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

Practical exercise to apply the concepts learned.

Ejercicio: Construir DAG con dependencias complejas

DAG de procesamiento de ventas:

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

def extraer_ventas():
    """Simular extracción de datos de ventas"""
    print("Extrayendo datos de ventas...")
    return {"registros": 1000}

def validar_datos(ventas):
    """Validar calidad de datos"""
    print(f"Validando {ventas['registros']} registros...")
    return {"validos": 950, "errores": 50}

def transformar_datos(datos):
    """Aplicar transformaciones de negocio"""
    print(f"Transformando {datos['validos']} registros válidos...")
    return {"transformados": datos['validos']}

def cargar_data_warehouse(transformados):
    """Cargar a data warehouse"""
    print(f"Cargando {transformados['transformados']} registros...")
    return {"cargados": transformados['transformados']}

def enviar_reporte(resultado):
    """Enviar reporte de ejecución"""
    print(f"Pipeline completado: {resultado['cargados']} registros procesados")

# Configurar DAG
dag = DAG(
    'pipeline_ventas_complejo',
    description='Pipeline ETL de ventas con dependencias complejas',
    schedule_interval='@daily',
    start_date=datetime(2024, 1, 1),
    catchup=False,
    default_args={
        'retries': 2,
        'retry_delay': timedelta(minutes=5)
    }
)

# Tareas de extracción (pueden ejecutarse en paralelo)
extraer_api = PythonOperator(
    task_id='extraer_api_ventas',
    python_callable=extraer_ventas,
    dag=dag
)

extraer_db = PythonOperator(
    task_id='extraer_db_productos',
    python_callable=lambda: {"productos": 500},
    dag=dag
)

# Tarea de preparación
preparar_entorno = BashOperator(
    task_id='preparar_entorno',
    bash_command='mkdir -p /tmp/etl_ventas',
    dag=dag
)
```

```
# Tareas de validación (dependen de extracción)
validar_api = PythonOperator(
    task_id='validar_datos_api',
    python_callable=lambda: validar_datos({"registros": 1000}),
    dag=dag
)

validar_db = PythonOperator(
    task_id='validar_datos_db',
    python_callable=lambda: {"productos_validos": 480},
    dag=dag
)

# Tareas de transformación (dependen de validación)
transformar_ventas = PythonOperator(
    task_id='transformar_ventas',
    python_callable=lambda: transformar_datos({"validos": 950}),
    dag=dag
)

transformar_productos = PythonOperator(
    task_id='transformar_productos',
    python_callable=lambda: {"productos_transformados": 480},
    dag=dag
)

# Tarea de join (une ventas y productos)
join_datos = PythonOperator(
    task_id='join_ventas_productos',
    python_callable=lambda: {"registros_completos": 920},
    dag=dag
)

# Carga final
cargar_dw = PythonOperator(
    task_id='cargar_data_warehouse',
    python_callable=lambda: cargar_data_warehouse({"transformados": 920}),
    dag=dag
)

# Reporte final
enviar_reporte = PythonOperator(
    task_id='enviar_reporte_ejecucion',
    python_callable=lambda: enviar_reporte({"cargados": 920}),
    dag=dag
)

# Definir dependencias complejas
# Preparación inicial
preparar_entorno >> [extraer_api, extraer_db]

# Extracción → Validación
extraer_api >> validar_api
extraer_db >> validar_db

# Validación → Transformación
validar_api >> transformar_ventas
validar_db >> transformar_productos

# Transformaciones → Join
[transformar_ventas, transformar_productos] >> join_datos

# Join → Carga → Reporte
join_datos >> cargar_dw >> enviar_reporte
```

Visualizar el grafo de dependencias:

```
# Ver el DAG en Airflow Web UI
# Ir a Graph View para ver el flujo visual

# El grafo debería verse así:
# preparar_entorno → [extraer_api, extraer_db]
# extraer_api → validar_api → transformar_ventas ↘
# extraer_db → validar_db → transformar_productos ↘ → join_datos → cargar_dw → enviar_reporte
```

Probar diferentes escenarios:

```
# Para probar: airflow dags test pipeline_ventas_complejo
# Para ejecutar: airflow dags trigger pipeline_ventas_complejo
# Para ver logs: airflow tasks logs pipeline_ventas_complejo enviar_reporte_ejecucion 2024-01-01
```

Verificación: ¿Cómo decidirías entre usar PythonOperator vs BashOperator para una tarea específica? ¿Qué ventajas tiene definir dependencias explícitas en lugar de ejecutar tareas en orden secuencial?

Requerimientos:

Apache Airflow configurado
Conocimiento de operadores básicos

Familiaridad con grafos y dependencias