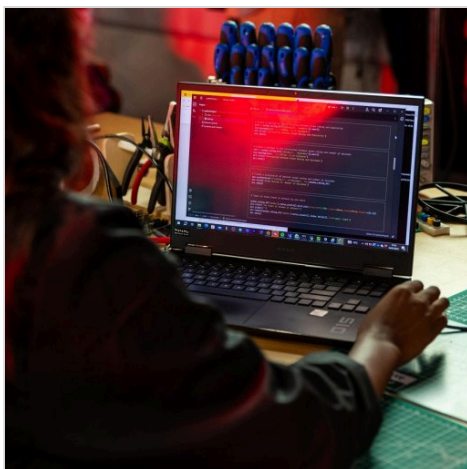


L U M B R E



Orquestación con DAGs

Grado: 6

Materia: Ingeniería de Datos · Ingeniería de Datos

11/10/2026

Antes de empezar

Un pipeline sin orquestar es una colección de scripts rezando para que se ejecuten en el orden y a la hora adecuados. Los DAGs convierten esas dependencias en grafo explícito que Airflow y similares resuelven, reintentan y vigilan.

QUÉ APRENDERÁS

- Modelar un pipeline como grafo acíclico dirigido, programando dependencias, horarios, reintentos y observabilidad para que las tareas se ejecuten en el orden correcto.

Orquestación con DAGs

El orden como grafo

Un grafo acíclico dirigido dice qué tarea depende de cuál y en qué orden. La orquestación lee ese grafo, dispara cada nodo cuando sus padres terminan, reintenta los fallos y registra todo. El orden deja de ser un acuerdo tácito entre scripts.

VIDEO

Orquestación de datos con Apache Airflow

Vídeo en español que muestra cómo definir y vigilar pipelines como grafos de tareas.



Escanea para verlo en Lumbre

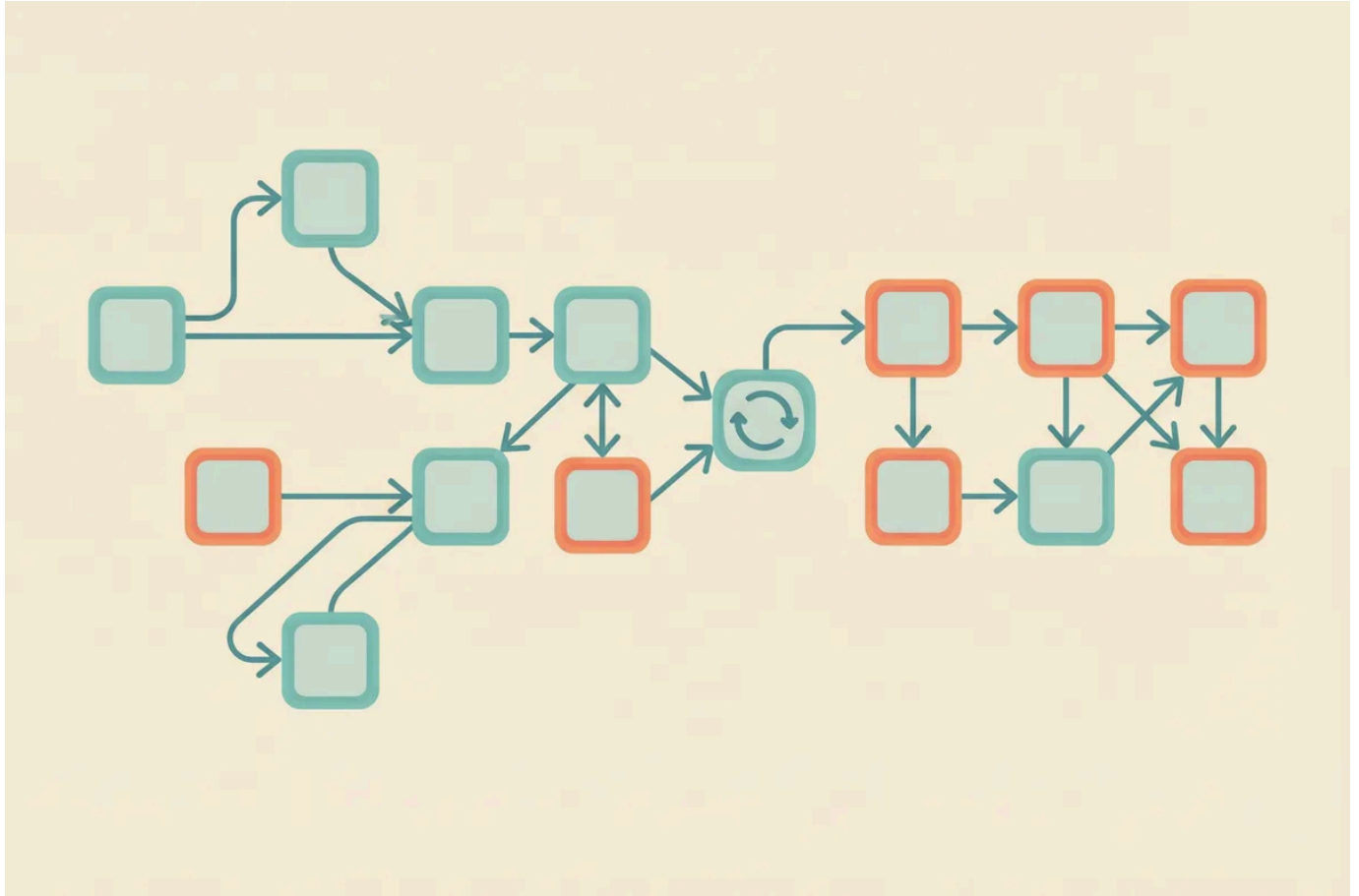
<https://www.youtube.com/watch?v=qD1UF589K6E>

Nodos y aristas

Cada tarea es un nodo; cada arista una dependencia que obliga a esperar. Acíclico significa que no hay bucles: el flujo avanza sin volver atrás, lo que garantiza que el grafo siempre se pueda ejecutar.

Programación

Los DAGs declaran su cadencia: horario tipo cron o ejecución tras la llegada de un archivo. La planificación separa cuándo debería correr de si de verdad puede, respetando dependencias y concurrencia.



El orquestador respeta las aristas, dispara cada nodo cuando sus padres terminan y reintenta el que falló.

Reintentos y alertas

Una tarea transientemente fallida se reintenta con política y espera configurables; una que agota reintentos alerta. Distinguir el error recuperable del de diseño es parte del oficio de orquestar.

Idempotencia requerida

- Reintentar obliga a que cada tarea pueda repetirse sin duplicar efectos.
- Marcar el progreso por lotes o particiones hace el reintento seguro.
- Sin idempotencia, un reintento a media carga corrompe el destino.

Dependencias entre DAGs

Cuando un flujo consume la salida de otro, se enlazan con sensores o detecciones de éxito. Bien hecho, evita leer datos incompletos; mal hecho, crea carreras entre flujos que se pisan.

Retroprocesar el pasado

La orquestación permite volver sobre ejecuciones anteriores para corregir un dato, con cuidado de rehacer solo lo afectado y en orden. Reprocesar sin control puede duplicar trabajo y romper garantías.

Observabilidad

Un panel con el estado de cada nodo, duraciones y historial convierte un pipeline opaco en algo diagnosticable. Sin métricas de latencia y fallo, corregir es adivinar.

Errores frecuentes

- Diseñar tareas que no son idempotentes y corromper el destino al reintentar a media carga.
- Encadenar DAGs leyendo la salida del otro sin un sensor, y acabar consumiendo datos incompletos.
- Lanzar un backfill sin control y duplicar trabajo o romper garantías del histórico.

Ejemplo resuelto: rescatar un nodo que falla cada madrugada

Un nodo de ingesta agota sus reintentos cada noche porque el origen aún no tiene los datos. La corrección:

1. Diagnosticar con el panel si el fallo es transitorio o de diseño, mirando duraciones e historial.
2. Añadir un sensor previo que espere a que el origen esté disponible antes de lanzar la tarea.
3. Garantizar que la escritura sea idempotente para que el reintento no duplique filas.
4. Configurar alertas solo al agotar reintentos, no en cada intento fallido.
5. Medir latencia y tasa de fallo para decidir si conviene separar la ingesta en su propio DAG.

¿Para qué sirve en la realidad?

Todo equipo con pipelines nocturnos, desde marketing hasta logística, necesita que las tareas salgan en orden y se recuperen solas. La orquestación convierte scripts frágiles en flujos observables y repetibles.

1. Una tarea de un DAG falla por un pico puntual del origen y el orquestador la reintenta con éxito a los pocos minutos. Para que ese reintento no duplique filas, la tarea debe ser:

- ☐ la más larga del grafo
- ☐ idempotente respecto a su destino
- ☐ ejecutada en paralelo
- ☐ programada con cron

2. Un DAG puede contener un ciclo siempre que el orquestador sea lo bastante avanzado.

- ☐ Verdadero
- ☐ Falso

ACTIVIDAD INTERACTIVA DISPONIBLE EN LUMBRE

3. DAG



4. Ordena la resolución de un DAG en ejecución:

Elementos:

- decidir que la ejecución corresponde a este instante
- bloquear el nodo hasta que sus padres terminen
- lanzar la tarea y reintentar si falla
- guardar estado y métricas de la ejecución

Planificar	. . .
Esperar	. . .
Ejecutar	. . .
Registrar	. . .

Apache Airflow <https://airflow.apache.org/docs/Documentación del orquestador de referencia para definir DAGs, horarios y dependencias>.

5. Une cada elemento del DAG con su función.

5.	un paso concreto del pipeline	-----
5.	la espera obligatoria entre pasos	-----
5.	disparar al llegar la hora o el archivo	-----
5.	esperar la salida de otro flujo	-----

Opciones: nodo · arista · programación · sensor

El orden deja de ser tácito

Orquestación con DAGs

Grafo

nodos y aristas

sin ciclos

Plan

cron o sensor

dependencias y concurrencia

Robustez

reintentos

idempotencia

backfill

Observar

panel de estado

métricas de fallo

El mapa mental interactivo es interactivo en la versión online.



Conceptos de DAGs en Airflow <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html> Documentación sobre cómo declarar tareas, dependencias y programación en Airflow.

6. Un nodo crítico falla cada madrugada por el origen y consumes todos los reintentos. ¿Aumentas reintentos, añades un sensor previo o separas la ingesta? Razona con idempotencia y observabilidad.

Referencia curricular

Grado:	6
Materia:	Ingeniería de Datos · Ingeniería de Datos

Versión interactiva completa (con actividades, videos y autoevaluación):



<https://univ.lumbre.lat/object/cdia-ingd-003-orquestacion-con-dags>

Material educativo generado desde Lumbre · lumbre.lat