Engineer

Реплікація

підтримання актуальних копій бази даних на інших серверах

Що це таке

Реплікація тримає копії бази даних на інших серверах і надсилає кожній копії всі зміни, внесені в оригінал, щоб копії не розходилися з ним. Там, де з контексту не ясно, про що мова, кажуть «реплікація бази даних». Документація PostgreSQL називає єдиний сервер, що приймає записи, основним (primary), а сервери, які йдуть за ним, резервними (standby); далі тут вони зватимуться репліками. У Redis це головний вузол (master) і його репліки.

У PostgreSQL є два види реплікації, і копіюють вони різне.

Фізична реплікація копіює журнал попереднього запису (WAL), куди кожна зміна потрапляє раніше, ніж до файлів даних. Репліка відтворює цей журнал і стає поблоковою копією всього кластера. За потокової реплікації вона тримає відкрите з'єднання з основним сервером і отримує записи WAL, щойно ті з'являються, а не чекає, поки їй перешлють готові файли WAL. Репліка, що лише відтворює журнал і не приймає з'єднань, доки її не підвищать до основної, — це warm standby; та, що водночас відповідає на запити лише для читання, — hot standby.

Логічна реплікація копіює зміни рядків. Публікатор передає вставки, оновлення й видалення у вибраних таблицях, упізнаючи кожен рядок за його replica identity, зазвичай за первинним ключем, а підписник застосовує їх до власних таблиць. Оскільки вона працює з рядками, а не з блоками диска, можна передавати кілька таблиць замість усього кластера, навіть на сервер з іншою мажорною версією PostgreSQL.

Потокова реплікація за замовчуванням асинхронна. Основний сервер підтверджує коміт, щойно WAL опинився на його власному диску, а пересилає його вже потім; документація оцінює затримку як зазвичай меншу за секунду, якщо репліка встигає. Коли ввімкнено синхронну реплікацію, коміт чекає, доки WAL отримає ще й репліка.

Застосуйте поняття до рішення

Застосуйте «Що таке реплікація?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Резервні копії, які перевірено, а не просто вважають робочими, і записаний план на той ранок, коли сервер не ввімкнеться.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Автоматизувала 100 критичних резервних копій — 100 критичних резервних копій автоматизовано в усіх базах

Автоматизувала 100 критичних резервних копій Зв'язатися з нами →

Чому це важливо

Перша причина — пережити втрату сервера. Коли основний сервер виходить з ладу, репліку, на якій дані вже є, можна підвищити до основної, і вона прийме трафік набагато швидше, ніж вдалося б відновити резервну копію на новій машині.

Друга — розподілити читання. У багатьох застосунках більшість запитів лише читає, і на них може відповідати hot standby, а основний сервер лишається вільним для записів, які приймає тільки він.

Третя — переносити дані. Логічна реплікація може копіювати саме ті таблиці, які потрібні базі для звітів, або перенести дані на сервер з іншою мажорною версією PostgreSQL.

Кожна копія ставить перед вибором: чекати чи втрачати. Асинхронна реплікація нічого не коштує під час коміту, але репліка завжди трохи відстає: читання з неї може не побачити щойно зроблений запис, а якщо основний сервер упаде, коміти, які він уже підтвердив, можуть так і не дійти до репліки, що його заступить. Синхронна реплікація закриває цю прогалину, але додає до кожного коміту обмін повідомленнями через мережу туди й назад.

Реплікація також не замінює резервної копії. Репліка сумлінно відтворить випадковий `DELETE` без `WHERE`, тож уже за мить помилка буде в усіх копіях. Щоб її відкотити, все одно потрібна резервна копія.

Як це виглядає на практиці

Інтернет-магазин тримає базу на одному основному сервері `10.0.0.1` і додає другий сервер, `db2`, як потокову репліку. На `db2`, де PostgreSQL встановлено, а сервер зупинено:

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

`pg_basebackup` копіює весь кластер, не зупиняючи його, у каталог, указаний у `-D`, який буде створено, якщо його немає, а наявний має бути порожнім, і за замовчуванням паралельно передає потоком WAL, що утворюється під час копіювання. `-C -S db2_slot` спершу створює на основному сервері слот реплікації, щоб той зберігав кожен запис WAL, доки нова репліка його не отримає. `-R` створює файл `standby.signal` і дописує параметри з'єднання, разом зі слотом, у `postgresql.auto.conf`. `-P` показує хід копіювання. Роль `replicator` повинна мати атрибут `REPLICATION` або права суперкористувача, а `pg_hba.conf` на основному сервері має дозволяти з'єднання для реплікації.

Коли PostgreSQL запускається на `db2`, файл `standby.signal` змушує його стартувати як репліку й отримувати потік з основного сервера, а оскільки `hot_standby` за замовчуванням увімкнено, репліка приймає запити лише для читання, щойно досягне узгодженого стану. На основному сервері представлення `pg_stat_replication` показує по рядку на кожну репліку:

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

Справна репліка показує `streaming` у `state` і `async` у `sync_state`. `replay_lag` приблизно показує, скільки часу знадобилося нещодавнім комітам, щоб стати там видимими; коли репліка наздогнала основний сервер і активність WAL припинилася, значення повертається до NULL, а не показує нуль.

Нотатка про те, щоб оптимізувати базу, перш ніж її шардувати, пояснює, навіщо магазин це робить. Більшість трафіку — читання: сторінка товару, кошик, історія замовлень. Жодному з цих читань не потрібен основний сервер, який має лишатися вільним для записів. Ці читання бере на себе потокова репліка, і нотатка наголошує, що додати її — справа конфігурації, тоді як шард — цілий проєкт. Вона називає два застереження. Репліка трохи відстає, тож те, що користувач щойно записав, слід читати з основного сервера. А довгий звіт на репліці можуть скасувати, коли він затримує зміни, які репліка має відтворити.

Щоб коміт чекав на репліку, на основному сервері досить одного параметра. Кожна репліка отримує ім’я через `application_name` у своєму `primary_conninfo`, а основний сервер перелічує імена, на які може чекати:

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

Тепер кожен коміт чекає, доки його WAL опиниться на диску і на основному сервері, і щонайменше на одній із реплік `db2` та `db3`. `synchronous_commit` можна задати для окремого користувача, бази даних чи транзакції, тож замовлення, яке приймає оплату, може чекати, а лічильник переглядів сторінок комітиться з локальною швидкістю. Нотатка про те, як продати останній квиток рівно один раз, показує, чому це важливо для залишків. За асинхронної реплікації основний сервер може підтвердити продаж і впасти, перш ніж репліка його отримає; репліку підвищують, продажу немає, і ту саму одиницю продають знову. Нотатка радить синхронну репліку або сховище на Raft, і ціна цього — додаткова затримка на кожному коміті.

Логічна реплікація налаштовується двома командами. Публікатору потрібен `wal_level = logical`, а таблиці вже мають існувати в підписника, бо їхні визначення не копіюються:

```
-- на публікаторі
CREATE PUBLICATION shop_pub FOR TABLE orders, customers;

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

Підписка спершу копіює рядки, які вже є в таблицях, а потім застосовує всі пізніші зміни.

Де виникають проблеми

Користувач не бачить того, що щойно зберіг. Застосунок пише на основний сервер, а результат читає з репліки, яка ще на мить відстає, тож збережене налаштування ніби повертається назад. Спрямовуйте читання користувача на основний сервер ще якийсь час після кожного його запису. Коли дані мають бути видимі на репліці тієї ж миті, як коміт повернув відповідь, `synchronous_commit = remote_apply` змушує чекати, доки синхронні репліки відтворять транзакцію, і кожен коміт коштує дорожче.

Довгі запити на репліці скасовуються. Репліка мусить відтворювати зміни, які конфліктують із запитом, що виконується: vacuum, що прибирає версії рядків, які запит ще бачить, або ексклюзивне блокування, яке беруть `DROP TABLE` і багато форм `ALTER TABLE`. Відтворення чекає до `max_standby_streaming_delay`, за замовчуванням 30 секунд, а тоді скасовує запит. Репліці для звітів можна дати довшу затримку, але, поки вона чекає, відставатиме ще більше. `hot_standby_feedback = on` прибирає скасування через vacuum, але документація попереджає, що основний сервер від цього може роздуватися. `pg_stat_database_conflicts` на репліці рахує скасування за причинами, а застосунок має бути готовим повторити скасований запит.

Забутий слот заповнює диск основного сервера. Слот тримає WAL, доки його репліка цей WAL не отримає, навіть якщо репліку вимкнули чи давно прибрали, а `max_slot_wal_keep_size` за замовчуванням дорівнює -1, тобто без обмеження. Регулярно перевіряйте слоти:

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

Видаляйте слот, чия репліка зникла. Параметр `max_slot_wal_keep_size` обмежує, скільки слот може тримати; репліка, яка відстане більше, виявить свій слот анульованим, а `wal_status` показуватиме `lost`. `idle_replication_slot_timeout`, за замовчуванням вимкнений, анулює слоти, що лишаються неактивними довше за заданий час.

Записи зупиняються, коли зупиняється синхронна репліка. Якщо названо лише одну репліку і вона впала, кожен коміт на основному сервері чекає, можливо, без кінця. Називайте більше кандидатів, ніж потрібно, як у `ANY 1 (db2, db3)`. Якщо це неможливо, зменште число або очистьте `synchronous_standby_names` і перезавантажте конфігурацію.

Два сервери водночас вважають себе основними. PostgreSQL підвищує репліку на вимогу, через `pg_ctl promote` або `SELECT pg_promote()`, але не має власних засобів, щоб помітити відмову основного сервера, і ніщо не заважає старому повернутися й знову приймати записи. Документація попереджає, що це призводить до плутанини і зрештою до втрати даних. Ізолюйте старий основний сервер, вимкнувши його або перерізавши йому мережу, перш ніж підвищувати репліку, і перебудуйте його як репліку, перш ніж пускати назад.

Логічна реплікація дещо лишає позаду. Зміни схеми не реплікуються, тож стовпець, доданий на публікаторі, зупиняє реплікацію, доки його не матиме й підписник; зміни, які лише щось додають, застосовуйте спершу на підписнику. Послідовності теж не реплікуються: рядки приходять зі своїми ідентифікаторами, але послідовності підписника стоять на початкових значеннях, тож після перемикання вставки падають на дублікатах ключа. Перед перемиканням виставте кожну послідовність за найбільше значення, що вже використовується.

Головний вузол Redis перезапускається порожнім і тягне за собою репліки. Якщо на головному вузлі вимкнено збереження на диск (persistence), а щось автоматично його перезапускає, він повертається з порожнім набором даних, і репліки копіюють цей стан, стираючи власні дані. Тримайте збереження ввімкненим або не дозволяйте головному вузлу перезапускатися автоматично.

Що почитати далі

- High availability — огляд у документації PostgreSQL: основні й резервні сервери, синхронна та асинхронна реплікація, перемикання на резерв.
- Log-Shipping Standby Servers — потокова реплікація, слоти реплікації та налаштування синхронної реплікації.
- Hot Standby — запити лише для читання на репліці, конфлікти, що їх скасовують, і параметри, які зважують відставання проти скасувань.
- pg_basebackup — копіювання кластера, що працює, щоб запустити репліку, з усіма вжитими вище параметрами.

- Restrictions — чого логічна реплікація не переносить: схеми, послідовностей, великих об’єктів і всього, крім таблиць.
- Redis replication — як репліки Redis стежать за головним вузлом, чому реплікація асинхронна і чим небезпечний перезапуск без збереження на диск.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке реплікація?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/replication/