IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

Efficient Devops Workflow With Jenkins

Srungarapu Rama Krishna[1], Yenumula Venkata Durga[2], Lekkala Prem Venkatesh[3], P.S.V.S. Sridhar[4]

UG students, Department of CSE, Koneru Lakshmaiah University[1-3]

Assistant Professor, Department of CSE, Koneru Lakshmaiah University[4]

Abstract: In today's fast-paced software development environment, DevOps practices have become essential for organizations to achieve efficient collaboration, continuous integration, and delivery. Jenkins, an open-source automation server, plays a pivotal role in facilitating DevOps workflows by automating various stages of the software development lifecycle. This paper aims to provide a comprehensive analysis of how Jenkins enhances DevOps workflows, examining its key components, features, and integration capabilities. Through an exploration of relevant literature and case studies, this research sheds light on the benefits, challenges, and best practices associated with incorporating Jenkins into DevOps environments. Insights gained from this study can guide organizations in optimizing their DevOps processes and leveraging Jenkins effectively for improved software delivery.

Keywords: DevOps, Jenkins, Automation, Continuous Integration, Continuous Delivery, Software Development Lifecycle.

I. INTRODUCTION

The adoption of DevOps practices has revolutionized the way software is developed, tested, and deployed. DevOps emphasizes collaboration, automation, and integration between development and operations teams to achieve faster and more reliable software delivery. At the heart of DevOps workflows lies Jenkins, an open-source automation server that enables continuous integration and continuous delivery (CI/CD).

Jenkins automates various tasks throughout the software development lifecycle, from building and testing code to deploying applications in production environments. This paper aims to delve into the role of Jenkins in enhancing DevOps workflows, examining its functionalities, integrations, and impact on software delivery.

II. KEY INSIGHTS FROM LITERATURE

Moreover, recognized as the software development, project development, and deployment life cycle (SDLC), it delineates the standardized methodologies or procedures indispensable for software evolution. This cycle entails numerous stages guiding the software life cycle that development sectors adhere to for generating software products. SDLC has witnessed various adaptations, transitioning from conventional and agile methodologies to the more contemporary DevOps culture.

Within the domain of DevOps research papers, a myriad of contemporary and relevant literature is scrutinized and examined. DevOps, denoted as "Development and Operations," emerges as a novel concept in the software evolution narrative, with the primary objective of mitigating disparities among different teams by bridging communication gaps between development and operations teams [3]. Conventional methodologies often overlook these collaborative imperatives. Lwakatare et al., in their investigation of five organizations, endorse the coordination of development and operations teams, corroborating the significance of DevOps in fostering collaboration [4].

While there exist several theories opposing the implementation of DevOps, citing challenges and the absence of performance metrics, prominent researchers have advocated caution in its practical application [5]. For instance, in their survey, authors [6] extensively examined the barriers to adopting DevOps. Other studies have explored DevOps practices as a means for companies to achieve enhanced performance and deliver higher-quality software [7].

Echoing this sentiment, DevOps emerges as a trend aimed at automating continuous software release tasks while ensuring accuracy and resilience, as highlighted by Ronny Olguin [8] and Ramtiin Jabbari et al. [9]. The definition of DevOps has been thoroughly scrutinized in the literature, with authors [9] concurring that DevOps enhances the adaptability of software development methodologies.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

III. RESEARCH METHOD

For our study, we utilized project management and deployment tools such as Jenkins, alongside a range of hybrid model tools including GitLab, Nagios, Selenium, and Ansible. These tools were integrated into Jenkins through various plugins. To facilitate implementation, we developed a web-based interactive interface as a prototype for the software application. Below are the specifics:

Phase I:

- In this phase, the creation of the best web-based graphical interface application will be initiated and integrated into the Jenkins Climate environment.
- For source code management, an interactive web-based Java application has been designed and made accessible as a repository on GitHub.

Phase II:

The aim of this step is to determine the hybrid model tools recommended in the toolchain and the environments required for their proper installation, as per the findings of the literature review. • Creation of a Jenkins Project Management tool pipeline from Java. • Installation of various plugins recommended by a hybrid automated approach to enhance Jenkins' capabilities.

Phase III:

- The objective of this step is to evaluate quality criteria based on the implementation results. • Subsequently, the delivery pipeline will be demonstrated, illustrating how each build/release incorporates the verification of various metrics as quality parameters. • For continuous deployment, the current research considers automation tools such as Jenkins, Selenium, GitLab, and Tomcat Server. The utilization of these automation tools as integral components of the project implementation will be detailed in the following section.

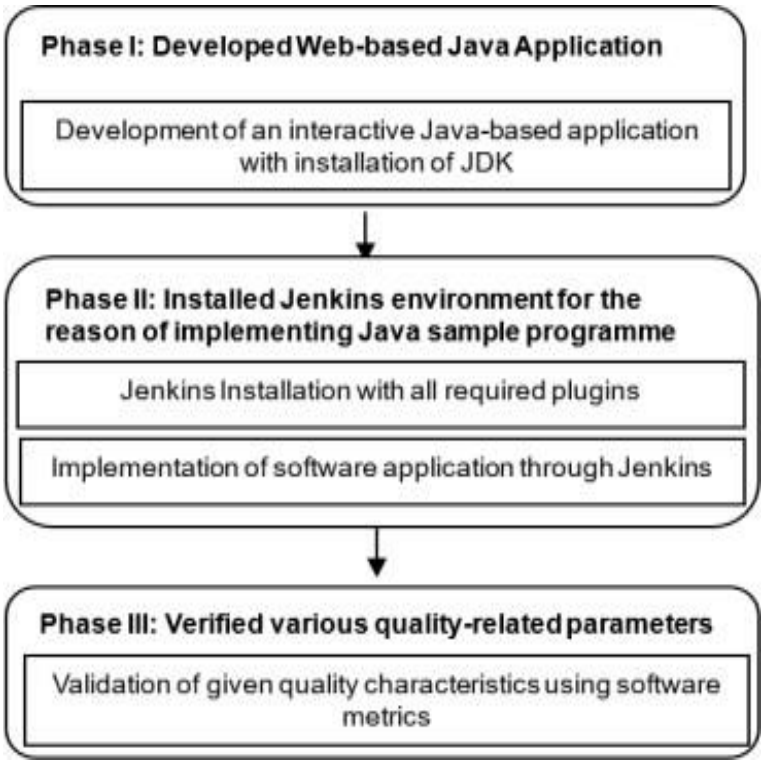

Figure1. Research methodology adopted for current research work.

The diagrammatic representation of the phases and various actions prescribed by the methodology is depicted in Figure 1 above, also known as the design of the current research.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

IV. HYBRID MODEL PROPOSAL FOR DEVOPS-BASED AUTOMATION TOOLS

The proposed hybrid model article [2] examined the integration of DevOps culture in IT sectors. DevOps was identified as a mechanism for fully automating development and operations processes using a diverse range of technologies. Its primary objective is to address various challenges encountered in the industry, such as delayed software releases, delivery, deployment, and maintenance issues. This endeavor facilitated the development of an Integrated Tool Chain (ITC) through performance evaluation comparisons of different automation tools. By eliminating barriers at each level, this selective chain of automation tools enhances the efficiency of the development life cycle. Subsequently, the design of the ITC leads to the emergence of a hybrid DevOps paradigm for software development automation. The diagram below illustrates the hybrid automated approach encompassing these distinct tool chains across various stages of the DevOps Culture.

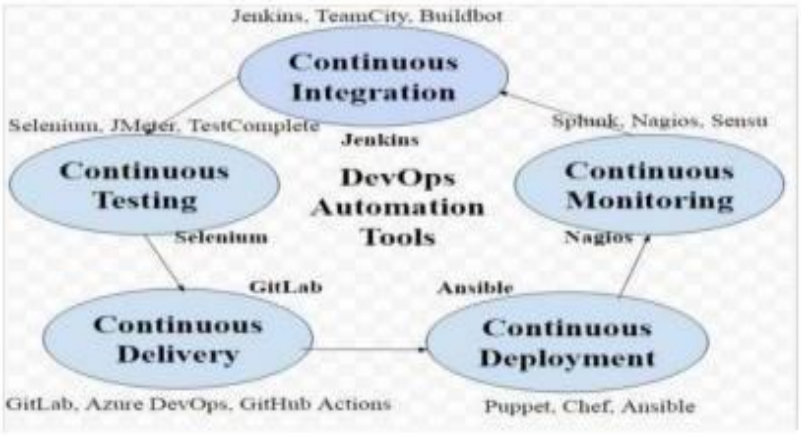

Figure2.Hybrid model for DevOps-based chain of selective automation tools [2]

Figure (2) illustrates our proposed hybrid approach of selected DevOps automation tools from a distinct group. The underlying paper employs hybrid automation models to assess the reliability of work, a task that was not feasible with previously published case studies. We developed a web-based application using the JDK environment to evaluate an automation model, which serves as a data source for this study.

V. IMPLEMENTATION OF DEVOPS-BASED HYBRID MODEL THROUGH SAMPLE JAVA APPLICATION

The paper utilizes a three-phased architecture to construct the DevOps hybrid automation tool paradigm. The flowchart below depicts these three phases along with other procedures necessary for the correct implementation of the model.

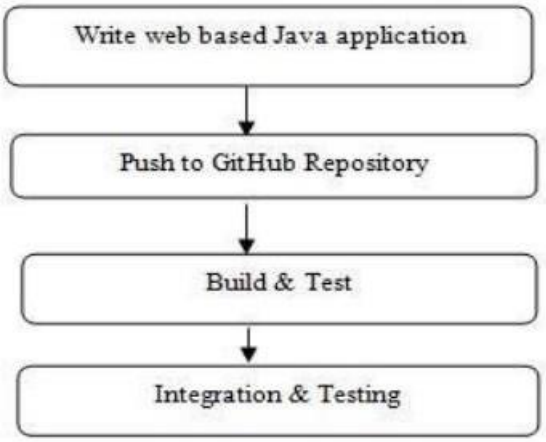

Figure3. Process flow of DevOps implementation approach

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

As illustrated in figure (3) above, Jenkins serves as a continuous integration and build tool, facilitating both continuous software delivery and integration within the CI/CD pipeline. GitHub and GitLab repositories are utilized for code management, complemented by various automation technologies integrated into Jenkins as plugins, akin to those in the Integrated Tool Chain (ITC) [2]. Selenium is employed as a plugin for creating and executing test cases, while Nagios oversees operational processes, and Ansible handles deployment.

The research methodology begins with the development of a prototype Java program, initiated within a local repository and subsequently uploaded to GitHub, where it is seamlessly linked with Jenkins, as outlined in the flowchart. Maven is utilized within Jenkins to orchestrate the build process, providing insights into the Java application. Selenium is tasked with executing both unit and integration tests. Jenkins automates continuous integration seamlessly, followed by continuous delivery and deployment facilitated by various plugins. The ensuing screenshots provide a visual representation of the system's implementation outcomes, supplemented by detailed descriptions.

Step1: The initial step in developing the Java-based online application website is to install the JDK. This website provides a user interface for adding or removing completed tasks. The Java web code of the website maintains a record of both active and completed tasks, as well as a list of pending tasks. The status of these tasks can be modified to incomplete, active, or complete, or they can be deleted from the list altogether. The local code structure, which alters the relevant XML file, is housed within the application's src directory.

Step 2: After completing the Java code, the next step is to install the Jenkins tool for continuous integration and build purposes. To accomplish Jenkins, build tasks with the website route provided as the repository address, a freestyle project type must be defined. Additionally, for the development of a Java project and the execution of Java code in Jenkins, integration of numerous plugin tools, as specified by the hybrid model in [2], is necessary. Many of these plugins are already pre-installed, including Ansible, Selenium, Nagios, and GitLab. The integration of all these tools as plugins in a single window not only enhances the functionality of the tool but also facilitates communication between the operations team and developers.

Step 3: The subsequent step involves installing the Tomcat server to create the web application's WAR file. The Tomcat directory structure is utilized to generate the application's WAR file. Following this, a new Jenkins build process needs to be developed to build an application and deploy the packaged WAR file to Tomcat, now that the Jenkins Tomcat deployment plugin has been installed. Any modifications to the code necessitate rebuilding the application and restarting the Tomcat server. This process entails adding various jobs following the installation of an application on a Tomcat server in a continuous DevOps environment.

VI. SELECTION AND OF EVALUATION OF DIFFERENT SOFTWARE METRICS

The performance of software or systems is consistently a pivotal factor in assessing software quality. Software quality encompasses the desired attributes of software products, as well as its individual components and processes. To gauge the quality of software products and processes, various metrics are devised. These metrics are essential for assessing the accuracy and efficiency of software. Additionally, evaluating software development performance is another critical step in enhancing software productivity and effectiveness.

There are two main categories of software metrics: Process metrics, Project metrics. Process metrics analyze various elements of the development process, while Project metrics assess product development phase by phase.

In our research, we selected several predefined software metrics from current literature to validate both software processes and products. The table below presents the metrics considered along with the expected outcomes:

Software Metric Type	Name of metric	Best outcome
Process	Project Risk Identification	High
Project	Project Defect Density	Low
	Project Deployment Frequency	High

Table 1. Categorization of different Software Metrics

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

(i) Project Risk Coverage (PRC)

Various approaches and techniques have been developed to conceptualize, evaluate, and manage software risks [1]. Software metrics provide a measurable method for assessing and monitoring quality factors, as well as for identifying potential hazards associated with a particular software product [10].

The risk coverage of each component is given as CRC = $\sum Wf\ n\ f$=1

Where W is the risk weightage and n are the total number of criteria. The estimate for project risk coverage is given as –

PRC = $\sum RCn\ n\ i$=1, n > 0

Where n is total number of components/ modules in system

(ii) Project Defect Density (PDD)

Every software inevitably contains flaws or faults. While some errors may be minor and overlooked, developers should consistently address errors that have already occurred or pose a risk to the product, as they can significantly impact the software's performance.

Defect density for individual software components is calculated as – CDD = $\frac{\text{Number of defects known}}{\text{Software size in KLOC}}$

Where KLOC = LOC /1000

(iii) Project Deployment Frequency (PDF)

Perera et al. and Ziadoon Otaiwi et al. examined deployment frequency as a primary objective of DevOps in their studies on DevOps Quality [11] [12]. According to their findings, increasing deployment frequency enables DevOps to surpass non-DevOps performance by more than 40 times. Deployment frequency measures the frequency at which small-sized code changes are distributed. Consequently, frequent deployments or releases with minimal change requirements outperform infrequent deployments with significant change needs. Deployment frequency is closely associated with continuous delivery and serves as a metric of success for high-performing organizations.

The expression for each component deployment frequency measurement is provided as –

PDF= $\frac{\text{Total number of Deployments}}{\text{Per unit Time}}$

The time unit for deployment frequency is determined by the magnitude of the project. For example, if the project size is measured in Kilo Lines of Code (KLOC), the deployment frequency may be weekly. However, since the data collection in our study is relatively small, the time unit is measured in hours.

The phrase below gives an estimate for project deployment frequency throughout the whole project – PDF= $\sum_{i=1}^{n} \text{DFn}$ and n > 0

Here n is count of system components.

A high deployment frequency number indicates that the system is operating efficiently and smoothly.

VII. CONCLUSION

This study focuses on the hybrid DevOps automation tool paradigm, which has been previously proposed and is utilized to expedite the production of high-quality software products. The current research project combines the Jenkins project integration tool with a Java web application deployed on an Apache Tomcat server. Incorporating DevOps into a continuous environment not only reduces development and delivery times but also enables frequent software updates and continuous delivery of the same level of quality and efficiency, as evidenced by tabular metric values. Due to its ongoing advantages, DevOps has become the latest buzzword in the IT sector. Young researchers and students can gain valuable insights into DevOps and its automation technologies by examining the tabular results. In addition to selecting the appropriate tool chain, software developers can expedite high-quality software delivery. Furthermore, the current research can be expanded to include new metrics that compare DevOps with other traditional methods already in use.

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 4, April 2024

DOI: 10.17148/IJARCCE.2024.13486

REFERENCES

[1]. TERJE AVEN (2016), RISK ASSESSMENT AND RISK MANAGEMENT: REVIEW OF RECENT ADVANCES ON THEIR FOUNDATION, EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, ELSEVIER, VOL 253, ISSUE 1, PP 1-13.

[2]. Poonam Narang, Pooja Mittal, Hybrid Model for Software Development: An Integral Comparison of DevOps Automation Tools, Indonesian Journal of Electrical Engineering and Computer Science (IJEECE), IAES Publishers, ISSN 2502-4752, Scopus, SJR 2020 (Q3 0.241), Vol 27, No 1, July 2022, pp 456-465.

[3]. Debois P., (2008), Agile infrastructure and operations: how infra-gile are you? Agile 2008 Conference, IEEE, Toronto, ON, Canada, ISBN: 978-0-7695-3321-6

[4]. Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. Information and Software Technology, 114, 217-230.

[5]. Khan AA, Shameem M. Multicriteria decision-making taxonomy for DevOps challenging factors using analytical hierarchy process. J Softw-Evol Proc. 2020; 32(10):11-13, e2263.

[6]. Leite L, Rocha C, Kon F, Milojicic D, Meirelles P. (2019), A survey of DevOps concepts and challenge, ACM Computing Surveys (CSUR). 2019; 52(6):1-35

[7]. Bolscher R, Daneva M. (2019), Designing software architecture to support continuous delivery and DevOps: a systematic literature review, ICSOFT. 2019: 27-39.

[8]. Ronny Olguin (2019), DevOps Challenges and Implications, University of Murcia, Spain, 2019.

[9]. Ramdin Jabbari, Nauman Bin Ali, Binish Tanveer, Kai Petersen (2016), what is DevOps? A Systematic Mapping Study on Definitions and Practices, ACM Digital Library, published in Proc. of The Scientific Workshop Conference XP2016.

[10]. Dr Issa Traore (2006), Software Architecture, Chapter 6, EOW 415.

[11]. Pulasthi Perera, Roshali Silva, Indika Perera (2017), Improve Software Quality through Practicing DevOps, 2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ITCer): 013-018.

[12]. Alok Mishra, Ziadoon Otaiwi (2020), DevOps and software quality: a systematic review, Computer Science Review, Elsevier, Vol 38, 100308.

