BEYOND AUTOMATION

R. Sampath[1], S Vishwa[2], S Pranaav[3] , S P Murali Magesu[4] and A Sakthi Meenakshi[5]
[1,2,3,4,5]Sri Sai Ram Institute Of Technology, Sairam College Rd, Sai Leo Nagar, West Tambaram, Chennai, 600044, Tamil Nadu

Article Information:

Received: 15-11-2023
Revised: 30-11-2023
Published: 15-12-2023

Keywords:
DevOps; Continuous Integration; Continuous Delivery; Automating Pipelines; CI,CD.

Correspondence Email:

sit20it113@sairamtap.edu.in
sit20it048@sairamtap.edu.in
sit20it025@sairamtap.edu.in
sit20ad024@sairamtap.edu.in
sampath.it@sairamit.edu.in

ABSTRACT

Within the field of software engineering, CI/CD refers to the combination of continuous integration (CI) and continuous delivery (usually) or deployment (sporadically). The foundation of contemporary DevOps operations is the continuous integration and continuous delivery pipeline, or CI/CD pipeline. Continuous practices, such continuous integration, delivery, and deployment, are used by the software development industry to assist businesses in reliably and consistently releasing new features and products.We automate the software delivery process with a CI/CD pipeline. Code is developed, tests are run (CI), and a new version of the application is securely delivered (CD) as part of the pipeline. Pipelines that are automated remove human error, give developers standardized feedback loops, and improve the efficiency of product iteration. Our software development lifecycle can be more effectively automated and allows us to keep an eye on code changes, new features, possible problem fixes, and other things by integrating a CI/CD pipeline. Continuous deployment can also be denoted by the "CD" in CI/CD, which stands for continuous integration and delivery. Automating steps in the CI/CD pipeline is what is meant by both continuous delivery and continuous deployment. but Continuous deployment, however, goes a bit beyond. Making the deployment of new code simple is the goal of continuous delivery. Continuous deployment automates the deployment phase, enabling teams to take a "hands-off" approach to the process. Teams may ship changes on a daily, weekly, monthly, or even hourly basis using an automated CI/CD pipeline, and every step of the process can be optimized. We can swiftly introduce new features and adjustments, enabling us to react to emerging trends and resolve any problems. We don't want the system to abruptly shut down when periodic maintenance is required.

1. INTRODUCTION

CI/CD is revolutionary in the fast-moving world of software development. It is comparable to an extremely productive assembly line for producing and distributing software. Continuous Integration, or CI for short, is the process of continuously assembling the code that developers write. CD can either mean Continuous Delivery, making it easy to launch new code, or Continuous Deployment, where the process is so automated that teams can be hands-off. Imagine a pipeline where code is automatically built, tested, and a new version of the software is safely deployed. This pipeline is the backbone of modern DevOps operations, a way of working in software development that relies on teamwork and automation. The magic of CI/CD lies in automation. By automating these processes, we eliminate manual errors, provide quick feedback to developers, and speed up the whole cycle of creating, testing, and releasing software. This is crucial in a world where staying up-to-date and fixing issues swiftly is essential.In this paper, we'll explore the ins and outs of CI/CD. We'll see how it helps teams ship changes regularly – whether it's every hour, day, or week – allowing them to respond quickly to trends and address any problems. We'll also look at how CI/CD helps in routine maintenance without shutting down the entire system. This is achieved by using microservices, a way of organizing code so that only specific parts of the system are taken down for maintenance instead of the whole thing. So, buckle up as we dive into the world of CI/CD, where automation meets innovation in the quest for faster, more reliable software development.

1.1 LITERATURE REVIEW

[1] The groundbreaking study emphasizes the value of automation in the delivery process and establishes the foundation for understanding continuous practices in software development. In addition to presenting helpful implementation ideas for CI/CD pipelines, the authors give insights into the fundamentals of continuous integration and delivery.

[2] Drawing on extensive research, it explores the science underlying high-performing technology companies, drawing on a wealth of research. It draws attention to the relationship that exists between corporate culture, business results, and continuous delivery methods. The authors offer a data-driven method for comprehending how software delivery performance is affected by CI/CD.

[3] Although it does not only concentrate on CI/CD, the insightful information about the fundamentals of large-scale system dependability. It examines how software development and operations interact, placing a focus on automation and ongoing development. It is essential to comprehend these ideas in order to build efficient CI/CD pipelines.

[4] With an emphasis on architecture, this review explores the usage of microservices in CI/CD deployments. It looks at how normal maintenance and deployment procedures may be made more efficient by dividing large programmes into smaller units called microservices. In order to provide light on the advantages and difficulties of implementing a microservices architecture inside the CI/CD framework, the research assesses case studies and industry practices.

[5] A comparison of several CI/CD approaches and how well they work in diverse software development contexts. It explores the nuances of continuous integration, delivery, and deployment models by looking at factors including project size, team size, and development speed. The study aims to help practitioners select the most appropriate CI/CD model for their specific project needs and organizational environment.

2. RESEARCH METHODS

A. Automated WebDevelopment Environment

During this phase, we integrated Jenkins, GitHub, Git/GitHub, and the HTTPD web server into an automated web development environment running on a Linux server. The code just has to be committed from the developer's local repository. GitHub will use webhooks to automatically transmit the code to Jenkins whenever there is a commit to the GitHub repository. Jenkins will then immediately launch the website into the Linux server's apache2 httpd server.

B. Automated WebSite Deployment within Docker

We have utilized Git/GitHub, Jenkins, and Docker Container to make an automated website deployment in this phase, which is an Automated Website Deployment within a Docker Container.

C. Containerization Within Container

This phase, we have made advantage of the Docker ideas to create an Automation Project that launches Docker within Docker, allowing the Project to be delivered without any issues from one system to another. When you need to transmit your project together with its entire environment, Docker Concepts come in extremely handy.

D. Deployment of CI/CD DevEnv/ProductionEnv

We developed Continuous Deployment in this phase with Git, Jenkins, and Docker, with a primary focus on automating the deployment of the Developer and Production environments.

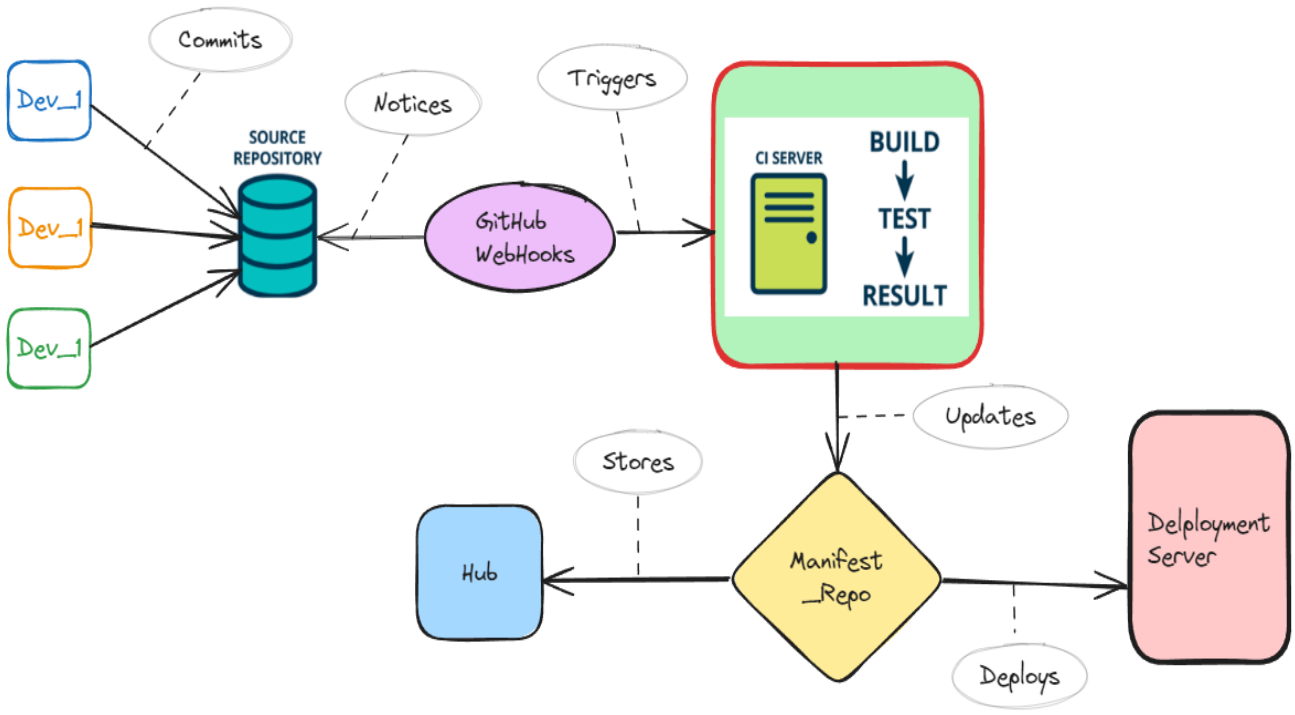

Fig 1. End to End CI/CD Implementation

3. RESULT AND DISCUSSION

Significant improvements in developer productivity and error reduction are achieved by automating the CI/CD workflow. Standardized feedback loops are included to guarantee that developers obtain timely code insights, facilitating the rapid detection and resolution of issues. Consequently, this speeds up the iteration process and encourages an agile and responsive development lifecycle. In order to automate processes and streamline monitoring, a continuous integration/delivery (CI/CD) pipeline must be integrated into the software development life cycle. Organizations may monitor code updates, new features, and possible bug fixes more carefully and reduce manual mistakes by automating the delivery process. Teams are able to make data-driven decisions because of this increased visibility, which makes the software ecosystem more effective and flexible.

4. CONCLUSIONS

4.1 Lastly, examining Continuous Integration and Continuous Delivery/Deployment (CI/CD) within the framework of modern DevOps operations illuminates a groundbreaking avenue for software development techniques. The CI/CD pipeline, the cornerstone of this advancement, has proven successful in automating software delivery, cutting errors, and encouraging a continuous improvement culture.

4.2 This study shows that pipeline automation is important for reasons other than just operational effectiveness. Accelerated product iteration and increased response to market trends are enabled by the CI/CD pipeline, which removes human mistakes and gives developers standardized feedback loops. This in turn gives development teams the ability to precisely and nimbly navigate the constantly shifting terrain of software requirements.

4.3 The software development lifecycle's incorporation of a CI/CD pipeline is a calculated step towards effective automation and careful observation. The adaptability and robustness of the whole development ecosystem are increased by the pipeline's capacity to manage code changes, roll out new features, and handle possible issue fixes in a methodical manner.

4.4 Teams may be flexible in their deployment strategy by carefully examining the subtleties of both continuous delivery and deployment. Teams may ship changes at their desired frequency using the automated CI/CD pipeline, regardless of whether they choose the "hands-off" approach of continuous deployment or the simplicity of continuous delivery. This adaptability helps firms to react proactively to new trends and quickly to problems, while also optimizing each step of the development process.

4.5Routine maintenance brings a practical aspect to the conversation about CI/CD. The solution's use of microservices architecture highlights a careful strategy for preserving system operation throughout upgrades. Organizations balance continuous service availability and system maintainability by handling maintenance selectively at the microservices level.

4.6 All things considered, the CI/CD paradigm as it has been dissected in this work goes beyond simple technological progress. It appears as a revolutionary force reshaping the software development culture, where automation, effectiveness, and flexibility come together. It's clear that the transition to a more robust and agile software development environment is well under way as more and more companies adopt the CI/CD concepts.

5. REFERENCES

[1] "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" by Jez Humble and David Farley (2010)

[2] "Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations" by Nicole Forsgren, Jez Humble, and Gene Kim (2018):

[3] "Site Reliability Engineering: How Google Runs Production Systems" by Niall Richard Murphy, Betsy Beyer, Chris Jones, and Jennifer Petoff (2016)

[4] "Microservices Architecture: Enhancing Maintenance and Deployment in CI/CD Pipelines"

[5] "Continuous Practices in Software Development: A Comparative Study of CI/CD Models"

[6] Debois, P. (2010). Ops, Dev, and PaaS. Javalobby.

[7] Kim, G., et al. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press.

[8] Somasundaram, S., & Sabitha, R. (2016). Comparative Analysis of Continuous Integration and Continuous Deployment. International Journal of Computer Applications.

[9] Paul, P. (2013). Continuous Delivery: What Huge Enterprises Can Learn from the Lean Startup Movement. InfoQ.

[10] Gousios, G., & Spinellis, D. (2012). GHTorrent: GitHub's Data from a Firehose. MSR.

[11] Newman, S. (2016). Microservices: Evolution of Building Modern Applications. NGINX.

[12] Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A Software Architect's Perspective. Addison-Wesley.