Jaxon Froderberg
Final Project Post-Effort Write Up
CPSC 334, DevOps
Professor Crandall

The Project:

For my final project, I planned on converting the first programming assignments I made in CPSC 122: A morse-code encryption tool, "**Encrypty**", which could take basic morse or english text files and convert them into the other's language.

The Tools I used:

My first step in updating this tool was to implement continuous integration. For general automation in later steps, I added a basic makefile that outlined building, running, testing, linting, packaging, and cleaning targets to be called when needed. I used the additional following tools in order to implement other features:

- Doctest was installed to provide unit testing in the c++ language.
- Docker Desktop was downloaded in order to run docker images / packaging of the final project.
- GitHub Action workflows were added to the project to complete CI.

My Packaging Plan:

While I was unable to complete the packaging due to some issues, here are the general steps that I have / would have taken if it was completed.

To build and release Encrypty, I first generated a standard Dockerfile from the template outline on the main docker website and previous assignments in CPSC 334. Once installed, I refactored the Dockerfile to install the necessary c++ compiling files, which would be used to build the final Encrypty program in the image on launch. Lastly, my workflows were updated to build and launch the docker image featuring Encrypty.

Unexpected Challenges and Obstacles:

Over the course of upgrading my final project, I ran into a plethora of problems and issues, not all of which I was able to solve.

I had difficulty determining which c++ unit testing frame I should use. There were several options that all seemed quite good, and I didn't have a lot of time to deliberate on their benefits or potential drawbacks. Whatever I ended up choosing I was most likely stuck with. These drawbacks would soon come back to bite me, as my unfamiliarity with the unit testing software I chose left me unsure at first as to how I was supposed to get my unit tests to display in the output. I also struggled with getting unit tests set up to compare the contents of files against strings. This was very different from my previous experiences with unit tests, as I have never had to work with checking file contents and only did basic number comparisons.

In a similar vein to the unit testing, I was similarly pressed for time on the linter, and ultimately was unable to find a working one in a reasonable amount of time that I could learn to use.

Another critical issue I had to deal with was the nature of my project itself. Most of my functions used variables obtained through passed reference, and featured prompts for user inputs. I struggled to find any reliable way to simulate these user inputs / bypass them short of rewriting all the functions. User input continued to plague me as my main function continued to run after unit tests were called, causing my workflow to stall as it awaited user input. I have found no working solution to this problem despite trying adding premature exits with command-line arguments.

Another issue would be the packaging of my docker program. When packaging my program and attempting to compile / run it from my docker container, the docker container is unable to install g++ to compile the program. This required special configuration I was not able to solve.

On top of all of these issues, Ihad to account for the fact that I had not coded a project in c++ for quite some time, and was therefore generally out of practice with the language.

How could DevOps methodologies benefit my future projects?

The core benefits of having an automated pipeline / system, which can test, lint, and otherwise review your code, or the code of several team members, and provide results in the same day is fairly obvious. As a college student who has up to this point worked with teams that either merge only occasionally at best, or right before the submission deadline of each project, the stress I can be saved by having a system like this to keep everyone synced and up-to-date is invaluable.

How has this Linux & DevOps course influenced my perspective on software development and related technologies?

While my previous courses have for the longest time made a clear point to emphasize the importance of automation and consistent coding standards in professional or casual work environments for the long-term success of a project, I feel that this is the only class which really managed to put the power of that automation into perspective.

The simplistic example of the ever-developing project we built over this course, coupled with the simplistic of the GitHub workflows that outlined this process, clicked with me in a way that previous explanations of unit testing and merging had so far failed to get across. WHile I can't say for certain if coding will be my primary activity moving forward in life, I am fully aware now that the importance of these systems cannot be understated, and should never be ignored.

Meanwhile, the networking side of the class was definitely more daunting. I can see why it's so important in the role it plays with continuous deployment and delivery, and understand that learning about it in even a basic capacity will help immensely with endearing myself to future professional projects that concern themselves with the back-half of the development pipeline.

General notes / sources cited:

For general overall assistance with Doctest, I consulted the doctest repository.

- Pages of particular importance included the test suites page,

CodeProject's blog also covered the docTest setup overall quite well.

PlatformIO also has dedicated pages to provide doctest examples.

This article on GameDev.net was invaluable for actually seeing how the doctest was called in the terminal runner.

For my file-comparison function, I heavily based mine off of Nahla's design from cplusplus. Much of the code was copied where needed.