Anishkumar Pankajkumar Patel

101504708

# BCDV 4032

# Building Scalable Blockchain Apps

# Lab 04

28th Jan 2024

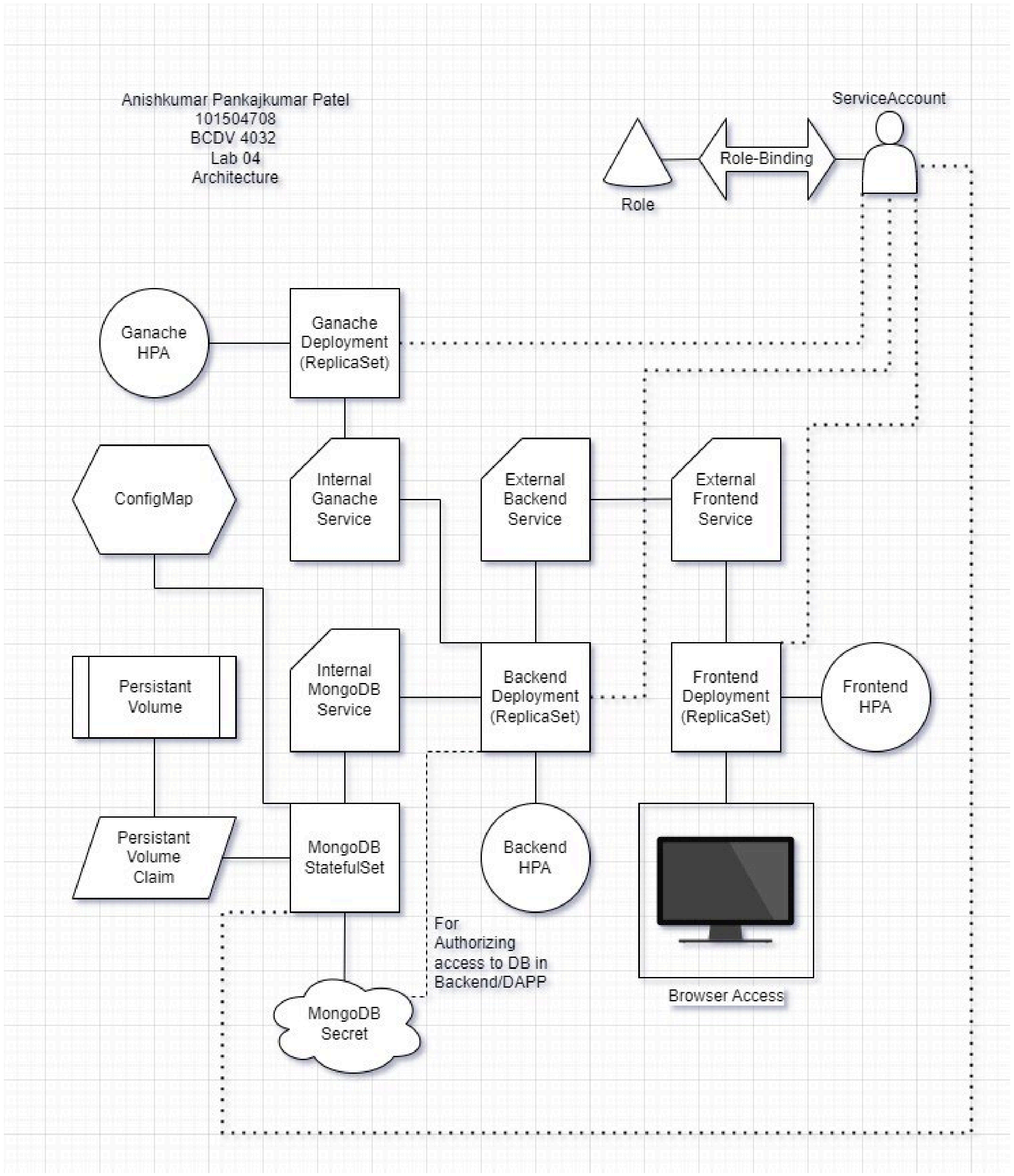
Keerthi Nelaturu

# Lab 04

## Section 01 – YouTube Link for Demonstration of Lab 04

## Link - [https://youtu.be/fdxWK64F1zI](https://youtu.be/fdxWK64F1zI)

## Section 02 – Architecture Diagram



## Section 03 – Explanation of the elements

The rationale behind each element in the Kubernetes architecture for the Docker Ethereum application:

1. Deployment Options: StatefulSets and ReplicaSets
   - MongoDB (StatefulSet): I chose a StatefulSet for MongoDB because it's a database requiring persistent storage and a stable identity. The StatefulSet ensures that each MongoDB pod maintains its state across restarts, and the data persists via the PersistentVolumeClaim (PVC). This approach is essential for databases where data consistency and durability are paramount.
   - DApp, React, Ganache (Deployments using ReplicaSets): The components here are stateless applications. Their state doesn't need to persist across pod restarts. Deployments, managed via ReplicaSets, provide a suitable abstraction for running stateless pods. They facilitate easy scaling and rollouts, ideal for stateless applications like DApp, React frontend, and the Ganache Ethereum simulator.
2. Storage (PersistentVolumeClaim for MongoDB)
   - PersistentVolumeClaim (PVC): I used a PVC for MongoDB to ensure data persistence. This abstraction allowed me to manage storage details independently of how it's consumed. I've set it to request 200Mi of storage, fitting for development and testing. This PVC ensures that MongoDB data remains intact across pod restarts or node changes.
3. Scaling (Horizontal Pod Autoscaler)
   - Horizontal Pod Autoscaler (HPA): I've configured HPAs for DApp, React, and Ganache deployments. These autoscalers enable automatic scaling of pod replicas based on observed CPU and/or memory usage, vital for handling changing loads and maintaining responsiveness during peak traffic.
4. Load Balancing (Services)
   - Kubernetes Services: I created Services for each component (MongoDB, DApp, React, Ganache) to provide consistent network endpoints. They effectively distribute incoming network traffic across the underlying pods, crucial for balancing client or internal requests and ensuring reliability and high availability.

5. Secrets (Kubernetes Secrets)
    - Kubernetes Secrets (mongodb-secret): I managed sensitive information, the MongoDB credentials, using Kubernetes Secrets. This approach enhances security by avoiding the hardcoding of sensitive data into the application code or Docker images. The Secrets are securely mounted into MongoDB pods, ensuring sensitive data is well-managed and accessible only where necessary.
6. User and Role Management (ServiceAccount, Role, RoleBinding)
    - ServiceAccount, Role, and RoleBinding: These components are crucial for access control within the Kubernetes cluster. The ServiceAccount (***'all-purpose-user'***) provides an identity for the application pods to interact with the Kubernetes API. The Role defines specific permissions, and the RoleBinding assigns these permissions to the ServiceAccount. This setup enforces the principle of least privilege, enhancing the security of the Kubernetes environment.

Each element in this architecture plays a vital role, addressing needs such as data persistence, state management, security, scalability, and load balancing. Together, these components form a robust and scalable architecture for the Docker Ethereum application.

## Section 04 – Commands to run the yaml files and to make the Lab work for your own.

Lab 04 Commands (Make sure to change the folder to where the files are saved)

- minikube addons enable metrics-server

- kubectl apply -f mongodb-secret.yaml
- kubectl apply -f mongodb-data-persistentvolumeclaim.yaml
- kubectl apply -f mongo-init-configmap.yaml
- kubectl apply -f mongodb-stateful-set.yaml

start shell in mongodb pod

- printenv
- mongosh
- db.getUsers()
- use admin
- db.createUser({user: "root", pwd:"password", roles:[{role:"readWrite", db:"myDatabase"}],});
- db.getUsers()

- kubectl apply -f mongodb-service.yaml
- kubectl port-forward svc/mongodb 32000:27017

Open MongoDBCompass → Connect with root and password using Authentication at port 32000

- kubectl apply -f backend-deployment.yaml
- kubectl apply -f backend-service.yaml
- kubectl apply -f frontend-deployment.yaml
- kubectl apply -f frontend-service.yaml
- kubectl apply -f ganache-deployment.yaml
- kubectl apply -f ganache-service.yaml
- kubectl apply -f all-purpose-roles-users.yaml
- kubectl apply -f hpa.yaml
- kubectl port-forward svc/dapp 4000:4000
- kubectl port-forward svc/react 3000:3000

- Frontend → localhost:3000