Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2023, 10(3):105-109

Research Article ISSN: 2394 - 658X

GitHub Actions vs. Jenkins: Choosing the Optimal CI/CD Pipeline for your GCP Ecosystem

Srinivas Adilapuram

Software Engineer, Equifax Inc, USA

ABSTRACT

GitHub Actions and Jenkins are both Continuous Integration/Continuous Delivery (CI/CD) tools that can create CI/CD pipelines in various environments and ecosystems, including the Google Cloud Platform (GCP). Jenkins is an open-source, older, and more comprehensive tool requiring significant skills and effort. GitHub Actions is simpler and closely tied to the GitHub platform. This article compares the two tools in ten dimensions: workflow definition and configuration, scalability and performance, authentication, security and monitoring, cost management, community, ecosystem and plugin support, parallelism and concurrency, deployment strategies, vendor lock-in, pipeline maintenance, and dependency management and speed of innovation. How they compare in each dimension within the GCP ecosystem can help organizations make an informed decision about which CI/CD pipeline to use.

Keywords: CI/CD Pipeline, GCP Ecosystem, Jenkins, GitHub Actions

INTRODUCTION

Whether you are developing for the Google Cloud Platform (GCP) ecosystem or within it, augmenting your process with automation and DevOps practices can significantly streamline it. Continuous Integration/Continuous Delivery (CI/CD) pipelines are a core part of such processes and allow developers to automate a significant segment of code development and deployment, accelerating the project's pace while simultaneously enhancing the overall quality of the build. However, a critical factor that influences everything from ease of integration to development quality is making the right choice regarding the CI/CD pipeline. GitHub Actions and Jenkins are both strong contenders, but it's important to compare their various strengths and limitations in the context of the GCP ecosystem if that's the focus of your development.

LITERATURE REVIEW

There is limited scientific literature (papers and research articles) review on the specific topic (especially in the GCP context), but there are several detailed comparisons of the two pipelines – GitHub Actions and Jenkins. Most of them are in the form of blogs and articles, but there are papers as well, some of them comparing not just these but other CI/CD pipelines as well [1]. However, if we break the topic apart into its individual components, the amount of literature available for review is massive. DevOps has been a major area of study in the last decade, with hundreds of papers explaining everything from its core principles and practices to its dimensions [2]. The same is true for CI/CD pipelines, albeit to a lesser extent, and several papers discussing everything from their use cases to evolution over the years [3]. There is a book on applying DevOps practices in the GCP with specific CI/CD pipelines (including Jenkins), which closely relates to this topic [4]. The foundational source for both CI/CD pipelines is their documentation and Google's own documentation, which discusses integration with both Jenkins and GitHub Actions. But there are papers as well. This includes papers and books discussing features and specific use cases of Jenkins [5], [6]. The literature on GitHub Actions is similar but more recent since it was released later. This includes papers that discuss its uses in software development repositories and empirical studies on the tool [7], [8].

GITHUB ACTIONS AND JENKINS AS CI/CD TOOLS

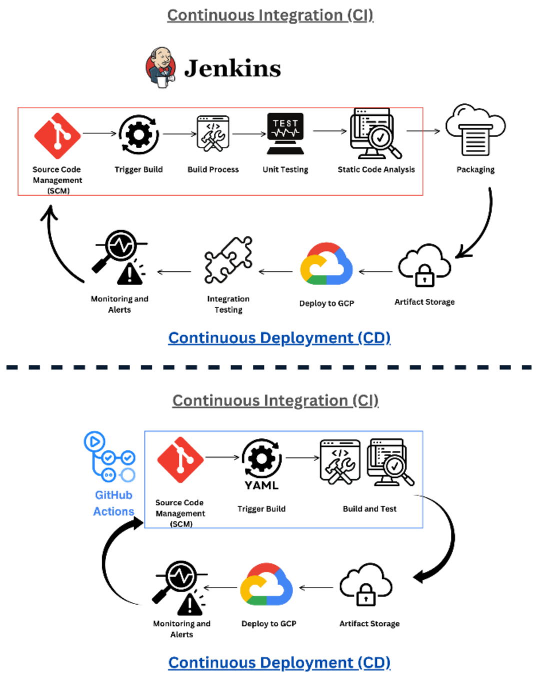

Figure 1: Jenkins vs. GitHub Actions CI/CD Flows

Jenkins, an open-source automation server, was first released in 2011. It is a Java-based platform widely used for implementing continuous integration and continuous delivery (CI/CD) pipelines. Jenkins is known for its flexibility, offering extensive plugin support that enables integration with various tools, including those in the GCP ecosystem. Its highly configurable nature allows users to tailor pipelines for complex workflows. While Jenkins excels in scalability and customization, it requires significant setup and maintenance effort, often necessitating dedicated resources. It is well-suited for organizations seeking a mature and extensible solution with minimal vendor lock-in.
GitHub Actions, launched in 2019, is a CI/CD solution integrated within the GitHub platform. It's not open source per se, but some "Actions" are. GitHub Actions provides a seamless way to automate workflows directly in repositories using YAML configuration files. Designed with simplicity and accessibility in mind, it supports building, testing, and deploying applications, including those targeting GCP. GitHub Actions benefits from its tight integration with GitHub, reducing the need for external tools and plugins. Its marketplace offers reusable actions to streamline setup for various environments. While GitHub Actions is beginner-friendly, its reliance on the GitHub ecosystem may present limitations for complex or multi-cloud workflows.

COMMON CI/CD CHALLENGES IN THE GCP ECOSYSTEM

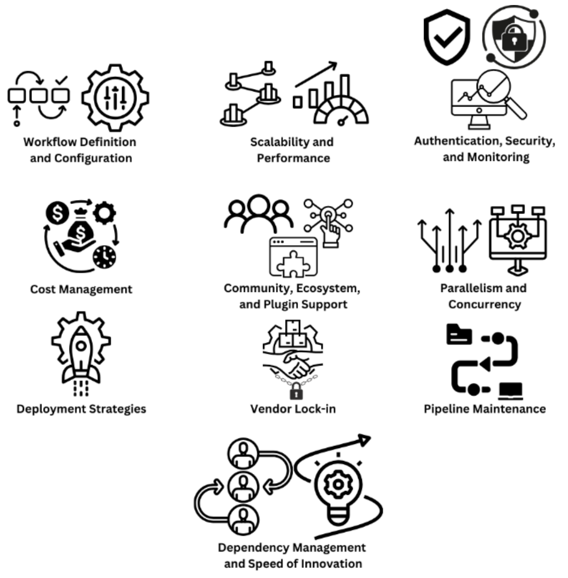
Workflow Definition and Configuration
Scalability and Performance
Authentication, Security, and Monitoring
Cost Management
Community, Ecosystem, and Plugin Support
Parallelism and Concurrency
Deployment Strategies
Vendor Lock-in
Pipeline Maintenance
Dependency Management and Speed of Innovation

Figure 2: Common CI/CD Challenges

Here are some of the challenges associated with CI/CD pipelines when integrating them with the GCP ecosystem. Many of them might be present in other cloud environments as well. It's worth noting that many of these challenges are essentially moot when you choose the right CI/CD pipeline, as we will discuss in the next segment.

Workflow Definition and Configuration: Challenges include ensuring compatibility with GCP-native tools and balancing simplicity with the complexity of multi-service orchestration. Declarative configurations must support GCP-specific requirements like IAM roles and region-specific deployments.

Scalability and Performance: Optimizing pipelines to handle burst workloads and large repositories while keeping resource costs manageable are among common challenges here. Inefficient resource allocation can lead to bottlenecks, especially when scaling workflows across multiple GCP zones or regions. Tools must adapt to both GCP's horizontal scaling capabilities and its usage-based cost model.

Authentication, Security, and Monitoring: Authentication challenges include securely managing GCP service account credentials and integrating with tools like Secret Manager. Security concerns escalate with multi-cloud or hybrid setups, where pipelines access external and internal GCP resources. Monitoring pipelines in GCP requires leveraging Cloud Monitoring and Logging, which must be configured to detect pipeline failures and anomalies effectively.

Cost Management: The pay-as-you-go model of GCP introduces challenges in tracking and controlling CI/CD-related expenses. Inefficient use of compute resources, such as long-running instances or underutilized VMs, can inflate costs.

Community, Ecosystem, and Plugin Support: While GCP offers robust native tools, integrating third-party CI/CD solutions can be challenging if plugin support or marketplace actions lack compatibility. Tools must cater to specific GCP services through well-maintained extensions or plugins.

Parallelism and Concurrency: Executing parallel jobs in GCP introduces challenges in optimizing compute resources like Kubernetes Engine or Compute Engine instances. Resource contention or misconfigurations can lead to failures or inefficient usage, impacting pipeline performance.

Deployment Strategies: Configuring pipelines to handle multi-region rollouts and ensuring rollback mechanisms work seamlessly with GCP services can be quite challenging.

Vendor Lock-in: Using tightly integrated CI/CD tools in GCP risks creating dependencies on GCP-native features, limiting flexibility to adopt multi-cloud or hybrid setups. This also impacts migrations to other environments.

Pipeline Maintenance: Maintaining pipelines in the GCP ecosystem requires frequent updates to accommodate changes in GCP services, APIs, or infrastructure.

Dependency Management and Speed of Innovation: Managing dependencies, such as container images and libraries, in GCP requires efficient use of services like Artifact Registry. Rapid changes in GCP tools or APIs pose challenges to keeping CI/CD pipelines up to date.

SOLUTION: GITHUB ACTIONS OR JENKINS?

Comparing the two different CI/CD pipelines against each challenge dimension can help you choose the right one for your GCP ecosystem.

Workflow Definition and Configuration

GitHub Actions offers simple, YAML-based workflow definitions directly within repositories. It provides reusable actions via its marketplace, enabling quick setup for GCP-specific tasks. The tightly integrated GitHub interface ensures streamlined configuration, making it beginner-friendly. However, it may lack flexibility for highly complex workflows compared to dedicated CI/CD solutions.

Jenkins provides unparalleled flexibility in workflow definitions, supporting both scripted and declarative configurations. Its extensive plugin ecosystem allows for seamless integration with GCP services. However, setting up workflows can be time-consuming due to its complexity. Advanced configurations often require significant expertise and manual effort to manage.

Scalability and Performance

GitHub Actions leverages GitHub-hosted or self-hosted runners, providing scalability for typical GCP workflows. While it handles moderate scaling well, resource limits for free-tier users can hinder performance under heavy workloads. For large-scale projects, users must manage runner capacities manually or invest in self-hosted solutions.

Jenkins is highly scalable and capable of distributing workloads across nodes using its master-agent architecture. It excels in handling large-scale projects but requires significant effort to optimize performance. Scalability relies heavily on the proper configuration of agents and underlying infrastructure, making it resource-intensive to manage.

Authentication, Security, and Monitoring

GitHub Actions integrates with GCP service accounts and Secret Manager, ensuring secure access to resources. Its native secrets management simplifies sensitive data handling. Monitoring, however, is limited compared to dedicated solutions, often requiring additional tools like Cloud Monitoring for detailed insights.

Jenkins supports robust authentication via plugins and integrates well with GCP IAM for secure access. Security management is more manual, requiring plugins for secret storage and role-based access control. Monitoring pipelines in GCP often necessitates third-party tools, adding to setup complexity.

Cost Management

GitHub Actions provides cost-efficient CI/CD for small projects, especially with its free-tier offering. However, heavy usage or large-scale GCP pipelines can incur significant costs, especially for additional runner capacity. Clear usage metrics help track expenses, but optimization is user-dependent.

Jenkins, being open source, has no direct licensing costs but can lead to high operational expenses. Infrastructure costs on GCP increase with complex setups involving numerous agents or plugins. Its scalability allows cost optimization but demands careful monitoring and configuration.

Community, Ecosystem and Plugin Support

GitHub Actions benefits from an active community and a growing marketplace of reusable actions. While its ecosystem is robust, it is smaller compared to Jenkins, with fewer advanced integrations for GCP-specific scenarios. Community support is strong but relies heavily on GitHub's ecosystem.

Jenkins boasts one of the largest plugin ecosystems, offering extensive integrations with GCP services. Its open-source nature encourages contributions, ensuring community-driven innovation. The ecosystem is mature, making it ideal for complex workflows, though plugin conflicts can arise, requiring expert management.

Parallelism and Concurrency

GitHub Actions supports parallel jobs and matrix builds, enabling efficient execution of concurrent tasks. However, limits on concurrent jobs for free-tier accounts can be restrictive. Advanced concurrency handling may require configuring custom runners or leveraging additional GCP resources.

Jenkins excels in parallelism, allowing unlimited concurrent jobs depending on infrastructure capacity. Its agent-based architecture enables efficient resource allocation for parallel tasks. However, achieving optimal concurrency often requires manual configuration and substantial infrastructure management.

Deployment Strategies

GitHub Actions supports GCP deployment strategies like canary and blue-green via marketplace actions and custom scripts. Its tight GitHub integration simplifies deployment workflows but may lack advanced features for multi-region or large-scale GCP deployments without additional customization.

Jenkins offers extensive support for advanced deployment strategies, including rolling updates and blue-green deployments. It integrates with GCP tools like Kubernetes Engine and Cloud Run via plugins. However, configuring such strategies can be complex, demanding significant expertise.

Vendor Lock-in

GitHub Actions is deeply tied to the GitHub ecosystem, increasing the risk of vendor lock-in. While workflows are transferable, their tight GitHub integration can limit portability. Users relying heavily on GitHub-hosted runners may face challenges migrating to alternative tools.

Jenkins is open-source and platform-agnostic, minimizing vendor lock-in. Its flexibility allows pipelines to integrate seamlessly with GCP and other platforms. This portability ensures users can adapt workflows for multi-cloud or hybrid setups without major constraints.

Pipeline Maintenance

GitHub Actions benefits from YAML-based workflows, making maintenance straightforward for small to medium pipelines. However, managing complex pipelines, especially with frequent GCP service changes, can become challenging. Versioning and debugging rely on GitHub's repository features, which are intuitive but limited.

Jenkins requires significant effort to maintain, especially for pipelines involving numerous plugins or custom integrations. Updates and compatibility issues can cause disruptions, requiring careful oversight. However, its modular nature allows pipelines to evolve alongside GCP services with proper management.

Dependency Management and Speed of Innovation

GitHub Actions integrates well with GCP Artifact Registry and dependency tools, streamlining builds. Its pace of innovation aligns closely with GitHub, introducing new features frequently. However, its dependency on GitHub's ecosystem can delay support for cutting-edge GCP advancements.

Jenkins excels in managing complex dependencies using plugins and custom configurations. Its open-source community ensures rapid adaptation to emerging tools and technologies. While innovation is community-driven, plugin reliance may lead to inconsistencies, requiring careful testing during updates.

CONCLUSION

Choosing between Jenkins and GitHub Actions for your CI/CD pipeline within the GCP ecosystem depends largely on the specific needs, resources, and context of your organization. Jenkins offers unmatched flexibility and scalability, making it ideal for complex, large-scale pipelines where advanced customization is a priority. On the other hand, GitHub Actions excels in simplicity, seamless GitHub integration, and ease of use, making it a strong contender for teams seeking streamlined workflows and faster adoption.

Your decision should consider factors such as the complexity of your workflows, budget constraints, available technical expertise, and your reliance on GCP-specific services. Organizations with established infrastructure and resources to manage and maintain pipelines might find Jenkins more suitable, while those prioritizing agility and quick setup may lean toward GitHub Actions. Ultimately, understanding your unique environment and aligning tool capabilities with your goals will ensure a CI/CD pipeline that maximizes efficiency, security, and scalability in your GCP ecosystem.

REFERENCES

[1]. S. T. Makani and S. Jangampeta, "THE EVOLUTION OF CICD TOOLS IN DEVOPS FROM JENKINS TO GITHUB ACTIONS," Int. J. Comput. Eng. Technol. IJCET, vol. 13, no. 02, Art. no. 02, May 2022.

[2]. L. Zhu, L. Bass, and G. Champlin-Scharff, "DevOps and Its Practices," IEEE Softw., vol. 33, no. 3, pp. 32–34, May 2016, doi: 10.1109/MS.2016.81.

[3]. F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study," in 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), Sep. 2021, pp. 471–482. doi: 10.1109/ICSME52107.2021.00048.

[4]. P. Riti, Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3897-4.

[5]. C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," in 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Jan. 2019, pp. 7–12. doi: 10.1109/CONFLUENCE.2019.8776985.

[6]. R. Leszko, Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines. Packt Publishing Ltd, 2022.

[7]. T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's Supercharge the Workflows: An Empirical Study of GitHub Actions," in 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), Dec. 2021, pp. 01–10. doi: 10.1109/QRS-C55045.2021.00163.

[8]. A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, "On the Use of GitHub Actions in Software Development Repositories," in 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), Oct. 2022, pp. 235–245. doi: 10.1109/ICSME55016.2022.00029.