Understanding DevOps & bridging the gap from continuous integration to continuous delivery

Ravi Teja Yarlagadda. *Department of Information Technology, USA* yarlagaddaraviteja58@gmail.com

Abstract—*DevOps is considered an emerging concept that is seen by various experts as a strong solution in eliminating the traditional division and barriers that exist between the operations and developers in many organizations today. As the adoption of Agile transition has occurred in the last few years, IT companies have begun to incorporate continuous integration concepts in their software development lifecycle, and therefore improving the efficiency and reliability of the overall development process [1]. The biggest advantage of DevOps is its quick and regular product updates in continuous delivery of software. These features allow fast responses to customers' evolving needs. This is a huge benefit for businesses looking at gaining a competitive advantage over other market players. Continuous delivery is dependent on efficient teamwork and automation to respond to various customer needs faster. It is now known that the time spent on formalizing and integrating processes, alone, does not accelerate delivery speed or promote overall organizational performance. If the whole delivery chain functions smoothly, efficiency in an organization will be achieved [1]. This paper will focus on the concept of DevOps as well as how the gaps from continuous integration to continuous delivery can be bridged. The information that will be covered will involve how the aspects of DevOps apply to different SDLC phases in terms of customer needs, how to switch from continuous integration to continuous delivery, including the various benefits. It briefly discusses different factors one must think about before implementing DevOps, including the advantages that one might expect.*

Keywords: *DevOps, continuous integration, continuous delivery, Deployment Automation, Cloud Computing*

I. INTRODUCTION

DevOps is currently bridging the gap between software development and its deployment within major organizations. The primary focus of the DevOps initiative is to utilize frameworks like continuous integration, microservices, continuous delivery, as well as continuous deployment for an agile software development process [2]. Some of the developments in this field are whether the software is distributed via the internet through the servers (that is as a service) or distributed to the target user by a dedicated mobile interface or software distribution network. These new trends enable shorter distribution times in the rapidly changing Internet age. Formal work on DevOps has gotten a lot of coverage in the software development industry and within the practitioner literature.

As the software market becomes more competitive, companies devote more time and resources to developing and delivering high-quality software at a quicker pace. Continuous Integration (CI), and Continuous Delivery (CDE) are some of the two continuous practices designed to assist organizations in accelerating the growth and delivery of product features while maintaining efficiency [2]. While CI supports combining work-in-progress several times a day, CDE is concerned with the potential to release values to consumers quickly and efficiently by utilizing automation as much as possible. Although CI is the first step toward applying CDE practice, it is important to completely incorporate CDE practice to facilitate automated and consistent program rollout to production or customer settings (i.e., CD practice) [2]. No dedicated work has gone into systematically analyzing and rigorously synthesizing the research on continuous practices comprehensively. By the interconnected manner, it means jointly researching CI, CDE, and CD practices, resources, obstacles, and activities to discover and recognize the interaction between them and what steps can be taken to transition from one practice to another effectively and seamlessly. By undertaking a Systematic Literature Review of the practices, tools, problems, and procedures for embracing and applying continuous practices, this research aim to narrow the gaps [2].

Entrepreneurs are feeling a need to respond to the current trends because it is becoming demanding, and there is also more urgency to respond to consumer demands. They can no longer continue to allow customers to wait months or even a year for a software to be launched before inviting their input on how it performs. Consumers want an engaged relationship so that they can get ongoing input [3]. More companies need to be lean and flexible during the life cycle of product development to address the needs of the current problems. It has been common for companies to apply process transformations (for instance agile processes) to their application development for several years. Even so, throughout the whole software development process, tasks are usually abandoned. Development speed is higher than the pace at which teams will deploy new technologies. It is often said that the biggest weakness in a supply line is the one that controls the others. That is why they must address the weaknesses in the cycle at any point. The annual 'State of DevOps' studies indicates that the percentage of DevOps teams rose from 19% in 2015 to 22% in 2016 then to 27% in 2017 [4]. Thus, amid its rising success, there is a lack of analytical studies into DevOps' real-world implementation outside discussions in blogs and surveys. Few case studies exist that explore DevOps in pursuit of continuous software development. This paper explores DevOps and how the gaps that exist between continuous integration to continuous delivery can be bridged.

II. RESEARCH PROBLEM

The main research problem under study is to understand DevOps and identify the aspects that can bridge the gap from continuous integration to continuous delivery.

This research problem is particularly significant for the practices in the software development industry which allow organizations to introduce new products and features regularly and consistently. With a growing interest in and documentation on DevOps and continuous practices, it is necessary to consistently review and analyze the strategies, approaches, issues, and processes identified for the adoption and implementation of continuous practices. While the DevOps approach has been here for some while now, it is still the subject of intense debates. Organizations like it, but they're not confident how to approach it [5]. Even so, considering the growing success of recent studies, there seems to be a lack of systematic study on the practical practice of DevOps beyond the review of industrial surveys and blog posts. Apart from a few case studies, the latest literature does not offer any insight into the bridge of gaps from continuous integration to continuous delivery, application of DevOps, and their usefulness in facilitating continuous software development. In this analysis, these concerns are explored based on an in-depth exploratory literature review. At its core, DevOps seeks to bridge the difference between what the customer wanted and what the production team delivered.

III. LITERATURE REVIEW

A. *Understanding DevOps*

DevOps is a theory and methodology that emphasizes resilience, teamwork, and automation in IT and software production processes. Historically, software development was done in silos, with IT and development units and systems operating separately [6]. This division and conflicting beliefs provided an atmosphere fraught with miscommunication, weak alignment, and development setbacks (the operations department has also been dubbed the "War Room" by some). DevOps is a solution to the programming culture of "us vs. them." The aim is to increase coordination and teamwork between IT operations and production, build more seamless processes, and coordinate strategies and strategies for quicker and more effective delivery.

Multiple parties, agencies, associations, and suppliers are interested in the overall product development life cycle in the conventional approach to software development life cycle (SDLC). The phases of software life cycle management are common: market specifications are obtained by a business analyst, then developed by a production team (or outsourced), and assessed for functionality and technical performance by QA teams (also outsourced) [7]. Output and stress assessment is also administered in appropriate situations, using appropriate instruments, by appropriate classes. The organization's IT teams then handled the production implementation process, which involved a checklist and authorizations, as well as oversight and assistance from maintenance teams. Each phase of the maturity period, through functionality development to usability and maintenance, is handled by various staff, divisions, procedures, and resources. Techniques, systems, procedures, personnel, and resources also fragment this strategy, influencing the finished product in terms of functionality, cost, scheduling, consistency, efficiency, and other administrative overheads such as vendor interfacing and integration [7]. Also, repair, service costs, and ability specifications are often ignored in this process. Maintenance and maintenance operations, on the other hand, are crucial and must be evaluated, reviewed, and determined well in advance, both from an implementation development cycle and from a business perspective.

The DevOps model promotes collaboration and consistency by allowing implementation and support teams to exchange baselines and data. It specifies approval conditions for effective automated production movements, including creation, testing, and operations. From implementation and rollout to release and closure, DevOps guarantees that support plans are developed to ensure progress. It uses automation, software, and tried-and-true management approaches to speed and streamline progress while also ensuring that communication lines are still available, fostering trust and confidence between development and operations [8]. DevOps is in high demand, especially in today's cloud-based environment, where enterprise users can take control of their destiny. It necessitates good corporate leadership—those dedicated to dismantling the old siloed mindset and establishing a common mission and emphasis with consumers at the core of the model. DevOps facilitates a quick flow of planned tasks, including fast deployment speeds, while also enhancing the development environment's reliability, stability, resilience, and efficiency [9]. The solution provides a straightforward roadmap for more competitive IT operations by narrowing this gap.

B. *DevOps Principles*

DevOps is a practical approach, but it is still a mindset and culture change within an organization. This concept is supported by some of the fundamental principles:

a) Automation: To reduce redundancy and overwork, companies need to automate everything, including workflows, new code training, and how the infrastructure is configured.

b) Iteration: During a time-box sprint, it is important to write small chunks of code to facilitate releases and sub-releases, increasing the frequency and pace of deployments.

c) Continuous improvement: A continuous test often helps to learn from mistakes, and respond to feedback to improve efficiency, cost, and implementation time. This is where silos are broken down between development, IT operations, and quality assurance by bringing teams together, facilitating collaboration, and breaking down silos.

C. *DevOps Practices*

Organizations must be familiar with DevOps principles and practices to respond to what they need to do to boost the software development process and organizational agility. DevOps practices are a series of core concepts and guidelines that a company uses to simplify software development and technology management [10]. The following is a list of DevOps practices, along with a brief detailed breakdown of the Continuous Integration (CI) and Continuous Delivery (CDE) practices which are the focus of this research. These are continuous integration, continuous delivery, and microservices.

i. *Continuous integration*

Continuous integration is a software development methodology in which developers are in charge of incorporating code updates into a single, shared repository. Following the merging of such updates, automatic checks and builds are run. The main aim of this approach is to find and correct bugs early in the product development life cycle, reducing the time it takes to release software [11]. Following the check-in of code updates to a registry, each of these check-ins is tested using an automatic build tool,

followed by acceptable testing methodologies to identify any issues with the program (if any). Every day, developers should hopefully have several integrations, each of which is accompanied by an automatic build phase. Continuous integration means that developers still have access to the most up-to-date and tested code [12]. CI can avoid expensive production problems by encouraging many developers to confidently work on the same source code, rather than waiting for release day to merge different parts of code all at once. This protocol is an important part of the DevOps process flow, which strives to incorporate speed, agility, and protection. It is a well-known software development method in which members of a team combine and integrate development work (e.g., code) on a routine basis, for example, several times per day. Software businesses may use CI to have a faster and more regular delivery period, boost software consistency, and maximize the efficiency of their teams. This practice entails the development and testing of automated applications. Continuous integration assists in the identification and resolving of issues. It's worth mentioning that as the time interval between integrations gets longer, it becomes increasingly more difficult to find and repair bugs. Continuous incorporation supports developers and their businesses in many respects. It reduces the integration issues, detecting mistakes more quickly, and increasing visibility and coordination [13].

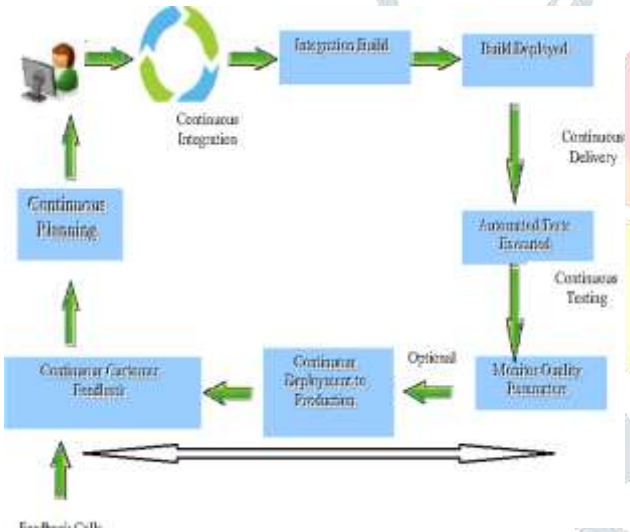

Fig i: DevOps life cycle

ii. Continuous Delivery

Continuous delivery is a software development practice in which code updates are incorporated in a shared repository, then developed and reviewed before being released in the production environment. Improvements to the code are designed, checked, and packaged automatically before being released into development. The aim is to provide users with alerts in a timely and reliable manner [13]. CDE accomplishes this by automating the update process (building on CI's automated testing) such that new projects can be released with a single click. In essence, this is a process that brings continuous integration to the next stage by introducing code updates to the production environment. In general, continuous delivery is a framework that helps developers to release software more easily and get feedback faster. It's a process that allows businesses to change applications to meet the demands of users and industry dynamics. Continuous integration and continuous delivery are inextricably linked; the latter is simply an extension of the former. CDE simply ensures that all updates are released on time. Faster feedback increases the consistency of the apps delivered; you get feedback sooner, allowing you to respond quickly to problems. Continuous Delivery (CDE) is a methodology for ensuring that an application is still ready for shipment after passing automatic reviews and quality checks [13].

CDE uses a series of practices, such as continuous delivery and implementation automation, to distribute applications to a production-like system automatically. According to, this approach has many advantages, including lower rollout risk, lower costs, and quicker user reviews. Getting continuous implementation experience necessitates continuous integration practice.

Luckily, there are methods out there that alter the entire test paradigm. Switching from manual testing to automated testing makes the whole process go Agile with Selenium. One such method I've come across in the past few years is Qualitia, which utilizes Selenium to quickly create automation workflows leveraging their script-less methodology. It also allows you to do interface validation, database validation, web servers, and all the way to file validation. Qualitia is capable of greatly reducing repair activities and saving rework efforts to tackle traceability and the smart warning system [1]5.

D. Closing the test gap

Why is it so difficult to automate the different test processes (performance, regression, tension, QA, GUI, interoperability, and security testing)? Five main elements need to be automated successfully:

1. Infrastructure— It begins with one very significant test-critical feature that is not automated – the capacity to model the infrastructure under which the program (or hardware) under test is supposed to work when it comes to production [15]. Getting the opportunity to assign each tester (or series of automated tests) its own personal copy of the target production infrastructure—a sandbox—is crucial to ensuring that the test program can be implemented automatically and constantly in production. Automating the tests is not quite as complicated as automating the underlying infrastructure under which the tests are carried out.
2. The workload of the manufacturing system – often ignored in automated testing, mimics the workloads that will be normal in production. This covers network traffic, additional program tasks, and network security profiles [15]. These tasks need to be automated inside the sandbox to build a more practical training environment.
3. Test Automation – In most courses, the evaluations themselves must also be standardized and there are some tools available to aid with this task. As coding grows more complex, automation tends to be an obstacle, with mobile testing substituting GUI testing as one of the most difficult forms of testing to automate [15].
4. Reporting and review of data – To ensure consistent automation from production through testing to execution, it is necessary to see and interpret test results automatically.
5. Tool integration – At the end of the day, automation into and out of testing has to be allowed. This ensures that sandboxes and test automation suites need to be API-driven so that a preceding framework in the DevOps toolchain can be initiated and a tool that matches testing in the toolchain can also be started [15].

DevOps projects begin in one portion of the spectrum from development to delivery. There should be nothing unusual with this solution. However, it is still important to

"Mind the Gap" and work constantly to extend automation from across the spectrum. One approach to ensure that automation can spread quickly is to implement sandbox technologies around the spectrum, starting with production and progressing through configuration and release management [16]. The sandbox maintains that the development setup and workload are repeated at each stage. DevOps relies on continuous automation from development, through testing and QA, to configuration management and production implementation. Sandboxes offer a generic automation framework that can be used to close the gap between production and execution at each stage of the DevOps cycle. Sandboxing technology has been used for production in the past, but is comparatively new to the entire DevOps process. Even so, as more organizations step towards continuous integration and continuous delivery, the need for sandboxes is increasing [16].

The following should be given for the sandbox:

- The opportunity to build a replica of the actual production environment within which the implementation is planned. This could include physical, virtual, and cloud networks, as well as software, network traffic, and loading [17].
- The ability to launch a sandbox using APIs such that the sandbox can be activated by DevOps software as an important part of the full end-to-end DevOps automation. • The ability to automate DevOps processes within the sandbox (e.g. automatic regression testing) and to automate sandbox extraction and enable the next stage in the DevOps cycle as these processes are completed.
- Market intelligence on anything that happens within sandboxes such that DevOps processes can concentrate on quality development.
- The opportunity of certain users to get their sandboxes working at the same time as solitude in shared experiments.

E. Benefits

Automated testing is one of the top priorities for the progress of DevOps. This demonstrates the recognition that DevOps cannot function if only one aspect of the DevOps spectrum is automated. Companies who have begun to rely on DevOps understand that they should be aware of the gaps from CI to CDE. The intended outcome of DevOps is speedy delivery of the same or better content, and there are several ways to do this. Of all, one of the key items is to operate together and automate [16]. The perfect combination of the two will make a great achievement. DevOps' philosophy means no more waiting for resources, failing quickly, failing sometimes, getting quick input, and offering quicker deliveries. This need for speed in DevOps requires continuous integration (CI) and continuous delivery (CD) channels. CI helps maintain continuous development and quick integration, and CD enhances the development and faster release. When these are paired together, it helps to reduce the average delivery time.

The more stable CI/CD pipelines, the higher the frequency of implementation, which is a typical mindset. It is a fact that there are gaps between the CI and the CD which should not be ignored. These gaps can be addressed by continuous testing (CT) to allow the DevOps process much smoother. When developers find problems early in the software development, it is easier to fix them without much impact and thus successful testing is necessary to produce quality products. Assessing the performance of DevOps is very critical and understand that strong code coverage and test coverage will guarantee a thinner defect density. Detecting bugs, or testing, might be a time-consuming task that involves a lot of precision [17]. For DevOps, once developers try to get quicker releases and rollouts of CI/CD pipelines, testing could become a bottleneck. Automation is the foundation of DevOps and will be important to use for testing. Automated tests offer continuous rapid input, which acts as a safety net that is needed to add new functionality and release more. Early testing and rapid testing face a significant problem when dealing with Selenium. Testers cannot proceed with automation until the test program is ready and open [18]. Developers would miss the deadline if they start the test automation when the development is completed. Likewise, they need to reduce the coverage of tests on a new build that might not be the reasonable thing to do.

IV. SIGNIFICANCE TO THE U.S

This research is significant to the U.S in closing the gaps that exist in the DevOps implementation especially from continuous integration to continuous delivery. The implementation of CI/CD methodologies allows service providers to promote a new approach to operations. An agile way of operating in close partnership between engineering and operations is required to allow rapid innovation and market value safely and securely. Without transformation, the strong gains of OPEX would not be understood – echoing for many of the first NFV eras. It is also significant in policymaking which is the leading collection of decisions that result in the transformation o the technological industry. The strategies implemented will help U.S companies to reshape and refocus their market goals, reassess the risk management, make critical decisions, and tangible business and organizational objectives for CI/CD into actionable directives. When it comes to bridging the gaps between continuous integration to continuous delivery service providers will determine the reach of the opportunity to implement CI/CD over time and not just the resources to enforce it [18]. It is also significant for many organizations to adopt SAFe (Scaled Agile) architecture to allow a continuous integration pipeline and process agility to handle multiple teams and multi-vendor deliveries. This requires the establishment of a capability growth plan that includes the evaluation of the skills shortage, the definition of the tasks required, their organizational planning, and the development of a formal training plan on how to bridge the gaps between CI and CDE and agile operating methods.

V. CONCLUSION

There are lots of hurdles to overcome in bridging the gaps between CI and CDE, but It's worth taking a systematic approach when adopting DevOps in any business. Embracing change is preferable to slipping behind the market. Despite the difficulties, adopting a DevOps philosophy is not just doom and gloom. This paper provided a summary of DevOps, including its advantages and drawbacks, as well as strategies for bridging the gaps from CI to CDE. The paper has demonstrated that on the benefits of bridging the gaps from CI to CDE. Continuous practices have many advantages, including receiving more and faster input from the app development process and clients; providing regular and timely updates, which leads to increased consumer satisfaction and quality standards; and strengthening the connections between development and operations teams and eliminating manual tasks via CD. Continuous practices are gaining traction in software development industrial practices through diverse sectors and dimensions of companies, according to an increasing number of industrial cases. Implementing continuous practices is not a simple task because organizational structures, practices, and tools may not be prepared to support the increasingly nuanced and demanding existence

of these practices. Since continuous practices are becoming more relevant, an increasing number of literature detailing methods, resources, practices, and problems has been published. These practices are heavily correlated and interconnected, making it difficult to differentiate them, and their interpretations are highly dependent on how a particular entity interprets and utilizes them.

References

[1] M. Virmani, "Understanding DevOps & bridging the gap from continuous integration to continuous delivery," *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015.

[2] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industry software development", *Journal of Systems and Software*, vol. 87, pp. 48-59, 2014. Available: 10.1016/j.jss.2013.08.032.

[3] D. Farley and J. Humble, *Continuous delivery*. [Erscheinungsort nicht ermittelbar]: Addison-Wesley Professional, 2010.

[4] D. Lee, T. Lim and D. Arditi, "Automated stochastic quality function deployment system for measuring the quality performance of design/build contractors", *Automation in Construction*, vol. 18, no. 3, pp. 348-356, 2009. Available: 10.1016/j.autcon.2008.10.002.

[5] J. Kanjilal, "DevOps - Bridging the Gap between Dev and Ops - InsightsSuccess", *InsightsSuccess*, 2017. [Online]. Available: https://www.insightssuccess.com/devops-bridging-the-gap-between-dev-and-ops/.

[6] P. Ajibade, E. M. Ondari-Okemwa, and M. M. Matlhako, "Information technology integration for accelerated knowledge sharing practices: challenges and prospects for small and medium enterprises", *Problems and Perspectives in Management*, vol. 17, no. 4, pp. 131-140, 2019. Available: 10.21511/ppm.17(4).2019.11.

[7] J. Wettinger, U. Breitenbücher, M. Falkenthal and F. Leymann, "Collaborative gathering and continuous delivery of DevOps solutions through repositories", *Computer Science - Research and Development*, vol. 32, no. 3-4, pp. 281-290, 2016. Available: 10.1007/s00450-016-0338-z.

[8] S. Asmus, A. Fattah and C. Pavlovski, "Enterprise Cloud Deployment: Integration Patterns and Assessment Model", *IEEE Cloud Computing*, vol. 3, no. 1, pp. 32-41, 2016. Available: 10.1109/MCC.2016.11.

[9] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too", *IEEE Software*, vol. 32, no. 2, pp. 50-54, 2015. Available: 10.1109/ms.2015.27.

[10] M. Meyer, "Continuous Integration and Its Tools", *IEEE Software*, vol. 31, no. 3, pp. 14-16, 2014. Available: 10.1109/ms.2014.58.

[11] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges, and Practices", *IEEE Access*, vol. 5, pp. 3909-3943, 2017. Available: 10.1109/access.2017.2685629.

[12] R. Vaasanthi, S. Philip and V. Prasanna, "Comparative Study of DevOps Build Automation Tools", *International Journal of Computer Applications*, vol. 170, no. 7, pp. 5-8, 2017. Available: 10.5120/ijca2017914908.

[13] M. Ilyas, "Software Integration Challenges for GSD Vendors: An Exploratory Study Using a Systematic Literature Review", *Journal of Computers*, pp. 416-422, 2017. Available: 10.17706/jcp.12.5.416-422.

[14] G. Adzic, Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing. London: Neuri, 2009.

[15] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review", *Software Testing, Verification and Reliability*, vol. 27, no. 8, p. e1639, 2017. Available: 10.1002/stvr.1639.

[16] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage, and validation of the Eiffel framework", *Empirical Software Engineering*, vol. 22, no. 3, pp. 967-995, 2016. Available: 10.1007/s10664-016-9457-1.

[17] M. Paul, "Fill the Gap Between CI and CD Pipelines With Continuous Testing - DZone DevOps", *dzone.com*, 2017. [Online]. Available:https://dzone.com/articles/fill-the-gap-between-ci-and-cd-pipelines-with-cont.

[18] T. Binz, C. Fehling, F. Leymann, A. Nowak and D. Schumm, "Formalizing the Cloud through Enterprise Topology Graphs", *2012 IEEE Fifth International Conference on Cloud Computing*, 2012. Available: 10.1109/cloud.2012.143 [Accessed 15 March 2021].