Kinesis Analyticsを触ってみよう

2017/3/11

株式会社　NTTドコモ
サービスイノベーション部 ビッグデータ担当
高田　雅人

目次

- 自己紹介
- Kinesisの紹介
- ハンズオン
- 現時点での評価

自己紹介

- **所属/名前**
  - NTTドコモ　サービスイノベーション部　ビッグデータ担当
  - 高田　雅人(たかだ　まさと)

- **業務**
  - ビッグデータ処理基盤の構築・活用推進、ストリーム基盤開発
  - NW分析　※無線通信品質の分析

- **趣味**
  - 短距離（100m)：10秒5
  - 運動全般（サッカー、ゴルフ、ボウリング）
  - 飲み会（二日酔いがひどい）

- **好きなAWSサービス**
  - Kinesis
  - Redshift

Qiita

- **Qiitaに整理したので事前にアクセスしてください**
  - **Kinesis AnalyticsのSQLクエリ**
  - **http://qiita.com/takada_tf/items/f03c36eed9e22eb74744**
- **アクセス方法は「Qiita」内検索欄で[kinesis analytics]と検索してください**

モチベーション

- 社内でリアルタイム処理基盤を開発することに
- 処理した結果を複数システムに送る必要がある
  → kafka or Kinesis Streams
- しかも機械学習を行う
  → Spark Streaming or Storm or Kinesis Analytics

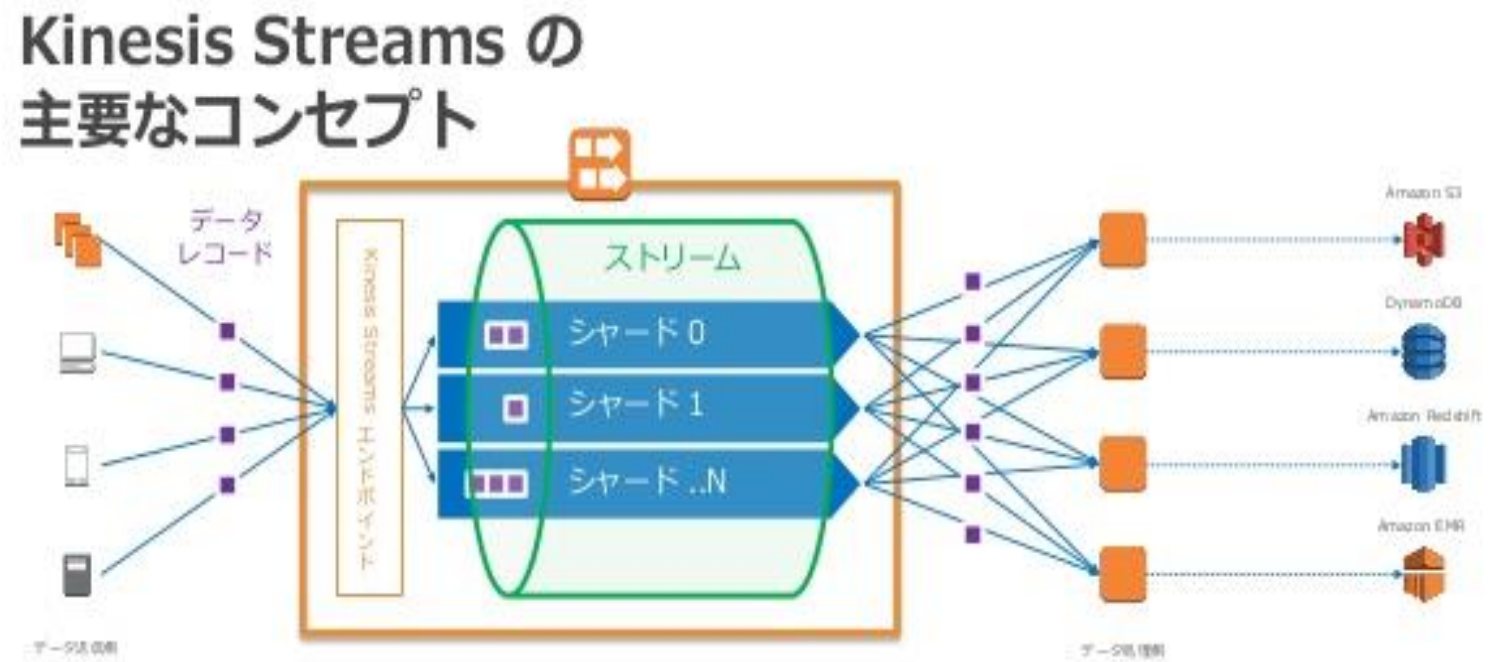
Amazon Kinesis

- **Kinesisは３つサービス**

資料参考：
http://www.slideshare.net/AmazonWebServicesJapan/amazon-kinesis-analytics

Kinesis Streams

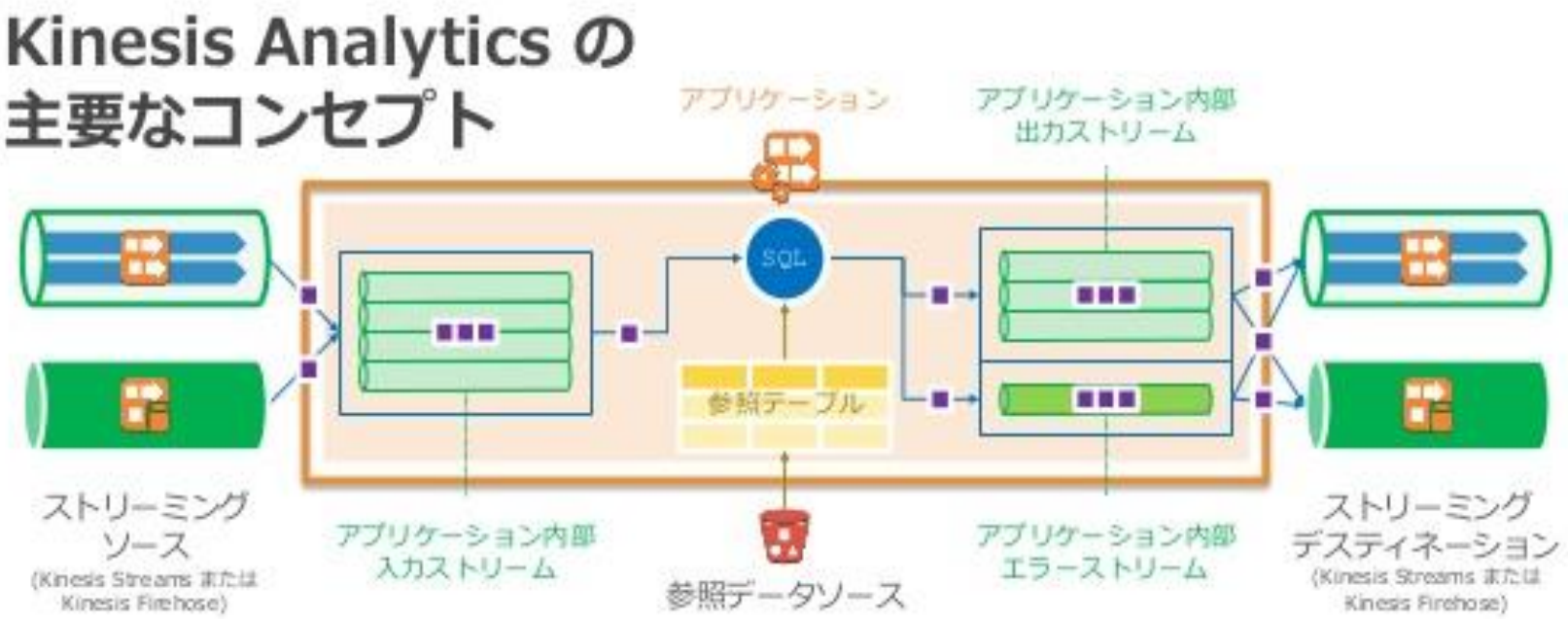
- **一言でいえば、マネージドkafka**



- データの種類や処理の用途に応じて「ストリーム」を作成。ストリームは 1 つ以上の「シャード」で構成
- 保存されるデータの単位を「データレコード」と呼び、保持期間はデフォルトで 24 時間／最長で 7 日間
- 1 データレコードの最大サイズは 1 MB
- データ送信側のキャパシティは 1 シャードあたり秒間 1 MB もしくは 1,000 PUT レコード
- データ処理側のキャパシティは 1 シャードあたり秒間 2 MB もしくは 5 回の読み取りトランザクション
- ストリーム内のシャード数を増減することでスループットをコントロール

資料参考：
http://www.slideshare.net/AmazonWebServicesJapan/amazon-kinesis-analytics

Kinesis Analytics



- 分析単位に「アプリケーション」を作成し、入力/出力となる「ストリーミングソース/デスティネーション」を設定
- ストリーミングソース/デスティネーションはアプリケーション内部の「入力/出力ストリーム」に対応
- SQL でアプリケーション内部の入力ストリームを分析し、結果を出力ストリームへ出力
- アプリケーション内部ストリームの最大行サイズは 50 KB/参照データソースの最大サイズは 1 GB
- クエリーの複雑さとデータのスループットに応じて処理能力 (KPU – Kinesis Processing Units) を自動伸縮
- 米国東部（バージニア北部）/米国西部（オレゴン）/欧州（アイルランド）リージョンで利用可能

本日はAnalyticsのハンズオン

資料参考：
http://www.slideshare.net/AmazonWebServicesJapan/amazon-kinesis-analytics

Analyticsの構築フロー

①入力側を決定する（Streams or firehose）

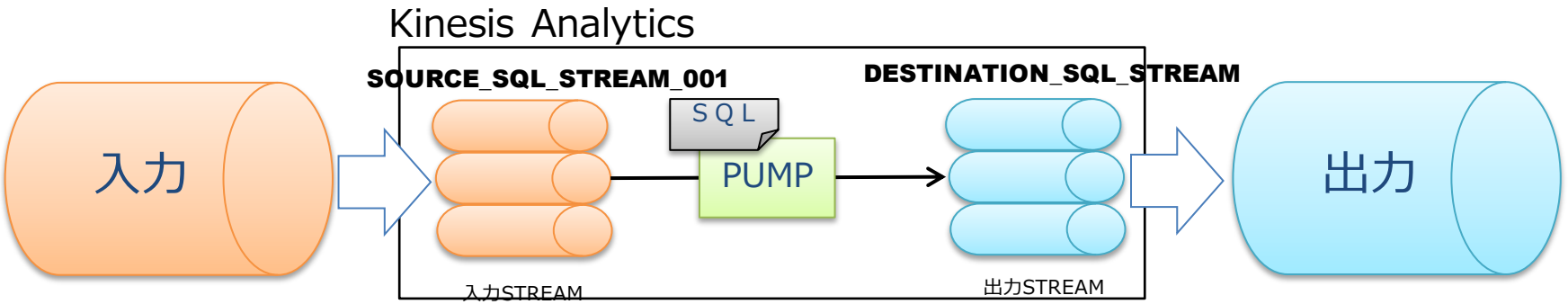
②入力データの型定義を行う

Streams
Demoを使用する

③SQL文を作成

④SQL文をデプロイ

⑤出力先を決定する（Streams or firehose）



ハンズオン

- Step1. Kinesis Analyticsを立てる
- Step2. 基本的なSQLクエリ
  - CREATE STREAM
  - SELECT
  - WHERE
  - INNER JOIN(Reference Data 追加)
- Step3. windowを使用したSQLクエリ
  - GROUP BY
- Step4. 高度な分析（Distinct Count）
- Step5. Firehose →ELS → Kibanaで可視化

赤文字：ハンズオン対象

ハンズオンをする前のお願い

・結構シーンとなるのでその際はご質問ください

　- コードデプロイで時間が要する場合がございます

・エラーが発生すると思うので、その際はコピー&ペーストしてください

・時々、詳細不明なエラーが発生します

　- 1アカウントで5以上のanalyticsを立てると、kinesis demo streams からデータを取得できなくなる

　- kinesis demo streams は良く停止する

Step#1
Kinesis Analyticsを立てる

Step#2
基本的なクエリ

CREATE STREAM & SELECT

```
CREATE OR REPLACE STREAM “DESTINATION_SQL_STREAM”
( TiCKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(12),
   CHANGE REAL,
   PRICE REAL);

-- Create pump to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT
INTO "DESTINATION_SQL_STREAM"
SELECT
    STREAM ticker_symbol,
           sector,
           change,
           price
FROM "SOURCE_SQL_STREAM_001";
```

Multi Streamsの場合、
列名およびテーブル名は大文字が良い

「”」で囲わない場合、
大文字になるので注意。

WHERE

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
( TiCKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(12),
   CHANGE REAL,
   PRICE REAL);

-- Create pump to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO
"DESTINATION_SQL_STREAM"
SELECT
    STREAM ticker_symbol,
           sector,
           change,
           price
FROM "SOURCE_SQL_STREAM_001"
WHERE change > 0.0
```

INNER JOIN

- Reference Dataの追加はAWS CLIのみ
- 参照可能なReference Dataの数は１つ
  - 詳細はQiitaに記載

Create StreamとInsert文は省略

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO
"DESTINATION_SQL_STREAM"
  SELECT
        STREAM ticker_symbol
                , C."companyname“
                , sector
                , change
                , price
  FROM "SOURCE_SQL_STREAM_001"
  Inner Join "companyname" as C
  ON "SOURCE_SQL_STREAM_001".ticker_symbol = C."com";
```

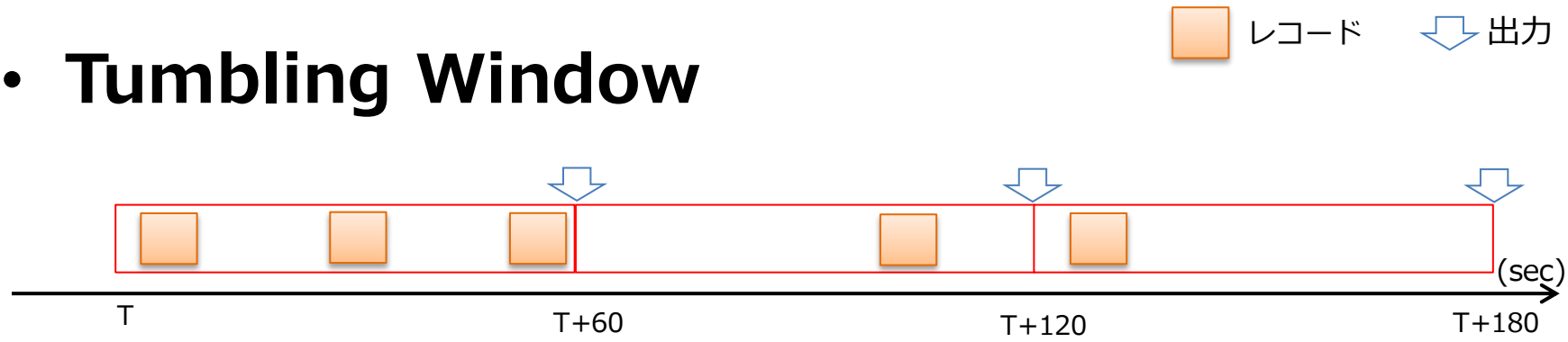
Reference Data

Reference Dataを追加すると、[Could not find Stream]とエラーが発生する？

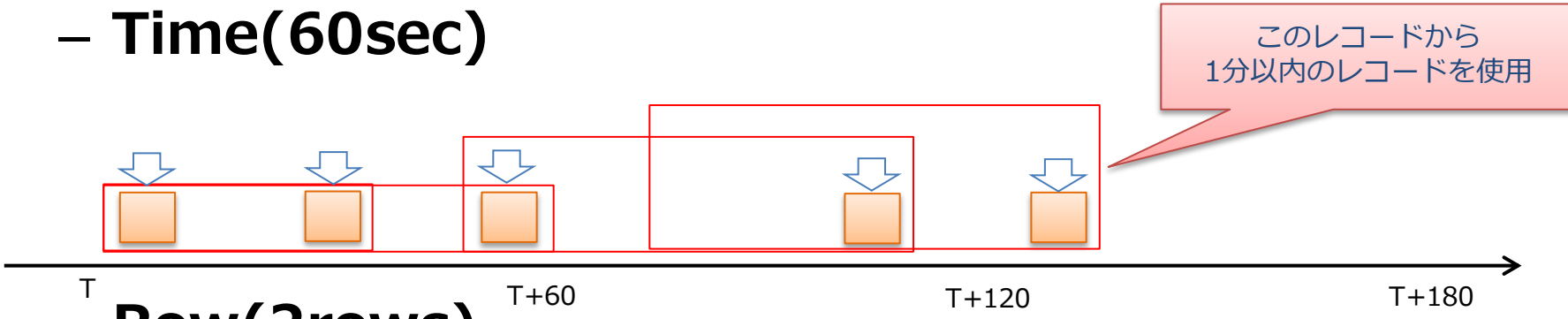
Step#3.
windowを使用したクエリ

WINDOWの概念

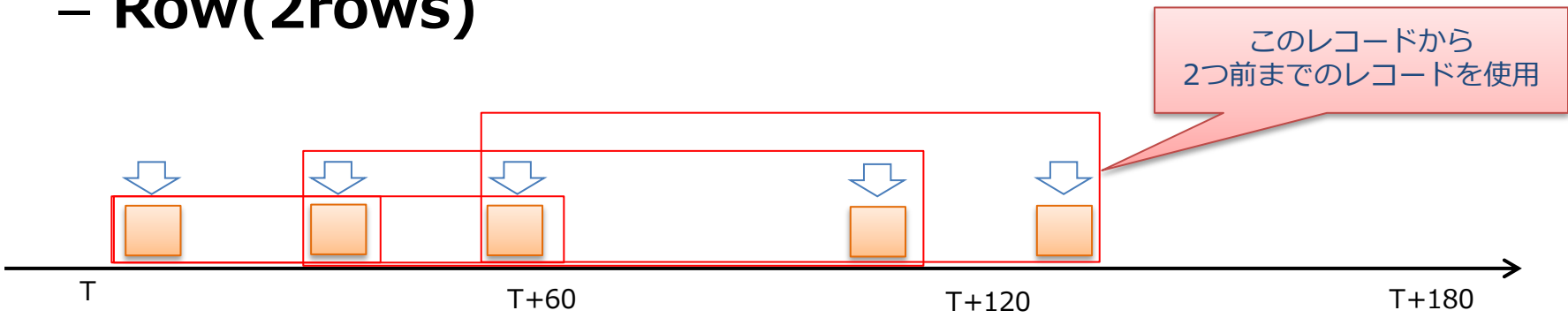
- **Tumbling Window**



- **Sliding Window**
  - **Time(60sec)**



  - **Row(2rows)**



GROUP BY句

Tumbling Window

Create StreamとInsert文は省略

```
SELECT STREAM
        ticker_symbol
        ,avg(price) AS ticker_symbol_avg
FROM “SOURCE_SQL_STREAM_001“ as A
GROUP BY ticker_symbol,
FLOOR((A.ROWTIME - TIMESTAMP '1970-01-01 00:00:00') SECOND
/ 10 TO SECOND);
```

Sliding Window

書き方が異なるので注意

```
SELECT STREAM
        ticker_symbol
        ,avg(price) OVER TTLW (or TRSW) AS ticker_symbol_avg
FROM “SOURCE_SQL_STREAM_001"
WINDOW TTSW AS ( PARTITION BY ticker_symbol  RANGE
INTERVAL '10' SECOND PRECEDING);
```

OR

```
WINDOW TRSW AS (PARTITION BY ticker_symbol ROWS 2
PRECEDING);
```

Step#4.
高度な分析
(Distinct Count)

Distinct Count

- 高度な分析(top-k等)は構文が異なるので注意
- CURSOR関数内で入力対象Streamを指定し、残りは関数毎の異なるオプションを記載する

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM(
        NUMBER_OF_DISTINCT_ITEMS BIGINT
);

CREATE OR REPLACE PUMP STREAM_PUMP AS INSERT INTO
DESTINATION_SQL_STREAM
SELECT
        STREAM NUMBER_OF_DISTINCT_ITEMS
FROM TABLE(COUNT_DISTINCT_ITEMS_TUMBLING(
  CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
  'TICKER_SYMBOL',    -- カウント対象の列
10   – Tumbling window size)
);
```

Step#5
Kibanaで可視化

Kibanaで可視化

- **Analyticsで作成したデータをKibanaで可視化します**
- **Firehose -> Elastic Search -> Kibanaとすべて AWSコンソールで設定可能**
  - **ELSは立ち上がるのに時間が掛かります**
- **こちらはハンズオンではなく、画面をご覧ください**

現時点でのAnalyticsへの評価

- **Good**
  - **構築は非常に楽でテンプレートも参考になる**
  - **標準SQLのため簡単?**
  - **Kinesis FirehoseでS3、Redshift、Elastics Searchとも簡単に接続可能**

- **Bad**
  - **Kinesis StreamsにKPLで入力したデータはロード不可**
  - **バグが多い（Referenceデータを追加等)**
  - **KPUにキャップ機能がない**
  - **性能評価がしにくい（入出力レコード数が不明）**

所感

- 構築が非常に楽
- 非常にSQLで馴染みやすいようで、WINDOWの概念や大文字小文字でのエラーにかなり戸惑う
- Streams、Analyticsともにフルマネージドという点にはかなり魅力的
  - Kafkaで言うと、zookeeperの管理が無くなる
  - Kinesis StreamsはKPLと併用すれば非常に安い
  - Analyticsは使いやすい？と疑問に思う点も。

- 簡単な集計処理ならよいが複雑な処理はSpark Streaming等を使用したほうがよい

その他情報

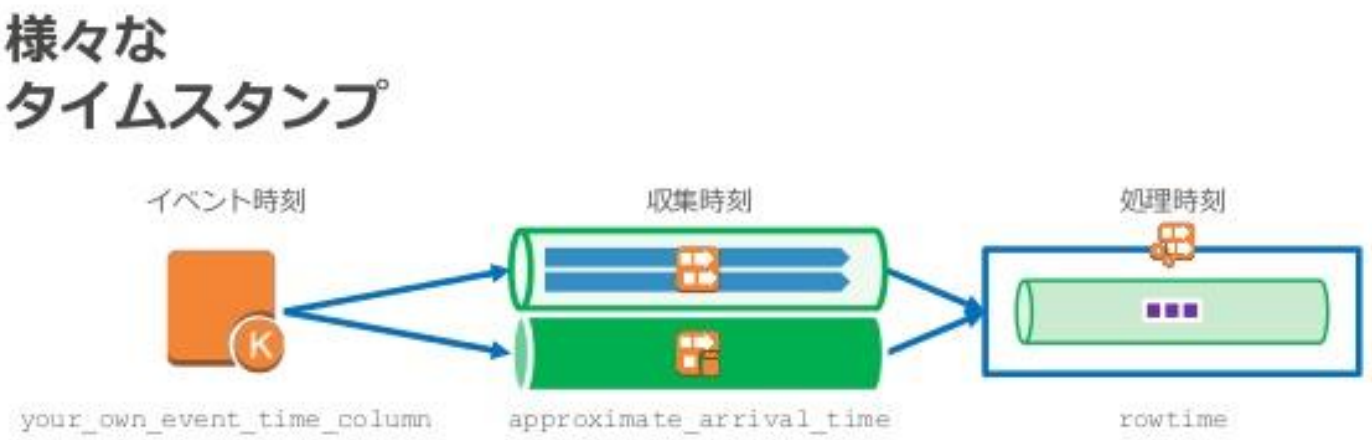
- **Kinesis Analytics以外でも、ストリーム処理についても議論ができればと思っていますので、是非お声掛けください**

- **本日の内容やKinesis Streamsへのデータ投入方法等をQiitaで紹介していきます**
  - **http://qiita.com/takada_tf/items/7c3898bb63d5d4ffc480**
  - **http://qiita.com/takada_tf/items/f03c36eed9e22eb74744**

参考

時間軸

- Kinesis analyticsには3種類の時間軸が用意されている
  - Your_own_event_time_columns:ログ発生時刻
  - Approximate_arrival_time:収集時刻
  - Rowtime:処理時刻
- 処理性能を計測したい場合、これら時間軸を使用する




```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
  SELECT STREAM
    your_own_event_time_column,
    approximate_arrival_time,
    rowtime
  FROM "SOURCE_SQL_STREAM_001";
```