Engineer

Конвеєр

послідовність кроків, що веде дані від джерела до вжитку

Що це таке

Конвеєр, повніше конвеєр даних, а в професійному мовленні часто пайплайн (від англійського pipeline), — це набір завдань, які виконуються в чітко заданому порядку, причому кожне бере те, що підготувало попереднє, аж поки сирі записи із систем, де вони виникають, не опиняться очищеними, зведеними й збереженими там, звідки їх читають звіти, дашборди чи інші програми. Назва прийшла з оболонки Unix, де символ `|` передає вивід однієї команди на вхід наступної. Більшість конвеєрів працює без нагляду: за розкладом або щойно надходить новий файл.

Рідко який конвеєр є прямою лінією. Завданню, що завантажує денні продажі, можуть спершу знадобитися і курси валют, і список клієнтів, тоді як ці два завдання цілком можуть виконуватися паралельно. Завдання разом зі стрілками «має завершитися раніше» між ними утворюють орієнтований ациклічний граф: кожна стрілка вказує в один бік, і жодне завдання не може опинитися в очікуванні самого себе. Apache Airflow, платформа з відкритим кодом для планування й моніторингу таких робочих процесів, називає кожен такий граф Dag — саме так, а не DAG, бо значення цього терміна в Airflow давно вийшло за межі суто математичної структури.

Пакетний (batch) конвеєр прокидається за розкладом чи за подією, обробляє обмежений зріз даних, наприклад показання лічильників за один день, і зупиняється. Потоковий (streaming) конвеєр працює безперервно й передає кожну подію далі, щойно вона надходить. Airflow описує себе як платформу для оркестрування пакетних робочих процесів, призначену насамперед для роботи з чітким початком і кінцем, що виконується за розкладом.

Найпростіший конвеєр — це рядок у crontab, що о другій ночі запускає скрипт. Цього досить, поки скрипти незалежні. Щойно кілька з них мають іти по черзі, а один падає, комусь доводиться з'ясовувати, що встигло виконатися і що запускати знову. Оркестратор бере цю облікову роботу на себе: він тримає граф, запускає кожне завдання, щойно успішно завершилися ті, від яких воно залежить, повторює завдання, що впали, веде журнал для кожного завдання й кожного запуску і фіксує, які періоди даних охопив кожен запуск. В Airflow кожен запуск за розкладом охоплює інтервал даних (data interval). Щоденний запуск за 1 січня починається одразу після півночі 2 січня, коли день уже минув, і його завдання можуть прочитати межі на кшталт `data_interval_start`, щоб точно знати, який зріз даних їхній.

Застосуйте поняття до рішення

Застосуйте «Що таке конвеєр?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Цифри з'являються там, де потрібні, за розкладом, і ніхто не копіює таблицю наприкінці місяця.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Один інфраструктурний план для 14 підрозділів — 14 дослідницьких підрозділів, охоплених спільним інфраструктурним планом

Один інфраструктурний план для 14 підрозділів Зв'язатися з нами →

Чому це важливо

Конвеєр перетворює процедуру на код. Порядок завдань, перевірки між ними, те, що відбувається, коли одне з них падає, і як часто все це запускається, переходять із пам'яті однієї людини у файли, які можна переглядати, зберігати в системі контролю версій і тестувати, як будь-яке інше програмне забезпечення. Airflow описує робочі процеси повністю на Python — підхід, який проєкт називає workflows as code, — і серед переваг називає контроль версій і тестування.

Збій до того ж отримує адресу. Коли весь процес — це один довгий скрипт, падіння на сороковій хвилині означає почати з нульової. Якщо ж розбити його на завдання із записаним станом, невдалий запуск показує, яке завдання зупинилося і чому, і перезапускати треба лише його та ті, що йдуть після нього. Airflow дає змогу операторові перезапустити тільки завдання, що впали, або виконати backfill для попереднього періоду, і саме так виправлення, зроблене в середу, доходить до даних за понеділок.

Повторення стає безпечним лише тоді, коли кожне завдання для цього написано. Посібник Airflow з найкращих практик вимагає, щоб кожне завдання поводилося як транзакція бази даних, яка ніколи не залишає часткових результатів, і давало однаковий результат за кожного повторного запуску. Таке завдання планувальник може повторити о третій ночі, людина може перезапустити його після виправлення, а за потреби його можна прогнати знову за минулий місяць, і ніхто не мусить спершу перевіряти цільову таблицю.

Ціна — ще одне програмне забезпечення, яке треба утримувати. Оркестратор — це окремий сервіс із власною базою даних, оновленнями й правами доступу, а кожен конвеєр у ньому — код, що потребує рев'ю й тестів. Для двох скриптів з одним розкладом чесною відповіддю можуть бути cron і файл журналу. Аргументи на користь оркестратора зростають разом із кількістю завдань, що залежать одне від одного, кількістю людей, яким треба бачити, що виконалося, і ціною ранкового звіту, побудованого на завантаженні, що зупинилося на півдорозі.

Як це виглядає на практиці

Двадцять GIS-конвеєрів даних і ETL для даних застосунку на одній платформі запускали й відстежували вручну, і весь час існував ризик, що якийсь із них непомітно зупиниться, а дані в подальших кроках виявляться хибними раніше, ніж хтось це помітить. Усі вони перейшли на автоматизоване постачання. Розклад, журналювання й обробку збоїв звели до одного стандарту, тож усі конвеєри поводилися однаково, а той, що зламався, ставав видимим, і їх включили в уже наявну автоматизацію інфраструктури та звітність. Відтоді всі двадцять запускалися самі й поводилися передбачувано, даючи актуальні дані, на які можна покластися, без ручного супроводу.

Конвеєр може також збирати щось ціле однією командою. На одному задокументованому етапі генератор CV і портфоліо на основі бази даних створював кожен документ на основі однієї бази даних SQLite із 23 таблицями, яку збирав конвеєр із 19 кроків, що проходив шлях від порожнього файлу до повної бази даних. Кроки мали сталий порядок, і кожен був написаний так, щоб його можна було безпечно повторювати, тож конвеєр можна було запускати й на вже наявній базі без дублювання жодного рядка. Далі 41 модуль Python загальним обсягом 10 580 рядків

створював із цього єдиного джерела шість вихідних форматів, причому генератор мав лише дві залежності під час виконання. Ці числа описують реалізацію на тому задокументованому етапі.

Усередині окремого завдання швидкість важить не менше за порядок. Конвеєр погодинного агрегування споживання електроенергії бере дані JSONL, що надходять щогодини, кожні 15 хвилин або в нерегулярні моменти, вирівнює їх за строгими погодинними інтервалами, підсумовує виробництво за міксом, додає імпорт і віднімає експорт, а на виході дає один рядок на годину на зону. Його реалізували трьома способами — на Python, на JQ і на SQL у PostgreSQL — і кожен варіант перевірили на даних за один день і за 30 днів. Найшвидшим виявився Python: 34 мс для одного дня і 180 мс для 30 днів, а його код можна було розширити на більше зон або вбудувати в ETL-конвеєр.

Де виникають проблеми

Повторний запуск дає іншу відповідь. Щоденне завдання дописує свої підсумки через `INSERT`, падає на півдорозі, планувальник повторює його, і рядки, записані до збою, тепер лежать двічі. Завдання, що вибирає все від поточного моменту мінус доба, під час повтору читає інший зріз, ніж за першої спроби, а завдання, що читає найновішу версію вхідних даних, підхоплює все, що змінилося в проміжку. Посібник Airflow з найкращих практик називає всі три випадки: замінити `INSERT` на upsert, читати й писати конкретну партицію, прив’язану до `data_interval_start` запуску, і тримати `now()` подалі від будь-якого критичного обчислення. Для таблиці погодинних підсумків у PostgreSQL один простий спосіб — замінювати весь інтервал однією транзакцією.

```
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;
```

Якщо запускати його як `psql -v ON_ERROR_STOP=1 -v start=... -v end=... -f hourly_load.sql`, а початок і кінець інтервалу даних запуску передає оркестратор, завдання видаляє й переписує рядки рівно цього інтервалу, тож другий запуск залишає таблицю такою самою, як перший. Форма `:'start'` — це власний синтаксис psql для підстановки змінної, взятої в лапки як SQL-літерал.

Завдання завершується зеленим, а дані хибні. Оркестратор бачить, чи процес завдання завершився чисто, і на цьому його знання закінчуються. psql — показовий приклад: за замовчуванням він продовжує роботу після помилки у скрипті й повертає оболонці 0, тож крок, що виконує `psql -f`, звітує про успіх, навіть коли одна з інструкцій усередині впала. Зі встановленою змінною `ON_ERROR_STOP` psql зупиняється на першій помилці й завершується з кодом 3, і крок падає там, де має. Але й чисте завершення доводить лише, що програма відпрацювала, тому посібник Airflow радить самоперевірки: завдання далі в графі, яке підтверджує, що партицію записано і дані виглядають правильно, перш ніж їх хтось прочитає. Для запуску, що зависає, а не падає, є Deadline Alerts, які з'явилися в Airflow 3.1 на заміну функції SLA, вилученої у 3.0. Вони запускають callback, наприклад лист електронною поштою, коли запуск перевищує часовий поріг.

Повторне ввімкнення конвеєра спричиняє лавину запусків. З `catchup=True` планувальник створює запуск для кожного інтервалу даних, пропущеного від дати початку, а документація Airflow додає, що catchup спрацьовує й тоді, коли Dag на якийсь час вимкнули, а потім увімкнули знову. Щоденний конвеєр, що місяць простояв на паузі, може повернутися з тридцятьма запусками в черзі до одного й того самого джерела. Значення за замовчуванням, `catchup_by_default=False`, змушує планувальник створювати запуск лише для останнього інтервалу, а документація каже, що Dag, чиї завдання читають поточний час замість свого інтервалу, має тримати catchup вимкненим. Історію краще заповнювати свідомо, командою `airflow backfill create`, яка приймає діапазон дат і обмеження на кількість паралельних запусків через `--max-active-runs`.

Важкий код на початку файлу з Dag гальмує планувальник. Airflow розбирає кожен файл із Dag знову і знову, щонайменше з інтервалом `min_file_process_interval` секунд між розборами, тож код на верхньому рівні файлу, поза будь-яким завданням, виконується значно частіше за сам конвеєр. Запит до бази даних, мережевий виклик чи повільний імпорт там коштують часу під час кожного розбору, і посібник з найкращих практик наводить приклад, у якому один такий виклик додає до розбору Dag ще 1000

секунд. Залишайте на верхньому рівні лише оголошення завдань і їхнього порядку, а дорогі імпорти й виклики переносьте всередину функцій завдань, яким вони потрібні.

Що почитати далі

- Apache Airflow — огляд від самого проєкту: робочі процеси як код на Python, оркестрування пакетних процесів, backfill і перезапуск лише тих завдань, що впали.
- Best Practices — завдання як транзакції, однаковий результат за кожного повторного запуску, код верхнього рівня й самоперевірки.
- Dags — як Dag оголошує свій розклад, завдання, залежності й типові аргументи на кшталт retries і де місце Deadline Alerts.
- Dag Runs — інтервали даних, catchup, backfill і очищення завдань, щоб вони виконалися знову.
- Tasks — залежності між завданнями, тайм-аути, політики повторних спроб і перехід від SLA до Deadline Alerts.
- psql — змінні та їх взяття в лапки, `ON_ERROR_STOP` і коди завершення, які читає оркестратор.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке конвеєр?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/pipeline/