Relatório 11 - Prática: Pipelines de Dados I - Airflow (I)

Lucas Augusto Nunes de Barros

Descrição das atividades

1. O básico do Apache Airflow

Componentes base do Airflow:

- web server: é a interface gráfica do Airflow, onde os usuários podem visualizar, monitorar e interagir com os DAGs (*Directed Acyclic Graphs*), que são os fluxos de trabalho (*workflows*) definidos. Fornecendo uma visão geral das tarefas, logs, dependências e depurações.

- scheduler: é responsável por orquestrar a execução das tarefas. Ele monitora continuamente os DAGs, verificando quando as tarefas estão prontas para serem executadas e as envia para o Executor. O Scheduler garante que as tarefas sejam executadas na ordem correta. Sem o Scheduler, o Airflow não conseguiria coordenar a execução dos workflows.

- metadata database: é o banco de dados central do Airflow, onde todas as informações sobre DAGs, tarefas, execuções, conexões, variáveis e logs são armazenadas. Ele serve como a base do estado do sistema, permitindo que o Scheduler, o Web Server e outros componentes façam atualizações consistentes. Bancos de dados como PostgreSQL, MySQL ou SQLite são comumente usados para esse fim.

- executor: componente responsável por determinar como e onde as tarefas serão executadas. Ele recebe as tarefas do Scheduler e decide se as executa localmente (SequentialExecutor, LocalExecutor) ou em um ambiente distribuído (CeleryExecutor, KubernetesExecutor). O Executor atua como uma camada intermediária entre o Scheduler e os Workers, garantindo que as tarefas sejam distribuídas e executadas de forma eficiente.

- worker: é o componente que executa as tarefas de fato. Ele recebe as tarefas e executa o código definido, retornando o resultado. Em ambientes distribuídos múltiplos Workers podem estar em execução, permitindo a escalabilidade do Airflow. Cada Worker pode processar uma ou mais tarefas simultaneamente, dependendo das configurações aplicadas.

Componentes de um *Workflow*:

- Operator: é um bloco de construção do Airflow, ele define uma ação a ser executada e representa uma unidade de trabalho dentro de um DAG. Existem diferentes tipos de Operators para diferentes finalidades, como:
  - BashOperator: Executa um comando Bash.
  - PythonOperator: Executa uma função Python.
  - EmailOperator: Envia um e-mail.
  - Sensor: Espera por uma condição específica ser atendida (ex.: arquivo chegar em um diretório).

- Task: é uma instância específica de um *Operator* dentro de um DAG. Em outras palavras, quando você define um *Operator* em um DAG, ele se torna uma Task.

- TaskInstance: representa uma execução específica de uma Task em um determinado momento.

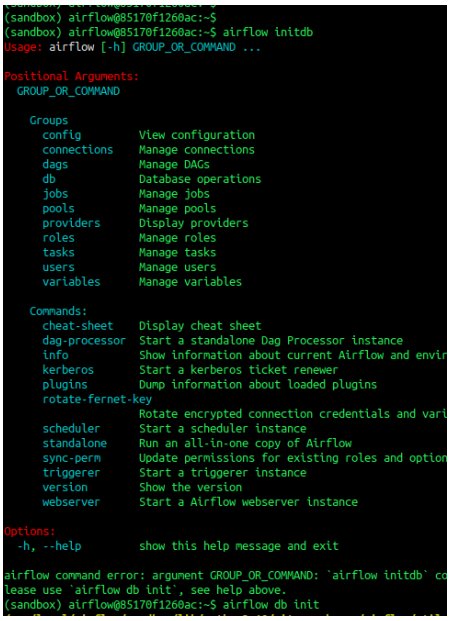
O Apache Airflow é uma plataforma de orquestração de workflows open source, projetada para programar, monitorar e gerenciar pipelines de dados de forma programática e escalável. O Airflow é amplamente utilizado em ambientes de engenharia de dados para automatizar fluxos de trabalho complexos, como ETL (Extract, Transform, Load), processamento de dados em lote, treinamento de modelos de machine learning e integração com diversas ferramentas de big data, como Apache Spark, Apache Hive, HDFS(*Hadoop Distributed File System*), entre outras.

No Apache *Airflow*, um conceito importante é o DAG - *Directed Acyclic Graph,* uma estrutura que define um fluxo de trabalho composto por tarefas e suas dependências. Cada tarefa representa uma ação a ser executada, como executar uma consulta a um banco de dados por exemplo, e as dependências determinam a ordem em que essas tarefas devem ser realizadas. O DAG é não cíclico, garantindo que as tarefas não dependam de si mesmas direta ou indiretamente. Ele pode ser agendado para execução automática em intervalos regulares e é visualizado na interface web do Airflow.

1.1 Problemas na inicialização do Docker (airflow-section-3)

Em algumas etapas da instalação foram encontrados erros devido a diferença de versão entre o software usado no curso e o implementado para a elaboração deste relatório. Na corrente instalação foram utilizadas as versões 3.10.0 do Python e 2.9.0 do Apache Airflow.

Durante a instalação do airflow o seguinte comando foi alterado no python3.10



No arquivo do docker foi necessário atualizar as versões do python e airflow.

```
# Base Image
FROM python:3.10
```

```
5  # Arguments that can be set with docker buil
6  ARG AIRFLOW_VERSION=2.9.0
7  ARG AIRFLOW_HOME=/usr/local/airflow
```

No arquivo entrypoint.sh foi necessário atualizar os comandos abaixo:

```
3  # Initiliase the metastore
4  airflow db init  # comando atualizado
5
```

```
# Initiliase the metastore
airflow db migrate  # comando atualizado
```

Durante a instalação dos primeiro teste foi necessário atualizar o pacote utilizado para a instalação do kubernets no container

pip3.10 show apache-airflow-providers-cncf-kubernetes

pip3.10 install --upgrade apache-airflow-providers-cncf-kubernetes

```
augusto@terminator:~/LAMIA/card11/airflow-materials/airflow-section-2$ pip3.10 show apache-airflow-providers-cncf-kubernetes
Name: apache-airflow-providers-cncf-kubernetes
Version: 10.1.0
Summary: Provider package apache-airflow-providers-cncf-kubernetes for Apache Airflow
Home-page:
Author:
Author-email: Apache Software Foundation <dev@airflow.apache.org>
License:
Location: /home/augusto/.local/lib/python3.10/site-packages
Requires: aiofiles, apache-airflow, asgiref, cryptography, google-re2, kubernetes, kubernetes_asyncio
Required-by:
augusto@terminator:~/LAMIA/card11/airflow-materials/airflow-section-2$
```

```
augusto@terminator:~/LAMIA/card11/airflow-materials/airflow-section-2$ airflow db migrate
DB: sqlite:////home/augusto/airflow/airflow.db
Performing upgrade to the metadata database sqlite:////home/augusto/airflow/airflow.db
[2025-02-15T12:29:26.499-0300] {migration.py:207} INFO - Context impl SQLiteImpl.
[2025-02-15T12:29:26.500-0300] {migration.py:210} INFO - Will assume non-transactional DDL.
[2025-02-15T12:29:26.502-0300] {db.py:1650} INFO - Creating tables
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
Database migrating done!
augusto@terminator:~/LAMIA/card11/airflow-materials/airflow-section-2$
```

Para criar o usuário airflow no container docker foi seguido o seguinte algoritmo:

1. Para entrar no bash do container:

   sudo docker exec -it id_container_airflow /bin/bash

2. Para criar o usuário do airflow dentro do container:

airflow users create --username admin --firstname admin --lastname admin --role Admin --email admin

```
augusto@terminator:/var/local$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS                                       NAMES
14b6e8c734d7   airflow-basic   "/entrypoint.sh"   About a minute ago   Up About a minute   0.0.0.0:8083->8080/tcp, [::]:8083->8080/tcp   vibrant_sutherland
augusto@terminator:/var/local$ sudo docker exec -it 14b6e8c734d7 /bin/bash
airflow@14b6e8c734d7:~$ airflow users  create --role Admin --username admin --email admin --firstname admin --lastname admin --password admin
/usr/local/lib/python3.10/site-packages/flask_limiter/extension.py:333 UserWarning: Using the in-memory storage for tracking rate limits as no storage was explicitly specif
ied. This is not recommended for production use. See: https://flask-limiter.readthedocs.io#configuring-a-storage-backend for documentation about configuring the storage bac
kend.
[2025-02-15T18:53:55.027+0000] {override.py:983} WARNING - No user yet created, use flask fab command to do it.
[2025-02-15T18:53:56.059+0000] {override.py:1615} INFO - Added user admin
User "admin" created with role "Admin"
airflow@14b6e8c734d7:~$
```

Alteração no código *Dockerfile*.

```
# Adding dependencies for PySpark
RUN apt-get install -y curl python3.10
#RUN update-alternatives --install /usr/bin/python python /usr/bin/python3.7 1
#RUN update-alternatives --set python /usr/bin/python3.7
RUN curl https://bootstrap.pypa.io/get-pip.py
RUN apt-get install -y python3-numpy python3-pandas python3-matplotlib python3-scipy
#python3-simpy
RUN apt-get install pip
RUN pip install simpy
```

-

Durante a instalação do airflow 2.9.0 em ./start.sh o sistema sofreu um *crash* e devido a situação foi necessário reinstalar o sistema, já aproveitando a ocasião para migrar de um HD para um SSD.

Após a reinstalação do sistema e o retorno ao ponto de executar o arquivo ./start.sh foi necessário adicionar a seguinte linha de código no dockerfile do airflow, devido a um conflito de arquivos entre diferentes versões do PyYAML.

```
RUN apt install -y pkg-config
RUN pip install --ignore-installed PyYAML==6.0.1
RUN pip install --upgrade pip && \
```

Outra problemática encontrada durante a criação dos containers foi a necessidade de adicionar uma *secret_key* válida no arquivo *docker_compose.yml*. Sem uma chave válida a imagem do airflow reinicia constantemente, de forma que o container *airflow-section-3-airflow* nunca finaliza a iniciação da forma correta.

```
    - 8080:8080
  environment:
    - AIRFLOW__WEBSERVER__SECRET_KEY=c45653c1c20f3c536f84280c8bdbe9636f231507f1d978d30c86f50801b6
  healthcheck:
    test: [ "CMD", "nc", "-z", "airflow", "8080" ]
```

Com as alterações no código, finalmente o docker compose conseguiu iniciar todos os containers.

```
(venv_airflow) augusto@terminator:~/LAMIA/novo-card11/airflow-materials/airflow-section-3$ sudo docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED         STATUS                    PORTS
                                                                                                                 NAMES
7560b24420e2   airflow-section-3-hive-webhcat     "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   10000-1000
2/tcp, 50111/tcp                                                                                                 hive-webhcat
24c481647b17   airflow-section-3-hue              "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:32
762->8888/tcp, [::]:32762->8888/tcp                                                                              hue
b60ae02b7132   airflow-section-3-hive-server      "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   10001/tcp,
 0.0.0.0:32760->10000/tcp, [::]:32760->10000/tcp, 0.0.0.0:32759->10002/tcp, [::]:32759->10002/tcp                hive-server
dd8da352713e   airflow-section-3-hive-metastore   "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   10000-1000
2/tcp, 0.0.0.0:32761->9083/tcp, [::]:32761->9083/tcp                                                             hive-metastore
9fe63c618905   airflow-section-3-livy             "./entrypoint"           3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:32
758->8998/tcp, [::]:32758->8998/tcp                                                                              livy
bb9cf5cbd35d   airflow-section-3-spark-worker     "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   10000-1000
2/tcp, 0.0.0.0:32764->8081/tcp, [::]:32764->8081/tcp                                                             airflow-sectio
n-3-spark-worker-1
c1e858699f8e   airflow-section-3-datanode         "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   9864/tcp
                                                                                                                 datanode
5f01d5292aa4   wodby/adminer:latest               "/entrypoint.sh php …"   3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:32
767->9000/tcp, [::]:32767->9000/tcp                                                                              adminer
52da536f7451   airflow-section-3-airflow          "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:80
80->8080/tcp, [::]:8080->8080/tcp, 10000-10002/tcp                                                               airflow
9f5bbde18cce   airflow-section-3-spark-master     "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   6066/tcp,
10000-10002/tcp, 0.0.0.0:32765->7077/tcp, [::]:32765->7077/tcp, 0.0.0.0:32766->8082/tcp, [::]:32766->8082/tcp   spark-master
2b3738de6132   airflow-section-3-postgres         "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:32
769->5432/tcp, [::]:32769->5432/tcp                                                                              postgres
e94f79b0e00f   airflow-section-3-namenode         "./entrypoint.sh ./s…"   3 minutes ago   Up 3 minutes (healthy)   0.0.0.0:32
763->9870/tcp, [::]:32763->9870/tcp                                                                              namenode
(venv_airflow) augusto@terminator:~/LAMIA/novo-card11/airflow-materials/airflow-section-3$
```

Feita a devida inicialização de todos os containers e a importação de todos os módulo, foi desenvolvido um pipeline para coletar taxas de câmbio de uma API externa, processar esses dados e armazená-los em um formato estruturado para posterior análise.

1.2 Operadores Airflow

Os operadores no Airflow são fundamentais para definir tarefas em uma DAG. Cada operador representa uma unidade de trabalho específica, como executar um comando Bash, chamar uma função Python, enviar um e-mail ou interagir com um banco de dados. Além disso, o Airflow permite a criação de operadores personalizados para atender a necessidades específicas. Cada operador é configurado com parâmetros que definem seu comportamento, como comandos, funções ou conexões com sistemas externos. Ao combinar operadores e definir dependências entre eles, é possível criar pipelines de dados complexos e automatizados, garantindo que cada tarefa seja executada na ordem correta e com as configurações adequadas. Abaixo é apresentado um exemplo de DAG com 4 *tasks,* cada uma dela com o campo Operador vazio.

De acordo com a documentação oficial do Airflow, os operadores mais comuns são:

- `HttpOperator`
- `MySqlOperator`
- `PostgresOperator`
- `MsSqlOperator`
- `OracleOperator`
- `JdbcOperator`
- `DockerOperator`
- `HiveOperator`
- `S3FileTransformOperator`
- `PrestoToMySqlOperator`
- `SlackAPIOperator`

2. Prática

2.1 Seção 3 : Forex Data Pipeline

A primeira etapa envolveu a utilização de sensores e operadores para desenvolver um pipeline de dados. Para essa etapa foram utilizados os seguintes sensores.

HttpSensor	Verifica se a URL está disponível antes de prosseguir com o fluxo.
FileSensor	Verifica e aguarda até que um determinado arquivo esteja disponível.

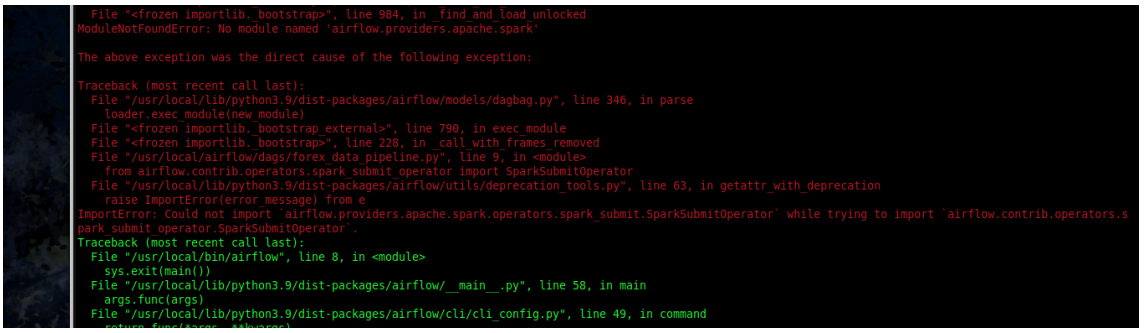
Juntamente com os seguintes operadores:

PythonOperator	Executa uma função Python
BashOperator	Executa comandos Bash
HiveOperator	Executa consultas SQL
SparkSubmitOperator	Envia tarefas para serem executadas pelo Apache Spark
EmailOperator	Faz o envio de emails
SlackAPIPostOperator	Envia mensagens através de canais Slack

Comando para testar funções foi alterado alterado para:

airflow tasks test forex_data_pipeline is_forex_rates_available 2025-02-15

Utilizando o SparkSubmitOperator para enviar dados ao cluster Spark foram encontrados erros relacionados à configuração do ambiente, como a ausência do executável do Python no sistema onde o Spark estava sendo executado. Esse problema foi resolvido garantindo que o Python estivesse instalado e configurado corretamente no ambiente do Spark, além de especificar a versão correta do Python por meio da variável de ambiente PYSPARK_PYTHON.



Erro durante importação do SparkSubmitOperator devido a mudança de versão do Apache Spark, o operador se encontra em um novo pacote que precisou ser instalado no container com o comando:

pip install apache-airflow-providers-apache-spark

Utilizando o BashOperator para executar comandos HDFS (*Hadoop Distributed File System*), como a criação de diretórios e a movimentação de arquivos. A integração com o Apache Hive utilizado para criar tabelas e realizar consultas SQL sobre os dados armazenados no HDFS facilita bastante o fluxo de trabalho, ainda mais com o Hue, que foi configurado como uma interface para facilitar a visualização e análise.

Após resolvidas essas intercorrências foi dado seguimento na seção 3 do card, com o desenvolvimento do pipeline *forex_data_pipeline* sem maiores problemas.

2.2 Seção 4 : Mastering your DAGs

2.2.1 Adaptando Códigos e Versões

No início da seção 4, foi necessário readaptar os códigos da mesma forma que foi feito para a seção 3, onde foram alteradas as versões do Python para 3.10 a do Airflow para 2.9.0 e consequentemente, do arquivo *requirements.txt* que foi atualizado de acordo com o *constraint* do Airflow 2.9.0.

Nessa seção o arquivo *Dockerfile* possuía mais dependências sendo instaladas, o que acabou causando mais alguns problemas de incompatibilidade. O

primeiro problema ocorreu durante a construção da imagem configurada em *./start.sh* pois era solicitada a instalação do pacote *netcat* que foi descontinuado. A solução foi remover a instalação do pacote e adicionar uma versão alternativa.

```
46      apt-utils \
47      curl \
48      rsync \
49      #netcat \
50      netcat-traditional \
51      locales \
```

A instalação de dois pacotes solicitados gerou conflito de dependências e devido a serem pacotes presentes do arquivo *constraint_airflow2.9.txt* (atualização do arquivo *requirements-python3.6.txt)* a instalação dos pacotes foi omitida do *Dockerfile*.

```
56      && pip install -U pip setuptoo
57      && pip install pytest \
58      #&& pip install pytz \
59      #&& pip install pyOpenSSL \
60      && pip install ndg-httpsclient
61      && pip install pyasn1 \
```

Essas alterações foram o suficiente para permitir a configuração total da imagem e do container, porém ao executar o comando *./start.sh* foi notado que apesar de criado o Airflow permanecia reiniciando constantemente, problema similar encontrado na seção 3, porém a *secret_key* já havia sido adicionada ao ambiente Airflow. Ao executar o comando *docker compose up* o terminal retornava a seguinte mensagem de erro:

```
airflow  |     import _pytest.hookspec
airflow  |   File "/usr/local/lib/python3.10/site-packages/_pytest/hookspec.py", line 305, in <module>
airflow  |     @hookspec(
airflow  | TypeError: HookspecMarker.__call__() got an unexpected keyword argument 'warn_on_impl_args'
airflow exited with code 1
airflow  | Traceback (most recent call last):
```

Após pesquisa foi constatado que o pacote *pluggy* depreciou a implementação de *warn_on_impl_args* nas versões >=1.0.0 , o que gerava um erro de dependências, pois algum dos pacotes requeria o dicionário *warn_on_impl_args* e não o encontrava nas versões mais atuais das quais o Airflow 2.9.x necessita. Sendo assim, foi necessário instalar adicionalmente a versão *pluggy==0.13.1* que ainda possuía a implementação requerida.

```
75
76  RUN pip install --ignore-installed pluggy==0.13.1
77
```

Contornado o problema das dependências, foi necessário atualizar o código ao final do *Dockerfile* para a sintaxe compatível com o Airflow 2.9.x.

```
90
91  ENTRYPOINT ["/entrypoint.sh", "webserver"]
92
```

No código *entrypoint.sh* foi necessário atualizar os comandos do Airflow:

```
71
72  case "$1" in
73    webserver)
74      airflow db upgrade
75      airflow db migrate
76      if [ "$AIRFLOW__CORE__EXECUTOR"
```

Com todas as alterações acima implementadas, o *docker compose* finalizou a inicialização do container.

```
section-4) augusto@terminator:~/LAMIA/novo-card11/airflow-materials/airflow-section-4$ sudo docker ps
sudo] senha para augusto:
ONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS
       NAMES
c2ba67a4796   airflow-section-4-airflow   "/entrypoint.sh webs…"   29 minutes ago   Up 29 minutes (healthy)
/tcp   airflow
section-4) augusto@terminator:~/LAMIA/novo-card11/airflow-materials/airflow-section-4$
```

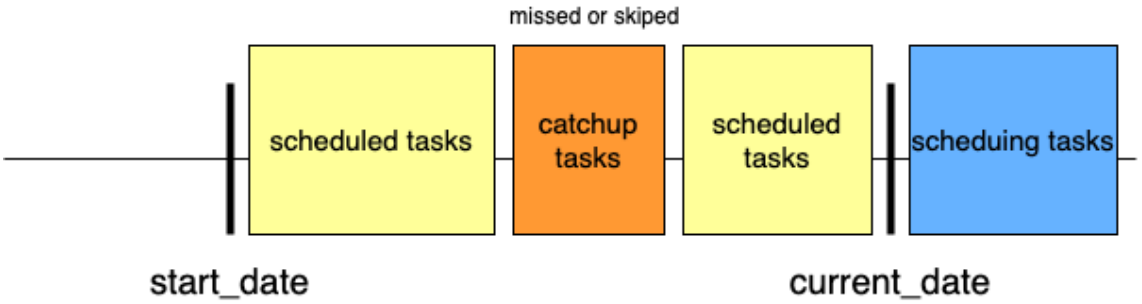
2.2.2 Start date e Schedule interval

O start_date e o schedule_interval são parâmetros fundamentais, utilizados para definir quando e com que frequência uma DAG deve ser executada. O *start_date* define a data e hora da qual a DAG deve começar a ser agendada, enquanto o *schedule_interval* determina a periodicidade das execuções.

2.2.3 Backfill e Catchup

No contexto do *Airflow, Backfill e Catchup* são conceitos relacionados à execução de DAGs em intervalos de tempo passados, especialmente quando uma DAG é reativada após um período de inatividade. O *backfill* é uma operação manual

que permite executar uma DAG para intervalos de tempo específicos no passado. Diferente do *catchup*, que é automático, o *backfill* é acionado explicitamente pelo usuário. Essa funcionalidade é útil para reprocessar dados históricos, corrigir falhas em execuções passadas ou preencher lacunas que não foram cobertas pelo *catchup*.



2.2.4 Timezones

O tratamento de *timezones* no *Airflow* é um aspecto crucial para garantir que as execuções ocorram nos horários corretos, especialmente em ambientes que abrangem múltiplos fusos horários. O *Airflow* foi projetado para lidar com *timezones*, mas requer uma configuração adequada para evitar problemas como execuções fora do horário ou inconsistências nos dados temporais. É importante sempre especificar o fuso horário para gerar objetos python do tipo *aware* e não objetos tipo *naive.*

2.2.5 Task Dependentes

A configuração *depends_on_past* é definida juntamente com a *task* e determina uma dependência entre a instância anterior ser bem sucedida e a execução da próxima instância. A tarefa intermediária não terá status definido, porém a primeira instância da *task* poderá ser executada normalmente.

Outra configuração que permite definir dependência entre *tasks* é a *wait_for_downstream*, quando essa função é habilitada em uma *task*, o Airflow verifica se todas as instâncias subsequentes dessa *task* (execuções que ocorreram após a instância atual) foram concluídas com sucesso. Caso alguma dessas instâncias subsequentes esteja pendente ou tenha falhado, a instância configurada com *wait_for_downstream* não será executada.

2.2.6 Estrutura do diretório DAG's

Manter a organização da estrutura de diretórios das DAGs é essencial para garantir um ambiente eficiente, organizado e seguro. Devido a quantidade de DAG's é preciso manter a organização para facilitar a manutenção do fluxo de trabalho, além de reduzir a chance de erros e conflitos.

Nessa etapa é apresentado o conceito de DagBag, uma coleção de Dag's estruturados na forma de árvore. Sua principal vantagem é centralizar o gerenciamento, permitindo que o Airflow carregue, valide e organize as DAGs de forma eficiente. Isso facilita a detecção de erros e otimiza o desempenho ao evitar a leitura repetida de arquivos.

2.2.7 Falhas, reexecução e alertas

Próximo ao fim do card, o curso aborda a importância de configurar mecanismos para gerenciar erros em DAGs, tanto a nível da DAG quanto a nível das *tasks*. Esses mecanismos são essenciais para garantir que falhas sejam monitoradas, tratadas e comunicadas de forma adequada, melhorando a confiabilidade e facilitando a manutenção dos pipelines de dados.

A nível de DAG algumas das funções de monitoramento de erro são:

dagrun_timeout	Define um tempo limite máximo para a execução da DAG
sla_miss_callback	Define a função de *callback* a ser executada quando uma SLA não for cumprida.
on_failure_callback	Define uma função que será acionada caso a DAG falhe.
on_success_callback	Especifica a função que será executada quando a DAG for bem-sucedida.

A nível de *task* algumas das funções de monitoramento de erro são:

Configurações de Notificação por E-mail

email	Permite configurar uma lista de e-mails que serão notificados sobre o status da *task.*
email_on_failure	Se definido como True, um e-mail será enviado automaticamente quando a tarefa falhar.
email_on_retry	Se definido como True, um e-mail será enviado sempre que a tarefa for reexecutada após uma falha

Configurações de Retry (Reexecução)

retries	Define o número máximo de tentativas de reexecução da tarefa em caso de falha
retry_delay	Especifica o tempo de espera entre as tentativas de reexecução
retry_exponential_backoff	Se definido como True, o tempo de espera entre as tentativas aumenta exponencialmente

max_retry_delay	Define o tempo máximo de espera entre as reexecuções quando o *retry_exponential_backoff* está ativado

Configurações de Timeout

execution_timeout	Define o tempo máximo permitido para a execução da tarefa.

Funções de *Callbacks*

on_failure_callback	Define a função de callback que é executada quando a tarefa falha.
on_success_callback	Define a função de callback que é executada quando a tarefa é concluída com sucesso.
on_retry_callback	Define a função de callback que é executada sempre que a tarefa for reexecutada após uma falha.

2.2.8 Testando DAG's

A biblioteca pytest é uma ferramenta poderosa para testar DAGs, ela permite a criação de testes unitários, testes de integração e testes de pipeline do tipo *end-to-end*.

Os testes unitários focam em verificar o comportamento individual de cada tarefa ou função, como validar se uma função Python retorna o resultado esperado.

Já os testes de integração avaliam a interação entre tarefas e sistemas externos, como bancos de dados ou APIs, garantindo que as dependências funcionem corretamente.

Os testes de pipeline *end-to-end* simulam a execução completa da DAG em um ambiente que replica a produção, funcionando como teste de validação. Com o pytest, é possível criar testes automatizados e integrar os testes em *pipeline*, garantindo a robustez e a confiabilidade das DAG's

Conclusão

Com as aulas foi possível compreender conceitos básicos e gerenciamento avançado de DAGs. Além disso, foram abordados a estrutura e o funcionamento de ferramentas como o Apache Airflow, Apache Spark, Docker e outras tecnologias, bem como a utilização de suas configurações para implementar pipelines robustos e eficientes.

Referências

[1] STACKOVERFLOW.COM. How to install python3-distutils on ubuntu 24.04

Disponível em: <https://stackoverflow.com/questions/78438738/how-to-install-python3-distutils-on-ubuntu-24-04> Acesso em: 27 fev. 2025

[2] GITHUB.COM. Airflow constraints file constraints-1.10.7/constraints-3.7 incorrect constraints preventing install #13045

Disponível em: <https://github.com/apache/airflow/issues/13045> Acesso em: 01 mar. 2025

[3] GITHUB.COM. docker-compose up airflow-init produces airflow command error #14365

Disponível em: <https://github.com/apache/airflow/issues/14365> Acesso em: 09 mar. 2025

[4] STACKOVERFLOW.COM. Trying to run Docker resulted in exit code 127

Disponível em: <https://stackoverflow.com/questions/27820268/trying-to-run-docker-resulted-in-exit-code-127> Acesso em: 14 mar. 2025

[5] PLUGGY.READTHEDOCS.IO. Pluggy Documentation

Disponível em: <https://pluggy.readthedocs.io/en/stable/> Acesso em: 18 mar. 2025

[6] GITHUB.COM. pluggy 1.0.0 is incompatible with pytest #9044

Disponível em: <https://github.com/pytest-dev/pytest/issues/9044> Acesso em: 18 mar. 2025

[7] AIRFLOW.APACHE.ORG. Operators

<https://airflow.apache.org/docs/apache-airflow/2.9.0/core-concepts/operators.html> Acesso em: 18 mar. 2025