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

Practical exercise to apply the concepts learned.

Ejercicio: Crear DAG con operadores y sensores

DAG de procesamiento con sensores:

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

def procesar_datos():
    print("Procesando datos de ventas...")
    return "Datos procesados"

def generar_reporte():
    print("Generando reporte ejecutivo...")
    return "Reporte generado"

dag = DAG(
    'pipeline_con_sensores',
    description='Pipeline que espera archivos antes de procesar',
    schedule_interval='@hourly',
    start_date=datetime(2024, 1, 1),
    catchup=False
)

# Sensor que espera archivo de entrada
esperar_datos = FileSensor(
    task_id='esperar_archivo_datos',
    filepath='/tmp/datos_ventas.csv',
    poke_interval=60,    # Revisar cada minuto
    timeout=3600,        # Máximo 1 hora
    mode='poke',         # Modo de verificación
    dag=dag
)

# Procesar datos una vez que el archivo llegue
procesar = PythonOperator(
    task_id='procesar_datos_ventas',
    python_callable=procesar_datos,
    dag=dag
)

# Generar reporte
reporte = PythonOperator(
    task_id='generar_reporte',
    python_callable=generar_reporte,
    dag=dag
)

# Limpiar archivos temporales
limpiar = BashOperator(
    task_id='limpiar_archivos',
    bash_command='rm -f /tmp/datos_ventas.csv',
    dag=dag
)

# Definir flujo: esperar → procesar → reportar → limpiar
esperar_datos >> procesar >> reporte >> limpiar
```

Crear operador personalizado:

```
from airflow.models.baseoperator import BaseOperator
from airflow.utils.decorators import apply_defaults
import pandas as pd

class ValidadorDatosOperator(BaseOperator):

    @apply_defaults
    def __init__(self, archivo_entrada, umbral_calidad=0.9, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.archivo_entrada = archivo_entrada
        self.umbral_calidad = umbral_calidad

    def execute(self, context):
        self.log.info(f"Validando archivo: {self.archivo_entrada}")

        # Leer datos
        try:
            df = pd.read_csv(self.archivo_entrada)
        except Exception as e:
            raise Exception(f"Error leyendo archivo: {e}")

        # Validaciones
        total_registros = len(df)
        registros_completos = df.dropna().shape[0]
        calidad = registros_completos / total_registros

        self.log.info(f"Calidad de datos: {calidad:.2%}")

        if calidad < self.umbral_calidad:
            raise Exception(f"Calidad insuficiente: {calidad:.2%} < {self.umbral_calidad:.2%}")

        return {
            'registros_totales': total_registros,
            'registros_validos': registros_completos,
            'calidad': calidad
        }

# Usar operador personalizado en DAG
validar_datos = ValidadorDatosOperator(
    task_id='validar_datos_ventas',
    archivo_entrada='/tmp/datos_ventas.csv',
    umbral_calidad=0.95,
    dag=dag
)

# Actualizar dependencias
esperar_datos >> validar_datos >> procesar >> reporte >> limpiar
```

Probar el DAG:

```
# Crear archivo de prueba
echo "id,nombre,ventas
1,Producto A,100
2,Producto B,200
3,Producto C,150" > /tmp/datos_ventas.csv

# Ejecutar DAG
airflow dags trigger pipeline_con_sensores

# Monitorear en web UI
```

Verificación: ¿En qué situaciones usarías un sensor en lugar de ejecutar tareas inmediatamente? ¿Cuáles son las ventajas de crear operadores personalizados?

Requerimientos:

Apache Airflow con sensores instalados
Familiaridad con operadores estándar
Conocimiento de Python para operadores personalizados