Pipelines Complejos y Best Practices - Día 5

Change Status

pending 40 min

Learning Objectives

1. Aprender patrones avanzados de diseño de DAGs
2. Comprender estrategias de escalado y optimización
3. Conocer mejores prácticas para producción
4. Entender testing y CI/CD para DAGs

Practical exercise to apply the concepts learned.

Ejercicio: Implementar DAG complejo con best practices

DAG con subDAGs y branching:

```
from airflow import DAG
from airflow.operators.python import PythonOperator, BranchPythonOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.task_group import TaskGroup
from datetime import datetime, timedelta
import random

def validar_calidad_datos(**context):
    """Simular validación de calidad"""
    calidad = random.choice(['alta', 'media', 'baja'])
    context['task_instance'].xcom_push(key='calidad', value=calidad)
    return calidad

def decidir_procesamiento(**context):
    """Decidir ruta basado en calidad"""
    calidad = context['task_instance'].xcom_pull(task_ids='validar_calidad', key='calidad')

    if calidad == 'alta':
        return 'procesamiento_rapido'
    else:
        return 'procesamiento_completo'

def procesamiento_rapido():
    print("Procesamiento optimizado para datos de alta calidad")
    return "Procesado rápido"

def procesamiento_completo():
    print("Procesamiento completo con validaciones adicionales")
    return "Procesado completo"

def procesamiento_pesado():
    print("Procesamiento intensivo para datos complejos")
    return "Procesado intensivo"

# Configurar DAG
dag = DAG(
    'pipeline_avanzado_complejo',
    description='Pipeline con patrones avanzados y best practices',
    schedule_interval='@daily',
    start_date=datetime(2024, 1, 1),
    catchup=False,
    default_args={
        'retries': 2,
        'retry_delay': timedelta(minutes=5),
        'execution_timeout': timedelta(hours=1)
    }
)

# Inicio del pipeline
inicio = DummyOperator(task_id='inicio', dag=dag)

# Validación inicial
validar = PythonOperator(
    task_id='validar_calidad',
    python_callable=validar_calidad_datos,
    provide_context=True,
    dag=dag
)

# Decisión de branching
decidir = BranchPythonOperator(
    task_id='decidir_ruta',
```

```
    python_callable=decidir_procesamiento,
    provide_context=True,
    dag=dag
)

# Rutas alternativas
ruta_rapida = PythonOperator(
    task_id='procesamiento_rapido',
    python_callable=procesamiento_rapido,
    dag=dag
)

ruta_completa = PythonOperator(
    task_id='procesamiento_completo',
    python_callable=procesamiento_completo,
    dag=dag
)

# Task Group para procesamiento complejo
with TaskGroup('procesamiento_pesado', dag=dag) as procesamiento_group:
    paso1 = PythonOperator(task_id='paso1', python_callable=lambda: print("Paso 1"))
    paso2 = PythonOperator(task_id='paso2', python_callable=lambda: print("Paso 2"))
    paso3 = PythonOperator(task_id='paso3', python_callable=lambda: print("Paso 3"))

    paso1 >> paso2 >> paso3

# Unión de rutas
union = DummyOperator(task_id='union_rutas', dag=dag)

# Finalización
fin = DummyOperator(task_id='fin', dag=dag)

# Definir dependencias complejas
inicio >> validar >> decidir

# Branching: decidir lleva a rutas alternativas
decidir >> [ruta_rapida, ruta_completa, procesamiento_group]

# Todas las rutas convergen en union
[ruta_rapida, ruta_completa, procesamiento_group] >> union >> fin
```

Implementar testing para DAGs:

```
import pytest
from airflow.models import DagBag
from datetime import datetime

class TestDAGComplejo:

    @pytest.fixture
    def dagbag(self):
        """Cargar DAGs para testing"""
        return DagBag(dag_folder='/path/to/dags', include_examples=False)

    def test_dag_cargado(self, dagbag):
        """Verificar que el DAG se carga correctamente"""
        dag = dagbag.get_dag('pipeline_avanzado_complejo')
        assert dag is not None
        assert len(dag.tasks) > 5

    def test_dependencias_dag(self, dagbag):
        """Verificar dependencias del DAG"""
        dag = dagbag.get_dag('pipeline_avanzado_complejo')

        # Verificar tarea inicial
        inicio = dag.get_task('inicio')
        assert len(inicio.upstream_task_ids) == 0

        # Verificar tarea final
        fin = dag.get_task('fin')
        assert len(fin.downstream_task_ids) == 0

    def test_branching_logic(self):
        """Test unitario de lógica de branching"""
        # Test directo de la función decidir_procesamiento
        # (requiere mock del context)
        pass
```

Configurar CI/CD para DAGs:

```
# .github/workflows/test-dags.yml
name: Test Airflow DAGs

on:
  push:
    paths:
      - 'dags/**'

jobs:
  test-dags:
```

```
runs-on: ubuntu-latest

steps:
- uses: actions/checkout@v2

- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.8'

- name: Install dependencies
  run: pip install apache-airflow pytest

- name: Test DAGs
  run: python -m pytest tests/dags/ -v

- name: Validate DAG syntax
  run: python -c "
  from airflow.models import DagBag
  dagbag = DagBag()
  if dagbag.import_errors:
      print('DAG import errors:', dagbag.import_errors)
      exit(1)
  print('All DAGs loaded successfully')
  "
```

Verificación: ¿Cuándo usarías SubDAGs vs TaskGroups? ¿Qué estrategias de escalado son más efectivas para diferentes tipos de carga de trabajo?

Requerimientos:

Apache Airflow con configuración avanzada
Conocimiento de patrones de diseño
Familiaridad con testing y CI/CD