Desafio técnico - Stack Tecnologias.

Objetivo: Desenvolver um pipeline de dados completo para transformação e carga de dados do dataset público https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data, utilizando serviços da AWS ou Databricks. **Importante**: O candidato ficará à vontade para escolher qual solução utilizar: Databricks ou AWS.

O objetivo é usar Spark para processamento dos dados, para mostrar como processar esses dados com qualidade, performance e escalabilidade. Técnicas de processamento, otimização de tabelas e boas práticas de código serão avaliadas.

Dataset:

https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data

Requisitos Técnicos:

1. **Ingestão**:
    - Carregue esse dataset para um bucket s3 manualmente.

2. **Transformação com Databricks ou AWS**:
    - Utilize **Databricks ou AWS** com **PySpark** para realizar as seguintes transformações:
        - **Limpeza**: Trate valores nulos, duplicados e inconsistências (ex.: normalização de strings, conversão de tipos de dados).
        - **Enriquecimento**: Crie pelo menos duas colunas derivadas, status, ou qualquer outra sugestão.
        - **Agregação**: Gere uma tabela utilizando alguma engine lakehouse mais otimizada possível.
    - Persista os dados transformados em um novo Bucket, simulando uma arquitetura Bronze, Silver e Gold.
3. **Carga**:
    - **Base Relacional**: Persista os dados transformados em um banco **Amazon Redshift** (ou PostgreSQL na AWS RDS ou SQL Warehouse na Databricks, se preferir). Crie uma tabela com schema otimizado para consultas analíticas.
    - **Catalogação**: Utilize **AWS Glue Data Catalog** ou Databricks Unity Catalog para catalogar os dados armazenados no S3 (formato Parquet) e no Redshift, permitindo integração com outras ferramentas.
4. **Orquestração**:
    - Crie um pipeline orquestrado usando **Databricks Workflows** ou **AWS Step Functions** para automatizar a transformação e carga.

    - Inclua pelo menos um mecanismo de monitoramento (ex.: logging de erros, notificações via SNS em caso de falha).
5. **Segurança e Governança**:
    - Configure permissões de acesso no S3 e no Redshift/DynamoDB usando **IAM Roles** para garantir o princípio do menor privilégio.
    - Aplique criptografia nos dados em trânsito e em repouso (ex.: usar KMS para criptografia no S3 e Redshift).
    - Documente as escolhas de modelagem de dados e particionamento no README do projeto.
6. **Validação**:
    - Escreva uma consulta SQL no Redshift ou SQL Warehouse para validar a integridade dos dados carregados (ex.: comparar o total de registros com o dataset original).
    - Opcional: Gere uma visualização simples (ex.: gráfico de barras com doses por estado) usando **Databricks Notebooks** ou outra ferramenta de visualização.

Entregáveis:

- Código organizado em um repositório no **GitHub**, com README detalhando as escolhas técnicas, instruções de execução e justificativas.
- Scripts Python/PySpark e configurações de infraestrutura (ex.: scripts Terraform ou YAML para AWS).
- Evidências de testes (ex.: capturas de tela ou logs mostrando a execução do pipeline e resultados das consultas).

Critérios de Avaliação:

1. **Qualidade do Código**:
    - Estrutura modular, uso de funções reutilizáveis e comentários claros.
2. **Boas Práticas de ETL**:
    - Tratamento robusto de erros, encoding, nulos e duplicatas.
    - Particionamento eficiente no S3 e no Redshift para performance.
3. **Conhecimento em AWS**:
    - Uso correto de serviços como S3, Redshift,Databricks Clusters, Glue e IAM.
    - Configuração de segurança e criptografia.
4. **Conhecimento em Databricks**:
    - Proficiência em PySpark para transformações e agregações.
    - Uso de Databricks Workflows para orquestração.
    - Uso do Step Functions para ambientes AWS.
5. **Modelagem de Dados**:
    - Escolha justificada de colunas, schemas e particionamentos.
    - Modelagem adequada SQL (Redshift) ou SQL Warehouse
6. **Documentação**:
    - README claro com instruções, justificativas e diagramas (se aplicável).
7. **Escalabilidade e Performance**:

    - Pipeline projetado para lidar com grandes volumes de dados.
    - Otimização de consultas e particionamento.

Diferenciais:

- Implementação de testes unitários para validar transformações no PySpark.
- Entendimento de como escalar pipelines de engenharia de dados, desde ingestão até o processamento massivo de dados.
- Entendimento mais profundo dos produtos utilizados durante o teste, soluções escolhidas e o porquê de cada uma.
- Configuração de monitoramento avançado (ex.: integração com CloudWatch ou Databricks Jobs).

Tempo Estimado: 3 a 4 para entrega do projeto.