**Engineer**

# Pipeline

Aufgabenfolge, die Daten von der Quelle zur Nutzung bringt

## Was es ist

Eine Pipeline, genauer eine Datenpipeline, ist eine Reihe von Aufgaben, die in fester Reihenfolge laufen. Jede übernimmt das Ergebnis der vorherigen, sodass Rohdatensätze aus den erzeugenden Systemen am Ende bereinigt, zusammengeführt und dort gespeichert sind, wo Berichte, Dashboards oder andere Programme sie lesen können. Der Name kommt aus der Unix-Shell, in der | die Ausgabe eines Befehls an den nächsten weiterreicht. Die meisten Pipelines laufen unbeaufsichtigt, nach einem Zeitplan oder sobald eine neue Datei eintrifft.

Nur wenige Pipelines bilden eine gerade Linie. Eine Aufgabe, die die Tagesumsätze lädt, braucht vielleicht zuerst sowohl die Wechselkurse als auch die Kundenliste, während diese beiden parallel geladen werden können. Die Aufgaben und die Pfeile für „muss vorher fertig sein“ dazwischen bilden einen gerichteten azyklischen Graphen: Jeder Pfeil zeigt in eine Richtung, und keine Aufgabe kann am Ende auf sich selbst warten. Apache Airflow, eine quelloffene Plattform zum Planen und Überwachen solcher Abläufe, nennt jeden Graphen Dag. Die Schreibweise lautet bewusst Dag statt DAG, weil die Bedeutung in Airflow über die reine mathematische Struktur hinausgewachsen ist.

Eine Batch-Pipeline startet nach einem Zeitplan oder bei einem Ereignis, verarbeitet einen begrenzten Datenausschnitt, etwa die Zählerstände eines Tages, und endet. Eine Streaming-Pipeline läuft weiter und reicht jedes Ereignis bei seiner Ankunft weiter. Airflow beschreibt sich als Plattform zur Orchestrierung von Batch-Abläufen, die sich am besten für zeitgesteuerte Arbeit mit eindeutigem Anfang und Ende eignet.

Die einfachste Pipeline ist eine crontab-Zeile, die morgens um zwei ein Skript startet. Das genügt, bis mehrere Skripte nacheinander laufen müssen, eines davon scheitert und jemand herausfinden muss, welche gelaufen sind und was erneut zu starten ist. Ein Orchestrator übernimmt diese Buchführung: Er speichert den Graphen, startet jede Aufgabe nach dem Erfolg ihrer Vorgänger, wiederholt fehlgeschlagene Aufgaben, führt Protokolle je Aufgabe und je Lauf und hält fest, welche Datenzeiträume jeder Lauf abgedeckt hat. In Airflow deckt jeder geplante Lauf ein Datenintervall ab. Ein Tageslauf für den 1. Januar startet kurz nach Mitternacht am 2., wenn der Tag vorbei ist. Seine Aufgaben können Grenzen wie `data_interval_start` lesen und wissen so genau, welcher Datenausschnitt ihnen gehört.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Pipeline?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Zahlen kommen termingerecht dort an, wo sie gebraucht werden, ohne dass am Monatsende jemand eine Tabelle kopieren muss.

**Unsere Leistungen entdecken** Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

**Ein Infrastrukturplan für 14 Abteilungen** Kontakt aufnehmen →

## Warum es wichtig ist

Eine Pipeline macht aus einem Verfahren Code. Die Reihenfolge der Aufgaben, die Prüfungen dazwischen, das Verhalten bei einem Fehler und die Häufigkeit der Ausführung wandern aus dem Gedächtnis einer Person in Dateien, die wie andere Software geprüft, versioniert und getestet werden können. Airflow definiert seine Abläufe vollständig in Python, nennt diesen Ansatz Workflows as Code und führt Versionsverwaltung und Tests als Vorteile an.

Auch ein Fehler bekommt einen Ort. Besteht der gesamte Ablauf aus einem langen Skript, bedeutet ein Absturz in Minute vierzig einen Neustart bei Minute null. Bei einer Aufteilung in Aufgaben mit gespeichertem Zustand zeigt ein fehlgeschlagener Lauf, welche Aufgabe aus welchem Grund angehalten hat. Nur diese Aufgabe und ihre Nachfolger müssen erneut laufen. Airflow erlaubt dem Betriebspersonal, ausschließlich fehlgeschlagene Aufgaben zu wiederholen oder Läufe für einen früheren Zeitraum nachzuholen. So erreicht eine am Mittwoch vorgenommene Korrektur auch die Daten vom Montag.

Sicher wird die Wiederholung erst, wenn jede Aufgabe dafür geschrieben ist. Airflows Leitfaden zu bewährten Verfahren fordert dazu auf, jede Aufgabe wie eine Datenbanktransaktion zu gestalten, die keine Teilergebnisse hinterlässt und „bei jeder erneuten Ausführung dasselbe Ergebnis erzeugt“. Eine so gebaute Aufgabe kann der Scheduler morgens um drei erneut versuchen, nach einer Korrektur wiederholen oder für den vergangenen Monat nochmals ausführen, ohne dass jemand zuerst das Ziel prüfen muss.

Der Preis ist zusätzliche Software, für die jemand verantwortlich ist. Ein Orchestrator ist ein weiterer Dienst mit eigener Datenbank, Aktualisierungen und Berechtigungen. Jede Pipeline darin ist Code, der Reviews und Tests braucht. Bei zwei Skripten mit einem gemeinsamen Zeitplan können cron und eine Protokolldatei die angemessene Antwort sein. Der Nutzen eines Orchestrators wächst mit der Zahl voneinander abhängiger Aufgaben, der Zahl der Personen, die die ausgeführten Schritte sehen müssen, und den Kosten eines Morgenberichts, dessen Datenladung auf halbem Weg abgebrochen ist.

## Wie es in der Praxis aussieht

Viele Pipelines auf dieselbe Weise zu betreiben, war auf einer Plattform das Problem. Dort wurden zwanzig GIS-Datenpipelines und der ETL-Ablauf für die Anwendungsdaten von Hand ausgeliefert und überwacht. Dabei bestand ständig das Risiko, dass eine unbemerkt ausfiel und nachgelagerte Daten falsch waren, bevor es jemand bemerkte. Alle wurden in eine automatisierte Auslieferung überführt. Planung, Protokollierung und Fehlerbehandlung wurden vereinheitlicht, damit jede Pipeline gleich arbeitete und ein Fehler sichtbar wurde. Alles wurde in die bestehende Infrastrukturautomatisierung und Berichterstattung eingebunden. Danach liefen alle zwanzig automatisch und vorhersehbar, mit aktuellen Daten und ohne jemanden, der sie von Hand durch die Schritte führte.

Eine Pipeline kann auch mit einem einzigen Befehl etwas Vollständiges aufbauen. In einem dokumentierten Entwicklungsstand bezog ein datenbankgestützter Generator für Lebenslauf und Portfolio jedes Dokument aus einer SQLite-Datenbank mit 23 Tabellen. Eine Pipeline aus 19 Schritten baute sie von einer leeren Datei bis zur vollständigen Datenbank auf. Die Schritte hatten eine feste Reihenfolge, und jeder war so geschrieben, dass er gefahrlos wiederholt werden konnte. Dadurch konnte die Pipeline auch gegen eine vorhandene Datenbank laufen, ohne eine Zeile zu duplizieren. Einundvierzig Python-Module mit

insgesamt 10.580 Zeilen erzeugten anschließend sechs Ausgabeformate aus dieser einen Quelle, mit nur zwei Laufzeitabhängigkeiten. Diese Zahlen beschreiben die Implementierung in jenem dokumentierten Entwicklungsstand.

Innerhalb einer einzelnen Aufgabe zählt Geschwindigkeit ebenso wie die Reihenfolge. Eine Pipeline zur stündlichen Aggregation von Stromdaten übernimmt JSONL-Daten, die stündlich, alle 15 Minuten oder in unregelmäßigen Abständen gemeldet werden. Sie richtet sie an strikten Stundenintervallen aus, summiert den Erzeugungsmix, addiert Importe, zieht Exporte ab und erzeugt pro Stunde und Zone eine Zeile. Sie wurde auf drei Arten gebaut: in Python, in JQ und in SQL auf PostgreSQL. Jede Variante wurde mit Daten für einen Tag und für dreißig Tage gemessen. Die Python-Variante war mit 34 ms für einen Tag und 180 ms für dreißig Tage am schnellsten. Ihr Code wurde so geschrieben, dass er auf weitere Zonen erweitert oder in eine ETL-Pipeline eingebunden werden konnte.

## Wo es schiefgeht

**Ein erneuter Lauf liefert ein anderes Ergebnis.** Eine tägliche Aufgabe hängt ihre Summen mit `INSERT` an, scheitert auf halbem Weg und wird vom Scheduler wiederholt. Dadurch sind die vor dem Fehler geschriebenen Zeilen nun doppelt vorhanden. Eine Aufgabe, die alles seit der aktuellen Zeit minus einem Tag abfragt, liest beim zweiten Versuch einen anderen Ausschnitt als beim ersten. Eine Aufgabe, die die neueste Version einer Eingabe liest, übernimmt alles, was sich dazwischen geändert hat. Airflows Leitfaden nennt alle drei Fälle: Ersetzen Sie `INSERT` durch ein Upsert, lesen und schreiben Sie eine bestimmte, nach dem `data_interval_start` des Laufs ausgewählte Partition und halten Sie `now()` aus jeder kritischen Berechnung heraus. Bei einer stündlichen Übersichtstabelle in PostgreSQL ist es eine einfache Möglichkeit, das gesamte Intervall innerhalb einer einzigen Transaktion zu ersetzen.

```
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;
```

Mit `psql -v ON_ERROR_STOP=1 -v start=... -v end=... -f hourly_load.sql` ausgeführt, wobei der Orchestrator Anfang und Ende des Datenintervalls übergibt, löscht und schreibt die Aufgabe genau die Zeilen dieses Intervalls neu. Ein zweiter Lauf hinterlässt die Tabelle deshalb so wie der erste. Die Form `:'start'` ist die eigene Syntax von psql, um eine Variable als quotiertes SQL-Literal einzusetzen.

**Eine Aufgabe wird grün abgeschlossen, aber die Daten sind falsch.** Ein Orchestrator sieht, ob der Prozess einer Aufgabe sauber beendet wurde, und sonst nichts. psql ist ein besonders deutliches Beispiel: Standardmäßig läuft es nach einem Fehler im Skript weiter und gibt der Shell 0 zurück. Ein Schritt mit `psql -f` meldet daher auch dann Erfolg, wenn eine Anweisung darin fehlgeschlagen ist. Mit gesetztem `ON_ERROR_STOP` hält psql beim ersten Fehler an und endet mit Code 3. So schlägt der Schritt an der richtigen Stelle fehl. Ein sauberes Ende beweist trotzdem nur, dass das Programm gelaufen ist. Deshalb empfiehlt Airflows Leitfaden Selbstprüfungen: eine nachgelagerte Aufgabe, die vor dem ersten Lesen bestätigt, dass die Partition geschrieben wurde und die Daten plausibel sind. Hängt ein Lauf, statt mit einem Fehler abzubrechen, lösen Deadline Alerts nach Überschreiten einer Zeitgrenze einen Callback aus, etwa eine E-Mail. Sie wurden in Airflow 3.1 als Ersatz für die in 3.0 entfernte SLA-Funktion eingeführt.

**Das erneute Einschalten einer Pipeline löst eine Flut von Läufen aus.** Mit `catchup=True` erzeugt der Scheduler einen Lauf für jedes seit dem Startdatum ausgelassene Datenintervall. Die Airflow-Dokumentation ergänzt, dass Catchup auch dann ausgelöst wird, wenn ein Dag eine Zeit lang ausgeschaltet und danach wieder eingeschaltet wird. Eine tägliche Pipeline, die einen Monat pausiert war, kann mit dreißig anstehenden Läufen gegen dieselbe Quelle zurückkehren. Der Standard `catchup_by_default=False` lässt den Scheduler nur einen Lauf für das neueste Intervall erzeugen. Laut Dokumentation sollte Catchup bei einem Dag ausgeschaltet bleiben, dessen Aufgaben die aktuelle Zeit statt ihres Intervalls lesen. Vergangene Zeiträume werden besser gezielt mit `airflow backfill create` nachgeholt. Der Befehl übernimmt einen Datumsbereich und über `--max-active-runs` eine Grenze für parallele Läufe.

**Aufwendiger Code auf oberster Ebene einer Dag-Datei bremst den Scheduler.** Airflow parst jede Dag-Datei immer wieder, mit mindestens `min_file_process_interval` Sekunden zwischen den Durchläufen. Code auf oberster Dateiebene außerhalb einer Aufgabe läuft deshalb viel häufiger als die Pipeline selbst. Eine Datenbankabfrage, ein Netzwerkaufruf oder ein langsamer Import dort kostet bei jedem Parsen Zeit. Der Leitfaden zeigt ein Beispiel, in dem ein solcher Aufruf das Parsen eines Dags um zusätzliche 1.000 Sekunden

verlängert. Beschränken Sie die oberste Ebene auf die Deklaration der Aufgaben und ihrer Reihenfolge. Verschieben Sie aufwendige Importe und Aufrufe in die Aufgabenfunktionen, die sie benötigen.

## Weiterführende Lektüre

- Apache Airflow — der eigene Überblick des Projekts: Workflows als Python-Code, Batch-Orchestrierung, das Nachholen vergangener Zeiträume und das Wiederholen ausschließlich fehlgeschlagener Aufgaben.
- Bewährte Verfahren — Aufgaben als Transaktionen, dasselbe Ergebnis bei jeder Wiederholung, Code auf oberster Ebene und Selbstprüfungen.
- Dags — wie ein Dag seinen Zeitplan, seine Aufgaben, Abhängigkeiten und Standardargumente wie Wiederholungen deklariert und wie Deadline Alerts hineinpassen.
- Dag-Läufe — Datenintervalle, Catchup, Backfill und das Zurücksetzen von Aufgaben, damit sie erneut laufen.
- Aufgaben — Abhängigkeiten zwischen Aufgaben, Zeitlimits, Wiederholungsregeln und der Wechsel von SLAs zu Deadline Alerts.
- psql — Variablen und ihre Quotierung, `ON_ERROR_STOP` und die Exitcodes, die ein Orchestrator liest.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

**Preise ansehen** Kontakt aufnehmen →

# Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Pipeline?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

**Unsere Leistungen entdecken**

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

**Mit Sorgfalt und Liebe gestaltet von D.**

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

**Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.**

**Gedruckt von https://engineer.company/de/glossary/pipeline/**