Engineer

Pipeline

en rekke steg som fører data fra kilde til bruk

Hva det er

En pipeline, fullt ut en datapipeline, er et sett jobber som kjører i en fast rekkefølge, der hver tar imot det jobben før den produserte, slik at rådata fra systemene som skaper dem, ender opp renset, satt sammen og lagret der rapporter, dashbord eller andre programmer kan lese dem. På norsk brukes det engelske ordet, med flertall pipelines eller pipeliner. Navnet kommer fra Unix-skallet, der | sender utdataene fra én kommando inn i den neste. De fleste pipelines kjører uten tilsyn, etter en tidsplan eller hver gang en ny fil kommer inn.

Få pipelines er en rett linje. En jobb som laster inn dagens salg, kan trenge både valutakursene og kundelisten først, mens de to kan kjøre side om side. Jobbene og pilene «må bli ferdig først» mellom dem danner en rettet asyklisk graf: hver pil peker én vei, og ingen jobb kan ende opp med å vente på seg selv. Apache Airflow, en plattform med åpen kildekode for å planlegge og overvåke slike arbeidsflyter, kaller hver graf en Dag, skrevet slik og ikke DAG, fordi betydningen i Airflow har vokst ut over den rene matematiske strukturen.

En batchpipeline våkner etter en tidsplan eller ved en hendelse, behandler et avgrenset utsnitt av data, f.eks. én dags måleravlesninger, og stopper. En strømmepipeline kjører hele tiden og sender hver hendelse videre etter hvert som den kommer. Airflow beskriver seg selv som en plattform for å orkestrere batcharbeidsflyter, best egnet til arbeid med en klar start og slutt som kjører etter en tidsplan.

Den enkleste pipelinen er en linje i crontab som starter et skript klokken to om natten. Det holder helt til flere skript må kjøre i rekkefølge, ett av dem knekker, og noen må finne ut hvilke som kjørte og hva som må startes på nytt. En orkestrator tar over den bokføringen: den holder grafen, starter hver jobb når jobbene oppstrøms for den har lyktes, prøver på nytt de som knekker, fører en

logg per jobb og per kjøring og registrerer hvilke perioder med data hver kjøring dekket. I Airflow dekker hver planlagte kjøring et dataintervall (data interval). En daglig kjøring for 1. januar starter like etter midnatt den 2., når døgnet er over, og jobbene i den kan lese grenser som `data_interval_start` for å vite nøyaktig hvilket utsnitt av data som er deres.

Bruk begrepet i en beslutning

Bruk «Hva er en pipeline?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Tall som kommer fram dit de trengs, til faste tider, uten at noen må kopiere et regneark ved månedsslutt.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

En pipeline gjør en prosedyre om til kode. Rekkefølgen på jobbene, kontrollene mellom dem, hva som skjer når én knekker, og hvor ofte alt kjører, flytter ut av hukommelsen til én person og inn i filer som kan gjennomgås, versjoneres og testes som all annen programvare. Airflow definerer arbeidsflytene sine helt i Python, en tilnærming prosjektet kaller workflows as code, arbeidsflyter som kode, og nevner versjonskontroll og testing blant fordelene.

En feil får også en adresse. Når hele prosessen er ett langt skript, betyr et krasj etter førti minutter å begynne på nytt fra minutt null. Delt opp i jobber med registrert tilstand viser en knekt kjøring hvilken jobb som stoppet og hvorfor, og bare den jobben og de etter den må kjøres igjen. Airflow lar en operatør kjøre bare de mislykkede oppgavene (tasks) på nytt, eller etterfylle (backfill) kjøringer for en tidligere periode, og det er slik en rettelse gjort på onsdag når mandagens data.

Gjentakelse blir trygg først når hver jobb er skrevet for det. Veiledningen for beste praksis i Airflow ber om at hver oppgave oppfører seg som en databasetransaksjon som aldri etterlater delvise resultater, og at den gir det samme utfallet ved hver ny kjøring; en jobb med den egenskapen kalles idempotent. En jobb bygd slik kan prøves på nytt av planleggeren klokken tre om natten, kjøres igjen etter en rettelse eller spilles av på nytt for forrige måned, uten at noen må inspisere målet først.

Prisen er programvare å eie. En orkestrator er én tjeneste til, med egen database, egne oppgraderinger og egne tilganger, og hver pipeline i den er kode som trenger gjennomgang og tester. For to skript på én tidsplan kan cron og en loggfil være det ærlige svaret. Argumentet for en orkestrator vokser med antallet jobber som er avhengige av hverandre, antallet folk som må se hva som kjørte, og hva det koster når en morgenrapport bygger på en innlasting som stoppet halvveis.

Slik ser det ut i praksis

Tre prosjekter viser pipelines fra hver sin side: å kjøre mange av dem likt, å bygge én som tåler å kjøres igjen, og å gjøre én rask.

På én plattform ble tjue GIS-datapipelines og ETL-en for applikasjonsdata levert og fulgt opp for hånd, med en konstant risiko for at én av dem stoppet i det stille og etterlot dataene lenger ned i kjeden feil før noen merket det. Alle ble lagt under automatisert leveranse, med planlegging, logging og feilhåndtering standardisert slik at alle pipelines oppførte seg likt og en som knakk, ble synlig, og de ble innlemmet i den eksisterende infrastrukturautomatiseringen og rapporteringen. Etterpå kjørte alle tjue automatisk og forutsigbart, med oppdaterte data og uten at noen måtte lose dem gjennom for hånd.

En pipeline kan også bygge noe helt med én kommando. På et dokumentert stadium i utviklingen hentet en databasedrevet generator for CV og portefølje hvert dokument fra én SQLite-database med 23 tabeller, som ble satt sammen av en pipeline i 19 trinn som gikk fra en tom fil til en komplett database. Trinnene hadde en fast rekkefølge, og hvert av dem var skrevet slik at det var trygt å gjenta, så pipelinen kunne kjøres mot en eksisterende database uten å duplisere en rad. Førtien Python-moduler, 10 580 linjer i alt, gjengav så seks utdataformater fra den ene kilden, med bare to avhengigheter under kjøring. Disse tallene beskriver implementasjonen på det dokumenterte stadiet.

Inne i én enkelt jobb betyr fart like mye som rekkefølge. En pipeline for timesaggregering av strømforbruk tar JSONL-data som rapporteres hver time, hvert 15. minutt eller på uregelmessige tidspunkter, justerer dem til strenge timesintervaller, summerer produksjonsmiksen, legger til import og trekker fra eksport, og lager én rad per time per sone. Den ble bygd på tre måter, i Python, i JQ og i SQL på PostgreSQL, og hver versjon ble benchmarket på én dag og tretti

dager med data. Python-versjonen var raskest, med 34 ms for én dag og 180 ms for tretti, og koden ble skrevet slik at den kunne utvides til flere soner eller integreres i en ETL-pipeline.

Hvor det går galt

En ny kjøring gir et annet svar. En daglig jobb legger til summene sine med `INSERT`, knekker halvveis, og planleggeren prøver igjen, så radene som ble skrevet før bruddet, ligger der nå to ganger. En jobb som velger alt siden nåtiden minus ett døgn, leser et annet utsnitt ved det nye forsøket enn ved det første, og en jobb som leser den nyeste versjonen av en inndatakilde, får med seg det som endret seg i mellomtiden. Veiledningen for beste praksis i Airflow nevner alle tre: erstatt `INSERT` med en upsert, les og skriv en bestemt partisjon med kjøringens `data_interval_start` som nøkkel, og hold `now()` unna alle kritiske beregninger. En enkel løsning for en sammendragstabell per time i PostgreSQL er å erstatte hele intervallet innenfor én transaksjon.

```
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;
```

Kjørt som `psql -v ON_ERROR_STOP=1 -v start=... -v end=... -f hourly_load.sql`, med start og slutt på kjøringens dataintervall sendt inn av orkestratoren, sletter jobben og skriver på nytt nøyaktig radene i det intervallet, så en ny kjøring etterlater tabellen slik den første gjorde. Formen `:'start'` er psqls egen syntaks for å sette inn en variabel sitert som en SQL-literal.

En jobb blir grønn, og dataene er feil. En orkestrator ser om prosessen til en jobb avsluttet rent, og det er alt den ser. Verktøyet psql er et tydelig eksempel: som standard fortsetter det etter en feil i et skript og returnerer 0 til skallet, så et trinn som kjører `psql -f`, melder suksess selv når en setning inne i det knakk. Med `ON_ERROR_STOP` satt stopper psql ved første feil og avslutter med kode 3, så trinnet feiler der det skal. En ren avslutning beviser likevel bare at programmet kjørte, og derfor foreslår Airflow-veiledningen egenkontroller: en oppgave nedstrøms som bekrefter at partisjonen ble skrevet og at dataene ser riktige ut,

før noe leser dem. For en kjøring som henger i stedet for å knekke, utløser Deadline Alerts, lagt til i Airflow 3.1 for å erstatte SLA-funksjonen som ble fjernet i 3.0, et tilbakekall (callback), f.eks. en e-post, når en kjøring passerer en tidsgrense.

En pipeline som slås på igjen, starter en flom av kjøringer. Med `catchup=True` oppretter planleggeren en kjøring for hvert dataintervall den hoppet over siden startdatoen, og Airflow-dokumentasjonen legger til at catchup også utløses når en Dag slås av en stund og så aktiveres igjen. En daglig pipeline som har stått på pause i en måned, kan komme tilbake med tretti kjøringer som venter på å treffe den samme kilden. Standardverdien, `catchup_by_default=False`, får planleggeren til å opprette en kjøring bare for det siste intervallet, og dokumentasjonen sier at en Dag der jobbene leser nåtiden i stedet for intervallet sitt, bør ha catchup slått av. Historikk fylles heller ut med vilje med `airflow backfill create`, som tar et datointervall og en grense for parallelle kjøringer gjennom `--max-active-runs`.

Tung kode øverst i en Dag-fil gjør planleggeren treg. Airflow parser hver Dag-fil igjen og igjen, med minst `min_file_process_interval` sekunder mellom hver gang, så kode på toppnivået i filen, utenfor alle oppgaver, kjører langt oftere enn selve pipelinen. En databasespørring, et nettverkskall eller en treg import der koster tid ved hver parsing, og veiledningen for beste praksis viser et eksempel der ett slikt kall gjør at en Dag bruker 1 000 sekunder ekstra på å bli parset. Hold toppnivået til å deklarere oppgavene og rekkefølgen deres, og flytt dyre importer og kall inn i oppgavefunksjonene som trenger dem.

Les mer

- Apache Airflow — prosjektets egen oversikt: arbeidsflyter som Python-kode, orkestrering av batcher, etterfylling og nye kjøringer av bare de mislykkede oppgavene.
- Best Practices — oppgaver behandlet som transaksjoner, det samme utfallet ved hver ny kjøring, kode på toppnivå og egenkontroller.
- Dags — hvordan en Dag deklarerer tidsplan, oppgaver, avhengigheter og standardargumenter som nye forsøk, og hvor Deadline Alerts passer inn.
- Dag Runs — dataintervaller, catchup, backfill og nullstilling av oppgaver så de kjører igjen.
- Tasks — avhengigheter mellom oppgaver, tidsavbrudd, regler for nye forsøk og overgangen fra SLA-er til Deadline Alerts.
- psql — variabler og siteringen av dem, `ON_ERROR_STOP` og avslutningskodene en orkestrator leser.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en pipeline?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/pipeline/