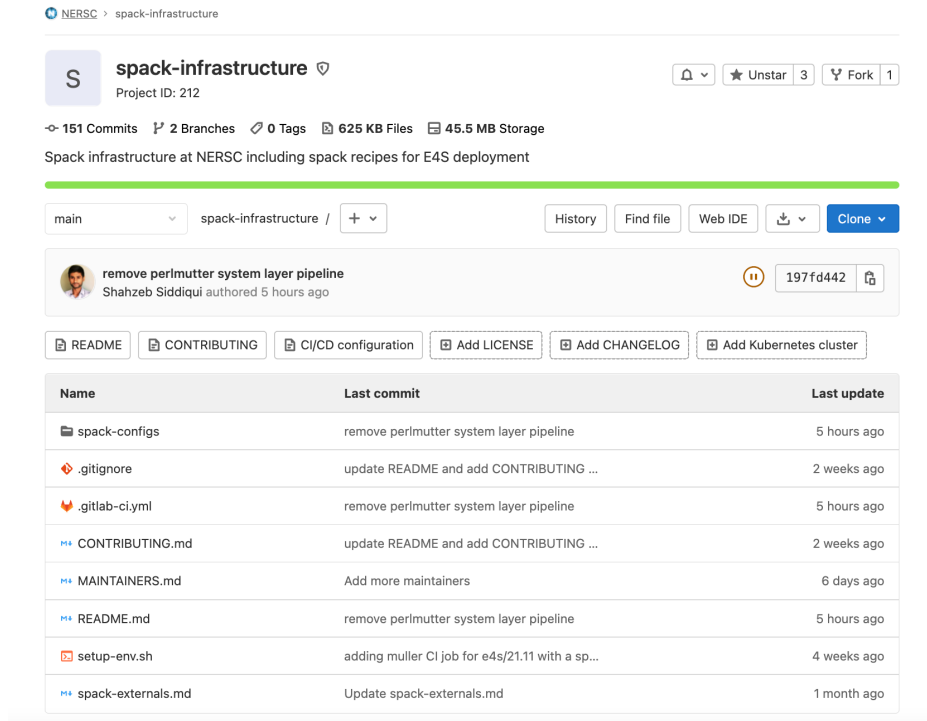
The E4S/21.11 is based on spack version 0.17 which had some significant changes including the **clingo** concretizer being the default going forward. This affected spack since it now required additional dependency to be installed during the bootstrapping process. We first encountered that spack was unable to bootstrap clingo on Perlmutter so we reported the issue <https://github.com/spack/spack/issues/28315> to spack project to get this resolved. Our current workaround was to install clingo via **pip** in order to satisfy the dependency.

There was a bug in spack in how system detection worked <https://github.com/spack/spack/issues/25914> which impacted how we do builds, we were unable to use spack command on Perlmutter which was a serious issue and this issue was addressed promptly by spack team.

## Automation

Recently we started a project to centralize our spack configuration and automate our spack deployments. We leverage Gitlab to automate our deployments using scheduled pipelines to perform full source builds of all of our spack stacks. This project is called spack-infrastructure and located at <https://software.nersc.gov/NERSC/spack-infrastructure>.

We have set up a public facing repo on Github at <https://github.com/NERSC/spack-infrastructure> which is a mirror of the original repo. In addition, we have user documentation available at <https://nersc-spack-infrastructure.readthedocs.io/en/latest/>



**FIGURE 20. NERSC Spack Infrastructure Project**

We have configured a few scheduled pipelines that perform full source builds of our E4S for various systems, these scheduled pipelines will mimic our production deployment but run in a unique directory per CI job.

Description	Target	Last Pipeline	Next Run	Owner	
cori-spack-develop	main	#49133	in 2 days	Shahzeb Siddiqui	▶ ✎ 🗑
perlmutter-spack-ci-develop	main	#47160	Inactive	Shahzeb Siddiqui	▶ ✎ 🗑
gerty-e4s-21.11	main	#49482	in 5 days	Shahzeb Siddiqui	▶ ✎ 🗑
muller-e4s-21.11	main	#49614	in 6 days	Shahzeb Siddiqui	▶ ✎ 🗑
perlmutter-e4s-21.11	main	#48875	in 21 hours	Shahzeb Siddiqui	▶ ✎ 🗑
perlmutter-spack-develop	main	#49253	in 3 days	Shahzeb Siddiqui	▶ ✎ 🗑

**FIGURE 21. An Overview of Scheduled Pipelines for each spack stack**

We have configured gitlab runners to run CI jobs on Cori, Perlmutter and our test systems so we can perform builds on all of our systems where E4S will be deployed. Our spack builds are performed using a single user account which avoids issues with differences between user environments.

Recently, we started building E4S using the spack [develop](#) branch which contains the bleeding edge of the spack codebase where incoming PRs get merged. We build this stack on a weekly basis which will build

the latest for each software product as new versions are added in spack codebase. These stacks are accessible via modules named **e4s/spack-develop** where we expose users with a spack instance.

```
e4s@login28> ml e4s/spack-develop
=====
Welcome to E4S stack built with spack develop
=====

This stack is rebuilt regularly with tip of spack 'develop' branch which means packages will change over
time. You can access the log files via $SPACK_INSTALL_LOG, $SPACK_CONCRETIZE_LOG, $SPACK_GITLAB_LOG. You can
view the spack configuration (spack.yaml, spack.lock) via environment $SPACK_YAML and $SPACK_LOCK.

For more information regarding this stack see https://software.nersc.gov/NERSC/spack-infrastructure.

e4s@login28> which spack
/global/cfs/cdirs/m3503/spackstacks/perlmutter/spack-develop/spack/bin/spack
```

We plan to leverage this stack as feedback into our E4S deployments and gain insight into what packages can build successfully in future deployments. Take for instance our spack develop stack for Perlmutter has deployed most recent versions of *kokkos@3.5.00* whereas our most recent deployment (e4s/21.11) contains *kokkos@3.4.01*. We have a high degree of confidence that packages installed via spack develop pipeline will most likely build in our future E4S deployments and this will ease our deployment process since this work is done in advance.

```
e4s@login28> spack find --format "{name}@{version} %{compiler.name}@{compiler.version}" kokkos
kokkos@3.5.00 %cce@13.0.0 kokkos@3.5.00 %gcc@9.3.0 kokkos@3.5.00 %nhpc@21.9
kokkos@3.5.00 %gcc@9.3.0 kokkos@3.5.00 %gcc@9.3.0
e4s@login28> ml e4s/21.11-tcl

The following have been reloaded with a version change:
 1) e4s/spack-develop => e4s/21.11-tcl

e4s@login28> spack find --format "{name}@{version} %{compiler.name}@{compiler.version}" kokkos
kokkos@3.4.01 %gcc@9.3.0 kokkos@3.4.01 %gcc@9.3.0
```

We recently deployed e4s/22.02 on Cori which followed a major system OS upgrade. The E4S deployment contained 385 installed specs, the most we have built so far, and the entire deployment was complete within 2 weeks. This stack was built with *gcc@11.2.0* and *intel@19.1.2.254*, shown below is a breakdown of specs by each compiler.

Compiler	Root Specs	Implicit Specs	Total Specs
gcc@11.2.0	70	166	236
intel@19.1.2.254	57	92	149
Total	127	258	385

We plan on supporting this release till the end of Cori lifetime (2023) and be our last deployment of E4S on Cori.

**FIGURE 22. Breakdown of installed specs by compilers for e4s/22.02**



## MPI Support

We are working with the [MVAPICH2](#) team from Ohio State University to experiment with **mvapich2** as an MPI provider for building the E4S stack on Perlmutter. The mvapich2-gdr is an optimized version of mvapich that takes advantage of GPU Direct RDMA technology to improve inter-node data movement on NVIDIA GPUs which is relevant for Perlmutter since we support NVIDIA A100 GPUs. Currently, we are using cray-mpich as our MPI provider which is available on our system but we have run into build errors with certain packages which expect mpi wrapper *mpicc* instead of *cc*. We plan on using cray-mpich as the MPI provider for building the stack and introduce **mvapich2** for building a subset of packages for future e4s release. The collaboration between the E4S and the NERSC teams has helped install MVAPICH2 and 87 packages with 575 total installed specs from E4S 22.02 as shown in the figures below. These packages use mvapich2-gdr configured with SLURM and CUDA 11.5 on Perlmutter. The total time for installation of these packages was less than one day! The E4S packages may be accessed using the module or spack commands as shown below.

```
sameer@perlmutter:login21:~> module use /global/common/software/spackcep/perlmutter/mvapich2/modulefiles
sameer@perlmutter:login21:~> module avail e4s/22.02

----- /global/common/software/spackcep/perlmutter/mvapich2/modulefiles -----
e4s/22.02 (D)

Where:
D: Default Module

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

sameer@perlmutter:login21:~> module load e4s

Lmod is automatically replacing "nvidia/21.11" with "gcc/10.3.0".

Due to MODULEPATH changes, the following have been reloaded:
1) cray-mpich/0.1.13

sameer@perlmutter:login21:~> spack find
==> 575 installed packages
-- cray-sles15-x86_64 / gcc@10.3.0 -----
adiak@0.2.1             heffte@2.2.0           nlohmann-json@3.10.5   py-jupyter-client@7.0.6   py-setuptools-scm@6.3.2
adios@1.13.1           hpc@2.2.0             nrm@0.1.0              py-jupyter-client@7.0.6   py-setuptools-scm-git-archive@1.1
adios2@2.7.1          hpctoolkit@2022.01.15 numactl@2.0.14         py-jupyter-core@4.7.1    py-six@1.16.0
adlbx@1.0.0            hpctoolkit@2022.01.15 nvhpc@2.1              py-jupyter-packaging@11@0.11.1  py-sniffio@1.2.0
alquimia@1.0.9        hpctoolkit@2022.01.15 omega-h@9.34.1         py-jupyter-packaging@7@0.7.12  py-sqlalchemy@1.4.20
aml@0.1.0              hpcviewer@2022.01    oniguruma@6.9.4        py-jupyter-server@1.11.2      py-statsmodels@0.12.2
amrex@22.02           hp@1.7.1              openblas@0.3.19        py-jupyter-telemetry@0.1.0     py-tables@3.6.1
ant@1.10.7            hpx@1.7.1             openjdk@11.0.12_7      py-jupyterhub@1.4.1          py-tblib@1.6.0
antlr@2.7.7           hwloc@2.7.0           openmpi@4.1.2          py-jupyterlab@3.2.1          py-terminado@0.12.1
arborx@1.1            hwloc@2.7.0           openmpi@4.1.2          py-jupyterlab-server@2.6.0     py-threadpoolctl@3.0.0
archer@2.0.0          hwloc@2.7.0           openmpi@4.1.2          py-jupyterlab-widgets@1.0.2   py-tomli@0.10.2
argobots@1.1          hypre@2.23.0          openmpi@4.1.2          py-kiwisolver@1.3.2          py-tomli@1.2.2
arpack-ng@3.8.0       hypre@2.24.0          openmpi@4.1.2          py-lazy-object-proxy@1.4.3    py-tomlkit@0.7.2
asio@1.21.0           hypre@2.24.0          openmpi@4.1.2          py-lhsmdu@1.1                py-tornado@5.1.1
autoconf@2.69         inputproto@2.3.2      openturns@master       py-libensemble@0.8.0         py-tornadoc@1.1.1
autoconf-archive@2019.01.06 intel-tbb@2020.3      otf2@2.3                py-mako@1.1.5                py-traitlets@5.1.1
autotake@1.16.5       intel-xed@12.0.1     pagmo2@2.18.0          py-markupsafe@2.0.1          py-typeguard@2.12.1
axl@0.3.0             jansson@2.13.1       papi@6.0.0.1           py-markuptool@3.5.1          py-typing-extensions@3.10.0.2
axl@0.5.0             jq@1.6                papi@6.0.0.1           py-matplotlib@3.5.1         py-urllib3@1.26.6
axom@0.6.1            json-c@0.15           papi@6.0.0.1           py-matplotlib-inline@0.1.2    py-vcversioner@2.16.0.0
berkeley-db@18.1.40   jsoncpp@1.9.4         papyrus@1.0.1          py-mccabe@0.6.1              py-warlock@1.3.3
binutils@2.33.1       kbroto@1.0.7          parallel-netcdf@1.12.2 py-mistune@0.8.4             py-warpx@22.02
binutils@2.36.1       kim-api@2.2.1         paraview@5.10.0        py-mock@4.0.3                py-warpx@22.02
bison@3.8.2           kokkos@3.5.00         parametris@4.0.3       py-mpi4py@3.1.2              py-warpx@22.02
blaspp@2021.04.01     kokkos@3.5.00         parsec@3.0.2012        py-mpi4py@3.1.2              py-wcwidth@0.2.5
blaspp@2021.04.01     kokkos@3.5.00         pcre@8.44               py-msgpack@1.0.2             py-websocket-client@1.2.1
blaspp@2021.04.01     kokkos@3.5.00         pcre2@10.36            py-natsort@7.1.1             py-wheel@0.37.0
```

FIGURE 23. How to access e4s/22.02 stack built with mvapich2-gdr

```
1: adios2 /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/adios2-2.7.1-e32a7r2jmxo1c1schmf55NryJqzklc
2: alquimia /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/alquimia-1.0.9-ivqo5v14326hbx62j5e3zqweiy6
3: aml /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/aml-1.0-mqj3dfp3rjwgxvqaoaf7eqzqq4xeb
4: amrex /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/amrex-22.02-776hps2725bpaq2top6651371lo4c
5: arborx /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/arborx-1.1-2rjaszm7s19y7u0t6kb3vtvcvndj4j5
6: archer /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/archer-2.0-0-wm53g2v4yzvovtoipmp44ijp26h7ap7
7: argobots /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/argobots-1.1-rj454sub3oj36cdpzbpbv4wacz5sm
8: axom /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/axom-2.1.0-bwa7576p72efsvutohxkaaf7ag76b5v7
9: bolt /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/bolt-2.0-pkmj3wj2gac32cvflatbh27awb4leqr
10: butterflypack /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/butterflypack-2.1.0-rbbfvyk4kagjcdnccq6luouubzsfte
11: cabana /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/cabana-0.4.0-w7j5cnyv5b0p161ckeks5wof1zpl2
12: caliper /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/caliper-2.7.0-d25ai5epf6m3j1myyqppn7bnmnekbh
13: chai /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-11.2.0/chai-2.4.0-63jmijay7uvncmj4y5zic6365
14: charliecloud /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/charliecloud-0.26-ah3k515oat424hy27l3vtfv1qishxp
15: conduit /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/conduit-0.8.2-nuyhi2g2z361xv363nmmwyxz2m7i
16: darshan-runtime /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/darshan-runtime-3.3.1-ksd2llesmejdfk46j0ngmtofxyure
17: datatransferkit /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/datatransferkit-3.1-rc3-df7ybbprkra15wt6k0k3l5pn25obx
18: dyninst /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/dyninst-12.0.1-1mstpkadccya24roge6v4yh11335td3e
19: faodel /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/faodel-1.2108l-1-agnopokupkzewj4jshjzutt1kiahgm47b
20: flecsi /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/flecsi-2.1.0-1-c8071em56ber7b7sr7h5bkb2otmt2
21: flit /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/flit-2.1.0-xop344w4r45r4sd20ps44k59k44nc
22: flux-core /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/flux-core-0.35.0-qcliktq4ntl4m45024rtobcrgxm
23: fortrilinos /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/fortrilinos-2.0.0-yvowkhy4v6kgkx6e1gxfexjagix4xhl
24: gasnet /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/gasnet-2021.9.0-zod233yapkhgu6n9ihub3n34idz
25: ginkgo /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/ginkgo-1.4.0-lzxbftoktmjzpcntjpnba77frnl06
26: globallarrays /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/globallarrays-5.8-ewoxa2x7f12lglab16m9hd4u4h3grvry
27: gotcha /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/gotcha-1.0.3-z5w5piunmjk4s2c3252vvo1ahdKfy3z
28: gptune /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/gptune-2.1.0-7o2auwfvfem31a5b51jcaclwrh6lk22ae
29: hpctoolkit /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/hpctoolkit-2022.01.15-3m5e25326vtyrtdmhwj545uvs1oa2g
30: hpcx /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/hpcx-1.7.1-57fy2mtzdpwkwkndshmhv2jpe1xhkr
31: hypr /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/hypr-2.24.0-y5ybjul1euz5g51odmb4mrc4rl2
32: kokkos /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/kokkos-3.5.0-ptvov6rd1ndm3vrsrat1ajcwnzoyasad
33: kokkos-kernels /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/kokkos-kernels-3.5.0-3kx3i3gyxvaauk12sef5lberhivr
34: lammps /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/lammps-20220107-bphr6p6ocw4lweakz26aulshsf6fw
35: legion /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/legion-21.03.0-yohu6640ilc34mj730tk2woezatq
36: libnrm /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/libnrm-0.1.0-sngm35btw3p3ypp6cygdj1m6736yfp
37: libquo /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/libquo-1.3.1-6v27godixtrh506c5yq01a3pshs3j
38: loki /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/loki-1.7-41byadoc32byl3r3ep254tmban3j3zt
39: magma /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/magma-2.6.1-4wn51j26btyrdhmmjy54kvg55uvs1oa2g
40: mercury /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/mercury-2.1.0-6wkyl94k6q3j7owm9nj33c3cmuh
41: metall /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/metall-0.17-wuz4afad7f7rrvooaeuuyqgv50dbnj
42: mfem /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/mfem-4.3.0-vvgip4c4dyvub5cm23gtvdk1ve7b
43: mparc-variant /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/mparc-variant-1.4.0-3ma9sxd6cavxonin4nboeusa3d3ae
44: mpifileutils /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/mpifileutils-0.11.1-pddpzmxof6t5j3dr2r6ajy5lghyag
45: netlib-scalapack /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/netlib-scalapack-2.1.0-iicv6t6wysqy34r7ntdf37dqmgwv
46: ncemp /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/ncemp-1.9.0.1-zb725hcfbznyaht1tsuqxgvkrkq505
47: nco /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/nco-5.0.1-ky3p7d77ebtcnkw3v4kg753mo79sp2k
48: njia /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/njia-1.10.2-nk5xh3rhv9uq3j7owm9nj33c3cmuh
49: nrm /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/nrm-0.1.0-vxyx4esgwrd5tbny4uh5jfr6a27roq
50: omega-h /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/omega-h-9.34.1-4gwbhr54femhuibutvcf236hjjpfb7h
51: openpmd-api /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/openpmd-api-0.14.4-uwrsxmuuzywzkxx5l1dmguyibenmdm
52: openmpi /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/openmpi-4.1.2-fkgs4yrb3vnguuqj32cbm2p50s6f6w
53: papi /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/papi-6.0.0-1-quimmd22mbkmffdvcsq33hjcra5jw
54: papyrus /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/papyrus-1.0.1-253qw045mdg4m375xweznyby5bzqf
55: parallel-netcdf /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/parallel-netcdf-1.12.2-2ab4mouhza2mguzfxkrce4tm5laqbn
56: paraview /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/paraview-5.10.0-15uh7vm7o2hpf0sh4w24173vued6qg
57: parsec /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/parsec-3.0.2012-x5z14t4bop6gf7n7aoy616u1m6oqsw
58: pdt /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/pdt-3.25.1-fnysmpjas72ip7k7akbhn2cd4l3r2
59: petsc /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/petsc-3.16.4-mzy13xprthtt5xf7a7ahj251entg
60: phist /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/phist-1.9.5-42bebw7oqdsj5pxjlbjncxjyymb25
61: plasma /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/plasma-21.8.29-4er443xvrrat232bcoedlrcrcyga0
62: plumed /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/plumed-2.6.3-v2jllt3jossdjakgs1bv35dy17kkr636
63: precision /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/precision-2.3.0-gy3ivcvncpe7jotdmo6byj733krk636
64: pumi /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/pumi-2.2.6-inwmiumym24oy7zm2z6e7nd73l
65: py-cinemas /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/py-cinemas-1.3-5jcgd2w54zn1awvq6612z4hyojlrfy
66: py-jupyterhub /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/py-jupyterhub-1.4.1-1p1uxdrux14aqep54zdz2thjvhyt7ryz
67: py-libensemble /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/py-libensemble-0.8.0-42158v4u7e7e7m12697gzmzuw7
68: qthreads /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/qthreads-1.16-xkxykuo6ntsd1lcrs4k0kibcaag3y3
69: raja /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/raja-0.14.0-mzunvttmxnsx7jfhj3q7uwend7htjup
70: scr /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/scr-3.0rc2-tsyt4dsidtpid2o1k6isabthpmpvflp
71: slate /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/slate-2021.05.02-tfvg5cha3zq6kh3xhmoa2v6hy21o2
72: slepc /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/slepc-3.16.2-esjwua3vm02kai1vkk15a6w500oqcy
73: strumpack /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/strumpack-6.3.0-h5hw6ppqkird7thtqmf6f6gkjznsz
74: sundials /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/sundials-6.1.1-dhhzydlxoyem6whbhue6wae5ehu17f
75: superlu-dist /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/superlu-dist-7.2.0-ffwbjokonxd34m35px7tux6d554dvkg
76: swig /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/swig-4.0.2-fortran-c3fme33qrkumwdeg75xutz3a46dhyh1
77: sz /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/sz-2.1.12-24oshntcac224eobq5kqnm2f2cnz
78: tasmannian /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/tasmannian-7.7-kfmgvyy2vzh2b6b77k7adapxag5cgvf
79: tau /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/tau-2.31.1b2-4r423k3vj3u594dnnmfcdnca406h2a
80: trilinos /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/trilinos-13.0.1-u0qbyzzj2thxh5uyl3cia5f5pe3j
81: umap /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/umap-2.1.0-h2sf1ggmkb5k5khen24oxg7r4qfyyg535c
82: umpire /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-11.2.0/umpire-6.0.0-q3en73h243gdilw4743240vwdpes4p
83: upcxx /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/upcxx-2021.9.0-3xkq454nm0z3kuabq6l1nf42mbnqo44
84: veloc /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/veloc-1.5-ymh1yiuwennpnwmr1z7lpbmevdewc
85: vtk-m /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/vtk-m-1.7.1-hwtmgvyy2vzh2b6b77k7adapxag5cgvf
86: wannier90 /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/wannier90-3.1.0-t24yqoy73p6a1imkcc3bqv2kqarouy
87: zfp /global/cfs/cdirs/m3896/sameer/E4S/22.02/spack/opt/spack/cray-sles15-x86_64/gcc-10.3.0/zfp-0.5.5-we5hdzuxkvr9736ai1mckcc3bqv2kqarouy
```

FIGURE 24. Listing of E4S 22.02 packages built using MVAPICH2-GDR on Perlmutter

# Container-based deployment of E4S

Besides bare-metal installation of E4S, NERSC also supports [Shifter](#), a mature HPC container runtime and both base and full-featured E4S images are installed on Perlmutter. These images contain 100+ HPC and AI/ML packages (such as TensorFlow/PyTorch) with total 621 specs and support the A100 GPUs. Shifter provides a viable deployment option for E4S where only one image needs to be downloaded and is immediately available to all the users. They may continue to use module or spack commands to access the

packages, as shown in the figure below. Using E4S base or full-featured containers, users may build their own compact, custom configured containers to deploy on any HPC system.

```
sameer@perlmutter:login37:~> shifterimg images | grep e4s | grep 22.02
perlmutter docker      READY    ebac4ca421  2022-01-28T12:04:27  ecpe4s/e4s-base-cuda:22.02
perlmutter docker      READY    a6e82b4a2e  2022-03-04T16:16:38  ecpe4s/ubuntu20.04-gpu-x86_64:22.02
sameer@perlmutter:login37:~> shifter -E --image=ecpe4s/ubuntu20.04-gpu-x86_64:22.02 /bin/bash --ccfile /etc/bashrc
(base) sameer@login37:/global/u1/s/sameer$ nvidia-smi
Tue Mar 29 12:41:11 2022
```

NVIDIA-SMI 450.162 Driver Version: 450.162 CUDA Version: 11.5									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.		
0	A100-PCIE-40GB	Off	00000000:03:00:0	Off	0%	Default	0		
N/A	39C	P0	39W / 250W	0MiB / 40537MiB		Disabled			

```
(base) sameer@login37:/global/u1/s/sameer$ which spack
/spack/bin/spack
(base) sameer@login37:/global/u1/s/sameer$ module avail
Rebuilding cache, please wait ... (written to file) done.
```

GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
ID	ID	ID				
No running processes found						

```
----- /extra-modules -----
julia/1.7.2    visit/3.2.2

----- /spack/share/spack/modules/linux-ubuntu20.04-x86_64 -----
adios/1.13.1-sch          mpich/3.4.2-b2h
adios2/2.7.1-vrq          mpich/3.4.2-rwz          (D)
alquimia/1.0.9-5px        mpifileutils/0.11.1-x6m
aml/0.1.0-52b             nccmp/1.9.0.1-2hk
amrex/22.02-cuda70-5pl    nco/5.0.1-nzq
amrex/22.02-cuda80-xjb    netlib-scalapack/2.1.0-mjx
amrex/22.02-l4b           ninja/1.10.2-2gp
amrex/22.02-rocm90a-dlz   nrm/0.1.0-omw
amrex/22.02-rocm908-wne   (D)  nvhpc/22.1-367
arborx/1.1-x54            omega-h/9.34.1-syd
arborx/1.1-5ta            openjdk/11.0.12_7-pnc
arborx/1.1-666           (D)  openmpi/4.1.2-5yq
archer/2.0.0-wkd          openmpi/4.1.2-zzl          (D)
argobots/1.1-6vb         openpmd-api/0.14.4-k6w
ascent/0.7.1-y6o         papi/6.0.0.1-bri
axom/0.6.1-vbd           papi/6.0.0.1-mjw          (D)
bolt/2.0-fh7             papyrus/1.0.1-oaw
butterflypack/2.1.0-vwr  parallel-netcdf/1.12.2-ojo
cabana/0.4.0-uba         paraview/5.10.0-ai6
caliper/2.7.0-cuda70-74r  parsec/3.0.2012-cuda70-jhe
caliper/2.7.0-cuda80-xvp  parsec/3.0.2012-cuda80-mm
caliper/2.7.0-nml         (D)  parsec/3.0.2012-etb          (D)
catalyst/5.6.0-5ao       pdt/3.25.1-c57
chai/2.4.0-cuda70-2jg    petsc/3.16.4-cuda70-kqu
chai/2.4.0-cuda80-nkj    petsc/3.16.4-cuda80-zde
chai/2.4.0-fgi           petsc/3.16.4-cvs
```

FIGURE 25. Using E4S 22.02 container with Shifter on Perlmutter

The use of conda and the cuda environment in Shifter to use TensorFlow and PyTorch packages with support for A100 GPU on Perlmutter is shown in the figure below.

```
(base) sameer@login37:/global/u1/s/sameer$ conda activate cuda
(cuda) sameer@login37:/global/u1/s/sameer$ python
Python 3.9.7 (default, Sep 16 2021, 13:09:58)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>> import torch
>>> import cv2
>>> import numpy
>>> tensorflow.__version__
'2.8.0'
>>> torch.__version__
'1.10.2+cu113'
>>> tensorflow.test.is_gpu_available()
WARNING:tensorflow:From <stdin>:1: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
2022-03-29 12:49:15.351054: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with one API Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-03-29 12:49:32.537543: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /device:GPU:0 with 38421 MB memory: -> device: 0, name: A100-PCIE-40GB, pci bus id: 0000:c3:00.0, compute capability: 8.0
True
>>>
(cuda) sameer@login37:/global/u1/s/sameer$ spack find
==> 621 installed packages
-- linux-ubuntu20.04-x86_64 / dpcpp@2022.0.0 -----
kokkos@develop


-- linux-ubuntu20.04-x86_64 / gcc@9.3.0 -----
adiak@0.2.1          kbproto@1.0.7          protobuf@3.18.0       py-ruamel-yaml@0.17.16
adios@1.13.1        kim-api@2.2.1          pugixml@1.11.4        py-ruamel-yaml-club@0.2.0
adios2@2.7.1        kokkos@3.5.00         pumi@2.2.6            py-scikit-learn@1.0.2
adlbx@1.0.0         kokkos@3.5.00         py-alembic@1.5.5      py-scikit-optimize@master
alquimia@1.0.9      kokkos@3.5.00         py-anyio@3.3.4        py-scipy@1.8.0
alsa-lib@1.2.3.2    kokkos@3.5.00         py-apache-libcloud@1.2.1
aml@0.1.0           kokkos@3.5.00         py-argon2-cffi@21.1.0
amrex@22.02         kokkos@3.5.00         py-astroid@2.8.3      py-send2trash@1.8.0
amrex@22.02         kokkos-kernels@3.5.00 py-async-generator@1.10
amrex@22.02         kokkos-kernels@3.5.00 py-attrs@21.2.0        py-serpent@1.40
amrex@22.02         kokkos-kernels@3.5.00 py-babel@2.9.1         py-setproctitle@1.1.10
amrex@22.02         kokkos-nvcc-wrapper@3.2.00
antlr@2.7.7         kokkos-nvcc-wrapper@3.2.00
arborx@1.1          kvtree@1.2.0           py-backcall@0.2.0     py-setuptools-rust@0.12.1
arborx@1.1          lammps@20220107        py-bcrypt@3.2.0       py-six@1.16.0
arborx@1.1          lapackpp@2021.04.00    py-beniget@0.4.1      py-sniffio@1.2.0
archer@2.0.0        lapackpp@2021.04.00    py-blinker@1.4         py-sqlalchemy@1.4.20
argobots@1.1        lapackpp@2021.04.00    py-bottleneck@1.3.2   py-statsmodels@0.12.2
arpack-ng@3.8.0     lapackpp@2021.04.00    py-bottleneck@1.3.2   py-tables@3.6.1
ascent@0.7.1        legion@21.03.0          py-certifi@2021.10.8  py-tblib@1.6.0
asio@1.21.0         libarchive@3.5.2        py-certipy@0.1.3      py-terminado@0.12.1
axl@0.3.0           libbsd@0.11.3          py-cffi@1.15.0        py-threadpoolctl@3.0.0
axl@0.5.0           libcap@2.25             py-charset-normalizer@2.0.7
axom@0.6.1          libcircle@0.3.0        py-cinemasci@1.3       py-toml@0.10.2
berkeley-db@18.1.40 libdwarf@20180129      py-cloudpickle@1.6.0  py-tornado@5.1.1
binutils@2.33.1     libedit@3.1-20210216  py-colorama@0.4.4     py-tornadoc@6.1
binutils@2.36.1     libevent@2.1.12        py-configspace@0.4.20 py-traitlets@5.1.1
blaspp@2021.04.01  libfabric@1.14.0       py-cryptography@3.4.8 py-typeguard@2.12.1
blaspp@2021.04.01  libffi@3.3              py-cryptography@35.0.0 py-typing-extensions@3.10.0.2
                    py-cycler@0.11.0      py-cython@0.29.24     py-urllib3@1.26.6
                    py-cytoolz@0.12.0     py-warlock@1.3.3      py-warp@22.02
                    py-dask@2021.12.0     py-wrapt@1.12.1
```

FIGURE 26. Running TensorFlow from Shifter container

## Testing E4S Post Deployment

We test our E4S stack post deployment, we utilize [buildtest](#) a testing framework to build and run tests on HPC systems. We utilize gitlab to run a subset of E4S tests on Cori and Perlmutter for each of our E4S stacks along with the shifter based container. Our test can be found on our NERSC gitlab server at <https://software.nersc.gov/NERSC/buildtest-nersc>. Shown below is a listing of scheduled pipelines that will run a subset of tests at different schedules. Since we have over 200+ tests we can't run all of them at once, but instead we run at different intervals.

NERSC > buildtest-nersc > Schedules



### Scheduling Pipelines

The pipelines schedule runs pipelines in the future, repeatedly, for specific branches or tags. Those scheduled pipelines will inherit limited project access based on their associated user.

Learn more in the [pipeline schedules documentation](#).

All 6 Active 6 Inactive 0 New schedule

Description	Target	Last Pipeline	Next Run	Owner	
Perlmutter Check	✓ devel	🔴 #54736	in 1 day	Shahzeb Siddiqui	▶ ✎ 🗑️
Perlmutter E4S Tests	✓ devel	🟢 #54591	in 7 hours	Shahzeb Siddiqui	▶ ✎ 🗑️
Cori Benchmark	✓ devel	🔵 #55201	in 6 days	Shahzeb Siddiqui	▶ ✎ 🗑️
Cori Application Test	✓ devel	🟢 #55001	in 4 days	Shahzeb Siddiqui	▶ ✎ 🗑️
Cori E4S Test	✓ devel	🔵 #55223	in 6 days	Shahzeb Siddiqui	▶ ✎ 🗑️
Cori Daily Check	✓ devel	🔵 #55062	in 5 days	Shahzeb Siddiqui	▶ ✎ 🗑️

FIGURE 27. Scheduled pipeline for testing E4S stack

buildtest will publish results to CDASH upon completion of all tests, shown below is an output from the gitlab job that runs E4S tests on the Perlmutter system. Buildtest will show a link to CDASH report file which can be viewed in your browser.

```

1206 $ buildtest cdash upload e4s
1207 Reading report file: /global/cfs/cdirs/m3503/ci-builds/perlmutter/yUW7FC66/0/NERSC/buildtest-nersc/buildtest/var/report.json
1208 build name: e4s
1209 site: perlmutter
1210 stamp: 20220322-2103-Experimental
1211 MD5SUM: 3bb737d4a3e3a4f44f24ac11eaea493c
1212 PUT STATUS: 200
1213 You can view the results at: https://my.cdash.org/viewTest.php?buildid=2144892

```

The CDASH report will contain metadata for each test such as name of test, test description, hostname, start and endtime, test duration. CDASH will report the test failures in **RED**.

5 passed, 7 failed, 0 not run, 0 missing.

Name	Status	Time	Details	Labels	History	Summary	compiler	description	endtime	hostname	starttime	user
amrex_single_vortex	Failed	16m 43s 780ms		e4s		Unstable Broken		AmrLevel SingleVortex Build and Run	2022/03/22 21:20:40	login16	2022/03/22 21:03:56	e4s
e4s_testsuite_for_e4s_21.11	Failed	24m 16s 210ms		e4s		Broken Broken		Run E4S Test suite for e4s/21.11- tci stack	2022/03/22 21:28:12	login16	2022/03/22 21:03:56	e4s
kokkos_CUDA_axpby	Failed	8s 810ms		e4s		Unstable Broken		kokkos axpby CUDA example	2022/03/22 21:04:05	login16	2022/03/22 21:03:56	e4s
spack_test_hypre_e4s_21.11	Failed	1m 37s 300ms		e4s		Broken Broken		Test hypre build with for e4s/21.11 stack on Perlmutter via spack test	2022/03/22 21:05:33	login16	2022/03/22 21:03:56	e4s
spack_test_kokkos_e4s_21.11	Failed	33s 510ms		e4s		Broken Broken		Test kokkos build for e4s/21.11 stack on Perlmutter	2022/03/22 21:04:29	login16	2022/03/22 21:03:56	e4s
spack_test_superlu_e4s_21.11	Failed	42s 500ms		e4s		Broken Broken		Test superlu build for e4s/21.11 stack on Perlmutter	2022/03/22 21:04:38	login16	2022/03/22 21:03:56	e4s
spack_test_upcxx_e4s_21.11	Failed	47s 210ms		e4s		Broken Broken		Test upcxx build with for e4s/21.11 stack on Perlmutter via spack test	2022/03/22 21:04:43	login16	2022/03/22 21:03:56	e4s
kokkos_OpenMP_axpby	Passed	8m 10s 610ms		e4s		Stable Stable		kokkos axpby OpenMP example	2022/03/22 21:12:06	login16	2022/03/22 21:03:56	e4s
spack_test_gasnet_e4s_21.11	Passed	41s 300ms		e4s		Stable Stable		Test gasnet build with for e4s/21.11 stack on Perlmutter via spack test	2022/03/22 21:04:37	login16	2022/03/22 21:03:56	e4s

FIGURE 28. CDASH output for E4S runs on Perlmutter

Shown below is an output from one of our test which will validate trilinos package by testing Zoltan from our shifter container on Perlmutter using two nodes. This test will calculate Preconditioned Conjugate Gradient for problem **Epetra:VbrMatrix** which will run for 20 iterations.

```

- for initial setup = 0.011301 (s)
- for hierarchy setup = 0.503612 (s)
- for smoothers setup = 0.000122315 (s)
- for coarse setup = 0.000708673 (s)
- for final setup = 3.6179e-05 (s)
Total for this setup = 0.519751 (s)
-----

***** Problem: Epetra:VbrMatrix
***** Preconditioned CG (with condnum) solution
***** ML (L=4, Cheby_pre0/Cheby_post0, ~/Amesos_KLU_3)
***** No scaling
*****

iter: 0      residual = 1.000000e+00
iter: 1      residual = 1.883147e-01
iter: 2      residual = 5.699805e-02
iter: 3      residual = 1.241147e-02
iter: 4      residual = 3.504143e-03
iter: 5      residual = 8.412063e-04
iter: 6      residual = 2.174022e-04
iter: 7      residual = 5.355959e-05
iter: 8      residual = 1.314152e-05
iter: 9      residual = 3.285323e-06
iter: 10     residual = 7.857157e-07
iter: 11     residual = 2.023098e-07
iter: 12     residual = 4.753736e-08
iter: 13     residual = 1.206472e-08
iter: 14     residual = 2.891014e-09
iter: 15     residual = 7.290301e-10
iter: 16     residual = 1.776775e-10
iter: 17     residual = 4.391331e-11
iter: 18     residual = 1.117004e-11
iter: 19     residual = 2.712616e-12
iter: 20     residual = 6.942872e-13

-----

Analysis of the Lanczos matrix of
the preconditioned system:

smallest eigenvalue      = 3.696450e-01
largest eigenvalue       = 9.988483e-01
estimated condition number = 2.702183e+00

-----

Solution time: 0.635757 (sec.)
total iterations: 20
||b-Ax||_2 = 2.05413e-10
-----

ML time information (seconds)      total      avg
1- Construction                    = 0.504594  0.504594
2- Preconditioner apply             = 0.521902
  a- first application(s) only      = 0.0306913 0.0306913
  b- remaining applications         = 0.491211  0.023391
3- Total time required by ML so far is 1.0265 seconds
   (constr + all applications)
-----

TEST PASSED

```

FIGURE 29. Output for Trilinos Zoltan test in Shifter container

Shown below is the generated build script and test script by buildtest. The test will utilize image **ecpe4s/ubuntu20.04-gpu-x86\_64:21.11** which is E4S 21.11 stack built with GPU support. The Zoltan test is available in E4S Testsuite (<https://github.com/E4S-Project/testsuite>), test will allocate 2 nodes with 4 GPUs and run the test from shifter container via **srun**. In order to run the Zoltan trilinos test, we need to load trilinos via **spack load trilinos**.

```

Build Script Content
#!/bin/bash
export BUILDTEST_TEST_NAME=trilinos_zoltan_cuda
export BUILDTEST_TEST_ROOT=/global/cfs/cdirs/m3503/buildtest/runs/perlmutter_e4s/2022-03-22/perlmutter.slm.regular/zoltan/trilinos_zoltan_cuda/a05fe01c
export BUILDTEST_BUILDSPC_DIR=/global/cfs/cdirs/m3503/ci-builds/perlmutter/yUW7FC66/0/NERSC/buildtest-nersc/buildspecs/apps/trilinos
export BUILDTEST_STAGE_DIR=/global/cfs/cdirs/m3503/buildtest/runs/perlmutter_e4s/2022-03-22/perlmutter.slm.regular/zoltan/trilinos_zoltan_cuda/a05fe01c/stage
# source executor startup script
source /global/cfs/cdirs/m3503/ci-builds/perlmutter/yUW7FC66/0/NERSC/buildtest-nersc/buildtest/var/executor/perlmutter.slm.regular/before_script.sh
# Run generated script
sbatch --parsable -q regular --account=m3503_g /global/cfs/cdirs/m3503/buildtest/runs/perlmutter_e4s/2022-03-22/perlmutter.slm.regular/zoltan/trilinos_zoltan_cuda/a05fe01c/stage/trilinos_zoltan_cuda.sh
# Get return code
returncode=$?
# Exit with return code
exit $returncode

Test Content
#!/bin/bash
#SBATCH -N 2
#SBATCH -t 5
#SBATCH -G 4
#SBATCH -C gpu
#SBATCH -A m3503_g
#SBATCH --image=cepe4s/ubuntu20.04-gpu-x86_64:21.11
#SBATCH --job-name=trilinos_zoltan_cuda
#SBATCH --output=trilinos_zoltan_cuda.out
#SBATCH --error=trilinos_zoltan_cuda.err
# Content of run section
git clone https://github.com/E4S-Project/testsuite
cd testsuite/validation_tests/trilinos-cuda
shifter --image=cepe4s/ubuntu20.04-gpu-x86_64:21.11 -E -- ./compile.sh
srun -n 2 shifter -- /bin/bash -c 'unset CRAYPE_VERSION; unset MODULEPATH ; . /spack/share/spack/setup-env.sh; spack load trilinos+cuda cuda_arch=80 ; spack unload mpich; export LD_LIBRARY_PATH=/opt/udlImage/modules/mpich/dep:/SLD_LIBRARY_PATH ; ./build/Zoltan'

```

FIGURE 30. Build Script and Generated Test for Trilinos Zoltan

## Conclusion

Managing an HPC software environment can be a challenging and time-consuming process for any HPC center. Deploying a software stack requires intimate knowledge of the HPC system with in-depth knowledge of the software packages to ensure each package is built optimally for the system. E4S accelerates the development, deployment and use of HPC software, lowering the barriers for HPC users. The E4S software stack community effort creates policies and necessary infrastructure to more easily and quickly deploy software at extreme-scale.

The sheer size of the E4S deployment and the constant upgrades in cutting-edge HPC system technology requires tight integration with HPC facility staff and across the community. The continued success and development of E4S and similar efforts will need to additionally emphasize building the community of support to maintain longevity and impact of the software. In particular, workforce development and community building:

**Workforce Development:** The software deployment team is an integral part of HPC centers, and more focused efforts are needed towards training our existing staff and/or increasing the workforce to support initiatives like E4S at the facilities. An HPC center may have multiple HPC systems and if one wants to deploy E4S for every system we should work towards a sustainable solution where we can deploy E4S relatively quickly while having additional resources so work can be done in parallel. Across the three DOE labs (NERSC, OLCF, ALCF) we noticed that the software deployment group is led by 1-2 individuals who are responsible for building the entire software stack for multiple HPC systems. Since E4S leverages spack as the driver for building E4S stack, this means staff also need spack expertise and a strong sense of how to design software stacks and interface through modules.

**Community Building:** HPC centers can benefit from each other by sharing best practices in the software deployment process, especially for centers that don't have a well-established process or are trying to deploy an E4S stack for the first time. This report is an effort to share more detailed deployment process information with the community - a behind the scenes look. We encourage others to do similarly.



## How to Get Involved

There are several ways to get involved to better support E4S at the facilities.

**Are you an application developer for an E4S product that we install? We need your assistance in troubleshooting the build errors.** Each facility wants to accelerate the software deployment process and provide as many software products as possible to satisfy the user's needs. However we need your help to debug build failures during our deployment. You will need access to NERSC resources along with our gitlab project <https://software.nersc.gov/NERSC/spack-infrastructure/> which contains our spack configuration along with build logs and current issues that need to be addressed. We will try to post these issues in [spack issue tracker](#) to get more visibility. We can get you set up!

**Are you an application developer or user of an E4S software package? We seek guidance on package variants when building a package.** Take for instance, trilinos which comes with several dozen variants

```
trilinos +amesos +amesos2 +anasazi +aztec +belos +boost +epetra +epetraext +ifpack
+ifpack2 +intrepid +intrepid2 +isorropia +kokkos +ml +minitensor +muelu +nox
+piro +phalanx +rol +rhythm +sacado +stk +shards +shylu +stokhos +stratimikos
+teko +tempus +tpetra +trilinoscouplings +zoltan +zoltan2 +superlu-dist gotype=long_long
```

We are unsure if all of these variants are appropriate for our system. Some of these selections were provided by E4S which we incorporated in our spack configuration. We leverage multiple compilers for building E4S stack including gcc, cce, nvhpc, and intel and we would suggest application teams to provide feedback into our decision process since we may choose incompatible compilers or compiler versions.

**Are you an application developer for an E4S software package that we install? We seek your guidance in testing the software on our system.** We are trying to increase test coverage for our e4s deployment by having at least 1-2 sanity tests that can test the software product to increase confidence. You will need access to NERSC resources and our gitlab server <https://software.nersc.gov/NERSC/buildtest-nersc> with all of our E4S tests. We encourage user contribution to help sustain this effort. Our focus is to test the software provided by E4S stack which will be accessible via **module load e4s**. Currently we are trying to add tests for the latest E4S release with an emphasis on developing tests for Perlmutter.

## Acknowledgement

This work was supported by the Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

We gratefully acknowledge Hai Ah Nam (LBNL, IDEAS-ECP Better Scientific Software Fellowship Coordinator) for her many edits and guidance on this report and the feedback from the ECP E4S collaboration and NERSC staff.

## Bio

### Shahzeb Siddiqui

Shahzeb Siddiqui is a HPC Consultant/Software Integration Specialist at [Lawrence Berkeley National Laboratory](#) at NERSC. He is part of the [User Engagement Team](#) that is responsible for engaging with NERSC user community through user support tickets, user outreach, training, documentation. Shahzeb is part of the [Exascale Computing Project \(ECP\)](#) in [Software Deployment \(SD\)](#) group where he is responsible for building Spack [Extreme-Scale Scientific Software Stack \(E4S\)](#) at the DOE facilities. He is the creator of few open source projects including [buildtest](#), [lmodule](#) and [jobstats](#). Shahzeb has experience installing and managing large software stack, managing HPC clusters including cluster managers (Bright Cluster Manager, Cobbler) and configuration management tools such as Ansible.

Shahzeb Siddiqui started out his career in High Performance Computing (HPC) in 2012 at [King Abdullah University of Science and Technology \(KAUST\)](#) while pursuing his Masters. His focus in HPC includes Parallel Programming, Performance Tuning, Containers (Singularity, Docker), Linux system administration, Scientific Software Installation and testing, Scheduler Optimization, and Job Metrics. Shahzeb has held multiple roles in his HPC career in the following companies: Dassault-Systemes, Pfizer, Penn State, and IBM. Prior to 2012, he was a software engineer holding multiple roles at Global Science & Technology, Northrop Grumman, and Penn State.

### Sameer Shende

Dr. Sameer Shende has helped develop the [TAU Performance System](#), the [Program Database Toolkit \(PDT\)](#), the [Extreme-scale Scientific Software Stack \(E4S\)](#) and the HPCLinux distro. His research interests include tools and techniques for performance instrumentation, measurement, analysis, runtime systems, HPC container runtimes, and compiler optimizations. He serves as a Research Associate Professor and the Director of the [Performance Research Laboratory](#) at the [University of Oregon](#), and as the President and Director of [ParaTools, Inc.](#), [ParaTools, SAS](#), and [ParaTools, Ltd.](#) He leads the SDK project for the Exascale Computing Project (ECP), in the Programming Models and Runtime (PMR). He received his B.Tech. in Electrical Engineering from IIT Bombay, and his M.S. and Ph.D. in Computer and Information Science from the University of Oregon.