IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

The Role of DevOps in Streamlining Software Delivery: Key Practices for Seamless CI/CD

Godavari Modalavalasa
Independent Researcher
godavarimdv23@gmail.com

Abstract: *Software development and delivery practice underwent a fundamental change with DevOps implementation because it brings development teams together with operations teams. The purpose of this study is to examine DevOps platform significance in software delivery acceleration through Continuous Integration (CI) together with Continuous Delivery (CD) system integration. The text explores the fundamental approaches that drive automation together with collaboration enhancements and increased efficiency during software development. DevOps implemented automation together with collaboration along with persistent improvement to revolutionize software development through complete clearance of operational and development boundaries. The research investigates how DevOps technology enables easy software delivery by utilizing CI and CD methods. DevOps dismantles traditional organizational silos between development and operations teams to establish open cooperation among employees so they can take collective responsibility for work outcomes. Through deliberate solutions to these issues and DevOps best practice adoption, organizations can create effortless scalable software delivery that matches present technological evolution requirements. The paper provides an in-depth examination of core DevOps principles, including automation, version control, infrastructure as code (IaC), and real-time monitoring, which collectively streamline the software development lifecycle.*

Keywords: DevOps, Continuous Integration (CI), Continuous Delivery (CD), automation, infrastructure as code (IaC), software delivery

I. INTRODUCTION

The introduction of modern processes alongside methods has transformed software development throughout recent years. Software delivery may be made more efficient, faster, and more reliable by using these practices and approaches. The popular approach in this field today is DevOps. The DevOps terminology results from joining the development and operations domains together. DevOps tools enhance employee work environments while decreasing software development time along boosting delivery speeds for software products. DevOps automation enables the performance of fast and agile testing together with deployment operations at large capabilities [1].

Software development methodology DevOps enables joint teamwork between operations staff and developers during application creation. The DevOps framework allows operational teams and developers to form better working relationships while maintaining open trust. The continuous development agenda of Agile finds additional depth through DevOps because this system automates integration along with release operations. DevOps describes an organizational cultural transformation which involves automation of development processes together with infrastructure automation to achieve quality consistency in software products. More emphasis on implementation and more regular releases are two of the many advantages that software organizations may get from DevOps [2]. Through DevOps methodology testing, together with deployment and development cycles, gain automated functionality.

DevOps practices enhance product quality while simultaneously shortening time-to-market wait periods, which form an essential base for business competition. The applicability of DevOps practices on Research and development (R&D) efficiency and source code management (SCM) continues to be undefined, mostly in large enterprise environments[3]. New research investigates the role of DevOps best practices in achieving software delivery success by improving research and development performance along with SCM robustness to demonstrate operational and strategic advantages that benefit enterprises[4].

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

Modern digital technology advancements have created urgent requirements for optimal software delivery within enterprises that generate substantial stress on research departments to fulfill faster production cycles. The traditional framework which develops software requires extensive modifications to handle scalability and flexibility thus inhibits organizations from fast market adaptations. DevOps has become a revolutionary mindset which connects development teams to operations teams for the purpose of process optimization and collaborative work leading to more frequent deployments. As a result of development operations integration DevOps allows organizations to perform continuous integration and deploy continuously (CI/CD) while optimizing resource utilization[5].

A. Structure of the paper

Here is the outline of the paper: Section II provides an overview of DevOps practices in streamlining software delivery; Section III discusses the principles and methodologies of CI and CD; Section IV explores the tools and technologies that enhance CI/CD pipelines, such as Jenkins, Docker, and Kubernetes; Section V presents relevant literature and case studies; and Section VI offers conclusions and recommendations for future research directions.

II. DEVOPS IN STREAMLINING SOFTWARE DELIVERY: AN OVERVIEW

Software development and operations are brought together in the DevOps methodology. In order to carry out its combination and integration function, DevOps requires a collection of tools [6]. To rephrase, a DevOps team coordinates both the development and testing processes as well as the operations side of things. Every step of the product lifecycle remains uninterrupted in DevOps.

The name "DevOps" is a combination of the words "Dev" and "Ops." The acronym "Ops" refers to operations, whereas "Dev" stands for development. Its organization was searching for a method that could bridge the gap between the operations and development departments and was superior to existing methods. "Dev and Ops Co-operation" was the first usage of the term "DevOps" at a 2009 conference by John Allspaw. After that, DevOps becomes very important in changing software development, testing, deployment, and maintenance workflows. The DevOps methodology simplifies the software delivery lifecycle by eliminating silos between operations and development teams and placing an emphasis on automation, collaboration, and continuous improvement[7].

The goal and promise of this approach is to better integrate the development, production, and operations business processes with the right technology; to eliminate the need for highly artificial process models; and to provide a continuous delivery process with incremental improvements. Amazon, Google, and Netflix are just a few examples of how this method has been optimized to achieve cycle durations in minutes. While the ease of facilitating software delivery to embedded devices is dependent on the deployment methodology, it is simpler to handle a single cloud service. Figure 1 depicts the DevOps cycle visualizes below:

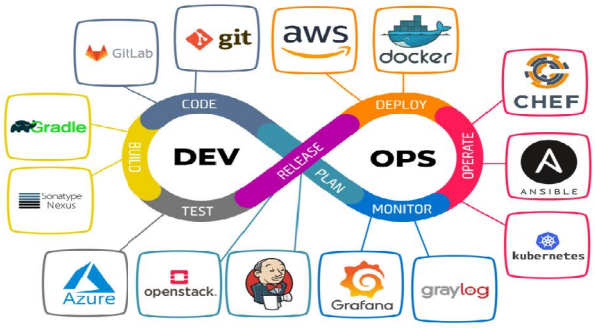

Fig. 1. DevOps Cycle

A. Continuous Integration

The term "DevOps" describes the merging of the "development" and "ops" departments. With DevOps, a single team can handle all of the tasks required to build a product from start to finish since all of the processes are continuously integrated. The main issue with corporate code entering a mutual storeroom a few times a day is inadequate communication. Each check-in serves as confirmation that DevOps must be put into place. Engineers are expected to be

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

validated by a robotized assembly in Continuous Integration, which allows organizations to discover errors at the same moment they occur in the cycle[8].

B. Continuous Delivery and Deployment

Automated and trustworthy software releases were made possible by the principles of Continuous Delivery and CD, which were essential to DevOps. The result was a shorter time between committing code and deploying it to production [9]. Important parts of persistent delivery include establishing a standard for base setup and controlling arrangement details in the same way as controlling source code.

C. Continuous Operation

Implementing and overseeing updates to both software and hardware while keeping end users unaffected. This department is responsible for tasks including patching and compliance. Planned maintenance may take servers and software down, but continuous operations ensure that users continue to use the older version of the program until the new one is tested and released [8].

D. Continuous Assessment

Assessing an application continuously in light of three different types of criticism

- Feedback circles: Continuously assessing the application's usability, execution, and accessibility; also, recording the customer experience throughout its life cycle (i.e., development, quality assurance, organization, and generation) and supporting it back to the appropriate groups. By doing this, you enable continuous optimization and modification of the application and end-client experience.
- Planning prioritization: Business needs and end-user requests inform the planning group's ongoing evaluation and organization of new features, capabilities, and defect settlement.
- Portfolio venture: Constantly surveying and organizing guesses in light of business drivers is possible as the planning group receives input.

E. Benefits of DevOps in Software Delivery

More responsive development environments allow it to meet business requirements and eliminate human error from project lifecycles. By streamlining processes, DevOps delivers numerous benefits to organizations:

- Shorten the months it takes to implement new services to just a few minutes.
- Faster Time-to-Market: Speeds up and minimizes delays in the rollout of updates and new features.
- Boost the efficiency of both the business and IT departments.
- Reduce expenditures for maintenance and upgrades, and avoid unnecessary capital expenditure.
- Standardize procedures for quicker delivery and simple replication [10].
- All system components should be made better in terms of quality, reliability, and reusability.
- Make digitalization initiatives and transformation efforts more successful.
- Make sure that the funds allocated for cloud storage, analytics, and data management are not going to waste.

III. CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY IN DEVOPS

Even though it has been around for a while, CI/CD is now a must for most software development jobs. Integrating CI/CD into software development processes has greatly improved the speed and reliability of software creation, testing, and deployment [9].

Software development lifecycle automation and streamlining is the goal of DevOps, which aims to automate and streamline CI and CD to enable frequent, dependable, and fast delivery of high-quality software. The subject is explained here.

A. Continuous Integration

The software development industry has long used the concept of Continuous Integration (CI) to describe the process by

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

which a team's development work (e.g., code) is continuously integrated and merged, sometimes several times a day. CI provides software organizations the advantage of better-quality software alongside reduced release time and more productive development teams. As part of this process, automated software development and testing are encompassed. Software engineers using this approach often merge their work into a central repository before automating builds and testing. CI often denotes the build or integration phase of software release and necessitates the use of both manual and automated processes[11]. The procedure for continuous integration is illustrated in Figure 2.

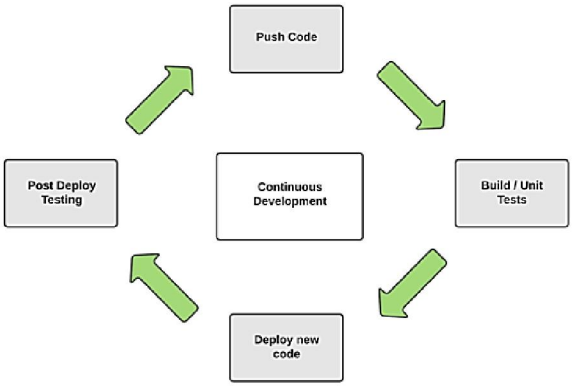

Fig. 2. Continuous Integration Process

CI aims to speed up the process of detecting and correcting bugs, improve software quality, and reduce the time it takes to verify and deliver new software updates. With continuous integration, the emphasis is on merging smaller code changes and commits. Code commits occur frequently, at least once daily, for developers. The developer gets code from the code repository, integrates it on the local host, and then uploads it to the build server. The build server then determines whether to approve or reject the code modification based on the results of the conducted tests. A single source code repository, more frequent updates to the shared codebase, automation of builds, and automation of tests are the primary problems of implementing CI [12].

a. Benefits of Using Continuous Integration

The following are just a few of the many ways in which your company might profit from continuous integration in the context of software delivery:

1. Reduced integration risk

Having more than one person working on different aspects of a project or piece of code is commonplace. Integration becomes riskier as the population grows. Depending on the severity of the problem, fixing it and debugging it might be a big nuisance and need a lot of code modifications.

2. Higher code quality

A better product is the outcome of not worrying about the issues and instead concentrating on the code's usefulness.

3. The quality-of-life improvement for testers

Maintaining several code versions and builds facilitates faster bug isolation and tracing and saves time for the quality assurance team.

4. version control

You and your team are notified right away if you make a commit that breaks the build, and the issue is resolved before anybody else pulls the "broken" code.

5. Less time deploying

Making projects live requires a lot of time and effort. Thus, it makes sense to automate that process [13].

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

B. Continuous Delivery

The goal of the software engineering methodology known as "continuous delivery" is to guarantee the reliable deployment of software at any moment by encouraging teams to consistently produce high-quality code in short iterations. The concept of CDs is a recent development. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" by Humble and Farley was published in 2010, which is when it first gained widespread recognition. The CD method, on the other hand, has been trending up in Google search volume recently [6]. Companies that utilize CD have been said to have a lot of benefits, such as a shorter time to market, better goods, more consistent releases, greater efficiency and productivity, and the ability to test and make the best product quicker[14].

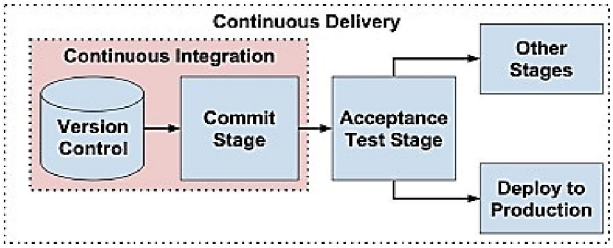

Fig. 3. Continuous Delivery process

Automating the release process and conducting continuous integration tests to ensure production-quality software are two ways in which CD expands CI. Figure 3 further emphasizes the distinction between CI and CD, demonstrating that whilst CI only includes one step, CD includes several steps that confirm whether the software is in a release-ready state[15].

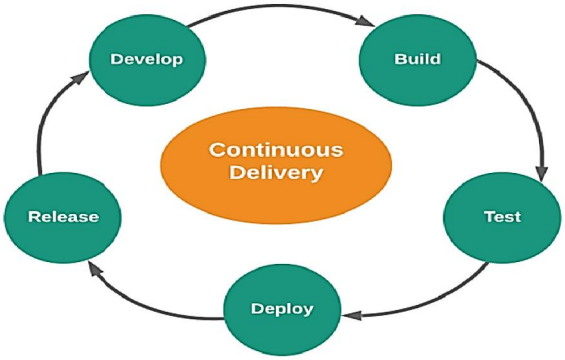

Fig. 4. Continuous Delivery

Figure 4 shows a crucial approach for improving the reliability and speed of software development is continuous delivery. In order to facilitate automation and improvement, the developer is provided with feedback from the Operations and Production teams at frequent phases. Full automation is not achieved with CDE. That the application could be deployed is all it means. Making decisions is obviously a waste of time. On the other hand, CD implies that the code for the application may be automatically deployed to production without the need for human interaction [16]. This enables early and regular feedback while saving a significant amount of time in software development.

a. Benefits of Continuous Delivery

Among the many advantages that CD offers your software development team are process automation, increased developer productivity, enhanced code quality, and update delivery.

1. It Automates the Software Release Process

Continuous delivery makes software delivery more efficient and quicker by enabling us to automatically generate, evaluate, and get code changes ready for production release.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

2. It Improves the Developer's Productivity

It increases their team's productivity by relieving developers of non-automatic chores and influencing processes that reduce the number of errors and problems reported to customers.

3. It Checks and Addresses Bugs Faster

Before the bugs become a major headache, their staff can quickly identify and fix them. Because the entire process is automated, continuous delivery enables us to test their code in many ways [17].

4. It Quickly Delivers the Updates

It lets us update you more often and with greater speed. Assuming everything goes as planned, they will perpetually maintain a standard deployment-ready-built monument that has been tested and found to be successful.

IV. DEVOPS PRACTICES FOR CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT

The creation of software has evolved to prioritize the timely and effective delivery of high-quality products throughout its lifecycle. The significance of combining CD with CI techniques has grown in this setting. A goal of CI is to automatically and regularly incorporate code changes into a common repository. This idea is expanded upon by continuous deployment, which makes it possible to distribute software to end users quickly and reliably. These methods are foundational to the DevOps paradigm, which arose in reaction to the necessity to merge the Dev and Ops processes between 2007 and 2010.

DevOps works to automate development processes, which enables frequent, reliable delivery of high-quality software through its core component of CI/CD.

A. Here are the key concepts of DevOps and CI/CD practices:

This highlights the practices of DevOps and CI/CD:

- Version Control: Implementing Git as a version control system improves tracking and management of code alterations. Your branching approaches and commit rules must be established clearly to prevent codebase deterioration.
- Automated Testing: The automated testing system must contain unit tests along with integration tests and end-to-end tests. Testing procedures must test fundamental functionalities as well as extreme situations and performance standards to verify code quality.
- Build Automation: Automate the build process to generate executable code from source code automatically. A consistent process for code compilation, as well as dependency packaging and deployable artifact generation, runs throughout development.
- CI/CD pipeline: A CI/CD pipeline enables automated software development operations, which speed up the process. Developers may more easily incorporate changes into the project as it enables organizations to build, test, and deploy software projects constantly[12]. Jenkins serves as the open-source CI/CD pipeline throughout the research because it handles multiple languages and code repository types.
- Deployment Automation: The deployment process should run automatically to eliminate manual tasks that often lead to errors. Applications should be deployed through scripts and tools that establish uniform deployment across multiple environments.
- Infrastructure as Code (IaC): The usage of "Infrastructure as Code" describes a system that manages infrastructure by defining it with programmed technologies and automated code execution. IaC enables developers to manage infrastructure through code treatments so they can execute infrastructure changes with automated testing and versioning.
- Use Infrastructure as Code principles to handle infrastructure configurations via automated code management. Enhance consistency and scalability by versioning, automating, and replicating infrastructure changes using tools like Terraform and Ansible.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

- Monitoring and Feedback Loops: Incorporate logging and monitoring technologies that reveal the status and performance of applications in real time. Create feedback loops that alert the team to issues, allowing for rapid response and continuous improvement.
- Incremental Changes and Feature Toggles: Break down development tasks into small, incremental changes. Utilize feature toggles or feature flags to allow or prohibit new features in production, offering flexibility and the opportunity to undo modifications in the event that problems occur [18].

B. Tools for DevOps and CI/CD Practices

The DevOps toolchain catalogs all the resources that contribute to and facilitate DevOps processes across the software development and delivery lifecycle.

- Plan: To ensure the production of a top-notch product, the first stage in the DevOps lifecycle is to have a complete understanding of the project's requirements. Google Apps, Jira, Confluence, Slack, and many more are among the available technologies.
- Code: Developing the project includes building the system architecture, coding features, and implementing automated procedures and test cases as part of the DevOps lifecycle. IDEs like Eclipse and IntelliJ are common, as are project management systems like Jira.
- Build: The application is really built by the integration of several programs developed in the previous steps. Commonly used tools encompass Sonatype Nexus, Gradle, Maven, and others.
- Test: Developing the application reaches a critical point at this point. This process includes thoroughly testing the application and, if needed, rebuilding it from scratch. To conduct the tests, they used Junit, Maven, and Sonarqube[19].
- Deployment: The goal of DevOps is to automate the process of activating features by streamlining the automation of deployment using scripts and tools. Docker, Azure, AWS, Heroku, etc., are among the most popular tools. They deployed using Kubernetes clusters that are based on Docker.
- Operations: This stage entails adjusting to the dynamics of the infrastructure, which presents chances to improve availability, scalability, and product transformation. The following tools are frequently utilized during this stage: Loggly, Chef, Ansible, Kubernetes, and many more.
- Monitor: An essential part of the DevOps process is continuous monitoring. Its main objective is to keep tabs on data in order to evaluate the health of software programs. Nagios, Splunk, Data Dog, and other similar technologies are commonly used during this phase.
- Source code management (SCM) tools: An approach to managing software's source code is known as source code management. Versioning and remote team collaboration are two of SCM's primary uses [19]. Version control governs developer collaboration during coding and oversees source code modifications. Because it is open-source and freemium, and because it enables distributed systems, GitHub is the most widely used version control application. Mercurial, Bitbucket, and Subversion are more well-known source code tools.

V. LITERATURE REVIEW

This section presents a literature review on DevOps practices for seamless CI and CD. It examines the methodologies, tools, and challenges associated with implementing DevOps in modern software delivery pipelines. Table I provides a brief description of the studies that were reviewed.

Virmani (2015) study focuses on the business requirements, methods for transitioning from CI to continuous delivery, and the advantages of DevOps while attempting to address all elements relevant to the different stages of the software delivery lifecycle. This paper explains continuous delivery transformation using a real-life case study of IAAC maintenance. Finally, this paper discusses the benefits of DevOps and other factors to think about before implementing it [20].

Soni (2016) the goal of this study is to demonstrate how the build pipeline concept can be used to automate various tasks such as code analysis, packaging, deployment, notifications, infrastructure provisioning, and continuous integration, testing, and delivery. Rapid application development and automated deployment are hallmarks of the

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

DevOps culture, which builds on the agile technique to boost performance and quality. With the introduction of CI and CD, traditional approaches to application development and release management have been greatly improved. Clients may now receive high-quality artefacts continuously, with integrated feedback provided at every stage of the process[21].

Steffens, Lichter and Döring (2018) this study provides a model for future software delivery systems and the key decisions made during their design to address these challenges. Two distinct obstacles to CD adoption are the focal point of their strategy. Software distribution systems are not adaptable or easily maintained, which is the first issue. Secondly, there is a lack of user assistance when it comes to managing and modeling delivery pipelines and processes. Modern agile software development approaches include Continuous Delivery and CI. The DevOps movement took this method and made it their own, centering on the deployment pipeline as a means to automate software development, improve software delivery and operation, and boost overall quality[22].

Stahl, Hallen and Bosch (2016) study highlights traceability as an important obstacle to continuous integration and delivery and details Eiffel, a framework created by the industry, to address this issue. Eiffel allows for real-time tracking of integration and delivery processes. Not only is it necessary for software development to be traceable in order to ensure legality and certification, but it is also essential for the development process itself. The industry's traceability processes have not been thoroughly studied, particularly in the areas of tools and infrastructure, and organizations often fail to meet traceability criteria[23].

Eddy et al. (2017) this study examines the results of an exploratory study on the feasibility of incorporating a continuous delivery instructional pipeline into an undergraduate software engineering course. The purpose of the pipeline was to provide a visual representation of the procedures needed to assist teachers in integrating continuous integration and delivery into their existing classes. The demand for training in CI and continuous delivery is rising in tandem with their adoption rates in major industries. Since there is a learning curve associated with the instruments used and limited time for teaching these subjects, introducing them might be challenging[24].

Table 1: Comparative Table Highlighting Key DevOps Practices for Seamless Continuous Integration and Continuous Delivery

Reference	Study On	Approach	Key Findings	Challenges	Limitations
[20]	Transition from CI to CD; Business needs; Benefits	Real-life case study showcasing Infrastructure as Code (IaaC) for CD transformation	Demonstrated how IaaC improves CD transformation and overall DevOps maturity.	Requires cultural change and expertise in IaaC.	Limited to specific case study; lacks generalizability.
[21]	Framework for CI/CD/CT and automation in pipeline	Proof of concept for automating compilation, analysis, testing, packaging, deployment, and notifications	Highlighted benefits of integrating DevOps culture with agile for rapid application delivery and quality improvement.	Managing dependencies and tool complexity.	Focused on conceptual framework, lacks empirical validation.
[22]	Design of modern software delivery systems for CD	Proposed conceptual model addressing flexibility, maintainability, and pipeline management	Emphasized the deployment pipeline as a core component for automating robust, high-quality software delivery.	Flexibility and user support in modeling pipelines.	Limited coverage of practical implementation challenges.
[23]	Traceability in CI/CD through the Eiffel framework	Industry-developed framework for real-time traceability in CI/CD pipelines	Highlighted the importance of traceability for compliance and development efficiency.	Difficulties in tool adoption and infrastructure.	Focused on traceability; limited coverage of broader CI/CD practices.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

[24]	Educational pipeline for CI/CD in software engineering courses	Designed educational CI/CD pipeline to integrate into existing undergraduate courses	Demonstrated the effectiveness of visualizing CI/CD processes for better student understanding.	Learning curve for tools; limited teaching time.	Geared towards educational settings; lacks industrial application focus.

VI. CONCLUSION AND FUTURE WORK

DevOps is a game-changing method for software delivery that helps businesses improve their development operations in terms of speed, reliability, and scalability. Development and operations teams work together more efficiently in a DevOps environment because of the integration of CI and CD processes. The importance of DevOps in improving software delivery efficiency via CI and CD methods has been discussed in this paper. By leveraging automation, version control, infrastructure as code (IaC), and real-time monitoring, DevOps enhances the efficiency, reliability, and scalability of software delivery pipelines. Tools such as Jenkins, Docker, Kubernetes, and Terraform have been instrumental in implementing robust CI/CD pipelines, improving code quality, and accelerating time-to-market.

The incorporation of new technology, such AI and ML, into CI/CD pipelines should be the primary goal of future research. In order to further improve the efficiency of software delivery, these technologies can provide predictive analytics, automated issue detection, and resource optimization.

REFERENCES

[1] L. E. Lwakatare *et al.*, "DevOps in practice: A multiple case study of five companies," *Inf. Softw. Technol.*, vol. 114, pp. 217–230, 2019, doi: 10.1016/j.infsof.2019.06.010.

[2] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Relationship of DevOps to Agile, Lean and Continuous Deployment," 2016. doi: 10.1007/978-3-319-49094-6_27.

[3] M. Senapathi, J. Buchan, and H. Osman, "DevOps capabilities, practices, and challenges: Insights from a case study," in *ACM International Conference Proceeding Series*, 2018. doi: 10.1145/3210459.3210465.

[4] D. Yang *et al.*, "DevOps in practice for education management information system at ECNU," in *Procedia Computer Science*, 2020. doi: 10.1016/j.procs.2020.09.148.

[5] M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," *J. Syst. Softw.*, vol. 170, 2020, doi: 10.1016/j.jss.2020.110798.

[6] A. Mishra and Z. Otaiwi, "DevOps and software quality: A systematic mapping," *Computer Science Review*. 2020. doi: 10.1016/j.cosrev.2020.100308.

[7] P. Batra and A. Jatain, "Measurement Based Performance Evaluation of DevOps," *2020 Int. Conf. Comput. Perform. Eval. ComPE 2020*, pp. 757–760, 2020, doi: 10.1109/ComPE49325.2020.9200149.

[8] P. Jha and R. Khan, "A Review Paper on DevOps: Beginning and More To Know," *Int. J. Comput. Appl.*, 2018, doi: 10.5120/ijca2018917253.

[9] A. Proulx, F. Raymond, B. Roy, and F. Petrillo, "Problems and Solutions of Continuous Deployment: A Systematic Review," 2018.

[10] A. Hasan, "A Review Paper on DevOps Methodology," vol. 8, no. 6, pp. 2583–2589, 2020.

[11] A. Hakli, D. Taibi, and K. Systa, "Towards cloud native continuous delivery: An industrial experience report," *Proc. - 11th IEEE/ACM Int. Conf. Util. Cloud Comput. Companion, UCC Companion 2018*, pp. 314–320, 2018, doi: 10.1109/UCC-Companion.2018.00074.

[12] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*. 2017. doi: 10.1109/ACCESS.2017.2685629.

[13] Yasmine Ska and Habeebullah Hussaini Syed, "A Study and Analysis of Continuous Delivery, Continuous Integration in Software Development Environment," *Researchgate*, vol. 6, no. September, pp. 96–107, 2019.

[14] L. Chen, "Continuous Delivery: Overcoming adoption challenges," *J. Syst. Softw.*, 2017, doi: 10.1016/j.jss.2017.02.013.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 5.731
Volume 1, Issue 2, January 2021

[15] E. Laukkanen, J. Itkonen, and C. Lassenius, "Problems, causes and solutions when adopting continuous delivery—A systematic literature review," *Information and Software Technology*. 2017. doi: 10.1016/j.infsof.2016.10.001.

[16] A. Agarwal, S. Gupta, and T. Choudhury, "Continuous and Integrated Software Development using DevOps," *2018 Int. Conf. Adv. Comput. Commun. Eng.*, no. June, pp. 290–293, 2018.

[17] J. Kalra, Pardeep, and P. B. Nagpal, "A Review of Continuous Delivery Process," *Int. J. Res. Publ.*, 2018.

[18] V. Sharma, "Continuous Integration and Continuous Delivery (CI / CD): A Comprehensive Overview," vol. 8, no. 10, pp. 1835–1839, 2019.

[19] S. M. Mohammad, "Streamlining DevOps automation for Cloud applications," *Int. J. Creat. Res. Thoughts*, vol. 6, no. 4, pp. 2320–2882, 2018.

[20] M. Virmani, "Understanding DevOps & bridging the gap from continuous integration to continuous delivery," in *5th International Conference on Innovative Computing Technology, INTECH 2015*, 2015. doi: 10.1109/INTECH.2015.7173368.

[21] M. Soni, "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery," in *Proceedings - 2015 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2015*, 2016. doi: 10.1109/CCEM.2015.29.

[22] A. Steffens, H. Lichter, and J. S. Döring, "Designing a next-generation continuous software delivery system: Concepts and architecture," in *Proceedings - International Conference on Software Engineering*, 2018. doi: 10.1145/3194760.3194768.

[23] D. Stahl, K. Hallen, and J. Bosch, "Continuous Integration and Delivery Traceability in Industry: Needs and Practices," in *Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016*, 2016. doi: 10.1109/SEAA.2016.12.

[24] B. P. Eddy *et al.*, "A Pilot Study on Introducing Continuous Integration and Delivery into Undergraduate Software Engineering Courses," in *Proceedings - 30th IEEE Conference on Software Engineering Education and Training, CSEE and T 2017*, 2017. doi: 10.1109/CSEET.2017.18.