Operadores y Sensores - Día 3

Change Status

pending 40 min

Learning Objectives

1. Conocer operadores principales de Airflow
2. Aprender a usar sensores para esperar condiciones
3. Comprender operadores personalizados
4. Entender triggers y callbacks

Activities and Learning

Task 1: Operadores Comunes (10 minutos)

Operadores más utilizados en Airflow:

1. BashOperator - Ejecuta comandos shell

```
from airflow.operators.bash import BashOperator

tarea_bash = BashOperator(
    task_id='procesar_archivos',
    bash_command='find /data -name "*.csv" -exec wc -l {} \;',
    dag=dag
)
```

2. PythonOperator - Ejecuta funciones Python

```
from airflow.operators.python import PythonOperator

def mi_funcion(execution_date, **context):
    print(f"Ejecutando en: {execution_date}")
    return "Completado"

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

3. EmailOperator - Envía correos electrónicos

```
from airflow.operators.email import EmailOperator

tarea_email = EmailOperator(
    task_id='notificar_equipo',
    to='data@empresa.com',
    subject='Pipeline completado',
    html_content='<p>El pipeline ETL ha finalizado exitosamente.</p>',
    dag=dag
)
```

Task 2: Sensores para Esperar Condiciones (10 minutos)

¿Qué son los sensores?

Los sensores esperan hasta que se cumpla una condición antes de continuar.

Ejemplos comunes:

FileSensor - Espera archivo

```
from airflow.sensors.filesystem import FileSensor

esperar_archivo = FileSensor(
    task_id='esperar_datos_entrada',
```

```
    filepath='/data/input/datos.csv',
    poke_interval=30,  # Revisar cada 30 segundos
    timeout=3600,      # Máximo 1 hora
    dag=dag
)
```

HttpSensor - Espera respuesta HTTP

```
from airflow.sensors.http_sensor import HttpSensor

esperar_api = HttpSensor(
    task_id='esperar_api_disponible',
    http_conn_id='mi_api',
    endpoint='health',
    poke_interval=60,
    timeout=300,
    dag=dag
)
```

Task 3: Operadores Personalizados (10 minutos)

¿Por qué crear operadores personalizados?

Para encapsular lógica reutilizable y mantener código limpio.

Estructura básica:

```
from airflow.models.baseoperator import BaseOperator
from airflow.utils.decorators import apply_defaults

class MiOperadorPersonalizado(BaseOperator):

    @apply_defaults
    def __init__(self, mi_parametro, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.mi_parametro = mi_parametro

    def execute(self, context):
        self.log.info(f"Ejecutando con parámetro: {self.mi_parametro}")

        # Lógica del operador
        resultado = f"Procesado con {self.mi_parametro}"

        return resultado
```