SAND2022-6989C

Spack-Manager: A case study for managing complex software development workflows with Spack

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.

5/25/2022

SAND NUMBER PLACE HOLDER

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

- ExaWind software
- Philosophies that led to Spack-Manager
- Organization of Spack-Manager
- Tools for software development
- Getting developers on board
- Containers
- Conclusions

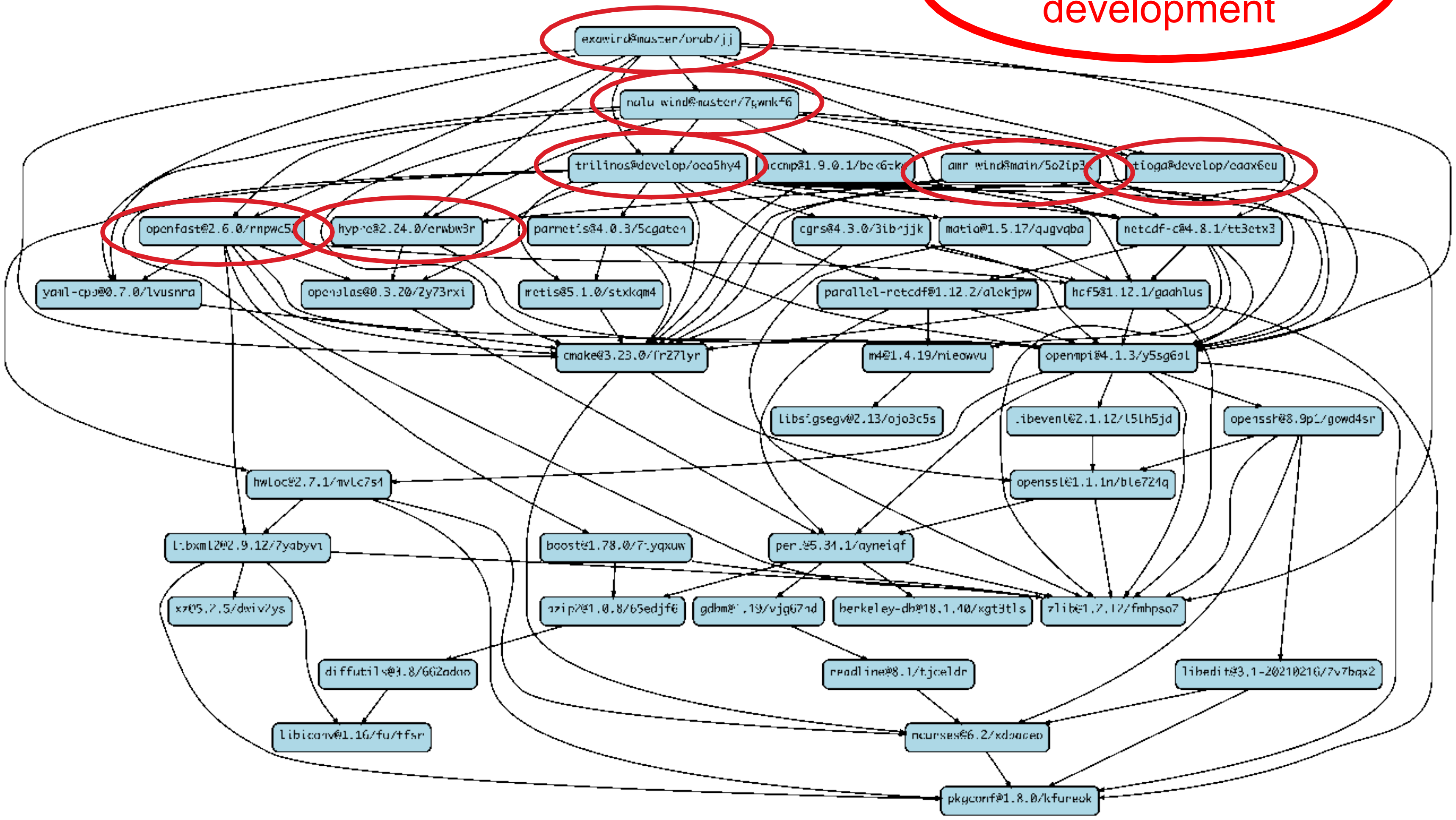
ECP: Funding Statement

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

ExaWind: The Motivating Application

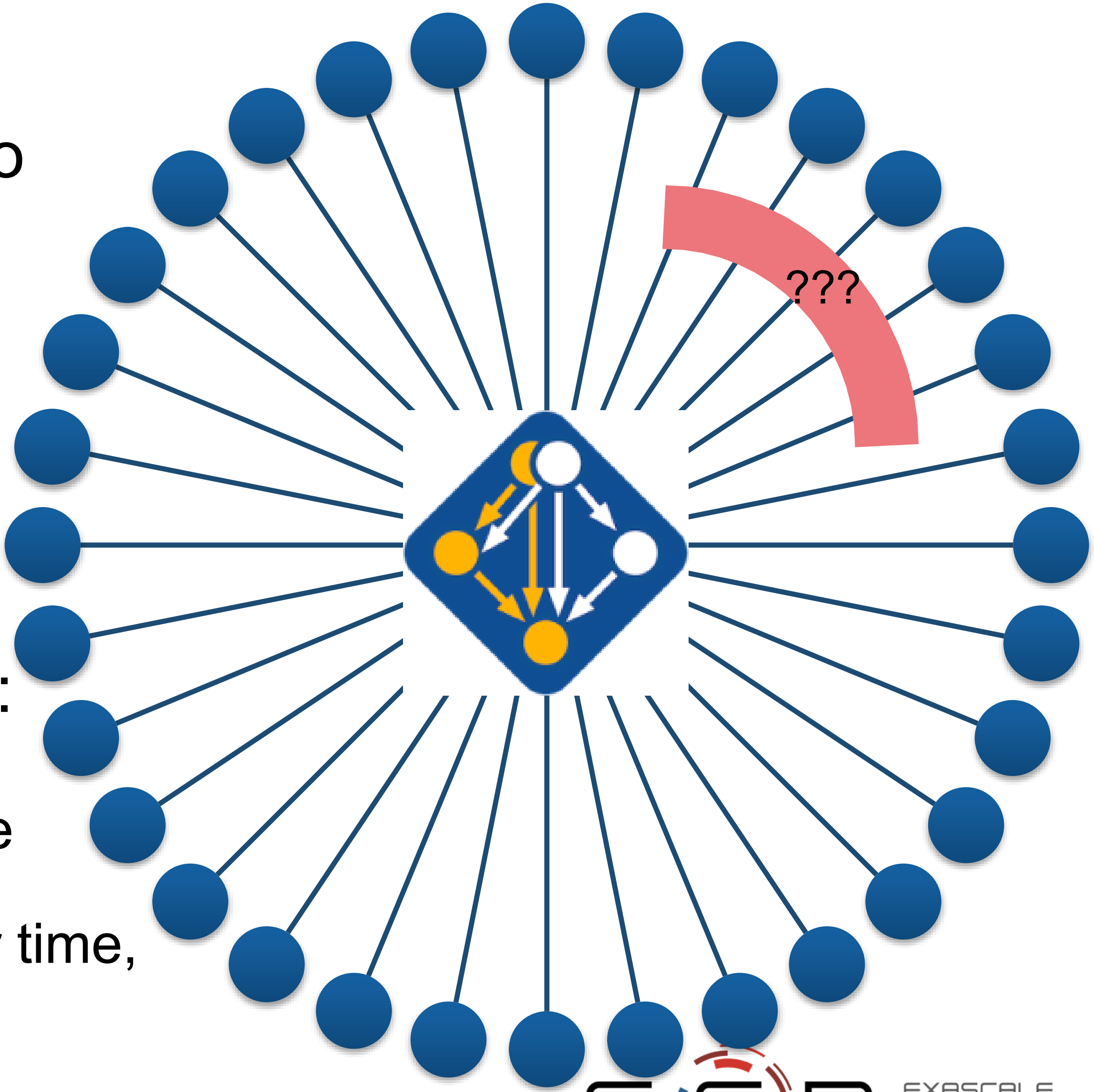
- ExaWind software stack:
    - Combine two loosely coupled CFD codes with entirely different software stacks (Trilinos and AMReX)
    - Living on the develop branch of multiple dependencies
    - Project is actively supporting development of 7+ software packages in the stack (CPU+GPU)
- Challenges: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



Packages under active development

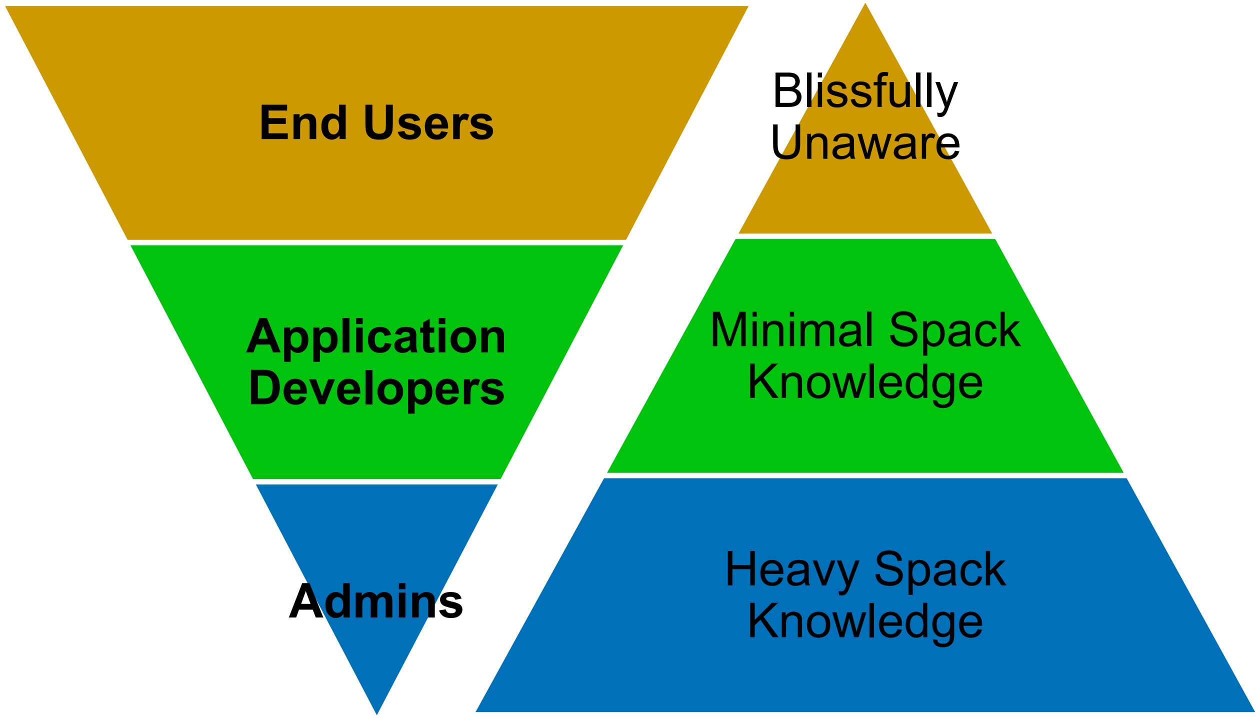
Spack: Package Manager++

- Managing these dependencies leads to Spack
- Spack has many attractive features:
  - Complex package and environment configurations
  - Embedded tribal HPC knowledge
  - A unique, scalable, multicomponent development tool (spack develop)
- Using Spack has some challenges too:
  - A large project with a lot of moving parts
  - Things happen quickly and slowly all at the same time
  - Sensitivity to changes has decreased over time, but is still non trivial



Spack-Manager Philosophies

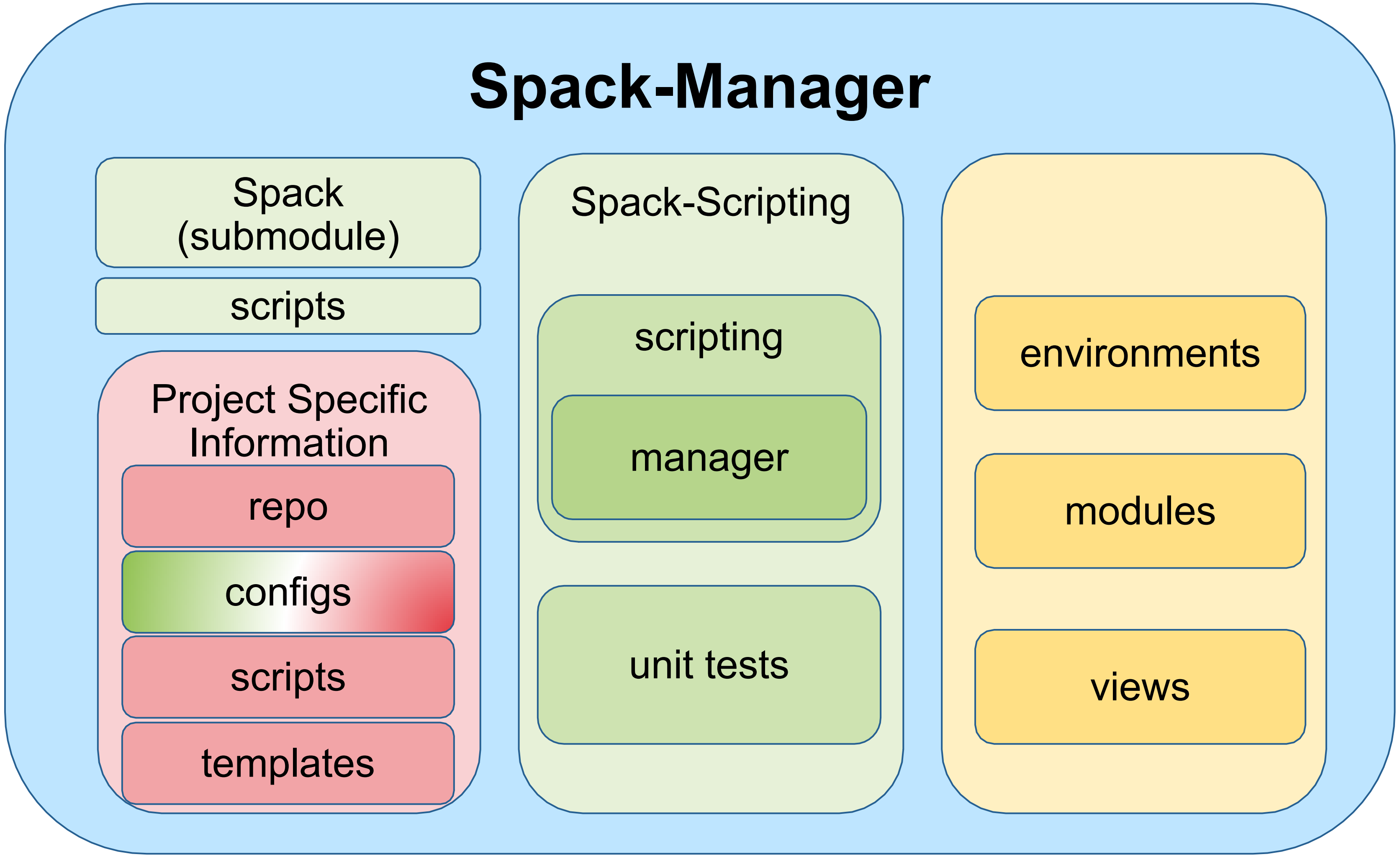
- Spack-Manager is an **extension** to Spack that aims to act as a buffer between Spack and our end application
  - Increases our agility
  - Framework to prototype new Spack features
  - Manage machine specific configurations and create a machine agnostic interface
- Spack-Manager also seeks to unify a workflow that serves 3 distinct user profiles
  - Administrators
  - Application developers
  - End users/analysts



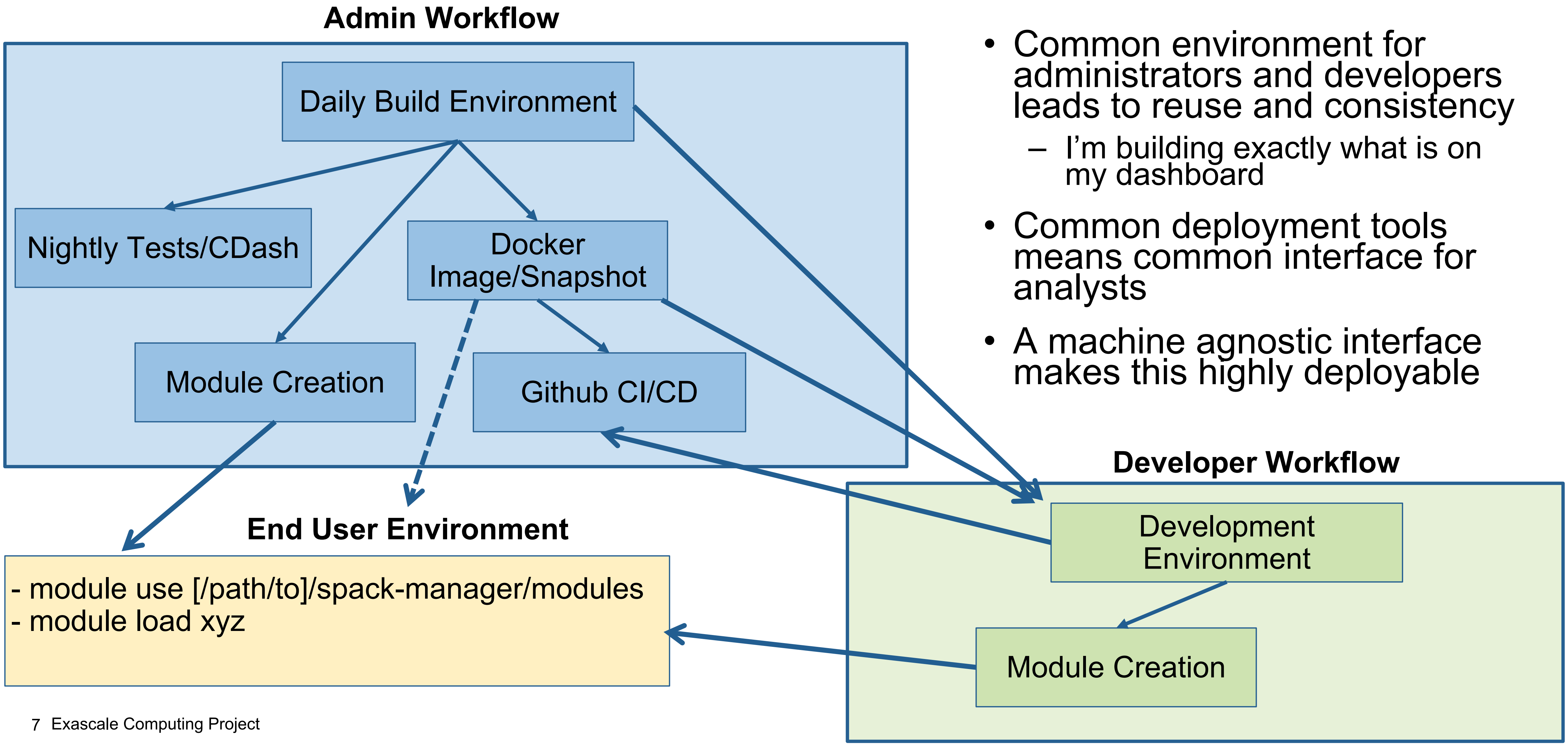
Population size of the user profile has an inverse relationship with required understanding of Spack

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



The Vision: Unified Tooling and Environments



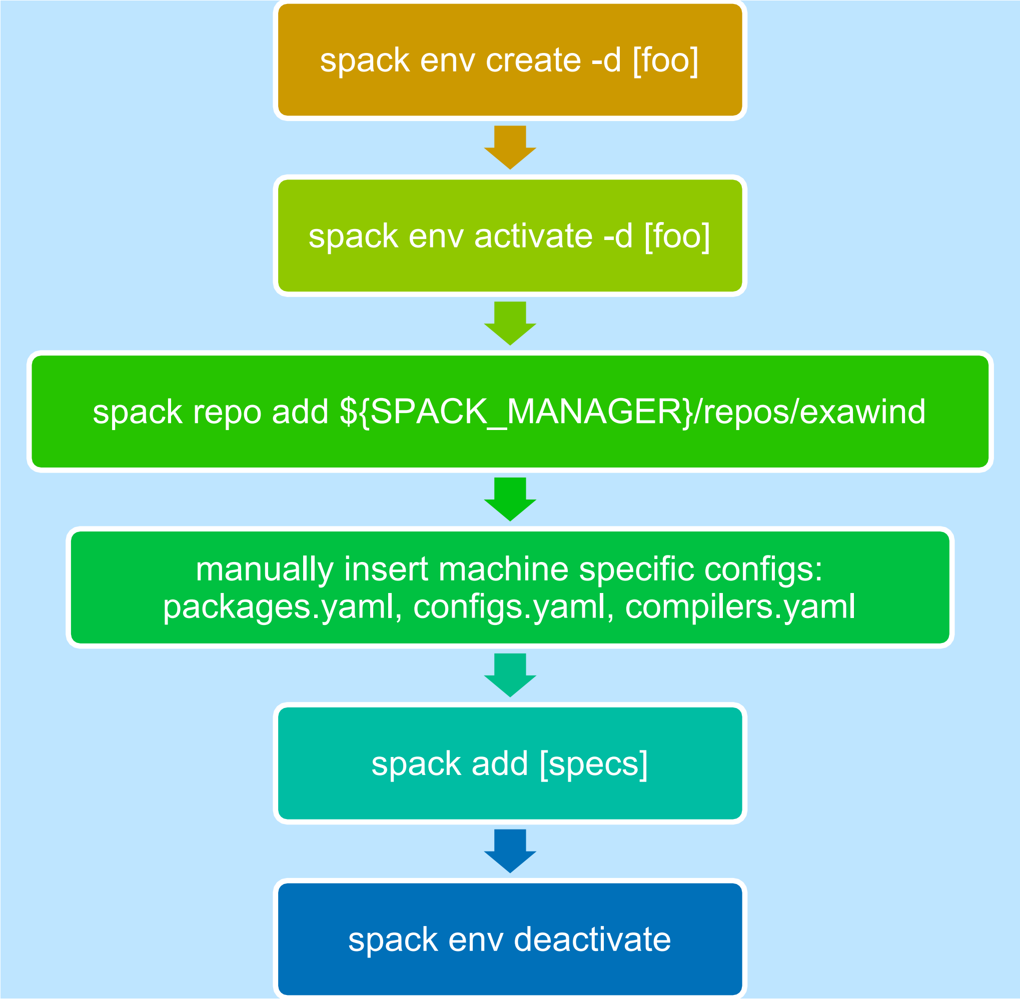
```
- module use [/path/to]/spack-manager/modules
- module load xyz
```

- 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

How do we do this?

- 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

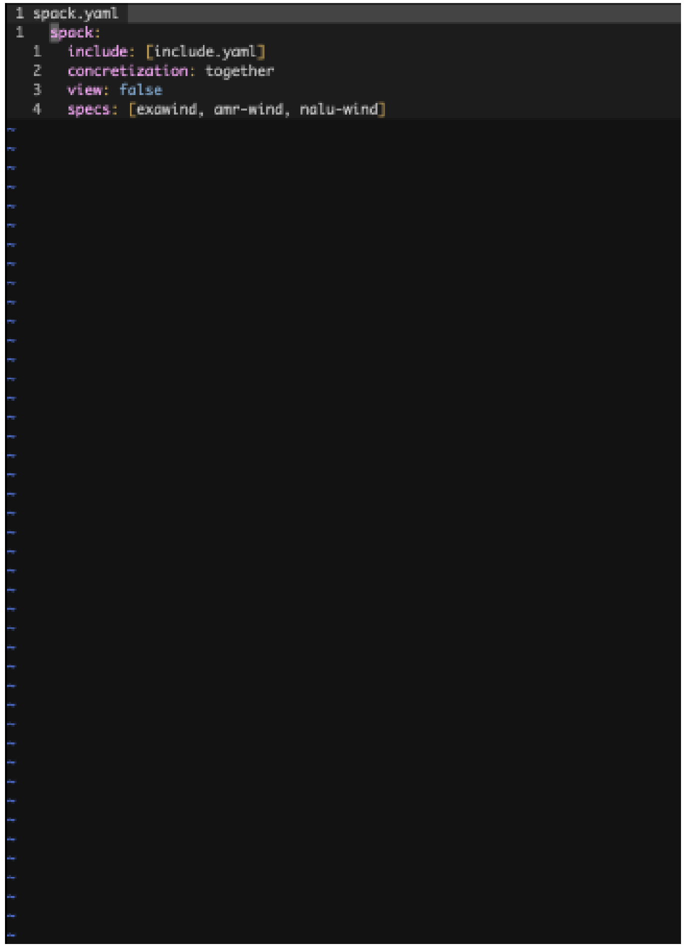
spack manager create-env –d [foo] –s [specs]



spack env create -d [foo]
spack env activate -d [foo]
spack repo add ${SPACK_MANAGER}/repos/exawind
manually insert machine specific configs: packages.yaml, configs.yaml, compilers.yaml
spack add [specs]
spack env deactivate

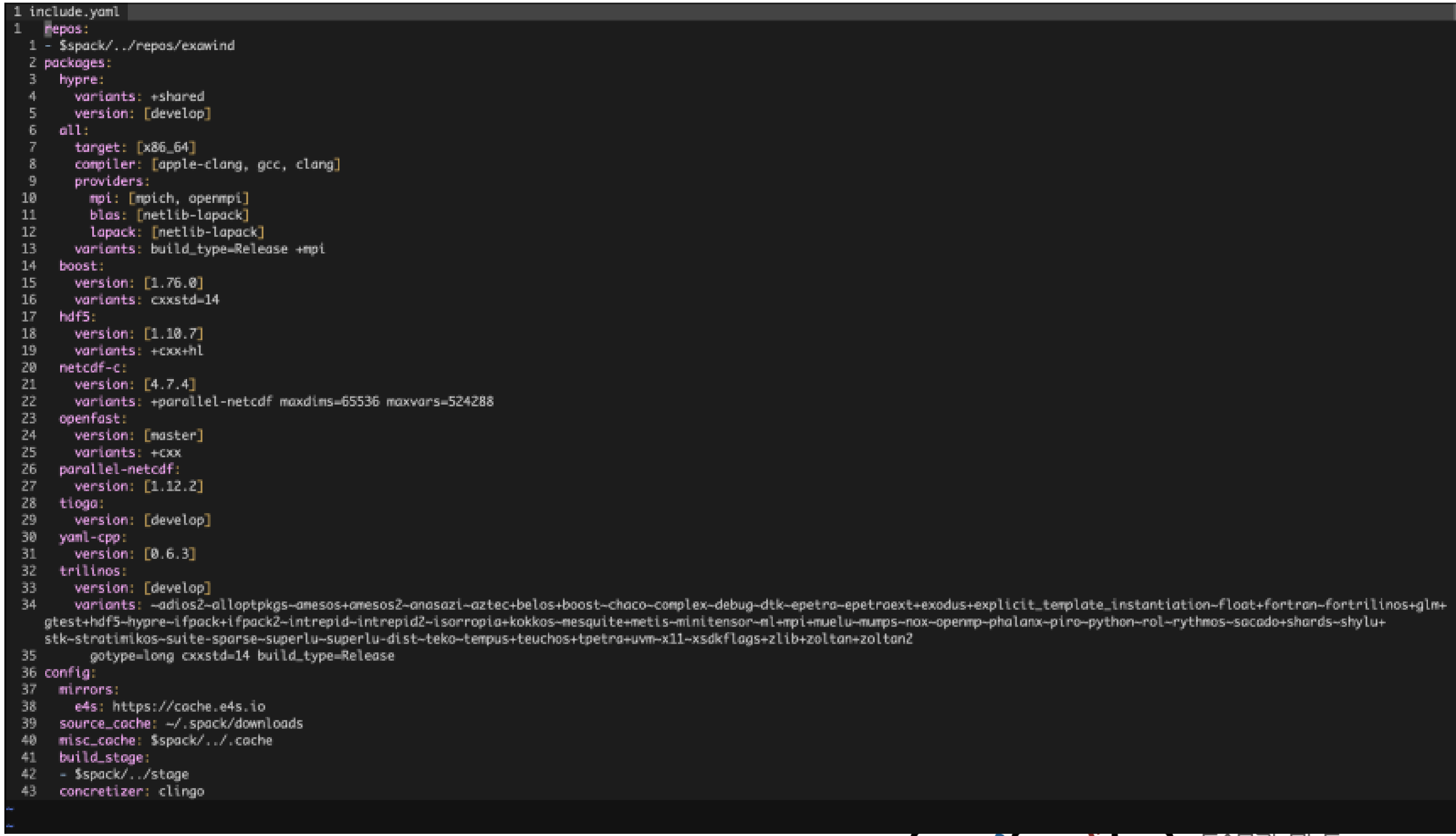
What does it look like?

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



1 spack.yaml

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



1 include.yaml

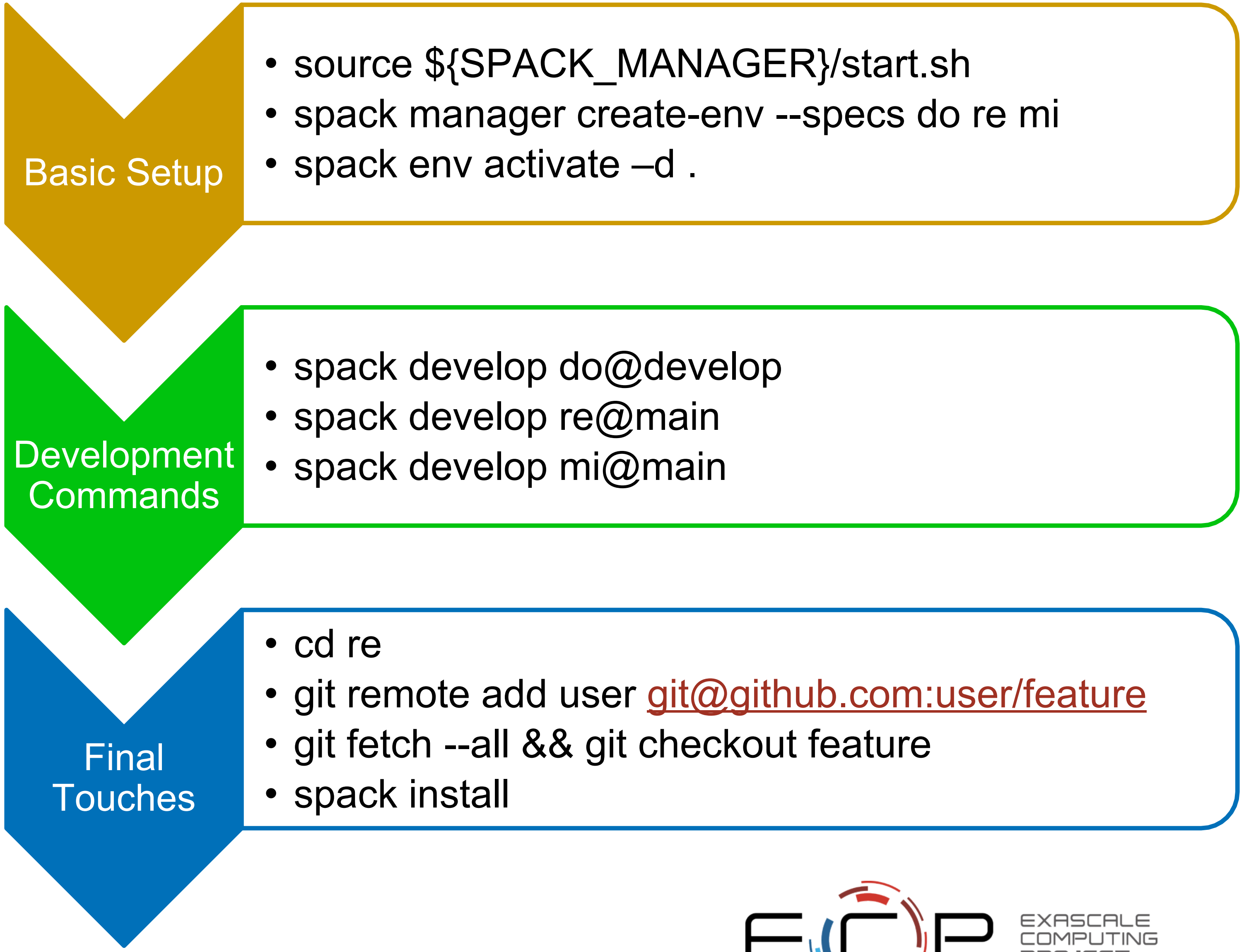
```
repos:
- $spack/../repos/exawind
packages:
  hypre:
    variants: +shared
    version: [develop]
  all:
    target: [x86_64]
    compiler: [apple-clang, gcc, clang]
    providers:
      mpi: [mpich, openmpi]
      blas: [netlib-lapack]
      lapack: [netlib-lapack]
    variants: build_type=Release +mpi
  boost:
    version: [1.76.0]
    variants: cxxstd=14
  hdf5:
    version: [1.10.7]
    variants: +cxx+hl
  netcdf-c:
    version: [4.7.4]
    variants: +parallel-netcdf maxdims=65536 maxvars=524288
  openfast:
    version: [master]
    variants: +cxx
  parallel-netcdf:
    version: [1.12.2]
  tioga:
    version: [develop]
  yaml-cpp:
    version: [0.6.3]
  trilinos:
    version: [develop]
    variants: ~adios2~alloptpkgs~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+muelu~mumps~nox~openmp~phalanx~piro~python~rol~rythmos~sacado+shards~shylu+stk~stratimikos~suite-sparse~superlu~superlu-dist~teko~tempus+teuchos+tpetra+uvm~x11~xsdkflags+zlib+zoltan+zoltan2
      gotype=long cxxstd=14 build_type=Release
config:
  mirrors:
    e4s: https://cache.e4s.io
  source_cache: ~/.spack/downloads
  misc_cache: $spack/../.cache
  build_stage:
  - $spack/../stage
  concretizer: clingo
```

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
- Since roll out only 1-2 issues a month from entire team of developers

Development Environment

- spack develop is amazingly powerful but …
- Setting up a development environment can still be a lot of work
- Can start to feel tedious when done often
- Number of commands can be reduced with some basic assumptions



Basic Setup

- source ${SPACK_MANAGER}/start.sh
- spack manager create-env --specs do re mi
- spack env activate –d .

Development Commands

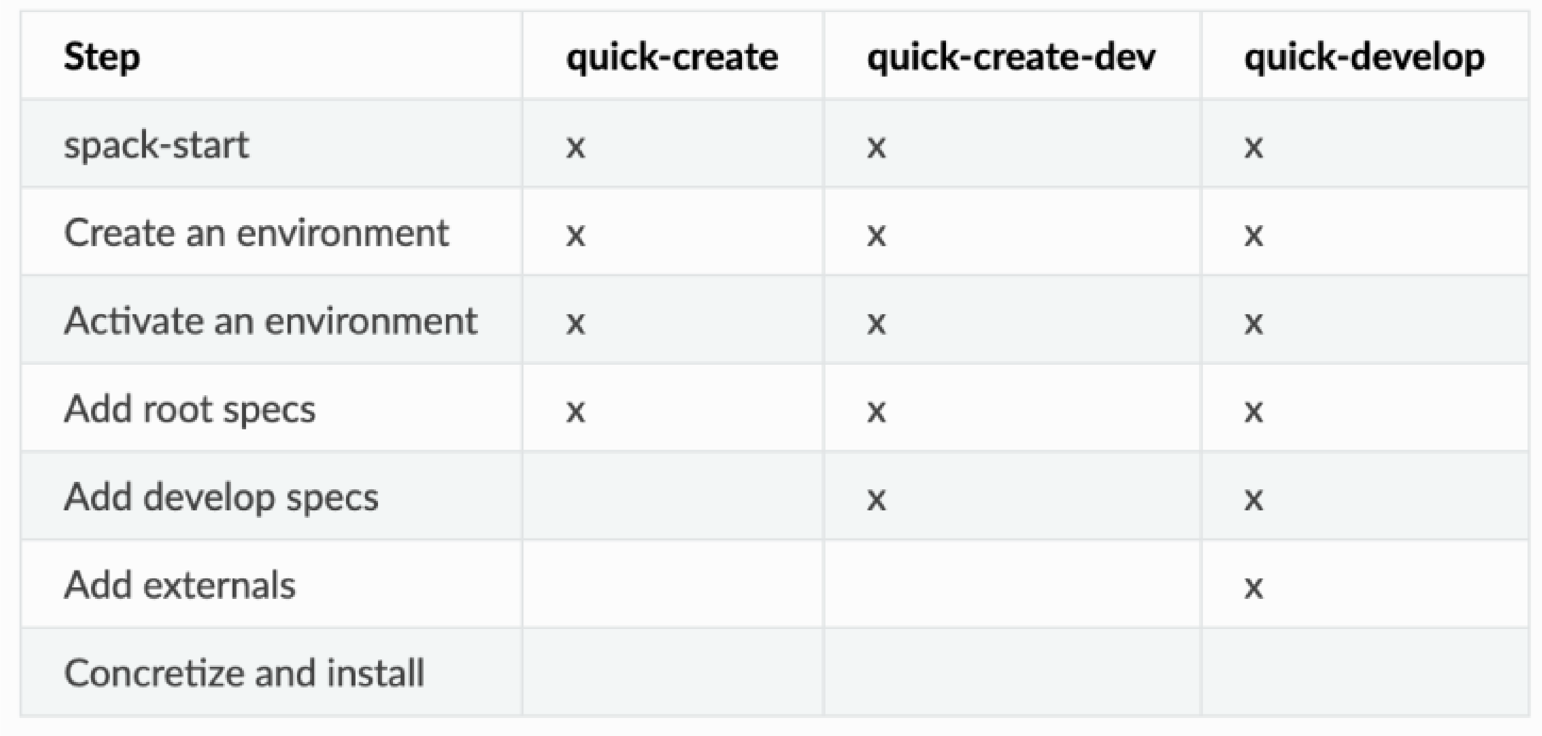
- spack develop do@develop
- spack develop re@main
- spack develop mi@main

Final Touches

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

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

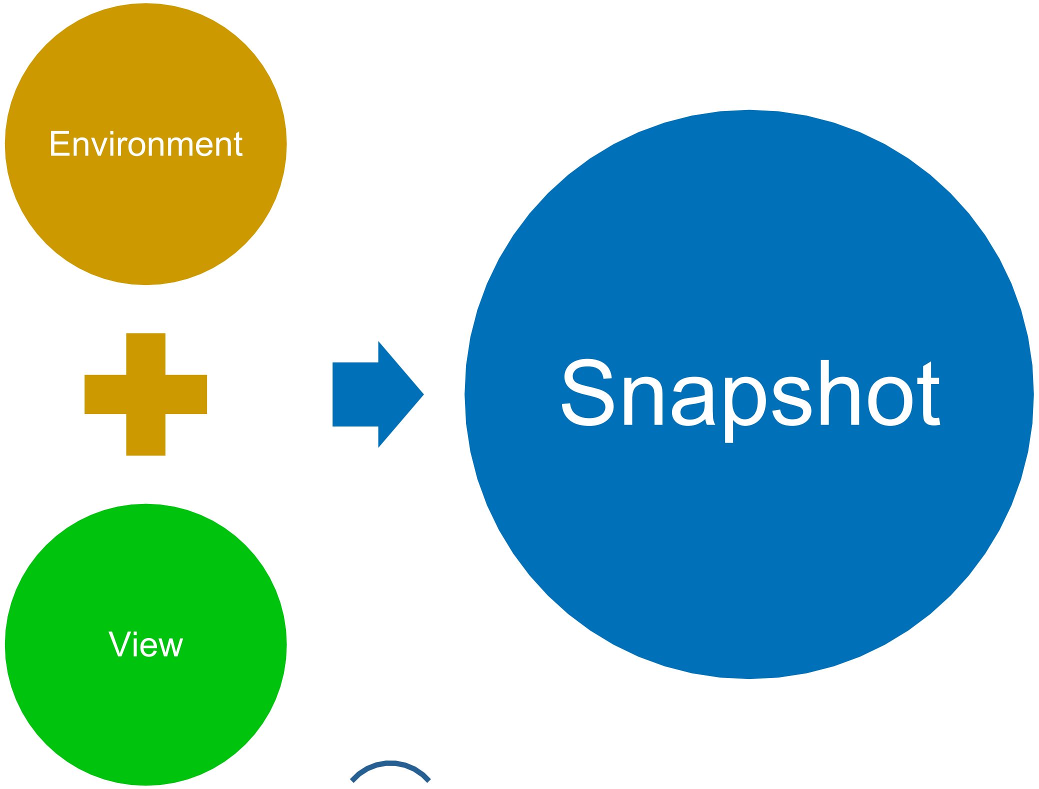


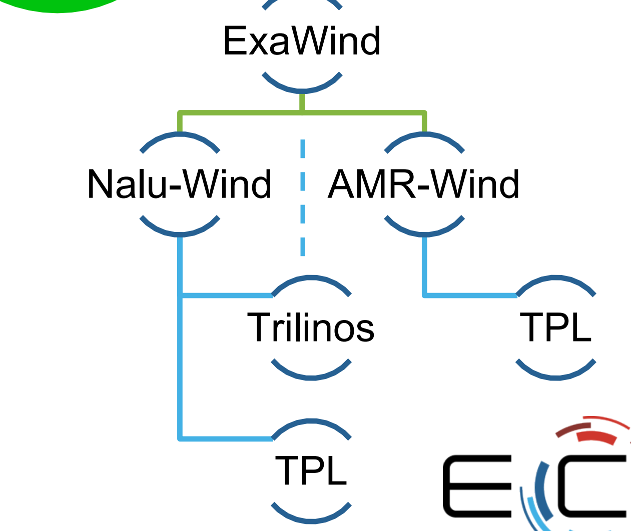
Step	quick-create	quick-create-dev	quick-develop
spack-start	x	x	x
Create an environment	x	x	x
Activate an environment	x	x	x
Add root specs	x	x	x
Add develop specs		x	x
Add externals			x
Concretize and install			

- 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



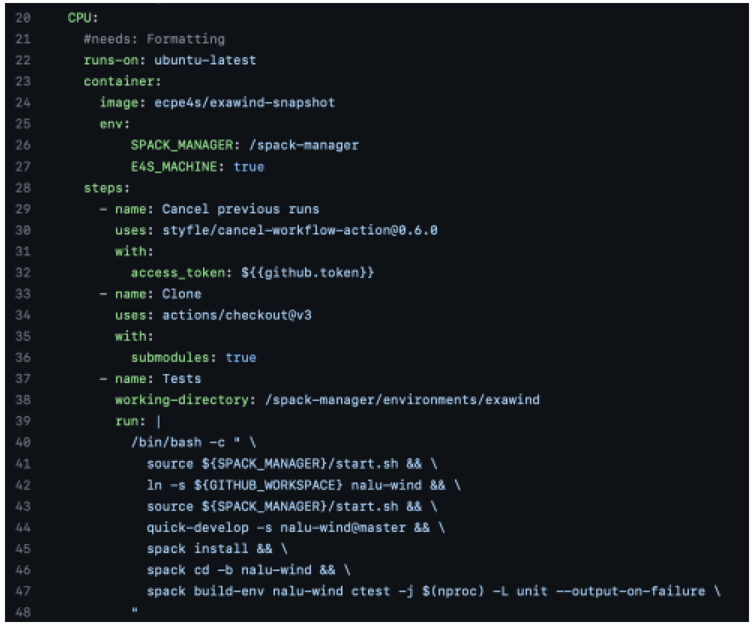


Containers

- 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

docker

GitHub Actions



```
  CPU:
    #needs: Formatting
    runs-on: ubuntu-latest
    container:
      image: ecpe4s/exawind-snapshot
      env:
          SPACK_MANAGER: /spack-manager
          E4S_MACHINE: true
    steps:
      - name: Cancel previous runs
        uses: styfle/cancel-workflow-action@0.6.0
        with:
          access_token: ${{github.token}}
      - name: Clone
        uses: actions/checkout@v3
        with:
          submodules: true
      - name: Tests
        working-directory: /spack-manager/environments/exawind
        run: |
          /bin/bash -c " \
            source ${SPACK_MANAGER}/start.sh && \
            ln -s ${GITHUB_WORKSPACE} nalu-wind && \
            source ${SPACK_MANAGER}/start.sh && \
            quick-develop -s nalu-wind@master && \
            spack install && \
            spack cd -b nalu-wind && \
            spack build-env nalu-wind ctest -j $(nproc) -L unit --output-on-failure \
          "
```

Pros and Cons of Spack Driven Development

Pros

- Spack is already solving the dependency issues
- Spack is scalable
  - DAG parallelism
    - 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

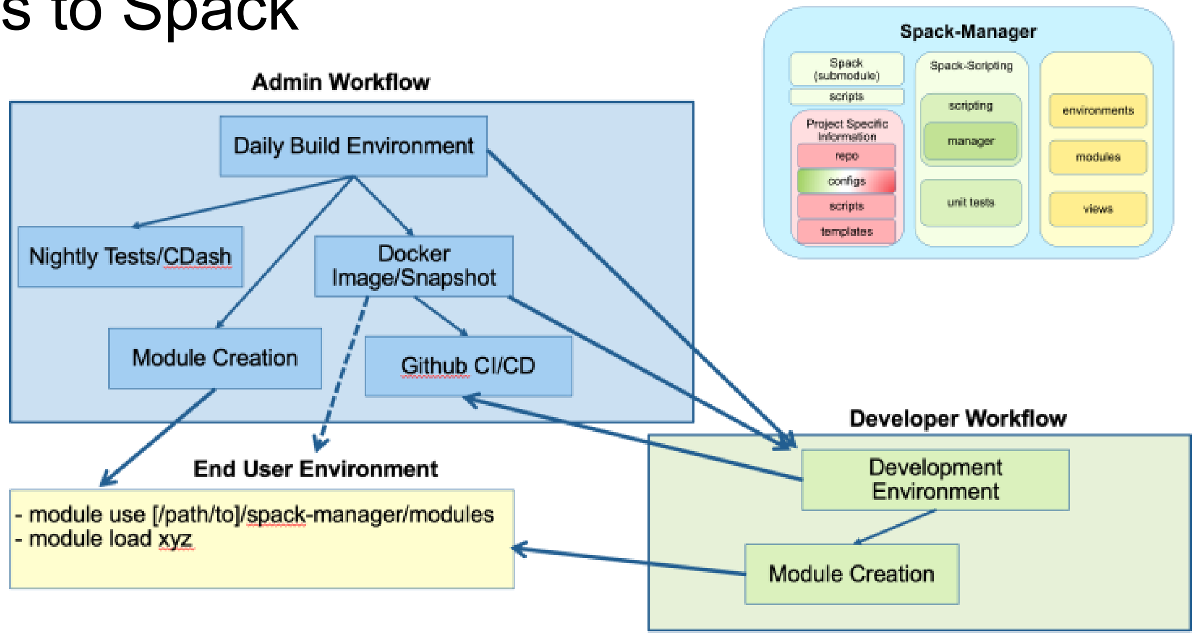
- Spack can be overwhelming
  - 3-5 ways to do just about everything
- Spack build process has some quirks
  - Hash based issues and confusion
  - Bootstrapping and occasional ssl issues
- Spack data management and logs make developers uncomfortable
  - spack-build-[hash]
  - spack cd -b [package]
-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

Conclusions

- Very happy with Spack as the driver for development
  - Unified API dramatically reduces infrastructure needs
  - Gives developers the tools to customize their own environments
- Cons can be mitigated with education and light scripts
- Highly recommend extension to wrapping when interfacing with Spack
  - Light buffer for applications
  - Less code is more
- Spack-Manager is one example of how this can be done
  - With Spack there are 3-5 ways to do everything ☺

What’s next for Spack-Manager?

- Immediate future:
  - – Repo migration:
    - https://github.com/psakievich/spack-manager to https://github.com/sandialabs
  - – Improve unit-testing
  - – Upstream more package improvements to Spack



- Long term:
  - – Upstream features to Spack and formalize pipeline for future efforts
  - – Add additional projects
    - Pele-C
    - ???

Contact: psakiev@sandia.gov