Spack Driven Software Development and Spack-Manager

Philip Sakievich (SNL)
psakiev@sandia.gov

10/27/2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.
Overview

• Spack Overview
• Introduction to Spack Develop
• Overview of Spack Develop API
• Making it simpler with Spack-Manager

ECP: Funding Statement
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative.

Acknowledgements: Jon Rood, Timothy Smith, Luke Peyralans, Spack Dev Team, Spack community

This presentation will just be a simple overview to highlight capabilities.
Spack: Package Manager++

- Package manager focused on HPC applications
- Spack has many attractive features:
  - Complex package and environment configurations
  - Embedded tribal HPC knowledge
  - A unique, scalable, multicomponent development tool (spack develop)
- Spec:
  - trilinos@develop+fortran build_type=Release %gcc@10.3.0
- Environment:
  - Constrain what software is available and gets built (pyenv, conda, etc)

```bash
# This is a Spack Environment file.
# It describes a set of packages to be installed, along with configuration settings.
spack:
  specs:
  - nalu-wind
  - trilinos@develop
view: false
develop:
  nalu-wind:
    spec: nalu-wind@master
  trilinos:
    spec: trilinos@develop
```

---

Exascale Computing Project
ExaWind: A Motivating Application

• ExaWind software stack:
  – Combine two loosely coupled CFD codes with entirely different software stacks (Trilinos and AMReX)
  – Living on the develop branch of multiple dependencies
  – Project is actively supporting development of 7+ software packages in the stack (CPU+GPU)

• Challenges:
  – Building
  – Developing
  – Testing
  – Deploying

Packages under active development
Spack Develop

- In a spack environment *develop specs* can be added
- Develop specs are
  - If DAG_spec.satisfies(develop_spec)
    - Do a build from the users source code rather than from spack’s staging procedure
    - Perform incremental builds based on timestamp of files in the source directory
- Allows for arbitrary development of packages in the DAG
  - Dependencies will get automatically rebuilt
- Allows for multiple builds from the same source
  - Cuda and Non-Cuda builds from the same source code at the same time
  - DAG level parallelism is available in builds

```bash
# In this configuration you will get
# 4 develop builds: cuda and non-cuda
# nalu-wind and trilinos coming from
# the same sources
spack:
  specs:
  - nalu-wind +cuda cuda_arch=70
  - nalu-wind ~cuda
  view: false
  develop:
    nalu-wind:
      spec: nalu-wind@master
    trilinos:
      spec: trilinos@develop
```
Development Environment API

• Utilize develop feature
  – Create environment
  – Tag the specs/packages you wish to develop
  – Make sure the source code is correct (several ways to do this)
  – Install

• To develop
  – Make code changes
  – Spack install (incremental build)

Setup Environment

- spack env create foo
- spack env activate foo
- spack add do re mi

Development Commands

- spack develop do@develop
- spack develop re@main
- spack develop mi@main

Final Touches

- spack cd --environment
- cd re
- git remote add user git@github.com:user/feature
- git fetch --all && git checkout feature
- spack install
Spack-Manager: API Reduction

- Spack-Manager:
  - Embed machine specific natively
  - Reduce the API for using spack develop

- Utilize Spack API’s to write Spack extensions
  - Environment curation
  - All of our scripts serve to reduce the end user API
  - Can be replicated through core Spack commands and a little manual intervention

- A core example of this is:
  - find-machine + create-env
    - find-machine: a utility that allows custom python scripts to identify the current machine
    - create-env: uses find-machine and stored configs to automate platform specific environments

\[
\text{spack manager create-env} \ -d \ [foo] \ -s \ [specs]
\]
What does it look like?

spack manager create-env --spec exawind amr-wind nalu-wind
Onboarding Developers

• Conflict: 1 command build vs a learning curve
  – Made significant efforts to reduce the API

• Ask developers to learn 3 things about Spack:
  – How to query the API for help i.e. --help and spack info
  – How to read and write a Spack spec
  – What the major steps in the Spack build process are

• Learn to speak the basics of the language

I […] was able to install Exawind using Spack fairly easily as a new hire. I have definitely had a good experience so far
- Ilker Topcuoglu (NREL)

I have to type a whole 12 characters to compile just 2 different codes with a zillion dependencies to debug my code
- Ganesh Vijayakumar (NREL)

Spack Manager and Spack have saved me an incredible amount of time and headache, providing an intuitive framework that ensures dependency resolution and repeatable, shareable, self-documenting builds.
- Nate deVelder (SNL)
Pros and Cons of Spack Driven Development

Pros

- Spack is already solving the dependency issues
- Spack is scalable
  - DAG parallelism
    - HPC Case study: 3 compiler configurations for ExaWind
      - 1.5 hours with DAG parallelism
      - 4.5+ hours without
- Spack is configurable
  - +cuda and ~cuda in same environment (DAG parallel)
- Spack is extendable
- Spack is testable
- Simplified and unified API dramatically reduces Dev-Ops workload

Cons

- Spack can be overwhelming
  - 3-5 ways to do just about everything
- Spack build process has some quirks
  - Hash based issues and confusion
  - Bootstrapping and occasional ssl issues
- Spack data management and logs make developers uncomfortable
  - spack-build-[hash]
  - spack cd -b [package]
- Spack still has some optimization to do
  - spack install is a too big of a hammer for incremental builds
Conclusions

• Spack is taking on a lot of challenges in the HPC software space
  – Not everything is perfect, but the progress is rapid
  – We can help make it better!

• Very happy with Spack as the driver for development on ExaWind
  – Unified API dramatically reduces infrastructure needs
  – Gives developers the tools to customize their own environments

• Cons can be mitigated with education and light scripts

• Spack-Manager is a tool for managing and reducing the Spack API with a particular emphasis on development
  – We’d love to have more Trilinos developers test it out
The Vision: Unified Tooling and Environments

Admin Workflow

• Common environment for administrators and developers leads to reuse and consistency
  – I’m building exactly what is on my dashboard

• Common deployment tools means common interface for analysts

• A machine agnostic interface makes this highly deployable

End User Environment

- module use [/path/to]/spack-manager/modules
- module load xyz

Nightly Tests/CDash

Docker Image/Snapshot

Module Creation

Github CI/CD

Developer Workflow

Development Environment

Module Creation
Spack-Manager Layout

• Spack-Manager
  – Project agnostics code/scripts
    • Tooling and testing
  – Pre-configured locations
  – Project specific information
    • Customize packages
    • Create machine specific implementations
    • Add machine specific templates
Bash ”quick-commands”

- Wrap the functionality of basic setup and development commands together

- Common features:
  - Shell source Spack/Spack-Manager
  - Create an anonymous Spack environment
  - Activate the created environment

- Development specific assumptions:
  - All concrete spec's are intended as develop specs ([name]@[version])
  - Anything not pre-cloned should be fetched via spack develop

<table>
<thead>
<tr>
<th>Step</th>
<th>quick-create</th>
<th>quick-create-dev</th>
<th>quick-develop</th>
</tr>
</thead>
<tbody>
<tr>
<td>spack-start</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Create an environment</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Activate an environment</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Add root specs</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Add develop specs</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Add externals</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Concretize and install</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• quick-create-dev --spec do@develop re@main mi@main
Externals: Re-Using Binaries

- Spack has several different ways to reuse binaries
  - Upstreams
  - Binary Caches
  - --reuse
  - Externals

- First 3 rely directly on the concertizer to make the “best” decision

- Development workflow often wants specific binaries

- Created a way to auto generate externals in an externals.yaml file

- “Snapshots” are time-dated versions of the software installed on each system
Containers

- Partnered with E4S to create nightly containers
- Software provenance preserved through history of containers on Docker Hub
- Infrastructure makes containerization trivial
  - E4S added 4 lines to their base Ubuntu docker configuration
- With externals + container we can drive our CI for every package through 1 image
- Developers can download image and have same environment on laptops