DevOps HW3

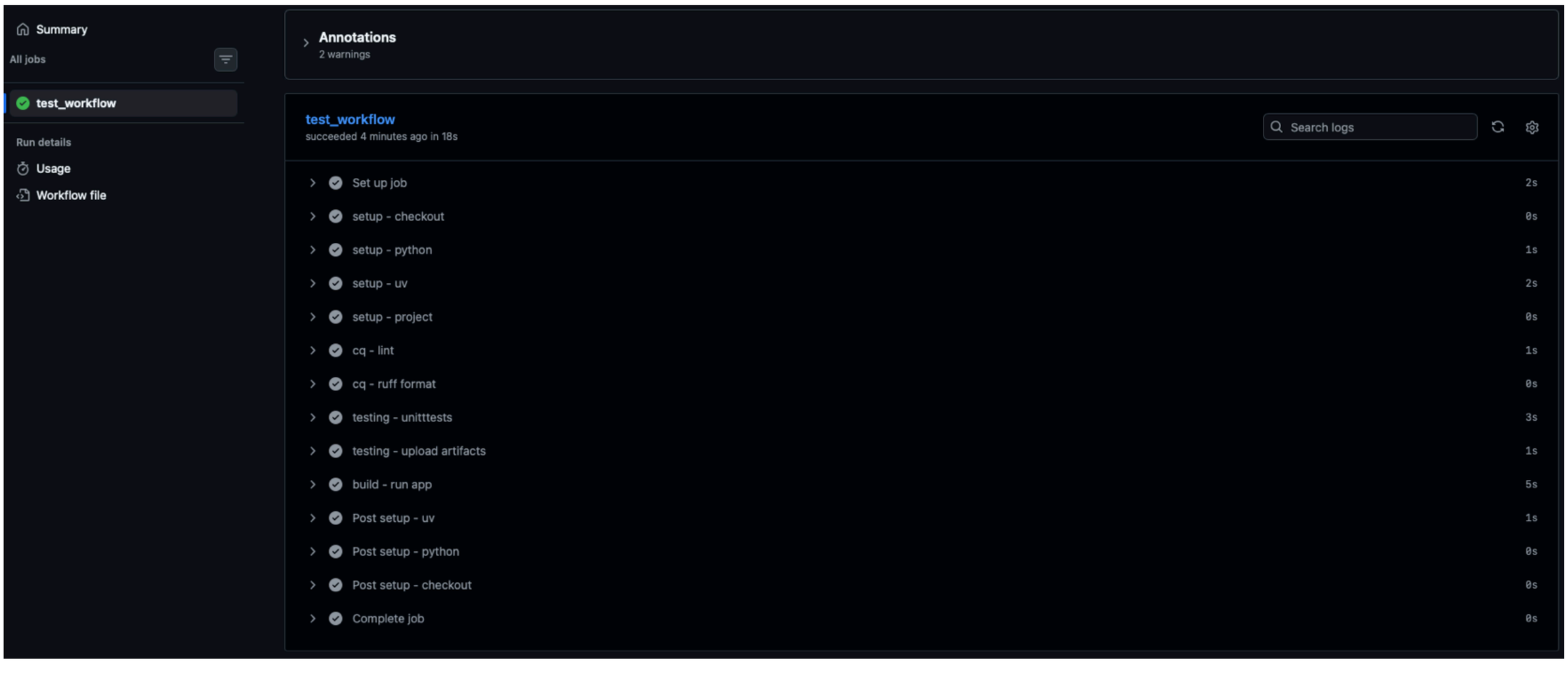
Pablo Hernandez Pedraza

Q1. Jenkins vs. Github Actions vs Azure DevOps

Tool	Pricing Model	Setup Ease	Integration	Features	Use Cases
Jenkins	Self-hosted: Free	Manual setup and ongoing maintenance	Manual integration	Large plugin ecosystem, highly customizable	Legacy or complex pipelines
GitHub Actions	*Usage-based*: - Self-hosted: Free - Cloud-hosted: Public repos free, Private repos $0.002–0.062 USD/min	Easy	Native GitHub integration, Actions Marketplace	YAML-based workflows, reusable actions	GitHub-oriented development, small to mid-scale projects
Azure DevOps	*Fixed*: - Self-hosted: First pipeline free, then $15USD/pipeline/month < br > -Cloud-hosted$: $Firstpipelinefree, then$ 40 USD/pipeline/month	Moderate	Integration with Azure and Microsoft ecosystem	Part of a broader DevOps suite	Enterprise and Azure-based projects

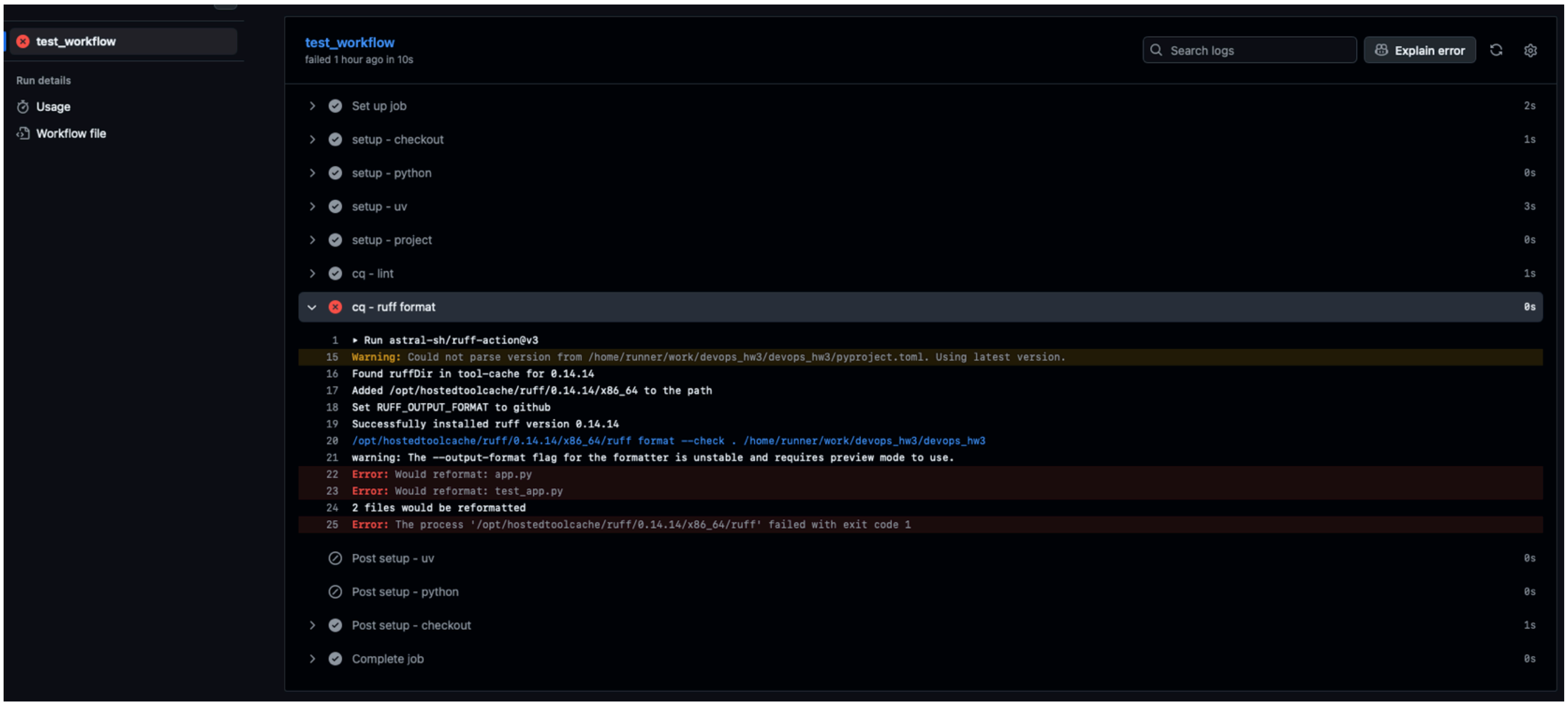
Q2. Web Application

Here is an example of a sucessful run through the CI steps (setup, cq, testing and build):



To verify that each of the tasks is well defined and executes successfully, please refer to the trigger definitions on the yaml file header and the workflow history.

Here is an example of an older version of the workflow that was failing because the formatter identified improvements to the code quality that were configured to be fixed only locally:



Q3. Reflection

The main challenge surrounding this CI implementation was the design choice of developing and testing out the application locally rather than containerizing it, which in turn left more loose ends in terms of configuration that I then had to tinker with during the process of writing each instruction on the .yaml file. While I had worked with GitHub Actions before and knew that a lot of the work could be configured by importing pre-written actions from the marketplace, but struggled with some of the expected shell behavior and the right approach to gating the code to enforce sucess/failure behavior. I also did not have prior knowledge of artifacts and found it rather confusing to draw the line between them and actual data being used or modified by the main script during run time.

I think that the steep learning curve implied in learning and experimenting with CI configurations is mostly compensated by the productivity improvements once things are working correctly. It is a powerful tool to take an idea a step further and ensure that it evolves in a way that is impactful and future-proof. It promises to help me give shape to projects through minimum quality standards and layered development so that the behemoth that is concieving a code-based solution can be modularized and tackled strategically.

With more time, I would opted for a more complex architecture enhancing my work from the previous assignment to take advantage of containerization and data management features while the convenience of automated checks associated with CI tools. With a containerized approach, it would have been easier to run the build process in the background and ping the live website and create a set of build artifacts rather than the current approach, which consists of forcing a timeout gracefully without any particularly useful build artifacts. I would also be curious on stress-testing the pipeline using alternative run-time configurations, as well as attempting to translate this setup to the other tools mentioned in question 1 to benchmark performance and cost and validate which tool would be better fit to execute this simple workflow. Lastly, I would have liked to experiment with configuring the workflow to gauge the implications of granting write permissions to update files, modify the repo automatically or better protect the codebase from outside changes.