Engineer

Flujo de procesamiento

Pasos que llevan datos desde su origen hasta su uso

Qué es

Un flujo de procesamiento, o flujo de procesamiento de datos, es un conjunto de trabajos que se ejecutan en un orden fijo y utilizan, cada uno, lo que produjo el anterior, para que los registros sin procesar de los sistemas que los crean terminen limpios, combinados y almacenados donde puedan leerlos informes, paneles u otros programas. El nombre inglés, pipeline, procede de la shell de Unix, donde `|` envía la salida de un comando al siguiente. La mayoría de los flujos se ejecutan sin intervención, según una programación o cuando llega un archivo nuevo.

Pocos flujos siguen una línea recta. Un trabajo que carga las ventas del día puede necesitar antes tanto los tipos de cambio como la lista de clientes, mientras que estos dos pueden ejecutarse en paralelo. Los trabajos y las flechas que indican «debe terminar antes» forman un grafo dirigido acíclico: cada flecha apunta en un sentido y ningún trabajo puede acabar esperando su propia finalización. Apache Airflow, una plataforma de código abierto para programar y monitorizar estos flujos de trabajo, llama Dag a cada grafo, escrito así en lugar de DAG porque su significado en Airflow ha superado la mera estructura matemática.

Un flujo por lotes se activa según una programación o un evento, procesa una porción acotada de datos, como las lecturas de contadores de un día, y se detiene. Un flujo continuo permanece en ejecución y transmite cada evento a medida que llega. Airflow se describe como una plataforma para orquestar flujos de trabajo por lotes, especialmente adecuada para trabajos con un inicio y un final claros que se ejecutan de forma programada.

El flujo más sencillo es una línea de crontab que inicia un script a las dos de la mañana. Eso basta hasta que varios scripts deben ejecutarse en orden, uno falla y alguien tiene que averiguar cuáles se ejecutaron y qué debe volver a iniciar. Un

orquestador se encarga de ese seguimiento: conserva el grafo, inicia cada trabajo cuando los que lo preceden han terminado correctamente, reintenta los que fallan, mantiene un registro por trabajo y ejecución, y anota qué periodos de datos cubrió cada ejecución. En Airflow, cada ejecución programada cubre un intervalo de datos. Una ejecución diaria para el 1 de enero comienza justo después de medianoche del día 2, una vez terminado el día, y sus trabajos pueden leer límites como `data_interval_start` para saber exactamente qué porción les corresponde.

Utilice el concepto en una decisión

Aplique «¿Qué es un flujo de procesamiento?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Las cifras llegan donde hacen falta según un calendario, sin que nadie tenga que copiar una hoja de cálculo al terminar el mes.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Un plan de infraestructura para 14 departamentos — 14 departamentos de investigación con un único plan de infraestructura

Un plan de infraestructura para 14 departamentos Póngase en contacto →

Por qué importa

Un flujo de procesamiento convierte un procedimiento en código. El orden de los trabajos, las comprobaciones entre ellos, qué ocurre cuando uno falla y con qué frecuencia se ejecuta todo pasan de la memoria de una persona a archivos que pueden revisarse, versionarse y probarse como cualquier otro software. Airflow define sus flujos de trabajo íntegramente en Python, un enfoque que llama flujos de trabajo como código, y cita el control de versiones y las pruebas entre sus ventajas.

Un fallo también pasa a tener una ubicación concreta. Cuando todo el proceso es un script largo, un fallo en el minuto cuarenta obliga a empezar de nuevo desde el minuto cero. Al dividir el proceso en trabajos cuyo estado queda registrado, una ejecución fallida muestra qué trabajo se detuvo y por qué, y solo hay que repetir ese trabajo y los posteriores. Airflow permite al operador repetir únicamente las tareas fallidas o ejecutar trabajos de periodos anteriores; así, una corrección del miércoles llega a los datos del lunes.

Repetir solo es seguro si cada trabajo está escrito para ello. La guía de buenas prácticas de Airflow pide que cada tarea se comporte como una transacción de base de datos que nunca deje resultados parciales y que «produzca el mismo resultado en cada nueva ejecución». Un trabajo construido así puede ser

reintentado por el planificador a las tres de la mañana, repetirse después de una corrección o ejecutarse de nuevo para el mes pasado, sin que nadie tenga que inspeccionar antes el destino.

El precio es hacerse cargo de más software. Un orquestador es otro servicio con su propia base de datos, actualizaciones y permisos, y cada flujo que contiene es código que necesita revisión y pruebas. Para dos scripts con una misma programación, cron y un archivo de registro pueden ser la respuesta adecuada. La justificación de un orquestador aumenta con el número de trabajos que dependen entre sí, el número de personas que necesitan ver qué se ejecutó y el coste de un informe matinal construido sobre una carga detenida a medias.

Cómo se aplica en la práctica

Ejecutar muchos flujos del mismo modo era el problema en una plataforma donde veinte flujos de datos SIG y el ETL de los datos de la aplicación se entregaban y supervisaban manualmente, con el riesgo constante de que uno se detuviera en silencio y dejara incorrectos los datos posteriores antes de que nadie lo advirtiera. Todos pasaron a una entrega automatizada, con programación, registros y tratamiento de fallos estandarizados para que cada flujo se comportara igual y un fallo fuera visible, y se integraron en la automatización de infraestructura y los informes existentes. Los veinte pasaron entonces a ejecutarse de forma automática y predecible, con datos actualizados y sin que nadie tuviera que acompañarlos manualmente.

Un flujo también puede construir algo completo con un solo comando. En una etapa documentada, un generador de CV y portafolios basado en una base de datos obtenía todos los documentos de una base SQLite de 23 tablas, construida por un flujo de 19 pasos que iba de un archivo vacío a una base de datos completa. Los pasos estaban ordenados y cada uno estaba escrito para poder repetirse con seguridad, de modo que el flujo podía ejecutarse sobre una base existente sin duplicar ninguna fila. Cuarenta y un módulos de Python, con 10.580 líneas en total, generaban después seis formatos de salida desde esa única fuente, con solo dos dependencias de ejecución. Estos recuentos describen la implementación en aquella etapa documentada.

Dentro de un trabajo, la velocidad importa tanto como el orden. Un flujo de agregación horaria de electricidad toma datos JSONL comunicados cada hora, cada 15 minutos o a intervalos irregulares, los alinea con intervalos horarios estrictos, suma la combinación de fuentes de producción, añade las

importaciones y resta las exportaciones, y produce una fila por hora y zona. Se construyó de tres formas, en Python, en JQ y en SQL sobre PostgreSQL, y cada versión se midió con datos de un día y de treinta días. La versión Python fue la más rápida, con 34 ms para un día y 180 ms para treinta, y su código se escribió para poder ampliarlo a más zonas o integrarlo en un flujo ETL.

Dónde surgen los problemas

Una nueva ejecución da una respuesta distinta. Un trabajo diario añade sus totales con `INSERT`, falla a mitad y el planificador lo reintenta, por lo que las filas escritas antes del fallo aparecen dos veces. Un trabajo que selecciona todo desde la hora actual menos un día lee una porción distinta al reintentarse que en el primer intento, y uno que lee la versión más reciente de una entrada recoge lo que haya cambiado entre ambos. La guía de buenas prácticas de Airflow nombra los tres casos: sustituir `INSERT` por un upsert, leer y escribir una partición concreta identificada por el `data_interval_start` de la ejecución y dejar `now()` fuera de cualquier cálculo crítico. Para una tabla de resúmenes horarios en PostgreSQL, una forma sencilla es sustituir todo el intervalo dentro de una sola transacción.

```
BEGIN;
DELETE FROM hourly_load
WHERE hour_start >= :'start' AND hour_start < :'end';
INSERT INTO hourly_load (zone, hour_start, mwh)
SELECT zone, date_trunc('hour', read_at), sum(mwh)
FROM reading
WHERE read_at >= :'start' AND read_at < :'end'
GROUP BY zone, date_trunc('hour', read_at);
COMMIT;
```

Ejecutado como `psql -v ON_ERROR_STOP=1 -v start=... -v end=... -f hourly_load.sql`, con el inicio y el final del intervalo de datos de la ejecución enviados por el orquestador, el trabajo borra y vuelve a escribir exactamente las filas de ese intervalo, de modo que una segunda ejecución deja la tabla igual que la primera. La forma `:'start'` es la sintaxis propia de psql para insertar una variable entrecomillada como literal SQL.

Un trabajo termina correctamente y los datos son incorrectos. Un orquestador ve si el proceso de un trabajo terminó con un código de salida satisfactorio, y eso es todo lo que ve. psql es un ejemplo claro: de forma predeterminada continúa después de un error en un script y devuelve 0 a la shell, por lo que un paso que

ejecuta `psql -f` informa de éxito aunque haya fallado una instrucción interna. Con `ON_ERROR_STOP` activado, psql se detiene en el primer error y sale con el código 3, para que el paso falle donde corresponde. Una salida correcta sigue demostrando únicamente que el programa se ejecutó; por eso la guía de Airflow propone autocomprobaciones: una tarea posterior que confirme que la partición se escribió y que los datos parecen correctos antes de que algo los lea. Para una ejecución que se queda bloqueada en lugar de fallar, las alertas Deadline Alerts, añadidas en Airflow 3.1 para sustituir la función de SLA eliminada en 3.0, activan una función de respuesta, como el envío de un correo, cuando se supera un umbral de tiempo.

Reactivar un flujo inicia una avalancha de ejecuciones. Con `catchup=True`, el planificador crea una ejecución por cada intervalo de datos omitido desde la fecha de inicio, y la documentación de Airflow añade que esa recuperación también se activa cuando un Dag se desactiva durante un tiempo y vuelve a habilitarse. Un flujo diario pausado durante un mes puede volver con treinta ejecuciones en cola contra la misma fuente. El valor predeterminado, `catchup_by_default=False`, hace que el planificador cree una ejecución solo para el intervalo más reciente. La documentación indica que un Dag cuyos trabajos leen la hora actual en lugar de su intervalo debería mantener desactivada la recuperación automática. Es mejor completar el histórico deliberadamente con `airflow backfill create`, que acepta un intervalo de fechas y un límite de ejecuciones en paralelo mediante `--max-active-runs`.

El código pesado en el nivel superior de un archivo Dag ralentiza el planificador. Airflow vuelve a analizar cada archivo Dag una y otra vez, con al menos `min_file_process_interval` segundos entre análisis, por lo que el código en el nivel superior del archivo, fuera de cualquier tarea, se ejecuta mucho más a menudo que el propio flujo. Una consulta a la base de datos, una llamada de red o una importación lenta en ese lugar consume tiempo en cada análisis, y la guía de buenas prácticas muestra un ejemplo en el que una de esas llamadas añade 1.000 segundos al análisis de un Dag. Reserve el nivel superior para declarar las tareas y su orden, y traslade las importaciones y llamadas costosas al interior de las funciones de tarea que las necesitan.

Lecturas adicionales

- Apache Airflow — Visión general del propio proyecto: flujos de trabajo como código Python, orquestación por lotes, ejecuciones de periodos anteriores y repetición solo de las tareas fallidas.

- Buenas prácticas — Tareas tratadas como transacciones, el mismo resultado en cada nueva ejecución, código de nivel superior y autocomprobaciones.
- Dags — Cómo declara un Dag su programación, tareas, dependencias y argumentos predeterminados, como los reintentos, y dónde encajan Deadline Alerts.
- Ejecuciones de Dag — Intervalos de datos, recuperación automática, ejecuciones de periodos anteriores y restablecimiento del estado de las tareas para repetirlas.
- Tareas — Dependencias entre tareas, tiempos de espera, políticas de reintento y transición de SLA a Deadline Alerts.
- psql — Variables y su entrecomillado, `ON_ERROR_STOP` y códigos de salida que lee un orquestador.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un flujo de procesamiento?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/pipeline/