DAGs y Dependencias - Día 2

Change Status

completed 40 min

Learning Objectives

1. Aprender a definir DAGs complejos con múltiples tareas
2. Entender tipos de dependencias entre tareas
3. Comprender operadores y su configuración
4. Conocer patrones comunes de diseño de DAGs

Activities and Learning

Task 1: Estructura de un DAG Complejo (10 minutos)

Elementos de un DAG bien diseñado:

Configuración clara: Nombre, descripción, schedule
Tareas modulares: Cada tarea hace una cosa específica
Dependencias explícitas: Flujo claro de ejecución
Manejo de errores: Reintentos y alertas configurados

Ejemplo de DAG estructurado:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'data_team',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(minutes=5)
}

dag = DAG(
    'etl_pipeline_diario',
    default_args=default_args,
    description='Pipeline ETL completo diario',
    schedule_interval='@daily',
    catchup=False,
    max_active_runs=1,
    tags=['etl', 'diario', 'produccion']
)
```

Task 2: Tipos de Dependencias (10 minutos)

Formas de definir dependencias:

1. Sintaxis de flechas:

```
# Dependencias simples
tarea_a >> tarea_b  # A luego B
tarea_c << tarea_d  # D luego C (equivalente)

# Dependencias múltiples
tarea_a >> [tarea_b, tarea_c]  # A luego B y C en paralelo
[tarea_x, tarea_y] >> tarea_z  # X e Y luego Z
```

2. Método set_upstream/downstream:

```
# Más explícito para casos complejos
tarea_b.set_upstream(tarea_a)    # A antes que B
tarea_a.set_downstream(tarea_c)  # A antes que C
```

Patrones comunes:

Secuencial: A → B → C
Paralelo: A → [B, C] → D
Diamond: A → [B, C] → D
Fan-out/fan-in: Uno → muchos → uno

Task 3: Operadores Principales (10 minutos)

Tipos de operadores en Airflow:

1. PythonOperator: Ejecuta funciones Python

```
def procesar_datos(**context):
    print("Procesando datos...")
    return "Completado"

tarea_python = PythonOperator(
    task_id='procesar_datos',
    python_callable=procesar_datos,
    provide_context=True,
    dag=dag
)
```

2. BashOperator: Ejecuta comandos shell

```
tarea_bash = BashOperator(
    task_id='limpiar_archivos',
    bash_command='rm -f /tmp/*.tmp',
    dag=dag
)
```

3. DummyOperator: Para estructura del grafo

```
from airflow.operators.dummy import DummyOperator

inicio = DummyOperator(task_id='inicio', dag=dag)
fin = DummyOperator(task_id='fin', dag=dag)
```