Design Document: Kubernetes Deployment Architecture for Docker-Ethereum Application

Introduction:

The purpose of this document is to outline the Kubernetes deployment architecture for the Docker-Ethereum application that was given to us as a Lab task. The Docker-Ethereum application is a decentralized application (DApp) running on the Ethereum blockchain, and its deployment architecture needs to be designed for scalability, reliability, and maintainability.

1. Kubernetes Cluster Configuration:

Design Rationale:

Utilizing a multi-node Kubernetes cluster: To ensure high availability and distribute the application workload across multiple nodes.

-Different deployment objects are created for different components

Will be using kubectl for deploying a kubernetes cluster.

2. Pod Architecture:

Design Rationale:

Stateless pods for backend components: The backend components of the Docker-Ethereum application are designed to be stateless to facilitate horizontal scaling. Stateless pods enable easy replication and distribution of workloads across nodes without the need for shared state, improving overall system flexibility.

Stateful pods for Ethereum blockchain nodes: The Ethereum nodes require persistent storage for maintaining the blockchain state. StatefulSets can be used for these nodes, ensuring stable network identifiers and persistent storage for blockchain data. This choice ensures data consistency and reliability across node restarts.

3. Service and Ingress Configuration:

Design Rationale:

Kubernetes Services for internal communication: Services are used to create a stable network endpoint for communication between different microservices within the application. This

helps in abstracting the underlying pod IPs, providing a consistent and reliable way for services to communicate.
Ingress for external access: Ingress resources are employed to expose the Docker-Ethereum application to the external world. This allows for the configuration of routing rules, SSL termination, and load balancing for incoming traffic. Utilizing Ingress simplifies the management of external access and provides flexibility in handling different paths and subdomains.

4. Persistent Storage:

Design Rationale:

Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) for stateful components: Ethereum nodes require persistent storage to maintain blockchain data. PVs and PVCs ensure that the data is retained even if a pod is rescheduled or replaced. This guarantees data consistency and integrity, crucial for the decentralized nature of the application.

P.S Still working on it

5. ConfigMaps and Secrets:

Design Rationale:

ConfigMaps for configuration data: ConfigMaps are used to store configuration data that can be mounted as volumes or environment variables in pods. This facilitates easy configuration changes without the need to rebuild the Docker images.
Secrets for sensitive information: Kubernetes Secrets are employed to store sensitive information, such as API keys or private keys required by the Docker-Ethereum application. This ensures a secure way of handling confidential data within the cluster.