Deployment Process Overview

To host my application, I decided to take a Monolithic Deployment method and use Microsoft Azure to host both my front-end and back-end code. Microsoft Azure is a secure, reliable cloud service that additionally provides easy integration with GitHub, allowing me to setup a CI/CD pipeline using GitHub Actions. In the following section, I will outline the steps of my pipeline.

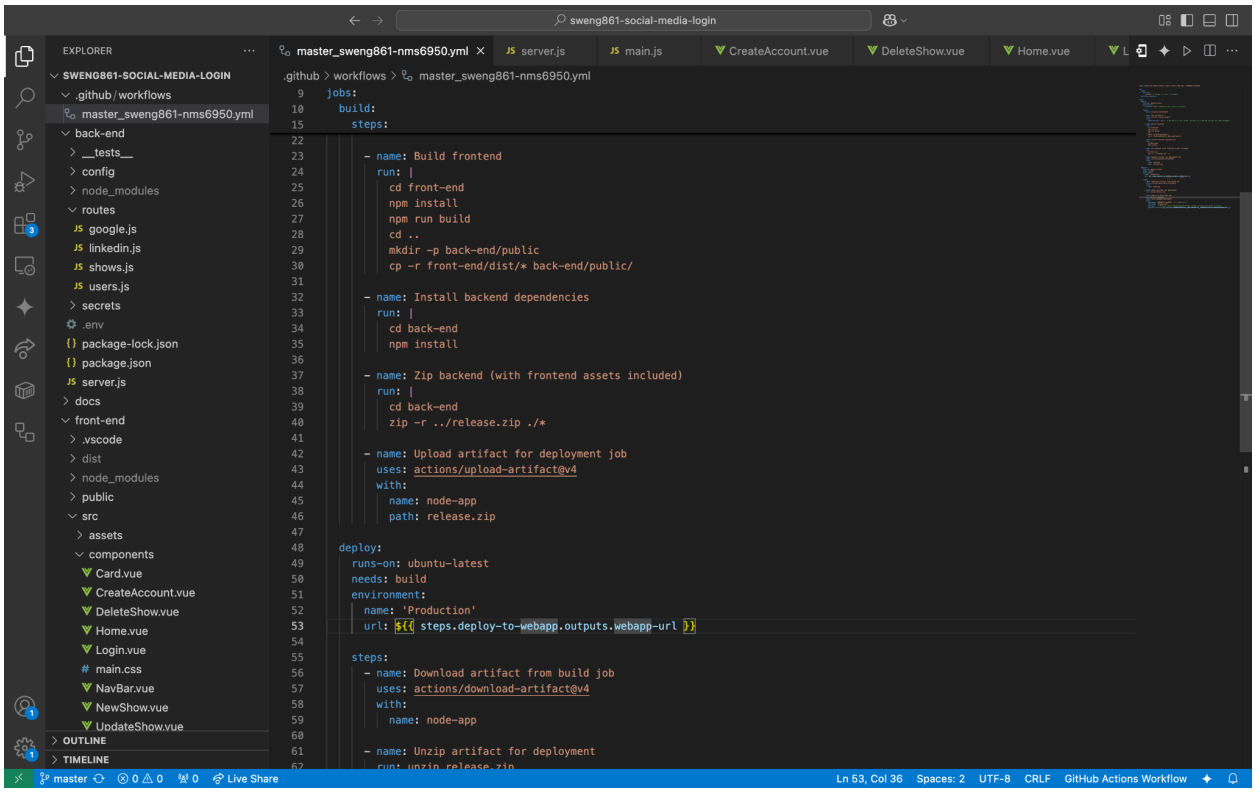
Deployment Process with GitHub Actions CI/CD Pipeline

Displayed below is a screenshot of my .yml workflow file, essential for creating any CI/CD pipeline. This file tells Azure exactly what to do to deploy my latest code.

In the first section of the file, it outlines when to run the file. For my code, I want my code to be continuously deployed every time there is a push to my master branch.

In the section section of the file, it outlines the steps to build the release. Since this is a monolithic deployment, I had to figure out a way to deploy both the front end build and the backend routes at one time. I did this by first installing node packages, and then building my front end code using the command npm run build, which is a build script I setup that uses Vite to build my front-end files. Once this is complete, the pipeline creates a directory in my backend directory called public. It then transfers all build files, located in my front-end dist folder, to the newly created public directory. Then, after installing backend node packages, it zips up the back-end folder and uploads the zip folder for the next job.

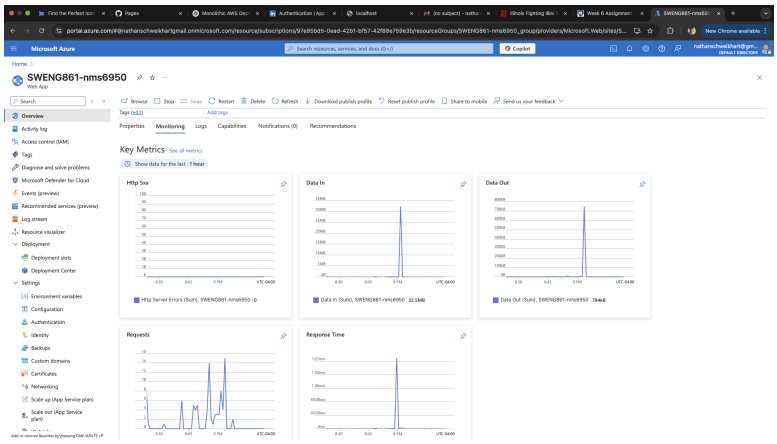
In the last section of the file, the code is retrieved and deployed. After the previous job uploads the release zip folder, the next job downloads that folder, unzips it, and deploys the files to my Azure application. This part of the pipeline is secured by a “publish-profile” secret that matches the secret code from my application.

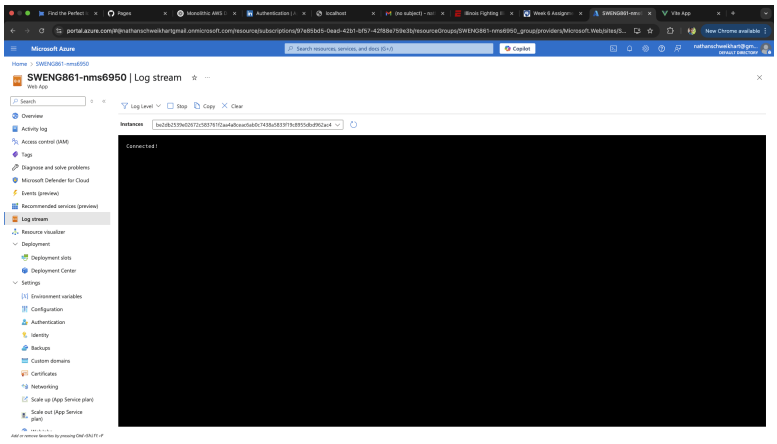


Monitoring and Logging Setup

One benefit of using Microsoft Azure is their built-in tools for monitoring, logging, and diagnostics, regardless of the type of application you are hosting. On the Microsoft Azure portal for my application, I can view all these different tools.

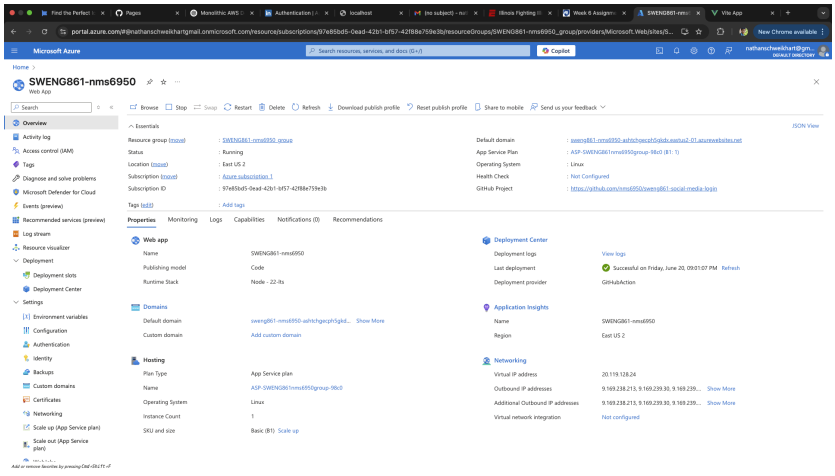
The first screenshot below depicts the Monitoring Tool. This tool provides a dashboard that shows metrics such as data in, data out, requests, and response time.





The second screenshot, above to the right, depicts the Logging Tool. This tool provides an easy location to view any logs related to my application. If there is something that went wrong, I can go here and read the exact error that was thrown from my application. For now, there are none, so it just reads “Connected”.

The third screenshot (below), depicts the Dashboard. This tool is helpful in a number of ways. It gives me basic information, such as application name, domain name, runtime stack, IP Addresses, hosting plan, etc. It also provides a tool to give my application a custom domain name. However, the most useful tool I have found from this page is the deployment center. Now, you can click the link for the deployment center and it will take you to a page that outlines all past deployments with their logs, but I prefer to use this tool mostly to view the status of my current / last deployment. Oftentimes, this location is the most up to date and responsive in terms of which stage my deployment is at, so monitoring this page is incredibly useful.



Testing the Application on Live Server

As a developer should do, once a deployment is done and successful, you need to visit the application domain and ensure the functionality works as intended. I did just this. The screenshots below depict my application was hosted correctly, and the backend code functions with the front-end code as intended and as it was in the past / during the development cycle.

I ran some simple tests to ensure everything was correct. Firstly, I logged in to my application using all three methods of doing so. I had to update my OAuth 2.0 redirect URLs, due to my application being hosted at a new domain, so testing this part of the application was critical. Everything worked as expected, and I was redirected to my home screen. The home screen appeared with all the shows listed, indicating that the shows APIs are working. I ran an additional test, updating one of the test shows I created a while back, and it worked as intended as well.