**Engineer**

# Redis

ett minnesbaserat lager för saker som lever kort

## Vad det är

Redis är en databasserver som håller hela sin datamängd i minnet som typade strukturer och låter många applikationsprocesser läsa och ändra dem över nätverket, ett kort kommando i taget. Projektet beskriver sig självt som en ”data structure server”, en server för datastrukturer. Den fritt tillgängliga utgåvan heter Redis Open Source, och Redis Ltd säljer dessutom Redis Cloud och Redis Software under kommersiella licenser.

Varje nyckel rymmer ett värde av en enda typ. Strängar är rena byte och fungerar också som räknare, hashar är poster med fält och värden, listor behåller den ordning som elementen kom i, mängder rymmer unika medlemmar och sorterade mängder rangordnar sina medlemmar efter en poäng. Listan över typer fortsätter med strömmar, JSON-dokument, geospatiala index, tidsserier, vektormängder och probabilistiska strukturer som Bloomfilter och HyperLogLog.

Kommandona arbetar direkt på de typerna. `INCR` lägger till ett till en räknare och returnerar det nya värdet, `SADD` lägger in medlemmar i en mängd och `SPOP` tar ut en, och varje nyckel kan bära en tidsgräns efter vilken Redis raderar den på egen hand. Redis kallar en sådan nyckel volatile, flyktig. När flera steg måste ske som ett körs ett Lua-skript inne i servern, och Redis garanterar att inget annat körs medan skriptet pågår.

Minnet är där datan bor, och disken är valfri. Redis kan skriva ögonblicksbilder, kända som RDB-filer, eller logga varje skrivning till en fil som bara växer, en append-only file (AOF), och spela upp den vid start, eller göra båda, eller ingetdera. Repliker kopierar en primärnod och kan besvara läsningar. Ett tak för minnet, `maxmemory`, och en policy för vad som ska kastas ut, i dokumentationen kallad eviction policy, avgör vad som händer när datan växer ur det utrymme den fick.

Licensen ändrades med Redis 8. Från den versionen erbjuds Redis Open Source under ett val mellan tre licenser: RSALv2, SSPLv1 och AGPLv3. De två första är licenser med tillgänglig källkod, source-available, och AGPLv3 är alternativet med öppen källkod.

Använd begreppet i ett beslut

Använd ”Vad är Redis?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hittar databasfrågorna och de saknade indexen bakom ett system som har blivit tyngre för varje år, och åtgärdar dem som spelar roll.

**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. 30 dagars eldata på 180 ms — 180 ms för att aggregera en månads mätaravläsningar

**30 dagars eldata på 180 ms** Hör av er →

## Varför det spelar roll

En webbapplikation körs sällan som en enda process. Bakom en lastbalanserare står flera instanser, var och en med sitt eget minne, och allt som hålls i det minnet — ett antal inloggningsförsök, en session, en cachad sida — finns bara i den process som råkade hantera begäran. En gräns som räknas så går att undkomma genom att hamna på en annan instans. Redis ger alla instanser ett gemensamt ställe för den sortens tillstånd, snabbt nog att anropa vid varje begäran.

De atomära kommandona spelar lika stor roll som farten. Två instanser som var för sig läser en räknare, lägger till ett och skriver tillbaka kan båda skriva samma tal, och en ökning försvinner. `INCR` gör läsningen och skrivningen inne i servern som ett enda steg, så att hundra samtidiga begäranden lämnar räknaren exakt hundra högre. Ett Lua-skript utsträcker samma garanti till en kort sekvens: en kontroll och en ändring som inte får delas upp, som att bekräfta att det fortfarande finns plats och ta platsen, körs som en.

Att nycklar går ut av sig själva är den tredje delen. Ett fönster för hastighetsbegränsning, en session eller en engångskod borde försvinna när dess tid är ute, och med en tidsgräns på nyckeln blir städningen lagrets uppgift snarare än ett schemalagt jobbs.

Priset är beständigheten, och det är värt att känna till det i siffror innan man väljer. Med append-only-filen och dess standardpolicy med en fsync per sekund kan en krasch kosta ungefär en sekunds skrivningar. Med enbart ögonblicksbilder,

som oftast tas med några minuters mellanrum, säger dokumentationen att man ska vara beredd på att förlora de senaste minuternas data. Replikeringen är asynkron som standard, så en primärnod kan kvittera en skrivning och stanna innan någon replik har tagit emot den. Det gör Redis till ett bra hem för tillstånd som kan byggas upp igen, eller förloras utan skada, och ett dåligt hem för den enda kopian av en order, en betalning eller ett saldo. Det designbeslut som väger tyngst gäller vad applikationen gör när Redis inte går att nå, eller har glömt något som den fick veta.

## Så ser det ut i praktiken

Redis har här tre roller: delat tillstånd för 22 hastighetsbegränsare, en snabb grind framför en databas och en cache i varje provisionerad miljö.

Den första är en publik katalog över företag och yrkespersoner, där 22 hastighetsbegränsare håller sitt tillstånd i Redis så att en gräns gäller över alla instanser i stället för att leva i en enda process, där den vore trivial att komma runt. Varje begränsare är byggd för den väg den skyddar, eftersom ett inloggningsförsök, en sökning och en massexport blir missbruk vid helt olika anropsfrekvenser. Det avgörande valet är vad som händer när Redis saknas: begränsarna stänger vid fel, och trafiken nekas i stället för att vinkas igenom. Sidan anger kostnaden rakt ut. Ett Redis-problem blir ett problem som användarna ser, accepterat med avsikt, eftersom alternativet är att ett Redis-problem blir ett faktureringsproblem.

Den andra är en anteckning om att sälja ett begränsat lager, där en Redis-räknare som räknas ned av ett Lua-skript släpper in ungefär lika många köpare som det finns enheter och avvisar alla andra på mikrosekunder. Skriptet gör kontrollen och nedräkningen till ett enda steg. Anteckningen är noga med vad grinden kan lova. Redis replikerar asynkront, så efter en failover kan den nya primärnoden ha glömt några insläpp och släppt igenom några för många. En villkorad `UPDATE` i databasen avgör fortfarande varje försäljning, och en avstämningsprocess återställer grinden från databasens tal. När enheterna är få och unika, som numrerade sittplatser, kan deras id ligga i en Redis-mängd, och `SPOP` delar ut dem en i taget utan någon räknare att slåss om.

Den tredje är den mest vardagliga. En infrastruktur i Azure som definieras i Bicep bygger varje miljö ur samma mallar, testing, staging och product på samma sätt, och var och en har Redis för cachning bredvid en PostgreSQL Flexible Server, de Container Apps som kör api:et och webbplatsen och Front Door med en WAF-

policy. Eftersom cachen deklareras i mallarna tillsammans med allt annat är en ändring av den en pull request, precis som vilken annan ändring av en miljö som helst.

## Var det går fel

Det mesta besväret med Redis kommer av att ett snabbt, glömskt lager behandlas som ett beständigt, eller av antagandet att det minns vad det fick veta om tid.

**En nyckel som skulle gå ut blir kvar för alltid.** `SET` ersätter värdet och kastar den tidsgräns som nyckeln hade, så kod som förnyar ett cachat värde med en vanlig `SET` gör en post på tio minuter till en permanent:

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

Skicka med tidsgränsen vid varje skrivning som behöver en, eller lägg till `KEEPTTL`, som finns sedan 6.0.0, för att behålla den gamla. Samma läcka gömmer sig i en räknare som byggs av `INCR` följt av ett separat `EXPIRE`: när det andra kommandot aldrig kommer fram går nyckeln aldrig ut. Skicka båda inne i ett och samma block med `MULTI` och `EXEC`, eller i ett enda Lua-skript.

**Skrivningar börjar returnera fel medan läsningar fortfarande fungerar.** Datan har nått `maxmemory`, och policyn är `noeviction`, eller en `volatile-` -policy på en datamängd där ingen nyckel har en tidsgräns, vilket beter sig likadant. Välj policy efter jobbet: en `allkeys-` -policy för en ren cache, där vilken nyckel som helst får försvinna, och en `volatile-` -policy bara när cachade nycklar delar instans med nycklar som måste stanna. För den blandningen råder dokumentationen att överväga två separata instanser.

**Skrivningar som Redis kvitterade saknas efter en omstart eller en failover.** En krasch inträffade inom fsync-fönstret, eller en replik befordrades innan skrivningen nådde den. Så är Redis konstruerat, så lösningen ligger i vad som

läggs där. Lagra allt som måste överleva i en databas med beständiga commits, ställ Redis framför den som en grind eller en cache och stäm av de två från databasens sida, så som anteckningen om lagret gör.

**Varje klient står still i samma ögonblick.** Ett långt Lua-skript körs, och Redis blockerar all annan aktivitet på servern under hela skriptets körtid. Håll skripten korta och begränsade, och håll loopar över stora samlingar utanför dem.

**En begränsare vinkar igenom trafik under ett avbrott.** Klientkoden behandlar ett fel från Redis som ”limit not reached”, gränsen inte nådd, så begränsaren öppnar precis när dess lager inte går att nå. Bestäm felläget med avsikt och testa det genom att stoppa Redis: en begränsare som skyddar en kostsam väg ska neka, så som de 22 begränsarna ovan gör, med kostnaden att ett Redis-problem blir synligt för användarna.

## Läs mer

- Develop with Redis — projektets startpunkt: kommandon att prova, klientbibliotek per språk och datatyperna.
- Redis persistence — ögonblicksbilder, append-only-filen, policyer för fsync och vad var och en kan förlora.
- Redis replication — hur repliker följer en primärnod och varför ett fönster för dataförlust finns kvar.
- Key eviction — `maxmemory`, varje policy för att kasta ut nycklar och råd om hur man väljer en.
- EXPIRE — tidsgränser för nycklar och vilka kommandon som rensar dem eller låter dem vara.
- Scripting with Lua — att köra ett skript atomärt på servern och vad det blockerar under tiden.

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 är Redis?” 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/redis/