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

Activities and Learning

Task 1: Patrones Avanzados de DAGs (10 minutos)

Patrones comunes para DAGs complejos:

1. SubDAGs para modularidad:

```
from airflow.operators.subdag import SubDagOperator
from airflow.utils.task_group import TaskGroup

# Agrupar tareas relacionadas
with TaskGroup('extraccion_datos', dag=dag) as extract_group:
    extraer_api = PythonOperator(task_id='api', ...)
    extraer_db = PythonOperator(task_id='db', ...)
    validar_datos = PythonOperator(task_id='validar', ...)

# Usar como una sola tarea en el DAG principal
extract_group >> procesar_datos
```

2. Dynamic DAGs:

```
def crear_dag_para_cliente(cliente):
    """Crear DAG específico para cada cliente"""
    dag = DAG(f'procesar_{cliente}', ...)

    # Lógica específica del cliente
    return dag

# Generar DAGs dinámicamente
for cliente in ['cliente_a', 'cliente_b', 'cliente_c']:
    globals()[f'dag_{cliente}'] = crear_dag_para_cliente(cliente)
```

3. Branching basado en condiciones:

```
from airflow.operators.python import BranchPythonOperator

def decidir_ruta(**context):
    """Decidir qué rama ejecutar basado en condición"""
    if context['task_instance'].xcom_pull(task_ids='validar_datos') == 'alta_calidad':
        return 'procesamiento_rapido'
    else:
        return 'procesamiento_completo'

branch_task = BranchPythonOperator(
    task_id='decidir_ruta',
    python_callable=decidir_ruta,
    dag=dag
)

branch_task >> [ruta_rapida, ruta_completa]
```

Task 2: Estrategias de Escalado (10 minutos)

Cómo manejar crecimiento en pipelines:

1. Paralelización horizontal:

```
# Procesar particiones en paralelo
particiones = [f'partition_{i}' for i in range(10)]

procesar_particiones = [
    PythonOperator(
        task_id=f'procesar_{particion}',
        python_callable=procesar_particion,
        op_kwargs={'particion': particion},
        dag=dag
    )
    for particion in particiones
]

# Unir resultados
unir_resultados = PythonOperator(
    task_id='unir_resultados',
    python_callable=unir_particiones,
    dag=dag
)

procesar_particiones >> unir_resultados
```

2. Pool de recursos:

```
# Limitar concurrencia con pools
tarea_limitada = PythonOperator(
    task_id='tarea_con_limite',
    python_callable=mi_funcion,
    pool='data_processing_pool',  # Pool definido en UI
    dag=dag
)
```

3. Escalado vertical (mejor hardware):

Workers dedicados para tareas pesadas
Memoria optimizada para procesamiento de datos
Almacenamiento rápido para I/O intensivo

Task 3: Best Practices para Producción (10 minutos)

Prácticas recomendadas:

1. Configuración robusta:

```
# Variables de entorno y configuración externa
from airflow.models import Variable

db_config = {
    'host': Variable.get('DB_HOST'),
    'user': Variable.get('DB_USER'),
    'password': Variable.get('DB_PASSWORD')
}

# Conexiones definidas en UI, no en código
conn = BaseHook.get_connection('mi_base_datos')
```

2. Manejo de errores comprehensivo:

```
default_args = {
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
    'retry_exponential_backoff': True,
    'max_retry_delay': timedelta(hours=1),
    'email_on_failure': True,
    'email_on_retry': False,
    'execution_timeout': timedelta(hours=2)
}
```

3. Logging estructurado:

```
import json
import logging

def log_estructurado(mensaje, extra_data=None):
    """Logging con estructura JSON"""
    log_entry = {
        'timestamp': datetime.now().isoformat(),
        'mensaje': mensaje,
        'dag_id': '{{ dag.dag_id }}',
        'task_id': '{{ task.task_id }}',
        'extra': extra_data or {}
```

```
    }

    logger.info(json.dumps(log_entry))
```