Engineer

captura de cambios en datos

leer y transmitir los cambios de una base de datos

Qué es

La captura de cambios en datos, normalmente abreviada como CDC, consiste en leer los cambios realizados en una base de datos y transmitirlos a otros sistemas a medida que ocurren. Cada inserción, actualización y eliminación se convierte en un evento al que otro componente puede reaccionar: una copia para informes que se mantiene al día, un índice de búsqueda u otro servicio que necesita saber que se ha realizado un pedido. La base de datos donde ocurre el cambio sigue siendo el sistema de registro de referencia, y los demás sistemas la siguen.

Hay dos grandes formas de encontrar los cambios. La más sencilla es preguntar: ejecutar cada pocos minutos una consulta de las filas cuya marca de tiempo de «última actualización» sea posterior a la ejecución anterior. La otra es leer el registro que mantiene la propia base de datos de lo que ha cambiado. PostgreSQL conserva ese registro en su registro de escritura anticipada, o WAL, y su documentación establece la regla en la que se basa: los cambios en archivos de datos solo se escriben después de que los registros que los describen se hayan volcado al almacenamiento permanente. Como el registro ya contiene cada cambio en el orden en que ocurrió, leerlo proporciona un historial completo sin consultar las tablas. Este segundo enfoque, CDC basado en registros, es lo que suele significar el término hoy.

En PostgreSQL, el mecanismo se llama decodificación lógica, que la documentación define como «extraer todos los cambios persistentes de las tablas de una base de datos» a un formato que puede leerse sin conocer sus detalles internos. Tres partes trabajan juntas. El servidor debe escribir suficiente detalle en el WAL, algo que controla el ajuste `wal_level` : su valor predeterminado, `replica` , basta para copiar datos entre servidores, y `logical` añade lo necesario para decodificar. Un slot de replicación representa un flujo de

cambios de una base de datos, entregado a su lector en la secuencia original, y recuerda hasta dónde ha leído ese cliente. Un complemento de salida convierte los registros internos del WAL al formato que quiera el cliente.

La mayoría de los equipos no escribe el cliente por su cuenta. Debezium proporciona a Apache Kafka Connect conectores que obtienen los cambios mediante el mecanismo CDC de cada base de datos de origen. Su conector de PostgreSQL lee mediante `pgoutput`, que su documentación describe como el complemento de salida integrado de PostgreSQL, o mediante `decoderbufs`, un complemento mantenido por la comunidad de Debezium. Debezium también puede ejecutarse como biblioteca dentro de una aplicación JVM o como Debezium Server, que envía eventos de cambio a sistemas como Amazon Kinesis, Google Cloud Pub/Sub y Apache Pulsar.

Utilice el concepto en una decisión

Aplique «¿Qué es la captura de cambios en datos?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Las cifras llegan donde hacen falta según un calendario, sin que nadie tenga que copiar una hoja de cálculo al terminar el mes.

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. Un plan de infraestructura para 14 departamentos — 14 departamentos de investigación con un único plan de infraestructura

Un plan de infraestructura para 14 departamentos Póngase en contacto →

Por qué importa

El sondeo periódico parece más barato y, para una tabla pequeña que cambia despacio, a menudo lo es. La documentación de Debezium enumera lo que ofrece la lectura del registro y no puede ofrecer el sondeo. Captura cada cambio, mientras que una consulta solo ve el estado de una fila cuando se ejecuta, de modo que una fila modificada tres veces entre dos ejecuciones aparece una sola vez. Captura las eliminaciones, mientras que una fila borrada simplemente falta en el siguiente resultado. No necesita una columna adicional en el modelo, como un campo «Last Updated» que todas las rutas de escritura deben acordarse de actualizar. Puede transportar el estado anterior de una fila y metadatos como el ID de la transacción que hizo el cambio. Y entrega cambios con muy poco retraso, sin el coste de CPU del sondeo frecuente: para PostgreSQL y MySQL, Debezium sitúa ese retraso en el orden de los milisegundos.

El segundo motivo es la concordancia entre una base de datos y los mensajes que reciben otros sistemas. Debezium compara su enfoque con las escrituras dobles: la aplicación guarda un cambio en su base de datos y después envía un mensaje al respecto. Si el proceso se detiene entre esos pasos, la base de datos y el flujo de mensajes discrepan, y nada registra cuál es incorrecto. CDC elimina la segunda escritura. La aplicación confirma una vez y el mensaje se deriva de lo que se confirmó. El patrón outbox va un paso más allá: la aplicación escribe el propio mensaje en una tabla outbox dentro de la misma transacción que el cambio, y CDC publica las filas de esa tabla. Outbox Event Router de Debezium describe el resultado como evitar incoherencias entre el estado propio de un servicio y los eventos que consumen otros servicios.

El tercer motivo es el orden. Un slot de replicación conserva la secuencia del servidor de origen al entregar los cambios, agrupados por la transacción que los realizó, como muestra el ejemplo siguiente. Un consumidor que los aplica en ese orden reconstruye el mismo estado paso a paso. Para una carga ETL, esto hace que la carga incremental refleje lo ocurrido: un proceso nocturno puede tomar exactamente los cambios desde su última ejecución, incluidas las eliminaciones, en lugar de volver a cargar tablas enteras o confiar en una columna de marca de tiempo que alguna ruta de código olvidó actualizar.

Cómo se aplica en la práctica

El ejemplo funcional más pequeño solo utiliza PostgreSQL. Necesita `wal_level = logical` en la configuración del servidor, que solo entra en vigor después de reiniciar, y un slot de replicación libre; el valor predeterminado de `max_replication_slots` es 10. La sesión siguiente se ejecuta como superusuario, igual que el ejemplo de la propia documentación, y utiliza `test_decoding`, el complemento de salida de ese ejemplo, que imprime cada cambio como una línea de texto.

```
-- postgresql.conf: wal_level = logical (takes effect after a restart)
SELECT * FROM pg_create_logical_replication_slot('shop_cdc', 'test_decoding');

BEGIN;
INSERT INTO orders (id, status) VALUES (42, 'pending');
INSERT INTO outbox (id, event) VALUES (7, 'order 42 placed');
COMMIT;

SELECT data FROM pg_logical_slot_peek_changes('shop_cdc', NULL, NULL);
-- BEGIN 10301
-- table public.orders: INSERT: id[integer]:42 status[text]:'pending'
-- table public.outbox: INSERT: id[integer]:7 event[text]:'order 42 placed'
```

```
-- COMMIT 10301

SELECT data FROM pg_logical_slot_get_changes('shop_cdc', NULL, NULL);
-- the same four rows; this call consumes them, so a second call returns none

SELECT pg_drop_replication_slot('shop_cdc');
```

El pedido y su mensaje outbox salen juntos, dentro de la transacción que los escribió. `pg_logical_slot_peek_changes` lee sin avanzar la posición del slot, de modo que devuelve las mismas filas en la llamada siguiente, mientras que `pg_logical_slot_get_changes` las consume. Un `CREATE TABLE` solo produciría un `BEGIN` y un `COMMIT` vacíos porque la decodificación lógica no transporta DDL. La última línea es la más importante: un slot abandonado sigue reteniendo recursos del servidor.

En producción, Debezium hace de lector. En el primer arranque, su conector de PostgreSQL toma una instantánea coherente de las tablas que captura y después transmite cambios desde el punto en que se tomó esa instantánea. El valor predeterminado de `snapshot.mode` , `initial` , solo toma esa instantánea cuando no hay posiciones de lectura guardadas, por lo que un reinicio reanuda desde la última posición registrada. Cada evento incluye imágenes `before` y `after` de la fila, un bloque `source` , una marca de tiempo y un código `op` : `c` para creación, `u` para actualización, `d` para eliminación, `r` para una fila leída durante una instantánea y `t` para truncado. Tras una eliminación, el conector también emite un tombstone: un evento con la misma clave y un valor nulo, para que la compactación del registro de Kafka pueda retirar los mensajes anteriores de esa clave.

Ambos patrones de este artículo aparecen en las propias páginas de la empresa. Entre 2022 y 2024, la fundadora desarrolló 600 flujos ETL/ELT en PL/pgSQL en varios entornos de desarrollo y producción de PostgreSQL, incluidas cargas incrementales con CDC y deltas basados en marcas de tiempo. La nota Vender la última entrada exactamente una vez diseña una venta relámpago alrededor de una outbox: cada paso registra su mensaje en la outbox dentro de la misma transacción que el cambio, la captura de cambios en datos publica la outbox y, como un mensaje puede llegar más de una vez, cada consumidor es idempotente, con el pedido como clave.

Dónde surgen los problemas

Nada limita lo que puede retener un slot. Un slot conserva cada fragmento de WAL que su consumidor no haya confirmado, y los valores predeterminados de PostgreSQL no imponen un límite. `max_slot_wal_keep_size` vale `-1`, lo que permite retener una cantidad ilimitada de WAL, y `idle_replication_slot_timeout` vale `0`, lo que desactiva la invalidación de los slots que no se utilizan. Un conector detenido durante un despliegue fallido puede, por tanto, llenar el disco mucho después de que todos lo hayan olvidado. Vigile `pg_replication_slots` para encontrar slots que ya no estén activos y ajuste `max_slot_wal_keep_size` a lo que admita el disco, sabiendo que un consumidor que se retrase más puede dejar de poder continuar. Recuperarse de ello exige un slot nuevo, y un slot nuevo empieza en el presente: Debezium advierte que eliminar y recrear un slot puede omitir silenciosamente los eventos intermedios, por lo que después debe tomarse una instantánea nueva.

Una tabla con pocos cambios retiene el registro. La documentación de Debezium describe una variante más sutil. Un slot pertenece a una base de datos, y el conector solo avanza su posición confirmada cuando ve cambios que captura. Si las tablas capturadas cambian rara vez mientras otras tablas u otras bases de datos del mismo servidor escriben intensamente, la posición apenas se mueve y el WAL que queda detrás sigue creciendo. La solución es una señal de actividad: `heartbeat.interval.ms` hace que el conector emita eventos periódicos de actividad, y `heartbeat.action.query` permite escribir regularmente en una tabla pequeña de la base de datos capturada, que debe formar parte de la publicación que lee el conector, para que el slot avance aunque no ocurra nada más allí.

Los consumidores suponen que cada cambio llega una sola vez. Un slot lógico emite cada cambio una vez en funcionamiento normal, pero su posición solo se guarda en los puntos de control, por lo que, tras un fallo, puede volver a un punto anterior y reenviar cambios recientes. La documentación de PostgreSQL deja las consecuencias en manos del cliente. Un consumidor que suma cada importe a un total acumulado cuenta esos cambios dos veces. Haga que todos los consumidores sean idempotentes: aplique cada cambio como un upsert con la clave primaria de la fila o registre los ID de los eventos ya procesados, para que una segunda entrega no cambie nada.

Las actualizaciones y eliminaciones llegan sin los valores anteriores. Lo que dice un evento de actualización o eliminación sobre la fila previa depende de la identidad de réplica de la tabla. Con `REPLICA IDENTITY DEFAULT`, los eventos de Debezium contienen como máximo los valores anteriores de la clave primaria, y

el conector no produce eventos de actualización ni eliminación cuando la tabla no tiene clave primaria. `REPLICA IDENTITY FULL` coloca todos los valores anteriores de las columnas en `before`, con un coste: la documentación de PostgreSQL señala que aumenta el volumen del WAL, en especial cuando se ejecutan muchas instrucciones `UPDATE` y `DELETE`. Dé una clave primaria a cada tabla capturada y cambie a `FULL` solo las tablas cuyos consumidores necesiten los valores antiguos.

Lecturas adicionales

- Funciones de Debezium — lo que captura CDC basado en registros y omiten el sondeo y las escrituras dobles, además de instantáneas, filtros y enmascaramiento.
- Conector de Debezium para PostgreSQL — complementos de salida, instantáneas, formato de los eventos de cambio, identidad de réplica y espacio en disco del WAL.
- Conceptos de decodificación lógica — slots de replicación, complementos de salida y por qué debe eliminarse un slot que nadie necesita.
- Ejemplos de decodificación lógica — crear un slot, consultar y consumir los cambios y volver a eliminarlo.
- Replicación: configuración del servidor — `max_replication_slots`, `max_slot_wal_keep_size` e `idle_replication_slot_timeout`.
- Outbox Event Router — el patrón outbox y cómo Debezium dirige las filas de la outbox a temas.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la captura de cambios en datos?» 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/change-data-capture/