Engineer

Idempotent

Même effet voulu après une ou plusieurs exécutions

Ce que c’est

Une opération est idempotente si l’exécuter plusieurs fois a le même effet qu’une seule fois ; cette propriété s’appelle l’idempotence. Le terme vient des mathématiques : une fonction est idempotente si l’appliquer à son propre résultat ne change rien, donc f(f(x)) égale f(x). Arrondir vers le bas un nombre déjà arrondi vers le bas donne le même nombre.

En logiciel, le test utile est une seconde exécution. Affecter une valeur est idempotent : la seconde écrit la valeur déjà présente. Ajouter à une valeur ne l’est pas, car la seconde ajoute encore :

```
-- idempotent: a second run sets the same value
UPDATE accounts SET plan = 'pro' WHERE id = 42;

-- not idempotent: a second run adds ten more
UPDATE accounts SET credits = credits + 10 WHERE id = 42;
```

HTTP donne au terme sa définition la plus précise. Le RFC 9110 qualifie une méthode d’idempotente si plusieurs requêtes identiques sont censées avoir sur le serveur le même effet qu’une seule. Il énumère PUT, DELETE et les méthodes sûres GET, HEAD, OPTIONS et TRACE. Deux détails comptent. La propriété couvre l’effet demandé par le client : le serveur peut donc journaliser chaque requête séparément. La réponse peut aussi différer : un second DELETE de la même ressource peut répondre 404 après un premier 200, tout en restant idempotent, car la ressource a disparu dans les deux cas. PATCH est hors de la liste. Le RFC 5789 le dit ni sûr ni idempotent, mais un client peut rendre un PATCH particulier idempotent en le conditionnant à un ETag fort dans un en-tête If-Match : une copie répétée est refusée dès que la première a changé la ressource.

La sûreté est la plus stricte des deux promesses. Une méthode sûre au sens du RFC 9110 est essentiellement en lecture seule : toute méthode sûre est idempotente, tandis que DELETE est idempotent sans être sûr.

Utilisez le concept pour prendre une décision

Utilisez « Que signifie idempotent ? » 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

Les réseaux perdent des réponses. Si le client envoie une requête et que la connexion se coupe avant la réponse, il ne sait pas si le serveur ne l’a jamais vue, l’a vue puis s’est arrêté, ou l’a exécutée et a perdu la réponse. Répéter est le recours évident ; l’idempotence rend cette répétition inoffensive. Le RFC 9110 fonde précisément là-dessus ses règles de nouvelle tentative. Une requête idempotente peut être répétée automatiquement après un échec de communication. Un client ne devrait pas réessayer automatiquement une requête non idempotente, sauf s’il sait qu’elle l’est en pratique ou peut déterminer que l’originale n’a jamais été appliquée ; un proxy ne doit jamais la réessayer de sa propre initiative.

Les files de messages posent la même question à grande échelle. Beaucoup de systèmes promettent une livraison au moins une fois : un consommateur qui plante après son travail mais avant l’accusé de réception reçoit le même message à nouveau. Une livraison exactement une fois sur un réseau est bien plus difficile à promettre. La réponse pratique est donc un consommateur pour lequel une seconde livraison ne change rien. La gestion de configuration en dépend aussi. Un outil amenant les serveurs à l’état déclaré s’exécute encore et encore, après chaque changement et souvent selon un calendrier ; chaque passage doit laisser un serveur déjà correct exactement comme il l’a trouvé.

La vraie décision est de savoir comment obtenir l’idempotence quand une opération ne l’a pas naturellement. Trois réponses sont courantes. La première est la conception : exprimer un état cible plutôt qu’une étape, « s’assurer que cette ligne figure dans le fichier » plutôt qu’« ajouter cette ligne à la fin ». La

deuxième est une clé : le client joint un identifiant unique à la requête, le serveur l'enregistre sous contrainte d'unicité et une répétition retrouve l'enregistrement et le résultat initial, sans seconde action. La troisième est de vérifier avant d'agir, comme la plupart des modules Ansible : comparer l'état actuel à l'état souhaité et terminer sans agir s'ils correspondent déjà. Chacune a un coût. L'état cible exige un modèle clair du résultat final ; les clés, du stockage et une durée de conservation ; la vérification, un moyen fiable de lire l'état actuel.

À quoi cela ressemble en pratique

Les serveurs de l'entreprise sont configurés avec Ansible. Sur ce projet, l'idempotence est un contrat : un hôte convergé annonce zéro changement. Le provisionnement sans argument amène chaque hôte à l'état déclaré ; un play incapable d'annoncer zéro changement sur un hôte déjà correct est inachevé. L'idempotence est ainsi mesurée à chaque passage, plutôt qu'espérée.

La même idée appliquée aux données sous-tend un test qui exécute deux fois le pipeline de construction à dix-neuf étapes. La page précise le défaut visé : un générateur qui fonctionne au premier passage et corrompt les données au second, par des insertions sans vérification ou une dépendance à l'ordre de sortie d'une étape antérieure. Le test construit la base depuis un fichier vide, prend un instantané, réexécute tout le pipeline sur le résultat et exige que le second passage ne change rien, jusqu'aux nombres de lignes, contenus et identifiants. Le test fournit des preuves sur les exécutions répétées avec les entrées et l'état de la base qu'il vérifie.

Pour vendre un stock limité, la note sur la vente de la dernière place exactement une fois combine une clé et des écritures conditionnelles. Chaque processus de commande porte une clé d'idempotence et la table des réservations porte `UNIQUE (tenant_id, idempotency_key)` : un clic répété renvoie la réservation existante, pas une seconde unité. Une libération exécutée deux fois, par exemple par expiration d'un délai et par bouton d'annulation, rend l'unité une seule fois : elle ne recrédite le stock que si sa propre mise à jour a modifié exactement une ligne. Plus loin, chaque consommateur de messages est idempotent, avec la commande comme clé : les messages arrivent au moins une fois et l'idempotence ramène leur effet à une seule fois en pratique.

Où les problèmes surviennent

Une requête répétée facture deux fois. Un POST créant un paiement, une commande ou un courriel n'est pas idempotent par défaut ; si le client ou proxy réessaie après un délai dépassé, il en crée un second. Donnez à ces requêtes une clé d'idempotence que le serveur stocke sous contrainte d'unicité, puis renvoyez le résultat enregistré quand la même clé revient.

Un résultat vide est pris pour une erreur. Dans PostgreSQL, `INSERT ... ON CONFLICT DO NOTHING` saute une ligne qui violerait une contrainte d'unicité, et `RETURNING` ne renvoie que les lignes réellement insérées ou mises à jour. Réinsérer la même ligne ne renvoie donc rien : c'est le résultat correct du second passage. Un code qui y voit un échec transforme une répétition inoffensive en page d'erreur ; il devrait rechercher la ligne existante.

Une tâche de commande change l'hôte à chaque passage. La documentation Ansible dit que la plupart des modules vérifient si l'état souhaité est déjà atteint, mais aussi que « les playbooks et les modules n'ont pas tous ce comportement ». Une tâche exécutant une commande est l'exception habituelle : la commande est arbitraire et Ansible ne peut savoir ce qu'elle fait. L'option `creates` du module command saute l'étape si un fichier donné existe déjà ; `removes` ne l'exécute que tant qu'il est présent ; `changed_when` définit le résultat comptant comme changement. Sans ces options, une tâche de commande se relance à chaque convergence, quel que soit l'état de l'hôte.

Un compteur compte la nouvelle tentative. Avec une livraison au moins une fois, un consommateur qui incrémente un compteur ou envoie une notification par message le fait deux fois si le message arrive deux fois. Stockez sa clé métier avec l'effet, dans la même transaction, et sautez toute clé déjà enregistrée.

Le second passage corrompt ce qu'a construit le premier. Une étape peut être correcte sur une base vide et incorrecte sur une base pleine, en insérant un doublon, renumérotant les identifiants ou dépendant d'un ordre laissé par le premier passage. Un test partant de rien et exécuté une fois réussira quand même. Exécutez tout le travail deux fois et exigez que le second passage ne change rien, comme le test de pipeline précédent.

Pour aller plus loin

- RFC 9110, section 9.2.2 : méthodes idempotentes — Définition HTTP, méthodes idempotentes et règles de nouvelle tentative automatique.
- RFC 5789 : méthode PATCH pour HTTP — Pourquoi PATCH n'est ni sûr ni idempotent, et comment une requête conditionnelle permet de le répéter sans effet supplémentaire.
- Ansible : introduction aux playbooks — État souhaité, idempotence et mode de vérification expliqués par Ansible.
- Ansible : module command — Options `creates` et `removes` rendant une commande répétable sans effet supplémentaire.
- PostgreSQL : INSERT — `ON CONFLICT` et ce que renvoie `RETURNING` si la ligne existe déjà.

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 « Que signifie idempotent ? » 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/idempotent/