Documentação de Implantação da Aplicação To Do List no Kubernetes (Minikube)

Caio Cadini - 800383

Lucca Couto Barberato - 800257

1. Introdução

Esta documentação descreve os componentes da aplicação To Do List e os artefatos Kubernetes utilizados para sua implantação no ambiente Kubernetes (Minikube). A aplicação consiste em um backend desenvolvido com **Flask** (Python), um frontend desenvolvido com **React** e **Vite**, e um banco de dados **SQLite**. Serão apresentados os componentes, os artefatos de deployment, serviços e o uso de **Ingress** para a exposição dos serviços.

---

2. Arquitetura da Aplicação

A aplicação é composta pelos seguintes componentes:

- **Frontend**: Desenvolvido em React + Vite, responsável pela interface com o usuário, onde o usuário pode adicionar, visualizar e remover tarefas.
- **Backend**: Desenvolvido em Flask (Python), responsável pela lógica da aplicação. Este componente expõe uma API REST que gerencia as tarefas da To Do List, comunicando-se com o banco de dados.
- **Banco de Dados (SQLite)**: Armazena os dados das tarefas. Está configurado para ser persistido em um volume montado no container SQLite.

---

3. Componentes Kubernetes

A seguir estão os principais artefatos Kubernetes utilizados para a implantação da aplicação.

3.1 Backend (Flask)

O backend é implantado como um **Deployment** e exposto através de um **Service**.

- **Arquivo: `backend-deploy.yaml`**
  - Este deployment cria um pod com o container do backend (imagem Docker).
  - O pod monta o volume persistente (`sqlite-pvc`) onde o banco de dados SQLite está armazenado.
  - Variável de ambiente `DATABASE_URL` configurada para o SQLite.
- **Arquivo: `backend-service.yaml`**
  - O serviço (`Service`) mapeia o tráfego externo para o backend na porta 8000.

3.2 Frontend (React + Vite)

O frontend é implantado como um **Deployment** e exposto através de um **Service** do tipo **LoadBalancer**.

- **Arquivo: `frontend-deploy.yaml`**
  - Este deployment cria um pod com o container do frontend, que escuta na porta 80.
- **Arquivo: `frontend-service.yaml`**
  - O serviço (`Service`) expõe a aplicação do frontend na porta 80.

3.3 Banco de Dados (SQLite)

O banco de dados SQLite é gerenciado como um deployment simples com um volume persistente.

- **Arquivo: `bd-deploy.yaml`**
  - Este deployment cria um pod com o container Alpine, que roda o SQLite.
  - Um volume persistente é montado para armazenar o banco de dados SQLite.
- **Arquivo: `pvc.yaml`**
  - Este Persistent Volume Claim (PVC) garante que o armazenamento do banco de dados seja persistente entre os pods, permitindo que o banco de dados sobreviva a recriações dos pods.

---

4. Ingress para Publicação dos Serviços

Para garantir que os serviços sejam acessíveis externamente via Minikube, foi configurado um **Ingress** que publica os serviços na URL `k8s.local`.

- **Arquivo: `ingress.yaml`**
  - O Ingress direciona as requisições:
    - **Para o frontend (/) na porta 80.**

5. Script de Automação

5.1 Build dos Containers

Não foi necessário criar um script de build, pois disponibilizamos as imagens Docker no Dockerhub, facilitando a implantação da aplicação no Kubernetes.

5.2 Deploy da Aplicação

Um segundo script (`deploy.sh`) automatiza a implantação da aplicação no Kubernetes.

```
kubectl apply -f pvc.yaml

kubectl apply -f bd-deploy.yaml

kubectl apply -f backend-deploy.yaml
kubectl apply -f backend-service.yaml

kubectl apply -f frontend-deploy.yaml
kubectl apply -f frontend-service.yaml

kubectl apply -f ingress.yaml

echo "Aplicação implantada com sucesso no Minikube"
```