Comment Anywhere

Week 11 Report, 4/10/2023

Karl Miller

On April 3rd, I created an account at Kamatera for cloud hosting. I purchased a single virtual 1A CPU, 2048 MB of virtual RAM, and a 20GB virtual SSD for $6/month. I first attempted to deploy in the same way I had been developing; by running, on our cloud server, a postgres docker container and then connecting to the exposed container port with our HTTP server, written in Go, running as a regular system process.

I was able to *ssh* and *scp* into the server, copying our source code over. I was able to quickly install docker and run the database without issue. It took a few steps to install Go on the server; I had to *curl* the tarball and add go's bin directory to the path to access the global Go installs I needed, such as *migrate*. The environment-variable driven *Makefile* already created for development largely worked well for deployment.

At this point, running the server and configuring the front end's environment variables to point to the server yielded a seemingly working comment anywhere back end. Unfortunately, I quickly realized that when I closed my SSH terminal, the server attached to it would terminate and the back-end would go down. I either had to keep my SSH terminal up permanently or run the process on the server headless.

On April 7th, I tried several solutions for running the server process headless, including *systemctl* and *nohup*. They were all buggy, however, and they felt like hacky workarounds that involved a lot of SSH to get working properly, even with well-configured make commands. I

knew that a robust solution would be to reutilize the technology we were already using for the postgres database and create a container for the HTTP server as well.

I was able to create a custom image that compiled and ran the server fairly quickly, but I ran into issues when it came to connecting to the database. The server was expecting the database to be at a localhost port, but now that they were in separate containers, they were no longer sharing a localhost. The solution was *docker network*, which allows networked containers to reference each other's container names in place of localhost to access each other's exposed ports. Building this image on the server and running it yielded a working and persistent HTTP server which now serves the same comment anywhere data to any front end requesting it.

I made some changes to the source code. I added various Makefile commands to the back end for configuring the network and building the image. I added several optional flags that can be parsed by the server when it runs. The flag *–nocli=true* starts the server without starting the command-line interface. The flag *–env=/path/to/.env* gives the server a custom path to an env variable, in case the .env file is not located in the current working directory at the time the server is started. The flag *–docker=true* starts the server in docker mode, generating a different connection string based on the database image name set in the .env file, allowing a container instance of the server to utilize the docker network. These flags mean that the server can be run in development exactly as it has been but can also be deployed successfully in docker or as a regular server process.

On April 8th, I also made some tweaks on the front end. I fixed an issue with the navbar pane not displaying. I added another .env file called .env.production which *vite*, our front-end packing and developing system will read when it is set to *–mode production*. I added some *npm*

scripts utilizing this. This allows us to easily test the front end against a local instance of the server or to connect to our remote server, depending on our development needs.

Next, I would like to standardize a process for backing up our database in case the server ever goes down. I also want to implement the email features for password recovery and account authentication. Further work can also be done on the front end to make it prettier, though Luke made some large strides in that area last week.