Construindo Processos em Python com Airflow: Da Forma Clássica ao TaskFlow API

Por Engenharia De Dados Academy

Introdução

O Apache Airflow se tornou uma ferramenta essencial para orquestração e automação de fluxos de trabalho em data engineering. Neste ebook, exploraremos como construir processos robustos em Python utilizando o Airflow, com foco na transição da forma clássica de escrita para a moderna TaskFlow API. Através de um exemplo prático de extração de dados de uma API de Bitcoin, demonstraremos as melhores práticas e técnicas para criar DAGs eficientes e manuteníveis.

1. Fundamentos do Airflow e DAGs

O Apache Airflow é uma plataforma de código aberto para orquestrar fluxos de trabalho complexos. No coração do Airflow estão as DAGs (Directed Acyclic Graphs), que representam a sequência e dependências entre tarefas em um pipeline de dados.

Uma DAG é composta por:

- Tarefas individuais (operators)
- Dependências entre tarefas
- Agendamento de execução
- Parâmetros e configurações

Ao construir DAGs, temos duas abordagens principais:

1. Forma Clássica: Utilizando a sintaxe tradicional do Python
2. TaskFlow API: Uma abstração de alto nível introduzida no Airflow 2.0

Neste ebook, focaremos inicialmente na forma clássica, para então contrastar com as vantagens da TaskFlow API.

2. Construindo uma DAG na Forma Clássica

Vamos criar um processo que extrai dados de preço do Bitcoin de uma API, processa essa informação e armazena o resultado. Este exemplo ilustrará os principais conceitos da forma clássica de escrita de DAGs.

2.1 Estrutura Básica

```
from airflow import DAGfrom airflow.operators.python import
PythonOperatorfrom datetime import datetime# Definição da DAGwith
DAG('bitcoin_price_etl',         start_date=datetime(2023, 1, 1),
schedule_interval='@daily') as dag:        # Tarefas serão
definidas aqui
```

2.2 Definindo as Tarefas

Vamos criar três funções Python que representarão nossas tarefas:

```
import requestsimport loggingdef extract_bitcoin():    api_url =
"https://api.exemplo.com/bitcoin"    response =
requests.get(api_url).json()    return response['bitcoin']def
process_bitcoin(**kwargs):    ti = kwargs['ti']    bitcoin_data =
ti.xcom_pull(task_ids='extract_bitcoin_from_api')    processed_data
= {        'price': bitcoin_data['price'],        'change_24h':
bitcoin_data['change_24h']    }
ti.xcom_push(key='processed_data', value=processed_data)def
store_bitcoin(**kwargs):    ti = kwargs['ti']    processed_data =
ti.xcom_pull(task_ids='process_bitcoin_from_api',
key='processed_data')    logging.info(f"Dados processados do
Bitcoin: {processed_data}")
```

2.3 Criando os Operadores

Agora, vamos criar os operadores Python para cada uma dessas funções:

```
extract_task = PythonOperator(
task_id='extract_bitcoin_from_api',
python_callable=extract_bitcoin,    dag=dag)process_task =
PythonOperator(    task_id='process_bitcoin_from_api',
python_callable=process_bitcoin,    provide_context=True,
dag=dag)store_task = PythonOperator(
task_id='store_bitcoin_data',    python_callable=store_bitcoin,
provide_context=True,    dag=dag)
```

2.4 Definindo as Dependências

Por fim, estabelecemos a ordem de execução das tarefas:

```
extract_task >> process_task >> store_task
```

3. Compartilhando Dados entre Tarefas com XComs

XComs (Cross-Communication) é o mecanismo do Airflow para compartilhar pequenas quantidades de dados entre tarefas. Na forma clássica, o uso de XComs é explícito e requer mais código.

3.1 Enviando Dados com XCom

Na função `process_bitcoin`, usamos `ti.xcom_push()` para armazenar dados processados:

```
ti.xcom_push(key='processed_data', value=processed_data)
```

3.2 Recuperando Dados com XCom

Na função `store_bitcoin`, recuperamos os dados com `ti.xcom_pull()`:

```
processed_data = ti.xcom_pull(task_ids='process_bitcoin_from_api',
key='processed_data')
```

4. Desafios da Forma Clássica

Embora funcional, a forma clássica apresenta alguns desafios:

1. Verbosidade: Requer mais código para definir tarefas e gerenciar XComs.
2. Tipagem: Falta de tipagem estática pode levar a erros difíceis de detectar.
3. Complexidade: À medida que o DAG cresce, o gerenciamento de dependências pode se tornar complicado.
4. Testabilidade: Testar funções isoladamente pode ser desafiador devido à dependência do contexto do Airflow.

5. Introdução à TaskFlow API

A TaskFlow API, introduzida no Airflow 2.0, visa simplificar a criação de DAGs e resolver muitos dos desafios da forma clássica.

5.1 Principais Vantagens

- Sintaxe mais limpa e pythônica
- Gerenciamento automático de XComs
- Melhor suporte para tipagem
- Facilita a criação de testes unitários

5.2 Exemplo Básico com TaskFlow API

Veja como nosso exemplo ficaria usando a TaskFlow API:

```
from airflow.decorators import dag, taskfrom datetime import
datetime@dag(start_date=datetime(2023, 1, 1),
schedule_interval='@daily')def bitcoin_price_etl_taskflow():
@task    def extract_bitcoin():        # Código de extração aqui
pass    @task    def process_bitcoin(bitcoin_data):        # Código
de processamento aqui        pass    @task    def
store_bitcoin(processed_data):        # Código de armazenamento
aqui        pass    # Definição do fluxo    bitcoin_data =
extract_bitcoin()    processed_data = process_bitcoin(bitcoin_data)
```

```
store_bitcoin(processed_data)# Instanciação da DAGbitcoin_etl_dag =
bitcoin_price_etl_taskflow()
```

6. Melhores Práticas e Considerações

Ao decidir entre a forma clássica e a TaskFlow API, considere:

1. Compatibilidade: Verifique se todos os operadores que você precisa são suportados pela TaskFlow API.
2. Complexidade do DAG: Para DAGs simples, a TaskFlow API geralmente é mais adequada.
3. Equipe: Considere a familiaridade da equipe com Python decorators e tipagem.
4. Migração: Para projetos existentes, avalie o esforço de migração versus os benefícios.

Conclusão

A evolução do Apache Airflow com a introdução da TaskFlow API representa um passo significativo na simplificação e melhoria da criação de DAGs. Enquanto a forma clássica ainda tem seu lugar, especialmente em projetos legados ou cenários específicos, a TaskFlow API oferece uma abordagem mais moderna e pythônica para o desenvolvimento de fluxos de trabalho.

Ao dominar ambas as abordagens, você estará bem equipado para criar pipelines de dados robustos e eficientes, adaptando-se às necessidades específicas de cada projeto. Lembre-se sempre de que a escolha entre a forma clássica e a TaskFlow API deve ser baseada nas características do seu projeto, na expertise da equipe e nos requisitos de manutenção a longo prazo.

À medida que o ecossistema do Airflow continua a evoluir, mantenha-se atualizado com as melhores práticas e novas funcionalidades. A flexibilidade e poder do Airflow, combinados com o seu conhecimento em Python, permitirão que você crie soluções de engenharia de dados cada vez mais sofisticadas e eficientes.