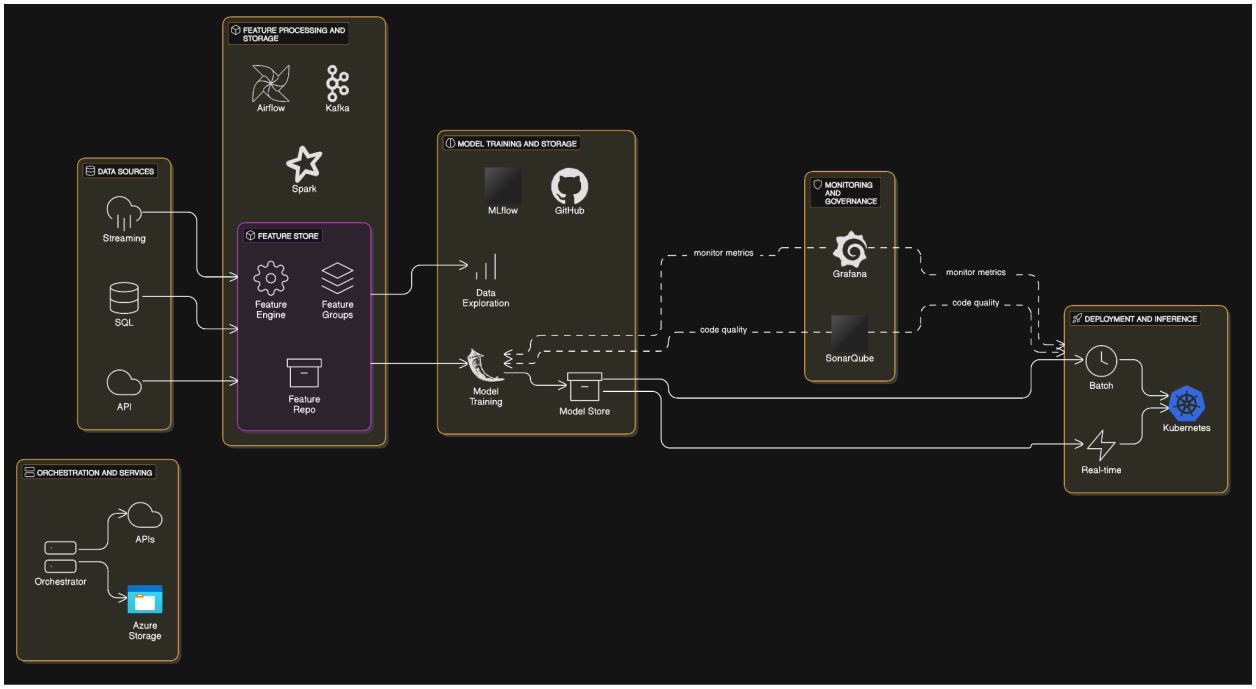
Arquitetura MLOps



- GitOps: O repositório Git é a fonte da verdade para o código da aplicação/modelo e para a configuração da infraestrutura (manifestos Kubernetes).
- Automação: CI/CD para construir, testar e implantar. Retreino acionado por monitoramento ou agendamento.
- Observabilidade: Monitoramento centralizado com Datadog.
- Modularidade: Uso de componentes proprietários (Feature Store, Model Store) e ferramentas padrão (Kubernetes, Glue, etc.).

Fluxo Detalhado:

Preparação de Dados (Batch):

Fonte: Banco de dados relacional SQL com histórico de decisões.

ETL:

- Um job AWS Glue (ou Databricks Job) é agendado para extrair periodicamente novos dados do banco SQL.
- O job realiza transformações, limpeza, engenharia de features (usando Spark/Python).
- As features processadas são carregadas na Proprietary Feature Store para consistência entre treino e inferência.
- Os dados brutos/intermediários e os dados prontos para treino (features + target) são armazenados em buckets AWS S3 para linhagem e reprocessamento, se necessário.

- Desenvolvimento e Experimentação (Data Scientist):

Ambiente: O Cientista de Dados trabalha localmente ou em um ambiente Databricks (Notebooks).

Acesso aos Dados: Ele consulta a Feature Store para obter features consistentes ou acessar dados pré-processados no databricks.

Treinamento: Utiliza Python e bibliotecas de ML (Scikit-learn, XGBoost, etc.).

Tracking: MLflow Tracking (executado em Databricks ou servidor separado) é usado para registrar parâmetros, métricas, e artefatos durante a experimentação.

Registro Final: Uma vez satisfeito, o cientista usa um script/SDK para: registrar a versão final do modelo treinado (e.g., arquivo .pkl ou outro formato) e seus metadados (métricas de avaliação, versão do código, link para run do MLflow) na Model Store.

Versionamento e Trigger CI/CD:

Código Fonte: O cientista (ou engenheiro de ML) commita e envia o código atualizado para um repositório GitHub:

- Código de treinamento.
- Código da API de inferência (que sabe como carregar o modelo da Model Store).
- Dockerfile para construir a imagem da API.
- Testes (unitários, de integração, de qualidade de dados/modelo).
- Potencialmente, arquivos de configuração ou scripts auxiliares.

Trigger: O push para uma branch específica (e.g., main ou staging) dispara um pipeline no Drone CI/CD.

- CI - Continuous Integration (Drone CI/CD):
- Build:
- Checkout do código do GitHub.
- Execução de linters e formatadores.
- Execução de testes unitários (pytest) para o código da API e lógica auxiliar.
- Execução de testes de validação do modelo: Carrega a versão candidata do modelo (referenciada no commit ou obtida da Model Store) e avalia contra um dataset de teste S3, verificando métricas mínimas, talvez fairness/bias. Pode usar MLflow para buscar métricas de runs anteriores para comparação.
- Construção da imagem Docker da API de inferência.

Push: A imagem Docker é tagueada (e.g., com o Git commit SHA ou versão semântica) e enviada para um registro de contêiner (e.g., AWS ECR).

- CD - Continuous Deployment (Drone CI/CD + Argo CD):
- Atualização de Manifestos (GitOps):
- O pipeline do Drone CI/CD atualiza os manifestos Kubernetes (Deployment, Service, Ingress, etc.) em outro repositório Git dedicado à configuração do K8s. A atualização principal é a tag da nova imagem Docker no manifesto do Deployment. Pode também atualizar ConfigMaps ou Secrets se necessário.
- Sincronização (Argo CD):
- Argo CD, rodando no cluster Kubernetes, monitora o repositório Git de configuração.
- Ao detectar a mudança nos manifestos (feita pelo Drone), Argo CD aplica automaticamente as mudanças ao cluster Kubernetes, realizando um rolling update (ou outra estratégia definida) para implantar a nova versão da API sem downtime.

Inferência Online (Kubernetes):

Roteamento: Requisições do aplicativo do banco chegam a um Ingress Controller ou Load Balancer no Kubernetes, que as direciona para o Service da API de empréstimo.

Execução: O Pod da API (executando a imagem Docker):

- Recebe a requisição (body JSON).
- Opcional/Recomendado: Pode consultar a Feature Store para obter features frescas/consistentes se nem todas vieram no body.
- Carrega a versão específica do modelo definida durante o deploy (a referência pode estar em uma variável de ambiente/ConfigMap injetada via K8s, apontando para a versão na Model Store ou S3 se o modelo for empacotado de outra forma).
- Realiza a inferência.
- Retorna a predição

Monitoramento (Datadog + Ferramentas Custom):

Captura: A API de inferência loga informações relevantes para cada predição (input features selecionadas, timestamp, predição, probabilidade, versão do modelo) em formato estruturado (JSON).

Coleta:

- Datadog Agent (rodando como DaemonSet no K8s) coleta:

- Logs da API.
- Métricas de performance da aplicação (APM).
- Métricas de infraestrutura (uso de CPU/memória dos pods, status do K8s).
- Opcional: Logs podem ser enviados também para um bucket S3 para arquivamento de longo prazo ou análise batch.

Análise e Alertas (Datadog):

Dashboards: Visualização de métricas operacionais (latência, taxa de erros, throughput), métricas de recursos e logs.

- Monitores (Alarmes): Configurados para:
- Problemas operacionais (erros 5xx, alta latência).
- Data Drift: Um job agendado (Databricks/Glue/Lambda) pode analisar os logs de inferência (do Datadog Logs ou S3), comparar distribuições de features com o perfil do dataset de treino (possivelmente armazenado com o modelo na Model Store ou MLflow) e enviar métricas customizadas de drift para o Datadog. Alarmes são criados sobre essas métricas.
- Concept Drift/Performance Decay: Similar ao Data Drift, mas quando o ground truth (resultado real da solicitação) se torna disponível, um job compara as predições logadas com o ground truth, calcula métricas de negócio/ML (precisão, recall) ao longo do tempo e envia para o Datadog para alertar sobre degradação.

Retreino Automático:

Trigger:

- Um monitor do Datadog (detectando drift significativo ou queda de performance) dispara um webhook ou evento.
- Um agendador (e.g., Kubernetes CronJob, Airflow, ou até mesmo um agendamento no Databricks/Glue) executa periodicamente.

Execução: O trigger inicia um processo (pode ser um pipeline no Drone CI/CD dedicado ao retreino, ou um Databricks Job):

- Executa o script de treinamento mais recente (do Git).
- Usa os dados mais recentes da Feature Store / S3.
- Loga o novo treinamento no MLflow.
- Avalia o novo modelo.

Se Aprovado: Registra a nova versão na Model Store.

Trigger Deploy: A ação de registrar uma nova versão validada na Model Store pode, por sua vez, disparar automaticamente o pipeline principal de CI/CD (passo 4) para iniciar o deploy da nova versão do modelo.