




# ADR: Arquitetura do Order Service

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

## 1. Contexto

O **Order Service** é um dos principais microserviços do e-commerce, responsável pelo processamento de pedidos. Ele precisa garantir:

- ☒ Consistência transacional entre pagamento, estoque e envio.
- ☒ Alta disponibilidade e escalabilidade para lidar com picos de tráfego.
- ☒ Baixa latência para proporcionar uma boa experiência ao usuário.
- ☒ Tolerância a falhas para evitar pedidos incorretos ou duplicados.

## 2. Decisão

Optamos por uma arquitetura baseada em microserviços utilizando as seguintes tecnologias e padrões:

### Tecnologias

Componente	Tecnologia	Justificativa
Linguagem	Node.js (NestJS) ou Java (Spring Boot)	Performance e escalabilidade
Banco de Dados	PostgreSQL	Consistência transacional
Cache	Redis	Melhor tempo de resposta
Mensageria	Kafka / RabbitMQ	Comunicação assíncrona
Monitoramento	Prometheus + Grafana	Observabilidade e métricas


### Arquitetura do Order Service

O serviço segue um design orientado a eventos, garantindo que pedidos sejam processados de forma assíncrona e resiliente.

1. API Gateway → Recebe requisições de pedidos.
2. Order Service → Processa pedidos e gerencia a lógica de negócios.
3. Integração com Pagamentos → Usa Payment Client para confirmar o pagamento.
4. Validação de Estoque → Usa Inventory Client para verificar disponibilidade.
5. Geração de Eventos → Usa Kafka/RabbitMQ para processar pedidos assincronamente.

## 3. Alternativas Consideradas

### 3.1 Processamento Síncrono

-  Rejeitado – Aumentaria a latência e dependeria de respostas imediatas do Payment Service e Inventory Service.

### 3.2 Banco de Dados Centralizado

-  Rejeitado – Criaria um ponto único de falha e dificultaria a escalabilidade.




### 3.3 Arquitetura Event-Driven

- ☒ Aprovado – Permite processamento assíncrono, escalabilidade e maior resiliência.



---

## 4. Consequências

### Benefícios




-  Maior escalabilidade → O uso de eventos desacopla os serviços.
-  Menor tempo de resposta → Redis e filas assíncronas otimizam a performance.
-  Melhor tolerância a falhas → Se um serviço falhar, o pedido pode ser reprocessado.

### Desafios

-  Maior complexidade operacional → Requer monitoramento avançado e mecanismos de retry.
-  Gestão de consistência eventual → Sincronização de eventos pode ser desafiadora.

---

## 5. Próximos Passos

-  Implementação de SAGA Pattern para garantir consistência entre serviços.
-  Monitoramento com OpenTelemetry para rastrear transações distribuídas.
-  Testes de carga para validar a escalabilidade do serviço.

---

## Conclusão

Essa arquitetura garante que o Order Service seja confiável, escalável e resiliente, suportando o crescimento do e-commerce da startup.