MODELS FOR AUTOMATING CI/CD PROCESSES IN RUBY WEB APPLICATION DEVELOPMENT

Topalidi A.
specialist degree, Moscow State University of Geodesy and Cartography
https://doi.org/10.5281/zenodo.15025001

Abstract

This paper explores the automation of Continuous Integration/Continuous Delivery (CI/CD) processes in Ruby web application development and its impact on software reliability and delivery speed. It examines modern automation tools such as GitHub Actions, Jenkins, and GitLab CI, focusing on their architectural characteristics, operational principles, and integration capabilities within the Ruby ecosystem. Deployment strategies, including blue-green deployment, rolling deployment, and canary deployment, are analyzed alongside approaches for optimizing CI pipelines, dependency management, and testing. Special attention is given to methods for enhancing fault tolerance, enabling automatic scaling, implementing monitoring solutions, and managing application environments effectively.

Keywords: automation, Continuous Integration/Continuous Delivery (CI/CD), web applications, Ruby, deployment, testing.

Introduction

In modern software development, the rapid evolution of technology has established Continuous Integration and Delivery (CI/CD) automation as an essential component of an efficient and reliable development pipeline. Automated approaches save considerable time-to-market, promote high-quality code, and minimize software failure occurrences. Because of its inherent malleability and versatility, web programs developed with Ruby require a tailor-made model for CI/CD; therefore, it is important for developers to explore relevant automation tools specifically for such programs.

A key field of inquiry in such a model is a comparative evaluation of CI/CD tools, such as GitHub Actions, Jenkins, and GitLab CI, each with its specific integration, testing, and delivery-related features. Comparison of these tools enables a critical evaluation of individual strengths and weaknesses, determination of best use cases, and development of integration approaches for enhancing web application delivery efficiency and dependability. The aim of this article is to conduct a comparative assessment of automation tools, such as GitHub Actions, Jenkins, and GitLab CI, focusing on the development and evaluation of integration and deployment models for Ruby-based web applications.

Main part. The concept of automation of CI/CD processes in web application development and comparative analysis of automation tools

Modern software development practices focus on improving team efficiency, accelerating release cycles, and ensuring software stability. A crucial aspect of this process is the automation of Continuous Integration (CI) and Continuous Delivery (CD), or in some cases, Continuous Deployment. These methodologies minimize risks associated with code integration, expedite testing, and guarantee seamless application deployment into production environments [1].

The process of automatically validating new changes before merging them into the main branch of a project is a fundamental aspect of **CI.** This is achieved through automated testing, static code analysis, and other verification steps that help detect errors early and prevent regressions. CI also reduces merge conflicts, simplifies project maintenance, and ensures adherence to quality standards. A crucial aspect of CI is its integration with version control systems, which streamlines validation processes and enhances collaboration.

The automation of software releases into production environments is a fundamental aspect of CD. **Continuous Delivery** ensures that software is always in a deployable state, allowing it to be released on demand, while **Continuous Deployment** removes manual intervention by automatically deploying updates after successful testing. These approaches significantly shorten the time between code development and deployment, making them particularly beneficial for web applications requiring frequent updates.

The automation of CI/CD processes in web application development necessitates the selection of an appropriate tool capable of meeting project-specific requirements. Among the most widely adopted solutions are GitHub Actions, Jenkins, and GitLab CI, which provide robust capabilities for automating build, testing, and deployment workflows (fig.1).

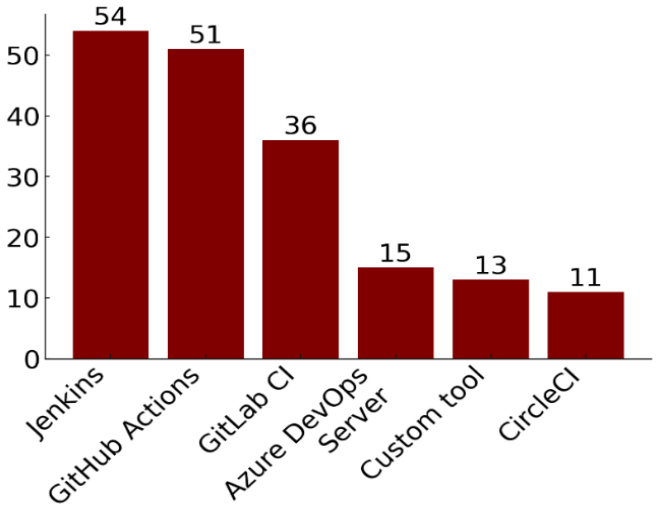

Figure 1. Distribution of regularly used CI systems among worldwide developers in 2023, % [2]

These tools differ in their architecture, functionality, workflow configuration approaches, and integration with various ecosystems, making their comparative evaluation essential for determining the most suitable option for a given development environment. Understanding the characteristics of these technologies enables the identification of their strengths and weaknesses, as well as the determination of the most effective use cases in Ruby web application development.

GitHub Actions is a native automation system integrated into GitHub, allowing developers to create and configure workflows directly within their repositories. Workflows are defined using YAML files, specifying execution triggers, job sequences, and associated actions. One of the primary advantages of GitHub Actions is its deep integration with GitHub, which enables seamless CI/CD setup without the need for additional servers or complex configurations [3].

GitLab CI is a built-in component of the GitLab platform, providing seamless integration with repositories, task management, and deployment environments. Workflows are defined in a **.gitlab-ci.yml** file, following a structure similar to that of GitHub Actions [4]. However, GitLab CI distinguishes itself through its robust runner management system, which enables efficient allocation of computing resources for executing CI/CD tasks. This feature makes it a highly versatile solution, particularly in enterprise environments where load balancing across multiple deployment environments is essential. In GitLab CI/CD workflow, stages include branch creation, automated testing, review and approval, and deployment to production (fig. 2).

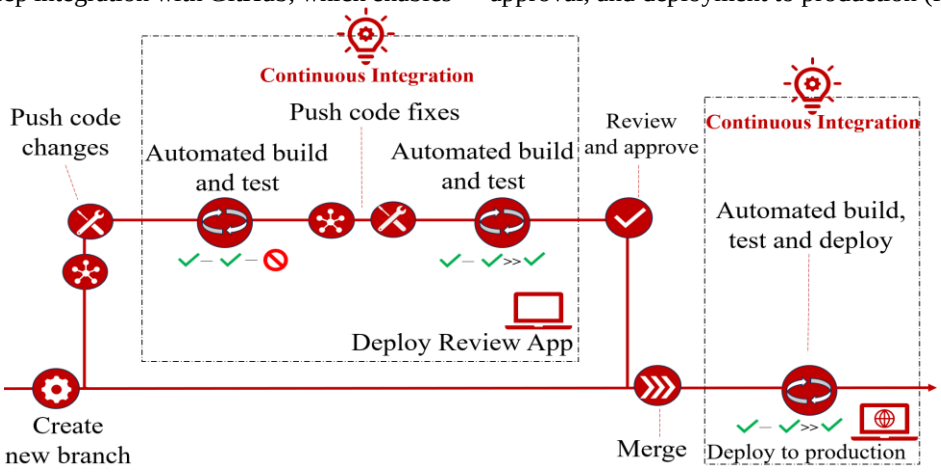

Figure 2. GitLab CI/CD pipeline structure [5]

The image illustrates a GitLab CI/CD pipeline structure, demonstrating the flow of code from development to production. It begins with creating a new branch and pushing code changes, which triggers an automated build and test process as part of continuous integration. If issues are detected, developers push fixes, and after successful testing and approval, the code is merged, leading to automated deployment to production.

Jenkins, in contrast, is a standalone automation server that requires installation and configuration on a dedicated machine or within a cloud environment. Unlike GitHub Actions, which is tightly integrated with GitHub, Jenkins is version control-agnostic, supporting Git, Subversion, Mercurial, and other repositories. This makes it a highly flexible CI/CD tool, particularly for projects requiring integration with multiple source control systems and deployment environments. However, its setup and maintenance demand considerable effort, particularly in managing dependencies and ensuring server reliability and fault tolerance.

Selecting an appropriate CI/CD tool for Ruby web application development requires careful consideration

of specific requirements related to testing, building, and deployment (table 1).

Table 1.

Analysis of CI/CD tools and supported features [6, 7]

Criterion	GitHub Actions	Jenkins	GitLab CI
Architecture	Integrated into GitHub, serverless.	Self-hosted automation server.	Integrated into GitLab, supports cloud and self-hosted runners.
Configuration approach	YAML-based workflow files stored in the repository.	Declarative and scripted pipelines requiring manual setup.	YAML-based configuration managed within GitLab.
Flexibility	Limited by GitHub's ecosystem, but extensible through actions.	Highly flexible with extensive plugin support	Flexible but tightly coupled with GitLab infrastructure
Scalability	Limited to GitHub-hosted or self-hosted runners.	Scalable with dedicated infrastructure, requires manual optimization.	Scalable with built-in runner management and custom allocation.
Automatic event-based triggering	Yes	Yes (requires webhook configuration)	Yes
Cloud integration	Seamless integration with GitHub Cloud, third-party services supported via actions.	Cloud-agnostic but requires plugins for cloud deployment.	Deep integration with GitLab Cloud and third-party cloud providers.
Prebuilt templates for Ruby applications	Yes (Ruby setup action)	No	Yes

For Ruby web applications, it is pivotal to consider testing requirements, dependency management, and deployment strategies. In smaller projects hosted on GitHub, GitHub Actions can serve as an optimal solution due to its seamless integration and ease of use. For large-scale enterprise projects requiring complex CI/CD workflows, Jenkins remains a powerful tool, offering extensive configurability and scalability. In teams operating within the GitLab ecosystem, GitLab CI provides the most streamlined pipeline management experience. The selection of a CI/CD tool should be based on the specific needs of the project, the availability of administrative resources, and scalability requirements.

Integration and deployment approaches: enhancing reliability and accelerating delivery of Ruby applications

Efficient automation of CI/CD processes in Ruby web application development requires not only selecting the right tools but also implementing effective integration and deployment strategies. With frequent updates, the need for fault tolerance, and the demand for minimal downtime, it is crucial to adopt approaches that ensure stability while streamlining the delivery pipeline.

In Ruby web development, continuous testing and seamless integration of code changes play a pivotal role. Automated testing enables early detection of errors, preventing defects from reaching the production environment. By incorporating robust testing pipelines into CI/CD workflows, development teams can maintain high code quality, reduce deployment risks, and ensure stable application performance.

A widely adopted approach involves the separate execution of unit tests (RSpec, Minitest), integration tests, and static code analysis (RuboCop, Brakeman). Segmenting tests into independent stages reduces the computational load on the CI system and accelerates error detection. Additionally, dependency caching (Bundler, Yarn, Node.js) is employed to minimize pipeline execution time during subsequent runs.

Efficient CI pipeline performance also necessitates database optimization in the testing environment. Parallel test execution, using the **database_cleaner** gem and multiple worker processes, reduces testing duration. For projects utilizing PostgreSQL or MySQL, employing **database snapshots** instead of executing migrations before each test run is recommended to enhance efficiency and reduce setup time.

Deploying Ruby applications presents obstacles related to dependency management, database migrations, and interactions with web servers like Puma, Unicorn, and Passenger. To mitigate these challenges, several deployment strategies are commonly used: **Blue-Green Deployment**, which maintains two identical environments, applying updates to a standby instance before switching traffic to minimize downtime and simplify rollbacks; **Rolling Deployment**, where updates are gradually distributed across servers to reduce the risk of system-wide failures; and **Canary Deployment**, which initially releases new versions to a small subset

of servers or users to assess potential issues before full-scale deployment [8].

A significant aspect of deployment is database migration management. When using relational databases in Ruby on Rails, it is recommended to separate migration execution from the deployment process to prevent schema inconsistencies during code updates. A common practice involves running migrations as a distinct stage before the primary deployment or leveraging zero-downtime migration techniques, such as those provided by the **strong_migrations** gem. These techniques prevent schema inconsistencies by ensuring that database changes are backward-compatible, avoiding downtime during deployments.

To minimize downtime during the deployment of Ruby web applications, the preboot strategy is widely employed. This approach involves launching a new server process in advance, followed by a smooth traffic switchover, ensuring minimal disruption to users.

The reliability of CI/CD pipelines depends not only on the correctness of automated processes but also on environment management, logging, and production monitoring. For Ruby applications, it is crucial to maintain distinct environments (development, test, staging, production) with isolated configurations. The use of environment variables, like "**dotenv"**, Rails credentials, AWS Parameter Store) enables centralized management of sensitive data without exposing it in the codebase, thereby enhancing security and maintainability.

Effective monitoring and logging are critical for failure prevention and rapid incident response. Modern CI/CD pipelines utilize tools such as Prometheus, Grafana, New Relic, and Datadog to enable real-time tracking of performance metrics, server health, and error rates. Centralized logging solutions like Logstash, Fluentd, Loki, and Papertrail enable real-time event analysis for identifying system failures, while automated alerting tools such as PagerDuty, Slack, and OpsGenie provide instant notifications to development and operations teams in the event of critical failures. To ensure high availability, autoscaling and load balancing mechanisms (e.g., NGINX, HAProxy, AWS Elastic Load Balancer) are widely used in Ruby web applications. These techniques allow for dynamic allocation of active servers based on workload fluctuations, which is particularly crucial in cloud environments.

Optimizing CI/CD workflows for Ruby web applications requires a comprehensive approach that includes test acceleration, deployment strategy selection, and effective production environment management [9]. Implementing separate test stages, dependency caching, and parallel task execution reduces pipeline execution time, while deployment strategies such as blue-green, rolling, and canary deployment enhance release reliability and minimize downtime.

Conclusion

Efficient automation of CI/CD processes in Ruby web application development accelerates the release cycle, enhances software reliability, and minimizes deployment-related risks. Leveraging tools such as GitHub Actions, Jenkins, and GitLab CI enables flexible management of integration and delivery workflows while accounting for project architecture and scalability requirements. The implementation of optimized testing, deployment, and monitoring strategies plays a crucial role in ensuring application stability, reducing the likelihood of errors, and minimizing downtime. Advanced techniques, including parallel task execution, automated scaling, and continuous monitoring, contribute to high fault tolerance and maintain a consistently high standard of software quality throughout the entire development lifecycle.

References:

1. Zampetti F., Geremia S., Bavota G, Di Penta M. CI/CD pipelines evolution and restructuring: A qualitative and quantitative study //2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021. P. 471-482. DOI: 10.1109/ICSME52107.2021.00048

2. The State of Developer Ecosystem 2023 Report: Team Tools / Jetbrains // URL: https://www.jetbrains.com/lp/devecosystem-2023/team-tools/ (date of application: 25.01.2025).

3. Kinsman T., Wessel M., Gerosa M. A., Treude C. How do software developers use github actions to automate their workflows? //2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, 2021. P. 420-431. IEEE. DOI: 10.1109/MSR52588.2021.00054

4. Calefato F., Lanubile F., Quaranta L. A preliminary investigation of MLOps practices in GitHub //Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2022. P. 283-288. DOI: 10.1145/3544902.3546636

5. CI/CD concepts / Gitlab // URL: https://docs.gitlab.co.jp/ee/ci/introduction/ (date of application: 28.01.2025).

6. Dudak A.A. Optimize development and testing processes with vite, storybook and vitest // Innovacionnaya nauka. 2024. No. 9-1. P. 21-25. EDN: CCGPLL

7. Rostami Mazrae P., Mens T., Golzadeh M., Decan A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis //Empirical Software Engineering. 2023. Vol. 28. №. 2. P. 52. DOI: 10.1007/s10664-022-10285-5 EDN: DQCFVE

8. Amgothu S. Innovative CI/CD Pipeline Optimization through Canary and Blue-Green Deployment //International Journal of Computer Applications. 2024. Vol. 186. №. 50. P. 1-5. DOI: 10.5120/ijca2024924141 EDN: GZBIIC

9. Garifullin R. Development and implementation of high-speed frontend architectures for complex enterprise systems // Cold Science. 2024. №12. P. 56-63. EDN: ZZUNWR