Kubernetes Deployment Architecture Design Document

1. Introduction

This document outlines the Kubernetes Deployment Architecture for the docker-ethereum application. The architecture includes Horizontal Pod Autoscaling, User Role Creation, and Load Balancing. Each section provides detailed instructions and rationale for the design choices.

2. Horizontal Pod Autoscaling (HPA)

2.1 Scaling Files

- **dapp-hpa.yaml**

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: dapp-hpa
spec:
  scaleTargetRef:
apiVersion: apps/v1
    kind: Deployment
    name: dapp
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

- **frontend-hpa.yaml**

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: dapp-hpa
spec:
  scaleTargetRef:
apiVersion: apps/v1
    kind: Deployment
    name: react
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

2.2 Scaling Addon Service

- Enable metrics-server: `minikube addons enable metrics-server`

2.3 Apply Scaling YAML Files

- `kubectl apply -f dapp-hpa.yaml`
- `kubectl apply -f frontend-hpa.yaml`

2.4 Check HPA Status

- `kubectl get hpa`

2.5 Rationale

- Horizontal Pod Autoscaling is implemented to dynamically adjust the number of replicas based on CPU utilization.
- Min and Max replicas are set to ensure resource efficiency while accommodating varying workloads.

3. User Role Creation

3.1 User Creation Commands

- Generate key and CSR:
```
openssl genpkey -algorithm RSA -out test-user.key
openssl req -new -key test-user.key -out test-user.csr -subj "/CN=test-user/O=group1"
openssl x509 -req -in test-user.csr -CA ~/.minikube/ca.crt -CAkey ~/.minikube/ca.key -CAcreateserial -out test-user.crt -days 500
```

- Set credentials: `kubectl config set-credentials test-user --client-certificate=test-user.crt --client-key=test-user.key`

3.2 Create User Context

- `kubectl config set-context test-user-context --cluster=minikube --user=test-user`

3.3 Permissions and Role Binding

- Deny access to pods: `kubectl --context=test-user-context get pods` (should result in permission denied)
- Create role and binding:

```
kubectl create role example-role --verb=get,list,watch --resource=pods
kubectl create rolebinding example-binding --role=example-role --user=test-user
```

3.4 Verify User Permissions

- Allow read access: `kubectl --context=test-user-context get pods`
- Demonstrate lack of write permissions: `kubectl --context=test-user-context delete pod <YOUR pod name>`

3.5 Rationale

- User roles are created to control access to Kubernetes resources.
- The example demonstrates the creation of a read-only user with restricted permissions.

4. Load Balancing

4.1 Service Specification

- Update `react-service.yaml`:
  spec:
    type: LoadBalancer
    ports:
      - name: "http"
        port: 3001
        targetPort: 80
    selector:
      io.kompose.service: react

4.2 Apply Service Specification

- Apply changes: `kubectl apply -f react-service.yaml`

4.3 Verify Load Balancer Type

- Check service types: `kubectl get services`

4.4 Access Application

- Run: `minikube service react`
- Access the application using the provided IP.

4.5 Rationale

- Load balancing is implemented for the `react` service to distribute traffic effectively.
- A LoadBalancer type service ensures external access to the application.

5. Conclusion

This Kubernetes Deployment Architecture provides scalability, user access control, and load balancing for the docker-ethereum application. Each component helps ensure optimal performance, security, and ease of management.