Engineer

Replikering

att hålla kopior av en databas i takt på andra servrar

Vad det är

Replikering håller kopior av en databas på andra servrar och skickar varje kopia alla ändringar som görs i originalet, så att de hålls i takt. Många använder också det engelska ordet replication, och där sammanhanget lämnar utrymme för tvivel heter det databasreplikering. PostgreSQL-dokumentationen kallar den enda server som tar emot skrivningar primary och servrarna som följer den standby; på svenska säger man gärna primärnoden och standbyservrarna. Redis talar om en master och dess repliker.

PostgreSQL har två sorter, och de kopierar olika saker.

Fysisk replikering kopierar write-ahead-loggen (WAL), loggen som varje ändring skrivs till innan den når datafilerna. En standby spelar upp loggen och blir en block-för-block-kopia av hela klustret. Vid strömmande replikering håller standbyn en anslutning öppen mot primärnoden och tar emot WAL-posterna allt eftersom de skrivs, i stället för att vänta på att färdiga WAL-filer skickas över.
En standby som bara spelar upp och inte tar emot anslutningar förrän den befordras är en warm standby; en som också svarar på skrivskyddade frågor är en hot standby.

Logisk replikering kopierar ändringar i rader. En publisher skickar vidare de infogningar, uppdateringar och borttagningar som görs i utvalda tabeller, varje rad identifierad genom sin replica identity, oftast primärnyckeln, och en subscriber tillämpar dem på sina egna tabeller. Eftersom den arbetar med rader och inte diskblock kan den nöja sig med några få tabeller i stället för hela klustret, till och med över till en annan huvudversion av PostgreSQL.

Strömmande replikering är asynkron som standard. Primärnoden bekräftar en commit så snart WAL ligger på dess egen disk och skickar den vidare efteråt; dokumentationen anger fördröjningen till normalt under en sekund när standbyn hinner med. Med synkron replikering påslagen väntar varje commit tills en standby också har WAL.

Använd begreppet i ett beslut

Använd ”Vad är replikering?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Säkerhetskopior som testas i stället för att tas för givna, och en skriftlig plan för den morgon då en server inte startar igen.

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. Automatiserade 100 kritiska säkerhetskopior — 100 automatiserade kritiska säkerhetskopior av databaserna

Automatiserade 100 kritiska säkerhetskopior Hör av er →

Varför det spelar roll

Det första skälet är att överleva förlusten av en server. När primärnoden går ner kan en standby som redan har datan befordras och ta över trafiken, långt snabbare än en backup kan läsas tillbaka på en ny maskin.

Det andra är att sprida läsningarna. I många applikationer läser de flesta frågor bara, och en hot standby kan besvara dem, så att primärnoden är ledig för skrivningarna, som bara den kan ta emot.

Det tredje är att flytta data. Logisk replikering kan kopiera just de tabeller en rapportdatabas behöver, eller föra över data till en server med en annan huvudversion av PostgreSQL.

Varje kopia kommer med ett val mellan att vänta och att förlora. Asynkron replikering kostar ingenting när man committar, men en standby ligger alltid lite efter: en läsning från den kan missa en skrivning som just har skett, och om primärnoden dör kan commits som den redan har bekräftat aldrig nå fram till den standby som tar över. Synkron replikering stänger den luckan och lägger en tur fram och tillbaka över nätverket i varje commit.

Replikering är inte heller en backup. En standby spelar troget upp en oavsiktlig `DELETE` utan `WHERE`, så några ögonblick senare har alla kopior felet. Det krävs fortfarande en backup för att ångra det.

Så ser det ut i praktiken

En webbutik kör sin databas på en primärnod på `10.0.0.1` och lägger till en server till, `db2`, som strömmande replik. På `db2`, där PostgreSQL är installerat och stoppat:

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

`pg_basebackup` kopierar hela det körande klustret till katalogen som `-D` pekar på; katalogen skapas om den saknas och måste annars vara tom. Som standard strömmar den samtidigt med sig den WAL som skrivs under kopieringen. `-C -S db2_slot` skapar först en replikeringsslot på primärnoden, så att den sparar varje WAL-post tills den nya standbyn har tagit emot den. `-R` skapar filen `standby.signal` och skriver anslutningsinställningarna i `postgresql.auto.conf`, där även slotnamnet hamnar. `-P` visar förloppet. Rollen `replicator` behöver attributet `REPLICATION` eller superanvändarrättigheter, och primärnodens `pg_hba.conf` måste tillåta replikeringsanslutningen.

När PostgreSQL startar på `db2` får `standby.signal` den att gå som standby och strömma från primärnoden, och eftersom `hot_standby` är påslaget som standard tar den emot skrivskyddade frågor så snart den är konsistent. På primärnoden visar en rad per standby hur det går:

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

En frisk replik visar `streaming` i `state` och `async` i `sync_state`. `replay_lag` är en uppskattning av hur lång tid nya commits tog på sig att bli synliga där; när standbyn har kommit ikapp och WAL-aktiviteten upphör står den på NULL i stället för noll.

Anteckningen om att optimera databasen innan du shardar den förklarar varför en butik gör så. Det mesta av trafiken är läsning: produktsidan, varukorgen, orderhistoriken. Inget av det behöver primärnoden, som måste hållas fri för skrivningar. En strömmande replik tar de läsningarna, och anteckningen påpekar att det är konfiguration att lägga till en, där en shard är ett projekt. Den nämner

två förbehåll. En replik ligger lite efter, så det en användare just har skrivit bör läsas från primärnoden. Och en lång rapport på en replik kan avbrytas när den håller upp ändringar som repliken ska spela upp.

Det krävs en inställning på primärnoden för att låta en commit vänta på en standby. Varje standby får ett namn genom `application_name` i sin `primary_conninfo`, och primärnoden listar de namn den får vänta på:

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

Nu väntar varje commit tills dess WAL ligger på disk både på primärnoden och på minst en av `db2` och `db3`. `synchronous_commit` kan sättas per användare, per databas eller per transaktion, så att ordern som tar betalt kan vänta medan en räknare för sidvisningar committar i lokal takt. Anteckningen om att sälja den sista biljetten exakt en gång visar varför det spelar roll för lagersaldot. Med asynkron replikering kan en primärnod bekräfta en försäljning och dö innan repliken har fått den; repliken befordras, försäljningen är borta, och samma enhet säljs igen. Anteckningens lösning är en synkron standby eller ett lagringssystem byggt på Raft, och priset är extra latens vid varje commit.

Logisk replikering sätts upp med två satser. Publishern behöver `wal_level = logical`, och tabellerna måste redan finnas hos subscribern, eftersom deras definitioner inte kopieras:

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

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

Prenumerationen kopierar först de rader som redan finns i tabellerna och tillämpar sedan alla senare ändringar.

Var det går fel

Användaren ser inte det som just sparades. Applikationen skriver till primärnoden och läser tillbaka resultatet från en replik som fortfarande ligger ett ögonblick efter, så en sparad inställning verkar hoppa tillbaka. Skicka en användares läsningar till primärnoden en kort stund efter varje skrivning. Där data måste synas på standbyn i samma ögonblick som commit returnerar väntar `synchronous_commit = remote_apply` tills de synkrona standbyservrarna har spelat upp transaktionen, till ett högre pris per commit.

Långa frågor på repliken avbryts. Standbyn måste spela upp ändringar som krockar med en fråga som körs: vacuum som tar bort radversioner som frågan fortfarande kan se, eller det exklusiva lås som `DROP TABLE` och många former av `ALTER TABLE` tar. Uppspelningen väntar upp till `max_standby_streaming_delay` , 30 sekunder som standard, och avbryter sedan frågan. En replik för rapporter kan få en längre frist och halkar då längre efter medan den väntar. `hot_standby_feedback = on` stoppar de avbrott som vacuum orsakar, men dokumentationen varnar för att det kan blåsa upp primärnoden. `pg_stat_database_conflicts` på standbyn räknar avbrotten per orsak, och applikationen bör köra om en avbruten fråga.

En bortglömd slot fyller primärnodens disk. En replikeringsslot håller kvar WAL tills dess standby har tagit emot den, även när standbyn är avstängd eller sedan länge borta, och `max_slot_wal_keep_size` är -1 som standard, alltså ingen gräns. Kontrollera dem regelbundet:

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

Ta bort en slot vars standby är borta. Sätter man `max_slot_wal_keep_size` får varje slot ett tak för hur mycket den får hålla kvar; en standby som hamnar längre efter får sin slot ogiltigförklarad, och `wal_status` visar `lost` . `idle_replication_slot_timeout` , avstängd som standard, ogiltigförklarar slots som har varit inaktiva längre än den angivna tiden.

Skrivningarna stannar när en synkron standby gör det. Är bara en standby namngiven och den ligger nere väntar varje commit på primärnoden, kanske för alltid. Namnge fler kandidater än det antal som krävs, som i `ANY 1 (db2, db3)` . Går inte det, sänk antalet eller töm `synchronous_standby_names` och läs in konfigurationen på nytt.

Två servrar tror båda att de är primärnoden. PostgreSQL befordrar en standby när man ber om det, med `pg_ctl promote` eller `SELECT pg_promote()`, men har ingen egen programvara som upptäcker att primärnoden har fallit bort, och ingenting hindrar den gamla från att komma tillbaka och ta emot skrivningar. Dokumentationen varnar för att det leder till förvirring och till sist dataförlust. Stäng ute den gamla primärnoden, genom att stänga av den eller kapa dess nätverk, innan standbyn befordras, och bygg om den som standby innan den släpps in igen.

Logisk replikering lämnar saker efter sig. Schemaändringar replikeras inte, så en kolumn som läggs till hos publishern stoppar replikeringen tills subscribern också har den; gör ändringar som bara lägger till något hos subscribern först. Sekvenser replikeras inte heller: raderna kommer med sina id:n, men subscriberns sekvenser står kvar på sina startvärden, och efter ett byte misslyckas infogningar på dubblettnycklar. Sätt varje sekvens förbi det högsta värdet som används innan bytet.

En Redis-master startar om tom och tar replikerna med sig. Är persistens avstängd på mastern och något startar om den automatiskt kommer den tillbaka med en tom datamängd, och replikerna kopierar den och raderar sina egna data. Låt persistensen vara påslagen, eller låt inte mastern starta om automatiskt.

Läs mer

- High availability — PostgreSQL-dokumentationens översikt över primär- och standbyservrar, synkron och asynkron replikering och failover.
- Log-Shipping Standby Servers — strömmande replikering, replikeringsslots och hur synkron replikering ställs in.
- Hot Standby — skrivskyddade frågor på en standby, konflikterna som avbryter dem och inställningarna som väger fördröjning mot avbrott.
- pg_basebackup — att kopiera ett körande kluster för att starta en standby, med alla flaggor som används ovan.
- Restrictions — det som logisk replikering inte tar med sig: schema, sekvenser, large objects och allt annat än tabeller.
- Redis replication — hur Redis-repliker följer en master, varför replikeringen är asynkron och faran i att starta om utan persistens.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är replikering?” 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/replication/