ESTI – ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Matéria
TESTE DE PERFORMANCE

Rafael Soares de Souza

https://github.com/RafaellSouzza/Gerenciador_de_Usuarios

Documentação do Projeto - Gerenciador de Usuários

1. Visão Geral do Projeto

Este projeto tem como objetivo desenvolver e implementar um sistema de gerenciamento de usuários baseado em uma arquitetura de microserviços. O sistema é composto por três APIs principais: **UsuariosAPI**, **RelatoriosAPI** e **AutenticacaoAPI**, cada uma responsável por uma funcionalidade específica. Inicialmente, o ambiente de desenvolvimento será gerido localmente usando **Docker Compose**, e, em seguida, os serviços serão migrados para um cluster Kubernetes, utilizando o **Minikube** para testes locais.

O uso de **Docker Compose** facilita o gerenciamento dos contêineres durante o desenvolvimento, enquanto a migração para **Kubernetes** visa proporcionar uma solução escalável e robusta para ambientes de produção.

2. Objetivos

- Implementar três APIs independentes, cada uma rodando em contêineres separados.
- Orquestrar os contêineres localmente utilizando Docker Compose.
- Migrar as APIs para um ambiente Kubernetes, utilizando Minikube como ferramenta de teste local.
- Demonstrar a capacidade de escalar os serviços e garantir sua integração no Kubernetes.

3. Tecnologias Utilizadas

- **Java**: Linguagem de programação usada para desenvolver as APIs.
- **Docker**: Plataforma de contêinerização utilizada para criar e gerenciar os contêineres das APIs.
- **Docker Compose**: Ferramenta para orquestrar múltiplos contêineres localmente.
- **Minikube**: Ferramenta usada para rodar Kubernetes em um ambiente local, simulando um cluster.
- **Kubernetes**: Plataforma de orquestração de contêineres usada para gerenciar e escalar as APIs.
- **Kompose**: Ferramenta utilizada para converter arquivos de configuração do Docker Compose em manifestos Kubernetes.
- **Maven**: Gerenciador de dependências e build utilizado para compilar e construir os projetos Java.

4. Arquitetura do Sistema

O sistema foi desenvolvido com uma arquitetura baseada em **microserviços**, onde cada API é responsável por uma função específica. A arquitetura foi projetada para permitir que cada serviço seja desenvolvido, testado e implantado independentemente, garantindo maior flexibilidade e escalabilidade. Abaixo estão os principais componentes do sistema:

- **UsuariosAPI**: Responsável por gerenciar o cadastro, edição, exclusão e consulta de usuários.
- **RelatoriosAPI**: Gera relatórios com base nas informações armazenadas sobre os usuários.
- **AutenticacaoAPI**: Gerencia a autenticação e autorização de usuários, garantindo que apenas usuários autenticados tenham acesso às funcionalidades do sistema.

Essas três APIs são independentes, mas interagem entre si para fornecer uma solução completa de gerenciamento de usuários. Cada API é containerizada e orquestrada inicialmente com Docker Compose, e posteriormente no Kubernetes.

5. Configuração e Execução com Docker Compose

Para o ambiente de desenvolvimento local, utilizamos **Docker Compose**. Ele permite que todos os serviços (APIs) sejam executados simultaneamente em contêineres separados. Essa abordagem simplifica o processo de desenvolvimento e teste, garantindo que cada serviço esteja isolado, mas ainda assim funcionando como parte de um todo.

Passos para execução local:

1. **Instale o Docker e o Docker Compose**: Certifique-se de que as ferramentas necessárias estão instaladas e configuradas corretamente.
2. **Clonar o repositório do projeto**: Obtenha o código-fonte e navegue até o diretório raiz onde o arquivo de configuração do Docker Compose está localizado.
3. **Construção e execução dos contêineres**: O Docker Compose irá construir as imagens das APIs e criar os contêineres que executarão os serviços.
4. **Testar o funcionamento das APIs**: Após a execução, as APIs estarão disponíveis em suas respectivas portas, permitindo que o desenvolvedor faça requisições HTTP para testar suas funcionalidades.

Benefícios do Docker Compose:

- **Simplicidade**: Permite que os desenvolvedores rodem vários serviços em contêineres separados com apenas um comando.
- **Ambiente isolado**: Cada serviço roda em seu próprio contêiner, garantindo que não haja interferência entre eles.

6. Migração para Kubernetes

Após o desenvolvimento e testes locais, a arquitetura é migrada para o **Kubernetes**. O Kubernetes oferece recursos avançados de orquestração, permitindo que o sistema seja escalado automaticamente conforme a demanda.

Por que usar Kubernetes?

O Kubernetes é uma ferramenta poderosa para gerenciar ambientes de produção, pois facilita:

- **Escalabilidade**: O Kubernetes pode escalar automaticamente as APIs de acordo com o tráfego.
- **Alta disponibilidade**: Garante que os serviços estejam sempre disponíveis, distribuindo os contêineres em diferentes nós.
- **Gerenciamento de configuração e atualização**: Permite que as aplicações sejam atualizadas sem tempo de inatividade.

Ferramenta de migração: Kompose

Para converter as configurações do Docker Compose em manifestos Kubernetes (arquivos YAML), utilizamos a ferramenta **Kompose**. Com ela, podemos transformar as definições do Docker Compose em deployments e serviços que são facilmente aplicáveis no Kubernetes.

Passos para migração:

1. **Instalação do Minikube e Kompose**: O Minikube é usado para rodar um cluster Kubernetes localmente, enquanto o Kompose converte o arquivo Docker Compose.
2. **Conversão das definições**: O arquivo Docker Compose é convertido em manifestos Kubernetes.
3. **Implantação no Kubernetes**: Os manifestos são aplicados no cluster Kubernetes via `kubectl`, criando os pods, serviços e deployments necessários para rodar as APIs.

Vantagens da migração para Kubernetes:

- **Ambiente escalável**: O Kubernetes ajusta automaticamente o número de réplicas de cada API com base na carga.
- **Facilidade de gerenciamento**: O Kubernetes permite que as atualizações sejam feitas de maneira contínua, sem causar interrupções no serviço.
- **Monitoramento e resiliência**: Se um contêiner falhar, o Kubernetes automaticamente recria o pod, garantindo maior resiliência.

7. Monitoramento e Manutenção

Após a implantação das APIs no Kubernetes, é importante monitorar o estado dos serviços e manter o ambiente. O Kubernetes oferece ferramentas integradas para

monitoramento e logging, permitindo que os desenvolvedores acompanhem o desempenho do sistema e identifiquem possíveis problemas em tempo real.

Monitoramento:

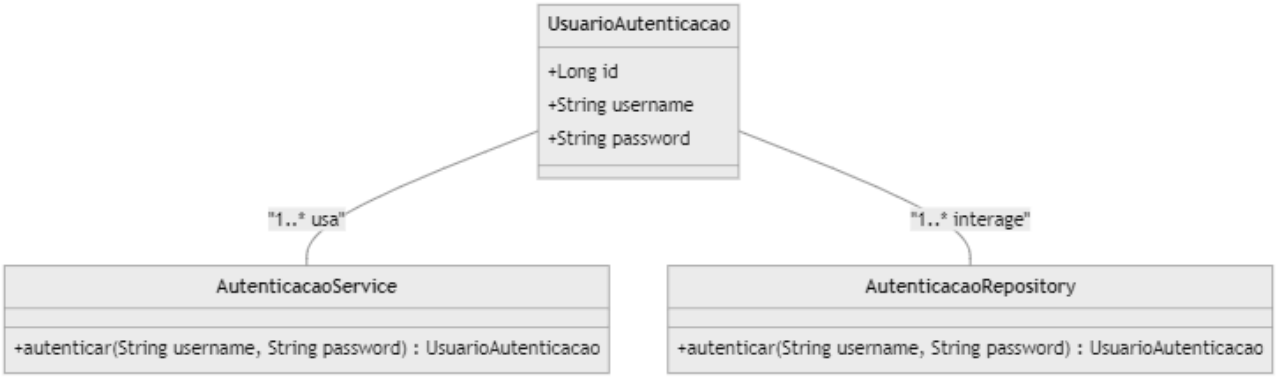
- **Logs dos pods**: Permitem visualizar os eventos de cada pod individualmente.
- **Status dos serviços**: Através do `kubectl`, é possível verificar o estado atual dos serviços, deployments e pods.

Manutenção:

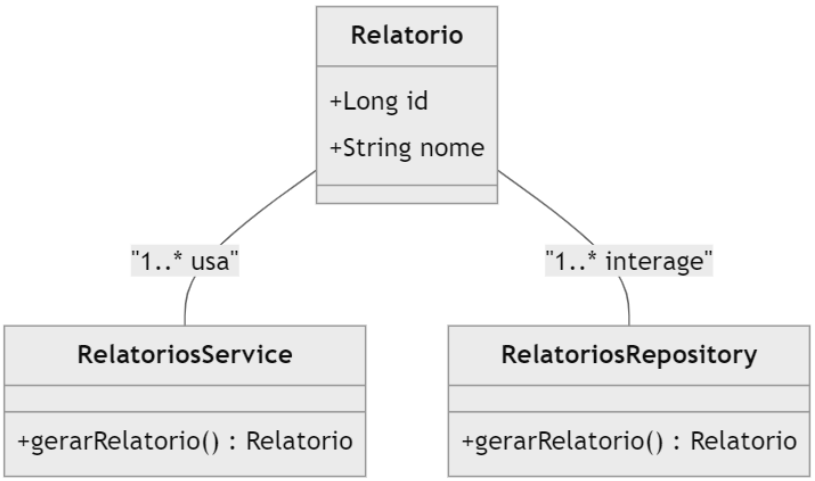
- **Escalonamento**: O Kubernetes facilita o aumento ou a redução do número de réplicas dos serviços.
- **Atualizações contínuas**: Possibilita aplicar novas versões das APIs sem causar downtime.

Diagrama de Classes

Autenticação



Relatórios



Usuários