Explanation of Cloud-Native Principles Used

Overview

My Belote project follows key **cloud-native principles** to ensure that the system is scalable, portable, resilient, and easy to deploy in a cloud environment. These principles are reflected in the way the application is designed, built, and monitored.
Through containerization, stateless microservices, and a strong observability layer, the system can be deployed consistently across environments and scaled horizontally when needed.

1. Containerization

All major components of the system — the **Client**, **Server**, **Lobby Service**, and **Game Service** — are fully containerized using **Docker**. Each service has its own Dockerfile, defining its runtime environment, dependencies, and startup commands.

By isolating each microservice in its own container:

- The environment becomes **consistent** across all stages — from development to production.
- The system is **easily portable** and deployable to any cloud platform that supports containers (such as Kubernetes, AWS ECS, or Google Cloud Run).
- **Scaling** can be achieved by running multiple replicas of any individual service without interfering with others.

The container images are automatically built and pushed to **GitHub Container Registry (GHCR)** through a **GitHub Actions CI/CD pipeline**, ensuring versioned, reproducible builds. This approach aligns with cloud-native deployment models where containers are the primary unit of execution and scaling.

2. Stateless Services

The architecture is designed around **stateless microservices**, where each service handles requests independently without relying on in-memory session data or persistent local state.
For example:

- The **Server** only routes client messages and publishes events; it does not store game or player data internally.
- The **Lobby Service** manages matchmaking logic and uses **PostgreSQL** for persistence, allowing multiple lobby instances to run concurrently.
- The **Game Service** processes game logic based on incoming events and uses NATS for inter-service communication, making it replaceable or horizontally scalable.

Because no service relies on local memory for business-critical data, new instances can be started or terminated dynamically without disrupting user sessions. This design supports **elastic scaling** and **fault tolerance**, which are fundamental characteristics of cloud-native systems.

3. Observability

A core aspect of cloud-native development is building systems that are **observable** — capable of exposing meaningful insights into their internal state.
To achieve this, the Belote system integrates a complete observability stack consisting of:

- **Prometheus**, which collects time-series metrics from each service (e.g., request latency, CPU usage, network I/O).
- **NATS Exporter**, which exposes broker metrics such as publish/subscribe rates and queue performance.
- **Grafana**, which visualizes these metrics through real-time dashboards.

This setup provides full visibility into the behavior of distributed services and enables quick diagnosis of performance issues or failures. It also supports **data-driven scaling**, as load patterns can be observed and acted upon automatically in future cloud deployments (for example, through Kubernetes Horizontal Pod Autoscaling).

4. Future Cloud Integration Plan

The next step is to deploy (parts of) the system to a managed cloud environment. I will do this in the following ways:

1. **Kubernetes Deployment**: The main services (Server, Lobby Service, Game Service, and NATS) will be deployed to a Kubernetes cluster.
2. **Managed Database and Storage**: I will replace the current PostgreSQL instance with a cloud database.
3. **Cloud Monitoring and Logging**: The Prometheus and Grafana stack will be integrated with cloud monitoring tools.
4. **CI/CD Integration for Cloud Deployment**: The existing pipeline will be extended with a deployment stage to automatically deploy the newly built and pushed images to a Kubernetes cluster after successful tests.