

M W
S

DEBEZIUM И POSTGRESQL ПОСЛЕ HAPPY-PATH: КАКИЕ ПРОБЛЕМЫ ЖДУТ В ПРОДЕ И КАК ИХ РЕШАТЬ



Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

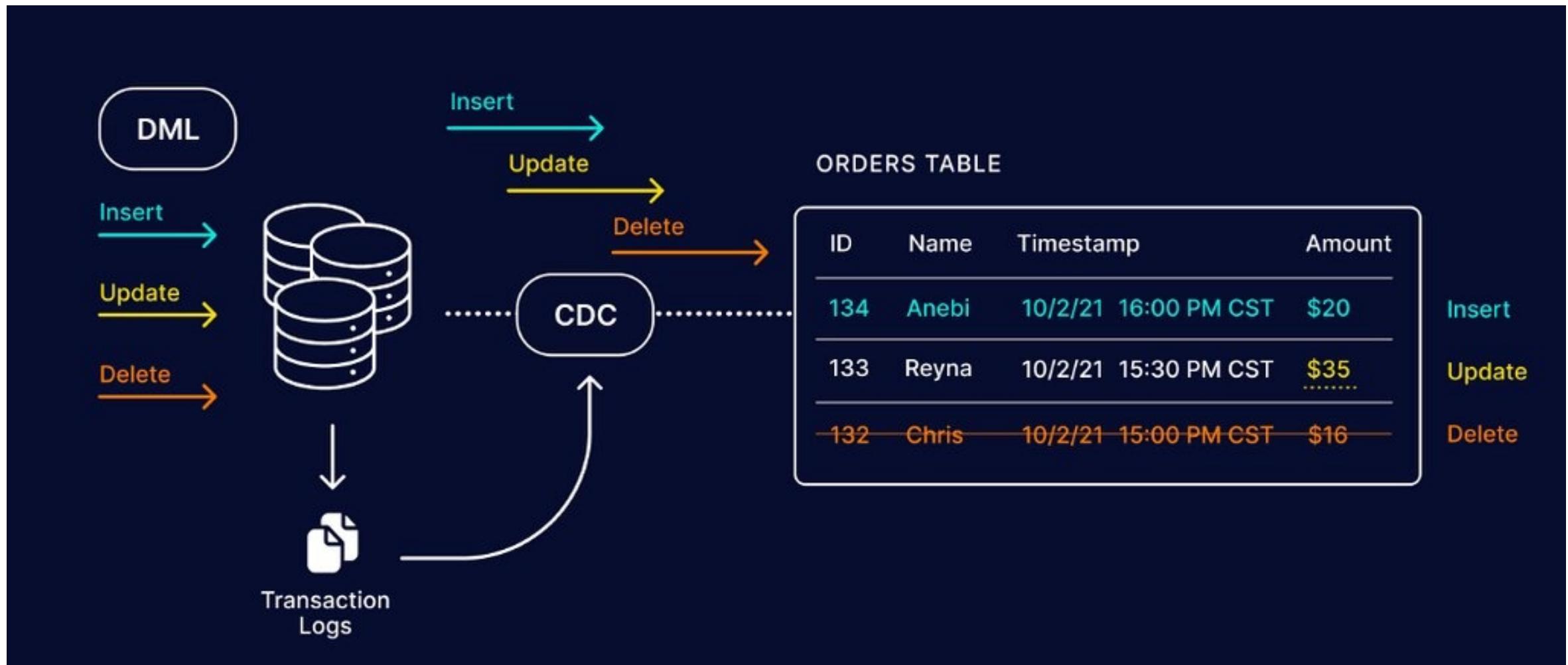
05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

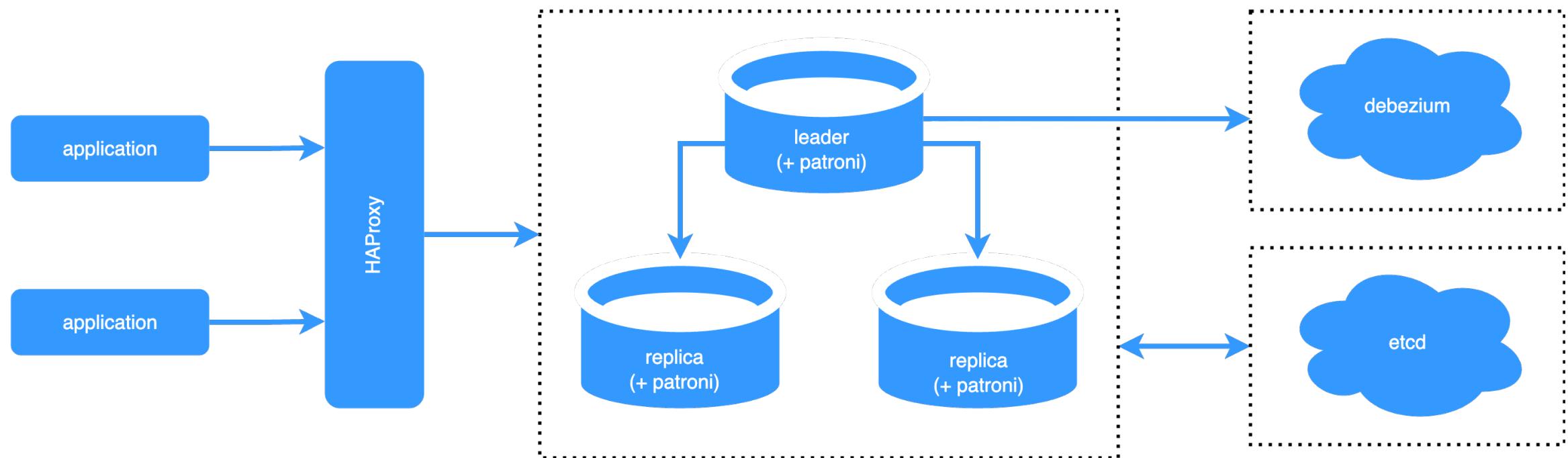
Что такое “CDC”



Подробнее про CDC: https://en.wikipedia.org/wiki/Change_data_capture/

Источник: <https://blog.nashtechglobal.com/streaming-data-with-log-based-cdc-using-debezium-kafka-and-ksql/>

Типовая схема



Как попробовать самостоятельно

Дополнительная информация

1. Все примеры выполнены с использованием СУБД PostgreSQL на версиях 15, 16 и 17. Если версия не указана, то по умолчанию подразумевается использование 17
2. Для проведения тестов, максимально приближенных к реальности, использовался кластер postgres, поднятый с помощью postgres-zalando-operator в k8s
<https://github.com/zalando/postgres-operator>
3. Все примеры реализованы в виде runtime-«обертки» вокруг Debezium Engine, но они также справедливы и для kafka-connect коннектора, Debezium Server и др.

Примеры кода



Начальные условия репликации

```
CREATE TABLE debezium_offsets
(
    id              TEXT PRIMARY KEY,
    offset_key      TEXT,
    offset_val      TEXT,
    record_insert_ts TIMESTAMP NOT NULL,
    record_insert_seq INTEGER NOT NULL
);

CREATE TABLE data
(
    id    uuid PRIMARY KEY,
    value TEXT
);

SELECT pg_create_logical_replication_slot('debezium_slot', 'pgoutput');

CREATE PUBLICATION debezium_publication;
ALTER PUBLICATION debezium_publication ADD TABLE public.data;
```

Начальные условия репликации

```
CREATE TABLE debezium_offsets
(
    id              TEXT PRIMARY KEY,
    offset_key      TEXT,
    offset_val      TEXT,
    record_insert_ts TIMESTAMP NOT NULL,
    record_insert_seq INTEGER NOT NULL
);

CREATE TABLE data
(
    id    uuid PRIMARY KEY,
    value TEXT
);

SELECT pg_create_logical_replication_slot('debezium_slot', 'pgoutput');

CREATE PUBLICATION debezium_publication;
ALTER PUBLICATION debezium_publication ADD TABLE public.data;
```

Начальные условия репликации

```
CREATE TABLE debezium_offsets
(
    id              TEXT PRIMARY KEY,
    offset_key      TEXT,
    offset_val      TEXT,
    record_insert_ts TIMESTAMP NOT NULL,
    record_insert_seq INTEGER NOT NULL
);

CREATE TABLE data
(
    id    uuid PRIMARY KEY,
    value TEXT
);

SELECT pg_create_logical_replication_slot('debezium_slot', 'pgoutput');

CREATE PUBLICATION debezium_publication;
ALTER PUBLICATION debezium_publication ADD TABLE public.data;
```

Начальные условия репликации

```
CREATE TABLE debezium_offsets
(
    id              TEXT PRIMARY KEY,
    offset_key      TEXT,
    offset_val      TEXT,
    record_insert_ts TIMESTAMP NOT NULL,
    record_insert_seq INTEGER NOT NULL
);

CREATE TABLE data
(
    id    uuid PRIMARY KEY,
    value TEXT
);

SELECT pg_create_logical_replication_slot('debezium_slot', 'pgoutput');

CREATE PUBLICATION debezium_publication;
ALTER PUBLICATION debezium_publication ADD TABLE public.data;
```

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Начальный snapshot таблицы

Задача

Нужно получить начальное состояние таблицы перед стартом репликации

Начальный snapshot таблицы

Задача

Нужно получить начальное состояние таблицы перед стартом репликации

Решение

- `snapshot.mode=INITIAL`

Изменение списка реплицируемых таблиц

Задача

Нужно изменить список реплицируемых таблиц, при этом получив начальное состояние новой таблицы. При `snapshot.mode=INITIAL` снепшоты не запускаются для новых таблиц

Изменение списка реплицируемых таблиц

Задача

Нужно изменить список реплицируемых таблиц, при этом получив начальное состояние новой таблицы. При `snapshot.mode=INITIAL` снепшоты не запускаются для новых таблиц

Решения

1. Создать новый слот и публикацию рядом
2. `snapshot.mode=INITIAL_ALWAYS & ALTER PUBLICATION {name} ADD TABLE {table}; & рестарт`

Как избежать бесконечных снепшотов?

Проблема

При `snapshot.mode=INITIAL_ALWAYS`
каждый рестарт триggerит повторный
снепшот по всем таблицам

Как избежать бесконечных снепшотов?

Проблема

При `snapshot.mode=INITIAL_ALWAYS`
каждый рестарт триggerит повторный
снепшот по всем таблицам

Решение

- Запускать снепшоты самостоятельно по
нужному подмножеству таблиц

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Ручное управление снепшотами

Задача

Нужно научиться управлять запуском снепшотов самостоятельно, чтобы иметь возможность при добавлении новых таблиц получать их начальное состояние

Ручное управление снепшотами

Задача

Нужно научиться управлять запуском снепшотов самостоятельно, чтобы иметь возможность при добавлении новых таблиц получать их начальное состояние

Решение

- snapshot.mode=INITIAL или snapshot.mode=NO_DATA
- Использовать сигналы (kafka, jmx, source, custom)

Ручное управление снепшотами



```
CREATE TABLE signals
(
    id TEXT NOT NULL PRIMARY KEY,
    type TEXT NOT NULL,
    data TEXT
);
```

```
INSERT INTO signals(id, type, data)
VALUES (gen_random_uuid(), 'execute-snapshot', '{"type": "BLOCKING", "data-collections": ["public.data"]}')
```

Ручное управление снэпшотами



```
CREATE TABLE signals
(
    id TEXT NOT NULL PRIMARY KEY,
    type TEXT NOT NULL,
    data TEXT
);
```

```
INSERT INTO signals(id, type, data)
VALUES (gen_random_uuid(), 'execute-snapshot', '{"type": "BLOCKING", "data-collections": ["public.data"]}')
```

После снепшота не запускается стриминг

Проблема

После выполнения снепшота стриминг данных не продолжается

После снепшота не запускается стриминг

Проблема

После выполнения снепшота стриминг данных не продолжается

Решения

- Обновиться до >3.1.2.Final (Баг: [DBZ-9055](#))

Длинные транзакции

Проблема

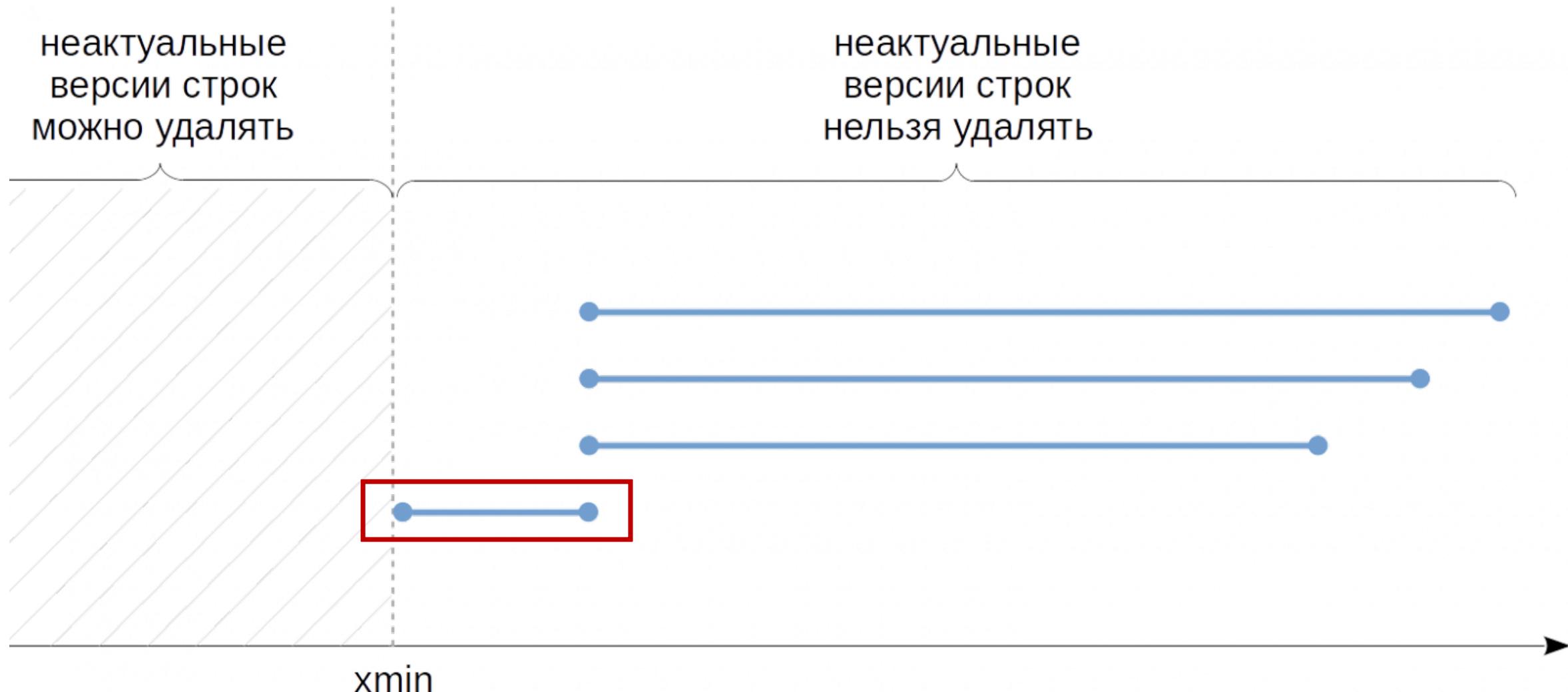
BLOCKING снепшот для больших таблиц держит транзакцию слишком долго, что оказывает негативный эффект на весь кластер:

- Растет WAL
- Растет общее занимаемое место
- Страдают запросы к другим таблицам

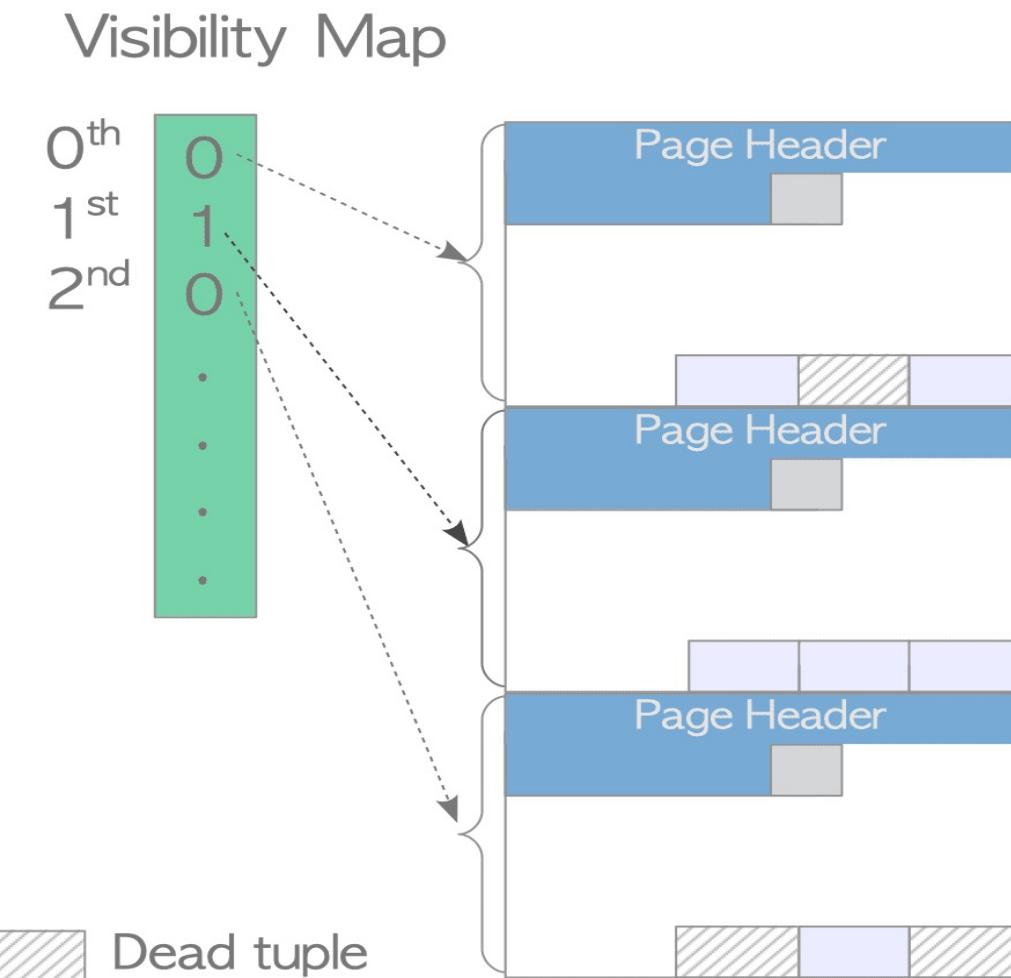
Длинные транзакции и горизонт событий



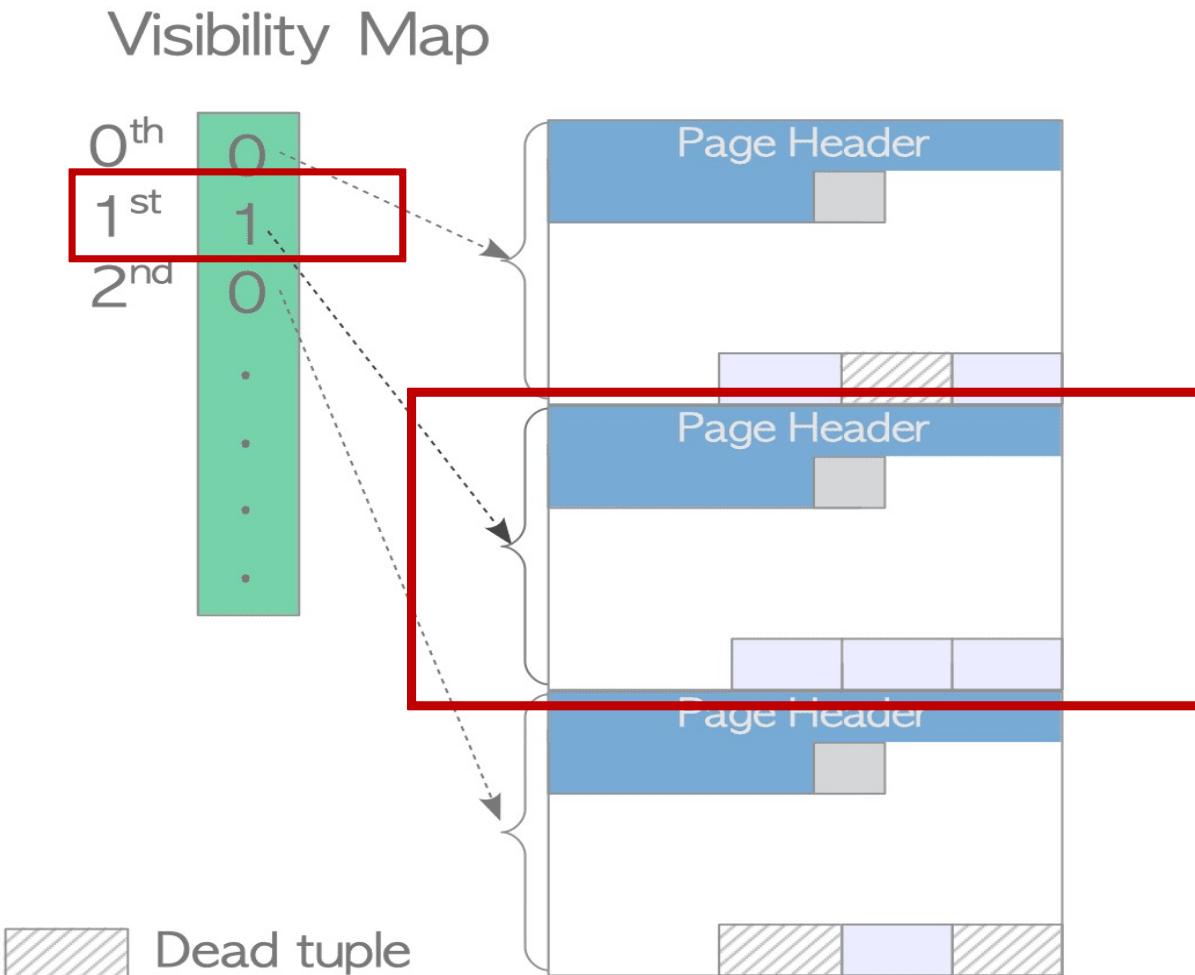
Длинные транзакции и горизонт событий



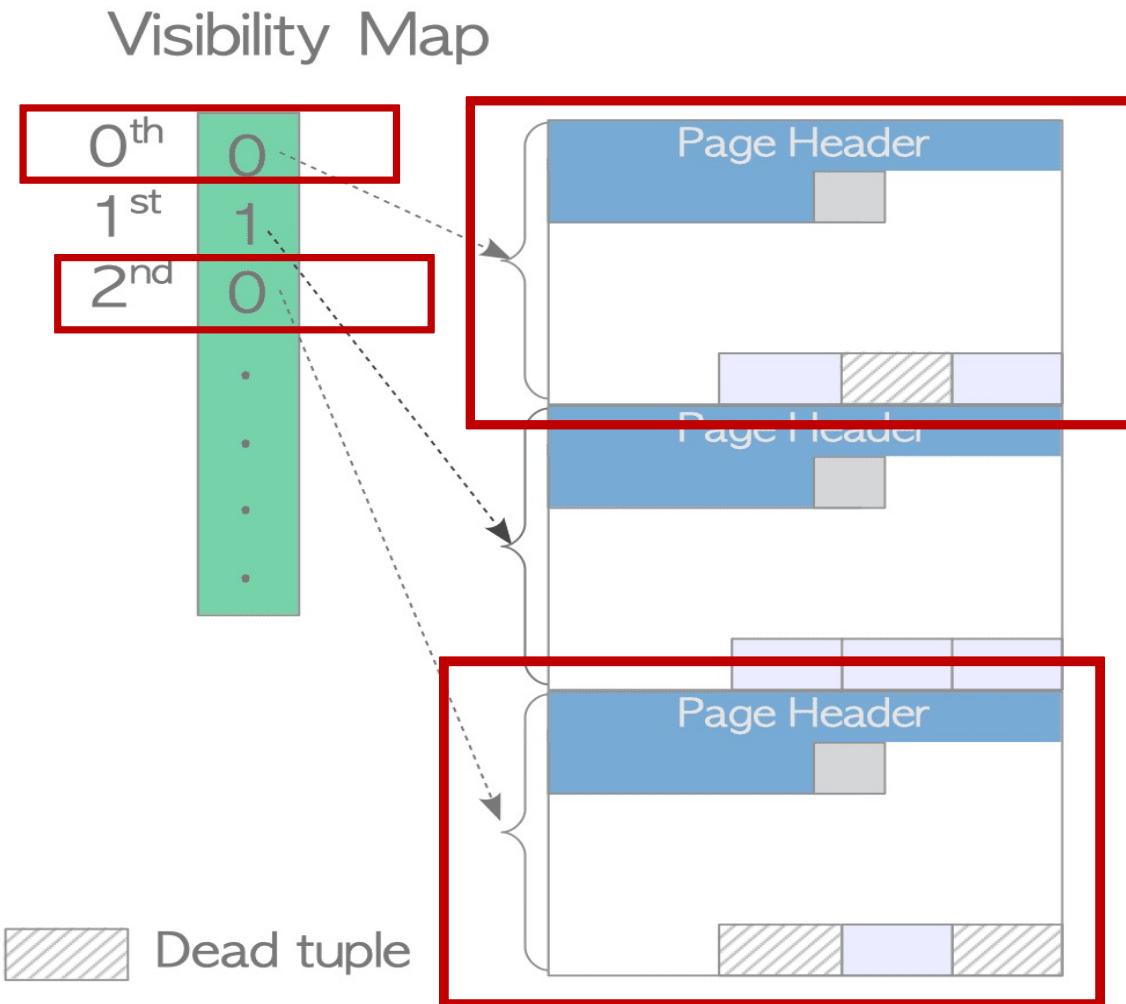
Длинные транзакции и горизонт событий



Длинные транзакции и горизонт событий



Длинные транзакции и горизонт событий



Длинные транзакции

Проблема

BLOCKING снепшот для больших таблиц держит транзакцию слишком долго, что оказывает негативный эффект на весь кластер:

- Растет WAL
- Растет общее занимаемое место
- Страдают запросы к другим таблицам

Решение

- Использовать INCREMENTAL снепшот
- Работает только с типом source

Incremental ad-hoc snapshot



```
INSERT INTO signals(id, type, data)
VALUES (
    gen_random_uuid(),
    'execute-snapshot',
    {"type": "INCREMENTAL", "data-collections": ["public.data"]}
);
```

Incremental ad-hoc snapshot

table

c1 (pk)	c2	c3	c4
1
3
4
8
19
20
...

chunk 1

chunk 2

Размер чанка:
`incremental.snapshot.chunk.size`

Incremental ad-hoc snapshot без PK

Проблема

INCREMENTAL снепшот работает только для таблиц с PK

Incremental ad-hoc snapshot без PK

Проблема

INCREMENTAL снепшот работает только для таблиц с PK

Решение

- Начиная с версии debezium 2.2 есть возможность в сигналах использовать surrogate.key

Incremental ad-hoc snapshot без PK



```
INSERT INTO signals(id, type, data)
VALUES (
    gen_random_uuid(),
    'execute-snapshot',
    '{"type": "INCREMENTAL", "data-collections": ["public.data"], "surrogate-key": "field"}'
);
```

Долгий процесс снятия снепшота

Проблема

Процесс снятия снепшота для больших таблиц занимает много времени (часы / десятки часов)

Долгий процесс снятия снепшота

Проблема

Процесс снятия снепшота для больших таблиц занимает много времени (часы / десятки часов)

Решение

- Реализовать кастомную логику снятия снепшота
- Использовать другие готовые решения (e.g. Apache Flink CDC)

Incremental снепшот падает с ошибкой

Проблема

При запуске инкрементального снепшота
процесс снепшота завершается с ошибкой
об отсутствии схемы таблицы

Incremental снепшот падает с ошибкой

Проблема

При запуске инкрементального снепшота процесс снепшота завершается с ошибкой об отсутствии схемы таблицы

Решения

- Предварительно стриггерить любое изменение по таблице
- Обновить debezium до версии >3.1.2.Final
- Запустить BLOCKING снепшот с фильтром и LIMIT 0

Как снизить нагрузку от снепшотов

Проблема

Владельцы БД категорически против генерации любой дополнительной нагрузки от снепшотов

Как снизить нагрузку от снепшотов

Проблема

Владельцы БД категорически против генерации любой дополнительной нагрузки от снепшотов

Решение

- Начиная с pg16 и debezium 2.5+, можно настроить логическую репликацию на репликах

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Репликация редко обновляемых таблиц

Проблема

Реплицируется редко обновляемая таблица(ы), например, справочник, по которой апдейты приходят редко или не приходят вообще. В БД при этом параллельно существуют другие таблицы активной записью

Репликация редко обновляемых таблиц

Проблема

Реплицируется редко обновляемая таблица(ы), например, справочник, по которой апдейты приходят редко или не приходят вообще. В БД при этом параллельно существуют другие таблицы активной записью

Решение

- Использовать heartbeat таблицу
- Важно добавить ее в список реплицируемых таблиц

Репликация редко обновляемых таблиц

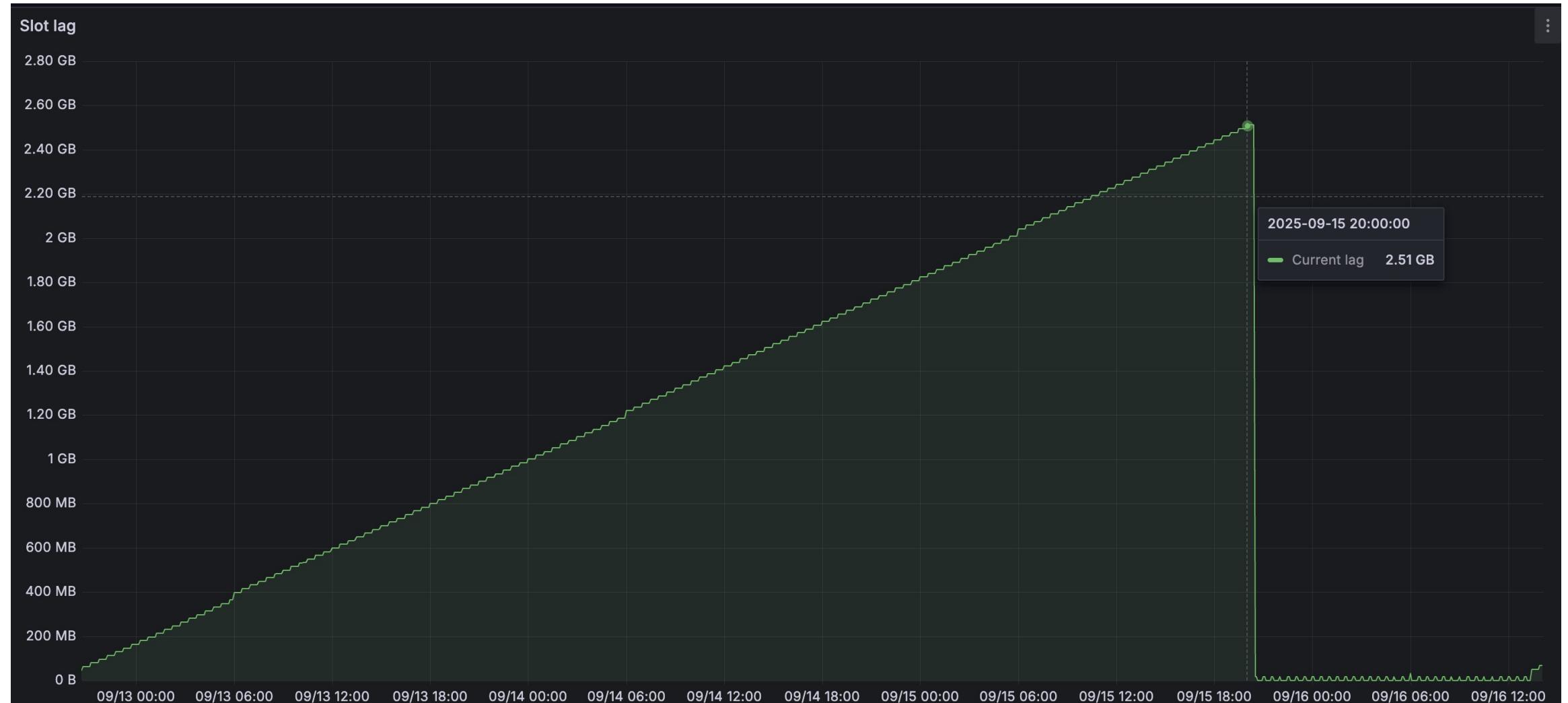


График лага слота ДО и ПОСЛЕ включения heartbeat таблицы

Репликация редко обновляемых таблиц



```
CREATE TABLE heartbeat
(
    single_row bool PRIMARY KEY DEFAULT TRUE,
    "timestamp" TIMESTAMP NOT NULL,
    CONSTRAINT single_row_check CHECK (single_row)
);
```

Как ограничить допустимый лаг слота

Проблема

Репликация долго не работала и WAL занял все свободное место. БД перешла в RO

Как ограничить допустимый лаг слота

Проблема

Репликация долго не работала и WAL занял все свободное место. БД перешла в RO

Решение

- Настроить параметр **max_slot_wal_keep_size**, чтобы ограничить размер WAL'а, который удерживается слотом

Как ограничить допустимый лаг слота



График лага слота и текущее установленное значение допустимого лага

Как обезопасить БД от RO?

Задача

Нужно обеспечить гарантии, что БД НЕ перейдет в RO (read-only)

Как обезопасить БД от RO?

Задача

Нужно обеспечить гарантии, что БД НЕ перейдет в RO (read-only)

Решение

- Мониторинг статуса слота
- Мониторинг лага слота
- * Мониторинг общего объема занимаемого места с учетом резерва `max_wal_keep_slot_size`

Как обезопасить БД от RO?



График общего размера БД, включая резерв под слот

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Дедупликация данных

Задача

Нужно обеспечить дедупликацию данных,
чтобы приблизиться к exactly-once

Дедупликация данных

Задача

Нужно обеспечить дедупликацию данных, чтобы приблизиться к exactly-once

Решение

- Если нет бизнес ключа идемпотентности, то можно использовать LSN эвента
- При INCREMENTAL снепшоте LSN не будет

Восстановление данных

Задача

Из-за сбоя потеряли часть UPDATE/INSERT операций. Целиком делать снепшот таблицы -- дорого

Восстановление данных

Задача

Из-за сбоя потеряли часть UPDATE/INSERT операций. Целиком делать снепшот таблицы -- дорого

Решение

- Использовать ad-hoc снепшот с фильтром

Восстановление данных

```
INSERT INTO signals(id, type, data)
VALUES (
    gen_random_uuid(),
    'execute-snapshot',
    '{"type": "BLOCKING", ' ||
    ' "data-collections": ["public.data"], ' ||
    '"additional-conditions": ' ||
        '[{"data-collection": "public.data", "filter": "SELECT * FROM public.data WHERE timestamp >= X AND timestamp < Y"}]' ||
);
;

INSERT INTO signals(id, type, data)
VALUES (
    gen_random_uuid(),
    'execute-snapshot',
    '{"type": "INCREMENTAL", ' ||
    ' "data-collections": ["public.data"], ' ||
    '"additional-conditions": ' ||
        '[{"data-collection": "public.data", "filter": "timestamp >= X AND timestamp < Y"}]' ||
);
;
```

Потерянный DELETE

Задача

Из-за сбоя потеряли эвенты с DELETE.
Снепшот с фильтром не решает проблему

Потерянный DELETE

Задача

Из-за сбоя потеряли эвенты с DELETE.
Снепшот с фильтром не решает проблему

Решения

1. Повторный полный снепшот с предварительной очисткой таргета
2. Повторный полный снепшот с инкрементов epoch_id и БЕЗ очистки таргета

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

REPLICA IDENTITY FULL

Задачи

- Нужно реплицировать таблицу(ы) без РК
- С TOAST-колонками
- Иметь возможность получить DIFF между текущим и предыдущим состоянием строки

[TOAST](#) – метод хранения сверхбольших атрибутов таблицы

REPLICA IDENTITY FULL

Задачи

- Нужно реплицировать таблицу(ы) без PK
- С TOAST-колонками
- Иметь возможность получить DIFF между текущим и предыдущим состоянием строки

Решение

- Для таблиц без PK: Использовать REPLICA IDENTITY FULL
- Для таблиц с TOAST полями: REPLICA IDENTITY FULL или сделать TOAST-поля частью PK
- Для DIFF также использовать REPLICA IDENTITY FULL
- **REPLICA IDENTITY FULL приводит к доп.нагрузке на WAL**

REPLICA IDENTITY FULL



```
{"before": null, "after": {"id": 1, "field": "value1", ...}, "source": {...}, ...}
```

```
{"before": {"id": 1, "field1": "value1", ...}, "after": {"id": 1, "field1": "value2", ...}, "source": {...}, ...}
```

Репликацияパーティционированных таблиц

Задача

Необходимо реплицировать
партиционированную таблицу

Репликацияパーティционированных таблиц

Задача

Необходимо реплицировать
партиционированную таблицу

Решение

- При создании публикации выставить флаг **publish_via_partition_root**

Репликацияパーティционированных таблиц



```
// publish_via_partition_root=false
{"before": null, "after": {...}, "source": {..., "schema": "public", "table": "data_partitioned_p1", ...}, ...}

// publish_via_partition_root=true
{"before": null, "after": {...}, "source": {..., "schema": "public", "table": "data_partitioned", ...}, ...}
```

Репликация по транзакциям

Задача

Нужно реплицировать «строго» по транзакциям

Репликация по транзакциям

Задача

Нужно реплицировать «строго» по транзакциям

Решение

- provide.transaction.metadata=true

Репликацияパーティционированных таблиц



```
{"status": "BEGIN", "id": "799:22677648", "event_count": null, "data_collections": null, ...}  
[..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 1, "data_collection_order": 1}, ...]  
{"before": null, "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 2, "data_collection_order": 2}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 3, "data_collection_order": 3}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 4, "data_collection_order": 4}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 5, "data_collection_order": 5}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 6, "data_collection_order": 6}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 7, "data_collection_order": 7}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 8, "data_collection_order": 8}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 9, "data_collection_order": 9}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 10, "data_collection_order": 10}, ...}  
{"status": "END", "id": "799:22680192", "event_count": 10, "data_collections": [{"data_collection": "public.data", "event_count": 10}], ...}
```

Репликацияパーティционированных таблиц



```
{"status": "BEGIN", "id": "799:22677648", "event_count": null, "data_collections": null, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 1, "data_collection_order": 1}, ...}  
{"before": null, "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 2, "data_collection_order": 2}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 3, "data_collection_order": 3}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 4, "data_collection_order": 4}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 5, "data_collection_order": 5}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 6, "data_collection_order": 6}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 7, "data_collection_order": 7}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 8, "data_collection_order": 8}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 9, "data_collection_order": 9}, ...}  
{..., "after": {...}, "source": {..., "txId": 799, ...}, "transaction": {"id": "799:lsn#", "total_order": 10, "data_collection_order": 10}, ...}  
{"status": "END", "id": "799:22680192", "event_count": 10, "data_collections": [{"data_collection": "public.data", "event_count": 10}], ...}
```

Репликация hypertable (timescaledb)

Задача

Нужно реплицировать hypertable

Репликация hypertable (timescaledb)

Задача

Нужно реплицировать hypertable

Решение

- Заранее настроить публикацию
- Использовать трансформ для TimescalDB
- *Начальный снепшот сделать самостоятельно

Репликация hypertable (timescaledb)

```
CREATE PUBLICATION pub_name FOR TABLES IN SCHEMA _timescaledb_internal;
```

```
transforms=timescaledb
transforms.timescaledb.type=io.debezium.connector.postgresql.transforms.timescaledb.TimescaleDb
transforms.timescaledb.database.hostname=localhost
transforms.timescaledb.database.port=5432
transforms.timescaledb.database.user=postgres
transforms.timescaledb.database.password=postgres
transforms.timescaledb.database.dbname=postgres
```

Оглавление

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных таблиц

07 Эволюция схем и репликация шардов

08 Выводы

Эволюция схем

Задача

Используется выходной формат данных avro в связке с schema registry. При некоторых* изменениях структуры таблицы репликация останавливается

Эволюция схем

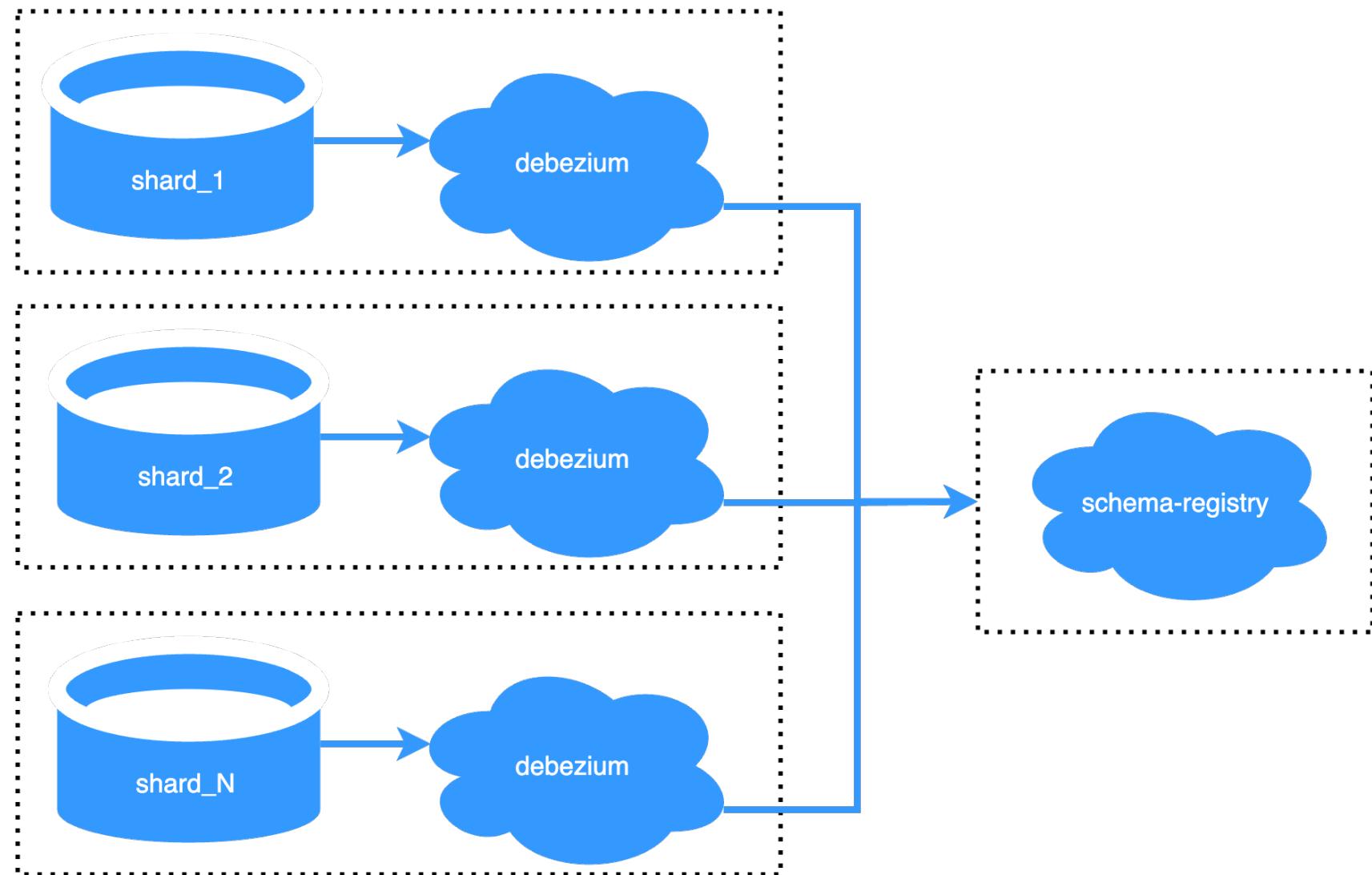
Задача

Используется выходной формат данных avro в связке с schema registry. При некоторых* изменениях структуры таблицы репликация останавливается

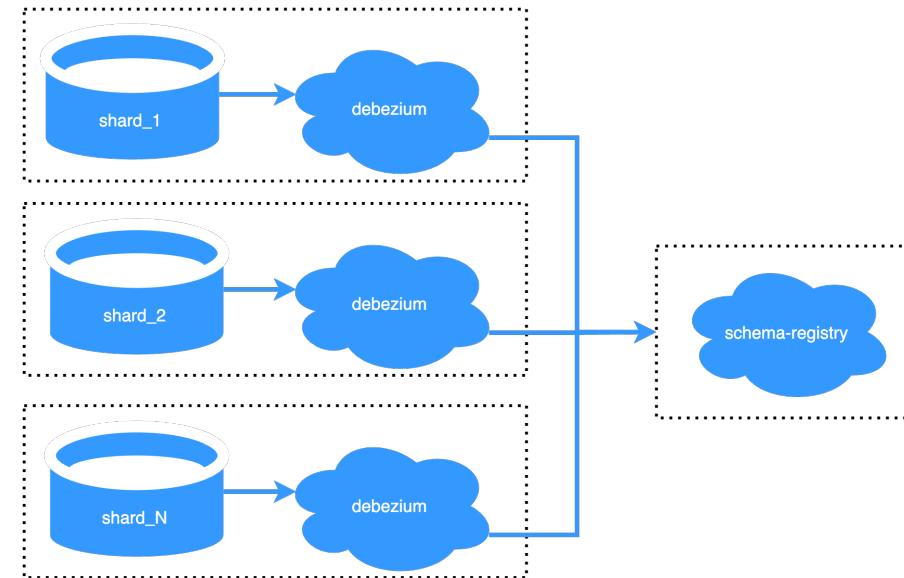
Решения

1. Запретить выполнять несовместимые изменения (набор таких изменений зависит от уровня поддерживаемой совместимости)
2. Использовать compatibility=NONE и разрешать конфликты за пределами источника

Репликация шардов



Репликация шардов



```
topic-prefix=shard_X
transforms=renameTableTopic
transforms.renameTableTopic.type=org.apache.kafka.connect.transforms.RegexRouter
transforms.renameTableTopic.regex=shard_X.public.data
transforms.renameTableTopic.replacement=public_data_table
```

Репликация шардов

```
{  
  "type": "record",  
  "name": "Envelope",  
  "namespace": "shard_1.public.data",  
  "fields": [  
    {  
      "name": "before",  
      "type": [  
        "null",  
        {  
          "type": "record",  
          "name": "Value",  
          "fields": []  
        }  
      ]  
    },  
    {  
      "name": "after",  
      "type": [  
        "null",  
        "Value"  
      ]  
    }  
  ]  
}
```

```
{  
  "type": "record",  
  "name": "Envelope",  
  "namespace": "shard_2.public.data",  
  "fields": [  
    {  
      "name": "before",  
      "type": [  
        "null",  
        {  
          "type": "record",  
          "name": "Value",  
          "fields": []  
        }  
      ]  
    },  
    {  
      "name": "after",  
      "type": [  
        "null",  
        "Value"  
      ]  
    }  
  ]  
}
```

Репликация шардов

```
{  
    "type": "record",  
    "name": "Envelope",  
    "namespace": "shard_1.public.data",  
    "fields": [  
        {  
            "name": "before",  
            "type": [  
                "null",  
                {  
                    "type": "record",  
                    "name": "Value",  
                    "fields": []  
                }  
            ]  
        },  
        {  
            "name": "after",  
            "type": [  
                "null",  
                "Value"  
            ]  
        }  
    ]  
}
```

```
{  
    "type": "record",  
    "name": "Envelope",  
    "namespace": "shard_2.public.data",  
    "fields": [  
        {  
            "name": "before",  
            "type": [  
                "null",  
                {  
                    "type": "record",  
                    "name": "Value",  
                    "fields": []  
                }  
            ]  
        },  
        {  
            "name": "after",  
            "type": [  
                "null",  
                "Value"  
            ]  
        }  
    ]  
}
```

Репликация шардов



```
transforms=renameTableTopic, renameNamespace
transforms.renameTableTopic.type=org.apache.kafka.connect.transforms.RegexRouter
transforms.renameTableTopic.regex=shard_X.public.data
transforms.renameTableTopic.replacement=public data table
transforms.renameNamespace.type=org.apache.kafka.connect.transforms.SetSchemaMetadata$Value
transforms.renameNamespace.schema.name=shards.envelope
```

Репликация шардов



```
{  
  "type": "record", "name": "Envelope", "namespace": "shards",  
  "fields": [  
    {  
      "name": "before",  
      "type": [  
        "null",  
        {  
          "type": "record", "name": "Value", "namespace": "shard_X.public.data",  
          "fields": []  
        }  
      ]  
    },  
    {  
      "name": "after",  
      "type": ["null", "shard_X.public.data.Value"]  
    }  
  ]  
}
```

Репликация шардов

```
{  
  "type": "record", "name": "Envelope", "namespace": "shards",  
  "fields": [  
    {  
      "name": "before",  
      "type": [  
        "null",  
        {  
          "type": "record", "name": "Value", "namespace": "shard_X.public.data",  
          "fields": []  
        }  
      ]  
    },  
    {  
      "name": "after",  
      "type": ["null", "shard_X.public.data.Value"]  
    }  
  ]  
}
```

Репликация шардов



```
transforms=renameTableTopic,renameNamespace, renameInnerNamespace
transforms.renameTableTopic.type=org.apache.kafka.connect.transforms.RegexRouter
transforms.renameTableTopic.regex=shard_X.public.data
transforms.renameTableTopic.replacement=public_data_table
transforms.renameNamespace.type=org.apache.kafka.connect.transforms.SetSchemaMetadata$Value
transforms.renameNamespace.schema.name=shards.envelope
transforms.renameInnerNamespace.type=smartdata.postgres.debezium.transforms.SetBeforeAndAfter
transforms.renameInnerNamespace.name=shards_inner.value
```

Репликация шардов



```
{  
    "type": "record", "name": "envelope", "namespace": "shards",  
    "fields": [  
        {  
            "name": "before",  
            "type": [  
                "null",  
                {  
                    "type": "record", "name": "value", "namespace": "shards_inner",  
                    "fields": []  
                }  
            ]  
        },  
        {  
            "name": "after",  
            "type": ["null", "shards_inner.value"]  
        }  
    ]  
}
```

ОГЛАВЛЕНИЕ

01 Introduction

02 Initial snapshots

03 Ad-hoc snapshots

04 Надежность и мониторинг

05 Качество данных

06 Репликация нестандартных
таблиц

07 Эволюция схем и репликация
шардов

08 Выводы

ВЫВОДЫ

01

Главное в репликации – мониторинг.

Достаточно иметь две метрики:

- Лаг слота
- Общий размер БД с учетом доступного места и резерва под WAL

02

Для больших таблиц стандартный механизм снепшота debezium'a НЕ лучший вариант и стоит рассмотреть альтернативы

03

Обновление как часть культуры.

Debezium достаточно зрелый инструмент, но как и в любом инструменте баги неизбежны. Каждая новая версия устраняет часть старых багов-~~и создает новые~~

04

Репликация таблиц с «подвижной» схемой может сломаться в неожиданных местах. Обязательно проверяйте поведение репликации в разных сценариях эволюции схемы. Не ограничивайтесь happy-path'ом

**Спасибо за
внимание**

tg / linkedin / github: nryanov

