Sistema de Recomendação de Playlists com Kubernetes e CI/CD

Raissa Miranda Maciel
2020006965

Computação em Nuvem
Professor: Ítalo Fernando Scota Cunha

Janeiro de 2025

Sumário

1 Introdução

Este projeto tem como objetivo implementar um sistema de recomendação de playlists baseado em aprendizado de máquina utilizando tecnologias modernas de virtualização e orquestração de containers. A solução inclui a criação de um pipeline de integração e entrega contínuas (CI/CD) para automatizar o processo de treinamento do modelo, atualização de código e implantação no cluster Kubernetes.

As principais tecnologias utilizadas incluem:

- **Docker**: Para criar containers que encapsulam o código e suas dependências.
- **Kubernetes**: Para orquestrar os containers, gerenciar serviços e réplicas.
- **ArgoCD**: Para implementar a automação do CI/CD, monitorando mudanças no repositório Git.
- **Persistent Volumes**: Para compartilhar dados entre os containers, como o modelo de recomendação gerado.

O sistema funciona em duas etapas principais. O primeiro container (ML container) é responsável por processar o dataset e gerar o modelo de recomendação, que é salvo em um volume persistente. O segundo container (Frontend container) utiliza este modelo para responder a requisições de recomendação por meio de uma API REST. As atualizações no dataset, código ou configuração de Kubernetes são automaticamente aplicadas utilizando o ArgoCD.

1.1 Arquitetura do Sistema

O sistema é composto por três principais componentes:

- **ML container**: Container responsável por processar o dataset e gerar as regras de recomendação utilizando o algoritmo FP-Growth. As regras são salvas em um arquivo no volume persistente.
- **Frontend container**: Container que expõe uma API REST para fornecer recomendações de playlists. Ele utiliza as regras geradas pelo ML container armazenadas no volume persistente.
- **Volume Persistente (PVC)**: Um espaço compartilhado entre os containers que armazena o modelo de recomendação. Garante a integridade e a disponibilidade dos dados.

Os containers interagem por meio do Kubernetes, que gerencia a criação e execução dos pods correspondentes. O ArgoCD monitora um repositório Git configurado para detectar mudanças nos arquivos YAML de configuração ou no código-fonte. Assim, sempre que há uma atualização, o ArgoCD realiza o redeploy automático, garantindo que o sistema esteja sempre atualizado.

1.2 Metodologia

Os passos realizados neste projeto incluem:

1. **Criação das imagens Docker**: Foram criados dois Dockerfiles, um para o ML container e outro para o Frontend container. As imagens foram construídas localmente e enviadas para o repositório no Quay.io.
2. **Configuração dos arquivos YAML do Kubernetes**: Arquivos YAML foram criados para especificar os deployments, serviços, volumes persistentes e jobs necessários para o funcionamento do sistema.

3. **Configuração e integração do ArgoCD**: O ArgoCD foi configurado para monitorar o repositório Git do projeto e aplicar as mudanças automaticamente no cluster Kubernetes.

4. **Configuração do volume persistente**: Um Persistent Volume Claim (PVC) foi configurado para armazenar o modelo de recomendação gerado pelo ML container e compartilhá-lo com o Frontend container.

5. **Detecção de mudanças e atualização**: As mudanças no repositório, como novos datasets, versões de código ou configurações, são detectadas pelo ArgoCD. O sistema é automaticamente atualizado com base nessas alterações, incluindo a regeneração do modelo de recomendação ou o redeploy dos serviços.

2 Casos de Teste e Resultados

2.1 Atualização da versão do código

Para avaliar a habilidade do sistema em realizar atualizações contínuas de código, os seguintes passos foram realizados:

1. Modificação no código do *ML container* para refletir uma nova versão. Por exemplo, uma mensagem de depuração no arquivo `rule_generator.py` foi alterada.

2. Recriação da imagem Docker do *ML container* com uma nova tag para refletir a versão atualizada do código:

```
docker build -t quay.io/raissamaciel/ml-container:4.0 .
docker push quay.io/raissamaciel/ml-container:4.0
```

3. Atualização do arquivo de deployment do *ml-deployment.yaml* para usar a nova imagem Docker:

```
image: quay.io/raissamaciel/ml-container:4.0
```

4. Commit das alterações no repositório Git e *push*, para que o ArgoCD detectar a mudança.

Resultados: O ArgoCD identificou automaticamente a nova tag da imagem no repositório e realizou a sincronização do estado desejado com o cluster Kubernetes. O processo foi concluído em aproximadamente **10 segundos**. Durante o período de atualização, o serviço continuou acessível.

2.2 Atualização do número de réplicas

Para testar a capacidade do sistema em lidar com atualizações de código, foram realizados os seguintes passos:

1. Atualização do número de réplicas no código do *Frontend container*.

2. Incremento da tag da imagem Docker para refletir a nova versão do código, usando o comando:

```
docker build -t quay.io/raissamaciel/frontend-container:3.0 .
docker push quay.io/raissamaciel/frontend-container:3.0
```

3. Atualização do arquivo de deployment do *frontend-deployment.yaml* para usar a nova imagem Docker:

```
image: quay.io/raissamaciel/frontend-container:3.0
```

4. Commit das alterações no repositório Git e *push*. O ArgoCD detectou automaticamente a mudança no repositório e iniciou o processo de atualização no cluster Kubernetes.

Resultados: Após o *commit* e o *push*, o ArgoCD identificou a nova versão do código e sincronizou o estado desejado com o cluster Kubernetes. O processo de atualização foi concluído em aproximadamente **30 segundos**.

Durante o teste, o serviço permaneceu acessível durante todo o tempo ao realizar novas requisições.

2.3 Atualização do dataset

Para avaliar o impacto de mudanças no dataset e verificar se o sistema realiza a regeneração do modelo de recomendação de forma contínua, os seguintes passos foram realizados:

1. Substituição do dataset utilizado pelo *ML container*, alterando o valor da variável de ambiente `DATASET` no arquivo `ml-deployment.yaml`, de `ds1` para `ds2`.
2. Commit das alterações no repositório Git e realização do *push* para que o ArgoCD detectasse as mudanças e iniciasse a sincronização.
3. Observação do ArgoCD para monitorar a sincronização e regeneração do modelo de recomendação pelo *ML container*. A nova versão do modelo foi salva no volume persistente compartilhado.
4. Teste do *REST API* do *Frontend container* para verificar se o modelo atualizado estava sendo utilizado. Isso foi feito enviando uma requisição via *wget* com uma lista de músicas de entrada:

```
wget --server-response \
  --output-document response.out \
  --header='Content-Type: application/json' \
  --post-data '{"songs": ["Shape of You", "DNA.", "Bad and Boujee"]}' \
  http://<IP-do-service>:52056/api/recommend
```

5. Após a requisição, um arquivo *response.out* foi corretamente gerado com as recomendações, assim como anteriormente.

Resultados: Após a alteração do dataset, o ArgoCD realizou a sincronização do estado atualizado com o cluster Kubernetes. O *ML container* regenerou o modelo de recomendação em aproximadamente **20 segundos**, e o novo modelo foi detectado e carregado pelo *Frontend container* automaticamente. Durante o processo de atualização, o serviço permaneceu acessível.

3 Apêndice

3.1 Links Relevantes

- Repositório GitHub: `https://github.com/raissamiranda/tp2-cloud`
- Imagens Docker no Quay.io:
  - ML Container: `https://quay.io/repository/raissamaciel/ml-container`
  - Frontend Container: `https://quay.io/repository/raissamaciel/frontend-container`