Autonomous Continuous Integration and Continuous Deployment Pipeline

Sanjay Baitha
Department of Computer Science
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidypeetham, India
BL.EN.U4CSE21185@bl.students.amrita.edu

P. Radha Nishant
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
BL.EN.U4CSE21161@bl.students.amrita.edu

Vinay Nambiar
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
BL.EN.U4CSE21221@bl.students.amrita.edu

***Abstract*—In current world, CI/CD pipelines (Continuous Integration and Continuous Deployment) have proved better examples in the field of autonomous technology changing the understanding of software processes. This paper discuss about the future evolution of the continuous Integration/ continuous Delivery(CI/CD) pipeline, which takes the utilization Jenkins, and GitHub. Primarily using an integrated pipeline that performs flawless integration, computerized testing and deployment, the outcome pipeline can makes certain on the availability of new code versions and efficient roll-out of software updates. The details of what CI/CD principles, advantages, and key implementation equipment are enacted in the body of the content will help explain how automation leads to the improvement of quality, performance, and agility in the software development lifecycle. Implementing CI/CD pipelines as a release method lets agile development teams to quickly and agilely achieve the features they want, and to fix which errors there are, thus ensuring the rapid advance of the whole release process.**

***Index Terms*—CI/CD pipeline, automation, Jenkins, GitHub, sonarQube, software improvement**

I. Introduction

In a world where software is always growing CI/CD is essential to preserving workflow efficiency. This research is aimed at achieving an autonomous CI-CD pipeline that incorporates GitHub, Jenkins, and SonarQube. This project also seeks to improve the way in which codes are assessed for changes, facilitate smooth software delivery through testing building and code scanning. The aspect of the high complexity of the software development projects prompts the use of a complex CI/CD pipelines that can withstand the rapid iterations and constant alterations that the world is currently witnessing. In the past, this was achieved through manual processes that have been shown not to be effective and efficient; this results in delays as well as problems in software quality. An autonomous CI/CD pipeline solves these issues because it automates integration and deployment tasks, resulting in a decreased chance of errors, faster release cycles, and improved code or feature quality.

This research is being necessitated by the increased need for a credible software evaluation system that can be used to enhance efficiency. As for the fast development view, a business model that allows for integration and deployment of changes without interruptions is crucial for the competition. This attempts to set up an integrated pipeline that utilizes other tools such as GitHub for version control, Jenkins for automation, and SonarQube for static code analysis to ensure that every code or source code modification can be tested and analyzed carefully before deployment.

- **Travis CI**: Travis CI is one of the tools that contribute in the process of testing changes of code and releasing it for use. It works in conjunction with your repository on GitHub to test the code whenever a change is made. github is a browser-based service incorporated within or alongside a user's repositories. There are, of course, some disadvantages such as having limited access to only certain GitHub repositories, some amount of delays if the base is shared and fewer possibilities to change settings of the service as compared to self-hosted ones.
- **CircleCI**:CircleCI is a cloud-native helper tool in control systems such as GitHub and Bitbucket. It enables the developers test their code, build, and deploy applications remotely. However, the significant limitations are high costs for larger teams or projects compared to other tools, decreased configurability and flexibility in comparison with Jenkins, and difficulties to use it without resorting to cloud services with all the privacy issues that have arisen because of it.

In order to tackle this problems, this research is conducted which contributes:

- Seamless Integration: As a result of this pipeline, every code change is 'Tested' 'Built' and 'Analysed' through the use of GitHub, Jenkins and SonarQube respectively. This way of integration helps in identifying code defects at an early stage and ensures code integrity at a high

level.

- Customization and Flexibility: Jenkins is a powerful, open-source automation server that can be easily customized and scaled. This flexibility is crucial as it allows the pipeline to meet projects' requirements and workflows.
- Enhanced Code Quality: SonarQube is an excellent and popular tool of choice in this aspect as it provides analysis of the code quality, looking for potential bugs, security vulnerabilities, and code maintainability. This is crucial since it ensures that code stays healthy and at a high quality.

This research divided into multiple sections: Section 1 describes brief introduction on Autonomous CI/CD pipeline. Section II presents the Literature survey conducted for this paper. Section III describes the Workflow whereas section IV describes the Results generated. Section V presents the conclusion and possible future work.

II. Literature Review

With the ever-increasing need for software updates and the development geared towards increasing efficiency in the entire process, automating the pipeline for continuous integration and deployment has become essential for CI/CD applications. The current body of work about pipeline automation in the context of CI/CD is critical in this literature review: The type of techniques that are reported as well as the tools that are involved as well as the impact on the development lifecycle. The survey papers refer to different tools in the CI/CD pipelines such as SonarQube, Jenkins, GitHub,Docker, and Kubernetes and their roles as well as advantages in this process.

K. Gallaba et al. [1] Advocate use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. The research paper explores the utilization and misutilization of CI features in TRAVIS CI configurations across a substantial corpus of open-source projects. It identifies a notable concentration of configuration code in job processing nodes, indicating underutilization of CI tools for continuous delivery purposes. The prevalent use of CI features, providing insights into the distribution of CI configuration code among different sections and identifying common anti-patterns in Travis CI specifications. While the study effectively analyzes feature usage and misuse in Travis CI specifications, there remains a gap in understanding the reasons behind the observed patterns and anti-patterns.

M. Shahin et al. [2] Advocate use and Misuse of Continuous Integration Features: Factors that Predicts the Success or Failure of Projects Using Travis CI: An Empirical Study. This research paper delves into the application and misapplication of CI features in various TRAVIS CI examples from a large set of open-source repositories. It finds several major clusters that contain configuration code for job processing nodes and suggests that CI tools are used inefficiently for continuous delivery. : The widespread applicability of CI features; understanding the distribution of CI configuration code across different sections and revealing common anti-patterns involved in Travis CI specifications. Although the study is quite well done in terms of micro-feature analysis and definition, the absence of macro-feature examination and explanation of why specific patterns of feature use and misuse are observed in Travis CI specifications is a certain problem.

Jose L et al. [3]Advocate to CI and CD: Recent Advances of the Application of Information Theory in Software Systems Development: A Systematic Literature Review of Approaches, Tools, Challenges and Practices. The main objective of the research that was done by Shahin et al. (2017) was to analyze the practices on Continuous Integration, Delivery and Deployment (CIDD) in the software engineering field using what is known as a Systematic Literature Review (SLR). It also highlighted some of the challenges that faced this study including those outlined with concern to the methods used in analyzing the data where some articles did not by any chance outline where and how the data was analyzed. The prevalence of the literature on continuous practices while the larger proportions of the reviewed papers were academic in scope and conceptual in nature, there is a need for more publications on industry based research in this area.

I. Karabey et al. [4] Offering Systematic Strategy to Design Feasible Deployment Options for Microservices. The aim of the research is to define a mechanism that is capable of producing viable deployment options for microservices especially in situations when demand is limited and resources are scarce. It gives deployment options to processes and development teams early on for identifying and fixing issues in the design at the earliest stage of development. This is means there is efficiency and effectiveness in the assembly models for deployment. The lack of evidence-based information in existing literature about the detailed implementation of feasible microservices to cloud resources is an indication that such information is still paramount. It also does not deal with concerns of business such as scaling of microservices.

A. F. Nogueira et al. [5] Recommend increasing the productivity of the La Redoute's CI/CD pipeline and DevOps practices by implementing some of the Machine Learning techniques. The aim of the given research is to enhance the effectiveness of the CI/CD pipeline and DevOps practices at La Redoute on the basis of machine learning. It supports the ability to monitor automatically and is able to identify errors and defects faster. This helps in faster implementation of feedback loops making it easier for an organization to execute the decision-making and problem-solving capabilities and evaluate historical data and machine learning models. While the concept of DevOps and the use of CI/CD pipelines are being adopted in the industry, there are currently not many research works on the efficient integration of machine learning on the optimization of CI/CD pipelines.

Golzadeh et al. [6] predicts the cyclical nature of continuous integration services in github. The purpose of this research is to identify the usage rates and CI services offered by various companies in the market today. In the paper, 91,810 Github repositories of energetic NPM applications having used at the

least one CI provider had been analyzed and the evolution of 7 famous CIs, particularly that according to their co-usage and migration had been quantitatively studied inside the taken into consideration repositories. This particular article is geared towards the reader's understanding of the CI Services and does not deal with the readers all about the CI/CD workflow and tools that can be used to achieve CI/CD as a pipeline.

Arachchi et al. [7] Recommend for the CI and CD Pipeline to automate the agile Software Program Assignment. The main objectives of this study are to combine CI/CD pipeline with Agile Software Project Management and explore the testing method in CI/CD pipeline. The article is dedicated to the topic of implementing CI/CD pipeline with Agile is talking about testing before software can be delivered to customers. It is more about Agile and testing, dedicated for not many tools and not explaining all the parts of the CI/CD pipeline so it is not that good choice probably.

Mahboob et al. [8] Recommend Kubernetes CI/CD pipelines with as ylo as a security execution environment abstraction. The purpose of this work is to integrate, engineer and deploy a CI/CD pipeline using Kubernetes via GCP. The following paper also describes the deployment of CI pipeline using Kubernetes and also describes the deployment using Google Cloud Platform. It also addresses the shared responsibilities of Development and Operations and also automates the PipeLine. The limitation is that the paper does not speak to the implementation between other cloud service providers and only addresses Kubernetes as a tool. It also does not define how it would work for migration to another cloud service.

Charanjot et al. [9] Recommend that CI/CD equipment provided with cloud platform includes comparison with other products. The aim of this study is to assess some aspects like performance monitoring after deployments, pipeline integration, cloud compatibility and server monitoring in order to determine whether the tools are effective to automate deployment processes. They automate certain deployment processes further decreasing manual work and less system downtime. Continuous integration/Continuous deployment or delivery pipelines enhance the levels of scalability and reliability by allowing for faster upgrades and deployments of microservices. The strengths of this study in assessing CI/CD solutions for delivering microservices are complete but there is an area for further investigation in assessing tools' scalability and flexibility to different project needs.

Rizky et al. [10] Suggest use of CI/CD on Computerized General Performance Testing. The intention of this work is to integrate the CI and the CD practices into performance testing methodologies in the direction of automating and simplifying testing procedures. It facilitates the practical and frequent checking of performance runs without the need for manual intervention and effort/time as well as aligns it with the CI/CD pipeline. Although the CI/CD methodology has shown its usefulness in a variety of aspects of software development, there is still a lack of translation into performance testing. While existing literature focuses on the automation testing and the end to end testing process.

J. Mahboob et. al. [8] explains how an application development and deployment process went from traditional monolithic systems and why this was a change with paradigm through the use of CI/CD to obtain high levels of agility and scalability or Cloud time. It spends time defining microservice architecture advantages, for example: the modularity and expansibility that is favorable for comparison with monolithic solutions. Intake also considers the role of virtualization technologies such as virtual machines and Docker regarding virtualization with special attention given to Docker and the implementation of microservices. It also reflects on the CI/CD practices – not only on the automation and continuous deployment but also fast and efficient convergence of the software. It also contains Jenkins and Gitlab and the way it is possible to integrate them with version control systems such as Git.

K. Gallaba et. al. [1] is an explanation of the security aspect that has been reinforced in the current software development and delivery bring about the rise of DevSecOps. SecDevOps incorporates security solutions into these stages; sometimes as the primary approach in the form of automation. The pipeline will be a CI/CD pipeline that stands for continuous integration and continuous deployment – a series of automated processes that ensure correct code delivery. Think again, the probability of runtime security in solving tangled problems is very low especially in the case of third-party edition when the syndrome of scareware is guarding the encrypted. Kubernetes is used to manage resources within the CI/CD process and privacy hardware like Intel SGX are considered for the binaries to prevent insider attacks. It also provided information about various steps like Kubernetes pipeline architecture, artifact confidentiality in Asylo which it also thoroughly tested.

P. A .G. Permana et al. [11] suggested the use of Jenkins for automation deployment where we will discuss the implementation of scheduling and notifications to improve workflow in software development. It consists of stages such as checkout, build, test, and deploy. They used Jenkins which is Core CI/CD tool for automation, Docker that is used for containerization of applications; Version Control used Git or SVN for source code management and Notification Services Integration with messaging platforms like Telegram as well as WhatsApp . Build, test and deployment automation eliminates the heavy reliance on human interactions and therefore supports faster development cycles. Even though each of these disadvantages can easily become a problem in software development, the time-saving features and quick identification of errors associated with the approach make it a worthwhile option for modern software development.

A. Cepuc et al. [12] provide an example of Continuous Integration and Deployment (CI/CD) pipeline for containerized applications through the use of Jenkins, Ansible, and Kubernetes within the AWS environment. The model used here involves Jenkins for Continuous Integration that will check for any changes in the source code of the Java – based web application and then build the code while the continuous deployment is done by Ansible for the creation of the Image and for the deployment in a Kubernetes cluster in the AWS

environment. This approach guarantees no service disruption and makes heavy usage of Jenkins and Ansible effectively. But the use of the multiple tools increases complexity to the extent that it demands extreme levels of configuration and resource management to keep the system running smoothly.

A. H. Rakshitha et al. [13] published a paper outlining the automation of software configuration and deployment based on Jenkins and the Yocto project. The model used includes Jenkins for CI and CD and shell Scripting for Linux based operations command while Yocto Project offers the tools to create custom Linux based systems for embedded systems. This improves efficiency and eliminates human errors by incorporating computerization of mundane activities and offering other strong build tools like Bitbake for creating executable applications. But using more multi-purpose tools and their customization in many settings could be complicated and require much setup time and maintenance.

A. Deshpande et al. [14]As discussed in this paper, the current application of test-based automation and continuous integration to enhance test coverage and speed up execution of manual testing has important benefits to software testing by eliminating human effort and mitigating errors resulting from manual testing. The model used is Jenkins master-slave architecture with the support of various tools such as Apache Subversion for version control, Visual Studios, and PowerShell for command-line scripting. One huge benefit of this approach is that it is able to support complex pipelines to facilitate continuous integration and delivery in software development processes. But there is a major negative aspect – Jenkins' plugin's complexity and redundancy, which is hard to navigate.

A literature review of published research papers on the domain of software engineering and continuous integration/continuous deployment (CI/CD) pipelines reveals that most projects focus on a few tools and do not provide detailed descriptions of automated processes. This lack of detail can make it difficult others to reproduce their findings and therefore practically apply it. The main objective of this work is to address the mentioned problem by adopting a well-structured and fully automated CI/CD process based on Jenkins for building, GitHub for the source code management and SonarQube as the code quality analyzing tool. This way not only takes care of the deficiencies pointed out regarding previous studies but also has a framework that can be implemented within the industries. Another distinctive feature of our approach is the applicability of the discussed models to those common analysis tools. As these plugins use the Jenkins API; Jenkins can be customized to meet different project needs and conditions. This is well in line with the flexibility employed in integrating different development tools and cloud services as pointed out in the review of literature above to address the limitations. In addition, the integrated use of the Jenkins, GitHub, and SonarQube tools for CI/CD can be easily used again by other development teams.

III. Methodology

A. GitHub

GitHub is an online web portal as well as a distributed version control system (DVCS) that is utilized to handle and monitor the changes made to the files by the various users on a project. GitHub can function as an online file sharing system for the collection, management, and change of the code created by developers. Amongst others GitHub offers a rich set of features such as pull requests for a developer to accept pull requests on modifications to code base and issues that are used to track and assign bugs, feature requests, and tasks within a repository. Furthermore, GitHub provides support of tools and services on a mass scale, which makes it the premier platform for software development teams.

B. Jenkins

Jenkins is one of the widely used open source automation server is used for building, testing and deploying the software project. It offers a way to facilitate CI/CD, as it allows software development to be automated for repetitive tasks. Jenkins is said to be very extensible with a huge environment of plug-ins that support current status to be modified as and when needed by the users. The main features of Jenkins include the fact that it displays the ability to work in GIT to integrate and trigger automatically or build every time changes are being made in a repository of the developer. Successful automation also moves to developmental process where the code is tested and deployed instantly.

The question of a specific technology stack is not critical due to the wide range of build and test tools that is supported by Jenkins. It gives a web-based user interface for the users to see the current state of their builds, the logs showing their build status, and reports indicating the test results from time to time. Jenkins also has the ability of sending email, messages to team members handling the project, so everyone gets the status of the project.

C. SonarQube

SonarQube is an open-source tool to monitor the code quality as part of continuous integration. It is the static code analysis tool used to check code for bugs and errors, code smells, and security vulnerabilities in codebases to help the developers in enhancing the quality and quality of code. SonarQube is supported by a wide range of programming languages like Java, Javascript, C#, python, etc for a comprehensive analysis.

The main advantage when developing with SonarQube is analysis and reporting on code quality metrics. Some of the reports that include information on code duplication; complexity; unit test coverage and coding standards to name but a few Fig. This information can help developers understand where changes in the metrics should be made in order to enhance the quality of their products.

SonarQube supports much of the CI/CD tools like Jenkins so that sonar analysis can be triggered/aper as a part of build process. This is a great resource for the developer and allows

them to have some feedback on the quality of their code and work to correct and resolve any issues that could lead to errors. Secondly, SonarQube is very easy to use and easy to integrate with developers' projects as the platform makes it easy for developers to visualize their code quality trends over time for the purpose of tracking the code quality improvement.

D. Docker

Docker is the Docker industry is a suite of software tool using for containerization is a packaging method of an application into a standardized unit for software development. Docker containers are small, and self-sufficient, such that they include only the application code, library, settings, and system tools – in other words, a perfect little artificial ecosystem. This can be said to cut across all physical boundaries of the application to allow for smooth processing and deployment from development to production with minimal changes in the infrastructure used.

Docker provides a number of advantages and has made the process of development and deployment significantly easier. It is easily to develop and run containers; therefore, the developers will less focus on dependencies and environments that it will reduce the deployment time directly. Docker is also helpful in making it possible for different developers to work together since the Docker environment can be copied across several computers while preserving an identical environment for testers and developers to work on the application.

We thus set out a sequence of steps for the implementation of our CI/CD pipeline which relies on integration of GitHub, Jenkins, and SonarQube by starting with the creation of a version control system. GitHub can be considered a crucial link in the chain since it will host source code for the project. The creation of a new GitHub repository allows the team to create a central location for this component and achieve better improvement and collaboration results.

E. Workflow

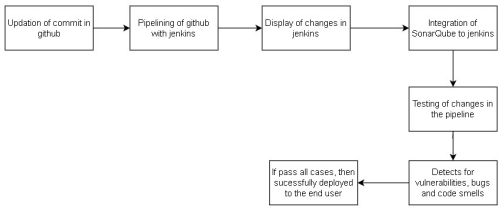

Fig. 1. Proposed Architecture

The fig. 1 explains the process done step-by-step. With the set out a sequence of steps for the implementation of our CI/CD pipeline which relies on integration of GitHub, Jenkins, and SonarQube by starting with the creation of a version control system. GitHub can be considered a crucial link in the chain since it will host source code for the project. The creation of a new GitHub repository allows the team to create a central location for this component and achieve better improvement and collaboration results.

The next important step in this process will be to establish HTTPS connection between our Jenkins server and the GitHub repository. This is accomplished by using SSH keys that are used for the authentication process and that ensures a trusted connection is created between the two systems. Use of SSH keys means that you can only use the Jenkins server with authorized access codes which improves the security of your pipeline.

First authentication keys are exchanged between the machines via SSH after which JDK 111 is installed since Jenkins is a Java-based application. JDK 11 provides the required libraries and utilities to perform Jenkins operation efficiently and effectively. This is necessary to allow the Jenkins server to run correctly and fulfill the needs of our CI/CD pipeline.

After completing the JDK 11 installation, we can go ahead and get in the installation process of Jenkins itself. This includes downloading the gpg key for Jenkins, adding the Jenkins repository to our system, updating the package index, and finally installing Jenkins using the apt-get package manager. Jenkins is setup by installing it and then confirming that it is running properly using the status command systemctl. The installation in ubuntu invovles:

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" https://pkg.jenkins.io/debian-stable binary/ — sudo tee /etc/apt/sources.list.d/jenkins.list ¿ /dev/null sudo apt-get update sudo apt-get install jenkins

After we succeed to start Jenkins server, we establish configuration to our existing GitHub repository. This configuration entails creating a new item under Jenkins and choosing the freestyle project as the type of proj8ct and then entering the URL for the GitHub repository in the Source Code Management option. This way Jenkins can automatically build our project every time a member in our team makes any changes to the project in the GitHub repository.

We then create a new webhook in the repository settings to notify Jenkins of any changes within the GitHub repository. In the payload URL and path of webhook configuration we specify the Jenkins server localhost link. The information in this webhook setup means that Jenkins is notified when there is a change made to our GitHub repository and can kick off the required build or deployment.

Next we need to update the hostname using hostnamectl set-hostname in SonarQube, a static code analysis tool. This step is to confirm that SonarQube is configured properly to the communication between our system. We then install the prerequisite Java JDK 11 since SonarQube requires Java 11 to operate.

Before we begin testing JDK 11, we download the SonarQube Community Edition from the official website. After successful download, we extract the contents of this zip file to a folder and go to bin folder inside it. From there, we begin SonarQube through the . /sonar. sh console command, which

allows us to check if SonarQube is working correctly before accessing it through the localhost:by typing 9000 in a web browser address.

Here we proceed with the Jenkins configuration to be set for the SonarQube server by adding to Jenkins the SonarQube URL and the global analysis token provided by the global analysis in the admin section of the SonarQube server. We can also add SonarQube scanner in the tools section of Jenkins and configure the SonarQube server in Jenkins to use this scanner to scan the code.

Last but not least, we create a Jenkins pipeline that utilizes GitHub, Jenkins, and SonarQube. This pipeline reduces the need to manually test, build, and analyze our code, giving us valuable input information on the quality of our code. If these implementation steps are followed we can develop a highly efficient CI/CD pipeline that will complement the existing SDLC process and improve software development efficient by delivering high quality software in a time manner.

Here the last installation step is performed using the systemctl command and Jenkins status can be checked to see whether the server is running. With a browser visit Jenkins You should enter administrator password from installation and finish the first configuration process, including creating a new password and registration.

To navigate the Jenkins menu and complete its configuration, the user needs to select “New Item” and “Freestyle project” and enter a name for the new project. The project description is set to the source code management system ”Git” as well as the GitHub project URL is specified. This enables Jenkins to connect to the GitHub repository and act as a pull that launches automated builds.

For Jenkins to access GitHub, a webhook is created in the below screen from the GitHub repository settings. Jenkins webhook fetches status about the change of repository and automatical builds/deployments. Configuring the webhook involves setting the Payload URL to http:The configuration that needs to be modified was your_jenkins_server github-webhook.

Lastly to try out the setup when a build is triggered in Jenkins use the click “Build Now” in the project tab. Jenkins fetches the code from the GitHub repository and test the build logs and displays the build logs to validate that the CI/CD pipeline is working fine. This integration makes the development process easier as the development is concentrated on writing the code and the Jenkins will takes care of building compiling testing and even deploying the code.

IV. Experimental Analysis and Result Discussion

After following the steps described in the previous section, the following results are obtained:

The overall structure in fig. 2 displayed above explains how CI/CD process works through a smooth and efficient integration into the operation of an organization. Which will consume all the stages including the code pull, building, and verification process, to make sure that if there is any changes in any part of code, it will be tested before being released

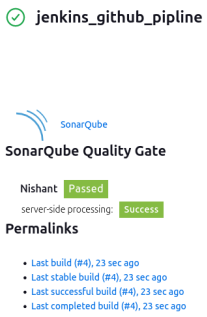

Fig. 2. Pipeline Output

to consumers. This automation requires minimal human involvement and less scope for human errors and fastens the process of delivering the software. Fig. 1 provides insight that I aimed to demonstrate in this paper – GitHub, Jenkins and SonarQube integration to some extent works as expected and is rather powerful. It demonstrates the way these tools interact with each other to deliver robust Iterative Automated Build & Deployment. This integration does not only ensure that the developers work effectively and efficiently, but also that the software being developed meets certain standards of quality and security.

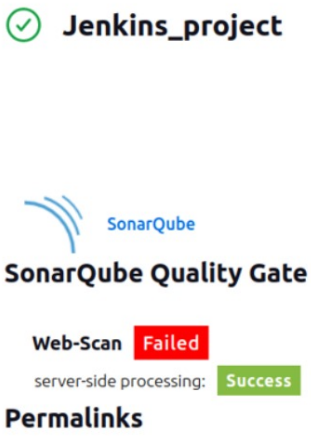

Fig. 3. Failed Pipeline Output

Fig. 3 illustrates how the CI/CD pipeline approach was invaluable for avoiding mass deployment of faulty code. The results of the Web Scan as a result of the introduction of an infinite loop into the code are shown in the form of the pipeline output where the Web Scan was marked as failure due to the fact that an infinite loop condition was detected. This failsafe feature ensures that the developers do not release the updates having defects in the codes to production and this may end up in the application crashing. Using test automation and code scanning in CI/CD leverage the advantage of system and its ability to detect defects as part of the release process. It not only improves the quality of software but also reduces the time and effort for getting from one phase to next one.

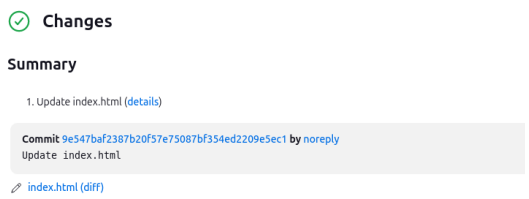

Fig. 4. View Changes

Referring to fig. 4, the integration of GitHub and Jenkins can be seen as a success accompanied by the ability to effectively pipeline the GitHub through the Jenkins machine. As shown in the figure below, the Jenkins dashboard can be filled in a different way after the commit pushed to the GitHub repository. This is especially when it comes to highlighting the change of index. original html file which is found in the source code for the project. This CI/CD pipeline practises both in GitHub and Jenkins is the most effective use of the approach. Jenkins successfully found that the index had been changed. Git push which initiated the automatic build and deployment process to an HTML file created.

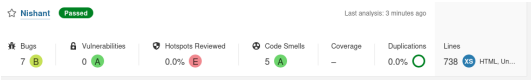

Fig. 5. Sonar Results

fig. 5 depicts the analysis that sonarqube did to the project including the code analysis Summary of 5 fig. SonarQube is a static code analysis tool that is used to look for all sorts of quality problems in code like bugs, complex code fragments and potential security vulnerabilities. These elements are critical for demonstrating how code will be health and maintainable. The following figure shows the dashboard that serves as the platform for SonarQube to report its findings. It indicates the total number of bugs that were present or not in the code – bugs are defects in the code that may cause the program to run erratically or crash. These include conditions that pose a threat to the system that may be countermanded by malevolent sources.

V. Conclusion

The development of an autonomous CI/CD pipeline using GitHub/ Jenkins/ SonarQube for distributed systems represents a significant development in distributed systems. This project has demonstrated how automation can be used in important processes such as testing, building and code analysis for a software. The use of such tools gives developers the confidence that every release will be of high quality without wasting too much time correcting manual errors, thus increasing the rate at which releases are made. The combination of these three software has created a strong environment for the CI/CD process through the toolset for tracking changes, identifying problems, and enforcing quality. The use of Docker containers has gone a long way in enhancing the pipeline's portability and scalability with ease of deployment in varying environments. Future research areas include Testing activities such as integration and performance testing might also improve the software reliability and quality.

References

[1] K. Gallaba and S. McIntosh, "Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci," *IEEE Transactions on Software Engineering*, vol. 46, no. 1, pp. 33–50, 2018.

[2] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[3] J. L. B. Justo, N. M. Araujo, and A. G. Garcia, "Software reuse and continuous software development: A systematic mapping study," *IEEE Latin America Transactions*, vol. 16, no. 5, pp. 1539–1546, 2018.

[4] I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, "Systematic approach for generation of feasible deployment alternatives for microservices," *IEEE Access*, vol. 9, pp. 29505–29529, 2021.

[5] A. F. Nogueira, J. C. Ribeiro, M. A. Zenha-Rela, and A. Craske, "Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques," in *2018 11th international conference on the quality of information and communications technology (QUATIC)*, pp. 282–286, IEEE, 2018.

[6] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of ci services in github," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 662–672, IEEE, 2022.

[7] S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, pp. 156–161, IEEE, 2018.

[8] J. Mahboob and J. Coffman, "A kubernetes ci/cd pipeline with asylo as a trusted execution environment abstraction framework," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0529–0535, IEEE, 2021.

[9] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different ci/cd tools integrated with cloud platform," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 7–12, IEEE, 2019.

[10] M. R. Pratama and D. S. Kusumo, "Implementation of continuous integration and continuous delivery (ci/cd) on automatic performance testing," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pp. 230–235, IEEE, 2021.

[11] P. A. G. Permana, E. Triandini, and N. L. G. P. Suwirmayanti, "Implementation jenkins automation deployment with scheduler and notification," in *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)*, pp. 1–5, IEEE, 2021.

[12] A. Cepuc, R. Botez, O. Craciun, I.-A. Ivanciu, and V. Dobrota, "Implementation of a continuous integration and deployment pipeline for containerized applications in amazon web services using jenkins, ansible and kubernetes," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–6, IEEE, 2020.

[13] A. Rakshitha, P. Jayanthi, and S. Manyam, "Software configuration using jenkins and yocto project," in *2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, pp. 1–4, IEEE, 2023.

[14] A. Deshpande, S. Veenadevi, and S. Aleti, "Test automation and continuous integration using jenkins for smart card os," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 01–05, IEEE, 2021.