CI/CD Pipelines

Abstract

Continuous Integration (CI) and Continuous Deployment (CD) pipelines are critical components of modern software development. By automating essential processes, they allow teams to develop software faster, more securely, and with fewer bugs, resulting in a far more efficient cycle. According to a "survey of 600 software developers, managers, and executives in the United States and the United Kingdom, only 3% of the respondents said they had no plans to adopt CD" [1]. This essay highlights the essential role of CI/CD pipelines and their impact on the software development lifecycle.

Introduction

In today's software development, the combination of speed and quality updates is key in creating successful products. One of the solutions is the practice of Continuous Integration and Continuous Deployment, in brief CI/CD pipelines. CI/CD pipelines streamline the process of quality assurance of the software. By automating the processes of running tests, integrating code, deploying software, checking for bugs and potentially other monitoring tools focused on code quality, it ensures faster and more reliable releases. Given the increasing level of complexity in the software world, the importance of these pipelines is constantly growing. This essay presents the role of CI/CD pipelines, focusing on their main features and the impact on development teams.

Opinion

Continuous integration (CI) helps developers to merge their code in a shared codebase. By automating the testing of code for compilation errors and for meeting the business requirements, it offers quick feedback and represents a proactive approach, which greatly reduces the number of integration problems or potential bugs in the production environment. By automating testing, it reduces the likelihood of creating new bugs or missing edge cases. With every push to the repository, it is usually tested by unit tests, integration tests and regression tests. This is a crucial aspect in a company as the product starts to become complex and team members change. Another key feature that is growing in popularity various scanners, for example the vulnerabilities scanner which highlights potential attacking vectors or code scanners that suggest changes to adhere to the best practices in the industry.

Continuous Deployments (CD) is the second part of the pipeline. After checking the new code to meet the quality checks, the deployment process begins. Usually, the deployment is done first in the staging environment, but it can be automated also to bring the code in production in the same way. By offering a seamless delivery process, it improves the users' experience, but also the developers' lives. The last part of the pipeline consists in monitoring tools for performance, health, error rates and other metrics. These quick feedback tools permit the teams to identify and address issues in the final environments.

A great deal of the benefits is already suggested by the definitions of Continuous Integration and Continuous Deployment pipelines. Below is a summary of the key advantages:

- Improve code quality: The errors are identified and fixed early in the full process of development and deployment. Fewer bugs mean a big cut in the cost of the future maintainability of the product. Fast feedback is key in the dynamics of modern software development. Another advantage is that it promotes early refactoring.

- Faster development cycles: The cycle of development is accelerated by lowering the effort of the developers and testers to integrate and deploy their code. Basically, this reduces the risk of delivery delays.
- Faster feedback because of the frequent deployment: “Frequent deployment is valuable because it allows your users to get new features more rapidly, to give more rapid feedback on those features, and generally become more collaborative in the development cycle. This helps break down the barriers between customers and development - barriers which I believe are the biggest barriers to successful software development.” [2]
- The human error is reduced: Humans are prone to mistakes every day and the repetitive routines increase this likelihood. By automating the process, the chance of bugs is decreased and significantly reduces the number of careless mistakes. Also, it standardizes the process.
- Lower costs: Better code quality and faster feedback, means lower costs in terms of maintainability in the longer run.
- “Better Business Advantage: Moving to a continuous development model gives our team the flexibility to make alterations to our software on-the-go to meet new market trends and user needs. We’ll be able to meet rapidly changing demands and will turn our release process into a competitive advantage.”[3]

Implementation description

The architecture of the implementation is divided into two main components: Git Fetcher and Agents Orchestrator. The primary technologies used are Java 21, Spring Boot 3, Maven, Thymeleaf for UI and Docker containers.

The Git Fetcher component is responsible for monitoring added repositories for updates at fixed intervals. The user can submit a Git URL via a form on a dedicated HTML page. The repository is validated and fetched. At regular intervals of time, the repository is checked for new commits. If a new commit is detected, it is fetched and a notification is sent to the Agents Orchestrator component.

Agents Orchestrator has multiple roles and includes several HTML pages for various settings and actions:

- Project settings page - users can activate and deactivate repositories to show or hide projects on the main CI/CD actions page. Additionally, they can enable or disable the automatic pipeline. When enabled, any notification from the Git Fetcher component about a new commit will automatically trigger the full CI/CD pipeline.
- Orchestrator settings page – this page allows users to update the number of agents live on the server
- Project actions – this is the main page for CI/CD actions. Active projects are listed along their possible actions. Users can execute actions using various buttons. One of them is the “Open terminal” button, which open a small form for submitting commands written by hand. Another button is the “Load actions from file..” button which executes the contained actions from an uploaded file. According to the progress, the actions buttons will be colored. An action in progress means that the button will become yellow. For a failed job, the color will be in red. For a job finished with success, it will become green. The possible actions are:
    - Clean – cleans the project build
    - Fetch – copies the repo on a docker container
    - Build – builds the project

- Run Tests – execute the implemented tests
- Last output – displays the last available output created at the previous actions
- All steps – runs the entire pipeline performing all the above actions, in order.

Agents Orchestrator contains a queue of submitted jobs and a pool of agents. Each job is assigned to an agent, in order. If all agents are busy, the jobs will remain in the queue until one of the agents becomes available. Each agent is implemented using a dedicated Docker container. If an agent is restarted, its containers will be automatically recreated.

Conclusion

CI/CD pipelines are crucial in the reliability and development of today's software. They are no longer just a good practice, but have become a mandatory part of the software development lifecycle. Automating workflows and promoting quality helps tremendously to deliver faster updates, improve user satisfaction, and in the end, greatly reduce the maintenance costs for companies.

References

[1] Chen, L. (2017). Continuous delivery: overcoming adoption challenges. Journal of Systems and Software, 128, 72-86.

[2] Fowler, M., & Foemmel, M. (2006, May). Continuous integration.

[3] Nath, M., Muralikrishnan, J., Sundarrajan, K., & Varadarajanna, M. (2018). Continuous integration, delivery, and deployment: a revolutionary approach in software development. *International Journal of Research and Scientific Innovation (IJRSI)*, *5*(7), 185-190.

Pulkkinen, V. (2013, August). Continuous deployment of software. In *Proc. of the Seminar* (Vol. 58312107, p. 46).