Engineer

レプリケーション

別のサーバーにデータベースの複製を置き、変更を反映する仕組み

どのようなものか

レプリケーションは、データベースの複製を別のサーバーに置き、元のデータベースの各変更を送って、状態をそろえる仕組みです。文脈が曖昧なら、データベースのレプリケーションと呼びます。PostgreSQLのドキュメントは、書き込みを受け付ける一台をプライマリー、それに追従するサーバーをスタンバイと呼びます。Redisは、マスターとレプリカという言葉を使います。

PostgreSQLには二種類あり、複製するものが違います。

物理レプリケーションは、データファイルへ届く前に、すべての変更を記録する、先行書き込みログ（WAL）を複製します。スタンバイは、ログを再生し、クラスター全体の、ブロック単位で同じ複製になります。ストリーミングレプリケーションでは、完成したWALファイルを転送するまで待たず、プライマリーとの接続を開いたまま、生成されたWALレコードを受け取ります。再生だけを行い、昇格するまで接続を受け付けないものを、ウォームスタンバイ、読み取り専用のクエリーにも答えるものを、ホットスタンバイと呼びます。

論理レプリケーションは、行への変更を複製します。パブリッシャーが、選んだテーブルへの挿入、更新、削除を送り、サブスクライバーが、自分のテーブルへ適用します。各行は、通常は主キーを使うレプリカアイデンティティーで識別します。ディスクのブロックではなく行を扱うため、クラスター全体ではなく、一部のテーブルだけを運べ、違うメジャーバージョンのPostgreSQLへも送れます。

ストリーミングレプリケーションは、既定で非同期です。プライマリーは、WALを自分のディスクへ置くと、コミットに成功応答を返し、その後で転送します。ドキュメントによると、スタンバイが追い付ける場合の遅れは、通常一秒未満です。同期レプリケーションを有効にすると、スタンバイにもWALが保存されるまで、コミットが待ちます。

概念を判断に生かす

「レプリケーションとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 あると信じるだけでなく、復元を試したバックアップと、サーバーが起動しない朝に使える手順書を用意します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Barmanで重要データのバックアップ100件を自動化 — 100 データベース全体で自動化した重要バックアップ

Barmanで重要データのバックアップ100件を自動化 お問い合わせ →

なぜ重要なのか

一つ目の理由は、サーバーを失っても継続することです。プライマリーが故障したら、既にデータを持つスタンバイを昇格させ、通信を引き継がせることができます。新しいマシンへバックアップを復元するより、はるかに早く行えます。

二つ目は、読み取りの分散です。多くのアプリケーションでは、クエリーの大半は読み取りだけなので、ホットスタンバイが応答し、プライマリーは、自分にしかできない書き込みに専念できます。

三つ目は、データの移動です。論理レプリケーションは、レポート用データベースが必要とするテーブルだけを複製したり、別のメジャーバージョンのPostgreSQLのサーバーへデータを運んだりできます。

どの複製でも、待ち時間を受け入れるか、データを失うリスクを受け入れるかを選びます。非同期レプリケーションは、コミットの経路に待ち時間を加えませんが、スタンバイは、常に少し遅れます。読み取りが、直前の書き込みを見落とす場合があり、プライマリーが止まると、成功応答済みのコミットが、引き継ぐスタンバイへ届かないままになる場合もあります。同期レプリケーションは、その隙間を閉じ、すべてのコミットへ、ネットワークの一往復を加えます。

レプリケーションは、バックアップではありません。スタンバイは、誤って `DELETE` から `WHERE` を省いた場合も忠実に再生し、すぐに、すべての複製が同じ誤りを持ちます。取り消すには、やはりバックアップが必要です。

実際の活用例

店が、一台のプライマリー `10.0.0.1` でデータベースを動かし、二台目の `db2` をストリーミングレプリカとして加える例です。 `db2` にPostgreSQLをインストールし、そのPostgreSQLを停止した状態で、次を実行します。

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

`pg_basebackup` は、稼働中のクラスター全体を、 `-D` で指定したディレクトリーへ複製します。なければ作成し、既存なら空でなければなりません。既定では、複製中に生成されるWALも、並行してストリーミングします。 `-C -S db2_slot` は、まずプライマリーにレプリケーションスロットを作り、新しいスタンバイが受け取るまで、各WALレコードを保持させます。 `-R` は、 `standby.signal` ファイルを作り、スロットを含む接続設定を `postgresql.auto.conf` へ追記します。 `-P` は、進捗を表示します。 `replicator` ロールには、 `REPLICATION` 属性か、スーパーユーザー権限が必要で、プライマリーの `pg_hba.conf` も、複製の接続を許可する必要があります。

`db2` でPostgreSQLを起動すると、 `standby.signal` によって、スタンバイとして立ち上がり、プライマリーからストリーミングで受信します。 `hot_standby` は既定で有効なため、整合性のある状態になると、読み取り専用のクエリーを受け付けます。プライマリーでは、スタンバイごとに一行で、状態を確認できます。

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

正常なレプリカは、 `state` が `streaming` 、 `sync_state` が `async` になります。 `replay_lag` は、最近のコミットが、そのレプリカで見えるまでの時間の概算です。スタンバイが追い付き、WALの活動が止まると、ゼロではなくNULLへ戻ります。

シャーディングの前にデータベースを最適化するノートは、店がこうする理由を説明します。通信の大半は、商品ページ、カート、注文履歴などの読み取りです。どれにも、書き込みの余力を保つ必要があるプライマリーは不要です。ストリーミングレプリカが読み取りを引き受けます。ノートは、レプリカの追加は設定だが、シャードは一つのプロジェクトだと指摘します。注意点も二つ挙げています。レプリカは少し遅れるため、利用者が今書いたものは、プライマリーから読みます。また、レプリカ上の長いレポートは、変更の再生を妨げると、キャンセルされる場合があります。

コミットをスタンバイ待ちにするには、プライマリーで一つ設定します。各スタンバイの名前は、 `application_name` で指定し、その値は `primary_conninfo` に含めます。プライマリー側に、待機対象にできるスタンバイの名前を列挙します。

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

これで、各コミットは、プライマリーと、`db2` か `db3` の少なくとも一方のディスクへWALが保存されるまで待ちます。`synchronous_commit` は、利用者、データベース、トランザクションごとに設定できるため、決済を伴う注文は待ち、ページビューのカウンターは、ローカルの速さでコミットできます。最後の一枚を一度だけ売るノートは、在庫にとっての重要性を説明します。非同期では、プライマリーが販売に成功応答を返した後、レプリカへ届く前に止まると、昇格したレプリカには、その販売がなく、同じ一個を再び売ってしまいます。ノートの対策は、同期スタンバイか、Raftに基づくストアで、すべてのコミットにレイテンシーが増える代償を伴います。

論理レプリケーションには、二つの文を使います。パブリッシャーには、`wal_level = logical` が必要で、定義は複製されないため、テーブルは、サブスクライバーに既に存在する必要があります。

```
-- 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;
```

サブスクリプションは、まず、これらのテーブルに既にある行を複製し、その後のすべての変更を適用します。

問題が起きるところ

利用者が、今書いたものを読めません。 アプリケーションがプライマリーへ書き、まだ少し遅れたレプリカから読むため、保存した設定が元へ戻ったように見えます。書き込み後、短い間は、その利用者の読み取りを、プライマリーへ送ります。コミットが返った瞬間に、スタンバイでも見える必要があれば、`synchronous_commit = remote_apply` が、同期スタンバイでトランザクションを再生するまで待ちますが、コミットごとのコストは上がります。

レプリカの長いクエリーが、キャンセルされます。 スタンバイは、実行中のクエリーと衝突する変更を再生する必要があります。たとえば、まだクエリーから見える行の版をvacuumが除く場合や、`DROP TABLE` や多くの形式の `ALTER TABLE`

が、排他的ロックを取る場合です。再生は、`max_standby_streaming_delay`で設定する時間（既定は30秒）を上限に待ち、それからクエリーをキャンセルします。レポート用のレプリカには、長い待ち時間を与えられますが、その間、さらに遅れます。`hot_standby_feedback = on`は、vacuumが原因のキャンセルを止めますが、ドキュメントは、プライマリーを肥大化させ得ると警告します。スタンバイの`pg_stat_database_conflicts`が、原因別にキャンセルを数え、アプリケーションは、キャンセルされたクエリーを再試行すべきです。

忘れられたスロットが、プライマリーのディスクを埋めます。 スロットは、スタンバイが停止中や、既に存在しなくても、受け取るまでWALを保持します。`max_slot_wal_keep_size`は、既定で-1、つまり無制限です。定期的に、スロットを確認します。

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

スタンバイがなくなったスロットは、削除します。`max_slot_wal_keep_size`を設定すると、保持できる量に上限が付き、それ以上に遅れたスタンバイのスロットは無効になり、`wal_status`は`lost`を示します。既定ではオフの`idle_replication_slot_timeout`は、指定時間より長く非稼働だったスロットを無効にします。

同期スタンバイが止まると、書き込みも止まります。 名前を指定したスタンバイが一台だけで、それが停止すると、プライマリーの全コミットが待ち、いつまでも完了しない場合があります。`ANY 1 (db2, db3)`のように、必要な台数より多くの候補を指定します。それが無理なら、台数を下げるか、`synchronous_standby_names`を空にし、設定を再読み込みします。

二台が、どちらもプライマリーだと考えます。 PostgreSQLは、`pg_ctl promote`や`SELECT pg_promote()`で指示されると、スタンバイを昇格させますが、プライマリーの故障を検知する独自のソフトウェアはなく、以前のプライマリーが戻って、書き込みを受け付けることも止めません。ドキュメントは、「混乱と、最終的にはデータ損失」を招くと警告します。昇格前に、以前のプライマリーを、電源を切るかネットワークを断って隔離し、戻す前に、スタンバイとして再構築します。

論理レプリケーションでは、複製しないものがあります。 スキーマ変更は複製しないため、パブリッシャーで列を追加すると、サブスクライバーにもできるまで、複製が止まります。スキーマに要素を追加する変更は、先にサブスクライバ

ーへ適用します。シーケンスも複製しません。行はID付きで届いても、サブスクライバーのシーケンスは、開始値のままで、切り替え後の挿入は、キーの重複で失敗します。切り替える前に、各シーケンスを、使用中の最大値より先へ進めます。

Redisのマスターが、空で再起動し、レプリカも巻き込みます。 マスターの永続化が無効で、何らかの仕組みがマスターを自動再起動すると、空のデータセットで戻り、レプリカも、それを複製して、自分のデータを削除します。永続化を有効に保つか、マスターを自動再起動させないようにします。

参考資料

- 高可用性 — PostgreSQLの、プライマリーとスタンバイ、同期と非同期の複製、フェイルオーバーの概要です。
- ログ転送によるスタンバイサーバー — ストリーミングレプリケーション、レプリケーションスロット、同期の設定方法です。
- ホットスタンバイ — スタンバイでの読み取り専用クエリー、キャンセルを起こす衝突、遅れとキャンセルのトレードオフを決める設定です。
- pg_basebackup — 稼働中のクラスターを複製してスタンバイを始める方法で、前述の全オプションを説明します。
- 制限事項 — 論理レプリケーションが運ばない、スキーマ、シーケンス、ラージオブジェクト、テーブル以外のすべてです。
- Redisのレプリケーション — マスターへ追従する方法、非同期である理由、永続化せずに再起動する危険です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「レプリケーションとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/replication/