International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 07 | July - 2023 SJIF Rating: 8.176 ISSN: 2582-3930

CODEFLOW AUTOMATION ENGINE

PROF. SURESH NAGPURE[1], ADITYA BHUTMARE[2], SUDHANSHU DHOKE[3], SHREYASH KULKARNI[4], VARUN DOLE[5]

[1] Assistant Professor, in Department of Cloud Technology at Rashtrasant Tukdoji Maharaj Nagpur University, NAGPUR, MAHARASHTRA, INDIA-440033

[2,3,4,5]Students, in Department of Cloud Technology at Rashtrasant Tukdoji Maharaj Nagpur University, NAGPUR, MAHARASHTRA, INDIA-440033

ABSTRACT:

A CI CD pipeline, in essence, specifies the steps a developer should follow to release a new version of a software product. The builder would yet have to if the pipeline is not automated, execute the same operations manually, which is far less effective. Therefore, the majority of software releases go via the processes outlined below. We provide an overview of the automated deployment of a new version of an application using a safe SDLC process, continuous integration, and continuous deployment. In doing so, code quality and software security are ensured while enabling software development teams to concentrate on fulfilling business needs.

INTRODUCTION:

Every business strives to shorten the time it takes for its goods to reach the market in order to provide better customer service to its staff. As a result, we launched our application using the appropriate technology and a DevOps strategy. By lessening the demand on your servers, DevOps can help lower the cost of your server environment (Amazon Web Services). This speeds up server responses and automates procedures. Add resources to a virtual computer as an example. Our strategy's main objective is automation of the development and deployment processes in order to boost organizational efficiency, lower downtime, and accelerate deployment. A software delivery technique called deployment automation enables businesses to roll out new features more quickly and often. The manual deployment procedure is simple. Human error could occur during this process. Pre-production mistakes have a direct correlation to production mistakes, and manual methods are more prone to these mistakes. He uses GitHub Actions as his CI/CD platform as a result to automate the deployment of applications. CI/CD is a method for frequently delivering apps to clients by automating the various app development steps. The three primary CI/CD ideas are continuous delivery, continuous deployment, and continuous integration. Particularly, CI/CD introduces continuous monitoring and ongoing automation throughout the entirety of an app's lifecycle, from the integration and testing phases through the delivery and deployment stages.

LITERATURE REVIEW:

1] Meghamala Ramidi, Sai Priya Sinde, and Bhavika Thakkalapally The goal of this investigation was to learn more about how programmers anticipate using GitHub actions to automate work processes and how activity reception affects pull requests. They gathered and analyzed data from 3,190 active GitHub repositories and discovered that just a small percentage of them used GitHub actions, along with 708 unusual predefined actions that were being used in work processes. Additionally, they compiled and examined GitHub action-related issues, discovering that the majority of the comments were supportive. The findings generally indicate that designers largely approved of GitHub's conduct. In addition to a review of security research in the area of cloud security, this article provides the idea of on-request advantages, which alludes to the utilization of cloud assets on request and the ability to scale assets as per request. Utilizing GitHub actions allowed us to launch an application quickly and with the least amount of tools possible. The flow of deployment is continuous and error-free when using GitHub activities as a CI/CD pipeline with an AWS EKS cluster. Jenkins as a CI/CD pipeline takes about 1 minute and 43 seconds to deploy an application.[1]

2] Robert Botez, Ovidiu Craciun, and Artur Cepuc One of the most pervasive ideas in the IT sector is cloud computing, which has developed over time. Instead of installing a full infrastructure, which would incur additional costs like buying and maintaining the equipment, paying the workers, etc., it succeeds by providing on-demand services. Broad network access, quick elasticity, measurable service, on-demand self-service, and resource pooling are the qualities that the National Institute of Standards and Technology (NIST) uses to categorise services as cloud services or not. We were able to improve overall scalability and send commands directly to the Kubernetes cluster because of this. The results of the experiments demonstrated that the proposed solution is fast, scalable, and reliable, with no downtime. As a result, in just 37.6 seconds, any change to the application's source code is automatically detected and sets off a whole series of six different technologies. The system rolls back the most recent stable version whenever a Jenkins job fails to deploy a new version of the application.[2]

Leonardo, et al. [3] Explain the results of the recent poll and the issues with DevOps from the perspectives of Leite's engineers, managers, and researchers. We conduct a review of the literature and develop a conceptual diagram of DevOps, linking its guiding principles to its automation tools. We then discuss their practical implications for scientists, managers, and engineers. Finally, we critically review some of the most significant DevOps challenges addressed in the literature. In this study, we have discussed the DevOps tenets and problems that have been brought up in the literature. We assisted practitioners in choosing the right toolset by relating these concepts to particular technology.

IT workers benefit from the logical presentation of the most crucial concepts, materials, and outcomes from the perspectives of academics, managers, and engineers—including developers and operators. We believe that after reading each profile, our reader may better understand how DevOps impacts routine chores. The two pillars of DevOps are automation and cross-functional teamwork. The main objective of this project is to automate the process of building a project in Maragathavalli [4], which includes the tedious processes of writing code, testing it, and project deployment. to provide a method for automating, tracking, and addressing application issues. 2) A continuous pipeline process of development, testing, and deployment is utilized to

produce a system. Every update necessitates a different build because the organization will be altered numerous times each day.

MOTIVATION:

The goal of this project is to create a pipeline for CI/CD, or continuous integration and continuous delivery, which is a methodology for driving software development. The objective of automating the procedure is to reduce human error and preserve a standardized software release process. Code compilation, unit testing, code analysis, and security are some of the tools in the pipeline. The code is packaged into a container image for containerized environments as well as deployed on a public cloud. To tackle the difficulties brought on by more sophisticated software systems and the demand for quicker, more effective software development. The project intends to automate and streamline the software development lifecycle by putting in place a CI/CD pipeline, enabling frequent releases, easy collaboration, and continuous improvement. The automated building, testing, code analysis, and deployment made possible by the integration of tools like Jenkins, GitHub, SonarQube, and Docker leads to better code quality, quicker feedback loops, and effective resource use. Enhancing efficiency, code consistency, and the capacity to provide high-quality software with shorter time-to-market are the driving forces behind these initiatives, which are intended to help organizations stay competitive in the rapidly evolving software sector.

OBJECTIVE:

I. Automation: The process of automating software development and deployment is the main goal. The project seeks to automate operations including building, testing, code analysis, and deployment through the use of a CI/CD pipeline, thereby lowering manual labour requirements, reducing error rates, and quickening the software delivery cycle.

II. Collaboration and version control: The project uses version control tools like GitHub to improve developer collaboration. It is intended to facilitate seamless code collaboration, versioning, and simple management of code changes by integrating GitHub into the CI/CD pipeline, assuring a centralized and regulated development environment.

III. Code Quality Improvement: By integrating SonarQube, the project aims to raise the general level of code quality. To identify and fix code smells, bugs, vulnerabilities, and other quality issues early in the development process, SonarQube's code analysis tools will be used. The project seeks to improve the software's dependability, security, and maintainability by upholding high standards for code quality.

IV. Portability and Containerization: The goal is to use Docker to containerize pipeline-based services and applications. The project attempts to ensure consistency, portability, and scalability across many settings, such as development, testing, and production, by containerizing the programme. This enables simpler deployment, effective resource management, and seamless migration between different infrastructure configurations.

PROBLEM STATEMENT:

Traditional software development methods include a number of issues and restrictions that make it difficult to produce high-quality software quickly. The lack of collaboration, variable code quality, manual and error-prone methods, and lengthy deployment cycles are some of these difficulties. These problems lead to a longer time to market, lower productivity, and a higher risk of software flaws.

The issue is that organizations require a comprehensive and automated CI/CD pipeline that takes these issues into account and makes it possible for them to optimize their software development and deployment procedures. The software development lifecycle suffers from inefficiencies and a lack of agility as a result of the existing approaches' frequent lack of integration, automation, and scalability.

By developing a thorough CI/CD pipeline that incorporates tools like Jenkins, GitHub, SonarQube, and Docker, the project seeks to solve these issues. The build, test, code analysis, and deployment processes are all automated by the pipeline, resulting in quicker and more dependable software releases. It improves developer cooperation, offers code quality analysis, and makes ensuring the development environment is consistent.

The goal of the project is to solve these issues in order to increase productivity, code quality, and the effectiveness of the software development process as a whole. In order to construct a simplified and automated CI/CD pipeline that enables organizations to deliver high-quality software more effectively and efficiently, it seeks to overcome the limits of conventional software development methodologies.

PROPOSED APPROACH:

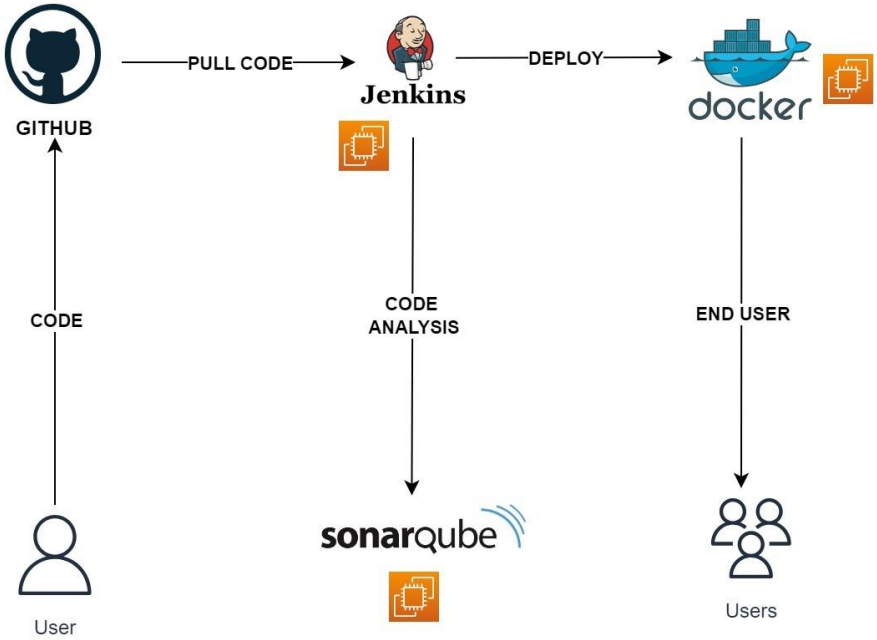

FIG NO. 1 PROPOSED APPROACH

CONFIGURATION OF PROJECT:

1. Jenkins Configuration:

   a. Launch an EC2 instance with an appropriate Amazon Machine Image (AMI) that supports Jenkins.

   b. Connect to the EC2 instance using SSH and install Jenkins by following the official Jenkins installation instructions for Linux.

   c. Set up the Jenkins server by accessing the Jenkins web interface and completing the initial setup wizard.

   d. Install and configure any necessary Jenkins plugins for Git, SonarQube, Docker, and other required integrations.

   e. Configure security settings, including user authentication, authorization, and access controls within Jenkins.

2. GitHub Configuration:
   a. Create a new GitHub repository for your project or use an existing repository.
   b. Generate SSH keys on the Jenkins EC2 instance and add the public key to the GitHub repository settings for secure communication.
   c. Set up a webhook in the GitHub repository to trigger Jenkins builds upon code changes.
   d. Configure branch protection rules in GitHub to enforce code quality checks and ensure secure code commits.
3. SonarQube Configuration:
   a. Launch a separate EC2 instance to host the SonarQube server.
   b. Install and configure SonarQube on the EC2 instance by following the official SonarQube installation guide for AWS.
   c. Set up the necessary security groups and firewall rules to allow communication between the Jenkins and SonarQube instances.
   d. Configure SonarQube server settings, including database connection, security configurations, and quality profiles.
4. Docker Configuration:
   a. Launch additional EC2 instances or use the existing Jenkins EC2 instance to host Docker.
   b. Install Docker on the EC2 instance(s) by following the official Docker installation guide for Amazon Linux or the chosen operating system.
   c. Set up Docker daemon configurations, including network settings, resource limitations, and security options.
   d. Create Docker images for the application or services within the CI/CD pipeline and push them to a Docker registry or repository.
5. Pipeline Configuration:
   a. Define the stages and steps of the CI/CD pipeline in a Jenkins file or through Jenkins pipeline configuration.

b. Configure build triggers, such as Git webhooks, to initiate pipeline execution upon code changes.

c. Integrate SonarQube into the pipeline to perform code analysis and quality checks.

d. Define deployment configurations for deploying the application or services using Docker containers on the AWS EC2 instances.

6. AWS EC2 Instance Configuration:

a. Set up and configure AWS EC2 instances to host the application or services.

b. Configure security groups and network settings to control inbound and outbound traffic to the EC2 instances.

c. Ensure that the EC2 instances have the necessary IAM roles and permissions to interact with other AWS services, such as S3 or RDS.

7. Testing and Validation:

a. Test the CI/CD pipeline by triggering builds manually or automatically through code commits.

b. Validate the successful execution of each stage in the pipeline, including code analysis,testing, and deployment.

c. Monitor the deployed application or services on the AWS EC2 instances to ensure proper functioning in the target environment.

RESULT:

The result of the challenge are as follows:
Enhanced Code Quality: The integration of SonarQube and code analysis tools enables continuous code quality checks, identifying code issues, bugs, and vulnerabilities early in the development cycle. This leads to improved code quality, reduced technical debt, and increased overall software requirement.

Increased Collaboration and Version Control: The integration with GitHub enables seamless code collaboration, version control, and parallel development. Developers can work together efficiently, merging code changes, and managing code versions effectively.

Scalability and Portability: By leveraging Docker and Kubernetes, the pipeline enables scalability and portability of the application or services. Containers provide a consistent and isolated environment, making it easier to scale the application and deploy it across different environments.

Streamlined Deployment Process: The CI/CD pipeline automates the deployment process, ensuring consistent and reliable deployments across different environments. This reduces manual errors and ensures that the application is deployed in a standardized and efficient manner.

Continuous Integration and Continuous Deployment: The pipeline facilitates continuous integration by allowing developers to merge code changes frequently, ensuring early identification of integration issues. It also enables continuous deployment by automating the release of tested and approved changes to production environments, reducing the time and effort required for deployment.

CONCLUSION:

By implementing a comprehensive CI/CD pipeline, the project has achieved significant improvements in efficiency, code quality, collaboration, and deployment processes.

The integration of Jenkins, GitHub, SonarQube, and Docker has enabled automation and streamlining of the software development lifecycle. The project has realized faster time-to-market through automated builds, testing, and deployments, reducing manual effort and minimizing errors. The inclusion of code analysis tools like SonarQube has enhanced code quality by detecting and addressing issues early in the development process, resulting in more reliable and maintainable software.

The pipeline's seamless integration with GitHub has facilitated smooth collaboration, version control, and parallel development among team members, improving productivity and code consistency. The adoption of Docker containerization has provided scalability and portability, enabling efficient resource utilization and consistent deployments across different environments.

Overall, the project has demonstrated the value of implementing a robust CI/CD pipeline using Jenkins, GitHub, SonarQube, and Docker. It has proven to be an effective solution for enhancing software development processes, accelerating time-to-market, improving code quality, and promoting collaboration among development teams. The results achieved through the project have laid a strong foundation for organizations to adopt modern software development practices, embrace automation, and continuously improve their software delivery capabilities.

REFERENCES:

1. Sai Priya Sinde, Bhavika Thakkalapally, Meghamala Ramidi, Sowmya Veeramalla “Continuous Integration and Deployment Automation in AWS Cloud Infrastructure” 30 June 2022.
2. Artur Cepuc, Robert Botez, Ovidiu Craciun, Iustin-Alexandru Ivanciu and Virgil Dobrota, “Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes”, 19th RoEduNet Conference: Networking in Education and Research, 2020. DOI:10.1109/ROEDUNET51892.2020.9324857.
3. Leite, Leonardo, et al (2020) - A Survey of DevOps Concepts and Challenges, ACM Computing Surveys, vol. 52, no. 6
4. Maragathavalli, P.(2020) - Automation Pipeline and Build Infrastructure Using DevOps, International Journal for Research in Applied Science and Engineering Technology, vol. 8, no. 11
5. Mohammad, Sikender Mohsienuddin. (2018). Improve Software Quality through practicing DevOps Automation. SSRN Electronic Journal. 6
6. Erich, F. M., et al. (2017)- A Qualitative Study of DevOps Usage in Practice, Journal of Software: Evolution and Process, vol. 29, no. 6

7. Arachchi, S. A. I. B. S., Perera, I., “Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management,” 2018 Moratuwa Engineering Research Conference (MERCon) DOI:10.1109/mercon.2018.8421965.
8. Sriniketan Mysari; Vaibhav Bejgam “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible” 27 April 2020. DOI:10.1109/icETITE47903.2020.239
9. Noor Fathima.F;Vani H.Y ”Building, Deploying and Validating a Home Location Register (HLR) using Jenkins under the Docker and Container Environment” November 02,2020. DOI: 10.1109/ICOSEC49089.2020.9215458
10. Malathi. S1, Ganeshan. M2, ”Building and Deploying a Static Application using Jenkins and Docker in AWS” 4, June 2020. https://www.ijtsrd.com/papers/ijtsrd30835.pdf
11. Weber, I.; Nepal, S.; Zhu, L., (2016) Developing Dependable and Secure Cloud Applications. IEEE Internet Comput. 20
12. Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, and Liming Zhu. (2015). Securing a Deployment Pipeline. In Proceedings of the 2015 IEEE/ACM 3rd International Workshop on Release Engineering (RELENG '15). IEEE Computer Society, USA