CI + Build + Docker repo

The world of CI is large, unfortunately I was just too used to Gitlab CI (The best IMHO). Since the code was hosted on GitHub I was out of luck with using Gitlab CI. I took it as a chance to explore other options- Jenkins and Github Actions in particular

Jenkins needs to be run somewhere, and running it on the same server as the deployment might mean both go down at the same time. Also, its just more work and I didn't feel like doing it/ dealing with certs, bla bla.

Github Actions wins by default, the best kind of victory. Github Actions supports building docker images through their `build-push` action. But before that, we need to talk about secret management.

Secrets are a tough nut to crack when it comes to CI/CD - How do you reliably get an API key into the build system without just committing it in code. The solution I went for is still not the most ideal, I can talk about the negative of it in the end. But I went with using the env-method approach. Environment variables can easily be coded into the program, the code simply expects this environment variable to exist. These variables are however injected into the Docker image during the build stage. And they are injected by Github Actions - which reads the value from the Github Secrets feature.

TL;DR:

image-1597600383422.png

Caveats:

In this whole step, the API key is never directly exposed, it is simply referenced. The Github Secret is setup such that a secret can only be aletered and never read once it is entered. Implying one cannot steal an API key, but one can destroy it. This makes the entire CI process relatively safe. Code-scanners wouldn't find any API key in the code to exploit, any attack to find the key needs to be extremely targeted.

The keen eyed amongst you might have noticed a flaw, `Environment Variable`! This implies any person that has access to the docker image, need only run it and then they have complete access to the API key. The mitigration strategy for this is to set the Docker image access as private, enabling a need-only access to it. Narrowing the attack surface. Since the Melton API isn't a critical piece of software, such a narrow attack surface is deemed acceptable.

Since we have the Github Action that can build and push our image, we only need to find an endpoint to push it to. Gitlab private container repositories would be amazing (Ahh Gitlab <3). But since we don't have that, we will defer to just using a private repository on DockerHub.

To summarize,

- API keys are "inserted" as Github Secrets, made accessible to the program as Environment Variables
- Github Actions build the Dockerfile using the default action
- The built docker image is pushed to a private DockerHub repository

---

Revision #6
Created 2 August 2020 07:44:52 by pari
Updated 16 August 2020 17:54:21 by pari