Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th February 2026. Vol. 5. No. 2; e-ISSN: 2808-4519

Analysis of the Effectiveness of Manual Deployment and CI/CD Github Actions in the Braisee Application

Nenda Alfadil Seputra[1*], Odi Nurdiawan[2],Arif Rinaldi Dikananda[3] , Denni Pratama[4], Dian Ade Kurnia[5]

[1]*Informatics Enineering, STMIK IKMI Cirebon*
[2,4,5]*Information Management, STMIK IKMI Cirebon*
[3] *Software Engineering, STMIK IKMI Cirebon*
nendaseputra@gmail.com[1*], odynurdiawan@gmail.com[2], rinaldi21crb@gmail.com[3],
pratamadenni@gmail.com[4], dianade2012@gmail.com[5]

Abstract

In the modern cloud-based software development ecosystem, the speed and reliability of the deployment process are critical elements. This study aims to evaluate the effectiveness of implementing Continuous Integration/Continuous Deployment (CI/CD) using GitHub Actions compared to manual methods for the machine learning API of the Braisee application hosted on Google Cloud Run. Using a quantitative approach with a comparative experimental design across ten testing iterations, this research measures deployment time efficiency, error rates, and system stability. The experimental results show a significant performance disparity, where the automated method based on GitHub Actions is considerably more efficient, with an average total duration of 111–167 seconds, reducing operational time by 40–60% compared to the manual method, which requires 297–364 seconds. In terms of reliability, the automated method achieves a 100% success rate with high consistency, whereas the manual method demonstrates substantial vulnerability to human errors such as mistyped project IDs and inconsistent image tagging. It is concluded that implementing CI/CD through GitHub Actions is a superior solution that improves time efficiency and ensures the stability of cloud-based applications compared to manual procedures.

Keywords: *CI/CD, GitHub Actions, deployment, Cloud Run, efficiency.*

1. Introduction

The process of bringing software development results to the production environment is a crucial stage because it determines whether the value built by the engineering team can actually be utilized by end users. Many organizations still rely on manual deployment processes that involve repetitive steps and a high degree of operator dependency. This approach is prone to human error, ranging from inconsistent configurations to commands being executed differently between releases [1]. In addition, the use of manual procedures creates unpredictable variations in results between environments and increases the risk of release or rollback failures, even for small-scale updates [2].

These challenges are even more complex in the context of cloud-native architectures such as Google Cloud Run, which require precise container configuration to maintain environment parity. Even small deviations can cause configuration drift, a state in which the development and production environments are no longer identical, thereby reducing system reliability [3]. In applications operating in distributed and serverless architectures such as the Braisee Machine Learning API configuration inconsistencies can slow down the release process, complicate replication, and increase the workload for operations teams [4]. This complexity increases with the growing need for service orchestration and dependencies in modern cloud-native ecosystems [5].

As a solution to these various problems, the Continuous Integration and Continuous Deployment (CI/CD) approach provides automated, standardized, and consistent workflows. GitHub Actions, as a modern CI/CD platform, offers full integration with code repositories so that the build, testing, and deployment processes can be triggered automatically upon every code change. Various studies show that pipeline automation through CI/CD not only speeds up release times, but also improves reliability, accelerates feedback cycles, and improves coordination between development teams [6], [7]. Other studies have also found that the use of GitHub Actions positively affects the pull request process by reducing wait times and improving the consistency of code quality checks [8].

Although the benefits of CI/CD have been widely discussed, research that directly compares the performance of manual CLI-based deployment with automated deployment using GitHub Actions especially in medium-scale Machine Learning applications is still limited. Previous studies have either used simple prototypes or focused on other CI/CD platforms such as Jenkins or AWS CodePipeline [9], [10].

In addition, local studies discussing the effectiveness of CI/CD and manual methods are more often conducted on conventional web applications, so they do not fully reflect the complexity of container-based and serverless applications [11].

Therefore, this study aims to provide empirical contributions through direct analysis of the effectiveness of two deployment methods—manual and automated—in the context of cloud-native applications. Measurements focus on process time efficiency, vulnerability to errors, and service stability after the deployment process. The results of this study are expected to serve as a reference for developers and organizations in choosing the optimal deployment strategy, especially for cloud-based Machine Learning applications that require high release speed, continuous reliability, and strong environment consistency.

2. Research Metodology

2.1 Research Phase

This study uses a quantitative approach with a comparative experimental design (quasi-experiment). Two treatment groups were established: (1) The control group used the manual method via Google Cloud SDK (gcloud), and (2) The experimental group used GitHub Actions automation. Both methods were run on identical infrastructure environments to ensure the validity of the results.

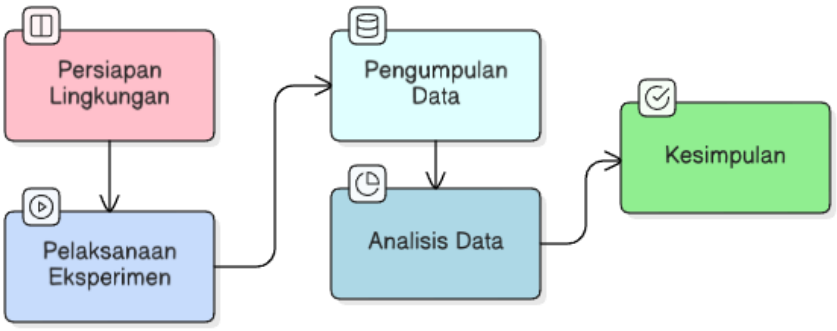

Fig. 1. Flow Of Reaseach Metodology

2.1. Preparation of the Experimental Environment

This research began with the preparation of the experimental environment, which aimed to ensure that the entire testing process ran under uniform and replicable conditions. At this stage, the FastAPI application environment was set up, the Machine Learning model was loaded, and all necessary dependencies were installed. In addition, a Dockerfile was created to ensure that the resulting containers remained consistent across each experiment. An automated pipeline using GitHub Actions was also configured, including Workload Identity Federation (WIF) authentication, the use of Docker Buildx, and a series of automated build, push, and deployment processes. For comparison purposes, a manual script utilizing the gcloud builds submit and gcloud run deploy commands is also prepared so that both methods can be tested fairly. This preparation stage results in a stable, uniform, and ready-to-use experimental environment for all testing iterations.

2.2. Experiment Implementation

The next step is to conduct experiments involving the execution of two deployment methods: manual and automatic. In the manual method, the build and push processes are performed using gcloud builds submit, while service implementation is carried out through gcloud run deploy. Meanwhile, the automatic method is run through GitHub Actions and triggered directly by commit or push activity to the main branch of the repository. Both methods are tested ten times to ensure consistency of results and reduce bias. The entire process, including build duration, deployment duration, execution status, and technical logs, was recorded in detail for further analysis. This stage produced a collection of log data representing the performance of both deployment methods.

2.3. Data Collection and Processing

The next step is data collection and processing. At this stage, data is taken from two main sources, namely GitHub Actions logs for the automated method and Cloud Build and Cloud Run logs for the manual method. Each piece of data is checked based on timestamp, process duration, and execution status. The synchronization process is carried out using run ID, commit SHA, or timestamp to ensure that each iteration can be mapped accurately. The data is then cleaned to remove anomalies, standardize the time format, and convert all information into tables or CSV files for easy analysis. The result of this stage is a quantitative dataset that is ready for computational processing.

2.4. Data Analysis

Data analysis was performed using a quantitative descriptive approach with the aim of objectively comparing the performance of the two deployment methods. Data was analyzed based on key variables, namely build duration, deploy duration, total process time, and success status. All durations were normalized to seconds to ensure consistency between data sets. Iteration sequence synchronization was also performed so that result patterns could be observed accurately. Next, the average value, duration variation, and time consistency patterns for both methods were calculated. Reliability analysis was also carried out by observing the success rate in each iteration, so that sources of error and failure trends could be identified. This analysis shows that the CI/CD method has higher stability and more efficient execution time compared to the manual method.

2.5. Conclusion

The final stage is drawing conclusions based on all research findings. The conclusions summarize the effectiveness of each method, particularly the differences in deployment time, process consistency, and reliability of results. These findings provide a clear picture of the advantages of the automated GitHub Actions method over the manual method, and form the basis for recommendations for developers in choosing the optimal deployment strategy.

3. Result

3.1. Comparison Analysis of Manual and CI/CD

a. Manual
In the manual deployment method, all data is collected directly from the terminal output resulting from the execution of two main commands, namely gcloud builds submit for the build process and pushing the image to Artifact Registry, as well as gcloud run deploy to apply the image as a new revision on Google Cloud Run. Since all steps are executed manually via the CLI, time and status variables are obtained from terminal logs without a fixed structure like those found in GitHub Actions.

Table 1. Manual *deployment method*

Run ID	Revision Name	SHA256 Digest	Build Time	Deploy Time	Total Duration	Total (detik)	Status
1	braisee-00015-b24	sha256:6241ec134f...	4 m 34 s	1 m 26.82 s	6 m 0.82 s	360.82	Success
2	braisee-00016-vzn	sha256:030ea84d9e...	4 m 47 s	53.06 s	5 m 40.06 s	340.06	Success
3	braisee-00017-rtm	sha256:31f17aadb1...	4 m 37 s	52.01 s	5 m 29.01 s	329.01	Success
4	braisee-00018-9tr	sha256:27b75fb41c...	4 m 29 s	1 m 15.287 s	5 m 44.287 s	344.287	Success
5	braisee-00019-q98	sha256:27b75fb41c...	4 m 29 s	28 s	4 m 57 s	297	Success
6	braisee-00020-shb	sha256:7785463dd8...	3 m 44 s	1 m 30 s	5 m 14 s	314	Success
7	braisee-00021-mzl	sha256:904afef637...	5 m 2 s	53 s	5 m 55 s	355	Success
8	braisee-00022-kzp	—	—	—	—	—	**Failure**
9	braisee-00023-j65	sha256:904afef637...	5 m 2 s	17 s	5 m 19 s	319	Success
10	braisee-00024-zf8	sha256:8ce4e274cf...	4 m 5 s	1 m 59 s	6 m 4 s	364	Success

In the manual method, the deployment process takes between 297 and 364 seconds, with an average of 330.5 seconds. This wide range indicates that the manual method has a high degree of variability. These fluctuations in duration are mainly influenced by differences in local environmental conditions, such as network performance, system load, and the devices used by operators. In addition, reliance on command execution via CLI makes the process highly susceptible to human error, whether in the form of typing errors, inconsistent parameter settings, or delays in executing each step sequentially. These factors result in unstable and unpredictable execution durations.

b. CI/CD
In the automated deployment method, all performance data is obtained from each GitHub Actions workflow run that is executed when there is a push or commit to the main branch. GitHub Actions provides structured, consistent logs that automatically record each stage of the workflow, so that data extraction can be done easily and accurately. Based on ten successful workflow iterations that were collected, each run produced four main variables that were used as the basis for performance analysis.

Table 2 CI/CD *automated deployment method*

ID	Commit	Waktu Build (Detik)	Waktu Deploy (Detik)	Total Workflow (Detik)	Status
1	ac4af4d	96	28	162	Success
2	c074bef	59	29	126	Success
3	da520a7	96	28	167	Success
4	3d89507	61	31	131	Success
5	e8fe1cd	78	33	140	Success
6	b97b365	37	38	111	Success
7	b321331	48	43	117	Success
8	cc378f4	37	37	111	Success
9	e842586	55	35	131	Success
10	41cd034	37	37	111	Success

In the automated method, the deployment process takes between 111 and 167 seconds, with an average of 139 seconds. This duration shows a significant increase in efficiency compared to the manual method. In addition to being faster, the execution time is also much more consistent. This consistency is influenced by the standardized GitHub Actions runner environment, so that each stage in the workflow is executed under the same conditions without depending on local device performance. The use of caching mechanisms in Docker Buildx also plays a role in speeding up the build process, as unchanged image layers do not need to be rebuilt on each iteration. Overall, the resulting stability in time demonstrates that the automated pipeline is capable of providing a more deterministic, predictable, and minimally variable deployment process.

Based on ten iterations of experiments that have been conducted, a series of deployment duration data from manual and automatic methods was obtained. The measurement results provide an initial overview of the performance differences between the two approaches and are summarized visually in the figure 1.

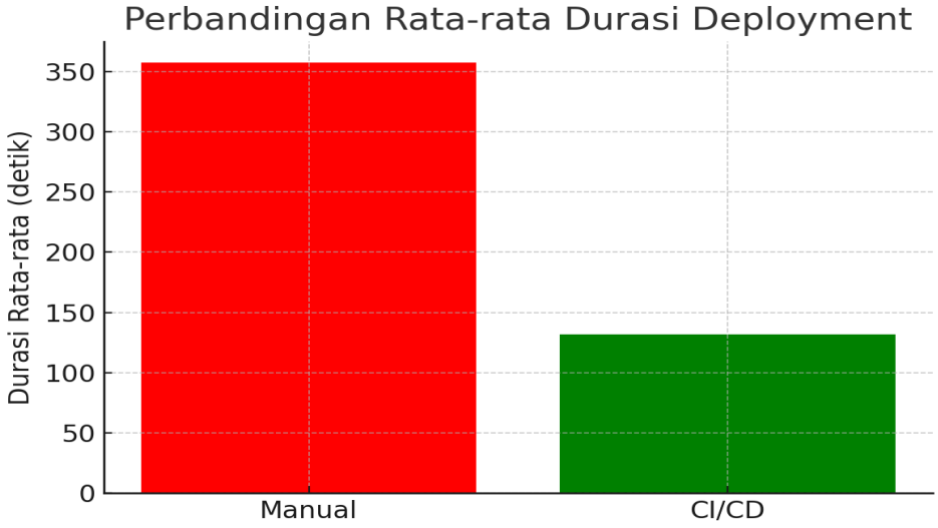

Fig .2. Deployment Duration Comparison

The data shows that the automated method is much faster than the manual method. In addition, incidents of failure were found in the manual method due to human error, as seen in the system error log.

The significant difference between the two methods occurs because the entire build and deployment process on GitHub Actions runs automatically without operator intervention. This automation eliminates the need to manually type commands, verify parameters one by one, or wait for processes interactively, making execution time much more efficient. In addition, the YAML workflow on GitHub Actions is deterministic, ensuring that each step is executed in the same order and configuration on each iteration, minimizing process variability. The use of Docker Buildx also provides additional benefits through a caching mechanism that reduces build time, especially when there are no significant changes to the image layer. The combination of these factors enables the CI/CD process to achieve deployment speeds that are up to 60% faster than manual methods.

3.2. Reliability Analysis

A comparison of consistency and reliability between the two methods shows that automated deployment via GitHub Actions performs more stably than the manual method. The manual process is highly dependent on operator accuracy and local device conditions, making it prone to input errors such as incorrect Project ID entry, inappropriate image tagging, or inaccurate deploy commands. These variations cause significant differences in duration and some failures in certain iterations.

In contrast, the automated method is characterized by high reliability with a 100% success rate across all iterations. The structured YAML workflow ensures that every step—from authentication, build, push, to deploy—is executed identically on each execution. The uniform GitHub runner environment also eliminates the variations that typically arise on local devices. As a result, process duration is more consistent, errors can be minimized, and process reproducibility is improved.

Thus, CI/CD is not only more consistent in duration but also more reliable in the overall execution of the deployment process, making it a more dependable method compared to manual approaches.

4. Conclusion

Based on the results of experiments and analyses that have been conducted, it can be concluded that the automatic deployment method using GitHub Actions provides consistently superior performance compared to the manual method. The most striking difference is seen in terms of time efficiency, where the automatic approach is able to speed up the deployment process by 40–60% compared to the manual

method. This speed does not only come from a more optimized build process through the Docker Buildx caching mechanism, but also from a fully automated pipeline execution flow that eliminates delays caused by human intervention.

In addition to time efficiency, the reliability of the automated method also shows excellent performance. All iterations on GitHub Actions run with a 100% success rate, while manual methods are still vulnerable to various forms of human error such as command input errors, inconsistencies in configuration, and operator delays. This confirms that automation not only improves speed but also the stability of the deployment process.

Automated pipelines also provide advantages in terms of stability and reproducibility. Each step in the YAML workflow is executed in a consistent runner environment, resulting in a predictable and uniform process for each iteration. In contrast, manual processes are highly influenced by network conditions, operator accuracy, and local device conditions that can change. This difference shows that GitHub Actions is capable of providing a more deterministic and easily replicable deployment process.

Overall, the results of this study confirm that GitHub Actions is a more effective, reliable, and suitable deployment method to support the needs of applications running in cloud environments, especially Machine Learning services that demand release speed, configuration accuracy, and high consistency. These findings provide a strong basis for developers and organizations to adopt the CI/CD approach as a key strategy in managing the modern deployment cycle.

References

[1] I.-C. Donca, M. Misaros, D. Gota, and L. Miclea, "Method for continuous integration and deployment using a pipeline generator for Agile software projects," *Sensors*, vol. 22, no. 12, p. 4637, 2022, doi: 10.3390/s22124637.
[2] B. Erdenebat, B. Bud, T. Batsuren, and T. Kozsik, "Multi-project multi-environment approach—An enhancement to existing DevOps and CI/CD tools," *Computers*, vol. 12, no. 12, p. 254, 2023, doi: 10.3390/computers12120254.
[3] J. Dobaj, A. Riel, G. Macher, and M. Egretzberger, "Towards DevOps for cyber-physical systems facilitated by digital twins," *Machines*, vol. 11, no. 10, p. 973, 2023, doi: 10.3390/machines11100973.
[4] L.-N. Lévy, J. Bosom, G. Guerard, S. B. Amor, M. Bui, and H. Tran, "DevOps model approach for monitoring smart energy systems," *Energies*, vol. 15, no. 15, p. 5516, 2022, doi: 10.3390/en15155516.
[5] J. Lin, D. Xie, J. Huang, Z. Liao, and L. Ye, "A multi-dimensional extensible cloud-native service stack for enterprises," *J. Cloud Comput.*, vol. 11, p. 83, 2022, doi: 10.1186/s13677-022-00366-7.
[6] A. D. Setyoko and A. Zahra, "Perbandingan efisiensi proses CI/CD multi-lingkungan melalui implementasi paralel dan berurutan," *MALCOM Indones. J. Mach. Learn. Comput. Sci.*, vol. 4, no. 3, pp. 911–925, 2024.
[7] A. M. Buttar, "Optimization of DevOps transformation for cloud-based applications," *Electronics*, vol. 12, no. 2, p. 357, 2023, doi: 10.3390/electronics12020357.
[8] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, "GitHub Actions: The impact on the pull request process," *Empir. Softw. Eng.*, vol. 28, p. 131, 2023, doi: 10.1007/s10664-023-10369-w.
[9] R. Bagai, A. Masrani, P. Ranjan, and M. Najana, "Implementing continuous integration and deployment (CI/CD) for machine learning models on AWS," *Int. J. Glob. Innov. Solut.*, 2024.
[10] A. Farid and I. Gita Anugrah, "Implementasi CI/CD pipeline pada framework Androbase menggunakan Jenkins (studi kasus: PT. Andromedia)," *J. Nas. Komputasi dan Teknol. Inf.*, vol. 4, no. 6, 2021.
[11] R. Setiabudi, "Analisis efektivitas CI/CD dan manual (tradisional) dalam pengembangan website Rakyatweb.com," *ISMETEK*, vol. 17, no. 2, 2024.