Engineer

Replikering

at holde kopier af en database i takt på andre servere

Hvad det er

Replikering holder kopier af en database på andre servere og sender hver kopi alle de ændringer, der laves i originalen, så de holdes i takt. Mange bruger også det engelske ord replication, og hvor sammenhængen kan efterlade tvivl, taler man om databasereplikering. PostgreSQL-dokumentationen kalder den ene server, der tager imod skrivninger, den primære og de servere, der følger den, standbyservere. Redis taler om en master og dens replikaer.

PostgreSQL har to slags, og de kopierer forskellige ting.

Fysisk replikering kopierer write-ahead-loggen (WAL), den log, hver ændring skrives til, før den når datafilerne. En standby afspiller loggen og bliver en blok-for-blok-kopi af hele clusteret. Ved streaming-replikering holder standbyen en forbindelse åben til den primære og modtager WAL-posterne, efterhånden som de skrives, i stedet for at vente på, at færdige WAL-filer bliver sendt over. En standby, der kun afspiller og ikke tager imod forbindelser, før den forfremmes, er en warm standby; en, der også svarer på skrivebeskyttede forespørgsler, er en hot standby.

Logisk replikering kopierer ændringer i rækker. En publisher sender de indsættelser, opdateringer og sletninger videre, der laves i udvalgte tabeller, hver række kendt på sin replica identity, normalt primærnøglen, og en subscriber anvender dem på sine egne tabeller. Fordi den arbejder med rækker og ikke med diskblokke, kan den nøjes med nogle få tabeller frem for hele clusteret, endda over til en anden hovedversion af PostgreSQL.

Streaming-replikering er asynkron som standard. Den primære bekræfter en commit, så snart WAL er på dens egen disk, og sender den videre bagefter; dokumentationen sætter forsinkelsen til typisk under et sekund, når standbyen

kan følge med. Med synkron replikering slået til venter hver commit, til en standby også har WAL.

Brug begrebet i en beslutning

Brug »Hvad er replikering?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Sikkerhedskopier, der er testet og ikke bare antaget at virke, og en nedskrevet plan for den morgen, en server ikke starter igen.

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. Har automatiseret 100 kritiske databackups — 100 kritiske backups af databaserne automatiseret

Har automatiseret 100 kritiske databackups Kontakt os →

Hvorfor det betyder noget

Den første grund er at overleve tabet af en server. Når den primære går ned, kan en standby, der allerede har dataene, forfremmes og overtage trafikken, langt hurtigere end en backup kan lægges tilbage på en ny maskine.

Den anden er at sprede læsningerne. I mange applikationer læser de fleste forespørgsler kun, og en hot standby kan besvare dem, så den primære er fri til skrivningerne, som kun den kan tage imod.

Den tredje er at flytte data. Logisk replikering kan kopiere netop de tabeller, en rapporteringsdatabase har brug for, eller bringe data over på en server med en anden hovedversion af PostgreSQL.

Hver kopi kommer med et valg mellem at vente og at miste. Asynkron replikering koster intet, når der committes, men en standby er altid en smule bagefter: en læsning fra den kan overse en skrivning, der lige er sket, og hvis den primære dør, når commits, den allerede har bekræftet, måske aldrig frem til den standby, der tager over. Synkron replikering lukker hullet og lægger en tur frem og tilbage over netværket ind i hver commit.

Replikering er heller ikke en backup. En standby afspiller trofast en utilsigtet `DELETE` uden `WHERE`, så få øjeblikke senere har alle kopier fejlen. Det kræver stadig en backup at rulle den tilbage.

Sådan ser det ud i praksis

En webshop kører sin database på én primær server på `10.0.0.1` og tilføjer en server mere, `db2`, som streaming-replika. På `db2`, hvor PostgreSQL er installeret og stoppet:

```
pg_basebackup -h 10.0.0.1 -U replicator -D /srv/pgdata \
    -R -C -S db2_slot -P
```

`pg_basebackup` kopierer hele det kørende cluster over i den mappe, `-D` peger på; mappen oprettes, hvis den mangler, og skal ellers være tom. Som standard streamer den samtidig den WAL, der skrives under kopieringen. `-C -S db2_slot` opretter først en replikeringsslot på den primære, så den gemmer hver WAL-post, indtil den nye standby har modtaget den. `-R` opretter filen `standby.signal` og tilføjer forbindelsesindstillingerne, slotten medregnet, til `postgresql.auto.conf`. `-P` viser, hvor langt kopieringen er nået. Rollen `replicator` skal have attributten `REPLICATION` eller være superuser, og den primæres `pg_hba.conf` skal tillade replikeringsforbindelsen.

Når PostgreSQL starter på `db2`, får `standby.signal` den til at køre som standby og streame fra den primære, og da `hot_standby` er slået til som standard, tager den imod skrivebeskyttede forespørgsler, så snart den er konsistent. På den primære viser én række pr. standby, hvordan det går:

```
SELECT application_name, state, sync_state, replay_lsn, replay_lag
FROM   pg_stat_replication;
```

En sund replika viser `streaming` i `state` og `async` i `sync_state`. `replay_lag` er et skøn over, hvor lang tid nye commits var om at blive synlige dér; når standbyen har indhentet det hele, og WAL-aktiviteten stopper, står den på NULL i stedet for nul.

Noten om at optimere databasen, før man sharder den forklarer, hvorfor en butik gør det. Det meste af trafikken er læsning: produktsiden, kurven, ordrehistorikken. Intet af det har brug for den primære, som skal holdes fri til skrivninger. En streaming-replika tager de læsninger, og noten påpeger, at det er et spørgsmål om konfiguration at tilføje én, mens en shard er et projekt. Den

nævner to forbehold. En replika halter en smule bagefter, så det, en bruger lige har skrevet, skal læses fra den primære. Og en lang rapport på en replika kan blive annulleret, når den holder ændringer tilbage, som replikaen skal afspille.

Det kræver én indstilling på den primære at få en commit til at vente på en standby. Hver standby får et navn via `application_name` i sin `primary_conninfo`, og den primære oplister de navne, den må vente på:

```
synchronous_standby_names = 'ANY 1 (db2, db3)'
```

Nu venter hver commit, til dens WAL ligger på disken både på den primære og på mindst én af `db2` og `db3`. `synchronous_commit` kan sættes pr. bruger, pr. database eller pr. transaktion, så ordren, der tager imod betalingen, kan vente, mens en tæller for sidevisninger committer i lokalt tempo. Noten om at sælge den sidste billet præcis én gang viser, hvorfor det betyder noget for lageret. Med asynkron replikering kan en primær bekræfte et salg og dø, før dens replika har det; replikaen forfremmes, salget er væk, og den samme enhed sælges igen. Notens løsning er en synkron standby eller et lager bygget på Raft, og prisen er ekstra latens på hver commit.

Logisk replikering sættes op med to sætninger. Publisheren skal have `wal_level = logical`, og tabellerne skal allerede findes hos subscriberen, fordi deres definitioner ikke bliver kopieret:

```
-- på publisheren
CREATE PUBLICATION shop_pub FOR TABLE orders, customers;

-- på subscriberen
CREATE SUBSCRIPTION shop_sub
    CONNECTION 'host=10.0.0.1 dbname=shop user=replicator'
    PUBLICATION shop_pub;
```

Abonnementet kopierer først de rækker, der allerede ligger i tabellerne, og anvender derefter alle senere ændringer.

Hvor det går galt

En bruger kan ikke se det, vedkommende lige har skrevet. Applikationen skriver til den primære og læser resultatet tilbage fra en replika, der stadig halter et øjeblik bagefter, så en gemt indstilling ser ud til at springe tilbage. Send en brugers læsninger til den primære i et kort stykke tid efter hver skrivning. Hvor data skal være synlige på standbyen i samme øjeblik, som commit melder tilbage, venter `synchronous_commit = remote_apply`, til de synkrone standbyer har afspillet transaktionen, til en højere pris pr. commit.

Lange forespørgsler på replikaen bliver annulleret. Standbyen skal afspille ændringer, der støder sammen med en igangværende forespørgsel: vacuum, der fjerner rækkeversioner, forespørgslen stadig kan se, eller den eksklusive lås, som `DROP TABLE` og mange former for `ALTER TABLE` tager. Afspilningen venter op til `max_standby_streaming_delay`, som standard 30 sekunder, og annullerer så forespørgslen. En replika til rapporter kan få en længere frist og halter så længere bagefter, mens den venter. `hot_standby_feedback = on` stopper de annulleringer, vacuum giver, men dokumentationen advarer om, at det kan få den primære til at svulme op. `pg_stat_database_conflicts` på standbyen tæller annulleringerne efter årsag, og applikationen bør prøve en annulleret forespørgsel igen.

En glemt slot fylder den primæres disk. En slot holder på WAL, til dens standby har modtaget den, også når standbyen er slukket eller for længst væk, og `max_slot_wal_keep_size` er som standard -1, altså ingen grænse. Tjek slottene jævnligt:

```
SELECT slot_name, active, wal_status, safe_wal_size
FROM   pg_replication_slots;
```

Slet en slot, når dens standby er væk. Sætter man `max_slot_wal_keep_size`, får hver slot et loft over, hvad den må holde på; en standby, der kommer længere bagud, finder sin slot ugyldiggjort, og `wal_status` viser `lost`. `idle_replication_slot_timeout`, der som standard er slået fra, ugyldiggør slots, som har været inaktive længere end den angivne tid.

Skrivningerne går i stå, når en synkron standby gør. Er der kun navngivet én standby, og er den nede, venter hver commit på den primære, måske for evigt. Navngiv flere kandidater end det antal, der kræves, som i `ANY 1 (db2, db3)`. Kan

det ikke lade sig gøre, så sænk antallet eller tøm `synchronous_standby_names`, og genindlæs konfigurationen.

To servere tror begge, at de er den primære. PostgreSQL forfremmer en standby, når man beder om det, med `pg_ctl promote` eller `SELECT pg_promote()`, men har ingen software, der selv opdager, at den primære er faldet ud, og intet forhindrer den gamle i at komme tilbage og tage imod skrivninger. Dokumentationen advarer om, at det fører til forvirring og i sidste ende til datatab. Luk den gamle primære ude, ved at slukke den eller kappe dens netværk, før standbyen forfremmes, og genopbyg den som standby, før den får lov at komme med igen.

Logisk replikering efterlader noget. Skemaændringer replikeres ikke, så en kolonne, der tilføjes hos publisheren, stopper replikeringen, til subscriberen også har den; læg tilføjende ændringer på subscriberen først. Sekvenser replikeres heller ikke: rækkerne kommer med deres id'er, men subscriberens sekvenser står stadig på deres startværdi, og efter et skifte fejler indsættelser på dublerede nøgler. Sæt hver sekvens forbi den højeste værdi i brug, før der skiftes.

En Redis-master genstarter tom og tager sine replikaer med sig. Er persistens slået fra på masteren, og genstarter noget den automatisk, kommer den tilbage med et tomt datasæt, og replikaerne kopierer det og sletter deres egne data. Hold persistens slået til, eller lad være med at lade masteren genstarte automatisk.

Læs mere

- High availability — PostgreSQL-dokumentationens overblik over primære servere og standbyservere, synkron og asynkron replikering og failover.
- Log-Shipping Standby Servers — streaming-replikering, replikeringsslots og opsætning af synkron replikering.
- Hot Standby — skrivebeskyttede forespørgsler på en standby, de konflikter, der annullerer dem, og indstillingerne, der afvejer forsinkelse mod annulleringer.
- pg_basebackup — kopiering af et kørende cluster for at starte en standby, med alle de tilvalg, der er brugt ovenfor.
- Restrictions — det, logisk replikering ikke tager med: skema, sekvenser, large objects og alt andet end tabeller.
- Redis replication — hvordan Redis-replikaer følger en master, hvorfor replikeringen er asynkron, og faren ved at genstarte uden persistens.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er replikering?« 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/replication/