Arquitetura de Processamento de Eventos em Tempo Real com NestJS

Uma solução completa baseada em microserviços para processamento assíncrono de eventos com alta disponibilidade, escalabilidade horizontal e resiliência.

Visão Geral da Arquitetura

API REST

Recebe eventos HTTP

RabbitMQ

Fila de mensagens assíncronas

Batch Processing

Processamento em lote

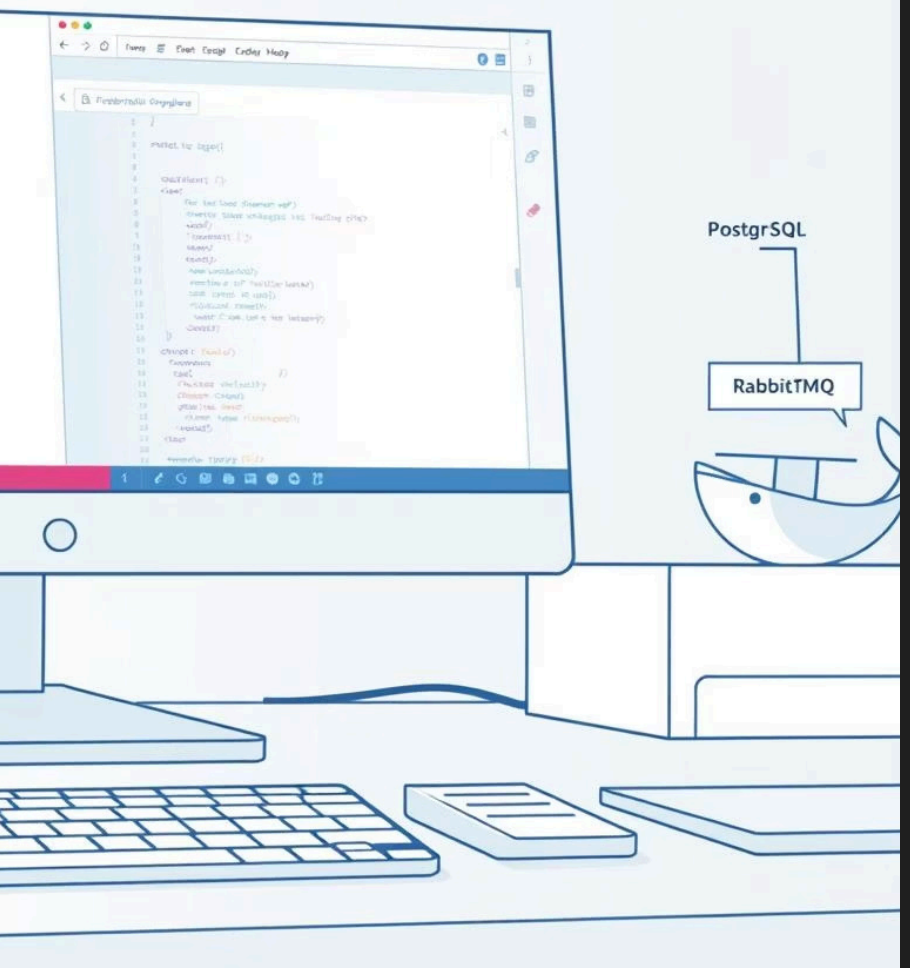
PostgreSQL

Armazenamento persistente

Event Emitter

Eventos internos e métricas

O projeto implementa uma **arquitetura orientada a eventos** (Event-Driven Architecture) que permite processamento assíncrono e escalável de dados em tempo real.



Stack Tecnológica

Framework e Infraestrutura

- NestJS (Node.js) como framework principal
- RabbitMQ como message broker
- PostgreSQL para armazenamento persistente
- TypeORM como ORM
- Docker + Docker Compose para containerização
- Class Validator para validação

Dependências Principais

- @nestjs/common, @nestjs/core: ^11.0.1
- @nestjs/event-emitter: ^3.0.1
- @nestjs/microservices: ^11.1.6
- amqp-connection-manager: ^4.1.14
- typeorm: ^0.3.26
- pg: ^8.16.3

Estrutura de Módulos

App Module

Módulo principal que configura conexões com banco de dados, RabbitMQ e importa os demais módulos da aplicação.

Events Module

Contém EventsController para receber eventos via API REST e EventsConsumer para consumir mensagens do RabbitMQ.

Batch Processing Module

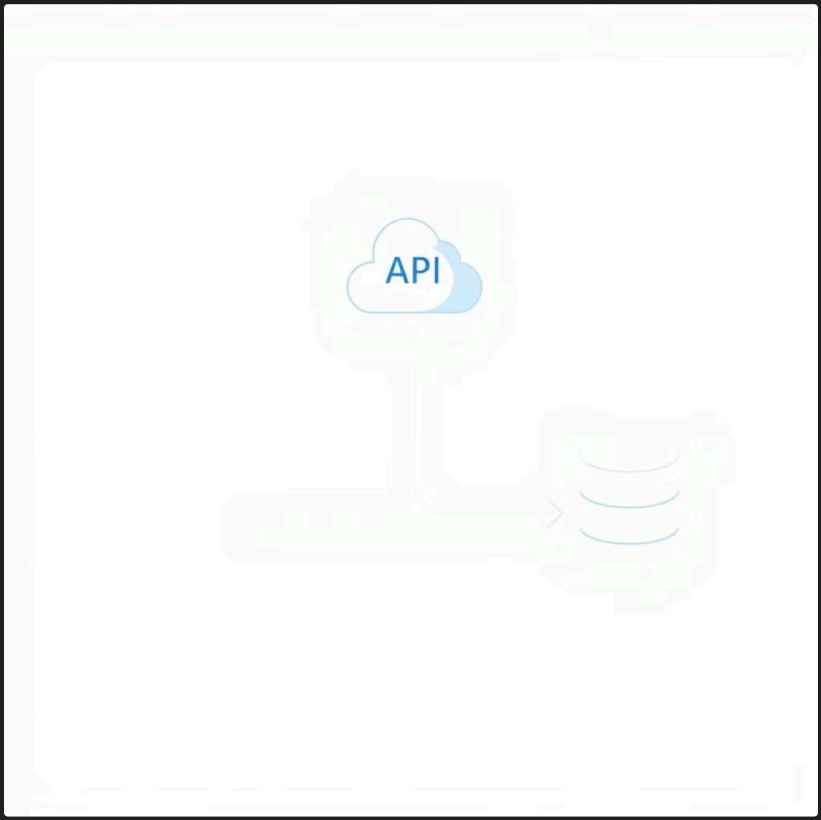
Gerencia buffer de eventos, processamento em lote e métricas de processamento.

RabbitMQ Module

Módulo global que gerencia conexões e publicação de mensagens no broker.

A arquitetura modular do NestJS permite separação clara de responsabilidades e facilita a manutenção do código.

Fluxo de Processamento Detalhado



Recepção de Eventos

API REST recebe eventos e envia para o RabbitMQ

Publicação no RabbitMQ

Eventos são publicados na fila de mensagens

Consumo e Buffer

Eventos são armazenados em buffer até atingir limite

Processamento Agendado

Lotes são processados por tamanho ou tempo (30s)

Padrões Arquiteturais Implementados

Event Sourcing Pattern

- Eventos são capturados e armazenados sequencialmente
- Cada evento representa uma mudança de estado

Batch Processing Pattern

- Eventos agrupados em lotes para processamento eficiente
- Triggers: tamanho máximo (100) ou tempo (30s)

Dead Letter Queue Pattern

- Mensagens com falha são enviadas para fila específica
- Configuração via x-dead-letter-exchange

Circuit Breaker / Retry Pattern

- Tentativas automáticas de reprocessamento
- Limite configurável de retentativas

Estes padrões garantem resiliência, eficiência e confiabilidade no processamento de eventos.

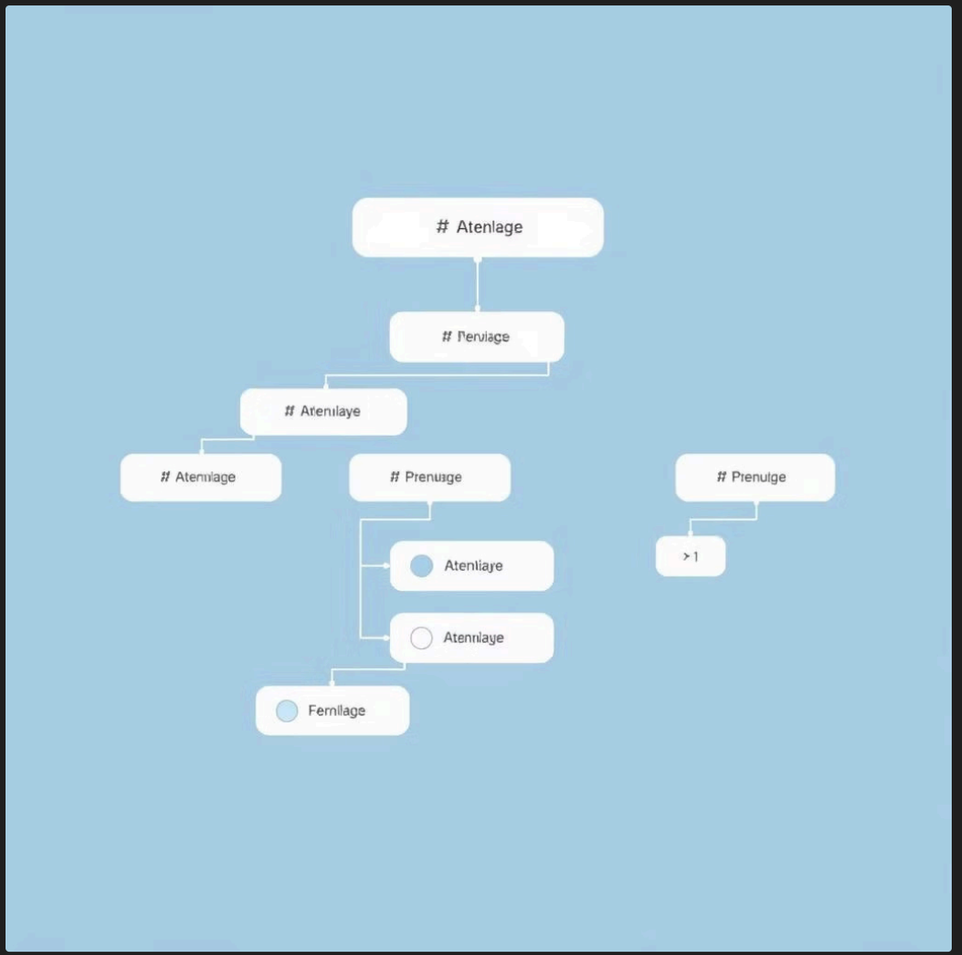
Modelo de Dados

UserInteraction Entity

```
@Entity('user_interactions')
@Index(['userId', 'eventType'])
export class UserInteraction {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  userId: string;

  @Column()
  eventType: string;

  @Column('jsonb')
  eventData: any;

  @Column()
  sessionId: string;

  @CreateDateColumn()
  timestamp: Date;

  @Column({ default: false })
  processed: boolean;
}
```

Características

- Estrutura otimizada para consultas por usuário e tipo de evento
- Suporte a dados JSON com campo eventData
- Rastreamento de sessão para análise de comportamento
- Flag de processamento para controle de estado
- Timestamp automático para ordenação cronológica



Infraestrutura com Docker

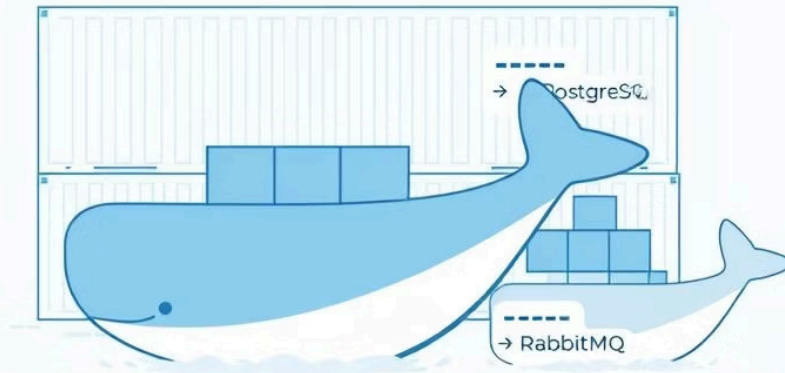
Componentes Containerizados

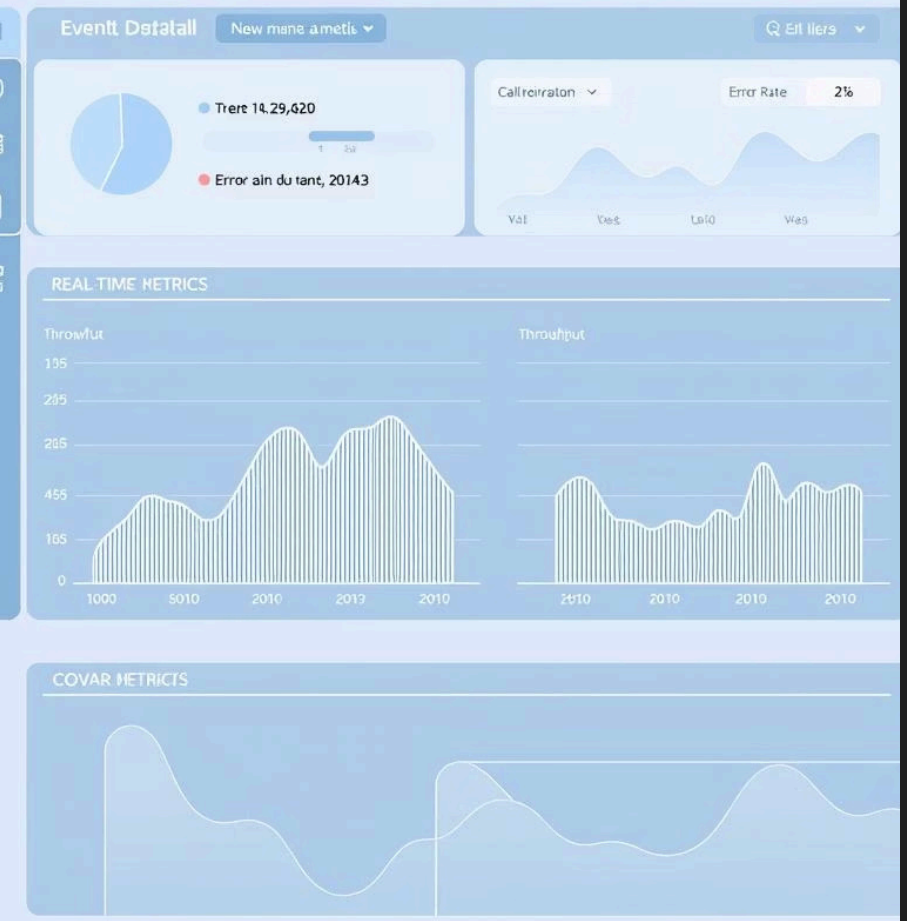
- Aplicação NestJS (porta 3000)
- PostgreSQL 15 (porta 5432)
- RabbitMQ com interface de gerenciamento (portas 5672 e 15672)

A infraestrutura completa é definida em Docker Compose, permitindo fácil implantação em qualquer ambiente.

Variáveis de Ambiente

- NODE_ENV: development/production
- Configurações de banco de dados (host, porta, credenciais)
- URL do RabbitMQ e nome da fila
- MAX_RETRY_ATTEMPTS: número máximo de retentativas





Observabilidade e Monitoramento

100%

Cobertura de Logs

Logs estruturados em todos os componentes críticos

3

Níveis de Log

Debug, Info e Error para diferentes contextos

5+

Métricas Chave

Lotes processados, eventos por tipo, taxa de sucesso

O sistema implementa métricas detalhadas através do Event Emitter, permitindo monitoramento em tempo real do processamento de eventos e integração com sistemas externos de observabilidade.

Vantagens e Casos de Uso

Vantagens da Arquitetura

- Escalabilidade Horizontal: Adição de mais consumers
- Resiliência: Dead Letter Queue e retry automático
- Performance: Processamento em lote otimiza I/O
- Desacoplamento: Componentes independentes
- Observabilidade: Logs estruturados e métricas
- Transações ACID: Consistência no PostgreSQL

Casos de Uso Ideais

- E-commerce: Tracking de interações de usuário
- Analytics: Coleta de eventos de aplicações
- IoT: Processamento de telemetria de sensores
- Auditoria: Log de ações em sistemas críticos
- Real-time Dashboards: Alimentação de dashboards

Esta arquitetura implementa as melhores práticas para sistemas de **alta disponibilidade**, **processamento em tempo real** e **escalabilidade horizontal**.