ADR: Arquitetura Baseada em Microserviços para um E-commerce

Data: 22/02/2025
Status: Aprovado
Autor: Clever Santoro Lopes

---

1. Contexto

Nossa startup está construindo uma plataforma de e-commerce que precisa ser **escalável, resiliente e modular**. A arquitetura deve permitir que cada parte do sistema evolua independentemente e suporte alto volume de transações.

Decidimos seguir o **Modelo C4** para documentar a arquitetura e garantir um design claro e compreensível para todos os stakeholders.

Requisitos-chave

- ✅ Alta disponibilidade e escalabilidade.
- ✅ Modularidade para facilitar manutenção e evolução.
- ✅ Facilidade para integrar novos serviços, como novos gateways de pagamento.
- ✅ Observabilidade para monitoramento e depuração.

---

2. Decisão

Optamos por uma **arquitetura baseada em microserviços** com os seguintes elementos:

Camadas da Arquitetura

1. **Frontend Web (React/Next.js)**
2. **API Gateway (NestJS ou Spring Cloud Gateway)**
3. **Microserviços Independentes**
    - **User Service:** Gerenciamento de usuários.
    - **Order Service:** Processamento de pedidos.
    - **Payment Service:** Integração com gateways de pagamento.
    - **Inventory Service:** Controle de estoque.
    - **Shipping Service:** Gerenciamento de entregas.
4. **Bancos de Dados**
    - **PostgreSQL** para dados transacionais.
    - **Redis** para cache e filas assíncronas.
5. **Observabilidade**
    - **Log Aggregation** (Elastic Stack, Loki).
    - **Tracing** (OpenTelemetry, Jaeger).
    - **Metrics** (Prometheus, Grafana).

Tecnologias Escolhidas

Componente	Tecnologia	Justificativa
Frontend	Next.js / React	SEO, SSR/SSG, UX moderna

Componente	Tecnologia	Justificativa
API Gateway	NestJS / Spring Gateway	Gerenciamento de rotas, segurança
Microserviços	Node.js (NestJS) ou Java (Spring Boot)	Performance, escalabilidade
Banco Relacional	PostgreSQL	Consistência de dados
Cache	Redis	Melhor resposta e escalabilidade
Mensageria	Kafka / RabbitMQ	Comunicação assíncrona entre serviços
Observabilidade	Prometheus, Grafana, Jaeger	Monitoramento proativo

---

3. Alternativas Consideradas

3.1 Monólito

🔴 **Rejeitado** – Não atende à escalabilidade e modularidade exigidas.

3.2 Microsserviços sem API Gateway

🔴 **Rejeitado** – Difícil gerenciar comunicação direta entre os serviços, aumento da complexidade.

3.3 Arquitetura Serverless

🟡 **Parcialmente Considerado** – Possível no futuro para algumas funções, mas alto acoplamento com provedores cloud pode limitar flexibilidade.

---

4. Consequências

✅ **Vantagens**

- Maior **escalabilidade** e **resiliência**.
- Manutenção facilitada com equipes independentes para cada serviço.
- Possibilidade de **desenvolvimento paralelo** sem grandes dependências.
- Uso de filas assíncronas para comunicação eficiente entre serviços.

⚠ **Desafios**

- Requer um **maior investimento inicial** em automação e infraestrutura.
- **Gerenciamento de consistência** entre bancos de dados distribuídos.
- A complexidade de **monitoramento e debugging** é maior do que em um monólito.

---

5. Próximos Passos

- 🚀 Configuração de **CI/CD** para automação de deploys.
- 🔍 Implementação de **observabilidade completa** (logging, tracing e métricas).
- 📈 Testes de carga para garantir a **resiliência e escalabilidade**.

🎯 Conclusão

Essa decisão de arquitetura possibilita uma plataforma **robusta, flexível e escalável** para nosso e-commerce, garantindo a evolução da startup sem gargalos tecnológicos.