A DevOps Approach to Integration of Software Components in an EU Research Project

Mark Stillwell    Jose G. F. Coutinho

Department of Computing
Imperial College London, UK

September 1, 2015
An article about computational science in a scientific publication is not the science itself, it is merely advertising of the scholarship. The actual scholarship is in the complete software development environment, [the complete data] and the complete set of instructions which generated the figures. — David Donoho “Wavelab and Reproducible Research”, 1995
Funder Expectations

- multi-partner collaborations
- results data and code as research outputs
- reusable and maintainable software
- plans for long-term stewardship
HARNESS Project Summary

- EU FP7 funded cloud computing project
- makes available various heterogeneous resources
- multiple sub-projects developed by independent teams
- need to provide coherent demonstrator platform
HARNESS Project Teams

Integration Team

Storage Team

Compute Team

Network Team

Platform Team
HARNESS Project Architecture

ConPaaS

Users

submit app, manifest, SLO

feedback

Application Manager (AM)

Platform Layer

ConPaaS

submit application

feedback

Allocation request

feedback

Cross-Resource Scheduler (CRS)

IRM-NEUTRON network resources

IRM-NOVA VMs

IRM-SHEPARD HW accelerators

IRM-XtreemFS storage devices

Available resources

available resources

feedback

Neutron Agent

OpenStack Neutron Controller

manage

schedulers

OpenStack Nova Controller

manage

servers

Nova Compute

PCIe device reservation

SHEPARD Compute

Infrastructure Layer

deploy and execute services + applications

ConPaaS agent

virtual machines

Service Layer

OpenCL MaxelerOS AlphaData XtreemFS client

Executive POSIX

Application Module
Testbed Environments

- Imperial Testbed
  - small scale
  - static environment with shared systems
  - specialized hardware (GPU, MPC-X, SSD cards)

- Grid5000 Testbed
  - medium to large scale, some multi-site deployments
  - dynamic environment
  - virtual networking links
  - some specialized hardware (GPU, Intel Phi)
Initial Approach

- developer virtual machine images
- interactive configuration with some scripting (bash, devstack)
- scheduled releases of updated images
Significant Issues

- difficulty merging, managing, and tracking changes
- individual developer VMs tend to “drift” over time...
- fragmentation: hard to point to a definitive latest version
- difficult to debug or identify differences between images
- time-consuming and error-prone deployment to testbeds
Objectives for New Approach

- let developers easily work individually
- turn configuration/setup issues into software issues
- allow for version control, merging
- allow for automated acceptance testing
Differences from Commercial Requirements

- priority is individual research contributions
- lower focus on ease-of-use
- more need for customization
- need for reproducibility
Technologies

- Git / GitLab / GitHub
- Ansible
- Docker
- Vagrant
- Buildbot
DevOps Workflow

- Development Team
- Version Control
- Automated Unit Tests
- Automated Integration Tests
- Integration Team
- Testbed Deployment
- Release

- check-in
- trigger
- feedback
- P = pass
- F = fail
- deployment and automated testing in a virtual development environment
- deployment and testing in Grid'5000 and Imperial Cluster testbeds
- release approval
Role of Docker

- Docker used whenever possible
  - some services need global machine state
- provides static release images, with some configuration
- isolates projects from each other
Deployment Projects

- use ansible for orchestration and configuration management
- unify sub-projects, pull in from multiple repositories
- ansible ensures configuration changes are “idempotent”, so can be run repeatedly on static testbed
Virtual Machine Environments

- configured using vagrant+ansible
  - developer just checks out deployment, runs “vagrant up”
- allows developers to work independently
- easy to re-initialize or update
Reproducible Deployment

- developers work in the same environment, changes easily merged
- experiments and benchmarks can be validated at a later date
- software can be deployed on novel testbeds
- rapid recovery in case of hardware failure
Automated Testing

- unit tests for individual projects
- integration test for full deployment
Each component is tested in isolation within a Docker container.

Multi-VM Vagrant deployment running all the HARNESS cloud services, which interact with each other. Some of the VMs run services deployed within Docker containers.
Shortcomings

- difficult for non-experts to make configuration changes
- difficult to manage temporary branch changes for developers
- considerable development burden shifted to integration team
- running the full virtual deployment in vagrant on server takes a long time
Lessons Learned

- ability to refresh developer vms *extremely* useful
- automated deployment also very useful, but requires expert supervision
- testing results often ignored
Future Recommendations

- gatekeeper for authoritative versions
  - do not publish/merge changes until tests are passed!
- look into other techniques for projects with multiple repositories
- run integration tests against cloud back end
Discussion Points

request for feedback/discussion about a particular point in our work:

How can we better manage projects that pull from multiple repositories?

thought-provoking statement or discussion question about the area:

What are appropriate metrics of success for this type of project?