Relatório 12 - Prática: Pipelines de Dados I - Airflow (II)

Ronny Gabryel Colatino de Souza

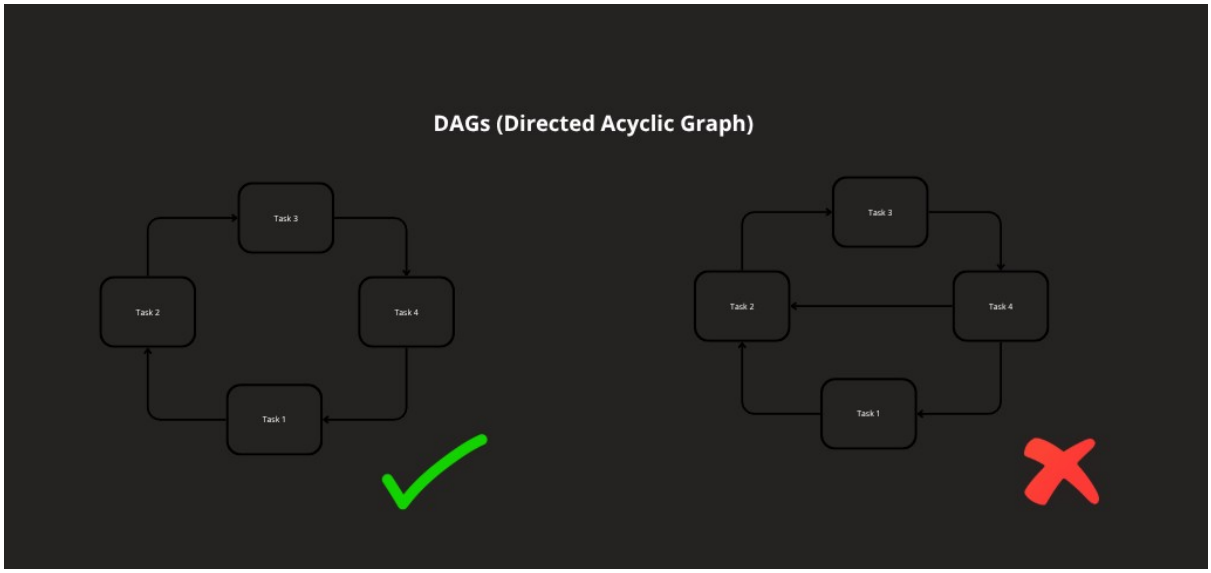
Introdução

Nesse card foi introdizo o objetivo desta prática foi implementar pipelines de dados automatizados com Apache Airflow. A tarefa envolveu configurar o ambiente via Docker desenvolver DAGs para orquestração de tarefas e aplicar conceitos de scheduling e gerenciamento de dependências Esperava-se compreender a arquitetura modular do Airflow e sua utilização em cenários de produção.

Desenvolvimento

DAGs (Directed Acyclic Graph)

DAGs são a forma como o Airflow organiza tarefas cada tarefa é um nó no grafo e as setas mostram a ordem de execução o "acíclico" significa que não pode ter loops a tarefa a vai pra B, B vai pra C, mas C nunca volta pra A isso é importante porque evita que o pipeline fique preso em loops infinitos.



[Criação propria no canvas]

O que mais me chamou atenção foi a modularidade você cria cada tarefa separada e só depois define as dependências com o operador. Isso facilita demais a manutenção, porque se precisar adicionar uma tarefa nova no meio do fluxo é só ajustar quem depende de quem, sem mexer no código das outras tarefas.

Percebi que essa estrutura te força a pensar melhor na arquitetura desde o início em vez de fazer um script gigante que faz tudo em sequência, você é obrigado a quebrar em

componentes menores e pensar nas dependências Isso deixa o código mais limpo e mais fácil de testar.

Arquitetura do Airflow

Arquitetura do Airflow

O Airflow divide as responsabilidades em componentes diferentes e cada um tem um papel específico:

Scheduler: Fica monitorando a pasta de DAGs o tempo todo verifica quais tarefas estão prontas pra rodar baseado nas dependências e no agendamento e manda elas pro Executor é o componente mais crítico porque se ele cair o sistema inteiro para ele é o cérebro que coordena tudo.

Executor: Define como as tarefas vão ser executadas. A escolha do Executor muda completamente a performance

Executor	Quando usar	Paralelismo	Infraestrutura	Principal limitação
SequentialExecutor	Testes e desenvolvimento	Não (1 por vez)	Máquina local	Muito lento
LocalExecutor	Projetos pequenos e médios	Sim (vários processos)	Servidor único	Só escala verticalmente
CeleryExecutor	Produção com alto volume	Sim (workers distribuídos)	Cluster com Redis/RabbitMQ	Complexo de configurar
KubernetesExecutor	Ambientes cloud	Sim (pods sob demanda)	Kubernetes	Precisa de K8s configurado

[Criação propria canvas]

Metadata Database: Guarda tudo relacionado ao Airflow histórico de execuções logs configurações status das DAGs se você perder esse banco perde toda a auditoria e não consegue mais fazer backfill confiável dos dados por isso é fundamental ter backup dele

Web Server: Interface visual onde você acompanha tudo mostra o status das DAGs em tempo real logs de cada tarefa e permite executar coisas manualmente facilita muito o debugging porque você não precisa ficar caçando informação no terminal

DAG Folder: Pasta onde você coloca os arquivos Python com as definições das DAGs o Scheduler fica observando essa pasta pra detectar DAGs novas ou modificadas a forma como você organiza essa pasta impacta a performance se tiver muitos arquivos pode sobrecarregar o Scheduler

O fluxo completo funciona assim Scheduler lê as DAGs da pasta identifica as tarefas prontas manda pro Executor que coloca numa fila os Workers pegam as tarefas da fila executam e registram o resultado no banco o Web Server consulta o banco e mostra tudo isso em tempo real e bunitinho

Operators

Operators são os blocos básicos que você usa pra construir tarefas cada um tem uma função específica

Operator	O que faz	Quando usar
BashOperator	Roda comandos shell	Scripts bash, comandos CLI
PythonOperator	Executa funções Python	Qualquer lógica customizada
EmailOperator	Envia emails	Notificações
HttpSensor	Espera uma API ficar disponível	Sincronizar com sistemas externos
PrestoToMysqlOperator	Move dados entre bancos	ETL de volumes pequenos

[Criação propria canvas]

Uma coisa importante o instrutor alertou que operadores de transferência não devem ser usados pra grandes volumes de dados o motivo é que esses operadores movem dados através do Airflow o que sobrecarrega a memória pra volumes grandes é melhor usar BashOperator pra chamar ferramentas especializadas como Spark que transferem dados diretamente entre sistemas.

Conceitos Avançados

Backfill e Catchup: Backfill serve pra reprocessar dados antigos quando você descobre um bug ou muda a lógica o Catchup vem habilitado por padrão e faz o Airflow executar automaticamente todas as instâncias que ficaram pendentes isso é útil às vezes mas se você criar uma DAG nova que não faz sentido rodar pra datas antigas tem que desabilitar o Catchup senão vai gastar recursos

Dependências temporais

Parâmetro	O que faz	Quando usar
depends_on_past	Tarefa só roda se a mesma tarefa da execução anterior funcionou	Dados que dependem dos anteriores
wait_for_downstream	Espera todas as tarefas seguintes da execução anterior terminarem	Evitar processar dados novos antes de terminar os antigos

Dependências entre DAGs dá pra sincronizar DAGs diferentes usando ExternalTaskSensor ou TriggerDagRunOperator isso é essencial quando vários pipelines compartilham os mesmos dados por exemplo uma DAG de agregação só deve rodar depois que todas as DAGs de ingestão terminaram

Agendamento: Usa expressões cron ou valores prontos como daily ou hourly um erro comum é agendar todas as DAGs pro mesmo horário o que sobrecarrega o servidor tem que distribuir melhor os horários

Gerenciamento de Falhas: Configurar retries e retry_delay permite que o Airflow tente de novo automaticamente quando dá algum erro temporário timeout de rede, API indisponível. Isso reduz muito a necessidade de intervenção manual

---

Conclusão

O Airflow resolve o problema de orquestração de pipelines complexos através de DAGs que garantem execução ordenada paralela e monitorável a arquitetura modular com Scheduler, Executor e Metadata Database permite escalar conforme a demanda cresce.

O principal aprendizado foi entender que a escolha do Executor e a configuração de dependências impactam diretamente a integridade dos dados e a performance do sistema. Recursos como backfill e gerenciamento de falhas com retries mostram que o Airflow foi desenhado pensando em cenários reais de produção onde erros acontecem e dados precisam ser reprocessados em resumo, automatizar pipelines com Airflow é essencial porque elimina processos manuais propensos a erro e dá visibilidade total do fluxo através da interface web.

Referências

Vídeo do card: 11 - Prática: Pipelines de Dados - Airflow (I) da pasta que esta no card

Usei a documentação para explicar coisas de forma mais didatica - https://airflow.apache.org/docs/apache-airflow/2.7.1/

Todos os Insider visuais foram feitos no Canvas da minha aquina e fiz sozinho

acessei o canal do MarcLamberti para tirar algumas duvidas - https://www.youtube.com/@MarcLamberti
(foram muitos videos)