Computação em Nuvem - Trabalho Prático 2

Victor Henrique Silva Ribeiro: 2020022723
Arthur Linhares Madureira: 2021031599
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
victor.henrique5800@gmail.com
arthurlm1234@gmail.com

1 Introdução

Neste trabalho, aplicamos os conceitos de DevOps e Computação em Nuvem para desenvolver, testar e implantar uma aplicação web de forma automatizada e escalável. A aplicação consiste em um sistema de recomendação de playlists que faz suas recomendações com base no padrão de músicas escutadas pelo usuário. A aplicação foi desenvolvida usando a linguagem Python, o framework Flask para tratar as requisições REST e o gerador de regras Apriori para construir as recomendações de playlists. Para realizar o CI/CD (Continuous Integration/Continuous Delivery), usamos as ferramentas GitHub Actions, ArgoCD, Docker e Kubernetes.

2 Organização das Pastas do Projeto

A estrutura de pastas do projeto é organizada da seguinte maneira:

- **.github/workflows**: Contém os arquivos de configuração do GitHub Actions, como `ci.yaml`, que define o processo de integração contínua.
- **api**: Esta pasta inclui os arquivos relacionados à API, como:
  - `Dockerfile`: Para criar a imagem Docker da API.
  - `api.py`: O script principal da API.
  - `client.py`: Um cliente para testar a API.
- **kubernetes**: Contém arquivos de configuração do Kubernetes, como `deployment.yaml`, `pvc.yaml` e `service.yaml`, para gerenciar a implantação e os serviços.
- **ml**: Esta pasta abriga os componentes de Machine Learning, incluindo:
  - `Dockerfile`: Para a imagem Docker do componente de ML.

  - `datasets`: Contém conjuntos de dados, como `2023_spotify_ds1.csv`, `2023_spotify_ds2.csv`, e `2023_spotify_songs.csv`.
  - `ml_trainer.py`: Script para treinamento de modelos de ML.

- **models**: Inclui modelos de Machine Learning salvos, como `rules.pkl` e `songs_artists.pkl`.
- **Arquivos Diversos**: Como `response.out`, `start_pod.sh` (script para iniciar um pod), e `test_api.sh` (script para testar a API).

Esta estrutura de pastas facilita a organização e o gerenciamento do projeto, permitindo uma separação clara entre os diferentes componentes, como a API, o Machine Learning e a configuração do Kubernetes.

3 Método

3.1 Machine Learning

Desenvolvemos um gerador de regras de associação para um sistema de recomendação. O modelo é salvo como um arquivo pickle em um volume persistente. O objetivo é aplicar um algoritmo de Mineração de Conjuntos Frequentes (FIM) para encontrar padrões frequentes e suas regras (se A, então B), permitindo a recomendação de playlists que os usuários possam gostar.

3.2 API REST

Descrevemos a implementação de uma API REST usando Flask, que serve como interface para o sistema de recomendação. A API lê as regras de recomendação do volume persistente para fornecer as sugestões de playlists.

3.3 GitHub Actions

O workflow do GitHub Actions é responsável por automatizar o processo de construção e publicação de imagens Docker para o Docker Hub, acionado sempre que há uma alteração na branch principal do repositório.

3.4 Kubernetes e ArgoCD

Utilizamos Kubernetes para gerenciar os containers Docker da aplicação e ArgoCD para atualizações automáticas dos pods do Kubernetes.

4 Função do Kubernetes e ArgoCD no Projeto

4.1 Kubernetes

No nosso projeto, o Kubernetes é utilizado para gerenciar a implantação e operação dos contêineres da aplicação. A configuração é especificada no arquivo

`deployment.yaml`, que define um Deployment para a aplicação. As principais características incluem:

- **Definição de Contêineres**: Dois contêineres são especificados, `cloud-api` e `cloud-ml-trainer`, cada um com sua própria imagem Docker.
- **Política de Pull de Imagem**: A política `imagePullPolicy: Always` garante que a versão mais recente da imagem seja sempre utilizada.
- **Portas e Volumes**: As portas necessárias são expostas e os volumes são montados para garantir a persistência dos dados.
- **Variáveis de Ambiente**: Variáveis como `DATASET_URL` são definidas para o contêiner de treinamento de Machine Learning.

Além disso, o arquivo `service.yaml` define um Service para expor a aplicação, especificando o protocolo e a porta para acesso externo.

4.2 ArgoCD

Embora o ArgoCD não seja explicitamente mencionado nos arquivos analisados, sua função no projeto pode ser inferida. ArgoCD seria responsável por:

- **Sincronização Automática**: Monitorar as mudanças no repositório e sincronizar automaticamente o estado do Kubernetes com o estado desejado definido nos arquivos de configuração.
- **Gerenciamento Declarativo**: Permitir a definição declarativa do estado desejado da aplicação, facilitando atualizações e manutenção.

5 Avaliação de Testes em Kubernetes e ArgoCD

5.1 Configuração do Ambiente de Testes

Configuramos um ambiente de teste que replica nossa infraestrutura de produção, utilizando Kubernetes e ArgoCD.

5.2 Testes de Integração Contínua

Avaliamos o tempo de implantação desde a submissão de uma mudança no código até a implantação efetiva no ambiente de Kubernetes e a disponibilidade da aplicação durante o processo.

5.3 Resultados e Discussão

Observamos que a construção da imagem está funcionando corretamente, o upload da imagem para o Docker Hub está eficiente, e a comunicação com o servidor dentro da VM está operacional. Além disso, confirmamos que o volume persistente está funcionando como esperado. As mudanças no código levam em média 15 segundos para serem completamente implantadas.

6 Conclusões

Os testes demonstraram que a combinação de Kubernetes e ArgoCD oferece um processo robusto e eficiente de integração contínua, com rápido tempo de implantação e alta disponibilidade da aplicação.