National College of Ireland

Continuous Security; Investigation of the DevOps Approach to Security

MSc Project Report
Cloud Computing

Conor Deegan
Student ID: x15023257

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Conor Deegan
Student ID:	x15023257
Programme:	Cloud Computing
Year:	2020
Module:	MSc Project Report
Supervisor:	Vikas Sahni
Submission Due Date:	17/8/20
Project Title:	Continuous Security; Investigation of the DevOps Approach to Security
Word Count:	6772
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	24th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Continuous Security; Investigation of the DevOps Approach to Security

Conor Deegan
x15023257

Abstract

The DevOps approach to application development and the continuing shift to the 'Cloud first' model has brought about a paradigm shift in the delivery of Cloud applications. 'Continuous Delivery and Integration' describes the pipeline from the DevOps team to customer in which much of the build and deployment process is automated, ensuring faster time to market. This has resulted in a change in the development patterns of new applications. New modules of code are developed and released hourly. This brings about the need for a paradigm shift in the way security is thought about within this context. Manual security processes as an afterthought are a bottleneck for many companies attempting to implement a Continuous Delivery(CD) pipeline. An imbalance in the present research in integrating security into CI/CD is present. This project will address this imbalance by adding quantitative research to the literature in this area.This project also presents a solution to this issue by integrating security into the CI/CD pipeline by automating the process. This system will bring about a true realisation of the term 'Continuous Security' in the context of a DevOps infrastructure environment.

1 Introduction

The advent of the cloud computing paradigm coupled with advancements in the processes of development has forced the software development community into re-evaluating how new applications are developed and deployed to the cloud. DevOps is often the answer that companies are coming up with.The adoption of DevOps principles and methodologies allows organisations to achieve agility and velocity previously impossible using tradition methods of development and delivery.(1)

Automating many of the tasks which traditionally slowed down this process has meant that organisations could transition from monolithic code development to fast and frequent 'sprints'. This means faster development cycles, releasing new features rapidly with the ultimate goal being faster time-to-market.This is achieved using a series of practices and tools known as Continuous Integration/Continuous Delivery. This refers to the automation of many of the aspects of the process of the delivery of new application and integration of new releases from Source Code Management, through build , testing and deployment to the production environment.

Researchers and industry specialists alike are finding a gap in the research and real-world implementation of this new approach. This is regarding the security of these rapidly built applications and micro-services along with securing the delivery pipeline . Many

organisations are still utilising traditional security practices with security teams separate to the DevOps team, reviewing . Security will have to adapt to the change in boundaries in applications, infrastructures and networks(2) Researchers are now exploring various models of integration of security processes into CD. This is referred to in the research as 'continuous security' or 'DevSecOps', both refer to the practice of modelling a new CI/CD process which includes some or all of the security processes into one, automated delivery pipeline.

Security is vital in the context detailed above. Breaches in the era of cloud have cost companies billions(3; 4) and loss of confidence from stakeholders. The implementation of CI/CD presents its own unique security challenges.

As examined by (5), there are cost implications to attempting to deliver applications using cloud platforms which have been developed in large, monolithic blocks of code. Edge computing, in particular AWS Lamda make it possible to have a lot of application logic in the front end where it is less network intensive to perform certain frequent operations, depending on the application. The practice of continuous delivery and integration in the context of cloud application development and deployment is redefining the processes and outcomes of these tasks. Automation servers such as the open source project Jenkins, allow for an integrated pipeline from development, testing and deployment phase which, when optimised for an individual organisations needs, can significantly increase the quality and frequency of development of new applications and new features to existing applications to customers.These new design and deployment principles and tools coupled with the cloud computing paradigm have enabled organisations engaged in software development to decrease the 'time-to-market' of new applications and new features to existing applications.

However continuous delivery and integration mark a significant change in the process of development and delivery of applications. Therefore the question arises; how do we accurately measure a successful implementation of a continuous delivery pipeline? Defining key metrics for this is a major topic of research.

Also, as the process of building and deploying applications through the CI/CD model continues to be increasingly automated, how do organisations ensure that security of both the pipeline and the applications being developed? This is another key research question of this project.

In order to test and find solutions to these research question, this project will firstly construct a lab in which a typical CI/CD pipeline will be developed using the tools and methodologies found to be most popular and effective during a consultation of the literature. A series of varying experiments designed to test typical workloads found in development, testing and production environments will be designed and carried out.

As identified by Buyya et al(2), continuous delivery and integration is a key area . The ability to quickly react to issues in the production environment rather than a focus on strictly preventing those issues through a lengthy and more methodical design process is evident in much of the literature. Automation of a larger number of the processes that support application development and delivery is the obvious solution. This would allow for better fault/failure management without compromising the the agile approach that is enabled by Continuous integration and delivery.

When it comes to the delivery of new software in a cloud-native context and the testing of said software, it is no longer acceptable to simply report the results of failed tests and stop the process in its tracks there and then. Whether that process be the build, test, or deployment phase, failure management is becoming a major area of research in

the context of continuous delivery. Speed of delivery, in many organisations is taking precedence over meticulous planning and time is of such value that anything which slows down the pipeline from developer and end user comes under scrutiny.

There is also the question of the social challenges that arise from organisations attempting to transition from traditional development to the principles of continuous integration. This is due to the fact that many organisations are used to more traditional development and deployment strategies. The aim of this project is to bridge that gap by providing a blueprint for the adoption of Continuous Security processes into their new CI/CD pipeline

This project will provide an in-depth critical analysis of the current practices in testing and security in the context of the Continuous Delivery/Continuous Integration pipeline. An analysis of the literature in the fields relevant to this area of research such as cloud application security, test automation and DevOps approach to continuous delivery and integration was conducted. In reflection of this research a system is proposed in which DevOps principles for small and frequent software releases is combined with security are combined in a functioning CI/CD pipeline with automated security processes.

This iteration of security testing would be in contrast to traditional security testing which usually takes place post development and causes a bottleneck to the continuous delivery and integration of new applications and new features in that applications. This project sets out to promote the inclusion of the security testing process into the software delivery cycle. An automated software delivery and integration pipeline calls for an automated approach to security testing. This project will set out to firstly implement a typical continuous delivery pipeline based on research. This will involve using the most commonly found toolset for building, testing and pushing an application from development to production. This simulated CI/CD pipeline will then be benchmarked with and without the proposed automated security and with to give an indication of the performance impact of this approach.

2 Related Work

In this section the existing approaches in the various key areas related to the integration of security processes into the DevOps approach to cloud-native application development, specifically Continuous Integration/Continuous Delivery pipelines. In the first section the principles of DevOps and CI/CD are analysed by identifying key works in these fields, the paper then moves on to examine current security issues in application development.

2.1 DevOps, Continuous Integration and Continuous Delivery

DevOps is a blanket term which describes the integration of development and operations. This integration is achieved through the automation of the tasks and functions of both the development and operations team's work. Adapting the DevOps philosophy is an attempt to restructure the way an organisation produces new software by changing focus from large individual 'pushes' from each department towards a large deliverable outcome, to a culture of continuous delivery of smaller software releases. In DevOps, communication between various teams during this process is key. (6) provides an analysis on the DevOps approach to software engineering.

As outlined in the analysis of the previous research paper, continuous delivery and continuous integration can be seen as DevOps promises shorter SDLC(Software Develop-

ment Life Cycle) and an improved and streamlined design, development and implementation process. Collaboration between developers and operations enhances the quality of the code produced, and the frequency of version releases is significantly more than a traditional waterfall approach Continuous Delivery and Continuous delivery represent a paradigm shift in the software development and delivery process. The group of tools and methodologies which enable this vary greatly in complexity and role within the process. This section will cover an analysis of literature in this area as well as a critique of related works utilising the same technologies and methodologies demonstrated in this project.

Jez Humble can be seen as the most prolific academic researcher in the area of Continuous delivery and integration.His work on the development of the agile methodologies which became know as continuous integration and continuous delivery are well referenced in the research around the pursuit of faster, more reliable code releases.One such work in his 2010 book (1) is widely referenced in the research. In it Humble presents the ideologies which underpin the implementation of continuous delivery and integration. This includes best practices for implementing automated build, test and deploy stages within the process, utilising toolsets including tools used in this project such as Jenkins.

Humble's(1) comprehensive book will be used as this reference guide and was chosen as it is well-cited in similar academic projects as this one and is written on Continuous Integration/Delivery best practices. The book is not very recent but Jez Humble, the author is a respected author/researcher in the area of DevOps and Continuous Delivery and this book has been cited in numerous works on similar topics as this project. The book goes into detail on the various elements that make up a successful continuous delivery pipeline, including a guide on test automation. Although security testing is not specifically mentioned, the book will still be useful as an overview of the main concepts. More specific academic journals will be used to build upon this and narrow the focus of the research as the literature review builds and gains depth.

It has also been necessary to gather information on how CI/CD is implemented. this includes research into the tools used and how these tools interact to form a complete CI/CD pipeline from Source Code Management. CI/CD is characterised by a modular architecture in that each 'stage' is a separate yet connected component of the overall pipeline. Shahin et al (7) produced a study which examines the variations in architectures of CI/CD pipelines depending on the specific needs of 19 organisations. The empirical study is an in-depth analysis of CI/CD and how it impacts the structure of the applications it's framework and principles produce and how a wide variety of both open-source and proprietary tools interact to form a pipeline from development to production environments.

Another area of research which is of particular significance to the development of the hypothesis of this project, is defining metrics for the successful implementation of the pipeline. Traditional metrics for measuring the efficiency/performance of the software development and delivery process are not compatible to this new process. The various tools and methodologies used to create an automated pipeline between development and deployment require a set of metrics which measure the specific metrics which give the researcher a clear picture/definition of what is desirable and successful and what is not. Defining a set of key metrics is a key process in effectively quantitatively analysing a set of processes or system.

One such piece of research which addresses this is (8). This academic paper is produced based on a case study of a mid-sized software company and their experiences in implementing and maintaining continuous delivery pipeline as their mission critical soft-

ware delivery platform/practice. As a CI/CD job is pushed through the pipeline, various forms of metadata are produced in a variety of forms. For example, the results of JUnit tests are published in XML format, and a build history demonstrating the fail/success history of the job. Execution time and other metadata is also produced.

Lethonen et al propose metrics such as 'Time-to-market' which will quantify the number of successful releases of the software produced to the production environment

Lethonen et al also brings up an important characteristic of the CI/CD pipeline and how automation enhances the software delivery process. The immediate feedback in the form of metadata that a CI/CD pipeline, primarily the CI server which is Jenkins in this case, gives the development team unparalleled responsiveness not possible in more traditional forms of deployment.

Although Lethonen's paper is a useful insight and will become useful in interpreting the metadata which is created from the pipeline, it does have a lack of consideration for the role of security testing in this process. This is the gap in research which this project is attempting to address. Can security testing be integrated into the pipeline in such a way which improves the performance of the pipeline. However assessing this performance will mean that the metrics defined within this project will become useful.

2.2 Cloud Application Security

Central to developing a thesis in the field of securing the continuous delivery pipeline is to first identify and analyse the best practices in the field of cloud application security. Using current best practices and examining the security requirements of cloud applications in the context of a Continuous Delivery pipeline will be central in developing an adequate testing framework in which to express the automated security test cases.

DevOps and the adoption of Continuous Delivery workloads has he potential of closing the gap between software developers. It also significantly changes the relationship the security team has with the applications that are developed in this small, modular pattern with frequent releases.

The European Agency for Network and Information Security(ENISA) releases regular publications with regards to security threats and breaches of cloud service providers and cloud applications. A review of these articles and the resulting recommendations from ENISA on mitigating the threat of these security attacks will be important during the course of this research.One such publication which is the 'Cloud Incident Report' by ENISA(9) will give vital insight into the most common security threats to cloud providers and the infrastructure and applications developed within this environment. The document examines these threats and makes suggestions for mitigation of these threats for both Cloud service providers and the users of these services. The document was released in 2017 making it slightly outdated, but it does come from ENISA which is an EU agency on information security, meaning the data collected for the report is reputable and the insight it provides will be invaluable in the design of security tests which ensure that all vulnerabilities are identified prior to an application being pushed to the production environment.

Companies that adopt a cloud-native approach to a wide-variety of their IT including infrastructure and application development have to be aware of the unique security concerns of this approach. The Cloud Security Alliance(CSA) is an organisation set up to promote best practices in security for cloud computing and achieves this by producing research and guidelines based on this research in order to educate companies on the

importance of securing their cloud infrastructure. The work this organisation produces will be useful in determining the security practices implemented within this project. One such work is Mogull et al (10). This guide gives security guidance for each specific area of cloud computing including Application Security and Infrastructure Security, two areas closely associated with the work being undertake in this research project.

Risk assessment is a key process in securing the continuous delivery pipeline. Identifying the specific ways in which cloud computing and cloud application development are vulnerable to attacks will be key in developing a system to systematically test cloud applications for these vulnerabilities and integrating this mechanism into the continuous delivery pipeline. Carthensen's well-known book; 'Cloud Computing:Assessing the Risks'(**?**). The book covers cloud computing and the associated security risks that arise from it.The book will be important in gaining a deeper understanding of how security must adapt itself to this new form of distributed computing. The book will add insight into the design of the security approach of this research project. The book is from 2010 but is a well established reference in the area of cloud security and many of the design principles and approaches to security within the book will be relevant in a successful implementation of this research project.

It will also be necessary to gain a deep understanding of security in the context of cloud applications specifically. Open Web Application Security Project(OWASP) provide a yearly report on the top 10 security threats in this area and have been mentioned in numerous journal articles during the course of the research for this document. One such journal article is Nagpure and Kukure's paper(11) on risk assessment and testing methodologies of web applications. In this paper the top 10 OWASP security threats are identified and analysed. Also various methods of manual and automated penetration tests using a number of noteworthy tools are implemented and the results explained. This paper will be useful in identifying the current security concerns specific to cloud applications. The paper does not refer specifically to the DevOps approach to application development or continuous delivery work flows, but it will provide useful when isolating the security requirements of applications in the context of the cloud.

The OWASP Top 10 project is a well established, open-source publication for outlining the most common security threats to web applications(12). This paper is released by OWASP every year and provides an overview of the biggest security threats and vulnerabilities to web applications and also outlines the ways in which security teams can protect their software from these vulnerabilities. Cross-site scripting is outlined as a major concern as well as vulnerable or outdated dependencies. This will be a valuable source in developing the security test cases as part of this project. Making sure the test cases cover the important areas of vulnerability will be vital. OWASP also have developed a number of tools which could be integrated into the CI/CD pipeline, including OWASP dependency checker.

2.3 Integrating Security into CI/CD

This research project is underpinned with a variety of research papers which delve into various aspects of CI/CD as a DevOps practice. The main body of this research tends to focus on qualitative studies which survey developers and businesses for their opinions on certain tools and principles used to implement a CI/CD process. These often ask for the user experience and focus on surveys as a means of data collection. This is particularly true in the emerging research area of security integration into the DevOps practice of

CI/CD. The impact of integrating security processes into the CI/CD pipeline is measured in most of the available research by surveying users and customers of their experiences. This research project will attempt to adapt a quantitative approach to analysing the impact of continuous security to answer the research question; What key metrics can be identified as an accurate measure of performance of a continuous security approach?. Central to meaningful research into this will be the identification of key metrics with which reproducible experiments and results can be obtained for future work in this area.

To begin to answer this question, an evaluation of the current research in this area is conducted, starting wuth gaining an understanding about the specific security challenges faced by an organisation adopting the DevOps approach. Rafi et al.(13) addresses the need to identify these challenges in a paper which attempts to prioritise these challenges in order of necessity.

The next area of related works to focus on is that of the response to these challenges. This refers to how the cloud application development industry, specifically the DevOps community are responding to the challenges posed by securing their CI/CD process. There has been multiple references in the research conducted to the concept of 'moving security left'. Cope(3) explains this concept as moving security further to the beginning of the SDLC. The article goes on to give an analysis of the state-of-the-art in terms of bringing this about in terms of frameworks, toolsets and best practices. It is noteworthy that much of the research in this area refers to qualitative analysis. It becomes apparent as one goes through the literature that there is a lack of quantitative analysis that this particular project will attempt to address.

Kumar et al(14) proposes a model for this adoption of security in a comprehensive research paper into the impact of continuous security on a DevOps SDLC. The paper refers to research covering the principles, tools and methodologies for implementing such an approach in a real-world business scenario. This paper is significant to the research being conducted in this project, because of its identification of the business objectives which motivate an adoption of CI/CD and the key metrics that be used to measure the performance of a successful implementation of CI/CD and continuous security. Velocity is identified as a central business objective of the companies which adopt a CI/CD model of software development. Brunnert et al(15) identifies velocity in this context to be the constant flow of applications, new features and bug fixes through the CI/CD pipeline so that the pipeline from DevOps team to production environment is seamless. As previously identified in the consultation of the research as part of this project, security is often one of the main bottlenecks to achieving true velocity in a CI/CD pipeline. Although this particular paper is older, it is a well respected piece of research in this field and is useful in identifying How can velocity as a business objective of the DevOps approach to SDLC, of which CI/CD is a methodology of, be measured? Kumar(14) identifies a number of key metrics which can be used to measure the performance of a successful implementation of CI/CD principles.Frequency of deployment, number of deployments in production in a given period, mean time of deployment, post approval average time taken to deploy a release in production and mean time to market are amongst these. Any implementation of CI/CD could be measured as a success in terms of velocity if it measured positively in these key metrics. Although this research project is influenced by the work done by Kumar and others in the field of modelling a pathway to adoption to continuous security, this project will provide unique value to this field of research by attempting to quantify the performance of CI/CD with continuous security using the key metrics identified in the research. This will give further researchers a better platform with which to gain

even deeper understanding into the performance implications of security in the context of CI/CD.

Quantitative research into the impact of continuous security is central in the gaining of key insights into how this will impact comapnies which adopt this approach. Security is not an area of application delivery which is often measured in this way. Casola et al. (16) propose a model of assessing security's impact using SLA's as a means of quantifying this impact and collecting data. The research is motivated very similarly to the research conducted for this project, in that it identifies that the move to the cloud has prompted a shift in application development in the direction of more releases and jump in speed to market.

The paper examines how security can be automated within this environment and proposes 'a novel Security SLA-based Security-by-Design methodology' which includes the automation of risk analysis and security assessment of cloud-native applications. In order to test this approach Casola et al. set up an experiment where a typical DevOps cycle is simulated with the proposed security automation(involving two novel security automation tools) and one without. The metrics used to collect data were 'efficiency, usability and time consumed'(16). The methodology used in this paper was a useful framework for the methodology of this research project. Casola et al. differ in approach with this project in the key metrics used to quantify the impact of continuous security and also the SLA approach to security is slightly more arbitrary than the goal of this project which is to compare the performance of the CI/CD pipeline in terms of velocity.

There is a paradigm shift occurring in many organisation's development and deployment process. The Devops approach has significantly increased the speed to market of new applications and adding features to existing ones. However, as discussed in this paper by Mohan et al. (17), there is still widespread suspicion of the benefits of integrating security and DevOps. The paper consists of a report on IBM's road to integrating their security team and practices with their DevOps approach to development and deployment. A real real example/case study will aid this project in understanding the implications of integrating security processes in the DevPps development pipeline. The knowledge gained from this report will enable a project which is conscious of the common concerns and footfalls of this process. The process of integrating security into the DevOps development process involves the collaboration between teams normally kept separate in the traditional software development environment; developers and security experts. The purpose of this project is to determine if an automation solution can aid in bridging the gap between these two teams by integrating security processes into the development process. This paper is useful in outlining the many obstacles and concerns that developers and security experts have in integrating their processes into one seamless workflow.

Identifying the practices, technologies and principles which would underpin a successful integration of security into a DevOps application/services delivery model is a key research direction of this project. In this manner, papers such as Kumar et al. (14) will be useful identifying these elements. This research paper breaks down each component of a CI/CD process of application delivery from the principles, enablers, key elements and workflow of a CI/CD process and through this outlines the many obstacles in adopting a 'continuous security' approach into this process. This paper was central in isolating the research questions for this research project as Kumar attempts to discover the best approach to adopting continuous security.The most significant of these from the perspective of this project was Kumar's identification of key metrics in measuring the performance of a successful implementation of a CI/CD process with integrated security.

Velocity is identified as a business objective by companies that adopt a CI/CD approach to application/ software delivery. This is also identified as a key reason for adopting DevOps in the seminal research papers in this area. One such paper which is cited time and again in DevOps research is (15) which examines the impact of a performance-oriented approach to DevOps.

Larrucea et al.(18) present a different yet equally valuable approach to integrating security into DevOps practices of application development and delivery. Focusing on a real-world scenario based in the health sector, the paper proposes the inclusion of security measures in the DevOps pipeline; specifically 'code analysis at the integration phase'.

2.4 Test Automation

Identifying and exploring the concepts around testing and the automation of testing will be a fundamental aspect of this research project. In order to develop a system which improves on the current process of testing, the research must first identify the current state-of-the-art in test automation, and then explore how similar research has been conducted in test automation in the context of Continuous Integration/Continuous Delivery.There are several interesting aspects to the process of automating the testing process. The first and most obvious is the translation of test cases, which are generally written in natural language, into automatically executable scripts and/or code. Anand(19) in this paper outlines the most common tools and methodologies used in the test automation process.Test automation as a strategy is present in a number of engineering fields. The research conducted by Flemstrom(20) has the potential of being very useful to this project. Flemstrom and his colleagues present a case for the reuse of test cases as opposed to the generation of a whole new set of tests for every system/application as is usually the case in manual testing. This would further compress the development lifecycle: one of the key aims of this research. Flemstrom(20) proposes that identifying similarities in test cases could be used to fully automate the testing process. Behaviour Driven Development(BDD) is one design principle for developing security test cases which was discovered during the research. First developed as a concept by North(21) in response to the complexity of TDD(Test Driven Development).

Bingamingu et al.(22) discusses BDD in a wider context of engineering and design, as well as within the software testing context. The author also discusses the drawbacks associated with the implementation of BDD as it can become complicated to maintain the specifications of an application, especially in the context of continuous delivery.

Nezhad et al.(23) will provide us with an example of what BDD looks like in the context of development of test code.

During the course of the research, many algorithms were discovered which are used to generate test scripts from a set of functional test cases of a given system/application. (20)

Automating a process will have its positive implications as well as negative. The purpose of productive research is to analyse these implications and present a clearer picture to the academic and corporate communities of these implications. Blake(4) presents the argument that automating security will eliminate human error which can occur through manual application security testing. The article makes the suggestion of embedding application security scanning with the CI/CD pipeline. This automates the testing of new releases while allowing the DevOps team to interpret the scan results as part of the development process, rather than after. The article uses a case study of a software company

which switched to a fully automated CI/CD pipeline with 'continuous security. According to the article, the company's deployments went from four hours to less than 30 minutes. The article also reports that the DevOps teams were twice as efficient using the 'build-test-deploy' process of a CI/CD pipeline than before the introduction. The suggestion of embedding application security scanning into a CI/CD pipeline is a means by which the aims of this project could be achieved. By simulating a CI/CD process with embedded application security scanning and running code releases through it, this project could gather meaningful data on the performance impact of a continuous security approach.

2.5 Comparison of Tools/Methodolgies Used

As will be outlined in detail in the system architecture description and visual, the automation server is a central component to any CI/CD pipeline. The analysis of the academic research was conducted within the area of Continuous delivery and integration. Jenkins is one such tool. Jenkins is an open source CI server which can be hosted on most operating systems and platforms including within a container technology such as Docker. Versatility and ease of deployment are Jenkins most attractive traits. Jenkins is useful in such a wide variety of use cases thanks to its use of the plugin model. Many different types of 'jobs' can be automated using Jenkins in every facet of the software development, delivery and maintenance world. Within the context of continuous delivery, Jenkins can be used to automate otherwise manual processes.Jenkins is an open-source server and started life as a novel, nice to have addition to the software developers setup. However as its popularity grew, Jenkins is now a central component of many organisation's mission critical application delivery.

Jenkins also allows for the implementation of the concept of 'Pipeline-as-code'. This utilises the DSL format and allows for the various stages of the CI/CD pipeline to be defined. For example, the 'Build' stage in which the source code is converted into a JAR(in the case of a java application) will include the shell scripts which trigger the build tool which is Apache Maven. After the stage is deemed to been sucessfully completed by the Jenkins server, the next stage of the pipeline is automatically triggered.

Another key component of the continuous delivery pipeline is the source code management tools. There are many choices, both proprietary and open-source. Subversion, GitLab and GitHub offer slightly different variations of version control, suited to varying use cases. In the case of this project, Gitlab was chosen for a numver of reason. The first reason is that it allows for the download of an open-source docker container containing the software. This allows a user to set up a git server as a container, connected to the other components of the pipeline through Docker's internal network interface. The second is GitLab's focus on CI/CD integration. This is demonstrated in the feature of webhooks. This feature allows for triggers to be set which notify the Jenkins container every time changes are made to the source code repository

The source code and the means of automating the integration process are defined, next a build tool is required to take the source cold and configure the compilation into usable applications. Java is a popular language for the development suite and comes backed with a suite of tools for implementing and testing applications. Apache Maven is a well established build tool which allows for the easy configuration of the build process to suit the specific needs. It is also easily integrated into the pipeline due to its compatibility with Jenkins. Simply install the plugin and adding a maven build and package step to any job that is being configured.

One key aim of this project is to make the various components modular so as to ensure they reproducable in many experiments. As each component of the continuous delivery pipeline must be able to communicate with the other over a common network interface, Docker becomes an obvious answer to this problem. Docker is the most popular container orchestration solution used today. Containerization is a form of virtualization which can be described as a virtual machine in its most basic form. (24) describes containers as "operating-system-level virtualization". Containers are isolated from the underlying hardware but are also isolated from each other. This means that an application running in one container is completely isolated from another, each running on their own virtually provisioned container OS. Docker provides an easy-to-use platform for containers which, although less performance orientated than Kubernetes, will be ideal for the scope of the experiments performed for this project.

Docker has been chosen as the platform on which to build a containerized continuous delivery pipeline. The ability to create version controlled image at each stage of the development of the pipeline is invaluable in a research project. (25) analyses the benefits of using Docker to underpin all research in the area of computer science for a number of reasons. He references Docker's 'reproducability' as a key benefit as often one researches work is not easily reproducible by another researcher. The Docker compose feature allows a researcher to lay out all of the services and features of their system in one easy to decipher 'Dockerfile'. The author also refers to the portability of Docker as a key benefit. As each container is completely isolated and independent of the hardware and operating system on which it runs, a researcher can be more certain of the predictability of the application performance which is running within each container, regardless of the capabilities of the system which it is running on. This also ensures that each module of the pipeline is reusable. As the pipeline was developed, there were many times ideas had to be scrapped in replacement of better suited tools/methods. The ability to re use any valuable modules in the pipeline in future iterations of the project proved invaluable. Although this paper is not specific to the development of a CI/CD pipeline, it justifies the use of Docker within the project by demonstrating its benefits in the context of computer science research.

3 Methodology

This research project is a quantitative study on the performance impact of introducing a 'continuous security' approach in a CI/CD application development and delivery pipeline. In order to quantify this impact, it has been chosen to examine the impact on the velocity and agility business objectives.Kumar et al.(14)identify velocity as a key business objective for organisations that adopt CI/CD as their primary framework for application development and delivery. A holistic approach to evaluation of the Continuous Delivery and Continuous Integration pipeline was in that each component of the pipeline was evaluated in the context of the system as a whole, in order to benchmark a standard pipeline, and one with the automation of the security testing process.

Central to any successful experiment methodology is a consistent environment in which the experiments are conducted. The lab for this experiment was consistent throughout the experiments. this was achieved by running two VM with the desired specifications to ensure consistency of results. The conditions on the machine on which the VMs were running remained consistent throughout experimentation also.

Kumar et al.(14) outline a number of key metrics in measuring velocity of a successful CI/CD pipeline as follows;

- Frequency of Deployment(number of deployments in a given timeframe)
- Mean Time of Deployment(Average Time to deploy a release to production)
- Mean Time to Market

It is obvious that in measuring these metrics the unit which will be most used is time. The most efficient way of collecting meaningful data will be to compare the CI/CD pipeline by the above metrics with continuous security integrated and without and analyse the impact this has on the velocity of the pipeline

The key metrics for measuring Agility of the pipeline are as follows;

- Mean Time to Change(Time taken from when change made to version control, to production)

Again, time will be the main unit of measurement for this metric and a comparison of before and after implementation of 'continuous security' will be conducted based on the data collected.

Improving the usability for the developer is only one aspect of integrating and automating security testing into the continuous integration and delivery pipeline. Key metrics which determine a successful implementation of the pipeline must first be defined and then measured in order to ensure that the system is an improvement on manual security. When it comes to security, especially with the risks involved in the context of cloud computing, it is necessary that the process of vulnerability detection for any application is thorough and examines every potential vulnerability before reaching the production environment. Therefore any automation of the security testing must prove to as just as competent in detecting these potential vulnerabilities in any application pushed through the release pipeline.

Kumar et al(14) define a number of key metrics to measure the performance of a successful implementation of continuous security;

- Frequency of Vulnerability detection
- Security Scan pass rate
- Vulnerability fix rate

  In order to measure the Frequency of vulnerability detection, The experiments will compare the number of vulnerabilities in a code scan versus the number of scans. Security scan pass rate will be measured by comparing the number of scans reporting vulnerabilities versus the number of scans conducted. Vulnerability fix rate will measure the time it takes for patches to be developed for the vulnerabilities reported and for the fixed code to run through the pipeline.

In order to obtain meaningful data, the pipeline will be run firstly without the integration of security processes. This will act as the control. There are two applications which will run through the pipeline, representing two real-world applications of varying complexity. After a number of executions of this pipeline, which represents the CI/CD process before security integration, data will be collected and the next phase of the evaluation will be executed. This will involve the inclusion of automated vulnerability checks. The pipeline will be run the same amount of times with security integrated to ensure consistency of data collected.

4 Design Specification

Recreating a typical continuous integration and delivery pipeline first involved research in determining the most popular tools and frameworks used today in the area of cloud application delivery and deployment. This system of tools was firstly deployed locally using VirtualBox and then pushed to the cloud after a full automated build, deploy and test process was achieved.Tools were selected based on their use in the literature reviewed as well as the merits of their functionality.This section will outline the requirements of the system while also describing the toolset and the ways in which the various components interact to create a CI/CD pipeline.

A CI/CD pipeline is designed to automate the process of delivering a new release from the team to production in one automated pipeline. The system must be designed to have minimal human interaction in that once new code is committed to the Source Code Management, the process must be triggered. It is also required that any errors must be reported to the Infrastructure team for review. The pipeline must then build the application using the new code committed and package the artefact. After the artefact is built and packaged the testing process must begin. Unit tests must then be triggered to check that each component of the artefact has no errors. The proposed integrated security testing must then take place in order to simulate the proposed hypothesis of the project. There is an infrastructure required to achieve this which involves numerous components working and communicating with each other. Therefore a network is required to connect the components which execute each requirement of the pipeline. The most fundamental requirement of CI/CD is that the software is always in a production-ready state.

The most efficient way of achieving the requirements is to design the pipeline to work in stages. Each stage represents a requirement outlined above. The first stage begins when new code is committed to the version control system and is therefore referred to s the 'Commit' stage . This triggers the pipeline to begin the process of integration and delivery. The first stage involves the building of the application, referred to as 'Build'. When a change is made to the version control, the source code is compiled by the build tool and all dependencies resolved.Testing is the next stage and is fully automated so that tests are triggered once the artefact is built. The next phase is the security phase where the vulnerability scanner is triggered. This checks the application for any potential vulnerabilities to common security threats. The next phase pushes the new artefact to the production environment The idea is that any changes or new releases are pulled, built and tested continuously in order to form the CI/CD pipeline.

4.1 Jenkins: The CI Automation Server

Jenkins has proved the most popular and common tools found in the review of the literature surrounding CI servers. It is also one of the only open source CI servers

out there. Other CI servers such as GitLab and Circle CI are available but have less integration options with other popular tools within the CI/CD model of development, testing and deployment.

Jenkins is easily the most adaptable CI server available. This is down to its compatibility with a wide variety of tools using its plugin model of integration.Therefore Jenkins has become the centre of the CI/CD pipeline developed for this project. Not only are the tools which make up the source code control, build, test and push all automated and linked through the Jenkins server.

4.2 Docker: Containerized Components

Docker is a popular tool for managing a network of containers. Containers are useful in the context of research as they allow a researcher to simulate the functionality of what would be a large system in the real world in much smaller scale for the purposes of research and data collection. Docker is the industry standard in containerisation.

4.3 Maven: Build Tool

Maven is a build and integration tool which compiles, checks for dependencies and then packages the raw Java code into a JAR which is then deployable to prouduction. It utilises the Project Oriented Model(POM) which allows for the easy definition of all the dependencies of the code in one file. It simplifies the build process of Java applications and is also compatible with Jenkins which makes the build process automated, a requirement of this project.

4.4 JUnit: Testing

Unit tests are written in JUnit which check that each component of the JAR built by Maven is error free. Once these tests are passed the next phase of the pipeline is triggered.

4.5 OWASP Dependency Check

OWASP Dependency check scans the dependencies of an application against OWASP's extensive database for potential vulnerabilities.

5 Implementation

The final implementation and artefact produced by this project was developed in consultation of the literature and designed in support of the thesis proposed in the methodology in order to answer the key research questions. In order to fulfil this the lab must include a complete CI/CD process

Git is one of the most utilised source code management tools in both industry and academia. Git allows the development team the ability to collaborate and version control with ease. Git was used as the SCM for this laboratory set up and was, in the early stages deployed within a GitLAb container running in the test VM, later utilised Github once the CI server was deployed to the cloud. Git also has a useful tool called githooks or webhooks. These are simply trigger which notify the Jenkins server when a new commit has been pushed to the repository.

Maven is a build tool for Java projects. It was chosen for its seamless integration with Jenkins using the plugin method. It simplifies the build process for java applications. It enables the creation of a CI pipeline which is triggered by a new commit from Git and ends in the deployment of this code.

Jenkins was chosen as the CI server. Jenkins provides the link between the code which is developed and pushed to the source code repository, to the the build and testing stages and onto deployment. Jenkins automates this process so that, any time new code is pushed, the CI/CD pipeline is triggered automatically. Jenkins is such a powerful tool due to its integration capabilities through the plugins. It allows Jenkins to become the fulcrum on which the CI/CD pipeline swings from developer to production, speeding up time-to-market for new applications and new features in those applications.

Jenkins allows the user to define every stage of the pipeline and the tools which execute it by using a Jenkinsfile. This file is referred to as pipeline-as-code, in which each stage is defined and the corresponding scripts which trigger when each stage are referred to. It significantly simplifies the process of developing the CI/CD pipeline. Pipeline-as-code is achieved using a groovy script. This script, which is saved alongside the source code of the application in Git repository of each application, is effectively a reference guide for the Jenkins CI server. It outlines each stage of the pipeline, along with detailing which actions to execute at each stage. Jenkins also gives the user feedback as the pipeline executes and notifies the user of any errors and fails. An error immediately stops the pipeline, preventing applications with issues reaching the production environment.

The Jenkinsfile works alongside bash scripts to fully automate the CI/CD pipeline.These scripts contain all of actions which will take place throughout the CI/CD process. These bash scripts execute different actions depending on the stage. Maven builds a JAR(Java Archive) which is a packaged java application,JUnit tests then check that each component of the application is working correctly, a Docker image of this application is then built. This image is then pushed to the Docker registry, deployed to the production environment and published on this environment.

- mvn.sh Maven takes the source code from the Git server and packages the Java application
- test.sh Runs the JUnit tests against the application
- build.sh Builds a Docker image of the application
- push.sh Pushes the image to the DockerHub registry
- deploy.sh Deploys the container to the production VM, locally or in the cloud
- publish.sh Runs the containerized application in the production environment

Docker is used to containerize each component of the pipeline. This is of huge benefit to this project as it allows the system to run on Docker's internal network, simulating a CI/CD process with the properties of a much larger installation, simulating a real-world scenario. The Jenkins Container is connected to the Git Container via the docker internal network. Docker is a common tool utilised by companies who introduce Continuous Delivery/Integration into their cloud application development process. After Maven builds the Spring Java application from the Source Code, this application is then containerized. The image is then pushed to DockerHub, which allows for registry of each iteration of the application.

Docker-compose is a tool developed for Docker which allows the user to define and configure the characteristics of each container. It is installed alongside Docker and is very useful in organising the interaction of multiple containers and configurations for these containers as is required for this project. Docker compose utilises a declarative YAML in which the details of each container is defined.This is known as a Dockerfile and contains all the details of each container including where the container image is to be pulled from, the location of the storage volume on the VM and the network interface which the container will be connected to. The production VM also utilises docker-compose to start the containerized app in the production environment.

Spring Boot is a Java application Development Framework designed for the DevOps methodology. It is used to rapidly create Java applications using the MVC model. The model is comparable to Ruby On Rails for the Ruby development language. The application developed is a simple POST application which can populate the data in a SQL database. Spring integrates easily with Maven which compiles the application with all necessary dependencies.The application is then containerised and deployed to the production environment. These dependencies are then scanned for any potential vulnerabilities with the OWASP dependency scanner, integrating security processes in the pipeline.Securing the pipeline itself is also an important task in the implementation of the CI/CD pipeline.Role-based authentication ensures that only those system users with the required level of clearance can access each part of the pipeline i.e developers can only view the pipeline, security team can only view the test results.

Spring Boot is chosen for the development of the Java application which will pass through the pipeline. The application has been built to simulate a real-world scenario as much as possible. Spring boot is a state-of-the-art frameworks for Java development and is very popular in the DevOps community. It allows for the quick modelling of java applications.

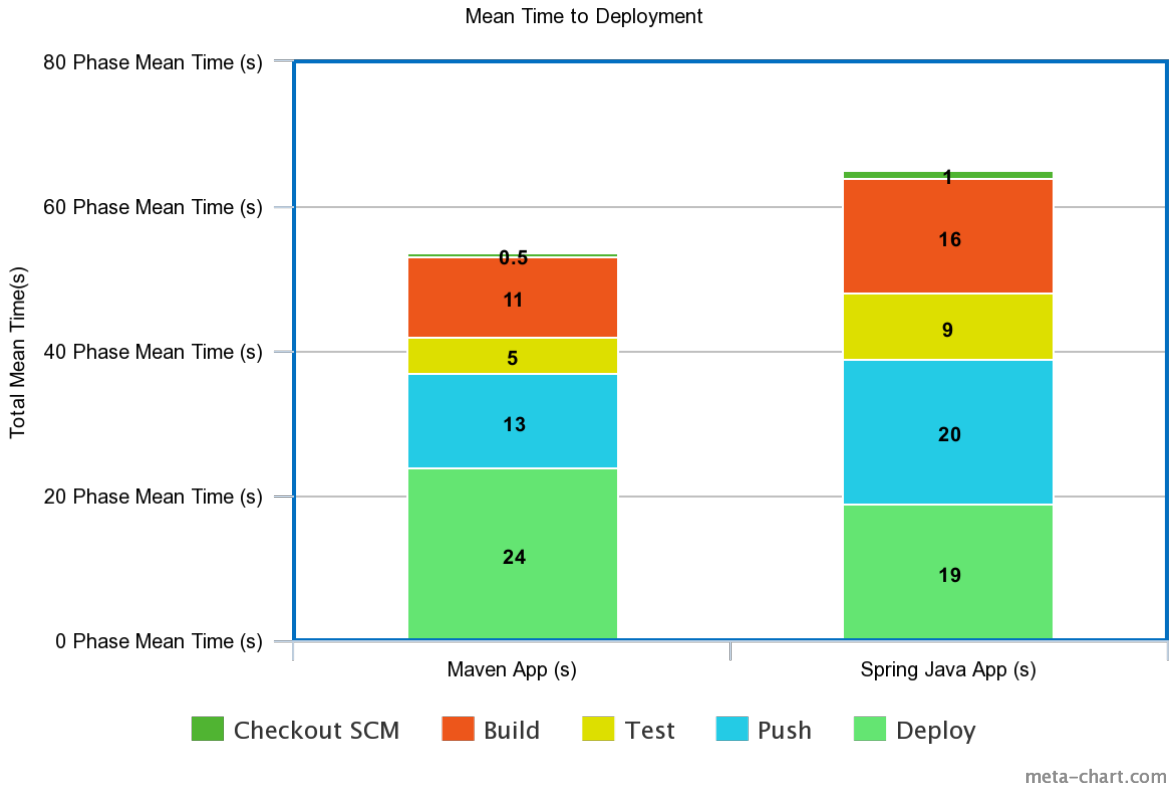
6 Evaluation

With the variety in use cases present in the literature and in many organisations as they implement CI/CD methods and toolsets, the experiments conducted for this research aim to be as close to these as possible to ensure ethe authenticity of the data collected.The system was benchmarked using a number of metrics specific to CI/CD pipelines. The experiments are conducted firstly by running the CI/CD pipeline without continuous security and then with the vulnerability scanner.

6.1 Control Experiment: Pipeline without Security

In this experiment, the pipeline is run without continuous security. Two applications are pushed through the CI/CD pipeline representing two Java web applications of varying complexity. Each pipeline is executed 5 times and the Mean-time-to-production is analysed as outlined in the Methodology. This metric represents the average completion time of the pipeline from checkout of the version control to deployment in the production VM. As we can in Figure 1, The Spring Java application took on average 65 seconds to deploy through the pipeline, while the simple Maven app took 54 seconds.

Figure 1: Average Deployment Time Data for pipeline without Security



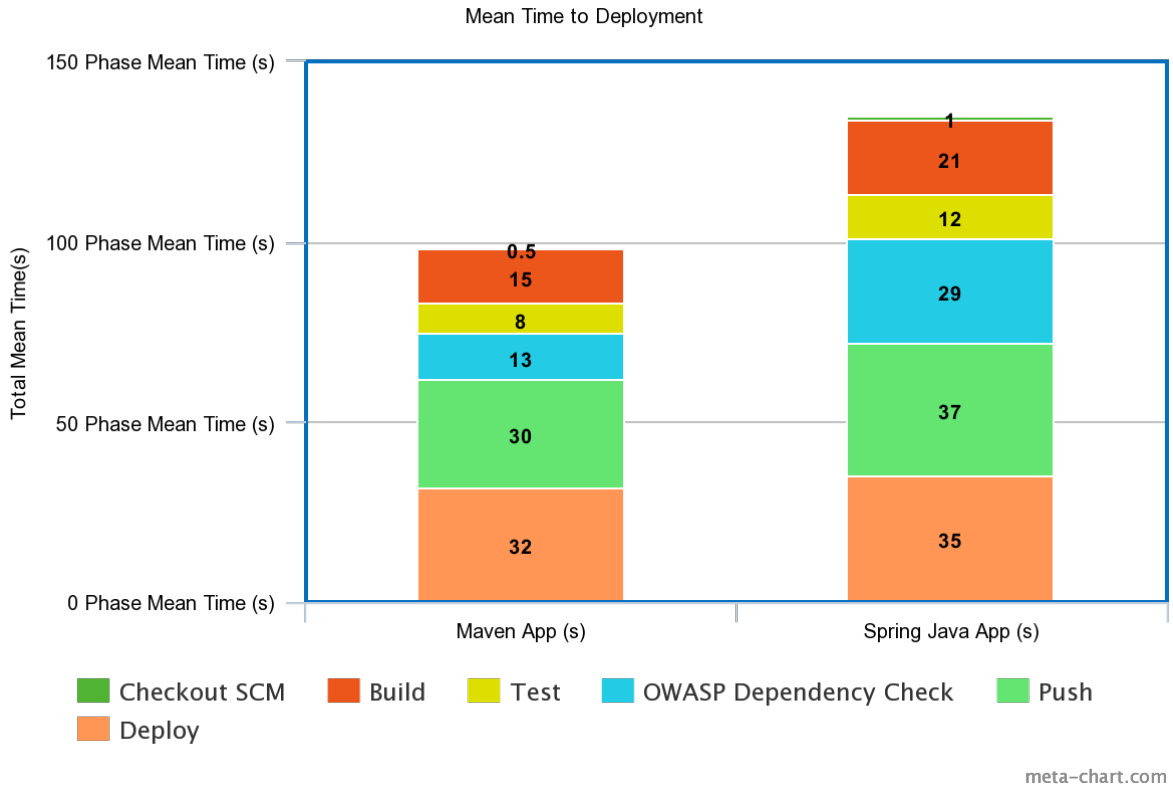
6.2 Main Experiment: Pipeline with Continuous Security

The second experiment includes the proposed automated security processes. OWASP Dependency Checker runs as a plugin with Jenkins CI. It is installed using the Jenkins plugin system and its specifications are then outlined in the Jenkinsfile. The same Java applications were pushed through the pipeline to ensure consistency of data collected with that of the first experiment.The OWASP Dependency checker checks the compiled application package for vulnerabilities and reports these vulnerabilities in order of severity. The pipeline was run 5 times in order to find the Mean-Time-Deployment. In comparison to the control experiment, the pipeline took an average of 135 seconds to complete in the case of the Spring Java application, and 83.5 seconds in the simpler Maven application.

6.3 Discussion

The first experiment demonstrates how a CI/CD pipeline can make the development and deployment of applications fast and automated. The second demonstrates that the application can run though a vulnerability check as part of the pipeline, without a huge amount of slowdown to the overall average time to deployment.Therefore these experiments provide a quantitative analysis of the performance impact of 'continuous security' and determine that it is a viable approach to solving the issue of manual security processes and the bottleneck they create.

Figure 2: Average Deployment Time Data for pipeline with Continuous Security



7 Conclusion and Future Work

As outlined in a number of the works cited in this project, security and its processes have for too long been considered secondary considerations within the SDLC. This is particularly true in the DevOps approach of CI/CD where time-to-market and speed of deployment are given precedence over security. However what this project has discovered is that this approach can actually have the reverse effect on release frequency. This is because if security is addressed in the traditional way, as in by a separate team which runs the various processes an assessments on any given application or feature release, this quickly becomes a bottleneck to production.This project proposes the integration of security into the Software Development Lifecycle within the framework of the increasingly dominant CI/CD framework for application delivery.This project advocates that this will be achieved through automation; enabling a 'continuous security approach to the various vital security processes. A methodology was developed to test the performance impact that this approach has on a CI/CD pipeline. This methodology included a review of industry standards in order to isolate the principles and tools which underpin a typical CI/CD pipeline.

After the experiments were conducted on the proposed system, it was determined that the performance impact of 'continuous security' did not outweigh the clear benefits of this approach and the hypothesis of this project can be said to have been proved.

There is the need for more quantitative research in the area of Continuous Security. Assessing the performance impact of such an approach is central in gaining the insight necessary to decide if this is a viable approach. Much of the existing literature is based on surveys of stakeholders such as developers, managers and customers. There is work

such as Kumar et al. (14) and the earlier work of Lethonen et al.(8) outline metrics by which a successful implementation of CI/CD can be measured. These could be used as the basis for further quantitative research with more experiments devised to assess these metrics.

Future work could also include running a CI/CD pipeline at scale, to fully simulate a real-world scenario in order to gain further insights into the performance impact of integrating security processes into the CI/CD pipeline.

References

[1] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[2] R. Buyya *et al.*, "A manifesto for future generation cloud computing: Research directions for the next decade," *CoRR*, vol. abs/1711.09123, 2017. `http://arxiv.org/abs/1711.09123` (Last Accessed: 10/Sep/2018).

[3] R. Cope, "Strong security starts with software development," *Network Security*, vol. 2020, no. 7, pp. 6 – 9, 2020.

[4] C. Blake, "Reducing risk with end-to-end application security automation," *Network Security*, vol. 2020, no. 2, pp. 6 – 8, 2020.

[5] M. Villamizar, O. Garcs, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures," *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 233–247, 2017. cited By 11.

[6] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, pp. 94–100, May 2016.

[7] M. Shahin, M. Zahedi, M. Babar, and L. Zhu, "An empirical study of architecting for continuous delivery and deployment," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1061–1108, 2019. cited By 1.

[8] T. Lehtonen, S. Suonsyrj, T. Kilamo, and T. Mikkonen, "Defining metrics for continuous delivery and deployment pipeline," vol. 1525, pp. 16–30, 2015. cited By 2.

[9] D. Liveri and C. Skouloudi, "Exploring cloud incidents," *The European Network and Information Security Agency (ENISA)*, pp. 1–14, 2016.

[10] R. Mogull, J. Arlen, F. Gilbert, A. Lane, D. Mortman, G. Peterson, and M. Rothman, "Security guidance for critical areas of focus in cloud computing, v4. 0," *Tokyo, Japan: Cloud Security Alliance*, 2017.

[11] S. Nagpure and S. Kurkure, "Vulnerability assessment and penetration testing of web application," 2018. cited By 0.

[12] T. OWASP, "Application security risks-2017. open web application security project (owasp)," 2017.

[13] S. Rafi, W. Yu, M. Akbar, A. Alsanad, and A. Gumaei, "Prioritization based taxonomy of devops security challenges using promethee," *IEEE Access*, vol. 8, pp. 105426–105446, 2020. cited By 0.

[14] R. Kumar and R. Goyal, "Modeling continuous security: A conceptual model for automated devsecops using open-source software over cloud (adoc)," *Computers Security*, vol. 97, p. 101967, 2020.

[15] A. Brunnert, A. van Hoorn, F. Willnecker, A. Danciu, W. Hasselbring, C. Heger, N. Herbst, P. Jamshidi, R. Jung, J. von Kistowski, A. Koziolek, J. Kro, S. Spinner, C. Vgele, J. Walter, and A. Wert, "Performance-oriented devops: A research agenda," 08 2015.

[16] V. Casola, A. De Benedictis, M. Rak, and U. Villano, "A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach," *Journal of Systems and Software*, vol. 163, p. 110537, 2020.

[17] V. Mohan, L. Ben Othmane, and A. Kres, "Bp: Security concerns and best practices for automation of software deployment processes: An industrial case study," pp. 21–28, 2018. cited By 0.

[18] X. Larrucea, A. Berreteaga, and I. Santamaria, "Dealing with security in a real devops environment," *Communications in Computer and Information Science*, vol. 1060, pp. 453–464, 2019. cited By 1.

[19] S. Anand, E. Burke, T. Chen, J. Clark, M. Cohen, W. Grieskamp, M. Harman, M. Harrold, and P. McMinn, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013. cited By 214.

[20] D. Flemstrm, P. Potena, D. Sundmark, W. Afzal, and M. Bohlin, "Similarity-based prioritization of test case automation," *Software Quality Journal*, vol. 26, no. 4, pp. 1421–1449, 2018. cited By 1.

[21] D. North, "Behavior modification: The evolution of behavior-driven development," *Better Software*, vol. 8, no. 3, 2006.

[22] L. P. Binamungu, S. M. Embury, and N. Konstantinou, "Maintaining behaviour driven development specifications: Challenges and opportunities," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 175–184, March 2018.

[23] A. S. Nezhad, J. J. Lukkien, and R. H. Mak, "Behavior-driven development for real-time embedded systems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 59–66, Sept 2018.

[24] D. C. Marinescu, *Cloud Computing: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2013.

[25] C. Boettiger, "An introduction to docker for reproducible research," vol. 49, pp. 71–79, 2015. cited By 206.