

DevOps e Computação em Nuvem

Carlos Henrique Brito Malta Leão

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
chbmleao@ufmg.br

Computação em Nuvem

Professor
Italo Fernando Scota Cunha
italocunha@ufmg.br

Belo Horizonte
2025

Introdução

O desenvolvimento e a entrega de aplicações em alta velocidade tornaram-se requisitos essenciais para organizações que buscam manter-se competitivas em um mercado dinâmico. Nesse contexto, o DevOps emerge como uma abordagem que combina práticas e ferramentas para integrar e automatizar os processos de desenvolvimento e operações. Essa metodologia promove uma entrega contínua e eficiente, permitindo que as equipes evoluam e melhorem seus produtos com rapidez e qualidade superior às abordagens tradicionais.

A computação em nuvem desempenha um papel crucial nessa transformação, viabilizando a colaboração contínua entre equipes e eliminando os desafios de sincronização manual de arquivos. Além disso, tecnologias como controle de versão (Git), containers (Docker e Kubernetes) e padrões arquiteturais como microsserviços permitem o desenvolvimento simultâneo, a criação de ambientes de teste experimentais avançados e a prototipagem rápida de soluções. Essas ferramentas possibilitam atualizações frequentes e aceleram a entrega de software.

No âmbito do DevOps, os conceitos de Integração Contínua (CI) e Entrega Contínua (CD) formam a espinha dorsal das operações automatizadas. A integração contínua envolve a fusão regular de alterações de código em um repositório central, seguido pela execução de builds e testes automatizados. Por sua vez, a entrega contínua automatiza etapas manuais de implantação, permitindo a disponibilização rápida de novos recursos e melhorias.

Neste projeto, foi implementado um sistema de recomendação de playlists baseado em microsserviços. O sistema contará com uma interface web e um módulo de aprendizado de máquina (Machine Learning), integrados por meio das práticas de DevOps, também conhecidas como MLOps no contexto de workflows de aprendizado de máquina. Além disso, foram utilizadas algumas ferramentas amplamente reconhecidas, como Docker, Kubernetes, GitHub e ArgoCD, para construir, testar e entregar o sistema em um ambiente de nuvem.

O código e os dados utilizados no projeto foram armazenados no repositório <https://github.com/Chbmleao/playlists-recommender>. Nesse sentido, o repositório apresenta a seguinte estrutura de arquivos:

- **.github/workflows:**
 - **update-deployment.yml:** define um fluxo de trabalho no GitHub Actions para atualizar automaticamente o arquivo YAML do gerador de jobs do Kubernetes sempre que alterações forem feitas nos arquivos `generator-job.yaml` ou `service.yaml`. Além disso, o workflow é acionado também quando o workflow anterior, responsável pela atualização do arquivo

generator-job.yaml, é completado. O processo inclui a modificação do arquivo de deployment do Kubernetes (deployment.yaml) para incluir o hash do commit atual no campo `'metadata.name'`, seguida de um commit e push das alterações para o repositório. É necessário alterar esse campo para que o ArgoCD reinicie o deployment automaticamente, dessa forma, o deployment poderá acessar os dados mais atualizados. Essa atualização é feita somente após uma pausa de 5 minutos, permitindo o tempo necessário para garantir que outras ações possam ser concluídas, como a atualização do dataset ou do modelo.

- **update-generator-job.yml:** define um fluxo de trabalho no GitHub Actions que é acionado sempre que há alterações no arquivo `spotify.csv` localizado no diretório `data/`. O objetivo deste workflow é atualizar o arquivo YAML do gerador de jobs do Kubernetes (generator-job.yaml) com o hash curto do commit atual, garantindo que a versão do job seja atualizada conforme mudanças no dataset. Após a modificação do arquivo YAML, o workflow realiza um commit e push das alterações para o repositório, utilizando o GitHub Actions Bot para gerenciar as credenciais de usuário. Esse processo atualiza o nome do job para que seja reiniciado pelo ArgoCD, que ao ser reprocessado gera um novo modelo com base no dataset atualizado.

- **data**

- **spotify.csv:** uma amostra reduzida de um conjunto de dados do Spotify. Este arquivo apresenta um conjunto de playlists presentes na plataforma e podem ser usados para emular atualizações no modelo de recomendação.
- **playlist_rules.pkl:** um arquivo no formato PICKLE que apresenta uma versão do modelo de Aprendizado de Máquina gerado.

- **manifests:**

- **deployment.yaml:** Este arquivo define um Deployment para o serviço de recomendação de playlists, configurando o número de réplicas, o contêiner com a imagem do sistema de recomendação e montando um volume persistente para armazenar dados compartilhados.
- **generator-job.yaml:** Este arquivo descreve um Job para gerar regras de playlists, configurando o contêiner com a imagem do gerador de regras e a execução de um script de entrada, além de montar o volume persistente para compartilhar dados.

- **pvc.yaml:** Este arquivo configura uma PersistentVolumeClaim para fornecer armazenamento persistente de 1 GB com acesso ReadWriteMany, utilizado por outros recursos como Deployment e Job.
- **service.yaml:** Este arquivo define um Service para o serviço de recomendação de playlists, mapeando a porta 18485 para a porta 5000 do contêiner, permitindo o acesso externo ao serviço.
- **model-generator:** contém os arquivos necessários para a geração e atualização de um modelo de recomendação de playlists. Ela inclui um Dockerfile para criar a imagem do contêiner, que instala as dependências e executa o arquivo entrypoint.sh, um script de inicialização que atualiza o dataset ao clonar um repositório Git, copia o dataset mais recente para o diretório apropriado e executa o script Python (app.py) para atualizar o modelo. O script é responsável por importar um dataset de playlists, aplicar o algoritmo de frequent pattern mining (FP-Growth) para gerar regras de associação, e salvar essas regras em um arquivo pickle.
- **server:** contém os arquivos necessários para a construção e execução de um servidor Flask, responsável por fornecer um serviço de recomendação de músicas. O Dockerfile configura a imagem Docker para o servidor, instalando as dependências listadas em requirements.txt e copiando o script app.py, que contém a lógica principal do servidor. O app.py carrega um modelo de recomendação, armazenado como um arquivo pickle, e expõe uma API RESTful com um único endpoint /api/recommend que recebe uma lista de músicas e retorna outras músicas recomendadas com base nas regras do modelo. O servidor também retorna a versão da imagem e a data de modificação do modelo.
- **argocd-app.yaml:** Arquivo YAML que descreve a aplicação do ArgoCD.
- **client.sh:** script da aplicação do cliente, que demonstra o acesso à API RESTful.

A estrutura da aplicação dentro do ArgoCD está representada na Figura 1.

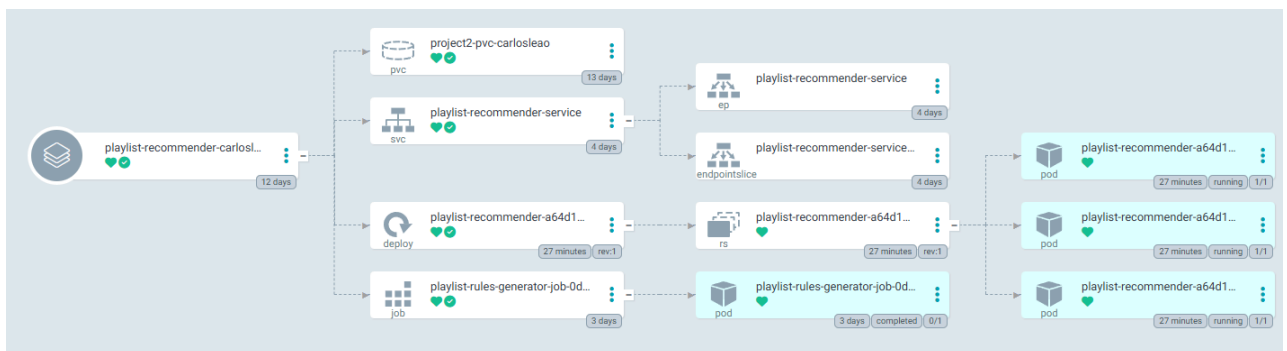
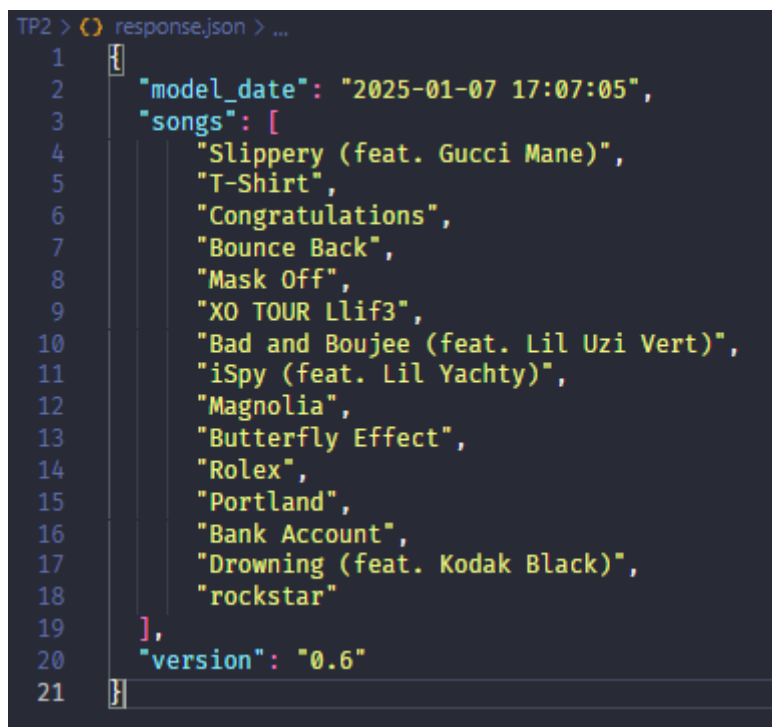


Figura 1: Estrutura da aplicação no ArgoCD

Casos de Teste

Nessa seção, temos alguns casos de teste que foram realizados durante e após o período de desenvolvimento. Todos os casos de teste, dependem de alguma alteração no repositório GitHub do projeto, já que o ArgoCD realiza o mapeamento do repositório supracitado. Além disso, ao realizar alguma alteração em algum arquivo *.yaml, é necessário alterar também o campo *metadata.name*, para evitar problemas de sincronização com o ArgoCD.

- **Simulação de uma solicitação do cliente:** O arquivo client.sh apresenta um script que utiliza um comando wget para realizar uma solicitação para o servidor. Dessa forma, ao executar o script disponível no repositório do Github dentro da máquina virtual do projeto, foi criado um arquivo 'response.json', com uma lista de recomendação de músicas, a última data de atualização do modelo e a versão do código do deployment. O arquivo response.json pode ser visualizado na Figura 2.



```
TP2 > response.json > ...
1  {
2    "model_date": "2025-01-07 17:07:05",
3    "songs": [
4      "Slippery (feat. Gucci Mane)",
5      "T-Shirt",
6      "Congratulations",
7      "Bounce Back",
8      "Mask Off",
9      "XO TOUR Llif3",
10     "Bad and Boujee (feat. Lil Uzi Vert)",
11     "iSpy (feat. Lil Yachty)",
12     "Magnolia",
13     "Butterfly Effect",
14     "Rolex",
15     "Portland",
16     "Bank Account",
17     "Drowning (feat. Kodak Black)",
18     "rockstar"
19   ],
20   "version": "0.6"
21 }
```

Figura 2: Arquivo response.json

- **Atualização da imagem do deployment:** Ao atualizar o código do servidor de recomendações, é necessário atualizar o número da versão da imagem no arquivo manifests/deployment.yaml. Dessa forma, o ArgoCD detecta a alteração automaticamente e reinicia o deployment assim como o seus pods, dependendo do número de réplicas. O novo deployment executado, utiliza o arquivo playlist_rules.pkl armazenado no PersistentVolumeClaim (PVC) da aplicação.

- **Atualização da imagem do gerador do modelo:** Ao atualizar o código do gerador do modelo, também é necessário atualizar o número da versão da imagem no arquivo manifests/generator-job.yaml. Dessa forma, o ArgoCD detecta a alteração automaticamente e executa o job, que clona o repositório do github e acessa o dataset mais atualizado. Com o novo dataset em memória, o job cria as novas regras de recomendação, e armazena no arquivo playlist_rules.pkl dentro do PersistentVolumeClaim (PVC) da aplicação.
- **Atualização do número de réplicas:** Ao atualizar o número de réplicas do arquivo manifests/deployment.yaml, automaticamente o novo deployment já é criado e, somente após a sua execução, o deployment antigo é terminado.
- **Atualização do dataset:** Esse caso de teste é o mais complexo, que demanda também a utilização de alguns workflows do GithubActions para seu funcionamento completo.
 - **Update Kubernetes Generator Job YAML on CSV Change:** Esse workflow é executado por qualquer alteração no arquivo data/spotify.csv. Ao ser executado, o arquivo manifests/generator-job.yaml tem o campo metadata.name modificado, e em seguida, a alteração é empurrada para o repositório do Github. Com essa alteração, o ArgoCD detecta a alteração do Job, e em seguida, o executa novamente para sua sincronização, o que gera um novo modelo que é salvo no PVC.
 - **Update Kubernetes Deployment YAML on CSV Change:** Esse workflow é executado após três possíveis cenários: atualização do arquivo manifests/generator-job.yaml; atualização do arquivo manifests/service.yaml; ou execução do workflow descrito acima. Em sua execução, primeiramente aguarda 5 minutos, para que seja garantido a geração do novo modelo, e, em seguida, é feito um processo muito semelhante ao workflow acima, mas dessa vez, é atualizado o campo metadata.name do arquivo deployment.yaml. Dessa forma, o ArgoCD realiza a reinicialização automática do deployment, acessando o modelo que acabou de ser atualizado.

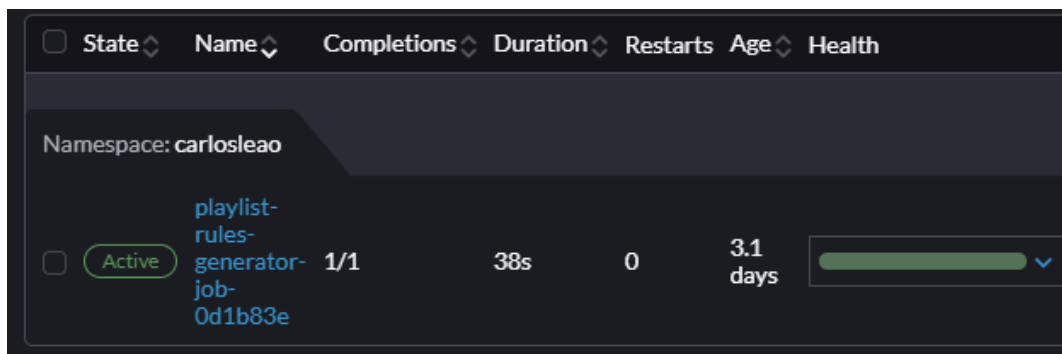
Impacto e Tempo de Implantação de Alterações no Sistema

Nesta seção, analisamos o impacto e o tempo necessário para que alterações no sistema sejam implantadas, bem como avaliamos se o aplicativo sofre períodos de indisponibilidade durante essas atualizações. Nesse sentido, a aplicação implantada nunca fica offline, como veremos nas análises seguintes.

Atualização do código de geração do modelo

Ao atualizar a versão do código do job, o ArgoCD remove o job anterior e implanta a nova versão, utilizando a imagem hospedada no DockerHub e o dataset atualizado hospedado no GitHub. O tempo de execução do script depende diretamente do número de músicas presentes nas playlists do dataset. No caso do dataset fornecido, que contém aproximadamente 240 mil músicas, o script leva cerca de 40 segundos para ser executado completamente e gerar o novo modelo de recomendação.

Durante a atualização do modelo, é importante destacar que o serviço permanece disponível e continua operando normalmente com o modelo antigo. Utilizando o GitHub Actions, cinco minutos após a atualização do arquivo de geração do modelo, o arquivo de deployment é atualizado. Como resultado, o ArgoCD cria um novo deployment que acessa o modelo recém-gerado. Somente após a verificação do funcionamento do novo deployment, o antigo é encerrado. Esse processo garante que a aplicação continue disponível para os usuários, sem interrupções durante a atualização.



The screenshot shows the Rancher UI interface for a job. At the top, there are columns for State, Name, Completions, Duration, Restarts, Age, and Health. Below this, the Namespace is set to 'carlosleao'. A single job is listed with the name 'playlist-rules-generator-job-0d1b83e'. The job is in the 'Active' state, with 1/1 completions, a duration of 38s, 0 restarts, and an age of 3.1 days. A green progress bar is visible next to the job name.

State	Name	Completions	Duration	Restarts	Age	Health
Active	playlist-rules-generator-job-0d1b83e	1/1	38s	0	3.1 days	Good

Figura 3: Captura de tela do Rancher do Job de geração do modelo

Atualização do deployment

Qualquer atualização do arquivo deployment.yaml resulta em um processo muito semelhante ao processo de atualização do deployment descrito acima. Primeiramente, o ArgoCD cria um novo deployment que acessa o modelo recém-gerado. Somente após a verificação do funcionamento do novo deployment, o antigo é encerrado. Esse processo garante que a aplicação continue disponível para os usuários, sem interrupções durante a atualização. É importante lembrar, que qualquer atualização do arquivo deployment.yaml necessita também atualizar o campo metadata.name, para não gerar erros de sincronização no ArgoCD.

Um exemplo desse tipo de atualização pode ser visto na Figura 4. Nesse exemplo, foi atualizado o número de réplicas do deployment de 4, para 3. Dessa forma, os 3 novos pods já

foram criados e já estão disponíveis para o acesso dos usuários. Somente após essa disponibilização que o ArgoCD começa a finalizar os 4 pods antigos.

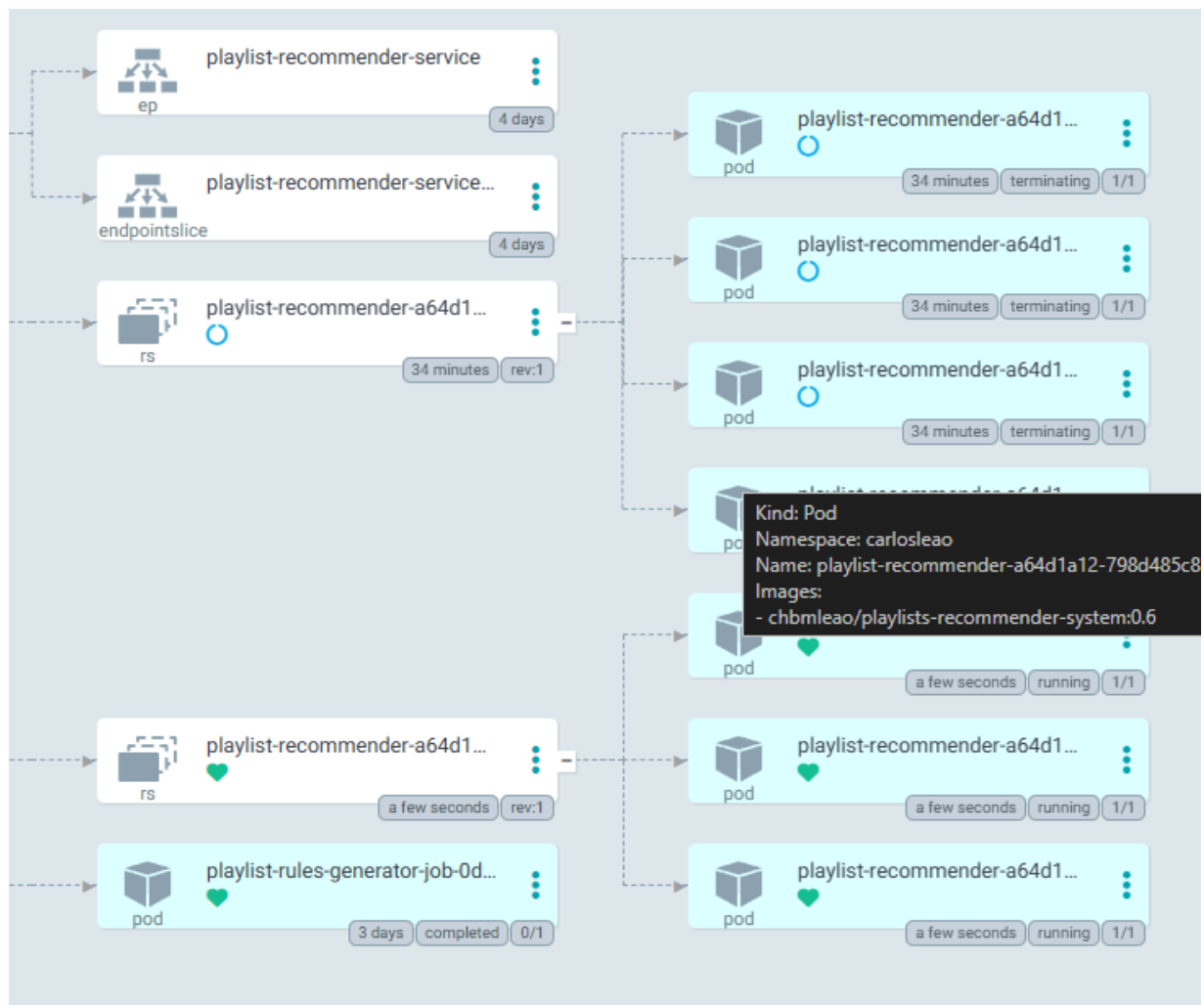


Figura 4: Captura de tela do ArgoCD ao atualizar o número de réplicas do deployment

Atualização do dataset

Ao atualizar o dataset armazenado no repositório do GitHub, um workflow do GitHub Actions é acionado, atualizando o nome do Job que é executado automaticamente pelo ArgoCD para gerar um novo modelo. Como mencionado anteriormente, a execução desse Job leva cerca de 40 segundos.

Logo após a conclusão do primeiro workflow, um segundo workflow é acionado automaticamente. Este segundo workflow inclui uma espera de 5 minutos para garantir que o modelo seja atualizado corretamente. Em seguida, o arquivo de deployment é atualizado, e o ArgoCD executa o mesmo processo descrito anteriormente. Assim, a atualização completa do deployment para utilizar o novo modelo gerado a partir do dataset leva aproximadamente 5 minutos. No entanto, o serviço permanece disponível durante todo o processo, permitindo

que os usuários continuem utilizando a aplicação normalmente com o modelo anterior até a transição ser concluída.

Deteção de atualizações do modelo

O modelo é atualizado em dois cenários principais, ambos monitorados pelo workflow ‘Update Kubernetes Deployment YAML on CSV Change’ do GitHub Actions. Esses cenários são:

- **Atualização do funcionamento ou da imagem do gerador do modelo:** O workflow verifica alterações no arquivo `generator-job.yaml`, permitindo identificar tanto modificações na estrutura do Job quanto atualizações no código, detectadas pela versão da imagem Docker associada.
- **Atualização do dataset:** Qualquer atualização no dataset aciona o workflow ‘Update Kubernetes Generator Job YAML on CSV Change’. Esse processo gera um novo modelo e, posteriormente, ativa o workflow de atualização do deployment tratado nesta seção.

Em ambos os casos, o workflow de atualização do deployment é executado com um atraso de 5 minutos para garantir que o novo modelo seja gerado e disponível. Isso assegura que o serviço esteja sempre utilizando o modelo mais recente, mantendo sua continuidade operacional sem interrupções.

Obtenção do Dataset para Regeneração do Modelo

A obtenção do novo dataset e a regeneração do modelo são realizadas por meio de um Job do Kubernetes, executado dentro do contêiner do gerador de modelos. O arquivo `entrypoint.sh` é responsável por automatizar esse processo, que segue as seguintes etapas:

1. **Clonagem do Repositório:** O repositório GitHub contendo o dataset atualizado é clonado para o contêiner no diretório temporário `./tmp/repo`.
2. **Atualização do Dataset:** O arquivo de dataset, armazenado no repositório clonado (`spotify.csv`), é copiado para o diretório `/app/data` do contêiner, substituindo o dataset antigo.
3. **Geração do Modelo:** O script `app.py` é executado, processando o novo dataset e gerando um novo modelo de recomendação.

Ambos, o dataset atualizado e o modelo gerado, são armazenados no Persistent Volume Claim (PVC), garantindo que estejam disponíveis de forma compartilhada entre os contêineres. Esse fluxo assegura que o processo de regeneração do modelo seja totalmente automatizado e integrado com as atualizações realizadas no repositório do GitHub.