International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Implementing continuous integration and continuous deployment (CI/CD) pipelines

Nagaraju Thallapally *

University of Missouri Kansas City.

International Journal of Science and Research Archive, 2021, 03(02), 248-253

Publication history: Received on 16 September 2021; revised on 21 October 2021; accepted on 28 October 2021

Article DOI: https://doi.org/10.30574/ijsra.2021.3.2.0073

Abstract

Continuous Integration (CI) and Continuous Deployment (CD) are fundamentals of today's software development to deliver quality code, deliver faster, and reduce the cost of deployment. Automation of integration and deployment helps teams find the faults before they are found and eliminate manual effort during the development lifecycle with CI/CD pipelines. This article talks about the main parts of CI/CD pipelines, their benefits, best practices, and the tools available to help them. We also cover common organizational barriers to CI/CD implementation and give you solutions for how to overcome them. We also share case studies and the future of CI/CD.

Keywords: Continuous Integration (CI); Continuous Deployment (CD); Quality code; Deployment automation; CI/CD pipelines; Automation; Fault detection; Manual effort reduction

1 Introduction

Advanced software development practices developed in recent years now require teams to deliver software faster and maintain better quality standards. Development teams are now required to produce software releases more rapidly without sacrificing quality and security while also keeping costs low. Software development methodologies that depend on manual testing and deployment processes demonstrate consistent inefficiencies along with higher error rates and delays, which lead to increased operational expenses and postponed product launches. The adoption of new methods that automate and streamline essential development stages emerged because existing processes presented too many limitations, particularly Continuous Integration (CI) and Continuous Deployment (CD) (Shahin et al., 2017).

Continuous Integration (CI) represents the automated process that merges code changes submitted by several contributors into one codebase multiple times throughout the day. The process works to identify integration problems at an early stage and gives developers feedback right away, which helps to prevent bigger conflicts later during development. Continuous Deployment (CD) takes automation to the next level by handling the full deployment pipeline, which allows teams to push updates to production multiple times daily. Through Continuous Deployment (CD), every code change that successfully passes automated testing is directly sent to production to sustain fast yet reliable update delivery.

CI/CD pipeline implementation has transformed software development through faster code deployment to production, which improves code quality and reduces bug introduction and regression risks. Automated systems streamline the code integration and deployment process by removing manual work steps so every modification gets automatic testing and error validation, which permits developers to concentrate on building superior features instead of repetitive work activities. CI/CD benefits developers while also improving collaboration between development teams and operations along with QA teams by offering shared insights into software performance and production readiness.

* Corresponding author: Nagaraju Thallapally

Through this paper, we examine CI/CD foundational principles as well as their practical implementation, employing various technological tools including Jenkins, Docker, and Kubernetes. Our analysis includes real-world benefits from CI/CD adoption, such as accelerated delivery cycles as well as improved code quality alongside stronger deployment processes. Our discussion will cover key best practices organizations need to follow when adding CI/CD into their workflows together with potential organizational obstacles they might face during implementation. Our final discussion topic will explore case studies of organizations that have implemented CI/CD successfully, together with future developments in these practices and emerging trends along with technologies.

This paper will teach readers the operation of CI/CD pipelines and their importance in contemporary software development, together with strategies for successful integration into organizational workflows.

2 Overview of Continuous Integration and Continuous Deployment

2.1 Continuous Integration (CI)

Continuous integration involves combining all developer code into a common repo at least daily. Automated builds and tests are run on all integrations to ensure there are no early errors. The fundamental tenets of CI are code commits frequently, builds automated, and developers are given immediate insight on the quality of their code (Duvall et al., 2007).

Key CI practices include:

- Automated code integration to the shared repository.
- Running unit tests and other automated tests on each integration.
- Code quality checks using static analysis tools.
- Fast feedback loops to developers regarding the status of their commits.

2.2 Continuous Deployment (CD)

Continuous deployment takes the CI approach one step further by pushing the updates to production on autopilot once the integration and testing go well. This enables faster and more consistent new feature and bug fix delivery to end users (Chen, 2018).

CD consists of two main aspects:

- **Continuous Delivery**: The ability to push changes into staging or production environments automatically, ensuring that the code is always ready for release.
- **Continuous Deployment**: An automated deployment process where each change that passes automated tests is directly pushed to production without manual intervention.

2.3 CI/CD Pipeline Overview

A CI/CD pipeline is a process in which code can be automatically built, tested, and deployed by developers. The pipeline usually includes:

- **Source Control**: Code is held in a version control system (e.g., Git).
- **Automation of Build**: The code is built and packaged into deployable units (e.g., Docker containers).
- **Automated Testing**: All the unit tests, integration tests, etc., are automatically performed.
- **Automation Deployment**: The pipeline pushes the app to staging and/or production with automation tools.

3 Benefits of CI/CD

The benefits of CI/CD pipelines for improving software development are many:

3.1 Faster Delivery

Automation in the build, test, and deployment process enables teams to provide code changes on a faster and more consistent basis with CI/CD. Since changes are integrated all the time, teams can push releases to production more often, which reduces time to market for new features or fixes.

3.2 Improved Code Quality

Automated testing and continuous integration also discover bugs and regressions early on in development. CI pipelines are often built with unit tests, integration tests, and more that make sure that the codebase is always stable and high quality.

3.3 Reduced Deployment Risk

If you integrate and deploy code incrementally over time, then there is less chance of serious deployment failures. Problems can be identified and fixed easily, and the rollback process is easier to deal with.

3.4 Enhanced Collaboration

CI/CD supports the development, operations, and quality assurance teams to work together with one shared vision: to write high-quality, deployable code. Automatic work avoids any manual input from teams so that they can concentrate on higher-level problems.

3.5 Scalability and Flexibility

CI/CD pipelines can be scaled to larger teams, large projects, and distributed environments. They help organizations have multiple deployment pipelines, including staging, pre-production, and production environments.

4 Tools for Implementing CI/CD Pipelines

CI/CD pipelines can be used in many tools that can benefit various parts of the process from versioning to deployment automation. Below are some of the most popular CI/CD tools:

4.1 Version Control Systems

Git: The most popular version control system distributed. Git repositories, on the likes of GitHub, GitLab, and Bitbucket, are the basis of most CI/CD projects.

4.2 Build and Test Automation Tools.

- **Jenkins:** A free automation server used for software building, testing, and deployment.
- **Travis CI:** A cloud-based CI provider that works with GitHub repos and supports integration and deployment pipelines.
- **Circle CI:** A CI/CD tool for fast builds and integrations with version control in the cloud.
- **GitLab CI:** GitLab CI, an integration and pipeline automation tool that is part of GitLab's DevOps platform.
- **Maven/Gradle:** Build automation platform for Java application dependency and packaging management.

4.3 Deployment and Orchestration Tools

- **Docker:** Containerization solution that creates the same environments for development, testing, and deployment at different points in the pipeline.
- **Kubernetes:** An open-source container orchestration system for running and scaling containerized applications.
- **Ansible:** A configuration manager that performs infrastructure provisioning and deployment tasks automatically.
- **Terraform:** Infrastructure-as-code library to provision and maintain the cloud, backed by CI/CD pipelines for deployment automation.

4.4 Monitoring and Logging Tools

- **Prometheus:** A free monitoring and alerting toolkit that can be integrated with CI/CD pipelines to help teams track the health and performance of deployed applications.
- **Grafana:** Graphical visualization of Prometheus monitoring and data-based data to visualize production system dashboards in real time.

5 Best Practices for CI/CD Implementation

Using CI/CD should be a best practice within organizations:

5.1 Start Small

Do not try automating the entire pipeline in one go; first create a small pipeline that automates the most basic steps (building, testing, deployment) and build it from there.

5.2 Automate Testing

The unit test, the integration test, and the other kinds of automated tests are crucial to keep the code clean and do not break the application by changing anything.

5.3 Use Feature Flags

Feature flags enable new features but do not enable them in real time to end users.
This gives scalability and eliminates risk in deployment.

5.4 Ensure Fast Feedback

The sooner the feedback from the CI/CD pipeline, the better. Developers should get notifications directly if the code crashes during integration, and they can quickly solve it.

5.5 Maintain a Clean Codebase

A clean codebase is key to successful CI/CD. Coding practices like code refactoring, code review, and following code standards can make your code more maintainable and simpler.

6 Challenges in Implementing CI/CD

As beneficial as it is, there are a few issues with CI/CD pipelines adoption for organizations:

6.1 Resistance to Change

This cultural resistance to automation and workflow transformations can be a big roadblock to adopting CI/CD. Teams who have already worked with processes might not be eager to use automation.

6.2 Tool Complexity

Organizations can find choosing and deploying the right tools daunting. There are various CI/CD tools and services out there that should be weighed in consideration of which best meets the requirements of the team.

6.3 Infrastructure and Resource Management

Ensuring that infrastructure resources can handle the extra load that is induced by build and test repeats is another problem. This can be done by optimizing server use or using the cloud to dynamically scale resources.

6.4 Ensuring Security

Security issues, like the handling of sensitive data (e.g., API keys, credentials), can make CI/CD pipelines difficult. You should use security techniques such as secrets that are stored safely and dependencies that will be vulnerability checked.

7 Real-World Case Studies

7.1 Netflix

Netflix is a pioneer of CI/CD. The company uses microservices architecture with pipelines to apply hundreds of updates per day, which allows for high availability and continual new features and improvements.

7.2 Spotify

Spotify uses CI/CD to drive the delivery cycle fast and Jenkins and Kubernetes to automate testing and deployment across their distributed teams.

8 Future Trends in CI/CD

The future of CI/CD will be more AI/ML-based testing, serverless architectures, and integration with cloud-native technologies. And there will be more DevSecOps that integrate security in the CI/CD pipeline.

9 Conclusion

CI/CD pipelines are the change to the methodology for software development and delivery. Automating some processes makes it easier for teams to make mistakes, collaborate better, and speed up release time. CI/CD isn't easy, but the payoffs are vastly greater and are a must-have for any company that wants to stay ahead of the game in the fast-paced software world. If you've got the right tools, the best practices, and are always striving for improvement, any company can embrace CI/CD and bring about better software quality and performance.

References

[1] Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: improving software quality and reducing risk*. Pearson Education.

[2] Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

[3] Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. It Revolution.

[4] Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., & Vasilescu, B. (2017, October). The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 60-71). IEEE.

[5] Martin, R. C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

[6] Railić, N., & Savić, M. (2021, March). Architecting continuous integration and continuous deployment for microservice architecture. In *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1-5). IEEE.

[7] Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021, December). On continuous integration/continuous delivery for automated deployment of machine learning models using mlops. In *2021 IEEE fourth international conference on artificial intelligence and knowledge engineering (AIKE)* (pp. 25-28). IEEE.

[8] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access*, *5*, 3909-3943.

[9] Chen, L. (2018, April). Microservices: architecting for continuous delivery and DevOps. In *2018 IEEE International conference on software architecture (ICSA)* (pp. 39-397). IEEE.

[10] Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.

[11] Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE software*, *32*(2), 50-54.

[12] Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, *87*, 48-59.

[13] Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, *57*, 21-31.

[14] Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, *82*, 55-79.

[15] Sethi, F. (2020). Automating software code deployment using continuous integration and continuous delivery pipeline for business intelligence solutions. *Authorea Preprints*.

[16] Mysari, S., & Bejgam, V. (2020, February). Continuous integration and continuous deployment pipeline automation using Jenkins Ansible. In *2020 International conference on emerging trends in information technology and engineering (IC-ETITE)* (pp. 1-4). IEEE.

[17] Pratama, M. R., & Kusumo, D. S. (2021, August). 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.

[18] Bobrovskis, S., & Jurenoks, A. (2018). A Survey of Continuous Integration, Continuous Delivery and Continuos Deployment. In *BIR workshops* (pp. 314-322).

[19] Deepak, R. D., & Swarnalatha, P. (2019). Continuous Integration-Continuous Security-Continuous Deployment Pipeline Automation for Application Software (CI-CS-CD). *International Journal of Computer Science and Software Engineering*, *8*(10), 247-253.

[20] Virtanen, J. (2021). Comparing Different CI/CD Pipelines.