Capstone Project Report

End-to-End Workflow

1. Developed a Django REST API with a `Book` model, and full CRUD functionality.
2. Containerized the application using a Dockerfile and the Docker Compose.
3. Wrote unit tests using Django's built-in test framework.
4. Used Git for version control and followed with Conventional Commit standards.
5. Pushed the code to GitHub and configured a CI/CD pipeline using GitHub Actions.
6. Created a Helm chart to define Kubernetes resources (Deployment, Service).
7. Deployed the app automatically to Kubernetes using the Helm chart.

How the Docker Image is Built

The Dockerfile installs Python, copies the application code, and installs all the required dependencies from `requirements.txt`. It exposes port 8000 and runs the Django development server. Docker Compose is used for local development, bundling the web app and optionally a database.

How the CI/CD Pipeline Works

The GitHub Actions workflow triggers on each push to `main` branch. It performs following steps:

- Checks out code
- Sets up Python and installs dependencies
- Runs unit tests
- Logs in to the Docker Hub
- Builds and pushes the Docker image
- Sets up `kubectl` and `helm`
- Deploys the image to Kubernetes using Helm Secrets are stored securely in GitHub:
    - `DOCKER_USERNAME`
    - `DOCKER_PASSWORD`
    - `DOCKER_IMAGE_NAME`
    - `KUBE_CONFIG_DATA`.

How the Helm Chart Deploys Application

The Helm chart contains templates for Kubernetes resources. `values.yaml` allows configuration of image repository, tag, and service details. When `helm install` or `helm upgrade` is run, it renders these templates into Kubernetes manifests and applies them. This automates the creation of the Deployment, Service, and optionally Ingress.

Lessons Learned and Challenges Faced

At first, setting up the Kubernetes deployment via GitHub Actions was problematic due to kubeconfig authentication issues. The solution was to install `kind` (Kubernetes-in-Docker) locally and use it to create a local Kubernetes cluster. The kubeconfig from `kind` was encoded in base64 and added as a GitHub Secret (`KUBE_CONFIG_DATA`), allowing GitHub Actions to deploy using Helm successfully. This helped me understand how local and remote K8s clusters work in CI/CD pipelines.

The link to the github repository is:
https://github.com/cct-gohar/summer-2025-devops