Change Data Capture入門: DebeziumでPostgreSQLのデータを解放しよう！

株式会社SRA OSS

会社紹介

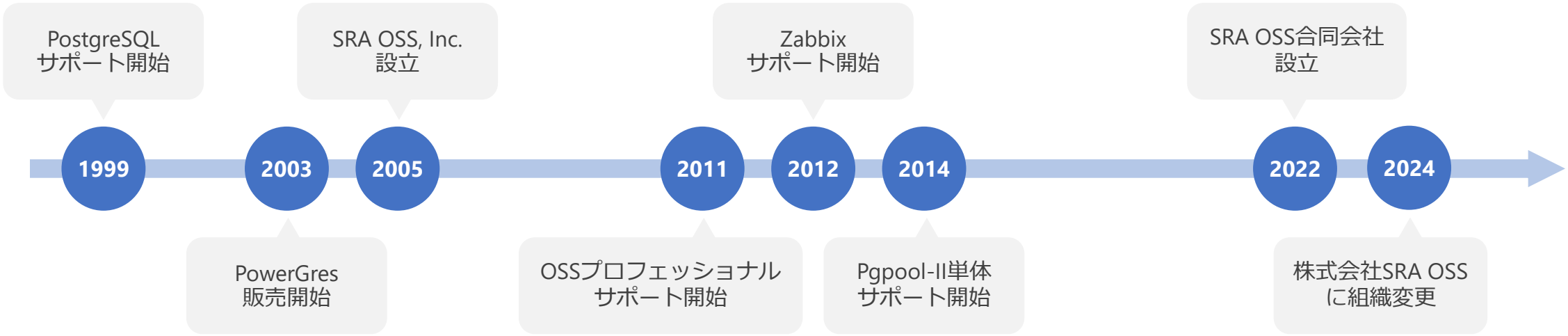
株式会社SRA OSS

所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主: 株式会社SRA
株式会社NTTデータ
資 本 金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧　　問: 石井 達夫
技術顧問: 増永 良文 (お茶の水女子大学名誉教授)



自己紹介

鳥越　淳

- SRA OSS OSS事業本部 データベース技術グループ所属
- PostgreSQLのサポート、案件支援などに従事
- PostgreSQL Contributor。主にモニタリング周りの機能開発に従事

馬 雪テイ

- システムインフラ開発室所属
- PostgreSQLサポート
- PostgreSQL HAクラスタ構築・コンサルティング
- 社内システム開発

目次

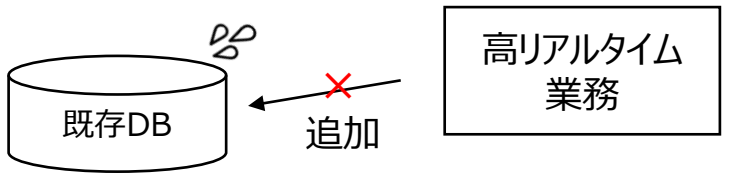
- CDC入門
- Debezium入門
- Debezium利用時のPostgreSQLからKafkaへのデータの流れ
- アーキテクチャなどから考えるDebeziumの注意点

CDC入門

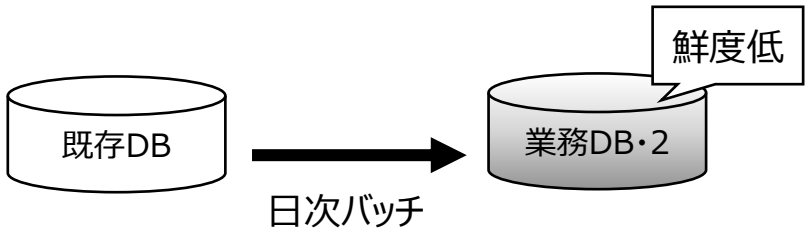
こんな要望はありませんか？

［リアルタイム性を活かしたい］

- リアルタイム性の高いデータを利用した業務に取り組みたいが、既存DBに業務処理を追加するのは、性能影響からNG



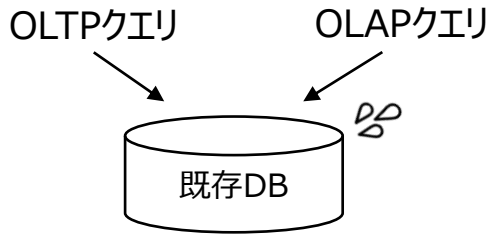
- DB間の連携はファイルなどを介して日次バッチ処理などで実施していたが、古いデータではお客様の要望を満たせなくなりつつある



こんな要望はありませんか？

[既存システムのオフロードをしたい]

- OLTP/OLAP系クエリの両方を既存DB上で実施していたが、データの蓄積などとともに過負荷状態に。別のDBへオフロードが必要(既存システムのオフロード)



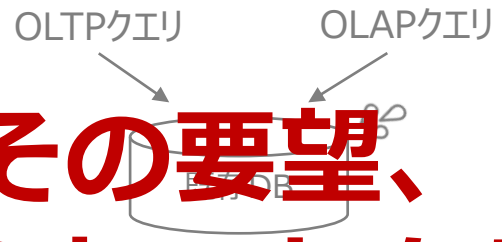
[複数データストアを連携したい]

- 既存のDBがいろいろある。サイロ化したデータストアから、分析に必要なデータを抽出するのが困難

身の回りにこんな要望はありませんか？

[既存システムのオフロードをしたい]

- OLTP/OLAP系クエリの両方を既存DB上で実施していたが、データの蓄積などとともに過負荷状態に。別のDBへオフロードが必要(既存システムのオフロード)



その要望、
CDCで叶えられるかもしれません

[複数データストアを連携したい]

- 既存のDBがいろいろある。サイロ化したデータストアから、分析に必要なデータを抽出するのが困難

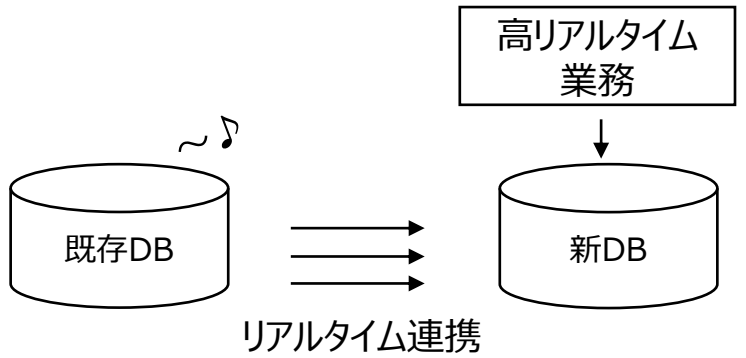
CDCとは

- Change Data Capture。データベースへの変更を観察し、他のシステムへレプリケーションできる形態で取り出すプロセス
  - 例えば、PostgreSQLへの更新内容を取り出し、分析用途でElasticsearchに投入
- 同期速度も比較的高速であることを謳う製品が一般的
  - ○数秒オーダー以下, ×日次バッチ
- レプリケーションとの違い
  - PostgreSQL本体に備わる物理/論理レプリケーションは、送信元・送信先いずれもPostgreSQL
  - 一方、CDCでは送信元がPostgreSQLでも、送信先は様々

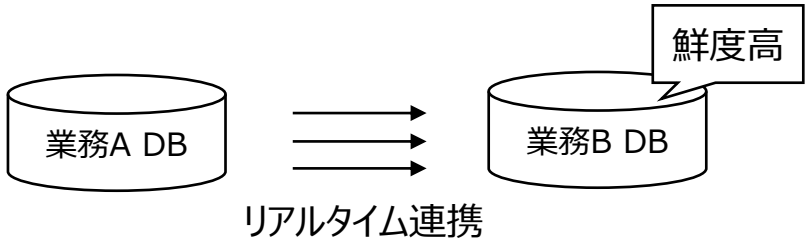
CDC実現後のイメージ

[リアルタイム性を活かした業務]

- データを新DBへ同期し、新DBにてリアルタイムな業務を実施



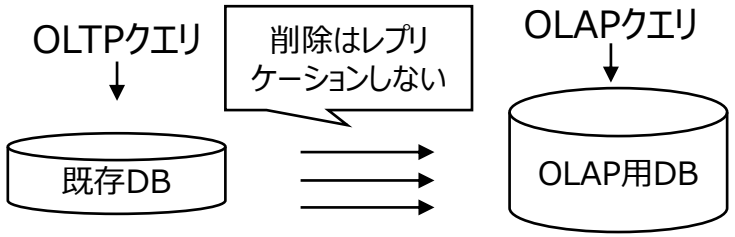
- 日次バッチ処理をCDCに置換、高鮮度のデータで業務を実施



CDC実現後のイメージ

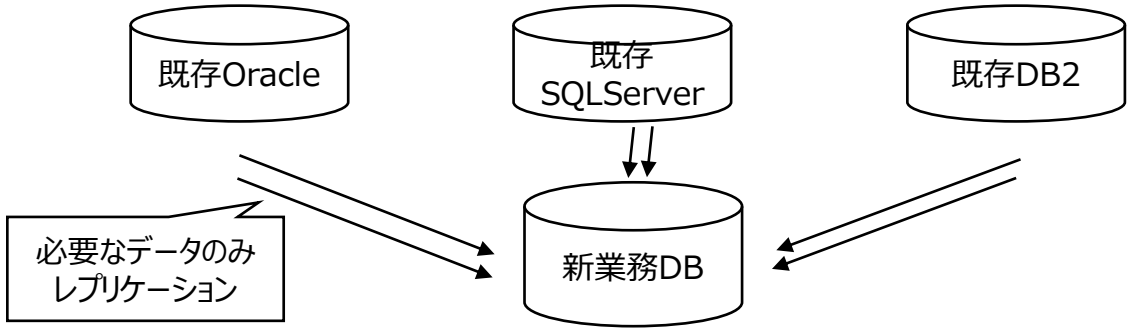
[既存システムのオフロード]

- OLAP用DBを構築。OLTPに不要な古いデータは既存DBから削除
- 削除はレプリケーションしないことで、OLAP用DBには古いデータも蓄積することも可能



[マルチデータストアの連携]

- 分析に必要なデータを各DBから抽出



CDCの実現方法

ソースDBから変更データを取り出す方法は大きく2種類:

トランザクションログベース: トランザクションログの内容をデコードし、変更内容を論理的に抽出、他DBへレプリケーション

Triggerベース: DBへの変更を監視するトリガーを利用。たとえば変更内容を更新ログテーブルに追加・定期的に更新ログテーブルの内容を同期先へ反映

CDCの実現方法

ソースDBから変更データを取り出す方法は大きく2種類:

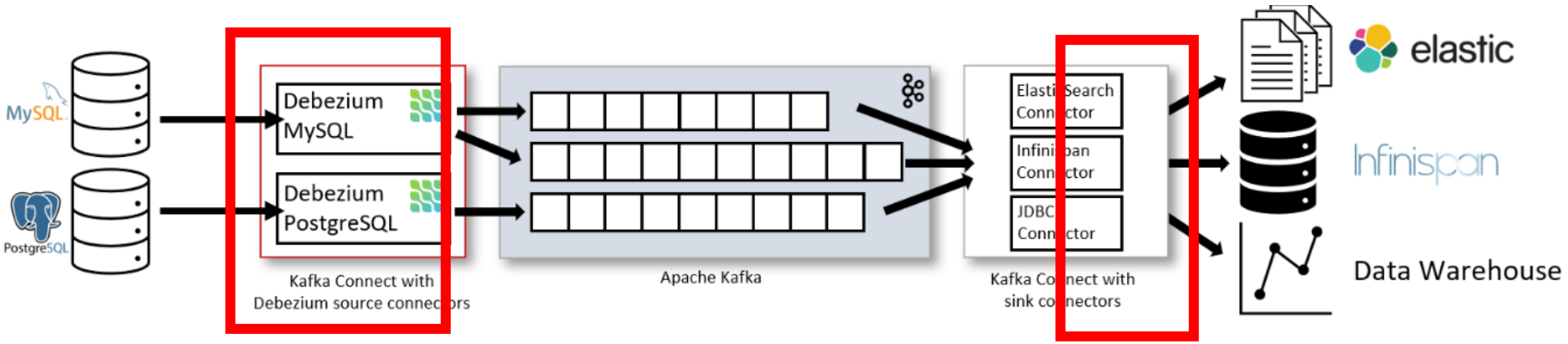
トランザクションログベース: トランザクションログの内容をデコードし、変更内容を論理的に抽出、他DBへレプリケーション

Triggerベース: DBへの変更を監視するトリガーを利用。たとえば変更内容を更新ログテーブルに追加・定期的に更新ログテーブルの内容を同期先へ反映

Debezium入門

Debeziumの概要

- オープンソースのCDCプラットフォーム
- MySQL, PostgreSQL, SQL Server, MongoDBなど各種DBのCDCに対応
- Apache License 2.0
- 主な開発元はRed Hat。Commonhaus Foundationに参加
- 基本的にApache Kafkaと連携してCDCを実現

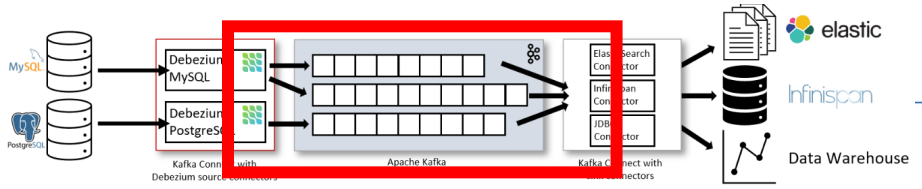


https://debezium.io/documentation/reference/stable/architecture.html

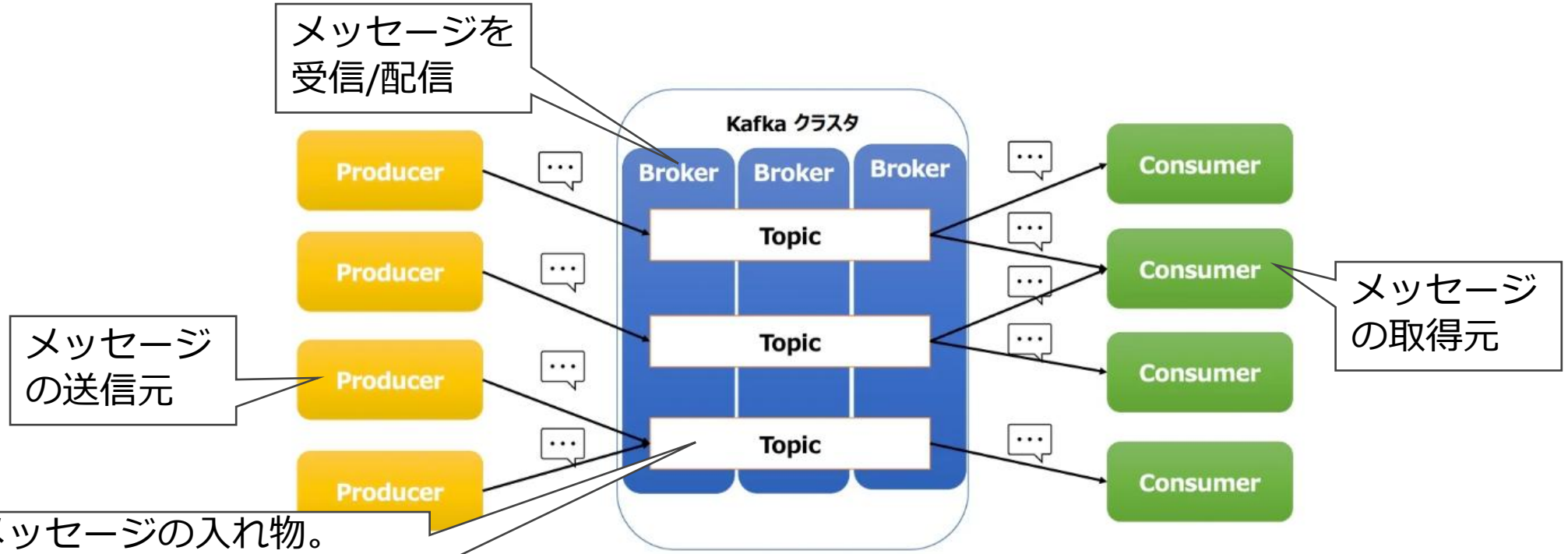
Debeziumの概要

- さまざまなDBと連携可能。以下バージョン3.3のsource connector一覧:
  - MySQL
  - MariaDB
  - MongoDB
  - PostgreSQL
  - Oracle
  - SQL Server
  - DB2
  - Cassandra
  - Vitess
  - Spanner
  - Informix

Kafkaの概要

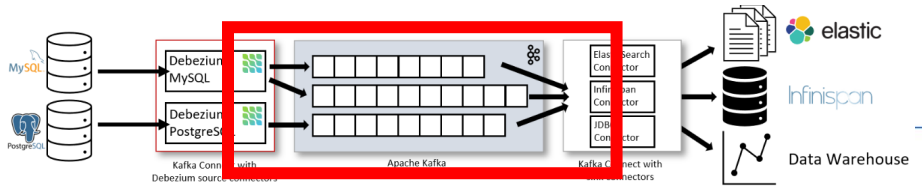


- オープンソースの分散メッセージングシステム
- あるシステムから送信されるメッセージを受け取り、別のシステムに渡すのが基本的な仕事
- 複数ノードで分散処理できスケーラブル、送達保証可能などの特徴

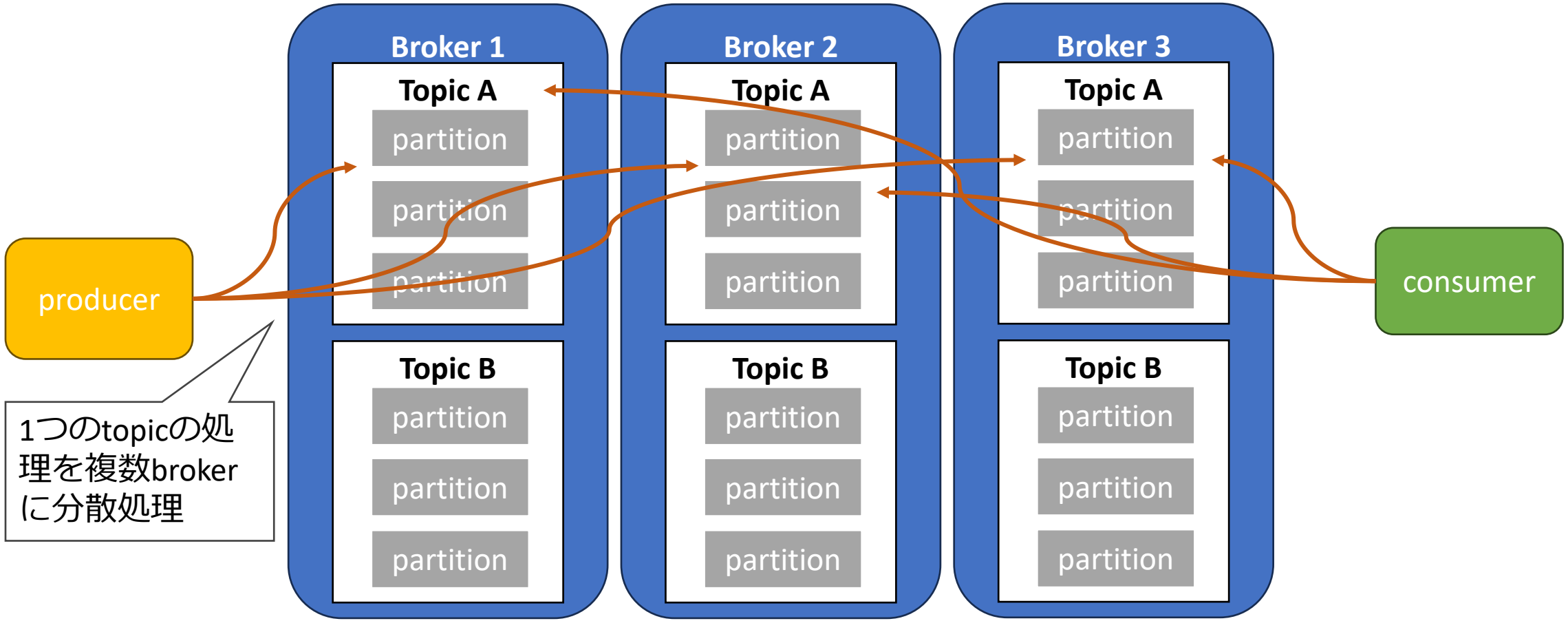


https://www.wantedly.com/companies/sraoss/post_articles/974841

Kafkaの概要

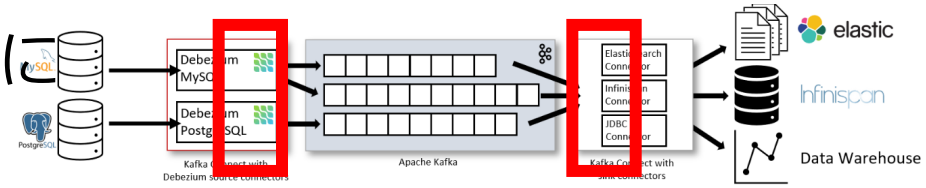


- 大量のメッセージを処理可能にするため、broker上の読み書きはパーティションという単位に分割し分散処理



Kafka Connectの概要

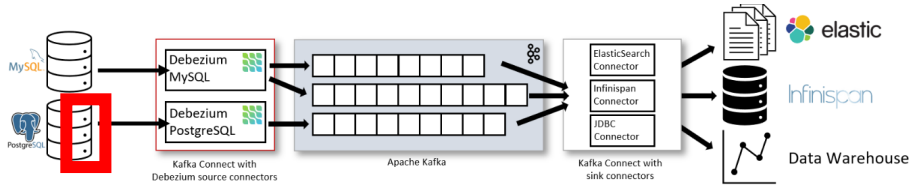
- Kafkaと他システム(例えばDB)のデータ連携に利用するフレームワーク
- Kafka Connectと連携先ごとのプラグインを利用。例. PostgreSQLからデータを取得するプラグイン
- DebeziumはKafka Connectorのプラグインの集合
- source connectorとsink connectorがある



DebeziumによるPostgreSQLからのデータ取得

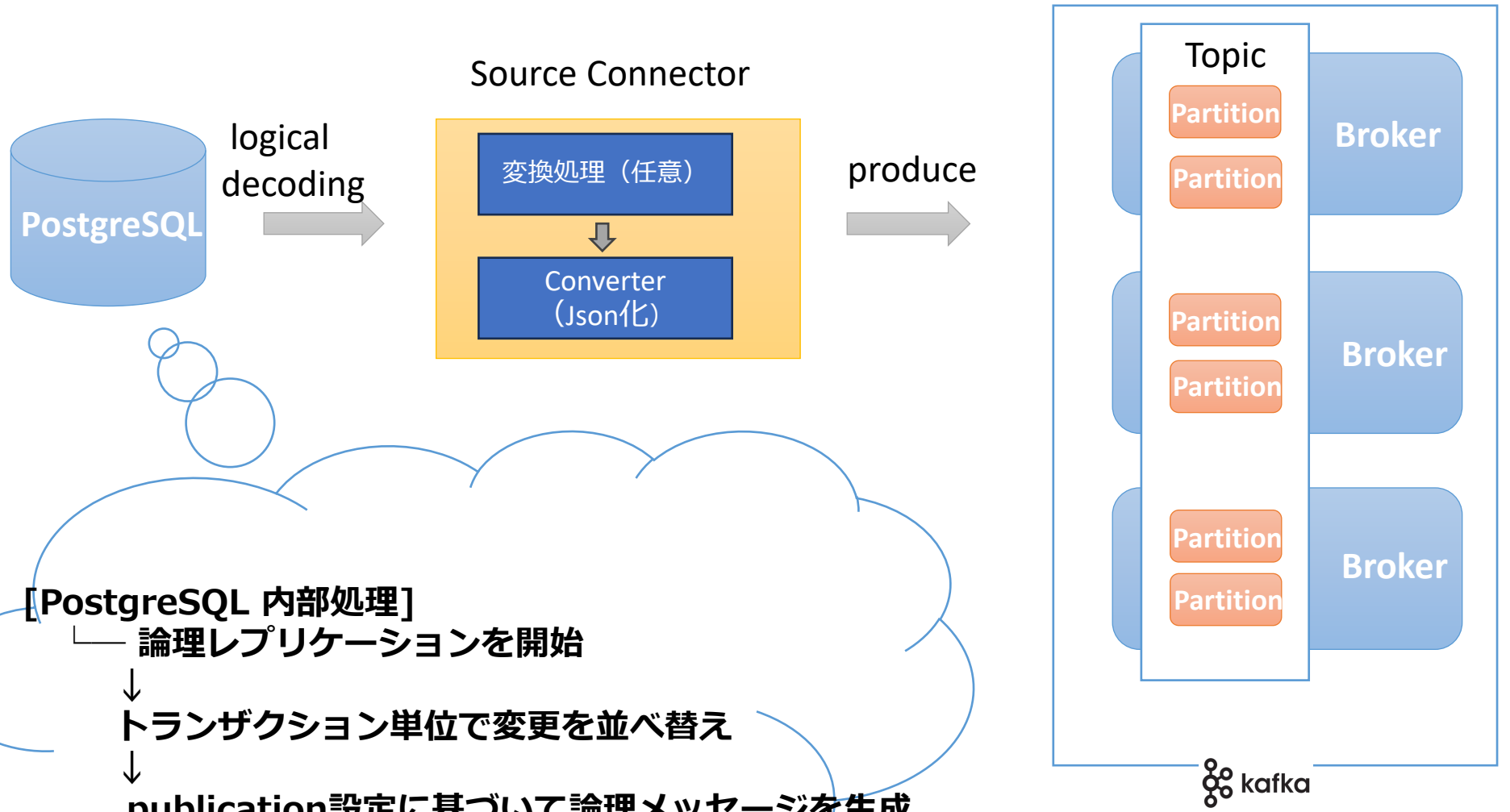
- DebeziumはPostgreSQLの**ロジカルデコーディング**機能を利用して変更内容（INSERT／UPDATE／DELETEなど）を取得する。

- ロジカルデコーディングとは
  - WALに記録された DB 変更内容を PostgreSQL 外でも読める形式に変換して出力する仕組み
  - 出力形式は使用するoutput pluginによって選択。Debeziumでは以下の2種類が利用可能
    - pgoutput（PostgreSQL 標準／論理レプリケーション用）
    - decoderbufs（Debezium 独自）
  - 本資料では以降 pgoutput前提で説明



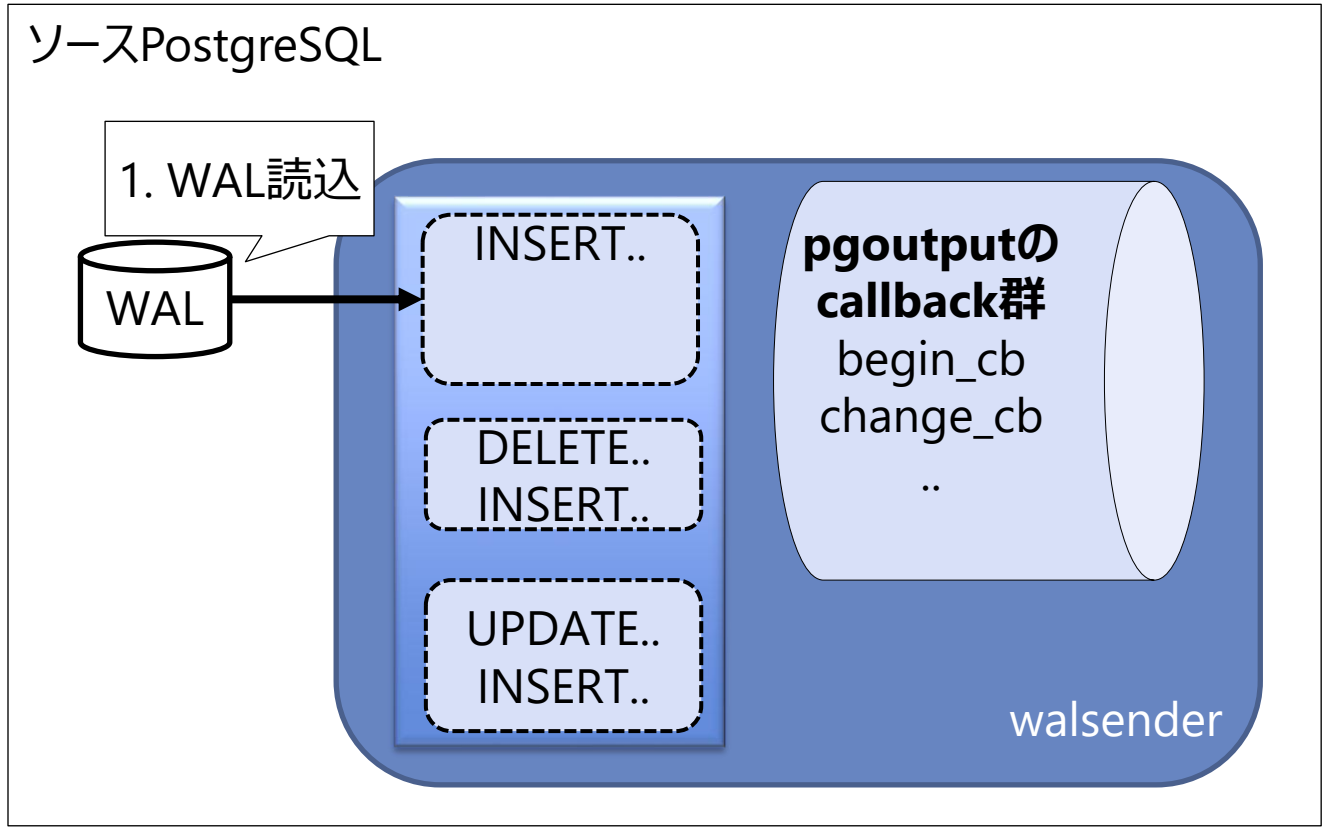
Debezium 利用時のPostgreSQLからKafka へのデータの流れ

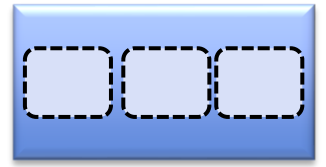
CDC アーキテクチャの全体像



PostgreSQLでのデータ処理

ロジカルデコーディングを実施。pgoutputを利用するので、PostgreSQL本体の論理レプリケーションと同じ処理となる。以下既存トランザクションに１件UPDATE・COMMIT実施した例:



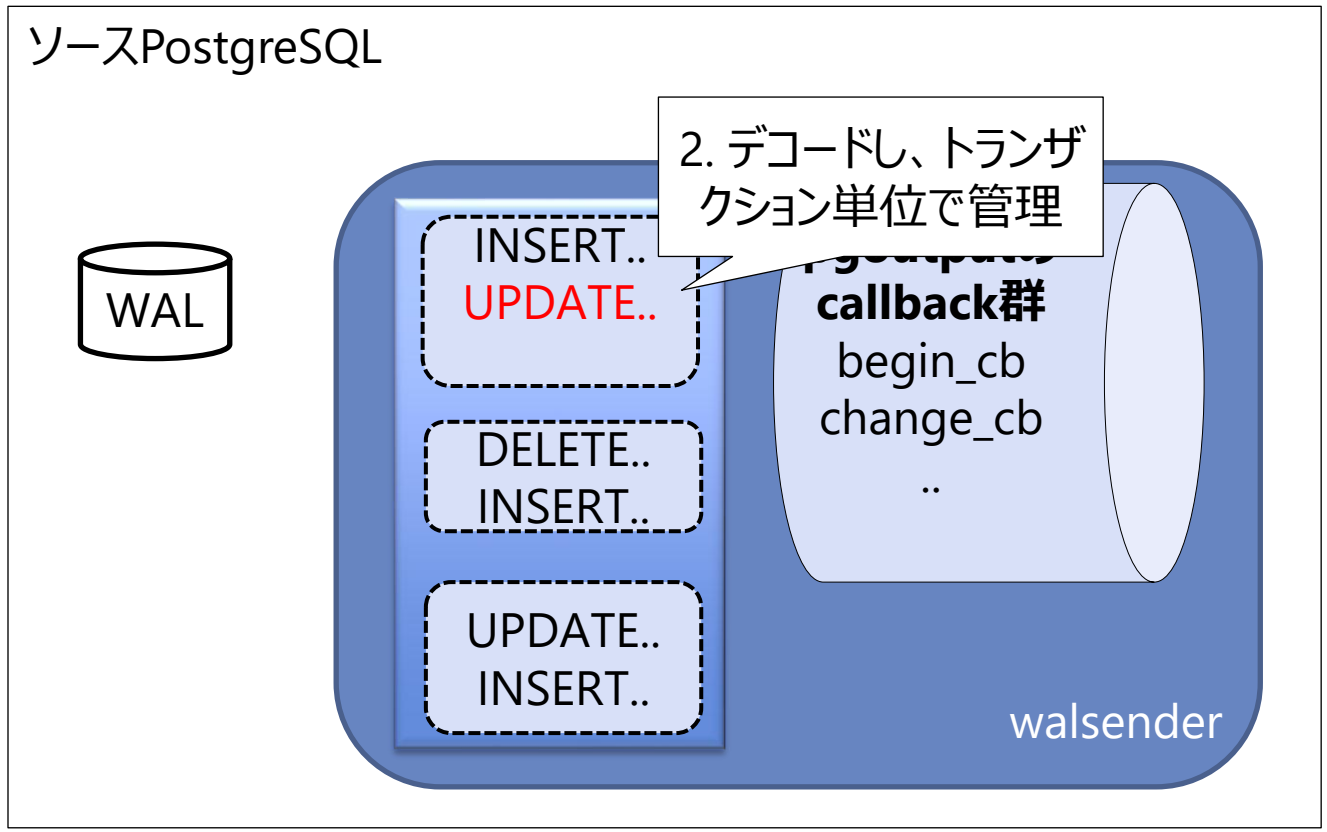


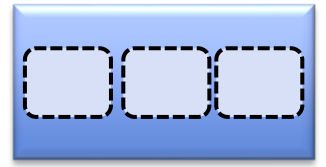
ReorderBuffer。
更新内容をトランザクションごとに管理

outputプラグイン。ここでは
pgoutput

PostgreSQLでのデータ処理

ロジカルデコーディングを実施。pgoutputを利用するので、PostgreSQL本体の論理レプリケーションと同じ処理となる。以下既存トランザクションに１件UPDATE・COMMIT実施した例:



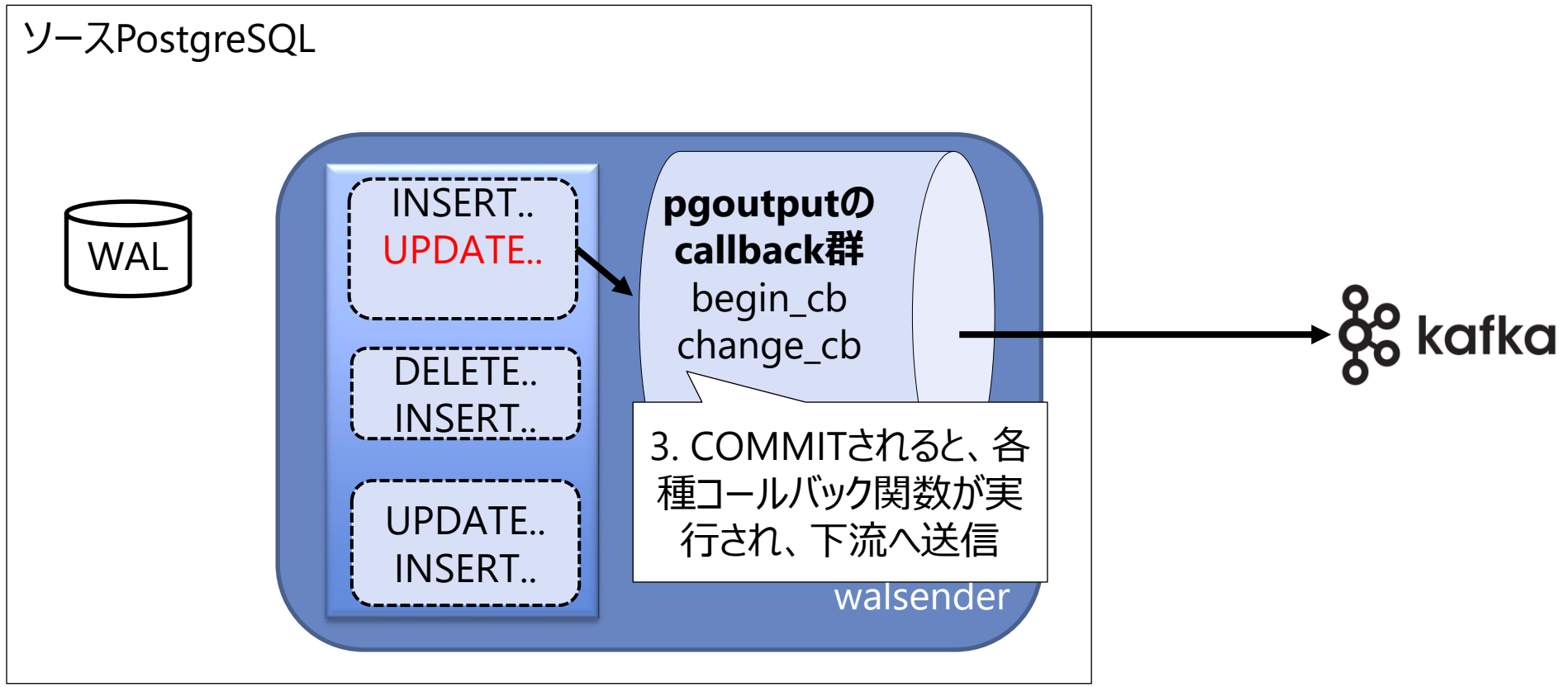


ReorderBuffer。
更新内容をトランザクションごとに管理

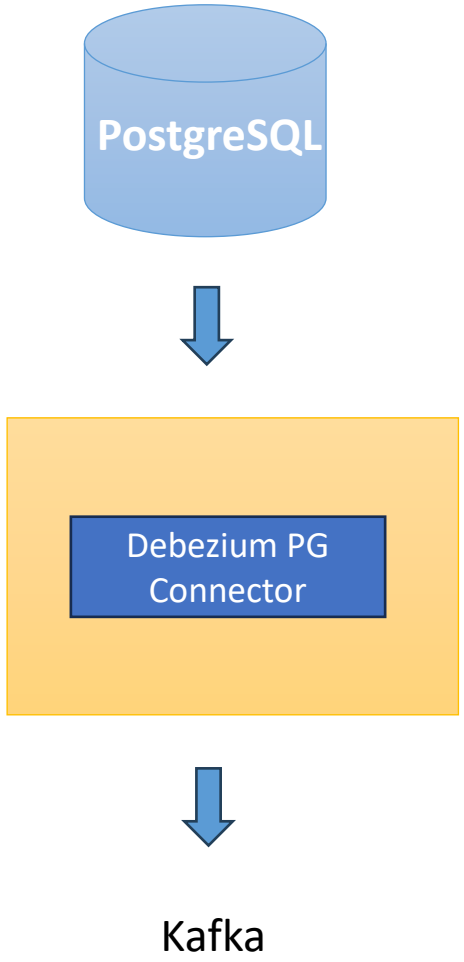
outputプラグイン。ここではpgoutput

PostgreSQLでのデータ処理

ロジカルデコーディングを実施。pgoutputを利用するので、PostgreSQL本体の論理レプリケーションと同じ処理となる。以下既存トランザクションに１件UPDATE・COMMIT実施した例:



Debezium PG Connectorのアーキテクチャ



[PostgreSQL]
　｜(デコードした変更イベントを送信)
　▼
[Debezium PostgreSQL Connector]
　├── 変更イベントを読み取る
　├── 変更イベント（before/after/op/source などの) オブジェクト生成
　├── Converter（JSON / Avro へシリアライズ）
　└── Kafka Topic に produce（書き込み）

INSERT / UPDATE / DELETE のイベント

INSERT（op=c）

アプリが INSERT を実行すると、PostgreSQL はその変更を WAL に記録。
pgoutput プラグインが 行データを論理メッセージ化 し、
Debezium がそれを受け取って JSON 形式に変換 → Kafka に送信

```
INSERT INTO customers (id, first_name, last_name, email)
VALUES (4, 'Debe', 'Test', 'd@example.com');
```

```
Key:
{"id": 4}

Value:
{
  "before": null,
  "after": {"id": 4, "first_name": "Debe", "last_name": "Test", "email":
"d@example.com"},
  "op": "c",
  "source": {"db": "inventory", "schema": "public", "table": "customers", "lsn":
27013880, "txId": 1069},
  "ts_ms": 1760601142773
}
```

1件INSERT / UPDATE / DELETEの流れ

UPDATE（op=u）

UPDATEはbeforeとafter。Keyは必須。

```
UPDATE customers SET last_name = ‘Updated’ WHERE id = 4;
```

```
Key:
{"id": 4}

Value:
{
  "before": {"id": 4, "first_name": "Debe", "last_name": "Test", "email":
"d@example.com"},
  "after":  {"id": 4, "first_name": "Debe", "last_name": "Updated",
"email": "d@example.com"},
  "op": "u",
  "source": {"db": "inventory", "schema": "public", "table": "customers",
"lsn": 27013912, "txId": 1071},
  "ts_ms": 1760601144801
}
```

1件INSERT / UPDATE / DELETEの流れ

DELETE（op=d）

DELETEはafter:null、op:d。キーで削除を表現。

```
DELETE FROM customers WHERE id = 4;
```

```
Key:
{"id": 4}

Value:
{
  "before": {"id": 4, "first_name": "Debe", "last_name":
"Updated", "email": "d@example.com"},
  "after": null,
  "op": "d",
  "source": {"db": "inventory", "schema": "public", "table":
"customers", "lsn": 27013944, "txId": 1072},
  "ts_ms": 1760601145902
}
```

初期同期スナップショット

概要：

- 初回起動時に、対象テーブルの既存データを整合性の取れた状態で一括取得する

仕組み：

- CDC対象のテーブルの全行を取得するSELECTクエリを実行して取得する
- このSELECTクエリと同時並行で進む書き込みはWALに記録されるので、スナップショット完了後にWALから追いかけて整合性を保つ

初期同期スナップショット完了後は これまでに説明した通常のCDCに移行

初期同期スナップショット

初期同期イベント（op=r）


```
{
  "before": null,
  "after": { "id": 1, "first_name": "Sally", "last_name": "Thomas", "email":
"sally.thomas@example.com" },
  "op": "r",
  "source": {
    "version": "3.2.3.Final",
    "connector": "postgresql",
    "name": "dbserver1",
    "snapshot": "first",
    "db": "inventory",
    "schema": "public",
    "table": "customers"
  },
  "ts_ms": 1759731208613
}
```


snapshot read event

初回スナップショットであること
を示す。以降、スナップショット
完了までは "true" が表示される。

アーキテクチャなどから考えるDebeziumの注意点

アーキテクチャなどから考えるDebeziumの注意点

① PostgreSQL論理デコードの制約とDebeziumの関係(1)

- DDL（CREATE / ALTER TABLE など）はCDC対象外
  → DDLのレプリケーションにはDebezium以外の仕組みが必要

- WALデコードは対象のDBクラスタのWAL全てに実施される
  → CDC対象外のDBについてWALや、publication対象外のテーブルについてのWALもデコードされる
  → CDC接続が増えるほど、WALデコードの処理負荷が増加
  → CDC対象の変更量が少なくても、DBクラスタのワークロードによってはデコードの負荷が大きい可能性あり

アーキテクチャなどから考えるDebeziumの注意点

① PostgreSQL論理デコードの制約とDebeziumの関係(2)

- PostgreSQLのREPLICA IDENTITYは、論理デコードで「どの列をキーとして行を識別するか」を決める

→ REPLICA IDENTITYがDEFAULTのまま、かつ主キーや一意キーがないテーブルでは、UPDATE／DELETE 時にbefore値（旧値）を取得できないため、UPDATE・DELETEがエラーとなる

対策：

→ 主キーまたは一意キーを設定する

→ REPLICA IDENTITYにFULLを指定して全列を出力。（※性能への影響が大きくなる可能性あり）

アーキテクチャなどから考えるDebeziumの注意点

① PostgreSQL論理デコードの制約とDebeziumの関係(3)

- 生成列の対応状況

→ 生成列（generated column） とは、**他の列の値を使って自動的に計算される列**のこと。

→ 構文例

```
total numeric GENERATED ALWAYS AS (price * qty) STORED
total numeric GENERATED ALWAYS AS (price * qty) VIRTUAL
```

※計算結果をテーブルに保存

※SELECT時に計算して返す

→ 生成列（STORED型）を含むテーブルをCDCの対象にしたい場合は、PostgreSQL 18以降を利用する

PostgreSQLバージョン	生成列サポート	論理デコード	備考
12 - 17	STORED型	対象外	Debeziumでは取得不可
18	STORED型、VIRTUAL型	STORED型のみ可能	STORED型はDebeziumで値を参照可能

アーキテクチャなどから考えるDebeziumの注意点

② commit 単位でまとめて送信

- Debezium利用時PostgreSQL側はトランザクションの COMMIT 時にイベントを送信

  → 大きなトランザクション（例：1トランザクションで数十万件更新／削除）があると、Kafkaへの反映が遅延する可能性がある。

- PostgreSQL 14に導入された実行中の大規模トランザクションをストリームする仕組み(トランザクション完了を待たずに、途中の変更データを逐次送信する機能)は **Debezium ではサポートしていない**

```
db=# SELECT slot_name, spill_txns, spill_bytes, stream_txns, stream_bytes, total_txns, total_bytes
    FROM pg_stat_replication_slots WHERE slot_name='debezium_slot';"
  slot_name   | spill_txns | spill_bytes | stream_txns | stream_bytes | total_txns | total_bytes
--------------+------------+-------------+-------------+--------------+------------+-------------
 debezium_slot |          1 |   135200000 |           0 |            0 |         91 |   274706394
(1 row)
```

アーキテクチャなどから考えるDebeziumの注意点

③ Debezium × Kafka における「1トピック1パーティション」問題

- 1テーブル = 1トピック
  → トピック名の形式：<serverName>.<schemaName>.<tableName>
  → デフォルト動作だが、変更は可能

- 各トピックはデフォルトでパーティション数 = 1
  → Kafkaは「同一パーティション内」のみで順序を保証する
  → 複数パーティション化すると、同じ行の更新順序が崩れる可能性がある

- Kafka 側でパーティション数を増やすことは可能だが、順序保証が崩れるため推奨されない

アーキテクチャなどから考えるDebeziumの注意点

③ Debezium × Kafka における「1トピック1パーティション」問題

単一パーティション

[Topic: dbserver1.public.customers]
Partition 0

---

① INSERT id=1　（新規登録）
② UPDATE id=1　（住所変更）
③ DELETE id=1　（削除）

Consumer側：同じ順で処理 ✅

1トランザクション内の複数操作（INSERT→UPDATE→DELETE）は、順序が保たれる

複数パーティション

Producer送信順：
① INSERT id=1
② UPDATE id=1
③ DELETE id=1

Kafkaの分配：
Partition 0 ← INSERT(id=1), DELETE(id=1)
Partition 1 ← UPDATE(id=1)

Consumer受信順：
INSERT → DELETE → UPDATE ❌

Consumer 側では順序が変わる可能性がある。例えば、削除済みデータを更新することが発生するかも

アーキテクチャなどから考えるDebeziumの注意点

④ 文字コードがUTF-8以外だと Debezium がエラー発生

Debezium は PostgreSQL側の文字コードはUTF-8のみサポート

JDBC 経由で UTF-8 セッションでスナップショット SELECT を行うため、その時点で DB 側の文字コード変換が失敗 → スナップショットが失敗し、リトライを繰り返しCDCできない

```
Caused by: org.postgresql.util.PSQLException: ERROR: invalid byte sequence for encoding "UTF8": 0x80
```

まとめ

- CDCによって、異種間データベースのレプリケーションが可能
- DebeziumはオープンソースのCDC基盤。基本的にApache Kafkaを利用
- Debezium PostgreSQL source connectorは、PostgreSQLの論理デコードを利用してDBへの変更を取得
- CDCは機能制約に注意が必要。Debezium PostgreSQL source connectorについていくつかご紹介。利用する環境・ワークロード上問題となる要因がないか要確認

参考

- Kleppmann, M. 著; 斉藤太郎 監訳; 玉川竜司 訳. 『データ指向アプリケーションデザイン - 信頼性、拡張性、保守性の高い分散システム設計の原理』. O'Reilly Japan, 2019. ISBN 978-4-87311-870-3
- Debezium Documentation
  https://debezium.io/documentation/reference/3.3/
- PostgreSQL Documentation
  https://www.postgresql.org/docs/devel/logicaldecoding.html
- 速習！論理レプリケーション@PostgreSQL Conference Japan 2022
  https://www.slideshare.net/slideshow/postgresql-logical-replication-postgresql-conference-japan-2022-nttdata/254219341

ご清聴ありがとうございました。

製品・サービスに関するお問い合わせ: sales@sraoss.co.jp 03-5979-2701