Kamari Fuller
CSC 325
26 April 2024

CI/CD Pipeline Report

Introduction

Continuous Integration and Continuous Delivery (CI/CD) pipelines play a pivotal role in modern software development by automating the processes of building, testing, and deploying applications. In building this pipeline I wanted to implement simple, efficient, reliable, and ultimately free systems. This report outlines the implementation of a CI/CD pipeline for a Flutter web application, focusing on its setup, tools utilized, and deployment strategies.

DevContainer Environment

For the development environment, Docker containers were utilized, offering a standardized setup for Flutter development. Docker's configuration features facilitated the creation of a consistent environment, ensuring that developers have access to the same tools, dependencies, and operating system across different machines. In a team setting this approach would streamline collaboration, eliminating the common issue of "works on my machine" discrepancies. However, I found it beneficial in my solo work as well. Allowing me to work on various projects on my personal machine without switching the configuration as I switch projects, only the container I'm using.

Version Control Integration

The project leveraged GitHub as the version control system due to its widespread adoption and robust feature set (outside of strictly version control). GitHub provided a centralized cloud platform for storing code, managing changes, and facilitating collaboration in a team setting. The Git workflow (GitHub Flow), particularly the branch-based development model, is a development paradigm provided to organize code changes and ensure a systematic approach to version control. This integration with GitHub formed the foundation for the CI/CD pipeline, enabling automated testing, building, and deployment processes.

CI/CD Pipeline Configuration

The CI/CD pipeline was configured using GitHub Actions, which automates the software development lifecycle from code commit to deployment. The pipeline consisted of multiple stages, including triggering workflows on code changes, executing build and test jobs, and deploying the application. GitHub Actions provided a seamless integration with GitHub repositories, allowing for efficient workflow configuration and execution. Notifications via email were set up to alert developers in case of job failures, ensuring prompt resolution of issues.

CI/CD Tools Utilization

GitHub Actions served as the primary CI/CD tool, offering features such as triggers, jobs, checkout, artifacts, and deployment capabilities. The pipeline workflow was defined to trigger on any code push, initiating the build, test, and deployment processes automatically. Utilizing GitHub Actions streamlined development workflows, reducing manual intervention, and enabling faster feedback loops. The integration with GitHub Pages facilitated deployment of the Flutter web application, allowing for easy hosting of static assets.

Flutter Web Application

The Flutter web application developed for this project was based on a simple template provided by Flutter, emphasizing functionality over complexity. The application features basic slider games, offering users an engaging experience while showcasing Flutter's capabilities for web development.

While the web application itself serves as a demonstration of Flutter's potential, the primary focus of the project was on the CI/CD pipeline implementation. Recognizing the niche nature of Flutter and Dart compared to broader programming ecosystems, the emphasis was placed on leveraging CI/CD knowledge to streamline development processes. By prioritizing the pipeline's setup and automation, the project aimed to demonstrate the transferable value of CI/CD expertise across diverse software projects, beyond the specific technology stack of Flutter and Dart.

Deployment Environment:

Deployment of the Flutter web application was achieved using GitHub Pages, a hosting service provided by GitHub. The GitHub runner in the deploy job of the CI/CD pipeline was configured to push the built assets to a dedicated branch called 'gh-pages', which GitHub Pages then served as a static website. This deployment strategy simplified the hosting process, eliminating the need for external hosting services and ensuring seamless integration with the CI/CD pipeline.

Conclusion:

Implementing the CI/CD pipeline for the Flutter web application presented challenges in DevContainer and workflow configuration. Overcoming unreliable builds on Apple Silicon required updating Rosetta, while consolidating workflows into a single file resolved deployment issues. Professionally, this project enhanced familiarity with development systems, emphasizing the importance of standardized environments, robust source control, and efficient automation. Ultimately, navigating these challenges not only delivered a functional application but also cultivated valuable insights and skills for future endeavors in software development. In the future I could adhere to the GitHub Flow paradigm, optimize my workflows by adjusting what triggers it, and of course build upon my basic Flutter Web App.