Project 2: DevOps and Cloud Computing

Nome: Gabriel Alves Reis
Matrícula: 2020006507

Introdução:

Este trabalho tem como objetivo exercitar conceitos de DevOps e computação em nuvem utilizando uma aplicação de ML que visa recomendar uma playlist de músicas baseada no gosto musical do usuário.

De início discutirei brevemente sobre os testes realizados para praticar o CI/CD bem como o tempo gasto para este passo. Depois discorro sobre como é detectado mudanças no modelo pela API REST e como o modelo detecta um novo dataset de treino. Por fim, explico como é realizada a comunicação do cliente com o servidor e é listado onde cada entregável se encontra no diretório enviado pelo moodle.

Kubernetes e ArgoCD test cases

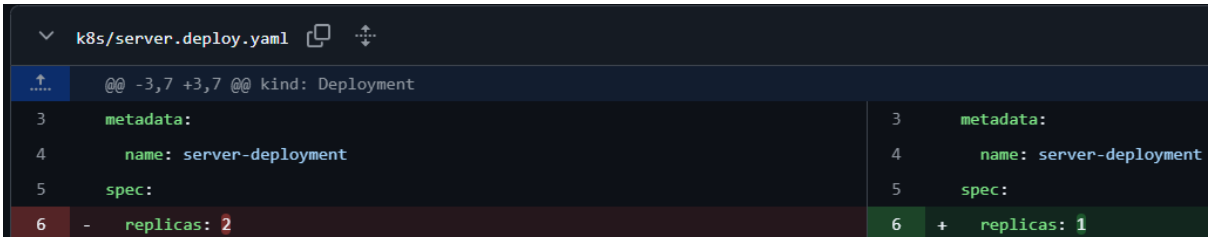
Entre os testes realizados temos: atualização do algoritmo que gera o modelo de sugestão de músicas, mudança de detalhes de configuração do deploy no Kubernetes (e.g. diminuir o número de réplicas) e update no dataset utilizado no treino. Todas essas alterações são realizadas por meio de mudanças nos arquivos de configuração do Kubernetes que estão disponíveis no diretório ./k8s do repositório.

É esperado que, ao serem disponibilizadas estas alterações no repositório do Github o ArgoCD se encarregue de verificar tais mudanças no diretório ./k8s e realize a integração e deploy do código no ambiente kubernetes.

1 - <u>Update na versão do código do modelo:</u>

Ao atualizar a versão do código para utilizar a nova imagem Docker quay.io/gaalves/playlists-recommender-model:0.6 demorou cerca de 2 minutos para a nova versão ficar disponível após commit no github. Vale ressaltar que se um cliente realizar requisições durante o processo de treinamento de uma nova versão o modelo anterior é utilizado como resposta e o serviço **não** fica fora do ar.

2 - Update no deploy da REST API (n° de réplicas):

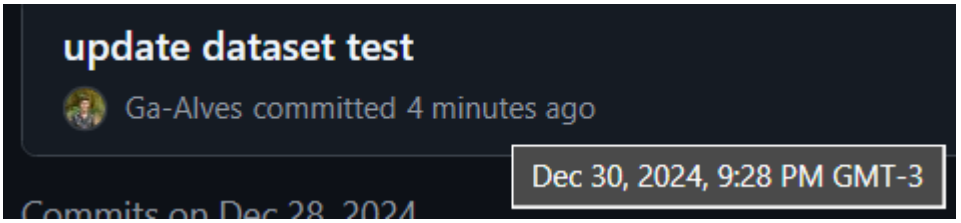


Ao atualizar o número de réplicas da REST API demorou em torno de 1 minuto para realizar a alteração de 2 réplicas para 1 desde o commit no github. A mudança foi tão rápida que não foi percebida durante testes empíricos de uso da API.

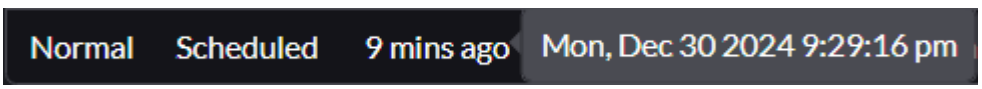
3 - Update no dataset utilizado pelo modelo:

Ao atualizar o dataset de treinamento do modelo de 2023_spotify_ds1.csv para 2023_spotify_ds2.csv demorou, aproximadamente, 1 minuto para iniciar o Job após commit no github, e mais 1 minuto para rodar o modelo em si, totalizando 2 minutos para o modelo estar disponível aos usuários desde o momento da alteração do repositório, como evidenciado nas imagens abaixo.

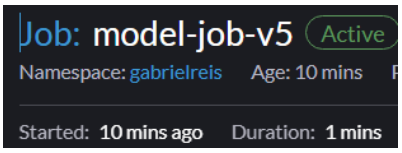
Commit no github:



Resposta inicial do ArgoCD (1 min depois):



Duração do treinamento (1 min):



Detecção de mudanças no modelo pela API REST

A forma que foi utilizada para detectar mudanças no modelo pela API REST é pela resposta da rota que informa a data de criação do último modelo (model_date).

Resposta da API REST:

```
return jsonify({
    "songs": recommendation,
    "version": VERSION,
    "model_date": createdAt
})
```

Resposta na CLI:

```
-----------------------------------------------------
| Informe músicas para receber uma playlist customizada |
-----------------------------------------------------
|   insira /send para finalizar o envio               |
-----------------------------------------------------
Unforgettable
/send
Aguarde enquanto processamos seu dados!

Sua playlist:
API version:  1.0.0
Latest Model Update Date:  2025-01-01T19:17:42.663201-03:00
Songs:
1 Unforgettable
2 Congratulations
3 HUMBLE.
4 XO TOUR Llif3
5 goosebumps
6 T-Shirt
7 Mask Off
```

Como o modelo detecta um novo dataset ?

Para detectar o novo dataset utilizado para gerar o modelo utiliza-se uma variável de ambiente passado no yaml do Job de geração do modelo.

```
env:
- name: DATASET
  value: "https://homepages.dcc.ufmg.br/~cunha/hosted/cloudcomp-2023s2-datasets/2023_spotify_ds2.csv"
```

Como utilizar o cliente CLI?

Este trabalho foi realizado utilizando a linguagem python tanto para o cliente quanto para o servidor, em que optei por utilizar um cliente no formato CLI. Para fazer a comunicação com a API utilizei um túnel ssh mapeando o ip e porta do service (e.g. 10.43.115.13:52023) para a mesma porta no localhost.

Primeiro descubra o ip do serviço com o seguinte comando:

```
gabrielreis@cloud2:~$ kubectl get services
NAME                                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)     AGE
gabrielreis-playlist-recommender-service   ClusterIP   10.43.115.13   <none>        52023/TCP   6d1h
```

Por fim, faça o túnel ssh com o ip e porta para seu local.

```
ssh -p 51979 gabrielreis@pugna.snes.dcc.ufmg.br -L
31443:localhost:31443 -L 30443:localhost:30443 -L
52023:10.43.115.13:52023
```

Agora basta executar a CLI localmente.

Lista dos Entregáveis:

1 -
O código do modelo de ML está no caminho ./model/app.py

2 -
O código da REST API está no caminho ./server/app.py

3 -
O código do Client APP está no caminho ./client/app.py.

4 -
O Dockerfile do modelo de ML está no caminho ./model/Dockerfile
O Dockerfile da REST AP está no caminho ./server/Dockerfile

Para buildar o modelo de machine learning e publicar a imagem no quay.io utilizei o seguinte código substituindo <v> pela versão do modelo:

```
cd model
docker build -t quay.io/gaalves/playlists-recommender-model:0.<v> .
docker push quay.io/gaalves/playlists-recommender-model:0.<v>
```

Para buildar a REST API e publicar a imagem no quay.io utilizei o seguinte código substituindo <v> pela versão do modelo:

```
cd server
build -t quay.io/gaalves/playlists-recommender-server:0.<v> .
push quay.io/gaalves/playlists-recommender-server:0.<v>
```

5 -
Todas as configurações yaml de Deployment, Service e Persistent Volume Claim do Kubernetes estão no diretório ./k8s

6 -
O manifesto da aplicação ArgoCD está no caminho ./argocd/manifest.yaml