Desafio MLOps: Pipeline de Inteligência de Mercado na Neoway

Introdução

Bem-vindo(a) ao desafio de MLOps da Neoway! Como a maior empresa de Data Analytics e Inteligência Artificial para negócios da América Latina, nossa missão é transformar dados em conhecimento para gerar valor e impulsionar decisões estratégicas. Para isso, precisamos de pipelines de dados que sejam robustos, escaláveis e confiáveis, utilizando as melhores ferramentas do ecossistema de Big Data.

Seu objetivo será construir um pipeline automatizado que enriquece e consolida dados brutos, criando features valiosas que alimentarão nossas plataformas de Sales & Marketing e Risk & Compliance.

O Cenário

A Neoway ingere diariamente dados de fontes públicas sobre novas empresas registradas no Brasil. Esses dados brutos precisam ser processados para gerar insights agregados por localidade. O time de Inteligência de Mercado quer consumir essas informações para identificar polos de crescimento econômico e novas oportunidades de negócio.

Sua tarefa é criar o pipeline que transforma esses dados brutos em features agregadas e as disponibiliza em um sistema de baixa latência para consulta.

O Desafio

Você deve criar um pipeline que leia um arquivo CSV contendo dados de novas empresas, calcule um conjunto de features agregadas por cidade e as armazene em uma *Feature Store* simplificada. Todo o ambiente deve ser containerizado, testável e automatizado.

Requisitos Técnicos

1. Engenharia e Armazenamento de Features:

- **Fonte de Dados:** Utilize o seguinte arquivo `novas_empresas.csv` como exemplo de input. Em um cenário real, novos arquivos como este seriam ingeridos continuamente. Snippet de código

```
None

cnpj,data_abertura,cidade,estado,capital_social

01.234.567/0001-89,2025-07-28T09:00:00Z,São Paulo,SP,50000.00
```

```
02.345.678/0001-90,2025-07-28T09:30:00Z,Rio de
Janeiro,RJ,75000.00

03.456.789/0001-00,2025-07-28T10:15:00Z,São Paulo,SP,120000.00

04.567.890/0001-11,2025-07-28T11:00:00Z,Belo
Horizonte,MG,30000.00

05.678.901/0001-22,2025-07-28T14:00:00Z,São Paulo,SP,25000.00

06.789.012/0001-33,2025-07-28T15:20:00Z,Rio de
Janeiro,RJ,150000.00
```

-
- **Lógica de Feature Engineering:** Crie um script Python (o uso de **PySpark** é recomendado) que calcule as seguintes features, agregando os dados por `cidade`. O objetivo é avaliar sua familiaridade com o ecossistema Spark, mesmo que o volume de dados do exemplo seja pequeno.
    - `capital_social_total`: Soma do capital social de todas as novas empresas na cidade.
    - `quantidade_empresas`: Contagem do número de novas empresas na cidade.
    - `capital_social_medio`: O capital social médio por empresa na cidade (`capital_social_total` / `quantidade_empresas`).
- **Feature Store:** Utilize o **Redis** como uma *Feature Store* simplificada (key-value). As features agregadas de cada cidade devem ser salvas usando o nome da `cidade` como chave. A estrutura pode ser um HASH do Redis, por exemplo.

2. Orquestração com Airflow:

- Crie uma DAG no Airflow que orquestre o processo.
- A DAG deve ter, no mínimo, duas tarefas:
    1. `iniciar_processamento`: Uma tarefa simples (e.g., `BashOperator`) que apenas loga o início do pipeline.
    2. `processar_e_salvar_features`: A tarefa principal que executa seu script PySpark para ler o CSV, calcular as features e salvá-las no Redis.

3. Containerização com Docker:

- Todo o ambiente (Airflow, Redis, a dependência do Spark e o código do pipeline) deve ser definido e executável através de um único arquivo `docker-compose.yml`.

- O comando `docker-compose up` deve ser suficiente para iniciar todos os serviços e deixar o ambiente pronto para uso.
- Seu código e a definição da DAG devem ser montados nos containers do Airflow através de volumes.

4. Testes:

- A qualidade é fundamental. Escreva **testes unitários** (usando `pytest`, por exemplo) para a sua lógica de engenharia de features em Spark.
- Os testes devem ser independentes e não devem depender de serviços externos como o Redis.

5. Integração Contínua (CI):

- Crie um workflow simples de Integração Contínua usando **GitHub Actions** (ou a ferramenta de sua preferência, como GitLab CI).
- O pipeline de CI deve ser acionado a cada `push` para o repositório.
- Ele deve, no mínimo, executar duas etapas:
    1. `Linting`: Verificar a qualidade do código com `flake8` ou `black`.
    2. `Testing`: Instalar as dependências e rodar a suíte de testes unitários.

6. Monitoramento:

- Implemente um **logging** claro no seu script de processamento. Os logs devem informar o progresso e registrar quaisquer erros que possam ocorrer.

Entregáveis

- O link de um repositório Git (GitHub, GitLab, etc.) contendo todo o código-fonte.
- O repositório deve incluir um `README.md` claro com:
    - Uma breve descrição da solução e das decisões de design que você tomou.
    - Instruções detalhadas sobre como configurar e executar o projeto localmente (e.g., `git clone ...`, `docker-compose up`).
    - Como verificar se o pipeline funcionou corretamente (ex: como inspecionar os dados no Redis).

Critérios de Avaliação

- **Funcionalidade:** O pipeline executa de ponta a ponta sem erros via `docker-compose`?
- **Qualidade do Código:** Clareza, organização, modularidade e aderência às boas práticas do PySpark.
- **Orquestração:** A DAG do Airflow está bem estruturada e funcional?
- **Testes:** A cobertura e a qualidade dos testes unitários para a lógica Spark.

- **CI/CD:** O pipeline de integração contínua está configurado corretamente e cumpre seu propósito?
- **Documentação:** A clareza e a completude do `README.md`.

Pontos Bônus (Opcional)

- Implementar um `health check` na DAG do Airflow para verificar a conexão com o Redis antes de iniciar o processamento.
- Tornar o nome do arquivo CSV de entrada um parâmetro configurável na DAG do Airflow.
- Adicionar tratamento de erros e retentativas (retries) na DAG.

Boa sorte! Estamos ansiosos para ver sua solução.