Desenvolver um serviço de envio de notificações em Java envolve várias etapas, incluindo o planejamento da arquitetura do serviço, a implementação do código, e a configuração de um ambiente de execução com Docker Compose. Vamos abordar esse projeto passo a passo.

**Etapa 1: Planejamento da Arquitetura**

**Componentes da Arquitetura**

1. **API REST**: Interface para receber requisições de notificação.
2. **Serviço de Processamento**: Processa as requisições e envia as notificações.
3. **Banco de Dados**: Armazena registros das notificações.
4. **Fila de Mensagens (Opcional)**: Para lidar com alta carga e garantir a entrega assíncrona das notificações.

**Tecnologias Sugeridas**

* **Spring Boot**: Para criar a API REST e o serviço de processamento.
* **PostgreSQL**: Como sistema de banco de dados.
* **RabbitMQ**: Como sistema de fila de mensagens (opcional).
* **Docker e Docker Compose**: Para containerização e orquestração local.

**Etapa 2: Desenho da Arquitetura**

Você pode criar um diagrama simples usando ferramentas como Draw.io, Lucidchart, ou mesmo papel e caneta. Aqui está um exemplo de como poderia ser:

css

Copy code

[Cliente] ---> [API REST (Spring Boot)] ---> [RabbitMQ] ---> [Serviço de Processamento] ---> [PostgreSQL] | +---> [Docker Compose] ---> [Containerização e Orquestração]

**Etapa 3: Configuração do Ambiente de Desenvolvimento**

**Instalação de Ferramentas**

* JDK para Java (recomenda-se Java 11 ou superior)
* IDE de sua preferência (Eclipse, IntelliJ IDEA, etc.)
* Docker e Docker Compose

**Etapa 4: Criação do Projeto**

**Inicializando o Projeto com Spring Initializr**

* Acesse [Spring Initializr](https://start.spring.io/)
* Escolha a versão do Spring Boot e selecione as dependências: Spring Web, Spring Data JPA, PostgreSQL Driver, RabbitMQ (opcional), Spring Boot Actuator.
* Gere e baixe o projeto, depois abra-o na sua IDE.

**Estrutura de Classes**

1. **Model**: Classes de domínio que representam entidades (e.g., **Notification**).
2. **Repository**: Interfaces para interagir com o banco de dados.
3. **Service**: Serviços que contêm a lógica de negócios.
4. **Controller**: Controladores que manipulam as requisições HTTP.

**Exemplo de uma Classe de Modelo**

java

Copy code

@Entity public class Notification { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; private String message; private LocalDateTime sendTime; private boolean sent; // Getters e Setters }

**Etapa 5: Dockerização**

**Dockerfile para a Aplicação**

dockerfile

Copy code

FROM openjdk:11 COPY target/notificationservice.jar notificationservice.jar EXPOSE 8080 ENTRYPOINT ["java", "-jar", "/notificationservice.jar"]

**Docker Compose**

yaml

Copy code

version: '3.8' services: app: build: . ports: - "8080:8080" depends\_on: - db db: image: postgres environment: POSTGRES\_DB: notificationdb POSTGRES\_USER: user POSTGRES\_PASSWORD: password

**Etapa 6: Implementação e Testes**

Implemente o código das classes, serviços e controladores. Use o Postman ou qualquer outro cliente HTTP para testar a API.

**Etapa 7: Execução com Docker Compose**

* Execute **docker-compose up --build** para construir e executar o serviço e o banco de dados.

**Conclusão**

Este exemplo aborda a criação básica de um serviço de notificações usando Java, Spring Boot, Docker, e opcionalmente RabbitMQ para filas de mensagens. Para sistemas em produção, você deve incluir configurações de segurança, testes mais robustos, e considerações sobre CI/CD, monitoramento e escalabilidade.