An Enhanced CICD Pipeline: A DevSecOps Approach

Olumide Bashiru Abiola
Beechnet Solutions Limited
2967 Dundas Street West,
#724D, Toronto, Ontario M6P 1Z2, Canada

Olusola Gbenga Olufemi
Hood College
401 Rosemont Ave
Frederick, MD, U.S.

ABSTRACT
This paper presents approaches to integrating more DevSecOps techniques into the CICD pipeline, how this can be done, and the benefits of the approaches. Subsequently, unique challenges software development is facing are identified, and some elaborate solutions to these problems are proposed. This work will enable organizations involved in software development, practicing agile enterprise application development to acquire more insight into securing CICD pipelines, which lead to fast and better releases. Secured DevOps as this can be termed, is DevSecOps in a nutshell, which is progressively becoming the only feasible solution to the many challenges organizations face with CICD handling [12]. In its simplest expression, DevSecOps is the process of providing a security enhancement to DevOps. Undoubtedly, attacks are on the rise, and threat actors are not resting, but are getting stronger at hacking targets willingly. Hence, organizations practicing CICD need to know how to combat this menace by getting better at incorporating effective security techniques and doing this very fast. This is exactly what DevSecOps helps to accomplish – providing necessary insights on better and stronger security techniques incorporation and execution [11]. Security moved closer to development and operations in the SDLC i.e., DevSecOps has brought so much dividend to software development [12]. The flexibility, know-how, and capacity to accomplish CICD security techniques are lacking in many organizations engaging in CICD practices [12]. Hence, DevOps transitioned to DevSecOps – is as much about culture as it is about the tools and processes that enable the rapid, frequent, and safe delivery of software [2]. When applying DevSecOps practices, the development lifecycle iterates frequently, providing the team with regular feedback on how safe software is, its behavior, and its usage in the real world. In total, incorporating security into CI/CD pipeline provides regular feedback on the security of the application, and can improve application functionality capacity.

General Terms
Security, Software, Code, Open-source, Cybercrime, SDLC

Keywords
AWS, CICD, DevOps, DevSecOps, Pipeline, Repository, CodeCommit, GitHub, Jenkins, Kubernetes, Ansible

1. INTRODUCTION
Though DevOps is made up of tools and technologies, DevOps also tends to inform and promote teamwork. One of the essential parts of DevOps is automation, which is achieved through tools and technologies that bridge gaps between teams and technologies [11]. Automation has reduced manual intervention on most DevOps processes and made these much faster. Automation is an embedded component of the application security strategy that should be embraced by all organizations involved in CICD, to minimize the surface of exposure and the risk of any compromise [9]. Many DevOps teams are devoid of DevSecOps knowledge and capability to implement security at multiple CICD points, to preserve teamwork, agility, and speed, and this put the whole CICD pipeline at great risk [3].

The agile development methodology is helping development teams complete projects on time and within budget. Moreso, it helps to improve communication between the development team and the product owner. Additionally, Agile development methods reduce the risks associated with complex software development projects [11]. When vulnerabilities are discovered in software provided through agile development, the participating parties usually point to inadequacy in security competence in DevOps teams [11].

Inadequate security competence in DevOps is a result of poor security testing, which leads to insecure systems. Cybercrime is growing at an alarming rate year in and year out. Nevertheless, there should be no fear in software development practices if adequate DevSecOps measures are put in place to check cyber criminals. It is indeed a general concept that modern security practices are the surest way to keep up with the agility and speed of DevOps.

In addition, the application of automation principles in DevSecOps is as helpful as it is in DevOps. Key tools to be put in place as part of the automation strategy to protect the software properly are as follows [9]:

a. Static Application Security Testing (SAST) tool
b. Software Composition Analysis (SCA) tool
c. Dynamic Application Security Testing (DAST) tool
d. Interactive Application Security Testing (IAST) tool
e. Runtime Application Self-Protection (RASP) tool
f. Web Application Firewalls (WAF) tool

DevSecOps can as well be described as incorporating security processes and practices in software development processes, thereby helping the SDLC team in ensuring the proper security of developed products [11]. Invariably, DevSecOps makes everyone in the SDLC team with different roles an important software security personnel. From the looks of the overall narratives, security testing becomes an essential part of each stage of DevOps pipelines. To make the CICD pipeline more viable and secure, these practices need to be ensured in CICD pipeline [10]:

a. Being able to run unit, functional, integration, security, etc. tests in separate jobs such that they can be run in parallel.
b. Being able to test multiple versions simultaneously.
c. Each commit to the remote repository should trigger a pipeline process.
d. Only anything necessary should be built.
e. Availability of the capacity to provide a method to configure the deployment strategy being used.

f. There should be clear test results in case of a pipeline failure.
g. Ability to select different tools for different projects allows DevOps teams to reuse the knowledge that has already been acquired by the team.

2. CICD Overview

CI/CD can be viewed as a pipeline since code is committed on one end, goes through tests performed over stages (source, build, staging, and production), and finally is made a viable code for release. Building a CI/CD pipeline should be a constant routine. Like the software under development, CI/CD practices also involve iterative approaches to keep analyzing data and waiting for feedback to refine the CI/CD process. A stage in CI/CD pipeline serves as a logical unit in the delivery process [10]. Each stage is a doorway that checks the overall code. The quality of the code becomes better in the later stages since many parts of the code would have been scrutinized. The code is stopped from progressing through the pipeline as problems are discovered at an early stage. All further builds and releases are stopped if the software does not pass the test stages. Hence, by building the solution and running a set of automated tests whenever a change is committed, a CI/CD pipeline provides rapid feedback to developers about their changes, and this helps make the code more worthwhile [10]. In a nutshell, CI/CD pipelines are workflows for taking source code through stages like building, functional testing, security scanning for vulnerabilities, packaging, and deployment supported by automated tools with feedback mechanisms [4]. The primary goal of modern software development teams is to build, test, and deploy applications rapidly and reliably [14].

3. CICD – The Makeups

3.1 Continuous Integration - CI

CI/CD starts with CI; hence expertise is needed in continuous integration. Making sure all source code, configuration files, scripts, libraries, and executables are in source control is an essential first step in implementing CI, enabling you to keep track of every change. Developers should regularly commit their code to a central repository (such as one hosted in CodeCommit or GitHub) and merge all changes to a release branch for the application [10]. No developer should keep code in a separate location from the central repository. Feature branches needed for a particular time frame should be updated from the remote repository by merging as often as possible. As code is pushed to a branch in a source code repository, the builder tool builds the code and runs the unit tests in a controlled environment, with the help of a workflow engine monitoring that branch.

Other quality checks, such as unit test coverage, style check, and static analysis, can happen at this stage as well. Finally, the builder tool creates one or more binary builds and other artifacts, like images, stylesheets, and documents for the application. However, when you get CI right, not only will security be an integral component of your entire SDLC, but it will also transform how your organization thinks and implements security [12]

3.2 Continuous Delivery - CD

Continuous delivery (CD) is the next phase and entails deploying the application code in a staging environment, which is a replica of the production stack and running more functional tests. The staging environment could be a static environment made in advance for testing. In the deployment/delivery pipeline sequence, after the staging environment, is the production environment, which is also built using infrastructure as code (IaC) like Terraform or AWS CloudFormation. The CD provides numerous benefits for your software development team, including automating the process, improving developer productivity, improving code quality, and delivering updates to your customers faster.

3.3 Continuous Deployment - CD

The final phase in the CI/CD deployment pipeline is continuous deployment, which may include full automation of the entire software release process including deployment to the production environment. In a fully mature CI/CD environment, the path to the production environment is fully automated, which allows code to be deployed with high confidence. Feedback about the pipeline is continuously collected and improvements in speed, scale, security, and reliability are achieved if collaboration between the different parts of the development team is involved [11].

3.4 Title and Authors

Olumide Bashiru Abiola - CISSP, CISA, CISM, CRISC | Olumide currently serves as a Project Manager for one of the top 5 banks in Canada and is an independent Cybersecurity Consultant. Olumide draws upon his experience in the financial and technology industry after holding a myriad of other positions in System, Network, and Infrastructure Administration. He is responsible for clients' end-to-end cybersecurity programs, coordinating cybersecurity efforts within the enterprise. He is an active member of ISACA Toronto.

Olusola Gbenga Olufemi – Olusola is experienced in Distributed Systems Implementations, Cloud Application & Infrastructure Provisioning, and Information Security Management. He is an active member of ISACA, PMI, and ACM. Olusola is known by colleagues as a life-long learner.

3.5 Implementing Continuous Integration & Continuous Delivery

CI/CD can be seen as a pipeline, looking at figure 1.0 and figure 2.0, where new code is submitted on one end, tested over a series of stages (source, build, staging, and production), and then published as production-ready code [10]. Each stage of the CI/CD pipeline is structured as a logical unit in the delivery process. In addition, each stage acts as a gate that vets a certain aspect of the code. As the code progresses through the pipeline, the assumption is that the quality of the code is higher in the later stages because more aspects of it continue to be verified. Problems uncovered in an early stage stop the code from progressing through the pipeline. Results from the tests are immediately sent to the team, and all further builds and releases are stopped if the software does not pass the stage. Some stages can be repeated several times for testing, security, and performance. Some stages can be repeated several times at different levels; however, this relies on the complexity of the project and the team's structure.

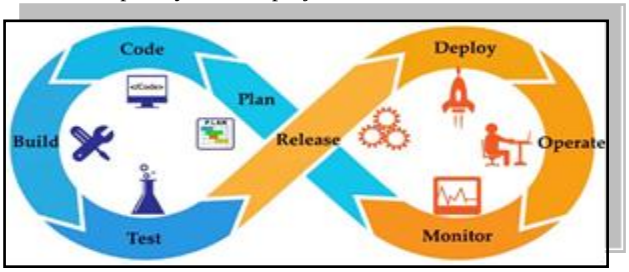

Fig 1.0 CICD Workflow – Source: [16]

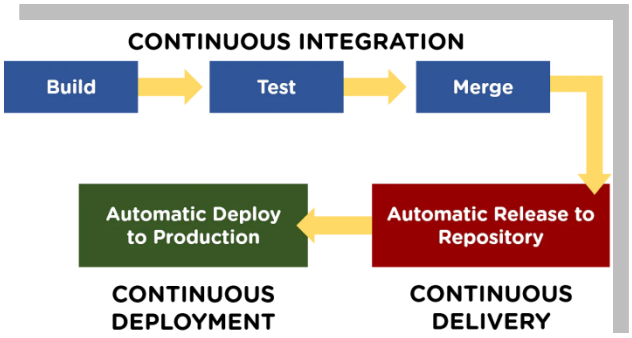

Fig 2.0 CICD Pipeline – Source: [16]

3.6 Security in Continuous Integration

Developers need to create unit tests as early as possible for any code written and run these tests before the code is pushed to the central repository. Any problem seen early in the software development process is still the cheapest and easiest to fix. Developers are encouraged to check their code in a central repository often since this helps catch errors early and expedite the whole development cycle [13]. There should be a suitably sized build process handling all activities, like pushes and tests that might happen during the commit stage, to get fast feedback [10]. Hence, with CI, developers share changes by committing code to source control regularly, at least once a day. They also check that the code builds and passes tests. If something breaks, then there are far fewer changes to go through to find the source of the problems encountered.

3.7 Security in Continuous Delivery

Continuous delivery (CD) is the next phase and entails deploying the application code in a staging environment, which is a representation of the production stack, and running more functional tests. The staging environment could be a static environment that is prepared for testing or could be a provisioned and configured dynamic environment with committed infrastructure and configuration code for testing and deploying the application code [4]. In the deployment/delivery pipeline sequence, after the staging environment, is the production environment, which is also built using infrastructure as code (IaC). Hence, in continuous delivery, each successful build is automatically deployed to each of the pre-production environments, with the mindset that quality is increasing with every stage.

3.8 Continuous Deployment & CICD Growth

The final phase in the CI/CD deployment pipeline is continuous deployment, which may include full automation of the entire software release process including deployment to the production environment. In a fully mature CI/CD environment, the path to the production environment is fully automated, which allows code to be deployed with high confidence. As such, codes will automatically be released to production if a build passes all previous stages in the pipeline successfully. This means software gets delivered to the users as soon as any change to the software has passed all tests. Continuous deployment fast-tracks the feedback loop from code changed to code used in production [4]. The team happily has quick insight into how changes made perform in the actual sense, without compromising on quality.

Although automating the deployment of software to production is not suitable for every product and organization, it's worth considering the steps required to get there as each element is valuable on its own. The organization will surely experience growth, and will continue to improve the CI/CD model by including more of the following [10]:

- Adding more staging environments for specific performance, compliance, security, and user interface (UI) tests
- Exploiting unit tests of infrastructure and configuration code along with the application code
- Integration with other systems and processes such as code review, issue tracking, and event notification
- Integration with database schema migration (if applicable)
- Additional steps for auditing and business approval

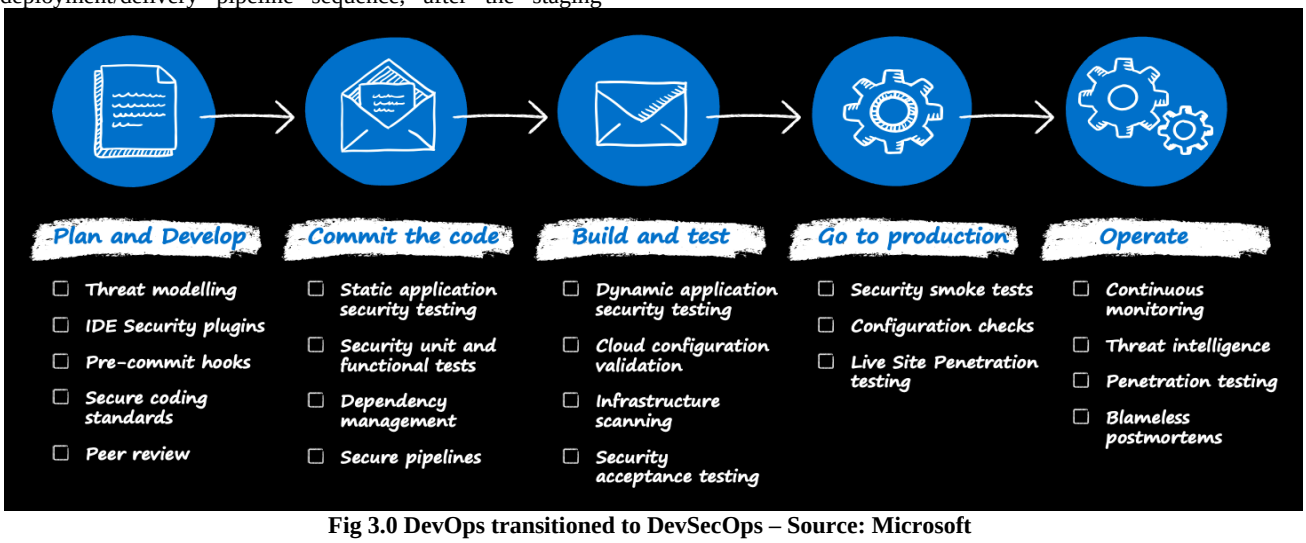

Fig 3.0 DevOps transitioned to DevSecOps – Source: Microsoft

4. TESTING STAGES IN CICD

Adopting DevSecOp approaches in CICD delivers several benefits, including an overall improvement in speed, agility, and security across the organization's entire SDLC (including all application development pipelines). DevSecOps also delivers the potential for rapid changes to existing processes, faster and more secure software development, and an increased speed to market. (DR). The Unit tests are at the bottom of the pyramid of testing stages, they are both the

fastest to run and the least expensive. Therefore, unit tests should make up the bulk of the testing strategy, recommended rule of thumb is about 70 percent [10]. Unit tests should have near-complete code coverage because bugs caught in this phase can be fixed quickly and cheaply.

Service, component, and integration tests are above unit tests on the pyramid. These tests require detailed environments and therefore, are more costly in infrastructure requirements and slower to run. Performance and compliance tests are the next levels, they require production-quality environments and are more expensive yet. UI and user acceptance tests are at the top of the pyramid and require production-quality environments as well. All these tests are part of a complete strategy to assure high-quality software development. However, for speed of development, the emphasis is on the number of tests and the coverage in the bottom half of the pyramid. The following sections discuss the CI/CD stages:

4.1 CICD Stages

These include setting up the source, setting up and running builds, staging, and production, more elaboration as follows [10]:

4.1.1 Source code setting up

A source where one can store raw code should be set up, and configuration and schema changes done. In the source stage, a source code repository should be chosen, like the one hosted in GitHub or AWS CodeCommit.

4.1.1.1 Setting up builds and running

Choosing the right build tool is the first task when setting up build automation. There are lots of build tools, such as Ant, Maven, and Gradle for Java, Make for C/C++, Grunt for JavaScript, and Rake for Ruby [11].

4.1.1.2 Building

The build tools will take as input any change to the source code repository, build the software, and run the following types of tests:

a. **Unit Testing –** Unit testing is performed by software developers during the development phase. At this stage, a static code analysis, data flow analysis, code coverage, and other software verification processes can be applied.
b. **Static Code Analysis –** This test is performed without executing the application after the build and unit testing. This analysis can help to find coding errors and security holes, and it also can ensure conformance to coding guidelines.

4.1.1.3 Staging

Full environments are being created that mirror the real production environments. Tests performed in staging are as follows:

a. **Integration testing** – Verifies the interfaces between components against a software design. Integration testing is an iterative process and facilitates building robust interfaces and system integrity.
b. **Component testing** – Tests message passing between various components and their outcomes. A key goal of this testing could be unchangeability in component testing. Tests can include extremely large data volumes, or edge situations and abnormal inputs.
c. **System testing** – Tests the system end-to-end and verifies if the software satisfies the business requirement. This might include testing the user interface (UI), API, backend logic, and end state.
d. **Performance testing –** Determines the responsiveness and stability of a system as it performs under a particular workload. Performance testing is also used to investigate, measure, validate, or verify other quality attributes of the system, such as scalability, reliability, and resource usage. Types of performance tests might include load tests, stress tests, and spike tests. Performance tests are used for benchmarking against predefined criteria.
e. **Compliance testing –** It determines if you are implementing and meeting the defined standards.
f. **User acceptance testing –** This testing is executed by an end user in a staging environment and confirms whether the system meets the requirements of the requirement specification. Typically, customers employ alpha and beta testing methodologies at this stage.

4.1.1.4 Production

Finally, after passing the previous tests, the staging phase is repeated in a production environment. In this phase, a final *Canary test* can be completed by deploying the new code only on a small subset of servers or even one server, or one AWS Region before deploying code to the entire production.
Environment.

4.2 Pipeline Building

Starting with aggregated automation of builds, environment creation, and testing reduces the time from development to release while providing confidence in the quality of the product for consumption. The stages of a CI/CD pipeline encompass performance tests, and manual testing, these require test environments that reflect production. When these environments are refreshed automatically, maximum efficiency and consistency of testing are derived. However, automation testing is ideally needed for repetitive tasks and produces more consistent results than manual testing [1]. To accomplish this efficiency and consistency of testing, DevOps skills, tools, several infrastructures compute capacities for the CI server, build agents, test environments and data stores are required. The size, complexity of the project, and the number of developers contributing to it determine the number of machines required to support the pipeline. The implementation of CI/CD processes can take place with adequate automation in three segments. Firstly, Continuous Integration, which involves development and testing; secondly Continuous Delivery - extending CICD with automated integration testing; and lastly adding Continuous Deployment to a production environment [1]. Start the pipeline with components needed for CI, then move to a continuous delivery pipeline with added components and stages [10]. Lastly, the pipeline deploys the code to a production environment, after the acceptance and security tests have passed and the testing version has been approved to be deployed [10].

4.3 The Gains of Building a Secured CICD

DevSecOps culture is a culture where overall security is germane within an organization CICD. Teams that are traditionally reluctant to change work closely together with DevSecOps, hence fostering stronger and better working relationships. Additionally, it shortens development cycles and faster development. It also improves flexibility and scalability since this provides both a rapid response capability and the means to adapt to change much easier. This allows visibility, traceability, and accountability – being able to see who made what changes and when is excellent for bug tracking, awareness, and accountability. The ability to address and alleviate errors at the source, as they arise, enables one to

build better software, and release it faster, and with fewer bugs and vulnerabilities. This not only results in greater cost savings but also minimizes overall disruption, improves workflow, helps build better software, and improves overall security. Transitioning an organization CICD to the DevSecOps CICD model doesn't happen instantly, it's both a strategic and continual improvement process [7]. To end up with secured software production, each phase (initiation, development, deployment, maintenance, and disposal) of software development needs to be protected from malicious attacks [6]. Creating software with fewer bugs and vulnerabilities, that is detecting and eliminating them at source, reduces time to market for finished products.

4.4 AWS CICD Tools & Capacities' Idea

CI/CD tools play a central role in coordinating and automating the various stages of the pipeline, from kicking off the process following a commit to managing the build, triggering automated tests, publishing artifacts, and collating and relaying feedback. Choosing the right tool for an organization will help realize the benefits of CI/CD and allow it to keep evolving and improving software development processes. Hence, tools should be able to provide integration options, tech stack support, and interfaces for everyone, should be customizable, provide feedback, provide infrastructure options, improve CICD performance, provide support, provide metrics, and provide security. Using the appropriate tools in CICD does not fully secure the pipeline but reinforces it to a certain level [6]. Securing pipelines against malicious actors should therefore be the highest priority. This should include managing secrets securely, applying the principle of least privilege, and monitoring activity across the pipeline. Investigating any unusual behavior should also be made possible by tool providing an audit trail. AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy, and AWS CodeArtifact are some of the AWS developer tools or services offering CI/CD tools' capabilities. In addition to these tools, AWS also provides security services such as Config, CloudTrail, CloudWatch, and custom Lambda services that incorporate the DevSecOps CICD [7]. These services rapidly and securely deliver software and are readily made available to Developers and IT operations engineers practicing DevOps. When services are in the stack, these services securely store and apply version control to the source code. AWS CodeStar or CodePipeline can be used to rapidly orchestrate an end-to-end software release workflow, together with other services [10]. AWS CodePipeline has the flexibility to integrate each service independently with other existing tools, for an existing environment. AWS Management Console, AWS application programming interfaces (APIs), and AWS software development toolkits (SDKs) are the means to access these AWS services, which are highly available, easily integrated services [10]. Other CICD tools like Jenkins, Kubernetes, Docker, Chef, Gradle and Ansible can also be integrated with AWS CICD tools to enhance CICD functionalities. The authors will discuss details about these tools in a follow up publication.

4.5 Conclusion - Securing the CICD Pipeline

In conclusion, the following security tasks should be practiced to the minimum for securing the CI/CD pipeline [4]:

1. Ensuring hardening servers hosting code & artifact repositories
2. Securing the credentials for accessing repositories like authorization tokens and for generating pull requests.
3. There should be controls on who can check in and checkout in container image registries since they are the storage for artifacts produced by the CI pipeline and serve as bridges between CI and CD pipelines.
4. Logging all code and build update activities.
5. Testing is carried out in all stages when necessary.
6. Assigning a security champion helps in defining clear project requirements, the pipeline you must protect, real-time feedback on pipelines, and providing adequate resources.

To round up this writeup, a CI/CD pipeline that is insecure will certainly lead to poor application production. Hence, by taking control of security risks at all stages of the pipeline via practices like access control, secrets management, registry scanning, and testing, surely the risk that insecure code will flow down the pipeline can be minimized. Measuring and comparing the DevSecOps capabilities of AWS CICD tools to other comparable open-source CICD tools are presently being researched. It is hoped that the outcome of the research will show the strength of each of these comparable CICD tools.

5. ACKNOWLEDGMENTS

Praise be to God for grace and making this happen. Many thanks also to families and friends for the inspiration and support all these years in this endeavor.

6. REFERENCES

[1] Philippa Ornell. August 20. 2020. Kth royal institute of technology school of electrical engineering and computer science. Security Assessment of Continuous Deployment Pipelines. https://www.diva-portal.org/smash/get/diva2:1471199/FULLTEXT01.pdf

[2] Bakary Jammeh. 2020. DevSecOps: Security Expertise a Key to Automated Testing in CI/CD Pipeline. https://www.researchgate.net/publication/347441415_DevSecOps_Security_Expertise_a_Key_to_Automated_Testing_in_CICD_Pipeline

[3] Justine Goldmith. 2000. Security first: Automating CI/CD pipelines and policing applications. https://events.redhat.com/accounts/register123/redhat/events/701f20000012gfuaay/Security_First_Security_Symposium_2019.pdf

[4] Ramaswamy Chandramouli. March. 2022. Implementation of DevSecOps for a Microservices-based Application with Service Mesh. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf

[5] Thorsten Rangnau. 2020. 2020 IEEE 24th International Conference on Enterprise Distributed Object Computing (EDOC). Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. https://www.researchgate.net/publication/346379276_Continuous_Security_Testing_A_Case_Study_on_Integrating_Dynamic_Security_Testing_Tools_in_CICD_Pipelines

[6] Faheem Ullah. 2017. Security Support in Continuous Deployment Pipeline. https://arxiv.org/ftp/arxiv/papers/1703/1703.04277.pdf

[7] AWS & Delloite. 2019. Integrating and automating security into a DevSecOps model.

https://www2.deloitte.com/content/dam/Deloitte/us/Documents/risk/us-integrating-and-automating-security-into-a-devsecops-model.pdf

[8] Geoffrey Sanders. 2021. Software Engineering Institute, Carnegie Mellon University. July White Paper. INTEGRATING ZERO TRUST AND DEVSECOPS. https://apps.dtic.mil/sti/pdfs/AD1145432.pdf

[9] Mike Heim. 2020. National Defense-ISAC Publication. Software Security Automation: A Roadmap toward Efficiency and Security. https://ndisac.org/wp-content/uploads/ndisac-security-automation-white-paper.pdf

[10] AWS Whitepaper. 2021. Practicing Continuous Integration and Continuous Delivery on AWS. https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/welcome.html

[11] Rangnau T. et al. Continuous Security Testing: 2020. A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. https://ieeexplore.ieee.org/abstract/document/9233212

[12] Guardrails whitepaper. 2023. https://www.guardrails.io/whitepapers/how-to-build-a-devsecops-pipeline/

[13] Akhil Jain. 2021. https://aws.plainenglish.io/devops-102-lifecycle-and-ci-cd-b18923240d49

[14] Daniel Pohl. 2020. https://www.logicworks.com/blog/2020/10/cicd-iac-pipeline-part-1/

[15] Rob Larter. 2020. https://www.linkedin.com/pulse/cicd-what-why-important-rob-larter

[16] Ishan Gaba. 2021. https://www.simplilearn.com/tutorials/devops-tutorial/continuous-delivery-and-continuous-deployment