

# BigQuery を中心とした オープンデータ レイクハウスと データ パイプライン

Google  
Cloud  
Next

Tokyo

Proprietary



# 唐澤 匠

Google Cloud  
Customer Engineer



# コンテンツ

**01** Google Cloud における  
オープン テーブル フォーマット

**02** データ パイプライン

データレイクとデータ パイプライン

Dataflow - Beam YAML とジョブビルダー  
継続的クエリとリバース ETL

# 01. Google Cloud における オープン テーブル フォーマット

# オープン テーブル フォーマットとは

- 概要:

- データレイクに保存されている大規模なデータセットを効率的に管理するためのオープンソースの**テーブル フォーマット**
- データレイク内の大量のファイル群を「テーブル」として扱えるようにする規格であり Apache Iceberg、Apache Hudi、Delta Lake などがある

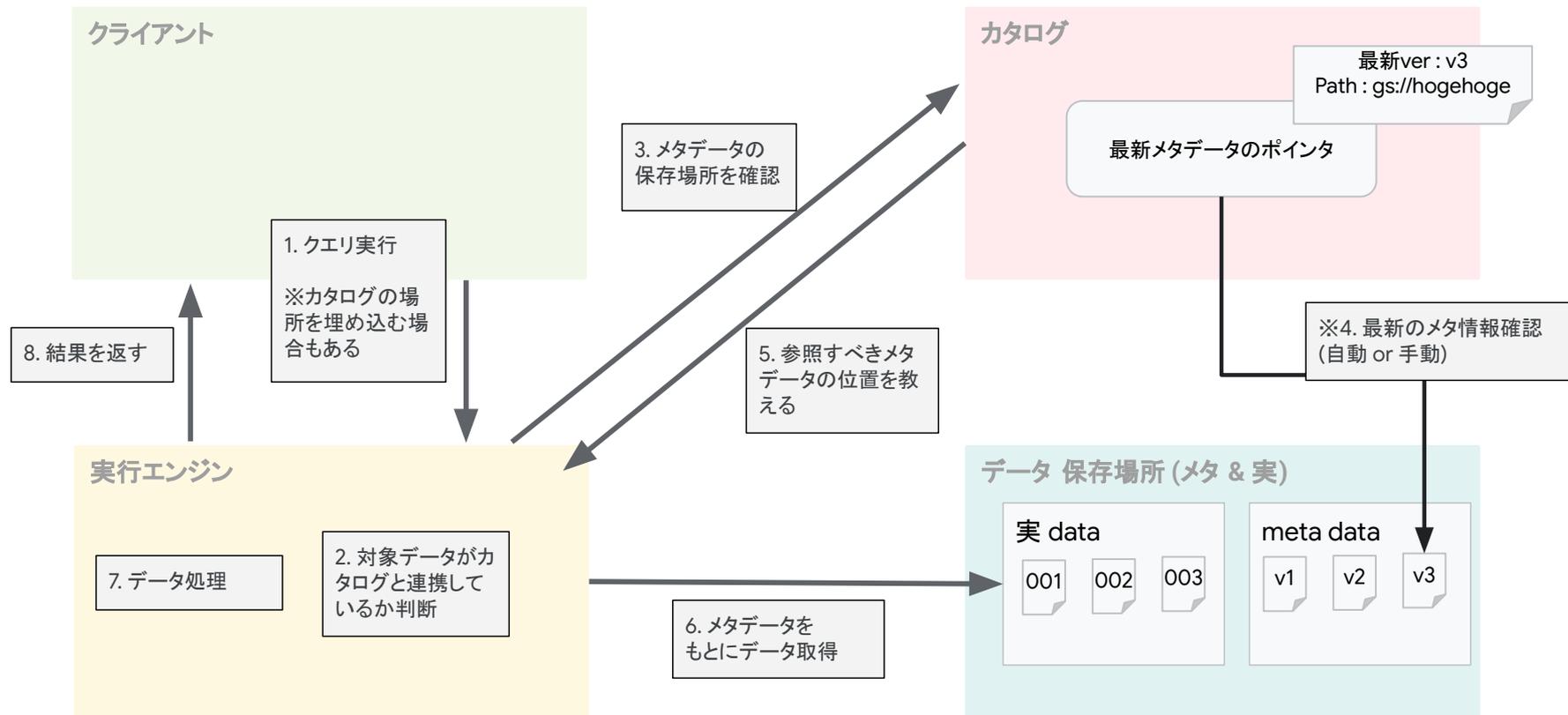
- 特徴:

- Google Cloud Storage、HDFS、他社 object storage のような分散ストレージ上に構築可能であり、大量のデータを保持しつつ、**高速なクエリを実現**
- **ACIDトランザクションをサポート**し、データの信頼性と整合性を保証
- **タイムトラベル機能の実装**により、過去の特定の時点のデータスナップショットにアクセスしたり、以前の状態にロールバックすることが可能
- 隠しパーティショニングにより、**物理的なパーティション構造を意識しない**
- Apache Spark、Apache Flink、Apache Hive、Presto、Trinoなど **様々なデータ処理エンジンと連携可能**

ICEBERG



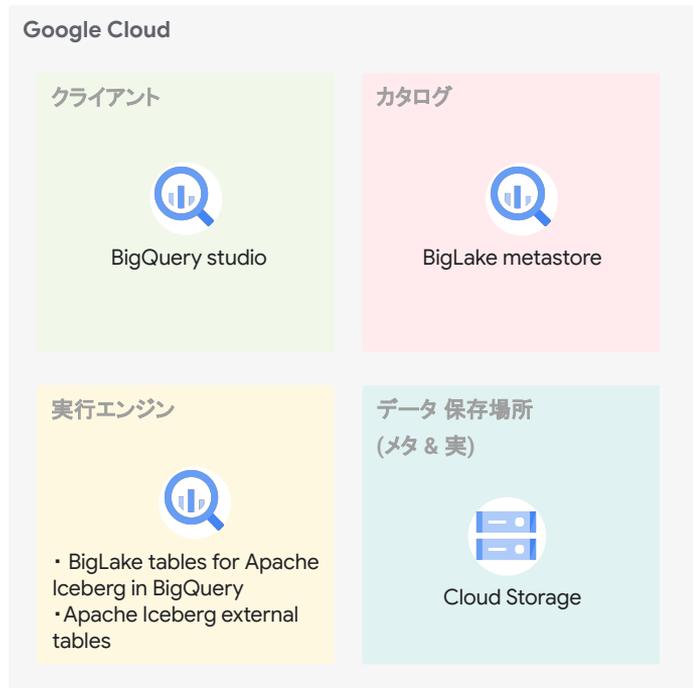
# 処理の流れ (Read)



# Google Cloud におけるコンポーネント

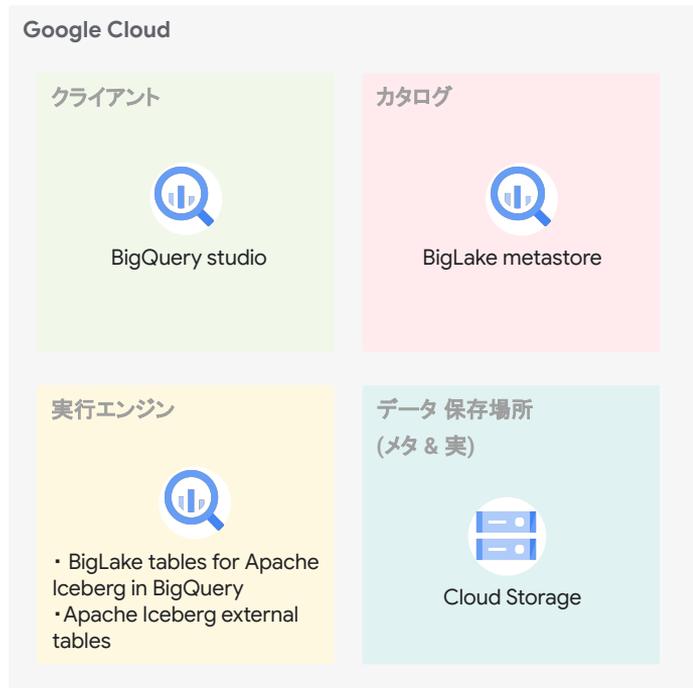
## BigQuery でオープン フォーマットを使うとき の構成

- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# Google Cloud におけるコンポーネント

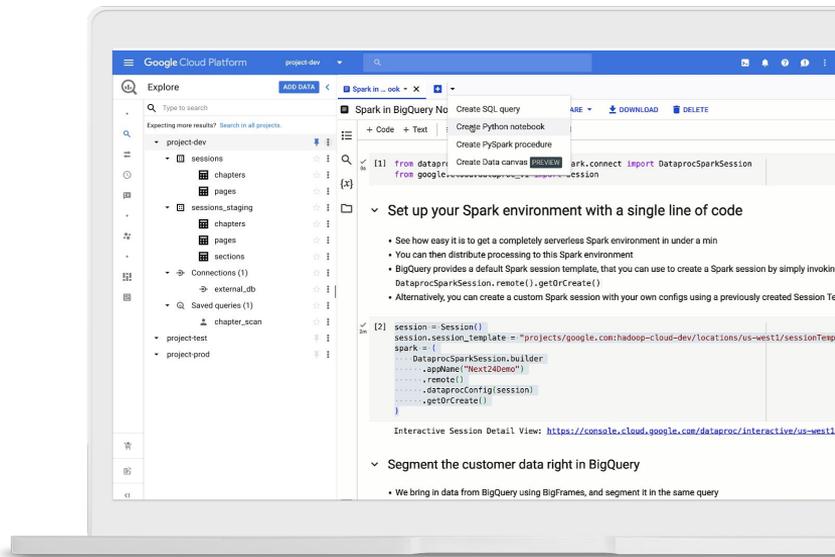
- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# BigQuery Studio

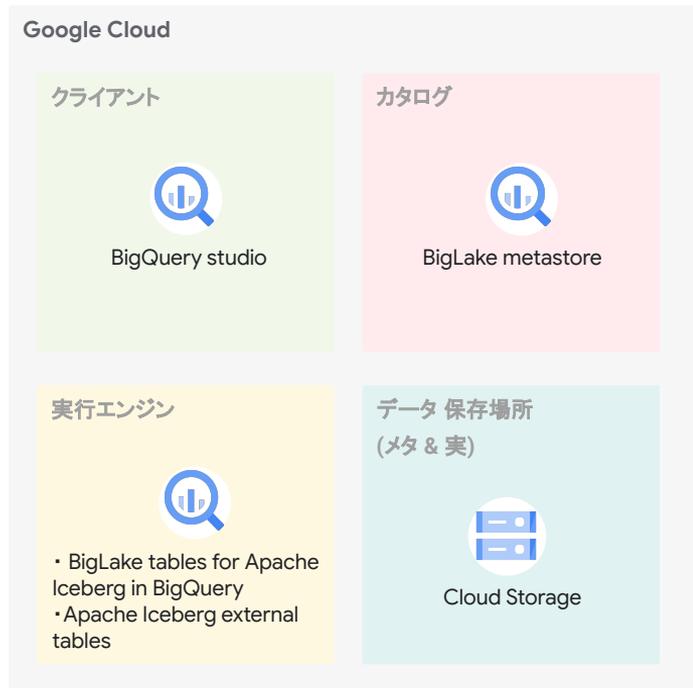
## SQL および Apache Spark の実行環境を提供

- **SQL 実行環境**
  - Gemini を利用した自然言語による SQL 生成
  - SQL 上での Gemini の呼び出し
- **BigQuery Studio Notebook**
  - データをエクスポートせずに BigQuery Studio ノートブックで Spark を簡単に記述・実行
  - pySpark Code Generation による、生成 AI を利用した自然言語による python コード生成
- **Spark スタアドプロシージャ**  
インフラ管理なしで SQL から Spark ルーチンを実行
- **統合されたセキュリティとガバナンス**  
Spark 内でリソース セキュリティ ポリシー  
行 / 列レベルのセキュリティをシームレスに適用



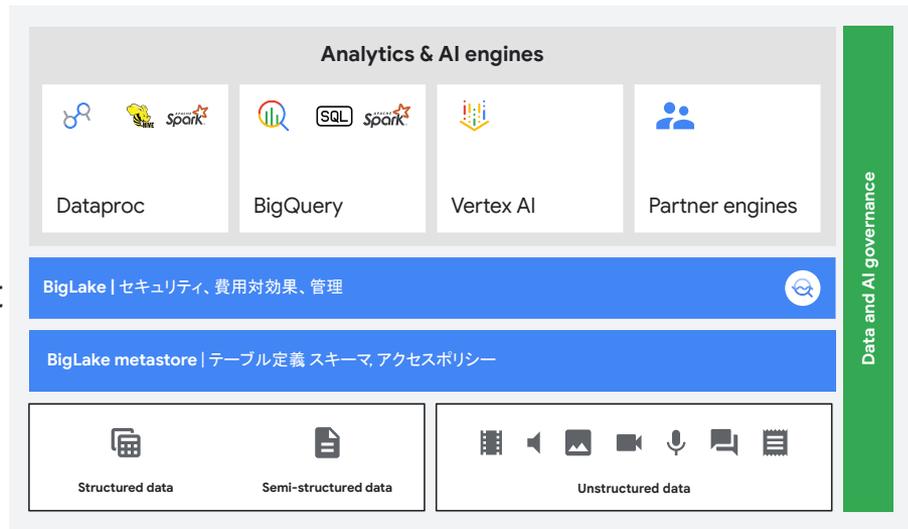
# Google Cloud におけるコンポーネント

- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# BigLake metastore

- **単一の共有メタストア**
  - SQL、オープンソース エンジン  
AI/ML のためのランタイム  
メタデータとコネクタ
- **構造化データと非構造化データのサポート**
  - 同じランタイム メタストアを使用して、構造化テーブルとオブジェクトテーブルの両方をクエリ可能
- **統合ガバナンス**
  - すべてのメタデータが大規模なポリシー管理に利用可能



# Apache Iceberg REST カタログ をサポート

- BigLake metastore は REST 互換カタログ (Unity、Polaris) と統合するための REST インターフェースを提供
- 3rd party 製の Iceberg 互換エンジンは、BigLake Tables for Apache Iceberg に対してクエリを実行可能

Iceberg REST head on BigLake metastore



# 参考: オープン カタログ テーブル

- BigLake metastore 経由で作成されたテーブルはオープン カタログ テーブルとして扱われる
- BigQuery Studio (SQL)からはReadのみ可能

iceberg\_table\_by\_spark [クエリ](#) [次で開く](#) [共有](#) [削除](#)

スキーマ [詳細](#) 分析情報 リネージ データ プロファイル データ品質

### テーブル情報 [詳細を編集](#)

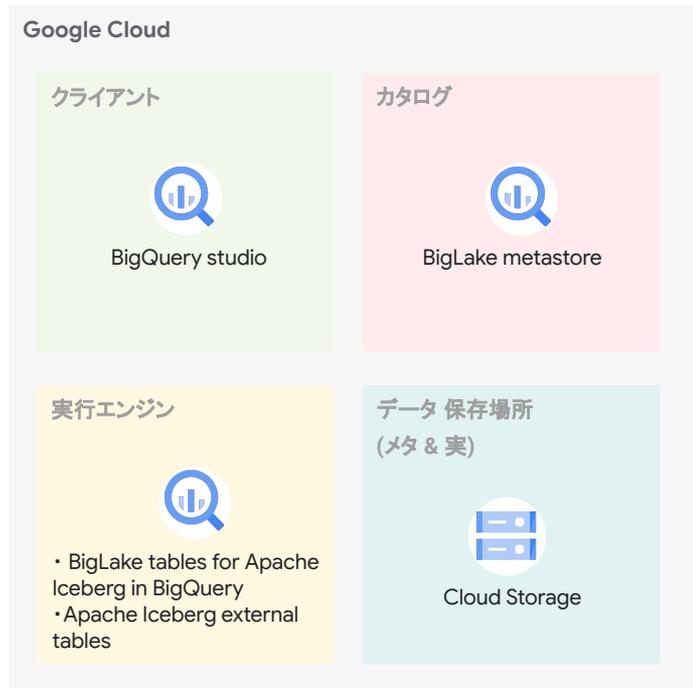
テーブル ID	dev-environment-371006.iceberg_demo_dataset_gcp_us_east.iceberg_table_by_spark
作成	2025/07/04, 11:42:56 UTC+9
最終更新	2025/07/04, 11:50:51 UTC+9
テーブルの有効期限	常にオフ
データのロケーション	us-east1
大文字 / 小文字の区別	false
なし	
説明	
ラベル	
主キー	
タグ	

### オープン カタログ テーブルの構成

ロケーション URI	gs://bucket_for_biglake_metastore/directory/iceberg_demo_dataset_gcp_us_east.db/iceberg_table_by_spark
入力形式	org.apache.hadoop.mapred.FileInputFormat
出力形式	org.apache.hadoop.mapred.FileOutputFormat
SerDe パラメータ	
パラメータ	owner : spark previous_metadata_location : gs://bucket_for_biglake_metastore/directory/iceberg_demo_dataset_gcp_us_east.db/iceberg_table_by_spark/metadata/000002da53ec34f5e4a7e872e-ddc8eeaabe70.metadata.json totalSize : 692    EXTERNAL : TRUE    numRows : 1    write.parquet.compression.codec : zstd    numFiles : 1 metadata_location : gs://bucket_for_biglake_metastore/directory/iceberg_demo_dataset_gcp_us_east.db/iceberg_table_by_spark/metadata/0000102af30976ea846e58ad6416fe86c2548.metadata.json uuid : 07ef8690b2744afc-aa69-d3788bfeadf9    table_type : iceberg

# Google Cloud におけるコンポーネント

- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# BigLake tables for Apache Iceberg in BigQuery

- フルマネージド の Iceberg テーブル
- 費用対効果を実現 するために Iceberg データを自動最適化
- BigQuery と OSS エンジン全体で読み取り遅延なし で  
1 秒あたり 10Gbps のスループットにスケール
- BigQuery ML や Vertex AI との統合は  
BigQuery ネイティブ テーブルと同様に Iceberg テーブル上で  
動作
- Iceberg テーブルに対するきめ細かいセキュリティと  
ガバナンス が、複数のエンジン間で適用可能



The screenshot shows the BigQuery console interface for a table named 'bq\_table\_for\_i...'. The 'Details' tab is selected, displaying the following information:

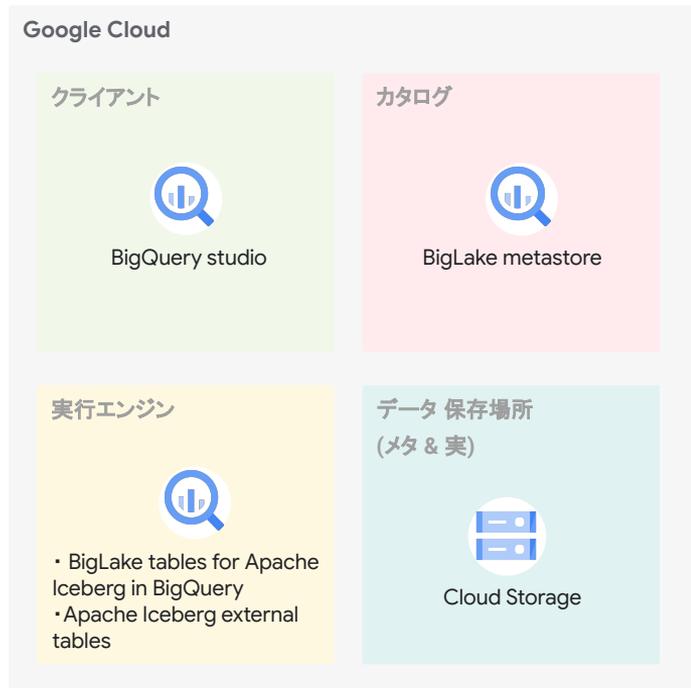
テーブル情報	
テーブル ID	dev-environment-371006.iceberg_demo_dataset_gcp_us_east.bq_table_for_iceberg
作成	2025/05/11, 18:39:24 UTC+9
最終更新	2025/05/12, 14:37:16 UTC+9
テーブルの有効期限	常にオフ
データのロケーション	us-east1
デフォルトの照合	
デフォルトの丸めモード	ROUNDING_MODE_UNSPECIFIED
大文字 / 小文字の区別	false
なし	
説明	
ラベル	
主キー	
タグ	

Below the table information, the 'Apache Iceberg 構成用の BigQuery テーブル' section is visible:

接続 ID	dev-environment-371006.us-east1.biglake_connection_for_iceberg_us_east
Storage URI	gs://bucket_for_bigquery_tables_for_apache_iceberg_us_east/
ファイル形式	PARQUET
表形式	ICEBERG

# Google Cloud におけるコンポーネント

- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# Apache Iceberg external tables

- オープンソーステーブルフォーマットであるIceberg を BigLake でサポート
- BigLake の外部テーブルの機能に加えて、カタログを利用したデータの整合性やスキーマの更新が可能
- Row, column level security, masking 等 BigLake のセキュリティ機能をIceberg テーブルに適用
- BigLake Metastore や AWS Glue Data Catalog をメタストアとして使用可能
- 書き込みは外部エンジン経由の書き込みのみサポート

## データセットを作成する

ロケーションタイプ 

リージョン  
他の Google Cloud サービスとデータセットをコロケーションするリージョンを指定します。

マルチリージョン  
BigQuery でグループ内のリージョンを選択し、割り当ての上限を引き上げることができます。

リージョン\*  
aws-eu-west-1

デフォルトのテーブルの有効期限

テーブルの有効期限を有効にする 

デフォルトのテーブル最長存続期間 日数

外部データセット

外部データセットへのリンク 

外部データセットの種類\*  
AWS Glue

外部ソース\*  
aws-glue://arn:aws:glue:us-east-1:123456789:database/test\_database.

接続 ID

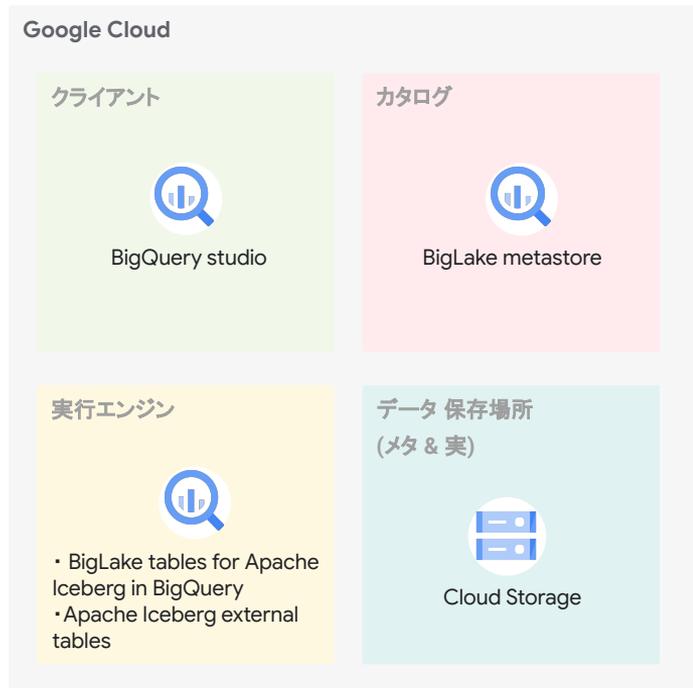
詳細オプション 

暗号化 

データセットを作成 キャンセル

# Google Cloud におけるコンポーネント

- クライアント
  - BigQuery Studio
- カタログ
  - BigLake Metastore
- 実行エンジン: BigQuery (Dremel)
  - テーブルパターン
    - BigLake tables for Apache Iceberg in BigQuery
    - Apache Iceberg external tables
- データ保存場所
  - Cloud Storage



# Cloud Storage

**シンプル:** ストレージ クラス全体で一貫した API レイテンシ、速度により開発をシンプルに

**信頼性:** 冗長化され、透過的なフェイルオーバーとフェイルバックにより、99.95% の可用性と年間 99.999999999% の耐久性を実現

**費用対効果:** アーカイブ ストレージは月額 0.0012 ドル/GB から利用可能。ミリ秒単位でアクセス可能で冗長オプションもご利用可能

**安全性:** データは保存時および転送時に暗号化され、WORM オプションと KMS および IAM と統合可能

**生成 AI 活用:** Gemini アシスタントによるデータ ストレージ分析情報を利用してデータに関する分析情報を提供

The screenshot shows the Google Cloud Storage console interface. The bucket name is 'bucket\_for\_bigquery\_tables\_for\_apache\_iceberg\_us\_east'. The location is 'us-east1 (サウスカロライナ)'. The storage class is 'Standard'. The public access is '非公開' (Private). The protection is '削除 (復元可能)' (Delete (Restoreable)).

The interface shows a folder browser for the bucket, with a 'metadata' folder selected. The table below lists the objects in this folder:

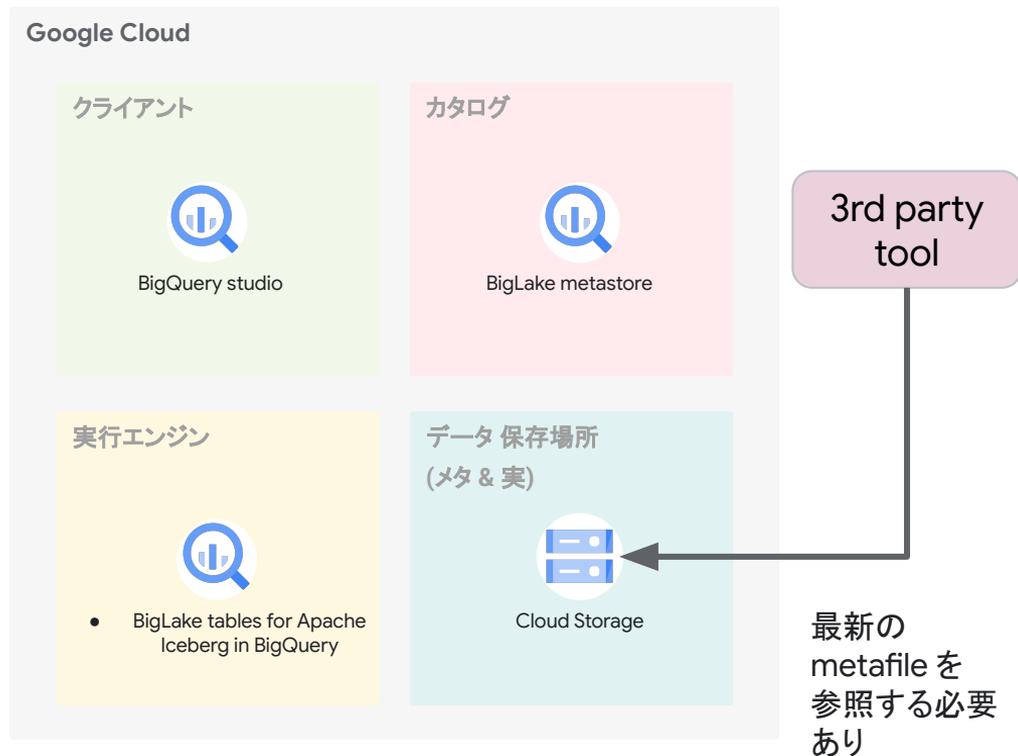
名前	サイズ	種類	作成日
9c758325-4f08-448e-9d4d-46b87...	2 KB	application/octet-stream	2025/05/12 14:42:48
9c758325-4f08-448e-9d4d-46b87...	495 B	application/octet-stream	2025/05/12 14:42:48
fc6f1b2c-c455-40bf-9133-4470e52...	2 KB	application/octet-stream	2025/05/12 12:17:54
fc6f1b2c-c455-40bf-9133-4470e52...	495 B	application/octet-stream	2025/05/12 12:17:54
v0.metadata.json	140 B	application/json	2025/05/11 18:39:24
v1747019874.metadata.json	852 B	application/octet-stream	2025/05/12 12:17:54
v1747028567.metadata.json	852 B	application/octet-stream	2025/05/12 14:42:48
version-hint.text	10 B	application/octet-stream	2025/05/12 14:42:48

# ユースケース その 1

- Google Cloud 上でアプリケーションが動作
- ログの保存先として Cloud Storage を選択
- BigQuery からは Read/Write が発生
- 3rd party 製のツールからも Iceberg 形式のデータに対して Read が発生する

# ユースケース その1 アーキテクチャ

- BigQuery 上で行われる Write に対して、BigQuery からは常に最新データを参照可能
- 3rd Party ツールからの参照に対しては、最新の metafile の情報が必要

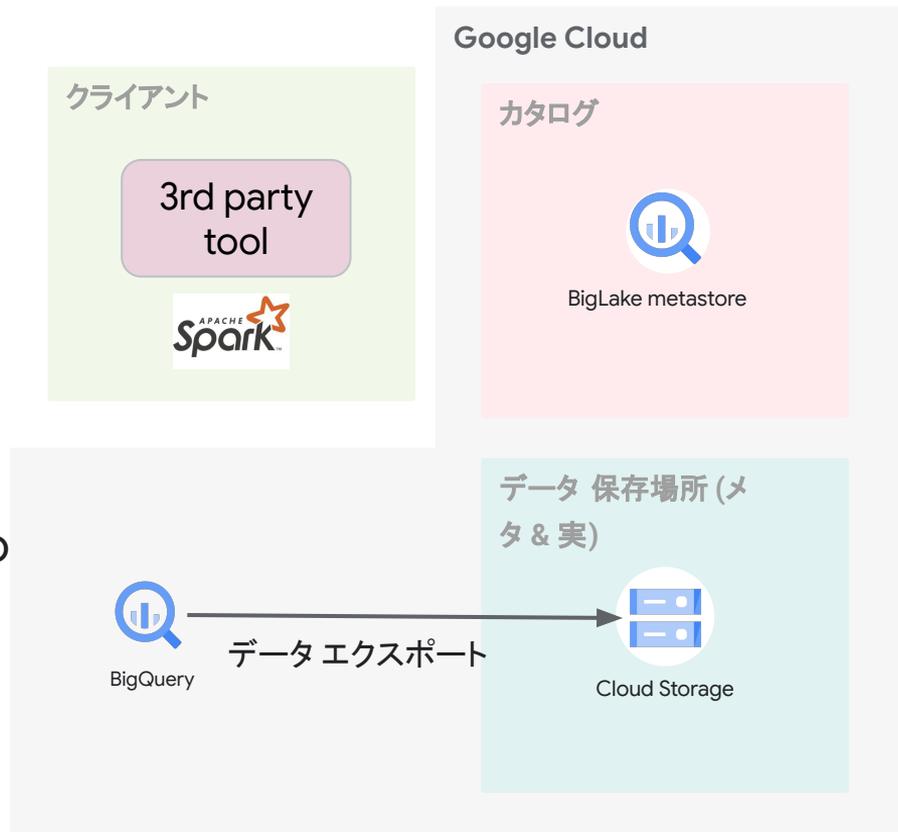


## ユースケース その 2

- 複数 Cloud 上にデータが分散
- クライアントから複数 Cloud に対して透過的なクエリを実現したい
- BigQuery にテーブルが存在しているが Iceberg 形式のテーブルとしても持たせておきたい
- BigQuery Studio (SQL 利用)からは Read だけ実現できればいい

# ユースケース その2 アーキテクチャ

- BigLake metastore を用いてオープン テーブル フォーマットのテーブルを作成
- 作成されたテーブルに対して BigQuery からデータをエクスポートして挿入する
- BigLake metastore と 3rd party tool が利用しているカタログを統合する



# 堤 崇行

Google Cloud  
Customer Engineer



## 02. データ パイプライン

# レイクハウスとデータ パイプライン



# データ パイプライン



 Pub/Sub 単一メッセージ変換 (SMT)

 BigQuery リバース ETL

 BigQuery Subscription

 BigQuery 継続的クエリ

 Dataflow

 Dataform

 Dataproc

 BigQuery データ準備

 Data Fusion

 Cloud Composer

 Workflows

 BigQuery パイプライン

ワークフロー

# Gemini in データ パイプライン



 BigQuery データ キャンバス

 BigQuery Notebooks

 Dataform

 BigQuery データ準備

 BigQuery Data Engineering Agent

 BigQuery パイプライン

ワークフロー

# データ パイプラインの位置付け

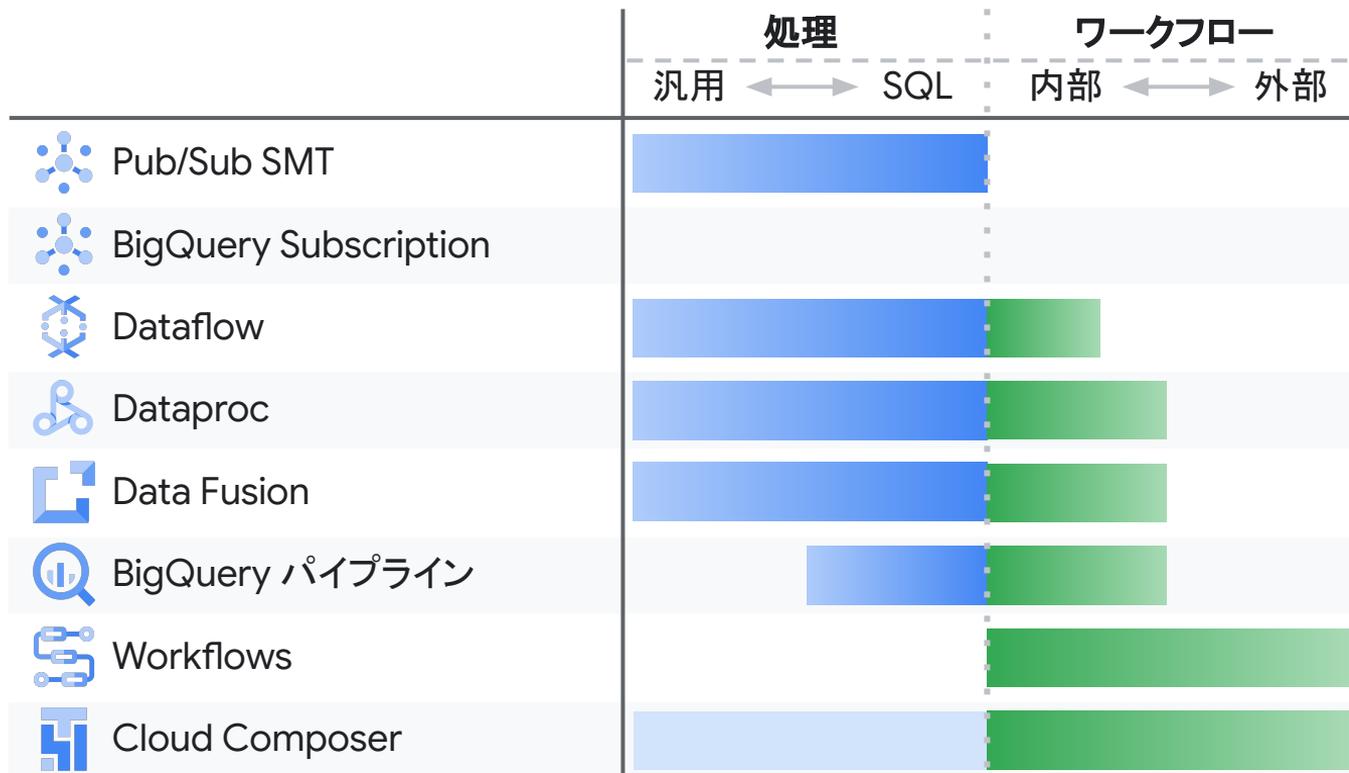
## 処理の実装言語

汎用 プログラミング言語	SQL
Python や Java など 汎用プログラミング言語で 実装	SQL や SQLX など SQL ベースの言語で 実装

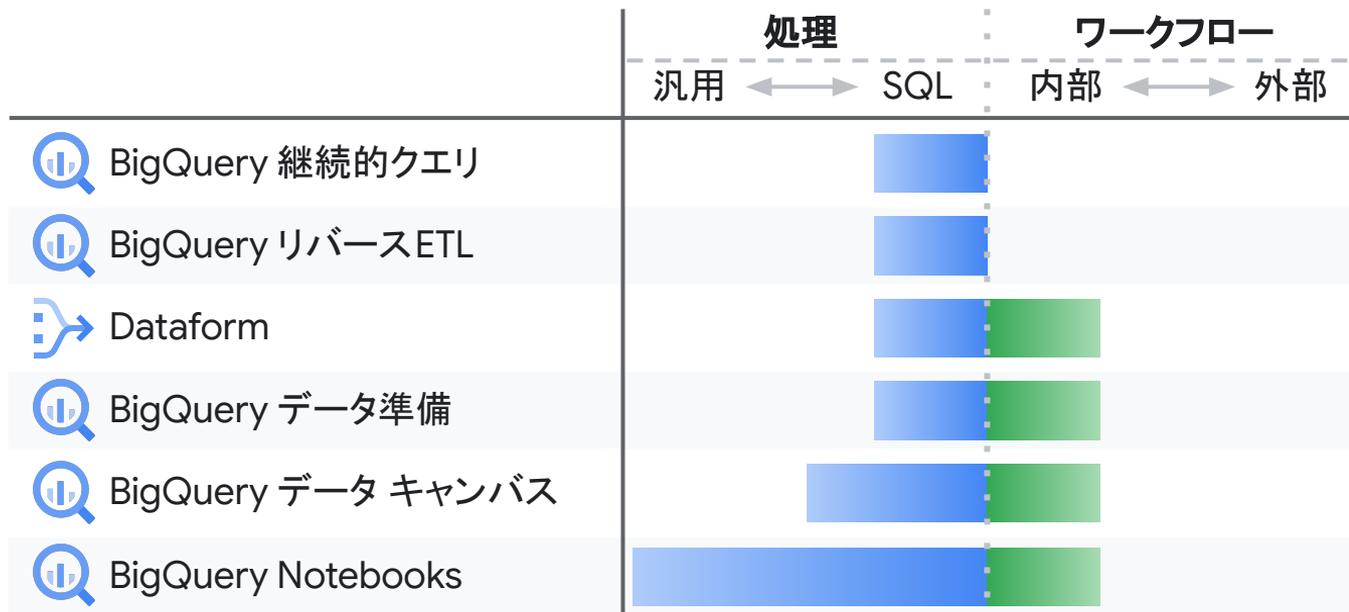
## ワークフロー

内部的	外部的
当該プロダクト内部の ステップやワークフロー 作成をサポート	当該プロダクト以外の 処理エンジンを含めた ワークフローを作成を サポート

# データ パイプラインの位置付け ( 1/2)



# データ パイプラインの位置付け ( 2/2 )



# Dataflow



 Pub/Sub 単一メッセージ変換 (SMT)

 BigQuery リバースETL

 BigQuery Subscription

 BigQuery 継続的クエリ

 **Dataflow**

 Dataform

 Dataproc

 Data Fusion

 BigQuery データの準備

 Cloud Composer

 Workflows

 BigQuery パイプライン

**ワークフロー**

# Dataflow

Dataflow と Apache Beam でデータパイプラインの新たな可能性を切りひらく

## リアルタイム ストリーム処理

Pub/Sub や Kafka、その他多様なソースから流入するデータストリームに対して、サーバーレスで継続的な分析を実行

## リアルタイムアプリ開発のための生成 AI

リアルタイムの異常検知、感情分析、  
レコメンデーションといったAI/ML の機能を用いて  
自動化パイプラインを開発

## オペレーショナル システムへのリバース ETL

レイクハウスのデータを変換・複製  
キュー、KVS、DB にリアルタイムでロード



BigQuery



Dataflow

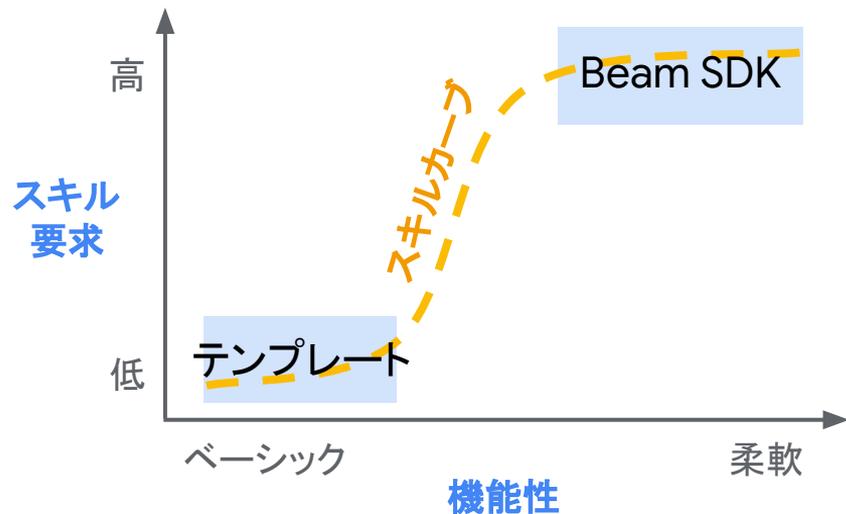


Vertex AI

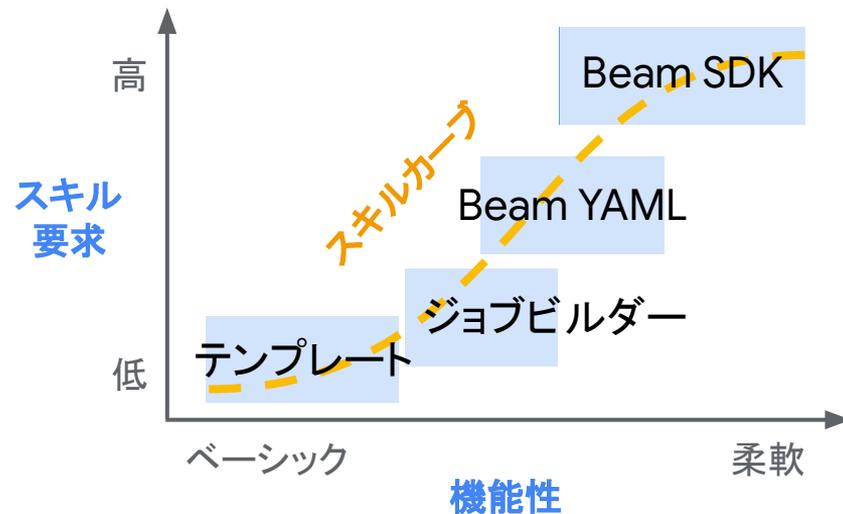


# Dataflow のスキルカーブ

ジョブビルダーがない時



ジョブビルダーがある時



# Beam YAML

YAML で Beam パイプラインを作成、実行

- ソース、変換、シンクの 3 フェーズを記述
- 簡単で、読みやすく、保守性も向上
- 複数ソースの結合、複数シンクにも対応
- SQL 変換 や Python 変換も使用可能
- テンプレート化、パッケージ化で再利用

## Kafka to BigQuery の例

Java

```
public static PipelineResult run(KafkaToBqOptions options) {
    Pipeline pipeline = Pipeline.create(options);
    Map<String, Object> kafkaConfig =
        ImmutableMap.of(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
    PCollectionTuple convertedTableRows = null;
    if (options.getMessageFormat().equals("JSON")) {
        return runJsonPipeline(pipeline, options, topicsList, bootstrapServers, kafkaConfig);
    } else if (options.getMessageFormat().equals("AVRO")) {
        return runAvroPipeline(pipeline, options, topicsList, bootstrapServers, kafkaConfig);
    } else {
        throw new IllegalArgumentException("Invalid format specified: " + options.getMessageFormat());
    }
}

public static PipelineResult runJsonPipeline(
    Pipeline pipeline,
    KafkaToBqOptions options,
    List<String> topicsList,
    String bootstrapServers,
    Map<String, Object> kafkaConfig) {
    PCollectionTuple convertedTableRows =
        pipeline
            .apply(
                "ReadFromKafka",
                KafkaTransform.readStringFromKafka(bootstrapServers, topicsList, kafkaConfig, null))
            .apply("ConvertMessageToTableRow", new StringMessageToTableRow(options));

    WriteResult writeResult =
        convertedTableRows
            .get(TRANSFORM_OUT)
            .apply(
                "WriteSuccessfulRecords",
                BigQueryIO.writeTableRows()
                    .withoutValidation()
                    .withCreatedDisposition(CreateDisposition.CREATE_NEVER)
                    .withWriteDisposition(WriteDisposition.WRITE_APPEND)
                    .withExtendedErrorInfo()
                    .withFailedInsertRetryPolicy(InsertRetryPolicy.retryTransientErrors()))
            .to(options.getOutputTableSpec());
}
```

YAML

```
pipeline:
  transforms:
  - type: ReadFromKafka
    config:
      format: JSON
      schema: {"type": "object", "properties": {"propertyName": {"type": "string"}}}
      topic: my_topic
      bootstrap_servers: some.host:9092
  - type: WriteToBigQuery
    input: ReadFromKafka
    config:
      table: myproject.mydataset.mytable

options:
  streaming: true
```

# Dataflow ジョブビルダー with YAML サポート

データ処理パイプラインを簡単に作成

- あらゆるスキルレベルのユーザーがストリーミング データを活用可能に
- GUI を使用してパイプラインを作成
- GUI で開始、YAML でカスタマイズ
- Java/Python SDK でカスタム開発

The screenshot shows the 'YAML エディタ' (YAML Editor) tab in the Dataflow Job Builder. The job name is 'wordcount'. The job type is 'バッチ' (Batch). The source is 'GCS からの読み取り' (Read from Cloud Storage). The transformation steps are: '単語の分割' (Split words), '単語配列の展開' (Expand word array), '単語数のカウント' (Count words), and '出力のフォーマット' (Format output). The sink is 'GCS への書き込み' (Write to Cloud Storage) and 'BigQuery テーブル への書き込み' (Write to BigQuery table).



# 継続的クエリ とリバーズ ETL



 Pub/Sub 単一メッセージ変換 (SMT)

 BigQuery リバーズ ETL

 BigQuery Subscription

 BigQuery 継続的クエリ

 Dataflow

 Dataform

 Dataproc

 BigQuery データの準備

 Data Fusion

 Cloud Composer

 Workflows

 BigQuery パイプライン

ワークフロー

# リバース ETL (RETL)

BigQuery の EXPORT 文でデータベースへリバース ETL

リバース ETL に対応する書き込み先:

## Spanner / Bigtable

低レイテンシで高スループットな DB を  
BigQuery の分析機能と組み合わせる

## Pub/Sub

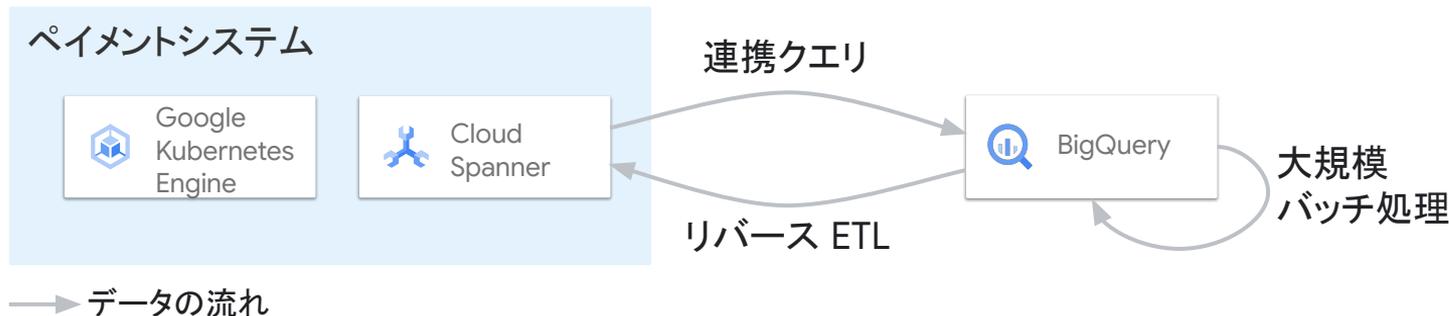
非同期でスケラブルな  
グローバル メッセージング サービスと  
BigQuery の分析機能を組み合わせる

リバース ETL の サンプル  
(to Spanner)

```
EXPORT DATA OPTIONS (  
  uri="https://spanner.googleapis.com/  
  projects/PROJECT_ID/instances/INSTAN  
  CE_ID/databases/DATABASE_ID",  
  format='CLOUD_SPANNER',  
  spanner_options="" { "table":  
  "TABLE_NAME" } ""  
)  
AS SELECT * FROM mydataset.table1;
```

# リバース ETL - ユースケース

ペイメントシステムなどのサービスレベル要求が高く  
トランザクション数も多いシステムで Cloud Spanner を使用している場合  
BigQuery の SQL を使用するだけで  
”BigQuery で大規模なバッチ処理を実行、結果をリバース ETL” が可能に



# BigQuery 継続的クエリ

## SQL ベースの 継続的な分析処理

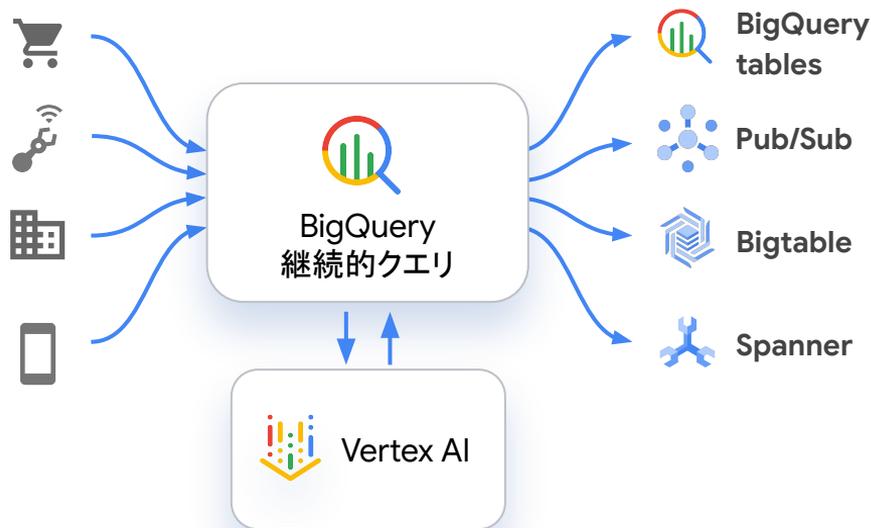
BigQuery スケールで  
受信データをストリーム処理

## リアルタイム アプリ開発への生成 AI

リアルタイムに  
感情分析、推奨事項、異常検出などの  
AI + ML 機能を備えたパイプラインを開発

## オペレーショナル システム への リバース ETL 連携

BigQuery のデータをリアルタイムで変換  
Spanner、Bigtable、Pub/Sub、  
BigQuery テーブル に連携



# 継続的クエリ - ユースケース

Google Analytics などのウェブ解析データをストリーミングで取得  
BigQuery 継続的クエリでストリーミング処理をした結果を Bigtable に反映  
Web アプリが最新の情報を取得



- フィルター
- マッピング
- フラット化
- フラグ化

```

EXPORT DATA
  OPTIONS (
    format = 'CLOUD_BIGTABLE',
    uri = "https://bigtable.googleapis.com/projects/my-project/instances/bg-export-demo/tables/online-storefront"
  ) AS (
  SELECT
    CAST(CONCAT(event_timestamp, event_bundle_sequence_id, user_pseudo_id) AS STRING) AS rowkey,
    STRUCT(
      event_timestamp,
      IF(event_name IN ('first_visit', 'first_open'), "true", "false") AS is_new_user,
      user_ltv.revenue AS user_lifetime_revenue,
      device.web_info.browser AS device_browser,
      geo.country AS user_country,
      traffic_source.source AS traffic_source,
      IF(event_name IN ('in_app_purchase', 'purchase'), 1, 0) AS purchase_event
    ) AS features
  FROM `ec_demo.ga4_obfuscated_sample_ecommerce_events`
  WHERE
    event_name IN ('first_visit', 'first_open')
  );
  
```

→ データの流れ

# おわりに

## データパイプラインを実装する手段は色々

- 要件、開発・運用スタイルに合わせて選べる
- Dataflow の実装手段も色々ある
- BigQuery の SQL ができることは増えている

## AI エージェントにより更に間口が大きく

- 自然言語で対話しながらパイプラインを作成できる
- 自然言語で対話しながらデータ分析できる
- もちろんアシスタントもしてくれる