Engineer

replicación

Copias de una base de datos al día en otros servidores

Qué es

La replicación mantiene copias de una base de datos en otros servidores y envía a cada una todos los cambios realizados en el original, para que se mantengan al día. Cuando el contexto deja dudas, se denomina replicación de bases de datos. La documentación de PostgreSQL llama primario al único servidor que acepta escrituras y servidores en espera a los que lo siguen. Redis habla de un maestro y sus réplicas.

PostgreSQL tiene dos tipos, que copian cosas distintas.

La replicación física copia el registro de escritura anticipada (WAL), donde se anota cada cambio antes de que llegue a los archivos de datos. Un servidor en espera reproduce ese registro y se convierte en una copia bloque a bloque de todo el clúster. En la replicación por transmisión, mantiene abierta una conexión con el primario y recibe los registros WAL conforme se generan, en vez de esperar a que se envíen archivos WAL completos. Un servidor en espera que solo reproduce el registro y no acepta conexiones hasta su promoción es un warm standby; uno que además responde a consultas de solo lectura es un hot standby.

La replicación lógica copia cambios en filas. Un publicador transmite las inserciones, actualizaciones y eliminaciones realizadas en las tablas elegidas; cada fila se identifica por su identidad de réplica, normalmente la clave primaria, y un suscriptor aplica los cambios a sus propias tablas. Como trabaja con filas en vez de bloques de disco, puede transportar unas pocas tablas en lugar de todo el clúster, incluso a una versión principal distinta de PostgreSQL.

La replicación por transmisión es asíncrona de forma predeterminada. El primario confirma un commit cuando el WAL está en su propio disco y lo envía después. La documentación sitúa el retraso normalmente por debajo de un segundo cuando

el servidor en espera puede seguir el ritmo. Al activar la replicación síncrona, el commit espera hasta que un servidor en espera también tenga el WAL.

Utilice el concepto en una decisión

Aplique «¿Qué es la replicación?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Copias de seguridad que se prueban en vez de darse por buenas y un plan escrito para la mañana en que un servidor no vuelva a encenderse.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 100 copias críticas automatizadas con Barman — 100 copias críticas automatizadas en las bases de datos

100 copias críticas automatizadas con Barman Póngase en contacto →

Por qué importa

El primer motivo es sobrevivir a la pérdida de un servidor. Si falla el primario, un servidor en espera que ya tiene los datos puede ser promovido y recibir el tráfico mucho antes de lo que llevaría restaurar una copia de seguridad en una máquina nueva.

El segundo es repartir las lecturas. En muchas aplicaciones, la mayoría de las consultas solo leen, y un hot standby puede responderlas, dejando libre al primario para las escrituras, que solo él puede aceptar.

El tercero es mover datos. La replicación lógica puede copiar únicamente las tablas que necesita una base de datos de informes o llevar datos a un servidor con otra versión principal de PostgreSQL.

Cada copia exige elegir entre esperar y perder. La replicación asíncrona no añade coste a la ruta del commit, pero un servidor en espera siempre va un poco por detrás: una lectura puede no mostrar una escritura recién realizada y, si muere el primario, los commits que ya había confirmado pueden no llegar nunca al servidor en espera que lo sustituye. La replicación síncrona cierra esa brecha y añade un viaje de ida y vuelta por la red a cada commit.

La replicación tampoco es una copia de seguridad. Un servidor en espera reproduce fielmente un `DELETE` accidental sin `WHERE`, de modo que en unos instantes todas las copias contienen también el error. Deshacerlo sigue exigiendo una copia de seguridad.

Cómo se aplica en la práctica

Una tienda ejecuta su base de datos en un primario en `10.0.0.1` y añade un segundo servidor, `db2`, como réplica por transmisión. En `db2`, con PostgreSQL instalado y detenido:

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

`pg_basebackup` copia todo el clúster en funcionamiento en el directorio indicado por `-D`, que crea si hace falta y que, de existir, debe estar vacío. De forma predeterminada, transmite a la vez el WAL generado durante la copia. `-C -S db2_slot` crea primero una ranura de replicación en el primario, para que este conserve cada registro WAL hasta que el nuevo servidor en espera lo haya recibido. `-R` crea un archivo `standby.signal` y añade los parámetros de conexión, incluida la ranura, a `postgresql.auto.conf`. `-P` muestra el progreso. El rol `replicator` necesita el atributo `REPLICATION` o derechos de superusuario, y el `pg_hba.conf` del primario debe permitir la conexión de replicación.

Cuando PostgreSQL arranca en `db2`, `standby.signal` hace que lo haga como servidor en espera y reciba la transmisión del primario. Como `hot_standby` está activado de forma predeterminada, acepta consultas de solo lectura en cuanto alcanza un estado coherente. En el primario, una fila por servidor en espera muestra su estado:

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

Una réplica en buen estado muestra `state` como `streaming` y `sync_state` como `async`. `replay_lag` aproxima cuánto tardaron los commits recientes en ser visibles allí. Una vez que el servidor en espera se ha puesto al día y cesa la actividad WAL, vuelve a NULL en lugar de mostrar cero.

La nota sobre optimizar la base de datos antes de dividirla en fragmentos explica por qué lo hace una tienda. La mayor parte de su tráfico son lecturas: la página del producto, la cesta y el historial de pedidos. Nada de eso necesita el primario, que debe quedar libre para las escrituras. Una réplica por transmisión asume esas lecturas, y la nota señala que añadirla es configuración, mientras que crear un

fragmento es un proyecto. Añade dos cautelas. Una réplica va un poco por detrás, por lo que lo que un usuario acaba de escribir debería leerse del primario. Y un informe largo en una réplica puede cancelarse si bloquea los cambios que esta reproduce.

Hacer que un commit espere a un servidor en espera requiere un parámetro en el primario. Cada servidor en espera se identifica mediante `application_name` en su `primary_conninfo`, y el primario enumera los nombres de los servidores a los que puede esperar:

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

Ahora cada commit espera hasta que su WAL esté en disco en el primario y en al menos uno de `db2` y `db3`. `synchronous_commit` puede establecerse por usuario, por base de datos o por transacción, de modo que el cobro de un pedido pueda esperar mientras un contador de visitas confirma a velocidad local. La nota sobre vender la última entrada exactamente una vez muestra por qué importa para las existencias. Con replicación asíncrona, un primario puede confirmar una venta y morir antes de que su réplica la tenga. La réplica es promovida, la venta desaparece y se vuelve a vender la misma unidad. La solución de la nota es un servidor en espera síncrono o un almacén basado en Raft, a costa de mayor latencia en cada commit.

La replicación lógica requiere dos instrucciones. El publicador necesita `wal_level = logical`, y las tablas ya deben existir en el suscriptor, porque sus definiciones no se copian:

```
-- 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;
```

La suscripción copia primero las filas que ya existen en esas tablas y después aplica todos los cambios posteriores.

Dónde surgen los problemas

Un usuario no ve lo que acaba de escribir. La aplicación escribe en el primario y vuelve a leer desde una réplica que aún lleva un pequeño retraso, por lo que parece que un ajuste guardado se revierte. Dirija las lecturas de un usuario al primario durante un breve periodo después de que escriba. Cuando los datos deban ser visibles en el servidor en espera en el momento en que el commit devuelve el control, `synchronous_commit = remote_apply` espera hasta que los servidores en espera síncronos hayan reproducido la transacción, con mayor coste por commit.

Se cancelan consultas largas en la réplica. El servidor en espera debe reproducir cambios incompatibles con una consulta en curso: vacuum elimina versiones de filas que la consulta todavía puede ver, o `DROP TABLE` y muchas formas de `ALTER TABLE` adquieren un bloqueo exclusivo. La reproducción espera hasta el límite `max_standby_streaming_delay`, fijado de forma predeterminada en 30 segundos, y después cancela la consulta. Una réplica para informes puede tener un retraso mayor y se rezaga más mientras espera. `hot_standby_feedback = on` evita las cancelaciones causadas por vacuum, pero la documentación advierte que puede provocar un crecimiento excesivo del almacenamiento del primario. `pg_stat_database_conflicts` en el servidor en espera cuenta las cancelaciones por causa, y la aplicación debería volver a intentar una consulta cancelada.

Una ranura olvidada llena el disco del primario. Una ranura retiene el WAL hasta que su servidor en espera lo recibe, aunque este esté apagado o haya desaparecido hace tiempo; `max_slot_wal_keep_size` vale -1 de forma predeterminada, lo que significa sin límite. Revise las ranuras periódicamente:

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

Elimine una ranura cuyo servidor en espera ya no exista. Establecer `max_slot_wal_keep_size` limita lo que puede retener una ranura. Si un servidor en espera se retrasa más allá de ese límite, su ranura se invalida y `wal_status` muestra `lost`. `idle_replication_slot_timeout`, desactivado de forma predeterminada, invalida las ranuras que lleven inactivas más tiempo del indicado.

Las escrituras se detienen si lo hace un servidor en espera síncrono. Si solo hay un servidor en espera designado y está caído, todos los commits del primario esperan y pueden no terminar nunca. Designe más candidatos que el número requerido, como en `ANY 1 (db2, db3)`. Si eso no es posible, reduzca el número o vacíe `synchronous_standby_names` y recargue la configuración.

Dos servidores creen ser el primario. PostgreSQL promueve un servidor en espera cuando se le pide, con `pg_ctl promote` o `SELECT pg_promote()`, pero no tiene software propio para detectar un primario caído y nada impide que el antiguo vuelva y acepte escrituras. La documentación advierte que esto produce «confusión y, en último término, pérdida de datos». Aísle el antiguo primario, apagándolo o cortando su red, antes de promover otro, y reconstrúyalo como servidor en espera antes de que se reincorpore.

La replicación lógica deja cosas atrás. Los cambios de esquema no se replican, de modo que añadir una columna en el publicador detiene la replicación hasta que el suscriptor también la tenga. Aplique primero al suscriptor los cambios que añadan elementos. Tampoco se replican las secuencias: las filas llegan con sus identificadores, pero las secuencias del suscriptor siguen en sus valores iniciales, y las inserciones después de un cambio de servidor fallan por claves duplicadas. Antes del cambio, sitúe cada secuencia más allá del mayor valor en uso.

Un maestro Redis se reinicia vacío y arrastra a sus réplicas. Si la persistencia está desactivada en el maestro y algo lo reinicia automáticamente, vuelve con un conjunto de datos vacío y sus réplicas lo copian, borrando sus propios datos. Mantenga activada la persistencia o no permita que el maestro se reinicie automáticamente.

Lecturas adicionales

- Alta disponibilidad — visión general de PostgreSQL sobre servidores primarios y en espera, replicación síncrona y asíncrona y conmutación por error.
- Servidores en espera con envío de registros — replicación por transmisión, ranuras de replicación y configuración de la replicación síncrona.
- Hot Standby — consultas de solo lectura en un servidor en espera, conflictos que las cancelan y parámetros que intercambian retraso por cancelaciones.
- pg_basebackup — copiar un clúster en funcionamiento para iniciar un servidor en espera, con todas las opciones usadas arriba.

- Restricciones — lo que no transporta la replicación lógica: esquema, secuencias, objetos grandes y cualquier cosa que no sean tablas.
- Replicación de Redis — cómo siguen las réplicas de Redis a un maestro, por qué la replicación es asíncrona y el peligro de reiniciar sin persistencia.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la replicación?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/replication/