Engineer

Redis

et minnebasert lager for ting som lever kort

Hva det er

Redis er en databaseserver som holder hele datasettet sitt i minnet som typede strukturer, og som lar mange applikasjonsprosesser lese og endre dem over nettverket, én kort kommando om gangen. Prosjektet beskriver seg selv som en datastrukturserver (data structure server). Utgaven som er fritt tilgjengelig, heter Redis Open Source, og Redis Ltd selger i tillegg Redis Cloud og Redis Software under kommersielle lisenser.

Hver nøkkel holder én verdi av én type. Strenger er rene bytes og fungerer også som tellere, hasher er poster med felt og verdier, lister holder på rekkefølgen elementene kom i, sett holder unike medlemmer, og sorterte sett rangerer medlemmene sine etter en poengsum. Listen over typer fortsetter med strømmer, JSON-dokumenter, geospatiale indekser, tidsserier, vektorsett og probabilistiske strukturer som Bloom-filtre og HyperLogLog.

Kommandoene virker direkte på disse typene. `INCR` legger én til en teller og returnerer den nye verdien, `SADD` legger medlemmer inn i et sett og `SPOP` tar ett ut, og hvilken som helst nøkkel kan få en levetid (timeout), og når den er ute, sletter Redis nøkkelen selv. Redis kaller en slik nøkkel flyktig (volatile). Når flere steg må skje som ett, kjører et Lua-skript inne i serveren, og Redis garanterer at ingenting annet kjører mens det gjør det.

Minnet er der dataene bor, og disk er valgfritt. Redis kan skrive øyeblikksbilder (snapshots), kjent som RDB-filer, eller logge hver skriving til en fil det bare legges til i (append-only file, AOF), som spilles av på nytt ved oppstart, eller gjøre begge deler, eller ingen av dem. Replikaer kopierer en primærnode og kan betjene lesing. Et tak på minnet, `maxmemory`, og en policy for utkasting (eviction policy) avgjør hva som skjer når dataene vokser ut av plassen de fikk.

Lisensen endret seg med Redis 8. Fra den utgivelsen av tilbys Redis Open Source med et valg mellom tre lisenser: RSALv2, SSPLv1 og AGPLv3. De to første er lisenser med tilgjengelig kildekode (source-available), og AGPLv3 er alternativet som er åpen kildekode.

Bruk begrepet i en beslutning

Bruk «Hva er Redis?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Finne spørringene og de manglende indeksene bak et system som har blitt tyngre for hvert år, og rette dem som betyr noe.

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. 30 dagers strømdata på 180 ms — 180 ms for å aggregere en måned med måleravlesninger

30 dagers strømdata på 180 ms Ta kontakt →

Hvorfor det er viktig

En webapplikasjon kjører sjelden som én prosess. Bak en lastbalanserer står flere instanser, hver med sitt eget minne, og alt som holdes i det minnet, som et antall innloggingsforsøk, en økt eller en side lagret i cache, finnes bare i prosessen som tilfeldigvis behandlet forespørselen. En grense som telles på den måten, kan unngås ved å havne på en annen instans. Redis gir alle instansene ett felles sted for slik tilstand, raskt nok til at hver forespørsel kan sjekke det.

De atomiske kommandoene betyr like mye som farten. To instanser som hver leser en teller, legger til én og skriver den tilbake, kan begge skrive det samme tallet, og en økning forsvinner. `INCR` gjør lesingen og skrivingen inne i serveren som ett steg, så hundre samtidige forespørsler etterlater telleren nøyaktig hundre høyere. Et Lua-skript utvider den samme garantien til en kort sekvens: en sjekk og en endring som ikke må deles opp, som å bekrefte at det fortsatt er plass og ta plassen, kjøres som én operasjon.

Utløp er den tredje delen. Et vindu for ratebegrensning, en økt eller en engangskode bør forsvinne når tiden er ute, og med en levetid på nøkkelen er oppryddingen lagerets oppgave og ikke en planlagt jobbs.

Prisen er varighet (durability), og den er verdt å kjenne i tall før du velger. Med append-only-filen og standardpolicyen med én fsync i sekundet kan et krasj koste omtrent ett sekund med skrivinger. Med bare øyeblikksbilder, som som regel tas med noen minutters mellomrom, sier dokumentasjonen at du må være forberedt på å miste de siste minuttene med data. Replikering er asynkron som standard,

så en primærnode kan bekrefte en skriving og stoppe før noen replika har mottatt den. Det gjør Redis til et godt hjem for tilstand som kan bygges opp igjen, eller mistes uten skade, og et dårlig hjem for den eneste kopien av en ordre, en betaling eller en saldo. Designspørsmålet som betyr mest, er hva applikasjonen gjør når Redis ikke kan nås, eller har glemt noe den fikk beskjed om.

Slik ser det ut i praksis

Delt tilstand for grenser, en rask sperre foran en database og en cache i et provisjonert miljø er de tre rollene Redis har i arbeidet her.

Den første rollen finnes i en offentlig katalog over selskaper og fagfolk, der 22 ratebegrensere holder tilstanden sin i Redis slik at en grense gjelder på tvers av alle instanser i stedet for å leve i én prosess, der den ville vært lett å slippe unna. Hver ratebegrenser er bygd for stien den beskytter, fordi et innloggingsforsøk, et søk og en masseeksport blir misbruk ved helt ulike rater. Valget som avgjør mest, er hva som skjer når Redis mangler: ratebegrenserne feiler lukket (fail closed), og trafikk avvises i stedet for å slippes gjennom. Siden sier rett ut hva det koster. Et Redis-problem blir et problem brukerne ser, akseptert med vilje, fordi alternativet er at et Redis-problem blir et faktureringsproblem.

Den andre rollen beskrives i et notat om å selge et begrenset lager, der en Redis-teller som telles ned av et Lua-skript slipper inn omtrent like mange kjøpere som det finnes enheter, og avviser alle andre på mikrosekunder. Skriptet gjør sjekken og nedtellingen til ett steg. Notatet er nøye med hva sperren kan love. Redis replikerer asynkront, så etter en failover kan den nye primærnoden ha glemt noen innslipp og sluppet noen få for mange gjennom. En betinget `UPDATE` i databasen avgjør fortsatt hvert salg, og en avstemmingsjobb nullstiller sperren fra tallet i databasen. Når enhetene er få og forskjellige, som nummererte seter, kan id-ene deres ligge i et Redis-sett, og `SPOP` deler dem ut én om gangen uten noen teller å slåss om.

Den tredje er den mest hverdagslige. Et Azure-oppsett definert i Bicep bygger hvert miljø fra de samme malene, testing, staging og product likt, og hvert av dem har Redis som cache ved siden av en PostgreSQL Flexible Server, Container Apps som kjører api og nettstedet, og Front Door med en WAF-policy. Fordi Redis er deklarert i malene sammen med alt annet, er en endring av den en pull request som enhver annen endring i et miljø.

Hvor det går galt

Det meste av trøbbelet med Redis kommer av at et raskt, glemsomt lager behandles som et varig, eller av antakelsen om at det husker det det ble fortalt om tid.

En nøkkel som skulle utløpe, blir liggende for alltid. `SET` erstatter verdien og forkaster levetiden nøkkelen hadde, så kode som frisker opp en verdi i cache med en vanlig `SET`, gjør en oppføring på ti minutter til en permanent oppføring:

```
> SET session:42 "v1" EX 600
OK
> SET session:42 "v2"
OK
> TTL session:42
(integer) -1
```

Send med levetiden ved hver skriving som trenger en, eller legg til `KEEPTTL`, tilgjengelig siden 6.0.0, for å beholde den gamle. Den samme lekkasjen skjuler seg i en teller bygd av `INCR` fulgt av en separat `EXPIRE`: kommer den andre kommandoen aldri fram, utløper nøkkelen aldri. Send begge inne i én blokk med `MULTI` og `EXEC`, eller i ett Lua-skript.

Skrivinger begynner å returnere feil mens lesing fortsatt virker. Dataene har nådd `maxmemory`, og utkastingspolicyen er `noeviction`, eller en `volatile-` -policy på et datasett der ingen nøkkel har levetid, noe som oppfører seg på samme måte. Velg policy etter jobben: en `allkeys-` -policy for en ren cache, der hvilken som helst nøkkel kan forsvinne, og en `volatile-` -policy bare når nøkler i cache deler instans med nøkler som må bli. For den blandingen er rådet i dokumentasjonen å vurdere to separate instanser.

Skrivinger Redis bekreftet, mangler etter en omstart eller en failover. Årsaken er et krasj innenfor fsync-vinduet, eller en replika som ble forfremmet før skrivingen nådde den. Slik er Redis designet, så løsningen ligger i hva som lagres der. Lagre alt som må overleve, i en database med varige commits, sett Redis foran den som sperre eller cache, og avstem de to fra databasens side, slik notatet om lageret gjør.

Hver klient stopper opp i samme øyeblikk. Et langt Lua-skript kjører, og Redis blokkerer all annen aktivitet på serveren så lenge et skript kjører. Hold skriptene korte og avgrensede, og hold løkker over store samlinger utenfor dem.

En ratebegrenser slipper trafikk gjennom under et avbrudd. Klientkoden behandler en feil fra Redis som «grensen er ikke nådd», så ratebegrenseren åpner nøyaktig når lageret dens ikke kan nås. Bestem feilmodusen med vilje, og test den ved å stoppe Redis: en ratebegrenser som vokter en kostbar sti, bør avvise, slik de 22 ratebegrenserne over gjør, med den kostnaden at et Redis-problem blir synlig for brukerne.

Les mer

- Develop with Redis — prosjektets startpunkt: kommandoer å prøve, klientbiblioteker per språk og datatypene.
- Redis persistence — øyeblikksbilder, append-only-filen, fsync-policyer og hva hver av dem kan miste.
- Redis replication — hvordan replikaer følger en primærnode, og hvorfor det fortsatt finnes et vindu for datatap.
- Key eviction — `maxmemory`, hver utkastingspolicy og råd om hvordan du velger.
- EXPIRE — levetid på nøkler, og hvilke kommandoer som fjerner den eller lar den stå.
- Scripting with Lua — å kjøre et skript atomisk på serveren, og hva det blokkerer imens.

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 er Redis?» 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/redis/