Engineer

Replikering

å holde kopier av en database i takt på andre servere

Hva det er

Replikering holder kopier av en database på andre servere og sender hver kopi alle endringene som gjøres i originalen, slik at de holdes i takt. Mange bruker også det engelske ordet replication, og der sammenhengen gir rom for tvil, heter det databasereplikering. PostgreSQL-dokumentasjonen kaller den ene serveren som tar imot skrivinger, primary, og serverne som følger den, standby; på norsk sier man gjerne primærnoden og standbyserverne. Redis snakker om en master og replikaene dens.

PostgreSQL har to slag, og de kopierer forskjellige ting.

Fysisk replikering kopierer write-ahead-loggen (WAL), loggen hver endring skrives til før den når datafilene. En standby spiller av loggen og blir en blokk-for-blokk-kopi av hele klyngen. Ved strømmende replikering holder standbyen en forbindelse åpen mot primærnoden og tar imot WAL-postene etter hvert som de skrives, i stedet for å vente på at ferdige WAL-filer blir sendt over. En standby som bare spiller av og ikke tar imot tilkoblinger før den forfremmes, er en warm standby; en som også svarer på skrivebeskyttede spørringer, er en hot standby.

Logisk replikering kopierer endringer i rader. En publisher sender videre innsettingene, oppdateringene og slettingene som gjøres i utvalgte tabeller, hver rad kjent på sin replica identity, vanligvis primærnøkkelen, og en subscriber legger dem inn i sine egne tabeller. Fordi den arbeider med rader og ikke diskblokker, kan den nøye seg med noen få tabeller i stedet for hele klyngen, til og med over til en annen hovedversjon av PostgreSQL.

Strømmende replikering er asynkron som standard. Primærnoden bekrefter en commit så snart WAL ligger på dens egen disk, og sender den videre etterpå; dokumentasjonen anslår forsinkelsen til vanligvis under ett sekund når

standbyen klarer å holde følge. Med synkron replikering slått på venter hver commit til en standby også har WAL.

Bruk begrepet i en beslutning

Bruk «Hva er replikering?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Sikkerhetskopier som er testet i stedet for tatt for gitt, og en skriftlig plan for morgenen da en server ikke kommer opp igjen.

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. Automatiserte 100 kritiske sikkerhetskopier — 100 kritiske sikkerhetskopier automatisert på tvers av databasene

Automatiserte 100 kritiske sikkerhetskopier Ta kontakt →

Hvorfor det er viktig

Den første grunnen er å overleve at en server går tapt. Når primærnoden svikter, kan en standby som allerede har dataene, forfremmes og ta over trafikken, langt raskere enn en backup kan legges tilbake på en ny maskin.

Den andre er å spre lesingene. I mange applikasjoner leser de fleste spørringene bare, og en hot standby kan svare på dem, slik at primærnoden er ledig for skrivingene, som bare den kan ta imot.

Den tredje er å flytte data. Logisk replikering kan kopiere akkurat de tabellene en rapporteringsdatabase trenger, eller flytte data over til en server med en annen hovedversjon av PostgreSQL.

Hver kopi kommer med et valg mellom å vente og å miste. Asynkron replikering koster ingenting når det committes, men en standby ligger alltid litt etter: en lesing fra den kan gå glipp av en skriving som nettopp skjedde, og hvis primærnoden dør, kan commits den allerede har bekreftet, aldri nå fram til standbyen som tar over. Synkron replikering tetter det hullet og legger en tur fram og tilbake over nettverket inn i hver commit.

Replikering er heller ikke en backup. En standby spiller trofast av en utilsiktet `DELETE` uten `WHERE`, så noen øyeblikk senere har alle kopiene feilen. Det trengs fortsatt en backup for å rette den opp.

Slik ser det ut i praksis

En nettbutikk kjører databasen sin på én primærnode på `10.0.0.1` og legger til en server til, `db2`, som strømmende replika. På `db2`, der PostgreSQL er installert og stoppet:

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

`pg_basebackup` kopierer hele den kjørende klyngen inn i katalogen `-D` peker på, som opprettes om den mangler, og ellers må være tom. Som standard strømmer den samtidig med seg WAL-en som skrives mens kopieringen pågår. `-C -S db2_slot` oppretter først et replikeringsspor (replication slot) på primærnoden, slik at den tar vare på hver WAL-post til den nye standbyen har mottatt den. `-R` oppretter filen `standby.signal` og legger tilkoblingsinnstillingene, sporet inkludert, til i `postgresql.auto.conf`. `-P` viser framdriften. Rollen `replicator` må ha attributtet `REPLICATION` eller være superbruker, og `pg_hba.conf` på primærnoden må tillate replikeringstilkoblingen.

Når PostgreSQL starter på `db2`, får `standby.signal` den til å kjøre som standby og strømme fra primærnoden, og siden `hot_standby` er slått på som standard, tar den imot skrivebeskyttede spørringer så snart den er konsistent. På primærnoden viser én rad per standby hvordan det går:

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

En frisk replika viser `streaming` i `state` og `async` i `sync_state`. `replay_lag` er et anslag over hvor lang tid nye commits brukte på å bli synlige der; når standbyen har tatt igjen alt og WAL-aktiviteten stopper, står den på NULL i stedet for null.

Notatet om å optimalisere databasen før du sharder den forklarer hvorfor en butikk gjør dette. Mesteparten av trafikken er lesing: produktsiden, handlekurven, ordrehistorikken. Ingenting av det trenger primærnoden, som må holdes fri for skrivinger. En strømmende replika tar de lesingene, og notatet påpeker at det å legge til en er konfigurasjon, der en shard er et prosjekt. Det

nevner to forbehold. En replika ligger litt etter, så det en bruker nettopp har skrevet, bør leses fra primærnoden. Og en lang rapport på en replika kan bli avbrutt når den holder igjen endringer replikaen skal spille av.

Det trengs én innstilling på primærnoden for å få en commit til å vente på en standby. Hver standby får et navn gjennom `application_name` i sin `primary_conninfo`, og primærnoden lister opp navnene den kan vente på:

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

Nå venter hver commit til WAL-en ligger på disk både på primærnoden og på minst én av `db2` og `db3`. `synchronous_commit` kan settes per bruker, per database eller per transaksjon, slik at ordren som tar betalt, kan vente mens en teller for sidevisninger committer i lokalt tempo. Notatet om å selge den siste billetten nøyaktig én gang viser hvorfor det betyr noe for lagerbeholdningen. Med asynkron replikering kan en primærnode bekrefte et salg og dø før replikaen har fått det; replikaen forfremmes, salget er borte, og den samme enheten selges på nytt. Løsningen i notatet er en synkron standby eller et datalager bygd på Raft, og prisen er ekstra latens på hver commit.

Logisk replikering settes opp med to setninger. Publisheren må ha `wal_level = logical`, og tabellene må allerede finnes hos subscriberen, fordi definisjonene deres ikke blir kopiert:

```
-- 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 radene som allerede ligger i tabellene, og legger deretter inn alle senere endringer.

Hvor det går galt

Brukeren ser ikke det som nettopp ble lagret. Applikasjonen skriver til primærnoden og leser resultatet tilbake fra en replika som fortsatt ligger et øyeblikk etter, så en lagret innstilling ser ut til å hoppe tilbake. Send en brukers lesinger til primærnoden en kort stund etter hver skriving. Der data må være synlige på standbyen i samme øyeblikk som commit returnerer, venter `synchronous_commit = remote_apply` til de synkrone standbyene har spilt av transaksjonen, til en høyere pris per commit.

Lange spørringer på replikaen blir avbrutt. Standbyen må spille av endringer som kolliderer med en spørring som kjører: vacuum som fjerner radversjoner spørringen fortsatt kan se, eller den eksklusive låsen som `DROP TABLE` og mange former for `ALTER TABLE` tar. Avspillingen venter opptil `max_standby_streaming_delay`, 30 sekunder som standard, og avbryter så spørringen. En replika for rapporter kan få en lengre frist og blir da liggende lenger etter mens den venter. `hot_standby_feedback = on` stopper avbruddene som vacuum gir, men dokumentasjonen advarer om at det kan blåse opp primærnoden. `pg_stat_database_conflicts` på standbyen teller avbruddene etter årsak, og applikasjonen bør prøve en avbrutt spørring på nytt.

Et glemt replikeringsspor fyller disken på primærnoden. Et spor holder på WAL til standbyen har mottatt den, også når standbyen er slått av eller for lengst borte, og `max_slot_wal_keep_size` er -1 som standard, altså ingen grense. Sjekk sporene jevnlig:

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

Slett et spor når standbyen er borte. Setter man `max_slot_wal_keep_size`, får hvert spor et tak på hva det kan holde på; en standby som kommer lenger bak, får sporet sitt ugyldiggjort, og `wal_status` viser `lost`. `idle_replication_slot_timeout`, som er slått av som standard, ugyldiggjør spor som har vært inaktive lenger enn tiden som er angitt.

Skrivingene stopper når en synkron standby gjør det. Er bare én standby navngitt, og den er nede, venter hver commit på primærnoden, kanskje for alltid. Navngi flere kandidater enn antallet som kreves, som i `ANY 1 (db2, db3)`. Går ikke

det, senk antallet eller tøm `synchronous_standby_names`, og last inn konfigurasjonen på nytt.

To servere tror begge at de er primærnoden. PostgreSQL forfremmer en standby når man ber om det, med `pg_ctl promote` eller `SELECT pg_promote()`, men har ingen egen programvare som oppdager at primærnoden har falt ut, og ingenting hindrer den gamle i å komme tilbake og ta imot skrivinger. Dokumentasjonen advarer om at det fører til forvirring og til slutt datatap. Steng den gamle primærnoden ute, ved å slå den av eller kutte nettverket dens, før standbyen forfremmes, og bygg den opp igjen som standby før den slippes inn igjen.

Logisk replikering etterlater noe. Skjemaendringer replikeres ikke, så en kolonne som legges til hos publisheren, stopper replikeringen til subscriberen også har den; gjør endringer som bare legger til noe, hos subscriberen først. Sekvenser replikeres heller ikke: radene kommer med ID-ene sine, men sekvensene hos subscriberen står fortsatt på startverdien, og etter et bytte feiler innsettinger på dupliserte nøkler. Sett hver sekvens forbi den høyeste verdien i bruk før byttet.

En Redis-master starter tom og tar replikaene med seg. Er persistens slått av på masteren, og noe starter den automatisk på nytt, kommer den tilbake med et tomt datasett, og replikaene kopierer det og sletter sine egne data. Ha persistens slått på, eller ikke la masteren starte på nytt automatisk.

Les mer

- High availability — PostgreSQL-dokumentasjonens oversikt over primær- og standbyservere, synkron og asynkron replikering og failover.
- Log-Shipping Standby Servers — strømmende replikering, replikeringsspor og oppsett av synkron replikering.
- Hot Standby — skrivebeskyttede spørringer på en standby, konfliktene som avbryter dem, og innstillingene som veier forsinkelse mot avbrudd.
- pg_basebackup — kopiering av en kjørende klynge for å starte en standby, med alle valgene som er brukt over.
- Restrictions — det logisk replikering ikke tar med: skjema, sekvenser, large objects og alt annet enn tabeller.
- Redis replication — hvordan Redis-replikaer følger en master, hvorfor replikeringen er asynkron, og faren ved å starte på nytt uten persistens.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er replikering?» 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/replication/