Relatório para o Card - 11 - Alura - Apache AirFlow - Orquestrando seu primeiro pipeline de dados

Jefferson korte junior

Descrição da atividade:

APACHE AIRFLOW PELO ALURA:

Começamos baixando um gerenciador de máquina virtual. Utilizamos o **VirtualBox.**

E também baixamos o ubuntu.

Assim que entrei na máquina virtual (linux) ja instalei o vscode, e dentro do vscode instalei uma extensão para mexer com python. Feito isso já entrei no terminal e dei dois comandos:

```
Unset
sudo apt update
```

```
Unset
sudo apt upgrade
```

```
Unset
sudo apt install build-essential gcc make perl dkms curl tcl
```

sudo apt update – Atualiza a lista de pacotes disponíveis nos repositórios. Ele não instala ou atualiza nada, apenas obtém as informações mais recentes.

sudo apt upgrade – Instala as versões mais recentes dos pacotes que já estão no sistema, de acordo com os repositórios atualizados.

sudo apt install build-essential gcc make perl dkms curl tcl – Instala um conjunto de ferramentas essenciais para desenvolvimento e compilação:

- build-essential – Inclui compilers, como gcc, g++, e make, além de bibliotecas básicas de desenvolvimento.
- gcc – O compilador GNU C
- make – Ferramenta usada para automatizar a compilação de programas.
- perl – Linguagem de programação amplamente usada para manipulação de texto e scripts.
- dkms – Permite que módulos do kernel sejam recompilados automaticamente após atualizações do kernel.
- curl – Ferramenta de linha de comando para transferir dados via URLs.
- tcl – Linguagem de script usada em diversas aplicações.

KEY = FCEBUQSX974B4B58F2X9QFMLW

```
Unset
Link API

https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/[location]/[date1]/[date2]?key=YOUR_API_KEY
```

Para conseguirmos ter acesso a previsão do tempo na cidade de boston, vamos utilizar uma API chamada **Visual Crossing** com ela, nós podemos fazer até 1000 requisições diárias de forma gratuita e ter acesso a dados passados e futuros em relação ao clima

Após copiar o link peguei e abri o terminal novamente e rodei os seguintes comandos:

sudo apt install python3-pip -y

pip install pandas

Após criamos uma pasta chamada `DataPipeline` onde a mesma abrimos no vscode

No módulo 1 do curso eu aprendi a:

1- Utilizar uma API com dados climáticos.

2- Extrair dados da previsão do tempo

3- Criar uma pasta utilizando python

4- Salvar os arquivos extraídos em csv

Módulo 02

O'Que é airflow?: Apache Airflow é um orquestrador de fluxos, ou seja, com ele você é capaz de decidir em qual momento e em quais condições algum programa seu irá rodar. Ele é um sistema de gerenciamento de fluxo de trabalho de código aberto (open source) projetado para criar, agendar e monitorar, de forma programática, pipelines de dados e fluxos de trabalho.

Casos de uso comuns do Airflow: Pipelines ETL/ELT, Treinamento de modelos de machine learning, Geração de relatórios automatizada, backups automáticos.

DAG: É basicamente um fluxo de trabalho, um pipeline de dados definido em Python que trata-se de um conjunto de instruções que precisam ser executadas em uma determinada ordem. Ao definirmos um *data pipeline* no *Airflow*, este passa a ser um DAG.

Uma **task** ("tarefa" em português) é a unidade mais básica de um DAG e é utilizada para implementar uma determinada lógica no pipeline. Podemos afirmar, portanto, que um DAG também é um conjunto de tasks onde cada uma dessas tarefas corresponde a uma etapa que precisa ser realizada no pipeline de dados.

DAG run, ou "execução de DAG" em português, como o próprio nome sugere, trata-se da execução propriamente dita de um DAG no tempo. Este DAG run inclui algumas informações sobre a execução do DAG, entra elas podemos citar o horário e tempo de execução de cada tarefa. Em suma, trata-se da instância de um DAG no tempo.

Task instanece, ou "instância de tarefa" em português, é a execução de uma tarefa em um ponto específico do DAG. Além disso, quando trabalhamos com DAGs, interagimos principalmente com **operators** (operadores), que são os blocos de construção de um DAG. Esses operators contêm a lógica de como os dados são processados em um *data pipeline* e cada tarefa é definida justamente pela instanciação de um operador.

Arquitetura do Airflow:

O *Airflow* possui 4 componentes principais que devem estar em execução para que ele funcione corretamente. São eles o **web server**, **scheduler**, **banco de dados** e **executor**.

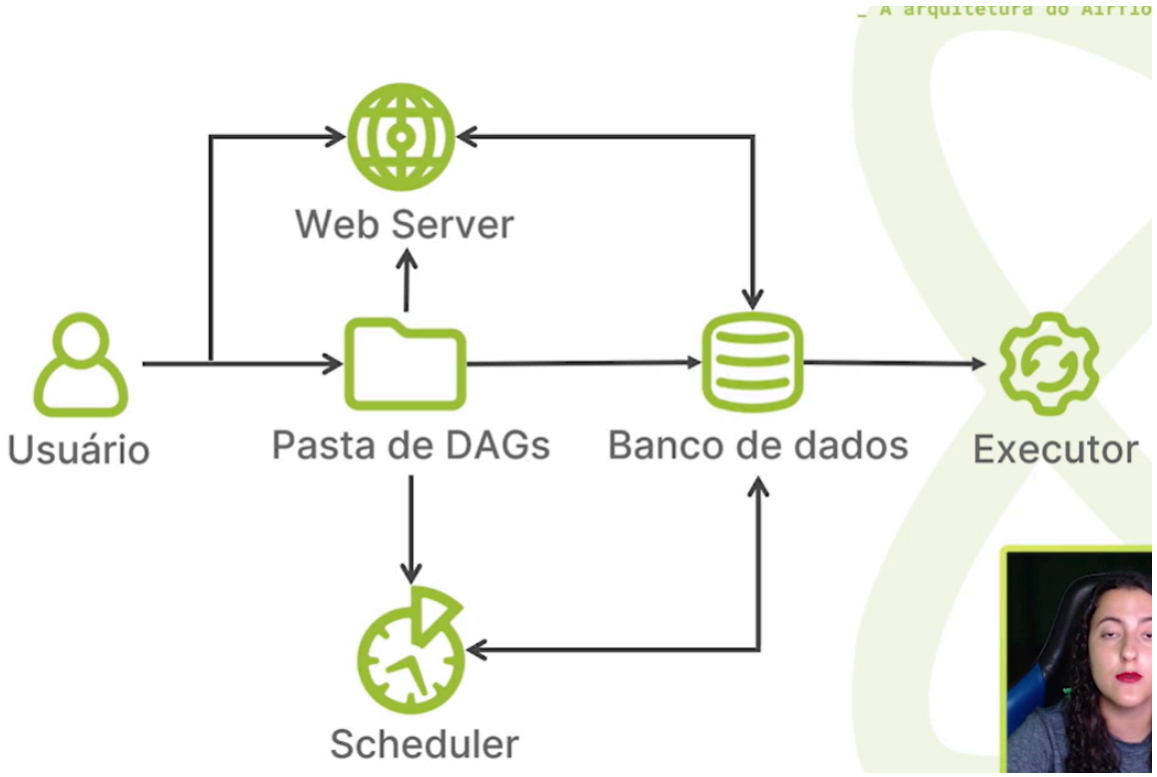
O **web server** é um servidor feito em *Flask*, um framework web Pyhton, que serve para nos apresentar a interface de usuário do *Airflow*, portanto, é por meio dele que acessamos esta interface.

Pasta de arquivos DAG: armazena os arquivos DAGs criados. Ela é lida pelo agendador e executor.

O **scheduler** ("agendador" em português) é responsável pelo agendamento da execução das tarefas dos DAGs, então ele determina quais tarefas serão realizadas, onde serão executadas e em qual ordem isso acontecerá.

O **banco de dados**, por sua vez, serve para armazenar todos os metadados referentes aos DAGs. Sendo assim, ele registra o horário em que as tarefas foram executadas, quanto tempo cada task levou para ser realizada e o estado de cada uma - se foram executadas com sucesso ou falha, e outras informações relacionadas.

Por fim, temos o **executor**, que é o mecanismo de execução das tarefas. Ou seja, ele é responsável por descobrir quais recursos serão necessários para executar essas tasks.



Apos baixar o **python 3.9** pelo terminal do linux.

INSTALANDO O AMBIENTE VIRTUAL:

Depois no terminal acessamos nosso document e criamos uma pasta chama **airflowalura**

Para acessar os nossos documentos: **cd Documents**

Para criar uma pasta dentro de **Documents** precisamos usar o **comando mkdir**

No Linux, o comando mkdir é utilizado para criar novas pastas dentro do sistema de arquivos. Ele pode ser usado de diferentes formas, dependendo do contexto. Se o usuário estiver dentro do diretório onde deseja criar a pasta, basta executar mkdir nomedapasta, e o sistema criará a pasta no local atual. Caso queira criar uma pasta em um diretório específico sem precisar mudar para ele primeiro, é possível usar um caminho absoluto, como mkdir ~/Documents/nomedapasta, onde ~ representa o diretório do usuário.

e após colocamos o seguinte comando:

sudo apt install python3.9-venv para para instalar o pacote python3.9-venv no Linux, usando o gerenciador de pacotes apt. O sudo é necessário para obter permissões de administrador e permitir a instalação. O apt install indica que você deseja instalar algo, e o python3.9-venv é o pacote responsável por permitir a criação de **ambientes virtuais no Python 3.9**. Os ambientes virtuais são úteis para separar as dependências dos projetos Python, evitando conflitos entre versões de bibliotecas instaladas no sistema. Isso ajuda a manter seu ambiente de desenvolvimento organizado e livre de problemas.

Ambientes virtuais no Python são como espaços isolados dentro do seu sistema onde você pode instalar bibliotecas e dependências sem afetar o resto do sistema. Imagine que você tem vários projetos Python, e cada um deles precisa de versões diferentes de uma mesma biblioteca. Se você instalar todas no sistema global, pode acabar criando conflitos entre versões, o que pode causar erros.

Com um ambiente virtual, você cria uma "caixa separada" para cada projeto. Dentro dessa caixa, você instala apenas as bibliotecas que aquele projeto precisa, sem interferir nas demais. Isso garante que tudo funcione corretamente e evita problemas futuros.

E depois **para ativarmos esse ambiente virtual** usamos o comando: **source venv/bin/activate ..**Porém para ativar esse ambiente virtual **precisamos** já estar **dentro** da **pasta que criamos o ambiente virtual.**

INSTALANDO O AIRFLOW:

Apos entrarmos na nossa pasta e ativarmos o nosso ambiente virtual, **vamos instalar o airflow** com o seguinte comando **: pip install "apache-airflow==2.3.2" --constraint "https://raw.githubusercontent.com/apache/airflow/constraints-2.3.2/constraints-3.9.txt"**

Após baixar o airflow rodamos o seguinte comando**: export AIRFLOW_HOME=~/Documents/airflowalura** que define uma variável de ambiente chamada AIRFLOW_HOME

AIRFLOW_HOME é o local onde o Apache Airflow guarda todos os seus **arquivos principais**, como:

- o banco de dados (airflow.db)
- os DAGs (workflows automatizados)
- os logs de execução
- o arquivo de configuração (airflow.cfg)

e também rodamos o comando: **airflow standalone**

O comando airflow standalone inicia uma instância local e simplificada do Apache Airflow, ideal para testes e aprendizado. Ele configura automaticamente o banco de dados (usando SQLite), cria um usuário administrador padrão, e inicia tanto o webserver quanto o scheduler em um único processo. Tudo isso permite que você comece a usar a interface do Airflow em http://localhost:8080 sem precisar fazer configurações manuais complexas.

Depois de tudo instalado e configurado, receberemos uma senha para entrar no airflow, podemos acessar essa senha dentro de Documents/airflowalura/e procurar por password ou algo do tipo.

E para entrar daí apenas acessamos pelo **naveg**

ador o nosso **localhot:8080** e colocamos a nossa senha e estamos dentro do airflow.

CONHECENDO A INTERFACE DO AIRFLOW

:

Logo que entramos já vemos uma seção com o nome de DAGS, porém como não criamos nenhuma DAG ainda essas são apenas DAGS de exemplos.**Sobre a coluna RUNS**: ela mostra pra gente os status de cada uma das execuções.

Sobre a coluna SCHEDULE: Serve para mostrar o intervalo de agendamento sobre cada DAG.

Sobre a coluna LAST RUN: Mostra a última data/horário do DAG run.

Sobre a coluna NEXT RUN: Mostra a próxima data que nosso DAG está agendado para ser executado.

Sobre a coluna RECENT TASKS: Mostra as tarefas do DAG. Quantas tarefas já foram executadas, quantas foram executadas com sucesso/falha e outros status.

Depois apontamos na DAG de exemplo:**example_branch_datetime_operator_2**

Conhecemos um pouco da aba GRID que podemos ver..

também conhecemos um pouco da área GRAPH onde podemos ver tipo um "Mapa" das nossas taks..

Também analisamos a aba CALENDAR que mostra os status dos DAGS RUNS em um calendário

E por último conhecemos a aba CODE que mostra o codigo por trás da DAG.

TASK DURATION: Apresenta um grafico de linhas com a duração de cada tarefa ao longo do tempo.

TAKS TRIES: Apresenta um gráfico de linhas com o número de tentativas para cada tarefa em um DAG RUN ao longo do tempo.

LANDING TIMES: Apresenta um gráfico de linhas com a hora e o dia em que cada tarefa foi iniciada ao longo do tempo.

GANTT: Apresenta um gráfico de GANTT com a duração de cada tarefa para um DAG run específico.

CRIANDO UM DAG NO AIRFLOW:

Primeiro abrimos nosso vscode, e abrimos a nossa pasta que criamos chamada **airflowalura.** Criamos uma pasta chamada **"dags"** pelo vscode e dentro dessa pasta criamos um arquivo chamado **"meu_primeiro_dag.py"**. Onde aprendemos a criar um **dag** e **definir** suas **task.**

Também aprendemos sobre os **Operator:**

EmptyOperator: Um operador “vazio” no Airflow — ou seja, uma tarefa que não faz nada quando executada.

Pra que serve:

1- Marcar o início ou fim de um fluxo de tarefas.
2- Organizar dependências entre tarefas de forma mais clara.
3- Dividir ou agrupar tarefas para facilitar o controle.

BashOperator: Um operador que executa comandos do terminal (Bash) diretamente como tarefas do Airflow.

Pra que serve:

1- Rodar scripts shell
2- Criar pastas, mover arquivos, baixar dados, ativar ambientes virtuais, etc.
3- Integrar o Airflow com qualquer ferramenta de linha de comando

PythonOperator: executa uma função Python.

KubernetesPodOperator: Executa uma imagem definida como imagem do docker em um pod do Kubernetes

SnowflakeOperator: executa uma consulta em um banco de dados Snowflake.

EmailOperator: Envia um email

Fui apresentado ao conceito de **Jinja Templates** tambem que é uma peça-chave para deixar os pipelines de dados mais inteligentes e dinâmicos. Basicamente, eles ajudam a automatizar tarefas dentro de um **DAG**.

Imagine que você tem um pipeline de ETL rodando diariamente e precisa gerar um relatório com a data da execução. Em vez de escrever isso manualmente toda dia, você pode usar **Jinja Templates** dentro de um BashOperator para deixar o processo automático.

Nesta curso aprendemos:

1 - Criar meu primeiro DAG.

2 - instanciar tarefas usando o **BashOperator**

3 - instanciar tarefas usando o **EmptyOperator**

4 - Utilizar o **Jinja Templates**

5 - Criar DAGs mais complexos

6 - Utilizar CRON Expressions para definir um intervalo de agendamento

7- Criar tarefas utilizando o pythonOperator

8 - Desenvolver uma DAG que extrai informações do tempo

Comparação entre os dois cursos

> No **primeiro curso**, há uma abordagem mais **manual**, com a criação de ambiente virtual e instalação do Airflow sem um gerenciador específico.
>
> No curso da **Alura**, o fluxo é mais estruturado, com a instalação e configuração via **VirtualBox e Ubuntu**, além de um foco maior no uso do **Apache Airflow** para orquestrar pipelines de dados.

Conclusão:

Nesse card foi apresentado Pipelines de Dados - Airflow. O Apache Airflow é uma ferramenta para orquestrar workflows, permitindo a criação e o gerenciamento de fluxos de tarefas de forma dinâmica e escalável. Usando DAGs, ele organiza as tarefas em pipelines, garantindo dependências e execução eficiente. Este material explica como estruturar DAGs, lidar com falhas, criar dependências entre tarefas e testar pipelines.

Referências:

Apache Airflow: The Hands-On Guide

Curso Alura - Orquestrando seu primeiro pipeline de dados

Curso de Docker - Youtube - **Introdução ao Docker para iniciantes | Docker Tutorial #docker**