# Proposta de Arquitetura

Sistema de Integração e Orquestração - Hub Distribuído

Desafio TechLead .NET Senior | Outubro/2025

Carmino Eduardo Vellutto

[eduardo.vellutto@gmail.com](mailto:eduardo.vellutto@gmail.com) / (16) 98151-6607

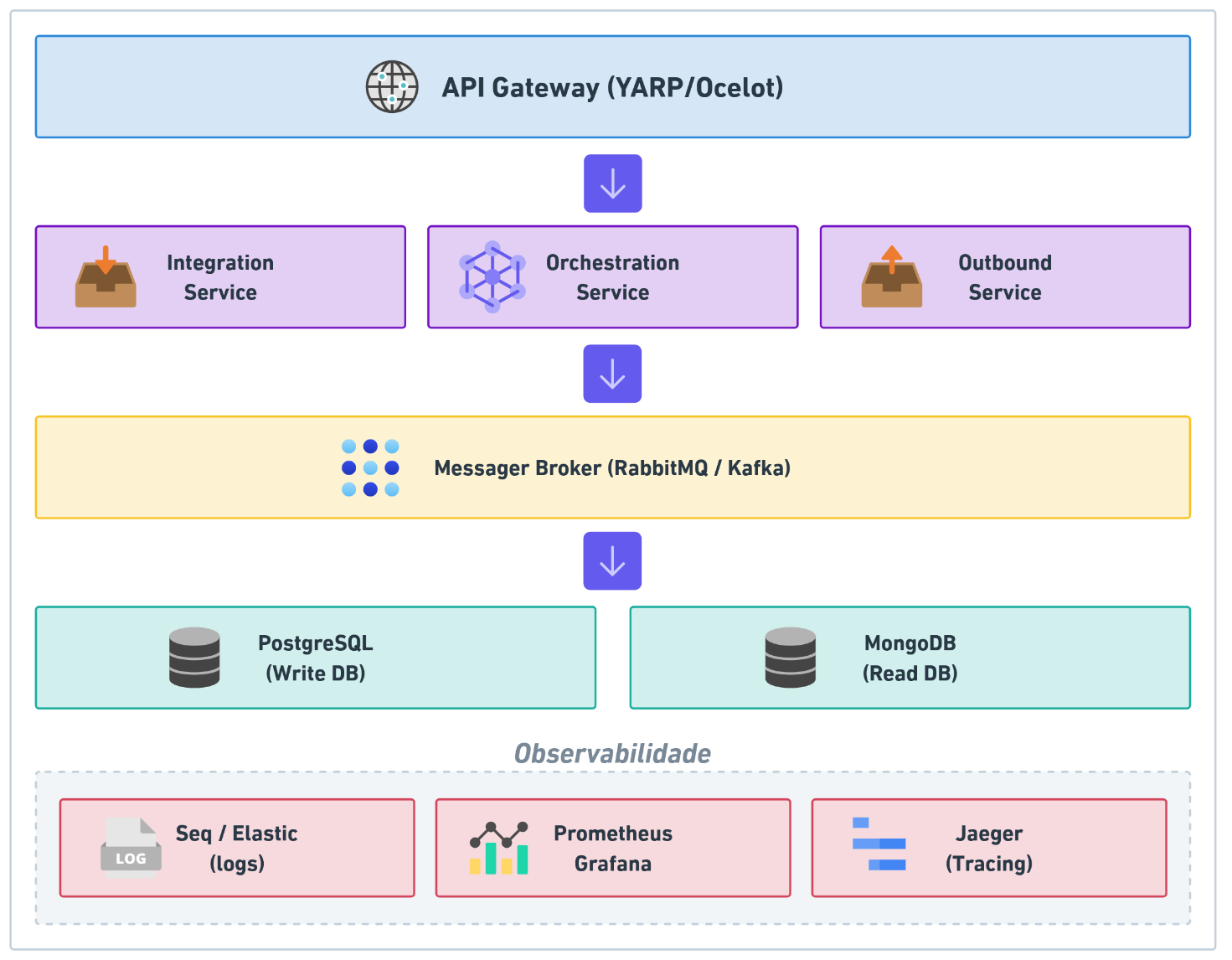
# 1 - Visão Geral da Solução

A arquitetura proposta implementa um **hub de integração baseado em microserviços**, utilizando o ecossistema .NET 8+ com padrões modernos de desenvolvimento distribuído. A solução garante alta disponibilidade, observabilidade completa e escalabilidade horizontal através de orquestração de containers e comunicação assíncrona.

### Princípios Arquiteturais

* **Event-Driven Architecture:** Comunicação assíncrona desacoplada
* **CQRS + Event Sourcing:** Separação de leitura/escrita com auditoria completa
* **API Gateway Pattern:** Ponto único de entrada com roteamento inteligente
* **Circuit Breaker & Retry:** Resiliência em comunicações externas
* **Idempotência:** Garantia de processamento único por requisição

# 2 - Diagrama de Arquitetura



# 3 - Stack Tecnológica

#### .NET 8+ (C#)

Runtime principal com suporte LTS, performance otimizada e native AOT

#### ASP.NET Core Web API

Framework para construção de APIs RESTful com OpenAPI/Swagger

#### MassTransit + RabbitMQ

Abstração para mensageria com suporte a Saga Pattern e retry policies

#### Entity Framework Core

ORM para persistência com suporte a migrations e change tracking

#### PostgreSQL

Banco relacional para write model com transações ACID

#### MongoDB

Banco NoSQL para read model otimizado (CQRS)

#### Redis

Cache distribuído e controle de idempotência

#### Polly

Biblioteca para resiliência (retry, circuit breaker, timeout)

#### Serilog

Logging estruturado com enrichers e sinks configuráveis

#### OpenTelemetry

Observabilidade unificada (traces, metrics, logs)

#### Docker + Kubernetes

Containerização e orquestração para escalabilidade horizontal

#### Azure Key Vault

Gerenciamento seguro de secrets e certificados

# 4 - Componentes da Arquitetura

### API Gateway

**Tecnologia:** YARP (Yet Another Reverse Proxy) ou Ocelot

**Responsabilidades:**

* Roteamento de requisições para microserviços apropriados
* Autenticação/Autorização centralizada (JWT + OAuth 2.0)
* Rate limiting e throttling por cliente
* Transformação de requisições/respostas
* Circuit breaking para serviços downstream
* Correlação de requisições (correlation-id)

**Justificativa:** YARP oferece alta performance, configuração via código e integração nativa com .NET, eliminando pontos de falha e simplificando deployments.

### Integration Service

**Responsabilidades:**

* Receber requisições externas via API REST ou gRPC
* Validar payloads usando FluentValidation
* Publicar eventos de integração no message broker
* Implementar idempotência usando Redis (deduplication)
* Registrar eventos de entrada no Event Store

**Padrões:** Mediator Pattern (MediatR), Repository Pattern, Unit of Work

### Orchestration Service

**Responsabilidades:**

* Consumir eventos de integração
* Executar regras de negócio complexas
* Orquestrar workflows usando Saga Pattern (MassTransit)
* Gerenciar estados de transação distribuída
* Aplicar compensações em caso de falha
* Publicar eventos de saída após processamento

**Padrões:** Saga Pattern (Orchestration-based), State Machine, Outbox Pattern

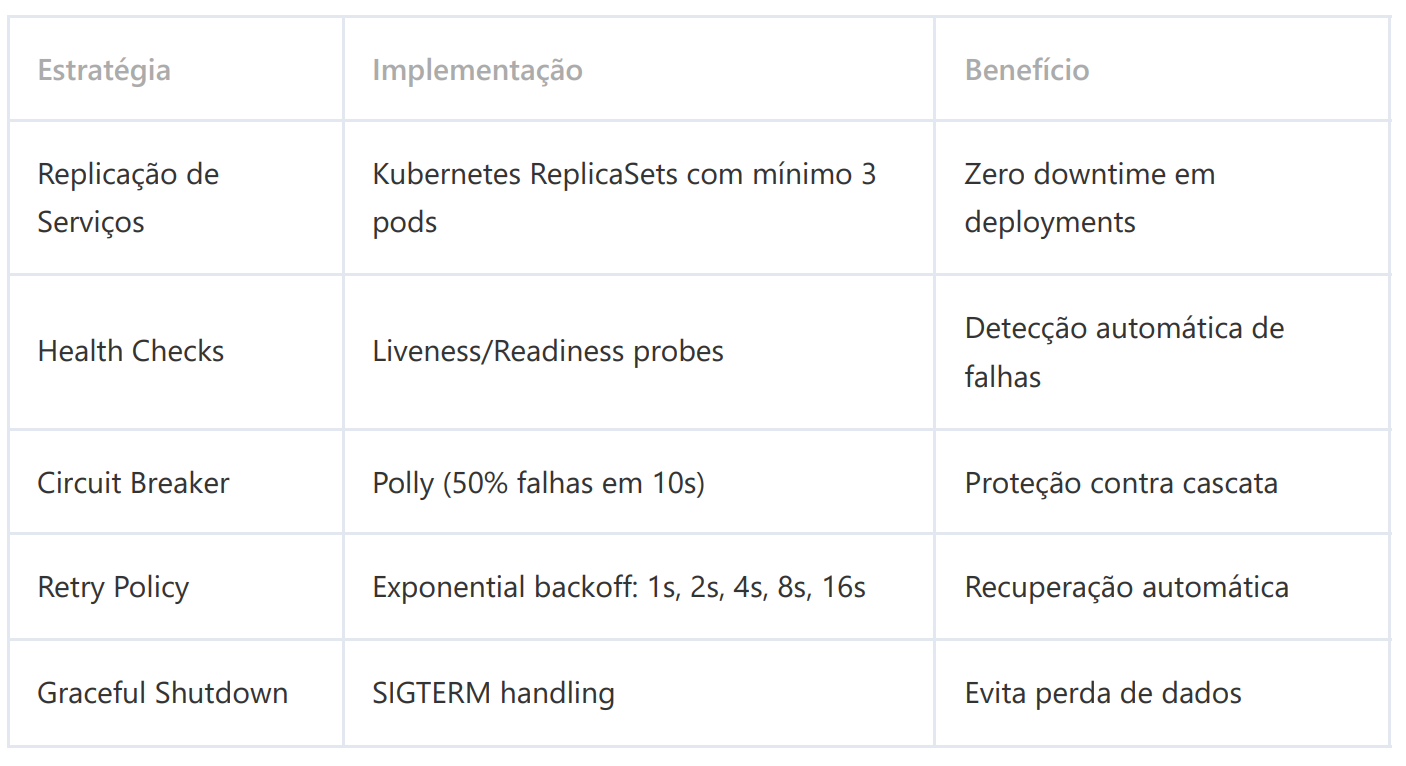
### Outbound Service

**Responsabilidades:**

* Consumir eventos de saída
* Integrar com APIs de sistemas terceiros
* Implementar retry com backoff exponencial (Polly)
* Circuit breaker para evitar cascata de falhas
* Registrar tentativas e resultados de integração
* Dead Letter Queue (DLQ) para falhas permanentes

# 5 - Estratégias de Qualidade

### Alta Disponibilidade e Tolerância a Falhas



### Observabilidade

#### Logs Estruturados (Serilog)

* **Formato:** JSON com campos padronizados
* **Níveis:** Debug (dev), Information (prod), Warning/Error (alertas)
* **Enrichers:** Machine name, thread id, correlation-id
* **Sinks:** Console (dev), Seq/Elasticsearch (prod)

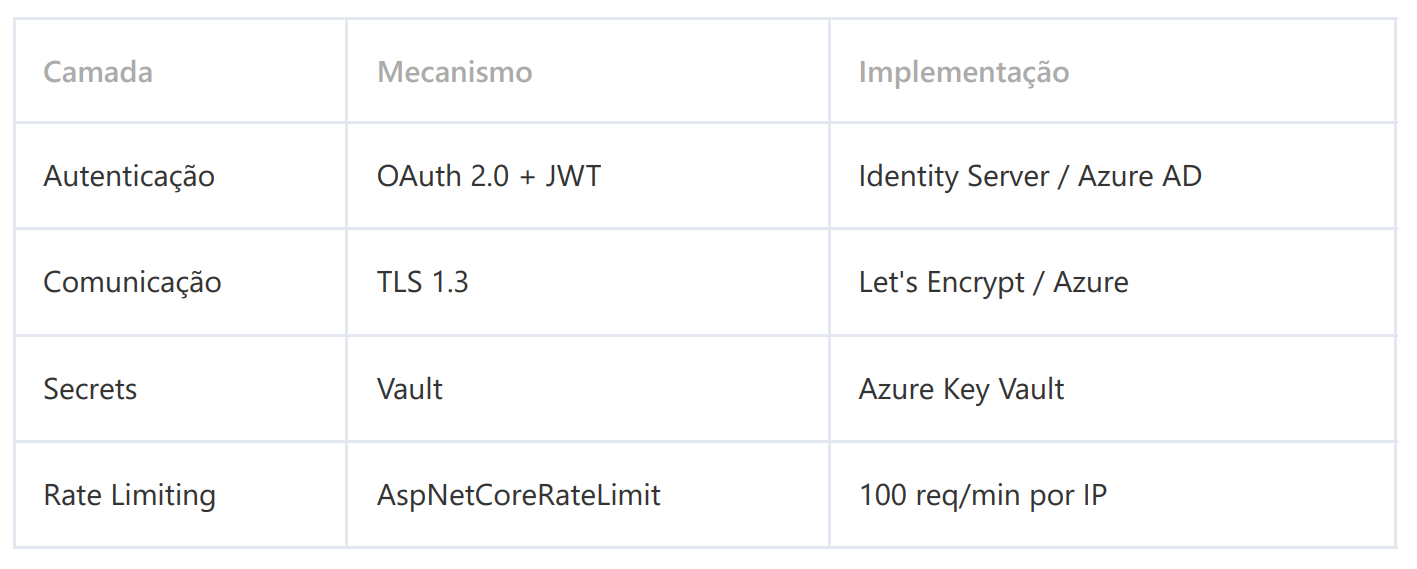
#### Métricas (Prometheus + Grafana)

* **RED Metrics:** Rate, Errors, Duration
* **Business Metrics:** Volume de integrações
* **Infrastructure:** CPU, memória, I/O

#### Distributed Tracing (Jaeger)

* Trace completo cross-service
* Identificação de gargalos
* W3C Trace Context

### Segurança



# 6 - Gestão de Workflows (Saga Pattern)

### Estados da Saga - Integração de Pedido

* **Initiated:** Pedido recebido, valida dados
* **Processing:** Aplica regras de negócio
* **PendingExternal:** Aguarda sistema externo
* **Completed:** Integração finalizada
* **Compensating:** Revertendo operações
* **Failed:** Falha permanente

**Características:** Persistência em PostgreSQL, timeouts configuráveis, compensating transactions automáticas

# 7 - Estratégias de Deploy e CI/CD

### Pipeline Automatizado

* **Build:** GitHub Actions / Azure DevOps
* **Testes:** xUnit + TestContainers
* **Quality Gates:** SonarQube (80% coverage)
* **Security:** Dependabot, Snyk, OWASP

### Deployment Strategy

* **Blue-Green:** Zero downtime
* **Canary:** 10% → 50% → 100%
* **Feature Flags:** LaunchDarkly

# 8 - Pontos de Atenção e Riscos

### Complexidade de Debugging Distribuído

* **Impacto:** Dificuldade em rastrear falhas cross-service
* **Mitigação:** Distributed tracing obrigatório, correlation IDs, dashboards centralizados

### Consistência Eventual

* **Impacto:** Dados temporariamente inconsistentes
* **Mitigação:** Design de UX considerando eventual consistency

### Message Broker como SPOF

* **Impacto:** Falha no broker paralisa sistema
* **Mitigação:** Clustering RabbitMQ (3+ nodes), filas duráveis

### Poison Messages

* **Impacto:** Mensagens inválidas bloqueiam consumidores
* **Mitigação:** DLQ, max retry (5x), alertas automáticos

# 9 - Evolução e Manutenibilidade

### Estratégias

* **Domain-Driven Design:** Bounded contexts claros
* **API Versioning:** Suporte a múltiplas versões
* **Contract Testing:** Pact para compatibilidade
* **Clean Architecture:** Separação de camadas

# 10 - Métricas de Sucesso (KPIs)



# 11 - Roadmap de Implementação

### Fase 1: Fundação (Mês 1-2)

* Setup de infraestrutura (K8s, PostgreSQL, Redis)
* API Gateway e Integration Service MVP

Observabilidade básica

### Fase 2: Orquestração (Mês 3-4)

* Message Broker (RabbitMQ)
* Orchestration Service (MassTransit)
* CQRS + Event Sourcing

### Fase 3-5: Resiliência e Otimização (Mês 5-8)

* Circuit breaker e retry refinados
* Distributed tracing avançado
* Autoscaling e performance tuning

# 12 - Comparação com Alternativas

### Por que não um Monolito?

Microserviços permitem escalabilidade granular, deploys independentes e menor risco em mudanças.

### Por que não Serverless?

Cold start adiciona latência imprevisível. Containers oferecem mais controle e previsibilidade.