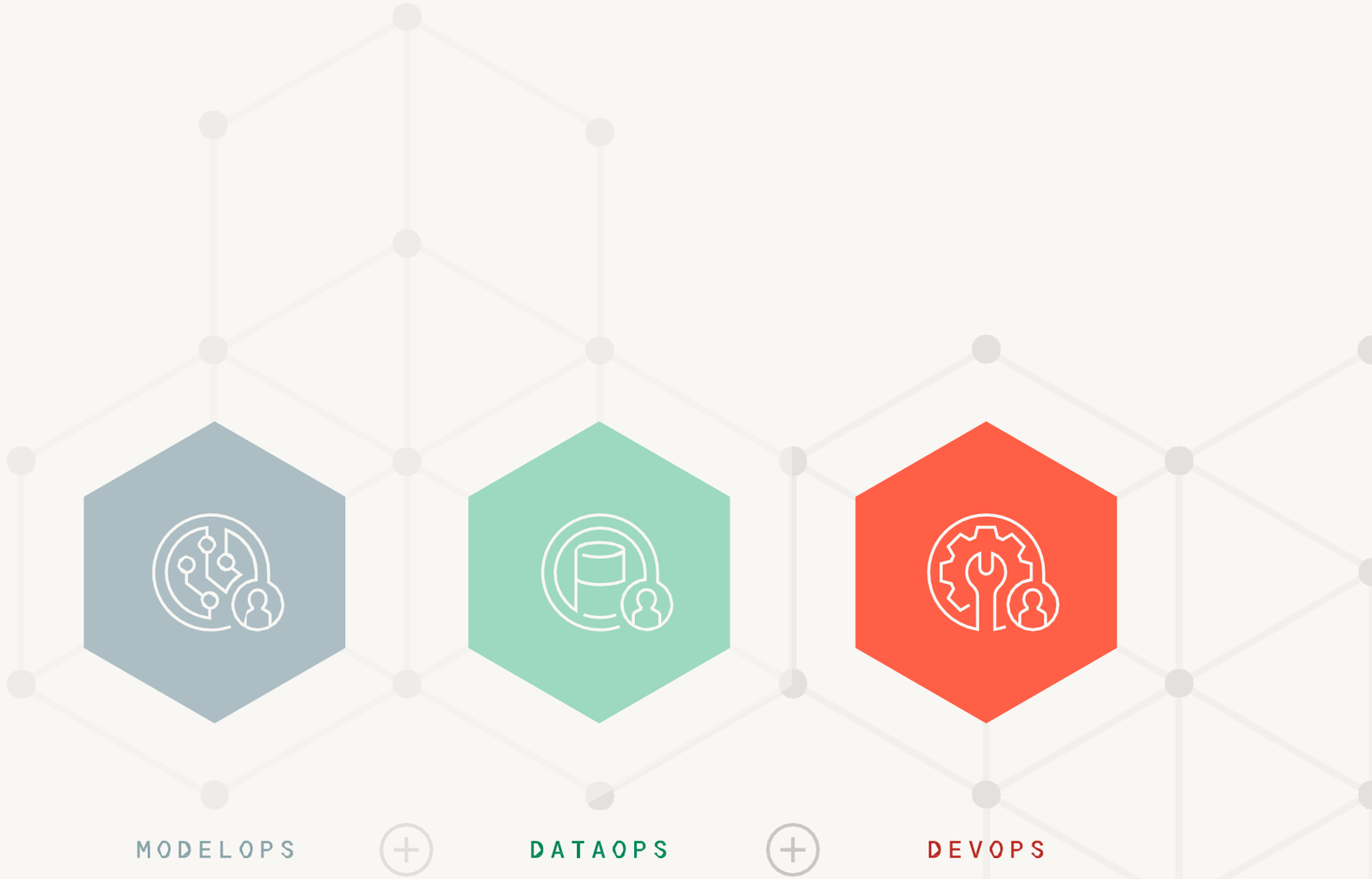
ebook

O Livro Completo de MLOps

Uma abordagem baseada em dados para construir e dimensionar IA, incluindo LLMOps



databricks

JOSEPH BRADLEY | RAFIKURLANSIK | MATTT THOMSON | NIALL TURBITT

Conteúdo

AUTORES :

Joseph Bradley
Especialista líder de produto

Rafi Kurlansik
Especialista líder de produto

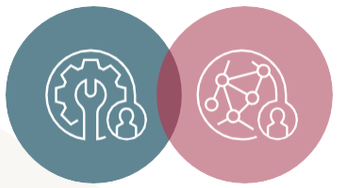
Matt Thomson
Diretor, especialistas em produtos EMEA

Niall Turbitt
Data scientist líder

databricks

CAPÍTULO 1:

Introdução



Observação: nossa indicação de MLOps é generalizada para qualquer conjunto de ferramentas e aplicações, embora demos exemplos concretos usando recursos e funcionalidades da Databricks. Também observamos que nem toda arquitetura ou indicação funcionará para todas as organizações ou casos de uso. Portanto, sempre que houver diretrizes para a construção de MLOps, destacaremos opções e variações importantes. Este whitepaper é escrito principalmente para engenheiros de ML e cientistas de dados que queiram saber mais sobre MLOps, com orientação de alto nível e indicações demais recursos.

Na última década, observou-se um rápido crescimento na adoção do machine learning (ML). Embora os primeiros usuários tenham sido um número pequeno de grandes empresas de tecnologia que tinham condições de pagar os recursos necessários, ultimamente, os casos de negócios orientados por ML estão presentes em todos os setores. De fato, de acordo com a MIT Sloan Management Review, 83% dos CEOs relataram que a inteligência artificial (IA) é uma prioridade estratégica.
Essa democratização do ML em todos os setores trouxe enormes benefícios econômicos, e a Gartner estima que US$ 3,9 trilhões em valor comercial serão criados pela IA em 2022.

No entanto, criar e implantar modelos de ML é complexo. Há muitas opções disponíveis, mas poucos padrões bem definidos e acessíveis. Como resultado, nos últimos anos, vimos o surgimento do campo de Machine Learning Operations (MLOps). **MLOps é um conjunto de processos e automação para gerenciar modelos, dados e código para melhorar a estabilidade de desempenho e a eficiência de longo prazo em sistemas de ML.** Simplificando, MLOps = ModelOps + DataOps + DevOps.

O conceito de Developer Operations (DevOps) não é novo. Ele já é usado há décadas para implantar aplicações de software, e a implantação de aplicações de ML tem muito a ganhar com isso. No entanto, práticas e ferramentas fortes de DevOps por si só são insuficientes, porque as aplicações de ML dependem de uma variedade de artefatos (como modelos, dados e código) que exigem tratamento especial. Qualquer solução de MLOps deve levar em conta as várias pessoas e processos que interagem com esses artefatos.

Aqui na Databricks, vimos em primeira mão como os clientes desenvolvem suas abordagens de MLOps, e algumas funcionam melhor que outras. Lançamos o projeto de código aberto MLflow para ajudar nossos clientes a obter sucesso com MLOps e, com mais de 10 milhões de downloads/mês do PyPI em maio de 2022, a adoção do MLflow confirma o desejo de operacionalizar modelos de ML.

Este whitepaper tem como objetivo explicar como sua organização pode criar práticas robustas de MLOps de forma incremental. Primeiro, descrevemos as pessoas e os processos envolvidos na implantação de aplicações de ML e a necessidade de rigor operacional. Também fornecemos princípios gerais para ajudar a orientar seu planejamento e tomada de decisões. Depois, passamos pelos fundamentos do MLOps, definindo termos e estratégias amplas para implementação. Finalmente, introduzimos uma arquitetura geral de referência de MLOps, os detalhes de seus processos e as práticas recomendadas.

Pessoas e processos

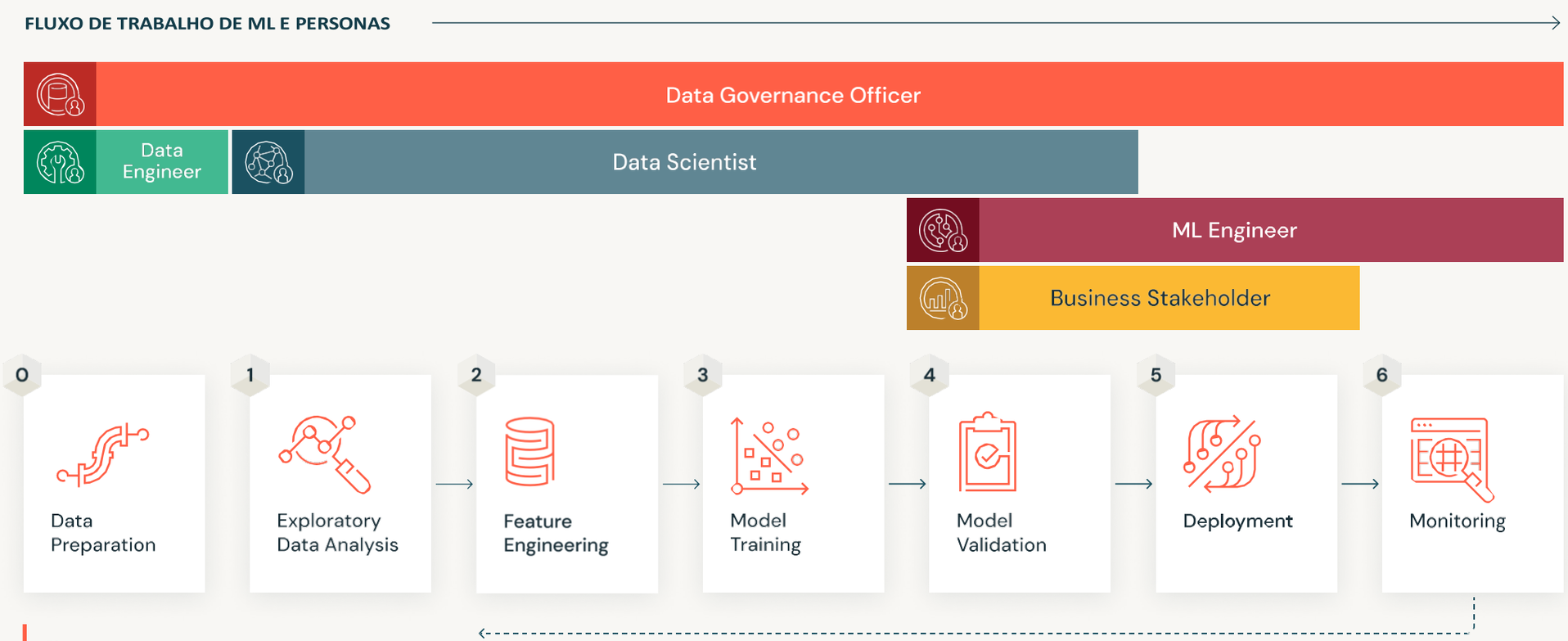


Figura 1

Pessoas

A criação de aplicações de ML é um esporte de equipe e, embora no mundo real as pessoas assumam várias funções ao mesmo tempo, ainda é útil pensar em termos de arquétipos. Eles nos ajudam a entender papéis e responsabilidades e onde as transferências são necessárias, e destacam áreas de complexidade dentro do sistema. Diferenciamos as seguintes personas:

PERSONAS DE ML

Engenheiro de dados

Responsável por criar pipelines de dados para processar, organizar e persistir conjuntos de dados para machine learning e outras aplicações downstream.

Cientista de Dados

Responsável por entender o problema de negócio, explorando os dados disponíveis para avaliar
se o machine learning é aplicável e, depois, treinar, ajustar e avaliar um modelo a ser implantado.

Engenheiro de ML

Responsável pela implantação de modelos de machine learning para produção com as práticas recomendadas apropriadas de governança, monitoramento e desenvolvimento de software, como integração e implementação contínuas (CI/CD).

Partes interessadas

Responsáveis por usar o modelo para tomar decisões para a empresa ou o produto, e também pelo valor de negócio que o modelo deve gerar.

Diretor de governança de dados

Responsável por garantir que a governança de dados, a privacidade dos dados e outras medidas de conformidade sejam adotadas em todo o processo de desenvolvimento e implantação do modelo. Não costuma se envolver nas operações do dia a dia.

Processo

Juntas, essas pessoas desenvolvem e mantêm aplicações de ML. Embora o processo de desenvolvimento siga um padrão distinto, ele não é inteiramente invariável. A maneira de implantar um modelo afeta as etapas a serem seguidas, e o uso de técnicas como aprendizado por reforço ou aprendizado online muda alguns detalhes.
No entanto, essas etapas e personas envolvidas são variações de um tema central, conforme ilustrado na Figura 1 acima.

Vamos analisar o processo passo a passo. Tenha em mente que este processo é iterativo, e sua frequência será determinada pelo caso de negócios e pelos dados específicos.

PROCESSO DE ML

Preparação dos dados

Antes de qualquer ciência de dados ou trabalho de ML está a engenharia de dados necessária para preparar os dados de produção e disponibilizá-los para consumo. Esses dados podem ser chamados de "dados brutos" e, em etapas posteriores, cientistas de dados extrairão recursos e rótulos dos dados brutos.

Análise exploratória de dados (EDA)

A análise é conduzida por cientistas de dados para avaliar as propriedades estatísticas dos dados disponíveis e determinar se eles abordam a questão de negócios. Isso requer comunicação e iteração frequentes com as partes interessadas da empresa.

Engenharia de recursos

Os cientistas de dados limpam os dados e aplicam lógica de negócios e transformações especializadas em recursos de engenharia para treinar o modelo. Esses dados, ou recursos, são divididos em conjuntos de treinamento, teste e validação.

Treinamento de modelos

Os cientistas de dados exploram vários algoritmos e configurações de hiperparâmetros usando os dados preparados, e um modelo de melhor desempenho é determinado de acordo com métricas de avaliação predefinidas.

Validação do modelo

Antes da implantação, um modelo selecionado está sujeito a uma etapa de validação para garantir que ele exceda um nível básico de desempenho, além de atender a quaisquer outros requisitos técnicos, comerciais ou regulamentares. Isso requer colaboração entre cientistas de dados, partes interessadas e engenheiros de ML.

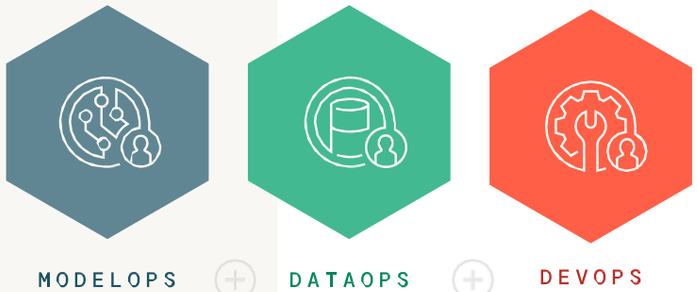
Implantação

Os engenheiros de ML implementam um modelo validado por meio de lote, streaming ou serviço online, dependendo dos requisitos do caso de uso.

Monitoramento

Os engenheiros de ML monitoram os modelos implantados em busca de sinais de degradação de desempenho ou erros. Os cientistas de dados geralmente estão envolvidos nas fases iniciais de monitoramento para garantir que os novos modelos tenham o desempenho esperado após a implantação. Isso confirma se e quando o modelo implantado deve ser atualizado retornando aos estágios anteriores do fluxo de trabalho.

O diretor de governança de dados é responsável por garantir que todo esse processo esteja em conformidade com as políticas regulatórias e corporativas.



Por que devo me preocupar com MLOps?

Considere que a aplicação típica de ML depende das pessoas e dos processos mencionados acima, bem como dos requisitos regulamentares e éticos. Essas dependências mudam ao longo do tempo — e seus modelos, dados e código também devem mudar. Os dados, que antes eram um sinal confiável, tornam-se ruídos; as bibliotecas de código aberto ficam desatualizadas; os ambientes regulatórios evoluem; e as equipes mudam. Os sistemas de ML devem ser resilientes a essas mudanças. No entanto, esse amplo escopo pode ser demais para as organizações gerenciarem, já que há muitas partes móveis! Enfrentar esses desafios com uma estratégia definida de MLOps pode reduzir drasticamente o ciclo de iteração da entrega de modelos à produção, acelerando assim o retorno sobre o investimento.

Há dois tipos principais de risco em sistemas de ML: **risco técnico** inerente ao próprio sistema e **risco de não conformidade** com sistemas externos. Ambos derivam das dependências descritas acima. Por exemplo, se faltarem infraestrutura de pipeline de dados, KPIs, monitoramento de modelos e documentação, você corre o risco de ter um sistema desestabilizado ou ineficaz. Por outro lado, um sistema bem projetado que não cumpra os requisitos corporativos, regulatórios e éticos corre o risco de perder financiamento, receber multas ou prejudicar a reputação. Recentemente, descobriu-se que as práticas de coleta de dados de uma empresa privada violaram a Lei de Proteção e Privacidade Online Infantil (COPPA). A FTC multou a empresa em US$ 1,5 milhão e ordenou a destruição ou exclusão dos dados coletados ilegalmente e de todos os modelos ou algoritmos desenvolvidos com esses dados.

Em relação à eficiência, a ausência de MLOps geralmente é marcada por um excesso de processos manuais. Essas etapas são mais lentas e mais propensas a erros, afetando a qualidade dos modelos, dados e códigos. Por fim, elas formam um gargalo, limitando a capacidade que uma equipe de dados tem de assumir novos projetos.

Visto por essa perspectiva, o objetivo do MLOps torna-se claro: melhorar a estabilidade do desempenho a longo prazo e a taxa de sucesso dos sistemas de ML, maximizando a eficiência das equipes que os desenvolvem. Na introdução, definimos MLOps para atender a esse objetivo: o MLOps é um **conjunto de processos e automação** para gerenciar **modelos, dados e código** e atender aos dois objetivos de **desempenho estável e eficiência de longo prazo em sistemas de ML**. *MLOps = ModelOps + DataOps + DevOps*.

Com metas claras, estamos prontos para discutir os princípios que orientam as decisões de design e planejamento para MLOps.

Princípios orientadores

Devido à complexidade dos processos de ML e às diferentes personas envolvidas, vale a pena começar com orientações mais simples e genéricas. Propomos vários princípios aplicáveis a diversas situações para orientar as decisões de MLOps. Eles informam nossas escolhas de design em seções posteriores, e esperamos que possam ser adaptados para dar suporte a qualquer caso de uso comercial.

Tenha sempre em mente suas metas de negócios

Assim como o objetivo principal do ML em uma empresa é permitir decisões e produtos orientados por dados, o objetivo principal do MLOps é garantir que essas aplicações orientadas por dados permaneçam estáveis, atualizadas e continuem a ter impactos positivos nos negócios. Ao priorizar o trabalho técnico em MLOps, considere o impacto nos negócios: ele permite novos casos de uso de negócios? Melhora a produtividade das equipes de dados? Reduz custos ou riscos operacionais?

Adote uma abordagem baseada em dados para machine learning

Engenharia de recursos, treinamento, inferência e pipelines de monitoramento são pipelines de dados. Por isso, eles precisam ser tão robustos quanto outros processos de engenharia de dados de produção. Uma vez que a qualidade dos dados é crucial em qualquer aplicação de ML, os pipelines de dados de ML devem empregar abordagens sistemáticas para monitorar e mitigar problemas de qualidade dos dados.

Evite ferramentas que dificultem a união de dados de previsões de ML, monitoramento de modelo etc. com o restante dos seus dados. A maneira mais simples de conseguir isso é desenvolver aplicações de ML na mesma plataforma usada para gerenciar os dados de produção. Por exemplo, em vez de baixar dados de treinamento para um notebook, onde é difícil controlar e reproduzir resultados, armazene os dados na nuvem e disponibilize esse armazenamento para seu processo de treinamento.

Implemente o MLOps de forma modular

Tal como acontece com qualquer aplicação de software, a qualidade do código é fundamental para uma aplicação ML. O código modularizado permite o teste de componentes individuais e reduz as dificuldades com a refatoração de código futura. Defina etapas claras (por exemplo, treinamento, avaliação ou implantação), superetapas (como pipeline de treinamento para implantação) e responsabilidades para esclarecer a estrutura modular da sua aplicação de ML.

O processo deve guiar a automação

Automatizamos processos para melhorar a produtividade e reduzir o risco de erros humanos, mas nem todas as etapas de um processo podem ou devem ser automatizadas. As pessoas ainda determinam a questão de negócios, e alguns modelos sempre precisarão de supervisão humana antes da implementação. Portanto, o processo de desenvolvimento é primário e cada módulo no processo deve ser automatizado conforme necessário. Isso permite a criação incremental de automação e personalização. Além disso, quando se trata de ferramentas de automação específicas, escolha as que se alinham ao seu pessoal e processo. Por exemplo, em vez de construir uma estrutura de registro de modelo em torno de um banco de dados genérico, você pode escolher uma ferramenta especializada como o MLflow, que foi projetada pensando no ciclo de vida do modelo ML.

CAPÍTULO 2:

Fundamentos do MLOps

Observação: em nossa experiência com clientes, pode haver variações nestes três estágios, como dividir o estágio em subestágios separados de “teste” e “controle de qualidade. No entanto, os princípios permanecem os mesmos e seguimos uma configuração de desenvolvimento, preparação e produção neste artigo.

Semântica de desenvolvimento, preparação e produção

Os fluxos de trabalho de ML incluem os seguintes ativos principais: código, modelos e dados. Esses ativos precisam ser desenvolvidos (desenvolvimento), testados (preparação) e implantados (produção). Para cada etapa, também precisamos operar em um ambiente de execução. Assim, todos os ambientes de execução, código, modelos e dados acima são divididos em desenvolvimento, preparação e produção.

Essas divisões podem ser mais bem compreendidas em termos de garantias de qualidade e controle de acesso. De um lado, os ativos em produção costumam ser críticos para os negócios, com a mais alta garantia de qualidade e controle mais rígido sobre quem pode modificá-los. Por outro, os ativos de desenvolvimento são mais acessíveis às pessoas, mas não oferecem garantia de qualidade.

Por exemplo, muitos data scientists trabalham juntos em um ambiente de desenvolvimento, produzindo livremente protótipos de modelos de desenvolvimento. Qualquer falha nesses modelos representa um risco relativamente baixo para a empresa, pois eles estão separados do produto real. Em contrapartida, o ambiente de preparação replica o ambiente de execução da produção. Aqui, as alterações de código feitas no ambiente de desenvolvimento são testadas antes de implantar o código na produção. O ambiente de preparação funciona como uma porta de entrada para o código chegar à produção e, portanto, menos pessoas têm acesso à preparação. O código promovido à produção é considerado um produto ativo. No ambiente de produção, o erro humano pode representar o maior risco para a continuidade dos negócios e, portanto, o menor número possível de pessoas tem permissão para modificar os modelos de produção.

Seria possível dizer que código, modelos e dados compartilham uma correspondência individual com o ambiente de execução — por exemplo, todos os códigos de desenvolvimento, modelos e dados estão no ambiente de desenvolvimento. Isso está bem próximo da verdade, mas raramente está correto. Portanto, discutiremos a seguir a semântica precisa de desenvolvimento, preparação e produção para ambientes de execução, código, modelos e dados. Também veremos os mecanismos para restringir o acesso a eles.

Ambientes de execução

Um ambiente de execução é o local onde modelos e dados são criados ou consumidos por código. Cada ambiente de execução consiste em instâncias de compute, tempos de execução e bibliotecas e trabalhos automatizados. Com a Databricks, um "ambiente" pode ser definido por meio da separação de desenvolvimento/preparação/produção em alguns níveis. Uma organização pode criar ambientes distintos em várias contas de nuvem, diferentes workspaces da Databricks na mesma conta de nuvem ou em um único workspace da Databricks. Esses padrões de separação são ilustrados na Figura 2 abaixo.

PADRÕES DE SEPARAÇÃO DE AMBIENTE

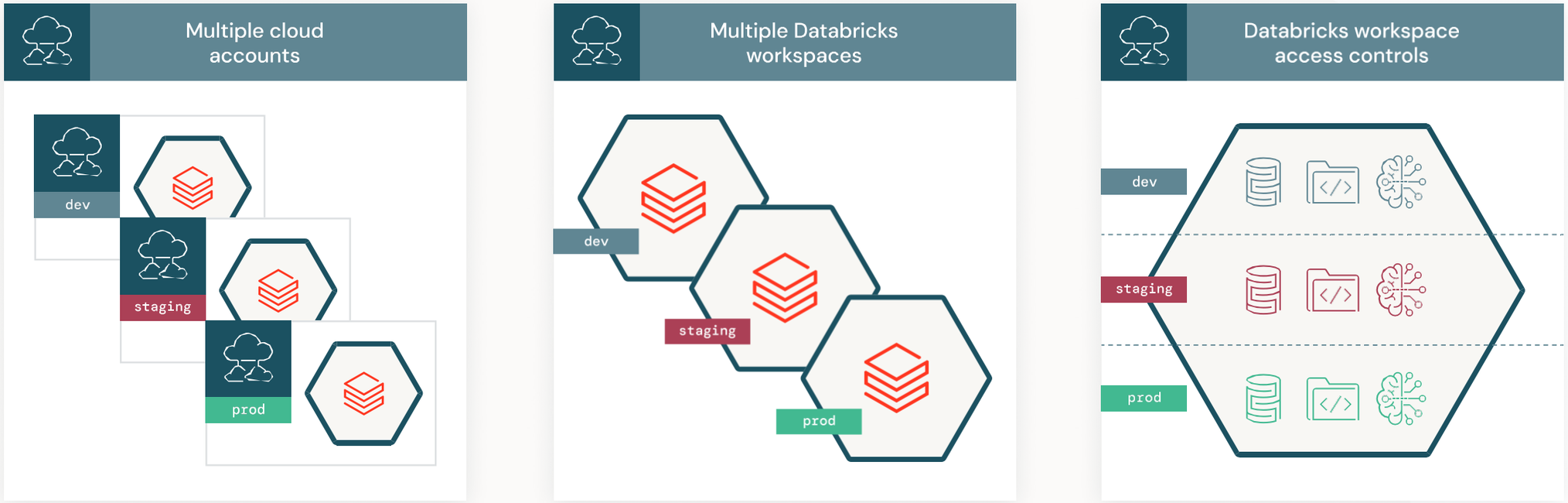


Figura 2

A Databricks lançou o Delta Lake para a comunidade de código aberto em 2019. O Delta Lake oferece todas as funções de gerenciamento do ciclo de vida de dados necessárias
para tornar os armazenamentos de objetos baseados em nuvem confiáveis e eficientes. Esse design permite que os clientes atualizem vários objetos de uma só vez e substituam um subconjunto dos objetos por outro de maneira serializável que ainda consegue obter alto desempenho paralelo de leitura/gravação a partir dos objetos, ao mesmo tempo que oferece recursos avançados como dados históricos (por exemplo, snapshots de query de ponto no tempo ou reversão de atualizações incorretas), otimização automática de layout de dados, upserts, cache e logs de auditoria.

Código

O código de projeto de ML normalmente é armazenado em um repositório de controle de versão (como Git), e a maioria das organizações usam ramificações correspondentes às fases do ciclo de vida de desenvolvimento, preparação ou produção. Existem alguns padrões comuns. Uns utilizam apenas branches de desenvolvimento (desenvolvimento) e um ramo principal (preparação/produção).
Outros usam branches principais e de desenvolvimento (desenvolvimento), branches cortadas para testar possíveis versões (preparação) e branches cortadas para versões finais (produção). Independentemente da convenção escolhida, a separação é imposta por meio de branches do repositório Git.

Como prática recomendada, o código só deve ser executado em um ambiente de execução que corresponda a ele ou em um que seja maior. Por exemplo, o ambiente de desenvolvimento pode executar qualquer código, mas o ambiente de produção só pode executar código de produção.

Modelos

Embora os modelos sejam geralmente marcados como desenvolvimento, preparação ou produção de acordo com a fase do ciclo de vida, **é importante observar que as fases do ciclo de vida do modelo e do código normalmente operam de forma assíncrona**. Ou seja, você pode querer enviar uma nova versão do modelo antes de enviar uma alteração de código, e vice-versa. Considere os seguintes cenários:

- Para detectar transações fraudulentas, você desenvolve um pipeline de ML que retém um modelo semanalmente. A implantação do código pode ser um processo relativamente pouco frequente; mas, a cada semana, um novo modelo passa por seu próprio ciclo de vida de ser gerado, testado e marcado como "produção" para prever as transações mais recentes. Nesse caso, o ciclo de vida do código é mais lento do que o ciclo de vida do modelo.

- Para classificar documentos usando grandes redes neurais profundas, o treinamento e a implantação do modelo costumam ser um processo único devido ao custo. As atualizações no código de atendimento e monitoramento no projeto podem ser implantadas com mais frequência do que uma nova versão do modelo. Nesse caso, o ciclo de vida do modelo é mais lento do que o código.

Como os ciclos de vida do modelo não correspondem exatamente aos ciclos de vida do código, faz sentido que o gerenciamento do modelo tenha seu próprio serviço. O MLflow e seu suporte ao registro do modelo gerenciam artefatos do modelo diretamente pela UI e por APIs. A combinação solta de artefatos do modelo e código fornece flexibilidade para atualizar modelos de produção sem alterar o código, simplificando o processo de implantação em muitos casos. Os artefatos do modelo são protegidos usando controles de acesso do MLflow ou permissões de armazenamento em nuvem.

Dados

Algumas organizações rotulam os dados como desenvolvimento, preparação ou produção, dependendo do ambiente em que se originaram. Por exemplo, todos os dados de produção são produzidos no ambiente de produção, mas ambientes de desenvolvimento e preparação podem ter acesso somente leitura a eles. Marcar dados dessa forma também indica uma garantia da qualidade dos dados: os dados de desenvolvimento podem ser temporários ou não destinados a um uso mais amplo, enquanto os dados de produção podem oferecer garantias mais fortes em termos de confiabilidade e atualização. O acesso aos dados em cada ambiente é controlado com controles de acesso à tabela

(AWS | Azure | GCP) ou permissões de armazenamento em nuvem.

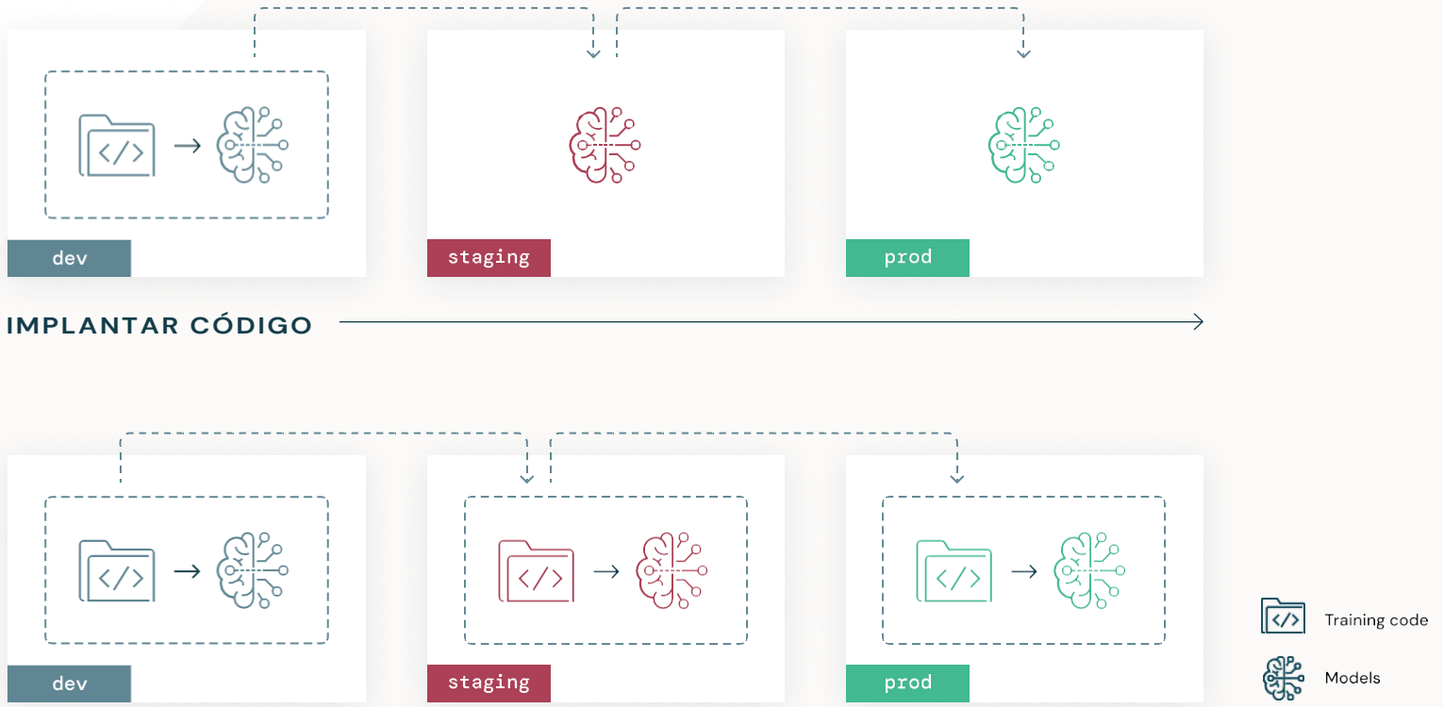
Em resumo, quando se trata de MLOps, sempre haverá separação operacional entre desenvolvimento, preparação e produção. Os ativos em desenvolvimento terão os controles de acesso e as garantias de qualidade menos restritivos, enquanto os ativos em produção terão a mais alta qualidade e controle rigoroso.

ATIVO	SEMÂNTICA	SEPARADO POR
Ambientes de execução	Rotulados de acordo com o local onde ocorre o desenvolvimento, os testes e as conexões com os sistemas de produção	Controles de acesso ao provedor de nuvem e ao Databricks Workspace
Modelos	Rotulado de acordo com a fase do ciclo de vida do modelo	Controles de acesso ao MLflow ou permissões de armazenamento em nuvem
Dados	Rotulados de acordo com sua origem em ambientes de desenvolvimento, preparação ou execução de produção	Controles de acesso à tabela ou permissões de armazenamento na nuvem
Código	Rotulado de acordo com a fase do ciclo de vida do desenvolvimento de software	Branches do repositório Git

Tabela 1

Padrões de implantação de ML

O fato de que modelos e códigos podem ser gerenciados separadamente resulta em vários padrões possíveis para obter artefatos de ML por meio de preparação e produção. Explicamos dois dos principais padrões abaixo.



Esses dois padrões diferem dependendo se o artefato do modelo ou o código de treinamento que produz o artefato do modelo é promovido para a produção.

Implantar modelos

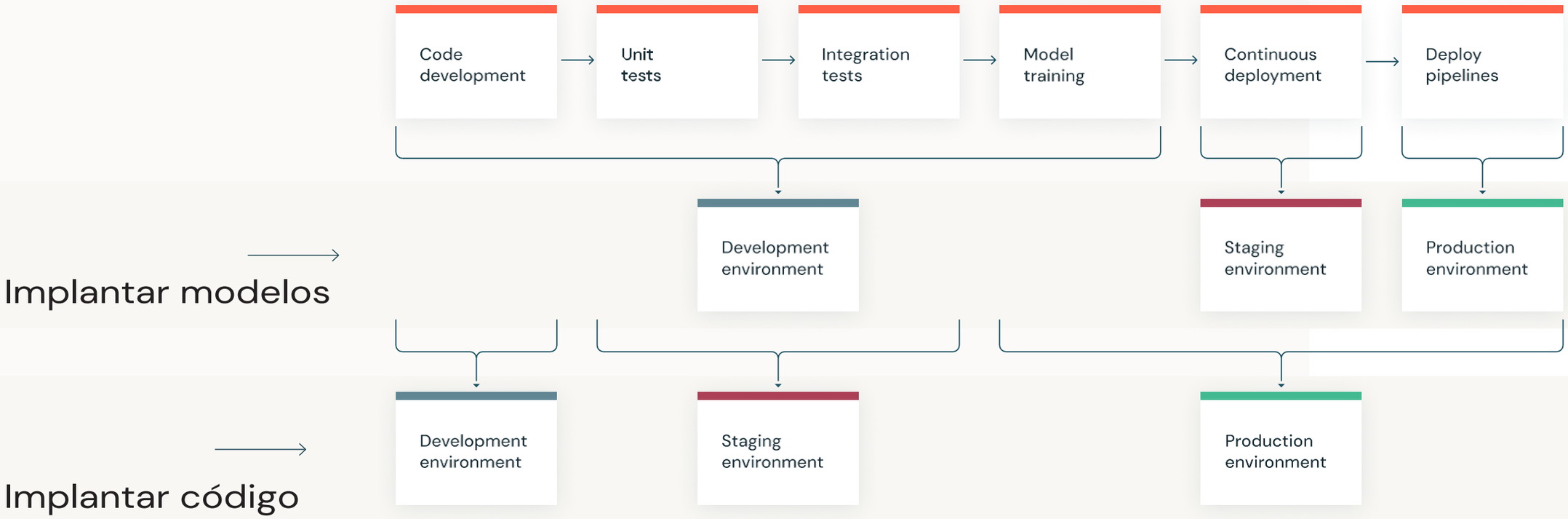
No primeiro padrão, o artefato do modelo é gerado pelo código de treinamento no ambiente de desenvolvimento.

Esse artefato é, então, testado na preparação para conformidade e desempenho antes de ser finalmente implementado na produção. Essa é uma entrega mais simples para data scientists e, em casos em que o treinamento de modelo é proibitivamente caro, pode ser mais eficiente treinar o modelo uma vez e gerenciar esse artefato. No entanto, essa arquitetura mais simples vem com limitações. Se os dados de produção não estiverem acessíveis a partir do ambiente de desenvolvimento (por exemplo, por motivos de segurança), essa arquitetura pode não ser viável. Essa arquitetura não oferece suporte natural ao retreinamento automatizado de modelos. Embora seja possível automatizar o retreinamento no ambiente de desenvolvimento, você trataria o código de treinamento de "desenvolvimento" como pronto para produção, o que muitas equipes de implementação não aceitariam. Essa opção oculta o fato de que o código auxiliar para qualificação, inferência e monitoramento precisa ser implantado na produção, exigindo um caminho de implantação de código separado.

Implantar código

No segundo padrão, o código para treinar modelos é desenvolvido no ambiente de desenvolvimento e esse código é movido para teste e, depois, para produção. Os modelos são treinados em cada ambiente: inicialmente, no ambiente de desenvolvimento como parte do desenvolvimento do modelo; na preparação (em um subconjunto limitado de dados) como parte de testes de integração; e, finalmente, no ambiente de produção (nos dados completos de produção) para produzir o modelo definitivo. Se uma organização restringir o acesso dos data scientists aos dados de produção de ambientes de desenvolvimento ou de preparação, a implantação do código permitirá o treinamento em dados de produção, respeitando os controles de acesso. Como o código de treinamento passa por revisão e teste de código, é mais seguro configurar o retreinamento automatizado. O código auxiliar segue o mesmo padrão do código de treinamento do modelo, e ambos podem passar por testes de integração na preparação. No entanto, a curva de aprendizagem para entregar o código aos colaboradores pode ser maior para muitos data scientists, de modo que modelos de projetos e fluxos de trabalho opinativos são úteis. Por fim, os data scientists precisam ter visibilidade dos resultados do treinamento no ambiente de produção, pois somente eles têm o conhecimento necessário para identificar e corrigir problemas específicos de ML.

O diagrama abaixo compara o ciclo de vida do código com os padrões de implantação acima nos diferentes ambientes de execução.



Em geral, recomendamos seguir a abordagem "implantar código" e a arquitetura de referência deste documento está alinhada a ele. No entanto, não há um processo perfeito que abranja todos os cenários, e as opções descritas acima não são mutuamente exclusivas. Dentro da mesma organização, você pode encontrar alguns casos de uso que implantam código de treinamento e outros que implantam artefatos de modelo. A escolha do processo depende do caso de uso de negócios, dos recursos disponíveis e do que é mais provável que seja bem-sucedido.

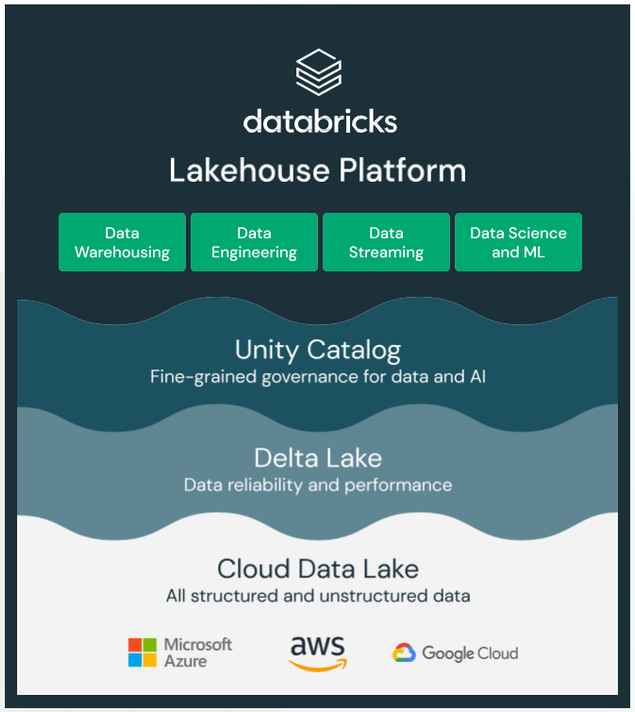
		IMPLANTAR MODELOS	IMPLANTAR CÓDIGO
Processo	Desenvolvimento	Desenvolver código de treinamento. Desenvolver código auxiliar.[1] Treinar modelo em dados de produção. ☐ Promover modelo e código auxiliar.	Desenvolver código de treinamento. Desenvolver código auxiliar. ☐ Promover código.
	Preparação	Testar modelo e código auxiliar. ☐ Promover modelo e código auxiliar.	Treinar modelo no subconjunto de dados. Testar código auxiliar. ☐ Promover código.
	Produção	Implantar modelo. Implantar pipelines auxiliares.	Treinar modelo com base em dados de produção. Testar modelo. Implantar modelo. Implantar pipelines auxiliares.
Vantagens e desvantagens	Automação	☐ Não aceita retreinamento automatizado em ambiente bloqueado.	☐ Aceita retreinamento automatizado em ambiente bloqueado.
	Controle do acesso aos dados	☐ O ambiente de desenvolvimento precisa de acesso de leitura aos dados de treinamento em produção.	☐ Somente o ambiente de produção precisa ter acesso de leitura aos dados de treinamento em produção.
	Modelos reproduzíveis	☐ Menos controle de engenharia sobre o ambiente de treinamento, muito mais difícil de garantir a reprodutibilidade.	☐ Controle de engenharia sobre o ambiente de treinamento, o que ajuda a simplificar a reprodutibilidade.
	Familiaridade de data science	☐ A equipe de DS cria e pode testar modelos diretamente no ambiente de desenvolvimento.	☐ A equipe de DS deve aprender a escrever e entregar código modular para a engenharia.
	Compatibilidade com grandes projetos	☐ Este padrão não força a equipe de DS a usar código modular para treinar o modelo e tem menos testes iterativos.	☐ Este padrão força a equipe de DS a usar código modular e testes iterativos, o que ajuda na coordenação e no desenvolvimento de projetos maiores.
	Configuração e manutenção de engenharia	☐ Tem a configuração mais simples, requer menos infraestrutura de CI/CD.	☐ Requer infraestrutura de CI/CD para testes unitários e de integração, mesmo para modelos únicos.
Quando usar		Use este padrão quando seu modelo for único ou quando for muito caro treinar o modelo. Use quando desenvolvimento, preparação e produção não forem ambientes estritamente separados.	**Use este padrão por default.** Use quando desenvolvimento, preparação e produção são ambientes estritamente separados.

Tabela 2

[1] "Código auxiliar" refere-se ao código para pipelines de ML diferentes do pipeline de treinamento do modelo. O código auxiliar pode ser preparação, inferência, monitoramento ou outros pipelines.

CAPÍTULO 3:

Arquitetura e processo de MLOps



Componentes da arquitetura

Antes de desinstalar a arquitetura de referência, familiarize-se com os recursos da Databricks usados para facilitar o MLOps no fluxo de trabalho indicado.

Data Lakehouse

Uma arquitetura de Data Lakehouse une os melhores elementos de data lakes e data warehouses, oferecendo gerenciamento e desempenho de dados normalmente encontrados em data warehouses com os armazenamentos de objetos flexíveis e de baixo custo oferecidos pelos data lakes. Os dados no lakehouse normalmente são organizados usando uma arquitetura "medalhão" de tabelas Bronze, Prata e Ouro de aprimoramento e qualidade crescentes.

MLflow

O MLflow é um projeto de código aberto para gerenciar o ciclo de vida completo do machine learning. Ele tem os seguintes componentes principais:

- **Acompanhamento:** permite acompanhar experimentos para registrar e comparar parâmetros, métricas e artefatos do modelo. Consulte a documentação do AWS | Azure | GCP.
- **Modelos ("variantes do MLflow"):** permite armazenar e implantar modelos de qualquer biblioteca de ML em diversas plataformas de disponibilização de modelos e inferência. Consulte a documentação do AWS | Azure | GCP.
- **Registro de modelos:** oferece uma loja de modelos centralizada para gerenciar as transições de fases do ciclo de vida completo dos modelos: da preparação à produção, com recursos para controle de versões e anotações. O registro também fornece webhooks para automação e implantação contínua. Consulte a documentação do AWS | Azure | GCP.

A Databricks também fornece uma versão totalmente gerenciada e hospedada do MLflow com recursos de segurança corporativa, alta disponibilidade e outros recursos do workspace da Databricks, como gerenciamento de experimentos e execuções e captura de revisões de notebooks. O MLflow na Databricks oferece uma experiência integrada para acompanhar e proteger execuções de treinamento de modelos de machine learning e executar projetos de machine learning.

Registro automático da Databricks e do MLflow

O Databricks Autologging é uma solução sem código que amplia o registro automático do MLflow para fornecer acompanhamento automático do experimento para sessões de treinamento na Databricks. O Databricks Autologging captura automaticamente parâmetros, métricas, arquivos e informações de linhagem do modelo ao treinar modelos com execuções de treinamento registradas como execuções de acompanhamento do MLflow. Consulte a documentação do AWS | Azure | GCP.

Feature Store

A Databricks Feature Store é um repositório centralizado de recursos. Ela permite o compartilhamento e a descoberta de recursos em uma organização e também garante que o mesmo código de compute de recurso seja usado para treinamento e inferência do modelo. Consulte a documentação do AWS | Azure | GCP.

Disponibilização de modelos do MLflow

A disponibilização de modelos do MLflow permite hospedar modelos de machine learning do Model Registry como endpoints REST que são atualizados automaticamente com base na disponibilidade das versões do modelo e seus estágios. Consulte a documentação do AWS | Azure | GCP.

Databricks SQL

O Databricks SQL oferece uma experiência simples para os usuários do SQL que desejam executar queries ad hoc rápidas em seu data lake, criar vários tipos de visualização para explorar os resultados de queries de diferentes perspectivas e construir e compartilhar painéis. Consulte a documentação do AWS | Azure | GCP.

Databricks Workflows e jobs

Os fluxos de trabalho da Databricks (Jobs e Delta Live Tables) podem executar pipelines de maneiras automatizadas e não interativas. Para ML, os jobs podem ser usados para definir pipelines para recursos de compute, modelos de treinamento ou outros tipos de passos ou pipelines de ML. Consulte a documentação do AWS | Azure | GCP.

Arquitetura de referência

Agora, estamos prontos para analisar uma arquitetura de referência geral para implementar MLOps na plataforma Databricks Lakehouse usando o padrão "implantar código" recomendado anteriormente. O objetivo é abranger a maioria dos casos de uso e técnicas de ML, mas não é de forma alguma exaustivo. Quando for o caso, destacaremos abordagens alternativas para implementar diferentes partes do processo.

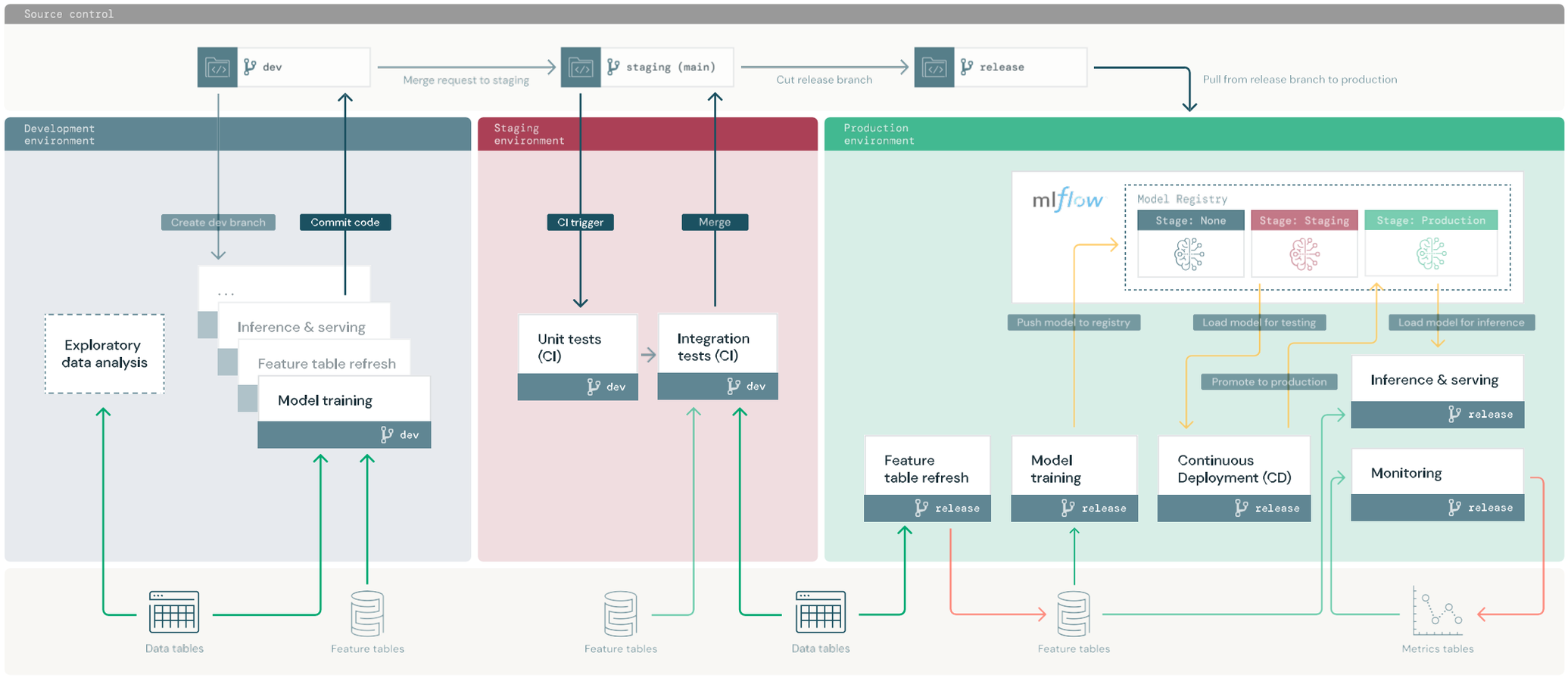
Começamos com uma visão geral do sistema de ponta a ponta, seguida por visões mais detalhadas do processo em ambientes de desenvolvimento, preparação e produção. Estes diagramas mostram o sistema operando em estado estacionário, e os detalhes mais sutis dos ciclos de desenvolvimento iterativos foram omitidos. Esta estrutura está resumida a seguir.

VISÃO GERAL

Desenvolvimento	preparação	Produção
▪ Dados ▪ Análise exploratória de dados ▪ (EDA) Código do projeto ▪ Atualização da ▪ tabela de recursos ▪ Treinamento do modelo Código de commit	▪ Solicitação de ▪ merge Testes ▪ unitários (CI) ▪ Testes de integração ▪ (CI) Merge Corte do branch de liberação	▪ Atualização da ▪ tabela de recursos ▪ Treinamento do ▪ modelo ▪ Implantação contínua (CD) ▪ Disponibilização online (APIs ▪ REST) Inferência: batch ou streaming Monitoramento

Visão geral

Figura 3



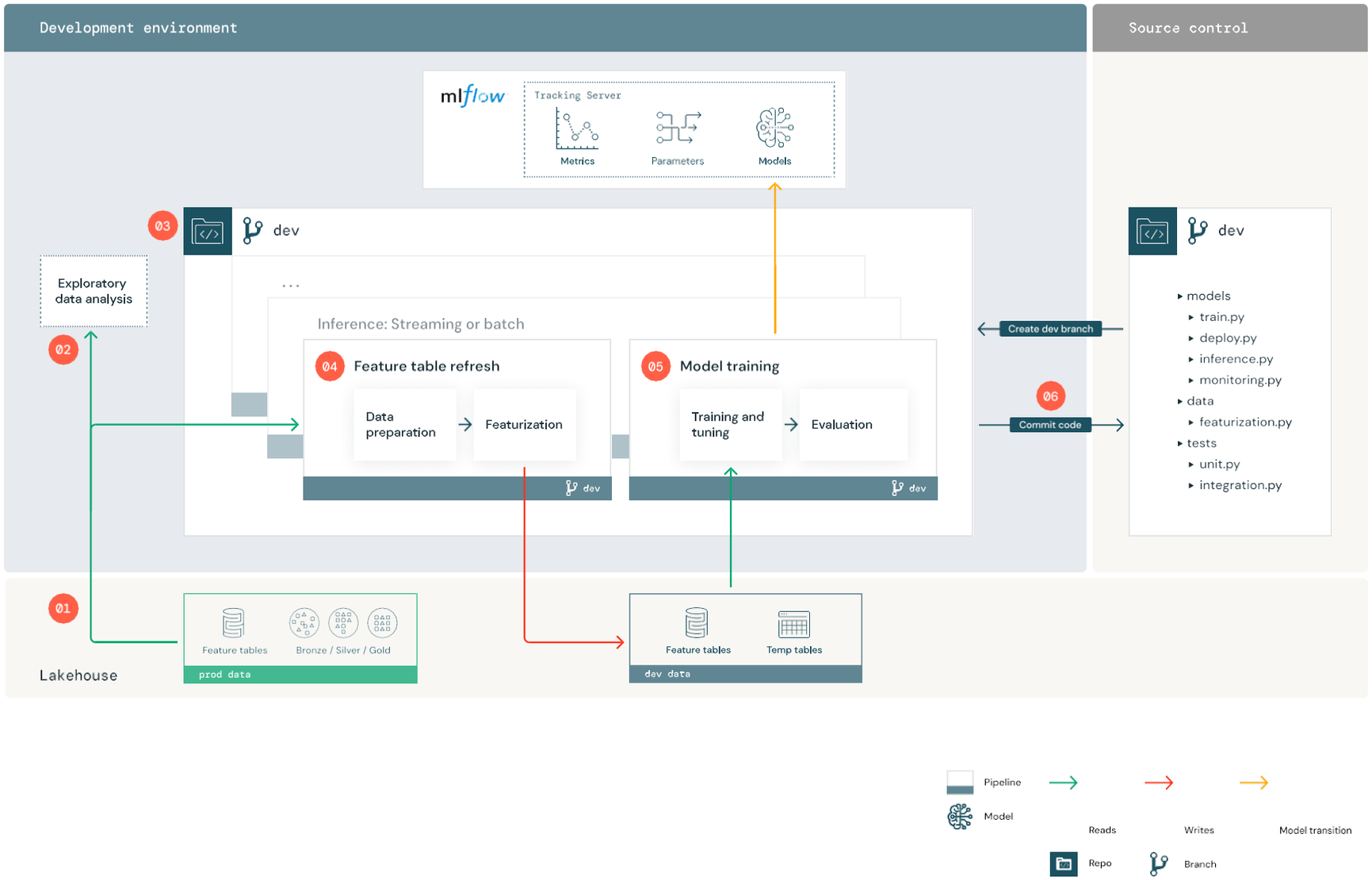
Aqui, vemos o processo geral de implantação de artefatos de código e modelo, entradas e saídas para pipelines e estágios do ciclo de vida do modelo em produção. O controle do código-fonte é a principal conduta para implantar pipelines de ML do desenvolvimento à produção. Os pipelines e os modelos são prototipados em um branch de desenvolvimento no ambiente de desenvolvimento, e é feito o commit das alterações na base de código para o controle do código-fonte. Mediante solicitação de merge para o branch de preparação (geralmente o branch "principal"), um processo de integração contínua (CI) testa o código no ambiente de preparação. Se os testes forem aprovados, um novo código poderá ser implantado na produção cortando uma versão do código. Na produção, um modelo é treinado nos dados completos de produção e enviado para o MLflow Model Registry. Um processo de implantação contínua (CD) testa o modelo e o promove em direção ao estágio de produção no registro. O modelo de produção do Model Registry pode ser disponibilizado por batch, streaming ou REST API. A engenharia contínua de recursos e os pipelines de monitoramento também são executados em produção.

databricks

Desenvolvimento

No ambiente de desenvolvimento, cientistas de dados e engenheiros de ML podem colaborar em todos os pipelines em um projeto de ML, fazendo o commit de suas alterações no controle do código-fonte. Embora os engenheiros possam ajudar a configurar esse ambiente, os cientistas de dados normalmente têm controle significativo sobre as bibliotecas, recursos de compute e código que usam.

Figura 4



Dados

Os cientistas de dados que trabalham no ambiente de desenvolvimento têm acesso somente leitura aos dados de produção. Eles também exigem acesso de leitura e escrita a um ambiente de armazenamento de desenvolvimento separado para desenvolver e experimentar com novos recursos e outras tabelas de dados.

Análise exploratória de dados (EDA)

O cientista de dados explora e analisa os dados em um processo interativo e iterativo. Esse processo é usado para avaliar se os dados disponíveis têm o potencial de resolver o problema do negócio. A EDA também é onde o cientista de dados começa a discernir qual preparação de dados e viabilidade são necessárias para treinar o modelo.

Este processo ad hoc geralmente não faz parte de um pipeline que será implantado em outros ambientes de execução.

Código do projeto

Este é um repositório de código que contém todos os pipelines ou módulos envolvidos no sistema ML. Os branches de desenvolvimento são usados para desenvolver alterações nos pipelines existentes ou para criar novos. Mesmo durante a EDA e as fases iniciais de um projeto, recomenda-se desenvolver dentro de um repositório para ajudar no acompanhamento de alterações e no compartilhamento de código.

Atualização da tabela de recursos

Este pipeline lê tabelas de dados brutos e tabelas de recursos e grava em tabelas na Feature Store. O pipeline consiste em duas etapas:

- **Preparação dos dados**

  Esta etapa verifica e corrige quaisquer problemas de qualidade de dados antes da preparação.

- **Preparação**

  No ambiente de desenvolvimento, novos recursos e a lógica de preparação atualizada podem ser testados gravando em tabelas de recursos no armazenamento de desenvolvimento, e essas tabelas podem ser usadas para prototipagem de modelos. Depois que esse código de preparação for promovido à produção, essas alterações afetarão as tabelas dos recursos de produção.

  Os recursos já disponíveis nas tabelas dos recursos de produção podem ser lidos diretamente para desenvolvimento.

Em algumas organizações, os pipelines de engenharia de recursos são gerenciados separadamente dos projetos de ML. Nesses casos, o pipeline de preparação pode ser omitido dessa arquitetura.

Treinamento de modelos

Os cientistas de dados desenvolvem o pipeline de treinamento do modelo no ambiente de desenvolvimento com tabelas de recursos de desenvolvimento ou produção.

- **Treinamento e ajuste**

  O processo de treinamento lê os recursos do armazenamento de recursos e/ou das tabelas de Lakehouse de nível Prata ou Ouro e registra os parâmetros, as métricas e os artefatos do modelo no servidor de acompanhamento do MLflow. Após o treinamento e o ajuste do hiperparâmetro, o artefato final do modelo é registrado no servidor de acompanhamento para registrar um vínculo robusto entre o modelo, seus dados de entrada e o código usado para gerá-lo.

- **Avaliação**

  A qualidade do modelo é avaliada por meio de testes em dados retidos. Os resultados desses testes são registrados no servidor de acompanhamento do MLflow.

  Se a governança exigir métricas adicionais ou documentação complementar sobre o modelo, esse é o momento de adicioná-las usando o acompanhamento do MLflow. Interpretações do modelo (por exemplo, gráficos produzidos por SHAP ou LIME) e descrições de texto simples são comuns, mas definir as especificações para essa governança exige a opinião de partes interessadas ou de um diretor de governança de dados.

- **Saída do modelo**

  A saída desse pipeline é um artefato do modelo de ML armazenado no servidor de acompanhamento do MLflow. Quando esse pipeline de treinamento é executado em preparação ou produção, os engenheiros de ML (ou seu código CI/CD) podem carregar o modelo por meio do URI (ou caminho) do modelo e, em seguida, enviar o modelo ao Model Registry para gerenciamento e teste.

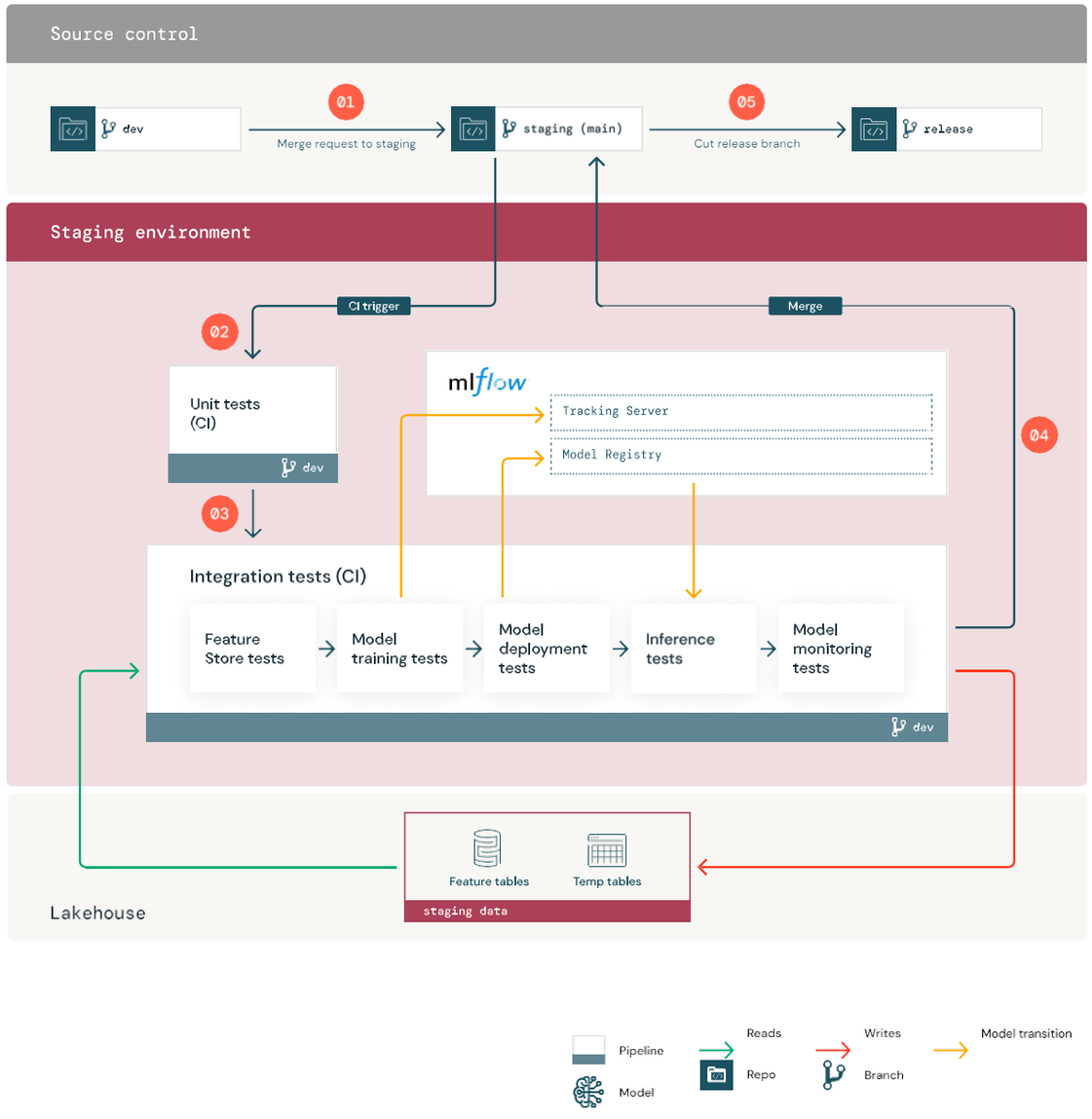
Código de commit

Depois de desenvolver código para preparação, treinamento, inferência e outros pipelines, o data scientist ou engenheiro de ML faz o commit das alterações do branch de desenvolvimento no controle do código-fonte. Esta seção não discute os pipelines de implantação contínua, inferência ou monitoramento em detalhes. Consulte a seção "Produção" abaixo para obter mais informações."

Preparação

A transição do código do desenvolvimento para a produção ocorre no ambiente de preparação. Esse código inclui código de treinamento de modelo e código auxiliar para preparação, inferência etc. Tanto cientistas de dados quanto engenheiros de ML são responsáveis por escrever testes de código e modelos, mas engenheiros de ML gerenciam os pipelines de integração contínua e orquestração.

Figura 5



Dados

O ambiente de preparação pode ter sua própria área de armazenamento para testar tabelas de recursos e pipelines de ML. Esses dados geralmente são temporários e ficam retidos apenas por tempo suficiente para executar testes e investigar falhas nos testes. Esses dados podem ser legíveis no ambiente de desenvolvimento para depuração.

Código de merge

- **Solicitação de merge**

  O processo de implantação começa quando uma solicitação de merge (ou pull) é enviada para o branch de preparação do projeto no controle do código-fonte. É comum usar o branch "principal" como o branch de preparação.

- **Testes unitários (CI)**

  Esta solicitação de merge cria automaticamente o código-fonte e aciona os testes unitários. Se os testes falharem, a solicitação de merge será rejeitada.

Testes de integração (CI)

Depois, a solicitação de merge passa por testes de integração, que executam todos os pipelines para confirmar se eles funcionam corretamente juntos. O ambiente de preparação deve imitar o ambiente de produção na medida do possível, executando e testando pipelines para preparação, treinamento de modelos, inferência e monitoramento.

Os testes de integração podem trocar a fidelidade dos testes por velocidade e custo. Por exemplo, quando o treinamento de modelos é caro, é comum testar o treinamento de modelos em pequenos conjuntos de dados ou em menos iterações para reduzir custos. Quando os modelos são implantados por trás das APIs REST, alguns modelos de alto SLA podem merecer testes de carga em grande escala nesses testes de integração, enquanto outros modelos podem ser testados com pequenos trabalhos em batches ou algumas consultas em endpoints REST temporários.

Assim que os testes de integração forem aprovados no branch de preparação, o código poderá ser promovido para produção.

- **Merge**

  Se todos os testes forem aprovados, será feito o merge do novo código no branch de preparação do projeto. Se os testes falharem, o sistema de CI/CD deverá notificar os usuários e publicar os resultados na solicitação de merge (pull).

Observação: pode ser útil programar testes de integração periódicos no branch de preparação, especialmente se o branch for atualizado com frequência com solicitações de merge simultâneas.

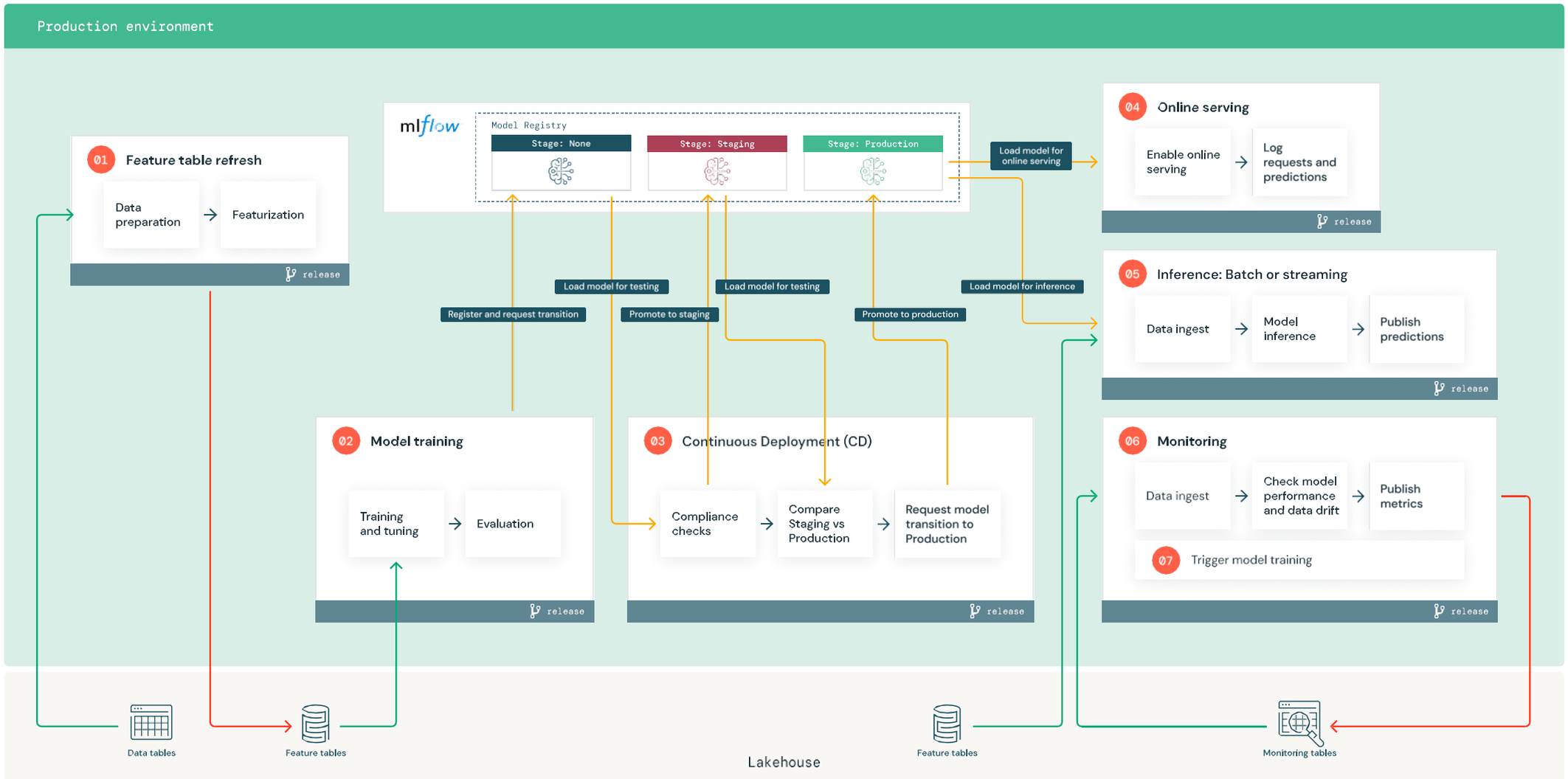
Corte do branch de liberação

Depois que os testes de CI passarem por um commit no branch de preparação, os engenheiros de ML poderão cortar um branch de liberação desse commit.

Produção

O ambiente de produção normalmente é gerenciado por um conjunto seleto de engenheiros de ML e é onde os pipelines de ML atendem diretamente aos negócios ou às aplicações. Esses pipelines calculam novos valores de recursos, treinam e testam novas versões de modelo, publicam previsões em tabelas ou aplicações downstream e monitoram todo o processo para evitar a degradação de desempenho e instabilidade. Apesar de ilustrarmos a inferência em batch e streaming junto com a disponibilização online abaixo, a maioria das aplicações de ML usará apenas um desses métodos, dependendo dos requisitos de negócios.

Figura 6



Embora os data scientists possam não ter acesso de gravação ou compute ao ambiente de produção, é importante que tenham visibilidade para resultados de testes, logs, artefatos de modelo e status dos pipelines de ML em produção. Essa visibilidade permite identificar e diagnosticar problemas na produção.

Atualização da tabela de recursos

Este pipeline transforma os dados mais recentes do Lakehouse em produção em tabelas de recursos de produção. Ele pode usar compute em batch ou streaming, dependendo dos requisitos de atualização para treinamento e inferência downstream. O pipeline pode ser definido como um job da Databricks agendado, acionado ou em execução contínua.

Treinamento de modelos

O pipeline de treinamento do modelo é executado quando as alterações no código afetam a preparação upstream ou a lógica de treinamento, ou quando o retreinamento automatizado é programado ou acionado. Este pipeline é executado com base nos dados completos de produção.

- **Treinamento e ajuste**

  Durante o processo de treinamento, os registros são gravados no servidor de acompanhamento do MLflow. Eles incluem métricas de modelo, parâmetros, tags e o próprio modelo.

  Durante o desenvolvimento, os data scientists podem testar muitos algoritmos e hiperparâmetros, mas é comum restringir essas opções às de melhor desempenho no código de treinamento de produção. Restringir o ajuste pode reduzir a variação do ajuste no treinamento automatizado e acelerar o treinamento e o ajuste.

- **Avaliação**

  A qualidade do modelo é avaliada por meio de testes em dados de produção retidos. Os resultados desses testes são registrados no servidor de acompanhamento do MLflow. Durante o desenvolvimento, os data scientists terão selecionado métricas de avaliação significativas para o caso de uso, e essas métricas ou sua lógica personalizada serão usadas nessa etapa.

- **Registrar e solicitar transição**

  Após o treinamento do modelo, o artefato do modelo é registrado no MLflow Model Registry do ambiente de produção, definido inicialmente como 'stage=None'. A etapa final desse pipeline é solicitar uma transição do modelo recém-registrado para 'stage=Staging'.

Implantação contínua (CD)

O pipeline de CD é executado quando o pipeline de treinamento termina e solicita a transição do modelo para ‘stage=Staging’. Há três tarefas importantes neste pipeline:

Verificações de conformidade

Estes testes carregam o modelo do Model Registry, realizam verificações de conformidade (para tags, documentação etc.) e aprovam ou rejeitam a solicitação com base nos resultados do teste. Se as verificações de conformidade exigirem experiência humana, esta etapa automatizada poderá computar estatísticas ou visualizações para as pessoas revisarem em uma etapa de aprovação manual no final do pipeline de CD. Independentemente do resultado, os resultados desta versão do modelo são registrados no Model Registry por meio de metadados em tags e comentários em descrições.

A UI do MLflow pode ser usada para gerenciar manualmente as solicitações de transição de estágio, mas as solicitações e transições podem ser automatizadas por meio de APIs do MLflow e webhooks. Se o modelo passar nas verificações de conformidade, a solicitação de transição será aprovada, e o modelo será promovido para ‘stage=Staging’. Se o modelo falhar, a solicitação de transição será rejeitada, e o modelo será movido para 'stage=Archived' no Model Registry.

Comparar preparação e produção

Para evitar a degradação do desempenho, os modelos promovidos para o ‘stage=Staging’ devem ser comparados aos modelos de ‘stage=Production’ que eles devem substituir. As métricas para comparação devem ser definidas de acordo com o caso de uso, e o método para comparação pode variar de implantações contínuas a testes A/B. Todos os resultados de comparação são salvos em tabelas de métricas no lakehouse.

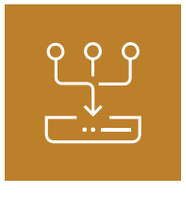
Se esta for a primeira implantação e ainda não houver um modelo 'stage=Production', o modelo 'stage=Staging' deverá ser comparado a uma heurística comercial ou a outro threshold como linha de base. Para uma nova versão
de um modelo de 'stage=Production' existente, o modelo de ‘stage=Staging’ é comparado ao modelo de ‘stage=Production’ atual.

- **Solicitar transição de modelo para produção**
  Se o modelo candidato passar nos testes de comparação, será feita uma solicitação para fazer a transição para 'stage=production' no Model Registry. Assim como em outras solicitações de transição de estágio, notificações, aprovações e rejeições podem ser gerenciadas manualmente por meio da UI do MLflow ou automaticamente por meio de APIs e webhooks. Esse também é um bom momento para considerar a supervisão humana, pois é a última etapa antes que um modelo esteja totalmente disponível para aplicações posteriores. Uma pessoa pode revisar manualmente as verificações de conformidade e as comparações de desempenho para realizar verificações difíceis de automatizar.

Disponibilização online (APIs REST)

Para casos de uso com menor throughput e menor latência, geralmente é necessária a disponibilização online. Com o MLflow, é simples implantar modelos no Databricks Model Serving, endpoints de disponibilização do provedor de nuvem ou camadas de disponibilização on-prem ou personalizadas.

Em todos os casos, o sistema de disponibilização carrega o modelo de produção do Model Registry após a inicialização. Em cada solicitação, ele obtém elementos a partir de uma Feature Store online, pontua os dados e devolve previsões. O sistema de serviço, a camada de transporte de dados ou o modelo em si podem registrar solicitações e previsões.



Inferência: batch ou streaming

Este pipeline é responsável por ler os dados mais recentes da Feature Store, carregando o modelo de 'stage=Production' no Model Registry, realizando previsões de inferência e publicação. Para maior throughput, casos de uso de latência mais alta, a inferência em batch ou streaming geralmente é a opção mais econômica.

Um trabalho em batch provavelmente publicaria previsões em tabelas do Lakehouse, em uma conexão JDBC ou em arquivos planos. Um trabalho de streaming provavelmente publicaria previsões para tabelas do Lakehouse ou para filas de mensagens como Apache Kafka.

Monitoramento

Os dados de entrada e as previsões do modelo são monitorados, tanto em relação às propriedades estatísticas (drift de dados, desempenho do modelo etc.) quanto ao desempenho computacional (erros, throughput etc.). Essas métricas são publicadas para painéis e alertas.

Ingestão de dados

Este pipeline lê logs de inferência em batch, streaming ou online.

Verifique a precisão e o drift de dados

Em seguida, o pipeline calcula métricas sobre os dados de entrada, as previsões do modelo e o desempenho da infraestrutura. As métricas que medem as propriedades estatísticas geralmente são escolhidas pelos data scientists durante o desenvolvimento, enquanto as métricas de infraestrutura geralmente são escolhidas pelos engenheiros de ML.

Publicar métricas

O pipeline grava em tabelas do Lakehouse para análise e geração de relatórios. Ferramentas como Databricks SQL são usadas para produzir painéis de monitoramento, permitindo verificações de integridade e diagnóstico. O trabalho de monitoramento ou a ferramenta de criação de painéis emite notificações quando métricas de integridade ultrapassam os limites definidos.

Acionar treinamento de modelo

Quando as métricas de monitoramento do modelo indicam problemas de desempenho, ou quando um modelo inevitavelmente se torna desatualizado, o data scientist pode precisar retornar ao ambiente de desenvolvimento e desenvolver uma nova versão do modelo.

Retreinamento

Esta arquitetura oferece suporte ao retreinamento automático usando o mesmo pipeline de treinamento de modelo acima. Embora seja recomendável começar com o treinamento acionado manualmente, as organizações podem adicionar o treinamento agendado e/ou acionado quando necessário.

- **Agendado**
  Se novos dados forem disponibilizados regularmente, a repetição do treinamento de modelos em um agendamento definido pode ajudar os modelos a acompanhar as mudanças de tendências e comportamento.

- **Acionado**
  Se o pipeline de monitoramento puder identificar problemas de desempenho do modelo e enviar alertas, ele também poderá acionar o retreinamento. Por exemplo, se a distribuição dos dados de entrada mudar significativamente ou se o desempenho do modelo piorar, o retreinamento e a reimplantação automáticos podem aumentar o desempenho do modelo com o mínimo de intervenção humana.

Quando os próprios pipelines de preparação ou retreinamento começam a exibir problemas de desempenho, o data scientist pode precisar retornar ao ambiente de desenvolvimento e retomar a experimentação para resolver esses problemas.

Observação: embora seja aceito nessa arquitetura, o treinamento automatizado não é necessário, e é preciso cuidado nos casos em que ele é implementado. É inerentemente difícil automatizar a seleção da ação correta a ser tomada a partir do monitoramento do modelo
Alertas. Por exemplo, se o drift de dados for observado, isso indica que devemos treinar automaticamente ou que devemos projetar recursos adicionais para codificar um novo sinal nos dados?

CAPÍTULO 4:

LLMOps – Operações de grandes modelos de linguagem

Grandes modelos de linguagem (LLMs)

Os LLMs estão onipresentes no mundo dos negócios e das notícias, e não há dúvida de que eles afetarão inúmeros setores. Além de trazer um grande potencial, eles apresentam um novo conjunto de perguntas para MLOps:

- A engenharia de prompt faz parte das operações e, em caso afirmativo, o que é necessário?
- Já que "grande" em "LLM" é pouco para representar as dimensões enormes, como muda questão de custo/desempenho?
- É melhor usar APIs pagas ou ajustar um modelo próprio?... e muito mais!

A boa notícia é que "LLMOps" (MLOps para LLMs) não é tão diferente do MLOps tradicional. No entanto, algumas partes da sua plataforma e processo de MLOps podem exigir mudanças, e sua equipe precisará aprender um modelo mental de como os LLMs coexistem junto com o tradicional ML em suas operações.

Nesta seção, explicaremos o que pode mudar para MLOps com a introdução dos LLMs. Discutiremos vários tópicos importantes em detalhes, da engenharia de prompt até empacotamento e questões de custo/desempenho. Também fornecemos um diagrama de arquitetura de referência para ilustrar o que pode mudar em seu ambiente de produção.

O que muda com os LLMs?

Se você não conhece os grandes modelos de linguagem (LLMs), consulte este resumo para uma introdução breve. Resumindo em uma frase: os LLMs são uma nova classe de modelos de processamento de linguagem natural (PLN) que superaram significativamente seus antecessores em desempenho em uma variedade de tarefas, como respostas a perguntas abertas, resumos e execução de instruções quase arbitrárias.

Do ponto de vista do MLOps, os LLMs trazem novos requisitos, com implicações para práticas e plataformas de MLOps. Resumimos brevemente as principais propriedades dos LLMs e as implicações para o MLOps aqui, e aprofundaremos mais detalhes na próxima seção.

Tabela 3

PRINCIPAIS PROPRIEDADES DOS LLMS	IMPLICAÇÕES PARA MLOPS
Os LLMs estão disponíveis em vários formatos: ▪ Modelos proprietários muito genéricos por meio de APIs pagas ▪ Modelos de código aberto que variam de aplicações genéricas a específicas ▪ Modelos personalizados ajustados para aplicações específicas	**Processo de desenvolvimento:** os projetos geralmente se desenvolvem incrementalmente, começando de modelos existentes, de terceiros ou de código aberto e terminando com modelos ajustados personalizados.
Muitos LLMs aceitam consultas gerais de linguagem natural e instruções como entrada. Essas consultas podem conter "solicitações" cuidadosamente projetadas para obter as respostas desejadas.	**Processo de desenvolvimento:** a criação de modelos de texto para consulta de LLMs geralmente é uma parte importante do desenvolvimento de novos pipelines de LLM. **Empacotar artefatos de ML:** muitos pipelines de LLM usam LLMs ou endpoints de disponibilização de LLM existentes; a lógica de ML desenvolvida para esses pipelines pode focar em templates de prompt, agentes ou "cadeias" em vez do próprio modelo. Os artefatos de ML empacotados e promovidos para produção muitas vezes podem ser esses pipelines, em vez de modelos.
Muitos LLMs podem receber prompts com exemplos e contexto, ou informações adicionais para ajudar a responder à query.	**Infraestrutura de disponibilização:** ao ampliar queries de LLM com contexto, é útil usar ferramentas que antes eram incomuns, como bancos de dados vetoriais, para pesquisar contexto relevante.
Os LLMs são modelos de aprendizagem profunda muito grandes, muitas vezes variando de gigabytes a centenas de gigabytes.	**Infraestrutura de disponibilização:** muitos LLMs podem exigir GPUs para disponibilização do modelo em tempo real. **Relação custo/desempenho:** uma vez que modelos maiores exigem mais compute e, portanto, são mais caros para disponibilizar, pode ser necessário usar técnicas para reduzir o tamanho e o compute do modelo.
Os LLMs são difíceis de avaliar por meio de métricas tradicionais de ML; pois, muitas vezes, não há uma resposta "certa".	**Feedback humano:** como o feedback humano é essencial para avaliar e testar LLMs, ele deve ser incorporado mais diretamente ao processo de MLOps, tanto para testes e monitoramento quanto para ajustes futuros.

A lista acima pode parecer longa, mas, como veremos na próxima seção, muitas ferramentas e processos existentes exigem apenas pequenos ajustes para se adaptarem a esses novos requisitos. Além disso, muitos aspectos não mudam:

- A separação entre desenvolvimento, preparação e produção continua a mesma
- O controle de versão do Git e os registros de modelos continuam sendo os principais canais para promover pipelines e modelos em direção à produção
- A arquitetura de lakehouse para gerenciar dados permanece válida e essencial para eficiência
- A infraestrutura de CI/CD existente não deve exigir alterações
- A estrutura modular do MLOps permanece a mesma, com pipelines para atualização de dados, ajuste de modelos, inferência de modelos etc.

Discussão dos principais tópicos de LLMOPs

Até agora, listamos as principais mudanças potenciais no MLOps à medida que você introduz os LLMs. Nesta seção, veremos os tópicos selecionados em mais detalhes.

Engenharia de prompt

Engenharia de prompt é a prática de ajustar o prompt de texto dado a um LLM para obter melhores respostas — usando técnicas de engenharia. É uma prática muito nova, mas algumas práticas recomendadas já estão surgindo. Abordaremos algumas dicas e práticas recomendadas e links para recursos úteis.

1. Prompts e engenharia de prompt são específicos de cada modelo. Um prompt dado a dois modelos diferentes geralmente *não* produz os mesmos resultados. Da mesma forma, dicas de engenharia de prompt não se aplicam a todos os modelos. No caso extremo, muitos LLMs foram ajustados para tarefas específicas de PLN e nem mesmo exigem prompts. Por outro lado, LLMs muito genéricos se beneficiam muito de prompts cuidadosamente elaborados.
2. Ao abordar a engenharia de prompt, vá do simples ao complexo: acompanhe, crie templates e automatize.
   - Comece acompanhando queries e respostas para poder compará-las e iterar para melhorar os prompts. Ferramentas existentes, como o MLflow, fornecem recursos de acompanhamento; consulte Acompanhamento de LLM do MLflow para obter mais detalhes. Verificar o código estruturado do pipeline do LLM no controle de versões também ajuda no desenvolvimento de prompts, pois os diffs do git permitem revisar as alterações nos prompts ao longo do tempo. Consulte também a seção abaixo sobre modelo de empacotamento e pipelines para obter mais informações sobre como acompanhar versões de prompt.
   - Em seguida, você pode usar ferramentas para criar templates de prompts, especialmente se os prompts se tornarem complexos. Ferramentas mais recentes específicas de LLMs, como LangChain e LlamaIndex, fornecem esses templates e muito mais.
   - Por fim, considere a possibilidade de automatizar a engenharia de prompt, substituindo a engenharia manual pelo ajuste automatizado. O ajuste de prompts transforma o desenvolvimento de prompts em um processo orientado por dados, semelhante ao ajuste de hiperparâmetros do ML tradicional. O Framework Demonstrate-Search-Predict (DSP) é um bom exemplo de uma ferramenta para ajuste de prompts.

Recursos

Há muitos bons recursos sobre engenharia de prompt, especialmente para modelos e serviços populares:

→ Curso da DeepLearning.AI sobre engenharia de prompt no ChatGPT

→ Guia de engenharia de prompt da DAIR.AI

→ Práticas recomendadas para engenharia de prompt com a API OpenAI

3 A maioria das dicas de engenharia de prompt publicadas online atualmente são para o ChatGPT, devido à sua imensa popularidade. Algumas delas também podem ser generalizadas para outros modelos. Veja a seguir algumas dicas:

- Use prompts claros e específicos, que podem incluir instrução, contexto (se necessário), query ou entrada do usuário e uma descrição do tipo ou formato de saída desejado
- Forneça exemplos em seu prompt ("aprendizado few shot") para ajudar o LLM a entender
- o que você deseja Diga ao modelo como se comportar. Por exemplo: admitir quando não conseguir responder a uma pergunta

Diga ao modelo para raciocinar passo a passo ou explicar seu raciocínio

Se o prompt incluir entrada do usuário, use técnicas para evitar hacking do prompt, como deixar bem claro quais partes do prompt correspondem à sua instrução e quais correspondem à entrada do usuário

Modelos de empacotamento ou pipelines para implementação

No ML tradicional, geralmente há dois tipos de lógica de ML para empacotar para implantação: modelos e pipelines. Esses artefatos geralmente são gerenciados para produção por meio de um controle de versões do Model Registry e do Git, respectivamente.

Com os LLMs, é comum empacotar a lógica de ML em novas formas.

Essas formas podem incluir:
- uma chamada leve para um serviço de API do LLM (de terceiros ou interno)
- Uma "corrente" da LangChain ou um pipeline análogo de outra ferramenta. A cadeia pode chamar uma API LLM ou um modelo de LLM local.
- Um pipeline LLM ou LLM+tokenizer, como um pipeline Hugging Face . Esse pipeline pode usar um modelo pré-treinado ou um modelo ajustado personalizado.
- Um prompt projetado, possivelmente armazenado como um modelo em uma ferramenta como o LangChain

Embora os LLMs adicionem novas terminologias e ferramentas para compor a lógica de ML, todas as opções acima ainda constituem modelos e pipelines. Portanto, as mesmas ferramentas, como o MLflow, podem ser usadas para empacotar LLMs e pipelines de LLM para implantação. As variantes de modelo integradas incluem:

- PyTorch e TensorFlow
- Hugging Face Transformers (relacionado a isso, consulte MLflowCallbackdo Hugging Face Transformers)
- LangChain
- API da OpenAI
- (Consulte a documentação para obter uma lista completa)

Para outros pipelines de LLM, o MLflow pode empacotar os pipelines por meio da variante MLflow pyfunc, que pode armazenar código Python arbitrário.

Observação sobre o controle de versões do prompt: assim como é útil acompanhar versões de modelos, também é válido acompanhar versões de prompts (e versões de pipelines de LLM, de forma mais geral). Empacotar prompts e pipelines como modelos do MLflow simplifica o controle de versões. Assim como um modelo recém-treinado pode ser acompanhado como uma nova versão de modelo no MLflow Model Registry, um prompt recém-atualizado pode ser acompanhado como uma nova versão de modelo.

Observação sobre a implantação de modelos vs. código: suas decisões sobre o empacotamento da lógica de ML como código controlado por versão vs. modelos registrados ajudam a informar sua decisão sobre a escolha entre os modelos de implantação, o código de implantação e as arquiteturas
híbridas. Analise a subseção abaixo sobre feedback humano e certifique-se de ter um processo de teste bem definido para quaisquer artefatos que escolher implantar.

Gerenciamento da relação custo/desempenho

Um dos principais tópicos de operações para LLMs é gerenciar a relação custo/desempenho, especialmente para inferência e disponibilização. Uma vez que LLMs "pequenos" têm centenas de milhões de parâmetros e LLMs grandes podem chegar a centenas de bilhões de parâmetros, o compute pode se tornar uma grande despesa. Felizmente, há muitas maneiras de gerenciar e reduzir custos quando necessário. Veremos algumas dicas importantes para equilibrar produtividade e custos.

1. Comece de forma simples, mas planeje a escala. Ao desenvolver uma nova aplicação com tecnologia de LLM, a velocidade de desenvolvimento é fundamental, portanto, é aceitável usar opções mais caras, como APIs pagas para modelos existentes. À medida que avança, certifique-se de coletar dados como queries e respostas. No futuro, você pode usar esses dados para ajustar um modelo menor e mais barato que você possa possuir.
2. Analise seus custos. Quantas queries por segundo você espera? As solicitações virão em sequência? Quanto custa cada query? Essas estimativas informam sobre a viabilidade do projeto e ajudam a decidir quando considerar trazer o modelo internamente com modelos de código aberto e ajuste fino.
3. Reduza os custos ajustando LLMs e queries. Existem muitas técnicas específicas de LLM para reduzir cálculos e custos. Isso inclui o encurtamento de queries, ajuste das configurações de inferência e uso de versões de modelos menores.
4. Obtenha feedback humano. É fácil reduzir os custos, mas é difícil dizer como as mudanças afetam os resultados, a menos que se obtenha feedback humano dos usuários finais.

Recursos

Ajuste fino

→ Ajuste fino de grandes modelos de linguagem com Hugging Face e DeepSpeed

→ Webinar: Crie seu próprio grande modelo de linguagem como Dolly: como ajustar e implantar seu LLM personalizado

Destilação, quantização e eliminação de modelos

→ Introdução suave à matriz de 8 bits Multiplicação para transformadores em escala usando transformadores do Hugging Face, Accelerate e bitsandbytes

→ Otimização de inferência em grande modelo transformador

→ https://lilianweng.github.io/posts/2023-01-10-inference-optimization/

Tornando os LLMs ainda mais acessíveis com bitsandbytes, quantização de 4 bits e QLoRA

Métodos para reduzir os custos de inferência

Usar um modelo menor

Escolha outro modelo existente. Experimente versões menores de modelos (como “t5-small” em vez de “t5-base”) ou arquiteturas alternativas.

Ajuste um modelo personalizado. Com os dados de treinamento certos, um modelo ajustado geralmente pode ser menor e/ou ter um desempenho melhor do que um modelo genérico.

Use destilação de modelo (ou destilação de conhecimento). Essa técnica "destila" o conhecimento do modelo original em um modelo menor.

Reduz a precisão do ponto flutuante (quantização). Às vezes, os modelos podem usar aritmética de menor precisão sem perder muito em qualidade.

Reduzir o compute para um determinado modelo

Encurte queries e respostas. O compute escalona de acordo com os tamanhos de entrada e saída, portanto, usar queries e respostas mais concisas reduz os custos.

Ajuste as configurações de inferência. Alguns tipos de inferência, como beam search, exigem mais compute.

Outros

Divida o tráfego. Se o seu retorno sobre o investimento (ROI) para uma query de LLM for baixo, considere dividir o tráfego para que as queries de ROI baixo sejam tratadas por modelos ou métodos mais simples e rápidos. Salve queries de LLM para tráfego de ROI alto.

Use técnicas de eliminação. Se você estiver treinando seus próprios LLMs, há técnicas de eliminação que permitem aos modelos usar compute esparso durante a inferência. Isso reduz o cálculo para a maioria das queries ou todas elas.

Recursos

Aprendizado por reforço com feedback humano (RLHF)

→ Publicação de Chip Huyen no blog: "RLHF: Reinforcement Learning from Human Feedback"

→ Publicação da Hugging Face no blog: "Illustrating Reinforcement Learning from Human Feedback (RLHF)"

→ Wikipédia

Feedback humano, testes e monitoramento

Embora o feedback humano seja importante em muitas aplicações tradicionais de ML, ele ganha muito mais importância para os LLMs. Como a maioria dos LLMs produz linguagem natural, é muito difícil avaliar os resultados por meio de métricas tradicionais. Por exemplo, suponha que um LLM tenha sido usado para resumir um artigo de notícias. Dois resumos igualmente bons podem ter palavras e ordens de palavras quase totalmente distintas, de modo que até mesmo definir um rótulo de "ground-truth" torna-se difícil ou impossível.

Humanos – de preferência os usuários finais – tornam-se essenciais para validar os resultados do LLM. Embora você possa pagar a rotuladores humanos para comparar ou avaliar os resultados do modelo, a melhor prática para aplicações voltadas ao usuário é criar feedback humano nas aplicações desde o início. Por exemplo, um chatbot de suporte técnico pode ter uma opção "clique aqui para conversar com um humano", que fornece feedback implícito indicando se as respostas do chatbot foram úteis.

Em termos de operações, não há muita mudança em relação aos MLOPs tradicionais:

Dados: os feedbacks humano são simplesmente dados e devem ser tratados como qualquer outro tipo de dados. Armazene-os em seu lakehouse e processe-os usando as mesmas ferramentas de pipeline de dados que outros dados.

Testes e monitoramento: testes A/B e implantações incrementais de novos modelos e pipelines podem se tornar mais importantes, superando testes de qualidade offline. Se você puder coletar feedback do usuário, esses métodos de implantação poderão validar modelos antes de serem totalmente implantados.

Ajuste fino: o feedback humano torna-se especialmente importante para os LLMs quando pode ser incorporado aos modelos de ajuste fino por meio de técnicas como o aprendizado por reforço com feedback humano (RLHF). Mesmo que você comece com um modelo existente ou genérico, é possível personalizá-lo para seus objetivos por meio do ajuste fino.

Outros tópicos

- **Dimensionamento:** as práticas de dimensionamento do treinamento, ajuste fino e inferência são semelhantes ao ML tradicional, mas algumas ferramentas podem mudar. Ferramentas como Apache Spark™ e Delta Lake permanecem genéricas o suficiente para seus pipelines de dados LLM e para inferência em batch e streaming, e podem ser úteis para distribuir ajustes finos. Para lidar com o ajuste fino e o treinamento do LLM, talvez seja necessário adotar novas ferramentas, como PyTorch distribuído, TensorFlow distribuído e DeepSpeed.
- **Disponibilização de modelos:** se você gerencia o sistema de disponibilização para seus LLMs, talvez seja necessário fazer ajustes para lidar com modelos maiores. Enquanto a disponibilização com CPUs pode funcionar para modelos menores de deep learning, a maioria dos LLMs se beneficiará ou exigirá GPUs para disponibilizar e inferir.
- **Bancos de dados vetoriais:** algumas aplicações de LLM, mas não todas, exigem bancos de dados vetoriais para pesquisas eficientes baseadas em semelhança de documentos ou outros dados. Os bancos de dados vetoriais podem ser uma adição importante à sua infraestrutura de serviço. Operacionalmente, é como uma Feature Store: uma ferramenta especializada para armazenar dados pré-processados que podem ser consultados por jobs de inferência ou sistemas de disponibilização de modelos.

Arquitetura de referência

Para ilustrar possíveis ajustes em sua arquitetura de referência a partir do MLOps tradicional, fornecemos uma versão modificada da arquitetura de produção anterior.

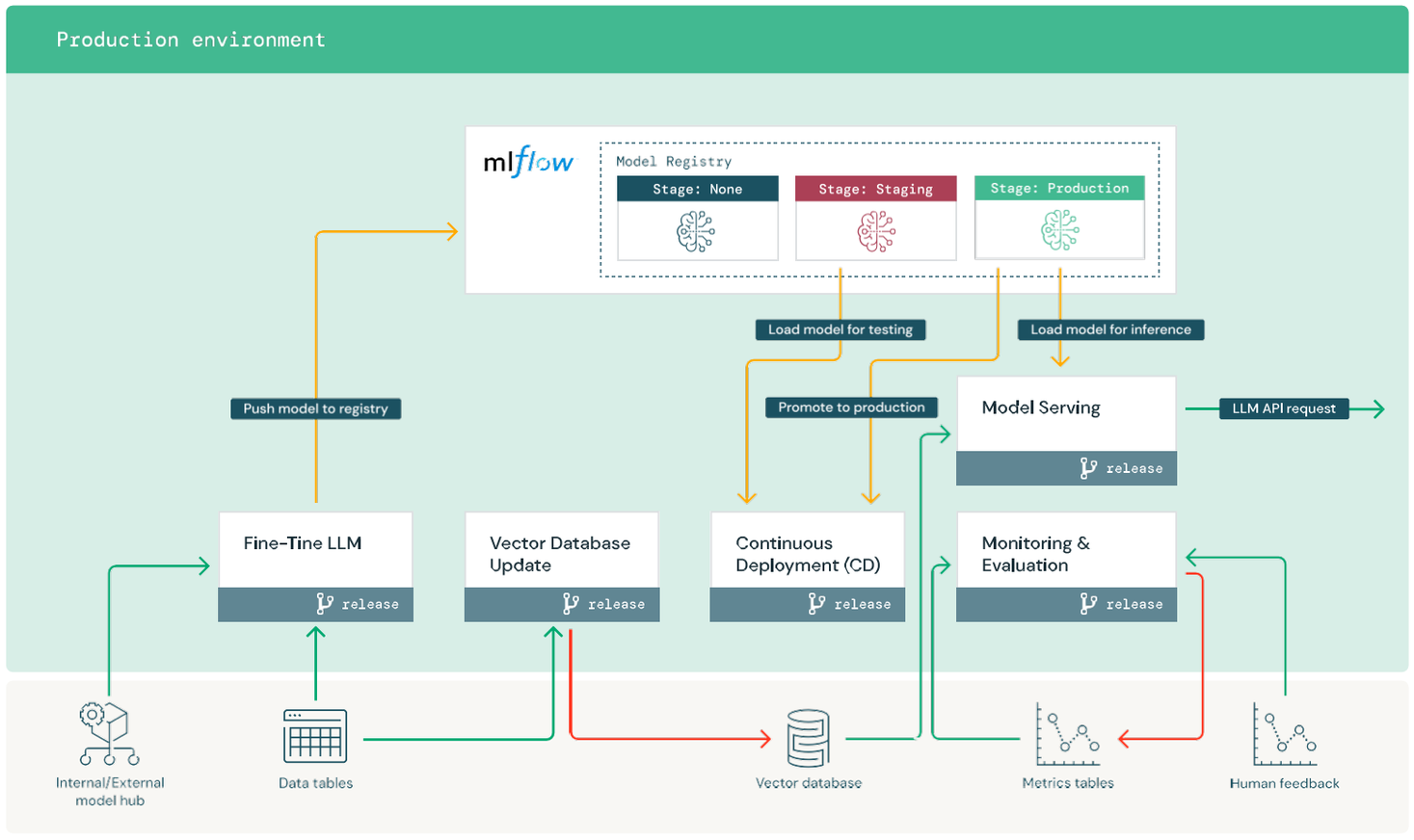


Figura 7

Recursos adicionais

Já que os LLMs são um campo bem novo, incluímos links para vários recursos de LLM abaixo, que não são necessariamente "LLMOps", mas podem ser úteis para você.

→ edX: Professional Certificate in Large Language Models

→ Publicação de Chip Huyen no blog: "Building LLM applications for production"

Listas e tabelas de classificação de LLMs

→ Tabela de classificação da LMSYS

→ Tabela de classificação da Hugging Face de LLMs abertos

→ Center for Research on Foundation Models de Stanford
  → Gráficos de
  → ecossistemas da HELM

→ Publicação no blog: "Open Source ChatGPT Alternatives"

As principais mudanças nesta arquitetura de produção são:

Hub de modelo interno/externo: como as aplicações de LLM geralmente usam modelos pré-treinados existentes, um hub de modelo interno ou externo torna-se uma parte valiosa da infraestrutura. Ele é mostrado aqui em produção para ilustrar usando um modelo base existente que depois é ajustado na produção. Sem ajustes finos, esse hub seria usado principalmente no desenvolvimento.

LLM com ajuste fino: em vez de um novo treinamento do modelo, as aplicações de LLM geralmente ajustam um modelo existente (ou usam um modelo existente sem qualquer ajuste). O ajuste fino é um processo mais leve do que o treinamento, mas é semelhante do ponto de vista operacional.

Banco de dados vetorial: algumas (mas não todos) aplicações de LLM usam bancos de dados vetoriais para pesquisas de similaridade rápidas, normalmente para fornecer contexto ou conhecimento de domínio em queries de LLM. Substituímos a Feature Store (e seu job Atualização da Feature Store) pelo banco de dados vetorial (e seu job Atualização do banco de dados vetorial) para ilustrar que esses armazenamentos de dados e jobs são análogos em termos de operações.

Disponibilização de modelos: a mudança arquitetônica ilustrada aqui é que alguns pipelines de LLM fazem chamadas de API externas, como APIs de LLM internas ou de terceiros. Operacionalmente, isso aumenta a complexidade em termos de latência potencial ou flakiness de APIs de terceiros, bem como de outra camada de gerenciamento de credenciais.

Feedback humano no monitoramento e avaliação: os ciclos de feedback humano podem ser usados no ML tradicional, mas se tornam essenciais na maioria das aplicações de LLM. O feedback humano deve ser gerenciado como outros dados, idealmente incorporado ao monitoramento com base em streaming quase em tempo real.

O que o futuro reserva

Os LLMs só se tornaram populares no final de 2022, e inúmeras bibliotecas e tecnologias estão sendo construídas para dar suporte e aproveitar casos de uso de LLM. Você pode esperar mudanças rápidas. Os LLMs mais poderosos serão de código aberto; ferramentas e técnicas para personalizar LLMs e pipelines de LLM serão mais abundantes e flexíveis; e uma explosão de técnicas e ideias se unirá gradualmente a práticas mais padronizadas.

Embora esse salto tecnológico ofereça a todos nós grandes oportunidades, o uso de tecnologias de ponta requer cuidado extra nos LLMOPs para criar e manter aplicações estáveis e confiáveis baseadas em LLM. A boa notícia é que muitas das ferramentas, práticas e conhecimentos existentes de MLOps serão transferidos sem problemas para os LLMs. Com as dicas e práticas adicionais mencionadas nesta seção, você estará bem preparado para aproveitar o poder dos grandes modelos de linguagem.

Sobre a Databricks

A Databricks é a empresa de dados e IA. Mais de 9.000 organizações em todo o mundo, incluindo a Comcast, Condé Nast e mais de 50% da Fortune 500, contam com a Plataforma Databricks Lakehouse para unificar seus dados, análises e IA. A Databricks tem sede em São Francisco, com escritórios em todo o mundo.

Fundada pelos criadores originais do Apache$^{Spark™}$, Delta Lake e MLflow, a Databricks tem a missão de ajudar as equipes de dados a resolver os problemas mais difíceis do mundo. Para saber mais, siga a Databricks no Twitter, LinkedIn e Facebook.

Inscreva-se para uma avaliação gratuita

databricks