Deployment Strategy for Managing Embeddings

Synthiq is a tool built for testing and proof of concept work, it's a "second brain" that gets it's information from

For a cost effective way to demonstrate what exactly embeddings are and how they will benefit or in fact make this project possible I will be deploying a **beta** embeddings management system. I'm specifically calling it a **Beta** version since it's not tested and could have some bugs.

To deploy such an application you'd usually need an average to powerful server with some good storage capability, loads of ram and a GPU for machine learning algorithms. So for this demo I am using Google's platform, specifically Google Kubernetes Engine (GKE) and below are the steps I am taking to deploy the demo application

Service Selection

Given that your application consists of two Docker containers, one for the frontend and another for the backend, Google Kubernetes Engine (GKE) would be the most suitable choice for deployment. GKE provides a managed Kubernetes service that allows you to run Docker containers at scale. It offers features like auto-scaling, load balancing, and automated rollouts/rollbacks, which are essential for production-level applications.

Steps to Deploy on GKE

1. **Install Google Cloud SDK**: If you haven't already, install the Google Cloud SDK on your local machine.

2. **Authenticate with GCP**: Run `gcloud auth login` and follow the prompts.

3. **Set up a GKE Cluster**: Create a new cluster in GKE via the GCP Console or by running:

```
gcloud container clusters create synthiq-cluster
```

4. **Configure kubectl**: After the cluster is created, configure kubectl to use it.

```
gcloud container clusters get-credentials synthiq-cluster
```

5. **Build Docker Images**: Navigate to your project directory and build your Docker images.

```
docker-compose build
```

6. **Push Docker Images to Google Container Registry (GCR)**:

```
# web is where the frontend code will be deployed and backend where the admin side
of synthiq will be deployed
docker tag web gcr.io/[PROJECT_ID]/web
docker tag backend gcr.io/[PROJECT_ID]/backend
docker push gcr.io/[PROJECT_ID]/web
docker push gcr.io/[PROJECT_ID]/backend
```

7. **Create Kubernetes Deployment Configs**: Create YAML files for your frontend and backend services. Update the image URLs to the ones in GCR.

8. **Apply Kubernetes Configs**:

```
kubectl apply -f frontend-deployment.yaml
kubectl apply -f backend-deployment.yaml
```

9. **Expose Services**: Finally, expose your services to access them publicly.

```
kubectl expose deployment web --type=LoadBalancer --port 80 --target-port 3000
kubectl expose deployment backend --type=LoadBalancer --port
```

10. **Deploy Synthiq**: Whe the services are exposed to the public, uts