Explanations

1. **Deployment Options**:

   React, DApp, Ganache (ReplicaSets): These components are stateless applications, which means they do not need to preserve their state when pods are restarted. ReplicaSets control deployments, which are a useful way to run stateless pods. This arrangement allows for easy scaling and updating, which is ideal for stateless applications like DApp, React frontend, and the Ganache Ethereum simulator.

   MongoDB (StatefulSet): I chose a StatefulSet configuration for MongoDB because it is a database that requires both persistent storage and a stable identity. The StatefulSet ensures that each MongoDB pod keeps its state when pods are restarted, and the data is safely stored using PersistentVolumeClaim (PVC). This approach is essential for databases where data consistency and reliability are very important.

2. **Storage**:

   I used a PVC to keep MongoDB data safe. A PVC lets me choose how much storage I need, no matter what I use it for. I am using 100Mi of storage, which is enough for testing and development. The PVC makes sure MongoDB data stays the same even if pods restart or move to different nodes.

3 **Scaling**:

I used an HPA to change the number of pods based on how much they are used. I set up HPAs for React and DApp deployments. These HPAs can automatically add or remove pod replicas depending on how much CPU and memory they use. This is important for handling different levels of demand and keeping the performance good when there is a lot of traffic.

4 **Services:**

- I used load balancing for DApp and React parts to split the network traffic evenly between internal and user requests. This helps improve the speed and responsiveness of my dapp. But I used Cluster IP for Ganache and MongoDB because I don’t want them to be exposed to the outside world. Cluster IP gives them a fixed internal IP address, so I can limit and secure their access.

- I made services for each part of my dapp (MongoDB, DApp, React, and Ganache) to connect them with reliable network endpoints. These services help balance the network traffic between internal and client requests by spreading it among the pods. This way, I can make sure my dapp is always available and reliable.

5 **Secrets**:

I used Kubernetes Secrets (mongodb-secret) to protect MongoDB credentials. This way, I don't have to put sensitive data in the code or the Docker images, which makes it more secure. The Secrets are only installed in MongoDB pods, so I can control who can see them and when.

6 **Roles**:

These parts help control who can do what in the Kubernetes cluster. The Role says what they can do, and the RoleBinding gives them the right to do it. This way, they only get the minimum access they need, which makes the Kubernetes environment more secure. The ServiceAccount lets the app pods talk to the Kubernetes API.