Terraform & Kubernetes Reflection Paragraph

Braden Gant

November 2025

Introduction

This project gave me a chance to move beyond running containers manually and instead allowed for me to describe infrastructure as code using Terraform. I implemented the same basic dice-rolling application in two different environments, a small Docker-based "local cloud" and a lightweight Kubernetes cluster running on my own machine. Working through both parts of this project forced me to think about networking, modularity, and overall deployment in a much more systematic way than just starting containers by hand.

Part 1: Docker Terraform Project

In the Docker portion of the project, I learned how Terraform can be utilized for a small, local cloud setup. Defining separate modules for the frontend, backend, and database pushed me to think in terms of modularity and the overall deployment. When developing the Terraform Project, I also had to be intentional with how services communicated to each other. This was done by wiring everything through a custom Docker network and carefully mapping ports so that the Nginx frontend could talk to the Flask API, and the API could then reach the Postgres database. I then used variables and a `.tfvars` file to pass database credentials more securely, which made me more aware on how to handle secrets safely within a database. Utilizing Terraform from inside the Docker container was another good lesson, as it kept my host environment clean but forced me to understand the practicality and use cases of volume mounts, Docker sockets, and how Terraform talks to the Docker daemon. By the end of Part 1 I felt much more comfortable reading Terraform plans, understanding what resources would be created, and using `apply` and `destroy` to bring the entire stack up and down in a repeatable way, I have also already thought of other projects that I want to create using Terraform after this class.

Part 2: Kubernetes Terraform Project

The Kubernetes portion of the project shifted my perspective from individual containers to higher level abstractions like Namespaces, Deployments, Services, and ConfigMaps. Instead of directly creating containers, I described the desired state of the cluster and let Kubernetes handle scheduling and lifecycle management. Getting the `kind` cluster working, loading my local `dice-backend` image into it, and then configuring the Terraform `kubernetes` provider taught me how many pieces have to line up for a cluster deployment to succeed. Defining a Namespace for the application helped learn how to isolate its resources, and create a NodePort Service that had to be configured to handle traffic that enters the cluster from the host. The ConfigMap enhancement was especially useful, at it showed me how to separate configuration from images and inject settings like `DEFAULT_DIE` into the pod at runtime. I also had to debug issues like image pull failures and connection errors, which forced me to read Kubernetes events, double-check ports, and use `kubectl port-forward` to verify that the Service was actually reachable. Overall, Part 2 gave me a much better understanding of how Terraform can manage not just local infrastructure, but an orchestrated environment where a declarative configuration and cluster state have to stay in sync.