Engineer

Change data capture

å lese av endringene i en database og sende dem videre

Hva det er

Change data capture, som regel forkortet til CDC, betyr å lese endringene i en database og sende dem videre til andre systemer etter hvert som de skjer. Hver innsetting, oppdatering og sletting blir en hendelse som noe annet kan reagere på: en rapportkopi som holder seg oppdatert, en søkeindeks, en annen tjeneste som må vite at en ordre er lagt inn. Databasen der endringen skjer, forblir den autoritative kilden, og de andre systemene følger etter.

Det finnes grovt sett to måter å finne endringene på. Den enkle er å spørre: med noen minutters mellomrom kjører en spørring etter rader der tidsstempelet for siste oppdatering er nyere enn forrige kjøring. Den andre er å lese databasens egen oversikt over hva den har endret. PostgreSQL fører den oversikten i write-ahead-loggen (WAL), og dokumentasjonen beskriver prinsippet bak: endringer i datafilene skrives først når postene som beskriver dem, er skrevet til permanent lagring. Fordi loggen allerede inneholder hver endring i den rekkefølgen den skjedde, gir den en fullstendig historikk uten at tabellene trenger å leses i det hele tatt. Det er den andre tilnærmingen, loggbasert CDC, man som regel mener med uttrykket i dag.

I PostgreSQL heter mekanismen logisk dekoding. Dokumentasjonen beskriver den som prosessen som trekker ut alle varige endringer i tabellene til en database i et sammenhengende format som kan leses uten kjennskap til databasens indre tilstand. Tre deler virker sammen. Loggen på serveren må inneholde nok detaljer, og det styres av innstillingen `wal_level`: standardverdien, `replica`, holder for å kopiere data mellom servere, mens `logical` legger til det dekodingen trenger. Et replikeringsspor (replication slot) står for én strøm av endringer fra én database, som spilles av for en klient i den rekkefølgen de ble gjort, og sporet husker hvor langt klienten har kommet. En output-plugin omsetter de interne postene i loggen til det formatet klienten ønsker.

De fleste skriver ikke klienten selv. Debezium beskriver seg selv som en samling kildekonnektorer for Apache Kafka Connect, der hver enkelt leser endringer fra en bestemt database gjennom databasens egne CDC-funksjoner. Konnektoren for PostgreSQL leser enten gjennom `pgoutput`, som Debezium-dokumentasjonen omtaler som den innebygde output-pluginen i PostgreSQL, eller gjennom `decoderbufs`, en plugin som Debezium-fellesskapet vedlikeholder. Debezium kan også kjøre som et bibliotek i en JVM-applikasjon eller som Debezium Server, som sender endringshendelsene videre til systemer som Amazon Kinesis, Google Cloud Pub/Sub og Apache Pulsar.

Bruk begrepet i en beslutning

Bruk «Hva er change data capture?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Tall som kommer fram dit de trengs, til faste tider, uten at noen må kopiere et regneark ved månedsslutt.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

Polling ser billigere ut, og for en liten tabell som sjelden endres, stemmer det ofte. Debezium-dokumentasjonen lister opp hva lesing av loggen får til som polling ikke gjør. Den fanger hver endring, der en spørring bare ser tilstanden til raden i det øyeblikket den kjører, så en rad som ble endret tre ganger mellom to kjøringer, bare dukker opp én gang. Den fanger slettinger, der en slettet rad rett og slett mangler i neste resultat. Den krever ingen ekstra kolonne i datamodellen, som et «Last Updated»-felt som all kode som skriver til tabellen, må huske å fylle ut. Den kan ta med den tidligere tilstanden til raden og metadata, for eksempel hvilken transaksjon som gjorde endringen. Og den leverer endringene med svært liten forsinkelse uten den CPU-belastningen som hyppig polling gir: for PostgreSQL og MySQL ligger forsinkelsen ifølge Debezium i millisekundområdet.

Det andre argumentet er at databasen og meldingene som andre systemer mottar, sier det samme. Debezium nevner dobbeltskriving (dual writes) som den andre tilnærmingen den sammenligner seg med: applikasjonen lagrer en endring i databasen sin og sender deretter en melding om den. Stopper prosessen mellom de to stegene, er databasen og meldingsstrømmen uenige, og ingenting registrerer hvilken av dem som tar feil. CDC fjerner den andre skrivingen. Applikasjonen lagrer én gang, og meldingen avledes av det som faktisk ble

lagret. Outbox-mønsteret går et steg videre: applikasjonen skriver selve meldingen i en outbox-tabell i den samme transaksjonen som endringen, og CDC publiserer radene i den tabellen. Outbox Event Router fra Debezium beskriver formålet som å unngå inkonsistens mellom en tjenestes egen tilstand og hendelsene som andre tjenester mottar.

Det tredje argumentet er rekkefølgen. Et replikeringsspor leverer endringene i den rekkefølgen de ble gjort på kildeserveren, samlet etter transaksjonen som gjorde dem, slik eksemplet nedenfor viser. En konsument som tar dem i bruk i den rekkefølgen, gjenskaper den samme tilstanden steg for steg. For en ETL-lasting gjør det den inkrementelle lastingen ærlig: en nattlig jobb kan ta nøyaktig de endringene som har kommet siden forrige kjøring, slettinger medregnet, i stedet for å laste hele tabeller på nytt eller stole på en tidsstempelkolonne som én kodesti glemte å oppdatere.

Slik ser det ut i praksis

Det minste eksemplet som fungerer, bruker bare PostgreSQL. Det krever `wal_level = logical` i serverkonfigurasjonen, noe som først trer i kraft etter en omstart, og et ledig replikeringsspor; `max_replication_slots` er 10 som standard. Økten nedenfor kjører som superbruker, slik dokumentasjonens eget eksempel gjør, og bruker `test_decoding`, output-pluginen fra det eksemplet, som skriver ut hver endring som en tekstlinje.

```
-- postgresql.conf: wal_level = logical (trer i kraft etter 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, 'ordre 42 lagt inn');
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]:'ordre 42 lagt inn'
-- COMMIT 10301

SELECT data FROM pg_logical_slot_get_changes('shop_cdc', NULL, NULL);
-- de samme fire radene; kallet bruker dem opp, så et nytt kall gir ingen

SELECT pg_drop_replication_slot('shop_cdc');
```

Ordren og outbox-meldingen kommer ut sammen, innenfor transaksjonen som skrev dem. `pg_logical_slot_peek_changes` leser uten å flytte sporet og gir derfor de samme radene ved neste kall, mens `pg_logical_slot_get_changes` bruker dem opp. En `CREATE TABLE` ville bare gitt en tom `BEGIN` og `COMMIT`, fordi logisk dekoding ikke tar med DDL. Den siste linjen er den viktigste: et spor som blir glemt, fortsetter å holde på ressursene til serveren.

I produksjon er det Debezium som leser. Første gang konnektoren for PostgreSQL starter, tar den et konsistent øyeblikksbilde (snapshot) av tabellene den fanger endringer fra, og strømmer deretter endringer fra punktet der øyeblikksbildet ble tatt. Standardverdien for `snapshot.mode`, `initial`, tar bare øyeblikksbildet når det ikke finnes lagrede offsets, så en omstart fortsetter fra den sist registrerte posisjonen. Hver hendelse har et `before` - og et `after` -bilde av raden, en `source` -blokk, et tidsstempel og en `op` -kode: `c` for opprettelse, `u` for oppdatering, `d` for sletting, `r` for en rad lest under et øyeblikksbilde og `t` for truncate. Etter en sletting sender konnektoren i tillegg en tombstone, en hendelse med samme nøkkel og verdien null, slik at log compaction i Kafka kan fjerne de tidligere meldingene med den nøkkelen.

Begge mønstrene fra denne artikkelen finnes på selskapets egne sider. Mellom 2022 og 2024 utviklet grunnleggeren 600 ETL/ELT-pipeliner i PL/pgSQL på tvers av flere utviklings- og produksjonsmiljøer i PostgreSQL, blant annet inkrementelle lastinger med CDC og tidsstempelbaserte deltaer. Notatet Å selge den siste billetten nøyaktig én gang utformer et lynsalg rundt en outbox: hvert trinn registrerer meldingen sin i outboxen i samme transaksjon som endringen, change data capture publiserer outboxen, og fordi en melding kan komme mer enn én gang, er hver konsument idempotent, med ordren som nøkkel.

Hvor det går galt

Ingenting setter tak på hva et spor kan holde på. Et spor holder på hver bit av write-ahead-loggen som konsumenten ikke har bekreftet, og standardinnstillingene i PostgreSQL setter ingen grense for det. `max_slot_wal_keep_size` er `-1`, så spor kan beholde en ubegrenset mengde logg, og `idle_replication_slot_timeout` er `0`, noe som slår av ugyldiggjøringen av spor som står ubrukt. En konnektor som stoppet under en mislykket produksjonssetting, kan derfor fylle disken lenge etter at alle har glemt den. Hold øye med `pg_replication_slots` for spor som ikke lenger er aktive, og sett `max_slot_wal_keep_size` til det disken tåler, vel vitende om at en konsument som havner lenger bak enn det, kanskje ikke kan fortsette. Veien tilbake går gjennom

et nytt spor, og et nytt spor begynner der loggen står nå: Debezium advarer om at et spor som slettes og opprettes på nytt, i det stille kan hoppe over hendelsene i mellomtiden, så ta et nytt øyeblikksbilde etterpå.

En stille tabell holder loggen tilbake. Debezium-dokumentasjonen beskriver en mer lumsk variant. Et spor hører til én database, og konnektoren flytter bare den bekreftede posisjonen sin fremover når den ser endringer den fanger. Endres tabellene den fanger, sjelden, mens andre tabeller eller andre databaser på den samme serveren skriver mye, flytter posisjonen seg nesten ikke, og loggen bak den fortsetter å vokse. Løsningen er et hjerteslag (heartbeat): `heartbeat.interval.ms` får konnektoren til å sende heartbeat-hendelser med et fast intervall, og `heartbeat.action.query` lar den skrive jevnlig til en liten tabell i databasen den fanger endringer fra. Tabellen må være med i publikasjonen (publication) som konnektoren leser, slik at sporet flytter seg fremover også når ingenting annet skjer der.

Konsumentene går ut fra at hver endring kommer én gang. Et logisk spor sender ut hver endring én gang under normal drift, men posisjonen lagres bare ved sjekkpunkter (checkpoints), så etter et krasj kan det gå tilbake til et tidligere punkt og sende de siste endringene på nytt. PostgreSQL-dokumentasjonen overlater konsekvensene til klienten. En konsument som legger hvert beløp til en løpende sum, teller de endringene to ganger. Gjør hver konsument idempotent: ta i bruk hver endring som en upsert med primærnøkkelen til raden som nøkkel, eller registrer hvilke hendelser som allerede er behandlet, slik at en ny levering ikke endrer noe.

Oppdateringer og slettinger kommer uten de gamle verdiene. Hva en hendelse for en oppdatering eller sletting forteller om den tidligere raden, avhenger av replica identity for tabellen. Med `REPLICA IDENTITY DEFAULT` inneholder hendelsene fra Debezium høyst de tidligere verdiene av primærnøkkelen, og for en tabell uten primærnøkkel sender konnektoren ingen hendelser for oppdateringer eller slettinger i det hele tatt. `REPLICA IDENTITY FULL` legger alle tidligere kolonner i `before`, men det koster: PostgreSQL-dokumentasjonen påpeker at det øker mengden WAL, særlig når mange `UPDATE`- og `DELETE`-setninger kjøres. Gi hver tabell som fanges, en primærnøkkel, og bytt bare til `FULL` på tabellene der konsumentene trenger de gamle verdiene.

Les mer

- Debezium Features — hva loggbasert CDC fanger som polling og dobbeltskriving går glipp av, pluss øyeblikksbilder, filtre og maskering.
- Debezium connector for PostgreSQL — output-plugins, øyeblikksbilder, formatet på hendelsene, replica identity og diskplass til WAL.
- Logical Decoding Concepts — replikeringsspor, output-plugins, og hvorfor et spor som ingen trenger, bør slettes.
- Logical Decoding Examples — å opprette et spor, kikke på og bruke opp endringer og slette sporet igjen.
- Replication (server configuration) — `max_replication_slots`, `max_slot_wal_keep_size` og `idle_replication_slot_timeout`.
- Outbox Event Router — outbox-mønsteret, og hvordan Debezium sender outbox-rader videre til topics.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er change data capture?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/change-data-capture/