Engineer

Change data capture

att läsa av ändringarna i en databas och skicka dem vidare

Vad det är

Change data capture, oftast förkortat CDC, innebär att läsa av ändringarna i en databas och skicka dem vidare till andra system i takt med att de sker. Varje insättning, uppdatering och borttagning blir en händelse som något annat kan reagera på: en rapportkopia som håller sig aktuell, ett sökindex, en annan tjänst som behöver veta att en order har lagts. Databasen där ändringen sker förblir den auktoritativa källan, och de andra systemen följer efter.

Det finns i stort sett två sätt att hitta ändringarna. Det enkla är att fråga: med några minuters mellanrum körs en databasfråga efter rader vars tidsstämpel för senaste uppdatering är nyare än förra körningen. Det andra är att läsa databasens egen förteckning över vad den har ändrat. PostgreSQL för den förteckningen i sin transaktionslogg, write-ahead log (WAL), och dokumentationen beskriver principen bakom: ändringar i datafilerna skrivs först när posterna som beskriver dem har skrivits till permanent lagring. Eftersom loggen redan innehåller varje ändring i den ordning den skedde, ger den en fullständig historik utan att tabellerna behöver läsas alls. Det är det andra tillvägagångssättet, loggbaserad CDC, som man oftast menar med uttrycket i dag.

I PostgreSQL heter mekanismen logisk avkodning. Dokumentationen beskriver den som processen som tar ut alla beständiga ändringar i tabellerna i en databas i ett sammanhängande format som kan läsas utan kännedom om databasens inre tillstånd. Tre delar samverkar. Loggen på servern måste innehålla tillräckligt med detaljer, och det styrs av inställningen `wal_level` : standardvärdet, `replica` , räcker för att kopiera data mellan servrar, medan `logical` lägger till det som avkodningen behöver. En replikeringsslot (replication slot) står för en ström av ändringar från en databas, som spelas upp för en klient i den ordning de gjordes, och sloten minns hur långt klienten har kommit. Ett output-plugin översätter loggens interna poster till det format som klienten vill ha.

De flesta skriver inte klienten själva. Debezium beskriver sig som en samling källkonnektorer för Apache Kafka Connect, där var och en läser ändringar från en viss databas via databasens egna CDC-funktioner. Konnektorn för PostgreSQL läser antingen via `pgoutput`, som Debeziums dokumentation kallar det inbyggda output-pluginet i PostgreSQL, eller via `decoderbufs`, ett plugin som underhålls av Debezium-gemenskapen. Debezium kan också köras som ett bibliotek i en JVM-applikation eller som Debezium Server, som skickar ändringshändelserna vidare till system som Amazon Kinesis, Google Cloud Pub/Sub och Apache Pulsar.

Använd begreppet i ett beslut

Använd ”Vad är change data capture?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Siffror som kommer fram dit de behövs, på fasta tider, utan att någon kopierar ett kalkylblad i slutet av månaden.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. En infrastrukturplan för 14 avdelningar — 14 forskningsavdelningar under en och samma infrastrukturplan

En infrastrukturplan för 14 avdelningar Hör av er →

Varför det spelar roll

Polling ser billigare ut, och för en liten tabell som sällan ändras stämmer det ofta. Debeziums dokumentation räknar upp vad läsning av loggen klarar som polling inte gör. Den fångar varje ändring, medan en databasfråga bara ser radens tillstånd i det ögonblick den körs, så en rad som ändrades tre gånger mellan två körningar bara dyker upp en gång. Den fångar borttagningar, medan en borttagen rad helt enkelt saknas i nästa resultat. Den kräver ingen extra kolumn i datamodellen, som ett ”Last Updated”-fält som all kod som skriver till tabellen måste komma ihåg att fylla i. Den kan ta med radens tidigare tillstånd och metadata, till exempel vilken transaktion som gjorde ändringen. Och den levererar ändringarna med mycket liten fördröjning utan den CPU-belastning som täta pollningar ger: för PostgreSQL och MySQL ligger fördröjningen enligt Debezium i millisekundområdet.

Det andra skälet är att databasen och meddelandena som andra system tar emot säger samma sak. Debezium nämner dubbelskrivning (dual writes) som det andra tillvägagångssättet den jämför sig med: applikationen sparar en ändring i sin databas och skickar sedan ett meddelande om den. Stannar processen mellan de två stegen är databasen och meddelandeströmmen oense, och ingenting registrerar vilken av dem som har fel. CDC tar bort den andra skrivningen. Applikationen sparar en gång, och meddelandet härleds från det som faktiskt sparades. Outbox-mönstret går ett steg längre: applikationen skriver själva

meddelandet i en outbox-tabell i samma transaktion som ändringen, och CDC publicerar raderna i den tabellen. Debeziums Outbox Event Router beskriver syftet som att undvika inkonsekvenser mellan en tjänsts eget tillstånd och de händelser som andra tjänster konsumerar.

Det tredje skälet är ordningen. En replikeringsslot lämnar ut ändringarna i den ordning de gjordes på källservern, grupperade efter transaktionen som gjorde dem, som exemplet nedan visar. En konsument som tillämpar dem i den ordningen bygger upp samma tillstånd steg för steg. För en ETL-laddning gör det den inkrementella laddningen ärlig: ett nattligt jobb kan ta exakt de ändringar som har kommit sedan förra körningen, borttagningar inräknade, i stället för att ladda om hela tabeller eller lita på en tidsstämpelkolumn som en kodväg glömde att uppdatera.

Så ser det ut i praktiken

Det minsta exemplet som fungerar använder bara PostgreSQL. Det kräver `wal_level = logical` i serverns konfiguration, vilket träder i kraft först efter en omstart, och en ledig replikeringsslot; `max_replication_slots` är 10 som standard. Sessionen nedan körs som superanvändare, precis som dokumentationens eget exempel, och använder `test_decoding`, output-pluginet från det exemplet, som skriver ut varje ändring som en textrad.

```
-- postgresql.conf: wal_level = logical (träder i kraft efter en omstart)
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 lagd');
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 lagd'
-- COMMIT 10301

SELECT data FROM pg_logical_slot_get_changes('shop_cdc', NULL, NULL);
-- samma fyra rader; anropet förbrukar dem, så ett nytt anrop ger inga

SELECT pg_drop_replication_slot('shop_cdc');
```

Ordern och dess outbox-meddelande kommer ut tillsammans, inom transaktionen som skrev dem. `pg_logical_slot_peek_changes` läser utan att flytta sloten och ger därför samma rader vid nästa anrop, medan `pg_logical_slot_get_changes` förbrukar dem. En `CREATE TABLE` skulle bara ge ett tomt `BEGIN` och `COMMIT`, eftersom logisk avkodning inte tar med DDL. Den sista raden är viktigast: en slot som blir kvar fortsätter att hålla fast vid serverns resurser.

I produktion är det Debezium som läser. Första gången konnektorn för PostgreSQL startar tar den en konsistent ögonblicksbild (snapshot) av tabellerna den fångar ändringar från och strömmar sedan ändringar från den punkt där ögonblicksbilden togs. Standardvärdet för `snapshot.mode`, `initial`, tar bara ögonblicksbilden när det inte finns några sparade offsets, så en omstart fortsätter från den senast registrerade positionen. Varje händelse har en `before`- och en `after`-bild av raden, ett `source`-block, en tidsstämpel och en `op`-kod: `c` för skapande, `u` för uppdatering, `d` för borttagning, `r` för en rad som lästs under en ögonblicksbild och `t` för truncate. Efter en borttagning skickar konnektorn dessutom en tombstone, en händelse med samma nyckel och värdet null, så att log compaction i Kafka kan ta bort de tidigare meddelandena med den nyckeln.

Båda mönstren från den här artikeln återkommer på företagets egna sidor. Mellan 2022 och 2024 byggde grundaren 600 ETL/ELT-pipelines i PL/pgSQL i flera utvecklings- och produktionsmiljöer för PostgreSQL, däribland inkrementella laddningar med CDC och tidsstämpelbaserade deltan. Anteckningen Att sälja den sista biljetten exakt en gång bygger en blixtrea kring en outbox: varje steg lägger sitt meddelande i outboxen i samma transaktion som ändringen, change data capture publicerar outboxen, och eftersom ett meddelande kan komma fram mer än en gång är varje konsument idempotent, med ordern som nyckel.

Var det går fel

Ingenting sätter tak för hur mycket en slot får behålla. En slot håller fast vid varje del av transaktionsloggen som dess konsument inte har bekräftat, och standardinställningarna i PostgreSQL sätter ingen gräns för det. `max_slot_wal_keep_size` är `-1`, så slots får behålla en obegränsad mängd logg, och `idle_replication_slot_timeout` är `0`, vilket stänger av ogiltigförklaringen av slots som står oanvända. En konnektor som stannade under en misslyckad driftsättning kan därför fylla disken långt efter att alla har glömt den. Håll koll på `pg_replication_slots` efter slots som inte längre är aktiva, och sätt `max_slot_wal_keep_size` till vad disken tål, med vetskap om att en konsument som

hamnar längre efter än så kanske inte kan fortsätta. Vägen tillbaka går via en ny slot, och en ny slot börjar där loggen står nu: Debezium varnar för att en slot som tas bort och skapas på nytt i tysthet kan hoppa över händelserna däremellan, så ta en ny ögonblicksbild efteråt.

En stillsam tabell håller tillbaka loggen. Debeziums dokumentation beskriver en lömskare variant. En slot hör till en databas, och konnektorn flyttar bara fram sin bekräftade position när den ser ändringar som den fångar. Om tabellerna den fångar sällan ändras medan andra tabeller, eller andra databaser på samma server, skriver mycket, flyttar sig positionen knappt, och loggen bakom den fortsätter att växa. Lösningen är ett hjärtslag (heartbeat): `heartbeat.interval.ms` får konnektorn att skicka heartbeat-händelser med ett fast intervall, och `heartbeat.action.query` låter den skriva regelbundet till en liten tabell i den databas den fångar ändringar från. Tabellen måste ingå i den publikation (publication) som konnektorn läser, så att sloten flyttar sig framåt även när inget annat händer där.

Konsumenterna utgår från att varje ändring kommer en gång. En logisk slot skickar varje ändring en gång vid normal drift, men dess position sparas bara vid kontrollpunkter (checkpoints), så efter en krasch kan den gå tillbaka till en tidigare punkt och skicka de senaste ändringarna igen. Dokumentationen för PostgreSQL lämnar följderna åt klienten. En konsument som lägger till varje belopp i en löpande summa räknar de ändringarna två gånger. Gör varje konsument idempotent: tillämpa varje ändring som en upsert med radens primärnyckel som nyckel, eller registrera vilka händelser som redan har behandlats, så att en ny leverans inte ändrar något.

Uppdateringar och borttagningar kommer utan de gamla värdena. Vad en händelse för en uppdatering eller borttagning säger om den tidigare raden beror på tabellens replica identity. Med `REPLICA IDENTITY DEFAULT` innehåller Debeziums händelser högst primärnyckelns tidigare värden, och för en tabell utan primärnyckel skickar konnektorn inga händelser alls för uppdateringar eller borttagningar. `REPLICA IDENTITY FULL` lägger alla tidigare kolumner i `before`, men det kostar: dokumentationen för PostgreSQL påpekar att det ökar mängden WAL, särskilt när många `UPDATE`- och `DELETE`-satser körs. Ge varje tabell som fångas en primärnyckel, och byt till `FULL` bara på de tabeller vars konsumenter behöver de gamla värdena.

Läs mer

- Debezium Features — vad loggbaserad CDC fångar som polling och dubbelskrivning missar, plus ögonblicksbilder, filter och maskering.
- Debezium connector for PostgreSQL — output-plugins, ögonblicksbilder, händelseformatet, replica identity och diskutrymme för WAL.
- Logical Decoding Concepts — replikeringsslots, output-plugins och varför en slot som ingen behöver bör tas bort.
- Logical Decoding Examples — att skapa en slot, titta på och förbruka ändringar och ta bort sloten igen.
- Replication (server configuration) — `max_replication_slots`, `max_slot_wal_keep_size` och `idle_replication_slot_timeout`.
- Outbox Event Router — outbox-mönstret och hur Debezium skickar outbox-rader vidare till topics.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är change data capture?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/change-data-capture/