

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Curso de Bacharelado em Ciência da Computação

Implantação da Aplicação *TodoApi* com Kubernetes

Relatório Técnico - DEVOPS

Autor: Pedro Cassiano Coleone
RA: 793249

São Carlos – SP
Julho de 2025

Sumário

1	Introdução	2
2	Visão Geral da Aplicação	2
3	Preparação do Ambiente	3
3.1	Ferramentas Utilizadas	3
3.2	Minikube	3
4	Helm Chart	3
4.1	Estrutura do Chart	3
4.2	Configuração via <code>values.yaml</code>	4
5	Artefatos Kubernetes Gerados pelo Helm	4
5.1	Deployment (Backend)	4
5.2	Service (Frontend)	4
5.3	Ingress	5
6	Automação do Deploy	5
7	Acesso e Validação	6
8	Conclusão	6

1 Introdução

Este relatório técnico apresenta o processo de migração da aplicação *TodoApi*, originalmente executada via Docker Compose, para um ambiente Kubernetes local, utilizando Minikube e Helm Charts. A proposta abrange a orquestração dos serviços com Deployments e Services, a exposição da aplicação via Ingress e a automação completa do processo de deploy.

Além da reorganização da infraestrutura, a utilização do Helm possibilita padronização, reusabilidade e controle de versões sobre os manifestos Kubernetes, por meio de templates e arquivos de valores configuráveis.

O objetivo principal é demonstrar, de forma prática e documentada, como empacotar, implantar e validar uma aplicação completa no Kubernetes com uso de ferramentas modernas de DevOps.

O código-fonte e os artefatos utilizados encontram-se disponíveis no repositório público: <https://github.com/pedro-coleone/ToDoApi-kubernetes-deployment>

2 Visão Geral da Aplicação

A *TodoApi* adota uma arquitetura *three-tier*, composta por:

- **Frontend:** SPA (Vite + React) servida por NGINX.
- **Backend:** API RESTful desenvolvida com ASP.NET Core 8.0.
- **Banco de Dados:** PostgreSQL 15.

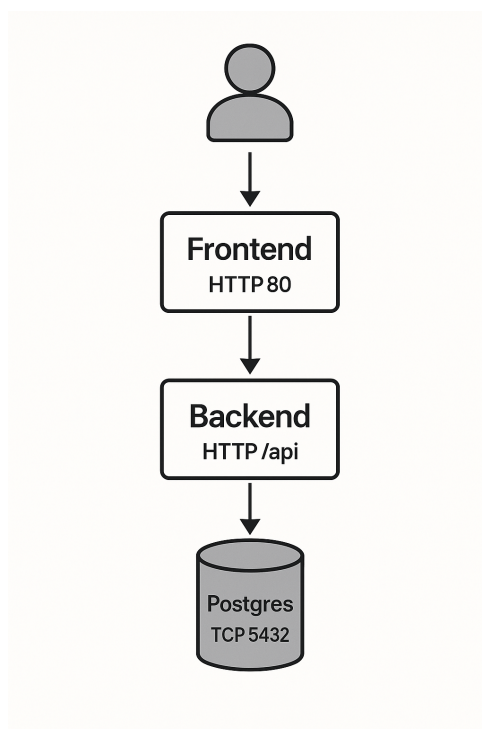


Figura 1: Arquitetura lógica da aplicação

3 Preparação do Ambiente

3.1 Ferramentas Utilizadas

- **Kubernetes / Minikube:** orquestração local.
- **Helm:** gerenciamento de pacotes (charts).
- **Docker:** construção das imagens.
- **NGINX Ingress Controller:** roteamento de tráfego HTTP.
- **Bash / PowerShell:** automação de deploy.

3.2 Minikube

Para iniciar o ambiente, foi utilizado o Minikube com driver Docker. Após a criação do cluster, habilitou-se o Ingress Controller nativo e configurou-se o terminal para direcionar o Docker local para dentro do nó do cluster:

1. Iniciar o cluster:

```
1 minikube start --driver=docker
```

2. Ativar o Ingress Controller:

```
1 minikube addons enable ingress
```

3. Usar o daemon Docker interno do Minikube:

```
1 eval $(minikube -p minikube docker-env)
```

4 Helm Chart

Para automatizar o deploy da aplicação no Kubernetes, foi criado o Helm chart `todoapi-chart`. Esse chart organiza os recursos Kubernetes em arquivos reutilizáveis e parametrizáveis, facilitando a manutenção e implantação em diferentes ambientes.

4.1 Estrutura do Chart

O chart é composto por três elementos principais:

- **Chart.yaml** – arquivo de metadados, que define nome, versão, descrição e dependências do chart.
- **values.yaml** – centraliza os valores configuráveis, como nomes de serviços, imagens Docker, portas e variáveis de ambiente.
- **templates/** – contém os templates dos manifestos Kubernetes (Deployment, Service, Ingress), que utilizam sintaxe Helm baseada em Go templates.

4.2 Configuração via `values.yaml`

O arquivo `values.yaml` foi estruturado para permitir o controle separado dos seguintes componentes:

- **Backend:** define o nome do serviço, a imagem utilizada, a porta do container e a string de conexão ao banco.
- **Frontend:** configura o nome e a imagem do container que serve a aplicação web.
- **PostgreSQL:** especifica imagem e porta do banco de dados.
- **Ingress:** habilita ou desabilita o uso de Ingress, além de configurar o hostname (`k8s.local`) e as anotações específicas do NGINX Controller.

Esses valores são injetados dinamicamente nos templates Kubernetes por meio da notação Helm `{{ .Values.nomeDoCampo }}`, promovendo flexibilidade e evitando duplicação de código.

O arquivo completo encontra-se no repositório Git do projeto.

5 Artefatos Kubernetes Gerados pelo Helm

A seguir, são descritos alguns dos principais artefatos Kubernetes gerados a partir dos templates Helm do chart `todoapi-chart`. Todos esses recursos são parametrizados e organizados dentro do diretório `templates/`, permitindo reaproveitamento e flexibilidade via `values.yaml`.

5.1 Deployment (Backend)

O template do `Deployment` cria uma réplica do container de backend, atribuindo rótulos e definindo a imagem e porta de escuta. Ele também injeta a variável de ambiente com a string de conexão ao banco de dados.

Como exemplo, o template utiliza os campos `{{ .Values.backend.image }}` para definir a imagem do container e `{{ .Values.backend.containerPort }}` para indicar a porta exposta.

Isso garante que os valores definidos no `values.yaml` sejam injetados diretamente no manifesto final renderizado.

5.2 Service (Frontend)

O template do `Service` define um serviço do tipo `ClusterIP`, responsável por expor o frontend internamente no cluster. Ele utiliza `selectors` para associar-se ao `Deployment` correspondente e expõe a porta 80.

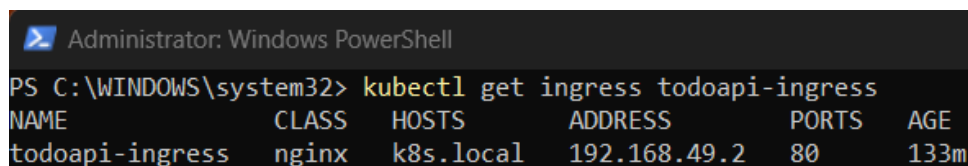
Esse serviço é fundamental para que o Ingress possa redirecionar requisições HTTP externas para o frontend de forma controlada.

5.3 Ingress

O recurso **Ingress** utiliza o NGINX Controller para mapear o hostname `k8s.local` aos serviços internos. Ele contém anotações para reescrita de caminho e define duas rotas: uma para o frontend e outra para a API backend, com base nos valores do `values.yaml`.

A presença do campo `enabled: true` no bloco `ingress` permite condicionalmente ativar esse recurso, tornando o chart mais flexível para ambientes diferentes.

A Figura 2 abaixo ilustra o Ingress em execução, com as rotas corretamente configuradas e expostas via domínio local.



```

Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl get ingress todoapi-ingress
NAME          CLASS  HOSTS      ADDRESS      PORTS      AGE
todoapi-ingress  nginx  k8s.local  192.168.49.2  80         133m
  
```

Figura 2: Ingress configurado e em execução

6 Automação do Deploy

Os scripts criados automatizam o processo de deploy, compilando imagens, aplicando o Helm chart e aguardando readiness dos pods:

```

1 eval $(minikube -p minikube docker-env)
2
3 docker build -t backend:latest -f Dockerfile .
4
5 docker build -t frontend:latest -f Dockerfile .
6
7 helm upgrade --install todoapi ./todoapi-chart
8
9 kubectl wait --for=condition=Ready pod -l app=backend --timeout=90s
10 kubectl wait --for=condition=Ready pod -l app=frontend --timeout=90s
11 kubectl wait --for=condition=Ready pod -l app=postgres --timeout=90s
  
```

Listing 1: Script de automação do deploy

7 Acesso e Validação

Linux / macOS

1. Executar script:

```
1 ./deploy.sh
```

2. Iniciar túnel:

```
1 minikube tunnel
```

3. Editar /etc/hosts:

```
1 127.0.0.1 k8s.local
```

4. Acessar: <http://k8s.local/>

Windows

1. Executar script:

```
1 .\deploy.ps1
```

2. Iniciar túnel:

```
1 minikube tunnel
```

3. Editar

C:\Windows\System32\drivers\etc\hosts:

```
1 127.0.0.1 k8s.local
```

4. Acessar: <http://k8s.local/>

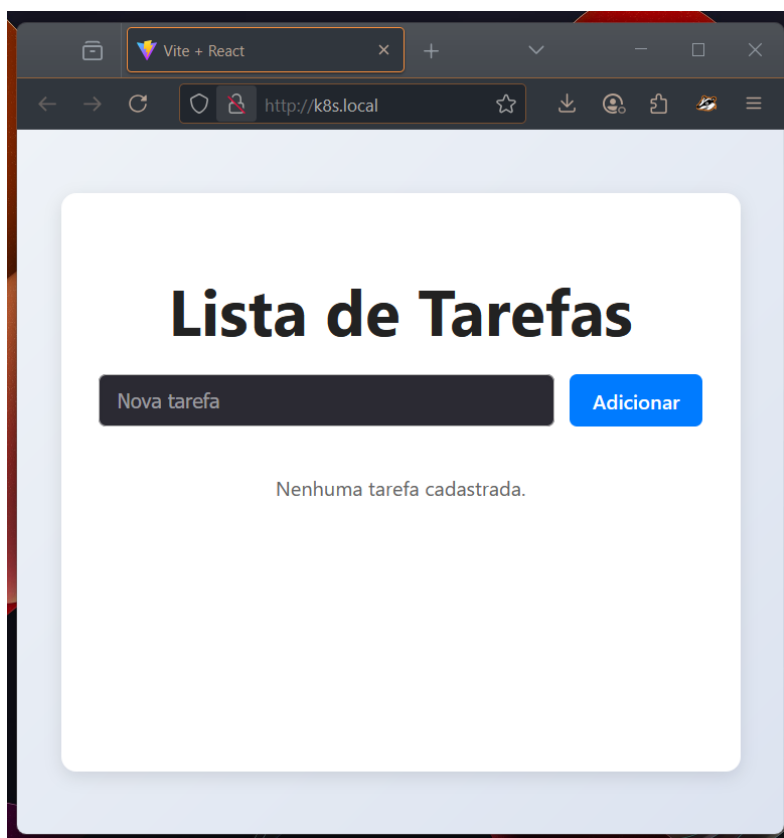


Figura 3: Frontend acessível via `k8s.local`

8 Conclusão

Acredito que este projeto integrou conceitos essenciais de DevOps: containerização com Docker, orquestração via Kubernetes, parametrização com Helm e automação através de scripts. A experiência proporcionou um ambiente de deploy reproduzível, com exposição de serviços via Ingress e domínio customizado local, aproximando-se de práticas comuns em ambientes de produção.