DRE/SREのプラクティス融合による クラウドネイティブなデータ基盤作り

Yuuki Takezawa / ytake

profile

- スターフェスティバル株式会社 / ほか
- データ処理やいろいろ
- Go / Scala

- マイクロサービスアーキテクチャ・リアクティブシステム、レガシー改善やSRE / DRE、各種モデリングの支援など
- X https://twitter.com/ex_takezawa

Agenda

- DREってどんなことをするの？
- クラウドならではのデータ基盤って？

DREってなんでしょう？

Data Reliability Engineering

- SREのプラクティスをデータ品質監視などに適用

- **ビジネスや意思決定に利用する**
  データの信頼性に関する活動

どんなことを進めていくのか

データ基盤ってなにをするの？

- 会社のドメインによっては捉え方は異なります
- データ活用を前提にデータ収集や、管理、作成など
  データに関する基盤
- データをアプリケーションにリバースETLしたり、
  機械学習に利用されたり

データ活用とは

- 会社活動などにおける「意思決定」や「業務効率化」、
  「マーケティング」などの向上に役立てる

- データドリブンな企業文化作りになくてはならないもの

- どういうデータをエビデンスにしていけば良いか、
  などは会社によって全く違うため、
  何かを参考にすると活用ができるわけではない

活用できるようにするためには

事業・会社を知る

- 事業のドメインをしっかりと理解すること
- ドメインを理解しなければデータに関連する活動が難しい
- 単純にデータを集約する、だけでは活用しにくい

事業・会社を知る

- 事業の理解度を上げることで未来を考えられるように

- 未来に行くために 現在何が足りていないのか、
  どういう戦略を立てる必要があるのか

- **データをどう活用すると・どんなものがあると**
  目指したい方向に向かえるのか

データを見る、業務を知る

- 現状のデータがどのようになっているのか
- どういった業務フロー、アプリケーションがあり
  どのタイミングでデータが作られるのか
- どのタイミングのデータが活用できるのか

データを見る、業務を知る

- データ設計はアプリケーション寄りなのか、
  活用にも耐えうる設計になっているのか

- 保管されるデータの品質は
  意思決定や事業活動に対して有用かどうか

あるものを信用しない

- 今あるものは不十分なものかもしれない
- こういうデータが必要だ
- 活用するために必要なデータがなければ
  どこからそのデータが作られそうか（パイプライン）

あるものを信用しない

- 事業ドメインを理解していなければ、
  データが**信用できるかどうかの判断がつかない**

- データがどのようになっているべきか

- 改善を待つのではなく、改善するために必要なものを
  自ら実行する

例えば

- CVRなどを正しく認知できるデータは揃っているか？

- カート落ちはどのくらい？
  その時カートにあるものは？
  購入情報とカート情報は紐づいてる？

- 利用しているSaasサービスとデータは結合できる？

言われた通りにデータの集約、
転送をすればよい、と思っていませんか？

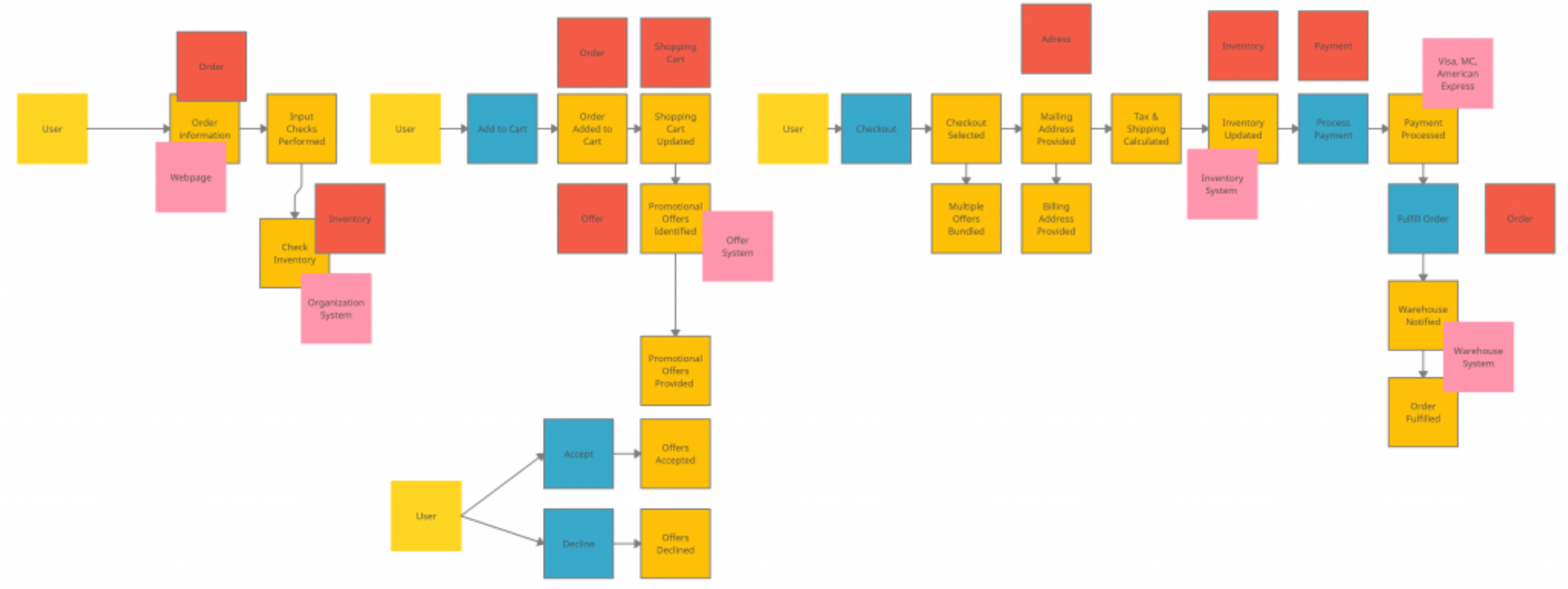
ユーザーインタラクションを理解する

- ユーザーとは社内外問わず利害関係者全て
- データは何かの事象のスナップショット
- どのようなものが事象と扱えるのか

インタラクションを理解するには

- アプリケーションのフローを網羅する
- ドメインイベントを見つけ出す
- ドメインエキスパートに並走してもらう
- トラッキングの設計などにも活用

どこから生まれてどこにいくのか



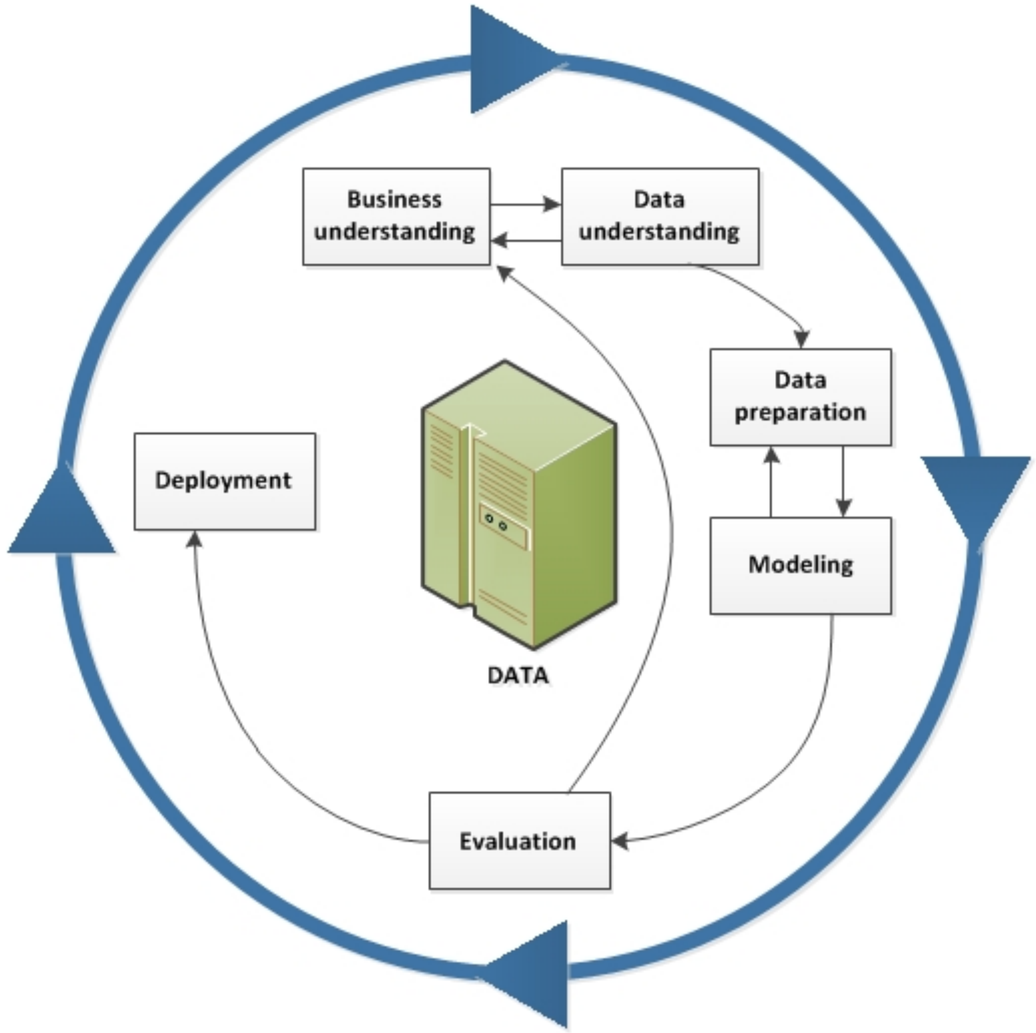
イベントストーミング

- サービスなどに関わるメンバーを集めて実施
- どのような出来事が起こるか
- コンテキストはどのようになっているか
- システムとドメインの関係性を紐解く

イベントストーミング

- DRE活動を行うメンバーだけでやらない
- 信頼性を上げていくためには文化作り
  意識作りが必要不可欠
- 関係者を巻き込み認識を合わせること

データに関する思考


Business understanding
Data understanding
Data preparation
Modeling
Evaluation
Deployment
DATA

CRISP DM

- データ分析のフレームワーク
- データに関する信頼性とはなにかを理解する
- 信頼性を改善・維持していくためには必要不可欠

ビジネスの理解

- 事業の方向性や成長戦略、ビジネスの目標・目的を決める
- 現在とのギャップや状況を把握
- 達成基準を設け、達成にはどのようにすべきかを決める

データの理解

- 現在どのようなデータがあるかを調査
- RDBMSだけでなく、エクセルなどのソフトウェア
  各種SaaSなどの利用状況も把握
- 業務フローなども理解

データの準備

- どのデータが有用か、どんなデータが不足しているか
- 統合や集約、データクレンジングなどを行う
- どのようなものを用いてデータ基盤を作るかに関わる

モデリング

- ビジネス目標などの達成をサポートできる
  モデリングなどを行う

- 分析軸や、利用方法に応じて複数の軸でモデル作成

- 時系列や集計なのか、しっかりと理解しましょう

評価

- ビジネス目標などの達成をサポートできるものか評価
- どの程度達成できるか、どういった成果になりそうか
- 作るだけではなくアウトカムを大事に

展開・共有

- 十分な評価が得られれば、フローや
  システム・アプリケーションなどに導入

- 計測可能な効果測定なども行い
  戦略などに併せてメンテナンスなどを継続する

利害関係者を理解する

- データが関係する戦略やフローを理解することで
  信頼性とは何かが理解できる

- どのような頻度、どのような戦略でデータに関する
  問題を解決していくか

データの準備

データの準備は難しい

- 複数のステークホルダ
- サイロ化・あちこちにあるデータ
- 連続性のないデータ

サイロ化

- プロダクトなどのチームで
  すでにデータ抽出や拡張が行われている
- 会社を横断した状況判断がしたい、などが難しい
- セキュリティやガバナンスも効かせられない

連続性がない

- プロダクト独自のID管理、複雑化したIDマッピング
- 一つのシステムでは一貫性があっても
  横断的なデータとしてみると混在
- データにおけるライフサイクルの違い

解決するには

- フィジカルSSoT or ロジカルSSoT

- サイロ化が進んでしまっている場合は
  フィジカルSSoTからでなければ難しい

- **データの信頼性に直結する問題**

クラウドネイティブなデータ基盤

Athena / フィジカルSSoT

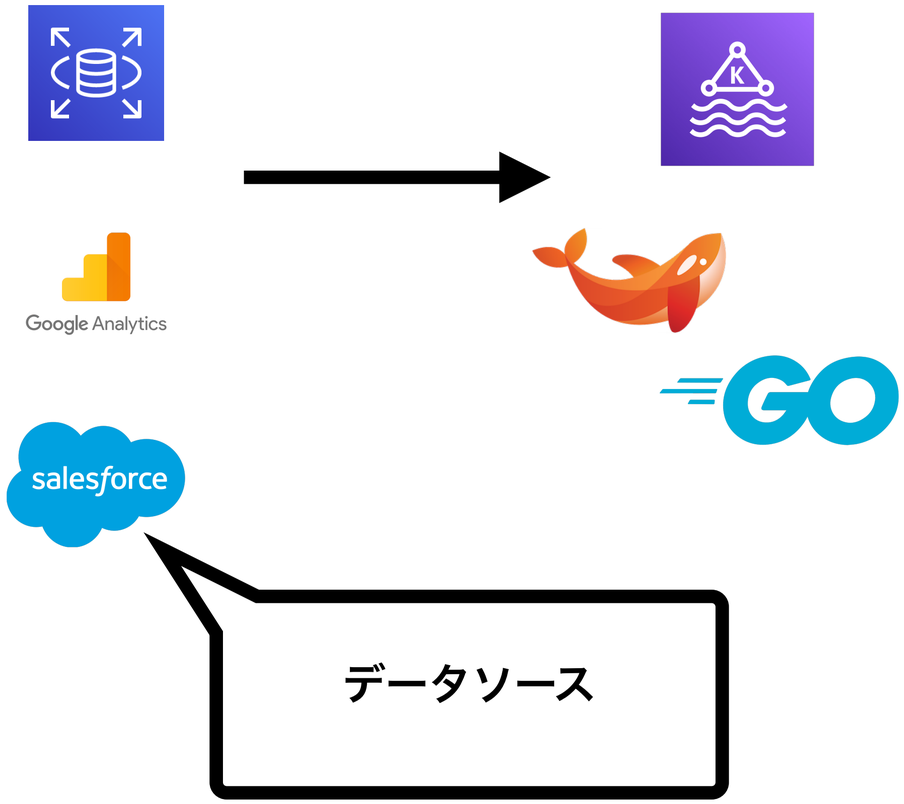
- ELTでS3にとにかく転送

- ドメインイベントなどを理解したうえで
  ほとんどが結果整合で問題ない

- チームの規模が小さいと不自由なし
- パーティション設計も思いのまま

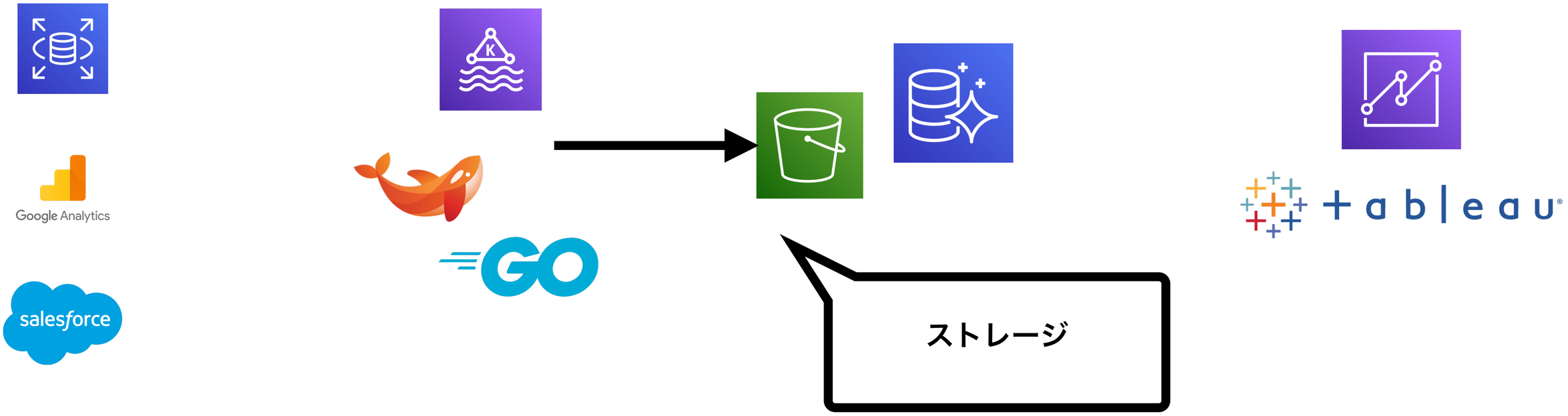
レイヤ

フィジカルSSOT



フィジカルSSOT

フィジカルSSOT



フィジカルSSOT

問題

- 関わる人が増えるとコントロールが難しい
- データ基盤に着手していくと
  思ってもいなかった事象に出会うことも
- すでにサイロ化が加速している場合
  フィジカルSSOTだけでは限界

問題

- 権限が難しい

- ちょっとした処理はSparkなどが必要なため
  セルフサービス化しても、浸透が難しい
  （慣れない技術スタックへの拒否）

Snowflakeへ移行

- フィジカル or ロジカルともに解決ができる
- 標準的なSQLだけで多くが解決できる
- 外部ネットワークアクセス可能
- Athenaなどで利用していたParquetがそのまま利用できる

モデリング・評価

データのライフサイクルを認識する

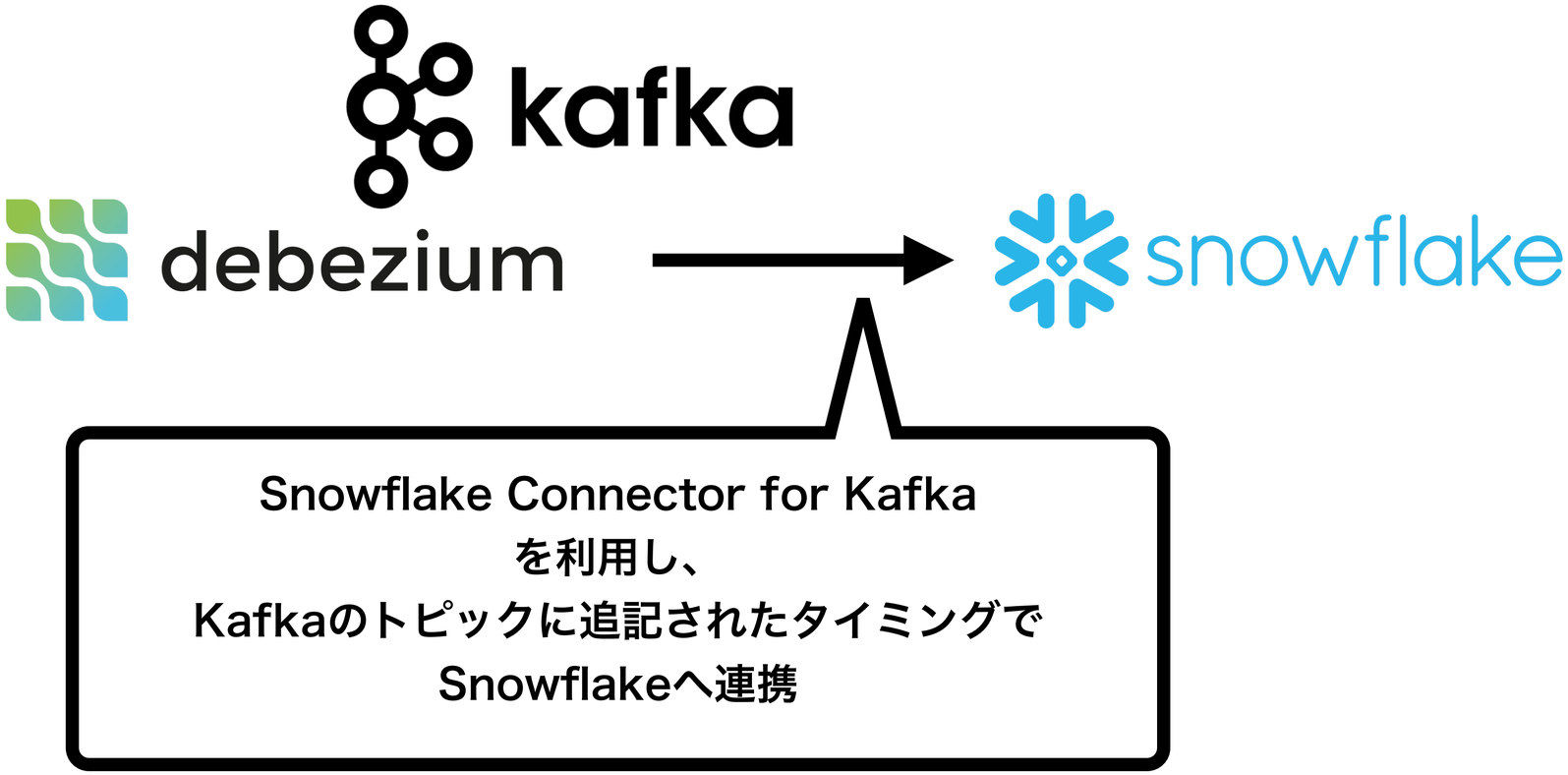
- 例えば ドメインイベント発生からいつまでに
  データが連携されるべきか

- SLIメニュー freshness などを参考に戦略、
  利用用途、データモデルに併せてデータの鮮度を決定

ニアリアルタイム


kafka
debezium
snowflake
RDSなどのレコード変更を検知して
Debeziumが行情報をKafkaへ転送
必要に応じてOutboxパターンなどを導入

ニアリアルタイム


kafka
debezium
snowflake
Snowflake Connector for Kafka
を利用し、
Kafkaのトピックに追記されたタイミングで
Snowflakeへ連携

レコード例

```
create or replace view HOGE_DB.RAW_DATA.TEST_IU_VIEW (
    ID,
    NAME,
    DEBEZIUM_PROCESSED_TS,
    SOURCE_PROCESSED_TS,
    SOURCE_SERVER,
    SOURCE_DB,
    SOURCE_TABLE,
    DML_OPERATOR
) as
SELECT
    record_content:"after"."id"::NUMBER as ID,
    record_content:"after"."name"::NUMBER as NAME,
    record_content:"ts_ms"::STRING::DATETIME as DEBEZIUM_PROCESSED_TS,
    record_content:"source"."ts_usec"::STRING::DATETIME as SOURCE_PROCESSED_TS,
    record_content:"source"."name"::STRING as SOURCE_SERVER,
    record_content:"source"."db"::STRING as SOURCE_DB,
    record_content:"source"."table"::STRING as SOURCE_TABLE,
    record_content:"op"::STRING as DML_OPERATOR
FROM HOGE_DB.RAW_DATA.CDC_RAW_TESTS
WHERE lower(DML_OPERATOR) in ('r', 'c', 'u');
```

レコード例

```
create or replace view HOGE_DB.RAW_DATA.TEST_IU_VIEW (
    ID,
    NAME,
    DEBEZIUM_PROCESSED_TS,
    SOURCE_PROCESSED_TS,
    SOURCE_SERVER,
    SOURCE_DB,
    SOURCE_TABLE,
    DML_OPERATOR
) as
SELECT
    record_content:"after"."id"::NUMBER as ID,
    record_content:"after"."name"::NUMBER as NAME,
    record_content:"ts_ms"::STRING::DATETIME as DEBEZIUM_PROCESSED_TS,
    record_content:"source"."ts_usec"::STRING::DATETIME as SOURCE_PROCESSED_TS,
    record_content:"source"."name"::STRING as SOURCE_SERVER,
    record_content:"source"."db"::STRING as SOURCE_DB,
    record_content:"source"."table"::STRING as SOURCE_TABLE,
    record_content:"op"::STRING as DML_OPERATOR
FROM HOGE_DB.RAW_DATA.CDC_RAW_TESTS
WHERE lower(DML_OPERATOR) in ('r', 'c', 'u');
```

物理削除の場合は beforeを参照

レコード例

```
create or replace view HOGE_DB.RAW_DATA.TEST_IU_VIEW (
    ID,
    NAME,
    DEBEZIUM_PROCESSED_TS,
    SOURCE_PROCESSED_TS,
    SOURCE_SERVER,
    SOURCE_DB,
    SOURCE_TABLE,
    DML_OPERATOR
) as
SELECT
    record_content:"after"."id"::NUM
    record_content:"after"."name"::N
    record_content:"ts_ms"::STRING::
    record_content:"source"."ts_usec
    record_content:"source"."name"::
    record_content:"source"."db"::ST
    record_content:"source"."table":
    record_content:"op"::STRING as DML
FROM HOGE_DB.RAW_DATA.CDC_RAW_TESTS
WHERE lower(DML_OPERATOR) in ('r', 'c', 'u');
```

物理削除の場合は d

```
MERGE INTO HOGE_DB.STAGING_DATA.TESTS as tgt
USING (
    SELECT * FROM (
        SELECT *,
            ROW_NUMBER() over (
                PARTITION BY ID
                ORDER BY DEBEZIUM_PROCESSED_TS DESC
        ) as row_num
    FROM (
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TEST_IU_VIEW
        UNION ALL
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TEST_D_VIEW
        ) as u
    ) as t1 WHERE t1.row_num = 1
) as src
ON tgt.ID = src.ID
WHEN MATCHED AND src.DML_OPERATOR = 'd' THEN DELETE
WHEN MATCHED AND src.DML_OPERATOR = 'u' THEN
UPDATE SET
    tgt.ID = src.ID,
    tgt.NAME = src.NAME
WHEN NOT MATCHED AND src.DML_OPERATOR IN ('c', 'r', 'u') THEN
INSERT (
    ID,
    NAME
)
VALUES  (
    src.ID,
    src.NAME
);
```

```
MERGE INTO HOGE_DB.STAGING_DATA.TESTS as tgt
USING (
    SELECT * FROM (
        SELECT *,
            ROW_NUMBER() over (
                PARTITION BY ID
                ORDER BY DEBEZIUM_PROCESSED_TS DESC
        ) as row_num
    FROM (
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TEST_IU_
        UNION ALL
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TEST_D_V
        ) as u
    ) as t1 WHERE t1.row_num = 1
) as src
ON tgt.ID = src.ID
WHEN MATCHED AND src.DML_OPERATOR = 'd' THEN DELETE
WHEN MATCHED AND src.DML_OPERATOR = 'u' THEN
UPDATE SET
    tgt.ID = src.ID,
    tgt.NAME = src.NAME
WHEN NOT MATCHED AND src.DML_OPERATOR IN ('c', 'r', 'u') THEN
INSERT (
    ID,
    NAME
)
VALUES  (
    src.ID,
    src.NAME
);
```

RDSなどからレコードが削除された場合は
Snowflakeからも削除

```
MERGE INTO HOGE_DB.STAGING_DATA.TESTS as tgt
USING (
    SELECT * FROM (
        SELECT *,
            ROW_NUMBER() over (
                PARTITION BY ID
                ORDER BY DEBEZIUM_PROCESSED_TS DESC
        ) as row_num
    FROM (
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TES
        UNION ALL
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TES
        ) as u
    ) as t1 WHERE t1.row_num = 1
) as src
ON tgt.ID = src.ID
WHEN MATCHED AND src.DML_OPERATOR =  ' THEN DELETE
WHEN MATCHED AND src.DML_OPERATOR = 'u' THEN
UPDATE SET
    tgt.ID = src.ID,
    tgt.NAME = src.NAME
WHEN NOT MATCHED AND src.DML_OPERATOR IN ('c', 'r', 'u') THEN
INSERT (
    ID,
    NAME
)
VALUES  (
    src.ID,
    src.NAME
);
```

レコードが更新された場合は
Snowflakeのレコードも更新

```
MERGE INTO HOGE_DB.STAGING_DATA.TESTS as tgt
USING (
    SELECT * FROM (
        SELECT *,
            ROW_NUMBER() over (
                PARTITION BY ID
                ORDER BY DEBEZIUM_PROCESSED_TS DESC
        ) as row_num
    FROM (
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_TS,
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TEST_IU_VIEW
        UNION ALL
        SELECT
            ID,
            NAME,
            DEBEZIUM_PROCESSED_T
            DML_OPERATOR
        FROM HOGE_DB.RAW_DATA.TE
        ) as u
    ) as t1 WHERE t1.row_num = 1
) as src
ON tgt.ID = src.ID
WHEN MATCHED AND src.DML_OPERATO
WHEN MATCHED AND src.DML_OPERATO
UPDATE SET
    tgt.ID = src.ID,
    tgt.NAME = src.NAME
WHEN NOT MATCHED AND src.DML_OPERATOR IN ('c', 'r', 'u') THEN
INSERT (
    ID,
    NAME
)
VALUES  (
    src.ID,
    src.NAME
);
```

Snowflakeにデータがないが、
削除以外のレコード操作があった場合は
Insert

鮮度が良い場でもシンプルに

- 多少の連携設定はありますが、SQLだけで完結
- 連携が失敗してもデータは壊れない
- 連携し直したい場合はDebeziumで頭から再連携
- **リスクも理解し、影響範囲を最小限に**

考慮

- レコード数が多い場合や、
  余計な通信が発生しないようにしたい場合などはS3なども併用

- リアルタイム性が本当に必要かどうか
  求められる鮮度・信頼性を明らかにする

- dbtなども利用

外部ネットワーク経由の連携

- APIが公開されているサービスなどはそのまま連携可能
- UDFやストアドプロシージャとして組み込めるもの
- データ基盤などのストレージになくても
  論理的に統合可能

トイルの削減

トイルの削減

- セルフサービスのための自動化
  ELT / Snowflakeなど

- プロダクトごとの可視化ツール導入などのための
  権限設計

- アラートの通知

トイルの削減

- Schema Evolution
- ジョブスケジュール管理
- リトライ

なんでも自動化すれば良いわけではない

- 自動化しすぎたことによる事故
- チームの認知度負荷
- チームの健康状態・モチベーション

品質テスト

正確性

- 他環境で作られたデータと差異がないかどうか
- 金額計算などのズレは正確な意思決定につながらない
- 計算式やレコード数、カラムなども期待値を満たすか
- **どのくらい許容されるものか認識をあわせる**

正確性

- **XXから連携されたデータが28日間で**
  取り扱い金額と99.99% 一致していること など

- 期間内で、定めた件数以上未連携やずれがある場合は
  対応をしなければならない

- 認識が合った上で軽微なものであれば再連携実行など

一貫性

- 一貫性のあるフォーマットに沿っているかどうか

- **おかしなデータフォーマットが混在していないかどうか**
  整合性に問題がないか

- 同様にSLOなどを明記し、
  データの修正 or フローなど**本質的な改善**を行う

ここにあるものはほんの一部

まとめ

- 事業理解とドメインをしっかりと噛み砕く
- 事業の未来を想像し、そこに必要なデータを理解する
- SREのプラクティスを取り入れ、多くを巻き込んで推進
- クラウドを活用して利用しやすいデータ基盤作り