Engineer

Idempotent

Gefahrlos wiederholbar: Der zweite Lauf ändert nichts mehr

Was es ist

Ein Vorgang ist idempotent, wenn seine mehrfache Ausführung dieselbe Wirkung hat wie eine einmalige. Die Eigenschaft heißt Idempotenz oder Idempotentheit. Das Wort stammt aus der Mathematik: Eine Funktion ist idempotent, wenn ihre Anwendung auf ihr eigenes Ergebnis nichts ändert, also f(f(x)) gleich f(x) ist. Eine bereits abgerundete Zahl noch einmal abzurunden ergibt dieselbe Zahl.

Bei Software ist ein zweiter Lauf ein hilfreicher Test. Einen Wert zu setzen ist idempotent, weil der zweite Lauf den bereits vorhandenen Wert schreibt. Etwas zu einem Wert zu addieren ist es nicht, weil der zweite Lauf erneut addiert:

```
-- 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 gibt dem Wort seine präziseste Definition. RFC 9110 nennt eine Anfragemethode idempotent, wenn beabsichtigt ist, dass mehrere identische Anfragen auf dem Server dieselbe Wirkung haben wie eine einzelne. Er zählt PUT, DELETE und die sicheren Methoden GET, HEAD, OPTIONS und TRACE auf. Zwei Details sind wichtig. Die Eigenschaft betrifft die vom Client gewünschte Wirkung; ein Server darf deshalb jede Anfrage gesondert protokollieren. Und die Antwort darf sich unterscheiden: Ein zweites DELETE derselben Ressource kann mit 404 antworten, während das erste 200 lieferte. Die Methode bleibt idempotent, weil die Ressource in beiden Fällen weg ist. PATCH steht nicht in der Liste. Nach RFC 5789 ist es weder sicher noch idempotent. Ein Client kann ein bestimmtes PATCH

jedoch idempotent senden, indem er es mit einem starken ETag im Header If-Match an eine Bedingung knüpft. Eine wiederholte Kopie wird dann abgelehnt, sobald die erste die Ressource geändert hat.

Eine „sichere“ Methode gibt das strengere der beiden Versprechen. Eine sichere Methode im Sinne von RFC 9110 liest im Wesentlichen nur. Deshalb ist jede sichere Methode idempotent, während DELETE idempotent, aber nicht sicher ist.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was bedeutet idempotent?“ 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

Netzwerke verlieren Antworten. Sendet ein Client eine Anfrage und bricht die Verbindung vor dem Eintreffen der Antwort ab, kann er nicht erkennen, ob der Server die Anfrage nie sah, sie erhielt und dann zum Stillstand kam oder sie ausführte und die Antwort verloren ging. Die Anfrage zu wiederholen ist die naheliegende Maßnahme zur Wiederherstellung, und Idempotenz macht die Wiederholung unschädlich. Darauf baut RFC 9110 seine Wiederholungsregeln auf. Eine idempotente Anfrage kann nach einem Kommunikationsfehler automatisch wiederholt werden. Ein Client sollte eine nicht idempotente Anfrage nicht automatisch wiederholen, außer er weiß, dass sie praktisch idempotent ist, oder kann erkennen, dass das Original nie angewendet wurde. Ein Proxy darf eine solche Anfrage niemals eigenständig wiederholen.

Warteschlangen stellen dieselbe Frage in größerem Maßstab. Viele Nachrichtensysteme versprechen mindestens einmalige Zustellung: Stürzt ein Consumer nach seiner Arbeit und vor der Bestätigung ab, erhält er dieselbe Nachricht erneut. Genau einmalige Zustellung über ein Netzwerk ist ein viel schwierigeres Versprechen. Die praktische Antwort ist daher ein Consumer, bei dem eine zweite Zustellung nichts ändert. Auch Konfigurationsmanagement hängt davon ab. Ein Werkzeug, das Server in einen deklarierten Zustand

bringt, läuft immer wieder, nach jeder Änderung und oft nach Zeitplan. Jeder Lauf muss einen bereits korrekten Server genau so hinterlassen, wie er ihn vorgefunden hat.

Die eigentliche Entscheidung ist, wie man Idempotenz erreicht, wenn ein Vorgang sie nicht von Natur aus besitzt. Drei Antworten sind verbreitet. Erstens das Design: die Änderung als Zielzustand statt als Schritt ausdrücken, also „sicherstellen, dass diese Zeile in der Datei steht“ statt „diese Zeile anhängen“. Zweitens ein Schlüssel: Der Client versieht die Anfrage mit einer eindeutigen Kennung, der Server speichert sie unter einer Eindeutigkeitsbedingung, und eine Wiederholung findet den Datensatz und erhält das ursprüngliche Ergebnis statt einer zweiten Aktion. Drittens die Prüfung vor dem Handeln, wie bei den meisten Ansible-Modulen: Sie vergleichen Ist- und Sollzustand und beenden sich ohne Eingriff, wenn beide bereits übereinstimmen. Jede Lösung kostet etwas. Ein Zielzustandsdesign braucht ein klares Modell des Endzustands, Schlüssel brauchen Speicher und eine Aufbewahrungsregel, und die Vorabprüfung braucht eine zuverlässige Möglichkeit, den Istzustand zu lesen.

Wie es in der Praxis aussieht

Die eigenen Server des Unternehmens werden mit Ansible konfiguriert. In diesem Projekt ist Idempotenz eine verbindliche Anforderung: Ein konvergierter Host meldet null Änderungen. Eine Bereitstellung ohne Argumente bringt jeden Host in seinen deklarierten Zustand. Ein Play, das auf einem bereits korrekten Host nicht null Änderungen melden kann, gilt als unfertig. Idempotenz wird damit bei jedem Lauf gemessen, statt nur erhofft.

Dieselbe Idee steckt, auf Daten angewendet, hinter einem Test, der die neunzehnstufige Buildpipeline zweimal ausführt. Die Seite nennt die Fehlerart, für die er gedacht ist: Ein Generator funktioniert beim ersten Mal und richtet beim zweiten Schäden an, weil Schritte ungeprüft einfügen oder von der Ausgabereihenfolge eines früheren Schritts abhängen. Der Test baut die Datenbank aus einer leeren Datei auf, erstellt einen Schnappschuss und führt die gesamte Pipeline erneut auf dem Ergebnis aus. Der zweite Durchlauf darf nichts ändern, bis hin zu Zeilenzahlen, Inhalten und Kennungen. Der Test liefert Nachweise für wiederholte Läufe mit den von ihm geprüften Eingaben und Datenbankzuständen.

Beim Verkauf begrenzter Bestände verbindet die Notiz zum genau einmaligen Verkauf des letzten Tickets einen Schlüssel mit bedingten Schreiboperationen. Jeder Bezahlvorgang trägt einen Idempotenzschlüssel, und die Tabelle der Reservierungen enthält `UNIQUE (tenant_id, idempotency_key)`. Wird der Klick wiederholt, kommt so die vorhandene Reservierung zurück statt einer zweiten Einheit. Eine zweimal ausgeführte Freigabe, etwa durch Zeitüberschreitung und Abbrechen-Schaltfläche, gibt die Einheit nur einmal zurück: Sie schreibt den Bestand nur gut, wenn ihr eigenes Update genau eine Zeile verändert hat. Im weiteren Ablauf ist jeder Nachrichtenconsumer idempotent, mit der Bestellung als Schlüssel. Nachrichten kommen mindestens einmal an; Idempotenz macht daraus praktisch eine einmalige Wirkung.

Wo es schiefgeht

Eine wiederholte Anfrage bucht zweimal ab. Ein POST zum Erstellen einer Zahlung, Bestellung oder E-Mail ist standardmäßig nicht idempotent. Wiederholt ein Client oder Proxy es nach einer Zeitüberschreitung, entsteht ein zweites Exemplar. Geben Sie diesen Anfragen einen Idempotenzschlüssel, den der Server unter einer Eindeutigkeitsbedingung speichert, und liefern Sie das gespeicherte Ergebnis, wenn derselbe Schlüssel wiederkommt.

Ein leeres Ergebnis gilt als Fehler. In PostgreSQL überspringt `INSERT ... ON CONFLICT DO NOTHING` eine Zeile, die eine Eindeutigkeitsbedingung verletzen würde. `RETURNING` liefert nur tatsächlich eingefügte oder aktualisierte Zeilen. Ein wiederholtes Einfügen derselben Zeile liefert deshalb nichts zurück, das korrekte Ergebnis eines zweiten Laufs. Code, der das leere Ergebnis als Fehler behandelt, macht aus einer unschädlichen Wiederholung eine Fehlerseite. Er sollte stattdessen die schon vorhandene Zeile nachschlagen.

Eine Befehlsaufgabe verändert den Host bei jedem Lauf. Laut Ansible-Dokumentation prüfen die meisten Module, ob der Sollzustand bereits erreicht ist. Sie sagt aber auch, dass „nicht alle Playbooks und nicht alle Module sich so verhalten“. Eine Aufgabe, die einen Befehl ausführt, ist die übliche Ausnahme: Der Befehl ist beliebig, und Ansible kann nicht wissen, was er tut. Die Option `creates` des Command-Moduls überspringt den Schritt, wenn eine bestimmte Datei bereits existiert. `removes` führt ihn nur aus, solange die Datei noch vorhanden ist, und `changed_when` legt fest, welches Ergebnis als Änderung zählt. Ohne diese Angaben läuft eine Befehlsaufgabe bei jeder Konvergenz erneut, unabhängig vom Zustand des Hosts.

Ein Zähler zählt die Wiederholung mit. Bei mindestens einmaliger Zustellung tut ein Consumer, der pro Nachricht einen Zähler erhöht oder eine Benachrichtigung sendet, das zweimal, wenn dieselbe Nachricht zweimal zugestellt wird. Speichern Sie den fachlichen Schlüssel der Nachricht zusammen mit ihrer Wirkung in derselben Transaktion und überspringen Sie jeden bereits gespeicherten Schlüssel.

Der zweite Lauf beschädigt den Aufbau des ersten. Ein Schritt kann auf einer leeren Datenbank korrekt und auf einer gefüllten falsch sein: Er fügt Duplikate ein, nummeriert Kennungen neu oder verlässt sich auf eine vom ersten Lauf hinterlassene Reihenfolge. Ein Test, der bei null beginnt und nur einmal läuft, besteht trotzdem. Führen Sie den gesamten Auftrag zweimal aus und verlangen Sie, dass der zweite Lauf nichts ändert, wie beim Pipelinetest oben.

Weiterführende Lektüre

- RFC 9110, Abschnitt 9.2.2: idempotente Methoden — Die HTTP-Definition, die idempotenten Methoden und die Regeln für automatische Wiederholungen.
- RFC 5789: PATCH-Methode für HTTP — Warum PATCH weder sicher noch idempotent ist und wie eine bedingte Anfrage eine sichere Wiederholung ermöglicht.
- Ansible: Einführung in Playbooks — Sollzustand, Idempotenz und Check-Modus in Ansibles eigenen Worten.
- Ansible: Command-Modul — Die Optionen `creates` und `removes`, die eine sichere erneute Ausführung ermöglichen.
- PostgreSQL: INSERT — `ON CONFLICT` und die Rückgabe von `RETURNING`, wenn eine Zeile bereits existiert.

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 bedeutet idempotent?“ 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/idempotent/