SAND2022-6778C

Sandia National Laboratories

DAKOTA

Explore and predict with confidence

Flexible CI/CD Software Tools for the Dakota project

J. Adam Stephens, Brian M. Adams, Wesley P. Coomber, Elliott M. Ridgway

Sandia National Laboratories

Tri-lab Advanced Simulation & Computing Sustainable Scientific Software Conference – May 25, 2022

U.S. DEPARTMENT OF ENERGY | NNSA

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.

Outline

- Overview of Dakota
  - What is Dakota?
  - Dakota's development, test, and deployment processes
- Dakota's new build/test/deploy software tools
  - Why we junked our old ones
  - Principles for the redesign
  - Implementation
- Conclusions

What's Dakota?

DAKOTA
Explore and predict with confidence.

Sandia-developed, open-source, CLI+GUI software for black-box ensemble analysis of computational simulations

History

- Began as an LDRD in 1994
- Initial optimization focus; now overtaken by UQ

Team

- Around 15 developers and 5 FTEs per year
- Primarily engineers and mathematicians

Usage

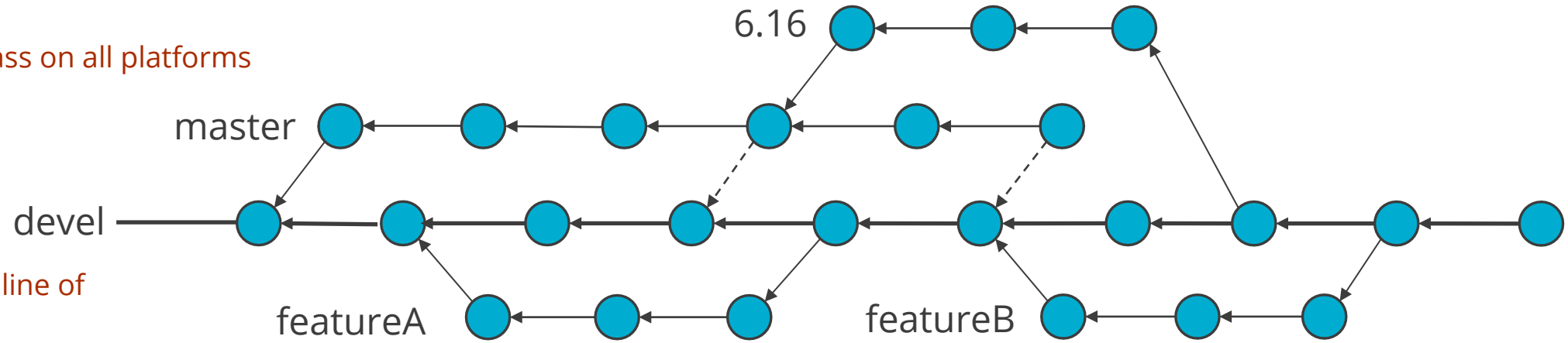
- Production – 100s of users at the Tri-labs and 1000s around the world
- Research - support collaborations between Sandia and external researchers and rapid deployment of research to mission problems

Platforms

- *Nix, OS X, Windows
- Desktops, HPCs
- Mix of internal and external builds

Development Process

Release branches drawn from master



Tests must pass on all platforms

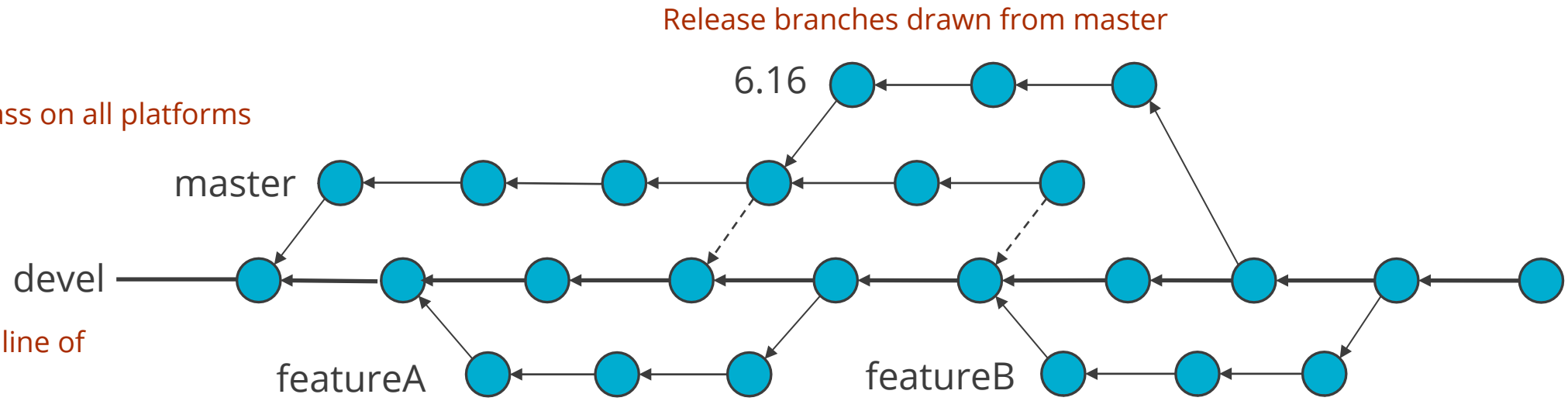
- devel is main line of development
- Tests must pass on baseline platform

Feature branches are drawn from devel

Testing Strategy

- Commits to devel trigger build/test on a representative subset of supported platforms
- Nightly build/test of devel provide feedback for all supported platforms

Development Process



- devel is main line of development
- Tests must pass on baseline platform

Deployment

Dakota has twice-yearly versioned releases and daily stable releases

- Both are installed on internal supported platforms and deployed to our website for download
- Stable releases are a way to get bugfixes and new features into users' hands quickly
- Versioned releases receive higher scrutiny, include updated documentation, and our GUI

CI/CD Tools Rewrite

Dakota embarked on a re-write of its CI/CD software tools in FY21

- Prior to the re-write
  - Used Jenkins to orchestrate build and test
  - Freestyle jobs with Bash or Windows Batch build steps
  - These ran layers of scripts and applications written in bash, bat, Python, Perl, Java, and CMake
- Shortcomings:
  - Missing features – We wanted to easily build and deploy any branch to our website and to internal supported platforms,
  - Low reliability – The existing tools were buggy, often in ways that masked problems
  - Maintainability – A hodgepodge of scripts that contained hard-coded configuration information
  - Poor insight - It was often difficult to pinpoint the cause of build system or test problems
  - Lack of version control – Some scripts and configuration lived in a git repo, but much was hard coded in Jenkins build steps and other web configuration

Principles for the Rewrite

Aimed at overcoming the limitations of the existing tools

- Strong separation of code from configuration
- Follow “clean code” guidelines
- Flexibility in configuration (Which platforms? What criteria?)
- Easy troubleshooting
- Code portability

Major Components

Jenkins

- Top-level Orchestration
- SEMS resource
- Primarily Pipeline projects

Build Configurations

- Specify what and how
- Platform specific Environment setup and build presets

jenkota

- Intermediate layer
- Custom written Python package

CMake

- Dakota's build system

Organization

Jenkins Pipeline Projects

```
stage("Test") {
  try {
    python("jenkota.builddriver test")
    result = attach_test_results(result)
    result.stages.test = 'SUCCESS'
  } catch(err) {
    result.stages.test = 'FAILURE'
    result = attach_test_results(result)
    throw err
  }
}
```

jenkota

```
def invoke_ctest(command, printing):
    """Run ctest"""
    command_str = " ".join(command)
    logger.info(f"Invoking ctest using the command: {command_str}")
    p = subprocess.Popen(command, universal_newlines=True,
        stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    with p.stdout:
        log_ctest_output(p.stdout, printing)
    exitcode = p.wait()
    return exitcode
```

Environment setup

strategy

Build Configurations

Build presets

Dakota's CMake build system

LEGEND

RUNS

CONFIGURES

Jenkins

Jenkins provides top-level orchestration of build, test, and deploy

- Jenkins was a natural choice because of Dakota team experience and SEMS support
- In Jenkins Pipeline projects, build/test/deploy processes are scripted in a DSL
- Pipeline projects offer several benefits:
  - Power and flexibility for creating build/test/deploy processes in code
  - Version control
  - Mechanisms for code reuse
  - Stage-oriented control and reporting

```
stage("Test") {
  try {
    python("jenkota.builddriver test")
    result = attach_test_results(result)
    result.stages.test = 'SUCCESS'
  } catch(err) {
    result.stages.test = 'FAILURE'
    result = attach_test_results(result)
    throw err
  }
}
```

	Setup	Clone	Configure	Build	Test	Package	Publish
Average stage times: (Average full run time: ~55min 37s)	32ms	27s	1min 57s	13min 47s	37min 3s	1min 53s	11s
#27 May 18 02:18 No Changes	34ms	28s	58s	14min 9s	22min 9s	1min 24s	11s
#26 May 18 01:26 No Changes	36ms	28s	1min 9s	16min 24s	31min 55s	1min 24s	10s

Build Configurations

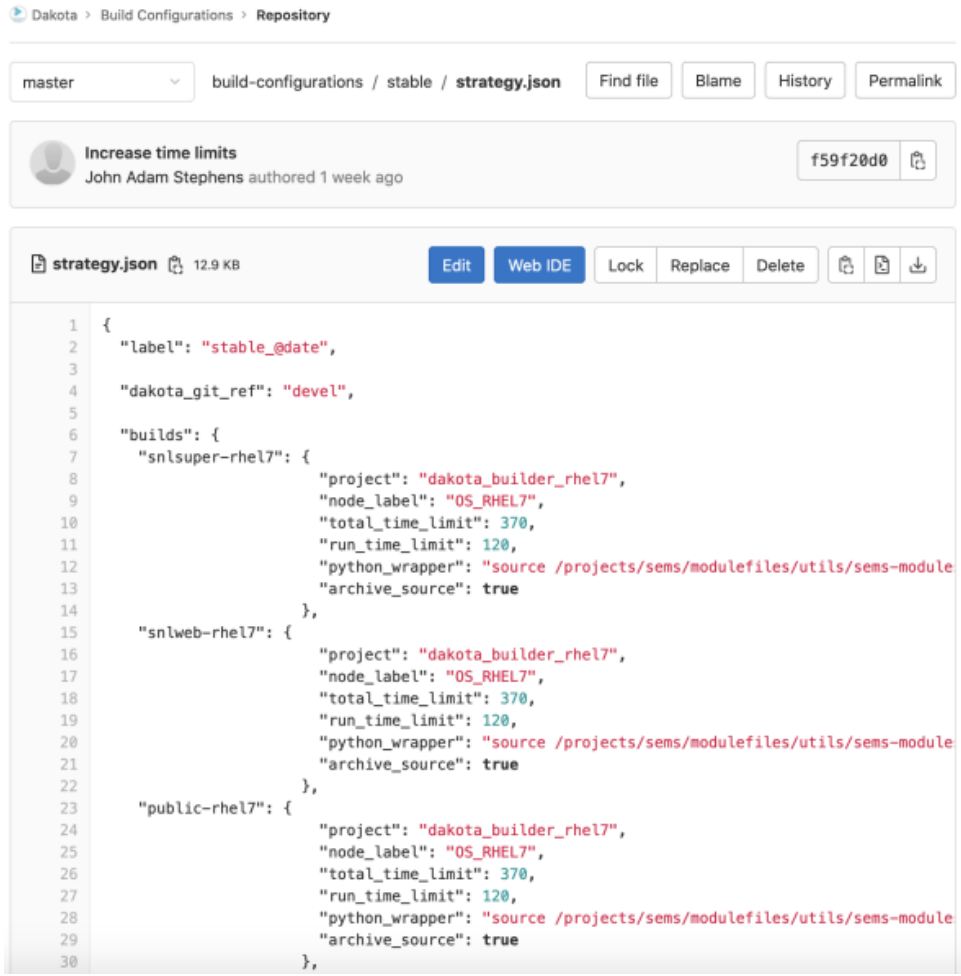
Build Configurations specify what and how to build, test, and deploy

Current configurations

- Continuous – Build/test a subset of platforms
- Stable – Build/test/deploy all platforms; merge devel to master
- Branch – On-demand, custom platform list and operations

Three parts -

- *strategy* for performing a set of builds and deployments (JSON configuration file)
- *Environment setup* – platform-specific procedures that contain limited operations (e.g. module load, env var setting)
- *Build presets* – Cmake cache files that tailor Dakota's build system to specific platform



Stable strategy.json

jenkota Python package

`jenkota` is used by Jenkins Pipelines to build, test, deploy, and other tasks

- Developed using “clean code” practices, including TDD
- Command line interface★: `$ python3 -m jenkota.builddriver configure stable snlsuper rhel7`

Module	Description
`builddriver`	Load the environment and run CTest to configure, build, test, etc.
`evaluate`	Compare test results and other artifacts to criteria
`install`	Install Dakota on an internal platform (CEE LAN, HPC)
`web_deploy`	Make Dakota available on our website
`paramify`	gzip and base64 encode
`stable`	Manage installation of stable builds
`report`	Prepare an email report

★ Credit: Jason Gates

CMake

Dakota's build system is CMake-based

- Dakota uses CMake for build, test, and packaging.
- A CTest script is the entry point for our automated build system
  - -D arguments specify the operations (configure, build, test, package, etc) to run
  - This interface is wrapped by jenkota.builddriver

Organization

Jenkins Pipeline Projects

```
stage("Test") {
  try {
    python("jenkota.builddriver test")
    result = attach_test_results(result)
    result.stages.test = 'SUCCESS'
  } catch(err) {
    result.stages.test = 'FAILURE'
    result = attach_test_results(result)
    throw err
  }
}
```

jenkota

```
def invoke_ctest(command, printing):
    """Run ctest"""
    command_str = " ".join(command)
    logger.info(f"Invoking ctest using the command: {command_str}")
    p = subprocess.Popen(command, universal_newlines=True,
        stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    with p.stdout:
        log_ctest_output(p.stdout, printing)
    exitcode = p.wait()
    return exitcode
```

Environment setup

strategy

Build Configurations

Build presets

Dakota's CMake build system

LEGEND

RUNS

CONFIGURES

Good Outcomes

- Increased confidence that things are working as they ought
- Easier to identify where processes went wrong
- Adding new platforms and features to our build system has become far easier
- Automated deployment of stable builds has had direct impact on an L2 milestone that the Dakota team is responsible for
- Releases are faster and require less manual work

- Not everything is perfect..
  - We designed a lot of flexibility into the system. Maybe too much.
  - Jenkins pipeline limitations
- Overall we’re happy