Docker would be our choice of software to containerize our application as it is the most popular one by far and as such it is easy to find resources and images for it. We can also expect that most services will support it due to its widespread usage. As of right now, our project already has Docker files. With the current setup, the MySQL server and the Flask application are containerized independently, essentially isolating the two as though they were running on separate systems. The Flask app uses an official Python image, specifically one based on Alpine since Alpine is minimal, while the MySQL app just uses the standard official MySQL image. Additionally, locally, a volume is made using Docker so that the database exists outside of the MySQL container and is not dependent on it. This ensures that data is preserved and safeguarded in the event that the SQL container fails or needs to be taken down. In a production environment, ensuring that the database is separate would be important for scalability and synchronization as well.

Render is a platform that Aditi has used in a previous class to deploy a project that used React and MongoDB. We found that it supports deploying Flask applications and MySQL instances as well as usage of Docker for deployment, so it should be fully capable of deploying our entire application. To deploy containerized instances of our application with Render, we can upload the appropriate Docker images to Docker Hub registries. For our application, this would essentially mean building Docker images, uploading them as private, and using the Render services to deploy it from there. Docker compose is not truly usable for deployment here, but a similar setup can still be done with Render. In the context of Render, The Flask application would be deployed as a “web service” and the MySQL server would be deployed as a “private service.” Since a volume through compose is not possible here, a “persistent disk” from Render would hold the database and be attached to the MySQL private service.