Engineer

Idempotent

säker att köra två gånger, då andra gången inte ändrar något

Vad det är

En operation är idempotent om en upprepning med samma indata ger samma avsedda effekt som en enda körning. Aktivera abonnemanget Pro på samma konto två gånger, så har kontot fortfarande samma abonnemang. Lägg i stället till tio krediter två gånger, så ökar saldot med tjugo. En andra körning är ett praktiskt sätt att se skillnaden.

Egenskapen kallas idempotens. Ordet kommer från matematiken, där en funktion är idempotent om den ger samma värde när den tillämpas på sitt eget resultat, så att f(f(x)) är lika med f(x). Avrunda ett tal nedåt till ett heltal. Vid en andra avrundning förblir heltalet detsamma.

De här två SQL-satserna visar skillnaden i en databas:

```
-- 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 ger ordet dess mest precisa definition. RFC 9110 kallar en metod idempotent om den avsedda effekten på servern av flera identiska begäranden är densamma som effekten av en enda, och räknar upp PUT, DELETE och de säkra metoderna GET, HEAD, OPTIONS och TRACE. Två detaljer i den definitionen gör skillnad. Egenskapen gäller den effekt som klienten bad om, så servern får gärna logga varje begäran för sig. Och svaret får skilja sig: en andra DELETE av samma resurs kan svara 404 där den första svarade 200, och metoden är ändå idempotent, eftersom resursen är borta i båda fallen.

PATCH står utanför listan. Enligt RFC 5789 är den varken säker eller idempotent, men en klient kan skicka en viss PATCH på ett idempotent sätt genom att göra den villkorad, med en stark ETag i ett huvudfält `If-Match`. En upprepad kopia avvisas då så snart den första har ändrat resursen.

Säker är det strängare av de två löftena. En säker metod i RFC 9110:s mening är i allt väsentligt en läsoperation, så varje säker metod är idempotent, medan DELETE är idempotent utan att vara säker.

Använd begreppet i ett beslut

Använd ”Vad betyder idempotent?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Siffror som kommer fram dit de behövs, på fasta tider, utan att någon kopierar ett kalkylblad i slutet av månaden.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. En infrastrukturplan för 14 avdelningar — 14 forskningsavdelningar under en och samma infrastrukturplan

En infrastrukturplan för 14 avdelningar Hör av er →

Varför det spelar roll

Nätverk tappar svar. När en klient skickar en begäran och anslutningen bryts innan svaret kommer fram kan den inte avgöra om servern aldrig såg begäran, såg den och avbröt, eller utförde den och tappade svaret. Att upprepa begäran är den självklara vägen tillbaka, och det är idempotensen som gör upprepningen ofarlig. RFC 9110 bygger sina regler för nya försök på just det. En idempotent begäran får upprepas automatiskt efter ett kommunikationsfel. En klient bör inte automatiskt försöka igen med en begäran som inte är idempotent, om den inte vet att begäran i praktiken är idempotent eller kan se att originalet aldrig verkställdes. En proxy får aldrig automatiskt upprepa en begäran som inte är idempotent.

Köer ställer samma fråga i större skala. Många meddelandesystem lovar leverans minst en gång: en konsument som kraschar efter att ha gjort sitt arbete men innan den har kvitterat meddelandet får samma meddelande en gång till. Leverans exakt en gång över ett nätverk är ett betydligt svårare löfte, så det praktiska svaret är en konsument för vilken en andra leverans inte ändrar något. Konfigurationshantering vilar också på egenskapen. Ett verktyg som för servrar till ett deklarerat tillstånd körs om och om igen, efter varje ändring och ofta enligt schema, och varje körning måste lämna en korrekt server exakt som den hittade den.

Det verkliga beslutet är hur ni får idempotens där en operation saknar den av naturen, och det finns tre vanliga svar.

- **Design.** Uttryck ändringen som ett måltillstånd i stället för ett steg, alltså ”se till att den här raden finns i filen” i stället för ”lägg till den här raden”.
- **En nyckel.** Klienten bifogar en unik identifierare till begäran, servern sparar den under ett unikhetsvillkor, och en upprepning hittar posten och får tillbaka det ursprungliga resultatet i stället för en andra åtgärd.
- **Kontroll före handling.** Så fungerar de flesta moduler i Ansible: de jämför det aktuella tillståndet med det önskade och avslutar utan att göra något när de redan stämmer överens.

Var och en har sin kostnad. Design kring måltillstånd kräver en tydlig modell av sluttillståndet, nycklar kräver lagring och en regel för hur länge de sparas, och kontroll före handling kräver ett pålitligt sätt att läsa av det aktuella tillståndet.

Så ser det ut i praktiken

Företagets egna servrar konfigureras med Ansible, och i det projektet är idempotens ett kontrakt: en konvergerad värd rapporterar noll ändringar. Provisionering utan argument konvergerar varje värd till dess deklarerade tillstånd, och en play som inte kan rapportera noll ändringar på en värd som redan är korrekt räknas som ofärdig. Det gör idempotens till något som varje körning mäter, i stället för något man hoppas på.

Samma idé tillämpad på data ligger bakom ett test som kör hela byggpipelinen i nitton steg två gånger. Sidan namnger den sorts bugg som testet finns till för: en generator som fungerar första gången och korrumperar andra gången, genom steg som infogar utan att kontrollera eller som beror på ordningen i ett tidigare stegs utdata. Testet bygger databasen från en tom fil, tar en ögonblicksbild, kör hela pipelinen igen över resultatet och kräver att den andra körningen inte ändrar någonting, ända ned till radantal, innehåll och identifierare. Testet ger belägg för upprepade körningar med de indata och det databastillstånd som det prövar.

För försäljning av begränsat lager kombinerar anteckningen om att sälja den sista biljetten exakt en gång en nyckel med villkorade skrivningar. Varje köp i kassan bär en idempotensnyckel, och reservationstabellen har `UNIQUE (tenant_id, idempotency_key)`, så att ett upprepat klick får tillbaka den reservation det redan har i stället för en andra enhet. En åtgärd som frigör en reservation och körs två

gånger, t.ex. från en timeout och en avbrytknapp, lämnar tillbaka enheten en gång, eftersom lagersaldot bara krediteras när åtgärdens egen uppdatering ändrade exakt en rad. Längre fram i flödet är varje meddelandekonsument idempotent, nycklad på ordern: meddelanden kommer fram minst en gång, och idempotensen gör det i praktiken till en gång.

Var det går fel

En begäran som görs om debiterar två gånger. En POST som skapar en betalning, en order eller ett mejl är inte idempotent som standard, och en klient eller proxy som försöker igen efter en timeout skapar en till. Ge sådana begäranden en idempotensnyckel som servern sparar under ett unikhetsvillkor, och returnera det sparade resultatet när samma nyckel kommer tillbaka.

Ett tomt resultat tolkas som ett fel. I PostgreSQL hoppar `INSERT ... ON CONFLICT DO NOTHING` över en rad som skulle bryta mot ett unikhetsvillkor, och `RETURNING` lämnar bara tillbaka rader som faktiskt infogades eller uppdaterades. En upprepad infogning av samma rad returnerar därför ingenting, vilket är det korrekta utfallet av en andra körning. Kod som behandlar det tomma resultatet som ett misslyckande gör ett ofarligt nytt försök till en felsida; den borde slå upp raden som redan finns.

En kommandotask ändrar värden vid varje körning. Ansibles dokumentation säger att de flesta moduler kontrollerar om det önskade tillståndet redan är uppnått, och också att inte alla playbooks och inte alla moduler beter sig så. En task som kör ett kommando är det vanliga undantaget, eftersom kommandot är godtyckligt och Ansible inte kan veta vad det gör. Kommandomodulens alternativ `creates` hoppar över steget när en viss fil redan finns, `removes` kör det bara så länge filen finns kvar, och `changed_when` anger vilket utfall som räknas som en ändring. En kommandotask utan något av dem körs igen vid varje konvergering, oavsett i vilket tillstånd värden är.

En räknare räknar med det nya försöket. Vid leverans minst en gång gör en konsument som räknar upp en räknare eller skickar en avisering för varje meddelande det två gånger för ett meddelande som levereras två gånger. Spara meddelandets affärsnyckel tillsammans med effekten, i samma transaktion, och hoppa över varje nyckel som redan är sparad.

Den andra körningen korrumperar det som den första byggde. Ett steg kan vara korrekt på en tom databas och fel på en full, genom att infoga en dubblett, numrera om identifierare eller luta sig mot en ordning som den första körningen lämnade efter sig. Ett test som börjar från ingenting och körs en gång går igenom ändå. Kör hela jobbet två gånger och kräv att den andra körningen inte ändrar något, precis som byggtestet ovan gör.

Läs mer

- RFC 9110, section 9.2.2: Idempotent Methods — HTTP:s definition, de idempotenta metoderna och reglerna för automatiska nya försök.
- RFC 5789: PATCH Method for HTTP — varför PATCH varken är säker eller idempotent, och hur en villkorad begäran gör den ofarlig att upprepa.
- Ansible: Introduction to playbooks — önskat tillstånd, idempotens och check mode med Ansibles egna ord.
- Ansible: command module — alternativen `creates` och `removes`, som gör ett kommando ofarligt att köra om.
- PostgreSQL: INSERT — `ON CONFLICT` och vad `RETURNING` lämnar tillbaka när en rad redan finns.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad betyder idempotent?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/idempotent/