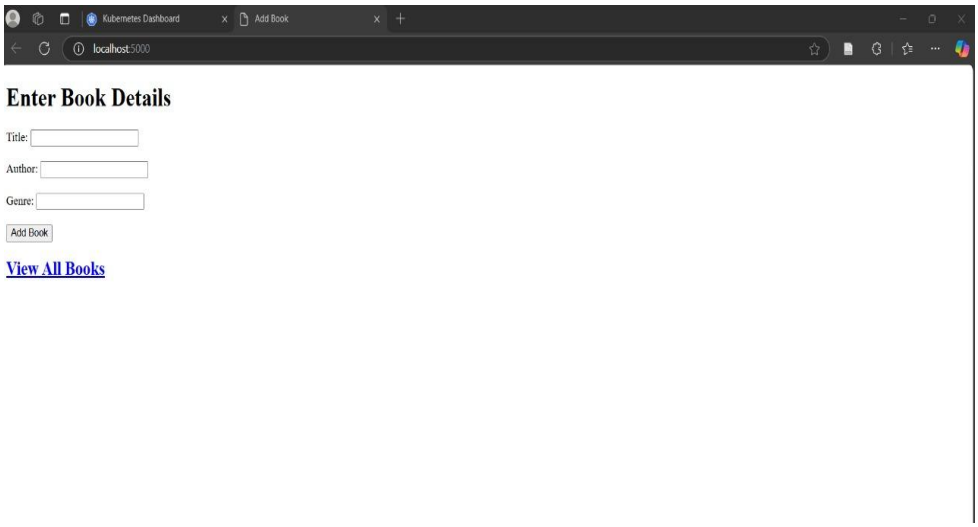
PA2577: Assignment: Build Something

Name: SumaVendrapu

Email: suve22@student.bth.se

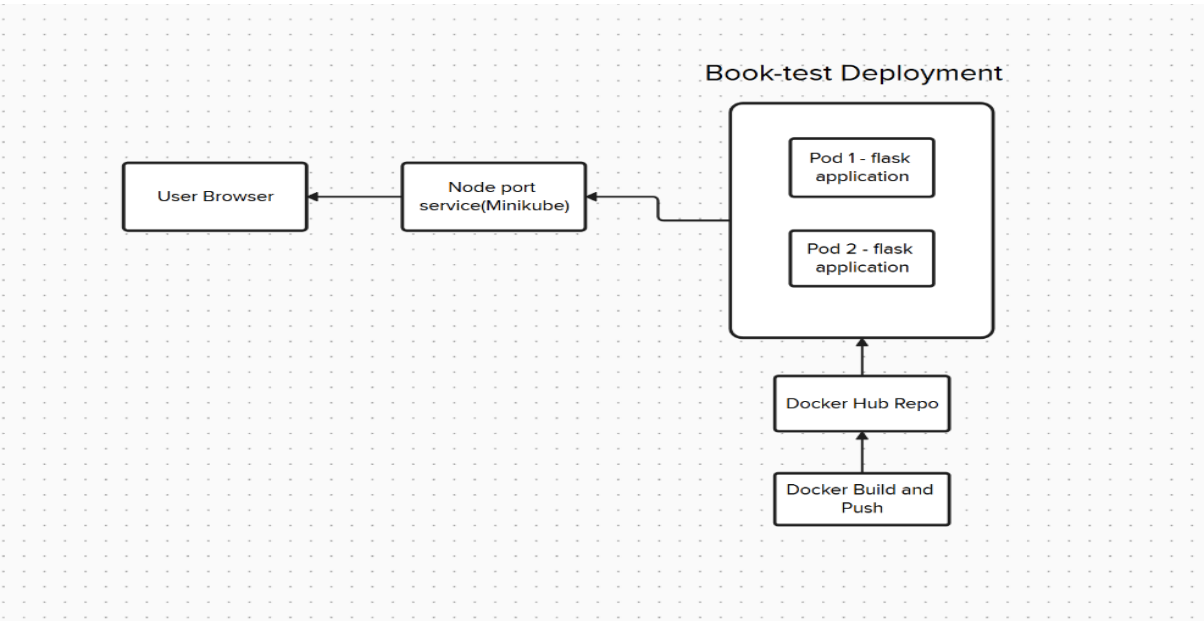
Application Overview

The **Book Service Application** is a Flask-based web app designed to manage a collection of books by allowing users to add and view details like title, author, and genre. The application is containerized using Docker and deployed on a Kubernetes cluster. Once deployed, users can access the application through their web browser. Requests are first routed to a Kubernetes Load Balancer Service, which exposes the application externally and forwards these requests to the Pods running the Flask application. Each Pod is part of a Deployment that ensures multiple replicas are running, providing fault tolerance and scalability. The Flask application in each Pod interacts with an SQLite database to store and retrieve book details,



maintaining data locally within the Pod’s filesystem.

In this Kubernetes deployment, the Load Balancer Service ensures smooth communication by routing user requests to the appropriate Pods based on availability and load. The Deployment manages the Pods' lifecycle, including scaling replicas up or down to handle varying traffic levels. The application image, hosted on Docker Hub, is pulled during Deployment creation, ensuring consistent environments across all replicas. This architecture leverages Kubernetes' built-in capabilities for automatic scaling, high availability, and load balancing, ensuring a seamless user experience with a reliable and scalable backend infrastructure.



Prerequisites

1. **Docker:**
    - To build the application image and push it to Docker Hub.
    - Install Docker and ensure it is configured correctly.
2. **Docker Hub Account:**
    - To host the Docker image for the application.
    - Ensure you have an active account and are logged in to Docker CLI.
3. **Minikube:**
    - To set up and manage a local Kubernetes cluster.
    - Install Minikube and verify it is working by starting a cluster using minikube start.
4. **Kubectl:**
    - To interact with the Kubernetes cluster and manage resources.
    - Install Kubectl and configure it to work with Minikube.
5. **Python and Flask Knowledge:**
    - Basic understanding of Python and Flask, as the application is written in Flask.
6. **YAML Files for Kubernetes Configurations:**

- Ensure you have the deployment.yaml and service.yaml files to define the Deployment and Service resources for Kubernetes.

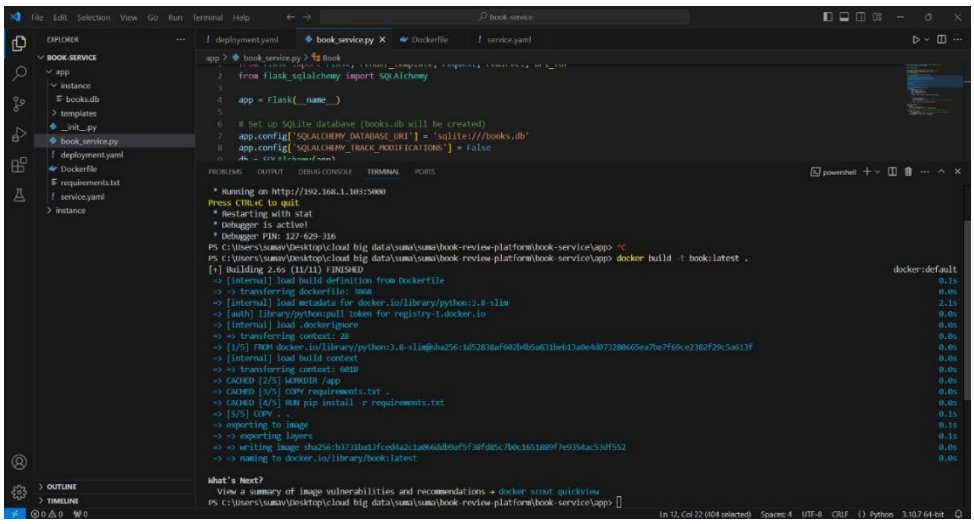
7. **SQLite (Optional for Local Testing):**
   - Required if you are testing the application locally before deployment.

Docker

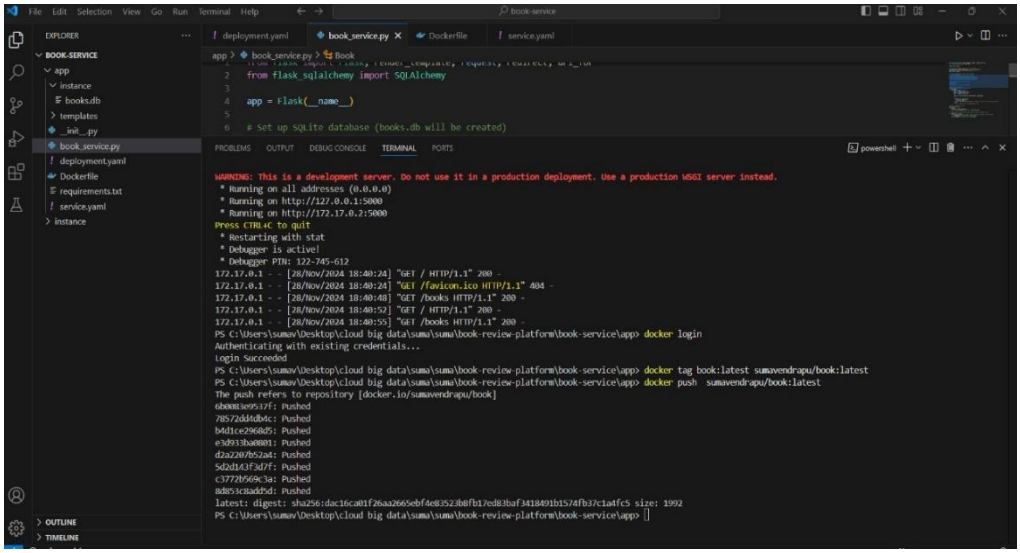
The **Dockerfile** is a script used to build a container image for the Book Service Application. It automates the setup of the application environment, including installing dependencies and configuring the Flask application to run inside the container.

The Dockerfile begins by specifying a lightweight Python image as the base for the container using the line FROM python:3.8-slim. This establishes a minimal environment suitable for running Python applications. Next, the WORKDIR /app instruction sets the working directory inside the container to /app, ensuring that all subsequent commands are executed from this location. The COPY . /app command then copies all files from the current directory, where the Docker build command is executed, into the /app directory inside the container. To install the necessary dependencies, the RUN pip install --no-cache-dir -r requirements.txt command is used to install all Python packages listed in the requirements.txt file, ensuring that the environment is properly configured. The EXPOSE 5000 directive informs Docker that the container will listen on port 5000, typically used for Flask applications. Finally, the CMD ["python", "book_service.py"] line specifies the command that will be run when the container starts, in this case, launching the Flask application by running the book_service.py script.

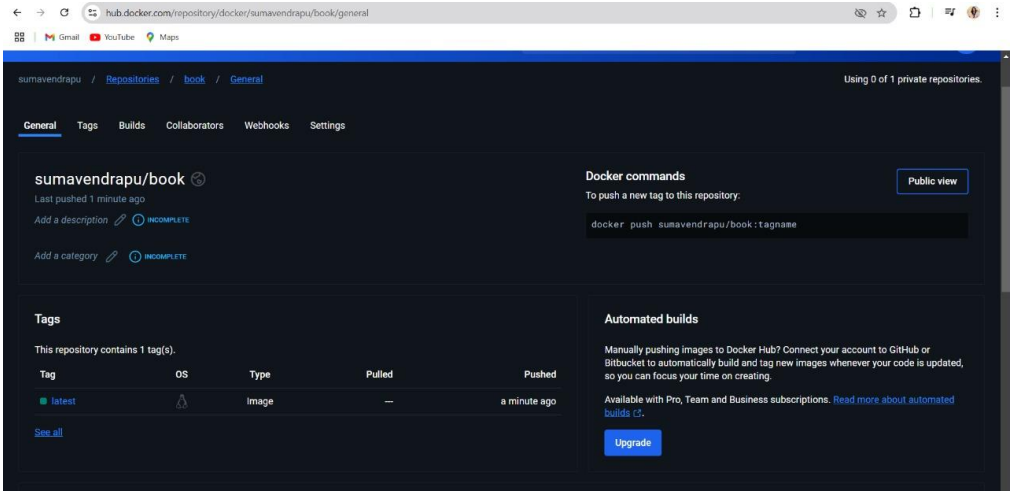
Docker build command is used to create an image of the application and below is the proof



After the image creation the image is needed to be pushed to docker hub and below



The push images of my application is seen in the docker hub and below is the proof.

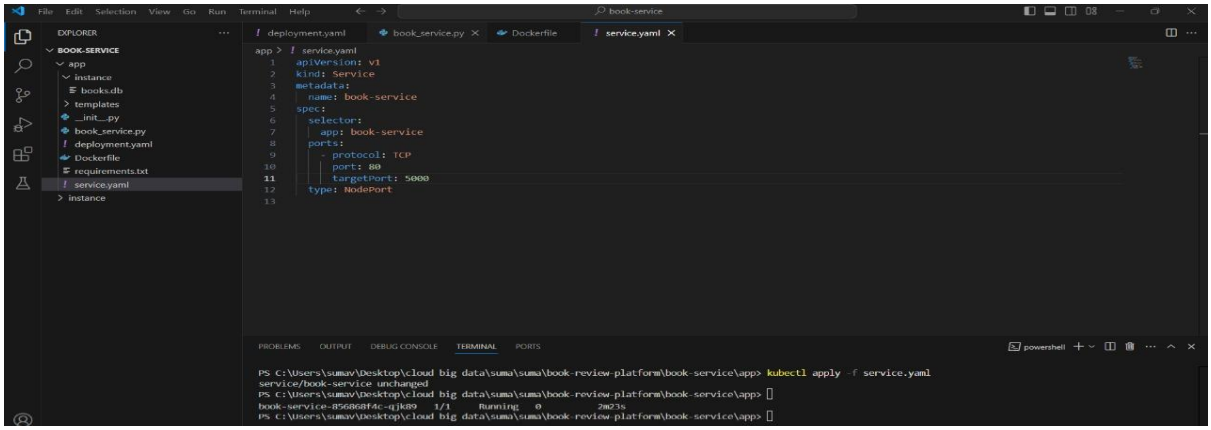


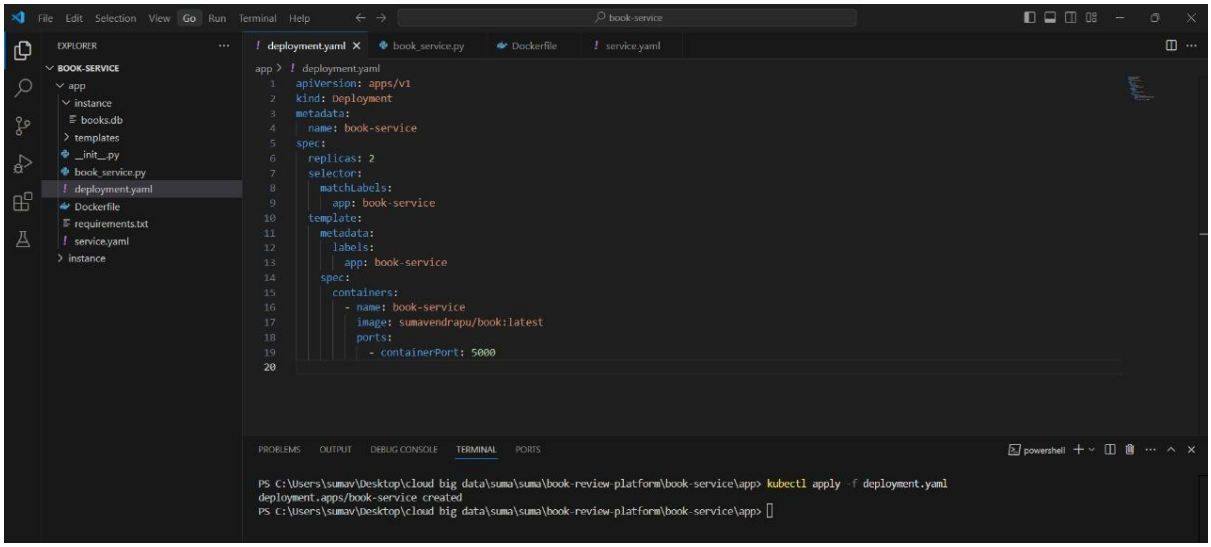
Kubernetes

First, start the Minikube service by running the necessary command. After that, check the status of the Kubernetes cluster using kubectl to ensure that everything is set up correctly. Once confirmed, create the deployment.yaml and service.yaml files, which contain configuration details about the service, including the number of replicas and the image names to be used. These image names should correspond to the Docker images that have been pushed to Docker Hub.

After deploying the configuration files, use the kubectl get pods command to verify if the pods are running correctly and whether the image has been pulled successfully

Below are the proofs of executions





Details of Your Deployment

The application is deployed as follows:

Containerization:

The application being developed is containerized with Docker and a Docker file to build and package Flask application, and its dependencies.

Deployment on Kubernetes:

- By means of a ReplicaSet, a Kubernetes Deployment manages three replica Pods to guarantee high availability. Every Pod executes the Flask web application and is linked to an SQLite database where book information will be stored.

Service Configuration:

A LoadBalancer Service helps put the application outside the cluster for users to access while distributing traffic to the Pods.

Image Hosting:

The Docker image is maintained on Docker Hub to ensure the stable image download during Kubernetes configuration.

Access:

End users might access the application, through a public IP address that is created by the Load Balancer. Security Problems Found and/or Addressed

Securing Communication:

Next versions can utilize HTTPS for safe connection between the client and the application.

Container Security:

The base image used (python:3.8-slim is as list below in order to minimize the area of vulnerabilities: Necessary packages are not installed which means that unnecessary services which can be utilized by a hacker are also not installed.

Data Security:

In case Pods are deleted, which is a local database, data stored in SQLite can also be lost. This can be solved by using persistent storage or moving to a cloud database solution as a resolver for the issue.

Image Integrity:

Images are based on Docker Hub where version control allows only relevant images to be used for a given application.

Pod Security:

Kubernetes Pod Security Policies in the production environment can be used to limit capability and also set isolation between the Pods.

These problems allow the application to be protected against possible threats, and the data as well as the operations of the user to remain secure.