Engineer

Redis

🧮 et hukommelseslager til ting, der kun skal gemmes kort

Hvad det er

Redis er en databaseserver, der holder hele sit datasæt i hukommelsen som typede strukturer og lader mange programprocesser læse og ændre dem over netværket, én kort kommando ad gangen. Projektet beskriver sig selv som en datastrukturserver, på engelsk a data structure server. Den frit tilgængelige udgave hedder Redis Open Source, og Redis Ltd sælger desuden Redis Cloud og Redis Software under kommercielle licenser.

Hver nøgle rummer én værdi af én type. Strenge er rå bytes og fungerer også som tællere, hashes er poster med felter og værdier, lister bevarer den rækkefølge, elementerne kom i, sæt rummer unikke medlemmer, og sorterede sæt rangerer deres medlemmer efter en score. Listen over typer fortsætter med streams, JSON-dokumenter, geospatiale indeks, tidsserier, vektorsæt og probabilistiske strukturer som Bloom-filtre og HyperLogLog.

Kommandoerne arbejder direkte på typerne. `INCR` lægger én til en tæller og returnerer den nye værdi, `SADD` lægger medlemmer i et sæt, og `SPOP` tager ét ud. Enhver nøgle kan få en udløbstid, hvorefter Redis selv sletter den, og Redis kalder sådan en nøgle volatil. Når flere trin skal ske som ét, kører et Lua-script inde i serveren, og Redis garanterer, at intet andet kører imens.

Hukommelsen er der, hvor data bor, og disken er valgfri. Redis kan skrive øjebliksbilleder, kaldet RDB-filer, eller logge hver skrivning i en append-only-fil (AOF), som den afspiller ved opstart, eller gøre begge dele, eller ingen af dem. Replikaer kopierer en primær og kan besvare læsninger. Et loft over hukommelsen, `maxmemory`, og en eviction-politik afgør, hvad der sker, når data vokser ud over den plads, de har fået.

Licensen skiftede med Redis 8. Fra den udgave tilbydes Redis Open Source under et valg mellem tre licenser: RSALv2, SSPLv1 og AGPLv3. De to første er source-available-licenser, og AGPLv3 er open source-muligheden.

Brug begrebet i en beslutning

Brug »Hvad er Redis?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. At finde de forespørgsler og manglende indeks bag et system, der er blevet tungere hvert år, og rette dem, der betyder noget.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. 30 dages eldata på 180 ms — 180 ms til at aggregere en måneds måleraflæsninger

30 dages eldata på 180 ms Kontakt os →

Hvorfor det betyder noget

Et webprogram kører sjældent som én proces. Bag en load balancer står flere instanser, hver med sin egen hukommelse, og alt, hvad der ligger i den hukommelse — et antal loginforsøg, en session, en cachet side — findes kun i den proces, der tilfældigvis tog sig af forespørgslen. En grænse, der tælles på den måde, kan man slippe uden om ved at ramme en anden instans. Redis giver alle instanserne ét fælles sted til den slags tilstand, hurtigt nok til at spørge ved hver eneste forespørgsel.

De atomare kommandoer betyder lige så meget som hastigheden. To instanser, der hver læser en tæller, lægger én til og skriver den tilbage, kan begge skrive det samme tal, og en optælling forsvinder. `INCR` gør læsning og skrivning inde i serveren som ét trin, så hundrede samtidige forespørgsler efterlader tælleren præcis hundrede højere. Et Lua-script udvider den garanti til en kort række trin: et tjek og en ændring, der ikke må skilles ad, som at bekræfte, at der stadig er plads, og tage pladsen, kører som ét.

Udløbstiden er den tredje brik. Et vindue for en rate limit, en session eller en engangskode bør forsvinde, når tiden er gået, og med en udløbstid på nøglen bliver oprydningen lagerets opgave frem for et planlagt jobs.

Prisen er holdbarhed, og den er værd at kende i tal, før man vælger. Med append-only-filen og dens standardpolitik med én fsync i sekundet kan et nedbrud koste omkring et sekunds skrivninger. Med øjebliksbilleder alene, som normalt tages med nogle minutters mellemrum, siger dokumentationen, at man skal være forberedt på at miste de seneste minutters data. Replikering er som

standard asynkron, så en primær kan bekræfte en skrivning og gå ned, før nogen replika har modtaget den. Det gør Redis til et godt hjem for tilstand, der kan genopbygges eller mistes uden skade, og til et dårligt hjem for den eneste kopi af en ordre, en betaling eller en saldo. Det designspørgsmål, der betyder mest, er, hvad programmet gør, når Redis ikke svarer eller har glemt noget, det fik at vide.

Sådan ser det ud i praksis

Et offentligt katalog over virksomheder og fagfolk bruger Redis som fælles tilstand for sine grænser: 22 rate limitere holder deres tilstand i Redis, så en grænse gælder på tværs af alle instanser i stedet for at bo i én proces, hvor den ville være triviel at slippe uden om. Hver limiter er bygget til den sti, den beskytter, for et loginforsøg, en søgning og en bulk-eksport bliver misbrug ved meget forskellige hastigheder. Det afgørende valg er, hvad der sker, når Redis mangler: limiterne er fail-closed, og trafik afvises frem for at blive vinket igennem. Siden nævner prisen direkte. Et Redis-problem bliver et problem, brugerne kan se, accepteret med vilje, fordi alternativet er, at et Redis-problem bliver et regningsproblem.

Redis kan også stå som en hurtig port foran en database. I en note om at sælge begrænset lager lukker en tæller i Redis, nedskrevet af et Lua-script omtrent lige så mange købere ind, som der er enheder, og afviser alle andre på mikrosekunder. Scriptet gør tjek og nedskrivning til ét trin. Noten er omhyggelig med, hvad porten kan love. Redis replikerer asynkront, så efter en failover kan den nye primære have glemt nogle adgange og lukket lidt for mange ind. Et betinget `UPDATE` i databasen afgør stadig hvert salg, og en afstemningsproces nulstiller porten ud fra databasens tal. Når enhederne er få og unikke, som nummererede pladser, kan deres id'er ligge i et Redis-sæt, og `SPOP` deler dem ud én ad gangen, uden en tæller at slås om.

Oftest er Redis en cache i et provisioneret miljø. En Azure-opsætning defineret i Bicep bygger hvert miljø ud fra de samme templates, testing, staging og product, og hvert af dem har Redis til caching ved siden af en PostgreSQL Flexible Server, de Container Apps, der kører api'et og websitet, og Front Door med en WAF-politik. Fordi cachen er erklæret i templates sammen med alt det andet, er en ændring af den en pull request som enhver anden ændring af et miljø.

Hvor det går galt

Det meste besvær med Redis kommer af at behandle et hurtigt, glemsomt lager som et holdbart, eller af at regne med, at det husker, hvad det fik at vide om tid.

En nøgle, der burde udløbe, bliver liggende for evigt. `SET` erstatter værdien og kasserer den udløbstid, nøglen havde, så kode, der fornyer en cachet værdi med et almindeligt `SET`, gør en post på ti minutter permanent:

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

Angiv udløbstiden ved hver skrivning, der skal have en, eller tilføj `KEEPTTL`, som findes siden 6.0.0, for at beholde den gamle. Den samme læk gemmer sig i en tæller bygget af `INCR` efterfulgt af et separat `EXPIRE`: når den anden kommando aldrig når frem, udløber nøglen aldrig. Send begge i én `MULTI` - og `EXEC` -blok eller i ét Lua-script.

Skrivninger begynder at returnere fejl, mens læsninger stadig virker. Data har nået `maxmemory`, og eviction-politikken er `noeviction`, eller en `volatile-` -politik på et datasæt, hvor ingen nøgle har en udløbstid, hvilket virker på samme måde. Vælg politikken efter opgaven: en `allkeys-` -politik til en ren cache, hvor enhver nøgle må forsvinde, og en `volatile-` -politik kun, når cachede nøgler deler instans med nøgler, der skal blive. Til den blanding råder dokumentationen til at overveje to separate instanser.

Skrivninger, som Redis bekræftede, mangler efter en genstart eller en failover. Et nedbrud ramte inden for fsync-vinduet, eller en replika blev forfremmet, før skrivningen nåede frem til den. Sådan er Redis bygget, så løsningen ligger i, hvad man gemmer der. Læg alt, hvad der skal overleve, i en database med holdbare commits, sæt Redis foran som port eller cache, og afstem de to fra databasens side, som noten om lager gør.

Alle klienter går i stå på samme tid. Et langt Lua-script kører, og Redis blokerer al anden serveraktivitet, så længe et script kører. Hold scripts korte og afgrænsede, og hold løkker over store samlinger ude af dem.

En limiter vinker trafik igennem under et udfald. Klientkoden behandler en fejl fra Redis som »grænsen er ikke nået«, så limiteren åbner præcis, når dens lager er utilgængeligt. Vælg fejltilstanden med vilje, og test den ved at stoppe Redis: en limiter, der vogter en dyr sti, bør afvise, som de 22 limitere ovenfor gør, med den pris, at et Redis-problem bliver synligt for brugerne.

Læs mere

- Develop with Redis — projektets indgang: kommandoer at prøve, klientbiblioteker efter sprog og datatyperne.
- Redis persistence — øjebliksbilleder, append-only-filen, fsync-politikker og hvad hver af dem kan miste.
- Redis replication — hvordan replikaer følger en primær, og hvorfor der altid er et vindue for datatab.
- Key eviction — `maxmemory`, alle eviction-politikker og råd om at vælge én.
- EXPIRE — udløbstider på nøgler, og hvilke kommandoer der fjerner dem eller lader dem stå.
- Scripting with Lua — at køre et script atomart på serveren, og hvad det blokerer imens.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er Redis?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/redis/