LibrumCI: Leveraging Container Cluster Management Framework Natives in Continuous Integration

Anthony Troy
School of Computing
Dublin City University
Dublin 9, Ireland
anthony.troy3@mail.dcu.ie

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Keywords

Continuous Integration; Kubernetes; Continuous Delivery; Docker; Cluster Management Frameworks; VCS; Agile

1. INTRODUCTION

With organisational structures diversifying to off-shore, insource and outsource development needs, software is increasingly being developed by fully- or partially-distributed teams. The contributions from team members must, at a minimum, be first validated under a test infrastructure that is automated and robust enough to allow for high levels of code churn. This development requirement is reflected in in the sustained and growing adoption of Continuous Integration (CI) practices in open-source and enterprise software projects (Duvall et al., 2007; Fitzgerald and Stol, 2014; Vasilescu et al., 2015).

Todo, mention the requirements and then pitfalls/challenges of CI (expense of provisioning prod-like envs, configuration management, slow feedback). Todo, mention the advent of containerisation and container cluster management frameworks. Todo, mention the lacking CI open-source tooling support for containers. Todo, define container cluster management frameworks and mention how they can remedy CI challenges, CI container support and more.

2. RELATED WORK

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3. BACKGROUND

Traditionally most software projects have been characterised as having poor integration practices. More often than often not a siloed team will conduct all development activities, arguably over a lengthy period, and defer acceptance testing and code integration. Importantly, during this period one can assert that the software is in an inoperative state, as there has been no motivation to actually run the software and use it in a production-like environment.

Humble and Farley (2010) argue that many projects "schedule lengthy integration phases at the end of development to allow the development team time to get the branches merged and the application working so it can be acceptance-tested". More concerning is that when some projects arrive at this point "their software is not fit for purpose". These long-winded and challenging phases, commonly referred to as "integration hell", pose difficulties for the engineers working through the integration and too for the leads and managers estimating the project delivery.

Applications of Continuous Integration (CI) practices have been found to effectively remove these phases at source (Vasilescu et al., 2015; Fitzgerald and Stol, 2014; Humble and Farley, 2010; Duvall et al., 2007). As a practice, CI seeks to reduce software deployment lead time by facilitating frequent code churn through verifying contributions with automated builds and tests. Depending on the structure of the team and software, Humble and Farley (2010) advise that a team member should integrate their work at least once daily. An effective undertaking of this process means that the software is always in a working state

Todo, highlight of challenges applying CI (provisioning build farms with things like Puppet, compute expense, configuration management, maintaining production parody). Todo, highlight the advent of containerisation (effective smart building, prod build and config parody can be more easily achieved). Todo, briefly discuss how distributed container technology works (distributed caching of layers and images across nodes) Todo, discuss production-grade container scheduling and management frameworks vs traditionally build farm technology (Kuberentes vs Jenkins).

3.1 Continuous Integration

Practices resembling Continuous Integration could be said to stem as far back as the original lean manufacturing method. Nevertheless in software engineering the practice was codified by Beck (2000) as a core tenant of the Extreme Programming (XP) method. Following, CI was also included in the Agile Manifesto as an encouraged quality assurance practice. Amidst its wide adoption and popularity however, there is some variability in how teams are employing the practice, resulting is some debate as to exactly which integration activities one should follow (Ståhl and Bosch, 2014).

Nevertheless, the practice forwards several core principles and high-level activities. Todo, mention these practices (use version control, daily committing to mainline, etc) Todo, detail the challenges

3.2 Containerisation

Todo, tie this section with the previous (CIs requirements of producing production-like environments effectively and efficiently). Containerisation is a recently resurged computing paradigm that is having a significant impact on how applications are being built, shipped and ran. Along with being less resource intensive and more portable, containers simplify dependency management, application versioning and service scaling, as opposed to deploying applications or application components directly onto a host operating system. Docker, albeit a relatively young project, has successfully established a container standard.

Containers have a long history in computing though much of their recent popularity surround the recent developments of both LXC and the Docker platform. The former can be described as a container execution environment, or more formally, a Linux user space interface to access new kernel capabilities of achieving process isolation through namespaces and cgroups (Pahl and Lee, 2015). The latter is an open-source suite of tools managed by Docker Inc. which extends upon container technology such as LXC, in turn allowing containers to behave like "full-blown hosts in their own right" whereby containers have "strong isolation, their own network and storage stacks, as well as resource management capabilities to allow friendly co-existence of multiple containers on a host" (Turnbull, 2015).

Uncertainties around Docker's maturity and production-readiness have been expressed (Kereki, 2015; Powers, 2015; Merkel, 2014), however over the last two years the states of both Docker and the containerisation ecosystem continue to rapidly progress. Last year Docker has seen an unprecedented increase in development, adoption and community uptake (Merkel, 2014). Most notably was the introduction of customisable container execution environments. This means as opposed to LXC one can "take advantage of the numerous isolation tools available" such as "OpenVZ, systemd-nspawn, libvirt-sandbox, qemu/kvm, BSD Jails and Solaris Zones". Also included in this 0.9 release was the new built-in container execution driver "libcontainer", which replaced LXC as the default driver. Going forward on all platforms Docker can now execute kernel features such as "namespaces, control groups, capabilities, apparmor profiles, network interfaces and firewalling rules" predictably "without depending on LXC" as an external dependency (Hykes, 2014).

Interestingly, libcontainer itself was the first project to provide a standard interface for making containers and managing their lifecycle. Subsequently the Docker CEO announced the coming together of industry leaders and others in partnership with the Linux Foundation to form a "minimalist, non-profit, openly governed project" named The Open Container Initiative (OCI), with the purpose of defining "common specifications around container format and runtime" (Golub, 2015). Thereafter Docker donated its base container format and runtime, libcontainer, to be maintained by the OCI.

Amidst establishing a container standard, Docker has made significant headway in supporting multi-host cloud production environments. In terms of native tooling, in the last year Docker has implemented a suite of tools for provisioning and orchestrating containers:

- **Docker Machine** allows one to provision Docker hosts, which are simply Linux virtual machines (VMs) supporting Docker, on a local machine or cloud. Its pluggable driver API currently supports "provisioning Docker locally with Virtualbox as well as remotely" on cloud providers such Digital Ocean, AWS, Azure and VMware.
- **Docker Swarm** is a clustering solution which takes the standard "Docker Engine and extends it to enable you to work on a cluster of containers". This in turn allows one to "manage a resource pool of Docker hosts and schedule containers to run transparently on top, automatically managing workload and providing failover services".
- **Docker Compose** is the "glue" allowing one to compose a multi-host application on top of a Swarm cluster whereby you can specify how each application is to be ran in the the cluster, in turn allowing one to orchestrate and choreograph local or cloud containers.

In many cases an existing cloud infrastructure depends upon one or more orchestration tools, for example Consul for service discovery. Typically, such tools cannot be migrated away from easily and in turn cause "vendor lock-in". Consequently, Docker have implemented this trio of orchestration tools in a generic way, providing "a standard interface to service providers so that they can almost be used as plug-and-play solutions" on top of the Docker platform (Holla, 2015).

3.3 Container Cluster Management

Todo, introduction to tie in CI and Docker. Todo, reference pains of provisioning CI 'production-like' clusters. Practitioners and industry experts note that cluster management tooling supporting Docker vary greatly in terms of capability, architecture and target cluster proportion (Goasguen, 2015; Holla, 2015). This is unsurprising when we consider that all infrastructures are not subject to same orchestration requirements and software release cycles. For instance, slow moving infrastructures can be characterised as having infrequent application deployments, hard-coded service configurations and rare service failures which may not have an urgent impact. In contrast, more fast moving infrastructures feature continuous deployments and strong automation in terms of service configuration and recovery.

3.3.1 Service orchestration

Central to cloud cluster management is the ability to elastically provision and tear down clusters. Many cloud providers have introduced their own service orchestration tools such as CloudFormation from AWS and Heat by OpenStack (Dudouet et al., 2015). On a high-level, these tools simply define a cluster template which can be later orchestrated with possibly extended configurations. As previously mentioned, the native Docker orchestration tools support similar features that can clusterise multi-host containers. Docker Compose conceptually defines a similar template to that of Amazon's CloudFormation and allows one to perform orchestration tasks such as provisioning, destroying and scaling on per container basis.

Pahl and Lee (2015) describe container-based clusters as consisting of several hosts which are "virtual servers on hypervisors or possibly bare-metal servers", each of which typically runs several containers that are responsible for scheduling, load balancing and serving an application or service. Meaning containers can be distributed across one or more host machines wherein these hosts might be virtual servers running other services that must also be orchestrated.

Slow moving infrastructures may not be availing of their provider's orchestration tools as doing so is simply not required. Clusters themselves are manually defined once and the scaling of nodes can be introduced during deployments or at scheduled downtime. Nevertheless, Docker Compose supports this manual workflow. Conversely, fast moving infrastructures profit from their provider's orchestration tools, leveraging them to automate tasks around cluster management. As discussed previously, Swarm is a native Docker clustering tool for containers which pools Docker engines together into a single virtual host. In conjunction with Docker Compose, it facilitates for transparent orchestration across container clusters. (Holla, 2015).

Cluster management frameworks aim to abstract and automate service orchestration activities such as provisioning, scaling, task scheduling, resource utilisation management and failover recovery. Some cloud providers have implemented such frameworks which sit on top of Swarm. For example, Amazon's EC2 Container Service (ECS) is one that uses a shared-state scheduling model to execute tasks on containerised EC2 instances via containers. Each host instance has a preinstalled ECS agent which allows clusterised containers communicate together and with the ECS console. Consequently, via scheduled tasks, ECS clusters can be transparently and dynamically orchestrated.

Stand-alone Swarm or ECS may be fitting orchestration solutions for fast moving infrastructures, however larger-scale clouds that host hundreds or thousands of containers require high-level cluster management platforms such as Apache Mesos and Kubernetes. The former abstracts "distributed hardware resources into a single pool of resources" and can provide similar cluster management facilities to ECS when integrated with scheduling and service management tools such as Marathon. The later is a higher-level platform specifically designed for managing containerised applications across multiple hosts including mechanisms for service deployment, scaling and maintenance.

3.3.2 Service discovery and configuration

Service discovery and configuration management are central cluster management concepts in distributed systems and microservices-based architectures. Both of which are argued to overlap in nature. Service discovery can be described as an approach to achieve "dynamic and automatic software system composition, configuration and adaptation" (Yang et al., 2006). Generally, service discovery implementations accomplish this by allowing application components/services discover information or configurations about their current and neighbouring environments through a distributed key-value store.

Whether operating under a fast or slow moving infrastructure, requiring a service discovery solution is generally related to having a service-orientated architecture style. The more distributed a system becomes, the more regularly do services require information about their own and neighbouring environments. The tooling around service discovery ranges in terms of complexity and provided features. DNS (Domain Name Systems) is a well-known and commonly understood standard which allows us "associate a name with the IP address of one or machines" where the name becomes an "entry point to the IP address of the host running that service" (Newman, 2015). More advanced tools like Consul and Apache Zookeeper support both configuration management and service discovery. The former is designed specifically for service discovery and can use service health checking features to route traffic away from unhealthy nodes. The later is used for wider variety of cases such "configuration management, synchronizing data between services, leader election, message queues and as a naming service" (Newman, 2015).

Container-based service discovery involves the ability to dynamically register and discover multi-host containers among their peers. Holla (2015) poses two techniques to accomplish discovery in Docker; integrating Swarm backend discovery tools or using default Docker features like names and links. Docker Swarm implements a hosted discovery service which uses generated tokens to discover cluster nodes. Being primarily concerned with orchestration, Swarm does not currently support dynamic service registration and configuration. However, to dynamically configure and manage the services in your containers one can use a discovery backend with Swarm such as Etcd, Consul or Zookeeper.

As previously highlighted, Docker Compose provides a mechanism to link named containers on the same host. This is accomplished by "inserting the first container's IP address in /etc/hosts when starting the second container". Importantly, the IP address of a container living on a different host "is not known by the docker daemon running in the current host". The ambassador container pattern achieves cross-host container linking between provider and consumer containers by dynamically configuring network connections through respective intermediate ambassador containers (Holla, 2015).

3.3.3 Kubernetes

Todo, mention project history. Todo, mention the cluster topology it forwards. Todo, feature run-down. Todo, mention some examples of applications of kubernetes as a build farm and/or in testing. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4. EVALUATION

Todo, describe Travis and Jenkins in some detail. Todo, note their poor container/Docker support. Todo, matrix of comparison. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5. CONCLUSIONS

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6. REFERENCES

P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007. [Online]. Available: http://www.amazon.com/Continuous-Integration-Improving-Software-Reducing/dp/0321336380%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321336380

B. Fitzgerald and K.-J. Stol, “Continuous software engineering and beyond: Trends and challenges,” in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE 2014. New York, NY, USA: ACM, 2014, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/2593812.2593813

B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in github,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 805–816. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786850

J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional, 2010. [Online]. Available: http://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321601912

K. Beck, *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

D. Ståhl and J. Bosch, “Modeling continuous integration practice differences in industry software development,” *J. Syst. Softw.*, vol. 87, pp. 48–59, Jan. 2014. [Online]. Available: http://dx.doi.org.dcu.idm.oclc.org/10.1016/j.jss.2013.08.032

C. Pahl and B. Lee, “Containers and clusters for edge cloud architectures - a technology review,” in *3rd International Conference on Future Internet of Things and Cloud)*, ser. FiCloud-2015. IEEE, 2015.

J. Turnbull, *The Docker Book: Containerization is the new virtualization*, 1st ed. Amazon Digital Services, Inc., 2015.

F. Kereki, “Concerning containers’ connections: On docker networking,” *Linux J.*, vol. 2015, no. 254, Jun. 2015.

S. Powers, “The open-source classroom: Doing stuff with docker,” *Linux J.*, vol. 2015, no. 254, Jun. 2015.

D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, Mar. 2014.

S. Hykes, “Docker 0.9: Introducing execution drivers and libcontainer,” http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/, 2014.

B. Golub, “Docker and broad industry coalition unite to create open container project,” http://blog.docker.com/2015/06/open-container-project-foundation/, 2015.

S. Holla, *Orchestrating Docker : manage and deploy Docker services to containerize applications efficiently.* Birmingham: Packt Publishing, 2015.

S. Goasguen, *Docker Cookbook.* O’ Reilly & Associates Inc, 2015.

F. Dudouet, A. Edmonds, and M. Erne, “Reliable cloud-applications: An implementation through service orchestration,” in *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, ser. AIMC ’15. New York, NY, USA: ACM, 2015, pp. 1–6.

K. Yang, C. Todd, and S. Ou, “Model-based service discovery for future generation mobile systems,” in *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*, ser. IWCMC ’06. New York, NY, USA: ACM, 2006, pp. 973–978.

S. Newman, *Building Microservices.* O’Reilly Media, 2015.

E. A. Brewer, “Kubernetes and the path to cloud native,” in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC ’15. ACM, 2015, pp. 167–167.

A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at google with borg,” in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys ’15. ACM, 2015, pp. 18:1–18:17.