Warsaw University of Technology
Faculty of Electrical Engineering

Comparative Analysis of Continuous Software Delivery Tools Using Github Actions and Jenkins as Examples

Paweł Prokopiuk
Faculty of Electrical Engineering
Warsaw University of Technology

Introduction

Modern software development relies on Continuous Integration and Continuous Deployment (CI/CD) to automate and streamline delivery. This study compares two prominent CI/CD tools:

- **GitHub Actions**: A cloud-based platform integrated with GitHub, launched in 2019, offering over 12,000 reusable actions in its Marketplace by 2022.
- **Jenkins**: A self-managed, open-source automation server with over 1,800 plugins, known for its customization and flexibility.

Objective: Assess configuration complexity, integration, performance, scalability, and cost to guide tool selection. *Research Gap*: Limited empirical comparisons in real-world scenarios.

Challenges in CI/CD

CI/CD tools must tackle several challenges:

- **Configuration Complexity**: Simplifying pipeline setup and maintenance.
- **Performance**: Ensuring fast build and deployment times.
- **Integration**: Supporting external tools, cloud platforms, and microservices.
- **Scalability**: Managing concurrent jobs and large-scale projects.
- **Cost**: Balancing cloud-based vs. self-hosted infrastructure costs.

Additional challenges include automating testing across environments and ensuring security compliance in pipelines.

Results

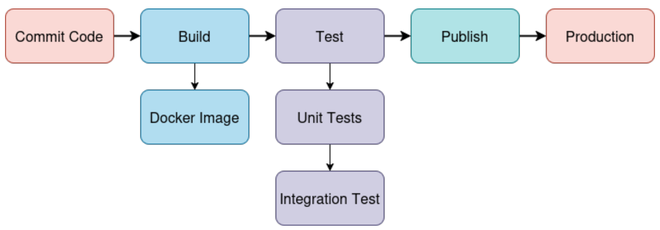
The analysis highlights key differences between GitHub Actions and Jenkins. GitHub Actions uses 50 YAML lines for a simple, declarative setup, while Jenkins requires 150 lines with detailed scripting and plugins. Performance-wise, self-hosted Jenkins is faster with ample resources, but GitHub Actions on the cloud has variable times, though its self-hosted runners align closer to Jenkins. For scalability, cloud-based GitHub Actions stays stable for concurrent jobs, unlike Jenkins and self-hosted setups, which slow down as concurrency rises due to resource competition.

Methodology

The study evaluates GitHub Actions and Jenkins based on:

- **Criteria**:
  - Configuration complexity (lines of code, setup time).
  - Integration with external tools and cloud platforms.
  - Performance (build and deployment times).
  - Scalability (concurrent job handling).
- **Setup**:
  - Identical pipelines across platforms: Jenkins, GitHub Actions (cloud-hosted), and GitHub Actions (self-hosted).
  - Self-hosted runners used cloud infrastructure matching GitHub Actions' resources.
  - Stages: Build, Test, Deliver, Deploy.
  - Triggered by Git commits.
  - Projects analyzed: Network, mobile, monolithic, and microservices architectures.
  - Deployment types: Containerized, serverless, and VM-deployed applications.

Containerized applications were prioritized due to their encapsulated environments, ensuring consistent deployment across platforms. This approach allowed for a fair comparison by standardizing the pipeline triggers and workloads. The variety of project types ensured the evaluation captured diverse CI/CD use cases. This methodology provides a robust framework for assessing tool performance in real-world scenarios.



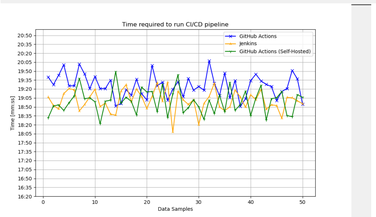
Rysunek 1. CI/CD pipeline structure.

Performance Summary

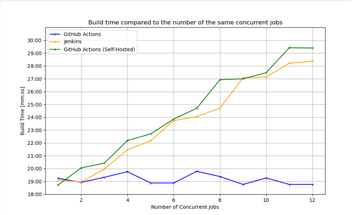
Platform	Config.	Build Time	Scalability	Integration
GitHub Actions (Cloud)	50 YAML	Variable	High	GitHub, Cloud
GitHub Actions (Self-Hosted)	50 YAML	Comparable	Moderate	GitHub, Cloud
Jenkins	150 Lines	Faster	Moderate	Plugins

Tabela 1. Comparative performance metrics.

Visual Insights



Rysunek 2. Build times across 50 runs.



Rysunek 3. Concurrent job performance.

Visualizations show GitHub Actions' scalability for concurrent jobs and Jenkins' speed in resource-rich environments.

Discussion

The comparison highlights:

- **GitHub Actions**: Best for rapid setup and cloud-native projects, with simpler workflows.
- **Jenkins**: Ideal for complex, customized pipelines with extensive integration needs.
- **Trade-offs**:
  - GitHub Actions: Limited by cloud pricing plans.
  - Jenkins: Requires infrastructure maintenance.
- **Insight**: Performance varies with environment; self-hosted runners reduce differences.

Conclusion and Future Work

- **GitHub Actions**: Suited for cloud-native, mid-sized projects with ease of use.
- **Jenkins**: Best for complex, resource-intensive projects needing customization.
- **Future Work**: Develop frameworks for hybrid CI/CD tool integration to optimize performance.
- **Takeaway**: Choose based on project needs for configuration, scalability, and integration.