Projeto Data Reliability Engineer
Teste técnico - Audson Yuri F. Rozendo

Repositório de referência: https://github.com/cauimchagas/dre-3-test

Sumário

1. Introdução

Este documento descreve o processo de resolução dos problemas encontrados no projeto do teste técnico para Data Reliability Engineer. O projeto envolve a configuração de um ambiente utilizando Apache Airflow via Docker Compose, com o objetivo de investigar e solucionar problemas existentes, além de propor uma arquitetura em nuvem para hospedar os componentes envolvidos.

2. Problemas Encontrados

Nesta seção serão documentados os problemas encontrados durante a investigação do projeto, com detalhes sobre as causas e impactos no funcionamento do ambiente.

- No arquivo **compose.yaml** serão apontadas as linhas que encontrei erros:
    - #14 - Nas declarações de volumes, o diretório referenciado aqui está no singular, **dag/** mas a pasta que vem no projeto é no plural **dags/;**
    - #17 - Aqui o parâmetro user: estava com “5000” e o padrão do Airflow é 50000;
    - #31 - Nas variáveis de ambiente do Postgres a variável POSTGRES_USER estava como **admin**, mas em todos os demais pontos do arquivo era mencionado o usuário **airflow;**
    - #59 - o endpoint de checagem health estava com a porta xxxx, causando erro de inacessibilidade, a porta web exposta desse container é a 8080.

- No arquivo **dags/smooth.py** encontrei um erro de sintaxe na linha 12, onde falta **:** para finalizar a declaração da função.

- No host que executa a orquestração faz-se necessárias as seguintes adequações:
    - Criação dos diretórios **logs/** e **plugins** no diretório raiz do projeto**;**
    - Os diretórios **logs/**, **dags/** e **plugins/** precisam ser acessíveis de modo pleno, escrita e leitura para o usuário padrão do Airflow(UID e GID 50000) permitindo que os containers que estão bindando esses volumes possam persistir os dados.

3. Soluções Aplicadas

Aqui serão descritas as soluções aplicadas para corrigir os problemas identificados. Para cada problema, será apresentada a solução com detalhes técnicos e o resultado obtido após a implementação.

- No arquivo **compose.yaml** foram aplicadas as seguintes modificações:
    - #14 - Nas declarações de volumes, o diretório foi alterado para **dags/**
        - **- ./dags:/opt/airflow/dags**
    - #17 - Parâmetro user: alterado para o padrão do Airflow é 50000; (ref)
        - **user: "50000"**
    - #31 - Nas variáveis de ambiente do Postgres a variável POSTGRES_USER foi alterada para **airflow;**
        - **POSTGRES_USER: airflow**
    - #59 - O endpoint de checagem health foi alterado para a porta 8080
        - **test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]**

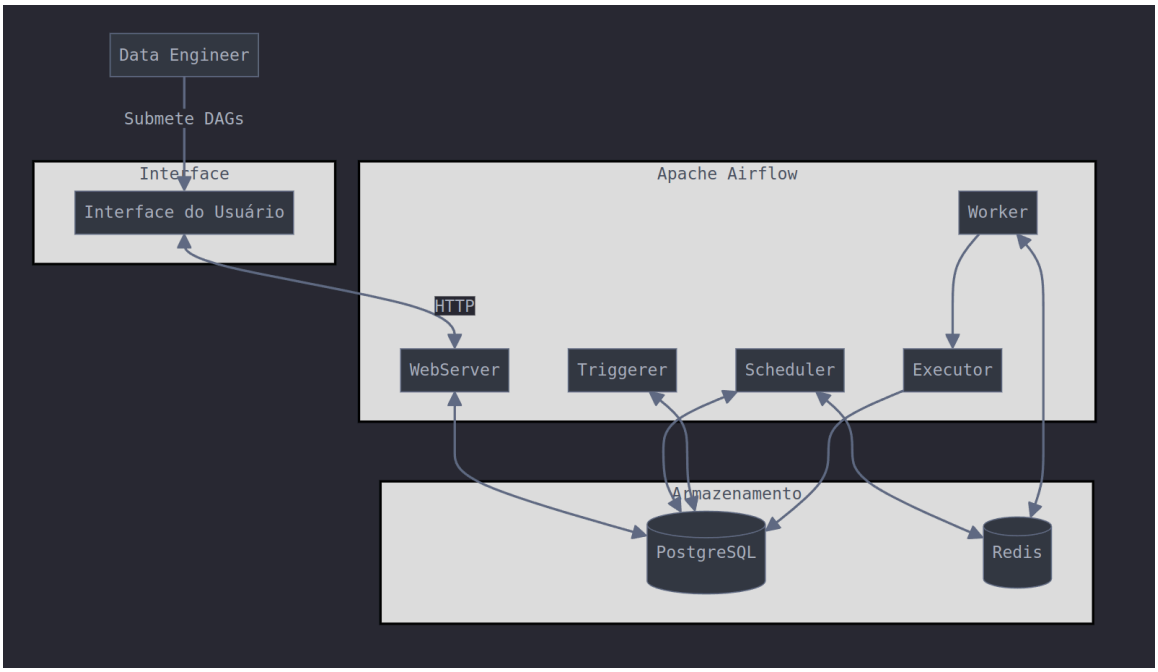
- No arquivo **dags/smooth.py** foi aplicada somente uma inclusão:
    - #12 adicionado **:** ao final da linha
        - **def smooth():**

- No host que executa a orquestração elevei os direitos de acesso nos diretórios de volumes bindados:
    - Os diretórios **logs/**, **dags/** e **plugins/** agora passam a ser gerenciados pelo usuário padrão do Airflow(UID e GID 50000) de modo pleno, escrita e leitura.
        - Estando no diretório raiz do projeto, executar os comandos abaixo;
        - **mkdir logs plugins**
        - **sudo chown -R 50000:50000 ./logs ./dags ./plugins**

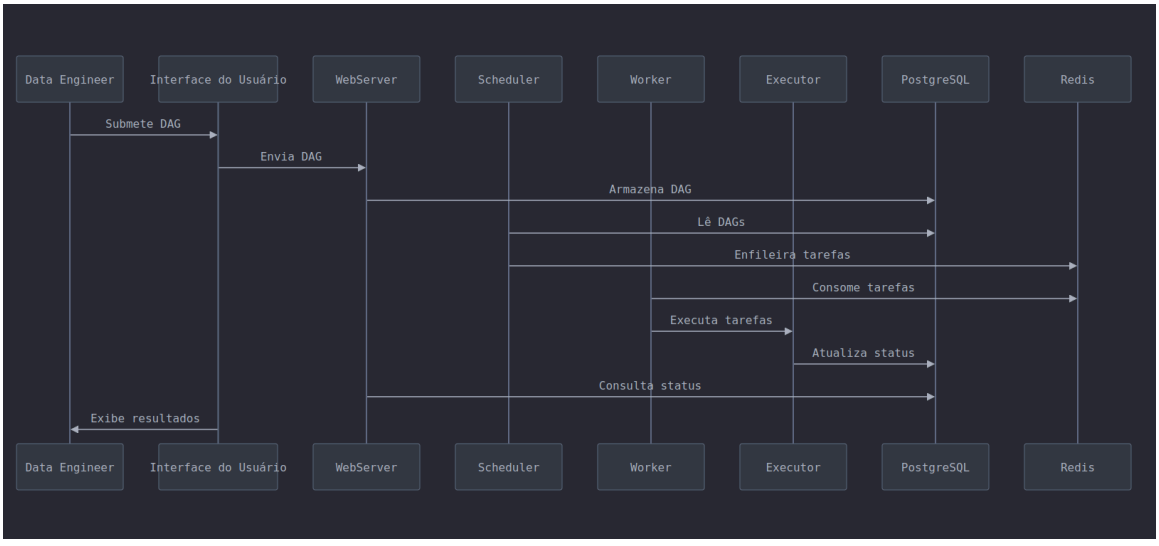
4. Diagrama de Arquitetura e Fluxo de uso

Nesta seção será apresentado o diagrama de arquitetura da solução Airflow e o seu fluxo comum de uso, incluindo componentes como PostgreSQL, Redis e outros serviços essenciais ao funcionamento do sistema.

Componentes da arquitetura:



Fluxo comum de uso:



5. Health Check

Esta seção conterá a implementação de um health check para garantir a saúde e disponibilidade do sistema, incluindo a monitorização de componentes críticos.

1. Postgres

- **Comando**: `pg_isready -U airflow`
- **Intervalo**: 5s | **Tentativas**: 5
- **Descrição**: Verifica se o Postgres está pronto para conexões.

2. Redis

- **Comando**: `redis-cli ping`
- **Intervalo**: 5s | **Timeout**: 30s | **Tentativas**: 50
- **Descrição**: Testa se o Redis responde corretamente com "PONG".

3. Airflow Webserver

- **Comando**: `curl --fail http://localhost:8080/health`
- **Intervalo**: 10s | **Timeout**: 10s | **Tentativas**: 5
- **Descrição**: Verifica a acessibilidade do webserver do Airflow.

4. Airflow Scheduler

- **Comando**: `airflow jobs check --job-type SchedulerJob --hostname "$${HOSTNAME}"`
- **Intervalo**: 10s | **Timeout**: 10s | **Tentativas**: 5
- **Descrição**: Checa a saúde do Scheduler do Airflow.

5. Airflow Worker

- **Comando**: `celery inspect ping -d "celery@$${HOSTNAME}"`
- **Intervalo**: 10s | **Timeout**: 10s | **Tentativas**: 5
- **Descrição**: Verifica se os Workers do Airflow estão respondendo.

6. Airflow Triggerer

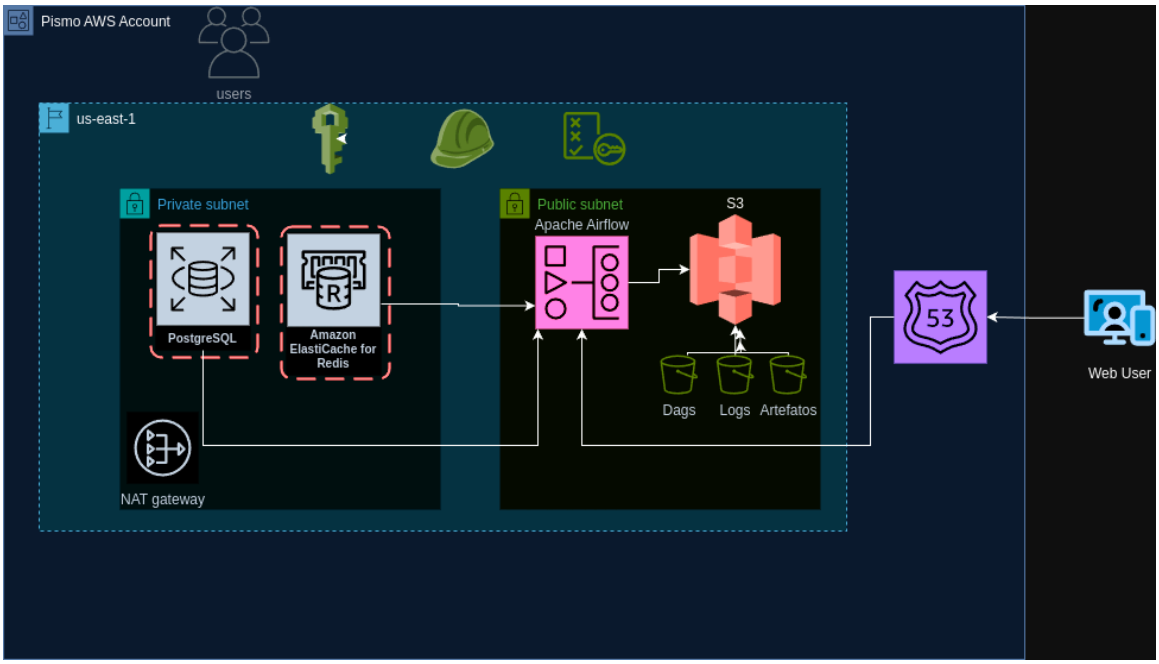
- **Comando**: `airflow jobs check --job-type TriggererJob --hostname "$${HOSTNAME}"`
- **Intervalo**: 10s | **Timeout**: 10s | **Tentativas**: 5
- **Descrição**: Monitora o funcionamento do Triggerer.

7. Flower

- **Comando**: `curl --fail http://localhost:5555/`
- **Intervalo**: 10s | **Timeout**: 10s | **Tentativas**: 5
- **Descrição**: Verifica a interface Flower para monitoramento dos workers.

6. Proposta de Arquitetura em Nuvem

Aqui será apresentada uma proposta de arquitetura em nuvem para hospedar os componentes do projeto, com base em um provedor de nuvem (AWS, por exemplo). Serão descritos os serviços utilizados, suas interações e a configuração da rede.



1. IAM (Identidade e Acesso)

- Criação de **usuários** e **grupos** com permissões específicas utilizando políticas personalizadas para garantir que apenas recursos necessários sejam acessados.
- Definição de **roles** (papeis) específicas para os serviços da AWS, como RDS, Elasticache, S3, e MWAA (Managed Workflows for Apache Airflow).
- Configuração de **políticas** de segurança refinadas para controle de acesso, com **MFA** habilitado para usuários críticos.

2. Conectividade e Rede

- Configuração de uma **VPC** (Virtual Private Cloud) para isolar os recursos do Airflow e seus bancos de dados.
- Criação de **subnets públicas e privadas**. A subnet pública será usada para o acesso externo ao Airflow (via Internet Gateway), enquanto as subnets privadas conterão o banco de dados RDS e Elasticache.
- Definir **Security Groups** para controlar o tráfego dentro da VPC, permitindo apenas as conexões necessárias para os componentes internos (ex.: Webserver -> RDS, Workers -> Redis).
- Implementação de **NAT Gateway** para permitir que instâncias em subnets privadas acessem a internet para atualizações e pacotes sem abrir o acesso direto.

3. Banco de Dados Relacional (RDS)

- Utilizar **Amazon RDS for Postgres** com configuração de alta disponibilidade (Multi-AZ deployment) e **auto-scaling** para gerenciar a carga.
- Implementar backups automáticos e manutenção programada.
- Garantir que o **Postgres** esteja em subnets privadas e que as credenciais de acesso sejam gerenciadas via **AWS Secrets Manager**.

4. Banco de Dados NoSQL (Elasticache)

- Utilizar **Elasticache for Redis** para gerenciar a fila de tarefas do Airflow, configurando as instâncias conforme a carga de trabalho.
- Elasticache será configurado com **subnets privadas** e integrado ao Airflow para comunicação segura.

5. Armazenamento de Arquivos (S3)

- Criar **buckets S3** para armazenar os arquivos de **DAGs**, **logs** e **artefatos** do Airflow.
- Implementar políticas de retenção e ciclo de vida para excluir logs antigos e otimizar o custo de armazenamento.
- Configurar o **versionamento** para garantir a integridade dos arquivos, especialmente os DAGs.

6. Arquitetura Airflow com AWS MWAA

- Utilizar **AWS Managed Workflows for Apache Airflow (MWAA)** para gerenciar o ambiente do Airflow, simplificando a implementação e integração com os serviços da AWS.
- Configurar o MWAA para se conectar ao S3 para armazenamento de DAGs e logs, Redis como broker (via Elasticache) e RDS como banco de dados.

7. Considerações Finais

Esta seção trará uma conclusão sobre o processo de solução dos problemas, incluindo lições aprendidas e possíveis melhorias futuras para o ambiente.

- Ao longo deste projeto, enfrentei uma série de desafios ao configurar e otimizar o **Apache Airflow** em diferentes ambientes, tanto local quanto em nuvem. O processo de solução dos problemas envolveu desde ajustes nas permissões locais, gerenciamento adequado de diretórios bindados para execução em Docker, até a elaboração de uma arquitetura robusta e escalável na AWS, integrando serviços como **RDS**, **Elasticache**, e **S3**.
- Lições importantes foram aprendidas, como a importância de configurar corretamente permissões para usuários e diretórios, e como o monitoramento e os health checks desempenham um papel fundamental na garantia da alta disponibilidade dos serviços. Além disso, a migração para um ambiente cloud trouxe insights sobre a gestão eficiente de recursos e a segurança em ambientes distribuídos.
- Para futuras melhorias, há oportunidades de implementar **auto-scaling** para otimizar ainda mais os recursos de banco de dados e cache, além de aprofundar as práticas de monitoramento com **CloudWatch** para uma visão mais detalhada do desempenho do ambiente. A adoção de pipelines CI/CD automatizados também pode melhorar o processo de deploy das DAGs e componentes do Airflow.
- Este projeto é um marco no aprendizado contínuo de orquestração de workflows e infraestrutura cloud, e oferece uma base sólida para evoluções futuras.