Senior Platform Engineer Exercise (DCS)

Scenario

In this exercise, you will be tasked with deploying a high-availability transaction processing system between a Store and a Bank. The system simulates periodic transactions from the Store to the Bank, which processes them and writes the results to a database. Your primary goal is to ensure the high availability of the entire solution, focusing on both the database and the communication interface between the Store and Bank.

You will be given the initial code for the Store, Bank, and transaction logic. Your role will be to deploy and configure the solution on Kubernetes while maintaining a robust, highly available infrastructure.

Provided Components

You will receive:

1. Store Code: Sends transactions via HTTPS or a queue system (your choice) to the Bank.
2. Bank Code: Receives transactions, processes them, and writes the result into the database.
3. Database Schema: For recording the transactions.

Tasks

1. CI (Continuous Integration):
    - Create a CI pipeline (e.g., GitHub Actions) to automatically run unit tests and linting on every push. Ensure that the system is tested for functionality before deployment.
2. CD (Continuous Deployment):
    - Develop a CD pipeline to deploy the solution automatically to a Kubernetes cluster. The deployment should support rolling updates and rollback functionality in case of failure.
3. High Availability Database:
    - Ensure the database (e.g., PostgreSQL, MongoDB, etc.) is deployed using a StatefulSet on Kubernetes with high availability and failover capabilities.

Demonstrate how your deployment can handle database node failures without transaction loss.

4. High Availability Interface Between Store and Bank:
    - Create a resilient interface between the Store and Bank. You may choose between direct HTTPS communication or using a queue system (e.g., RabbitMQ). Demonstrate how this interface handles failure and ensure that no transactions are lost during communication failures. Show how the Store retries or queues transactions when the Bank is unavailable.

Requirements

The solution should be deployed on Kubernetes and include at least the following components:

1. Bank Deployment
2. Store Deployment
3. Database StatefulSet
4. (Optional) Queue System StatefulSet (depending on your design)

Focus on high availability:

- Use Kubernetes mechanisms (e.g., StatefulSets, ReplicaSets, readiness and liveness probes) to ensure that your system remains available even in the event of component failures.
- Ensure the database high availability.
- The communication layer (between Store and Bank) should be robust and handle failovers.
- Provide instructions for running and testing your solution locally using Minikube or a cloud-based Kubernetes service.

Submission

Your submission should include:

A GitHub repository with:

- The provided code (Store and Bank)
- Your deployment configurations (excluding terraform/ansible)

- CI/CD pipelines for testing and deploying the solution
- Documentation on how to set up and run the solution on Kubernetes

Bonus Question

1. Describe how you would scale this solution for a global financial system handling millions of transactions per minute. Discuss strategies for database scaling, partitioning, and inter-region communication.
2. Provide infra deployment code (terraform/ansible)