Long-term plans for Spack

NITRD MAGIC meeting on Software Sustainability
December 1, 2021

Todd Gamblin
Advanced Technology Office
Livermore Computing
What is Spack?

- Supercomputing PACKage manager
- Manages scientific software ecosystem
  - With flexibility needed to build packages for diverse HPC machines
- Language-agnostic
  - Focused originally on build from source
  - Now focused on both source and binary
- Has become a de-facto standard for packaging HPC software

Spack builds for machines like these
(and for your laptop/cloud node/cluster)

Current top systems
- Lawrence Berkeley National Lab
  - AMD Zen / NVIDIA GPU
- ORNL/LLNL
  - Power9 / NVIDIA GPU
- RIKEN
  - Fujitsu/ARM a64fx

Machines coming soon
- Oak Ridge National Lab
  - AMD Zen / MI200 GPU
- Aurora
  - Intel Xeon / Xe
- Fugaku
- Summit & Sierra
- Perlmutter
- Lawrence Livermore National Lab
  - AMD Zen / AMD GPU
  - AMD Zen / NVIDIA GPU
Scientific libraries span C++, C, Fortran, Python, Lua, and more

- **MFEM**: Higher-order finite elements
  - 31 packages, 69 dependencies

- **LBANN**: Neural Nets for HPC
  - 71 packages, 188 dependencies

- **ARES**: LLNL Multi-physics
  - 115 packages, 335 dependencies
What does the Spack project look like?
What does the Spack project look like?

Spack Community
What does the Spack project look like?

Core tool (CLI + Solver)

Spack Community
What does the Spack project look like?

- **Core tool (CLI + Solver)**
- **Package Recipes**

Spack Community
What does the Spack project look like?

- **Spack Community**
- **Infrastructure**
- **Package Recipes**
- **Core tool (CLI + Solver)**
What does the Spack project look like?

External Stacks

- Infrastructure
- Package Recipes
- Core tool (CLI + Solver)

Spack Community
What does the Spack project look like?

- **Core tool (CLI + Solver)**
- **Package Recipes**
- **E4S**
- **Infrastructure**
- **External Stacks**

Spack Community
What does the Spack project look like?

External Stacks

- E4S
- LLNL stack

Infrastructure

Package Recipes

Core tool (CLI + Solver)

Spack Community
What does the Spack project look like?

**Spack Community**

**External Stacks**
- E4S
- LLNL stack
- Vis SDK

**Infrastructure**

**Package Recipes**

**Core tool (CLI + Solver)**
What does the Spack project look like?

- **Core tool (CLI + Solver)**
  - Package Recipes
  - E4S
  - LLNL stack
  - Vis SDK
  - xSDK

**External Stacks**
- Lawrence Livermore National Laboratory
What does the Spack project look like?

Spack Community

External Stacks

- E4S
- LLNL stack
- Vis SDK
- xSDK
- App

Infrastructure

Package Recipes

Core tool (CLI + Solver)
What does the Spack project look like?

Spack Community

Core tool (CLI + Solver)

Package Recipes

Infrastructure

External Stacks

E4S

LLNL stack

Vis SDK

xSDK

App

...
Spack provides a spec syntax to describe customized package configurations

- Each expression is a **spec** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

$ spack install mpileaks unconstrained
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```bash
$ spack install mpileaks  # unconstrained
$ spack install mpileaks@3.3  # custom version
```
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```
$ spack install mpileaks unconstrained
$ spack install mpileaks@3.3 @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3 % custom compiler
```
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```
$ spack install mpileaks unconstrained
$ spack install mpileaks@3.3 @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3 % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads +/- build option
```
Spack provides a spec syntax to describe customized package configurations

- Each expression is a spec for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```
$ spack install mpileaks          unconstrained
$ spack install mpileaks@3.3     @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3 % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 -g3" set compiler flags
```
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```bash
$ spack install mpileaks  unconstrained
$ spack install mpileaks@3.3  @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3  % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads  +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 -g3"  set compiler flags
$ spack install mpileaks@3.3 target=cascadelake  set target microarchitecture
```
Spack provides a *spec* syntax to describe customized package configurations

- Each expression is a *spec* for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

```
$ spack install mpileaks          unconstrained
$ spack install mpileaks@3.3     @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3 % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 -g3" set compiler flags
$ spack install mpileaks@3.3 target=cascadelake set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3 ^ dependency constraints
```
Spack packages are parameterized using the spec syntax
Python DSL defines many ways to build

```python
from spack import *

class Kripke(CMakePackage):
    """Kripke is a simple, scalable, 3D Sn deterministic particle transport mini-app."""

    homepage = 'https://computation.llnl.gov/projects/co-design/kripke'
    url = 'https://computation.llnl.gov/projects/co-design/download/kripke-openmp-1.1.tar.gz'

    version('1.2.3', sha256='3f7f2ee0d1ba5825780d626741eb0b3f026a096048d7ec4794d27dfe6b8ae5')
    version('1.2.2', sha256='eaf9dfff62416974157b34d00c3a1c880f5296f2ce2aa2efa039a86e0976f3a3')
    version('1.1', sha256='232d74072f7b848f9a2ac81bcb39f8eb5f96d50224186601f55554a25f64a')

    variant('mpi', default=True, description='Build with MPI.')
    variant('openmp', default=True, description='Build with OpenMP enabled.')

    depends_on('mpi', when='+mpi')
    depends_on('cmake@3.0:', type='build')

    def cmake_args(self):
        return [
            '-DENABLE_OPENMP=%s' % ('+openmp' in self.spec),
            '-DENABLE_MPI=%s' % ('+mpi' in self.spec),
        ]

    def install(self, spec, prefix):
        mkdirp(prefix.bin)
        install('..//spack-build/kripke', prefix.bin)
```

One package.py file per software project!

- **Base package**
  (CMake support)
- **Metadata**
  at the class level
- **Versions**
- **Variants**
  (build options)
- **Dependencies**
  (same spec syntax)
- **Install logic**
  in instance methods

Don’t typically need install() for CMakePackage, but we can work around codes that don’t have it.
Spack environments enable users to build customized stacks from an abstract description

- spack.yaml describes project requirements
  - Facility stack
  - Application development environment
  - ML framework + simulations built together
  - Etc.

- spack.lock describes exactly what versions/configurations were installed, allows them to be reproduced.
Spack's concretizer leverages ASP solvers to turn abstract constraints into a fully specified, buildable graph.

- new versions
- new dependencies
- new constraints

---

Contributors

pack developers

admins, users

users

Command line constraints

spack install hdf5@1.12.0 +debug

---

Concrete spec is fully constrained and can be built.
Spack’s model lowers the maintenance burden of optimized software stacks

Traditional OS package manager

Recipe per package configuration (need rewrites for new systems)

Build farm

Portable (unoptimized) x86_64 binaries

One software stack upgraded over time
Spack’s model lowers the maintenance burden of optimized software stacks

Traditional OS package manager

Recipe per package configuration (need rewrites for new systems)

Build farm

Portable (unoptimized) x86_64 binaries

One software stack upgraded over time

Spack

Parameterized recipe per package (Same recipe evolves for all targets)

Build farm / CI

Optimized Graviton2 binaries

Optimized Skylake binaries

Optimized GPU binaries

Many software stacks

Built for specific: Systems, Compilers, OS's, MPI's, etc.
Spack’s model lowers the maintenance burden of optimized software stacks

Traditional OS package manager

Recipe per package configuration
(need rewrites for new systems)

Build farm

Portable (unoptimized) x86_64 binaries

One software stack upgraded over time

Spack

Parameterized recipe per package
(Same recipe evolves for all targets)

Build farm / CI

Optimized Graviton2 binaries

Optimized Skylake binaries

Optimized GPU binaries

Many software stacks

Built for specific:
Systems
Compilers
OS’s
MPIs
etc.

Users/developers can also build directly from source
What are the sustainability challenges?

- **Community**
  - Must keep up with incoming pull requests and package updates
  - Identify strong contributors and prioritize their work (e.g., HEP, CSCS, others)

- **Infrastructure**
  - Critical for keeping the package builds working
  - Help from U. Oregon and AWS has been essential

- **Deep technical challenges**
  - Package model + semantics are constantly being improved
  - Deeper modeling of compatibility & ABI
  - Scaling solvers to ever-more-complex ecosystems

- **Maintenance**
  - Keeping core features working and integrating new research
Spack help to sustain the HPC software ecosystem by relying on the efforts of many contributors

6,000+ software packages
960+ contributors

Contributions (lines of code) over time in packages, by organization

Most package contributions are not from DOE
But they help sustain the DOE ecosystem!
Spack help to sustain the HPC software ecosystem by relying on the efforts of many contributors

6,000+ software packages
960+ contributors

Most package contributions are not from DOE
But they help sustain the DOE ecosystem!

Over 4,600 monthly active users of documentation site in October 2021
It takes many maintainers to manage all of the contributions

- **~6 core developers**
  - Core feature development + community management
  - Paid by ASC and ECP

- **Extended core team at Kitware, TechX**
  - Build farm maintenance, CI automation
  - Windows support (temporary push)
  - Subcontracted through LLNL (ASC) and ECP

- **~30 trusted package maintainers on GitHub**
  - Can merge pull requests
  - Picked from the community based on quality of contributions

- **~150 "package maintainers" (so far)**
  - Can't approve merges
  - Are notified of changes to packages of interest
  - Provide reviews on packages to help trusted maintainers

With ECP ending in 2023, ~50% of this funding goes away
Spack’s long-term strategy is based around broad adoption & collaboration

- **Spack is not sustainable without a community**
  - Broad adoption incentivizes contributors
  - Cloud resources and automation critical to scale

- **Continue to prioritize features that get us external buy-in**
  - Niche HPC features aren’t sustainable alone
  - Cloud, containerization, Windows, C++ community features are all aimed at adoption in the broader market

- **Wide adoption in HPC gets us industry attention**
  - Cloud HPC is a growth area
  - Use Spack to bridge between traditional HPC and cloud
  - Work to ensure that good Spack support is an essential feature for vendors to provide to their customers

- **Portability and generality will become increasingly important as cloud environments diversify architectures**
We are already collaborating with key vendors who can help us sustain parts of the software stack

- **AWS** invests significant $$ in cloud credits for Spack build farm
  - Joint Spack tutorial in July with AWS had 125+ participants
  - Joint AWS/AHUG Spack Hackathon drew 60+ participants

- **AMD** has contributed ROCm packages and compiler support
  - 55+ PRs mostly from AMD, also others
  - ROCm, HIP, aocc packages are all in Spack now

- **HPE/Cray** is doing internal CI for Spack packages, in the Cray environment

- **Intel** contributing OneApi support and licenses for our build farm

- **NVIDIA** contributing NVHPC compiler support and other features

- **Fujitsu and RIKEN** have contributed dmany packages for ARM/a64fx support on Fugaku

- **ARM + Linaro** members contributing 100s of PRs for ARM support
Spack is critical for supporting ECP's E4S stack, which we hope will be sustained after ECP

- Spack will be used to build software for the three upcoming U.S. exascale systems
- ECP has built the Extreme Scale Scientific Software Stack (E4S) with Spack – more at https://e4s.io
- Spack will be integral to upcoming ECP testing efforts.

Spack is the most depended-upon project in ECP
We have added security and scaled our public CI so that it can build entire stacks on every pull request

Spack Contributions on GitHub
We have added security and scaled our public CI so that it can build entire stacks on every pull request.
We have added security and scaled our public CI so that it can build entire stacks on every pull request.

Spack Contributions on GitHub

gitlab.spack.io

spack ci

spack.yaml configurations (E4S, SDKs, others)
We have added security and scaled our public CI so that it can build entire stacks on every pull request.

Spack Contributions on GitHub

spack ci

gitlab.spack.io

spack.yaml configurations (E4S, SDKs, others)

- x86_64 and aarch64 pipelines in AWS
- ppc64le, GPU pipelines at U. Oregon
- Pipelines at LLNL (new)
We have added security and scaled our public CI so that it can build entire stacks on every pull request.

- New security model supports untrusted contributions from forks
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

Spack Contributions on GitHub

spack.ci

gitlab.spack.io

spack.yaml configurations (E4S, SDKs, others)

GitLab CI builds (changed) packages
- On every pull request
- On every release branch

- x86_64 and aarch64 pipelines in AWS
- ppc64le, GPU pipelines at U. Oregon
- Pipelines at LLNL (new)
We have added security and scaled our public CI so that it can build entire stacks on every pull request.

- New security model supports untrusted contributions from forks
  - Sandboxed build caches for test builds
  - Authoritative builds on mainline only after approved merge

Spack Contributions on GitHub

- gitlab.spack.io

spack ci

- spack.yaml configurations (E4S, SDKs, others)
- GitLab CI builds (changed) packages
  - On every pull request
  - On every release branch

- x86_64 and aarch64 pipelines in AWS
- ppc64le, GPU pipelines at U. Oregon
- Pipelines at LLNL (new)
BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can’t generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable automatic and ABI-compatible reuse of system binaries, foreign binary packages

- WIP: add ABI constraints to the solver
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors
BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can’t generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable automatic and ABI-compatible reuse of system binaries, foreign binary packages

- WIP: add ABI constraints to the solver
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    • Link correctly
    • Run w/o symbol and certain type errors
BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can’t generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable *automatic* and ABI-compatible reuse of system binaries, foreign binary packages

- **WIP: add ABI constraints to the solver**
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    - Link correctly
    - Run w/o symbol and certain type errors
BUILD is a 3-year strategic initiative, aimed at reducing human maintenance burden

- Basic premise: humans can’t generate all the compatibility constraints
  - Version ranges, conflicts, in Spack packages not precise
  - Rely on maintainers to get right.

- BUILD aims to understand software compatibility at the binary level
  - Develop ABI compatibility models
  - Enable automatic and ABI-compatible reuse of system binaries, foreign binary packages

- WIP: add ABI constraints to the solver
  - Don't just check with coarse compiler/target/version info
  - Guarantee that the executable will:
    • Link correctly
    • Run w/o symbol and certain type errors

![Diagram showing compatibility models and solver]
Key Spack priorities for future sustainability

1. Preserve Spack core team and feature development after ECP
   - Seek out sustainability funding
   - Ensure we can manage the growing community
   - Look at alternate governance models (foundations?)
     • Would this make it easier to scale?

2. Increase build automation to match rate of contribution
   - More CI, more binaries, more platforms

3. Generalize Spack's model to make the software stack as portable as possible
   - ABI research on BUILD
   - Compiler dependency model
   - Better modeling of runtime libraries for GPUs, OpenMP
   - Improved solver constraints

4. Continue to grow collaborator base with key features
   - Windows support
   - More developer features
   - Continuous integration
   - Public binary cache for faster installations
Spack DSL allows **declarative** specification of complex constraints

CudaPackage: superclass for packages that use CUDA

```python
class CudaPackage(PackageBase):
    variant('cuda', default=False,
            description='Build with CUDA')

    with when('+cuda'):
        variant('cuda_arch',
                description='CUDA architecture',
                values=any_combination_of(*cuda_arch_values),
                when='+cuda')

        depends_on('cuda', when='+cuda')

    depends_on('cuda@9.0:', when='cuda_arch=70')
    depends_on('cuda@9.0:', when='cuda_arch=72')
    depends_on('cuda@10.0:', when='cuda_arch=75')

    conflicts('%gcc@9:', when='+cuda ^cuda@:10.2.89 target=x86_64:')
    conflicts('%gcc@9:', when='+cuda ^cuda@:10.1.243 target=ppc64le:')
```

cuda is a variant (build option)
+cuda = cuda is on
~cuda = cuda is off

cuda_arch and dependency on cuda are only present if cuda is enabled

Map compute capability to cuda version

Compiler support determined by architecture and CUDA version

DSL is designed to model software in **all** configurations (not just one)
In Spack, *concretization* converts an abstract spec to a real (concrete) installation

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints
In Spack, *concretization* converts an abstract spec to a real (concrete) installation.

User input: *abstract* spec with some constraints
In Spack, **concretization** converts an abstract spec to a real (concrete) installation

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints

Abstract, normalized spec with dependencies known *a priori.*
In Spack, *concretization* converts an abstract spec to a real (concrete) installation.

User input: *abstract* spec with some constraints

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

*Abstract*, normalized spec with dependencies known *a priori*.
In Spack, *concretization* converts an abstract spec to a real (concrete) installation.

**User input:** *abstract* spec with some constraints

**Concrete spec is fully constrained and can be passed to install.**
In Spack, *concretization* converts an abstract spec to a real (concrete) installation.

**Abstract** spec, normalized spec with dependencies known *a priori.*

**Concrete** spec is fully constrained and can be passed to install.

User input: *abstract* spec with some constraints.
In Spack, *concretization* converts an abstract spec to a real (concrete) installation.

**User input:** abstract spec with some constraints

**spec.yaml**

```yaml
spec:
- mpileaks:
  arch: linux-x86_64
  compiler:
  name: gcc
  version: 4.9.2
  dependencies:
  - adept-utils
    arch: linux-x86_64
    compiler:
    name: gcc
    version: 4.9.2
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0
  - boost:
    arch: linux-x86_64
    compiler:
    name: gcc
    version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
  - libdwarf:
    arch: linux-x86_64
    compiler:
    name: gcc
    version: 4.9.2
    dependencies: {}
    hash: kartkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 0.10.4
```

**Abstract**, normalized spec with dependencies known *a priori*.

**Concrete** spec is fully constrained and can be passed to install.

Detailed provenance is stored with the installed package.
Package solving is **combinatorial search with constraints and optimization**

- Search over a solution space:
  - Possible dependency graphs (nodes, edges)
  - Assignment of node and edge attributes
    - Version
    - Dependency, dependency type
    - Compiler, compiler version
  - Target
  - Compiler, compiler version

- Subject to validity constraints:
  - Version requirements
  - Target/compiler compatibility
  - Virtual providers

- Optimization picks “best” among valid solutions:
  - Most recent versions
  - Preferred variant values
  - Preferred compilers that support best targets (e.g., AVX-512)
  - Minimize number of builds

This problem is NP-hard!
Most language runtimes only support one package version in memory at a time
   - Must pick exactly one version of each package in the graph

Impossible to choose a version of D that satisfies both B and C
   - Must back out and choose new B or C versions
   - Repeat until we find ones with compatible constraints on D
"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov