GMOペパボ

機械学習研修
Day 1

author = ‘Toshifumi Tsutsumi’ AND
presented_at = ‘2021-07-15’

堤 利史
Twitter: @tosh2230

データエンジニア

GMO Pepabo, inc.
技術部 データ基盤チーム
(2020/12~)

機械学習研修: 前半2日間のゴール

- 大規模データを ETL (Extract, Transform, Load)する方法を知る

- データウェアハウスの使い方やETLの基本的な考え方に慣れて、自分が利用するデータを抽出・加工できるようになる

NOTE:
演習では、一般社団法人 データサイエンティスト協会が提供する
“データサイエンス100本ノック（構造化データ加工編）” のデータを BigQuery にロードしています
https://github.com/The-Japan-DataScientist-Society/100knocks-preprocess

機械学習研修: 前半2日間の予定

2021/7/15(木) 15:00 - 18:00

1. データエンジニアリングってなんですか？
2. Bigfoot 最速入門
3. Google BigQuery 入門
4. Google DataStudio 入門

機械学習研修: 前半2日間の予定

2021/7/16(金) 15:00 - 17:00

5. ETLってなんですか？
6. ETL Ultraquick Tutorial
7. データエンジニアリングってなんですか？

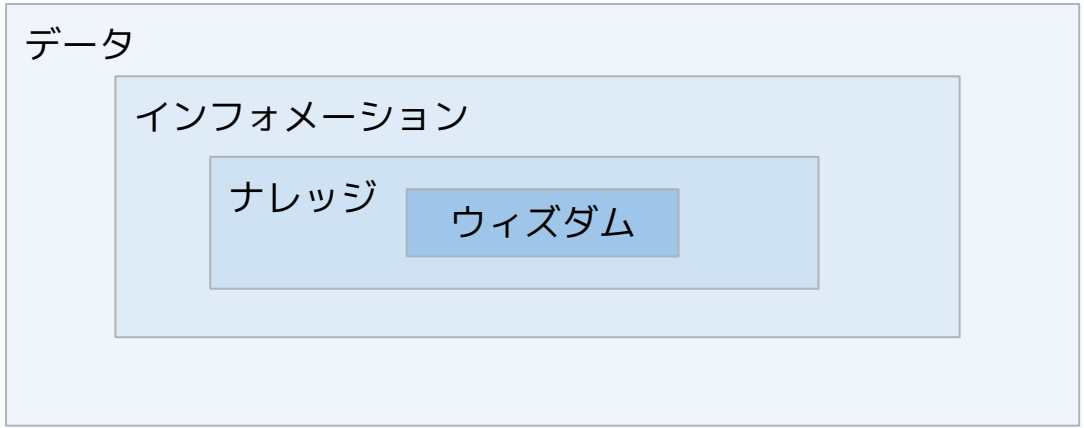
Section 1

データエンジニアリングってなんですか？

データとは

データ ⊃ インフォメーション *

- データは「インフォメーションの原材料」
- インフォメーションは「コンテキストを持ったデータ」
  - コンテキストを持った = 意味付けされている



* DAMA International (2018)「データマネジメント知識体系ガイド 第二版」 日経BP社

```
date-id-name-num0-num1-num2-num3
2021-07-15-0123456789-test0-12-34-567-890
2021-07-15-9876543210-test1-98-76-543-210
```

```
date-id-name-num0-num1-num2-num3
2021-07-15-0123456789-test0-12-34-567-890
2021-07-15-9876543210-test1-98-76-543-210
```

```
date,id,name,num0,num1,num2,num3
2021-07-15,0123456789,test0,12,34,567,890
2021-07-15,9876543210,test1,98,76,543,210
```

手段1: インフォメーションになりうるデータをつくる

- そもそも、データを集めるのはなぜか？
  - 何かを知りたいから
  - 適切な理解は適切なデータ作成から

- データをつくるにはナレッジ(=ドメイン知識)が必要
  - 目的に沿ったデータをつくることが大事
  - システムだけではなく人もデータをつくる

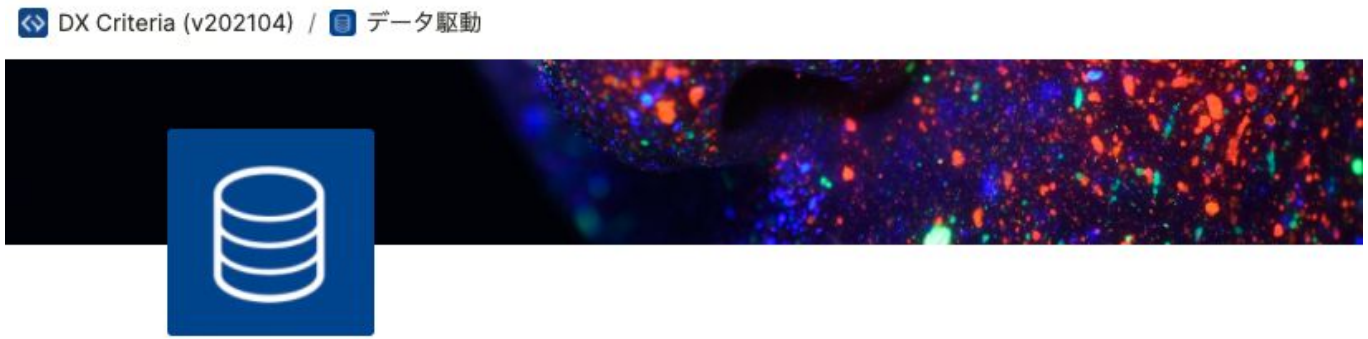
手段2: 既存のデータをインフォメーションに変換する

- コンテキストがわかりにくい場合
  - 誰もがわかる表現に加工する
    - enum(列挙型): ex, 0-6で曜日を表現
    - 信号処理
  - 別のデータとつなぎ合わせる
    - ユーザー情報 * 注文情報
  - メタデータ(データを説明するデータ)を追加する
    - 5W1H
    - 表現したいこと
    - 大元の発生源

手段2: 既存のデータをインフォメーションに変換する

- コンテキスト自体をまだ見いだせていない場合
  - データの中から、隠れたコンテキストを探す
  - 暗黙知を形式知に
    - 統計
    - パターン認識
    - 機械学習

インフォメーションをもとに行動を起こす



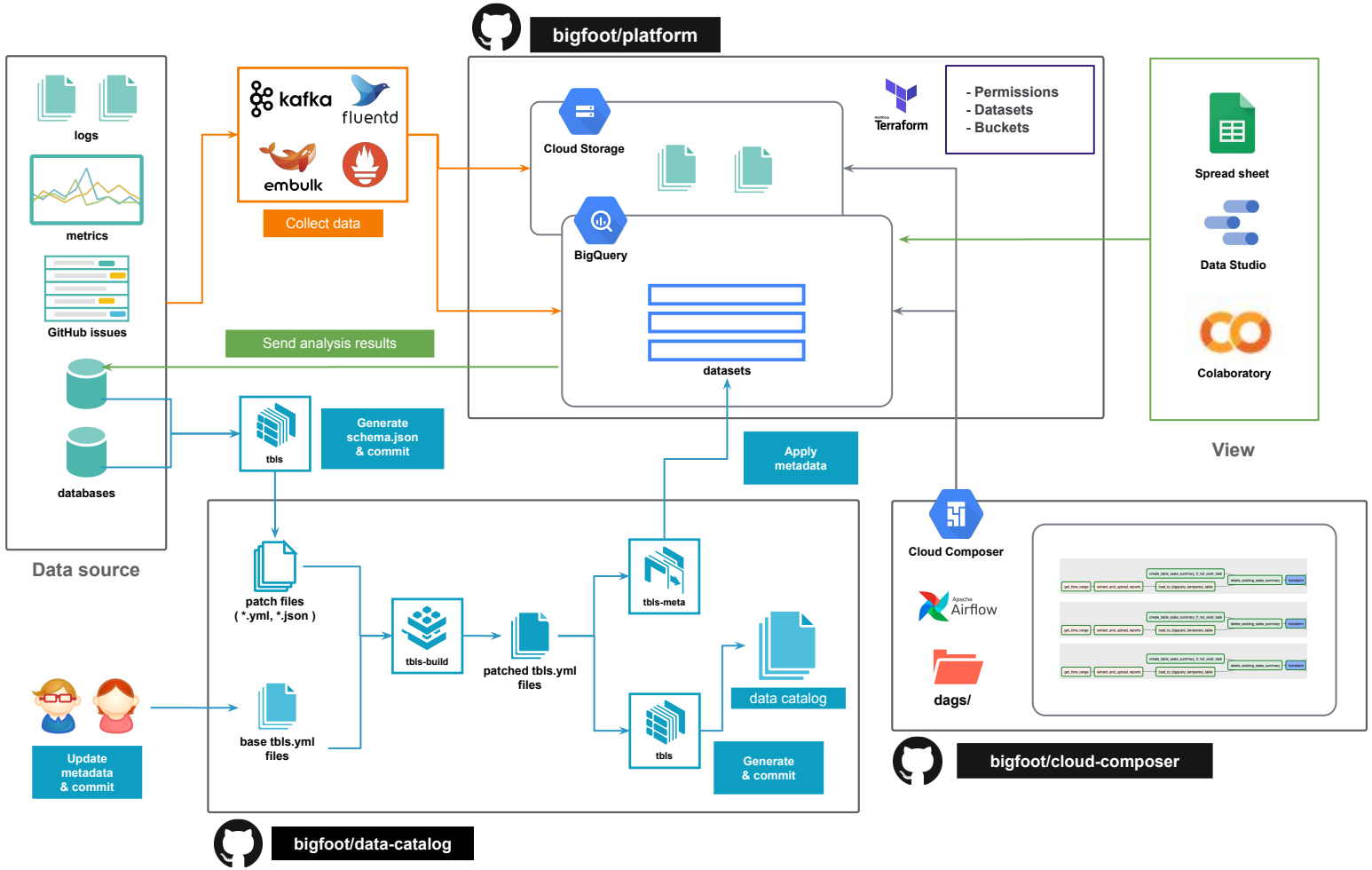
https://dxcriteria.cto-a.org/d44c9631ebfb4c8f8ff1ae69361dfa65

Section 2

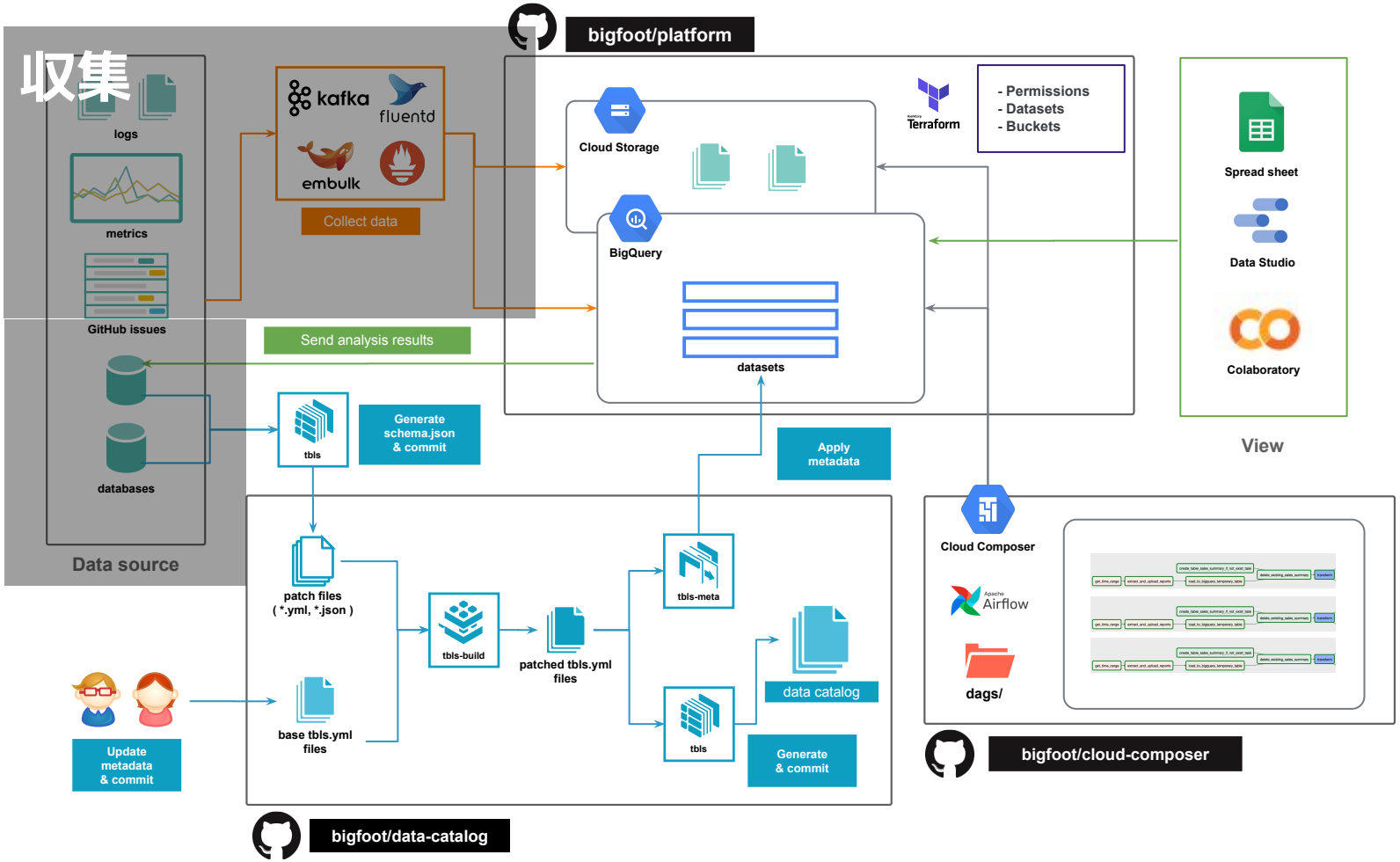
Bigfoot 最速入門

ペパボのデータ利活用基盤をご紹介

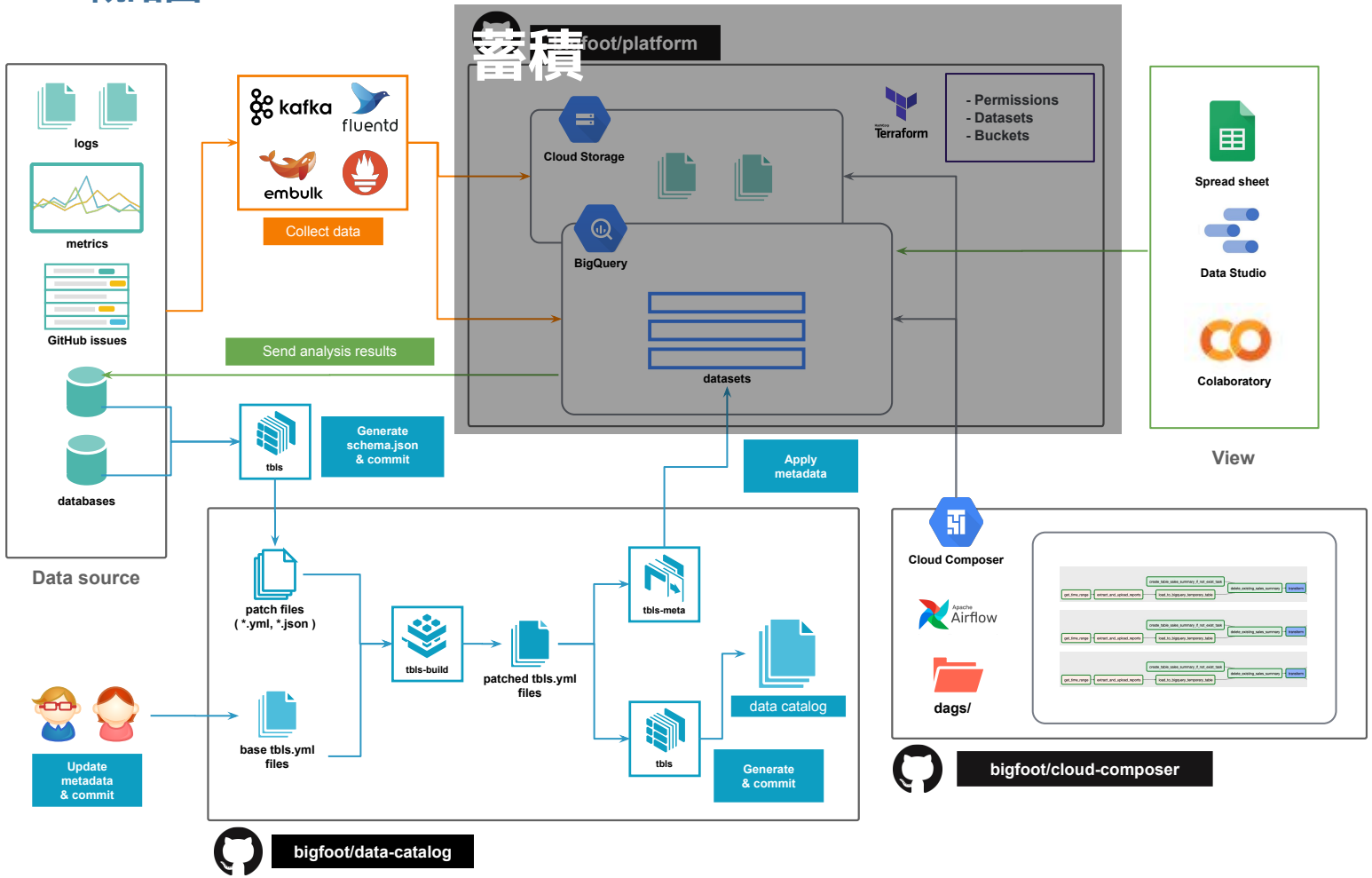
Bigfoot 概略図


bigfoot/platform
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
Spread sheet
Data Studio
Colaboratory
View
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
base tbls.yml files
tbls-build
patched tbls.yml files
tbls-meta
tbls
data catalog
Generate & commit
Update metadata & commit
bigfoot/data-catalog
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

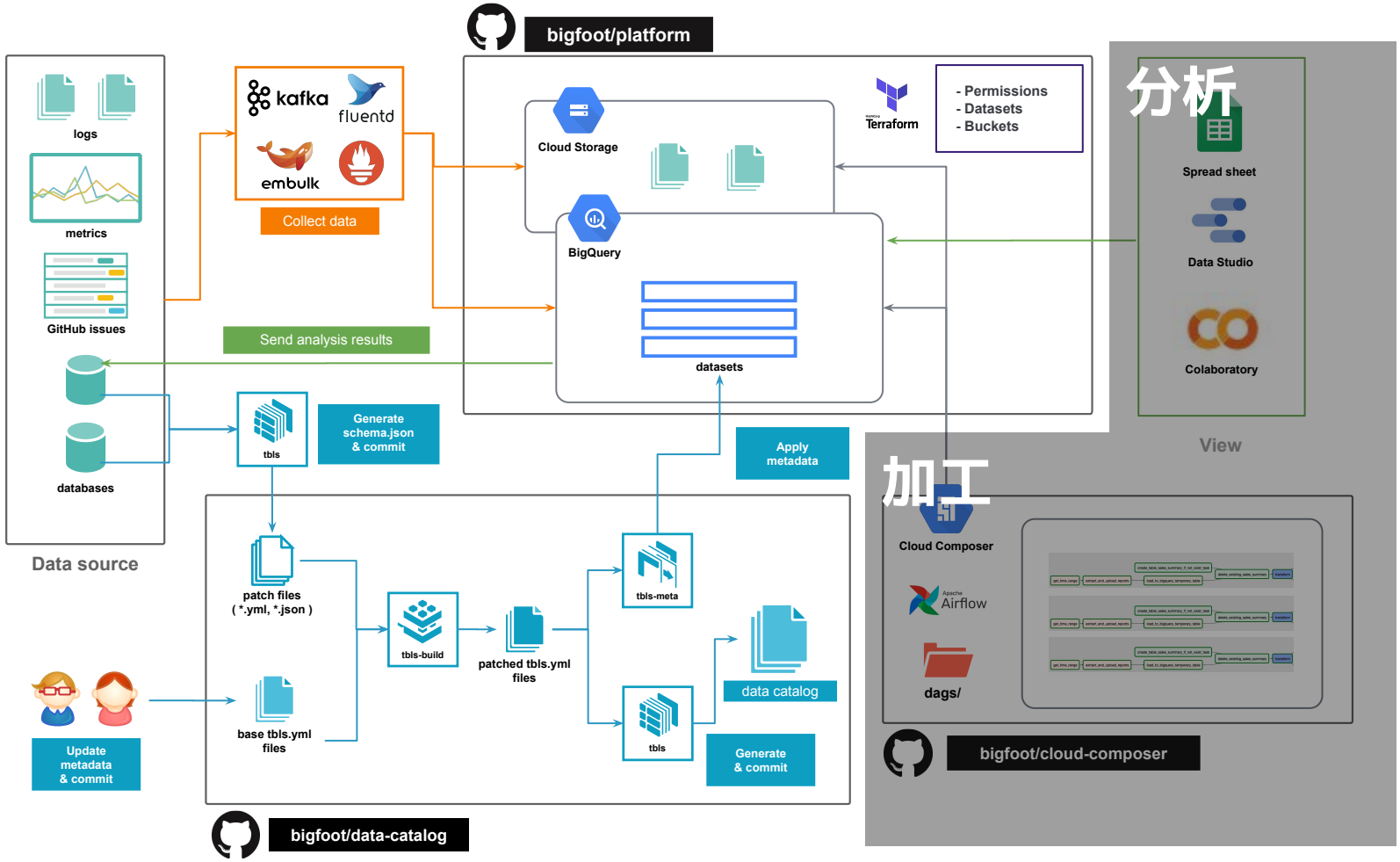
Bigfoot 概略図


収集
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
bigfoot/platform
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
Spread sheet
Data Studio
Colaboratory
View
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
tbls-build
patched tbls.yml files
tbls-meta
tbls
data catalog
Generate & commit
base tbls.yml files
Update metadata & commit
bigfoot/data-catalog
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

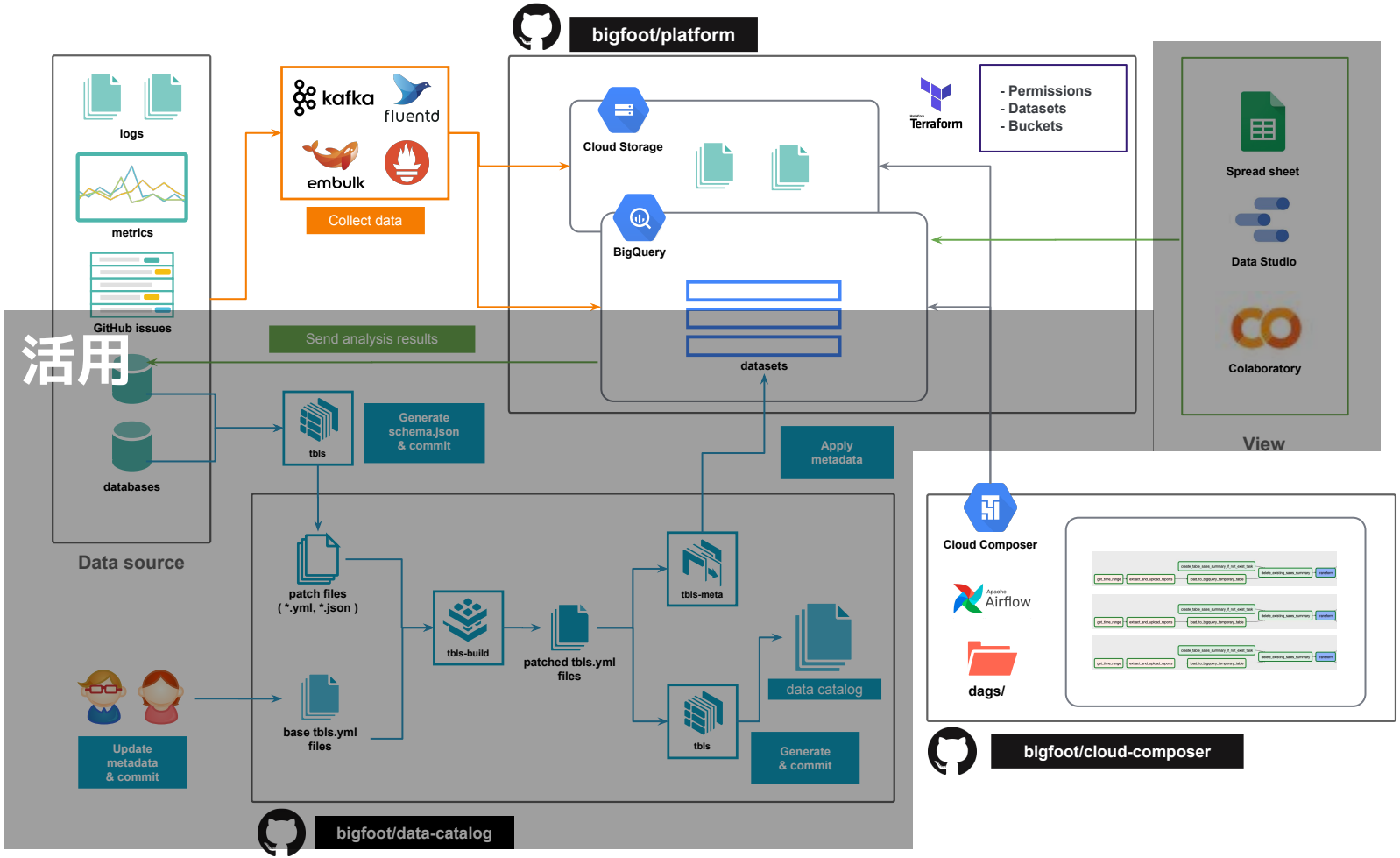
Bigfoot 概略図


蓄積
foot/platform
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
Spread sheet
Data Studio
Colaboratory
View
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
base tbls.yml files
tbls-build
patched tbls.yml files
tbls-meta
tbls
data catalog
Generate & commit
Update metadata & commit
bigfoot/data-catalog
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

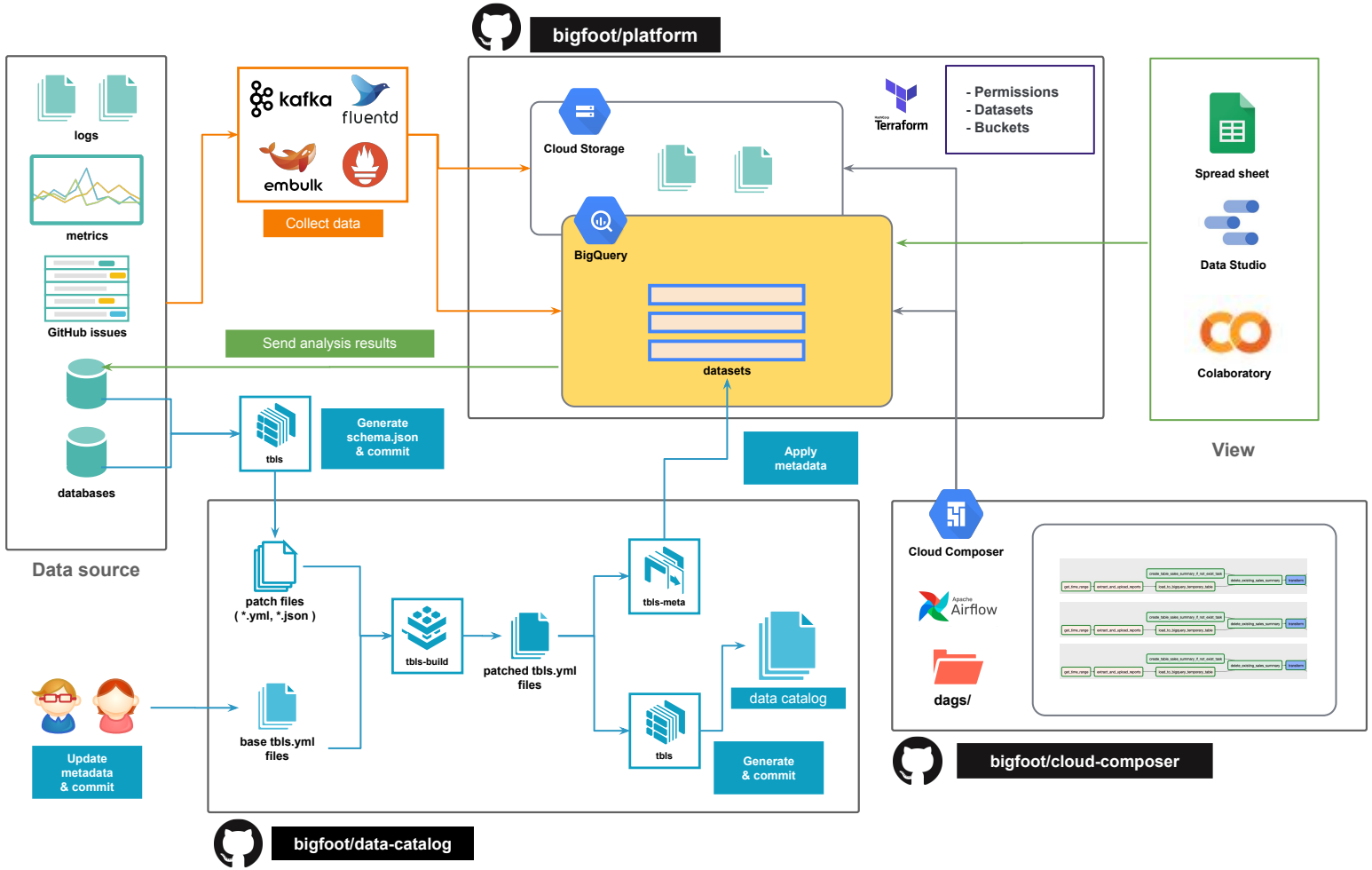
Bigfoot 概略図


bigfoot/platform
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
分析
Spread sheet
Data Studio
Colaboratory
View
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
base tbls.yml files
tbls-build
patched tbls.yml files
tbls-meta
tbls
data catalog
Generate & commit
Update metadata & commit
bigfoot/data-catalog
加工
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

Bigfoot 概略図


bigfoot/platform
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
Spread sheet
Data Studio
Colaboratory
View
活用
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
tbls-build
patched tbls.yml files
tbls-meta
data catalog
tbls
Generate & commit
base tbls.yml files
Update metadata & commit
bigfoot/data-catalog
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

Bigfoot 概略図


bigfoot/platform
logs
metrics
GitHub issues
databases
Data source
kafka
fluentd
embulk
Collect data
Cloud Storage
BigQuery
datasets
Terraform
- Permissions
- Datasets
- Buckets
Spread sheet
Data Studio
Colaboratory
View
Send analysis results
tbls
Generate schema.json & commit
Apply metadata
patch files (*.yml, *.json)
base tbls.yml files
tbls-build
patched tbls.yml files
tbls-meta
tbls
data catalog
Generate & commit
Update metadata & commit
bigfoot/data-catalog
Cloud Composer
Apache Airflow
dags/
bigfoot/cloud-composer

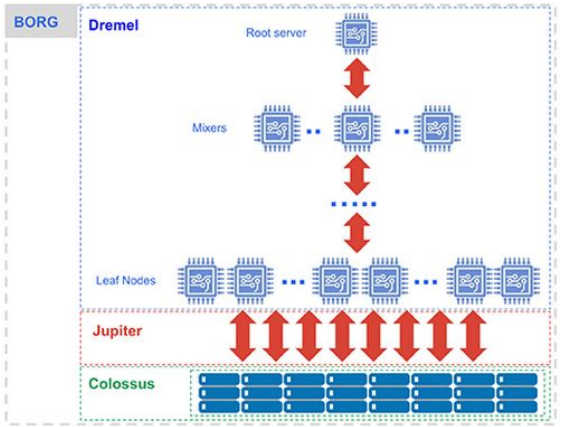
Section 3

Google BigQuery 入門

Google BigQuery

大規模データ分析対応のエンタープライズ向け
フルマネージドデータウェアハウス

- コンピュート、ストレージ、メモリがそれぞれ分離している構造
- それぞれ自動でスケールアウトしてくれるので、
  - データの保存量は事実上無制限
  - クエリ実行速度が超速(特に集計処理)



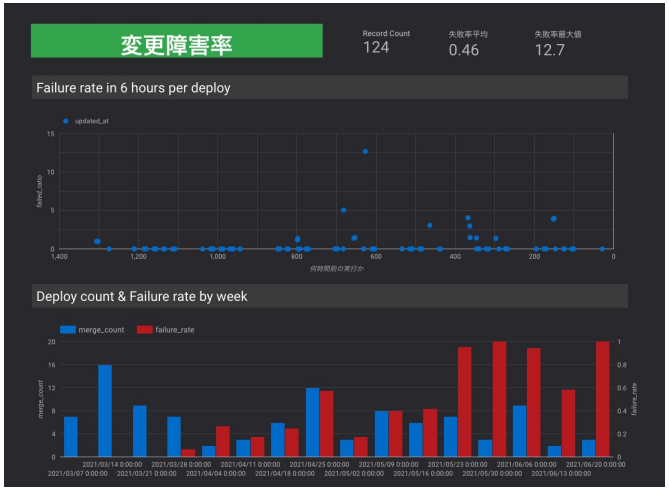
https://cloud.google.com/blog/products/data-analytics/new-blog-series-bigquery-explained-overview

Google BigQuery: インターフェース

- Google Data Studio
- Google Sheets
- BigQuery Web Console
- Google Apps Script
- Google Colaboratory
- bq command-line tool
- API Client Libraries

  C#, Go, Java, Node.js, PHP,

  Python, Ruby
- REST API




```ruby
require "google/cloud/bigquery"

def query
  bigquery = Google::Cloud::Bigquery.new
  sql = "SELECT name FROM `bigquery-public-data.usa_names.usa_1910_2013` " \
        "WHERE state = 'TX' " \
        "LIMIT 100"

  # Location must match that of the dataset(s) referenced in the query.
  results = bigquery.query sql do |config|
    config.location = "US"
  end

  results.each do |row|
    puts row.inspect
  end
end
```

Google BigQuery: コスト

GMOペパボ

- ストレージ料金
  - アクティブ ストレージ: $0.020 per GB
  - 長期保存: $0.010 per GB
  - 90日間連続で使用していない場合は、自動的に”長期保存”と判定される
  - (やさしさ)

- スキャン料金
  - 通常は 1TB スキャンで $5
  - 2021年6月より、Bigfoot はクエリ実行定額プランに移行しました
  - いまならなんと！クエリ投げ放題！お得！！1

余談: RDB と DWH の違い

GMOペパボ

RDB: 行指向

1. レコードに対する細かい操作が得意
   - トランザクション処理
   - インデックスによる行の特定

2. 列方向の集計処理効率はよくない
   - もちろん集計はできますが、リソース効率的に向いていない
   - 行データは、一定の大きさのブロックとしてファイルストレージに保存される
   - 抽出対象の行が格納されているブロックをすべて参照して、目的の行を探す
   - たとえ1列だけ欲しいとしても、構造的に全列を走査する

余談: RDB と DWH の違い

DWH: 列指向(が多い)

1. 特定列に対する集計処理が得意
   - 例: 1億行・10列のテーブルから、列Aの平均を算出
   - 列ごとにデータを保存しているので、列Aのみ走査する

2. Primary Key, Foreign Key の概念がない
   - 行の一意性はテーブル設計者が担保
   - テーブル結合はできる

余談: RDB と DWH の違い

GMOペパボ

DWH: 列指向(が多い)

3. “あえて” 非正規化して保存することが多い

- そもそも、なぜ RDB では正規化するのか？
  - Insert, Update, Deleteで発生する更新時異状(Update anomaly)を防ぐため
- 可視化用/分析用に加工したデータを、行レベルで更新することは基本的にない
  - データ加工については、Day2でお話しします
- ひとつのテーブルとして保存しているほうが、使う人にとってわかりやすい
- そしてテーブル結合が不要なので、(一般的に)クエリ速度は速くなる

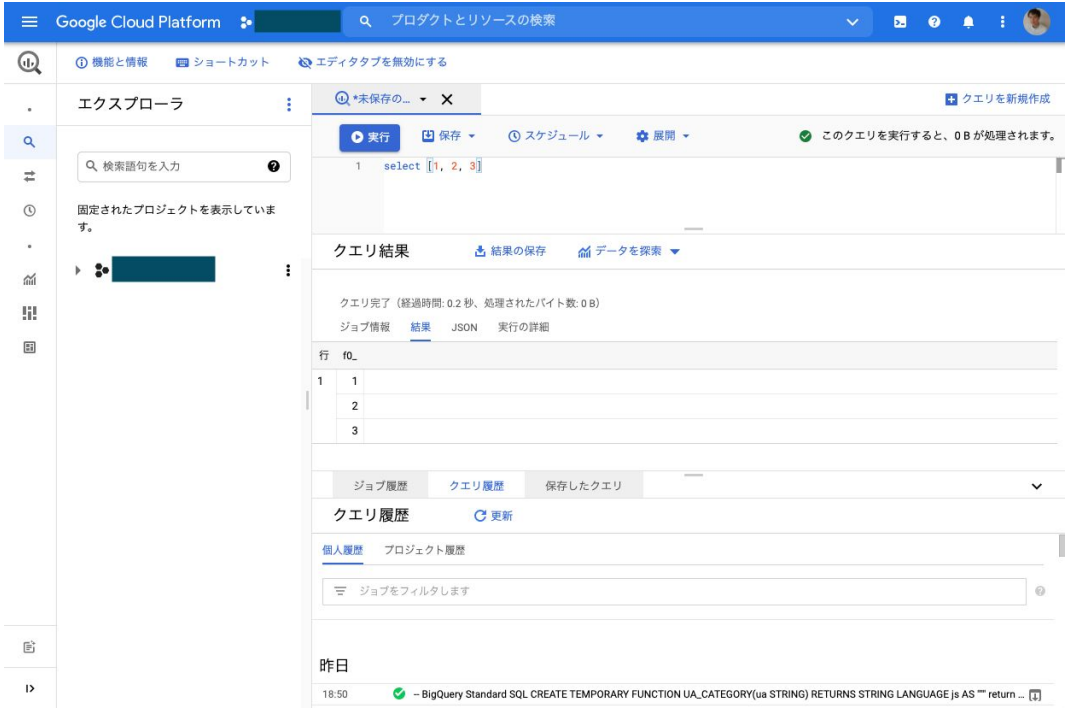
ではそろそろ実物を...

BigQuery SQLワークスペース

https://console.cloud.google.com/bigquery

エクスプローラ

プロジェクト
└ データセット
　└ テーブル
　　ビュー



タブ領域

- クエリエディタ
- クエリスキャン見積
- データセット情報参照
- テーブル情報参照

クエリ結果

- クエリ実行結果の表示
- クエリ実行結果の保存
- ジョブ実行情報

履歴関連

- ジョブ実行履歴
- クエリ実行履歴
- 過去に保存したクエリ

エクスプローラ

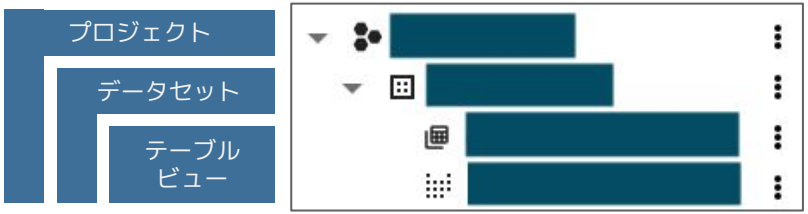
GMOペパボ

プロジェクト

- GCPのリソースやコストの管理をするためのグループ

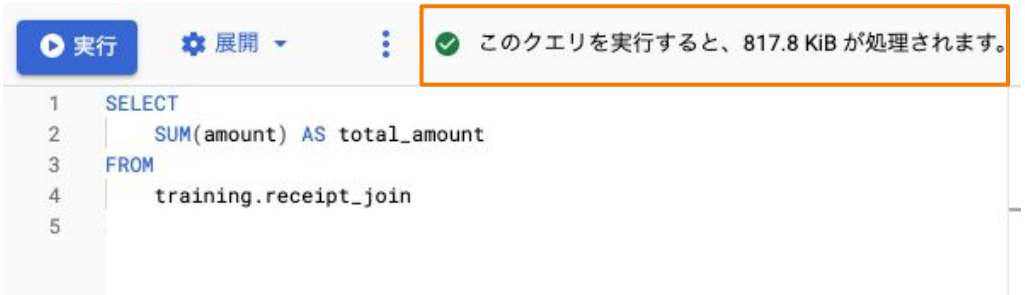
データセット

- テーブルやビューをグループ化する概念
- Bigfootでは、サービス単位で作成している
- データセットでアクセス権限を管理



列を指定するとスキャン量が減る → 速

- 全ての列を指定すると
  表全体をスキャン
- 特定の列を指定すると
  その列のみをスキャン



列を指定するとスキャン量が減る → 速

以下のクエリを実行してみてください(画面右上のスキャン量に注目)

1. `SELECT * FROM training.receipt_join`

2. `SELECT receipt_no FROM training.receipt_join`

3. `SELECT receipt_no FROM training.receipt_join LIMIT 100`

ARRAY型

- 公式の説明:

  “ARRAY 型ではないゼロ以上の要素の順序付きリスト。”

  トートロジー...

- 配列をそのままフィールドへ挿入できる
- 配列内の値は同一の型でなければならない

```
SELECT 'a' AS char, [1, 2, 3] AS nums
```

ジョブ情報 結果 JSON 実行の詳細

行	char	nums
1	a	1
		2
		3

クエリ結果は1行

```
WITH base AS (
  SELECT 'a' AS char, [1, 2, 3] AS nums
)

SELECT char, u_nums FROM base, UNNEST(nums) AS u_nums
```

ジョブ情報 結果 JSON 実行の詳細

行	char	u_nums
1	a	1
2	a	2
3	a	3

UNNEST関数で配列を展開してCROSS JOIN
クエリ結果は3行

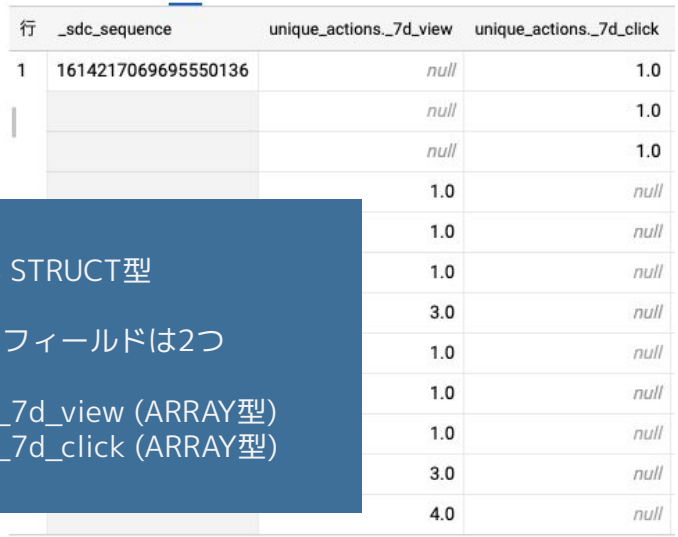
STRUCT型

- 公式の説明:

  “ 順序付きフィールドのコンテナ。
  各フィールドはデータ型（必須）とフィールド名（オプション）を持ちます。”

- 子テーブルが列の中に存在するイメージ

- 1:N の関係にあるテーブル同士が、
  他の列をキーとしてあらかじめ
  結合された状態



行	_sdc_sequence	unique_actions._7d_view	unique_actions._7d_click
1	1614217069695550136	null	1.0
		null	1.0
		null	1.0
		1.0	null
		1.0	null
		1.0	null
		3.0	null
		1.0	null
		1.0	null
		1.0	null
		3.0	null
		4.0	null

この例では、
unique_actions が STRUCT型

unique_actions のフィールドは2つ

- unique_actions._7d_view (ARRAY型)
- unique_actions._7d_click (ARRAY型)

SQL演習(1)

training.receipt_join テーブル (receipt テーブルをベースに非正規化したもの) を使って計算してみましょう

列情報は テーブルの ”スキーマ” タブを参照ください

1. customer_id = 'CS029512000063' の方がこれまで購入した金額は
   合計でいくらでしょうか？

2. category_small_name = 'その他駄菓子' の商品を販売した実績のある店舗は
   いくつあるでしょうか？

SQL演習(1) 解答

training.receipt_join テーブル (receipt テーブルをベースに非正規化したもの) を使って計算してみましょう

列情報は テーブルの ”スキーマ” タブを参照ください

1. customer_id = 'CS029512000063' の方がこれまで購入した金額は合計でいくらでしょうか？

行	total_amount
1	776

```
SELECT
  SUM(amount) AS total_amount
FROM
  training.receipt_join
WHERE
  customer.customer_id = 'CS029512000063'
```

SQL演習(1) 解答

training.receipt_join テーブル を使って計算してみましょう

列情報は テーブルの ”スキーマ” タブを参照ください

2. category_small_name = 'その他駄菓子' の商品を販売した実績のある店舗は
いくつあるでしょうか？

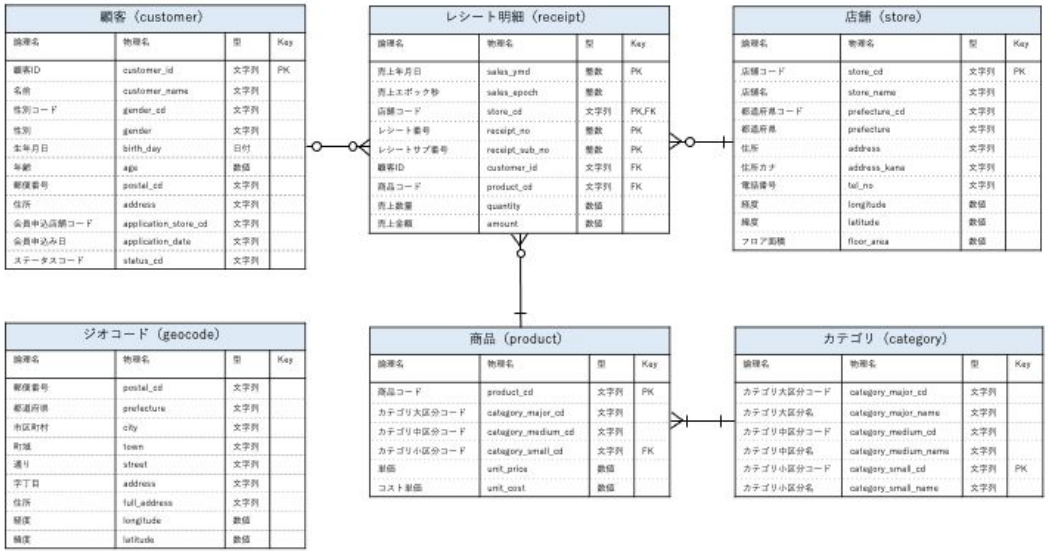
行	unique_store_count
1	18

```
SELECT
  COUNT(DISTINCT store.store_cd) AS unique_store_count
FROM
  training.receipt_join
WHERE
  product.category.category_small_name = 'その他駄菓子'
```

SQL演習(2)

training.receipt_join テーブル と同じデータを取得できる SELECT 文を書いてみてください。

training.receipt_join テーブル は、下のER図のリレーションをもとに結合しています。



https://github.com/The-Japan-DataScientist-Society/100knocks-preprocess/blob/master/docker/doc/100knocks_ER.pdf

SQL演習(2) 解答

training.receipt_join テーブル と同じデータを取得できる SELECT 文を書いてみてください。

```
WITH product_with_category AS (
    SELECT
        product.* EXCEPT(category_major_cd, category_medium_cd, category_small_cd),
        category,
    FROM
        training.product AS product
        LEFT OUTER JOIN training.category AS category
            USING (category_major_cd, category_medium_cd, category_small_cd)
)

SELECT
    receipt.* EXCEPT (customer_id, product_cd, store_cd),
    customer,
    store,
    product
FROM
    training.receipt AS receipt
    LEFT OUTER JOIN training.customer AS customer USING (customer_id)
    LEFT OUTER JOIN training.store AS store USING (store_cd)
    LEFT OUTER JOIN product_with_category AS product USING (product_cd)
```

Section 4

Google DataStudio 入門

Data Studio とは？

- Google マーケティングプラットフォーム で提供されているBIサービス
- Google アナリティクスも、Google マーケティングプラットフォームのサービスのひとつ
- 日本では「データポータル」と呼ばれていますが、ここでの表記は Data Studio に統一します。

どうやってアクセスするの？

こちらへ

https://datastudio.google.com/

ダッシュボードの管理

- 自分が作成したものや共有されているものが表示される
- Google Drive に似ているが、別管理になっている
- 個人の所有物扱いなので、他の人へ見せるには共有の設定をする

新規作成で表示される編集画面

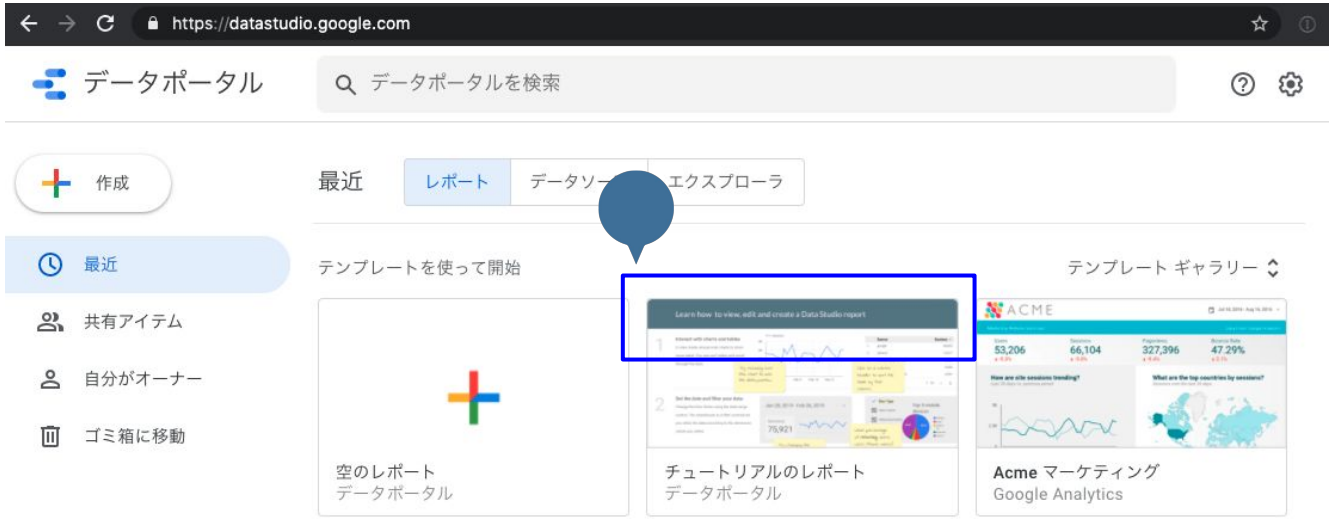
用語解説

- ページ
  - グラフや画像、コントロールを載せる場所。
  - スプレッドシートでいうシートにあたる。
- レポート
  - ページをまとめたもの。ダッシュボード=レポート。
  - スプレッドシートでいうファイル(ブック)にあたる。
- データ
  - グラフが参照するデータソース。
  - グラフにつき一つだけ指定できる。
- コントロール
  - プルダウンやチェックボックス、期間指定、などフィルタリングを行うためにレポートに設置する部品。

実際につくってみましょう

公式のヘルプページ の解説がわかりやすいです。

Data Studio のホーム画面に表示されているチュートリアルもおすすめです。



なんか、エリアマネージャーがグラフ作りたいらしい

GMOペパボ

1. 全店舗の売上を集計した総売上推移を、日付ごとにわかるようにしたいです。
   期間指定とか、月次・年次に切り替えられたりできるといいなあ

2. 1のグラフとは別に、もうひとつお願いします！！
   地域差があるかを知りたいので、売上の累計を都道府県別・店舗別にみたいです。

3. 全店舗行ったことないのですが、女性のお客様が多い気がする…
   2のグラフの内訳で、性別がみれるようにしたいです。

ヒント: データソースは ”training.receipt_join” だけです

1. 全店舗の売上(receipt.amount)を集計した総売上推移を、日付(receipt.sales_ymd)ごとにわかるようにしたいです。
   期間指定とか、月次・年次に切り替えられたりできるといいなあ

2. 1のグラフとは別に、もうひとつお願いします！！
   地域差があるかを知りたいので、売上(receipt.amount)の累計を都道府県(store.prefecture)別・店舗(store.store_name)別にみたいです。

3. 全店舗に行ったことはないのですが、女性のお客様が多い気がする...
   2のグラフの内訳で、性別(customer.gender)がみれるようにしたいです。

つくったグラフを
みんなで見てみましょう

ダッシュボードを社内に共有

ダッシュボードを Notion に埋め込む