Optimizing Continuous Integration and Continuous Deployment (CI/CD) Pipelines

Aman Lath
AIT-CSE
Chandigarh University
Mohali, India
21CDO1026@cuchd.in

Raghav Sadotra
AIT-CSE
Chandigarh University
Mohali, India
21BCS10309@cuchd.in

Hirday Mahajan
AIT-CSE
Chandigarh University
Mohali, India
21CDO1001@cuchd.in

Sejal Sepal
AIT-CSE
Chandigarh University
Mohali, India
21CDO1059@cuchd.in

***Abstract*— Continuous Integration and Continuous Delivery (CI/CD) are methods used in agile development to automate and speed up the process of building, testing, and validating services. To support and simplify all development and deployment processes, several methods such as containerized and CI/CD automation are needed. In this research, a DevOps Practice is carried out which includes process integration, deployment, and testing automatically using a tool called Jenkins. These tools are open-source automation servers to help the Continuous Integration and Continuous Deployment process. Jenkins is equipped with various open-source plugins that can be used to simplify and assist CI/CD automation and testing processes. The implementation of CI/CD in performance testing makes the testing process integrated, automated, and can be run regularly.**

Keywords*— *Automation, CI/CD, DevOps, Jenkins.

I. INTRODUCTION

CI/CD is a method of delivering applications regularly to customers by bringing automation into the application development phase. However, organizations often face obstacles such as inefficiency, implementation delays, sluggish behavior, and lack of automation when practicing CI/CD. These constraints can cause confusion between product delivery paths and take up system resource capacity. Thus, emphasizing the impact of human factors, CI/CD performance has become a popular field in software development research. In its application, CI/CD requires supporting tools to facilitate the process. Such as Git which is used as source code management and Jenkins as a tool to support the automation process Often developers work and collaborate with teams to complete their work assignments. Because the challenges of working with a team require continuous communication and good task resource management, developers need supporting tools used to perform resource management and interact with each other. That way jobs that have a high load can be handled easily using these supporting tools. In addition, because of the need for developers who work in teams, in the process of working on a task until the task is completed, developers need to review each other so that the work they are doing is sustainable and in line with expectations. Therefore, source code management is a supporting tool needed in the application system development process.

Process automation is often used in the industrial world to make work easier. With the automation process, developers will find it easy to integrate on a regular basis. This automation process is done with the help of supporting tools integrated with source code management. This automation process is the most important part of the CI/CD process. With the automation process, work that is done manually can be done and completed quickly by the automation process. In general, developers do tests in application development before the application is sent to the production environment to ensure whether the application is feasible to launch and will not throw errors when the application is running. One of these tests is performance testing where this test has an important role in the application development process. This performance test includes stress point testing with additional load methods, scalability testing, and stability testing.

In this study, a performance analysis of the CI/CD implementation will be done on a simple web application. Analysis activities include deployment and integration by utilizing an application called MyNotes. In addition, tools used for this project are Dockers, Jenkins, Docker Hub, Git and GitHub. It is hoped that the automation process for CI/CD performance in this study can be done automatically and reduce the role of humans in deployment of application. In addition, this research analysis is expected to reduce the problems of inefficiency, implementation delays, and sluggish behavior when practicing CI/CD.

II. Research Method

A. The research method carried out by the author can be seen in the flow chart shown in Figure 1.

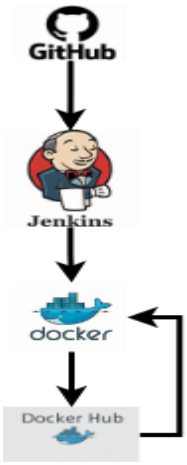

Fig. 1. Pipeline Flow

Based on the research flow chart, there are several stages carried out in this study. The stages of the research are as:

Design Stage: In this stage, a topology design is carried out which will be used as a reference for performance analysis of Web-app deployments on Jenkins. Figure 1. shows how the topology is used for this study. The topology describes the CI/CD pipeline process flow using Jenkins. The CI/CD process starts from the developer user building a python source code. Then by using Git, the python source code is pushed to the GitHub repository that has been created. Then the role of Jenkins begins to perform code integration by checkout python source code management and build configuration. At the build stage, Jenkins integrates the code that has been obtained from checkout to build an image which, when executed, will become a container containing the application to be run. All these processes will be recorded by a Jenkins plugin which integrates the build process with the real-time notification system using the. This process from building source code to running container is called CI/CD.

Installation and Configuration Stage: At this stage, the installation and configuration of the tools that will be used in this research are carried out. It can be seen that the installation and configuration steps required are as follows:

a) Installing Docker Engine Community (Docker Desktop): The Docker Engine Community (Docker Desktop) installation is used to unify applications with the containerized method. An indication that the Docker Desktop installation was successful is that you can pull "helloworld". The pull will display a message that the Docker installation went smoothly.

b) Installation and Configuration Environment: The installation environment for Jenkins uses a Jenkins image that has been pulled from the Docker Image Registry. To be able to access Jenkins via localhost, it must be ensured that the host has allowed the associated port to be accessed from outside. The port used for Jenkins environment is port 8080

c) Jenkins Configuration: To run the pipeline CI/CD process. The first CI/CD. Process is to pull the source code from the previously created GitHub Repository. Then after making sure all the configurations and source code are correct, then the next step is to carry out the build process. The indication of the build process was successful if the test script execution did not experience an error.

III. Result And Discussion

This study has a scope of testing to compare performance tests between web-app deployments using Jenkins services and conventional web-app deployments via local machine terminals. The web-app integration outputs a simple dynamic web application display that displays the client's IP address when the web-app is accessed.

A. Web-App Display Integration: The web-app integration results from the build on the job pipeline. The job of the configured job pipeline is to run the web-app integration via the build job. Figure 2 is a sequential web-appintegration process with job pipelines.

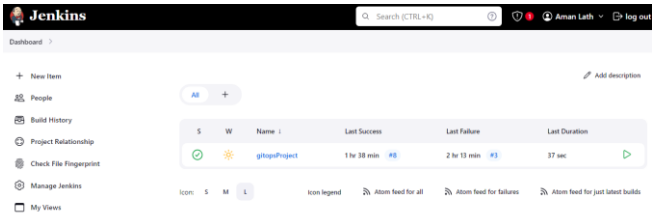

Fig. 2. Jenkins Dashboard

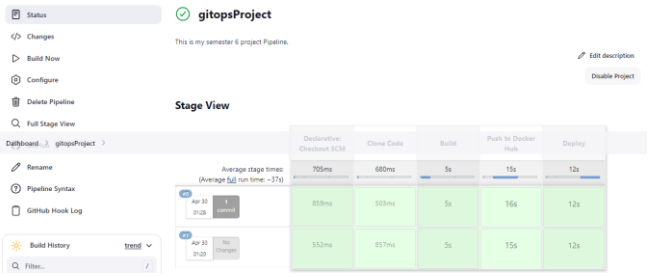

Fig. 3. Pipeline Stage view

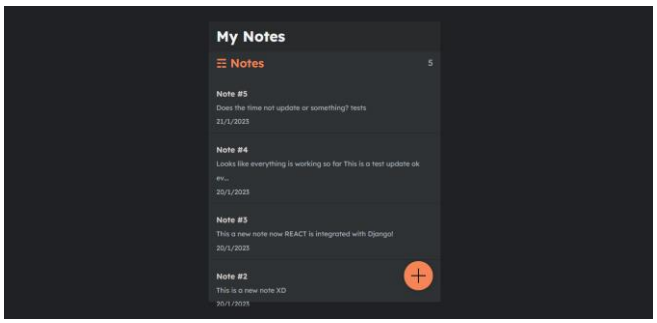

Fig. 4. Web-App view

In the pipeline there are several stages that represent the sub-tasks for the integration process. The green pipeline indicates that the stages have a successful status. Within each stage is displayed a time called a timestamp which is the time span of the process from which the stages took place. The jobs run by the pipeline are shown in Figure 4. The build jobs and test jobs can be seen on the Jenkins dashboard together with the job pipeline. To view the detailed contents and configuration of each job, click on the job name. The build job (sample-app) is tasked with integrating Docker containers derived from Docker web-app files so that web-apps can be run. Meanwhile, the test job will verify whether the web-app has been running successfully. Each job has a console output that is used to view the job process when it is run. The display of the results of the integration and deployment of the web-app is shown in Figure 4. The appearance of the web-app is generated by the css configuration that has been created and integrated with the html script. In the middle of the web-app display, it can be seen that the accessing ip is 20.244.106.200:8000 which is the ip of the client who is accessing the web-app.

IV. Conclusion

CI/CD is a series of activities in DevOps practice that simplify the application integration process and enable applications to continue to integrate and deploy on an ongoing basis. The integration process using Jenkins makes it easy and time efficient for developers to do application development because it uses an automation system from the CI/CD process. Performance testing is carried out to find out and ascertain whether a system will not produce errors at run time and has a value worth launching. In this study, the performance of web app integration using Jenkins has an interval of difference that is not too far from conventional integration with an average response time of 37.7 ms for integration using Jenkins and an average response time of 30.9 ms for conventional integration.

A. *ACKNOWLEDGEMENT*

1. Mr. Alok Das, Technical Trainer, Chandigarh University, as the supervisor of the final project and has been pleased to guide and support the author in the work of the final project.

2. Various parties who cannot be mentioned one by one for all the help and support in completing this final research article.

References

[1] S. Mysari and V. Bejgam, “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible,” Feb. 2020. doi: 10.1109/ic-ETITE47903.2020.239.

[2] J. Benjamin and J. Mathew, “Enhancing the efficiency of continuous integration environment in DevOps,” IOP Conference Series: Materials Science and Engineering, vol. 1085, no. 1, p. 012025, Feb. 2021, doi: 10.1088/1757-899x/1085/1/012025.

[3] J. Mahboob and J. Coffman, “A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework,” 2021 IEEE 11th Annual Computing and Communication Workshop and Conference, CCWC 2021, pp. 529–535, Jan. 2021, doi: 10.1109/CCWC51732.2021.9376148.

[4] G. Lettieri, V. Maffione, and L. Rizzo, “A Study of I/O Performance of Virtual Machines,” The Computer Journal, vol. 61, no. 6, pp. 808–831, Jun. 2018, doi: 10.1093/COMJNL/BXX092.

[5] Y. Luo, “Network I/O Virtualization for Cloud Computing,” IT Professional, vol. 12, no. 05, pp. 36–41, Sep. 2010, doi: 10.1109/MITP.2010.99.

[6] B. Igli Tafa et al., “The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines,” Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc, vol. 11, 2011.

[7] S. Jia, G. Juell-Skielse, and E. Uppström, “Integrating Conventional ERP System with Cloud Services From the Perspective of Cloud Service Type INTEGRATING CONVENTIONAL ERP SYSTEM WITH CLOUD SERVICES: FROM THE PERSPECTIVE OF CLOUD SERVICE TYPE”.

[8] J. Shah, D. Dubaria, and J. Widhalm, “A Survey of DevOps tools for Networking,” in 2018 9th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2018, Nov. 2018, pp. 185–188. doi: 10.1109/UEMCON.2018.8796814