Engineer

Pipeline de traitement

Suite d'étapes menant les données de la source à l'usage

Ce que c'est

Un pipeline de traitement, ou pipeline de données, est un ensemble de tâches exécutées dans un ordre déterminé. Chacune reprend ce que la précédente a produit, de sorte que les enregistrements bruts issus des systèmes qui les créent finissent nettoyés, combinés et stockés là où des rapports, des tableaux de bord ou d'autres programmes peuvent les lire. Le nom vient du shell Unix, où | transmet la sortie d'une commande à la suivante. La plupart des pipelines fonctionnent sans intervention humaine, selon un calendrier ou à l'arrivée d'un nouveau fichier.

Peu de pipelines suivent une simple ligne droite. Une tâche qui charge les ventes du jour peut avoir besoin des taux de change et de la liste des clients au préalable, alors que ces deux tâches peuvent s'exécuter en parallèle. Les tâches et les flèches « doit se terminer avant » qui les relient forment un graphe orienté acyclique : chaque flèche pointe dans un seul sens, et aucune tâche ne peut se retrouver à attendre sa propre fin. Apache Airflow, une plateforme open source de planification et de supervision de ces flux de travail, appelle chaque graphe un Dag. Cette graphie remplace DAG parce que, dans Airflow, le sens du terme dépasse désormais la seule structure mathématique.

Un pipeline par lots se déclenche selon un calendrier ou un événement, traite une tranche de données délimitée, par exemple une journée de relevés de compteurs, puis s'arrête. Un pipeline en flux continu reste en fonctionnement et transmet chaque événement à son arrivée. Airflow se présente comme une plateforme d'orchestration de flux de travail par lots, particulièrement adaptée aux traitements planifiés ayant un début et une fin clairement définis.

Le pipeline le plus simple est une ligne de crontab qui lance un script à deux heures du matin. Cela suffit jusqu'au jour où plusieurs scripts doivent s'exécuter dans l'ordre, où l'un échoue et où quelqu'un doit déterminer lesquels ont tourné et quoi relancer. Un orchestrateur prend en charge ce suivi : il conserve le graphe, lance chaque tâche après la réussite de celles qui la précèdent, réessaie celles qui échouent, tient un journal par tâche et par exécution et consigne les périodes de données couvertes. Dans Airflow, chaque exécution planifiée couvre un intervalle de données. L'exécution quotidienne du 1er janvier commence juste après minuit le 2, une fois la journée terminée. Ses tâches peuvent lire des bornes telles que `data_interval_start` pour savoir exactement quelle tranche leur revient.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un pipeline de traitement ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des chiffres disponibles là où il faut, à la fréquence prévue, sans recopier un tableur à la fin du mois.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Un plan d'infrastructure pour 14 départements — 14 départements de recherche sur un même plan d'infrastructure

Un plan d'infrastructure pour 14 départements Contactez-nous →

Pourquoi c'est important

Un pipeline transforme une procédure en code. L'ordre des tâches, les contrôles intermédiaires, la conduite à tenir en cas d'échec et la fréquence des exécutions quittent la mémoire d'une personne pour des fichiers que l'on peut relire, versionner et tester comme tout autre logiciel. Airflow définit entièrement ses flux de travail en Python, selon l'approche dite « flux de travail sous forme de code » (workflows as code), et cite la gestion de versions et les tests parmi ses avantages.

Une défaillance devient aussi localisable. Lorsque tout le traitement tient dans un long script, un plantage à la quarantième minute impose de repartir de la minute zéro. Avec des tâches dont l'état est enregistré, une exécution en échec indique quelle tâche s'est arrêtée et pourquoi. Seules cette tâche et les suivantes doivent être relancées. Airflow permet de ne relancer que les tâches en échec ou de reprendre les exécutions d'une période antérieure : une correction apportée le mercredi peut ainsi être appliquée aux données du lundi.

La répétition n'est sûre que si chaque tâche est conçue pour cela. Le guide de bonnes pratiques d'Airflow demande à chaque tâche de se comporter comme une transaction de base de données qui ne laisse jamais de résultats partiels et de « produire le même résultat à chaque réexécution ». Une tâche ainsi conçue peut être réessayée par l'ordonnanceur à trois heures du matin, relancée après une correction ou rejouée pour le mois précédent, sans que personne ait à examiner la cible au préalable.

Le prix à payer est un logiciel à prendre en charge. Un orchestrateur est un service de plus, avec sa base de données, ses mises à niveau et ses permissions. Chaque pipeline qu'il héberge est du code qui exige une revue et des tests. Pour deux scripts suivant le même calendrier, cron et un fichier journal peuvent être la réponse raisonnable. L'intérêt d'un orchestrateur augmente avec le nombre de tâches interdépendantes, le nombre de personnes qui doivent voir ce qui a tourné et le coût d'un rapport matinal fondé sur un chargement interrompu à mi-chemin.

À quoi cela ressemble en pratique

Sur une plateforme, il fallait faire fonctionner de manière uniforme de nombreux pipelines. Vingt pipelines de données SIG et les traitements ETL des données applicatives étaient livrés et surveillés manuellement. Le risque persistait que l'un s'arrête sans bruit et fausse les données en aval avant que quelqu'un s'en aperçoive. Leur livraison a été automatisée, avec une planification, des journaux et une gestion des échecs standardisés, afin que tous se comportent de la même manière et qu'un échec soit visible. Ils ont été intégrés à l'automatisation et aux rapports d'infrastructure existants. Les vingt fonctionnaient alors automatiquement et de manière prévisible, avec des données à jour et sans accompagnement manuel.

Un pipeline peut aussi construire un ensemble complet en une seule commande. À un stade documenté de son développement, un générateur de CV et de portfolio piloté par une base de données produisait chaque document à partir d'une unique base SQLite de 23 tables, assemblée par un pipeline de 19 étapes allant d'un fichier vide à une base complète. Les étapes étaient ordonnées et chacune était conçue pour pouvoir être répétée sans risque : le pipeline pouvait donc être exécuté sur une base existante sans dupliquer une ligne. Quarante et un modules Python, totalisant 10 580 lignes, produisaient ensuite six formats de sortie à partir de cette source unique, avec seulement deux dépendances à l'exécution. Ces chiffres décrivent l'implémentation à ce stade documenté.

À l'intérieur d'une tâche, la vitesse compte autant que l'ordre. Un pipeline d'agrégation horaire de l'électricité reçoit des données JSONL relevées toutes les heures, toutes les 15 minutes ou à des moments irréguliers. Il les aligne sur des intervalles horaires stricts, additionne les composantes du mix de production, ajoute les importations, soustrait les exportations et produit une ligne par heure et par zone. Trois versions ont été réalisées, en Python, en JQ et en SQL sur PostgreSQL, puis chacune a été mesurée sur une journée et sur trente jours de données. Python était la version la plus rapide, avec 34 ms pour une journée et 180 ms pour trente. Le code était conçu pour pouvoir être étendu à davantage de zones ou intégré dans un pipeline ETL.

Où les problèmes surviennent

Une réexécution donne un résultat différent. Une tâche quotidienne ajoute ses totaux avec `INSERT`, échoue à mi-chemin, puis l'ordonnanceur la réessaie : les lignes écrites avant l'échec sont alors présentes deux fois. Une tâche qui sélectionne tout depuis l'heure courante moins un jour lit une autre tranche lors du nouvel essai. Une tâche qui lit la dernière version d'une entrée récupère ce qui a changé entre-temps. Le guide de bonnes pratiques d'Airflow cite ces trois cas : remplacer `INSERT` par un upsert, lire et écrire une partition précise dont la clé est le `data_interval_start` de l'exécution et exclure `now()` de tout calcul critique. Pour une table de synthèse horaire dans PostgreSQL, une solution simple consiste à remplacer l'intervalle entier au sein d'une transaction unique.

```
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;
```

Exécutée avec `psql -v ON_ERROR_STOP=1 -v start=... -v end=... -f hourly_load.sql`, l'orchestrateur lui transmettant les bornes de l'intervalle de données, la tâche supprime et réécrit exactement les lignes de cet intervalle. Une seconde exécution laisse donc la table dans le même état que la première. La forme `:'start'` est la syntaxe propre à psql pour insérer une variable mise entre apostrophes comme un littéral SQL.

Une tâche est au vert, mais les données sont fausses. L'orchestrateur voit si le processus d'une tâche s'est terminé normalement, et rien de plus. psql en fournit un exemple frappant : par défaut, il continue après une erreur dans un script et renvoie 0 au shell. Une étape qui lance `psql -f` annonce donc une réussite même si une instruction a échoué. Lorsque `ON_ERROR_STOP` est activé, psql s'arrête à la première erreur et se termine avec le code 3 : l'étape échoue là où elle le doit. Une fin normale prouve encore seulement que le programme a tourné. Le guide d'Airflow propose donc des autocontrôles : une tâche en aval vérifie que la partition a été écrite et que les données paraissent correctes avant toute lecture. Pour une exécution bloquée plutôt qu'en échec, la fonctionnalité Deadline Alerts, ajoutée dans Airflow 3.1 pour remplacer la fonction SLA supprimée dans 3.0, déclenche une fonction de rappel, par exemple un envoi de courriel, dès qu'un seuil de durée est dépassé.

La remise en service déclenche une avalanche d'exécutions. Avec `catchup=True`, l'ordonnanceur crée une exécution pour chaque intervalle de données manqué depuis la date de début. La documentation d'Airflow ajoute que ce rattrapage se déclenche aussi lorsqu'un Dag est désactivé pendant un certain temps puis réactivé. Après un mois d'arrêt, un pipeline quotidien peut revenir avec trente exécutions en attente sur la même source. Le réglage par défaut, `catchup_by_default=False`, ne crée une exécution que pour le dernier intervalle. La documentation recommande de laisser le rattrapage désactivé pour un Dag dont les tâches lisent l'heure courante plutôt que leur intervalle. Mieux vaut remplir l'historique volontairement avec `airflow backfill create`, qui reçoit une plage de dates et une limite d'exécutions parallèles via `--max-active-runs`.

Du code lourd au niveau supérieur d'un fichier Dag ralentit l'ordonnanceur. Airflow analyse sans cesse chaque fichier Dag, avec au moins `min_file_process_interval` secondes entre deux analyses. Le code au niveau supérieur du fichier, en dehors de toute tâche, s'exécute donc bien plus souvent que le pipeline lui-même. Une requête de base de données, un appel réseau ou un import lent à cet endroit coûte du temps à chaque analyse. Le guide de bonnes pratiques montre un exemple où un seul appel ajoute 1 000 secondes à l'analyse d'un Dag. Réservez le niveau supérieur à la déclaration des tâches et de leur ordre, et déplacez les imports et appels coûteux dans les fonctions des tâches qui en ont besoin.

Pour aller plus loin

- Apache Airflow — Présentation du projet : flux de travail en code Python, orchestration par lots, reprises historiques et réexécution des seules tâches en échec.
- Bonnes pratiques — Tâches traitées comme des transactions, résultat identique à chaque réexécution, code au niveau supérieur et autocontrôles.
- Dags — Déclaration du calendrier, des tâches, des dépendances et des arguments par défaut, comme les nouvelles tentatives, et place de Deadline Alerts.
- Exécutions des Dag — Intervalles de données, rattrapage, reprise historique et réinitialisation des tâches pour les relancer.
- Tâches — Dépendances entre tâches, délais d'expiration, politiques de nouvelle tentative et passage des SLA à Deadline Alerts.
- psql — Variables et emploi des guillemets, `ON_ERROR_STOP` et codes de sortie lus par l'orchestrateur.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un pipeline de traitement ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/pipeline/