IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 11, Issue 1, November 2021
Impact Factor: 5.731

A Review on CI/CD Pipeline with Technologies, Tools and Challenges

Dharmendra Ahuja
DevOps Engineer, IBM
dharmendradevops11@gmail.com

Abstract: *Software development approaches that priorities continuous integration, delivery, and deployment enable companies to consistently provide new features and products. CI/CD has become one of the most important DevOps methods that automates the software development, testing, and delivery processes. This paper provides a detailed description of the CI/CD architecture, pipeline design, tools, optimization techniques, and adoption considerations. It describes the evolution of Continuous Integration into Continuous Delivery and Continuous Deployment, and how these methods are applied to improve code quality, accelerate delivery, and foster collaboration between development teams. This review provides a high-level overview of the CI/CD pipeline and the steps involved, including code commit, build, test, deploy, and monitor, along with popular platforms such as Jenkins, GitLab CI, Azure DevOps, CircleCI, Docker, and AWS services. Besides, the key optimization techniques, parallel execution, dependency caching and incremental builds, are described to enhance pipeline performance. The paper concludes that the barriers to CI/CD adoption can be technical, organisational, and cultural, especially in legacy and regulated industries, and it defines how to achieve effective and sustainable automation in software engineering today.*

Keywords: Continuous Integration (CI) and Delivery (CD), Continuous Deployment, CI/CD Pipeline, Test Automation, Monitoring and Logging, DevSecOps

I. INTRODUCTION

Modern software development requires Continuous Integration and Continuous Delivery (CI/CD), therefore delivery speed and flexibility are crucial. CI/CD is a more rigorous approach to software development, testing and implementation based on the emphasis on automation, rapid feedback and continuous improvement [1]. CI/CD helps development teams create high-quality software that can be delivered faster through automated integration, testing, and delivery pipelines, meeting rising user demand and market requirements [2].

The CI/CD basically streamlines the software development lifecycle. The idea behind Continuous Integration (CI) is to integrate the working copies of all developers into a single mainline multiple times a day to ensure the codebase is in a deployable state and to identify integration issues as soon as possible [3]. Continuous distribution (CD), on the other hand, automates the distribution of code to production environments, expanding on the idea of Continuous Integration (CI), allowing teams to deploy bug fixes, upgrades, and new features. The role of CI/CD in contemporary software development cannot be overestimated. It solves some of the greatest problems that development teams are grappling with today, including the requirement to have faster release cycles, maintain code quality, and ensure that the software is always in a releasable state.

By using CI/CD, businesses may shorten the time between code commit and production, enabling faster response to changes in the market, customer needs, and competitive pressures. The DevOps idea was created to address the disconnect between software development and product deployment in large software companies [4]. The main purpose of DevOps is to enable a rapid software development cycle through continuous software cycles, including microservices, continuous delivery, and continuous deployment. Other trends include the increasing provision of software over the Internet, either on a server-based or direct-to-consumer basis, and the rising ubiquity of mobile

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

platforms and technologies on which this software is exploited [5]. Continuous Delivery (CD) is a fundamental practice in DevOps, though achieving scalability and consistency is difficult when legacy systems and limited automation hinder implementation. While most research emphasizes build, test, and deployment automation, the architectural and organizational aspects of CD adoption remain underexplored.

The challenges of interoperability between tools, scalability of pipelines, integration of security, limitations of the legacy system, and resistance within the organization are still seen as barriers to successful adoption [6][7]. In addition, although the available studies are mainly focused on the automation features of the tools, including build and testing, the systematic inspection of CI/CD tools, deployment methods, and the real challenges in the actual environment is underrepresented.

A. Structure of the Paper

This paper is structured as follows: Section II presents the fundamental concepts of Continuous Deployment and Continuous Process (CI/CD). Section III discusses CI/CD tools and platforms, including popular automation, testing, and monitoring tools. In Section IV, the author is interested in integrating CI/CD methods and optimization approaches. The literature related to the study is reviewed in Section V. Lastly, Section VI outlines future research directions, with particular emphasis on applications in pressure piping systems, including cloud-native CI/CD pipelines, DevSecOps, and AI-driven pipeline optimization.

II. CI/CD ARCHITECTURE AND PIPELINE DESIGN

CI/CD is an automation technique that speeds up development by establishing routines for the build process, application testing, and deployment [8]. A development process called continuous integration incorporates code into central repositories using automated procedures to ensure that code modifications don't cause functional issues. Continuous Deployment goes one step farther by putting verified changes into practice to operational environments, allowing businesses to provide enhanced capabilities to their end customers more quickly. Benefits of CI/CD:

• Implementing CI/CD pipelines has several advantages, including:

• Organizations may respond to client requests more rapidly by bringing their goods to market thanks to object automation, which makes software upgrades quicker.

• In the validation of every code change prior to the deployment of new software into the production environment, continuous testing enhances the quality of code. Defects are therefore kept separate from production.

• Teams may collaborate more effectively through CI/CD when they have a common testing platform, since it fosters teamwork.

A. Continuous Delivery (CI)

In continuous integration (CI), software development teams routinely integrate their work and automate the build, test, and validation procedures. It streamlines the process of finding and fixing issues, which in turn improves software quality [9]. In a study using an agile development model, students found that CI helped detect and correct faulty changes 65% more quickly and increased code quality by almost 50% . Approximately 70% of students reported that CI improved their overall development workflow [10]. The relationship between CI, deployment, and delivery is shown in Figure 1 below:

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 11, Issue 1, November 2021
Impact Factor: 5.731

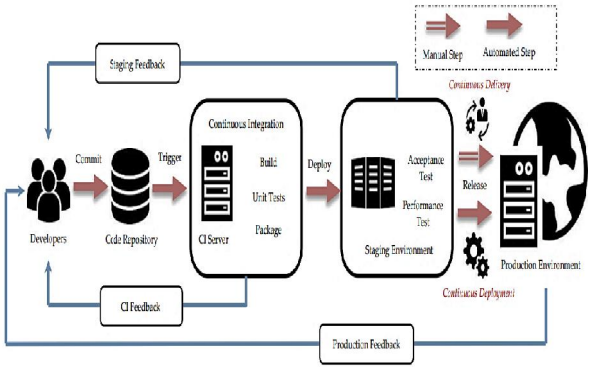

Fig. 1. The Relationship Between Continuous Integration, Delivery And Deployment

The key elements of CI are the version control system, source repository, and CI server. Amazon Web Services (AWS) cites the following challenges when attempting to use continuous integration (CI) methods: automating builds, automating testing, keeping a single source repository, and making changes to a common codebase more often [11]. The advantages of implementing a continuous integration (CI) approach include increased productivity and code quality, build automation, analytics, CD enablement, code stability, quicker releases, and cost savings.

B. Continuous Delivery and Deployment

The ability to quickly, safely, and sustainably introduce updates of all kinds—continuous delivery—such as new features, configuration adjustments, bug patches, and experiments—into production or into the hands of users. Krusche has assessed the use, advantages, and experience of CD in courses including multi-customer projects. According to Chen, there is a rapid trend in CD investment because of its advantages, which include quicker time to market, consistent releases, improved customer satisfaction, enhanced productivity and efficiency, and producing the appropriate product.

According to AWS, Continuous Delivery (CD) ensures that every committed change is production-ready, whereas Continuous Deployment automatically implements these changes in production. Currently, several studies use continuous deployment automation to improve efficiency [12]. Continuous deployment accelerates processes in agile methodologies, including Facebook, GitHub, Netflix, and Rally Software, exemplifying organizations that effectively implement continuous deployment in their production environments.

Table I illustrates the progressive evolution from code integration to fully automated production deployment as below:

TABLE I. COMPARATIVE OVERVIEW OF DEVOPS PIPELINE PRACTICES

Aspect	Continuous Integration (CI)	Continuous Delivery (CD)	Continuous Deployment (CDE)
Key Practices	Automated building and testing; frequent code merges.	Extends CI with automated release pipelines up to production readiness.	Fully automates deployment to production without manual approval.
Level of Automation	Automates build and test phases.	Automates build, test, and staging deployment phases.	Automates build, test, staging, and production deployment phases.
Main Benefit	Identifies conflicts and bugs early, thereby improving software quality and team productivity.	Enables fast, reliable releases; reduces human errors; and provides quick responses to feedback.	Delivers changes to users immediately; maximizes speed and agility.
Deployment Frequency	Multiple integrations occur daily, with releases dependent on manual steps.	Ready for release anytime; production deployment may still be manual.	Production deployments occur automatically and frequently (even multiple times a day).

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

Dependency	Foundation for CD and CDE.	Builds on CI; required for CDE.	Builds on CI and CD; full automation.
Typical Use Case	Any modern software development team.	Teams aiming for frequent, predictable releases.	Organizations need continuous customer-facing updates.
Key Difference	Focus on integrating and testing code.	Adds automation up to staging and release readiness.	Pushes automation through to production deployment.

C. CI/CD Pipeline Architecture

A CI/CD pipeline is used to automate the processes of code integration, testing, deployment, and monitoring. It enables software to be delivered more quickly, reliably, and of high quality, with continuous automation and feedback. As illustrated by Figure 2, the movement between code changes is depicted by the CI/CD pipeline architecture, automated testing, staging and production environments, continuous integration (CI), continuous delivery/deployment (CD), and artifact control.

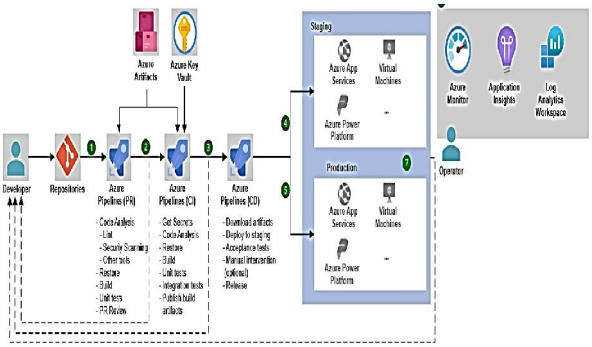

Fig. 2. The Architecture of the CI/CD Pipeline

1) Code Commit

Developers begin the CI/CD process by committing their code to a VCS, such Git. The process is as follows:

• Local Development: Developers work on code and run tests on their local PCs in local development.

• Commit: Once satisfied, developers finalize their modifications and commit them to the shared repository. This step activates the CI/CD pipeline.

• Pull Requests: Pull requests (PRs) are common among developers to make their improvements in a collaborative environment [13][14]. PRs are evaluated by peers to determine adherence to coding standards and high quality.

2) Build

The build process is initiated by the CI server (such as Jenkins or Travis CI) once the code has been committed. Retrieving Source Codes:

• CI: The CI server fetches the latest code from the repository.

• Dependency Installation: Project configuration files (such as package. Son and pom.xml) specify the libraries and dependencies that must be installed.

• Compilation: The source code is transformed into binary code or executable code.

• Artifact Creation: The build procedure creates artefacts that are distributed to various environments, such as JAR files and Docker images.

3) Test

The functionality and quality of the code are determined using automated testing. Multiple stages of testing are present in this step:

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

• Unit Tests: Small tests, discrete tests that verify certain application features or components. Examples of frequently used tools are Mocha (JavaScript), pytest (Python) and JUnit (Java).
• Integration Tests: These are tests that can confirm the interplay of various modules or services. These tests are used to ensure that the integrated components have performed as expected.
• End-to-End Tests: Wide-ranging testing which is simulative of real-life situations of the users. To ensure the entire application works as intended, browser-based testing is performed using programs such as Selenium and Cypress.
• Static Analysis: Code-quality tools, like SonarQube, examine the code for errors, security vulnerabilities and code smells.
• Performance Tests: Performance of the application when it is under load is measured with the help of such tools as JMeter and Gatling.

4) Deploy

The code may be deployed to any environment when all tests are completed:
• Staging Environment: The code is initially tested in a staging area that is intended to resemble the production setup. This context allows for the last round of testing and validation before deployment to production.
• Canary Deployment: The updated version of the code is deployed to a limited number of customers to allow any issues to be noticed before a large-scale deployment.
• Production Environment: The code gets transferred to the production environment when all functionality is running well in the staging environment. Tools like Kubernetes, Octopus, and AWS CodeDeploy handle the deployment process.

5) Monitor

Constant monitoring ensures that the application deployed is working as expected:
• Performance Monitoring: The performance of the application is monitored with the help of such programs as Grafana and Prometheus, the parameters to be measured are the error rate, throughput, and response time.
• Log Management: The ELK Stack (Elasticsearch, Logstash, Kibana) can be used to gather and analyze log data so as to detect issues and patterns.
• Alerting: To enable prompt response and problem-solving, monitoring systems send out warnings when the performance threshold is crossed or even when mistakes are made.
• Staging Deployment: The code is initially released into an imitation environment that replicates the production environment. Prior to deployment to production, this is the context where all necessary testing and validation can be carried out.

III. CI/CD TOOLS AND PLATFORMS

Continuous Integration (CI) is often first, followed by Continuous Delivery (CD) and Continuous Deployment (CD) in CI/CD pipelines, which are typically executed slowly and methodically. Initially, organizations implement CI practices to speed up code integration and testing, which, in turn, provides a foundation for further automation. A significant decrease in the amount of manual development work is required when the pipeline creates a completely automated software delivery process by progressing from continuous integration (CI) to continuous deployment, testing, and release (CD) [15]. Continuous Delivery includes the automation of all steps of the pipeline that consist of code integration, build, testing, and staging, and it is only the deployment to a production environment that still needs someone's consent.

A. CI/CD Tools

The many types of CICD tools include building tools, tools for automation, test automation, version control and repository management, and monitoring.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

1) Repository and Version Control Tools

The open-source build tools Gradle, Ant, and Maven are three of the most popular versions. Some examples of continuous integration servers are TeamCity, Sysphus, Bamboo, Hudson, Cruise Control, and Jenkins. These servers offer a centralised solution for managing the build tools. Their primary continuous integration server is Jenkins because of its cross-platform compatibility, scalability, flexibility, and security.

2) Automation Tools

The primary objective of automation is to ensure consistency in infrastructure, is known as configuration management (CM). It offers advantages like lower costs, simpler release processes, appropriate host management, and simple configuration, efficiency, and dependability as well as optimized resource use. In 1993, the CF Engine was released as the first modern open-source CM tool [16]. Google's cloud platform offers a solution for "Managed Compute Engine" that makes use of the CM tools Chef, Ansible, Puppet, and Salt. According to AWS, development teams should be proficient with Puppet, Chef, Salt, and Ansible as CM technologies. Ansible is chosen over Chef and Puppet because it is the most straightforward and quick option for configuration management.

3) Test Automation

In the agile process, Mike Cohn created the idea of the "test automation pyramid." The three tiers of this pyramid are Unit, Service, and UI. On top of the service test layer, AWS has added a layer known as Performance/Compliance. Unit, functional, sniff, and performance tests are the steps of the deployment pipeline that test automation orchestration explains. Stable code bases, quicker reaction times, and simple decision-making are advantages of continuous testing.

4) Monitoring and Logging Tools

The aforementioned dangers may be avoided by using efficient tools and implementing functional management controls in all areas of continuous integration and deployment, including development, code check-in, build, test, deployment, and operations. Despite having similar features, continuous integration and continuous deployment systems continue to be distinct entities carrying out distinct activities [17]. These technologies replace the work that was previously done by developers or operations staff by facilitating continuous integration and deployment. A plethora of potential difficulties in continuous integration and deployment systems lure careless developers and operations staff, who mistakenly think their work is over after application deployment after making the mental leap to post-deployment concerns. The difficulty of locating the point of failure when sporadic issues arise increases with The magnitude of the infrastructure for continuous deployment and integration.

B. Automation Tools Analyzed

A number of well-known automation tools were assessed based on their capabilities in order to determine how they affected CI/CD performance. Change-related procedures that implement best practices by using new management concepts and technological tools comprise the DevOps culture. The tools that were analyzed include the following:

1) Jenkins

Jenkins is an open-source, free automation server that can oversee CI/CD procedures. It can be extended through its plugins to offer an automation of build, testing, and deployment of many languages and technologies. This makes it possible to generate code changes automatically to start builds and testing, and deployments to enhance efficiency and reliability. An open-source community supports Jenkins, one of the CI/CD automation systems.

2) CircleCI

A cloud-based CI/CD platform, CircleCI provides a number of automation options. It provides effective caching, robust dependency management, and parallel task execution. CircleCI is a desirable alternative for businesses looking to improve their CI/CD procedures because of its intuitive UI and extensive configurable possibilities.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

3) Container and Docker

A program plus all of its dependencies, libraries, and other binaries needed to run it make up a container's whole runtime environment. All of this is included in a single package [18][19]. Application and infrastructure dependencies are eliminated by containerization. Docker is a prominent software provider that allows us to develop and bundle programs for deployment. Linux containers may be created using Docker. Docker is a virtualization solution that makes use of a Docker engine rather than containers. Virtual machines employ hypervisors, and Docker makes it easier to execute several programs on a single computer. Each of which operates in a container, which is a separate environment.

4) GitLab CI

As a component of the GitLab platform, GitLab CI is a powerful automation tool. It offers an extensive array of options for deployment and continuous integration, including rollbacks, dependency caching, and support for parallel testing. GitLab CI is an easy-to-use tool with seamless integration with version control; this is why it is a favorite among teams that need to simplify their development processes.

5) Azure DevOps

The platform of development tools and services known as Azure DevOps includes the CI/CD functionality. It enables sophisticated deployment strategies, including dependency management, parallelization, and blue-green and canary deployments. Azure DevOps represents a unified space where the entire development cycle can be examined, which is why it can be considered an optimal solution when the team members are interested in including CI/CD to their processes [20]. The evaluation of these automation tools points at their features of improving the CI/CD performance. Knowing how these tools can be used, organizations are able to make the wise decision regarding their CI/CD practice, and eventually able to achieve better efficiency, quick deployment cycle, and overall high reliability.

6) AWS

AWS Code Pipeline is a service which is managed fully, a service which automates build process, test and deployment process within AWS ecosystem. It is also closely connected with third-party tools and services including Code Commit, Code Build, and Code Deploy [21]. Code pipeline. AWS codepipeline is completely integrated to enable infrastructure automation and is scalable and event-driven, security, and compliance upon cloud deployment.

IV. INTEGRATION OF CI/CD WITH TECHNIQUES AND CHALLENGES

The principal CI/CD optimization techniques, which make the pipeline efficient such as parallel-running, dependency caching, incremental builds etc. and which also describes the technical, organizational and cultural obstacles that render a successful CI/CD deployment difficult. It concentrates on the combination of automation to have performance improvements and the potential impediments that can go along with infrastructure, legacy systems and collaboration.

A. CI/CD Techniques

Current CI/CD Optimization Techniques the optimization techniques have been advanced in many studies to improve the performance of CI/CD. It provides several methods for businesses to improve the CI/CD pipelines' efficiency. These methods include:

1) Parallelizing Build and Test Processes

Organizations can greatly decrease the time taken in such activities by carrying out various builds and tests concurrently. Parallelization of build and test processes is one of the basic optimization strategies whereby several tests and build processes are carried out simultaneously as opposed to carrying them out in a sequence [22]. This approach significantly cuts down on the entire execution duration of the CI/CD pipeline, enabling faster feedback loops and deployment cycles. For instance, parallel execution enables businesses to simultaneously perform acceptance,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

integration, and unit tests. Parallelization can be supported using different tools and frameworks that allow teams to optimize their pipelines to utilize the maximum of resources and spend minimum idle time.

2) Dependency Caching

The use of dependency caching mechanisms can reduce the build time tremendously. Using the same dependencies downloaded before each compilation, rather than downloading it again, allows the teams to make the most of the resource usage and make the CI/CD pipeline faster. Dependency caching guarantees the storage and reuse of popular libraries and resources in more than one build. The practice may result in significant time savings on build times, especially in large applications which depend on large third-party libraries. Organizations can prevent the recurring process of downloading and installing dependencies with each build with dependency caching, as developers can spend more time writing code and not ask the build to complete.

3) Incremental Builds

Instead of recreating the whole codebase each time the change is introduced, incremental builds only recompile and retest the changed parts. The method saves resources and time and allows teams to concentrate on the most topical points of their code. An incremental build is one of the efficient optimization methods that is the opposite of the old normal full builds. With incremental builds, only the modified components are recompiled after each code change, rather than the complete program. Build times are significantly shortened with this method, particularly for large applications with many files and modules.

B. Challenges in CI/CD Adoption

The CI/CD methodology has demonstrated efficacy in improving software quality and identifying flaws, it has not yet found acceptance, especially among groups migrating out of the traditional methods of development. Resistance to change, adherence to current practices, and the need for a change in the developer's mindset play a significant role in successful implementation. Some of the infrastructural challenges that many organizations have included in the list of why they are not fully or not using CI at all are a lack of server procurement, difficulty in cloud set-ups, and additional overhead to operations [23]. Adoption is further impeded by cultural and organizational barriers (e.g. lack of awareness, management support, and skills) and is particularly slower in DevOps-oriented environments. Technical issues are encountered when applying the use of CI/CD into legacy systems and monolithic architecture, and processes that are not automated in testing. Heterogeneity, inconsistency, missing information, traceability, and visualization are also data-related problems that complicate the process of pipeline automations. Also, some disagreements between development and operation teams during deployment and rigorous regulatory requirements, such as the isolation of artefact storage, the necessity of approvals, restricted code visibility, and a lack of collaboration, can become strong impediments to a successful CI/CD adoption.

V. LITERATURE REVIEW

The study progression from basic pipeline automation using tools like Jenkins and Ansible has led to increasingly complex, specialized, and even codified CI/CD systems. According to the literature on Continuous Integration and Continuous Deployment (CI/CD), modern systems have evolved significantly from their simpler origins. The main contributions, techniques, and findings of the recent research in this field are presented in Table II.

Zhao et al. (2020) introduced a Cloud DevOps infrastructure to the software engineering community and showed how heterogeneous agents could use it to recreate experiments in areas of computer science. A DevOps deployment using self-hosted compute engines for large-scale computation and publicly available cloud computing resources for medium-scale research can more reliably share the findings of their experiments with others [24].

Rangnau et al. (2020) proposed integrating three automated dynamic testing techniques into a continuous-delivery and continuous-integration pipeline, and tested the overhead of this strategy empirically. The paper offers early solutions to the identified problems by highlighting several scientific and technological barriers that the DevSecOps community

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

may face. The findings allow making informed choices in application engineering and agile enterprise security to adopt DevSecOps strategies [25].

Mysari and Bejgam (2020) provide a fundamental explanation of the pipeline idea, with an emphasis on its development and using Jenkins for integration and Ansible for deployment. Integration is made easy by Jenkins's open source design and wide selection of plug-ins. An open-source configuration management tool called Ansible makes use of a simple YAML configuration syntax. Saving time is a significant challenge for developing enterprises, thus Jenkins Ansible's automation of integration and deployment saves a lot of time. Additionally, it makes it simple to update the project if changes need to be made on a regular basis. Shh access with Ansible is easy, and it can operate on a Jenkins node [26].

Zdun et al. (2019) conducted a thorough, qualitative analysis of 25 practitioner-provided deployment practice descriptions that included unofficial deployment pipeline models. also produced a model of deployment pipeline topologies that was carefully defined. Additionally, it has been demonstrated that the formal model significantly improves modelling precision in comparison to the original sources' loosely modelled pipelines [27].

Singh et al. (2019) They evaluate several continuous integration and delivery systems, taking into account factors like server monitoring, cloud compatibility, pipeline integration, and post-deployment performance monitoring. A microservice architecture is a design paradigm for enhancing cloud services that helps large companies expand their applications in response to increasing demand. Containers, virtual machines, or serverless capabilities like AWS Lambda can all be used to deliver these microservices. However, it is harder to manage and implement microservices as their number rises [28].

Arachchi and Perera (2018) This method has expanded the three automated stages of the CICD process are load testing, scaling, and benchmarking. When load testing with production traffic and benchmarking using the test bench approach, system disturbances are reduced and more accurate findings are obtained. After the system has completed its load tests and benchmarks, it can be scaled. The Ansible automation server, Jenkins, the Git repository, and Nexus were first used to develop the pipeline. Then, use Go Replay to replicate the production environment's traffic in the test bench. Although scaling does not dramatically improve the program's reaction time, it is able to manage the same load while modifying the application software. This three-step process is an improved aspect of CICD automation that lowers the risk of application deployment. Each phase's system activity is examined using Nagios monitoring. Thus, the study provides a productive approach to Agile-based CICD project management [29].

TABLE II. CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT

Authors	Focus Area	Objectives	Approach	Key Findings	Future Work
Zhao et al. (2020)	Cloud DevOps Infrastructure for Reproducible Research	To enable reproducible experiments for heterogeneous agents in computer science using DevOps principles	Introduced Cloud DevOps infrastructure leveraging free cloud resources for medium-scale experiments and Self-hosted large-scale computing engines	Demonstrated that Cloud DevOps improves experiment reproducibility, reliability, and sharing of results among researchers	Extension to larger-scale, cross-domain experiments and tighter integration with research data/versioning standards
Rangnau et al. (2020)	DevSecOps in CI/CD Pipelines	To evaluate costs and incorporate dynamic testing into pipelines for continuous integration and delivery using automated	Integrated three automated dynamic testing techniques and conducted empirical overhead analysis	Identified performance overheads, research gaps, and technology challenges in DevSecOps adoption; provided	Optimization of testing overhead, improved automation maturity, and broader empirical validation in real-world enterprise

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

		methods		preliminary mitigation strategies	systems
Mysari & Bejgam (2020)	CI/CD Automation using Jenkins and Ansible	To design a basic CI/CD pipeline skeleton emphasizing automation and resource efficiency	Implemented CI using Jenkins pipelines and deployment/configuration management using Ansible with YAML-based scripts	Showed that Jenkins–Ansible integration reduces time, simplifies SSH access, improves update flexibility, and saves organizational resources	Enhancing scalability, security automation, and integration with containerization and cloud-native CI/CD tools
Zdun et al. (2019)	Deployment Pipeline Architecture Modeling	To formalize deployment pipeline designs based on descriptions provided by practitioners	Conducted a qualitative study of 25 practitioner deployment pipelines and developed a precise formal model	Formal models significantly improved precision and clarity compared to informal pipeline descriptions	Application of formal pipeline models to automated validation, optimization, and tool-supported pipeline generation
Singh et al. (2019)	Comparison of Microservices CI/CD Tools	In order to assess CI/CD solutions, it look at their cloud compatibility, monitoring capabilities, and pipeline integration	Comparison of CI/CD solutions in cloud environments with microservices	Highlighted deployment and management complexity as microservices scale; emphasized need for robust CI/CD tooling	Development of intelligent orchestration, improved monitoring automation, and support for large-scale microservice ecosystems
Arachchi & Perera (2018)	Extended CI/CD Pipeline with Benchmarking and Scaling	To minimize deployment risk by extending CI/CD pipelines with benchmarking, load testing, and scaling phases	Implemented an extended CI/CD pipeline using Jenkins, Git, Nexus, Ansible, GoReplay, and Nagios	Reduced deployment risk and system interruption; validated system scalability, though response time optimization was limited	Enhancing response-time optimization, intelligent auto-scaling strategies, and advanced performance analytics

VI. CONCLUSION AND FUTURE WORK

Organisations have regularly delivered software with increased reliability, quality, and speed using CI/CD, which are now acknowledged as fundamental components of modern DevOps operations. CI/CD pipelines minimize the impact of human errors because they automate code integration, testing, deployment, and monitoring to provide better collaboration and early defect detection. This paper has described CI/CD architecture, pipeline stages, supporting tools and optimization methods including parallel execution, dependency caching and incremental builds. It also pointed out

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

the high advantages of CI/CD implementation such as a better level of productivity, a shortened time-to-market, and a regular release. Nonetheless, the implementation process is not always smooth, and organizations usually encounter limitations with technologies, limitations of the legacy systems, cultural resistance, and skills gap. To solve these problems, it is necessary not only to have superior equipment but also to establish an organizational preparedness and a DevOps-oriented mental state. On the whole, CI/CD could be considered a vital facilitator of agile, scaled, and sustainable software development.

Future studies can be aimed at AI-powered CI/CD pipelines of intelligent testing, intelligent detection of imminent failures, and intelligent optimization. By investigating the incorporation of CI/CD into new technologies, large software systems may be able to be scaled, automated, and more reliable in the future.

REFERENCES

[1] S. Pawar and Y. Ghodke, "Macro Software Integration In Operating System For Office Automation," IJARIIE, vol. 4, no. 4, 2018, [Online]. Available: https://ijariie.com/AdminUploadPdf/MACRO_SOFTWARE_INTEGRATION_IN_OPERATING_SYSTEM_FOR_OFFICE_AUTOMATION_ijariie8875.pdf?srsltid=AfmBOopnbZaZgZbTAI_Y_yuWPx8lWNo9gkfy95GWJ49PRPKEBKsECGBD

[2] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Aug. 2017, pp. 197–207. doi: 10.1145/3106237.3106270.

[3] L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps," 2018. doi: 10.1109/ICSA.2018.00013.

[4] 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.

[5] S. Garg, "Predictive Analytics and Auto Remediation using Artificial Inteligence and Machine learning in Cloud Computing Operations," Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci., vol. 7, no. 2, pp. 1–5, 2019, doi: 10.5281/zenodo.15362327.

[6] F. Erich, C. Amrit, and M. Daneva, "A Qualitative Study of DevOps Usage in Practice," J. Softw. Evol. Process, vol. 00, 2017, doi: 10.1002/smr.1885.

[7] G. Abbas and H. Nicola, "Optimizing Enterprise Architecture with Cloud-Native AI Solutions: A DevOps and DataOps Perspective," 2018.

[8] R. C. Thota, "CI/CD Pipeline Optimization: Enhancing Deployment Speed and Reliability with AI and Github Actions," Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci., vol. 8, no. 2, pp. 1–11, 2020, doi: 10.37082/ijirmps.v8.i2.232185.

[9] R. S. S. A. Pushkala and R. Carvalho, "Systems and methods for rapid processing of file data," US9594817B2, 2017

[10] Y. Ska, "A Study and Analysis of Continuous Delivery, Continuous Integration in Software Development Environment," Researchgate, vol. 6, no. September, pp. 96–107, 2019.

[11] M. Shahin, M. Zahedi, M. A. Babar, and L. Zhu, "An empirical study of architecting for continuous delivery and deployment," Empir. Softw. Eng., vol. 24, no. 3, pp. 1061–1108, Jun. 2019, doi: 10.1007/s10664-018-9651-4.

[12] Z. Sampedro, A. Holt, and T. Hauser, "Continuous Integration and Delivery for HPC: Using Singularity and Jenkins," in Proceedings of the Practice and Experience on Advanced Research Computing, ACM, Jul. 2018, pp. 1–6. doi: 10.1145/3219104.3219147.

[13] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A Survey of DevOps Concepts and Challenges," ACM Comput. Surv., vol. 52, no. 6, pp. 1–35, Nov. 2019, doi: 10.1145/3359981.

[14] B. R. Cherukuri, "Future of cloud computing: Innovations in multi-cloud and hybrid architectures," World J. Adv. Res. Rev., vol. 1, no. 1, pp. 068–081, Feb. 2019, doi: 10.30574/wjarr.2019.1.1.0002.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 11, Issue 1, November 2021 Impact Factor: 5.731

[15] 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.
[16] A. A. Ur Rahman and L. Williams, "Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices," in Proceedings of the International Workshop on Continuous Software Evolution and Delivery, May 2016, pp. 70–76. doi: 10.1145/2896941.2896946.
[17] S. Achouche, U. Bhaskar, Yalamanchi, and N. Raveendran, "Method, apparatus, and computer-readable medium for performing a data exchange on a data exchange framework," 10387195, 2019
[18] S. Chinamanagonda, "Enhancing CI / CD Pipelines with Advanced Automation - Continuous integration and delivery becoming mainstream," Int. J. Nov. Res. Dev., vol. 5, no. 4, pp. 1–15, 2020.
[19] B. R. Cherukuri, "Microservices and containerization: Accelerating web development cycles," World J. Adv. Res. Rev., vol. 6, no. 1, pp. 283–296, Apr. 2020, doi: 10.30574/wjarr.2020.6.1.0087.
[20] N. K. Ale, "Integrating Performance Testing into CI/CD Pipelines for Test Automation," J. Sci. Eng. Res., vol. 2020, no. 6, pp. 272–278, 2020, doi: 10.5281/zenodo.12754783.
[21] 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.
[22] A. D'Ambrogio, A. Falcone, A. Garro, and A. Giglio, "On the importance of simulation in enabling continuous delivery and evaluating deployment pipeline performance," CEUR Workshop Proc., vol. 2248, 2018.
[23] A. N. Azizan and A. Shah, "Effectiveness in using Continuous Integration and Deployment for software development," J. Inf. Syst. Digit. Technol., vol. 2, no. 2, pp. 18–27, Nov. 2020, doi: 10.31436/jisdt.v2i2.83.
[24] F. Zhao, X. Niu, S.-L. Huang, and L. Zhang, "Reproducing Scientific Experiment with Cloud DevOps," in 2020 IEEE World Congress on Services (SERVICES), 2020, pp. 259–264. doi: 10.1109/SERVICES48979.2020.00058.
[25] T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," in Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC 2020, 2020. doi: 10.1109/EDOC49727.2020.00026.
[26] S. Mysari and V. Bejgam, "Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible," in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), IEEE, Feb. 2020, pp. 1–4. doi: 10.1109/ic-ETITE47903.2020.239.
[27] U. Zdun, E. Ntentos, K. Plakidas, A. El Malki, D. Schall, and F. Li, "On the Design and Architecture of Deployment Pipelines in Cloud- and Service-Based Computing - A Model-Based Qualitative Study," in 2019 IEEE International Conference on Services Computing (SCC), IEEE, Jul. 2019, pp. 141–145. doi: 10.1109/SCC.2019.00033.
[28] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different CI/CD Tools integrated with cloud platform," in Proceedings of the 9th International Conference On Cloud Computing, Data Science and Engineering, Confluence 2019, 2019. doi: 10.1109/CONFLUENCE.2019.8776985.
[29] S. A. I. B. S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference, 2018. doi: 10.1109/MERCon.2018.8421965.