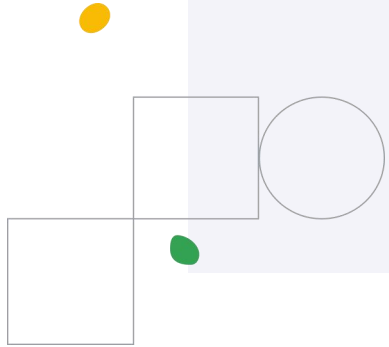
Ingesting Data into BigQuery

03



Google Cloud

Topics

01	Data Ingestion Options
02	Batch Ingestion
03	Streaming Ingestion
04	Legacy Streaming API
05	BigQuery Storage Write API

Google Cloud

Topics

06	Query Materialization
07	Query External Data Sources
08	Data Transfer Service

Google Cloud

Data Ingestion Options

Data ingestion options

Batch ingestion

Data from Cloud Storage or via
HTTP POST

Multiple file formats supported

Snapshot-based arrival:
All data arrives at once, or not at all

Streaming ingestion

Continuous ingestion from many sources (web/mobile apps, point of sale, supply chain)

Immediate query availability from buffer

Deferred creation of managed storage

Query materialization

SELECT results yield data in the form of tables, either anonymous (cached) or named destinations

ETL/ELT ingest and transform via federated query

Data Transfer Service

Managed ingestion of other sources (DoubleClick, Google Ads, YouTube)

Scheduled queries, Scheduled Cloud Storage ingestion

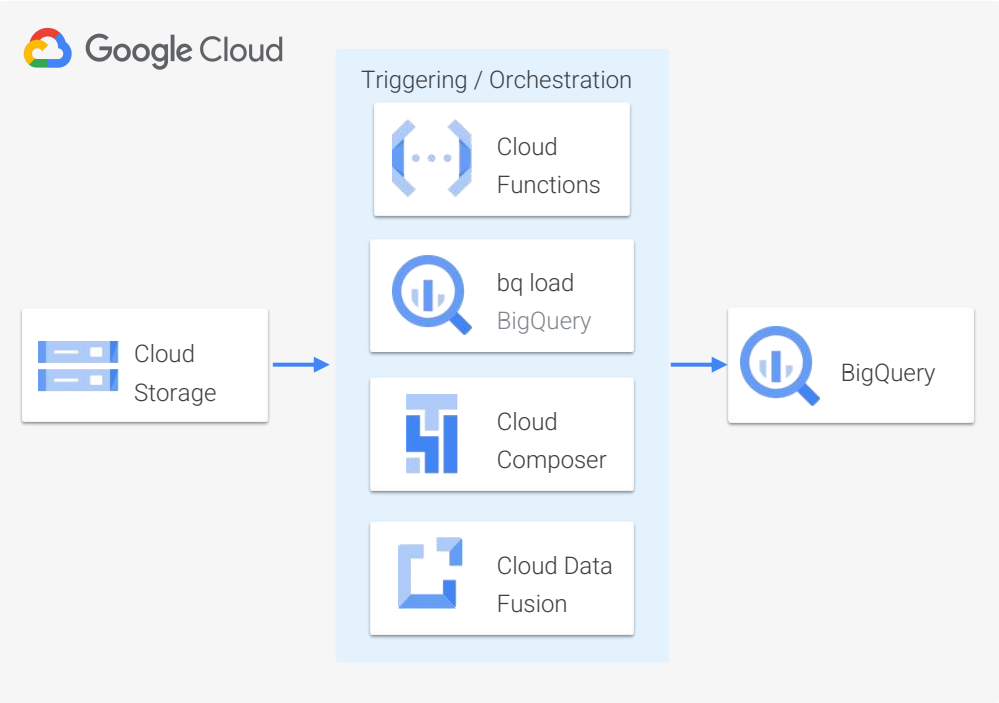
Options for third-party integration

Google Cloud

The method you use to load data depends on how much transformation is needed

Extract
and **L**oad

When would you use EL?



Architecture

Extract data from files on Cloud Storage and load it into BigQuery's native storage.

You can trigger this from Cloud Composer, Cloud Functions, or scheduled queries.

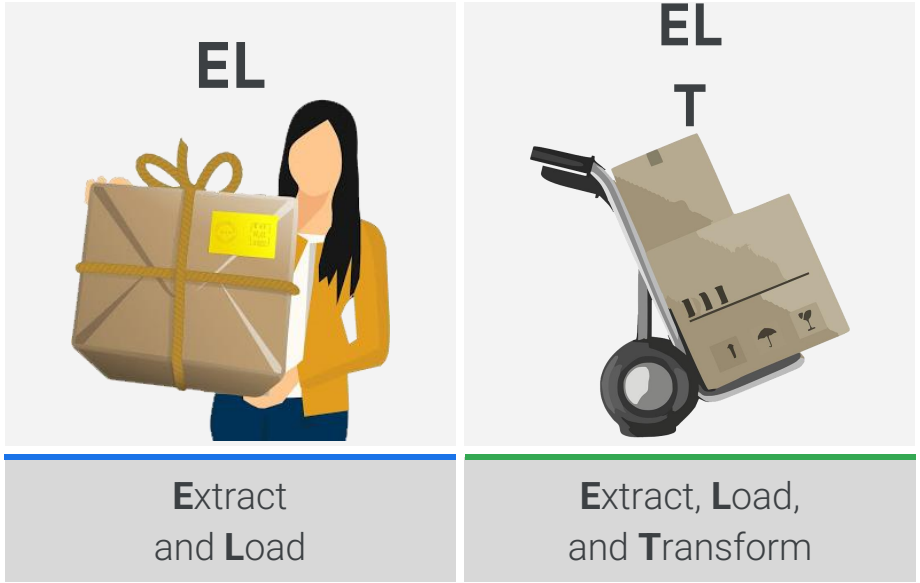
When you use it

Batch load of historical data

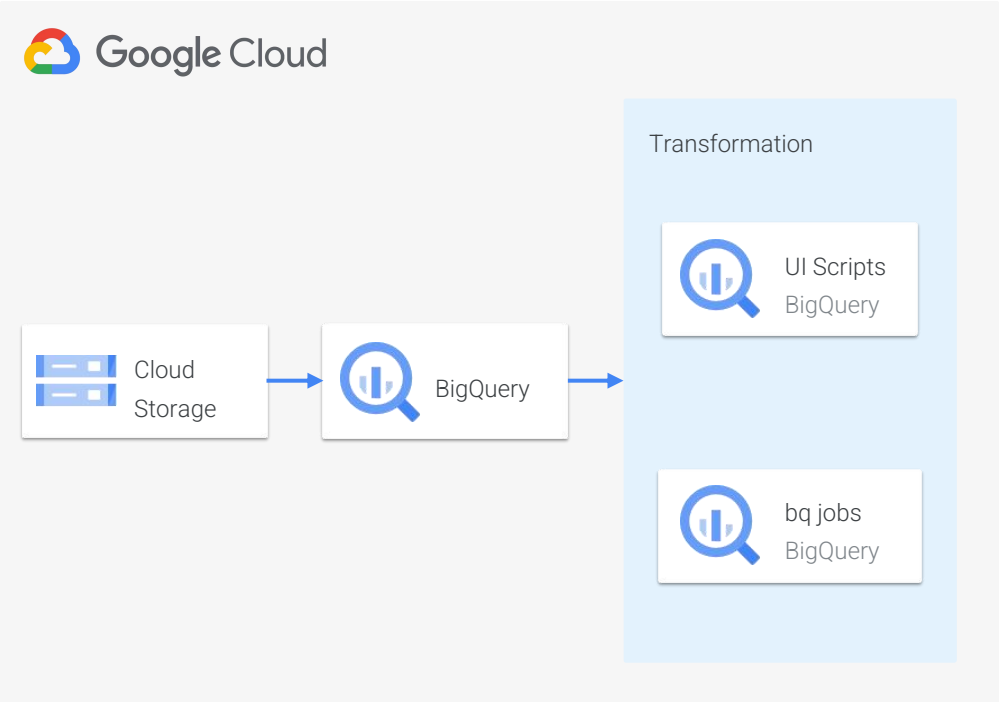
Scheduled periodic loads of log files (e.g., once a day)

But only if the data is already clean and correct!

The method you use to load data depends on how much transformation is needed



When would you use ELT?



Architecture

Extract data from files in Cloud Storage into BigQuery.

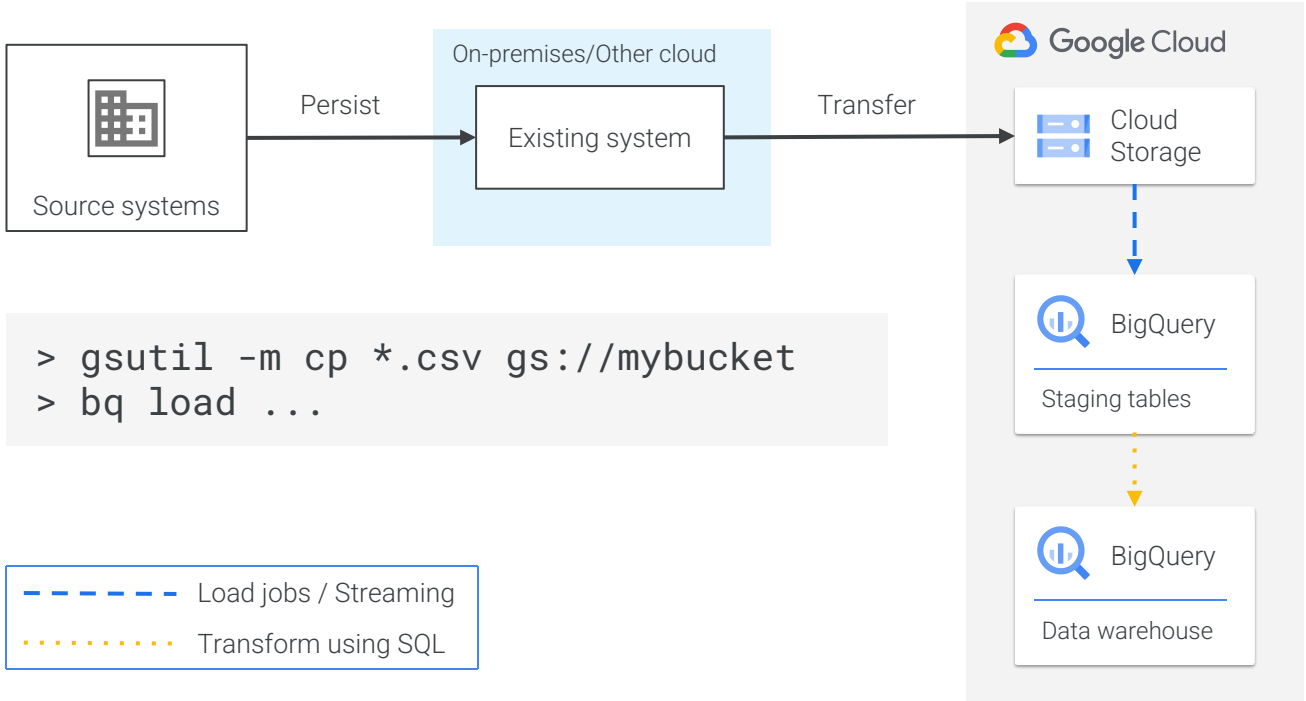
Transform the incoming data using BigQuery views, or store into new tables.

When you use it

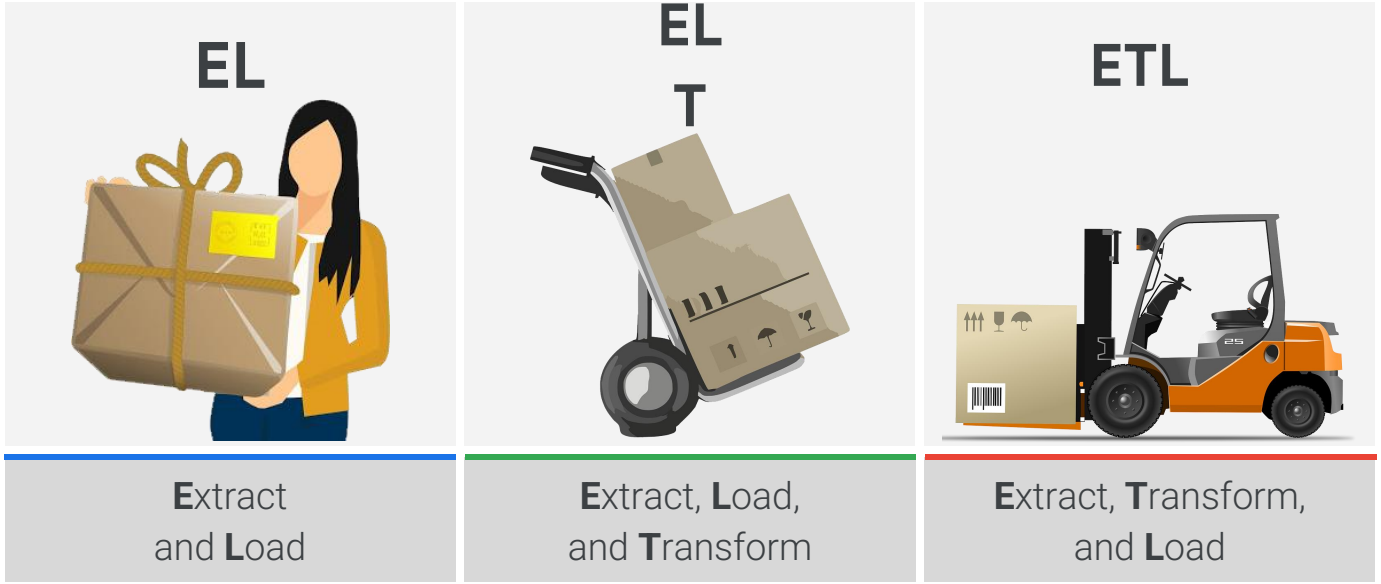
Experimental datasets where you are not yet sure what kinds of transformations are needed to make the data useable

Any production dataset where the transformation can be expressed in SQL

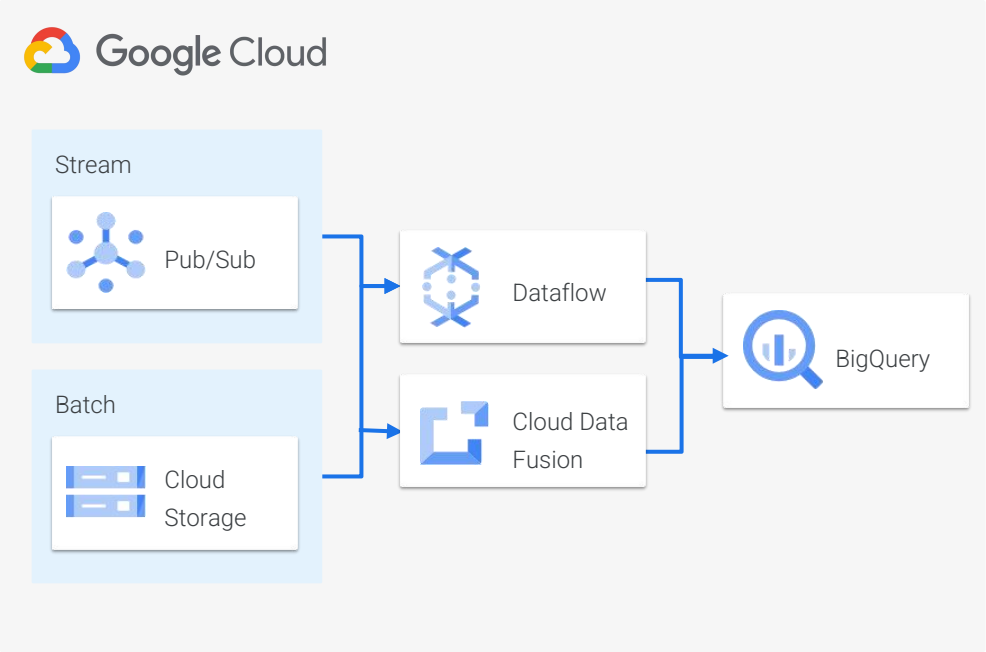
Loading and transforming data in BigQuery



The method you use to load data depends on how much transformation is needed



Build ETL pipelines and land the data in BigQuery



Architecture

Extract data from Pub/Sub, Cloud Storage, Cloud Spanner, Cloud SQL, etc.

Transform the data.

Have the pipeline write to BigQuery.

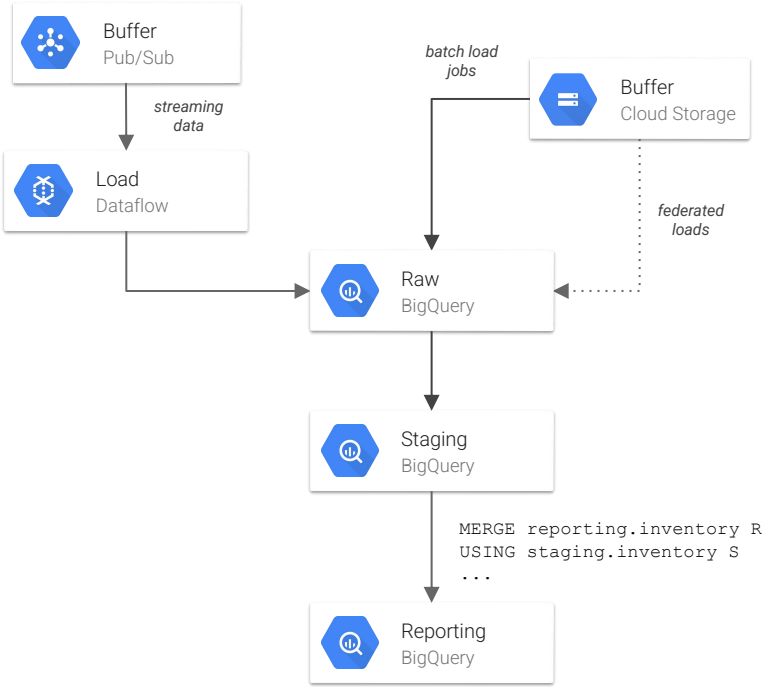
When would you do it

Quality control, transformation, or enrichment before loading data into BigQuery

Continuous data loading, i.e., if the use case requires streaming

Integration with continuous integration/ continuous delivery (CI/CD) systems; unit testing on all components

Best practice: ELT/ETL



Prefer ELT into BigQuery over ETL where possible.

Leverage federated queries to Cloud Storage to load and transform data in a single step.

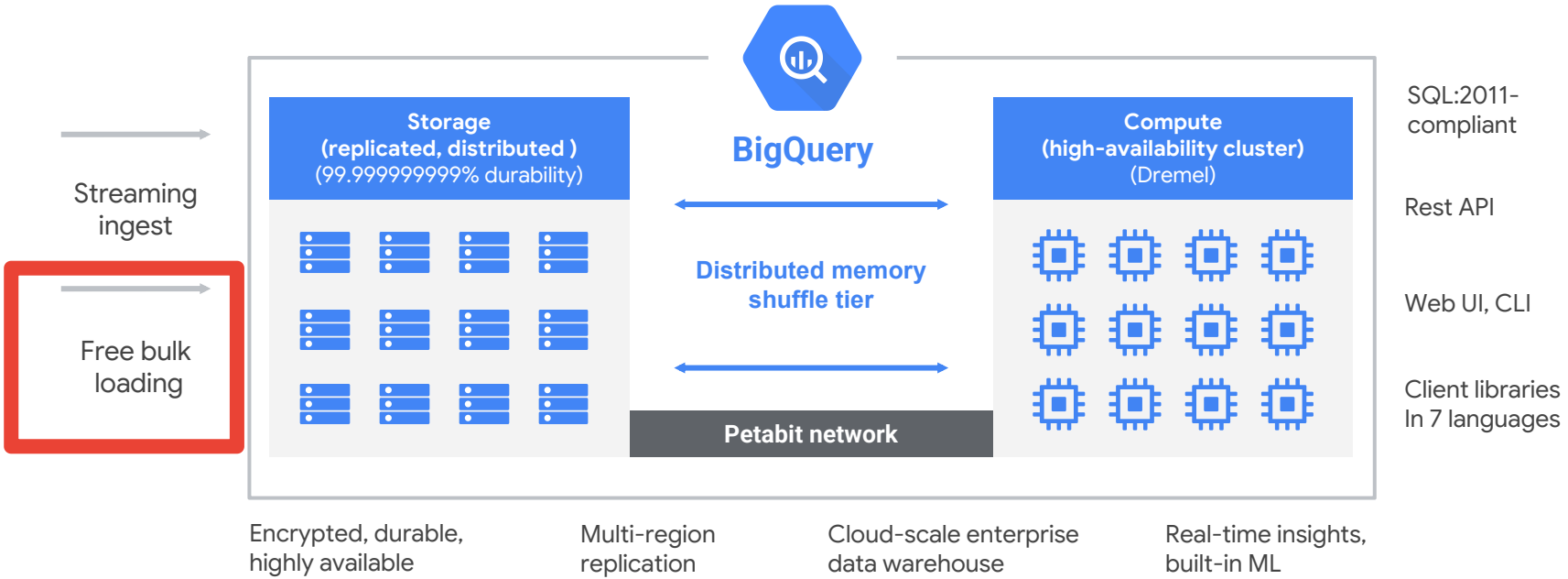
Load data into raw and staging tables before publishing to reporting tables.

Use Dataflow for streaming pipelines and to speed up large, complex batch jobs.

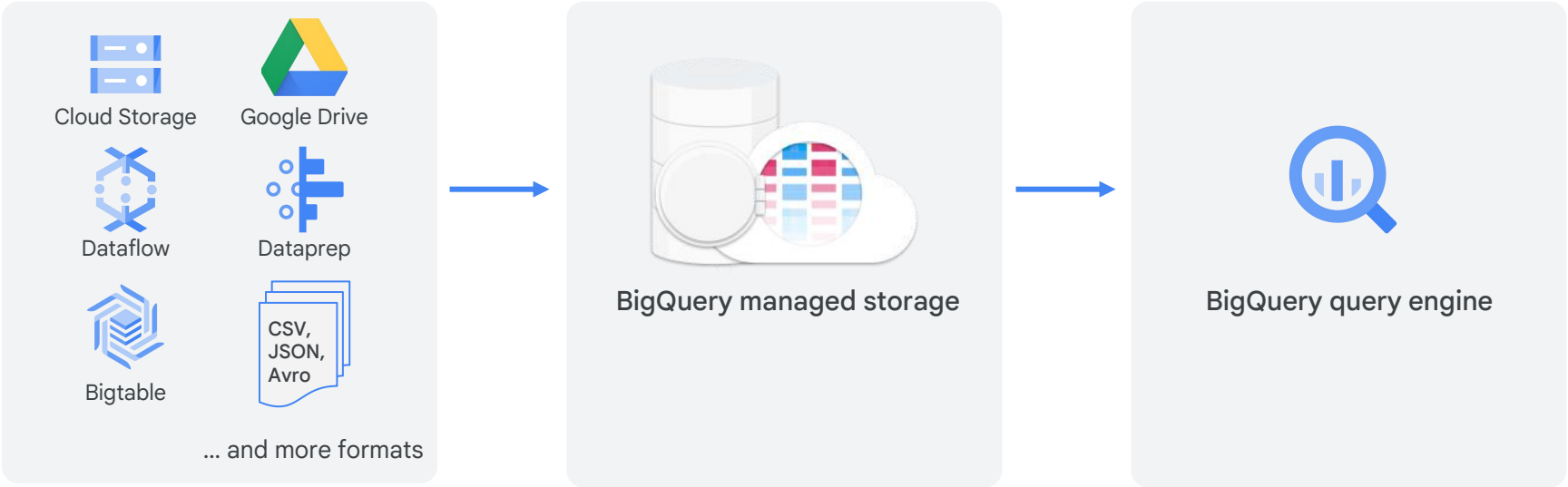
Get started streaming using Google-Provided Dataflow Templates, and modify the open source pipeline for more complex needs.

Batch Ingestion

BigQuery architecture



Ingest data permanently into BigQuery from a variety of formats

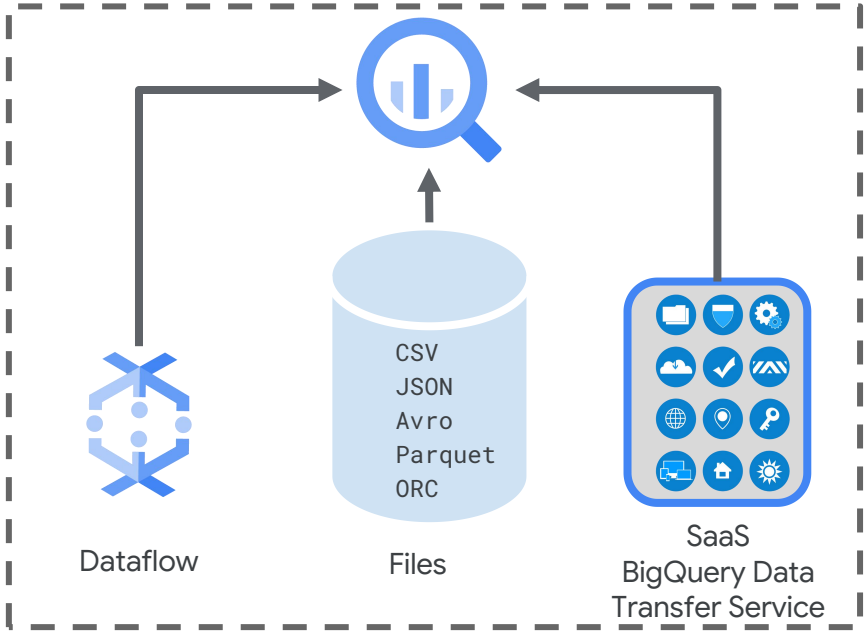


Data ingested into BigQuery is stored in permanent tables, and this storage is scalable and fully managed.

Data ingestion: Load jobs

- Batch loading is free
- Doesn't consume query capacity
- ACID semantics
- Load petabytes per day
- Streaming API for real time

Loading data into BigQuery tables (batch, periodic) offers the best performance



supported file formats

Best practice: Data format

Faster

Avro (Compressed)

Avro (Uncompressed)

Parquet/ORC

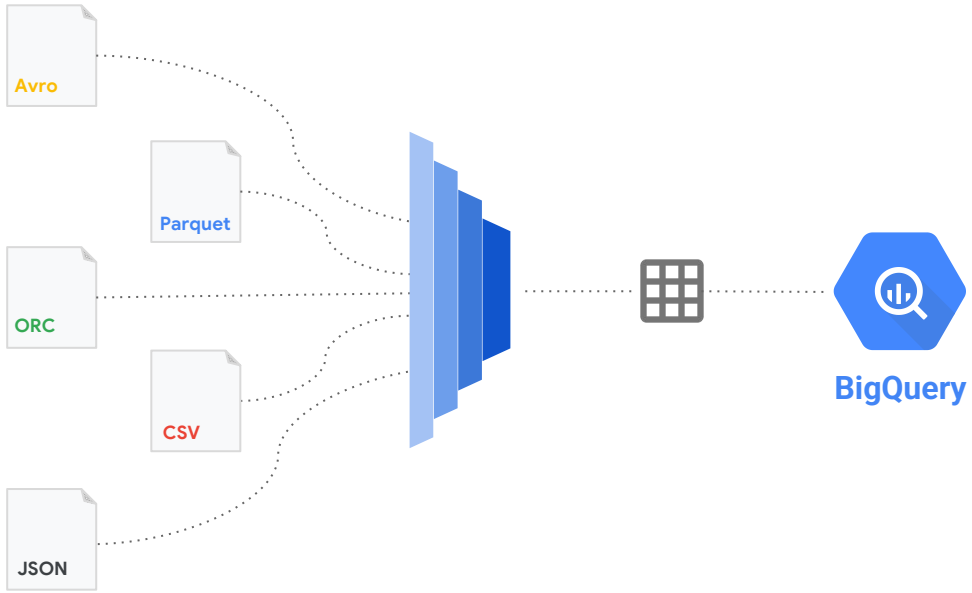
CSV

JSON

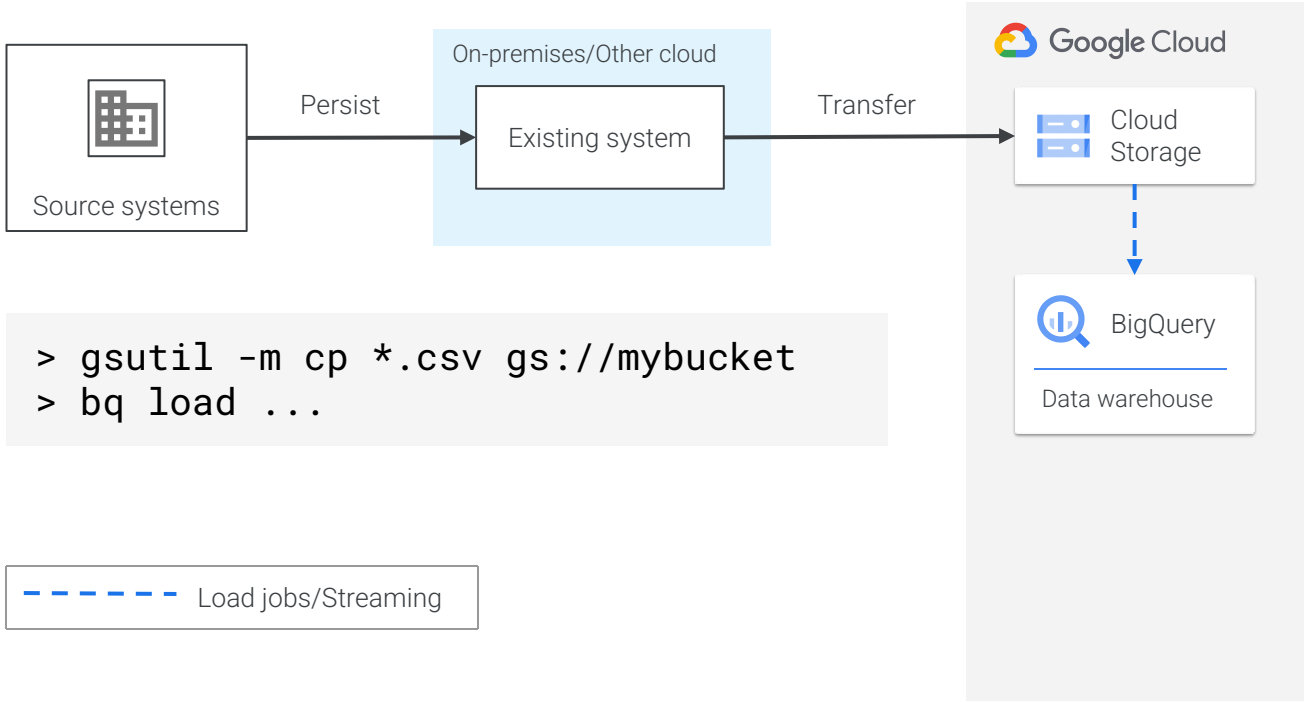
CSV (Compressed)

JSON (Compressed)

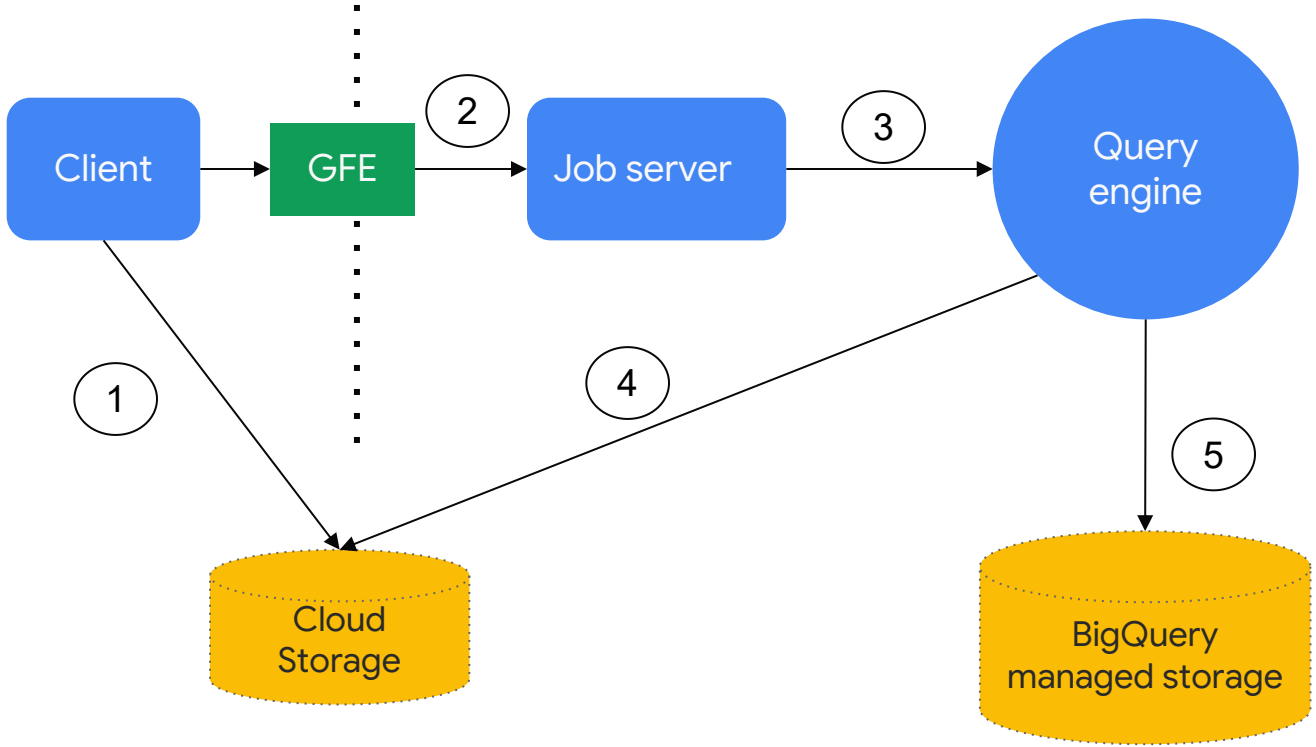
Slower



Loading data through Cloud Storage



Batch ingestion



Batch ingestion

- Read from multiple formats:
  - CSV, JSON, Avro, Parquet, ORC
- Multiple data sources:
  - Cloud Storage, Bigtable, Firestore backup, direct upload
- Run on Dremel just like a regular query that writes to a table.
- Uses the query engine to perform the ingestion and the required recoding.
- Performance can be managed through reservations.
- Per table limit of 1500 loads per day to prevent rapid metadata increase due to microbatching.

In case something happens during load: BigQuery addresses backup and disaster recovery at the service level (time travel)

```
CREATE OR REPLACE TABLE ch10eu.restored_cycle_stations AS
SELECT
*
FROM bigquery-public-data.london_bicycles.cycle_stations
FOR SYSTEM_TIME AS OF
TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 24 HOUR)
```

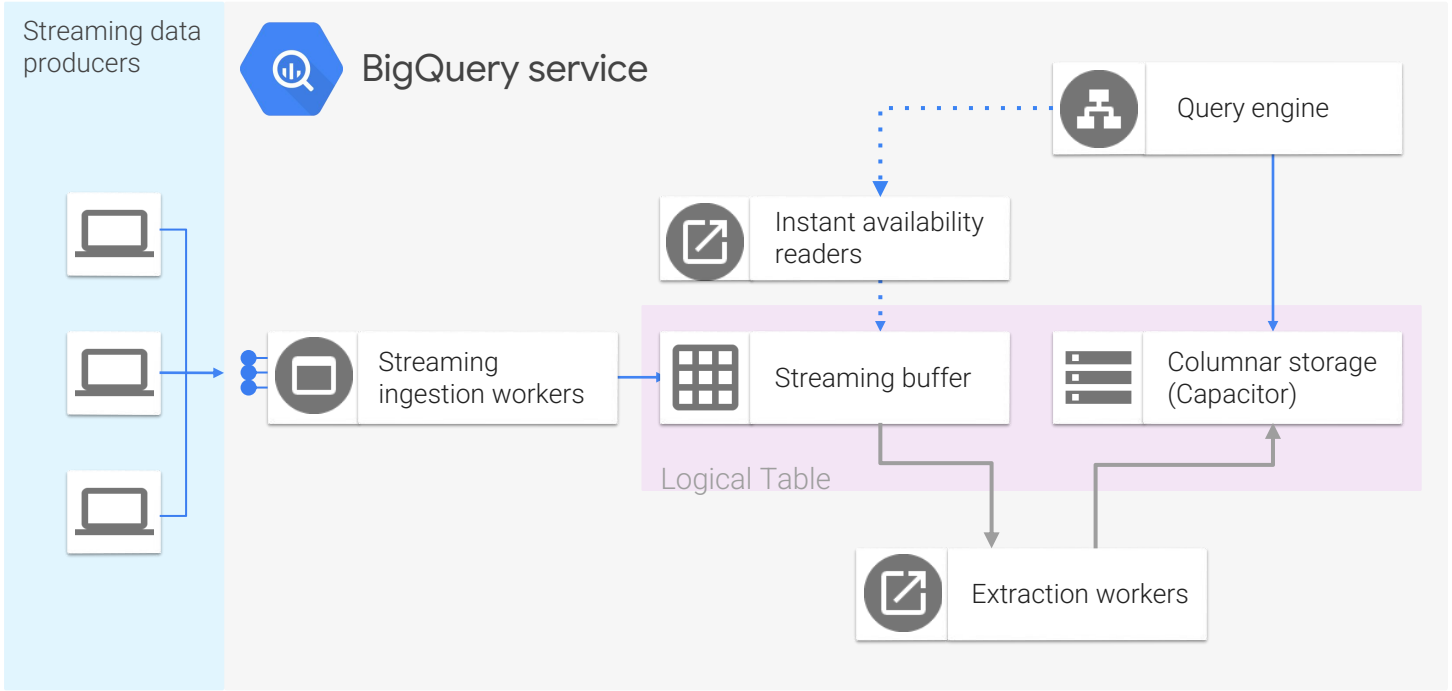
Query a point-in-time snapshot (up to 7 days).

Use it to create a backup.

```
NOW=$(date +%s)
SNAPSHOT=$(echo "($NOW - 120)*1000" | bc)
bq --location=EU cp \
ch10eu.restored_cycle_stations@$SNAPSHOT \
ch10eu.restored_table
```

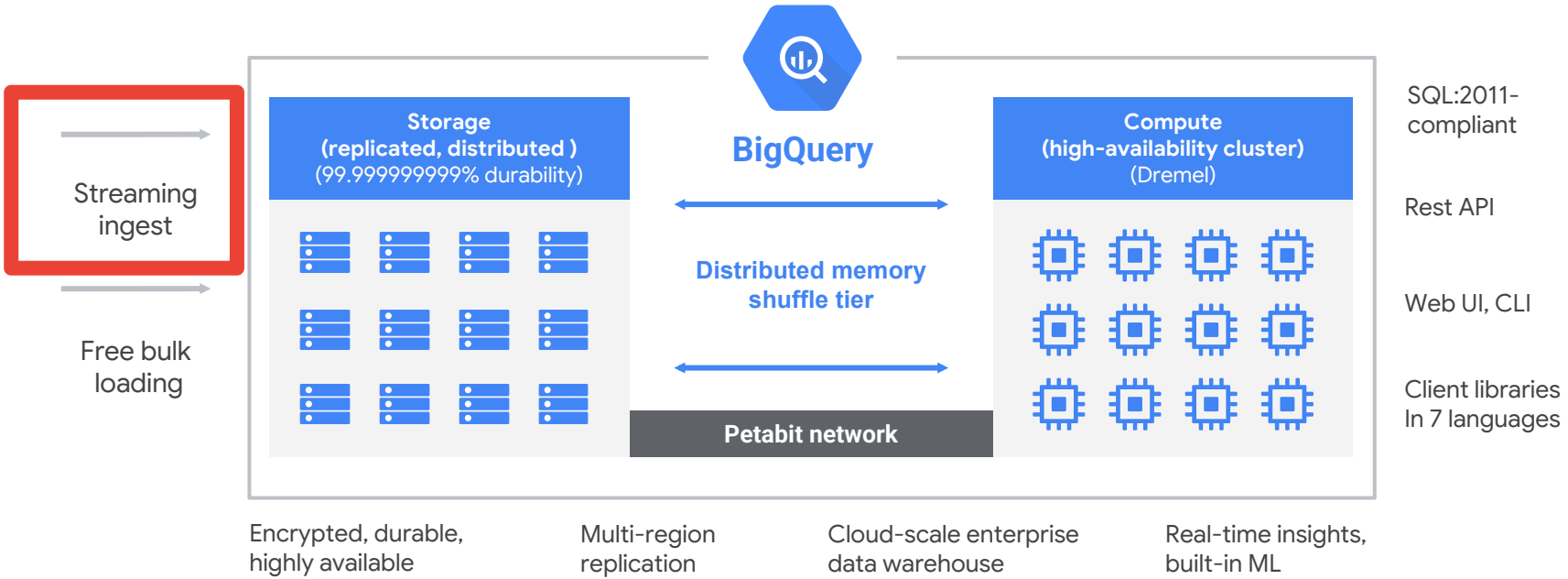
Restore a 2-minute-old copy.

For an alternate and complementary approach, you can also stream data directly into BigQuery



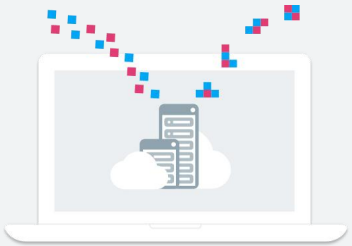
Streaming Ingestion

BigQuery architecture



All large datasets are generated over time

Streaming records into BigQuery through the API



Streaming record inserts

BigQuery managed storage

BigQuery query engine

Streaming data allows you to query data without waiting for a full batch load.

Legacy Streaming API

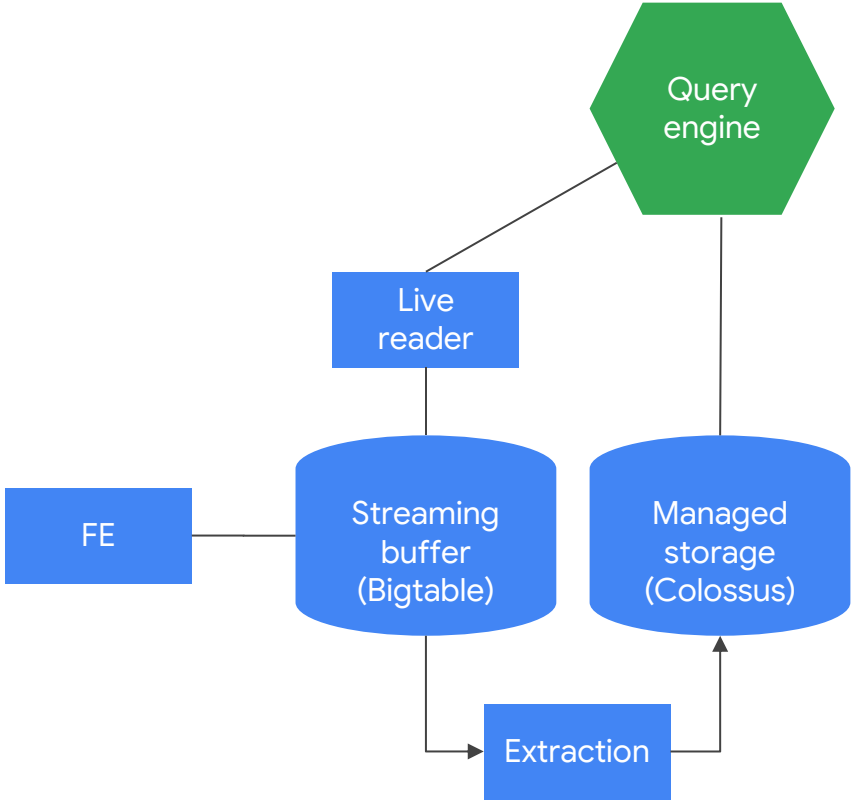
The legacy streaming system

Pros:

- Simple contract (here are some json keys and values)

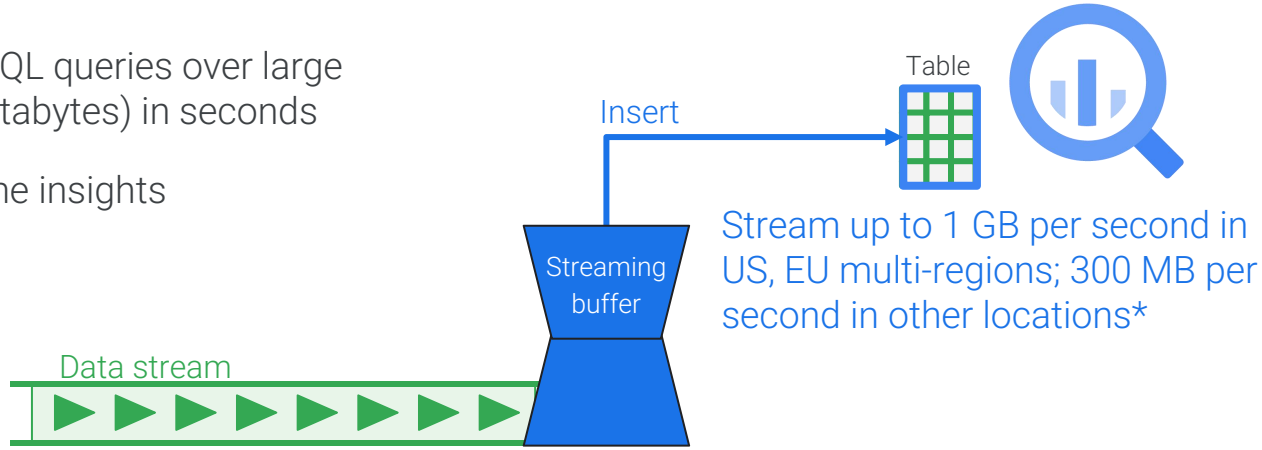
Problems:

- At-least-once append semantics
- Eventual consistency
  - streaming systems cache table metadata
- Best-effort deduplication
  - Users: no guarantee
  - BigQuery: costly



BigQuery allows you to stream records into a table; query results incorporate latest data

- Interactive SQL queries over large datasets (petabytes) in seconds
- Near-real-time insights



Note: Unlike for load jobs, there is a cost for streaming inserts (see **quota and limits**).

Insert streaming data into a BigQuery table

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/user/Downloads/[FILE_NAME].json"
```

Credentials

The service must have Cloud IAM permissions set in the web UI.

```
pip install google-cloud-bigquery
```

Install API

Python

```
from google.cloud import bigquery
client = bigquery.Client(project='PROJECT_ID')

dataset_ref = bigquery_client.dataset('my_datasset_id')
table_ref = dataset_ref.table('my_table_id')
table = bigquery_client.get_table(table_ref)

# read data from Cloud Pub/Sub and place into row format
# static example customer orders in units:
rows_to_insert = [
    (u'customer 1', 5),
    (u'customer 2', 17),
    ]
errors = bigquery_client.insert_rows(table, rows_to_insert)
```

Get table access from API

Create a client

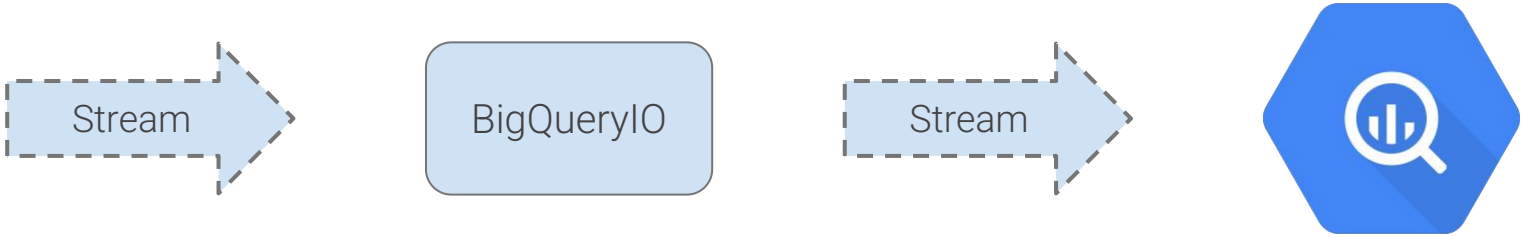
Access dataset and table

Insert rows into table

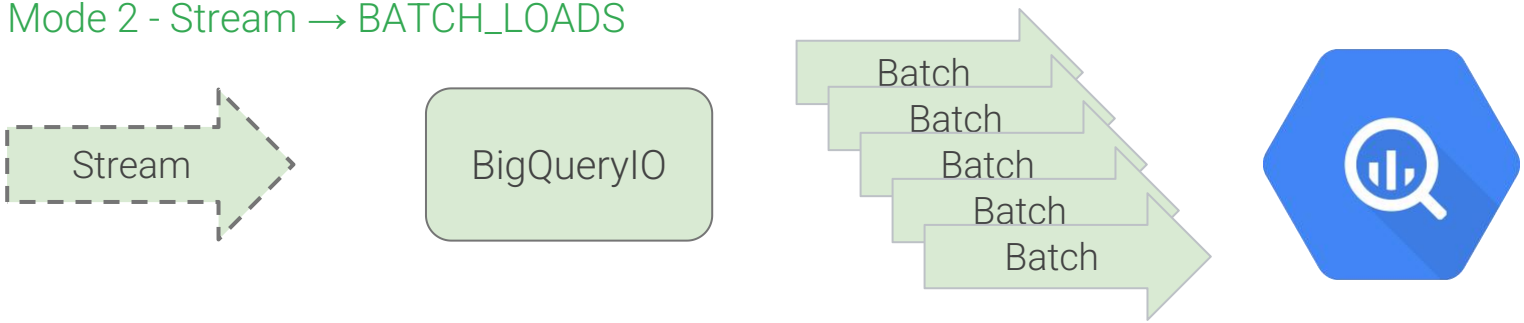
Perform insert

BigQueryIO connector for Dataflow: Sink performance considerations

Mode 1 - Stream → STREAMING_INSERT



Mode 2 - Stream → BATCH_LOADS



Writing into BigQuery with Dataflow

Streaming inserts

`withMethod(STREAMING_INSERTS)`

- Optimized for low latency
- 100K QPS quota per table. (BigQuery team can provide more)
- Pay for every insert
- Might sometimes duplicate data

Batch loads

`withMethod(FILE_LOADS)` or `withTriggeringFrequency(..)`

- Write output to files and periodically load into BigQuery
- Free!
- BigQuery has daily quotas (don't load too often; every 10 min is usually safe)
- No duplicates
- Requires more tuning to get right

BigQuery Storage Write API

Why exactly-once semantics are useful

- The main objective with streaming is to make data available even faster for analysis.
- Data needs to be clean (no duplicates).
- Typically happens post ingestion:
  - Extends the time from ingestion to making the data available for analysis.
- Guaranteeing exactly-once delivery means:
  - No custom deduplication logic
  - New data is available for analysis quickly

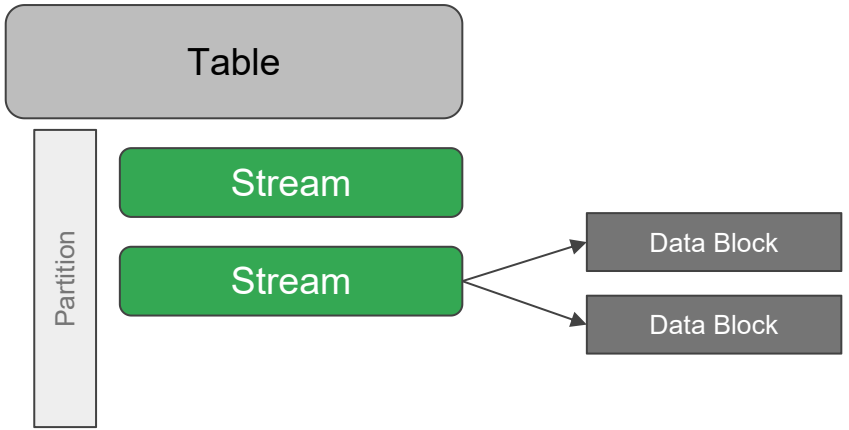
Streaming ingestion with BigQuery Storage Write API

- Unified batch and streaming API powered by the new streaming backend
- gRPC streaming efficient protocol
- Exactly-once ingestion
- Write and commit data as a single transaction
- Can detect schema changes during ingest
- Significantly lower cost than the older insertAll streaming API.
  - In addition, you can ingest up to 2 TB per month for free.

Stream-oriented storage

The core abstraction in the Storage Write API is a *stream*.

A stream writes data to a BigQuery table. More than one stream can write concurrently to the same table.



Two stream origins

Default stream

A special stream, which automatically commits all data as the backend processes it, is present on all tables.

Application-created stream

Users can choose to create one or more streams targeting a given table, with commit semantics governed by the specific configuration of each stream.

BigQueryWrite: Stream modes

COMMITTED

Data will commit (be visible) automatically as requests are processed by the backend.

Most like the old tabledata.insertAll semantics.

Default streams are of type COMMITTED.

BUFFERED

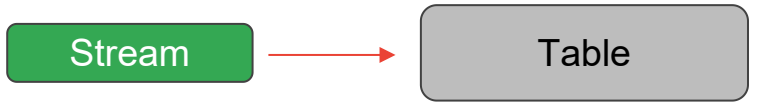
Data is buffered as it is appended, but doesn't appear until the row is beyond the flush point.

PENDING
(most transactional mode)

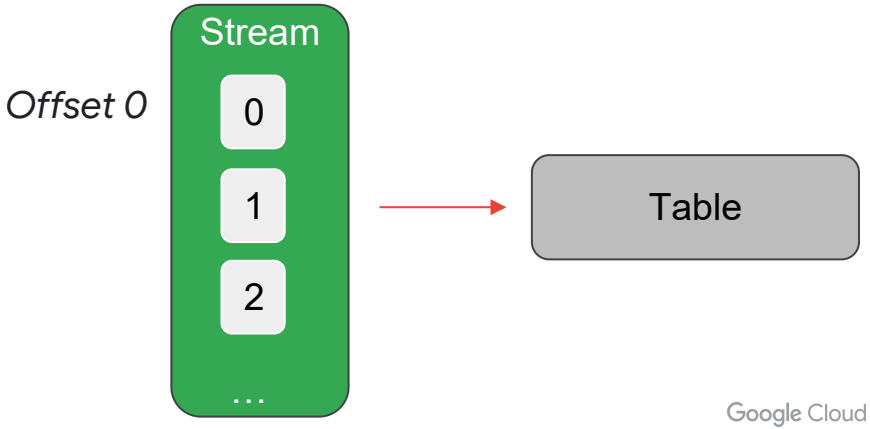
Data is buffered to a stream, but the stream must be finalized and committed before data is visible.

Stream mechanics

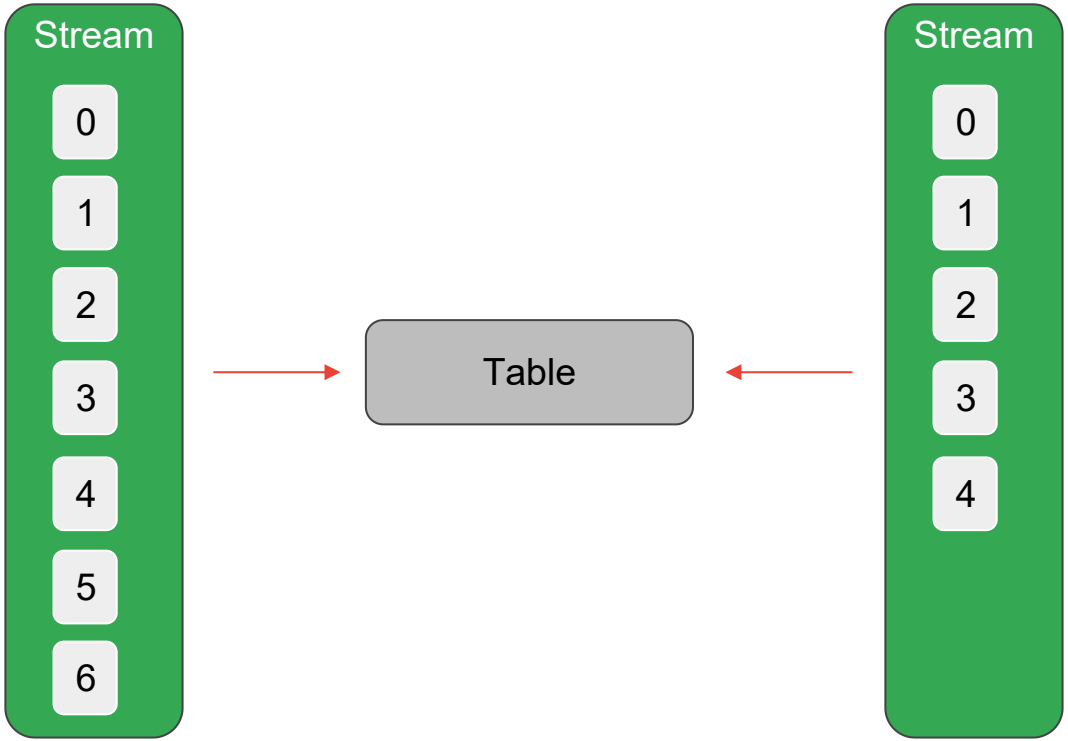
Streams provide persistent, named write buffers to BigQuery table storage.



Every append request must be made with an append offset.



Multiple streams can be created



Handling append errors

ALREADY_EXISTS error:

This happens when the client tries to append records before the stream offset.

OUT_OF_RANGE error:

This happens when an append is attempted beyond the current stream offset.

API flow for COMMITTED mode

1. Create a stream against a table with `CreateWriteStream`.
   - Not required for a **default** stream.

2. Append rows to a stream with `AppendRows`. (loop)
   - Creates a connection to a stream.

3. `FinalizeWriteStream` finalizes the stream so that no new data can be appended to it.
   - Optional for COMMITTED mode. Cannot use this method on a default stream.

API flow for BUFFERED mode

1. Create a stream against a table with `CreateWriteStream`.

2. Append rows to a stream with `AppendRows`, and then flush the stream using `FlushRows` in order for the rows to become available for reading. (loop)
   - `AppendRows` creates a connection to a stream.
   - `FlushRows` operation flushes up to any previously flushed offset.

3. `FinalizeWriteStream` finalizes the stream so that no new data can be appended to it.
   - Optional for BUFFERED mode.

API flow for PENDING mode

1. Create a stream against a table with `CreateWriteStream`.

2. Append rows to a stream with `AppendRows`.
   - Creates a connection to a stream.

3. `FinalizeWriteStream` finalizes the stream so that no new data can be appended to it.
   - Required for PENDING mode.

4. `BatchCommitWriteStreams` is used in PENDING streams to make data available for reading.
   - Committed atomically for a group of PENDING streams.
   - Streams must be finalized first and cannot be committed multiple times.

Client libraries

Java

Comes with two writer objects:

- `StreamWriter`
- `JSONStreamWriter`

Python

- Lower-level client that wraps the gRPC API.
- You must send the data as protocol buffers.

Writing into default stream (at-least-once semantics)

```
// Default Stream name
    String defaultStreamName = "projects/p/datasets/d/tables/t/streams/_default";
    try (StreamWriter streamWriter = StreamWriter.newBuilder(defaultStreamName).build())
{
      // Use the same Stream to write 10K Protobuf rows
      for (int i = 0; i < 100000; i++) {
        // Create Protobuf rows with example Protobuf type
        FooType fooType = FooType.newBuilder().setFoo(String.format("message %03d",
i)).build();
        ProtoRows protoRows =
            ProtoRows.newBuilder().addSerializedRows(fooType.toByteString()).build();
        // Append the row
        streamWriter.append(protoRows);
      }
      FinalizeWriteStreamResponse finalizeResponse =
          bigqueryWriteClient.finalizeWriteStream(defaultStreamName);
      System.out.println("Rows written: " + finalizeResponse.getRowCount());
    } catch (IOException e) {
      System.out.println("Failed to append records. \n" + e.getMessage());
       }
```

Writing into committed stream for exactly-once semantics

```
  WriteStream stream = WriteStream.newBuilder().setType(WriteStream.Type.COMMITTED).build();

  CreateWriteStreamRequest createWriteStreamRequest =
     CreateWriteStreamRequest.newBuilder()
              .setParent(parentTable.toString())
              .setWriteStream(stream)
              .build();
  WriteStream writeStream = client.createWriteStream(createWriteStreamRequest);
```

```
streamWriter =
         JsonStreamWriter.newBuilder(writeStream.getName(), writeStream.getTableSchema()).build();

ApiFuture<AppendRowsResponse> future = streamWriter.append(data, offset);
```

```
FinalizeWriteStreamResponse finalizeResponse =
            client.finalizeWriteStream(streamWriter.getStreamName());
```

Using the BigQueryIO connector

```
WriteResult writeResult = rows.apply("Save Rows to BigQuery",
BigQueryIO.writeTableRows()
        .to(options.getFullyQualifiedTableName())
        .withWriteDisposition(WriteDisposition.WRITE_APPEND)
        .withCreateDisposition(CreateDisposition.CREATE_NEVER)
        .withMethod(Method.STORAGE_WRITE_API);
   );
```

Remember:

- Set the Beam SDK version to 2.36.0 (or later).
- Set the **number of streams** and **triggering frequency** parameters.

Limits and quotas

- You can commit up to 10,000 streams per table in each *BatchCommitWriteStream* call.
- The maximum *AppendRows* request size is 10 MB.
- You can call *CreateWriteStream* up to 30,000 times per four hours per project.
- You can commit up to 10 TB in the **US** and **EU** multi-regions and up to 100 GB in other regions.

Limits and quotas (continued)

- Your project can operate on 10,000 concurrent connections in the US and EU multi-regions and on 100 concurrent connections in other regions.
- You can stream up to 3 GBps in the US and EU multi-regions and up to 300 MBps in other regions per project.

Best practices

Manage stream offsets to achieve exactly-once semantics.

Limit the rate of stream creation.

Handle schema updates.

Do not block on *AppendRows* calls.

Limit the number of concurrent connections.

Query Materialization

All query results are saved to a table

- All query results are saved to either a temporary or permanent table.
- If you specify a destination table, that table becomes permanent; if not, it's a new temporary table.
- Temporary tables are the basis of query cached results.
- Temporary tables last 24 hours only.

How to create a new permanent table

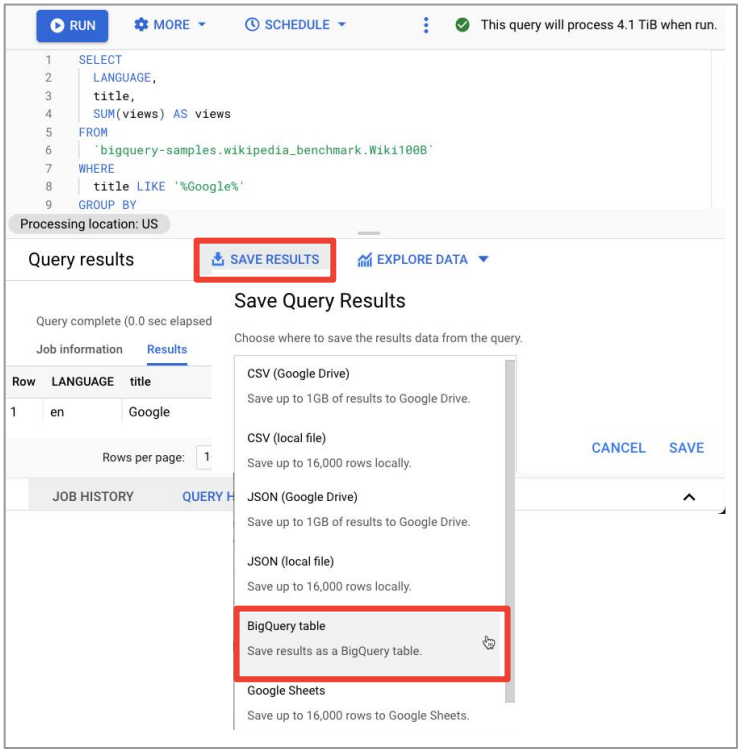
1. Write SQL query.
2. Click **More > Query Settings.**
3. Specify the **destination table** (can be existing).
4. Choose **Write Preference** (if table already exists).
5. Run query.

If the destination table exists:

- Write if empty.
- Append records.
- Overwrite table.

Forgot to specify a table? Store query results after running

- Use **SAVE RESULTS > BigQuery table.**
- **All query results are stored in tables** (regardless of whether you save as a table).
- If you don't save as a permanent table, a temporary one is automatically created and saved for 24 hours.
- Re-running the same query will probably retrieve the cached temporary table.



Create new tables from data with SQL DDL

```
SELECT
    *
FROM
    movielens.movies_raw
WHERE
    movieId < 5;
```

	movieId	title	genres
0	3	Grumpier Old Men (1995)	Comedy|Romance
1	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
2	2	Jumanji (1995)	Adventure|Children|Fantasy
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

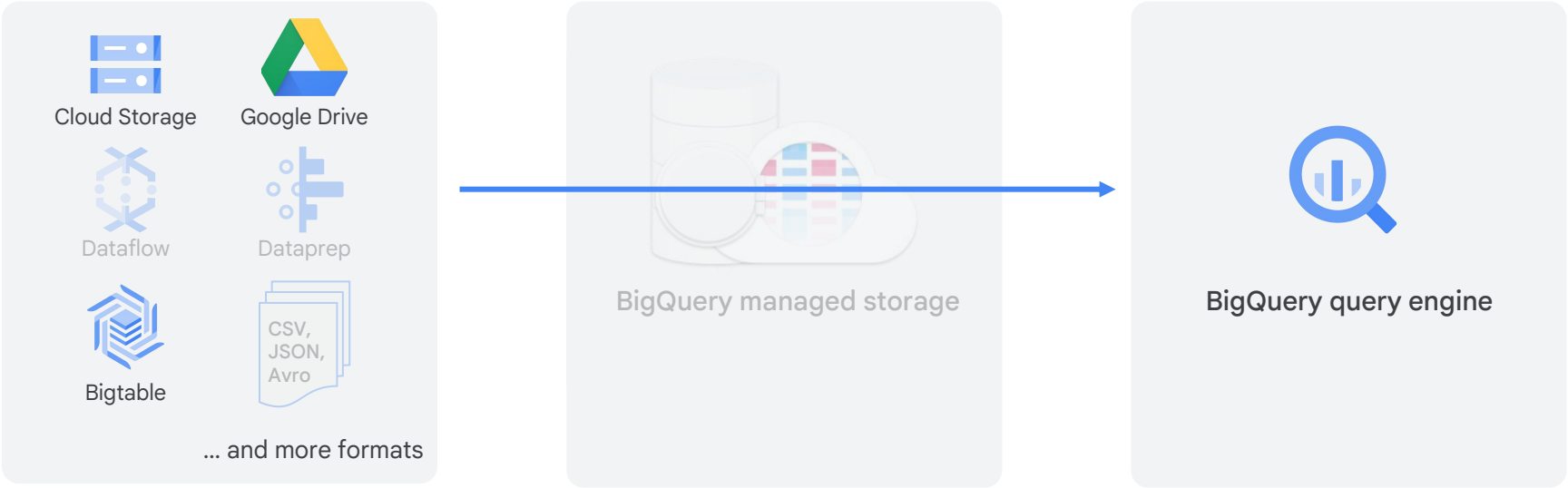
```
CREATE OR REPLACE TABLE
movielens.movies AS
SELECT
    * REPLACE(SPLIT(genres, "|
AS genres)
FROM
    Movielens.movies_raw;
-- Execute multiple statements.
SELECT * FROM movielens.movies;
```

	movieId	title	genres
0	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]
1	3	Grumpier Old Men (1995)	[Comedy, Romance]
2	2	Jumanji (1995)	[Adventure, Children, Fantasy]
3	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]

Question: What's the difference between CREATE OR REPLACE TABLE and CREATE TABLE IF NOT EXISTS? When would you use each?

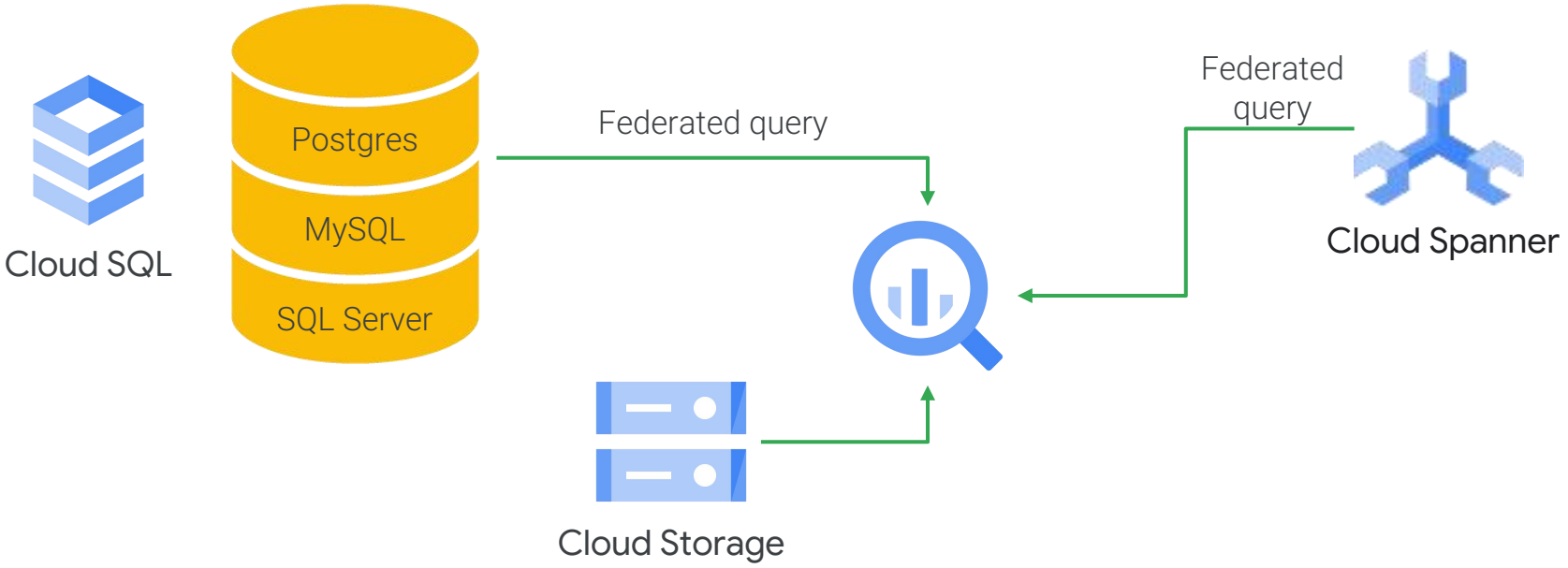
Query External Data Sources

BigQuery can query external data sources in Cloud Storage and others directly



You can query external data sources directly from BigQuery, which bypasses managed storage.

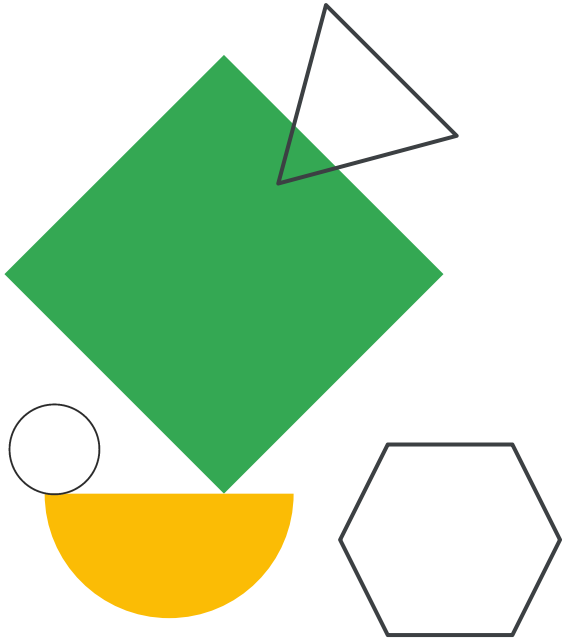
You can simplify data warehouse ETL pipelines with external connections



Demo

Querying External Data Sources from BigQuery

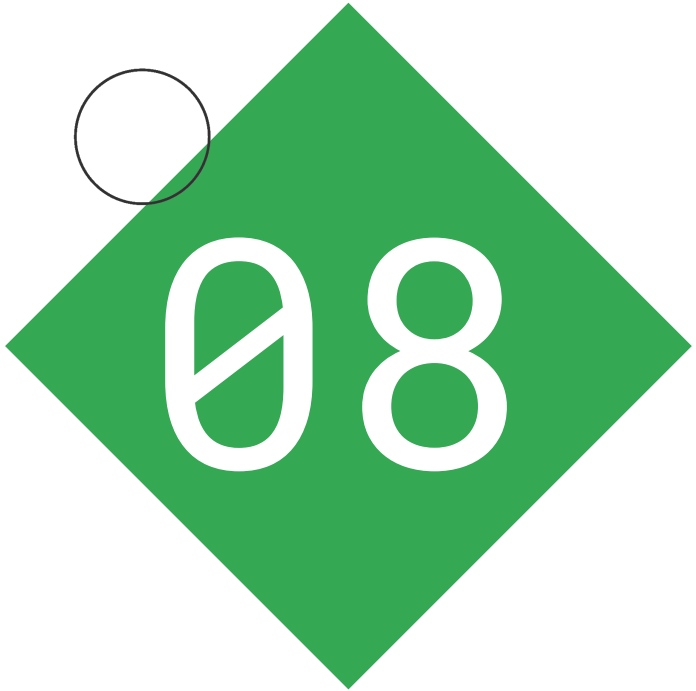
Live updates from Google Spreadsheets



Pitfalls: Querying from external data sources directly

Limitations

- Strong performance disadvantages
- Data consistency not guaranteed
- Can’t use table wildcards



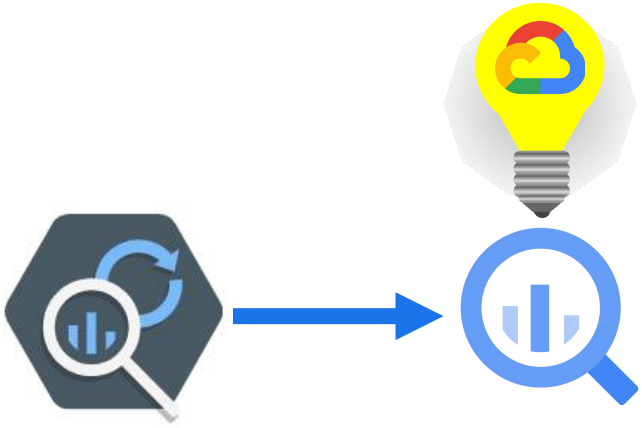
Data Transfer Service

If the data is usable in its original form, just load it

BigQuery Data Transfer Service

BigQuery

BigQuery Data Transfer Service helps you build and manage your data warehouse



EL

- Managed service
- Automatic transfers
- Scheduled
- Data staging
- Data processing
- Data backfills

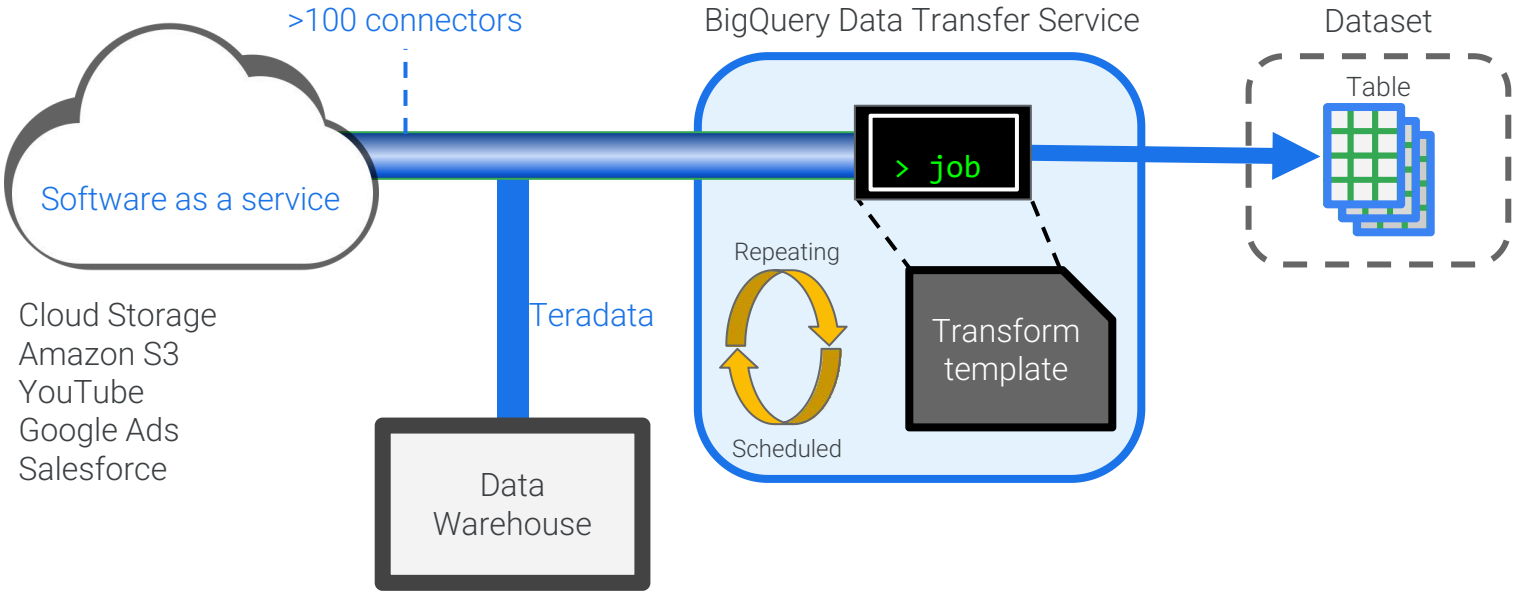
BigQuery Data Transfer Service

The BigQuery Data Transfer Service simplifies and automates data movement to BigQuery on a scheduled, managed basis.

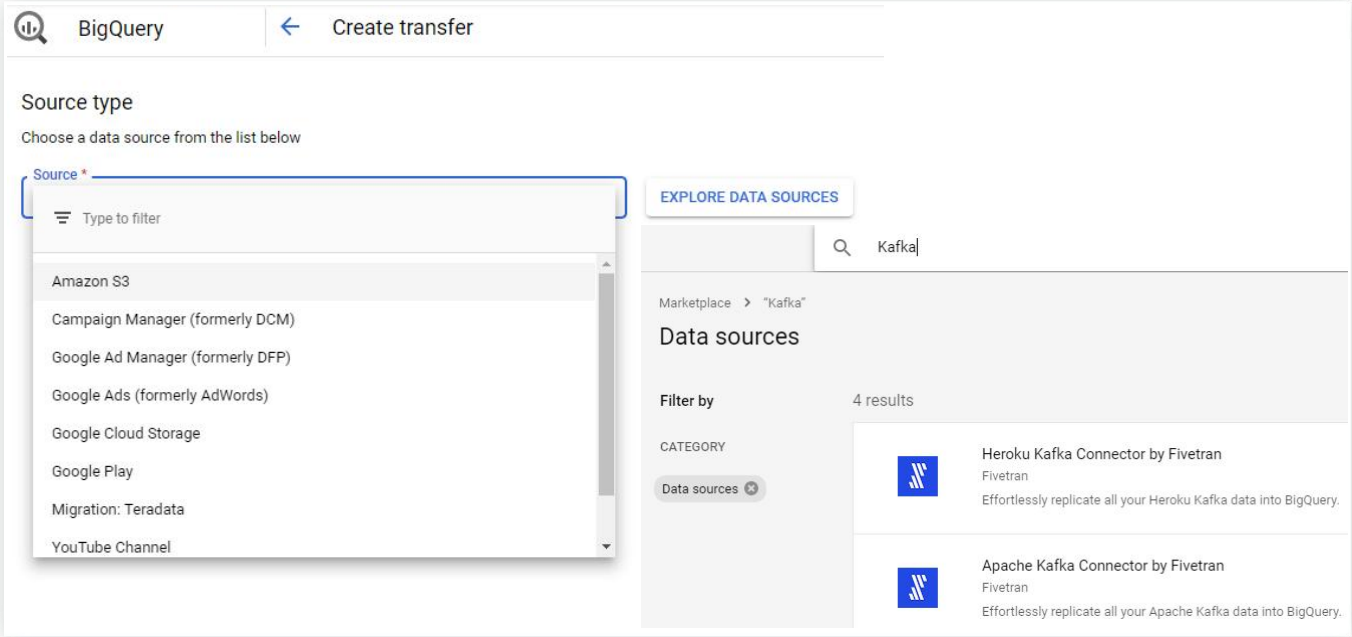
With its flexible infrastructure, the BigQuery Data Transfer Service is also used to power:

- BigQuery scheduled queries
- Cloud Storage/Amazon S3 mirroring
- Recurring copy jobs
- Data warehouse migration
- Third-party data source transfers

BigQuery Data Transfer Service provides SaaS connectors

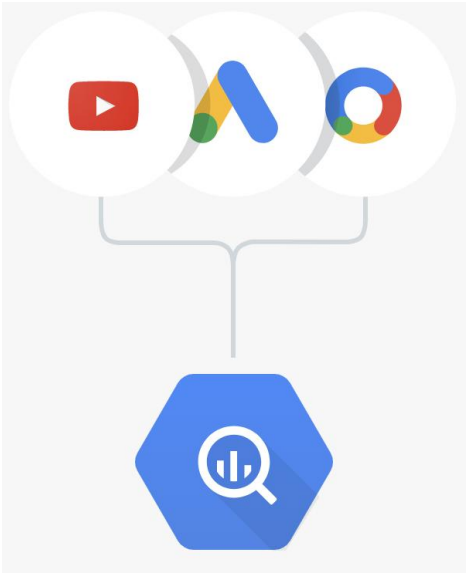


BigQuery Data Transfer Service supports 100+ SaaS applications



Data Transfer Service: Options

- Automatically transfers data from multiple sources (YouTube, Google Ads, Google Play, Teradata, Amazon S3 and Redshift, Cloud Storage, BigQuery, etc.).
- Transfer jobs can be **scheduled** as Daily, Weekly, Monthly, Custom, or On Demand.
- Provides **wildcard support** for Cloud Storage URIs.
- Provides **run time parameterization** of source and destination paths.
- Enables capturing transfer job run **notifications** via Pub/Sub topic, and failure notifications via email.
- Tracks multiple runs of the transfer jobs in the UI (Transfer Details and Job History) and provides detailed logs via Google Cloud's operations suite.



Data Transfer Service: Terminologies

Data source

Data source refers to where the Data Transfer Service reads data from.

In most cases, Data Transfer Service downloads data from the source to a staging bucket on Cloud Storage before loading to BigQuery.

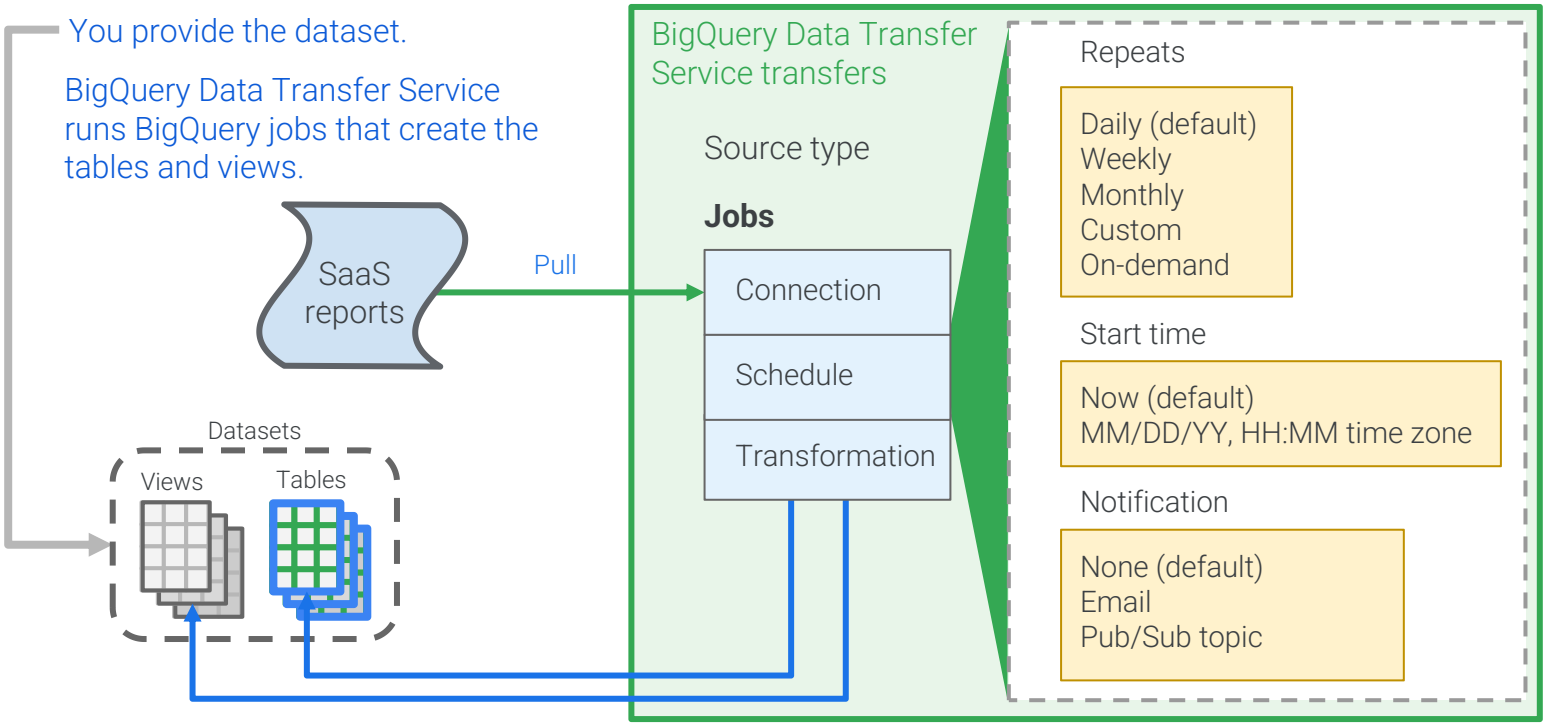
Transfer configuration

A transfer configuration is created by the users with the credentials, schedule, BigQuery destination, and metadata required to schedule transfer runs.

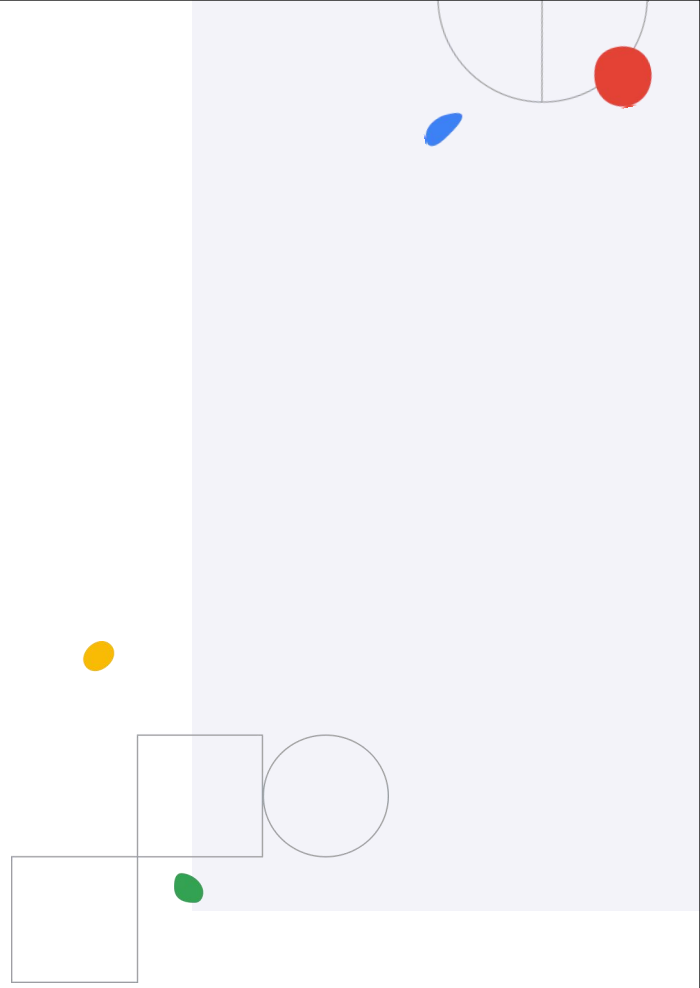
Transfer run

A transfer run is the scheduled job created by the Data Transfer Service from the corresponding transfer configuration. The transfer run is sent to the corresponding data source for processing.

How BigQuery Data Transfer Service transfers work



Questions?



Google Cloud

Lab (45 min)

Ingesting New Datasets into BigQuery

Google Cloud