Engineer

Capture de données modifiées

lire les changements d'une base pour les transmettre

Ce que c’est

La capture de données modifiées, généralement abrégée CDC, consiste à lire les changements effectués dans une base de données et à les transmettre à d’autres systèmes au fur et à mesure. Chaque insertion, mise à jour et suppression devient un événement sur lequel un autre système peut agir : une copie de reporting qui reste à jour, un index de recherche ou un autre service devant savoir qu’une commande a été passée. La base où le changement a lieu reste le système de référence, et les autres systèmes la suivent.

Il existe deux grandes façons de trouver les changements. La plus simple consiste à interroger la base : toutes les quelques minutes, une requête cherche les lignes dont l’horodatage de « dernière mise à jour » est postérieur à l’heure de l’exécution précédente. L’autre consiste à lire le journal de modifications tenu par la base elle-même. PostgreSQL conserve ce journal dans son journal de préécriture, le WAL. Sa documentation en donne la règle : les modifications des fichiers de données ne sont écrites qu’une fois les enregistrements correspondants effectivement écrits sur un support de stockage permanent. Comme le journal contient déjà tous les changements dans leur ordre d’apparition, sa lecture donne un historique complet sans interroger les tables. Cette seconde approche, la CDC fondée sur le journal, est ce que désigne généralement le terme aujourd’hui.

Dans PostgreSQL, ce mécanisme s’appelle le décodage logique. La documentation le définit comme l’« extraction de toutes les modifications persistantes apportées aux tables d’une base de données » dans un format lisible sans connaître les mécanismes internes de la base. Trois éléments coopèrent. Le serveur doit écrire assez de détails dans le WAL, ce que contrôle `wal_level` : sa valeur prédéfinie, `replica`, suffit pour copier des données entre serveurs, tandis que `logical` ajoute ce dont le décodage a besoin. Un slot de réplication

représente un flux de changements d'une base, livrés à son lecteur dans l'ordre d'origine, et mémorise jusqu'où ce client a lu. Un plugin de sortie transforme les enregistrements internes du WAL dans le format voulu par le client.

La plupart des équipes n'écrivent pas elles-mêmes ce client. Debezium fournit à Apache Kafka Connect des connecteurs qui obtiennent les changements par le mécanisme CDC de chaque base source. Son connecteur PostgreSQL lit au moyen de `pgoutput`, décrit dans sa documentation comme le plugin de sortie intégré à PostgreSQL, ou de `decoderbufs`, un plugin maintenu par la communauté Debezium. Debezium peut aussi fonctionner comme bibliothèque dans une application JVM ou comme Debezium Server, qui envoie les événements de changement à des systèmes comme Amazon Kinesis, Google Cloud Pub/Sub et Apache Pulsar.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la capture de données modifiées ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des chiffres disponibles là où il faut, à la fréquence prévue, sans recopier un tableur à la fin du mois.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Un plan d'infrastructure pour 14 départements — 14 départements de recherche sur un même plan d'infrastructure

Un plan d'infrastructure pour 14 départements Contactez-nous →

Pourquoi c'est important

Le polling paraît moins coûteux, et il l'est souvent pour une petite table qui change lentement. La documentation Debezium énumère ce que la lecture du journal apporte par rapport au polling. Elle capture chaque changement, tandis qu'une requête ne voit l'état d'une ligne qu'au moment de son exécution : une ligne modifiée trois fois entre deux exécutions n'apparaît qu'une fois. Elle capture les suppressions, alors qu'une ligne supprimée manque simplement au résultat suivant. Elle n'exige aucune colonne supplémentaire dans le modèle, comme un champ « Dernière mise à jour » que chaque chemin d'écriture doit penser à renseigner. Elle peut transporter l'ancien état d'une ligne et des métadonnées comme l'ID de la transaction à l'origine du changement. Enfin, elle livre les changements avec très peu de retard, sans le coût CPU d'interrogations fréquentes : pour PostgreSQL et MySQL, Debezium situe ce retard dans l'ordre des millisecondes.

La deuxième raison est la cohérence entre une base de données et les messages reçus par les autres systèmes. Debezium cite les doubles écritures comme autre approche à laquelle il se compare : l'application enregistre une modification dans sa base, puis envoie un message à son sujet. Si le processus s'arrête entre les deux étapes, la base et le flux de messages divergent, sans qu'aucune trace n'indique lequel est erroné. La CDC élimine la seconde écriture. L'application valide une seule fois, et le message est dérivé de ce qui a été validé. Le pattern outbox va plus loin : l'application écrit le message lui-même dans une table outbox, au sein de la même transaction que le changement, et la CDC publie les lignes de cette table. Le module Outbox Event Router de Debezium présente ce fonctionnement comme évitant les incohérences entre l'état propre d'un service et les événements consommés par les autres services.

La troisième raison est l'ordre. Un slot de réplication préserve la séquence du serveur source lorsqu'il livre les changements, regroupés par transaction d'origine, comme dans l'exemple ci-dessous. Un consommateur qui les applique dans cet ordre reconstruit le même état, étape par étape. Pour un chargement ETL, cela rend le chargement incrémental fidèle aux changements réels : un traitement nocturne peut prendre exactement les changements depuis sa dernière exécution, suppressions comprises, au lieu de recharger des tables entières ou de se fier à une colonne d'horodatage qu'un chemin de code a oublié d'actualiser.

À quoi cela ressemble en pratique

Le plus petit exemple opérationnel utilise uniquement PostgreSQL. Il exige `wal_level = logical` dans la configuration du serveur, réglage qui ne prend effet qu'après un redémarrage, et un slot de réplication libre ; `max_replication_slots` vaut 10 sans configuration contraire. La session ci-dessous s'exécute en superutilisateur, comme l'exemple de la documentation, et emploie son plugin de sortie `test_decoding`, qui affiche chaque changement sous forme d'une ligne de texte.

```
-- 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');
```

La commande client et son message outbox sortent ensemble, dans la transaction qui les a écrits. `pg_logical_slot_peek_changes` lit sans avancer le slot et renvoie donc les mêmes lignes à l'appel suivant, tandis que `pg_logical_slot_get_changes` les consomme. Un `CREATE TABLE` ne produirait qu'un `BEGIN` et un `COMMIT` vides, car le décodage logique ne transporte pas le DDL. La dernière ligne est la plus importante : un slot abandonné continue à retenir des ressources du serveur.

En production, Debezium joue le rôle de lecteur. Au premier démarrage, son connecteur PostgreSQL prend un instantané cohérent des tables capturées, puis diffuse les changements depuis le point où cet instantané a été pris. La valeur prédéfinie de `snapshot.mode`, `initial`, ne prend cet instantané qu'en l'absence d'offsets enregistrés : un redémarrage reprend donc à la dernière position mémorisée. Chaque événement porte les images `before` et `after` de la ligne, un bloc `source`, un horodatage et un code `op` : `c` pour une création, `u` pour une mise à jour, `d` pour une suppression, `r` pour une ligne lue pendant un instantané et `t` pour une troncature. Après une suppression, le connecteur émet aussi un événement tombstone, de même clé avec une valeur nulle, afin que la compaction du journal Kafka puisse retirer les messages antérieurs de cette clé.

Les deux patterns de cet article apparaissent dans les propres pages de l'entreprise. Entre 2022 et 2024, la fondatrice a conçu 600 pipelines ETL/ELT en PL/pgSQL dans plusieurs environnements PostgreSQL de développement et de production, dont des chargements incrémentaux utilisant la CDC et des deltas fondés sur les horodatages. La note Vendre le dernier billet exactement une fois conçoit une vente éclair autour de l'outbox : chaque étape y inscrit son message dans la même transaction que le changement, la capture de données modifiées publie l'outbox et, comme un message peut arriver plusieurs fois, chaque consommateur est idempotent, avec la commande comme clé.

Où les problèmes surviennent

Rien ne plafonne ce qu'un slot peut conserver. Un slot retient chaque portion du WAL que son consommateur n'a pas confirmée, et les valeurs prédéfinies de PostgreSQL ne fixent aucun plafond. `max_slot_wal_keep_size` vaut `-1`, ce qui permet aux slots de conserver une quantité illimitée de WAL, et `idle_replication_slot_timeout` vaut `0`, ce qui désactive l'invalidation des slots inutilisés. Un connecteur arrêté pendant un déploiement échoué peut donc remplir le disque longtemps après que tout le monde l'a oublié. Surveillez les slots devenus inactifs dans `pg_replication_slots` et fixez `max_slot_wal_keep_size` selon ce que le disque peut accueillir, en sachant qu'un consommateur dont le retard dépasse la quantité de journal conservée peut ne plus pouvoir reprendre. La récupération exige alors un nouveau slot, qui commence au présent : Debezium avertit qu'un slot supprimé puis recréé peut silencieusement sauter les événements intermédiaires. Prenez donc un nouvel instantané ensuite.

Une table peu active retient le journal. La documentation Debezium décrit un cas plus subtil. Un slot appartient à une seule base, et le connecteur n'avance sa position confirmée que lorsqu'il voit des changements qu'il capture. Si les tables capturées changent rarement alors que d'autres tables, ou d'autres bases du même serveur, reçoivent beaucoup d'écritures, la position avance à peine et le WAL retenu continue à croître. La solution est un battement de cœur : `heartbeat.interval.ms` fait émettre au connecteur des événements de battement de cœur à un intervalle défini, et `heartbeat.action.query` lui permet d'écrire régulièrement dans une petite table de la base capturée. Celle-ci doit appartenir à la publication lue par le connecteur, pour que le slot avance même si rien d'autre ne s'y produit.

Les consommateurs supposent que chaque changement n'arrive qu'une fois. Un slot logique émet chaque changement une fois en fonctionnement normal, mais sa position n'est enregistrée qu'aux points de contrôle. Après un plantage, il peut donc revenir à un point antérieur et renvoyer les changements récents. La documentation PostgreSQL laisse au client la gestion des conséquences. Un consommateur qui ajoute chaque montant à un cumul compterait ces changements deux fois. Rendez chaque consommateur idempotent : appliquez chaque changement comme un upsert utilisant la clé primaire de la ligne, ou enregistrez les ID des événements déjà traités, pour qu'une seconde livraison ne change rien.

Les mises à jour et suppressions arrivent sans les anciennes valeurs. Les informations d'un événement de mise à jour ou de suppression sur l'ancienne ligne dépendent de l'identité de réplication de la table. Avec `REPLICA IDENTITY DEFAULT`, les événements Debezium portent au plus les anciennes valeurs de clé primaire, et le connecteur ne produit aucun événement de mise à jour ni de suppression si la table n'a pas de clé primaire. `REPLICA IDENTITY FULL` place les anciennes valeurs de toutes les colonnes dans `before`, moyennant un coût : la documentation PostgreSQL indique que cela augmente le volume du WAL, surtout avec de nombreuses instructions `UPDATE` et `DELETE`. Donnez une clé primaire à chaque table capturée et ne passez à `FULL` que les tables dont les consommateurs ont besoin des anciennes valeurs.

Pour aller plus loin

- Fonctionnalités de Debezium — ce que la CDC fondée sur le journal capture et que le polling et les doubles écritures manquent, ainsi que les instantanés, filtres et masquages.
- Connecteur Debezium pour PostgreSQL — plugins de sortie, instantanés, format des événements de changement, identité de réplication et espace disque du WAL.
- Concepts du décodage logique — slots de réplication, plugins de sortie et raisons de supprimer un slot devenu inutile.
- Exemples de décodage logique — créer un slot, consulter et consommer les changements, puis supprimer le slot.
- Réplication : configuration du serveur — `max_replication_slots`, `max_slot_wal_keep_size` et `idle_replication_slot_timeout`.
- Outbox Event Router — le pattern outbox et le routage des lignes outbox vers des topics par Debezium.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la capture de données modifiées ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/change-data-capture/