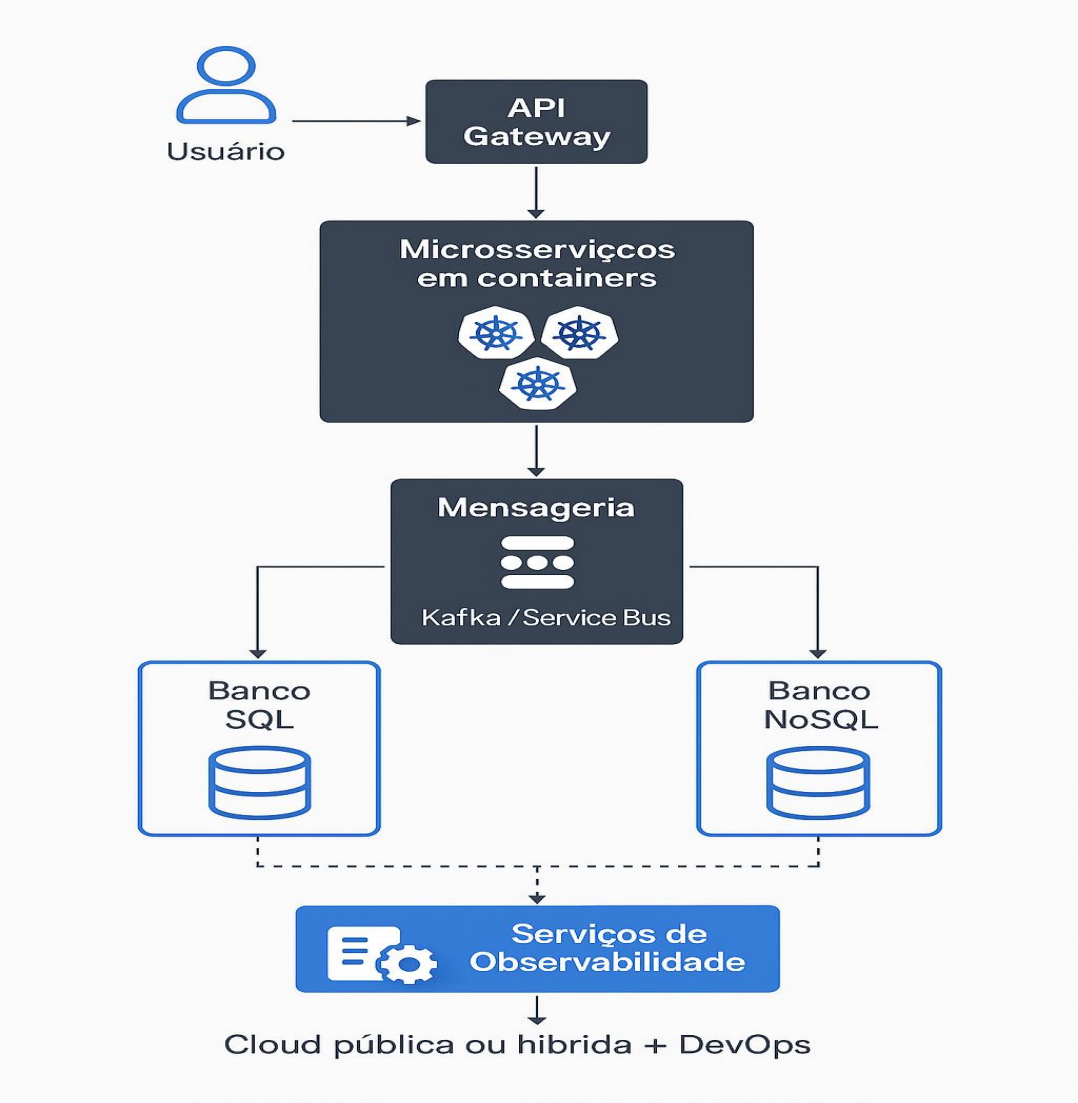
Modernização da arquitetura de um sistema bancário legado

Justificativa Técnica

- 🔹 Modularização e Escalabilidade
- Adoção de microsserviços permite escalar componentes independentemente.
- Containers (ex: AKS, EKS) facilitam replicação e resiliência.
- 🔹 Alta Disponibilidade
- Orquestração via Kubernetes + replica sets garantem tolerância a falhas.
- Serviços distribuídos mitigam pontos únicos de falha.
- 🔹 Custo-benefício
- Infra em cloud sob demanda reduz custos com servidores ociosos.
- Observabilidade e automação diminuem custos operacionais e retrabalho.
- 🔹 Governança e Dados
- Separação entre dados estruturados (SQL) e não estruturados (NoSQL).
- Estratégias de backup, replicação e compliance apoiadas por serviços gerenciados.
- 🔹 Manutenção e Evolução
- CI/CD garante entregas rápidas e rastreáveis.
- Versionamento e testes automatizados facilitam manutenção.



Usuário: Representa o cliente acessando os serviços via web, mobile ou APIs.

API Gateway: Camada de entrada única para chamadas externas — protege, roteia e monitora requisições.

Microsserviços em Containers (Kubernetes):

- Cada serviço é responsável por uma funcionalidade específica (ex: Pagamentos, Autenticação).
- Implantados em containers que rodam em clusters gerenciados.

Mensageria (Kafka ou Service Bus):

- Facilita comunicação assíncrona entre microsserviços.
- Suporta troca de eventos e desacoplamento entre componentes.

- **Banco SQL:** Usado para dados transacionais e estruturados (ex: movimentação bancária, registros de clientes).
- **Banco NoSQL:** Utilizado para dados não estruturados e alta escalabilidade (ex: logs, sessões, documentos).

Serviços de Observabilidade:

- Monitoramento (Prometheus), Logs estruturados (ELK Stack), e Rastreamento distribuído (OpenTelemetry).

Cloud Pública ou Híbrida + DevOps:

- Ambiente de infraestrutura com escalabilidade sob demanda.
- CI/CD automatizado para implantações contínuas e testes.

Análise Crítica da Arquitetura Atual

- **Pontos fracos identificados**:
- **Monolito** → baixa escalabilidade, deploy lento e complexidade para evoluções
- **Banco de dados único** → risco de gargalo, indisponibilidade impacta todo o sistema
- **Mensageria local** → pouca flexibilidade, limita integração com serviços externos ou cloud
- **Infra on-premise** → menor elasticidade, manutenção e escalabilidade mais caras

Pontos de Melhoria

- **Modularizar o monolito** → aplicar estratégia de strangler pattern para transição gradual
- **Desacoplar serviços** → adotar arquitetura baseada em microsserviços ou serviços distribuídos
- **Mensageria escalável** → migrar para soluções como Apache Kafka, RabbitMQ gerenciado ou serviços cloud nativos
- **Camadas bem definidas** → separação clara entre apresentação, domínio, aplicação e infraestrutura
- **Monitoramento integrado** → incluir observabilidade desde o design (logging, tracing, metrics)

Estratégias Complementares

- **Strangler Fig Pattern** para migrar gradualmente funcionalidades do monolito
- **Event-driven architecture** para desacoplamento de serviços e escalabilidade
- **Domain-driven design (DDD)** para organização de microsserviços por contexto de negócio
- **Cloud híbrida** → legado pode ir para VMs, novos serviços vão para containers em cloud

Alinhamento com os Critérios de Avaliação

Critério	Como é atendido
Escalabilidade	Microsserviços, cloud elástica, mensageria distribuída
Alta Disponibilidade	Infra redundante, containers orquestrados, replicação de dados
Custo-benefício	Uso sob demanda em cloud, modularidade reduz custos operacionais
Manutenção	CI/CD, versionamento, testes automatizados
Gestão de Dados	Camada de dados separada, replicação, backup automatizado
Observabilidade	Logging centralizado, tracing, alertas proativos