

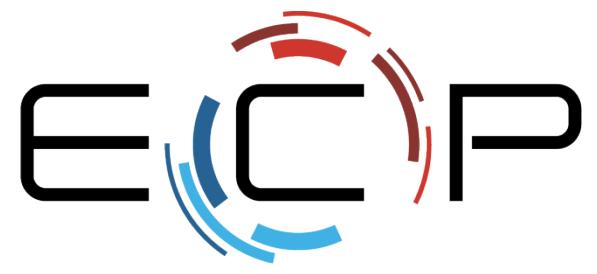
Spack Driven Software Development and Spack-Manager

Philip Sakievich (SNL)

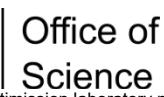
psakiev@sandia.gov

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.

10/27/2022



EXASCALE COMPUTING PROJECT



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.

SAND2022-XXXXX

Overview

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

This presentation will just be a simple overview to highlight capabilities.

ECP: Funding Statement

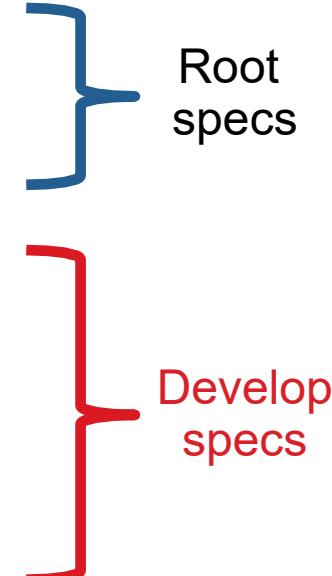
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including 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

Spack: Package Manager++

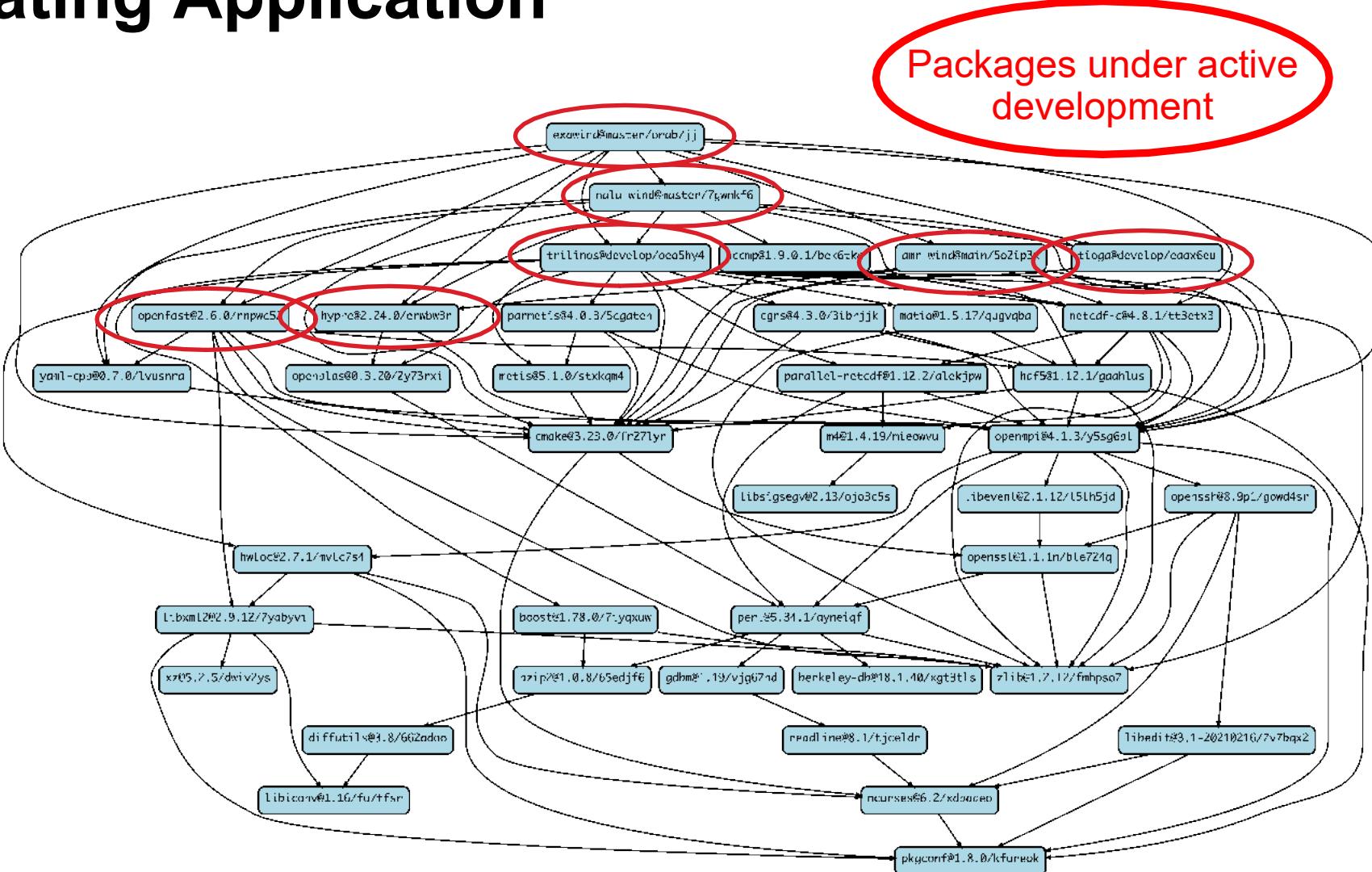
- Package manager focused on HPC applications
- Spack has many attractive features:
 - Complex package and environment configurations
 - Embedded tribal HPC knowledge
 - **A unique, scalable, multicomponent development tool (spack develop)**
- Spec:
 - `trilinos@develop+fortran build_type=Release %gcc@10.3.0`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

```
# 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
```



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



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

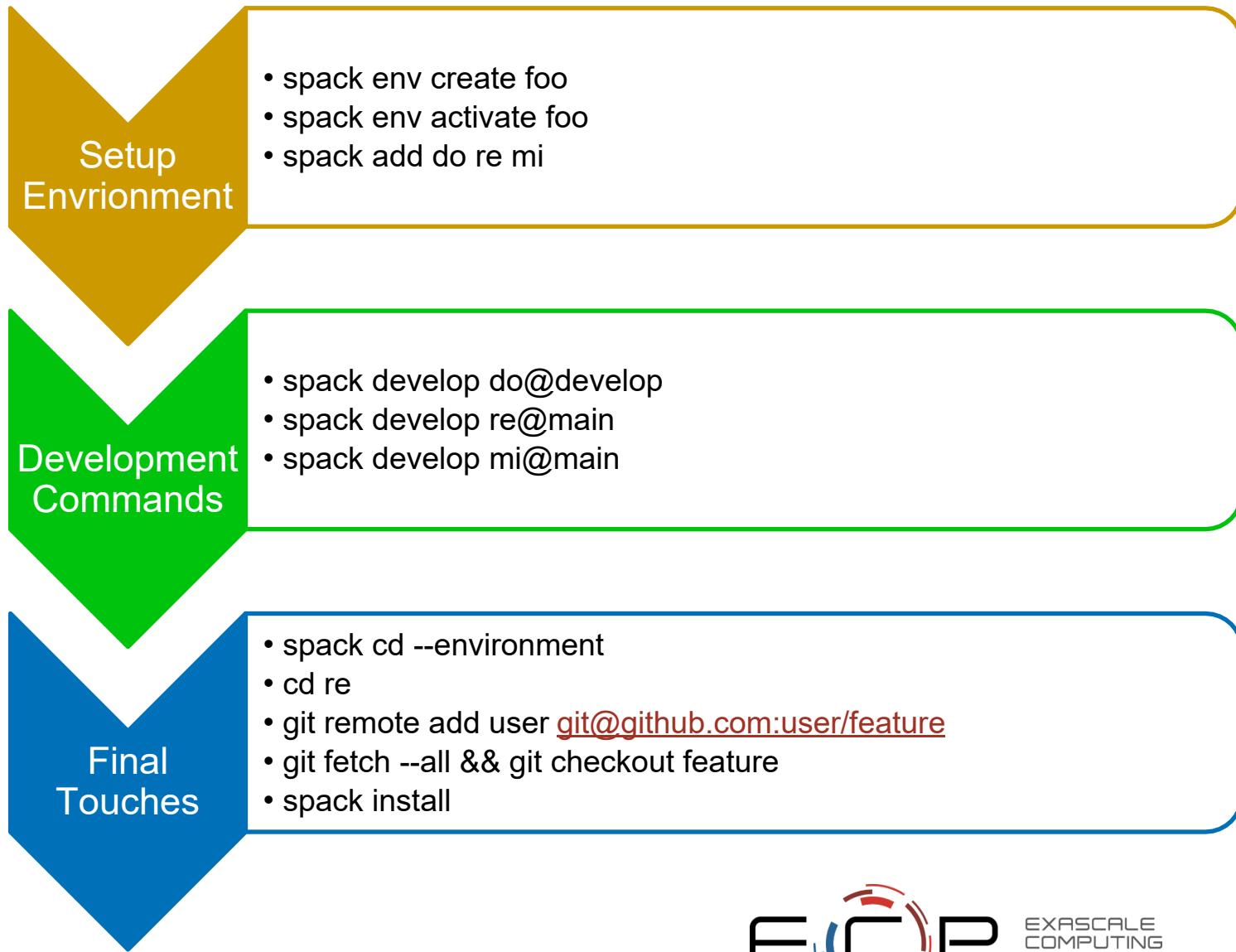
```
# In this configuration you will get
# 4 develop builds: cuda and non-cuda
# nalu-wind and trilinos coming from
# the same sources

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

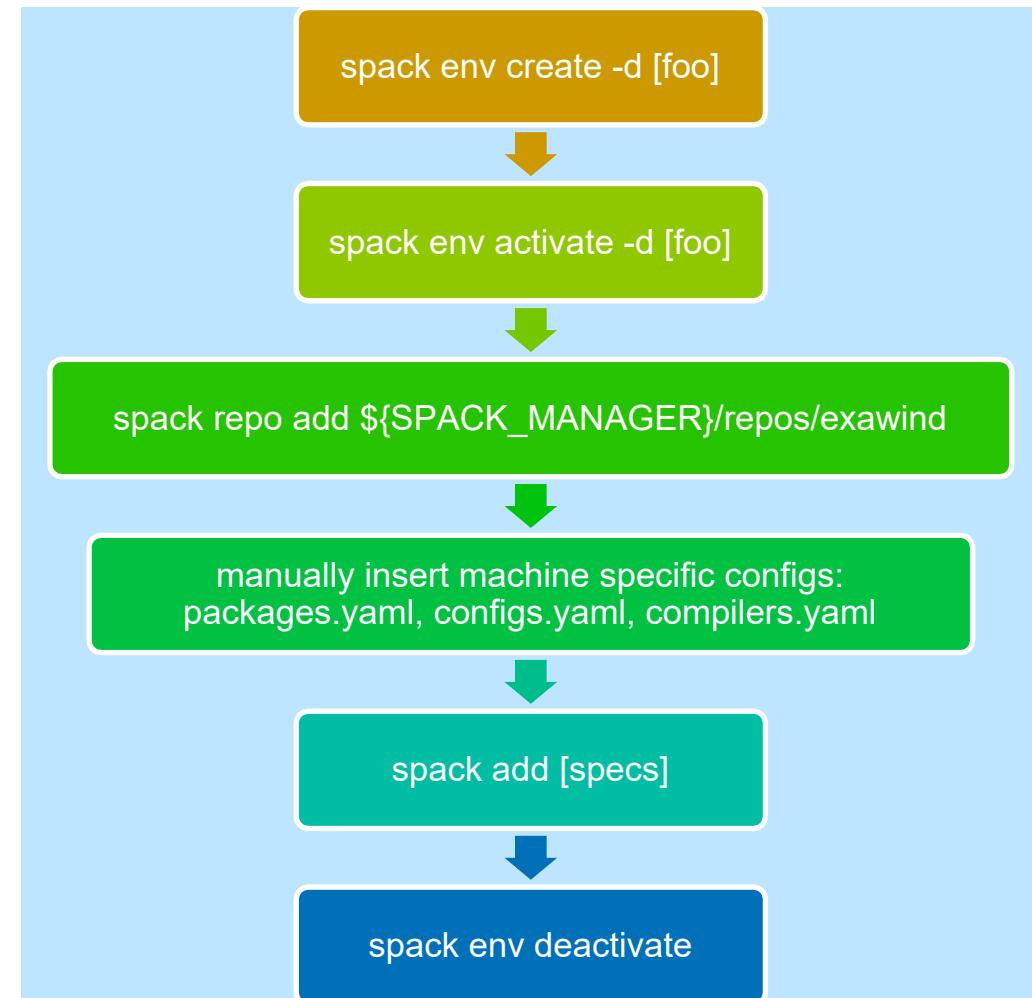
- Utilize develop feature
 - Create environment
 - Tag the specs/packages you wish to develop
 - Make sure the source code is correct (several ways to do this)
 - Install
- To develop
 - Make code changes
 - Spack install (incremental build)



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

```
spack manager create-env -d [foo] -s [specs]
```



What does it look like?

`spack manager create-env --spec exawind amr-wind nalu-wind`

```
1 spack.yaml
1 spack:
1   include: [include.yaml]
2   concretization: together
3   view: false
4   specs: [exawind, amr-wind, nalu-wind]
```

```
1 include.yaml
1 repos:
1   - $spack/..repos/exawind
2 packages:
3   hypre:
4     variants: +shared
5     version: [develop]
6   all:
7     target: [x86_64]
8     compiler: [apple-clang, gcc, clang]
9     providers:
10    mpi: [mpich, openmpi]
11    blas: [netlib-lapack]
12    lapack: [netlib-lapack]
13    variants: build_type=Release +mpi
14   boost:
15     version: [1.76.0]
16     variants: cxxstd=14
17   hdf5:
18     version: [1.10.7]
19     variants: +cxx+hl
20   netcdf-c:
21     version: [4.7.4]
22     variants: +parallel-netcdf maxdims=65536 maxvars=524288
23   openfast:
24     version: [master]
25     variants: +cxx
26   parallel-netcdf:
27     version: [1.12.2]
28   tioga:
29     version: [develop]
30   yaml-cpp:
31     version: [0.6.3]
32   trilinos:
33     version: [develop]
34     variants: -adios2-allocptkgs-amesos+amesos2-anasazi-aztec+belos+boost+chaco-complex-debug-dtk-epetra+epetraext+exodus+explicit_template_instantiation-float+fortran+fortrilinos+glm+gtest+ hdf5+hypre+ifpack+ifpack2+intrepid+intrepid2+isorropia+kokkos+mesquite+metis+minitensor+ml+mpi+muellu+mumps+nox+openmp+phalanx+piro+python+rol+rythmos+socado+shards+shylu+stk+stratimikos+suite+sparse+superlu+superlu-dist+teko+tempus+teuchos+tpetra+uvm+x11+xsdkflags+zlib+zoltan+zoltan2
35     gotype=long cxxstd=14 build_type=Release
36 config:
37   mirrors:
38     e4s: https://cache.e4s.io
39   source_cache: ~/.spack/downloads
40   misc_cache: $spack/..cache
41   build_stage:
42     - $spack/..stage
43   concretizer: cling
```

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

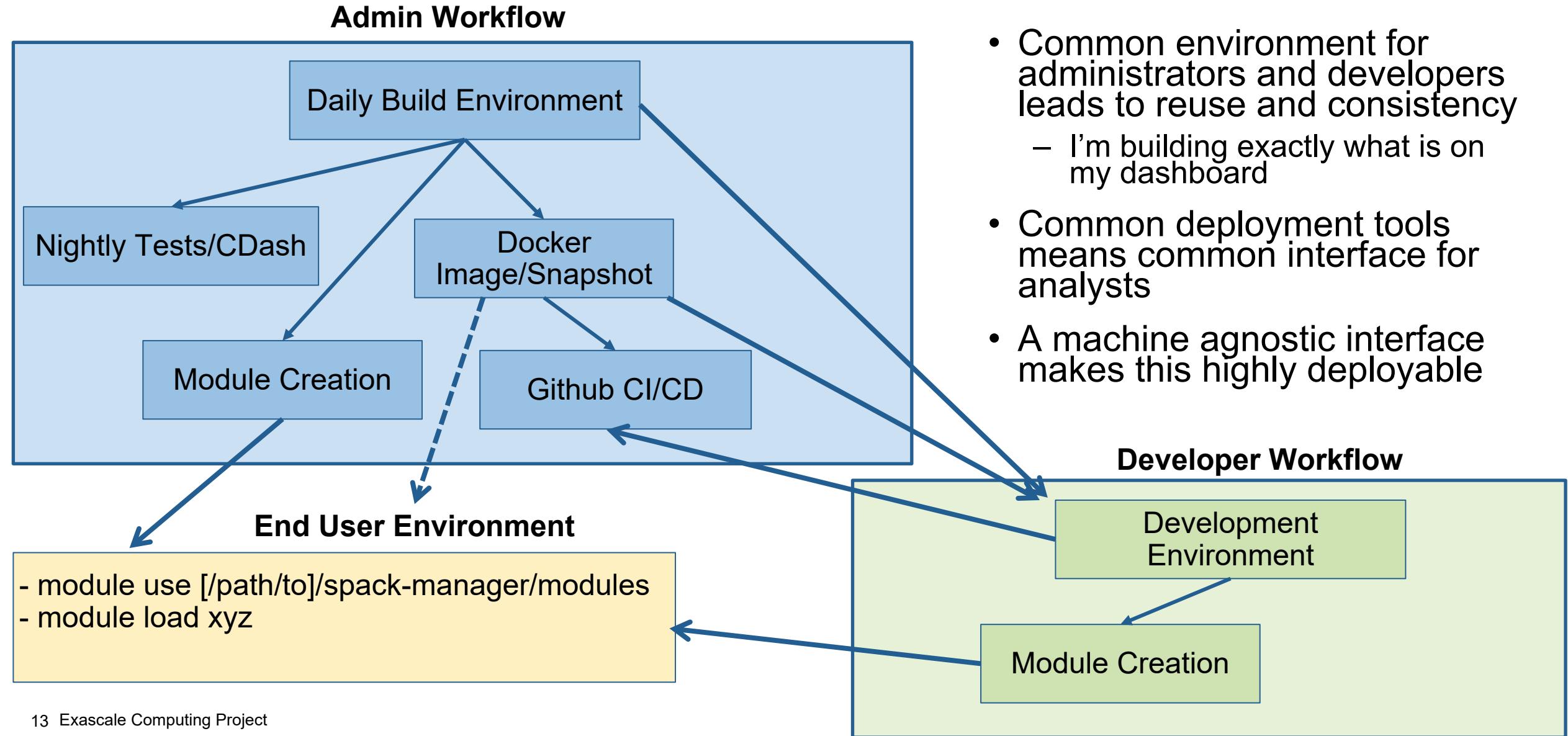
Conclusions

- 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

Supplementary Slides

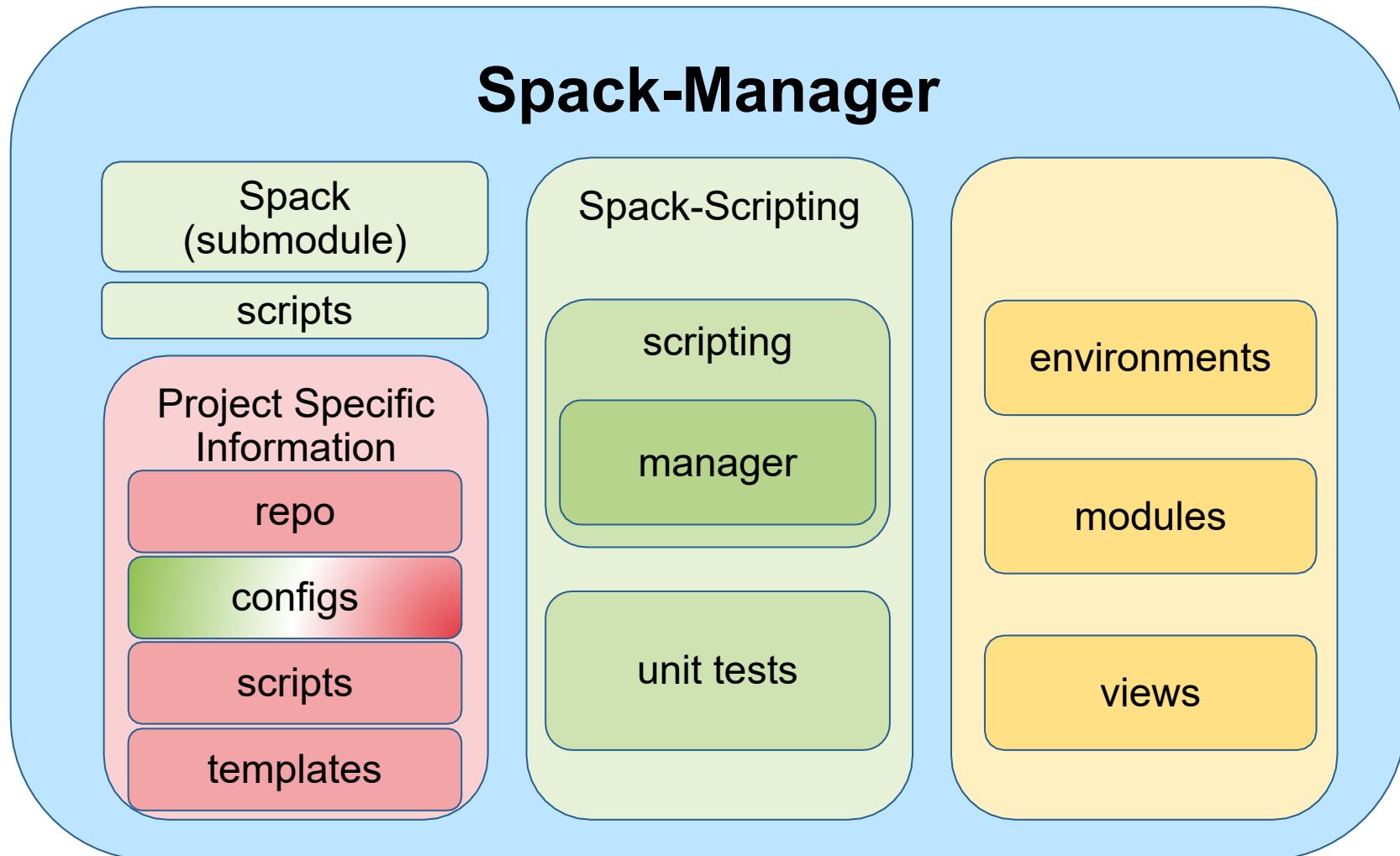


The Vision: Unified Tooling and Environments



Spack-Manager Layout

- Spack-Manager
 - Project agnostic code/scripts
 - Tooling and testing
 - Pre-configured locations
 - Project specific information
 - Customize packages
 - Create machine specific implementations
 - Add machine specific templates



Bash "quick-commands"

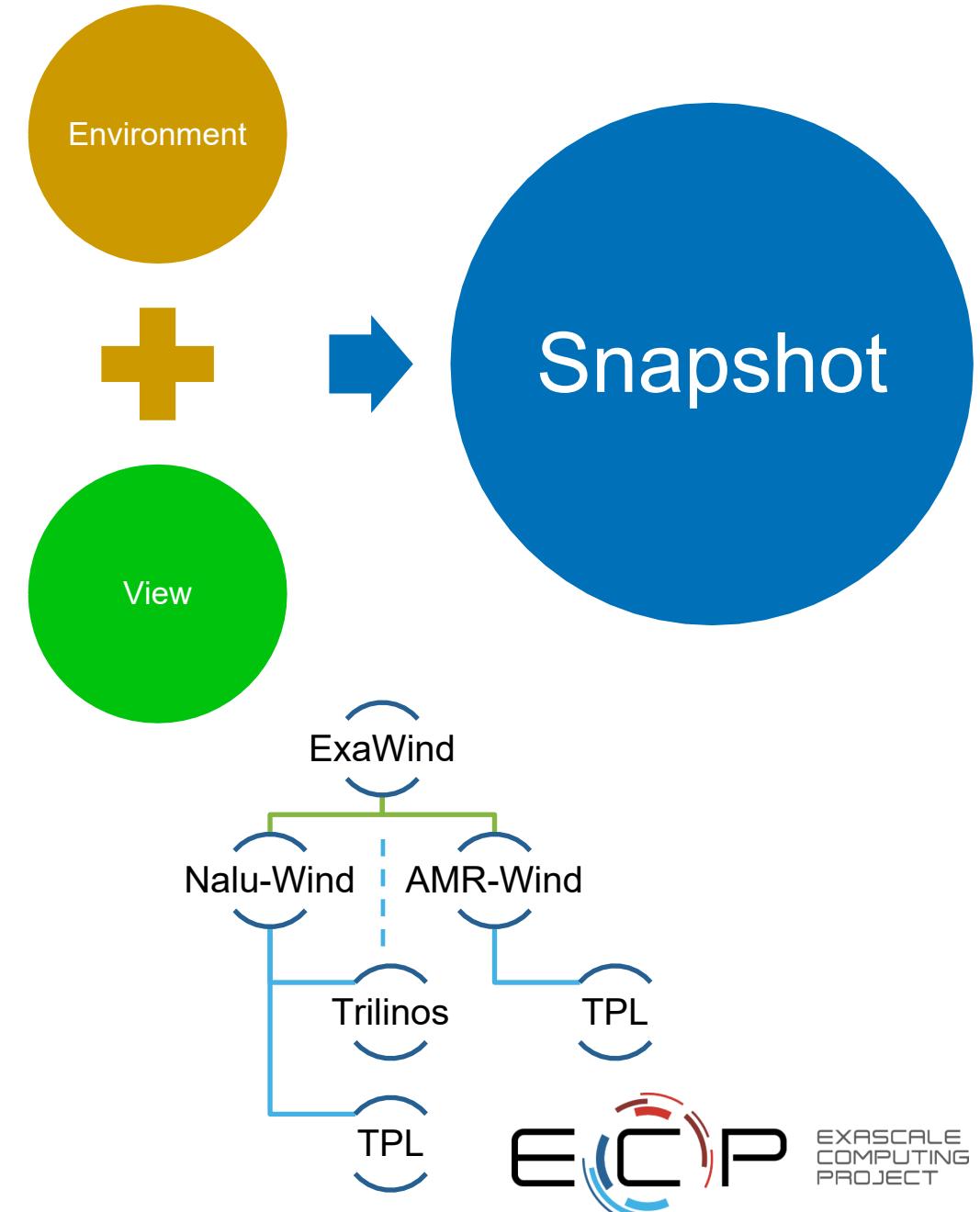
- Wrap the functionality of basic setup and development commands together
- Common features:
 - Shell source Spack/Spack-Manager
 - Create an anonymous Spack environment
 - Activate the created environment
- Development specific assumptions:
 - All concrete spec's are intended as develop specs ([name]@[version])
 - Anything not pre-cloned should be fetched via spack develop

Step	quick-create	quick-create-dev	quick-develop
spack-start	x	x	x
Create an environment	x	x	x
Activate an environment	x	x	x
Add root specs	x	x	x
Add develop specs		x	x
Add externals			x
Concretize and install			

• `quick-create-dev --spec do@develop re@main mi@main`

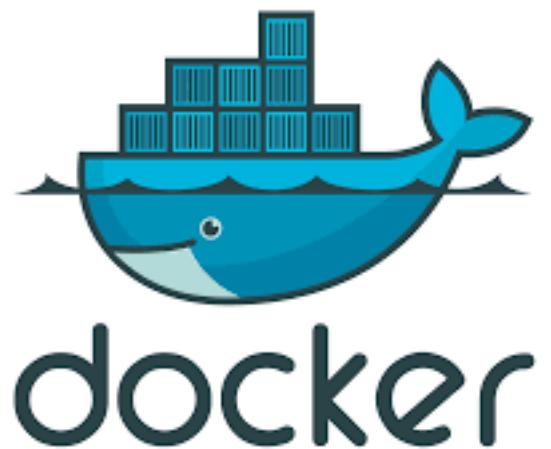
Externals: Re-Using Binaries

- 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



GitHub Actions

```
20  CPU:
21    #needs: Formatting
22    runs-on: ubuntu-latest
23    container:
24      image: ecp4s/exawind-snapshot
25    env:
26      SPACK_MANAGER: /spack-manager
27      E4S_MACHINE: true
28    steps:
29      - name: Cancel previous runs
30        uses: style/cancel-workflow-action@0.6.0
31        with:
32          access_token: ${{github.token}}
33      - name: Clone
34        uses: actions/checkout@v3
35        with:
36          submodules: true
37      - name: Tests
38        working-directory: /spack-manager/environments/exawind
39        run: |
40          /bin/bash -c " \
41            source ${SPACK_MANAGER}/start.sh && \
42            ln -s ${GITHUB_WORKSPACE} nalu-wind && \
43            source ${SPACK_MANAGER}/start.sh && \
44            quick-develop -s nalu-wind@master && \
45            spack install && \
46            spack cd -b nalu-wind && \
47            spack build-env nalu-wind ctest -j $(nproc) -L unit --output-on-failure \
48          "
```