2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS) | 979-8-3503-0085-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSSAS57918.2023.10331631

Efficient Automation of Web Application Development and Deployment using Jenkins: A Comprehensive CI/CD Pipeline for Enhanced Productivity and Quality

Badisa Naveen
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India
naveenbadisa2708@gmail.com

Jayanth Krishna Grandhi
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India
grandhijayanth637@gmail.com

Kallam Lasya
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India lasyakallam111@gmail.com

Eda Mokshita Reddy
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India
mokshitareddyeda@gmail.com

Nulaka Srinivasu
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India srinivasu28@kluniversity.in

Sunitha Bulla
Department of Computer Science Koneru Lakshmaiah Education Foundation
Guntur, India
sunithabulla@kluniversity.in

Abstract— **Nowadays software developers are building web applications, to the main requirement is that it work well, be free from issues and reach users quickly. But doing this can be challenging. That is where Continuous Integration (CI) and Continuous Deployment (CD) come with a concept called Jenkins. It has become a vital tool for software developers globally because of automation. This research study explores the impact of Jenkins automation on improving web application development by automating essential tasks like testing and deployment. It also examines notable benefits such as early issue detection, increased developer focus, time savings, and enhanced reliability. Through a practical example using the Angular framework for web applications, this study underscores the considerable benefits of integrating Jenkins automation. Overall, this research highlights the importance of Jenkins in enhancing software development processes, making them more efficient and effective.**

Keywords— CI/CD Pipeline, Jenkins Automation Server, Web Application Development, Docker Containerization, Code Quality, Automated Testing, Linting, Software Development Lifecycle, Efficiency Improvement, Early Issue Detection, Jenkins Integration.

I. INTRODUCTION

In the fast-paced landscape of modern software development, the demand for web applications that are not only robust and efficient but also swiftly delivered to end-users has reached unprecedented heights [1]. Developers are tasked with creating applications that not only cater to a wide range of user needs but also adhere to the highest standards of quality. However, achieving this equilibrium can be a daunting challenge, given the complexities of debugging, testing, and continuous deployment in an agile development environment. This is where Continuous Integration (CI) and Continuous Deployment (CD) strategies, empowered by automation, have emerged as a lifeline for developers striving to meet these formidable demands [2]. CI/CD pipelines are designed to improve the effectiveness of the software development by procedure of building, testing and deploying. This automation will allow the developers to consistently deliver new features and fix the bugs faster by reducing the time and effort required for the manual tasks. Continuous Integration involves the seamless merging of code contributions from multiple developers into a shared repository, followed by an automated testing process to identify and rectify conflicts and issues at the earliest stages of development. Continuous Deployment takes this concept a step further by automating the deployment of code changes to production environments, ensuring swift delivery to end-users. At the heart of these strategies lies Jenkins, a pivotal tool that has redefined how developers approach web application development [3].

Jenkins, an open-source automation server, has rapidly evolved into an indispensable asset for software developers globally. By orchestrating various stages of the development pipeline, from code integration and testing to deployment, Jenkins eliminates manual interventions, reduces human error, and streamlines the development lifecycle. The pivotal role of Jenkins automation in this context underscores its potential to revolutionize web application development.

Jenkins serves as the backbone of continuous integration and continuous delivery (CI/CD) pipelines, facilitating seamless integration of code changes, automated testing, and deployment to various environments. Its user-friendly web interface allows developers, testers, and operations personnel to configure and monitor complex workflows with ease, reducing manual intervention and increasing overall productivity.

Jenkins automation transcends the boundaries of mere convenience; it encapsulates a paradigm shift in how developers perceive and execute their responsibilities [4]. The automation of routine and critical tasks liberates developers from the shackles of monotonous manual procedures, empowering them to allocate more time and cognitive resources to the creative aspect of their work. In doing so, Jenkins contributes not only to the efficiency of the development process but also to the quality of the end product.

This research study delves into the profound impact of Jenkins automation on enhancing web application development. By exploring its benefits through practical implementation and analysis, this study seek to shed light on the ways in which Jenkins automates essential tasks, enables early issue detection, heightens developer focus, saves time, and bolsters overall reliability. Through a case study centred around the Angular framework, the study unveil the tangible advantages of integrating Jenkins automation, emphasizing its potential to elevate software development processes to new heights.

II. Literature Survey

To gain valuable insights into the impact of Jenkins automation on web application development, we turn to the work of Fowler and Foemmel [1]. They introduced the practice of Continuous Integration, highlighting the frequent integration of code changes and the vital role of automated testing in preventing issues at an early stage. This resonates with Jenkins' core functionality, which automates testing processes, aiding developers in spotting and addressing problems before they escalate.

DuVall, Matyas, and Glover [2] explored the benefits of Continuous Integration within agile development teams. They emphasized that automation tools like Jenkins streamline the collaboration process by automatically integrating code changes, thereby reducing the effort required for manual integration. This not only accelerates development but also enhances team productivity. Jenkins, by automating integration tasks and facilitating continuous feedback, reinforces this collaborative approach.

Farley and Humble [3] extended the concept of Continuous Integration to Continuous Delivery, where the goal is to quickly deliver new software updates to users. They highlighted the significance of automation in achieving this goal, as manual processes can hinder the pace of deployment. Jenkins plays a crucial role in automating the deployment pipeline, ensuring that new updates are efficiently and reliably delivered to end-users, aligning with the principles of Continuous Delivery.

Cawood and Gerber [4] specifically examined the implementation of Continuous Integration and Deployment in web applications built using AngularJS and Node.js. Their research shed light on the benefits of integrating automation tools like Jenkins into the development workflow. By ensuring that code changes are consistently tested and deployed, Jenkins contributes to the creation of stable, high-quality web applications

These insights collectively underscore the importance of Jenkins automation in web application development. Early issue detection, efficient team collaboration, rapid software updates, and enhanced application stability emerge as recurring themes in the literature.

III. Jenkins Automation: An Overview

Jenkins is a powerful open-source automation server that has become a cornerstone in modern software development processes [5]. It serves as an orchestration tool that automates various aspects of the software development lifecycle, from integrating code changes to deploying applications. Its significance lies in its ability to alleviate the manual burdens faced by developers, enabling them to streamline their workflows and deliver higher-quality software more efficiently.

Jenkins achieves automation through a structured workflow that integrates seamlessly into the development process [1]. At its core, Jenkins automates the process of integrating code changes from different contributors into a shared repository. This integration triggers an automated build process, which involves compiling the code and executing a series of tests to identify issues. Once the code passes these tests, Jenkins automates the deployment of the application to staging or production environments.

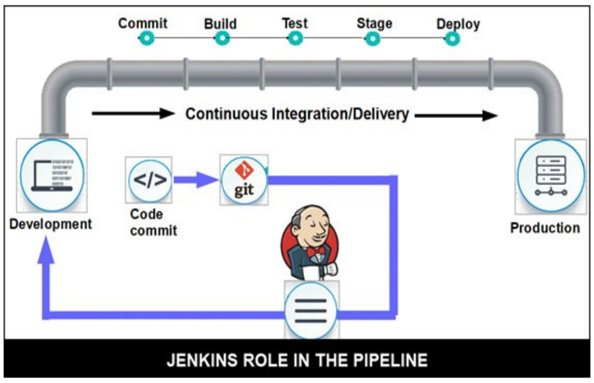


Fig. 1. Jenkins Role in the pipeline

The adoption of Jenkins automation offers a plethora of benefits that significantly impact the efficiency and reliability of software development [7]. One of the standout advantages is early issue detection. By automatically running tests and checks after every code change, Jenkins promptly identifies defects, allowing developers to address problems before they escalate. This not only ensures the quality of the software but also reduces the effort required for troubleshooting later in the process.

Jenkins also contributes to time savings by reducing the manual effort needed for repetitive tasks [8]. Developers can focus on writing code and designing features instead of spending excessive time on integration, testing, and deployment tasks. This increased developer productivity translates into faster development cycles and quicker delivery of software updates.

Moreover, Jenkins automation sharpens developer focus by removing the distraction of manual tasks [9]. This enables developers to concentrate on creating innovative solutions and improving the user experience of the application. Additionally, the consistency and repeatability of automated processes enhance the reliability of the development pipeline, ensuring that the software is delivered consistently and predictably.

IV. INTEGRATION OF CI/CD

The integration of Jenkins into the development workflow involves the establishment of a Continuous Integration and Continuous Deployment (CI/CD) pipeline. In this context, Jenkins becomes a key orchestrator of the pipeline, automating the entire process from code integration to deployment [5].

In a typical scenario, as developers commit code changes, Jenkins automatically detects these changes and triggers the CI/CD pipeline. Jenkins fetches the latest code, compiles it, and performs automated tests to verify its quality. This stage ensures that code changes do not break existing functionality and maintain code standards.

After successful testing, Jenkins moves on to the deployment phase. Depending on the setup, Jenkins can deploy the application to a staging environment for final testing or directly to production. This automated deployment ensures that the latest changes are available to users quickly and reliably.

Through this integration, Jenkins provides visibility into the status of the development pipeline. Developers can monitor the progress of builds, tests, and deployments, allowing for early detection of issues and facilitating prompt corrective actions. The result is a streamlined and consistent workflow that promotes collaboration and accelerates software delivery.

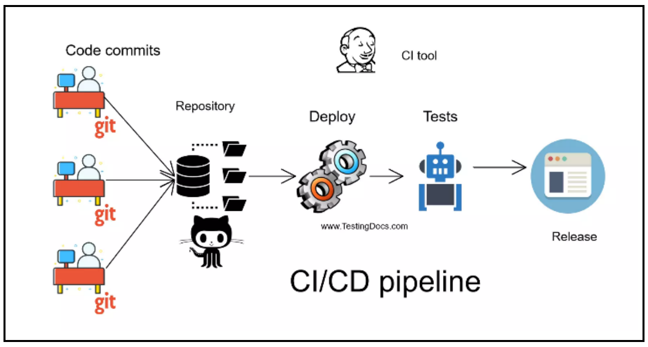


Fig. 2. CI/CD pipeline

Jenkins Pipeline is a powerful tool that takes automation a step further by allowing developers to define the entire software delivery process as code [5]. This approach, known as "Pipeline as Code," offers a structured and reproducible way to define, manage, and automate the steps involved in building, testing, and deploying software.

Traditionally, automation in Jenkins was achieved through configuring jobs using the graphical user interface. Jenkins Pipeline introduces a more flexible and programmable approach. With Jenkins Pipeline, the entire workflow is described in code, typically in a file named Jenkinsfile, which is stored alongside the application's source code.

Implementing a Jenkins Pipeline involves writing a Jenkinsfile that outlines the stages of the software delivery process. A pipeline can include stages like code checkout, building, testing, and deployment. Each stage can be customized with specific instructions, tools, and parameters.

Automating Continuous Integration and Continuous Deployment pipeline in Jenkins involves setting up the configurations by ensuring that the code is continuously integrated, tested and deployed.

V.JENKINS FILE

In the dynamic landscape of contemporary software engineering, the Jenkins file emerges as a pivotal enabler of seamless Continuous Delivery, where the entire deployment pipeline is codified and automated [5]. Jenkin file embodies the essence of "Pipeline as Code," encapsulating the delivery process within a versioned text file that resides alongside the application source code. This paradigm shift revolutionizes the way development teams orchestrate and manage their delivery workflows, bringing a host of benefits that resonate throughout the development lifecycle.

The adoption of Jenkins file heralds numerous advantages, augmenting the agility and quality of Continuous Delivery practices [10]. By residing within the version control repository, Jenkins file inherently inherits the benefits of versioning, facilitating collaborative code reviews, traceable modifications, and audit trails. Reproducibility and consistency flourish as the pipeline's configuration, steps, and dependencies become part of the source code, thus minimizing configuration discrepancies across various stages of the development lifecycle. This seamless integration between pipeline definition and source code fosters collaboration among diverse teams and eliminates the "works on my machine" phenomenon.

Creating a Jenkins file revolves around defining distinct stages within the Continuous Delivery pipeline. These stages encapsulate tasks such as code checkout, building, testing, and deployment. The structured syntax of Jenkins file, be it declarative or scripted, empowers developers to succinctly articulate these stages, their interdependencies, and the requisite tools.

The significance of Jenkins file's impact on modern software delivery is substantiated by an array of authoritative voices [11][12]. Gruver and Matyas shed light on practical Jenkins file implementations, emphasizing integration with version control, thereby safeguarding the pipeline's integrity and facilitating collaborative improvements. The exposition by Quick and Davis unveils Jenkins file's contemporary relevance, particularly in the context of Continuous Delivery within Kubernetes environments.

Creating a Jenkins file revolves around defining distinct stages within the Continuous Delivery pipeline. These stages encapsulate tasks such as code checkout, building, testing, and deployment. The structured syntax of

Jenkins file, be it declarative or scripted, empowers developers to succinctly articulate these stages, their interdependencies, and the requisite tools.

In the ever-evolving landscape of software engineering, Jenkins file emerges as an influential bridge between automation and code, amplifying the capabilities of Continuous Delivery while fostering collaboration and consistency across development teams.

VI. Jenkins Plugins

In the realm of automation, Jenkins Plugins shine as indispensable components, enriching the capabilities and adaptability of the Jenkins automation server [5]. By offering an extensive assortment of modular extensions, these plugins empower developers with the means to customize Jenkins to cater to a wide spectrum of software development requirements.

Jenkins Plugins play a pivotal role in enhancing the functionality and adaptability of the Jenkins automation server, offering a vast array of tools to cater to diverse software development needs [5]. These modular extensions provide a seamless way to integrate third-party tools, extend core functionality, and customize Jenkins according to specific requirements.

The strength of Jenkins lies not only in its core features but also in its extensibility through plugins. With a plugin-driven architecture, developers can augment Jenkins with functionalities that span from version control integrations and build tools to reporting frameworks and deployment orchestration.

Integrating plugins into Jenkins is a straightforward process that involves browsing the Jenkins Plugin Index, selecting the desired plugin, and installing it directly from the Jenkins web interface. Plugins can be configured and managed via the Jenkins dashboard, allowing users to fine-tune their behavior and interactions with other tools.

The advantages of harnessing the Jenkins Plugin ecosystem are multifaceted [6]. First and foremost, the adaptability conferred by plugins enables Jenkins to embrace diverse development stacks, tools, and frameworks, making it a versatile asset across various projects. Scalability, a paramount concern in software development, becomes streamlined as teams can dynamically augment Jenkins with plugins that cater to evolving needs. This specialization ensures that Jenkins remains agile, able to adopt specialized practices without overburdening its core.

Furthermore, the Jenkins community's collaborative efforts in plugin development amplify the ecosystem's value [13]. Joakim Verona's comprehensive guide, "Jenkins: The Definitive Guide," delves into the essentials of plugins and their integration, providing insights into their pivotal role. Additionally, "Jenkins 2: Up and Running" by Jason Smart and Ian Ferguson outlines the significance of plugins in extending Jenkins' capabilities, ensuring adaptability and efficiency across projects.

In essence, Jenkins Plugins constitutes an instrumental avenue through which automation reaches new dimensions of versatility, facilitating seamless integration and tailoring to the intricacies of modern software development.

Steps to follow to Automate CI/CD pipeline in Jenkins:

1.Creating a new job by clicking on "New Item" and choosing the type of job.
2.Select the version control system which is under the "Source Course Management".
3.Choose how you wanted to trigger the build in the "Build Triggers" section.
4.Define the build steps in the "Build" section so that we can compile code, build commands and test.
5.After build step, configure actions in order to achieve artifacts.
6.Include unit tests or integration tests based on the application.
7.Use Jenkins plugins or scripts to copy the artifacts to a server.
8.Configure messages or email notifications to alert the stakeholders about build and deployment results.
9.Define the pipeline script in "JenkinsFile" and configure the pipeline.
10.Save the job configuration and monitor the pipeline regarding build and deployment status.
11.Increase the efficiency of pipeline by optimizing build time and add more tests.

By following the above steps, we can automate the CI/CD pipeline in Jenkins.

VII. Advantages of Jenkins

In the dynamic landscape of modern software development, Jenkins stands as a pillar of efficiency and innovation, offering a range of advantages that collectively shape the way software projects are conceived, developed, and delivered.

One of Jenkins' standout advantages is its unparalleled automation prowess, a cornerstone that significantly enhances development workflows. The automation of tasks such as code integration, testing, and deployment through Jenkins not only expedites processes but also minimizes the risk of human-induced errors, ultimately resulting in more reliable and consistent software releases [5]. As developers are freed from the burden of manual, repetitive tasks, their productivity soars, enabling them to dedicate more time and attention to crafting high-quality code.

Jenkins helps in modularizing the pipeline. The pipeline will be broken down into smaller and reusable pieces which makes the pipeline more efficient, and it is easy to maintain. Jenkins excels in fostering the principles of Continuous Integration (CI) and Continuous Deployment (CD) [1]. By continuously integrating code changes into a shared repository and automating testing processes, Jenkins ensures that issues are caught early, reducing the likelihood of bugs and defects making their way into the final product.

The automated deployment capabilities of Jenkins further empower teams to release updates swiftly and confidently, enhancing the user experience and ensuring a seamless software delivery pipeline.

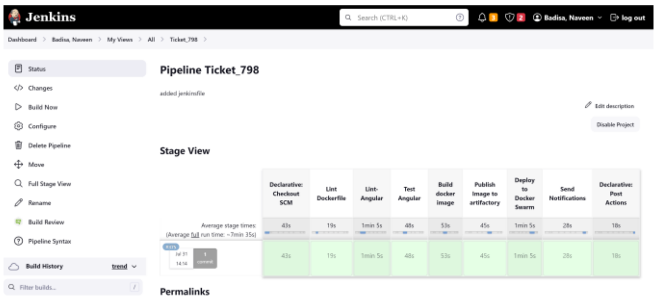
Early issue detection is a cornerstone benefit of Jenkins that contributes to software stability and quality. By automating testing and verification of code changes, Jenkins acts as a vigilant gatekeeper, identifying problems in their infancy and allowing developers to rectify them before they escalate [10]. This proactive approach not only prevents last-minute crises but also cultivates a culture of continuous improvement.

Jenkins also shines as a hub of collaboration and transparency within development teams [2]. With its centralized dashboard, team members gain insights into building statuses, test results, and deployment processes, facilitating open communication and swift decision-making. This transparency fosters a sense of ownership and responsibility among team members, as they collectively work toward delivering top-notch software [14].

Jenkins encapsulates the essence of efficiency, quality, and collaboration that modern software development demands. Its automation prowess, support for CI/CD practices, early issue detection capabilities, and collaborative features position it as an invaluable asset for development teams striving to meet the demands of today's rapidly evolving software landscape [15].

VIII. RESULTS AND ANALYSIS

To evaluate the effectiveness of Jenkins in automating various stages of development, a practical pipeline was executed using the Angular framework for web applications. The pipeline consisted of several stages, including Lint Docker file, Test stage, building a docker image and publishing Image to JFrogArtifactory



Pipeline Ticket_798

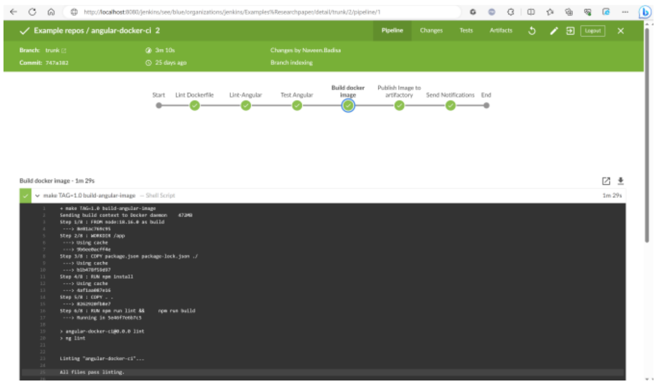
added jenkinsfile

Stage View

	Declarative: Checkout SCM	Lint Dockerfile	Lint-Angular	Test Angular	Build docker image	Publish Image to artifactory	Deploy to Docker Swarm	Send Notifications	Declarative: Post Actions
Average stage times: (Average full run time: ~7min 35s)	43s	19s	1min 5s	48s	53s	45s	1min 5s	28s	18s
Jul 31 14:14 1 commit	43s	19s	1min 5s	48s	53s	45s	1min 5s	28s	18s

Fig. 3. Jenkins pipeline executed stages.

The pipeline execution demonstrated successful automation of each stage. Code compilation ensured that the codebase was syntactically correct and adhered to coding standards. Automated testing, a crucial aspect of the development process, allowed for comprehensive testing of the application's functionalities. Jenkins seamlessly handled the deployment process, moving the application from the staging environment to the production environment once all tests passed successfully.



Build docker image - 1m 29s

make TAG=1.0 build-angular-image — Shell Script 1m 29s

```
+ make TAG=1.0 build-angular-image
Sending build context to Docker daemon   472MB
Step 1/8 : FROM node:18.16.0 as build
 ---> 8e81ac760c95
Step 2/8 : WORKDIR /app
 ---> Using cache
 ---> 9b6ee0acff4e
Step 3/8 : COPY package.json package-lock.json ./
 ---> Using cache
 ---> b1b470f50d97
Step 4/8 : RUN npm install
 ---> Using cache
 ---> 4af1aa087e16
Step 5/8 : COPY . .
 ---> 8262920fb8e7
Step 6/8 : RUN npm run lint &&    npm run build
 ---> Running in 5e40f7e6b7c5

> angular-docker-ci@0.0.0 lint
> ng lint


Linting "angular-docker-ci"...

All files pass linting.
```

Fig. 4. Jenkins pipeline with each stage console output

In the above figure, the Jenkins pipeline stages are visible, and their results are evident in the console output. Utilizing this Jenkins stage enables the execution of essential commands automatically, resulting in considerable time savings.

Observations:

1. Early Issue Detection: The automated testing stage facilitated the early detection of issues, ranging from syntax errors to functional discrepancies. This aligns with the research's emphasis on finding problems early in the development process.

2. Developer Focus and Time Savings: With Jenkins handling Important tasks such as code compilation, testing and deployment, developers were able to allocate more time to designing and refining the application's features. This aligns with the research's observation that Jenkins automation frees up developers to create better applications.

3. Streamlined Processes: The seamless execution of each pipeline stage showcased the efficiency gains achieved through automation. Jenkins simplified complex tasks such as deployment, reducing the likelihood of manual errors.

IX. CONCLUSION

This research highlights the significant role of Jenkins automation in enhancing web application development. By integrating CI/CD, Jenkins simplifies processes and improves efficiency. A practical example using the Angular framework demonstrated how Jenkins handles important tasks effortlessly, showcasing the power of automation. It helps developers by catching problems early and making sure everything works smoothly. By automating these steps, developers can focus on making better applications and save time. Also, it simplifies tasks, improves efficiency, and ultimately empowers developers to create better applications. The results of this research have shown that using Jenkins to automate the development of web applications is a highly effective approach.

REFERENCES

[1] Fowler, M., & Foemmel, M, "Continuous Integration",2006,Martin Fowler.

[2] DuVall, P., Matyas, S., & Glover, A,"Continuous Integration in Agile Teams",2007,Addison-Wesley Professional.

[3] Farley, D., & Humble, J,"Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation",2010, Addison-Wesley Professional.

[4] Cawood, S., & Gerber, A,"Continuous Integration and Deployment for AngularJS and Node.js Applications",2015,Springer International Publishing.

[5] Joakim Verona, "Jenkins: The Definitive Guide. O'Reilly Media.",2011

[6] Monk, M."Jenkins 2.x Continuous Integration Cookbook",2016,Packt Publishing.

[7] Stellman, A., & Greene, J." Continuous Integration: Improving Software Quality and Reducing Risk",2014, Pearson Education.

[8] Smart, J "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.",2011, Addison-Wesley Professional.

[9] Humble, J., & Farley, D. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation",2010,Addison-Wesley Professional.

[10] Smart, J., & Ferguson, I."Jenkins 2: Up and Running.",2020, O'Reilly Media.

[11] Gruver, T., & Matyas, T."A Practical Guide to Jenkins: Get Started With Jenkins, Git, and Maven",2019, O'Reilly Media.

[12] Quick, R., & Davis, A."Jenkins X: Continuous Delivery with Kubernetes",2020,O'Reilly Media.

[13] Silvia Puglisi, Maria Teresa Vlachopoulou, Domenico Amalfitano, & Porfirio Tramontana. "Evaluation of Jenkins Plugins for Integration of Testing Tools in CI/CD Processes",2020, IEEE.

[14] Newman, S."Building Microservices: Designing Fine-Grained Systems",2017,O'Reilly Media.

[15] Chambers, D., Harris, I., & Köhler, S,"Jenkins, The Ultimate Guide",2020,Udemy.