Continuous Delivery of Research Application in a Distributed Environment CODE-RADE

Sakhile Masoka (851667@students.wits.ac.za)
Witwatersrand University

May 4, 2015

Contents

1 Introduction **1**

2 Continuous Integration Review **2**

2.1 Single Source Repository 2

2.2 Automate the Software build 2

2.3 Automate Software Testing and Deployment 2

3 Continuous Integration Tools Review **3**

3.1 CernVM-FS Repository 3

3.2 Container - Docker 3

3.3 Jenkins 3

4 Conclusion **4**

Abstract

One of the aims of e-Infrastructure is to provide easy access to powerful computational and data platforms, to as many eligible users as possible. While user access is being simplified and made easy, the community of application developers and technical support in scientific collaborations does not yet have an easy way to integrate and manage applications. This gap has been identified and solutions to integrate applications into infrastructure in a fast, flexible, distributed and reproducible way have been developed by a few. This review presents some of the Continuous Integration principles and developed software tools based on CI, to help fill the gap.

1 Introduction

The South African National Grid (SAGrid) as part of the National Integrated Cyber-infrastructure System aims to provide access to powerful computational and data platforms, to as many eligible users as possible. While user access is being simplified greatly by the adoption of science gateways and identity federations, the community of application developers and technical support in scientific collaborations does not yet have an easy way to integrate these applications in the first place. SAGrid has identified this as a gap in the services it provides.

Using existing tools and services, there are simple set of tests which applications need to pass in order to be considered valid for the infrastructure. These can been encoded as automated tests using a Continuous Integration (CI) platform such as Jenkins. Interoperability between source code repositories, automated build systems, artifact creation and content delivery systems is crucial for the sustainability and uptake of the system. This is the aim of this literature review. Below is the description of Continuous integration.

[Fowler 2006] describes Continuous Integration (CI) *a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.* Mathias Meyer presented the basics of Continuous Integration, here summarized in a listed format, (1) All code is kept in a repository (2) Revised code is submitted daily to the repository (2) The automated system checks the code, runs tests and verifies that the code is good, it doesn't break anything [Meyer and CI 2014].

Finding literature for this review was difficult. There are publication for design principles, but not enough on the actual technology. The open source community tends to blog and upload slides instead of publishing papers. The rest of this paper is structured as follows, Section 2 reviews a set of principals that makes up Continuous Integration. Section 3 reviews the software tools used for a CI system followed by a conclusion.

2 Continuous Integration Review

2.1 Single Source Repository

Maintaining a single source repository is key for CI principles to work, especially when there are multiple people involved. [joa] states that repositories allow for a distributed and asynchronous development and these repositories should not only house artifacts, but also includes less formal parts relevant for the development process, such as organizational documents, sheets etc. To reduce cost, Fowler suggests Subversion, an open source version control system [sub 2000], while Meyer suggests a range of tools from self-hosted systems such as Jenkins, Teamcity and Bamboo to hosted products like CloudBees and Travis CI (an open-source hosted, distributed continuous integration service used to build and test projects hosted at GitHub).

Research done by [dab] found that four key features of visible feedback drove a rich set of inferences around commitment, work quality, community significance and personal relevance. These inferences supported collaboration, learning, and reputation management in the community. Their results informs the design of social media for large-scale collaboration, and imply a variety of ways that transparency can support innovation, knowledge sharing, and community building.

2.2 Automate the Software build

Getting the sources turned into a running system can often be a complicated process involving compilation, moving files around, loading schemas into the databases etc, however like most tasks in CI software development, this task can be automated [Fowler 2006]. A software *build* is an executable artifact of the knowledge base, that is ready for deployment into a productive setting [joa]. The artifact should be complete on its own, being transferred to production with all its dependencies, meaning the artifact should be platform independent or according to Fowler, the build should allow for alternative targets for different cases.

2.3 Automate Software Testing and Deployment

Testings should be automated on all software levels, individual components, integration, and on system level. A test is *automated*, if the expected and correct result of the test is known a priori and it can be applied without manual interaction. After its execution the computed results are compared with expected results automatically [joa]. The results should indicate if any of the tests failed, and if so, the whole test should cause the build to fail. The whole test suit should not require human interaction.

To ensure continuous integration, automating deployment becomes an important issue as well. Either deploying on test or production environment, automation speeds up the process and reduces errors [Fowler 2006]. Fowler also suggests building automated rollbacks when implementing automated deployment as test suites are not perfect and fast rollbacks reduces a lot of the tension of deployment, encouraging people to deploy more frequently and thus get new features out to users quickly.

3 Continuous Integration Tools Review

3.1 CernVM-FS Repository

CERN created cernVM, a lightweight Virtual Software Appliance intended to run on any operating system providing consistent and effortless installation of High Energy Physics experiment software to allow physicists working on these experiment to effectively use their desktops, laptop and Grid interfaces. The appliance runs a file system optimized for software distribution, CernVM-FS (CMVFS). [bun 2010]. CernVM File System (CernVM-FS) is a read-only file system widely used to access High Energy Physics experiment software and conditions data. Files and directories are hosted on cernVM server instances and mounted in the universal namespace **/cvmfs**. Grid sites compute nodes are able to mount the global namespace, providing the software needed to run experiments jobs locally without installing suite of software [blo 2012].

CERN softwares is used in over 140 Grid computing sites all over the world, size ranging from a couple of cores to thousands, and the experiment software needs to be running each compute node in order to fully utilize all available cores. Deployment and maintaining this software proved to be difficult. CernVM-FS solved this problem by allowing softwares to be installed centrally but when mounted on compute nodes behaves as a local file system [jak 2011].

3.2 Container - Docker

Docker is an open source platform for developers and systems administrators that automates deployment of applications inside containers called *Docker images*. The core technologies behind Docker are Namespaces, Control group (Cgroup) and union File System (UFS) [Lui and Zhao 2014]. With all these technologies combined, Docker is able create isolated workspace where application live independently, but still share the kernel with other spaces. This makes Docker more portable and efficient compared to Virtual machines.

Docker's features enables reproducible research by solving issues scientist face when reproducing another's research. Docker is able to solve "Dependency Hell", Imprecise documentation, Code rot and Barriers to adoption and reuse in existing solutions [Boettiger 2015]. Solving application dependencies is one of the main principles of Continuous Integration as applications need to be self-contained when deployed to different platforms.

3.3 Jenkins

Wikipedia describes Jenkins as an open source continuous integration tool for software development [wik]. From the official website, Jenkins is described as application that monitors executions of repeated jobs, such as building a software project or jobs run by cron [jen 2015]. The focus of Jenkins is (1) Building/testing software projects continuously (2) Monitoring executions of externally-run jobs. These two focus points are principles of continuous integration.

4 Conclusion

Continuous Integration principals were presented, namely (1) single source repository for code (2) automation of software build (3) automation of testing and deployment. This led to the software tools currently available to enforce these principles in software development. These tools will be used in linking the gap identified by SAGrid and fully be able to provide easy access to as many eligible users as possible to powerful computational and data platforms.

References

[blo 2012] *Status and future perspectives of CernVM-FS*, volume 396. IOP Publishing, 2012. International Conference on Computing in High Energy and Nuclear Physics 2012.

[Boettiger 2015] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49:71–79, 2015.

[bun 2010] *CernVM a virtual software appliance for LHC applications*, volume 219. IOP Publishing, 2010. 17th International Conference on Computing in High Energy and Nuclear Physics.

[dab] *Social coding in GitHub: transparency and collaboration in an open software repository*. ACM, New York. Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work.

[Fowler 2006] Martin Fowler. *Continuous Integration*. `http://www.martinfowler.com/articles/continuousIntegration.html`, 2006. [Online; accessed 2015-04-25].

[jak 2011] *CernVM-FS: delivering scientific software to globally distributed computing resources*. ACM, 2011. proceedings in The International Conference for High Performance Computing, Networking, Storage, and Analysis.

[jen 2015] *Jenkins Continuous Integration*. `http://jenkins-ci.org`, 2015. [Online; accessed 2015-04-25].

[joa] *Developing knowledge systems with continuous integration*. ACM, New York. Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies.

[Lui and Zhao 2014] Di Lui and Libin Zhao. The research and implementation of cloud computing platform based on docker. pages 475–478, 2014.

[Meyer and CI 2014] Mathias Meyer and Travis CI. Continuous integration and its tools. *IEEE Software*, 31:14–16, 2014.

[sub 2000] *Subversion*. `http://subversion.apache.org/`, 2000. [Online; accessed 2015-05-01].

[wik] *Jenkins*. `http://en.wikipedia.org/wiki/Jenkins_(software)`. [Online; accessed 2015-04-25].