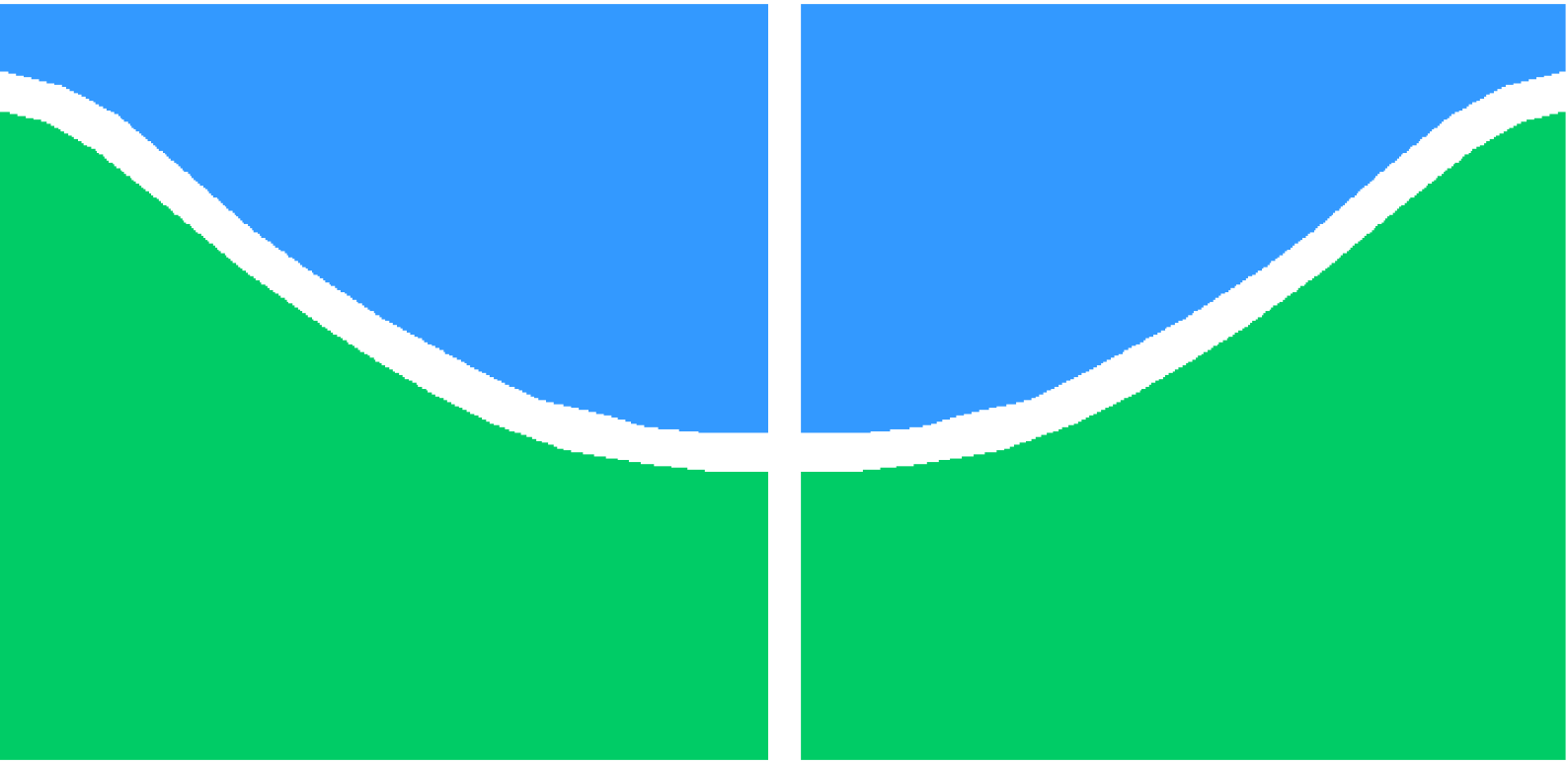

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Ferramentas de entrega contínua de sistemas de Machine Learning em comunidades Open Source: como caracterizar projetos de MLOps

Autor: Fábio Teixeira

Orientador: Dra. Carla Rocha

Brasília, DF

2022

Fábio Teixeira

Ferramentas de entrega contínua de sistemas de Machine Learning em comunidades Open Source: como caracterizar projetos de MLOps

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dra. Carla Rocha

Brasília, DF

2022

Fábio Teixeira

Ferramentas de entrega contínua de sistemas de Machine Learning em comunidades Open Source: como caracterizar projetos de MLOps/ Fábio Teixeira. – Brasília, DF, 2022-

45 p. : il. (algumas color.) ; 30 cm.

Orientador: Dra. Carla Rocha

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. MLOps, Software Livre, Open Source, Machine Learning, DevOps, entrega contínua. 2. Palavra-chave02. I. Dra. Carla Rocha. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Ferramentas de entrega contínua de sistemas de Machine Learning em comunidades Open Source: como caracterizar projetos de MLOps

CDU 02:141:005.6

Fábio Teixeira

Ferramentas de entrega contínua de sistemas de Machine Learning em comunidades Open Source: como caracterizar projetos de MLOps

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 01 de fevereiro 2022:

Dra. Carla Rocha
Orientador

Dr. Renato Coral
Convidado 1

Arthur Temporim
Convidado 2

Brasília, DF
2022

Agradecimentos

Agradeço a Deus, pela oportunidade de poder estudar na Universidade de Brasília. A minha mãe por todo apoio e incentivo. Aos professores do Laboratório Avançado de Produção, Pesquisa & Inovação em Software (LAPPIS), que me permitiram ser membro de excelentes projetos com profissionais do mercado com os quais pude aprender muito. Finalmente agradeço a todos que me auxiliaram de alguma forma em qualquer momento.

Resumo

Na engenharia de software a entrega contínua é um fator central para o sucesso de um modelo de negócios, mas na atualidade vem se tendo uma crescente adoção de machine learning por parte do mercado, demanda essa que traz desafios em como aplicar a entrega contínua juntamente com machine learning. Uma proposta do mercado para a solução desse problema é a prática do que é chamado de MLOps, que nada mais é que a utilização de técnicas e DevOps porém devidamente ajustadas para ambientes que dependem de machine learning, tendo em vista a constante necessidade de re-treinar modelos e já colocar seus resultados em produção. Porém, MLOps ainda é uma área nova do conhecimento com pouca documentação na literatura acadêmica o que proporciona um terreno fertil para o nascimento das mais diversas ferramentas para atender essa demanda do mercado. Sendo assim, este estudo tem por objetivo caracterizar a comunidade open source de MLOps, identificando as principais características e métricas que podem guiar práticos da engenharia de software na adoção de ferramentas de automação de MLOps. Seguindo uma metodologia de pesquisa exploratória, onde por uma mineração e análise dos principais projetos de MLOps possa-se eleger uma forma de diagnosticar a maturidade de uma ferramenta OSS a fim de auxiliar profissionais de MLOps na escolhas das ferramentas de seus projetos. Tendo como resultados esperados um conjuntos de boas práticas e guidelines além de uma ferramenta que auxilia práticos na adoção de ferramentas de automação de MLOps.

Palavras-chaves: Integração contínua, machine learning, DevOps, MLOps, open source, maturidade.

Lista de ilustrações

Lista de tabelas

Sumário

Introdução

O uso de componentes de machine learning em sistemas de software é uma tendência na indústria de software. Porém, implementar machine learning em produtos de software e adotar práticas referente à manutenção e evolução desses modelos em ambiente de produção não é um trabalho trivial. Normalmente, uma arquitetura de entrega contínua é construída para servir as aplicações que possui somente código fonte, e o problema de como inserir componentes de machine learning sistemática em uma pipeline de entrega contínua é um desafio. Em 2015, a Google apresentou o primeiro artigo levantando a complexidade e desefios de desenvolver, e principalmente manter, sistemas de software com módulos de machine learning (SCULLEY et al., 2015). Isso porque, além do código fonte, em um sistema com módulos de machine learning, o modelo treinado e os dados de treinamentos precisam ser gerenciados de forma idependente, de preferencialmente com pipelines de entrega independentes.

Para tal, foi cunhado um termo que representa práticas, cultura e automações relacionados a entrega contínua de produtos de machine learning, o MLOps (VISENGERIYEVA et al., 2021). MLOps adapta práticas de DevOps e ciência de dados para o contexto de sistemas de machine learning. O aumento da demanda de serviços especifico para gestão de produtos de machine learning fez com que grandes empresas provedoras de infraestrutura como serviço coloquem MLOps no seu menu de serviços (CLOUD, 2021).

Objetivo

O objetivo da pesquisa é fazer uma estudo exploratório sobre os desafios relativo a adoção de ferramentas open software que automatizam atividade referente a MLOps. Atualmente, há uma grande quantidade de ferramentas de automação disponíveis, que possuem os mais diversos níveis de maturidade e adesão por parte de usuários e organizações. A escolha de uma ferramenta de MLOps faz parte do processo de adoção da prática de MLOps por uma organização e possui diversas complexidades, tanto técnicas quanto organizacionais, relacionadas à implementação, manutenção e uso de tais ferramentas.

Objetivos Específicos

Esse trabalho tem como objetivo analisar e identificar a maturidade de soluções de MLOps e as complexidades referente à sua adoção. Os objetivos específicos da pesquisa são:

- Identificar práticas de MLOps já aplicadas no mercado;

- Mapear o pipeline de MLOps;
- Analisar ferramentas *Open Sorce* no contexto específico de MLOps;
- Caracterizar essas ferramentas em termos da comunidade de *Open Sorce* e em qualidade técnica;
- Mapear desafios de implementação e adoção de uma ferramenta de MLOps em um projeto piloto.

Estrutura do documento

O documento está divido em duas grandes seções. No capítulo 1 encontra-se a fundamentação teórica que aborda a prática de entrega contínua em machine leanring, dando uma visão geral indo desde de CI/CD até uma guideline do modelo de MLOps. No capítulo 2 está a Proposta da pesquisa, onde são explorados os desafios técnicos da adoção de MLOps e a Metodologia utilizada, além dos resultados encontrados, onde são categorizados alguns critérios para a qualificação de de ferramentas *Open Source* e a escolha de uma ferramenta que foi estudada bem como os levantamentos dessa análise.

1 Entrega contínua de Produtos de Machine Learning

A entrega contínua é um dos pilares da engenharia de software moderna, presente nos princípios do manifesto ágil: “Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado” (BECK K., 2001). Uma forma de promover a entrega contínua é utilizando de pequenos pacotes ao invés de uma grande entrega que pode levar meses ou até mesmo anos de desenvolvimento.

Além da entrega do código fonte, é necessário colocá-lo em ambiente de produção e realizar a manutenção e evolução. Outros fatores são necessários para garantir a qualidade do produto de software: segurança, monitoramento, desempenho, usabilidade, disponibilidade, entre outros. Ou seja, uma vez que o software está em produção, mantê-lo funcionando e estável passa a ser uma prioridade nas organizações.

Este primeiro capítulo tem por objetivo contextualizar a entrega contínua em machine learning, abordando os principais conceitos.

1.1 Integração Contínua

Integração contínua comumente abreviada para CI (do inglês *Continuous Integration*) é uma prática de desenvolvimento de software em que os membros de uma equipe integram com frequência as suas modificações e/ou adições, de forma que cada membro realiza pelo menos uma integração por dia (FOWLER, 2006). Levando a várias integrações ao dia, no qual cada integração é verificada por um processo automático (em que se incluem os testes) a fim de detectar problemas de integração o quanto antes e poder corrigi-los antes que estes entrem em produção.

Durante o processo de implementação de novas funcionalidades, várias pessoas trabalham em uma mesma base de código, que por sua vez pode levar a ter bugs de inconsistência de código e baixa coesão. Nesta situação um ambiente de CI pode detectar tais problemas e impedir que isto escale em uma típica situação de *Integration Hell* (CUNNINGHAM, 2012).

Fowler comenta em (CUNNINGHAM, 2012) que, embora bugs ainda irão ocorrer, um ambiente de CI irá facilitar sua detecção e correção. Fazendo com que um dos melhores benefícios dessa prática seja a redução de riscos. Porém, este grau de benefício está diretamente relacionado à capacidade de testar, detectar erros e corrigi-los.

Quanto ao uso de uma CI, Martin Fowler destaca alguns pontos chave que devem

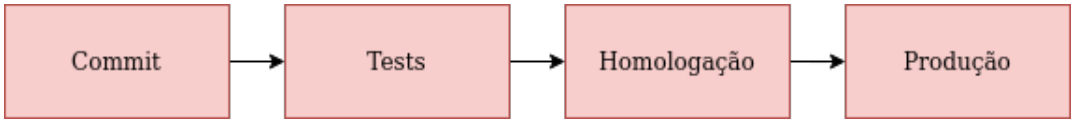

Figura 1 – Etapas principais de um pipeline.

ser seguidos:

- Manter um único repositório fonte;
- Realizar builds automáticas e testá-las também de forma automática;
- Fazer commits com mais frequência e manter uma comunicação com os membros do time;
- Todos os commits devem passar pelo pipeline de CI;
- Sempre corrigir builds quebradas;
- Testar em um ambiente próximo ao de produção.

1.2 Entrega Contínua

Entrega contínua comumente abreviada para CD (do inglês *Continuous Delivery*) é uma prática da engenharia de software em que as equipes lançam produtos de software com frequência. Uma *pipeline* é um processo com as verificações e ações necessárias antes de uma integração de código ou deploy do código, que vai deste realizar testes, testar a criação da build e colocar o software em produção. Opta-se por fazer pequenos lançamentos, reduzindo assim as possibilidades de erros e promovendo mais segurança.

1.2.1 Pipeline

Um típico pipeline de entrega contínua consiste em: fazer o commit para o repositório, realizar testes automatizados (unitários, aceitação, carga, etc), deploy em ambiente de homologação, deploy em produção(após aprovação no de homologação).

Em uma pipeline de CI/CD, cada mudança feita na aplicação, seja ela código, configuração, ambiente e etc, dispara uma nova instancia de execução desse ambiente de CI/CD.

É esperado em um ambiente de entrega contínua, que o processo de *deploy* seja executado de forma automatizada pelo pipeline, já que realizar esse processo manualmente é considerado um *antipattern*. "Dever haver somente duas tarefas a serem executadas por um ser humano: escolher uma versão, um ambiente e pressionar o botão de deploy" (HUMBLE, 2010).

1.3 DevOps

DevOps é uma junção dos termos *development* e *operations*, e consiste em práticas, ferramentas e cultura entre entre esses dois times a fim de garantir entrega contínua. Diferentemente do Manifesto Ágil, DevOps não possui um manifesto definido, o que acaba por gerar uma maior pluralidade em como ele é definido e aplicado (LEITE et al., 2019). Optamos pela seguinte definição de DevOps: “O DevOps é a combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura. Essa velocidade permite que as empresas atendam melhor aos seus clientes e consigam competir de modo mais eficaz no mercado" (AMAZON, 2021).

Por ser uma cultura de automação de processos de infraestrutura e de desenvolvimento, as equipes de desenvolvimento e operações não trabalham mais separadas, ou em silos, seja esse silo físico ou cultural. Por exemplo, a Microsoft antes de adotar práticas DevOps, trabalhava com tríades de desenvolvimento, consistindo em um project manager, um desenvolvedor e um testador. Com a adoção do DevOps, as funções de desenvolvedor, testador, operações e diagnósticos foram alocadas na equipe de software (AMERSHI, 2019).

A adoção de DevOps requer uma mudança de cultura e práticas nas organizações: “precisávamos fazer uma mudança cultural em que entregamos mais cedo, com frequência e falhamos rápido... nossa definição de pronto passou de potencialmente entregável para estar em produção e coletando métrica” (WOODWARD, 2018). Com isso, as duas equipes (dev e ops) passam a trabalhar juntas, adotando práticas ágeis e automações, a fim de otimizar a produtividade dos desenvolvedores e a confiabilidade das operações apoiando-se em toda uma toolchain ferramental e cultural. É comumente utilizada a Figura 2 para representar DevOps. Ela simboliza a união das práticas de "Dev"e "Ops"em um fluxo contínuo.

1.4 Workflow DevOps

Há uma variadade de ferramentas com o objetivo de automatizar o pipeline DevOps e facilitar a colaboração entre times multifuncionais.

Em um típico workflow de devops, primeiramente o código é enviado para um serviço de CI/CD. O código é empacotado, testando, verficado a qualidade estática, e a imagem gerada pode ser enviada a uma registro de imagens. Durante a execução do pipeline de CI/CD mensagens (Logs) podem ser enviadas à equipe de DevOps informando

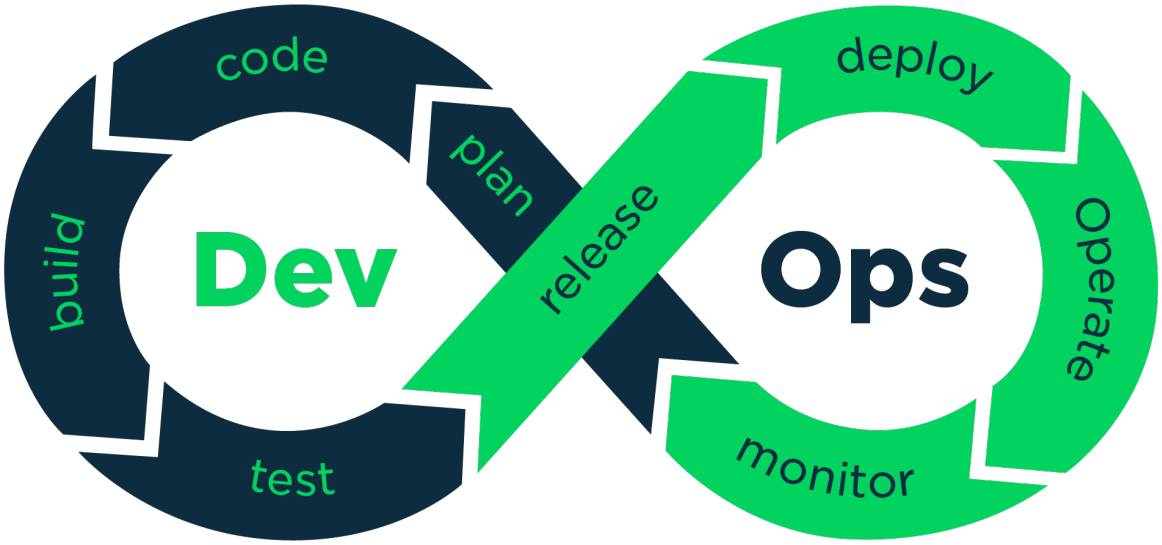

Figura 2 – Imagem adotada pela comunidade de DevOps

Fonte: https://neonrocket.medium.com/devops-is-a-culture-not-a-role-be1bed149b0

sobre erros e demais registros sobre a execução do pipeline.

Após execução do pipeline de CI/CD é feita a implantação, atualização do serviço em ambiente de produção/homologação, tarefa comumente atribuida à equipe de operação. As organizações estão cada vez mais optando por utilizar serviços na nuvem, como infraestrutura como serviço (IAAS: *infrastructure as a service*) ao invés de manter sua própria infraestrutura. Serviços conhecidos e amplamente utilizados de IAAS são: AWS(*Amazon Web Services*), Google Cloud, Microsoft Azure.

As aplicações executadas nas plataformas de IAAS estão na forma de imagens containers, logo, alguma forma de orquestração de containers é necessária. A ferramenta mais popular na orquestração de containers atualmente é o Kubernetes. Além de provisionamento e orquestração dos conteiners, o monitoramento da execução dos mesmos é necessário para notificar e automatizar tomadas de decisão relacionadas aos serviços. Para isso, algumas ferramentas de monitoramento de clusters são Prometheus e Nagios.

Idealmente, ter um ambiente de homolação/stage o mais próximo possível com o de produção, seja para desenvolvimento, execução de testes ou homologação, facilita na entrega contínua. Sendo assim, vários ambientes com a mesma ou similar configuração precisam ser criados. Criar e manter todos esses ambientes é repetitivo e, feitos manualmente, são mais suscetíveis a erros. Uma solução para esse problema é a prática da infraestrutura como código (abreviado como IaC do inglês *infrastructure as code*), no qual criar, configurar e fazer o implantação dos diversos ambientes é realizado de forma automatizada com o auxílio de ferramentas para provisionar a infraestrutura e gerenciar

configuração.

Todo o ferramental da infraestrutura como código é facilmente mantido com auxílio de um repositório, uma vez que o gerenciamento da infraestrutura como código possibilita automatizar todo esse processo.

1.5 MLOps

MLOps, assim como o DevOps, é a colaboração entre equipes de produtos de Machine Learning, composto por cientistas de dados, desenvolvedores de software, e operadores. O nome foi adaptado do DevOps, pois apresenta desafios similares, desde a reestruturação organizacional, adaptação de papéis, automações e cultura. Apesar do termo já ser amplamente utilizado no mercado (CLOUD, 2021), ainda é relativamente novo na comunidade academica.

De acordo com Amershi (AMERSHI, 2019), podemos distribuir o workflow simplificado de MLOps em 9 etapas, que vão desde a coleta dos dados até monitoramento do serviço em ambiente de produção.

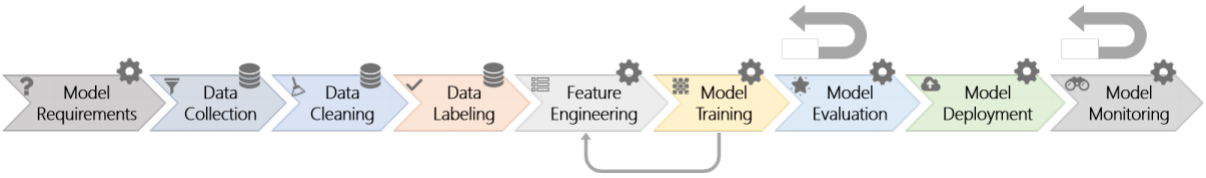

Figura 3 – Workflow MLOPS simplificado

Fonte: (AMERSHI, 2019)

A Figura 3 apresenta as 9 etapas do MLOps e vamos a seguir detalhar as características de cada etapa.

Requisitos do modelo (*model requirements*):

Etapa em que são decididos quais requisitos são viáveis para serem implementados com algoritmos de aprendizado de máquina. Mais importante ainda, nesta fase, é decidido quais tipos de modelos são mais apropriados para determinado problema.

Coleta de dados (*data collection*):

São integrados conjuntos de dados disponíveis. Pode ser treinado um modelo parcial usando conjuntos de dados genéricos disponíveis (por exemplo, ImageNet para detecção de objetos). Em seguida, usar o aprendizado de transferência junto com dados mais especializados para treinar um modelo mais específico (por exemplo, detecção de pedestres).

Limpeza de dados (*Data cleaning*):

Remoção de registros imprecisos ou com ruído do conjunto de dados, uma atividade comum a todas as formas de ciência de dados.

Rotulagem de dados (*Data labeling*):

Atribui rótulos a cada registro. A maioria das técnicas de aprendizagem supervisionada requer rótulos para induzir um modelo. Outras técnicas (por exemplo, aprendizagem por reforço) usam dados amostrais ou padrões ambientais para ajustar suas regras. Os rótulos podem ser fornecidos pelos próprios engenheiros, especialistas no domínio ou por *crowd workers* em plataformas online de *crowdsourcing*.

Engenharia de features (*Feature engineering*):

Refere-se a todas as atividades realizadas para extrair e selecionar features para modelos de aprendizado de máquina. Para alguns modelos (por exemplo, redes neurais convolucionais), esta etapa é menos explícita e muitas vezes combinada com a próxima etapa, o treinamento do modelo.

Treinamento de modelo (*model training*):

Os modelos escolhidos (usando os recursos selecionados), os hiperparâmetros são calibrados e os modelos são treinados e ajustados aos dados limpos coletados e seus respectivos rótulos.

Avaliação de modelo (*model evaluation*):

São avaliados os dados de saída em conjuntos de dados testados usando métricas predefinidas. Para domínios críticos, esse estágio também pode envolver avaliação humana extensiva.

Deploy e monitoramento(*model deploy, monitoring*):

O código de inferência do modelo é o implantado. E passa a ser monitorado continuamente para possíveis erros durante a execução no mundo real.

1.5.1 Pipelines MLOps

Com a introdução de machine learning nas pipelines de DevOps, algumas diferenças notáveis acontecem, por exemplo: A integração contínua além de testar e validar o código também passa a testar e validar os modelos, dados e esquemas de dados. A entrega contínua, já não é mais apenas responsável por fazer *deploys*, ela também passa

a executar toda uma *pipeline* de ML para criar modelos de previsão para então fazer o *deploy* destes. O treinamento contínuo(CT) passa a ser inserido, este por sua vez é uma nova propriedade, exclusiva para sistemas de ML, que se preocupa em treinar e exibir automaticamente os modelos.

A documentação do *google cloud* (CLOUD, 2021) na sua seção de machine learning, possui um tópico em que uma pipeline de ML é descrita passo a passo. O foco desse modelo apresentado está na automatização de uma pipeline de ML de forma que é introduzida o treinamento contínuo dos dados e modelos (Fig. 4)

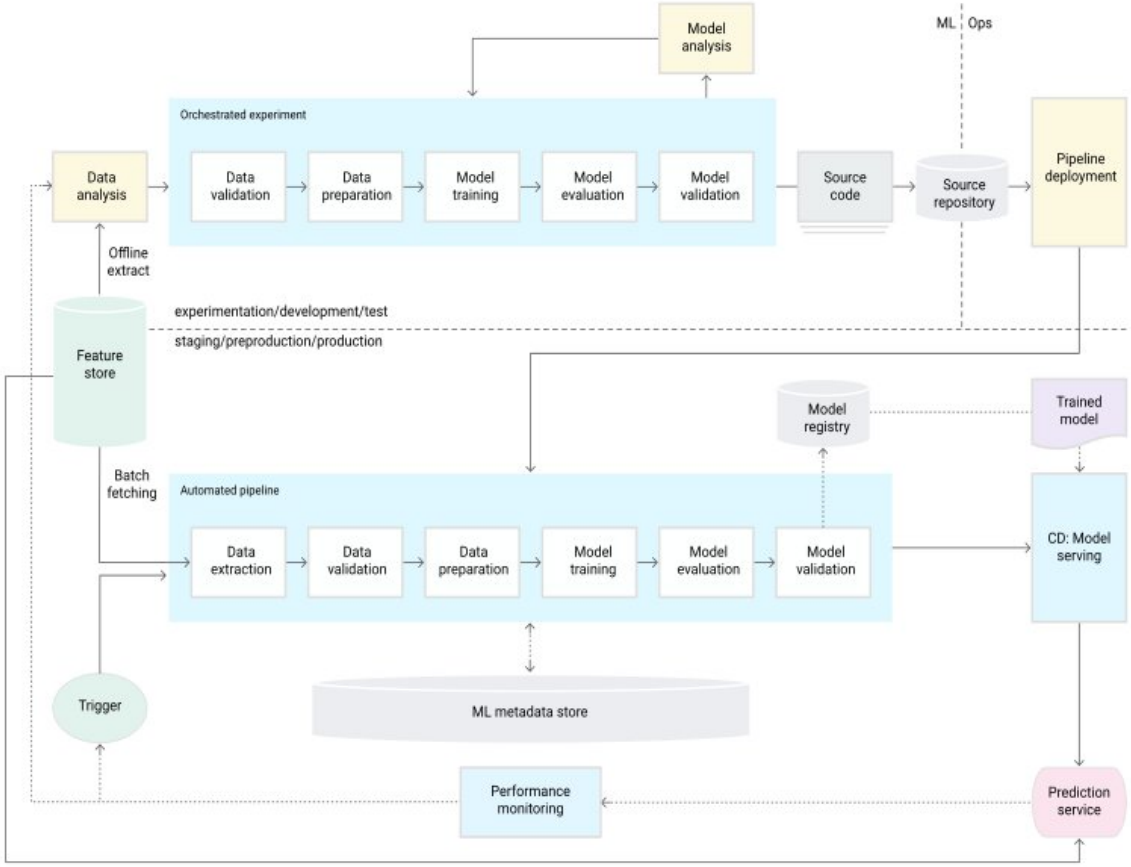

Figura 4 – Modelo de automação de pipeline de ML

Fonte: (CLOUD, 2021)

Uma característica notável desse modelo é a presença de uma Feature Store, que nas palavras de Jim Dowling em sua apresentação na *PyData London Meetup* é "uma forma de encontrar features em um mar de dados" (DOWLING, 2019).

Em machine leaning, *features* são dados usados como um sinal de entrada para um modelo. Mike Del Balso (BALSO, 2020) em seu artigo sobre features stores exemplifica uma feature de uma situação hipotética: "se uma empresa de cartão de crédito está tentando prever se uma transação é fraudulenta ou não, uma feature útil poderia ser se a

transação foi feita em um país ou no exterior. Outra forma de detecção dessa fraude seria comparar o valor dessa transação com outras compras do mesmo cliente".

Identificar e criar *features* é um processo da engenharia de *features*, crítico para a criação de modelos de machine learning. Por ser uma etapa crítica, ela demanda um maior esforço dos cientistas de dados, parte dessa dificuldade se deve ao fato de não haver um lugar centralizado onde pode-se facilmente buscar features, muitas vezes esses dados estão dispersos em várias fontes de dados. Sendo assim, ter uma forma de centralizar o acesso às features é um componente obrigatório para o sucesso de uma pipeline de MLOps.

Feature Store é a camada responsável por gerenciar o armazenamento e busca das features. Mike lista algumas das vantagens de se utilizar uma Feature Store:

- Produtizar novas features sem a necessidade de um grande esforço de engenharia;
- Automatizar o cálculo e processamento de features, backfills, e geração de logs;
- Compartilhamento e reutilização de pipelines de features entre times;
- Versionamento de features, linha do tempo (lineage) e metadados;
- Garantir consistência entre dados de treino e do mundo real;
- Monitoramento da saúde dos pipelines de features em produção.

Feature Store é o processo que transforma dados brutos em features, armazena essas features e as serve para modelos.

"As Feature Stores gerenciam todos os recursos relacionados às funcionalidades (processamento, armazenamento, serving) de forma holística através do ciclo de vida dessa funcionalidade. Ao automatizar as tarefas repetitivas de engenharia necessárias para produtizar uma funcionalidade, elas criam um caminho simples e rápido para a produção. Além disso, elas permitem o gerenciamento de otimizações (como retirar features que não estão sendo usadas por nenhum modelo, ou retirar duplicidades de features entre modelos), que traz uma eficiência significante, especialmente à medida que as equipes aumentam cada vez mais a complexidade do gerenciamento manual de funcionalidades"(BALSO, 2020).

1.5.2 Ferramentas de automação Open Source

Em um modelo mais simples e tradicional de machine learning o workflow se resume a obter dados, fazer limpezas nos dados, selecionar as features desejadas, construir um modelo e quando se estiver satisfeito colocar esse modelo em uma aplicação. Porém, conforme o time cresce, os requisitos crescem e consequentemente a complexidade do

projeto como um todo cresce, servir esses modelos vai ficando cada vez mais difícil, ao ponto que até mesmo fazer o rastreamento de problemas se torna um problema. Dessa forma, idealmente, a infraestrutura por baixo desses projetos também deve crescer para acomodar as crescentes demandas.

Felizmente, a própria comunidade está se organizando para facilitar a adoção de ferramentas que podem ser usadas em cada segmento de uma pipeline de MLOps. Um exemplo de iniciativa é a o repositório *Awesome MLOps* que possui uma lista de ferramentas utilizadas em MLOps separadas pela sua área de uso. O *Awesome MLOps* pode ser acessado em https://github.com/kelvins/awesome-mlops.

2 Proposta

A proposta da pesquisa é explorar os desafios técnicos relacionados à adoção de MLOps por engenheiros de software com experiência em DevOps. Para isso definimos a seguinte questão de pesquisa (QP):

QP - Quais são os desafios técnicos na adoção de ferramentas de automação no contexto de MLOps para profissionais com experiência em DevOps?

Para responder essa questão de pesquisa, será realizada uma pesquisa exploratória, no qual o pesquisador irá executar o processo de adoção de uma ferramenta de automação MLOps do projeto Rasa Boilerplate. O Rasa Boilerplate foi escolhido por este já possuir toda uma gama de ferramentas *Open Source* prontas para serem utilizadas e eferece um modelo para a adição de novas ferramentas.

2.1 Metodologia

A metodologia utilizada neste estudo é pesquisa exploratória e prática. Primeiramente, foi feita uma revisão da bibliografia, não só em artigos acadêmicos, mas também textos da literatura cinzenta, composto por posts de blogs, guias técnicos de empresas, e documentação de comunidades de open source. O objetivo da revisão bibliográfica é mapear o estado da prática da indústria de software e da pesquisa em relação à MLOPs. A partir do estudo das práticas adotas pela comunidade DevOps em ambientes de machine learning indicadas pela literatura e utilizadas pelo mercado, foram mapeadas características que possam categorizar, qualificar e identificar o nível de maturidade de ferramentas de automação OSS de MLOps e suas comunidades.

Dentre as características levantadas, ter o suporte de uma comunidade ativa é crítico para adoção da ferramenta.

O cientista de dados é um novo perfil no time de projetos de produtos de machine learning, presente em boa parte do processo e tomadas de decisões importantes. Logo, ferramentas e automações e abstraiam complexidade de treinar modelos, versionar dados de treinamentos e instanciar modelos na núvem foram o maior foco do presente trabalho. A partir das ferramentas de MLOps mapeadas e descritas, foram identificadas aquelas que tem como objetivo auxiliar o workflow do cientista de dados e sua interação com DevOps. O processo de escolha da ferramenta foi documentado e analisado. Foram analisados os conceitos necessários para a compreensão da ferramenta no ecossistema de entrega contínua de produtos de machine learning e paralelo com ferramenta de automações DevOps. O suporte da comunidade e da documentação técnica, foram examinados

ao executar tutoriais iniciais (get started) propostos pela comunidade. O objetivo dessa etapa é mapear as dificuldades que um engenheiro com conhecimento prático em DevOps enfrentaria para adotar uma ferramenta de MLOps para auxiliar o time de ciência de dados. E se a ferramenta escolhida seria adotada pela equipe de ciência de dados com baixa curva de aprendizagem. Após a escolha da ferramenta, ela será configurada para um projeto real, com módulos de machine learning e testada, a fim de mapear os reais benefícios de tal automação.

Memorandos são gerados continuamente durante toda adoção da ferramenta pelo pesquisador. Dificuldades técnicas, de comunicação com a comunidade, complexidades não previstas, conceitos nessários para entendimento das ferramentas são registradas e posteriormente analisadas qualitativamente. Os resultados e lições aprendidas no processo contribuem com a comunidade de profissionais de engenharia de software e DevOps no levantamento dos principais desafios e benefícios obtidos na adoção de uma ferramenta de automação de projetos de machine learning.

2.2 Resultados

2.2.1 Categorização e Qualificação

Não há um guia específico de como qualificar e rankear a qualidade de softwares similares. A falta de critérios objetivos para comparação de soluções similares, pode levar a um processo adoção não otimizada de ferramentas.

Sendo assim, tendo como base a lista de ferramentas MLOps disponíveis como open source em Awesome MLOps. O fato de quase todos esses softwares estarem acessíveis no github, é possível tomar como ponto de partida as métricas presentes no próprio github, que qualificam a maturidade da comunidade open source e o tamanho da comunidade, tais como apresentado no Quadro 1.

Quadro 1 – Características que qualificam uma comunidade de open source.

Qualificador	**Indicativo**
Tipo de licença	Este estudo limita-se a somente a ferramentas de código aberto.
Número de estrelas	Como um indicativo de interesse na ferramenta por parte da comunidade
Número de forks	Como um indicativo de interesse de crescimento e contribuição.
Relação issues abertas x fechadas	Como um indicativo de maturidade no desenvolvimento.
Último commit na branch default	Como um indicativo desenvolvimento ativo na ferramenta.
Total commits branch default	Como um indicativo de comunidade ativa e possivelmente saúde da ferramenta

Outros qualificadores poderão ser adicionados bem como outras fontes de software, mas, pelo menos neste estudo preliminar, somente os descritos no Quadro 1 foram observados. Dentre estes, os qualificadores tipo de licença e data do último commit na branch principal são de caráter eliminatório, os demais servirão para melhor qualificar as ferramentas em suas respectivas categorias.

O Awesome MLOps distribui as ferramentas em categorias que abrangem todas as necessidades de um fluxo de MLOps. Mas, para termos de simplificação e alinhamento com a pipeline da Figura. 4, podemos redistribuir algumas dessas categorias de acordo com o Quadro 2.

Quadro 2 – Categorias de ferramentas de MLOps, de acordo com o Awesome MLOps.

Pipeline MLOps	**Awesome MLOps**
Data/Model Analysis	Data Exploration, Data Management, Data Processing, Data Validation, Data Visualization, Simplification Tools
Feature Store	Feature Store
ML Metadata Store	Model Lifecycle
Monitoring	Cron Job Monitoring, Model Lifecycle, Visual Analysis and Debugging
Model Serving	Model Serving, Optimization Tools
Pipeline Orchestration	Workflow Tools

Tomando como ponto de partida as ferramentas do Awesome MLOps, foi criada uma planilha no google docs com os qualificadores e as áreas de atuação. Alinhado a essa planilha há um script que percorre os repositórios listados na coluna Github e atualiza os dados desta além de gerar um arquivo csv com o dia da execução como nome, para poder futuramente fazer um estudo de como os qualificadores de cada ferramenta mudam ao longo do tempo. Tanto a planilha quanto o código que faz a atualização dos dados podem ser acessados pelos links:

- Script MLOpsTools - https://github.com/fabio1079/MLOpsTools
- Planilha - https://github.com/fabio1079/MLOpsTools/blob/main/planilhas

2.2.2 Licenças utilizadas

Como descrito nos qualificadores, este estudo limita-se na utilização de ferramentas de licença de código aberto. Felizmente, umas das vantagens do Awesome MLOps é que a grande maioria das ferramentas listadas são de código aberto, como pode ser visto na Figura 5.

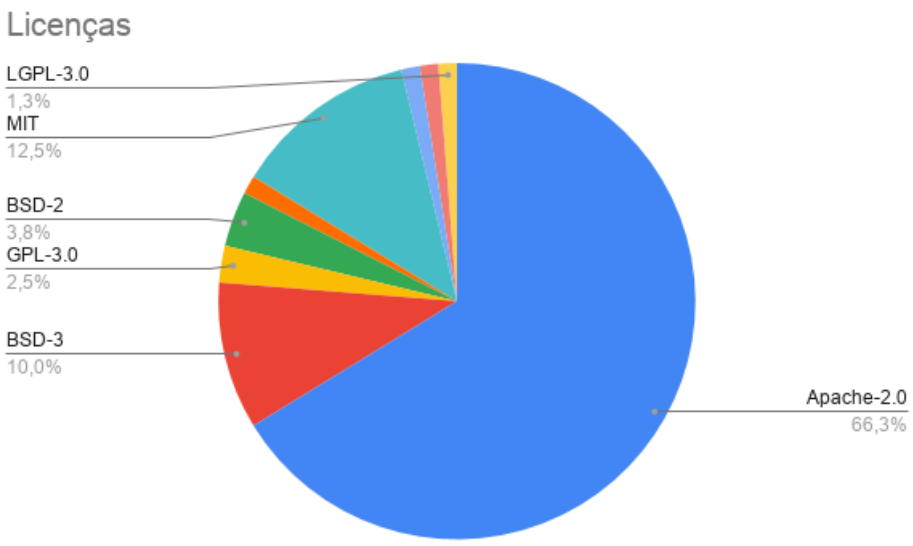

Figura 5 – Licenças mais usadas nas ferramentas analisadas

2.2.2.1 Licença Apache-2.0

Como visto na Figura 5 a licença mais utilizada por projetos de machine learning(dentre os listado na Awesome MLOps) é a licença Apache-2.0. Como essa licença permite livre uso, redistribuição e alteração, sem exigir reciprocidade, Faz com que projetos que usam essa licença, possam ser proprietários que é o caso de muitas das aplicações de ML executando mundo afora.

2.2.3 Último commit na branch default

Este qualificador é de caráter eliminatório. Um dos motivos de se ter colocado a data do último commit na branch default como um qualificador foi para poder filtrar ferramentas que estariam possivelmente sendo descontinuadas pelos seus desenvolvedores. Analisando os dados da planilha é possível notar que há ferramentas listadas no Awesome MLOps cuja data do último commit na sua branch default é maior que 6 meses. Sendo assim essas ferramentas já podem ser consideradas eliminadas neste estudo como um todo.

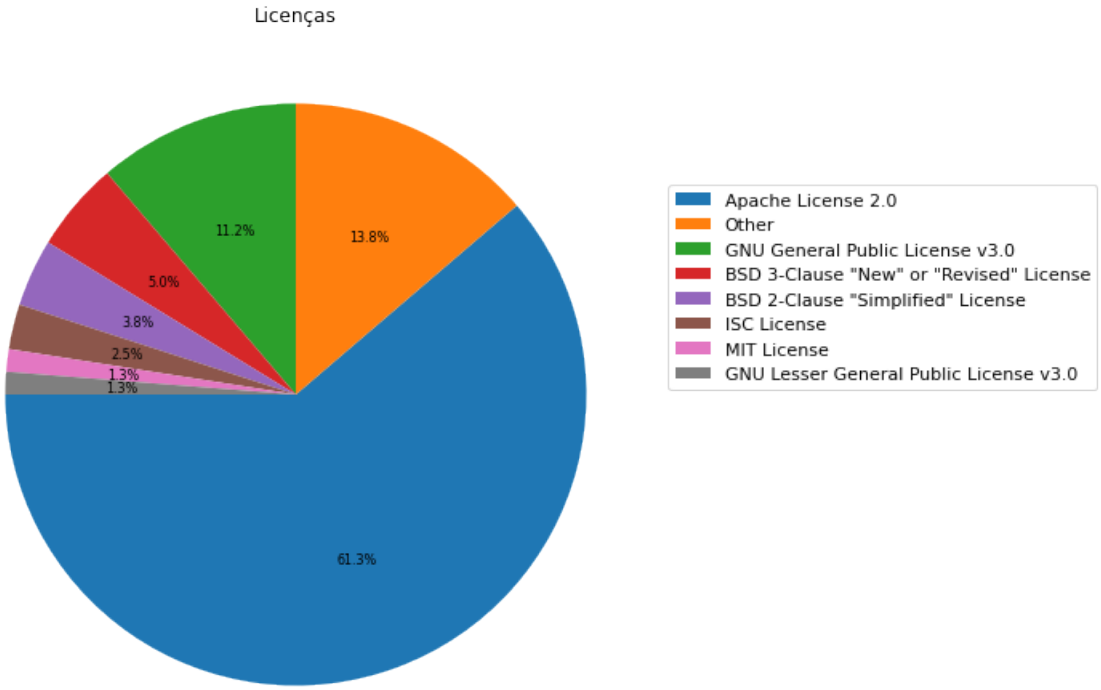

Figura 6 – Licenças mais usadas com base na API do Github

2.2.4 Observações sobre os dados dos qualificadores

Com base nos dados obtidos pela api do github sobre os qualificadores foi possível fazer algumas observações. Por exemplo, em relação às licenças utilizadas, de acordo com a api do Github, 13.8% delas são “Other” que é quando o Github não consegue identificar qual licença está sendo usada. O que indica que em relação ao qualificador licença, quando o resultado da api do github for "Other", uma análise manual é mais indicada. Um gráfico que demonstra como essa falha da api afeta a análise das licenças pode ser visto na Figura 6.

A diferença entre as Figuras 5 e 6 pode ser explicada no fato de que primeira foi elaborada com na base nos dados da planilha, sendo essa, criada antes do script que utiliza a api do github, sendo assim, foi feita uma análise manual da licença utilizada por cada ferramenta. Atualmente o script elaborado não está sobrescrevendo as licenças na planilha.

Em uma tentativa preliminar de se obter um rank de atratividade das ferramentas perante suas comunidades, utilizando-se de uma média aritmética entre número de estrelas, forks e observadores de cada repositório, obteve-se os seguintes dados.

Ao todo, 5 ferramentas foram desqualificadas por falharem no requisito da data do último commit na branch principal ser menor que seis meses. Dentre as 73 restantes, obteve-se uma média de 2638 em nível de atratividade. Porém, vendo a disparidade

entre as avaliações em áreas distintas pela Figura 7, é de se notar a ingenuidade dessa abordagem.

Sendo assim, um possível ranqueamento das ferramentas apesar de ser uma métrica ideal, não é algo que se pode concretizar apenas com os dados obtidos pelo script MLOpsTools.

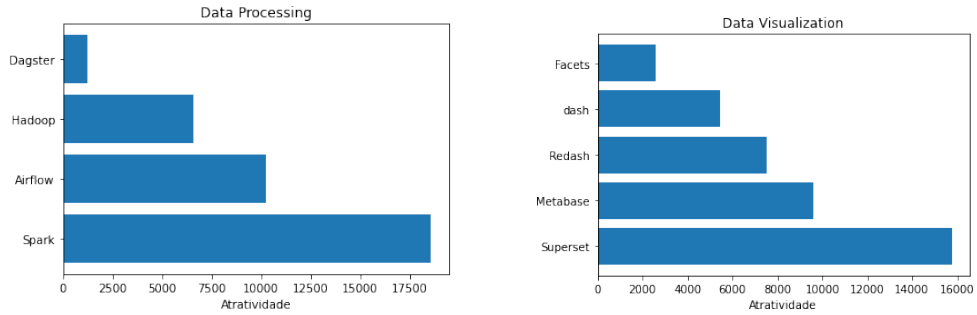

Figura 7 – Disparidade na atratividade das ferramentas

2.2.4.1 Outras formas de caracterizar a maturidade de uma ferramenta

Em relação as informações da comunidade de uma ferramenta que poderiam auxiliar nesta possível análise de maturidade, temos como exemplo a dinamicidade das issues. Em que pode se levar em consideração quanto tempo uma issue fica aberta, assim podemos dizer que uma comunidade é mais ativa, quanto menor for o tempo que as issues da ferramenta permanecem abertas, inversamente, pode-se atribuir valores negativos caso haja issues que permanecem um longo período abertas, indicando um desinteresse da comunidade e seus mantenedores.

Indo além da sua comunidade e olhando outros fatores mais técnicos, um aspecto que faz ficar caro adotar uma ferramenta OSS, são as suas dependências, por exemplo, quando uma ferramenta é adotada, indiretamente também é adicionado o custo de suas dependências. Sendo assim, também colocar as dependências em análise torna-se um dado interessante mas, para fins de simplificação, limitar a profundidade desse tipo de análise também seria importante para não ficar preso analisando dependências de dependências de dependências.

Há vários outros dados que também poderiam ser utilizados como métricas, vulnerabilidades e etc. Mas não dá pra criar um modelo de análise perfeito e com isso alguns fatores teriam que ser desconsiderados. Sendo assim para a continuidade deste estudo, é proposto a criação de um modelo para analisar a maturidade de ferramentas OSS de MLOps que seja tangível, com base em estudos acadêmicos.

2.3 Escolha da ferramenta

Mas, dado que MLOps é um guarda-chuva que abrange várias atividades, é necessário focar em uma que atenderia a maioria dos problemas enfrentados quando se tratando na adoção dessas práticas, principalmente pelo ponto de vista do cientista de dados.

Com o auxílio das características levantadas no Quadro 1 onde ser de alguma variante de licença open source e ter o suporte uma comunidade ativa foram as que mais se destacaram e analogamente, caso a ferramenta não fosse de licença open source e ou sua comunidade tivesse pouca atividade(baixo número de estrelas e forks, longos períodos sem novos commits) seriam características desqualificadoras.

Ao analisar as ferramentas, ficou evidente a influência que grandes empresas têm sobre os repositórios de MLOps, nomes como Netflix, Spotify e Microsoft estão entre os mantenedores das ferramentas mais utilizadas. Não que isso seja um problema e sim um indicativo da importância e destaque que MLOps vem tendo, ao ponto que mesmo grandes empresas estão dedicando parte dos seus esforços em manter essas ferramentas e ainda disponibilizá-las como open source.

Como um todo, o processo de adoção de MLOps é árduo, longo e complexo, é necessário a preparação de todo um ambiente de ML CI/CD, porém, construir do zero esse ambiente é um grande compromisso que irá demandar bastante esforço e consequentemente, recursos. São poucas as empresas com a disposição necessária para trilhar esse caminho, o que exemplifica um dos motivos de nomes de grandes empresas se destacarem no repositórios de MLOps, já que elas têm os recursos necessários para manter essas ferramentas.

Porém, nem tudo são flores, apesar de ter o suporte de grandes empresas é um bom sinal de que a ferramenta será devidamente mantida, isso também significa que a ferramenta está associada às necessidades e demandas dessa empresa em primeiro lugar. E quando adotada, irá acarretar em também adotar as escolhas que a empresa desenvolvedora estiver executando. Este é o preço a se pagar por optar por ter o suporte de uma grande empresa.

Mas, olhando as que melhor auxiliam o cientista de dados e considerando que este profissional já tem o seu próprio arcabouço de outras ferramentas que utiliza, dadas as necessidades do projeto que estiver trabalhando, optou-se por focar em ferramentas que ajudam em orquestrar o fluxo de trabalho, dentre essas há alguns nomes que se destacam como a Mlrun, Luigi e Metaflow. Todas focadas em gerenciar a execução de tarefas e armazenamento e manipulação dos artefatos gerados pelas tarefas. Olhando os seus repositórios, todas possuem bons qualificadores, especialmente números de estrelas o que denota a atratividade dessas ferramentas.

Quando olhando as funcionalidades que elas possibilitam, uma se destaca por

facilitar a conexão entre o ambiente local do cientista e o ambiente de homologação em um servidor remoto. Em que os dados são todos processados em servidores na nuvem, porém a integração que essa ferramenta possibilita, faz com que a experiência de execução seja como se tudo tivesse sido executado na máquina do próprio cientista, facilitando a orquestração de tareas com um simples comando pelo lado do cientista. Essa ferramenta é a Meraflow.

Sendo assim, levando em conta as necessidades de um cientista de dados, foi escolhida a ferramenta Metaflow para esse estudo de caso, o porque dessa ferramenta em específico ter se destacado dentre as demais e como ela poderia ser útil é melhor abordado nas seções subsequentes.

2.3.1 Metaflow

O Metaflow se destaca por ser uma ferramenta voltada ao gerenciamento e execução de algoritmos de ciência de dados que vai desde a prototipagem à implementação final do modelo. O Metaflow foi criado pela Netflix para dar a seus cientistas de dados um ambiente capaz de proporcionar uma fácil manipulação dos dados utilizando do potencial ferramental da Amazon AWS sem que os cientistas de dados tenham que se preocupar com a parte de infraestrutura que seus modelos requisitam, auxiliando inclusive com problemas de escalabilidade, configuração de dependências e tolerância a falhas. Do ponto de vista do usuário da ferramenta, não há uma barreira entre o seu ambiente local e a nuvem, o Metaflow realiza a integração entre esses ambientes, bastando ao cientistas de dados criar suas pipelines sequenciais ou paralelas , realizar o treinamento dos modelos e coletar os resultados.

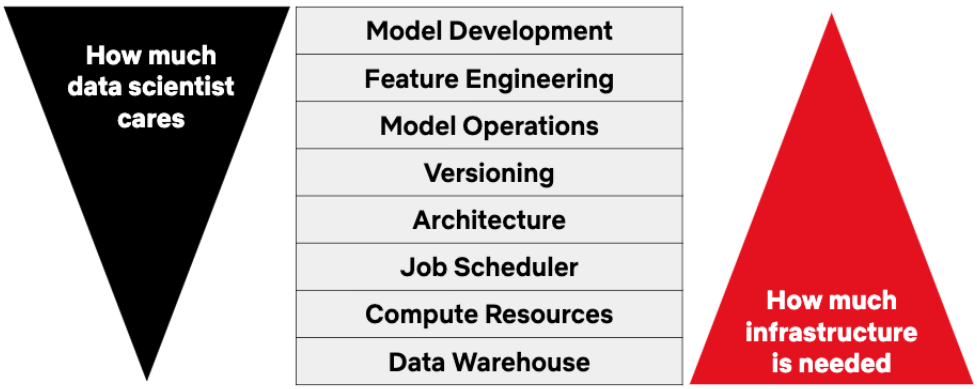

Figura 8 – Infraestrutura para ciência de dados

Fonte: https://docs.metaflow.org/introduction/what-is-metaflow

Um dos objetivos do Metaflow é simplificar para o cientistas de dados o que ocorre no lado da infraestrutura fornecendo uma abordagem unificada e amigável em que inter-

namente, o Metaflow aproveita a infraestrutura existente da Amazon AWS e faz a parte de integração entre o ambiente local e a nuvem.

Nas próprias palavras da documentação oficial da ferramenta: *"Projetos de ciência de dados bem-sucedidos são entregues por cientistas de dados que podem construir, melhorar e operar fluxos de trabalho ponta a ponta de forma independente, focando mais em ciência de dados, menos em engenharia."*(https://metaflow.org)

2.3.2 Experimento

Na atualidade, está se tornando cada vez mais comum o uso de *chatbots* em *web sites*, programas esses que se utilizam de aprendizado de máquina para proporcionar uma conversação mais humana e possuem uma grande variabilidade de utilizações práticas. Por serem produtos advindos da aprendizagem de máquina, precisam de modelos e treinamento, com a diferença que os dados de teste são a interação entre pessoas e o *chatbot*. Sendo assim, o objetivo do experimento é verificar se com o auxílio do Metaflow é possível realizar as etapas de treinamento do modelo do *chatbot*, disponibilizar o *chatbot* para testes com usuários e coletas de dados dessas interações. Nesse experimento, o treinamento e disponibilização do *chatbot* para testes deve ocorrer na nuvem pelo uso do Metaflow.

De forma geral, o objetivo é analisar se com o auxílio da ferramenta, um cientista de dados consegue colocar o seu modelo em testes e coletar dados das respostas dos usuários sem a necessidade de um grande investimento em questões relacionadas à infraestrutura necessária para a execução desses testes. A ferramenta deve ser capaz de tirar parte desse esforço das mãos do cientista de dados, permitindo que ele foque no aprimoramento do modelo que ele está trabalhando.

2.3.2.1 Rasa Boilderplate

Pela própria descrição no repositório da ferramenta: "*O boilerplate nasceu como uma abstração genérica do projeto Tais. Hoje, tem o objetivo de tornar mais fácil a criação de um chatbot Rasa. Com a evolução do framework, atualmente o foco do boilerplate é uma documentação em código viva.*" (TEMPORIM, 2021). O Rasa Boilder Plate pode ser obtido acessando o repositório:
https://github.com/lappis-unb/rasa-ptbr-boilerplate

Com o auxílio desse boilerplate é possível preparar um ambiente Rasa completo, com suporte a integração com Telegram e Rocket Chat e ainda oferece a possibilidade do uso de Jupyter Notebook, ElasticSearch e RabbitMQ. O uso de todas essas ferramentas e integrações é totalmente opcional, o boilerplate apenas facilita já oferecendo as configurações necessárias. Para esse experimento optou-se em utilizar apenas o módulo básico do Rasa e integrar o Metaflow ao boilerplate de forma similar a como as outras ferramentas

são integradas nele.

2.3.2.2 Preparando o Metaflow

Para o settup do Metaflow apesar de tecnicamente ser possível colocar a ferramenta em qualquer provedor de nuvem, na prática, devido a experiência dos criadores com a infraestrutura oferecida pela Amazon AWS, a ferramenta está por enquanto, suportando apenas a nuvem da AWS como back-end remoto. Mas isso não significa que ela só possa ser utilizada quando conectada a algum back-end, é possível utilizar de algumas funcionalidades que podem ser executadas localmente e já oferecem um ganho de flexibilidade no controle de projetos de aprendizado de máquina.

Tanto que, para cada recurso oferecido pelo Metaflow, há um serviço na AWS que será relacionado a esse recurso, exemplo: Datastore utiliza da Amazon S3 e Scheduling utiliza AWS Step Functions. Sendo assim, para poder fazer a devida configuração do Metaflow em produção, da forma como a ferramenta está atualmente, é obrigatório o uso dos recursos da Amazon ASW.

Para auxiliar na configuração do Metaflow em produção, a documentação da ferramenta oferece um guia com o passo a passo. Esse manual é longo e possui muitas etapas que podem facilmente confundir ou deixar perdido um usuário da AWS com pouca experiência na plataforma.

Durante o preparo desse experimento, ao seguir a documentação, alguns pontos não ficam bem claros, como por exemplo nos valores dados para serem utilizados por variáveis de ambiente no Metaflow, são muitas as variáveis que precisam ser configuradas e algumas descrição delas usam um vocabulário que claramente faz mais sentido para um usuário mais experiente com a AWS.

Outro fator a ser levantado é que o processo descrito no manual é bem longo e por se tratar de uma configuração tão alinhada aos recursos da AWS, em alguns pontos, há mudanças de interface por parte dos serviços da AWS que não batem da forma como está descrito no manual. O que complica ainda mais a instalação do Metaflow em produção, já que conforme ocorrem mudanças na forma como os serviços da AWS se conectam pela sua interface de configuração, a documentação do Metaflow acaba por não conseguir corresponder corretamente, deixando para a pessoa que estiver realizando essa etapa, buscar em como está a forma atual de realizar essa parte específica no modelo atual da interface da AWS.

2.3.2.3 Execução do experimento

Como descrito anteriormente optou se por integrar o Metaflow ao Rasa Boilerplate de forma similar a como outras ferramentas já estão integradas ao boilerplate. A arquite-

tura do boilerplate pode ser divida em 2 partes principais, criar e prover. Em criar ficam as etapas relacionadas a inteligência do Chatbot e em prover estão as integrações com serviços que permitem ao usuários de diversas plataformas se comunicarem com o bot. O Metaflow poder ser facilmente integrado na parte de criar da arquiterura do boilerplate.

Com o auxílio do Metaflow é possível realizar uma melhor seleção de modelos de ML, que no caso do Rasa, seria qual treino do *chatbot* melhor se adequa às interações com o usuário, algo próximo de um teste AB. É de se entender que para a realização desse tipo de teste, normalmente seria necessário o preparo de dois bots, subindo ambos em um ambiente que usuários possam interagir e coletar os resultados desse teste. Todo esse trabalho para a realização desse tipo de experimento envolve diversos profissionais, somente o cientista de dados não seria capaz. Com isso, o objetivo do uso do Metaflow nesse cenário, seria para validar se com apenas o cientista de dados executando um ao alguns comandos já seria o suficiente para treinar e colocar ambos os bots para teste com usuários. Assim facilitando o processo de desenvolvimento e testes e dando mais poder de controle aos desenvolvedores do bot.

Para tal é necessário a utilização de uma funcionalide do Metaflow, a execução paralela de etapas, onde há um ponto de entrada que realiza as configurações necessárias e então são executadas de forma paralela o treinamento de ambos os modelos do chat. A etapa de treino do bot demanda bastante processamento, mas com o Metaflow essa execução ocorre na nuvem e o cientista de dados apenas colhe esses resultados no seu ambiente local.

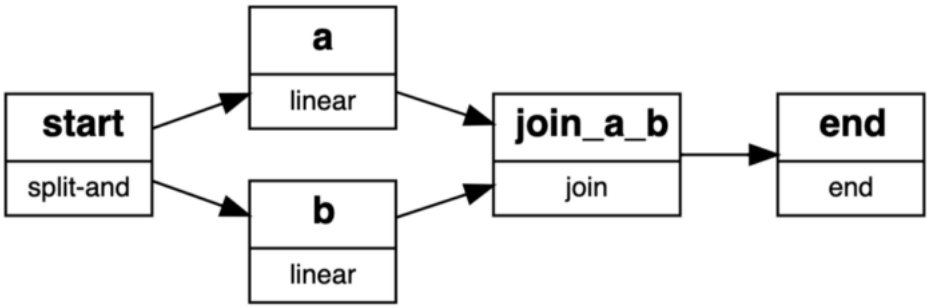

Figura 9 – Execução paralela de etapas do Metaflow

Fonte: https://miro.medium.com/max/2400/1*HuQLSqRj9oKjNLEwCPtaKw.png

A nível de código, as etapas paralelas são apenas funções da linguagem python que apontam para uma mesma etapa(*step* na notação de decoradores do Metaflow) para fazer a junção delas, o que simplifica e facilita para o cientista de dados a execução de várias tarefas de forma paralela usando apenas funções simples em python.

Após o treinamento dos modelos, basta executar o bot. Dentro do guarda-chuva de ferramentas oferecido pelo Rasa há o Rasa X, uma ferramenta que se utiliza desen-

```python
@step
def run_model_a(self):
    print("*"*80)
    print("rasa run A")
    rasa.run(self.model_path_a,
             self.endpoints_a,
             credentials=self.credentials_a,
             port=self.port_a)
    self.next(self.join)

@step
def run_model_b(self):
    print("*"*80)
    print("rasa run B")
    rasa.run(self.model_path_b,
             self.endpoints_b,
             credentials=self.credentials_b,
             port=self.port_b)
    self.next(self.join)

@step
def join(self, inputs):
    print("*"*80)
    print("join branchs")
    self.next(self.end)
```

Figura 10 – Exemplo de código em execução paralela pelo Metaflow

volvimento orientado a conversa, em que as conversas com usuários oferecem dados que são utilizados para melhorar a inteligência artificial do bot. Porém, infelizmente ao tentar instalar o Rasa X ocorrem vários erros de conflitos com dependências com o Metaflow e não é possível instalá-los separadamente visto que o Metaflow precisa executar o rasa, nessa parte fica bem visível o peso que as dependências de uma ferramenta exercem como já descrito em 2.2.4.1.

Mas, felizmente existem diversas interfaces gráficas como a rasa-webchat ou chatbot-widget por exemplo, que possibilitam a um usuário se comunicar com o bot, porém perdendo as vantagens oferecidas pelo Rasa X. Além de que a instalação de uma interface de conversa é mais uma etapa fora do escopo das atividades do cientista de dados e provavelmente iria depender do suporte do time de DevOps para levantar mais essa ferramenta online apesar de algo trivial tecnicamente mas ainda assim é um esforço desnecessário por parte de um cientista de dados que só quer testar o seu bot com usuários.

Conclui-se com a execução desse experimento, que embora seja possível realizar esse tipo de teste, a melhor solução que seria com a utilização do Rasa X para um melhor aperfeiçoamento do bot e dados os conflitos de instalação acaba que sendo inviável no momento. E ainda tem que se levar em consideração a utilização de mais uma ferramenta fora do escopo de conhecimento do cientista de dados como a chatbot-widget. Não que o Metaflow seja uma ferramenta ruim, justo pelo contrário ela é bem confiável e útil, apenas que essa combinação de chatbot + metaflow não seria uma alternativa boa visto que o

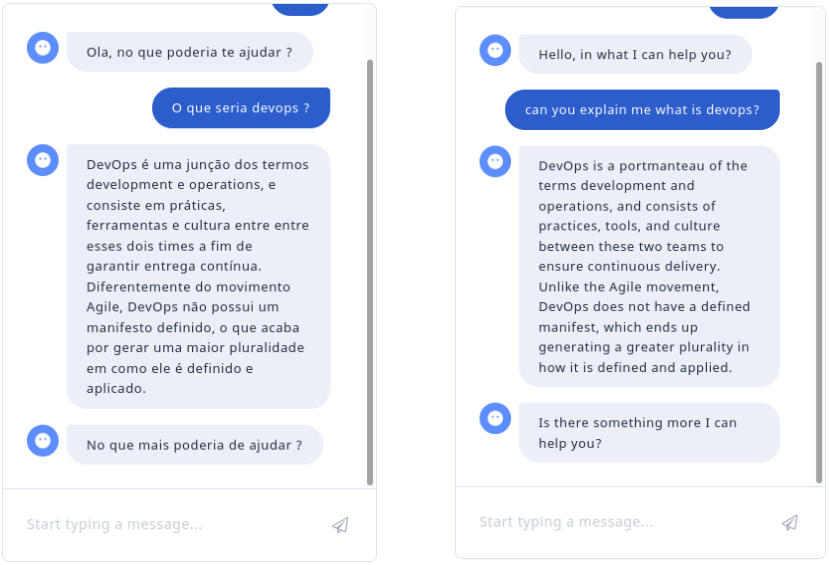

Figura 11 – Conversando ambos os bots usando um chatbot-widget

Rasa X oferece um arcabouço melhor para o aperfeiçoamento do bot. Mas ainda assim, a integração do Metaflow com Jupyter Notebook e TensorFlow pelo menos para projetos que se utilizam dessas ferramentas e possuem muitos dados a serem processados é sim algo que vale pena.

2.4 Lições aprendidas

Nessa jornada de aprendizado, foi possível notar alguns detalhes, sobre os desafios de implementação e adoção de ferramentas de MLOps. Primeiramente, podemos ressaltar que a pluralidade de ferramentas a disposição é uma faca de dois gumes, já que embora seja bom ter várias alternativas, isso ao mesmo tempo dificulta na escolha uma delas.

2.4.1 Escolha da Ferramenta

Projetos de aprendizado de máquina possuem necessidades bem intrínsecas dessa área, o que acaba por escapar dos conhecimentos mais comuns de um engenheiro de software mais voltados para DevOps, não que não haja intersecções, mas é uma área recente com demandas novas.

Como essa área do conhecimento ainda está passando com um rápido crescimento, não há uma comunidade sólida, mas é possível notar pela adoção de grandes empresas que liberam as suas soluções de MLOps como open source bem como em repositórios similares ao Awesome MLOps que sinalizam que há um interesse de se fomentar uma comunidades dedicadas ao crescimento de MLOps.

No caso do porquê da escolha da ferramenta Metaflow é justamente por causa do foco que dá ao cientista de dados sendo este o profissional que mais conhece as necessidades

de machine learning, sendo que a ferramenta dá ao cientista a possibilidade de ter o seu modelo sendo colocado em produção de forma mais rápida e assim podendo coletar mais dados para o enriquecimento e aperfeiçoamento do modelo. Não que isso exclua o profissional de DevOps, até porque ainda é preciso fazer toda a configuração da ferramenta na AWS, o que não é algo trivial e requer bons conhecimentos de infraestrutura.

Um dos motivos da criação do Metaflow pela Netflix foi justamente pelo fato que vários times dentro da empresa tinham necessidades bem distintas quanto a machine learning e precisavam estar devidamente operando em produção, coletando métricas e realizando todo o ciclo de evolução do algoritmo. O que os engenheiros da Netflix buscavam eram pontos comuns desses projetos de ML. A Figura 12 ilustra bem essa necessidade e a busca por pontos em comum nos projetos.

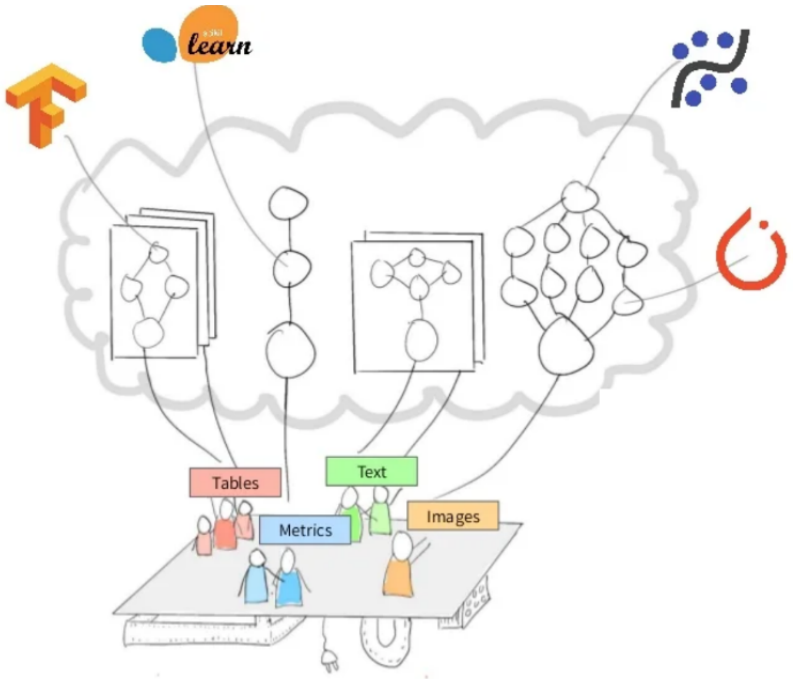

Figura 12 – As diferentes necessidades dos projetos de ML

Fonte: https://www.slideshare.net/BillLiu31/metaflow-the-ml-infrastructure-at-netflix

Que no caso da necessidade deles foi criar uma ferramenta que simplifica o processo de infraestrutura entre a prototipagem e o ambiente de produção, facilitando para o cientista de dados ter o seu modelo sendo executado e coletando dados, como diria Fowler: “entregando mais cedo, com frequência e falhando rápido... estar em produção e coletando métrica" (WOODWARD, 2018).

2.4.2 Desafios de adoção

Como dito na sessão anterior, diferentes projetos de ML possuem diferentes necessidades, o que dificulta a adoção de ferramentas de MLOps já que será preciso ter em mente as devidas diferenças de cada projeto e como elas afetam o processo de DevOps. No caso do Metaflow, a escolha dos engenheiros da Netflix foi simplificar o processo de DevOps para os cientistas de dados, mas a que custo ? Ter uma ferramenta que funciona muito bem quando atrelada à Amazon AWS, porém qualquer outra empresa que for usar do Metaflow também irá ficar presa aos serviços da AWS. Ainda é possível utilizar o Metaflow sem se atrelar a AWS mas boa parte das funcionalidades mais avançadas da ferramenta se perdem.

Outro problema identificado é com relação a documentação do Metaflow, embora seja excelente na parte prática mais voltada ao cientista de dados, a parte da instalação na infraestrutura da AWS é uma longa lista com os passos necessários, que em algumas etapas não representam bem a interface da AWS, deixando a pessoa que for realizar o procedimento de instalação confuso.

2.4.3 Beneficios de adoção

De longe o maior benefício no uso do Metaflow é a sua capacidade de abstração do processo de DevOps para o cientista de dados, dessa forma o cientista pode focar em melhorar o seu modelo com dados coletados de produção, o Metaflow ainda oferece outros recursos interessantes como branching e coleta de recursos direto do banco de dados na AWS.

O Metaflow é ótimo quando usado em projetos que necessitam processar grandes volumes de dados com bibliotecas de aprendizado de máquina como o caso da Netflix, em que seus cientistas de dados precisam manipular grandes quantidades de dados e uma máquina local seria inviável nesse caso. Mas com o auxílio da ferramenta, basta dar um comando em sua máquina local, que tudo é processado na nuvem e os resultados podem ser colhidos em sua máquina local como se tudo tivesse sido processado localmente.

Outro fator importante é ter o suporte de uma grande empresa por trás da ferramenta, dessa forma há uma garantia de que se terá atualizações de funcionalidades e principalmente de correção de bugs. Por outro lado, é importante ressaltar que quando

se tem o suporte de uma grande empresa em uma ferramenta, nem tudo são flores. Já que essa empresa tem as suas próprias necessidades e demandas quanto a ferramenta em questão.

Um exemplo fora do escopo de MLOps mas que abrange a questão da demanda empresa mantenedora tem sobre uma ferramenta, foi as controversas mudanças de licenças que o Facebook realizou sobre o ReactJs em 2014, atualmente o ReactJs está licenciado sobre a licença MIT mas originalmente foi lançado com a licença Apache V2 OSS, porém em 2014 o Facebook mudou a licença para a BSD mais algumas cláusulas extras que davam ao Facebook o copyright do ReactJs. Fazendo com que se alguma outra empresa desenvolvesse um produto competidor ao Facebook com o ReactJs, este poderia utilizar de seu copyright para atacar a outra empresa. Porém, devido ao forte crescimento no uso da biblioteca e a pressão da comunidade, o Facebook resolveu alterar novamente a licença, mas dessa vez para a MIT. Embora fora de MLOps, este é um bom exemplo dos riscos de se utilizar uma ferramenta mantida por uma grande empresa.

Ainda assim, uma das vantagens de quando há uma grande empresa por trás de uma ferramenta, é que essa empresa busca fomentar a adoção de sua ferramenta e com isso auxilia na adoção de soluções de MLOps por outras empresas, sendo um grande fator de propaganda e promoção de um ecossistema.

3 Considerações Finais

Quando iniciou-se o trabalho de pesquisa, constatou-se que há uma crescente demanda por implementar aprendizado de máquina em produtos de software, porém ainda mantendo todo o devido processo que se é esperado a nível de entrega contínua para essas aplicações. E com isso foi possível notar a pluralidade ferramental dessa área e o quão difícil pode ser a escolha de uma ferramenta para auxiliar no processo de MLOps. Embora nesse estudo tenha abordado a utilização de apenas uma, já é notável as dificuldades que a escolha de uma ferramenta pode ter para uma empresa.

Como pode ser visto pelo estudo, o processo de MLOps como um todo, é extenso e subdivido em várias etapas o que amplia ainda mais a dificuldade de sua implementação. Sendo assim, a pesquisa optou por focar em ferramentas com foco no gerenciamento do fluxo de trabalho e constatou-se que as alternativas dispostas oferecem grandes possibilidades, embora sua devida integração não seja algo trivial e que há um peso considerável na infraestrutura quando introduzida uma nova ferramenta.

Já com relação às ferramentas dispostas, foi possível constatar que a crescente demanda por aplicações de aprendizado de máquina resultou em um terreno fertil para o surgimento de várias alternativas open source que focam justamente nas mais diversas áreas do fluxo de MLOps. E ainda que essa demanda está até mesmo influenciando as grandes empresas a demandar esforços nessa área e até mesmo disponibilizar suas soluções como open source.

Outra característica observada pelo estudo foi que as ferramentas open source de MLOps são em sua grande maioria distribuídas sob a licença Apache 2.0 o que dada a sua natureza permissiva, facilita em sua adoção.

Diante disso, a pesquisa teve como objetivo geral, testar as dificuldades de integração de mais uma ferramenta em uma aplicação de aprendizado de máquina, em que essa ferramenta deveria auxiliar no processo de MLOps deste software em questão.

Consideramos então com os resultados obtidos pelo estudo que, embora há um preço a se pagar, como acoplamento a uma infraestrutura necessária para o uso de uma dada ferramenta por exemplo, quando as possibilidades abertas pela ferramenta escolhida acarretam em simplificar o trabalho do cientista de dados, profissional que está no coração das atividades de aprendizado de máquina, as dificuldades enfrentadas por essa integração valem sim o seu preço, desde que se esteja trabalhando em um software de grande porte.

Dadas as dificuldades encontradas na elaboração da parte prática do estudo fica observado que quando já se tem todo um fluxo de trabalho, a adoção de mais ferramentas

é um peso que pode não trazer bons resultados. No caso deste estudo em particular, o maior peso da adoção da ferramenta escolhida a Metaflow, foi o seu acoplamento com a infraestrutura da Amazon AWS, talvez uma outra ferramenta como a Mlrun teria sido uma escolha melhor, mas fica aqui a indicação de uma possível realização utilizando-se de outras ferramentas.

Ainda sobre a parte prática, há também a possibilidade do estudo sobre outros casos de uso, nesse estudo optou-se por utilizar um sistema de chatbot mas a área de aprendizado de máquina é bem vasta e com isso testar se a ferramenta auxilia no fluxo do cientista de dados em outras áreas ML seria uma ótima contribuição.

De forma geral, consideramos que os resultados obtidos por esse trabalho foram bastante proveitosos e de certa forma satisfatórios diante das lições aprendidas, foi possível notar as dificuldades enfrentadas por empresas na adoção de soluções de MLOps e reproduzir parte dessas dificuldades.

Referências

AMAZON, A. O que é o devops? 2021. Disponível em: <https://aws.amazon.com/devops/what-is-devops/>. Citado na página 19.

AMERSHI, S. Software engineering for machine learning: A case study. 2019. Disponível em: <https://www.microsoft.com/en-us/research/uploads/prod/2019/03/amershi-icse-2019_Software_Engineering_for_Machine_Learning.pdf>. Citado 2 vezes nas páginas 19 e 21.

BALSO, M. D. What is a feature store ? 2020. Disponível em: <https://www.tecton.ai/blog/what-is-a-feature-store>. Citado 2 vezes nas páginas 23 e 24.

BECK K., e. a. Princípios por trás do manifesto Ágil. 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/principles.html>. Citado na página 17.

CLOUD, G. Mlops: pipelines de entrega contínua e automação no aprendizado de máquina. 2021. Disponível em: <https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>. Citado 3 vezes nas páginas 15, 21 e 23.

CUNNINGHAM, W. Integration hell. 2012. Disponível em: <https://wiki.c2.com/?IntegrationHell>. Citado na página 17.

DOWLING, J. The feature store - jim dowling. 2019. Disponível em: <https://www.youtube.com/watch?v=EI2QisCvEM4>. Citado na página 23.

FOWLER, M. Continuous integration. 2006. Disponível em: <https://martinfowler.com/articles/continuousIntegration.html>. Citado na página 17.

HUMBLE, D. F. J. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010. ISBN 978-0-321-60191-9. Disponível em: <https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/>. Citado na página 18.

LEITE, L. et al. A survey of devops concepts and challenges. *ACM Compututer Survey*, 2019. Citado na página 19.

SCULLEY, D. et al. Hidden technical debt in machine learning systems. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 2503–2511. Citado na página 15.

TEMPORIM, A. Rasa boilerplate. 2021. Disponível em: <https://github.com/lappis-unb/rasa-ptbr-boilerplate>. Citado na página 35.

VISENGERIYEVA, L. et al. Mlops principles. 2021. Disponível em: <https://ml-ops.org/content/mlops-principles>. Citado na página 15.

WOODWARD, M. Why microsoft does devops. 2018. Disponível em: <https://www.youtube.com/watch?v=X4uziBlC728>. Citado 2 vezes nas páginas 19 e 41.