Global Information Assurance Certification Paper

Continuous Security: Implementing the Critical Controls in a DevOps Environment

GIAC GCCC Gold Certification, MSISM
Author: Alyssa Robinson, lyssanr@yahoo.com
Advisor: Stephen Northcutt
Accepted: 12/20/2016

Abstract

DevOps is an agile-aligned software development methodology that is growing quickly in popularity, expected to reach nearly 25% of Global 2000 organizations (Gartner, 2015) by 2016. Adoption of DevOps practices introduces complications for implementing and auditing standardized security controls, presenting issues such as constantly changing assets, continuous deployment and a breakdown in the traditional segregation of duties. DevOps tools and philosophies also provide advantages, providing opportunity for integration of security automation as part of the development and deployment of applications and giving Security early input into design and implementation.

1.0 Introduction

Since the first DevOps Days conference was held in 2009, adoption of DevOps strategies has been growing rapidly, with 25% of global IT companies predicted to have moved towards DevOps by 2016 (Gartner, 2015). The very definition of DevOps is still evolving, but most agree it encompasses a set of cultural values in addition to the tools and practices that enable continuous delivery (Loukides, 2015). Continuous delivery provides a competitive advantage to software companies (Humble, 2014) by lowering the risk and cost associated with releases. It also enables near-immediate feedback on new features; practicing continuous delivery requires collaboration and empathy amongst the teams involved in the delivery process (Fowler, 2013).

The changes in roles, processes, and tools that accompany DevOps can in some cases vary quite widely from the guidelines recommended by the 20 Critical Controls or other control frameworks. The changes affect the implementation of some controls far more than others: methods for keeping an accurate inventory of authorized devices in a

public cloud environment, with automated scale-up of resources are extremely different from those used to track hardware inventory in a physical datacenter; in both DevOps and traditional development organizations, however, the same methods can be used to develop a reasonable training program for security skills. While the DevOps focus on automation and continuous improvement can have a positive impact on control adoption (DeLuccia, Gallimore, et al., 2015), the loss of visibility and control in certain elements of the infrastructure that comes with public cloud environments can be a hindrance (Netwirx, 2015) or necessitate the introduction of compensating controls.

2.0 What is DevOps?

Some define DevOps as a cultural movement, others as a set of tools or technology practices. In reality it is both; the tools and practices influence cultural change in an organization and vice-versa (Davis & Daniels, 2015). DevOps has its roots in the ideas of Lean Manufacturing: driving out waste in the form of features that never get used, unnecessary processes and bad quality software (Davis & Daniels, 2015). DevOps culture focuses on bringing development and operations teams together, improving trust and removing barriers to getting useful features delivered to customers (DevOps Enterprise Forum 2015). Damon Edwards, co-host of the DevOps Café podcast series, explains DevOps as "removing the bottlenecks, conflicts, and risk from the lifecycle between business decision and customer outcome." One core tenet of DevOps involves giving teams that develop code the capability to deploy it when it is ready and the responsibility for supporting it in production; this both increases deployment speed and provides immediate feedback that promotes the building of more resilient services (Pais, 2015). According to ThoughtWorks consultant Chris Hilton, "The most valuable thing you can do as a software developer is make that connection between creation and use."

DevOps practices focus on building quality into the code, on automated testing and on a culture of continuous improvement that leads to improved stability and throughput. Using continuous deployment techniques, DevOps practitioners have found a way to improve both speed and stability in the systems they support (Puppet Labs, 2015). The 2015 Puppet Labs State of DevOps Report shows high performing DevOps organizations achieving 30x more frequent deployments, with a 200x increase in speed from code commit to deploy and 60% fewer production failures. Bank of America cites a

6x reduction in production defects after moving to a DevOps deployment model, while Ticketmaster has reduced their Mean Time to Repair by 90% (Pais, 2015).

DevOps focuses on quickly moving new features out to the customers, where valuable insights about the changes can be gained (Davis & Daniels, 2015). Even though DevOps is not about specific tools, the push to reduce this cycle time does drive the adoption of many common toolsets. Version control systems track changes to files and allow collaboration between teams, facilitating comparisons and merging between versions, and rollback in the case of issues (Davis & Daniels, 2015). Configuration management systems automate the provisioning of new systems, enforcing consistent application installation, system and application configuration across classes of servers. The configuration information lives in a source code repository, and systems such as Chef, Puppet, Salt, or Ansible allow developers to treat the configuration of the servers that will run application software as code. This "infrastructure as code" can itself be versioned and tested, providing assurances that identical configurations will be in place everywhere, and improving the odds that software that tested fine in the staging system will be fine in production as well (Riley, 2014). Finally, an automated system for reliably moving software through the build -> deploy -> test -> release process is the key component (Humble & Farley, 2010) in any DevOps system. Continuous integration tools such as Jenkins make a formerly slow and error-prone task easy and repeatable, enabling the deployment of small changes and giving fast feedback about how the code operates and what customers think about new features.

2.1 DevOps Complications

Adopting DevOps practices can create challenges when implementing standardized security control infrastructures, particularly when facing audit of a security program (DeLuccia, Gallimore, et al., 2015). Many of even the most popular configuration management and continuous integration tools are brand new to the market or are open-sourced. The relative immaturity and lack of corporate backing lead to concerns about how meticulously the developers have adhered to secure development standards or how quickly patches will be released when a security flaw is found (Vadalasetty, 2003). Cloud deployment is the most common option for DevOps practitioners (Linthicum, 2014), which often means relying on an external Infrastructure

as a Service (IaaS) or Platform as a Service (PaaS) provider. This reliance reduces control and visibility at the hardware and network layers; the flexibility of Cloud providers to quickly scale up and down that make them attractive in DevOps environments may also complicate the tracking of hardware assets over time (Minjar, 2015). DevOps by design blurs lines between developer and operator of an application, which can lead to questions regarding segregation of duties if proper checkpoints aren't introduced to limit a particular developer's end-to-end control over the system (DeLucia, Duval, et al, 2015).

2.2 DevOps Advantages

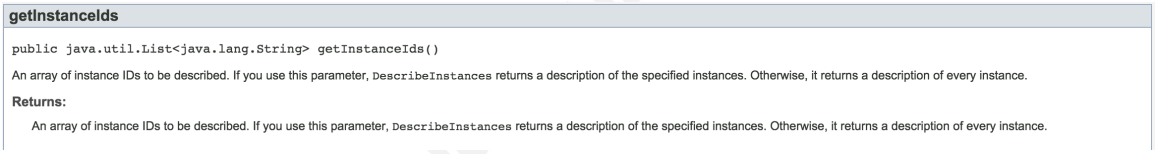
Even with these challenges, the DevOps movement has several major advantages in improving software security. The DevOps focus on fast deployment, continual improvement, and automation naturally forces collaboration with security teams; without this collaboration, a potential deployment barrier exists in the form of last-minute manual audits and reviews (Davis & Daniels, 2015). Security teams must be engaged early in the design process to ensure the ability to deploy continuously (Bellis, 2015). Reducing these barriers to deployment makes it easy to deploy small changes rather than bundling a group of changes together. This practice reduces risk by making it easier to test changes completely and easier to pinpoint any issues and fix or roll back quickly (Bellis, 2015). Configuration management systems provide a mechanism to standardize configurations for logging, alerting and security metrics that are needed to determine when a security incident is in progress, (Davis & Daniels, 2015) and to enforce policies around secure configuration.

3.0 CC1: Inventory of Authorized and Unauthorized Devices

Perhaps the most obvious deviation from the Critical Controls in a DevOps environment is seen in Critical Control One, the Inventory of Authorized and Unauthorized Devices. In the traditional IT environment, it is easy to create a baseline of authorized devices by listing all physical hardware and cross-reference it with a scan of the local address space that might turn up a long-forgotten system in a closet or hidden under a desk. In a DevOps environment, however, it is far more likely that managed systems are run by an IaaS provider in the cloud, or that the very idea of a "device" is obscured by layers of virtual machines and deployed containers. There may be no

contiguous set of IPs to scan and find lost servers, or doing so may violate terms of service of a cloud provider (Amazon, 2011).

Automated deployment of both software and infrastructure allows for automated scale-up and scale-down of deployments in response to changing demand (Newman, 2015), rollout of new software, or changed infrastructure configurations (Humble & Farley, 2010). This practice introduces time as an important element in the asset inventory, relevant both to understanding what systems need to be monitored and updated at the current point in time and later to audit the state of the system at any given point in the past. Deployment automation makes it simple to meet the "quick win" of automatically updating the inventory system (Cole & Terala, 2015) in the common case; the configuration management system or service discovery mechanism may even serve as the asset inventory. Independent verification or control using mechanisms like scanning, DHCP logging or NAC/802,1x (Cole & Terala 2015), however, are impossible in situations where the network is controlled by a third party.



getInstanceIds

```
public java.util.List<java.lang.String> getInstanceIds()
```

An array of instance IDs to be described. If you use this parameter, `DescribeInstances` returns a description of the specified instances. Otherwise, it returns a description of every instance.

Returns:

An array of instance IDs to be described. If you use this parameter, `DescribeInstances` returns a description of the specified instances. Otherwise, it returns a description of every instance.

Figure 1: Amazon API for polling all instances

Since virtual machines or containers may still be spun up outside of the automated system, or glitches in automation may leave a VM in a state that is untracked but running, the system must still be verified. Cloud provider portals, invoices and APIs can all provide independent verification of the automated inventory (Barr, 2014) and asset tracking software providers like Qualys have begun to provide support for tracking cloud assets as well.

```
{
    "servers": [
        {
            "id": "616fb98f-46ca-475e-917e-2563e5a8cd19",
            "links": [
                {
                    "href": "http://openstack.example.com/v2/openstack/servers/616fb98f-46ca-475e-917e-25
63e5a8cd19",
                    "rel": "self"
                },
                {
                    "href": "http://openstack.example.com/openstack/servers/616fb98f-46ca-475e-917e-2563e
5a8cd19",
                    "rel": "bookmark"
                }
            ],
            "name": "new-server-test"
        }
    ]
}
```


Figure 2: Openstack API server instance response

CC2: Inventory of Authorized and Unauthorized Software

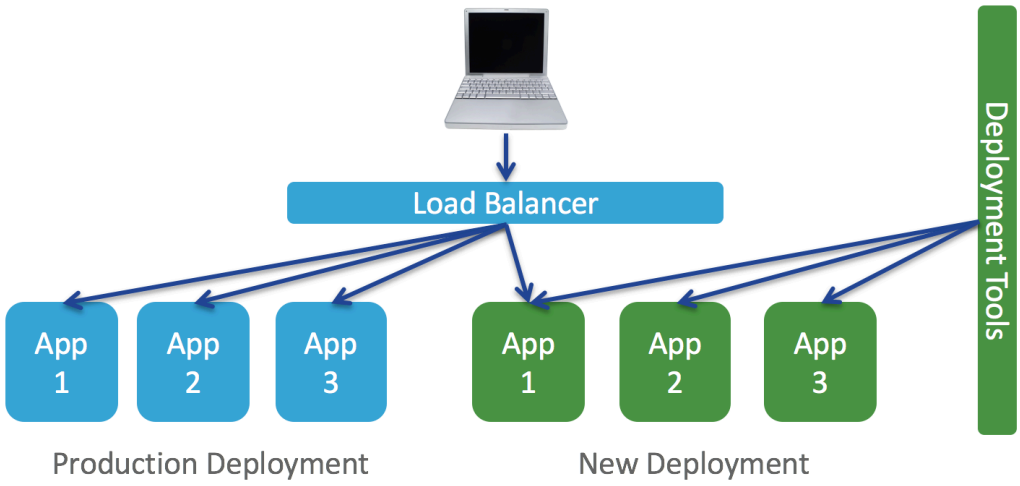


Figure 3: Blue-Green Deployment

Automated deployment of infrastructure and software is associated with several practices that complicate tracking of authorized software across the application servers. In a blue-green deployment scenario, a second sent of applications (the green deployment) is brought up in parallel to the original deployment (the blue deployment) to allow for testing and cutover with no downtime. In some cases, only a subset of traffic may move to the green deployment, to allow comparison of the two running sets of software in production, looking for cost increases or performance changes (Humble &

Farley, 2011) or to allow a quick rollback to the previous version. Once traffic is fully cut over, the blue deployment gets decommissioned. The same scenario might be used to roll out changes to "infrastructure as code", treating server configuration or database schema the same way application code gets treated (Fowler, 2010). Similarly, feature flags may be used to turn new features on for particular sets of users and allow analysis of system or user behavior (Humble & Farley, 2011). These practices mean that multiple versions of software – either developed by a third party or in-house – will likely be in use at a time and must be tracked and considered through each phase of the deployment lifecycle. This includes versions of images for the virtual machines or containers running the application software (Minjar, 2015).

Common DevOps practices offer advantages in limiting servers to an approved list of installed software. Many hosting providers allow upload of a custom image file or container image, which gives an agreed-upon hardened baseline, with only whitelisted software packages installed. From there, configuration management tools can be used to configure access only to approved software repositories, providing another control point. Configuration management tools can also restrict software packages to specific required or excluded versions, providing an ability to block versions with known vulnerabilities.

CC3: Secure Configurations for Hardware and Software (on Servers)

Default accounts, unnecessary services and unpatched software all provide attackers with easy pathways to gain control of their targeted systems (Terala & Cole, 2015). Installing and running only the required set of software and services and keeping that software up to date and configured according to best practices minimizes the available attack surface (Terala & Cole, 2015). While many DevOps environments run in private or public cloud environments and don't give the option of hardware shipped with a hardened configuration, custom hardened system images can be used in many cases to spin up new virtual machines or containers. The Center for Internet Security (CIS), which currently maintains the Critical Controls, offers images configured according to

their security benchmarks for multiple operating systems on the AWS Elastic Compute Cloud.

Once hardened configurations for operating systems and application components are developed, DevOps deployment tools and configuration management services like Puppet, Chef, Ansible and Salt greatly simplify the process of rolling these out to all systems and keeping the configurations in sync over time. Configuration management tools run periodically and provide assurance that security configurations have not changed (Terala & Cole, 2015), alerting or automatically return a system to a known good state if configurations get changed manually. Any updates to the "infrastructure as code" security configuration will undergo peer review and can kick off automated application scans or SCAP checks (Terala & Cole, 2015) prior to deployment, triggering an approval process before security-relevant changes make it to production (DeLuccia, Gallimore, et al, 2015). Puppet offers modules for implementing many of the CIS Benchmark hardening elements for Red Hat operating systems and Amazon Red Hat AMIs, while Chef offers recipes that will audit, though not automate, enforcement of the CIS Benchmarks (Timberman, 2015).

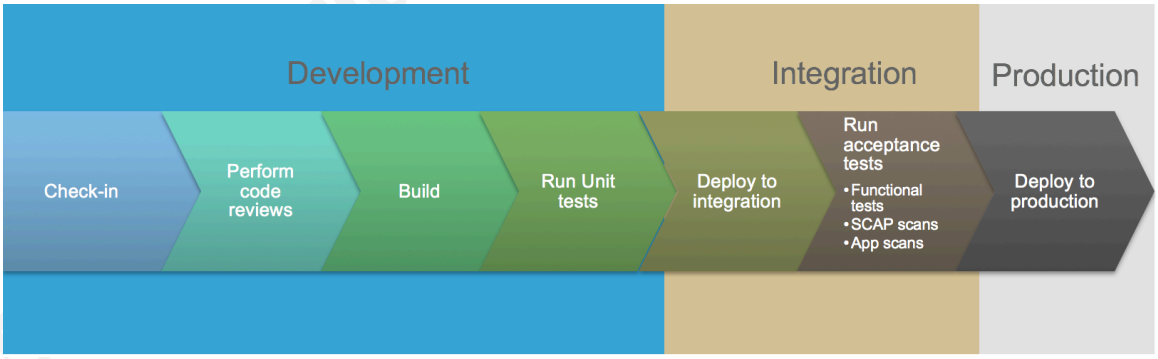


Figure 4 Security Testing and Continuous Deployment

Docker and other container technologies are increasingly popular methods for deploying applications in DevOps environments, due to advantages in portability, efficiency in resource sharing and speed of deployment (Suleman, 2015). Docker also offers some security advantages, in the form of increased isolation of applications, particularly in multi-tenant environments (Zeltser, 2015). Docker images, however, cannot be patched and updated or have

running configuration changed on the fly; updated software or secure configuration must be baked in as part of the image build and new containers deployed, (Doran, 2015) leading to situations where multiple container versions of varying security may be running (Zeltser, 2015). While publicly-available container images make it easy to test out new technologies, these images may contain outdated software or other vulnerabilities. As Docker gains popularity, however, tools are becoming available to scan Docker images (Doran, 2015) and recently the Center for Internet Security has even provided a Docker security benchmark (Petazzoni, 2015).

CC4: Continuous Vulnerability Assessment and Remediation

Keeping up with the relentless pace of newly-announced vulnerabilities is a challenge across traditional and DevOps systems alike. Once again, however, the focus on automation, testing, and continuous monitoring in DevOps environments can be advantageous; the same systems that allow automated deployments of new application code via thorough unit and functional testing provide a strong foundation for testing new patches. Deployment strategies for Blue-Green deployments and A-B testing allow gradual rollout and immediate feedback regarding issues and changes in system behavior. Security scans that happen as part of the deployment process provide verification that updates address known issues and reach all intended targets.

Another security advantage for container systems like Docker is that application containers are treated as transient; rather than worrying about patching individual systems, with all of the change history and possible differences they contain, container images can be patched as part of the application build and identical containers are tested and deployed to replace the old (Zeltser, 2015). A similar (but heavier-weight) strategy could of course be used with virtual machines, rolling out newly patched image files by scaling the system up using the new images and then removing the older, un-patched systems when scaling down (Minjar, 2015).

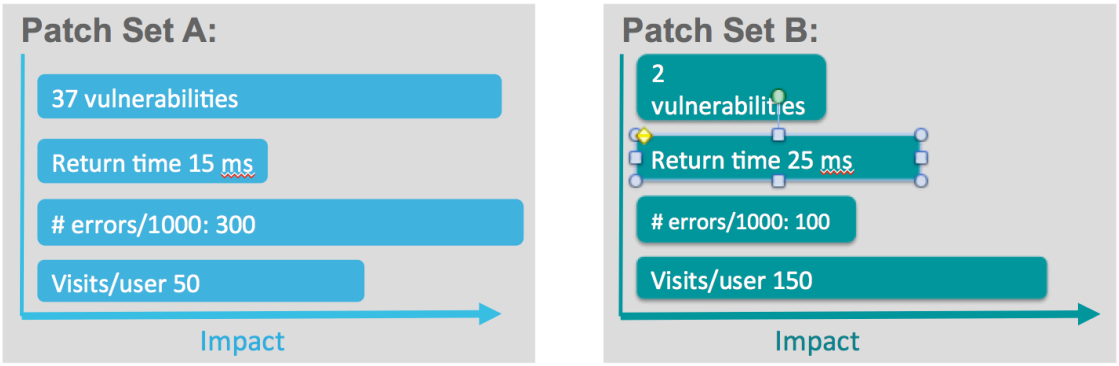


Figure 5 A/B Testing of Patch Set Deployments

CSC6: Application Software Security

In the 2015 Trustwave Global Security Report, vulnerabilities were found in 98% of the applications scanned with the Trustwave App Scanner product. These included issues with data leakage, cross-site scripting, SQL injection and authorization, among others. Efficiently preventing these vulnerabilities from making it to production is essential for success in a DevOps environment. The automated deployment pipeline provides a mechanism to require that defenses like code review, static analysis and web application scanning (Cole & Terala, 2015) are performed before new software is moved to production.

Class	Fail	Error	Skip	Success	Total
[redacted]tests.unit_tests.datacenterproviders.test_openstack_network_provider.TestOpenstackBase	0	0	0	4	4
[redacted]tests.unit_tests.servertype.test_[redacted]n_servertype.Test[redacted]nServertype	0	0	0	3	3
[redacted]tests.unit_tests.test_volumemeta.TestVolumemeta	0	0	0	3	3
[redacted]tests.unit_tests.service.test_[redacted]_service.Test[redacted]Service	0	0	0	9	9
[redacted]tests.unit_tests.datacenterproviders.test_openstack_network_provider.TestNetworkProvider	0	0	0	12	12
[redacted]tests.unit_tests.servertype.test_repos_servertype.TestReposServertype	0	0	0	3	3

Figure 6 Jenkins-integrated static analysis

Jenkins, Hudson, and most other popular deployment tools provide easy support via plugins both for requiring code review and for running static analysis as part of the pipeline. These acceptance tests should be designed to complete quickly and can be run before new code is even deployed to the integration/staging environment (Humble & Farley, 2011). Further security testing, such as tests of security-related functionality, vulnerability scanning, and application security scans can then be run in parallel to other acceptance testing within the staging environment (De Vries, 2015). Keeping the production deployment infrastructure separate from the infrastructure that pushes code to non-production envionments like load or staging

can help to provide the access needed to automate and debug while enforcing segregation of duties and limited access to the production environment (Smith, 2014).

CSC12: Controlled Use of Administrative Privileges

In the DevOps model, where everyone has the potential to administer systems and debug production issues, controlling administrative credentials becomes even more important. In a continuous deployment, "infrastructure as code" environment, the code itself acts as a privileged user (Lawler, 2015); administrative privileges will be used by the configuration management and orchestration systems that spin up new servers, install software and make configuration changes in response to events and alerts in the environment (Andre, 2015). These credential "secrets" must be used by the orchestration systems, but not available to anyone with access to the code repositories, enabling untraceable administrative actions and greatly increasing the potential for exploitation.

Secrets Management systems aim to role-based access control and auditability to the DevOps system (Lawler, 2015), while preserving the automation and lack of human intervention that is key to continuous deployment (Gilman, 2015). Configuration management systems like Chef and Puppet provide their own solutions for protecting secrets stored within the infrastructure code using public-key encryption. While this meets the goal of hashing or encrypting stored passwords (Cole & Terala, 2015), they lack more advanced features such as role-based controlled access to the secrets, or full featured support for rotating passwords and SSH keys. – for example, not every server managed by Puppet needs to access the PII database – In the idealized DevOps environment, no individual developer or administrator should be SSHing into a system at all (Parsons, 2014); a popular quote from Piston Cloud CEO Joshua McKenty reads "OpenStack is a system for managing your servers like cattle -- you number them, and when they get sick and you have to shoot them in the head, the herd can keep moving. It takes a family of three to care for a single puppy, but a few cowboys can drive tens of thousands of cows over great distances, all while drinking whiskey." Following this model, problematic application

servers are simply killed off in response to an alert and the orchestration engine spins up servers to replace the lost capacity. If this were always the case, the goal of minimizing administrative privileges (Cole & Terala, 2015) would be simple to achieve. The reality, however, is rarely that simple, and there will be times that developers and sysadmins need access to determine root cause for a recurring issue. Secrets management systems like Hashicorp's Vault and Conjur's SSH Management solution provide methods to automatically provision temporary access via one-time passwords or SSH keys and to enable SSH key rotation for service accounts. When an application server needs direct access for debugging, orchestration services can quarantine the problematic system, removing it from the set of "production" servers used by customers and then automatically grant the necessary access.

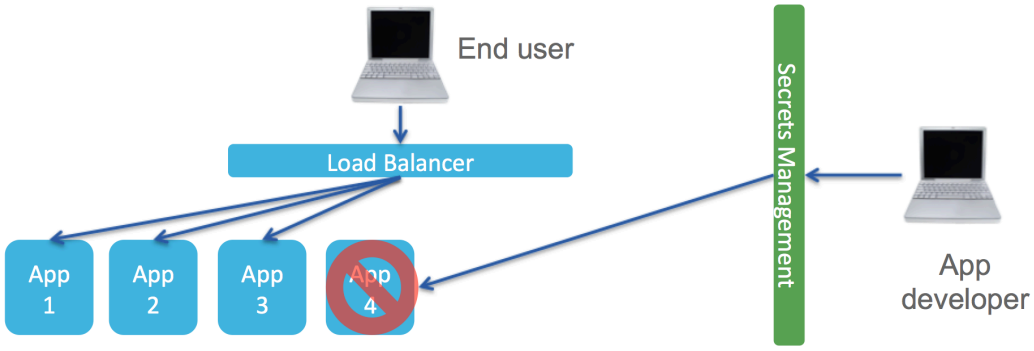


Figure 7: One-time access for debug

Conclusion

The success of the DevOps movement means that DevOps practices are being adopted by diverse organizations, from small startups to Fortune 500 companies. As the movement matures, security is no longer an afterthought and consensus is building about the right ways to integrate security best practices into the DevOps cultural and technical evolution. In the last year alone, there has been an explosion in the numbers of tools available to help secure DevOps environments, from repository firewalls (Weeks, 2015) to new application scanners and security functional test infrastructures (DeVries, 2015), to new SSH Management solutions and the ability to scan Docker containers (Doran, 2015). Contained within the DevOps philosophy and the typical microservices architecture is the freedom to

choose the tools that are best for a particular culture and environment (Newman, 2015) and every month there are more tools to choose from.

The cultural and technical practices that comprise the DevOps shift have both advantages and disadvantages when implementing a Critical Controls-based control infrastructure. In a regulated environment, DevOps teams will need to involve security early in the process to ensure a smooth deployment for new features; the opportunity for greater collaboration with security teams can only be a positive step. The glut of new security tools adapted for DevOps environments has the ability to provide new levels of visibility and automation for implementing security controls. Such new tools may not be fully mature, however, and may have flaws or lack features present in more established products. There is also a lack of precedent when it comes to using such tools for audit against security standards. As the shift towards DevOps continues, we can expect increased maturity for DevOps security tools and best practices that should make implementation of these important controls easier in the future.

References

Amazon. (2011, November). AWS Acceptable Use Policy. Retrieved from https://aws.amazon.com/aup/

Andre, J. (2015, February). *Who Watches the Watchmen? Securing Configuration Management Systems*. Retrieved from http://blog.threatstack.com/who-watches-the-watchmen-securing-configuration-management-systems

Barr, J. (2014, November). Track AWS Resource Configurations With AWS Config | AWS Official Blog. Retrieved from https://aws.amazon.com/blogs/aws/track-aws-with-config/

Belis, E. (2015, November). *DOES15 - Ed Bellis - Security as Code A SecDevOps Use Case* [Video file]. Retrieved from https://www.youtube.com/watch?v=JQ0yLU26j5I

Cole, E. & Tarala, J..(2015). *Implementing & Auditing the Critical Security Controls – In Depth.* The SANS Institute.

Davis, J., & Daniels, K. (2015) *Effective DevOps.* Sebastopol: O'Reilly Media, Inc.

Alyssa Robinson, lyssanr@yahoo.com

DeLuccia, J., Duval, P., Kapadia, M., Kim, G., Mangot, D., Pal, T., Wickett, J., & Yoo, J. (2015) *An Unlikely Union: DevOps and Audit.* Portland: IT Revolution. Retrieved from http://devopsenterprise.io/media/DOES_forum_security_102015.pdf

DeLuccia, J., Gallimore, J., Kim, G., & Miller, B. (2015). *DevOps Audit Defense Toolkit.* Retrieved from http://images.itrevolution.com/documents/DevOps_Audit_Defense_Toolkit_v1.0.pdf

DevOps Forum. (2015). *Mythbusting DevOps in the Enterprise*. Retrieved from IT Revolution website: http://devopsenterprise.io/media/DOES_forum_addressing_culture_102015.pdf

De Vries, S. (2015, April). *Automated Security Testing in a Continuous Delivery PipelineDevOps.com*. Retrieved from http://devops.com/2015/04/06/automated-security-testing-continuous-delivery-pipeline

Doran, J. (2015, July). *Is your Docker container secure? Ask Vulnerability Advisor! - BlueMix Dev*. Retrieved from https://developer.ibm.com/bluemix/2015/07/02/vulnerability-advisor/

Dunn, J. (2014, September). *Detecting & Repairing Shellshock with Chef | Chef Blog*. Retrieved from https://www.chef.io/blog/2014/09/30/detecting-repairing-shellshock-with-chef/

Edwards, D. (2012, November). *The History Of DevOps - IT Revolution IT Revolution*[Video file]. Retrieved from http://itrevolution.com/the-history-of-devops/

Fowler, M. (2010, March). BlueGreenDeployment. Retrieved from http://martinfowler.com/bliki/BlueGreenDeployment.html

Fowler, M. (2013, May). ContinuousDelivery. Retrieved from http://martinfowler.com/bliki/ContinuousDelivery.html

Gartner (2015, March). *DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations*. (n.d.). Retrieved from http://www.gartner.com/newsroom/id/2999017

Gilpin, K. (2015, April). *What is a DevOps Secrets Server?*. Retrieved from https://blog.conjur.net/what-is-a-devops-secrets-server

Hay, A. (2013, May). *Automating Secure Server Baselines with Chef*. Paper presented at ChefConf.

Humble, J. (2014, February). The case for continuous delivery - O'Reilly Media. Retrieved from https://www.oreilly.com/ideas/the-case-for-continuous-delivery

Humble, J., & Farley, D. (2011). *Continuous delivery*. Upper Saddle River, NJ: Addison-Wesley

Linthicum, D. (2014, October). Devops has moved out of the cloud | InfoWorld. Retrieved from http://www.infoworld.com/article/2836372/cloud-computing/does-devops-drive-the-cloud-or-vice-versa.html

Loukides, M. (2015, February). What is DevOps (yet again)? - O'Reilly Media. Retrieved from https://www.oreilly.com/ideas/what-is-devops-yet-again

Minjar, & Amazon Web Services. (2015). *ITIL Asset and Configuration Management in the Cloud*. Retrieved from http://www.minjar.com/documents/resources/AWS_asset_configuration_management_whitepaper.pdf

Netwirx. (2015, December). Security Concerns and Lack of Visibility Hinder Cloud Adoption, Say 65% of IT Pros. Retrieved from http://www.netwrix.com/netwrix_reveals_security_concerns_in_cloud.html

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. Sebastopol: O'Reilly.

Pais, M. (2015, November). Speeding and Scaling the DevOps Enterprise - IT Revolution IT Revolution. Retrieved from http://itrevolution.com/speeding-scaling-devops-enterprise/

Parsons, T. (2014, August). *5 Ways to Make Your DevOps Team More Efficient | Logentries*. Retrieved from https://blog.logentries.com/2014/08/make-your-devops-team-more-efficient/

Petazzoni, J. (2015, May). *Someone said that 30% of the images on the Docker Registry contain vulnerabilities*. Retrieved from https://jpetazzo.github.io/2015/05/27/docker-images-vulnerabilities/

Puppet Labs. (2015). *2015 State of Devops Report*.

Reid, J. (2014). *DevOps in Practice*. Retrieved from O'Reilly website: http://www.oreilly.com/webops-perf/free/devops-in-practice.csp

Smith, S. (2014, April). *Atlassian Blogs: Practical Continuous Deployment*. Retrieved from http://blogs.atlassian.com/2014/04/practical-continuous-deployment/

Suleman, A. (2015). *8 Proven Real-World Ways to Use Docker*. Retrieved from https://www.airpair.com/docker/posts/8-proven-real-world-ways-to-use-docker

Riley, C. (2014, May). Meet Infrastructure as Code - DevOps.com Retrieved from http://devops.com/2014/05/05/meet-infrastructure-code/

Timberman, J. (2015, April). *Chef Audit Mode: CIS Benchmarks | Chef Blog*. Retrieved from https://www.chef.io/blog/2015/04/09/chef-audit-mode-cis-benchmarks/

Vadalasetty, S. (2003, October). *Security Concerns in Using Open Source Software for Enterprise Requirements*. Retrieved from https://www.sans.org/reading-room/whitepapers/awareness/security-concerns-open-source-software-enterprise-requirements-1305

Trustwave. (2015). *Trustwave Global Security Report*. Retrieved from https://www2.trustwave.com/rs/815-RFM-693/images/2015_TrustwaveGlobalSecurityReport.pdf

Weeks, D. (2015, December). DevOpsSec: Survival is Not Mandatory - DevOps.comDevOps.com. Retrieved from http://devops.com/2015/12/08/devopssec-survival-not-mandatory/

Zeltser, L. (2015, December). *Security Risks and Benefits of Docker Application Containers*. Retrieved from https://zeltser.com/security-risks-and-benefits-of-docker-application/