Continuing to Catch Up with State of the Art Continuous Integration Pipelines in Palladio – The Experience Report Strikes Back

Lucas Domingo Alber
lucas.alber@student.kit.edu
Karlsruhe Institute of Technology

Nicolas Boltz
boltz@kit.edu
Karlsruhe Institute of Technology

Larissa Schmid
schmid@kit.edu
Karlsruhe Institute of Technology

Abstract

The Palladio organization comprises over 50 software artifacts and is actively developed by over 32 members. This research paper presents a case study on the migration from Jenkins to GitHub Actions for automated builds in the development workflow for Palladio. The transition has yielded significant improvements in continuous integration, review processes, and deployment efficiency. The adoption of GitHub Actions' modular and reusable workflows has further optimized our build pipeline, resulting in enhanced maintainability and reduced redundancy. Additionally, by leveraging dependency analysis, we applied the idea of incremental builds to the whole organization and automated the generation of build workflows, leading to improved resource utilization and an average speed-up in build times of 11.7. This study highlights the benefits of embracing GitHub Actions and provides valuable insights for development teams seeking to streamline their build processes.

1 Introduction

In today's fast-paced software development landscape, ensuring code quality, continuous integration, and swift deployment has become crucial for teams seeking to deliver reliable products to their users. Among the various methodologies employed, we use nightly builds in Palladio [1] to enable continuous testing and integration, allowing developers to identify and address issues early in the development cycle. Employing automated build systems such as Jenkins [1] has been a common approach to managing nightly builds. These systems can be directly integrated with version control services such as GitHub using webhooks. However, this involves substantial maintenance overhead.

As the technology landscape continually evolves, GitHub Actions [2] has emerged as an automation platform, revolutionizing the way teams orchestrate their development workflows. Recognizing the limitations and management complexities associated with maintaining a Jenkins-based setup, we aimed to transition our builds to GitHub Actions. We re-engineered a preliminary build pipeline based on GitHub Actions that mimicked the build in Jenkins. In the process, we identified further points of improvement, e.g. that organization-internal internal dependencies could be calculated dynamically, that new projects could be included in the nightly build automatically, and that builds could be done incrementally based on changes in the repositories. Additionally, GitHub Actions' reusable workflows provided us with the flexibility to modularize different stages of our build, leading to enhanced maintainability and reduced redundancy.

In this research paper, we share our reasons for the migration, the benefits realized in continuous integration and deployment efficiency, and the implications on sustainability and modularization. We present how we use dependency analysis in Palladio to harness the potential of modern automation platforms and drive sustainable development practices. To that end, we apply the idea of incremental builds to the whole organization, automatically generating build workflows and reducing build executions.

2 Switching to GitHub Actions

The Palladio organization consists of over 50 projects that need to be kept up to date by the nightly build. To make all projects reflect the latest changes of their dependencies, the nightly build workflow must build the projects according to their complex dependency relationship. The following sections describe issues we encountered with the old build system, how our updated system overcomes these issues and the new infrastructure setup based on GitHub Actions.

2.1 Build Pipeline Stages

Previously, the build dependencies were declared manually by editing the build definition. This time-consuming and error-prone process made adding projects to the nightly build hard and could result

[1]`https://www.jenkins.io/`
[2]`https://docs.github.com/en/actions`

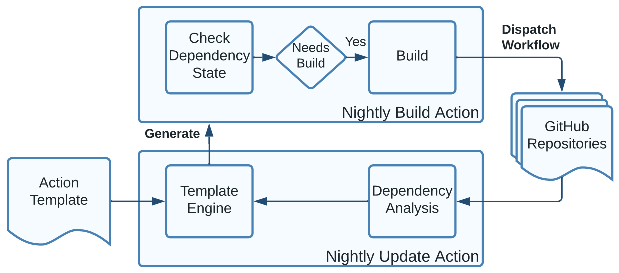

Figure 1: Overview of the new incremental CI/CD pipeline for nightly builds in Palladio.

in missing, as well as redundant, dependencies. Missing dependencies did not only lead to build failures because of version mismatches but made it difficult to determine the origin of a failure since the build would fail one iteration later, for example when the API of a dependency changed. This is because the dependency could be built and deployed after the project which depends on it, and the changes would only be visible in the next iteration. Redundant dependencies could reduce the maximum concurrency of builds or even break the build system if a cycle was introduced.

The updated system overcomes these issues by dynamically generating and maintaining the build definition with minimal user intervention. An overview is illustrated in Figure 1. A *Nightly Update* action scans the organization periodically for new or changed projects that are configured to be part of the nightly build[3]. For those projects, the dependencies are calculated directly from the project files. Using an extensible template engine, the dependencies are then turned into a GitHub workflow file for the *Nightly Build* action. Each project is translated to a job in the workflow file, and the project dependencies are translated to job dependencies. This way, GitHub automatically orchestrates the builds such that all projects are built in the correct order and concurrency is maximized. An excerpt of the Palladio dependency graph as displayed in the GitHub Actions overview can be found in Figure 2. If the definition changes, a Pull Request is opened on the build repository and maintainers are notified for a review of the changes.

Projects opt-in to the nightly build by simply providing a build workflow in their repository. If necessary, additional projects and build steps can be added manually using a template file. The Nightly Build action dispatches the CI workflow of each repository by a remote workflow script that waits for the workflow to finish and mirrors the conclusion to the nightly build. This keeps the build flexible and project-agnostic, which is advantageous for project maintainers because the build workflow can be updated and customized independently. Additionally, the workflow status on the repository represents the build status after the last dependency update rather than the status of the last

[3]`https://github.com/PalladioSimulator/Palladio-Build-DependencyTool`

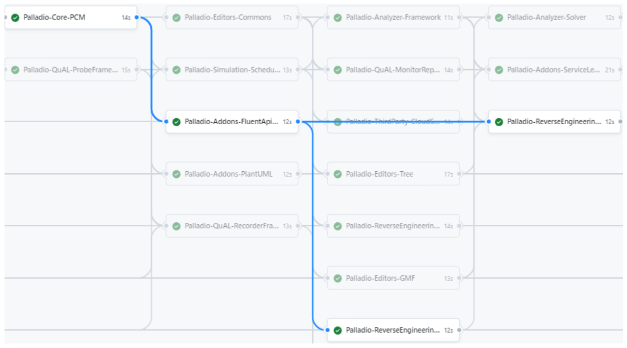
Palladio-Core-PCM 14s
Palladio-Editors-Commons 17s
Palladio-Analyzer-Framework 11s
Palladio-Analyzer-Solver 12s
Palladio-QuAL-ProbeFrame... 15s
Palladio-Simulation-Schedu... 13s
Palladio-QuAL-MonitorRep... 14s
Palladio-Addons-ServiceLe... 21s
Palladio-Addons-FluentApi... 12s
Palladio-ThirdParty-CloudS... 14s
Palladio-ReverseEngineerin... 12s
Palladio-Addons-PlantUML 12s
Palladio-Editors-Tree 17s
Palladio-QuAL-RecorderFra... 13s
Palladio-ReverseEngineerin... 14s
Palladio-Editors-GMF 13s
Palladio-ReverseEngineerin... 12s

Figure 2: Excerpt of the dependency graph visualization of GitHub for the Palladio organization.

push, and maintainers are notified about issues during the nightly build.

2.2 Scalable Infrastructure Setup

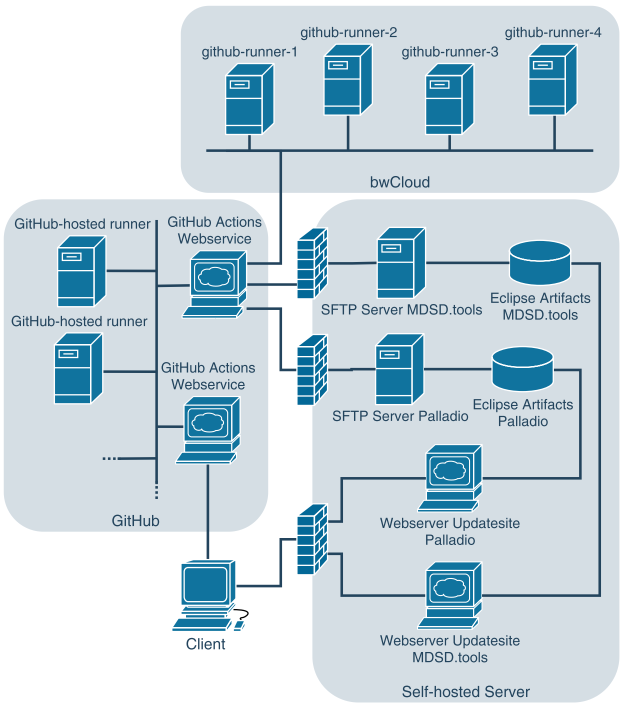

Figure 3: Overview of the current CI/CD infrastructure for the Palladio organization.

Our previous setup was based on Jenkins and integrated into GitHub using webhooks [2]. While this setup fulfilled our functional requirements of a build system, it turned out to be unreliable in practice and involved major management overhead. For example, the Jenkins server and multiple build agents needed to be kept up to date. Adding new repositories required expertise in both Jenkins and GitHub to set up the build plan and the required webhooks.

For the updated build system, the builds are orchestrated by the GitHub web service, as described in Subsection 2.1. The builds themselves are executed on shared runners managed by GitHub as well as self-hosted runners which provide a minimum user interface for GUI tests that are hosted on bwCloud[4], an IaaS platform for science and education. See Figure 3 for an overview of the new setup.

The build artifacts are deployed to a self-hosted Eclipse updatesite[5]. The updatesite consists of a database, an SFTP server, and a web server behind a firewall, which are deployed using Docker containers.

Management overhead is reduced since GitHub provides a highly available, reliable, and well-documented platform that is easy to understand and which developers can quickly familiarize themselves with. A new project is integrated by adding a build definition to the repository. We further simplify this process by providing a *reusable workflow* for common project layouts, which builds and deploys the project. The reusable workflow provides a sensible default configuration that can be overwritten by project maintainers, for example, to select the required runner by label (GitHub or self-hosted) and Java version.

We make use of the tight integration of GitHub Actions with the platform by sharing secrets across the organization, extensive use of user and rights management as well as automated pull request reviews.

3 Incremental Build Schedule

The nightly build harnesses the potential of modern automation platforms to keep the build sustainable by using the dependency information not only to build the projects in the correct order but also to determine if a build is necessary at all. Therefore, we apply the idea of incremental builds to the whole organization. A build can be skipped if none of the dependencies were updated and the last build on the repository started after the latest commit. As a build is also dispatched after a push as part of most project-specific CI/CD workflows, the latter is often already the case. As a result, the build times, as well as resource consumption, are significantly reduced. Overall, this should lead to reduced energy consumption.

In the Palladio organization, we analyzed 24 regular nightly builds and 27 incremental nightly builds after filtering outliers caused by infrastructure outages. We were able to reduce the average build times from 139.5 minutes down to 11.9 minutes, resulting in an average speed-up in build times of 11.7. Figure 4 presents the distribution of workflow durations for regular full builds compared to the described incremental and more sustainable build schedule.

4 Conclusion

The migration from Jenkins to GitHub Actions for our nightly builds addressed challenges with our previous setup and has proven to streamline our software development workflows. We have observed improvements in continuous integration, review processes, and

[4]https://www.bw-cloud.org

[5]https://updatesite.palladio-simulator.com/

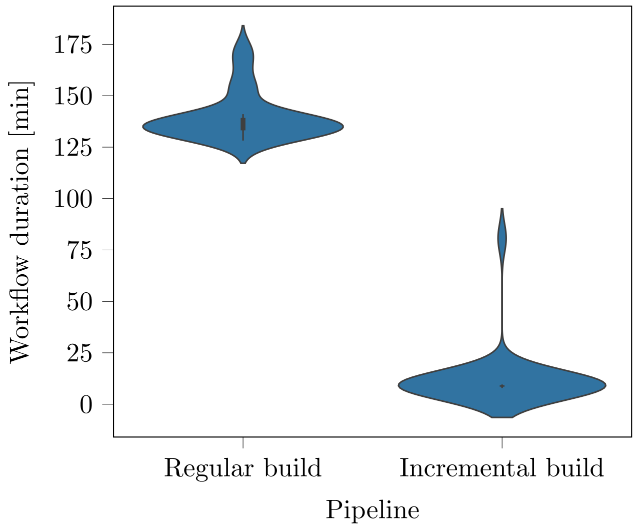

Figure 4: Distribution of workflow durations for regular and incremental builds.

deployment efficiency, thanks to the seamless integration within the GitHub ecosystem. Our build pipeline was optimized using GitHub Actions' modular and reusable workflows, leading to enhanced maintainability and reduced redundancy in our projects.

By leveraging dependency analysis, we have further unlocked the potential of modern automation platforms, generating build workflows and reducing build execution frequency. This data-driven approach has not only improved resource utilization but also contributed to sustainable development practices.

Throughout this process, we have also gained additional valuable insights into the benefits of embracing GitHub Actions. With GitHub Actions, we have found a robust and future-proof automation platform that has met our expectations.

Acknowledgements

This publication is partially based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. This work was also supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Az: 7712.14-0821-2), and by funding from the pilot program Core Informatics of the Helmholtz Association (HGF).

References

[1] R. H. Reussner et al., eds. *Modeling and Simulating Software Architectures – The Palladio Approach.* MIT Press, 2016. 408 pp.

[2] S. Seifermann and S. Krach. "Catching Up with State of the Art Continuous Integration Pipelines in Palladio-An Experience Report." In: *Softwaretechnik-Trends* 40.3 (2020), pp. 52–54.