Engineer

Idempotent

samme tilsiktede effekt ved én eller flere kjøringer

Hva det er

En operasjon er idempotent når det å utføre den flere ganger har samme virkning som å utføre den én gang, og egenskapen kalles idempotens. Ordet kommer fra matematikken, der en funksjon er idempotent når den kan brukes på sitt eget resultat uten at noe endres, slik at f(f(x)) er lik f(x): runder du ned et tall som allerede er rundet ned, får du det samme tallet.

I programvare er den nyttige testen en kjøring nummer to. Å sette en verdi er idempotent, fordi den andre kjøringen skriver verdien som allerede står der. Å legge til på en verdi er det ikke, fordi den andre kjøringen legger til én gang til:

```
-- 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 gir ordet sin mest presise definisjon. RFC 9110 kaller en forespørselsmetode idempotent hvis den tilsiktede virkningen på serveren av flere identiske forespørsler er den samme som virkningen av én, og lister opp PUT, DELETE og de trygge metodene GET, HEAD, OPTIONS og TRACE. To detaljer i den definisjonen betyr noe. Egenskapen gjelder virkningen klienten ba om, så serveren står fritt til å logge hver forespørsel for seg. Og svaret kan være forskjellig: en ny DELETE av den samme ressursen kan svare 404 der den første svarte 200, og metoden er likevel idempotent, fordi ressursen er borte i begge tilfeller. PATCH står utenfor listen. RFC 5789 sier at den verken er trygg eller idempotent, men en klient kan sende en bestemt PATCH på en idempotent måte ved å gjøre den betinget, med en sterk ETag i en If-Match-header, slik at en gjentatt kopi blir avvist så snart den første har endret ressursen.

Trygg er det strengeste av de to løftene. En trygg metode i betydningen fra RFC 9110 er i det vesentlige en leseoperasjon, så hver trygg metode er idempotent, mens DELETE er idempotent uten å være trygg.

Bruk begrepet i en beslutning

Bruk «Hva betyr idempotent?» 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

Nettverk mister svar. Når en klient sender en forespørsel og forbindelsen brytes før svaret kommer, har klienten ingen måte å vite om serveren aldri fikk forespørselen, fikk den og stoppet underveis, eller utførte den og svaret gikk tapt på veien tilbake. Å gjenta forespørselen er den åpenbare måten å komme seg på, og idempotens er det som gjør gjentakelsen ufarlig. RFC 9110 bygger reglene for nye forsøk på nettopp det. En idempotent forespørsel kan gjentas automatisk etter en kommunikasjonsfeil. En klient bør ikke prøve en ikke-idempotent forespørsel på nytt automatisk med mindre den vet at forespørselen er idempotent i praksis, eller kan se at originalen aldri ble utført, og en proxy skal aldri prøve en slik forespørsel på nytt på egen hånd.

En kø reiser det samme spørsmålet i stor skala. Mange meldingssystemer lover levering minst én gang: en konsument som krasjer etter å ha gjort jobben, men før den har kvittert for meldingen, får den samme meldingen en gang til. Levering nøyaktig én gang over et nettverk er et langt vanskeligere løfte, så det praktiske svaret er en konsument der en ny levering ikke endrer noe. Styring av konfigurasjon er også avhengig av egenskapen. Et verktøy som bringer servere til en deklarert tilstand, kjører igjen og igjen, etter hver endring og ofte etter en tidsplan, og hver kjøring må la en server som allerede er riktig, stå nøyaktig slik den var.

Det du faktisk må avgjøre, er hvordan du får idempotens der en operasjon mangler det av natur, og det finnes tre vanlige svar. Det første er design: uttrykk endringen som en måltilstand i stedet for et steg, altså «sørg for at denne linjen står i filen» i stedet for «legg til denne linjen». Det andre er en nøkkel: klienten

fester en unik identifikator til forespørselen, serveren lagrer den under en unikhetsbegrensning, og en gjentakelse finner posten og får det opprinnelige resultatet tilbake i stedet for en ny handling. Det tredje er å sjekke før man handler, og slik virker de fleste moduler i Ansible: de sammenligner den gjeldende tilstanden med den ønskede og avslutter uten å gjøre noe når de to allerede stemmer overens. Hvert svar har sin pris. Design etter måltilstand krever en klar modell av sluttilstanden, nøkler krever lagring og en regel for hvor lenge de beholdes, og å sjekke først krever en pålitelig måte å lese den gjeldende tilstanden på.

Slik ser det ut i praksis

Selskapets egne servere konfigureres med Ansible, og i det prosjektet er idempotens en kontrakt: en konvergert vert rapporterer null endringer. Den samme klargjøringen, startet uten argumenter, skal bringe hver vert til tilstanden den er deklarert med, og når den kjøres på nytt mot en vert som allerede er riktig, skal den rapportere null endringer. En play som ikke klarer det, regnes som uferdig. Dermed blir idempotens noe hver kjøring måler, og ikke noe man håper på.

Den samme tanken, brukt på data, ligger bak en test som kjører hele byggepipelinen på nitten trinn to ganger. Siden navngir hvilken type feil testen finnes for: en generator som virker første gang og ødelegger andre gang, gjennom steg som setter inn uten å sjekke, eller som avhenger av rekkefølgen i utdataene fra et tidligere steg. Testen bygger databasen fra en tom fil, tar et øyeblikksbilde, kjører hele pipelinen på nytt over resultatet og krever at den andre kjøringen ikke endrer noe, ned til radantall, innhold og identifikatorer. Testen gir dokumentasjon for gjentatte kjøringer med inndataene og databasetilstanden den prøver ut.

For salg av en begrenset beholdning kombinerer notatet om å selge den siste billetten nøyaktig én gang en nøkkel med betingede skrivinger. Hver utsjekking har en idempotensnøkkel, og reservasjonstabellen har `UNIQUE (tenant_id, idempotency_key)`, slik at et klikk som prøves på nytt, får tilbake reservasjonen det allerede har, i stedet for en enhet til. En frigivelse som kjører to ganger, for eksempel fra et tidsavbrudd og en avbryt-knapp, gir enheten tilbake bare én gang, fordi den fører beholdningen tilbake bare når dens egen oppdatering endret nøyaktig én rad. Lenger ut i flyten er hver meldingskonsument idempotent, med ordren som nøkkel: meldinger kommer minst én gang, og idempotensen gjør det i praksis til én gang.

Hvor det går galt

En forespørsel som prøves på nytt, belaster kunden to ganger. En POST som oppretter en betaling, en ordre eller en e-post, er ikke idempotent i utgangspunktet, og en klient eller proxy som prøver den på nytt etter et tidsavbrudd, oppretter en til. Gi slike forespørsler en idempotensnøkkel som serveren lagrer under en unikhetsbegrensning, og returner det lagrede resultatet når den samme nøkkelen kommer igjen.

Et tomt resultat leses som en feil. I PostgreSQL hopper `INSERT ... ON CONFLICT DO NOTHING` over en rad som ville brutt en unikhetsbegrensning, og `RETURNING` gir bare tilbake rader som faktisk ble satt inn eller oppdatert. En gjentatt innsetting av den samme raden returnerer derfor ingenting, og det er det riktige utfallet av en kjøring nummer to. Kode som behandler det tomme resultatet som en feil, gjør et ufarlig nytt forsøk om til en feilside; den bør heller slå opp raden som allerede står der.

En kommandooppgave endrer verten ved hver kjøring. Dokumentasjonen til Ansible sier at de fleste moduler sjekker om den ønskede tilstanden allerede er nådd, og også at ikke alle playbooks og ikke alle moduler oppfører seg slik. En oppgave som kjører en kommando, er det vanlige unntaket, fordi kommandoen er vilkårlig og Ansible ikke kan vite hva den gjør. Valget `creates` i kommandomodulen hopper over steget når en gitt fil allerede finnes, `removes` kjører det bare mens filen fortsatt er der, og `changed_when` definerer hvilket utfall som teller som en endring. En kommandooppgave uten noen av dem kjører igjen ved hver konvergering, uansett hvilken tilstand verten er i.

En teller teller også det nye forsøket. Ved levering minst én gang gjør en konsument som øker en teller eller sender et varsel for hver melding, det to ganger for en melding som leveres to ganger. Lagre meldingens forretningsnøkkel sammen med virkningen, i den samme transaksjonen, og hopp over enhver nøkkel som allerede er lagret.

Den andre kjøringen ødelegger det den første bygde. Et steg kan være riktig på en tom database og galt på en full, ved at det setter inn et duplikat, nummererer om identifikatorer eller stoler på en rekkefølge den første kjøringen etterlot seg. En test som starter fra ingenting og kjører én gang, går likevel gjennom. Kjør hele jobben to ganger og krev at den andre kjøringen ikke endrer noe, slik pipelinetesten ovenfor gjør.

Les mer

- RFC 9110, section 9.2.2: Idempotent Methods — definisjonen i HTTP, de idempotente metodene og reglene for automatiske nye forsøk.
- RFC 5789: PATCH Method for HTTP — hvorfor PATCH verken er trygg eller idempotent, og hvordan en betinget forespørsel gjør den trygg å gjenta.
- Ansible: Introduction to playbooks — ønsket tilstand, idempotens og check mode med Ansibles egne ord.
- Ansible: command module — valgene `creates` og `removes`, som gjør en kommando trygg å kjøre på nytt.
- PostgreSQL: INSERT — `ON CONFLICT` og hva `RETURNING` gir tilbake når en rad allerede finnes.

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 betyr idempotent?» 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/idempotent/