Nathan Johnson

CSC325

Project Report

Introduction:

Continuous Integration/Continuous Deployment, or CI/CD pipelines are used in automating the software delivery process. They allow developers to integrate code changes, automate testing, building, and deployment to production environments. CI/CD pipelines are important in modern software development because they allow developers to automate testing, building, and deployment, making the development cycle more efficient. For this project, we implemented a CI/CD pipeline using Docker, GitHub, and GitHub Actions to create and deploy a basic Flutter app.

DevContainer Environment:

In our project, we used a Docker container running on Ubuntu to create a consistent dev environment. Our devcontainer.json file contains the necessary dependencies and tools required for Flutter development, including the VSCode extension required for Flutter development. The Dockerfile defines our workspace as being the latest version of Ubuntu, and includes the configurations for updating and installing OpenJDK, then cloning the Flutter repository, allowing us to work with the Flutter SDK.

Source Code Version Control Tools:

We used Git with GitHub as our VCS for this project, as it is what we personally are most familiar with. Using GitHub enables seamless integration with the build and deploy stages of the

CI/CD pipeline, which we will discuss later, as well as easy access to older versions of our codebase. We used GitHub to manage code changes, as well as using integrated GitHub Actions to automate our pipeline.

CI/CD Pipeline Environment:

Our CI/CD pipeline employs workflows with GitHub Actions for customizable CI/CD capabilities integrated directly with GitHub for ease of automation. The pipeline environment includes stages for building, testing, and deploying our Flutter application, which we will discuss under CI/CD Tools.

CI/CD Tools:

As stated before, we used GitHub Actions as the CI/CD tool for this project. It is seamlessly integrated with our GitHub repository, So we were able to create workflows using .yml files directly in the repository, under the .github/workflows directory. GitHub Actions allowed for us to use predefined actions made by other users, which we took advantage of in our deployment phase. Our workflows are triggered automatically on code pushes and pull requests to our main branch, allowing them to run when any file is updated, which ensures that if any code changes cause a failed test, we know immediately. We used GitHub Actions for everything in our build, test, and deploy stages. We used subosito's flutter-action@v2 Action to clone into Flutter on the runner, and ran the "flutter test" command to do a basic smoke test for our testing stage. Our deployment stage uses peaceiris' GitHub Pages action to deploy via a gh-pages branch, separate from our main branch.

Deployment Environment

For our deployment environment we used GitHub Pages, because of its seamless integration with GitHub and GitHub Actions. It is simple and free to use, making it the ideal candidate for a short-term project like this one. We didn't have to do any extreme setup configurations, as peaceiris' GitHub Pages action worked almost immediately for deployment. We did have to enable Jekyll, a popular site generator compatible with GitHub Pages, but after that, the site published largely without issue.

Flutter Web Application

Our Flutter application is a basic button press counter. There is a single widget that increments a counter every time it is clicked. We chose to do this for ease of testing, as this was our first encounter with Flutter and we expected errors to occur. The simplicity of the app mitigated the amount of errors we received, and made these errors easy to diagnose and counter. We test it via the "flutter test" command, as discussed earlier. If that succeeds, the app builds using "flutter build web," which automatically formats it for a web page. The application then deploys to our GitHub Pages site through peaceiris' GitHub Pages Action. One issue we are still having is that our GitHub Pages site does not properly display the application, despite our tests, build phase, and deployment phase succeeding. However, the application works perfectly fine when run in Edge through VSCode using the same series of commands.

Conclusion

In conclusion, the CI/CD pipeline project was a learning experience. We gained experience with GitHub at an in-depth level, and worked with Flutter to develop a web-app whose build and deployment processes we automated using the CI/CD pipeline. Challenges encountered include our own tendency to procrastinate, and the GitHub Pages display issue as mentioned before. Potential improvements include condensing the build and deploy workflows into a single .yml file, and fixing the aforementioned display issue. Overall, our pipeline is functional and succeeds in its purpose of automating web application development.