Engineer

Replikation

Datenbankkopien auf anderen Servern aktuell halten

Was es ist

Replikation hält Kopien einer Datenbank auf anderen Servern und übermittelt jeder Kopie jede Änderung am Original, sodass alle auf demselben Stand bleiben. Lässt der Kontext Zweifel zu, heißt sie Datenbankreplikation. Die PostgreSQL-Dokumentation nennt den einen Server, der Schreibzugriffe annimmt, primary und die ihm folgenden Server standbys. Redis spricht von einem Master und seinen Replikaten.

PostgreSQL kennt zwei Arten, die unterschiedliche Dinge kopieren.

Physische Replikation kopiert das Write-Ahead Log (WAL), in dem jede Änderung vor ihrer Übernahme in die Datendateien festgehalten wird. Ein Standby spielt dieses Log nach und wird zu einer blockgleichen Kopie des gesamten Clusters. Bei Streaming-Replikation hält er eine Verbindung zum Primärserver offen und empfängt WAL-Einträge bei ihrer Entstehung, statt auf die Übermittlung abgeschlossener WAL-Dateien zu warten. Ein Standby, der nur nachspielt und bis zur Beförderung keine Verbindungen annimmt, heißt Warm Standby. Einer, der außerdem nur lesende Abfragen beantwortet, heißt Hot Standby.

Logische Replikation kopiert Zeilenänderungen. Ein Publisher übermittelt die Einfügungen, Aktualisierungen und Löschungen ausgewählter Tabellen. Jede Zeile wird durch ihre Replikatidentität erkannt, gewöhnlich den Primärschlüssel. Ein Subscriber wendet die Änderungen auf seine eigenen Tabellen an. Weil es um Zeilen statt Datenträgerblöcke geht, können wenige Tabellen statt des ganzen Clusters übertragen werden, auch an eine andere PostgreSQL-Hauptversion.

Streaming-Replikation ist standardmäßig asynchron. Der Primärserver bestätigt einen Commit, sobald das WAL auf seinem eigenen Datenträger liegt, und sendet es danach weiter. Die Dokumentation nennt eine typische Verzögerung

von unter einer Sekunde, wenn der Standby mithalten kann. Bei eingeschalteter synchroner Replikation wartet der Commit, bis auch ein Standby das WAL hat.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Replikation?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Sicherungen, deren Funktion geprüft und nicht vorausgesetzt wird, und ein schriftlicher Plan für den Morgen, an dem ein Server nicht wieder startet.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 100 wichtige Sicherungen mit Barman automatisiert — 100 automatisierte geschäftskritische Datenbanksicherungen

100 wichtige Sicherungen mit Barman automatisiert Kontakt aufnehmen →

Warum es wichtig ist

Der erste Grund ist, den Verlust eines Servers zu überstehen. Fällt der Primärserver aus, kann ein Standby mit den schon vorhandenen Daten befördert werden und den Verkehr übernehmen. Das geht deutlich schneller, als sich eine Sicherung auf einem neuen Rechner wiederherstellen lässt.

Der zweite Grund ist die Verteilung von Lesezugriffen. In vielen Anwendungen lesen die meisten Abfragen nur. Ein Hot Standby kann sie beantworten und den Primärserver für Schreibzugriffe freihalten, die nur dieser annehmen kann.

Der dritte Grund ist der Datentransport. Logische Replikation kann nur die für eine Berichtsdatenbank nötigen Tabellen kopieren oder Daten auf einen Server mit anderer PostgreSQL-Hauptversion übertragen.

Jede Kopie bringt die Wahl zwischen Warten und Verlieren mit. Asynchrone Replikation kostet auf dem Commitpfad nichts, ein Standby liegt jedoch immer etwas zurück: Eine gerade erfolgte Schreiboperation kann beim Lesen fehlen. Stirbt der Primärserver, erreichen schon bestätigte Commits möglicherweise niemals den übernehmenden Standby. Synchrone Replikation schließt diese Lücke und fügt jedem Commit einen Netzwerk-Hin- und Rückweg hinzu.

Replikation ist auch kein Backup. Ein Standby spielt ein versehentliches `DELETE` ohne `WHERE` zuverlässig nach. Binnen Augenblicken hat damit jede Kopie denselben Fehler. Ihn rückgängig zu machen erfordert weiterhin ein Backup.

Wie es in der Praxis aussieht

Ein Shop betreibt seine Datenbank auf einem Primärserver unter `10.0.0.1` und ergänzt einen zweiten Server, `db2`, als Streaming-Replikat. Auf `db2`, mit installiertem und angehaltenem PostgreSQL:

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

`pg_basebackup` kopiert den gesamten laufenden Cluster in das mit `-D` bezeichnete Verzeichnis. Es wird bei Bedarf angelegt und muss andernfalls leer sein. Standardmäßig wird gleichzeitig das während der Kopie erzeugte WAL übertragen. `-C -S db2_slot` legt zuerst einen Replikationsslot auf dem Primärserver an, damit dieser jeden WAL-Eintrag bis zum Empfang durch den neuen Standby behält. `-R` erzeugt eine Datei `standby.signal` und hängt die Verbindungseinstellungen einschließlich Slot an `postgresql.auto.conf` an. `-P` meldet den Fortschritt. Die Rolle `replicator` braucht das Attribut `REPLICATION` oder Superuserrechte. Die `pg_hba.conf` des Primärservers muss die Replikationsverbindung zulassen.

Startet PostgreSQL auf `db2`, lässt `standby.signal` es als Standby starten und vom Primärserver streamen. Weil `hot_standby` standardmäßig eingeschaltet ist, nimmt es nach Erreichen eines konsistenten Zustands nur lesende Abfragen an. Auf dem Primärserver zeigt eine Zeile pro Standby dessen Zustand:

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

Ein gesundes Replikat zeigt `state` als `streaming` und `sync_state` als `async`. `replay_lag` liefert einen Näherungswert dafür, wie lange die jüngsten Commits brauchten, um dort sichtbar zu werden. Hat der Standby aufgeholt und die WAL-Aktivität aufgehört, kehrt der Wert zu NULL zurück, statt null anzuzeigen.

Die Notiz zum Optimieren der Datenbank vor dem Sharding erklärt, warum ein Shop das tut. Die meisten Anfragen sind Lesezugriffe: Produktseite, Warenkorb, Bestellverlauf. Nichts davon benötigt den für Schreibzugriffe freizuhaltenden Primärserver. Ein Streaming-Replikat übernimmt diese Lesezugriffe. Ein solches hinzuzufügen ist laut Notiz Konfiguration, während ein Shard ein Projekt ist. Sie

nennt zwei Vorbehalte. Ein Replikat liegt etwas zurück; gerade geschriebene Daten eines Nutzers sollten deshalb vom Primärserver gelesen werden. Ein langer Bericht auf einem Replikat kann zudem abgebrochen werden, wenn er die nachgespielten Änderungen aufhält.

Damit ein Commit auf einen Standby wartet, genügt eine Einstellung auf dem Primärserver. Jeder Standby erhält seinen Namen durch `application_name` in seiner `primary_conninfo`. Der Primärserver listet die Namen auf, auf die er warten darf:

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

Jeder Commit wartet jetzt, bis sein WAL auf dem Primärserver und mindestens einem von `db2` und `db3` auf Datenträger liegt. `synchronous_commit` kann pro Benutzer, Datenbank oder Transaktion gesetzt werden. Die Zahlungsannahme für eine Bestellung kann daher warten, während ein Seitenaufrufzähler mit lokaler Geschwindigkeit committet. Die Notiz über den genau einmaligen Verkauf des letzten Tickets zeigt die Bedeutung für Bestände. Bei asynchroner Replikation kann der Primärserver einen Verkauf bestätigen und ausfallen, bevor sein Replikat ihn hat. Das Replikat wird befördert, der Verkauf fehlt und dieselbe Einheit wird erneut verkauft. Die Notiz nennt als Abhilfe einen synchronen Standby oder einen auf Raft aufgebauten Speicher. Der Preis ist zusätzliche Latenz bei jedem Commit.

Logische Replikation benötigt zwei Anweisungen. Der Publisher braucht `wal_level = logical`. Die Tabellen müssen auf dem Subscriber schon existieren, weil ihre Definitionen nicht kopiert werden:

```
-- on the publisher
CREATE PUBLICATION shop_pub FOR TABLE orders, customers;

-- on the subscriber
CREATE SUBSCRIPTION shop_sub
    CONNECTION 'host=10.0.0.1 dbname=shop user=replicator'
    PUBLICATION shop_pub;
```

Die Subskription kopiert zuerst die bereits vorhandenen Tabellenzeilen und wendet danach jede weitere Änderung an.

Wo es schiefgeht

Ein Nutzer sieht nicht, was er gerade geschrieben hat. Die Anwendung schreibt auf den Primärserver und liest anschließend von einem noch etwas zurückliegenden Replikat. Eine gespeicherte Einstellung scheint dadurch zurückzuspringen. Führen Sie die Lesezugriffe eines Nutzers nach einer Schreiboperation für kurze Zeit zum Primärserver. Müssen Daten auf dem Standby bereits beim Zurückkehren des Commitaufrufs sichtbar sein, wartet `synchronous_commit = remote_apply` auf das Nachspielen der Transaktion auf den synchronen Standbys. Das kostet mehr pro Commit.

Lange Abfragen auf dem Replikat werden abgebrochen. Der Standby muss Änderungen nachspielen, die mit einer laufenden Abfrage kollidieren: durch vacuum entfernte, für die Abfrage noch sichtbare Zeilenversionen oder die Exklusivsperre von `DROP TABLE` und vielen Formen von `ALTER TABLE`. Das Nachspielen wartet bis zu `max_standby_streaming_delay`, standardmäßig 30 Sekunden, und bricht dann die Abfrage ab. Ein Replikat für Berichte kann länger warten, gerät währenddessen aber weiter in Rückstand. `hot_standby_feedback = on` verhindert die durch vacuum verursachten Abbrüche. Laut Dokumentation kann dies jedoch den Datenbestand des Primärservers aufblähen. `pg_stat_database_conflicts` auf dem Standby zählt die Abbrüche nach Ursache. Die Anwendung sollte eine abgebrochene Abfrage wiederholen.

Ein vergessener Slot füllt den Datenträger des Primärservers. Ein Slot behält WAL, bis sein Standby es empfangen hat, auch wenn dieser ausgeschaltet oder längst entfernt ist. `max_slot_wal_keep_size` steht standardmäßig auf -1, also ohne Grenze. Prüfen Sie die Slots regelmäßig:

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

Löschen Sie einen Slot, dessen Standby verschwunden ist. `max_slot_wal_keep_size` begrenzt, wie viel ein Slot behalten darf. Ein Standby, dessen Rückstand diese Grenze überschreitet, findet seinen Slot ungültig vor; `wal_status` zeigt `lost`. Das standardmäßig ausgeschaltete `idle_replication_slot_timeout` erklärt Slots für ungültig, die länger als die angegebene Zeit inaktiv waren.

Schreibzugriffe halten an, wenn ein synchroner Standby anhält. Ist nur ein Standby benannt und dieser ausgefallen, wartet jeder Commit auf dem Primärserver und wird womöglich nie fertig. Benennen Sie mehr Kandidaten als die benötigte Zahl, wie in `ANY 1 (db2, db3)`. Geht das nicht, senken Sie die Zahl oder leeren Sie `synchronous_standby_names` und laden Sie die Konfiguration neu.

Zwei Server halten sich beide für den Primärserver. PostgreSQL befördert einen Standby auf Wunsch mit `pg_ctl promote` oder `SELECT pg_promote()`. Es hat aber keine eigene Software, die einen ausgefallenen Primärserver erkennt. Nichts hindert den alten daran, zurückzukehren und Schreibzugriffe anzunehmen. Die Dokumentation warnt vor „Verwirrung und schließlich Datenverlust“. Isolieren Sie den alten Primärserver vor der Beförderung, indem Sie ihn ausschalten oder seine Netzverbindung trennen. Bauen Sie ihn vor seiner Rückkehr als Standby neu auf.

Logische Replikation lässt Dinge zurück. Schemaänderungen werden nicht repliziert. Eine auf dem Publisher ergänzte Spalte stoppt deshalb die Replikation, bis der Subscriber sie ebenfalls besitzt. Nehmen Sie additive Änderungen zuerst am Subscriber vor. Sequenzen werden ebenfalls nicht repliziert: Die Zeilen kommen mit ihren Kennungen an, die Sequenzen des Subscribers stehen aber noch auf den Anfangswerten. Einfügungen nach einem Umschalten scheitern dann an doppelten Schlüsseln. Setzen Sie vor dem Wechsel jede Sequenz hinter den höchsten verwendeten Wert.

Ein Redis-Master startet leer und zieht seine Replikate mit. Ist die Persistenz auf dem Master ausgeschaltet und startet ein Mechanismus ihn automatisch neu, kehrt er mit leeren Daten zurück. Die Replikate kopieren dies und löschen ihre eigenen Daten. Halten Sie die Persistenz eingeschaltet oder verhindern Sie den automatischen Neustart des Masters.

Weiterführende Lektüre

- Hochverfügbarkeit — PostgreSQL-Überblick zu Primärservern und Standbys, synchroner und asynchroner Replikation sowie Failover.
- Standby-Server mit Logversand — Streaming-Replikation, Replikationsslots und die Konfiguration synchroner Replikation.
- Hot Standby — nur lesende Abfragen auf einem Standby, die sie abbrechenden Konflikte und Einstellungen zur Abwägung zwischen Verzögerung und Abbrüchen.

- pg_basebackup — einen laufenden Cluster zum Start eines Standbys kopieren, mit jeder oben verwendeten Option.
- Einschränkungen — was logische Replikation nicht überträgt: Schema, Sequenzen, große Objekte und alles außer Tabellen.
- Redis-Replikation — wie Redis-Replikate einem Master folgen, warum Replikation asynchron ist und die Gefahr eines Neustarts ohne Persistenz.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Replikation?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/replication/