Engineer

Change Data Capture

Datenbankänderungen als Ereignisse an andere Systeme leiten

Was es ist

Change Data Capture, meist als CDC abgekürzt, bedeutet, Änderungen in einer Datenbank zu lesen und an andere Systeme weiterzugeben, sobald sie geschehen. Jedes Einfügen, Aktualisieren und Löschen wird zu einem Ereignis, auf das etwas anderes reagieren kann: eine aktuell gehaltene Berichtskopie, ein Suchindex oder ein anderer Dienst, der wissen muss, dass eine Bestellung aufgegeben wurde. Die Datenbank, in der die Änderung geschieht, bleibt das führende System. Die anderen Systeme folgen ihr.

Es gibt zwei grundsätzliche Wege, Änderungen zu finden. Der einfachere ist Nachfragen: Alle paar Minuten läuft eine Abfrage nach Zeilen, deren Zeitstempel der letzten Aktualisierung nach dem Zeitpunkt des vorherigen Durchlaufs liegt. Der andere liest die eigenen Aufzeichnungen der Datenbank über ihre Änderungen. PostgreSQL hält diese in seinem Write-Ahead-Log, kurz WAL. Seine Dokumentation nennt die Regel dahinter: Änderungen an Datendateien werden erst geschrieben, nachdem die sie beschreibenden Protokolleinträge dauerhaft gespeichert wurden. Weil das Protokoll bereits jede Änderung in ihrer ursprünglichen Reihenfolge enthält, liefert sein Auslesen eine vollständige Historie, ohne die Tabellen abzufragen. Dieser zweite Ansatz, protokollbasiertes CDC, ist heute meist gemeint.

In PostgreSQL heißt der Mechanismus logische Dekodierung. Die Dokumentation definiert ihn als das „Extrahieren aller dauerhaften Änderungen an den Tabellen einer Datenbank“ in ein Format, das sich ohne Kenntnis der Datenbankinterna lesen lässt. Drei Teile arbeiten zusammen. Der Server muss genügend Details ins WAL schreiben. Das steuert die Einstellung `wal_level` : Ihr Standardwert `replica` reicht zum Kopieren von Daten zwischen Servern; `logical` ergänzt, was die Dekodierung benötigt. Ein Replikationsslot steht für einen Änderungsstrom aus einer Datenbank, der seinen Leser in der ursprünglichen

Reihenfolge erreicht. Er merkt sich, wie weit dieser Client gelesen hat. Ein Ausgabe-Plugin wandelt die internen WAL-Einträge in das vom Client gewünschte Format um.

Die meisten Teams schreiben den Client nicht selbst. Debezium liefert für Apache Kafka Connect Konnektoren, die Änderungen über den CDC-Mechanismus der jeweiligen Quelldatenbank beziehen. Sein PostgreSQL-Konnektor liest über `pgoutput`, laut Dokumentation das eingebaute Ausgabe-Plugin von PostgreSQL, oder über `decoderbufs`, ein von der Debezium-Community gepflegtes Plugin. Debezium kann auch als Bibliothek in einer JVM-Anwendung laufen oder als Debezium Server, der Änderungsereignisse an Systeme wie Amazon Kinesis, Google Cloud Pub/Sub und Apache Pulsar sendet.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Change Data Capture?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Zahlen kommen termingerecht dort an, wo sie gebraucht werden, ohne dass am Monatsende jemand eine Tabelle kopieren muss.

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. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

Warum es wichtig ist

Polling wirkt günstiger und ist es bei einer kleinen, selten geänderten Tabelle häufig auch. Debeziums Dokumentation zählt auf, was das Lesen des Protokolls leistet und Polling nicht. Es erfasst jede Änderung. Eine Abfrage sieht dagegen nur den Zustand einer Zeile im Moment ihrer Ausführung: Eine zwischen zwei Durchläufen dreimal geänderte Zeile erscheint einmal. Es erfasst Löschungen, während eine gelöschte Zeile im nächsten Ergebnis einfach fehlt. Es braucht keine zusätzliche Spalte im Datenmodell, etwa ein Feld für die letzte Aktualisierung, das jeder Schreibpfad setzen muss. Es kann den alten Zustand einer Zeile und Metadaten wie die ID der auslösenden Transaktion mitführen. Und es liefert Änderungen mit sehr geringer Verzögerung ohne die CPU-Kosten häufiger Abfragen. Für PostgreSQL und MySQL gibt Debezium diese Verzögerung im Millisekundenbereich an.

Der zweite Grund ist die Übereinstimmung zwischen einer Datenbank und den Nachrichten, die andere Systeme erhalten. Debezium nennt doppeltes Schreiben als Vergleichsansatz: Die Anwendung speichert eine Änderung in ihrer Datenbank und sendet danach eine Nachricht darüber. Stoppt der Prozess zwischen diesen Schritten, widersprechen sich Datenbank und Nachrichtenstrom. Nirgends ist festgehalten, welcher der beiden Zustände falsch ist. CDC beseitigt den zweiten Schreibvorgang. Die Anwendung schreibt die Transaktion einmal fest, und die Nachricht wird aus dem festgeschriebenen Zustand abgeleitet. Das Outbox-Muster geht weiter: Die Anwendung schreibt die Nachricht selbst in derselben Transaktion wie die Änderung in eine Outbox-Tabelle. CDC veröffentlicht deren Zeilen. Debeziums Outbox Event Router beschreibt das Ergebnis als Vermeidung von Widersprüchen zwischen dem eigenen Zustand eines Dienstes und den von anderen Diensten konsumierten Ereignissen.

Der dritte Grund ist die Reihenfolge. Ein Replikationsslot liefert die Änderungen in der Reihenfolge des Quellservers und nach der auslösenden Transaktion gruppiert aus, wie das folgende Beispiel zeigt. Ein Verbraucher, der die Änderungen in dieser Reihenfolge anwendet, baut Schritt für Schritt denselben Zustand wieder auf. Bei einem ETL-Ladevorgang macht das inkrementelles Laden verlässlich: Ein nächtlicher Auftrag kann genau die Änderungen seit seinem letzten Durchlauf übernehmen, einschließlich Löschungen, statt ganze Tabellen neu zu laden oder einer Zeitstempelspalte zu vertrauen, deren Aktualisierung ein Codepfad vergessen hat.

Wie es in der Praxis aussieht

Das kleinste funktionsfähige Beispiel verwendet nur PostgreSQL. Es benötigt `wal_level = logical` in der Serverkonfiguration, das erst nach einem Neustart wirksam wird, sowie einen freien Replikationsslot. `max_replication_slots` hat den Standardwert 10. Die folgende Sitzung läuft wie das Beispiel der Dokumentation als Superuser und verwendet dessen Ausgabe-Plugin `test_decoding`, das jede Änderung als Textzeile ausgibt.

```
-- 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');
```

Die Bestellung und ihre Outbox-Nachricht erscheinen gemeinsam innerhalb der Transaktion, die sie geschrieben hat. `pg_logical_slot_peek_changes` liest, ohne den Slot weiterzusetzen, und liefert beim nächsten Aufruf dieselben Zeilen. `pg_logical_slot_get_changes` verbraucht sie dagegen. Ein `CREATE TABLE` würde nur `BEGIN` und `COMMIT` ohne Änderungen dazwischen erzeugen, weil logische Dekodierung keine DDL überträgt. Die letzte Zeile ist die wichtigste: Ein zurückgelassener Slot hält weiter Serverressourcen fest.

In Produktion übernimmt Debezium das Lesen. Beim ersten Start erstellt sein PostgreSQL-Konnektor einen konsistenten Snapshot der erfassten Tabellen und streamt danach Änderungen ab dem Zeitpunkt dieses Snapshots. Der Standardwert für `snapshot.mode` , `initial` , erstellt den Snapshot nur, wenn keine gespeicherten Offsets existieren. Ein Neustart setzt deshalb stattdessen an der zuletzt aufgezeichneten Position fort. Jedes Ereignis enthält `before` - und `after` -Abbilder der Zeile, einen `source` -Block, einen Zeitstempel und einen `op` -Code: `c` für Anlegen, `u` für Aktualisieren, `d` für Löschen, `r` für eine beim Snapshot gelesene Zeile und `t` für Truncate. Nach einer Löschung sendet der Konnektor außerdem einen Tombstone: ein Ereignis mit demselben Schlüssel und dem Wert null, damit Kafkas Log-Kompaktierung frühere Nachrichten für diesen Schlüssel entfernen kann.

Beide Muster aus diesem Artikel kommen auf den eigenen Seiten des Unternehmens vor. Zwischen 2022 und 2024 entwickelte die Gründerin 600 ETL/ELT-Pipelines in PL/pgSQL über mehrere PostgreSQL-Entwicklungs- und Produktionsumgebungen hinweg. Dazu gehörten inkrementelle Ladevorgänge mit CDC und zeitstempelbasierten Deltas. Die Notiz Das letzte Ticket genau einmal verkaufen entwirft einen Blitzverkauf rund um die Outbox: Jeder Schritt hält seine Nachricht in derselben Transaktion wie die Änderung in der Outbox fest. Change Data Capture veröffentlicht die Outbox. Weil eine Nachricht mehrmals eintreffen kann, ist jeder Verbraucher idempotent, mit der Bestellung als Schlüssel.

Wo es schiefgeht

Nichts begrenzt, was ein Slot behalten darf. Ein Slot hält jeden Teil des WAL fest, den sein Verbraucher noch nicht bestätigt hat. Die PostgreSQL-Standardwerte setzen dafür keine Obergrenze. `max_slot_wal_keep_size` ist `-1` und erlaubt Slots damit unbegrenzt viel WAL. `idle_replication_slot_timeout` ist `0` und schaltet so die Invalidierung ungenutzter Slots ab. Ein während einer fehlgeschlagenen Bereitstellung gestoppter Konnektor kann deshalb die Platte füllen, lange nachdem ihn alle vergessen haben. Beobachten Sie `pg_replication_slots` auf nicht mehr aktive Slots und begrenzen Sie `max_slot_wal_keep_size` auf das, was die Platte verkraftet. Ein Verbraucher, dessen Rückstand diese Grenze überschreitet, kann dann womöglich nicht mehr fortsetzen. Zur Wiederherstellung braucht es einen neuen Slot, und dieser beginnt in der Gegenwart. Debezium warnt, dass ein gelöschter und neu angelegter Slot die dazwischenliegenden Ereignisse stillschweigend überspringen kann. Erstellen Sie deshalb danach einen neuen Snapshot.

Eine ruhige Tabelle hält das Protokoll zurück. Debeziums Dokumentation beschreibt eine subtilere Variante. Ein Slot gehört zu einer Datenbank. Der Konnektor setzt seine bestätigte Position nur weiter, wenn er Änderungen sieht, die er erfasst. Ändern sich die erfassten Tabellen selten, während andere Tabellen oder andere Datenbanken desselben Servers viel schreiben, bewegt sich die Position kaum und das zurückgehaltene WAL wächst weiter. Die Abhilfe ist ein Lebenszeichen: `heartbeat.interval.ms` lässt den Konnektor in festgelegten Abständen Heartbeat-Ereignisse senden. `heartbeat.action.query` erlaubt ihm, regelmäßig in eine kleine Tabelle der erfassten Datenbank zu schreiben. Diese muss Teil der vom Konnektor gelesenen Publikation sein, damit der Slot auch weiterläuft, wenn dort sonst nichts geschieht.

Verbraucher erwarten jede Änderung genau einmal. Ein logischer Slot gibt im Normalbetrieb jede Änderung einmal aus. Seine Position wird aber nur an Checkpoints gespeichert. Nach einem Absturz kann er deshalb zu einem früheren Punkt zurückkehren und jüngste Änderungen erneut senden. PostgreSQLs Dokumentation überlässt die Folgen dem Client. Ein Verbraucher, der jeden Betrag zu einer laufenden Summe addiert, zählt diese Änderungen doppelt. Machen Sie jeden Verbraucher idempotent: Wenden Sie jede Änderung als Upsert mit dem Primärschlüssel der Zeile als Schlüssel an oder halten Sie die IDs bereits verarbeiteter Ereignisse fest, damit eine zweite Zustellung nichts verändert.

Aktualisierungen und Löschungen kommen ohne die alten Werte an. Was ein Aktualisierungs- oder Löschereignis über die vorherige Zeile aussagt, hängt von der Replikationsidentität der Tabelle ab. Bei `REPLICA IDENTITY DEFAULT` enthalten Debeziums Ereignisse höchstens die bisherigen Primärschlüsselwerte. Hat die Tabelle keinen Primärschlüssel, erzeugt der Konnektor weder Aktualisierungs- noch Löschereignisse. `REPLICA IDENTITY FULL` nimmt die vorherigen Werte sämtlicher Spalten in `before` auf, hat aber einen Preis: PostgreSQLs Dokumentation weist auf mehr WAL-Volumen hin, besonders bei vielen `UPDATE` - und `DELETE` -Anweisungen. Geben Sie jeder erfassten Tabelle einen Primärschlüssel und wechseln Sie nur bei den Tabellen zu `FULL` , deren Verbraucher die alten Werte benötigen.

Weiterführende Lektüre

- Debezium-Funktionen — was protokollbasiertes CDC erfasst und Polling sowie doppeltem Schreiben entgeht, außerdem Snapshots, Filter und Maskierung.
- Debezium-Konnektor für PostgreSQL — Ausgabe-Plugins, Snapshots, Änderungsereignisformat, Replikationsidentität und WAL-Plattenplatz.
- Konzepte der logischen Dekodierung — Replikationsslots, Ausgabe-Plugins und warum ein nicht mehr benötigter Slot gelöscht werden sollte.
- Beispiele der logischen Dekodierung — einen Slot erstellen, Änderungen vorab lesen oder verbrauchen und den Slot wieder löschen.
- Replikation: Serverkonfiguration — `max_replication_slots` , `max_slot_wal_keep_size` und `idle_replication_slot_timeout` .
- Outbox Event Router — das Outbox-Muster und Debeziums Weiterleitung von Outbox-Zeilen an Topics.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Change Data Capture?“ 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/change-data-capture/