

# Dominando o Airflow: Taskflow API e Pipelines de Dados Modernos

Por Engenharia De Dados Academy

## Introdução

O Apache Airflow revolucionou a forma como gerenciamos e orquestramos fluxos de trabalho complexos de dados. Com o lançamento da Taskflow API, o Airflow deu mais um salto evolutivo, simplificando drasticamente o desenvolvimento de DAGs (Directed Acyclic Graphs) e tornando o código mais limpo, legível e eficiente.

Neste ebook, mergulharemos fundo no mundo da Taskflow API, explorando suas vantagens, funcionalidades e melhores práticas. Veremos como essa nova abordagem transforma a criação de pipelines de dados, tornando-os mais intuitivos e poderosos.

## 1. A Evolução do Airflow: Da Abordagem Clássica à Taskflow API

O Apache Airflow surgiu como uma solução robusta para orquestração de fluxos de trabalho, permitindo que engenheiros de dados e cientistas de dados criassem, agendassem e monitorassem pipelines complexos. Inicialmente, a abordagem clássica do Airflow envolvia a criação de DAGs utilizando operadores e definindo explicitamente as dependências entre tarefas.

No entanto, com o crescimento e a evolução da comunidade, ficou claro que havia espaço para melhorias. A necessidade de um código mais limpo, uma sintaxe mais intuitiva e uma melhor gestão de dados entre tarefas levou ao desenvolvimento da Taskflow API.

A Taskflow API representa um novo paradigma na criação de DAGs no Airflow. Ela introduz o conceito de decoradores, permitindo que os desenvolvedores definam tarefas como funções Python simples, decoradas com `@task`. Isso não apenas simplifica o código, mas também facilita o compartilhamento de dados entre tarefas, eliminando a necessidade de usar explicitamente XComs.

Vamos explorar um exemplo prático para ilustrar a diferença:

```
# Abordagem Clássica
from airflow import DAG
from airflow.operators.python_operator import PythonOperator

def extract():
    # código de extração
    return {"data": "extracted_data"}

def transform(**kwargs):
    ti = kwargs['ti']
    extracted_data = ti.xcom_pull(task_ids='extract')
    # código de transformação
    return {"transformed_data": "processed_data"}

with DAG('classic_dag') as dag:
    extract_task = PythonOperator(
        task_id='extract',
        python_callable=extract
    )
    transform_task = PythonOperator(
        task_id='transform',
        python_callable=transform
    )
    extract_task >> transform_task

# Abordagem com Taskflow API
from airflow.decorators import dag, task

def taskflow_dag():
    @task()
    def extract():
        # código de extração
        return {"data": "extracted_data"}

    @task()
    def transform(extracted_data):
        # código de transformação
        return {"transformed_data": "processed_data"}

    dag = taskflow_dag()
```

Como podemos ver, a Taskflow API resulta em um código mais conciso e legível, eliminando a necessidade de gerenciar explicitamente XComs e simplificando a definição de dependências entre tarefas.

## 2. Principais Características e Vantagens da Taskflow API

A Taskflow API traz consigo uma série de características e vantagens que a tornam uma escolha atraente para desenvolvedores de pipelines de dados. Vamos explorar algumas das principais:

### 2.1 Decoradores Intuitivos

O uso de decoradores como `@task` e `@dag` torna a definição de DAGs e tarefas mais intuitiva e alinhada com o estilo de programação Python moderno. Isso reduz a curva de aprendizado e torna o código mais acessível para desenvolvedores familiarizados com Python.

### 2.2 Tipagem e Compartilhamento de Dados Simplificados

Com a Taskflow API, o compartilhamento de dados entre tarefas torna-se tão simples quanto passar argumentos para funções. Isso não apenas simplifica o código, mas também permite aproveitar a tipagem do Python para garantir a integridade dos dados passados entre tarefas.

### 2.3 Otimização de Desempenho

A Taskflow API permite uma melhor otimização do parsing de DAGs. Ao mover o código para dentro de funções decoradas, evita-se o parsing desnecessário a cada ciclo do scheduler, resultando em melhor desempenho, especialmente em ambientes com muitas DAGs.

## 2.4 Maior Flexibilidade na Definição de Dependências

A definição de dependências entre tarefas torna-se mais flexível e intuitiva. Em vez de usar operadores bit shift ( `>>` ) ou métodos

`set_upstream` / `set_downstream` , as dependências são inferidas automaticamente com base na ordem de chamada das funções.

## 2.5 Integração com Ferramentas de Desenvolvimento Modernas

A estrutura baseada em funções da Taskflow API se integra melhor com ferramentas de desenvolvimento modernas, como IDEs com suporte a autocompletar e type checking, facilitando o desenvolvimento e a manutenção de DAGs complexas.

### 3. Implementando um Pipeline de Dados com Taskflow API

Vamos explorar um exemplo prático de como implementar um pipeline de dados para extrair, processar e armazenar informações sobre o preço do Bitcoin usando a Taskflow API. Este exemplo ilustrará como podemos criar um fluxo de trabalho eficiente e legível.

```
from airflow.decorators import dag, taskimport requestsfrom datetime
import datetime@dag(    schedule_interval='@daily',
start_date=datetime(2023, 1, 1),    catchup=False)def bitcoin_etl():
@task(retries=3)    def extract_bitcoin():        response =
requests.get("https://api.coindesk.com/v1/bpi/currentprice.json").json()
return response    @task()    def process_bitcoin(response):
return {        "USD": response["bpi"]["USD"]["rate"],
"change": response["bpi"]["USD"]["rate_float"] - float(response["bpi"
["USD"]["previous_close"])    }    @task()    def
store_bitcoin(data):        print(f"O valor do Bitcoin é:
{data['USD']}")        print(f"A mudança nas últimas 24 horas foi:
{data['change']}")        # Definindo o fluxo de execução
store_bitcoin(process_bitcoin(extract_bitcoin()))# Instanciando a
DAGbitcoin_etl_dag = bitcoin_etl()
```



Neste exemplo, podemos observar várias características-chave da Taskflow API:

1. Uso do decorador `@dag` para definir a DAG.
2. Tarefas definidas como funções simples decoradas com `@task`.
3. Compartilhamento de dados entre tarefas através de retornos de função e parâmetros.
4. Definição implícita de dependências através da ordem de chamada das funções.

Esta abordagem resulta em um código mais limpo, legível e fácil de manter, permitindo que os desenvolvedores se concentrem na lógica de negócios em vez de se preocupar com a mecânica de definição de DAGs.

Ao adotar a Taskflow API em seus projetos Airflow, é importante considerar algumas melhores práticas para maximizar seus benefícios:

Mantenha suas tarefas granulares e focadas em uma única responsabilidade. Isso não apenas melhora a legibilidade, mas também facilita a reusabilidade e o paralelismo.

Aproveite as vantagens da tipagem estática do Python para definir claramente os tipos de dados passados entre tarefas. Isso melhora a robustez do código e facilita a detecção precoce de erros.

<https://player.scaleup.com.br/print-ebook/player/eadffc7fb3d56c6ae38154a36bf5eb6a72c4339?authorization=eyJhbGciOiJIUzUxMiJ9.eyJzdWI...>

### 4.3 Tratamento de Erros

Utilize os parâmetros do decorador `@task` para definir estratégias de retry e timeout adequadas para cada tarefa. Isso aumenta a resiliência do seu pipeline.

```
@task(retries=3, retry_delay=timedelta(minutes=5))def
extract data():    # Código de extração aqui
```

## 4.4 Documentação Inline

Aproveite as docstrings do Python para documentar suas tarefas e DAGs. Isso melhora a manutenibilidade e facilita a colaboração em equipe.

## 4.5 Modularização

Organize suas DAGs em módulos reutilizáveis. A Taskflow API facilita a criação de funções utilitárias que podem ser compartilhadas entre diferentes DAGs.

## 4.6 Monitoramento e Logging

Implemente logging adequado em suas tarefas para facilitar o monitoramento e a depuração. A Taskflow API permite o uso direto das funções de logging do Python dentro das tarefas.

A Taskflow API oferece recursos avançados que podem elevar ainda mais a sofisticação e eficiência de seus pipelines de dados. Vamos explorar alguns desses recursos:

TaskGroups permitem organizar tarefas relacionadas em grupos lógicos, melhorando a visualização e organização de DAGs complexas.

## 5.2 Dynamic Task Mapping

```
@task()def process_item(item):    return item * 2@task()def
get_items():    return [1, 2, 3, 4, 5]processed =
process_item.expand(item=get_items())
```


A Taskflow API pode ser combinada com operadores tradicionais do Airflow, permitindo uma transição gradual e a utilização de recursos existentes.

```
from airflow.operators.bash import BashOperator@task()def
python_task():    return "output"bash_task = BashOperator(
task_id="bash_task",    bash_command="echo {{
task_instance.xcom_pull(task_ids='python_task') }}"python_task()
>> bash task
```

A Taskflow API representa um avanço significativo na forma como construímos e gerenciamos pipelines de dados com o Apache Airflow. Ao simplificar a sintaxe, melhorar o compartilhamento de dados entre tarefas e otimizar o desempenho, ela permite que os desenvolvedores se concentrem mais na lógica de negócios e menos na mecânica de orquestração.

Neste ebook, exploramos as principais características da Taskflow API, vimos exemplos práticos de sua implementação e discutimos melhores práticas para aproveitar ao máximo seus recursos. À medida que o ecossistema Airflow continua a evoluir, a Taskflow API se posiciona como uma ferramenta fundamental para o desenvolvimento de pipelines de dados modernos, eficientes e escaláveis.

Ao adotar a Taskflow API em seus projetos, você não apenas simplificará seu código, mas também abrirá portas para novas possibilidades de automação e orquestração de dados. Continue explorando, experimentando e aproveitando o



poder desta nova abordagem para elevar seus pipelines de dados ao próximo nível.