This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2022-7044C

History of CDE

ASC DevOps Core:

Gary Lawson

Jonathan Grzybowski

Christopher Sullivan

Paul Wolfenbarger

Scott Warnock

Etone Mbome

May 25th

2022 Tri-lab Advanced Simulation & Computing Sustainable Scientific Software Conference

Albuquerque, NM, USA

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.

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.

Introduction – What is ASC DevOps Core @ Sandia?

Product Owner:

- Scott Warnock

Scrum Master:

- Etone Mbome

Development Team:

- Jon Grzybowski
- Gary Lawson
- Chris Sullivan
- Paul Wolfenbarger

What do we do?

Code Scan

Code Vulnerability Scanning Tool

Common Development Environment

Consistent Multi-platform Software Stack

Dockerized Services

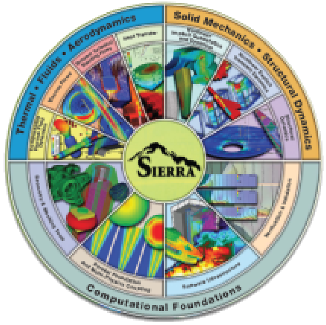
Jenkins
CDash

General Support

Hydra
VisIt

Introduction – Common Development Environment

- **Support ASC Code teams** on computing platforms critical to the ASC Mission



RAMSES

Radiation Analysis Modeling and Simulation of Electrical Systems

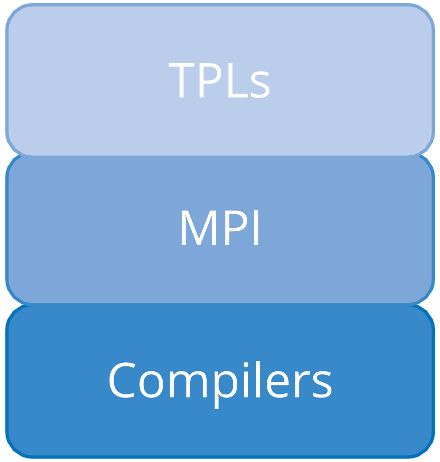
Excellence Through Tradition

CTH

Peerless Through Innovation

- Sandia's vision for **unified development environment** across computing platforms
  - HPC Clusters (x86_64, power9, aarch64, cuda)
  - CEE (Common Engineering Environment) Resources
  - Desktops & Workstations
  - Containers

- **Sustainability**
  - Will the same approach work in 10 years?
  - What is the measure of success?
  - How to minimize technical debt?



TPLs

MPI

Compilers

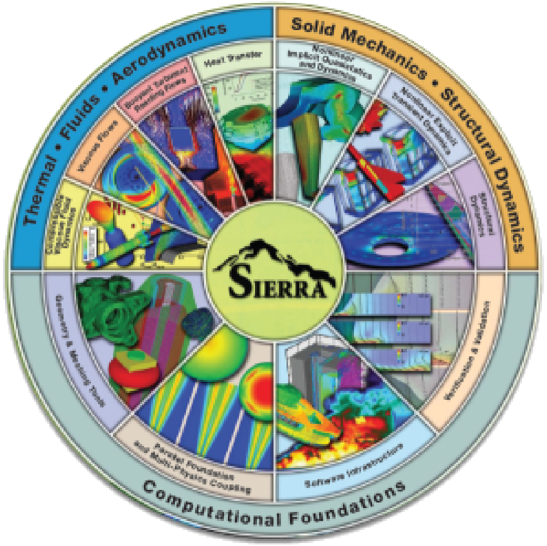
Early days of CDE

Interviewed an experienced DevOps team

- Met with SIERRA DevOps & NGS
- Asked questions on how to promote sustainability
  - Manage our technical debt
  - Explore new technologies early
  - Define interface boundaries

Smallest vertical slice

- Two platforms:
  - HPC (CTS1) - RHEL 7
  - CEE - RHEL 6 & 7
- Two software packages:
  - CMake
  - Anaconda


Thermal • Fluids • Aerodynamics
Heat Transfer
Gradient Turbulent Reacting Flows
Viscous Flows
Solid Mechanics • Structural Dynamics
Nonlinear Implicit Quasistatics and Dynamics
Nonlinear Explicit Transient Dynamics
Structural Dynamics
SIERRA
Geometry & Meshing Tools
Parallel Foundation and Multi-Physics Coupling
Software Infrastructure
Verification & Validation
Computational Foundations

Answer fundamental questions:

- How do we measure success?
- How to build software?
- What technology to automate?
- What about containerization?
- What is the process?

How do we measure success?

ASCDO Definition of Done

Monitored

At a glance, did the process PASS/FAIL

Automated

Every process is automated

Tested

Every product is tested as thoroughly as is feasible

Reviewed

Every process is reviewed for value added

Technical Debt

Minimize:

- Manual processes
- Maintained code, scripts, modifications
- Unbounded influences and one-offs

Identify:

- Interface boundaries
- Sources of productivity degradation

Sustainability MATR's to the ASC DevOps Core

How do we build software?

RPM Builds

Red Hat Package Manager
Commercially available

- Not designed for HPC ecosystems
  - MPI combinatorics
  - Exotic hardware
- Only works for Linux

Spack Builds

Alpha-stage multi-platform package manager

- Designed for HPC ecosystems
  - MPI combinatorics
  - Exotic hardware
- Linux, Windows, and MacOS supported

Spack provides the flexibility required to build for the target platforms

What technology do we use to automate?

Gitlab CI

- Commercially available
- System administrator or user -supported

- YAML interface
- Multiple execution types
  - SSH, Shell, Docker, etc
- Redundancy in scaling to multiple platforms
- Supports variables
- Spack supports designing Gitlab-CI pipelines*

Ansible

- Commercially available
- User-supported

- YAML interface
- Executes on the system
  - Can be integrated into container
- Requires Python on system
  - Supports templating
- Efficient scaling to multiple platforms
- State-driven resource model

Ansible chosen for state-driven resource model, efficient scaling, and templating capabilties

What about containerization?

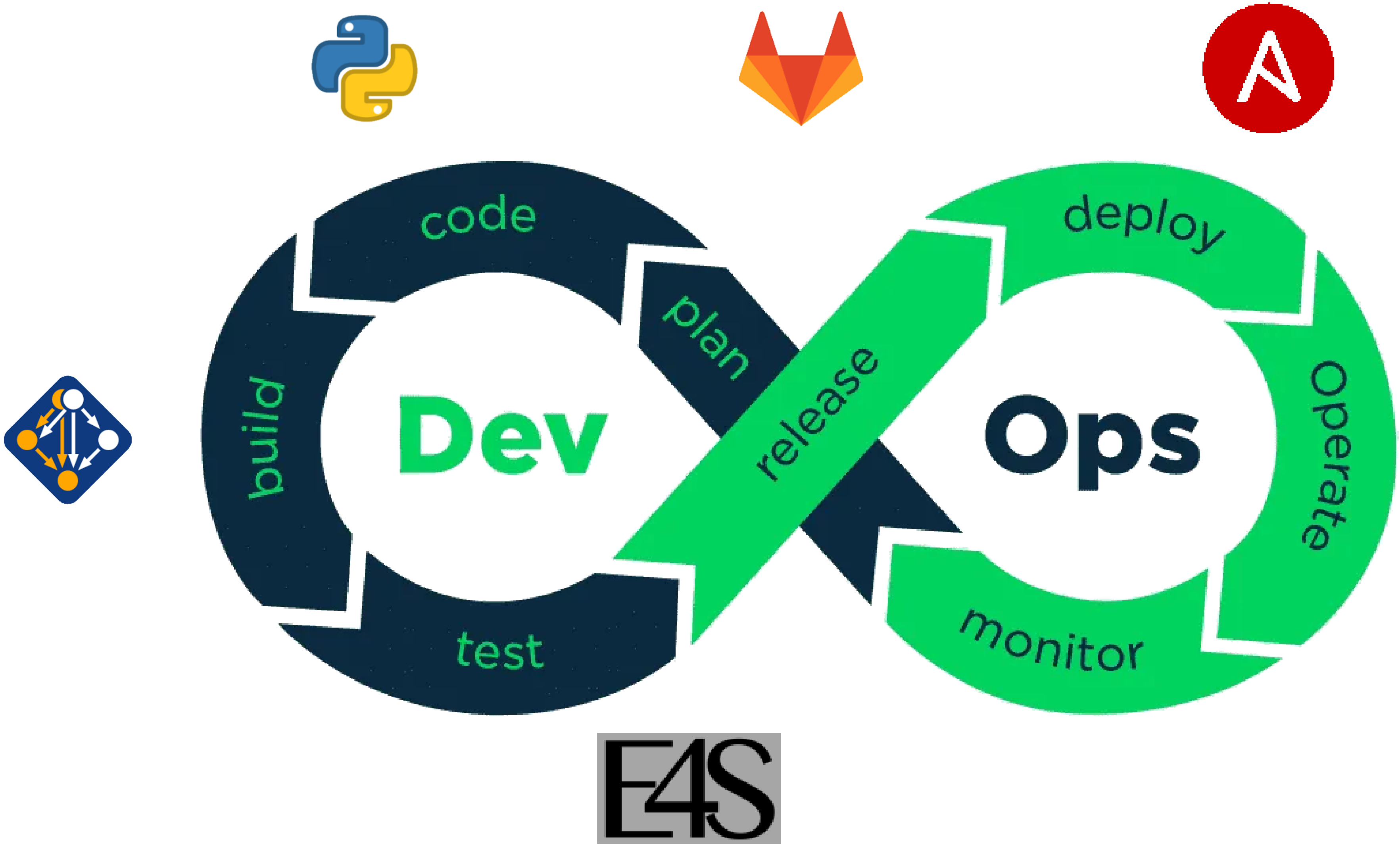
We initially designed our pipelines to build in a container

- Containers acted as clean rooms which could be discarded after a build
- Buildcache was generated after the build for deployment
  - Uploaded to Nexus Repository for reuse
- Deployment on bare-metal from buildcache
  - Encountered difficulties with maintaining the buildcache repository and indexes
  - Spack often would not/could not install from buildcache because of SHA-1 conflicts

For now, we are not supporting containerized deployments and builds

- Executing containers is not yet available for every target platform, i.e. HPC cluster's
- Growing interest in containerized deployments
  - We will revisit this in the near future

Build and Deployment Technologies



Ansible

Gitlab

Python

Spack

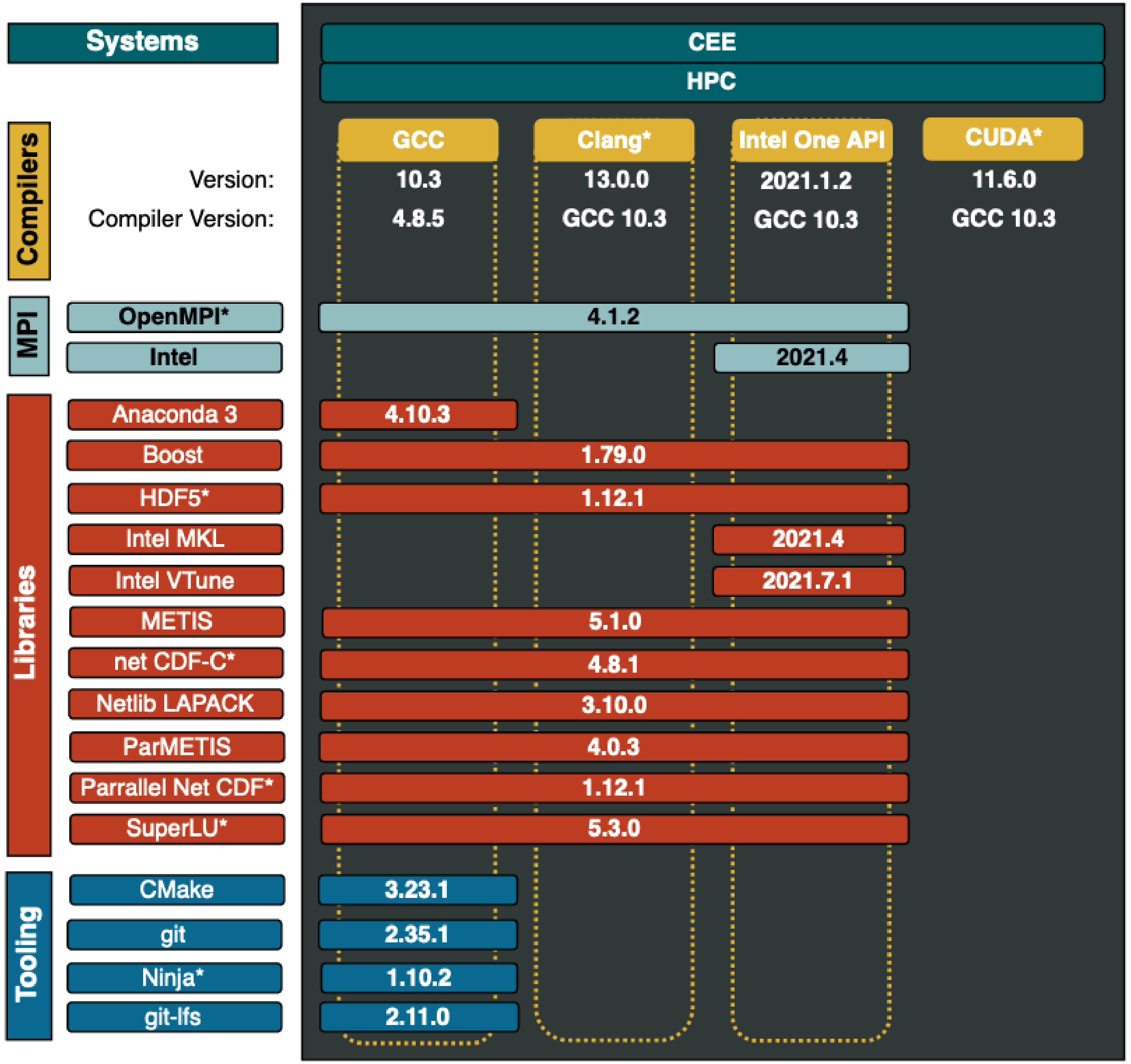
E4S

Approach to Software Deployment

- Vertical Slice
  - Support one code team
  - Support their software stack
  - Explore Technologies
  - Create Automated Process
    - Build
    - Test
    - Deploy

- Expand the Slice
  - Support additional teams
  - Extend software stack
  - Improve processes
  - Support additional platforms

Influences convergence on a unified stack

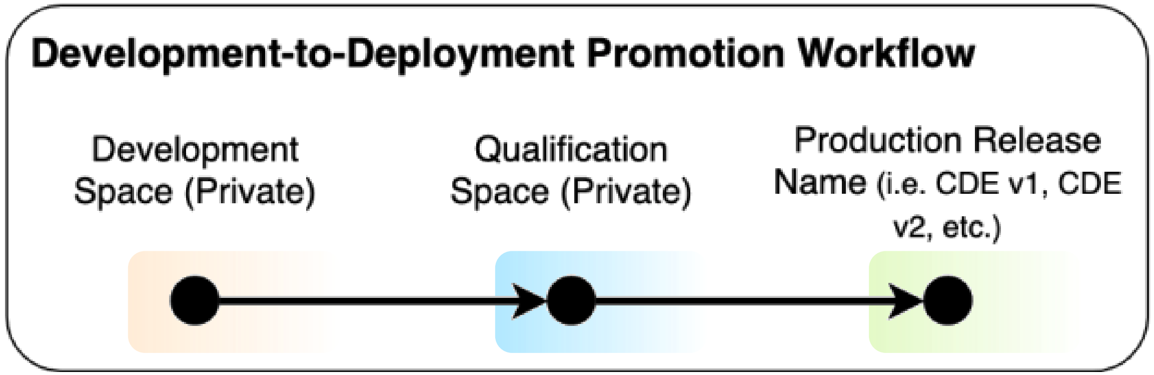


Systems: CEE, HPC

		GCC	Clang*	Intel One API	CUDA*
Compilers	Version:	10.3	13.0.0	2021.1.2	11.6.0
	Compiler Version:	4.8.5	GCC 10.3	GCC 10.3	GCC 10.3
MPI	OpenMPI*	4.1.2			
	Intel			2021.4	
Libraries	Anaconda 3	4.10.3			
	Boost	1.79.0			
	HDF5*	1.12.1			
	Intel MKL			2021.4	
	Intel VTune			2021.7.1	
	METIS	5.1.0			
	net CDF-C*	4.8.1			
	Netlib LAPACK	3.10.0			
	ParMETIS	4.0.3			
	Parralel Net CDF*	1.12.1			
	SuperLU*	5.3.0			
Tooling	CMake	3.23.1			
	git	2.35.1			
	Ninja*	1.10.2			
	git-lfs	2.11.0			

* E4S tested

Deployment Promotion Workflow



Development-to-Deployment Promotion Workflow

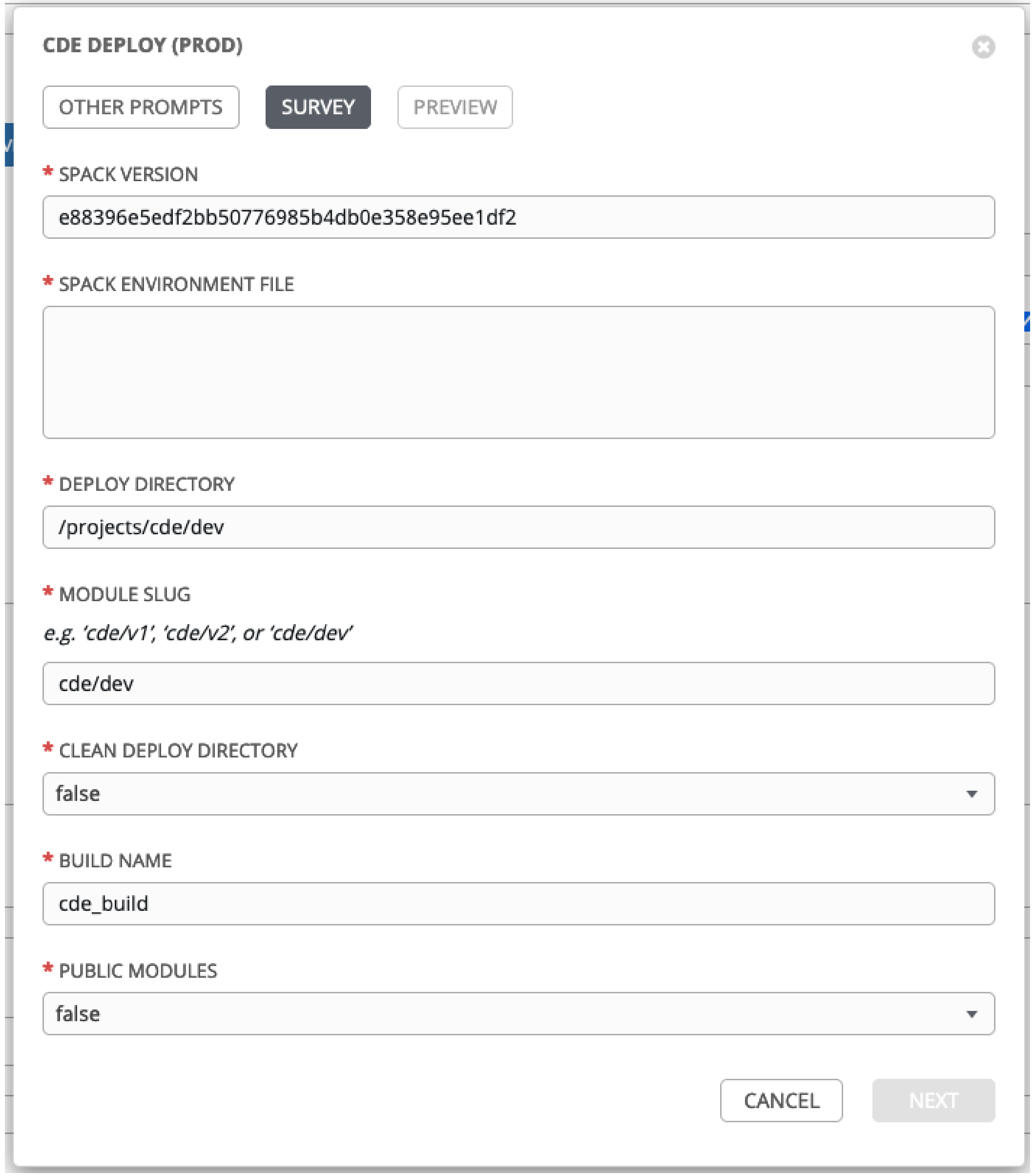
Development Space (Private)

Qualification Space (Private)

Production Release Name (i.e. CDE v1, CDE v2, etc.)

1. Development build is automated, but can require some manual development
   - Modify package versions and/or variants
   - Build and verify stability with testing
2. Once we have a stable build, verify automation can build it cleanly
   - No manual intervention
   - End-to-end installation without errors or failures
3. Once we can build in automation cleanly
   - Verify we can build weekly
   - Build out to qualification space for additional testing
4. Once the additional testing passes
   - Build in production space as a named release
   - Symlink to public module space

CDE: Current Workflow with Ansible Tower – Dev Space



CDE DEPLOY (PROD)

* SPACK VERSION

e88396e5edf2bb50776985b4db0e358e95ee1df2

* SPACK ENVIRONMENT FILE

* DEPLOY DIRECTORY

/projects/cde/dev

* MODULE SLUG

e.g. 'cde/v1', 'cde/v2', or 'cde/dev'

cde/dev

* CLEAN DEPLOY DIRECTORY

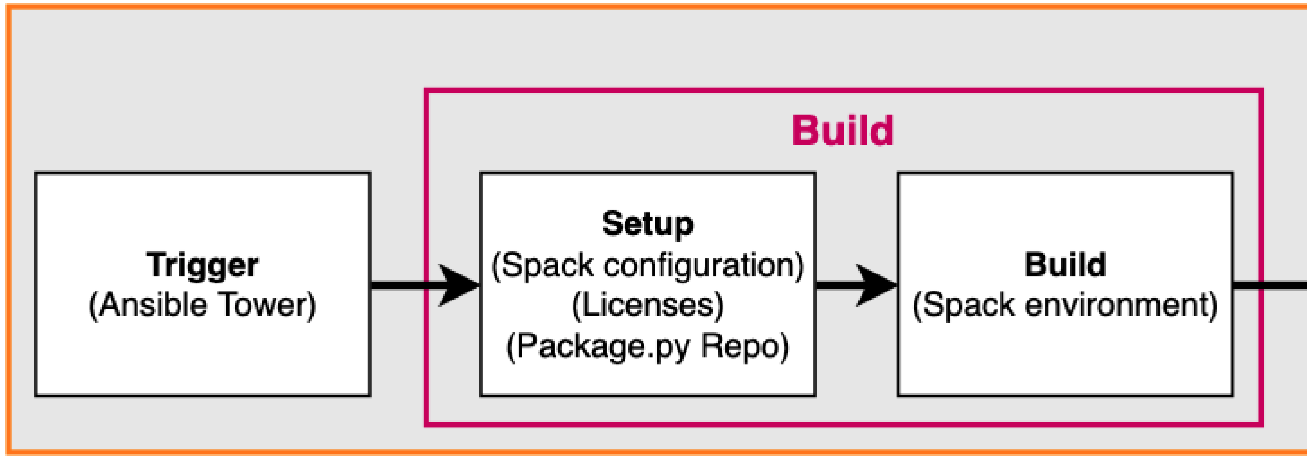
false

* BUILD NAME

cde_build

* PUBLIC MODULES

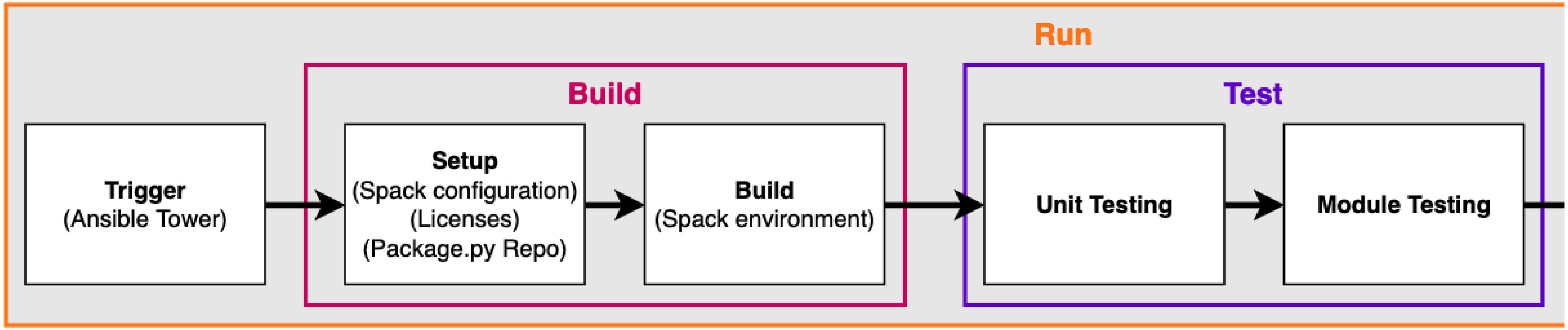
false



Workflow is triggered manually in Ansible Tower

- Specify target platforms
- Fill out survey options for automation
  - Environment File
  - Deploy Directory
  - Build Name

CDE: Current Workflow with Ansible Tower – Dev Space

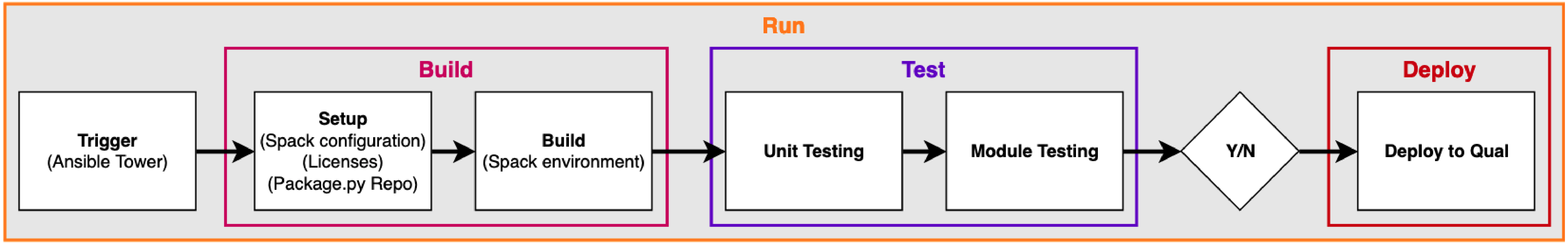


Execute workflow multiple times for staged builds:

1. GCC Compiler
2. Compilers – Intel, LLVM, Cuda, Platform Specific
3. Third-Party Libraries (TPL's)

Workflow executed on each target platform to a unique deployment directory

CDE: Current Workflow with Ansible Tower – Dev Space



Unit Testing

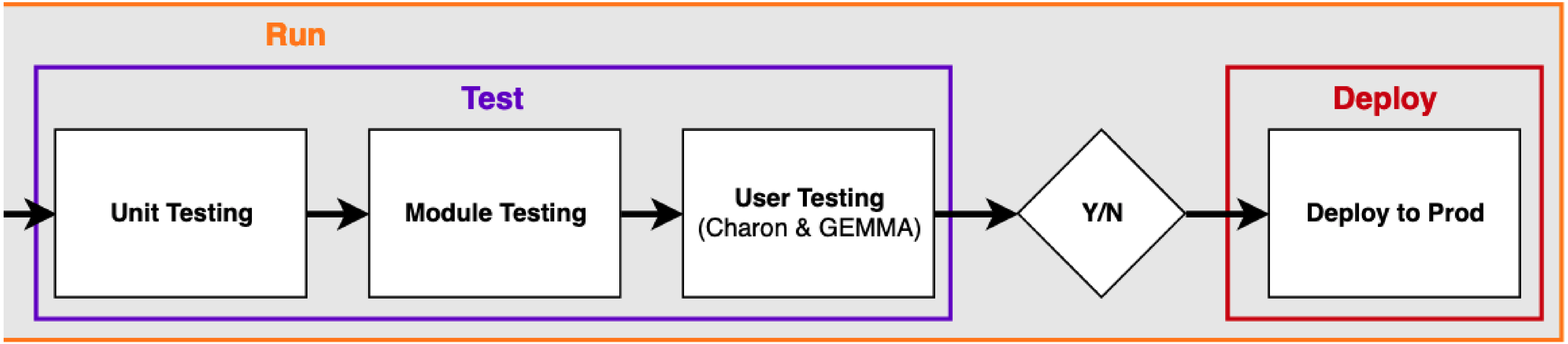
- Individual package tests
- Smoke and feature tests
- Tests provided with packages
- CDE developed tests

Module Testing

- Availability of the modules
- Load, Swap, Unload
- Autoload

With successful testing, promote to qualification space and build weekly

CDE: Current Workflow with Ansible Tower – Qual Space



User Testing

- Can our customers build on our stack?
- Integration testing with Charon & GEMMA
  - Trilinos as a dependency
- If Yes, deploy to production space

With successful testing, promote to production space and link modules publicly

Limitations of the Current Workflow

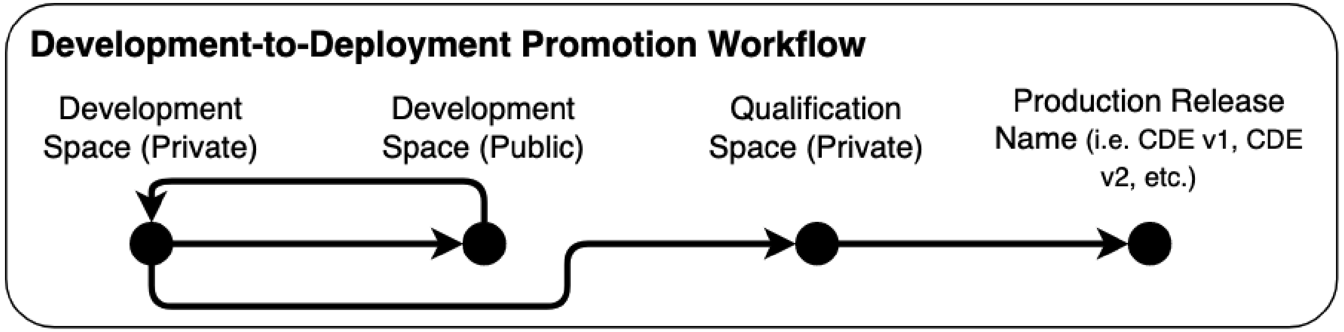
- **Limited parallelism**
  - Spack Pipelines – We currently lack sustainable gitlab runners
    - Desire for Jacamar runners as sustainable solution
  - Distributed CLI Spack

- **Coupling of Packages to Spack version**
  - Converging on need to build from Spack develop branch
  - Need process to verify builds in order to promote Spack develop branch commit

- Manual Trigger
  - Desire CI/CD workflow on our manifests

- Limited Testing & Analysis
  - Is the MPI we are currently building comparable in performance to prior releases?
  - Do the features the code teams need in software package X work?

Future Improvements


Development-to-Deployment Promotion Workflow
Development Space (Private)
Development Space (Public)
Qualification Space (Private)
Production Release Name (i.e. CDE v1, CDE v2, etc.)

- Improved parallelism
- Public development space for software rolling releases
- Improved automation
  - Automate build trigger's for Continuous Integration and Delivery
  - Sustainable integration testing
- Improved monitoring and user-statistics
- Caching builds
- Generating containers of the CDE software stack
- Provide our tooling for generating builds

Conclusions

1. We strive to provide the ASC Code Teams with the software they require to develop
   - And further, the tools they require to develop
     - Build caches for rapid deployment of an existing, stable software stack
     - Software stack deployment tooling for one-off and experimental builds

2. We envision a unified environment where the software needed by the code teams is readily available and with up-to-date versions

3. We strive to develop and maintain sustainable practices so the desired product is consistently delivered, no matter how the ecosystem and technology adapt

Points-of-Contact

Scott Warnock	sawarno@sandia.gov
ASC DevOps Core	asc-devops@sandia.gov