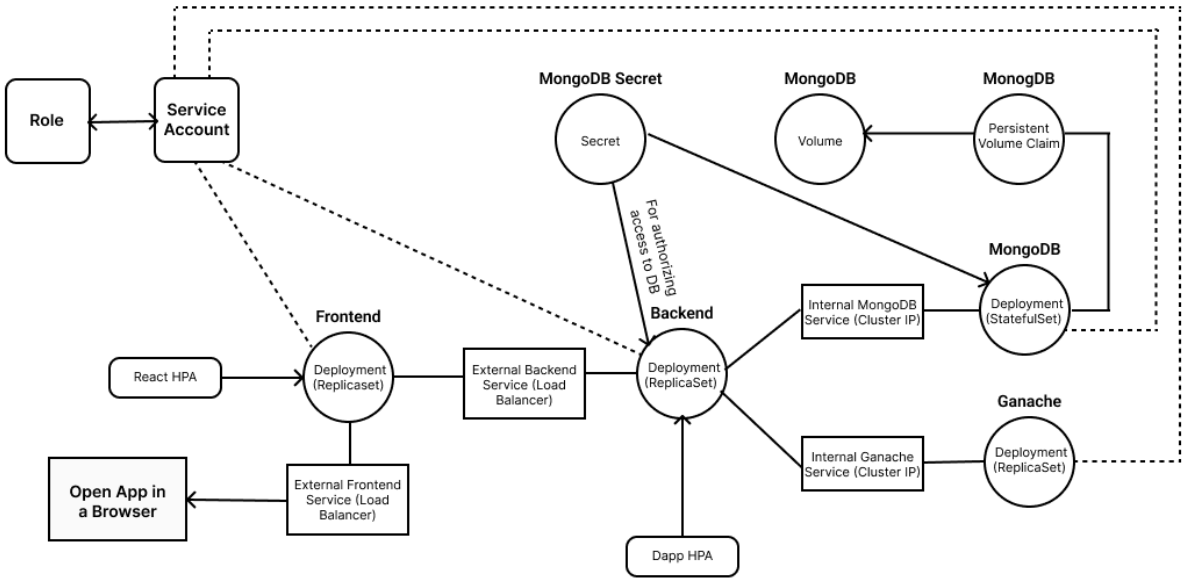
Kubernetes Deployment Architecture for the docker-ethereum application



Explanation :

1. Deployment Options: StatefulSets and ReplicaSets:

- React, DApp, Ganache (ReplicaSets): These three elements represent stateless applications, meaning their state doesn't have to endure through pod restarts. Deployments, controlled through ReplicaSets, offer an effective abstraction for executing stateless pods. This setup enables straightforward scaling and rollouts, making it well-suited for stateless applications such as DApp, React frontend, and the Ganache Ethereum simulator.

- MongoDB (StatefulSet): I opted for a StatefulSet configuration for MongoDB due to its nature as a database that demands both persistent storage and a consistent identity. The StatefulSet guarantees that each MongoDB pod retains its state through restarts, and the data is securely stored using PersistentVolumeClaim (PVC). This methodology is crucial for databases where maintaining data consistency and durability is of utmost importance.

2. Storage:

- PersistentVolumeClaim (PVC): To guarantee data persistence for MongoDB, I used a PVC. I was able to control storage details regardless of how they were used due to this abstraction. I have configured it to ask for 100Mi of storage, appropriate for testing and development. MongoDB data is preserved over pod restarts and node changes because of this PVC.

3. Scaling:

- Horizontal Pod Autoscaler (HPA): The Horizontal Pod Autoscaler (HPA) is a tool for scaling. For React, and DApp deployments, I've set up HPAs. These autoscalers allow pod replicas to be automatically scaled in response to observed CPU and/or memory utilization, which is essential for managing varying loads and preserving performance during periods of high traffic.

4. Services:

- In order to create dependable network endpoints, I created services for each of the various components (MongoDB, DApp, React, and Ganache) in Kubernetes load balancing. These Services are essential for maintaining a balance between internal and client requests because they effectively distribute incoming network traffic among the connected pods. To guarantee dependability and high availability, this strategy is necessary.

- In order to maximize performance and responsiveness, load balancing is also used for DApp and React components in order to divide internal and user traffic equally. However, the use of Cluster IP for Ganache and MongoDB is intentional since it gives these parts of the Kubernetes cluster a consistent internal IP address, allowing for restricted and secure access without exposing them to the outside world.

5. Secrets (Kubernetes Secrets):

- MongoDB credentials were one of the sensitive data I handled with Kubernetes Secrets (mongodb-secret). By not hardcoding critical data into the application code or Docker images, this method improves security. The Secrets are safely installed into MongoDB pods, guaranteeing that private information is controlled and only available when required.

6. User and Role Management (ServiceAccount, Role, RoleBinding):

- These elements are essential to Kubernetes cluster access control. The application pods need an identity from the ServiceAccount ('service-account') in order to communicate with the Kubernetes API. Certain permissions are defined

by the Role, and the RoleBinding grants the ServiceAccount access to these permissions. This configuration strengthens the security of the Kubernetes environment by enforcing the least privilege principle.

Application Demo Link :
https://drive.google.com/file/d/1eZyHZFdeQ1kKYOUNS8cz7x1dt1KrCVAQ/view?usp=sharing