**DESAFIO PISMO**

**Relatório de Post-Mortem e Melhoria Contínua: Diagramas de Arquitetura, Melhorias Implementadas e Futuras**

**Jefferson Cardoso**

Índice

[Post-Mortem 3](#_Toc187401304)

[Incidente 1: O usuário do Airflow não consegue se conectar ao banco de dados 3](#_Toc187401305)

[Incidente 2: Erro de inicialização do banco de dados no Airflow Triggerer 4](#_Toc187401306)

[Incidente 3: A página do Webserver está indisponível 5](#_Toc187401307)

[Incidente 4: Container Airflow-worker sofrendo diversos restarts 6](#_Toc187401308)

[Incidente 5: DAG smooth.py com erro ao ser iniciada 7](#_Toc187401309)

[Ações Extras de Melhorias Implementadas 8](#_Toc187401310)

[Ações de Melhorias Futuras 9](#_Toc187401311)

[Diagramas de Arquitetura 10](#_Toc187401312)

[Diagrama da Arquitetura Atual do Airflow 10](#_Toc187401313)

[Arquitetura do Airflow gerenciada pela AWS 11](#_Toc187401314)

[Arquitetura do Airflow Gerenciada Operacionalmente na AWS 12](#_Toc187401315)

[Conclusão 13](#_Toc187401316)

# Post-Mortem

## Incidente 1: O usuário do Airflow não consegue se conectar ao banco de dados

**Descrição do Erro**: Ao tentar executar o Docker-compose, me deparei com as seguintes mensagens de erro: “Password authentication failed for user 'airflow'” e “role 'airflow' does not exist”.

**Causa Raiz**: Na variável `POSTGRES\_USER` estava configurado o usuário `admin`, entretanto na string de conexão utilizava-se o usuário `airflow`.

**Impacto**: O impacto gerado foi que o Airflow não conseguia inicializar, tornando todo o sistema inoperante.

**Linha do Tempo**:

|  |  |
| --- | --- |
| Data/Horário | Evento |
| 01/01 - 14:53 | Primeira execução do projeto |
| 01/01 – 14:55 | Identificada a discrepância dos usuários |
| 01/01 – 14:56 | Realizado a correção do usuário |
| 01/01 – 14:57 | Realizado uma nova execução do projeto |
| 01/01 – 14:59 | O erro ainda persiste |
| 01/01 – 15:25 | Identificado que o volume do banco (Postgres) permanecia o mesmo antes da correção. |
| 01/01 – 15:26 | Realizado a remoção do volume |
| 01/01 – 15:27 | Realizado uma nova execução |
| 01/01 – 15:28 | Identificado que o erro foi sanado |

|  |  |
| --- | --- |

**Análise do Incidente**: A falha evidenciou a discrepância dos usuários e o uso indevido do usuário `admin` do banco de dados, além de expor em texto puro as credenciais. Isso gerou um grave erro de segurança e impediu que a receita do `compose` fosse reutilizada em outros projetos.

**Ações Corretivas**:

1. Realizada a alteração dos dados sensíveis para variáveis.
2. Realizada a criação específica para o Airflow.
3. Concedido o acesso para o usuário `airflow` apenas ao banco de dados `airflow`, evitando assim que o usuário tenha privilégios de administrador.

**Lições Aprendidas**:

* Sempre considerar a segurança no projeto e pensar em formas de reaproveitar o seu código. Além disso, é importante, em cenários de produção, incluir o projeto em uma esteira CI/CD e utilizar uma ferramenta para varredura de código em busca de possíveis vulnerabilidades. Um exemplo de ferramenta é o Checkov. Outro ponto importante é contar com a colaboração com o time de segurança e o time de banco de dados para melhores resultados.

## **Incidente 2**: Erro de inicialização do banco de dados no Airflow Triggerer

**Descrição do Erro**: Ao tentar inicializar o Airflow Trigger, foi identificado o seguinte erro: "You need to initialize the database. Please run `airflow db init`."

**Causa Raiz**: Identificou-se que o serviço `airflow-trigger` não estava recebendo as configurações/informações do banco de dados.

**Impacto**: A ação de disparar tarefas em resposta aos eventos estava indisponível.

**Linha do Tempo**:

|  |  |
| --- | --- |
| Data/Horário | Evento |
| 02/01 – 23:00 | Análise dos Logs e Identificação do Erro |
| 02/01 – 23:01 | Início da investigação |
| 02/01 – 00:06 | Após acessar o contêiner do `airflow-trigger`, constatou-se que ele não estava herdando as variáveis de conexão com o banco de dados |
| 02/01 – 00:07 | Incluídas as variáveis de conexão no ambiente (`environment`) do serviço `airflow-trigger`, utilizando o `airflow-common-env` para herdar as configurações. |
| 02/01 – 00:08 | Realizada a reexecução do projeto |
| 02/01 – 00:10 | Erro identificado e sanado |

|  |  |
| --- | --- |

**Análise do Incidente**: O incidente evidenciou uma falha na configuração do arquivo `compose`, devido à ausência da menção do ambiente `airflow-common-env` (que contém as configurações para conectar ao banco de dados) no serviço `airflow-trigger`.

**Ações Corretivas**:

1. Nas enviroments do serviço airflow-trigger foi incluído a seguinte configuração **“**<<: \*airflow-common-env” com isso ele contempla as configurações para conectar ao banco de dados.

**Lições Aprendidas**:

* O incidente, por si só, demonstrou a necessidade de implementar o processo de code review. Com o code review, temos a oportunidade de revisar o código e, assim, a ausência da variável seria percebida.

## **Incidente 3:** A página do Webserver está indisponível

**Descrição do Erro**: Container do Webserver está com status unhealthy

**Causa Raiz**: Healthcheck configurado de forma incorreta.

**Impacto**: Interface web para visualizar e gerenciar DAGs está inoperante.

**Linha do Tempo**:

|  |  |
| --- | --- |
| Data/Horário | Evento |
| 02/01 – 00:11 | Análise dos logs e identificação do erro. |
| 02/01 – 00:12 | Início da investigação. |
| 02/01 – 00:14 | Identificado que o healthcheck do serviço airflow-webserver estava configurado com a porta de forma incorreta. Ajuste realizado. |
| 02/01 – 00:15 | Realizado uma nova execução do projeto. |
| 02/01 – 00:17 | Erro identificado e sanado. |

|  |  |
| --- | --- |

**Análise do Incidente:** O incidente evidenciou uma falha na configuração da porta do healthcheck do serviço airflow-webserver. O serviço responde na porta 8080 e estava configurado como XXXX.

**Ações Corretivas**:

1. Realizada a alteração da porta na URL do teste de healthcheck do serviço airflow-webserver. Alterado de http://localhost:XXXX/health para http://localhost:8080/health.

**Lições Aprendidas**:

* O incidente, por si só, mostrou a importância de implementar o processo de code review e um monitoramento eficiente para antecipar a análise do incidente, diminuindo o tempo de downtime.

## **Incidente 4**: Container Airflow-worker sofrendo diversos restarts

**Descrição do Erro**: O serviço do `airflow-worker` estava sofrendo diversos restarts de forma incessante.

**Causa Raiz**: Identificou-se que, devido à falta de recursos computacionais, como memória, o serviço do `airflow-worker` não conseguia se inicializar, resultando em

**Impacto**: As tarefas programadas estão sem execução.

**Linha do Tempo**:

|  |  |
| --- | --- |
| Data/Horário | Evento |
| 03/01 - 22:53 | Identificação do erro |
| 03/01 – 22:54 | Início da investigação |
| 03/01 – 23:15 | Identificados os requisitos mínimos de infraestrutura para a execução do Airflow na [documentação](https://airflow.apache.org/docs/apache-airflow/stable/installation/prerequisites.html) oficial. |
| 03/01 – 23:16 | Realizado o upgrade de recursos da máquina local para suportar mais cargas, aumentando a memória RAM de 4GB para 10GB. |
| 03/01 – 23:20 | Configurados os limits no serviço do `airflow-worker` conforme orientado na [documentação](https://airflow.apache.org/docs/apache-airflow/stable/installation/prerequisites.html). |
| 03/01 – 23:21 | Realizada uma nova execução |
| 03/01 – 23:23 | Erro identificado e sanado |

|  |  |
| --- | --- |

**Análise do Incidente**: O incidente evidenciou a falta de recursos para o serviço do airflow-worker. Não houve geração de erro no log, porém o serviço estava sofrendo reinicializações contínuas.

**Ações Corretivas**:

1. Realizado o upgrade de recursos da máquina local para suportar mais cargas, aumentando a memória RAM de 4GB para 10GB.
2. Configurados os limites no serviço do airflow-worker para 8GB de RAM, conforme a documentação do fabricante.

**Lições Aprendidas**:

* O incidente destacou a importância de seguir as recomendações do fabricante para o uso de recursos.
* Outro ponto importante é a implementação de recursos de monitoramento para melhorar a observabilidade do ambiente.

## **Incidente 5:** DAG smooth.py com erro ao ser iniciada

**Descrição do Erro**: Ao tentar carregar a DAG no Airflow, foi notado o seguinte erro: Broken DAG: SyntaxError: invalid syntax.

**Causa Raiz**: Ao avaliar a mensagem de erro e revisar o código, foi identificada a ausência de dois pontos (:) na definição da função DAG

**Impacto**: O impacto gerado foi que a DAG não era criada devido ao erro citado.

**Linha do Tempo**:

|  |  |
| --- | --- |
| Data/Horario | Evento |
| 03/01 - 23:30 | Inclusão da DAG smooth.py na pasta de DAGs do projeto. |
| 03/01 – 23:31 | Identificação do erro ao criar a DAG |
| 03/01 – 23:32 | Início da investigação |
| 03/01 – 23:36 | Identificação da causa raiz: ausência de dois pontos (:) na definição da função. |
| 03/01 – 23:37 | Ajuste realizado na função. |
| 03/01 – 23:40 | Erro identificado e sanado. |

|  |  |
| --- | --- |

**Análise do Incidente**: O incidente evidenciou uma falha na criação do código da DAG smooth.py, que estava com uma função mal configurada.

**Ações Corretivas**:

1. Realizado o ajuste no arquivo smooth.py, incluindo dois pontos (:) no final da função smooth\_task.

**Lições Aprendidas**:

* O incidente destacou a importância de implementar o processo de code review.
* Além disso, é interessante incluir o processo de CI/CD, utilizando ferramentas de análise de código como SonarQube, Checkmarx, entre outras.

## Ações Extras de Melhorias Implementadas

* Onde eram utilizados dados sensíveis em texto puro, foram transformados em variáveis, aumentando a segurança do ambiente e tornando o código reaproveitável para outros projetos.
* Conforme [pré-requisitos](https://airflow.apache.org/docs/apache-airflow/stable/installation/prerequisites.html) orientandos pela airflow, foi alterada a versão do Python de 3.7 para 3.9.
* Criados dois usuários para o banco de dados (Postgres): um usuário administrador (com permissão total ao banco de dados) destinado ao time de banco de dados para administração, e outro usuário criado é o airflow, com permissão apenas na database do Airflow. Isso aumentou a segurança do ambiente.
* Utilizando a variável AIRFLOW\_UID, foi definido o mesmo UID do usuário que está implantando o ambiente ou, se não definido, utilizamos por padrão o usuário 50000 (também altamente recomendado na documentação do Airflow). Isso possibilita um uso controlado de apenas leitura e escrita, e não mais acesso root, proporcionando maior segurança ao ambiente e mitigando a possibilidade de acessos indevidos.
* Incluídos limites de CPU/memória para os serviços do Airflow, pois dependendo de onde o ambiente for implementado, pode haver um alto consumo de recursos, gerando graves problemas de falta de recursos computacionais e indisponibilizando demais processos. Ponto importante é que os limites foram configurados de acordo com a recomendação do fabricante. Para mais detalhes, na [documentação](https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html) oficial.
* Foi definida uma versão específica no serviço do Redis, alterando de redis:latest para redis:7.4.2. Isso garante maior segurança no ambiente, evitando vulnerabilidades desconhecidas, e proporciona consistência, já que trabalhamos com a mesma versão em diferentes ambientes (desenvolvimento e produção). Além disso, a compatibilidade é mantida, pois uma nova versão do Redis pode ser incompatível com o Airflow.
* Na DAG smooth.py, o parâmetro utilizado ‘schedule’ não era reconhecido pelo Airflow. Como não houve retorno de erro devido à configuração como ‘None’, foi realizado o ajuste para ‘schedule\_interval’, tornando a configuração utilizável para futuros agendamentos.
* Devido à mensagem de aviso "Deprecated API features detected! These feature(s) are not compatible with SQLAlchemy 2.0", e para evitar futuros problemas de incompatibilidade no ambiente, foi criado o arquivo requirements.txt e incluída a biblioteca SQLAlchemy<2.0. Com isso, garantimos a disponibilidade da aplicação. Para uma geração de logs mais limpa, foi incluído o parâmetro SQLALCHEMY\_SILENCE\_UBER\_WARNING: "1" para silenciar o aviso em questão, facilitando a leitura em caso de troubleshooting. Vale lembrar que o ambiente utiliza a versão 1.4.46 da biblioteca SQLAlchemy.
* Conforme encontrado em exemplos de composes disponibilizados pelo próprio Airflow, foram incluídas no processo do airflow-init algumas verificações, tais como: Verificação da utilização do AIRFLOW\_UID. Verificação se o ambiente possui memória e CPU disponíveis para executar o projeto. Disponibilização da documentação do Airflow informando sobre os requisitos necessários para iniciar o Airflow, caso os recursos não estejam disponíveis.

## Ações de Melhorias Futuras

* Implementação de uma ferramenta de gerenciamento de senhas para armazenar dados sensíveis. Exemplos de ferramentas: Vault, Secret Manager, entre outros.
* Implementar a parte de observabilidade. Sugestões de ferramentas para métricas: Flower, Prometheus, Grafana. Para logs: ELK ou Grafana Loki. Isso proporcionará melhor visibilidade do ambiente, abrangendo tanto a parte de logs quanto um monitoramento mais robusto.
* Implementação de uma esteira CI/CD que contemple os processos de code review, análise de código, build e deploy automáticos.
* Remover a permissão de root para os serviços do Redis e Postgres.
* Realizar uma investigação (discovery) para identificar e corrigir o erro "Triggerer's async thread was blocked".
* Implementar um método de autenticação no Redis.
* Realizar uma investigação (discovery) para identificar e corrigir a falha em alterar a permissão da pasta sources.

# Diagramas de Arquitetura

## Diagrama da Arquitetura Atual do Airflow

**A diagram of a computer network

Description automatically generated**

Conforme demonstrado acima, aqui está a arquitetura atual do Airflow. Mesmo sendo simples, ela é eficiente para ambientes não produtivos.

## Arquitetura do Airflow gerenciada pela AWS

**A diagram of a software system

Description automatically generated**

O Airflow gerenciado pela AWS traz alguns ônus e bônus, como:

* **Facilidade e Rapidez de Implantação**: Implantação simplificada e rápida.
* **Escalabilidade Automática**: O ambiente escala automaticamente conforme necessário.
* **Segurança**: Possui uma camada de segurança com o AWS KMS, garantindo a criptografia dos dados.
* **Armazenamento Seguro de Dados Sensíveis**: Utilização do AWS Secrets Manager para armazenar dados sensíveis de forma segura.
* **Observabilidade**: Métricas e logs são enviados automaticamente para o CloudWatch, facilitando a visualização e monitoramento.
* **Custo**: Embora essa abordagem elimine o custo operacional da administração, sem um controle adequado da quantidade de processamento e uma configuração correta do ambiente, pode haver custos inesperados no final do mês.
* **Integração com Recursos AWS**: Possibilidade de integração com diversos recursos da AWS.

## Arquitetura do Airflow Gerenciada Operacionalmente na AWS

**A computer screen shot of a diagram

Description automatically generated**

Conforme a imagem acima, essa arquitetura de gerência operacional na AWS traz alguns ônus e bônus:

* **Implantação/Administração:** A administração segue de forma operacional, demandando mais tempo e esforço na criação e manutenção do ambiente.
* **Custo:** Por ser uma arquitetura robusta, pode ser uma opção cara em casos de poucas utilizações.
* **Observabilidade:** Possui maior liberdade na coleta de métricas, permitindo a criação de métricas customizadas.
* **Segurança:** Com o uso do KMS os dados são criptografados, proporcionando uma camada adicional de segurança.
* **Armazenamento Seguro de Dados Sensíveis:** Utilização do AWS Secrets Manager para armazenar dados sensíveis de forma segura.

# Conclusão

Este desafio revelou-se interessante, pois algumas "simples" correções abriram portas para diversas oportunidades de melhorias no ambiente. Abrangendo desde a performance até a segurança, ele destacou a importância dos processos de arquitetura, documentação e observabilidade.

**Pontos de Melhoria Identificados**

* **Performance**: Identificação de áreas onde ajustes podem melhorar a eficiência do sistema.
* **Segurança**: Implementação de melhores práticas e ferramentas para proteger dados sensíveis.
* **Arquitetura**: Reavaliação e aperfeiçoamento da infraestrutura para suportar melhor as operações.
* **Documentação**: Importância de manter uma documentação atualizada para guiar as operações e facilitar o troubleshooting.
* **Observabilidade**: Implementação de ferramentas de monitoramento e logging para uma melhor visibilidade do ambiente.

Ao abordar esses pontos, podemos evitar incidentes futuros que resultem em downtime e, consequentemente, perda de receita financeira.